Antenna Toolbox™

Reference

MATLAB®

R2022b

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Antenna Toolbox™ Reference

Trademarks

Patents

Revision History

March 2015	Online only	New for Version 1.0 (R2015a)
September 2015	Online only	Revised for Version 1.1 (R2015b)
March 2016	Online only	Revised for Version 2.0 (R2016a)
September 2016	Online only	Revised for Version 2.1 (R2016b)
March 2017	Online only	Revised for Version 2.2 (R2017a)
September 2017	Online only	Revised for Version 3.0 (R2017b)
March 2018	Online only	Revised for Version 3.1 (R2018a)
September 2018	Online only	Revised for Version 3.2 (R2018b)
March 2019	Online only	Revised for Version 4.0 (R2019a)
September 2019	Online only	Revised for Version 4.1 (R2019b)
March 2020	Online only	Revised for Version 4.2 (R2020a)
September 2020	Online only	Revised for Version 4.3 (R2020b)
March 2021	Online only	Revised for Version 5.0 (R2021a)
September 2021	Online only	Revised for Version 5.1 (R2021b)
March 2022	Online only	Revised for Version 5.2 (R2022a)
September 2022	Online only	Revised for Version 5.3 (R2022b)

Contents

1

Objects

biquad

Create biquad or double-biquad antenna

Description

The `biquad` antenna is center fed and symmetric about its origin. The default length is chosen for an operating frequency of 2.8 GHz.

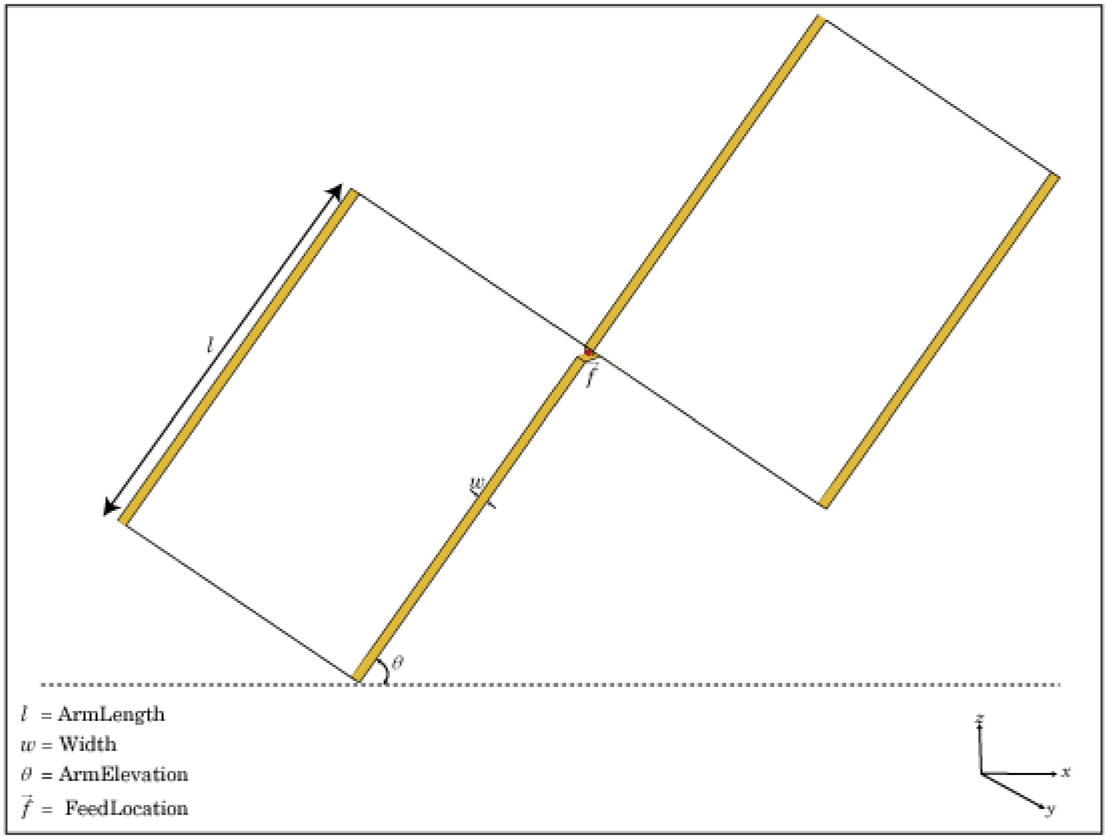
The width of the strip is related to the diameter an equivalent cylinder:

$$w = 2d = 4r$$

, where:

- d is the diameter of equivalent cylindrical dipole.
- r is the radius of equivalent cylindrical dipole.

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default strip dipole is center-fed. The feed point coincides with the origin. The origin is located on the *yz*- plane.



Creation

Syntax

```
bq = biquad
bq = biquad(Name,Value)
```

Description

`bq = biquad` creates a biquad antenna.

`bq = biquad(Name,Value)` creates a biquad antenna with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`NumLoops` — Number of loops
2 (default) | scalar integer

Number of loops for the biquad, specified as a scalar integer. Setting this property to 4 supports a double biquad antenna.

Example: `'NumLoops'`,4

Data Types: `double`

`ArmLength` — Length of two arms
0.0305 (default) | scalar

Length of two arms, specified as a scalar in meters. The default length is chosen for an operating frequency of 2.8 GHz.

Example: `'ArmLength'`,`0.0206`

Data Types: `double`

`Width` — Biquad arm width
1.0000e-03 (default) | scalar

Biquad arm width, specified as a scalar in meters.

Example: `'Width'`,`0.006`

Data Types: `double`

`ArmElevation` — Angle formed by biquad arms to *xy*- plane
45 (default) | scalar

Angle formed by biquad arms to the *xy*- plane, specified a scalar in meters.

Example: `'ArmElevation'`, `50`

Data Types: `double`

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified a lumped element object. For more information, see lumpedElement.

Example: 'Load', lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: bq.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: TiltAxis=[0 1 0]

Example: TiltAxis=[0 0 0;0 1 0]

Example: TiltAxis = 'Z'

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

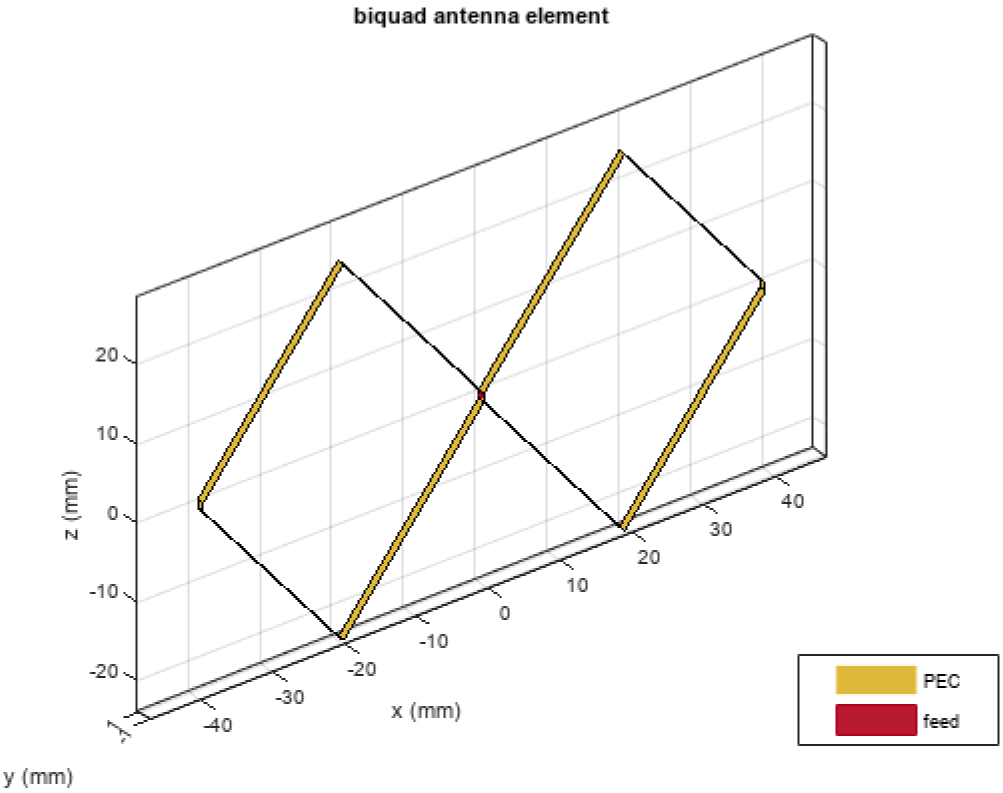
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Biquad Antenna

Create a biquad antenna with arm angles at 50 degrees and view it.

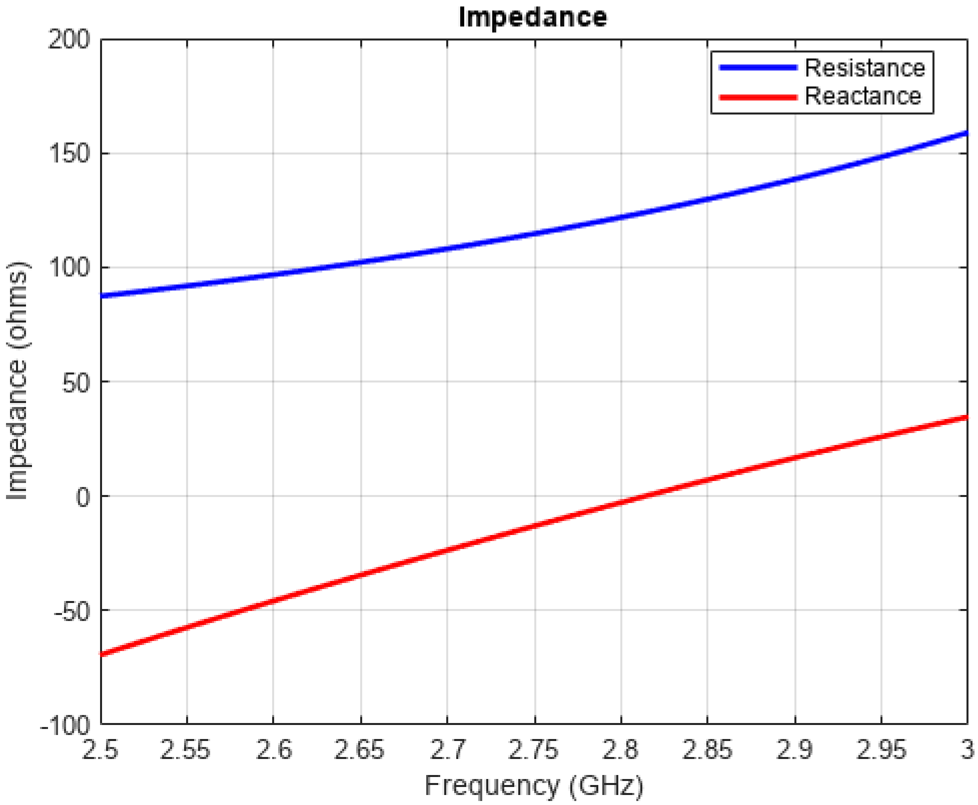
```
bq = biquad('ArmElevation',50);
show(bq)
```



Impedance of Biquad Antenna

Calculate the impedance of a biquad antenna over a frequency span 2.5GHz-3GHz.

```
bq = biquad('ArmElevation',50);
impedance(bq,linspace(2.5e9,3e9,51));
```



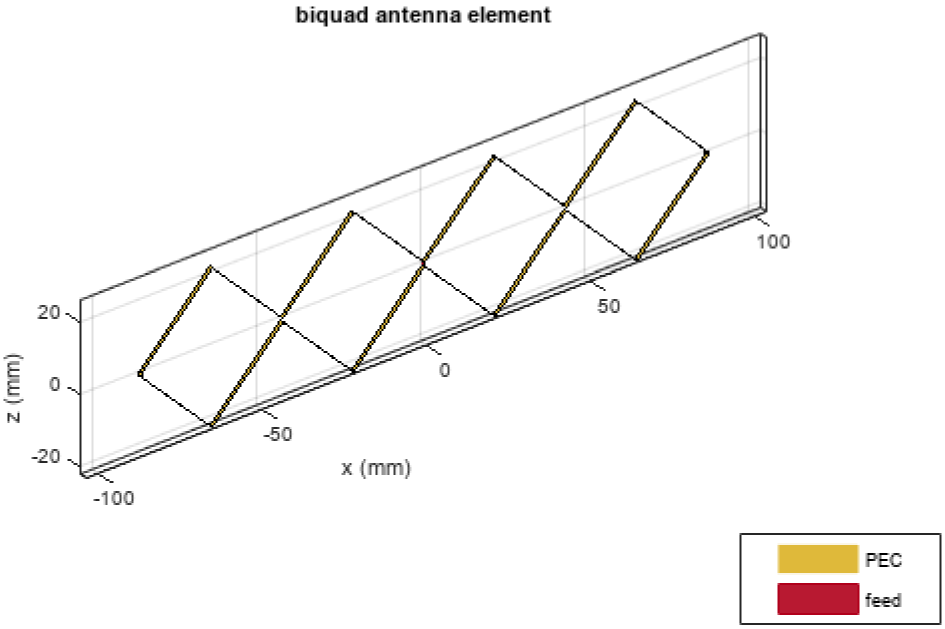
Double Biquad Antenna

Create and view a double biquad antenna using default property values.

```
ant = biquad('NumLoops',4)
```

```
ant = 
  biquad with properties:

        NumLoops: 4
       ArmLength: 0.0305
    ArmElevation: 45
           Width: 1.0000e-03
       Conductor: [1x1 metal]
            Tilt: 0
        TiltAxis: [1 0 0]
            Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2015b

See Also

`dipole` | `dipoleFolded` | `loopCircular`

Topics

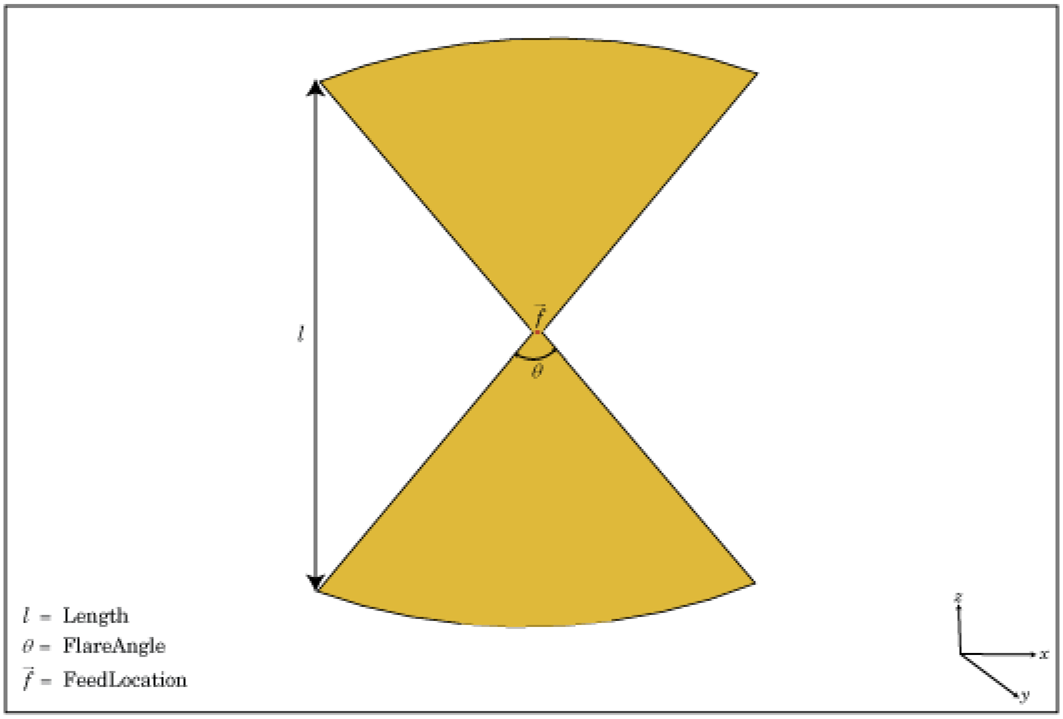
"Rotate Antennas and Arrays"

bowtieRounded

Create rounded bowtie dipole antenna

Description

The `bowtieRounded` object is a planar bowtie antenna, with rounded edges, on the *yz*- plane. The default rounded bowtie is center fed. The feed point coincides with the origin. The origin is located on the *yz*- plane.



Creation

Syntax

```
br = bowtieRounded
br = bowtieRounded(Name,Value)
```

Description

`br = bowtieRounded` creates a half-wavelength planar bowtie antenna with rounded edges.

`br = bowtieRounded(Name,Value)` creates a planar bowtie antenna with rounded edges, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Length` — Rounded bowtie length

0.2000 (default) | scalar

Rounded bowtie length, specified a scalar in meters. By default, the length is chosen for the operating frequency of 490 MHz.

Example: `'Length',3`

Data Types: `double`

`FlareAngle` — Rounded bowtie flare angle

90 (default) | scalar

Rounded bowtie flare angle, specified a scalar in degrees.

Note Flare angle should be less than 175 degrees and greater than 5 degrees.

Example: `'FlareAngle',80`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load'`, lumpedelement. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `br.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

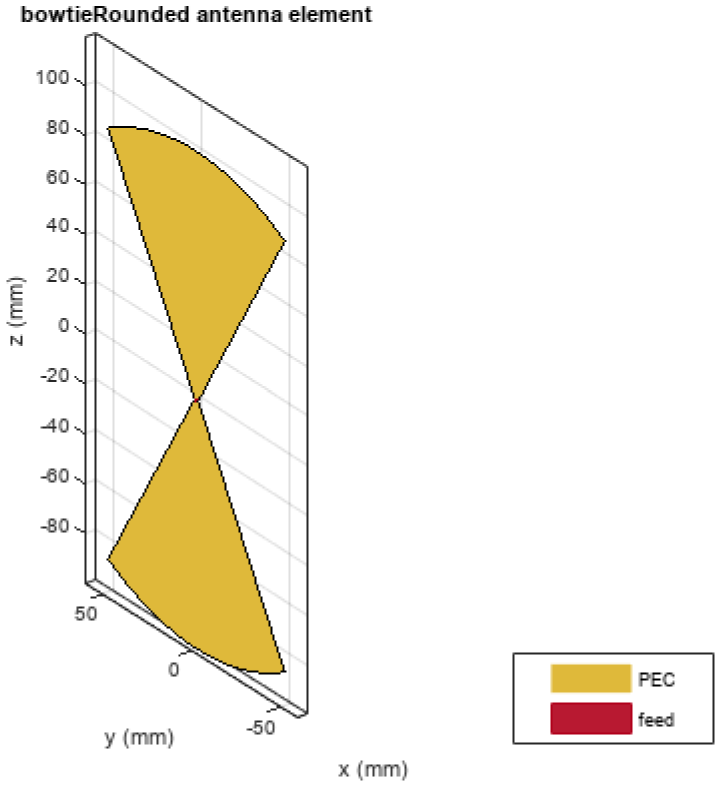
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Center-Fed Rounded Bowtie Antenna

Create and view a center-fed rounded bowtie that has a flare angle of 60 degrees.

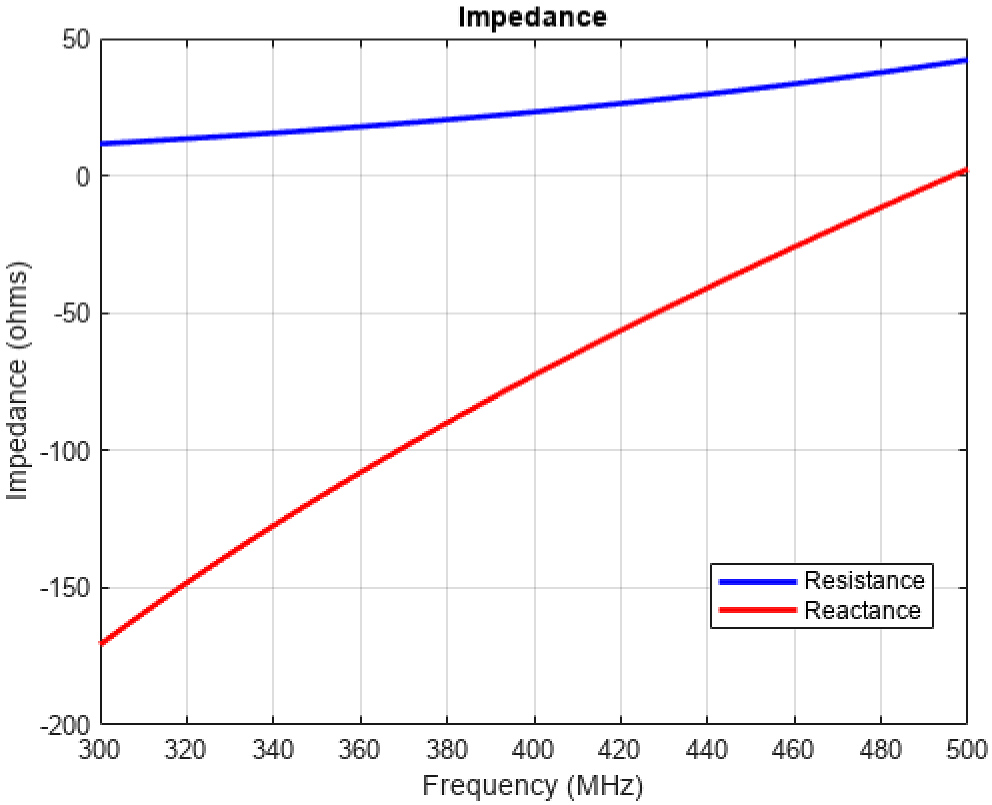
```
b = bowtieRounded('FlareAngle',60);
show(b);
```



Impedance of Rounded Bowtie Antenna

Calculate and plot the impedance of a rounded bowtie over a frequency range of 300 MHz-500 MHz.

```
b = bowtieRounded('FlareAngle',60);
impedance(b,linspace(300e6,500e6,51))
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A.*Antenna Theory: Analysis and Design*.3rd Ed. New York: Wiley, 2005.

[2] Brown, G.H., and O.M. Woodward Jr. “Experimentally Determined Radiation Characteristics of Conical and Triangular Antennas”. *RCA Review*. Vol.13, No.4, Dec.1952, pp. 425–452

See Also

`bowtieTriangular` | `dipole` | `dipoleFolded`

Topics

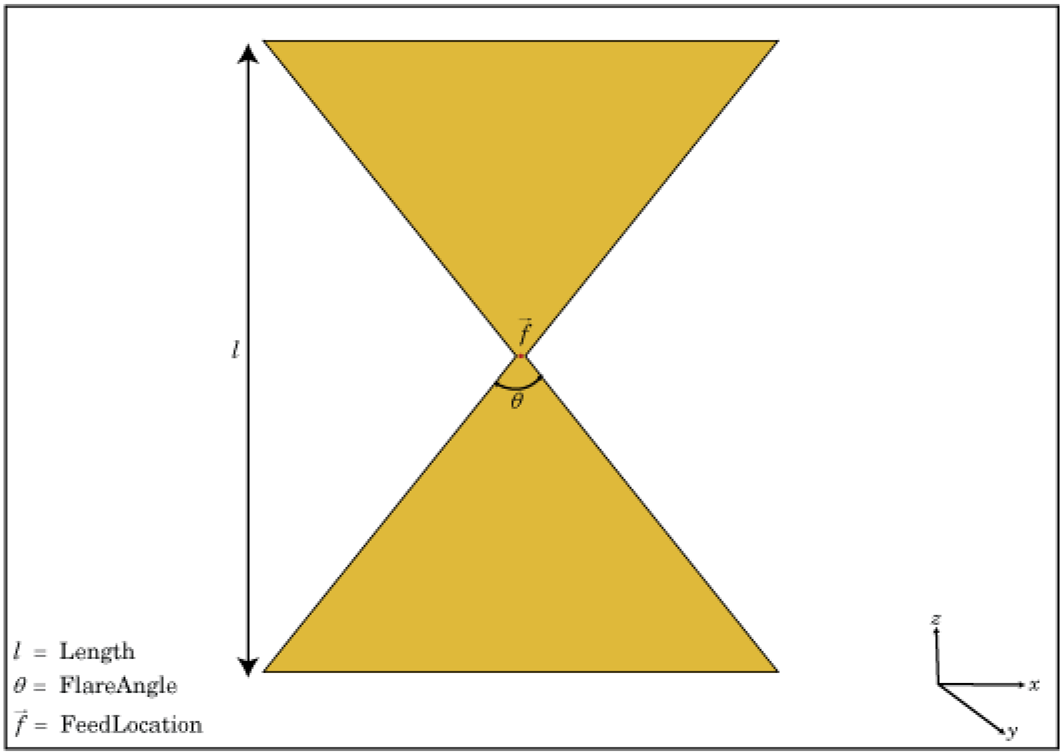
“Rotate Antennas and Arrays”

bowtieTriangular

Create planar bowtie dipole antenna

Description

The `bowtieTriangular` object is a planar bowtie antenna on the *yz*- plane. The default planar bowtie dipole is center-fed. The feed point coincides with the origin. The origin is located on the *yz*-plane.



Creation

Syntax

```
bt = bowtieTriangular
bt = bowtieTriangular(Name,Value)
```

Description

`bt = bowtieTriangular` creates a half-wavelength planar bowtie antenna.

`bt = bowtieTriangular(Name,Value)` creates a planar bowtie antenna with additional properties specified by one or more name-value pair arguments. `Name` is the property name and

`Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Length` — Planar bowtie length

0.2000 (default) | scalar

Planar bowtie length, specified as a scalar in meters. By default, the length is chosen for the operating frequency of 410 MHz.

Example: `'Length',3`

Data Types: `double`

`FlareAngle` — Planar bowtie flare angle

90 (default) | scalar

Planar bowtie flare angle near the feed, specified as a scalar in meters.

Note Flare angle should be less than `175` degrees and greater than `5` degrees.

Example: `'FlareAngle',80`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load'`, lumpedelement. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `bt.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array

rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

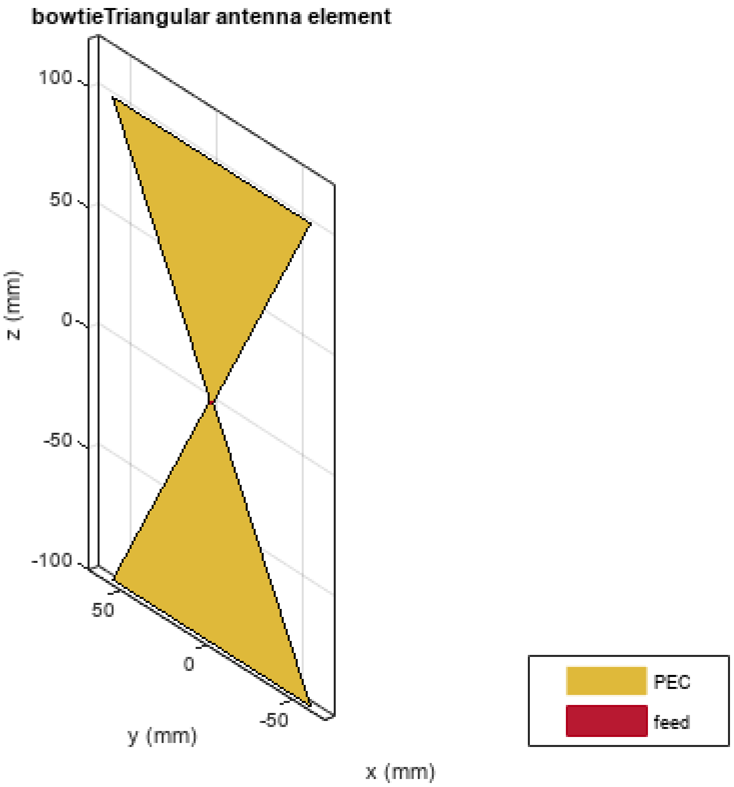
Create and View Center-Fed Planar Bowtie Antenna

Create and view a center-fed planar bowtie antenna that has a 60 degrees flare angle.

```
b = bowtieTriangular('FlareAngle',60)
```

```
b = 
  bowtieTriangular with properties:

        Length: 0.2000
    FlareAngle: 60
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

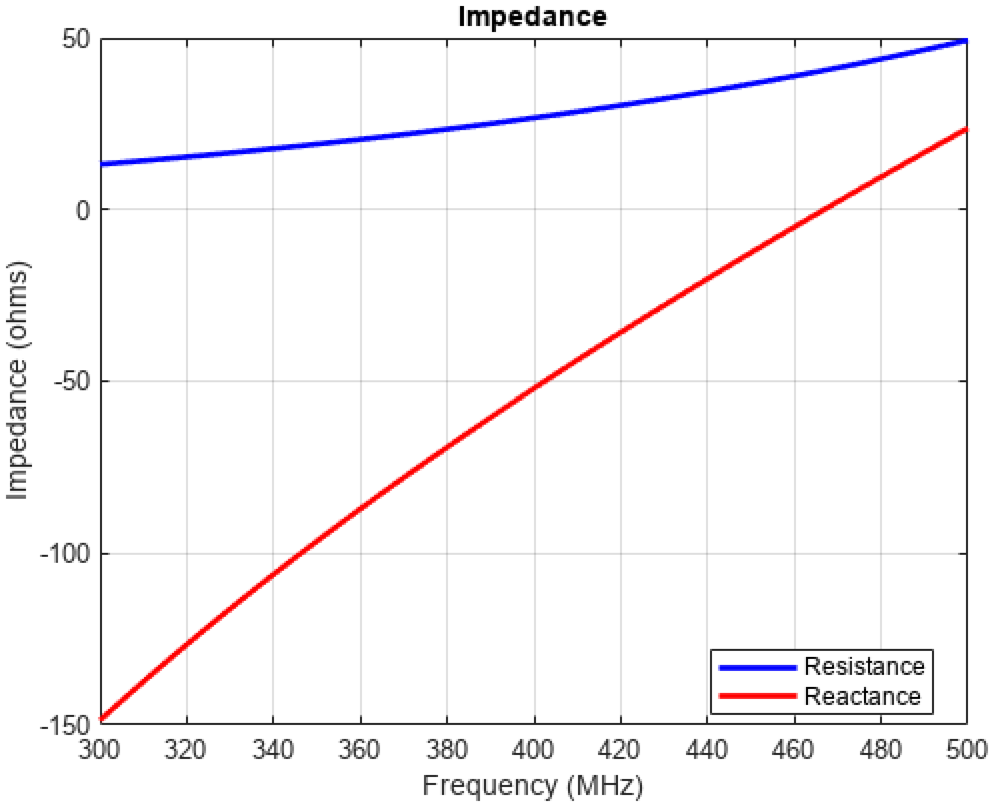
```
show(b)
```



Impedance of Planar Bowtie Antenna

Calculate and plot the impedance of a planar bowtie antenna over a frequency range of 300 MHz-500 MHz.

```
b = bowtieTriangular('FlareAngle',60);
impedance(b,linspace(300e6,500e6,51))
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A.*Antenna Theory: Analysis and Design*.3rd Ed. New York: Wiley, 2005.

[2] Brown, G.H., and O.M. Woodward Jr. “Experimentally Determined Radiation Characteristics of Conical and Triangular Antennas”. *RCA Review*. Vol.13, No.4, Dec.1952, pp. 425–452

See Also

`bowtieRounded` | `dipole` | `dipoleVee`

Topics

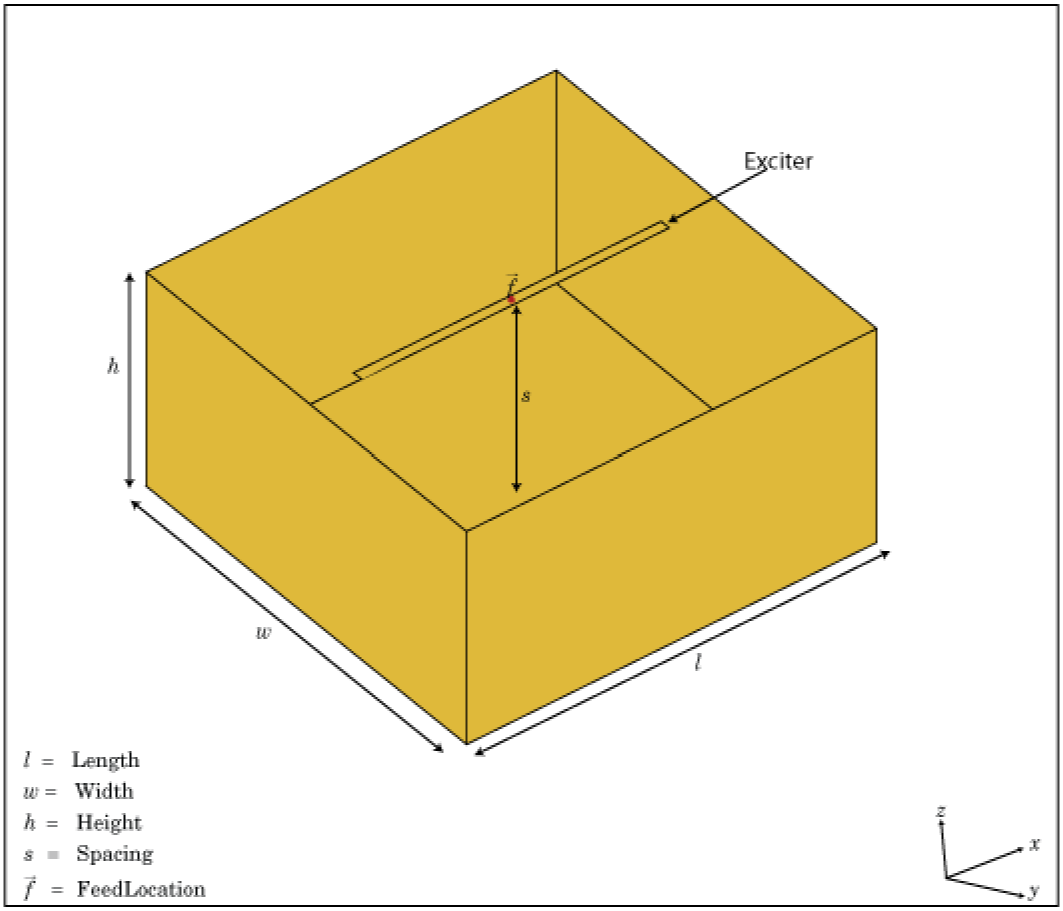
"Rotate Antennas and Arrays"

cavity

Create cavity-backed antenna

Description

The `cavity` object is a cavity-backed antenna located on the *xyz*- plane. The default cavity antenna has a dipole as an exciter. The feed point is on the exciter.



Creation

Syntax

```
c = cavity
c = cavity(Name,Value)
```

Description

`c = cavity` creates a cavity backed antenna located on the X-Y-Z plane. By default, the dimensions are chosen for an operating frequency of 1 GHz.

`c = cavity(Name,Value)` creates a cavity-backed antenna, with additional properties specified by one or more name–value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Exciter` — Antenna or array type used as exciter
`dipole` (default) | `antenna` object | `array` object

Antenna type used as an exciter, specified as any single-element antenna object. Except reflector and cavity antenna elements, you can use any of the antenna elements or array elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',horn`

Example: `ant.Exciter = horn`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Substrate` — Type of dielectric material
'Air' (default) | object

Type of dielectric material used as a substrate, specified as an object. For more information see, `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the ground plane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); cavity.Substrate = d`

`Length` — Length of rectangular cavity along *x*-axis
0.2000 (default) | scalar

Length of the rectangular cavity along the *x*-axis, specified as a scalar in meters.

Example: `'Length',30e-2`

Data Types: `double`

`Width` — Width of rectangular cavity along *y*-axis
0.2000 (default) | scalar

Width of the rectangular cavity along the *y*-axis, specified as a scalar in meters.

Example: `'Width',25e-2`

Data Types: `double`

`Height` — Height of rectangular cavity along *z*-axis
0.0750 (default) | scalar

Height of the rectangular cavity along the *z*-axis, specified as a scalar in meters.

Example: `'Height',7.5e-2`

Data Types: `double`

`Spacing` — Distance between exciter and base of cavity

0.0750 (default) | scalar

Distance between the exciter and the base of the cavity, specified as a scalar in meters.

Example: `'Spacing',7.5e-2`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load'`, lumpedelement. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `c.Load = lumpedElement('Impedance',75)`

`EnableProbeFeed` — Create probe feed from backing structure to exciter

`0` (default) | `1`

Create probe feed from backing structure to exciter, specified as a `0` or `1`. By default, probe feed is not enabled.

Example: `'EnableProbeFeed',1`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

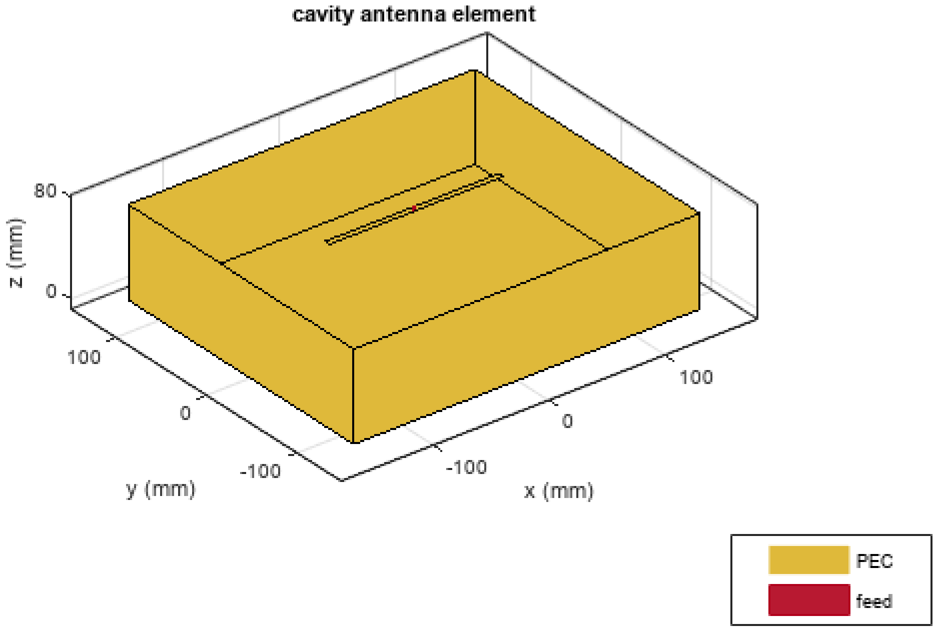
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Cavity-Backed Antenna

Create and view a cavity-backed dipole antenna with 30 cm length, 25 cm width, 7.5 cm height and spaced 7.5 cm from the bowtie for operation at 1 GHz.

```
c = cavity('Length',30e-2, 'Width',25e-2,'Height',7.5e-2,'Spacing',7.5e-2);
show(c)
```



Radiation Pattern of Cavity-Backed Antenna

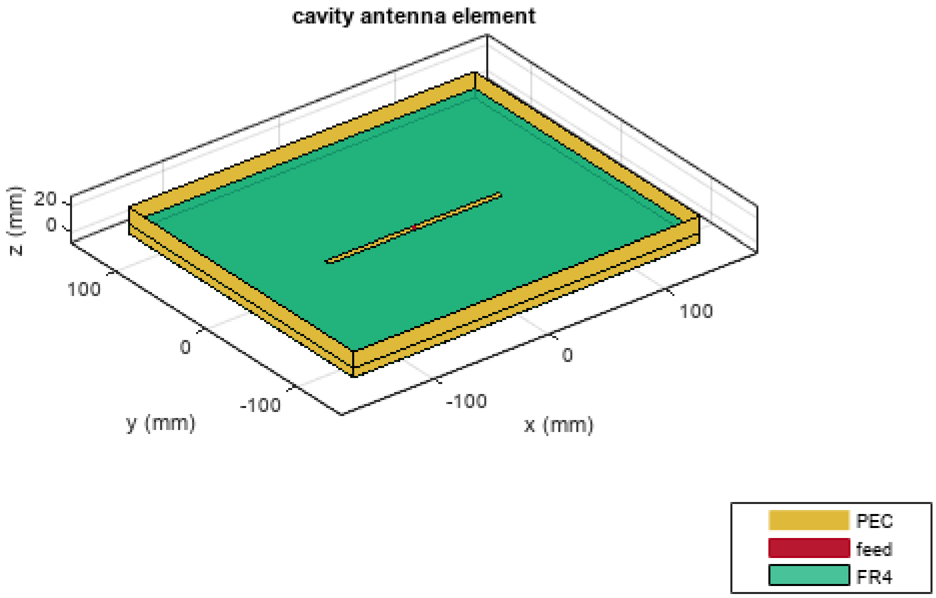
Create a cavity-backed antenna using a dielectric substrate **'FR4'**.

```
d = dielectric('FR4');
c = cavity('Length',30e-2,'Width',25e-2,'Height',20.5e-3,'Spacing',7.5e-3,...
    'Substrate',d)
```

```
c = 
  cavity with properties:

            Exciter: [1x1 dipole]
          Substrate: [1x1 dielectric]
             Length: 0.3000
              Width: 0.2500
             Height: 0.0205
            Spacing: 0.0075
```

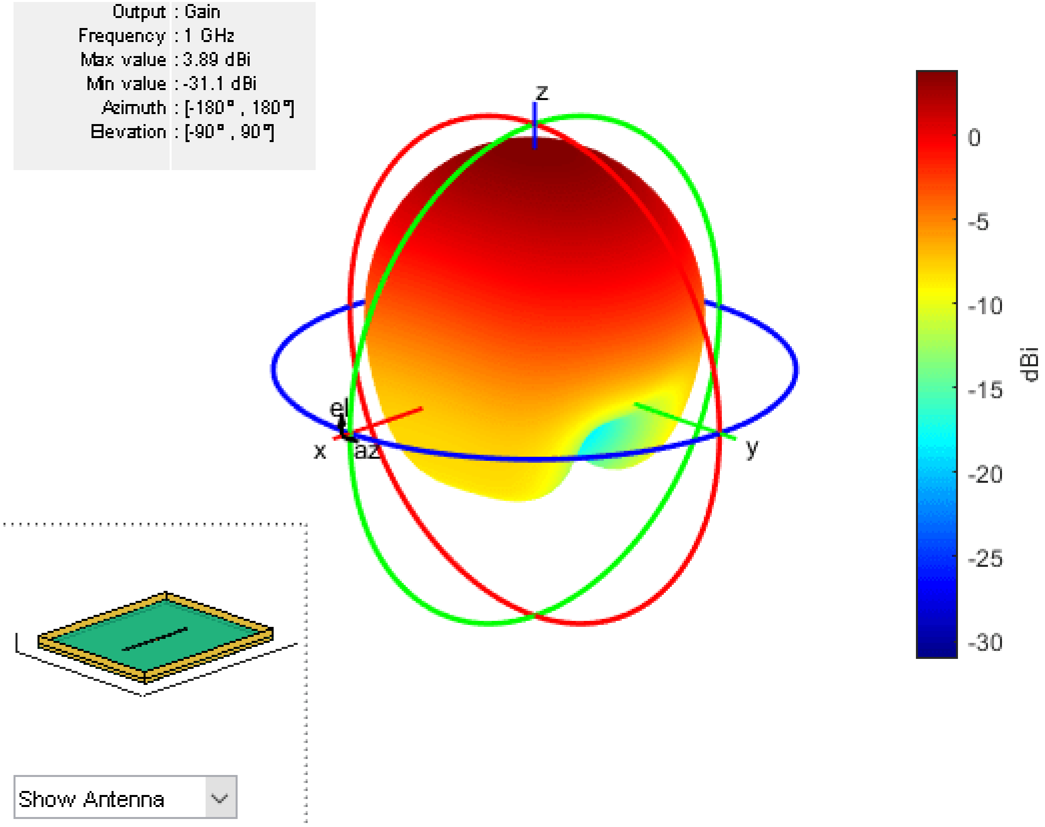
```
    EnableProbeFeed: 0
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(c)
```



Plot the radiation pattern of the antenna at a frequency of 1 GHz.

```
figure
pattern(c,1e9)
```



```
Output : Gain
Frequency : 1 GHz
Max value : 3.89 dBi
Min value : -31.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Create Rectangular Array with Cavity Backing Structure

Create a rectangular array of E-shaped patch antenna.

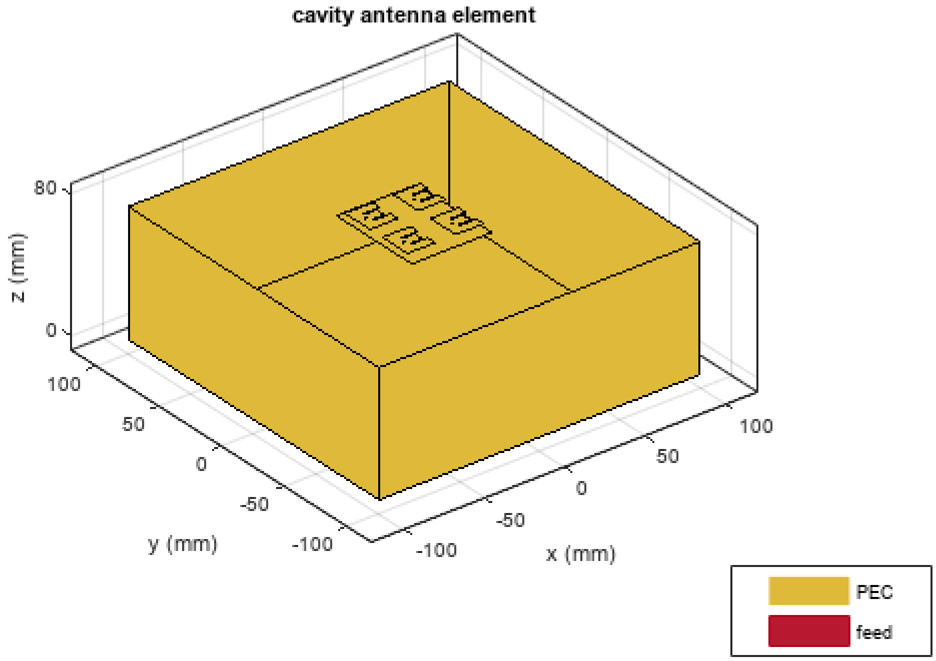
```
rectArr = rectangularArray('Element',patchMicrostripEnotch,'RowSpacing',0.03,'ColumnSpacing',0.0
```

Create a cavity-backed antenna with rectangular array exciter.

```
ant = cavity('Exciter',rectArr)
```

```
ant = 
  cavity with properties:

            Exciter: [1x1 rectangularArray]
          Substrate: [1x1 dielectric]
             Length: 0.2000
              Width: 0.2000
             Height: 0.0750
            Spacing: 0.0750
    EnableProbeFeed: 0
          Conductor: [1x1 metal]
               Tilt: 0
          TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(ant)
```



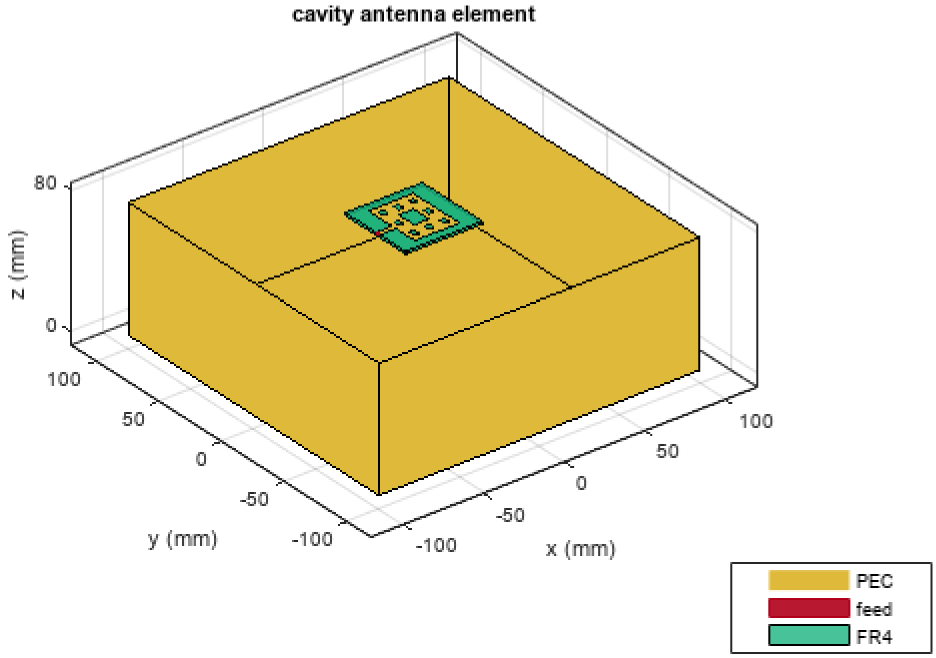
Create Fractal Carpet Antenna with Cavity Backing structure

Create and visualize a cavity-backed fractal carpet antenna

```
e = fractalCarpet('Substrate',dielectric('FR4'));
ant = cavity('Exciter',e)
```

```
ant = 
  cavity with properties:

            Exciter: [1x1 fractalCarpet]
          Substrate: [1x1 dielectric]
             Length: 0.2000
              Width: 0.2000
             Height: 0.0750
            Spacing: 0.0750
    EnableProbeFeed: 0
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A.*Antenna Theory: Analysis and Design*.3rd Ed. New York: Wiley, 2005.

See Also

`spiralArchimedean` | `spiralEquiangular` | `reflector`

Topics

“Rotate Antennas and Arrays”

dipole

Create strip dipole antenna

Description

The `dipole` object is a strip dipole antenna on the *yz*- plane.

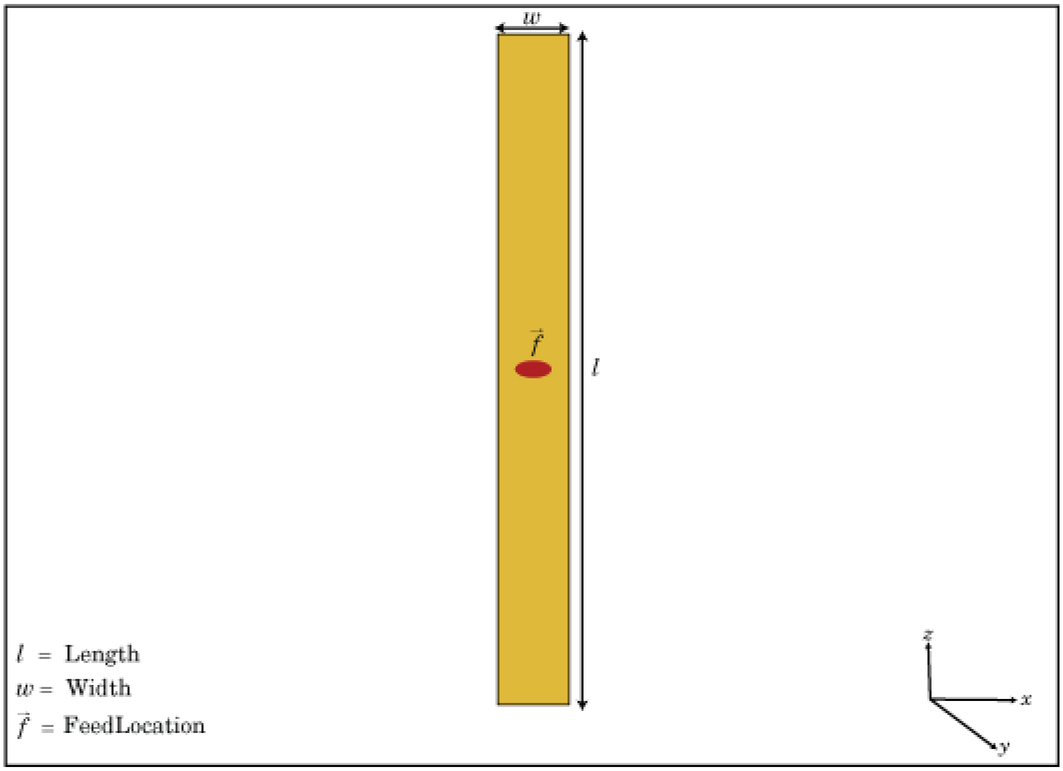
The width of the dipole is related to the diameter of an equivalent cylindrical dipole by the equation

$$w = 2d = 4r$$

where:

- *d* is the diameter of equivalent cylindrical dipole.
- *r* is the radius of equivalent cylindrical dipole.

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default strip dipole is center-fed. The feed point coincides with the origin. The origin is located on the *yz*- plane.



Creation

Syntax

```
d = dipole
d = dipole(Name,Value)
```

Description

`d = dipole` creates a half-wavelength strip dipole antenna on the Y-Z plane.

`d = dipole(Name,Value)` creates a dipole antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties you do not specify retains their default values.

Properties

`Length` — Dipole length
2 (default) | scalar

Dipole length, specified as a scalar in meters. By default, the length is chosen for an operating frequency of 75 MHz.

Example: `'Length',3`

Data Types: `double`

`Width` — Dipole width
0.1000 (default) | scalar

Dipole width, specified as a scalar in meters.

Note Dipole width should be less than `'Length'`/5 and greater than `'Length'`/1001. [2]

Example: `'Width',0.05`

Data Types: `double`

`FeedOffset` — Signed distance from center of dipole
0 (default) | scalar

Signed distance from center of dipole, specified as a scalar in meters. The feed location is on *yz*-plane.

Example: `'FeedOffset',3`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `d.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

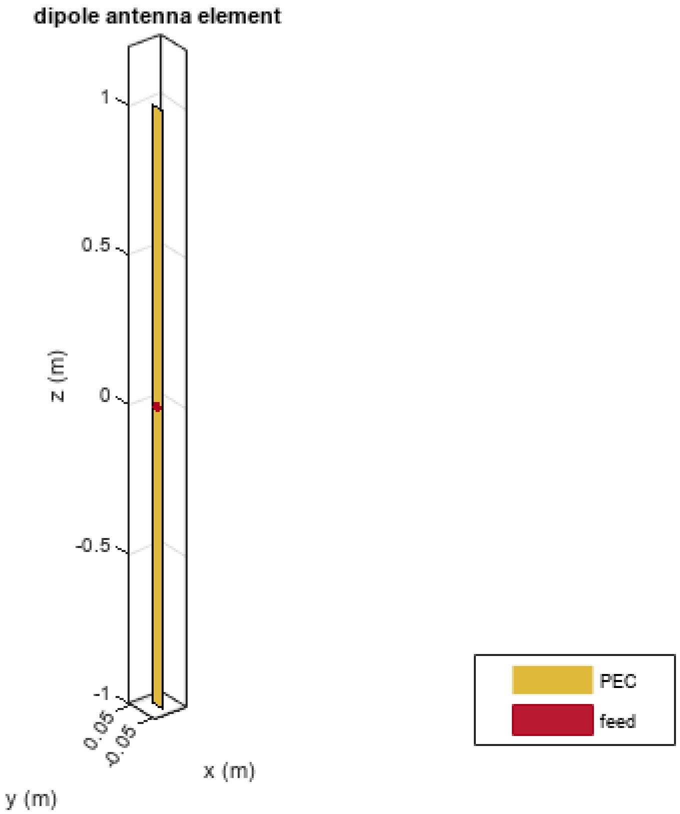
Create and View Dipole Antenna

Create and view a dipole with 2 m length and 0.5 m width.

```
d = dipole('Width',0.05)
```

```
d = 
  dipole with properties:

        Length: 2
         Width: 0.0500
    FeedOffset: 0
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

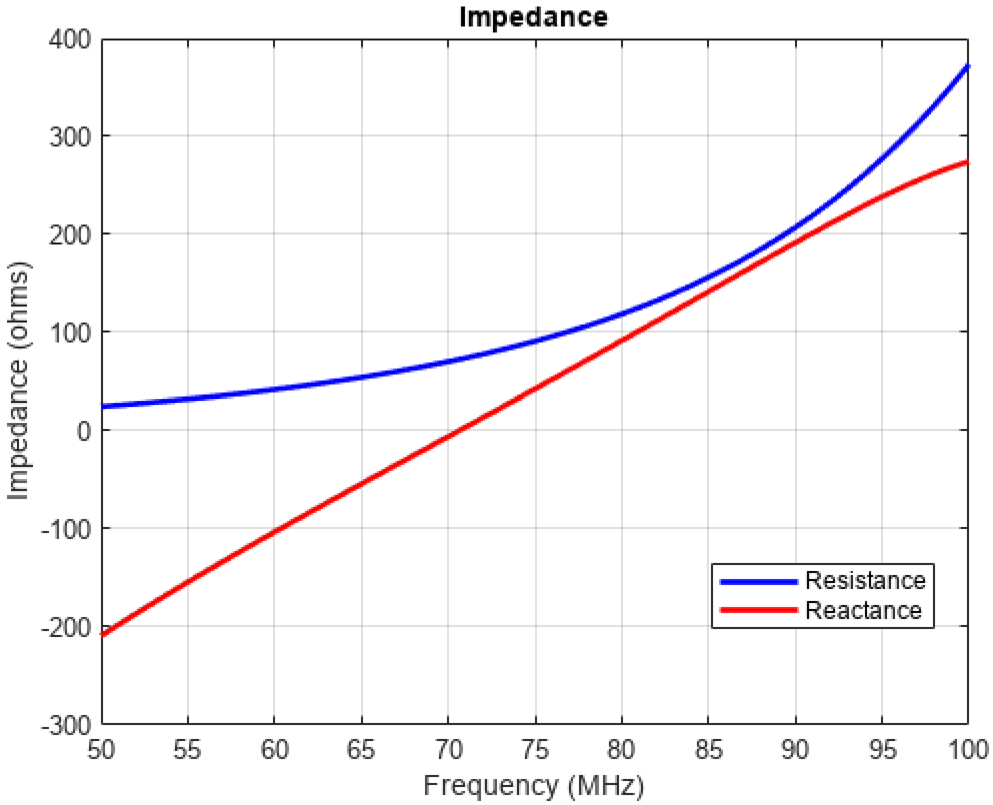
```
show(d)
```



Impedance of Dipole Antenna

Calculate the impedance of a dipole over a frequency range of 50 MHz - 100 MHz.

```
d = dipole('Width',0.05);
impedance(d,linspace(50e6,100e6,51))
```



Infinite Reflector Backed Dielectric Substrate Antenna

Design a dipole antenna backed by a dielectric substrate and an infinite reflector.

Create a dipole antenna of length, 0.15 m, and width, 0.015 m.

```
d = dipole('Length',0.15,'Width',0.015, 'Tilt',90,'TiltAxis',[0 1 0]);
```

Create a reflector using the dipole antenna as an exciter and the dielectric, `teflon` as the substrate.

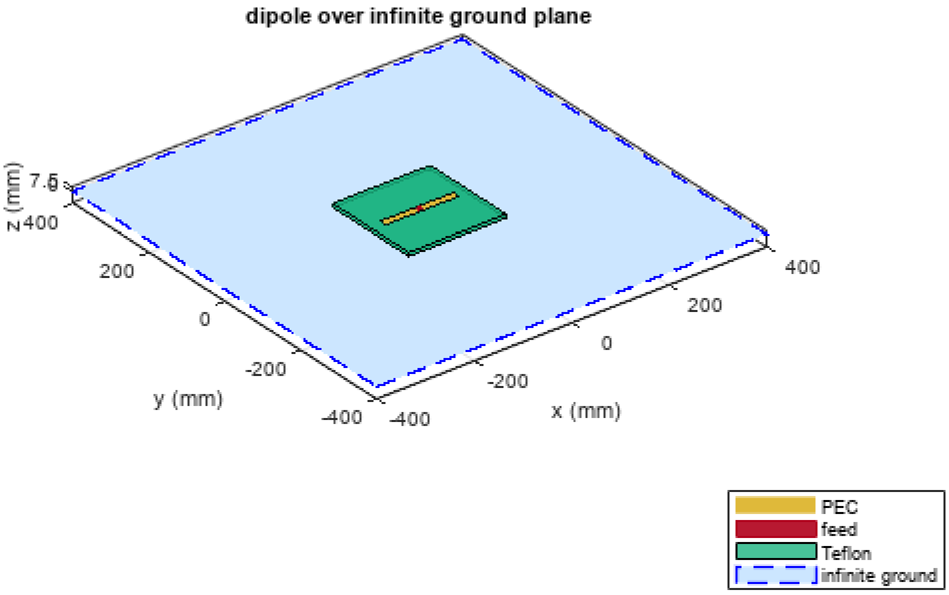
```
t = dielectric('Teflon')
```

```
t = 
  dielectric with properties:

           Name: 'Teflon'
       EpsilonR: 2.1000
    LossTangent: 2.0000e-04
      Thickness: 0.0060

For more materials see catalog
```

```
rf = reflector('Exciter',d,'Spacing',7.5e-3,'Substrate',t);
```

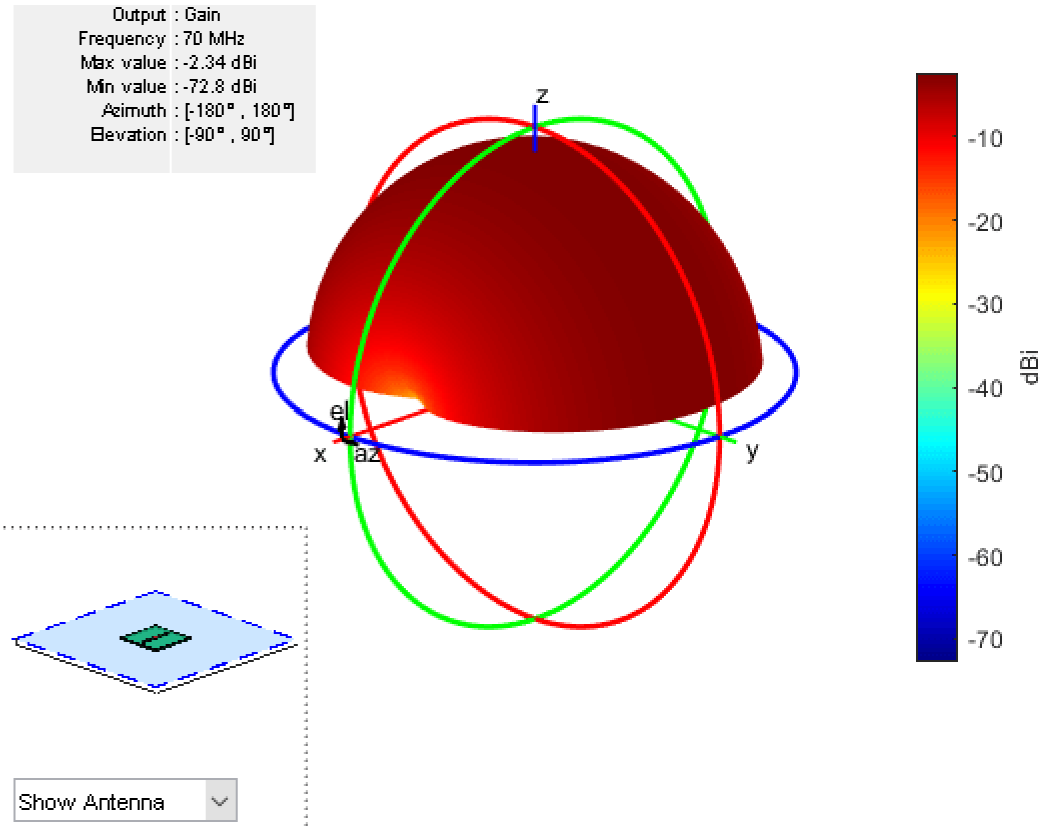
Set the groundplane length of the reflector to `inf`. View the structure.

```
rf.GroundPlaneLength = inf;
show(rf)
```



Calculate the radiation pattern of the antenna at 70 MHz.

```
pattern(rf,70e6)
```



Output : Gain
Frequency : 70 MHz
Max value : -2.34 dBi
Min value : -72.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

See Also

`loopCircular` | `monopole` | `slot` | `cylinder2strip`

Topics

"Rotate Antennas and Arrays"

dipoleFolded

Create folded dipole antenna

Description

The `dipolefolded` object is a folded dipole antenna on the *xy*- plane.

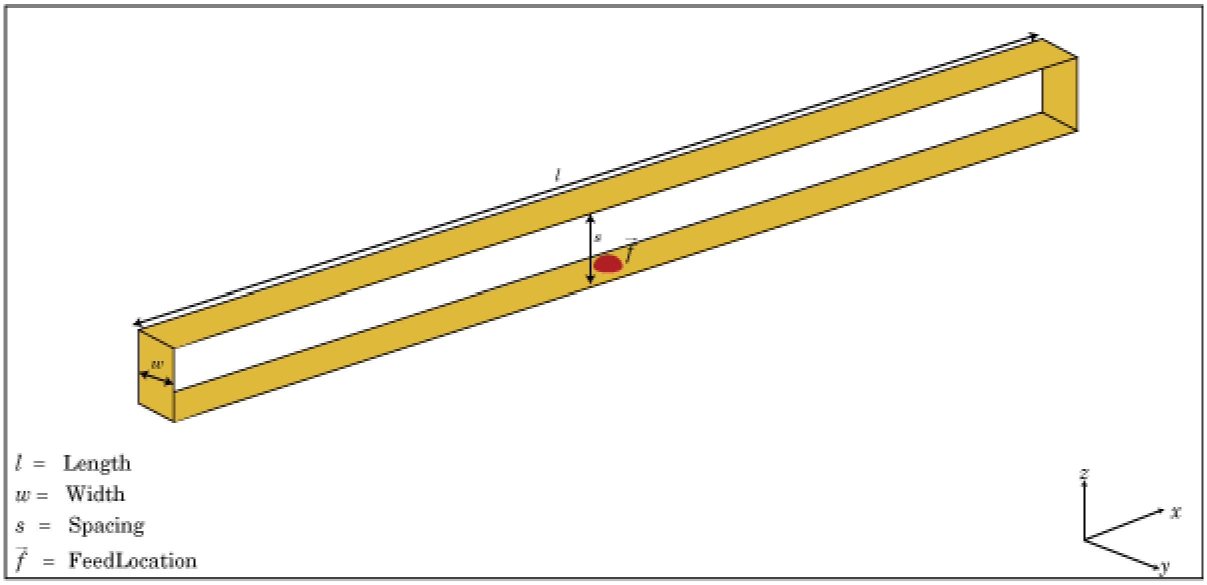
The width of the dipole is related to the diameter of an equivalent cylindrical dipole by the equation

$$w = 2d = 4r$$

, where

- *d* is the diameter of the equivalent cylindrical pole
- *r* is the radius of the equivalent cylindrical pole.

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default folded dipole is center-fed. The feed point of the dipole coincides with the origin. The origin is located on the *xy*- plane. When compared to the planar `dipole`, the folded dipole structure increases the input impedance of the antenna.



Creation

Syntax

```
dF = dipoleFolded
dF = dipoleFolded(Name,Value)
```

Description

`dF = dipoleFolded` creates a half-wavelength folded dipole antenna.

`dF = dipoleFolded(Name,Value)` creates a half-wavelength folded dipole antenna with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Length` — Folded dipole length

2 (default) | scalar

Folded dipole length, specified as a scalar in meters. By default, the length is chosen for an operating frequency of 70.5 MHz.

Example: `'Length',3`

Data Types: `double`

`Width` — Folded dipole width

0.0040 (default) | scalar

Folded dipole width, specified as a scalar in meters.

Note Folded dipole width should be less than `'Length'`/20 and greater than `'Length'`/1001. [2]

Example: `'Width',0.05`

Data Types: `double`

`Spacing` — Shorting stub lengths at dipole ends

0.0245 (default) | scalar

Shorting stub lengths at dipole ends, specified as a scalar in meters. The value must be less than `Length`/50.

Example: `'Spacing',3`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `dF.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90],TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna

charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

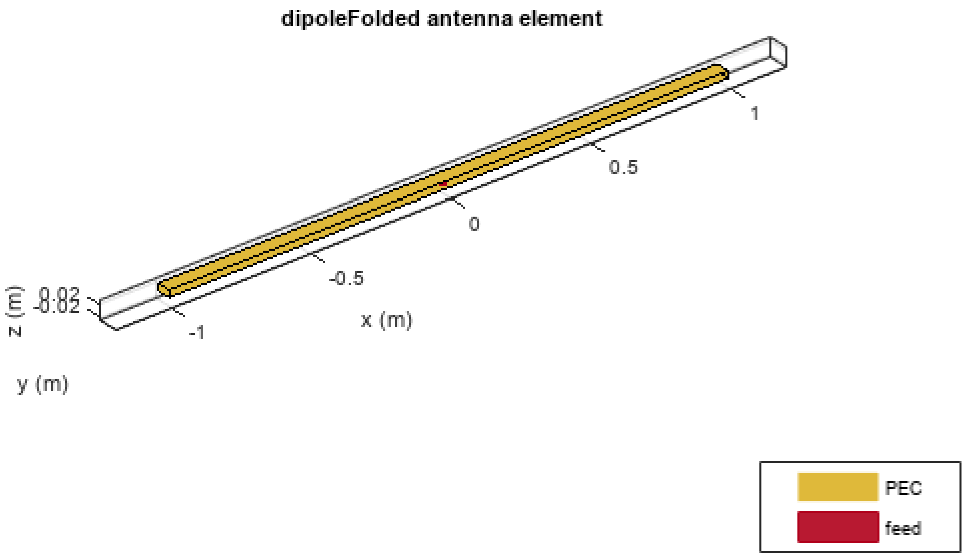
Create and View Folded Dipole Antenna

Create and view a folded dipole with 2 m length and 0.05 m width.

```
df = dipoleFolded('Length',2,'Width',0.05)
```

```
df = 
  dipoleFolded with properties:

       Length: 2
        Width: 0.0500
      Spacing: 0.0245
    Conductor: [1x1 metal]
         Tilt: 0
     TiltAxis: [1 0 0]
         Load: [1x1 lumpedElement]
```

```
show(df)
```



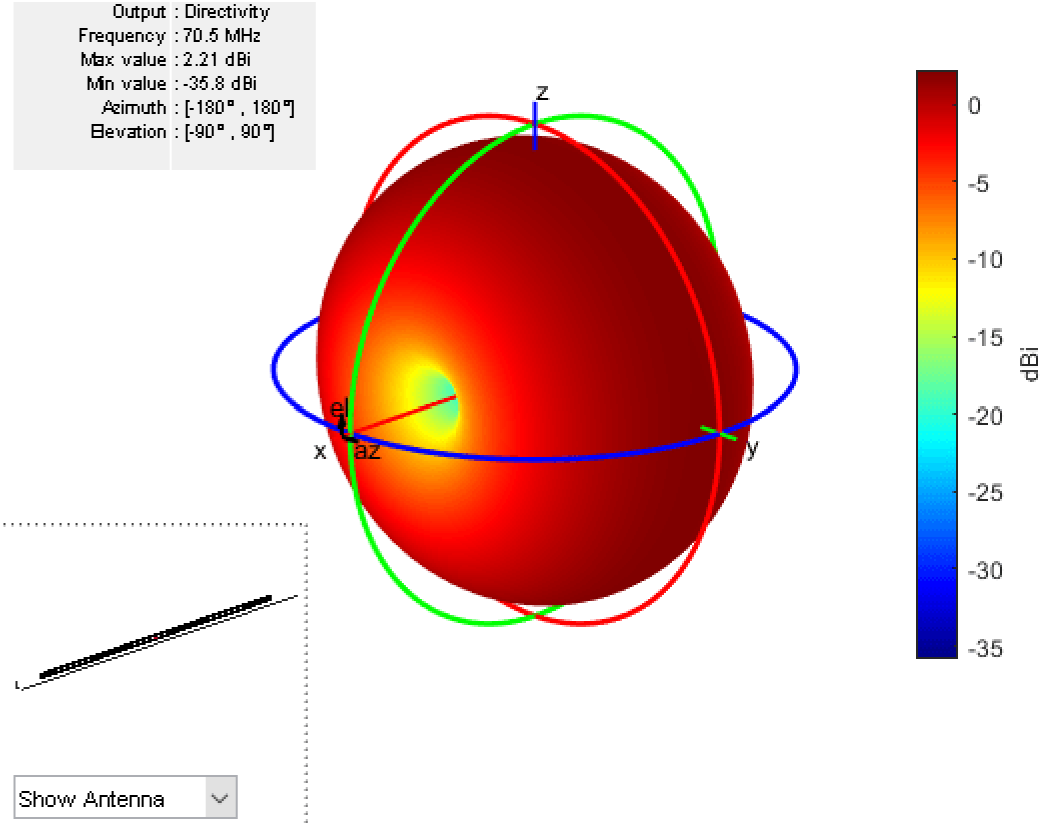
Radiation Pattern of Folded Dipole Antenna

Plot the radiation pattern of a folded dipole at 70.5 MHz.

```
df = dipoleFolded
```

```
df = 
  dipoleFolded with properties:

        Length: 2
         Width: 0.0180
       Spacing: 0.0245
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
pattern(df, 70.5e6);
```



Output : Directivity
Frequency : 70.5 MHz
Max value : 2.21 dBi
Min value : -35.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

See Also

`bowtieTriangular` | `dipole` | `monopole` | `cylinder2strip`

Topics

"Rotate Antennas and Arrays"

dipoleVee

Create V-dipole antenna

Description

The `dipoleVee` object is a planar V-dipole antenna in the *xy*- plane.

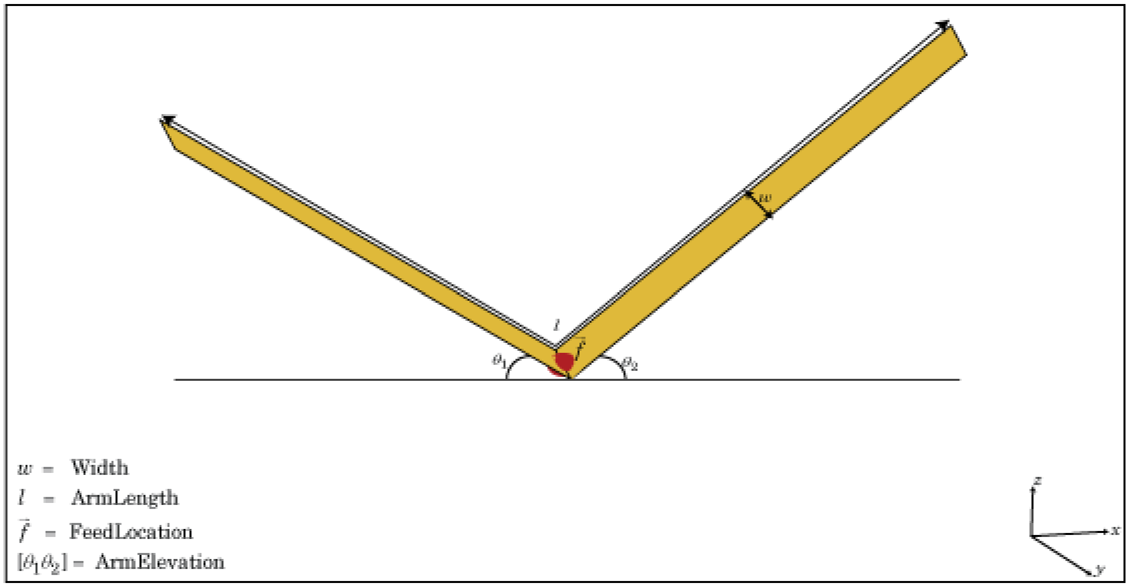
The width of the dipole is related to the circular cross-section by the equation

$$w = 2d = 4r$$

, where:

- d is the diameter of equivalent cylindrical pole
- r is the radius of equivalent cylindrical pole

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The V-dipole antenna is bent around the feed point. The default V-dipole is center-fed and is in the *xy*- plane. The feed point of the V-dipole antenna coincides with the origin.



Creation

Syntax

```
dv = dipoleVee
dv = dipoleVee(Name,Value)
```

Description

`dv = dipoleVee` creates a half-wavelength V-dipole antenna.

`dv = dipoleVee(Name,Value)` creates a half-wavelength V-dipole antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`ArmLength` — Length of two arms

`[1 1]` (default) | two-element vector

Length of two arms, specified as a two-element vector in meters. By default, the arm lengths are chosen for an operating frequency of 75 MHz.

Example: `'ArmLength',[1,3]`

Data Types: `double`

`Width` — V-dipole arm width

0.1000 (default) | scalar

V-dipole arm width, specified as a scalar in meters.

Note Dipole width should be less than `Total Arm Length`/5 and greater than `Total Arm Length`/1001. [2]

Example: `'Width',0.05`

Data Types: `double`

`ArmElevation` — Angle made by two arms about *xy*- plane

`[45 45]` (default) | two-element vector

Angle made by two arms about *xy*- plane, specified as a two-element vector in degrees.

Example: `'ArmElevation',[55 35]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `dv.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna

charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

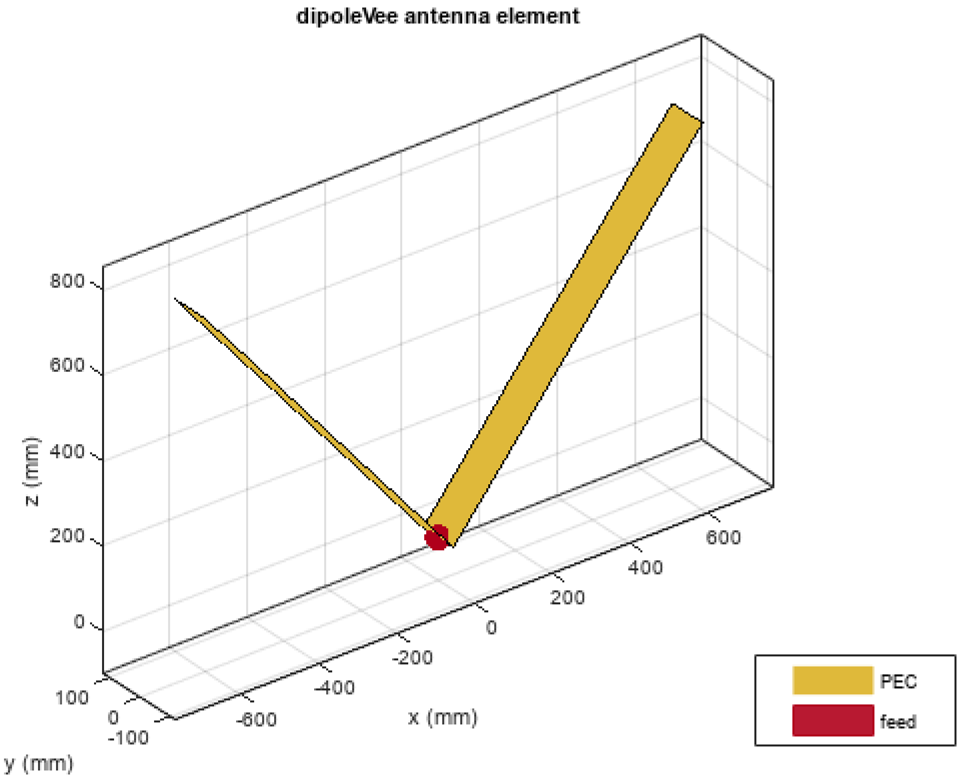
Create V-Dipole Antenna

Create and view a center-fed V-dipole that has 50 degree arm angles .

```
dv = dipoleVee('ArmElevation',[50 50])
```

```
dv = 
  dipoleVee with properties:

       ArmLength: [1 1]
    ArmElevation: [50 50]
           Width: 0.1000
       Conductor: [1x1 metal]
            Tilt: 0
        TiltAxis: [1 0 0]
            Load: [1x1 lumpedElement]
```

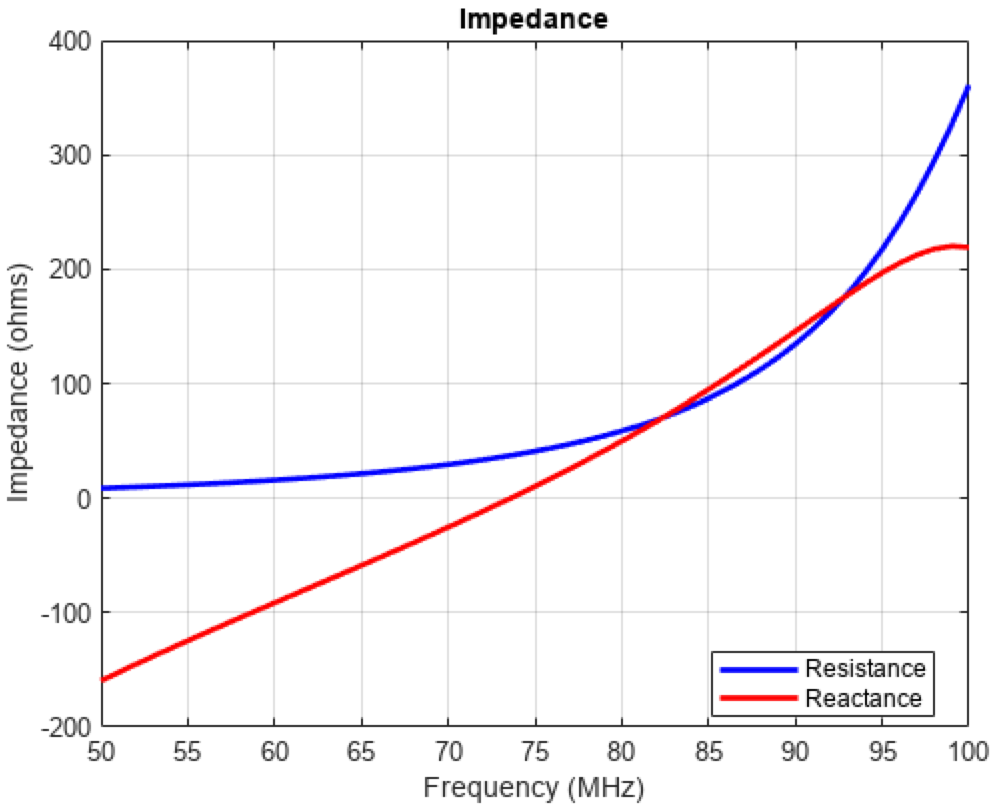
```
show(dv)
```



Impedance of V-Dipole Antenna

Calculate the impedance of a V-dipole antenna over the frequency range of 50 MHz - 100 MHz.

```
dv = dipoleVee('ArmElevation',[50 50]);
impedance(dv,linspace(50e6,100e6,51))
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*. 4th Ed. New York: McGraw-Hill, 2007.

See Also

`dipole` | `dipoleFolded` | `loopCircular` | `cylinder2strip`

Topics

“Modeling Wire Antenna and Arrays”
“Rotate Antennas and Arrays”

dipoleMeander

Create meander dipole antenna

Description

The `dipoleMeander` class creates a meander dipole antenna with four dipoles. The antenna is center fed and it is symmetric about its origin. The first resonance of meander dipole antenna is at 200 MHz.

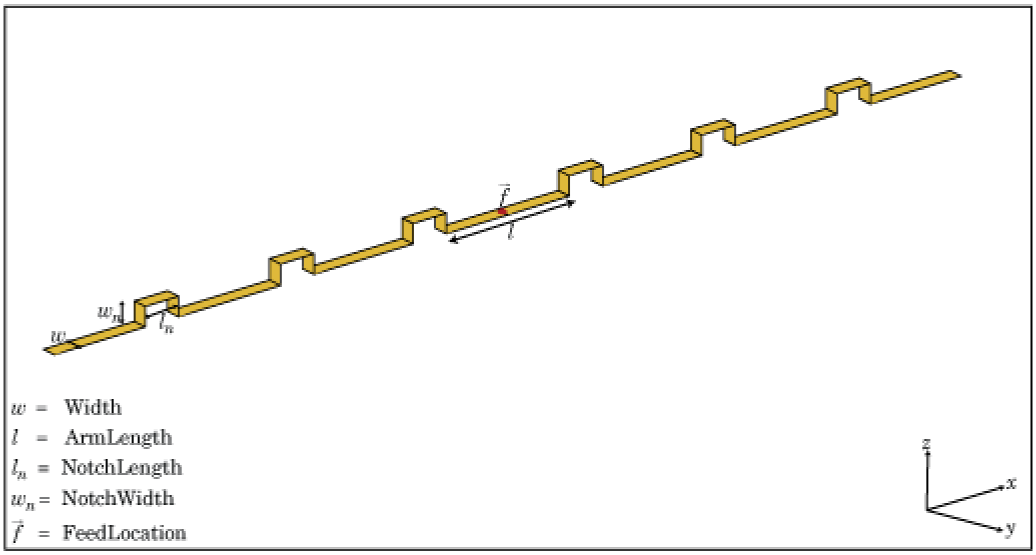
The width of the dipole is related to the diameter of an equivalent cylindrical dipole by the equation

$$w = 2d = 4r$$

, where:

- *d* is the diameter of equivalent cylindrical dipole.
- *r* is the radius of equivalent cylindrical dipole.

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default strip dipole is center-fed. The feed point coincides with the origin. The origin is located on the *xy*- plane.



Creation

Syntax

```
dm = dipoleMeander
dm = dipoleMeander(Name,Value)
```

Description

dm = dipoleMeander creates a meander dipole antenna with four dipoles.

dm = dipoleMeander(Name,Value) creates a meander dipole antenna with four dipoles, with additional properties specified by one or more name-value pair arguments. Name is the property name and Value is the corresponding value. You can specify several name-value pair arguments in any order as Name1, Value1, ..., NameN, ValueN. Properties not specified retain their default values.

Properties

Width — Dipole width
0.0040 (default) | scalar

Dipole width, specified as a scalar in meters.

Example: 'Width',0.05

Data Types: double

ArmLength — Length of individual dipole arms
[0.0880 0.0710 0.0730 0.0650] (default) | vector

Length of individual dipole arms, specified as a vector in meters. The total number of dipole arms generated is :

$$2 * N - 1$$

where *N* is the number of specified arm lengths.

Example: 'ArmLength',[0.6000 0.5000 1 0.4000]

Data Types: double

NotchLength — Notch length along length of antenna
0.0238 (default) | scalar

Notch length along the length of the antenna, specified as a scalar in meters.

For example, in a dipole meander antenna with seven stacked arms there are six notches.

Example: 'NotchLength',1

Data Types: double

NotchWidth — Notch width perpendicular to length of antenna
0.0238 (default) | scalar

Notch width perpendicular to the length of the antenna, specified as a scalar in meters.

Example: 'NotchWidth',1

Data Types: double

Conductor — Type of metal material
'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `dm.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

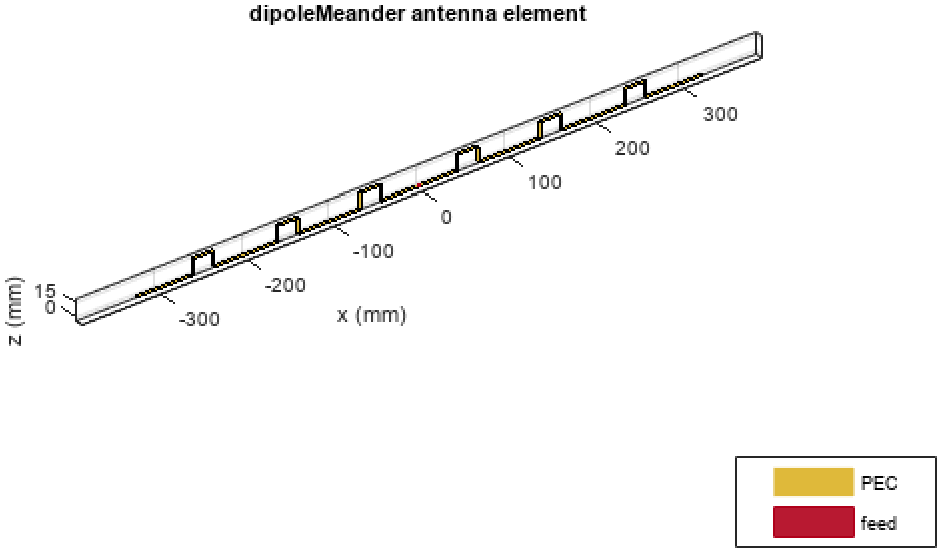
Create and View Meander Dipole Antenna

Create and view the default meander dipole antenna.

```
dm = dipoleMeander
```

```
dm = 
  dipoleMeander with properties:

          Width: 0.0040
      ArmLength: [0.0880 0.0710 0.0730 0.0650]
    NotchLength: 0.0238
     NotchWidth: 0.0170
      Conductor: [1x1 metal]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

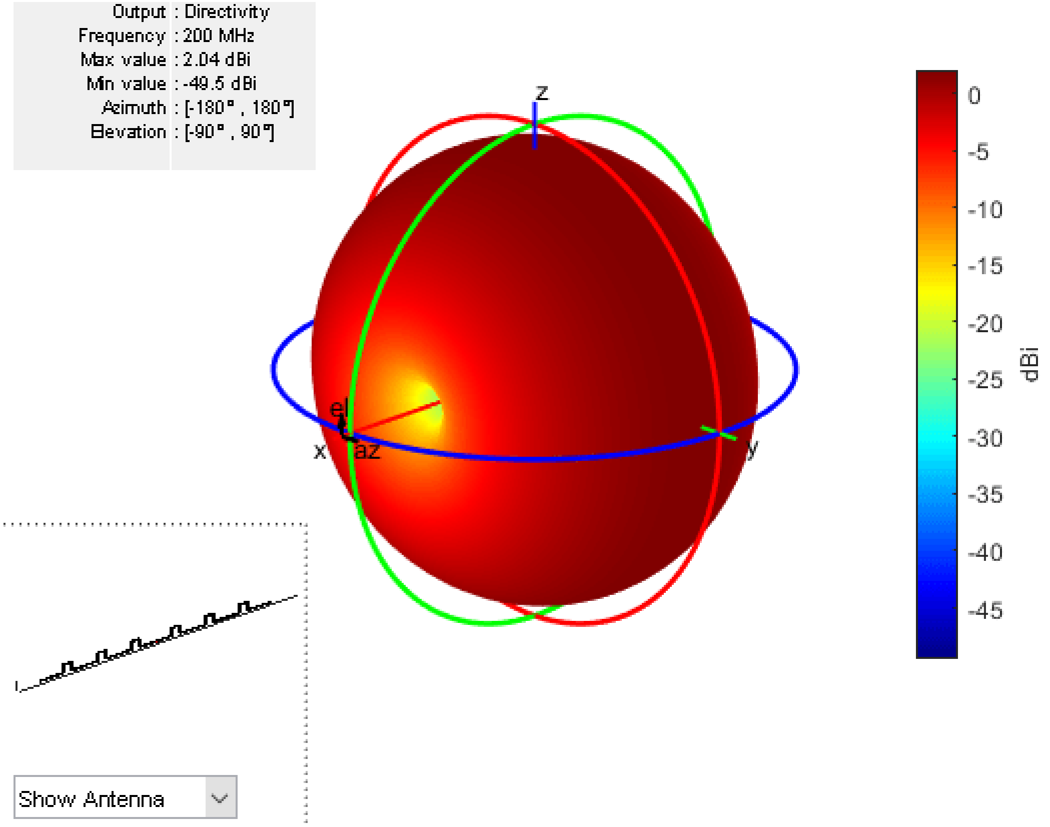
```
show(dm)
```



Plot Radiation Pattern Of Meander Dipole Antenna

Plot the radiation pattern of meander dipole antenna at a 200 MHz frequency.

```
dm = dipoleMeander;
pattern(dm,200e6)
```



Output : Directivity
Frequency : 200 MHz
Max value : 2.04 dBi
Min value : -49.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

See Also

`loopCircular` | `dipole` | `dipoleFolded`

Topics

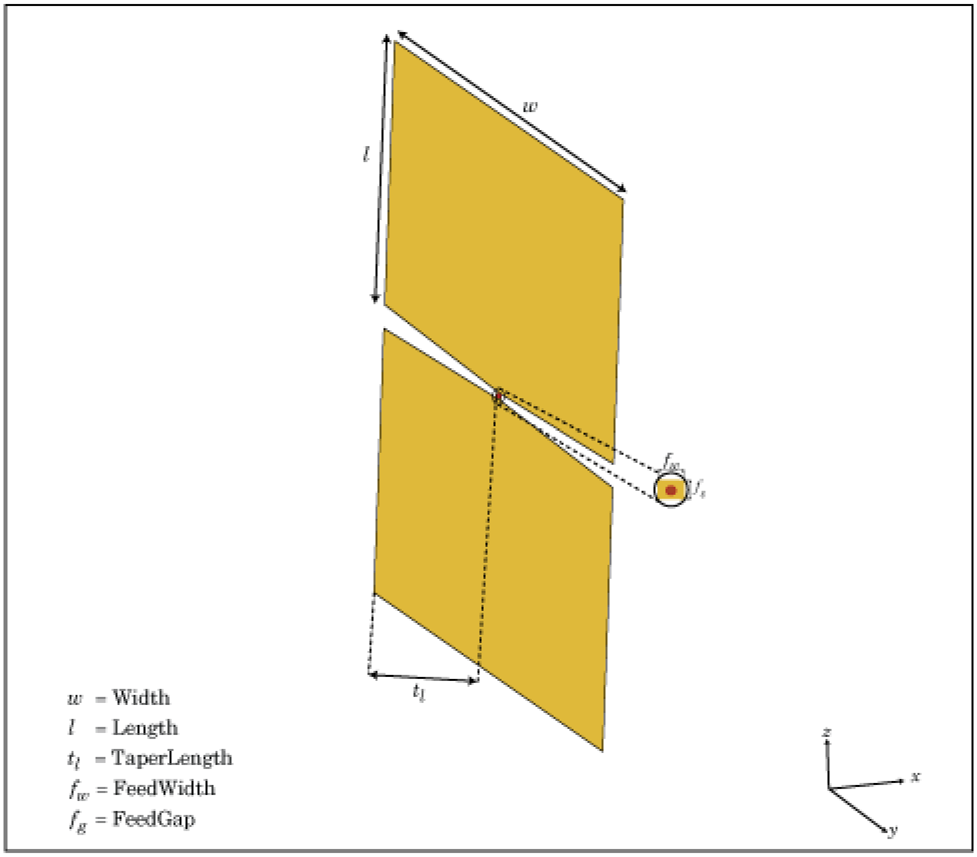
"Rotate Antennas and Arrays"

dipoleBlade

Create blade dipole antenna

Description

The `dipoleBlade` object is a wideband blade dipole antenna oriented along the *xy*- plane.



The width of the dipole is related to the circular cross-section by the equation,

$$w = 2d = 4r$$

, where:

- *d* is the diameter of equivalent cylindrical pole
- *r* is the radius of equivalent cylindrical pole

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width.

Creation

Syntax

```
db = dipoleBlade
db = dipoleBlade(Name,Value)
```

Description

`db = dipoleBlade` creates a wideband blade dipole antenna on the X-Y plane.

`db = dipoleBlade(Name,Value)` creates a wideband blade dipole antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Length` — Blade dipole length
`0.1170` (default) | scalar

Blade dipole length, specified as a scalar in meters.

Example: `'Length',0.5`

Data Types: `double`

`Width` — Blade dipole width
`0.1400` (default) | scalar

Blade dipole width, specified as a scalar in meters.

Example: `'Width',0.2`

Data Types: `double`

`TaperLength` — Taper length
`0.1120` (default) | scalar

Taper length, specified as a scalar in meters.

Example: `'TaperLength',0.500`

Data Types: `double`

`FeedWidth` — Blade dipole feed width
`0.0030` (default) | scalar

Blade dipole feed width, specified as a scalar in meters.

Example: `'FeedWidth',0.006`

Data Types: `double`

`FeedGap` — Blade dipole feed length or distance between the two wings of the dipole
`0.0030` (default) | scalar

Blade dipole feed length or distance between the two wings of the dipole, specified as a scalar in meters.

Example: `'FeedGap',0.006`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `db.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

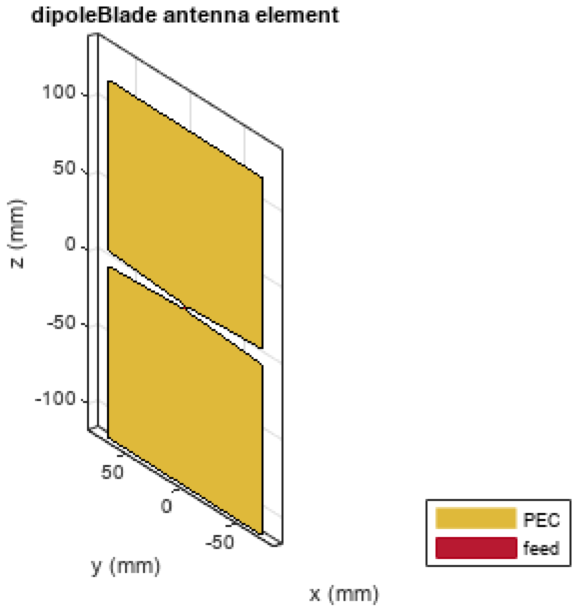
Default Blade Dipole and Radiation Pattern

Create and view a default blade dipole.

```
db = dipoleBlade
```

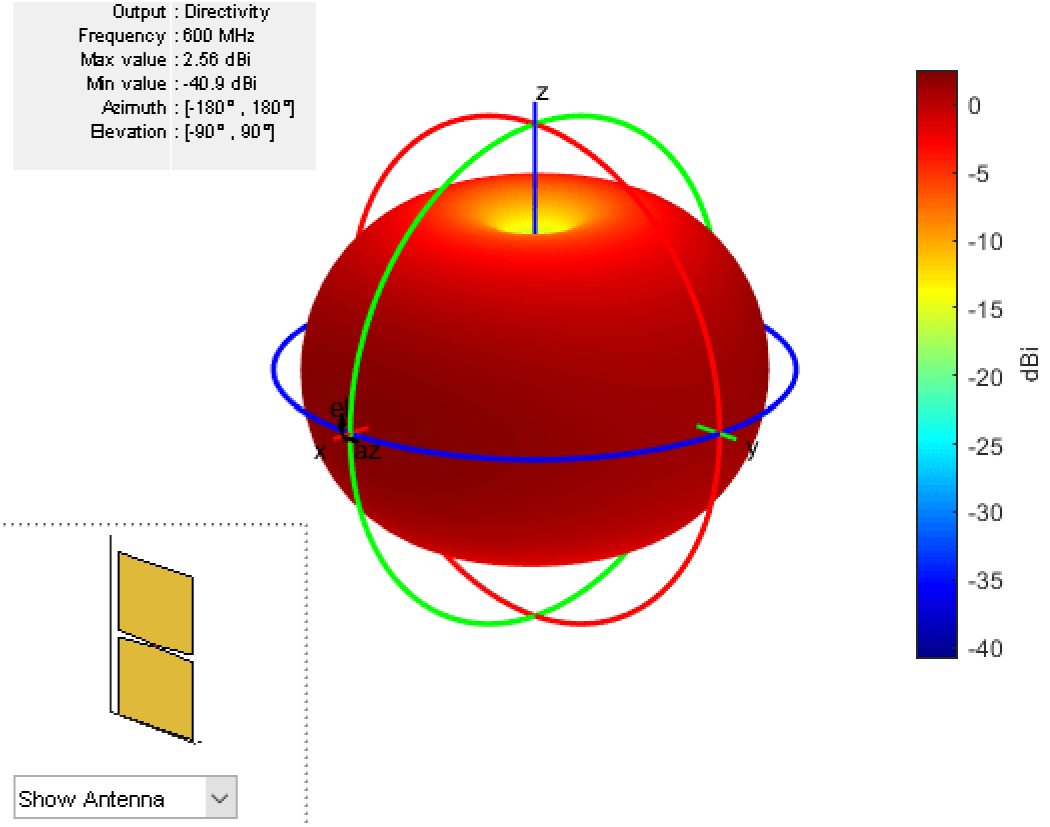
```
db = 
  dipoleBlade with properties:

         Length: 0.1170
    TaperLength: 0.1120
          Width: 0.1400
      FeedWidth: 0.0030
        FeedGap: 0.0030
      Conductor: [1x1 metal]
           Tilt: 0
      TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

```
show(db);
```



Plot the radiation pattern of the blade dipole at 600 MHz.

```
pattern(db,600e6)
```



Output : Directivity
Frequency : 600 MHz
Max value : 2.56 dBi
Min value : -40.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2017a

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*. 4th Ed. New York: McGraw-Hill, 2007.

See Also

`dipole` | `slot` | `loopCircular` | `cylinder2strip`

Topics

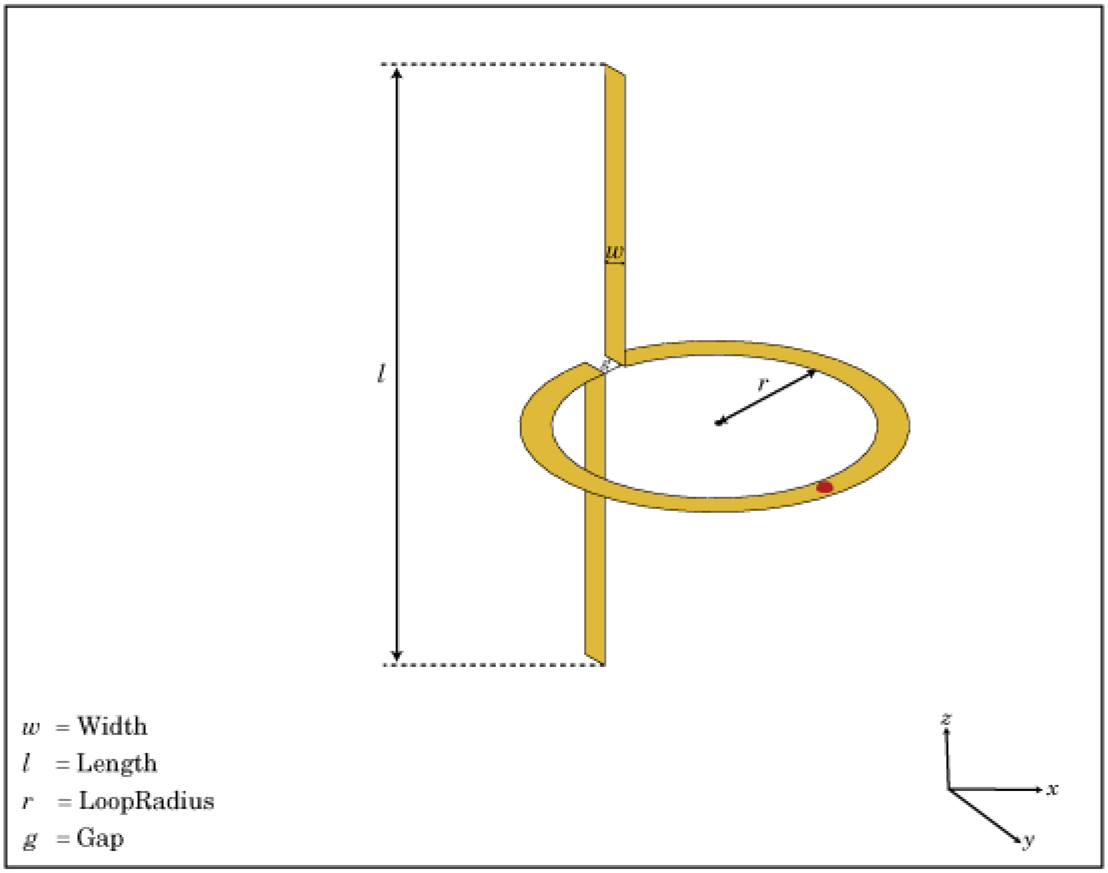
"Rotate Antennas and Arrays"

dipoleCycloid

Create cycloid dipole antenna

Description

The `dipoleCycloid` object is a half-wavelength cycloid dipole antenna. For the default cycloid dipole, the feed point is on the loop section. The default length is for an operating frequency of 48 MHz.



The width of the dipole is related to the circular cross-section by the equation

$$w = 2d = 4r$$

, where:

- *d* is the diameter of equivalent cylindrical pole
- *r* is the radius of equivalent cylindrical pole

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width.

Creation

Syntax

```
dc = dipoleCycloid
dc = dipoleCycloid(Name,Value)
```

Description

`dc = dipoleCycloid` creates a half-wavelength cycloid dipole antenna oriented along Z-axis.

`dc = dipoleCycloid(Name,Value)` creates a half-wavelength cycloid dipole antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Length` — Dipole length along *z*-axis
`1.2200` (default) | scalar

Dipole length along *z*-axis, specified as a scalar in meters. By default, the length is for an operating frequency of 48 MHz.

Example: `'Length',0.9`

Data Types: `double`

`Width` — Dipole width
`0.0508` (default) | scalar

Dipole width, specified as a scalar in meters.

Example: `'Width',0.09`

Data Types: `double`

`LoopRadius` — Circular loop radius in *xy*- plane
`0.3100` (default) | scalar

Circular loop radius in *xy*- plane, specified as a scalar in meters.

Example: `'LoopRadius',0.500`

Data Types: `double`

`Gap` — Gap of loop in *xy*- plane
`0.0400` (default) | scalar

Gap of loop in *xy*- plane, specified as a scalar in meters.

Example: `'Gap',0.006`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as the comma-separated pair consisting of `'Load'` and a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `dc.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: double

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

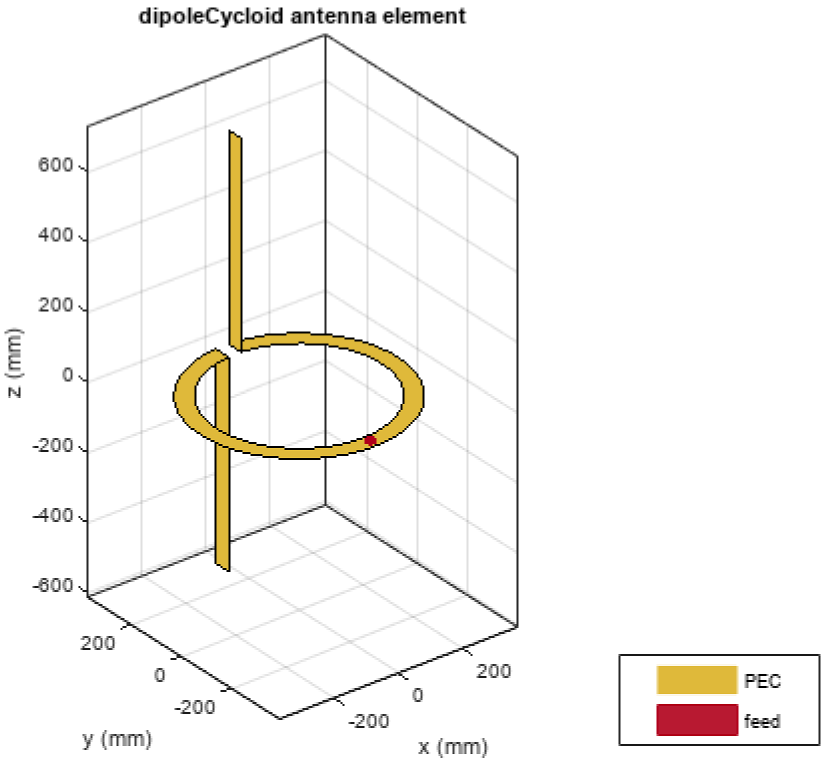
Default Cycloid Dipole

Create a default cycloid dipole antenna using the dipoleCycloid object and view it.

```
dc = dipoleCycloid
```

```
dc = 
  dipoleCycloid with properties:

        Length: 1.2200
         Width: 0.0508
    LoopRadius: 0.3100
           Gap: 0.0400
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

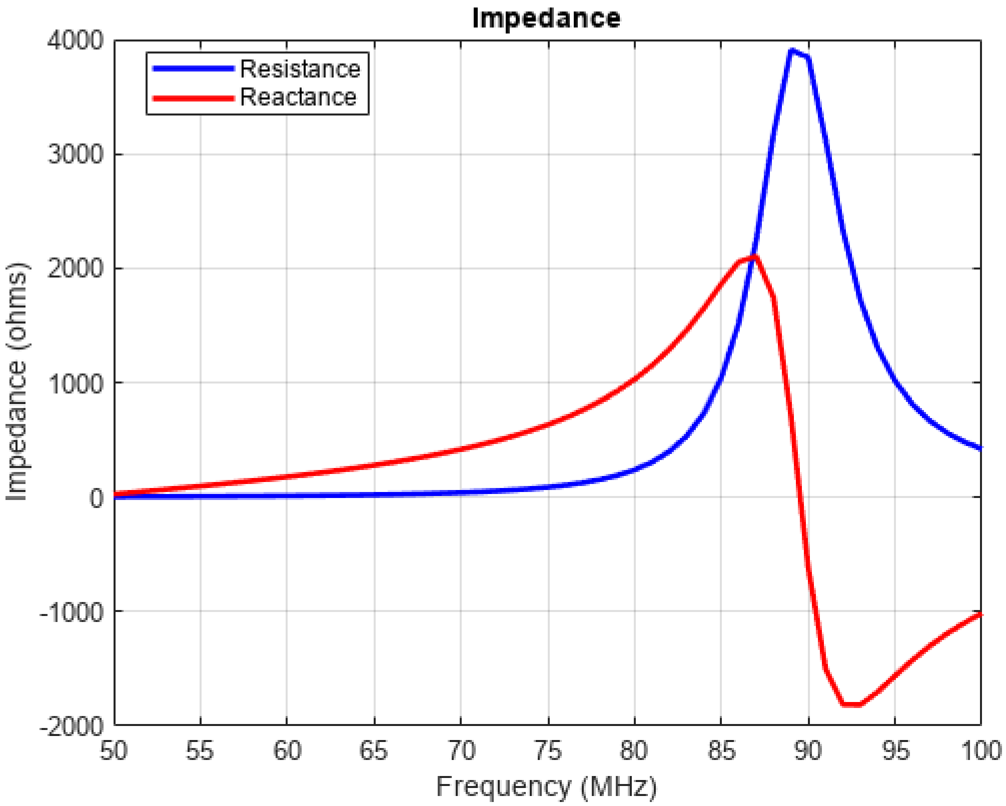
```
show(dc)
```



Impedance of Cycloid Dipole

Calculate the impedance of a cycloid dipole of width, 0.05 m, over a frequency span of 50 MHz - 100 MHz.

```
d = dipoleCycloid('Width',0.05);
impedance(d,linspace(50e6,100e6,51))
```



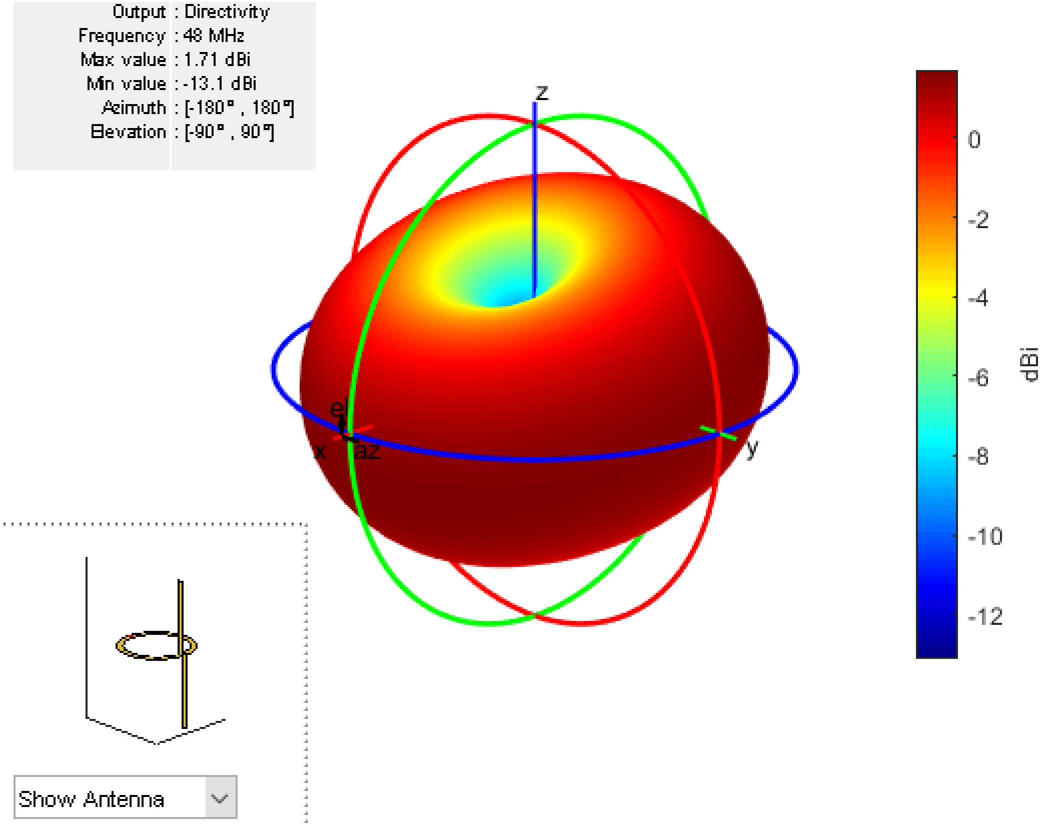
Radiation Pattern of Cycloid Dipole

Plot the radiation pattern of a cycloid dipole of width,0.05 m, at a frequency of 48 MHz.

```
d = dipoleCycloid('Width',0.05)
```

```
d = 
  dipoleCycloid with properties:

        Length: 1.2200
         Width: 0.0500
    LoopRadius: 0.3100
           Gap: 0.0400
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
pattern(d,48e6)
```



Output : Directivity
Frequency : 48 MHz
Max value : 1.71 dBi
Min value : -13.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2017a

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*. 4th Ed. New York: McGraw-Hill, 2007.

See Also

`dipole` | `slot` | `loopCircular` | `cylinder2strip`

Topics

"Rotate Antennas and Arrays"

dipoleHelix

Create helical dipole antenna

Description

The `dipoleHelix` object is a helical dipole antenna. The antenna is typically center-fed. You can move the feed along the antenna length using the feed offset property. Helical dipoles are used in satellite communications and wireless power transfers.

The width of the strip is related to the diameter of an equivalent cylinder by this equation

$$w = 2d = 4r$$

where:

- w is the width of the strip.
- d is the diameter of an equivalent cylinder.
- r is the radius of an equivalent cylinder.

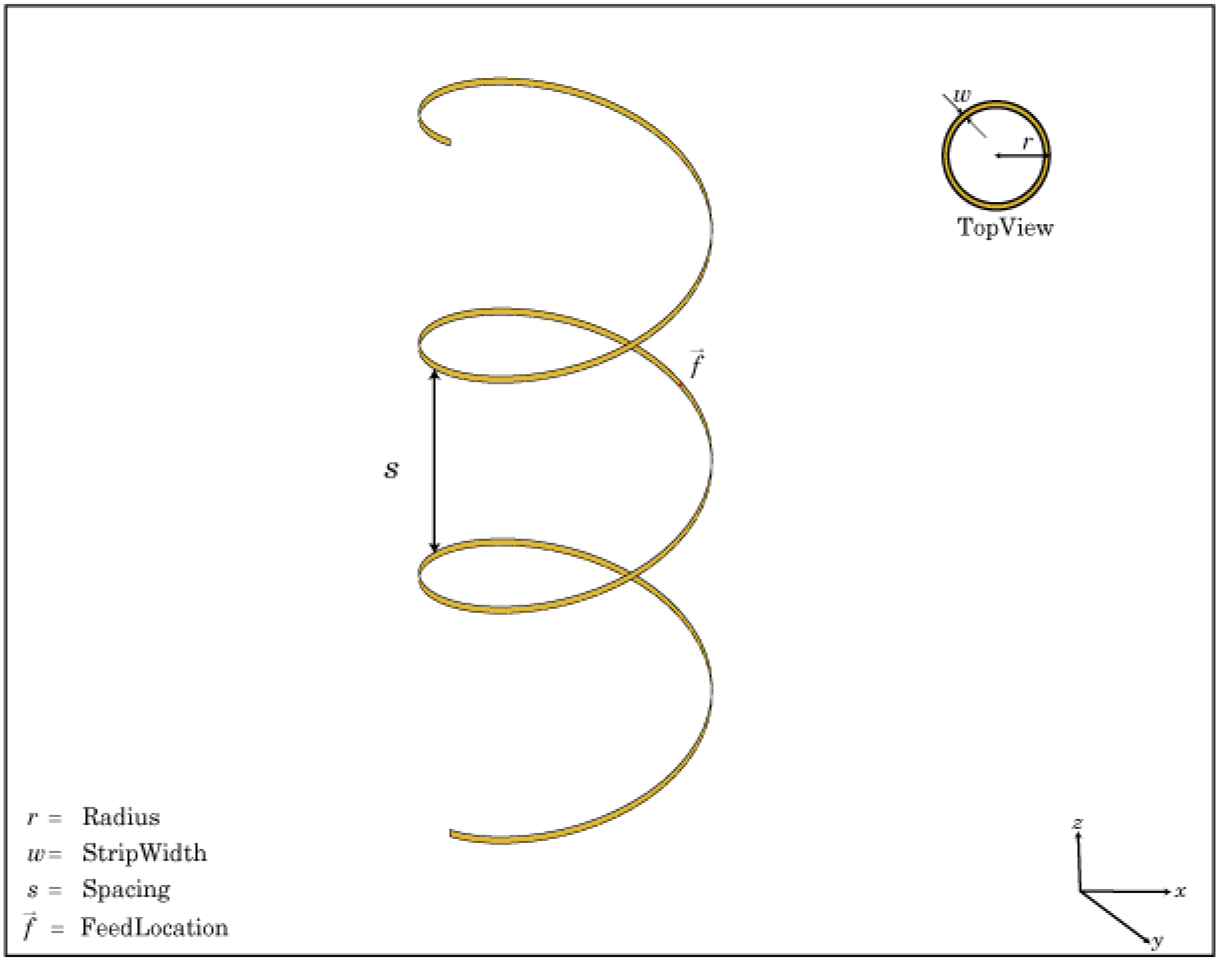
For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default helical dipole antenna is center-fed. Commonly, helical dipole antennas are used in axial mode. In this mode, the helical dipole circumference is comparable to the operating wavelength, and has maximum directivity along its axis. In normal mode, the helical dipole radius is small compared to the operating wavelength. In this mode, the helical dipole radiates broadside, that is, in the plane perpendicular to its axis. The basic equation for the helical dipole antenna is:

$$x = r\cos(\theta)$$
$$y = r\sin(\theta)$$
$$z = S\theta$$

where:

- r is the radius of the helical dipole.
- θ is the winding angle.
- S is the spacing between turns.

For a given pitch angle in degrees, use the `helixpitch2spacing` utility function to calculate the spacing between the turns in meters.



Creation

Syntax

```
dh = dipoleHelix
dh = dipoleHelix(Name=Value)
```

Description

`dh = dipoleHelix` creates a helical dipole antenna. The default antenna operates at around 2 GHz.

`dh = dipoleHelix(Name=Value)` sets "Properties" on page 1-69 using one or more name–value arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value arguments in any order as `Name1=Value1`, `...`, `NameN=ValueN`. Properties that you do not specify retain their default values.

Properties

`Radius` — Turn radius

`0.0220` (default) | scalar

Turn radius, specified as a scalar in meters.

Example: `Radius=2`

Data Types: `double`

`Width` — Strip width

`1.0000e-03` (default) | scalar

Strip width, specified as a scalar in meters.

Note Strip width should be less than `'Radius'`/5 and greater than `'Radius'`/250. [4]

Example: `Width=5`

Data Types: `double`

`Turns` — Number of turns of helical dipole

`3` (default) | scalar

Number of turns of the helical dipole, specified a scalar.

Example: `Turns=2`

Data Types: `double`

`Spacing` — Spacing between turns

`0.0350` (default) | scalar

Spacing between turns, specified as a scalar in meters.

Example: `Spacing=1.5`

Data Types: `double`

`WindingDirection` — Direction of helical dipole turns (windings)

`'CCW'` (default) | `'CW'`

Direction of helical dipole turns (windings), specified as `'CW'` or `'CCW'`.

Example: `WindingDirection='CW'`

Data Types: `char` | `string`

`Substrate` — Type of dielectric material

`'Air'` (default) | `dielectric` object

Type of dielectric material used as the substrate, specified as a `dielectric` object. You can specify only one dielectric layer in the `dipoleHelix` object. Specify the same radius for all the turns. When you use a dielectric material other than air, the number of turns in the dipole helix must be greater than 1. For more information on dielectric substrate meshing, see "Meshing".

Example: `Substrate=dielectric('Teflon')`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `Conductor=metal('Copper');`

Load — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `Load=lumpedElement(Impedance=75)`

`FeedOffset` — Signed distance of feedpoint from origin
`0` (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector in meters. Use this property to adjust the location of the feedpoint relative to the ground plane and patch.

Example: `FeedOffset=[0.01 0.01]`

Data Types: `double`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Helical Dipole Antenna

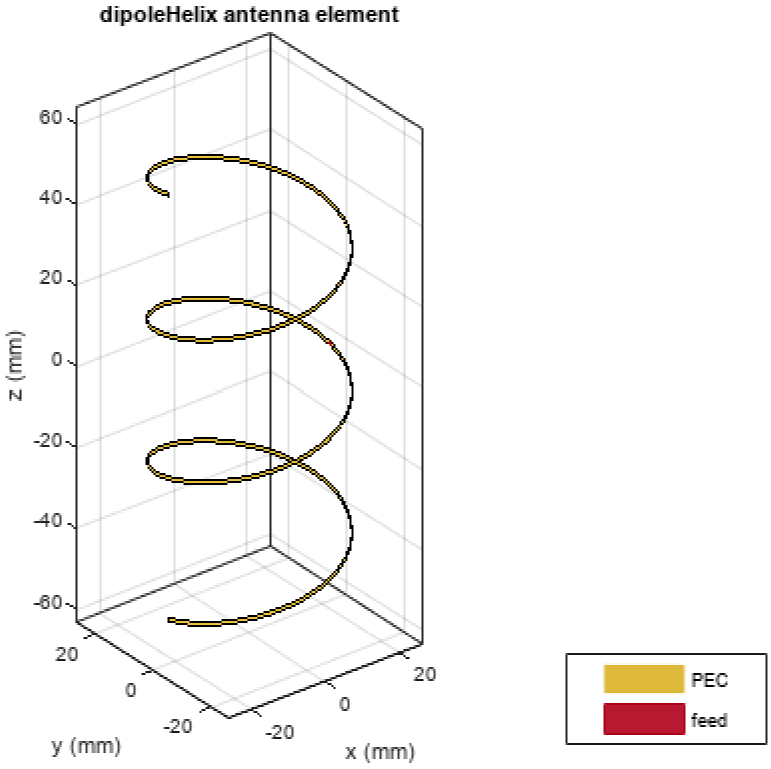
Create a default helical dipole antenna and view it.

```
dh = dipoleHelix
```

```
dh = 
  dipoleHelix with properties:

              Radius: 0.0220
               Width: 1.0000e-03
               Turns: 3
             Spacing: 0.0350
    WindingDirection: 'CCW'
          FeedOffset: 0
           Substrate: [1x1 dielectric]
           Conductor: [1x1 metal]
```

```
        Tilt: 0
    TiltAxis: [1 0 0]
        Load: [1x1 lumpedElement]
```

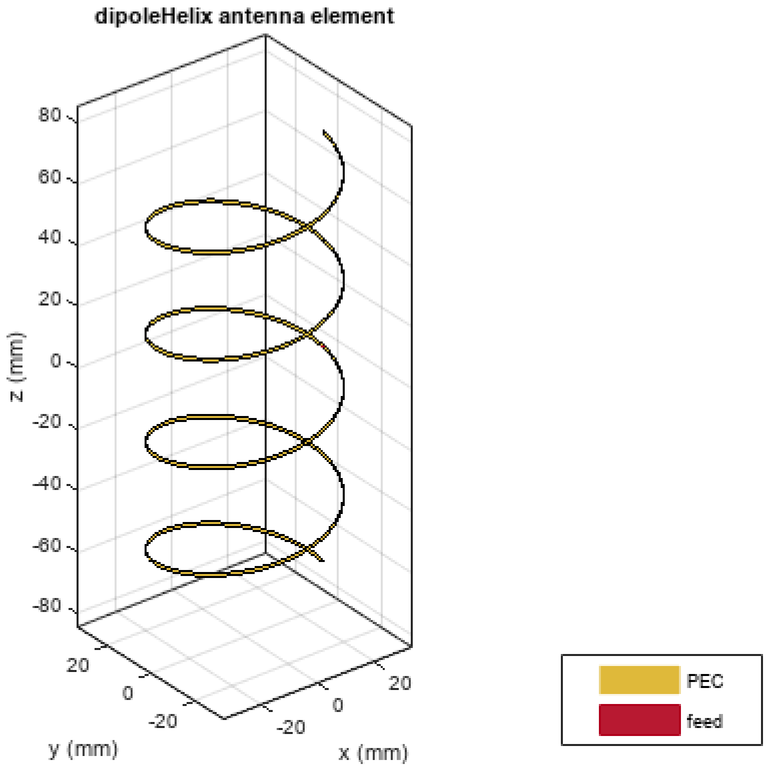
```
show(dh)
```



Radiation Pattern of Helical Dipole

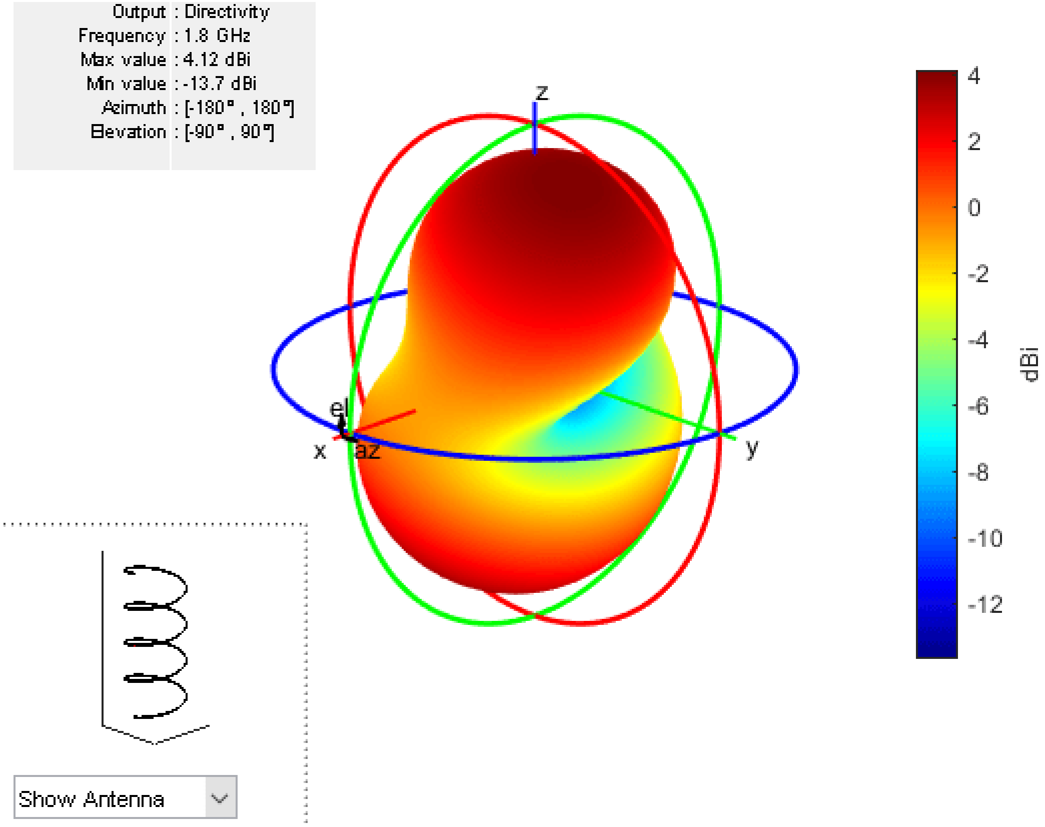
Create a four-turn helical dipole antenna with a turn radius of 28 mm and a strip width of 1.2 mm.

```
dh = dipoleHelix(Radius=28e-3, Width=1.2e-3, Turns=4);
show(dh)
```



Plot the radiation pattern of the helical dipole at 1.8 GHz.

```
pattern(dh, 1.8e9);
```



Output : Directivity
Frequency : 1.8 GHz
Max value : 4.12 dBi
Min value : -13.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Dipole Helix Antenna with Dielectric Substrate

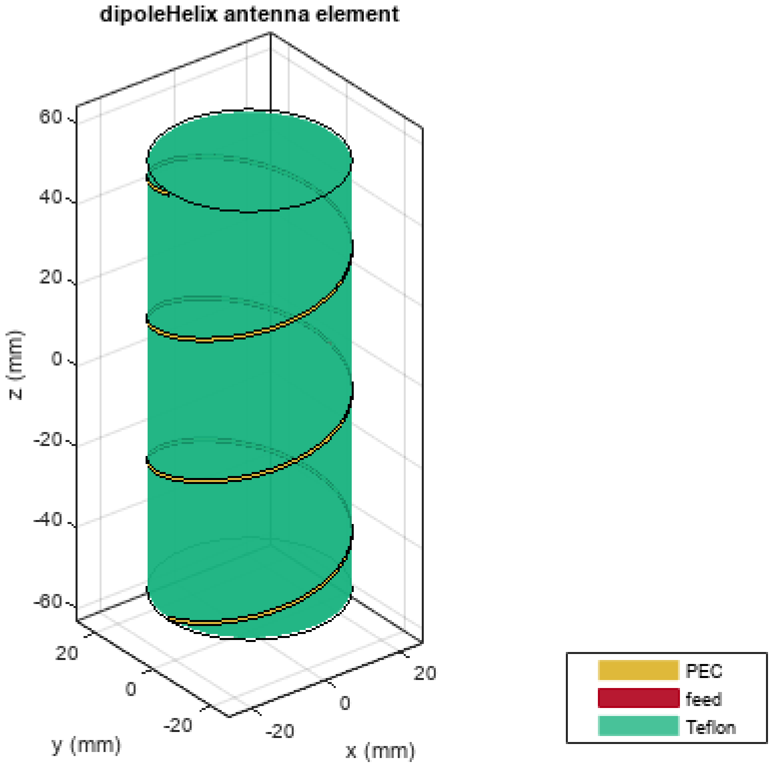
Create a custom dipole helix antenna with a Teflon dielectric substrate.

```
d = dielectric('Teflon');
dh = dipoleHelix(Radius=22e-3,Width=1e-3,Turns=3,Spacing=35e-3,FeedOffset=0,Substrate=d)
```

```
dh = 
  dipoleHelix with properties:

               Radius: 0.0220
                Width: 1.0000e-03
                Turns: 3
              Spacing: 0.0350
     WindingDirection: 'CCW'
           FeedOffset: 0
            Substrate: [1x1 dielectric]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

View the dipole helix antenna.

```
show(dh)
```



Version History

Introduced in R2016b

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*. 4th Ed. New York: McGraw-Hill, 2007.

See Also

`spiralArchimedean` | `helix` | `monopole` | `pifa` | `helixpitch2spacing` | `cylinder2strip`

Topics

“Rotate Antennas and Arrays”

helix

Create helix or conical helix antenna on ground plane

Description

Use the `helix` object to create a helix or conical helix antenna on a circular ground plane. The helix antenna is a common choice in satellite communication.

The width of the strip is related to the diameter of an equivalent cylinder by the equation

$$w = 2d = 4r$$

where:

- w is the width of the strip.
- d is the diameter of an equivalent cylinder.
- r is the radius of an equivalent cylinder.

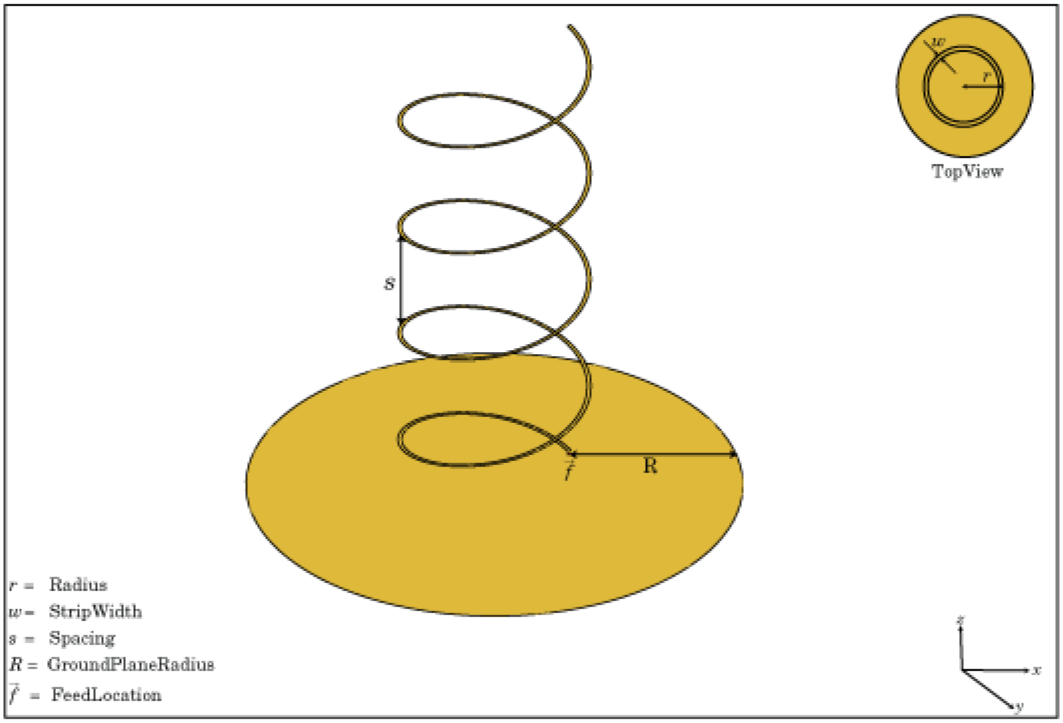
For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default helix antenna is end-fed. The circular ground plane is on the *xy*- plane. Commonly, helix antennas are used in axial mode. In this mode, the helix circumference is comparable to the operating wavelength and the helix has maximum directivity along its axis. In normal mode, the helix radius is small compared to the operating wavelength. In this mode, the helix radiates broadside, that is, in the plane perpendicular to its axis. The basic equation for the helix is

$$x = r\cos(\theta)$$
$$y = r\sin(\theta)$$
$$z = S\theta$$

where

- r is the radius of the helix.
- θ is the winding angle.
- S is the spacing between turns.

For a given pitch angle in degrees, use the `helixpitch2spacing` utility function to calculate the spacing between the turns in meters.



Note In an array of helix antennas, the circular ground plane of the helix is converted to rectangular ground plane.

Creation

Syntax

```
ant = helix
ant = helix(Name,Value)
```

Description

`ant = helix` creates a helix antenna operating in axial mode. The default antenna operates around 2 GHz.

`ant = helix(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = helix('Radius',28e-03)` creates a helix with turns of radius 28e-03 m.

Output Arguments

ant — Helix antenna
`helix` object

Helix antenna, returned as a `helix` object.

Properties

`Radius` — Radius of turns

`0.0220` (default) | positive scalar integer | two-element vector

Radius of the turns, specified as a positive scalar integer in meters or a two element vector with each element unit in meters. In the two-element vector, the first element specifies the bottom radius and the second element specifies the top radius of the conical helix antenna.

Example: `'Radius',[28e-03 30e-03]`

Example: `ant.Radius = [28e-03 30e-03]`

Data Types: `double`

`Width` — Strip width

1.0000e-03 (default) | scalar

Strip width, specified as a scalar in meters.

Note Strip width should be less than `'Radius'`/5 and greater than `'Radius'`/250. [4]

Example: `'Width',5`

Example: `ant.Width = 5`

Data Types: `double`

`Turns` — Number of turns of helix

3 (default) | scalar

Number of turns of the helix, specified as a scalar.

Example: `'Turns',2`

Example: `ant.Turns = 2`

Data Types: `double`

`Spacing` — Spacing between turns

0.0350 (default) | scalar

Spacing between turns, specified as a scalar in meters.

Example: `'Spacing',1.5`

Example: `ant.Spacing = 1.5`

Data Types: `double`

`WindingDirection` — Direction of helix turns (windings)

`'CW'` | `'CCW'`

Direction of helix turns (windings), specified as `'CW'` or `'CCW'`.

Example: `'WindingDirection'`,`CW`

Example: `ant.WindingDirection = CW`

Data Types: `char` | `string`

`GroundPlaneRadius` — Ground plane radius

0.0750 (default) | scalar in meters

Ground plane radius, specified as a scalar in meters. By default, the ground plane is on the X-Y plane and is symmetrical about the origin.

Example: `'GroundPlaneRadius',2.05`

Example: `ant.GroundPlaneRadius = 2.05`

Data Types: `double`

`FeedStubHeight` — Feeding stub height from ground

1.0000e-03 (default) | scalar

Feeding stub height from ground, specified as a scalar in meters.

Example: `'FeedStubHeight',2.000e-03`

Example: `ant.FeedStubHeight = 2.000e-03`

Note The default value is chosen to allow backward compatibility.

Data Types: `double`

`Substrate` — Type of dielectric material

`'Air'` (default) | `dielectric` object

Type of dielectric material used as the substrate, specified as a dielectric object. You can specify only one dielectric layer in the `helix` object. When using the `Substrate` property, specify the same radius for all the turns. When using a dielectric material other than air, the number of turns in the helix should be greater than 1. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Example: `d = dielectric('Teflon'); hx = helix('Substrate',d)`

Example: `d = dielectric('Teflon'); hx.Substrate = d`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface

current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

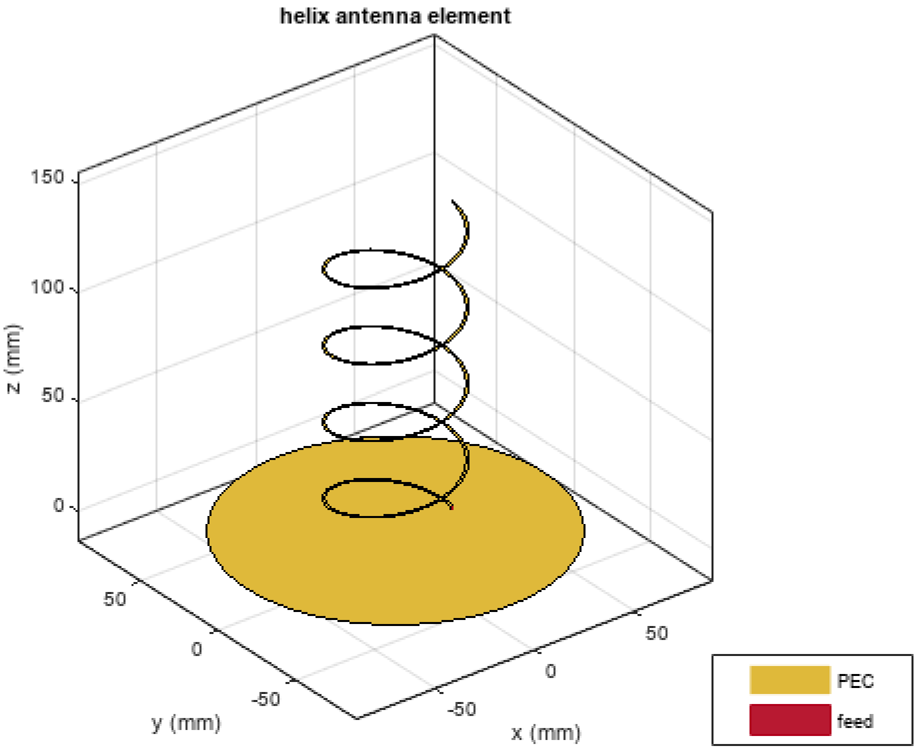
Create and View Helix Antenna

Create and view a helix antenna that has a 28 mm turn radius, 1.2 mm strip width, and 4 turns.

```
hx = helix('Radius',28e-3,'Width',1.2e-3,'Turns',4)
```

```
hx = 
  helix with properties:

               Radius: 0.0280
                Width: 0.0012
                Turns: 4
              Spacing: 0.0350
     WindingDirection: 'CCW'
       FeedStubHeight: 1.0000e-03
    GroundPlaneRadius: 0.0750
            Substrate: [1x1 dielectric]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

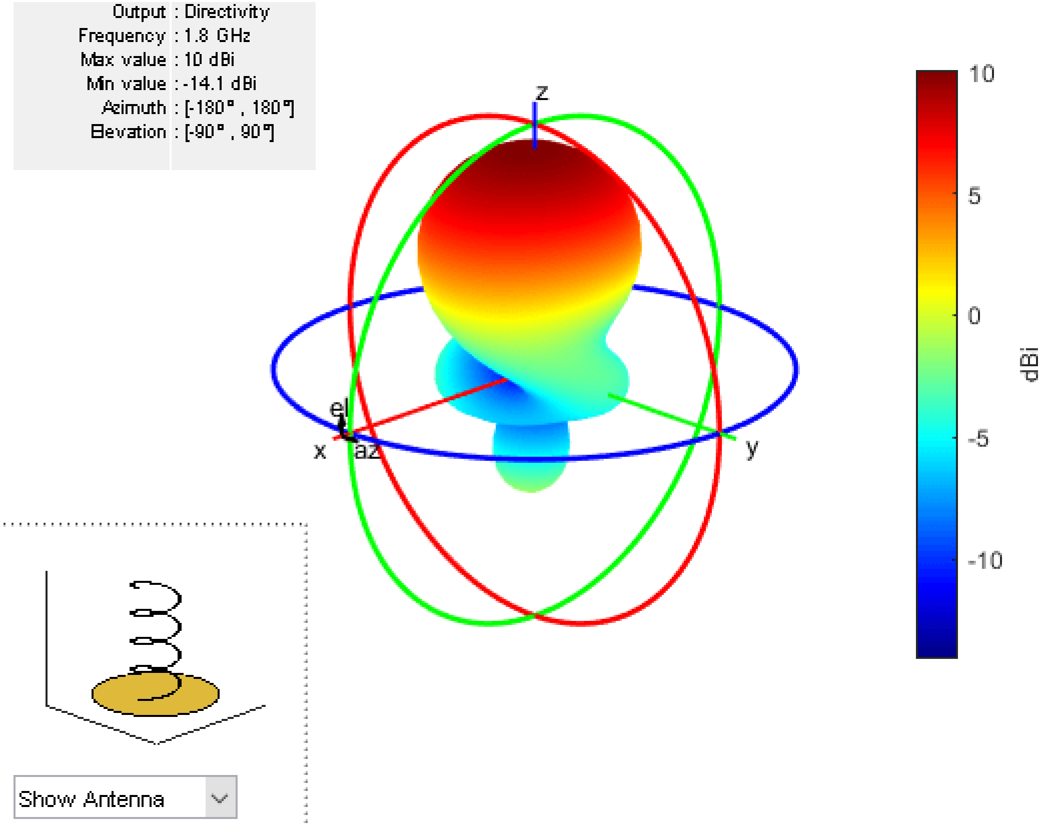
```
show(hx)
```



Radiation Pattern of Helix Antenna

Plot the radiation pattern of a helix antenna.

```
hx = helix('Radius',28e-3,'Width',1.2e-3,'Turns',4);
pattern(hx,1.8e9);
```



Calculate Spacing of Helix Antenna with Varying Radius

Calculate the spacing of a helix that has a pitch of 12 degrees and a radius that varies from 20 mm to 22 mm in steps of 0.5 mm.

```
s = helixpitch2spacing(12,20e-3:0.5e-3:22e-3)
```

```
s = 1×5

    0.0267    0.0274    0.0280    0.0287    0.0294
```

Radiation Pattern of Helix Antenna

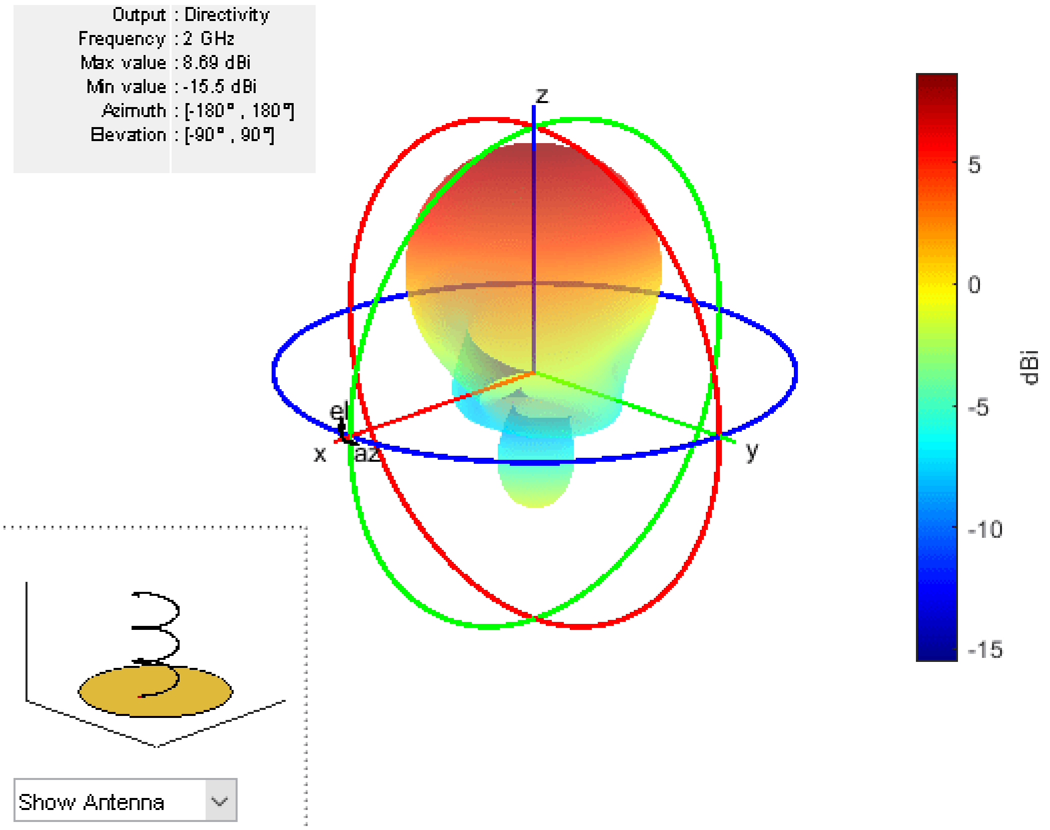
Plot the radiation pattern of a helix antenna with transparency specified as 0.5.

```
p = PatternPlotOptions
```

```
p = 
  PatternPlotOptions with properties:

      Transparency: 1
```

```
          SizeRatio: 0.9000
     MagnitudeScale: []
      AntennaOffset: [0 0 0]
```

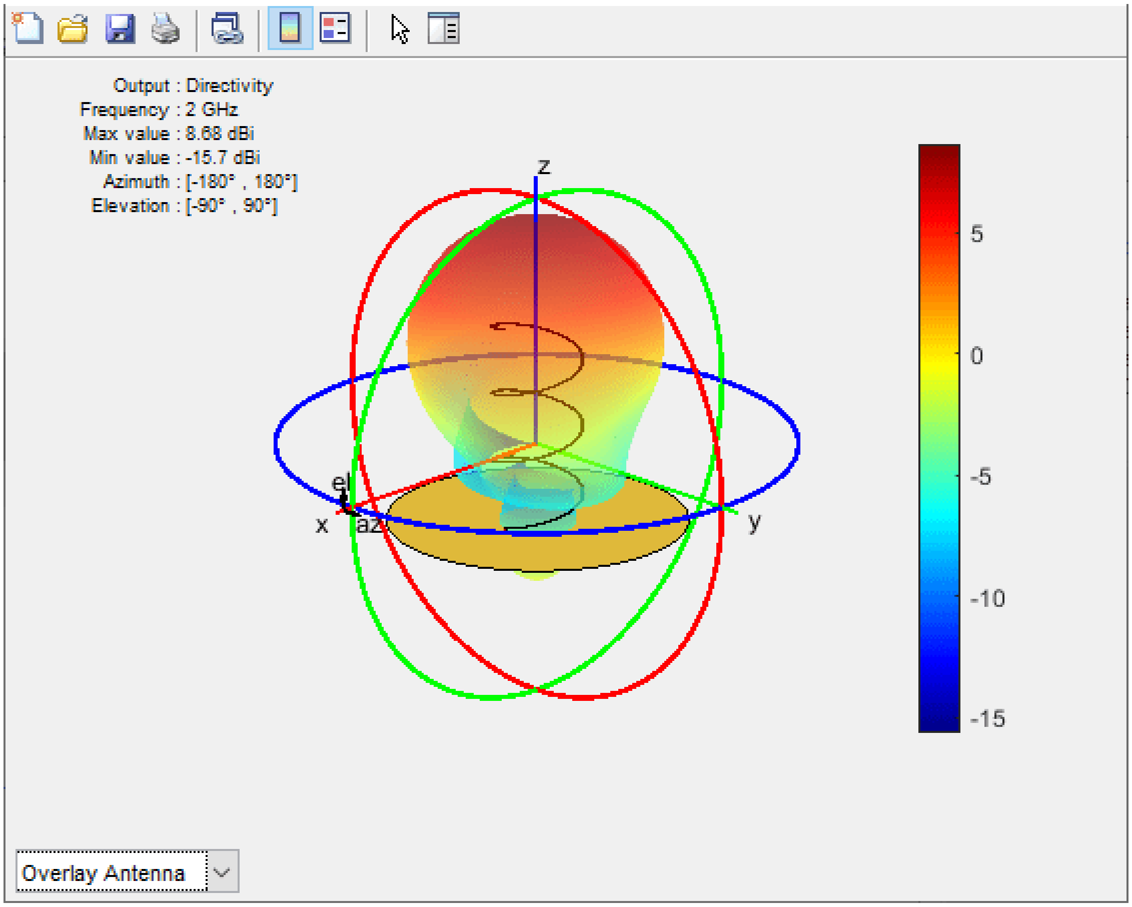
```
p.Transparency = 0.5;
ant = helix;
pattern(ant,2e9,'patternOptions',p)
```



```
 Output : Directivity
 Frequency : 2 GHz
 Max value : 8.69 dBi
 Min value : -15.5 dBi
 Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

To understand the effect of Transparency, chose `Overlay Antenna` in the radiation pattern plot.

This option overlays the helix antenna on the radiation pattern.



Output : Directivity
Frequency : 2 GHz
Max value : 8.68 dBi
Min value : -15.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Helix Antenna with Dielectric Substrate

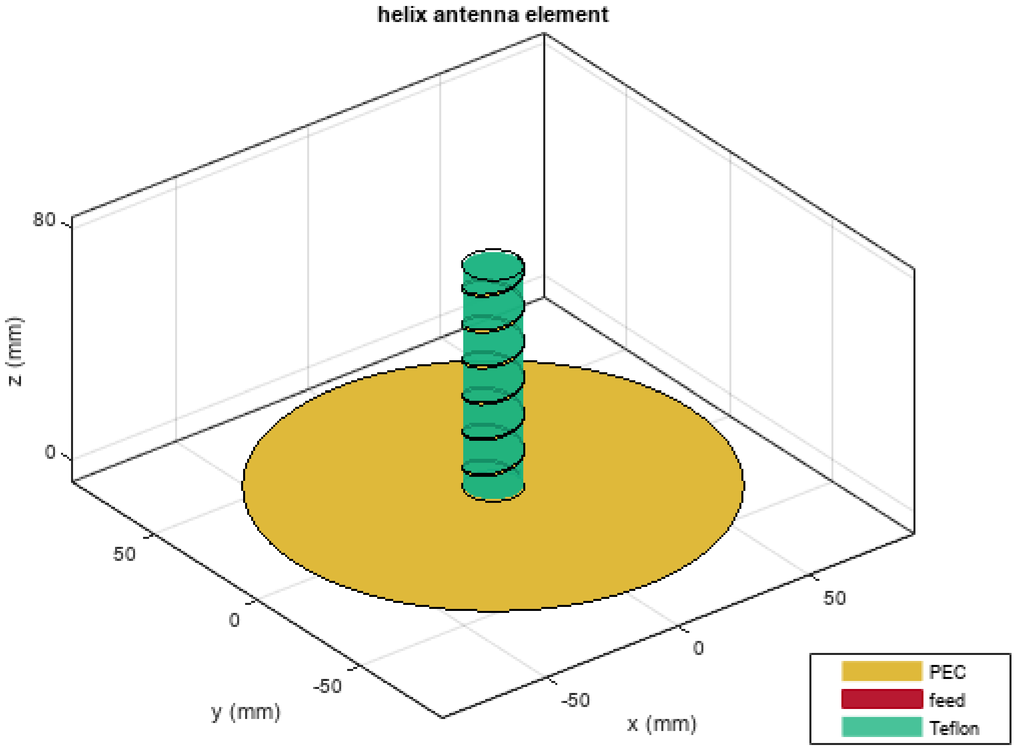
Create a custom helix antenna with a Teflon dielectric substrate.

```
d = dielectric('Teflon');
hx = helix('Width',0.815e-3,'Turns',6,'Radius',9.3e-3,'Spacing',12.4e-3,'Substrate',d)
```

```
hx = 
  helix with properties:

               Radius: 0.0093
                Width: 8.1500e-04
                Turns: 6
              Spacing: 0.0124
     WindingDirection: 'CCW'
       FeedStubHeight: 1.0000e-03
    GroundPlaneRadius: 0.0750
            Substrate: [1x1 dielectric]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

View the helix antenna.

```
show(hx)
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

[3] Zhang, Yan, Q. Ding, J. Chen, S. Lu, Z. Zhu and L. L. Cheng. “A Parametric Study of Helix Antenna for S-Band Satellite Communications.” *9th International Symposium on Antenna Propagation and EM Theory (ISAPE)*. 2010, pp. 193–196.

[4] Djordjevic, A.R., Zajic, A.G., Ilic, M. M., Stuber, G.L. “Optimization of Helical antennas (Antenna Designer's Notebook)” *IEEE Antennas and Propagation Magazine*. December, 2006, pp. 107, pp.115.

[5] B. Young, K. A. O'Connor and R. D. Curry, “Reducing the size of helical antennas by means of dielectric loading,” *2011 IEEE Pulsed Power Conference*, 2011, pp. 575-579, doi: 10.1109/PPC.2011.6191490

See Also

`spiralArchimedean` | `monopole` | `pifa` | `helixpitch2spacing` | `cylinder2strip` | `helixMultifilar`

Topics

"Rotate Antennas and Arrays"

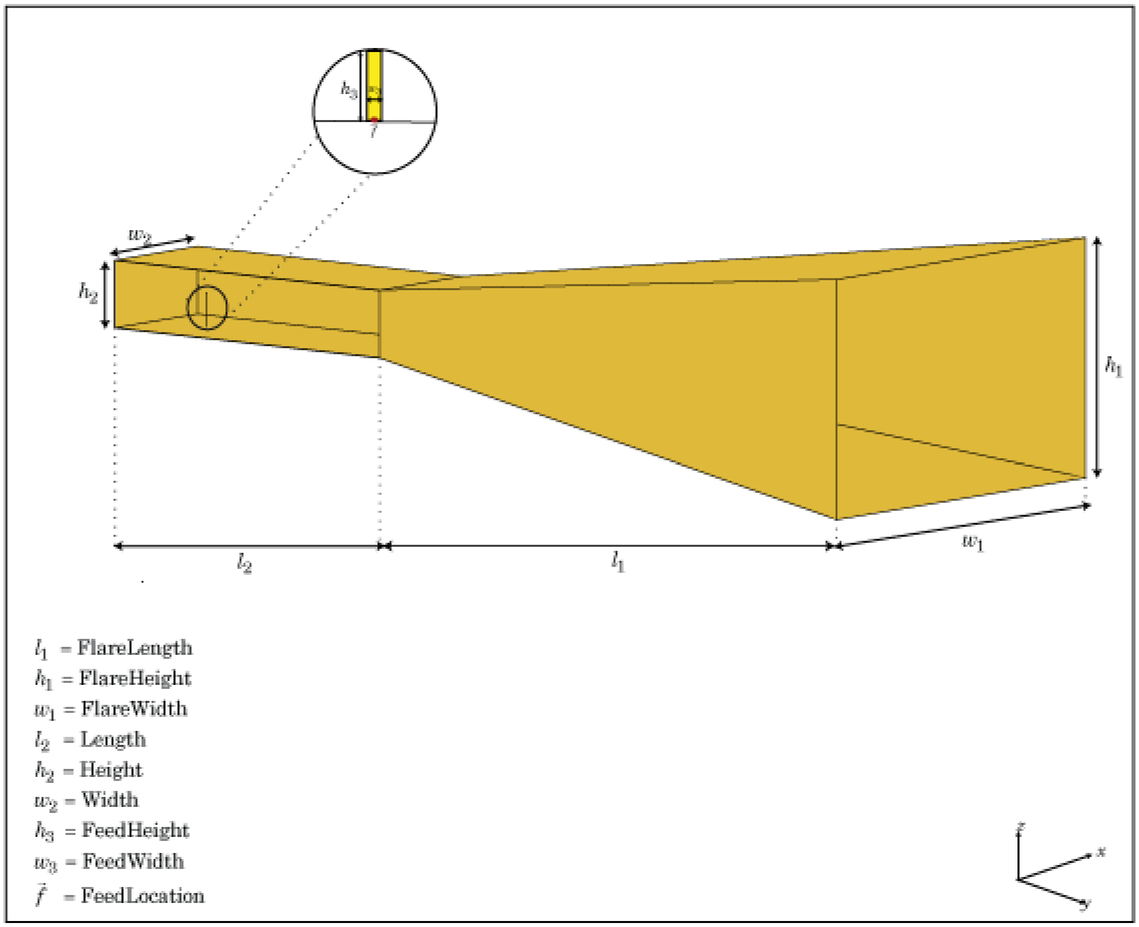
horn

Create horn antenna

Description

The `horn` object is a pyramidal horn antenna with a standard-gain, 15 dBi. The default horn antenna operates in the X-Ku band, which ranges from 10 GHz to 15 GHz. By default, the horn antenna feed is a WR-75 rectangular waveguide with an operating frequency at 7.87 GHz.

For a given flare angles of the horn and dimensions of the waveguide, use the `hornangle2size` utility function to calculate the equivalent flare width and flare height of the horn.



Creation

Syntax

```
hr = horn
hr = horn(Name,Value)
```

Description

`hr = horn` creates a standard-gain pyramidal horn antenna.

`hr = horn(Name,Value)` creates a horn antenna with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`FlareLength` — Flare length of horn
`0.1020` (default) | scalar

Flare length of horn, specified as a scalar in meters.

Example: `'FlareLength',0.35`

Data Types: `double`

`FlareWidth` — Flare width of horn
`0.0571` (default) | scalar

Flare width of horn, specified as a scalar in meters.

Example: `'FlareWidth',0.2`

Data Types: `double`

`FlareHeight` — Flare height of horn
`0.0338` (default) | scalar

Flare height of horn, specified as a scalar in meters.

Example: `'FlareHeight',0.15`

Data Types: `double`

`Length` — Rectangular waveguide length
`0.0500` (default) | scalar

Rectangular waveguide length, specified as a scalar in meters.

Example: `'Length',0.09`

Data Types: `double`

`Width` — Rectangular waveguide width
`0.0190` (default) | scalar

Rectangular waveguide width, specified as a scalar in meters.

Example: `'Width',0.05`

Data Types: `double`

Height — Rectangular waveguide height

0.0095 (default) | scalar

Rectangular waveguide height, specified as a scalar in meters.

Example: 'Height',0.0200

Data Types: double

FeedHeight — Height of feed

0.0048 (default) | scalar

Height of feed, specified as a scalar in meters.

Example: 'FeedHeight',0.0050

Data Types: double

FeedWidth — Width of feed

1.0000e-04 (default) | scalar

Width of feed, specified as a scalar in meters.

Example: 'FeedWidth',5e-05

Data Types: double

FeedOffset — Signed offset of feedpoint from center of ground plane

[–0.0155 0] (default) | two-element vector

Signed offset from center of ground plane, specified as a two-element vector in meters.

Example: 'FeedOffset',[–0.0070 0.01]

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: hr.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer

pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

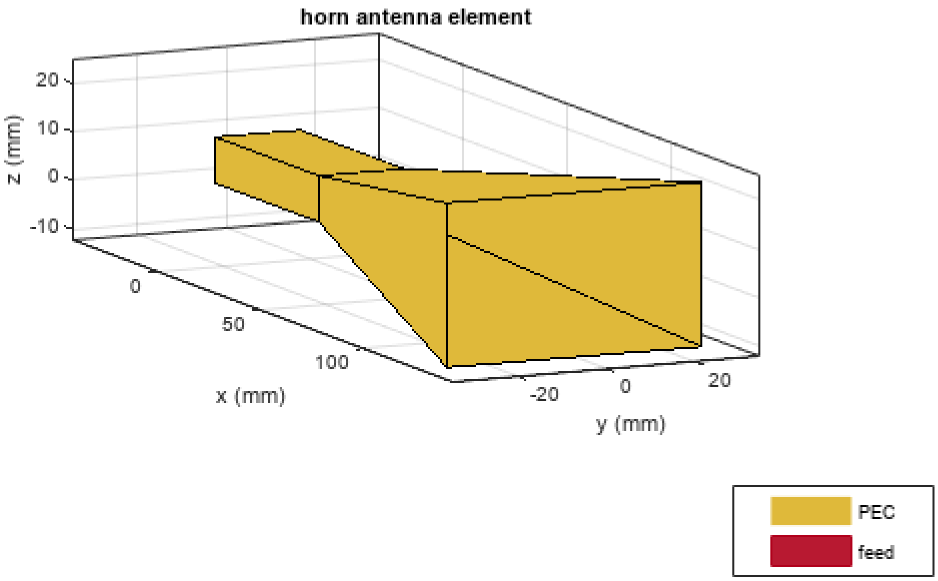
Default Horn Antenna

Create and view a default horn antenna.

```
h = horn
```

```
h = 
  horn with properties:

    FlareLength: 0.1020
     FlareWidth: 0.0571
    FlareHeight: 0.0338
         Length: 0.0500
          Width: 0.0190
         Height: 0.0095
      FeedWidth: 1.0000e-04
     FeedHeight: 0.0048
     FeedOffset: [-0.0155 0]
      Conductor: [1x1 metal]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

```
show(h)
```



Version History

Introduced in R2016a

References

[1] Balanis, Constantine A.*Antenna Theory. Analysis and Design*. 3rd Ed. New York: John Wiley and Sons, 2005.

See Also

`waveguide` | `hornangle2size`

Topics

"Rotate Antennas and Arrays"

invertedF

Create inverted-F antenna over rectangular ground plane

Description

The `invertedF` object is an inverted-F antenna mounted over a rectangular ground plane.

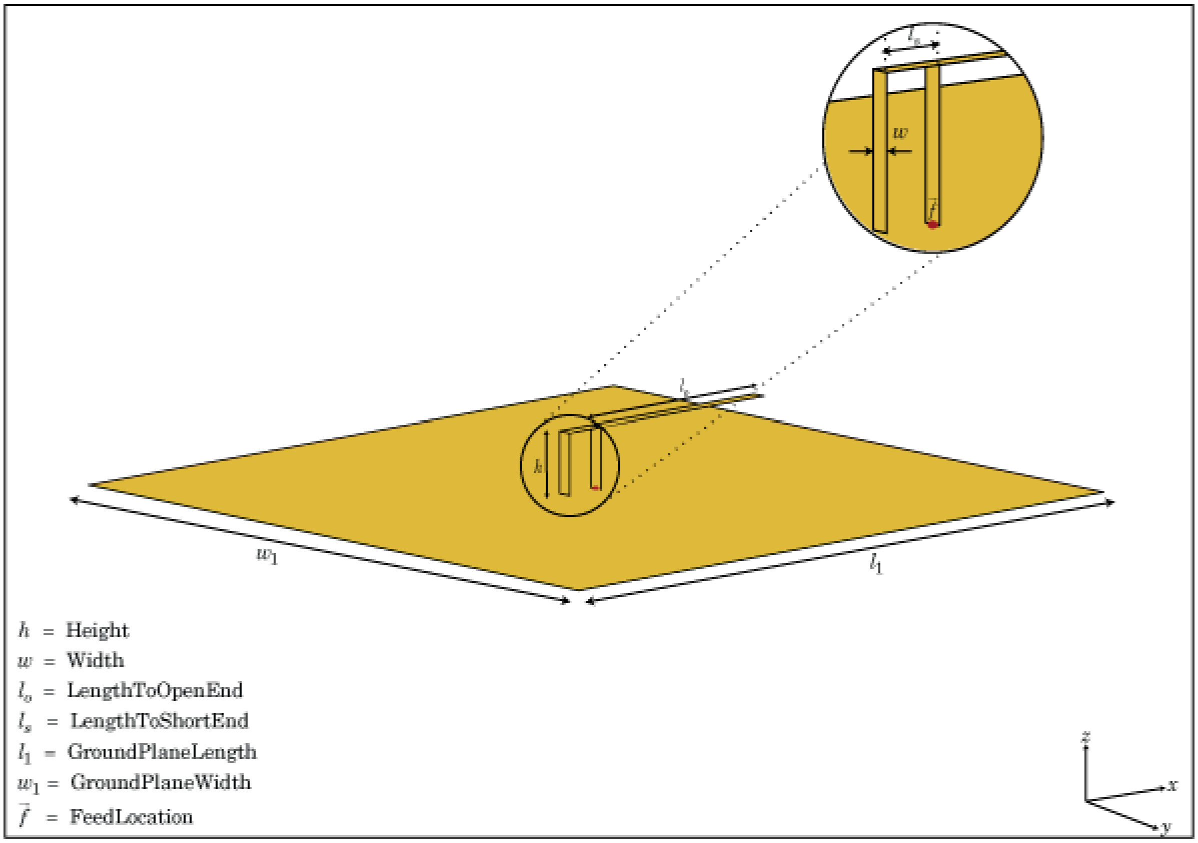
The width of the metal strip is related to the diameter of an equivalent cylinder by the equation

$$w = 2d = 4r$$

where:

- d is the diameter of equivalent cylinder
- r is the radius of equivalent cylinder

For a given cylinder radius, use the utility function `cylinder2strip` to calculate the equivalent width. The default inverted-F antenna is center-fed. The feed point coincides with the origin. The origin is located on the *xy*- plane.



Creation

Syntax

```
f = invertedF
f = invertedF(Name,Value)
```

Description

`f = invertedF` creates an inverted-F antenna mounted over a rectangular ground plane. By default, the dimensions are chosen for an operating frequency of 1.7 GHz.

`f = invertedF(Name,Value)` creates an inverted-F antenna, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Height` — Vertical element height along *z*-axis
0.0140 (default) | scalar

Vertical element height along *z*-axis, specified a scalar in meters.

Example: `'Height',3`

Data Types: `double`

`Width` — Strip width
0.0020 (default) | scalar

Strip width, specified as a scalar in meters.

Note Strip width should be less than `'Height'`/4 and greater than `'Height'`/1001. [2]

Example: `'Width',0.05`

Data Types: `double`

`LengthToOpenEnd` — Stub length from feed to open end
0.0310 (default) | scalar

Stub length from feed to open end, specified as a scalar in meters.

Example: `'LengthToOpenEnd',0.05`

`LengthToShortEnd` — Stub length from feed to shorting end
0.0060 (default) | scalar

Stub length from feed to shorting end, specified as a scalar in meters.

Example: `'LengthToShortEnd',0.0050`

GroundPlaneLength — Ground plane length along x-axis

0.1000 (default) | scalar

Ground plane length along *x*-axis, specified as a scalar in meters. Setting `'GroundPlaneLength'` to Inf, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',4`

Data Types: `double`

GroundPlaneWidth — Ground plane width along y-axis

0.1000 (default) | scalar

Ground plane width along *y*-axis, specified as a scalar in meters. Setting `'GroundPlaneWidth'` to Inf, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',2.5`

Data Types: `double`

FeedOffset — Signed distance from center along length and width of ground plane

`[0 0]` (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector.

Example: `'FeedOffset',[2 1]`

Data Types: `double`

Conductor — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `f.Load = lumpedElement('Impedance',75)`

Tilt — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array

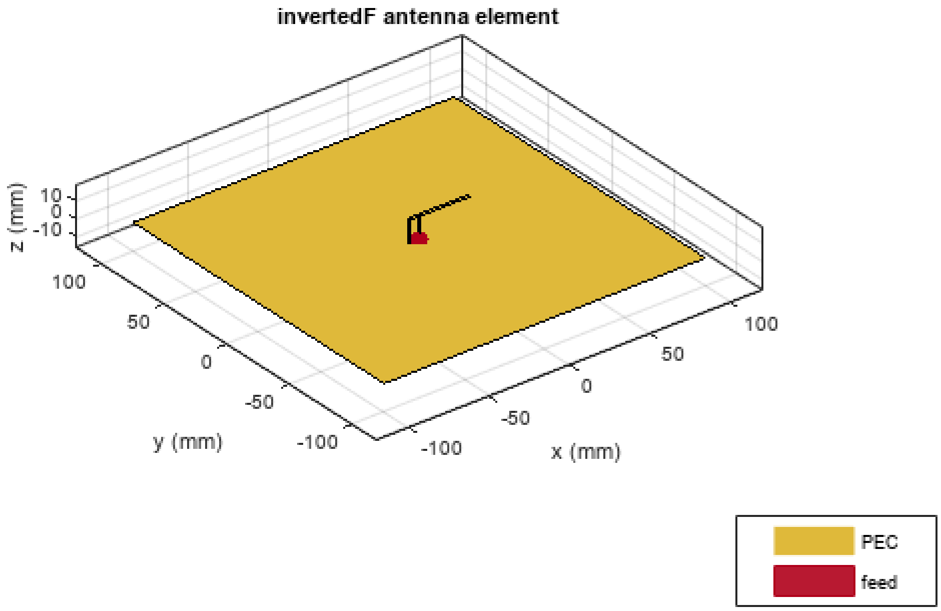
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Inverted-F Antenna

Create and view an inverted-F antenna with 14 mm height over a ground plane of dimensions 200 mm-by-200 mm.

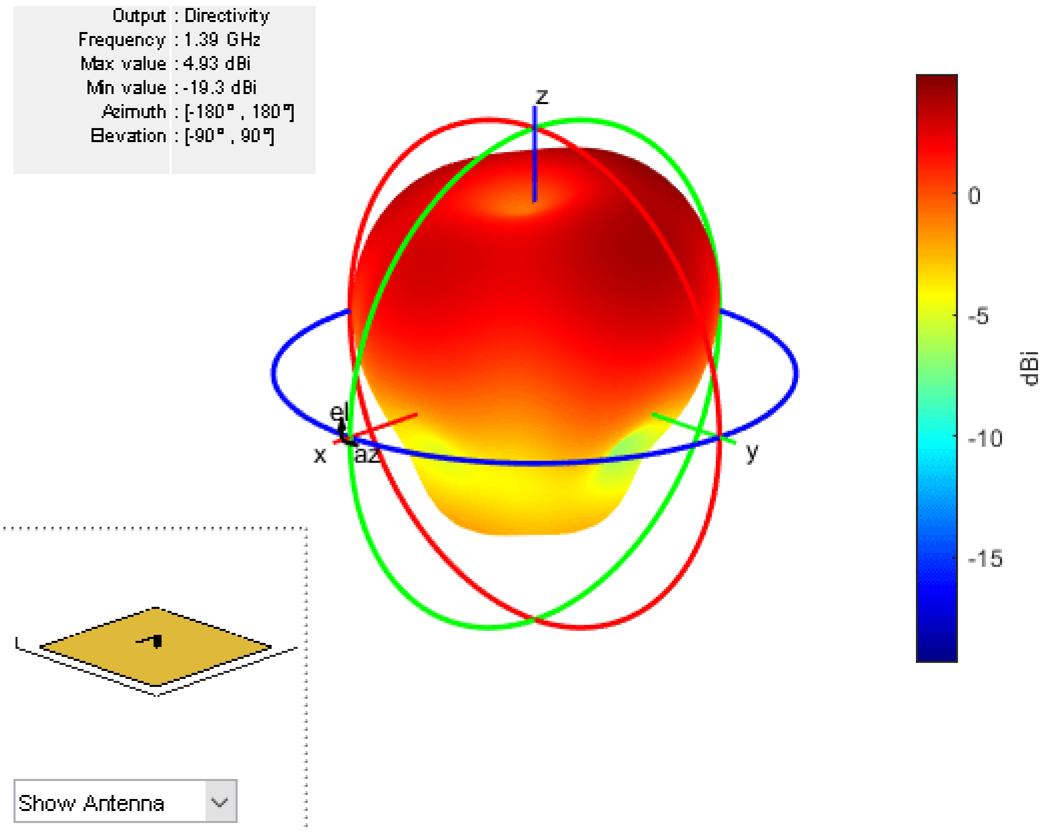
```
f = invertedF('Height',14e-3, 'GroundPlaneLength',200e-3,          ...
                   'GroundPlaneWidth',200e-3);
show(f)
```



Plot Radiation Pattern of Inverted-F

This example shows you how to plot the radiation pattern of an inverted-F antenna for a frequency of 1.3 GHz.

```
f = invertedF('Height',14e-3, 'GroundPlaneLength', 200e-3,          ...
                    'GroundPlaneWidth', 200e-3);
pattern(f,1.39e9)
```



```
   Output : Directivity
Frequency : 1.39 GHz
Max value : 4.93 dBi
Min value : -19.3 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

See Also

`invertedL` | `pifa` | `patchMicrostrip` | `cylinder2strip`

Topics

"Rotate Antennas and Arrays"

invertedL

Create inverted-L antenna over rectangular ground plane

Description

The `invertedL` object is an inverted-L antenna mounted over a rectangular ground plane.

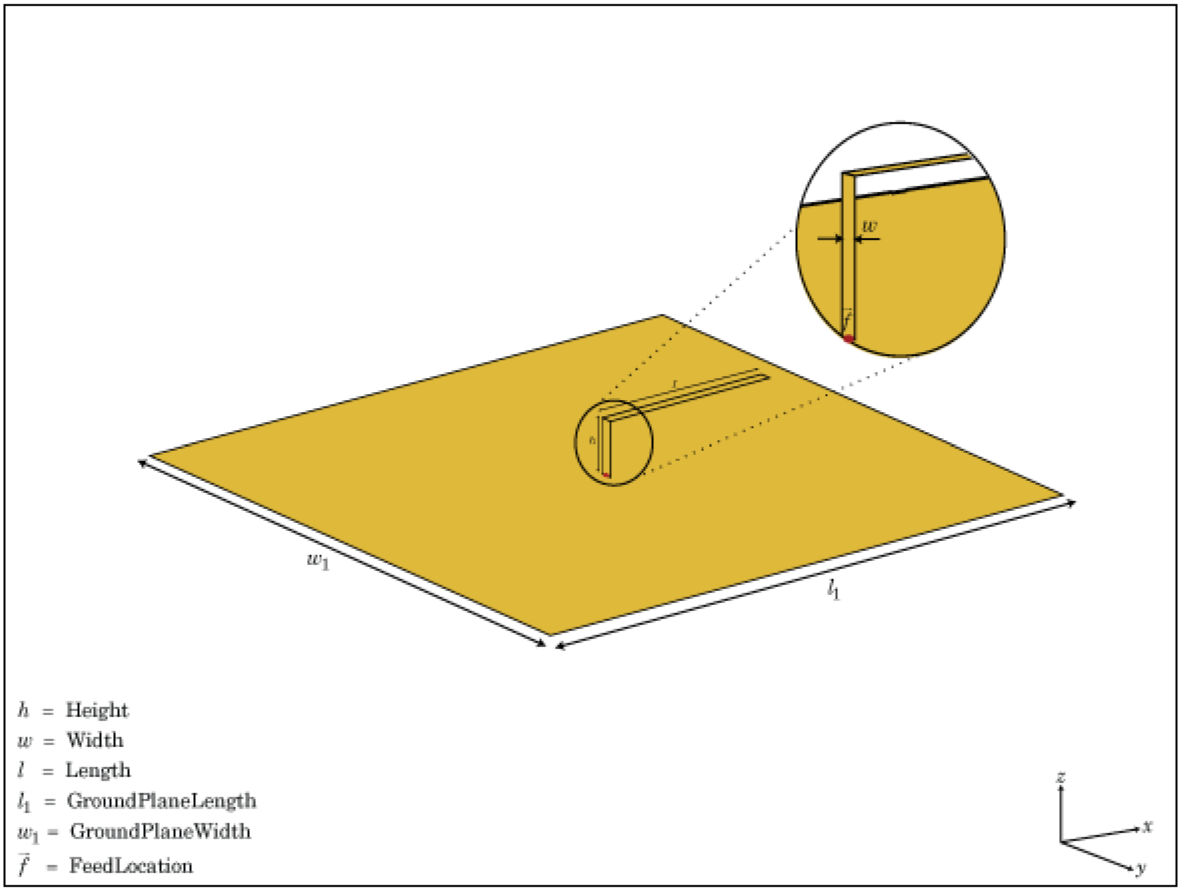
The width of the metal strip is related to the diameter of an equivalent cylinder by the equation

$$w = 2d = 4r$$

where:

- d = diameter of equivalent cylinder
- a = radius of equivalent cylinder

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default inverted-L antenna is center-fed. The feed point coincides with the origin. The origin is located on the *xy*- plane.



Creation

Syntax

```
l = invertedL
l = invertedL(Name,Value)
```

Description

`l = invertedL` creates an inverted-L antenna mounted over a rectangular ground plane. By default, the dimensions are chosen for an operating frequency of 1.7 GHz.

`l = invertedL(Name,Value)` creates an inverted-L antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Height` — Height of inverted element along *z*-axis
0.0140 (default) | scalar

Height of inverted element along *z*-axis, specified a scalar in meters.

Example: `'Height',3`

Data Types: `double`

`Width` — Strip width
0.0020 (default) | scalar

Strip width, specified as a scalar in meters.

Note Strip width should be less than `'Height'`/4 and greater than `'Height'`/1001. [2]

Example: `'Width',0.05`

Data Types: `double`

`Length` — Stub length along *x*-axis
0.0310 (default) | scalar

Stub length along *x*-axis, specified as a scalar in meters.

Example: `'Length',0.01`

`GroundPlaneLength` — Ground plane length along *x*-axis
0.1000 (default) | scalar

Ground plane length along *x*-axis, specified a scalar in meters. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',4`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

0.1000 (default) | scalar

Ground plane width along *y*-axis, specified as a scalar in meters. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',2.5`

Data Types: `double`

`FeedOffset` — Signed distance from center along length and width of ground plane

`[0 0]` (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector.

Example: `'FeedOffset',[2 1]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `l.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

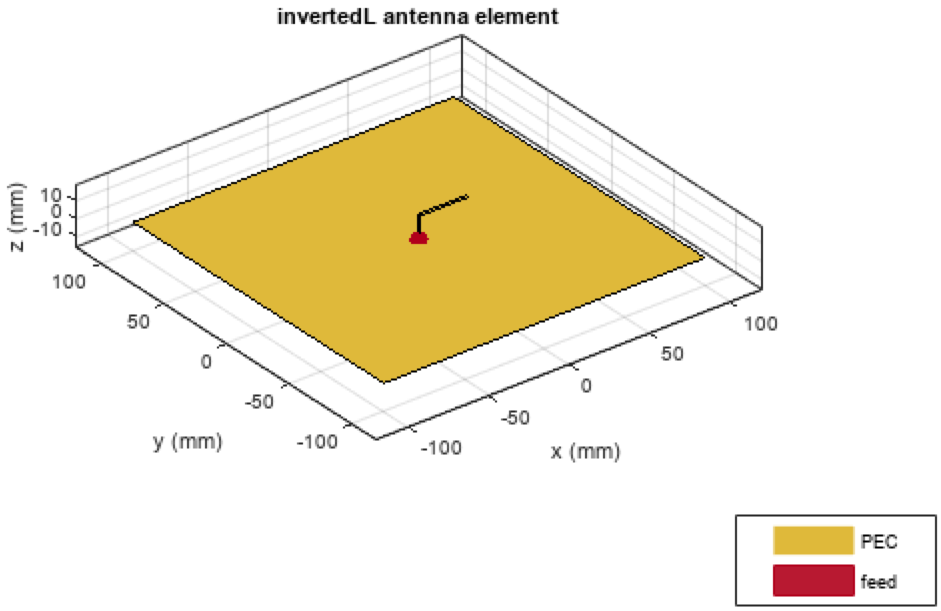
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Inverted-L Antenna

Create and view an inverted-L antenna that has 30mm length over a ground plane of dimensions 200mmx200mm.

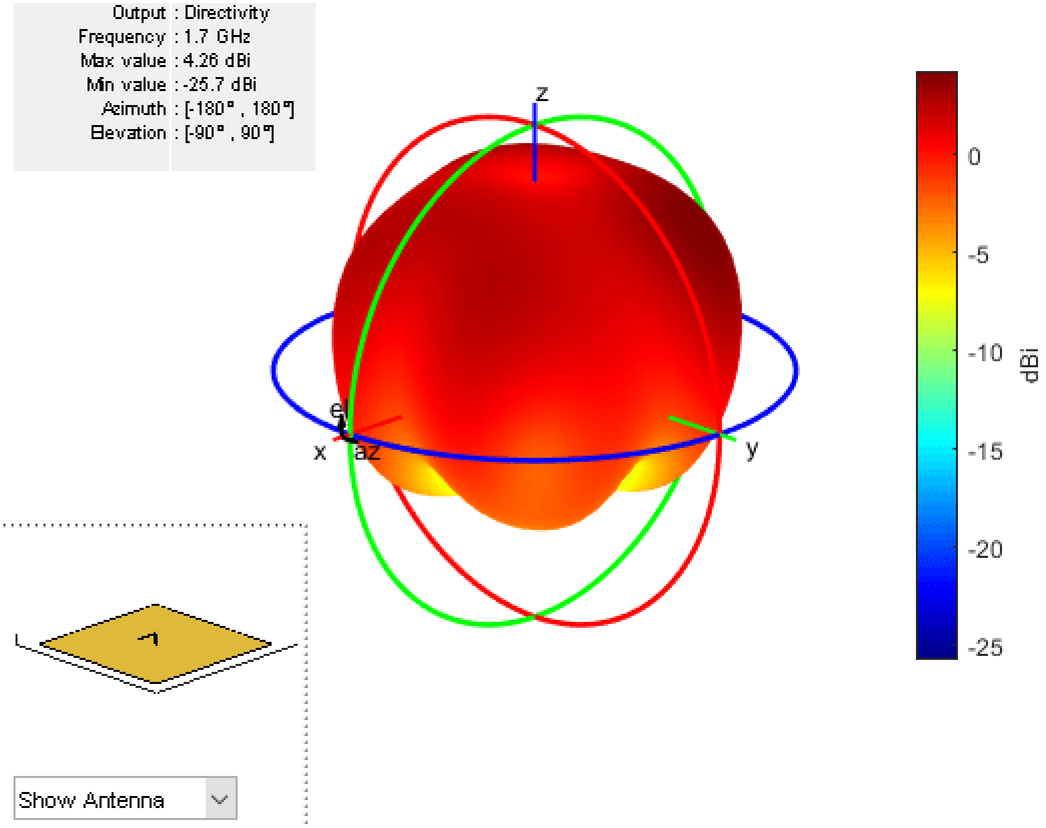
```
il = invertedL('Length',30e-3, 'GroundPlaneLength',200e-3,...
                  'GroundPlaneWidth',200e-3);
show(il)
```



Radiation Pattern of Inverted-L Antenna

Plot the radiation pattern of an inverted-L at a frequency of 1.7 GHz.

```
iL = invertedL('Length',30e-3, 'GroundPlaneLength',200e-3,...
                  'GroundPlaneWidth',200e-3);
pattern(iL,1.7e9)
```



```
   Output : Directivity
Frequency : 1.7 GHz
Max value : 4.26 dBi
Min value : -25.7 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

See Also

`invertedF` | `pifa` | `patchMicrostrip` | `cylinder2strip`

Topics

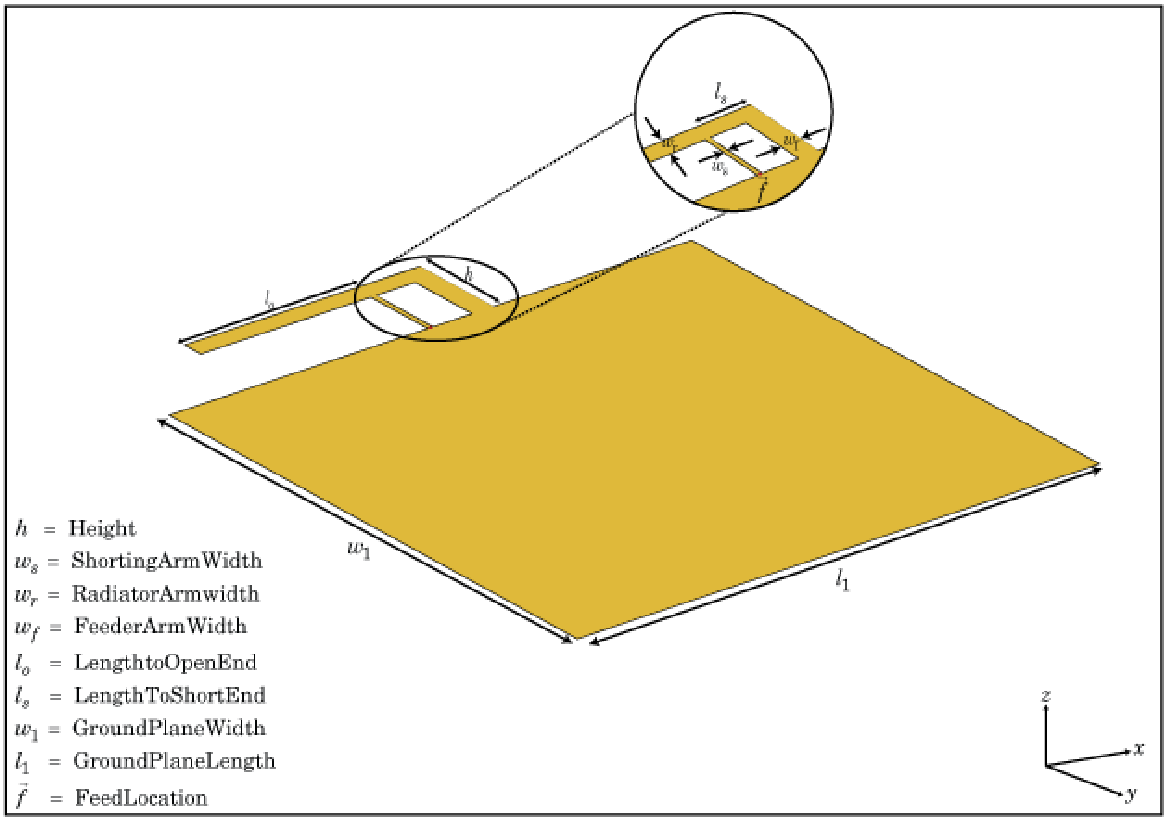
"Rotate Antennas and Arrays"

invertedFcoplanar

Create inverted-F antenna in same plane as rectangular ground plane

Description

The `invertedFcoplanar` object is a coplanar inverted-F antenna with a rectangular ground plane. By default, the dimensions are chosen for an operating frequency of 1.7 GHz. Coplanar inverted-F antennas are used in RFID tags and Internet of Things (IoT) applications. This antenna is an altered version of the inverted-F antenna, providing a low-profile antenna with more design parameters and a wider bandwidth.



Creation

Syntax

```
fco = invertedFcoplanar
fco = invertedFcoplanar(Name,Value)
```

Description

fco = invertedFcoplanar creates a coplanar inverted-F antenna with the rectangular ground plane. By default, the antenna dimensions are for an operating frequency of 1.7 GHz.

fco = invertedFcoplanar(Name,Value) creates a coplanar inverted-F antenna, with additional properties specified by one or more name-value pair arguments. Name is the property name and Value is the corresponding value. You can specify several name-value pair arguments in any order as Name1, Value1, ..., NameN, ValueN. Properties not specified retain their default values.

Properties

RadiatorArmWidth — Width of radiating arm

0.0040 (default) | scalar

Width of radiating arm, specified as the comma-separated pair consisting of 'RadiatorArmWidth'and a scalar in meters.

Example: 'RadiatorArmWidth',0.05

Data Types: double

FeederArmWidth — Width of feeding arm

1.0000e-03 (default) | scalar

Width of feeding arm, specified as a scalar in meters.

Example: 'FeederArmWidth',0.05

Data Types: double

ShortingArmWidth — Width of shorting arm

0.0040 (default) | scalar

Width of shorting arm, specified as a scalar in meters.

Example: 'ShortingArmWidth',1

Data Types: double

Height — Height of antenna

0.0100 (default) | scalar

Height of antenna from ground plane, specified as a scalar in meters.

Example: 'Height',0.0800

Data Types: double

LengthToOpenEnd — Length of stub from feed to open end

0.0350 (default) | scalar

Length of the stub from feed to the open-end, specified as a scalar in meters.

Example: 'LengthToOpenEnd',0.050

Data Types: double

`LengthToShortEnd` — Length of stub from feed to shorting end

`0.0100` (default) | scalar

Length of the stub from feed to the shorting end, specified as a scalar in meters.

Example: `'LengthToShortEnd',0.035`

Data Types: `double`

`GroundPlaneLength` — Length of ground plane

`0.0800` (default) | scalar

Length of the ground plane, specified as a scalar in meters.

Example: `'GroundPlaneLength',0.035`

Data Types: `double`

`GroundPlaneWidth` — Width of ground plane

`0.0700` (default) | scalar

Width of the ground plane, specified as a scalar in meters.

Example: `'GroundPlaneWidth',0.035`

Data Types: `double`

`FeedOffset` — Signed distance from center of ground plane

`0` (default) | scalar

Signed distance from center of ground plane, specified as a scalar in meters.

Example: `'FeedOffset',0.06`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `fco.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Analysis Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer

pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
show	Display antenna, array structures or shapes
vswr	Voltage standing wave ratio of antenna

Examples

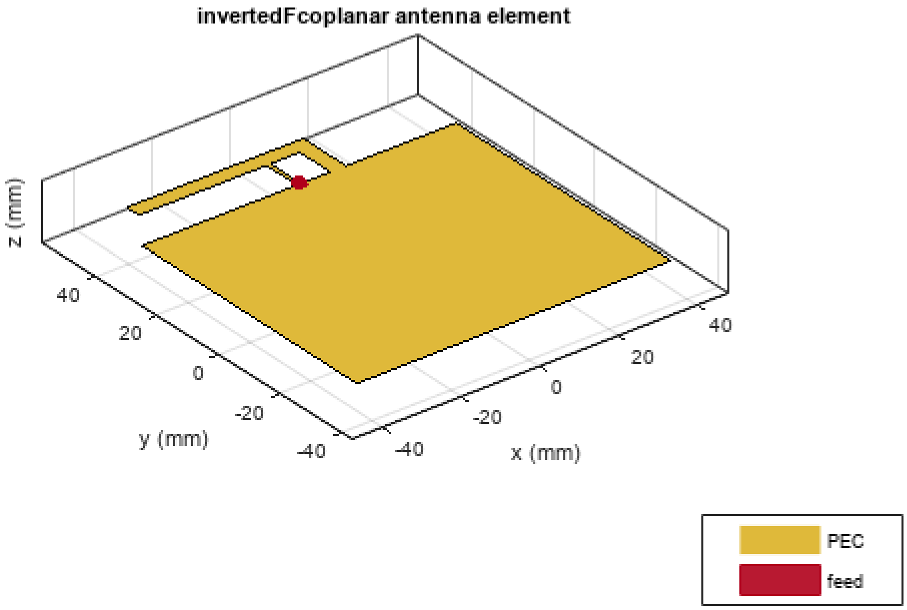
Coplanar Inverted-F Antenna

Create a default coplanar inverted-F antenna and view it.

```
fco = invertedFcoplanar
```

```
fco = 
  invertedFcoplanar with properties:

     RadiatorArmWidth: 0.0040
       FeederArmWidth: 1.0000e-03
     ShortingArmWidth: 0.0040
      LengthToOpenEnd: 0.0350
     LengthToShortEnd: 0.0100
               Height: 0.0100
    GroundPlaneLength: 0.0800
     GroundPlaneWidth: 0.0700
           FeedOffset: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(fco)
```



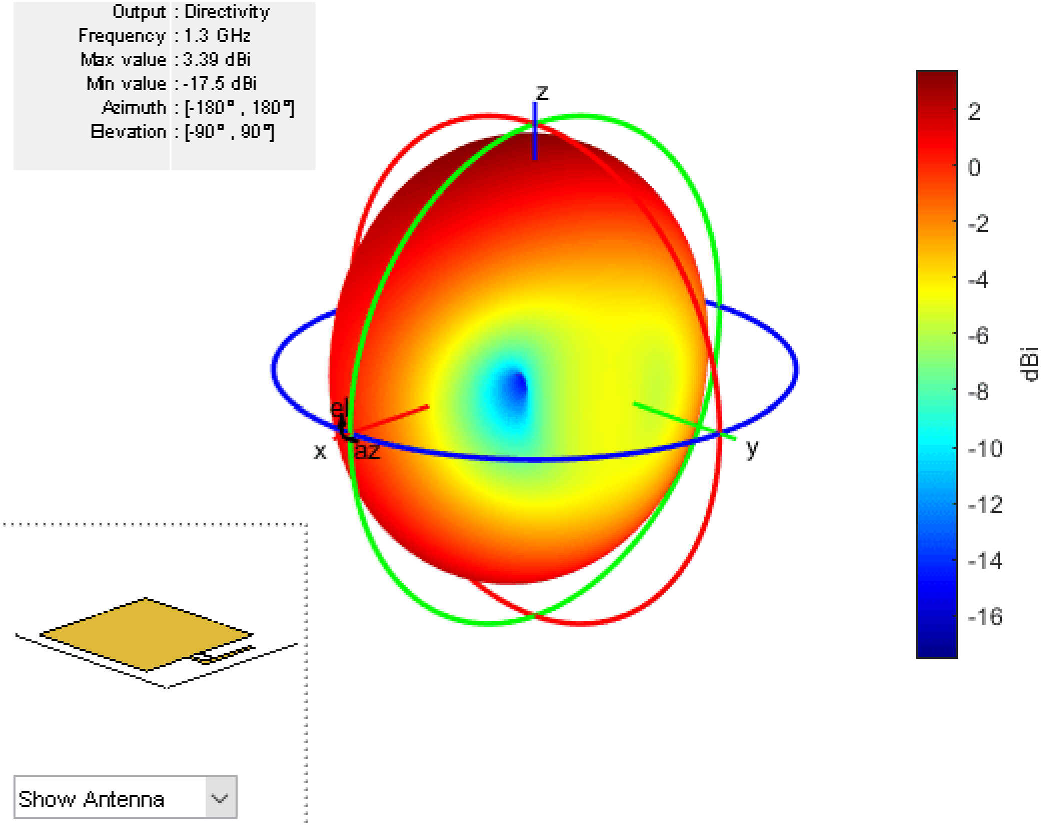
Radiation Pattern of Coplanar Inverted-F Antenna

Create a coplanar inverted-F antenna of height 0.014 m, ground plane length 0.1 m, and ground plane width 0.1 m.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                 'GroundPlaneWidth', 100e-3);
```

Plot the radiation pattern of the above antenna.

```
pattern(fco,1.30e9)
```



Output : Directivity
Frequency : 1.3 GHz
Max value : 3.39 dBi
Min value : -17.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2016b

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

[2] Stutzman, W. L. and Gary A. Thiele. *Antenna Theory and Design*. 3rd Ed. River Street, NJ: John Wiley & Sons, 2013.

See Also

`invertedL` | `invertedF` | `invertedLcoplanar`

Topics

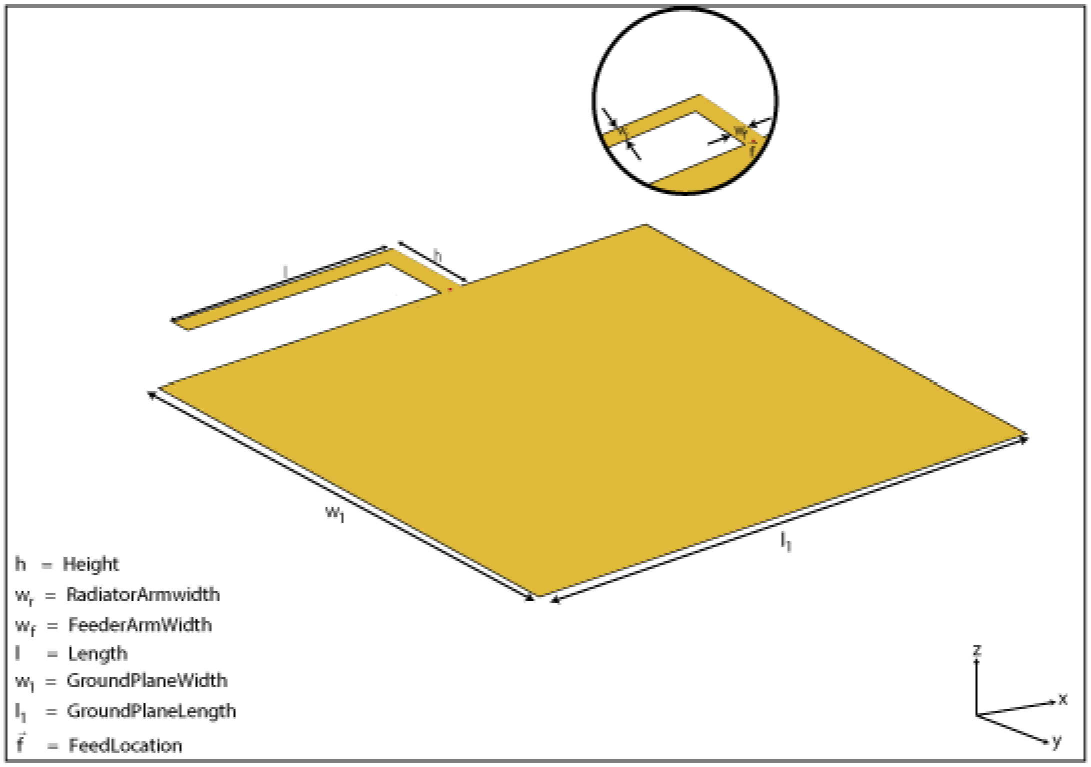
“Rotate Antennas and Arrays”

invertedLcoplanar

Create inverted-L antenna in same plane as rectangular ground plane

Description

The `invertedLcoplanar` object is a coplanar inverted-L antenna with the rectangular ground plane. By default, the dimensions are chosen for an operating frequency of 1.6 GHz. This antenna is used in applications that require low-profile narrow-bandwidth antennas, such as the transmitter for a garage door opener and Internet of Things (IoT) applications.



Creation

Syntax

```
lco = invertedLcoplanar
lco = invertedLcoplanar(Name,Value)
```

Description

`lco = invertedLcoplanar` creates a coplanar inverted-L antenna with the rectangular ground plane. By default, the antenna dimensions are for an operating frequency of 1.6 GHz.

`lco = invertedLcoplanar(Name,Value)` creates a coplanar inverted-L antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`RadiatorArmWidth` — Width of radiating arm

`0.0020` (default) | scalar

Width of radiating arm, specified as a scalar in meters.

Example: `'RadiatorArmWidth',0.05`

Data Types: `double`

`FeederArmWidth` — Width of feeding arm

`0.0020` (default) | scalar

Width of feeding arm, specified as scalar in meters.

Example: `'FeederArmWidth',0.05`

Data Types: `double`

`Height` — Height of antenna

`0.0100` (default) | scalar

Height of antenna from ground plane, specified as a scalar in meters.

Example: `'Height',0.0800`

Data Types: `double`

`Length` — Length of stub from feed to open end

`0.0350` (default) | scalar

Length of the stub from the feed to the open-end, specified as a scalar in meters.

Example: `'Length',0.0800`

Data Types: `double`

`GroundPlaneLength` — Length of ground plane

`0.0800` (default) | scalar in meters

Length of the ground plane, specified as a scalar in meters.

Example: `'GroundPlaneLength',0.035`

Data Types: `double`

`GroundPlaneWidth` — Width of ground plane

`0.0700` (default) | scalar

Width of the ground plane, specified as a scalar in meters.

Example: `'GroundPlaneWidth',0.035`

Data Types: `double`

`FeedOffset` — Signed distance from center of ground plane

`0` (default) | scalar

Signed distance from center of groundplane, specified a scalar in meters.

Example: `'FeedOffset',0.06`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `lco.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.

- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
show	Display antenna, array structures or shapes
vswr	Voltage standing wave ratio of antenna

Examples

Coplanar Inverted-L Antenna

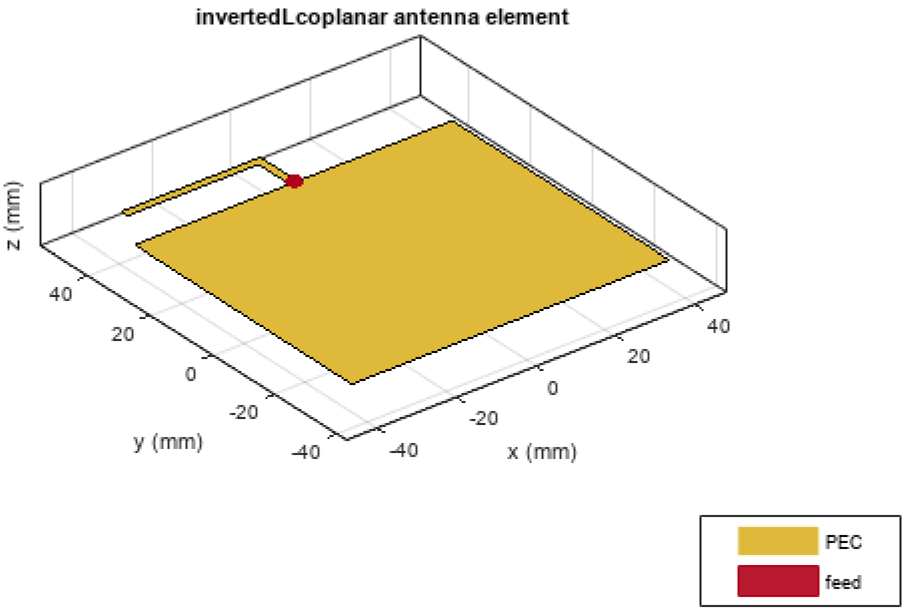
Create a default coplanar inverted-L antenna and view it.

```
lco =  invertedLcoplanar
```

```
lco = 
  invertedLcoplanar with properties:

     RadiatorArmWidth: 0.0020
       FeederArmWidth: 0.0020
```

```
                Length: 0.0350
                Height: 0.0100
     GroundPlaneLength: 0.0800
      GroundPlaneWidth: 0.0700
            FeedOffset: 0
             Conductor: [1x1 metal]
                  Tilt: 0
              TiltAxis: [1 0 0]
                  Load: [1x1 lumpedElement]
```

```
show(lco)
```



Impedance of Coplanar Inverted-L Antenna

Create a coplanar inverted-L antenna of length 0.050 m, height 0.014 m, ground plane length 0.1 m, and ground plane width 0.1 m.

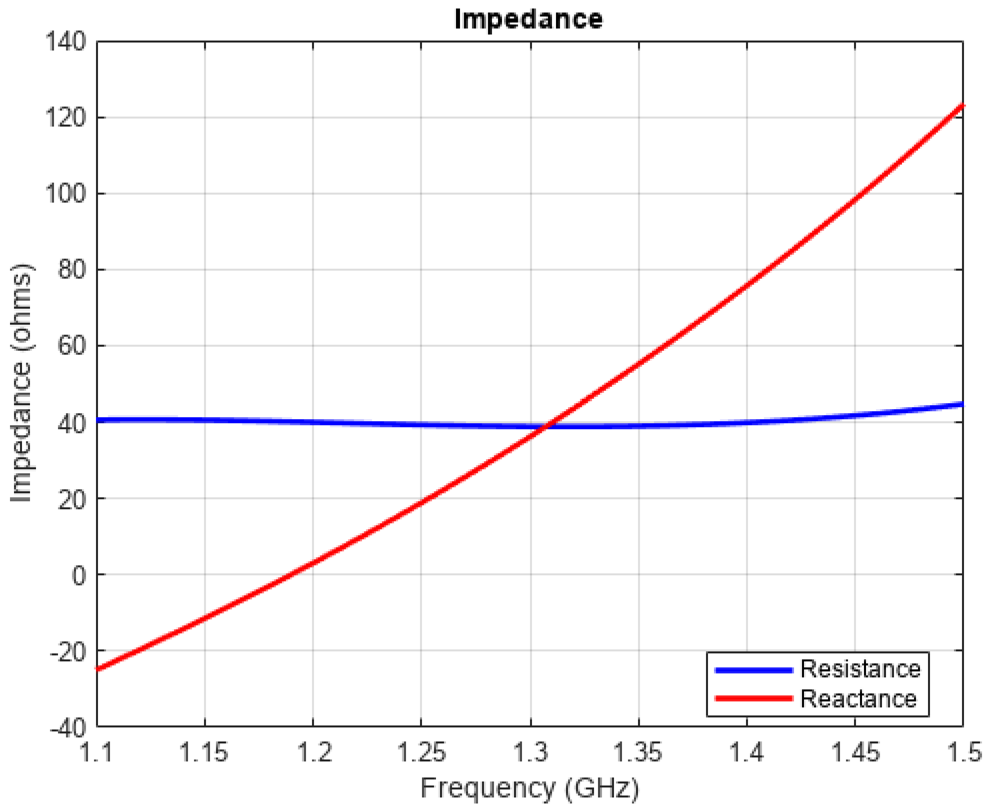
```
lco = invertedLcoplanar('Length',50e-3, 'Height',14e-3,...
    'GroundPlaneLength',100e-3,'GroundPlaneWidth',100e-3)
```

```
lco = 
  invertedLcoplanar with properties:

     RadiatorArmWidth: 0.0020
```

```
       FeederArmWidth: 0.0020
               Length: 0.0500
               Height: 0.0140
    GroundPlaneLength: 0.1000
     GroundPlaneWidth: 0.1000
           FeedOffset: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

Plot the impedance over 1.1 GHz to 1.5 GHz in steps of 10 MHz.

```
impedance(lco,1.1e9:10e6:1.5e9);
```



Version History

Introduced in R2016b

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

[2] Stutzman, W. L. and Gary A. Thiele. *Antenna Theory and Design*. 3rd Ed. River Street, NJ: John Wiley & Sons, 2013.

See Also

`invertedL` | `invertedF` | `invertedFcoplanar`

Topics

"Rotate Antennas and Arrays"

loopCircular

Create circular loop antenna

Description

The `loopCircular` object is a planar circular loop antenna on the *xy*- plane.

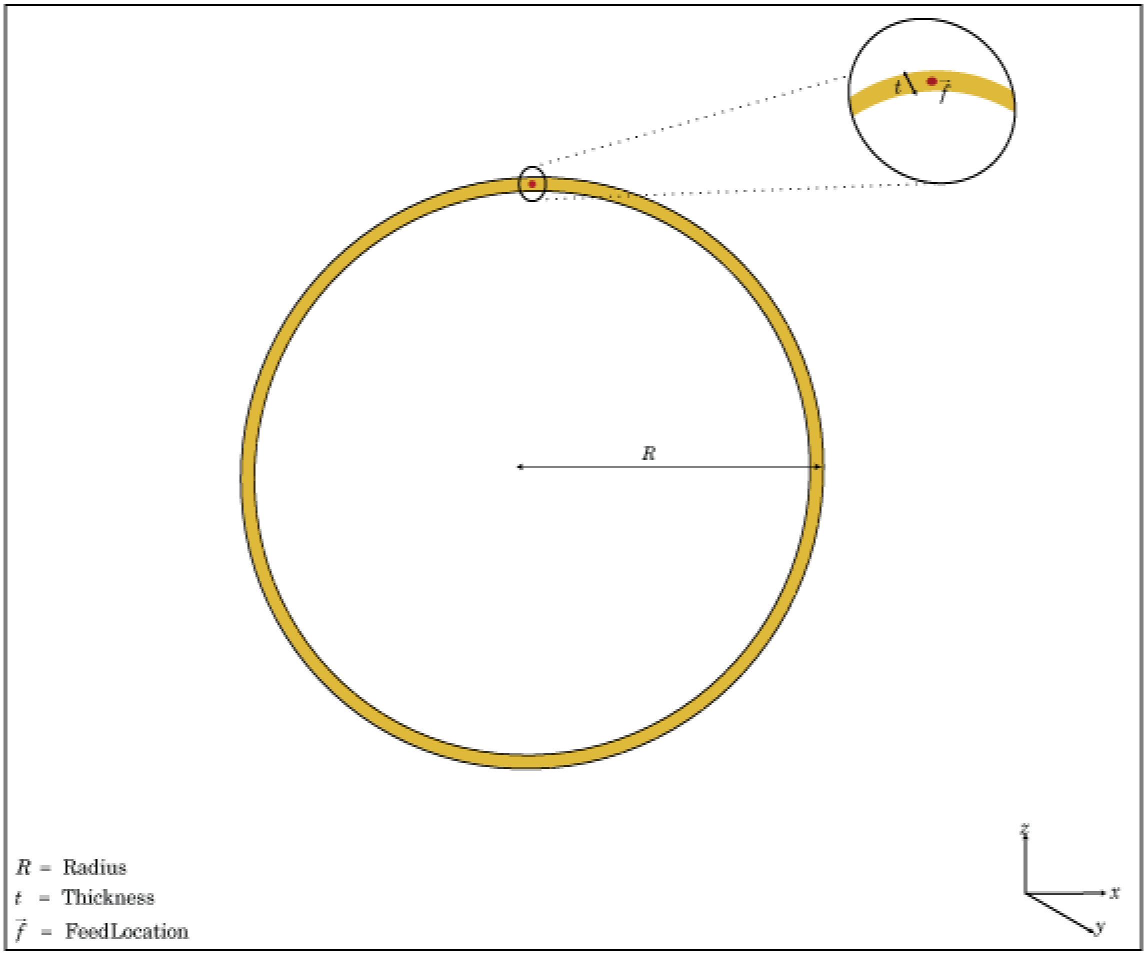
The thickness of the loop is related to the diameter of an equivalent cylinder loop by the equation

$$t = 2d = 4r$$

, where:

- *d* is the diameter of equivalent cylindrical loop
- *r* is the radius of equivalent cylindrical loop

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default circular loop antenna is fed at the positive *x*-axis. The point of the *x*-axis is at the midpoint of the inner and outer radii.



Creation

Syntax

```
lc = loopCircular
lc = loopCircular(Name,Value)
```

Description

`lc = loopCircular` creates a one wavelength circular loop antenna in the X-Y plane. By default, the circumference is chosen for the operating frequency 75 MHz.

`lc = loopCircular(Name,Value)` creates a one wavelength circular loop antenna, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Radius` — Outer radius of loop
0.6366 (default) | scalar

Outer radius of loop, specified as a scalar in meters.

Example: `'Radius',3`

Data Types: `double`

`Thickness` — Thickness of loop
0.0200 (default) | scalar

Thickness of loop, specified as a scalar in meters.

Example: `'Thickness',2`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `lc.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency

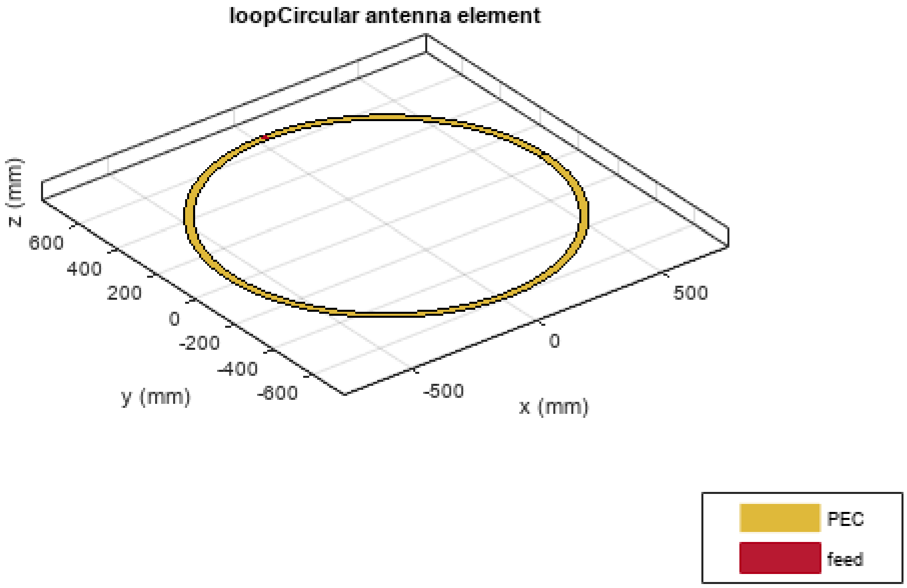
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Circular Loop Antenna

Create and view a circular loop with 0.65 m radius and 0.01 m thickness.

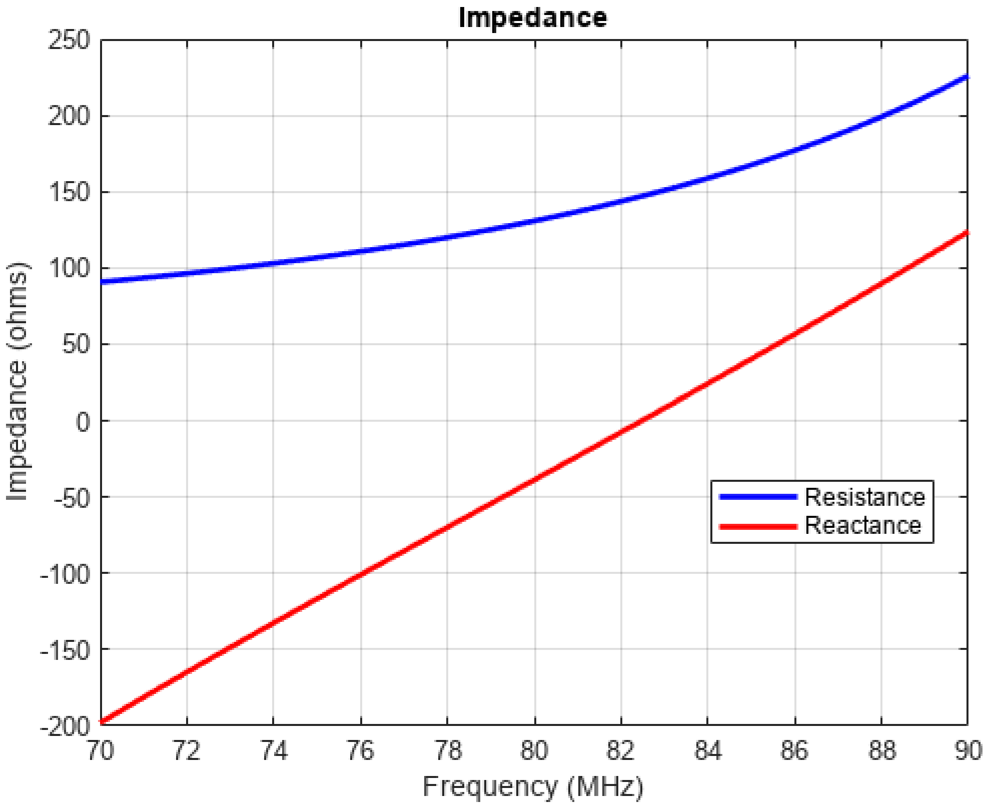
```
c = loopCircular('Radius',0.64,'Thickness',0.03);
show(c)
```



Impedance of Circular Loop Antenna

Calculate the impedance of a circular loop antenna over a frequency range of 70MHz-90MHz.

```
c = loopCircular('Radius',0.64,'Thickness',0.03);
impedance(c,linspace(70e6,90e6,31))
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`loopRectangular` | `dipole` | `slot`

Topics

"Rotate Antennas and Arrays"

loopRectangular

Create rectangular loop antenna

Description

The `loopRectangular` object is a rectangular loop antenna on the *xy*- plane.

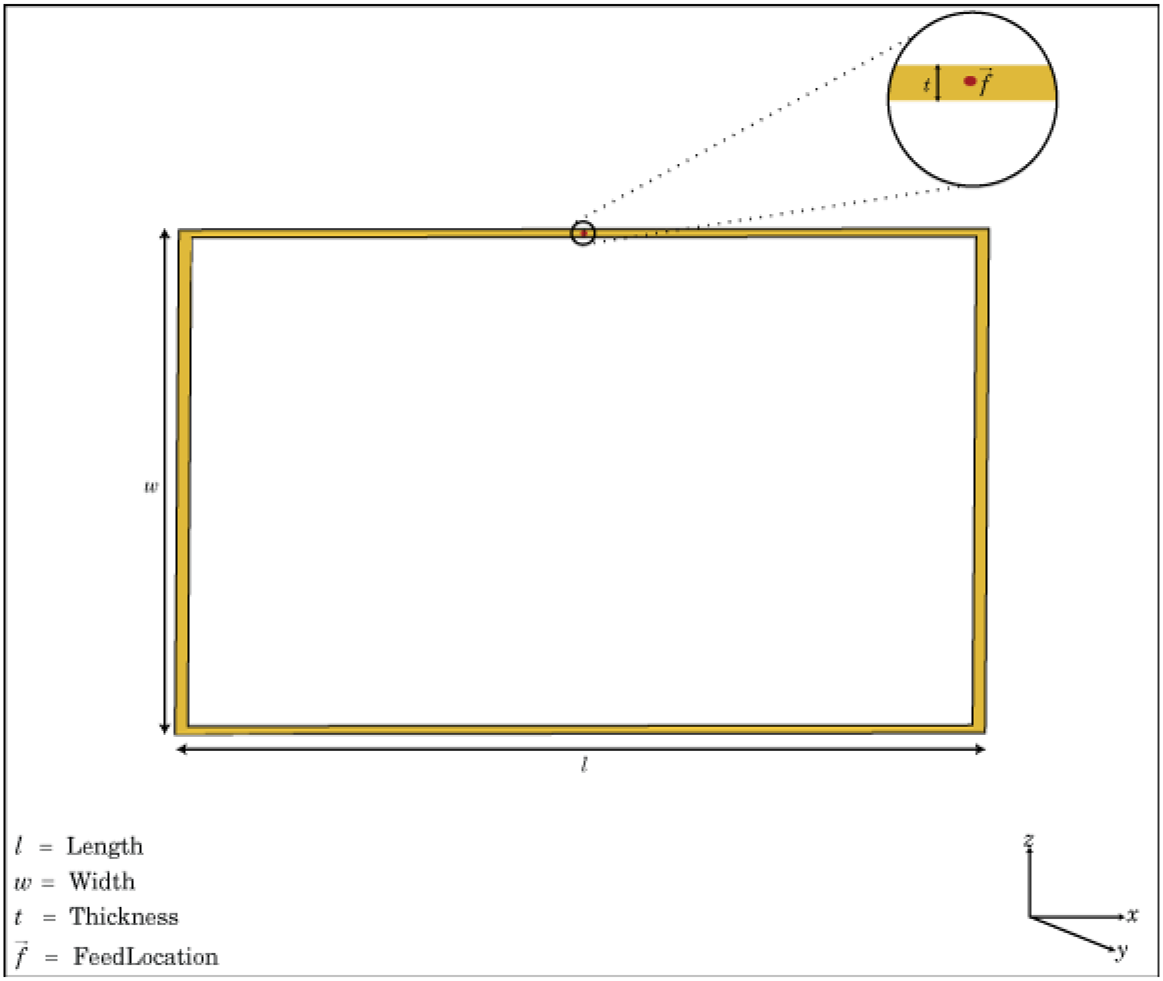
The thickness of the loop is related to the diameter of an equivalent cylinder loop by the equation

$$t = 2d = 4r$$

, where:

- *d* is the diameter of equivalent cylindrical loop
- *r* is the radius of equivalent cylindrical loop

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default circular loop antenna is fed at the positive *y*-axis. The point of the *y*-axis is the midpoint of the inner and outer perimeter of the loop.



Creation

Syntax

```
lr = loopRectangular
lr = loopRectangular(Name,Value)
```

Description

`lr = loopRectangular` creates a rectangular loop antenna in the X-Y plane. By default, the dimensions are chosen for the operating frequency 53 MHz.

`lr = loopRectangular(Name,Value)` creates a rectangular loop antenna, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retains their default values.

Properties

`Length` — Loop length along *x*-axis

2 (default) | scalar

Loop length along *x*-axis, specified as a scalar in meters.

Example: `'Length',3`

Data Types: `double`

`Width` — Loop width along *y*-axis

1 (default) | scalar

Loop width along *y*-axis, specified as a scalar in meters.

Example: `'Width',2`

Data Types: `double`

`Thickness` — Loop thickness

0.0100 (default) | scalar

Loop thickness, specified as a scalar in meters.

Example: `'Thickness',2`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `lr.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array

rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

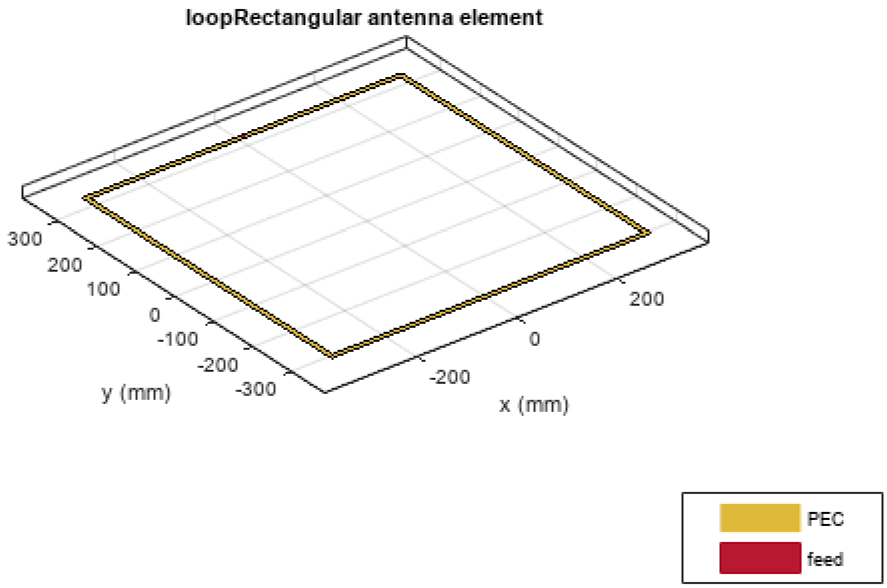
Create and View Rectangular Loop Antenna

Create and view a rectangular loop antenna with 0.64m length, 0.64m width.

```
r = loopRectangular('Length',0.64,'Width',0.64)
```

```
r = 
  loopRectangular with properties:

       Length: 0.6400
        Width: 0.6400
    Thickness: 0.0100
    Conductor: [1x1 metal]
         Tilt: 0
     TiltAxis: [1 0 0]
         Load: [1x1 lumpedElement]
```

```
show(r)
```



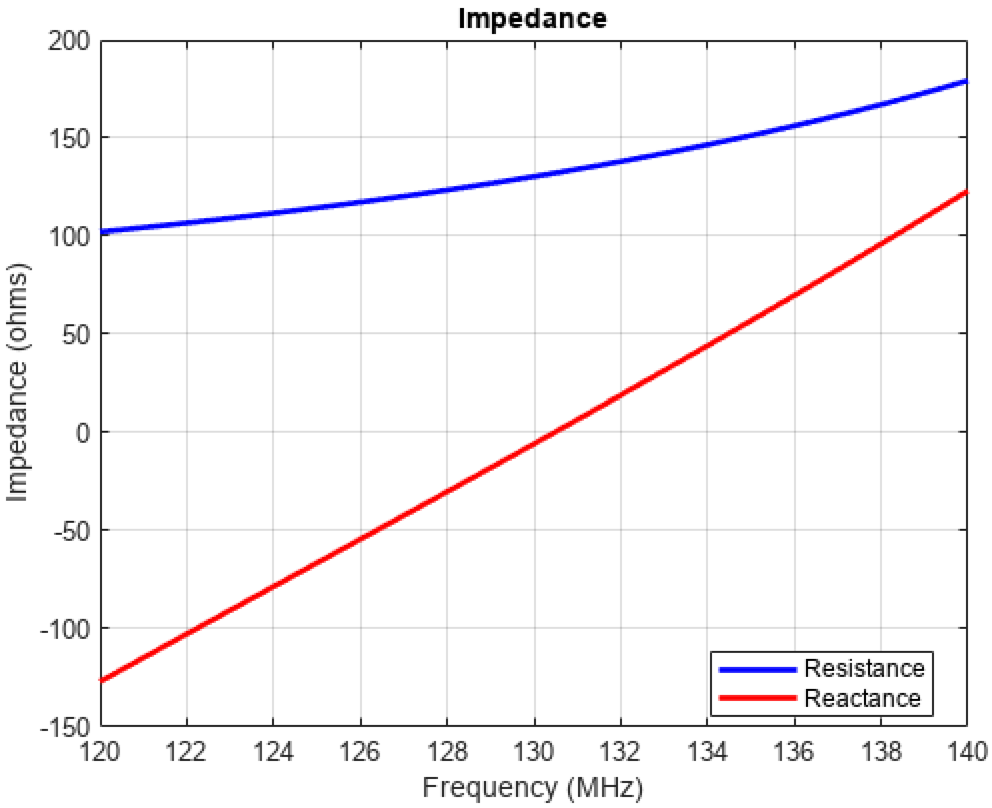
Impedance of Rectangular Loop Antenna

Calculate the impedance of a rectangular loop antenna over a frequency range of 120MHz-140MHz.

```
r = loopRectangular('Length',0.64,'Width',0.64)
```

```
r = 
  loopRectangular with properties:

        Length: 0.6400
         Width: 0.6400
     Thickness: 0.0100
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
impedance(r,linspace(120e6,140e6,31))
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`loopCircular` | `dipole` | `monopole`

Topics

“Rotate Antennas and Arrays”

monopole

Create monopole antenna over rectangular ground plane

Description

The `monopole` object is a monopole antenna mounted over a rectangular ground plane.

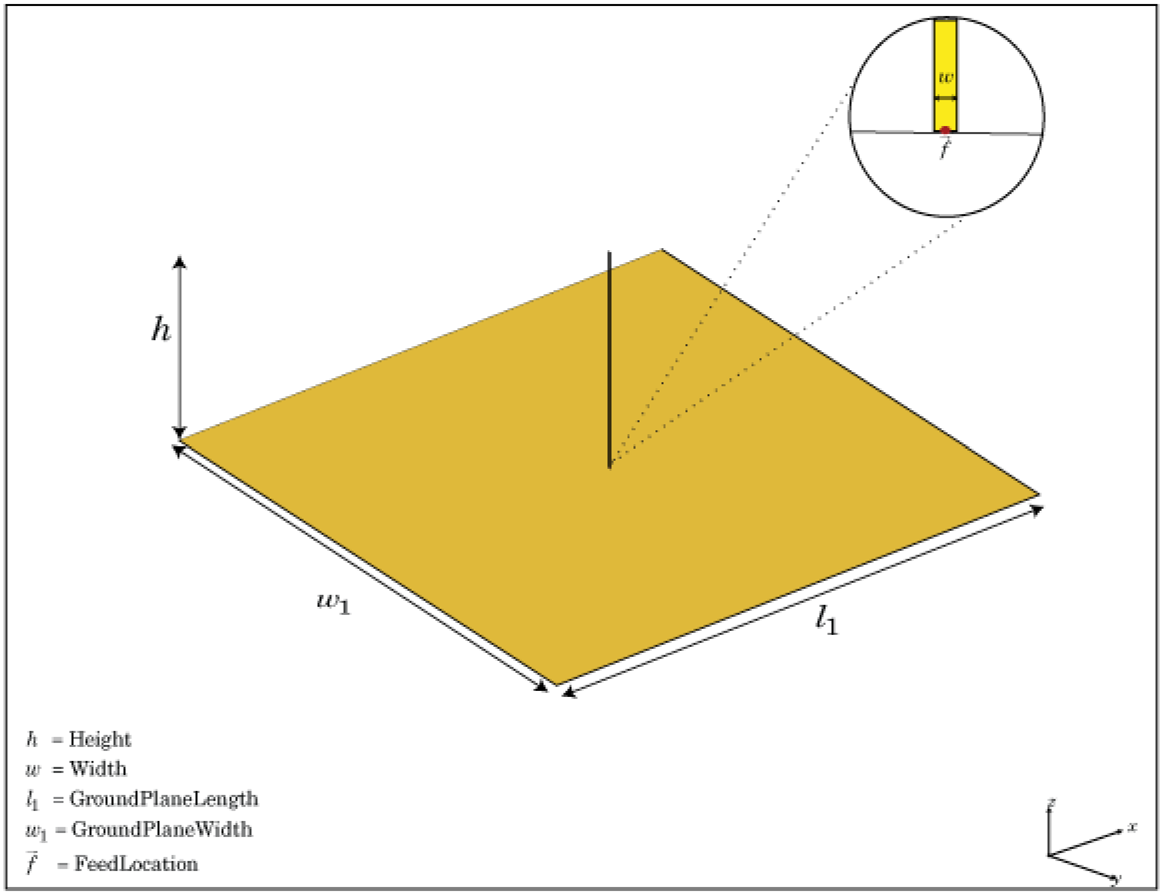
The width of the monopole is related to the diameter of an equivalent cylindrical monopole by the equation

$$w = 2d = 4r$$

, where:

- d is the diameter of equivalent cylindrical monopole
- r is the radius of equivalent cylindrical monopole.

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default monopole is center-fed. The feed point coincides with the origin. The origin is located on the *xy*- plane.



Creation

Syntax

```
mpl = monopole
mpl = monopole(Name,Value)
```

Description

`mpl = monopole` creates a quarter-wavelength monopole antenna.

`mpl = monopole(Name,Value)` creates a quarter-wavelength monopole antenna with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Height` — Height of vertical element along *z*-axis
1 (default) | scalar

Height of vertical element along *z*-axis, specified as a scalar in meters. By default, the height is chosen for an operating frequency of 75 MHz.

Example: `'Height',3`

Data Types: `double`

`Width` — Monopole width
0.1000 (default) | scalar

Monopole width, specified as a scalar in meters.

Note Monopole width should be less than `'Height'`/4 and greater than `'Height'`/1001. [2]

Example: `'Width',0.05`

Data Types: `double`

`GroundPlaneLength` — Ground plane length along *x*-axis
2 (default) | scalar

Ground plane length along *x*-axis, specified as a scalar in meters. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',4`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis
2 (default) | scalar

Ground plane width along *y*-axis, specified as a scalar in meters. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: 'GroundPlaneWidth',2.5

Data Types: double

FeedOffset — Signed distance from center along length and width of ground plane

[0 0] (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector.

Example: 'FeedOffset',[2 1]

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: mpl.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Monopole Antenna

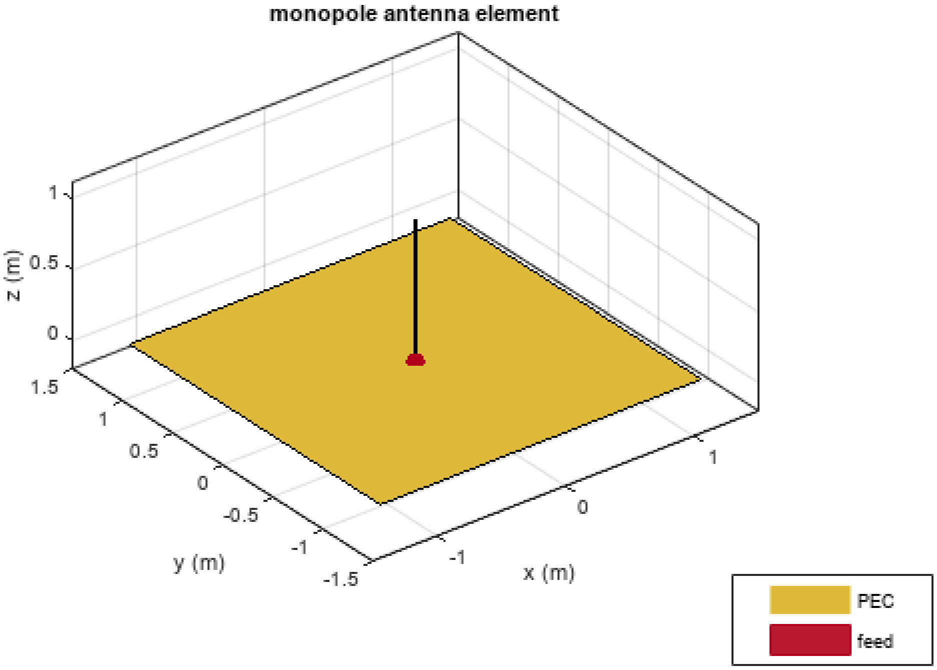
Create and view a monopole of 1 m length, 0.01 m width and ground plane of dimensions 2.5mx2.5m.

```
m = monopole('GroundPlaneLength',2.5,'GroundPlaneWidth',2.5)
```

```
m = 
  monopole with properties:
```

```
               Height: 1
                Width: 0.0100
    GroundPlaneLength: 2.5000
     GroundPlaneWidth: 2.5000
           FeedOffset: [0 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

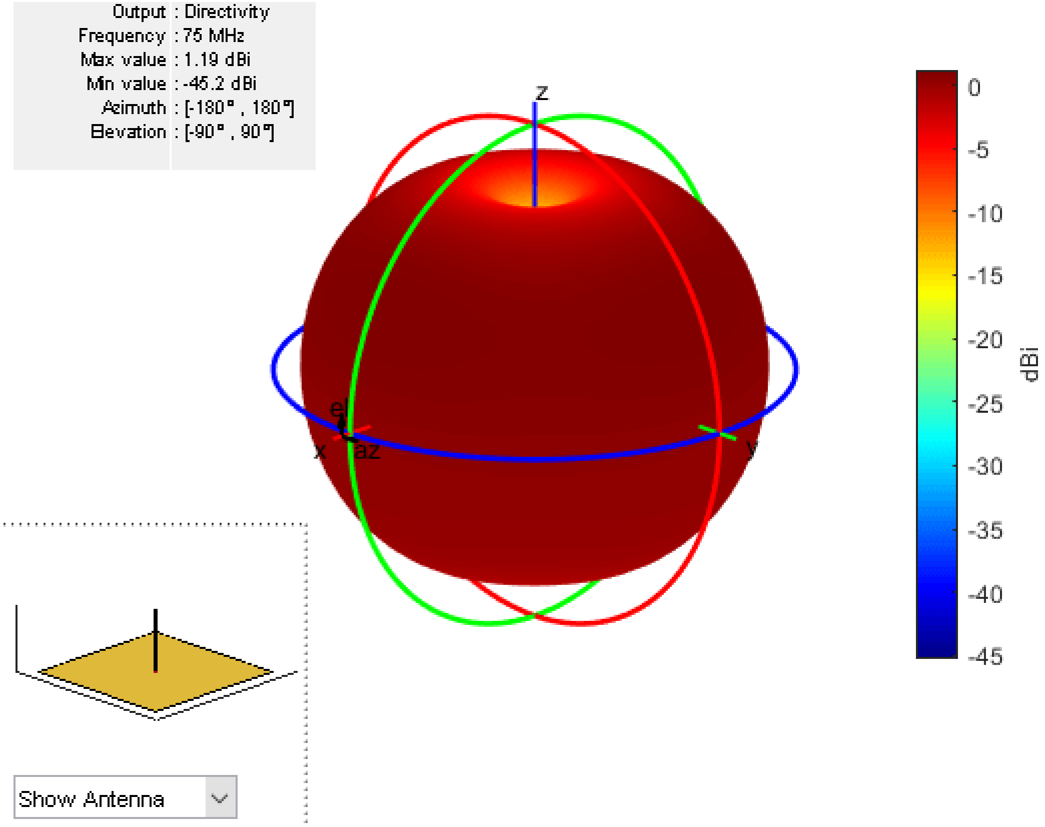
```
show(m)
```



Radiation Pattern of Monopole Antenna

Radiation pattern of a monopole at a frequency of 75 MHz.

```
m = monopole('GroundPlaneLength',2.5, 'GroundPlaneWidth',2.5);
pattern (m,75e6)
```



Output : Directivity
Frequency : 75 MHz
Max value : 1.19 dBi
Min value : -45.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

See Also

`monopoleTopHat` | `dipole` | `patchMicrostrip`

Topics

"Rotate Antennas and Arrays"

monopoleTopHat

Create capacitively loaded monopole antenna over rectangular ground plane

Description

The `monopoleTopHat` object is a top-hat monopole antenna mounted over a rectangular ground plane. The monopole always connects with the center of top hat. The top hat builds up additional capacitance to ground within the structure. This capacitance reduces the resonant frequency of the antenna without increasing the size of the element.

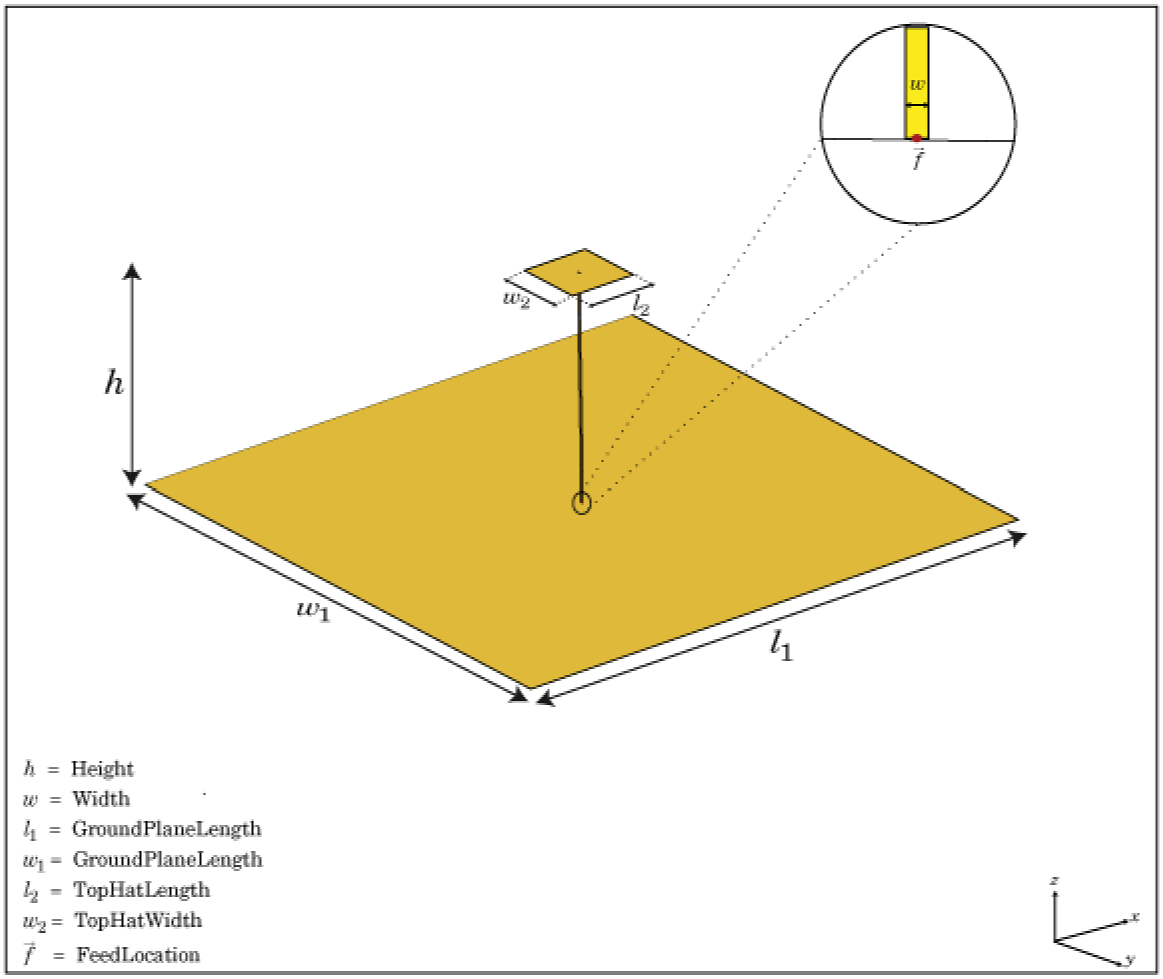
The width of the monopole is related to the diameter of an equivalent cylindrical monopole by the expression

$$w = 2d = 4r$$

where:

- d is the diameter of equivalent cylindrical monopole
- r is the radius of equivalent cylindrical monopole.

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default top-hat monopole is center-fed. The feed point coincides with the origin. The origin is located on the *xy*- plane.



Creation

Syntax

```
mth = monopoleTopHat
mth = monopoleTopHat(Name,Value)
```

Description

`mth = monopoleTopHat` creates a capacitively loaded monopole antenna over a rectangular ground plane.

`mth = monopoleTopHat(Name,Value)` creates a capacitively loaded monopole antenna with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`,`...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Height` — Monopole height

1 (default) | scalar

Monopole height, specified as a scalar in meters. By default, the height is chosen for an operating frequency of 75 MHz.

Example: `'Height',3`

Data Types: `double`

`Width` — Monopole width

0.1000 (default) | scalar

Monopole width, specified as a scalar in meters.

Note Monopole width should be less than `'Height'`/4 and greater than `'Height'`/1001. [2]

Example: `'Width',0.05`

Data Types: `double`

`GroundPlaneLength` — Ground plane length along *x*-axis

2 (default) | scalar

Ground plane length along *x*-axis, specified as a scalar in meters. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',4`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

2 (default) | scalar

Ground plane width along *y*-axis, specified as a scalar in meters. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',2.5`

Data Types: `double`

`TopHatLength` — Top hat length along *x*-axis

0.2500 (default) | scalar

Top hat length along *x*-axis, specified as a scalar in meters.

Example: `'TopHatLength',4`

Data Types: `double`

`TopHatWidth` — Top hat width along *y*-axis

0.2500 (default) | scalar

Top hat width along *y*-axis, specified as a scalar in meters.

Example: `'TopHatWidth',4`

Data Types: double

FeedOffset — Signed distance from center along length and width of ground plane

[0 0] (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector.

Example: 'FeedOffset',[2 1]

Data Types: double

Substrate — Type of dielectric material

'Air' (default) | dielectric object

Type of dielectric material used as the substrate, specified as a dielectric object. You can also specify multiple dielectric layers. When creating multiple dielectric layers, in the dielectric function, specify the name, thickness, loss tangent, and relative permittivity of each layer. For more information, see dielectric. For more information on dielectric substrate meshing, see “Meshing”.

Example: d = dielectric('FR4'); mth = monopoleTopHat('Substrate',d)

Example: d = dielectric('FR4'); mth = MonopoleTopHat; mth.Substrate = d

Example: d = dielectric('Name',{'FR4','Teflon'},'Thickness',[0.5 0.5],'LossTangent',[0.002 0.002],'EpsilonR',[4.8 2.1]); mth = monopoleTopHat('Substrate',d)

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: mth.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

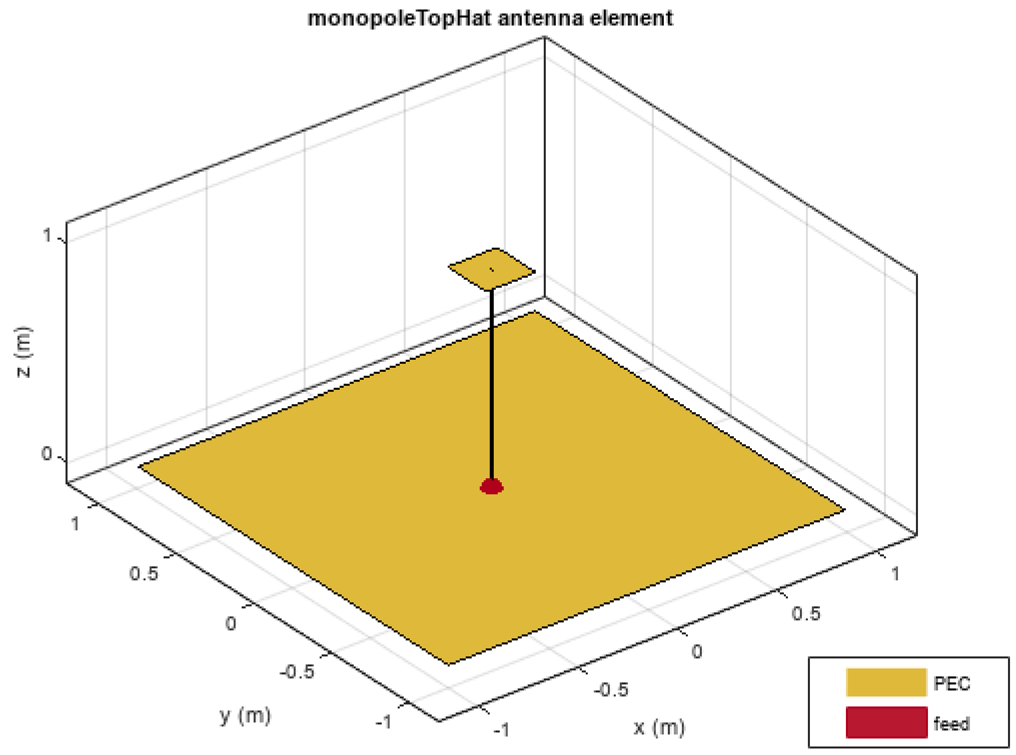
Create and View Top-Hat Monopole

Create and view a top-hat monopole with 1 m length, 0.01 m width, groundplane dimensions 2mx2m and top hat dimensions 0.25mx0.25m.

```
th = monopoleTopHat
```

```
th = 
  monopoleTopHat with properties:

               Height: 1
                Width: 0.0100
    GroundPlaneLength: 2
     GroundPlaneWidth: 2
         TopHatLength: 0.2500
          TopHatWidth: 0.2500
            Substrate: [1x1 dielectric]
           FeedOffset: [0 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

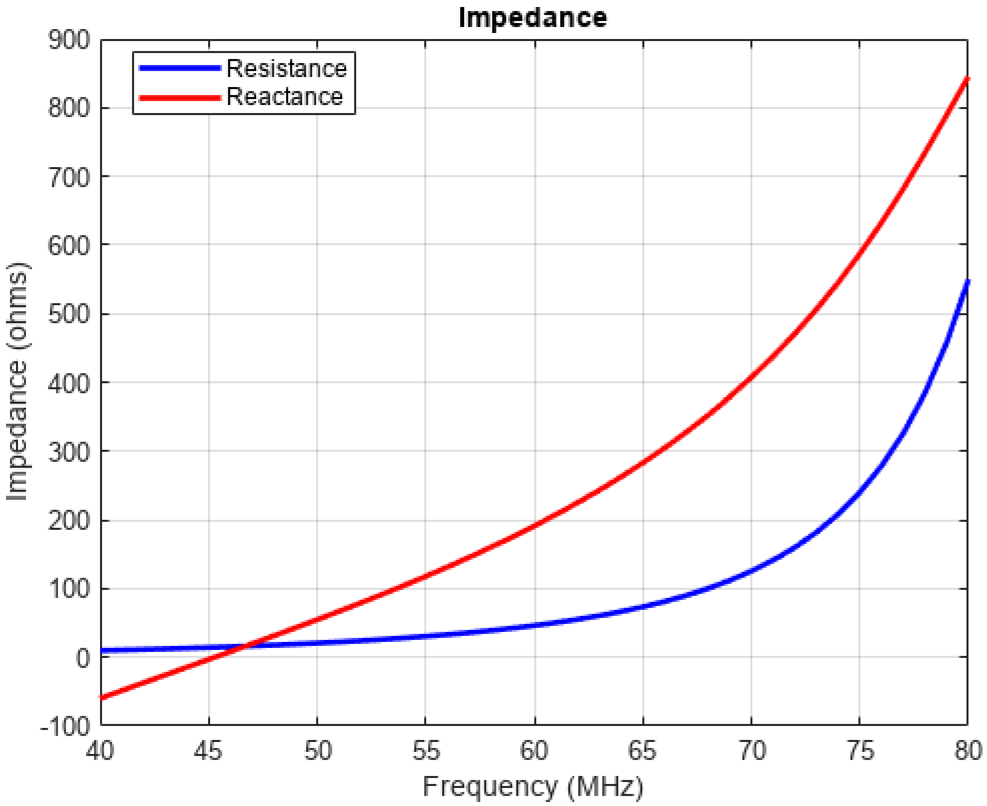
```
show(th)
```



Calculate Impedance of Top-Hat Monopole Antenna

Calculate and plot the impedance of a top-hat monopole over a frequency range of 40 MHz-80 MHz.

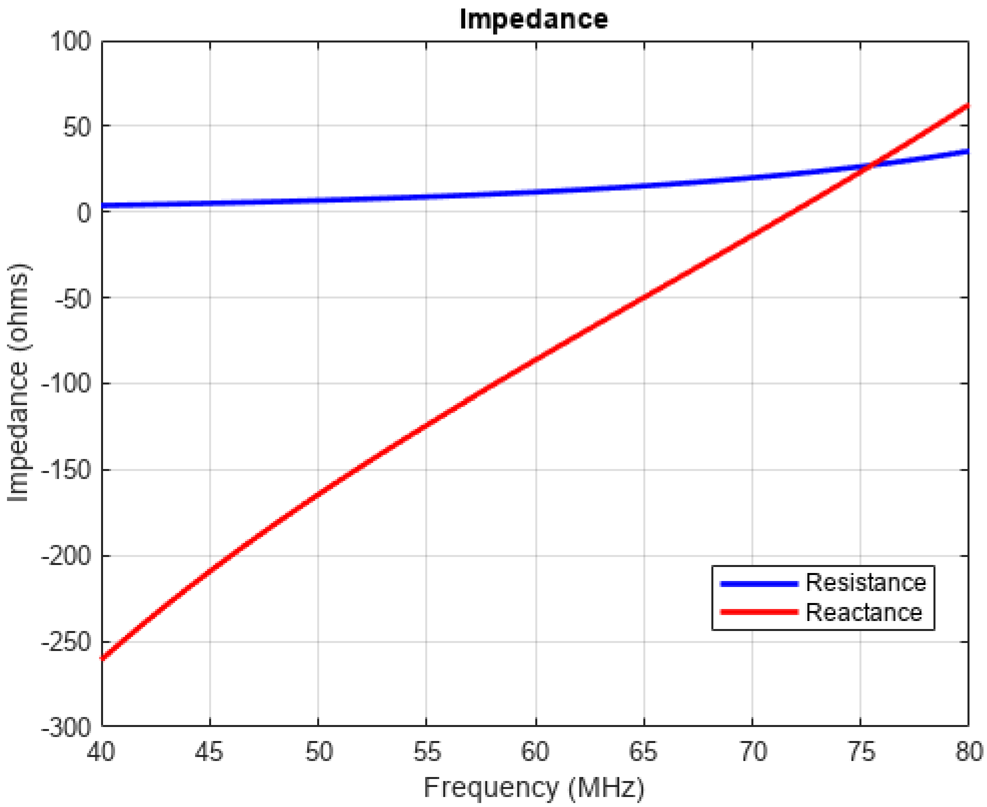
```
th = monopoleTopHat;
impedance(th,linspace(40e6,80e6,41));
```



Compare Impedance of Top-Hat Monopole Antenna and Monopole Antenna

Impedance comparison between a monopole of similar dimensions and the top-hat monopole in example 2.

```
m = monopole;
figure
impedance(m,linspace(40e6,80e6,41));
```



Top-Hat Monopole with Multiple Dielectric Layers

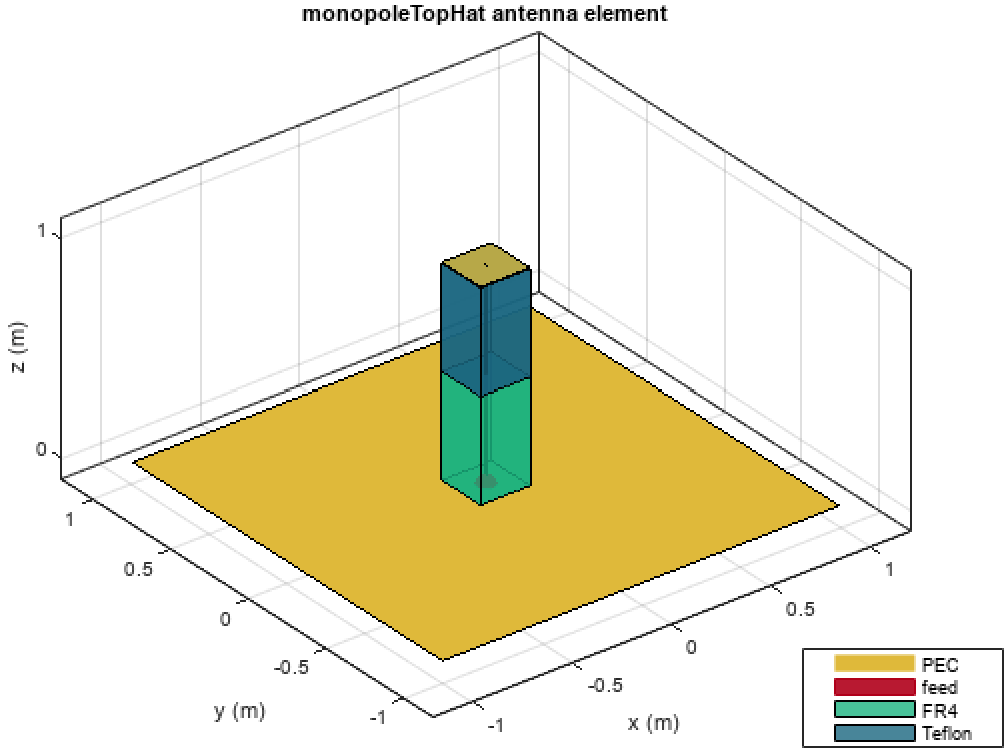
Create a top-hat monopole with default dimensions and a substrate with two dielectric layers.

```
mth = monopoleTopHat;
d = dielectric('Name',{'FR4','Teflon'},'Thickness',[0.5 0.5],'LossTangent',[0.002 0.002],'Epsilon
mth.Substrate = d
```

```
mth = 
  monopoleTopHat with properties:

               Height: 1
                Width: 0.0100
    GroundPlaneLength: 2
     GroundPlaneWidth: 2
         TopHatLength: 0.2500
          TopHatWidth: 0.2500
            Substrate: [1x1 dielectric]
           FeedOffset: [0 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

View the top-hat monopole antenna.

```
show(mth)
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

See Also

`monopole` | `dipole` | `patchMicrostrip`

Topics

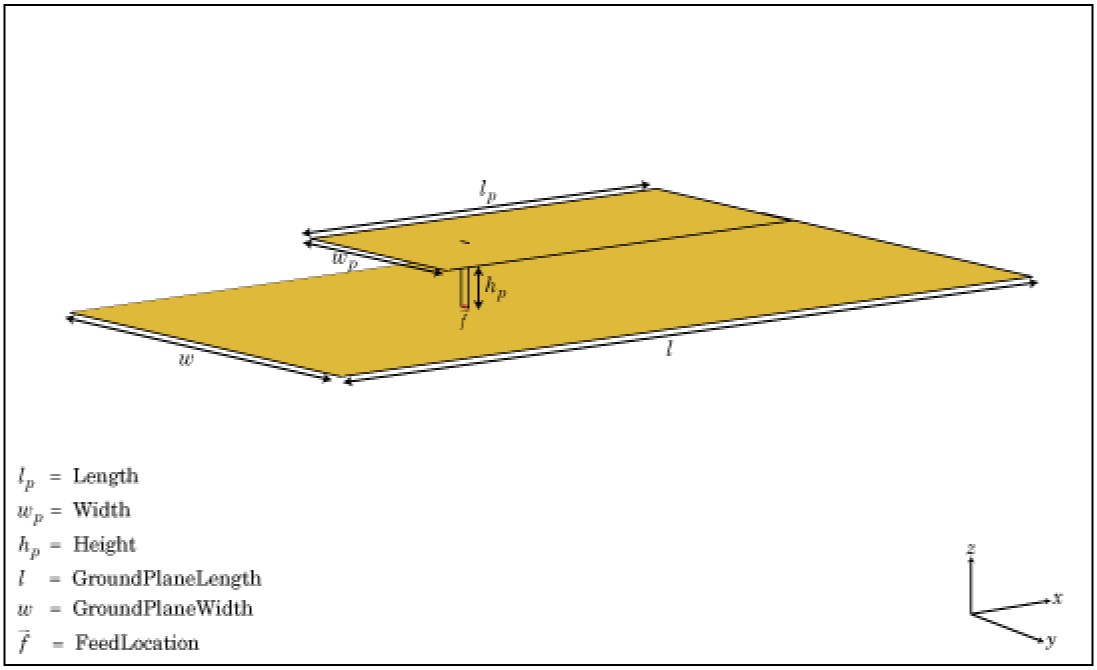
"Rotate Antennas and Arrays"

patchMicrostrip

Create microstrip patch antenna

Description

The `patchMicrostrip` object is a microstrip patch antenna. The default patch is centered at the origin. The feed point is along the length of the antenna.



Creation

Syntax

```
pm = patchMicrostrip
pm = patchMicrostrip(Name,Value)
```

Description

`pm = patchMicrostrip` creates a microstrip patch antenna.

`pm = patchMicrostrip(Name,Value)` creates a microstrip patch antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Length` — Patch length along *x*-axis

0.0750 (default) | scalar

Patch length, specified as a scalar in meters. By default, the length is measured along the *x*-axis.

Example: `'Length',50e-3`

Data Types: `double`

`Width` — Patch width along the *y*-axis

0.0375 (default) | scalar

Patch width, specified as a scalar in meters. By default, the width is measured along the *y*-axis.

Example: `'Width',60e-3`

Data Types: `double`

`Height` — Height of substrate

0.0060 (default) | scalar

Height of substrate, specified as a scalar in meters.

Example: `'Height',37e-3`

Data Types: `double`

`Substrate` — Type of dielectric material

'Air' (default) | `dielectric` function

Type of dielectric material used as a substrate, specified as a dielectric material object. You can choose any material from the `DielectricCatalog` or use your own dielectric material. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be lesser than the ground plane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); ant.Substrate = d`

`GroundPlaneLength` — Ground plane length along *x*-axis

0.1500 (default) | scalar

Ground plane length, specified as a scalar in meters. By default, ground plane length is measured along *x*-axis. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

0.0750 (default) | scalar

Ground plane width, specified as a scalar in meters. By default, ground plane width is measured along *y*-axis. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',120e-3`

Data Types: `double`

`PatchCenterOffset` — Signed distance from center along length and width of ground plane

`[0 0]` (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector in meters. Use this property to adjust the location of the patch relative to the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Data Types: `double`

`FeedOffset` — Signed distance from center along length and width of ground plane

[–0.0187 0] (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector. Use this property to adjust the location of the feedpoint relative to ground plane and patch. Place the feed sufficiently inside from the edges of the patch to successfully mesh it during analysis.

Example: `'FeedOffset',[0.01 0.01]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array

rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

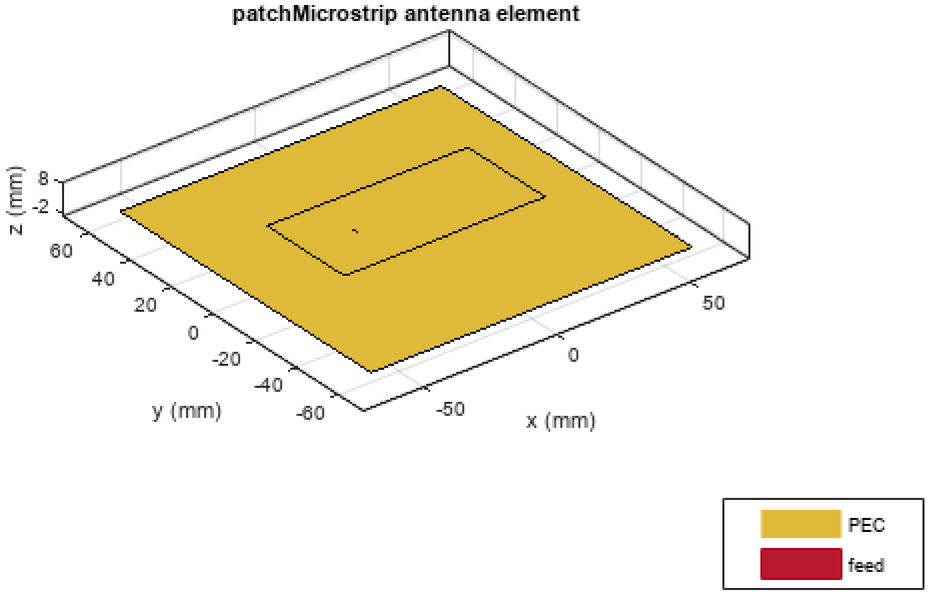
Create and View Microstrip Patch Antenna

Create and view a microstrip patch with specified parameters.

```
pm = patchMicrostrip('Length',75e-3, 'Width',37e-3,                 ...
        'GroundPlaneLength',120e-3, 'GroundPlaneWidth',120e-3)
```

```
pm = 
  patchMicrostrip with properties:

               Length: 0.0750
                Width: 0.0370
               Height: 0.0060
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1200
     GroundPlaneWidth: 0.1200
    PatchCenterOffset: [0 0]
           FeedOffset: [-0.0187 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show (pm)
```



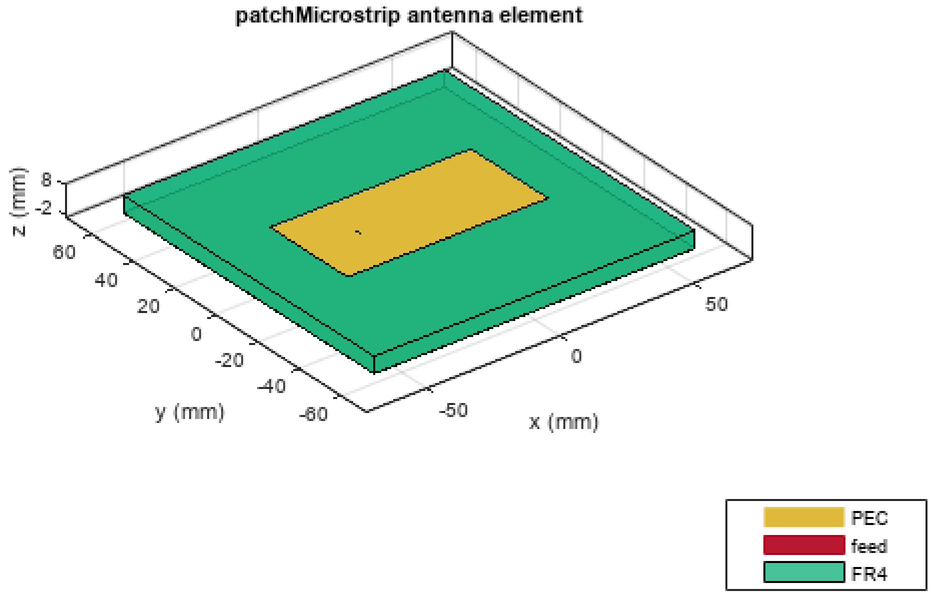
Radiation Pattern of Microstrip Patch Antenna

Create a microstrip patch antenna using **'FR4'** as the dielectric substrate.

```
d = dielectric('FR4');
pm = patchMicrostrip('Length',75e-3,'Width',37e-3,     ...
        'GroundPlaneLength',120e-3,'GroundPlaneWidth',120e-3, ...
        'Substrate',d)
```

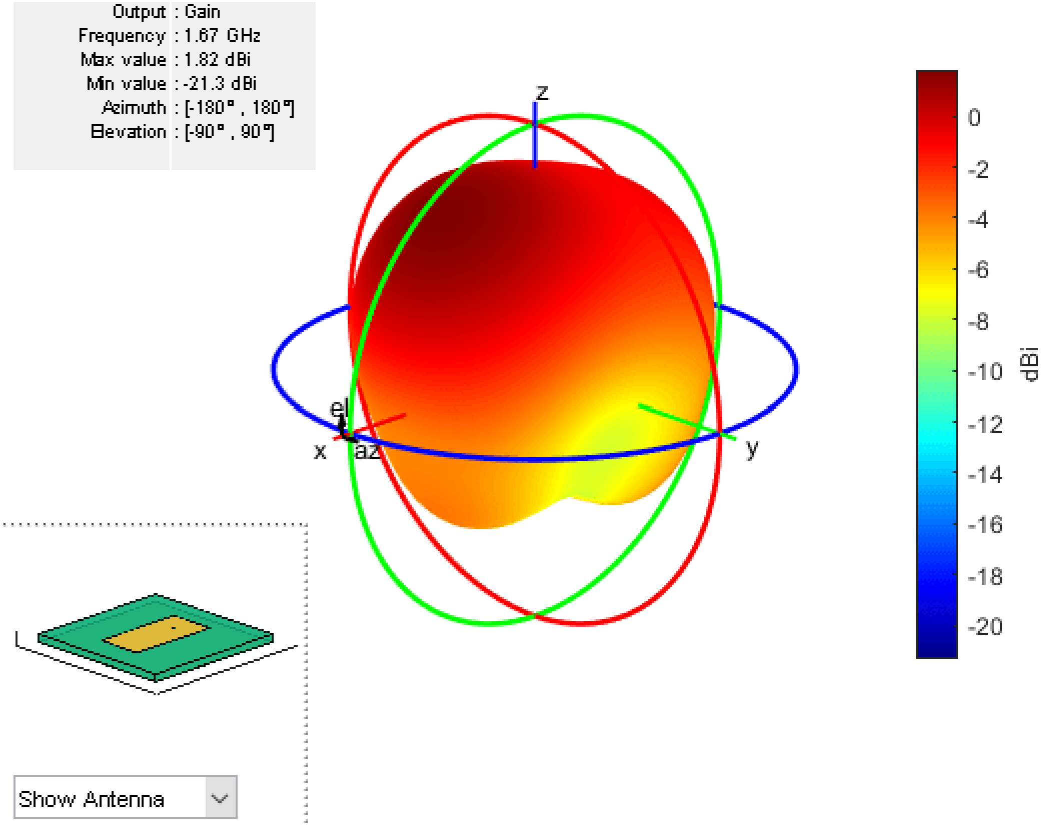
```
pm = 
  patchMicrostrip with properties:

             Length: 0.0750
              Width: 0.0370
             Height: 0.0060
          Substrate: [1x1 dielectric]
  GroundPlaneLength: 0.1200
   GroundPlaneWidth: 0.1200
  PatchCenterOffset: [0 0]
         FeedOffset: [-0.0187 0]
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(pm)
```



Plot the radiation pattern of the antenna at a frequency of 1.67 GHz.

```
figure
pattern(pm,1.67e9)
```



```
   Output : Gain
Frequency : 1.67 GHz
Max value : 1.82 dBi
Min value : -21.3 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

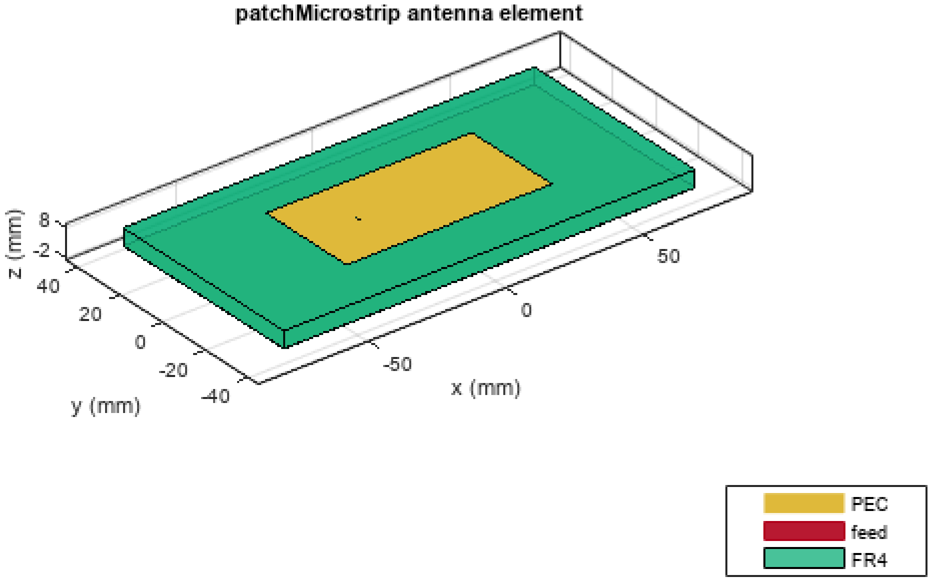
Impedance of Microstrip Patch Antenna

Create a microstrip patch antenna using **'FR4'** as the dielectric substrate.

```
d = dielectric('FR4');
pm = patchMicrostrip('Substrate',d)
```

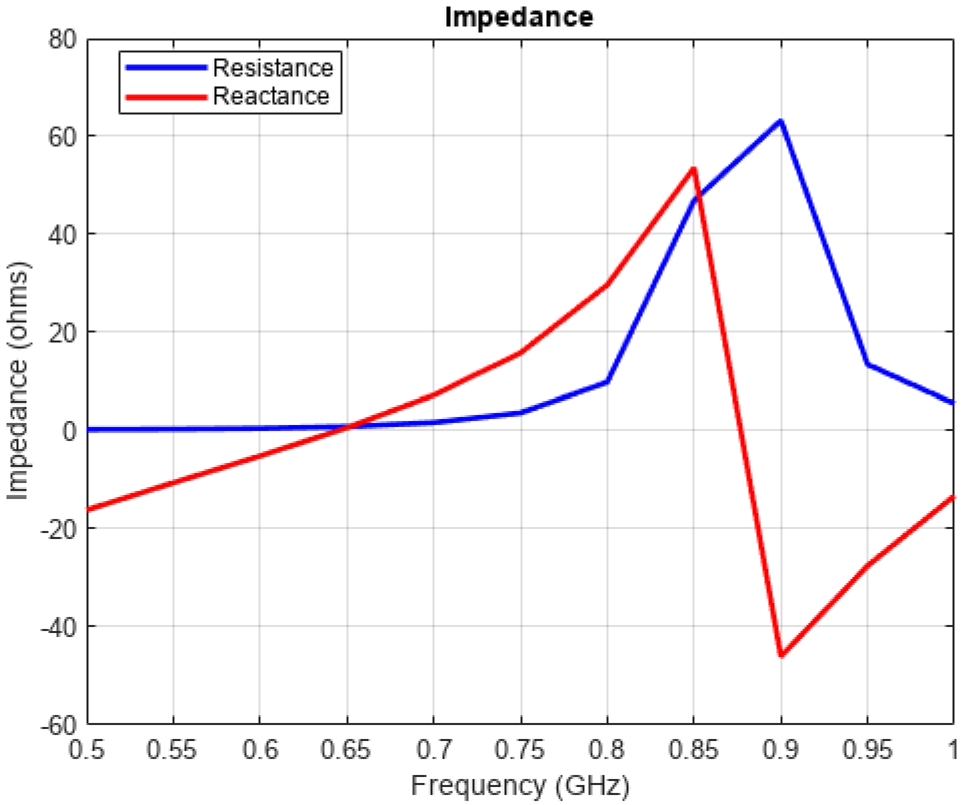
```
pm = 
  patchMicrostrip with properties:

               Length: 0.0750
                Width: 0.0375
               Height: 0.0060
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1500
     GroundPlaneWidth: 0.0750
    PatchCenterOffset: [0 0]
           FeedOffset: [-0.0187 0]
            Conductor: [1x1 metal]
                 Tilt: 0
            TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(pm)
```



Calculate and plot the impedance of the antenna over the specified frequency range.

```
impedance(pm,linspace(0.5e9,1e9,11));
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`pifa` | `vivaldi` | `yagiUda`

Topics

"ISM Band Patch Microstrip Antennas and Mutually Coupled Patches"
"Rotate Antennas and Arrays"

planeWaveExcitation

Create plane wave excitation environment for antenna or array

Description

The `planeWaveExcitation` object creates an environment in which a plane wave excites an antenna or array. Plane wave excitation is a scattering solution that solves the receiver antenna problem.

Creation

Syntax

```
h = planeWaveExcitation
h = planeWaveExcitation(Name=Value)
```

Description

`h = planeWaveExcitation` creates an environment in which a plane wave excites an antenna or array. The default receiver antenna is a dipole that is excited by a plane wave travelling along the positive *x*-axis with a *z*-polarization.

`h = planeWaveExcitation(Name=Value)` sets properties using one or more name-value arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value arguments in any order as `Name1=Value1`, `...`, `NameN=ValueN`. Properties you do not specify retain their default values.

Properties

`Element` — Antenna or array element
`dipole` (default) | antenna object | array object

Antenna or array element, specified as an antenna or array object from the catalog.

Example: `Element=linearArray`

`Direction` — Incidence direction of plane wave
`[1 0 0]` (default) | three-element real vector

Incidence direction of the plane wave, specified as a three-element real vector containing the Cartesian coordinates of a point in space. The object creates the direction vector by joining a line from origin to this point.

Example: `Direction=[0 0 1]`

Data Types: `double`

`Polarization` — Polarization of incident electric field
`[0 0 1]` (default) | three-element complex vector

Polarization of the incident electric field, specified as a three-element complex vector containing the Cartesian components of the electric field in V/m. The polarization gives the orientation and magnitude of the electric field.

Example: `Polarization=[0 1 0]`

Data Types: `double`

`SolverType` — Solver for antenna analysis
`'MoM'` (default) | `'FMM'`

Solver for the antenna analysis, specified as one of these values:

- 'MoM' --- Use the method of moments.
- 'FMM' --- Use the fast multipole method.

Example: `SolverType='FMM'`

Data Types: `string`

Object Functions

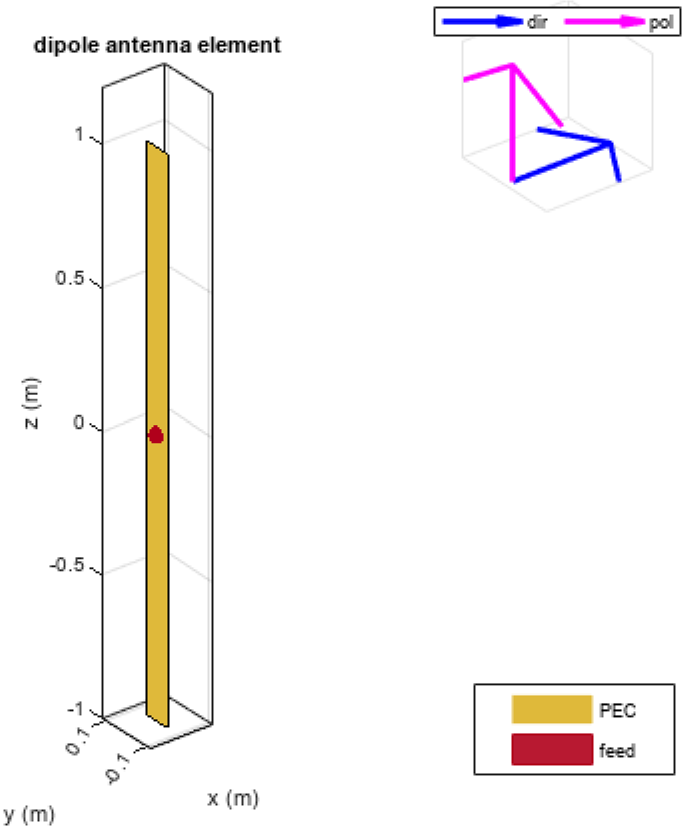
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
doa	Direction of arrival of signal
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
show	Display antenna, array structures or shapes

Examples

Default Plane Wave Excitation

Excite a dipole antenna using a plane wave and view it.

```
h = planeWaveExcitation;
show(h)
```



The blue arrow shows the direction of propagation of the plane wave. The default direction is along the *x*-axis. The pink arrow shows polarization of the plane wave. The default polarization is perpendicular to the direction of propagation. In this case, the polarization is along the *z*-axis.

Feed Current of Antenna Excited by Plane Wave

Excite a dipole antenna using plane wave. Calculate the feed current at 70 MHz.

```
h = planeWaveExcitation
```

```
h = 
  planeWaveExcitation with properties:

         Element: [1×1 dipole]
       Direction: [1 0 0]
    Polarization: [0 0 1]
      SolverType: 'MoM'
```

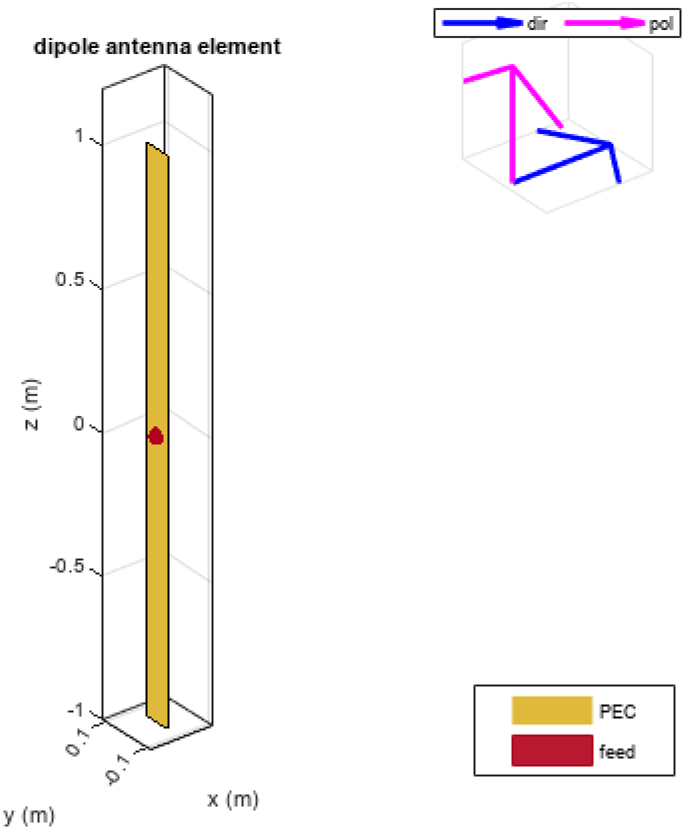
```
cur = feedCurrent(h,70e6)
```

```
cur = 0.0182 - 0.0032i
```

Current Distribution on Antenna

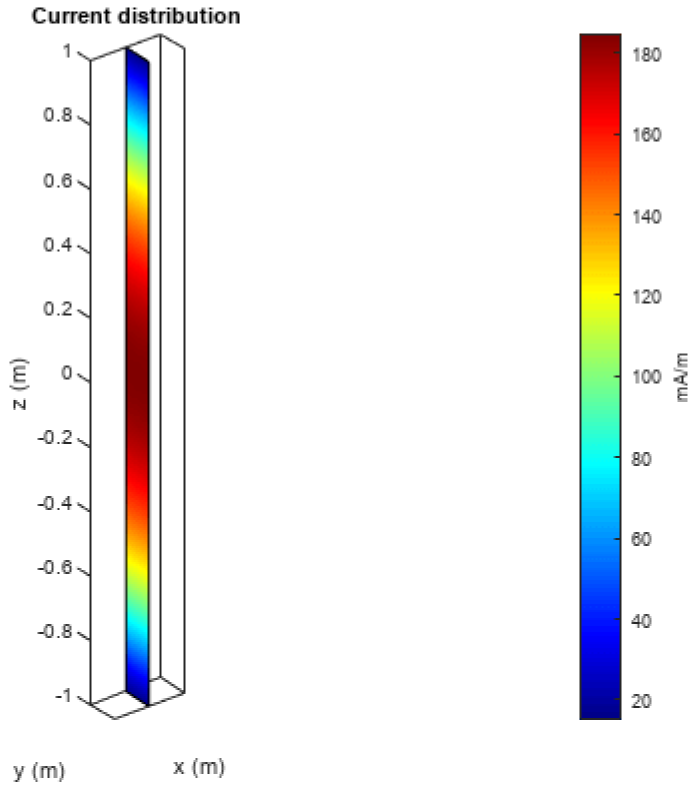
Excite a dipole antenna using a plane wave. The polarization of the wave is along the *z*-axis and the direction of propagation is along the negative *x*-axis. View the antenna.

```
p = planeWaveExcitation(Element=dipole,Direction=[-1 0 0],Polarization=[0 0 1]);
show(p)
```



Plot the current distribution on the dipole antenna at 70 MHz.

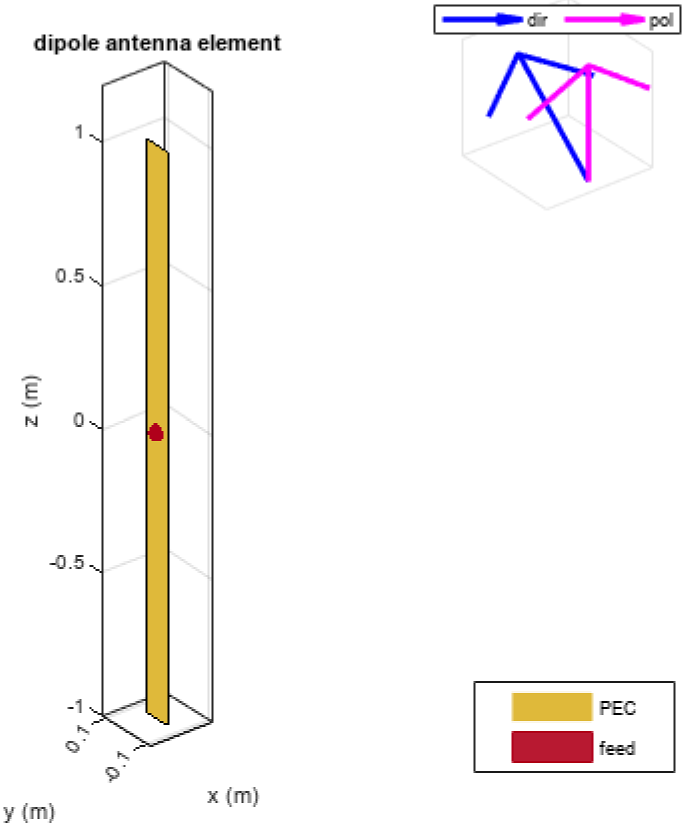
```
current(p,70e6);
```



Antenna Excited by Plane Wave in Arbitrary Direction

Consider a dipole excited by a plane wave.

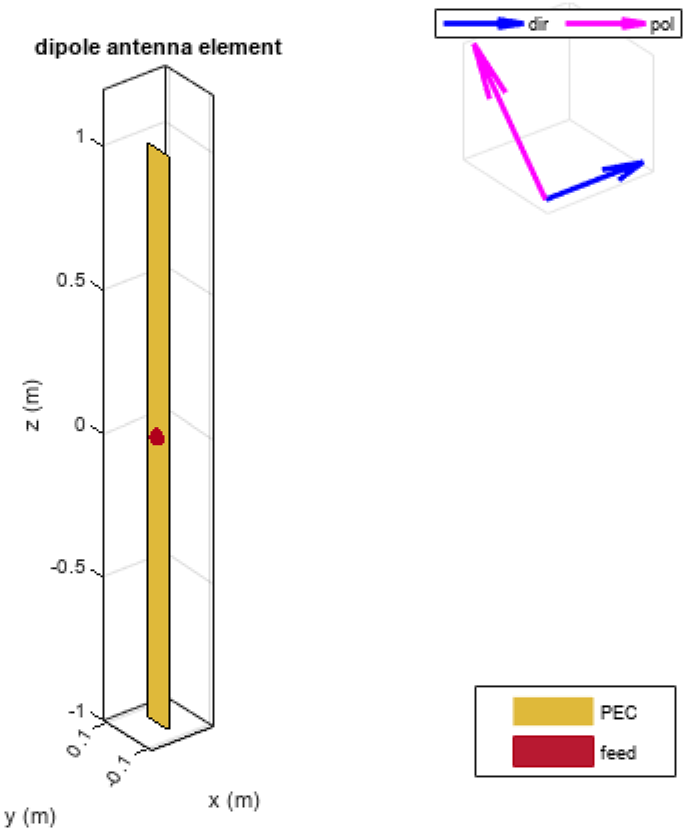
```
p = planeWaveExcitation;
p.Direction = [0 1 1];
show(p)
```



For this antenna, the polarization and direction are not orthogonal to each other and thus any analysis errors out.

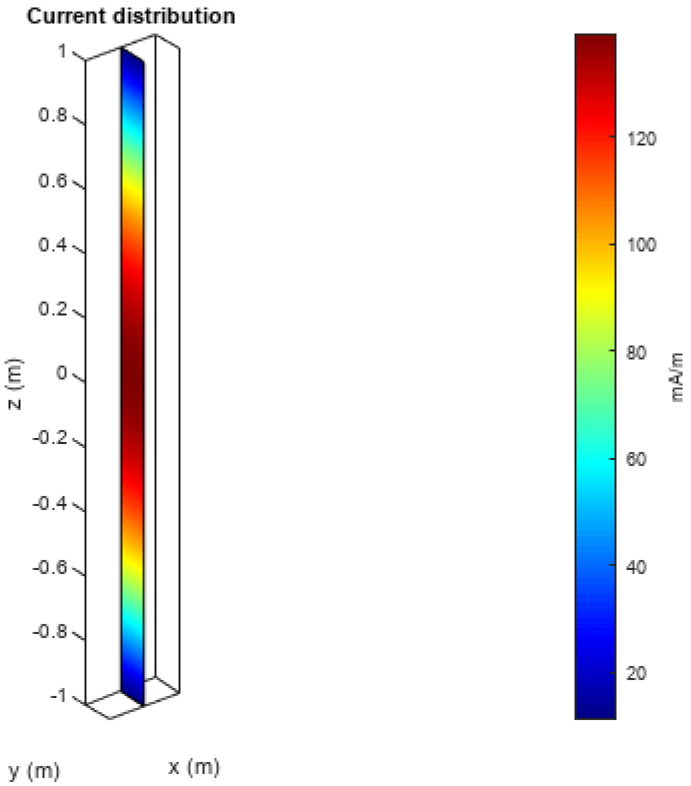
Use the cross-product function to find the appropriate polarization direction of such wave.

```
p = planeWaveExcitation;
p.Polarization = cross(p.Direction,[0 1 1]);
show(p)
```



Calculate the current distribution of the antenna.

```
current(p,75e6);
```



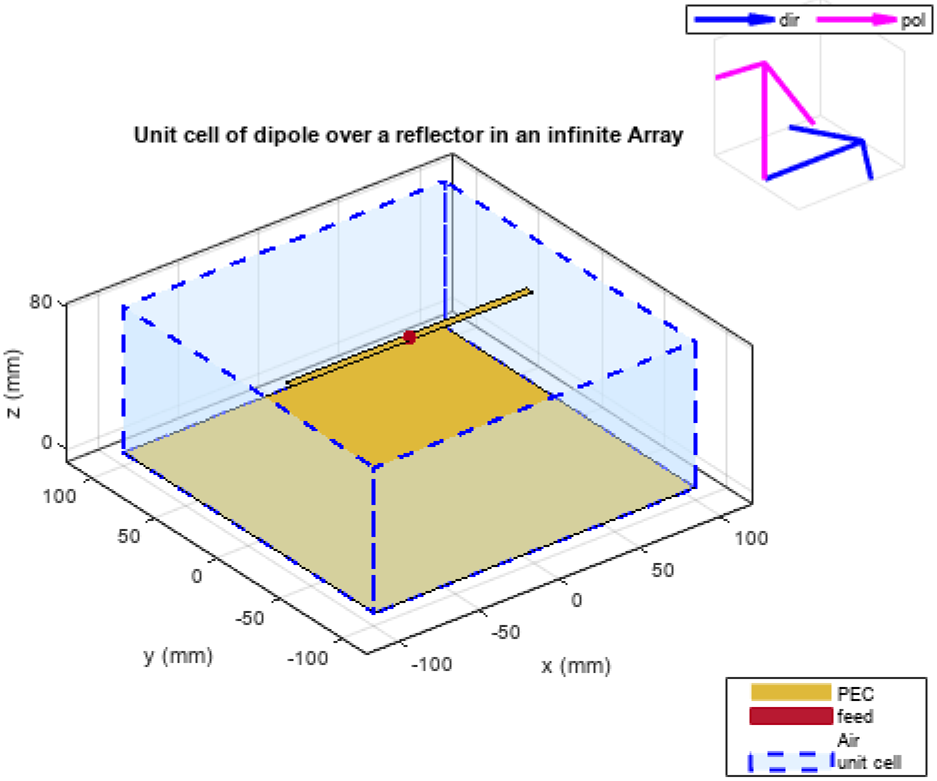
Plane Wave Excitation of Infinite Array

Excite an infinite array using a plane wave.

```
p = planeWaveExcitation(Element=infiniteArray)
```

```
p = 
  planeWaveExcitation with properties:

         Element: [1x1 infiniteArray]
       Direction: [1 0 0]
    Polarization: [0 0 1]
      SolverType: 'MoM'
```

```
show(p)
```



Version History

Introduced in R2017a

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

See Also

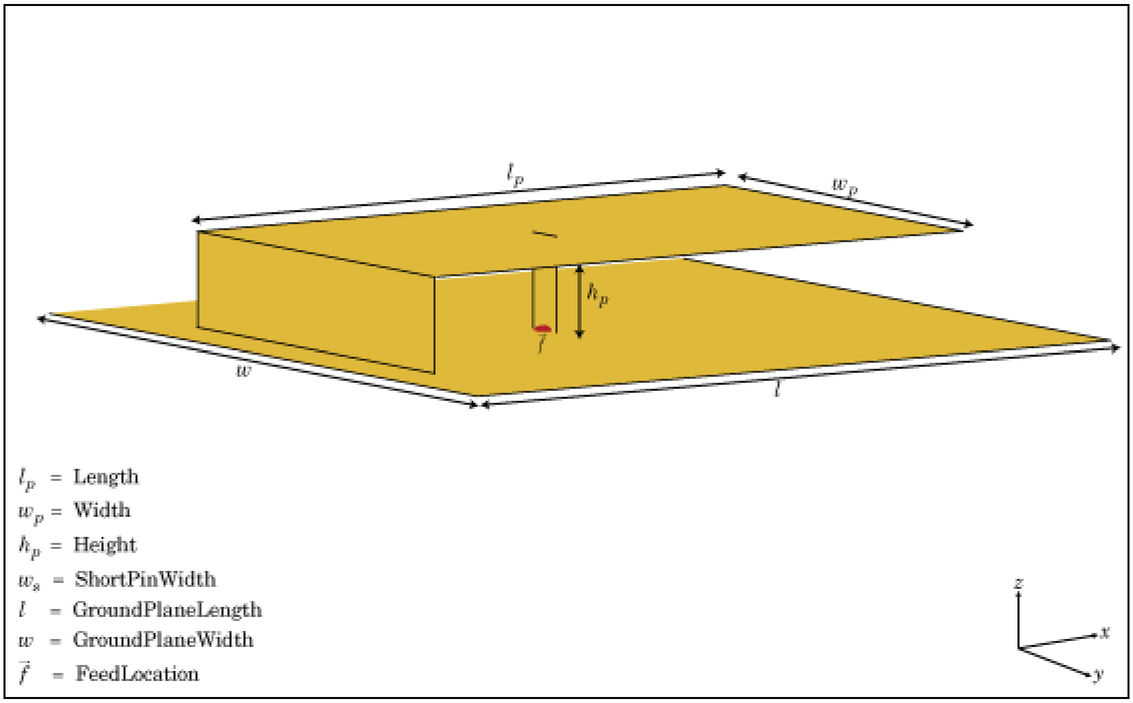
`dipole` | `linearArray`

pifa

Create planar inverted-F antenna

Description

The `pifa` object is a planar inverted-F antenna. The default PIFA antenna is centered at the origin. The feed point is along the length of the antenna.



Creation

Syntax

```
pf = pifa
pf = pifa(Name,Value)
```

Description

`pf = pifa` class to create a planar inverted-F antenna.

`pf = pifa(Name,Value)` class to create a planar inverted-F antenna, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Length` — PIFA antenna length

0.0300 (default) | scalar

PIFA antenna length, specified as a scalar in meters. By default, the length is measured along the *x*-axis.

Example: `'Length',75e-3`

Data Types: `double`

`Width` — PIFA antenna width

0.0200 (default) | scalar

PIFA antenna width, specified as a scalar in meters. By default, the width is measured along the *y*-axis.

Example: `'Width',35e-3`

Data Types: `double`

`Height` — Height of substrate

0.0100 (default) | scalar

Height of the substrate, specified as a scalar in meters.

Example: `'Height',37e-3`

Data Types: `double`

`Substrate` — Type of dielectric material

'Air' (default) | object

Type of dielectric material used as a substrate, specified as an object. For more information see, `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the groundplane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); pf.Substrate = d`

`GroundPlaneLength` — Ground plane length

0.0360 (default) | scalar

Ground plane length, specified as a scalar in meters. By default, ground plane length is measured along the *x*-axis. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width

0.0360 (default) | scalar

Ground plane width, specified as a scalar in meters. By default, ground plane width is measured along the *y*-axis. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',2.5`

Data Types: `double`

`PatchCenterOffset` — Signed distance from center along length and width of ground plane

`[0 0]` (default) | two-element vector

Signed distance from the center along length and width of the ground plane, specified as a two-element vector in meters. Use this property to adjust the location of the patch relative to the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Data Types: `double`

`ShortPinWidth` — Shorting pin width of patch

0.0200 (default) | scalar

Shorting pin width of patch, specified as a scalar in meters. By default, the shorting pin width is measured along the y-axis.

Example: `'ShortPinWidth',3`

Data Types: `double`

`FeedOffset` — Signed distance of feedpoint from origin

[–0.0020 0] (default) | two-element vector

Signed distance from center along length and width of ground plane, specified as a two-element vector. Use this property to adjust the location of the feedpoint relative to ground plane and patch.

Example: `'FeedOffset',[0.01 0.01]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `pf.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array

mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

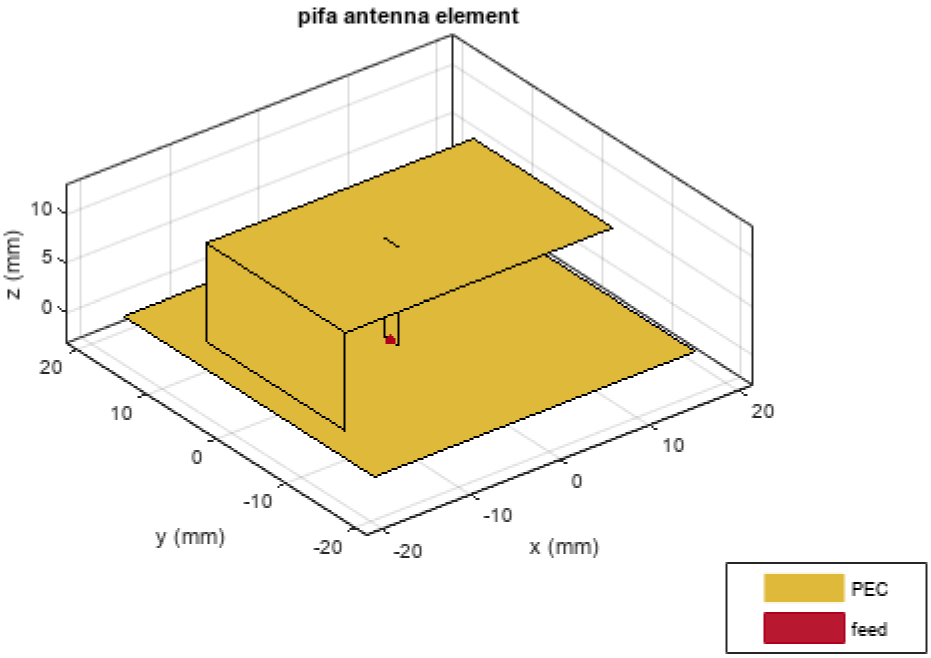
Create and View Planar Inverted-F Antenna(PIFA) Antenna

Create and view a PIFA antenna with 30 mm length, 20 mm width over a 35 mm x 35 mm ground plane, and feedpoint at (-2 mm,0,0).

```
pf = pifa
```

```
pf = 
  pifa with properties:

               Length: 0.0300
                Width: 0.0200
               Height: 0.0100
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.0360
     GroundPlaneWidth: 0.0360
    PatchCenterOffset: [0 0]
        ShortPinWidth: 0.0200
           FeedOffset: [-0.0020 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(pf)
```



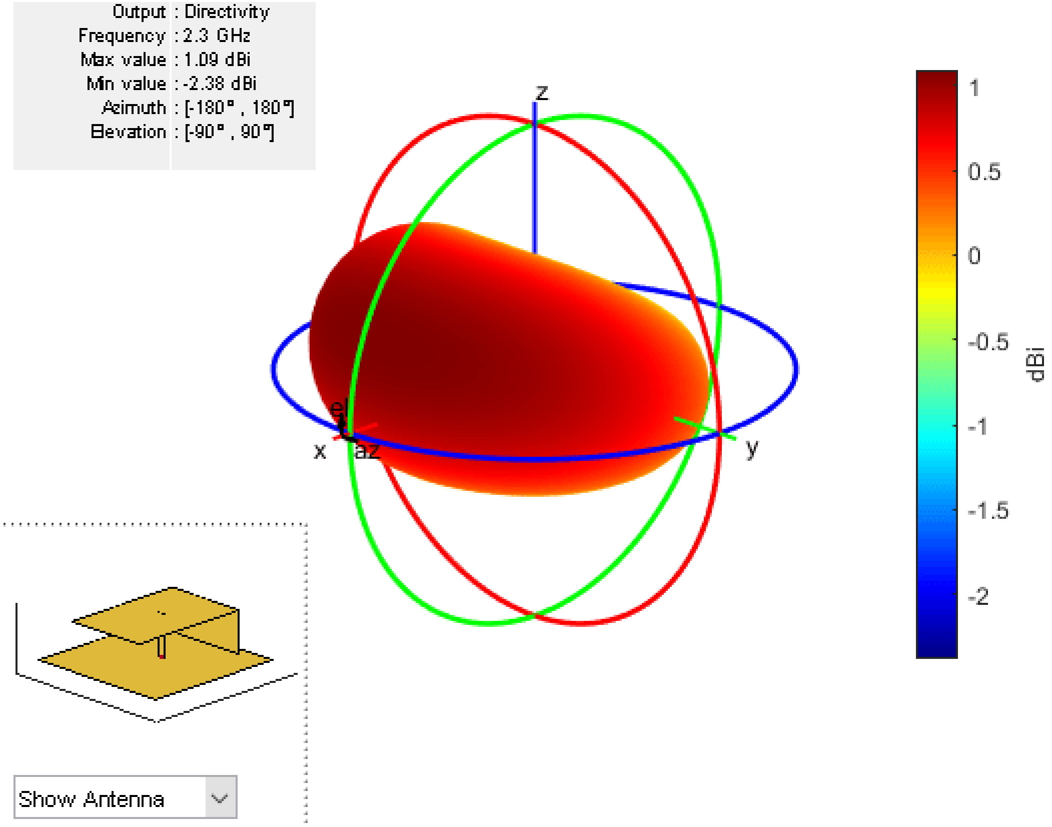
Radiation Pattern of PIFA Antenna

Plot the radiation pattern of a PIFA antenna at a frequency of 2.3 GHz.

```
pf = pifa('Length',30e-3, 'Width',20e-3, 'GroundPlaneLength',35e-3,...
         'GroundPlaneWidth',35e-3)
```

```
pf = 
  pifa with properties:

               Length: 0.0300
                Width: 0.0200
               Height: 0.0100
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.0350
     GroundPlaneWidth: 0.0350
    PatchCenterOffset: [0 0]
        ShortPinWidth: 0.0200
           FeedOffset: [-0.0020 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
pattern(pf,2.3e9);
```



```
    Output : Directivity
 Frequency : 2.3 GHz
 Max value : 1.09 dBi
 Min value : -2.38 dBi
   Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

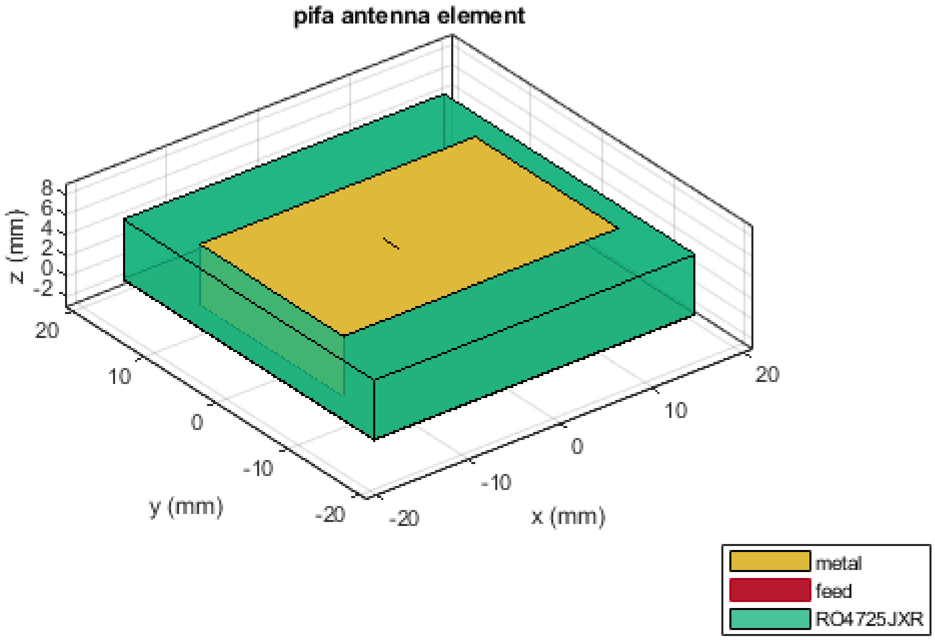
Impedance of PIFA Antenna

Create a PIFA antenna using a dielectric substrate **'RO4725JXR'**.

```
d = dielectric('RO4725JXR');
pf = pifa('Length',30e-3, 'Width',20e-3,'Height',0.0060, 'GroundPlaneLength',35e-3, ...
         'GroundPlaneWidth', 35e-3,'Substrate',d)
show(pf)
```

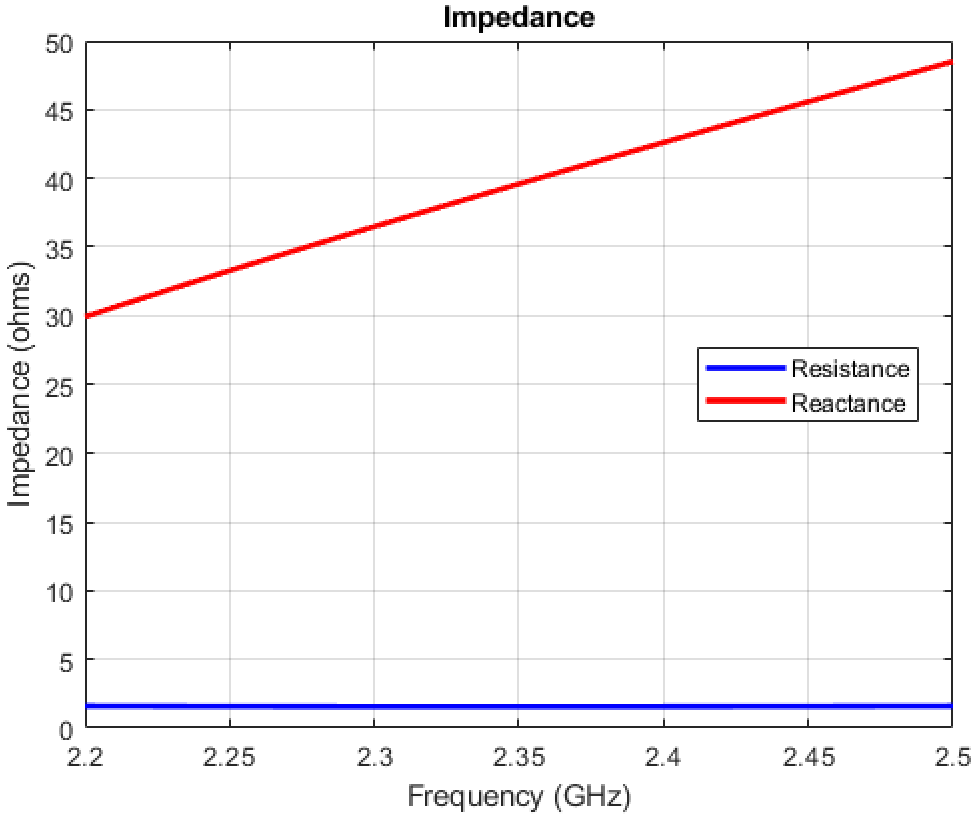
```
pf = 

  pifa with properties:

               Length: 0.0300
                Width: 0.0200
               Height: 0.0060
            Substrate: [1×1 dielectric]
    GroundPlaneLength: 0.0350
     GroundPlaneWidth: 0.0350
    PatchCenterOffset: [0 0]
          ShortPinWidth: 0.0200
           FeedOffset: [-0.0020 0]
                 Tilt: 0
             TiltAxis: [1 0 0]
```

```
                  Load: [1×1 lumpedElement]
```



Calculate the impedance of the antenna over the specified frequency range. GHz.

```
impedance(pf,linspace(2.2e9,2.5e9,31));
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`patchMicrostrip` | `invertedF` | `invertedL`

Topics

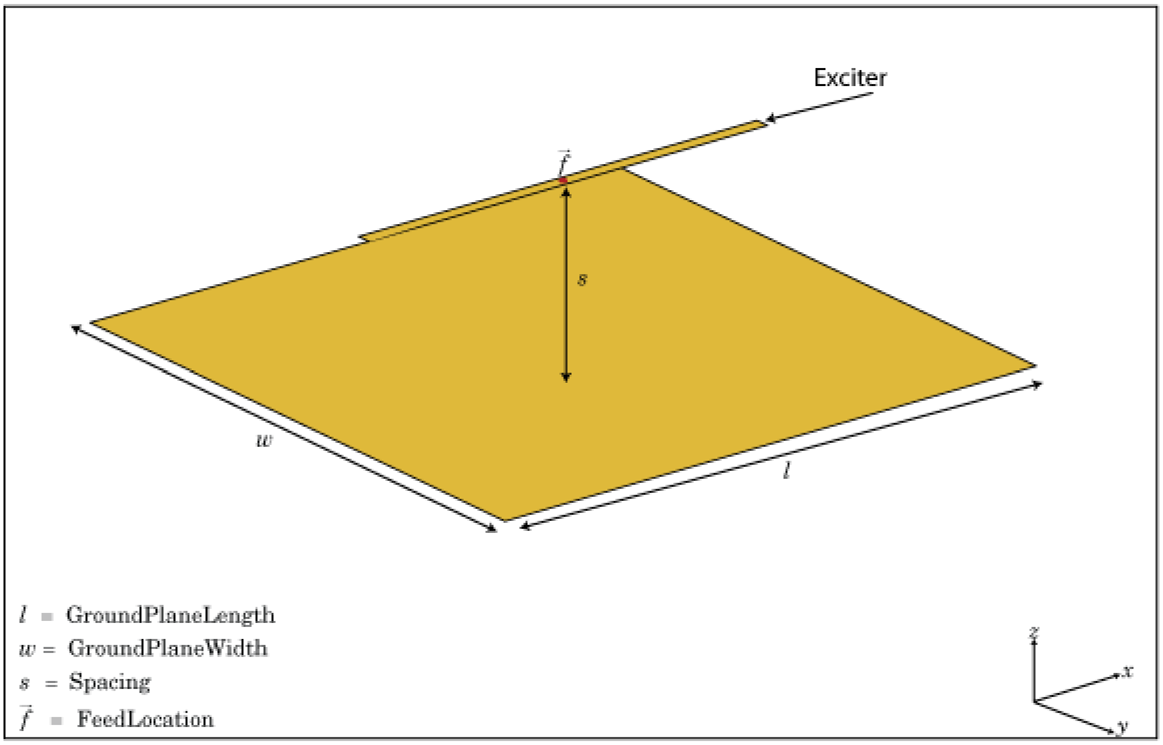
"Rotate Antennas and Arrays"

reflector

Create reflector-backed antenna

Description

The `reflector` object is a reflector-backed antenna on the *xyz*- plane. The default reflector antenna uses a dipole as an exciter. The feed point is on the exciter.



Creation

Syntax

```
rf = reflector
rf = reflector(Name,Value)
```

Description

`rf = reflector` creates a reflector backed antenna located in the X-Y-Z plane. By default, dimensions are chosen for an operating frequency of 1 GHz.

`rf = reflector(Name,Value)` creates a reflector backed antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Exciter` — Antenna or array type used as exciter

`dipole` (default) | `antenna` object | `array` object

Antenna type used as an exciter, specified as any single-element antenna object. Except reflector and cavity antenna elements, you can use any of the antenna elements or array elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',horn`

Example: `ant.Exciter = horn`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Substrate` — Type of dielectric material

'Air' (default) | object

Type of dielectric material used as a substrate, specified as an object. For more information see, `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the groundplane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); rf.Substrate = d`

`GroundPlaneLength` — Reflector length along *x*-axis

0.2000 (default) | scalar

Reflector length along the *x*-axis, specified a scalar in meters. By default, ground plane length is measured along the *x*-axis. Setting `'GroundPlaneLength'` to`Inf`, uses the infinite ground plane technique for antenna analysis. You can also set the `'GroundPlaneLength'` to zero.

Example: `'GroundPlaneLength',3`

Data Types: `double`

`GroundPlaneWidth` — Reflector width along *y*-axis

0.2000 (default) | scalar

Reflector width along the *y*-axis, specified as a scalar in meters. By default, ground plane width is measured along the y-axis. Setting `'GroundPlaneWidth'` to`Inf`, uses the infinite ground plane technique for antenna analysis. You can also set the `'GroundPlaneWidth'` to zero.

Example: `'GroundPlaneWidth',2.5`

Data Types: `double`

`Spacing` — Distance between reflector and exciter

0.0750 (default) | scalar

Distance between the reflector and the exciter, specified as a scalar in meters. By default, the exciter is placed along the x-axis.

Example: `'Spacing',7.5e-2`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `rf.Load = lumpedElement('Impedance',75)`

`EnableProbeFeed` — Create probe feed from backing structure to exciter

`0` (default) | `1`

Create probe feed from backing structure to exciter, specified as `0` or `1`. By default, probe feed is not enabled.

Example: `'EnableProbeFeed',1`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

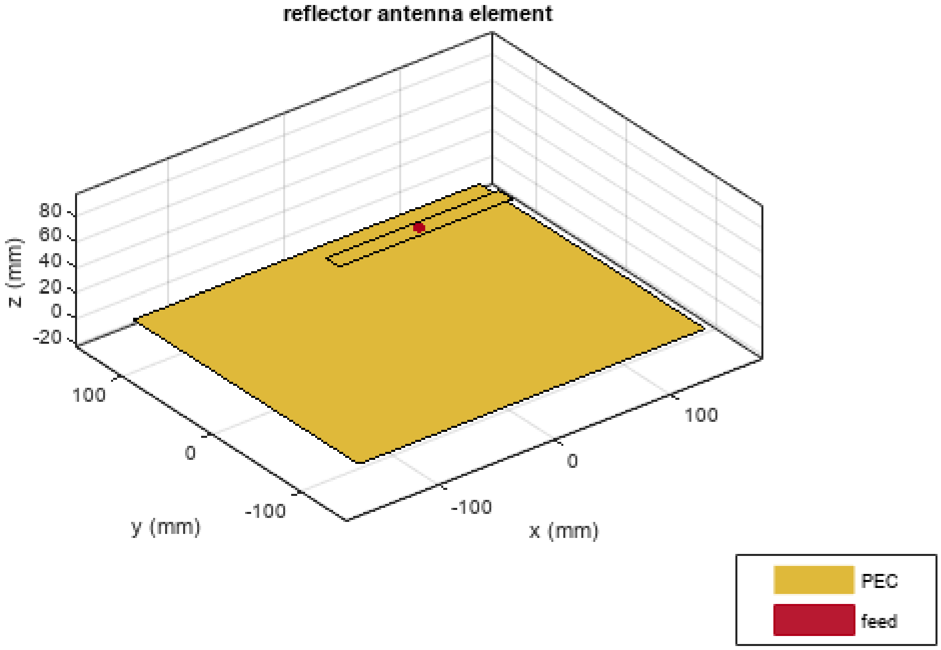
Create and View Reflector-Backed Dipole Antenna

Create a reflector backed dipole that has 30 cm length, 25 cm width and spaced 7.5 cm from the dipole for operation at 1 GHz.

```
d = dipole('Length',0.15,'Width',0.015, 'Tilt',90,'TiltAxis',[0 1 0]);
rf = reflector('GroundPlaneLength',30e-2, 'GroundPlaneWidth',25e-2,...
              'Spacing',7.5e-2);
rf.Exciter = d
```

```
rf = 
  reflector with properties:

              Exciter: [1x1 dipole]
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.3000
     GroundPlaneWidth: 0.2500
              Spacing: 0.0750
      EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

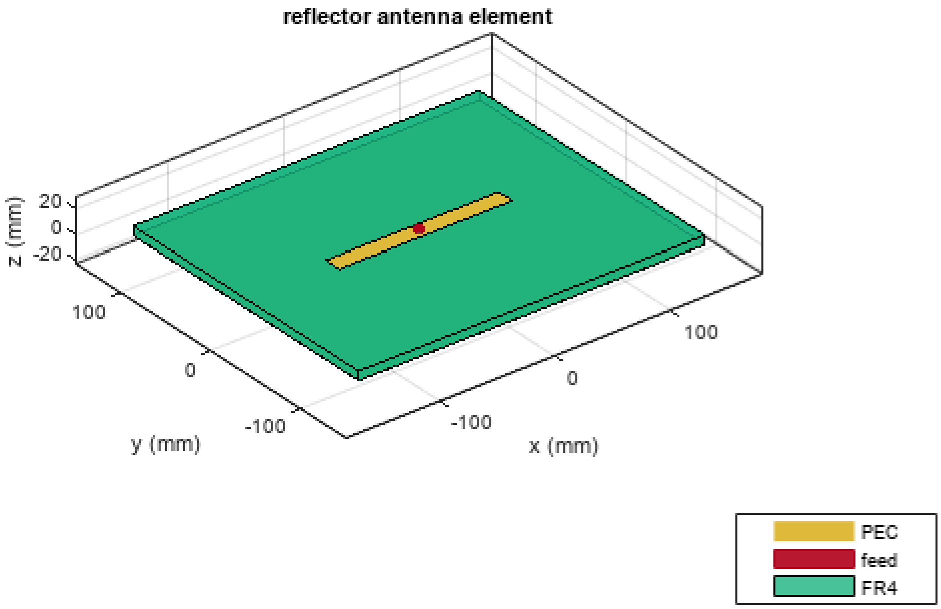
```
show(rf)
```



Radiation Pattern of Reflector Backed Antenna

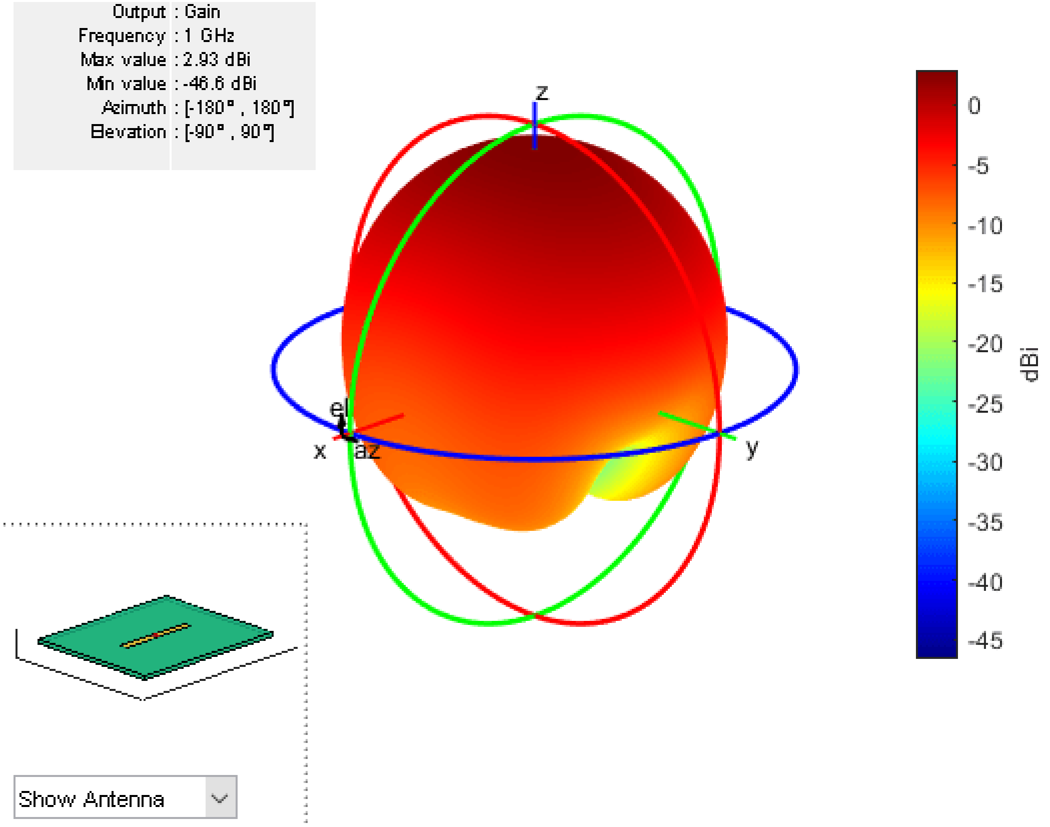
Create a reflector backed dipole antenna using a dielectric substrate **'FR4'**.

```
d = dielectric('FR4');
di = dipole('Length',0.15,'Width',0.015, 'Tilt',90,'TiltAxis','Y');
rf = reflector('GroundPlaneLength',30e-2, 'GroundPlaneWidth',25e-2, ...
               'Spacing',7.5e-3,'Substrate',d);
rf.Exciter = di;
show(rf)
```



Plot the radiation pattern of the antenna at a frequency of 1 GHz.

```
figure
pattern(rf,1e9)
```



Output : Gain
Frequency : 1 GHz
Max value : 2.93 dBi
Min value : -46.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

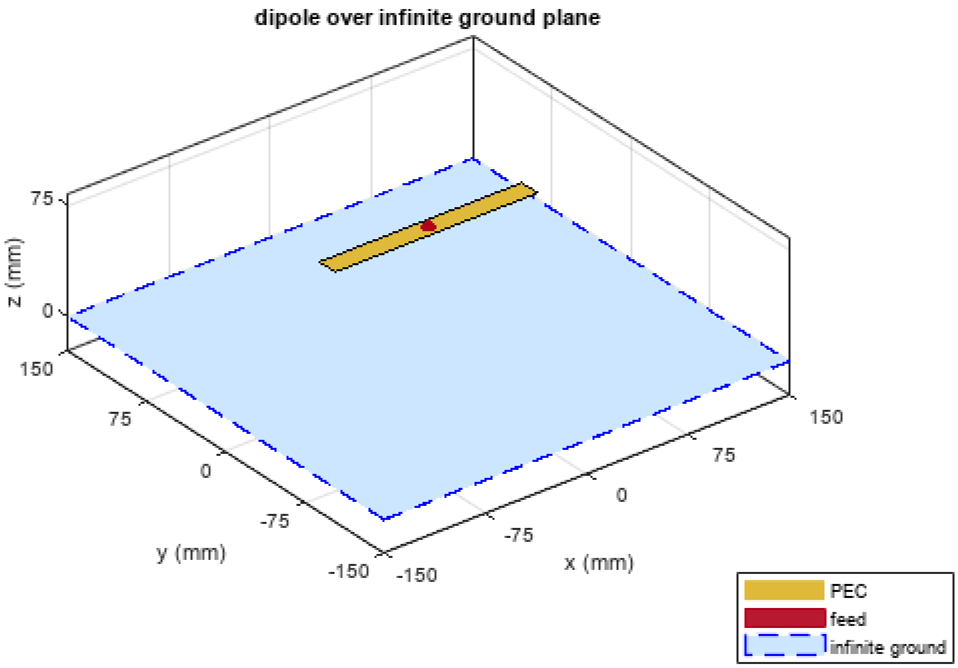
Create Reflector-Backed Antenna Over Infinite Ground Plane

Create a reflector backed dipole that has infinite length, 25 cm width and spaced 7.5 cm from the dipole for operation at 1 GHz.

```
d = dipole('Length',0.15,'Width',0.015, 'Tilt',90,'TiltAxis',[0 1 0]);
rf = reflector('GroundPlaneLength',inf, 'GroundPlaneWidth',25e-2,...
              'Spacing',7.5e-2);
rf.Exciter = d
```

```
rf =
  reflector with properties:

              Exciter: [1x1 dipole]
            Substrate: [1x1 dielectric]
    GroundPlaneLength: Inf
     GroundPlaneWidth: 0.2500
              Spacing: 0.0750
      EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(rf)
```



Antenna On Dielectric Substrate - Compare Gain Values

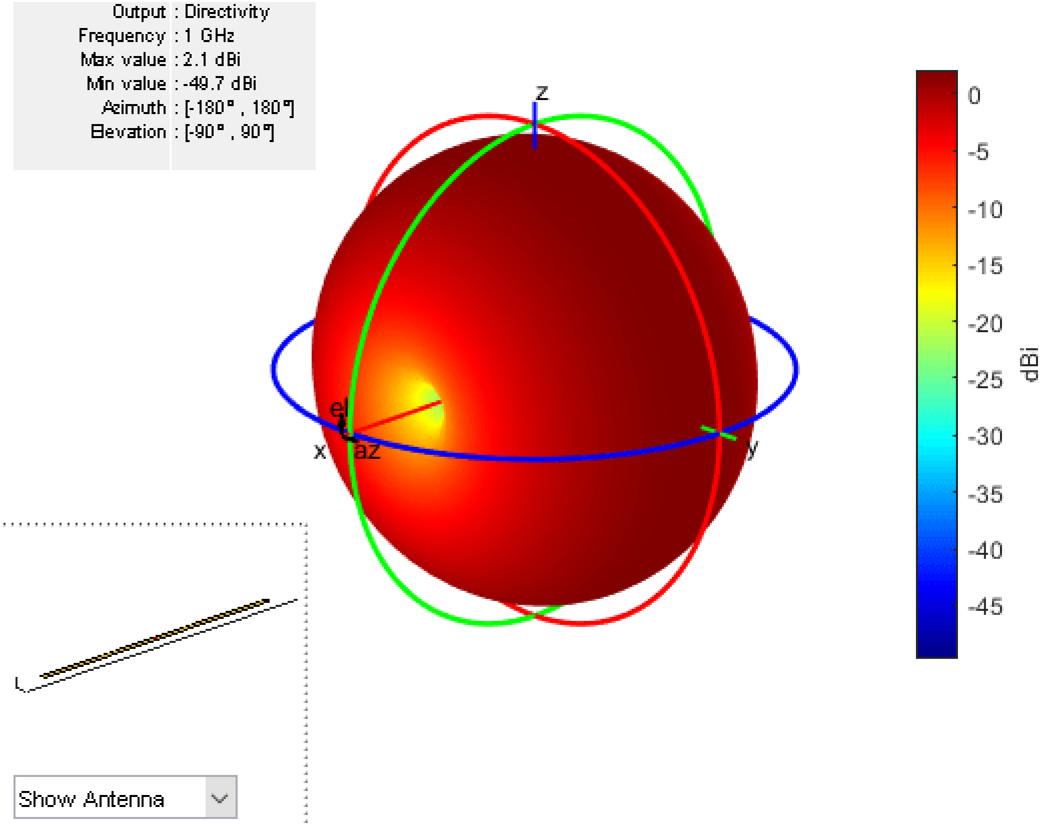
Compare the gain values of a dipole antenna in free space and dipole antenna on a substrate.

Design a dipole antenna at a frequency of 1 GHz.

```
d = design(dipole,1e9);
l_by_w = d.Length/d.Width;
d.Tilt = 90;
d.TiltAxis = [0 1 0];
```

Plot the radiation pattern of the dipole in free space at 1 GHz.

```
figure
pattern(d,1e9);
```



```
Output    : Directivity
Frequency : 1 GHz
Max value : 2.1 dBi
Min value : -49.7 dBi
Azimuth   : [-180° , 180°]
Elevation : [-90° , 90°]
```

Use FR4 as the dielectric substrate.

```
t = dielectric('FR4')
```

```
t = 
  dielectric with properties:

           Name: 'FR4'
       EpsilonR: 4.8000
    LossTangent: 0.0260
      Thickness: 0.0060

For more materials see catalog
```

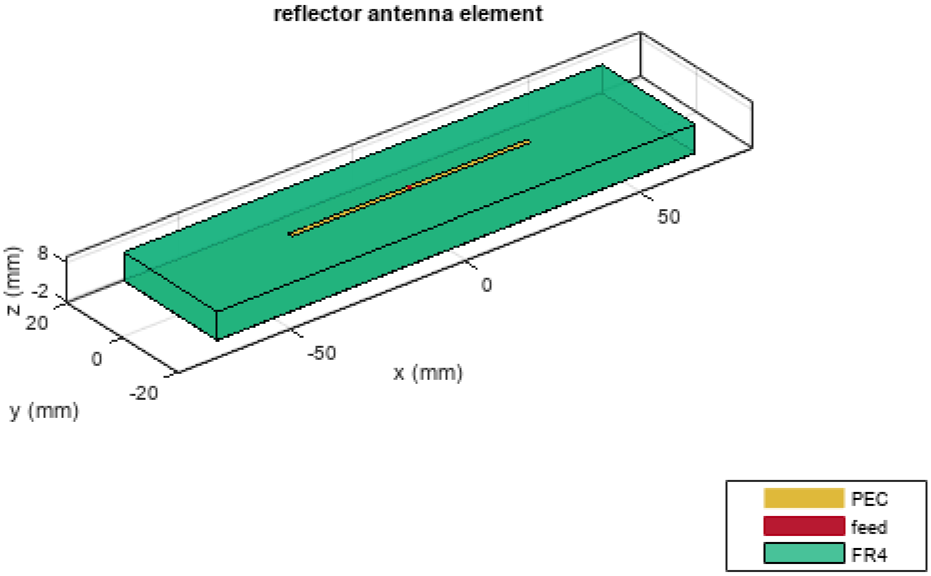
```
eps_r = t.EpsilonR;
lambda_0 = physconst('lightspeed')/1e9;
lambda_d = lambda_0/sqrt(eps_r);
```

Adjust the length of the dipole based on the wavelength.

```
d.Length = lambda_d/2;
d.Width = d.Length/l_by_w;
```

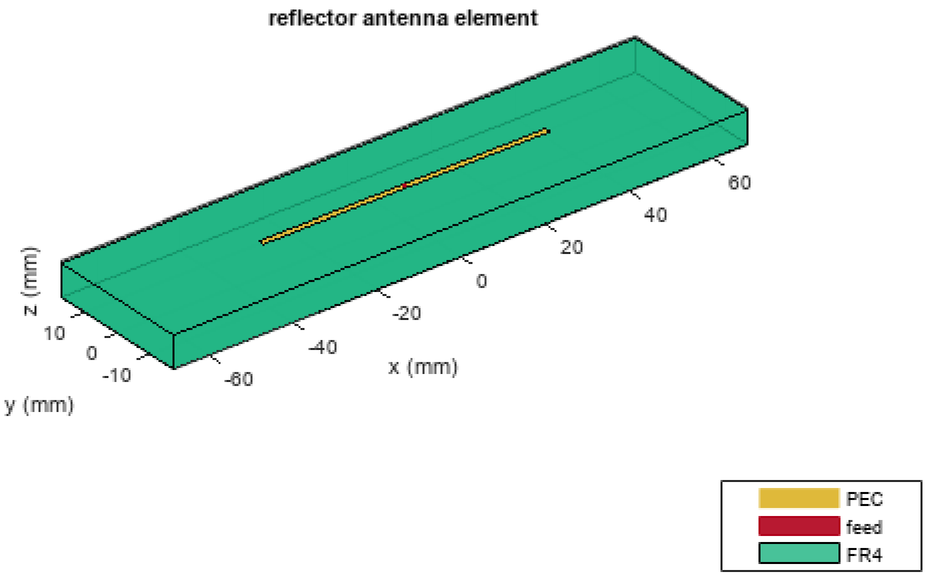
Design a reflector at 1 GHz with the dipole as the exciter and FR4 as the substrate.

```
rf = reflector('Exciter',d,'Spacing',7.5e-3,'Substrate',t);
rf.GroundPlaneLength = lambda_d;
rf.GroundPlaneWidth = lambda_d/4;
figure
show(rf)
```



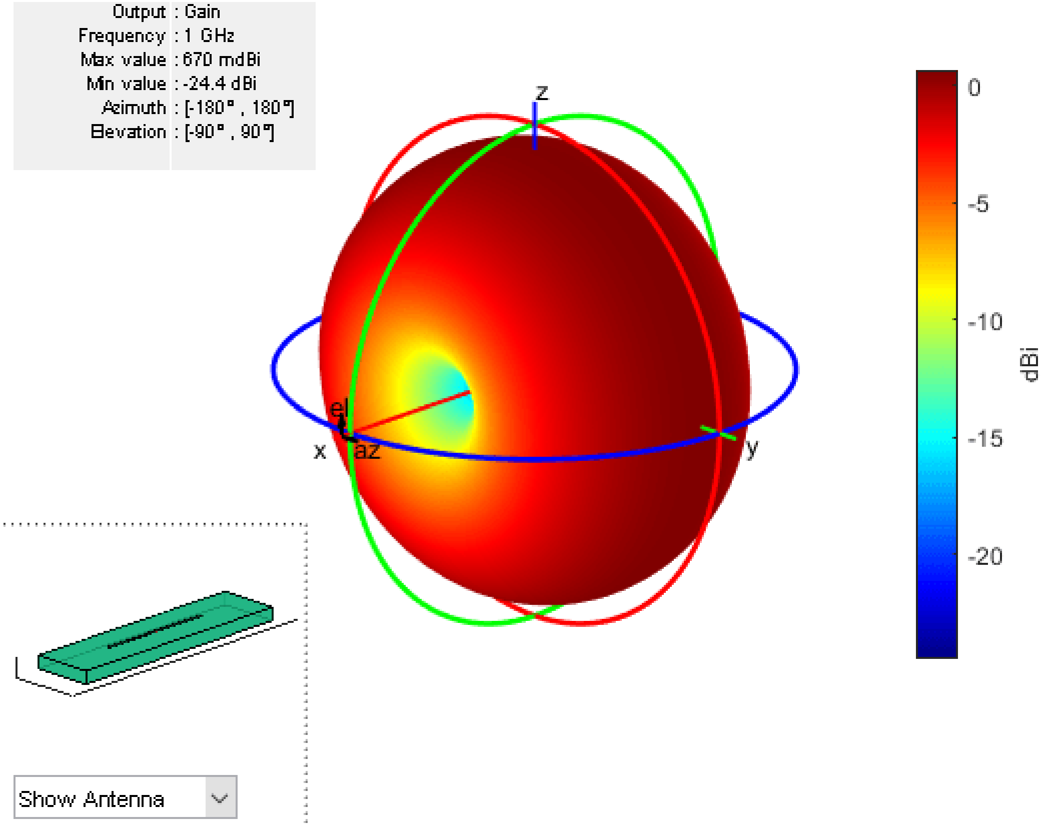
Remove the groundplane for plotting the gain of the dipole on the substrate.

```
rf.GroundPlaneLength = 0;
show(rf)
```



Plot the radiation pattern of the dipole on the substrate at 1 GHz.

```
figure
pattern(rf,1e9);
```



```
   Output : Gain
Frequency : 1 GHz
Max value : 670 mdBi
Min value : -24.4 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Compare the gain values.

- Gain of the dipole in free space = 2.11 dBi
- Gain of the dipole on substrate = 1.93 dBi

Create Reflector-Backed Rectangular Array of Bowtie Antennas

Create a rectangular array of the bowtie antennas.

```
b = bowtieTriangular('Length',0.05)
```

```
b = 
  bowtieTriangular with properties:

        Length: 0.0500
    FlareAngle: 90
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
rectArr = rectangularArray('Element',b,'RowSpacing',0.18,'ColumnSpacing',0.18)
```

```
rectArr = 
  rectangularArray with properties:
```

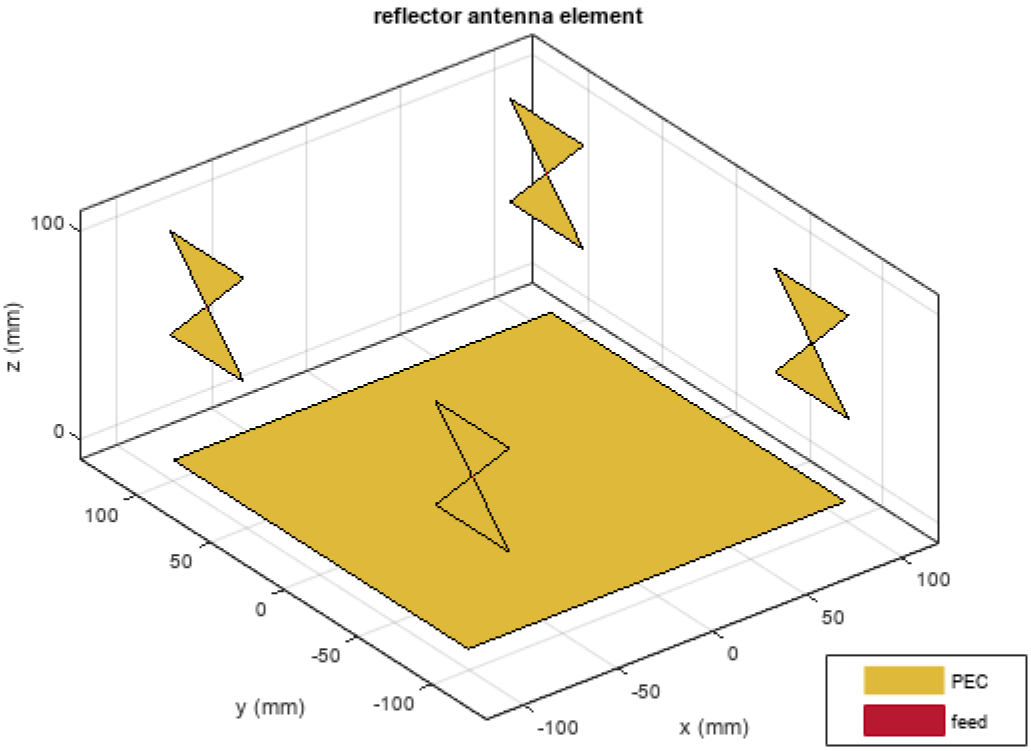
```
             Element: [1x1 bowtieTriangular]
                Size: [2 2]
          RowSpacing: 0.1800
       ColumnSpacing: 0.1800
             Lattice: 'Rectangular'
      AmplitudeTaper: 1
          PhaseShift: 0
                Tilt: 0
            TiltAxis: [1 0 0]
```

Create a rectangular array with reflector backing structure.

```
ant = reflector('Exciter',rectArr)
```

```
ant = 
  reflector with properties:

               Exciter: [1x1 rectangularArray]
             Substrate: [1x1 dielectric]
     GroundPlaneLength: 0.2000
      GroundPlaneWidth: 0.2000
               Spacing: 0.0750
       EnableProbeFeed: 0
             Conductor: [1x1 metal]
                  Tilt: 0
              TiltAxis: [1 0 0]
                  Load: [1x1 lumpedElement]
```

```
show(ant)
```



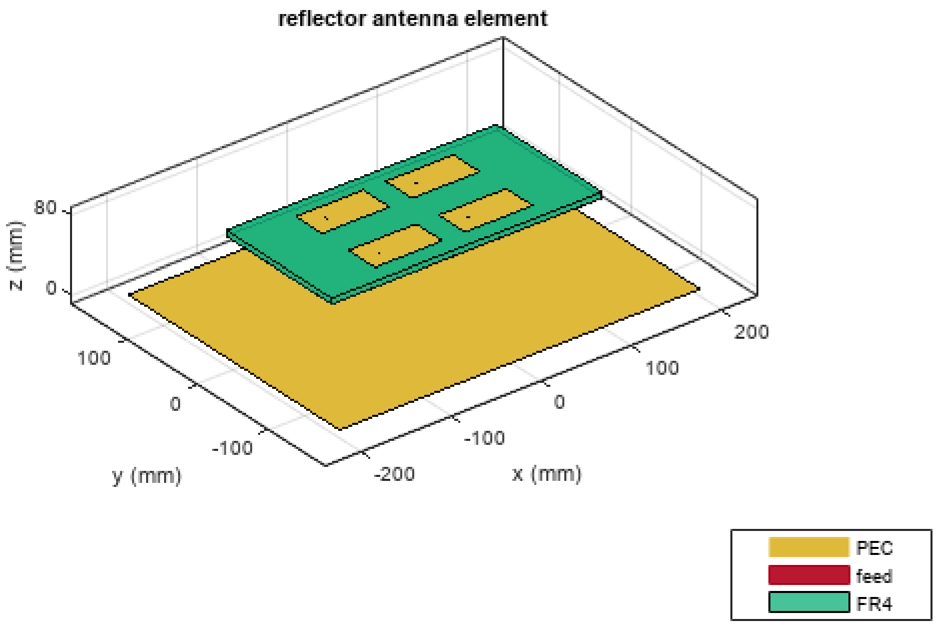
Create Rectangular Array of Microstrip Patch with Reflector Backing Structure

Create a reflector-backed rectangular array of microstrip patch antennas.

```
p = patchMicrostrip('Substrate',dielectric('FR4'));
ra = rectangularArray('Element',p,'RowSpacing',0.075,'ColumnSpacing',0.1);
ant = reflector('Exciter',ra,'GroundPlaneLength',0.4,'GroundPlaneWidth',0.3)
```

```
ant = 
  reflector with properties:

              Exciter: [1x1 rectangularArray]
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.4000
     GroundPlaneWidth: 0.3000
              Spacing: 0.0750
      EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`spiralArchimedean` | `spiralEquiangular` | `cavity`

Topics

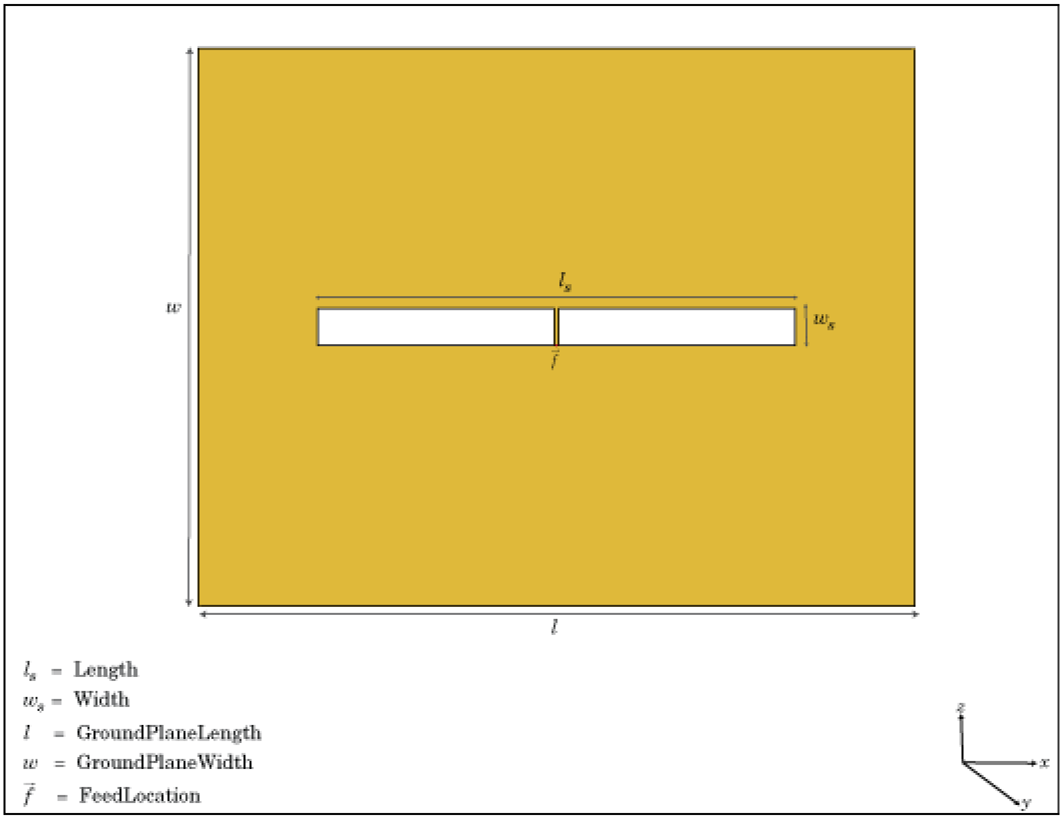
"Rotate Antennas and Arrays"

slot

Create rectangular slot antenna on ground plane

Description

The `slot` object is a rectangular slot antenna on a ground plane. The default slot has its first resonance at 130 MHz.



Creation

Syntax

```
s = slot
s = slot(Name,Value)
```

Description

`s = slot` creates a rectangular slot antenna on a ground plane.

`s = slot(Name,Value)` creates a rectangular slot antenna, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain default values.

Properties

`Length` — Slot length

1 (default) | scalar

Slot length, specified as a scalar in meters.

Example: `'Length',2`

Data Types: `double`

`Width` — Slot width

0.1000 (default) | scalar

Slot width, specified a scalar in meters.

Example: `'Width',0.02`

Data Types: `double`

`SlotCenter` — Slot antenna center

`[0 0 0]` (default) | three-element vector in Cartesian coordinates

Slot antenna center, specified as a three-element vector in Cartesian coordinates.

Example: `'SlotCenter',[8 0 0]`

Data Types: `double`

`GroundPlaneLength` — Ground plane length

1.5000 (default) | scalar

Ground plane length, specified as a scalar in meters. By default, the length is measured along the *x*-axis.

Example: `'GroundPlaneLength',3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width

1.5000 (default) | scalar

Ground plane width, specified as a scalar in meters. By default, the width is measured along the *y*-axis.

Example: `'GroundPlaneWidth',4`

Data Types: `double`

`FeedOffset` — Distance from center along *x*-axis

0 (default) | scalar

Distance from center along *x*-axis, specified as a scalar in meters. Offset from slot center is measured along the length.

Example: `'FeedOffset',3`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `s.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

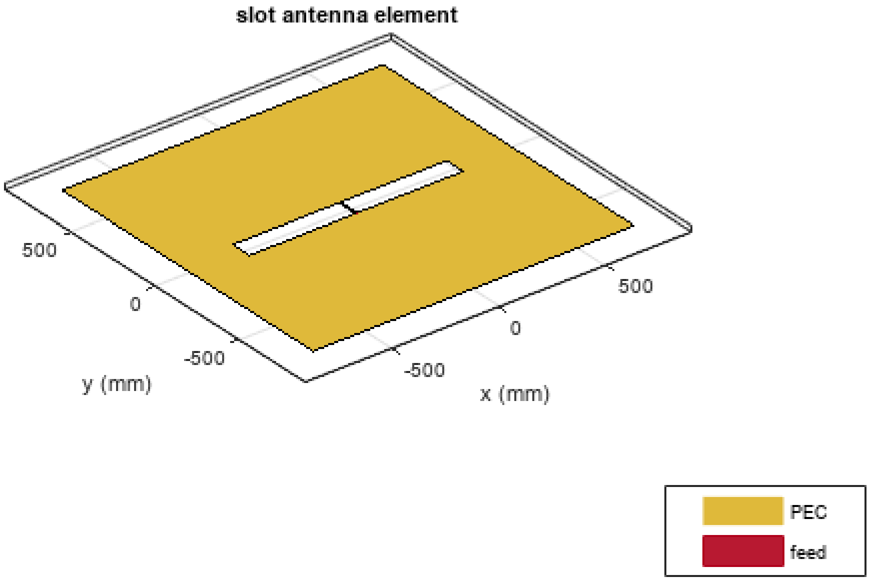
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Slot Antenna

Create and view a slot antenna that has 1 m length and 100 mm width.

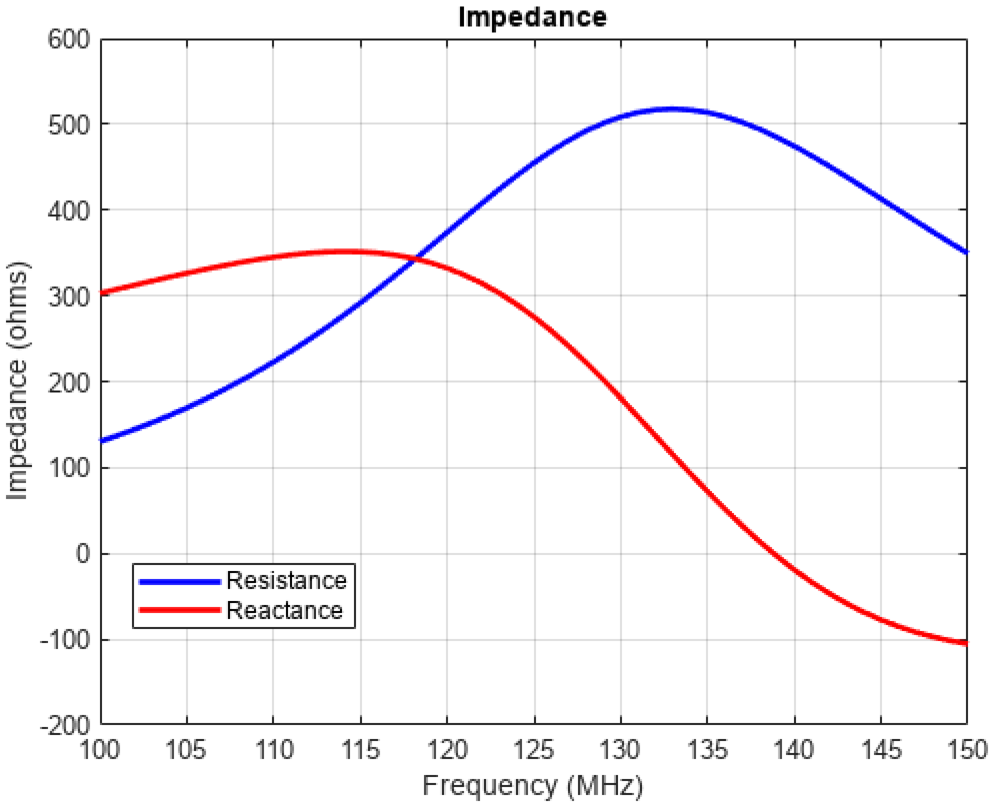
```
s = slot('Length',1,'Width',0.1);
show(s)
```



Impedance of Slot Antenna

Calculate and plot the impedance of a slot antenna over a frequency range of 100-150 MHz.

```
s = slot('Length',1,'Width',0.1);
impedance(s,linspace(100e6,150e6,51));
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`pifa` | `vivaldi` | `yagiUda`

Topics

"Rotate Antennas and Arrays"

spiralArchimedean

Create Archimedean spiral antenna

Description

The `spiralArchimedean` object creates a planar Archimedean spiral antenna on the *xy*- plane. The default Archimedean spiral is always center fed and has two arms. The field characteristics of this antenna are frequency independent. A realizable spiral has finite limits on the feeding region and the outermost point of any arm of the spiral. The spiral antenna exhibits a broadband behavior. The outer radius imposes the low frequency limit and the inner radius imposes the high frequency limit. The arm radius grows linearly as a function of the winding angle.

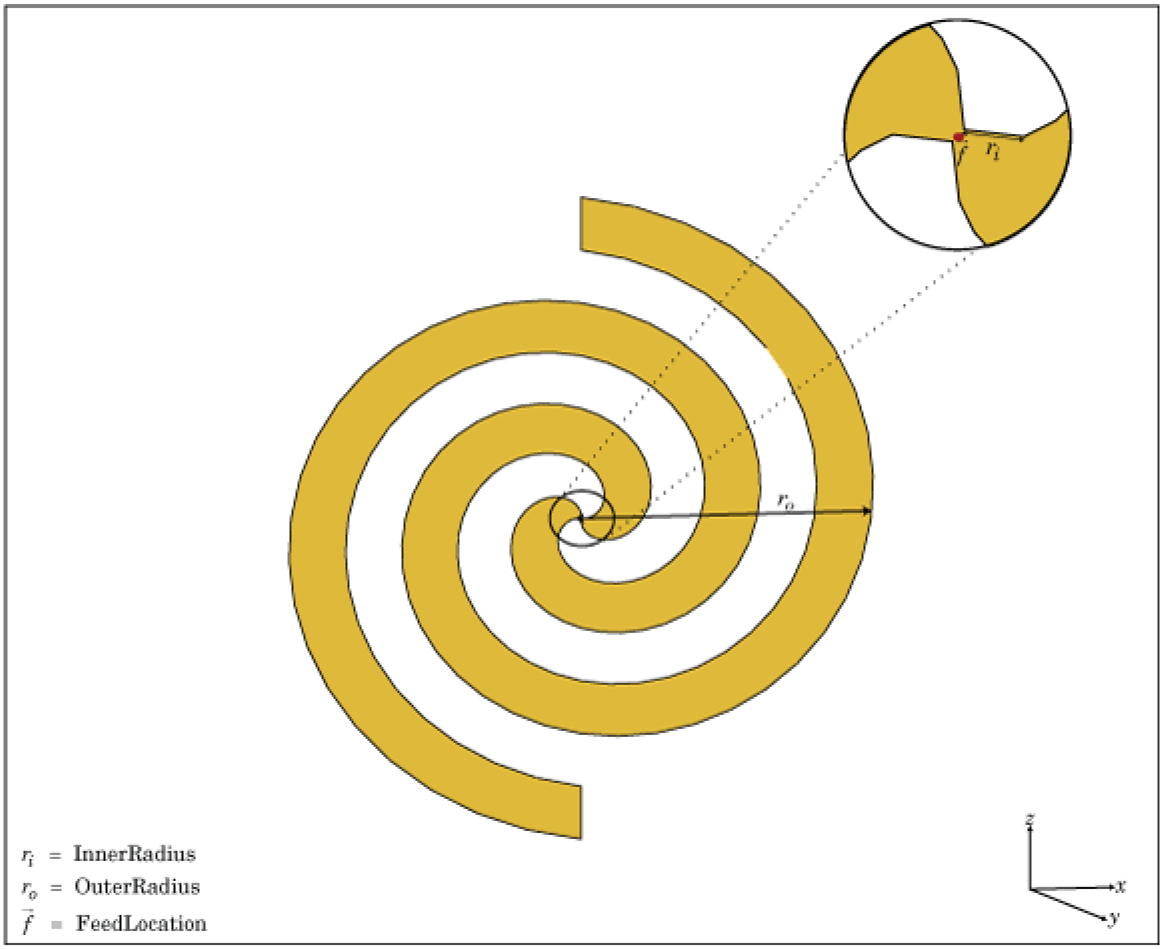
The equation of the Archimedean spiral is:

$$r = r_0 + a\phi$$

where:

- r_0 is the inner radius
- a is the growth rate
- ϕ is the winding angle of the spiral

Archimedean spiral antenna is a self-complementary structure, where the spacing between the arms and the width of the arms are equal. The default antenna is center fed. The feed point coincides with the origin. The origin is in the *xy*- plane.



Creation

Syntax

```
ant = spiralArchimedean
ant = spiralArchimedean(Name,Value)
```

Description

`ant = spiralArchimedean` creates a planar Archimedean spiral on the X-Y plane. By default, the antenna operates over a broadband frequency range of 3–5 GHz.

`ant = spiralArchimedean(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = spiralArchimedean('Turns',6.25)` creates a Archimedean spiral of 6.25 turns.

Output Arguments

ant — MATLAB object
scalar `spiralArchimedean` object (default)

MATLAB object, returned as scalar `spiralArchimedean` object.

Properties

`NumArms` — Number of arms

2 (default) | scalar integer

Number of arms, specified as a scalar integer. You can also create a single arm Archimedean spiral by specifying `NumArms` is equal to one.

Example: `'NumArms',1`

Example: `ant.NumArms = 1`

Data Types: `double`

`Turns` — Number of turns of spiral antenna

1.5000 (default) | scalar

Number of turns of the spiral antenna, specified as a scalar.

Example: `'Turns',2`

Example: `ant.Turns = 2`

Data Types: `double`

`InnerRadius` — Inner radius of spiral antenna

5.0000e-04 (default) | scalar

inner radius of the spiral antenna, specified as a scalar in meters.

Example: `'InnerRadius',1e-3`

Example: `ant.InnerRadius = 1e-3`

Data Types: `double`

`OuterRadius` — Outer radius of spiral antenna

0.0398 (default) | scalar

Outer radius of the spiral antenna, specified as a scalar in meters.

Example: `'OuterRadius',1e-3`

Example: `ant.OuterRadius = 1e-3`

Data Types: `double`

`WindingDirection` — Direction of spiral turns (windings)

`'CW'` | `'CCW'`

Direction of the spiral turns (windings), specified as `'CW'` or `'CCW'`.

Example: `'WindingDirection'`,`'CW'`

Example: `ant.WindingDirection = CW`

Data Types: `char` | `string`

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the spiral antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, it is at the origin. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: ant.Load = lumpedElement('Impedance',75)

Data Types: double

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: TiltAxis=[0 1 0]

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

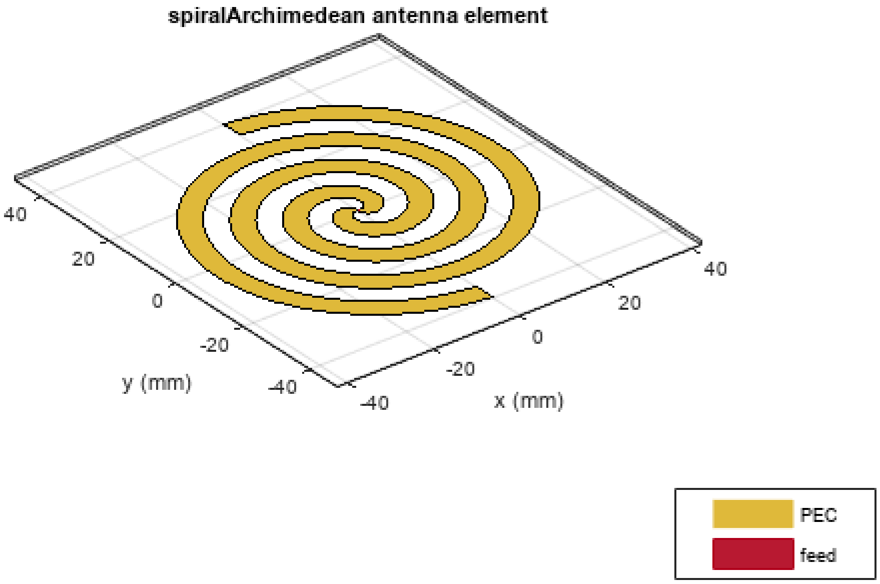
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Archimedean Spiral Antenna

Create and view a 2-turn Archimedean spiral antenna with a 1 mm starting radius and 40 mm outer radius.

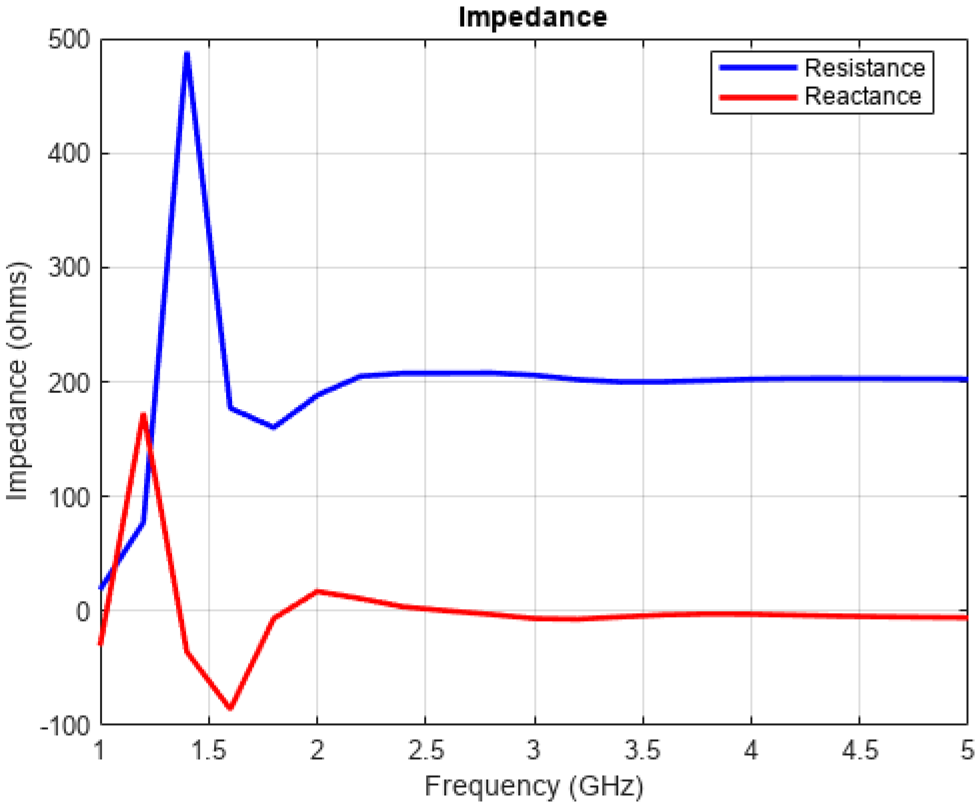
```
sa = spiralArchimedean('Turns',2, 'InnerRadius',1e-3, 'OuterRadius',40e-3);
show(sa)
```



Impedance of Archimedean Spiral Antenna

Calculate the impedance of an Archimedean spiral antenna over a frequency range of 1-5 GHz.

```
sa = spiralArchimedean('Turns',2, 'InnerRadius',1e-3, 'OuterRadius',40e-3);
impedance(sa, linspace(1e9,5e9,21));
```



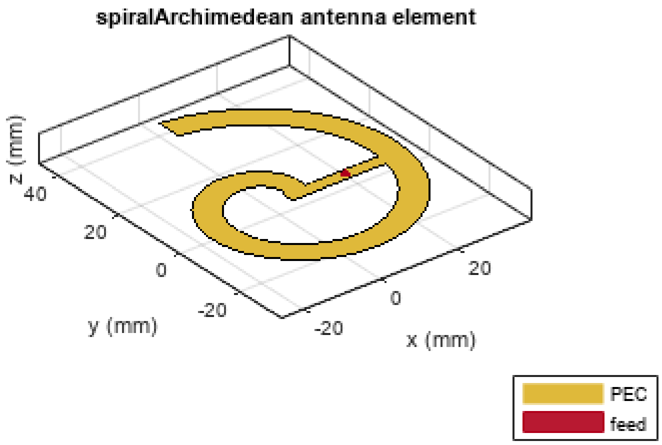
Single-Arm Archimedean Spiral

Create and view a single-arm Archimedean spiral.

```
ant = spiralArchimedean;
ant.NumArms = 1
```

```
ant = 
  spiralArchimedean with properties:

             NumArms: 1
               Turns: 1.5000
         InnerRadius: 5.0000e-04
         OuterRadius: 0.0398
    WindingDirection: 'CCW'
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

[2] Nakano, H., Oyanagi, H. and Yamauchi, J. “A Wideband Circularly Polarized Conical Beam From a Two-Arm Spiral Antenna Excited in Phase”. *IEEE Transactions on Antennas and Propagation*. Vol. 59, No. 10, Oct 2011, pp. 3518-3525.

[3] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. McGraw-Hill

See Also

`spiralEquiangular` | `helix` | `yagiUda`

Topics

“Rotate Antennas and Arrays”

spiralEquiangular

Create equiangular spiral antenna

Description

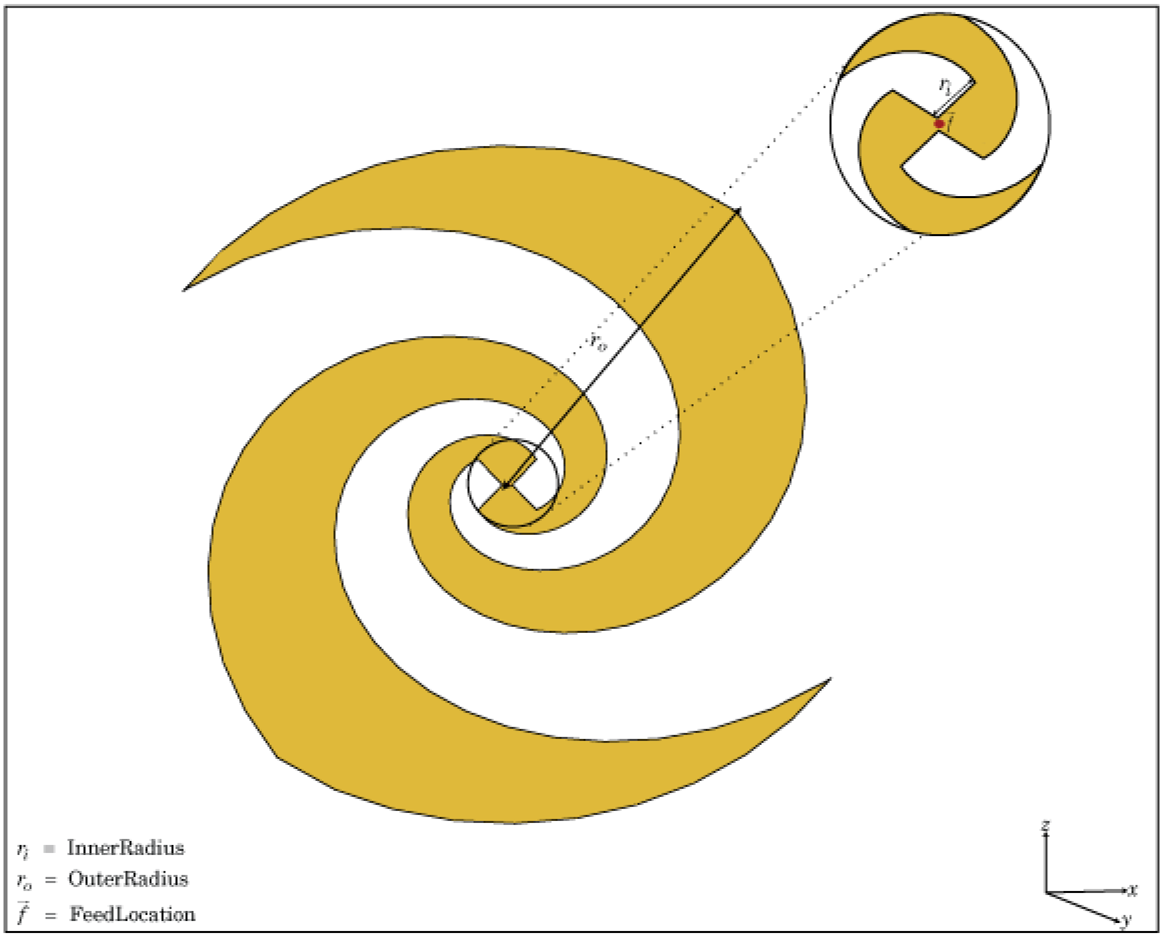
The `spiralEquiangular` object is a planar equiangular spiral antenna on the *xy*- plane. The equiangular spiral is always center fed and has two arms. The field characteristics of the antenna are frequency independent. A realizable spiral has finite limits on the feeding region and the outermost point of any arm of the spiral. This antenna exhibits a broadband behavior. The outer radius imposes the low frequency limit and the inner radius imposes the high frequency limit. The arm radius grows linearly as a function of the winding angle. As a result, outer arms of the spiral are shaped to minimize reflections.

The equation of the equiangular spiral is:

$$r = r_0 e^{a\phi}$$

, where:

- r_0 is the starting radius
- a is the growth rate
- ϕ is the winding angle of the spiral



Creation

Syntax

```
se = spiralEquiangular
se = spiralEquiangular(Name,Value)
```

Description

`se = spiralEquiangular` creates a planar equiangular spiral in the *xy*- plane. By default, the antenna operates over a broadband frequency 4–10 GHz.

`se = spiralEquiangular(Name,Value)` creates an equiangular spiral antenna, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`GrowthRate` — Equiangular spiral growth rate

0.3500 (default) | scalar

Equiangular spiral growth rate, specified as a scalar.

Example: 'GrowthRate',1.2

Data Types: double

InnerRadius — Inner radius of spiral
0.0020 (default) | scalar

Inner radius of spiral, specified as a scalar in meters.

Example: 'InnerRadius',1e-3

Data Types: double

OuterRadius — Outer radius of spiral
0.0189 (default) | scalar

Outer radius of spiral, specified as a scalar in meters.

Example: 'OuterRadius',1e-3

Data Types: double

WindingDirection — Direction of spiral turns (windings)
'CW' | 'CCW'

Direction of spiral turns (windings), specified as 'CW' or 'CCW'.

Example: 'WindingDirection','CW'

Data Types: char

Conductor — Type of metal material
'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: se.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna
0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array

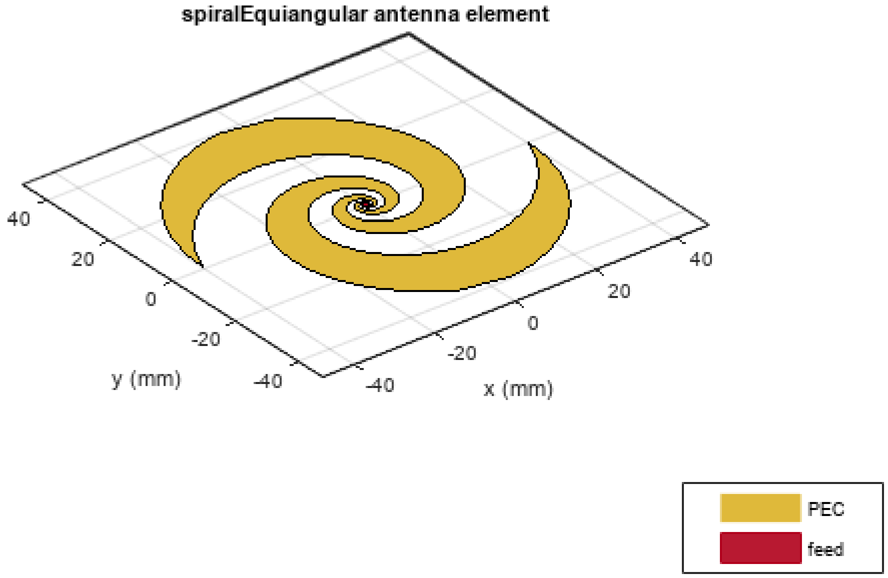
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Equiangular Spiral Antenna

Create and view an equiangular spiral antenna with 0.35 growth rate, 0.65 mm inner radius and 40 mm outer radius.

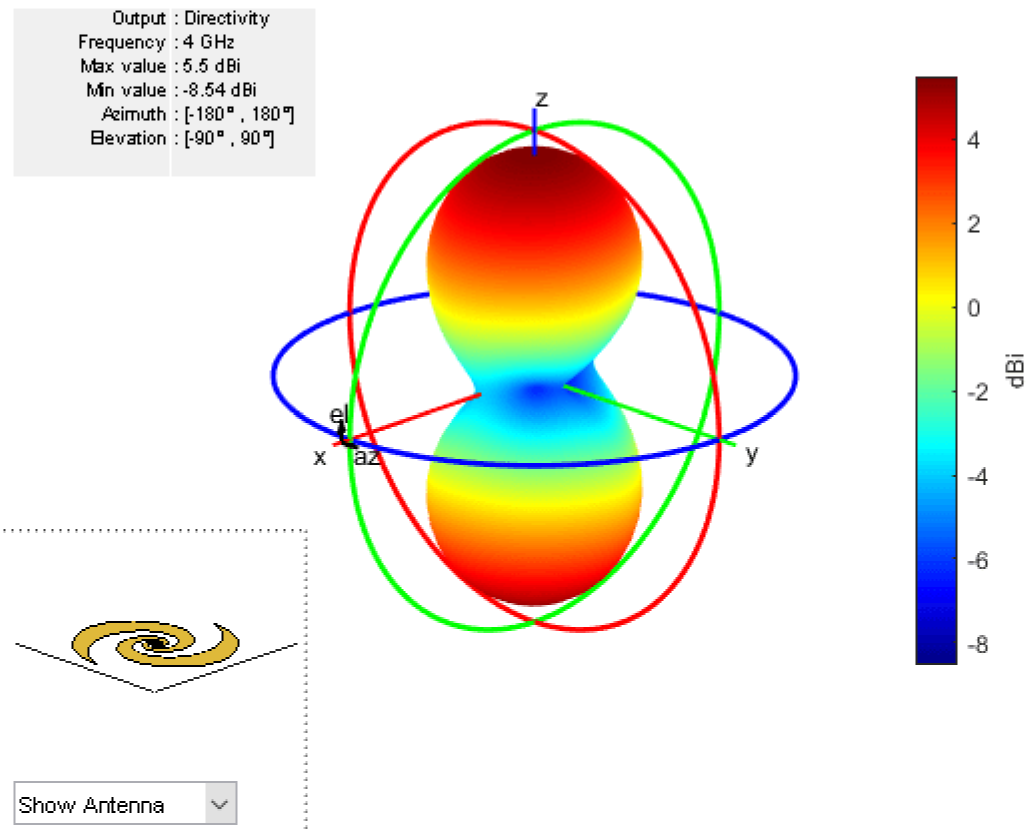
```
se = spiralEquiangular('GrowthRate',0.35, 'InnerRadius',0.65e-3,    ...
                          'OuterRadius',40e-3);
show(se)
```



Radiation Pattern of Equiangular Spiral Antenna

Plot the radiation pattern of equiangular spiral at a frequency of 4 GHz.

```
se = spiralEquiangular('GrowthRate',0.35, 'InnerRadius',0.65e-3, ...
                           'OuterRadius',40e-3);
pattern(se,4e9);
```



```
   Output : Directivity
Frequency : 4 GHz
Max value : 5.5 dBi
Min value : -8.54 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2015a

References

[1] Dyson, J. The equiangular spiral antenna." *IRE Transactions on Antennas and Propagation*. Vol.7, Number 2, pp. 181, 187, April 1959.

[2] Nakano, H., K.Kikkawa, N.Kondo, Y.Iitsuka, J.Yamauchi. "Low-Profile Equiangular Spiral Antenna Backed by an EBG Reflector." *IRE Transactions on Antennas and Propagation*. Vol. 57, No. 25, May 2009, pp. 1309–1318.

[3] McFadden, M., and Scott, W.R. "Analysis of the Equiangular Spiral Antenna on a Dielectric Substrate." *IEEE Transactions on Antennas and Propagation*. Vol. 55, No. 11, Nov. 2007, pp. 3163–3171.

[4] Violakis, John *Antenna Engineering Handbook*, 4th Ed., McGraw-Hill.

See Also

`spiralArchimedean` | `cavity` | `vivaldi`

Topics

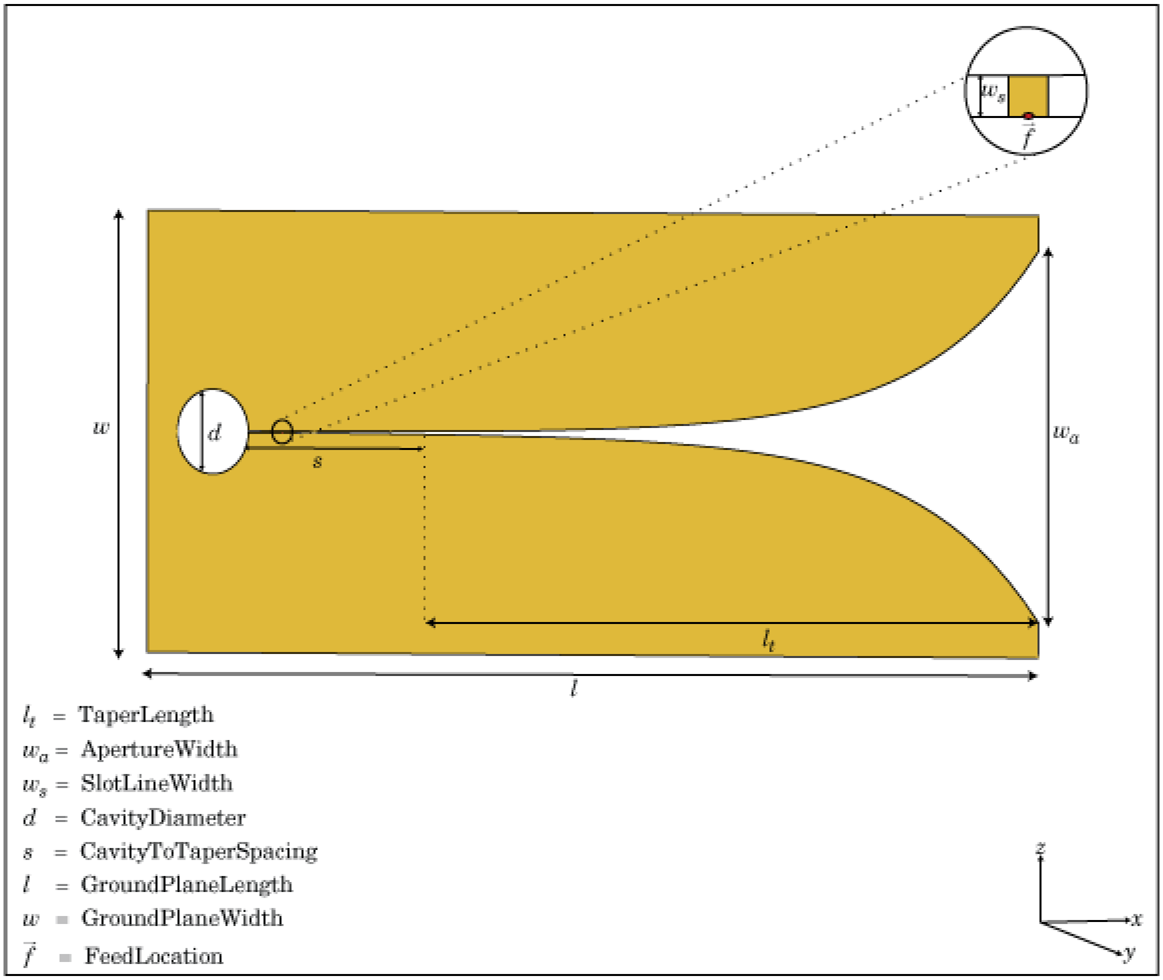
"Rotate Antennas and Arrays"

vivaldi

Create Vivaldi notch antenna on ground plane with exponential or linear tapering

Description

The `vivaldi` object is a Vivaldi notch antenna on a ground plane.



Creation

Syntax

```
vi = vivaldi
vi = vivaldi(Name,Value)
```

Description

`vi = vivaldi` creates a Vivaldi notch antenna on a ground plane. By default, the antenna operates at a frequency range of 1–2 GHz and is located in the X-Y plane.

`vi = vivaldi(Name,Value)` creates Vivaldi notch antenna, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties you do not specify retain their default values.

Properties

`TaperLength` — Taper length
0.2430 (default) | scalar

Taper length of vivaldi, specified a scalar in meters.

Example: `'TaperLength',2e-3`

`ApertureWidth` — Aperture width
0.1050 (default) | scalar

Aperture width, specified as a scalar in meters.

Example: `'ApertureWidth',3e-3`

`OpeningRate` — Taper opening rate
25 (default) | scalar

Taper opening rate, specified a scalar. This property determines the rate at which the notch transitions from the feedpoint to the aperture. When `OpeningRate` is `0`, the notch has a linear profile creating a linear tapered slot and for other values it has an exponential profile.

Example: `'OpeningRate',0.3`

Data Types: `double`

`SlotLineWidth` — Slot line width
5.0000e-04 (default) | scalar

Slot line width, specified as a scalar in meters.

Example: `'SlotLineWidth',3`

Data Types: `double`

`CavityDiameter` — Cavity termination diameter
0.0240 (default) | scalar

Cavity termination diameter, specified a scalar in meters.

Example: `'CavityDiameter',2`

Data Types: `double`

`CavityToTaperSpacing` — Cavity to taper distance of transition
0.0230 (default) | scalar

Cavity to taper distance of transition, specified as a scalar in meters. By default, this property is measured along the x-axis.

Example: `'CavityToTaperSpacing',3`

Data Types: `double`

`GroundPlaneLength` — Ground plane length

0.3000 (default) | scalar

Ground plane length, specified as a scalar in meters. By default, ground plane length is measured along the x-axis.

Example: `'GroundPlaneLength',2`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width

0.1250 (default) | scalar

Ground plane width, specified a scalar in meters. By default, ground plane width is measured along the *y*-axis.

Example: `'GroundPlaneWidth',4`

Data Types: `double`

`FeedOffset` — Distance from feed along *x*-axis

-0.1045 (default) | scalar

Distance from feed along *x*-axis, specified a scalar in meters.

Example: `'FeedOffset',3`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `vi.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer

pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

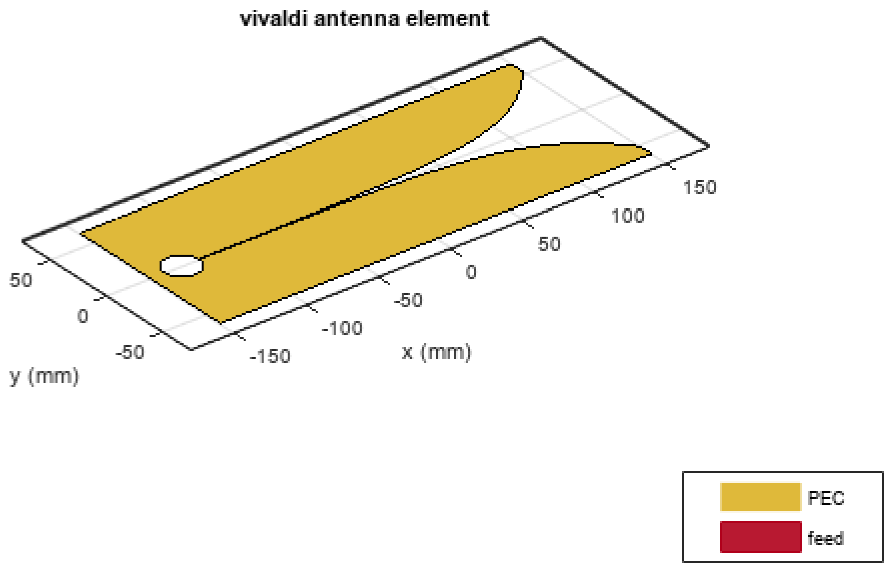
Create and View Vivaldi Antenna

Create and view the default Vivaldi antenna.

```
vi = vivaldi
```

```
vi = 
  vivaldi with properties:

             TaperLength: 0.2430
           ApertureWidth: 0.1050
             OpeningRate: 25
           SlotLineWidth: 5.0000e-04
          CavityDiameter: 0.0240
    CavityToTaperSpacing: 0.0230
       GroundPlaneLength: 0.3000
        GroundPlaneWidth: 0.1250
              FeedOffset: -0.1045
               Conductor: [1x1 metal]
                    Tilt: 0
                TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
```

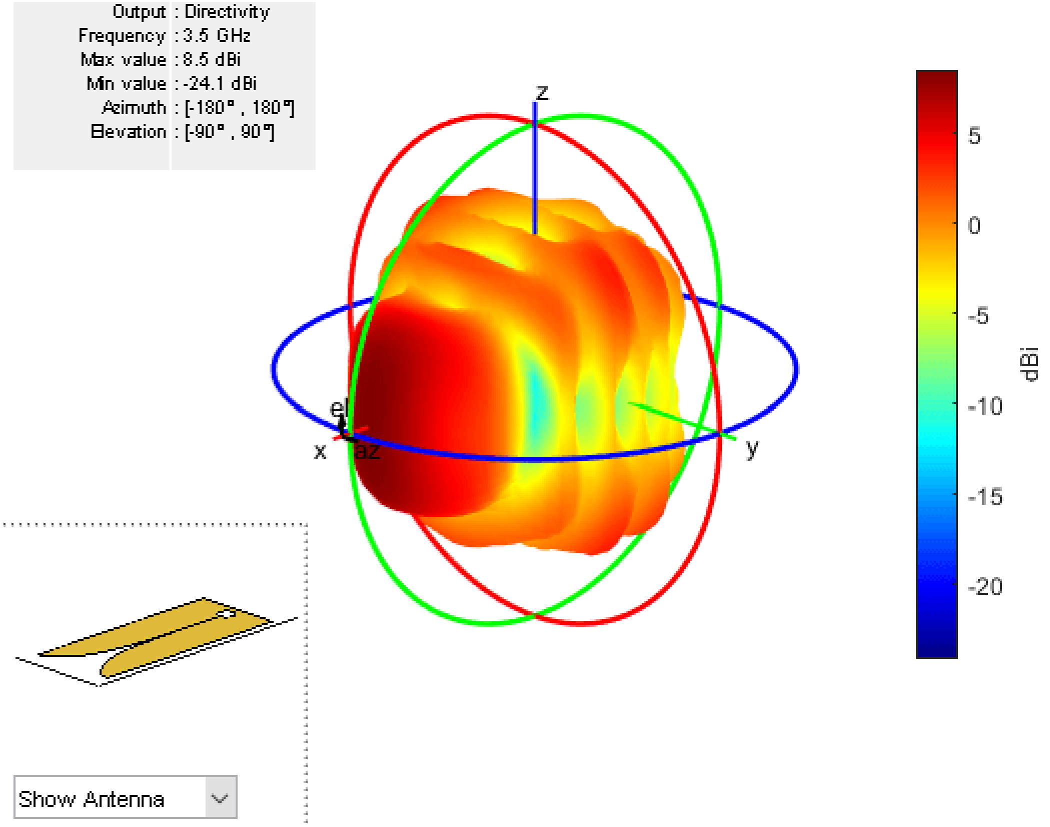
```
show(vi);
```



Radiation Pattern of Vivaldi Antenna

Plot the radiation pattern of a vivaldi antenna for a frequency of 3.5 GHz.

```
vi = vivaldi;
pattern(vi,3.5e9);
```



Output : Directivity
Frequency : 3.5 GHz
Max value : 8.5 dBi
Min value : -24.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`yagiUda` | `spiralArchimedean` | `slot`

Topics

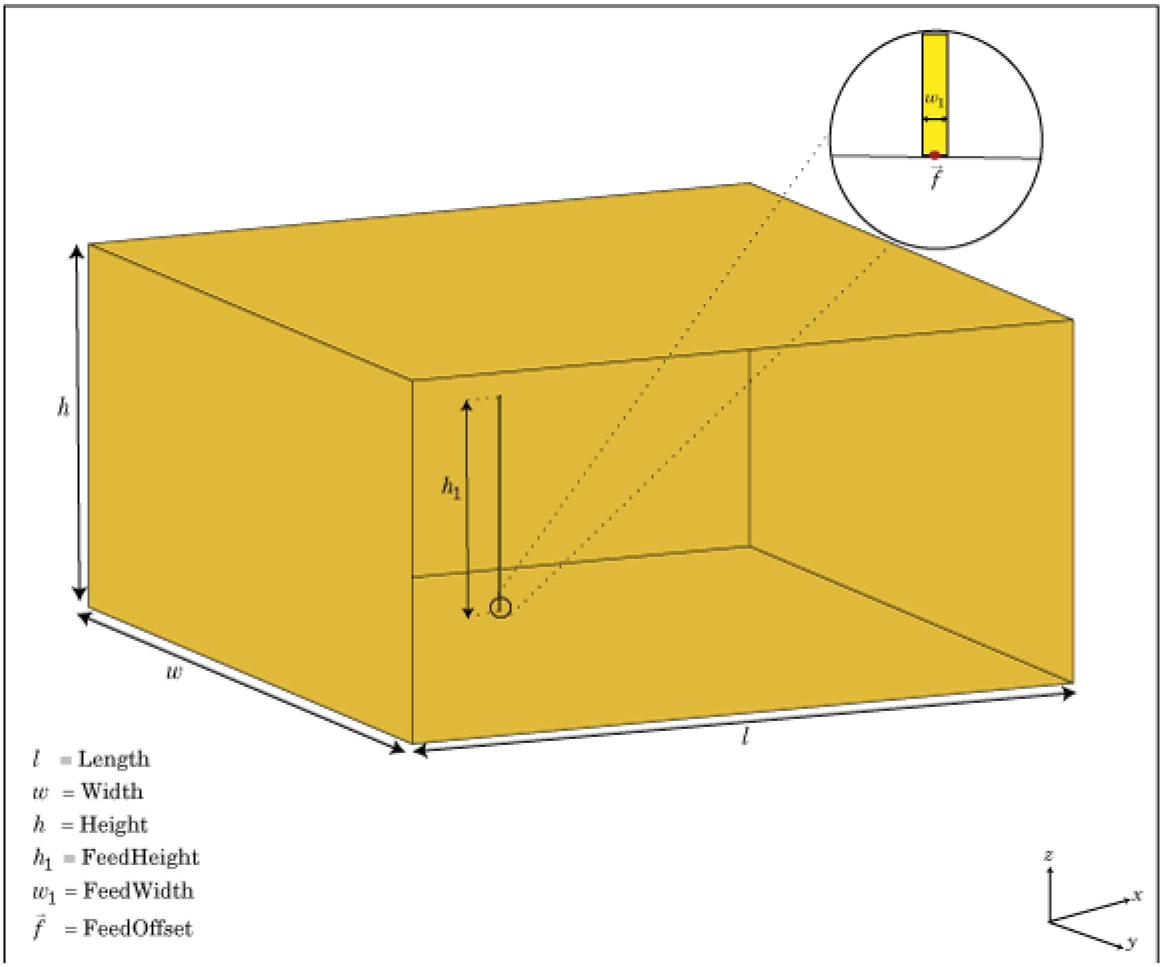
"Rotate Antennas and Arrays"

waveguide

Create rectangular waveguide

Description

The `waveguide` object is an open-ended rectangular waveguide. The default rectangular waveguide is the WR-90 and functions in the X-band. The X-band has a cutoff frequency of 6.5 GHz and ranges from 8.2 GHz to 12.5 GHz.



Creation

Syntax

```
wg = waveguide
wg = waveguide(Name,Value)
```

Description

`wg = waveguide` creates an open-ended rectangular waveguide.

`wg = waveguide(Name,Value)` creates a rectangular waveguide with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`FeedHeight` — Height of feed
`0.0060` (default) | scalar

Height of feed, specified as a scalar in meters. By default, the feed height is chosen for an operating frequency of 12.5 GHz.

Example: `'FeedHeight',0.0050`

Data Types: `double`

`FeedWidth` — Width of feed
`6.0000e-05` (default) | scalar

Width of feed, specified as a scalar in meters.

Example: `'FeedWidth',5e-05`

Data Types: `double`

`Length` — Rectangular waveguide length
`0.0240` (default) | scalar in meters

Rectangular waveguide length, specified as a scalar in meters. By default, the waveguide length is 1λ, where:

$$\lambda = c/f$$

- `c` = speed of light, 299792458 m/s
- `f` = operating frequency of the waveguide

Example: `'Length',0.09`

Data Types: `double`

`Width` — Rectangular waveguide width
`0.0229` (default) | scalar in meters

Rectangular waveguide width, specified as a scalar in meters.

Example: `'Width',0.05`

Data Types: `double`

`Height` — Rectangular waveguide height
`0.0102` (default) | scalar

Rectangular waveguide height, specified as a scalar in meters.

Example: `'Height',0.0200`

Data Types: `double`

`FeedOffset` — Signed distance of feedpoint from center of ground plane

[–0.0060 0] (default) | two-element vector

Signed distance of feedpoint from center of ground plane, specified as a two-element vector in meters. By default, the feed is at an offset of λ/4 from the shortened end on the *xy*- plane.

Example: `'FeedOffset',[–0.0070 0.01]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `wg.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

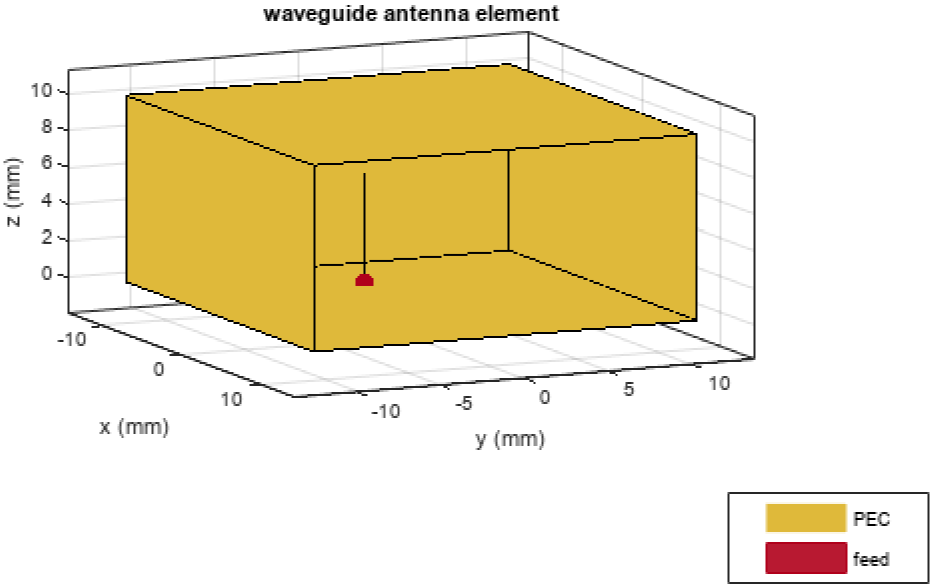
Default Rectangular Waveguide

Create a rectangular waveguide using default dimensions. Display the waveguide.

```
wg = waveguide
```

```
wg = 
  waveguide with properties:
        Length: 0.0240
         Width: 0.0229
        Height: 0.0102
     FeedWidth: 6.0000e-05
    FeedHeight: 0.0060
    FeedOffset: [-0.0060 0]
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

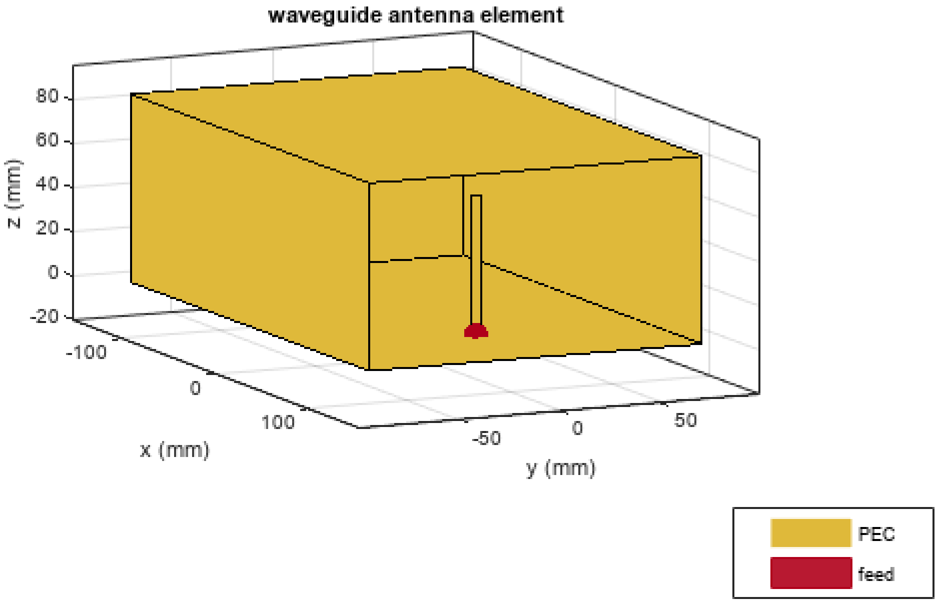
```
show(wg)
```



Radiation Pattern of WR-650 Rectangular Waveguide

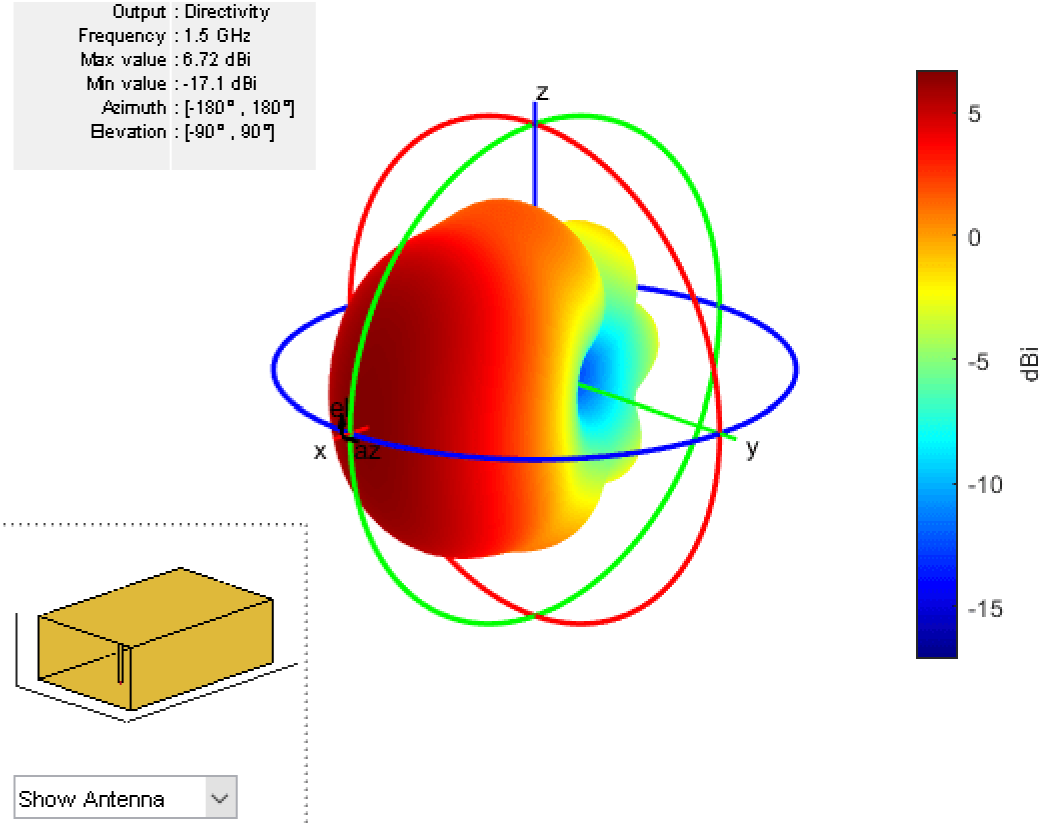
Create a WR-650 rectangular waveguide and display it.

```
wg = waveguide('Length',0.254,'Width',0.1651,'Height',0.0855,...
    'FeedHeight',0.0635,'FeedWidth',0.00508,'FeedOffset',[0.0635 0]);
show(wg)
```



Plot the radiation pattern of this waveguide at 1.5 GHz.

```
figure
pattern(wg,1.5e9)
```



Output : Directivity
Frequency : 1.5 GHz
Max value : 6.72 dBi
Min value : -17.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2016a

References

[1] Balanis, Constantine A.*Antenna Theory. Analysis and Design*. 3rd Ed. New York: John Wiley and Sons, 2005.

See Also

`horn`

Topics

"Rotate Antennas and Arrays"

yagiUda

Create Yagi-Uda array antenna

Description

The `yagiUda` class creates a classic Yagi-Uda array comprised of an exciter, reflector, and *N*-directors along the *z*-axis. The reflector and directors create a traveling wave structure that results in a directional radiation pattern.

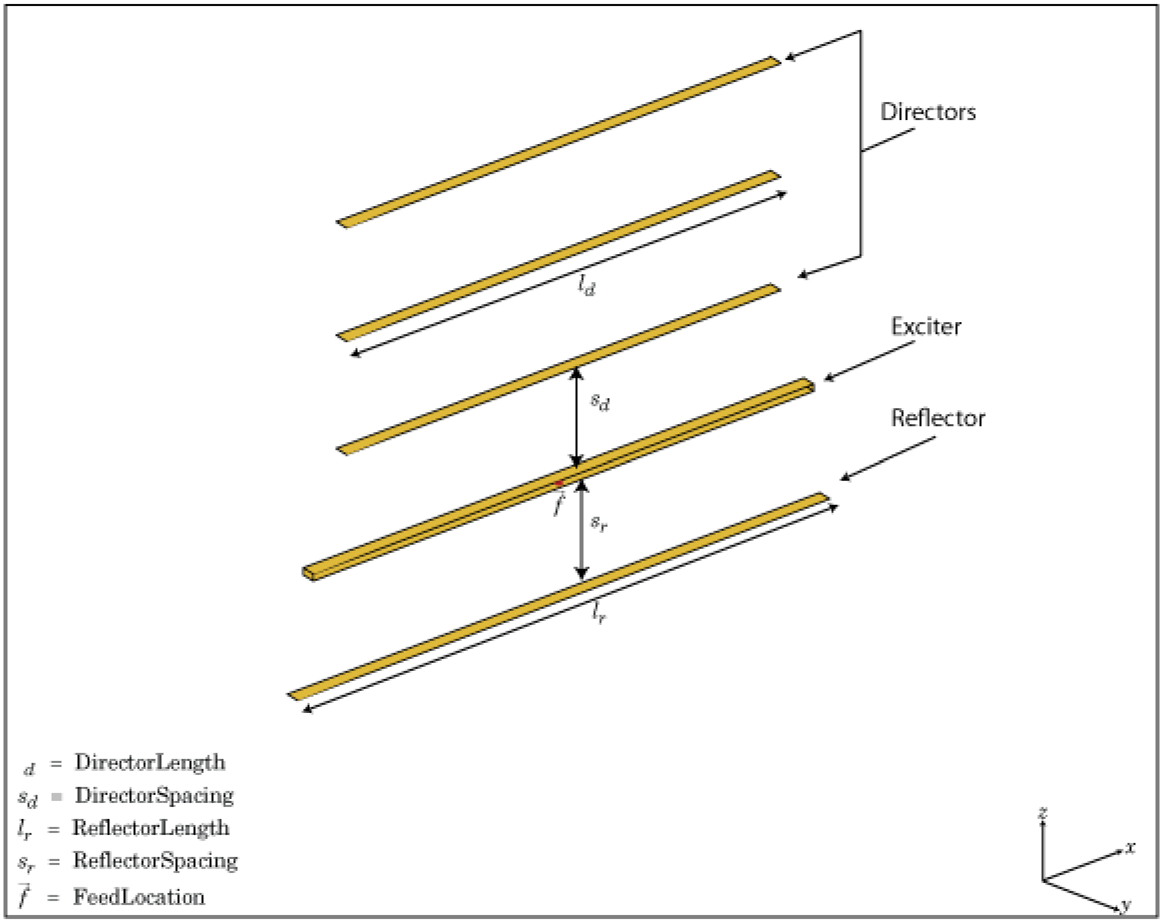
The exciter, reflector, and directors have equal widths and are related to the diameter of an equivalent cylindrical structure by the equation

$$w = 2d = 4r$$

where:

- d is the diameter of equivalent cylinder
- r is the radius of equivalent cylinder

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. A typical Yagi-Uda antenna array uses folded dipole as an exciter, due to its high impedance. The Yagi-Uda is center-fed and the feed point coincides with the origin. In place of a folded dipole, you can also use a planar dipole as an exciter.



Creation

Syntax

```
yu = yagiUda
yu = yagiUda(Name,Value)
```

Description

`yu = yagiUda` creates a half-wavelength Yagi-Uda array antenna along the Z-axis. The default Yagi-Uda uses folded dipole as three directors, one reflector, and a folded dipole as an exciter. By default, the dimensions are chosen for an operating frequency of 300 MHz.

`yu = yagiUda(Name,Value)` creates a half-wavelength Yagi-Uda array antenna, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain default values.

Properties

`Exciter` — Antenna type used as exciter

dipoleFolded (default) | object

Antenna Type used as exciter, specified as the comma-separated pair consisting of `'Exciter'` and an object.

Example: `'Exciter',dipole`

`NumDirectors` — Total number of director elements

3 (default) | scalar

Total number of director elements, specified as a scalar.

Note Number of director elements should be less than or equal to 20.

Example: `'NumDirectors',13`

Data Types: `double`

`DirectorLength` — Director length

0.4080 (default) | scalar | vector

Director length, specified as a scalar or vector in meters.

Example: `'DirectorLength',[0.4 0.5]`

Data Types: `double`

`DirectorSpacing` — Spacing between directors

0.3400 (default) | scalar | vector

Spacing between directors, specified as a scalar or vector in meters.

Example: `'DirectorSpacing',[0.4 0.5]`

Data Types: `double`

`ReflectorLength` — Reflector length

0.5000 (default) | scalar

Reflector length, specified as a scalar in meters.

Example: `'ReflectorLength',0.3`

Data Types: `double`

`ReflectorSpacing` — Spacing between exciter and reflector

0.2500 (default) | scalar

Spacing between exciter and reflector, specified as a scalar in meters.

Example: `'ReflectorSpacing', 0.4`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `yu.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

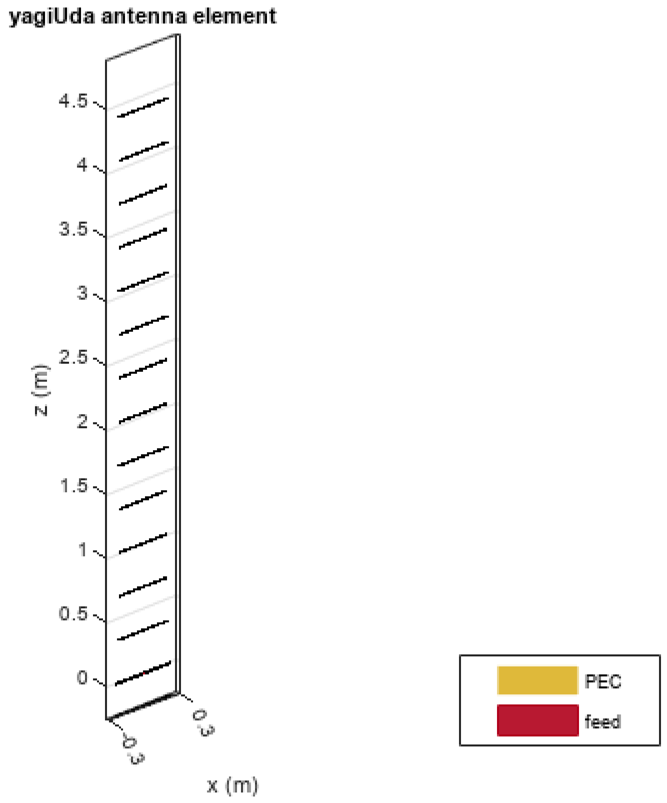
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create and View Yagi-Uda Array Antenna

Create and view a Yagi-Uda array antenna with 13 directors.

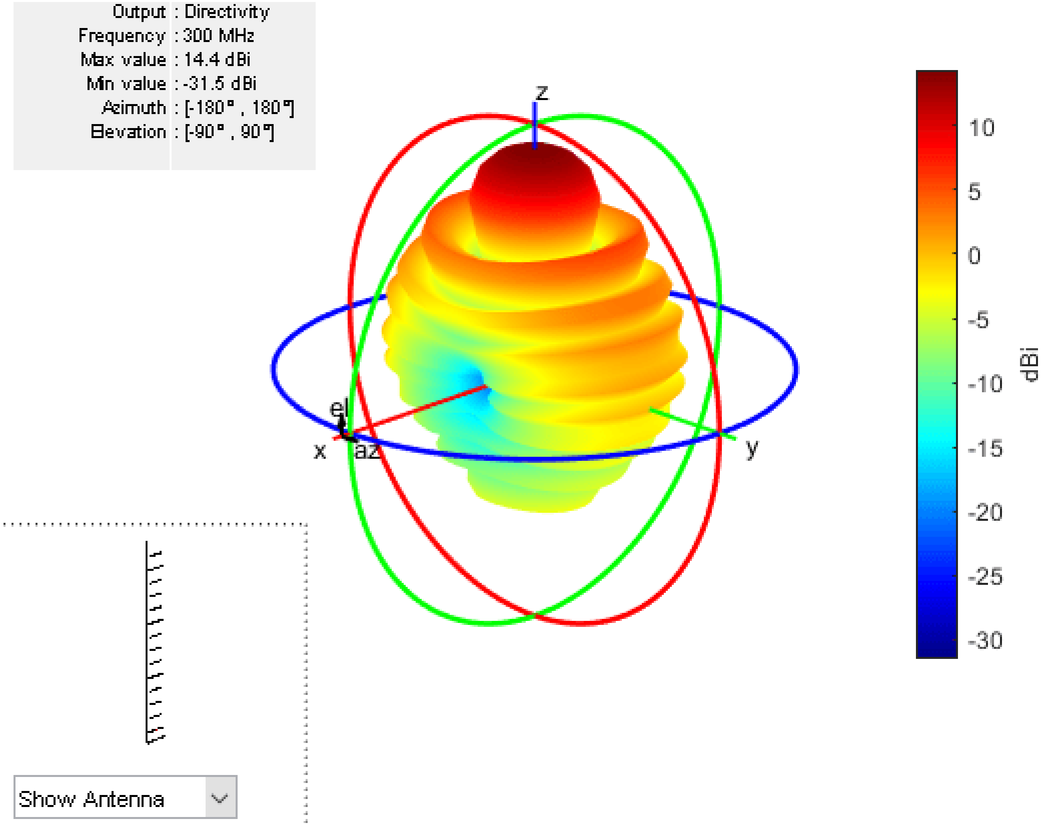
```
y = yagiUda('NumDirectors',13);
show(y)
```



Radiation Pattern of Yagi-Uda Array Antenna

Plot the radiation pattern of a Yagi-Uda array antenna at a frequency of 300 MHz.

```
y = yagiUda('NumDirectors',13);
pattern(y,300e6)
```



Calculate Cylinder to Strip Approximation

Calculate the width of the strip approximation to a cylinder of radius 20 mm.

```
w = cylinder2strip(20e-3)
```

```
w = 0.0800
```

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

dipole | dipoleFolded | slot | cylinder2strip

Topics

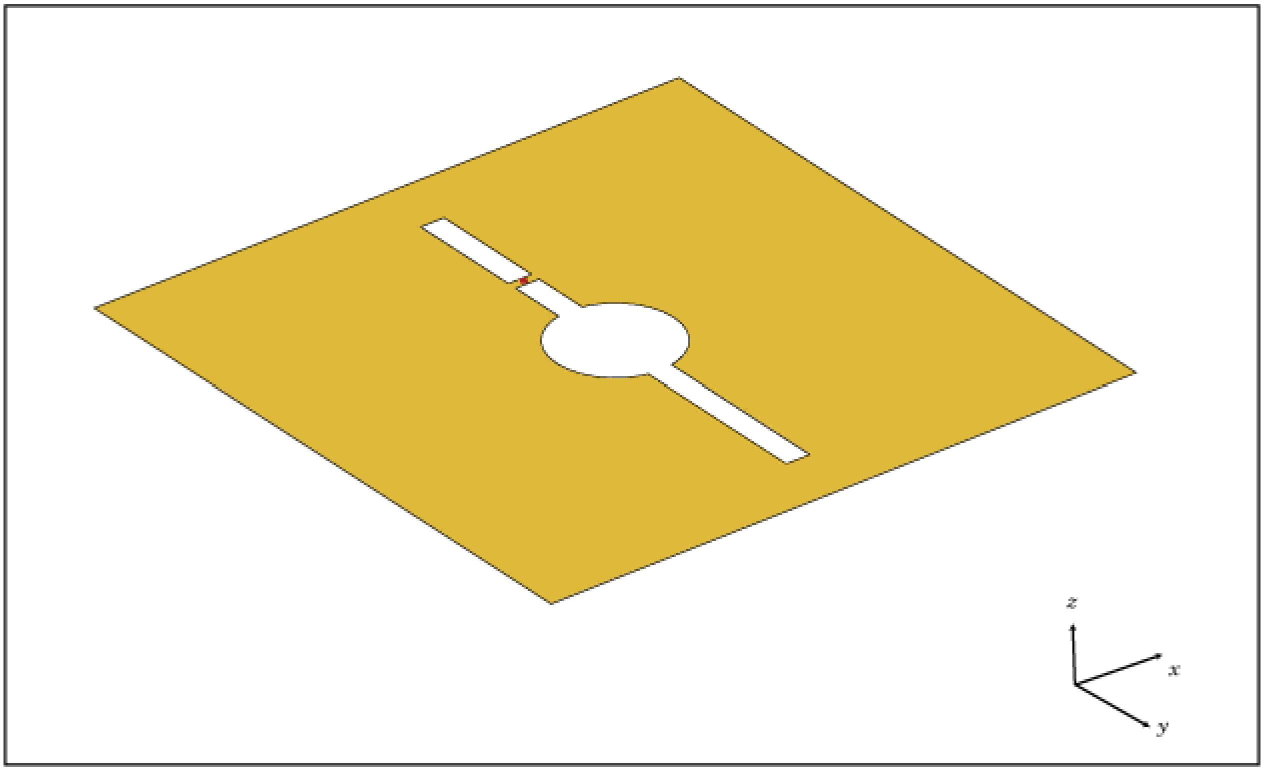
“Rotate Antennas and Arrays”

customAntennaGeometry

Create antenna represented by 2-D custom geometry

Description

The `customAntennaGeometry` object is an antenna represented by a 2-D custom geometry on the *xy*- plane. Using `customAntennaGeometry`, you can import a planar mesh, define the feed for this mesh to create an antenna, analyze the antenna, and use it in finite or infinite arrays. The image shown is a custom slot antenna.



Creation

Syntax

```
ca = customAntennaGeometry
ca = customAntennaGeometry(Name,Value)
```

Description

`ca = customAntennaGeometry` creates a 2-D antenna represented by a custom geometry, based on the specified boundary.

ca = customAntennaGeometry(Name,Value) creates a 2-D planar antenna geometry, with additional properties specified by one or more name-value pair arguments. Name is the property name and Value is the corresponding value. You can specify several name-value pair arguments in any order as Name1, Value1, ..., NameN, ValueN. Properties not specified retain their default values.

Properties

Boundary — Boundary information in Cartesian coordinates

cell array

Boundary information in Cartesian coordinates, specified as a cell array in meters.

Data Types: double

Operation — Boolean operation performed on boundary list

'P1' (default) | character vector

Boolean operation performed on the boundary list, specified as a character vector.

Example: 'Operation','P1-P2'

Data Types: double

FeedLocation — Antenna feed location in Cartesian coordinates

[0 0 0] (default) | three-element vector

Antenna feed location in Cartesian coordinates, specified as a three-element vector. The three-element vector is the *x*, *y*, and *z* coordinates respectively.

Example: 'FeedLocation', [0 0.2 0]

Data Types: double

FeedWidth — Width of feed section

0.0100 (default) | scalar

Width of feed section, specified as a scalar in meters.

Example: 'FeedWidth',0.05

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified in the metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified a lumped element object. For more information, see lumpedElement.

Example: `'Load'`, lumpedelement. `lumpedelement` is the object for the load created using `lumpedElement`.

`Tilt` — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of antenna, specified as a scalar or vector with each element unit in degrees.

Example: `'Tilt',90`

Example: `'Tilt',[90 90 0]`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array

Examples

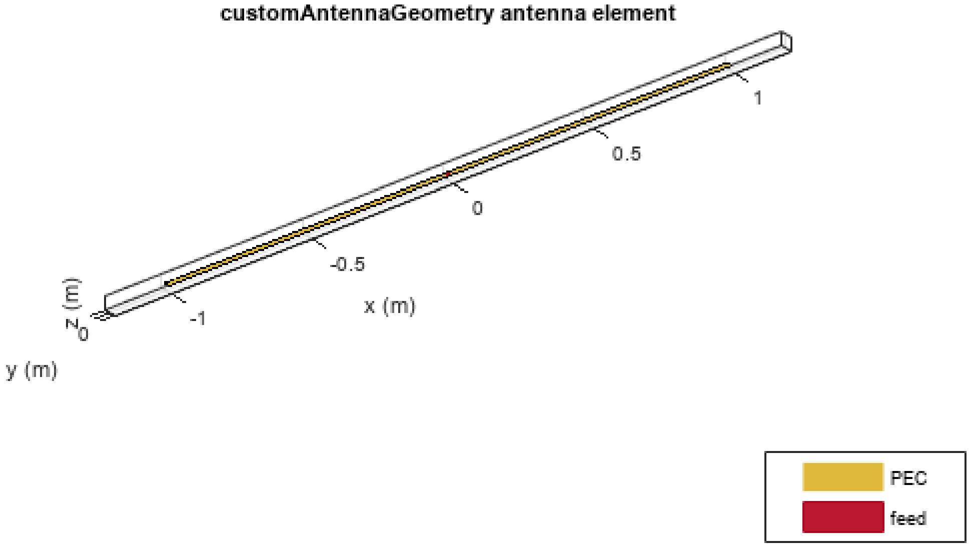
Custom Dipole Antenna

Create a custom dipole antenna and view it.

```
ca = customAntennaGeometry
```

```
ca = 
  customAntennaGeometry with properties:

        Boundary: {[4x3 double]}
       Operation: 'P1'
    FeedLocation: [0 0 0]
       FeedWidth: 0.0200
       Conductor: [1x1 metal]
            Tilt: 0
        TiltAxis: [1 0 0]
            Load: [1x1 lumpedElement]
```

```
show(ca)
```



Custom Slot Antenna

Create a custom slot antenna using three rectangles and a circle.

Create three rectangles of dimensions 0.5 m x 0.5 m, 0.02 m x 0.4 m and 0.03 m x 0.008 m.

```
pr = antenna.Rectangle('Length',0.5,'Width',0.5);
pr1 = antenna.Rectangle('Length',0.02,'Width',0.4);
pr2 = antenna.Rectangle('Length',0.03,'Width',0.008);
```

Create a circle of radius 0.05 m.

```
ph = antenna.Circle('Radius',0.05);
```

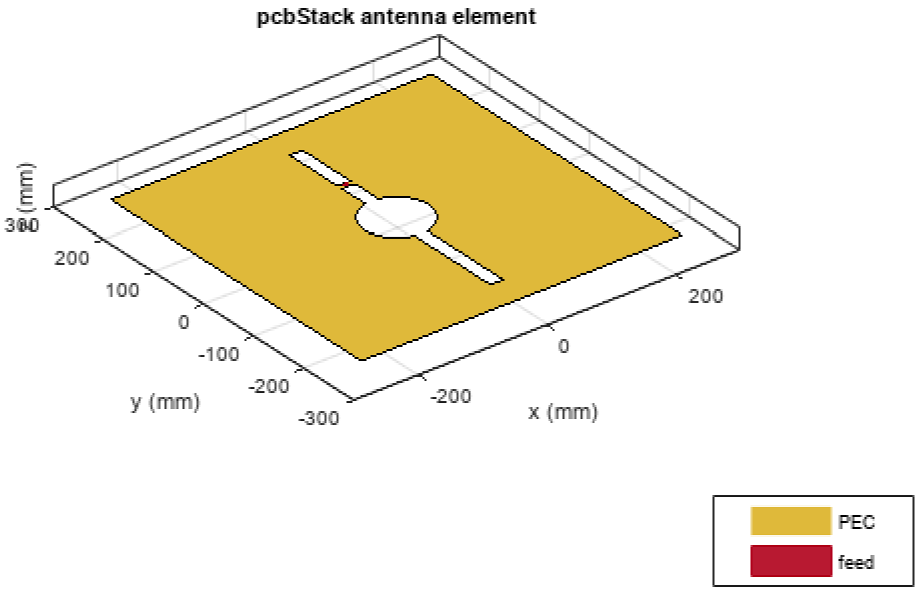
Translate the third rectangle to the X-Y plane using the coordinates [0 0.1 0].

```
pf = translate(pr2,[0 0.1 0]);
```

Create a custom slot antenna shape by performing add and subtract operations on the above shapes. Then assign this shape as a layer to the pcbStack object. Set the feed location and feed diameter.

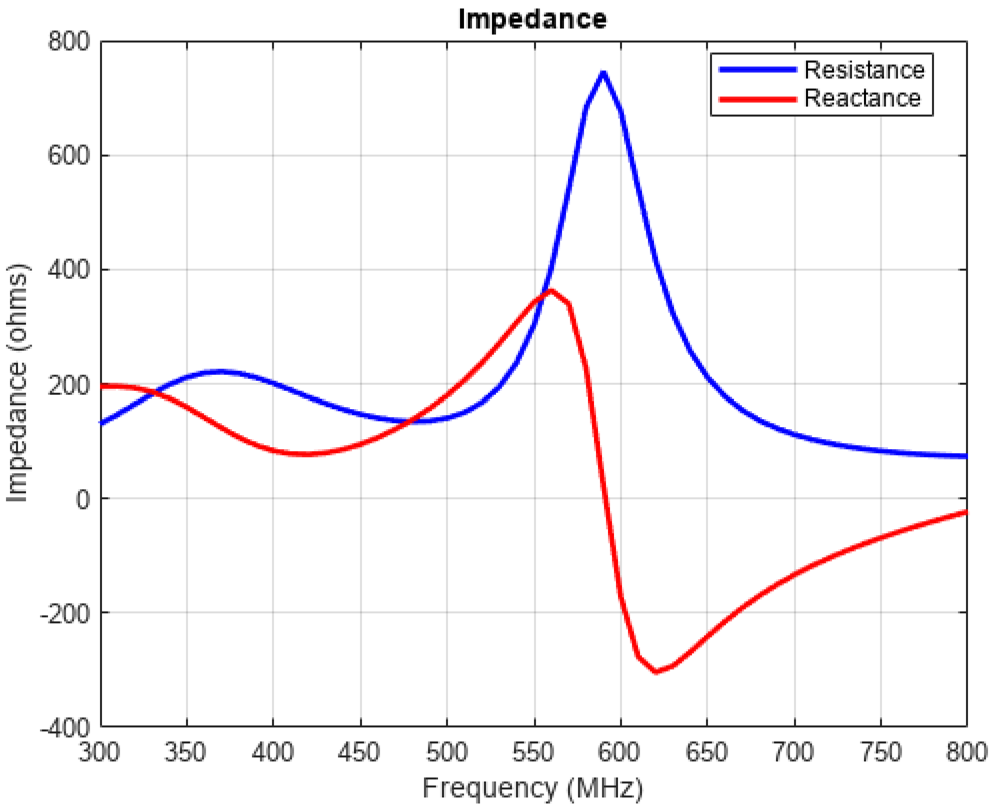
```
s = pr-ph-pr1+pf;
c = pcbStack;
boardShape = antenna.Rectangle('Length',0.6,'Width',0.6);
```

```
c.BoardShape = boardShape;
c.Layers = {s}

c =
  pcbStack with properties:

              Name: 'MyPCB'
          Revision: 'v1.0'
        BoardShape: [1x1 antenna.Rectangle]
    BoardThickness: 0.0100
            Layers: {[1x1 antenna.Polygon]}
     FeedLocations: [-0.0187 0 1 2]
      FeedDiameter: 1.0000e-03
      ViaLocations: []
       ViaDiameter: []
      FeedViaModel: 'strip'
       FeedVoltage: 1
         FeedPhase: 0
         Conductor: [1x1 metal]
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1x1 lumpedElement]


c.FeedDiameter = 0.005;
c.FeedLocations = [0 0.1 1];
% View the antenna
figure;
show(c);
```



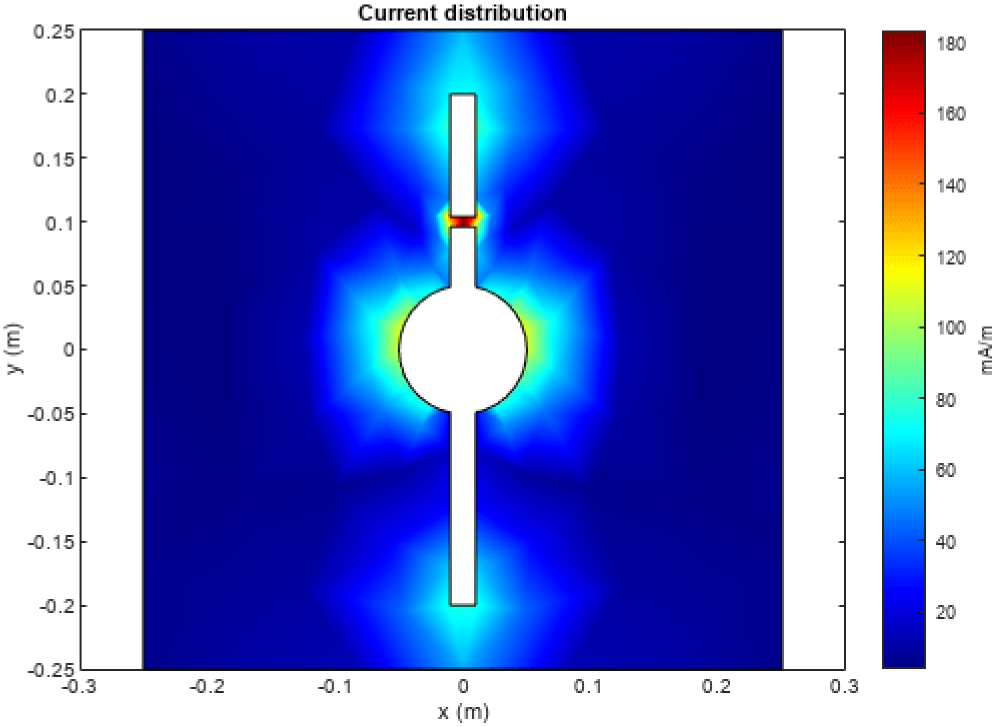
Analyze the impedance of the antenna from 300 MHz to 800 MHz.

```
figure;
impedance(c, linspace(300e6,800e6,51));
```



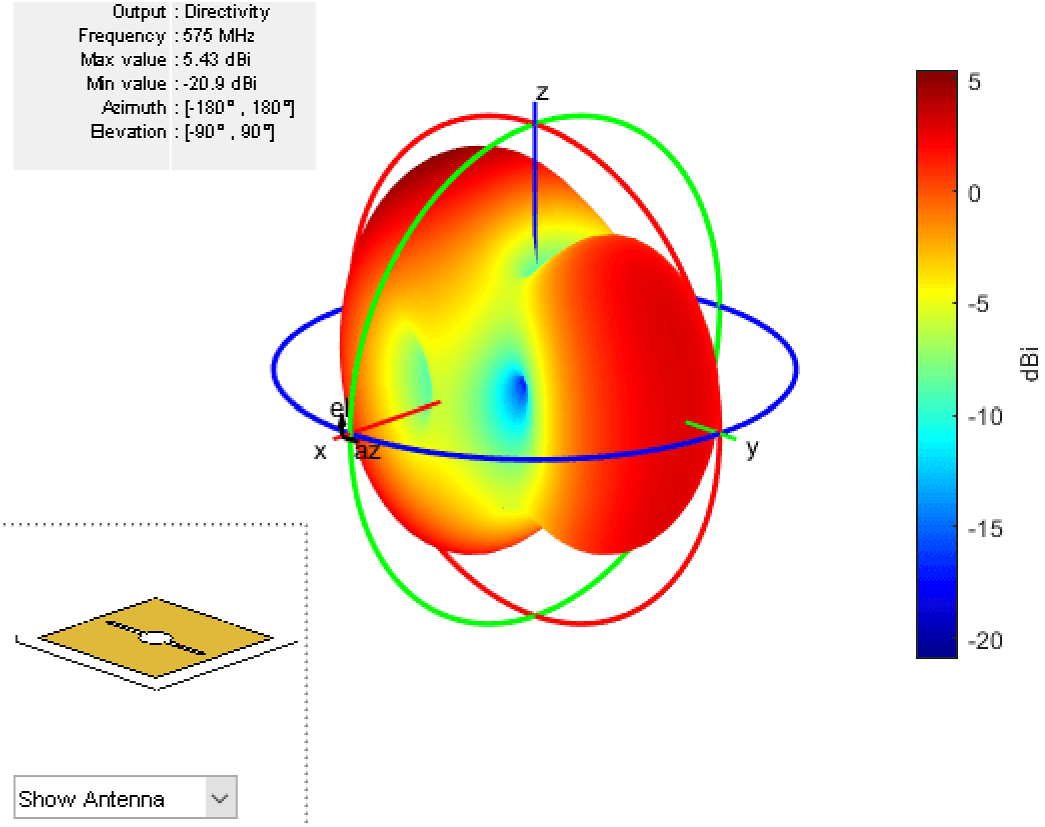
Analyze the current distribution of the antenna at 575 MHz.

```
figure;
current(c,575e6)
```



Plot the radiation pattern of the antenna at 575 MHz.

```
figure;
pattern(c,575e6)
```



```
      Output : Directivity
   Frequency : 575 MHz
   Max value : 5.43 dBi
   Min value : -20.9 dBi
     Azimuth : [-180° , 180°]
   Elevation : [-90° , 90°]
```

Version History

Introduced in R2016b

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

See Also

Topics

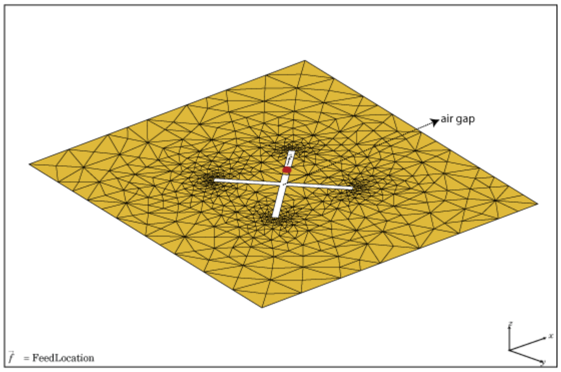
"Rotate Antennas and Arrays"

customAntennaMesh

Create 2-D custom mesh antenna on X-Y plane

Description

The `customAntennaMesh` object creates an antenna represented by a 2-D custom mesh on the X-Y plane. You can provide an arbitrary antenna mesh to the Antenna Toolbox and analyze this mesh as a custom antenna for port and field characteristics.



Creation

Description

`customantenna = customAntennaMesh(points,triangles)` creates a 2-D antenna represented by a custom mesh, based on the specified points and triangles.

Input Arguments

`points` — Points in custom mesh
`2`-`by`-*`N`* or `3`-`by`-*`N`* integer matrix of Cartesian coordinates in meters

Points in a custom mesh, specified as a `2`-`by`-*`N`* or `3`-`by`-*`N`* integer matrix of Cartesian coordinates in meters. *N* is the number of points. In case you specify a `3`x*`N`* integer matrix, the Z-coordinate must be zero or a constant value. This value sets the `'Points'` property in the custom antenna mesh.

Example: `[0 1 0 1;0 1 1 0]`

Data Types: `double`

`triangles` — Triangles in mesh
`4`-`by`-*`M`* integer matrix

Triangles in the mesh, specified as a `4`-`by`-*`M`* integer matrix. *M* is the number of triangles. The first three rows are indices to the points matrix and represent the vertices of each triangle. The fourth row is a domain number useful for identifying separate parts of an antenna. This value sets the `'Triangles'` property in the custom antenna mesh.

Data Types: `double`

Properties

`Points` — Points in custom mesh

`2-by-N` or `3-by-N` integer matrix of Cartesian coordinates

Points in a custom mesh, specified as a `2-by-N` or `3-by-N` integer matrix of Cartesian coordinates in meters. *N* is the number of points.

Example: `[0.1 0.2 0]`

Data Types: `double`

`Triangles` — Triangles in mesh

`4-by-M` integer matrix

Triangles in the mesh, specified as a `4-by-M` integer matrix. *M* is the number of triangles.

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified in the metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see "Meshing".

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.

- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

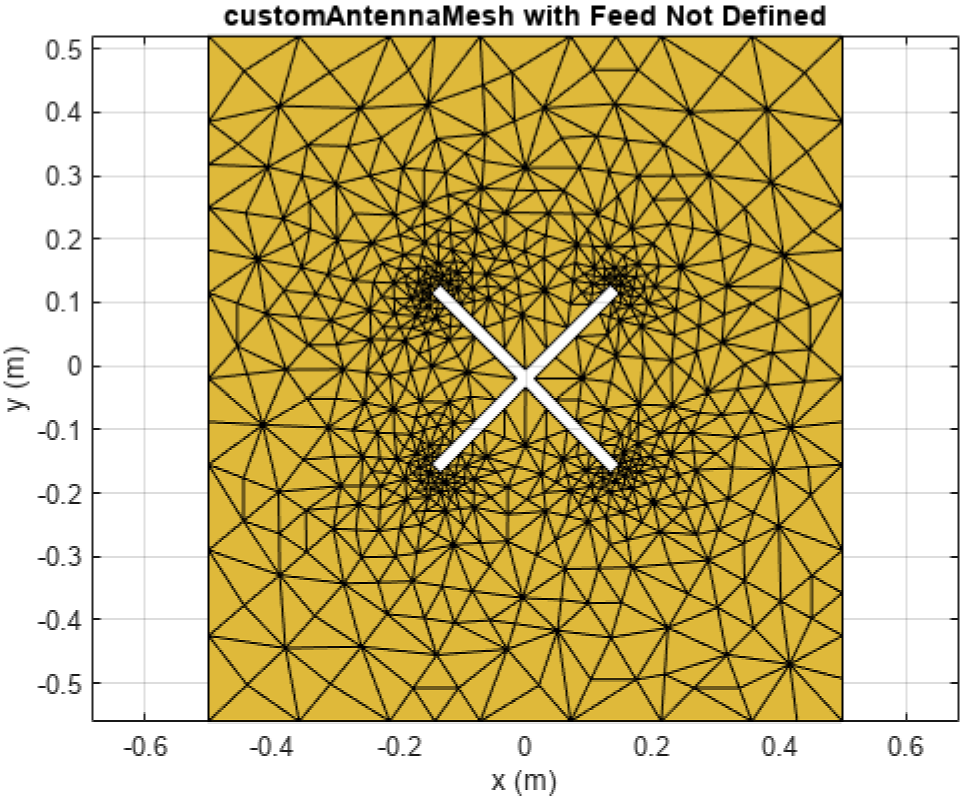
show	Display antenna, array structures or shapes
info	Display information about antenna or array
createFeed	Create feed location for custom antenna
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Custom Mesh Antenna

Load a custom planar mesh. Create the antenna and antenna feed. View the custom planar mesh antenna and calculate the impedance at 100 MHz.

```
load planarmesh.mat;
c = customAntennaMesh(p,t);
show(c)
```



```
createFeed(c,[0.07,0.01],[0.05,0.05]);
Z = impedance(c,100e6)
```

```
Z = 0.5091 + 57.2102i
```

Version History

Introduced in R2015b

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

See Also

`reflector` | `cavity`

Topics

"Rotate Antennas and Arrays"

pcbStack

Single-feed or multifeed PCB antenna

Description

The `pcbStack` object is a single-feed or multi-feed printed circuit board (PCB) antenna. The `pcbStack` object can be used

- To create single-layer, multilayer metal, or metal-dielectric substrate antennas
- To create an arbitrary number of feeds and vias in an antenna
- To create a PCB antenna with Antenna Toolbox antenna catalog elements
- To convert antenna array elements to PCB stack

Note To generate a Gerber file, a substrate layer is required. Use the `Substrate` property to create this layer in the PCB antenna. For more information, see "Stack Conversion" on page 1-256

Creation

Syntax

```
pcbant = pcbStack
pcbant = pcbStack(Name,Value)
pcbant = pcbStack(ant)
```

Description

`pcbant = pcbStack` creates an air-filled single-feed PCB with two metal layers.

`pcbant = pcbStack(Name,Value)` sets "Properties" on page 1-247 using name-value pairs. For example, `pcbStack('FeedDiameter', 2.000e-04)` creates a PCB antenna with a feed diameter of 2.000e-04 meters. You can specify multiple name-value pairs. Enclose each property name in quotes creates a PCB antenna, with additional properties specified by one or more name-value pair arguments. Properties not specified retain their default values.

`pcbant = pcbStack(ant)` converts any 2-D or 2.5-D antenna from the antenna catalog into a PCB antenna for further modeling and analysis. You can also use antenna array objects from the antenna array catalog elements them convert it into PCB antennas.

Properties

`Name` — Name of PCB antenna
`'MyPCB'` (default) | character vector

Name of PCB antenna, specified a character vector.

Example: `'Name','PCBPatch'`

Data Types: `char` | `string`

`Revision` — Revision details of PCB antenna design

`'1.0'` (default) | character vector

Revision details of PCB antenna design, specified as a character vector.

Example: `'Revision','2.0'`

Data Types: `char` | `string`

`BoardShape` — Shape of PC board

`antenna.Rectangle` (default) | object

Shape of PC board, specified as an object. The shape can be a rectangle or a polygon.

Example: `'BoardShape',antenna.Polygon`

`BoardThickness` — Thickness of PC board

`0.0100` (default) | positive scalar

Thickness of the PC board, specified as a positive scalar. The value of this property is the sum of the thicknesses of all the dielectric layers that lie below the top metal layer. The object treats dielectric layers that are above the top metal layer as coating. Set this property before you set the `Layers` property. For more information on `BoardThickness`, see “Board Thickness versus Dielectric Thickness in PCB”.

Example: `'BoardThickness',0.02000`

Data Types: `double`

`Layers` — Metal and dielectric layers

`{[1×1 antenna.Rectangle] [1×1 antenna.Rectangle]}` (default) | cell array of metal layer shapes and dielectric

Metal and dielectric layers, specified a cell array of metal layer shapes and dielectric. You can specify one metal shape or one dielectric per layer starting with the top layer and proceeding down.

Data Types: `cell`

`FeedLocations` — Feed locations for antenna in Cartesian coordinates

`[-0.0187 0 1 2]` (default) | *N* -by-3 or *N*-by-4 array

Feed locations for PCB antenna in Cartesian coordinates, specified as *N* -by-3 or *N*-by-4 array. You can place feed inside the board or at the edge of the board. The arrays translate to the following:

- *N* -by-3 – [*x*, *y*, *Layer*]
- *N*-by-4 – [*x*, *y*, *SigLayer*, *GndLayer*]

  Antenna Toolbox uses Delta-Gap source feed model to excite the antenna structure. For more information, see “Feed Model”

Example: `'FeedLocations',[-0.0187 0 1 2]`

Data Types: `double`

`FeedDiameter` — Center pin diameter of feed connector

`1.0000e-03` (default) | positive scalar in meters

Center pin diameter of feed connector, specified as a positive scalar in meters.

Example: `'FeedDiameter',2.000e-04`

Data Types: `double`

`ViaLocations` — Electrical short locations for antenna in Cartesian coordinates

real vector of size *M*-by-4 array

Electrical short locations for antenna in Cartesian coordinates, specified as a real vector of size *M*-by-4 array. The arrays translate to the following:

- *M*-by-4 – [*x*, *y*, *SigLayer*, *GndLayer*]

Example: `'ViaLocations',[0 -0.025 1 2]`

Data Types: `double`

`ViaDiameter` — Electrical shorting pin diameter between metal layers

positive scalar | positive vector

Electrical shorting pin diameter between metal layers, specified as a positive scalar in meters for a single pin or a positive vector in meters for multiple pins. Number of values specified in this property must match the number of pins.

Example: `'ViaDiameter',1.0e-3`

Data Types: `double`

`FeedVoltage` — Magnitude voltage applied at the feeds

`1` (default) | positive scalar in volts

Magnitude voltage applied at the feeds, specified as a positive scalar in volts.

Example: `'FeedVoltage',2`

Data Types: `double`

`FeedViaModel` — Model for approximating feed and via

`'strip'` (default) | `'square'` | `'hexagon'` | `'octagon'`

Model for approximating feed and via, specified as one of the following:

- `'strip'` – A rectangular strip approximation to the feed or via cylinder. This approximation is the simplest and results in a small mesh.
- `'square'` – A 4-sided polyhedron approximation to the feed or via cylinder.
- `'hexagon'` – A 6-sided polyhedron approximation to the feed or via cylinder.
- `'octagon'` – A 8-sided polyhedron approximation to the feed or via cylinder.

Example: `'FeedViaModel','octagon'`

Data Types: `char` | `string`

`FeedPhase` — Excitation phase at each feed

`0` (default) | real vector in degrees

Excitation phase at each feed, specified as a real vector in degrees.

Example: `'FeedPhase',2`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object handle

Lumped elements added to the antenna feed, specified as a lumped element object handle. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object handle for the load created using `lumpedElement`.

Example: `pcbant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
array	Create array of PCB stack objects
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
info	Display information about antenna or array
layout	Display array or PCB stack layout
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna
plot	Plot boundary of shape

Examples

End Loaded Planar Dipole

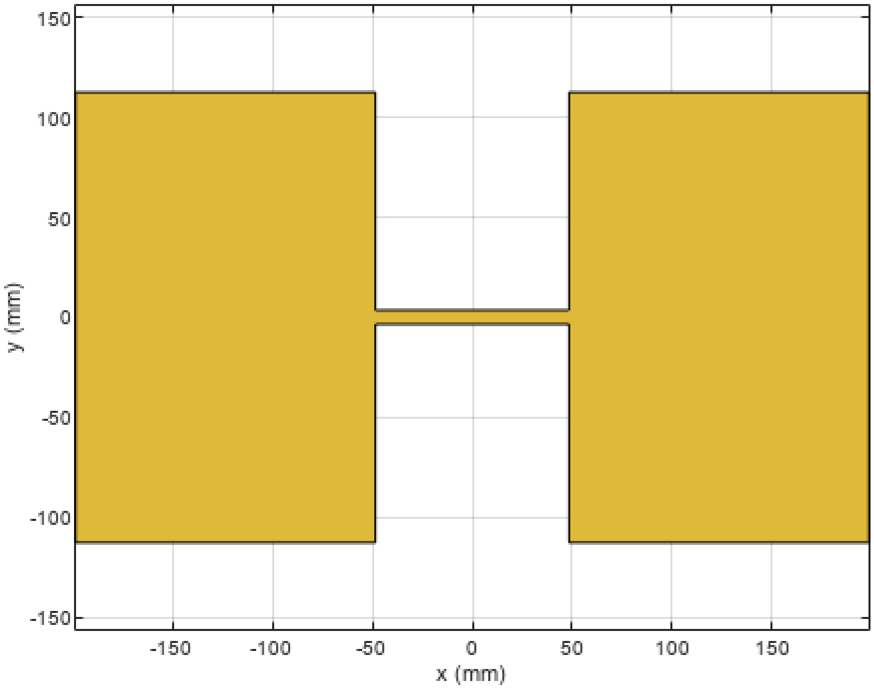
Setup parameters.

```
vp  = physconst('lightspeed');
f   = 850e6;
lambda = vp./f;
```

Build a planar dipole with capacitive loading at the ends.

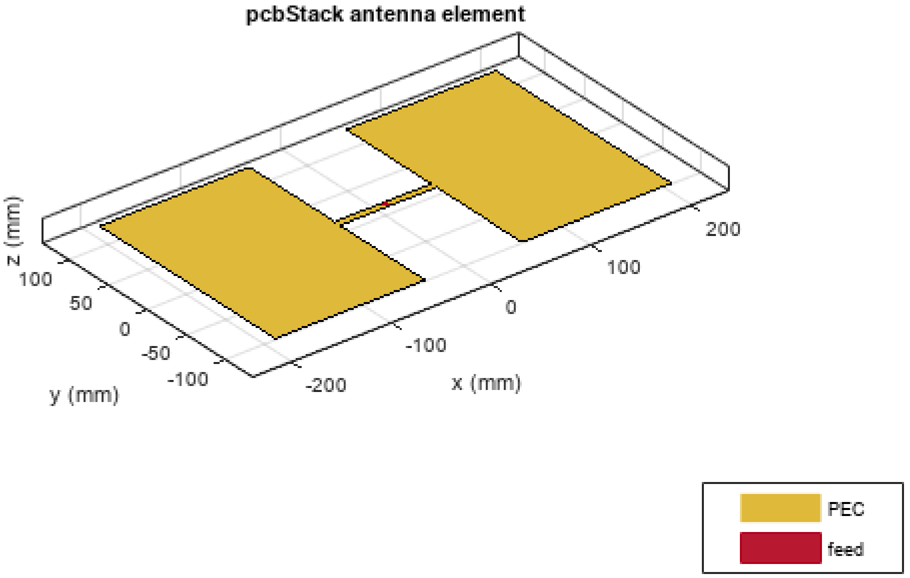
```
L = 0.15;
W = 1.5*L;
stripL = L;
gapx = .015;
gapy = .01;
r1 = antenna.Rectangle('Center',[0,0],'Length',L,'Width',W,'Center',[lambda*0.35,0]);
r2 = antenna.Rectangle('Center',[0,0],'Length',L,'Width',W,'Center',[-lambda*0.35,0]);
r3 = antenna.Rectangle('Length',0.5*lambda,'Width',0.02*lambda,'NumPoints',2);
s = r1 + r2 + r3;
```

```
figure
show(s)
```



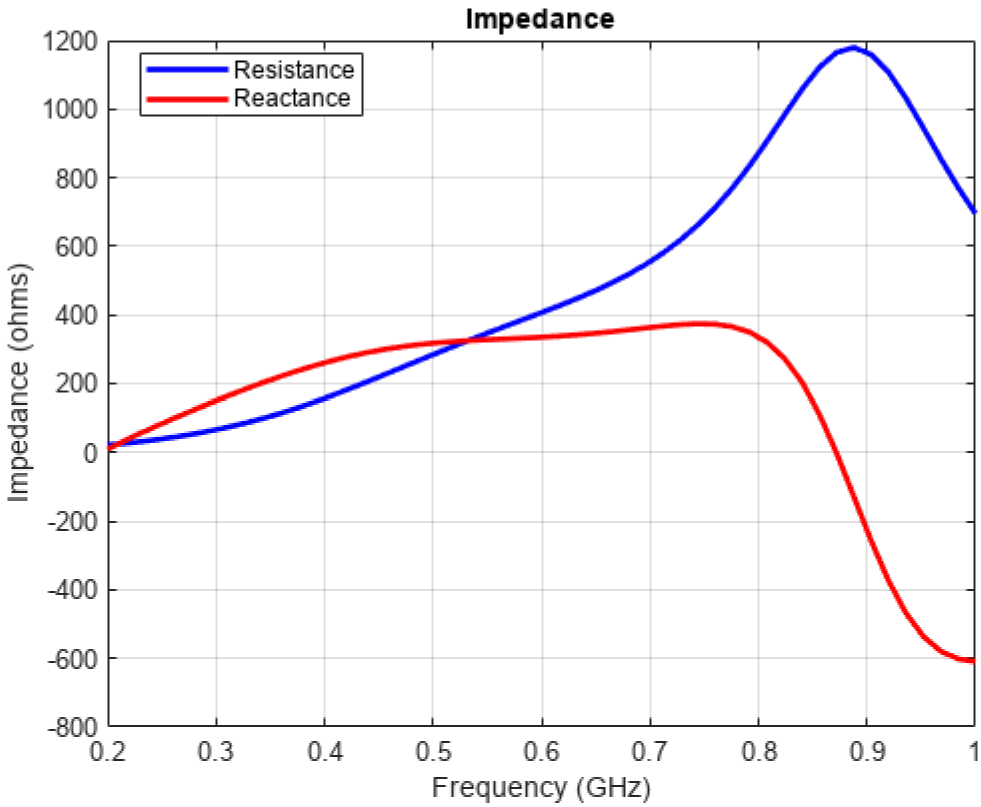
Assign the radiator shape to pcbStack and make the changes to the board shape and feed diameter properties.

```
boardShape = antenna.Rectangle('Length',0.6,'Width',0.3);
p = pcbStack;
p.BoardShape = boardShape;
p.Layers = {s};
p.FeedDiameter = .02*lambda/2;
p.FeedLocations = [0 0 1];
figure
show(p)
```



Analyze the impedance of the antenna. Effect of the end-loading should result in the series resonance to be pushed lower in the band.

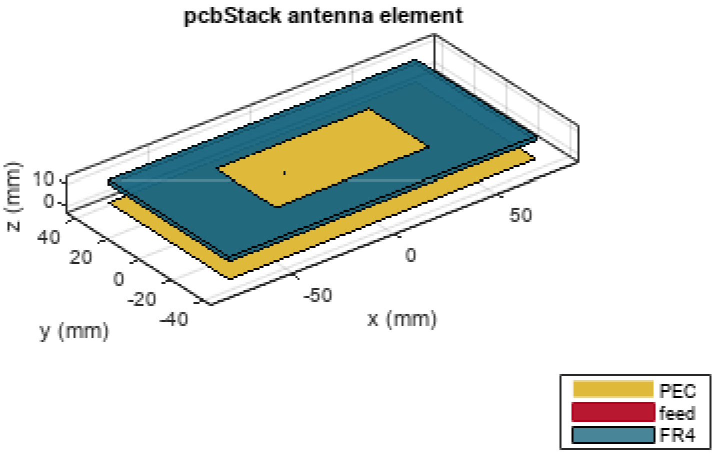
```
figure
impedance(p,linspace(200e6,1e9,51))
```



PCB Stack of Dielectric Antenna

Create a pcb stack antenna with 2 mm dielectric thickness at the radiator and air below it. Display the structure.

```
p = pcbStack;
d1 = dielectric('FR4');
d1.Thickness = 2e-3;
d2 = dielectric('Air');
d2.Thickness = 8e-3;
p.Layers = {p.Layers{1},d1,d2,p.Layers{2}};
p.FeedLocations(3:4) = [1 4];
show(p)
```



Directivity Pattern of PCB Stack Antenna

Create a PCB stack antenna from reflector backed bowtie.

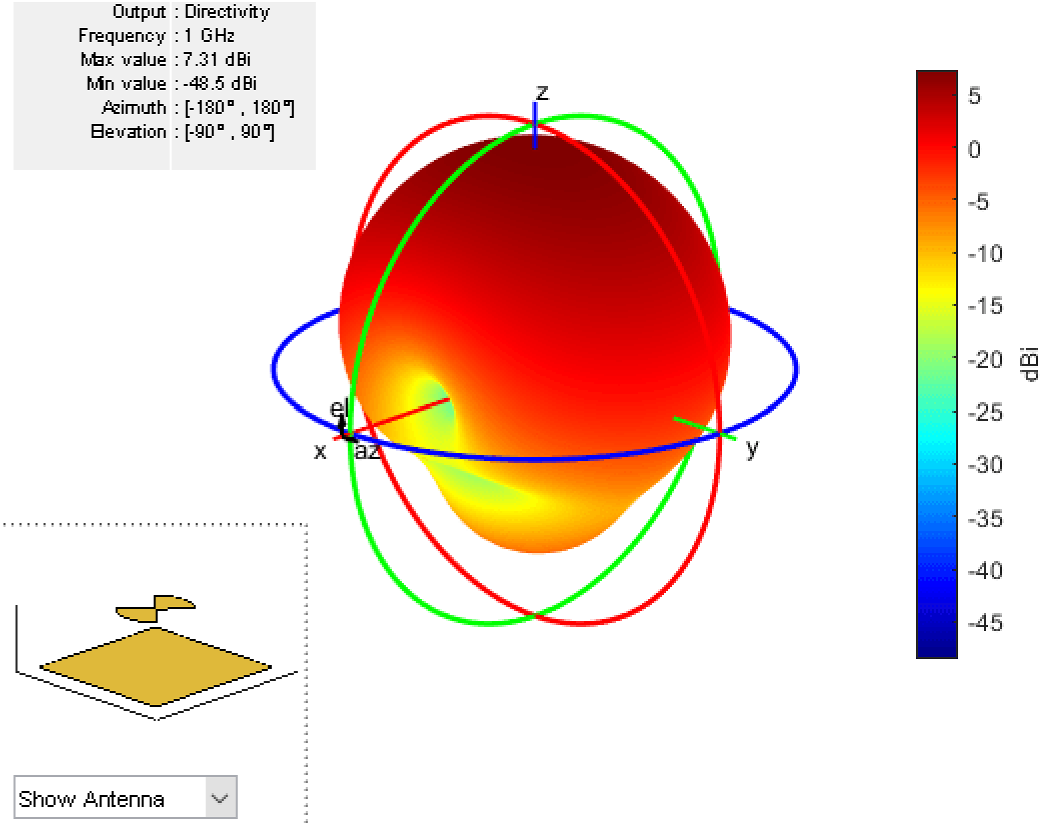
```
b = design(bowtieRounded,1e9);
b.Tilt = 90
```

```
b = 
  bowtieRounded with properties:

         Length: 0.0959
     FlareAngle: 90
      Conductor: [1x1 metal]
           Tilt: 90
      TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

```
b.TiltAxis = [0 1 0];
r = reflector('Exciter',b);
p = pcbStack(r);
```

Plot the directivity pattern of the antenna at 1 GHz.

```
pattern(p,1e9);
```



Output : Directivity
Frequency : 1 GHz
Max value : 7.31 dBi
Min value : -48.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

PCB Antenna From Antenna Library Elements

Create a coplanar inverted F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Use this antenna to create a `pcbStack` object.

```
p = pcbStack(fco);
```

Stack Conversion

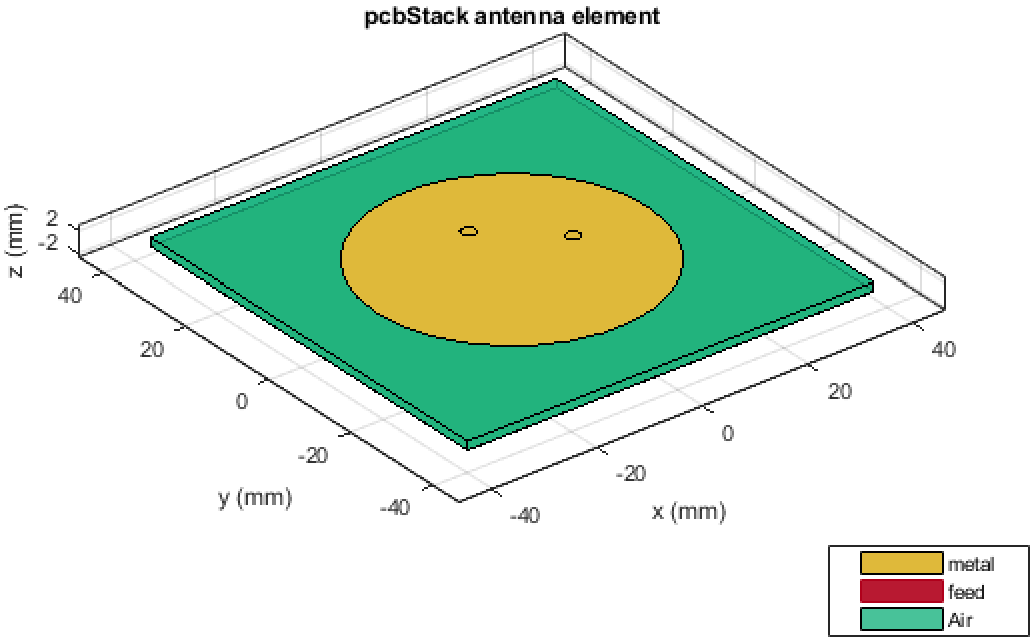
Create a circular microstrip patch.

```
p = patchMicrostripCircular;
d = dielectric;
d.EpsilonR = 4.4;
p.Radius = .0256;
p.Height = 1.6e-3;
p.Substrate = d;
p.GroundPlaneLength = 3*.0256;
```

```
p.GroundPlaneWidth = 3*.0256;
p.FeedOffset = [.0116 0];
```

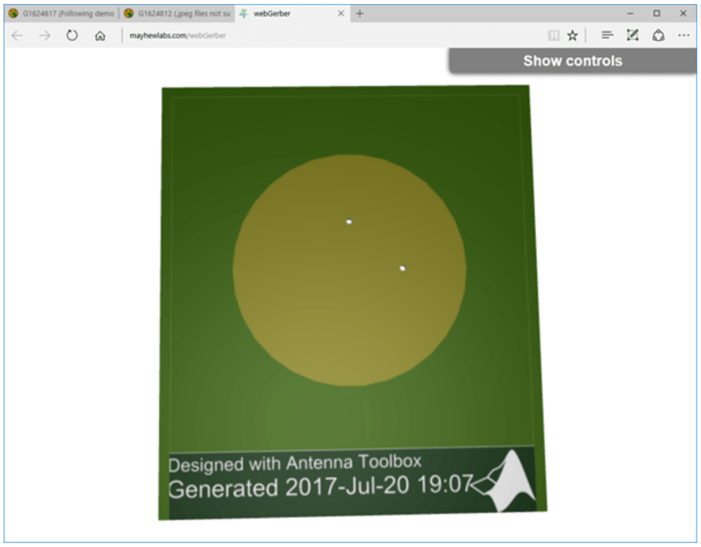
Create a PCB circular microstrip patch using `pcbStack`.

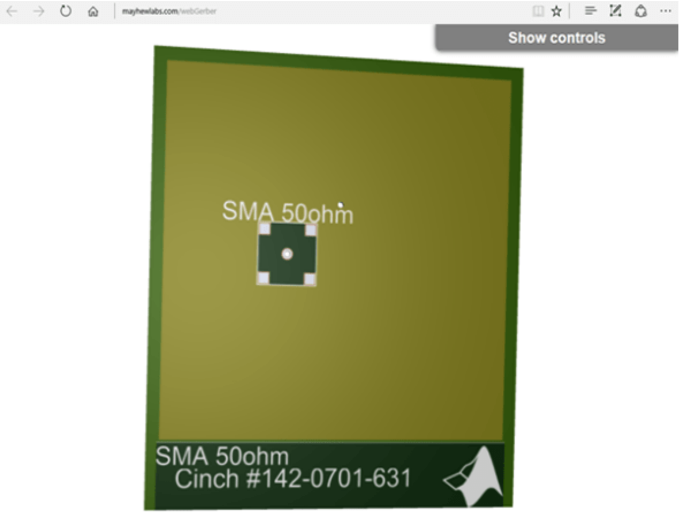
```
pb = pcbStack(p);
pb.FeedDiameter = 1.27e-3;
pb.ViaLocations = [0 pb.FeedLocations(1)/1.1 1 3];
pb.ViaDiameter = pb.FeedDiameter;
figure
show(pb)
```



```
C = SMA_Jack_Cinch;
O = PCBServices.MayhewWriter;
O.DefaultViaDiam = pb.ViaDiameter;
O.Filename = 'Microstrip circular patch-9a';
Am = PCBWriter(pb,O,C);
gerberWrite(Am)
```

Images using Mayhew Labs 3-D Viewer.





PCB Antenna from Antenna Array Library Elements

Create a coplanar inverted-F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Create a linear array with inverted-F antenna as its elements.

```
la = linearArray;
la.Element = fco;
la.NumElements = 4;
```

Use this antenna array to create the PCB antenna.

```
p = pcbStack(la);
```

PCB Stack from Linear Dipole Array and Dipole Antenna Element

Create a `dipole` antenna object and `linearArray` antenna array object. In the `linearArray` antenna object, leave the `Element` property set to its default value of dipole. Set the `ElementSpacing` property to 4."

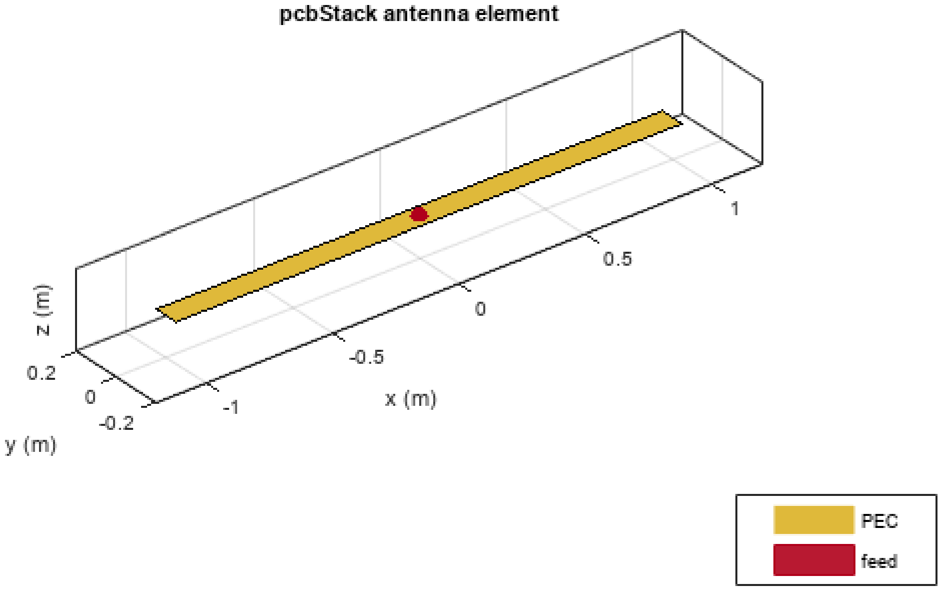
```
d1 = dipole;
d2 = linearArray('ElementSpacing', 4);
```

To set the Z-coordinate of `pcbStack` antenna object to zero, rotate the dipole and linear dipole array around 90 degrees using the `Tilt` property. Then set the `TiltAxis` property to [0 -1 0] for dipole and linear dipole array antennas.

```
d1.Tilt = 90;
d2.Element.Tilt = 90;
d1.TiltAxis = [0 -1 0];
d2.Element.TiltAxis = [0 -1 0];
```

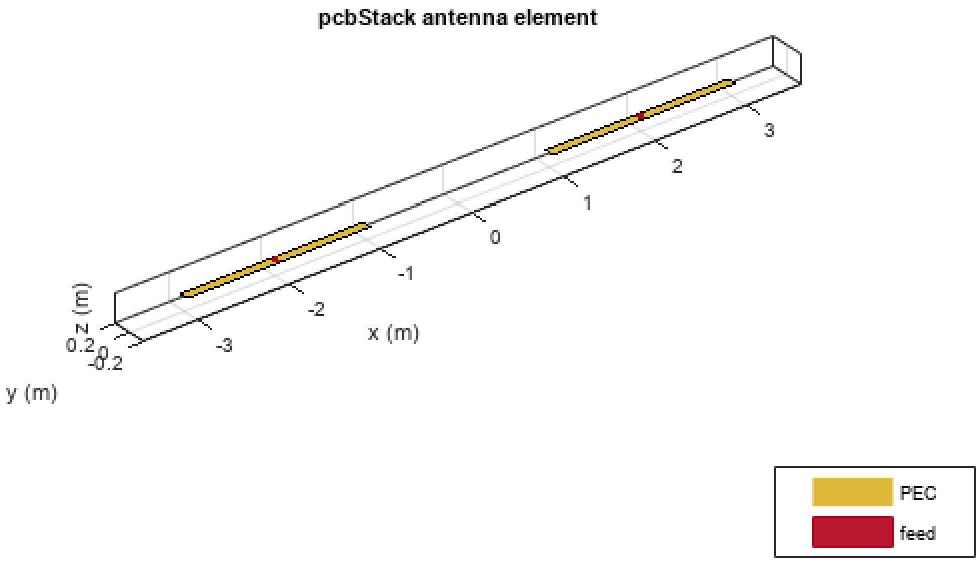
Create and view PCB stack antenna created using the `dipole` antenna object.

```
p1 = pcbStack(d1);
show(p1)
```



Create and view PCB stack antenna using the `linearArray` antenna array object.

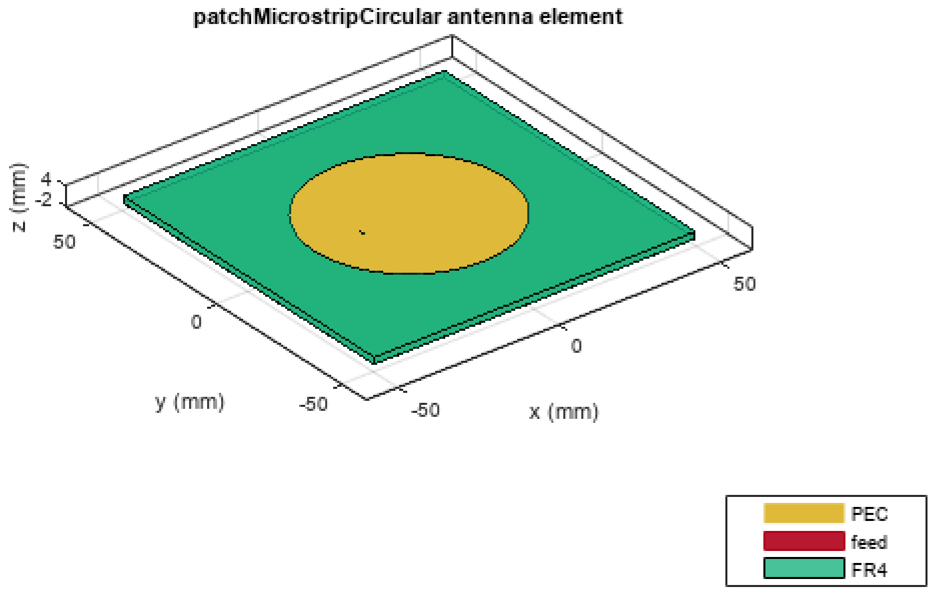
```
p2 = pcbStack(d2);
show(p2)
```



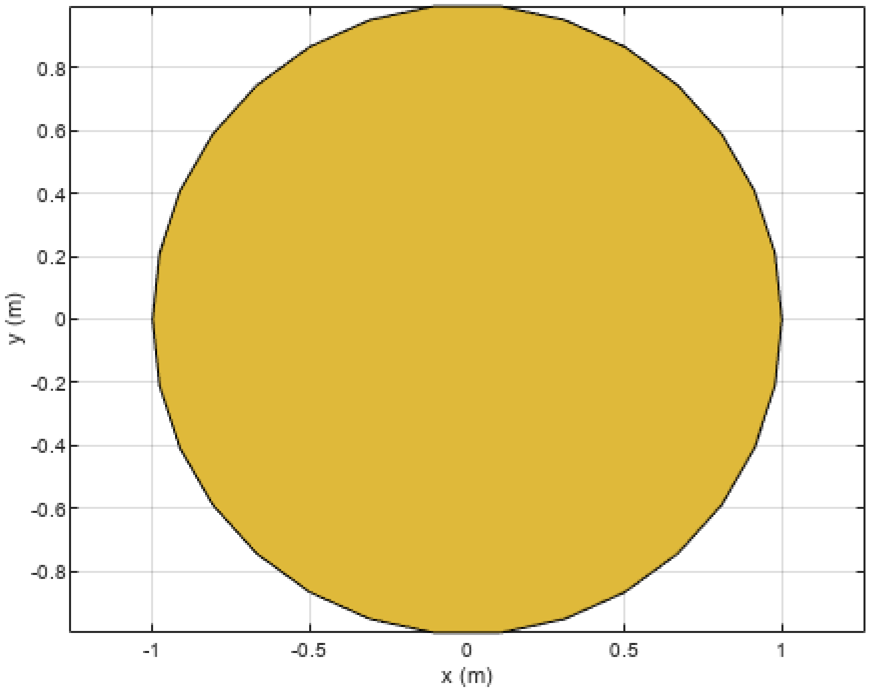
Circular Microstrip Patch Antenna on Polygon Shaped Board

Create a circular microstrip patch antenna.

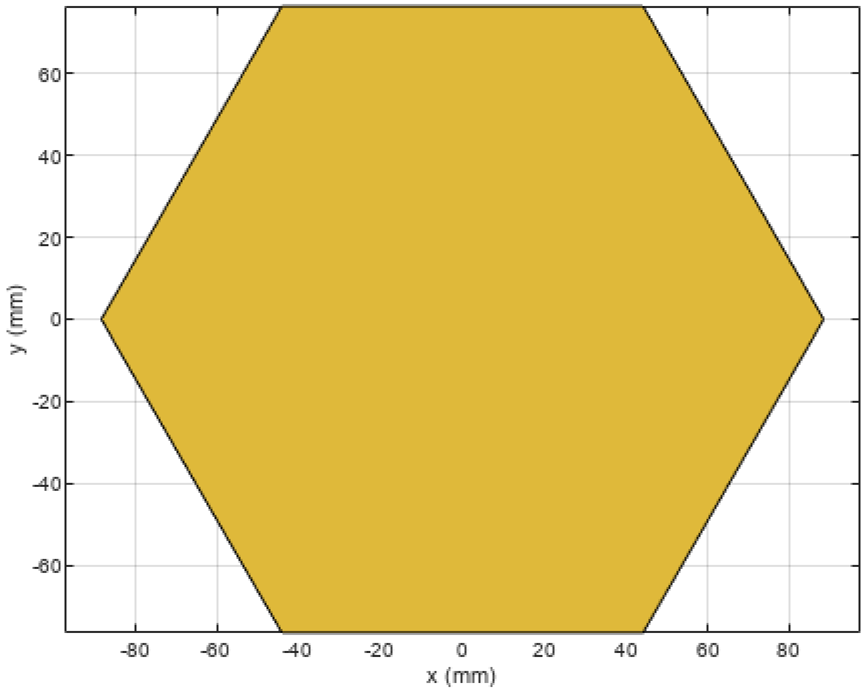
```
ant = design(patchMicrostripCircular,3e9);
ant.Substrate = dielectric( 'FR4' );
show(ant)
```



```
c = antenna.Circle;
show(c)
```

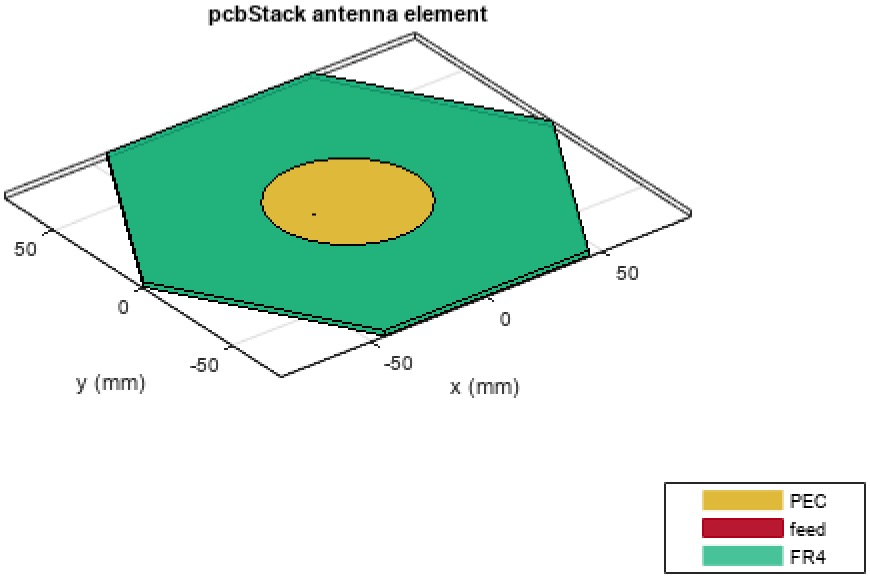


```
c.NumPoints = 6;
c.Radius = 3*ant.Radius;
figure
show(c)
```



Create the PCB stack using the vertices derived from the circle shape.

```
v = getShapeVertices(c);
cp = antenna.Polygon( 'Vertices' ,v);
pb = pcbStack(ant);
pb.Layers{3} = cp;
pb.BoardShape = cp;
show(pb)
axis equal
```



Version History

Introduced in R2017a

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

[2] Stutzman, W. L. and Gary A. Thiele. *Antenna Theory and Design*. 3rd Ed. River Street, NJ: John Wiley & Sons, 2013.

See Also

`customAntennaMesh` | `customArrayMesh` | `antenna.Circle` | `antenna.Polygon` | `antenna.Rectangle`

Topics

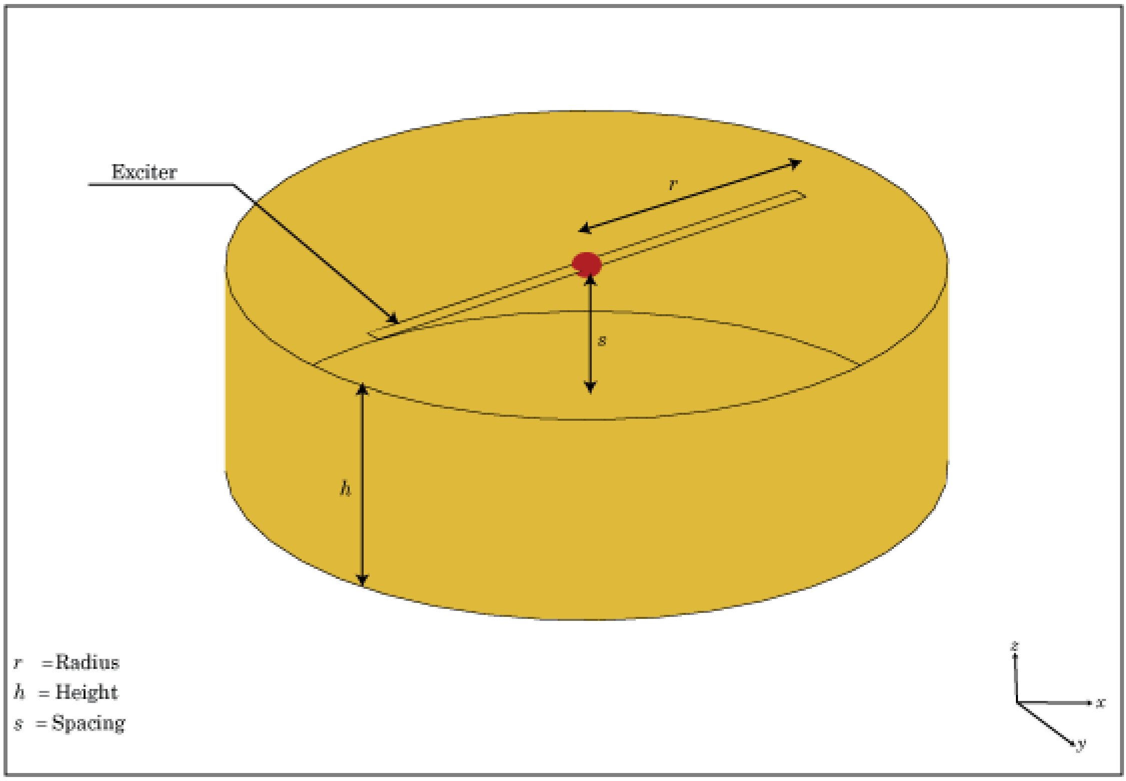
"Design Variations On Microstrip Patch Antenna Using PCB Stack"
"Rotate Antennas and Arrays"

cavityCircular

Create circular cavity-backed antenna

Description

Use the `circularCavity` object to create a circular cavity-backed antenna. By default, the exciter used is a dipole. The dimensions are chosen for an operating frequency of 1 GHz.



Creation

Syntax

```
circularcavity = cavityCircular
circularcavity = cavityCircular(Name,Value)
```

Description

`circularcavity = cavityCircular` creates a circular cavity-backed antenna.

`circularcavity = cavityCircular(Name,Value)` sets properties using one or more name-value pairs. For example, `circularcavity = cavityCircular('Radius',0.2)` creates a circular cavity of radius 0.2 m. Enclose each property name in quotes.

Properties

Exciter — Antenna or array type used as exciter

dipole (default) | antenna object | array object

Antenna type used as an exciter, specified as any single-element antenna object. Except reflector and cavity antenna elements, you can use any of the antenna elements or array elements in the Antenna Toolbox as an exciter.

Example: 'Exciter',horn

Example: ant.Exciter = horn

Example: ant.Exciter = linearArray('patchMicrostrip')

Radius — Cavity radius

0.1000 (default) | scalar

Radius of cavity, specified as a scalar in meters.

Example: 'Radius',0.2

Example: circularcavity.Radius = 0.2

Data Types: double

Height — Cavity height along *z*-axis

0.0750 (default) | scalar

Cavity height along z-axis, specified as a scalar in meters.

Example: 'Height',0.001

Example: circularcavity.Height = 0.001

Data Types: double

Spacing — Distance between exciter and base of cavity

0.0750 (default) | scalar

Distance between the exciter and the base of the cavity, specified a scalar in meters.

Example: 'Spacing',7.5e-2

Example: circularcavity.Spacing = 7.5e-2

Data Types: double

Substrate — Type of dielectric material

'Air' (default) | object

Type of dielectric material used as a substrate, specified as a object. For more information see, dielectric. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the ground plane dimensions.

Example: d = dielectric('FR4'); 'Substrate',d

Example: d = dielectric('FR4'); circularcavity.Substrate = d

EnableProbeFeed — Create probe feed from backing structure to exciter

0 (default) | 1

Create probe feed from backing structure to exciter, specified as 0 or 1 or a positive scalar. By default, probe feed is not enabled.

Example: 'EnableProbeFeed',1

Example: circularcavity.EnableProbeFeed = 1

Data Types: double | logical

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: circularcavity.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.

- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Circular Cavity-Backed Antenna

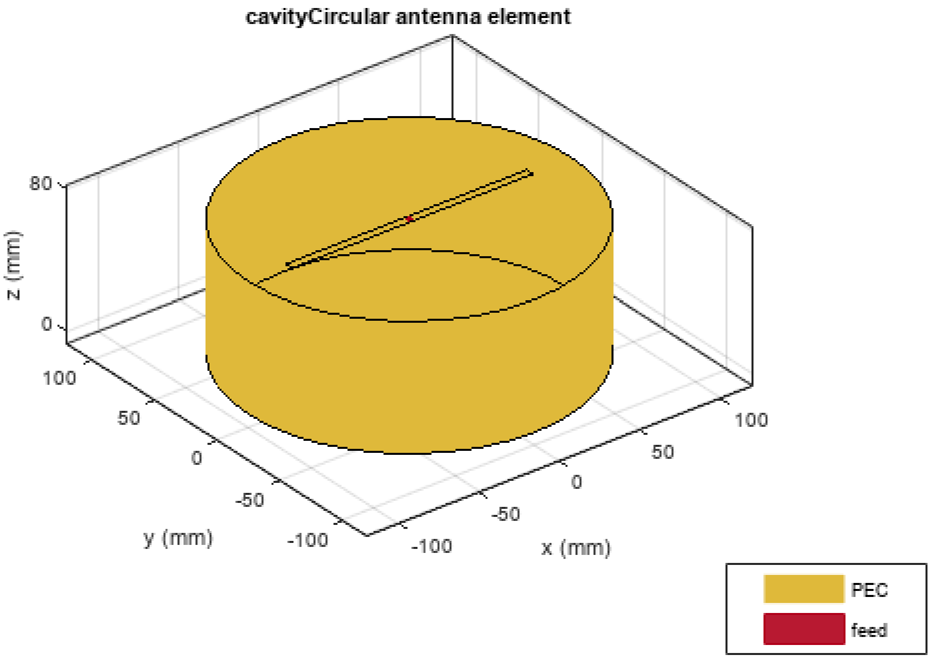
Create and view a default circular cavity-backed antenna.

```
a = cavityCircular
```

```
a = 
  cavityCircular with properties:

            Exciter: [1x1 dipole]
```

```
          Substrate: [1x1 dielectric]
             Radius: 0.1000
             Height: 0.0750
            Spacing: 0.0750
    EnableProbeFeed: 0
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(a)
```



Circular Cavity-Backed Equiangular Spiral

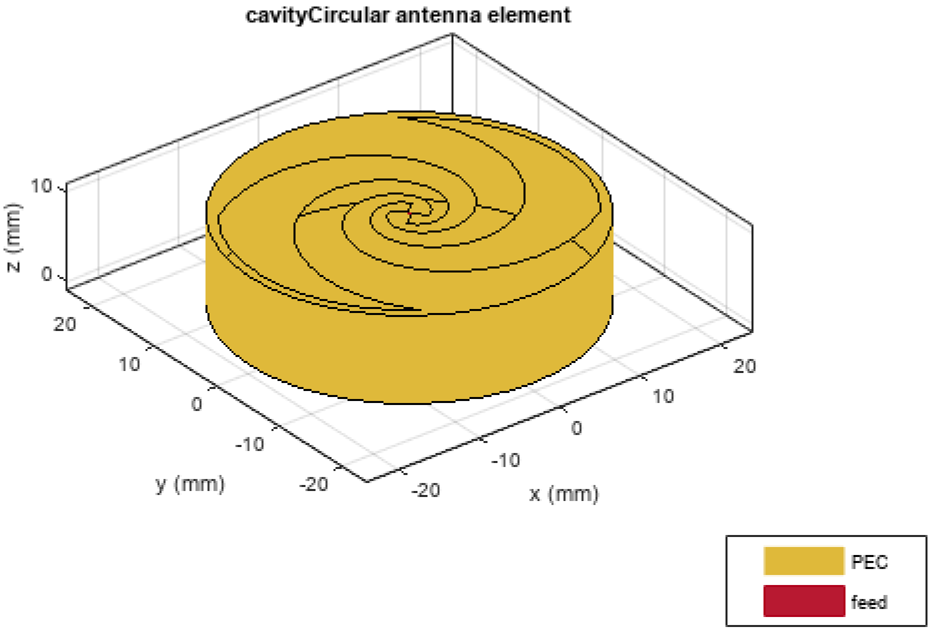
Create and view an equiangular spiral backed by a circular cavity. The cavity dimensions are:

Radius = 0.02 m

Height = 0.01 m

Spacing = 0.01 m

```
ant = cavityCircular('Exciter',spiralEquiangular,'Radius',0.02,    ...
         'Height',0.01,'Spacing', 0.01);
show(ant)
```



Create Circular Cavity-Backed Linear Array

Create a linear array of H-shaped patch microstrip antenna.

```
arr = linearArray('Element',patchMicrostripHnotch,'ElementSpacing',0.04);
```

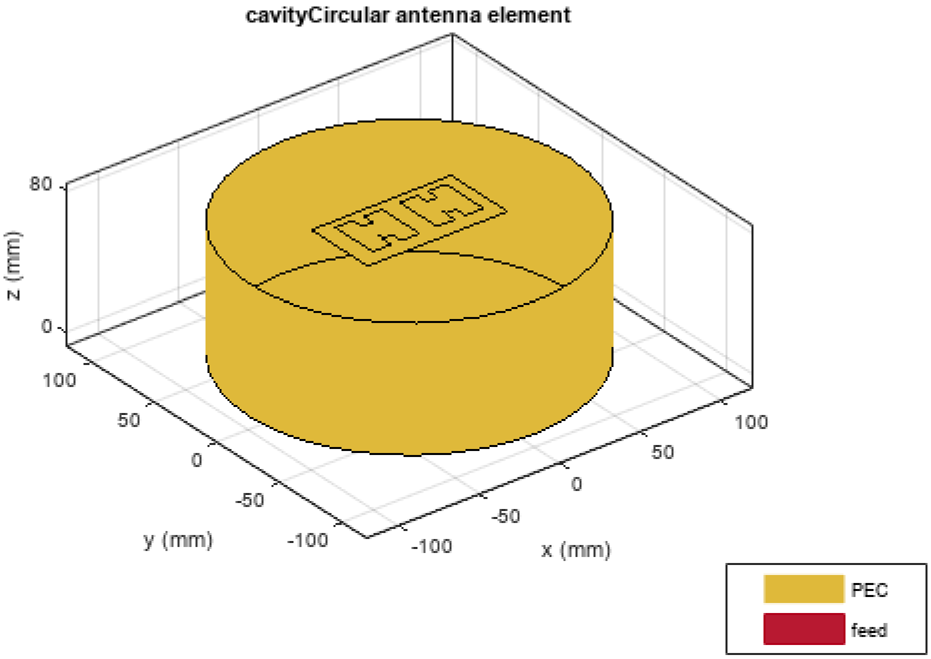
Create a circular cavity-backed antenna with linear array exciter.

```
ant = cavityCircular('Exciter',arr)
```

```
ant = 
  cavityCircular with properties:

            Exciter: [1x1 linearArray]
          Substrate: [1x1 dielectric]
             Radius: 0.1000
             Height: 0.0750
            Spacing: 0.0750
    EnableProbeFeed: 0
          Conductor: [1x1 metal]
               Tilt: 0
```

```
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(ant)
```



Create Cylindrical Dielectric Resonator Antenna with Circular Cavity Backing Structure

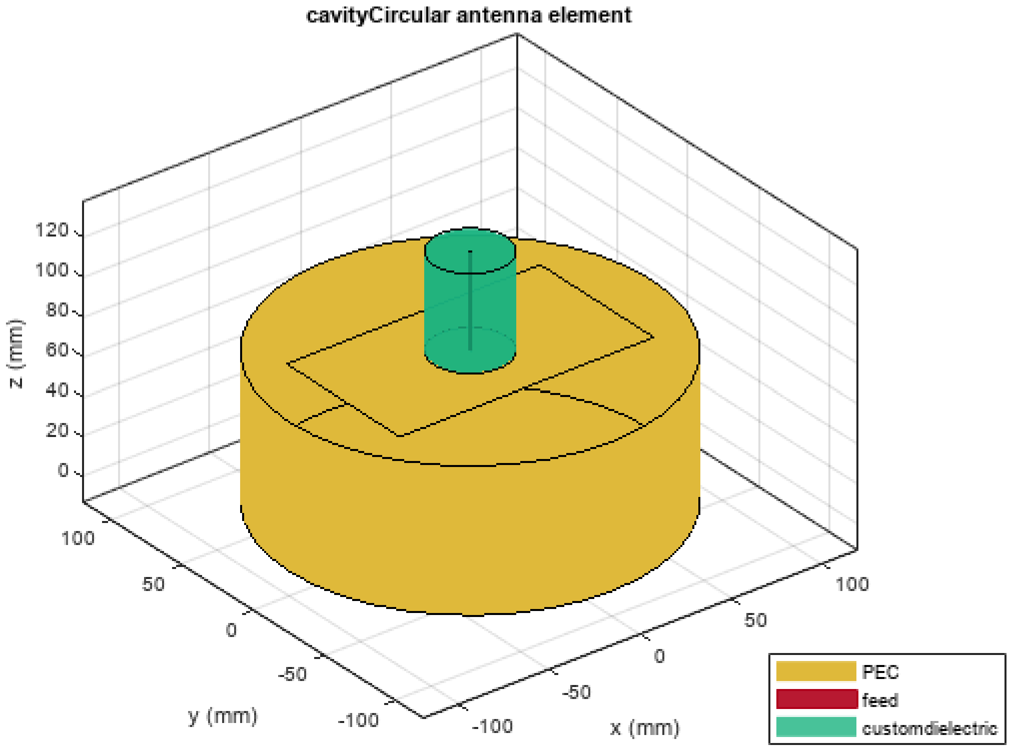
Create and visualize a circular cavity-backed cylindrical dielectric resonator antenna.

```
e = draCylindrical;
ant = cavityCircular('Exciter',e)
```

```
ant = 
  cavityCircular with properties:

            Exciter: [1x1 draCylindrical]
          Substrate: [1x1 dielectric]
             Radius: 0.1000
             Height: 0.0750
            Spacing: 0.0750
    EnableProbeFeed: 0
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
```

```
                Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2017b

See Also

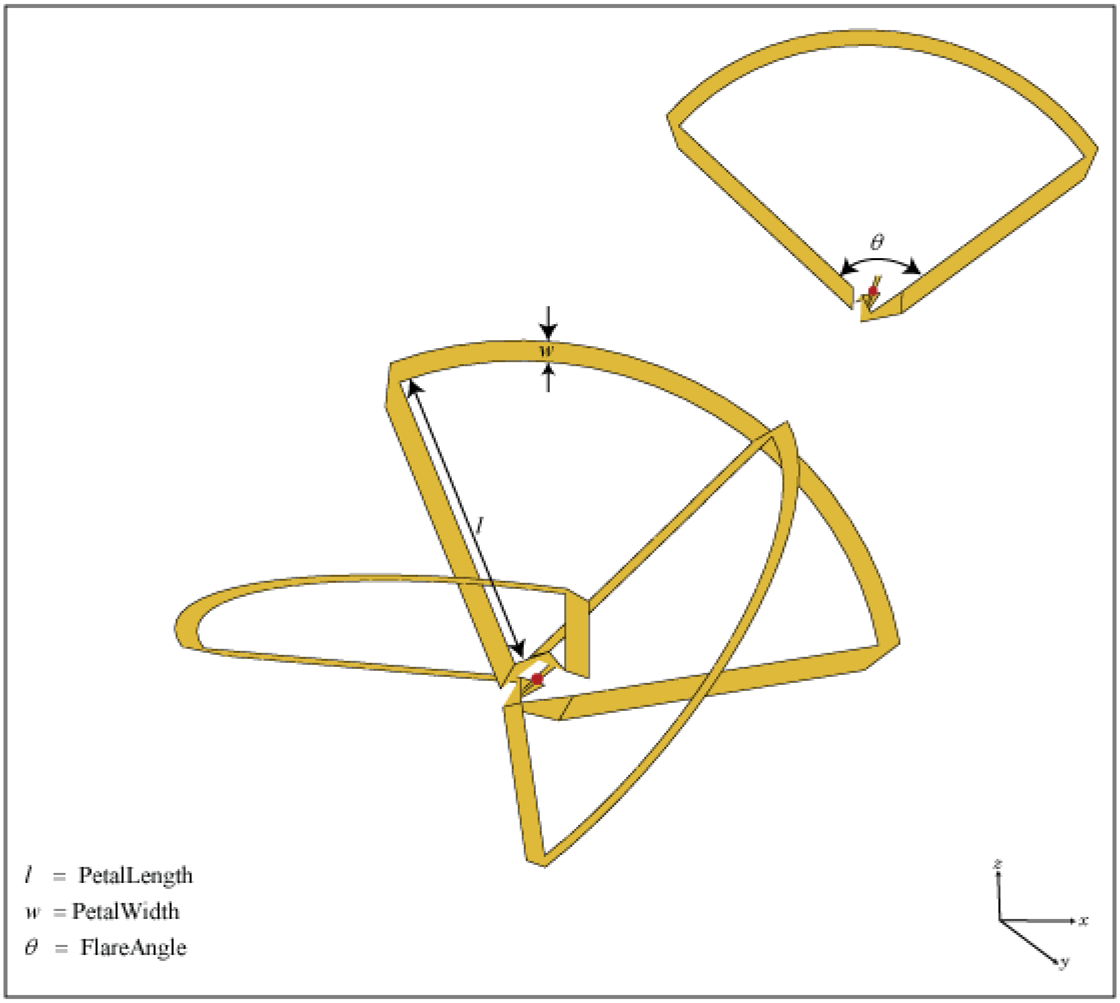
cavity | reflector | reflectorCircular

cloverleaf

Create three-petal cloverleaf antenna

Description

Use the `cloverleaf` object to create a three-petal cloverleaf antenna. The default cloverleaf has 3 petals and operates at around 5.8 GHz. It has a wideband circular polarization and an omnidirectional antenna.



Creation

Syntax

```
cl = cloverleaf
cl = cloverleaf(Name,Value)
```

Description

`cl = cloverleaf` creates a three-petal cloverleaf antenna.

`cl = cloverleaf(Name,Value)` sets properties using one or more name-value pairs. For example, `cl = cloverleaf('NumPetals',4)` creates a five-petal cloverleaf antenna. Enclose each property name in quotes.

Properties

`NumPetals` — Number of petals

`3` (default) | scalar

Number of petals, specified as a scalar.

Example: `'NumPetals',4`

Example: `cl.NumPetals = 4`

Data Types: `double`

`PetalLength` — Total length of leaf

`0.0515` (default) | scalar

Total length of leaf, specified as a scalar in meters.

Example: `'PetalLength',0.0025`

Example: `cl.PetalLength = 0.0025`

Data Types: `double`

`PetalWidth` — Leaf strip width

`8.0000e-04` (default) | scalar

Leaf strip width, specified as a scalar in meters.

Example: `'PetalWidth'`,`0.001`

Example: `cl.PetalWidth = 0.001`

Data Types: `double`

`FlareAngle` — Leaf flare angle

`105` (default) | scalar

Leaf flare angle, specified as a scalar in degrees.

Example: `'FlareAngle'`,`100`

Example: `cl.FlareAngle = 100`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, it is at the origin. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: cl.Load = lumpedElement('Impedance',75)

Data Types: double

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: TiltAxis=[0 1 0]

Example: TiltAxis=[0 0 0;0 1 0]

Example: TiltAxis = 'Z'

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

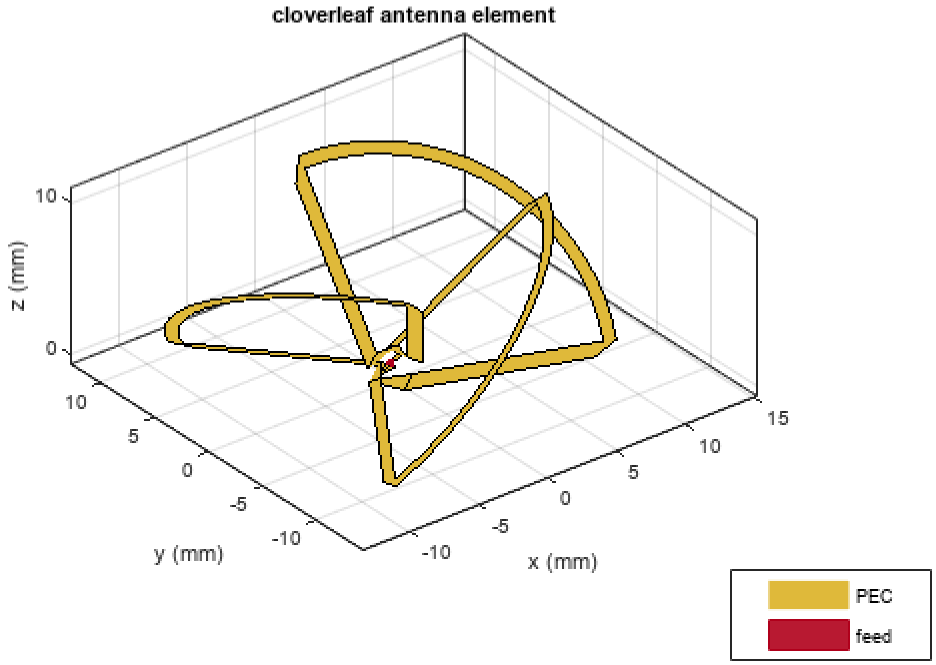
Clover Leaf Antenna

Create and view a default cloverleaf antenna.

```
cl = cloverleaf
```

```
cl = 
  cloverleaf with properties:

      NumPetals: 3
    PetalLength: 0.0515
     PetalWidth: 8.0000e-04
     FlareAngle: 105
      Conductor: [1x1 metal]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

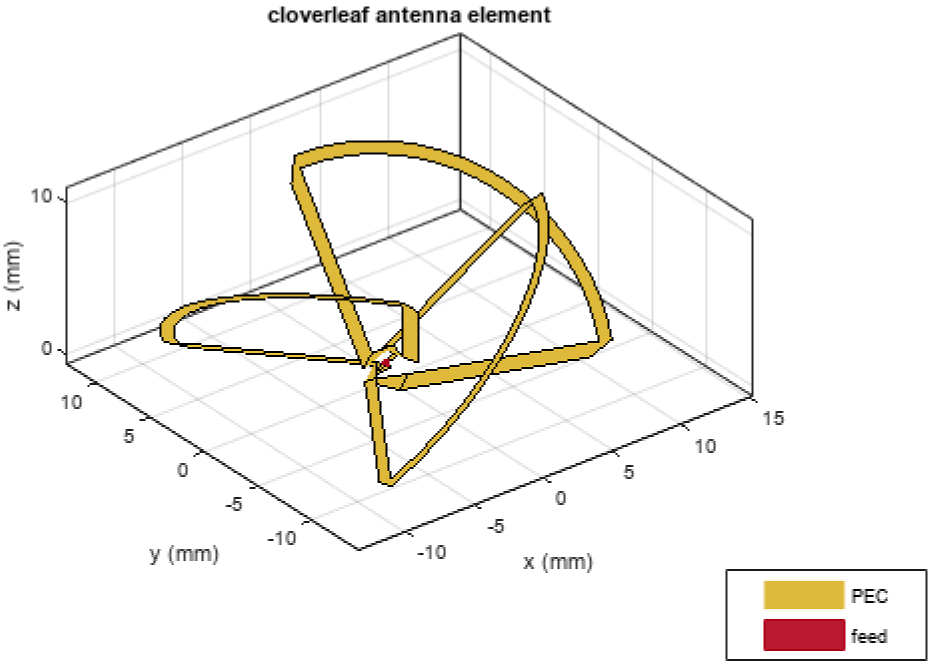
```
show(cl)
```



Axial Ratio of Cloverleaf Antenna

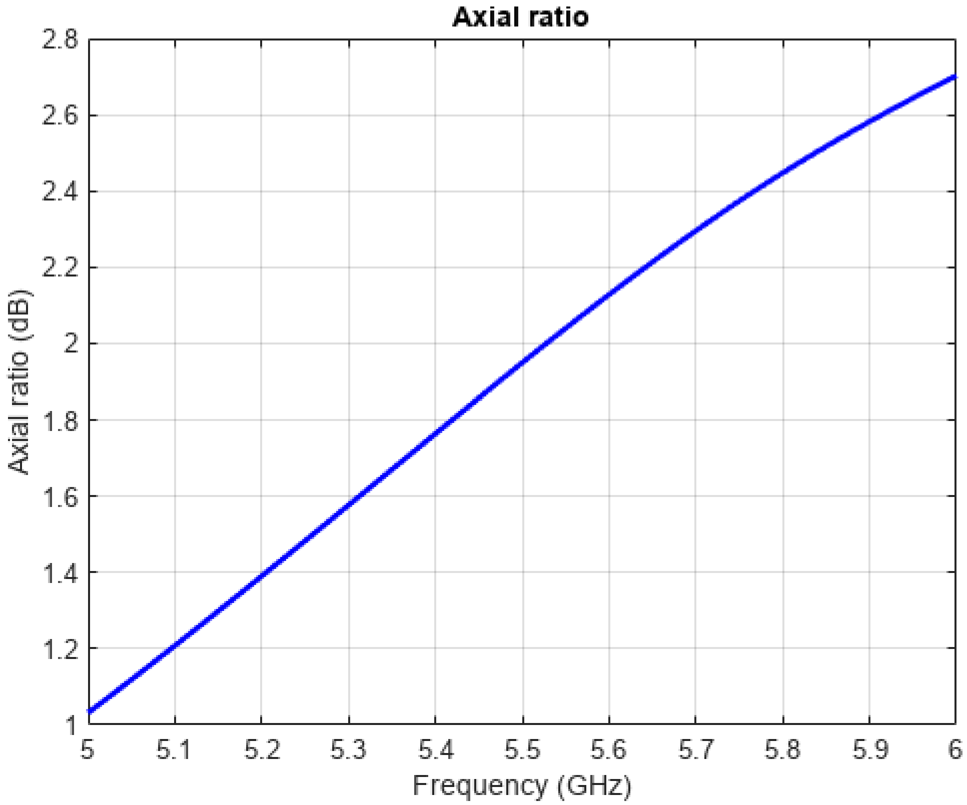
Create a cloverleaf antenna.

```
cl = cloverleaf;
show(cl);
```



Plot the axial ratio of the antenna from 5 GHz to 6 GHz.

```
freq = linspace(5e9,6e9,101);
axialRatio(cl,freq,0,0);
```



The axial ratio plot shows that the antenna supports circular polarization over the entire frequency range.

Version History

Introduced in R2017b

See Also

`spiralArchimedean` | `dipole`

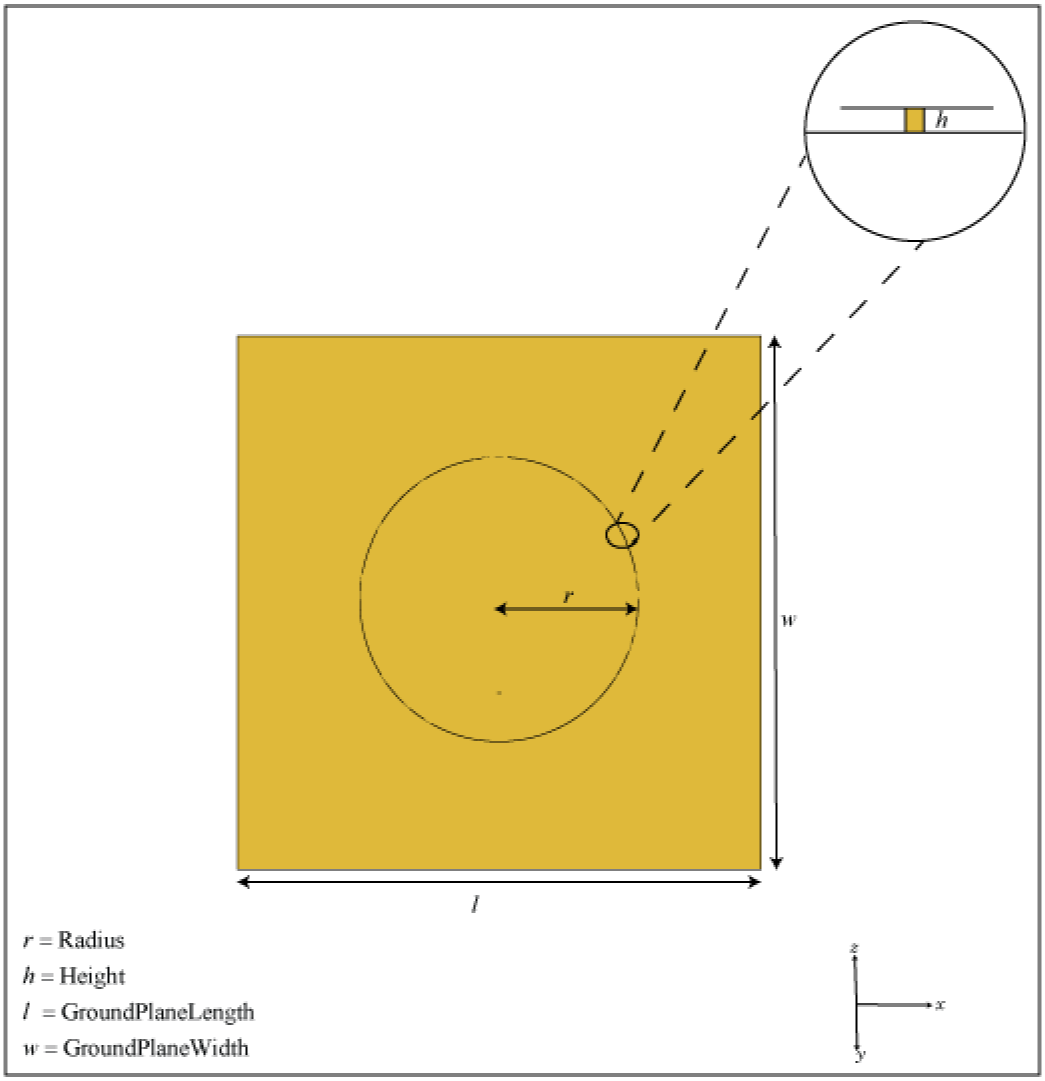
patchMicrostripCircular

Create probe-fed circular microstrip patch antenna

Description

Use the `patchMicrostripCircular` object to create a probe-fed circular microstrip patch antenna. By default, the patch is centered at the origin with feed point along the radius and the groundplane on the *xy*- plane at z = 0.

Circular microstrip antennas are used as low-profile antennas in airborne and spacecraft applications. These antennas also find use in portable wireless applications because they are lightweight, low cost, and easily manufacturable.



Creation

Syntax

```
circularpatch = patchMicrostripCircular
circularpatch = patchMicrostripCircular(Name,Value)
```

Description

`circularpatch = patchMicrostripCircular` creates a probe-fed circular microstrip patch antenna.

`circularpatch = patchMicrostripCircular(Name,Value)` sets properties using one or more name-value pairs. For example, `circularpatch = patchMicrostripCircular('Radius',0.2)` creates a circular patch of radius 0.2 m. Enclose each property name in quotes.

Properties

`Radius` — Patch radius
`0.0798` (default) | scalar

Patch radius, specified as a scalar in meters. The default radius is for an operating frequency of 1 GHz.

Example: `'Radius',0.2`

Example: `circularpatch.Radius = 0.2`

Data Types: `double`

`Height` — Height of patch
`0.0060` (default) | scalar

Height of patch above the ground plane along the *z*-axis, specified as a scalar in meters.

Example: `'Height',0.001`

Example: `circularpatch.Height = 0.001`

Data Types: `double`

`GroundPlaneLength` — Ground plane length
0.3000 (default) | scalar

Ground plane length along the *x*-axis, specified as a scalar in meters. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',120e-3`

Example: `circularpatch.GroundPlaneLength = 120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width
0.3000 (default) | scalar

Ground plane width along the *y*-axis, specified as a scalar in meters. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',120e-3`

Example: `circularpatch.GroundPlaneWidth = 120e-3`

Data Types: `double`

`Substrate` — Type of dielectric material

'Air' (default) | `dielectric` function

Type of dielectric material used as a substrate, specified as a dielectric material object. You can choose any material from the `DielectricCatalog` or use your own dielectric material. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be lesser than the ground plane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); ant.Substrate = d`

`PatchCenterOffset` — Signed distance from center along length and width of ground plane

`[0 0]` (default) | two-element real vector

Signed distance from center along length and width of ground plane, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the patch relative to the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Example: `circularpatch.PatchCenterOffset = [0.01 0.01]`

Data Types: `double`

`FeedOffset` — Signed distance from center along length and width of ground plane

[-0.0525 0] (default) | two-element real vector

Signed distance from center along length and width of ground plane, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the feedpoint relative to the ground plane and patch.

Example: `'FeedOffset',[0.01 0.01]`

Example: `circularpatch.FeedOffset = [0.01 0.01]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes

axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

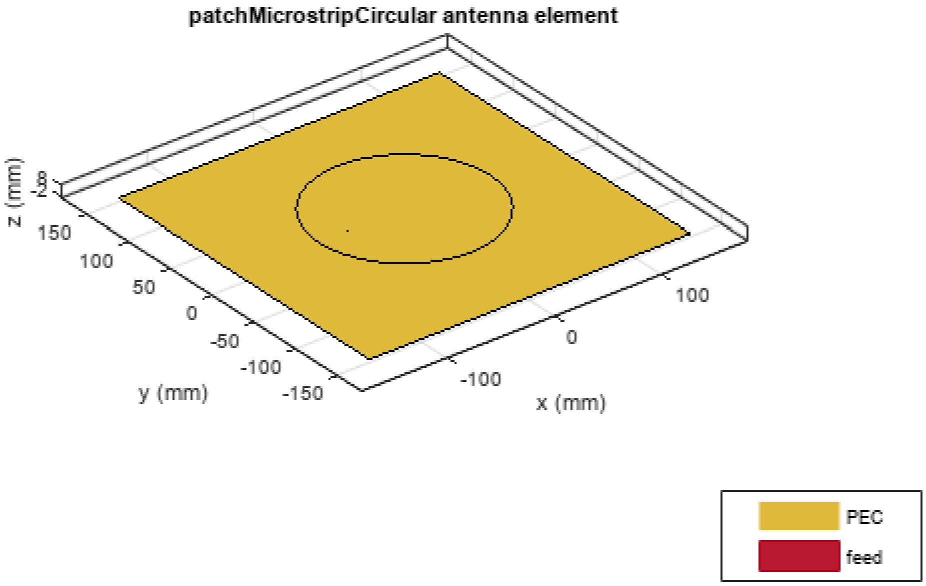
Circular Microstrip Patch

Create and view a default circular microstrip patch.

```
cp = patchMicrostripCircular
```

```
cp = 
  patchMicrostripCircular with properties:

               Radius: 0.0798
               Height: 0.0060
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.3000
     GroundPlaneWidth: 0.3000
    PatchCenterOffset: [0 0]
           FeedOffset: [-0.0525 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

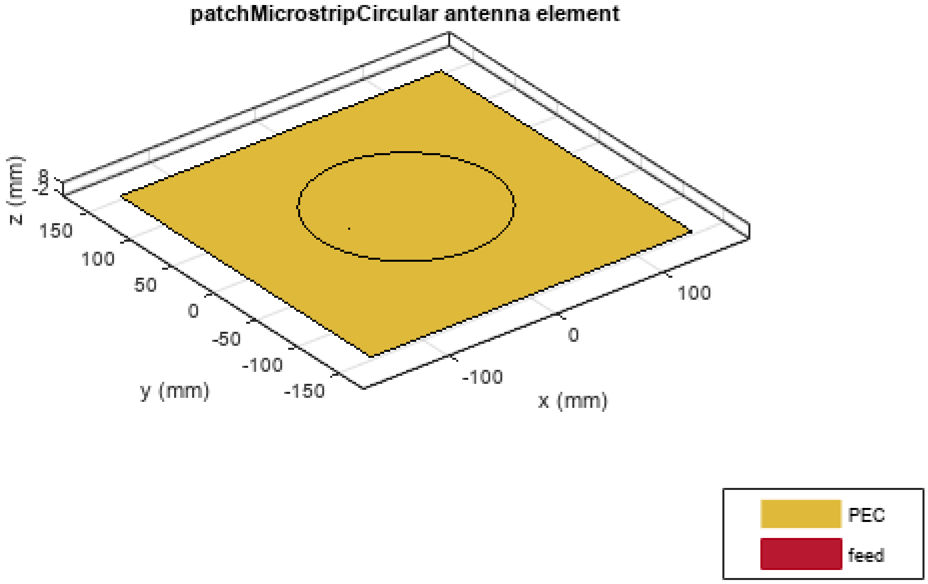
```
show(cp)
```



Radiation Pattern and Impedance of Circular Microstrip Patch

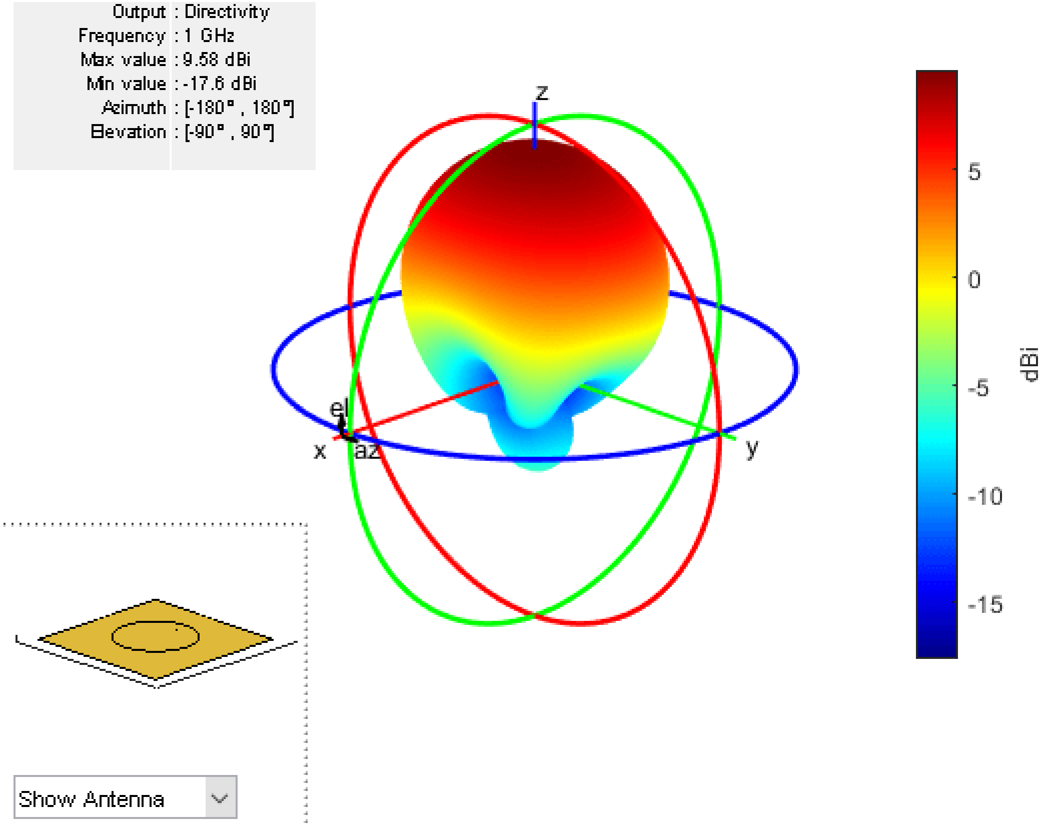
Create a circular patch antenna using given values. Display the antenna.

```
cp = patchMicrostripCircular('Radius',0.0798,'Height',6e-3,...
       'GroundPlaneLength',0.3,'GroundPlaneWidth',0.3,...
       'FeedOffset',[-0.0525 0]);

show(cp)
```



Plot the pattern of the patch antenna at 1 GHz.

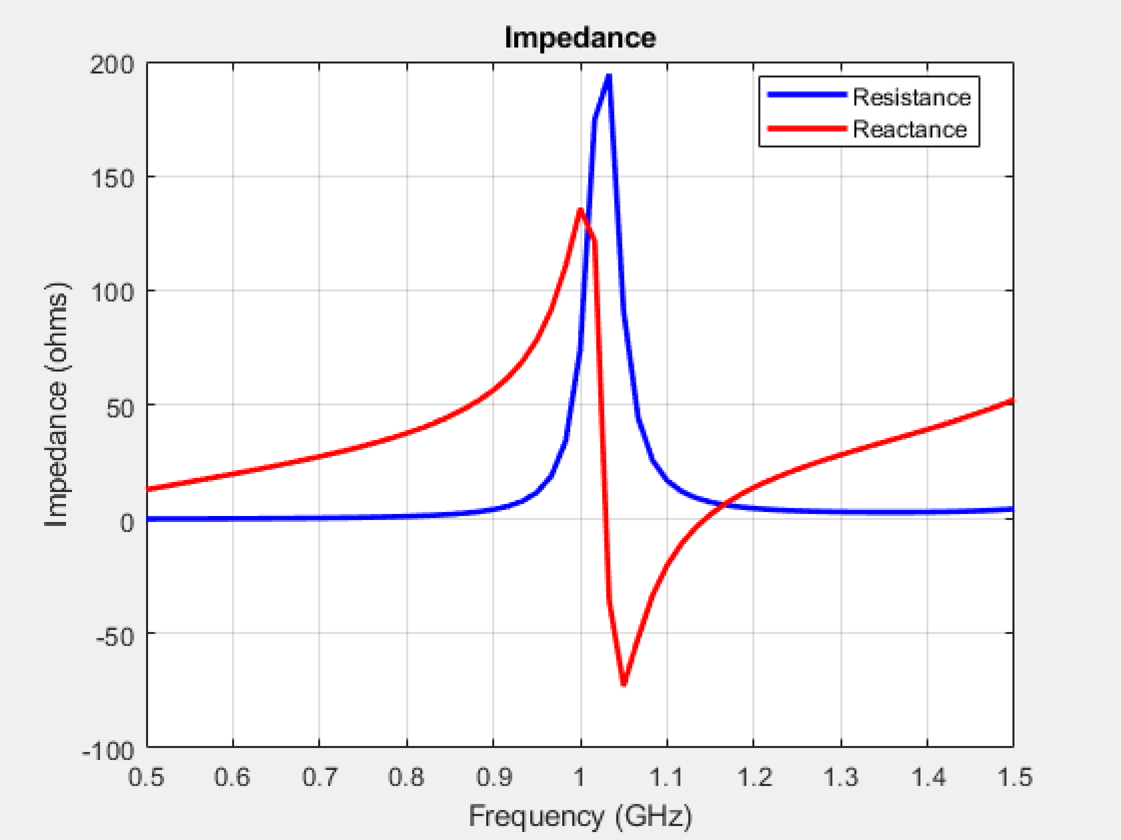
```
pattern(cp,1e9);
```



Output : Directivity
Frequency : 1 GHz
Max value : 9.58 dBi
Min value : -17.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Calculate the impedance of the antenna over a frequency span of 0.5 GHz to 1.5 GHz.

```
f = linspace(0.5e9,1.5e9,61);
impedance(cp,f);
```



Version History

Introduced in R2017b

See Also

`patchMicrostrip` | `patchMicrostripInsetfed`

Topics

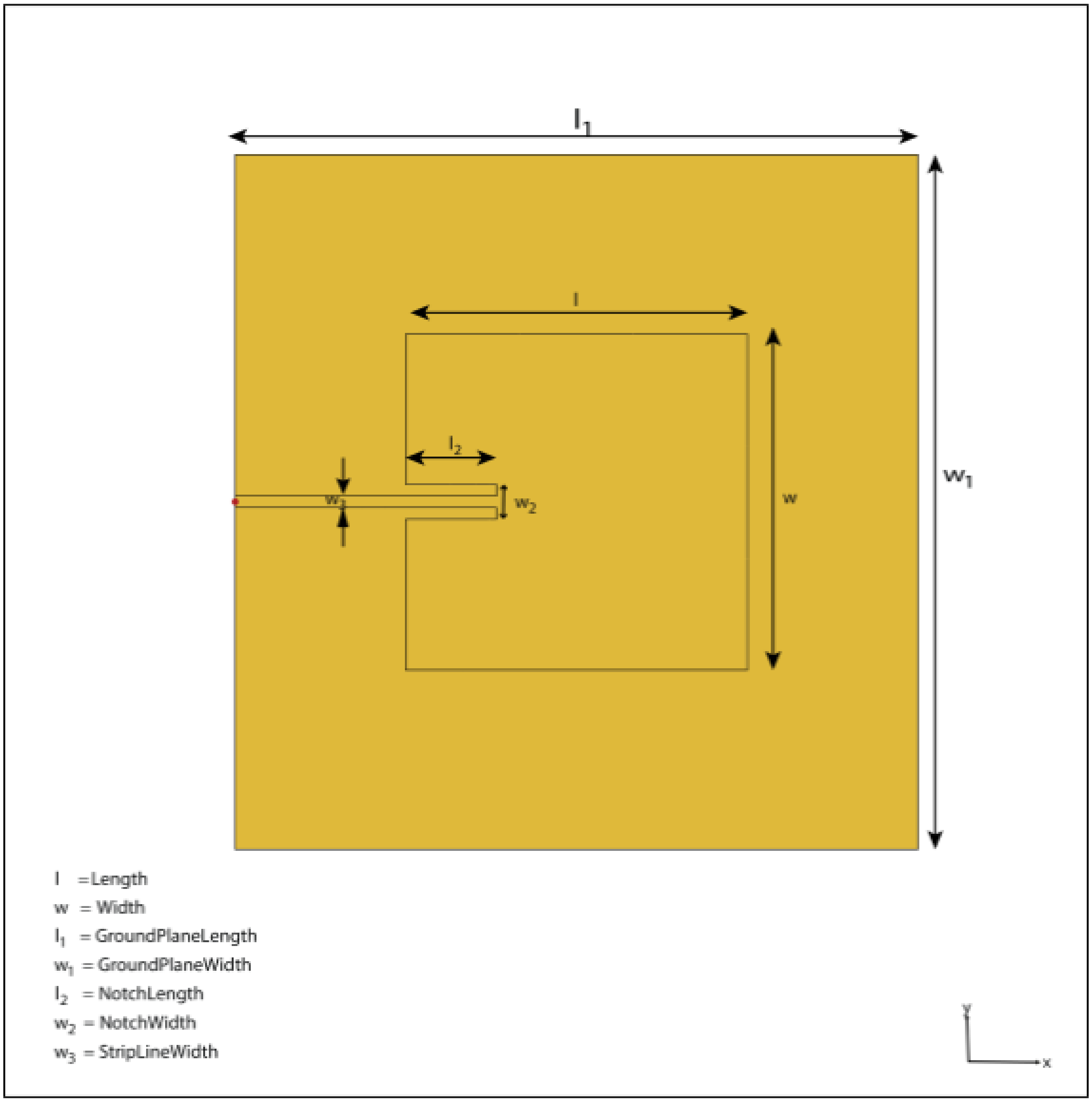
"ISM Band Patch Microstrip Antennas and Mutually Coupled Patches"

patchMicrostripInsetfed

Create inset-fed microstrip patch antenna

Description

Use the `patchMicrostripInsetfed` object to create an inset-fed microstrip patch antenna. The default patch is centered at the origin.



Creation

Syntax

```
insetpatch = patchMicrostripInsetfed
insetpatch = patchMicrostripInsetfed(Name,Value)
```

Description

`insetpatch = patchMicrostripInsetfed` creates an inset-fed microstrip patch antenna centered at the origin.

`insetpatch = patchMicrostripInsetfed(Name,Value)` sets properties using one or more name-value pair. For example, `insetpatch = patchMicrostripInsetfed('Length',0.2)` creates an inset-fed patch of length 0.2 m. Enclose each property name in quotes.

Properties

`Length` — Patch length along *x*-axis
`0.0300` (default) | scalar

Patch length along *x*-axis, specified as a scalar in meters. The default length is for an operating frequency of 4.5 GHz.

Example: `'Length',0.2`

Example: `insetpatch.Length = 0.2`

Data Types: `double`

`Width` — Patch width along *y*-axis
`0.0290` (default) | scalar

Patch width along *y*-axis, specified as a scalar in meters.

Example: `'Width',0.1`

Example: `insetpatch.Width = 0.1`

Data Types: `double`

`Height` — Patch height along *z*-axis
`0.0013` (default) | scalar

Patch height along *z*-axis, specified as a scalar in meters.

Example: `'Height',0.001`

Example: `insetpatch.Height = 0.001`

Data Types: `double`

`GroundPlaneLength` — Ground plane length along *x*-axis
0.0600 (default) | scalar

Ground plane length along *x*-axis, specified as a scalar in meters. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',120e-3`

Example: `insetpatch.GroundPlaneLength = 120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

0.0600 (default) | scalar

Ground plane width along *y*-axis, specified as a scalar in meters. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',120e-3`

Example: `insetpatch.GroundPlaneWidth = 120e-3`

Data Types: `double`

`Substrate` — Type of dielectric material

'Air' (default) | dielectric material object

Type of dielectric material used as a substrate, specified as a dielectric material object. For more information see, `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the ground plane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); insetpatch.Substrate = d`

`PatchCenterOffset` — Signed distance of patch from origin

`[0 0]` (default) | two-element real vector

Signed distance of patch from origin, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the patch relative to the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Example: `insetpatch.PatchCenterOffset = [0.01 0.01]`

Data Types: `double`

`FeedOffset` — Signed distance of feed from origin

[–0.0300 0] (default) | two-element real vector

Signed distance of feed from origin, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the feedpoint relative to the ground plane and patch.

Example: `'FeedOffset',[0.01 0.01]`

Example: `insetpatch.FeedOffset = [0.01 0.01]`

Data Types: `double`

`StripLineWidth` — Strip line width along *y*-axis

`1.0000e-03` (default) | scalar

Strip line width along *y*-axis, specified as a scalar in meters.

Example: `'StripLineWidth',0.1`

Example: `insetpatch.StripLineWidth = 0.1`

Data Types: `double`

`NotchLength` — Notch length along *x*-axis

`0.0080` (default) | scalar

Notch length along *x*-axis, specified as a scalar in meters.

Example: `'NotchLength',0.2`

Example: `insetpatch.NotchLength = 0.2`

Data Types: `double`

`NotchWidth` — Notch width along *y*-axis

`0.0030` (default) | scalar

Notch width along *y*-axis, specified as a scalar in meters.

Example: `'NotchWidth',0.1`

Example: `insetpatch.NotchWidth = 0.1`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `insetpatch.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90],TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

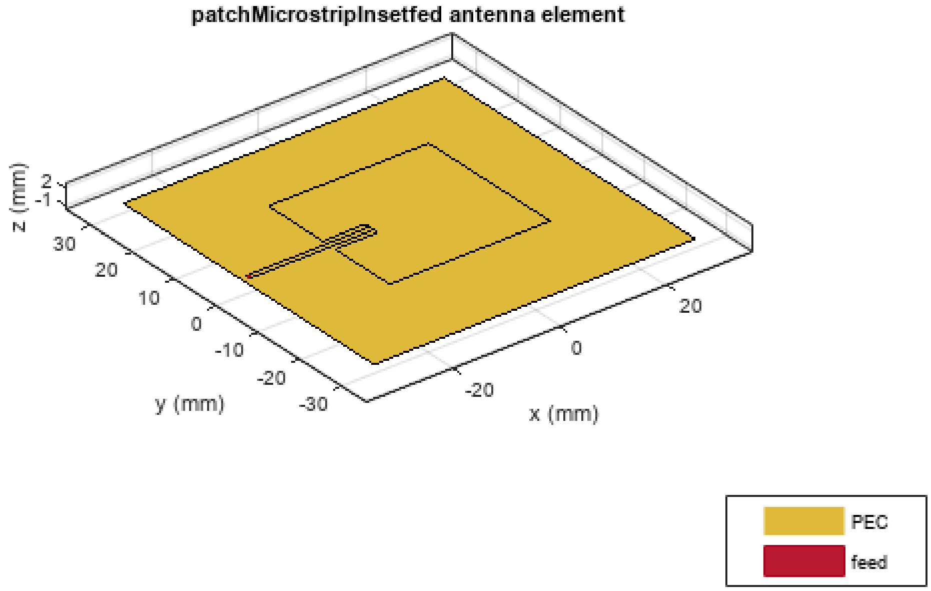
Inset-Fed Microstrip Patch

Create and view a default inset-fed microstrip patch.

```
insetpatch = patchMicrostripInsetfed
```

```
insetpatch = 
  patchMicrostripInsetfed with properties:

               Length: 0.0300
                Width: 0.0290
               Height: 0.0013
            Substrate: [1x1 dielectric]
    PatchCenterOffset: [0 0]
           FeedOffset: [-0.0300 0]
        StripLineWidth: 1.0000e-03
          NotchLength: 0.0080
           NotchWidth: 0.0030
    GroundPlaneLength: 0.0600
     GroundPlaneWidth: 0.0600
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(insetpatch)
```



Version History

Introduced in R2017b

See Also

`patchMicrostrip` | `patchMicrostripCircular`

Topics

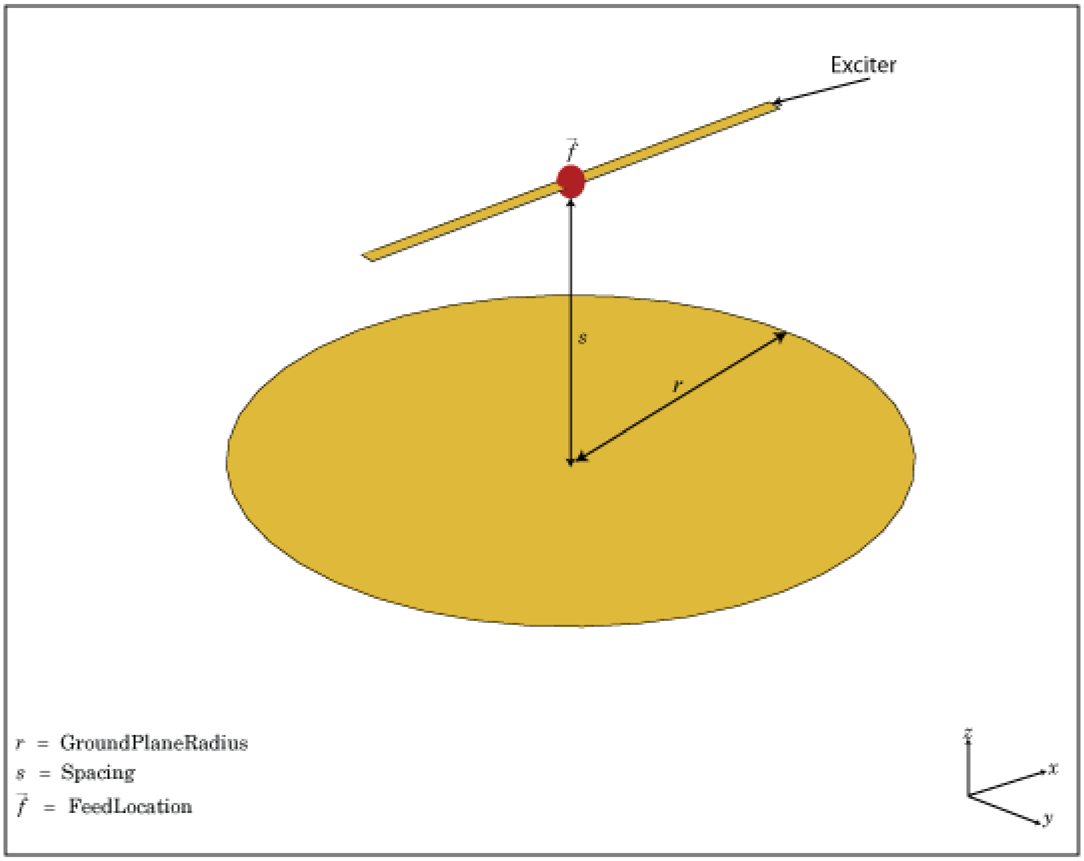
“Analysis of Inset-Feed Patch Antenna on Dielectric Substrate”

reflectorCircular

Create circular reflector-backed antenna

Description

Use the `reflectorCircular` object to create a circular reflector-backed antenna. By default the exciter is a dipole. The dimensions are chosen for an operating frequency of 1 GHz.



Creation

Syntax

```
rc = reflectorCircular
rc = reflectorCircular(Name,Value)
```

Description

`rc = reflectorCircular` creates a circular reflector backed antenna.

`rc = reflectorCircular(Name,Value)` sets properties using one or more name-value pair. For example, `rc = reflectorCircular('Radius',0.2)` creates a circular reflector of radius 0.2 m. Enclose each property name in quotes.

Properties

`Exciter` — Antenna or array type used as exciter

`dipole` (default) | `antenna` object | `array` object

Antenna type used as an exciter, specified as any single-element antenna object. Except reflector and cavity antenna elements, you can use any of the antenna elements or array elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',horn`

Example: `ant.Exciter = horn`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`GroundPlaneRadius` — Reflector radius

`0.1000` (default) | scalar

Radius of reflector, specified as a scalar in meters.

Example: `'Radius',0.2`

Example: `rc.Radius = 0.2`

Data Types: `double`

`Spacing` — Distance between exciter and reflector bottom

`0.0750` (default) | scalar

Distance between the exciter and the reflector, specified as a scalar in meters.

Example: `'Spacing',7.5e-2`

Example: `rc.Spacing = 7.5e-2`

Data Types: `double`

`Substrate` — Type of dielectric material

'Air' (default) | object

Type of dielectric material used as a substrate, specified as an object. For more information see, `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the groundplane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); rc.Substrate = d`

`EnableProbeFeed` — Create probe feed from backing structure to exciter

`0` (default) | `1` | scalar

Create probe feed from backing structure to exciter, specified as `0` or `1` or a scalar. By default, probe feed is not enabled.

Example: `'EnableProbeFeed',1`

Example: `rc.EnableProbeFeed = 1`

Data Types: `double` | `logical`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `rc.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90],TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.

- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Circular Reflector Backed Antenna

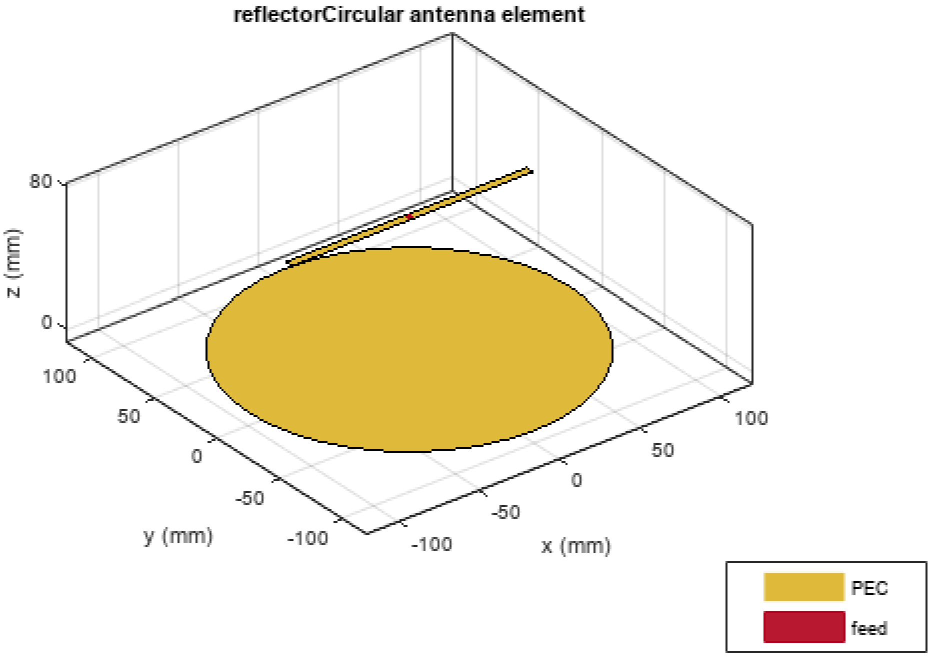
Create and view a default circular reflector backed antenna.

```
rc = reflectorCircular
```

```
rc = 
  reflectorCircular with properties:

              Exciter: [1x1 dipole]
            Substrate: [1x1 dielectric]
    GroundPlaneRadius: 0.1000
              Spacing: 0.0750
```

```
      EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

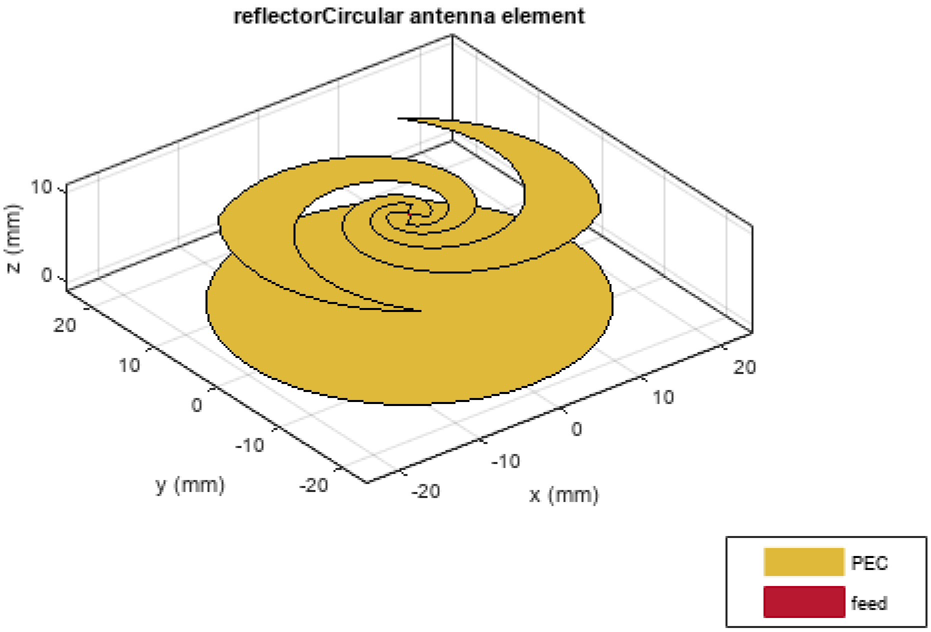
```
show(rc)
```



Radiation Pattern of Circular Reflector Backed Antenna

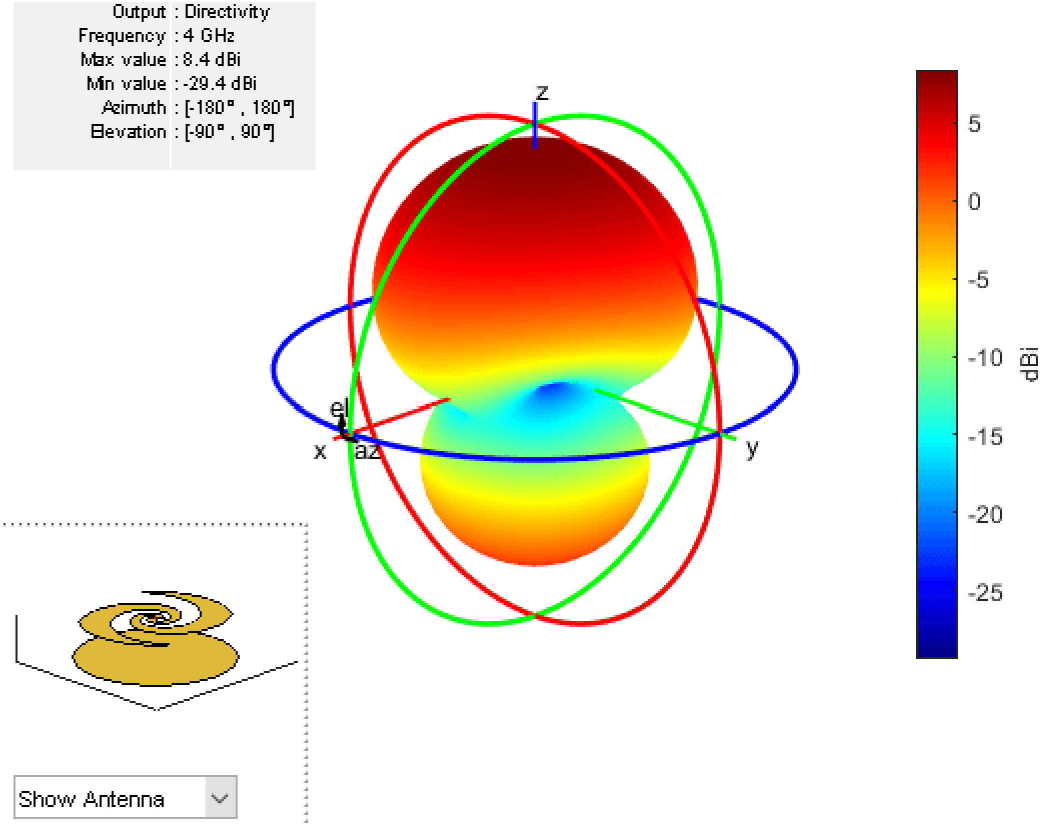
Create an equiangular spiral backed by a circular reflector.

```
ant = reflectorCircular('Exciter',spiralEquiangular,'GroundPlaneRadius',  ...
          0.02,'Spacing', 0.01);
show(ant)
```



Plot the radiation pattern of the antenna at 4 GHz.

```
pattern(ant,4e9)
```



```
   Output : Directivity
Frequency : 4 GHz
Max value : 8.4 dBi
Min value : -29.4 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Create Circular Reflector-Backed Linear Array of Dipole Antennas

Create a linear array of the dipole antennas.

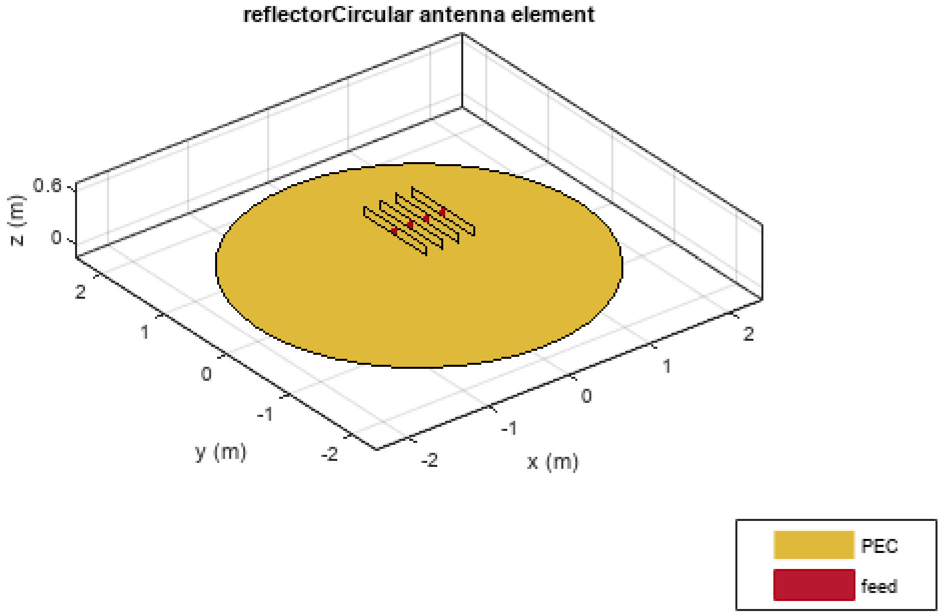
```
d = dipole('Length',1);
la = linearArray('Element',d,'NumElements',4,'ElementSpacing',0.2,'Tilt',90);
```

Create a linear array with circular reflector backing structure.

```
ant = reflectorCircular('Exciter',la,'GroundPlaneRadius',2,'Spacing',0.5)
```

```
ant = 
  reflectorCircular with properties:

              Exciter: [1x1 linearArray]
            Substrate: [1x1 dielectric]
    GroundPlaneRadius: 2
              Spacing: 0.5000
       EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



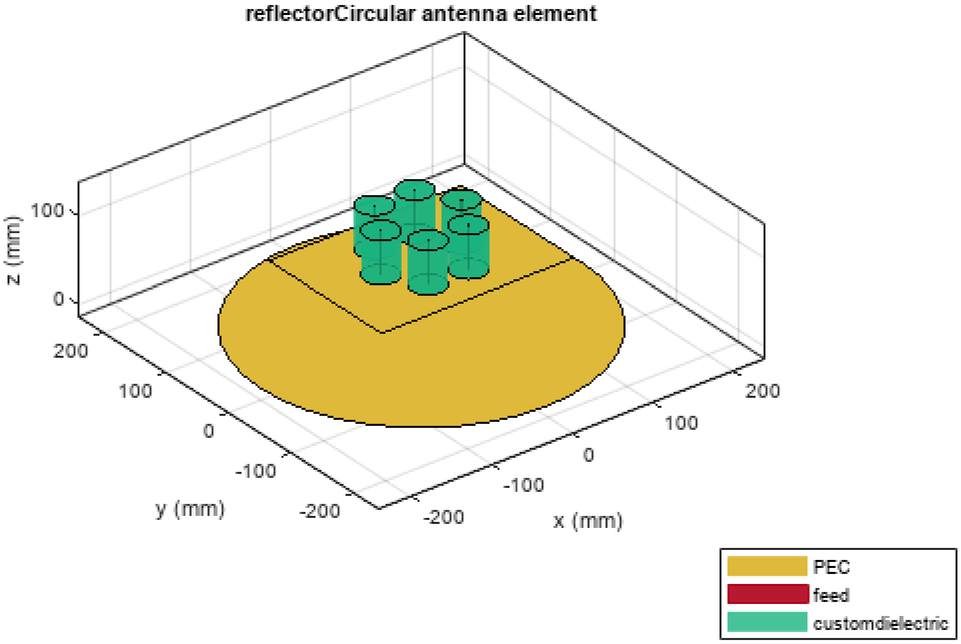
Create Circular Array of Cylindrical Dielectric Resonator Antennas (DRAs) with Circular Backing Structure

Create a circular reflector-backed circular array of cylindrical DRAs.

```
e = draCylindrical;
ca = circularArray('Element',e,'Radius',0.05);
ant = reflectorCircular('Exciter',ca,'GroundPlaneRadius',0.2)
```

```
ant = 
  reflectorCircular with properties:

              Exciter: [1x1 circularArray]
            Substrate: [1x1 dielectric]
    GroundPlaneRadius: 0.2000
              Spacing: 0.0750
      EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 0
            TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2017b

See Also

`cavity` | `reflector` | `cavityCircular`

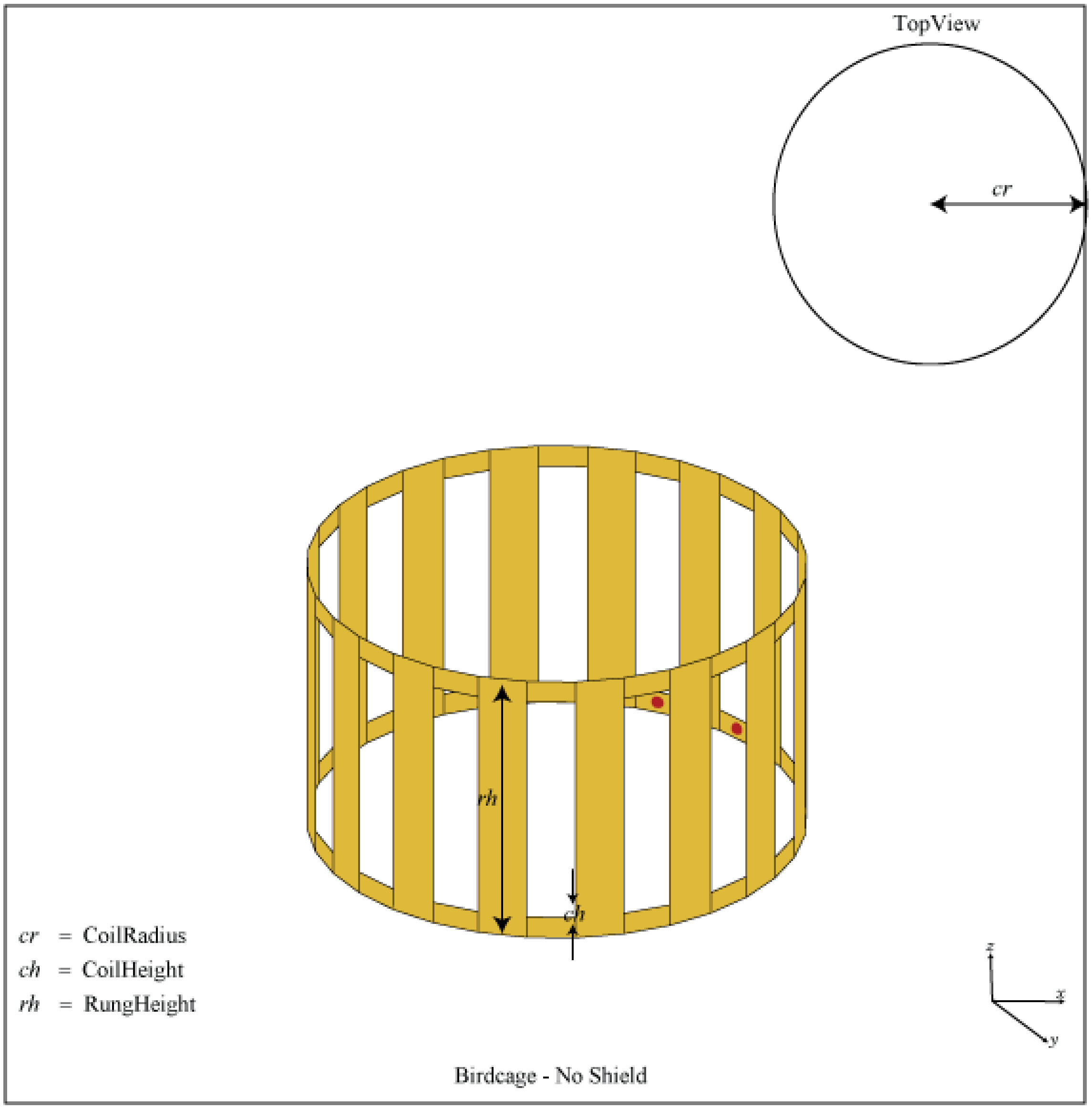
birdcage

Creates birdcage (MRI coil)

Description

The `birdcage` object creates to create a birdcage MRI coil. This antenna is most commonly used in clinical MRI. The antenna structure consists of two circular coils connected by conductive elements called `rungs`. The number of rungs depends on the size of the coil and is generally an even number.

The coil is operated at 64 MHz or 128 MHz. The birdcage can be loaded/excited to model a highpass or lowpass coil.



Creation

Syntax

```
bc = birdcage
bc = birdcage(Name,Value)
```

Description

`bc = birdcage` creates a birdcage antenna to model an MRI coil.

`bc = birdcage(Name,Value)` sets properties using one or more name-value pairs. For example, `bc = birdcage('NumRungs',8)` creates a birdcage with eight rungs. Enclose each property name in quotes.

Properties

`NumRungs` — Number of rungs
`16` (default) | scalar integer

Number of rungs, specified as a scalar.

Example: `'NumRungs',20`

Example: `bc.NumRungs = 20`

Data Types: `int8`

`CoilRadius` — Coil radius
`0.4000` (default) | scalar

Coil radius, specified as a scalar in meters.

Example: `'CoilRadius',0.2`

Example: `bc.CoilRadius = 0.2`

Data Types:

`CoilHeight` — Coil height
`0.0400` (default) | scalar

Coil height, specified as a scalar in meters.

Example: `'CoilHeight',0.089`

Example: `bc.CoilHeight = 0.089`

Data Types: `double`

`RungHeight` — Height of rungs
`0.4600` (default) | scalar

Height of rungs, specified as a scalar in meters. Distance is measured from the middle of the upper coil to the middle of the lower coil.

Example: `'RungHeight',0.56`

Example: `bc.RungHeight = 0.56`

Data Types: `double`

`ShieldRadius` — Shield radius

`0` (default) | scalar

Shield radius, specified as a scalar in meters. A value of zero indicates that the shield is absent.

Example: `'ShieldRadius',0.2`

Example: `bc.ShieldRadius = 0.2`

Data Types: `double`

`ShieldHeight` — Shield height

`0` (default) | scalar

Shield height, specified as a scalar in meters. A value of zero indicates that the shield is absent.

Example: `'ShieldHeight',0.089`

Example: `bc.ShieldHeight = 0.089`

Data Types: `double`

`Phantom` — Dielectric mesh to load birdcage

structure

Dielectric mesh to load birdcage, specified as a structure having the following fields:

`Points` — Points in custom dielectric mesh

N-by-3 matrix

Points in custom dielectric mesh, specified as an *N*-by-3 matrix in meters. *N* is the number of points.

You can use the phantom property to insert a dielectric mesh in the shape of a human head into the bird cage antenna. This dielectric cylinder has a permeability of 80. You can upload this mesh in the form of a mat file.

Data Types: `double`

`Tetrahedra` — Tetrahedra in custom dielectric mesh

M-by-4 integer matrix

Tetrahedra in custom dielectric mesh, specified as an *M*-by-4 integer matrix. *M* is the number of tetrahedra.

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric material

scalar

Relative permittivity of dielectric material, specified as a scalar.

Data Types: `double`

`LossTangent` — Loss in dielectric material

scalar

Loss in dielectric material, specified as a scalar.

Data Types: `double`

Data Types: `struct`

`FeedLocations` — Location of feeds in Cartesian coordinates
`0` (default) | *N*-by-3 matrix

Location of feeds in Cartesian coordinates, specified as an *N*-by-3 matrix. You can also use the `getLowPassLocs` and `getHighPassLocs` functions to determine the feed locations in low-pass or high-pass mode.

Example: `'FeedLocations'= [0.3981 0.0392 -0.2300;0.3528 0.1886 -0.2300]`

Example: `b.FeedLocations = getLowPassLocs(b)`

Data Types: `double`

`FeedVoltage` — Magnitude of voltage
`1` (default) | scalar | 1-by-*N* vector

Magnitude of voltage applied to each feed, specified as a scalar or 1-by-*N* vector with each element unit in volts.

Example: `'FeedVoltage',2`

Example: `bc.FeedVoltage = 2`

Data Types: `double`

`FeedPhase` — Phase shift to the voltage
`0` (default) | scalar | 1-by-*M* vector

Phase shift to the excitation voltage at each feed, specified as a scalar or 1-by-*M* vector with each element unit in degrees.

Example: `'FeedPhase',45`

Example: `bc.FeedPhase = 45`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, it is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `bc.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

getLowPassLocs	Feeding location to operate birdcage as lowpass coil
getHighPassLocs	Feeding location to operate birdcage as highpass coil
show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface

efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

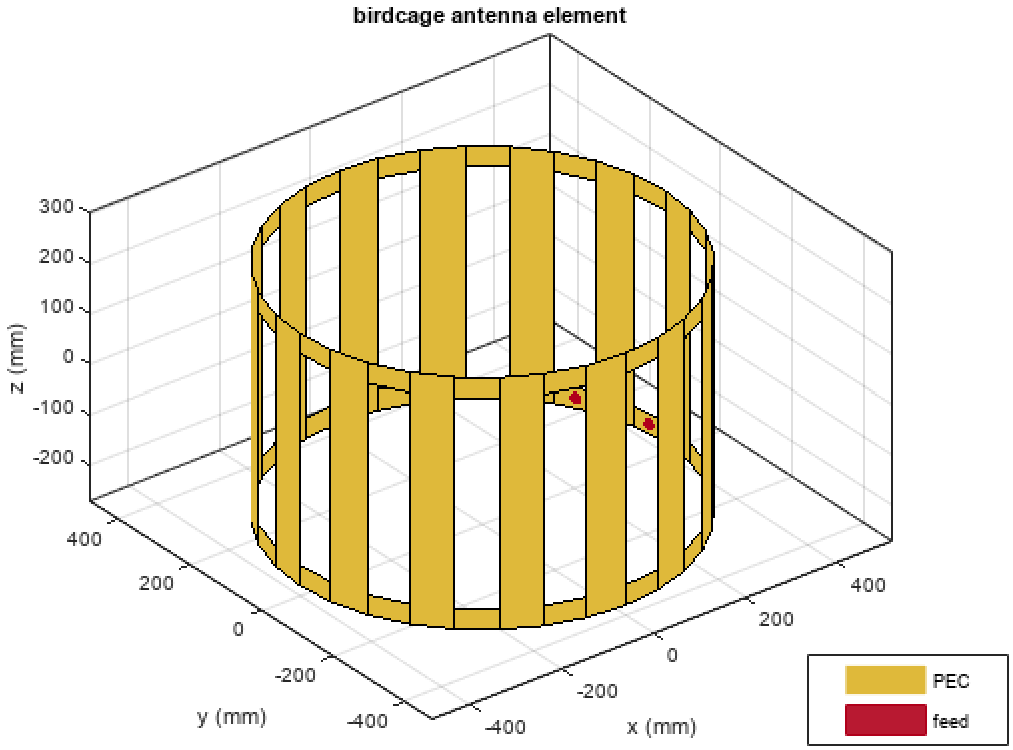
Birdcage Antenna

Create and view a default birdcage antenna.

```
bc = birdcage
```

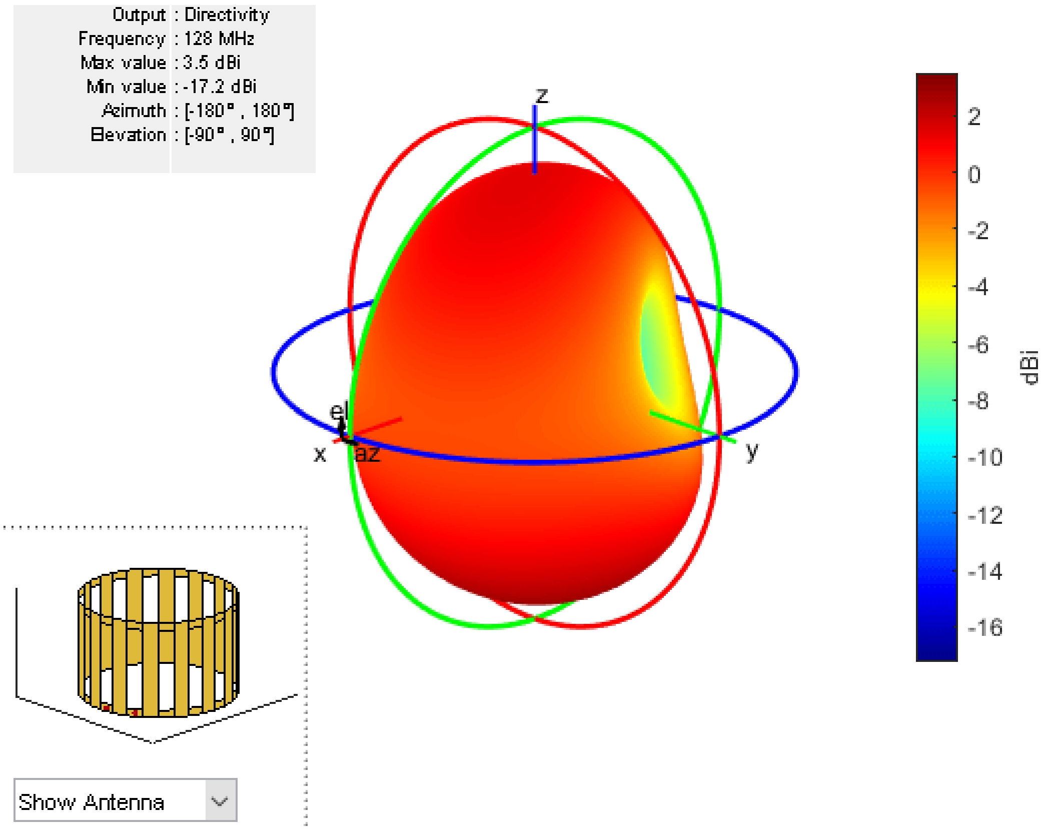
```
bc = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: []
    FeedLocations: [2x3 double]
      FeedVoltage: 1
        FeedPhase: 0
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

```
show(bc);
```



Plot the radiation pattern at 128 MHz.

```
pattern(bc,128e6)
```



Output : Directivity
Frequency : 128 MHz
Max value : 3.5 dBi
Min value : -17.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Human Head Model Inside BirdCage

Antenna Toolbox™ provides two `.mat` files to load a phantom human head model into a birdcage antenna. The humanheadcoarse.mat contains a coarse dielectric mesh of the human head model and the humanheadfine.mat provides the user with a finer dielectric mesh. Load the coarse human head model.

Load human head model file. Extract the values of `Points` and `Tetrahedra`. Add a relative permittivity (EpsilonR) of 10 and a dielectric loss (LossTangent) of 0.002. Scale the dielectric mesh to fit in the birdcage antenna. In this case, the mesh points are multiplied by 0.003.

```
load humanheadcoarse.mat
humanhead = struct('Points',0.003*P,'Tetrahedra',T,'EpsilonR',10,...
                  'LossTangent',0.002)
```

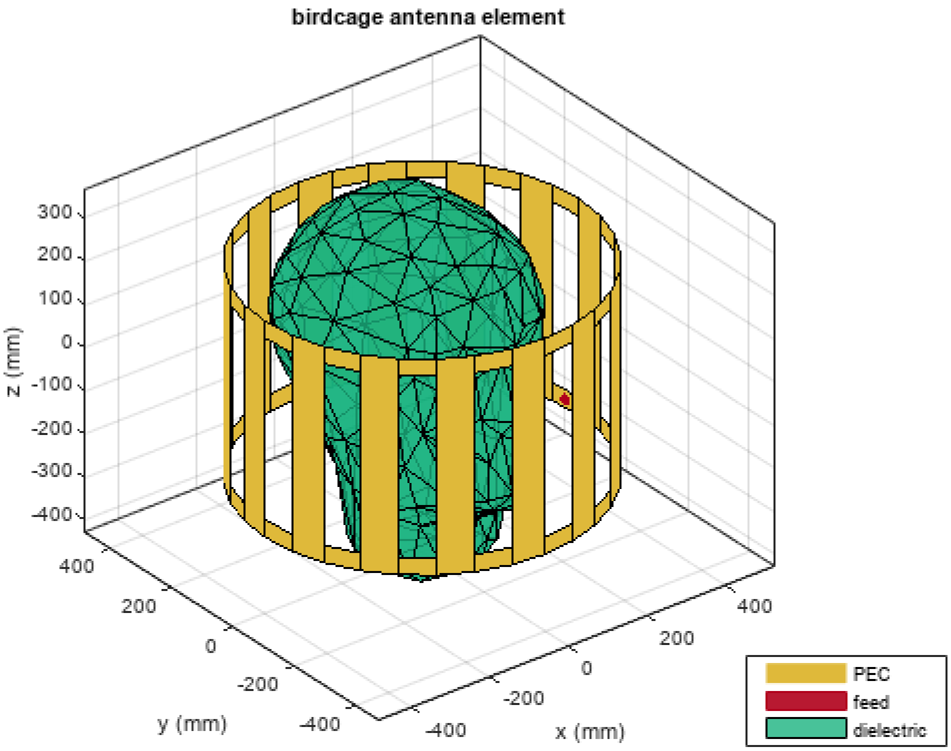
```
humanhead = struct with fields:
         Points: [584x3 double]
     Tetrahedra: [2818x4 double]
       EpsilonR: 10
    LossTangent: 0.0020
```

Add and view the human head mesh inside the birdcage.

```
b = birdcage('Phantom',humanhead)
```

```
b = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: [1x1 struct]
    FeedLocations: [2x3 double]
      FeedVoltage: 1
        FeedPhase: 0
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

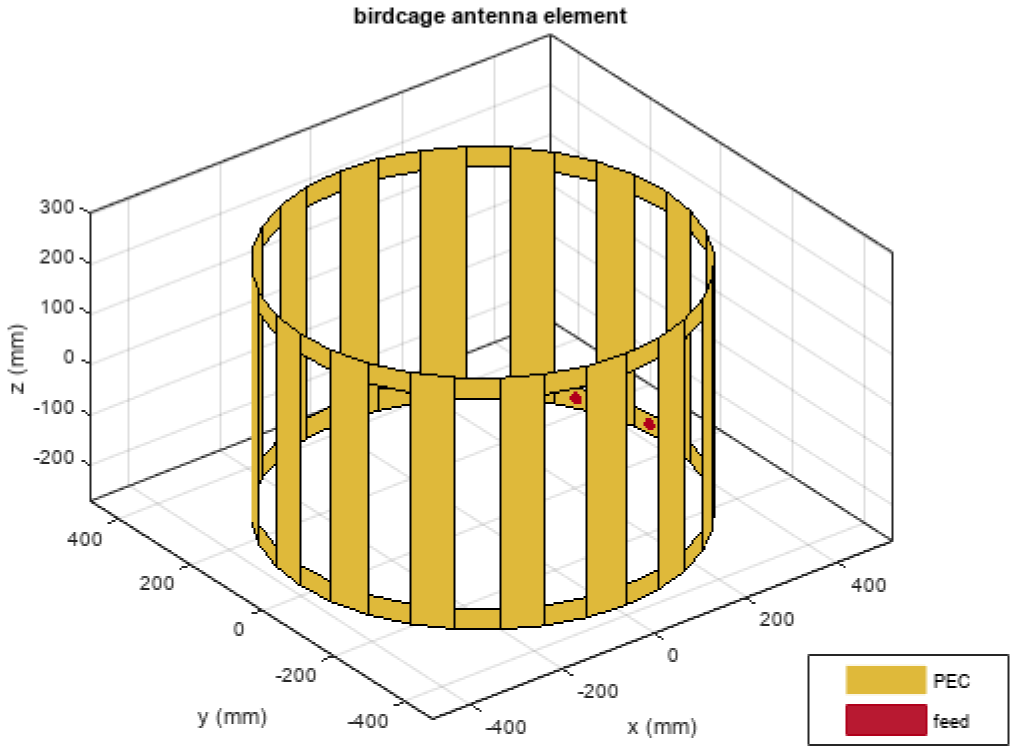
```
show(b)
```



Birdcage In High-Pass Operation

Create a birdcage antenna.

```
b = birdcage;
show(b);
```

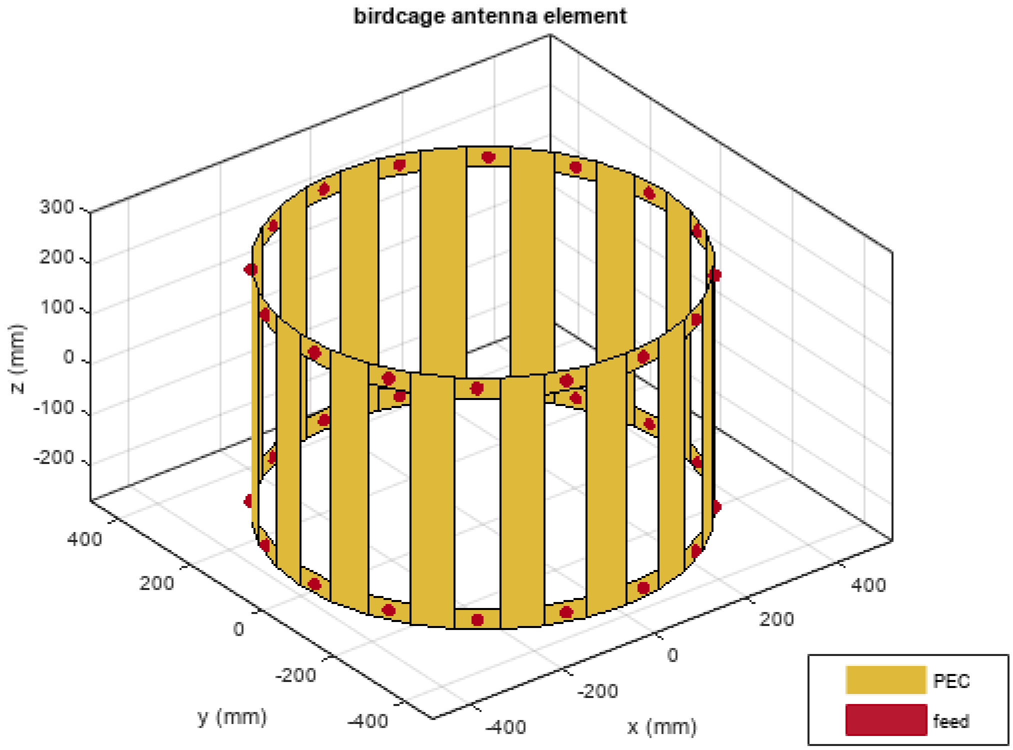


Use the birdcage as a high-pass coil.

```
b.FeedLocations = getHighPassLocs(b)
```

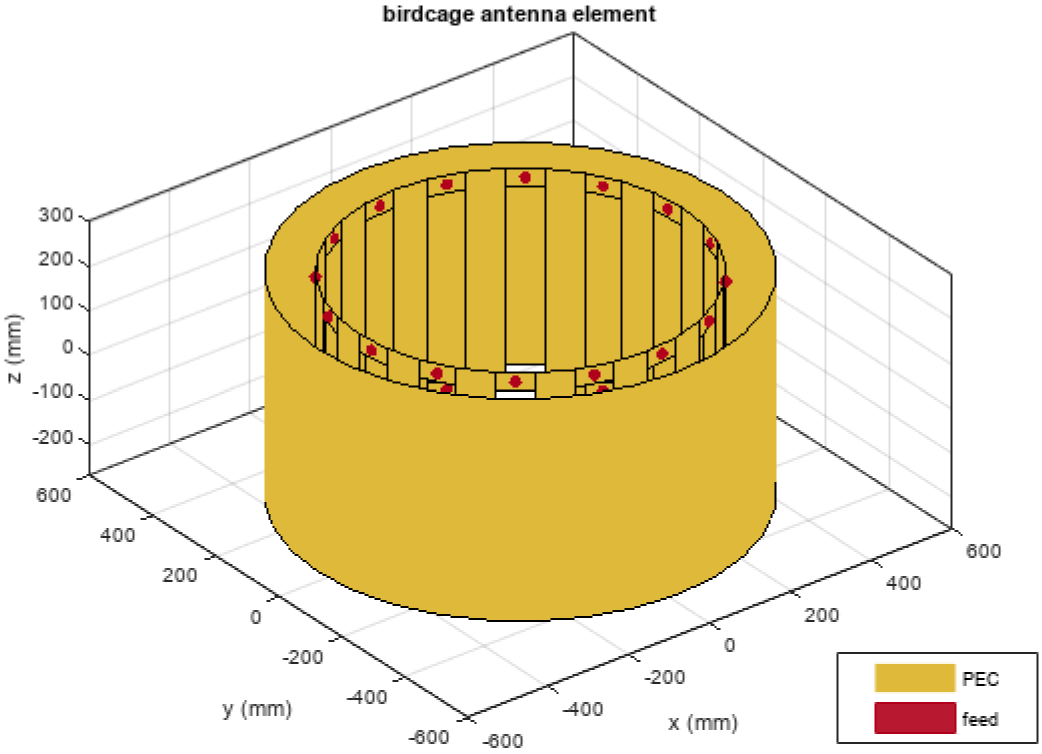
```
b = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: []
    FeedLocations: [32x3 double]
      FeedVoltage: 1
        FeedPhase: 0
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

```
show(b);
```



Shield the antenna to ensure that radiation does not leak out.

```
b.ShieldRadius = 0.5;
b.ShieldHeight = 0.5;
show(b) ;
```



Version History

Introduced in R2017b

See Also

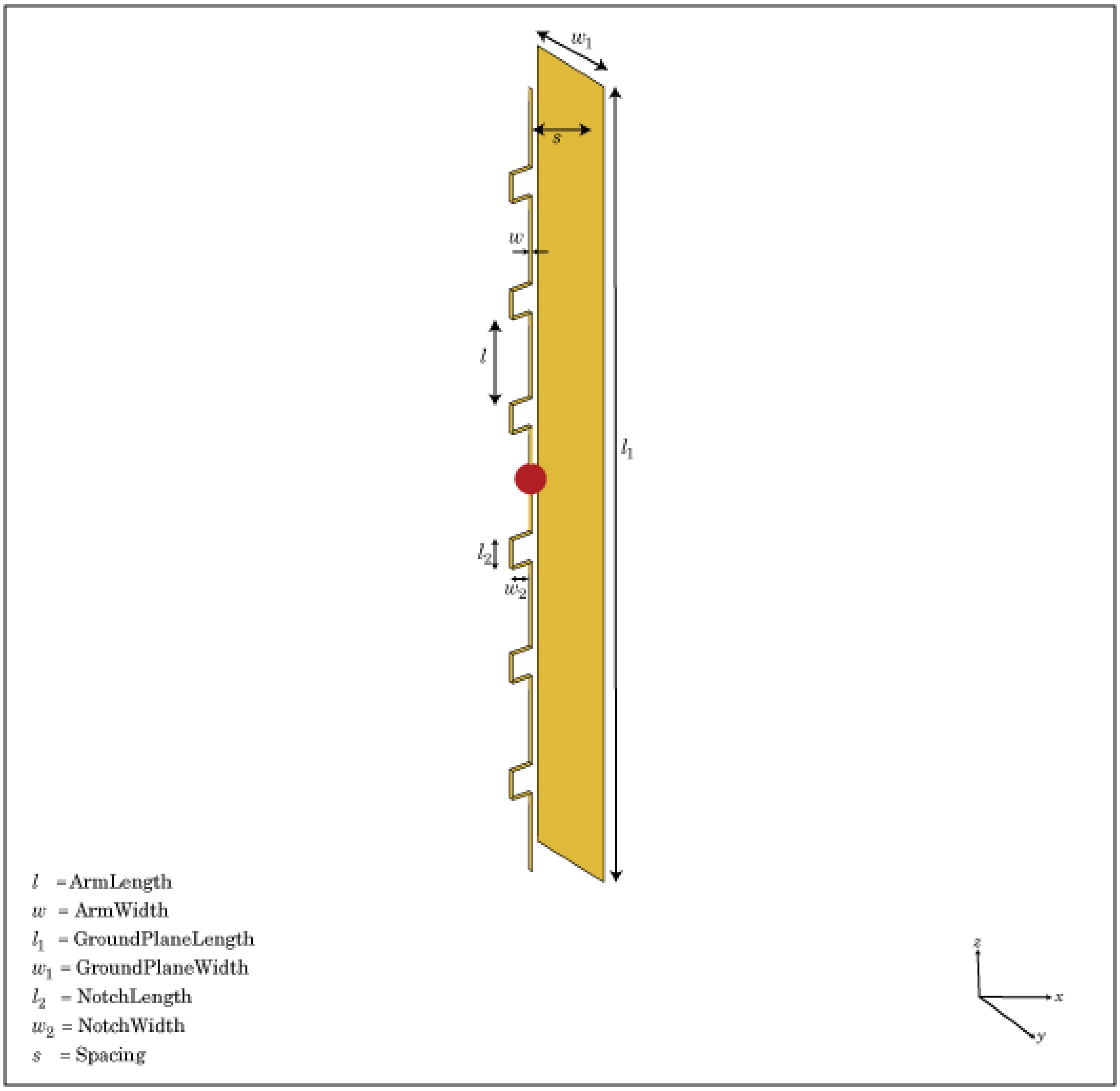
`dipole` | `loopCircular`

sectorInvertedAmos

Create inverted Amos sector antenna

Description

Use the `sectorInvertedAmos` object to create an inverted Amos sector antenna consisting of four dipole-like arms. The antenna is fed at the origin of the dipole. The dipole arms are symmetric about the origin. The operating frequency of the antenna is at 2.45 GHz wireless.



Creation

Syntax

```
amossector = sectorInvertedAmos
amossector = sectorInvertedAmos(Name,Value)
```

Description

`amossector = sectorInvertedAmos` creates an inverted Amos sector antenna with four dipole-like arms.

`amossector = sectorInvertedAmos(Name,Value)` sets properties using one or more name-value pair. For example, `amossector = sectorInvertedAmos('ArmWidth',0.2)` creates an inverted Amos sector with a dipole width of 0.2 m. Enclose each property name in quotes.

Properties

`ArmLength` — Individual dipole arm length
`[0.0880 0.0710 0.0730 0.0650]` (default) | vector

Length of individual dipole arms, specified as a vector with each element unit in meters.

Example: `'ArmLength',[0.0980 0.0810 0.0830 0.0750]`

Example: `amossector.ArmLength = [0.0980 0.0810 0.0830 0.0750]`

Data Types: `double`

`ArmWidth` — Dipole arm width
`0.0040` (default) | scalar

Width of dipole arms, specified as a scalar in meters.

Example: `'ArmWidth',0.0025`

Example: `amossector.ArmWidth = 0.0025`

Data Types: `double`

`NotchLength` — Notch length
`0.0238` (default) | scalar

Notch length, specified as a scalar in meters. For an inverted Amos sector antenna with seven stacked arms, six notches are generated. Notch length is measured along the length of the antennas.

Example: `'NotchLength',0.001`

Example: `amossector.NotchLength = 0.001`

Data Types: `double`

`NotchWidth` — Notch width
`0.0170` (default) | scalar

Notch width, specified as a scalar in meters. For an inverted Amos sector antenna with seven stacked arms, six notches are generated. Notch width is measured perpendicular to the length of the antenna.

Example: `'NotchWidth',0.00190`

Example: `amossector.NotchWidth = 0.00190`

Data Types: `double`

`GroundPlaneLength` — Ground plane length
0.6600 (default) | scalar

Ground plane length, specified as a scalar in meters. By default, ground plane length is measured along *x*-axis.

Example: `'GroundPlaneLength',0.7500`

Example: `amossector.GroundPlaneLength = 0.7500`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width
0.0750 (default) | scalar

Ground plane width, specified as a scalar in meters. By default, ground plane width is measured along *y*-axis.

Example: `'GroundPlaneWidth',0.0500`

Example: `amossector.GroundPlaneWidth = 0.0500`

Data Types: `double`

`Spacing` — Distance between ground plane and antenna element
0.0355 (default) | scalar

Distance between ground plane and antenna element, specified as a scalar in meters.

Example: `'Spacing',0.0355`

Example: `amossector.Spacing = 0.0355`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, it is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `amossector.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays

impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

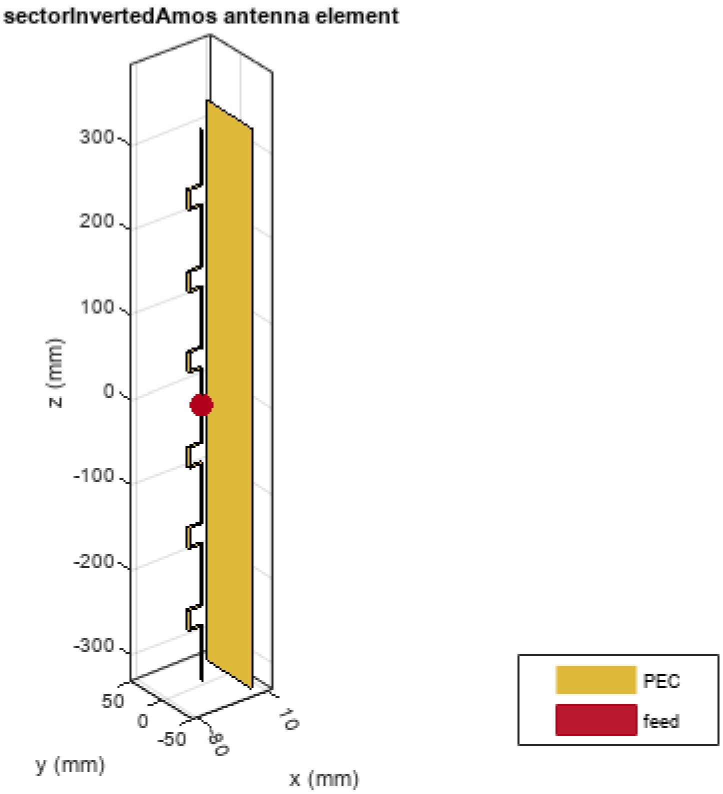
Inverted Amos Sector

Create and view an inverted Amos sector antenna.

```
sectoria = sectorInvertedAmos
```

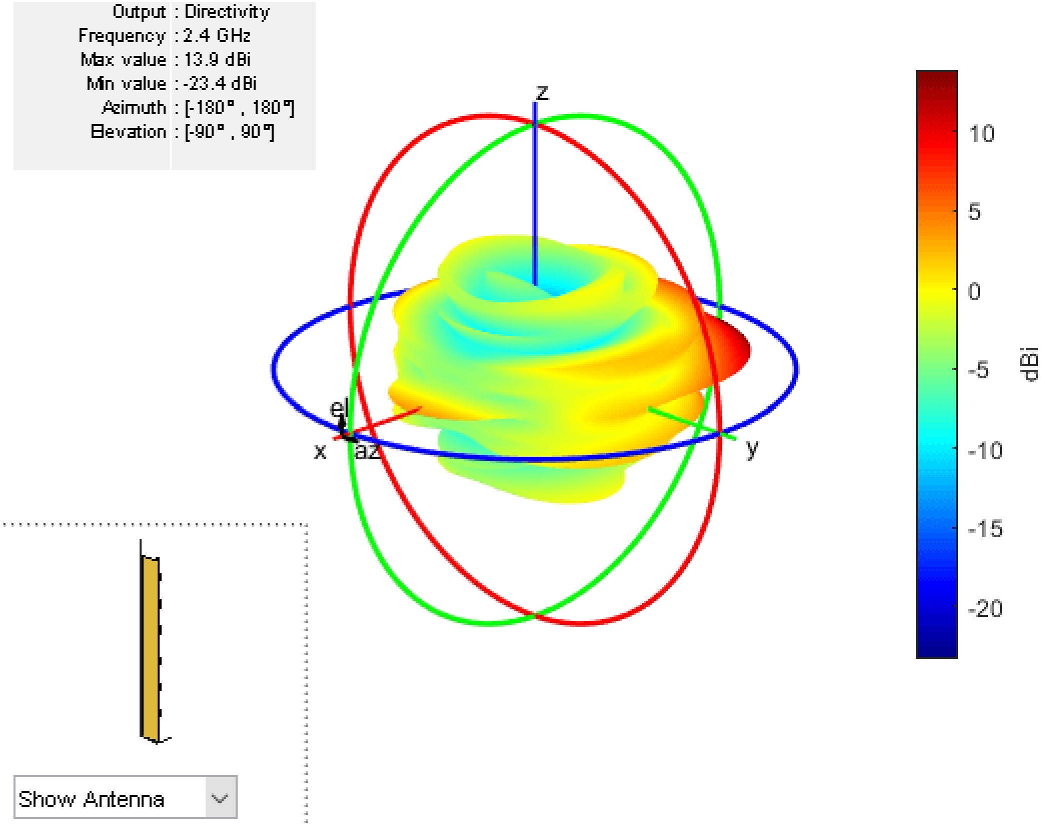
```
sectoria = 
  sectorInvertedAmos with properties:

            ArmLength: [0.0880 0.0710 0.0730 0.0650]
             ArmWidth: 0.0040
          NotchLength: 0.0238
           NotchWidth: 0.0170
    GroundPlaneLength: 0.6600
     GroundPlaneWidth: 0.0750
              Spacing: 0.0355
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(sectoria)
```



Plot Radiation Pattern at 2.4 GHz

```
pattern(sectoria,2.4e9)
```



Output : Directivity
Frequency : 2.4 GHz
Max value : 13.9 dBi
Min value : -23.4 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2017b

See Also

`dipoleMeander` | `reflector`

antenna.Circle

Create circle centered at origin on X-Y plane

Description

Use the `antenna.Circle` object to create a circle centered at the origin and on the X-Y plane. You can use `antenna.Polygon` to create single-layer or multi-layered antennas using `pcbStack`.

Creation

Syntax

```
circle = antenna.Circle
circle = antenna.Circle(Name,Value)
```

Description

`circle = antenna.Circle` creates a circle centered at the origin and on the X-Y plane.

`circle = antenna.Circle(Name,Value)` sets properties using one or more name-value pair. For example, `circle = antenna.Circle('Radius',0.2)` creates a circle of radius 0.2 m. Enclose each property name in quotes.

Properties

`Name` — Name of circle
`'mycircle'` (default) | character vector

Name of circle, specified a character vector.

Example: `'Name','Circle1'`

Example: `circle.Name= 'Circle1'`

Data Types: `char` | `string`

`Center` — Cartesian coordinates of center of circle
`[0 0]` (default) | 2-element vector

Cartesian coordinates of center of circle, specified a 2-element vector with each element measured in meters.

Example: `'Center',[0.006 0.006]`

Example: `circle.Center= [0.006 0.006]`

Data Types: `double`

`Radius` — Circle radius
1 (default) | scalar

Circle radius, specified a scalar in meters.

Example: `'Radius',2`

Example: `circle.Radius= 2`

Data Types: `double`

`NumPoints` — Number of discretization points on circumference

30 (default) | scalar

Number of discretization points on circumference, specified a scalar.

Example: `'NumPoints',16`

Example: `circle.NumPoints= 2`

Data Types: `double`

Object Functions

add	Boolean unite operation on two shapes
subtract	Boolean subtraction operation on two shapes
area	Calculate area of shape in square meters
intersect	Boolean intersection operation on two shapes
rotate	Rotate shape about axis and angle
rotateX	Rotate shape about x-axis and angle
rotateY	Rotate shape about y-axis and angle
rotateZ	Rotate shape about z-axis and angle
translate	Move shape to new location
show	Display antenna, array structures or shapes
mesh	Mesh properties of metal, dielectric antenna, or array structure
removeHoles	Remove holes from shape
removeSlivers	Remove sliver outliers from boundary of shape

Examples

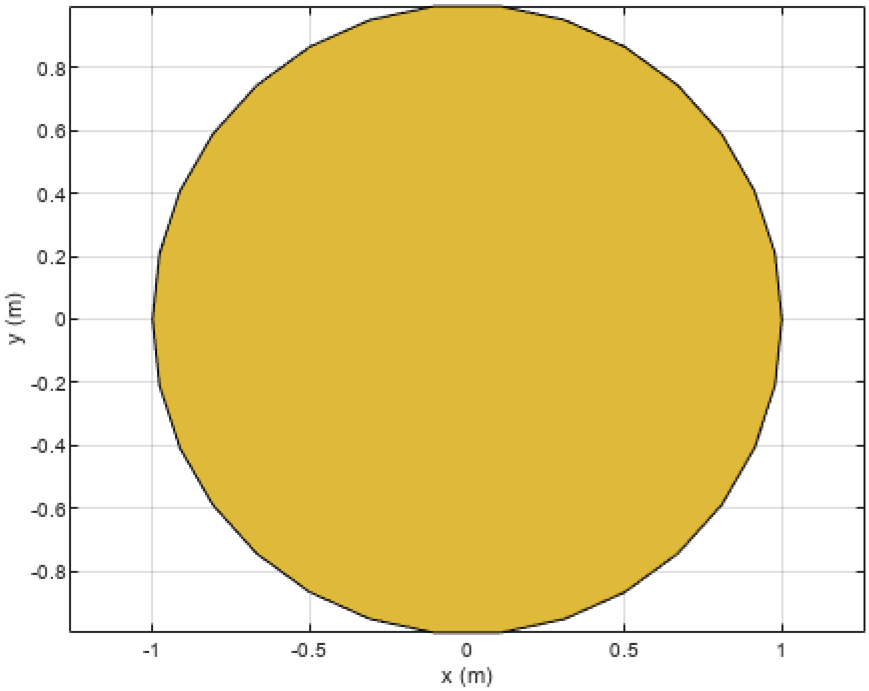
Create Circle with Default Properties

Create and view circle using `antenna.Circle` and view it.

```
c1 = antenna.Circle
```

```
c1 = 
  Circle with properties:

         Name: 'mycircle'
       Center: [0 0]
       Radius: 1
    NumPoints: 30
```

```
show(c1)
```



Create Circle with Specified Properties

Create a circle with a radius of 4 m.

```
c2 = antenna.Circle('Radius',4)
```

```
c2 = 
  Circle with properties:

         Name: 'mycircle'
       Center: [0 0]
       Radius: 4
    NumPoints: 30
```

Add Two Shapes

Create a circle with a radius of 1 m. The center of the circle is at [1 0].

```
circle1 = antenna.Circle('Center',[1 0],'Radius',1);
```

Create a rectangle with a length of 2 m and a width of 4 m centered at the origin.

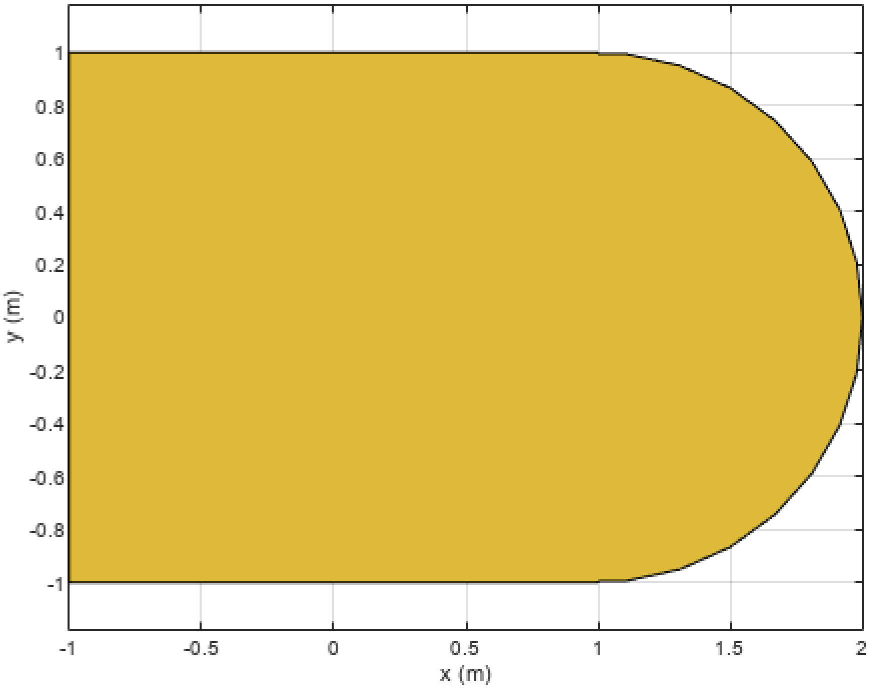
```
rect1 = antenna.Rectangle('Length',2,'Width',2);
```

Add the two shapes together using the + function.

```
polygon1 = circle1+rect1
```

```
polygon1 = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [21x3 double]
```

```
show(polygon1)
```



Version History

Introduced in R2017a

See Also

`antenna.Polygon` | `antenna.Rectangle`

antenna.Polygon

Create polygon on X-Y plane

Description

Use the `antenna.Polygon` object to create a polygonal board shape centered at the origin and on the X-Y plane. You can use `antenna.Polygon` to create single-layer or multilayered antennas using `pcbStack`.

Creation

Syntax

```
polygon = antenna.Polygon
polygon = antenna.Polygon(Name,Value)
```

Description

`polygon = antenna.Polygon` creates a polygonal board shape centered at the origin and on the X-Y plane.

`polygon = antenna.Polygon(Name,Value)` sets properties using one or more name-value pair. For example, `polygon = antenna.Polygon('Name','mypolygonboard')` creates a polygon board shape of the name `'mypolygonboard'`. Enclose each property name in quotes.

Properties

`Name` — Name of polygon board shape
`'mypolygon'` (default) | character vector | string

Name of the polygon board shape, specified a character vector or string.

Example: `'Name','Polygon1'`

Example: `polygon.Name = 'Polygon1'`

Data Types: `char` | `string`

`Vertices` — Cartesian coordinates of polygon vertices
3-by-3 matrix (default) | *N*-by-3 matrix

Cartesian coordinates of polygon vertices, specified as a *N*-by-3 matrix with each element measured in meters, *N* being the number of points.

Example: `'Vertices',[-1 0 0;-0.5 0.2 0;0 0 0]`

Example: `polygon.Vertices = [-1 0 0;-0.5 0.2 0;0 0 0]`

Data Types: `double`

Object Functions

add	Boolean unite operation on two shapes
area	Calculate area of shape in square meters
subtract	Boolean subtraction operation on two shapes
intersect	Boolean intersection operation on two shapes
rotate	Rotate shape about axis and angle
rotateX	Rotate shape about x-axis and angle
rotateY	Rotate shape about y-axis and angle
rotateZ	Rotate shape about z-axis and angle
translate	Move shape to new location
show	Display antenna, array structures or shapes
mesh	Mesh properties of metal, dielectric antenna, or array structure
removeHoles	Remove holes from shape
removeSlivers	Remove sliver outliers from boundary of shape

Examples

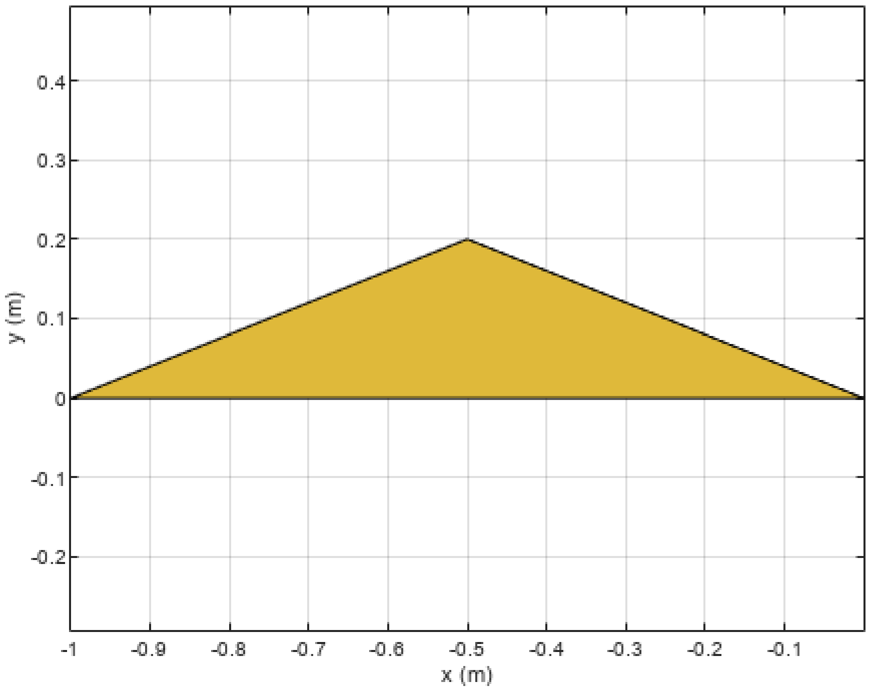
Create and Transform Polygon

Create a polygon using `antenna.Polygon` with vertices at `[-1 0 0;-0.5 0.2 0;0 0 0]` and view it.

```
p = antenna.Polygon('Vertices', [-1 0 0;-0.5 0.2 0;0 0 0])
```

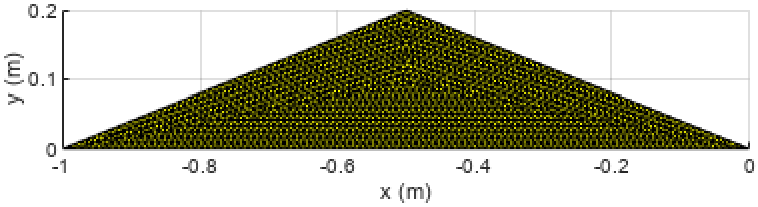
```
p = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [3x3 double]
```

```
show(p)
axis equal
```



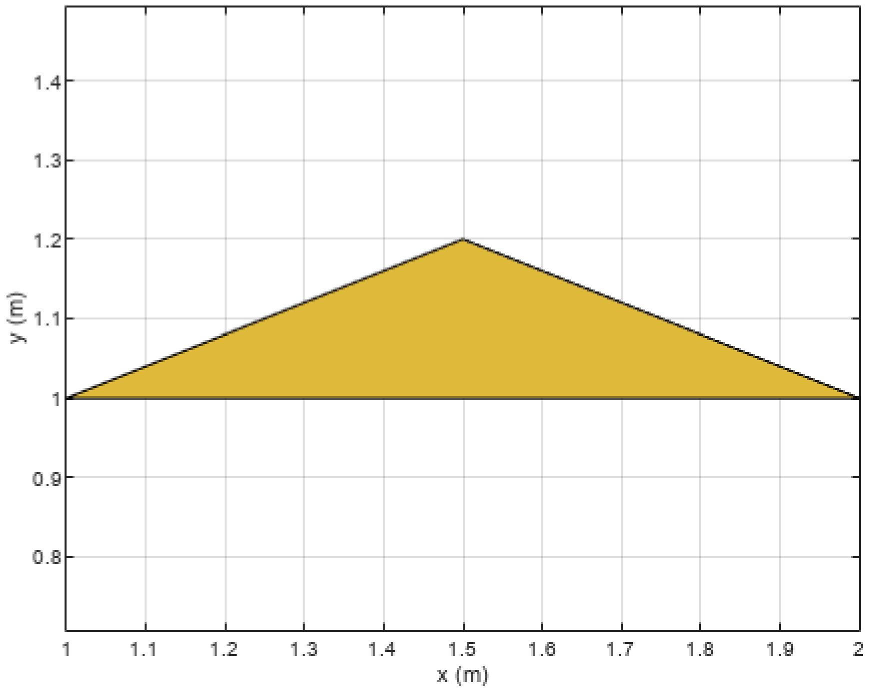
Mesh the polygon and view it.

```
mesh(p,0.2)
```



Move the polygon to a new location on the X-Y plane.

```
translate(p,[2,1,0])
axis equal
```



Version History

Introduced in R2017a

See Also

`antenna.Circle` | `antenna.Rectangle`

antenna.Rectangle

Create rectangle centered at origin on X-Y plane

Description

Use the `antenna.Rectangle` object to create a rectangle centered at the origin and on the X-Y plane. You can use `antenna.Polygon` to create single-layer or multi-layered antennas using `pcbStack`.

Creation

Syntax

```
rect = antenna.Rectangle
rect = antenna.Rectangle(Name,Value)
```

Description

`rect = antenna.Rectangle` creates a rectangle centered at the origin and on the X-Y plane.

`rect = antenna.Rectangle(Name,Value)` sets properties using one or more name-value pair. For example, `rectangle = antenna.Rectangle('Length',0.2)` creates a rectangle of length 0.2 m. Enclose each property name in quotes.

Properties

`Name` — Name of rectangle
`'myrectangle'` (default) | character vector

Name of rectangle, specified a character vector.

Example: `'Name','Rect1'`

Example: `rectangle.Name = 'Rect1'`

Data Types: `char` | `string`

`Center` — Cartesian coordinates of center of rectangle
`[0 0]` (default) | 2-element vector

Cartesian coordinates of center of rectangle, specified a 2-element vector with each element measured in meters.

Example: `'Center',[0.006 0.006]`

Example: `rectangle.Center = [0.006 0.006]`

Data Types: `double`

`Length` — Rectangle length
1 (default) | scalar

Rectangle length, specified a scalar in meters.

Example: `'Length',2`

Example: `rectangle.Length = 2`

Data Types: `double`

`Width` — Rectangle width

2 (default) | scalar

Rectangle width, specified a scalar in meters.

Example: `'Width',4`

Example: `rectangle.Width = 4`

Data Types: `double`

`NumPoints` — Number of discretization points per side

2 (default) | scalar

Number of discretization points per side, specified a scalar.

Example: `'NumPoints',16`

Example: `rectangle.NumPoints = 16`

Data Types: `double`

Object Functions

add	Boolean unite operation on two shapes
area	Calculate area of shape in square meters
subtract	Boolean subtraction operation on two shapes
intersect	Boolean intersection operation on two shapes
rotate	Rotate shape about axis and angle
rotateX	Rotate shape about x-axis and angle
rotateY	Rotate shape about y-axis and angle
rotateZ	Rotate shape about z-axis and angle
translate	Move shape to new location
show	Display antenna, array structures or shapes
mesh	Mesh properties of metal, dielectric antenna, or array structure
removeHoles	Remove holes from shape
removeSlivers	Remove sliver outliers from boundary of shape

Examples

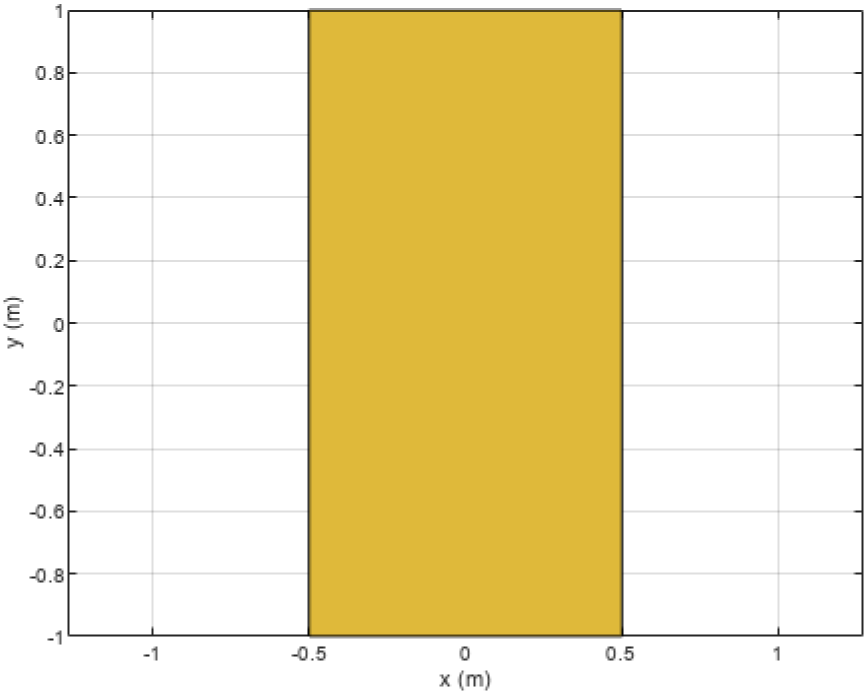
Create Rectangle with Default Properties

Create a rectangle shape using antenna.Rectangle and view it.

```
r1 = antenna.Rectangle
```

```
r1 = 
  Rectangle with properties:
         Name: 'myrectangle'
       Center: [0 0]
       Length: 1
        Width: 2
    NumPoints: 2
```

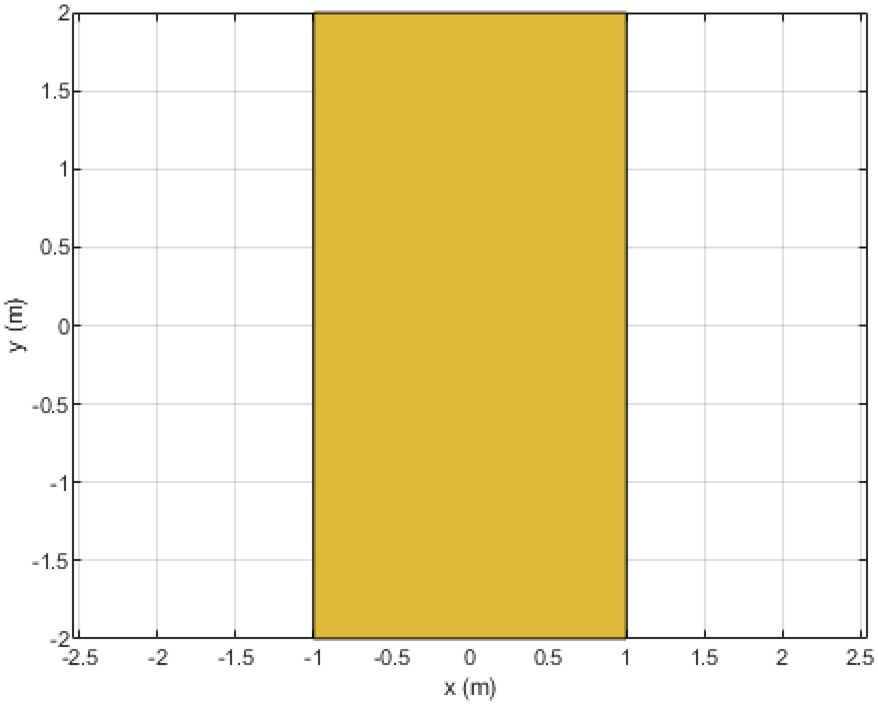
```
show(r1)
```



Create and Rotate Rectangle Using Specified Properties

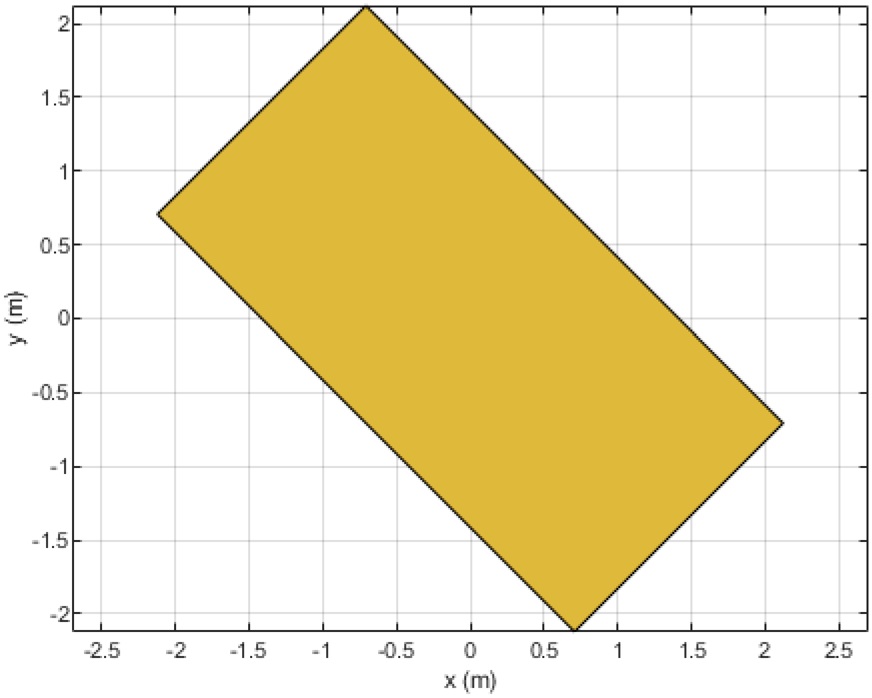
Create and view a rectangle with a length of 2 m and a width of 4 m.

```
r2 = antenna.Rectangle('Length',2,'Width',4);
show(r2)
axis equal
```



Rotate the rectangle.

```
rotateZ(r2,45);
show(r2)
```



Create Notched Rectangle

Create a rectangle with a length of 0.15 m, and a width of 0.15 m.

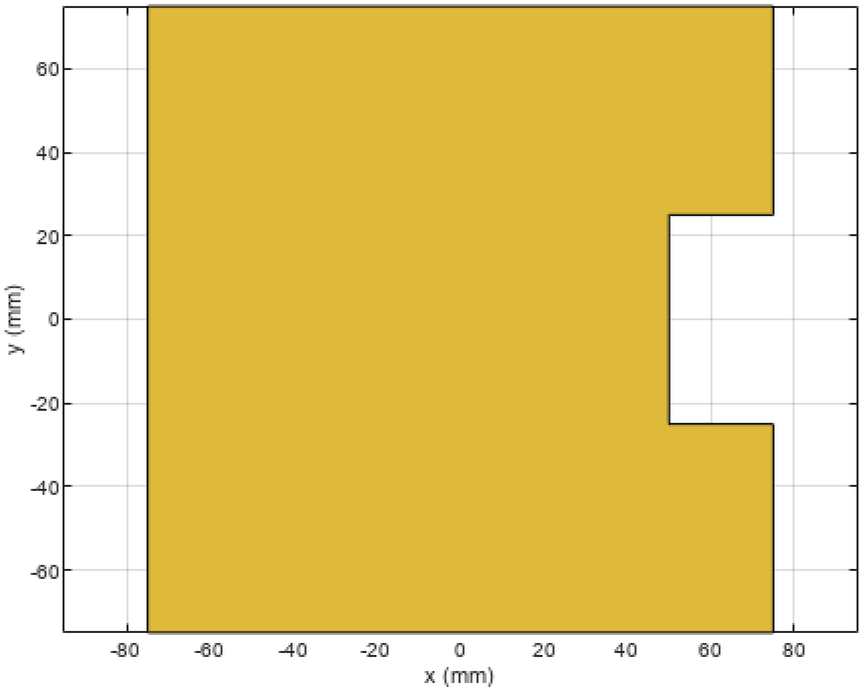
```
r  = antenna.Rectangle('Length',0.15,'Width',0.15);
```

Create a second rectangle with a length of 0.05 m, and a width of 0.05 m. Set the center of the second rectangle at half the length of the first rectangle r.

```
n = antenna.Rectangle('Center',[0.075,0],'Length',0.05,'Width',0.05);
```

Create and view a notched rectangle by subtracting n from r.

```
rn  = r-n;
show(rn)
```



Calculate the area of the notched rectangle.

```
area(rn)
```

```
ans = 0.0212
```

Version History

Introduced in R2017a

See Also

antenna.Circle | antenna.Polygon

PCBWriter

Create PCB board definitions from 2-D antenna designs

Description

Use the `PCBWriter` object to create a printed circuit board (PCB) design files based on multilayer 2-D antenna design. A set of manufacturing files known as Gerber files describes a PCB antennas. A Gerber file uses an ASCII vector format for 2-D binary images.

Creation

Syntax

```
b = PCBWriter(pcbstackobject)
b = PCBWriter(pcbstackobject,rfconnector)
b = PCBWriter(pcbstackobject,writer)
b = PCBWriter(pcbstackobject,rfconnector,writer)
```

Description

`b = PCBWriter(pcbstackobject)` creates a `PCBWriter` object that generates Gerber-format PCB design files based on a 2-D antenna design geometry using PCB stack.

`b = PCBWriter(pcbstackobject,rfconnector)` creates a customized PCB file using specified `rfconnector` type.

`b = PCBWriter(pcbstackobject,writer)` creates a customized PCB file using a specified PCB service, `writer`.

`b = PCBWriter(pcbstackobject,rfconnector,writer)` creates customised PCB file using specified PCB service and PCB connector type.

Input Arguments

`pcbstackobject` — Single feed PCB antenna
`pcbStack` object

Single feed PCB antenna, specified as a `pcbStack` object. For more information, see `pcbStack`.

Example: `p1 = pcbStack` creates a PCB stack object,`p1` `a = PCBWriter(p1)`, uses `p1` to create a `PCBWriter` object `a`.

`writer` — PCB service to view PCB design
object

PCB service to view PCB design, specified as `PCBServices` object.

Example: `s =PCBServices.MayhewWriter; a = PCBWriter(p1,s)` uses Mayhew Labs PCB service to view the PCB design. For more information on manufacturing services, see `PCBServices`

`rfconnector` — RF connector type

object

RF connector type for PCB antenna feedpoint, specified as `PCBConnectors` object. For information about connectors , see `PCBConnectors`.

Example: `c = PCBConnectors.SMA_Cinch;a = PCBWriter(p1,c)` uses SMA_Cinch RF connector at feedpoint.

Output Arguments

b — PCB Board definition of 2.5D antenna design

object

PCB Board definition of 2.5D antenna design, returned as an object.

Properties

`UseDefaultConnector` — Use default connector

`1` (default) | `0`

Use default connector, specified as `0` or `1`.

Example: `a.UseDefaultConnector = 1`, where `a` is a `PCBWriter` object.

Data Types: `logical`

`ComponentBoundaryLineWidth` — Line widths drawn around components on silk screens

`8` (default) | positive scalar

Line widths drawn around components on silk screens, specified as a positive scalar in mils.

Example: `a.ComponentBoundaryLineWidth = 10`, where `a` is a `PCBWriter` object.

Data Types: `double`

`ComponentNameFontSize` — Font size to label components on silk screen

positive scalar

Font size to label components on silk screen, specified as a positive scalar in points.

Example: `a.ComponentNameFontSize = 12`, where `a` is a `PCBWriter` object.

Data Types: `double`

`DesignInfoFontSize` — Font size for design information added outside board profile

positive scalar

Design information text font size added outside board profile, specified as a positive scalar.

Example: `a.DesignInfoFontSize = 12`, where `a` is a `PCBWriter` object.

Data Types: `double`

`Font` — Font used for component name and design info

`'Arial'` (default) | character vector

Font used for component name and design info, specified as a character vector.

Example: `a.Font = 'TimesNewRoman'`, where `a` is a `PCBWriter` object.

Data Types: `char` | `string`

`PCBMargin` — Copper free margin around board

`0.5e-3` (default) | positive scalar

Copper free margin around board, specified as a positive scalar in meters.

Example: `a.PCBMargin = 0.7e-3`, where `a` is a `PCBWriter` object.

Data Types: `double`

`Soldermask` — Add solder mask to top and bottom of PCB

`'both'` (default) | `'top'` | `'bottom'` | `'none'`

Add solder mask to top and bottom of PCB, specified as `'both'`, `'top'`, `'bottom'` or `'none'`.

Example: `a.SolderMask = 'top'`, where `a` is a `PCBWriter` object.

Data Types: `char` | `string`

`Solderpaste` — Generate solder paste files

`1` (default) | `0`

Generate solder paste files as a part of PCB stack, specified as `1` or `0`.

Example: `a.SolderPaste = 0`, where `a` is a `PCBWriter` object.

Data Types: `logical`

Object Functions

gerberWrite	Generate Gerber files

Examples

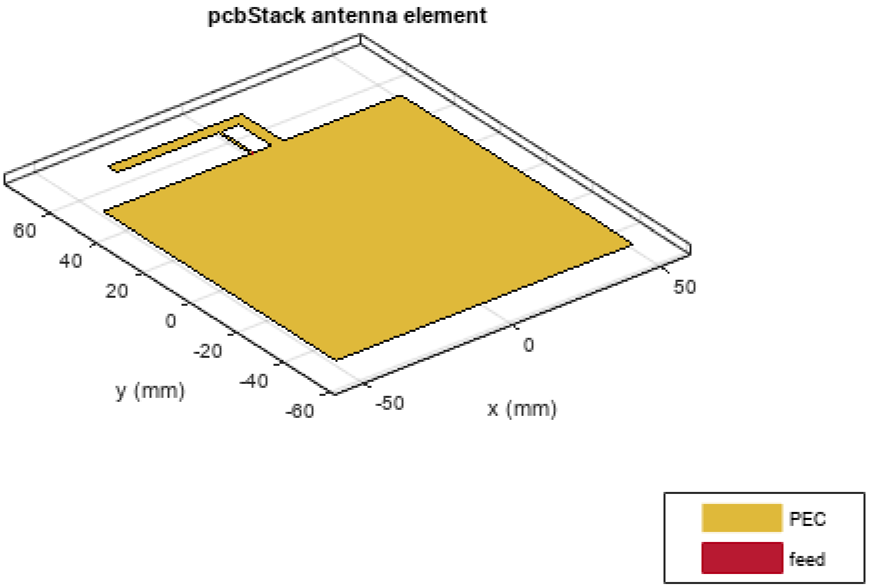
Generate Gerber Format Files From PCB Stack Object

Create a coplanar inverted F antenna

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3, ...
      'GroundPlaneWidth', 100e-3);
```

Create a `pcbStack` object.

```
p = pcbStack(fco);
show (p);
```



Generate a Gerber format design file using PCB Writer.

```
PW = PCBWriter(p)
```

```
PW = 
  PCBWriter with properties:

                       Design: [1x1 struct]
                       Writer: [1x1 Gerber.Writer]
                    Connector: []
          UseDefaultConnector: 1
    ComponentBoundaryLineWidth: 8
         ComponentNameFontSize: []
            DesignInfoFontSize: []
                         Font: 'Arial'
                    PCBMargin: 5.0000e-04
                   Soldermask: 'both'
                  Solderpaste: 1

   See info for details
```

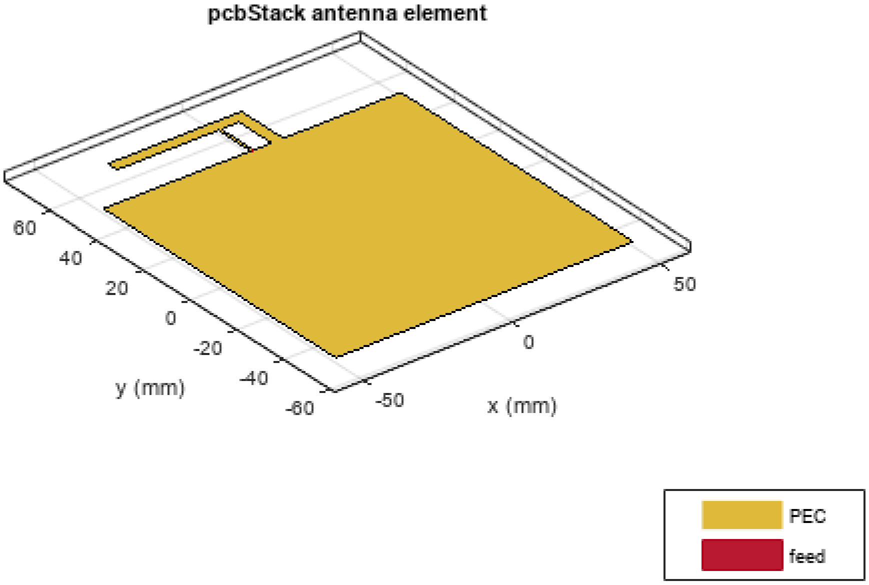
Antenna PCB Design Using SMA Cinch Connector

Create a coplanar inverted F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3, ...
      'GroundPlaneWidth', 100e-3);
```

Create a `pcbStack` object.

```
p = pcbStack(fco);
show(p)
```



Create an SMA_Cinch connector using the `PCBConnectors` object.

```
c = PCBConnectors.SMA_Cinch
```

```
c = 
  SMA_Cinch with properties:

                  Type: 'SMA'
                   Mfg: 'Cinch'
                  Part: '142-0711-202'
            Annotation: 'SMA'
             Impedance: 50
             Datasheet: 'https://belfuse.com/resources/Johnson/drawings/dr-142-0711-202.pdf'
              Purchase: 'https://www.digikey.com/product-detail/en/cinch-connectivity-solutions
             TotalSize: [0.0071 0.0071]
         GroundPadSize: [0.0024 0.0024]
     SignalPadDiameter: 0.0017
       PinHoleDiameter: 0.0013
         IsolationRing: 0.0041
   VerticalGroundStrips: 1
   Cinch 142-0711-202 (Example Purchase)
```

Create an antenna PCB design file using the connector.

```
PW = PCBWriter(p,c)
```

```
PW = 
  PCBWriter with properties:

                        Design: [1x1 struct]
                        Writer: [1x1 Gerber.Writer]
                     Connector: [1x1 PCBConnectors.SMA_Cinch]
           UseDefaultConnector: 0
    ComponentBoundaryLineWidth: 8
         ComponentNameFontSize: []
            DesignInfoFontSize: []
                          Font: 'Arial'
                     PCBMargin: 5.0000e-04
                    Soldermask: 'both'
                    Solderpaste: 1

   See info for details
```

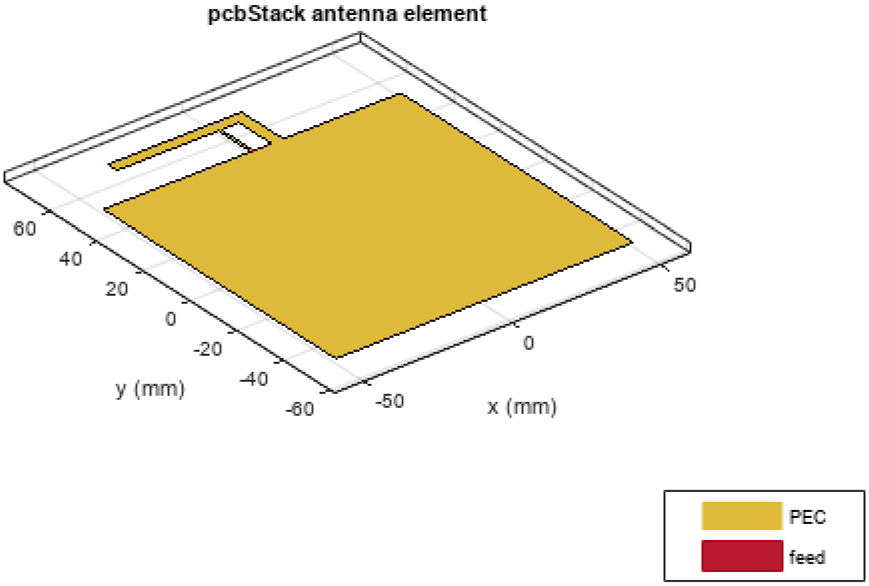
Antenna Design Files Using Advanced Circuits Writer Service

Create a coplanar inverted-F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3, ...
      'GroundPlaneWidth', 100e-3);
```

Create a `pcbStack` object.

```
p = pcbStack(fco);
show(p)
```



Use an Advanced Circuits Writer as a PCB manufacturing service.

```
s = PCBServices.AdvancedCircuitsWriter
```

```
s = 
  AdvancedCircuitsWriter with properties:

                BoardProfileFile: 'legend'
           BoardProfileLineWidth: 1
                  CoordPrecision: [2 6]
                      CoordUnits: 'in'
               CreateArchiveFile: 1
                  DefaultViaDiam: 3.0000e-04
              DrawArcsUsingLines: 0
                  ExtensionLevel: 1
                        Filename: 'untitled'
                           Files: {}
          IncludeRootFolderInZip: 0
                    PostWriteFcn: @(obj)sendTo(obj)
     SameExtensionForGerberFiles: 0
                     UseExcellon: 1
```

Create an antenna PCB design file using the above service.

```
PW = PCBWriter(p,s)
```

```
PW = 
  PCBWriter with properties:
```

```
                        Design: [1x1 struct]
                        Writer: [1x1 PCBServices.AdvancedCircuitsWriter]
                     Connector: []
           UseDefaultConnector: 1
    ComponentBoundaryLineWidth: 8
         ComponentNameFontSize: []
            DesignInfoFontSize: []
                          Font: 'Arial'
                     PCBMargin: 5.0000e-04
                    Soldermask: 'both'
                   Solderpaste: 1

   See info for details
```

Show Antenna PCB Design Using Mayhew Manufacturing Service

Create a coplanar inverted F antenna.

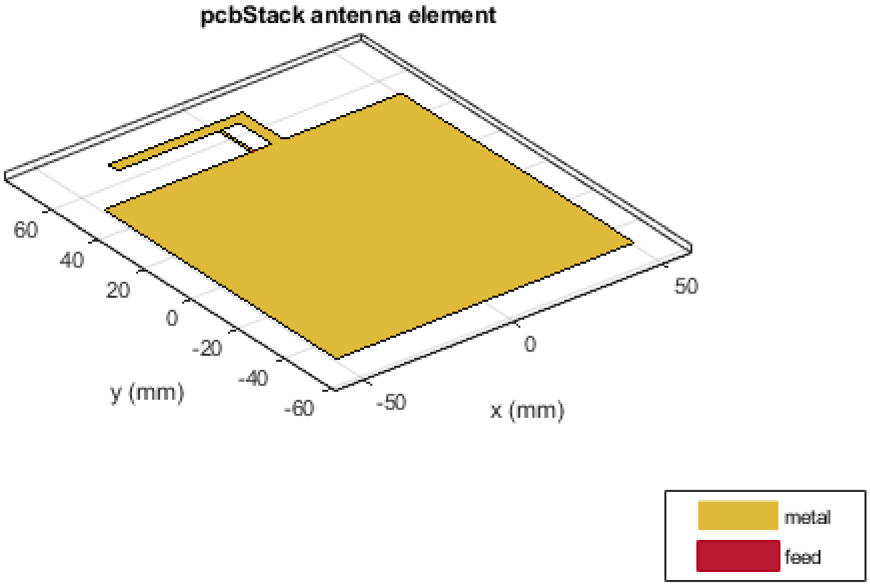
```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Use this antenna in creating a `pcbStack` object.

```
p = pcbStack(fco)
```

```
p = 
  pcbStack with properties:

              Name: 'Coplanar Inverted-F'
          Revision: 'v1.0'
        BoardShape: [1×1 antenna.Rectangle]
    BoardThickness: 0.0013
            Layers: {[1×1 antenna.Polygon]}
     FeedLocations: [0 0.0500 1]
      FeedDiameter: 5.0000e-04
      ViaLocations: []
       ViaDiameter: []
      FeedViaModel: 'strip'
       FeedVoltage: 1
         FeedPhase: 0
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1×1 lumpedElement]
```

```
figure
show(p)
```



Use an SMA_Cinch as an RF connector and Mayhew Writer as a 3-D viewer.

```
c = PCBConnectors.SMA_Cinch
```

```
c = 
  SMA_Cinch with properties:

                  Type: 'SMA'
                   Mfg: 'Cinch'
                  Part: '142-0711-202'
            Annotation: 'SMA'
             Impedance: 50
             Datasheet: 'https://belfuse.com/resources/Johnson/drawings/dr-142-0711-202.pdf'
              Purchase: 'https://www.digikey.com/product-detail/en/cinch-connectivity-solutions
             TotalSize: [0.0071 0.0071]
         GroundPadSize: [0.0024 0.0024]
     SignalPadDiameter: 0.0017
       PinHoleDiameter: 0.0013
         IsolationRing: 0.0041
   VerticalGroundStrips: 1

   Cinch 142-0711-202 (Example Purchase)
```

```
s = PCBServices.MayhewWriter
```

```
s = 
  MayhewWriter with properties:
```

```
               BoardProfileFile: 'legend'
          BoardProfileLineWidth: 1
                  CoordPrecision: [2 6]
                      CoordUnits: 'in'
               CreateArchiveFile: 0
                  DefaultViaDiam: 3.0000e-04
              DrawArcsUsingLines: 1
                  ExtensionLevel: 1
                        Filename: 'untitled'
                           Files: {}
         IncludeRootFolderInZip: 0
                     PostWriteFcn: @(obj)sendTo(obj)
    SameExtensionForGerberFiles: 0
                      UseExcellon: 1
```

Create an antenna design file using `PCBWriter` .

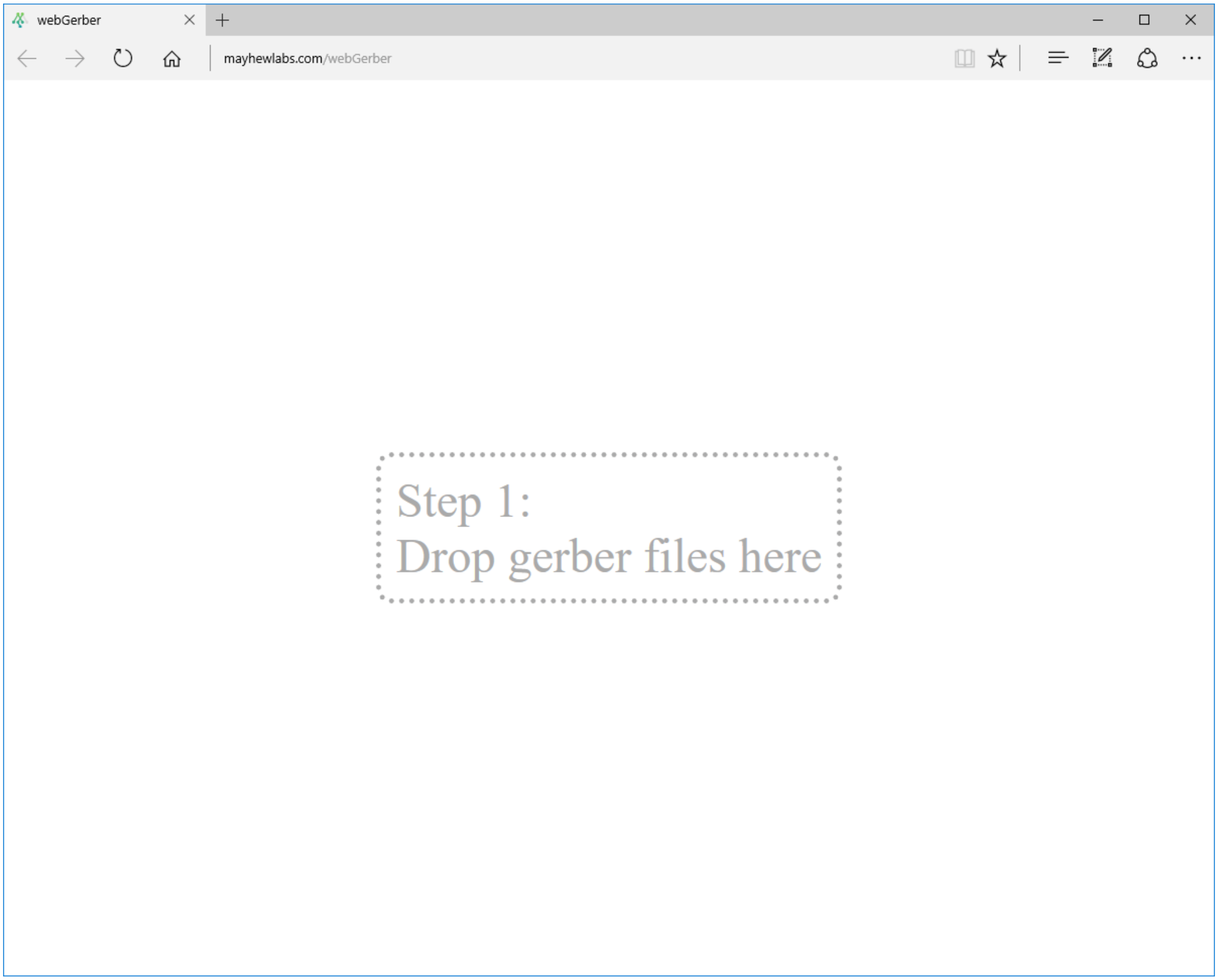
```
PW = PCBWriter(p,s,c)
```

```
PW = 
  PCBWriter with properties:

                        Design: [1×1 struct]
                        Writer: [1×1 PCBServices.MayhewWriter]
                     Connector: [1×1 PCBConnectors.SMA_Cinch]
           UseDefaultConnector: 0
    ComponentBoundaryLineWidth: 8
         ComponentNameFontSize: []
            DesignInfoFontSize: []
                          Font: 'Arial'
                     PCBMargin: 5.0000e-04
                    Soldermask: 'both'
                   Solderpaste: 1

   See info for details
```

Use the gerberWrite method to create gerber files from the antenna design files. The files generated are then send to the Mayhew writer manufacturing service.

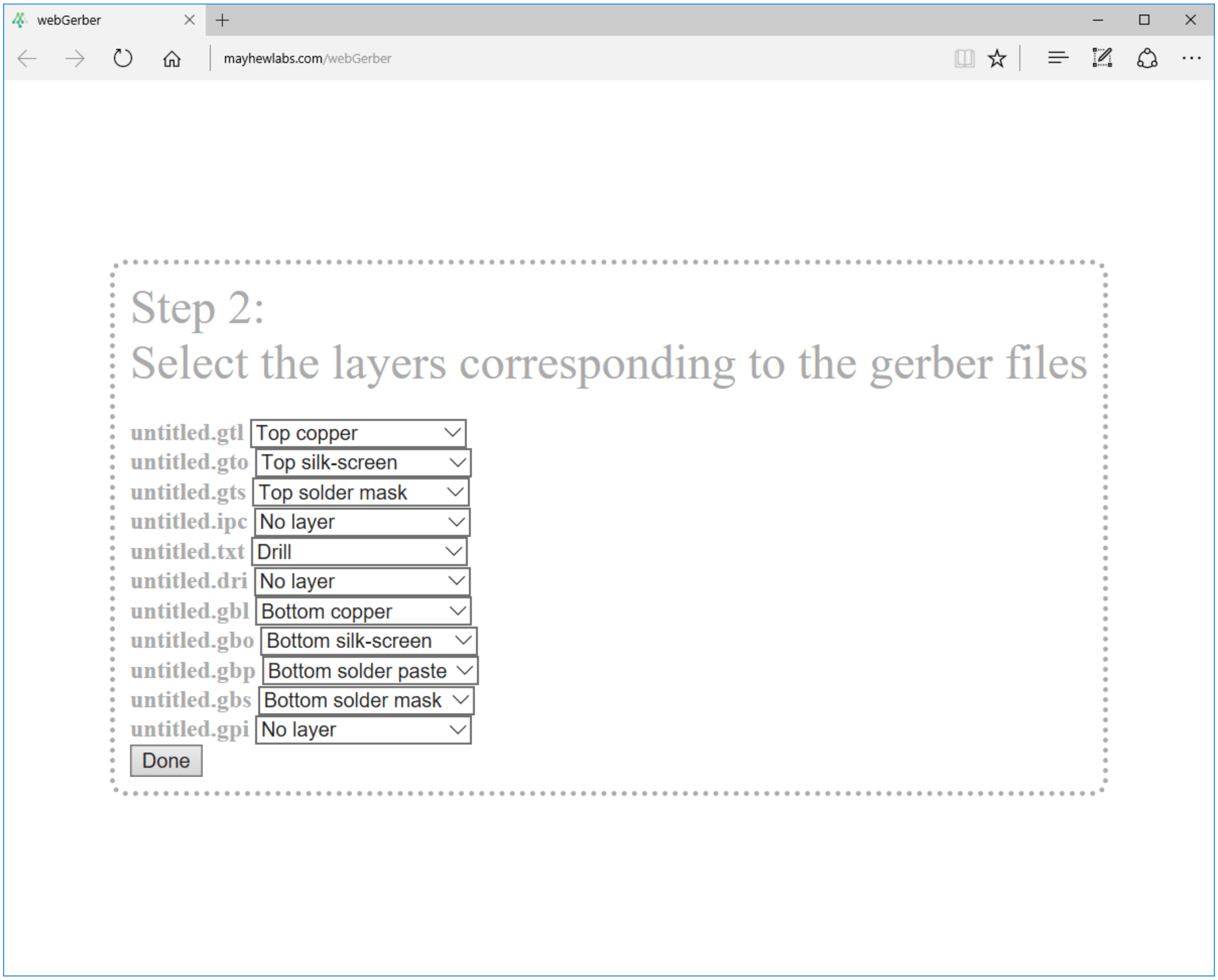
```
gerberWrite(PW)
```

By default, the folder containing the gerber files is called "untitled" and is located in your MATLAB folder. Running this example automatically opens up the Mayhew Labs PCB manufacturing service in your internet browser.



Step 1:
Drop gerber files here

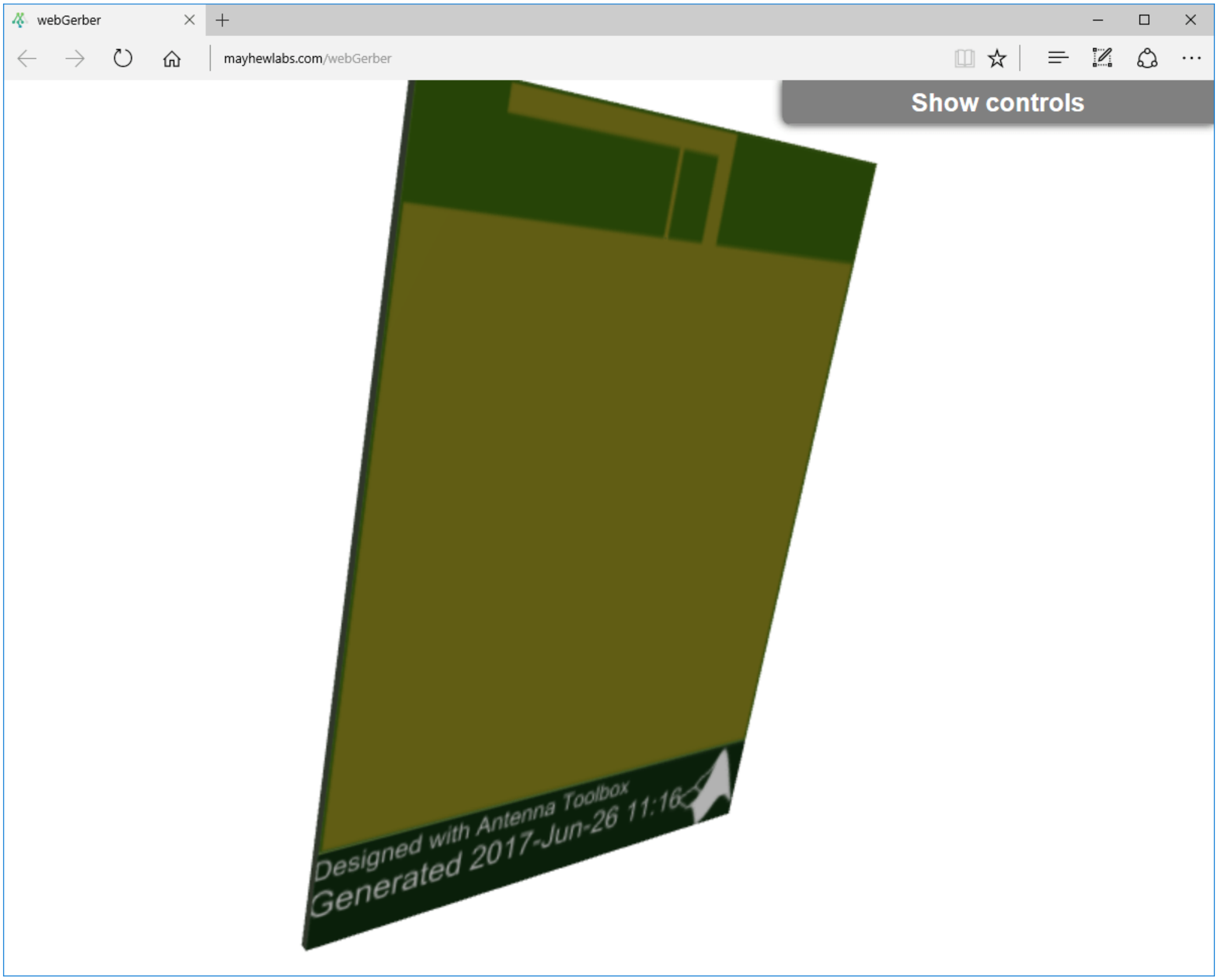
Drag and drop all your files from the "untitled" folder.



Step 2:
Select the layers corresponding to the gerber files

untitled.gtl Top copper
untitled.gto Top silk-screen
untitled.gts Top solder mask
untitled.ipc No layer
untitled.txt Drill
untitled.dri No layer
untitled.gbl Bottom copper
untitled.gbo Bottom silk-screen
untitled.gbp Bottom solder paste
untitled.gbs Bottom solder mask
untitled.gpi No layer

Click **Done** to view your Antenna PCB.



Version History

Introduced in R2017b

See Also

`PCBServices` | `PCBConnectors`

PCBServices

Customize PCB file generation for PCB manufacturing service

Description

Use the `PCBServices` object to customize printed circuit board (PCB) file generation for a PCB manufacturing service.

Creation

Syntax

```
w = PCBServices.servicetype
```

Description

`w = PCBServices.servicetype` creates a Gerber file based on the type of service specified in `servicetype`.

Input Arguments

`servicetype` — Type of service from PCB services package
character vector

Type of service from PCB services package, specified as one of the following:

- AdvancedCircuitsWriter – Configure Gerber file generation for Advanced Circuits manufacturing.
- CircuitPeopleWriter – Configure Gerber file generation for CircuitPeople online viewer.
- DirtyPCBsWriter – Configure Gerber file generation for Dirty PCBs manufacturing.
- EuroCircuitsWriter – Configure Gerber file generation for EuroCircuits online viewer.
- GerberLookWriter – Configure Gerber file generation for GerbLook online viewer.
- GerberViewerWriter – Configure Gerber file generation for GerberViewer online viewer.
- MayhewWriter – Configure Gerber file generation for Mayhew Labs online 3-D viewer.
- OSHParkWriter – Configure Gerber file generation for OSH Park PCB manufacturing.
- PCBWayWriter – Configure Gerber file generation for PCBWay PCB manufacturing.
- ParagonWriter – Configure Gerber file generation for Paragon Robotics online viewer.
- SeeedWriter – Configure Gerber file generation for Seeed Fusion PCB manufacturing.
- SunstoneWriter – Configure Gerber file generation for Sunstone PCB manufacturing.
- ZofzWriter – Configure Gerber file generation for Zofz 3-D viewer.

Example: `w = PCBServices.SunstoneWriter` creates Gerber files configured to use Sunstone PCB manufacturing service.

Output Arguments

`w` — PCB manufacturing service
object

PCB manufacturing service, returned as an object.

Properties

`BoardProfileFile` — File type for board profile
`'legend'` | `'profile'`

File type for board profile, specified as `'legend'` or `'profile'`.

Example: `w = PCBServices.SunstoneWriter`; `w.BoardProfileFile = 'profile'`.

Data Types: `char` | `string`

`BoardProfileLineWidth` — Width of line
`1` | positive scalar

Width of line, specified as a positive scalar in mils.

PCB manufacturers vary on board profile. The most common line width is zero of a fraction width in the chosen unit, for example, 0.1 mil.

Example: `w = PCBServices.SunstoneWriter`; `w.BoardProfileLineWidth = 0.1`

Data Types: `double`

`CoordPrecision` — Precision of X and Y coordinates written to file
`[2 6]` | 1-by-2 vector

Precision of X and Y coordinates written to file, specified as a 1-by2 vector [*I F*], where,

- *I* – Number of digits in the integer part, 0<=*I*<=6.
- *F* – Number of digits in the fractional part, 4<=*F*<=6.

Example: `w = PCBServices.SunstoneWriter`; `w.CoordPrecision = [1 3]`

Data Types: `double`

`CoordUnits` — Units of X and Y coordinate
`'in'` | `'mm'`

Units of X and Y coordinates, specified as inches or millimeters.

Example: `w = PCBServices.SunstoneWriter`; `w.CoordUnits = 'mm'`

Data Types: `char` | `string`

`CreateArchiveFile` — Creates single archive file with all Gerber files
`1` (default) | `0`

Creates single archive file with all Gerber files, specified as `1` or `0`.

Example: `w = PCBServices.SunstoneWriter`; `w.CreateArchiveFile = 0`

Data Types: `logical`

DefaultViaDiameter — Via drill diameter

3.0000e-04 | positive scalar

Via drill diameter, specified as a positive scalar in meters. PCB manufacturers also call it minimum drilling hole diameter.

Example: w = PCBServices.SunstoneWriter; w.DefaultViaDiameter = 0.1

Data Types: double

DrawArcsUsingLines — Force arcs to be drawn using lines

0 | 1

Force arcs to be drawn using lines, specified as 1 or 0.

Example: w = PCBServices.SunstoneWriter; w.DrawArcsUsingLines = 0

Data Types: logical

ExtensionLevel — Feature content for Gerber file format

1 (default) | 2

Feature content for Gerber file format, specified as:

- 1 - Extension 1 is the most compatible setting for downstream PCB manufacturing tools.
- 2 - Extension 2 adds file attributes %TF.<attr>*%" to the header and footer of Gerber files.

Example: w = PCBServices.SunstoneWriter; w.ExtensionLevel = 2

Data Types: double

Filename — Name of all files containing Gerber design

'untitled' (default) | character vector

Name of all files containing Gerber design, specified as a character vector.

Example: w = PCBServices.SunstoneWriter; w.Filename = 'antenna_design'.

Data Types: char | string

Files — Define stack of PCB files

character vector

Define stack of PCB files, specified as a character vector. This definition includes:

- Multiples files describing one PCB.
- A "file" as a memory object containing buffers that describe or hold the file content before the file is written.
- Cell vector of Gerber.FileFunction objects, one per file.

Data Types: cell | char | string

IncludeRootFolderInZip — Include top-level folder in zip archive

1 | 0

Include top-level folder in zip archive, specified as 1 or 0.

Example: w = PCBServices.SunstoneWriter; w.IncludeRootFolderInZip = 0

Data Types: logical

PostWriteFcn — Function to invoke after a successful write operation
function handle (default)

Function to invoke after a successful write operation, specified as a function handle. In this case, it is the sendTo function. This property makes sure that the location of the Gerber files and the website of the manufacturing service is open after a successful write function.

Example: w = PCBServices.SunstoneWriter; w.PostWriteFcn = @(obj)sendTo(obj)

Data Types: function_handle

SameExtensionForGerberFiles — Use .gbr to be file extension for all Gerber files
0 | 1

Use .gbr to be file extension for all Gerber files, specified as 0 or 1.

Example: w = PCBServices.SunstoneWriter; w.SameExtensionForGerberFiles = 1

Data Types: logical

UseExcellon — Generate Excellon drill files
1 | 0

Generate Excellon drill files, specified as 0 or 1.

Example: w = PCBServices.SunstoneWriter; w.UseExcellon = 1, generates Gerber format drill files with 'x2' extension.

Data Types: logical

Examples

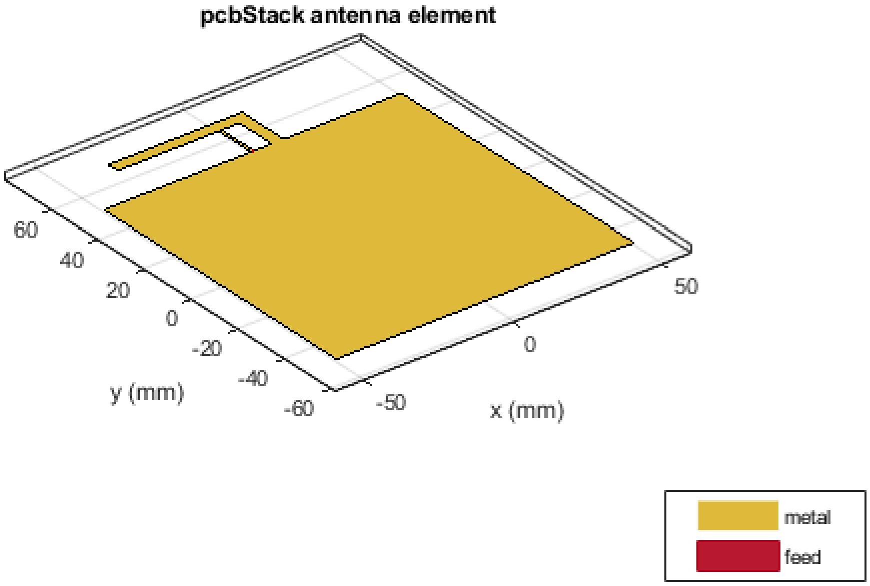
PCB Using Mayhew Labs 3-D Viewer

Create a coplanar inverted F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Use this antenna in creating a PCB stack object.

```
p = pcbStack(fco);
figure;
show(p)
```



Use a Mayhew Writer with a `profile` board for viewing the PCB in 3D.

```
s = PCBServices.MayhewWriter;
s.BoardProfileFile = 'profile'
```

```
s =
  MayhewWriter with properties:

             BoardProfileFile: 'profile'
        BoardProfileLineWidth: 1
               CoordPrecision: [2 6]
                   CoordUnits: 'in'
            CreateArchiveFile: 0
              DefaultViaDiam: 3.0000e-04
           DrawArcsUsingLines: 1
              ExtensionLevel: 1
                    Filename: 'untitled'
                       Files: {}
         IncludeRootFolderInZip: 0
                 PostWriteFcn: @(obj)sendTo(obj)
    SameExtensionForGerberFiles: 0
                  UseExcellon: 1
```

Create an antenna design file using `PCBWriter`.

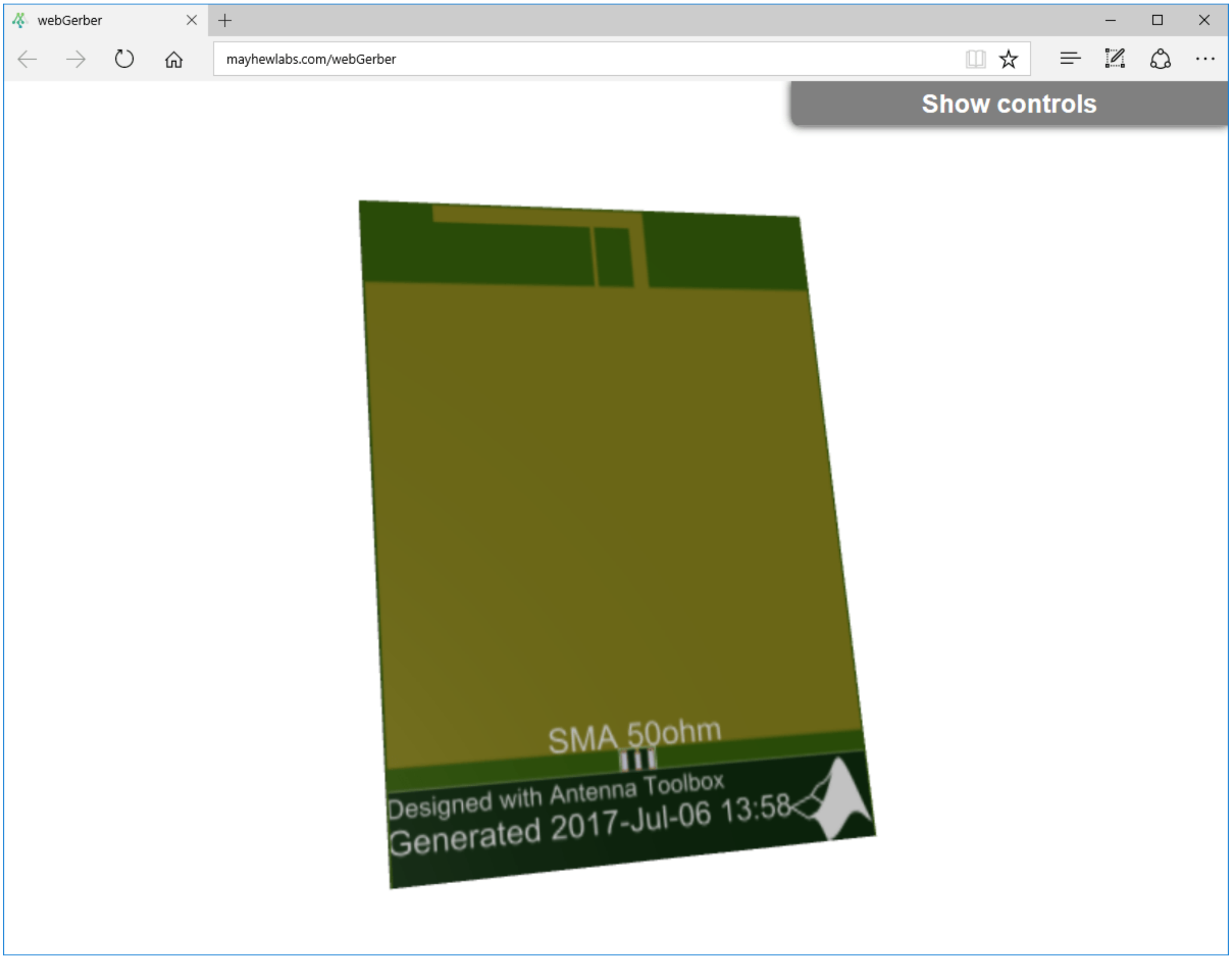
```
PW = PCBWriter(p,s);
```

Use the `gerberWrite` method to create Gerber files from the antenna design files.

```
gerberWrite(PW)
```

The location of the folder and the Mayhew labs website opens automatically.

To view the board, drag and drop the files. Click **Done**.



Version History

Introduced in R2017b

See Also

`PCBWriter` | `PCBConnectors` | `gerberWrite`

PCBConnectors

RF connector at antenna feedpoint

Description

Use `PCBConnectors` object to specify RF connectors used for antenna printed circuit board (PCB) feed points. The result is generally a set of modifications to the PCB design files. The changes to the PCB include new copper landing pads and traces, and changes to solder mask, silk screen, and solder paste files.

Creation

Syntax

```
c = PCBConnectors.connectortype
```

Description

`c = PCBConnectors.connectortype` creates Gerber files based on the type of connector to use at antenna feedpoint specified in `connectortype`.

Input Arguments

`connectortype` — Type of connector from PCB connector package
character vector

Type of connector from PCB connector package, specified as one of the following:

- Coax Connectors - Coax RG11, RG174, RG58, and RG59 connectors directly soldered to PCB pads.
- IPX Connectors - LightHorse IPX SMT jack or plug surface mount RF connector.
- MMCX Connectors - MMCX Cinch or Samtec surface mount RF connectors.
- SMA Connectors - Generic 5-pad SMA surface mount RF connectors, with four corner rectangular pads, one round center pin. Cinch and Multicomp SMA RF connectors.
- SMAEdge Connectors- Generic SMA edge-launch surface mount RF connector. Cinch and Samtec SMA edge-launch RF connectors.
- SMB Connectors - Johnson/Emerson and Pasternack SMB surface mount RF connectors.
- SMC Connectors - Pasternack SMC and SMC edge-launch surface mount RF connectors.
- Coaxial Cable Connectors - Semi-rigid `0.020 inch`, `0.034 inch`, `0.047 inch`, and `0.118 inch` coaxial cable soldered to PCB pads.

For list of connectors, see “PCB Connectors List” on page 1-367.

Example: `c = PCBConnectors.Semi_020` creates Gerber files configured to use semi-rigid `0.020 inch` coaxial cables.

Output Arguments

`c` — PCB connector
object

PCB connector, returned as an object.

Properties

Common Properties for All Connectors

`Type` — Type of connector
character vector

This property is read-only.

Type of connector, specified as a character vector.

Example: `'Coax_RG11'`

Data Types: `char` | `string`

`Mfg` — Name of component manufacturer
character vector

This property is read-only.

Name of component manufacturer, specified as a character vector.

Example: `'Belden'`

Data Types: `char` | `string`

`Part` — Manufacturer part number
character vector | string

This property is read-only.

Manufacturer part number, specified as a character vector or string.

Example: `'RG11'`

Data Types: `char` | `string`

`Annotation` — Text added to PCB to identify component
character vector

This property is read-only.

Text added to PCB to identify component, specified as a character vector.

Example: `'RG59U'`

Data Types: `char` | `string`

`Impedance` — Connector impedance
`50` | positive scalar

This property is read-only.

Connector impedance, specified as a positive scalar in ohms.

Example: `c = PCBConnectors.MMCX_Cinch; c.Impedance = 70;`

Data Types: `double`

`Datasheet` — URL for component specifications
character vector

This property is read-only.

URL for component specifications, specified as a character vector. Data sheets are typically PDF files.

Data Types: `char` | `string`

`Purchase` — URL for purchasing connector
character vector

This property is read-only.

URL for purchasing connector, specified as a character vector.

Data Types: `char` | `string`

Common Properties for All Coax Connectors

`PinDiameter` — Circular pad diameter
positive scalar

Circular pad diameter connecting the signal wire of the coax to the feedpoint, specified as a positive scalar in meters. The pin diameter is greater than the diameter of the signal wire.

Example: `c = PCBConnectors.Coax_RG59; c.PinDiameter = 1.0000e-03;`

Data Types: `double`

`DielectricDiameter` — Dielectric diameter
positive scalar

Dielectric diameter (white material around signal wire), specified as a positive scalar in meters. Dielectric diameter specifies the size of the non-conductive isolation ring on the PCB between the signal wire and the ground plane.

Example: `c = PCBConnectors.Coax_RG59; c.DielectricDiameter = 0.0073;`

Data Types: `double`

`ShieldDiameter` — Ground ring diameter
positive scalar

Ground ring diameters used to solder coax shield, specified as a positive scalar in meters.

Example: `c = PCBConnectors.Coax_RG59; c.ShieldDiameter = 0.0085;`

Data Types: `double`

`AddThermals` — Thermal relief
`1` | `0`

Thermal relief around coaxial shield connection, specified as `0` or `1`. Thermal relief reduces the heat needed to solder the coax shield to the ground.

Example: `c = PCBConnectors.Coax_RG59; c.AddThermals = 0;`

Data Types: `logical`

`ThermalsDiameter` — Arc-shaped gaps outer diameter

positive scalar

Arc-shaped gaps outer diameter in the ground plane, specified as a positive scalar in meters.

Example: `c = PCBConnectors.Coax_RG59; c.ThermalsDiameter = 0.0100;`

Data Types: `double`

`ThermalsBridgeWidth` — Width of four conductive bridges

positive scalar

Width of four conductive bridges created across thermal gap, specified as a positive scalar in meters. The bridges are established during electrical grounding.

Example: `c = PCBConnectors.Coax_RG59; c.ThermalBridgeWidth = 0.0015;`

Data Types: `double`

Common Properties for All 5-Pad Symmetric Surface Mount Connectors

`TotalSize` — Total length of each side of rectangular connector footprint

two-element vector

Total length of each side of rectangular connector footprint, specified as a two-element vector with each element unit in meters.

Example: `c = PCBConnectors.SMA_Multicomp; c.TotalSize = [0.0063 0.0063];`

Data Types: `double`

`GroundPadSize` — Length of each side of ground pad

two-element vector

Length of each side of ground pad, specified as a two-element vector with each element unit in meters. The pads are located in each of the four corners of the connector footprint.

Example: `c = PCBConnectors.SMA_Multicomp; c.GroundPadSize = [0.0016 0.0016];`

Data Types: `double`

`SignalPadDiameter` — Circular pad diameter

positive scalar

Circular pad diameter connecting the signal pin of the coax connector, specified as a positive scalar in meters. The pad is at the center of the connector footprint.

Example: `c = PCBConnectors.SMA_Multicomp; c.SignalPadDiameter = 0.0012;`

Data Types: `double`

`PinHoleDiameter` — Via pin diameter

positive scalar

Via pin diameter, specified as a positive scalar in meters.

Example: `c = PCBConnectors.SMA_Multicomp; c.ViaPinDiameter = 0.0012;`

Data Types: `double`

`IsolationRing` — Diameter of isolation ring that removes semicircle of copper from inner corner of ground pads

scalar

Diameter of isolation ring that removes semicircle of copper from inner corner of ground pads, specified as a scalar in meters.

Example: `c = PCBConnectors.SMA_Multicomp; c.IsoltationRing =0.0012;`

Data Types:

`VerticalGroundStrips` — Vertical ground strips between upper and lower ground pads

scalar

Vertical ground strips between upper and lower ground pads, specified as a scalar.

Example: `c = PCBConnectors.SMA_Multicomp; c.VerticalGroundStrips = 1;`

Data Types: `double`

Common Properties for All Edge-Launch Surface Mount Connectors

`GroundPadSize` — Ground pad size

two-element vector

Ground pad size, specified as a two-element vector with each element unit in meters.

Example: `c = PCBConnectors.SMAEdge; c.GroundPadSize = [0.0014 0.0042];`

Data Types: `double`

`GroundSeparation` — Space between ground pads

positive scalar

Space between ground pads on the ground side of the board, specified as a positive scalar in meters.

Example: `c = PCBConnectors.SMAEdge; c.GroundSeparation = 0.0043;`

Data Types: `double`

`GroundPadIsolation` — Width of copper removed around top layer ground pads

positive scalar

Width of copper removed around top layer ground pads, specified as a positive scalar in meters. This property isolates the ground pads from any signal traces or structures.

Example: `c = PCBConnectors.SMAEdge; c.GroundPadIsolation = 2.5000e-04;`

Data Types: `double`

`SignalPadSize` — Signal pad size

two-element vector

Signal pad size, specified as a two-element vector with each element unit in meters.

Example: `c = PCBConnectors.SMAEdge; c.SignalPadSize = [0.0013 0.0036];`

Data Types: `double`

`SignalGap` — Gap between PCB edge and start of signal pad copper

positive scalar

Gap between PCB edge and start of signal pad copper, specified as a positive scalar in meters.

Example: `c = PCBConnectors.SMAEdge; c.SignalGap = 1.0000e-04;`

Data Types: `double`

`SignalLineWidth` — Width of signal trace

positive scalar

Width of signal trace extending from the signal pad to the feedpoint location, specified as a positive scalar in meters.

Example: `c = PCBConnectors.SMAEdge; c.SignalLineWidth = 8.0000e-04;`

Data Types: `double`

`EdgeLocation` — PCB side that receives edge connector

`'north'` | `'south'` | `'east'` | `'west'`

PCB side that receives edge connector, specified as `'north'`, `'south'`, `'east'`, `'west'`.

Example: `c = PCBConnectors.SMAEdge; c.EdgeLocation = 'south';`

Data Types: `char`

`ExtendBoardProfile` — Extend PCB to add connector beyond design area

`0` | `1`

Extend PCB to add connector beyond design area, specified as `0` or `1`

Example: `c = PCBConnectors.SMAEdge; c.ExtendBoardProfile = 1;`

Data Types: `logical`

`FillGroundSide` — Fill connector region on ground side of board with copper

`0` | `1`

Fill connector region on ground side of the board with copper, specified as `0` or `1`

Example: `c = PCBConnectors.SMAEdge; c.FillGroundSide = 1;`

Data Types: `logical`

Common Properties for All Staggered Surface Mount Connectors

`GroundPadSize` — Ground pad size

two-element vector

Ground pad size, specified as a two-element vector with each element unit in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthorse; c.GroundPadSize = [0.0010 0.0022];`

Data Types: `double`

GroundPadXSeparation — Distance between pair of ground pads along X-axis

positive scalar

Distance between pair of ground pads along X-axis, specified as a positive scalar in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthorse; c.GroundPadXSeparation = 0.0019;`

Data Types: `double`

GroundPadYOffset — Y-offset from signal pad to signal pad center line

positive scalar

Y-offset from signal pad to signal pad center line, specified as a positive scalar in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthorse; c.GroundPadYOffset = 0.0015;`

Data Types: `double`

SignalPadSize — Signal pad size

2-element vector

Signal pad size, specified as a 2-element vector with each element unit in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthorse; c.SignalPadSize = [1.0000e-03 1.0000e-03];`

Data Types: `double`

SignalMinYSeparation — Minimum separation from ground at bottom or top for signal pad

positive scalar

Minimum separation from ground at bottom or top for signal pad, specified as a positive scalar in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthorse; c.SignalMinYSeparation = 1.0000e-03;`

Data Types: `double`

Examples

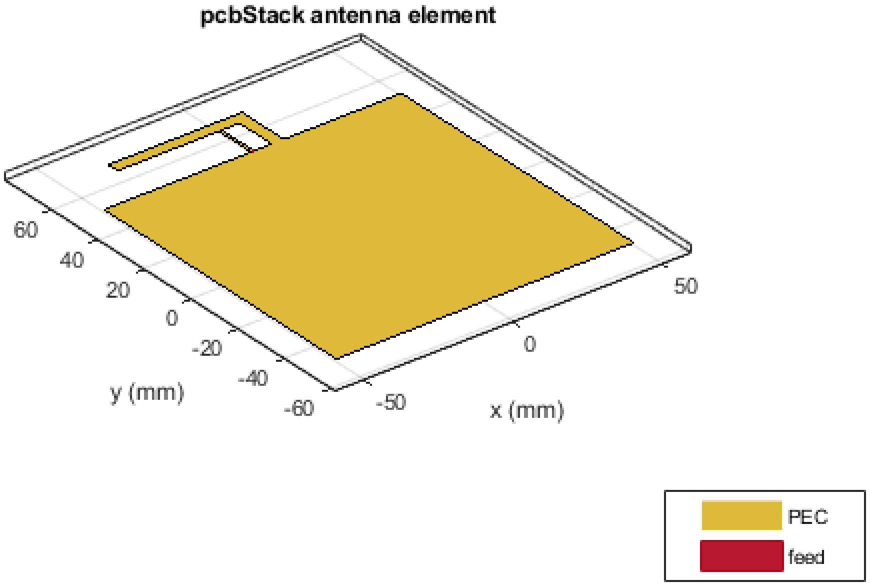
PCB Using Coax_RG11 Connector

Create a coplanar inverted F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Use this antenna to create a `pcbStack` object.

```
p = pcbStack(fco);
show(p);
```



Use a Coax_RG11 RF connector with a pin diameter of 2 mm.

```
c = PCBConnectors.Coax_RG11;
c.PinDiameter = 2.000e-03;
s = PCBServices.MayhewWriter;
```

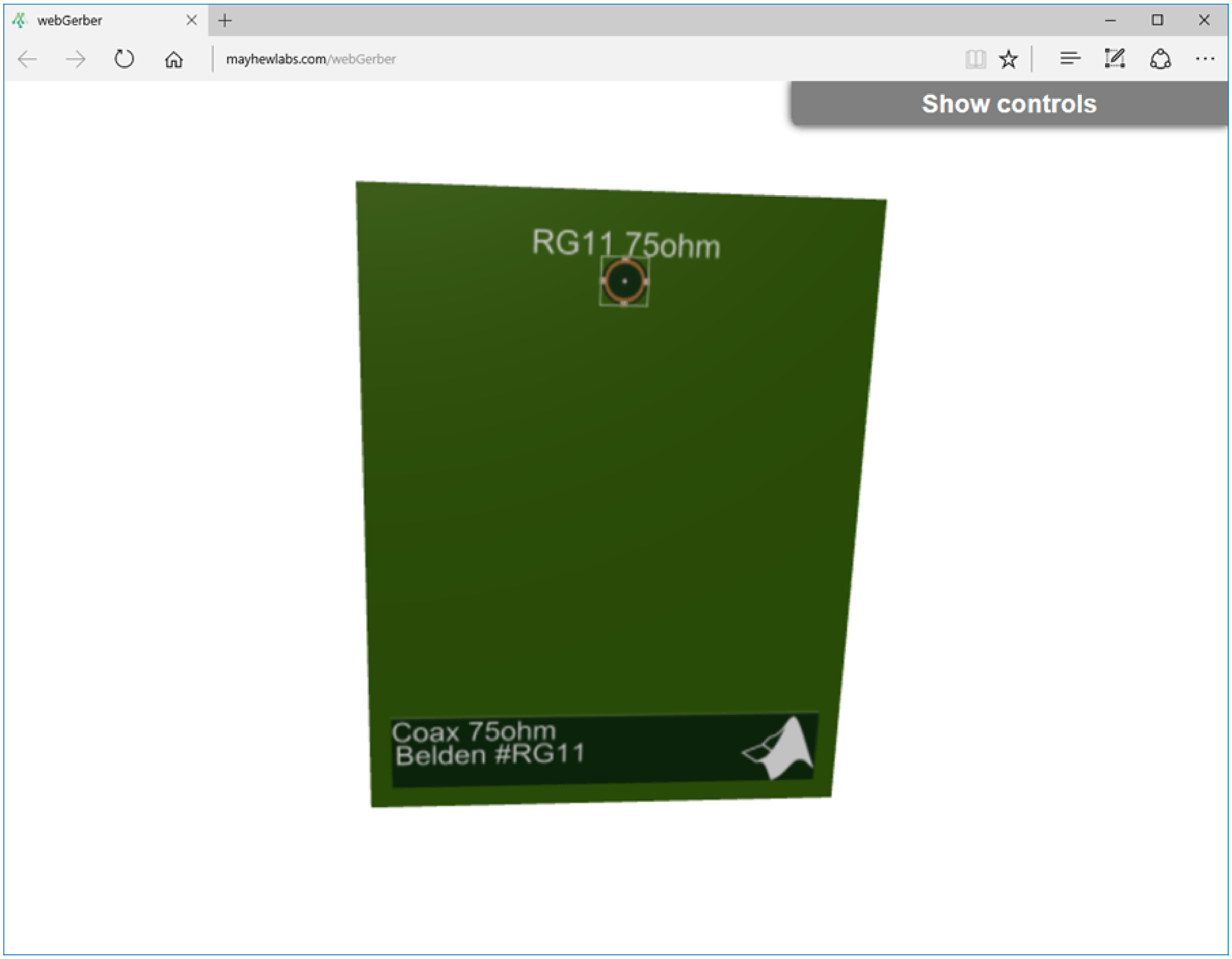
Create an antenna design file using `PCBWriter` .

```
PW = PCBWriter(p,s,c);
```

Use the `gerberWrite` method to create Gerber files from the antenna design files.

```
gerberWrite(PW)
```

To view the board, drag and drop the files. Click **Done**.



Authoring Custom RF Connector

This example shows how to define custom RF connector class.

```
classdef SMA_Jack_Cinch < PCBConnectors.BaseSMT5PadSymmetric
    % Cinch SMA surface mount RF connector.

    properties (Constant) % Abstract
        Type       = 'SMA'
        Mfg        = 'Cinch'
        Part       = '142-0701-631'
        Annotation = 'SMA'
        Impedance  = 50
        Datasheet  = 'https://www.farnell.com/datasheets/1720451.pdf?_ga=2.164811836.2075200750.1
        Purchase   = 'https://www.newark.com/johnson/142-0701-631/rf-coaxial-sma-jack-straight-50
    end

    methods
        function RFC = SMA_Jack_Cinch
            RFC.TotalSize           = [0.5 0.5]*25.4e-3;
            RFC.GroundPadSize       = [0.102 0.102]*25.4e-3;
            RFC.SignalPadDiameter   = 0.1*25.4e-3;
            RFC.PinHoleDiameter     = 1.27e-3;
            RFC.IsolationRing       = 0.22*25.4e-3;
            RFC.VerticalGroundStrips = false;
        end
    end
end
```

More About

PCB Connectors List

PCB Connectors	Descriptions
PCBConnectors.Coax_RG11	RG11 coaxial cable direct soldered to PCB pads.
PCBConnectors.Coax_RG58	RG58 coaxial cable direct soldered to PCB pads.
PCBConnectors.Coax_RG59	RG59 coaxial cable direct soldered to PCB pads.
PCBConnectors.Coax_RG174	RG174 coaxial cable direct soldered to PCB pads.
PCBConnectors.SMA	Generic 5-pad SMA surface mount RF connector, with four corner rectangular ground pads, one round.
PCBConnectors.SMAEdge	Generic SMA edge-launch surface mount RF connector.
PCBConnectors.SMACinch	Cinch SMA surface mount RF connector
PCBConnectors.SMAEdge_Cinch	Cinch SMA edge-launch surface mount RF connector
PCBConnectors.SMAEdge_Samtec	Samtec SMA edge-launch surface mount RF connector
PCBConnectors.SMAEdge_Amphenol	Amphenol SMA edge-launch surface mount RF connector
PCBConnectors.SMAEdge_Linx	Linx SMA edge-launch surface mount RF connector
PCBConnectors.SMA_Multicomp	Multicomp SMA surface mount RF connector
PCBConnectors.SMB_Johnson	Johnson/Emerson SMB surface mount RF connector
PCBConnectors.SMB_Pasternack	Pasternack SMB surface mount RF connector
PCBConnectors.SMC_Pasternack	Pasternack SMC surface mount RF connector
PCBConnectors.SMCEdge_Pasternack	Pasternack SMC edge-launch surface mount RF connector
PCBConnectors.MMCX_Cinch	Cinch MMCX surface mount RF connector
PCBConnectors.MMCX_Samtec	Samtec MMCX surface mount RF connector
PCBConnectors.IPX_Jack_LightHorse	LightHorse IPX SMT jack surface mount RF connector

PCB Connectors	Descriptions
PCBConnectors.IPX_Plug_LightHorse	LightHorse IPX SMT plug surface mount RF connector
PCBConnectors.UFL_Hirose	Hirose u.fl surface mount RF connector
PCBConnectors.Semi_020	Pasternack semi-rigid 0.020" coaxial cable soldered to PCB pads
PCBConnectors.Semi_034	Pasternack semi-rigid 0.020" coaxial cable soldered to PCB pads
PCBConnectors.Semi_047	Pasternack semi-rigid 0.047" coaxial cable soldered to PCB pads
PCBConnectors.Semi_118	Pasternack semi-rigid 0.118" coaxial cable soldered to PCB pads

Version History

Introduced in R2017b

See Also

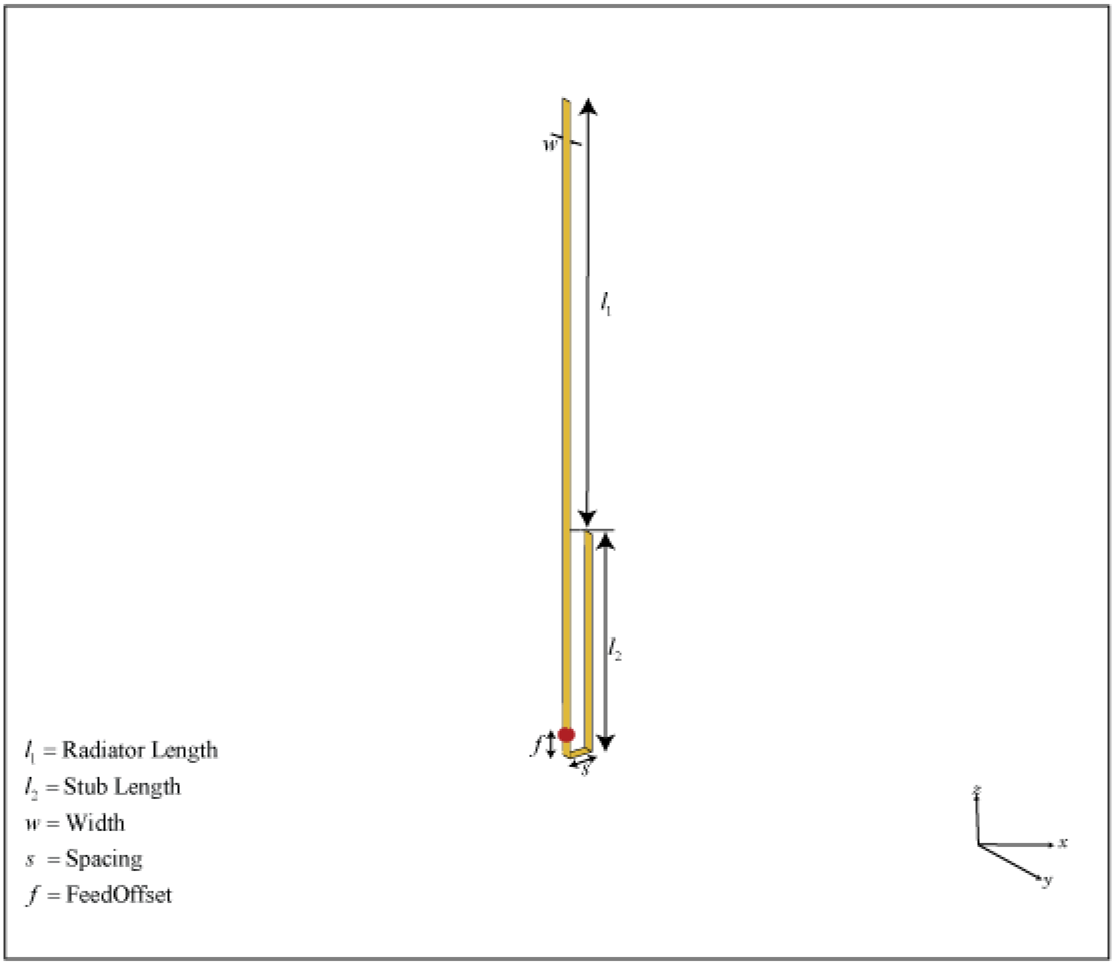
`PCBWriter` | `PCBServices` | `gerberWrite`

dipoleJ

Create J-dipole antenna

Description

Use the `dipoleJ` object to create a J-dipole on the *yz*- plane. The antenna contains a half-wavelength radiator and a quarter-wavelength stub. By default, the antenna dimensions are for an operating frequency of 144 MHz.



Creation

Syntax

```
jdipole = dipoleJ
jdipole = dipoleJ(Name,Value)
```

Description

`jdipole = dipoleJ` creates a J-dipole antenna for an operating frequency of 144 MHz.

`jdipole = dipoleJ(Name,Value)` creates a J-dipole antenna with additional properties specified by one or more name-value pair arguments. For example, `jdipole = dipoleJ('Width',0.2)` creates a J-dipole with a strip width of 0.2 m. Enclose each property name in quotes.

Properties

`RadiatorLength` — Radiator length

`0.9970` (default) | scalar

Radiator length, specified as a scalar in meters.

Example: `'RadiatorLength',0.9`

Example: `jdipole.RadiatorLength = 0.9`

Data Types: `double`

`StubLength` — Parallel line stub length

`0.4997` (default) | scalar

Parallel line stub length, specified as a scalar in meters.

Example: `'StubLength',0.3`

Example: `jdipole.StubLength = 0.3`

Data Types: `double`

`Width` — Strip width

`0.0200` (default) | scalar

Strip width, specified as a scalar in meters.

Example: `'StripWidth',0.0500`

Example: `jdipole.StripWidth = 0.0500`

Data Types: `double`

`Spacing` — Space between the stub and the radiator

`0.0460` (default) | scalar

Space between the parallel line stub and the radiator, specified as a scalar in meters.

Example: `'Spacing',0.0500`

Example: `jdipole.Spacing = 0.0500`

Data Types: `double`

`FeedOffset` — Signed distance to feed from base of stub on large arm

0.0490 (default) | scalar

Signed distance to the feed from the base of stub on the large arm, specified as a scalar in meters.

Example: `'FeedOffset',0.0345`

Example: `jdipole.FeedOffset = 0.0345`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where, `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `jdipole.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Default J-Dipole Antenna

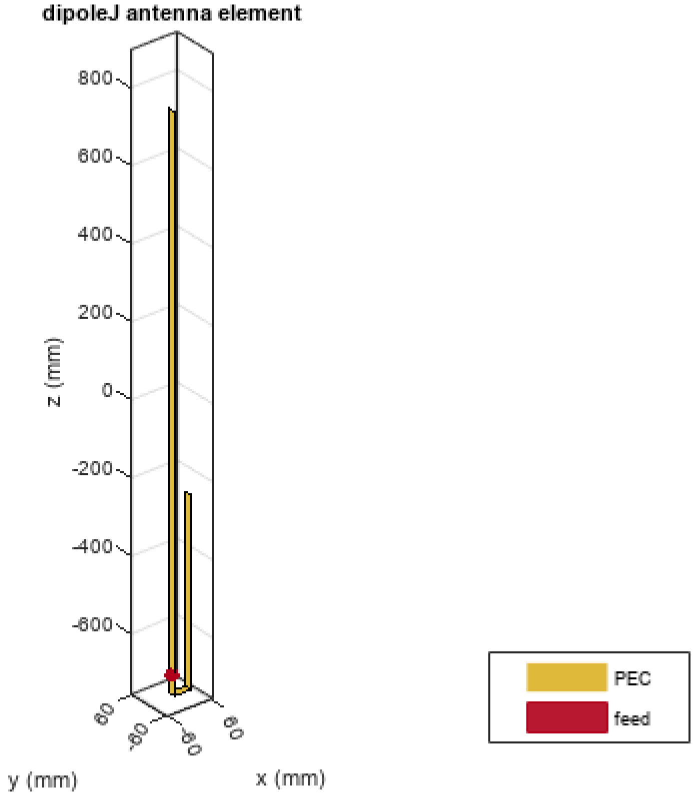
Create and view a default J-dipole antenna.

```
d = dipoleJ
```

```
d = 
  dipoleJ with properties:

    RadiatorLength: 0.9970
        StubLength: 0.4997
           Spacing: 0.0460
             Width: 0.0200
        FeedOffset: -0.6994
         Conductor: [1x1 metal]
              Tilt: 0
          TiltAxis: [1 0 0]
```

```
               Load: [1x1 lumpedElement]
```

```
show(d)
```



Impedance of J-Dipole Antenna

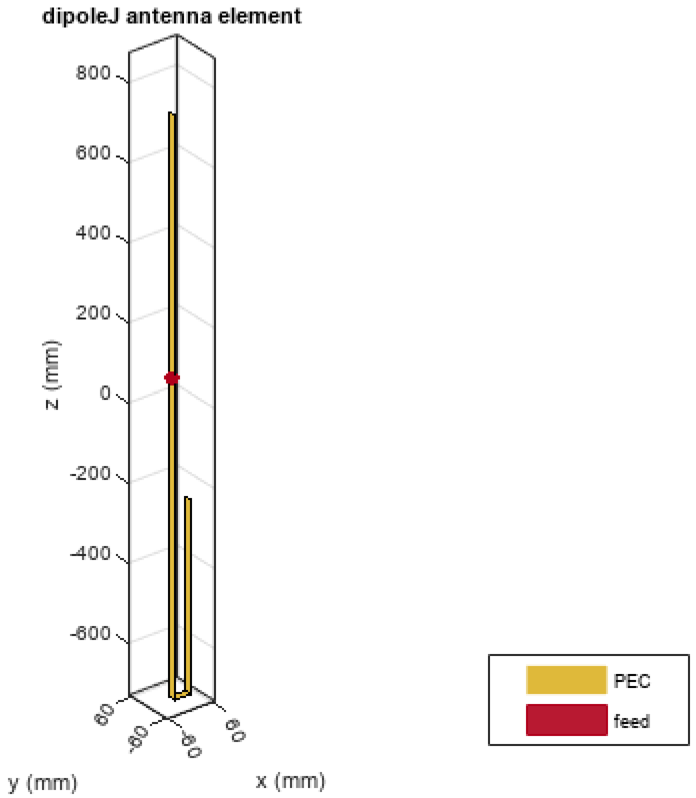
Create and view a J-dipole antenna with the following specifications:

Radiator length = 0.978 m

Stub length = 0.485 m

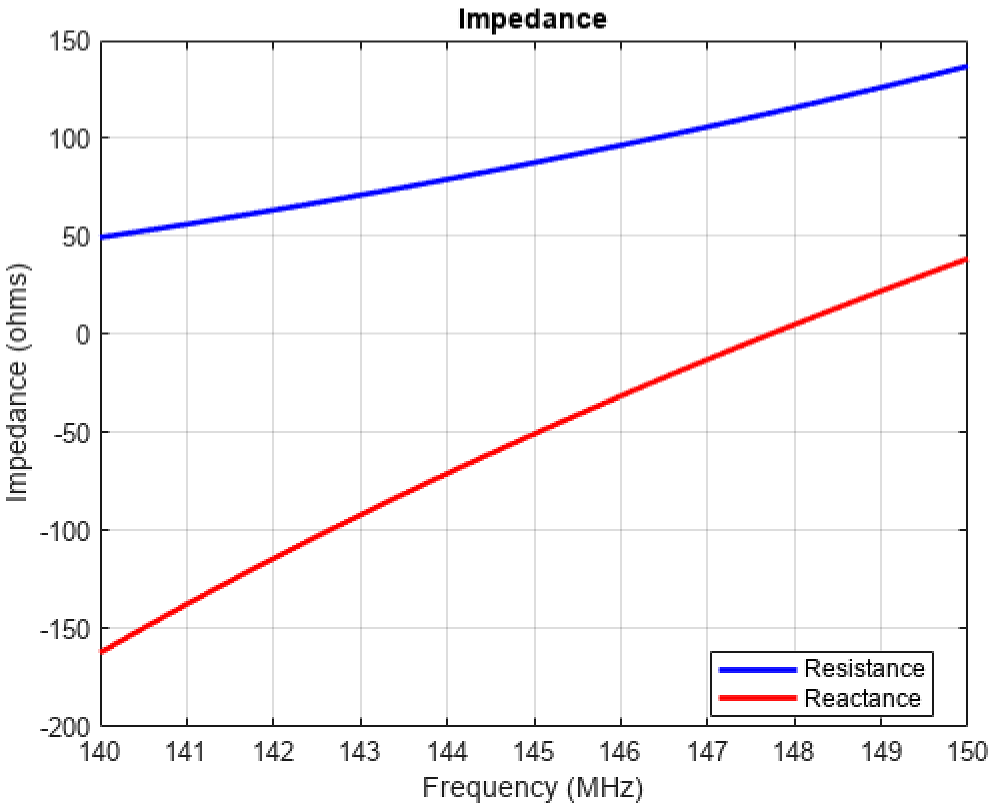
FeedOffset = 0.049 m

```
dj = dipoleJ('RadiatorLength',0.978,'StubLength',0.485, ...
      'FeedOffset',0.070);
show(dj)
```



Calculate the impedance of the antenna over a frequency span 140MHz - 150MHz.

```
impedance(dj,linspace(140e6,150e6,51));
```



Version History

Introduced in R2018a

See Also

`dipole` | `dipoleFolded` | `dipoleVee`

Topics

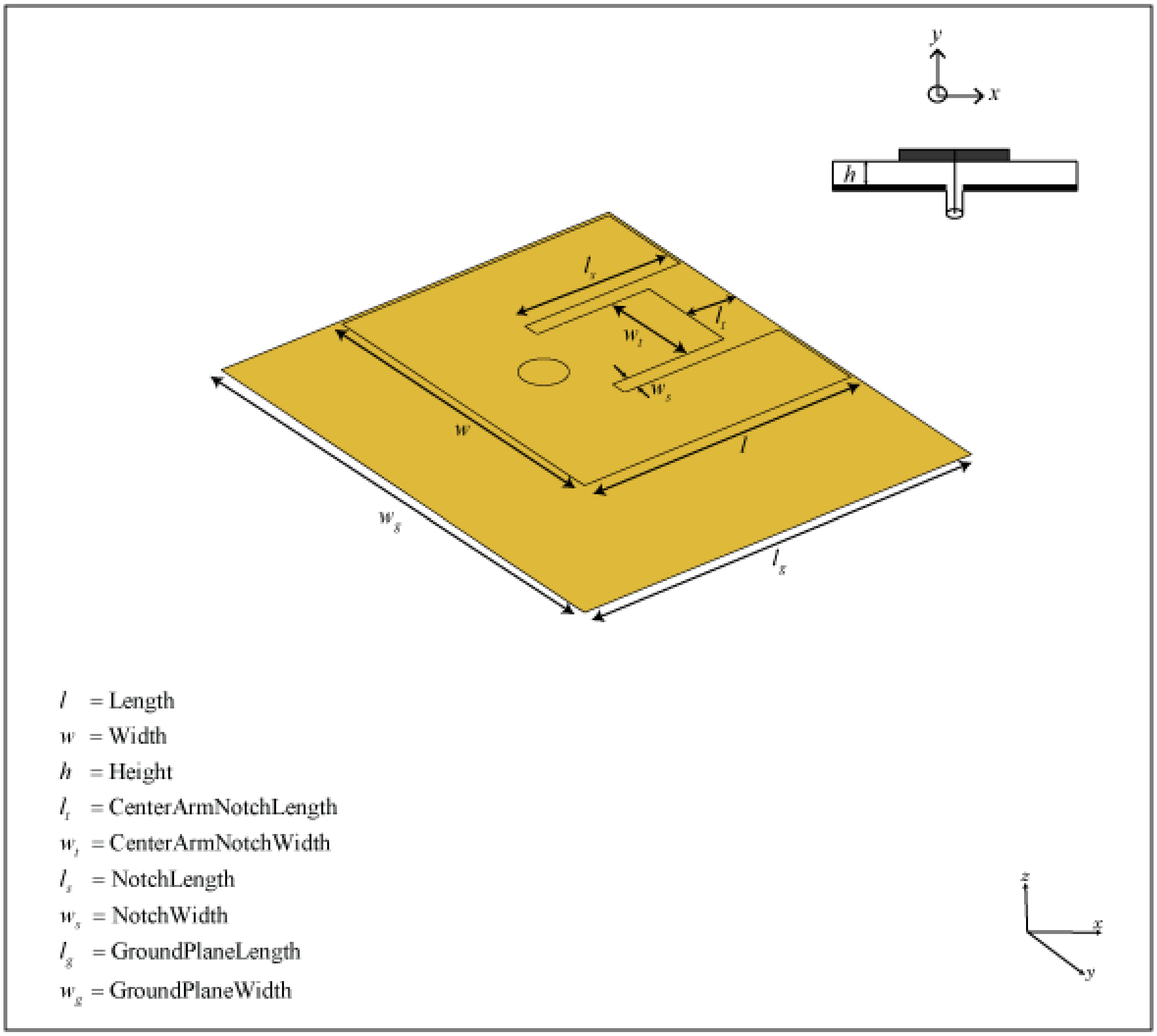
"Rotate Antennas and Arrays"

patchMicrostripEnotch

Create probe-fed E-shaped microstrip patch antenna

Description

Use the `patchMicrostripEnotch` object to create a probe-fed E-shaped microstrip patch antenna. The default patch is centered at the origin with the feedpoint along the length. By default, the dimensions are chosen for an operating frequency of 6.6 GHz for air or 5.5 GHz for Teflon.



Creation

Syntax

```
epatch = patchMicrostripEnotch
epatch = patchMicrostripEnotch(Name,Value)
```

Description

`epatch = patchMicrostripEnotch` creates an E-shaped microstrip patch antenna.

`epatch = patchMicrostripEnotch(Name,Value)` sets properties using one or more name-value pairs. For example, `epatch = patchMicrostripEnotch('Width',0.2)` creates a microstrip E-patch with a patch width of 0.2 m. Enclose each property name in quotes.

Properties

`Length` — Patch length along *x*-axis
`0.0172` (default) | scalar

Patch length along *x*-axis, specified as a scalar in meters.

Example: `'Length',0.9`

Example: `epatch.Length = 0.9`

Data Types: `double`

`Width` — Patch width along *y*-axis
`0.0200` (default) | scalar

Patch width along *y*-axis, specified as a scalar in meters.

Example: `'Width',0.0500`

Example: `epatch.Width = 0.0500`

Data Types: `double`

`Height` — Patch height above ground plane along *z*-axis
`0.0032` (default) | scalar

Patch height above ground plane along *z*-axis, specified as a scalar in meters.

Example: `'Height',0.00500`

Example: `epatch.Height = 0.00500`

Data Types: `double`

`CenterArmNotchLength` — Notch length on center E-arm along *x*-axis
`0.0028` (default) | scalar

Notch length on center E-arm along *x*-axis, specified as a scalar in meters.

Example: `'CenterArmNotchLength',0.100`

Example: `epatch.CenterArmNotchLength = 0.100`

Data Types: `double`

`CenterArmNotchWidth` — Notch width on center E-arm along *y*-axis

`0.0062` (default) | scalar

Notch width on center E-arm along *y*-axis, specified as a scalar in meters.

Example: `'CenterArmNotchWidth',0.0600`

Example: `epatch.CenterArmNotchWidth = 0.0600`

Data Types: `double`

`NotchLength` — Notch length along *x*-axis

`0.0100` (default) | scalar

Notch length along *x*-axis, specified as a scalar in meters.

Example: `'NotchLength',0.0200`

Example: `epatch.NotchLength = 0.0200`

Data Types: `double`

`NotchWidth` — Notch width along *y*-axis

`1.00003-03` (default) | scalar

Notch width along *y*-axis, specified as a scalar in meters.

Example: `'NotchWidth',0.00600`

Example: `epatch.NotchWidth = 0.00600`

Data Types: `double`

`GroundPlaneLength` — Ground plane length along *x*-axis

`0.0250` (default) | scalar

Ground plane length along *x*-axis, specified as a scalar in meters.

Example: `'GroundPlaneLength',120e-3`

Example: `epatch.GroundPlaneLength = 120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

`0.0300` (default) | scalar

Ground plane width along *y*-axis, specified as a scalar in meters.

Example: `'GroundPlaneWidth',120e-3`

Example: `epatch.GroundPlaneWidth = 120e-3`

Data Types: `double`

`PatchCenterOffset` — Signed distance of patch from origin

`[0 0]` (default) | two-element real-valued vector

Signed distance of patch from origin, specified as a two-element real-valued vector. Units are in meters. Use this property to adjust the location of the patch relative to the ground plane. Distances are measured along the length and width of the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Example: `epatch.PatchCenterOffset = [0.01 0.01]`

Data Types: `double`

`FeedOffset` — Signed distance of feed from origin

`[−0.0034 0]` (default) | two-element real-valued vector

Signed distance of feed from origin, specified as a two-element real-valued vector. Units are in meters. Use this property to adjust the location of the feedpoint relative to the ground plane and patch. Distances are measured along the length and width of the ground plane.

Example: `'FeedOffset',[0.01 0.01]`

Example: `epatch.FeedOffset = [0.01 0.01]`

Data Types: `double`

`FeedDiameter` — Feed diameter

`0.0013` (default) | scalar

Feed diameter, specified as a scalar in meters.

Example: `'FeedDiameter',0.0600`

Example: `epatch.FeedDiameter = 0.0600`

Data Types: `double`

`Substrate` — Type of dielectric material

`'Air'` (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. You place the patch over this dielectric substrate. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the groundplane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); epatch.Substrate = d`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `epatch.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

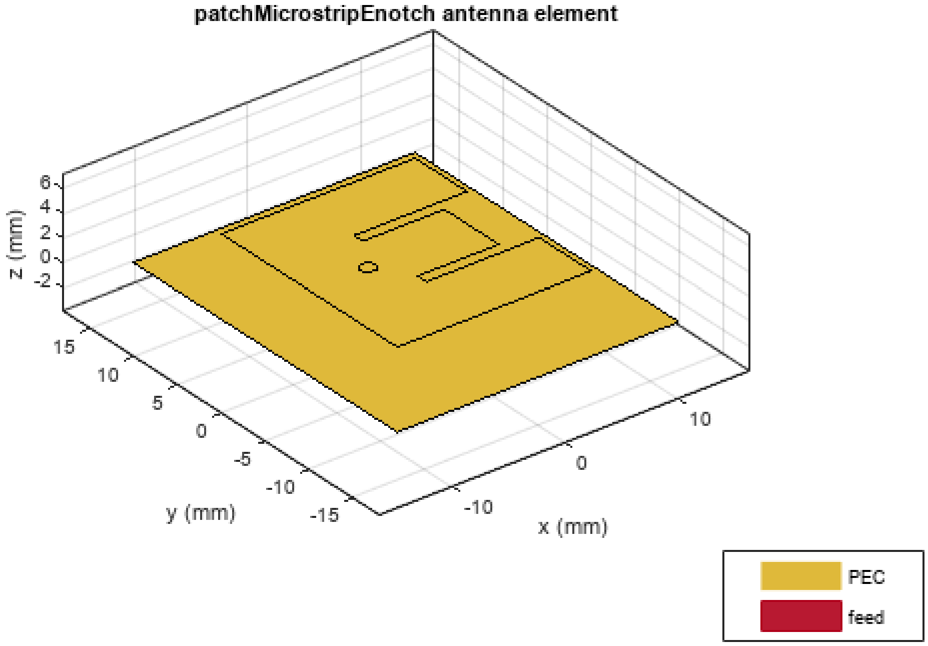
Default E-Shaped Patch Antenna

Create and view a default E-shaped patch antenna.

```
epatch = patchMicrostripEnotch
```

```
epatch = 
  patchMicrostripEnotch with properties:

                  Length: 0.0172
                   Width: 0.0200
             NotchLength: 0.0100
              NotchWidth: 1.0000e-03
    CenterArmNotchLength: 0.0028
     CenterArmNotchWidth: 0.0062
                  Height: 0.0032
               Substrate: [1x1 dielectric]
       GroundPlaneLength: 0.0250
        GroundPlaneWidth: 0.0300
       PatchCenterOffset: [0 0]
              FeedOffset: [-0.0034 0]
            FeedDiameter: 0.0013
               Conductor: [1x1 metal]
                    Tilt: 0
                TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
```

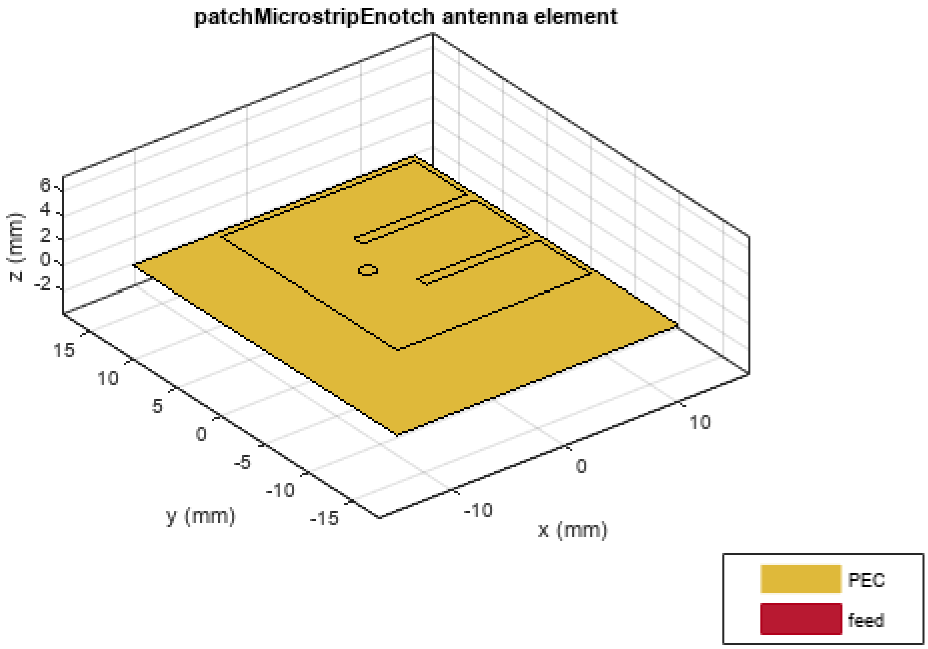
```
show(epatch)
```



E-Shaped Patch with No Slot Along Center E-Arm

Create and view an E-shaped patch with no slot on the center E-arm.

```
epatch = patchMicrostripEnotch('CenterArmNotchLength',0);
show(epatch);
```



Version History

Introduced in R2018a

See Also

`patchMicrostrip` | `patchMicrostripCircular` | `patchMicrostripTriangular`

Topics

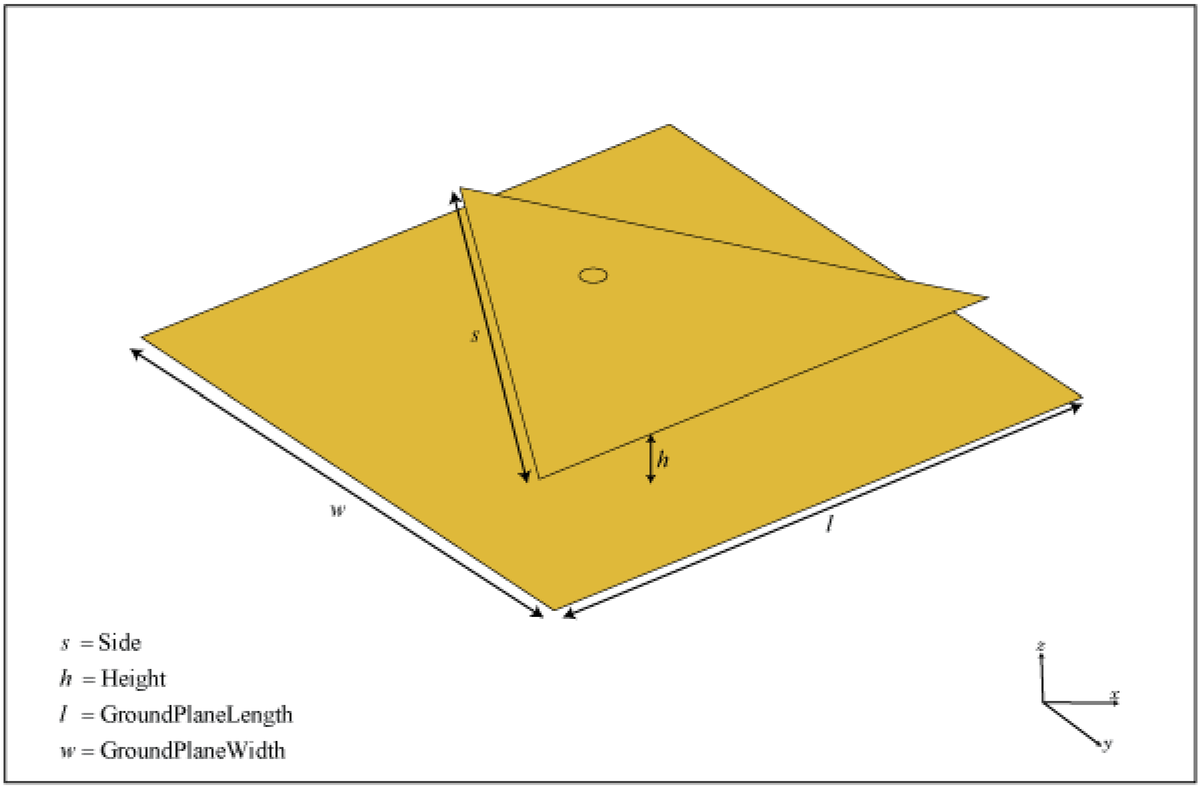
“Rotate Antennas and Arrays”

patchMicrostripTriangular

Create triangular microstrip patch antenna

Description

Use the `patchMicrostripTriangular` object to create a triangular microstrip patch antenna. The default patch is centered at the origin. By default, the dimensions are chosen for an operating frequency of 15 GHz. If you use a Teflon substrate, the default operating frequency is at 12.5 GHz.



Creation

Syntax

```
trianglepatch = patchMicrostripTriangular
trianglepatch = patchMicrostripTriangular(Name,Value)
```

Description

`trianglepatch = patchMicrostripTriangular` creates a triangular microstrip patch antenna.

`trianglepatch = patchMicrostripTriangular(Name,Value)` sets properties using one or more name-value pairs. For example, `trianglepatch =`

`patchMicrostripTriangular('Side',0.2)` creates a triangular microstrip patch with a side length of 0.2 m. Enclose each property name in quotes.

Properties

`Side` — Side lengths of triangular patch

`0.0102` (default) | scalar | two or three-element vector

Side lengths of triangular patch, specified as a scalar in meters or a two or three-element vector with each element unit in meters.

- Equilateral triangle - `Side` property value is a scalar. All three sides of the triangle are equal.
- Isosceles triangle - `Side` property value is a two-element vector. The first value specifies the base of the triangle along the *x*-axis. The second value specifies the other two sides of the triangle.
- Scalene triangle - `Side` property value is a three-element vector. The first value specifies the base of the triangle along the *x*-axis. The remaining two values specify the other two sides of the triangle.

Example: `'Side',0.2`

Example: `trianglepatch.Side = [0.2,0.3,0.4]` where the first value is the base of the scalene triangle along the x-axis.

Data Types: `double`

`Height` — Patch height above ground along *z*-axis

`0.0016` (default) | scalar

Patch height above ground along *z*-axis, specified as a scalar in meters.

Example: `'Height',0.2`

Example: `trianglepatch.Height = 0.002`

Data Types: `double`

`GroundPlaneLength` — Ground plane length along *x*-axis

`0.0120` (default) | scalar

Ground plane length along *x*-axis, specified as a scalar in meters.

Example: `'GroundPlaneLength',120e-3`

Example: `trianglepatch.GroundPlaneLength = 120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

`0.0120` (default) | scalar

Ground plane width along *y*-axis, specified as a scalar in meters.

Example: `'GroundPlaneWidth',120e-3`

Example: `trianglepatch.GroundPlaneWidth = 120e-3`

Data Types: `double`

`PatchCenterOffset` — Signed distance of patch from origin

`[0 0]` (default) | two-element real vector

Signed distance of patch from origin, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the patch relative to the ground plane. Distances are measured along the length and width of the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Example: `trianglepatch.PatchCenterOffset = [0.01 0.01]`

Data Types: `double`

`FeedOffset` — Signed distance of feed from origin

`[0 5.4173e-04]` (default) | two-element real vector

Signed distance of feed from origin, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the feedpoint relative to the ground plane and patch. Distances are measured along the length and width of the ground plane.

Example: `'FeedOffset',[0.01 0.01]`

Example: `trianglepatch.FeedOffset = [0.01 0.01]`

Data Types: `double`

`FeedDiameter` — Feed diameter

`2.5000e-04` (default) | scalar

Feed diameter, specified as a scalar in meters.

Example: `'FeedDiameter',0.0600`

Example: `trianglepatch.FeedDiameter = 0.0600`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Substrate` — Type of dielectric material

'Air' (default) | `dielectric` function

Type of dielectric material used as a substrate, specified as a dielectric material object. You can choose any material from the `DielectricCatalog` or use your own dielectric material. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be lesser than the ground plane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); ant.Substrate = d`

Load — Lumped elements
`[1x1 lumpedElement]` (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

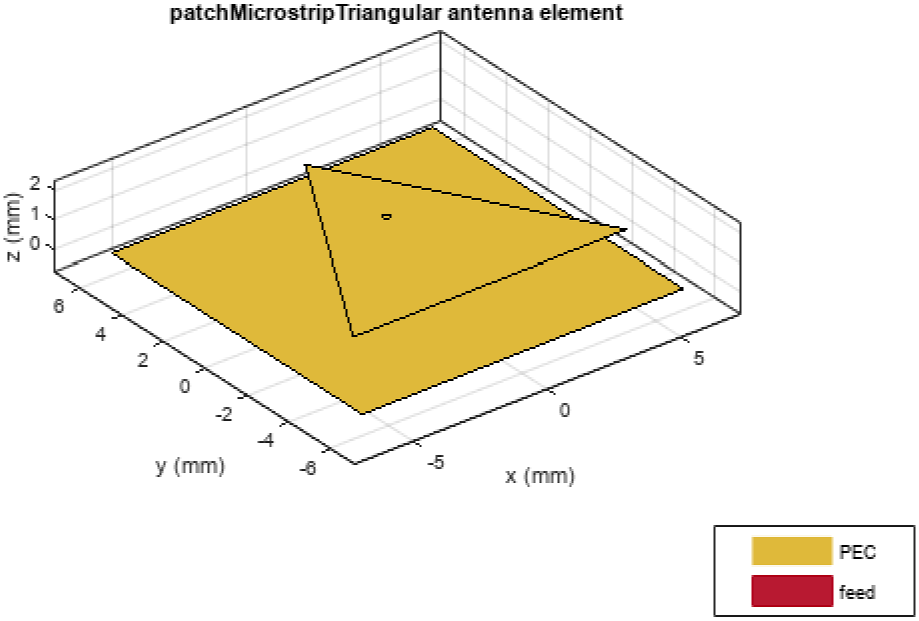
Default Triangular Microstrip Patch and Radiation Pattern

Create and view a default triangular microstrip patch.

```
p = patchMicrostripTriangular
```

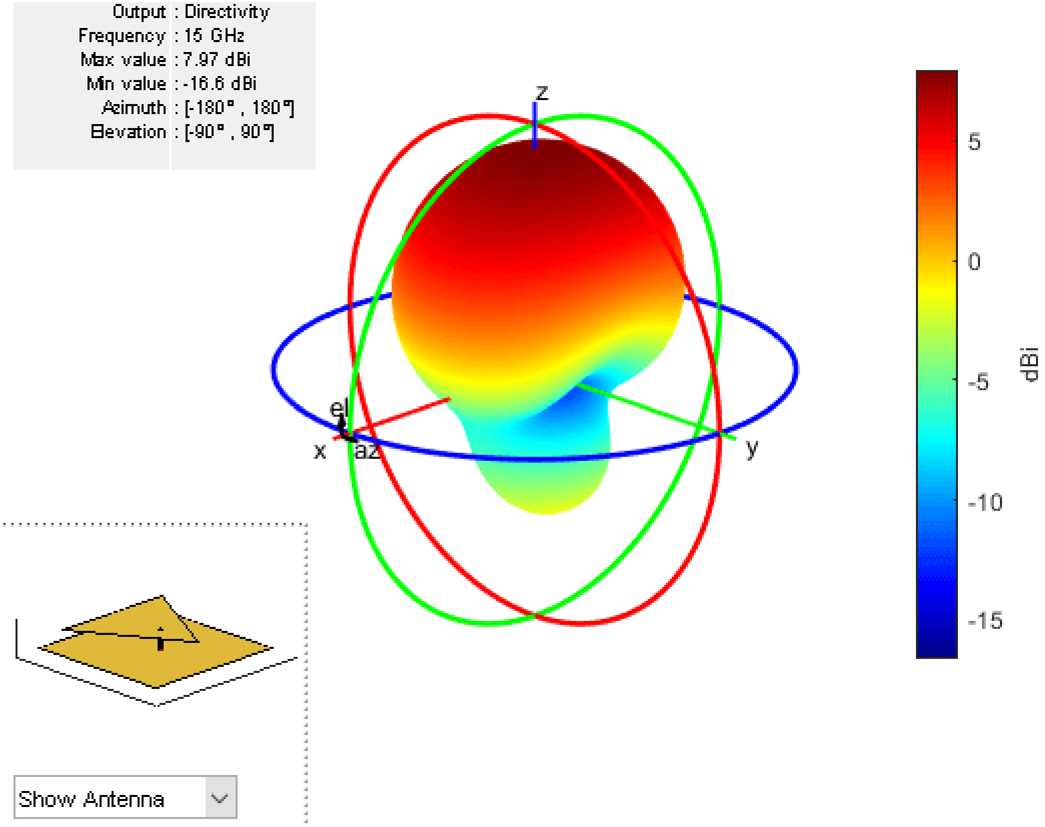
```
p = 
  patchMicrostripTriangular with properties:

                Side: 0.0102
              Height: 0.0016
           Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.0120
     GroundPlaneWidth: 0.0120
    PatchCenterOffset: [0 0]
          FeedOffset: [0 5.4173e-04]
        FeedDiameter: 2.5000e-04
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1x1 lumpedElement]
```

```
show(p)
```



Plot the radiation pattern at 15 GHz.

```
pattern(p,15e9)
```



Output : Directivity
Frequency : 15 GHz
Max value : 7.97 dBi
Min value : -16.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

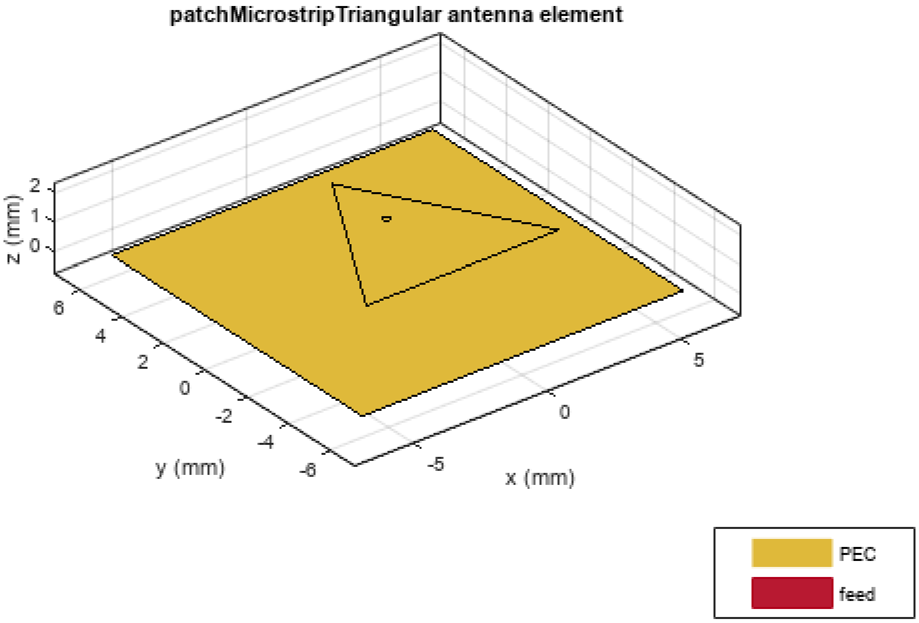
Different Types of Triangular Patch Antennas

Create different types of triangles to use in the patch.

Equilateral Triangle

Create an equilateral triangle patch of side 7.2 mm.

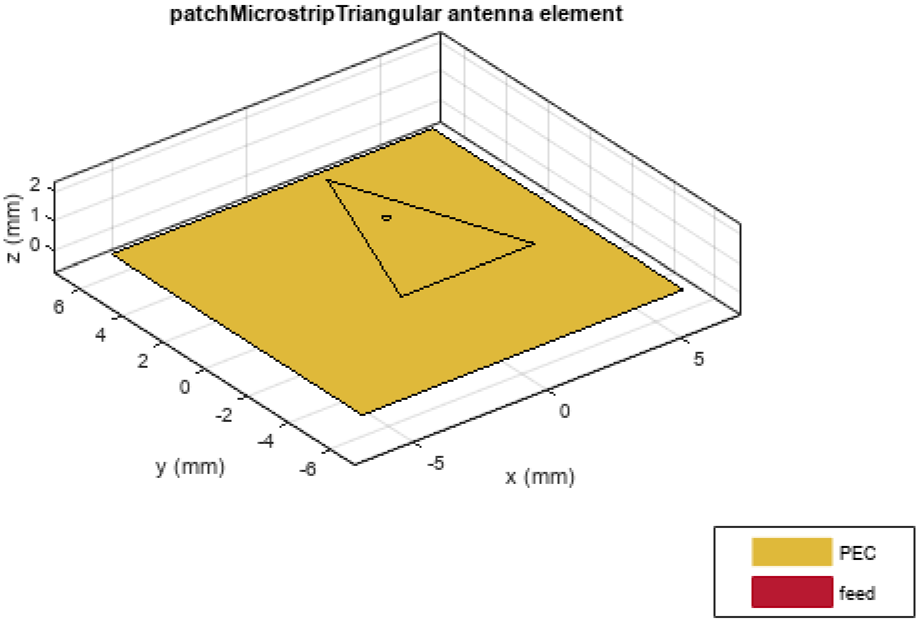
```
ant = patchMicrostripTriangular('Side',7.2e-3);
show(ant);
```



Isosceles Triangle

Create an isosceles triangular patch antenna with sides using the following dimensions: 5 mm and 7.2 mm.

```
ant =  patchMicrostripTriangular('Side',[5e-3,7.2e-3]);
show(ant);
```

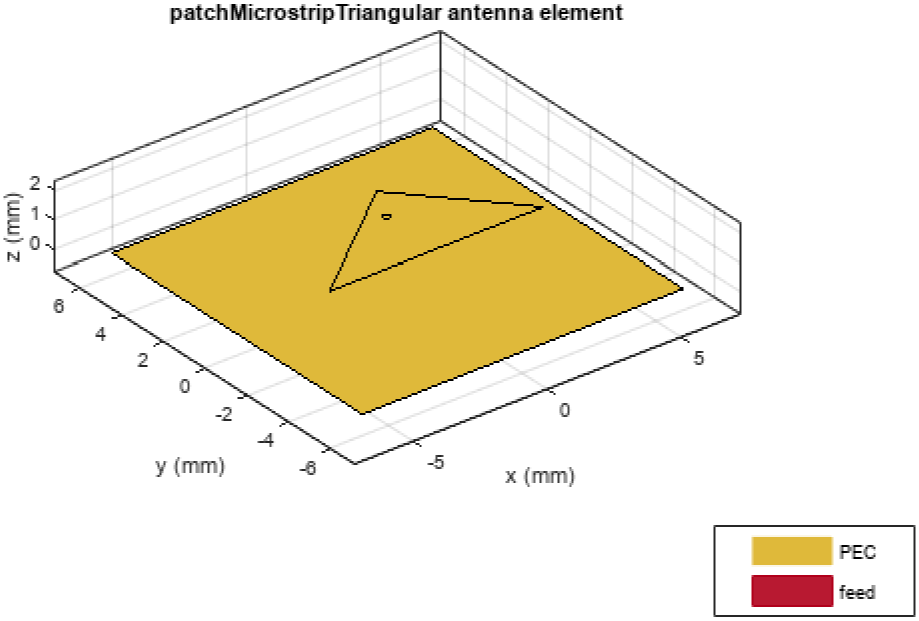


In the above figure, the first value of the side is chosen as the base of the triangle.

Scalene Triangle

Create a scalene triangular patch antenna with side using the following dimensions: 8 mm, 5 mm, and 4 mm.

```
ant = patchMicrostripTriangular('Side',[8e-3, 6e-3, 5e-3]);
show(ant);
```

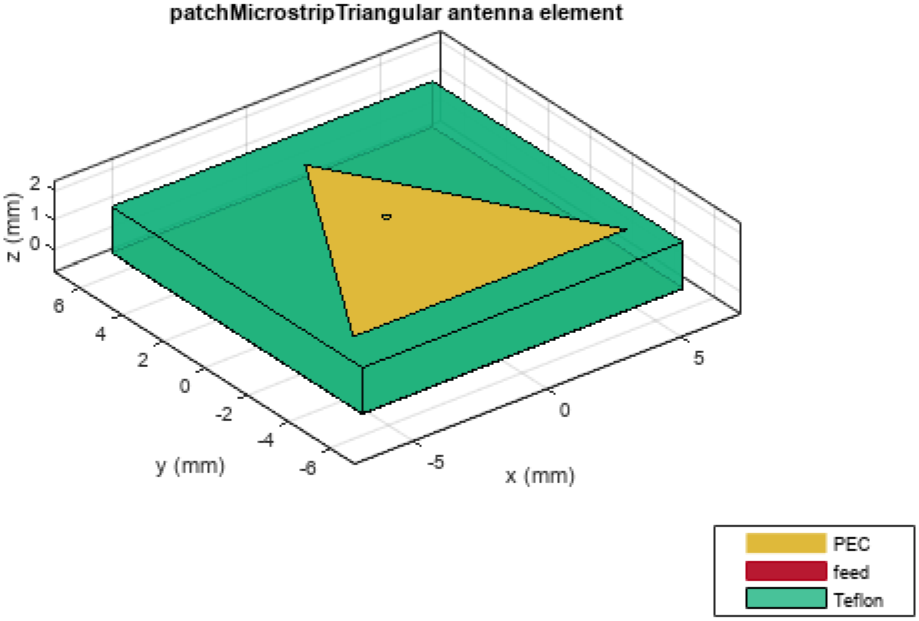


In the above figure, the first value of the side is chosen as the base of the triangle.

Triangle Patch Using Teflon Substrate and Radiation Pattern

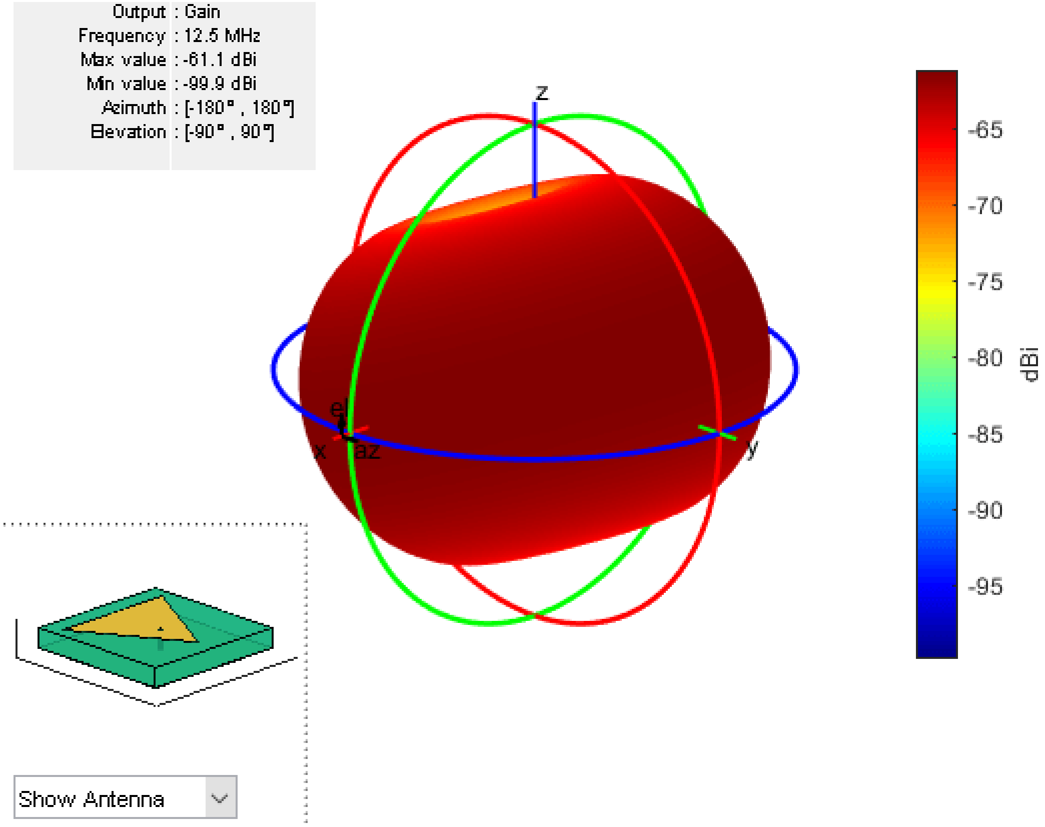
Create and view a triangular microstrip patch using Teflon substrate.

```
d = dielectric('Teflon');
p = patchMicrostripTriangular('Substrate',d);
show(p)
```



Plot the radiation pattern of the antenna.

```
pattern(p,12.5e6)
```



Output : Gain
Frequency : 12.5 MHz
Max value : -61.1 dBi
Min value : -99.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2018a

See Also

`patchMicrostrip` | `patchMicrostripCircular` | `patchMicrostripEnotch`

Topics

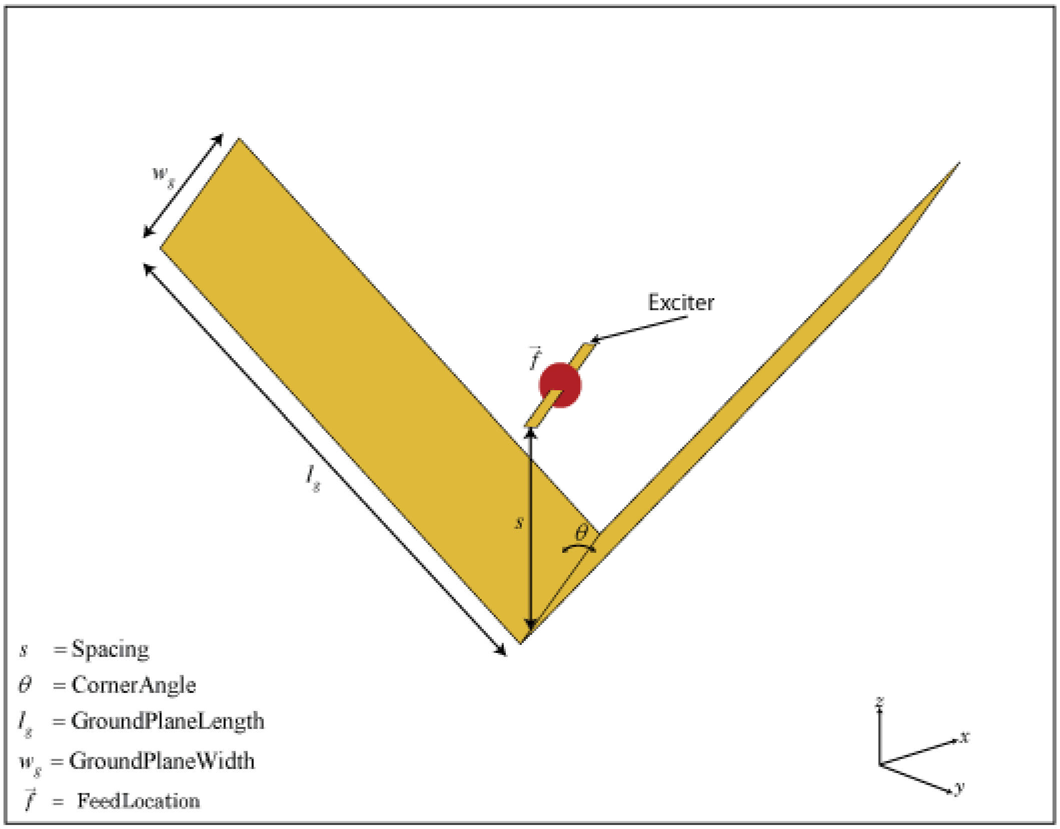
“ISM Band Patch Microstrip Antennas and Mutually Coupled Patches”
“Rotate Antennas and Arrays”

reflectorCorner

Create corner reflector-backed antenna

Description

Use the `reflectorCorner` object to create a corner reflector-backed antenna. By default, the exciter antenna is a dipole. The feedpoint of the dipole is at the origin. The default dimensions are for an operating frequency of 1 GHz.



Creation

Syntax

```
cornerreflector = reflectorCorner
cornerreflector = reflectorCorner(Name,Value)
```

Description

`cornerreflector = reflectorCorner` creates a corner reflector backed dipole antenna for an operating frequency of 1 GHz using default values.

`cornerreflector = reflectorCorner(Name,Value)` sets properties using one or more name-value pairs. For example, `cornerreflector = reflectorCorner('CornerAngle',45)` creates a corner reflector-backed antenna with a corner angle of 45 degrees. Enclose each property name in quotes.

Properties

`Exciter` — Antenna or array type used as exciter

`dipole` (default) | `antenna` object | `array` object

Antenna type used as an exciter, specified as any single-element antenna object. Except reflector and cavity antenna elements, you can use any of the antenna elements or array elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',horn`

Example: `ant.Exciter = horn`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Spacing` — Distance between exciter and reflector

`0.0750` (default) | scalar

Distance between exciter and reflector, specified as a scalar in meters.

Example: `'Spacing',0.0624`

Example: `cornerreflector.Spacing = 0.0624`

Data Types: `double`

`CornerAngle` — Angle made by corner reflector

`90` (default) | scalar

Angle made by corner reflector, specified as a scalar in degrees.

Example: `'CornerAngle',60`

Example: `cornerreflector.CornerAngle = 60`

Data Types: `double`

`GroundPlaneLength` — Reflector length along *x*-axis

`0.2000` (default) | scalar

Reflector length along the *x*-axis, specified as a scalar in meters. By default, ground plane length is measured along the *x*-axis. You can also set the `'GroundPlaneLength'` to zero.

Example: `'GroundPlaneLength',0.4000`

Example: `cornerreflector.GroundPlaneLength = 0.4000`

Data Types: `double`

`GroundPlaneWidth` — Reflector width along *y*-axis

`0.4000` (default) | scalar

Reflector width along the *y*-axis, specified as a scalar in meters. By default, ground plane width is measured along the *y*-axis. You can also set the `'GroundPlaneWidth'` to zero.

Example: 'GroundPlaneWidth',0.6000

Example: cornerreflector.GroundPlaneWidth = 0.6000

Data Types: double

Conductor — Type of metal material
'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Loads added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see lumpedElement.

Example: 'Load',lumpedelement, where, lumpedelement is the object for the load created using lumpedElement.

Example: cornerreflector.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna
0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna
[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Default Corner Reflector-Backed Antenna and Radiation Pattern

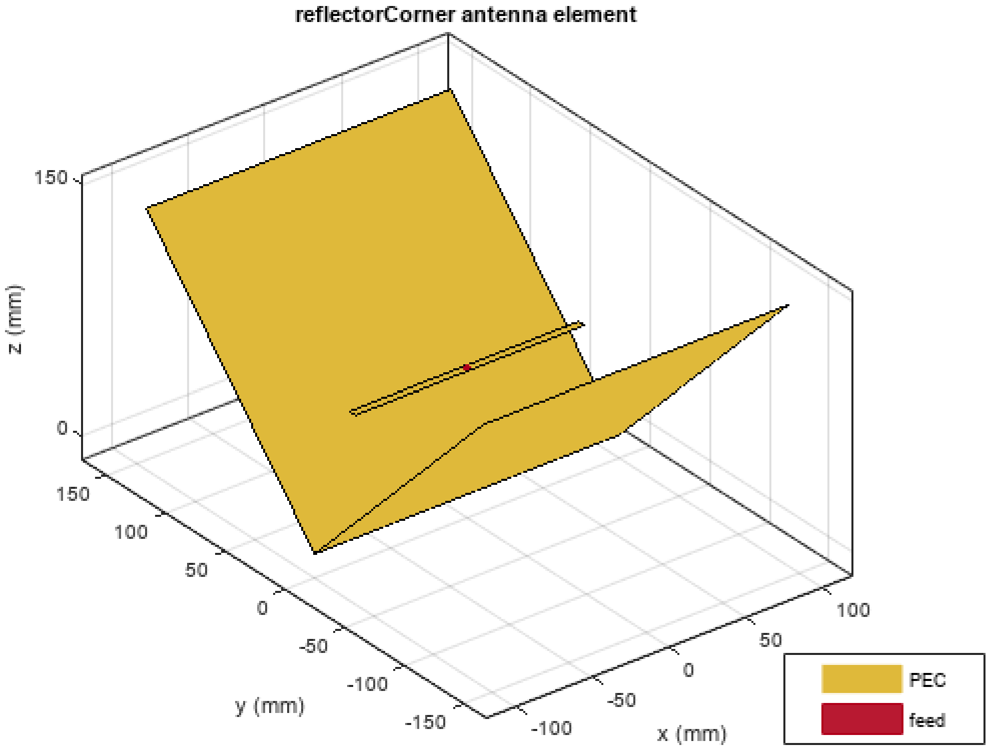
Create and view a corner reflector-backed dipole.

```
cornerreflector = reflectorCorner
```

```
cornerreflector = 
  reflectorCorner with properties:

              Exciter: [1x1 dipole]
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.4000
          CornerAngle: 90
              Spacing: 0.0750
            Conductor: [1x1 metal]
```

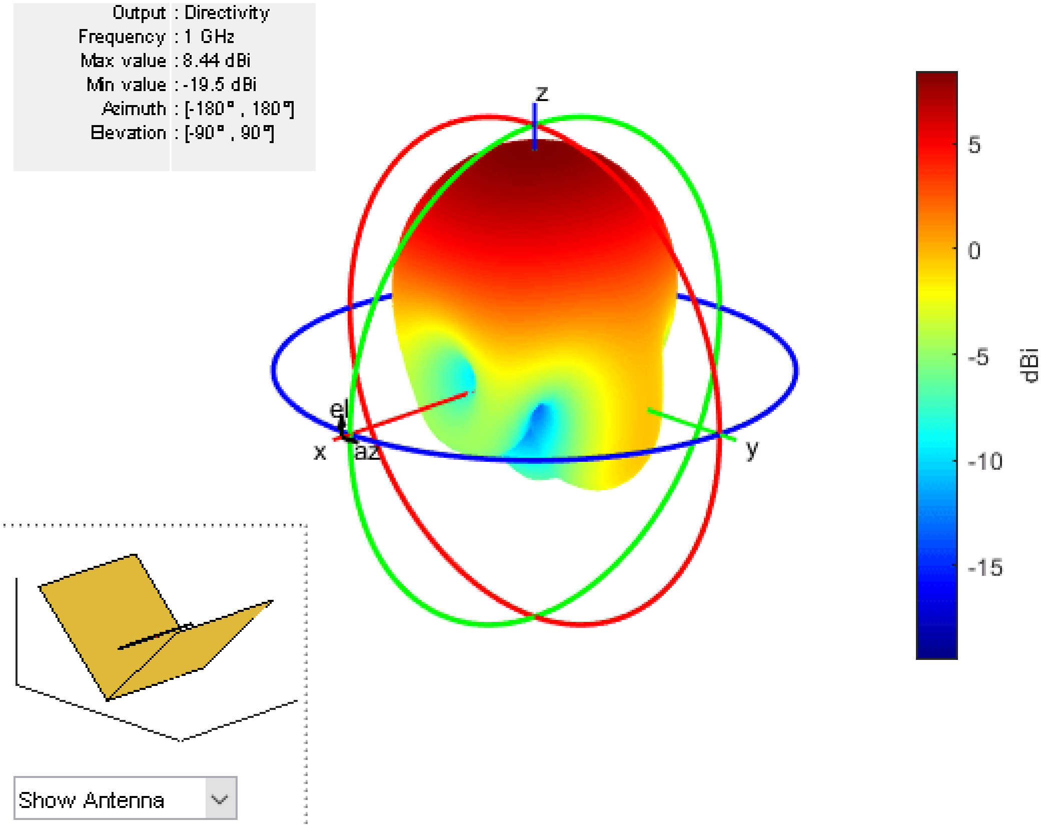
```
        Tilt: 0
    TiltAxis: [1 0 0]
        Load: [1x1 lumpedElement]
```

```
show(cornerreflector)
```



Plot the radiation pattern at 1 GHz.

```
pattern(cornerreflector,1e9)
```



Output : Directivity
Frequency : 1 GHz
Max value : 8.44 dBi
Min value : -19.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Corner Reflector-Backed Linear Array of Inverted-F Antennas

Create a linear array of inverted-F antennas.

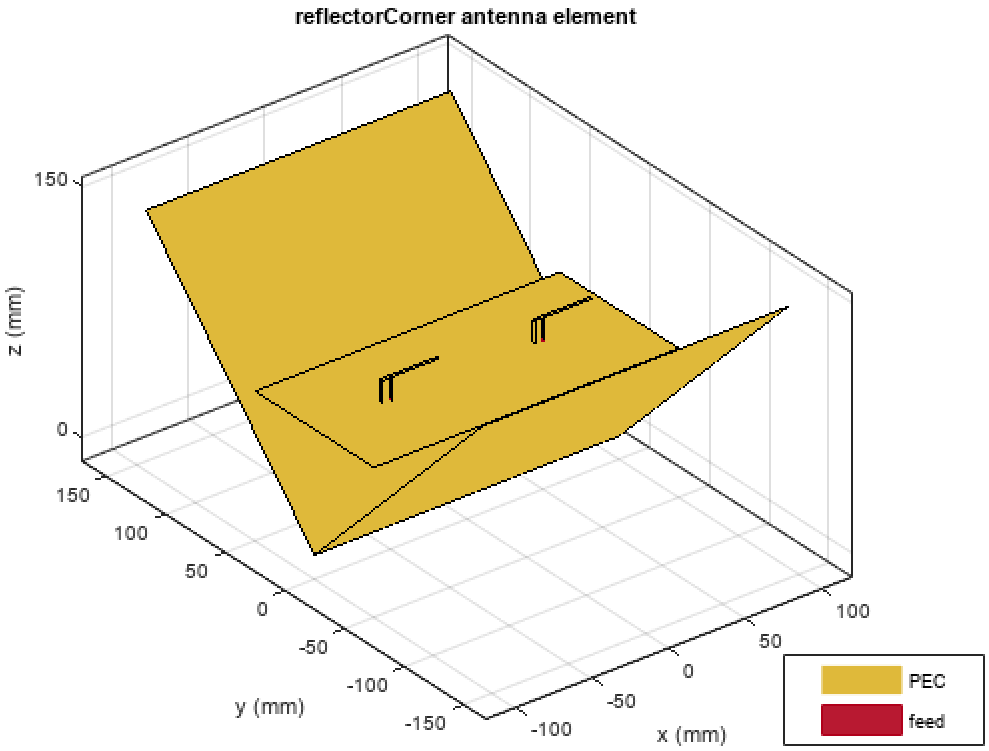
```
la = linearArray('Element',invertedF,'ElementSpacing',0.1);
```

Create a corner reflector-backed linear array of inverted-F antennas.

```
ant = reflectorCorner('Exciter',la)
```

```
ant = 
  reflectorCorner with properties:

              Exciter: [1x1 linearArray]
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.4000
          CornerAngle: 90
              Spacing: 0.0750
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



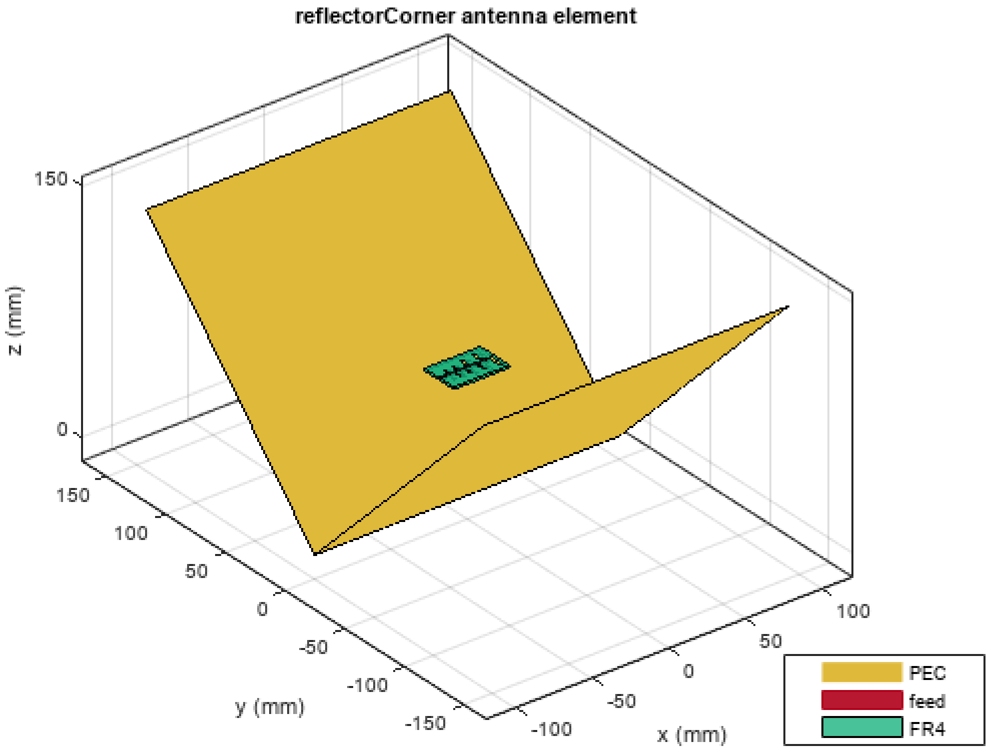
Create Log-Periodic Antenna with Corner Reflector Backing Structure

Create corner reflector-backed log-periodic antenna.

```
e = lpda;
ant = reflectorCorner('Exciter',e)
```

```
ant = 
  reflectorCorner with properties:

              Exciter: [1x1 lpda]
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.4000
          CornerAngle: 90
              Spacing: 0.0750
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2018a

See Also

`reflector` | `reflectorCircular`

Topics

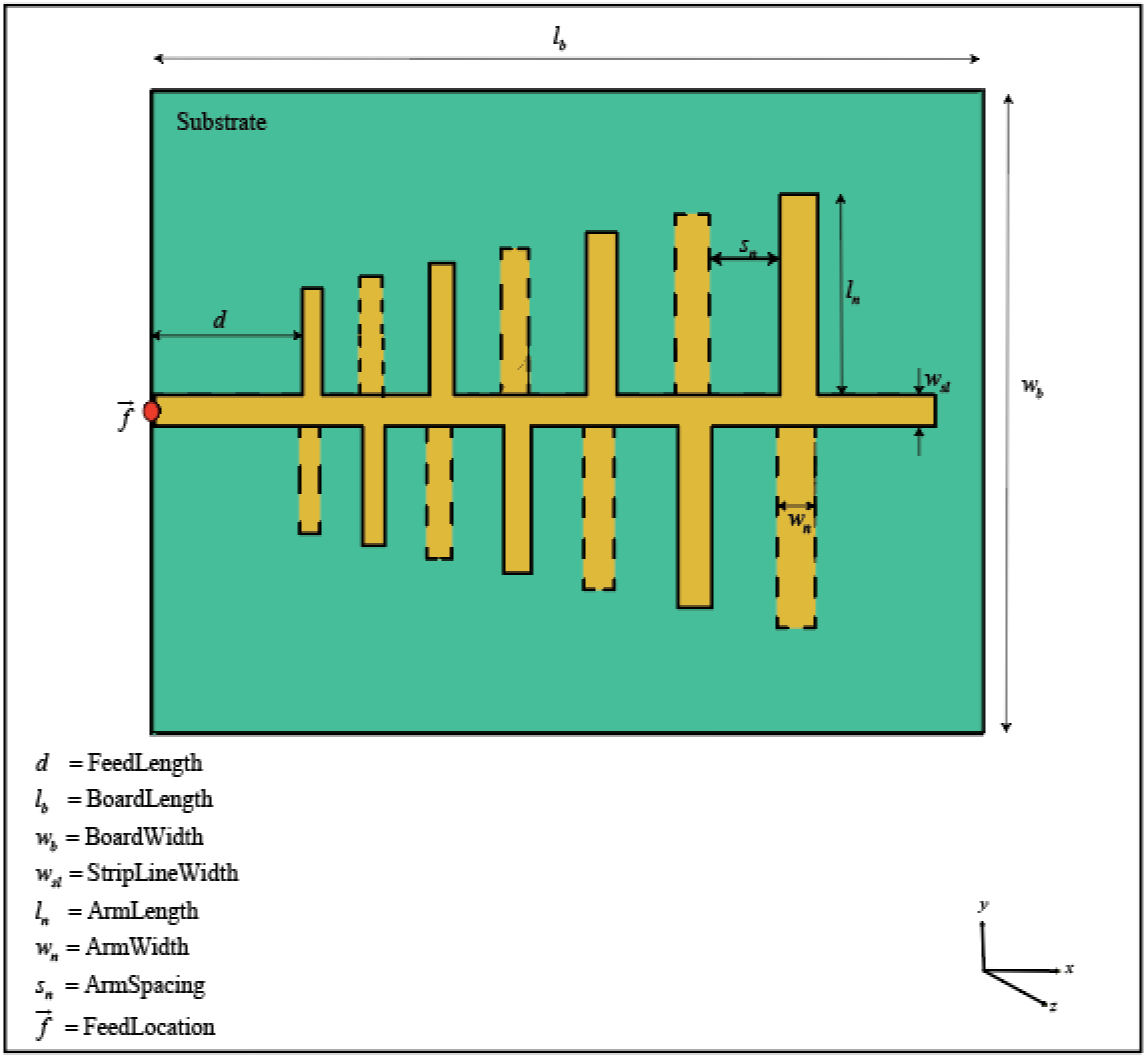
"Rotate Antennas and Arrays"

lpda

Create printed log-periodic dipole array antenna

Description

Use the `lpda` object to create a printed log-periodic dipole array antenna. The default antenna is centered at the origin and uses an FR4 substrate. This antenna is widely used in communication and radar due to advantages such as wideband, high gain, and high directivity.



Creation

Syntax

```
lpdipole = lpda
lpdipole = lpda(Name,Value)
```

Description

`lpdipole = lpda` creates a printed log-periodic dipole array antenna using default property values.

`lpdipole = lpda(Name,Value)` sets properties using one or more name-value pairs. For example, `lpdipole = lpda('BoardLength',0.2)` creates a printed log-periodic dipole array with a board length of 0.2 m.

Note Properties which are not specified retain their default values.

Properties

`BoardLength` — PCB length along *x*-axis
`0.0366` (default) | scalar

Printed circuit board (PCB) length along *x*-axis, specified as a scalar in meters.

Example: `'BoardLength',0.2`

Example: `lpdipole.BoardLength = 0.2`

Data Types: `double`

`BoardWidth` — PCB width along *y*-axis
`0.0244` (default) | two-element vector | scalar

PCB width along *y*-axis, specified in meters . Width of the PCB in meter. If the value is a scalar, a rectangular board is created and if the value is a vector with 2 elements, a trapezoidal board is created. The first element represents width of the board at the shortest end of the dipole and the second element represents width at the longest end of the dipole.

Example: `'BoardWidth',[0.06 0.06]`

Example: `lpdipole.BoardWidth = [10e-3 12e-3]`

Data Types: `double`

`Height` — PCB height along *z*-axis
`0.0016` (default) | scalar

PCB height along *z*-axis, specified as a scalar in meters.

Example: `'Height',0.0018`

Example: `lpdipole.Height = 0.0018`

Data Types: `double`

`StripLineWidth` — Parallel strip line width

`0.0012` (default) | scalar

Width of the parallel strip, specified as a scalar in meters.

Example: `'StripLineWidth',0.0014`

Example: `lpdipole.StripLineWidth = 0.0014`

Data Types: `double`

`FeedLength` — Distance from edge feed point to smallest dipole

`0.0065` (default) | scalar

The distance from the feed point to the smallest dipole , specified as a scalar in meters.

Example: `'FeedLength',0.0055`

Example: `lpdipole.FeedLength = 0.0055`

Data Types: `double`

`ArmLength` — Lengths of individual dipole arms

`[0.0040 0.0045 0.0050 0.0056 0.0062 0.0069 0.0076 0.0085]` (default) | vector

Lengths of individual dipole arms, specified as a vector with each element unit in meters.

Example: `'ArmLength',[0.0050 0.0055 0.0060 0.0066 0.0072 0.0079 0.0086 0.0095]`

Example: `lpdipole.ArmLength = [0.0050 0.0055 0.0060 0.0066 0.0072 0.0079 0.0086 0.0095]`

Data Types: `double`

`ArmWidth` — Widths of individual dipole arms

`[8.8000e-04 9.8000e-04 0.0011 0.0012 0.0013 0.0015 0.0017 0.0019]` (default) | vector

Widths of individual dipole arms, specified as a vector with each element unit in meters.

Example: `'ArmWidth',[9.8000e-04 10.8000e-04 0.0021 0.0022 0.0023 0.0025 0.0027 0.0029]`

Example: `lpdipole.ArmWidth = [9.8000e-04 10.8000e-04 0.0021 0.0022 0.0023 0.0025 0.0027 0.0029]`

Data Types: `double`

`ArmSpacing` — Spacing between individual dipole arms

`[0.0027 0.0030 0.0033 0.0037 0.0041 0.0046 0.0051]` (default) | vector

Spacing between individual dipole arms, specified as a vector with each element unit in meters.

Example: `'ArmSpacing',[0.0037 0.0040 0.0043 0.0047 0.0051 0.0056 0.0061]`

Example: `lpdipole.ArmSpacing = [0.0037 0.0040 0.0043 0.0047 0.0051 0.0056 0.0061]`

Data Types: `double`

`Substrate` — Type of dielectric material

`'FR4'` (default) | dielectric object

Type of dielectric material used as a substrate, specified as an dielectric object. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be equal to the groundplane dimensions.

Example: `d = dielectric('Teflon'); 'Substrate',d`

Example: `d = dielectric('Teflon'); lpdipole.Substrate = d`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `lpda.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.

- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Default Printed Log-Periodic Antenna

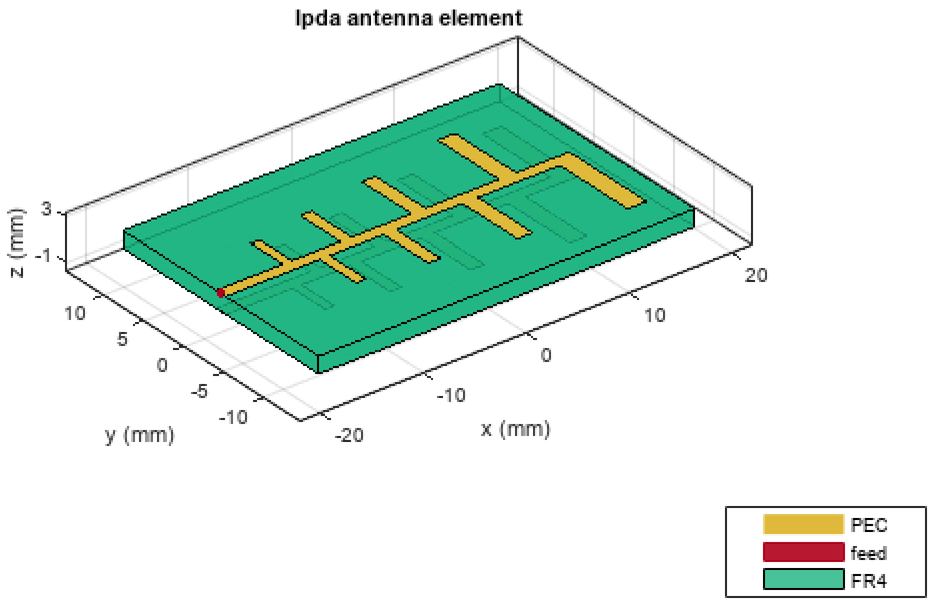
Create and view a printed log-periodic dipole array antenna.

```
lpdipole = lpda
```

```
lpdipole = 
  lpda with properties:

       BoardLength: 0.0366
        BoardWidth: 0.0244
```

```
            Height: 0.0016
    StripLineWidth: 0.0012
        FeedLength: 0.0065
         ArmLength: [0.0040 0.0045 0.0050 0.0056 0.0062 0.0069 0.0076 0.0085]
          ArmWidth: [8.8000e-04 9.8000e-04 0.0011 0.0012 0.0013 0.0015 ... ]
        ArmSpacing: [0.0027 0.0030 0.0033 0.0037 0.0041 0.0046 0.0051]
         Substrate: [1x1 dielectric]
         Conductor: [1x1 metal]
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1x1 lumpedElement]
```

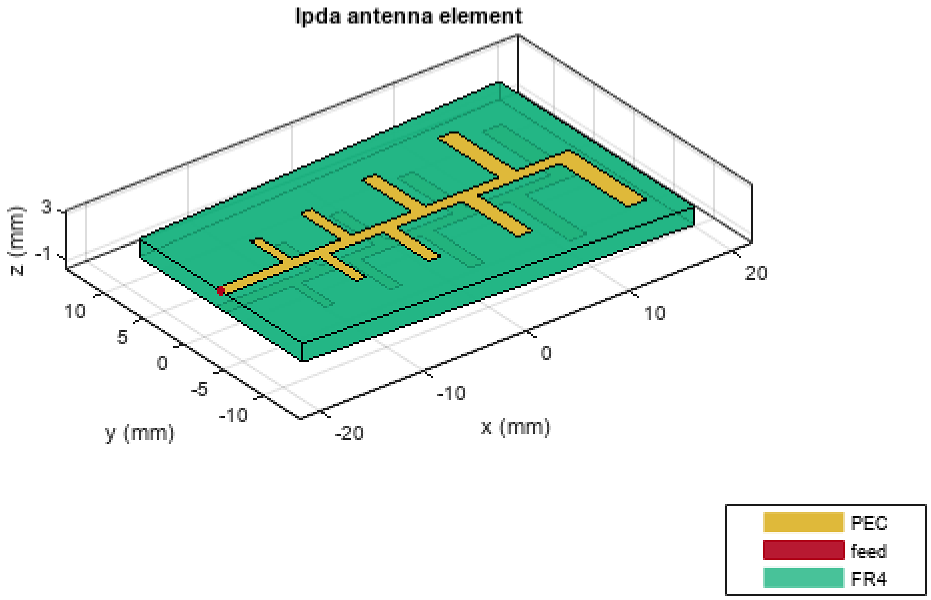
```
show(lpdipole)
```



Create and View Characteristics of Tapered LPDA

Create a tapered LPDA object and plot impedance over a frequency of 5 - 8GHz. This example also shows how to plot the 3-D radiation pattern of the antenna.

```
lpdipole = lpda('BoardWidth',[20.37e-3 24.37e-3]);
show(lpdipole)
```

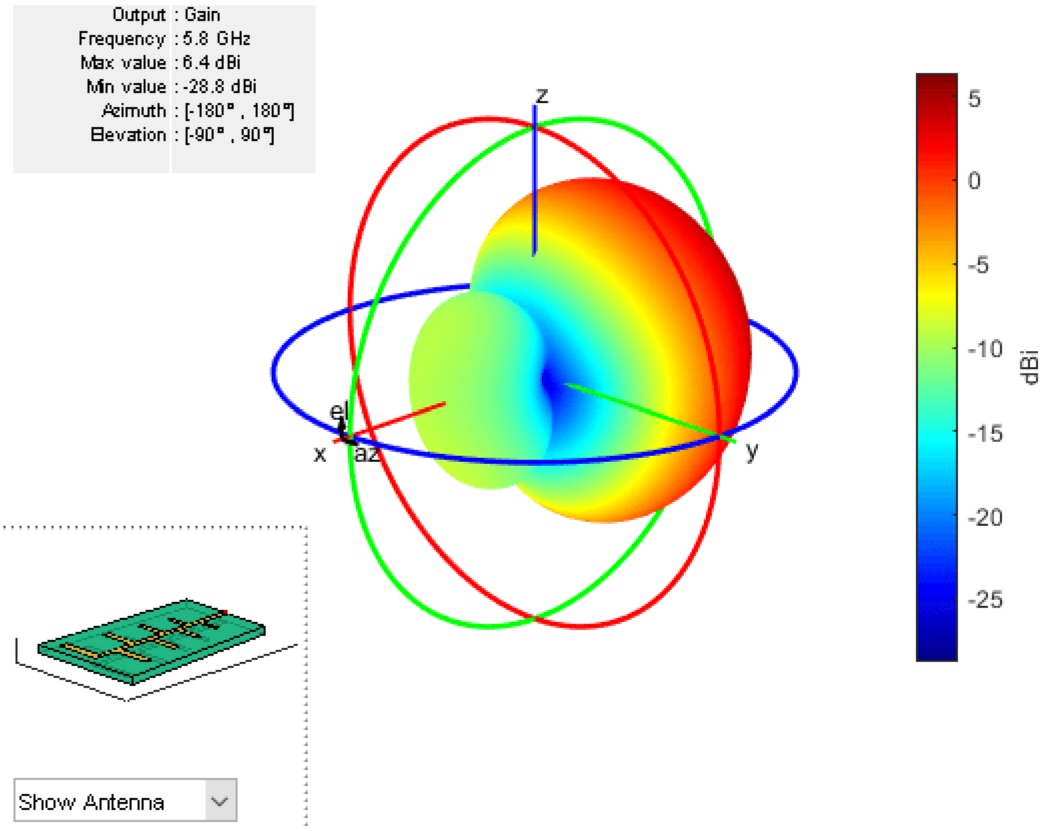


Plot Impedance over the specified frequency range.

```
freq = linspace(5e9, 8e9, 41);
figure;
impedance(lpdipole,freq)
```

Plot the 3-D radiation pattern at 5.8 GHz.

```
pattern(lpdipole,5.8e9)
```



Output : Gain
Frequency : 5.8 GHz
Max value : 6.4 dBi
Min value : -28.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2018a

See Also

`yagiUda` | `pcbStack`

Topics

"Rotate Antennas and Arrays"

helixMultifilar

Creates bifilar or quadrafilar helix or conical helix antenna on circular ground plane

Description

The `helixMultifilar` object creates a bifilar or quadrafilar helix or conical helix antenna on a circular ground plane. You can create both short-circuited and open-ended helix multifilar antennas. Bifilar and quadrafilar helix antennas are used in aerospace and defense applications.

The width of the strip is related to the diameter of an equivalent cylinder by the equation

$$w = 2d = 4r$$

where:

- w is the width of the strip.
- d is the diameter of an equivalent cylinder.
- r is the radius of an equivalent cylinder.

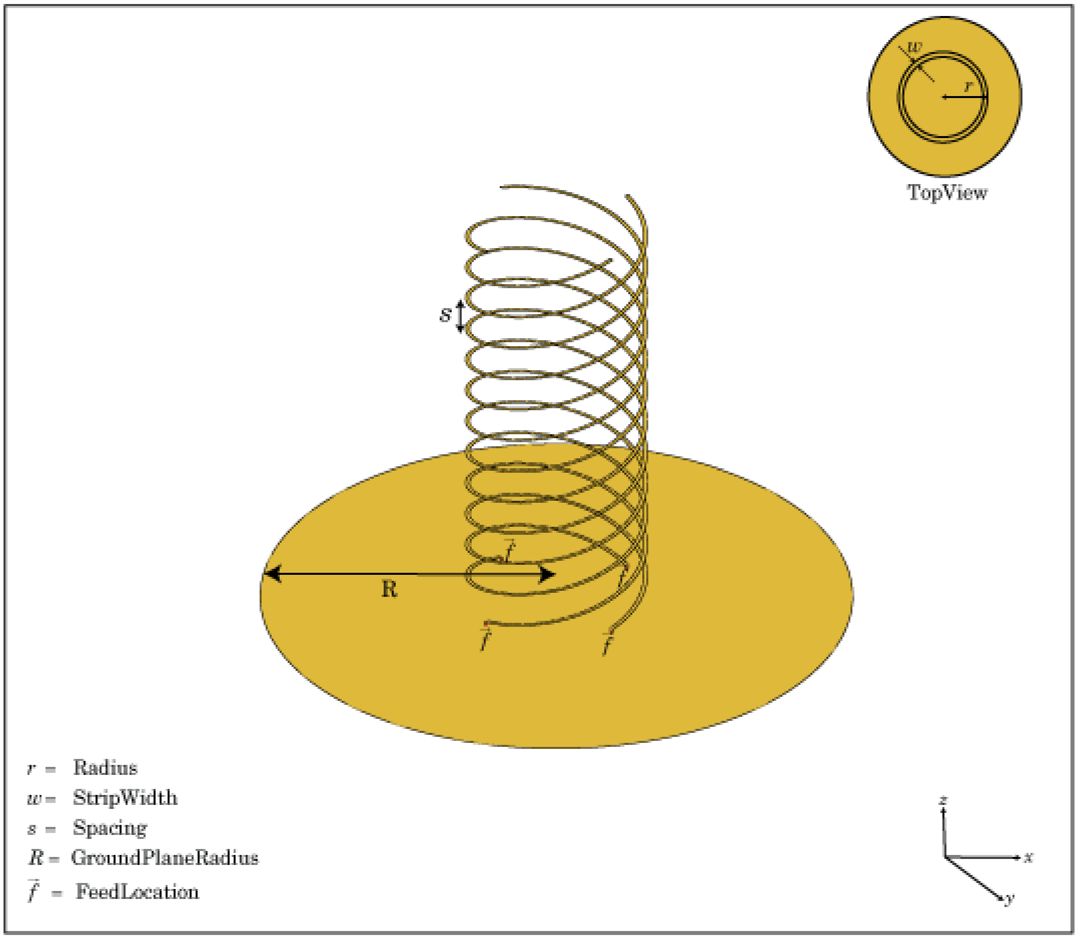
For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default helix antenna is end-fed. The circular ground plane is on the *xy*- plane. Helix antennas are commonly used in axial mode. In this mode, the helix circumference is comparable to the operating wavelength, and the helix has maximum directivity along its axis. In normal mode, the helix radius is small compared to the operating wavelength. In this mode, the helix radiates broadside, that is, in the plane perpendicular to its axis. The basic equations for the helix are

$$x = r\cos(\theta)$$
$$y = r\sin(\theta)$$
$$z = S\theta$$

where:

- r is the radius of the helical dipole.
- θ is the winding angle.
- S is the spacing between turns.

For a given pitch angle in degrees, use the `helixpitch2spacing` utility function to calculate the spacing between the turns in meters.



Creation

Syntax

```
ant = helixMultifilar
ant = helixMultifilar(Name,Value)
```

Description

`ant = helixMultifilar` creates a bifilar or quadrafilar helix or conical helix antenna operating in the axial mode. The default multifilar helical antenna is end-fed and has a circular ground plane on the *xy*- plane. The default operating frequency is around 2 GHz.

`ant = helixMultifilar(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = helixMultifilar('Radius',28e-03)` creates a multifilar helix with turns of radius 28e-03 m.

Properties

NumArms — Number of helical elements

4 (default) | 2

Number of helical elements, specified as 4 or 2. Specify two elements to create a bifilar helix antenna, and four elements to create a quadrafilar helix antenna.

Example: 'NumArms',2

Example: ant.NumArms = 2

Data Types: double

Radius — Radius of turns

0.0220 (default) | positive scalar integer | two-element vector

Radius of the turns, specified as a positive scalar integer in meters or a two element vector with each element unit in meters. In the two-element vector, the first element specifies the bottom radius and the second element specifies the top radius of the conical helix antenna.

Example: 'Radius',28e-03

Example: ant.Radius = 28e-03

Data Types: double

Width — Width of strip

1000e-03 (default) | positive scalar integer

Width of the strip, specified as a positive scalar integer in meters.

Example: 'Width',0.2

Example: ant.Width = 0.2

Data Types: double

Turns — Number of turns

3 (default) | scalar integer

Number of turns, specified as a scalar integer.

Example: 'Turns',4

Example: ant.Turns = 4

Data Types: double

Spacing — Spacing between turns

0.0350 (default) | positive scalar integer

Spacing between the turns, specified as a positive scalar integer in meters.

Example: 'Spacing',7.5e-2

Example: ant.Spacing = 7.5e-2

Data Types: double

ShortEnds — Status of helix ends

0 (default) | 1

Status of helix ends, specified as `0` or `1`. By default, the `helixMultifilar` is an open circuit. Setting the property to `1` makes the helix antenna short circuit.

Example: `'ShortEnds',1`

Example: `ant.ShortEnds = 1`

Data Types: `double`

`WindingDirection` — Direction of helix turns (windings)

`'CW'` | `'CCW'`

Direction of the helix turns (windings), specified as `'CW'` for clockwise or `'CCW'` for counter-clockwise.

Example: `'WindingDirection','CW'`

Example: `ant.WindingDirection = 'CW'`

Data Types: `char` | `string`

`FeedStubHeight` — Height of feeding stub from ground plane

`1.000e-03` (default) | positive scalar integer

Height of the feeding stub from the ground plane, specified as a positive scalar integer in meters.

Example: `'FeedStubHeight',7.5e-2`

Example: `ant.FeedStubHeight = 7.5e-2`

Data Types: `double`

`GroundPlaneRadius` — Ground plane radius

`0.0750` (default) | positive scalar integer

Ground plane radius, specified as a positive scalar integer in meters. By default, the ground plane is on the *xy*- plane and is symmetrical about the origin.

Setting this value to `Inf` uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneRadius',2.05`

Example: `ant.GroundPlaneRadius = 7.5e-2`

Data Types: `double`

`FeedVoltage` — Excitation voltage applied to individual antenna feeds

1 (default) | scalar integer | vector integers

Excitation voltage applied to individual antenna feeds, specified as a scalar integer or vector integers. A scalar value applies the same voltage to all feeds.

Example: `'FeedVoltage',[1 2]`

Example: `ant.FeedVoltage = [1 2]`

Data Types: `double`

`FeedPhase` — Excitation voltage phase applied to individual antenna feeds

0 (default) | scalar integer | vector integers

Excitation voltage phase applied to individual antenna feeds, specified as a scalar integer or vector integers. A scalar value applies the same voltage phase to all feeds.

Example: `'FeedPhase',[0 45]`

Example: `ant.FeedPhase = [0 45]`

Data Types: `double`

`Substrate` — Type of dielectric material

`'Air'` (default) | `dielectric` object

Type of dielectric material used as the substrate, specified as a dielectric object. You can specify only one dielectric layer in the `helixMultifilar` object. When using the `Substrate` property, specify the same radius for all the turns. When using a dielectric material other than air, the number of turns in the helix should be greater than 1. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Example: `d = dielectric('FR4'); ant = helixMultifilar('Substrate',d)`

Example: `d = dielectric('FR4'); ant.Substrate = d;`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

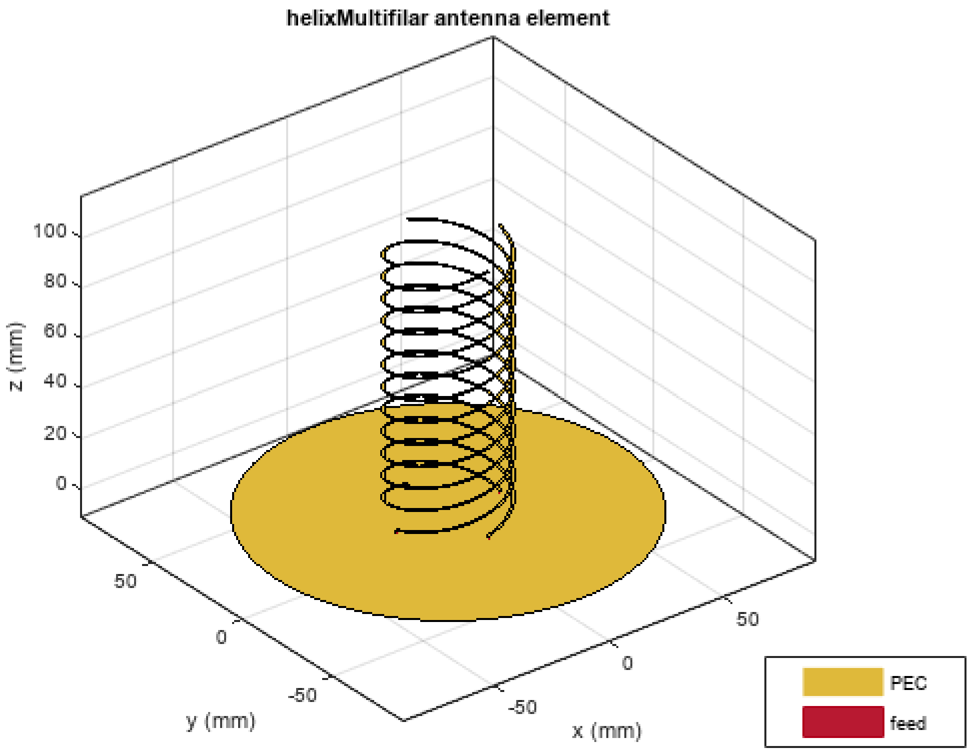
Quadrafilar Helix

Create and view a Quadrafilar helix antenna.

```
ant = helixMultifilar
```

```
ant = 
  helixMultifilar with properties:

              NumArms: 4
               Radius: 0.0220
                Width: 1.0000e-03
                Turns: 3
              Spacing: 0.0350
            ShortEnds: 0
     WindingDirection: 'CCW'
       FeedStubHeight: 1.0000e-03
     GroundPlaneRadius: 0.0750
          FeedVoltage: 1
            FeedPhase: 0
            Substrate: [1x1 dielectric]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



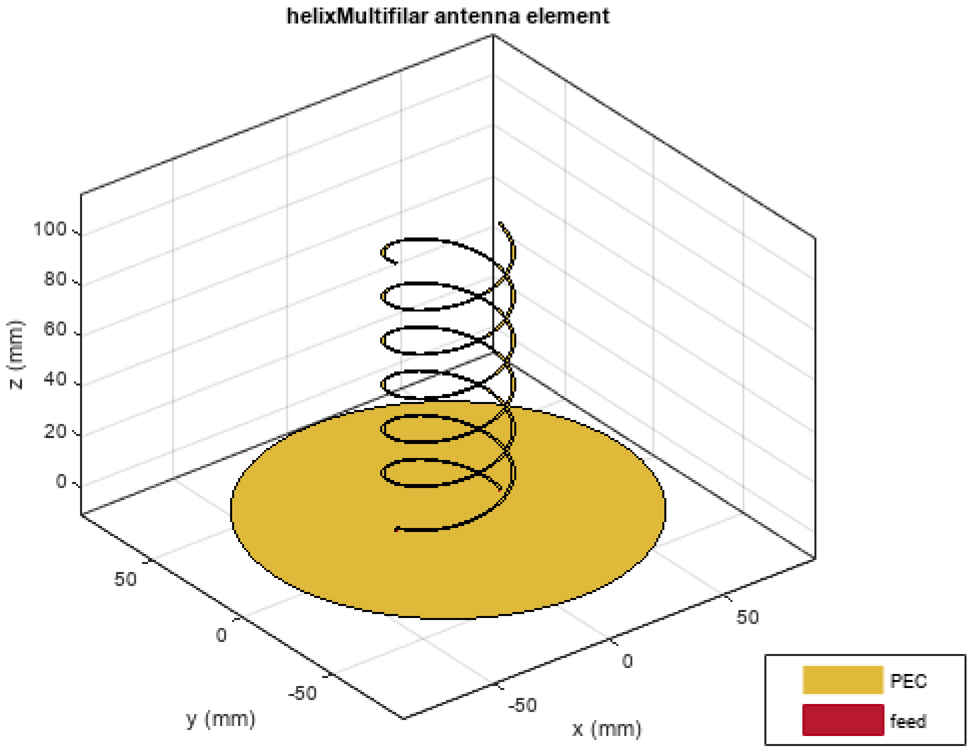
Bifilar Helix

Create and view a bifilar helix antenna.

```
ant=helixMultifilar('NumArms',2)
```

```
ant =
  helixMultifilar with properties:

              NumArms: 2
               Radius: 0.0220
                Width: 1.0000e-03
                Turns: 3
              Spacing: 0.0350
            ShortEnds: 0
     WindingDirection: 'CCW'
       FeedStubHeight: 1.0000e-03
    GroundPlaneRadius: 0.0750
          FeedVoltage: 1
            FeedPhase: 0
            Substrate: [1x1 dielectric]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



Radiation Pattern of Conical Multifilar Helix Antenna

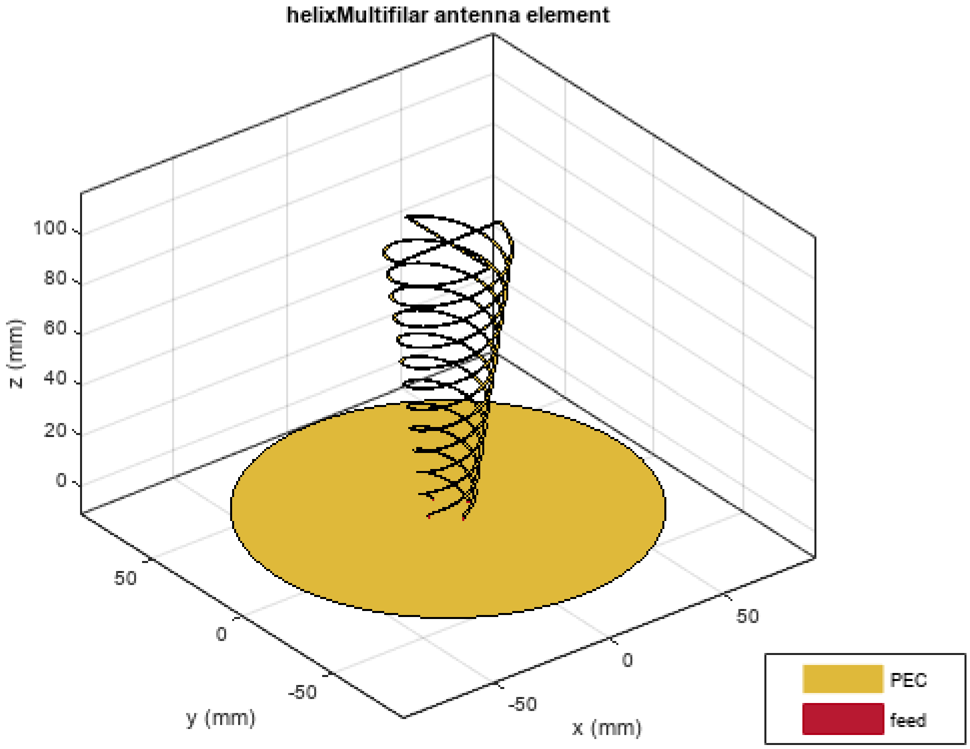
Create and view a conical multifilar helix antenna of radii, 0.0220 m and 0.00800 m respectively.

```
ant = helixMultifilar('Radius',[0.0080,0.0220],'ShortEnds',1)
```

```
ant = 
  helixMultifilar with properties:

             NumArms: 4
              Radius: [0.0080 0.0220]
               Width: 1.0000e-03
               Turns: 3
             Spacing: 0.0350
           ShortEnds: 1
    WindingDirection: 'CCW'
      FeedStubHeight: 1.0000e-03
    GroundPlaneRadius: 0.0750
         FeedVoltage: 1
           FeedPhase: 0
           Substrate: [1x1 dielectric]
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
```

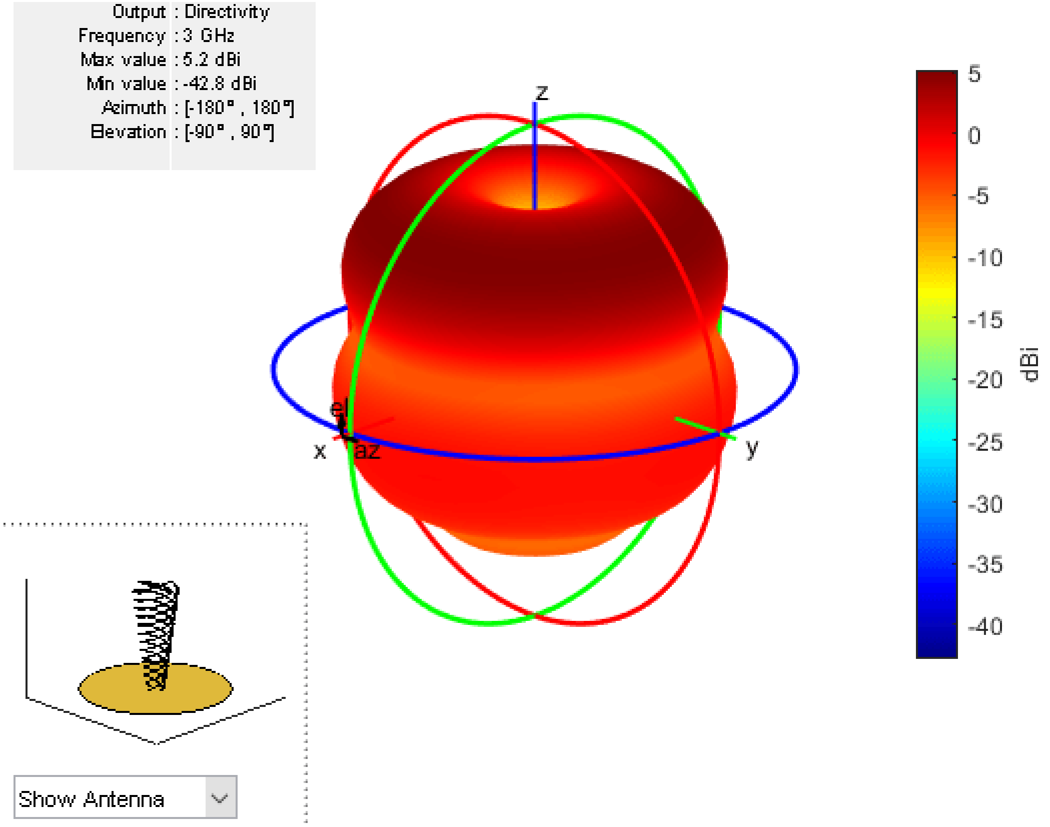
```
                      Load: [1x1 lumpedElement]
```

```
show(ant)
```



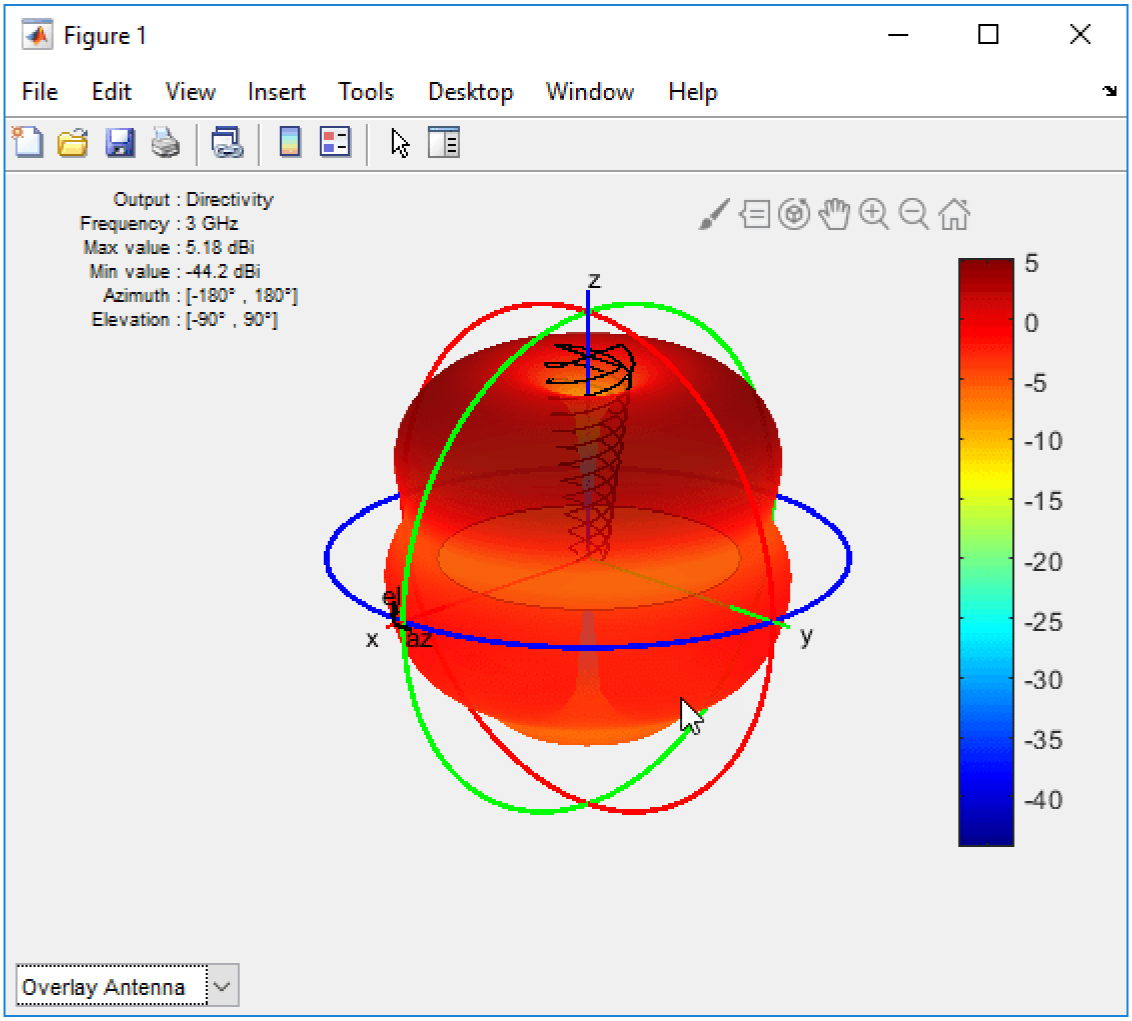
Plot the pattern of the antenna at 3 GHz.

```
pattern(ant,3e9)
```



Output : Directivity
Frequency : 3 GHz
Max value : 5.2 dBi
Min value : -42.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Overlay the antenna on the pattern.



Output : Directivity
Frequency : 3 GHz
Max value : 5.18 dBi
Min value : -44.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Quadfilar Helix Antenna with Dielectric Substrate

Create a custom quadfilar helix antenna with a FR4 dielectric substrate.

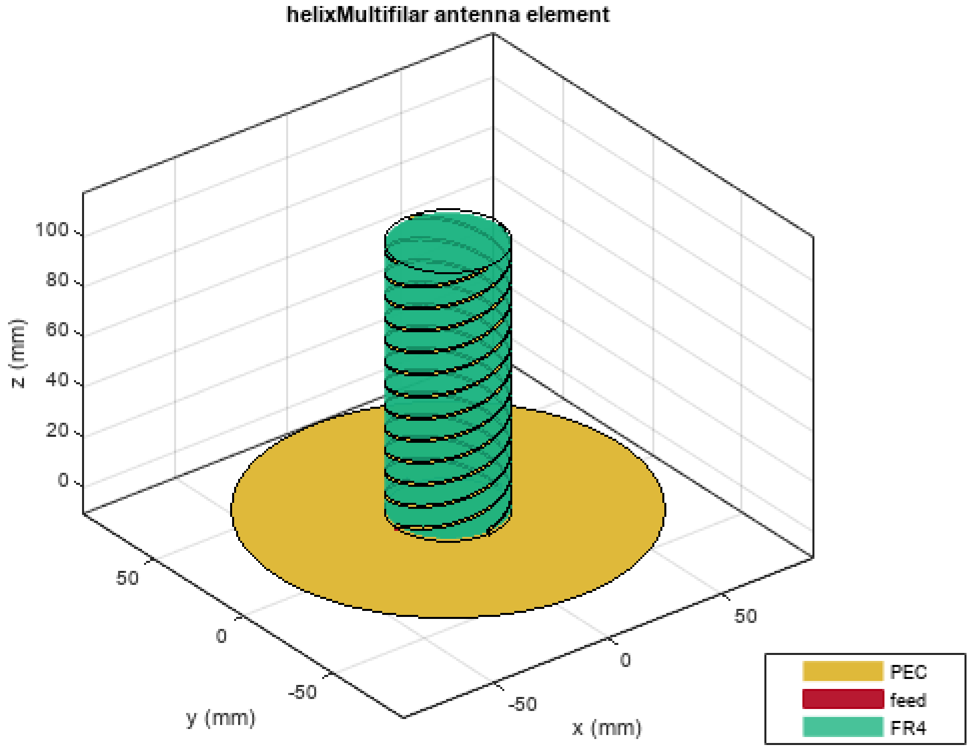
```
d = dielectric('FR4');
ant = helixMultifilar('Radius',22e-3 ,'NumArms',4,'Width',1e-3, 'Turns',3, 'Spacing',35e-3 ,'Fee
```

```
ant = 
  helixMultifilar with properties:

              NumArms: 4
               Radius: 0.0220
                Width: 1.0000e-03
                Turns: 3
              Spacing: 0.0350
            ShortEnds: 0
     WindingDirection: 'CCW'
       FeedStubHeight: 1.0000e-03
     GroundPlaneRadius: 0.0750
```

```
FeedVoltage: 1
  FeedPhase: 0
  Substrate: [1x1 dielectric]
  Conductor: [1x1 metal]
       Tilt: 0
   TiltAxis: [1 0 0]
       Load: [1x1 lumpedElement]
```

View quadfilar helix antenna.

```
show(ant)
```



Version History

Introduced in R2018b

See Also

helix | dipoleHelix | dipoleHelixMultifilar | cylinder2strip | helixpitch2spacing

dipoleHelixMultifilar

Create balanced bifilar or quadrafilar dipole helix antenna without circular ground plane

Description

The `dipoleHelixMultifilar` object creates a balanced bifilar or quadrafilar helix antenna without a circular ground plane. You can create both short-circuited and open-ended dipole helix multifilar antennas. Bifilar and quadrafilar helix antennas are used in aerospace and defense applications.

The width of the strip is related to the diameter of an equivalent cylinder by the equation

$$w = 2d = 4r$$

where:

- w is the width of the strip.
- d is the diameter of an equivalent cylinder.
- r is the radius of an equivalent cylinder.

For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default helix antenna is end-fed. The circular ground plane is on the *xy*- plane. Helix antennas are used commonly in axial mode. In this mode, the helix circumference is comparable to the operating wavelength, and the helix has maximum directivity along its axis. In normal mode, the helix radius is small compared to the operating wavelength. In this mode, the helix radiates broadside, that is, in the plane perpendicular to its axis. The basic equations for the helix are

$x = r\cos(\theta)$

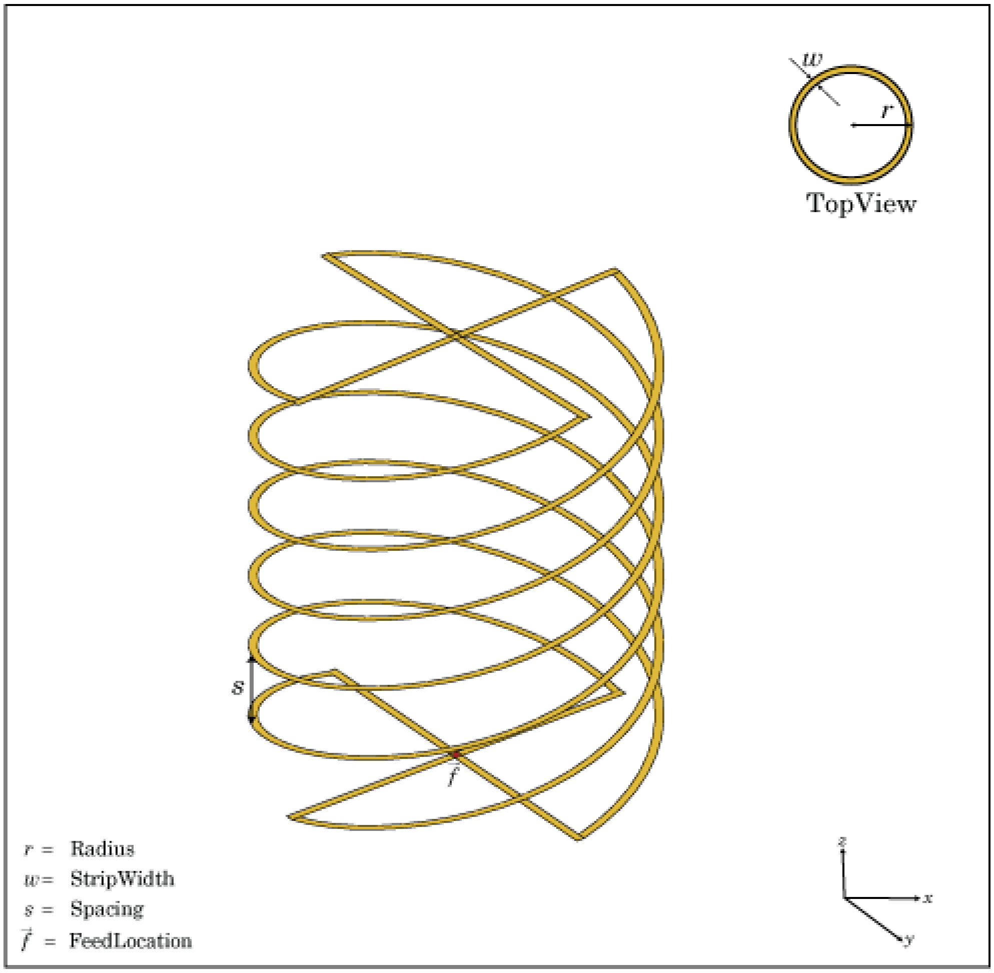
$y = r\sin(\theta)$

$z = S\theta$

where:

- r is the radius of the helical dipole.
- θ is the winding angle.
- S is the spacing between turns.

For a given pitch angle in degrees, use the `helixpitch2spacing` utility function to calculate the spacing between the turns in meters.



Creation

Syntax

```
ant = dipoleHelixMultifilar
ant = dipoleHelixMultifilar(Name,Value)
```

Description

`ant = dipoleHelixMultifilar` creates a bifilar or quadrafilar helix antenna without a circular ground plane. The default multifilar helical antenna is end-fed. The default helix operates around 2 GHz.

`ant = dipoleHelixMultifilar(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = dipoleHelixMultifilar('Radius',28e-03)` creates a multifilar helix with turns of radius 28e^{-03} m. Enclose each property name in quotes.

Properties

NumArms — Number of helical elements

4 (default) | 2

Number of helical elements, specified as a 4 or 2. Two elements create a bifilar dipole helix antenna, and four elements create a quadrafilar dipole helix antenna.

Example: 'NumArms',2

Example: ant.NumArms = 2

Data Types: double

Radius — Radius of turns

0.0220 (default) | positive real scalar

Radius of the turns, specified as a positive real scalar meter.

Example: 'Radius',28e-03

Example: ant.Radius = 28e-03

Data Types: double

Width — Width of strip

1.000e-03 (default) | positive real scalar

Width of the strip, specified as a positive real scalar in meters.

Example: 'Width',0.2

Example: ant.Width = 0.2

Data Types: double

Turns — Number of turns

3 (default) | scalar integer

Number of turns, specified as a scalar integer.

Example: 'Turns',4

Example: ant.Turns = 4

Data Types: double

Spacing — Spacing between turns

0.0350 (default) | positive real scalar

Spacing between the turns, specified as a positive real scalar in meters.

Example: 'Spacing',7.5e-2

Example: ant.Spacing = 7.5e-2

Data Types: double

ShortEnds — Status of ends of helix

1 (default) | 0

Status of ends of the helix, specified as 0 or 1. By default, the dipoleHelixMultifilar is short circuited. Setting the property to 0 makes the helix antenna an open circuit.

Example: 'ShortEnds',0

Example: ant.ShortEnds = 0

Data Types: double

WindingDirection — Direction of helix turns (windings)

'CCW' (default) | 'CW'

Direction of helix turns (windings), specified as CW or CCW.

Example: 'WindingDirection','CW'

Example: ant.WindingDirection = 'CW'

Data Types: char | string

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 LumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: ant.Load = lumpedElement('Impedance',75)

Data Types: double

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

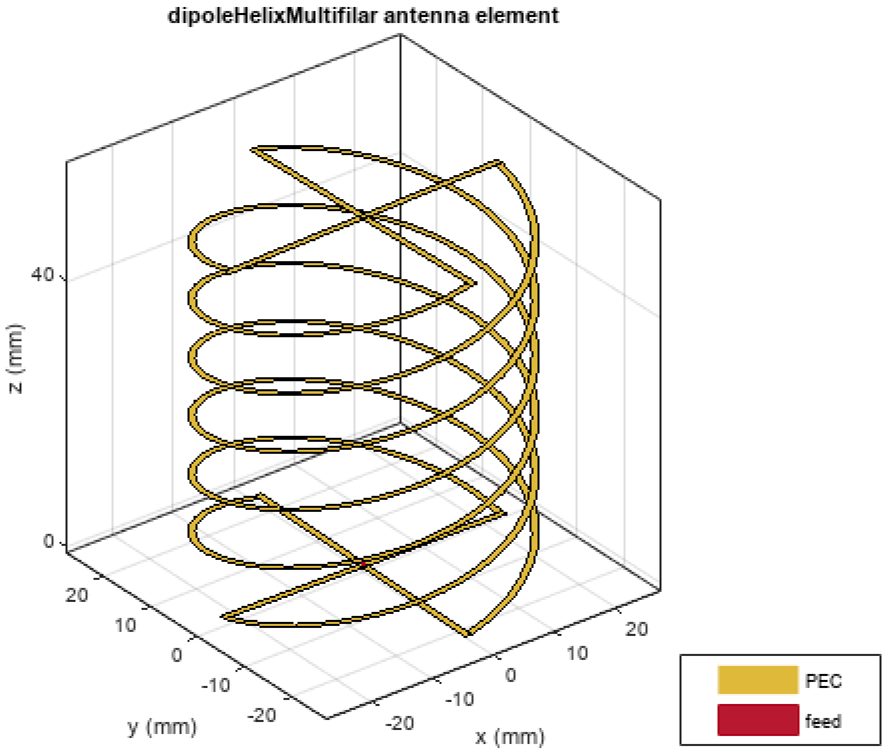
Default Multifilar Helical Dipole Antenna

Create and view a default multifilar helical dipole antenna.

```
ant = dipoleHelixMultifilar
```

```
ant = 
  dipoleHelixMultifilar with properties:

              NumArms: 4
               Radius: 0.0220
                Width: 1.0000e-03
                Turns: 3
              Spacing: 0.0350
            ShortEnds: 1
     WindingDirection: 'CCW'
            Conductor: [1x1 metal]
                 Tilt: 0
            TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

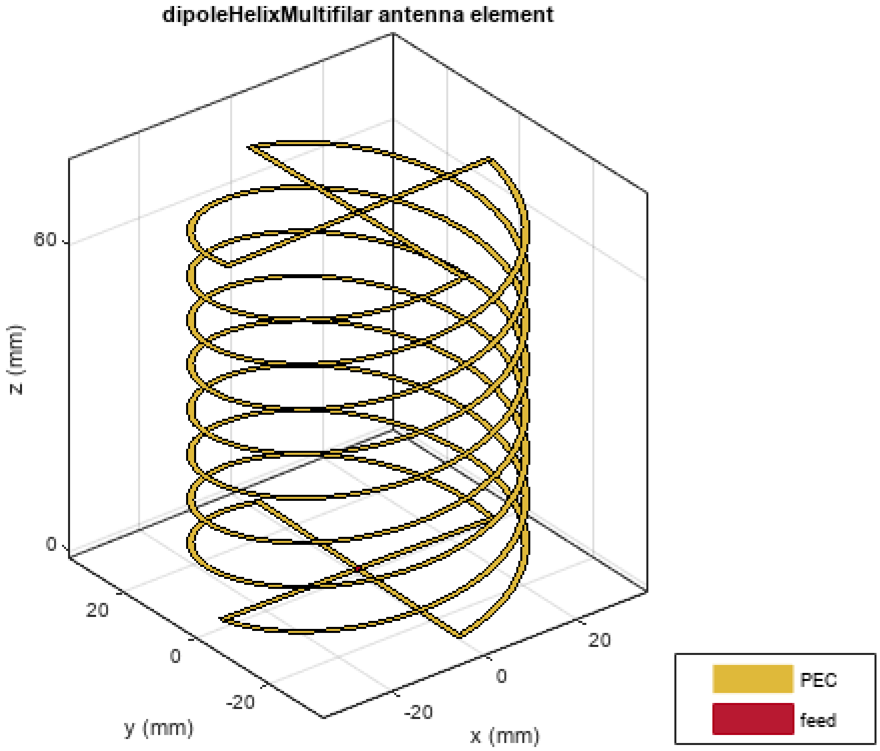
```
show(ant)
```



Quadrafilar Helical Dipole Antenna and Radiation Pattern

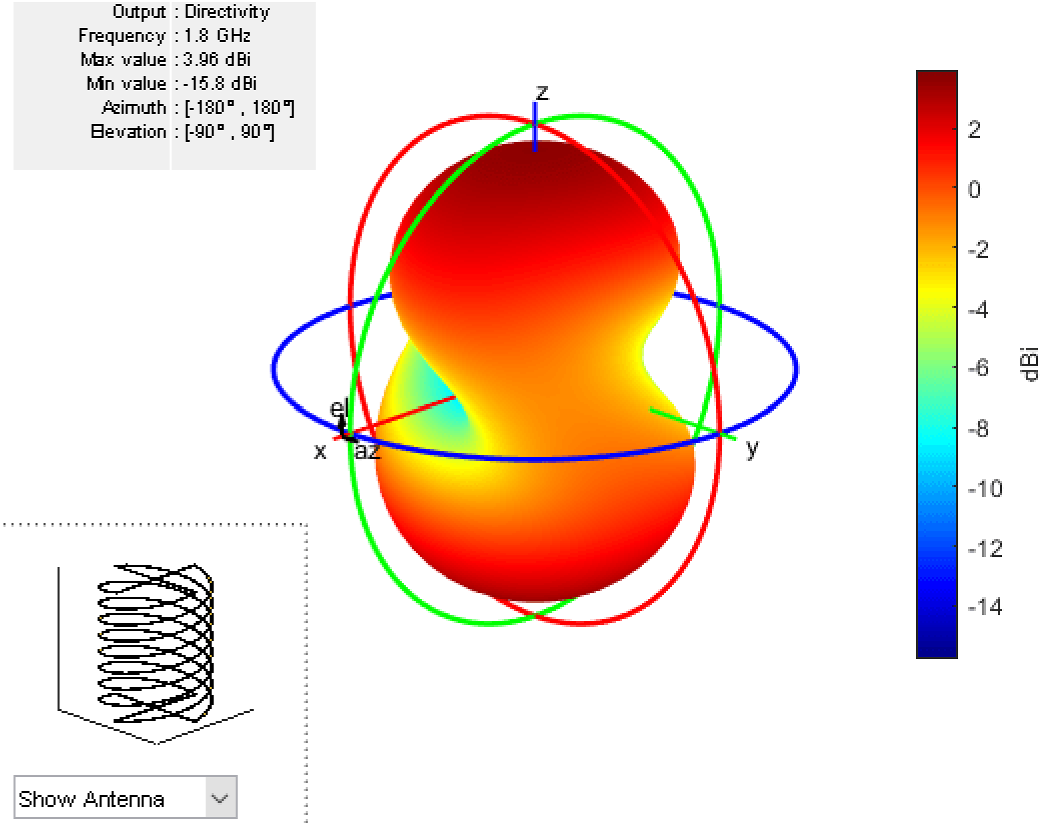
Create and view a quadrafilar helical dipole antenna with turn radius of 28 mm and strip width of 1.2 mm.

```
ant = dipoleHelixMultifilar('Radius',28e-3,'Width',1.2e-3,'Turns',4);
show(ant)
```



Plot the radiation pattern of the helical dipole at 1.8 GHz.

```
pattern(ant,1.8e9);
```



```
Output : Directivity
Frequency : 1.8 GHz
Max value : 3.96 dBi
Min value : -15.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2018b

See Also

`helix` | `dipoleHelix` | `helixMultifilar`

fractalGasket

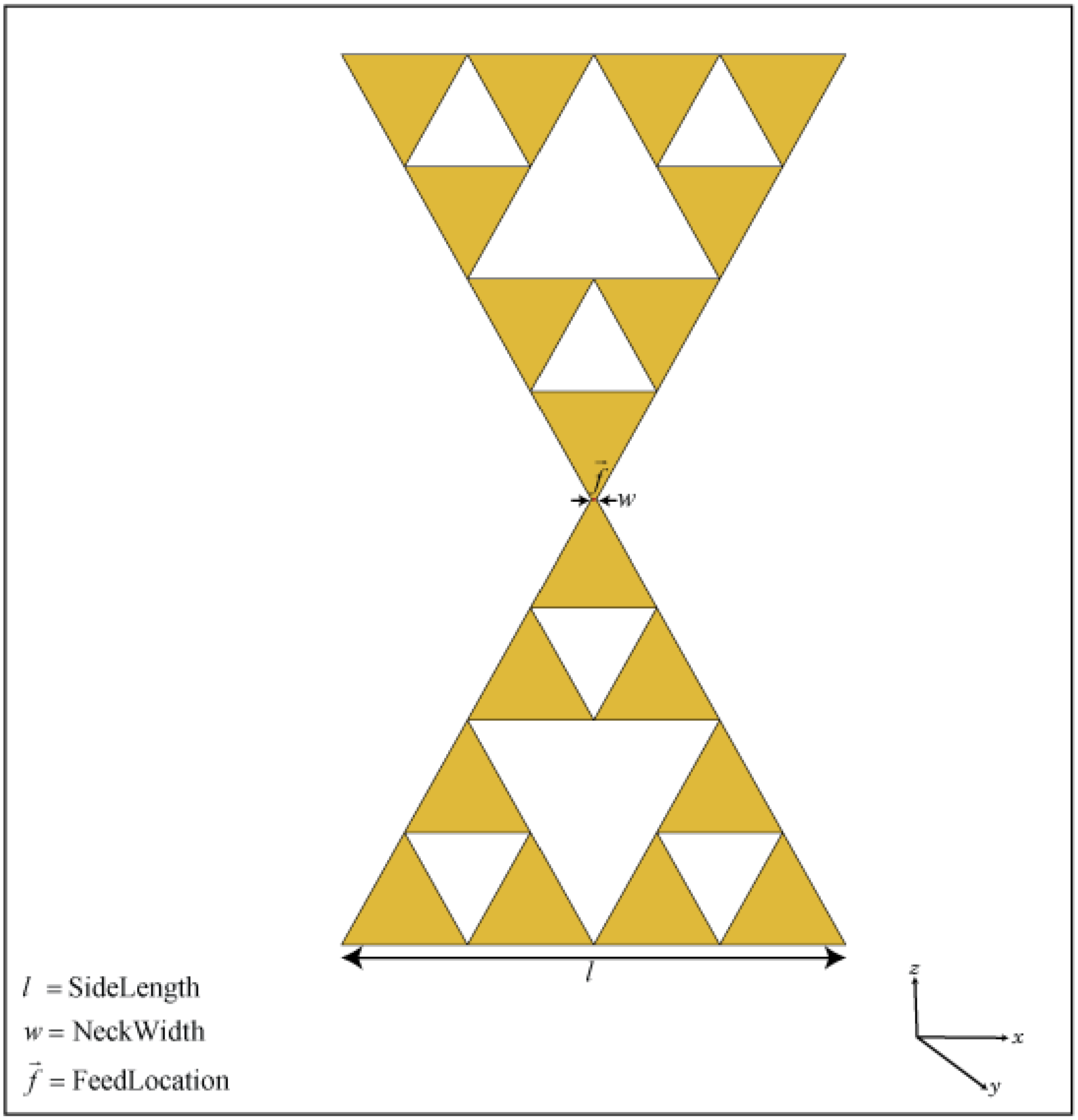
Create Sierpinski's Gasket fractal antenna on *xy*- plane

Description

The `fractalGasket` object creates an equilateral triangle-shaped Sierpinski's Gasket fractal antenna. These fractals are used in building communications systems, wireless networks, universal tactic communications systems, mobile devices, telematics, and radio frequency identification (RFID) antennas.

A fractal antenna uses a self-similar design to maximize the length or increase the perimeter of a material that transmits or receives electromagnetic radiation within a given volume or area. The main advantage of fractal antennas is that they are compact, which is important requirement for small and complex circuits. Fractal antennas also have more input impedance or resistance due to increased length or perimeter.

All fractal antennas are printed structures that are etched on a dielectric substrate.



Creation

Syntax

```
ant = fractalGasket
ant = fractalGasket(Name,Value)
```

Description

`ant = fractalGasket` creates an equilateral triangle-shaped Sierpinski's gasket fractal antenna. The default planar fractal antenna is in the shape of a bowtie which is center-fed. The antenna resonates at a frequency of 1.3 GHz.

`ant = fractalGasket(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = fractalGasket('NumIterations',4)` creates a Sierpinski's Gasket with four iterations.

Properties

`NumIterations` — Number of iterations of fractal antenna

`2` (default) | scalar integer

Number of iterations of the fractal antenna, specified as a scalar integer.

Example: `'NumIterations',2`

Example: `ant.NumIterations = 2`

Data Types: `double`

`Side` — Lengths for three sides of triangle

`0.2000` (default) | scalar | two-element vector | three-element vector

Lengths for three sides of the triangle, specified as a scalar in meters or a two- or three-element vector in meters.

- Scalar – The triangle is equilateral.
- Two-element vector – The first value specifies the base of the triangle along the *x*-axis. The second value specifies the other two sides of the triangle. The triangle is isosceles.
- Three-element vector – The first value specifies the base of the triangle along the *x*-axis. The remaining two values specify the other two sides of the triangle. The triangle is scalene.

Example: `'Side',[0.5000,1.000]`

Example: `ant.Side = [0.5000,1.000]`

Data Types: `double`

`NeckWidth` — Width at neck of fractal antenna

`0.0020` (default) | positive scalar integer

Width at the neck of the fractal antenna where the feed is located, specified as a positive scalar integer in meters.

Example: `'NeckWidth',0.0050`

Example: `ant.NeckWidth = 0.0050`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

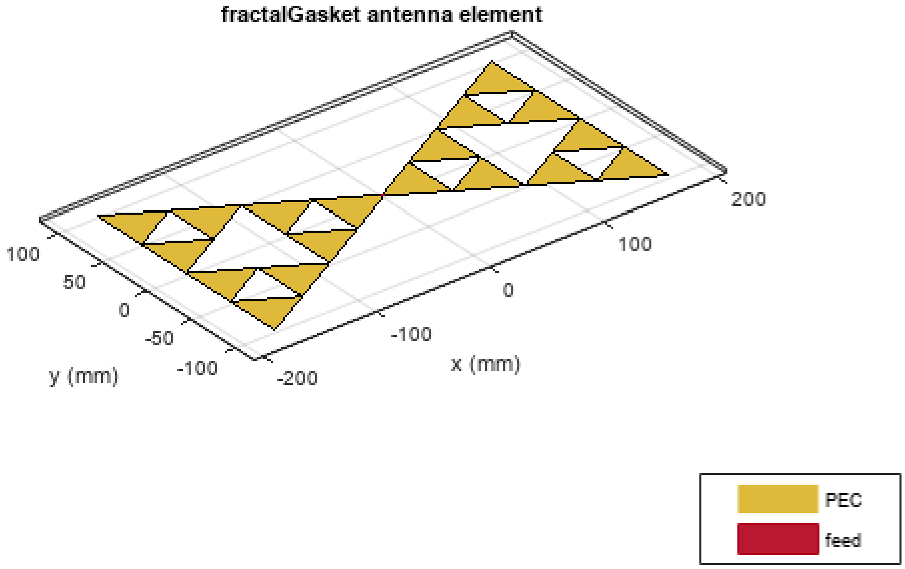
Default Sierpinski's Gasket

Create and view a default fractal Sierpinski's Gasket.

```
ant = fractalGasket
```

```
ant = 
  fractalGasket with properties:

    NumIterations: 2
             Side: 0.2000
        NeckWidth: 0.0020
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2018b

See Also

`fractalKoch` | `fractalCarpet` | `fractalIsland`

Topics

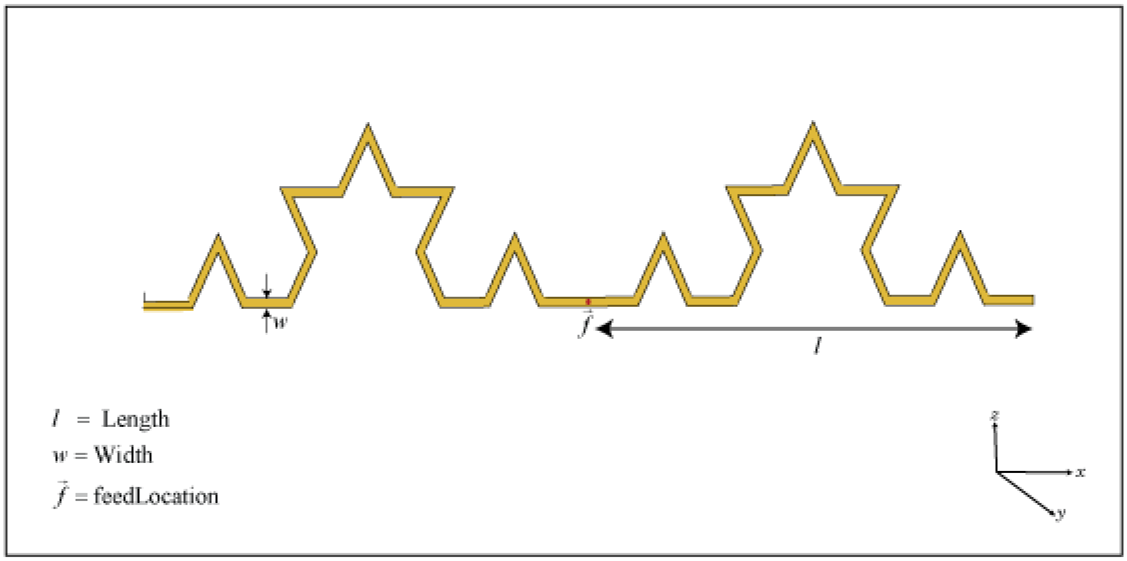
"Rotate Antennas and Arrays"

fractalKoch

Create Koch curve fractal dipole or loop antenna on *xy*- plane

Description

The `fractalKoch` object creates a Koch curve fractal dipole or loop antenna on an *xy*- plane. These fractals are used in multiband and wideband applications like Global System for Mobile Communications (GSM), Universal Mobile Telecommunication Service (UMTS), and Bluetooth.



A fractal antenna uses a self-similar design to maximize the length or increase the perimeter of a material that transmits or receives electromagnetic radiation within a given volume or area. The main advantage of fractal antennas is that they are compact, which is an important requirement for small and complex circuits. Fractal antennas also have more input impedance or resistance due to increased length or perimeter, respectively.

All fractal antennas are printed structures that are etched on a dielectric substrate.

Creation

Syntax

```
ant = fractalKoch
ant = fractalKoch(Name,Value)
```

Description

`ant = fractalKoch` creates a Koch curve fractal antenna on an X-Y plane. The default is a dipole with Koch curve length chosen for an operating frequency of 0.86 GHz.

`ant = fractalKoch(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = fractalKoch('NumIterations',4)` creates a Koch curve fractal antenna with four iterations. Enclose each property name in quotes.

Properties

`NumIterations` — Number of iterations of fractal antenna

`2` (default) | scalar integer

Number of iterations of the fractal antenna, specified as a scalar integer.

Example: `'NumIterations',2`

Example: `ant.NumIterations = 2`

Data Types: `double`

`Length` — Length of Koch curve along X-axis

`0.0600` (default) | positive scalar integer

Length of the Koch curve along the x-axis, specified as a positive scalar integer in meters.

Example: `'Length',0.5000`

Example: `ant.Length = 0.5000`

Data Types: `double`

`Width` — Width of Koch curve along Y-axis

`1.0000e-03` (default) | positive scalar integer

Width of the Koch curve along y-axis, specified as a positive scalar integer in meters.

Example: `'Width',0.0050`

Example: `ant.Width = 0.0050`

Data Types: `double`

`Type` — Type of Koch configuration

`'dipole'` (default) | `'loop'`

Type of Koch configuration, specified as `'dipole'` or `'loop'`.

Example: `'Type','loop'`

Example: `ant.Type = 'loop'`

Data Types: `char` | `string`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

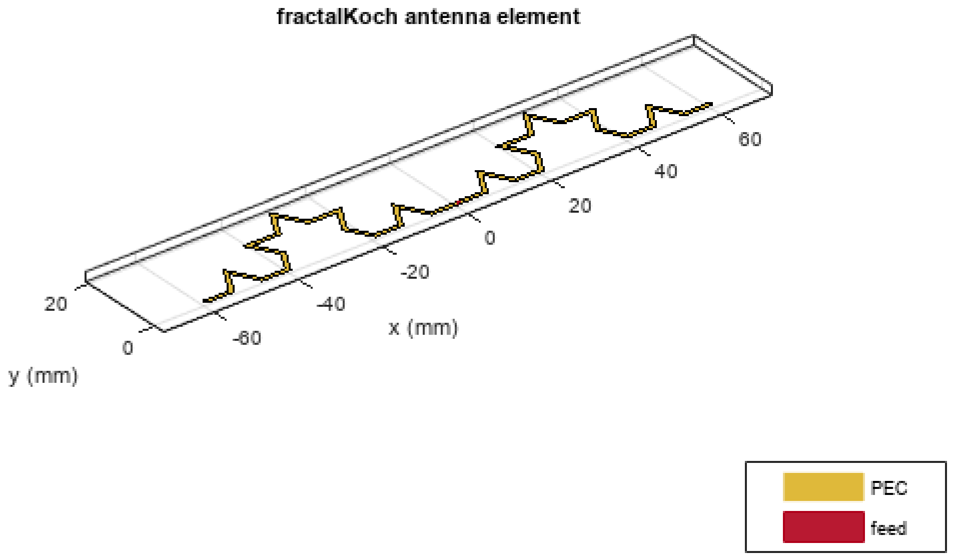
Default Koch Curve Fractal Antenna

Create and view a default Koch curve fractal antenna.

```
ant = fractalKoch
```

```
ant = 
  fractalKoch with properties:

    NumIterations: 2
           Length: 0.0600
            Width: 1.0000e-03
             Type: 'dipole'
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

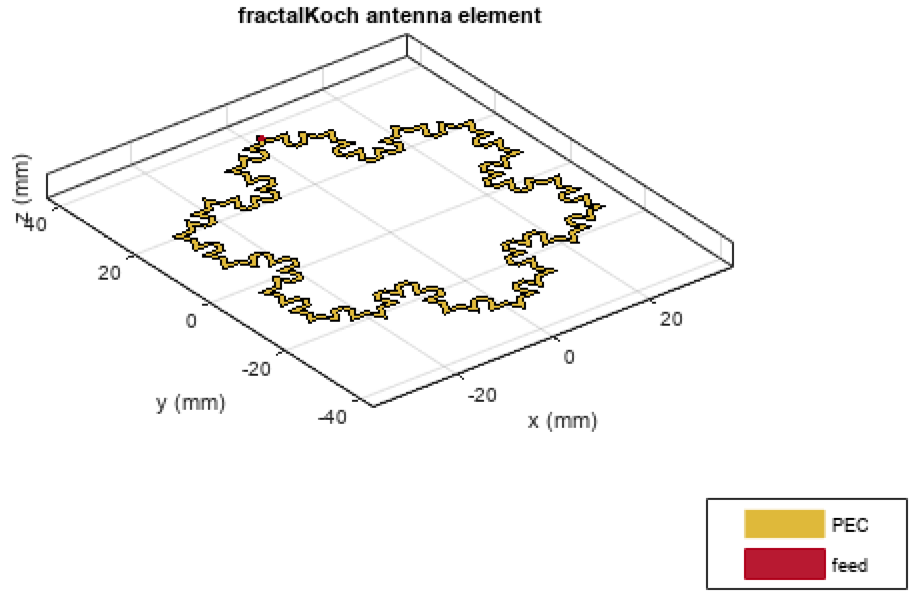
```
show(ant)
```



Koch Loop Fractal Antenna

Create and view a Koch loop fractal antenna with three iterations.

```
ant = fractalKoch('NumIterations',3,'Type','loop');
show(ant)
```



Version History

Introduced in R2018b

See Also

`fractalGasket` | `fractalCarpet` | `fractalIsland`

Topics

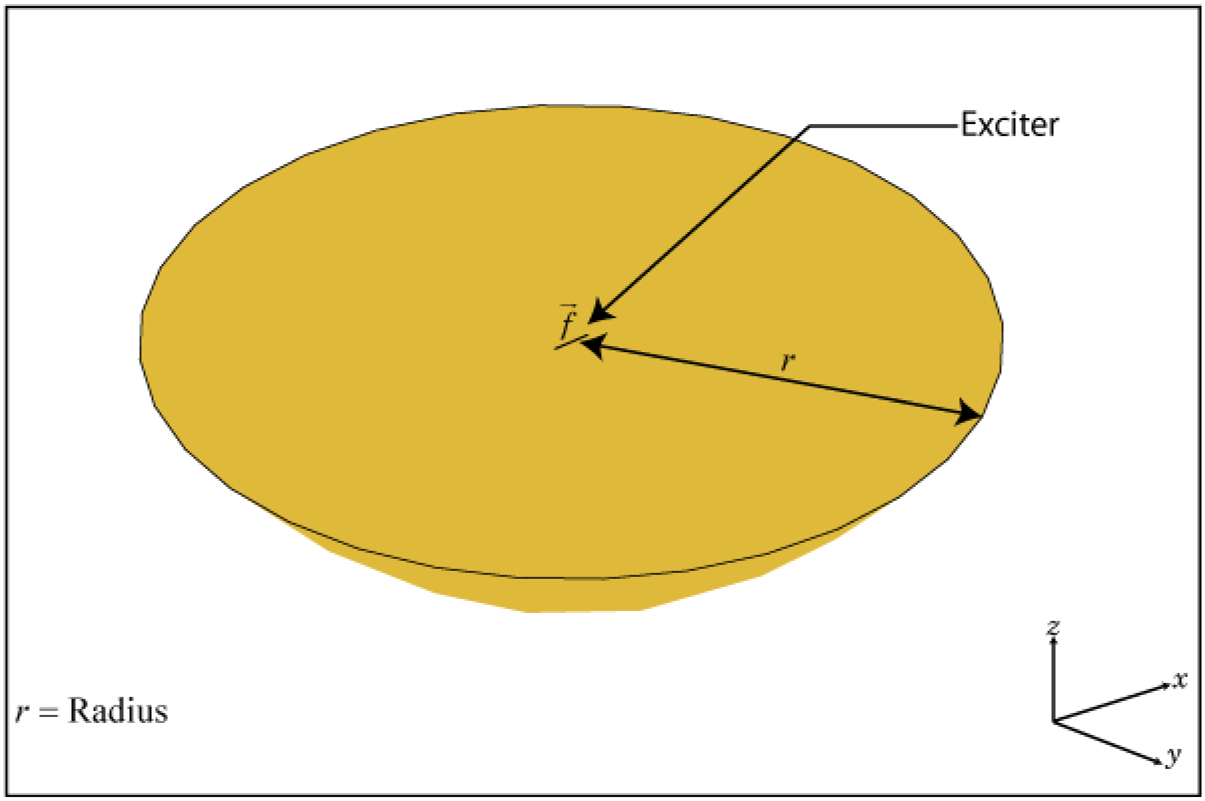
"Rotate Antennas and Arrays"

reflectorParabolic

Create parabolic reflector antenna

Description

The `reflectorParabolic` object creates a parabolic reflector antenna. Parabolic reflector antennas are electrically large structures and are at least 10 wavelengths in diameter. These reflectors are used in TV antennas and satellite communications, for example.



Creation

Syntax

```
ant = reflectorParabolic
ant = reflectorParabolic(Name,Value)
```

Description

`ant = reflectorParabolic` creates a dipole-fed parabolic reflector antenna. The default antenna exciter operates at 10 GHz. The reflector is 10λ in diameter, where λ corresponds to the value of wavelength.

`ant = reflectorParabolic(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = reflectorParabolic('FocalLength',0.5)` creates a parabolic reflector antenna of focal length 0.5 meters.

Properties

`Exciter` — Antenna or array type used as exciter

`dipole` (default) | `antenna` object | `array` object

Antenna type used as an exciter, specified as any single-element antenna object. Except reflector and cavity antenna elements, you can use any of the antenna elements or array elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',horn`

Example: `ant.Exciter = horn`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Radius` — Radius of parabolic reflector

`0.1500` (default) | positive scalar integer

Radius of the parabolic reflector, specified as a positive scalar integer in meters.

Example: `'Radius',0.22`

Example: `ant.Radius = 0.22`

Data Types: `double`

`FocalLength` — Focal length of parabolic dish

`0.0750` (default) | positive scalar integer

Focal length of the parabolic dish, specified as a positive scalar integer in meters.

Example: `'FocalLength',0.0850`

Example: `ant.FocalLength = 0.0850`

Data Types: `double`

`FeedOffset` — Signed distance from focus

`[0 0 0]` (default) | three-element vector

Signed distance from the focus of the parabolic dish, specified as a three-element vector in meters. By default, the antenna exciter is at the focus of the parabola. Using the `FeedOffset` property, you can place the exciter anywhere on the parabola.

Example: `'FeedOffset',[0.0850 0 0]`

Example: `ant.FeedOffset = [0.0850 0 0]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

SolverType — Solver for antenna analysis

'MoM-PO' (default) | 'MoM' | 'FMM'

Solver for antenna analysis, specified as the comma-separated pair consisting of 'SolverType' and 'MoM-PO' or 'MoM' (Method of Moments) or 'FMM' (Fast Multipole Method).

Example: 'SolverType','MOM'

Data Types: char

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

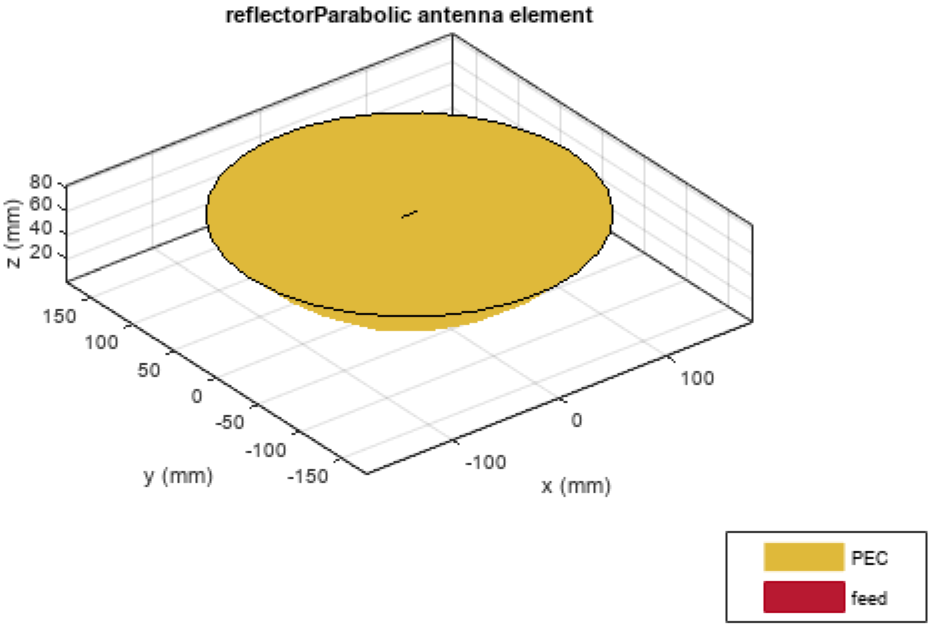
Default Parabolic Reflector and Radiation Pattern

Create and view a default parabolic reflector antenna.

```
ant = reflectorParabolic
```

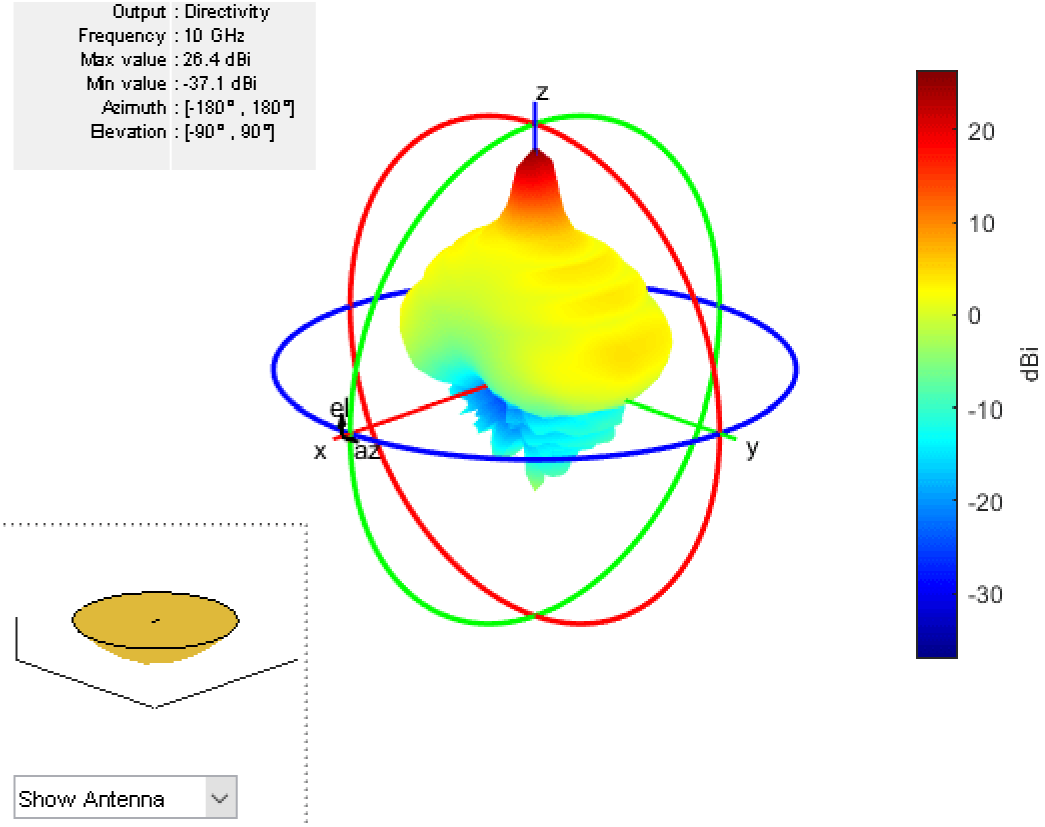
```
ant = 
  reflectorParabolic with properties:

        Exciter: [1x1 dipole]
         Radius: 0.1500
    FocalLength: 0.0750
     FeedOffset: [0 0 0]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
     SolverType: 'MoM-PO'
```

```
show(ant)
```



Plot the radiation pattern of the parabolic reflector at 10 GHz.

```
pattern(ant,10e9)
```



Output : Directivity
Frequency : 10 GHz
Max value : 26.4 dBi
Min value : -37.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Parabolic Reflector-Backed Circular Array of Equiangular Spiral Antenna

Create a circular array of equiangular spiral antennas.

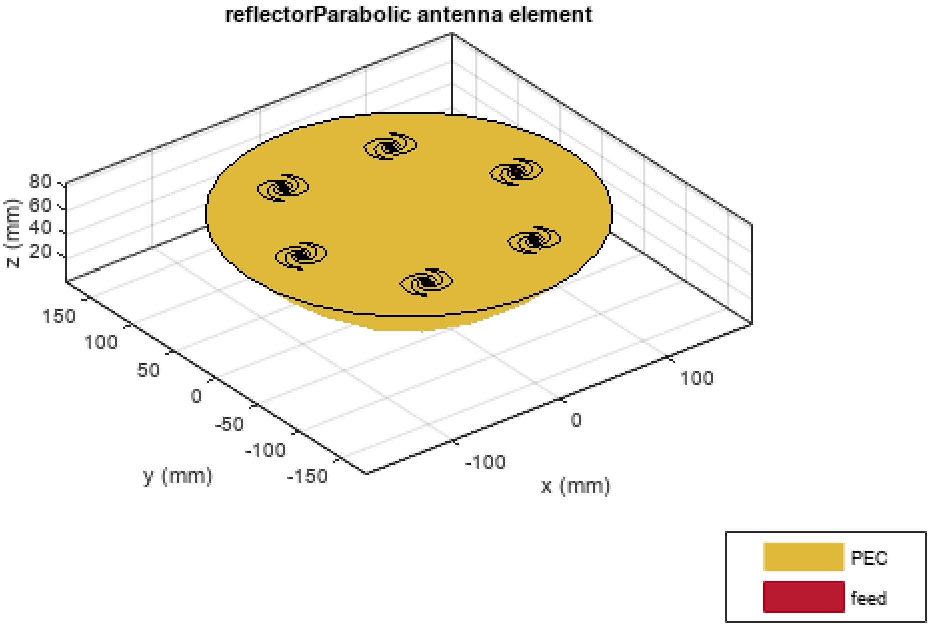
```
circA = circularArray('Element',spiralEquiangular,'Radius',0.1);
```

Create a parabolic reflector-backed antenna.

```
ant = reflectorParabolic('Exciter',circA)
```

```
ant = 
  reflectorParabolic with properties:

        Exciter: [1x1 circularArray]
         Radius: 0.1500
    FocalLength: 0.0750
     FeedOffset: [0 0 0]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
     SolverType: 'MoM-PO'
```

```
show(ant)
```



Version History

Introduced in R2018b

See Also

Topics

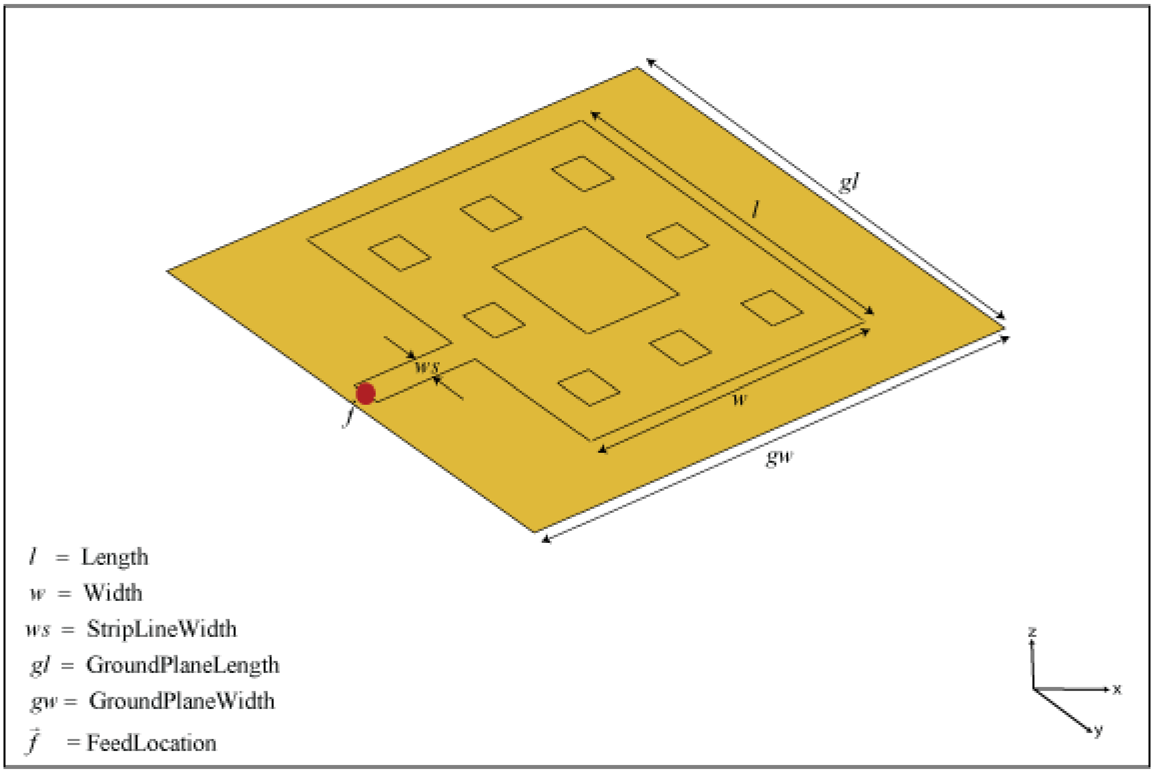
“Rotate Antennas and Arrays”

fractalCarpet

Create Sierpinski's carpet fractal antenna

Description

The `fractalCarpet` object creates a Sierpinski's carpet fractal antenna. These fractal antennas are used in mobile phone and Wi-Fi® communications.



A fractal antenna uses a self-similar design to maximize the length or increase the perimeter of a material that transmits or receives electromagnetic radiation within a given volume or area. The main advantage of fractal antennas is that they are compact, which is an important requirement for small and complex circuits. Fractal antennas also have more input impedance or resistance due to increased length or perimeter.

All fractal antennas are printed structures that are etched on a dielectric substrate.

Creation

Syntax

```
ant = fractalCarpet
ant = fractalCarpet(Name,Value)
```

Description

`ant = fractalCarpet` creates a Sierpinski’s carpet fractal antenna. The default fractal is centered at the origin, and the number of iterations is set to 2. The length of the fractal is for an operating frequency of 5.45 GHz.

`ant = fractalCarpet(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = fractalCarpet('NumIterations',4)` creates a Sierpinski's carpet with four iterations.

Properties

`NumIterations` — Number of iterations performed on fractal antenna
2 (default) | scalar integer

Number of iterations performed on the fractal antenna, specified as a scalar integer.

Example: `'NumIterations',4`

Example: `ant.NumIterations = 4`

Data Types: `double`

`Length` — Length of fractal carpet along *x*-axis
`0.0280` (default) | positive scalar integer

Length of the fractal carpet along the *x*-axis, specified as a positive scalar integer in meters.

Example: `'Length',0.5000`

Example: `ant.Length = 0.5000`

Data Types: `double`

`Width` — Width of fractal carpet along *y*-axis
`0.00370` (default) | positive scalar integer

Width of the fractal carpet along the *y*-axis, specified as a positive scalar integer in meters.

Example: `'Width',0.0050`

Example: `ant.Width = 0.0050`

Data Types: `double`

`Height` — Height of fractal carpet above ground
`0.0016` (default) | positive scalar integer

Height of the fractal carpet above the ground plane along the *z*-axis, specified as a positive scalar integer in meters.

Example: `'Height',0.0034`

Example: `ant.Height = 0.0034`

Data Types: `double`

`StripLineWidth` — Width of feeding strip line
`0.0030` (default) | positive scalar integer

Width of the feeding strip line, specified as a positive scalar integer in meters.

Example: 'StripLineWidth',0.0050

Example: ant.StripLineWidth = 0.0050

Data Types: double

Substrate — Type of dielectric material

'Air' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. For more information, see dielectric.

Example: d = dielectric('FR4'); ant = fractalCarpet('Substrate',d)

Example: d = dielectric('FR4'); ant = fractalCarpet; ant.Substrate = d;

Data Types: string | char

GroundPlaneLength — Length of ground plane

0.0480 (default) | positive scalar integer

Length of the ground plane, specified as a positive scalar integer in meters.

Example: 'GroundPlaneLength',0.0550

Example: ant.GroundPlaneLength = 0.0550

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0480 (default) | positive scalar integer

Width of the ground plane, specified as a positive scalar integer in meters.

Example: 'GroundPlaneWidth',0.0550

Example: ant.GroundPlaneWidth = 0.0550

Data Types: double

FractalCenterOffset — Signed distance of fractal carpet center from origin

[0 0] (default) | two-element real-valued vector

Signed distance of the fractal carpet center from the origin, specified as a two-element real-valued vector with each element unit in meters.

Example: 'FractalCenterOffset',[0 0.080]

Example: ant.FractalCenterOffset = [0 0.080]

Data Types: double

FeedOffset — Signed distance of feed from origin

[0 0] (default) | two-element real-valued vector

Signed distance of the feed from the origin, specified as a two-element real-valued vector with each element unit in meters.

Example: 'FeedOffset',[0 0.080]

Example: ant.FeedOffset = [0 0.080]

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Default Sierpinski's Carpet Antenna

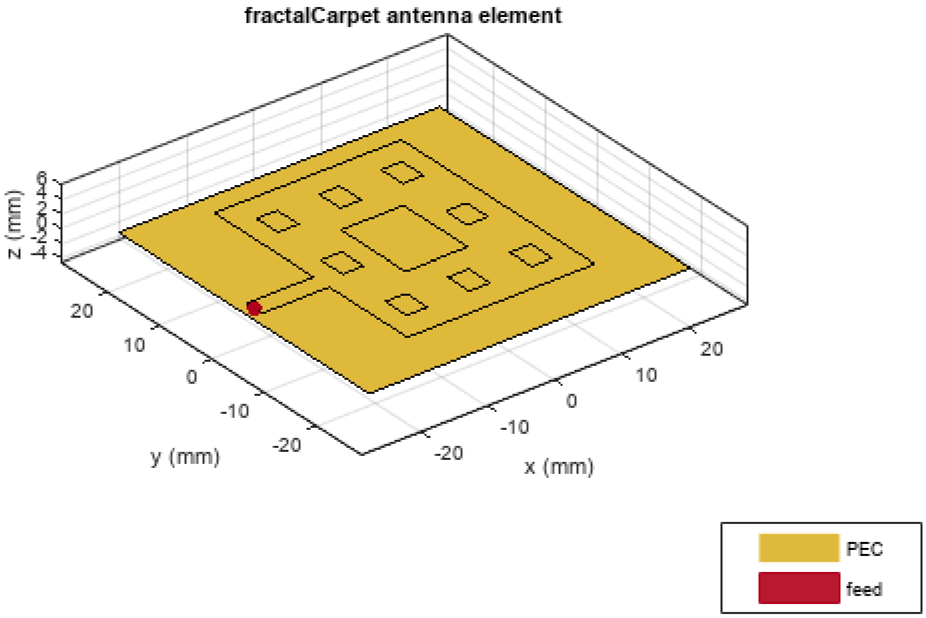
Create and view a Sierpinski's carpet fractal antenna with default property values.

```
ant = fractalCarpet
```

```
ant = 
  fractalCarpet with properties:

          NumIterations: 2
                 Length: 0.0280
                  Width: 0.0370
         StripLineWidth: 0.0030
             FeedOffset: [-0.0240 -0.0020]
                 Height: 0.0016
              Substrate: [1x1 dielectric]
      GroundPlaneLength: 0.0480
       GroundPlaneWidth: 0.0480
```

```
    FractalCenterOffset: [0 0]
              Conductor: [1x1 metal]
                   Tilt: 0
               TiltAxis: [1 0 0]
                   Load: [1x1 lumpedElement]
```

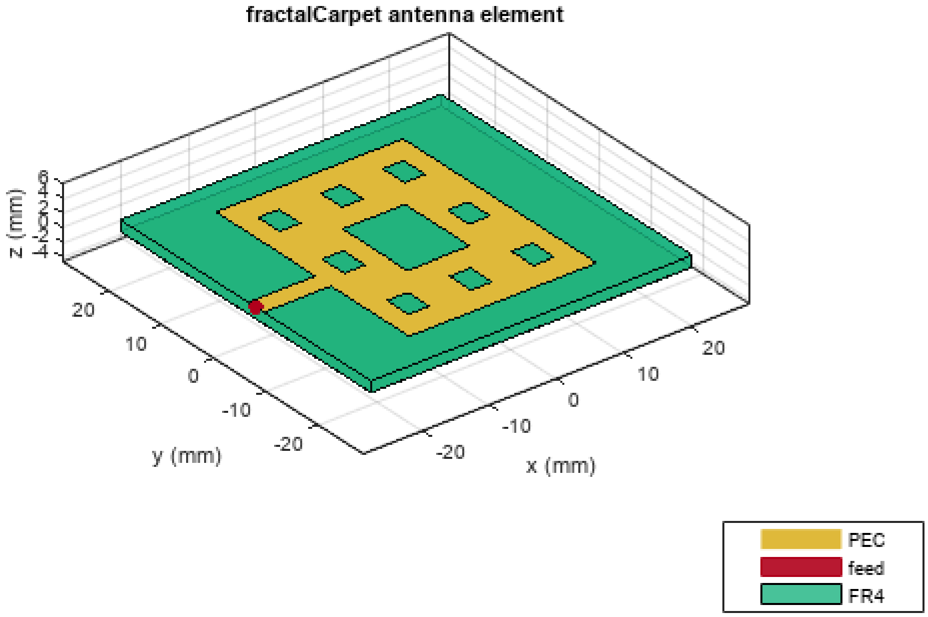
```
show(ant)
```



Radiation Pattern of Sierpinski's Carpet Antenna on FR4 Substrate

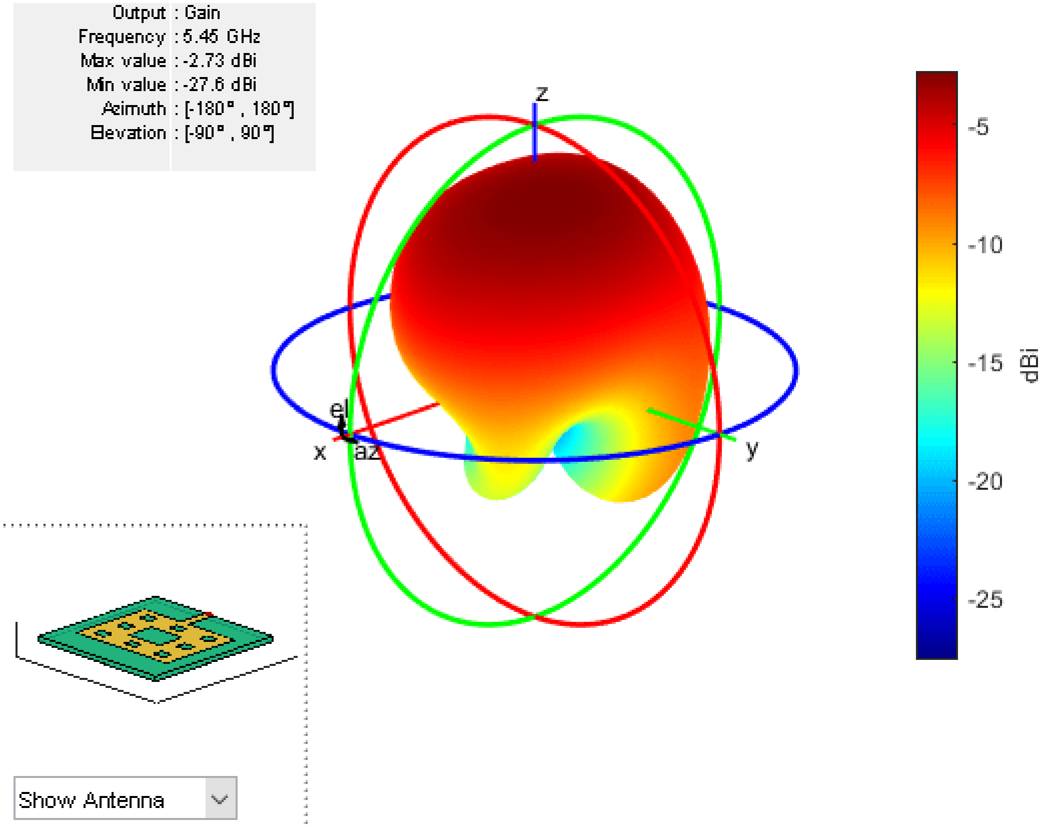
Create and view a Sierpinski's carpet fractal antenna on FR4 substrate.

```
ant = fractalCarpet('Substrate',dielectric('FR4'));
show(ant)
```



Plot the radiation pattern of the antenna at 5.45 GHz.

```
pattern(ant,5.45e9)
```



```
   Output : Gain
Frequency : 5.45 GHz
Max value : -2.73 dBi
Min value : -27.6 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2019a

See Also

`fractalKoch` | `fractalGasket` | `fractalIsland`

Topics

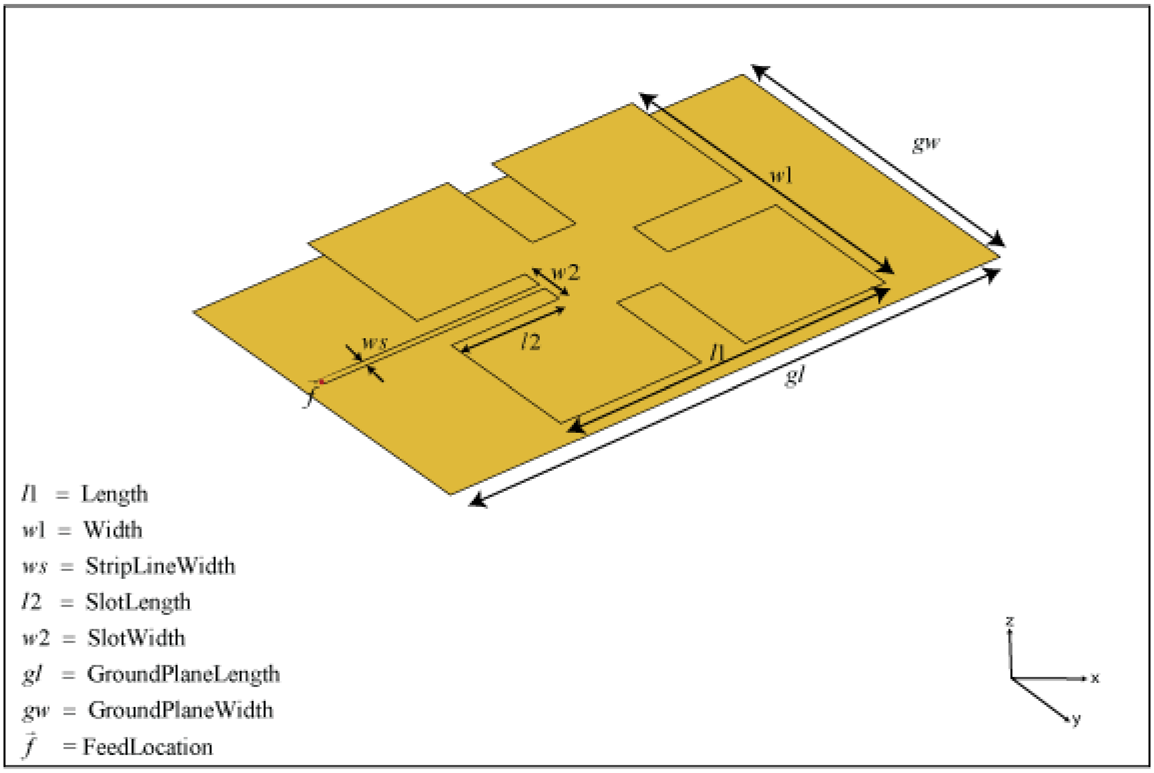
“Rotate Antennas and Arrays”

fractalIsland

Minkowski's loop fractal antenna

Description

The `fractalIsland` object creates a Minkowski's loop fractal antenna. These fractal antennas are used in mobile phone and Wi-Fi communications.



A fractal antenna uses a self-similar design to maximize the length or increase the perimeter of a material that transmits or receives electromagnetic radiation within a given volume or area. The main advantage of fractal antennas is that they are compact, which is an important requirement for small and complex circuits. Fractal antennas also have more input impedance or resistance due to increased length or perimeter.

All fractal antennas are printed structures that are etched on a dielectric substrate.

Creation

Syntax

```
ant = fractalIsland
ant = fractalIsland(Name,Value)
```

Description

`ant = fractalIsland` creates a Minkowski's loop fractal antenna. The default fractal is centered at the origin, and the number of iterations is set to 2. The length of the fractal is for an operating frequency of 6 GHz.

`ant = fractalIsland(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = fractalIsland('NumIterations',4)` creates a Minkowski's loop with four iterations.

Properties

`NumIterations` — Number of iterations performed on fractal antenna
`2` (default) | scalar integer

Number of iterations performed on the fractal antenna, specified as a scalar integer.

Example: `'NumIterations',4`

Example: `ant.NumIterations = 4`

Data Types: `double`

`Length` — Length of fractal island along *x*-axis
`0.0295` (default) | positive scalar integer

Length of the fractal island along the *x*-axis, specified as a positive scalar integer in meters.

Example: `'Length',0.5000`

Example: `ant.Length = 0.5000`

Data Types: `double`

`Width` — Width of fractal island along *y*-axis
`0.0295` (default) | positive scalar integer

Width of the fractal island along the *y*-axis, specified as a positive scalar integer in meters.

Example: `'Width',0.0050`

Example: `ant.Width = 0.0050`

Data Types: `double`

`StripLineWidth` — Width of feeding strip line
`6.0000e-04` (default) | positive scalar integer

Width of the feeding strip line, specified as a positive scalar integer in meters.

Example: `'StripLineWidth',3.0000e-04`

Example: `ant.StripLineWidth = 3.0000e-04`

Data Types: `double`

`SlotLength` — Length of slot along *x*-axis
`0.0040` (default) | positive scalar integer

Length of the slot along the *x*-axis, specified as a positive scalar integer in meters.

Example: `'SlotLength',0.0050`

Example: `ant.SlotLength = 0.0050`

Data Types: `double`

`SlotWidth` — Width of slot along *y*-axis

`0.0040` (default) | positive scalar integer

Width of the slot along the *y*-axis, specified as a positive scalar integer in meters.

Example: `'SlotWidth',0.0050`

Example: `ant.SlotWidth = 0.0050`

Data Types: `double`

`Height` — Height of fractal above ground

`0.0016` (default) | positive scalar integer

Height of the fractal above the ground plane along the *z*-axis, specified as a positive scalar integer in meters.

Example: `'Height',0.0034`

Example: `ant.Height = 0.0034`

Data Types: `double`

`Substrate` — Type of dielectric material

`'Air'` (default) | `dielectric` object

Type of dielectric material used as a substrate, specified as a dielectric object. For more information, see `dielectric`.

Example: `d = dielectric('FR4');` `ant = fractalIsland('Substrate',d)`

Example: `d = dielectric('FR4');` `ant = fractalIsland;` `ant.Substrate = d;`

Data Types: `string` | `char`

`GroundPlaneLength` — Length of ground plane

`0.0500` (default) | positive scalar integer

Length of the ground plane, specified as a positive scalar integer in meters.

Example: `'GroundPlaneLength',0.0550`

Example: `ant.GroundPlaneLength = 0.0550`

Data Types: `double`

`GroundPlaneWidth` — Width of ground plane

`0.0300` (default) | positive scalar integer

Width of the ground plane, specified as a positive scalar integer in meters.

Example: `'GroundPlaneWidth',0.0550`

Example: `ant.GroundPlaneWidth = 0.0550`

Data Types: `double`

`FractalCenterOffset` — Signed distance of fractal center from origin

`[0 0]` (default) | two-element real-valued vector

Signed distance of the fractal center from the origin, specified as a two-element real-valued vector with each element unit in meters.

Example: `'FractalCenterOffset',[0 0.080]`

Example: `ant.FractalCenterOffset = [0 0.080]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.

- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Default Minkowski's Loop Fractal Antenna

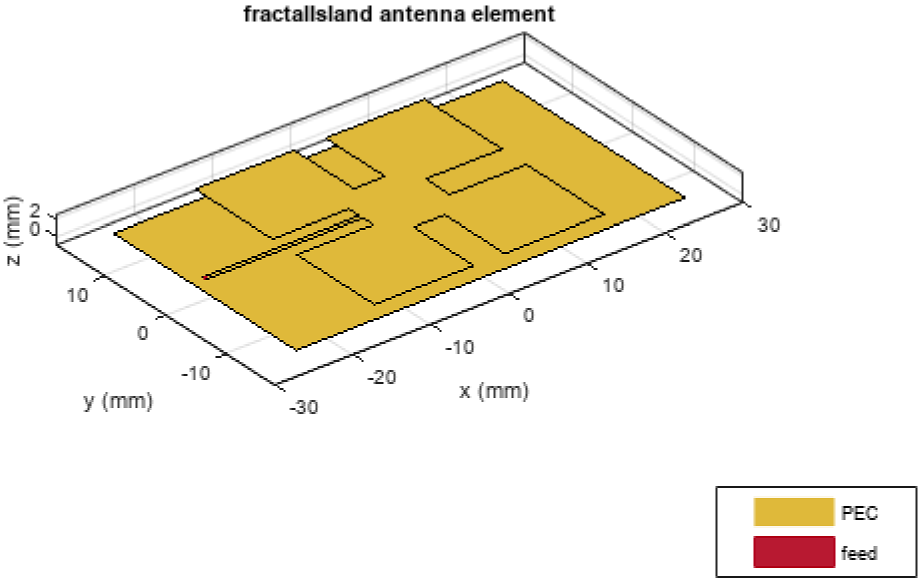
Create and view a Minkowski's loop fractal antenna with default property values.

```
ant = fractalIsland
```

```
ant = 
  fractalIsland with properties:

          NumIterations: 2
```

```
                 Length: 0.0295
                  Width: 0.0295
          StripLineWidth: 6.0000e-04
              SlotLength: 0.0040
               SlotWidth: 0.0040
                  Height: 0.0016
               Substrate: [1x1 dielectric]
         GroundPlaneLength: 0.0500
          GroundPlaneWidth: 0.0300
     FractalCenterOffset: [0 0]
               Conductor: [1x1 metal]
                    Tilt: 0
               TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
```

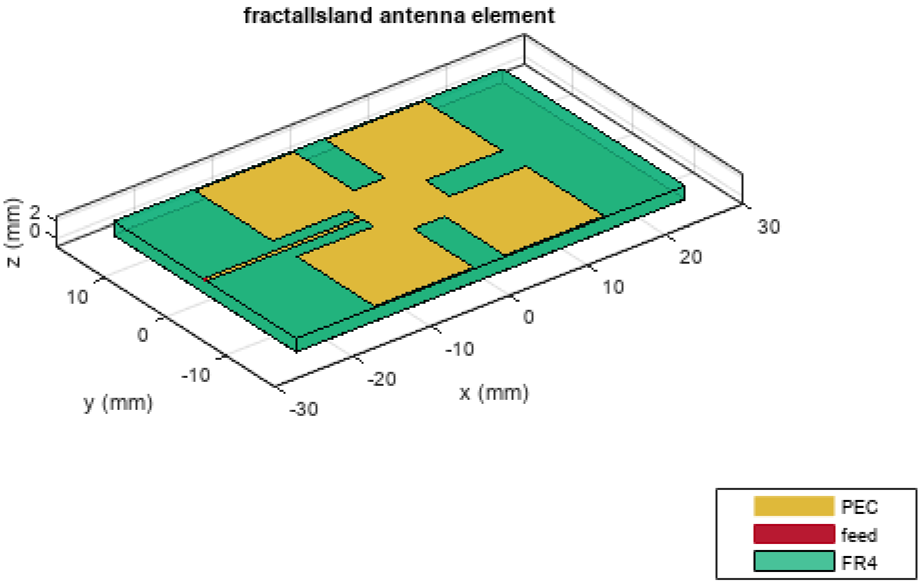
```
show(ant)
```



Radiation Pattern of Minkowski's Loop Fractal Antenna on FR4 Substrate

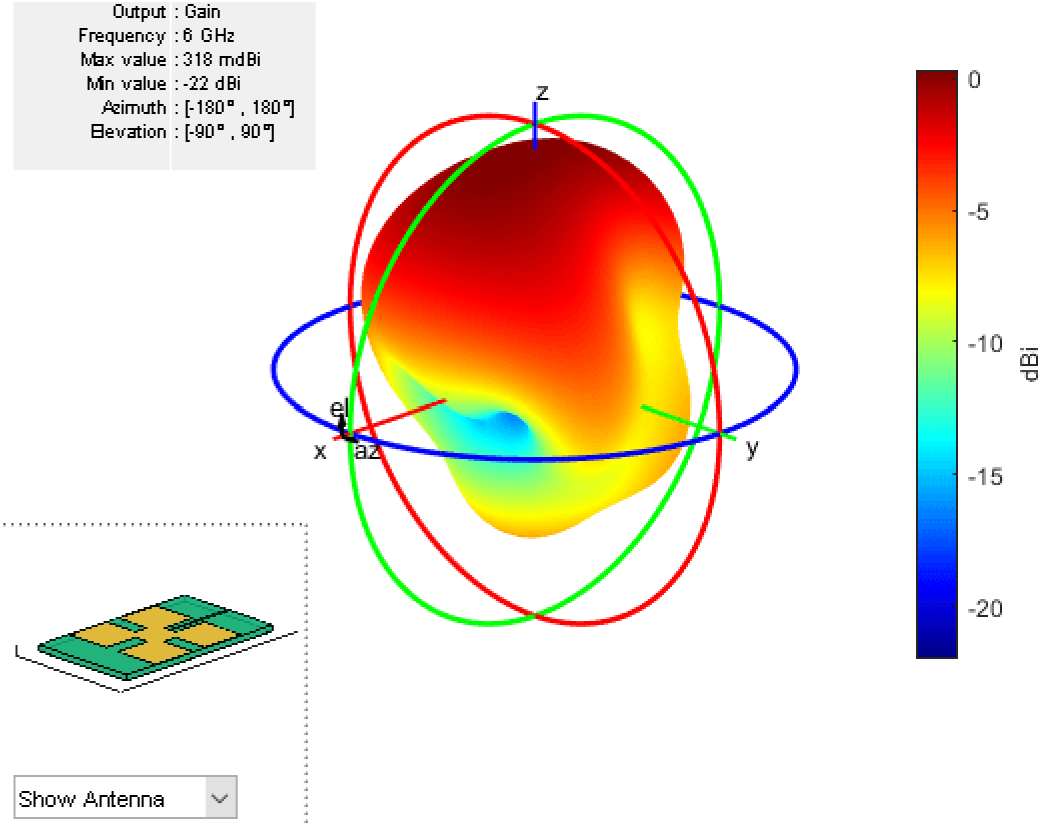
Create and view a Minkowski's loop fractal antenna on FR4 substrate.

```
ant = fractalIsland('Substrate',dielectric('FR4'));
show(ant)
```



Plot the radiation pattern of the antenna at a frequency of 6 GHz.

```
pattern(ant,6e9)
```



```
   Output : Gain
Frequency : 6 GHz
Max value : 318 mdBi
Min value : -22 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2019a

See Also

`fractalKoch` | `fractalGasket` | `fractalCarpet`

Topics

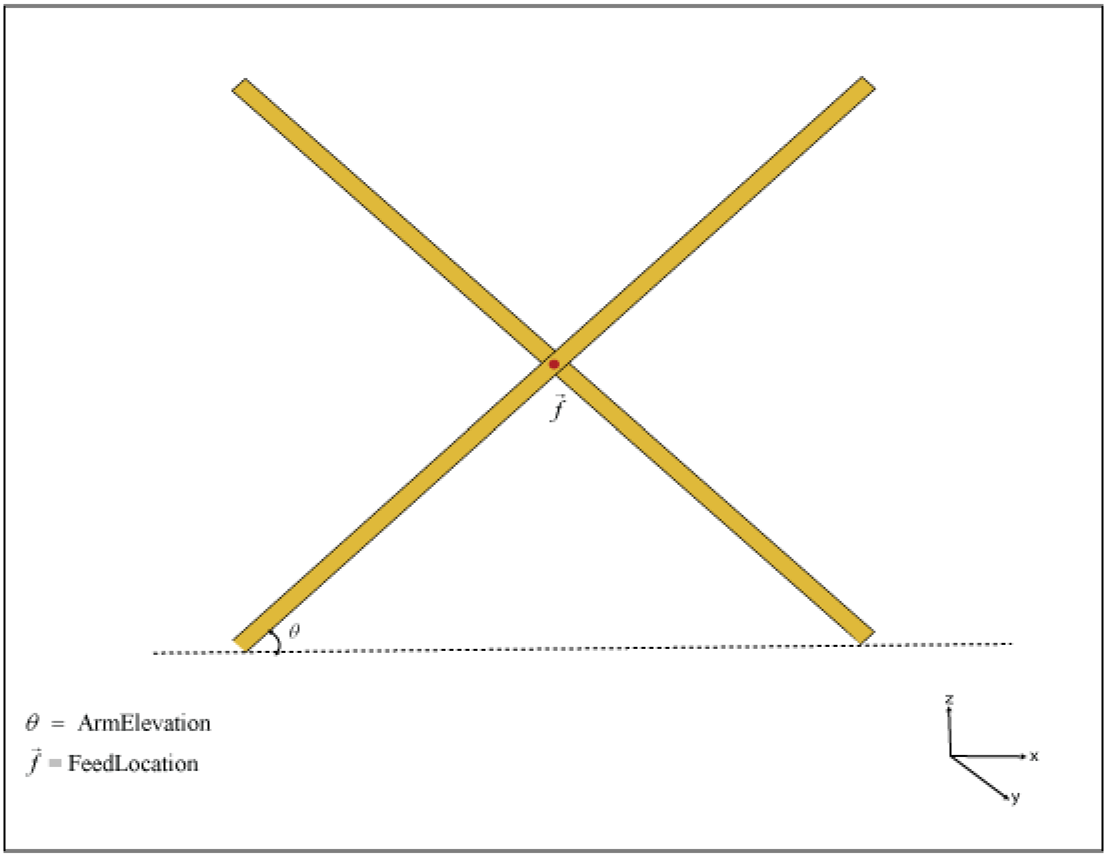
"Rotate Antennas and Arrays"

dipoleCrossed

Crossed dipole or turnstile antenna

Description

The `dipoleCrossed` object creates a turnstile antenna. By default, the turnstile antenna is center-fed and is on the Y-Z plane. This antenna operates at 6 GHz. You can also create a turnstile antenna using the following antenna elements: `bowtieTriangular`, `bowtieRounded`, and `dipoleBlade`.



Creation

Syntax

```
ant = dipoleCrossed
ant = dipoleCrossed(Name,Value)
```

Description

`ant = dipoleCrossed` creates a center-fed turnstile antenna operating at 6 GHz.

`ant = dipoleCrossed(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = dipoleCrossed('Element',dipoleBlade)` creates a turnstile antenna using a blade dipole antenna.

Properties

`Element` — Antenna element to create turnstile antenna

`dipole` (default) | antenna object

Antenna element to create a turnstile antenna, specified as an antenna object. You can also use the following antenna objects: `bowtieTriangular`, `bowtieRounded`, and `dipoleBlade`.

Example: `'Element',dipoleBlade`

Example: `ant.Element = dipoleBlade`

Data Types: `char` | `string`

`ArmElevation` — Angles made by antenna element arms

`[45 -45]` (default) | two-element signed vector

Angles made by the antenna element arms with respect to the X-Y plane, specified as a two-element signed vector.

Example: `'ArmElevation',[50 -60]`

Example: `ant.ArmElevation = [50 -60]`

Data Types: `double`

`FeedVoltage` — Magnitude of voltage applied to feeds

`[1 1]` (default) | two-element vector

Magnitude of voltage applied to the feeds, specified as a two-element vector with each element in volts.

Example: `'FeedVoltage',[2 2]`

Example: `ant.FeedVoltage = [2 2]`

Data Types: `double`

`FeedPhase` — Phase shift applied to voltage at feeds

`[0 90]` (default) | two-element vector

Phase shift applied to the voltage at the feeds, specified as a two-element vector with each element in degrees.

Example: `'FeedPhase',[0 50]`

Example: `ant.FeedPhase = [0 50]`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array

rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

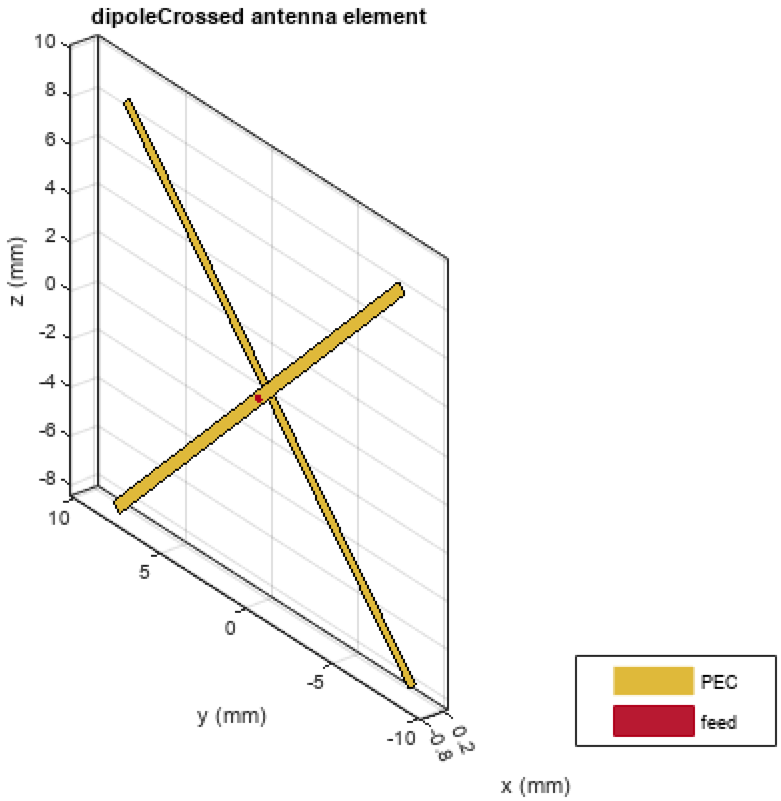
Default Crossed Dipole Antenna

Create and view a crossed dipole antenna with default property values.

```
ant = dipoleCrossed
```

```
ant = 
  dipoleCrossed with properties:

         Element: [1x1 dipole]
    ArmElevation: [45 -45]
     FeedVoltage: [1 1]
       FeedPhase: [0 90]
            Tilt: 0
        TiltAxis: [1 0 0]
```

```
show(ant)
```



Version History

Introduced in R2019a

See Also

`bowtieTriangular` | `bowtieRounded` | `dipoleBlade`

Topics

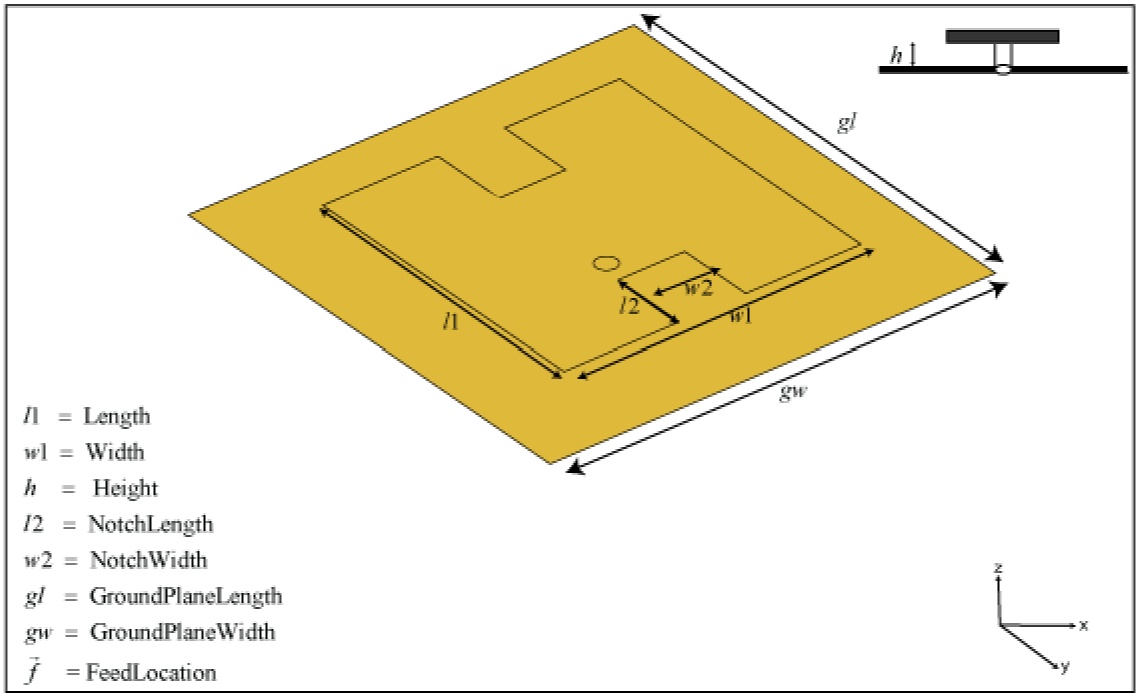
"Rotate Antennas and Arrays"

patchMicrostripHnotch

H-shaped microstrip patch antenna

Description

Use the `patchMicrostripHnotch` object to create an H-shaped microstrip patch antenna. The default patch is centered at the origin with the feedpoint along the length. By default, the dimensions are chosen for an operating frequency of 3.49 GHz for air or 2.61 GHz for Teflon.



Creation

Syntax

```
ant = patchMicrostripHnotch
ant = patchMicrostripHnotch(Name,Value)
```

Description

`ant = patchMicrostripHnotch` creates an H-shaped microstrip patch antenna.

`ant = patchMicrostripHnotch(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = patchMicrostripHnotch('Width',0.2)` creates a microstrip H-patch with a patch width of 0.2 m. Enclose each property name in quotes.

Properties

`Length` — Patch length along *x*-axis

`0.0290` (default) | scalar

Patch length along the *x*-axis, specified as a scalar in meters.

Example: `'Length',0.0450`

Example: `ant.Length = 0.0450`

Data Types: `double`

`Width` — Patch width along *y*-axis

`0.0300` (default) | scalar

Patch width along the *y*-axis, specified as a scalar in meters.

Example: `'Width',0.0500`

Example: `ant.Width = 0.0500`

Data Types: `double`

`NotchLength` — Notch length along *x*-axis

`0.0065` (default) | scalar

Notch length along the *x*-axis, specified as a scalar in meters.

Example: `'NotchLength',0.0200`

Example: `ant.NotchLength = 0.0200`

Data Types: `double`

`NotchWidth` — Notch width along *y*-axis

`0.0076` (default) | scalar

Notch width along the *y*-axis, specified as a scalar in meters.

Example: `'NotchWidth',0.00600`

Example: `ant.NotchWidth = 0.00600`

Data Types: `double`

`Height` — Patch height above ground plane along *z*-axis

`0.0016` (default) | scalar

Patch height above the ground plane along the *z*-axis, specified as a scalar in meters.

Example: `'Height',0.00500`

Example: `ant.Height = 0.00500`

Data Types: `double`

`Substrate` — Type of dielectric material

`'Air'` (default) | `dielectric` object

Type of dielectric material used as a substrate, specified as a dielectric object. For more information see, `dielectric`.

Example: `d = dielectric('FR4');` `ant = patchMicrostripHnotch('Substrate',d)`

Example: `d = dielectric('FR4');` `ant = patchMicrostripHnotch;` `ant.Substrate = d;`

Data Types: `string` | `char`

`GroundPlaneLength` — Ground plane length along *x*-axis

`0.0435` (default) | scalar

Ground plane length along the *x*-axis, specified as a scalar in meters. Setting the ground plane length to `Inf` uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',120e-3`

Example: `ant.GroundPlaneLength = 120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

`0.0450` (default) | scalar

Ground plane width along the *y*-axis, specified as a scalar in meters. Setting the ground plane width to `Inf` uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',120e-3`

Example: `ant.GroundPlaneWidth = 120e-3`

Data Types: `double`

`PatchCenterOffset` — Signed distance of patch from origin

`[0 0]` (default) | two-element real-valued vector

Signed distance of the patch from the origin, specified as a two-element real-valued vector with each element unit in meters. Use this property to adjust the location of the patch relative to the ground plane. Distances are measured along the length and width of the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Example: `ant.PatchCenterOffset = [0.01 0.01]`

Data Types: `double`

`FeedOffset` — Signed distance of feed from origin

`[–0.0025 -0.0050]` (default) | two-element real-valued vector

Signed distance of the feed from the origin, specified as a two-element real-valued vector with each element unit in meters. Use this property to adjust the location of the feedpoint relative to the ground plane and patch. Distances are measured along the length and width of the ground plane.

Example: `'FeedOffset',[0.01 0.01]`

Example: `ant.FeedOffset = [0.01 0.01]`

Data Types: `double`

`FeedDiameter` — Feed diameter

`1.0000e-03` (default) | scalar

Feed diameter, specified as a scalar in meters.

Example: `'FeedDiameter',0.0600`

Example: `ant.FeedDiameter = 0.0600`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see "Meshing".

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90],TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

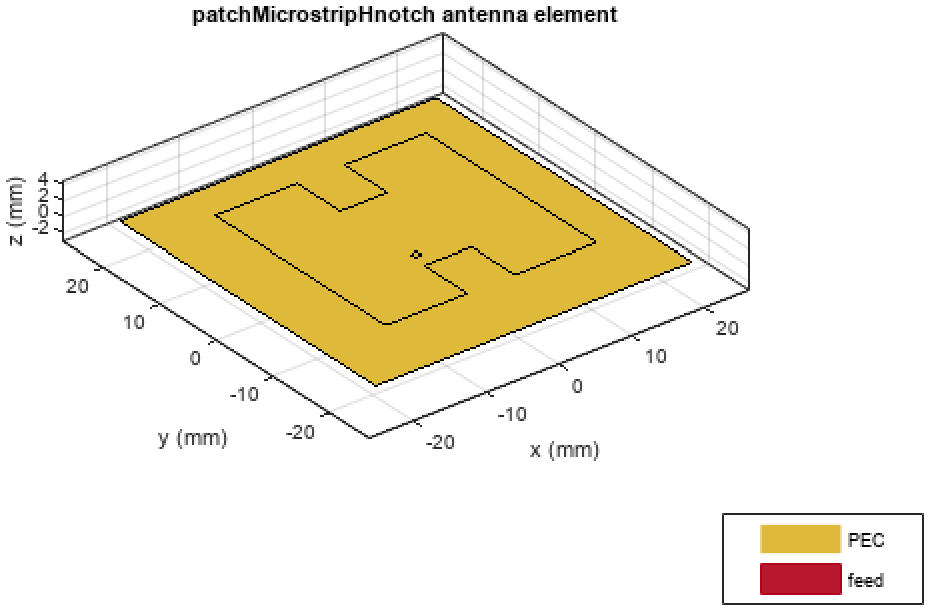
show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Default Microstrip Patch H-Notch

Create and view a microstrip patch H-notch with default property values.

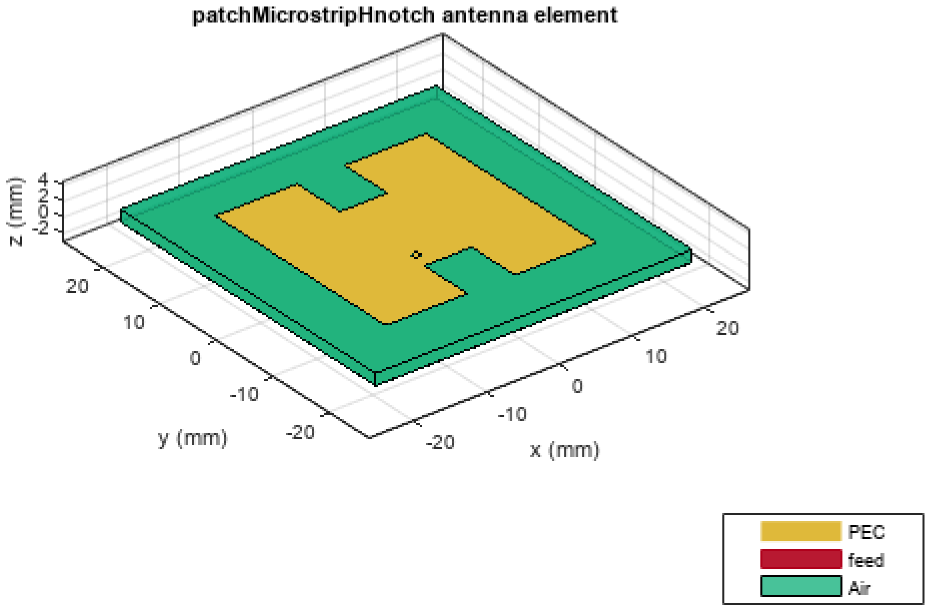
```
ant = patchMicrostripHnotch;
show(ant)
```



Microstrip Patch H-Notch with Dielectric Substrate

Create an H-shaped patch with dielectric substrate of permittivity 2.33.

```
ant = patchMicrostripHnotch('Substrate',dielectric('EpsilonR',2.33,'LossTangent',0.0012));
show(ant);
```



Version History

Introduced in R2019a

See Also

`patchMicrostrip` | `patchMicrostripCircular` | `patchMicrostripTriangular`

Topics

"Rotate Antennas and Arrays"

installedAntenna

Installed antenna setup

Description

The `installedAntenna` object creates an installed antenna setup that enables you to mount antennas on a platform for analysis.

Installed antenna analysis involves an electrically large structure called a platform. Around this platform, different antenna elements are placed. You can analyze the effects of the platform on the antenna performance. Installed antenna analysis is commonly used in aerospace, defense, and auto applications. The platforms in this case are planes, ships, or inside the bumper of a car.

Another common application of installed antenna analysis is to determine the interference of different antennas placed on a large platform.

Note `installedAntenna` only models pure metal structures.

Creation

Syntax

```
ant = installedAntenna
ant = installedAntenna(Name,Value)
```

Description

`ant = installedAntenna` creates an installed antenna setup. The default setup has a rectangular reflector in the X-Y plane as the platform with a dipole as the antenna. The dimensions of the dipole antenna are chosen for an operating frequency of 1 GHz.

`ant = installedAntenna(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = installedAntenna('Element',monopole)` creates an installed antenna setup using monopole as the antenna.

Output Arguments

`ant` — Installed antenna setup
`installedAntenna` object

Installed antenna setup, returned as an `installedAntenna` object.

Properties

`Platform` — Platform object file
`platform` object

Platform object file, specified as a `platform` object.

Example: `plat = platform('FileName','plate.stl'); ant = installedAntenna('Platform',plat)` This code creates a `platform` object called `plat` and uses it for installed antenna analysis.

Example: `plat = platform('FileName','plate.stl'); ant = installedAntenna;ant.Platform = plat` This code creates a `platform` object called `plat` and uses it for installed antenna analysis.

Data Types: `char`

`Element` — Single or multiple antenna elements

antenna object | vector of antenna objects

Single or multiple antennas, specified as an antenna object or a vector of antenna objects.

Example: `d = dipole; ant = installedAntenna('Element',d)` This code creates a `dipole` antenna object and uses it for installed antenna analysis.

Example: `d = dipole; ant = installedAntenna;ant.Element=d` This code creates a `dipole` antenna object and uses it for installed antenna analysis.

Example: `ant = installedAntenna('Element',{discone,monocone},'ElementPosition',[0.1 0.1 0.5; -0.1 -0.1 0.5])` This code creates `discone` and `monocone` antenna objects for installed antenna analysis.

Data Types: `char`

`ElementPosition` — Position of feed or origin of each antenna element

`[0 0 0.0750]` (default) | vector of [x,y,z] coordinates

Position of the feed or the origin of each antenna element, specified as a vector of [x,y,z] coordinates with each element unit in meters.

Example: `'ElementPosition',[0 0 0.0050]`

Example: `ant.ElementPosition = [0 0 0.0050]`

Data Types: `double`

`Reference` — Reference for positioning antenna elements

`'feed'` (default) | `'origin'`

Reference for positioning the antenna elements, specified as `'feed'` or `'origin'`.

Example: `'Reference','origin'`

Example: `ant.Reference = 'origin'`

Data Types: `string`

`FeedVoltage` — Excitation amplitude for antenna elements

`1` (default) | vector

Excitation amplitude for the antenna elements, specified as a scalar vector in volts.

Example: `'FeedVoltage',2`

Example: `ant.FeedVoltage = 2`

Data Types: `double`

FeedPhase — Phase shift of each antenna element

0 (default) | vector

Phase shift of each antenna element, specified as a scalar or vector in degrees.

Example: 'FeedPhase',50

Example: ant.FeedPhase = 50

Data Types: double

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: TiltAxis=[0 1 0]

Example: TiltAxis=[0 0 0;0 1 0]

Example: TiltAxis = 'Z'

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

SolverType — Solver for antenna analysis

'MoM-PO' (default) | 'MoM' | 'FMM'

Solver for antenna analysis, specified as the comma-separated pair consisting of 'SolverType' and 'MoM-PO' or 'MoM' (Method of Moments) or 'FMM' (Fast Multipole Method).

Example: `'SolverType','MOM'`

Data Types: `char`

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

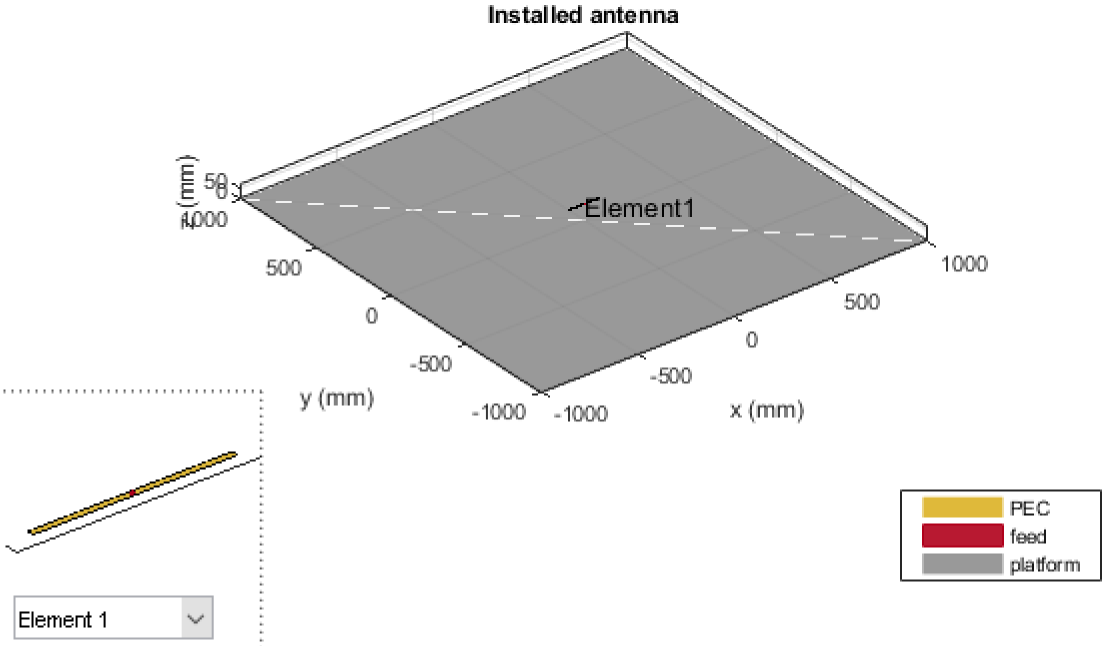
Installed Antenna Setup and Analysis

Create a default installed antenna.

```
ant = installedAntenna
```

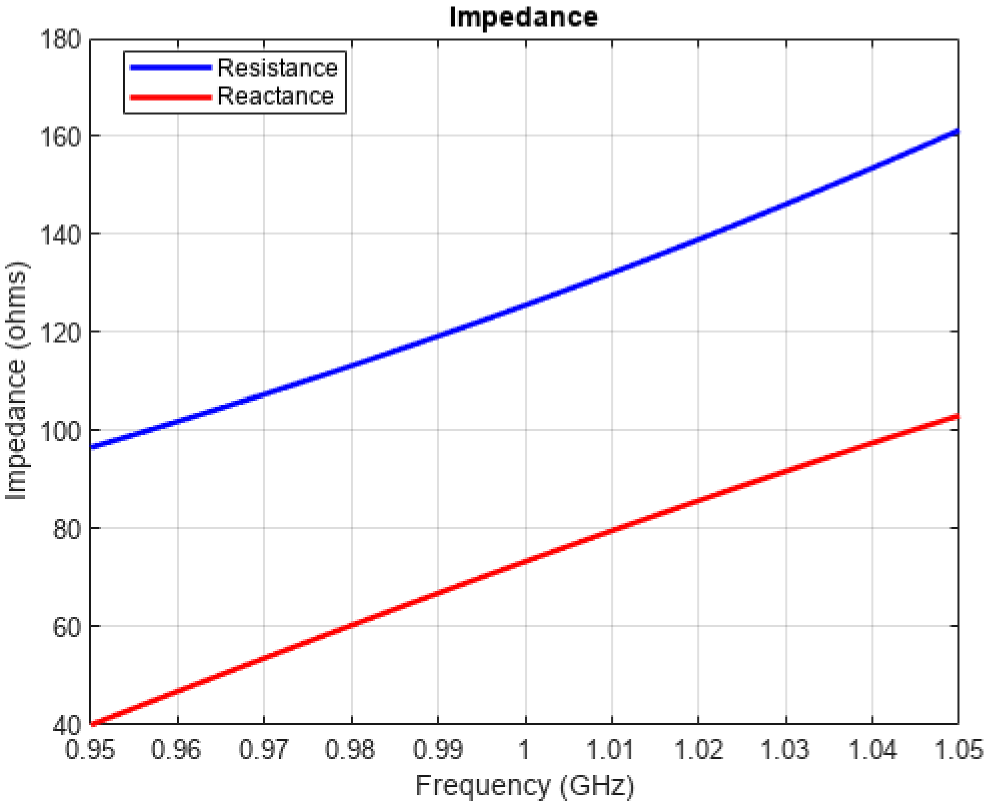
```
ant = 
  installedAntenna with properties:

           Platform: [1x1 platform]
            Element: [1x1 dipole]
    ElementPosition: [0 0 0.0750]
          Reference: 'feed'
        FeedVoltage: 1
          FeedPhase: 0
               Tilt: 0
           TiltAxis: [1 0 0]
         SolverType: 'MoM-PO'
```

```
show(ant);
```



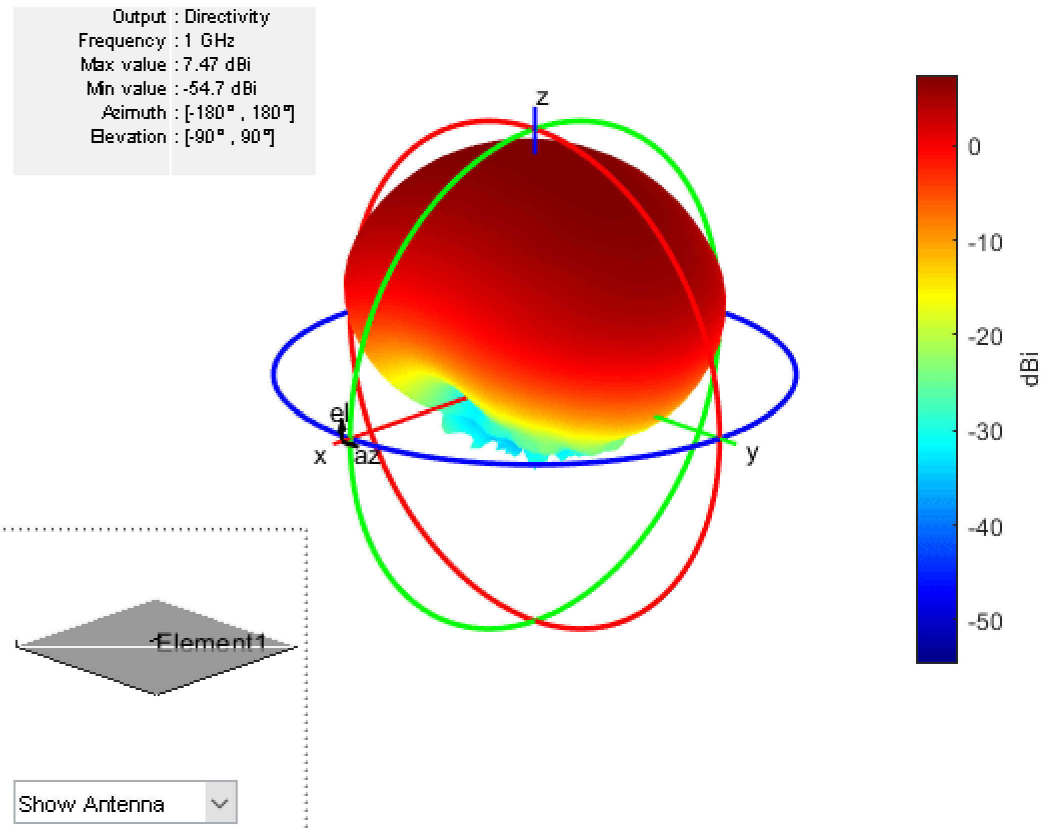
Calculate the impedance of the antenna.

```
figure;
impedance(ant, linspace(950e6, 1050e6, 51));
```



Visualize the pattern of the antenna.

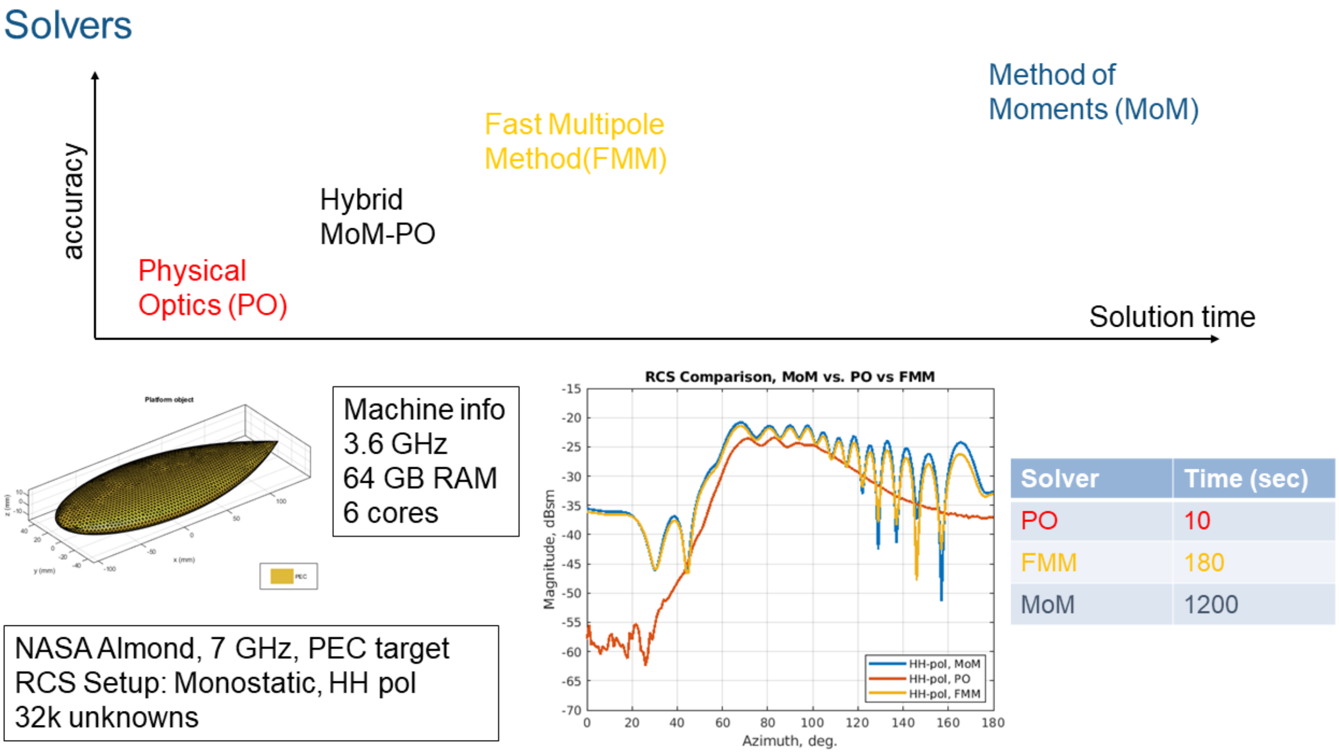
```
figure;
pattern(ant, 1e9);
```



```
   Output : Directivity
Frequency : 1 GHz
Max value : 7.47 dBi
Min value : -54.7 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Algorithms

Solver Recommendations for Installed Antenna Analysis



Solvers

Machine info
3.6 GHz
64 GB RAM
6 cores

NASA Almond, 7 GHz, PEC target
RCS Setup: Monostatic, HH pol
32k unknowns

Solver	Time (sec)
PO	10
FMM	180
MoM	1200

Hybrid MoM PO Solver

The default solver for installed antenna analysis is the hybrid MoM-PO (Method of moments and physical optics) solver. This is a hybrid solver, with less stringent requirements on the mesh. This approach does not have full-wave accuracy since the electrically large portion of the geometry is handled using the physical optics approach. For more information on this solver see, “Hybrid MoM-PO Method for Metal Antennas with Large Scatterers”.

FMM Solver

For full wave accuracy in installed antenna analysis you can use the FMM (fast multiple method) solver. This solver does not fill and store a matrix of interactions and enables the solution of large structures which is not possible when using the MoM solver. For open geometries, this solver builds a preconditioner matrix internally. The preconditioner matrix is sparse but might be less sparse for larger structures. Preconditioner matrices are not built for closed geometries. When using the FMM, you have an option to use the solver object to set up the number of iterations and the relative error. In some circumstances it might be useful to study the nature of the problem and its convergence characteristics by reducing the number of iterations or the relative error initially or both. The FMM solver does need approximately 10 elements per wavelength in the mesh. The number of elements does impact the convergence of the solution. For more information on FMM solvers, see “Fast Multipole Method for Large Structures”.

MoM Solver

For wavelength and sub-wavelength scale structures, with or without dielectric you can use an MoM (method of moments)solver. For more information on MoM solver, see “Method of Moments Solver for Metal Structures” and “Method of Moments Solver for Metal and Dielectric Structures”.

Version History

Introduced in R2019a

See Also

`platform`

Topics

“Rotate Antennas and Arrays”

platform

Create platform object for installed antenna setup

Description

The `platform` object creates a platform to be used in an installed antenna setup.

Installed antenna analysis involves an electrically large structure called a platform. Around this platform different antenna elements are placed. You can analyze the effects of the platform on the antenna performance. Installed antenna analysis is commonly used in aerospace, defense, and auto applications. The platforms in this case are planes, ships, or inside the bumper of a car.

Another common application of installed antenna analysis is to determine the interference of different antennas placed on a large platform.

Creation

Syntax

```
plat = platform
plat = platform(Name,Value)
```

Description

`plat = platform` creates a platform object for an installed antenna setup. The default platform is a rectangular reflector in the X-Y plane stored in the `plate.stl` file.

`plat = platform(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = platform('FileName','reflector.stl')` creates a platform object defined by the data in the file `reflector.stl`

Output Arguments

`plat` — Platform for installed antenna setup
`platform` object

Platform for installed antenna setup, returned as a `platform` object.

Properties

`FileName` — STL file defining platform
`'[]'` (default) | string array | character vector

STL file defining the platform, specified as a string or a character vector.

Example: `plat = platform('FileName','reflector.stl')` creates a platform with file name `reflector.stl`.

Example: `plat = platform; plat.FileName = 'reflector.stl'` creates a platform with file name `reflector.stl`.

Data Types: `char` | `string`

`Units` — Units for STL file

`'mm'` (default) | string | character

Units for the STL file, specified as a string array or character vector.

Example: `plat = platform('Units','m')` Creates a platform with STL file units in meters.

Example: `plat = platform;plat.Units = 'm'` Creates a platform with STL file units in meters.

Data Types: `char` | `string`

`UseFileAsMesh` — STL file used as the mesh for analysis

`'0'` (default) | '1' | string array | character vector

Use the .stl file directly as the mesh for analysis

Example: `plat = platform('UseFileAsMesh','1)`. Uses the .stl file in the `FileName` property directly as a mesh..

Example: `plat = platform; plat.UseFileAsMesh = '1'` . Uses the .stl file in the `FileName` property directly as a mesh..

Data Types: `logical`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
stlwrite	Write mesh to STL file

Examples

Platform from STL of DipoleHelix Antenna

Create a `DipoleHelix` antenna object at 2 GHz and compute the impedance.

```
w = design(dipoleHelix,2e9);
Z = impedance(w,2e9);
```

Create an STL file for `DipoleHelix` antenna object .

```
stlwrite(w,'dipoleHelix_2GHz.stl')
```

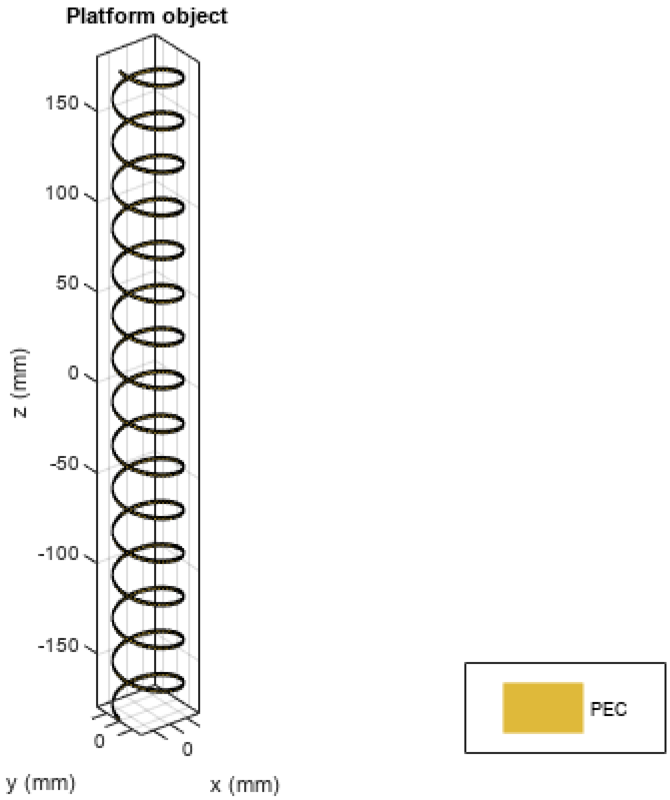
You will see the `dipoleHelix_2GHz.stl` file in your current folder.

Load `dipoleHelix_2GHz.stl` and visualize the platform.

```
plat = platform('FileName','dipoleHelix_2GHz.stl','Units','m')
```

```
plat = 
  platform with properties:

         FileName: 'dipoleHelix_2GHz.stl'
            Units: 'm'
    UseFileAsMesh: 0
             Tilt: 0
         TiltAxis: [1 0 0]
```

```
show(plat)
```



Version History

Introduced in R2019a

See Also

`installedAntenna` | `stlwrite`

Topics

“Rotate Antennas and Arrays”

“Hybrid MoM-PO Method for Metal Antennas with Large Scatterers”

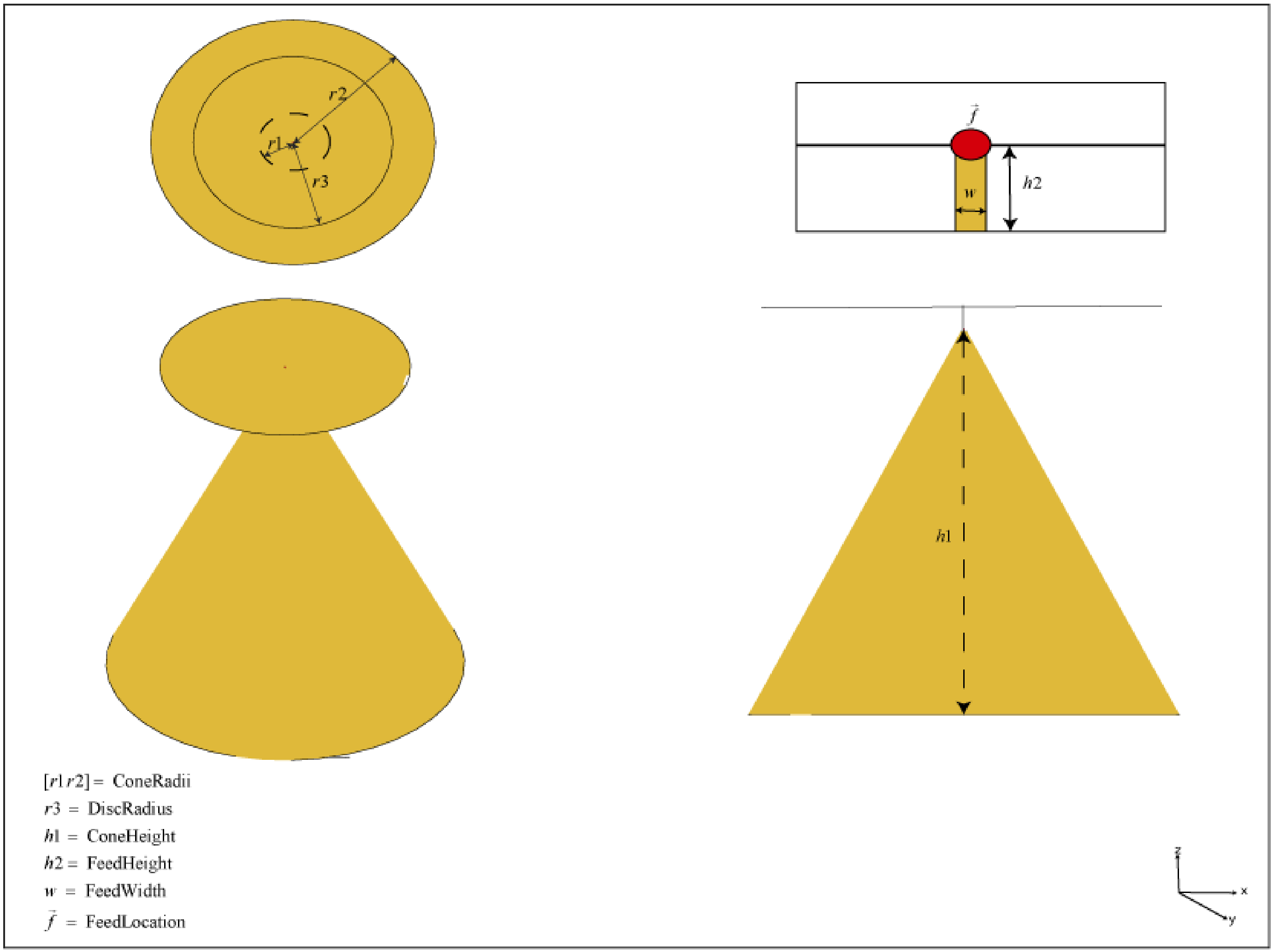
discone

Create discone antenna

Description

The `discone` object creates a discone antenna that consists of a circular disc and a cone whose apex approaches the center of the disc. A small gap exists between the disc and the cone through which the feed is connected.

A discone antenna is an omnidirectional vertically polarized antenna. This antenna has an exceptionally large coverage, offering a frequency range ratio of up to 10:1 between the upper cutoff frequency and the lower cutoff frequency. The discone antenna wideband coverage makes it useful in commercial, military, amateur radio, and radio scanner applications.



Creation

Syntax

```
ant = discone
ant = discone(Name,Value)
```

Description

`ant = discone` creates a discone antenna with dimensions for a resonant frequency of 2.12 GHz. The default discone has a feedpoint at the center of the disc.

`ant = discone(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = discone('Height',1)` creates a discone antenna with a cone of height 1 meter.

Properties

`Height` — Vertical height of cone
`0.0744` (default) | real-valued scalar

Vertical height of the cone from the center of the lower base of the cone to the center of the upper base of the cone, specified as a real-valued scalar in meters.

Example: `'Height',1`

Example: `ant.Height = 1`

Data Types: `double`

`ConeRadii` — Radii of cone
`[5.3300e-04 0.0426]` (default) | vector

Radii of the cone consisting of the broad radius and the narrow radius, specified as a vector with each element unit in meters. The first element of the vector is the narrow radius, and the second element of the vector is the broad radius.

Example: `'ConeRadii',[6.3300e-04 0.0546]`

Example: `ant.ConeRadii = [6.3300e-04 0.0546]`

Data Types: `double`

`DiscRadius` — Radius of disc
`0.0298` (default) | real-valued scalar

Radius of the disc, specified as a real-valued scalar in meters.

Example: `'DiscRadius',0.0050`

Example: `ant.DiscRadius = 0.050`

Data Types: `double`

`FeedHeight` — Gap between cone and disc
`3.1980e-04` (default) | real-valued scalar

Gap between the cone and the disc, specified as a real-valued scalar in meters.

Example: 'FeedHeight',0.0034

Example: ant.FeedHeight = 0.0034

Data Types: double

FeedWidth — Width of feed

4.2640e-04 (default) | real-valued scalar

Width of the feed, specified as a real-valued scalar in meters.

Example: 'FeedWidth',0.0050

Example: ant.FeedWidth = 0.0050

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see "Meshing".

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: ant.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

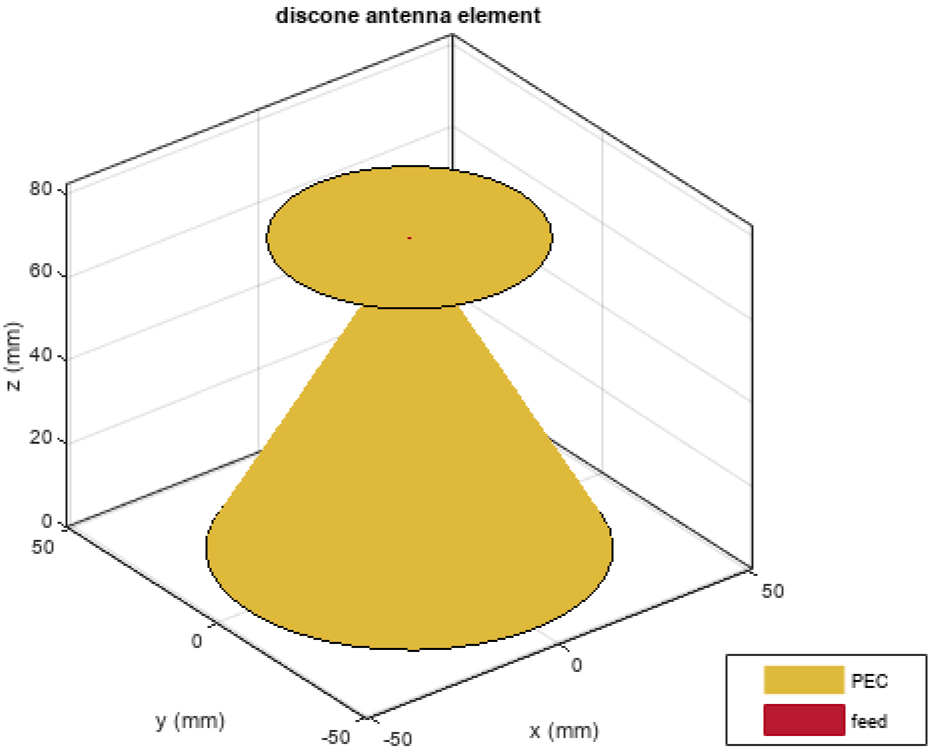
coneangle2size	Calculates equivalent cone height, broad radius, and narrow radius for cone
show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Default Discone Antenna and Radiation Pattern

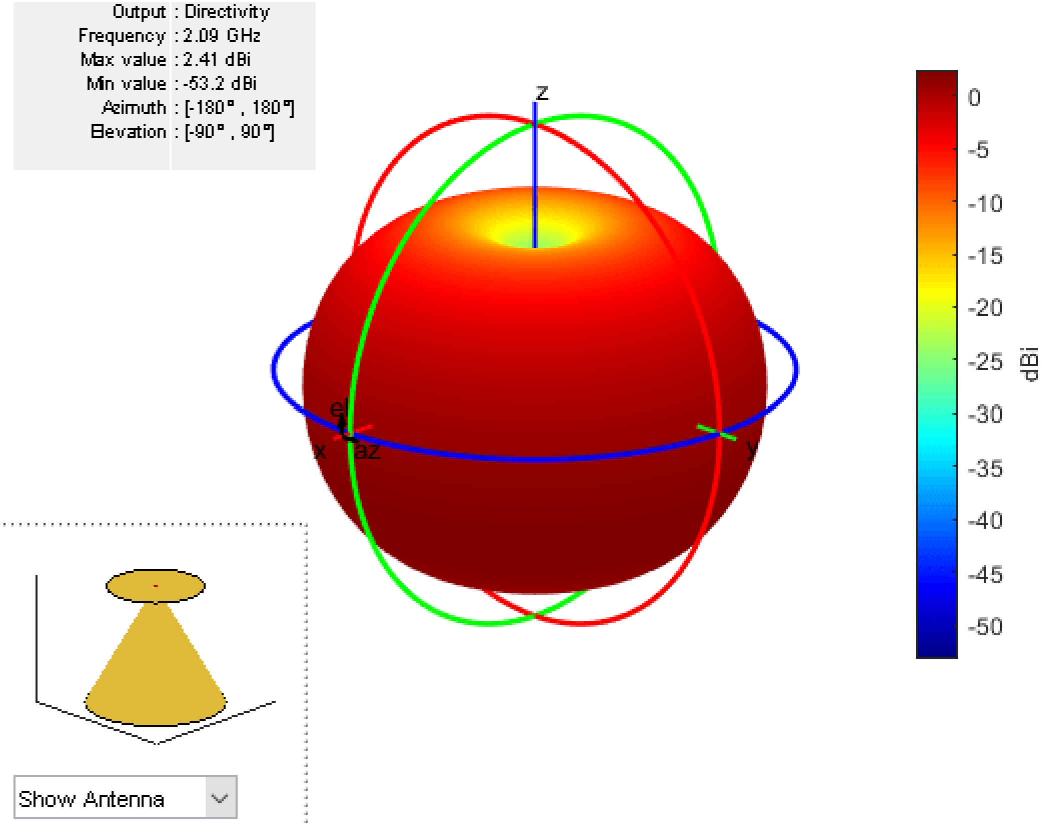
Create and view a default discone antenna.

```
ant = discone;
show(ant)
```



Plot the radiation pattern of the antenna at 2.09 GHz.

```
pattern(ant,2.09e9)
```

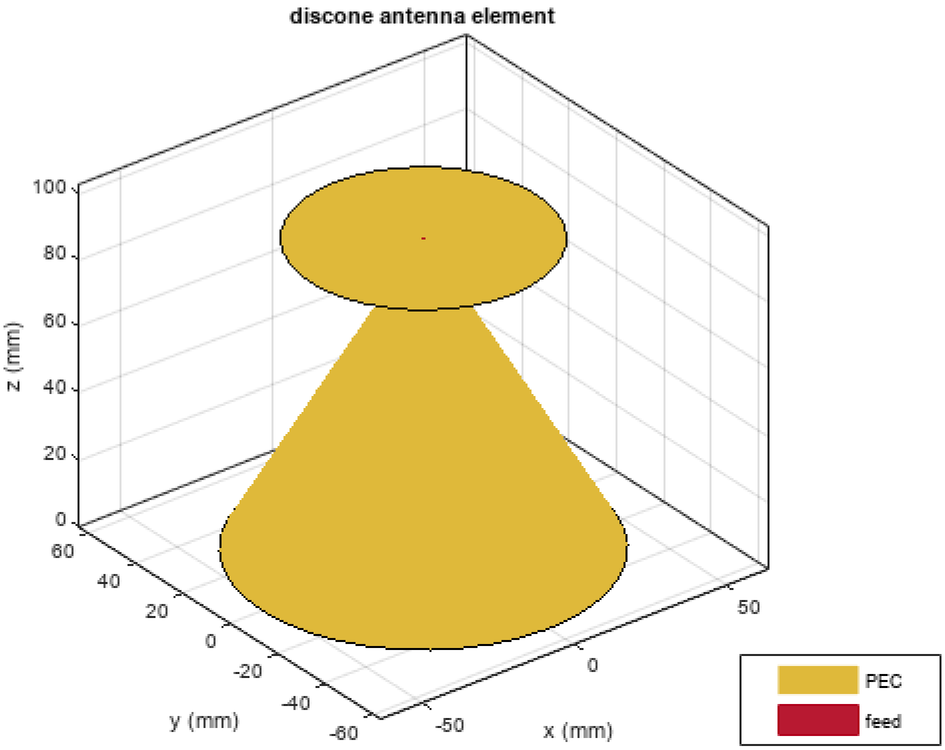


```
   Output : Directivity
Frequency : 2.09 GHz
Max value : 2.41 dBi
Min value : -53.2 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Impedance and Radiation Pattern of Custom Discone Antenna

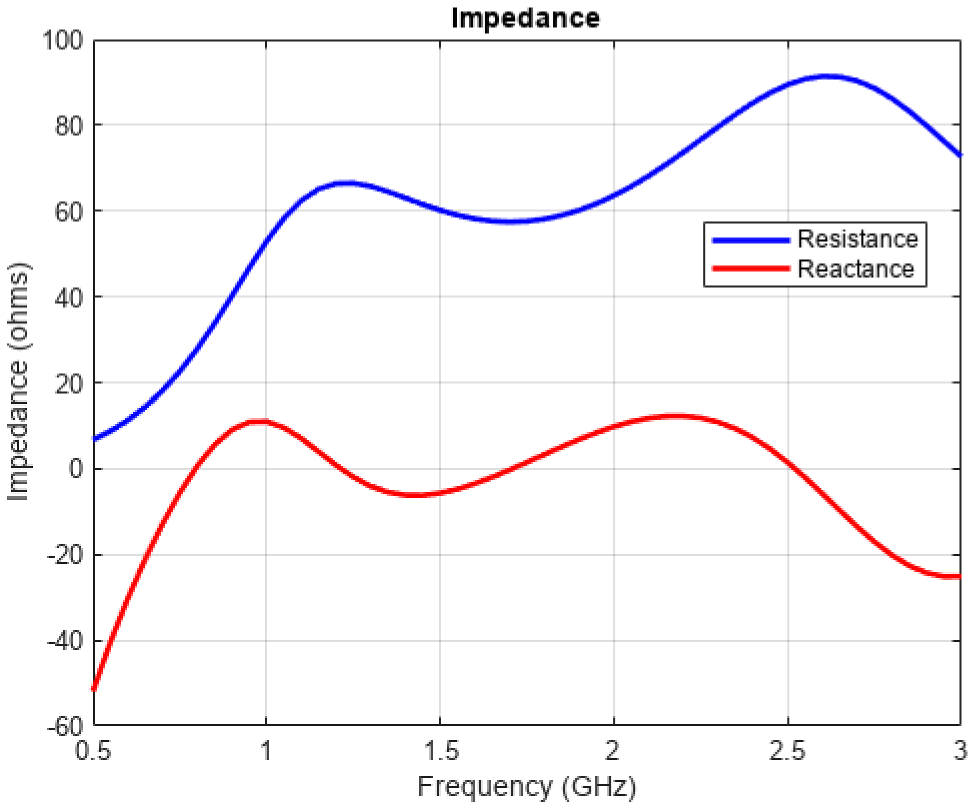
Create and view a discone antenna with specific dimensions.

```
ant = discone('Height',0.0925,'ConeRadii',[0.666e-3 53.2e-3],...
      'DiscRadius',37.25e-3,'FeedHeight',399.7e-6,'FeedWidth',0.553e-3);
show(ant)
```

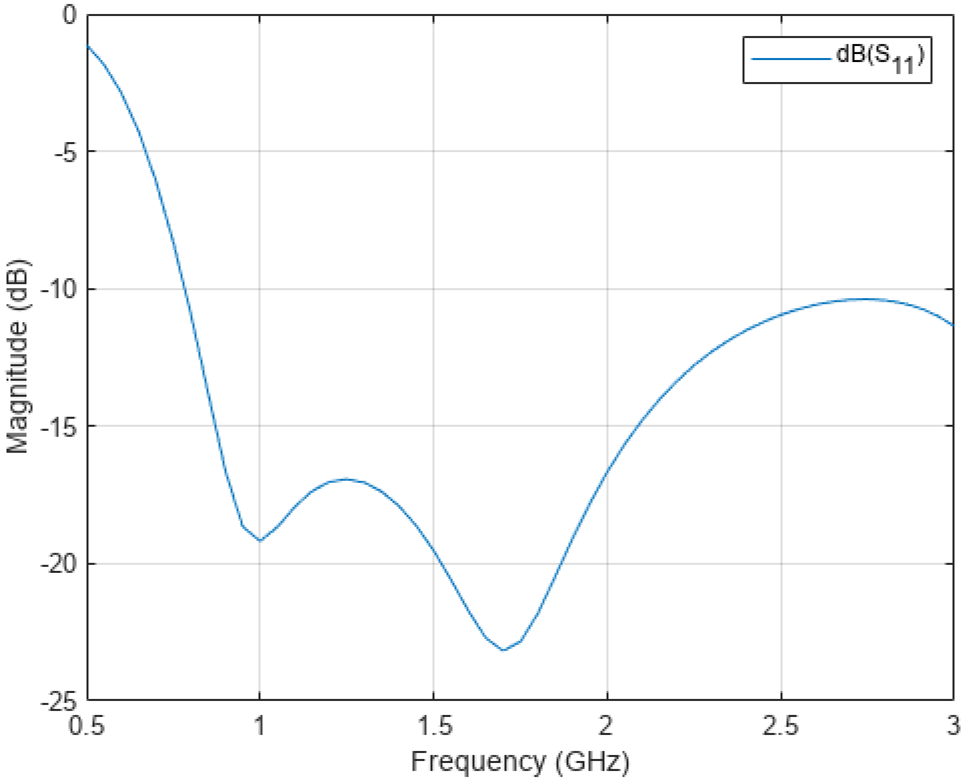


Calculate the impedance of the antenna over the frequency span of 500 MHz to 3 GHz and plot the S-parameters.

```
impedance(ant,linspace(0.5e9,3e9,51));
```

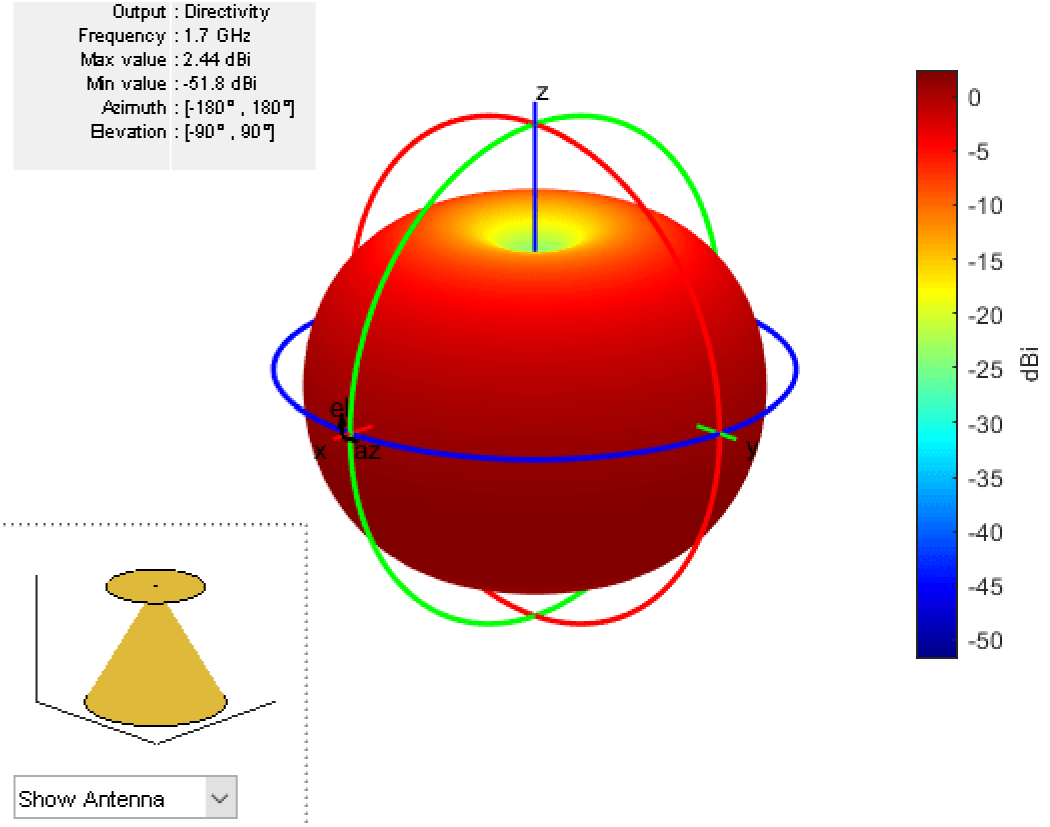


```
s = sparameters(ant,linspace(0.5e9,3e9,51));
figure;
rfplot(s);
```



Plot the radiation pattern of the antenna at 1.7 GHz.

```
pattern(ant,1.7e9);
```



```
   Output : Directivity
Frequency : 1.7 GHz
Max value : 2.44 dBi
Min value : -51.8 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2019b

References

[1] Verma, Saritha, Abhilash Mehta, and Rukhsana Khan. "Analysis of Variation of Various Parameters on Design of Discone Antenna." *Advanced Computational Techniques in Electromagnetics*. Volume 2012, 2012, pp.1-5.

See Also

`cavityCircular` | `bicone`

Topics

“Rotate Antennas and Arrays”

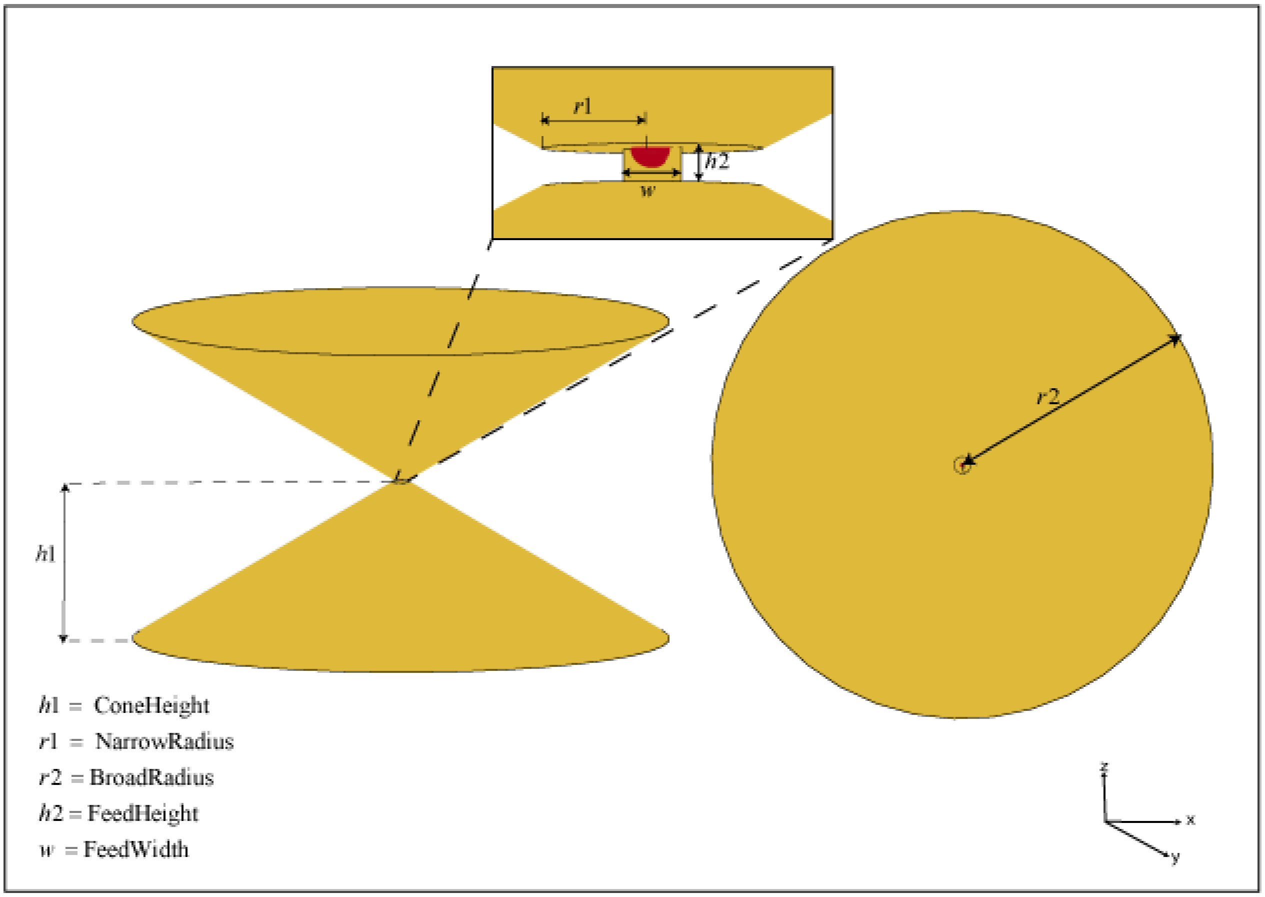
bicone

Create bicone antenna

Description

The `bicone` object creates a bicone antenna. A bicone antenna consists of two symmetrical or asymmetrical cones separated by a small gap. The feed spans the gap and connects both the cones.

Bicone antennas are broadband omnidirectional antennas used for electronic support measure (ESM) applications. Bicone antennas are often used in electromagnetic interference (EMI) testing for immunity testing or emissions testing.



Creation

Syntax

```
ant = bicone
ant = bicone(Name,Value)
```

Description

`ant = bicone` creates a bicone antenna with dimensions for a resonant frequency of 2.3 GHz. The default bicone has a feedpoint at the apex of the top cone.

`ant = bicone(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = bicone('Height',1)` creates a bicone antenna with a cone of height 1 meter.

Properties

`ConeHeight` — Vertical height of cones

`0.0215` (default) | real-valued scalar | two-element vector

Vertical height of the cones, specified as a real-valued scalar in meters or a two-element vector with each element unit in meters. A scalar value creates two cones of the same height. The two-element vector can create two cones of different heights. In the two-element vector, the first element specifies the height of the top cone, and the second element specifies the height of the bottom cone.

Example: `'ConeHeight',[0.0215 0.0315]`

Example: `ant.ConeHeight = [0.0215 0.0315]`

Data Types: `double`

`NarrowRadius` — Radius at apex of cones

`0.0013` (default) | real-valued scalar | two-element vector

Radius at the apex of the cones, specified as a real-valued scalar in meters or a two-element vector with each element unit in meters. A scalar value creates two cones with the same narrow radius. A two-element vector can create two cones with different narrow radii. In the two-element vector, the first element specifies the narrow radius of the top cone, and the second element specifies the narrow radius of the bottom cone.

Example: `'NarrowRadius',[6.3300e-04 0.0546]`

Example: `ant.NarrowRadius = [6.3300e-04 0.0546]`

Data Types: `double`

`BroadRadius` — Radius at broad opening of cones

`0.00385` (default) | real-valued scalar | two-element vector

Radius at the broad opening of the cones, specified as a real-valued scalar in meters or a two-element vector with each element unit in meters. A scalar value creates two cones with the same broad radius. A two-element vector can create two cones of different broad radii. In the two-element vector, the first element specifies the broad radius of the top cone, and the second element specifies the broad radius of the bottom cone.

Example: `'BroadRadius',[8.3300e-04 0.0846]`

Example: `ant.BroadRadius = [8.3300e-04 0.0846]`

Data Types: `double`

`FeedHeight` — Gap between two cones

`3.1980e-04` (default) | real-valued scalar

Gap between the two cones, specified as a real-valued scalar in meters.

Example: `'FeedHeight',0.0034`

Example: `ant.FeedHeight = 0.0034`

Data Types: `double`

`FeedWidth` — Width of feed

`4.2640e-04` (default) | real-valued scalar

Width of the feed, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0050`

Example: `ant.FeedWidth = 0.0050`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

coneangle2size	Calculates equivalent cone height, broad radius, and narrow radius for cone
show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

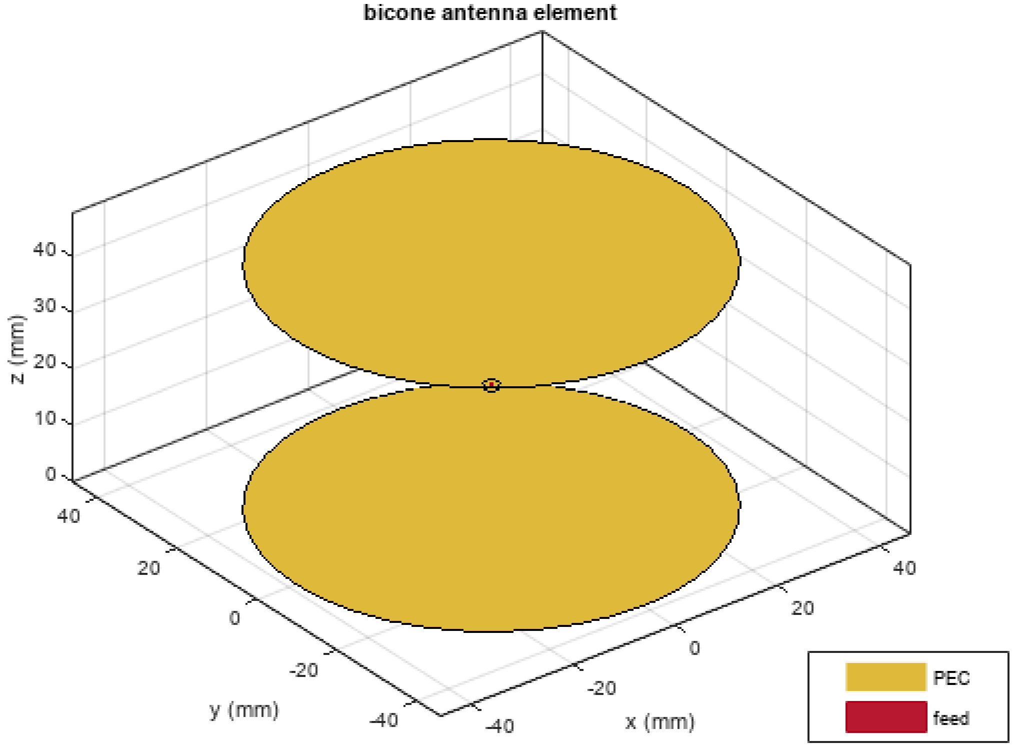
Examples

Default Bicone Antenna and Radiation Pattern

Create and view a default bicone antenna.

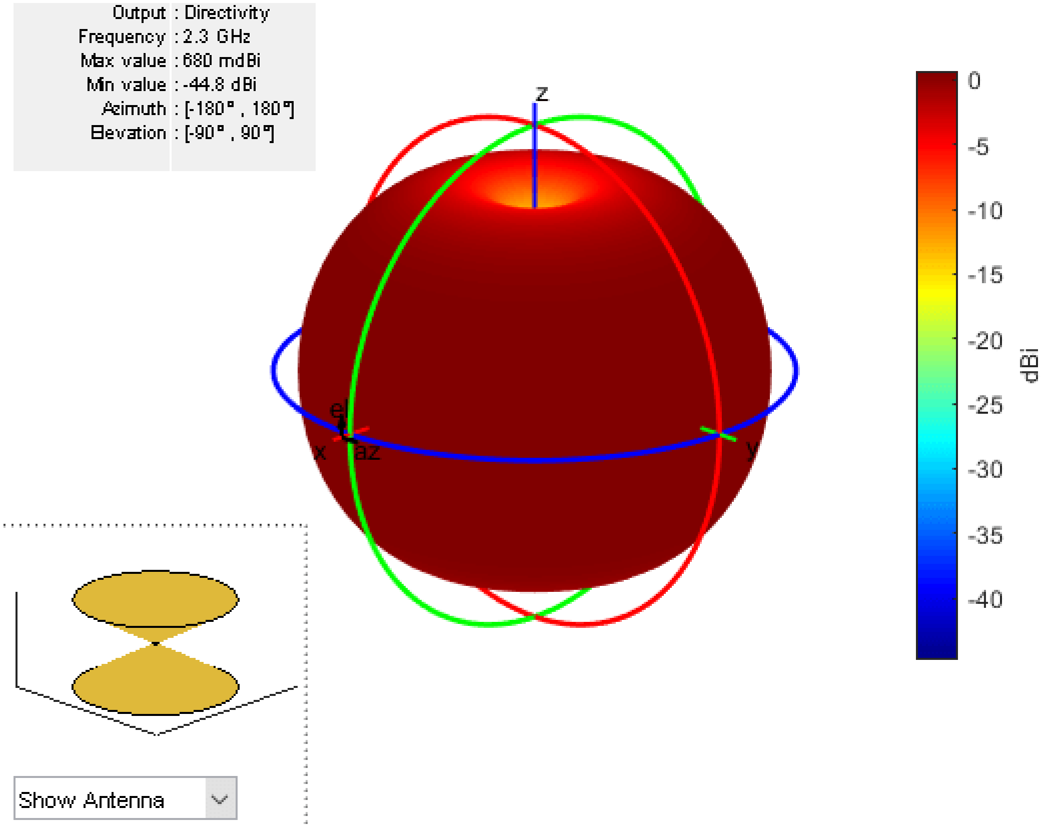
```
ant = bicone
ant = 
  bicone with properties:

      ConeHeight: 0.0215
    NarrowRadius: 0.0013
     BroadRadius: 0.0385
      FeedHeight: 5.0000e-04
       FeedWidth: 1.0000e-03
       Conductor: [1x1 metal]
            Tilt: 0
        TiltAxis: [1 0 0]
            Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 2.3 GHz.

```
pattern(ant,2.3e9)
```



```
Output : Directivity
Frequency : 2.3 GHz
Max value : 680 mdBi
Min value : -44.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

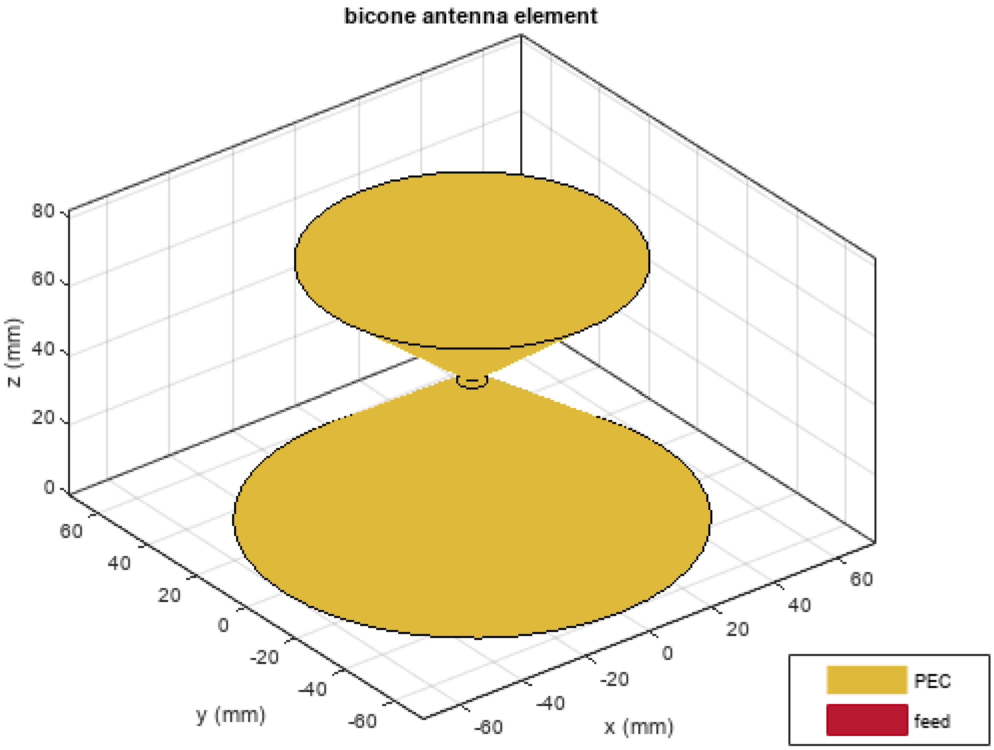
Impedance of Bicone Antenna with Asymmetrical Cones

Create a bicone antenna with asymmetrical cones.

```
ant = bicone('NarrowRadius',[2e-3 4e-3],'BroadRadius',...
           [44.7e-3,60e-3],'ConeHeight',[33.7e-3 40e-3],'FeedHeight',...
           1e-3,'FeedWidth',2e-3)
```

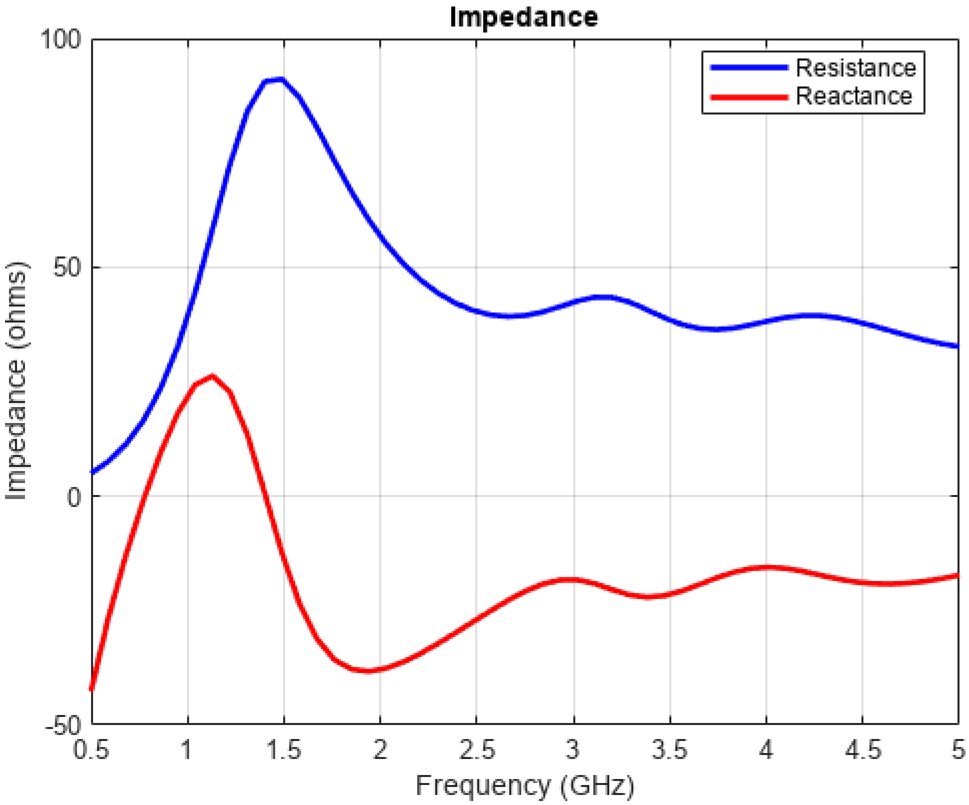
```
ant = 
  bicone with properties:

      ConeHeight: [0.0337 0.0400]
    NarrowRadius: [0.0020 0.0040]
     BroadRadius: [0.0447 0.0600]
      FeedHeight: 1.0000e-03
       FeedWidth: 0.0020
       Conductor: [1x1 metal]
            Tilt: 0
        TiltAxis: [1 0 0]
            Load: [1x1 lumpedElement]
```

```
show(ant)
```



Calculate the impedance of the antenna over the frequency span of 500 MHz - 5 GHz.

```
impedance(ant,linspace(0.5e9,5e9,51));
```



Version History

Introduced in R2019b

References

[1] Kudpik, Rapin & Komask Meksamoot, Nipapon Siripon, and Sompol Kosulvit. "Design of a Compact Biconical Antenna for UWB Applications." 10.1109/ISPACS.2011.6146212.

See Also

`cavityCircular` | `discone`

Topics

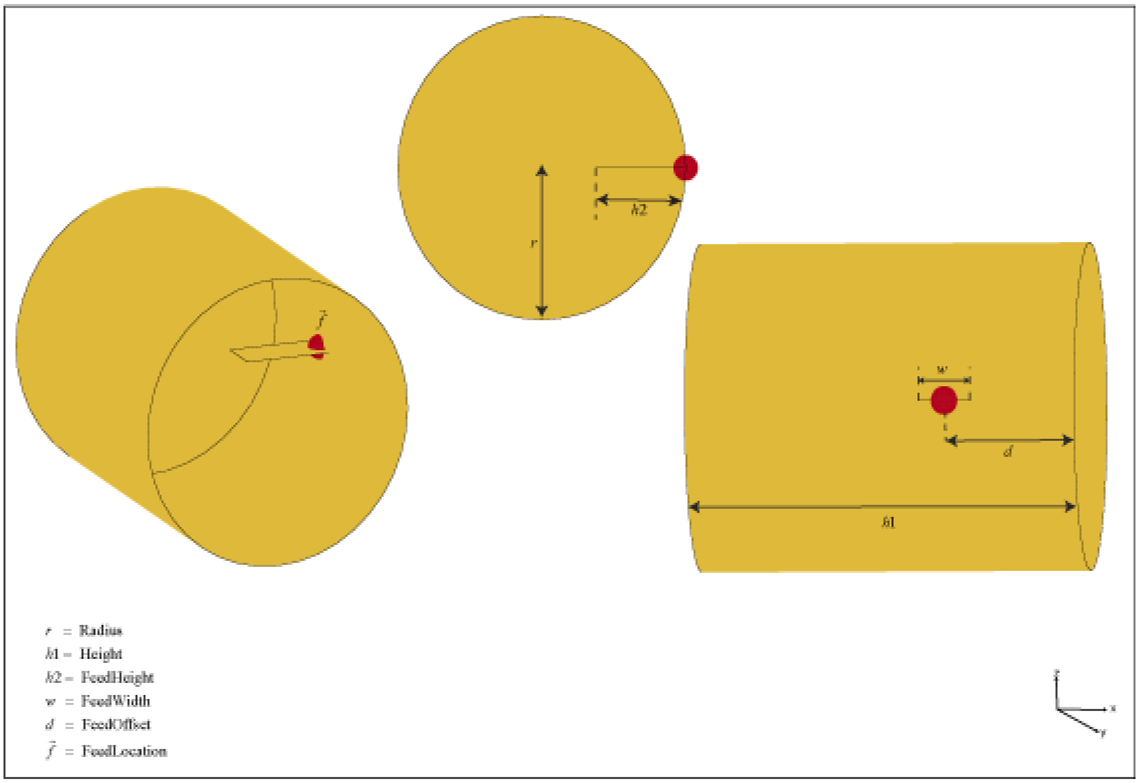
“Rotate Antennas and Arrays”

waveguideCircular

Create circular waveguide

Description

The `waveguideCircular` object creates a circular waveguide. A circular waveguide is a hollow tube of uniform cross section, that confines the electromagnetic wave. This antenna is used in radar and short and medium distance broadband communication.



Creation

Syntax

```
ant = waveguideCircular
ant = waveguideCircular(Name,Value)
```

Description

`ant = waveguideCircular` creates a circular waveguide with dimensions for an operating frequency of 8.42 GHz.

`ant = waveguideCircular(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = waveguideCircular('Height',1)` creates a circular waveguide with a height of 1 meter.

Properties

`Height` — Height of circular waveguide

`0.0300` (default) | real-valued scalar

Height of the circular waveguide, specified as a real-valued scalar in meters.

Example: `'Height',0.0215`

Example: `ant.Height = 0.0215`

Data Types: `double`

`Radius` — Radius of circular waveguide

`0.0120` (default) | real-valued scalar

Radius of the circular waveguide, specified as a real-valued scalar in meters.

Example: `'Radius',0.0546`

Example: `ant.Radius = 0.0546`

Data Types: `double`

`FeedHeight` — Height of feed

`0.0075` (default) | real-valued scalar

Height of the feed, which is equal to the height of the monopole, specified as a real-valued scalar in meters.

Example: `'FeedHeight',0.0034`

Example: `ant.FeedHeight = 0.0034`

Data Types: `double`

`FeedWidth` — Width of feed

`0.0040` (default) | real-valued scalar

Width of the feed, which is equal to the width of the monopole, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0050`

Example: `ant.FeedWidth = 0.0050`

Data Types: `double`

`FeedOffset` — Vertical distance of feed along Y-axis

`0.0100` (default) | real-valued scalar

Vertical distance of the feed along the Y-axis, specified as a real-valued scalar in meters.

Example: `'FeedOffset',0.0050`

Example: `ant.FeedOffset = 0.0050`

Data Types: double

Conductor — Type of metal material
'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements
[1x1 lumpedElement] (default) | lumped element object handle

Lumped elements added to the antenna feed, specified as a lumped element object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object handle for the load created using lumpedElement.

Example: ant.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna
0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna
[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: TiltAxis=[0 1 0]

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

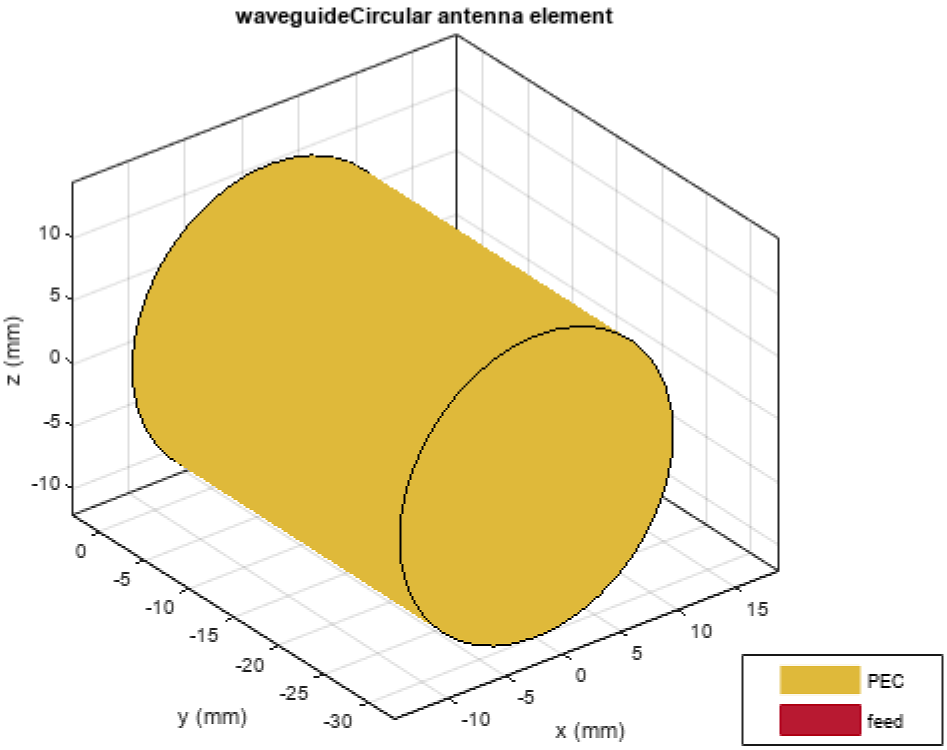
Default Circular Waveguide and Radiation Pattern

Create and view a default circular waveguide.

```
ant = waveguideCircular
```

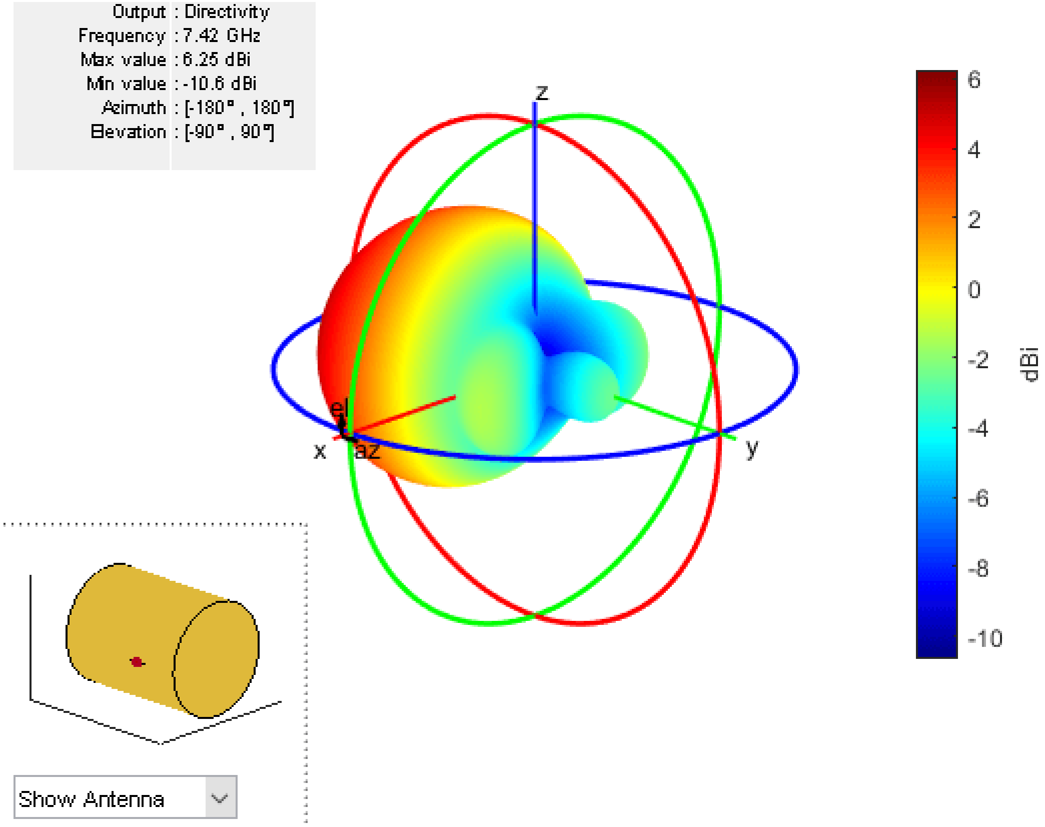
```
ant = 
  waveguideCircular with properties:

        Radius: 0.0120
        Height: 0.0300
    FeedHeight: 0.0075
     FeedWidth: 0.0040
    FeedOffset: 0.0100
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 7.42 GHz.

```
pattern(ant,7.42e9)
```

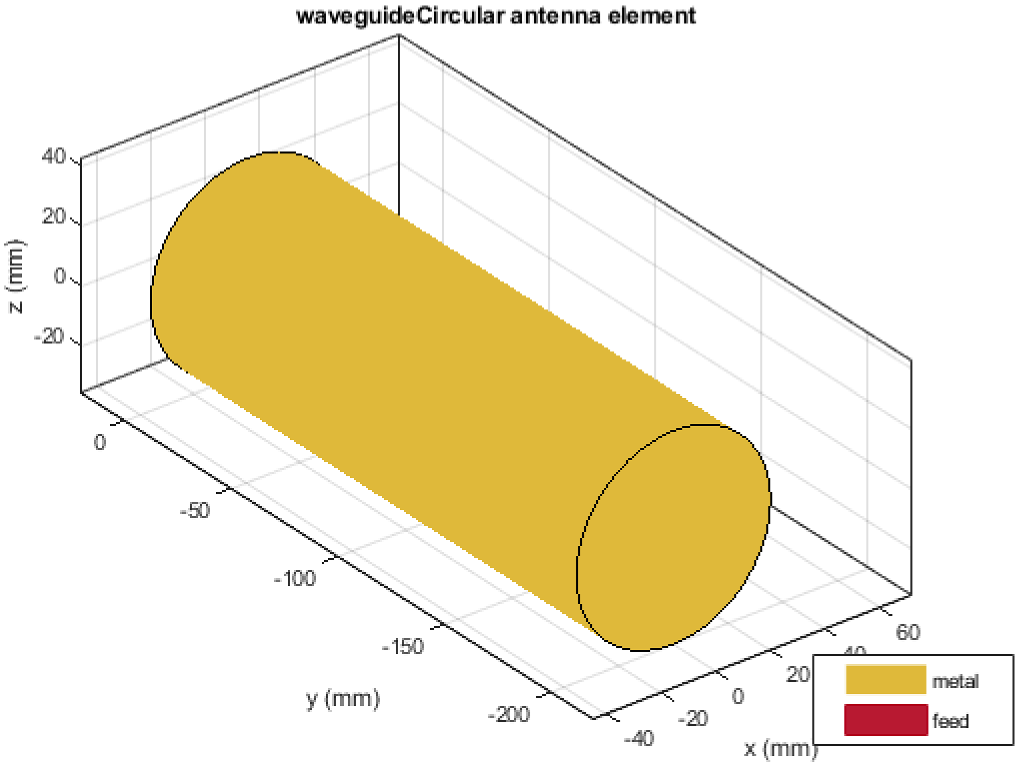


Output : Directivity
Frequency : 7.42 GHz
Max value : 6.25 dBi
Min value : -10.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

S-Parameters and Impedance of Custom Circular Waveguide

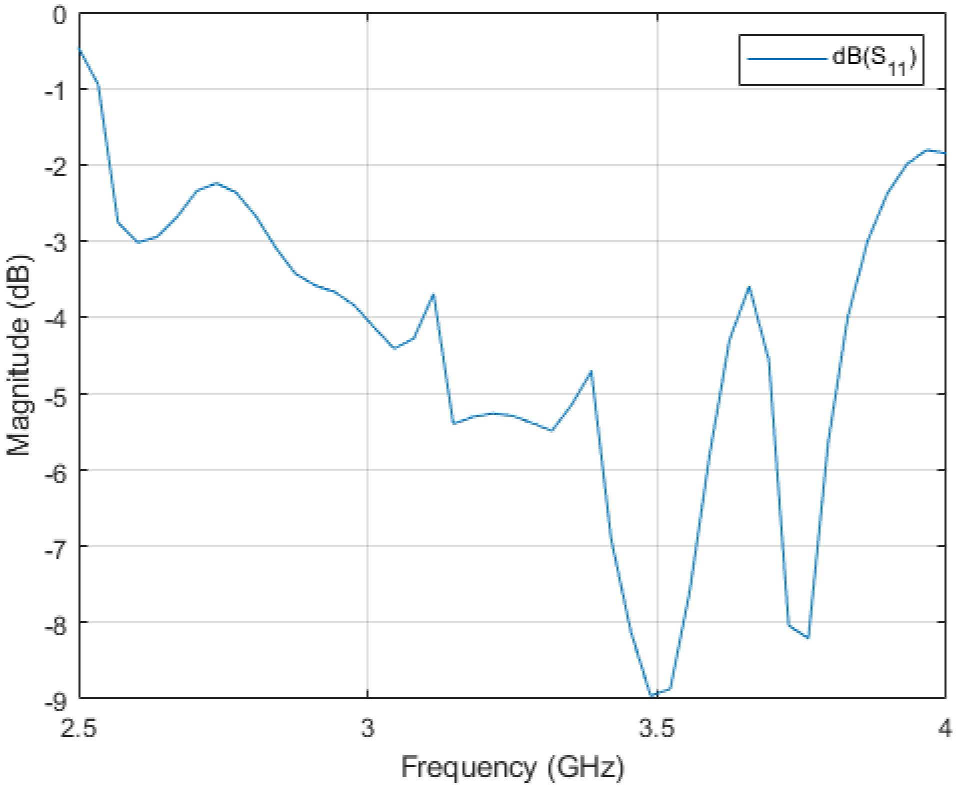
Create a circular waveguide with the following dimensions.

```
ant=waveguideCircular('Radius',35.7e-3,'Height',200e-3,...
       'Feedwidth',26e-3,'FeedHeight',34.71e-3,'FeedOffset', 42.42e-3);
show(ant);
```

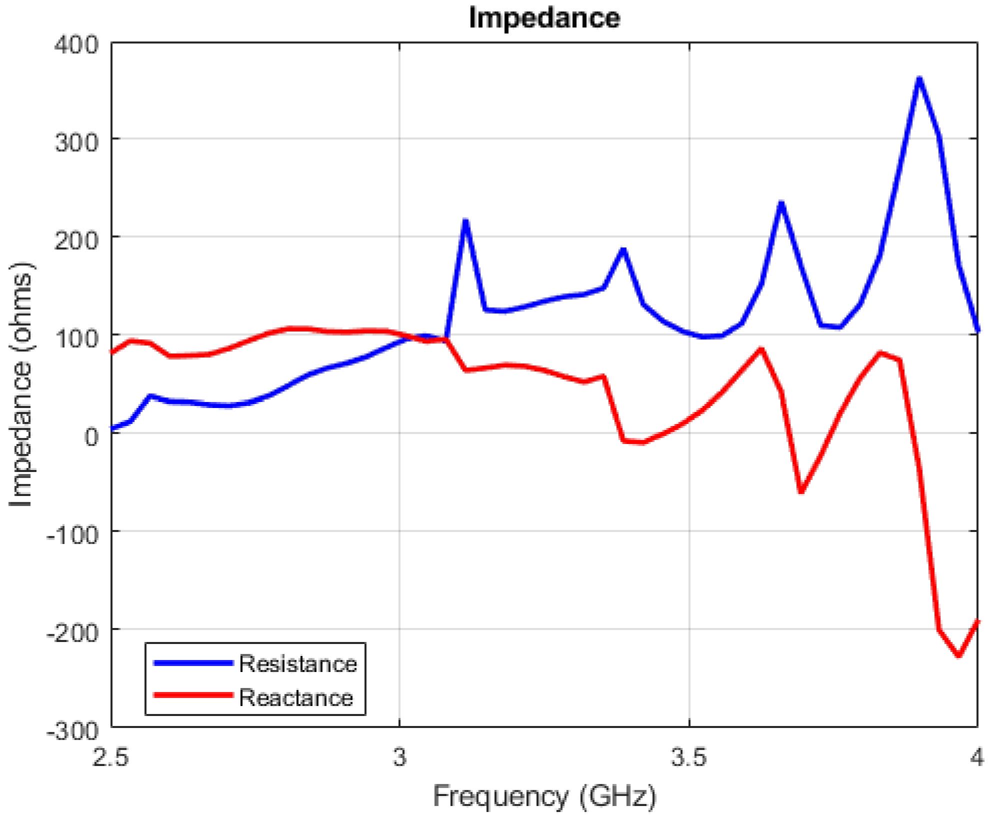


Plot the s-parameters and impedance of the waveguide.

```
s=sparameters(ant,linspace(2.5e9,4e9,45));
rfplot(s);
```



```
figure;
impedance(ant,linspace(2.5e9,4e9,45));
```



Version History

Introduced in R2019b

References

[1] Jadhav, Rohini.P, Vinithkurnar Javnrakash Dongre, Arunkumar Heddallikar. "Design of X-Band Conical Horn Antenna Using Coaxial Feed and Improved Design Technique for Bandwidth Enhancement." In *International Conference on Computing, Communication, Control, and Automation (ICCUBEA)*, 1-6. Pune, India: ICCUBEA 2017

See Also

`cavityCircular` | `waveguide` | `waveguideSlotted`

Topics

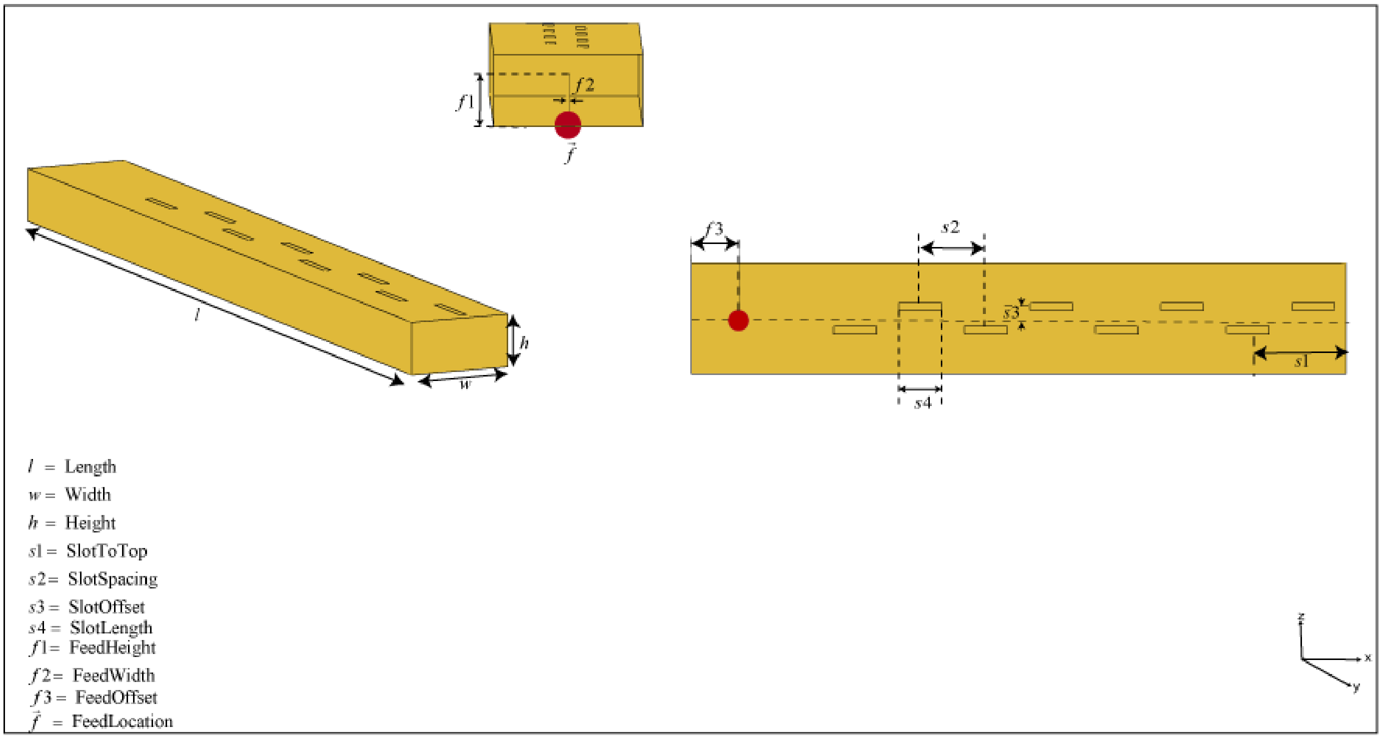
"Rotate Antennas and Arrays"

waveguideSlotted

Create slotted waveguide antenna

Description

The `waveguideSlotted` object creates a slotted waveguide antenna. There are different types of slotted waveguides, including longitudinal slots, transversal slots, center inclined slots, inclined slots, and inclined slots cut into a narrow wall. Slotted waveguide antennas are used in navigation radar as an array fed by a waveguide.



Creation

Syntax

```
ant = waveguideSlotted
ant = waveguideSlotted(Name,Value)
```

Description

`ant = waveguideSlotted` creates a slotted waveguide antenna on the *xy*- plane. The circumference of the antenna is chosen for an operating frequency of 2.45 GHz.

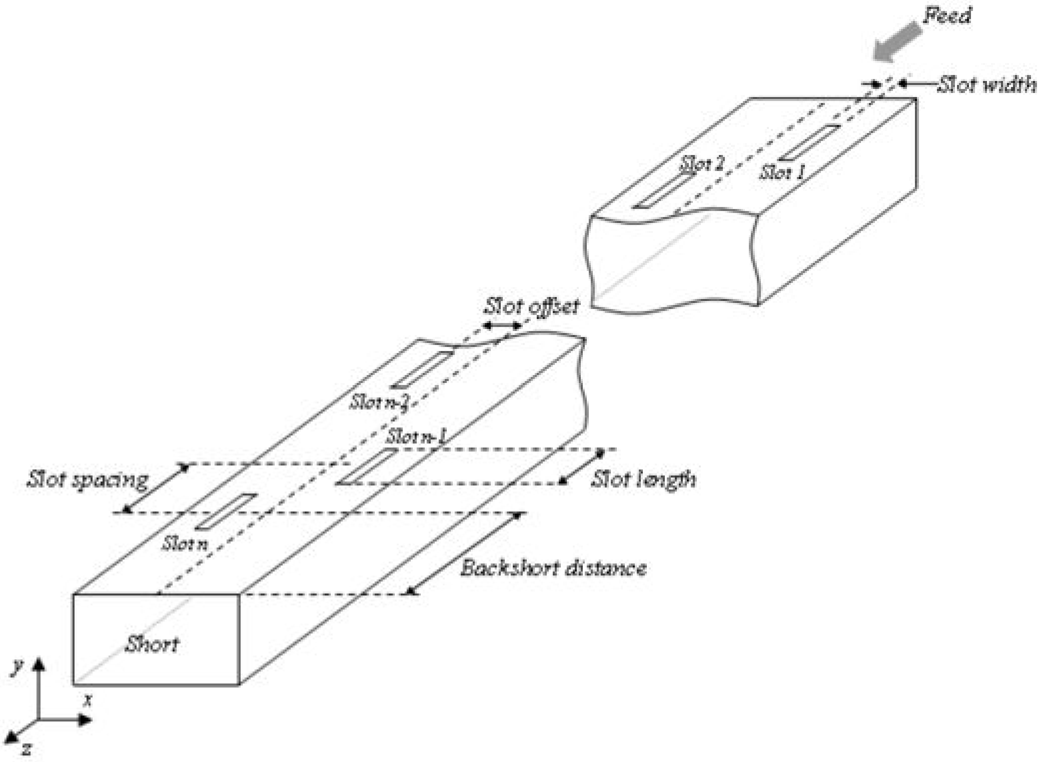
`ant = waveguideSlotted(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = waveguideSlotted('Height',1)` creates a slotted waveguide with a height of 1 meter.

Properties

`Length` — Length of waveguide (n times lambda)

`0.8060` (default) | real-valued scalar

Length of the waveguide (*n* times lambda), specified as a real-valued scalar in meters. *n* is the number of slots in the waveguide.



Example: `'Length',0.760`

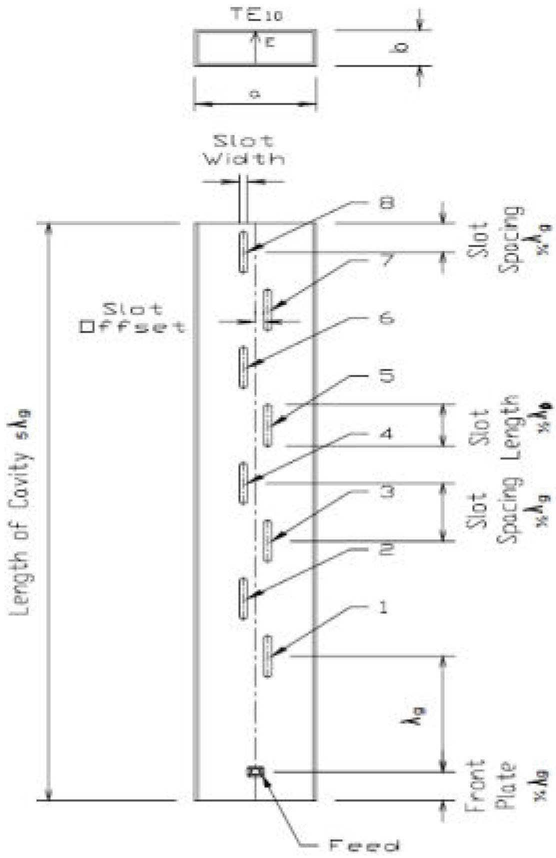
Example: `ant.Length = 0.760`

Data Types: `double`

`Width` — Width of waveguide (a)

`0.0857` (default) | real-valued scalar

Width of the waveguide (*a*), specified as a real-valued scalar in meters.



Example: `'Width',0.0840`

Example: `ant.Width = 0.0840`

Data Types: `double`

`Height` — Height of waveguide (b)
`0.0428` (default) | real-valued scalar

Height of the waveguide (*b*), specified as a real-valued scalar in meters. Please see image in `Width` property.

Example: `'Height',0.0340`

Example: `ant.Height = 0.0340`

Data Types: `double`

`Numslots` — Number of slots
`8` (default) | scalar integer

Number of slots (*n*), specified as a scalar integer.

Example: `'Numslots',7`

Example: `ant.Numslots = 7`

Data Types: `double`

`Slot` — Shape of slots

`antenna.Rectangle` object (default) | `antenna.Circle` object | `antenna.Polygon` object | `antenna.ellipse`

Shape of waveguide slot, specified as one of the following objects: `antenna.Circle`, `antenna.Polygon`, `antenna.Rectangle`, and `antenna.Ellipse`.

Example: `'Slot',antenna.rectangle['Length',0.035]`

Example: `ant.Slot = antenna.rectangle['Length',0.035]`

Data Types: `double`

`SlotToTop` — Distance from closed face edge to top slot center

`0.0403` (default) | real-valued scalar

Distance from the closed face edge to the top slot center, specified as a real-valued scalar in meters.

Example: `'SlotToTop',0.0503`

Example: `ant.SlotToTop = 0.0503`

Data Types: `double`

`SlotSpacing` — Space between centers of two adjacent slots

`0.0806` (default) | real-valued scalar

Space between the centers of two adjacent slots, specified as a real-valued scalar in meters.

Example: `'SlotSpacing',0.0906`

Example: `ant.SlotSpacing = 0.0906`

Data Types: `double`

`SlotOffset` — Slot displacement from centreline of width of waveguide to center of slot

`0.0123` (default) | real-valued scalar | vector

Slot displacement from the centreline of the width of the waveguide to the center of the slot, specified as a real-valued scalar or vector in meters.

Note If `SlotOffset` is a vector, it can be the size of 1-by-`n` where, `n` < `NumSlots`.

Example: `'SlotOffset',0.0560`

Example: `ant.SlotOffset = 0.0560`

Data Types: `double`

`SlotAngle` — Slot angle

`0` (default) | real-valued scalar | vector

Slot angle, specified as a real-valued scalar in degrees or a vector with each element unit in degrees. In slotted waveguide the slots are in pairs. You use a vector when you want one slot in the pair to be tilted at a different angle form the other. It varies from `- 180`° to `180`°.

Note If `SlotAngle` is a vector, it can be the size of 1-by-`n` where, `n` <= `NumSlots`.

Example: `'SlotAngle',[20 10]`

Example: `ant.SlotAngle = [20 10]`

Data Types: `double`

`ClosedWaveguide` — Plate or cover to close waveguide

`0` (default) | `1`

Plate to close the open-ended side, specified as `0` for open waveguide and `1` for closed waveguide.

Example: `'ClosedWaveguide',1`

Example: `ant.ClosedWaveguide = 1`

Data Types: `double`

`FeedHeight` — Height of feed

`0.0310` (default) | real-valued scalar

Height of the feed, specified as a real-valued scalar in meters.

Example: `'FeedHeight',0.0210`

Example: `ant.FeedHeight = 0.0210`

Data Types: `double`

`FeedWidth` — Width of feed

`0.0020` (default) | real-valued scalar

Width of the feed, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0300`

Example: `ant.FeedWidth = 0.0300`

Data Types: `double`

`FeedOffset` — Signed distances from origin

`[-0.3627 0]` (default) | two-element vector

Signed distances from the origin measured along the length and width of the waveguide, specified as a two-element vector with each element in meters.

Example: `'FeedOffset',[-0.3627 0]`

Example: `ant.FeedOffset = [-0.3627 0]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
efficiency	Radiation efficiency of antenna
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

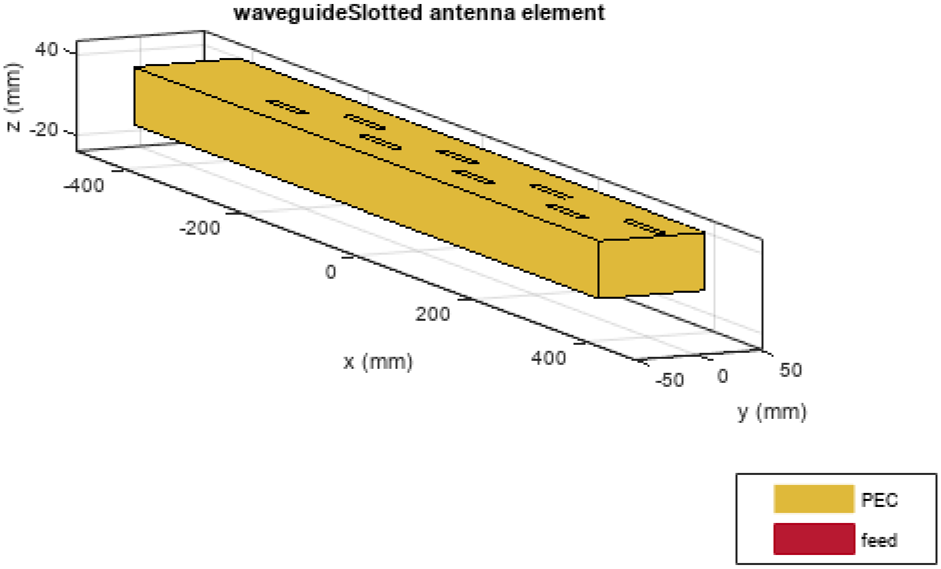
Default Slotted Waveguide Antenna and Radiation Pattern

Create and view a slotted waveguide antenna with default property values.

```
ant = waveguideSlotted
```

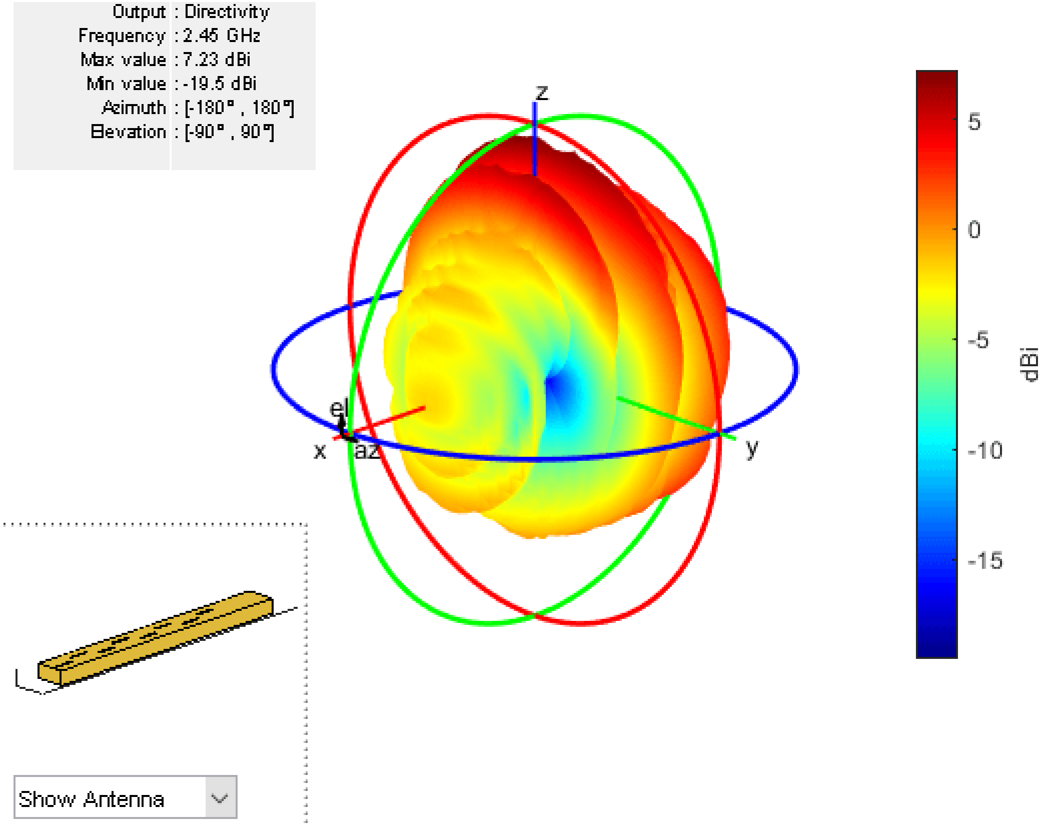
```
ant = 
  waveguideSlotted with properties:

             Length: 0.8060
              Width: 0.0857
             Height: 0.0428
           NumSlots: 8
               Slot: [1x1 antenna.Rectangle]
          SlotToTop: 0.0403
        SlotSpacing: 0.0806
         SlotOffset: 0.0123
          SlotAngle: 0
          FeedWidth: 0.0020
         FeedHeight: 0.0310
         FeedOffset: [-0.3627 0]
    ClosedWaveguide: 0
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 2.45 GHz.

```
pattern(ant, 2.45e9)
```

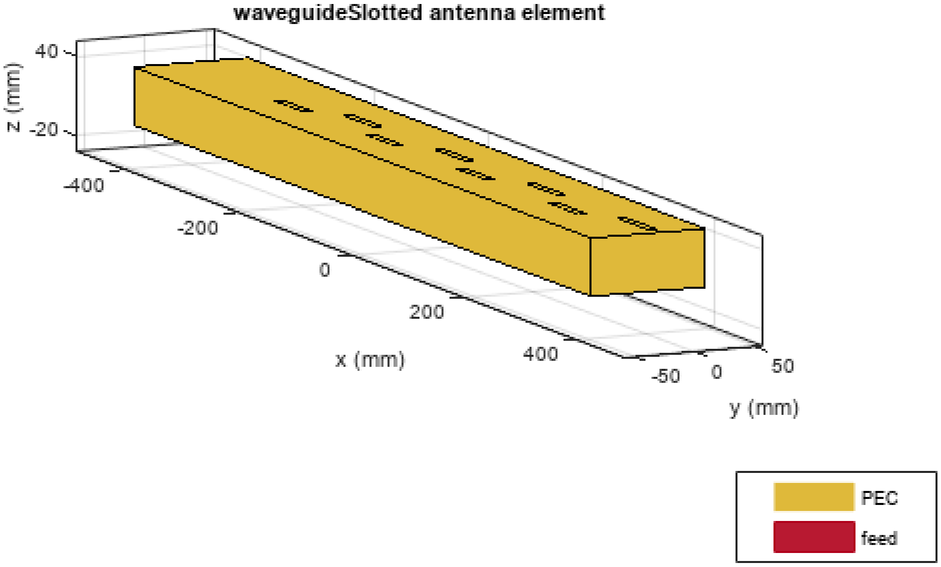


Output : Directivity
Frequency : 2.45 GHz
Max value : 7.23 dBi
Min value : -19.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Impedance and S-Parameters of Custom Slotted Waveguide Antenna

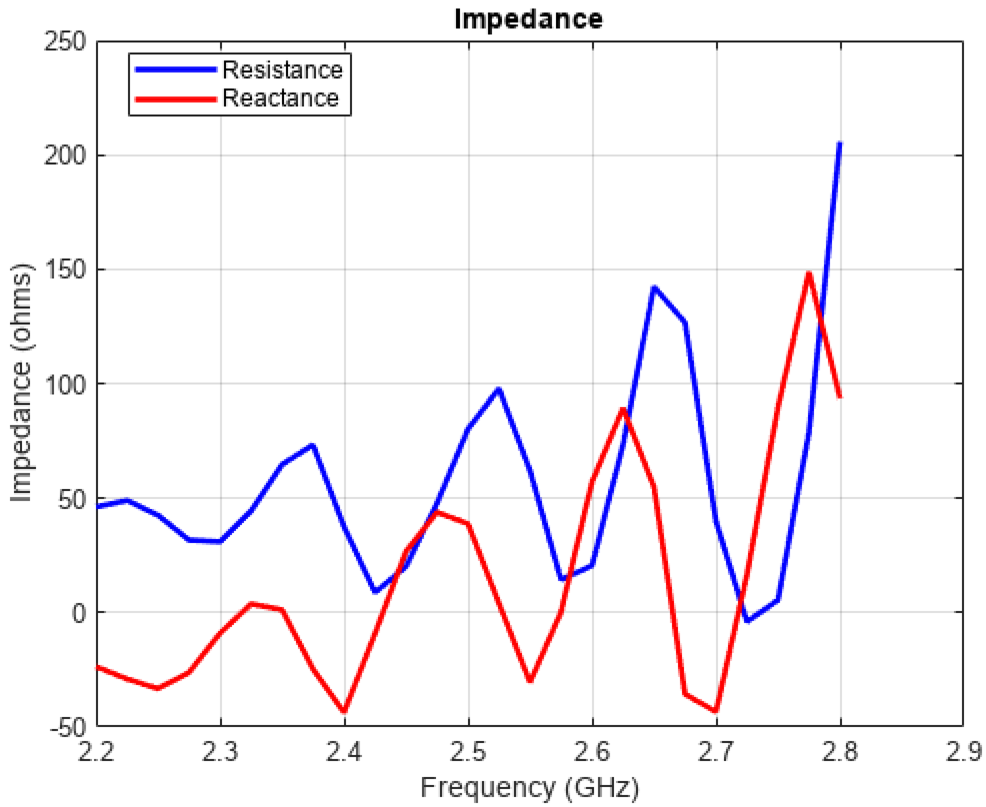
Create a slotted waveguide antenna with the following dimensions.

```
 ant = waveguideSlotted('Length',806e-3,'Width',94e-3, 'NumSlots',8,...
      'Height',44e-3,'Slot',antenna.Rectangle('Length',53e-3,'Width',6.5e-3),'SlotToTop',40.3e-3
      'SlotSpacing',80.6e-3,'SlotOffset',10e-3,'FeedHeight',31e-3, ...
      'FeedOffset',[-362.7e-3 0],'FeedWidth',2e-3);
show (ant)
```

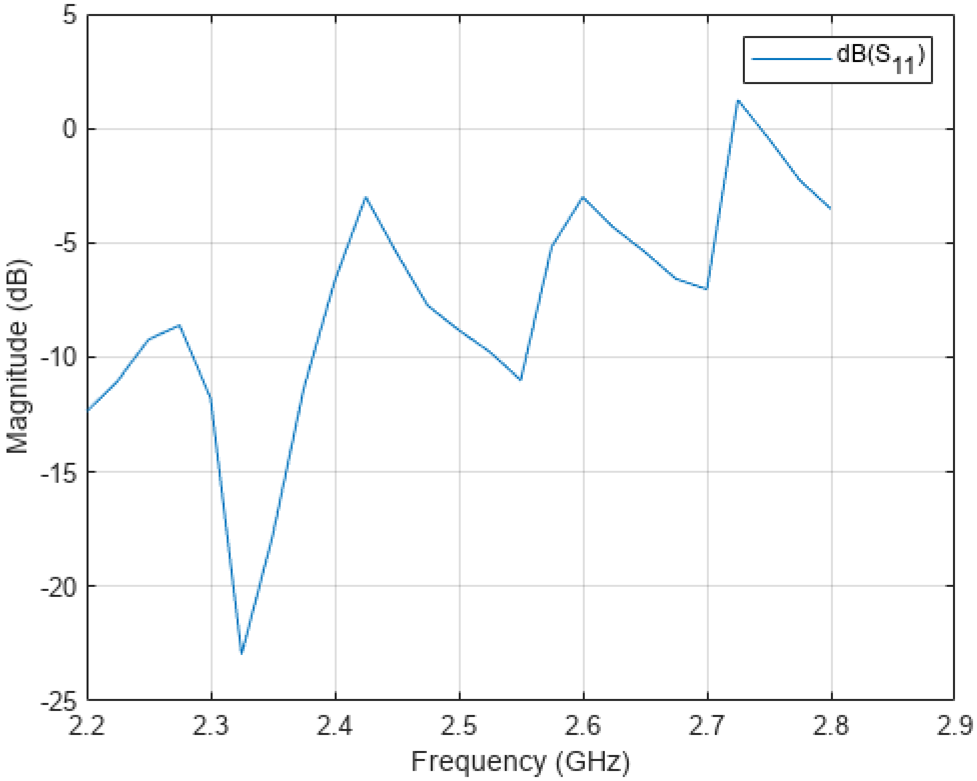


Plot impedance and S-parameters from 2.2 GHz to 2.8 GHz.

```
freq = 2.2e9:0.025e9:2.8e9;
figure;
impedance(ant,freq);
```



```
s = sparameters(ant,freq);
figure;
rfplot(s);
```



Version History

Introduced in R2019b

References

[1] Perovic, Una. " Investigation of Rectangular, Unidirectional, Horizontally Polarized Waveguide Antenna with Longitudinal Slotted Arrays Operating at 2.45 GHz".

See Also

`waveguide` | `waveguideCircular` | `cavityCircular`

Topics

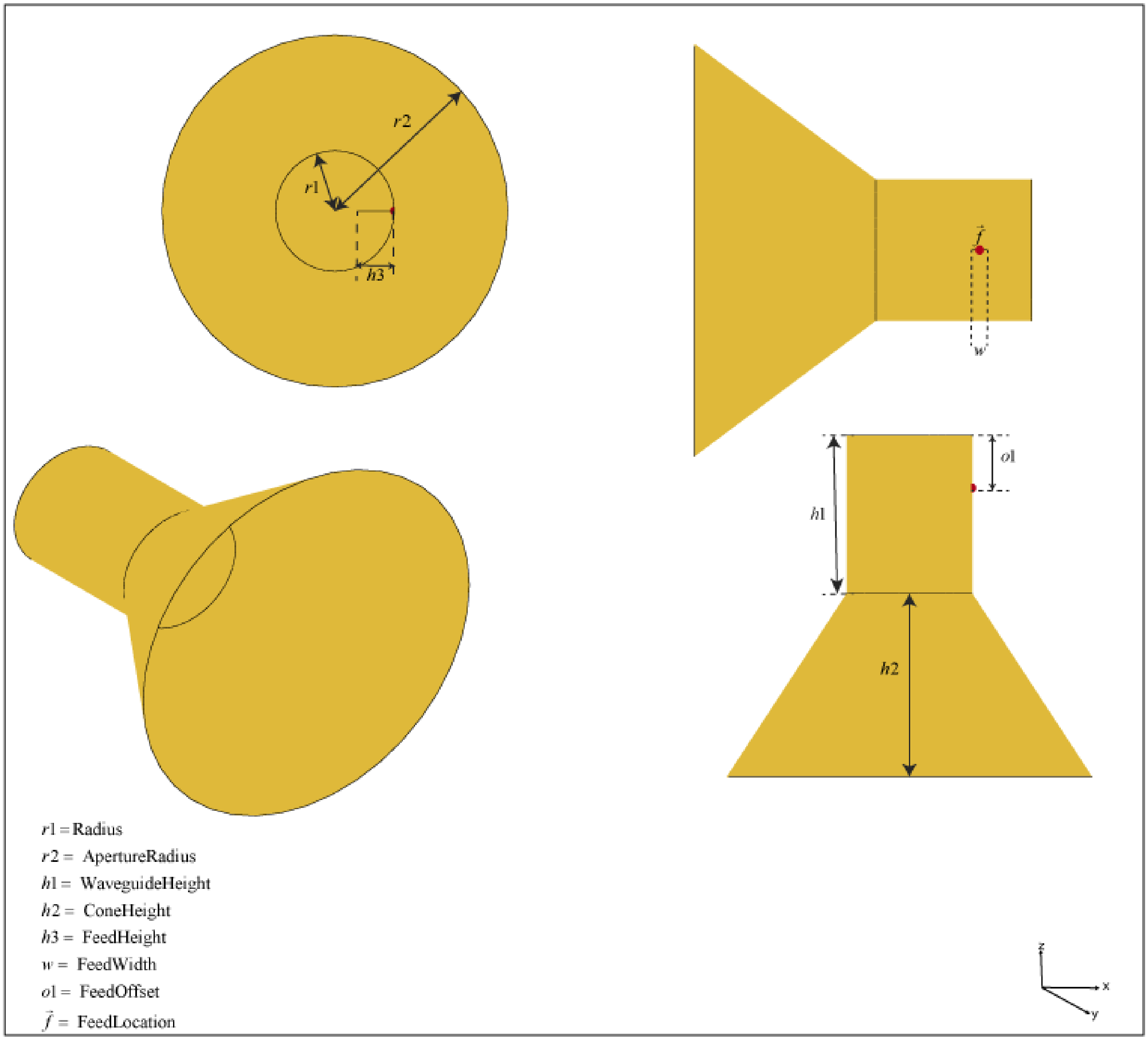
“Rotate Antennas and Arrays”

hornConical

Create conical horn antenna

Description

The `hornConical` object creates a waveguide shaped like a cone to direct radio waves in a beam. This type of horn is widely used as feed element for large radio astronomy telescopes, satellite tracking, and communication dishes.



Creation

Syntax

```
ant = hornConical
ant = hornConical(Name,Value)
```

Description

`ant = hornConical` creates a conical horn antenna with dimensions for an operating frequency of 7.58 GHz.

`ant = hornConical(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = hornConical('Radius',1)` creates a conical horn antenna with a radius of 1 meter.

Properties

`Radius` — Radius of waveguide
`0.0120` (default) | real-valued scalar

Radius of the waveguide, specified as a real-valued scalar in meters.

Example: `'Radius',0.760`

Example: `ant.Radius = 0.760`

Data Types: `double`

`WaveguideHeight` — Height of waveguide
`0.0300` (default) | real-valued scalar

Height of the waveguide, specified as a real-valued scalar in meters.

Example: `'WaveguideHeight',0.0340`

Example: `ant.WaveguideHeight = 0.0340`

Data Types: `double`

`FeedHeight` — Height of feed
`0.0075` (default) | real-valued scalar

Height of the feed, specified as a real-valued scalar in meters.

Example: `'FeedHeight',0.0085`

Example: `ant.FeedHeight = 0.0085`

Data Types: `double`

`FeedWidth` — Width of feed
`0.0030` (default) | real-valued scalar

Width of the feed, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0200`

Example: `ant.FeedWidth = 0.0200`

Data Types: `double`

`FeedOffset` — Signed distance along *y*-axis

`0.0100` (default) | real-valued scalar

Signed distances along the *y*-axis, specified as a real-valued scalar in meters.

Example: `'FeedOffset',0.03627`

Example: `ant.FeedOffset = 0.3627`

Data Types: `double`

`ConeHeight` — Height of cone

`0.0348` (default) | real-valued scalar

Height of the cone, specified as a real-valued scalar in meters.

Example: `'ConeHeight',0.0540`

Example: `ant.ConeHeight = 0.0540`

Data Types: `double`

`ApertureRadius` — Radius of cone aperture

`0.0350` (default) | real-valued scalar

Radius of the cone aperture, specified as a real-valued scalar in meters.

Example: `'ApertureRadius',0.0760`

Example: `ant.ApertureRadius = 0.0760`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer

pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

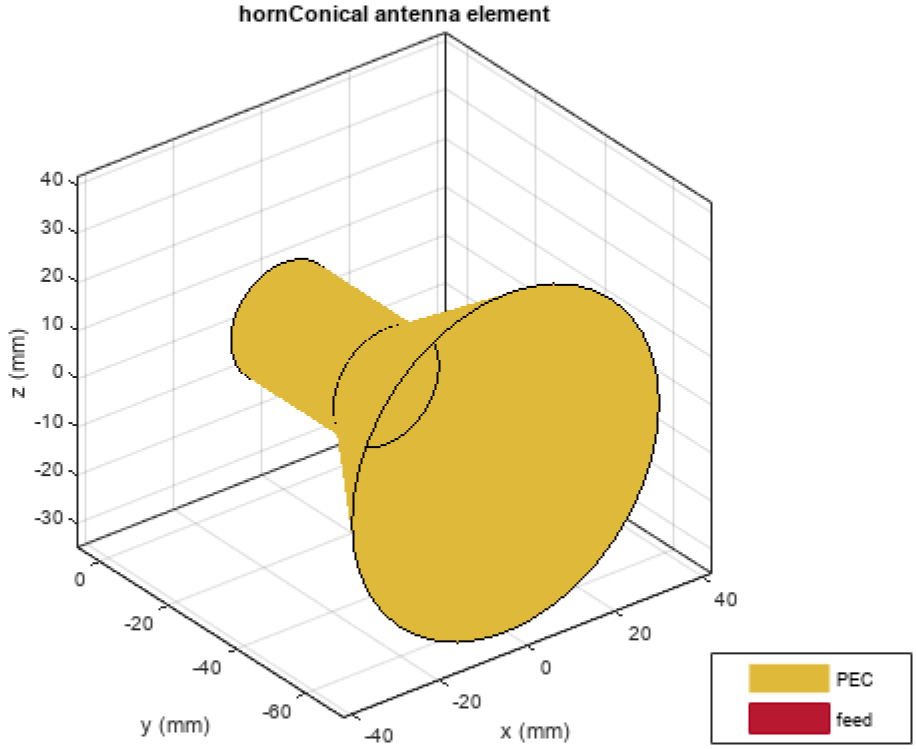
Default Conical Horn and Radiation Pattern

Create and view a default conical horn antenna.

```
ant = hornConical
```

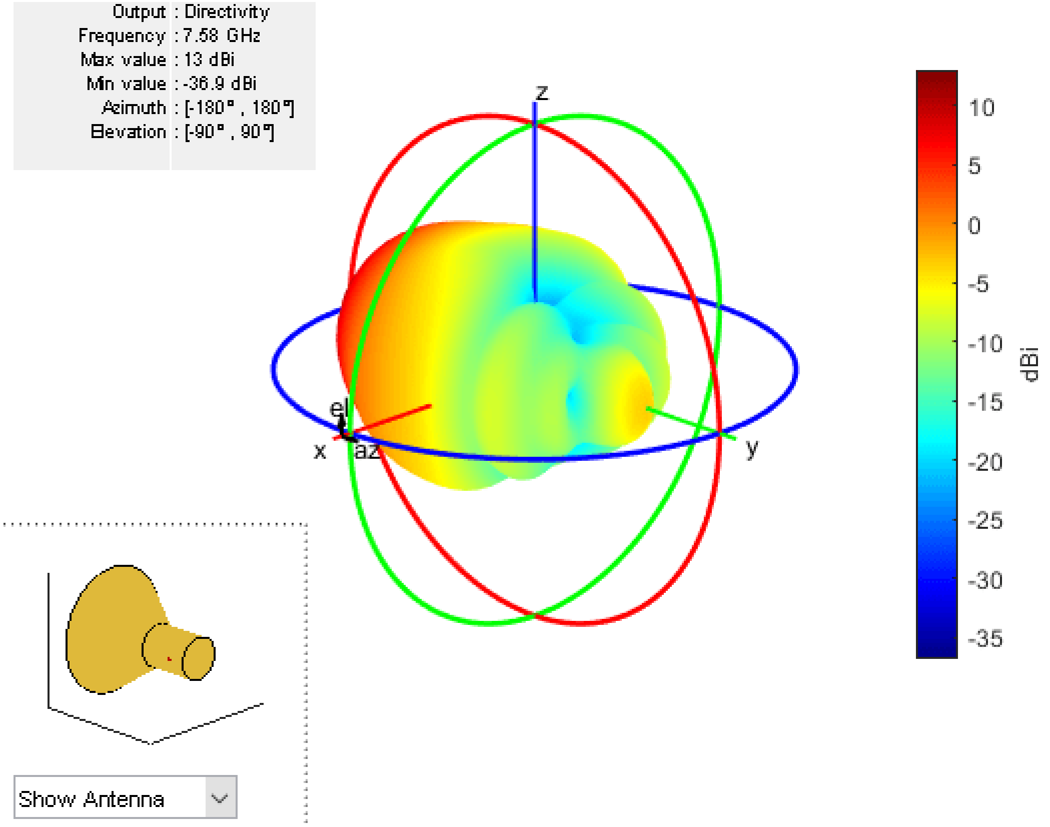
```
ant = 
  hornConical with properties:

             Radius: 0.0120
    WaveguideHeight: 0.0300
         FeedHeight: 0.0075
          FeedWidth: 0.0030
         FeedOffset: 0.0100
         ConeHeight: 0.0348
     ApertureRadius: 0.0350
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 7.58 GHz.

```
pattern(ant,7.58e9)
```

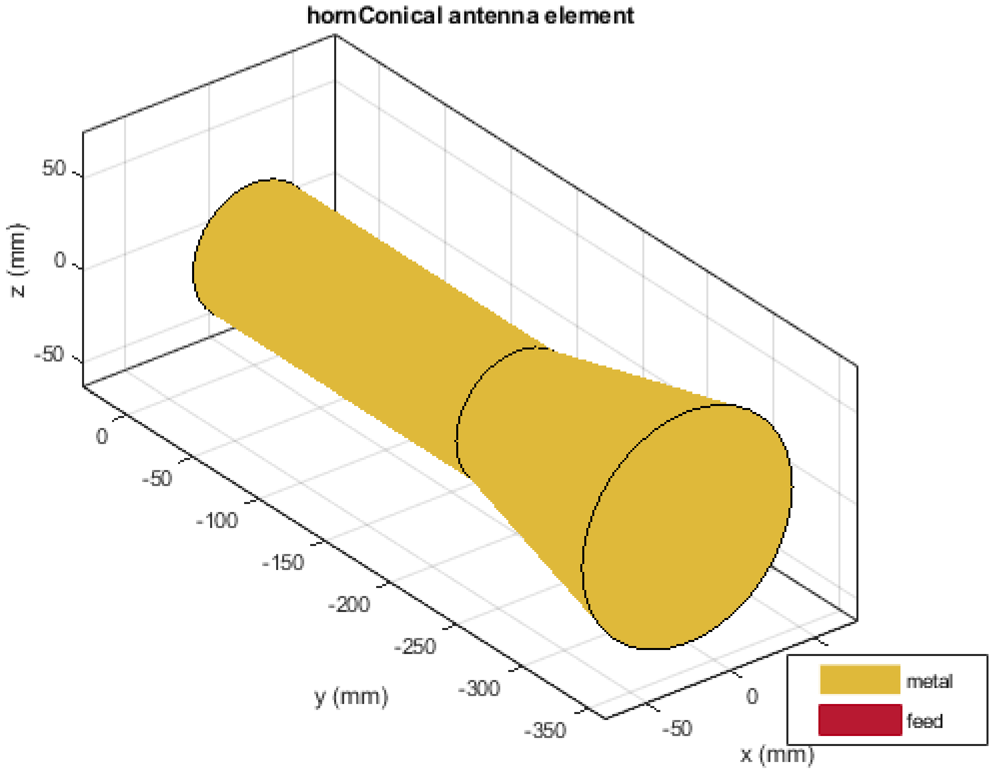


Output : Directivity
Frequency : 7.58 GHz
Max value : 13 dBi
Min value : -36.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Impedance and S-Parameters of Custom Conical Horn Antenna

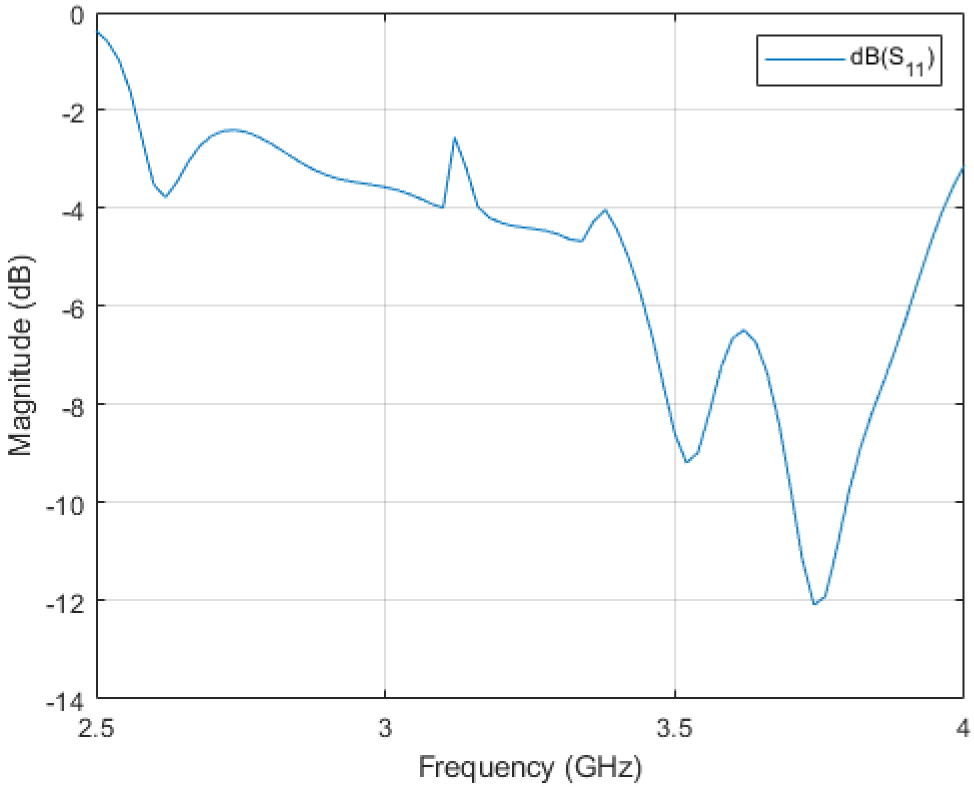
Create a conical horn antenna with the following dimensions.

```
ant=hornConical('Radius',35.71e-3,'WaveguideHeight',200e-3,...
       'Feedwidth',26e-3,'FeedHeight',34.71e-3,'FeedOffset',42.42e-3,...
       'ConeHeight',130e-3,'ApertureRadius',62.5e-3);
show(ant);
```

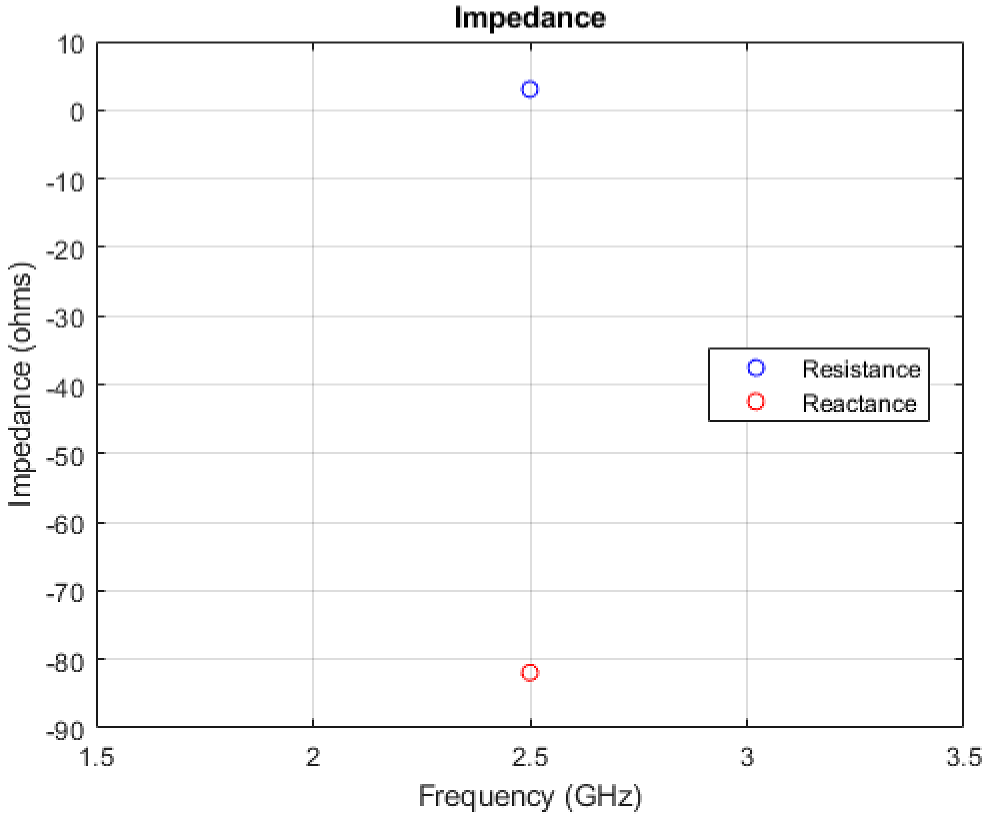


Plot the s-parameters and the impedance of the antenna.

```
s=sparameters(ant,2.5e9:20e6:4e9);
rfplot(s);
```



```
figure;
impedance(ant,2.5e9,20e6:4e9);
```



Version History

Introduced in R2019b

References

[1] Jadhav, Rohini.P, Vinithkurnar Javnrakash Dongre, Arunkumar Heddallikar. "Design of X-Band Conical Horn Antenna Using Coaxial Feed and Improved Design Technique for Bandwidth Enhancement." In *International Conference on Computing, Communication, Control, and Automation (ICCUBEA)*, 1-6. Pune, India: ICCUBEA 2017

See Also

`waveguide` | `horn` | `cavityCircular` | `hornangle2size`

Topics

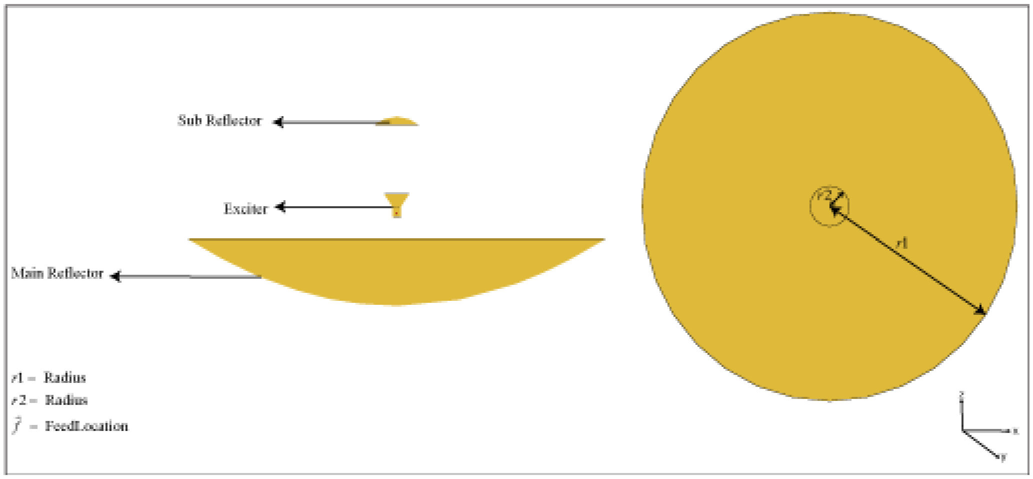
“Rotate Antennas and Arrays”

gregorian

Create Gregorian antenna

Description

The `gregorian` object creates a horn conical fed Gregorian antenna. A Gregorian antenna is a parabolic antenna. In this antenna, the feed antenna is mounted at or behind the surface of the main parabolic reflector and aimed at the subreflector. This antenna is used in radio telescopes and communication satellites. For more information see, “Architecture of Gregorian Antenna” on page 1-547.



Creation

Syntax

```
ant = gregorian
ant = gregorian(Name,Value)
```

Description

`ant = gregorian` creates a horn conical fed Gregorian antenna with a default operating frequency of 18.48 GHz. This antenna gives maximum gain when operated at 18.3 GHz.

`ant = gregorian(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = gregorian('FocalLength',[0.4 0.22])` creates a Gregorian antenna with the main reflector of focal length 0.4 m and the subreflector of focal length 0.22 m.

Properties

`Exciter` — Antenna or array type used as exciter
`hornConical` (default) | antenna object | `array` object

Antenna type used as exciter, specified as an antenna or an array object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Radius` — Radius of main and subreflector

`[0.3175 0.0330]` (default) | two-element vector

Radius of the main and subreflector, specified as a two-element vector with each element unit in meters. The first element specifies the radius of the main reflector, and the second element specifies the radius of the subreflector.

Example: `'Radius',[0.4 0.2]`

Example: `ant.Radius = [0.4 0.2]`

Data Types: `double`

`FocalLength` — Focal length of main and subreflector

`[0.2536 0.1416]` (default) | two-element vector

Focal length of the main and subreflector, specified as a two-element vector with each element unit in meters. The first element specifies the focal length of the main reflector, and the second element specifies the focal length of the subreflector.

Example: `'FocalLength',[0.35 0.2]`

Example: `ant.FocalLength = [0.35 0.2]`

Data Types: `double`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`SolverType` — Solver for antenna analysis

`'MoM-PO'` (default) | `'MoM'` | `'FMM'`

Solver for antenna analysis, specified as the comma-separated pair consisting of `'SolverType'` and `'MoM-PO'` or `'MoM'` (Method of Moments) or `'FMM'` (Fast Multipole Method).

Example: `'SolverType'`,`'MOM'`

Data Types: `char`

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array

patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

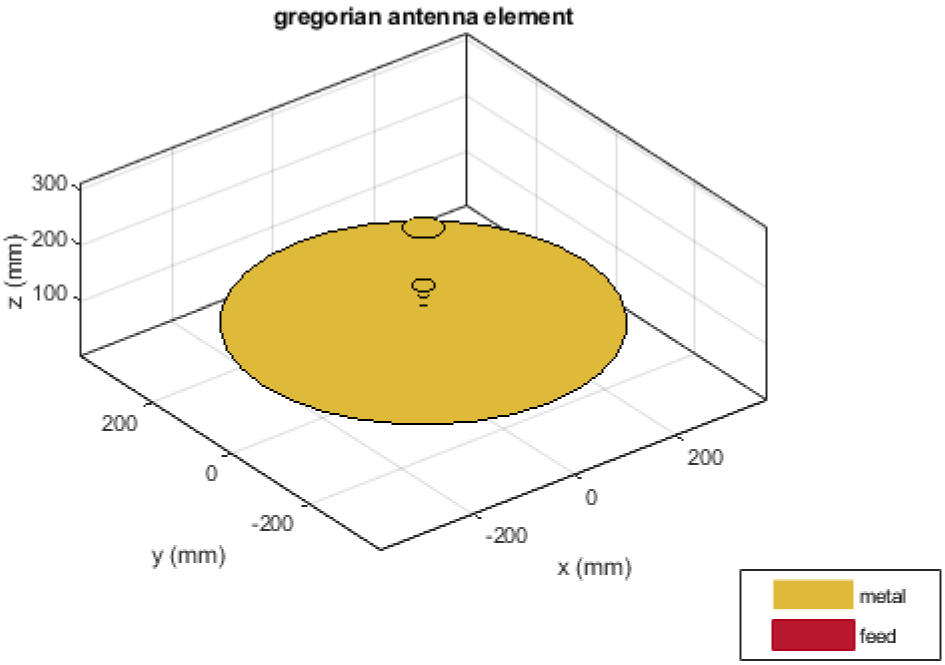
Default Gregorian Antenna and Radiation Pattern

Create and view a default Gregorian antenna.

```
ant = gregorian
```

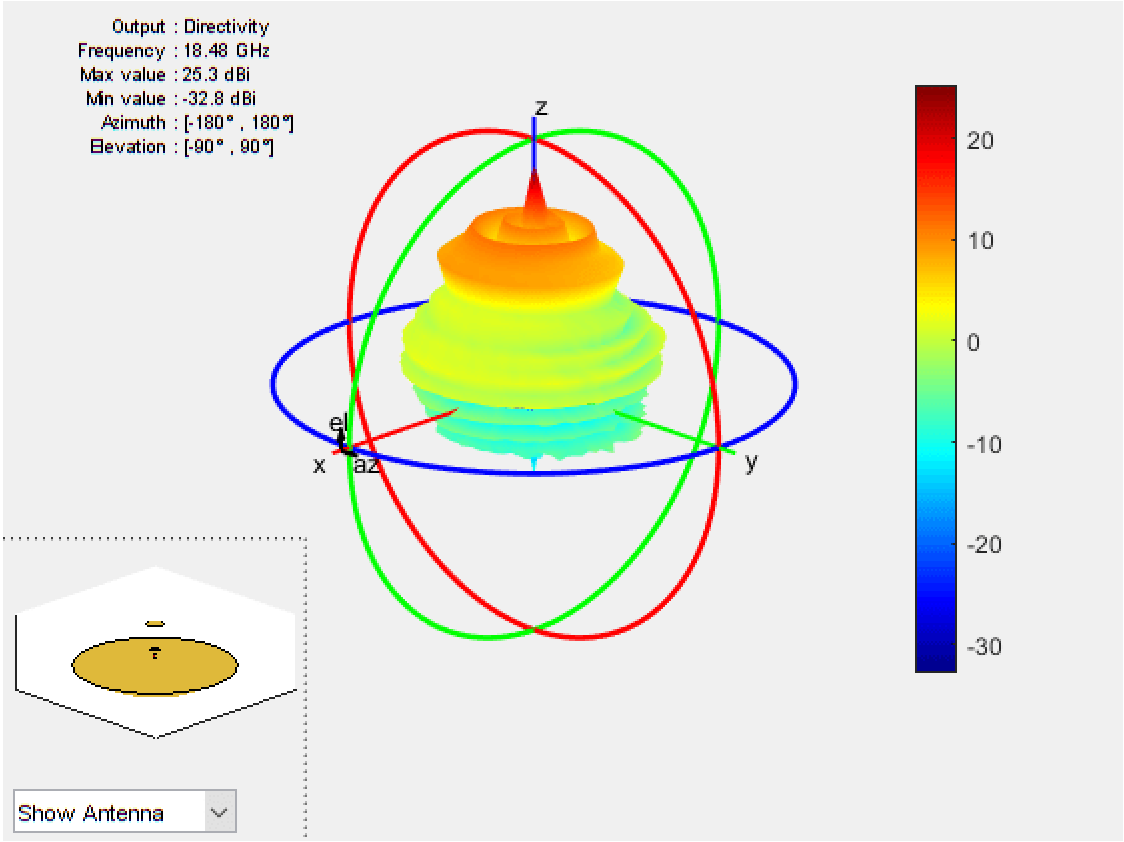
```
ant = 
  gregorian with properties:

        Exciter: [1×1 hornConical]
         Radius: [0.3175 0.0330]
    FocalLength: [0.2536 0.1416]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1×1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 18.48 GHz.

```
pattern(ant,18.48e9)
```



Output : Directivity
Frequency : 18.48 GHz
Max value : 25.3 dBi
Min value : -32.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

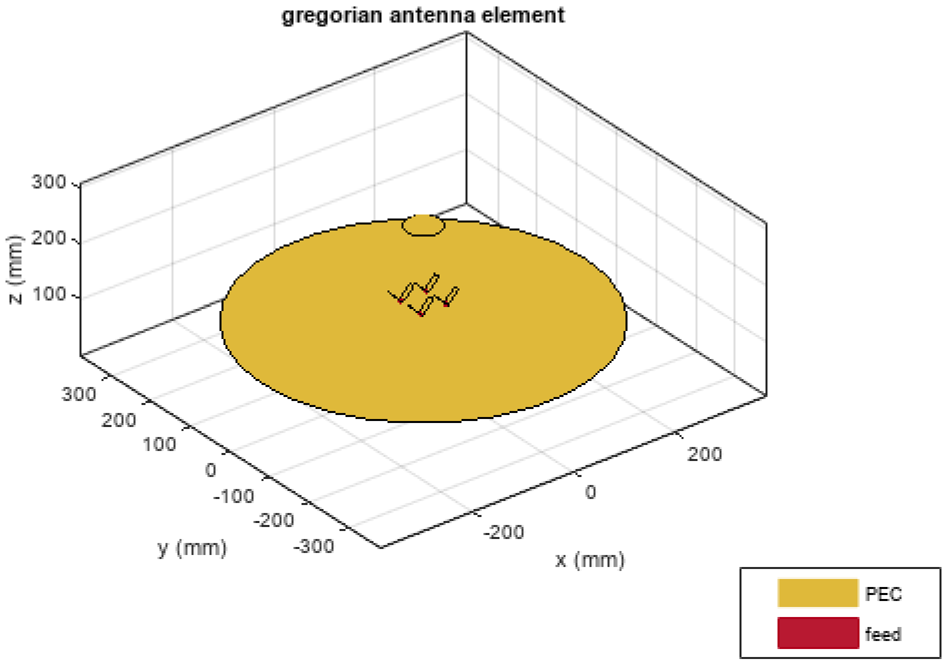
Create Array-fed Gregorian Antenna

Create an array of vee dipole antennas.

```
e = dipoleVee('ArmLength',[0.03 0.03],'Width',0.01);
arr = rectangularArray('Element',e,'RowSpacing',0.05,'ColumnSpacing',0.05);
```

Create a Gregorian antenna with rectangular array as exciter

```
ant = gregorian('Exciter',arr);
show(ant)
```



More About

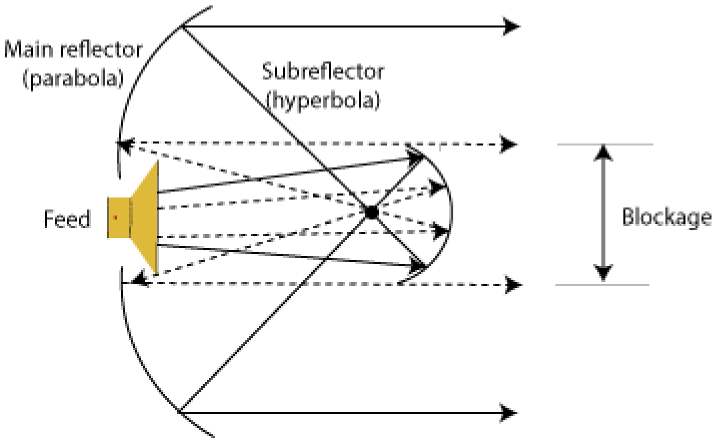
Parabolic Reflector Antennas

A typical parabolic antenna consists of a parabolic reflector with a small feed antenna at its focus. Parabolic reflectors used in dish antennas have a large curvature and short focal length and the focal point is located near the mouth of the dish, to reduce the length of the supports required to hold the feed structure. In more complex designs, such as the cassegrain antenna, a sub reflector is used to direct the energy into the parabolic reflector from a feed antenna located away from the primary focal point. Such type of antennas can be used in satellite communications and Astronomy and other emerging modes of communications

Architecture of Gregorian Antenna

Gregorian antenna consists of three structures:

- Primary parabolic reflector
- Hyperbolic convex subreflector
- Exciter element



Focus of the main reflector and the near focus of the subreflector in the region between the two dishes. Gregorian antenna forms a shorter focal length for the main dish.

Version History

Introduced in R2019b

See Also

`reflectorParabolic` | `cassegrain` | `hornConical`

Topics

"Rotate Antennas and Arrays"

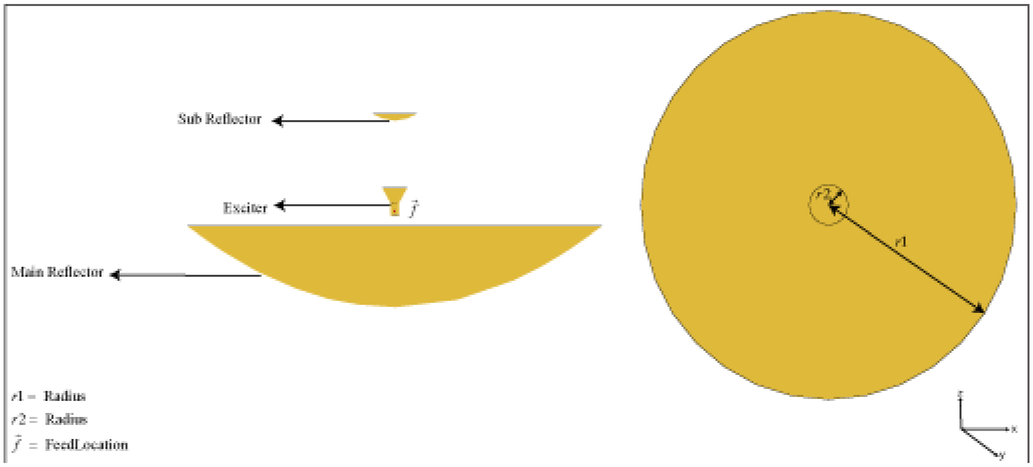
cassegrain

Create Cassegrain antenna

Description

The `cassegrain` object creates a Cassegrain antenna. A Cassegrain antenna is a parabolic antenna using a dual reflector system. In this antenna, the feed antenna is mounted at or behind the surface of the main parabolic reflector and aimed at the secondary reflector. For more information see, “Architecture of Cassegrain Antenna” on page 1-556.

Cassegrain antennas are used in applications such as satellite ground-based systems.



Creation

Syntax

```
ant = cassegrain
ant = cassegrain(Name,Value)
```

Description

`ant = cassegrain` creates a conical horn fed Cassegrain antenna with a resonating frequency of 18.51 GHz. This antenna gives maximum gain when operated at 18 GHz.

`ant = cassegrain(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = cassegrain('Radius',[0.4 0.22])` creates a Cassegrain antenna with the main reflector with radius 0.4 m and the secondary reflector with radius 0.22 m.

Properties

`Exciter` — Antenna or array type used as exciter
`hornConical` (default) | antenna object | `array` object

Antenna type used as exciter, specified as an antenna or an array object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Radius` — Radius of main and subreflector

`[0.3175 0.0330]` (default) | two-element vector

Radius of the main and subreflector, specified as a two-element vector with each element unit in meters. The first element specifies the radius of the main reflector, and the second element specifies the radius of the subreflector.

Example: `'Radius',[0.4 0.2]`

Example: `ant.Radius = [0.4 0.2]`

Data Types: `double`

`FocalLength` — Focal length of main and subreflector

`[0.2536 0.1416]` (default) | two-element vector

Focal length of the main and subreflector, specified as a two-element vector with each element unit in meters. The first element specifies the focal length of the main reflector and the second element specifies the focal length of the subreflector.

Example: `'FocalLength',[0.35 0.2]`

Example: `ant.FocalLength = [0.35 0.2]`

Data Types: `double`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`SolverType` — Solver for antenna analysis

`'MoM-PO'` (default) | `'MoM'` | `'FMM'`

Solver for antenna analysis, specified as the comma-separated pair consisting of `'SolverType'` and `'MoM-PO'` or `'MoM'` (Method of Moments) or `'FMM'` (Fast Multipole Method).

Example: `'SolverType'`,`'MOM'`

Data Types: `char`

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array

patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

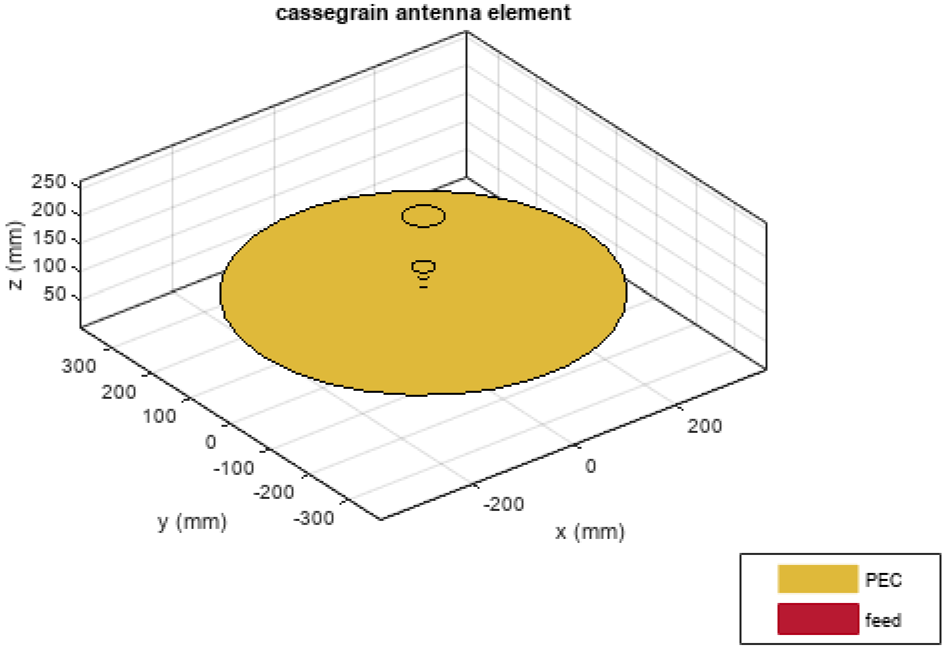
Default Cassegrain Antenna and Radiation Pattern

Create and view a Cassegrain antenna.

```
ant = cassegrain
```

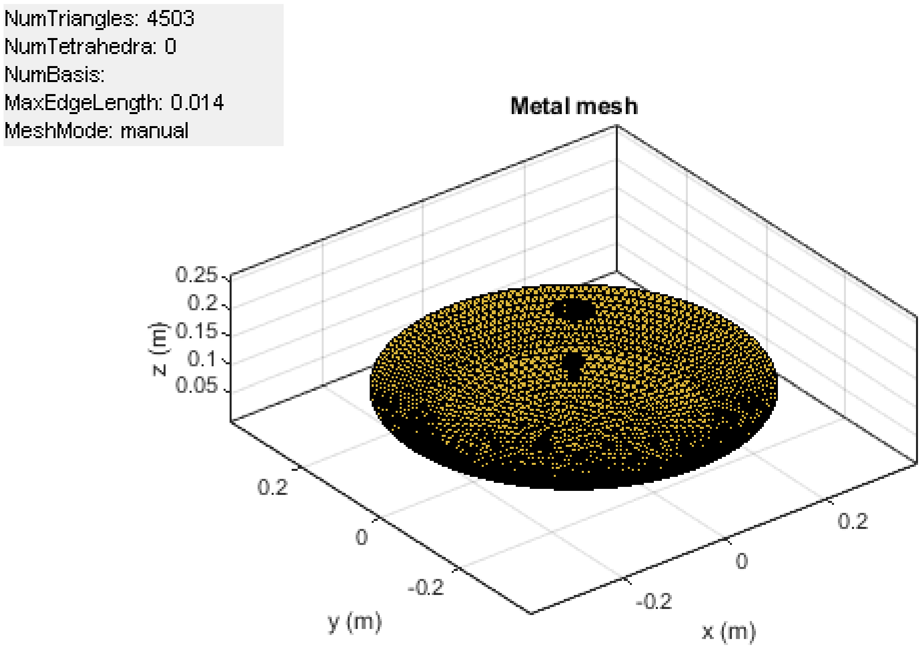
```
ant = 
  cassegrain with properties:

        Exciter: [1x1 hornConical]
         Radius: [0.3175 0.0330]
    FocalLength: [0.2536 0.1416]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
     SolverType: 'MoM-PO'
```

```
show(ant)
```

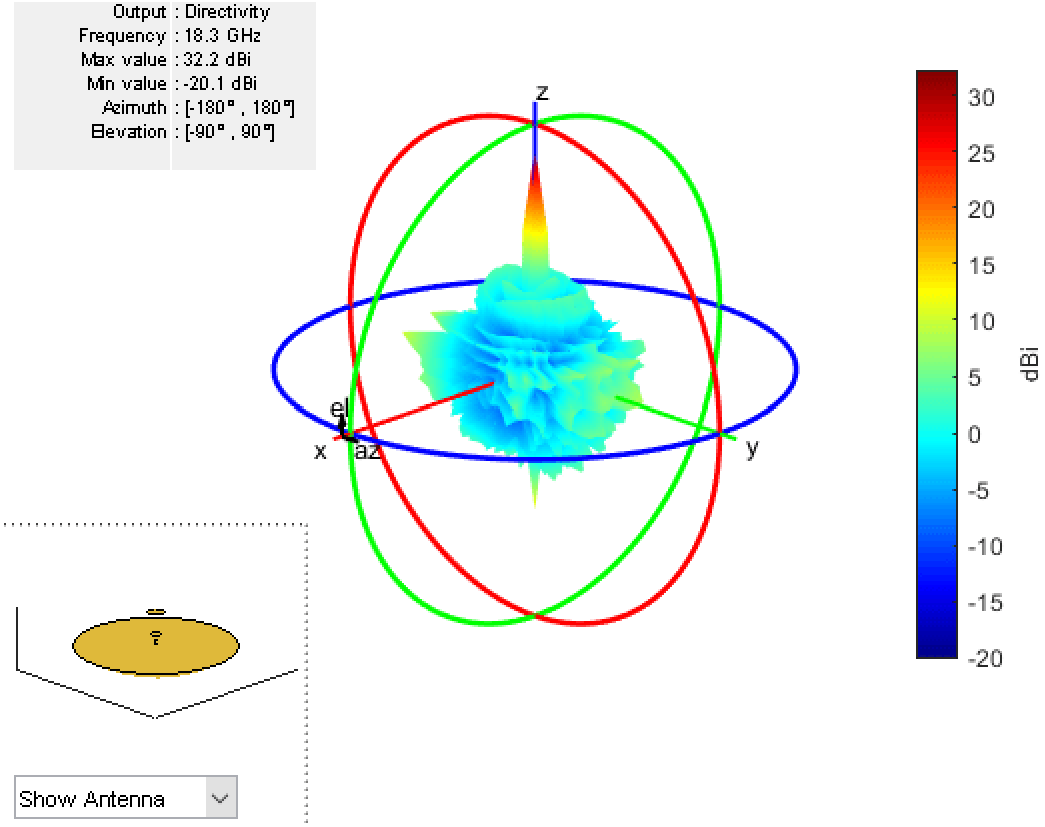


Plot the radiation pattern of the antenna at 18.3 GHz.

```
mesh(ant,'maxEdgeLength',14e-3)
```



```
figure;
pattern(ant,18.3e9)
```



```
      Output : Directivity
   Frequency : 18.3 GHz
   Max value : 32.2 dBi
   Min value : -20.1 dBi
     Azimuth : [-180° , 180°]
   Elevation : [-90° , 90°]
```

Create Array-fed Cassegrain Antenna

Create a rectangular array of crossed dipole antennas.

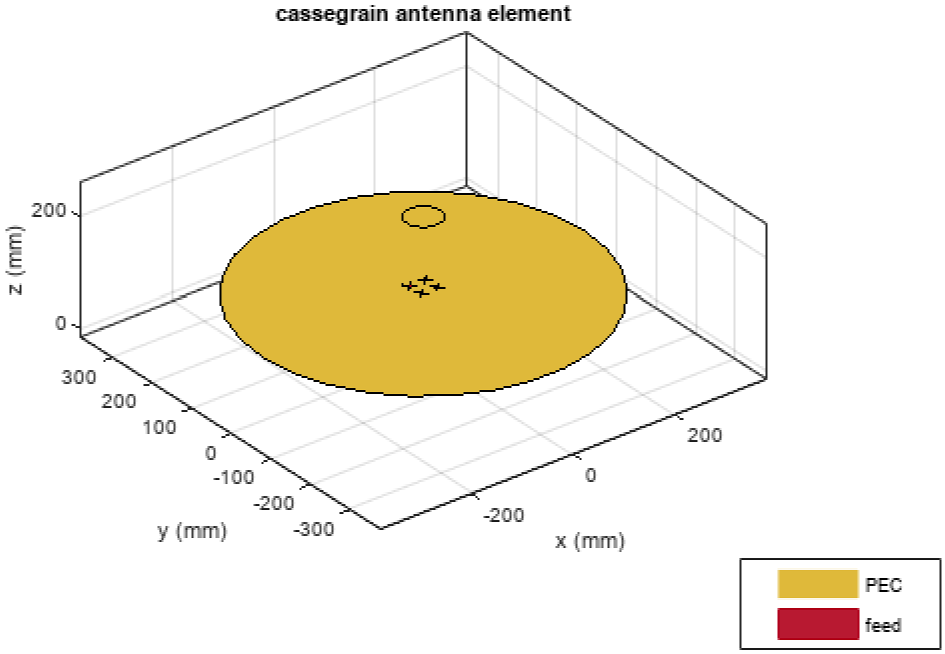
```
e = dipoleCrossed('Tilt',90,'TiltAxis',[0 1 0]);
arr = rectangularArray('Element',e,'Rowspacing',0.03,'ColumnSpacing',0.03);
```

Use the rectangular array `arr` to excite a Cassegrain antenna.

```
ant = cassegrain('Exciter',arr)
```

```
ant = 
  cassegrain with properties:

        Exciter: [1x1 rectangularArray]
         Radius: [0.3175 0.0330]
    FocalLength: [0.2536 0.1416]
           Tilt: 0
      TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
     SolverType: 'MoM-PO'
```

```
show(ant)
```



More About

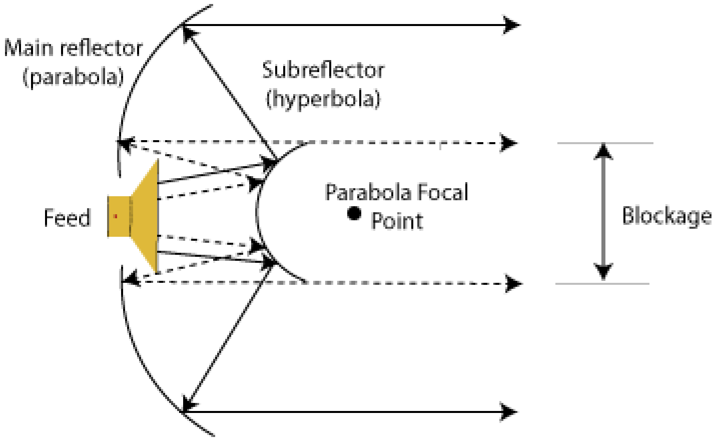
Parabolic Reflector Antennas

A typical parabolic antenna consists of a parabolic reflector with a small feed antenna at its focus. Parabolic reflectors used in dish antennas have a large curvature and short focal length and the focal point is located near the mouth of the dish, to reduce the length of the supports required to hold the feed structure. In more complex designs, such as the cassegrain antenna, a sub reflector is used to direct the energy into the parabolic reflector from a feed antenna located away from the primary focal point. Cassegrain provides an option to increase focal length, reducing side lobes. Such type of antennas can be used in satellite communications and Astronomy and other emerging modes of communications

Architecture of Cassegrain Antenna

Cassegrain antenna consists of three structures:

- Primary parabolic reflector
- Hyperbolic concave subreflector
- Exciter element



Focus of the main reflector and the near focus of the subreflector coincides. The energy is transmitted from the subreflector to the primary parabolic reflector. The parabolic reflector converts a spherical wavefront into a plane wavefront as the energy directed towards it appears to be coming from focus.

Cassegrain Antenna in Receive Mode

In the receive mode, consider that energy in the form of parallel waves is incident up on the reflector system. This energy is intercepted by the main reflector, a large concave surface,and reflected towards the subreflector. The convex surface of the subreflector collects this energy and directs it towards the vertex of the main dish. If the rays directed towards this main dish are parallel, then the main reflector is parabolic and the subreflector is hyperbolic and the rays will focus on a single point. You then place the receiver at this focusing point.

Cassegrain Antenna in Transmit Mode

In the transmit mode, repeat the experiment to find the focusing point as in the receive mode. Place the feed at the focusing point. The feed is usually small and the sub reflector is in the far-field region of the feed. The size of the subreflector is large enough that it intercepts most of the radiation from the feed point. Because of the geometry and the shape of the main reflector and the subreflector the rays from the main dish are usually parallel.

Version History

Introduced in R2019b

References

[1] Dandu, Obulesu. "Optimized Design of Axillary Symmetric Cassegrain Reflector Antenna Using Iterative Local Search Algorithm"

[2] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

See Also

`reflectorParabolic` | `hornConical` | `gregorian`

Topics

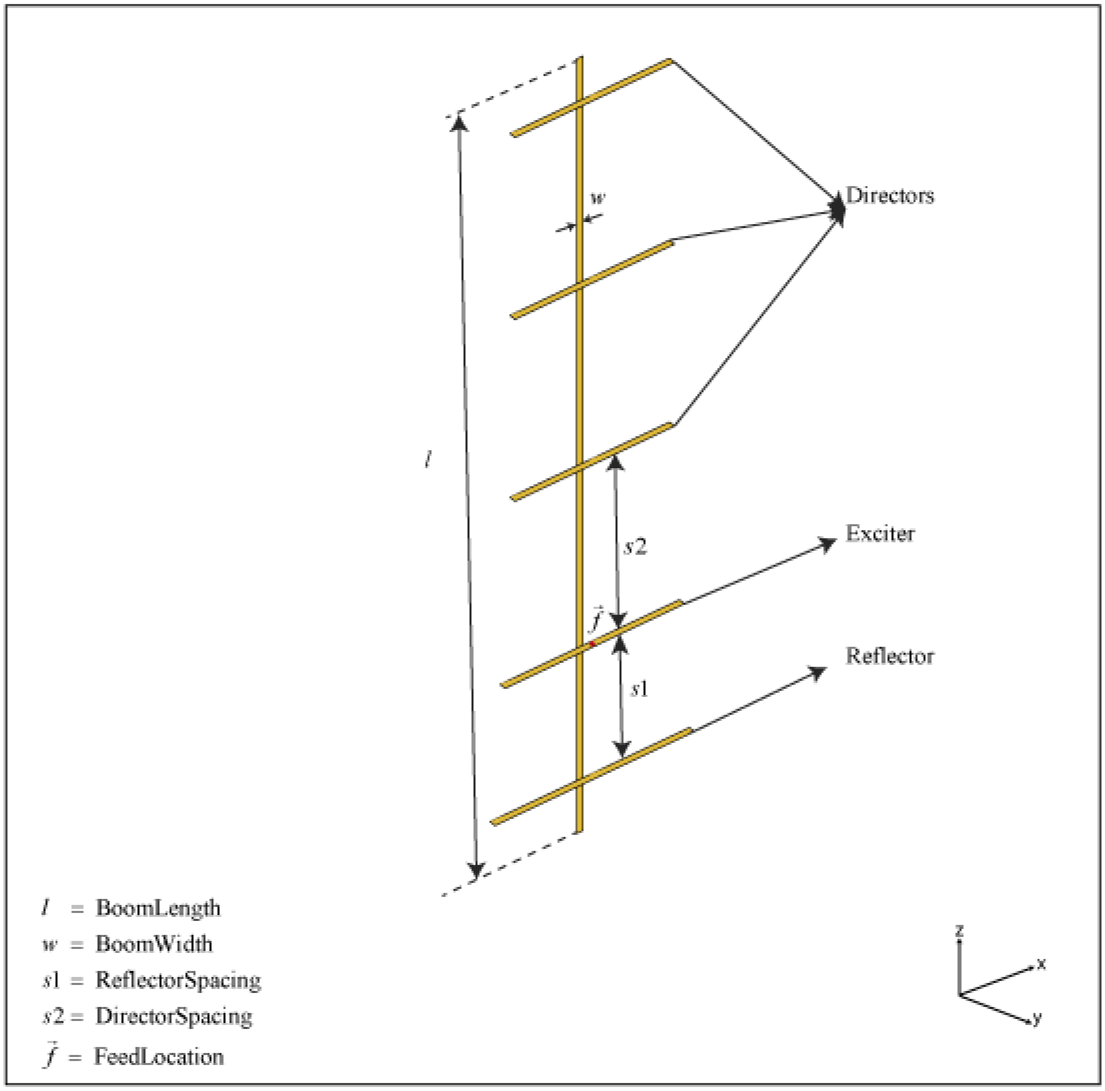
"Rotate Antennas and Arrays"

quadCustom

Create Yagi-Uda custom array antenna

Description

The `quadCustom` object creates a Yagi-Uda custom array along the *z*-axis.



Creation

Syntax

```
ant = quadCustom
ant = quadCustom(Name,Value)
```

Description

`ant = quadCustom` creates a half-wavelength Yagi-Uda custom array antenna along the *z*-axis. The default antenna is excited using a dipole and consists of three directors and one reflector. The default dimensions are chosen for an operating frequency of 2.4 GHz.

`ant = quadCustom(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = quadCustom('Exciter',dipoleFolded)` creates a Yagi-Uda custom array antenna with a folded dipole antenna as the exciter.

Properties

`Exciter` — Antenna type used as exciter

`dipole` (default) | antenna object

Antenna type used as an exciter, specified as a `dipoleFolded`, `biquad`, `dipole`, or `loopCircular` antenna object. This `quadCustom` supports a single exciter.

Example: `'Exciter',dipoleFolded`

Example: `ant.Exciter = dipoleFolded`

`Director` — Antenna type or antenna shape used as director elements

array of three `dipole` antennas (default) | cell array of one or more antenna objects

Antenna type or antenna shape used as director elements, specified as a cell array consisting of one or more of the following antennas: `dipole`, `dipoleVee`, `biquad`, `loopRectangular`, `loopCircular`, `antenna.Polygon`, `antenna.Circle`, or `antenna.Rectangle`. You can use single or multiple antenna elements as directors.

Example: `d = dipoleVee; ant = quadCustom('Director',{d d d d})`. Yagi-Uda custom array antenna uses V-dipole as its directors.

Example: `d = dipoleVee; ant = quadCustom; ant.Director= {d d d d}` . Yagi-Uda custom array antenna uses V-dipole as its directors.

`DirectorSpacing` — Spacing between director elements

`0.0423` (default) | real-valued scalar | vector

Spacing between the director elements, specified as a real-valued scalar in meters or a vector with each element unit in meters. You can specify a scalar value for equal spacing between the elements and vector value for unequal spacing between the elements. If you use a vector, the first value is the distance between the exciter and the first director element.

Example: `'DirectorSpacing',[0.234 0.324]`

Example: `ant.DirectorSpacing = [0.234 0.324]`

Data Types: `double`

`Reflector` — Antenna type used as reflector elements

`dipole` (default) | cell array of one or more antenna objects

Antenna type used as reflector elements, specified as a cell array. You can use single or multiple antenna elements as reflectors.

Example: `d = dipoleVee;ant = quadCustom('Reflector',{d d d d})` Yagi-Uda custom array antenna uses V- dipole as its reflectors.

Example: `d = dipoleVee;ant = quadCustom;ant.Reflector={d d d d}` Yagi-Uda custom array antenna uses V- dipole as its reflectors.

`ReflectorSpacing` — Spacing between reflector elements

`0.0423` (default) | real-valued scalar | vector

Spacing between the reflector elements, specified as a real-valued scalar in meters or a vector with each element unit in meters. You can specify a scalar value for equal spacing between the elements or a vector value for unequal spacing between the elements. If you use a vector, the first value is the distance between the exciter and the first reflector element.

Example: `'ReflectorSpacing',[0.234 0.324]`

Example: `ant.ReflectorSpacing = [0.234 0.324]`

Data Types: `double`

`BoomLength` — Length of boom

`0.1800` (default) | real-valued scalar

Length of the boom, specified as a real-valued scalar in meters.

Example: `'BoomLength',0.234`

Example: `ant.BoomLength = 0.234`

Data Types: `double`

`BoomWidth` — Width of boom

`0.0020` (default) | real-valued scalar

Width of the boom, specified as a real-valued scalar in meters.

Example: `'BoomWidth',0.00324`

Example: `ant.BoomWidth = 0.00324`

Data Types: `double`

`BoomOffset` — Signed distance from center of antenna elements

`[0 0.0050 0.0450]` (default) | three-element vector

Signed distance from center of antenna elements, specified as a three-element vector with each element unit in meters.

Example: `'BoomOffset',[0 0.0060 0.0350]`

Example: `ant.BoomOffset = [0 0.0060 0.0350]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements
[1x1 lumpedElement] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object for the load created using lumpedElement.

Example: ant.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna
0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna
[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: TiltAxis=[0 1 0]

Example: TiltAxis=[0 0 0;0 1 0]

Example: TiltAxis = 'Z'

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

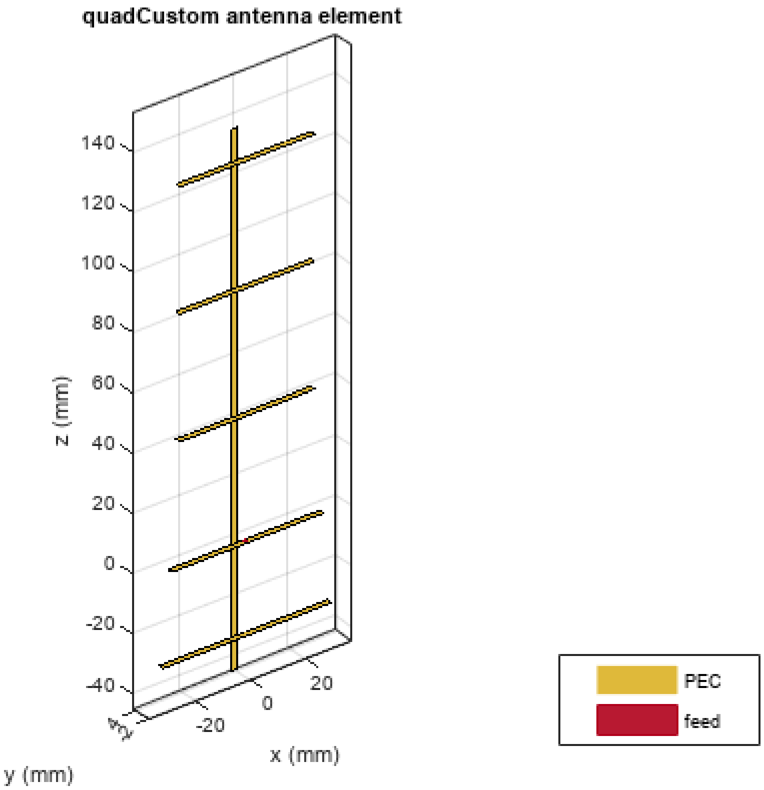
Default Custom Yagi-Uda Array Antenna (quadCustom) and Radiation Pattern

Create and view a custom Yagi-Uda array antenna.

```
ant = quadCustom
```

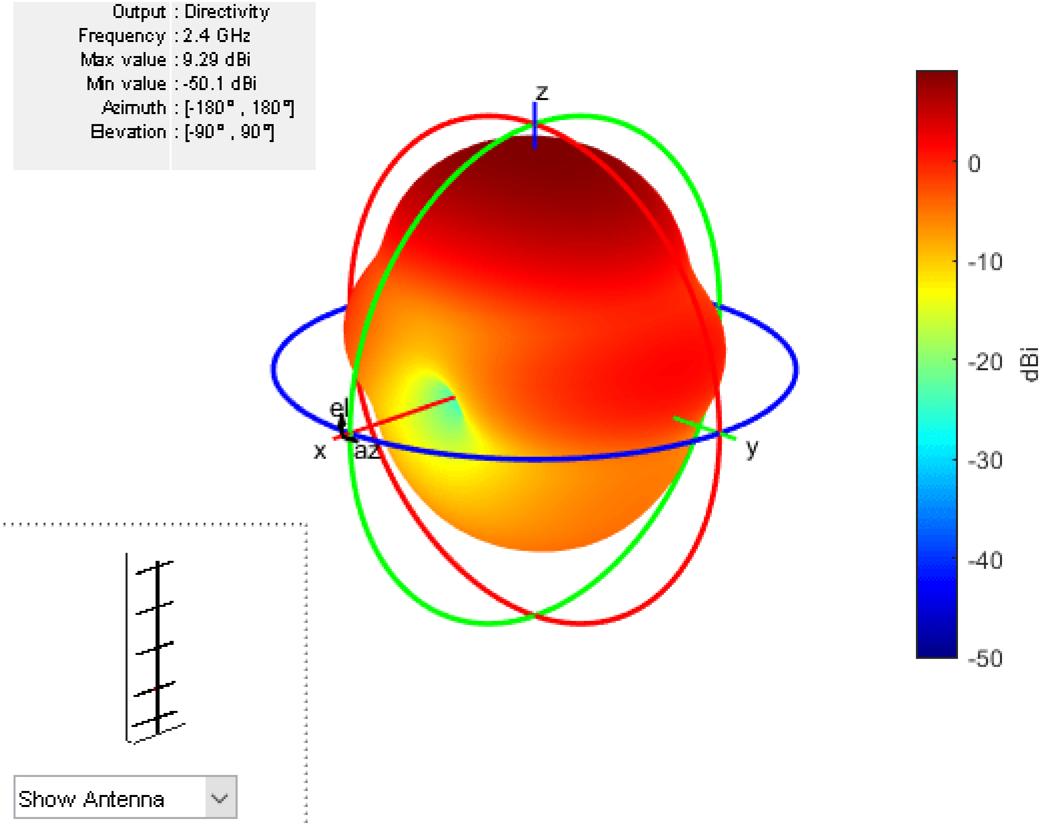
```
ant = 
  quadCustom with properties:

            Exciter: [1x1 dipole]
           Director: {[1x1 dipole]  [1x1 dipole]  [1x1 dipole]}
     DirectorSpacing: 0.0423
          Reflector: {[1x1 dipole]}
    ReflectorSpacing: 0.0308
         BoomLength: 0.1800
          BoomWidth: 0.0020
         BoomOffset: [0 0.0050 0.0450]
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 2.4 GHz.

```
pattern(ant,2.4e9)
```



Custom Yagi-Uda Array Antenna with Seven Directors

Create the default quadCustom, change the number of directors to seven, and view the structure.

```
 ant = design(dipole,2.4e9);
 ant.Tilt = 90
```

```
ant = 
  dipole with properties:

        Length: 0.0587
         Width: 0.0012
    FeedOffset: 0
     Conductor: [1x1 metal]
          Tilt: 90
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

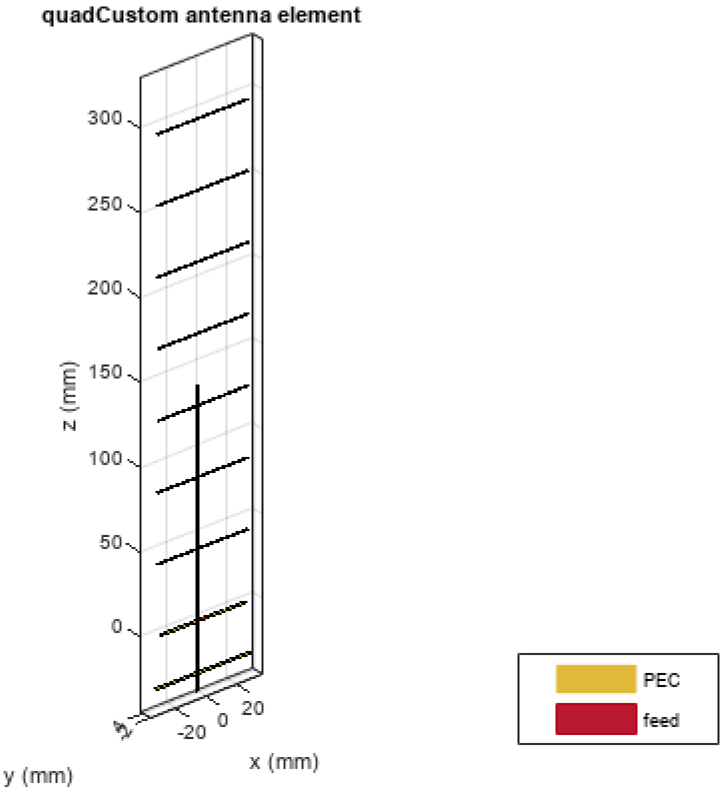
```
 ant.TiltAxis = [0 1 0]
```

```
ant = 
  dipole with properties:

        Length: 0.0587
        Width: 0.0012
   FeedOffset: 0
    Conductor: [1x1 metal]
         Tilt: 90
     TiltAxis: [0 1 0]
         Load: [1x1 lumpedElement]
```

```
 quad_ant = quadCustom('Director',{ant,ant,ant,ant,ant,ant,ant})
```

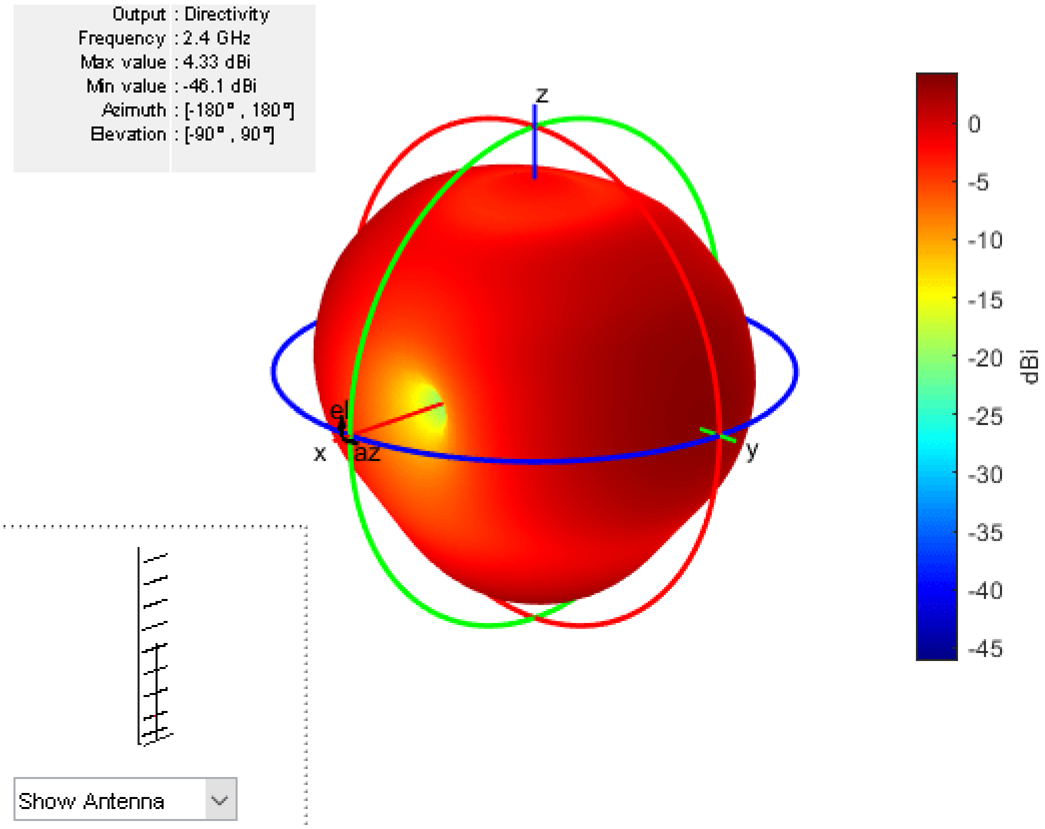
```
quad_ant = 
  quadCustom with properties:

              Exciter: [1x1 dipole]
             Director: {1x7 cell}
      DirectorSpacing: 0.0423
            Reflector: {[1x1 dipole]}
     ReflectorSpacing: 0.0308
           BoomLength: 0.1800
            BoomWidth: 0.0020
           BoomOffset: [0 0.0050 0.0450]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
 show(quad_ant)
```



Plot the radiation pattern of the antenna at the frequency 2.4 GHz.

```
pattern(quad_ant,2.4e9)
```



Output : Directivity
Frequency : 2.4 GHz
Max value : 4.33 dBi
Min value : -46.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2019b

References

[1] Bankey, Vinay, and N.Anvesh Kumar. "Design of a Yagi-Uda Antenna with Gain and Bandwidth Enhancement for Wi-Fi and Wi-Max Applications." *International Journal of Antennas*. Vol.2, Number 1, 2017

See Also

`cavityCircular`

Topics

“Rotate Antennas and Arrays”

antenna.Ellipse

Create ellipse centered at origin on X-Y plane

Description

Use the `antenna.Ellipse` object to create an ellipse centered at the origin on the X-Y plane.

Creation

Syntax

```
ellipse = antenna.Ellipse
ellipse = antenna.Ellipse(Name,Value)
```

Description

`ellipse = antenna.Ellipse` creates an ellipse centered at the origin on the X-Y plane.

`ellipse = antenna.Ellipse(Name,Value)` sets properties using one or more name-value pair arguments. For example, `ellipse = antenna.Ellipse('MajorAxis',2,'Minoraxis',0.800)` creates an ellipse with a longest diameter of 2 m and smallest diameter of 0.8 m. Enclose each property name in quotes.

Properties

`Name` — Name of ellipse
`'myEllipse'` (default) | character vector

Name of ellipse, specified as a character vector.

Example: `'Name','ellipse1'`

Example: `ellipse.Name= 'ellipse1'`

Data Types: `char` | `string`

`Center` — Cartesian coordinates of center of ellipse
`[0 0]` (default) | two-element vector

Cartesian coordinates of center of ellipse, specified as a two-element vector with each element measured in meters.

Example: `'Center',[0.006 0.006]`

Example: `Ellipse.Center= [0.006 0.006]`

Data Types: `double`

`Major axis` — Major axis of ellipse
1 (default) | scalar

Major axis of ellipse, specified as a scalar in meters.

Example: `'MajorAxis',1`

Example: `ellipse.MajorAxis= 2`

Data Types: `double`

`Minor axis` — Minor axis of ellipse

0.5 (default) | scalar

Minor axis of the ellipse, specified as a scalar in meters.

Example: `'MinorAxis',0.9`

Example: `ellipse.MinorAxis= 0.8`

Data Types: `double`

`NumPoints` — Number of discretization points on circumference

30 (default) | scalar

Number of discretization points on circumference, specified as a scalar.

Example: `'NumPoints',28`

Example: `ellipse.NumPoints= 60`

Data Types: `double`

Object Functions

add	Boolean unite operation on two shapes
subtract	Boolean subtraction operation on two shapes
intersect	Boolean intersection operation on two shapes
plus	Shape1 + Shape2
minus	Shape1 - Shape2
and	Shape1 & Shape2
area	Calculate area of shape in square meters
show	Display antenna, array structures or shapes
plot	Plot boundary of shape
mesh	Mesh properties of metal, dielectric antenna, or array structure
rotate	Rotate shape about axis and angle
rotateX	Rotate shape about x-axis and angle
rotateY	Rotate shape about y-axis and angle
rotateZ	Rotate shape about z-axis and angle
translate	Move shape to new location
scale	Change the size of the shape by a fixed amount
removeHoles	Remove holes from shape
removeSlivers	Remove sliver outliers from boundary of shape

Examples

Create an Ellipse with Default Properties

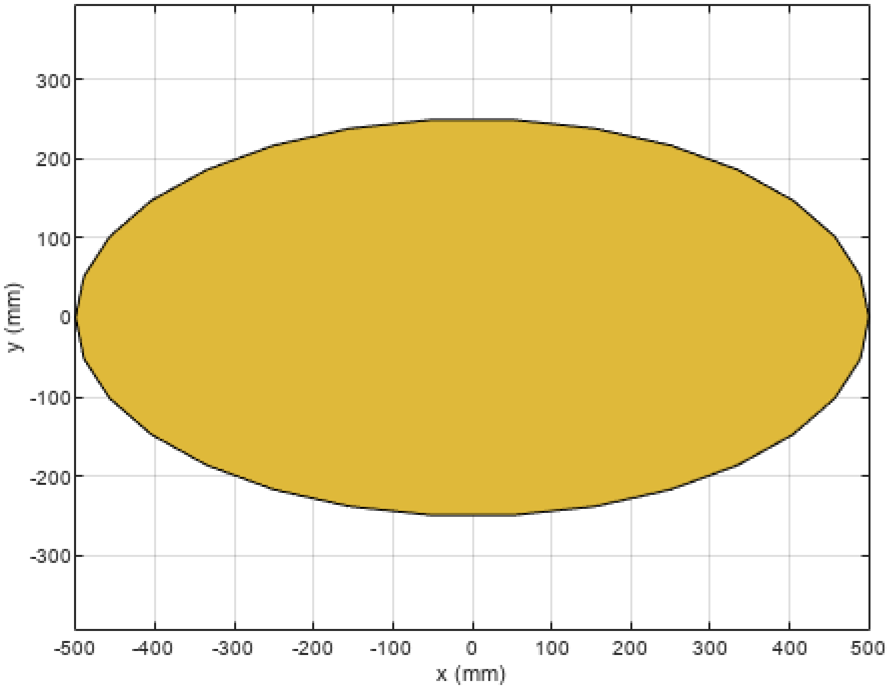
Create ellipse using `antenna.Ellipse` object.

```
e1 = antenna.Ellipse
```

```
e1 = 
  Ellipse with properties:

         Name: 'myEllipse'
       Center: [0 0]
    MajorAxis: 1
    MinorAxis: 0.5000
    NumPoints: 30
```

View the `antenna.Ellipse` object using the `show` function.

```
show(e1)
```



Create an Ellipse with Specified Properties

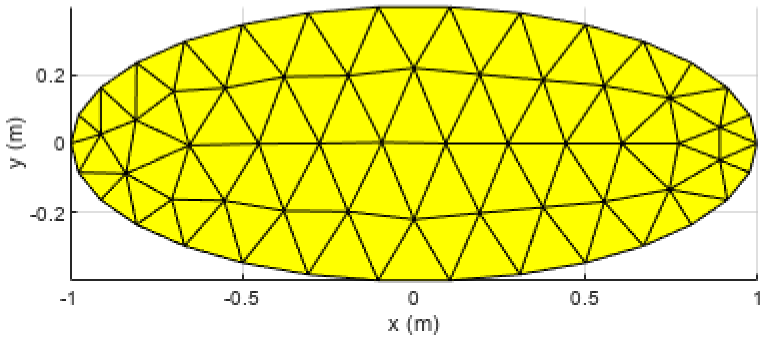
Create an ellipse with major axis of 2 m and a minor axis of 0.8 m.

```
e2 = antenna.Ellipse('MajorAxis',2,'MinorAxis',0.8)
```

```
e2 = 
  Ellipse with properties:
         Name: 'myEllipse'
       Center: [0 0]
    MajorAxis: 2
    MinorAxis: 0.8000
    NumPoints: 30
```

Create a mesh with a Maximum edge Length of 20 cm.

```
mesh(e2,'MaxEdgeLength',2e-1)
```



Subtract Two Shapes

Create an ellipse with default properties.

```
 e3 = antenna.Ellipse;
```

Create a rectangle with a length of 0.1 m and width of 0.2 m.

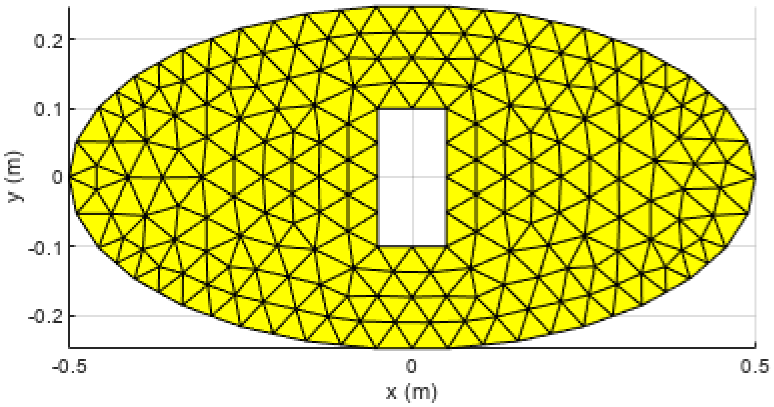
```
 r = antenna.Rectangle('Length',0.1,'Width',0.2);
```

Subtract the two shapes using the `minus` operator.

```
 s = e3-r;
```

Mesh the subtracted shape with a maximum edge length of 1 m.

```
mesh(s,1)
```



Version History

Introduced in R2020a

See Also

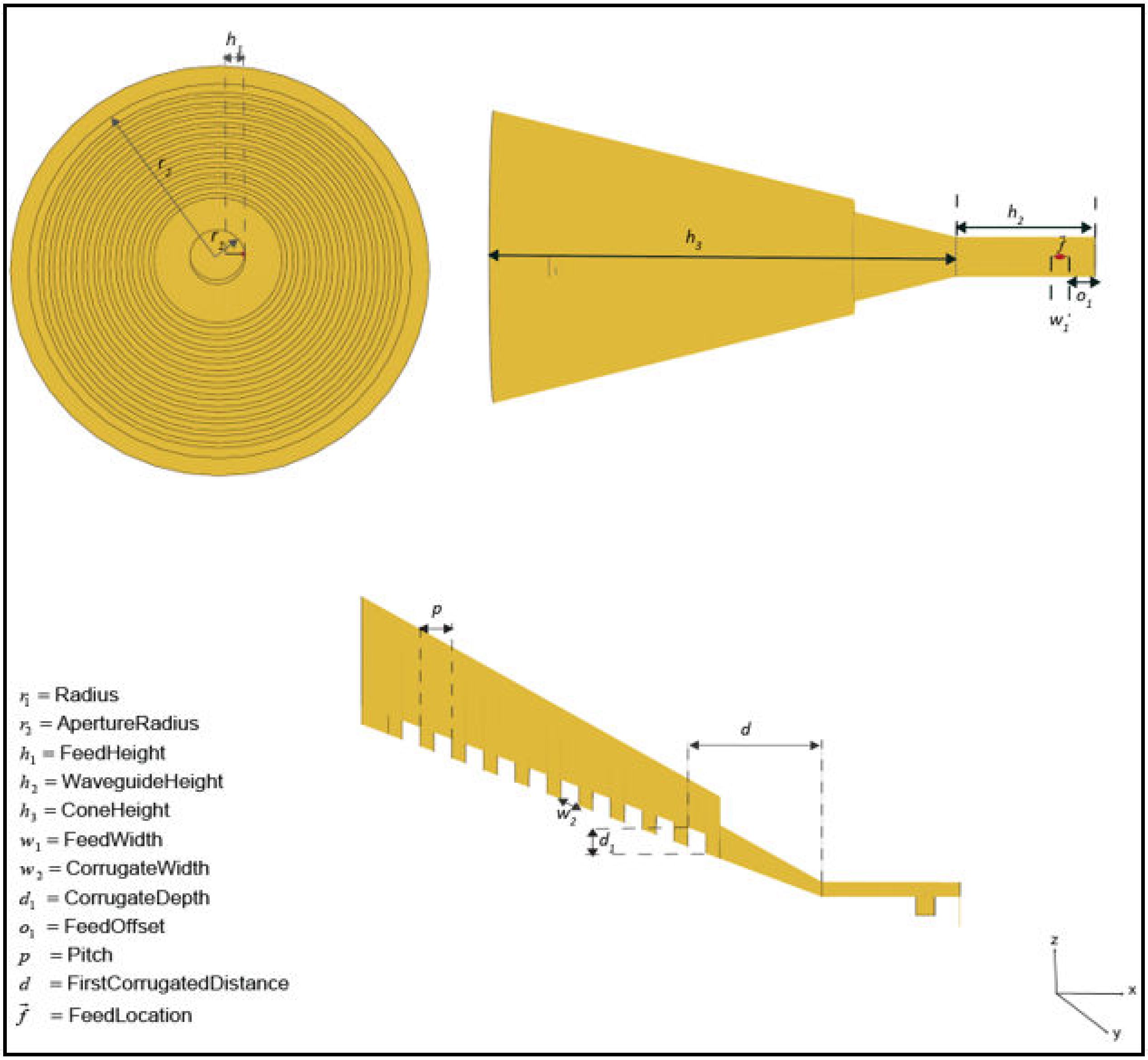
antenna.Polygon | antenna.Rectangle | antenna.Circle

hornConicalCorrugated

Create conical corrugated-horn antenna

Description

The `hornConicalCorrugated` object creates a conical corrugated-horn antenna, with grooves covering the inner surface of the cone. These antennas are widely used as feed horns for dish reflector antennas as they have smaller side lobes and low cross-polarization level.



Creation

Syntax

```
ant = hornConicalCorrugated
ant = hornConicalCorrugated(Name,Value)
```

Description

`ant = hornConicalCorrugated` creates a corrugated conical-horn antenna object with default dimensions for an operating frequency around 9.5 GHz.

`ant = hornConicalCorrugated(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = hornConicalCorrugated('Radius',1)`, creates a conical corrugated-horn antenna with a radius of 1 meter.

Properties

`Radius` — Radius of waveguide
`0.011` (default) | real-valued scalar

Radius of the waveguide, specified as a real-valued scalar in meters.

Example: `'Radius',0.760`

Example: `ant.Radius = 0.760`

Data Types: `double`

`WaveguideHeight` — Height of waveguide
`0.0300` (default) | real-valued scalar

Height of the waveguide, specified as a real-valued scalar in meters.

Example: `'WaveguideHeight',0.0340`

Example: `ant.WaveguideHeight = 0.0340`

Data Types: `double`

`FeedHeight` — Height of feed
`0.0075` (default) | real-valued scalar

Height of the feed, specified as a real-valued scalar in meters.

Example: `'FeedHeight',0.0085`

Example: `ant.FeedHeight = 0.0085`

Data Types: `double`

`FeedWidth` — Width of feed
`0.0040` (default) | real-valued scalar

Width of the feed, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0200`

Example: `ant.FeedWidth = 0.0200`

Data Types: `double`

`FeedOffset` — Signed distance along *y*-axis

`0.0075` (default) | real-valued scalar

Signed distance of the feed along the *y*-axis, specified as a real-valued scalar in meters.

Example: `'FeedOffset',0.03627`

Example: `ant.FeedOffset = 0.3627`

Data Types: `double`

`ConeHeight` — Height of cone

`0.1` (default) | real-valued scalar

Height of the cone, specified as a real-valued scalar in meters.

Example: `'ConeHeight',0.0540`

Example: `ant.ConeHeight = 0.0540`

Data Types: `double`

`ApertureRadius` — Radius of cone aperture

`0.0760` (default) | real-valued scalar

Radius of the cone aperture, specified as a real-valued scalar in meters.

Example: `'ApertureRadius',0.0560`

Example: `ant.ApertureRadius = 0.0790`

Data Types: `double`

`Pitch` — Distance between two successive corrugations

`0.0069` (default) | real-valued scalar

Distance between two successive corrugations, specified as a real-valued scalar in meters.

Example: `'Pitch',0.0060`

Example: `ant.Pitch = 0.0090`

Data Types: `double`

`FirstCorrugatedDistance` — Distance of first corrugation from waveguide

`0.0291` (default) | real-valued scalar

Distance of first corrugation from waveguide, specified as a real-valued scalar in meters.

Example: `'FirstCorrugatedDistance',0.0360`

Example: `ant.FirstCorrugatedDistance = 0.0190`

Data Types: `double`

`CorrugateWidth` — Corrugation width

`0.0039` (default) | real-valued scalar

Corrugation width, specified as a real-valued scalar in meters.

Example: `'CorrugateWidth',0.0058`

Example: `ant.CorrugateWidth = 0.0019`

Data Types: `double`

`CorrugateDepth` — Corrugation depth

`0.0072` (default) | real-valued scalar

Corrugation depth, specified as a real-valued scalar in meters.

Example: `'CorrugateDepth',0.0560`

Example: `ant.CorrugateDepth = 0.0790`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

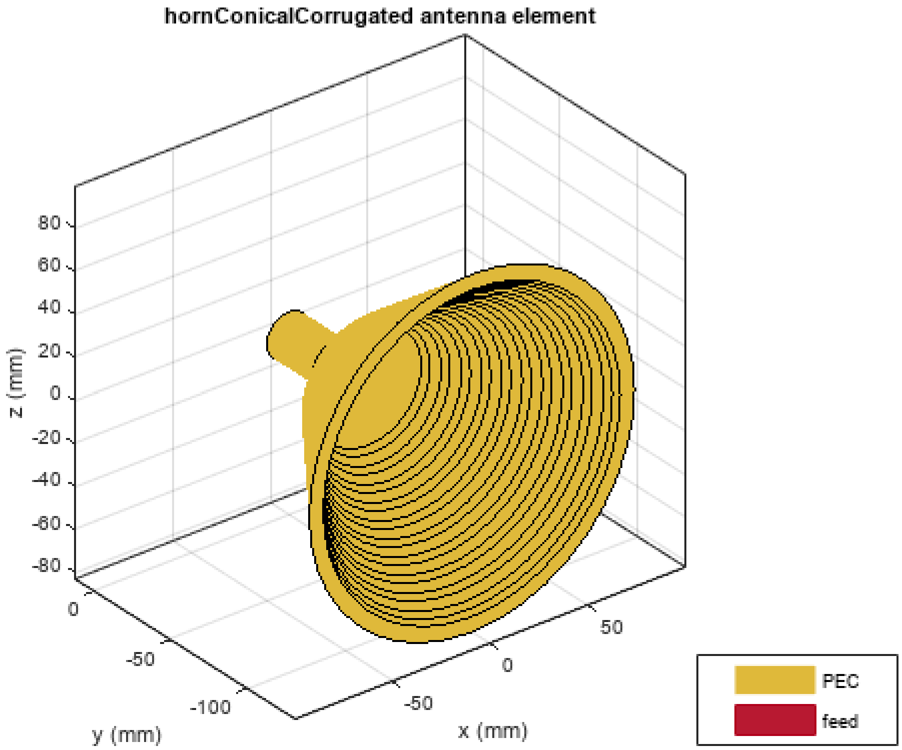
show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create a Conical Corrugated-Horn Antenna

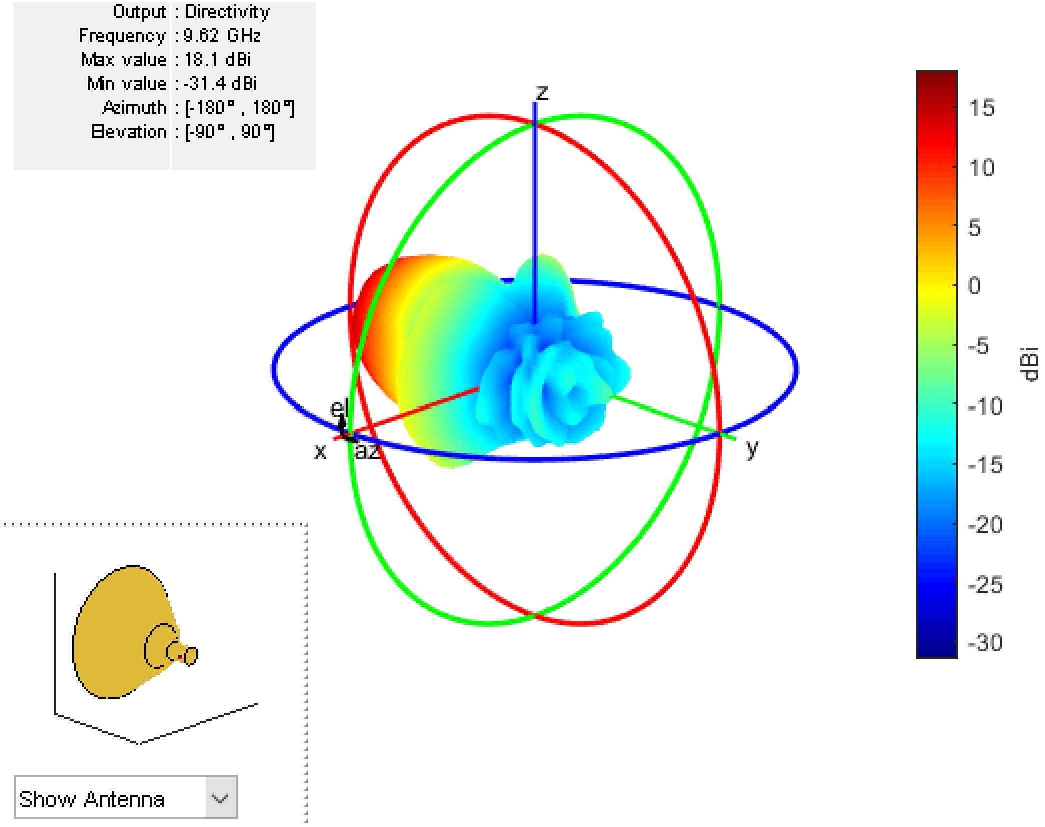
Create a conical corrugated-horn antenna object with the cone height set to 0.09 m

```
ant = hornConicalCorrugated('ConeHeight',0.09);
show(ant)
```



Plot the radiation pattern of the antenna at 9.62 GHz.

```
figure
pattern(ant,9.62e9)
```



Output : Directivity
Frequency : 9.62 GHz
Max value : 18.1 dBi
Min value : -31.4 dBi
Azimuth : [-180°, 180°]
Elevation : [-90°, 90°]

Version History

Introduced in R2020a

References

[1] Jadhav, Rohini.P, Vinothkurnar Javnrakash Dongre, Arunkumar Heddallikar. "Design of X-Band Conical Horn Antenna Using Coaxial Feed and Improved Design Technique for Bandwidth Enhancement". In *International Conference on Computing, Communication, Control, and Automation (ICCUBEA)*, 1-6. Pune, India: ICCUBEA 2017.

See Also

`waveguide` | `horn` | `cavityCircular` | `hornConical` | `hornangle2size`

Topics

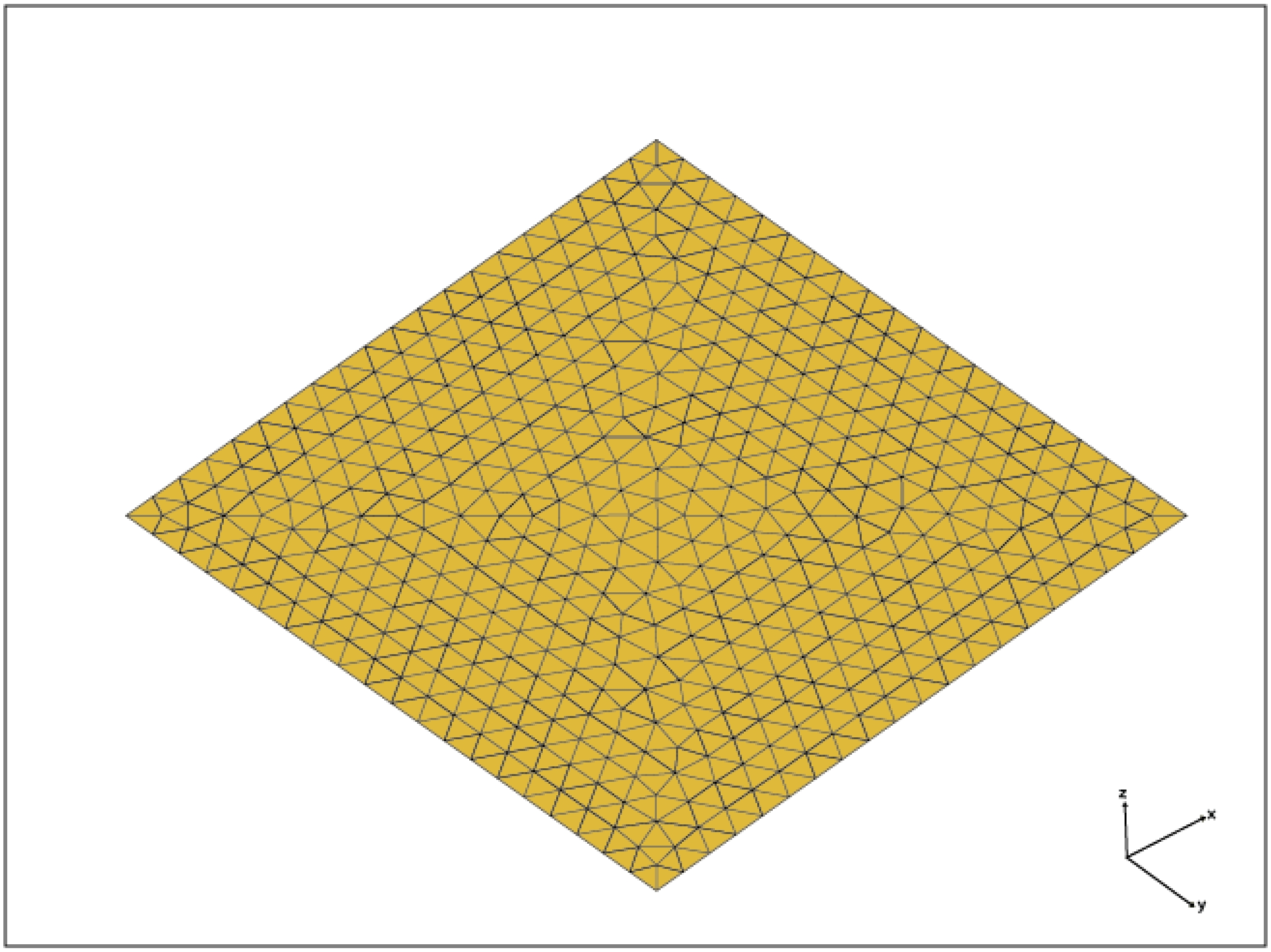
“Rotate Antennas and Arrays”

customAntennaStl

Create custom antenna 3-D geometry using STL files

Description

The `customAntennaStl` object creates a 3-D antenna geometry and mesh using Stereolithography (STL) files. The STL files are used to define any 3-D surface in the form of points and triangles.



Creation

Syntax

```
ca = customAntennaStl
```

Description

`ca = customAntennaStl` returns a 3D antenna represented by a custom geometry, based on the STL file specified.

Properties

`FileName` — Name the STL file

`'[]'` (default) | character vector

Name of the STL file where the structure resides, specified as character vector.

Example: `antenna = customAntennaStl('FileName','plate.stl')`

Example: `antenna = customAntennaStl; antenna.FileName = 'plate.stl'`

Data Types: `char`

`Units` — Units used in STL file

`'m'` (default) | `'mm'` | `'cm'` | `'um'` | `'ft'` | `'in'` | character vector

Units used in STL file, specified as a character vector.

Example: `'Units','mm'`

Data Types: `char`

`FeedLocation` — Antenna feed location in Cartesian coordinates

`[]` (default) | three-element real vector

This property is read-only.

Antenna feed location in Cartesian coordinates, specified as a three-element real vector. The three-element vector are the X-, Y-, and Z-coordinates, respectively.

Note `FeedLocation` property displays the antenna feed location you set using the `createFeed` function.

Data Types: `double`

`AmplitudeTaper` — Excitation amplitude of antenna elements

`1` (default) | scalar double

Excitation amplitude of antenna elements, specified as scalar double.

Example: `'AmplitudeTaper','1.8'`

Data Types: `double`

`PhaseShift` — Phase shift for antenna elements

`0` (default) | scalar

Phase shift for the antenna elements, specified as a scalar in degrees.

Example: `'PhaseShift',10`

Data Types: `double`

UseFileAsMesh — Use stl file as mesh

0 (default) | 1

Use the STL file directly as a mesh for analysis. The value can be either 0 or 1.

Example: 'UseFileAsMesh',1

Data Types: logical

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: TiltAxis=[0 1 0]

Example: TiltAxis=[0 0 0;0 1 0]

Example: TiltAxis = 'Z'

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
createFeed	Create feed location for customAntennaStl object

axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays

Examples

Create and Display Custom 3-D Antenna

Create a custom 3-D antenna using `customAntennaStl` object.

```
c = customAntennaStl('Filename','plateMesh.stl','Units','m');
```

Create antenna feed and calculate the antenna impedance at 110 GHz.

```
c.createFeed([0,0,0],1);
Z = impedance(c,110e6)
```

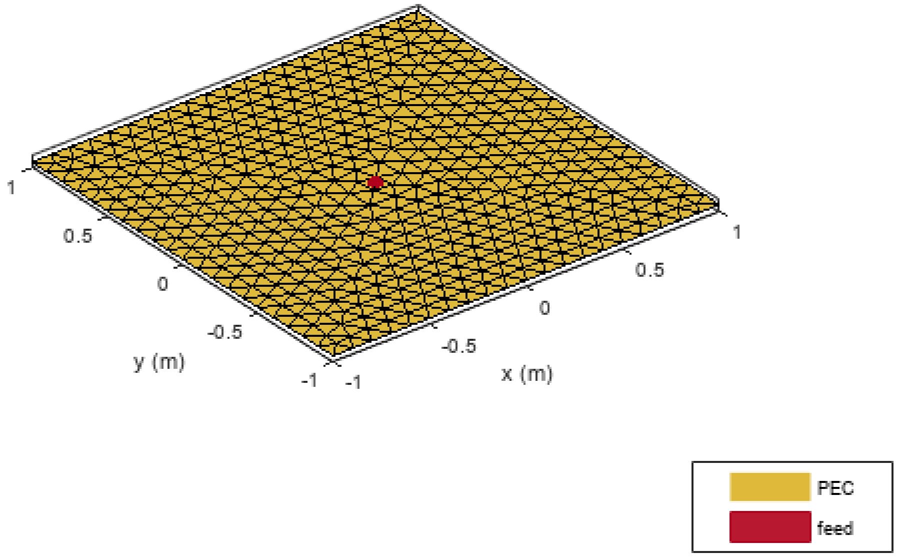
```
Z = 0.0287 + 34.3704i
```

```
disp(c)
```

```
  customAntennaStl with properties:

          FileName: 'plateMesh.stl'
             Units: 'm'
      FeedLocation: [0 0 0]
     AmplitudeTaper: 1
        PhaseShift: 0
      UseFileAsMesh: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

Display the structure of custom 3-D antenna.

```
show(c)
```



Create Antenna Feed in Custom Antenna STL Using Command Line Interface

Create a `customAntennaStl` object using the specified STL file.

```
ant = customAntennaStl
```

```
ant = 
  customAntennaStl with properties:

          FileName: []
             Units: 'm'
      FeedLocation: []
    AmplitudeTaper: 1
        PhaseShift: 0
     UseFileAsMesh: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

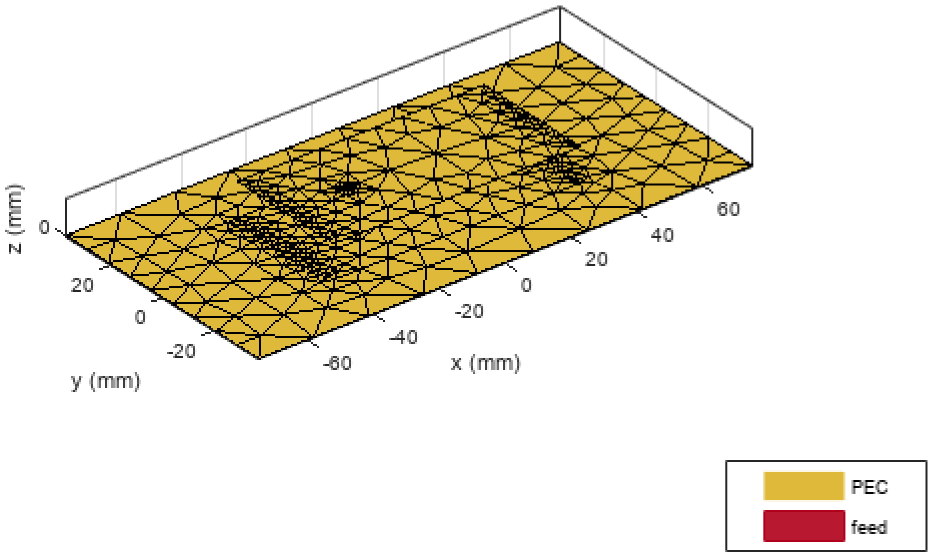
```
ant.FileName ='patchMicrostrip_ColumnFeed.stl'
```

```
ant = 
  customAntennaStl with properties:

          FileName: 'patchMicrostrip_ColumnFeed.stl'
```

```
           Units: 'm'
    FeedLocation: []
  AmplitudeTaper: 1
      PhaseShift: 0
    UseFileAsMesh: 0
            Tilt: 0
        TiltAxis: [1 0 0]
```

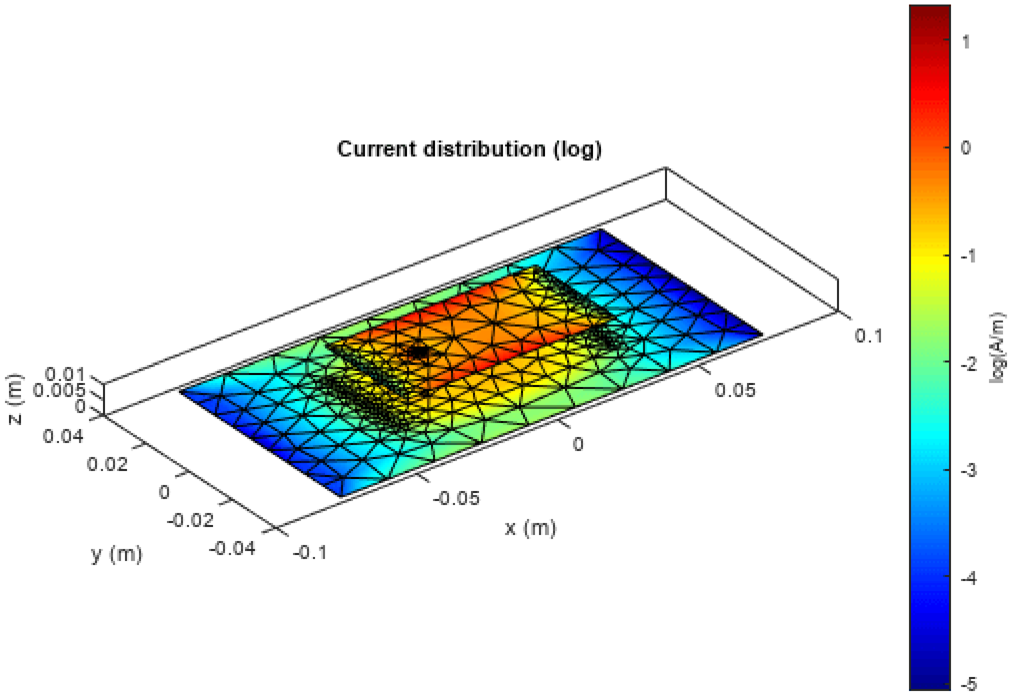
Specify `FeedLocation` and `NumEdges` in the `createFeed` function. The edges are selected based on distance between feed location and midpoints of the edges. Edges can be single feed or a closed polygon.

```
ant.createFeed([-0.018750000000000 0 0],8)
show (ant)
```



Plot the current distribution at 1.75 GHz.

```
figure
current(ant,1.75e9,'Scale','log')
```



Calculate the impedance at 1.75 GHz.

```
z = impedance(ant,1.75e9)
```

```
z = 85.7298 - 52.7332i
```

Create Antenna Feed Using UI Figure Window

Create a `customAntennaStl` object.

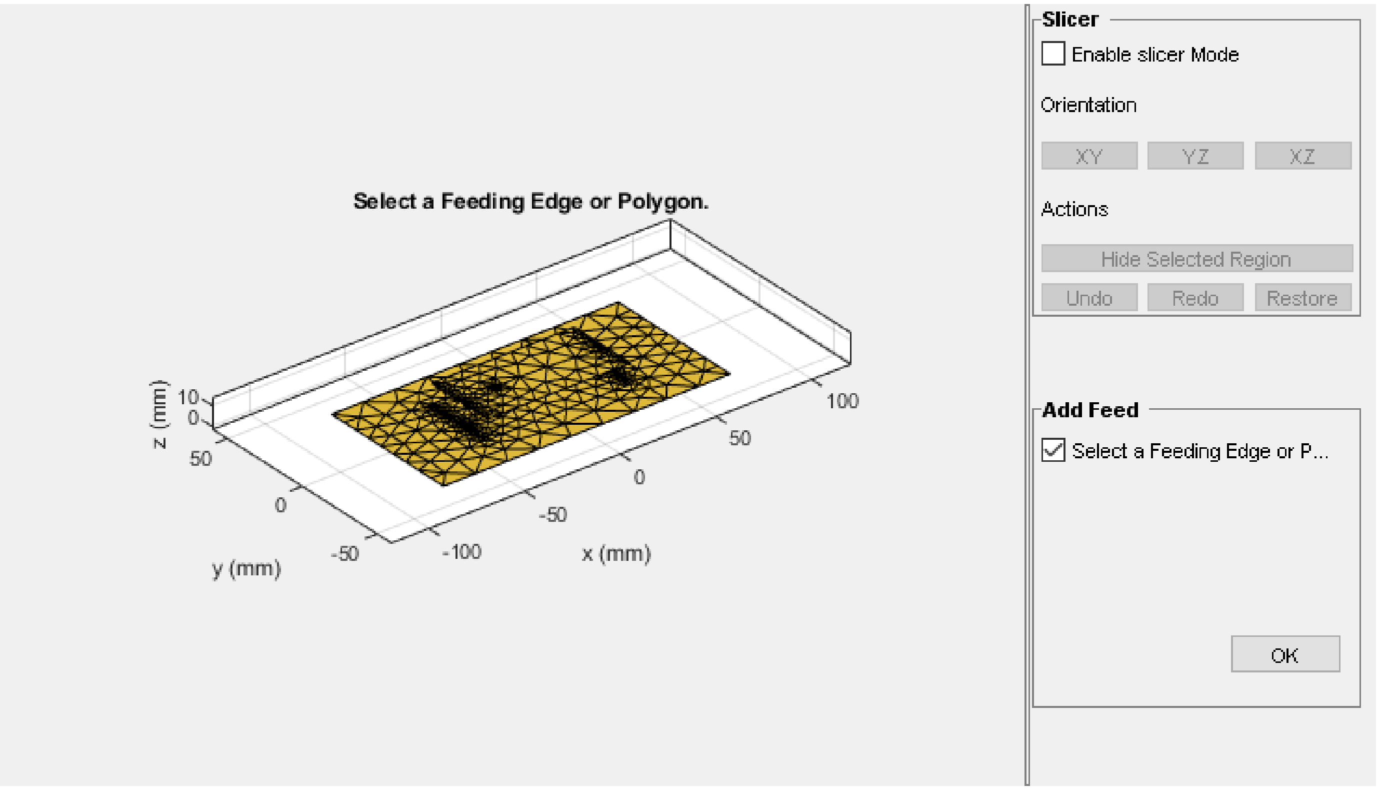
```
ant = customAntennaStl;
```

Import the STL files.

```
ant.FileName = 'patchMicrostrip_ColumnFeed.stl';
```

Create the antenna feed using UI figure window.

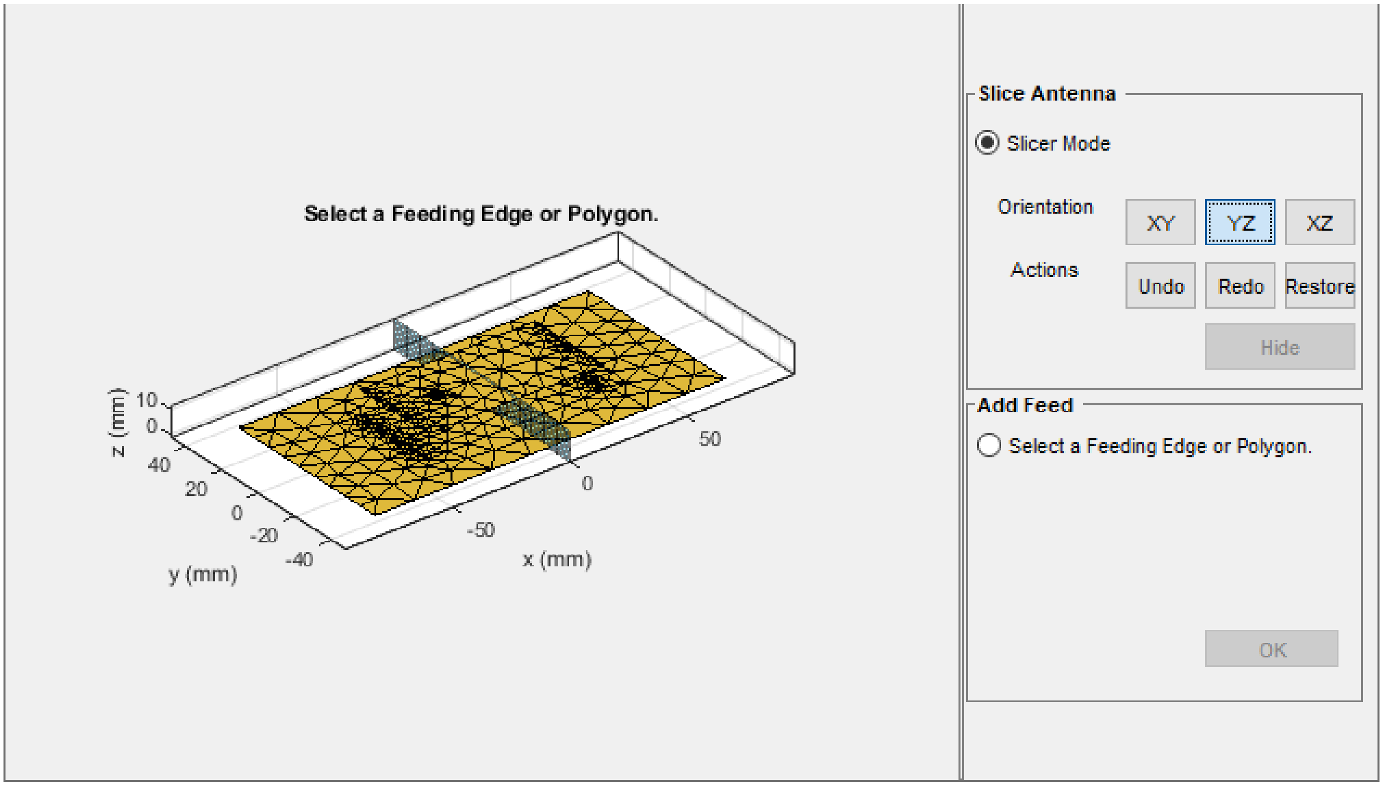
```
createFeed(ant);
```



☐ Enable slicer Mode

☑ Select a Feeding Edge or P...

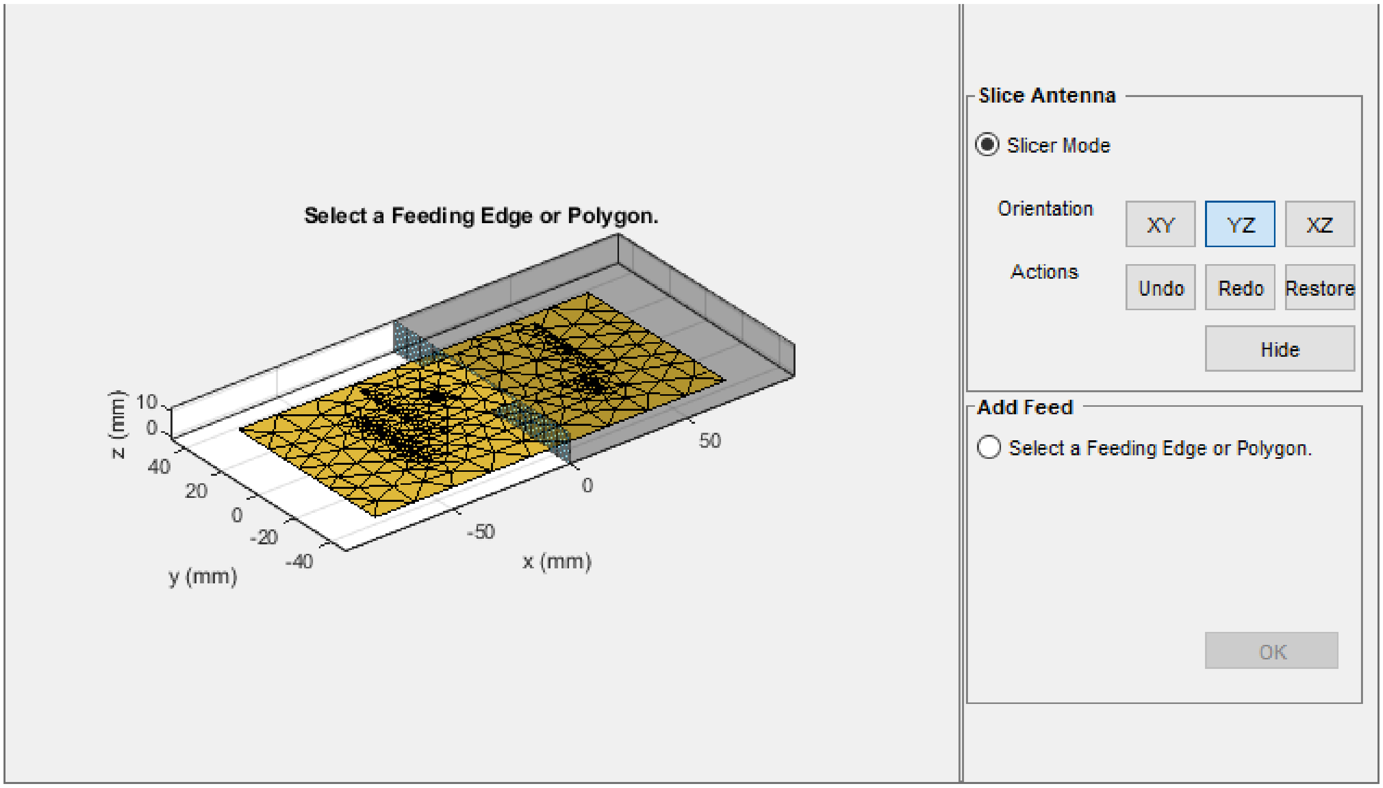
The UI figure window consists of two panes, the **Slice Antenna** panel and the **Add Feed** pane.



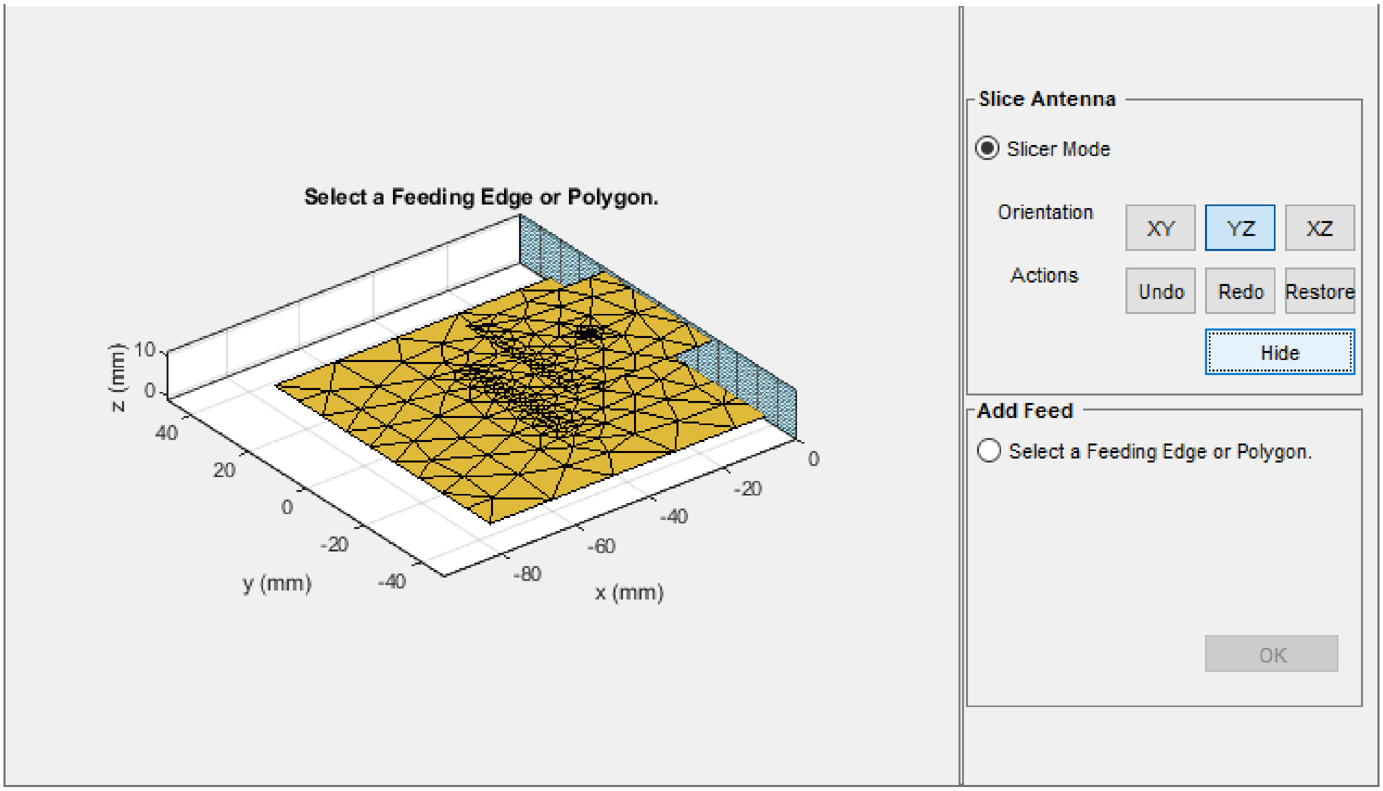
● Slicer Mode

○ Select a Feeding Edge or Polygon.

Click the **Slicer Mode,** then click **YZ** to select that as the plane along which to slice your antenna.



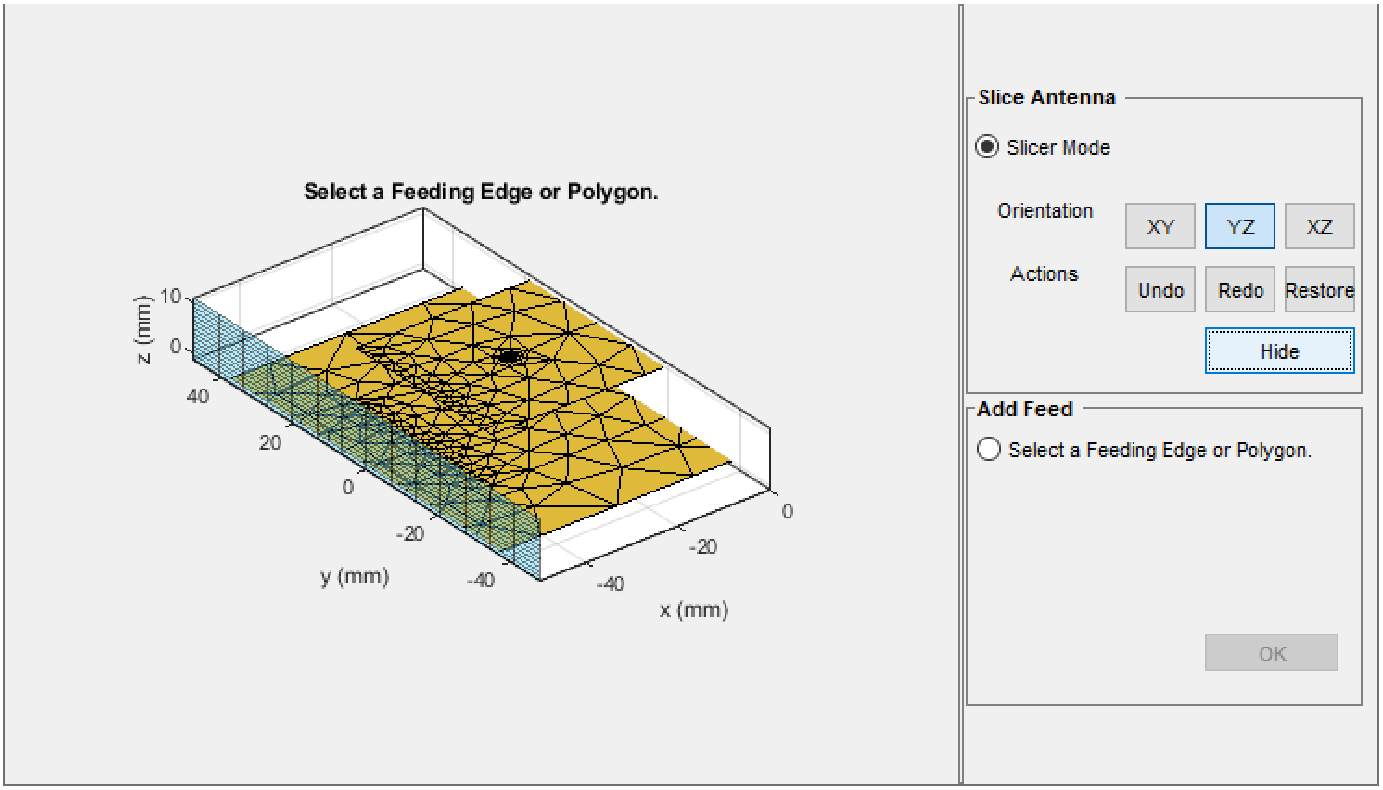
Select the region you want to hide and then click **Hide** to hide the selected region.



● Slicer Mode

○ Select a Feeding Edge or Polygon.

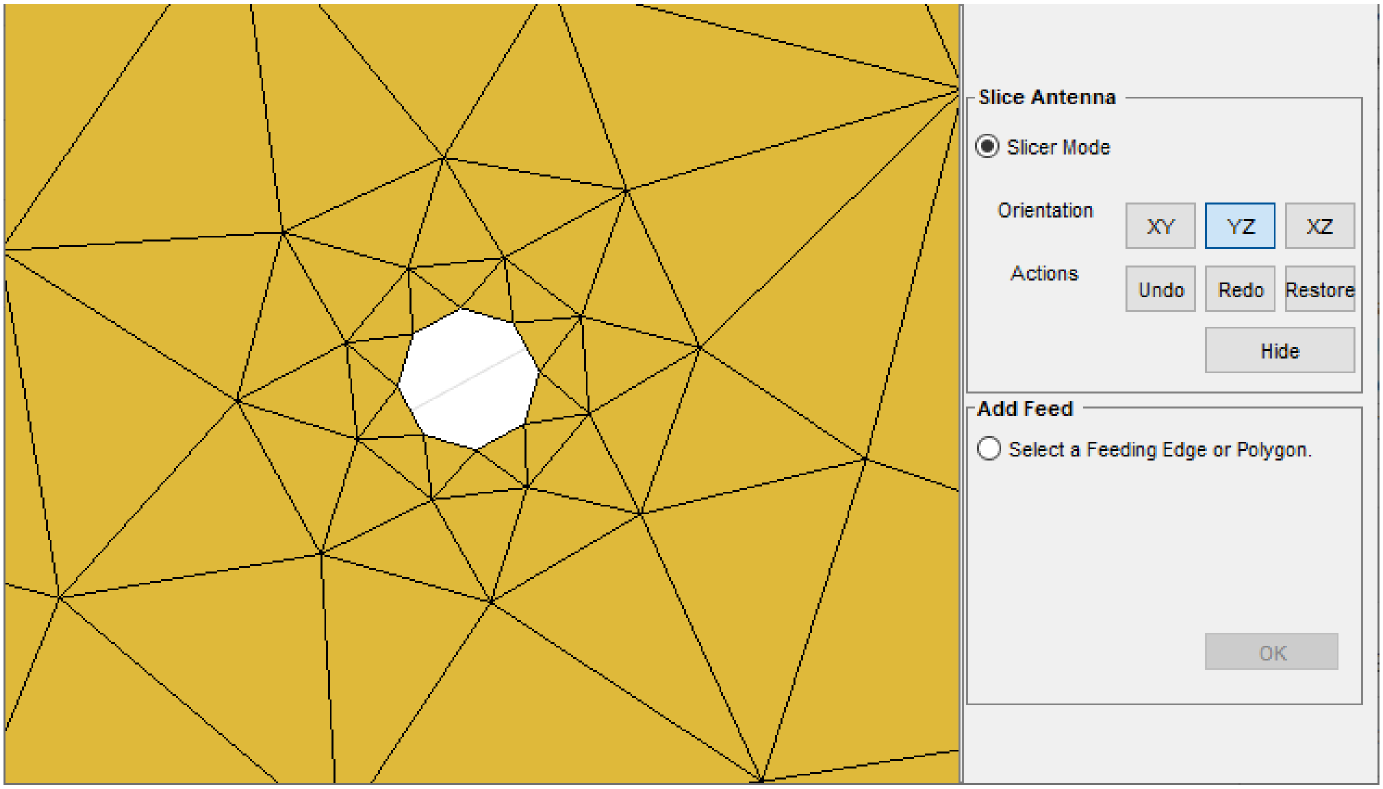
Repeat the process until you reach the region of interest.



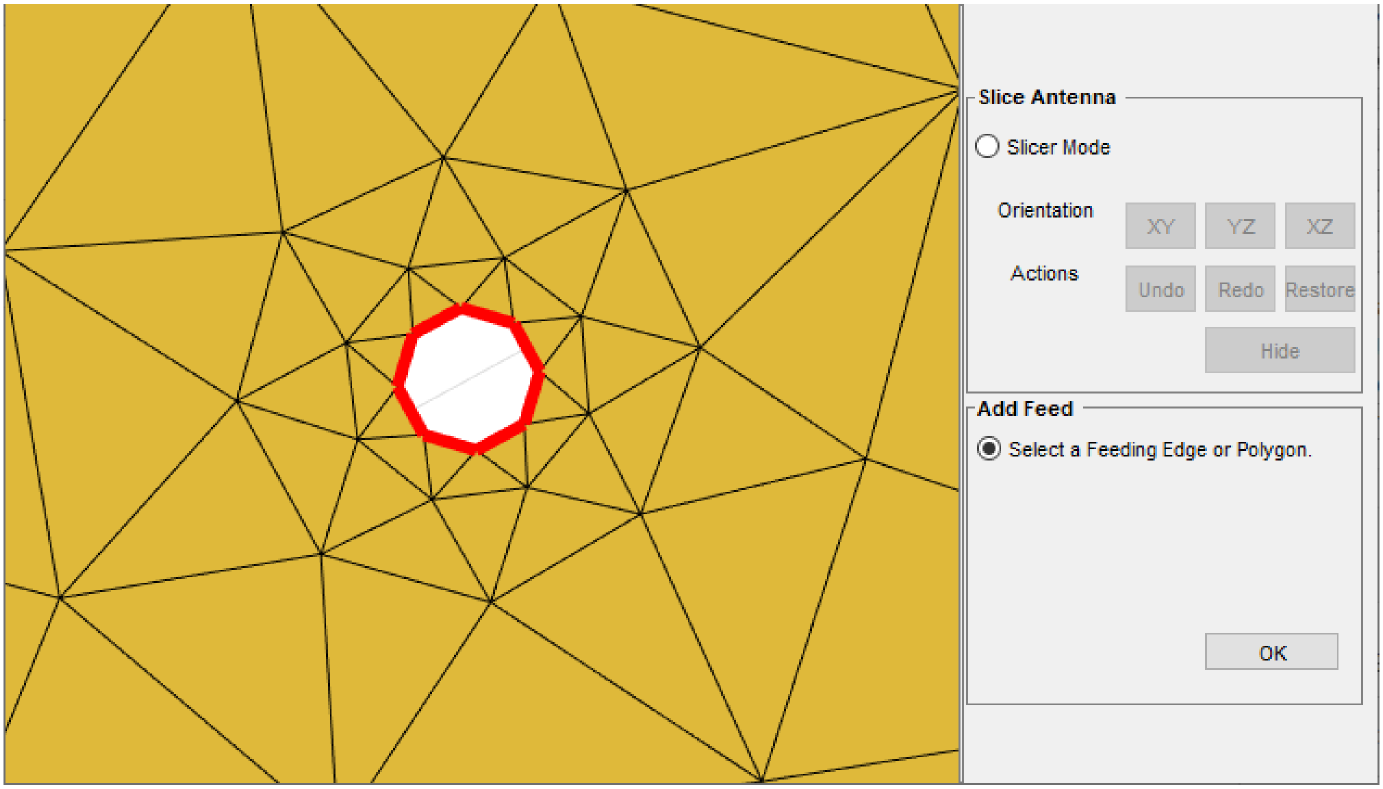
● Slicer Mode

○ Select a Feeding Edge or Polygon.

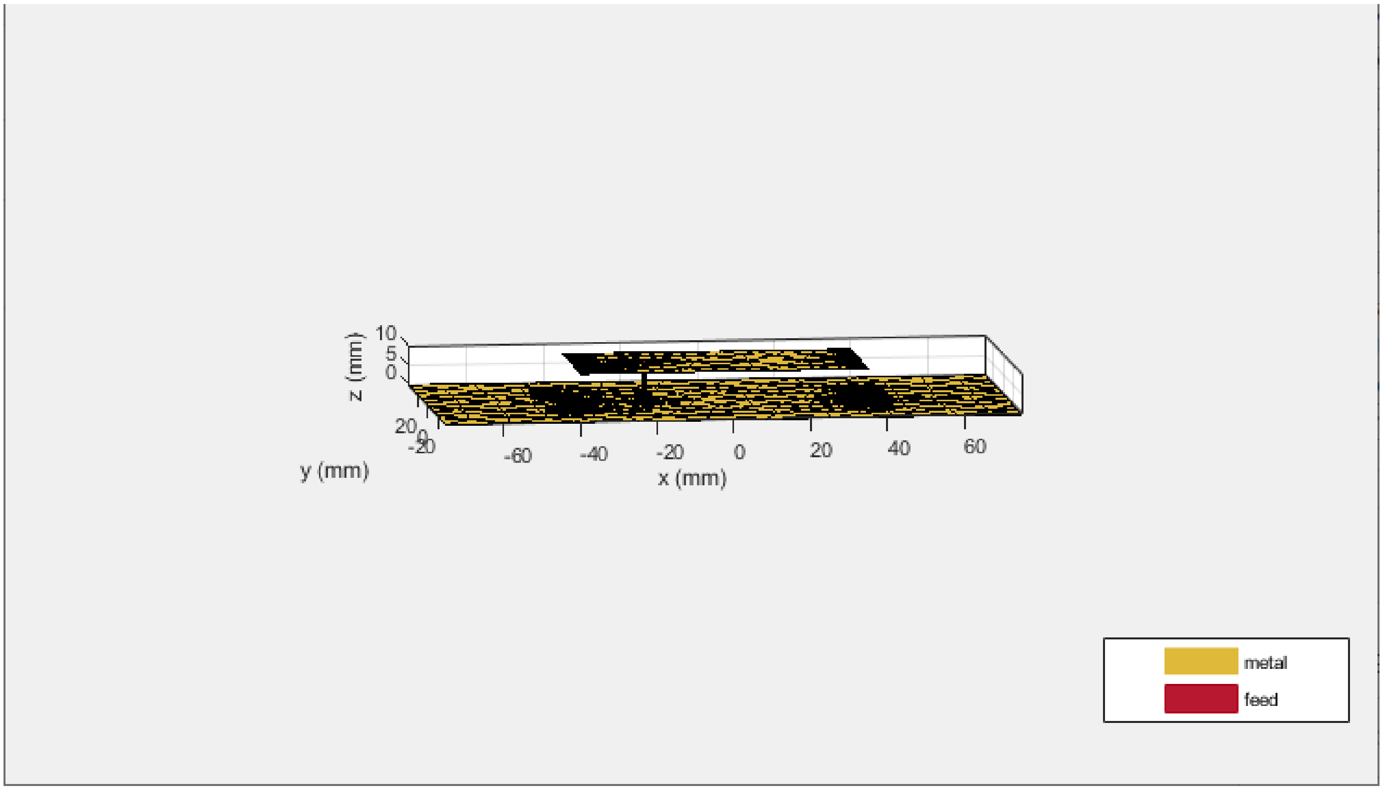
Select **Select a Feeding Edge or Polygon** under the **Add Feed** pane to select the desired feeding edge or feeding polygon.



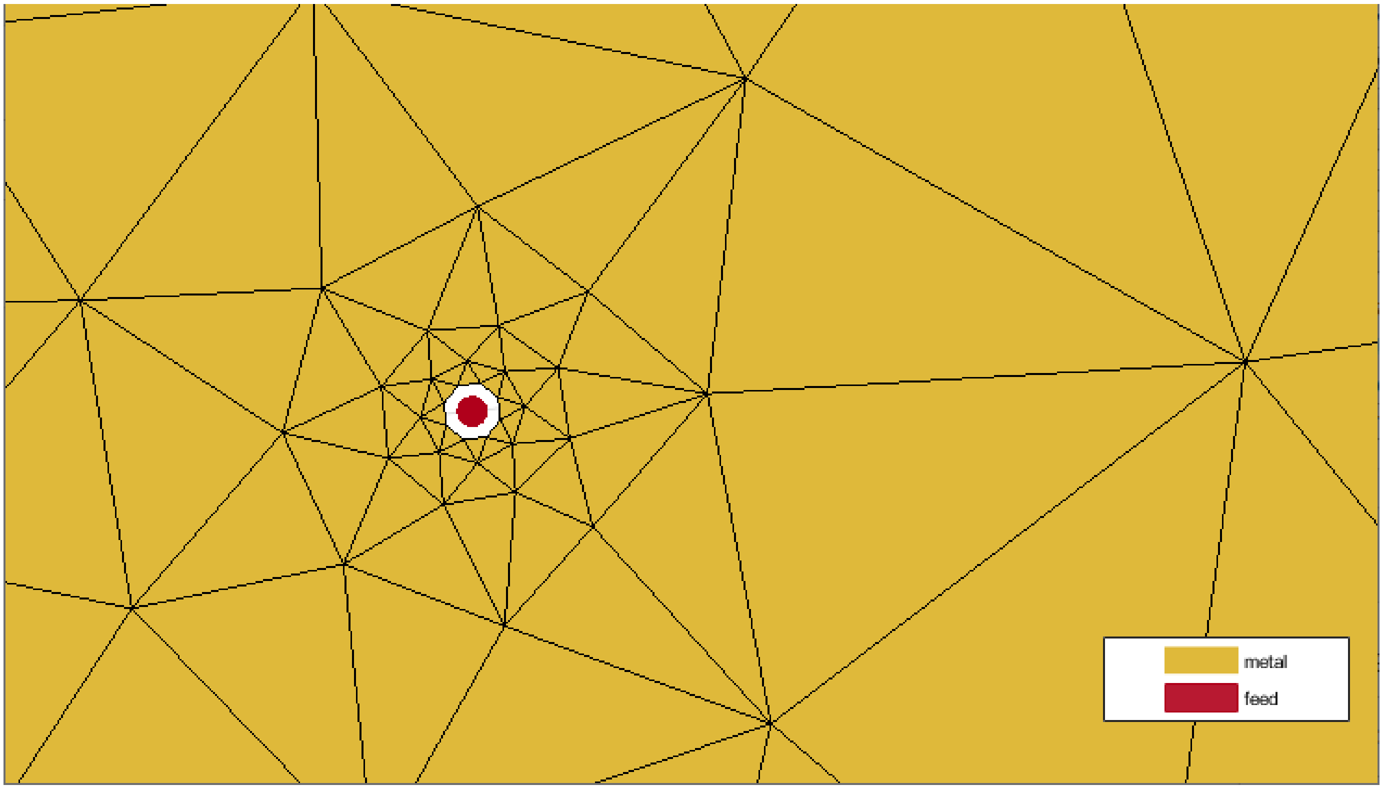
Select the edges of the column that forms a closed polygon. The selected edges must be connected to other edges, else the UI figure window will display an error.



Click **OK** to define the selected edges as feeding edges and the structure with the feed is displayed.



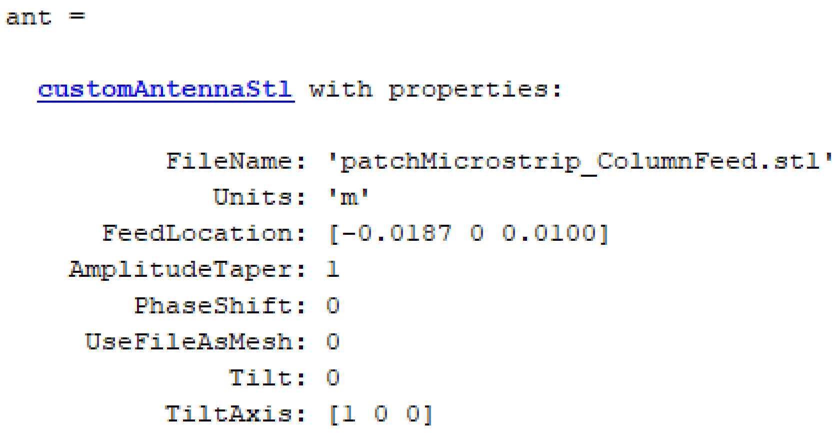
The FeedLocation is displayed.



Verify the location of the antenna feed in the commandline.

```
ant
```

```
ant =
  customAntennaStl with properties:

          FileName: 'patchMicrostrip_ColumnFeed.stl'
             Units: 'm'
      FeedLocation: []
    AmplitudeTaper: 1
        PhaseShift: 0
     UseFileAsMesh: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```



```
ant = 

  customAntennaStl with properties:

          FileName: 'patchMicrostrip_ColumnFeed.stl'
             Units: 'm'
      FeedLocation: [-0.0187 0 0.0100]
    AmplitudeTaper: 1
        PhaseShift: 0
     UseFileAsMesh: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

Version History

Introduced in R2020a

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

See Also

`platform` | `customAntennaGeometry` | `customAntennaMesh`

Topics

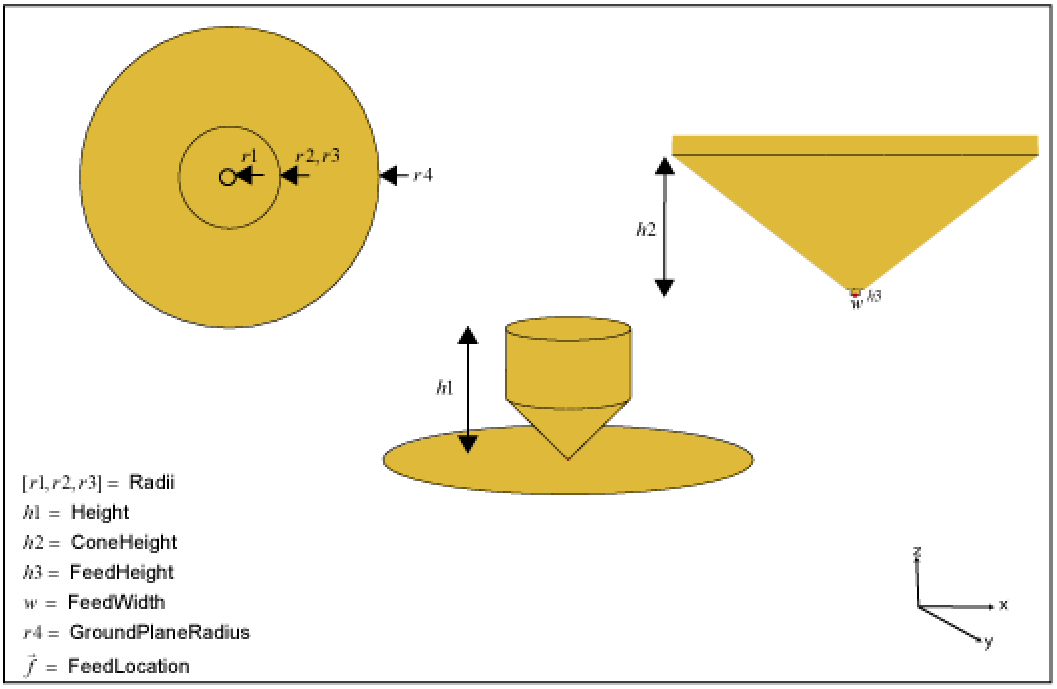
"Rotate Antennas and Arrays"

monocone

Create monocone antenna on circular ground plane

Description

The `monocone` object creates a monocone antenna on a circular ground plane. A classical monocone antenna consists of a cone and a ground plane. To increase the bandwidth of the antenna, you can modify the antenna by merging the cone with a circular cylinder. By default, the `monocone` object creates the modified version.



Create a classical monocone antenna (without the cylinder on top) using one of these methods:

- Set the height of the antenna to equal the sum of the cone height and the feed height.
- Set the cone height to equal half of the difference between the total height and the feed height. Then set the radius at the aperture to twice the radius at the junction.

Creation

Syntax

```
ant = monocone
ant = monocone(Name,Value)
```

Description

`ant = monocone` creates a monocone antenna with the feedpoint at the center of the ground plane. The default dimensions are for a resonant frequency of 3.8 GHz.

`ant = monocone(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = monocone('Height',0.0560)` creates a monocone antenna with a total height of 0.0560 meters.

Properties

`Radii` — Antenna radii

`[5.0000e-04 0.0110 0.0110]` (default) | three-element real vector

Antenna radii, specified as a three-element real vector with each element unit in meters.

- The first element represents the narrow radius of the cone.
- The second element represents the radius at the junction of the cone and the cylinder.
- The third element represents the radius at the top of the cylinder.

Example: `'Radii',[6.3300e-04 0.0546 0.0220]`

Example: `ant.Radii = [6.3300e-04 0.0546 0.0220]`

Data Types: `double`

`Height` — Total height of antenna

`0.0250` (default) | positive scalar

Total height of the antenna from the ground plane to the aperture of the antenna, specified as a positive scalar in meters.

Example: `'Height',0.0560`

Example: `ant.Height = 0.0560`

Data Types: `double`

`ConeHeight` — Vertical height of cone

`0.0115` (default) | positive scalar

Vertical height of the cone from the apex of the cone to the junction of the cone and the cylinder, specified as a positive scalar in meters.

Example: `'ConeHeight',0.02250`

Example: `ant.ConeHeight = 0.02250`

Data Types: `double`

`FeedHeight` — Gap between cone and ground plane

`5.0000e-04` (default) | positive scalar

Gap between the cone and the ground plane, specified as a positive scalar in meters.

Example: `'FeedHeight',0.0034`

Example: `ant.FeedHeight = 0.0034`

Data Types: double

FeedWidth — Width of feed

5.0000e-04 (default) | positive scalar

Width of the feed, specified as a positive scalar in meters.

Example: 'FeedWidth',0.0050

Example: ant.FeedWidth = 0.0050

Data Types: double

GroundPlaneRadius — Radius of ground plane

0.0325 (default) | positive scalar

Radius of the ground plane, specified as a positive scalar in meters.

Example: 'GroundPlaneRadius',0.0050

Example: ant.GroundPlaneRadius = 0.050

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Load — Lumped elements

[1x1 lumpedElement] (default) | lumpedElement object

Lumped elements added to the antenna feed, specified as a lumpedElement object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see lumpedElement.

Example: 'Load',lumpedElement, where lumpedElement is load added to the antenna feed.

Example: ant.Load = lumpedElement('Impedance',75)

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

coneangle2size	Calculates equivalent cone height, broad radius, and narrow radius for cone
show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

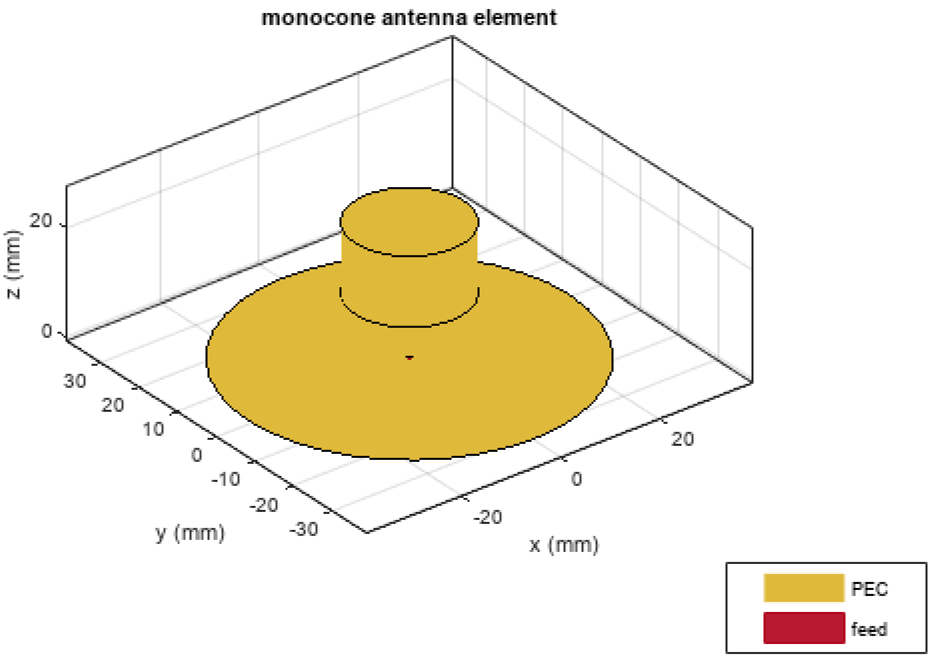
Default Monocone Antenna

Create and view a default monocone antenna.

```
ant = monocone
```

```
ant = 
  monocone with properties:

               Radii: [5.0000e-04 0.0110 0.0110]
    GroundPlaneRadius: 0.0325
           ConeHeight: 0.0115
               Height: 0.0250
           FeedHeight: 5.0000e-04
            FeedWidth: 5.0000e-04
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

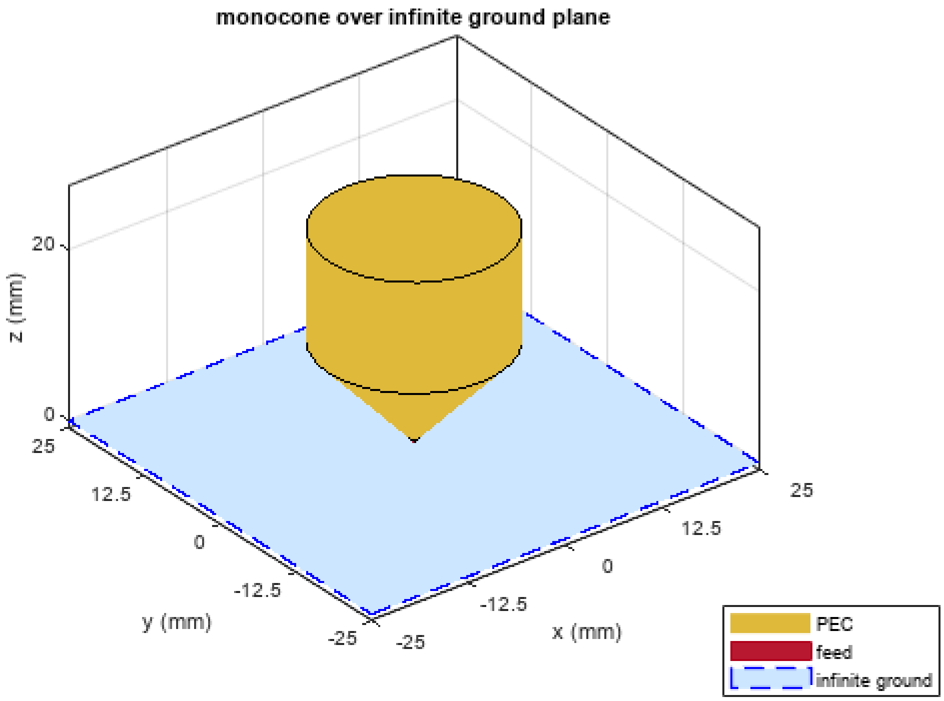
```
show(ant)
```



Monocone Antenna with Infinite Ground Plane

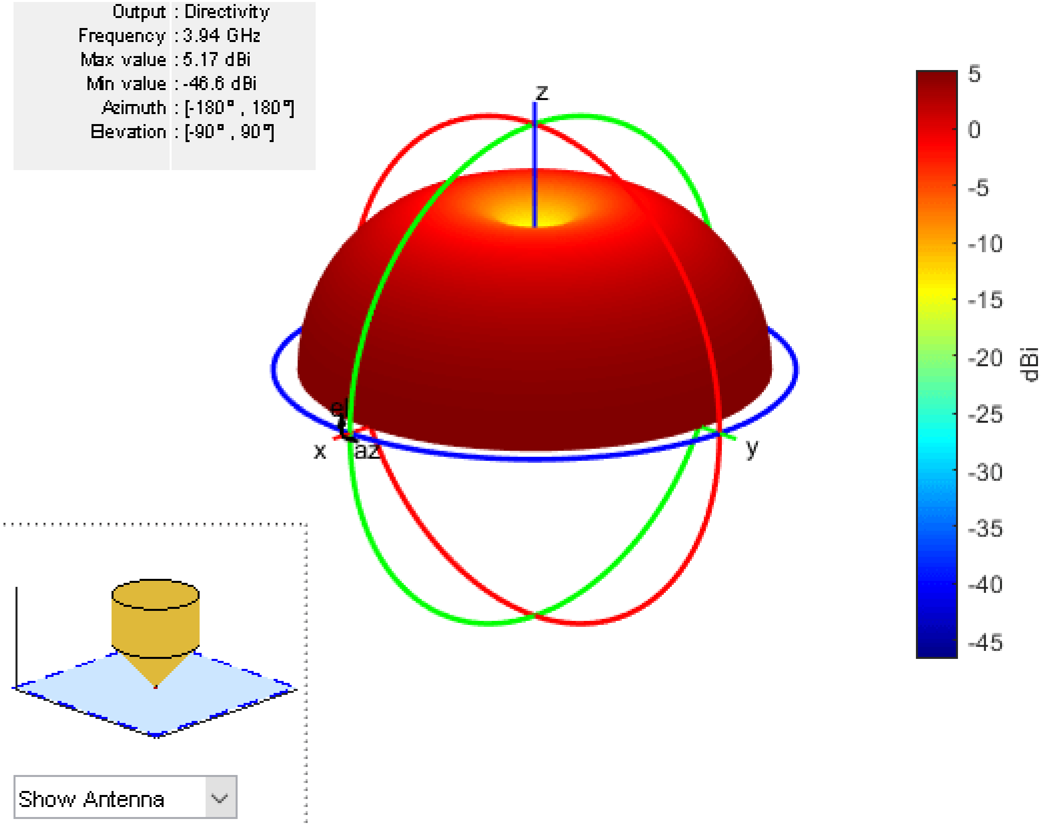
Create a monocone antenna with an infinite ground plane.

```
ant = monocone;
ant.GroundPlaneRadius = inf;
show(ant)
```



Plot the radiation pattern of the monocone antenna for the given frequency.

```
pattern(ant,3.94e9)
```

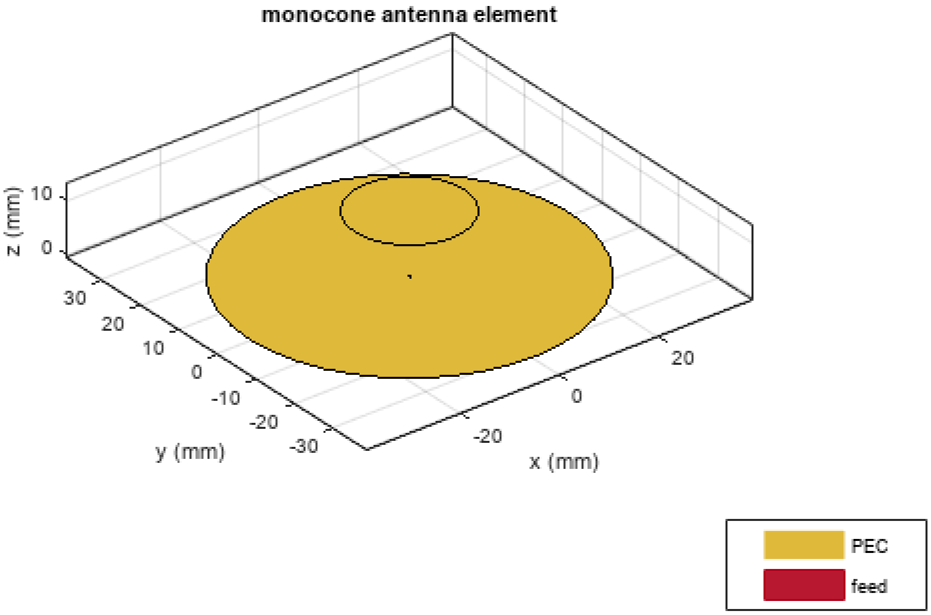


Output : Directivity
Frequency : 3.94 GHz
Max value : 5.17 dBi
Min value : -46.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Monocone Antenna Without A Cylinder

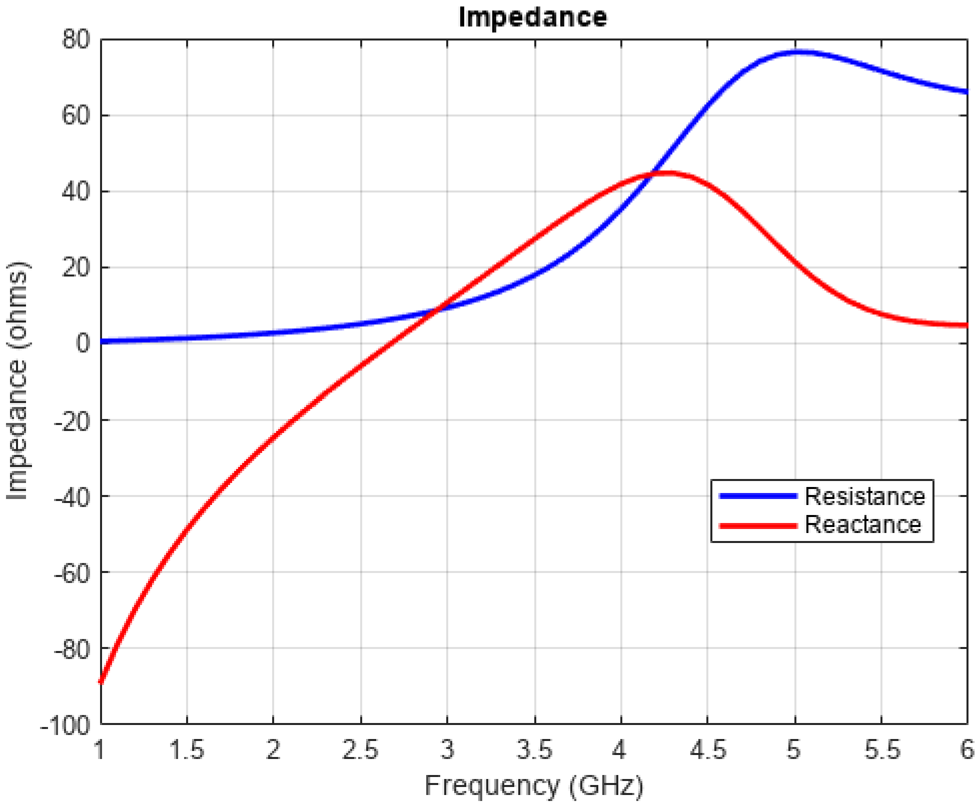
Create a classical monocone antenna by setting the total height of the antenna to equal the sum of cone height and feed height.

```
ant = monocone;
ant.Height = ant.ConeHeight+ant.FeedHeight;
show(ant)
```



Calculate antenna impedance over the given frequency span.

```
impedance(ant,(1e9:0.1e9:6e9))
```



Version History

Introduced in R2020a

References

[1] McDonald, James L., and Dejan S. Filipovic. “On the Bandwidth of Monocone Antennas.” *IEEE Transactions on Antennas and Propagation* 56, no. 4 (April 2008): 1196–1201. https://doi.org/10.1109/TAP.2008.919226.

See Also

`cavityCircular` | `bicone`

Topics

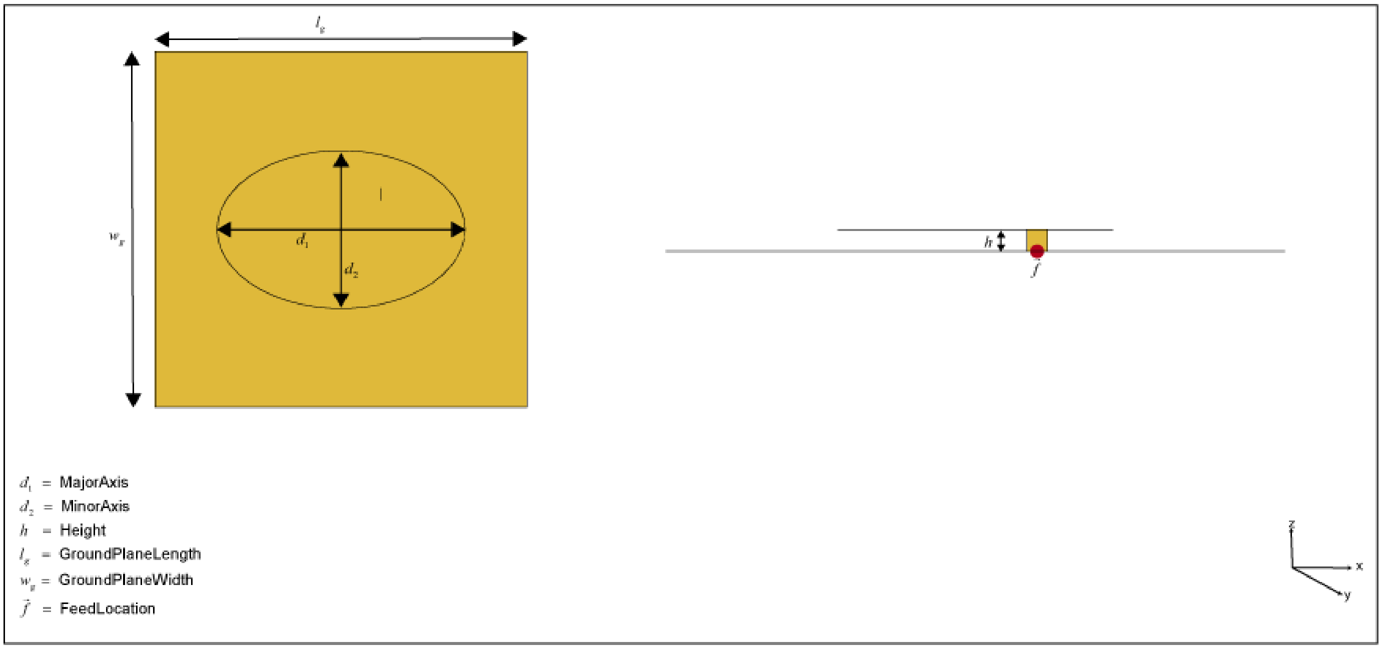
“Rotate Antennas and Arrays”

patchMicrostripElliptical

Create elliptical microstrip patch antenna

Description

The `patchMicrostripElliptical` object creates a probe-fed elliptical microstrip patch antenna. The default patch is centered at the origin. The ellipse is chosen for an operating frequency of around 5.45 GHz. Elliptical microstrip patch antennas are used in high-performance applications such as spacecraft, aircraft, missiles, and satellites. Elliptical microstrip patch antennas with optimum dimensions act as circularly polarized wave radiators.



Creation

Syntax

```
ant = patchMicrostripElliptical
ant = patchMicrostripElliptical(Name,Value)
```

Description

`ant = patchMicrostripElliptical` creates a probe-fed elliptical microstrip patch antenna operating at 5.45 GHz.

`ant = patchMicrostripElliptical(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = patchMicrostripElliptical('MajorAxis',0.0878)` creates an elliptical microstrip patch antenna with a major axis of 0.0878 meters. Enclose each property name in quotes.

Properties

`MajorAxis` — Longest diameter of ellipse

`0.0300` (default) | scalar

Longest diameter of the ellipse along the *x*-axis, specified as a scalar in meters.

Example: `'MajorAxis',0.0989`

Example: `ant.MajorAxis = 0.0989`

Data Types: `double`

`MinorAxis` — Shortest diameter of ellipse

`0.0200` (default) | scalar

Shortest diameter of the ellipse along the *y*-axis, specified as a scalar in meters.

Example: `'MinorAxis',0.0898`

Example: `ant.MinorAxis = 0.0898`

Data Types: `double`

`Height` — Height of patch

`0.0016` (default) | scalar

Height of patch above the ground plane along the *z*-axis, specified as a scalar in meters.

Example: `'Height',0.001`

Example: `ant.Height = 0.001`

Data Types: `double`

`Substrate` — Type of dielectric material

'Air' (default) | `dielectric` function

Type of dielectric material used as a substrate, specified as a dielectric material object. You can choose any material from the `DielectricCatalog` or use your own dielectric material. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be lesser than the ground plane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric('FR4'); ant.Substrate = d`

`GroundPlaneLength` — Ground plane length

0.0450 (default) | scalar

Ground plane length along the *x*-axis, specified as a scalar in meters. Setting `'GroundPlaneLength'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',120e-3`

Example: `ant.GroundPlaneLength = 120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width

0.0450 (default) | scalar

Ground plane width along the *y*-axis, specified as a scalar in meters. Setting `'GroundPlaneWidth'` to `Inf`, uses the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',120e-3`

Example: `ant.GroundPlaneWidth = 120e-3`

Data Types: `double`

`PatchCenterOffset` — Signed distance of patch from origin

`[0 0]` (default) | two-element real vector

Signed distance of the patch from the origin, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the patch relative to the ground plane. Distances are measured along the length and width of the ground plane.

Example: `'PatchCenterOffset',[0.01 0.01]`

Example: `ant.PatchCenterOffset = [0.01 0.01]`

Data Types: `double`

`FeedOffset` — Signed distance of feed from origin

`[0.0047 0.0045]` (default) | two-element real vector

Signed distance of the feed from the origin, specified as a two-element real vector with each element unit in meters. Use this property to adjust the location of the feed relative to the ground plane and patch.

Example: `'FeedOffset',[0.01 0.01]`

Example: `ant.FeedOffset = [0.01 0.01]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays

impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

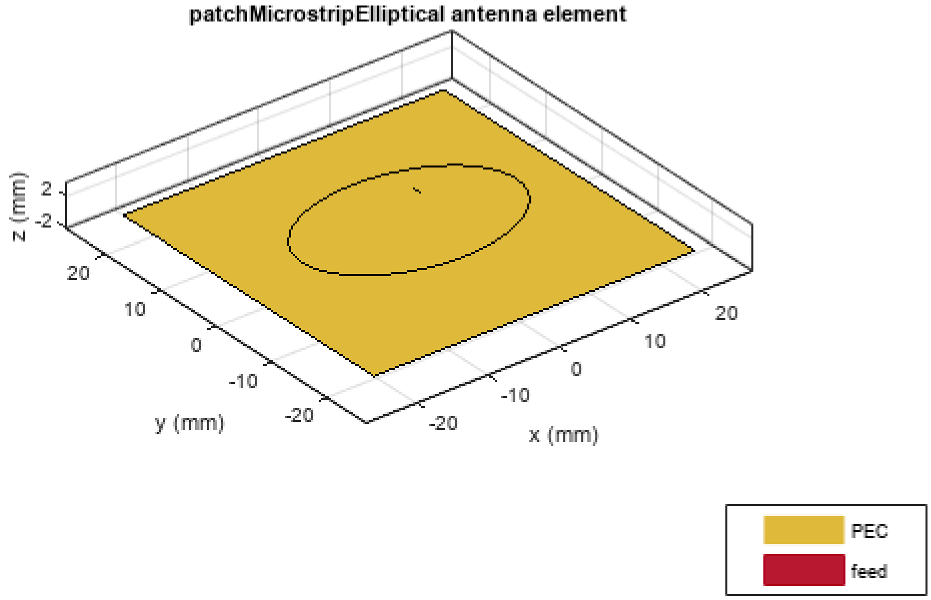
Default Elliptical Microstrip Patch Antenna

Create and view a default elliptical microstrip patch antenna.

```
ant = patchMicrostripElliptical
```

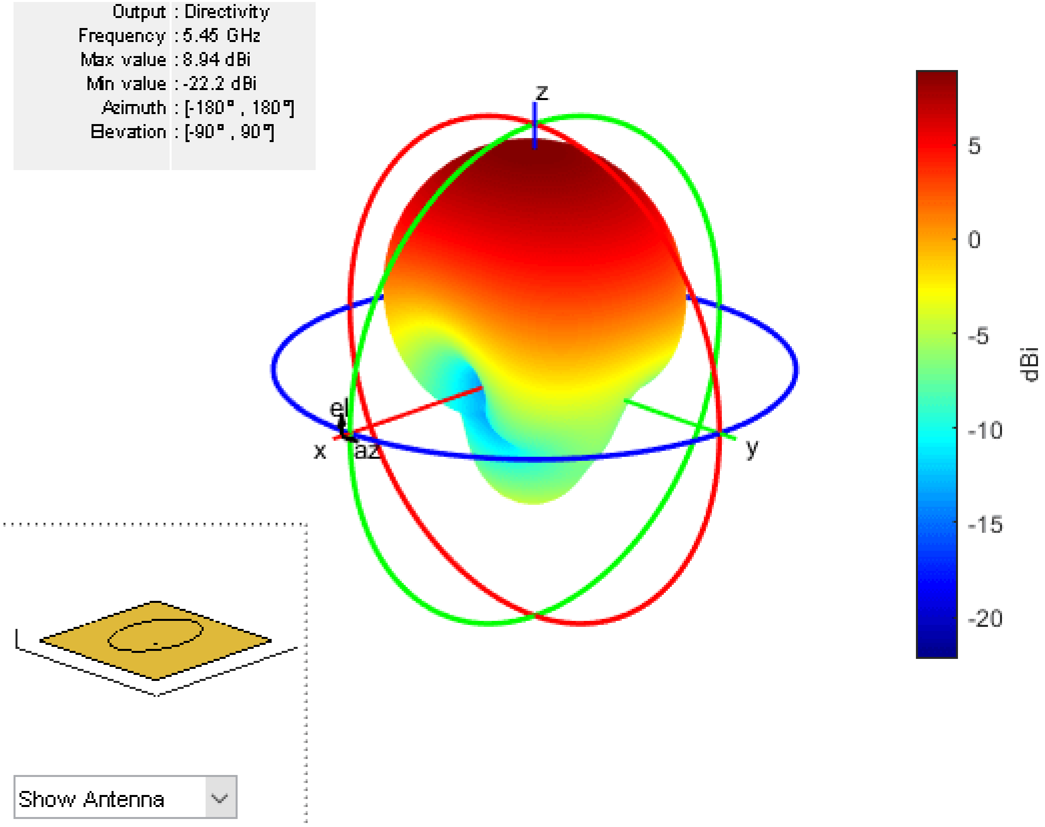
```
ant = 
  patchMicrostripElliptical with properties:

            MajorAxis: 0.0300
            MinorAxis: 0.0200
               Height: 0.0016
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.0450
     GroundPlaneWidth: 0.0450
    PatchCenterOffset: [0 0]
           FeedOffset: [0.0047 0.0045]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show (ant)
```



Visualize the radiation pattern of the antenna at 5.45 GHz.

```
pattern(ant,5.45e9)
```



```
   Output : Directivity
Frequency : 5.45 GHz
Max value : 8.94 dBi
Min value : -22.2 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Version History

Introduced in R2020a

See Also

`patchMicrostrip` | `patchMicrostripCircular`

Topics

"ISM Band Patch Microstrip Antennas and Mutually Coupled Patches"

"Rotate Antennas and Arrays"

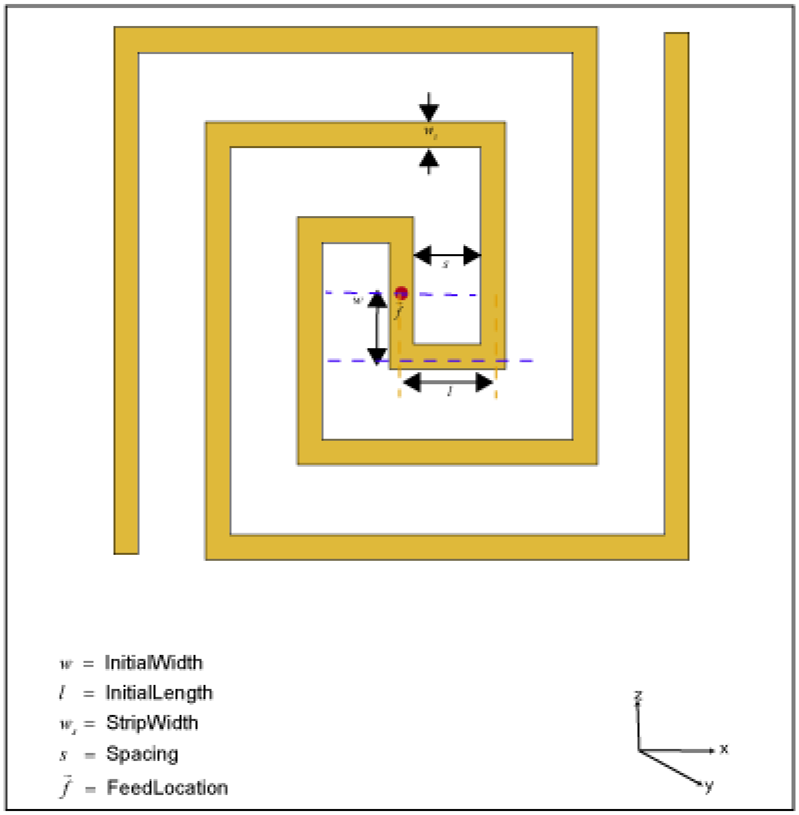
spiralRectangular

Create rectangular spiral antenna on X-Y plane

Description

The `spiralRectangular` object creates a single or two-arm rectangular spiral antenna. The default rectangular spiral has two arms, is center-fed and is on the X-Y plane. The default resonating frequency is 7.65 GHz.

A spiral rectangular antenna is made up of filaments. The distance between the two violet dashed lines in the diagram represents the first filament or the initial width. The distance between the two orange dashed lines in the diagram represents the second filament or the initial length.



Creation

Syntax

```
ant = spiralRectangular
ant = spiralRectangular(Name,Value)
```

Description

`ant = spiralRectangular` creates a default rectangular spiral antenna object operating at 7.65 GHz.

`ant = spiralRectangular(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = spiralRectangular('NumArms',1)` creates a rectangular spiral antenna object with one arm. Enclose each property name in quotes.

Properties

`NumArms` — Number of arms of spiral

`2` (default) | `1`

Number of arms of the spiral, specified as `1` or `2`.

Example: `'NumArms',1`

Example: `ant.NumArms = 1`

Data Types: `double`

`NumTurns` — Number of turns in spiral

`1.53` (default) | scalar

Number of turns in the spiral, specified as a scalar in meters. One turn length is taken as the length of a complete 360-degree revolution. To calculate the length of 1.25 turns, the first spiral is created up to one turn. Then the length of the second turn is scaled to the given fraction and added to the first turn length.

Example: `'NumTurns',2.0`

Example: `ant.NumTurns = 2.0`

Data Types: `double`

`InitialWidth` — Length of first filament along Y-axis

`0.0010` (default) | scalar

Length of the first filament along the Y-axis from the origin to the midline of the strip width of the second filament, specified as a scalar in meters. `InitialWidth` is the width between the dashed violet color lines in the antenna image.

Example: `'InitialWidth',0.0050`

Example: `ant.InitialWidth = 0.0050`

Data Types: `double`

`InitialLength` — Length of second filament along X-axis

`0.0015` (default) | scalar

Length of the second filament along the X-axis from the mid line of the first filament to half of the strip width of the third filament, specified as a scalar in meters. `InitialLength` is the width between the dashed orange color lines in the antenna image.

Example: `'InitialLength',0.0055`

Example: `ant.InitialLength = 0.0055`

Data Types: `double`

`StripWidth` — Width of strip
`4.0500e-04` (default) | scalar

Width of the strip, specified as a scalar in meters.

Example: `'StripWidth',5.0050e-04`

Example: `ant.StripWidth = 5.0050e-04`

Data Types: `double`

`Spacing` — Spacing between turns
`0.0011` (default) | scalar

Spacing between turns of the spiral, specified as a scalar in meters.

Example: `'Spacing',0.0015`

Example: `ant.Spacing = 0.0015`

Data Types: `double`

`WindingDirection` — Direction of spiral turns (windings)
`'CCW'` (default) | `'CW'`

Direction of the spiral turns (windings), specified as `'CW'` or `'CCW'`.

Example: `'WindingDirection','CW'`

Example: `ant.WindingDirection = CW`

Data Types: `char` | `string`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer

pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna
rectspirallength2turns	Calculate number of turns for specified arm length in rectangular spiral antenna

Examples

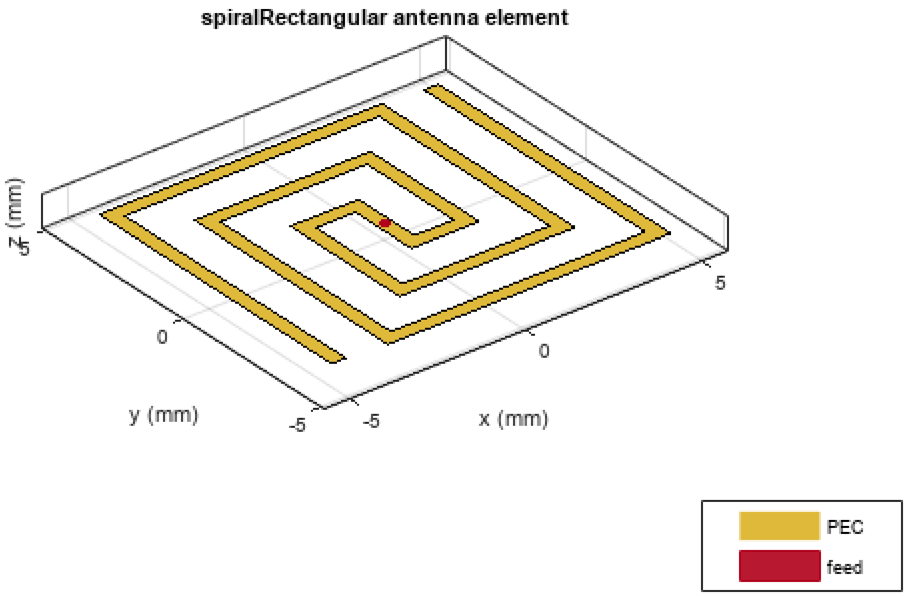
Default Rectangular Spiral Antenna

Create and view a default rectangular spiral antenna.

```
ant = spiralRectangular
```

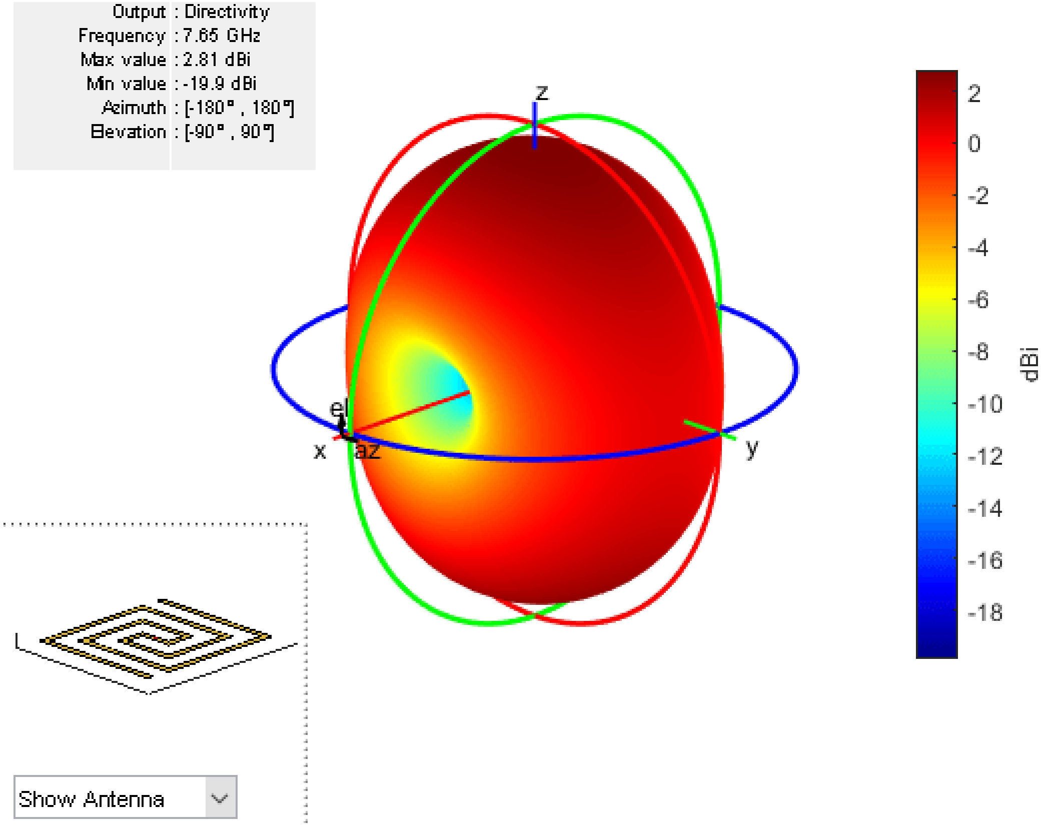
```
ant = 
  spiralRectangular with properties:

             NumArms: 2
            NumTurns: 1.5300
        InitialWidth: 0.0010
       InitialLength: 0.0015
          StripWidth: 4.0500e-04
             Spacing: 0.0011
    WindingDirection: 'CCW'
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at the default frequency.

```
pattern(ant,7.65e9)
```



Output : Directivity
Frequency : 7.65 GHz
Max value : 2.81 dBi
Min value : -19.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

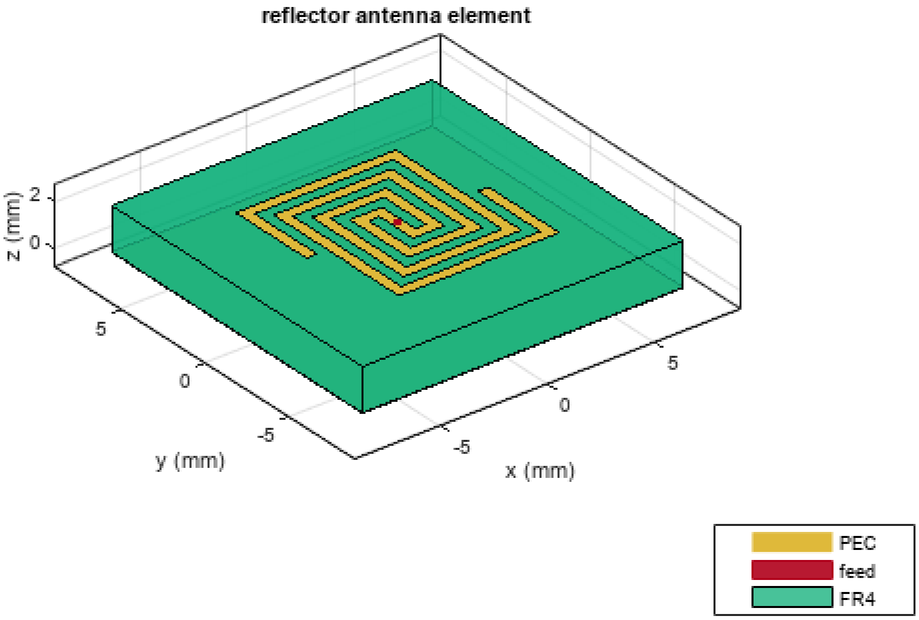
Radiation Pattern of Reflector Backed Rectangular Spiral

Create a rectangular spiral antenna object with two arms and two turns.

```
ant_d = spiralRectangular('NumArms',2,'NumTurns',2,'InitialLength',1e-3,...
            'InitialWidth',1e-3,'Spacing',0.5e-3,'StripWidth',0.5e-3);
```

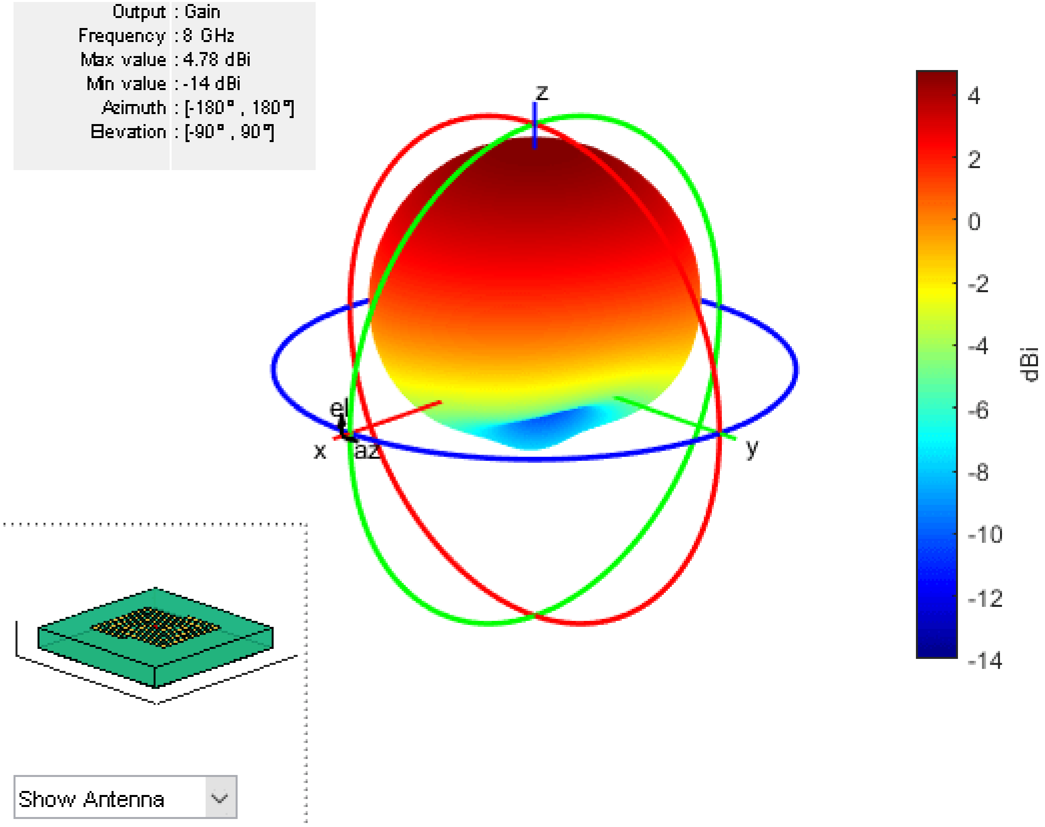
Back the spiral using a reflector antenna object.

```
    r = reflector('Exciter',ant_d,'GroundPlaneLength',15e-3,'GroundPlaneWidth',...
        15e-3,'Spacing',2e-3,'Substrate',dielectric('FR4'));
    figure;
    show(r);
```



Plot the radiation pattern of the antenna at the specified frequency.

```
figure;
pattern(r,8e9);
```

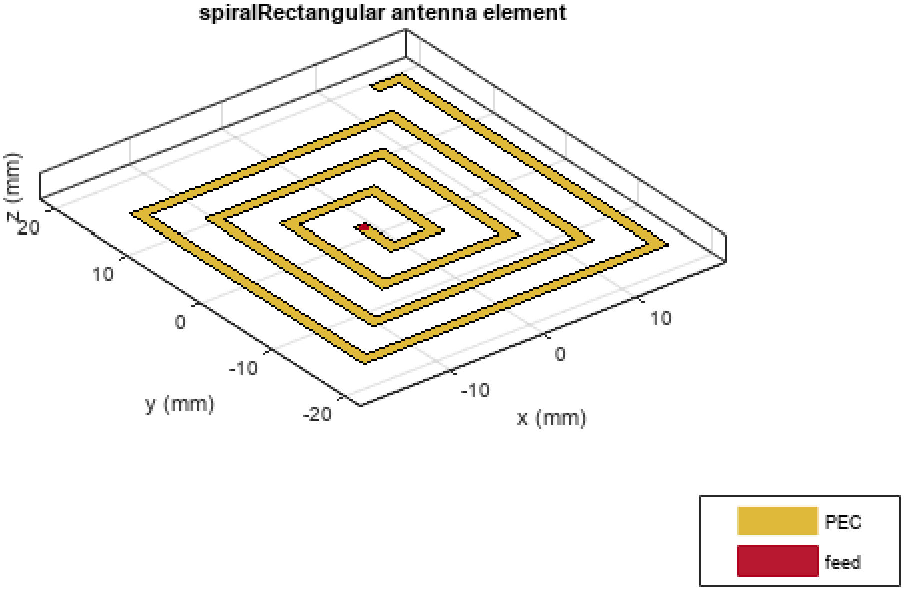


```
   Output : Gain
Frequency : 8 GHz
Max value : 4.78 dBi
Min value : -14 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Rectangular Spiral Antenna with Specified Arm Length

Create a single arm rectangular spiral antenna with a total arm length of 291 mm.

```
ant = spiralRectangular('NumArms',1,'NumTurns',3,'InitialLength',4.5e-3,...
              'InitialWidth',4.5e-3,'Spacing',3.3e-3,'StripWidth',1.2e-3);
nT = rectspirallength2turns(ant,291e-3);
ant.NumTurns = nT;
figure;
show(ant);
```



Version History

Introduced in R2020a

References

[1] Nakano, H., H. Yasui, and J. Yamauchi. “Numerical Analysis of Two-Arm Spiral Antennas Printed on a Finite-Size Dielectric Substrate.” *IEEE Transactions on Antennas and Propagation* 50, no. 3 (March 2002): 362–70. https://doi.org/10.1109/8.999628.

[2] Nakano, H., J. Eto, Y. Okabe, and J. Yamauchi. “Tilted- and Axial-Beam Formation by a Single-Arm Rectangular Spiral Antenna with Compact Dielectric Substrate and Conducting Plane.” *IEEE Transactions on Antennas and Propagation 50*, no. 1 (January 2002): 17–24. https://doi.org/10.1109/8.992557.

See Also

`spiralArchimedean` | `spiralEquiangular` | `rectspirallength2turns`

Topics

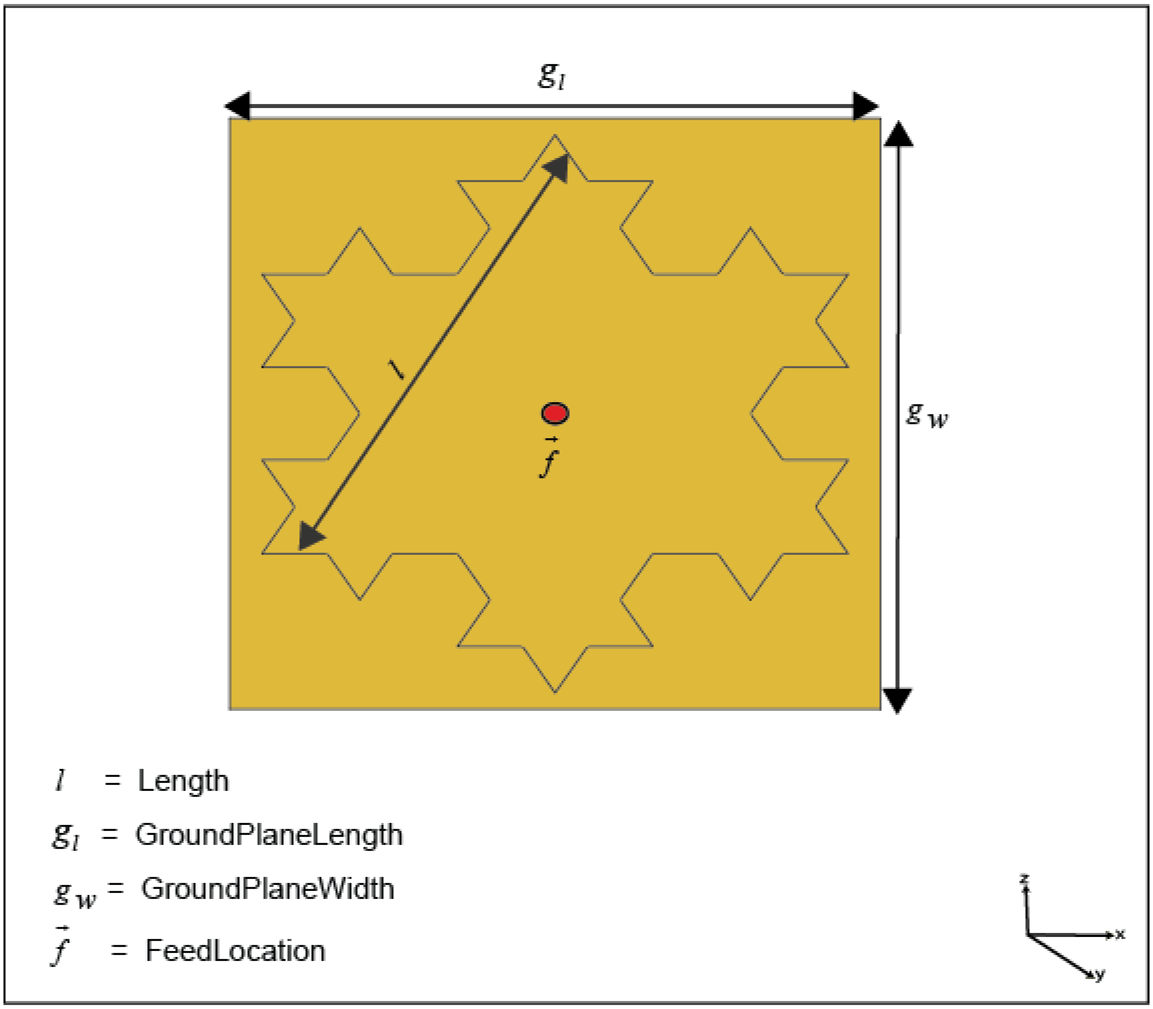
“Rotate Antennas and Arrays”

fractalSnowflake

Create fractal Koch snowflake antenna

Description

The `fractalSnowflake` object creates a Koch snowflake fractal antenna. These fractal antennas are used in mobile phone, Wi-Fi, and radar applications.



A fractal antenna uses a fractal, a self-similar design that is repeated in different dimensions so as to maximize effective the length or increase the perimeter of the material that transmits or receives electromagnetic radiation. This makes the fractal antennas compact and therefore suitable for use in small and complex circuits. Fractal antennas also have higher input impedance or resistance due to their length or increased perimeter.

All fractal antennas are printed structures that are etched on a dielectric substrate.

Creation

Syntax

```
ant = fractalSnowflake
ant = fractalSnowflake(Name,Value)
```

Description

`ant = fractalSnowflake` creates a Koch’s snowflake fractal antenna. The default fractal is centered at the origin, and the number of iterations is set to 2. The length of the fractal is for an operating frequency of 4.15 GHz.

`ant = fractalSnowflake(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = fractalSnowflake('Numiterations',4)` creates a Koch's snowflake with four iterations.

Properties

`NumIterations` — Number of iterations performed on fractal antenna
2 (default) | scalar integer

Number of iterations performed on the fractal antenna, specified as a scalar integer.

Example: `'NumIterations',4`

Example: `ant.NumIterations = 4`

Data Types: `double`

`Length` — Length of the sides of the equilateral triangle
`0.0900` (default) | positive scalar integer

Length of the side of the equilateral triangle in fractal snowflake, specified as a positive scalar integer in meters.

Example: `'Length',0.5000`

Example: `ant.Length = 0.5000`

Data Types: `double`

`Height` — Height of fractal
`0.0015` (default) | positive scalar integer

Height of the fractal from the ground plane along *z*-axis, specified as a positive scalar integer in meters.

Example: `'Height',0.0050`

Example: `ant.Height = 0.0050`

Data Types: `double`

`Substrate` — Type of dielectric material
air (default) | `dielectric` object

Type of dielectric material used as a substrate, specified as an `dielectric` object. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Example: `d = dielectric('FR4'); ant = fractalSnowflake('Substrate',d)`

Example: `ant= fractalSnowflake('Substrate',dielectric('Name','RO4003C','EpsilonR',3.38,'LossTangent',0.0027,'Thickness',0.508e-3))`

Data Types: `string` | `char`

`GroundPlaneLength` — Length of ground plane
`0.1000` (default) | positive scalar integer

Length of the ground plane, specified as a positive scalar integer in meters.

Example: `'GroundPlaneLength',0.0550`

Example: `ant.GroundPlaneLength = 0.0550`

Data Types: `double`

`GroundPlaneWidth` — Width of ground plane
`0.1100` (default) | positive scalar integer

Width of the ground plane, specified as a positive scalar integer in meters.

Example: `'GroundPlaneWidth',0.0550`

Example: `ant.GroundPlaneWidth = 0.0550`

Data Types: `double`

`FractalCenterOffset` — Signed distance of fractal snowflake center from origin
`[0 0]` (default) | two-element real-valued vector

Signed distance of fractal snowflake center from origin, specified as a two-element real-valued vector with each element unit in meters. The distance is measured along the length and width of the ground plane.

Example: `'FractalCenterOffset',[0 0.080]`

Example: `ant.FractalCenterOffset = [0 0.080]`

Data Types: `double`

`FeedOffset` — Signed distance of feed from origin
`[0 0]` (default) | two-element real-valued vector

Signed distance of the feed from the origin, specified as a two-element real-valued vector with each element unit in meters.

Example: `'FeedOffset',[0 0.080]`

Example: `ant.FeedOffset = [0 0.080]`

Data Types: `double`

`FeedDiameter` — Diameter of feed
`0.0020 `] (default) | positive scalar integer

Diameter of the feed, measured in meters.

Example: 'FeedDiameter',0.001

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see "Meshing".

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: Tilt=90

Example: Tilt=[90 90],TiltAxis=[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: TiltAxis=[0 1 0]

Example: TiltAxis=[0 0 0;0 1 0]

Example: TiltAxis = 'Z'

Note The wireStack antenna object only accepts the dot method to change its properties.

Data Types: double

Load — Lumped elements

[1x1] lumpedElement] (default) | lumped element

Lumped elements added to the antenna feed, specified as a `lumpedelement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. `lumpedelement` is the object for the load created using `lumpedElement`. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`.

Example: `ant.Load = lumpedElement('Impedance',75,'Frequency',2.9e6,'location', [20e-3 1e-3 1.5e-3])`

Object Functions

show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
current	Current distribution on antenna or array surface
charge	Charge distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
design	Design prototype antenna or arrays for resonance around specified frequency

Examples

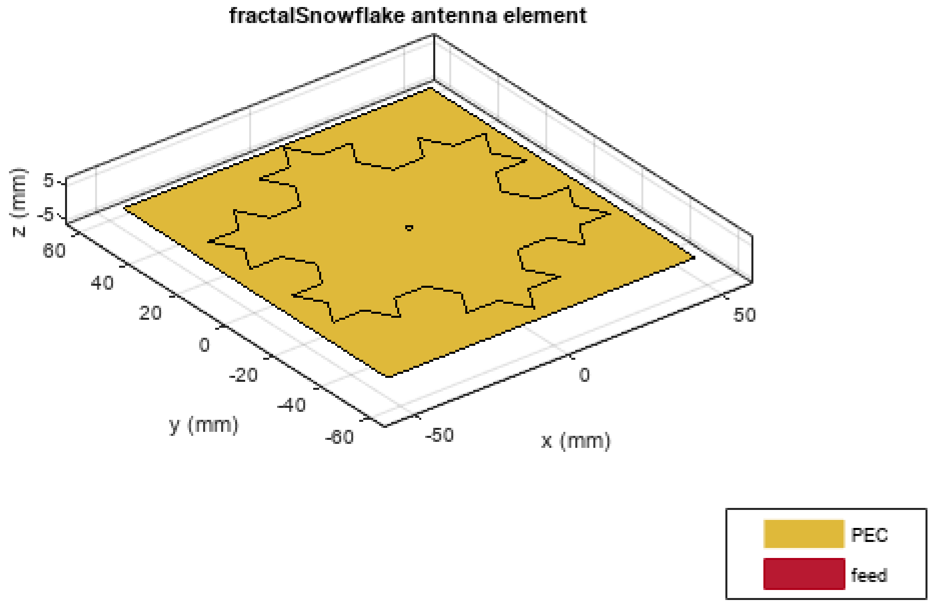
Fractal Snowflake with Default Properties

Create and View fractal Koch snowflake antenna object with default properties.

```
ant = fractalSnowflake
```

```
ant = 
  fractalSnowflake with properties:

               Length: 0.0900
        NumIterations: 2
               Height: 0.0015
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1000
     GroundPlaneWidth: 0.1100
  FractalCenterOffset: [0 0]
           FeedOffset: [0 0]
         FeedDiameter: 0.0020
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

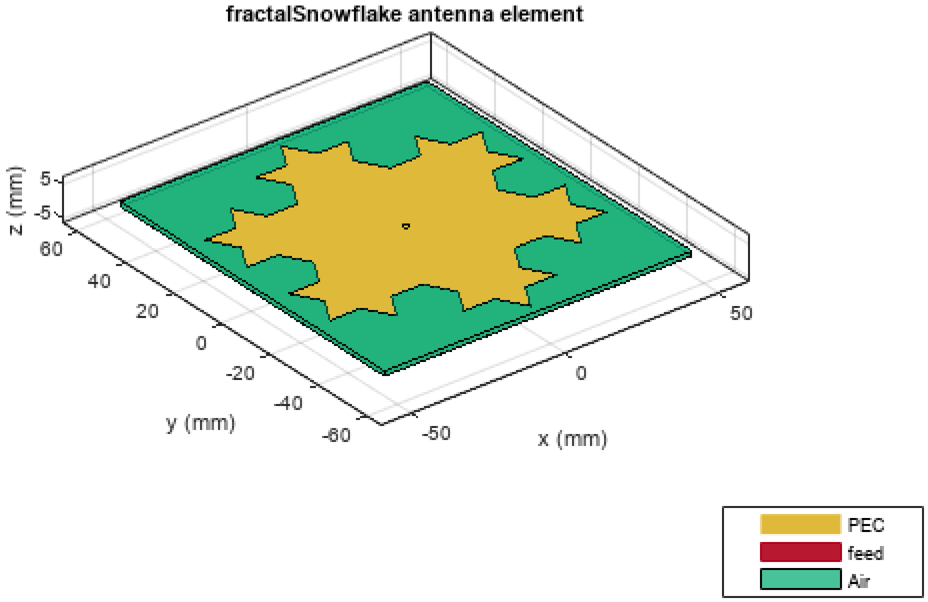
```
show(ant)
```



Fractal Snowflake Antenna with Specified Parameters

Create and view fractal Koch snowflake on a substrate with a dielectric constant of 4 and thickness of 1.5e-3.

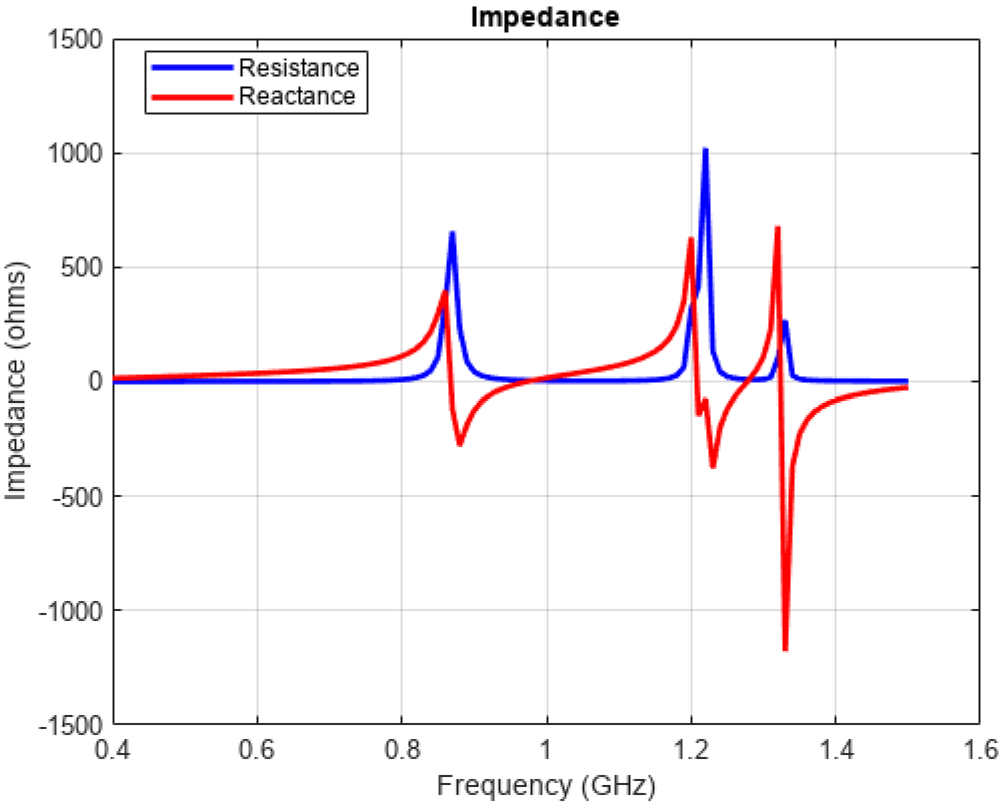
```
 ant = fractalSnowflake('Substrate', dielectric('EpsilonR',4,...
            'Thickness',1.5e-3));
    show(ant);
```



Impedance Plot of Fractal Koch Snowflake Antenna

Create a fractal Koch snowflake antenna and plot its impedance over a frequency range of 400-1500 MHz.

```
ant = fractalSnowflake('Length',180e-3,'GroundPlaneLength',280e-3,...
          'GroundPlaneWidth',240e-3,'Height',5e-3,'FeedOffset',...
          [75e-3,-45e-3]);
figure
impedance(ant,(400:10:1500)*1e6)
```



Version History

Introduced in R2020a

See Also

`fractalCarpet` | `fractalKoch` | `fractalGasket` | `fractalIsland`

Topics

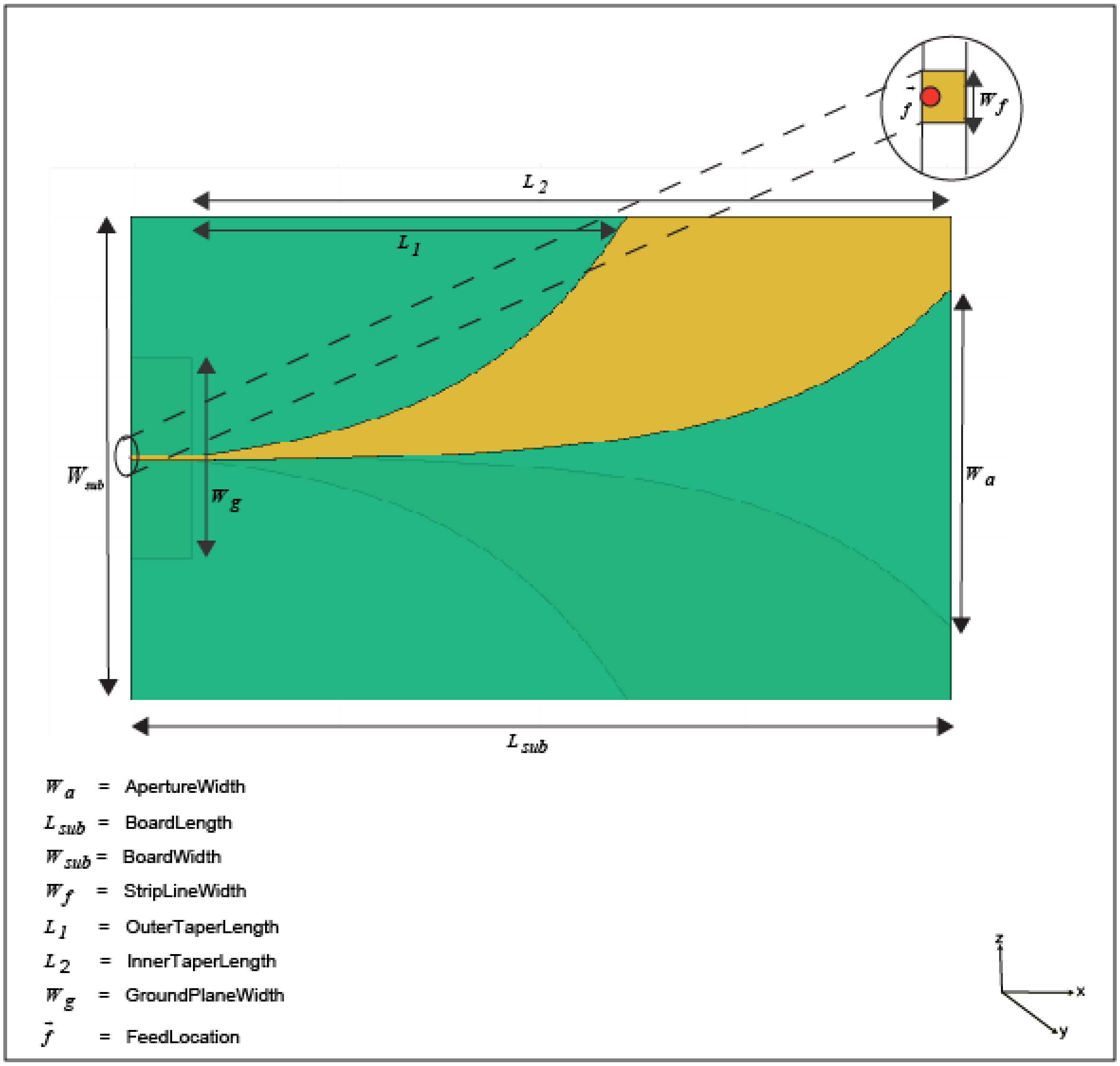
"Rotate Antennas and Arrays"

vivaldiAntipodal

Create an antipodal Vivaldi element

Description

The `vivaldiAntipodal` object creates an antipodal Vivaldi element. Antipodal Vivaldi come under the group of end-fire tapered slot antennas, and such antennas are expected to provide medium gain with less side lobes and wide bandwidth. These antennas are low cost, geometrically simple in shape, and mostly used in wireless communications and radar applications.



Creation

Syntax

```
ant = vivaldiAntipodal
ant = vivaldiAntipodal(Name,Value)
```

Description

`ant = vivaldiAntipodal` creates an antipodal Vivaldi object. By default, the antenna is centered at the origin and the dimension are chosen for an operating frequency of 3.22 GHz.

`ant = vivaldiAntipodal(Name,Value)` sets properties using one or more name-value pairs. For example, `aviv = vivaldiAntipodal('BoardLength',0.2)` creates a antipodal Vivaldi with a board length of 0.2 m.

Note Properties you do not specify retain their default values.

Properties

`BoardLength` — Printed circuit board (PCB) length along *x*-axis
0.202 (default) | scalar

Length of the PCB, specified as a scalar in meter.

Example: `'BoardLength',2e-3`

`BoardWidth` — Printed circuit board (PCB) length along *y*-axis
0.12 (default) | scalar

Width of the PCB, specified as a scalar in meter.

Example: `'BoardWidth',2e-3`

`Height` — Printed circuit board (PCB) length along *z*-axis
0.000508 (default) | scalar

Height of the PCB, specified as a scalar in meter.

Example: `'Height',1e-6`

`OpeningRate` — Taper opening rate
25 (default) | scalar

Opening rate of taper, specified as a scalar. This property determines the rate at which the notch transitions from the feedpoint to the aperture. Minimum value of `OpeningRate` is `1` and maximum value of is `80`.

Example: `'OpeningRate',1.2`

Data Types: `double`

`InnerTaperLength` — Inner taper length
0.187 (default) | scalar

Taper length at antenna's inner edge, specified as a scalar in meters.

Example: `'InnerTaperLength',2e-3`

`OuterTaperLength` — Outer taper length
0.08 (default) | scalar

Taper length at antenna's outer edge, specified as a scalar in meter.

Example: `'OuterTaperLength',2e-3`

`ApertureWidth` — Aperture width
0.084 (default) | scalar

Width of the aperture, specified as a scalar in meters.

Example: `'ApertureWidth',3e-3`

`StripLineWidth` — Strip width
0.0011 (default) | scalar

Width of the strip used at feedpoint, specified as a scalar in meters.

Example: `'StripLineWidth',0.3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width
0.05 (default) | scalar

Ground plane width, specified a scalar in meters. By default, ground plane width is measured along the *y*-axis.

Example: `'GroundPlaneWidth',4`

Data Types: `double`

`Substrate` — Type of dielectric material
[1x1 `dielectric`] (default) | `dielectric` object

Type of dielectric material used as a substrate, specified as an dielectric object. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”. By default, the `dielectric` is Rogers RO4003C with `EpsilonR` of 3.38, `LossTangent` of 0.0027, and `Thickness` of 0.000508

Example: `ant= vivaldiAntipodal('Substrate',dielectric('Name','RO4003C','EpsilonR',3.38,'LossTangent',0.0027,'Thickness',0.6e-3))`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | lumped element object

Lumped elements added to the antenna feed, specified as a lumped element object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object for the load created using `lumpedElement`.

Example: `avi.Load = lumpedElement('Impedance',75)`

Object Methods

show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
design	Design prototype antenna or arrays for resonance around specified frequency

Examples

Create and View Antipodal Vivaldi Antenna

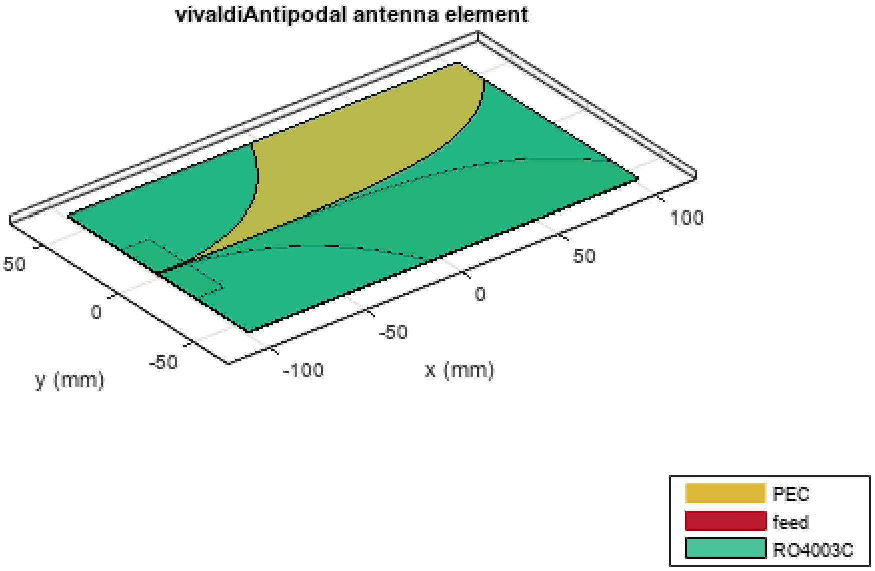
Create an antipodal Vivaldi antenna object with the specified properties.

```
avi = vivaldiAntipodal("OpeningRate",30,'Substrate',dielectric('Name','RO4003C','EpsilonR',3.38,
            'Thickness',0.508e-3))
```

```
avi = 
  vivaldiAntipodal with properties:

         BoardLength: 0.2020
          BoardWidth: 0.1200
              Height: 5.0800e-04
         OpeningRate: 30
      StripLineWidth: 0.0011
    OuterTaperLength: 0.0800
    InnerTaperLength: 0.1870
       ApertureWidth: 0.0840
    GroundPlaneWidth: 0.0500
           Substrate: [1x1 dielectric]
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1x1 lumpedElement]
```

View the antenna.

```
show(avi)
```



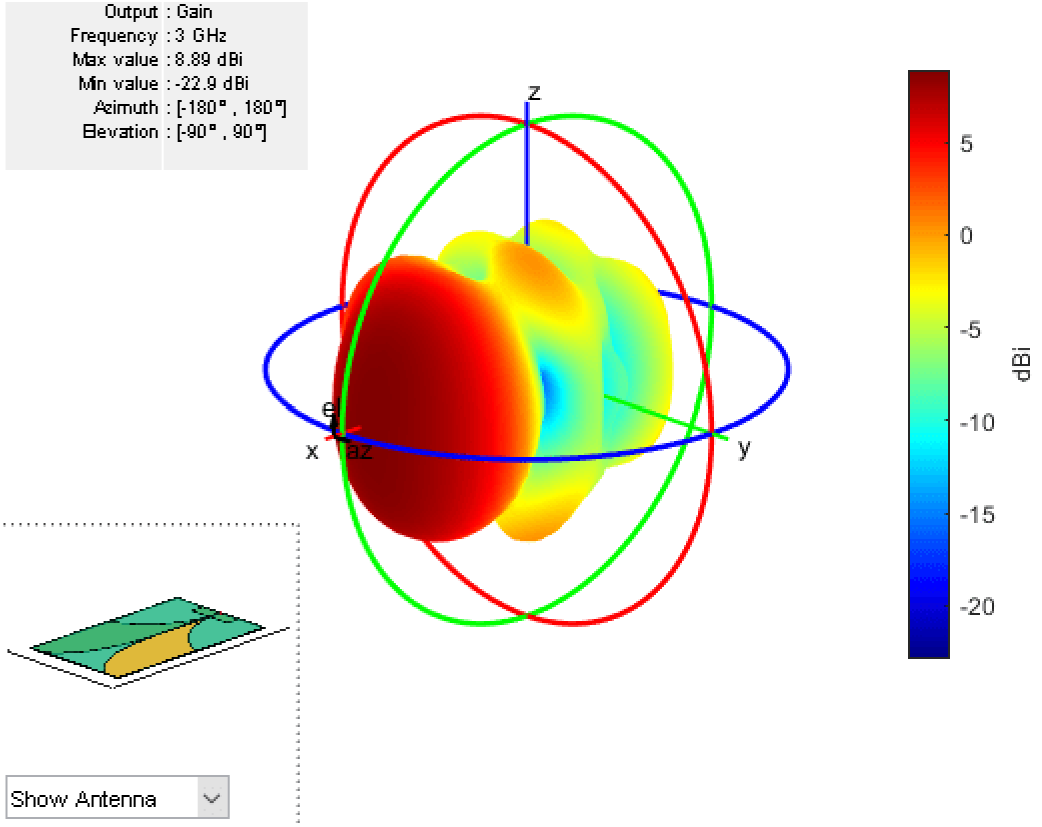
Radiation pattern of Antipodal Vivaldi Antenna

Plot the radiation pattern of the antipodal Vivaldi antenna at 3 GHz

```
avi=vivaldiAntipodal("OpeningRate",30,'Substrate',dielectric('Name','RO4003C','EpsilonR',3.38,'L
'Thickness',0.508e-3))
```

```
avi = 
  vivaldiAntipodal with properties:

         BoardLength: 0.2020
          BoardWidth: 0.1200
              Height: 5.0800e-04
         OpeningRate: 30
      StripLineWidth: 0.0011
    OuterTaperLength: 0.0800
    InnerTaperLength: 0.1870
       ApertureWidth: 0.0840
    GroundPlaneWidth: 0.0500
           Substrate: [1x1 dielectric]
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1x1 lumpedElement]
```

```
pattern(avi,3e9)
```



Output : Gain
Frequency : 3 GHz
Max value : 8.89 dBi
Min value : -22.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2020a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`vivaldi` | `yagiUda` | `spiralArchimedean` | `slot`

Topics

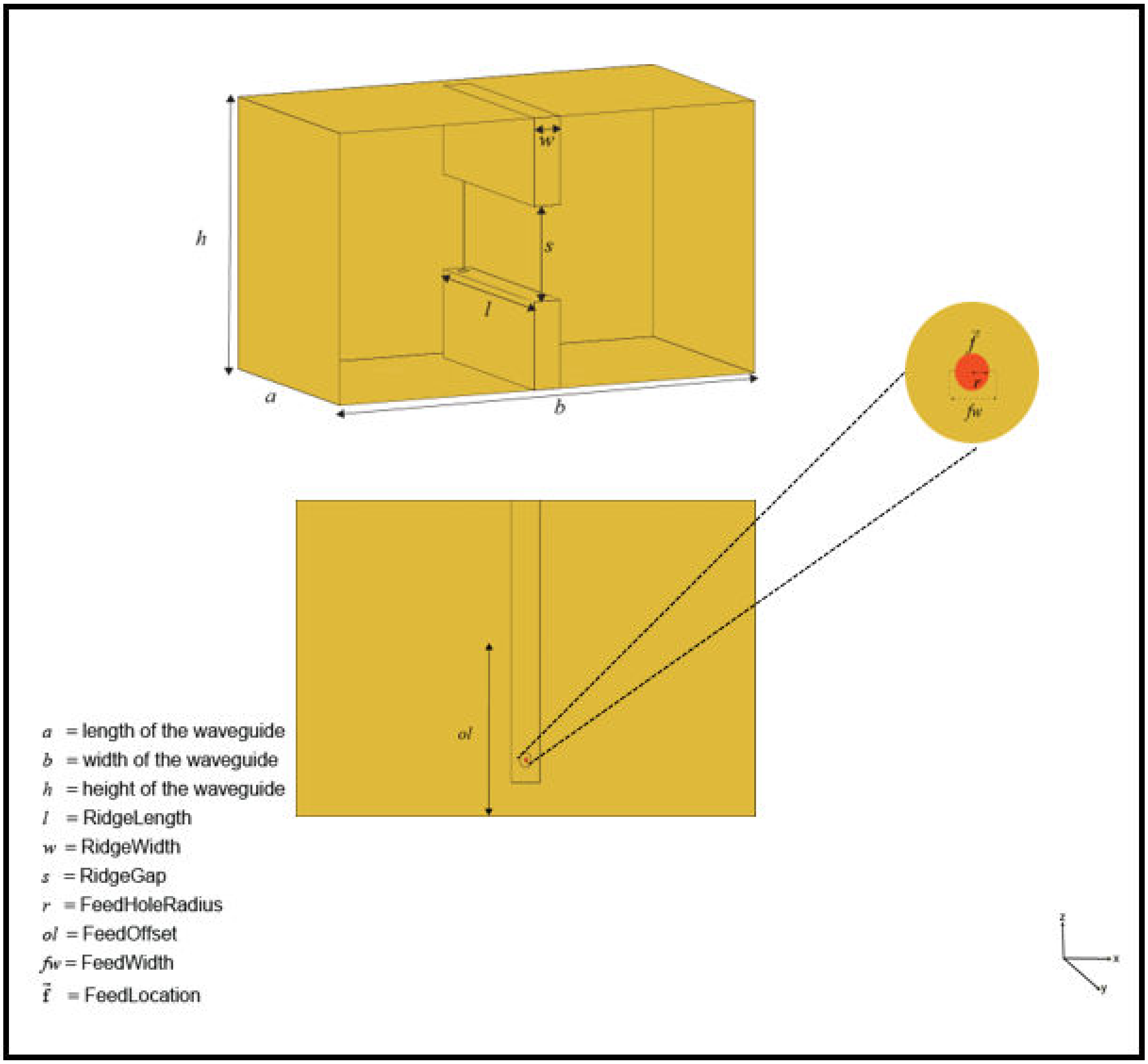
“Rotate Antennas and Arrays”

waveguideRidge

Create ridged waveguide antenna

Description

The `waveguideRidge` object creates a dual ridged waveguide antenna. The ridges ensure a smooth transition from an input impedance of 50 ohms to the impedance of free space (377 ohms). The dual ridged waveguide antenna widely used in ultra-wideband applications covering a large spectrum of frequencies. Ridged waveguide antennas are used in radio astronomy applications.



Creation

Syntax

```
ant = waveguideRidge
ant = waveguideRidge(Name,Value)
```

Description

`ant = waveguideRidge` creates a double-ridged waveguide antenna. The default `waveguideRidge` antenna object is centered on the *xy*-plane. The object dimensions are chosen for an operating frequency of 8-10 GHz.

`ant = waveguideRidge(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = waveguideRidge('Height',1)` creates a ridge waveguide with a height of 1 meter.

Note Properties you do not specify retain their default values.

Properties

`Length` — Length of waveguide
`0.0210` (default) | positive real-valued scalar

Length of the waveguide, specified as a real-valued scalar in meters.

Example: `'Length',0.0410`

Example: `ant.Length = 0.0410`

Data Types: `double`

`Width` — Width of waveguide
`0.0400` (default) | positive real-valued scalar

Width of the waveguide, specified as a real-valued scalar in meters.

Example: `'Width',0.0640`

Example: `ant.Width = 0.0640`

Data Types: `double`

`Height` — Height of waveguide
`0.0250` (default) | positive real-valued scalar

Height of the waveguide, specified as a real-valued scalar in meters.

Example: `'Height',0.0340`

Example: `ant.Height = 0.0340`

Data Types: `double`

`RidgeLength` — Length of Ridge
`0.01875` (default) | positive real-valued scalar

Length of the ridge, specified as a real-valued scalar in meters.

Example: `'RidgeLength',0.0220`

Example: `ant.RidgeLength = 0.0220`

Data Types: `double`

`RidgeWidth` — Width of Ridge
`0.0025` (default) | positive real-valued scalar

Width of the ridge, specified as a real-valued scalar in meters.

Example: `'RidgeWidth',0.0030`

Example: `ant.RidgeLength = 0.0060`

Data Types: `double`

`RidgeGap` — Gap between two ridges
`0.0088` (default) | real-valued scalar

Gap between two ridges, specified as a real-valued scalar in meters.

Example: `'RidgeGap',0.0070`

Example: `ant.RidgeGap = 0.0098`

Data Types: `double`

`FeedHoleRadius` — Radius of feeding hole
`0.00005` (default) | real-valued scalar

Radius of the feeding hole, specified as a real-valued scalar in meters.

Example: `'FeedHoleRadius',0.00006`

Example: `ant.FeedHoleRadius = 0.00010`

Data Types: `double`

`FeedWidth` — Width of feed
`0.0001` (default) | positive real-valued scalar

Width of the feed, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0010`

Example: `ant.FeedWidth = 0.0020`

Data Types: `double`

`FeedOffset` — Signed distances from origin
`[-0.0675 0]` (default) | two-element vector

Signed distances from the origin measured along the length and width of the waveguide, specified as a two-element vector with each element in meters.

Example: `'FeedOffset',[-0.0725 0]`

Example: `ant.FeedOffset = [-0.0830 0]`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedelement` object

Lumped elements added to the antenna feed, specified as `lumpedelement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. The`lumpedelement` is a object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure

Examples

Create a Ridged Waveguide Antenna with Default Properties

Create `a waveguideRidge antenna` object with default properties.

```
a = waveguideRidge
```

```
a = 
  waveguideRidge with properties:

           Length: 0.0210
            Width: 0.0400
           Height: 0.0250
      RidgeLength: 0.0187
       RidgeWidth: 0.0025
         RidgeGap: 0.0088
   FeedHoleRadius: 5.0000e-04
        FeedWidth: 1.0000e-04
       FeedOffset: [-0.0067 0]
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
```

```
              Load: [1x1 lumpedElement]
```

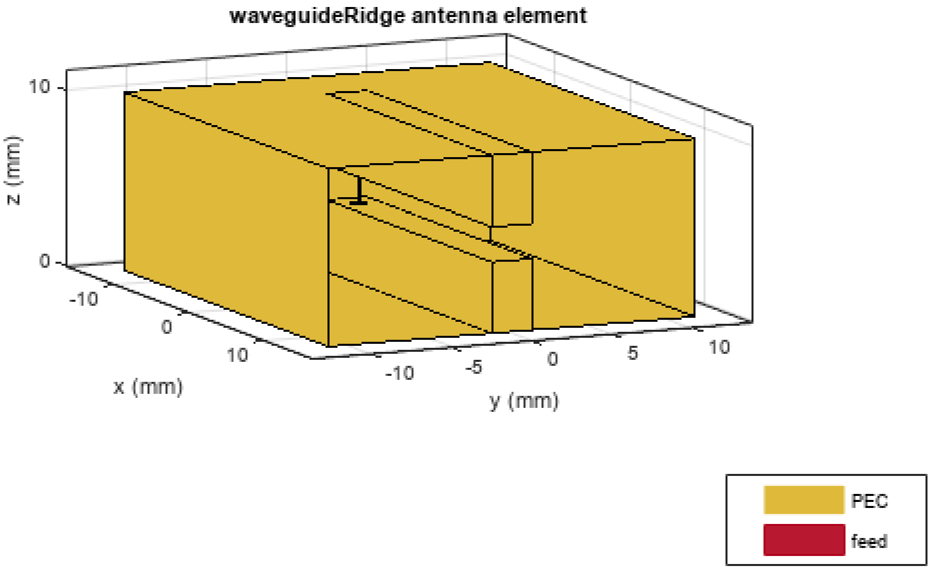
Create a Ridged Waveguide Antenna with Specified Properties

Create a `waveguideRidge` antenna object with the properties specified.

```
h = waveguideRidge('Length',0.0273,'Width', 0.02286,'RidgeGap',2e-3, ...
          'Height', 0.01016,'RidgeLength',0.022);
```

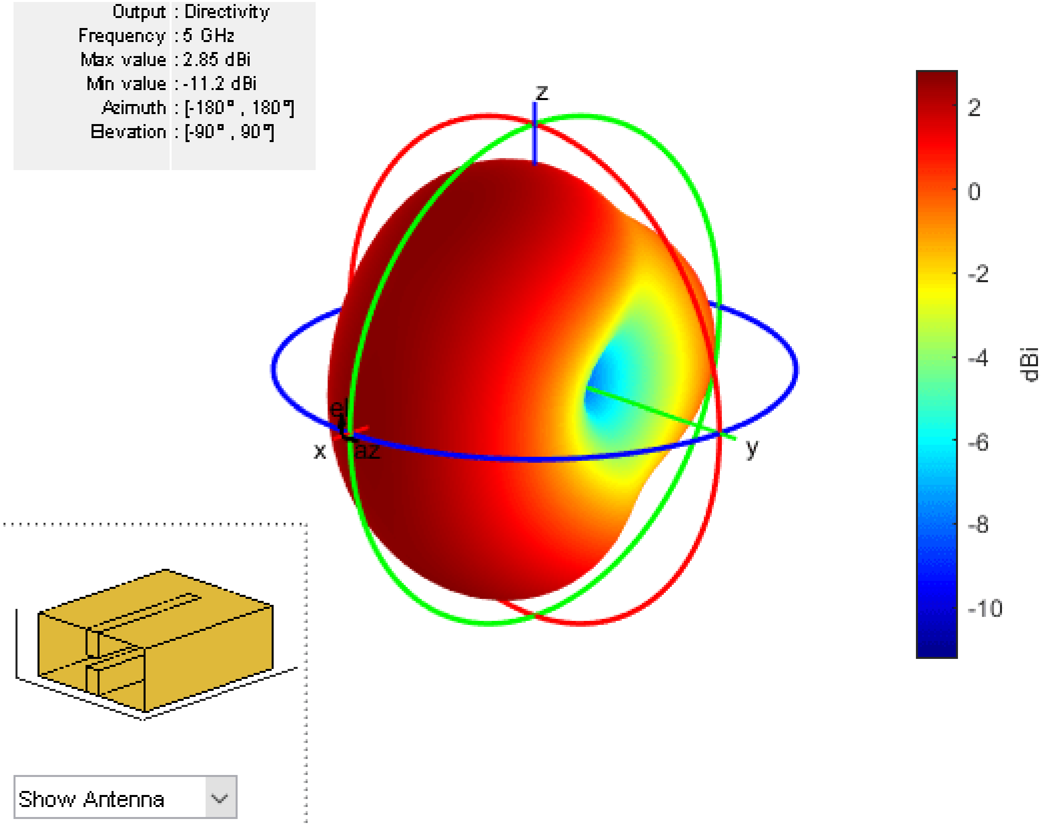
View the `waveguideRidge` antenna using a `show` function.

```
figure
show(h)
```



Plot the 3-D radiation pattern of the `waveguideRidge` antenna at 5 GHz.

```
pattern(h,5e9)
```



Output : Directivity
Frequency : 5 GHz
Max value : 2.85 dBi
Min value : -11.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2019b

See Also

`waveguide` | `waveguideCircular` | `cavityCircular` | `waveguideSlotted`

Topics

“Rotate Antennas and Arrays”

wireStack

Create single or multifeed wire antenna

Description

The `wireStack` object converts all applicable elements in the antenna library to wire antennas with single or multiple feeds. You can now create cylindrical thin-wire antennas and analyze them using with the existing antenna analysis functions.

Note For some antennas, the wire geometry may be altered to allow the placement of the feed. Please use the `show` function to view the resulting antenna and verify its shape.

Creation

Syntax

```
ant = wireStack
ant = wireStack(libant)
```

Description

`ant = wireStack` creates a half-wavelength wire dipole antenna. The default wire dipole is center fed with the feedpoint at the origin, and it is located along the Z-axis. The antenna length is chosen for an operating frequency of 75 MHz.

`ant = wireStack(libant)` converts a strip-based antenna from the Antenna Toolbox library to a wire antenna for further analysis. Conversion is based on the equivalent radius using the `strip2cylinder` utility function.

Input Arguments

`libant` — Antenna elements
antenna element object

Antenna elements, specified as any one of the following antenna element objects: `dipole`, `dipoleFolded`, `dipoleMeander`, `dipoleVee`, `dipoleHelix`, `dipoleJ`, `dipoleCycloid`, `dipoleCrossed`, `loopCircular`, and `loopRectangular`.

Output Arguments

`ant` — Wire antenna
`wireStack` object (default)

Wire antenna, returned as a `wireStack` object.

Properties

Name — Name of wire antenna
'Dipole' (default) | string scalar

Name of the wire antenna, specified as a string scalar.

Example: `ant.Name = 'monopole'`

Data Types: `string`

FeedLocation — Antenna feed locations
[0 0 0] (default) | *N*-by-3 array of Cartesian coordinates

This property is read-only.

Antenna feed locations, specified as an *N*-by-3 array of Cartesian coordinates with each element unit in meters.

Example: `ant.FeedLocation = [0 2 4]`

Data Types: `double`

FeedVoltage — Magnitude of excitation voltage at each feed
1 (default) | 1-by-*N* array of doubles

Magnitude of excitation voltage at each feed, specified as a 1-by-*N* array of doubles.

Example: `ant.FeedVoltage = 2`

Data Types: `double`

FeedPhase — Phase shift applied to voltage at each feed
0 (default) | 1-by-*M* array of doubles

Phase shift applied to voltage at each feed, specified as a 1-by-*M* array of doubles.

Example: `ant.FeedVoltage = 60`

Data Types: `double`

Tilt — Tilt angle of antenna
0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

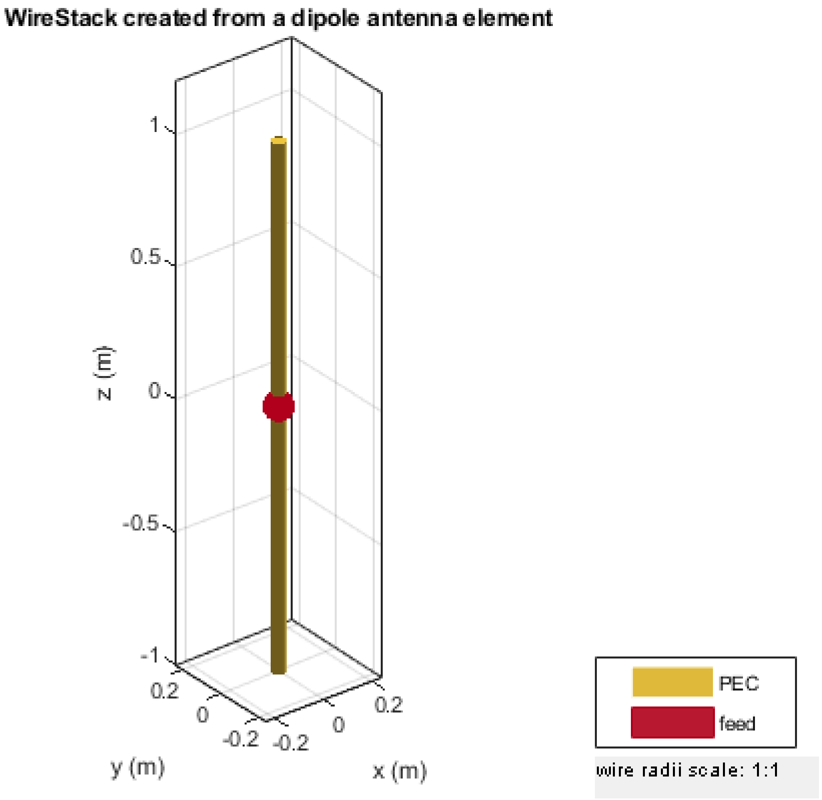
Examples

Default Wire Antenna

Create and view a default dipole wire antenna.

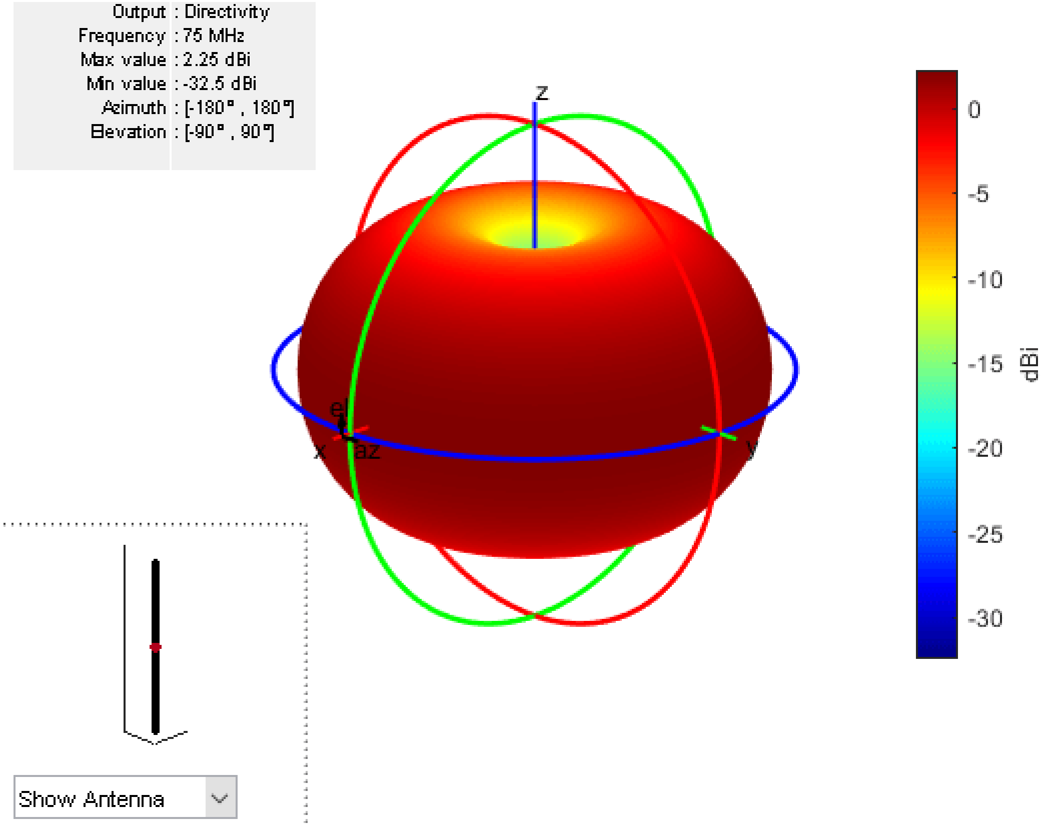
```
ant = wireStack
ant = 
  wireStack with properties:

            Name: 'Dipole'
    FeedLocation: [0 0 0]
     FeedVoltage: 1
       FeedPhase: 0
            Tilt: 0
       TiltAxis: [1 0 0]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at the specified frequency.

```
pattern(ant,75e6)
```



Output : Directivity
Frequency : 75 MHz
Max value : 2.25 dBi
Min value : -32.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2020a

See Also

`loopCircular` | `strip2cylinder` | `dipole` | `dipoleFolded` | `dipoleMeander` | `dipoleVee` | `dipoleHelix` | `dipoleJ` | `dipoleCycloid` | `dipoleCrossed` | `loopRectangular`

Topics

“Modeling Wire Antenna and Arrays”
“Rotate Antennas and Arrays”
“Wire Solver”

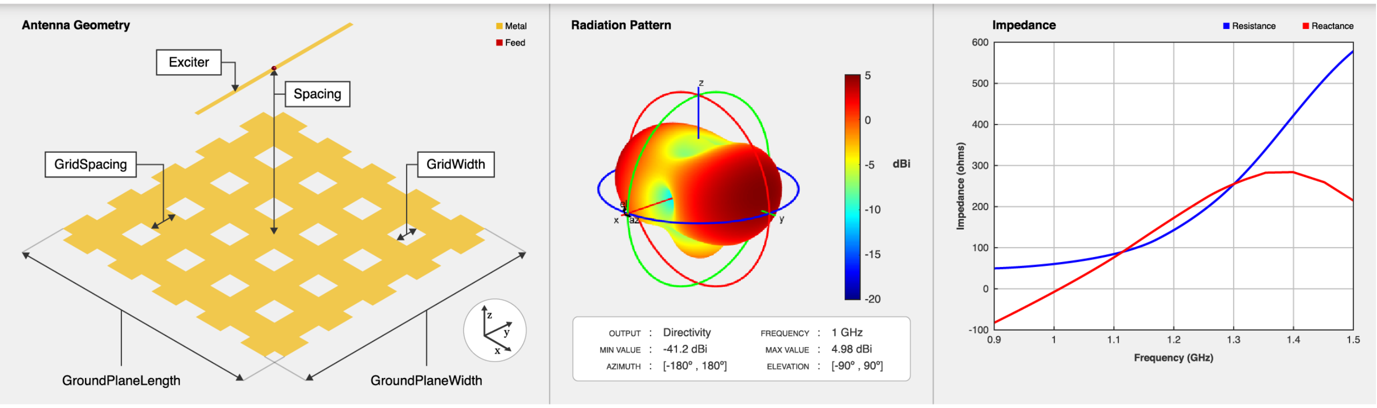
reflectorGrid

Create grid reflector-backed antenna

Description

The `reflectorGrid` object creates a grid reflector-backed antenna. The grid reflector uses a grid of parallel wires or bars oriented in one direction. Grid reflectors can be used as high-gain antennas in point-to-point communications.

Reflector Grid



Creation

Syntax

```
ant = reflectorGrid
ant = reflectorGrid(Name,Value)
```

Description

`ant = reflectorGrid` creates a grid reflector-backed antenna. The default antenna object has an exciter as a dipole with the feed point located at the origin on the X-Y plane, and the antenna dimensions are chosen for an operating frequency of 1 GHz.

`ant = reflectorGrid(Name,Value)` sets “Properties” on page 1-647 using name-value pairs. For example, `reflectorGrid('GroundPlaneWidth',0.6)` creates a grid reflector with a width of 0.6 meters. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Properties

`Exciter` — Antenna or array type used as exciter
`dipole` (default) | `antenna` object | `array` object

Antenna type used as an exciter, specified as any single-element antenna object. Except reflector and cavity antenna elements, you can use any of the antenna elements or array elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',horn`

Example: `ant.Exciter = horn`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Spacing` — Distance between reflector and exciter

`0.175` (default) | positive scalar

Distance between reflector and exciter, specified as a positive scalar in meters.

Example: `'Spacing',0.259`

Example: `ant.Spacing = 0.195`

Data Types: `double`

`GroundPlaneLength` — Reflector length

`0.2` (default) | positive scalar

Reflector length along the X-axis, specified as a positive scalar in meters.

Example: `'GroundPlaneLength',0.6`

Example: `ant.GroundPlaneLength = 0.18`

Data Types: `double`

`GroundPlaneWidth` — Reflector width

`0.2` (default) | positive scalar

Reflector width along the Y-axis, specified as a positive scalar in meters.

Example: `'GroundPlaneWidth',0.6`

Example: `ant.GroundPlaneWidth = 0.18`

Data Types: `double`

`GridType` — Type of grid used in reflector

`'HV'` (default) | `'VH'` | `'V'` | `'H'` | character vector | string scalar

Type of the grid used in the reflector, specified as either one of the following:

- `'H'` — grids are arranged horizontally in the reflector.
- `'V'` — grids are arranged vertically in the reflector.
- `'HV'` or `'VH'` — grids are arranged both horizontally and vertically in the reflector.

Example: `'GridType','H'`

Example: `ant.GridType = 'V'`

Data Types: `char`

`GridSpacing` — Distance between two grid cells

`0.018` (default) | positive scalar

Distance between the two grid cells, specified as a positive scalar in meters.

Example: `'GridSpacing',0.018`

Example: `ant.GridSpacing = 0.014`

Data Types: `double`

`GridWidth` — Width of each grid cell

`0.022` (default) | positive scalar

Width of each grid cell, specified as a positive scalar in meters.

Example: `'GridWidth',0.3`

Example: `ant.GridWidth = 0.28`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelements`, where `lumpedelements` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

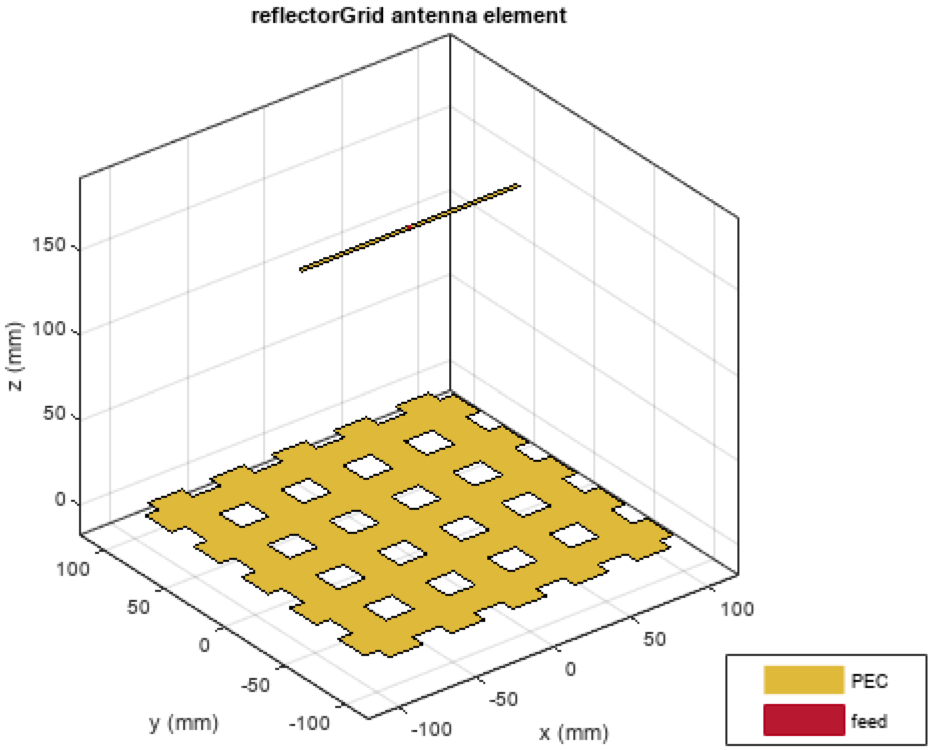
show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
current	Current distribution on antenna or array surface
charge	Charge distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
optimize	Optimize antenna or array using SADEA optimizer
design	Design prototype antenna or arrays for resonance around specified frequency
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
numGridsToSpacing	Calculate grid spacing in grid for reflectorGrid object

Examples

Design Grid Reflector-Backed Antenna with Default Properties

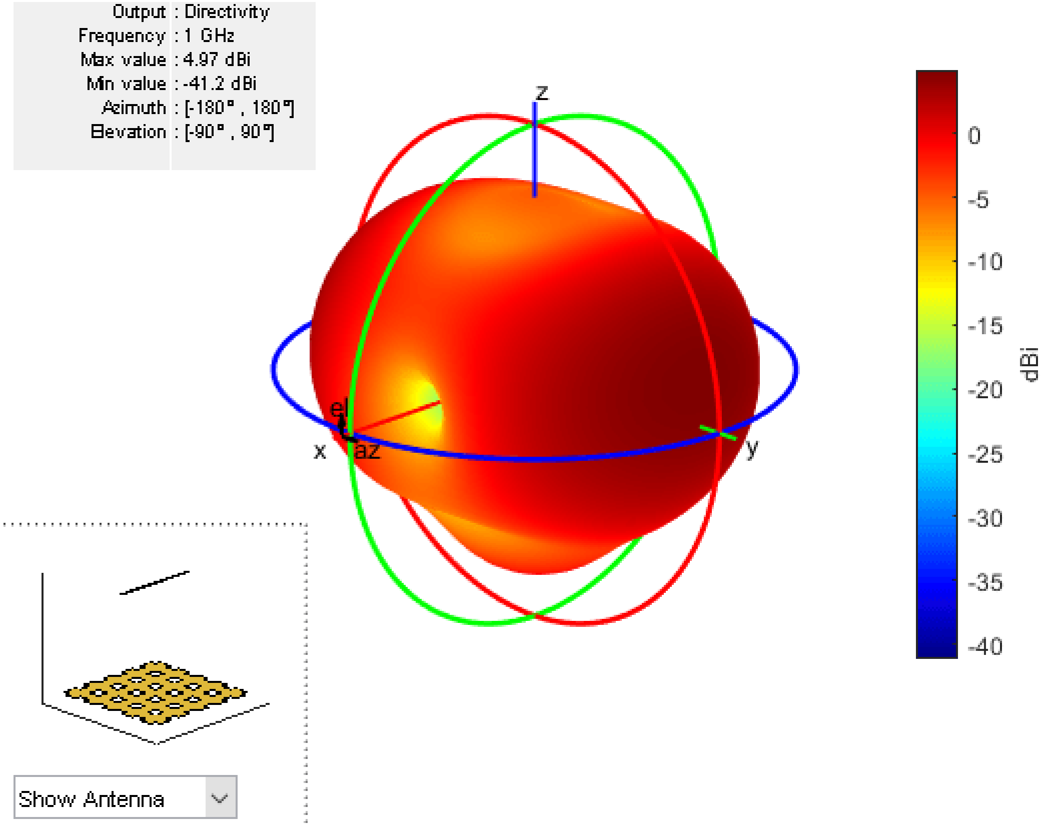
Create and view a grid reflector-backed antenna object with default properties.

```
ant = reflectorGrid;
show(ant)
```



Plot the radiation pattern of the antenna at 1 GHz.

```
pattern(ant,1e9)
```

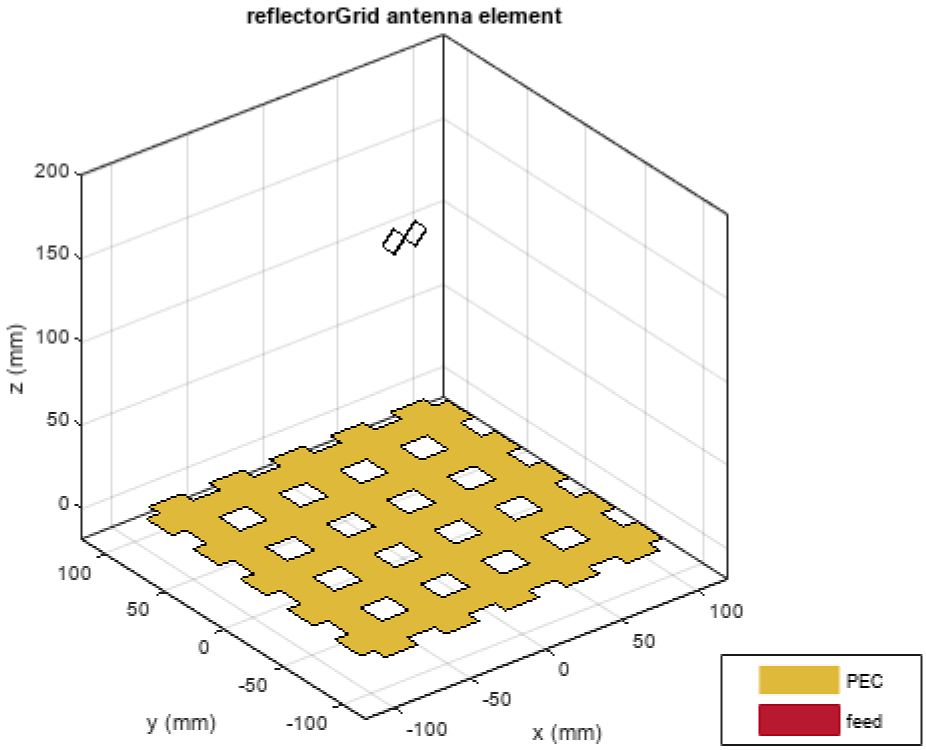


Output : Directivity
Frequency : 1 GHz
Max value : 4.97 dBi
Min value : -41.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Design Grid Reflector-Backed Biquad Antenna

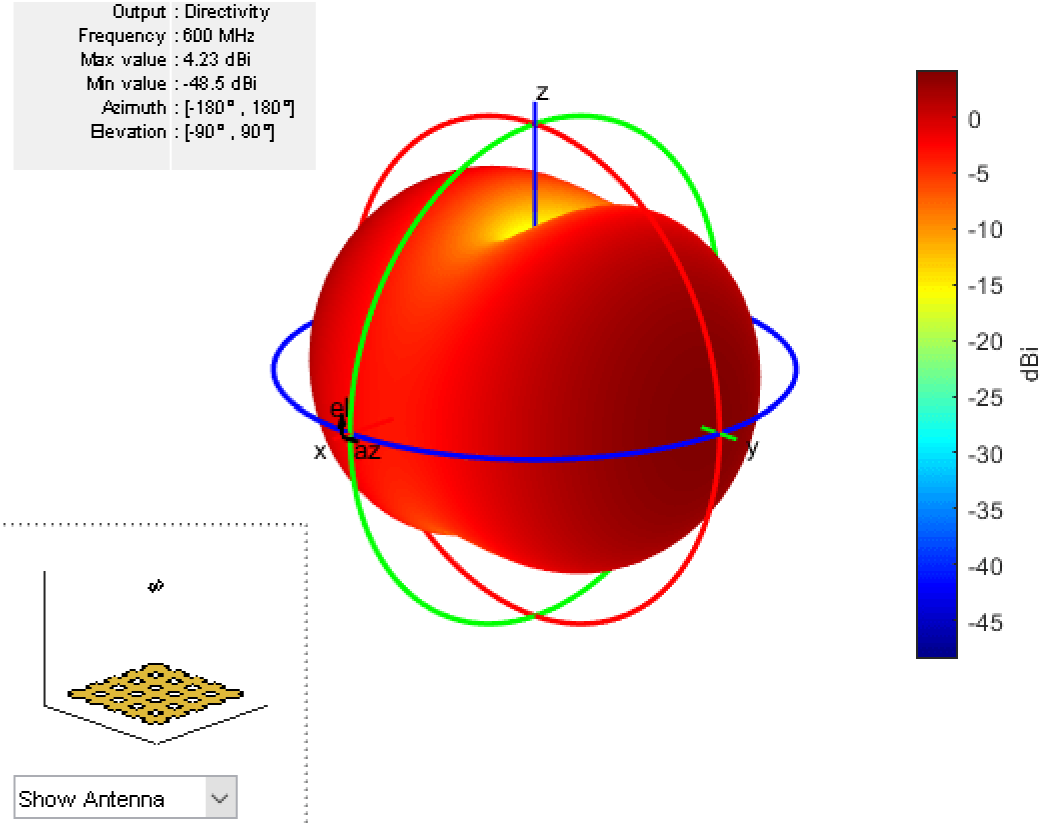
Create and view a grid reflector-backed biquad antenna with an arm length of 0.01 meters.

```
d = biquad('ArmLength',0.01);
h = reflectorGrid('Exciter',d);
show(h)
```



Plot the radiation pattern of the antenna at 0.6 GHz.

```
pattern(h,0.6e9)
```

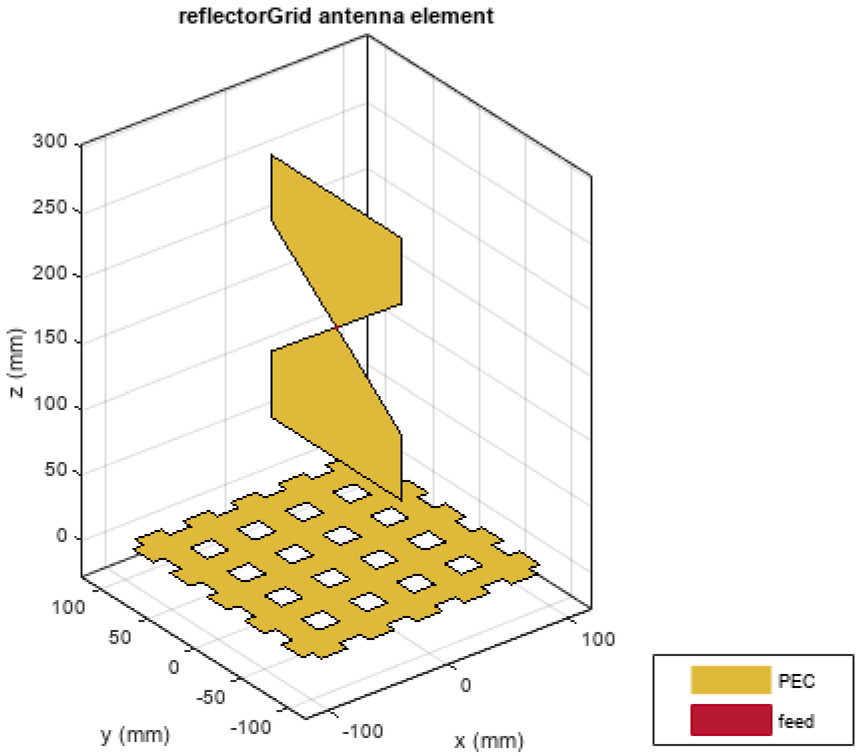


```
   Output : Directivity
Frequency : 600 MHz
Max value : 4.23 dBi
Min value : -48.5 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Change Grid Type in Grid Reflector-Backed Antenna

Create and view grid reflector-baked dipole blade antenna.

```
d = dipoleBlade('Length',0.1,'TaperLength',0.05,'FeedGap',0.002);
h = reflectorGrid('Exciter',d);
show(h)
```

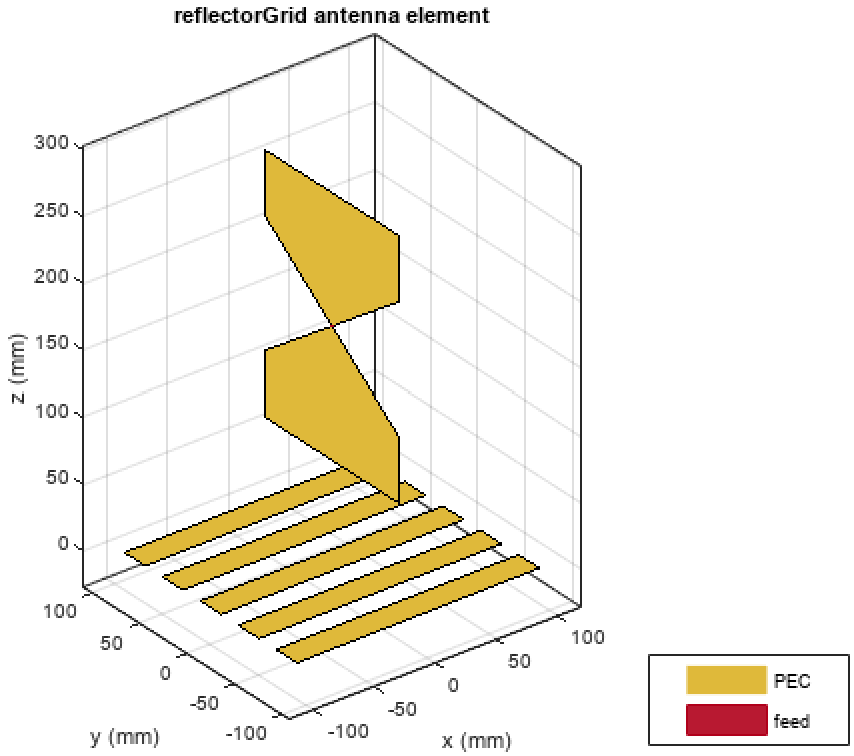


Change the grid type from `'HV'` to `'H'`.

```
h.GridType = 'H';
```

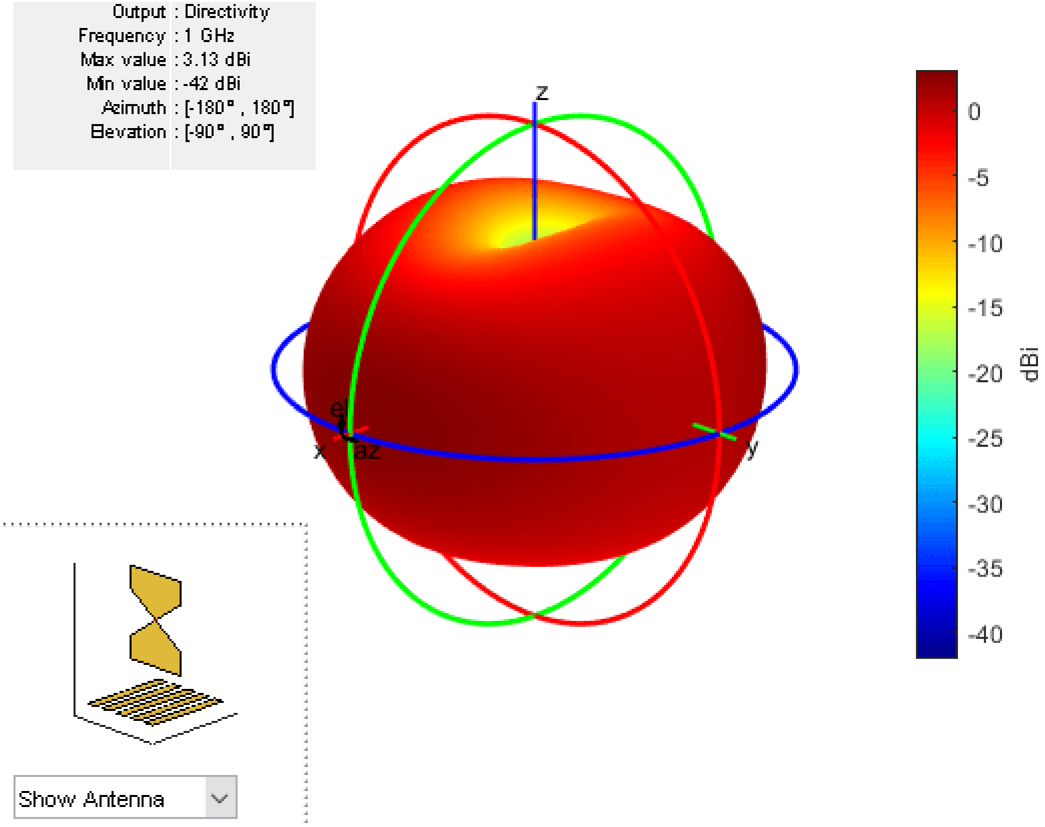
View the antenna with grid type `'H'`.

```
show(h)
```



Plot the radiation pattern at 1 GHz.

```
pattern(h,1e9)
```



```
   Output : Directivity
Frequency : 1 GHz
Max value : 3.13 dBi
Min value : -42 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Create Grid Reflector-Backed Rectangular Array

Create a rectangular array of cylindrical dipole antennas.

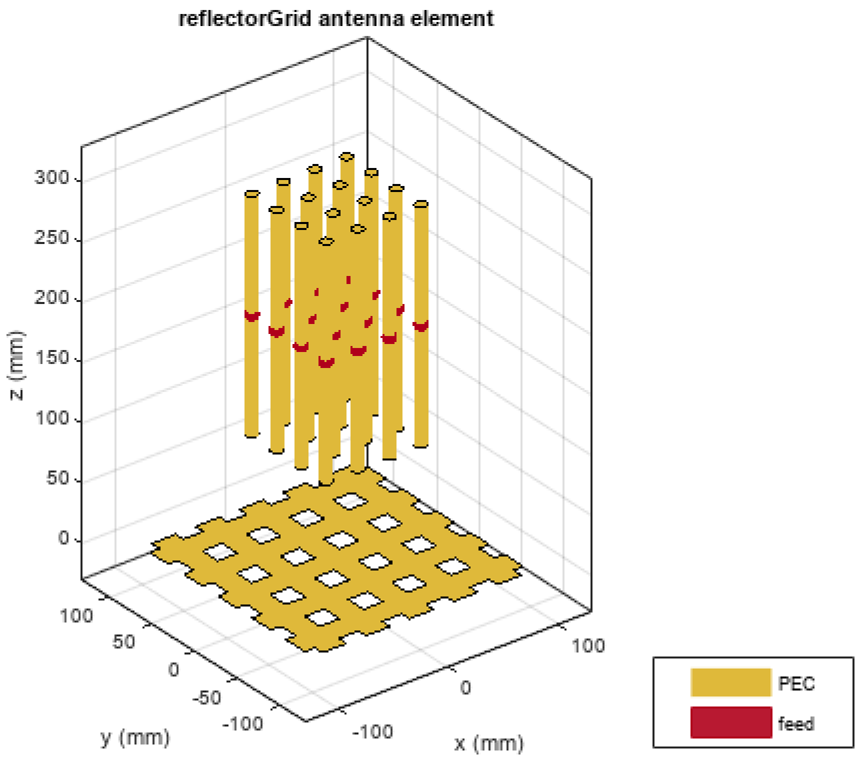
```
d = dipoleCylindrical('Length',0.2,'Radius',0.005);
arr = rectangularArray('Element',d,'Size',[4 4],'RowSpacing',0.029,'ColumnSpacing',0.029);
```

Create a grid reflector-backed rectangular array.

```
ant = reflectorGrid('Exciter',arr,'Spacing',0.2)
```

```
ant = 
  reflectorGrid with properties:

              Exciter: [1x1 rectangularArray]
              Spacing: 0.2000
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.2000
             GridType: 'HV'
          GridSpacing: 0.0180
            GridWidth: 0.0220
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



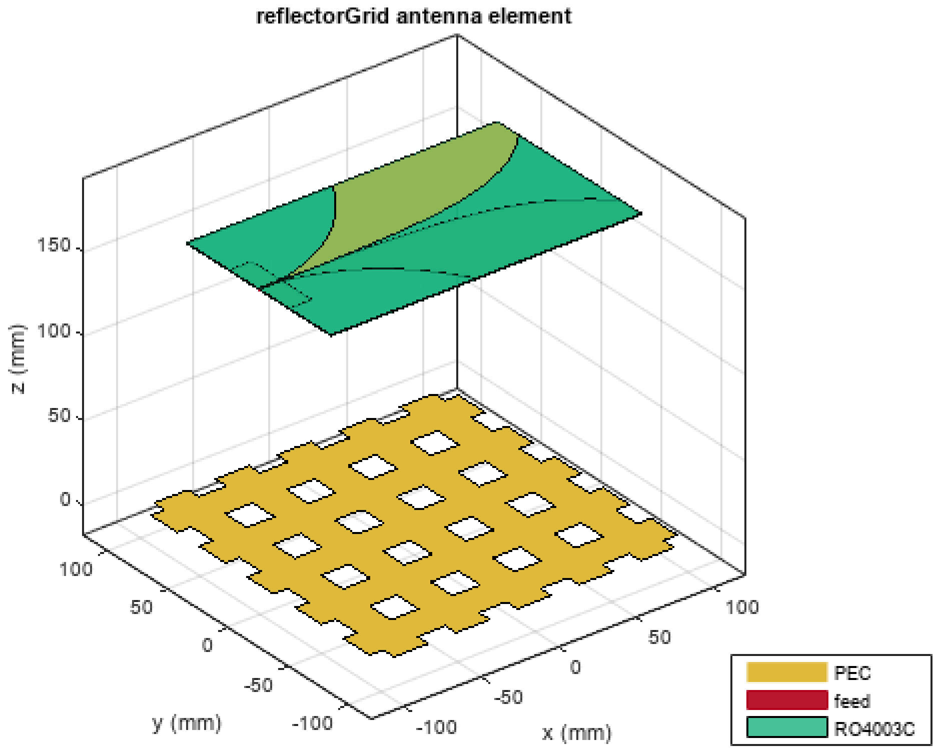
Create Antipodal Vivaldi Antenna with Grid Reflector Backing Structure

Create and visualize a grid reflector-backed antipodal Vivaldi antenna.

```
e = vivaldiAntipodal;
ant = reflectorGrid('Exciter',e)
```

```
ant = 
  reflectorGrid with properties:

              Exciter: [1x1 vivaldiAntipodal]
              Spacing: 0.1750
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.2000
             GridType: 'HV'
          GridSpacing: 0.0180
            GridWidth: 0.0220
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2020b

References

[1] Balanis, Constantine A. *Antenna Theory: Analysis and Design*. 3rd ed. Hoboken, NJ: John Wiley, 2005.

See Also

`reflector` | `reflectorParabolic` | `reflectorCorner` | `reflectorCircular`

Topics

"Rotate Antennas and Arrays"

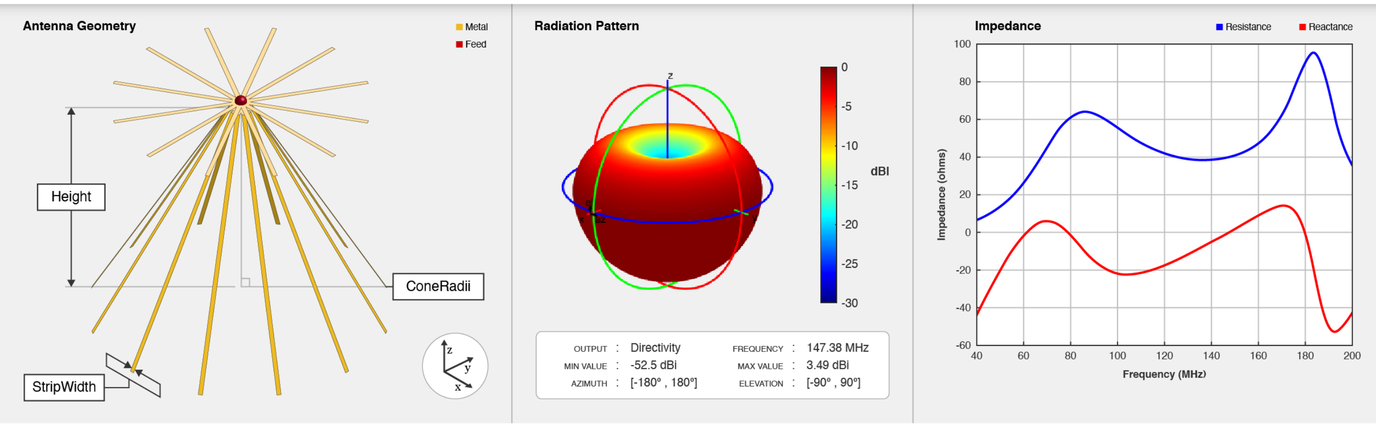
disconeStrip

Create strip discone antenna

Description

The `disconeStrip` antenna object creates a strip discone antenna. The strip discone antenna is an approximation to a solid discone antenna, where the cone and the disc are replaced with strips. The strip discone antennas are lighter in weight and suited for applications in high frequency (HF) and very high frequency (VHF) bands.

Discone Strip



Creation

Syntax

```
ant = disconeStrip
ant = disconeStrip(Name,Value)
```

Description

`ant = disconeStrip` creates a strip discone antenna with dimensions for a resonant frequency of 147.38 MHz. The default strip discone antenna has a feedpoint at the center of the disc.

`ant = disconeStrip(Name,Value)` sets “Properties” on page 1-660 using name-value pairs. For example, `disconeStrip('NumStrips',8)` creates a discone strip antenna with eight strips. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Properties

`NumStrips` — Number of strips

12 (default) | scalar in the range [6, 64]

Number of strips to form the cone and the disc, specified as a scalar in the range [6, 64]. The number of strips can be increased to increase the impedance bandwidth of the `disconeStrip` antenna object.

Example: `'NumStrips',8`

Example: `ant.NumStrips = 14`

Data Types: `double`

`StripWidth` — Width of strip
`20e-3` (default) | scalar

Width of each strip in the strip discone antenna, specified as a scalar in meters.

Example: `'StripWidth',10e-3`

Example: `ant.StripWidth = 15.8e-3`

Data Types: `double`

`Height` — Vertical height between broad and narrow diameter of cone
`1.308` (default) | scalar

Vertical height between the maximum or broad diameter and the minimum or narrow diameter of the cone, specified as a scalar in meters. The vertical height can be decreased to increase the operating frequency.

Example: `'Height',1.59`

Example: `ant.Height = 1.89`

Data Types: `double`

`ConeRadii` — Radii of cone
`[65e-3 810e-3]` (default) | two-element vector

Radii of the cone, specified as a two-element vector in meters. In the two element vector, the first element specifies the narrow or minimum radius and second element specifies the broad or maximum radius of the cone. The radii of the cone can be decreased to increase the operating frequency and high-frequency input impedance.

Example: `'ConeRadii',[63e-3 840e-3]`

Example: `ant.ConeRadii = [65e-3 910e-3]`

Data Types: `double`

`DiscRadius` — Radius of disc
`700e-3` (default) | scalar

Radius of the disc, specified as a scalar in meters. The radius of the disc can be decreased to increase the operating frequency and it can be increased to increase the low-frequency input impedance.

Note `DiscRadius` should be smaller than the “ConeRadii” on page 1-0 .

Example: `'DiscRadius',900e-3`

Example: `ant.DiscRadius = 829e-3`

Data Types: `double`

`FeedHeight` — Gap between cone and disc

`30e-3` (default) | scalar

Gap between the cone and the disc, specified as a scalar in meters. This gap represents height of the field and the gap can be decreased to increase the high-frequency input impedance.

Example: `'FeedHeight',34e-3`

Example: `ant.FeedHeight = 34e-3`

Data Types: `double`

`FeedWidth` — Diameter of feed

`20e-3` (default) | scalar

Diameter of the feed, specified as a scalar in meters.

Example: `'FeedWidth',25e-3`

Example: `ant.FeedWidth = 21e-3`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelements`, where `lumpedelements` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

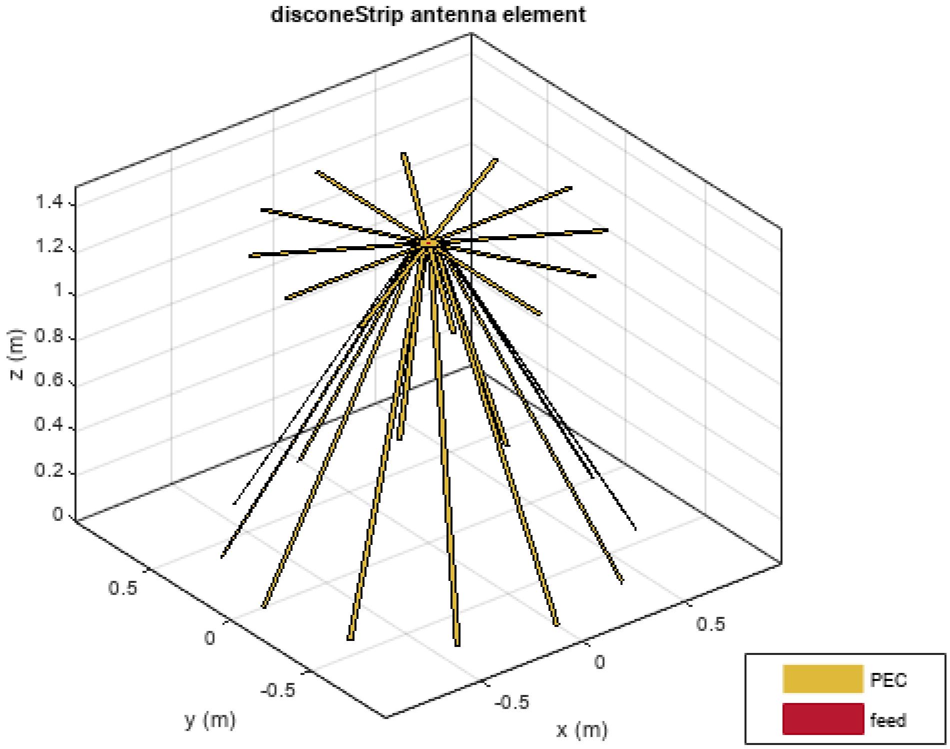
coneangle2size	Calculates equivalent cone height, broad radius, and narrow radius for cone
show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
current	Current distribution on antenna or array surface
charge	Charge distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
design	Design prototype antenna or arrays for resonance around specified frequency
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array

Examples

Design Strip Discone Antenna with Default Properties

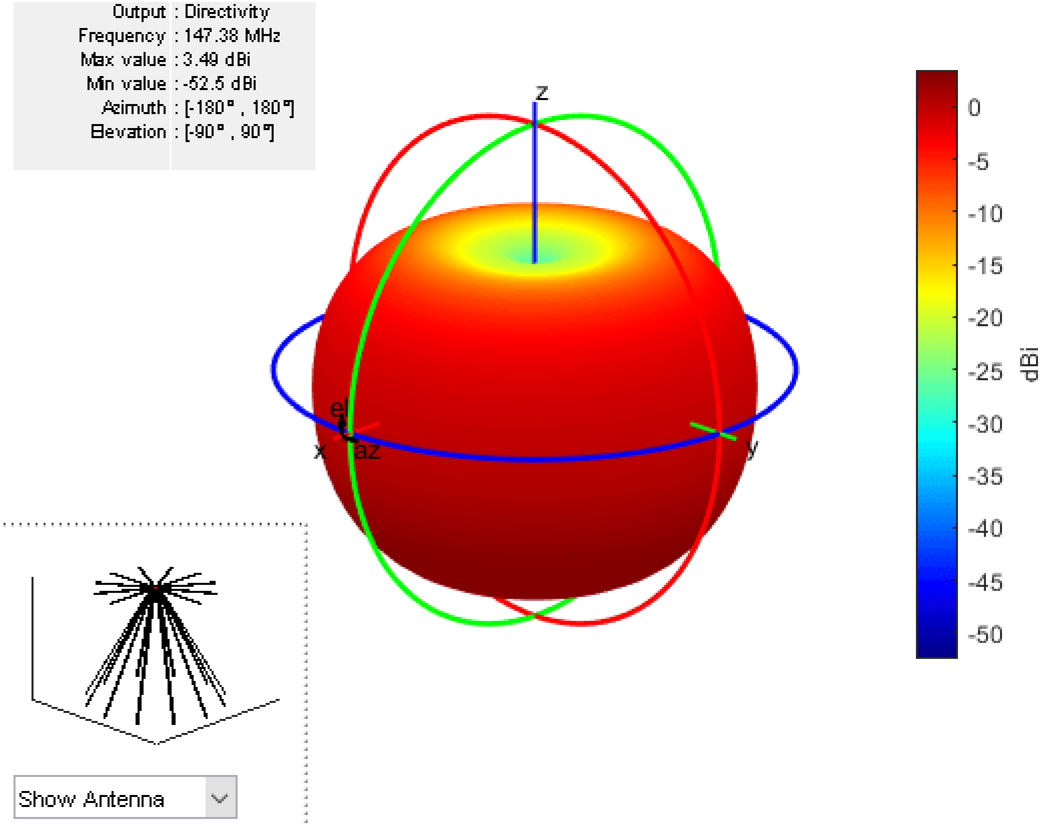
Create and view a strip discone antenna with default properties.

```
ant = disconeStrip;
show(ant)
```



Plot the radiation pattern of the antenna at 147.38 MHz.

```
pattern(ant, 147.38e6)
```

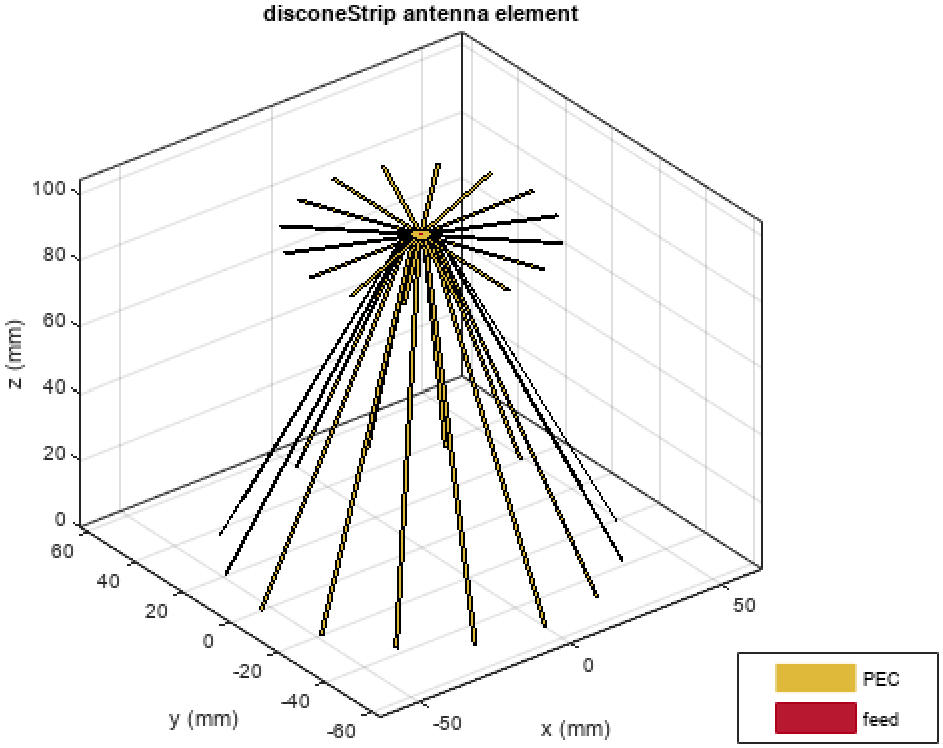


```
   Output : Directivity
Frequency : 147.38 MHz
Max value : 3.49 dBi
Min value : -52.5 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Create Strip Discone Antenna and Plot S-Parameters

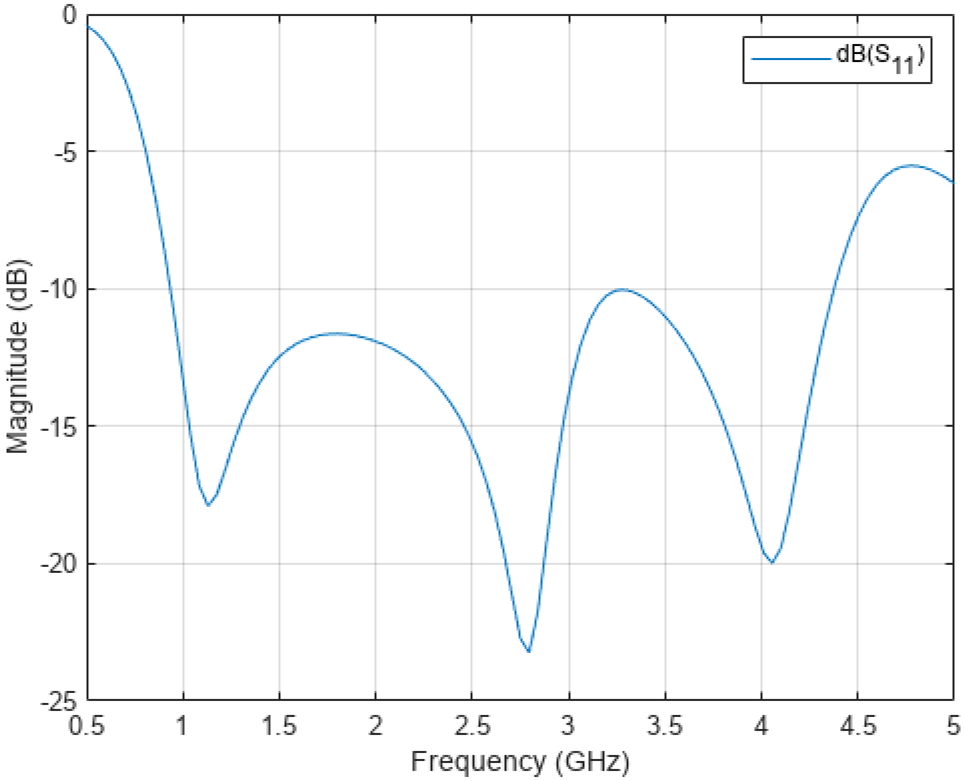
Create and view a strip discone antenna object with the specifed properties.

```
ant = disconeStrip('Height',92e-3,'ConeRadii',[5.5e-3 53e-3],'DiscRadius', 37e-3,'NumStrip',16,.
    'StripWidth',1e-3,'FeedWidth',0.5e-3,'FeedHeight',2.2e-3);
show(ant)
```



Plot the S-Parameters of the antenna over the frequency span of 500 MHz to 5 GHz.

```
s = sparameters(ant,linspace(500e6,5e9,101));
figure
rfplot(s)
```



More About

Parametric Analysis Guidelines

To understand how the properties of the `disconeStrip` antenna object influence the antenna design, use the following parametric analysis guidelines.

- To increase the operating frequency, decrease the dimensions of the `disconeStrip` antenna object using the “ConeRadii” on page 1-0 , “DiscRadius” on page 1-0 , and “Height” on page 1-0 properties.
- To increase the impedance bandwidth, increase the number of strips in the `disconeStrip` antenna object using the “NumStrips” on page 1-0 property.
- To improve high-frequency input impedance, decrease the cone radii and feed height of the antenna using the “ConeRadii” on page 1-0 and the “FeedHeight” on page 1-0 properties.
- To increase low frequency input impedance, increase the radius of the disc using the “DiscRadius” on page 1-0 property.

Version History

Introduced in R2020b

References

[1] Khumanthem.T., C.Sairam, S.D.Ahirwar and M.Balachary. ''Compact Discone Antenna with Small Form Factor in VHF Band'' *EWCI*, 2014.

[2] Ki-Hak Kim, Jin-U Kim, and Seong-Ook Park. “An Ultrawide-Band Double Discone Antenna with the Tapered Cylindrical Wires.” *IEEE Transactions on Antennas and Propagation* 53, no. 10 (October 2005): 3403–6. https://doi.org/10.1109/TAP.2005.856036.

[3] Tai C-T. and S. A. Long. ''Dipoles and Monopoles'' in *Antenna Engineering Handbook*, 4th ed., J. L. Volakis (Ed.), McGraw-Hill, 2007.

[4] McDonald, James L., and Dejan S. Filipovic. “On the Bandwidth of Monocone Antennas.” *IEEE Transactions on Antennas and Propagation* 56, no. 4 (April 2008): 1196–1201. https://doi.org/10.1109/TAP.2008.919226.

See Also

`discone` | `bicone` | `monocone` | `biconeStrip`

Topics

“Rotate Antennas and Arrays”

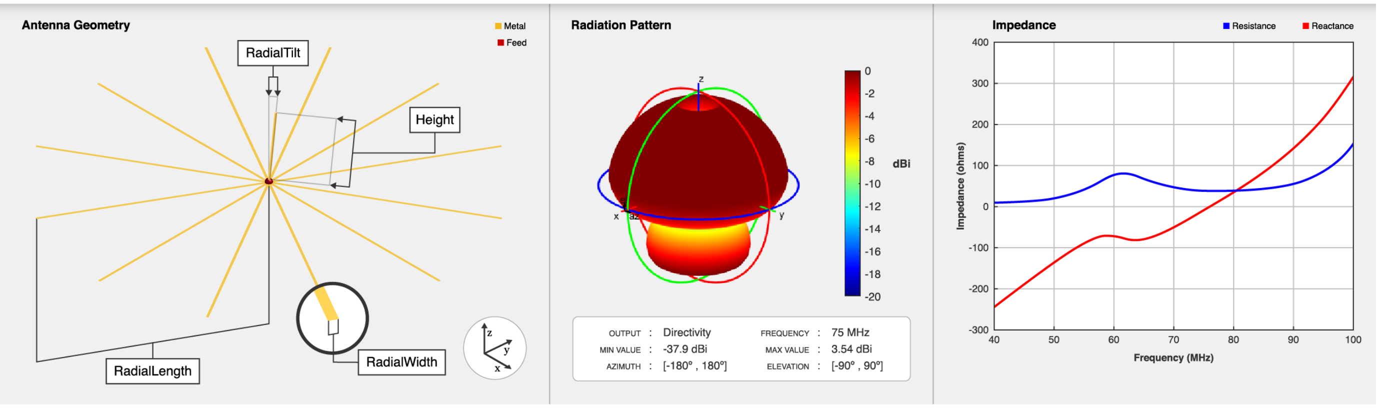
monopoleRadial

Create monopole antenna mounted on radial ground plane

Description

The `monopoleRadial` antenna object creates a monopole antenna mounted on a radial ground plane. The monopole radial antenna is a variant of the monopole antenna, where the antenna is mounted on radials as versus a rectangular ground plane. These antennas are commonly used in airborne and ground-based radio communications.

Monopole Radial



Creation

Syntax

```
mpr = monopoleRadial
mpr = monopoleRadial(Name,Value)
```

Description

`mpr = monopoleRadial` creates a quarter wavelength monopole antenna with a radial ground plane. The default antenna object is center-fed with the feed point located at the origin on the X-Y plane. The default antenna object resonates at 75 MHz.

`mpr = monopoleRadial(Name,Value)` sets “Properties” on page 1-669 using name-value pairs. For example, `monopoleRadial('Height',2.2)` creates a monopole antenna mounted on a radial ground plane with height of 2.2 meters. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Properties

`Height` — Monopole height
1 (default) | positive scalar

Monopole height, specified as a positive scalar in meter.

Example: `'Height',3`

Data Types: `double`

`Width` — Monopole width
`0.1000` (default) | positive scalar

Monopole width, specified as a positive scalar in meters.

Note Monopole width should be less than `'Height'`/4 and greater than `'Height'`/1001. For more information, see [2].

Example: `'Width',0.05`

Data Types: `double`

`NumRadials` — Number of radials
`12` (default) | positive scalar

Number of radials, specified as a positive scalar.

Example: `'NumRadials',14`

Data Types: `double`

`RadialWidth` — Width of each radial
`0.02` (default) | positive scalar

Width of each radial in the monopole radial antenna, specified as a positive scalar in meters.

Example: `'RadialWidth',0.05`

Data Types: `double`

`RadialLength` — Length of each radial
`2.85` (default) | positive scalar

Length of each radial in the monopole radial antenna, specified as a positive scalar in meters.

Example: `'RadialLength',3.13`

Data Types: `double`

`RadialTilt` — Tilt angle of radials with respect to ground plane
`0` (default) | scalar

Tilt angle of radials with respect to the ground plane, specified as a scalar in degree. Radials are tilted along the X-Y plane in negative Z direction.

Example: `'RadialTilt',10`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

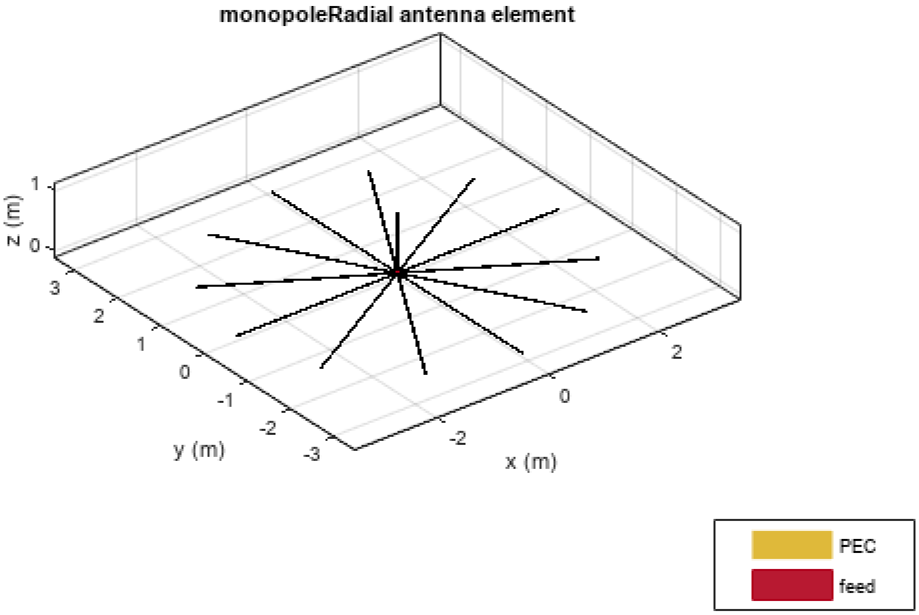
show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
current	Current distribution on antenna or array surface
charge	Charge distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
design	Design prototype antenna or arrays for resonance around specified frequency
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array

Examples

Design Monopole Radial Antenna with Default Properties

Create and view a monopole antenna mounted on radial ground plane with default properites.

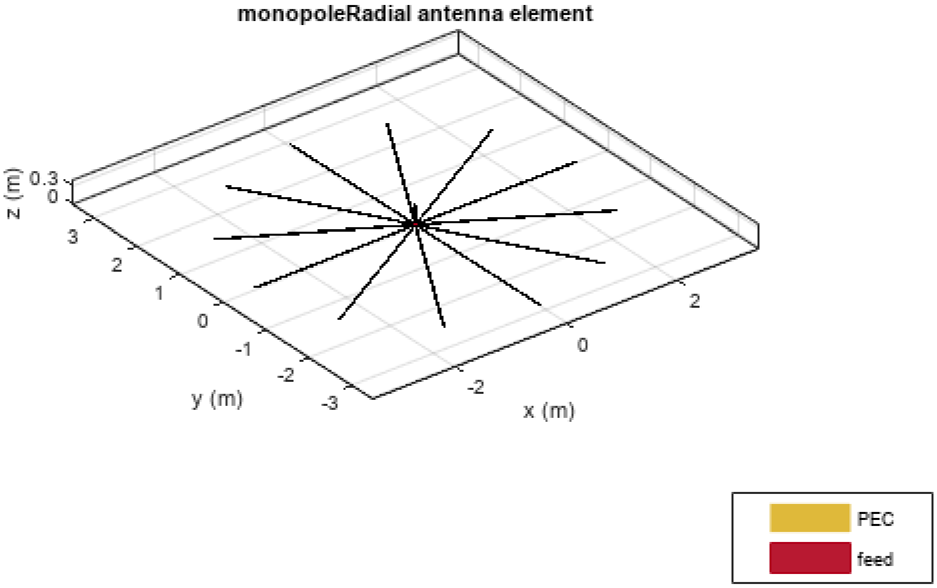
```
ant = monopoleRadial;
show(ant)
```



Plot Radiation Pattern of Monopole Radial Antenna

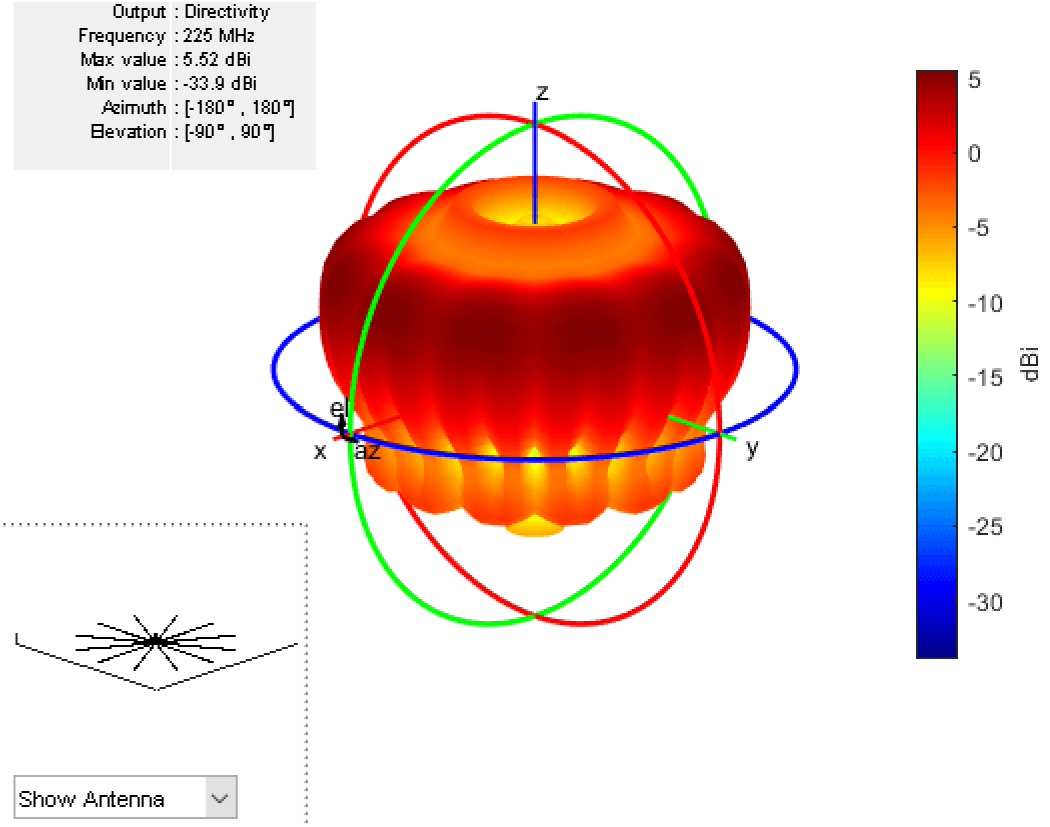
Create and view a monopole antenna on a radial ground plane with the width of 0.0067 meters and the height of 0.33331 meters.

```
m = monopoleRadial('Width',0.0067,'Height',0.3331);
show(m)
```



Plot the radiation pattern of the antenna at a frequency of 225 MHz.

```
pattern(m,225e6)
```



Output : Directivity
Frequency : 225 MHz
Max value : 5.52 dBi
Min value : -33.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2020b

References

[1] Balanis, Constantine A. *Antenna Theory: Analysis and Design*. 3rd ed. Hoboken, NJ: John Wiley, 2005.

[2] Volakis, John. *Antenna Engineering Handbook*, 4th Ed. New York: Mcgraw-Hill, 2007.

See Also

`monopoleTopHat` | `dipole` | `patchMicrostrip`

Topics

“Rotate Antennas and Arrays”

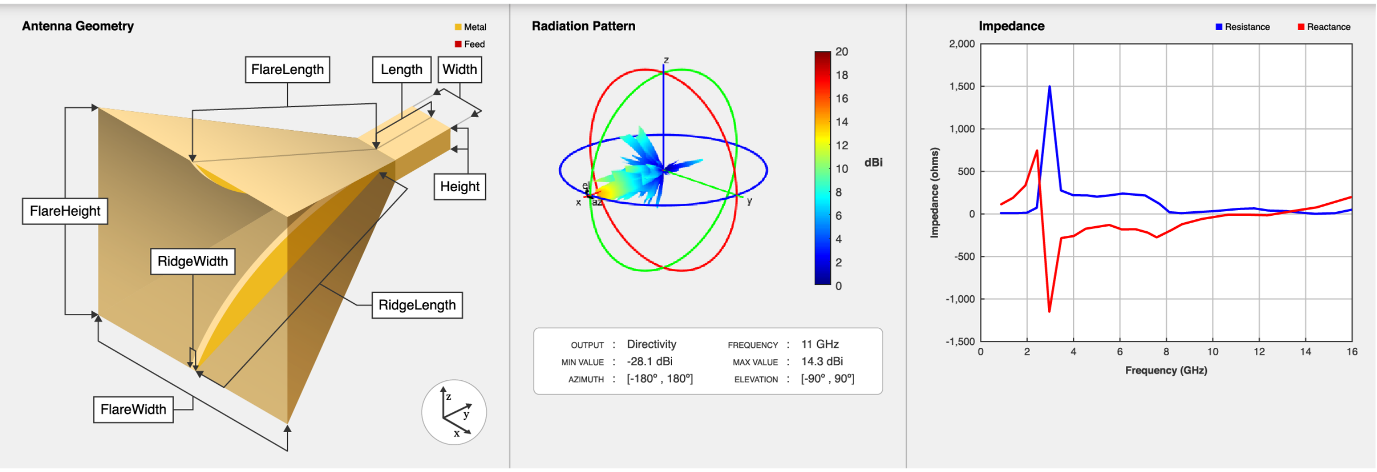
hornRidge

Create double-ridged rectangular horn

Description

The `hornRidge` object creates a double-ridged horn antenna with the default dimensions chosen for an operating frequency range of 10 GHz-12 GHz.

Horn Ridged



Ridged horn antennas are commonly used in electromagnetic interference and compatibility applications for generating electromagnetic fields. These antennas are also used in radio astronomy or radar cross-section (RCS) measurements.

Creation

Syntax

```
ant = hornRidge
ant = hornRidge(Name,Value)
```

Description

`ant = hornRidge` creates a double-ridged horn antenna with the default dimensions chosen for an operating frequency range of 10 GHz-12 GHz.

`ant = hornRidge(Name,Value)` sets properties using one or more name-value pairs. For example, `ant = hornRidge('FlareLength',178.38e-3)` creates a ridged horn antenna object with a flare length of 178.39 millimeters.

Properties

`NumFlares` — Number of flares

4 (default) | 2 | 0

Number of flares, specified as `0`, `2`, or `4`. Specify zero if you do not want any flares.

Example: `'NumFlares',2`

Example: `ant.NumFlares = 2`

Data Types: `double`

`FlareLength` — Length of flare

`0.1784` (default) | nonnegative scalar

Length of the flare, specified as a nonnegative scalar in meters.

Example: `'FlareLength',0.2760`

Example: `ant.FlareLength = 0.2760`

Data Types: `double`

`FlareWidth` — Width of flare

`0.1834` (default) | nonnegative scalar

Width of the flare, specified as a nonnegative scalar in meters.

Example: `'FlareWidth',0.3760`

Example: `ant.FlareWidth = 0.3760`

Data Types: `double`

`FlareHeight` — Height of flare

`0.1732` (default) | nonnegative scalar

Height of the flare, specified as a nonnegative scalar in meters.

Example: `'FlareHeight',0.2560`

Example: `ant.FlareHeight = 0.2560`

Data Types: `double`

`Length` — Length of waveguide

`0.0538` (default) | nonnegative scalar

Length of the waveguide, specified as a nonnegative scalar in meters.

Example: `'Length',0.0676`

Example: `ant.Length = 0.0676`

Data Types: `double`

`Width` — Width of waveguide

`0.0370` (default) | nonnegative scalar

Width of the waveguide, specified as a nonnegative scalar in meters.

Example: `'Width',0.0476`

Example: `ant.Width = 0.0476`

Data Types: `double`

`Height` — Height of waveguide
`0.0177` (default) | nonnegative scalar

Height of the waveguide, specified as a nonnegative scalar in meters.

Example: `'Height',0.0340`

Example: `ant.Height = 0.0340`

Data Types: `double`

`RidgeLength` — Length of waveguide ridge
`0.0370` (default) | nonnegative scalar

Length of the waveguide ridge, specified as a nonnegative scalar in meters.

Example: `'RidgeLength',0.0276`

Example: `ant.RidgeLength = 0.0276`

Data Types: `double`

`RidgeWidth` — Width of waveguide ridge
`0.0050` (default) | nonnegative scalar

Width of the waveguide ridge, specified as a nonnegative scalar in meters.

Example: `'RidgeWidth',0.0040`

Example: `ant.RidgeWidth = 0.0040`

`RidgeGap` — Gap between two ridges of waveguide
`0.0070` (default) | nonnegative scalar

Gap between the two ridges of the waveguide, specified as a nonnegative scalar in meters.

Example: `'RidgeGap',0.0060`

Example: `ant.RidgeGap = 0.0060`

Data Types: `double`

`FeedHoleRadius` — Radius of feeding hole
`0.0005` (default) | nonnegative scalar

Radius of the feeding hole, specified as a nonnegative scalar in meters.

Example: `'FeedHoleRadius',0.0008`

Example: `ant.FeedHoleRadius = 0.0008`

Data Types: `double`

`FeedWidth` — Width of feed
`0.0001` (default) | nonnegative scalar

Width of the feed, specified as a nonnegative scalar in meters.

Example: `'FeedWidth',0.0002`

Example: `ant.FeedWidth = 0.0002`

Data Types: `double`

`FeedOffset` — Signed distance from closed end of waveguide
`[-0.0076 0]` (default) | two-element vector

Signed distance from the closed end of the waveguide, specified as a two-element vector with each element unit in meters.

Example: `'FeedOffset',[-0.00626 0]`

Example: `ant.FeedOffset = [-0.00626,0]`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

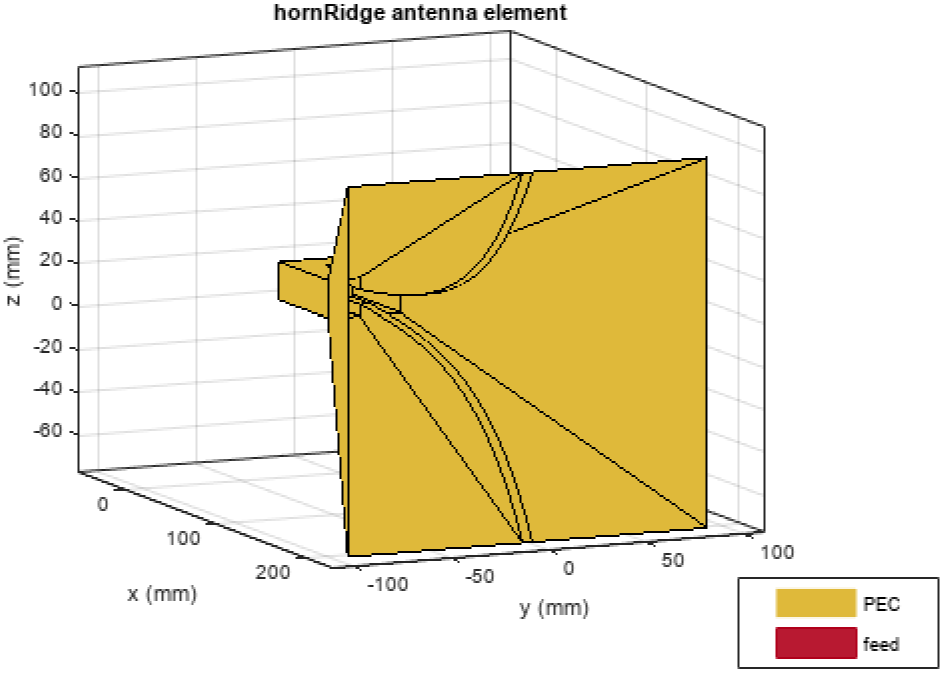
Examples

Default Ridged Horn Antenna and Radiation Pattern

Create and view a default double ridged horn antenna.

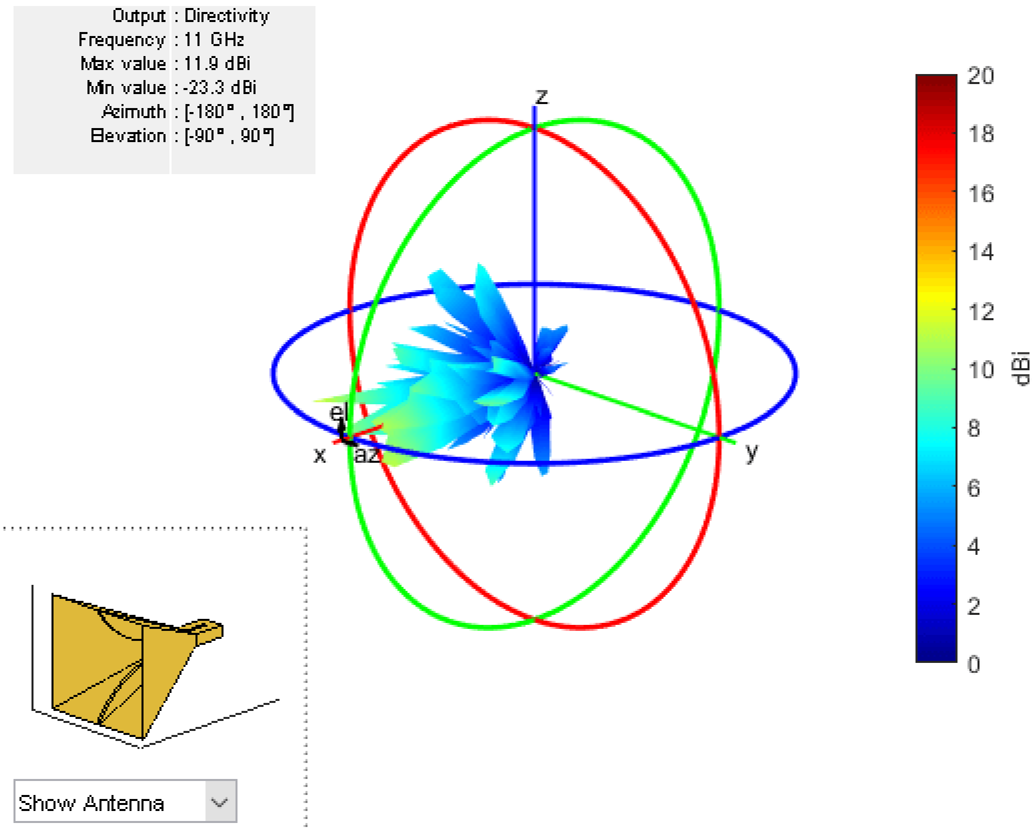
```
ant = hornRidge
ant = 
  hornRidge with properties:

         NumFlares: 4
       FlareLength: 0.1784
        FlareWidth: 0.1834
       FlareHeight: 0.1732
            Length: 0.0538
             Width: 0.0370
            Height: 0.0177
       RidgeLength: 0.0370
        RidgeWidth: 0.0050
          RidgeGap: 0.0070
    FeedHoleRadius: 5.0000e-04
         FeedWidth: 1.0000e-04
        FeedOffset: [-0.0076 0]
         Conductor: [1x1 metal]
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 11 GHz.

```
p = PatternPlotOptions('MagnitudeScale',[0 20]);
figure;
pattern(ant,11e9,'patternOptions',p);
```



```
   Output : Directivity
Frequency : 11 GHz
Max value : 11.9 dBi
Min value : -23.3 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Radiation Pattern of Ridged Horn Antenna with Two Flares

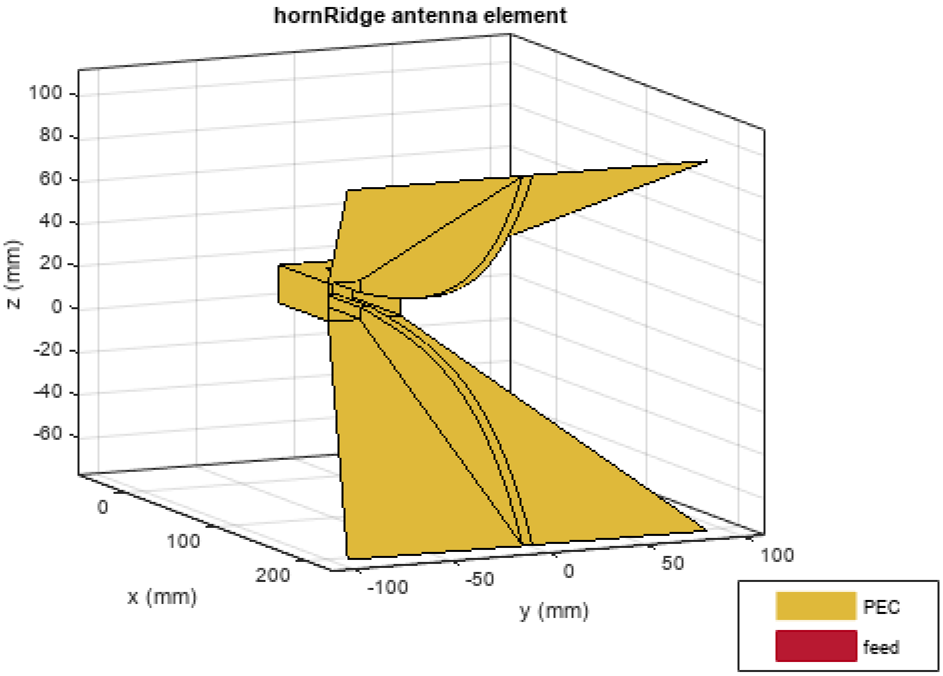
Create and view a ridged horn antenna with 2 flares.

```
ant = hornRidge('NumFlares',2)
```

```
ant = 
  hornRidge with properties:

         NumFlares: 2
       FlareLength: 0.1784
        FlareWidth: 0.1834
       FlareHeight: 0.1732
            Length: 0.0538
             Width: 0.0370
            Height: 0.0177
       RidgeLength: 0.0370
        RidgeWidth: 0.0050
          RidgeGap: 0.0070
    FeedHoleRadius: 5.0000e-04
         FeedWidth: 1.0000e-04
```

```
          FeedOffset: [-0.0076 0]
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1x1 lumpedElement]
```

```
show(ant)
```

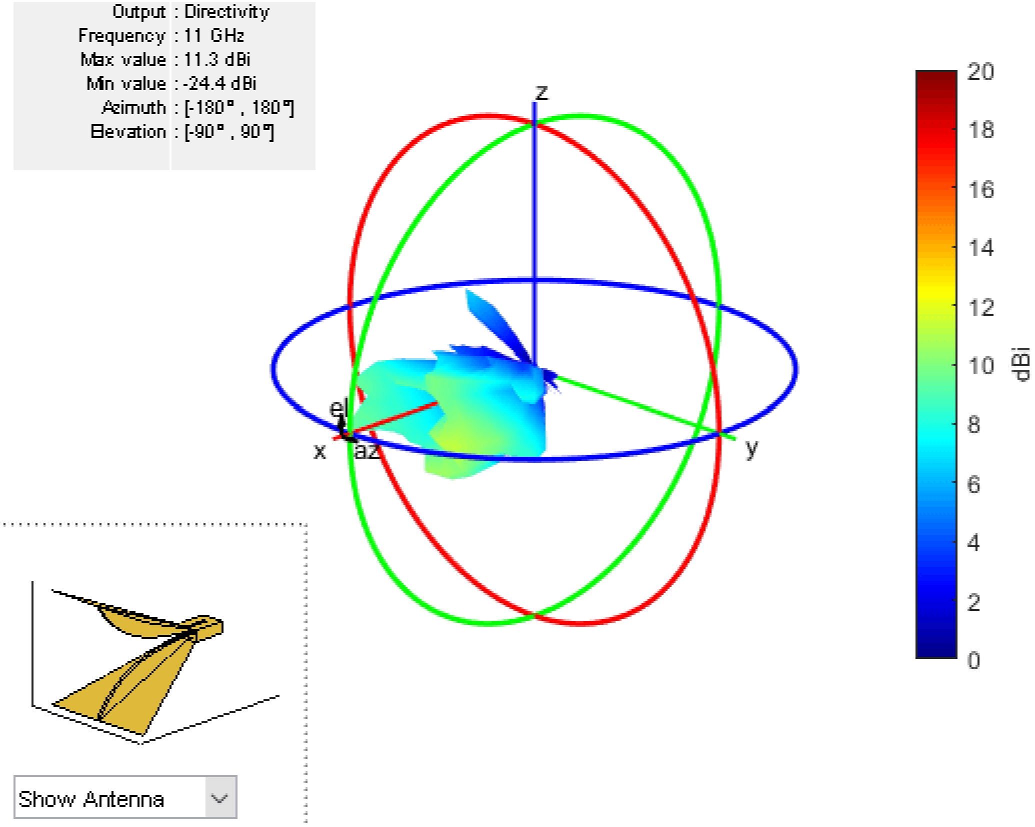


Plot the radiation pattern of the antenna at 11 GHz.

```
p = PatternPlotOptions('MagnitudeScale',[0 20])
```

```
p = 
  PatternPlotOptions with properties:

      Transparency: 1
         SizeRatio: 0.9000
    MagnitudeScale: [0 20]
     AntennaOffset: [0 0 0]
```

```
figure;
pattern(ant,11e9,'patternOptions',p)
```



Output : Directivity
Frequency : 11 GHz
Max value : 11.3 dBi
Min value : -24.4 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2020b

See Also

Topics

"Rotate Antennas and Arrays"

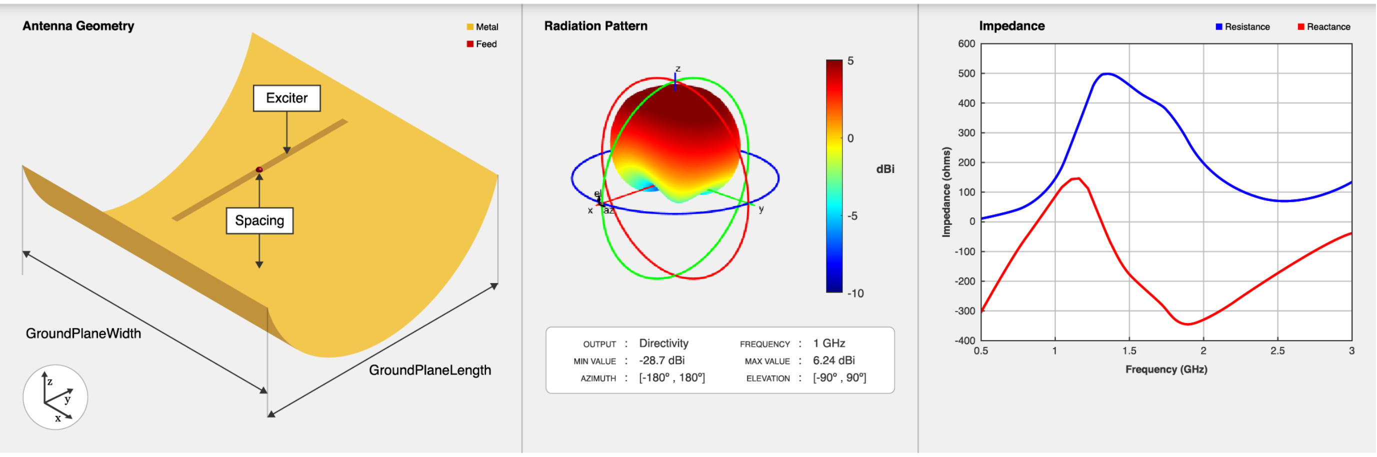
reflectorCylindrical

Create cylindrical reflector-backed antenna

Description

The `reflectorCylindrical` antenna object creates a cylindrical reflector-backed antenna. The cylindrical shape of the reflector allows you to focus the signal to the antenna surface. Cylindrical reflectors are widely used as high-gain apertures fed with line sources and in airborne navigational antennas where sharp azimuthal beams and wide-angle vertical coverage is required.

Reflector Cylindrical



Creation

Syntax

```
ant = reflectorCylindrical
ant = reflectorCylindrical(Name,Value)
```

Description

`ant = reflectorCylindrical` creates a cylindrical reflector-backed antenna. The default cylindrical reflector antenna object has an exciter as a center-fed dipole located on the X-Y plane and the dimensions are chosen for an operating frequency of around 1 GHz.

`ant = reflectorCylindrical(Name,Value)` sets “Properties” on page 1-686 using name-value pairs. For example, `reflectorCylindrical('GroundPlaneWidth',0.21)` creates a cylindrical reflector with a width of 0.21 meters. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Properties

`Exciter` — Antenna or array used as exciter

`dipole` (default) | `antenna` object | `array` object

Antenna used as an exciter, specified as an `antenna` or an array object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole('Length',0.1409,'Width',0.02,'FeedOffset',0,'Tilt',90,'TiltAxis',[0 1 0])`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`GroundPlaneLength` — Reflector length

`0.2` (default) | positive scalar

Reflector length along X-axis, specified as a positive scalar in meters.

Example: `'GroundPlaneLength',0.6`

Example: `ant.GroundPlaneLength = 0.18`

Data Types: `double`

`GroundPlaneWidth` — Reflector width

`0.2` (default) | positive scalar

Reflector width along Y-axis, specified as a positive scalar in meters.

Example: `'GroundPlaneWidth',0.4`

Example: `ant.GroundPlaneWidth = 0.18`

Data Types: `double`

`Spacing` — Distance between reflector and exciter

`0.075` (default) | scalar

Distance between reflector and exciter, specified as a scalar in meters.

Example: `'Spacing',0.059`

Example: `ant.Spacing = 0.195`

Data Types: `double`

`Depth` — Perpendicular distance between ground plane and reflector aperture

`0.075` (default) | positive scalar

Perpendicular distance between the ground plane and the aperture of the cylindrical reflector, specified as a positive scalar in meters.

Example: `'Depth',0.09`

Example: `ant.Depth = 0.049`

Note `Depth` should be less than or equal to half of the “GroundPlaneLength” on page 1-0 .

Data Types: `double`

`Enable Probe Feed` — Flag to enable probe feed

`0` (default) | `1`

Flag to enable the probe feed, specified as `0` or `1` . Setting the flag to `0` disables the probe feed, and setting the flag to `1` enables it.

Example: `'EnableProbeFeed',1`

Example: `ant.EnableProbeFeed = 1`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelements`, where `lumpedelements` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
current	Current distribution on antenna or array surface
charge	Charge distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
design	Design prototype antenna or arrays for resonance around specified frequency
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array

Examples

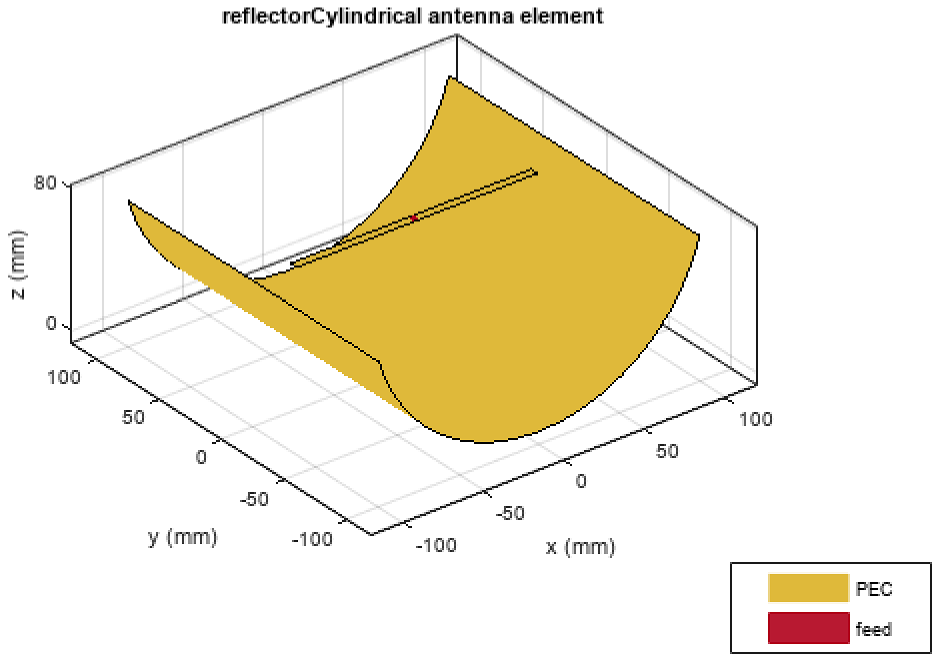
Design Cylindrical Reflector with Default Properties

Create a cylindrical reflector antenna object with default properties.

```
ant = reflectorCylindrical;
```

View the antenna object.

```
show(ant)
```



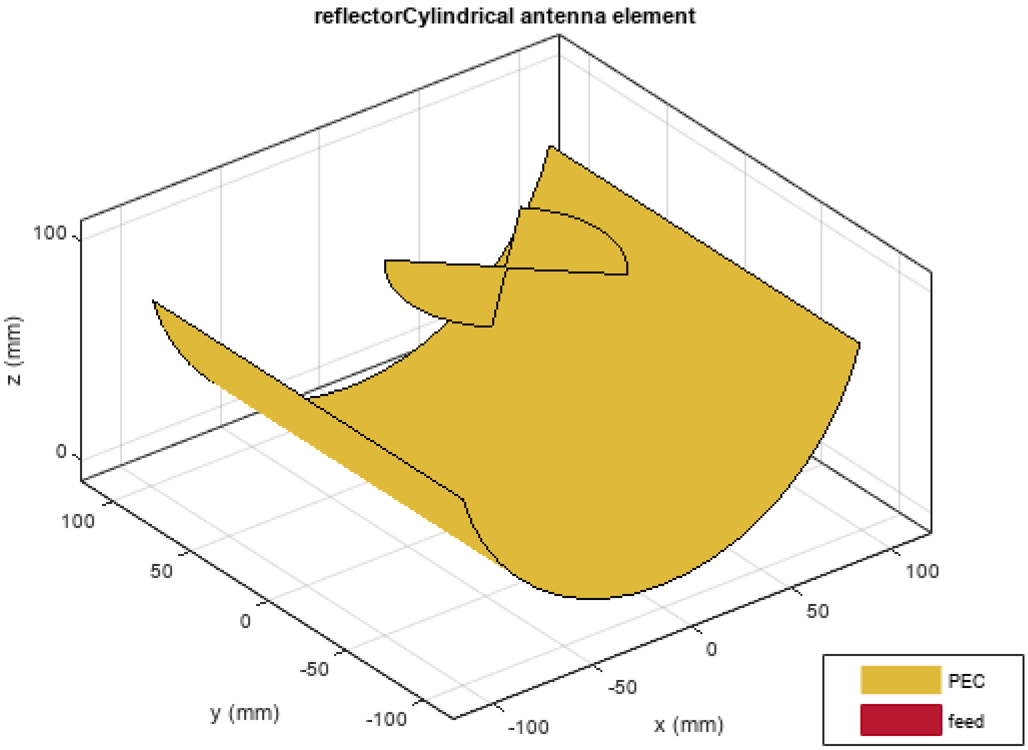
Use Rounded Bow-Tie Dipole Antenna as Exciter

Create a `reflectorCylindrical` antenna object with a rounded bow-tie dipole antenna as an exciter.

```
b = bowtieRounded('Length',96e-3,'Tilt',90,'TiltAxis',[0 1 0]);
r = reflectorCylindrical('Exciter',b,'Spacing',100e-3);
```

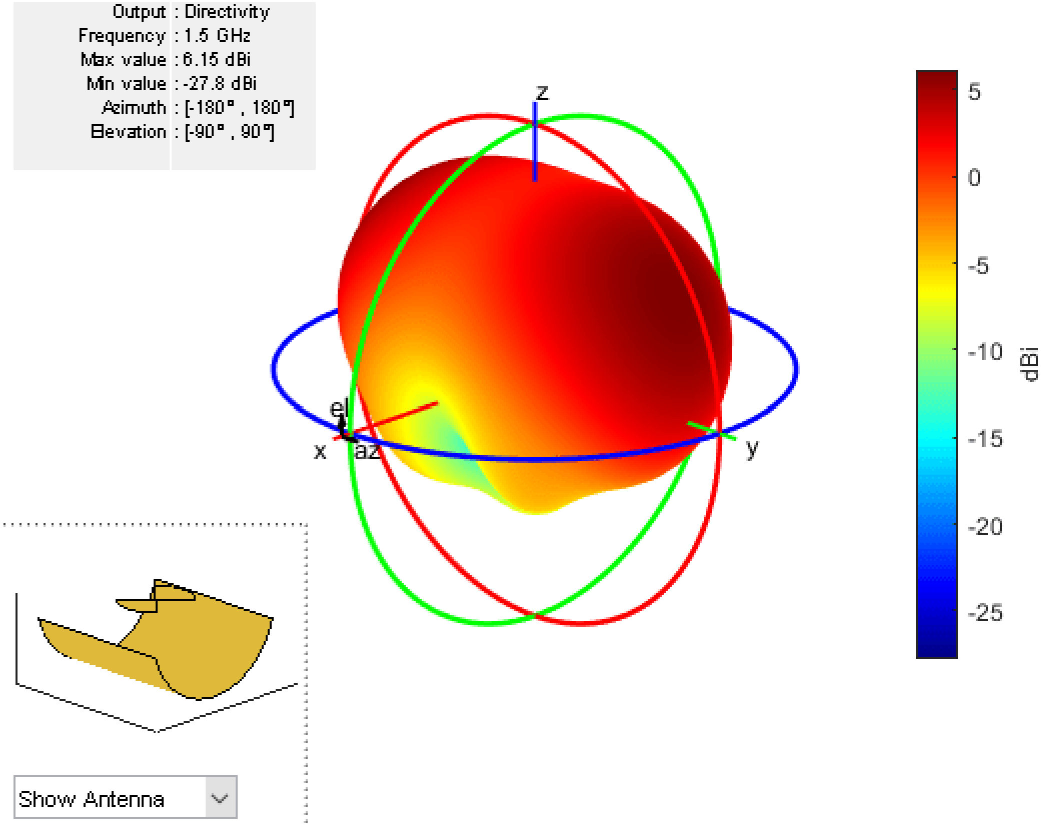
View the antenna object.

```
figure
show(r)
```



Plot the radiation pattern at 1.5 GHz.

```
figure
pattern(r,1.5e9)
```

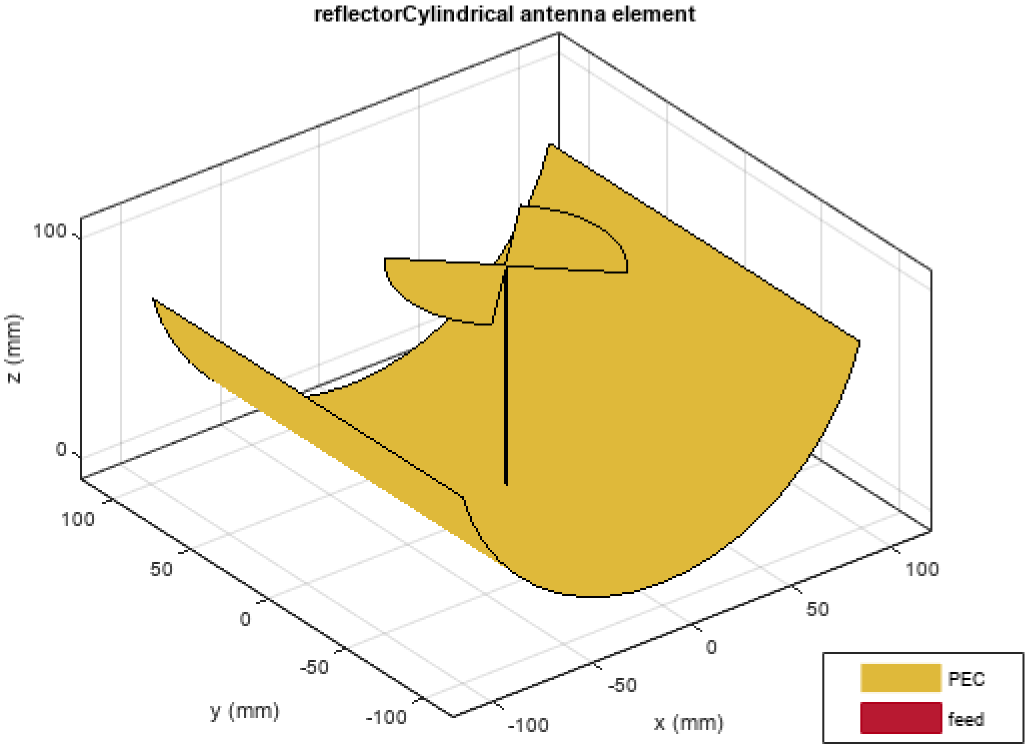


Enable the probe feed for the `reflectorCylindrical` antenna object.

```
re = reflectorCylindrical('Exciter',b,'Spacing',100e-3,'EnableProbeFeed',1);
```

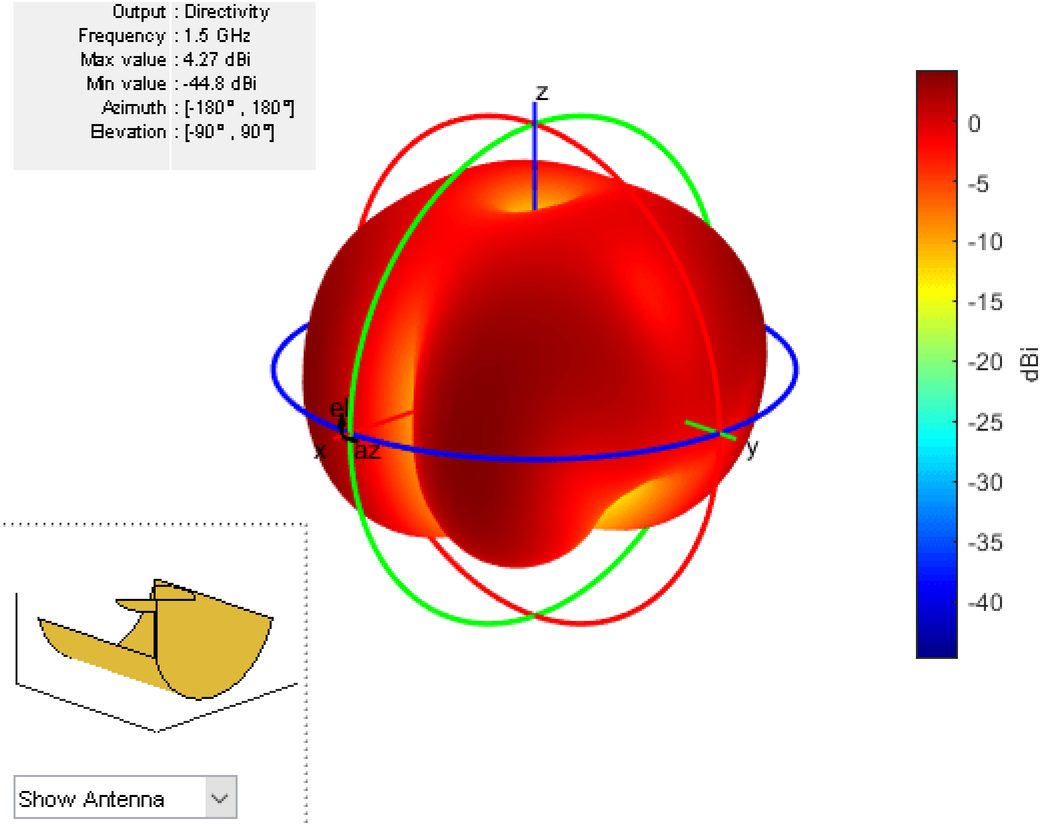
View the antenna object with the probe feed enabled.

```
figure
show(re)
```



Plot the radiation pattern of the antenna object at 1.5 GHz with the probe feed enabled.

```
figure
pattern(re,1.5e9)
```



Output : Directivity
Frequency : 1.5 GHz
Max value : 4.27 dBi
Min value : -44.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Linear Array of Crossed Dipole Backed With Cylindrical Reflector

Create a linear array of crossed dipole antenna.

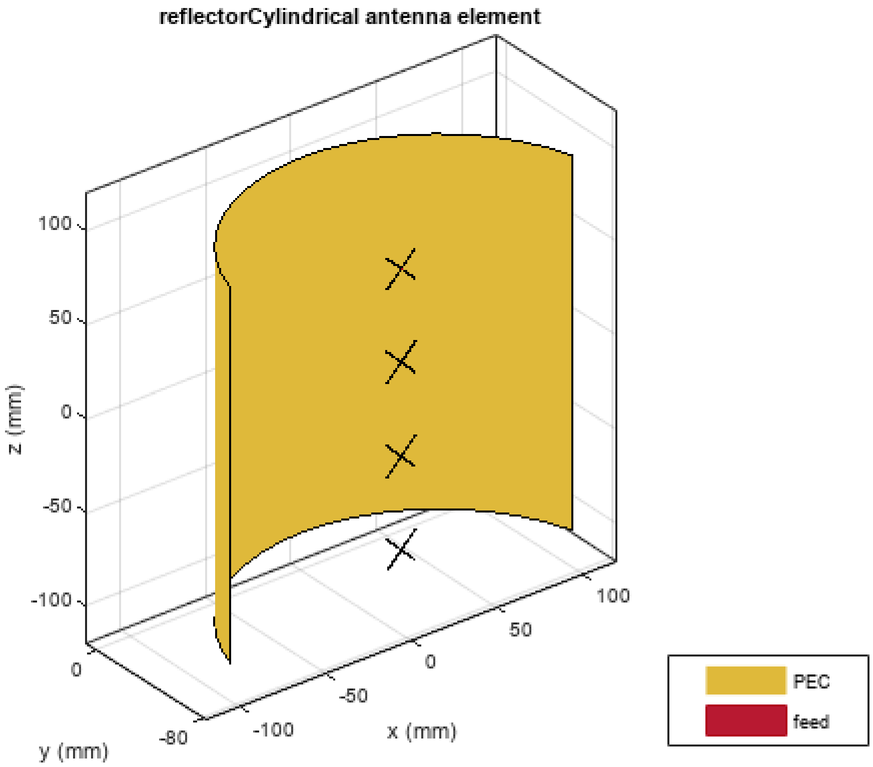
```
d = dipoleCrossed('Tilt',90,'TiltAxis',[0 1 0]);
la = linearArray('Element',d,'NumElements',4,'ElementSpacing',0.05,'Tilt',90,'TiltAxis',[0 0 1])
```

Create a cylindrical reflector backed array.

```
ant = reflectorCylindrical('Exciter',la,'Tilt',90)
```

```
ant = 
  reflectorCylindrical with properties:

              Exciter: [1x1 linearArray]
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.2000
              Spacing: 0.0750
                Depth: 0.0750
      EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 90
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



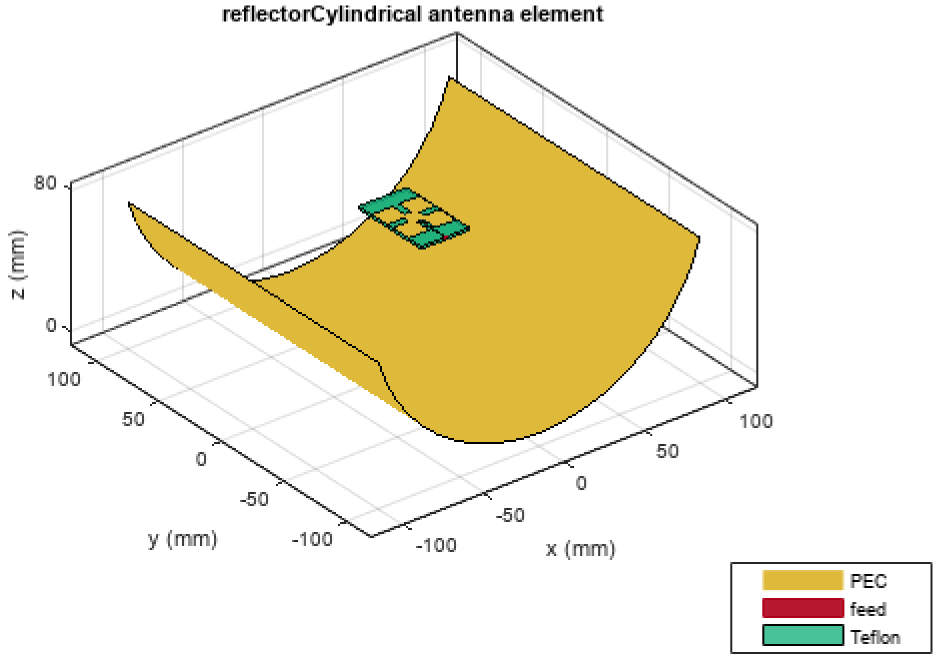
Create Minkowski's Loop Fractal Antenna with Cylindrical Reflector Backing Structure

Create a cylindrical reflector-backed Minkowski's loop fractal antenna.

```
e = fractalIsland('Substrate',dielectric('Teflon'),'Tilt',90,'TiltAxis',[0 0 1]);
ant = reflectorCylindrical('Exciter',e)
```

```
ant = 
  reflectorCylindrical with properties:

              Exciter: [1x1 fractalIsland]
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.2000
              Spacing: 0.0750
                Depth: 0.0750
      EnableProbeFeed: 0
            Conductor: [1x1 metal]
                 Tilt: 0
            TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```



Version History

Introduced in R2020b

References

[1] Balanis, Constantine A. *Antenna Theory: Analysis and Design*. 3rd ed. Hoboken, NJ: John Wiley, 2005.

See Also

`reflector` | `reflectorParabolic` | `reflectorCorner` | `reflectorCircular` | `reflectorGrid` | `reflectorSpherical`

Topics

"Design and Analyze Curved Reflectors"
"Rotate Antennas and Arrays"

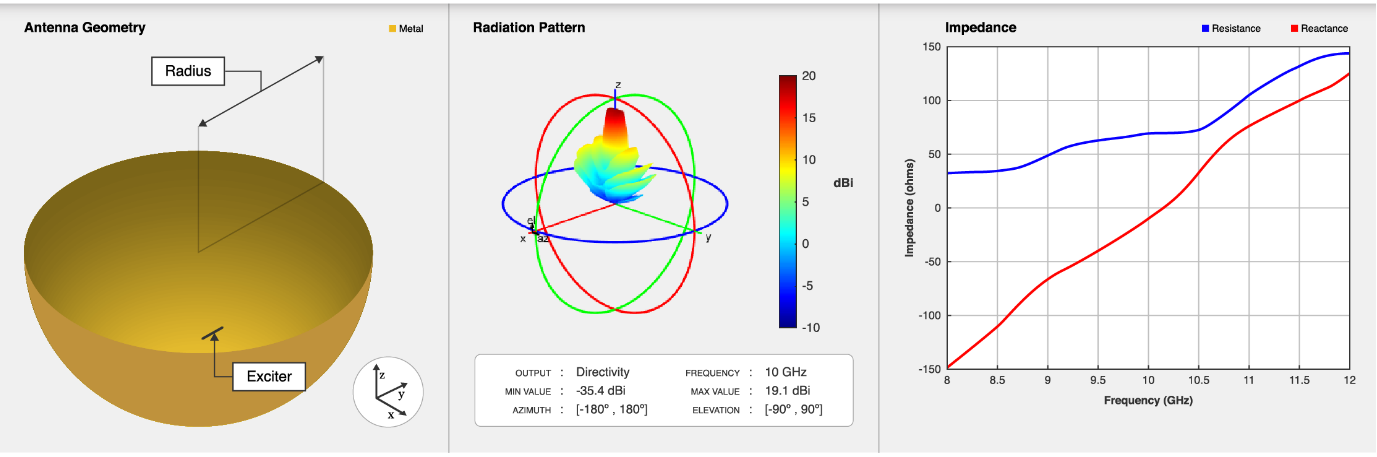
reflectorSpherical

Create spherical reflector-backed antenna

Description

The `reflectorSpherical` antenna object creates a spherical reflector-backed antenna. The reflector in the spherical reflector-backed antenna is one-half the size of the sphere. The antenna is used in wide-angle scanning on account of its perfectly symmetrical geometric configuration.

Reflector Spherical



Creation

Syntax

```
ant = reflectorSpherical
ant = reflectorSpherical(Name,Value)
```

Description

`ant = reflectorSpherical` creates a spherical reflector-backed antenna. The default antenna object has an exciter as a center-fed dipole located on the X-Y plane. The default antenna object dimensions are chosen for an operating frequency of 1 GHz.

`ant = reflectorSpherical(Name,Value)` sets “Properties” on page 1-696 using name-value pairs. For example, `reflectorSpherical('Radius',0.6)` sets the spherical reflector radius to 0.6 meters. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Properties

`Exciter` — Antenna or array used as exciter
`dipole` (default) | `antenna` object | `array` object

Antenna used as an exciter, specified as an `antenna` object or an `array` object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole('Length',0.1409,'Width',0.02,'FeedOffset',0,'Tilt',90,'TiltAxis',[0 1 0])`

Example: `ant.Exciter = linearArray('patchMicrostrip')`

`Radius` — Radius of spherical aperture

`0.15` (default) | positive scalar

Radius of the spherical reflector aperture along X and Y-axes, specified as a positive scalar in meters.

Example: `'Radius',0.259`

Example: `ant.Radius = 0.195`

Data Types: `double`

`FeedOffset` — Signed distance between feed point and origin

`[0 0 0.075]` (default) | three-element vector

Signed distance between feed point of the exciter and the origin, specified as a three-element vector with each element unit in meters.

Example: `'FeedOffset',[0 0 0.082]`

Example: `ant.FeedOffset = [0 0 0.082]`

Data Types: `double`

`Depth` — Perpendicular distance between origin and aperture of antenna

`0.15` (default) | positive scalar

Perpendicular distance between origin and the aperture of the spherical reflector-backed antenna, specified as a positive scalar in meters.

Example: `'Depth',0.6`

Example: `ant.Depth = 0.18`

Note `Depth` should be less than or half the “Radius” on page 1-0 .

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis = 'Z'`

Note The `wireStack` antenna object only accepts the dot method to change its properties.

Data Types: `double`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load'`,`lumpedelements`, where `lumpedelements` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`SolverType` — Solver for antenna analysis

`'MoM-PO'` (default) | `'MoM'` | `'FMM'`

Solver for antenna analysis, specified as the comma-separated pair consisting of `'SolverType'` and `'MoM-PO'` or `'MoM'` (Method of Moments) or `'FMM'` (Fast Multipole Method).

Example: `'SolverType','MOM'`

Data Types: `char`

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
current	Current distribution on antenna or array surface
charge	Charge distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
design	Design prototype antenna or arrays for resonance around specified frequency
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array

Examples

Design Spherical Reflector-Backed Antenna with Default Properties

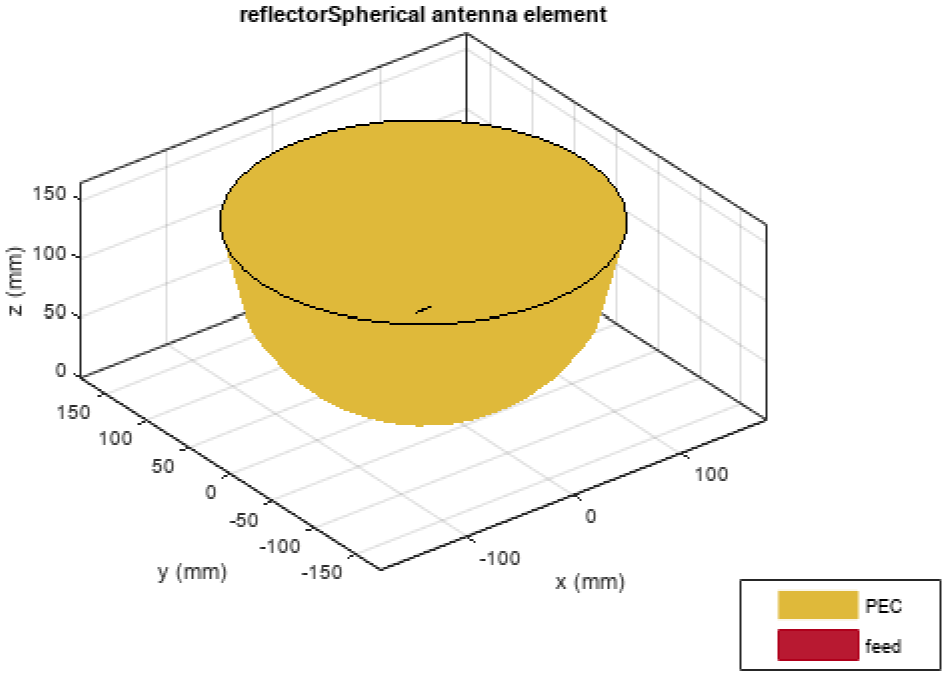
Create a spherical reflector-backed antenna object with default properties.

```
ant = reflectorSpherical
```

```
ant = 
  reflectorSpherical with properties:

       Exciter: [1x1 dipole]
        Radius: 0.1500
         Depth: 0.1500
    FeedOffset: [0 0 0.0750]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
    SolverType: 'MoM-PO'
```

View the antenna.

```
show(ant)
```



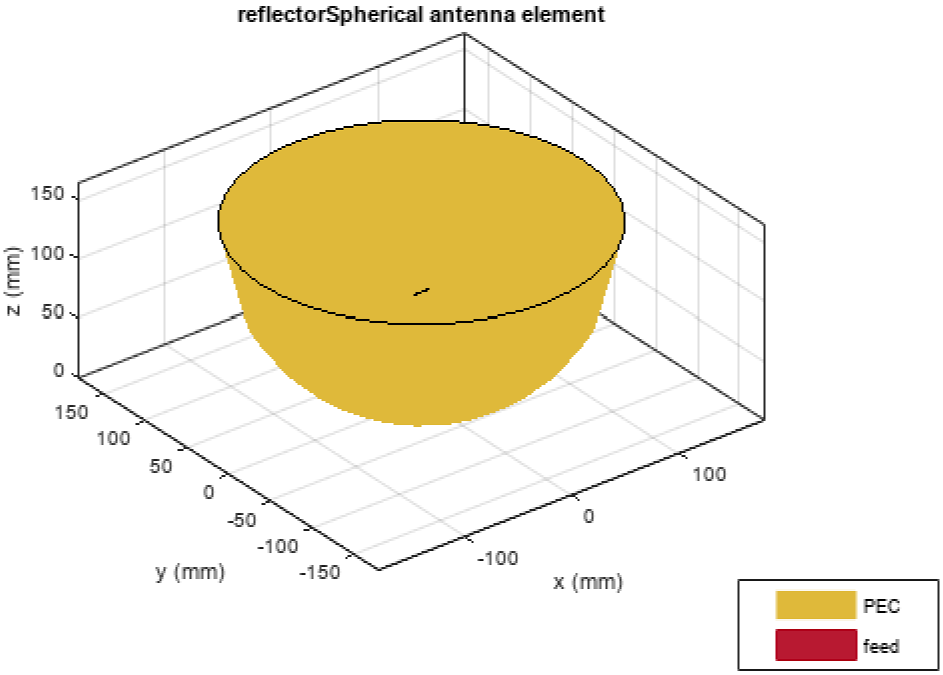
Plot S-Parameter of Spherical Reflector-Backed Dipole Antenna

Create a spherical reflector-backed antenna with a dipole as an exciter spaced at 90 millimeters.

```
rs = reflectorSpherical;
rs.FeedOffset(3) = 90e-3;
```

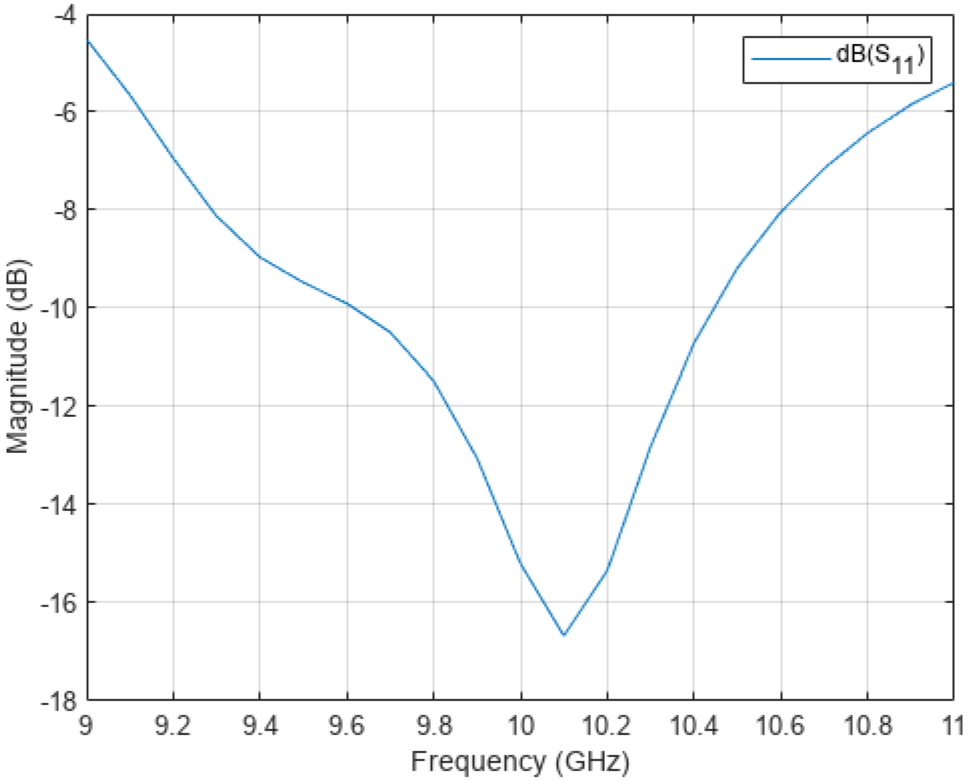
Visualize the antenna.

```
figure
show(rs)
```



Plot the S-parameters at 1 GHz.

```
s = sparameters(rs,(9:0.1:11)*1e9);
figure
rfplot(s)
```



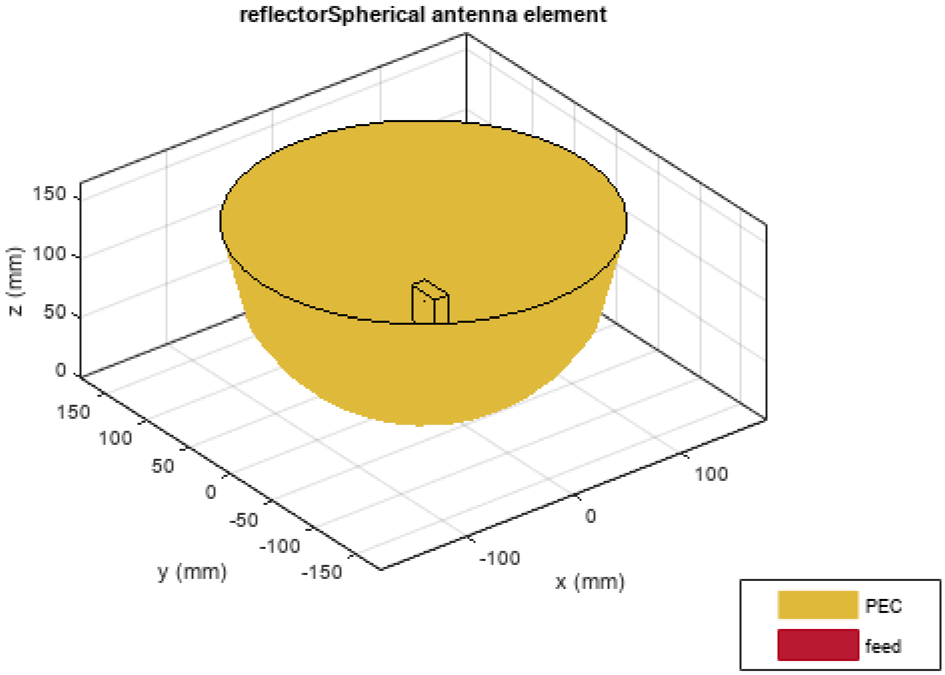
Plot Radiation Pattern of Spherical Reflector-Backed Waveguide Antenna

Create a waveguide designed at 10 GHz backed with a spherical reflector.

```
w = design(waveguide,10e9);
rs = reflectorSpherical('Exciter',w);
rs.Exciter.Tilt = 90;
rs.Exciter.TiltAxis = [ 0 1 0];
```

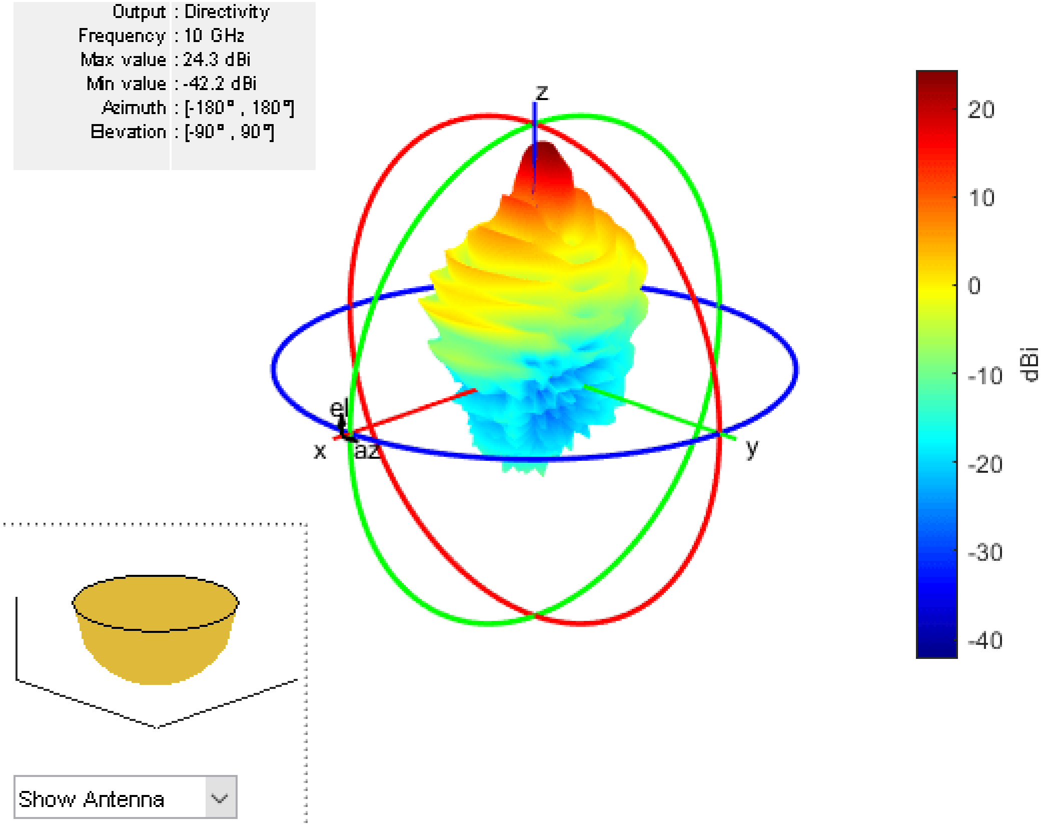
Visualize the antenna.

```
figure
show(rs)
```



Plot the radiation pattern at 10 GHz.

```
figure
pattern(rs,10e9)
```



Output : Directivity
Frequency : 10 GHz
Max value : 24.3 dBi
Min value : -42.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Spherical Reflector Antenna with Circular Array of Discone Antennas

Create a circular array with discone antennas.

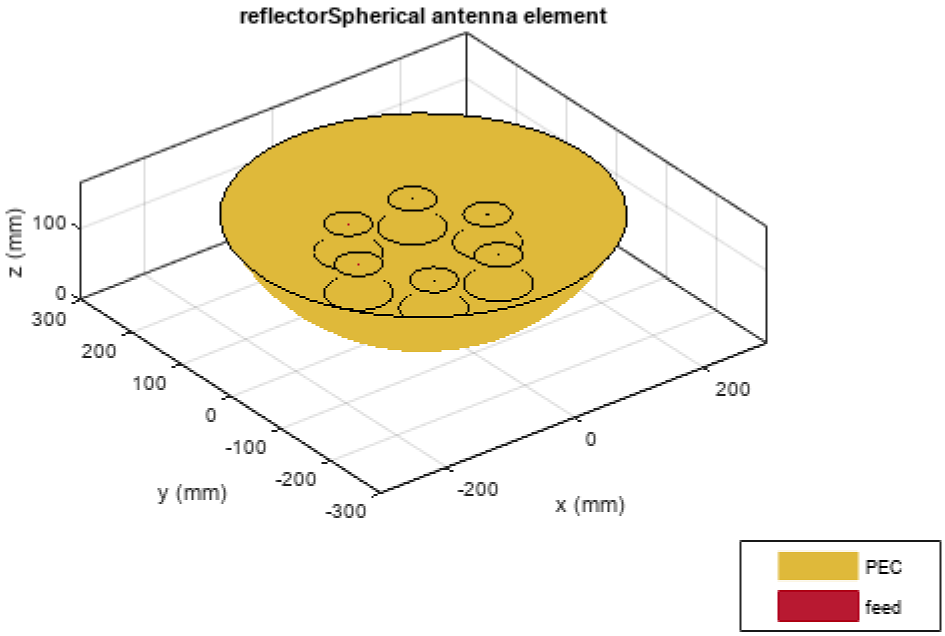
```
d = discone('Height',0.04);
circArr = circularArray('Element',d,'Radius',0.1);
```

Create a spherical reflector antenna with circular array exciter.

```
ant = reflectorSpherical('Exciter',circArr,'Radius',0.25)
```

```
ant = 
  reflectorSpherical with properties:

        Exciter: [1x1 circularArray]
         Radius: 0.2500
          Depth: 0.1500
     FeedOffset: [0 0 0.0750]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
     SolverType: 'MoM-PO'
```

```
show(ant)
```



Version History

Introduced in R2020b

References

[1] Balanis, Constantine A. *Antenna Theory: Analysis and Design*. 3rd ed. Hoboken, NJ: John Wiley, 2005.

See Also

`reflector` | `reflectorParabolic` | `reflectorCorner` | `reflectorCircular` | `reflectorCylindrical` | `reflectorGrid`

Topics

"Design and Analyze Curved Reflectors"
"Rotate Antennas and Arrays"

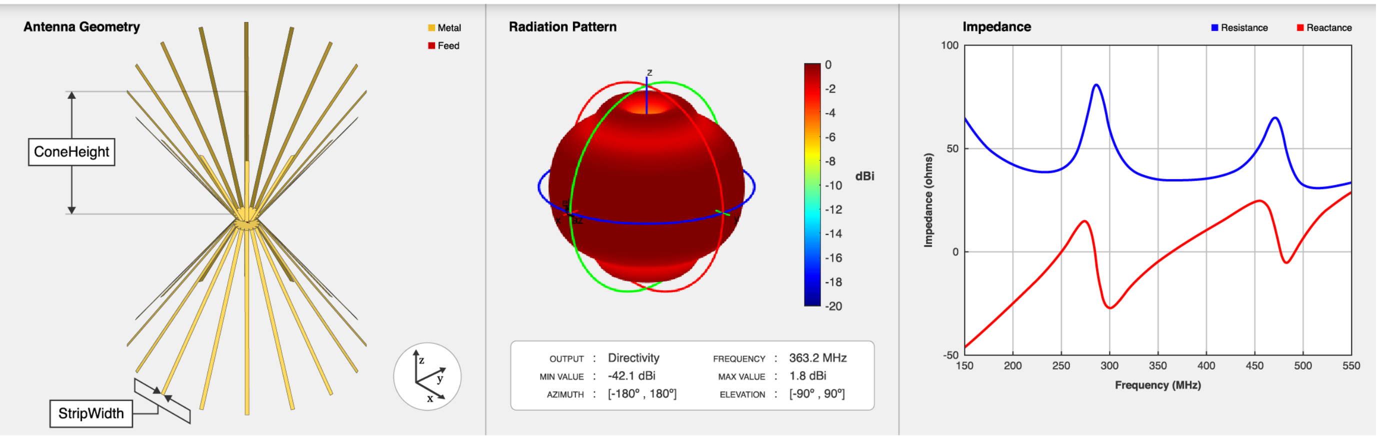
biconeStrip

Create strip bicone antenna

Description

The `biconeStrip` object creates a strip bicone antenna. The strip bicone antenna is an approximation of a solid bicone antenna, where strips are used to approximate the two cones. The strip configuration makes these antennas lightweight and reduces wind loading. These antennas are more suitable for use at low frequencies. Strip bicone antennas are popular for their wide-impedance bandwidth and omnidirectional radiation coverage. These antennas are used in applications like emission testing, field monitoring, and chamber characterization.

Bicone Strip



OUTPUT : Directivity
MIN VALUE : -42.1 dBi
AZIMUTH : [-180° , 180°]
FREQUENCY : 363.2 MHz
MAX VALUE : 1.8 dBi
ELEVATION : [-90° , 90°]

There are two types of bicone strip antennas, open-ended and phantom biconical. Specify the `“HatHeight” on page 1-0` property to create a phantom strip bicone antenna.

Creation

Syntax

```
ant = biconeStrip
ant = biconeStrip(Name,Value)
```

Description

`ant = biconeStrip` creates a strip bicone antenna with dimensions for a resonant frequency of 363.2 MHz.

`ant = biconeStrip(Name,Value)` sets `“Properties” on page 1-707` using one or more name-value pairs. For example, `ant = biconeStrip('NumStrips', 8)` creates a strip bicone antenna with eight strips.

Properties

`NumStrips` — Number of strips to form cones

`16` (default) | scalar in the range `[6,64]`

Number of strips to form the two cones of strip bicone antenna, specified a scalar integer in the range `[6,64]`.

Example: `'NumStrips',8`

Example: `ant.NumStrips = 8`

Data Types: `double`

`StripWidth` — Width of strip

`18e-3` (default) | positive scalar

Width of each strip, specified as positive scalar in meters.

Example: `'StripWidth',0.02`

Example: `ant.StripWidth = 0.02`

Data Types: `double`

`HatHeight` — Vertical height of hats

`0` (default) | scalar | two-element vector

Vertical height of the two hats, specified as either of the following:

- `0`— This creates open-ended strip bicone antenna.
- Positive scalar in meters— This creates two cone hats of same height.
- Two-element vector with each element unit in meters— This creates two cone hats of different heights. In the two-element vector, the first element specifies the hat height of the top cone, and the second element specifies the hat height of the bottom cone.

Example: `'HatHeight',0.045`

Example: `ant.HatHeight = 0.045`

Data Types: `double`

`ConeHeight` — Vertical height of cones

`665e-3` (default) | scalar | two-element vector

Vertical height of the two cones, specified as either of the following:

- Positive scalar in meters: This creates two cones of same height.
- Two-element vector with each element unit in meters: This creates two cones of different heights. In the two-element vector, the first element specifies the height of the top cone, and the second element specifies the height of the bottom cone.

Example: `'ConeHeight',0.5`

Example: `ant.ConeHeight = 0.5`

Data Types: `double`

NarrowRadius — Radius of apex
70e-3 (default) | scalar | two-element vector

Radius at the apex of the cones, specified as either of the following:

- Positive scalar in meters: This creates two cones with the same narrow radius.
- Two-element vector with each element unit in meters: This creates two cones with different narrow radii. In the two-element vector, the first element specifies the narrow radius of the top cone, and the second element specifies the narrow radius of the bottom cone.

Example: `'NarrowRadius',0.04`

Example: `ant.NarrowRadius = 0.04`

Data Types: `double`

BroadRadius — Radius at broad opening of cone
647e-3 (default) | scalar | two-element vector

Radius at the broad opening of the cones, specified as either of the following:

- Positive scalar in meters: This creates two cones with the same broad radius.
- Two-element vector with each element unit in meters: This creates two cones with different broad radii. In the two-element vector, the first element specifies the broad radius of the top cone, and the second element specifies the broad radius of the bottom cone.

Example: `'BroadRadius',0.7`

Example: `ant.BroadRadius = 0.7`

Data Types: `double`

FeedHeight — Height of feed
45e-3 (default) | positive scalar

Height of the feed spanning the gap between the two cones, specified as positive scalar in meters.

Example: `'FeedHeight',0.04`

Example: `ant.FeedHeight = 0.04`

Data Types: `double`

FeedWidth — Width of feed
40e-3 (default) | positive scalar

Width of the feed of the antenna, specified as a positive scalar in meters.

Example: `'FeedWidth',0.03`

Example: `ant.FeedWidth = 0.03`

Data Types: `double`

Conductor — Type of metal material
'PEC' (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'x'` | `'y'` | `'z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the *x*-, *y*-, and *z*-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, '*x*', '*y*', or '*z*'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`. `lumpedElement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

coneangle2size	Calculates equivalent cone height, broad radius, and narrow radius for cone
show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface

current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Open-ended Strip Bicone Antenna

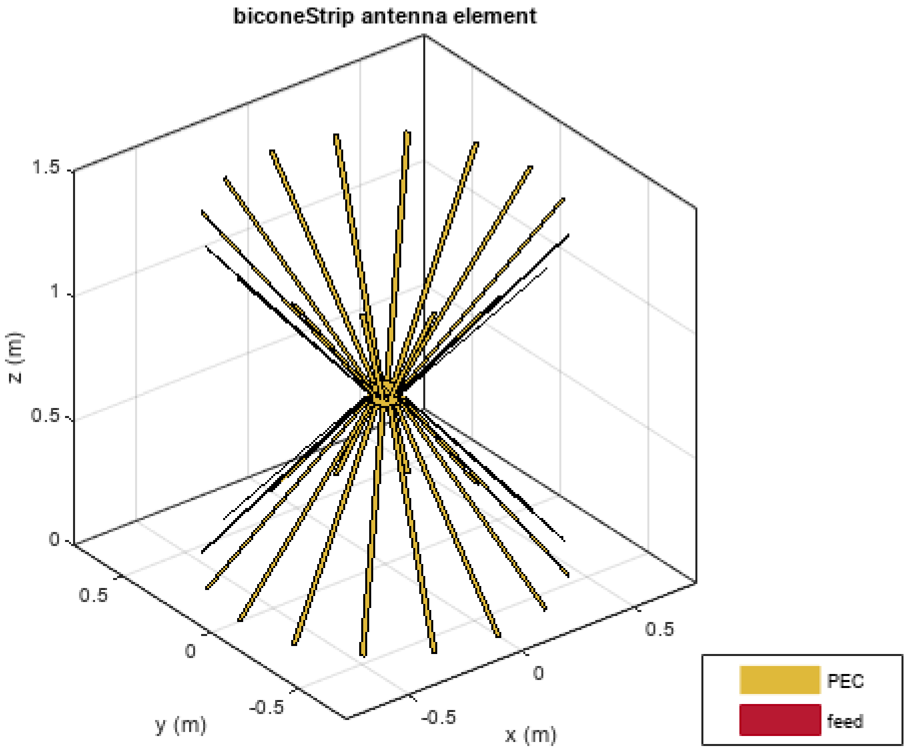
Create a strip bicone antenna with default properties.

```
ant = biconeStrip
```

```
ant = 
  biconeStrip with properties:

       NumStrips: 16
      StripWidth: 0.0180
       HatHeight: 0
      ConeHeight: 0.6650
    NarrowRadius: 0.0700
     BroadRadius: 0.6470
      FeedHeight: 0.0450
       FeedWidth: 0.0400
       Conductor: [1x1 metal]
            Tilt: 0
        TiltAxis: [1 0 0]
            Load: [1x1 lumpedElement]
```

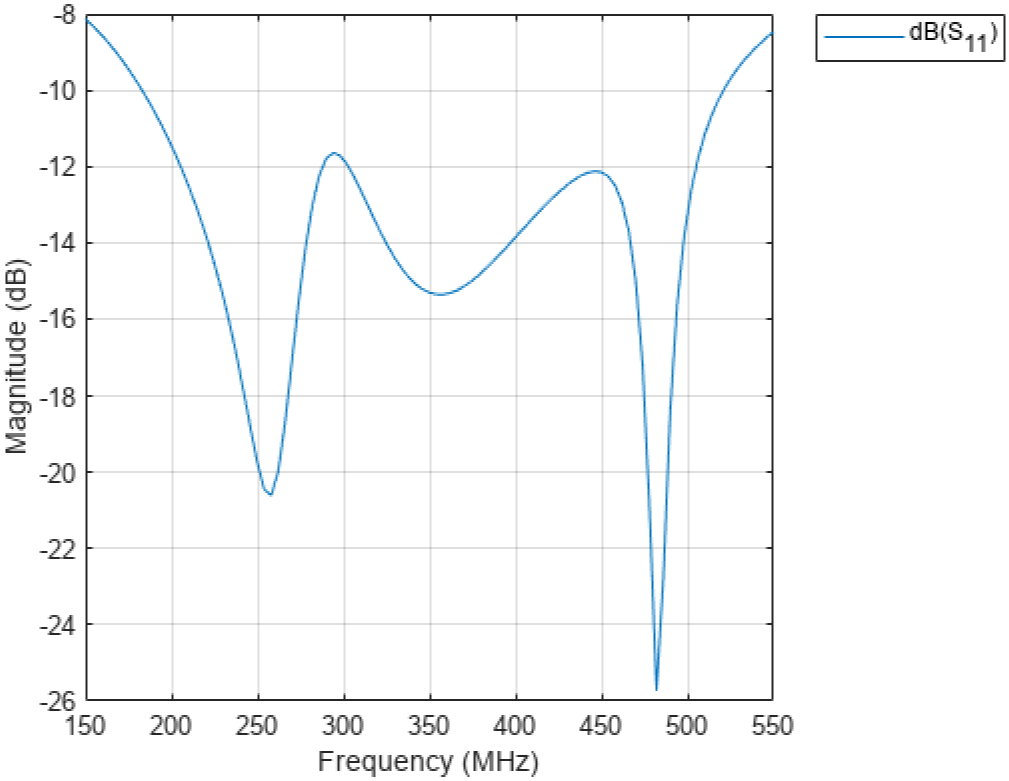
View the antenna using the `show` function.

```
show(ant);
```



Plot the S-parameters of the antenna over the frequency span of 150-550 MHz.

```
s = sparameters(ant,linspace(150e6,550e6,101));
rfplot(s)
```



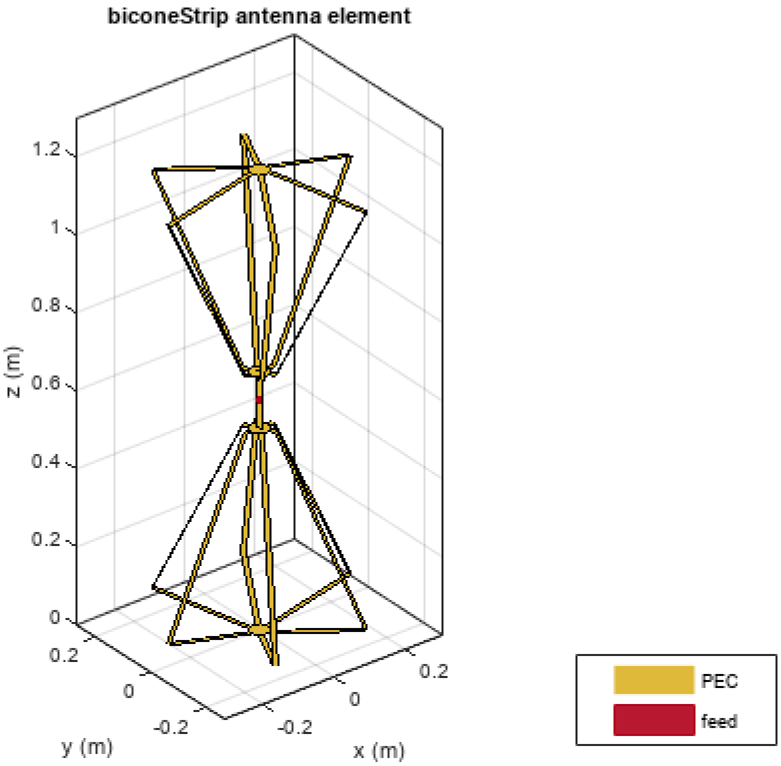
Create Strip Bicone Antenna with Hat

Create a strip bicone antenna with hat.

```
ant = biconeStrip("NumStrips",6,"StripWidth",12e-3,"HatHeight",53e-3, ...
    "ConeHeight",465e-3,"NarrowRadius",40e-3,"BroadRadius",257e-3, ...
    "FeedHeight",144e-3,"FeedWidth",25e-3);
```

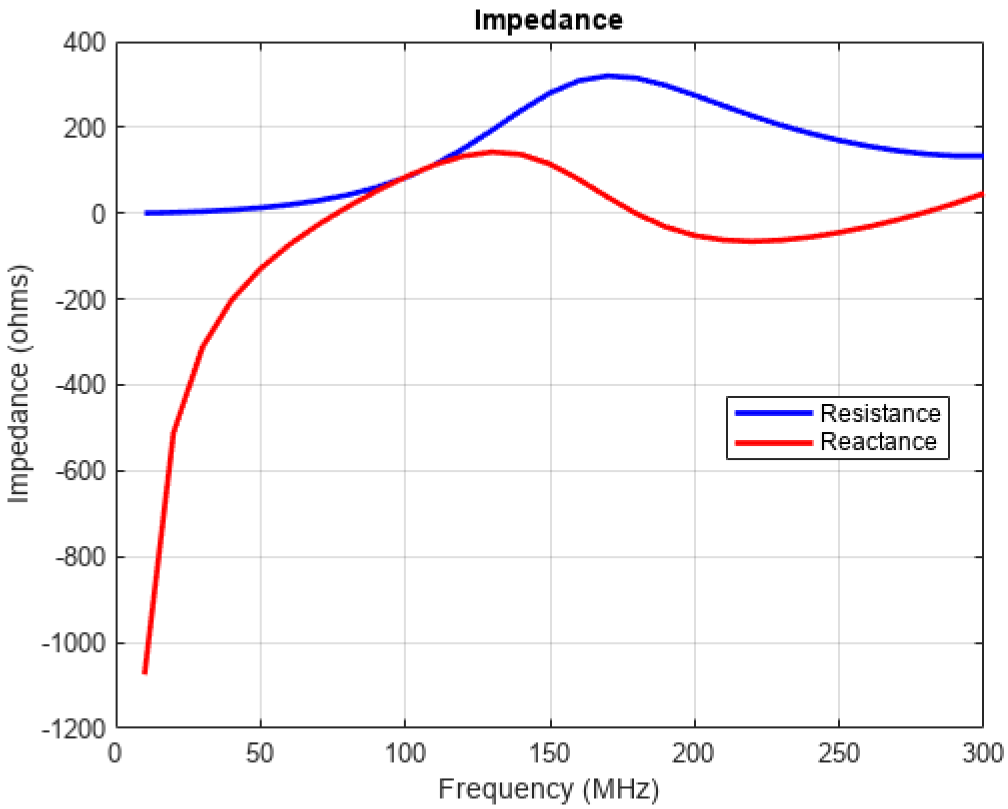
View the antenna using the `show` function.

```
show(ant)
```



Calculate antenna impedance over the frequency span of 10-300 MHz.

```
impedance(ant,10e6:10e6:300e6)
```



More About

Parametric Analysis Guidelines

To understand how the properties of the `biconeStrip` antenna object influence the antenna design, use the following parametric analysis guidelines.

- To increase the operating frequency, decrease the dimensions of the `biconeStrip` antenna.
- To increase the impedance bandwidth, use the “`NumStrips`” on page 1-0 property to increase the number of strips in the `biconeStrip` antenna object.
- To improve average input impedance, decrease the narrow radii and feed height of the antenna using the “`NarrowRadius`” on page 1-0 and the “`FeedHeight`” on page 1-0 properties.

Version History

Introduced in R2020b

References

[1] Brian A. Austin, Andre P. C. Fourie "Characteristics of the Wire Biconical Antenna Used for EMC Measurements", *IEEE Transaction on Electromagnetic Compatibility*, vol. 33, no. 3, August 1991.

See Also

`bicone` | `discone` | `disconeStrip`

Topics

"Rotate Antennas and Arrays"

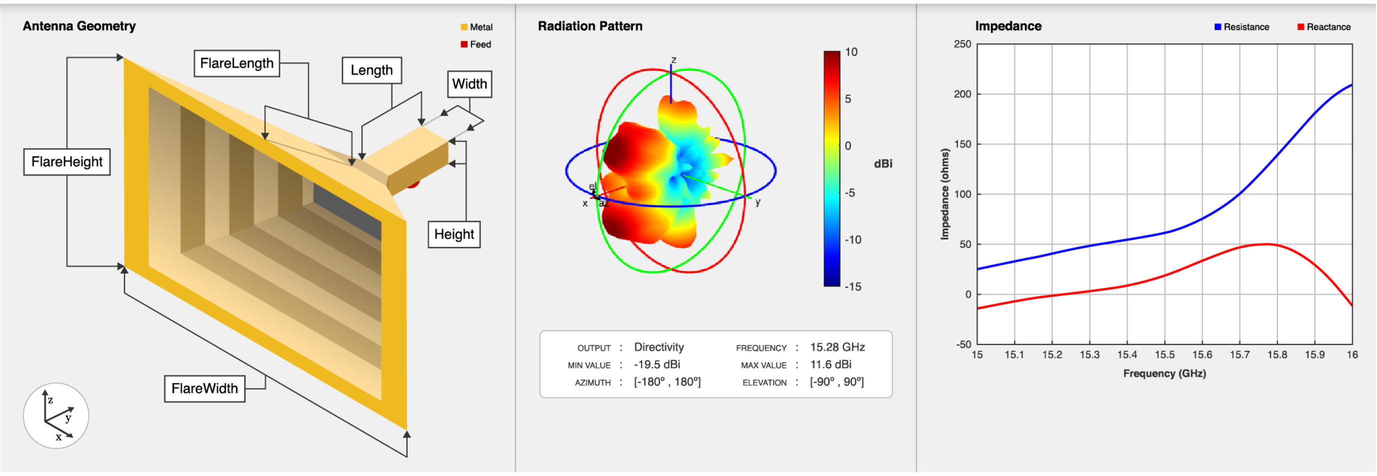
hornCorrugated

Create rectangular corrugated-horn antenna

Description

The `hornCorrugated` object creates a rectangular corrugated-horn antenna with grooves on the inner walls of the flare. These antennas provide spillover reduction and have beam symmetry and a low sidelobe level, so they are widely used as a feed in reflector antennas in broadcasting communications.

Horn Corrugated



Creation

Syntax

```
ant = hornCorrugated
ant = hornCorrugated(Name,Value)
```

Description

`ant = hornCorrugated` creates a rectangular corrugated-horn antenna for a resonant frequency around 15 GHz.

`ant = hornCorrugated(Name,Value)` sets “Properties” on page 1-716 using one or more name-value pairs. For example, `ant = hornCorrugated('FlareLength', 0.045)` creates a rectangular corrugated-horn antenna with the flare length of the horn set to 45 mm.

Properties

`FlareLength` — Flare length of horn

`0.0428` (default) | positive scalar

Flare length of the horn, specified as a positive scalar in meters.

Example: `'FlareLength',0.35`

Data Types: `double`

`FlareWidth` — Flare width of horn

`0.09` (default) | positive scalar

Flare width of the horn, specified as a positive scalar in meters.

Example: `'FlareWidth',0.2`

Data Types: `double`

`FlareHeight` — Flare height of horn

`0.06` (default) | positive scalar

Flare height of the horn, specified as a positive scalar in meters.

Example: `'FlareHeight',0.15`

Data Types: `double`

`Length` — Length of rectangular waveguide

`0.0229` (default) | positive scalar

Length of the rectangular waveguide, specified as a positive scalar in meters.

Example: `'Length',0.09`

Data Types: `double`

`Width` — Width of rectangular waveguide

`0.0102` (default) | positive scalar

Width of the rectangular waveguide, specified as a positive scalar in meters.

Example: `'Width',0.05`

Data Types: `double`

`Height` — Height of rectangular waveguide

`0.0075` (default) | positive scalar

Height of the rectangular waveguide, specified as a positive scalar in meters.

Example: `'Height',0.0200`

Data Types: `double`

`FeedHeight` — Height of feed

`0.0037` (default) | positive scalar

Height of the feed, specified as a positive scalar in meters.

Example: `'FeedHeight',0.0050`

Data Types: `double`

`FeedWidth` — Width of feed

`0.00008` (default) | positive scalar

Width of the feed, specified as a positive scalar in meters.

Example: `'FeedWidth',5e-05`

Data Types: `double`

`FeedOffset` — Signed distance of feedpoint from center of ground plane
`[–0.0020 0]` (default) | two-element vector

Signed distance of the feedpoint from the center of the ground plane, specified as a two-element vector in meters.

Example: `'FeedOffset',[–0.0070 0.01]`

Data Types: `double`

`Pitch` — Distance between two successive corrugations
`0.0060` (default) | positive scalar

Distance between two successive corrugations, specified as a positive scalar in meters.

Example: `'Pitch',0.0060`

Example: `ant.Pitch = 0.0090`

Data Types: `double`

`FirstCorrugatedDistance` — Distance of first corrugation from waveguide
`0.016` (default) | positive scalar

Distance of the first corrugation from the waveguide, specified as a positive scalar in meters.

Example: `'FirstCorrugatedDistance',0.0360`

Example: `ant.FirstCorrugatedDistance = 0.0190`

Data Types: `double`

`CorrugateWidth` — Corrugation width
`0.003` (default) | positive scalar

Corrugation width, specified as a positive scalar in meters.

Example: `'CorrugateWidth',0.0058`

Example: `ant.CorrugateWidth = 0.0019`

Data Types: `double`

`CorrugateDepth` — Corrugation depth
`[0.0050 0.0100]` (default) | two-element vector

Corrugation depth, specified as a two-element vector in meters. The first element corresponds to the width along E-plane, and the second element corresponds to the width along the H-plane.

Example: `'CorrugateDepth',[0.006 0.0560]`

Example: `ant.CorrugateDepth = [0.0050 0.0790]`

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'x'` | `'y'` | `'z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the *x*-, *y*-, and *z*-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, '*x*', '*y*', or '*z*'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`. `lumpedElement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes

axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
numCorrugationsToPitch	Calculate pitch for specified corrugations
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Rectangular Corrugated-Horn Antenna and Plot Radiation Pattern

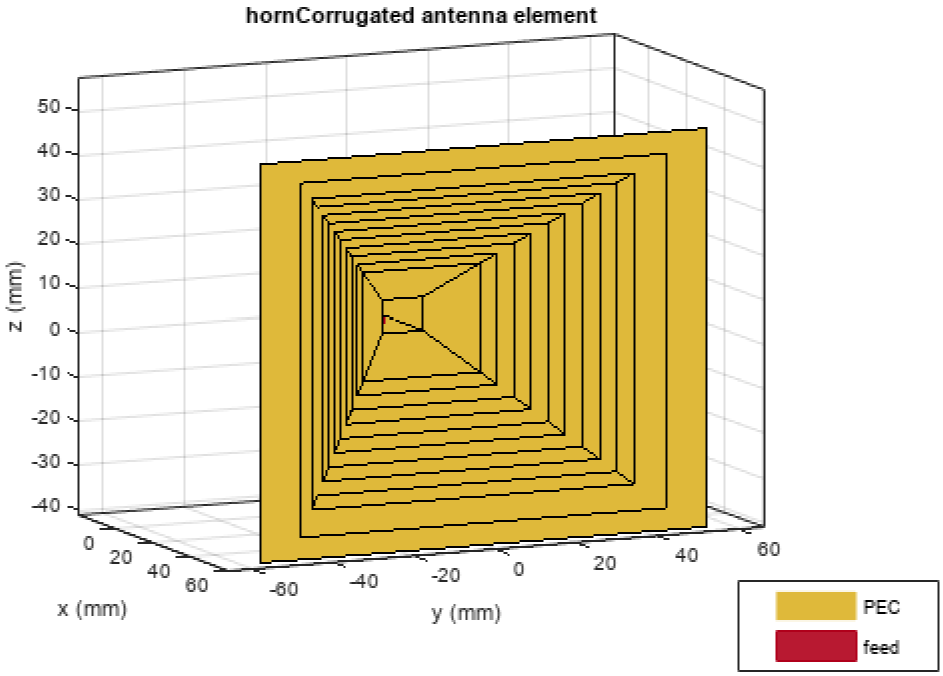
Create a default rectangular corrugated-horn antenna.

```
ant = hornCorrugated
```

```
ant = 
  hornCorrugated with properties:

               FlareLength: 0.0428
                FlareWidth: 0.0900
               FlareHeight: 0.0800
                    Length: 0.0229
                     Width: 0.0102
                    Height: 0.0075
                 FeedWidth: 8.0000e-05
                FeedHeight: 0.0037
                FeedOffset: [-0.0020 0]
    FirstCorrugateDistance: 0.0160
            CorrugateDepth: [0.0050 0.0100]
            CorrugateWidth: 0.0030
                     Pitch: 0.0060
                 Conductor: [1x1 metal]
                      Tilt: 0
                  TiltAxis: [1 0 0]
                      Load: [1x1 lumpedElement]
```

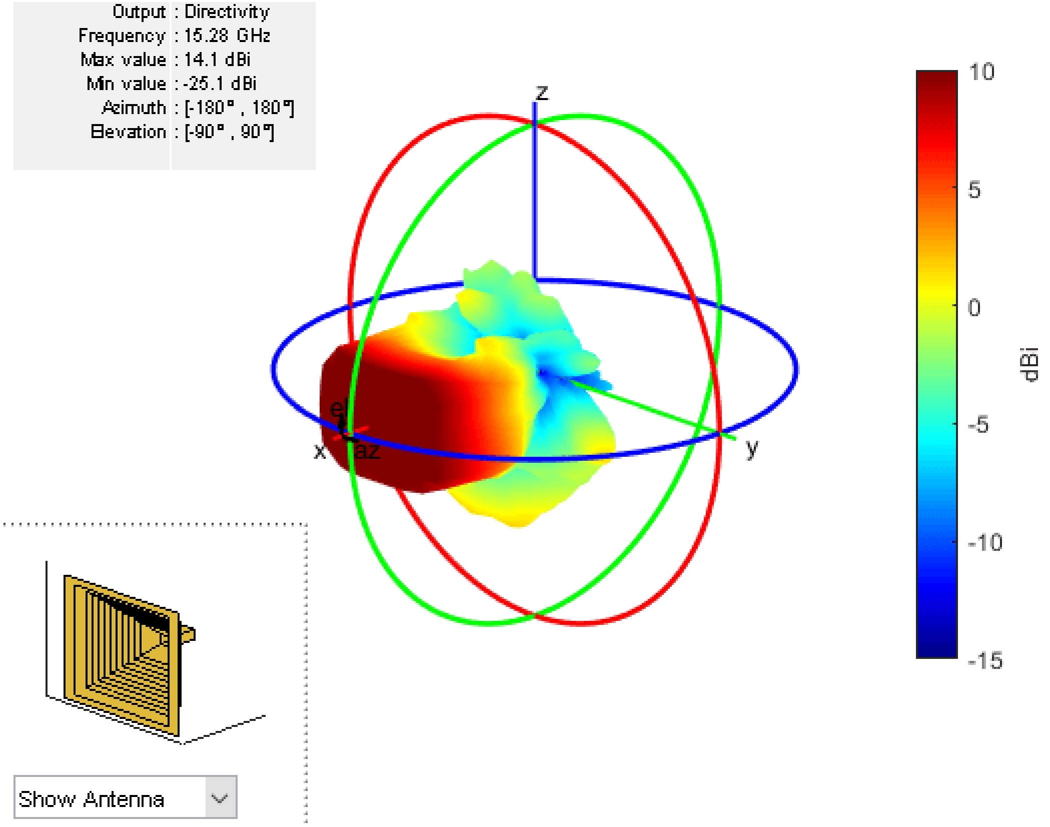
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of the antenna at a frequency 15.28 GHz.

```
p = PatternPlotOptions('MagnitudeScale',[-15 10]);
pattern(ant,15.28e9,'patternOptions',p)
```



Output : Directivity
Frequency : 15.28 GHz
Max value : 14.1 dBi
Min value : -25.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2020b

References

[1] Encinar, J., and J. Rebollar. “A Hybrid Technique for Analyzing Corrugated and Noncorrugated Rectangular Horns.” *IEEE Transactions on Antennas and Propagation*, vol. 34, no. 8, Aug. 1986, pp. 961–68.

See Also

`waveguide` | `horn` | `hornConical` | `hornConicalCorrugated`

Topics

“Rotate Antennas and Arrays”

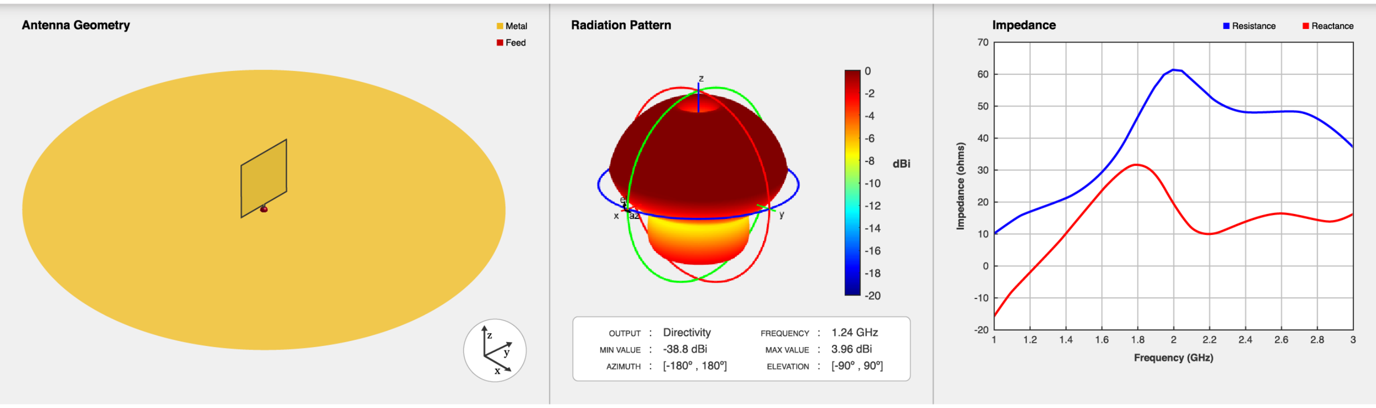
monopoleCustom

Create customized monopole antenna

Description

The `monopoleCustom` object creates a monopole radiator of any shape using the `antenna.Shape` class. The ground plane can take any shape. You can create any arbitrarily shaped monopole and analyze it for field, surface, and port characteristics. Monopole antennas have a simple structure and provide omnidirectional radiation patterns with wide impedance bandwidth. Monopole antennas are commonly used in airborne and ground-based communication systems.

Monopole Custom



OUTPUT : Directivity
MIN VALUE : -38.8 dBi
AZIMUTH : [-180° , 180°]
FREQUENCY : 1.24 GHz
MAX VALUE : 3.96 dBi
ELEVATION : [-90° , 90°]

Creation

Syntax

```
ant = monopoleCustom
ant = monopoleCustom(Name,Value)
```

Description

`ant = monopoleCustom` creates a default monopole antenna with a square radiator and a circular ground plane. The feed point is at the origin in the *xy*- plane. The default antenna resonates at an operating frequency of 1.24 GHz.

`ant = monopoleCustom(Name,Value)` sets “Properties” on page 1-723 using one or more name-value pairs. For example, `ant = monopoleCustom('RadiatorTilt',90)` creates a monopole antenna with tilt angle of the radiator at 90 degrees on the *z*-axis.

Properties

`Radiator` — Type of radiator

`antenna.Rectangle` object (default) | `antenna.Polygon` object

Type of radiator, specified as an `antenna.Polygon` object. You can specify any shape for the radiator. The feed strip is a part of the radiator. By default, the radiator is square in shape with a side length of 40e-3 meters. The feed strip is 2e-3 meters in length and 2.5e-3 meters in width at the edge of the radiator.

`GroundPlane` — Type of ground plane
`antenna.Circle` (default) | `antenna.Polygon` object

Type of ground plane, specified as an `antenna.Polygon` object. You can specify any shape for the ground plane. By default, the ground plane is circular in shape with a radius of 150e-3 meters.

`FeedOffset` — Signed distance from center along length and width of ground plane
`[0 0]` (default) | two-element vector

Signed distance from the center along the length and the width of the ground plane, specified as a two-element vector in meters.

Example: `'FeedOffset',[2 1]`

Data Types: `double`

`RadiatorTilt` — Tilt angle of radiator
`0` (default) | scalar

Tilt angle of the radiator, specified as a scalar in degrees.

Data Types: `double`

`Conductor` — Type of metal material
`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'x'` | `'y'` | `'z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the *x*-, *y*-, and *z*-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, '*x*', '*y*', or '*z*'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`. `lumpedElement` is the object for the load created using `lumpedElement`.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Display antenna radiation pattern in Site Viewer
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

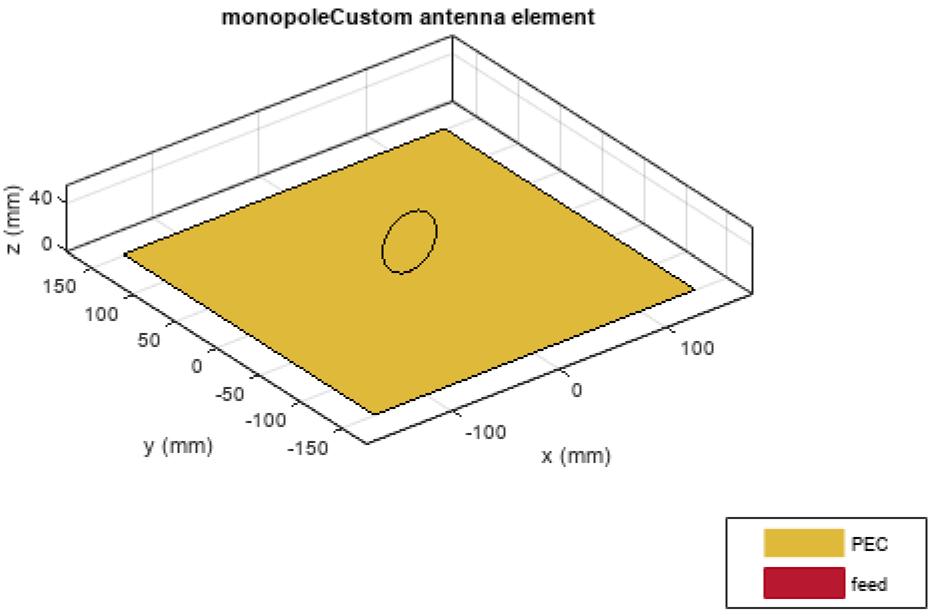
Create Disc Monopole Antenna on Square Ground Plane

Create a disc monopole with a of radius 25 mm, on a square ground plane of 30 cm, and with a feed gap of 0.7 mm.

```
Rad = antenna.Circle('Radius',25e-3);
FeedStrip = antenna.Rectangle('Length',1e-3,'Width',0.7e-3, ...
                               'Center',[0 -(Rad.Radius+(0.7e-3)*0.3)]);
m = monopoleCustom;
m.Radiator = Rad+FeedStrip;
m.GroundPlane = antenna.Rectangle('Length',300e-3,'Width',300e-3);
```

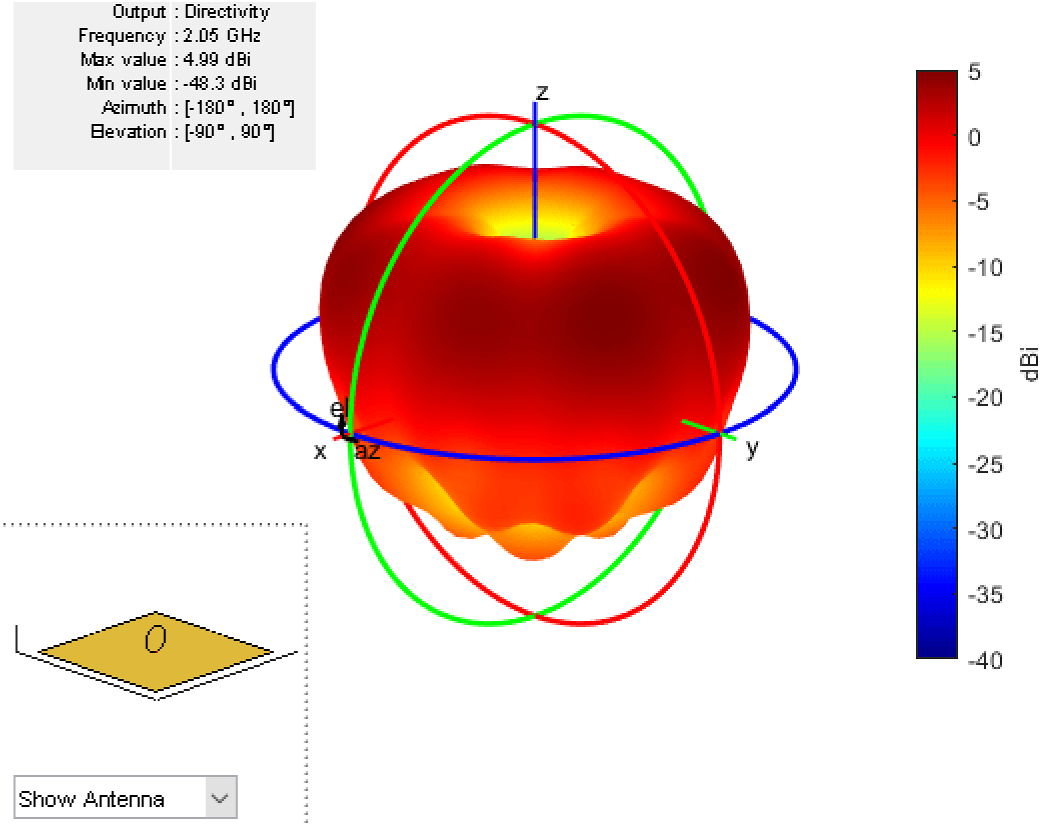
View the antenna using the show function.

```
show(m);
```



Plot the radiation pattern of the antenna at 2.05 GHz.

```
p = PatternPlotOptions('MagnitudeScale',[-40 5]);
pattern(m,2.05e9,'patternOptions',p);
```



Output : Directivity
Frequency : 2.05 GHz
Max value : 4.99 dBi
Min value : -48.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2020b

References

[1] Ammann, M. J. “Square Planar Monopole Antenna.” *IEE National Conference on Antennas and Propagation*, vol. 1999, IEE, pp. 37–40.

[2] Weiner, M. “Monopole Element at the Center of a Circular Ground Plane Whose Radius Is Small or Comparable to a Wavelength.” *IEEE Transactions on Antennas and Propagation*, vol. 35, no. 5, pp. 488–495.

[3] N. P. Agrawall, G. Kumar and K. P. Ray, "Wide-band planar monopole antennas," in *IEEE Transactions on Antennas and Propagation*, vol. 46, no. 2, pp. 294-295.

See Also

`monopole` | `monopoleTopHat` | `monopoleRadial`

Topics

“Rotate Antennas and Arrays”

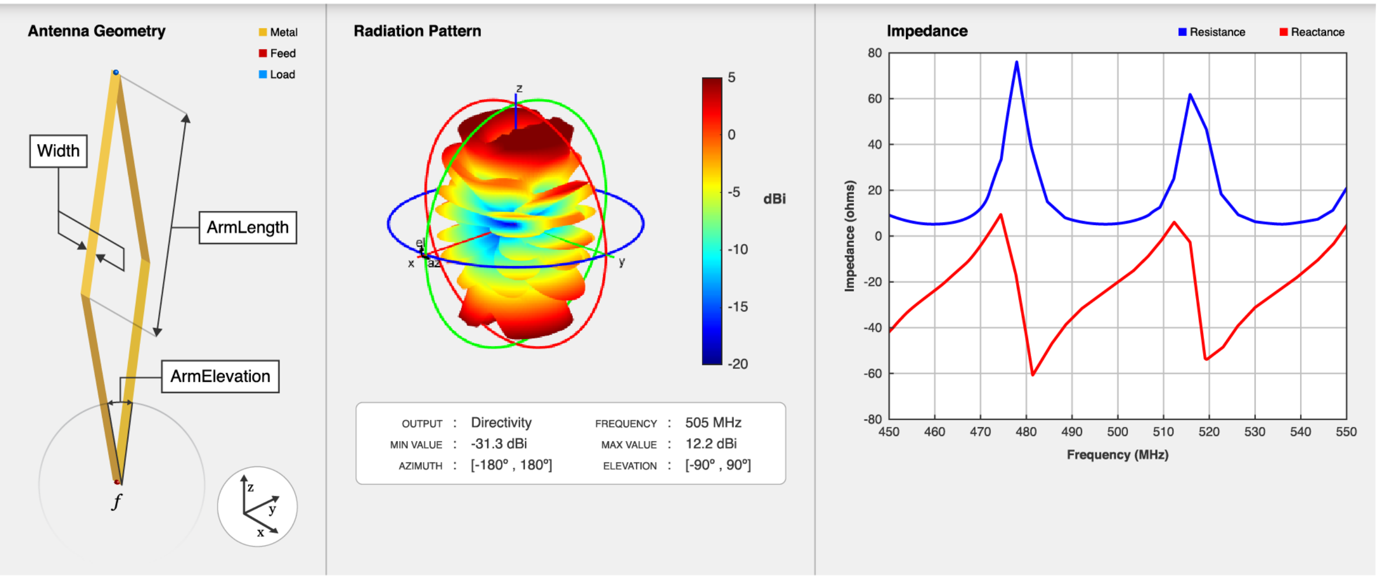
rhombic

Create a rhombic antenna

Description

The `rhombic` object creates a rhombic antenna. It consists of a rhombus with a feed at one acute angles and a termination resistor at the other acute angle. It has a simple design and is highly directional. These antennas are used in shortwave radio broadcasting and point-to-point communications.

Rhombic Antenna



OUTPUT : Directivity
MIN VALUE : -31.3 dBi
AZIMUTH : [-180° , 180°]
FREQUENCY : 505 MHz
MAX VALUE : 12.2 dBi
ELEVATION : [-90° , 90°]

Creation

Syntax

```
ant = rhombic
ant = rhombic(Name,Value)
```

Description

`ant = rhombic` creates a rhombic antenna. The dimensions are chosen for resonant frequency of 510 MHz. The default rhombic antenna is fed at one acute angle and the other acute angle is terminated with a load of 500 ohms.

`ant = rhombic(Name,Value)` sets “Properties” on page 1-729 using one or more name-value pairs. For example, `ant = rhombic('ArmLength', 3)` creates a rhombic antenna with an arm of length 3 meters.

Properties

`ArmLength` — Length of arm

`2` (default) | positive scalar

Length of each of the rhombus, specified as a scalar in meters.

Data Types: `double`

`ArmElevation` — Angle between adjacent arms at feed

`20` (default) | positive scalar

The acute angle between the adjacent arms at the feed location, specified as a scalar in degrees.

Data Types: `double`

`Width` — Width of arm

`0.1` (default) | positive scalar

Width of the arm of the rhombus, specified as a scalar in meters.

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. The load element is located opposite the feed at one of the acute angles of the rhombus. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is the where `lumpedElement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.

- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

`Tilt` — Tilt angle of antenna
`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

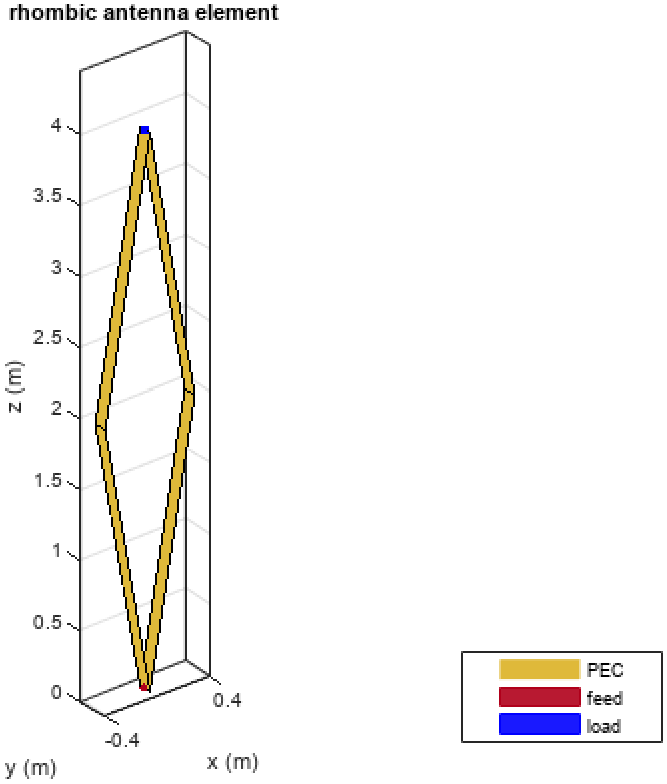
Create Default Rhombic Antenna and Plot Radiation Pattern

Create a default rhombic antenna.

```
ant = rhombic
ant = 
  rhombic with properties:

        ArmLength: 2
     ArmElevation: 20
            Width: 0.1000
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

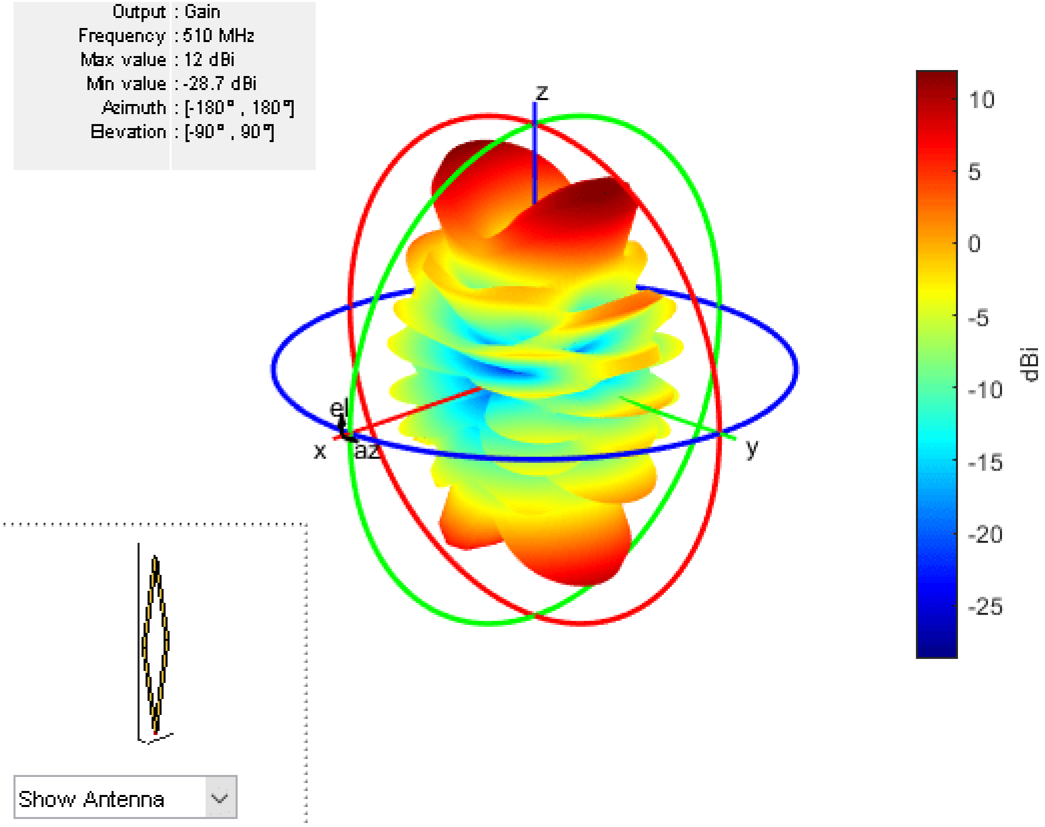
View the antenna using the `show` function.

```
show(ant);
```



Plot the radiation pattern of the antenna at 510 MHz.

```
pattern(ant, 510e6);
```



Output : Gain
Frequency : 510 MHz
Max value : 12 dBi
Min value : -28.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2020b

References

[1] Decker, R. “The Influence of Gain and Current Attenuation on the Design of the Rhombic Antenna.” *IRE Transactions on Antennas and Propagation* 7, no. 2 (April 1959): 188–196.

See Also

`dipole` | `dipoleVee` | `biquad`

Topics

“Rotate Antennas and Arrays”

PCBReader

Import and update Gerber files

Description

Use the PCBReader object to create a printed circuit board (PCB) reader to import Gerber files and to facilitate the creation of an antenna model. A Gerber file is a set of manufacturing files used to describe a PCB antenna. A Gerber file uses an ASCII vector format to describe 2-D binary images.

Creation

You can create a PCBReader object using the following methods:

- gerberRead — Create a PCBReader object with the specified Gerber and drill files.
- The PCBReader function described here.

Syntax

```
B = PCBReader(S)
B = PCBReader(Name,Value)
```

Description

B = PCBReader(S) creates a PCBReader object that imports multilayer PCB antenna design files described in the stackUp object .

Note

- To translate the center of an imported symmetrical or asymmetrical polygon to [0,0] please use the following MATLAB® functions, boundingbox and centroid. See examples.
- The PCBReader object reads RS-274X Gerber files. It does not support RS-274D Gerber files.

B = PCBReader(Name,Value) sets “Properties” on page 1-734 using name-value pairs. For example, B = PCBReader('StackUp',S,'Drillfile','ant.txt') imports the layer and drill files into the PCBReader. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Input Arguments

S — PCB stackup definition
stackUp object

PCB stackup definition, specified as a stackUp object. For more information, see stackUp.

Example: S = stackUp; B = PCBReader(S)

Example: `B = PCBReader('StackUp',S)`

Properties

`StackUp` — PCB stackup definition

`stackUp` object

PCB stackup definition, specified as a `stackUp` object.

Example: `S = stackUp; B.StackUp = S;`

Example: `B = PCBReader('StackUp',S)`

`DrillFile` — Name of Excellon drill file

[ ] (default) | character vector | string scalar

Name of Excellon drill file, specified as a character vector or string scalar. You can specify either a DRL or a TXT file.

Example: `B.DrillFile = 'ant.drl'`

`NumPointsOnCurves` — Discretization points on curved segments

`50` (default) | positive scalar

Discretization points on curved segments, specified as a positive scalar.

Example: `B.NumPointsOnCurves = 80`

Object Functions

pcbStack	Single-feed or multifeed PCB antenna
shapes	Extract and modify metal layers from PCBReader object

Examples

Import Gerber Files Using PCB Stackup Definition

Create a default PCB stackup definition object.

```
S = stackUp;
```

Set the thickness of the dielectric Air in layer 1 and layer 5 of the `stackUp` object to 0.1 mm.

```
S.Layer1.Thickness = 0.1e-3;
S.Layer5.Thickness = 0.1e-3;
```

Import a top layer Gerber file to layer 2.

```
S.Layer2 = 'antenna_design_file.gtl';
```

Import a bottom layer Gerber file to layer 4.

```
S.Layer4 = 'antenna_design_file.gbl';
```

Create a `PCBReader` object, `B`, using the `stackUp` object, `S`.

```
B = PCBReader('StackUp',S);
```

Update and Analyze Imported Gerber File

Create a default PCB stackup definition object.

```
s = stackUp;
```

Import a top layer Gerber file to layer 2.

```
s.Layer2 = 'patchMicrostripCircular_design_file.gtl';
```

Create a PCBReader object using the stackUp object.

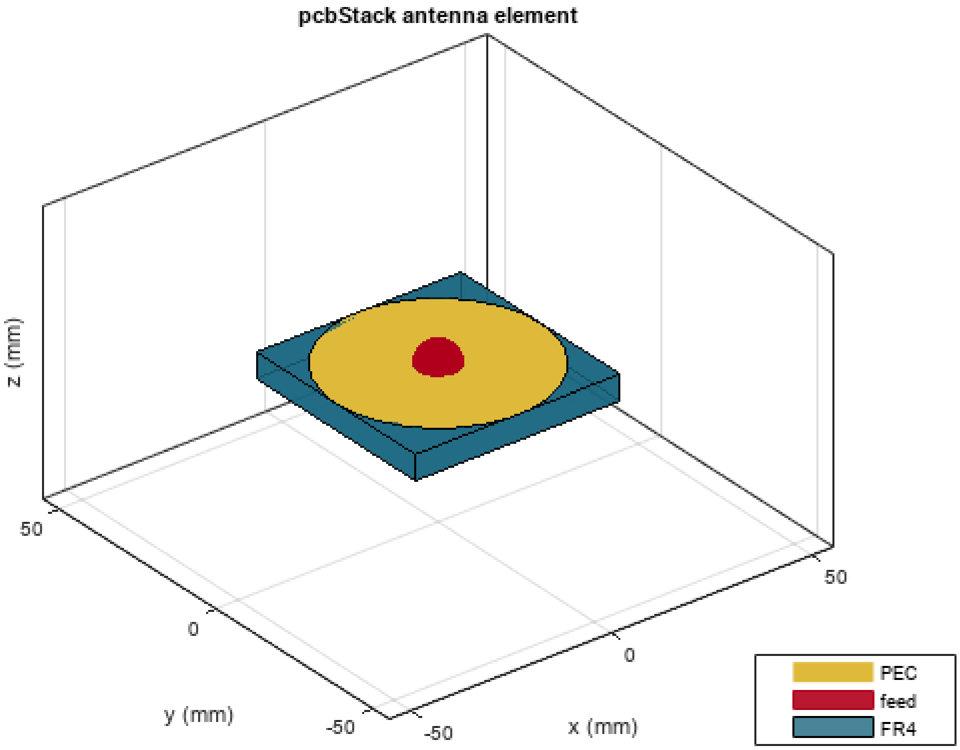
```
p = PCBReader('StackUp',s);
```

To update the Gerber file, convert the PCBReader object to a pcbStack object.

```
p3 = pcbStack(p);
```

View the pcbStack object.

```
figure
show(p3)
```

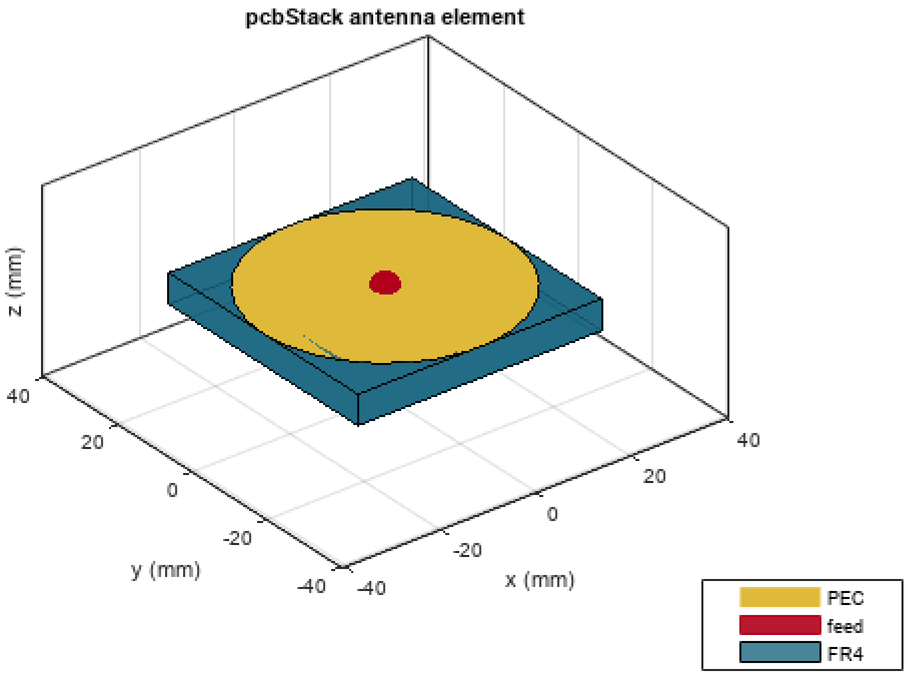


Update the feed diameter.

```
p3.FeedDiameter = 0.005;
```

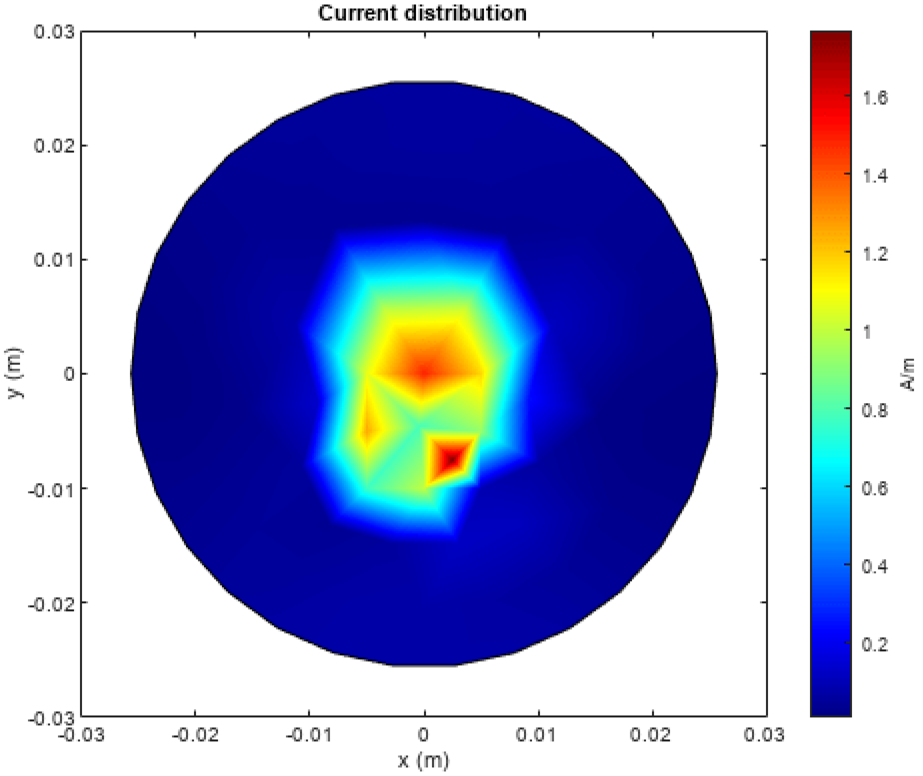
View the updated `pcbStack` object.

```
figure
show(p3)
```



Plot the current distribution on the antenna at 2.4 GHz.

```
figure
current(p3,2.4e9)
```



Extract Metal from Two-Layer Design PCBReader Object

Create a PCBReader object.

```
B = PCBReader;
```

Import a two-layer design.

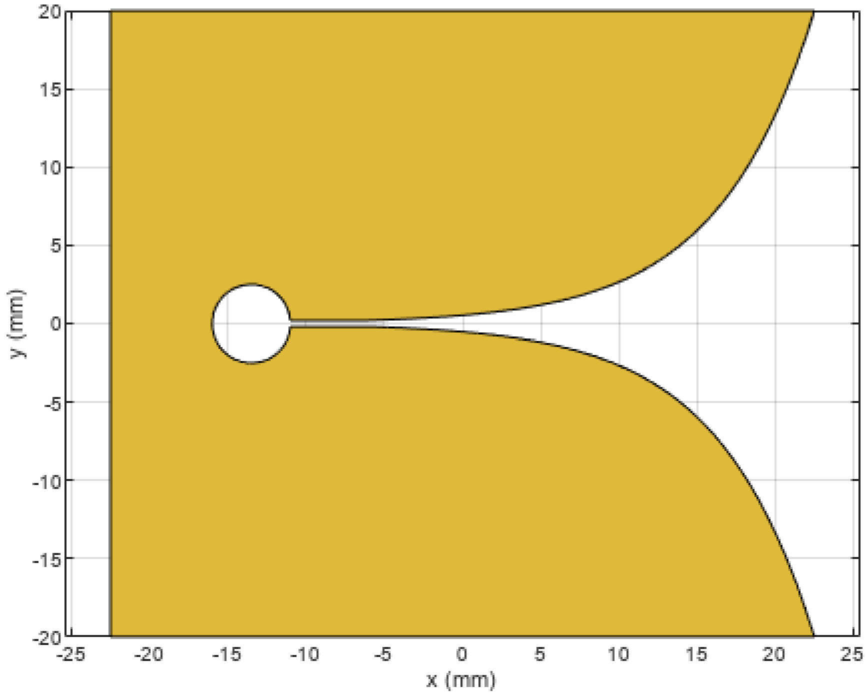
```
st = B.StackUp;
st.Layer2 = 'UWBVivaldi.gtl';
st.Layer4 = 'UWBVivaldi.gbl';
B.StackUp = st;
```

Extract shapes from the metal layers.

```
S = shapes(B);
```

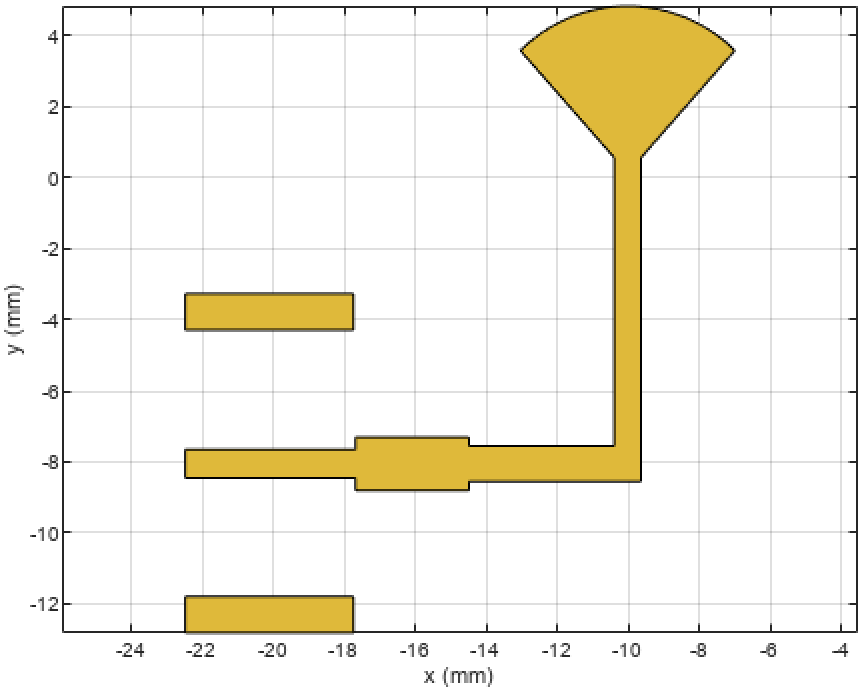
View the top-layer Gerber file.

```
figure
show(S(1))
```



View the bottom-layer Gerber file.

```
figure
show(S(2))
```



Translate Center of Imported Symmetrical Polygon to [0,0]

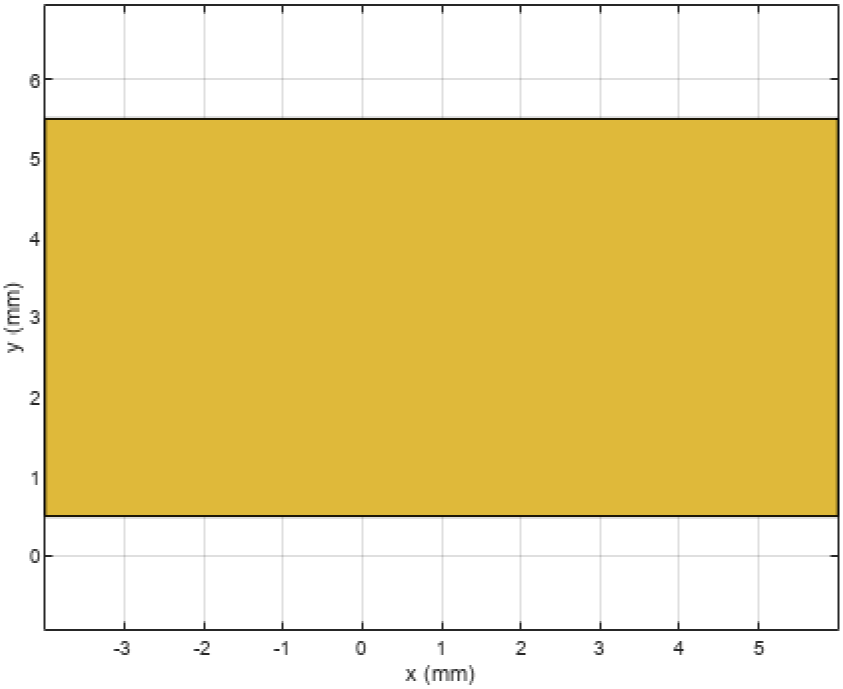
This example will show how to translate the symmetrical polygon imported from the Gerber file to the respective co-ordinates.

Create a PCB stackup and import rectangular patch on it.

```
S = stackUp;
S.Layer2 = 'PatchRectangular.gtl';
S.Layer3 = dielectric('Teflon');
```

Use a PCB Reader to read the polygon shape from the stackup.

```
p1 = PCBReader ('StackUp',S);
figure; show(p1.shapes);
```

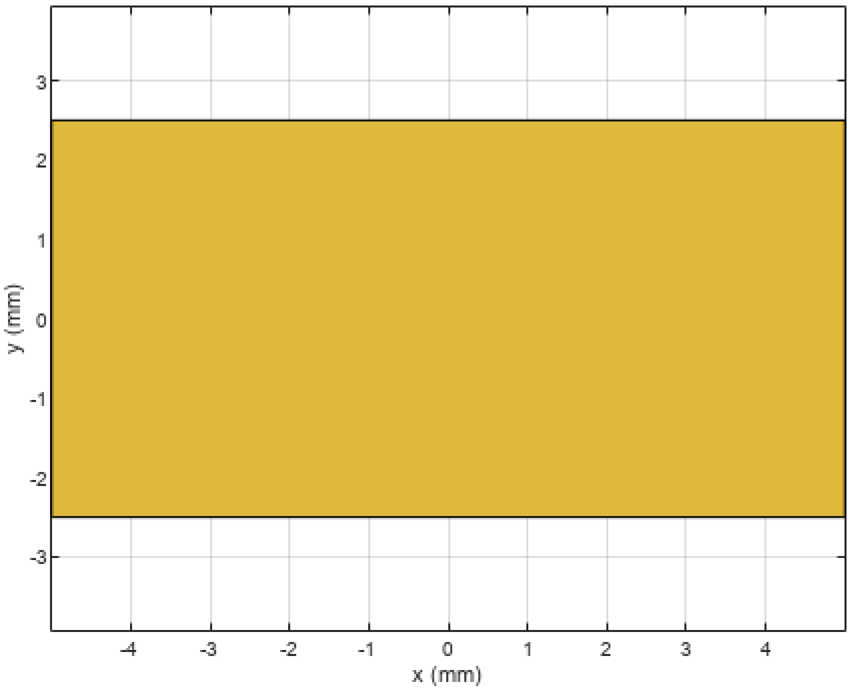


Translate the shape with center (0,0) using the `centroid` function from MATLAB.

```
s = p1.shapes
```

```
s = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [4x3 double]
```

```
polygon = s;
[x,y] = centroid(polygon);
translate(polygon,[-x, -y, 0]);
```



Translate Center of Imported Asymmetrical Polygon to [0,0]

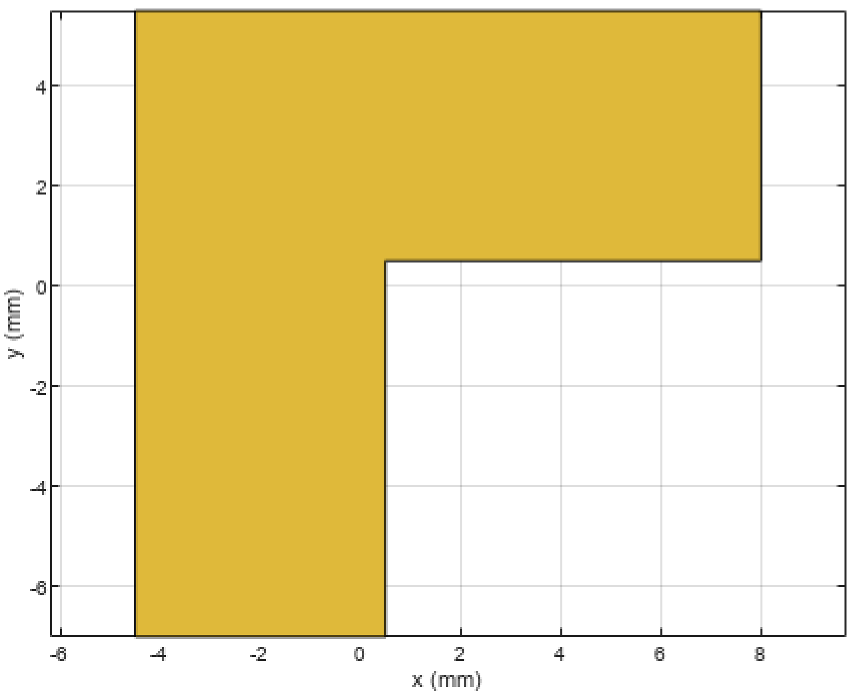
This example shows how to translate the asymmetrical polygon imported from the Gerber file to the respective co-ordinates.

Create a PCB stackup and import rectangular patch on it.

```
S = stackUp;
S.Layer2 = 'RightAngleBend.gtl';
S.Layer3 = dielectric('Teflon');
```

Use a PCB Reader to read the polygon shape from the stackup.

```
p1 = PCBReader ('StackUp',S);
figure; show(p1.shapes);
```

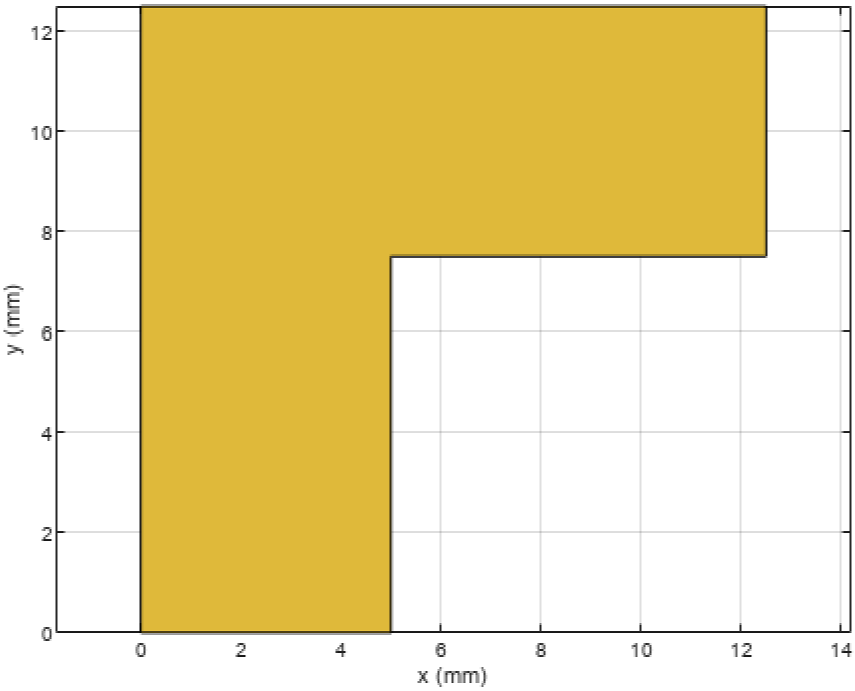


Translate the shape's bottom left corner to (0,0). Use the boundingbox function from MATLAB to convert the shape to polyshape and find the upper and lower bounds of the shape.

```
s = p1.shapes
```

```
s = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [6x3 double]
```

```
ver = s.Vertices(:,1:2);
polygon = polyshape(ver);
[xlim, ylim] = boundingbox(polygon);
translate(s,[-xlim(1), -ylim(1), 0]);
```



Version History

Introduced in R2020b

See Also

`PCBWriter` | `PCBServices` | `PCBConnectors` | `pcbStack` | `stackUp` | `gerberRead`

Topics

“Create Antenna Model from Gerber Files”

stackUp

Create PCB stackup definition

Description

Use the `stackUp` object to create a printed circuit board (PCB) stackup definition to import Gerber files. A Gerber file is a set of manufacturing files used to describe a PCB antenna. A Gerber file uses an ASCII vector format for 2-D binary images.

Creation

Syntax

`s = stackUp`

Description

`s = stackUp` creates a default PCB stackup object with five layers. Specify Gerber files as inputs to the second and fourth layers. Specify dielectric material objects as inputs to layers one, three, and five.

Properties

`NumLayers` — Number of layers in stackup
5 (default) | positive scalar

This property is read-only.

Number of layers in the stackup, returned as a positive scalar.

`Layer1` — First layer in stackup
`'Air'` (default) | `dielectric` object

First layer in the stackup definition object, specified as a `dielectric` object.

Example: `s = stackUp; d = dielectric('RO4725JXR'); s.Layer1 = d;`

`Layer2` — Second layer in stackup
character vector | string scalar

Second layer in the stackup definition object, specified as a character vector or string. The file should be saved as a GTL, GBL, or GBR file.

Example: `s = stackUp; s.Layer2 = 'antenna_design_file.gtl';`

Note The Gerber file must be imported to the MATLAB workspace before setting this property.

Layer3 — Third layer in stackup

'FR4' (default) | dielectric object

Third layer in the stackup definition object, specified as a dielectric object.

Example: s = stackUp; d = dielectric('RO4725JXR'); s.Layer3 = d;

Layer4 — Fourth layer in stackup

character vector | string scalar

Fourth layer in the stackup definition object, specified as a character vector or string. The file should be saved as a GTL, GBL, or GBR file.

Example: s = stackUp; s.Layer4 = 'antenna_design_file.gbl';

Note The Gerber file must be imported to the MATLAB workspace before setting this property.

Layer5 — Fifth layer in stackup

'Air' (default) | dielectric object

Fifth layer in the stackup definition object, specified as a dielectric object.

Example: s = stackUp; d = dielectric('RO4725JXR'); s.Layer5 = d;

Note The Gerber file must be imported to MATLAB workspace before executing the above command.

Examples

Create PC Board Stackup Definition

Create a default PCB stackup definition object.

```
s = stackUp;
```

Create a dielectric object with Air as the dielectric material and with a thickness of 0.1 mm.

```
d1 = dielectric('Name','Air','Thickness',0.1e-3);
```

Create another dielectric object with RO4725JXR as the dielectric material.

```
d3 = dielectric('Name','RO4725JXR');
```

Assign the dielectrics to the first and third layers.

```
s.Layer1 = d1;
s.Layer3 = d3;
```

Input Gerber files to the second and fourth layers.

```
s.Layer2 = 'antenna_design_file.gtl';
s.Layer4 = 'antenna_design_file.gbl';
```

Display the stackup definition object.

```
s

s = 
  stackUp with properties:

    NumLayers: 5
       Layer1: [1x1 dielectric]
       Layer2: 'antenna_design_file.gtl'
       Layer3: [1x1 dielectric]
       Layer4: 'antenna_design_file.gbl'
       Layer5: [1x1 dielectric]
```

Version History

Introduced in R2020b

See Also

PCBReader | gerberRead | shapes | DielectricCatalog | dielectric

Topics

"Create Antenna Model from Gerber Files"

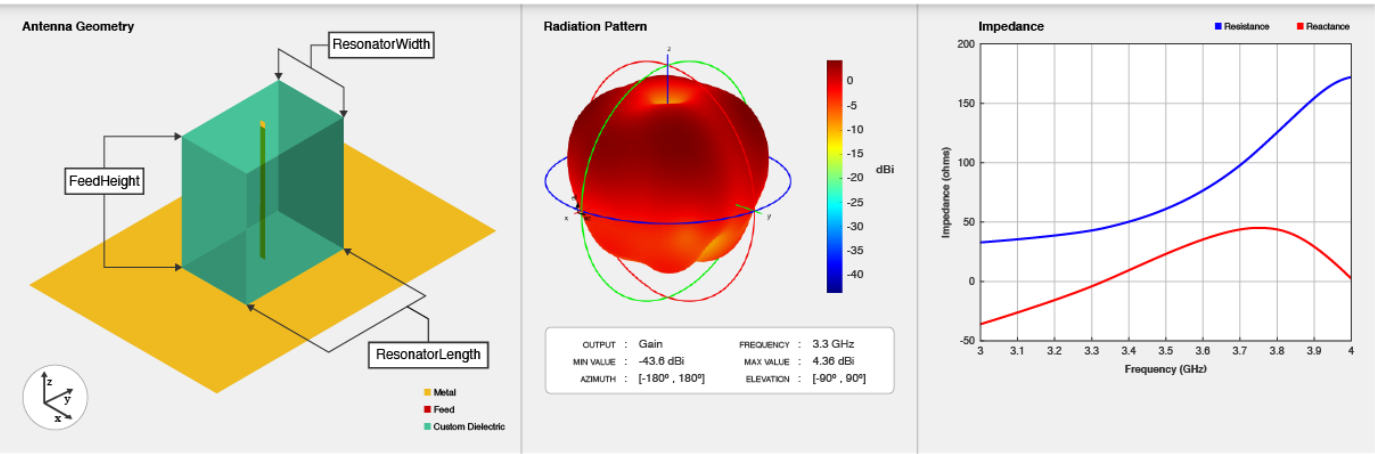
draRectangular

Create rectangular dielectric resonator antenna

Description

The `draRectangular` object creates a rectangular dielectric resonator antenna. The rectangular dielectric resonator antenna consists of a rectangular-shaped dielectric placed on a ground plane. It has high power-handling capacity and can provide high gain and bandwidth. The rectangular dielectric resonator antenna has the advantage of two aspect ratio that aids in the generation of various modes. These antennas are more suitable for use at microwave frequencies. Rectangular dielectric resonator antennas are widely used in satellite and radar systems.

draRectangular



Creation

Syntax

```
ant = draRectangular
ant = draRectangular(Name,Value)
```

Description

`ant = draRectangular` creates a rectangular dielectric resonator antenna with dimensions for a resonant frequency of 3.3 GHz. The default antenna is probe fed with the feedpoint at the origin.

`ant = draRectangular(Name,Value)` sets “Properties” on page 1-748 using one or more name-value pairs. For example, `draRectangular('ResonatorLength',0.04)` creates a rectangular dielectric resonator antenna with the length of the dielectric resonator set to 40 mm.

Properties

ResonatorLength — Length of dielectric resonator

0.03 (default) | positive scalar

Length of the dielectric resonator, specified as a positive scalar in meters.

Example: 'ResonatorLength',0.35

Data Types: double

ResonatorWidth — Width of dielectric resonator

0.015 (default) | positive scalar

Width of the dielectric resonator, specified as a positive scalar in meters.

Example: 'ResonatorWidth',0.30

Data Types: double

Substrate — Type of dielectric material

dielectric with EpsilonR 8.9, LossTangent 0.002 (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. You can choose any material from the DielectricCatalog or specify a dielectric of your choice. For more information, see dielectric. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be lesser than the ground plane dimensions.

Example: d = dielectric('FR4'); 'Substrate',d

Example: d = dielectric; d.Name = 'sub1'; d.EpsilonR = 2.3; d.LossTangent = 0.002; d.Thickness = 0.01; ant.Substrate = d;

GroundPlaneLength — Ground plane length along x-axis

0.16 (default) | positive scalar

Ground plane length, specified as a positive scalar in meters. By default, ground plane length is measured along the x-axis. Set 'GroundPlaneLength' to Inf to use the infinite ground plane technique for antenna analysis.

Example: 'GroundPlaneLength',120e-3

Data Types: double

GroundPlaneWidth — Ground plane width along y-axis

0.1 (default) | positive scalar

Ground plane width, specified as a positive scalar in meters. By default, ground plane width is measured along y-axis. Setting 'GroundPlaneWidth' to Inf to use the infinite ground plane technique for antenna analysis.

Example: 'GroundPlaneWidth',118e-3

Data Types: double

FeedWidth — Width of feed

0.001 (default) | positive scalar

Width of the feed, specified as a positive scalar in meters.

Example: 'FeedWidth',5e-05

Data Types: double

FeedHeight — Height of feed

0.02 (default) | positive scalar

Height of the feed, specified as a positive scalar in meters.

Example: 'FeedHeight',0.0050

Data Types: double

FeedOffset — Signed distance of feedpoint from center of ground plane

[0 0] (default) | two-element vector

Signed distance of the feedpoint from the center of the ground plane, specified as a two-element vector in meters.

Example: 'FeedOffset',[–0.0070 0.01]

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: 'Tilt',90

Example: ant.Tilt = 90

Example: 'Tilt',[90 90],'TiltAxis',[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Rectangular Dielectric Resonator Antenna

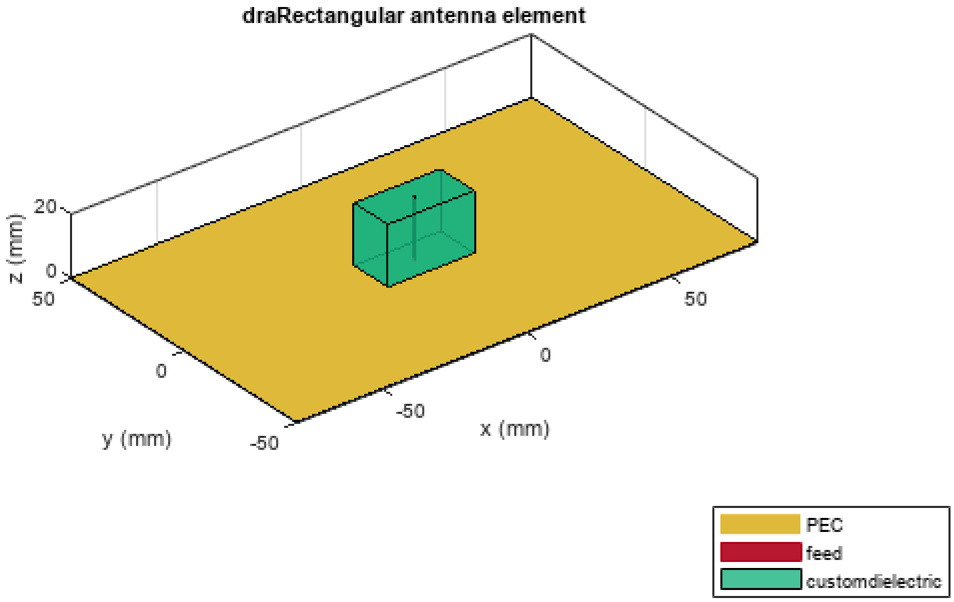
Create a rectangular dielectric resonator antenna with default properties.

```
ant = draRectangular
```

```
ant = 
  draRectangular with properties:

      ResonatorLength: 0.0300
       ResonatorWidth: 0.0150
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1600
     GroundPlaneWidth: 0.1000
            FeedWidth: 1.0000e-03
           FeedHeight: 0.0200
           FeedOffset: [0 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

View the antenna using the `show` function.

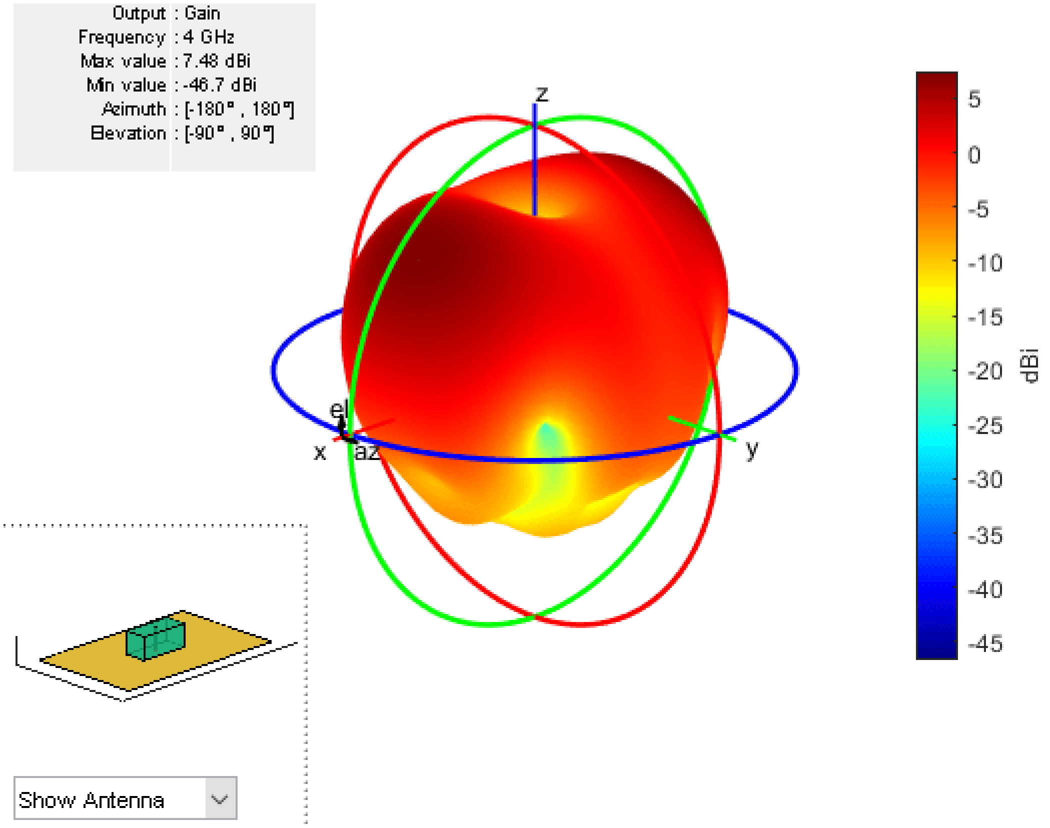
```
show(ant)
```



Radiation Pattern of Rectangular Dielectric Resonator Antenna

Plot the radiation pattern of a rectangular dielectric resonator antenna at a frequency of 4 GHz.

```
ant = draRectangular('ResonatorLength',0.045,'ResonatorWidth',0.02);
pattern(ant,4e9)
```

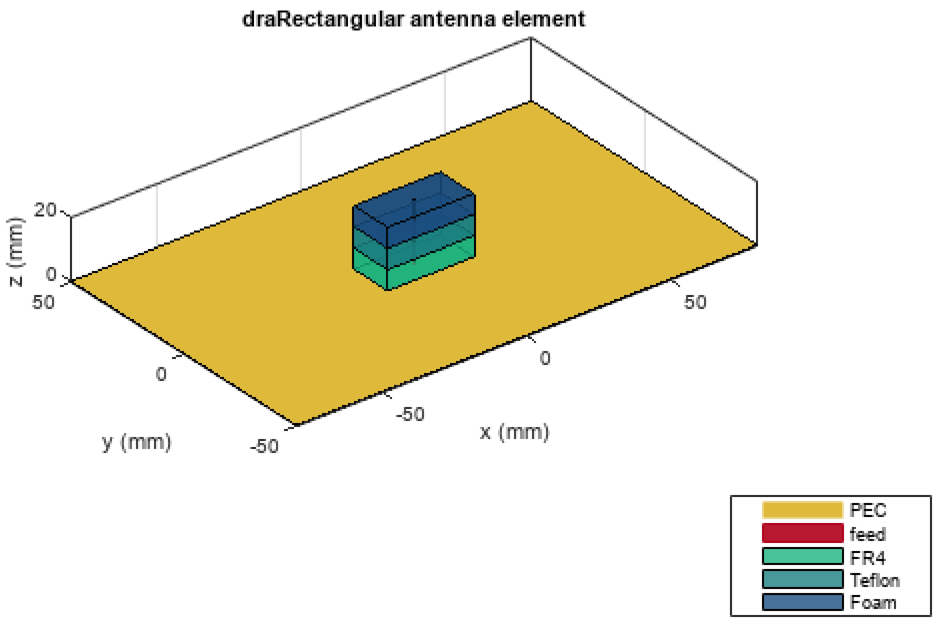


Output : Gain
Frequency : 4 GHz
Max value : 7.48 dBi
Min value : -46.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Rectangular Dielectric Resonator with Multiple Dielectric Substrates

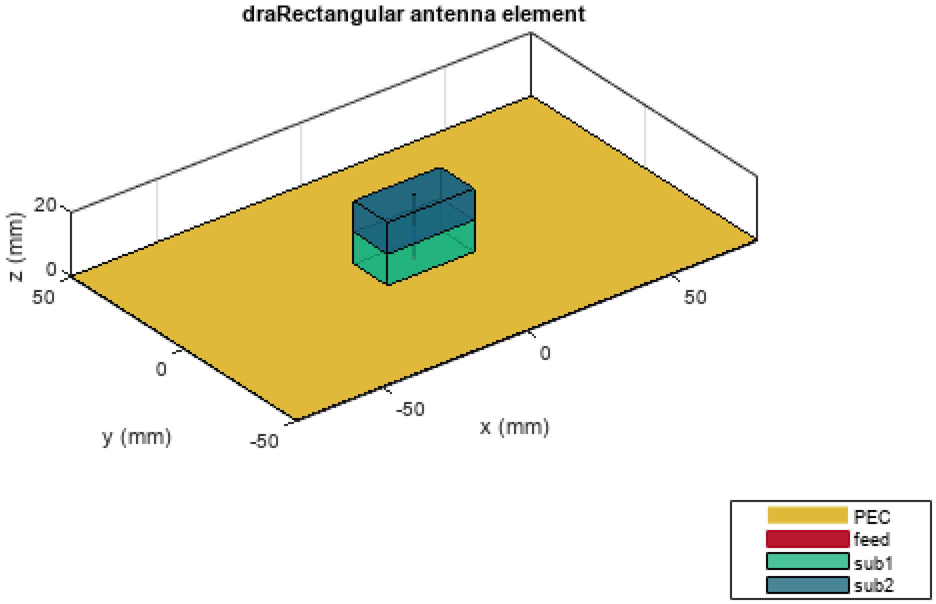
Create a rectangular dielectric resonator antenna with FR4, Teflon, and foam as substrates.

```
ant = draRectangular;
d = dielectric('FR4','Teflon','Foam');
d.Thickness = [ant.Substrate.Thickness/3 ant.Substrate.Thickness/3 ant.Substrate.Thickness/3];
ant.Substrate = d;
ant = draRectangular('Substrate',d);
show(ant)
```



Create a rectangular dielectric resonator antenna with substrates having relative permittivity of 2.3 and 4.5, respectively. The value of loss tangent for both the substrates is 0.002.

```
ant = draRectangular;
d = dielectric;
d.Name = {'sub1','sub2'};
d.EpsilonR = [2.3 4.5];
d.LossTangent = [0.002 0.002];
d.Thickness = [ant.Substrate.Thickness/2 ant.Substrate.Thickness/2];
ant.Substrate = d;
show(ant)
```



More About

Parametric Analysis Guidelines

To understand how the properties of the `draRectangular` antenna object influence the antenna design, use these parametric analysis guidelines.

- To increase the operating frequency, decrease the dimensions of the resonator of the `draRectangular` antenna using the “ResonatorLength” on page 1-0 and “ResonatorWidth” on page 1-0 properties.
- To increase the gain, increase the height of the `draRectangular` antenna object using the “FeedHeight” on page 1-0 property.

Version History

Introduced in R2021a

References

[1] McAllister, M.W., S.A. Long, and G.L. Conway. “Rectangular Dielectric Resonator Antenna.” *Electronics Letters* 19, no. 6 (1983): 218.

See Also

`draCylindrical` | `patchMicrostrip`

Topics

"Rotate Antennas and Arrays"

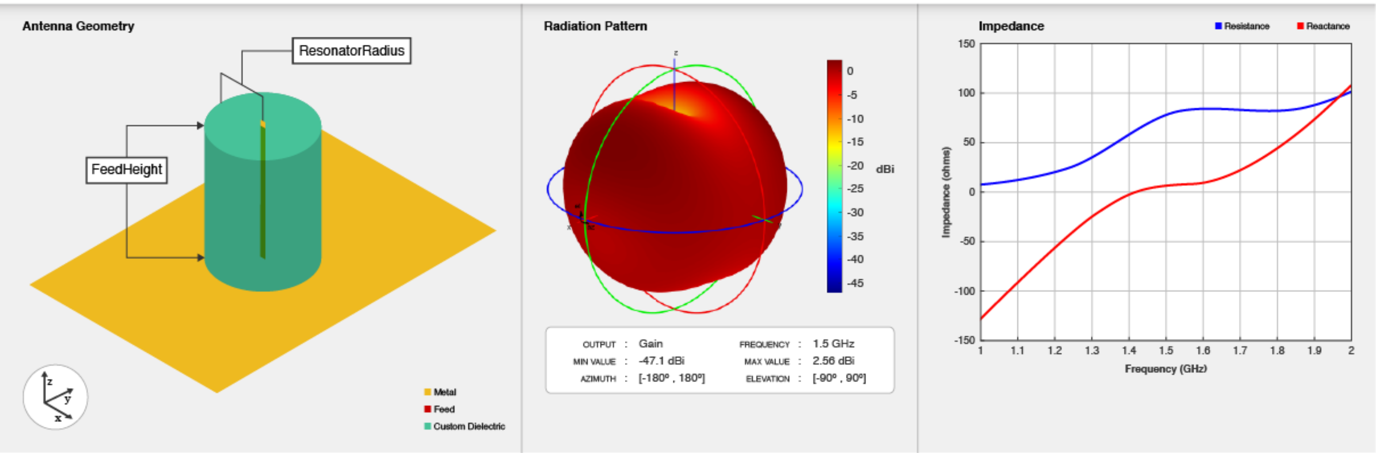
draCylindrical

Create cylindrical dielectric resonator antenna

Description

The `draCylindrical` object creates a cylindrical dielectric resonator antenna. The cylindrical dielectric resonator antenna consists of a cylindrical dielectric placed on the ground plane. It has high power-handling capacity and can provide high gain and bandwidth. These antennas are more suitable for use at microwave frequencies. Cylindrical dielectric resonator antennas are widely used in medium- and long-range communications.

draCylindrical



OUTPUT : Gain
MIN VALUE : -47.1 dBi
AZIMUTH : [-180º , 180º]
FREQUENCY : 1.5 GHz
MAX VALUE : 2.56 dBi
ELEVATION : [-90º , 90º]

Creation

Syntax

```
ant = draCylindrical
ant = draCylindrical(Name,Value)
```

Description

`ant = draCylindrical` creates a cylindrical dielectric resonator antenna with dimensions for a resonant frequency of 1.5 GHz. The default antenna is probe fed with the feedpoint at the origin.

`ant = draCylindrical(Name,Value)` sets “Properties” on page 1-756 using one or more name-value pairs. For example, `draCylindrical('ResonatorRadius',0.04)` creates a cylindrical dielectric resonator antenna with the radius of the dielectric resonator set to 40 mm.

Properties

`ResonatorRadius` — Radius of dielectric resonator

`0.02` (default) | positive scalar

Radius of the dielectric resonator, specified as a positive scalar in meters.

Example: `'ResonatorRadius',0.05`

Data Types: `double`

`Substrate` — Type of dielectric material
dielectric with `EpsilonR` 6, `LossTangent` 0.002 (default) | `dielectric` function

Type of dielectric material used as a substrate, specified as a dielectric material object. You can choose any material from the `DielectricCatalog` or use your own dielectric material. For more information, see `dielectric`. For more information on dielectric substrate meshing, see “Meshing”.

Note The substrate dimensions must be lesser than the ground plane dimensions.

Example: `d = dielectric('FR4'); 'Substrate',d`

Example: `d = dielectric; d.Name = 'sub1'; d.EpsilonR = 2.3; d.LossTangent = 0.002; d.Thickness = 0.01; ant.Substrate = d;`

`GroundPlaneLength` — Ground plane length along x-axis
`0.14` (default) | positive scalar

Ground plane length, specified as a positive scalar in meters. By default, ground plane length is measured along the x-axis. Set `'GroundPlaneLength'` to `Inf` to use the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneLength',120e-3`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along y-axis
`0.08` (default) | positive scalar

Ground plane width, specified as a positive scalar in meters. By default, ground plane width is measured along the y-axis. Set `'GroundPlaneWidth'` to `Inf` to use the infinite ground plane technique for antenna analysis.

Example: `'GroundPlaneWidth',118e-3`

Data Types: `double`

`FeedWidth` — Width of feed
`0.001` (default) | positive scalar

Width of the feed, specified as a positive scalar in meters.

Example: `'FeedWidth',5e-05`

Data Types: `double`

`FeedHeight` — Height of feed
`0.05` (default) | positive scalar

Height of the feed, specified as a positive scalar in meters.

Example: `'FeedHeight',0.060`

Data Types: `double`

`FeedOffset` — Signed distance of feedpoint from center of ground plane

`[0 0]` (default) | two-element vector

Signed distance of the feedpoint from the center of the ground plane, specified as a two-element vector in meters.

Example: `'FeedOffset',[–0.0070 0.01]`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements
[1x1 lumpedElement] (default) | lumpedElement object

Lumped elements added to the antenna feed, specified as a lumpedElement object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see lumpedElement.

Example: 'Load',lumpedelement, where lumpedelement is the load added to the antenna feed.

Example: ant.Load = lumpedElement('Impedance',75)

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Cylindrical Dielectric Resonator Antenna and Plot Radiation Pattern

Create a cylindrical dielectric resonator antenna with default properties.

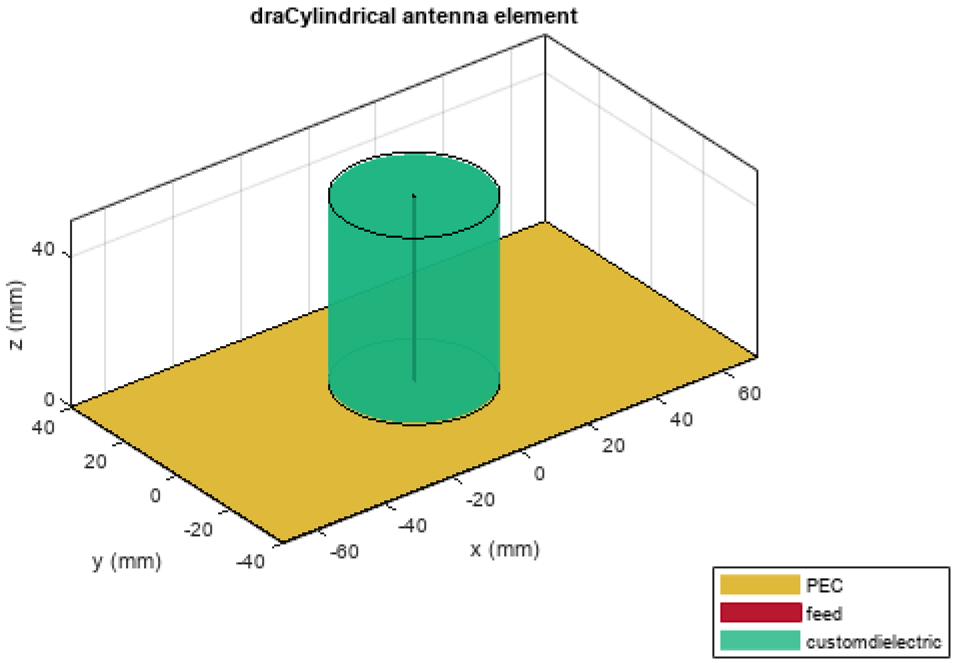
```
ant = draCylindrical
```

```
ant = 
  draCylindrical with properties:

      ResonatorRadius: 0.0200
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1400
     GroundPlaneWidth: 0.0800
            FeedWidth: 1.0000e-03
           FeedHeight: 0.0500
           FeedOffset: [0 0]
            Conductor: [1x1 metal]
```

```
        Tilt: 0
    TiltAxis: [1 0 0]
        Load: [1x1 lumpedElement]
```

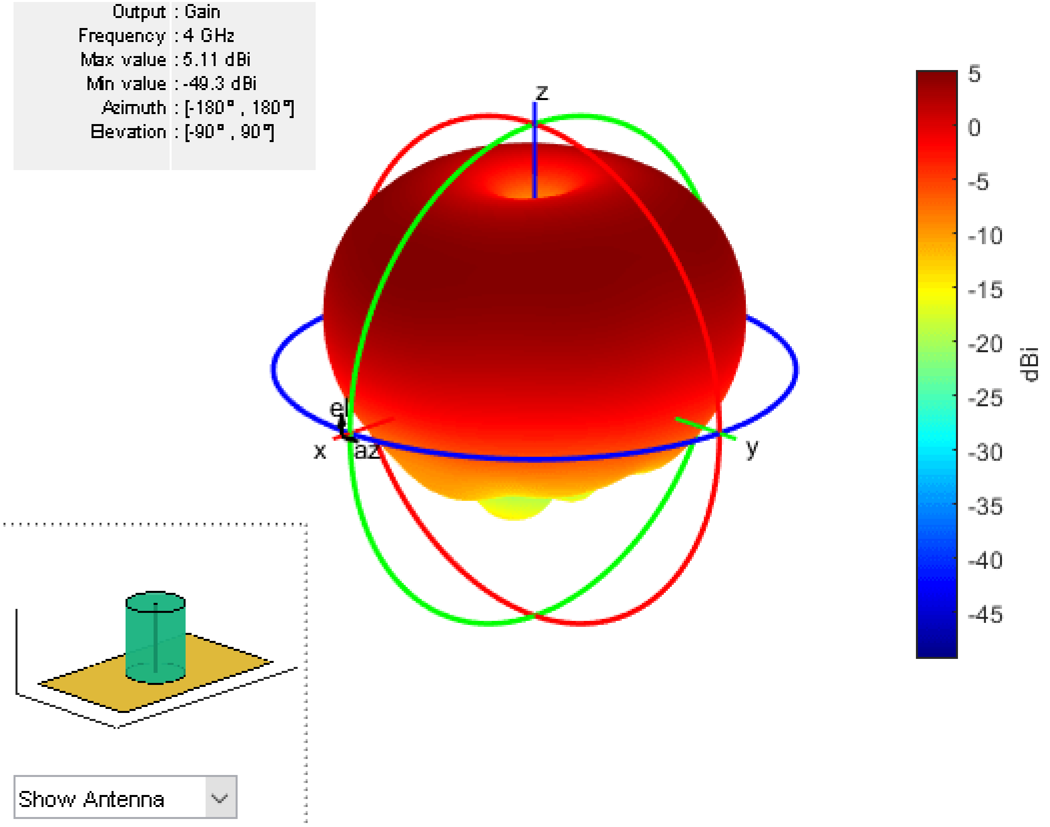
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of the cylindrical dielectric resonator antenna at a frequency of 4 GHz.

```
pattern(ant,4e9)
```

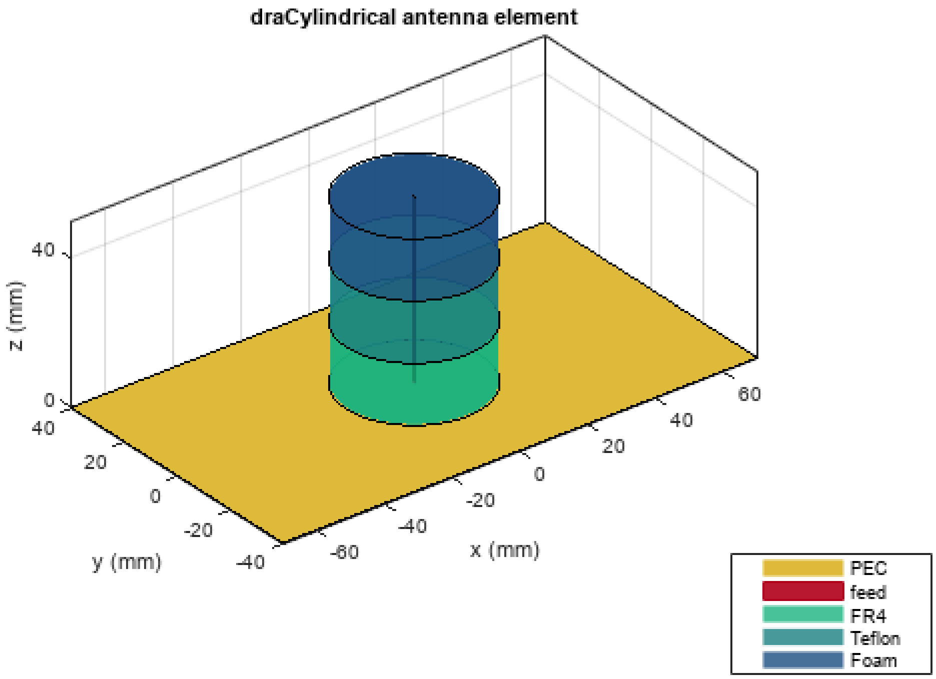


Output : Gain
Frequency : 4 GHz
Max value : 5.11 dBi
Min value : -49.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Cylindrical Dielectric Resonator with Multiple Dielectric Substrates

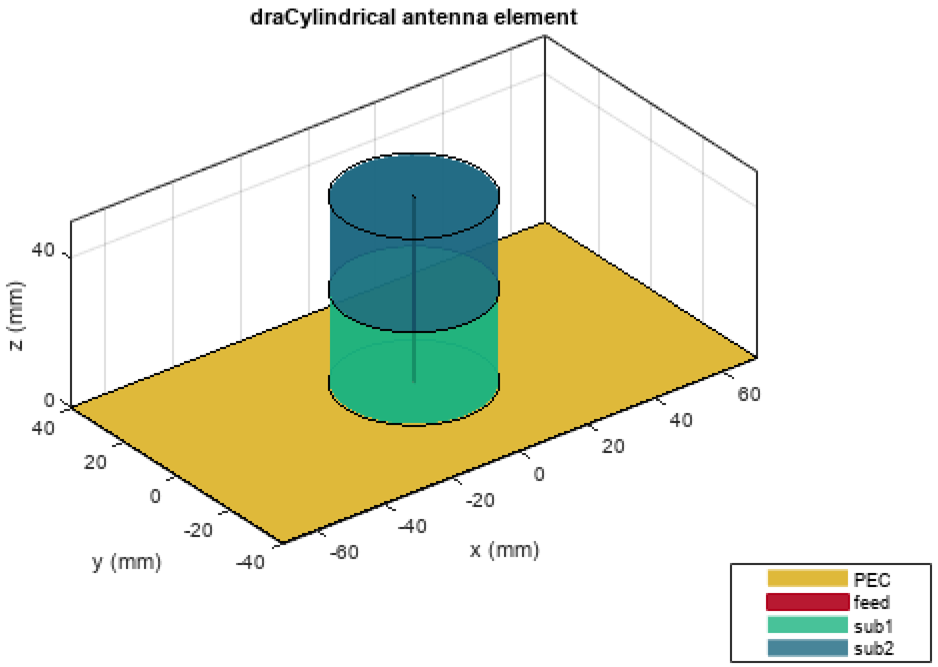
Create a cylindrical dielectric resonator antenna with FR4, Teflon, and foam as substrates.

```
ant = draCylindrical;
d = dielectric('FR4','Teflon','Foam');
d.Thickness = [ant.Substrate.Thickness/3 ant.Substrate.Thickness/3 ant.Substrate.Thickness/3];
ant.Substrate = d;
ant = draCylindrical('Substrate',d);
show(ant)
```



Create a cylindrical dielectric resonator antenna with substrates having relative permittivity as 2.3 and 4.5, respectively. The value of loss tangent for both the substrates is 0.002.

```
ant = draCylindrical;
d = dielectric;
d.Name = {'sub1','sub2'};
d.EpsilonR = [2.3 4.5];
d.LossTangent = [0.002 0.002];
d.Thickness = [ant.Substrate.Thickness/2 ant.Substrate.Thickness/2];
ant.Substrate = d;
show(ant)
```



More About

Parametric Analysis Guidelines

To understand how the properties of the `draCylindrical` antenna object influence the antenna design, use these parametric analysis guidelines.

- To increase the operating frequency, decrease the dimensions of the resonator of the `draCylindrical` antenna using the “ResonatorRadius” on page 1-0 property.
- To increase the gain, select proper feed location using the “FeedOffset” on page 1-0 property and increase the height of the `draCylindrical` antenna object using the “FeedHeight” on page 1-0 property.

Version History

Introduced in R2021a

References

[1] Keyrouz, S., and D. Caratelli. “Dielectric Resonator Antennas: Basic Concepts, Design Guidelines, and Recent Developments at Millimeter-Wave Frequencies.” *International Journal of Antennas and Propagation* 2016 (2016): 1–20.

See Also

`draRectangular` | `patchMicrostrip`

Topics

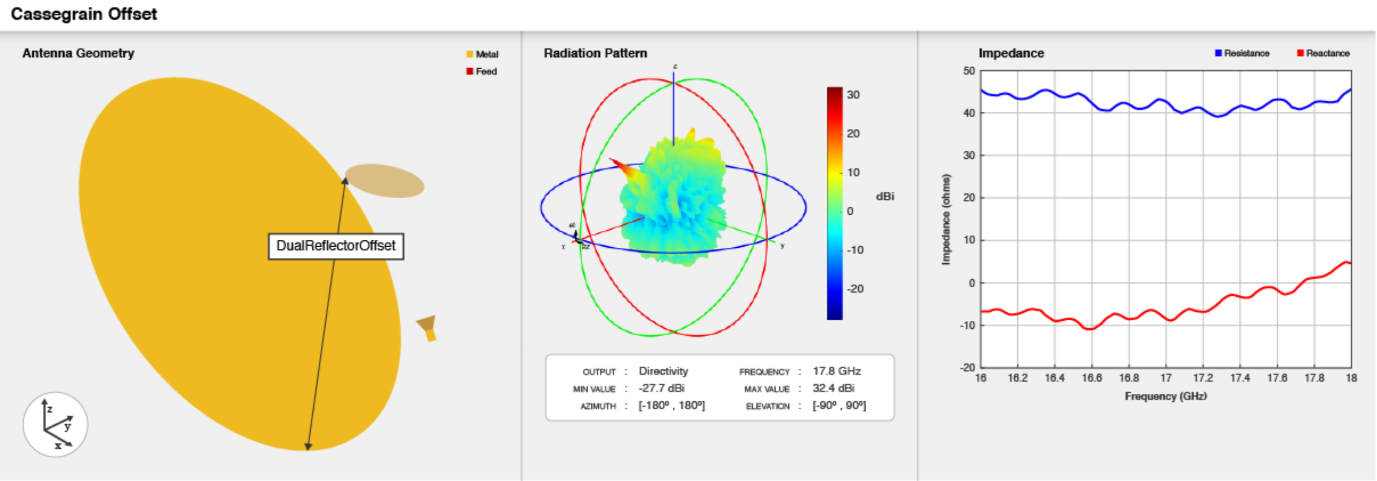
"Rotate Antennas and Arrays"

cassegrainOffset

Create offset Cassegrain antenna

Description

The `cassegrainOffset` object creates an offset Cassegrain antenna. The offset Cassegrain antenna is a parabolic antenna, where the feed antenna is mounted off-axis to convex sub reflector and concave main reflector. The asymmetric arrangement of reflectors provides less blockage for waves redirected from main reflector. The advantage of these antennas is high gain, reduced sidelobes and improved cross polarization. The offset Cassegrain antennas are used in satellite communication ground antennas, radar systems, and, radio telescopes among other applications.



OUTPUT : Directivity
MIN VALUE : -27.7 dBi
AZIMUTH : [-180° , 180°]
FREQUENCY : 17.8 GHz
MAX VALUE : 32.4 dBi
ELEVATION : [-90° , 90°]

Creation

Syntax

```
ant = cassegrainOffset
ant = cassegrainOffset(Name,Value)
```

Description

`ant = cassegrainOffset` creates a conical-horn-fed offset Cassegrain antenna with dimensions for a resonant frequency of 17.8 GHz.

`ant = cassegrainOffset(Name,Value)` sets “Properties” on page 1-766 using one or more name-value pairs. For example, `ant = cassegrainOffset('FocalLength', 0.04)` creates an offset Cassegrain antenna with the focal length of the main reflector set to 40 mm.

Properties

`Exciter` — Antenna or array type used as exciter
`hornConical` (default) | `antenna` object | `array` object

Antenna type used as exciter, specified as an `antenna` or an `array` object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole`

Example: `ant.Exciter = rectangularArray('invertedF')`

`Radius` — Radius of main and sub reflector
`[0.3175 0.0330]` (default) | two-element vector

Radius of the main and sub reflector, specified as a two-element vector with each element unit in meters. The first element specifies the radius of the main reflector, and the second element specifies the radius of the sub reflector.

Example: `'Radius',[0.4 0.2]`

Example: `ant.Radius = [0.4 0.2]`

Data Types: `double`

`FocalLength` — Focal length of main reflector
`0.5` (default) | positive scalar

Focal length of the main reflector, specified as a positive scalar in meters.

Example: `'FocalLength',0.0850`

Example: `ant.FocalLength = 0.0850`

Data Types: `double`

`MainReflectorOffset` — Distance of center of main reflector along *x*-axis from x = 0
`0.5` (default) | positive scalar

The distance between the main reflector and x=0 along X-axis, specified as a positive scalar in meters.

Example: `'MainReflectorOffset',0.8`

Example: `ant.MainReflectorOffset = 0.8`

Data Types: `double`

`DualReflectorSpacing` — Distance between bottom edge of main reflector and top edge of sub reflector along X-axis
`0.035` (default) | positive scalar

The distance between the bottom edge of the main reflector and the top edge of the sub reflector along *x* -axis, specified as a positive scalar in meters.

Example: `'DualReflectorSpacing',0.8`

Example: `ant.DualReflectorSpacing = 0.8`

Data Types: `double`

`InterAxialAngle` — Angle between main reflector and sub reflector coordinate systems

`5` (default) | positive integer

Angle between the main reflector and the sub reflector coordinate systems, specified as a positive integer in degrees.

Example: `'InterAxialAngle',8`

Example: `ant.InterAxialAngle = 8`

Data Types: `double`

`ReflectorTilt` — Tilt angle of reflectors

`[53.13 11.37]` (default) | two-element vector

Tilt angle of the reflectors, specified as a two-element vector with each element unit in degrees. The first element specifies the tilt of the main reflector, and the second element specifies the tilt of the sub reflector.

Note You can use “BasisReflectorTilt” on page 1-774 to obtain initial value of tilt angles of reflectors with respect to reflector dimensions.

Example: `'ReflectorTilt',[60 20]`

Example: `ant.ReflectorTilt = [60 20]`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the *x*-, *y*-, and *z*-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

`SolverType` — Solver for antenna analysis
`'MoM-PO'` (default) | `'MoM'` | `'FMM'`

Solver for antenna analysis, specified as the comma-separated pair consisting of `'SolverType'` and `'MoM-PO'` or `'MoM'` (Method of Moments) or `'FMM'` (Fast Multipole Method).

Example: `'SolverType','MOM'`

Data Types: `char`

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Offset Cassegrain Dual-Reflector Antenna

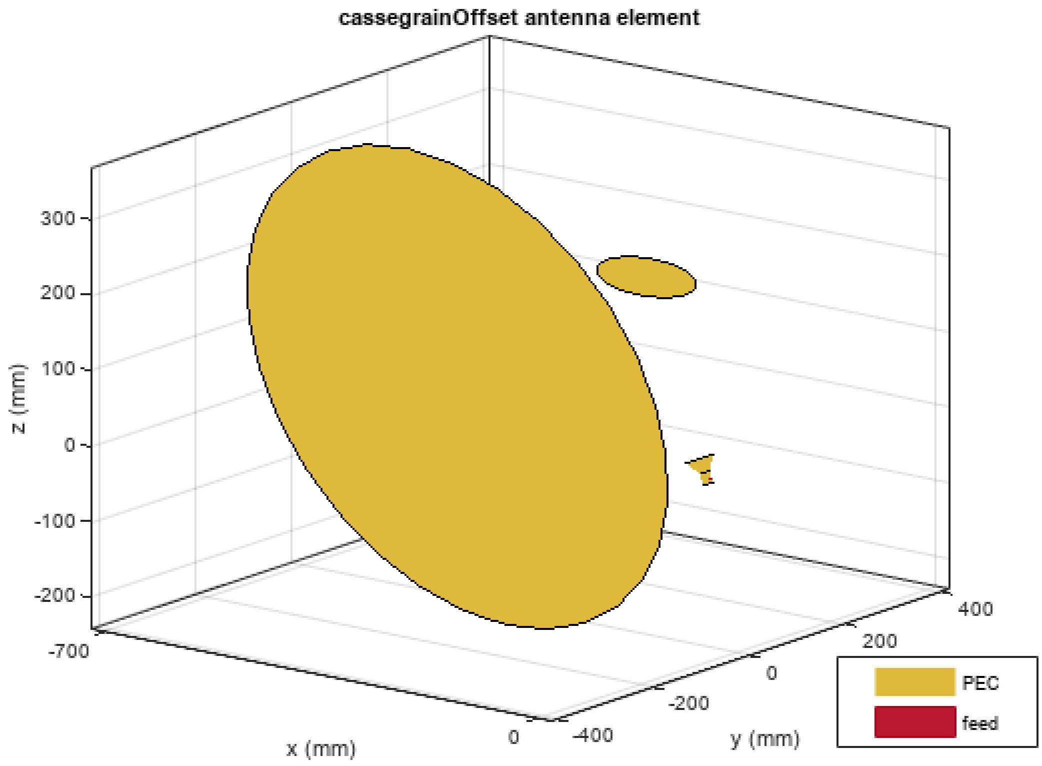
Create an offset Cassegrain dual-reflector antenna with default properties.

```
ant = cassegrainOffset
```

```
ant = 
  cassegrainOffset with properties:

                  Exciter: [1x1 hornConical]
                   Radius: [0.3475 0.0650]
              FocalLength: 0.5000
      MainReflectorOffset: 0.5000
          InterAxialAngle: 5
    DualReflectorSpacing: 0.0350
            ReflectorTilt: [53.1300 11.3700]
                     Tilt: 0
                 TiltAxis: [1 0 0]
                     Load: [1x1 lumpedElement]
               SolverType: 'MoM-PO'
```

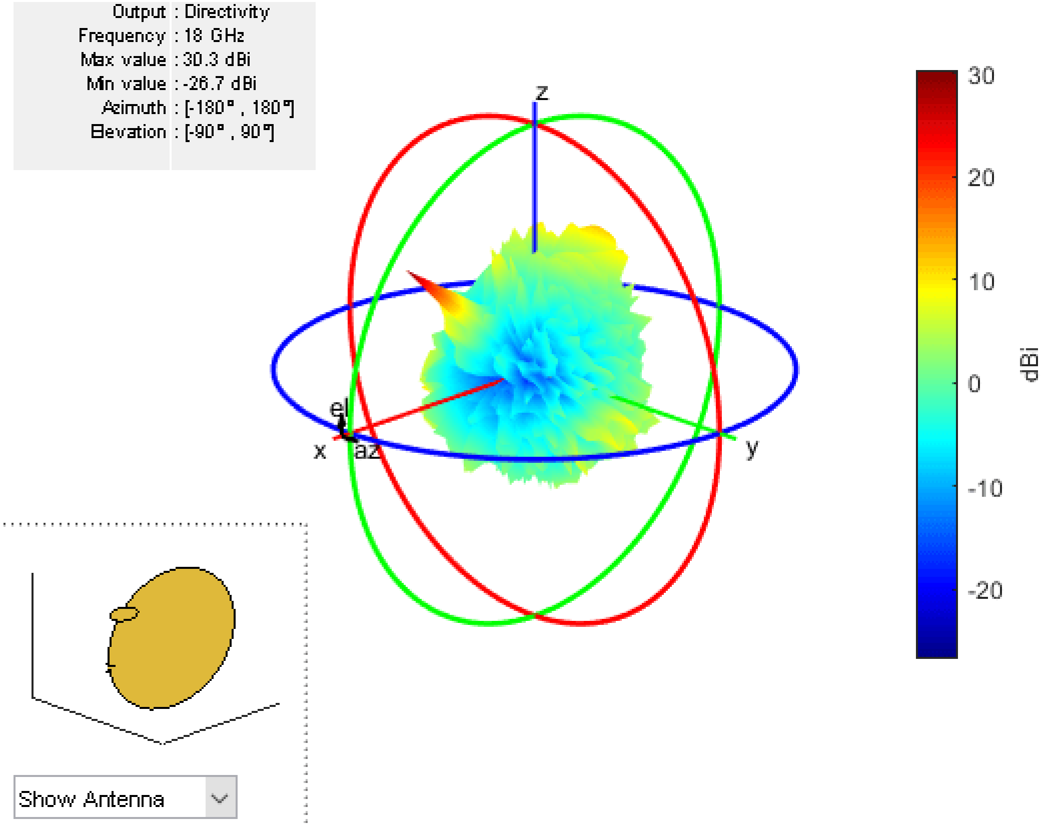
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of offset Cassegrain dual-reflector antenna at a frequency of 18 GHz.

```
pattern(ant,18e9)
```



Calculate Impedance of Offset Cassegrain Antenna

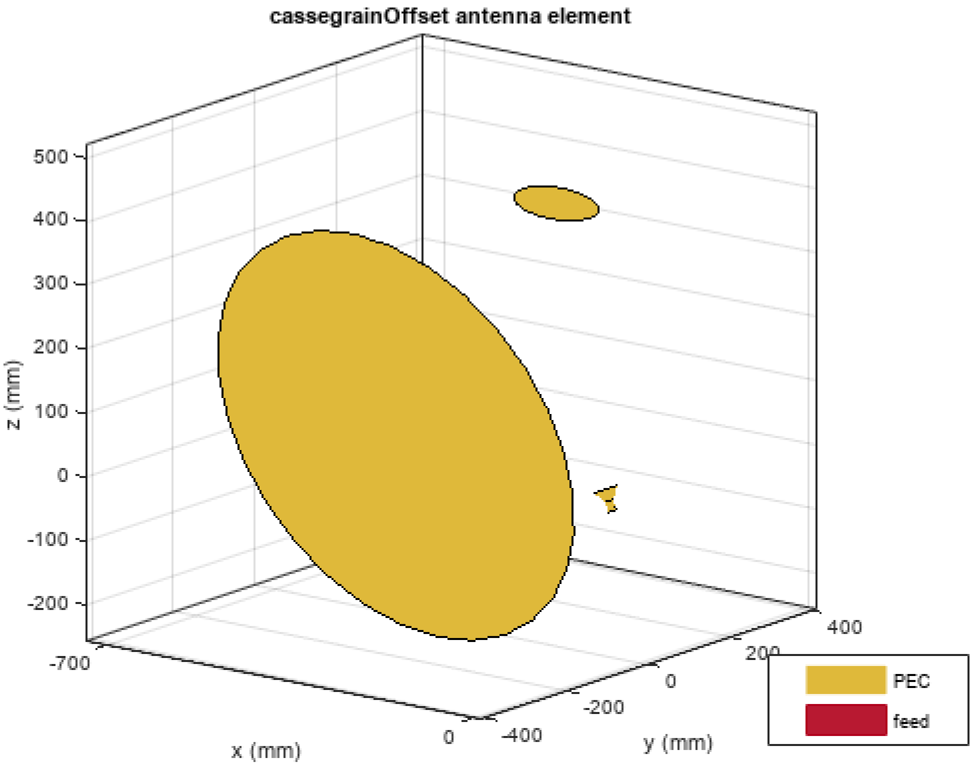
Create and view offset Cassegrain antenna with optimum reflector tilt angles and with a focal length of 0.8 meters and an interaxial angle of 5 degrees.

```
ant = cassegrainOffset('InterAxialAngle',5,'FocalLength',0.8)
```

```
ant = 
  cassegrainOffset with properties:

                Exciter: [1x1 hornConical]
                 Radius: [0.3475 0.0650]
            FocalLength: 0.8000
    MainReflectorOffset: 0.5000
        InterAxialAngle: 5
   DualReflectorSpacing: 0.0350
          ReflectorTilt: [53.1300 11.3700]
                   Tilt: 0
               TiltAxis: [1 0 0]
                   Load: [1x1 lumpedElement]
             SolverType: 'MoM-PO'
```

View the antenna using the `show` function.

```
show(ant)
```



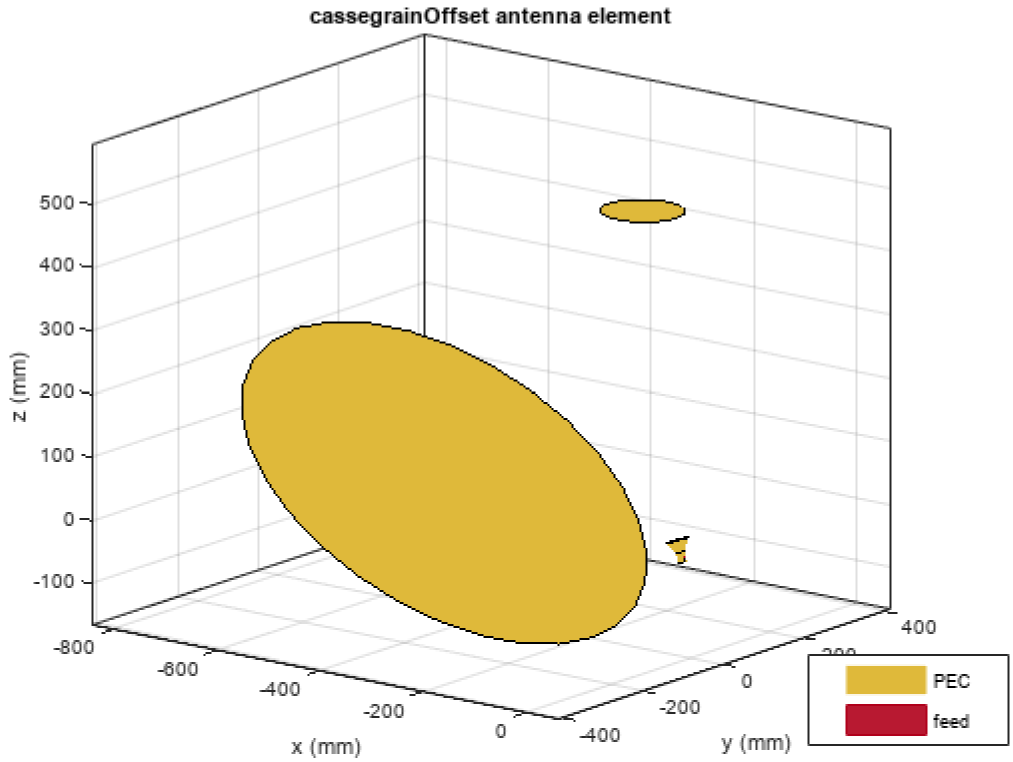
View offset cassegrain antenna with optimum reflector tilt angles.

```
ant.ReflectorTilt = ant.BasisReflectorTilt
```

```
ant = 
  cassegrainOffset with properties:

                 Exciter: [1x1 hornConical]
                  Radius: [0.3475 0.0650]
             FocalLength: 0.8000
     MainReflectorOffset: 0.5000
         InterAxialAngle: 5
    DualReflectorSpacing: 0.0350
           ReflectorTilt: [34.7080 0.9748]
                    Tilt: 0
                TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
              SolverType: 'MoM-PO'
```

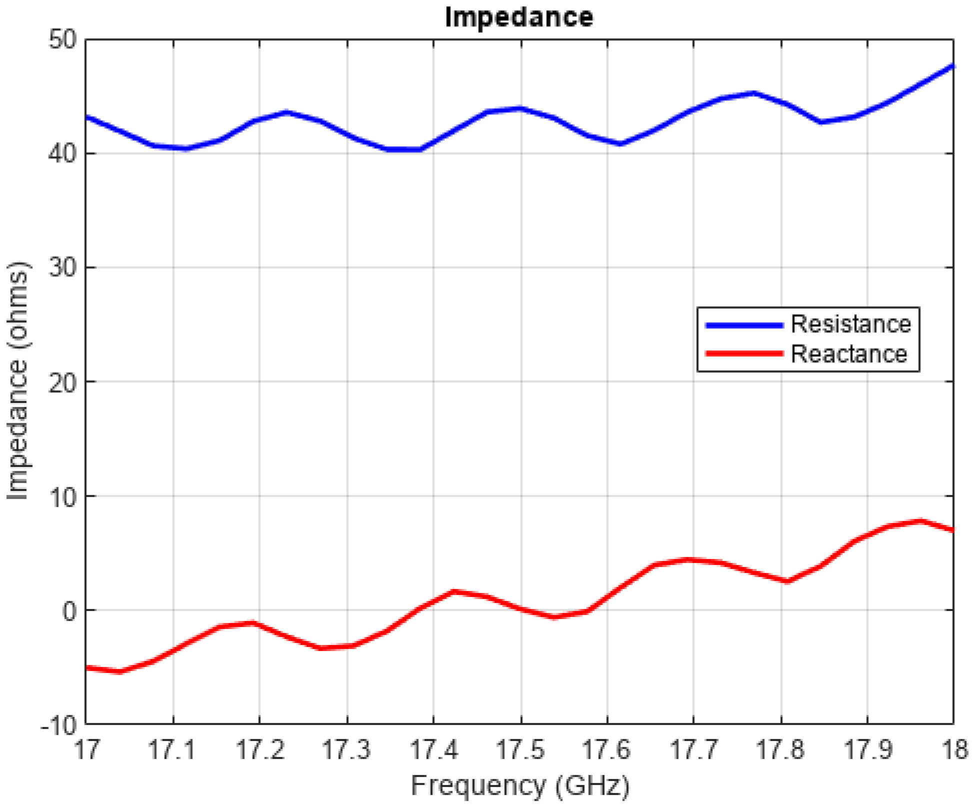
View the antenna using the `show` function.

```
show(ant)
```



Calculate the impedance of the antenna over a frequency span 17 GHz - 18 GHz.

```
impedance(ant,linspace(17e9,18e9,27))
```



Create Array-Fed Offset Cassegrain Antenna

Create a linear array of bowtie antennas.

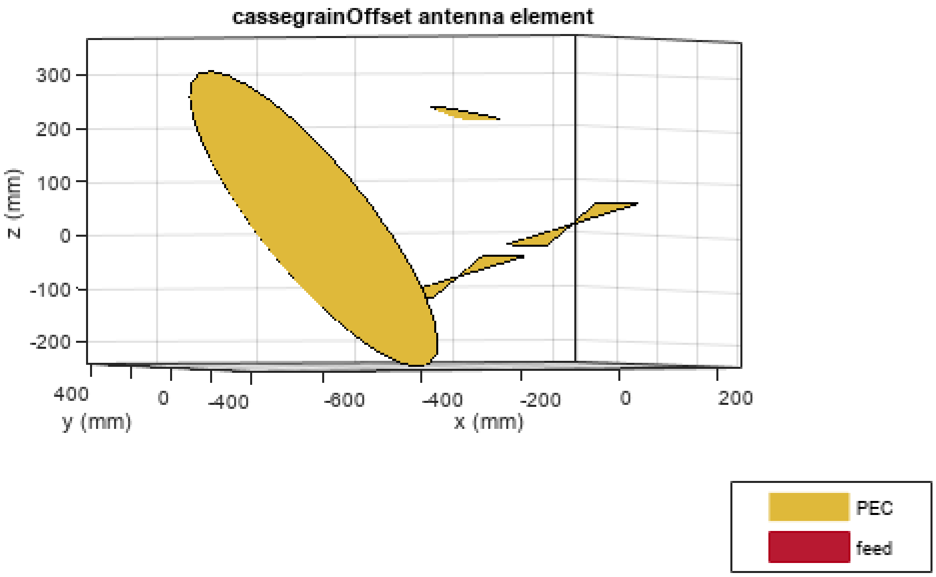
```
e = bowtieTriangular('Tilt',90,'TiltAxis',[0 1 0]);
arr = linearArray('Element',e,'ElementSpacing',0.25);
```

Create an offset Cassegrain antenna with the linear array as the exciter.

```
ant = cassegrainOffset('Exciter',arr)
```

```
ant = 
  cassegrainOffset with properties:

                Exciter: [1x1 linearArray]
                 Radius: [0.3475 0.0650]
            FocalLength: 0.5000
    MainReflectorOffset: 0.5000
         InterAxialAngle: 5
   DualReflectorSpacing: 0.0350
           ReflectorTilt: [53.1300 11.3700]
                    Tilt: 0
               TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
             SolverType: 'MoM-PO'
```

```
show(ant)
view([-22 1])
```



More About

BasisReflectorTilt

Calculates tilt angles of the reflectors as a starting parameter for the given reflector dimensions, specified as a two-element vector with each element unit in degrees. The first element specifies the tilt angle of the main reflector, and the second element specifies the tilt angle of the sub reflector.

Example: `ant.ReflectorTilt = ant.BasisReflectorTilt`

Parametric Analysis Guidelines

To understand how the properties of the `cassegrainOffset` antenna object influence the antenna design, use the following parametric analysis guidelines.

- To increase the operating frequency, use “Radius” on page 1-0 decrease the radius of the reflectors of the `cassegrainOffset` antenna.
- To increase the gain, select `hornCorrugated` or `hornConicalCorrugated` as the exciter for the `cassegrainOffset` antenna object.
- Gain variation depends on “InterAxialAngle” on page 1-0 and “DualReflectorSpacing” on page 1-0 properties.

- Use property `BasisReflectorTilt` (read only) to calculate theoretical "ReflectorTilt" on page 1-0 value.

Version History

Introduced in R2021a

References

[1] Granet, C. "Designing Classical Offset Cassegrain or Gregorian Dual-Reflector Antennas from Combinations of Prescribed Geometric Parameters." *IEEE Antennas and Propagation Magazine* 44, no. 3 (June 2002): 114–123.

See Also

`gregorianOffset` | `cassegrain` | `gregorian`

Topics

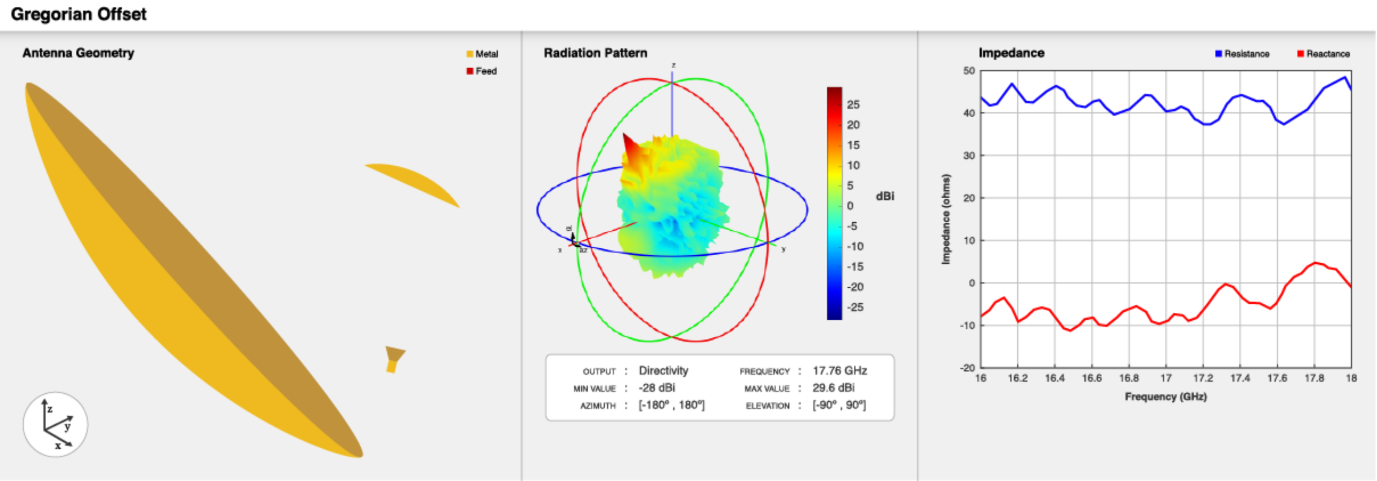
"Rotate Antennas and Arrays"

gregorianOffset

Create offset Gregorian antenna

Description

The `gregorianOffset` object creates a offset Gregorian antenna. The offset Gregorian antenna is a parabolic antenna. It consists of feed antenna mounted off-axis to concave sub reflector and concave main reflector. The asymmetric arrangement of reflectors provides less blockage for waves redirected from main reflector. The advantage of these antennas is high gain, to reduce side-lobes and to improve cross polarization. The offset Gregorian antennas are used in satellite communication ground antennas, radar systems, and, radio telescopes, etc.



OUTPUT : Directivity
MIN VALUE : -28 dBi
AZIMUTH : [-180° , 180°]
FREQUENCY : 17.76 GHz
MAX VALUE : 29.6 dBi
ELEVATION : [-90° , 90°]

Creation

Syntax

```
ant = gregorianOffset
ant = gregorianOffset(Name,Value)
```

Description

`ant = gregorianOffset` creates a conical horn fed offset Gregorian antenna with dimensions for a resonant frequency of 17.76 GHz.

`ant = gregorianOffset(Name,Value)` sets “Properties” on page 1-777 using one or more name-value pairs. For example, `ant = gregorianOffset('FocalLength', 0.04)` creates an offset Gregorian antenna with the focal length of main reflector set to 40 mm.

Properties

`Exciter` — Antenna type used as exciter
`hornConical` (default) | `antenna` object | `array` object

Antenna type used as exciter, specified as an antenna or an array object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole`

Example: `ant.Exciter = rectangularArray('invertedF')`

`Radius` — Radius of main and sub reflector
`[0.3 0.06]` (default) | two-element vector

Radius of the main and sub reflector, specified as a two-element vector with each element unit in meters. The first element specifies the radius of the main reflector, and the second element specifies the radius of the sub reflector.

Example: `'Radius',[0.4 0.2]`

Example: `ant.Radius = [0.4 0.2]`

Data Types: `double`

`FocalLength` — Focal distance of main reflector
`0.245` (default) | positive scalar

Focal length of the main reflector, specified as a positive scalar integer in meters.

Example: `'FocalLength',0.0850`

Example: `ant.FocalLength = 0.0850`

Data Types: `double`

`MainReflectorOffset` — Distance of center of main reflector along X-axis from x = 0
`0.26` (default) | positive scalar

The distance between the main reflector and x=0 along X-axis, specified as a positive scalar integer in meters.

Example: `'MainReflectorOffset',0.8`

Example: `ant.MainReflectorOffset = 0.8`

Data Types: `double`

`DualReflectorSpacing` — Spacing between bottom edge of main reflector and top edge of sub reflector along X-axis
`0.045` (default) | positive scalar

The spacing between the bottom edge of the main reflector and the top edge of the sub reflector along X-axis, specified as a positive scalar integer in meters.

Example: `'DualReflectorSpacing',0.8`

Example: `ant.DualReflectorSpacing = 0.8`

Data Types: `double`

`InterAxialAngle` — Angle between main reflector and sub reflector co-ordinate systems

`15` (default) | positive scalar

Angle between the main reflector and the sub reflector co-ordinate systems, specified as a positive scalar integer in degrees.

Example: `'InterAxialAngle',8`

Example: `ant.InterAxialAngle = 8`

Data Types: `double`

`ReflectorTilt` — Tilt angle of reflectors

`[55.9 31.6]` (default) | two-element vector

Tilt angle of the reflectors, specified as a two-element vector with each element unit in degrees. The first element specifies the tilt of the main reflector, and the second element specifies the tilt of the sub reflector.

Note You can use property "BasisReflectorTilt" on page 1-782 to obtain initial value of tilt angles of reflectors with respect to reflector dimensions.

Example: `'ReflectorTilt',[60 20]`

Example: `ant.ReflectorTilt = [60 20]`

Data Types: `double`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: 'TiltAxis',[0 1 0]

Example: 'TiltAxis',[0 0 0;0 1 0]

Example: ant.TiltAxis = 'Z'

Load — Lumped elements
[1x1 lumpedElement] (default) | lumpedElement object

Lumped elements added to the antenna feed, specified as a lumpedElement object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see lumpedElement.

Example: 'Load',lumpedelement, where lumpedelement is the load added to the antenna feed.

Example: ant.Load = lumpedElement('Impedance',75)

SolverType — Solver for antenna analysis
'MoM-PO' (default) | 'MoM' | 'FMM'

Solver for antenna analysis, specified as the comma-separated pair consisting of 'SolverType' and 'MoM-PO' or 'MoM' (Method of Moments) or 'FMM' (Fast Multipole Method).

Example: 'SolverType','MOM'

Data Types: char

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Offset Gregorian Dual-Reflector Antenna

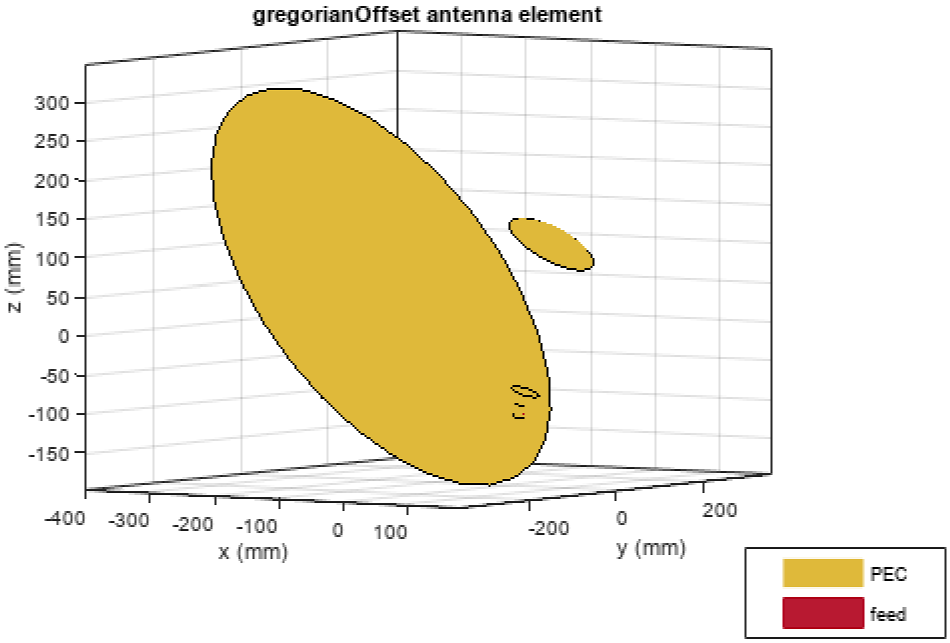
Create an offset Gregorian dual-reflector antenna with default properties.

```
ant = gregorianOffset
```

```
ant = 
  gregorianOffset with properties:

                  Exciter: [1x1 hornConical]
                   Radius: [0.3000 0.0600]
              FocalLength: 0.2450
      MainReflectorOffset: 0.2600
          InterAxialAngle: 15
     DualReflectorSpacing: 0.0450
            ReflectorTilt: [55.9000 31.6000]
                     Tilt: 0
                 TiltAxis: [1 0 0]
                     Load: [1x1 lumpedElement]
               SolverType: 'MoM-PO'
```

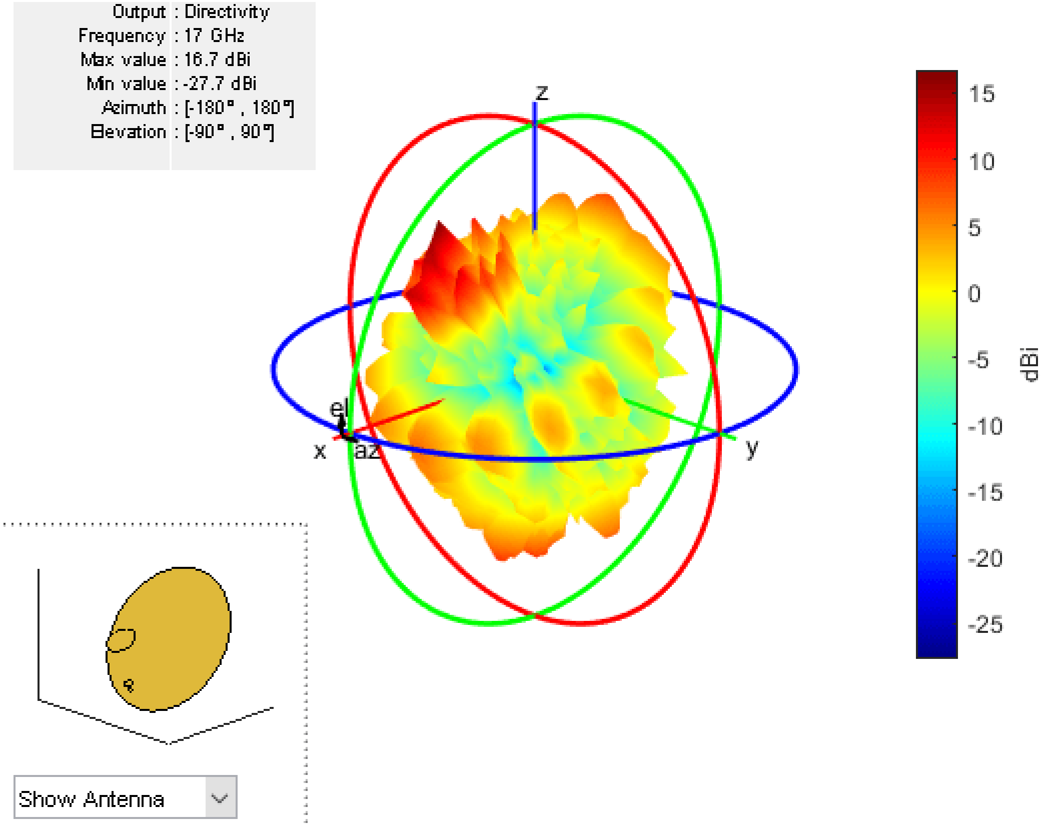
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of the offset Gregorian dual-reflector antenna at a frequency of 17 GHz.

```
pattern(ant,17e9)
```



Output : Directivity
Frequency : 17 GHz
Max value : 16.7 dBi
Min value : -27.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Array-Fed Offset Gregorian Antenna

Create a circular array of rectangular spiral antennas.

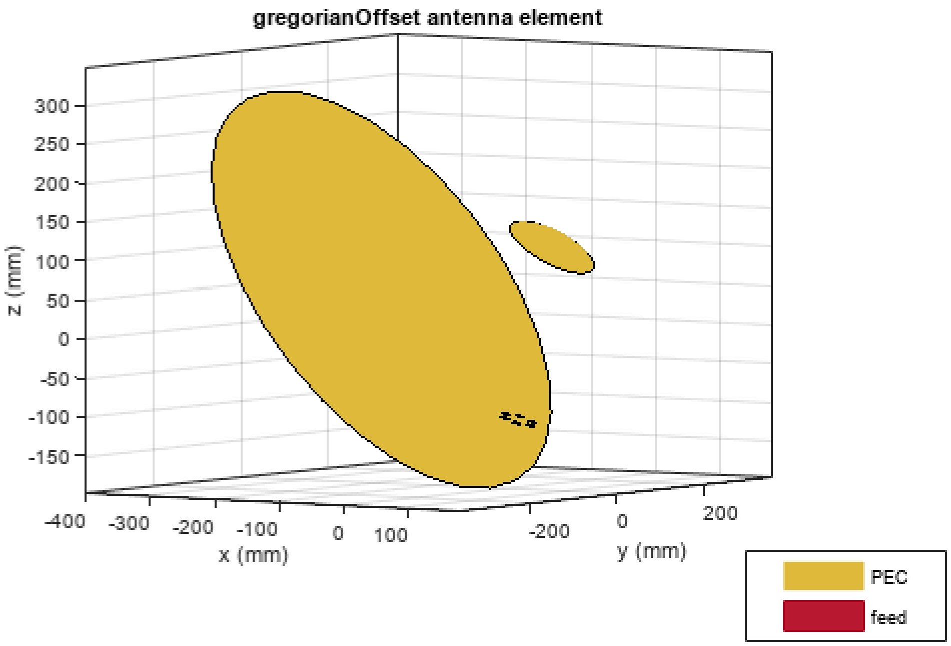
```
e = spiralRectangular;
arr = circularArray('Element',e,'Radius',0.02);
```

Create an offset Gregorian antenna with the circular array as the exciter.

```
ant = gregorianOffset('Exciter',arr)
```

```
ant = 
  gregorianOffset with properties:

                Exciter: [1x1 circularArray]
                 Radius: [0.3000 0.0600]
            FocalLength: 0.2450
    MainReflectorOffset: 0.2600
        InterAxialAngle: 15
   DualReflectorSpacing: 0.0450
          ReflectorTilt: [55.9000 31.6000]
                   Tilt: 0
               TiltAxis: [1 0 0]
                   Load: [1x1 lumpedElement]
             SolverType: 'MoM-PO'
```

```
show(ant)
```



More About

BasisReflectorTilt

Calculates tilt angles of the reflectors as a starting parameter for the given reflector dimensions, specified as a two-element vector with each element unit in degrees. The first element specifies the tilt angle of the main reflector, and the second element specifies the tilt angle of the sub reflector.

Example: `ant.ReflectorTilt = ant.BasisReflectorTilt`

Parametric Analysis Guidelines

To understand how the properties of the `gregorianOffset` antenna object influence the antenna design, use the following parametric analysis guidelines.

- To increase the operating frequency, use “Radius” on page 1-0 decrease the radius of the reflectors of the `gregorianOffset` antenna.
- To increase the gain, select `hornCorrugated` or `hornConicalCorrugated` as the exciter for the `gregorianOffset` antenna object.
- Gain variation depends on “InterAxialAngle” on page 1-0 and “DualReflectorSpacing” on page 1-0 properties.

- Use property `OptimumReflectorTilt` (read only) to calculate theoretical “ReflectorTilt” on page 1-0 value.

Version History

Introduced in R2021a

References

[1] Granet, C. “Designing Classical Offset Cassegrain or Gregorian Dual-Reflector Antennas from Combinations of Prescribed Geometric Parameters.” *IEEE Antennas and Propagation Magazine* 44, no. 3 (June 2002): 114–123.

See Also

`cassegrainOffset` | `cassegrain` | `gregorian`

Topics

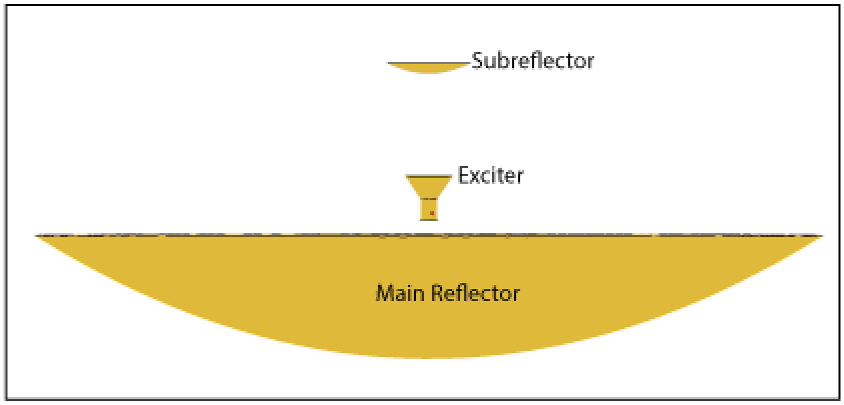
“Rotate Antennas and Arrays”

customDualReflectors

Create custom dual-reflector antenna

Description

The `customDualReflectors` object creates a dual-reflector antenna with empty geometries for the main and sub reflectors and the `hornConical` element as the default exciter. Once you create the object, you have to specify the geometry coordinates of the main and sub reflector surfaces in *N*-by-3 matrices and assign them to the `MainReflector` and `SubReflector` properties of the object before using the `show` function to view the antenna. *N* represents the total number of points to use for defining the geometry. Value of *N* can be different for the main and the sub reflector. Alternatively, you can also use a `triangulation` object to define the reflector geometry. You can also change the orientation of the reflectors and the exciter. You can use either a single antenna element or an array as the exciter for the object. Further, you can also create an antenna array using `customDualReflectors` object as its element. Dual-reflector antennas have very high gain and low spillover and are used in satellite communications.



Creation

Syntax

```
cdr = customDualReflectors
cdr = customDualReflectors(Name,Value)
```

Description

`cdr = customDualReflectors` creates a dual-reflector antenna with empty main and sub reflector geometries and conical horn antenna as the default exciter. After creating the object, specify the coordinates of the reflector surfaces in a *N*-by-3 matrix or import the coordinates from a MAT file.

`cdr = customDualReflectors(Name,Value)` sets “Properties” on page 1-785 using one or more name-value pairs. For example, `cdr = customDualReflectors('FeedOffset',[0.0850 0 0])` relocates the feed to the point (0.0850, 0, 0) with respect to the origin.

Properties

`Exciter` — Antenna or array to use as exciter

`hornConical` (default) | `antenna` object | `array` object

Antenna or array to use as an exciter, specified as an `antenna` or an `array` object.

Example: `cdr = customDualReflectors('Exciter',dipole)`

Example: `cdr.Exciter = dipole('Length',0.1409,'Width',0.02,'FeedOffset',0,'Tilt',90,'TiltAxis',[0 1 0])`

Example: `cdr.Exciter = linearArray('Element',patchMicrostrip)`

`MainReflector` — Cartesian coordinates of points that define main reflector surface

empty (default) | *N*-by-3 matrix | `triangulation` object

Cartesian coordinates of the main reflector surface, specified as an *N*-by-3 matrix with each element unit in meters. *N* represents the total number of points that define the main reflector surface. You can also import the coordinates from a MAT file. This property also accepts triangulation method for defining the main reflector surface.

Example: `'MainReflector',preflector`

Example: `cdr.MainReflector=preflector`

Example: `cdr=customDualReflectors('MainReflector',triangulation(T,P))`

Data Types: `double`

`SubReflector` — Cartesian coordinates of points that define sub reflector surface

empty (default) | *N*-by-3 matrix | `triangulation` object

Cartesian coordinates of the sub reflector surface, specified as an *N*-by-3 matrix with each element unit in meters. *N* represents the total number of points that define the sub reflector surface. You can also import the coordinates from a MAT file. This property also accepts triangulation method for defining the sub reflector surface.

Example: `'SubReflector',psubreflector`

Example: `cdr.SubReflector=psubreflector`

Example: `cdr=customDualReflectors('SubReflector',triangulation(T,P))`

Data Types: `double`

`ReflectorOffset` — Cartesian coordinates of main and sub reflector offsets

[0 0 0;0 0 0.2294] (default) | two three-element vectors

Cartesian coordinates of the main and sub reflector offsets with respect to the origin, specified as a 2-by-3 matrix with each element unit in meters. The first row corresponds to the main reflector offset and the second row corresponds to the sub reflector offset.

Example: `'ReflectorOffset',[-0.1 0 0;0.03 0 0.224]`

Example: `cdr.ReflectorOffset=[-0.1 0 0;0.03 0 0.224]`

Data Types: `double`

FeedOffset — Cartesian coordinates to offset exciter feed point

[0.0064 0 0.1173] (default) | three-element vector

Cartesian coordinates to offset the exciter feed point, specified as a three-element vector with each element unit in meters. If you have specified an array of exciters in the Exciter property, use this property to offset the center of the array.

Example: 'FeedOffset',[0.0850 0 0]

Example: cdr.FeedOffset = [0.0850 0 0]

Data Types: double

ReflectorTilt — Tilt angle of reflectors

[0 0] (default) | two-element vector

Tilt angle of the reflectors, specified as a two-element vector with each element unit in degrees. The first element specifies the tilt of the main reflector, and the second element specifies the tilt of the sub reflector. Specify values in the range [–360, 360].

Example: 'ReflectorTilt',[40 200]

Example: cdr.ReflectorTilt=[40 200]

Data Types: double

RemeshReflectors — Flag to re-mesh reflectors

1 (default) | 0 | true | false

Flag to re-mesh the reflectors, specified as a numeric or logical 1(true) or 0(false). Set this property to true to re-mesh the reflectors.

Example: 'RemeshReflectors',0

Example: cdr.RemeshReflectors=0

Data Types: logical | string

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: 'Tilt',90

Example: cdr.Tilt = 90

Example: 'Tilt',[90 90],'TiltAxis',[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the *x*-, *y*-, and *z*-axes.

- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedElement`, where `lumpedElement` is load added to the antenna feed.

Example: `cdr.Load = lumpedElement('Impedance',75)`

`SolverType` — Solver for antenna analysis

`'MoM-PO'` (default) | `'MoM'` | `'PO'` | `'FMM'`

Solver for antenna analysis, specified as the comma-separated pair consisting of `'SolverType'` and `'MoM-PO'`, `'PO'` (Physical Optics), `'MoM'` (Method of Moments), or `'FMM'` (Fast Multipole Method).

Example: `'SolverType'`,`'MOM'`

Data Types: `char`

Object Functions

show	Display antenna, array structures or shapes
impedance	Input impedance of antenna; scan impedance of array
sparameters	Calculate S-parameter for antenna and antenna array objects
returnLoss	Return loss of antenna; scan return loss of array
vswr	Voltage standing wave ratio of antenna
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
optimize	Optimize antenna or array using SADEA optimizer
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
current	Current distribution on antenna or array surface
charge	Charge distribution on antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal, dielectric antenna, or array structure
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array

Examples

Build Custom Dual-Reflector Antenna

Load the MAT files containing the variables which store the coordinates for the main and sub reflector surfaces.

```
load mainref.mat; %Loads a variable 'preflector' into the workspace
load subref.mat; %Loads a variable 'psubreflector' into the workspace
```

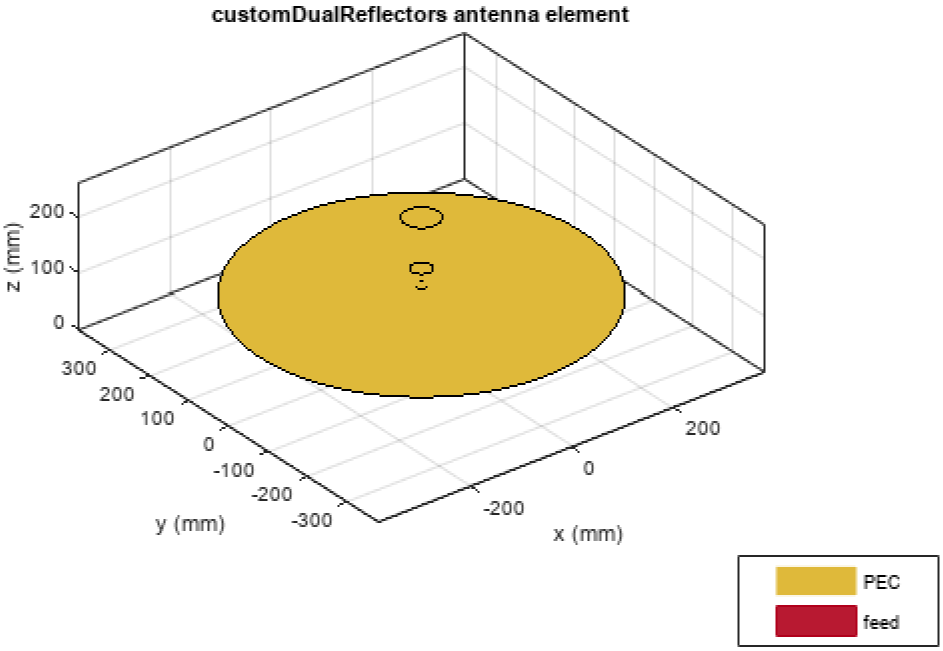
Create a `customDualReflectors` object by assigning the coordinates to the `MainReflector` and `SubReflector` properties.

```
cdr = customDualReflectors('MainReflector',preflector,'SubReflector',psubreflector)
```

```
cdr = 
  customDualReflectors with properties:

              Exciter: [1x1 hornConical]
        MainReflector: [3364x3 double]
         SubReflector: [144x3 double]
      ReflectorOffset: [2x3 double]
           FeedOffset: [0.0064 0 0.1173]
        ReflectorTilt: [0 0]
     RemeshReflectors: 1
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
           SolverType: 'MoM-PO'
```

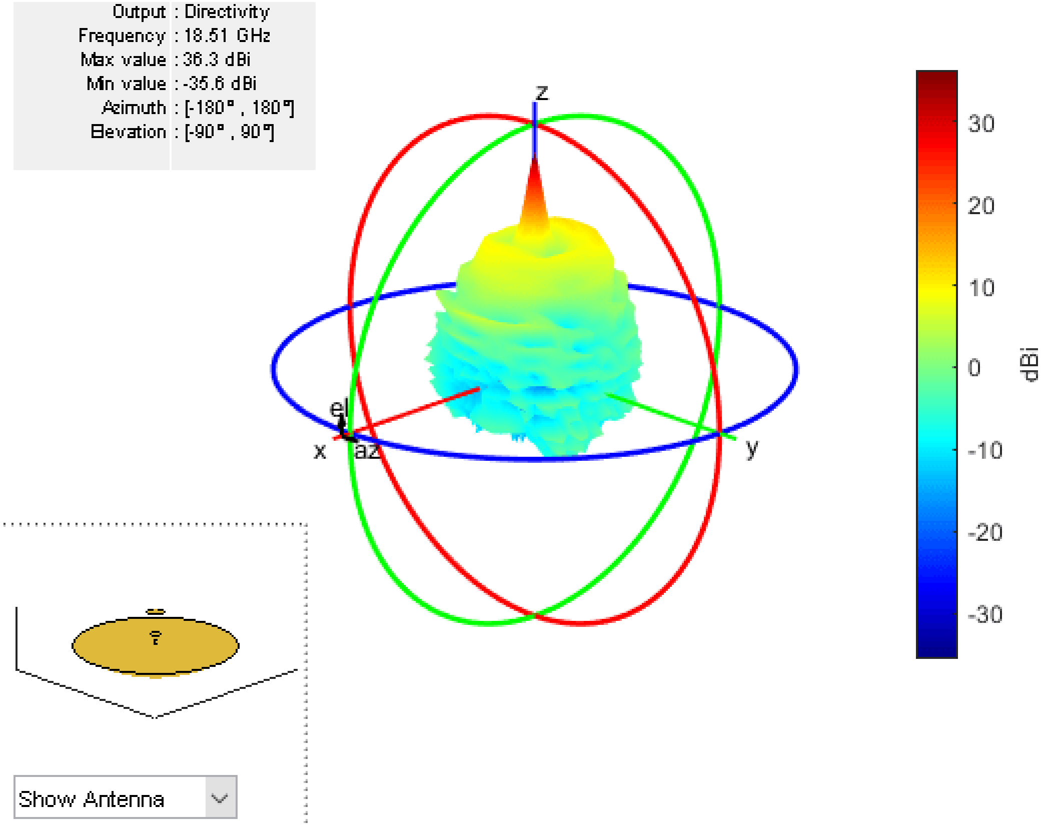
View the antenna.

```
show(cdr)
```



Plot a 3-D radiation pattern of this antenna at 18.51 GHz.

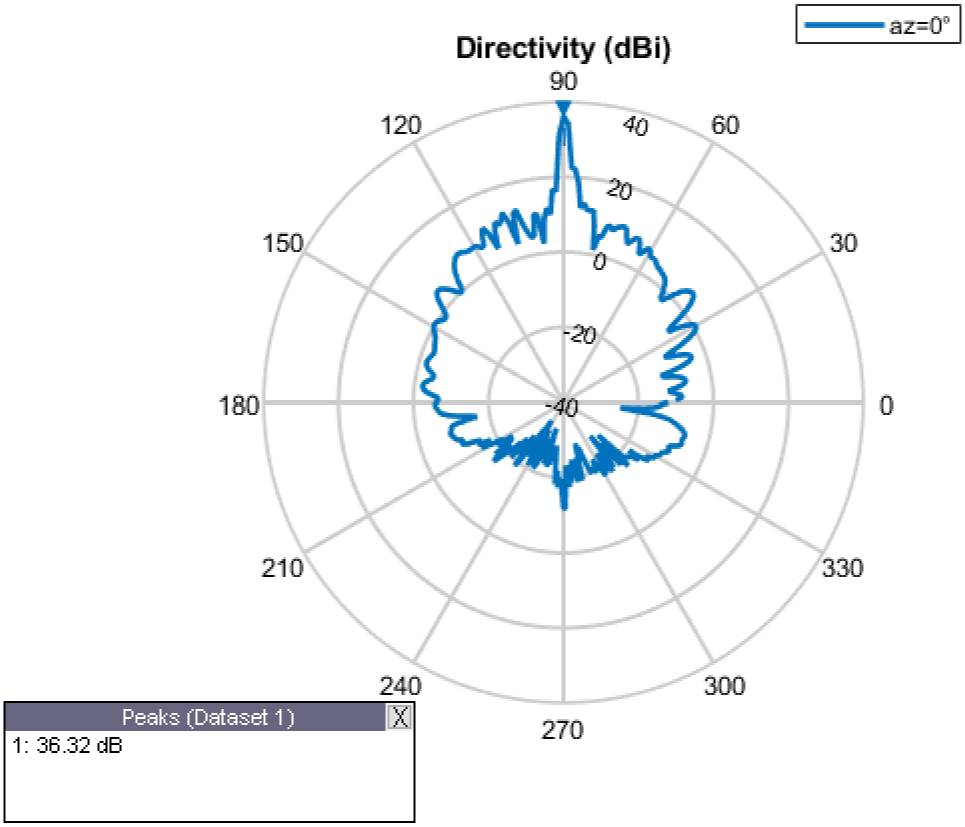
```
pattern(cdr,18.51e9)
```



Output : Directivity
Frequency : 18.51 GHz
Max value : 36.3 dBi
Min value : -35.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Plot the elevation pattern of this antenna in the *X-Z* plane.

```
pattern(cdr,18.51e9,0,1:1:360)
```

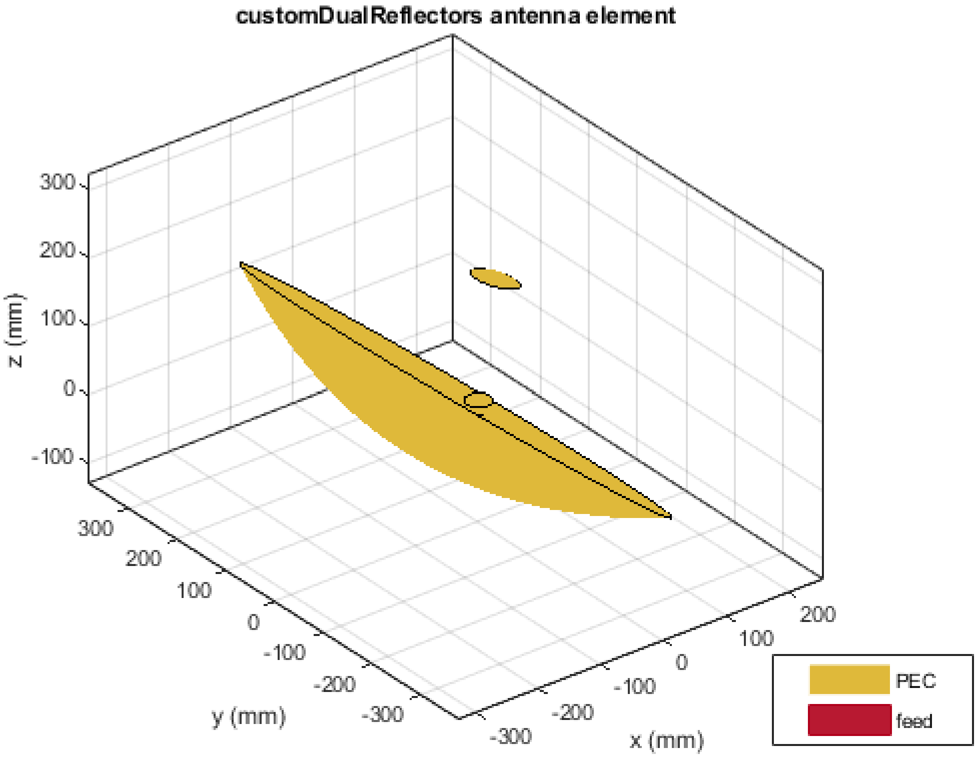


Peaks (Dataset 1)

1: 36.32 dB

Tilt and offset the reflectors and the feed. View the transformed antenna.

```
cdr.ReflectorTilt=[40 200];
cdr.ReflectorOffset=[-0.1 0  0;0.03 0 0.224];
cdr.FeedOffset=[0.0072 0 0.02];
show(cdr)
```



Version History

Introduced in R2022a

See Also

`reflectorParabolic` | `cassegrain` | `gregorian` | `cassegrainOffset` | `gregorianOffset` | `hornConical`

Topics

“Rotate Antennas and Arrays”

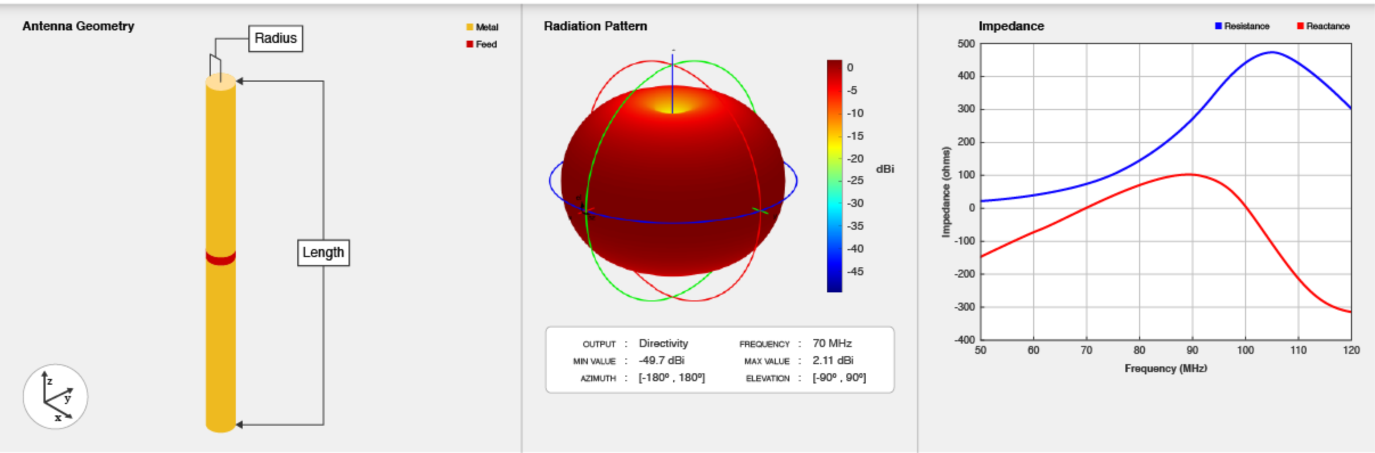
dipoleCylindrical

Create cylindrical dipole antenna

Description

The `dipoleCylindrical` object creates a cylindrical dipole antenna. The length of the cylindrical dipole corresponds to half of the wavelength at the operating frequency. These antennas are used in designing thicker dipole antennas. These antennas are mostly used in wireless communications due to their simple design.

Dipole Cylindrical



Creation

Syntax

```
ant = dipoleCylindrical
ant = dipoleCylindrical(Name,Value)
```

Description

`ant = dipoleCylindrical` creates a cylindrical dipole antenna object with default dimensions for an operating frequency of 70 MHz. The default dipole is center fed, with the feedpoint at the origin located on the *y*-*z* plane.

`ant = dipoleCylindrical(Name,Value)` sets “Properties” on page 1-793 using one or more name-value pairs. For example, `ant = dipoleCylindrical('Radius',0.04)` creates a cylindrical dipole antenna with radius of 0.04 meters.

Properties

`Length` — Length of dipole
2 (default) | positive scalar

Length of the dipole, specified as a positive scalar in meters.

Example: 'Length',3

Data Types: double

Radius — Radius of dipole

0.025 (default) | positive scalar

Radius of the dipole, specified as a positive scalar in meters.

Example: 'Radius',0.05

Data Types: double

FeedOffset — Signed distance from center along *z*-axis

0 (default) | real-valued scalar

Signed distance from the center along Z-axis, specified as a real-valued scalar in meters.

Example: 'FeedOffset',-0.2

Data Types: double

ClosedEnd — Determines if dipole ends are closed with cap

0 (default) | 1

Determine if the dipole ends are closed with a flat cap or left open, specified as 0 or 1. Specify 1 for closed ends of the dipole and 0 for open ends.

Example: 'ClosedEnd',1

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: 'Tilt',90

Example: ant.Tilt = 90

Example: 'Tilt',[90 90],'TiltAxis',[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: double

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Data Types: `double`

`Load` — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
cylinder2strip	Cylinder equivalent width approximation
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array

sparameters	Calculate S-parameter for antenna and antenna array objects
strip2cylinder	Calculates equivalent radius approximation for strip
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Cylindrical Dipole Antenna and Plot Radiation Pattern

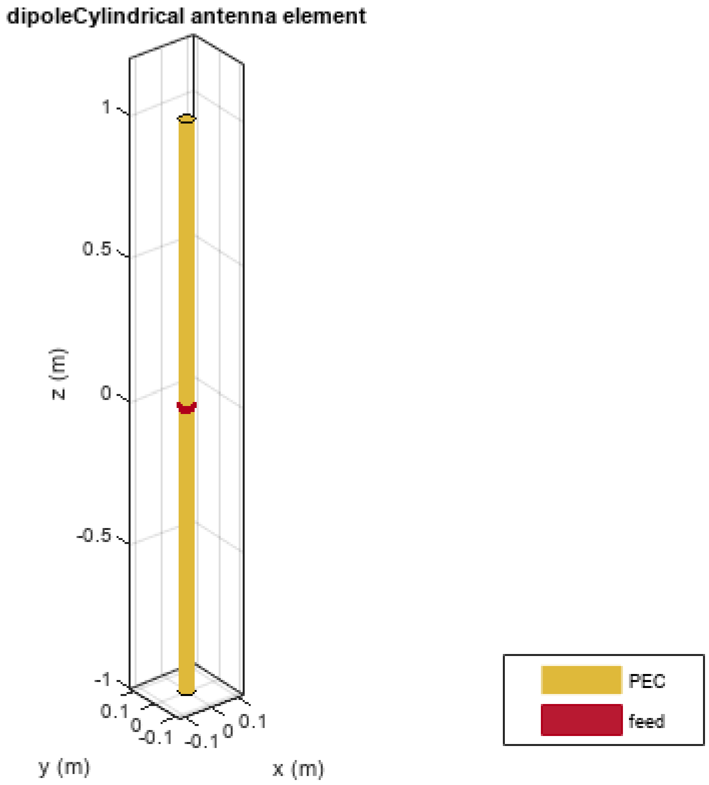
Create a cylindrical dipole antenna with default properties.

```
ant = dipoleCylindrical
```

```
ant = 
  dipoleCylindrical with properties:

        Length: 2
        Radius: 0.0250
    FeedOffset: 0
     ClosedEnd: 0
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

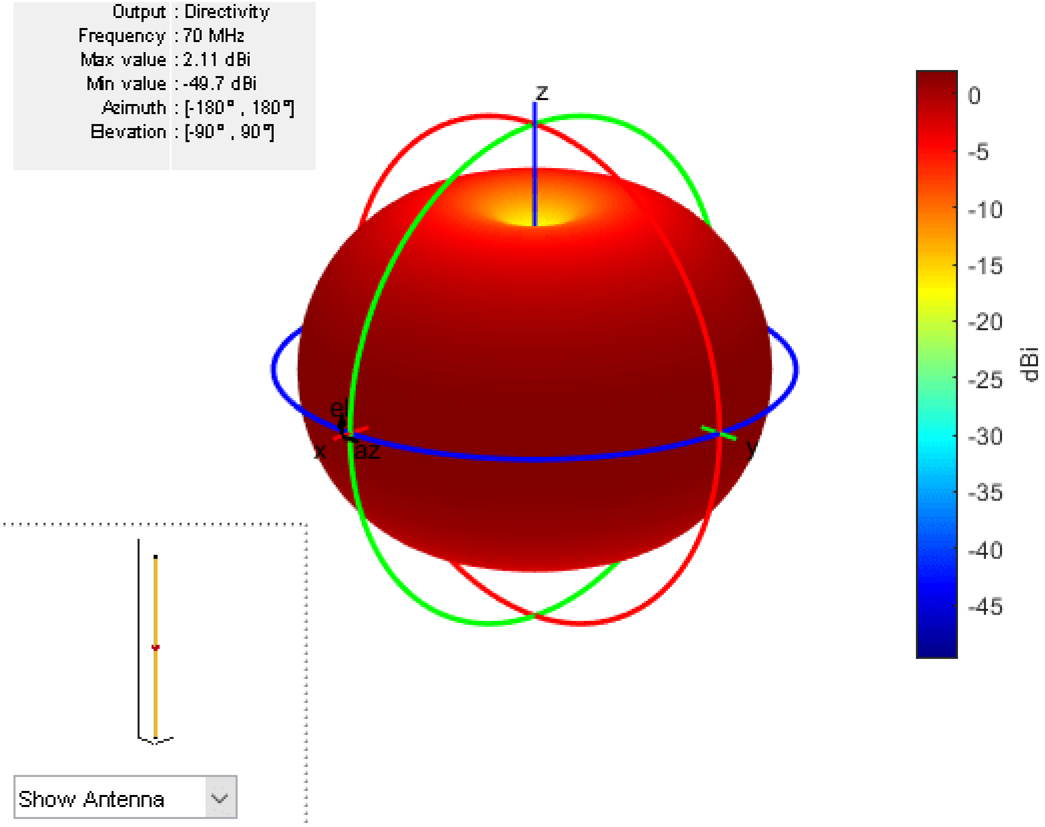
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of the cylindrical dipole antenna at a frequency of 70 MHz.

```
pattern(ant,70e6)
```



Output : Directivity
Frequency : 70 MHz
Max value : 2.11 dBi
Min value : -49.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Impedance of Cylindrical Dipole Antenna

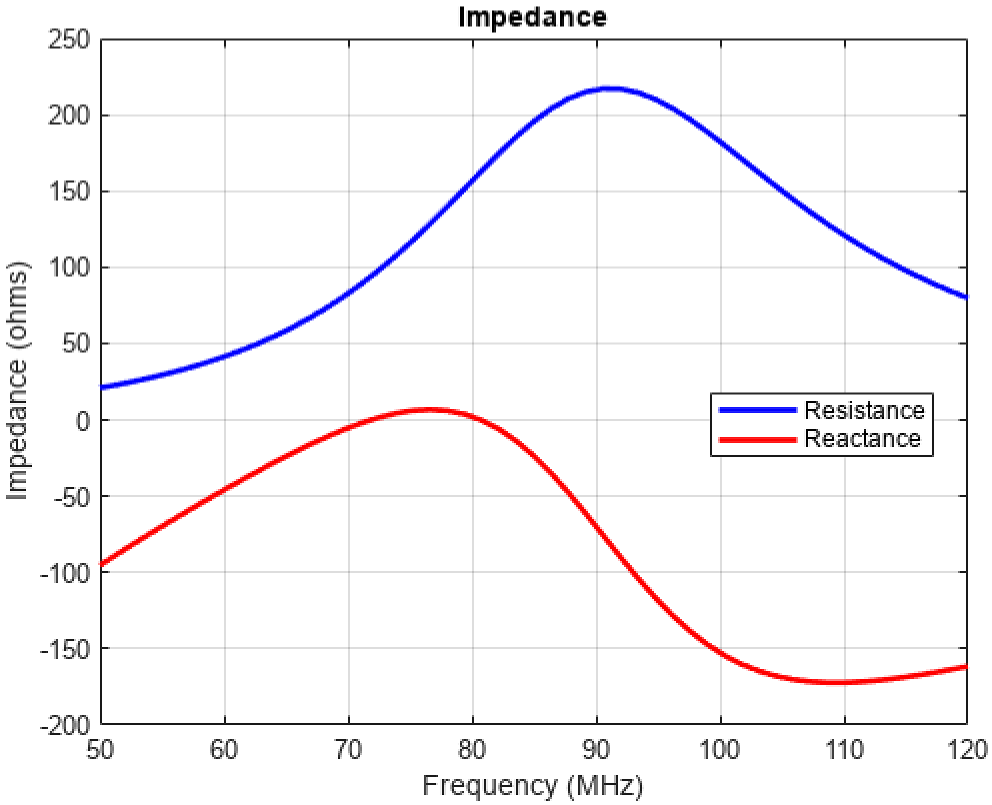
Create a center-fed cylindrical dipole with a length of 2 m and a radius of `0.06` m.

```
ant = dipoleCylindrical('Length',2,'Radius',0.06)
```

```
ant = 
  dipoleCylindrical with properties:

        Length: 2
        Radius: 0.0600
    FeedOffset: 0
     ClosedEnd: 0
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

Plot the impedance over a frequency range of 50 MHz to 120 MHz.

```
impedance(ant,linspace(50e6,120e6,51))
```



Compare Current Distribution of Open- and Close-ended Cylindrical Dipole Antennas

Create cylindrical dipole antennas with an open-ended top and a close-ended top, respectively.

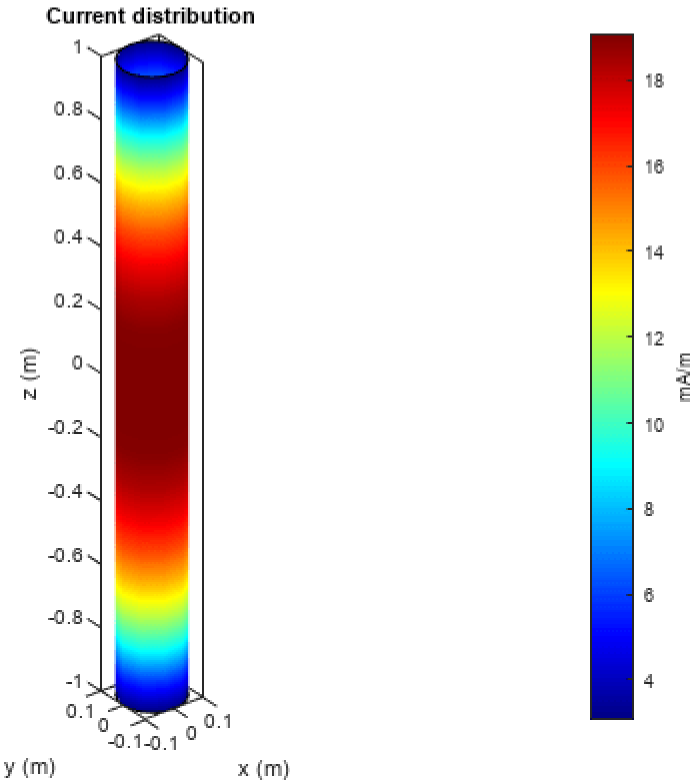
```
ant = dipoleCylindrical("Radius",0.1);
ant_ClosedEnded = dipoleCylindrical("Radius",0.1,"ClosedEnd",1);
```

Calculate and plot the current distribution for the cylindrical dipole antennas at frequency of 70 MHz.

```
I_OpenEnded  = current(ant,70e6)
```

```
I_OpenEnded = 3×400 complex

  -0.0000 - 0.0002i  -0.0000 - 0.0005i  -0.0000 - 0.0007i  -0.0000 - 0.0005i  -0.0000 - 0.0002i
   0.0000 + 0.0006i   0.0000 + 0.0004i   0.0000 + 0.0000i  -0.0000 - 0.0004i  -0.0000 - 0.0006i
  -0.0191 - 0.0016i  -0.0191 - 0.0016i  -0.0191 - 0.0016i  -0.0191 - 0.0016i  -0.0191 - 0.0016i
```

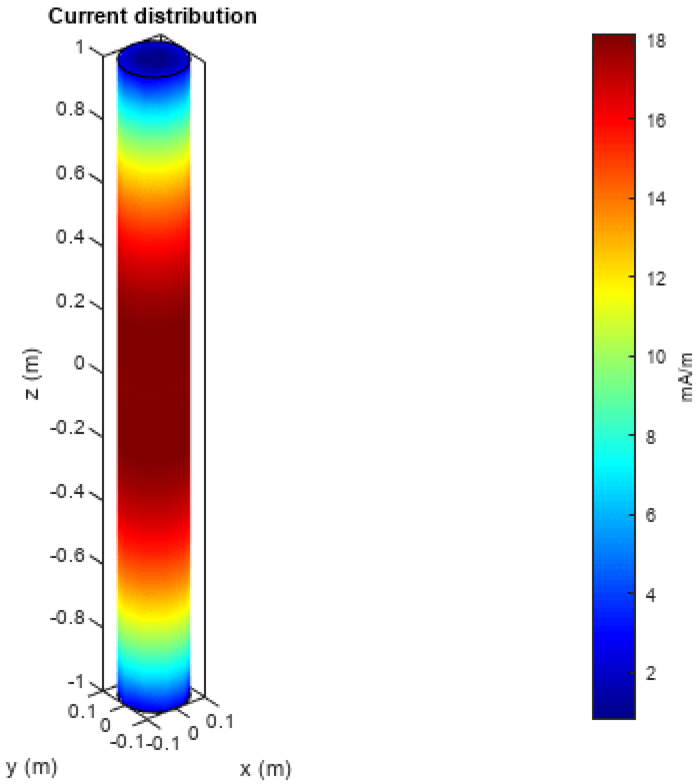
```
current(ant,70e6)
```



```
I_ClosedEnded = current(ant_ClosedEnded,70e6)
```

```
I_ClosedEnded = 3×428 complex

  -0.0000 - 0.0002i  -0.0000 - 0.0005i  -0.0000 - 0.0007i  -0.0000 - 0.0005i  -0.0000 - 0.0002i
   0.0000 + 0.0006i   0.0000 + 0.0004i   0.0000 + 0.0000i  -0.0000 - 0.0004i  -0.0000 - 0.0006i
  -0.0180 - 0.0011i  -0.0180 - 0.0011i  -0.0180 - 0.0011i  -0.0180 - 0.0011i  -0.0180 - 0.0011i
```

```
figure;
current(ant_ClosedEnded,70e6)
```



More About

Parametric Analysis Guidelines

To understand how the properties of the `dipoleCylindrical` antenna object influence the antenna design, use these parametric analysis guidelines.

- To increase the operating frequency, decrease the “Length” on page 1-0 of the `dipoleCylindrical` antenna.
- To increase the gain, increase the “Length” on page 1-0 of the `dipoleCylindrical` antenna object.
- To increase the bandwidth, increase the “Radius” on page 1-0 of the `dipoleCylindrical` antenna object.

Version History

Introduced in R2021a

References

[1] King Ronald W.P. *Characteristics of Cylindrical Dipoles and Monopoles*. Boston, MA: Springer, 1971.

See Also

`dipole` | `monopoleCylindrical`

Topics

“Rotate Antennas and Arrays”

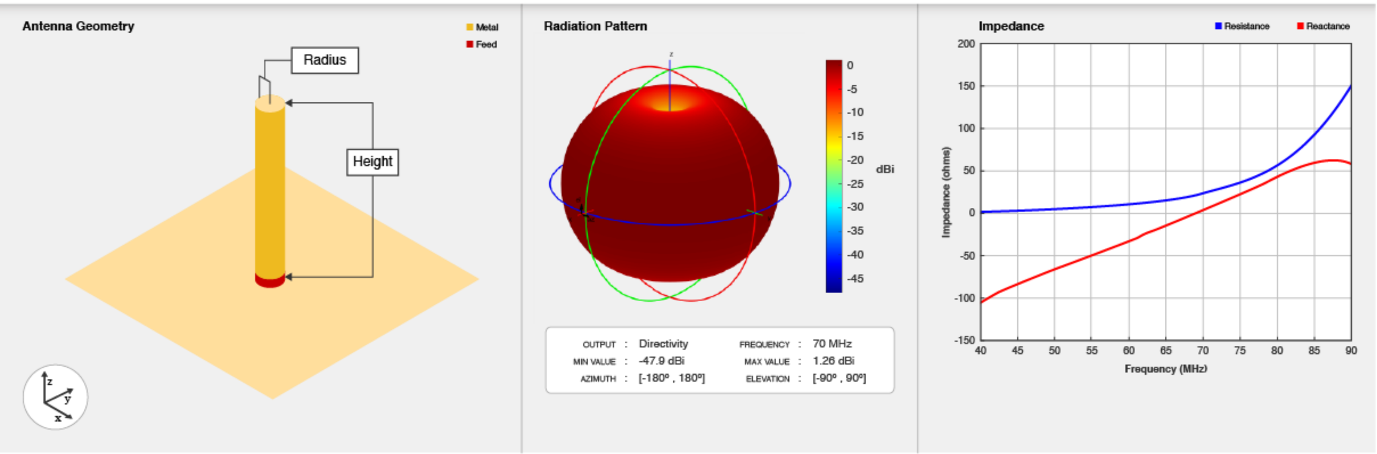
monopoleCylindrical

Create cylindrical monopole antenna over rectangular ground plane

Description

The `monopoleCylindrical` object is a cylindrical monopole antenna mounted over a rectangular ground plane. This antenna is useful for designing thicker monopole antennas. These antennas are mostly used in wireless mobile communication due to their broadband characteristics and simple design.

Monopole Cylindrical



Creation

Syntax

```
ant = monopoleCylindrical
ant = monopoleCylindrical(Name,Value)
```

Description

`ant = monopoleCylindrical` creates a cylindrical monopole antenna object with default dimensions for an operating frequency of 70 MHz.

`ant = monopoleCylindrical(Name,Value)` sets “Properties” on page 1-803 using one or more name-value pairs. For example, `ant = monopoleCylindrical('Radius',0.04)`, creates a cylindrical monopole antenna with a radius of 0.04 meters.

Properties

`Height` — Height of monopole along *z*-axis

1 (default) | positive scalar

Height of the monopole along the z-axis, specified as a positive scalar in meters.

Example: `'Height',3`

Data Types: `double`

`Radius` — Radius of monopole

`0.040` (default) | positive scalar

Radius of the monopole, specified as a positive scalar in meters.

Example: `'Radius',0.05`

Data Types: `double`

`GroundPlaneLength` — Ground plane length along *x*-axis

2 (default) | positive scalar

Ground plane length along x-axis, specified as a positive scalar in meters.

Example: `'GroundPlaneLength',4`

Data Types: `double`

`GroundPlaneWidth` — Ground plane width along *y*-axis

2 (default) | positive scalar

Ground plane width along y-axis, specified as a positive scalar in meters.

Example: `'GroundPlaneWidth',2.5`

Data Types: `double`

`FeedOffset` — Signed distance from center along length and width of ground plane

`[0 0]` (default) | two-element vector

Signed distance from the center along the length and the width of the ground plane, specified as a two-element vector in meters.

Example: `'FeedOffset',[2 1]`

Data Types: `double`

`ClosedEnd` — Determines if monopole top is closed with cap

`0` (default) | `1`

Determine if the monopole top is closed with a cap or left open, specified as 0 or 1. Specify 1 for a closed-top monopole, and 0 for an open-top monopole.

Example: `'ClosedEnd',1`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Data Types: `double`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
cylinder2strip	Cylinder equivalent width approximation

design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
strip2cylinder	Calculates equivalent radius approximation for strip
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Cylindrical Monopole Antenna and Plot Radiation Pattern

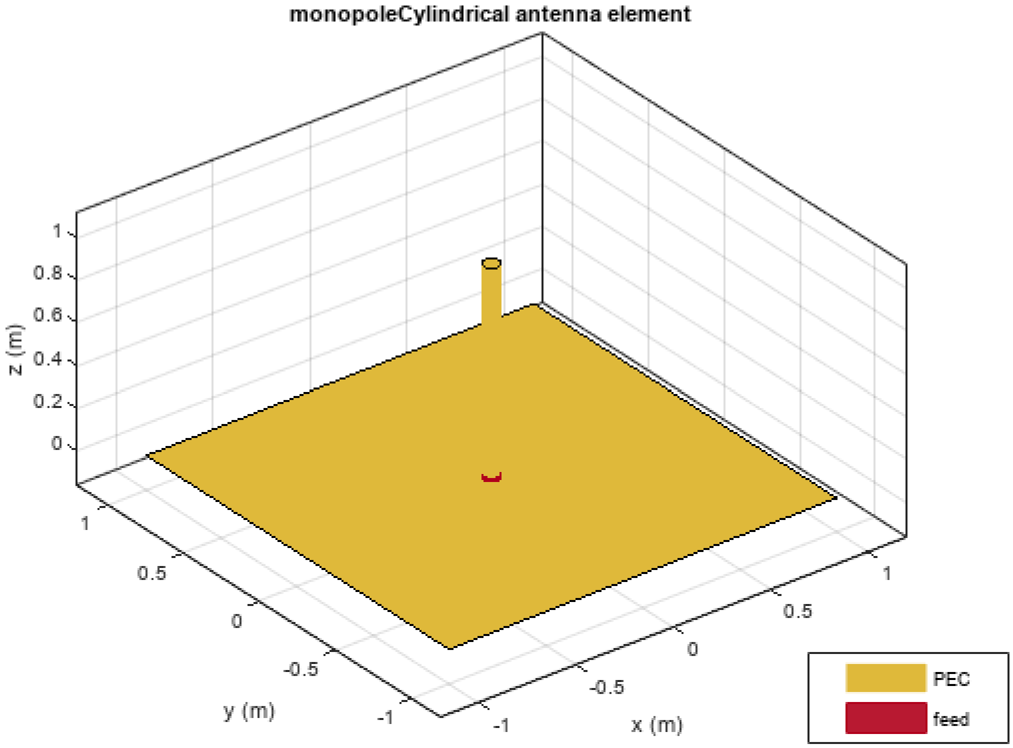
Create a cylindrical monopole antenna with default properties.

```
ant = monopoleCylindrical
```

```
ant = 
  monopoleCylindrical with properties:

               Height: 1
               Radius: 0.0400
    GroundPlaneLength: 2
     GroundPlaneWidth: 2
           FeedOffset: [0 0]
            ClosedEnd: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

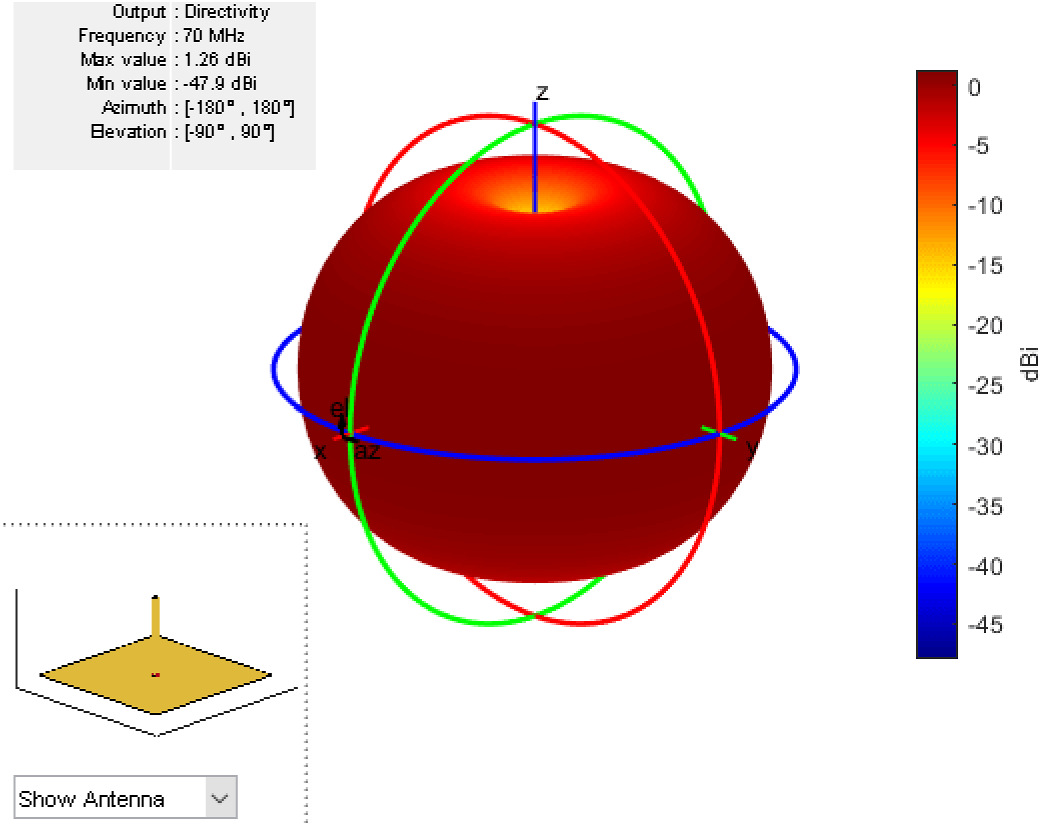
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of cylindrical monopole antenna at a frequency of 70 MHz.

```
pattern(ant,70e6)
```



Output : Directivity
Frequency : 70 MHz
Max value : 1.26 dBi
Min value : -47.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Plot S-Parameter of Cylindrical Monopole Antenna

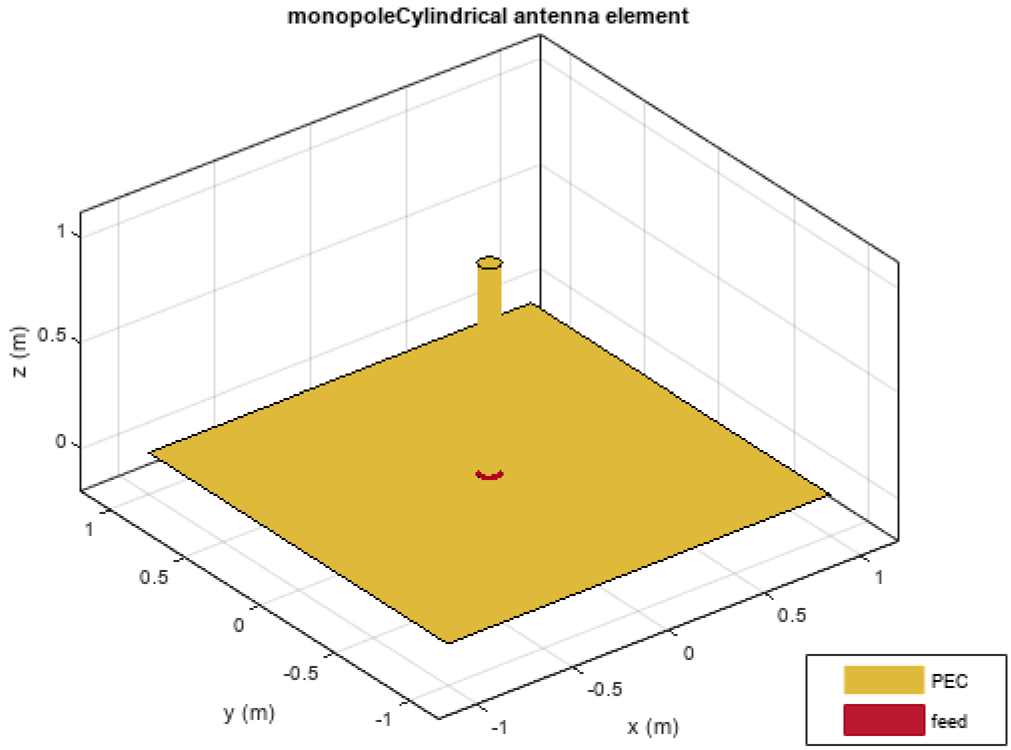
Create a cylindrical monopole antenna with length and radius as 1 m and 50 mm respectively.

```
ant = monopoleCylindrical;
ant.Height = 1;
ant.Radius = 50e-3
```

```
ant = 
  monopoleCylindrical with properties:

             Height: 1
             Radius: 0.0500
    GroundPlaneLength: 2
     GroundPlaneWidth: 2
           FeedOffset: [0 0]
            ClosedEnd: 0
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

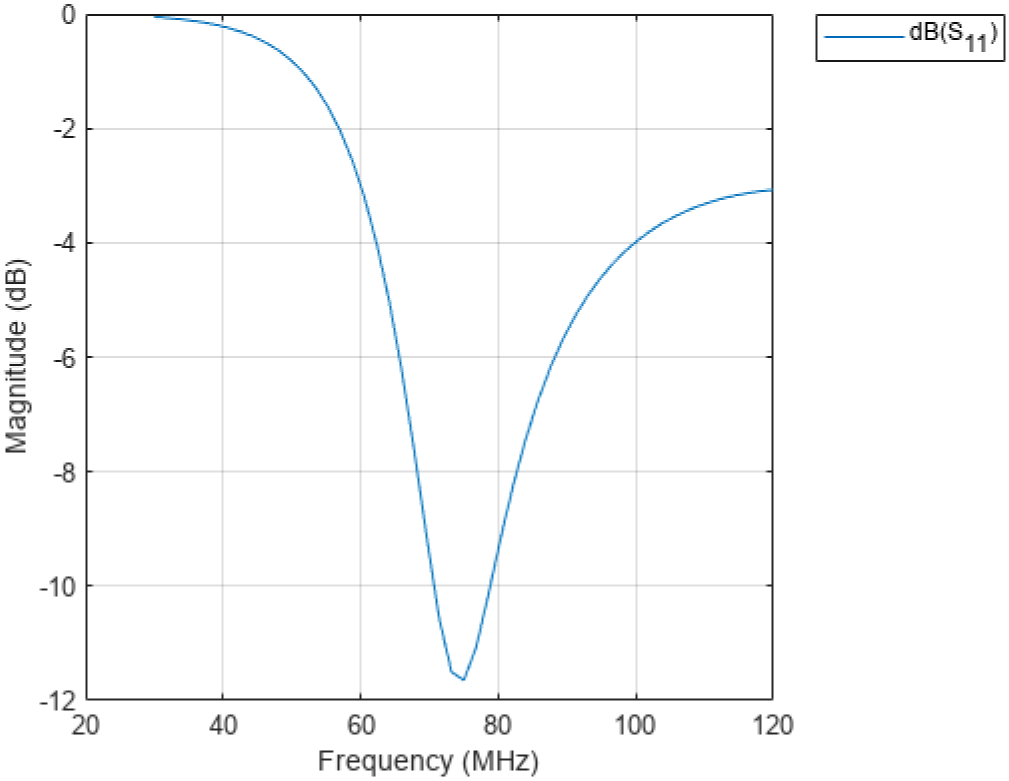
Visualize the antenna using `show` function.

```
show(ant)
```



Plot the S-parameters over a frequency range of 30 MHz to 120 MHz.

```
s = sparameters(ant,linspace(30e6,120e6,51));
rfplot(s)
```



Compare Current Distribution of Open-ended and Closed-ended Cylindrical Monopole Antenna

Create cylindrical monopole antennas with an open-ended top and a closed-ended top respectively.

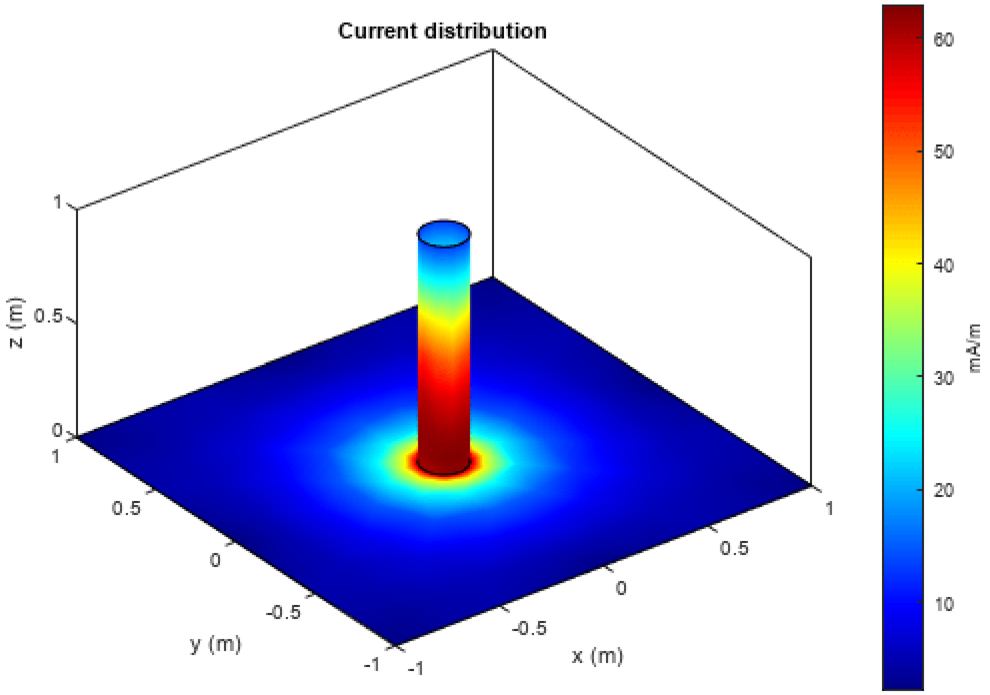
```
ant = monopoleCylindrical("Radius",0.1);
ant_ClosedEnded = monopoleCylindrical("Radius",0.1,"ClosedEnd",1);
```

Calculate and plot the current distribution for a cylindrical monopole antennas at 70 MHz frequency.

```
I_OpenEnded = current(ant,70e6)
```

```
I_OpenEnded = 3×334 complex

  -0.0105 + 0.0033i  -0.0001 + 0.0000i   0.0063 - 0.0024i  -0.0008 + 0.0004i   0.0222 - 0.0056i
   0.0096 - 0.0028i   0.0053 - 0.0020i   0.0030 - 0.0013i   0.0039 - 0.0019i   0.0073 - 0.0018i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

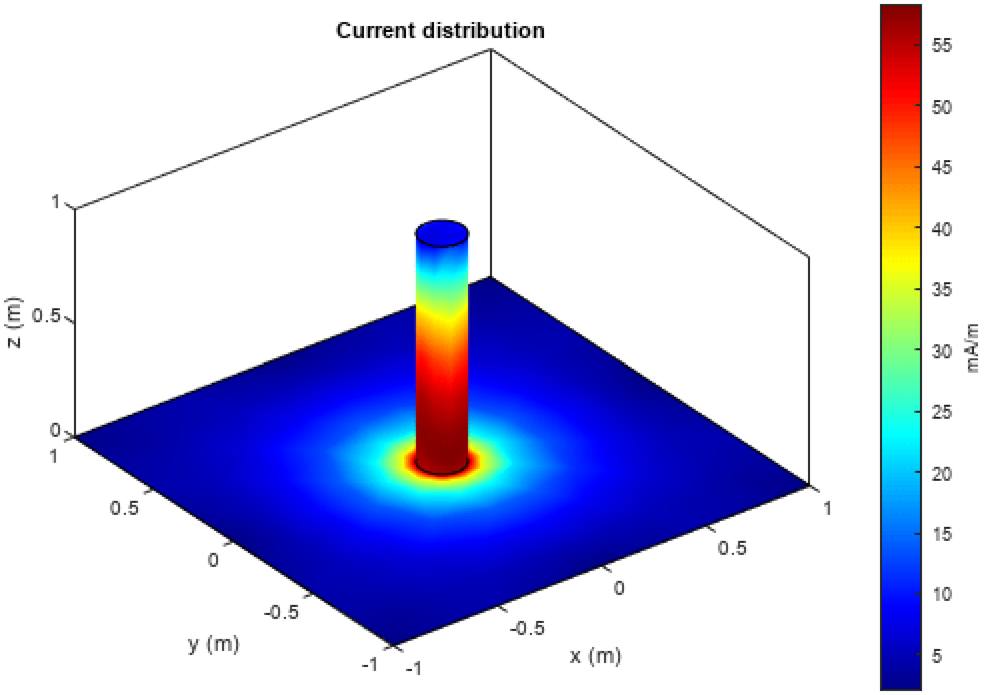
```
current(ant,70e6)
```



```
I_ClosedEnded = current(ant_ClosedEnded,70e6)
```

```
I_ClosedEnded = 3×342 complex

  -0.0095 + 0.0039i  -0.0001 + 0.0000i   0.0057 - 0.0027i  -0.0007 + 0.0004i   0.0201 - 0.0069i
   0.0087 - 0.0034i   0.0048 - 0.0023i   0.0028 - 0.0014i   0.0036 - 0.0021i   0.0067 - 0.0022i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
figure;
current(ant_ClosedEnded,70e6)
```



More About

Parametric Analysis Guidelines

To understand how the properties of the `monopoleCylindrical` antenna object influence the antenna design, use the following parametric analysis guidelines.

- To increase the operating frequency, decrease the “Height” on page 1-0 of the `monopoleCylindrical` antenna.
- To increase the gain, increase the ground plane dimensions of the `monopoleCylindrical` antenna object using the “GroundPlaneLength” on page 1-0 and the “GroundPlaneWidth” on page 1-0 properties.
- To increase the bandwidth, increase the radius and change the feed location of the `monopoleCylindrical` antenna object using the “Radius” on page 1-0 and the “FeedOffset” on page 1-0 properties.

Version History

Introduced in R2021a

References

[1] King, Ronald W.P. *Characteristics of Cylindrical Dipoles and Monopoles*. Boston, MA: Springer, 1971.

See Also

`monopole` | `monopoleTopHat` | `dipoleCylindrical`

Topics

"Rotate Antennas and Arrays"

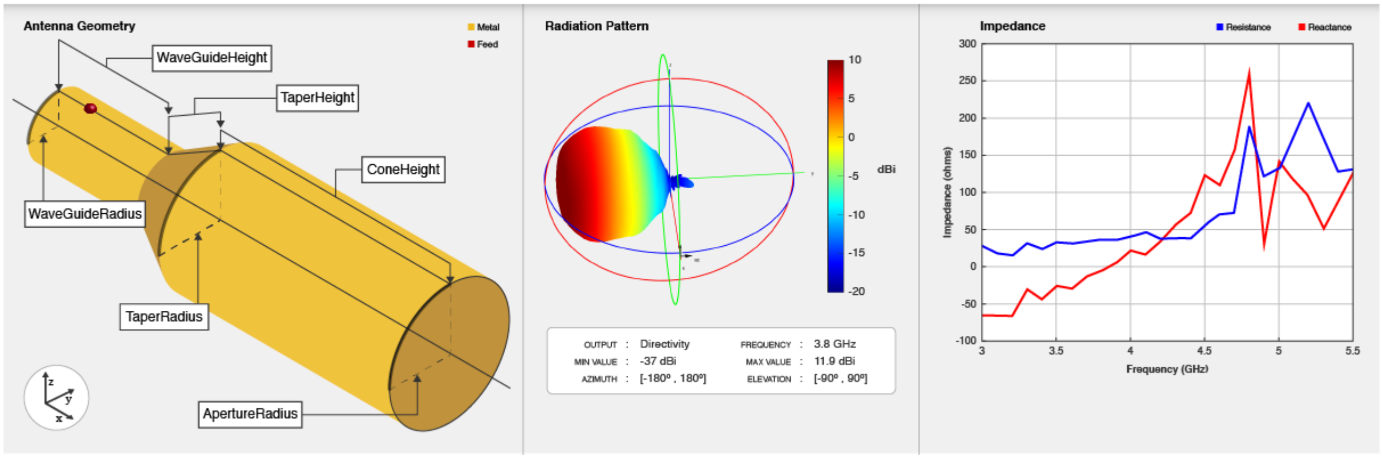
hornPotter

Create Potter horn antenna

Description

The `hornPotter` object creates a Potter horn antenna. The Potter horn antenna is a dual-mode, conical horn antenna with discontinuity. These antennas are used in wireless applications where properties like low cross polarization level, low side lobe level, and beam symmetry are required. Potter horns are widely used in satellite communications for frequency reuse.

hornPotter



OUTPUT : Directivity
MIN VALUE : -37 dBi
AZIMUTH : [-180º , 180º]
FREQUENCY : 3.8 GHz
MAX VALUE : 11.9 dBi
ELEVATION : [-90º , 90º]

Creation

Syntax

```
ant = hornPotter
ant = hornPotter(Name,Value)
```

Description

`ant = hornPotter` creates a default Potter horn antenna with tapered transition. The default dimensions are chosen for an operating frequency of 3.8 GHz.

`ant = hornPotter(Name,Value)` sets “Properties” on page 1-814 using one or more name-value pairs. For example, `hornPotter('ConeHeight',0.05)` creates a Potter horn antenna with the cone height of 50 mm.

Properties

Radius — Radius of circular waveguide
`0.0311` (default) | positive scalar

Radius of the circular waveguide, specified as a positive scalar in meters.

Example: `'Radius',0.760`

Example: `ant.Radius = 0.760`

Data Types: `double`

`WaveguideHeight` — Height of circular waveguide
`0.1200` (default) | positive scalar

Height of the circular waveguide, specified as a positive scalar in meters.

Example: `'WaveguideHeight',0.0340`

Example: `ant.WaveguideHeight = 0.0340`

Data Types: `double`

`FeedHeight` — Height of feed
`0.0185` (default) | positive scalar

Height of the feed, specified as a positive scalar in meters.

Example: `'FeedHeight',0.0085`

Example: `ant.FeedHeight = 0.0085`

Data Types: `double`

`FeedWidth` — Width of feed
`0.0005` (default) | positive scalar

Width of the feed, specified as a positive in meters.

Example: `'FeedWidth',0.0200`

Example: `ant.FeedWidth = 0.0200`

Data Types: `double`

`FeedOffset` — Signed distance along waveguide height
`0.0300` (default) | real-valued scalar

Signed distance along the waveguide height, specified as a real-valued scalar in meters.

Example: `'FeedOffset',0.03627`

Example: `ant.FeedOffset = 0.3627`

Data Types: `double`

`ConeHeight` — Height of cone
`0.2416` (default) | positive scalar

Height of the cone, specified as a positive scalar in meters.

Example: `'ConeHeight',0.0540`

Example: `ant.ConeHeight = 0.0540`

Data Types: `double`

ApertureRadius — Radius of cone aperture

0.0626 (default) | positive scalar

Radius of the cone aperture, specified as a positive scalar in meters.

Example: 'ApertureRadius',0.0560

Example: ant.ApertureRadius = 0.0790

Data Types: double

TaperRadius — Radius of taper

0.0626 (default) | positive scalar

Radius of the taper, specified as a positive scalar in meters.

Example: 'TaperRadius',0.0760

Example: ant.TaperRadius = 0.0760

Data Types: double

TaperHeight — Height of taper

0.0546 (default) | nonnegative scalar

Height of the taper, specified as a nonnegative scalar in meters. To design step-transition Potter horn antenna, set TaperHeight to 0.

Example: 'TaperHeight',0.0540

Example: ant.TaperHeight = 0.0540

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: 'Tilt',90

Example: ant.Tilt = 90

Example: 'Tilt',[90 90],'TiltAxis',[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: double

`TiltAxis` — Tilt axis of antenna
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

Load — Lumped elements
[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object handle. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Potter Horn Antenna and Plot Radiation Pattern

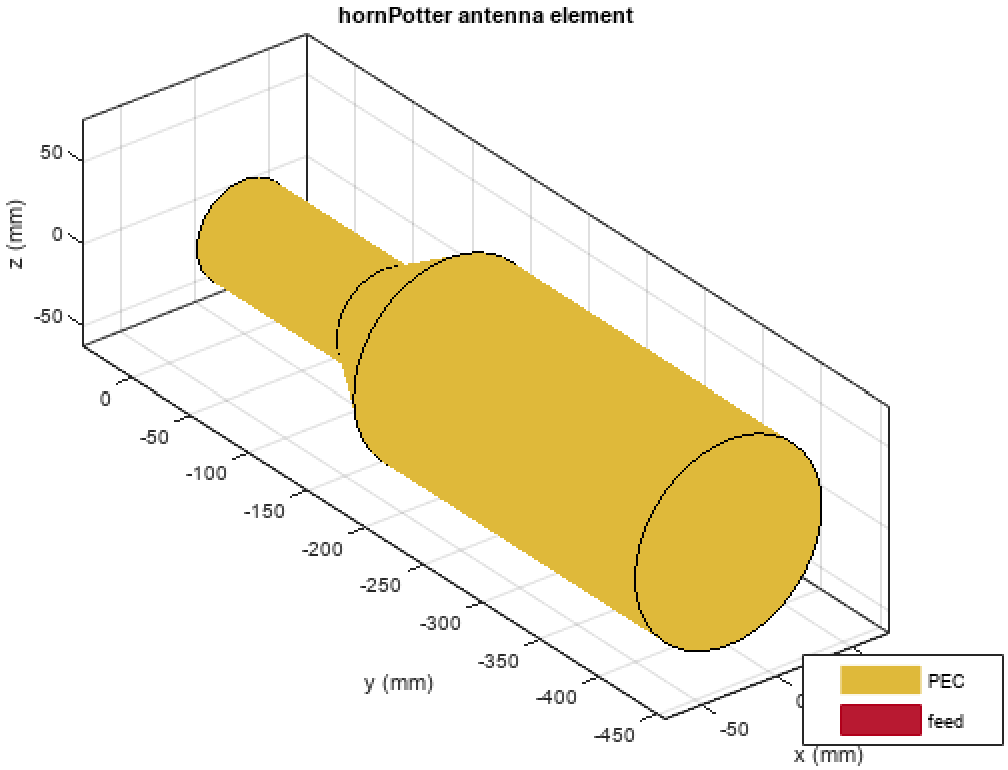
Create a Potter horn antenna with default properties.

```
ant = hornPotter
```

```
ant = 
  hornPotter with properties:

             Radius: 0.0311
    WaveguideHeight: 0.1200
         FeedHeight: 0.0185
          FeedWidth: 5.0000e-04
         FeedOffset: 0.0300
         ConeHeight: 0.2414
     ApertureRadius: 0.0626
        TaperRadius: 0.0626
        TaperHeight: 0.0546
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

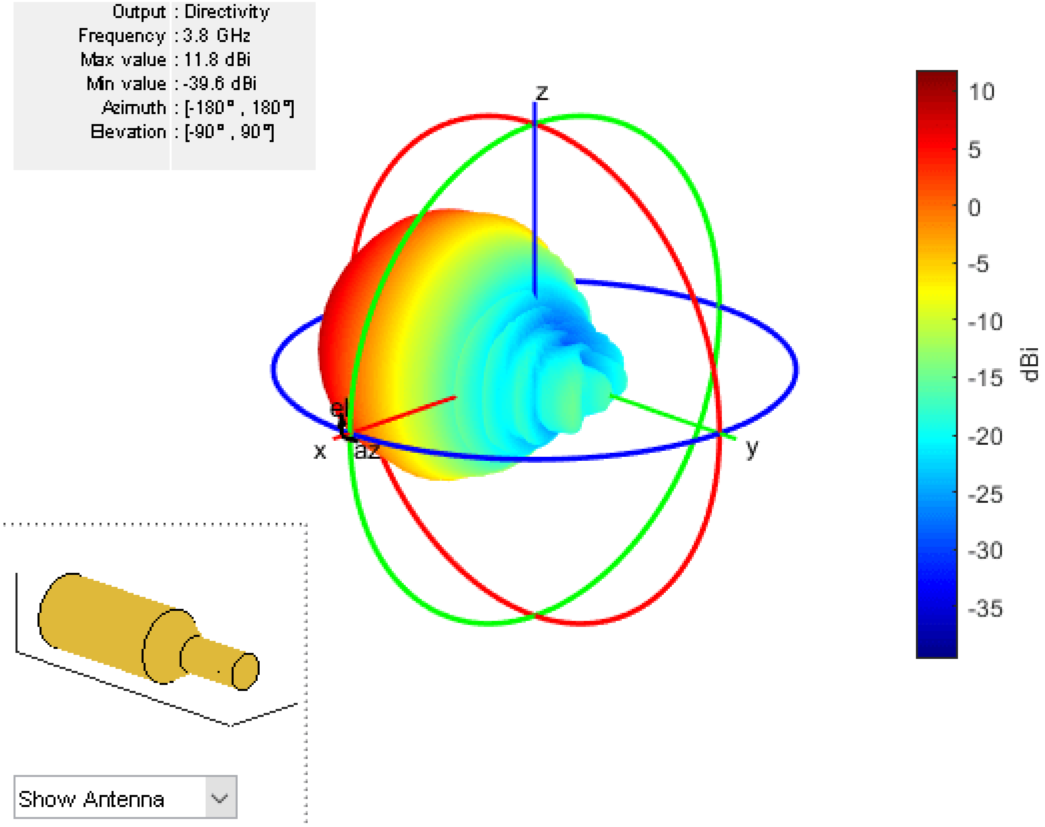
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of the Potter horn antenna at a frequency of 3.8 GHz.

```
pattern(ant,3.8e9)
```



```
Output : Directivity
Frequency : 3.8 GHz
Max value : 11.8 dBi
Min value : -39.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Logarithmic Charge Distribution on Potter Horn Antenna Surface

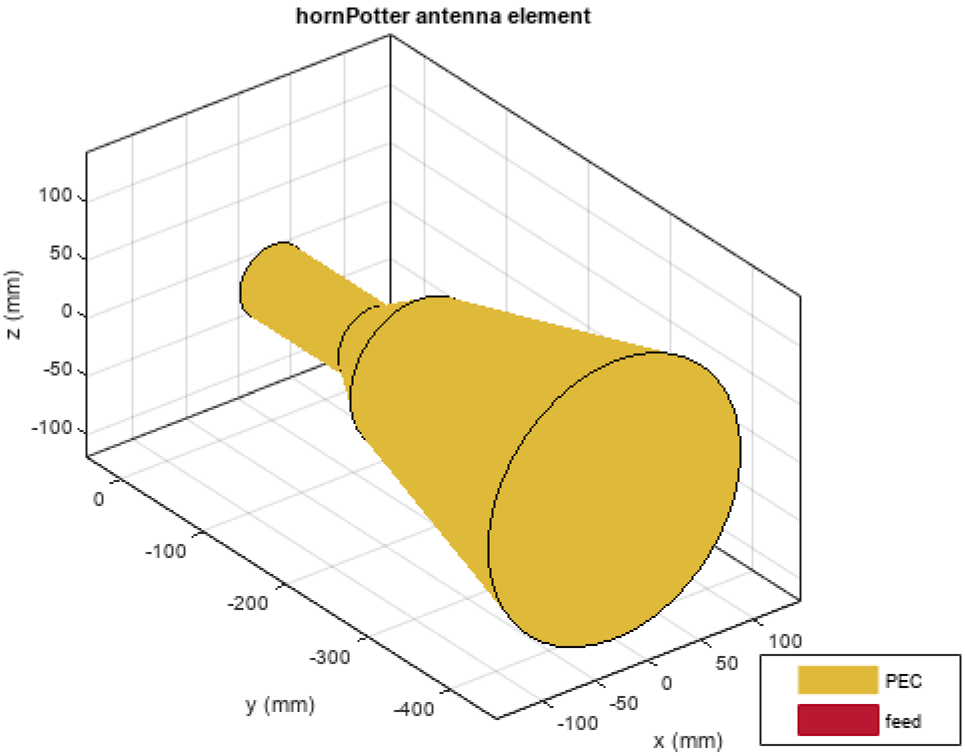
Create a Potter horn antenna with the aperture radius of 0.12 m.

```
ant = hornPotter('ApertureRadius',0.12)
```

```
ant = 
  hornPotter with properties:

              Radius: 0.0311
     WaveguideHeight: 0.1200
          FeedHeight: 0.0185
           FeedWidth: 5.0000e-04
          FeedOffset: 0.0300
          ConeHeight: 0.2414
      ApertureRadius: 0.1200
         TaperRadius: 0.0626
         TaperHeight: 0.0546
           Conductor: [1x1 metal]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1x1 lumpedElement]
```

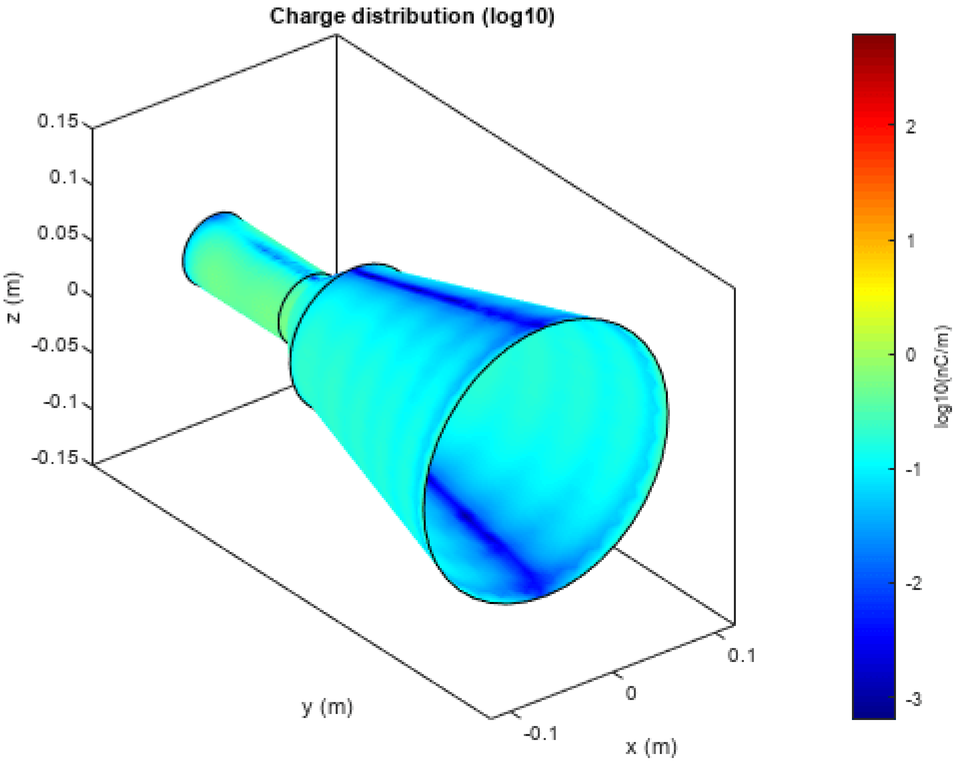
View the antenna using the `show` function.

```
show(ant)
```



Visualize the charge distribution on the Potter horn antenna in `log10` scale.

```
charge(ant,3.5e9,'Scale','log10')
```



Create Potter Horn Antenna with Step-Transition

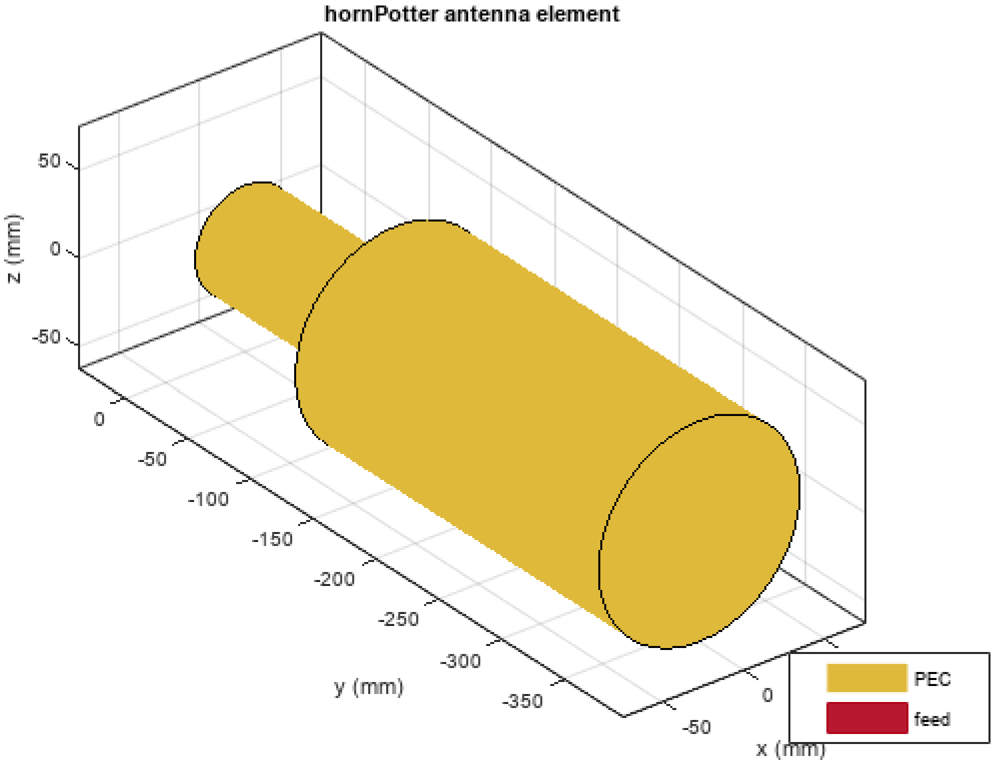
Create a Potter horn antenna with the taper height of 0 m. The Potter horn with taper height of 0 m is termed as step-transition Potter horn.

```
ant = hornPotter('TaperHeight',0)
```

```
ant = 
  hornPotter with properties:

             Radius: 0.0311
    WaveguideHeight: 0.1200
         FeedHeight: 0.0185
          FeedWidth: 5.0000e-04
         FeedOffset: 0.0300
         ConeHeight: 0.2414
     ApertureRadius: 0.0626
        TaperRadius: 0.0626
        TaperHeight: 0
          Conductor: [1x1 metal]
               Tilt: 0
          TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

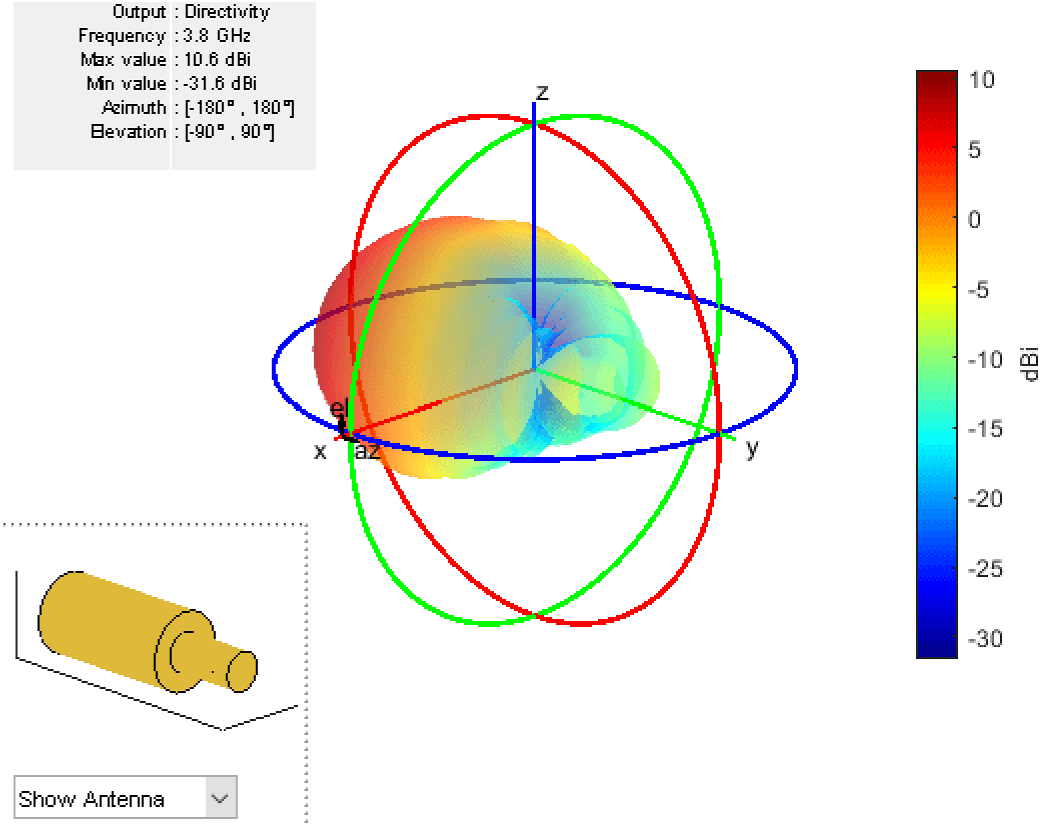
View the antenna using the show function.

```
show(ant)
```



Overlay the antenna on the radiation pattern.

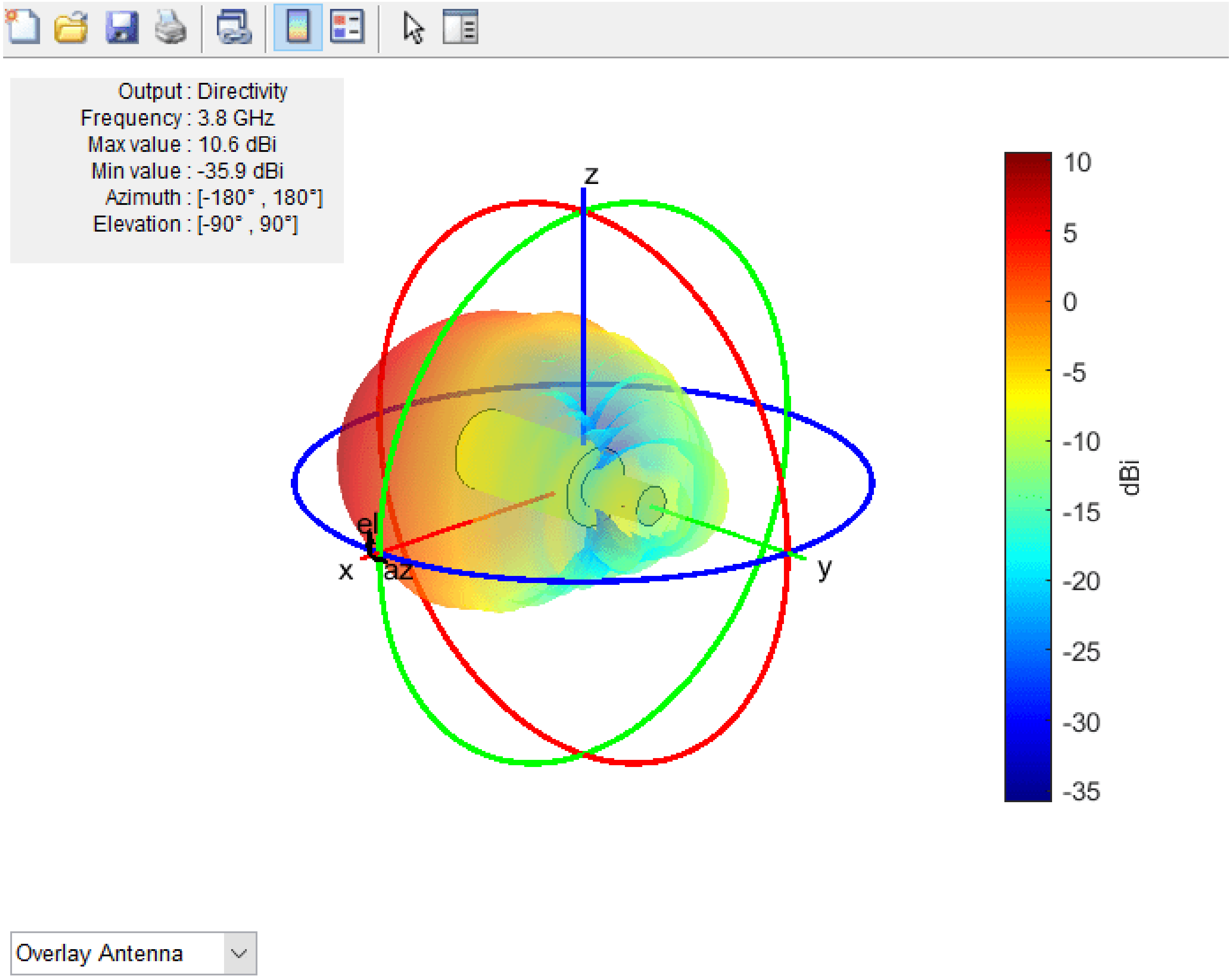
```
p = PatternPlotOptions;
p.Transparency = 0.5;
pattern(ant,3.8e9,'patternOptions',p)
```



```
     Output : Directivity
  Frequency : 3.8 GHz
  Max value : 10.6 dBi
  Min value : -31.6 dBi
    Azimuth : [-180° , 180°]
  Elevation : [-90° , 90°]
```

To understand the effect of Transparency, choose `Overlay Antenna` in the radiation pattern plot.

This option overlays the antenna on the radiation pattern.



Output : Directivity
Frequency : 3.8 GHz
Max value : 10.6 dBi
Min value : -35.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

More About

Parametric Analysis Guidelines

To understand how the properties of the `hornPotter` antenna object influence the antenna design, use these parametric analysis guidelines.

- To increase the operating frequency, decrease the “FeedHeight” on page 1-0 of the `hornPotter` antenna.
- To increase the gain, increase the radius of aperture of the `hornPotter` antenna object using the “ApertureRadius” on page 1-0 property.
- To increase s_{11} value, increase height of the waveguide using the “WaveguideHeight” on page 1-0 property.
- Gain obtained from tapered transition Potter horn is more than step-transition.

Version History

Introduced in R2021a

References

[1] Ahmed, Zobaer, Asif Zaman, and Lutfa Akter. “Parametric Analysis of Pickett Potter Horn Antenna.” In *2015 IEEE International Conference on Telecommunications and Photonics (ICTP)*, 1–4. Dhaka, Bangladesh: IEEE, 2015. https://doi.org/10.1109/ICTP.2015.7427926.

[2] Gohil, Shweta K, and Usha Neelakantan. “A Wideband Dual Mode Horn Antenna with Tapered Discontinuity.” In *International Research Journal of Engineering and Technology (IRJET)*, 02:913–16, 2015.

See Also

`hornScrimp` | `hornConical` | `horn` | `waveguideCircular`

Topics

“Rotate Antennas and Arrays”

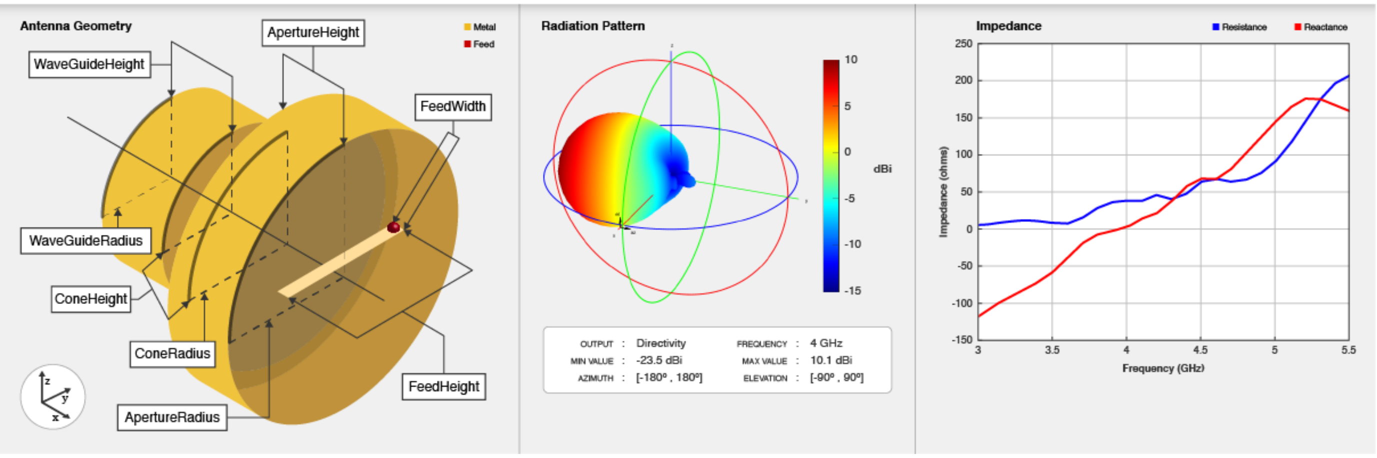
hornScrimp

Create Scrimp horn antenna

Description

The `hornScrimp` object creates a Scrimp horn antenna. Scrimp (short circular ring loaded horn with minimized cross-polarization) horn antenna is a short, axially corrugated horn antenna with a single slot. Using this antenna provides high aperture efficiency, low cross-polarization, and low voltage standing wave ratio (VSWR) over a broad frequency band. These antennas are used for navigation satellite feeder links in a medium earth orbit (MEO).

hornScrimp



Creation

Syntax

```
ant = hornScrimp
ant = hornScrimp(Name,Value)
```

Description

`ant = hornScrimp` creates Scrimp horn antenna with dimensions for an operating frequency of 4 GHz.

`ant = hornScrimp(Name,Value)` sets “Properties” on page 1-827 using one or more name-value pairs. For example, `hornScrimp('ConeHeight',0.05)` creates Scrimp horn antenna with the cone height of 0.05 meters.

Properties

Radius — Radius of circular waveguide
`0.02920` (default) | positive scalar

Radius of the circular waveguide, specified as a positive scalar in meters.

Example: `'Radius',0.0760`

Example: `ant.Radius = 0.0760`

Data Types: `double`

`WaveguideHeight` — Height of circular waveguide

`0.0250` (default) | positive scalar

Height of the circular waveguide, specified as a positive scalar in meters.

Example: `'WaveguideHeight',0.0340`

Example: `ant.WaveguideHeight = 0.0340`

Data Types: `double`

`FeedHeight` — Height of feed

`0.0175` (default) | positive scalar

Height of the feed, specified as a positive scalar in meters.

Example: `'FeedHeight',0.0085`

Example: `ant.FeedHeight = 0.0085`

Data Types: `double`

`FeedWidth` — Width of feed

`0.0003` (default) | positive scalar

Width of the feed, specified as a positive scalar in meters.

Example: `'FeedWidth',0.0200`

Example: `ant.FeedWidth = 0.0200`

Data Types: `double`

`FeedOffset` — Signed distance along waveguide height

`0.0200` (default) | real-valued scalar

Signed distance along the waveguide height, specified as a real-valued scalar in meters.

Example: `'FeedOffset',0.03627`

Example: `ant.FeedOffset = 0.03627`

Data Types: `double`

`ConeHeight` — Height of cone

`0.0362` (default) | positive scalar

Height of the cone, specified as a positive scalar in meters.

Example: `'ConeHeight',0.0540`

Example: `ant.ConeHeight = 0.0540`

Data Types: `double`

`ConeRadius` — Radius of cone

`0.0408` (default) | positive scalar

Radius of the cone, specified as a positive scalar in meters.

Example: `'ConeRadius',0.0540`

Example: `ant.ConeRadius = 0.0540`

Data Types: `double`

`ApertureRadius` — Radius of cone aperture

`0.0480` (default) | positive scalar

Radius of the cone aperture, specified as a positive scalar in meters.

Example: `'ApertureRadius',0.0760`

Example: `ant.ApertureRadius = 0.0760`

Data Types: `double`

`ApertureHeight` — Height of cone aperture

`0.0250` (default) | positive scalar

Height of the cone aperture, specified as a positive scalar in meters.

Example: `'ApertureHeight',0.0760`

Example: `ant.ApertureHeight = 0.0760`

Data Types: `double`

`StubHeight` — Height of stub

`0.0146` (default) | positive scalar

Height of the stub, specified as a positive scalar in meters.

Example: `'StubHeight',0.0760`

Example: `ant.StubHeight = 0.0760`

Data Types: `double`

`Conductor` — Type of metal material

`'PEC'` (default) | `metal` object

Type of the metal used as a conductor, specified as a metal material object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Meshing”.

Example: `m = metal('Copper'); 'Conductor',m`

Example: `m = metal('Copper'); ant.Conductor = m`

`TiltAxis` — Tilt axis of antenna

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `'TiltAxis',[0 1 0]`

Example: `'TiltAxis',[0 0 0;0 1 0]`

Example: `ant.TiltAxis = 'Z'`

`Tilt` — Tilt angle of antenna

`0` (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `'Tilt',90`

Example: `ant.Tilt = 90`

Example: `'Tilt',[90 90]`,`'TiltAxis',[0 1 0;0 1 1]` tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: `double`

Load — Lumped elements

[1x1 `lumpedElement`] (default) | `lumpedElement` object

Lumped elements added to the antenna feed, specified as a `lumpedElement` object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where `lumpedelement` is the load added to the antenna feed.

Example: `ant.Load = lumpedElement('Impedance',75)`

Object Functions

show	Display antenna, array structures or shapes
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array

patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
vswr	Voltage standing wave ratio of antenna

Examples

Create Default Scrimp Horn Antenna and Plot Radiation Pattern

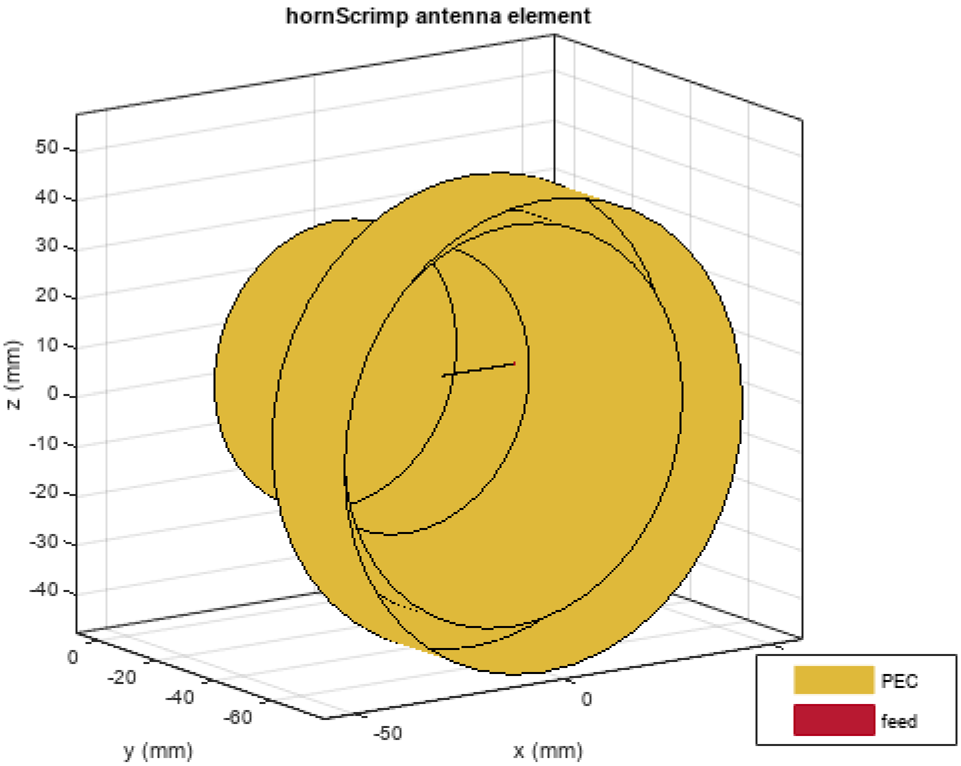
Create Scrimp horn antenna with default properties.

```
ant = hornScrimp
```

```
ant = 
  hornScrimp with properties:

             Radius: 0.0292
    WaveguideHeight: 0.0250
         FeedHeight: 0.0175
          FeedWidth: 3.0000e-04
         FeedOffset: 0.0200
         ConeHeight: 0.0362
         ConeRadius: 0.0408
     ApertureRadius: 0.0480
     ApertureHeight: 0.0250
         StubHeight: 0.0146
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

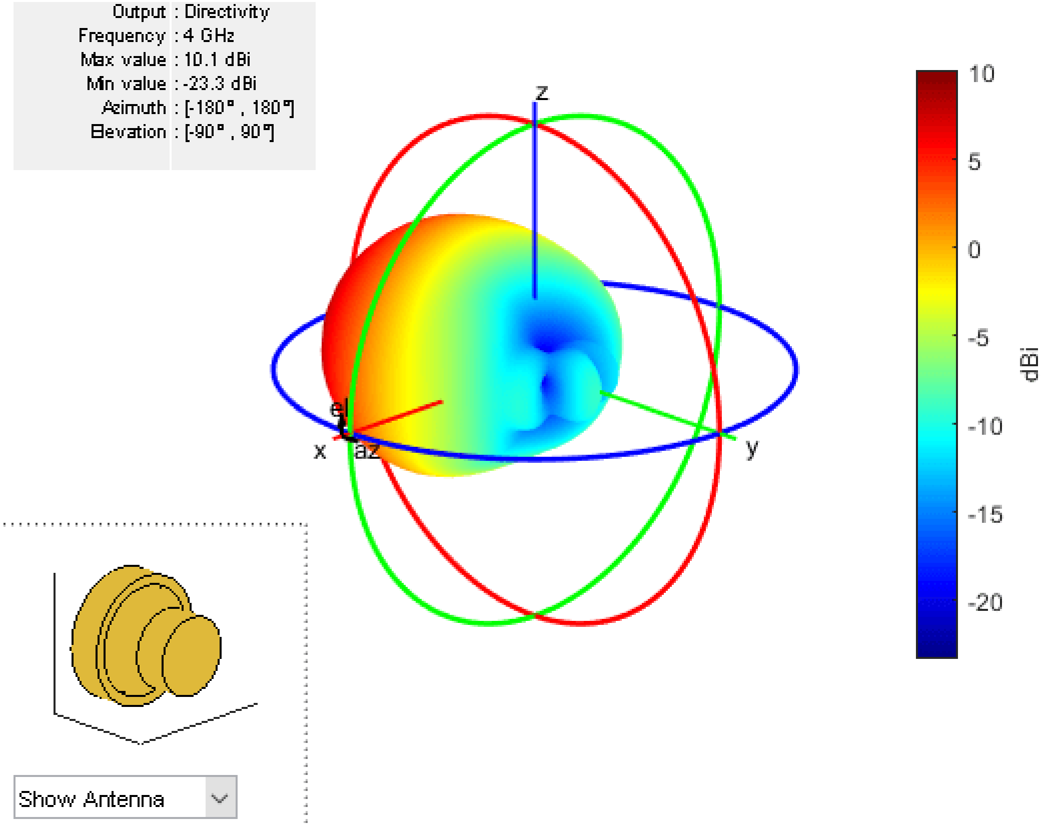
View the antenna using the `show` function.

```
show(ant)
```



Plot the radiation pattern of the Scrimp horn antenna at a frequency of 4 GHz.

```
pattern(ant,4e9)
```



Output : Directivity
Frequency : 4 GHz
Max value : 10.1 dBi
Min value : -23.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

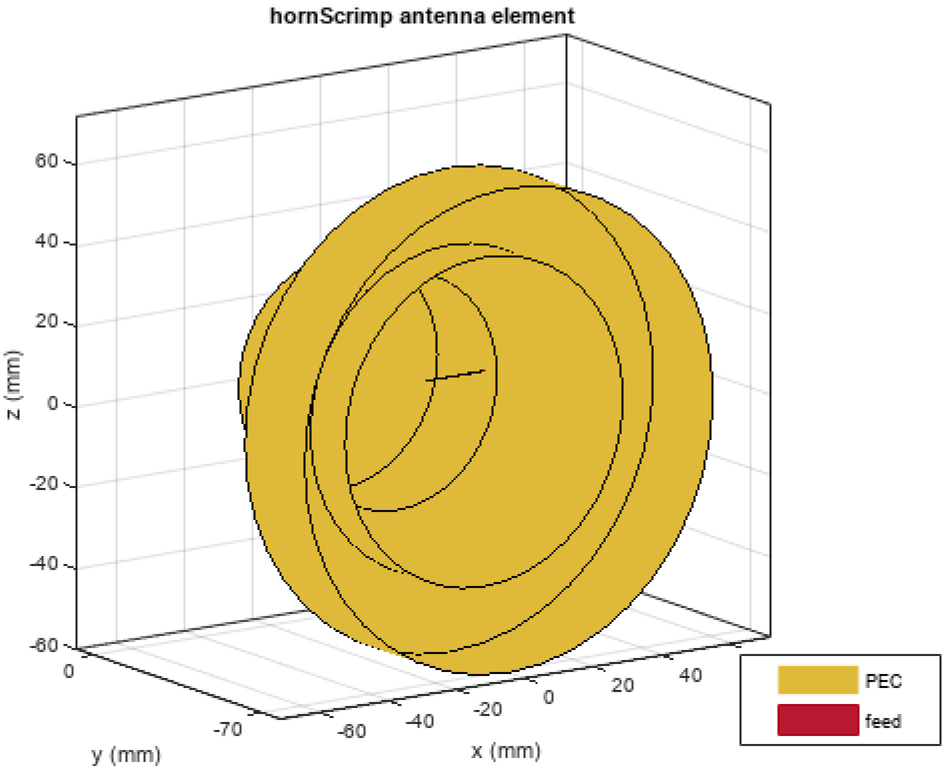
Plot S-Parameter of Scrimp Horn Antenna

Create Scrimp horn antenna with the aperture radius of 0.06 meters.

```
ant = hornScrimp;
ant.ApertureRadius = 0.06;
```

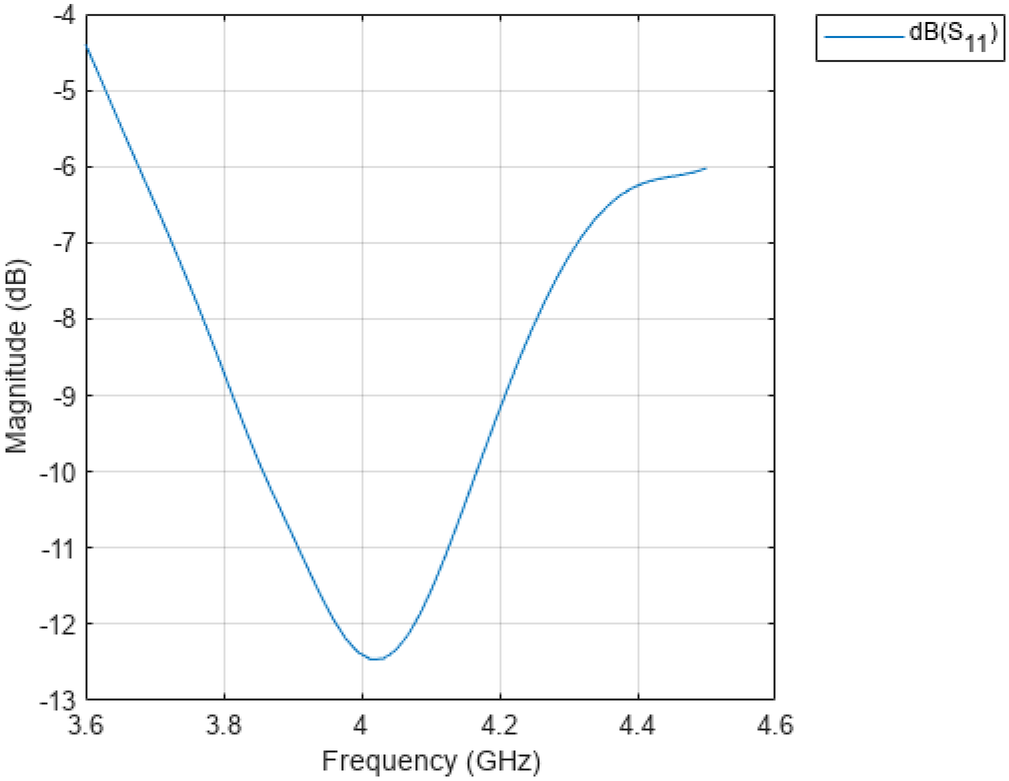
Visualize the antenna using the `show` function.

```
show(ant)
```



Plot the S-parameters over a frequency range of 3.6 GHz to 4.5 GHz.

```
s = sparameters(ant,linspace(3.6e9,4.5e9,51));
rfplot(s)
```



More About

Parametric Analysis Guidelines

To understand how the properties of the `hornScrimp` antenna object influence the antenna design, use these parametric analysis guidelines.

- To increase the operating frequency, decrease the “FeedHeight” on page 1-0 of the `hornScrimp` antenna.
- To increase the gain, increase the “ApertureRadius” on page 1-0 , “ApertureHeight” on page 1-0 , and “StubHeight” on page 1-0 of the `hornScrimp` antenna object.

Version History

Introduced in R2021a

References

[1] Muhammad, S. A., A. Rolland, S. H. Dahlan, R. Sauleau, and H. Legay. “Hexagonal-Shaped Broadband Compact Scrimp Horn Antenna for Operation in C-Band.” *IEEE Antennas and Wireless Propagation Letters* 11 (2012): 842–45.

[2] Muhammad, S., A. Rolland, S. H. Dahlan, R. Sauleau and H. Legay. “Comparison Between Scrimp Horns and Stacked Fabry-Perot Cavity Antennas with Small Apertures.” *2012 6th European Conference on Antennas and Propagation (EUCAP)* (2012): 817–820.

See Also

`hornPotter` | `hornConical` | `horn` | `waveguideCircular` | `metal`

Topics

“Rotate Antennas and Arrays”

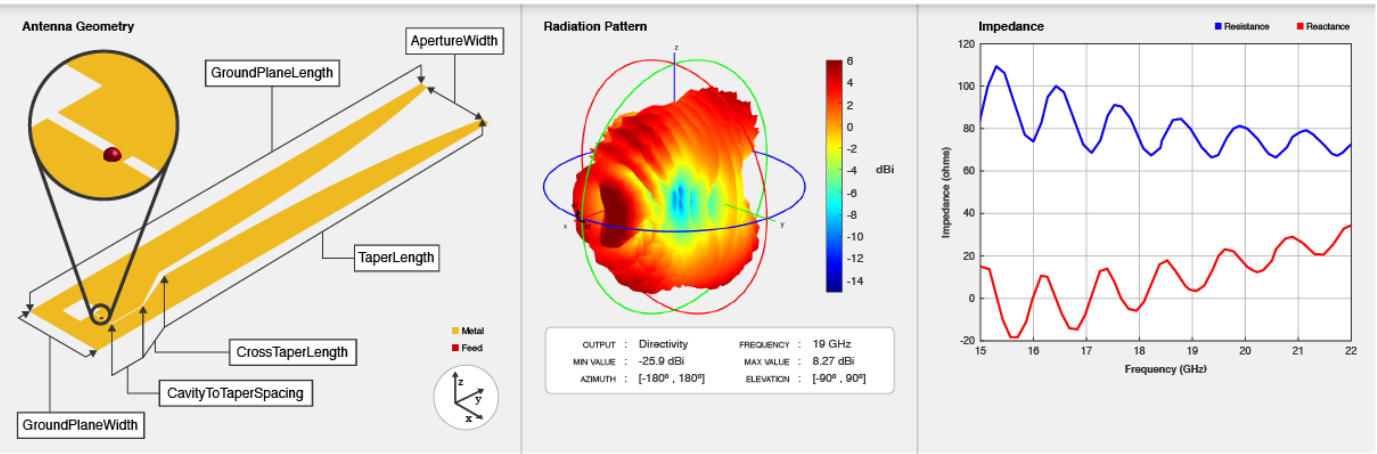
vivaldiOffsetCavity

Create Vivaldi antenna with rectangular or circular offset cavity

Description

The `vivaldiOffsetCavity` object creates a Vivaldi antenna with a rectangular or circular offset cavity on an exponential or linear taper ground plane. The Vivaldi offset cavity antenna has a metal structure which helps the antenna avoid the shortcomings of an antipodal Vivaldi antenna like large microstrip loss, complex installation, and integration. The wideband characteristics of a Vivaldi offset cavity antenna make it suitable for ultra-wideband phased array applications used in aviation and aerospace technologies.

Vivaldi Offset Cavity



Creation

Syntax

```
vi = vivaldiOffsetCavity
vi = vivaldiOffsetCavity(Name,Value)
```

Description

`vi = vivaldiOffsetCavity` creates a default Vivaldi antenna with a rectangular offset cavity on an exponential taper ground plane. By default, the antenna operates at a frequency range of 16–21 GHz and is located in the *xy*- plane.

`vi = vivaldiOffsetCavity(Name,Value)` sets “Properties” on page 1-838 using one or more name-value pairs. For example, `ant = vivaldiOffsetCavity('CavityShape','Circular')` creates a Vivaldi antenna with a circular offset cavity.

Properties

TaperLength — Taper length measured from cross taper end point
0.105 (default) | positive scalar

Taper length measured from the cross taper end point, specified as a positive scalar in meters.

Example: 'TaperLength',200e-3

ApertureWidth — Aperture width
0.02 (default) | positive scalar

Aperture width of the Vivaldi antenna, specified as a positive scalar in meters.

Example: 'ApertureWidth',3e-3

OpeningRate — Taper opening rate
25 (default) | positive scalar

Taper opening rate measured from cross taper end point, specified as a positive scalar. This property determines the rate at which the notch transitions from the TaperedSlotWidth to the aperture. When OpeningRate is 0, the notch has a linear profile creating a linear tapered slot. For other values, it has other values it has an exponential profile.

Example: 'OpeningRate',0.3

Data Types: double

TaperedSlotWidth — Width of tapered end
0.002 (default) | positive scalar

Width of the tapered end, specified as a positive scalar in meters.

Example: 'TaperedSlotWidth',0.003

Data Types: double

CrossTaperLength — Length of cross taper
0.0131 (default) | positive scalar

Length of the cross taper, specified as a positive scalar in meters.

Example: 'CrossTaperLength',2

Data Types: double

TaperOffset — Signed distance from mid-TaperedSlotWidth along *y*-axis
-0.0063 (default) | real-valued scalar

Signed distance from mid-TaperedSlotWidth along the *y*-axis, specified as a real-valued scalar in meters.

Example: 'TaperOffset',0.03

Data Types: double

SlotLineWidth — Width of slot line
5e-04 (default) | scalar

Width of the slot line, specified as a scalar in meters.

Example: 'SlotLineWidth',0.0002

Data Types: double

CavityToTaperSpacing — Transition distance from cavity to cross taper

0.0107 (default) | positive scalar

Transition distance from the cavity to the cross taper, specified as a scalar in meters.

Example: 'CavityToTaperSpacing',0.003

Data Types: double

CavityShape — Shape of cavity

'Rectangular' (default) | 'Circular'

Shape of the cavity, specified as a character array. The dimensions of cavity can be modified using "Circular Cavity" on page 1-0 and "Rectangular Cavity" on page 1-0 properties.

Example: 'CavityShape','Circular'

Data Types: char

CavityOffset — Distance from feedpoint of antenna along X and Y direction

[0.0048 0.0030] (default) | two-element positive vector

Distance from the feed point of the antenna along X and Y direction, specified as a two-element vector in meters. The first element of the vector is the distance from the feed point to the left edge of the cavity along the *x*-axis. The second element of the vector is the distance from the feed point to the bottom edge of the cavity along the *y*-axis.

Example: 'CavityOffset',[0.01 0.01]

Data Types: double

GroundPlaneLength — Ground plane length

0.1400 (default) | positive scalar

Ground plane length, specified as a positive scalar in meters. By default, the ground plane length is measured along the *x*-axis.

Example: 'GroundPlaneLength',2

Data Types: double

GroundPlaneWidth — Ground plane width

0.0240 (default) | positive scalar

Ground plane width, specified as a positive scalar in meters. By default, ground plane width is measured along the y-axis.

Example: 'GroundPlaneWidth',4

Data Types: double

FeedOffset — Signed distance from line of symmetry of GroundPlaneWidth to feed point

-0.0030 (default) | real-valued scalar

Signed distance from the line of symmetry of the GroundPlaneWidth to the feed point, specified as a real-valued scalar in meters.

Example: 'FeedOffset',0.001

Data Types: double

Conductor — Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal object. You can choose any metal from the MetalCatalog or specify a metal of your choice. For more information, see metal. For more information on metal conductor meshing, see “Meshing”.

Example: m = metal('Copper'); 'Conductor',m

Example: m = metal('Copper'); ant.Conductor = m

Tilt — Tilt angle of antenna

0 (default) | scalar | vector

Tilt angle of the antenna, specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: 'Tilt',90

Example: ant.Tilt = 90

Example: 'Tilt',[90 90],'TiltAxis',[0 1 0;0 1 1] tilts the antenna at 90 degrees about the two axes defined by the vectors.

Data Types: double

TiltAxis — Tilt axis of antenna

[1 0 0] (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | 'X' | 'Y' | 'Z'

Tilt axis of the antenna, specified as:

- Three-element vector of Cartesian coordinates in meters. In this case, each coordinate in the vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the antenna rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: 'TiltAxis',[0 1 0]

Example: 'TiltAxis',[0 0 0;0 1 0]

Example: ant.TiltAxis = 'Z'

Load — Lumped elements

[1x1 lumpedElement] (default) | lumpedElement object

Lumped elements added to the antenna feed, specified as a lumpedElement object. You can add a load anywhere on the surface of the antenna. By default, the load is at the feed. For more information, see lumpedElement.

Example: 'Load',lumpedelement, where lumpedelement is the load added to the antenna feed.

Example: ant.Load = lumpedElement('Impedance',75)

Circular Cavity

`CavityDiameter` — Circular cavity diameter

`0.0100` (default) | positive scalar

Circular cavity diameter, specified as a positive scalar in meters. This property is visible if `CavityShape` is set `'Circular'`.

Example: `'CavityDiameter',0.05`

Data Types: `double`

Rectangular Cavity

`CavityLength` — Length of rectangular cavity

`0.0073` (default) | positive scalar

Length of the rectangular cavity, specified as a positive scalar in meters. This property is visible if `CavityShape` is set `'Rectangular'`.

Example: `'CavityLength',0.003`

Data Types: `double`

`CavityWidth` — Width of rectangular cavity

`0.0066` (default) | positive scalar

Width of the rectangular cavity, specified as a positive scalar in meters. This property is visible if `CavityShape` is set `'Rectangular'`.

Example: `'CavityWidth',0.002`

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

vswr	Voltage standing wave ratio of antenna

Examples

Create Default Vivaldi Antenna with Offset Cavity and Plot its Radiation Pattern

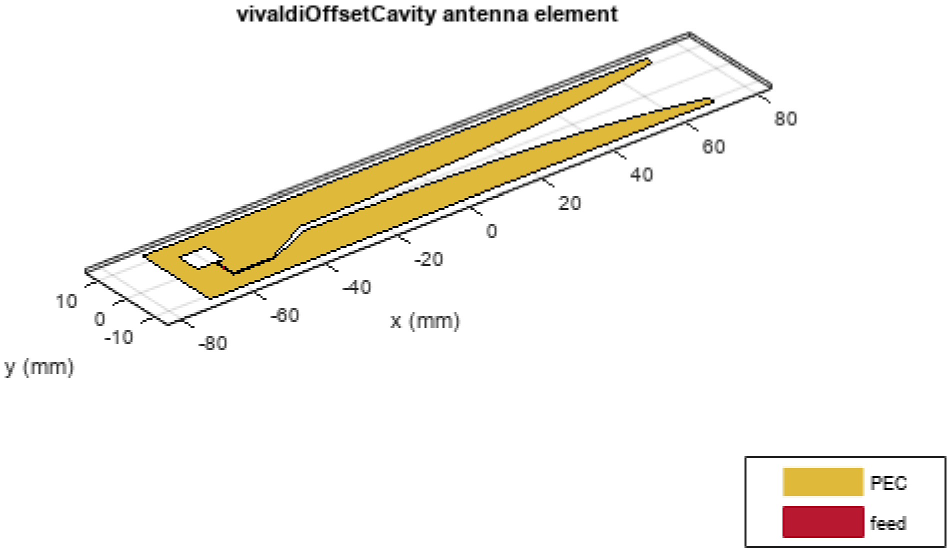
Create a default Vivaldi antenna with an offset cavity.

```
ant = vivaldiOffsetCavity
```

```
ant = 
  vivaldiOffsetCavity with properties:

             TaperLength: 0.1050
           ApertureWidth: 0.0200
             OpeningRate: 25
        TaperedSlotWidth: 0.0020
        CrossTaperLength: 0.0131
             TaperOffset: -0.0063
           SlotLineWidth: 5.0000e-04
    CavityToTaperSpacing: 0.0107
             CavityShape: 'Rectangular'
            CavityLength: 0.0073
             CavityWidth: 0.0066
            CavityOffset: [0.0048 0.0030]
       GroundPlaneLength: 0.1400
        GroundPlaneWidth: 0.0240
              FeedOffset: -0.0030
               Conductor: [1x1 metal]
                    Tilt: 0
                TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
```

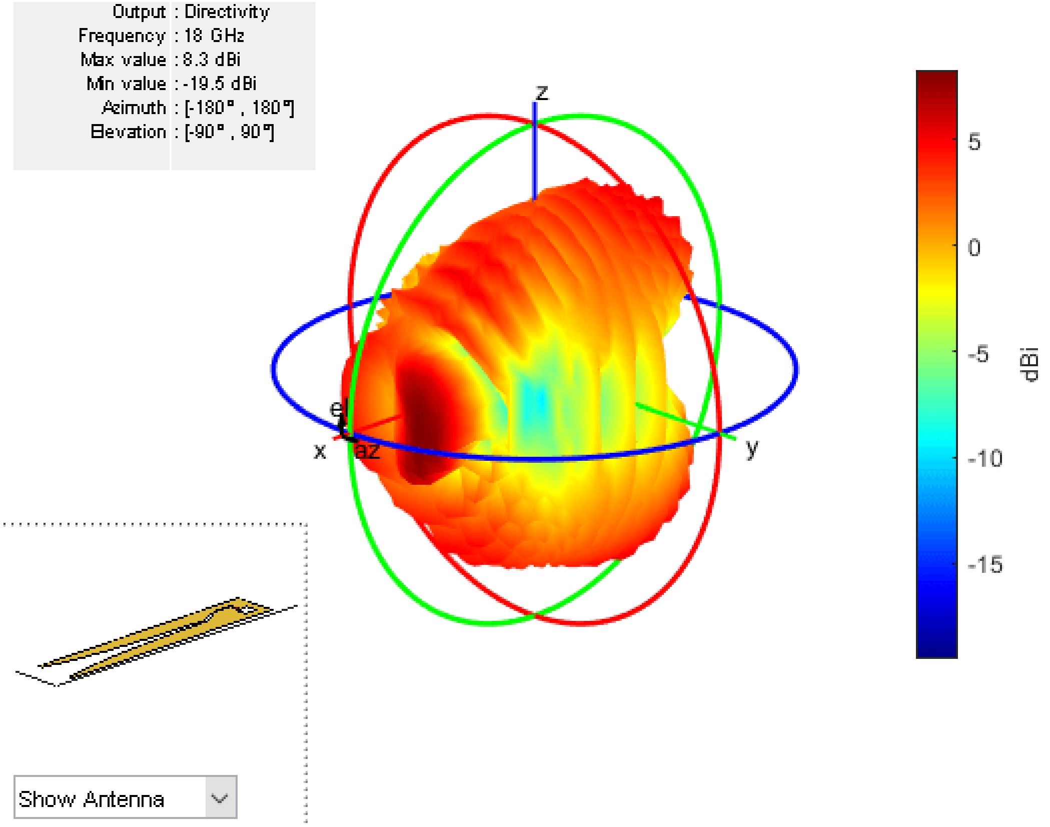
View the antenna using `show` function.

```
show(ant)
```



Plot the radiation pattern of the antenna at a frequency of 18 GHz.

```
pattern(ant,18e9)
```



Create Vivaldi Antenna with Offset Cavity and Plot its Radiation Pattern

Create Vivaldi antenna with offset rectangular cavity.

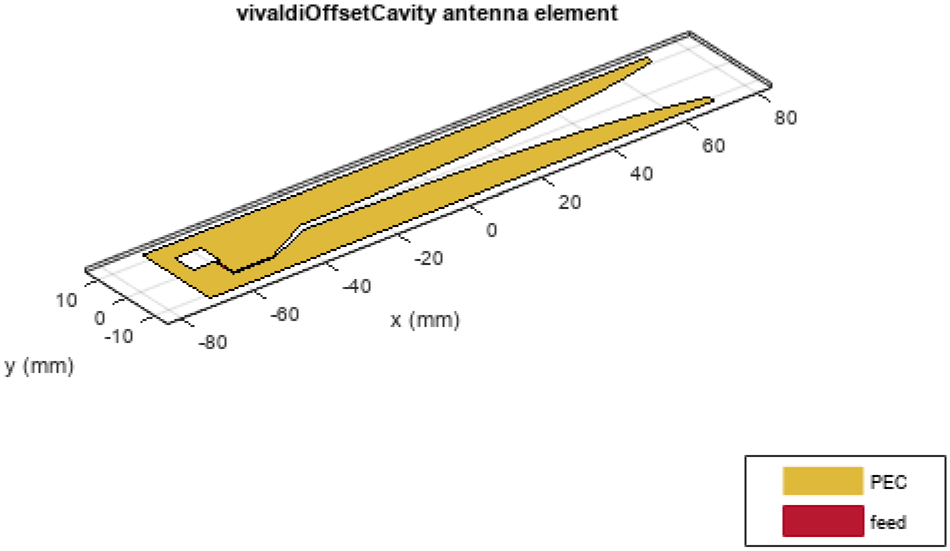
```
ant = vivaldiOffsetCavity('CavityOffset',[0.006 0.003])
```

```
ant = 
  vivaldiOffsetCavity with properties:

              TaperLength: 0.1050
            ApertureWidth: 0.0200
              OpeningRate: 25
          TaperedSlotWidth: 0.0020
          CrossTaperLength: 0.0131
              TaperOffset: -0.0063
            SlotLineWidth: 5.0000e-04
    CavityToTaperSpacing: 0.0107
              CavityShape: 'Rectangular'
             CavityLength: 0.0073
              CavityWidth: 0.0066
             CavityOffset: [0.0060 0.0030]
        GroundPlaneLength: 0.1400
         GroundPlaneWidth: 0.0240
               FeedOffset: -0.0030
                Conductor: [1x1 metal]
```

```
                    Tilt: 0
                TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
```

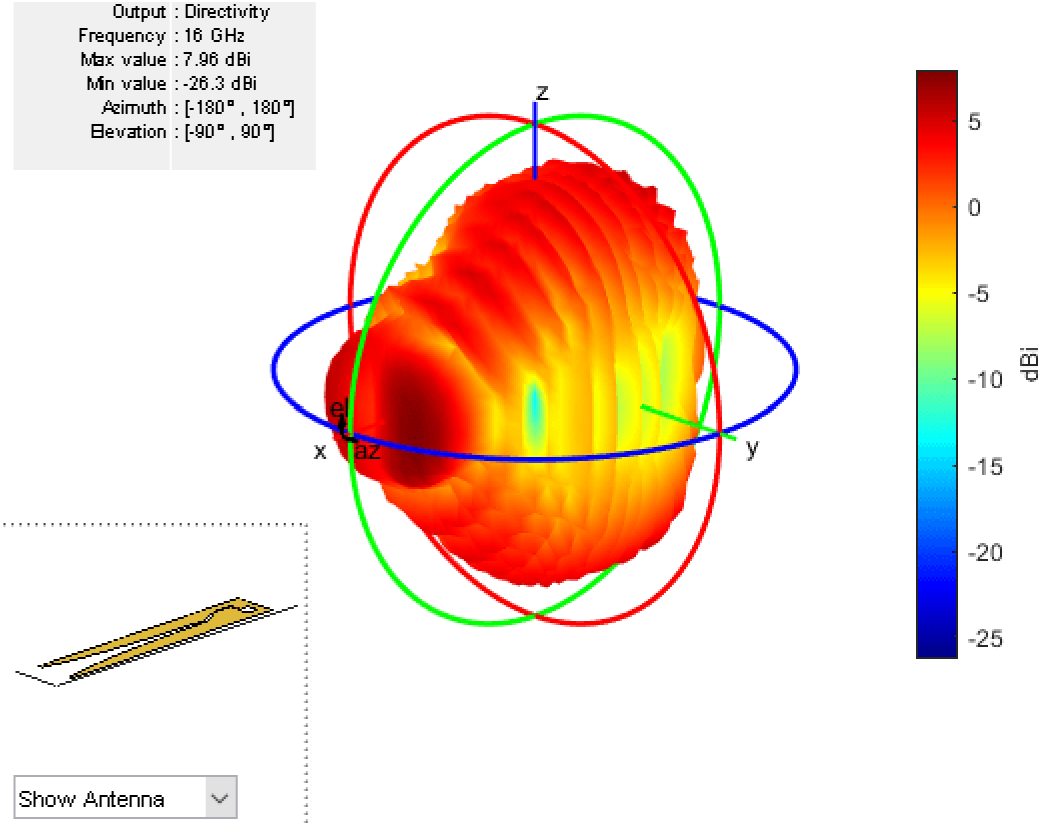
View the antenna using show function.

```
show(ant)
```



Plot radiation pattern of the antenna at a frequency of 16 GHz.

```
pattern(ant,16e9)
```

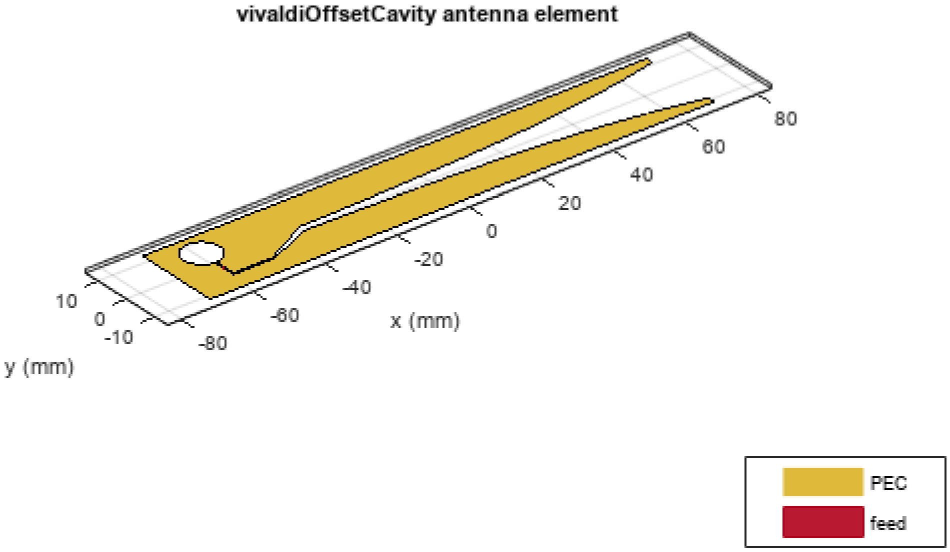


Output : Directivity
Frequency : 16 GHz
Max value : 7.96 dBi
Min value : -26.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Vivaldi Antenna with Circular Cavity and Plot its Return Loss

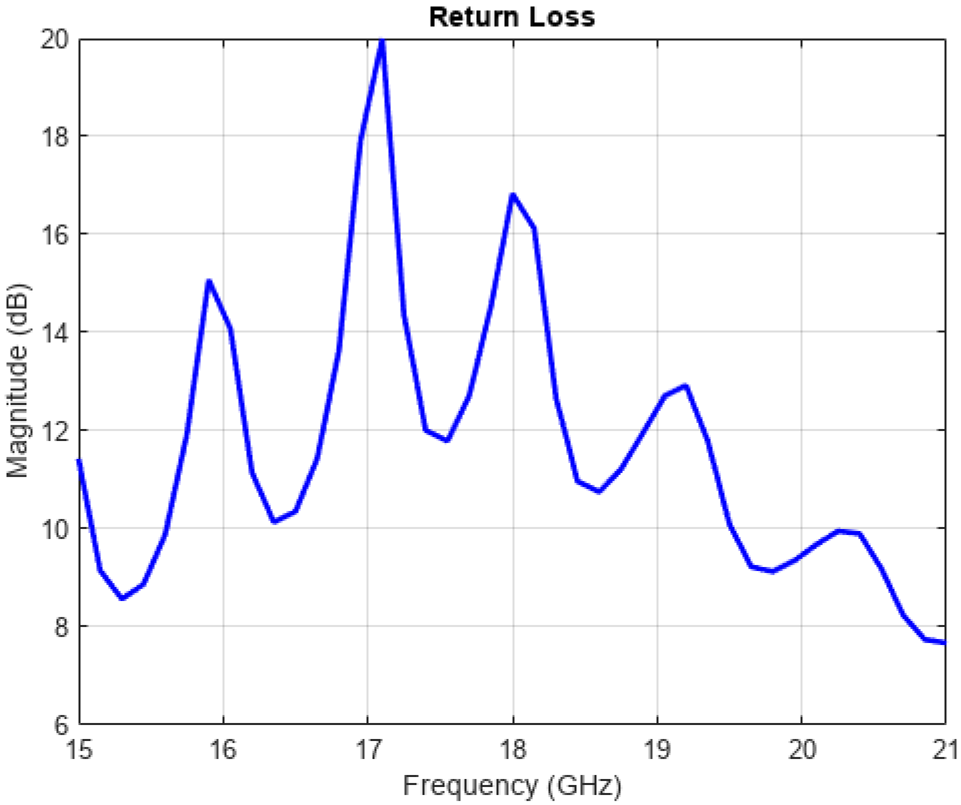
Create and visualize a Vivaldi antenna with a circular cavity.

```
ant = vivaldiOffsetCavity('CavityShape','Circular');
show(ant)
```



Plot the return loss of the antenna over a frequency range of 15 GHz - 20 GHz.

```
returnLoss(ant,linspace(15e9,21e9,41))
```



Version History

Introduced in R2021a

References

[1] X. Ma, S. Chai, K. Xiao and L. Ding. "Design of All-Metal Vivaldi Phased Array Antenna," *IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*, 2018, pp. 547-551, doi: 10.1109/SIPROCESS.2018.8600487.

[2] C. L. Prasanna, M. Bhagya Lakshmi, and N. N. Sastry. "A Parametric Analysis & Design of All Metal Vivaldi Antenna Covering 3.0-18 GHz for DF and Phased Array Applications," *Progress In Electromagnetics Research C*, vol. 92 (April 2019): pp. 57-69. doi:10.2528/PIERC19020601

See Also

`vivaldi` | `vivaldiAntipodal` | `eggCrate`

Topics

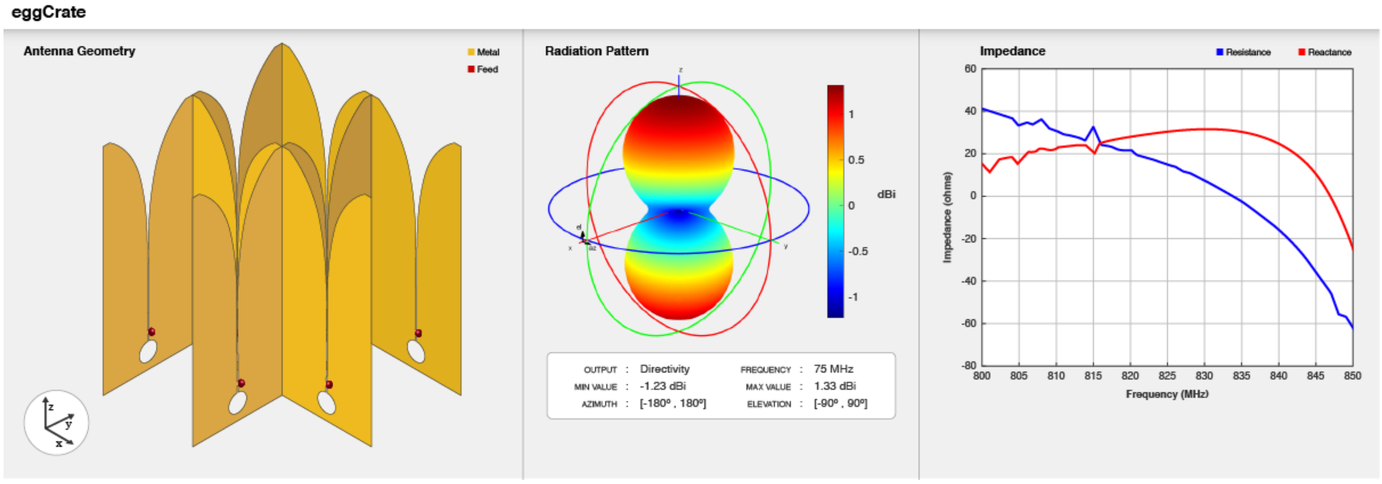
“Rotate Antennas and Arrays”

eggCrate

Create egg crate array of Vivaldi antenna elements

Description

The eggCrate object creates an array of Vivaldi antenna elements arranged in a rectangular egg-crate structure. Egg crate arrays are used in phased array applications in radar systems.



OUTPUT : Directivity
MIN VALUE : -1.23 dBi
AZIMUTH : [-180º , 180º]
FREQUENCY : 75 MHz
MAX VALUE : 1.33 dBi
ELEVATION : [-90º , 90º]

Creation

Syntax

```
array = eggCrate
array = eggCrate(Name,Value)
```

Description

`array = eggCrate` creates an array of Vivaldi antenna elements arranged in a rectangular egg-crate structure in the X-Y plane at an operating frequency of 825 MHz.

`array = eggCrate(Name,Value)` sets additional Properties using name-value pairs. For example, `arr = eggCrate('Element', vivaldiOffsetCavity)` creates an egg-crate array of offset Vivaldi antenna elements.

Properties

`Element` — Vivaldi antenna

`vivaldi` (default) | `vivaldiOffsetCavity`

Vivaldi antenna, specified as a `vivaldi` or `vivaldiOffsetCavity` object..

Example: `'Element',vivaldi`

`Size` — Number of rows and columns in array

`[2 2]` (default) | two-element vector

Number of rows and columns in the egg crate array, specified as a two-element vector.

Note You can use “NumElements” on page 1-857 to determines number of Vivaldi antenna elements in the egg-crate array.

Example: `'Size',[4 4]`

Data Types: `double`

`Gap` — Spacing between Vivaldi antenna elements

`[0 0]` (default) | two-element vector

Spacing between the Vivaldi antenna elements, specified as a two-element vector with each element in meters. The first element in the vector represents the spacing between the Vivaldi elements along x-axis. The second element represents the spacing between the Vivaldi elements along y-axis.

Example: `'Gap',[0.1 0.2]`

Data Types: `double`

`FeedVoltage` — Voltage applied to feed

`1` (default) | positive scalar | real positive vector

Magnitude of the voltage applied to the feed, specified as a as a positive scalar or a vector of positive elements in volts. If you specify a vector, the vector should be of the same size as `'NumElements'`.

Example: `'FeedVoltage',2`

Data Types: `double`

`FeedPhase` — Phase shift for each element in array

`0` (default) | real scalar | real vector

Phase shift for each element in the array, specified as a real scalar or a vector of real elements in degrees. If you specify a vector, the vector should be of the same size as `'NumElements'`.

Example: `'FeedPhase',-12`

Data Types: `double`

`Tilt` — Tilt angle of array

`0` (default) | scalar | vector

Tilt angle of the array specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`,

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the array at 90 degrees about the two axes, defined by vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of array

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the array, specified as:

- Three-element vectors of Cartesian coordinates in meters. In this case, each vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the array rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis='Z'`

Data Types: `double`

`SolverType` — Solver for antenna analysis

`'MoM'` (default) | `'MoM-PO'` | `'FMM'`

Solver for antenna analysis, specified as the comma-separated pair consisting of `'SolverType'` and `'MoM-PO'` or `'MoM'` (Method of Moments) or `'FMM'` (Fast Multipole Method).

Example: `'SolverType'`,`'MOM'`

Data Types: `char`

Object Functions

show	Display antenna, array structures or shapes
solver	Access FMM solver for electromagnetic analysis
info	Display information about antenna or array
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

Examples

Plot Radiation Pattern of Default Egg-Crate Array

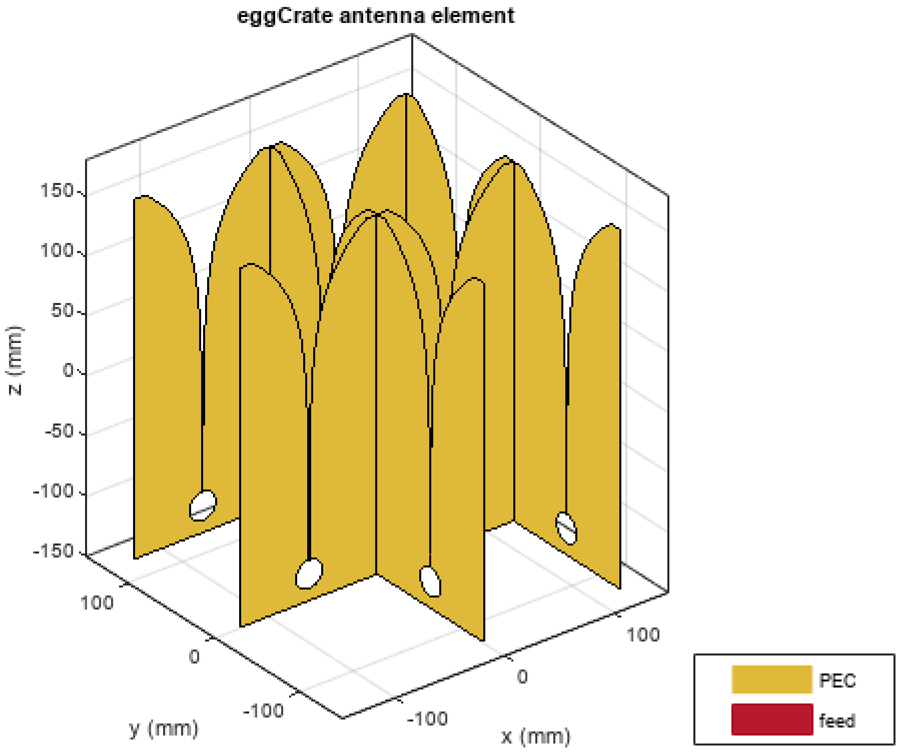
Create a default egg-crate array.

```
eca = eggCrate
```

```
eca = 
  eggCrate with properties:

        Element: [1x1 vivaldi]
           Size: [2 2]
            Gap: [0 0]
    FeedVoltage: 1
      FeedPhase: 0
           Tilt: 0
       TiltAxis: [1 0 0]
     SolverType: 'MoM'
```

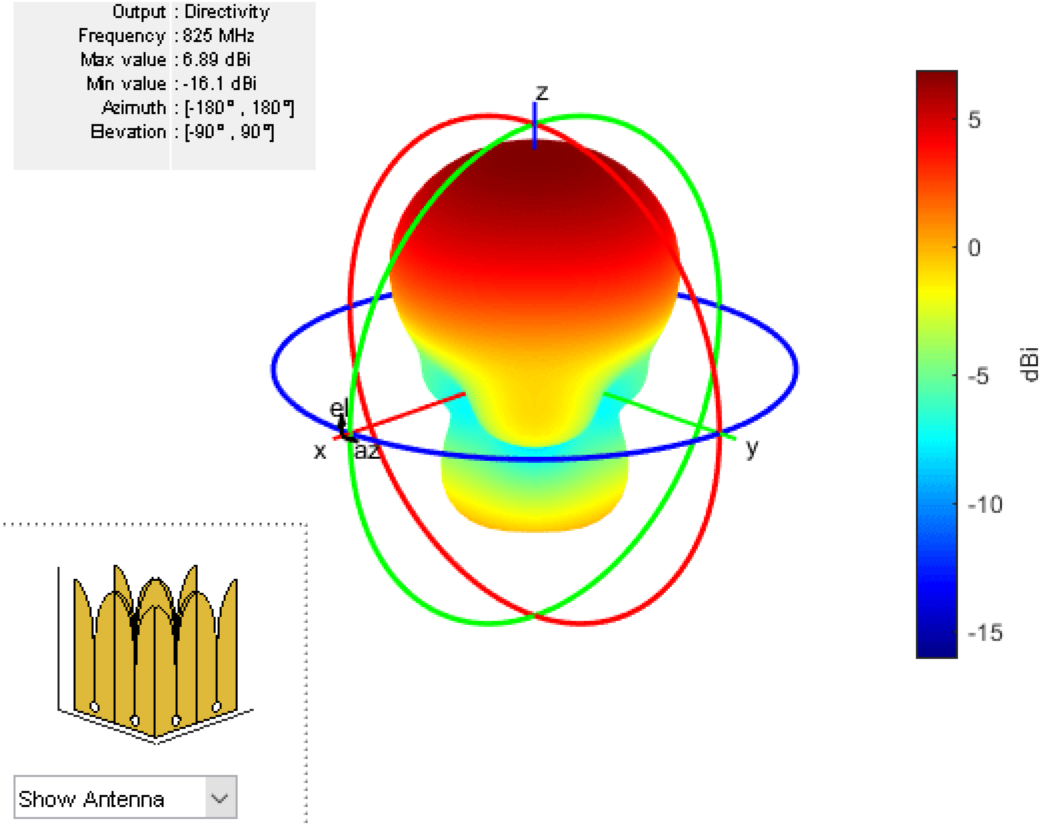
View the array using the `show` function.

```
show(eca)
```



Plot the radiation pattern of the egg-crate array at a frequency of 825 MHz.

```
pattern(eca,825e6)
```



```
Output : Directivity
Frequency : 825 MHz
Max value : 6.89 dBi
Min value : -16.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Create Egg-Crate Array and Plot Radiation Pattern

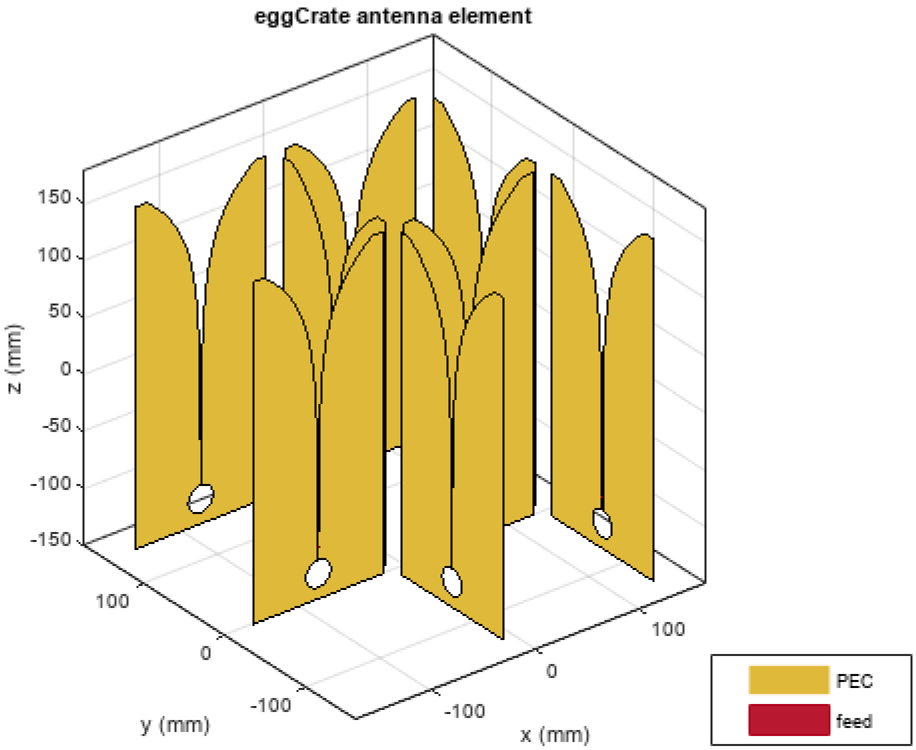
Create a 2-by-2 egg-crate array with a spacing of 20 mm between the Vivaldi elements.

```
eca = eggCrate('Size',[2 2],'Gap',[20e-3 20e-3])
```

```
eca = 
  eggCrate with properties:

        Element: [1x1 vivaldi]
           Size: [2 2]
            Gap: [0.0200 0.0200]
    FeedVoltage: 1
      FeedPhase: 0
           Tilt: 0
       TiltAxis: [1 0 0]
     SolverType: 'MoM'
```

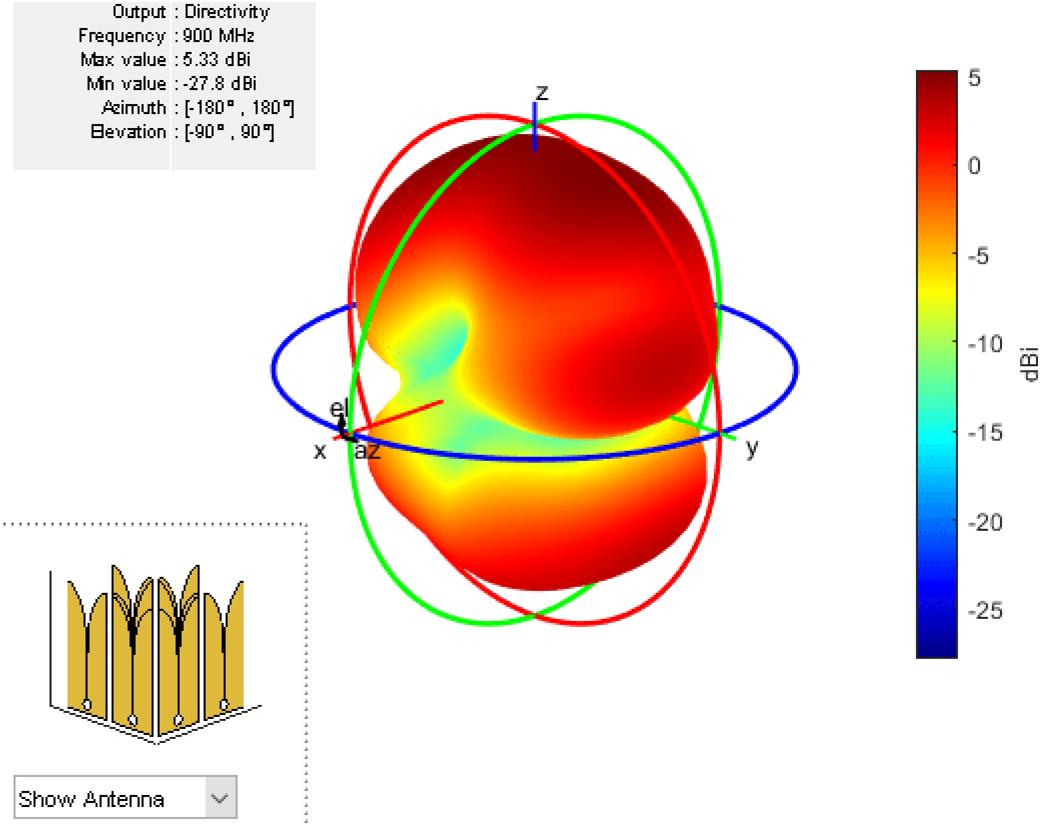
View the array using the `show` function.

```
show(eca)
```



Plot the radiation pattern of the array at a frequency of 0.9 GHz.

```
pattern(eca,0.9e9)
```



Output : Directivity
Frequency : 900 MHz
Max value : 5.33 dBi
Min value : -27.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Create Egg Crate Array and Plot Impedance of all Elements

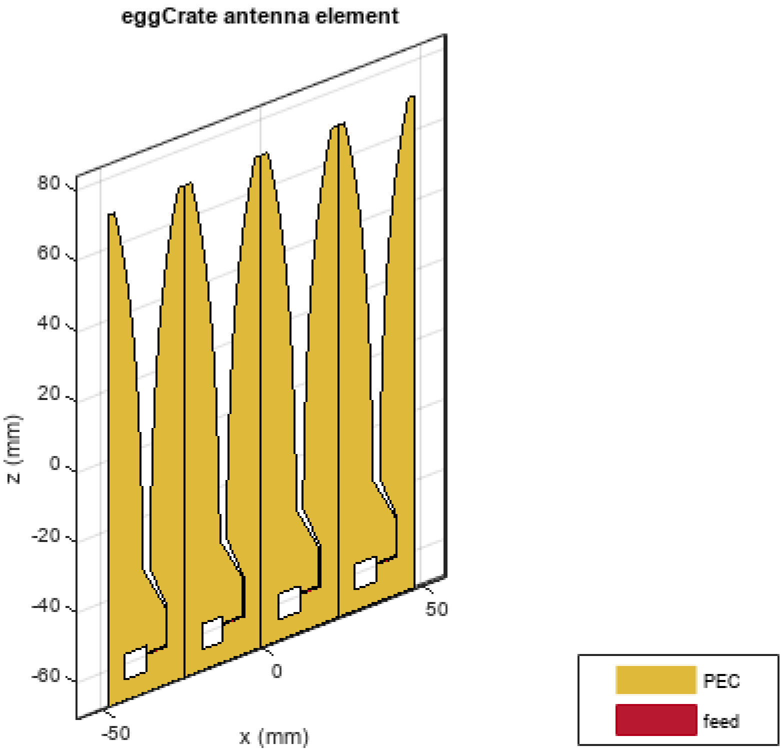
Create a 1-by-4 egg-crate array with `vivaldiOffsetCavity` elements.

```
eca = eggCrate('Element',vivaldiOffsetCavity,'Size',[1 4])
```

```
eca = 
  eggCrate with properties:

        Element: [1x1 vivaldiOffsetCavity]
           Size: [1 4]
            Gap: [0 0]
    FeedVoltage: 1
      FeedPhase: 0
           Tilt: 0
       TiltAxis: [1 0 0]
     SolverType: 'MoM'
```

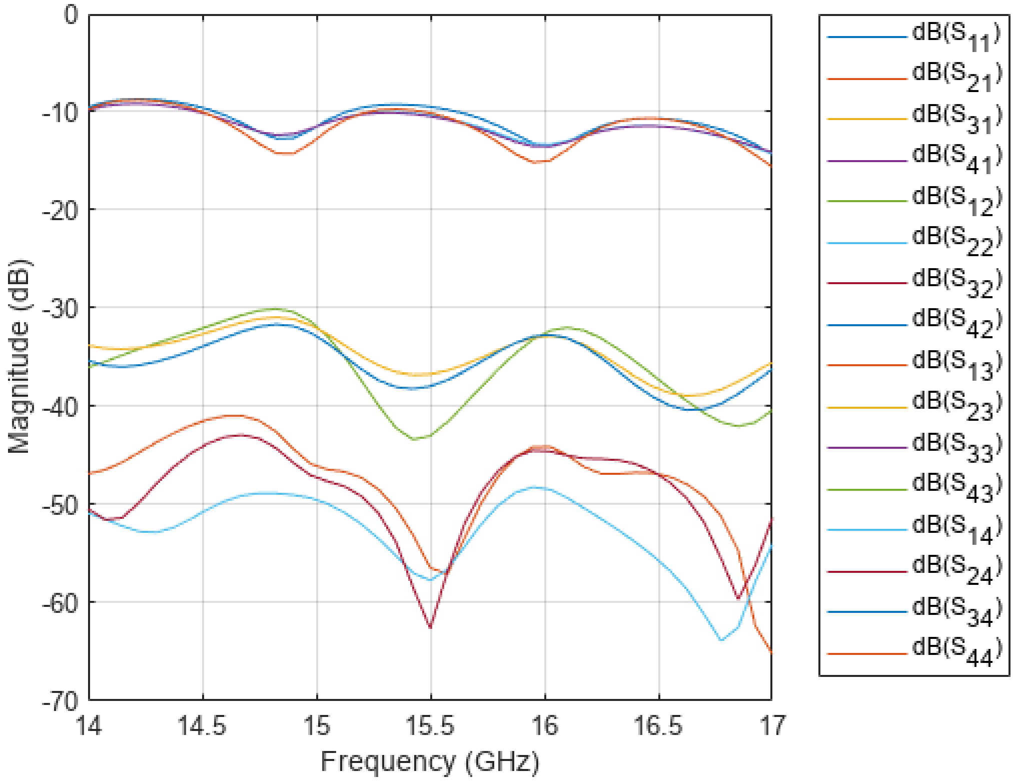
View array using the `show` function.

```
show(eca)
```



Plot the S-parameters of the array over a frequency range of 14--17 GHz.

```
s = sparameters(eca,linspace(14e9,17e9,41));
rfplot(s)
```



More About

NumElements

Determines number of antenna elements in egg crate array as a positive integer value. The `NumElements` for default egg crate array is 8.

Example: `N = arr.NumElements`

Version History

Introduced in R2021a

References

[1] Chu, Hao-Lung, Ghanshyam Mishra, and Satish K. Sharma. “Dual Polarized Wideband Vivaldi 4x4 Subarray Antenna Aperture for 5G Massive MIMO Panels with Simultaneous Multiple Beams.” In *2018 18th International Symposium on Antenna Technology and Applied Electromagnetics (ANTEM)*, 1–2. Waterloo, ON: IEEE, 2018. https://doi.org/10.1109/ANTEM.2018.8572871.

[2] R. Hahnel and D. Plettemeier, "Dual-polarized Vivaldi array for X- and Ku-Band," *Proceedings of the 2012 IEEE International Symposium on Antennas and Propagation*,2012, pp. 1-2.

[3] Yan, J., S. Gogineni, Bruno Camps-Raga and J. Brozena. “A Dual-Polarized 2–18-GHz Vivaldi Array for Airborne Radar Measurements of Snow.” *IEEE Transactions on Antennas and Propagation*, 2016, vol. 64, pp. 781-785.

See Also

`vivaldi` | `vivaldiOffsetCavity`

Topics

“Rotate Antennas and Arrays”

stlFileChecker

Detect and list bad features of STL files

Description

Use the `stlFileChecker` object to detect and list bad features in STL files. The bad features you can detect using the `stlFileChecker` object are:

- Non-manifold edges
- Non-manifold vertices
- Slivers
- Duplicate vertices
- Normal transition edges
- Free triangles
- T-vertices

Creation

Syntax

```
obj = stlFileChecker(filename)
obj = stlFileChecker(filename,Name=Value)
```

Description

`obj = stlFileChecker(filename)` detects bad features in the STL file specified by `filename` and lists the features in the `stlFileChecker` object.

`obj = stlFileChecker(filename,Name=Value)` sets "Input Properties" on page 1-860 using name-value arguments. For example, `stlFileChecker('s.stl',ShowLog=1)` displays a log of the bad features in file `s.stl`. You can specify multiple name-value arguments. Properties not specified retain their default values.

Input Arguments

`filename` — Name of STL file
character vector | string scalar

Name of the STL file, specified as a character vector or string scalar.

Example: `S = stlFileChecker('s1.stl')`

Properties

Input Properties

`ShowLog` — Display or hide log of bad features
`1` (default) | `0`

Display or hide a log of the bad features in the STL file, specified as `1` or `0`. Specify `1` to display the log of bad features. Specify `0` to hide the log of bad features.

Example: `ShowLog=0`

Data Types: `logical`

`MinimumSeparation` — Minimum separation between two distinct vertices
`1e-6` (default) | positive scalar

Minimum separation between two distinct vertices, specified as a positive scalar in meters. If the distance between two vertices is less than `MinimumSeparation`, then the object considers the vertices as duplicates. The smallest value that you can specify is `2.5e-7`.

Example: `MinimumSeparation=2e-6`

Data Types: `double`

`MinimumArea` — Minimum area of triangle
`1e-11` (default) | positive scalar

Minimum area of a triangle, specified as a positive scalar in square meters. If the area of a triangle is less than `MinimumArea`, then the object considers the triangle as a sliver. The smallest value that you can specify is `2.5e-12`.

Example: `MinimumArea=3e-10`

Data Types: `double`

Derived Properties

`NonManifoldEdges` — Edges shared by more than two triangles
positive scalar

This property is read-only.

Edges shared by more than two triangles, returned as a positive scalar.

Data Types: `double`

`NonManifoldVertices` — Vertex connected to two or more surfaces
positive scalar

This property is read-only.

Vertex connected to two or more surfaces, returned as a positive scalar. A pair of triangles with a common edge belong to the same surface.

Data Types: `double`

`Slivers` — Triangle with area less than `MinimumArea`
positive scalar

This property is read-only.

Triangle with area less than `MinimumArea`, returned as a positive scalar.

Data Types: `double`

`DuplicateVertices` — Vertices with separation less than `MinimumSeparation`
positive scalar

This property is read-only.

Vertices with separation less than `MinimumSeparation`, returned as a positive scalar.

Data Types: `double`

`NormalTransitionEdges` — Edges shared by triangles with normals in opposite directions
positive scalar

This property is read-only.

Edges shared by triangles with normals in opposite directions, returned as a positive scalar.

Data Types: `double`

`FreeTriangles` — Triangles with no shared vertex
positive scalar

This property is read-only.

Triangles with no shared vertex, returned as a positive scalar.

Data Types: `double`

`TVertices` — Edge connected to any point other than end point of triangle
positive scalar

This property is read-only.

Edge connected to any point other than the end point of a triangle, returned as a positive scalar.

Data Types: `double`

Examples

Detect and Visualize Bad Mesh Features

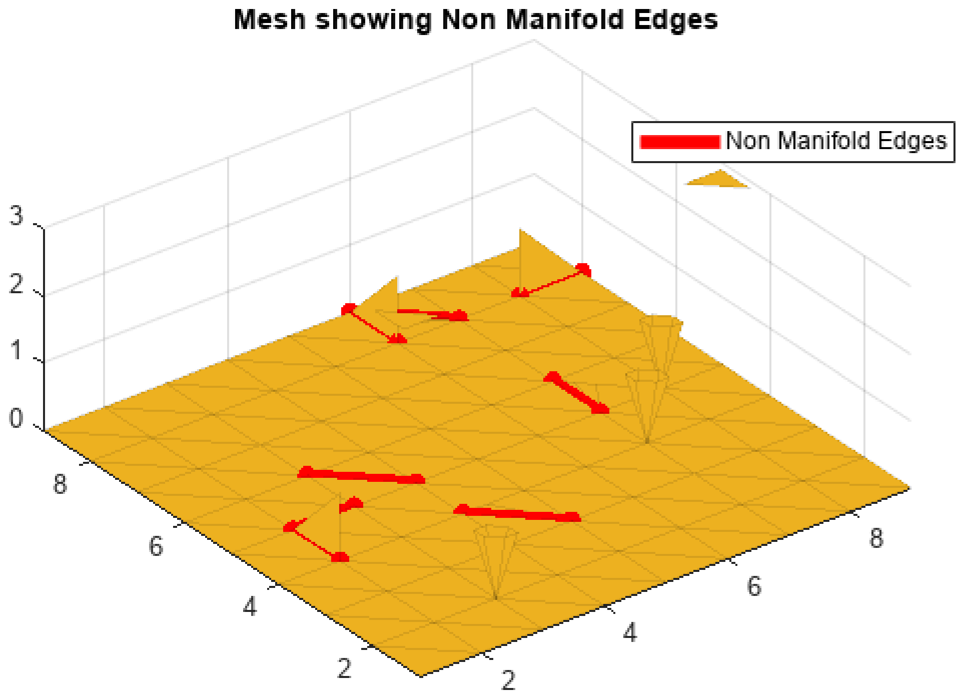
Use the `stlFileChecker` object to detect and display bad features in an STL file.

```
stlFilename = 'sample_stl_file.stl';
s = stlFileChecker(stlFilename)
```

```
detected 8 non manifold edges
detected 3 non manifold vertices
detected 0 duplicate Vertices
detected 0 Slivers
detected 12 Normal Transition Edges
detected 2 Free Triangles
detected 9 T-Vertices
detected 4 duplicate Vertices
detected 0 Slivers

s = 
  stlFileChecker with properties:

                 FileName: 'sample_stl_file.stl'
            Triangulation: [162x3 triangulation]
         NonManifoldEdges: [8x2 double]
      NonManifoldVertices: [3x1 double]
                  Slivers: [0x1 double]
        DuplicateVertices: [4x1 double]
    NormalTransitionEdges: [12x2 double]
            FreeTriangles: [2x1 double]
                TVertices: [9x1 double]
                  ShowLog: 1
        MinimumSeparation: 1.0000e-06
              MinimumArea: 1.0000e-11
```

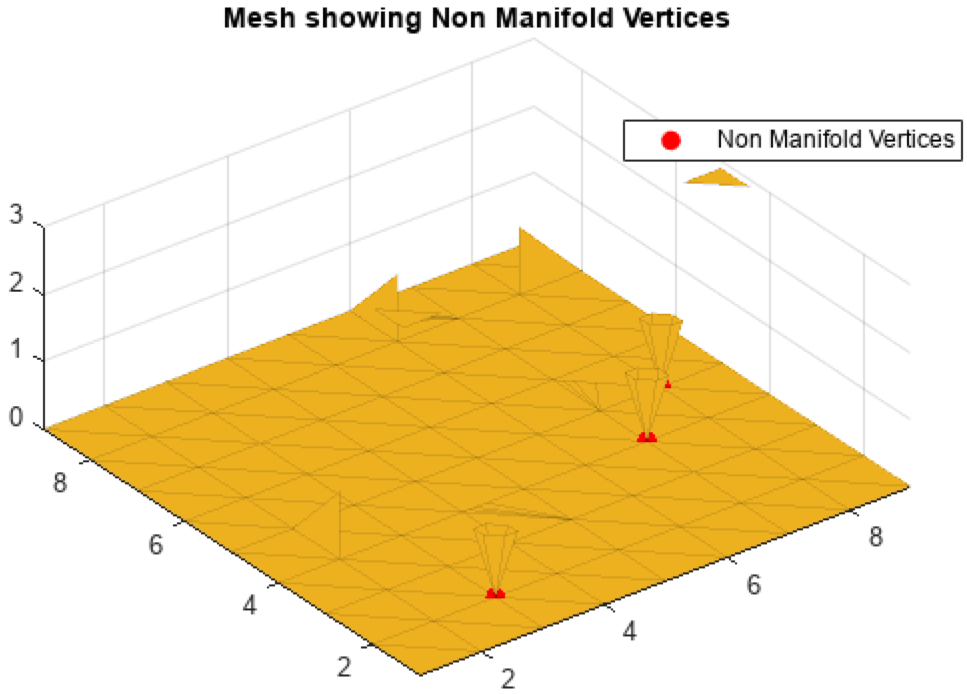
Visualize the non-manifold edges in the STL file.

```
showNonManifoldEdges(s)
```



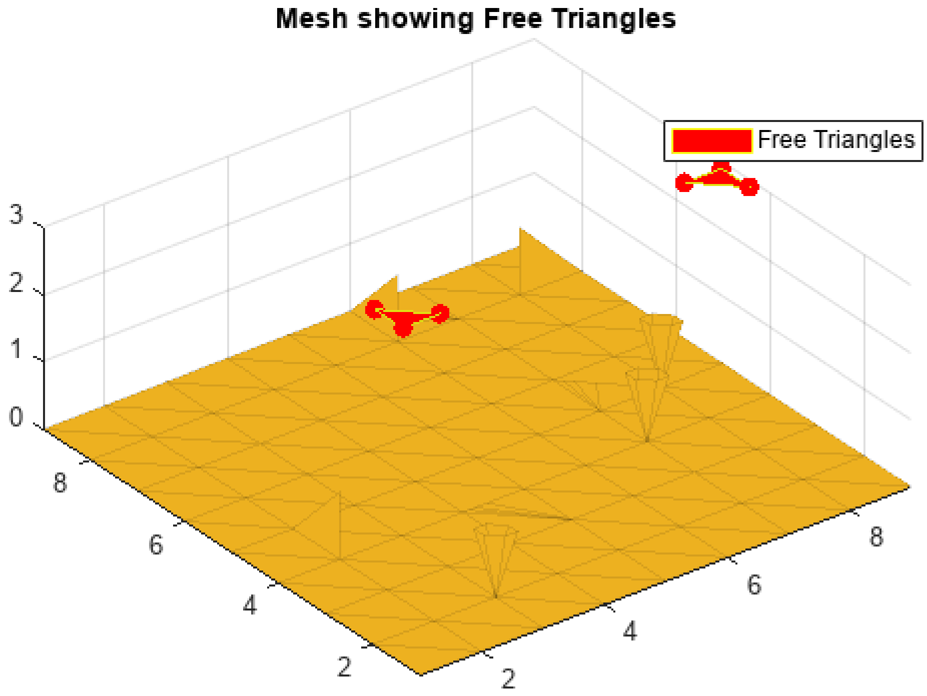
Visualize the non-manifold vertices in the STL file.

```
figure
showNonManifoldVertices(s)
```



Visualize the free triangles in the STL file.

```
figure
showFreeTriangles(s)
```



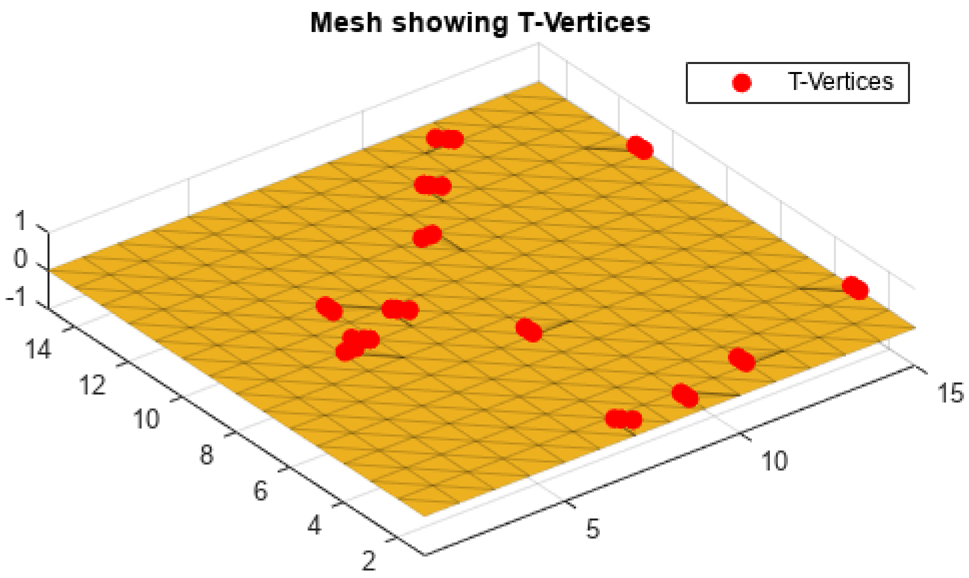
Detect and Visualize T-Vertices

Detect and visualize the T-vertices in an STL file.

```
s = stlFileChecker('sample_file.stl',ShowLog=0);
```

Visualize the T-vertices in the STL file.

```
showTVertices(s)
```

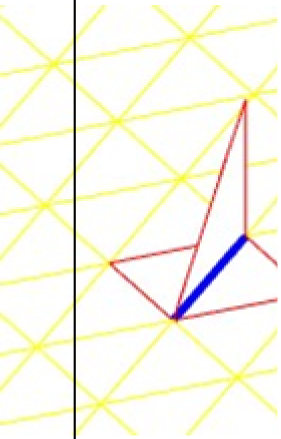
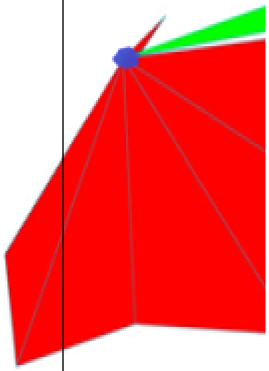
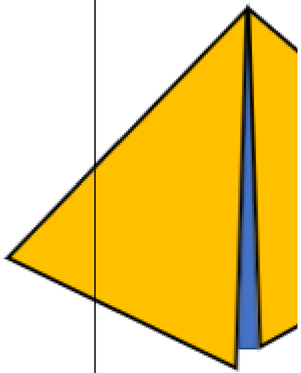
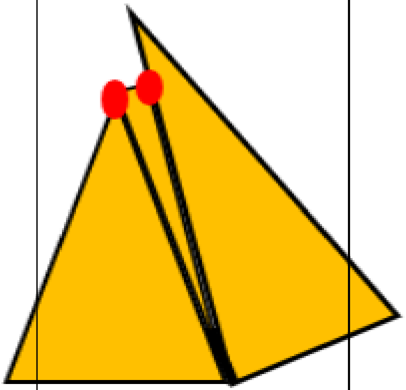
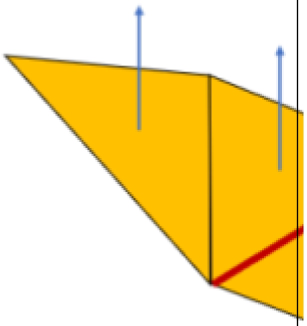
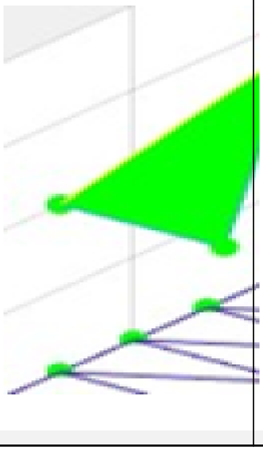
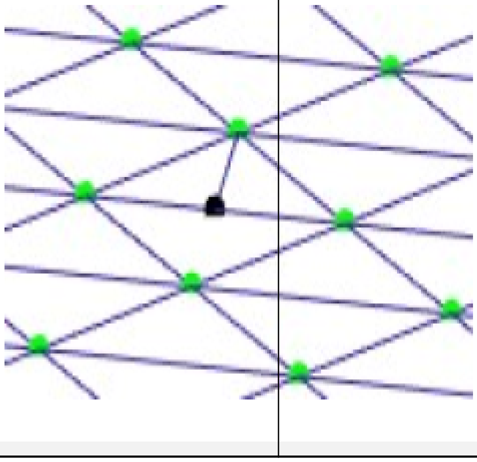


More About

STL File

An STL file describes the surface geometry of a three-dimensional object as a mesh of triangles. The three elements of a triangle are: edge, vertices, and face. This table shows the bad features in an STL file.

STL Bad Features

Non-manifold edge	Non-manifold vertex	Sliver	Duplicate vertices
			
Normal transition edge	Free triangle	T-vertex	
			

Version History

Introduced in R2021b

See Also

`customAntennaStl` | `stlwrite`

CloseIn

Close-in propagation model

Description

Model the behavior of electromagnetic radiation from a point of transmission as it travels through urban macro cell scenarios [1] by using a `CloseIn` object. Close-in propagation models have no enforced frequency range.

Creation

Create a `CloseIn` object by using the `propagationModel` function.

Properties

`ReferenceDistance` — Free-space reference distance
`1` (default) | scalar

Free-space reference distance, specified as a scalar in meters.

Note The close-in model is valid for distances greater than or equal to the `ReferenceDistance`. Path loss is `0` for distances less than `ReferenceDistance`.

Data Types: `double`

`PathLossExponent` — Path loss exponent
`2.9` (default) | scalar

Path loss exponent, specified as a scalar.

Data Types: `double`

`Sigma` — Standard deviation
`5.7` (default) | scalar

Standard deviation of the zero-mean Gaussian random variable, specified as a scalar in decibels (dB).

Data Types: `double`

`NumDataPoints` — Number of data points
`1869` (default) | integer

Number of data points of the zero-mean Gaussian random variable, specified as an integer.

Data Types: `double`

Object Functions

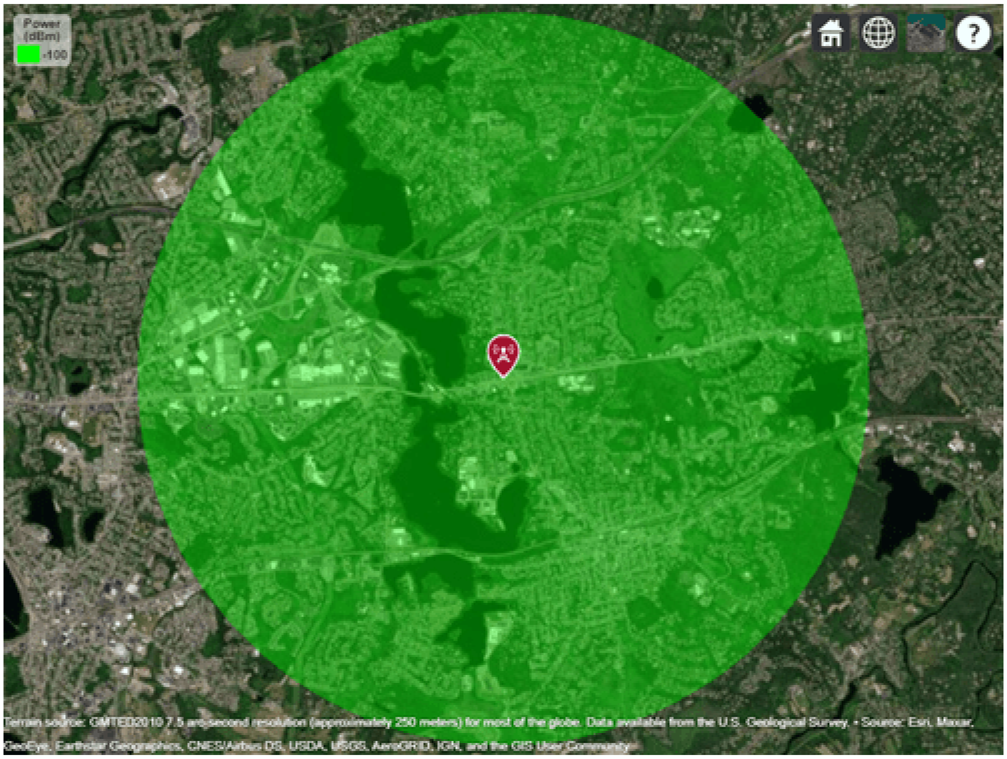
pathloss	Path loss of radio wave propagation
range	Range of radio wave propagation
add	Add propagation models

Examples

Model Coverage Using Close-In Model

Display the coverage area for a transmitter using the close-in propagation model.

```
pm = propagationModel("close-in");
tx = txsite("Name","Apple Hill","Latitude",42.3001,"Longitude",-71.3604);
coverage(tx,pm)
```


Power (dBm)
-100
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2017b

References

[1] Sun, Shu, Theodore S. Rappaport, Timothy A. Thomas, Amitava Ghosh, Huan C. Nguyen, Istvan Z. Kovacs, Ignacio Rodriguez, Ozge Koymen, and Andrzej Partyka. “Investigation of Prediction Accuracy, Sensitivity, and Parameter Stability of Large-Scale Propagation Path Loss Models

for 5G Wireless Communications.” *IEEE Transactions on Vehicular Technology* 65, no. 5 (May 2016): 2843–60. https://doi.org/10.1109/TVT.2016.2543139.

See Also

Functions
`propagationModel` | `coverage`

Objects
`FreeSpace` | `Rain` | `Gas` | `Fog` | `LongleyRice` | `TIREM` | `RayTracing`

Topics
“Choose a Propagation Model”

Fog

Fog propagation model

Description

Model the behavior of electromagnetic radiation from a point of transmission as it travels through fog or clouds [1] by using a `Fog` object. Propagation models for fog are valid from 10 to 1000 GHz, assume line-of-sight (LOS) conditions, and disregard terrain, the curvature of the Earth, and other obstacles.

Creation

Create a `Fog` object by using the `propagationModel` function.

Properties

`Temperature` — Air temperature
`15` (default) | scalar

Air temperature, specified as a scalar in degrees Celsius (C).

Data Types: `double`

`WaterDensity` — Liquid water density
`0.5` (default) | scalar

Liquid water density, specified as a scalar in grams per cubic meter (g/m^3).

Data Types: `double`

Object Functions

pathloss	Path loss of radio wave propagation
range	Range of radio wave propagation
add	Add propagation models

Examples

Model Coverage in Thick Fog

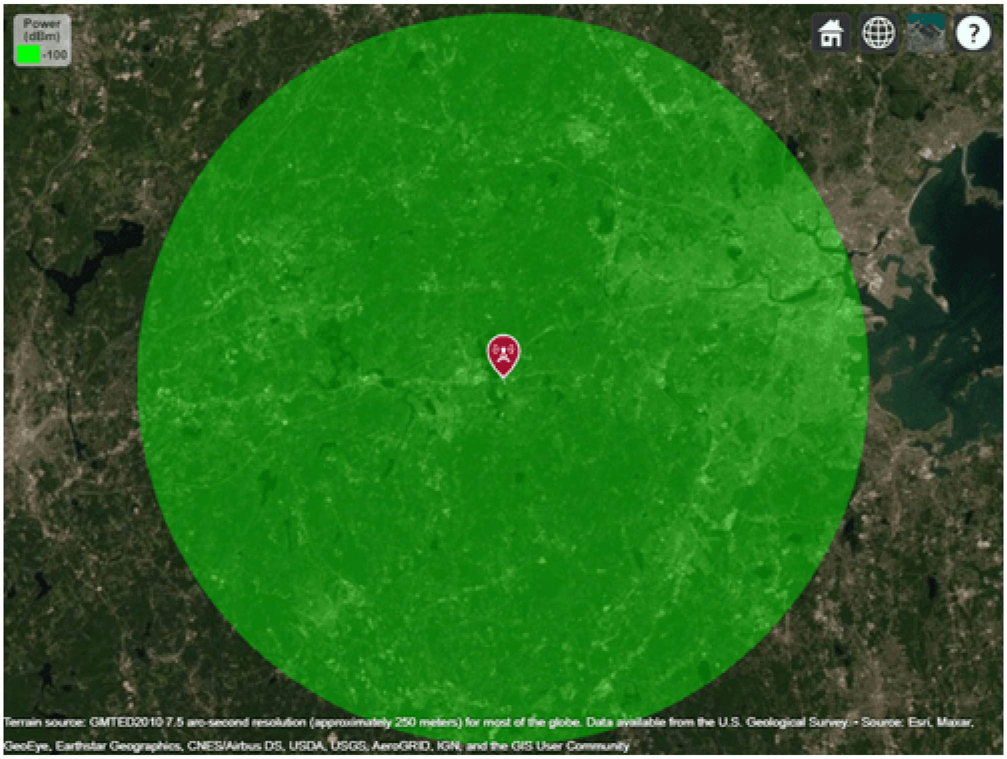
Display the coverage area for a transmitter in thick fog.

Create a propagation model for fog. Specify the water density as 0.5 grams per cubic meter.

```
pm = propagationModel("fog","WaterDensity",0.5);
```

Create a transmitter site. Display the coverage area for the transmitter. Propagation models for fog require transmitter frequencies between 10 and 1000 GHz.

```
tx = txsite("Name","Apple Hill","TransmitterFrequency",1e10, ...
    "Latitude",42.3001,"Longitude",-71.3604);
coverage(tx,pm)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2017b

References

[1] International Telecommunications Union Radiocommunication Sector. *Attenuation due to clouds and fog*. Recommendation P.840-6. ITU-R, approved September 30, 2013. https://www.itu.int/rec/R-REC-P.840-6-201309-S/en.

[2] Seybold, John S. *Introduction to RF Propagation*. Hoboken, N.J: Wiley, 2005.

See Also

Functions

`propagationModel` | `coverage` | `fogpl`

Objects

`FreeSpace` | `Rain` | `Gas` | `CloseIn` | `LongleyRice` | `TIREM` | `RayTracing`

Topics

“Choose a Propagation Model”

FreeSpace

Free space propagation model

Description

Model the behavior of electromagnetic radiation from a point of transmission as it travels through free space by using a `FreeSpace` object. Free space propagation models have no enforced frequency range, assume line-of-sight (LOS) conditions, and disregard terrain, the curvature of the Earth, and other obstacles.

`FreeSpace` objects have no properties.

Creation

Create a `FreeSpace` object by using the `propagationModel` function.

Object Functions

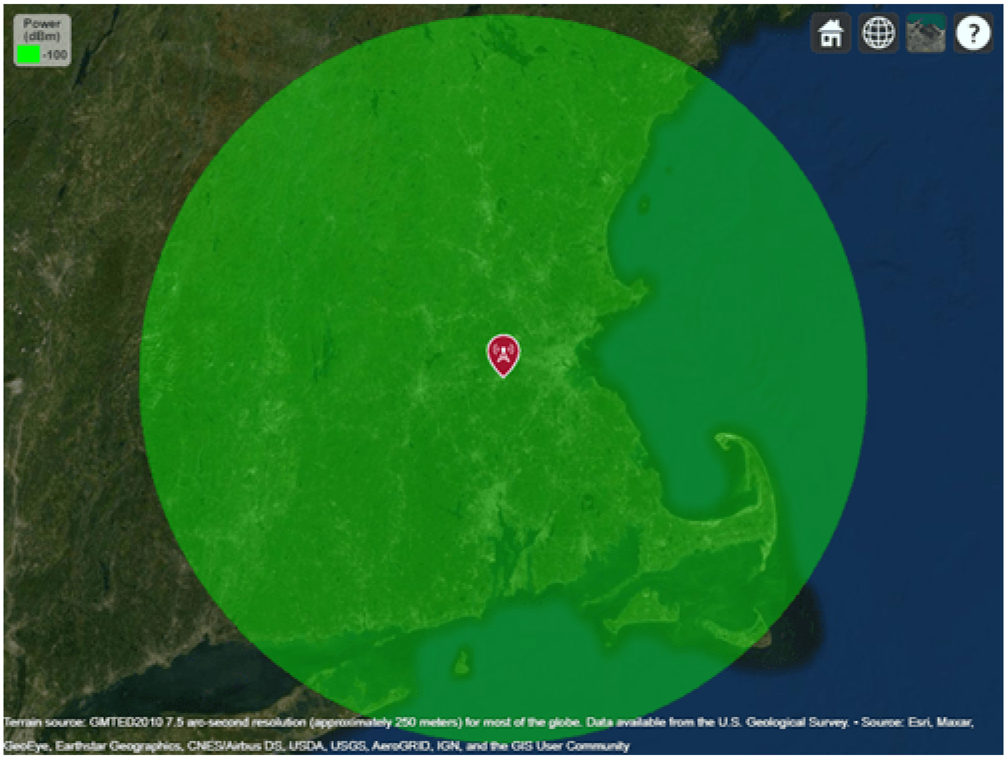
pathloss	Path loss of radio wave propagation
range	Range of radio wave propagation
add	Add propagation models

Examples

Model Coverage in Free Space

Display the coverage area for a transmitter in free space.

```
pm = propagationModel("freespace");
tx = txsite("Name","Apple Hill","Latitude",42.3001,"Longitude",-71.3604);
coverage(tx,pm)
```


Power
(dBm)
🟩 -100
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2017b

References

[1] Seybold, John S. *Introduction to RF Propagation*. Hoboken, N.J: Wiley, 2005.

See Also

Functions

`propagationModel` | `coverage` | `fspl`

Objects

`Rain` | `Gas` | `Fog` | `CloseIn` | `LongleyRice` | `TIREM` | `RayTracing`

Topics

“Choose a Propagation Model”

Gas

Gas propagation model

Description

Model the behavior of electromagnetic radiation from a point of transmission as it travels through gas [1] by using a `Gas` object. Propagation models for gas are valid from 1 to 1000 GHz, assume line-of-sight (LOS) conditions, and disregard terrain, the curvature of the Earth, and other obstacles.

Creation

Create a `Gas` object by using the `propagationModel` function.

Properties

`Temperature` — Air temperature
`15` (default) | scalar

Air temperature, specified as a scalar in degrees Celsius (C).

Data Types: `double`

`AirPressure` — Dry air pressure
`101300` (default) | scalar

Dry air pressure, specified as a scalar in pascals (Pa).

Data Types: `double`

`WaterDensity` — Water vapor density
`7.5` (default) | scalar

Water vapor density, specified as a scalar in grams per cubic meter (g/m^3).

Data Types: `double`

Object Functions

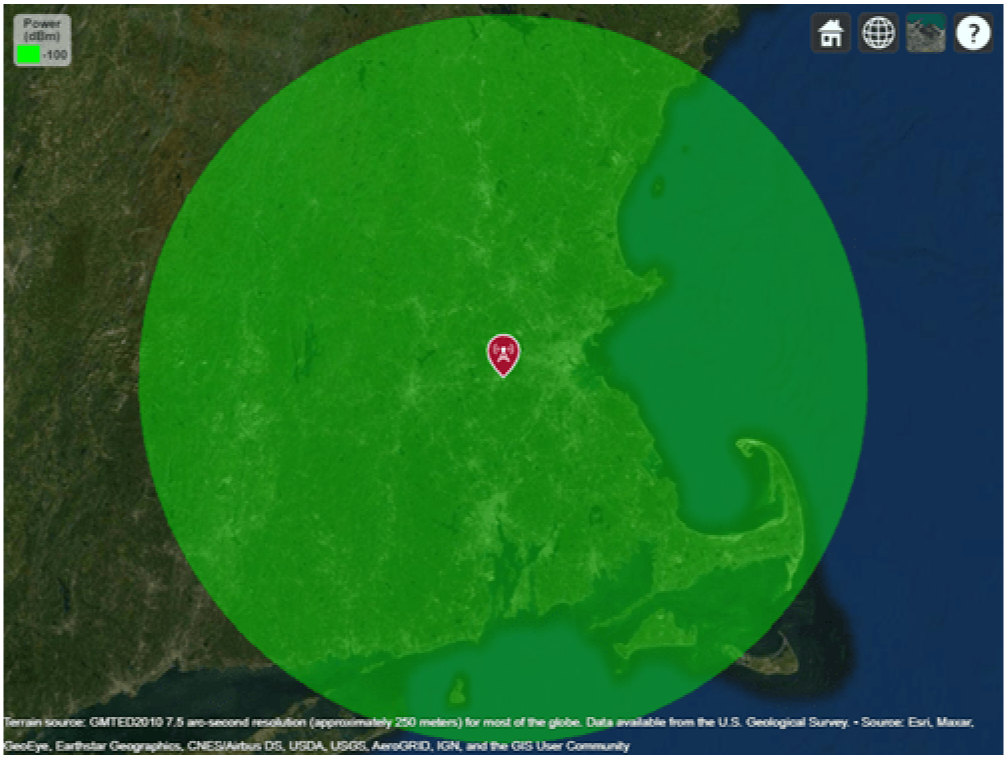
pathloss	Path loss of radio wave propagation
range	Range of radio wave propagation
add	Add propagation models

Examples

Model Coverage in Hot Air

Display the coverage area for a transmitter in hot air. Specify the air temperature as 35 degrees Celsius.

```
pm = propagationModel("gas","Temperature",35);
tx = txsite("Name","Apple Hill","Latitude",42.3001,"Longitude",-71.3604);
coverage(tx,pm)
```


Power
(dBm)
🟩 -100
?
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2017b

References

[1] International Telecommunications Union Radiocommunication Sector. *Attenuation by atmospheric gases*. Recommendation P.676-11. ITU-R, approved September 30, 2016. https://www.itu.int/rec/R-REC-P.676-11-201609-S/en.

[2] Seybold, John S. *Introduction to RF Propagation*. Hoboken, N.J: Wiley, 2005.

See Also

Functions
`propagationModel` | `coverage` | `gaspl`

Objects
`FreeSpace` | `Rain` | `Fog` | `CloseIn` | `LongleyRice` | `TIREM` | `RayTracing`

Topics
“Choose a Propagation Model”

“Urban Link and Coverage Analysis Using Ray Tracing”
“Planning a 5G Fixed Wireless Access Link over Terrain”

LongleyRice

Longley-Rice propagation model

Description

Model the behavior of electromagnetic radiation from a point of transmission over irregular terrain, including buildings, by using the Longley-Rice model, also known at the Irregular Terrain Model (ITM) [1]. Represent the model by using a `LongleyRice` object.

The Longley-Rice model:

- Is valid from 20 MHz to 20 GHz.
- Calculates path loss from free-space loss, terrain and obstacle diffraction, ground reflection, atmospheric refraction, and tropospheric scatter.
- Provides path loss estimates by combining physics with empirical data.

Creation

Create a `LongleyRice` object by using the `propagationModel` function.

Properties

`AntennaPolarization` — Polarization of transmitter and receiver antennas
`"horizontal"` (default) | `"vertical"`

Polarization of transmitter and receiver antennas, specified as `"horizontal"` or `"vertical"`. This object assumes both antennas have the same polarization. The model uses this value to calculate path loss due to ground reflection.

Data Types: `char` | `string`

`GroundConductivity` — Conductivity of ground
`0.005` (default) | scalar

Conductivity of the ground, specified as a scalar in siemens per meter (S/m). The model uses this value to calculate path loss due to ground reflection. The default value corresponds to average ground.

Data Types: `double`

`GroundPermittivity` — Relative permittivity of ground
`15` (default) | scalar

Relative permittivity of the ground, specified as a scalar. Relative permittivity is expressed as a ratio of absolute material permittivity to the permittivity of vacuum. The model uses this value to calculate the path loss due to ground reflection. The default value corresponds to average ground.

Data Types: `double`

`AtmosphericRefractivity` — Atmospheric refractivity near ground

`301` (default) | scalar in N-Units

Atmospheric refractivity near the ground, specified as a scalar in "N-Units" on page 1-880. The model uses this value to calculate the path loss due to refraction through the atmosphere and tropospheric scatter. The default value corresponds to average atmospheric conditions.

Data Types: `double`

`ClimateZone` — Radio climate zone

`"continental-temperate"` (default) | `"equatorial"` | `"continental-subtropical"` | `"maritime-subtropical"` | `"desert"` | `"maritime-over-land"` | `"maritime-over-sea"`

Radio climate zone, specified as one of these options:

- `"continental-temperate"`
- `"equatorial"`
- `"continental-subtropical"`
- `"maritime-subtropical"`
- `"desert"`
- `"maritime-over-land"`
- `"maritime-over-sea"`

The model uses this value to calculate the variability due to changing atmospheric conditions. The default value corresponds to average atmospheric conditions in a particular climate zone.

Data Types: `char` | `string`

`TimeVariabilityTolerance` — Time variability tolerance level

`0.5` (default) | scalar in the range [0.001, 0.999]

Time variability tolerance level of the path loss, specified as a scalar in the range [0.001, 0.999]. Time variability occurs due to changing atmospheric conditions. This value gives the required system reliability expressed as the fraction of time during which the actual path loss is expected to be less than or equal to the model prediction.

Data Types: `double`

`SituationVariabilityTolerance` — Situation variability tolerance level

`0.5` (default) | scalar in the range [0.001, 0.999]

Situation variability tolerance level of the path loss, specified as a scalar in the range [0.001, 0.999]. Situation variability occurs due to uncontrolled or hidden random variables. This value gives the required system confidence expressed as the fraction of similar situations for which the actual path loss is expected to be less than or equal to the model prediction.

Data Types: `double`

Object Functions

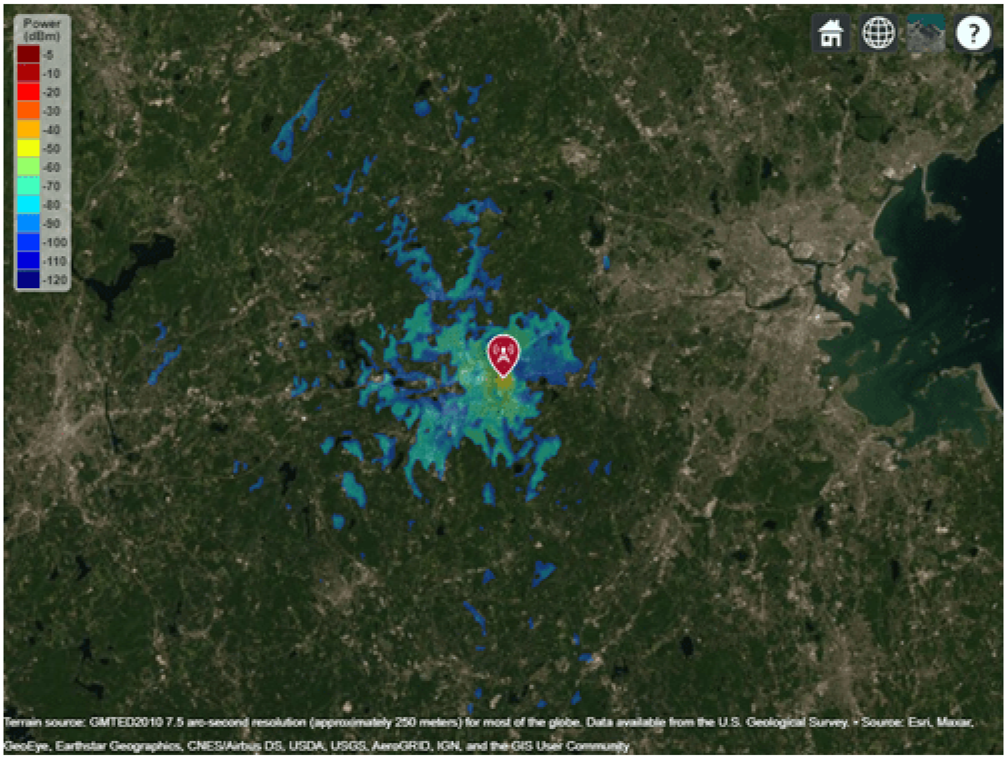
pathloss	Path loss of radio wave propagation
add	Add propagation models

Examples

Model Coverage Using Longley-Rice Model

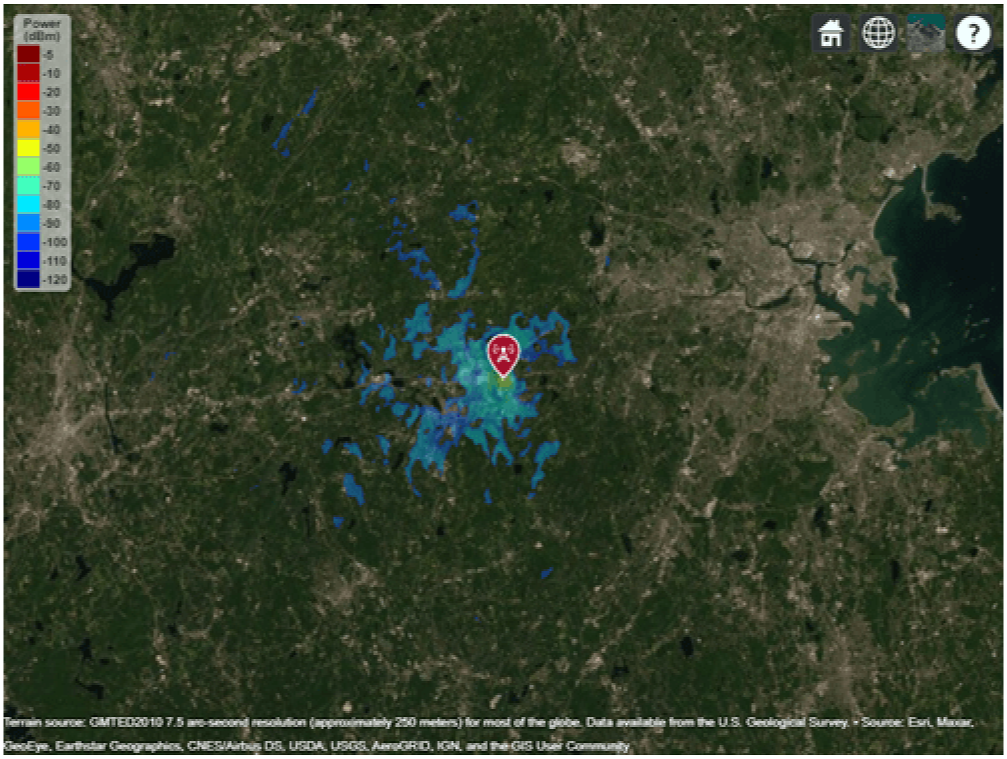
Display the coverage area for a transmitter using the Longley-Rice model.

```
pm = propagationModel("longley-rice");
tx = txsite("Name","Apple Hill","Latitude",42.3001,"Longitude",-71.3604);
coverage(tx,pm,"SignalStrengths",-100:-5)
```


Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Increase the time and situation variability tolerance levels from `0.5` (the default) to `0.9`. Display the coverage area for the transmitter using the updated propagation model.

```
pm.TimeVariabilityTolerance = 0.9;
pm.SituationVariabilityTolerance = 0.9;
coverage(tx,pm,"SignalStrengths",-100:5)
```


Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

The coverage area is smaller for the model with higher variability tolerance levels.

More About

N-Units

The refractive index of air, n, is related to the dielectric constants of the gas constituents of an air mixture. The numerical value of n is only slightly larger than 1. To make the calculation more convenient, you can use N units, which are given by the formula: $N = (n - 1) \times 10^6$.

Version History

Introduced in R2017b

References

[1] Hufford, George A., Anita G. Longley, and William A.Kissick. *A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode*. NTIA Report 82-100. National Telecommunications and Information Administration, April 1, 1982.

[2] Seybold, John S. *Introduction to RF Propagation*. Hoboken, N.J: Wiley, 2005.

See Also

Functions
`propagationModel` | `coverage`

Objects
`FreeSpace` | `Rain` | `Gas` | `Fog` | `CloseIn` | `TIREM` | `RayTracing`

Topics
"Choose a Propagation Model"
"Planning Radar Network Coverage over Terrain"
"Planning a 5G Fixed Wireless Access Link over Terrain"

Rain

Rain propagation model

Description

Model the behavior of electromagnetic radiation from a point of transmission as it travels through rain [1] by using a Rain object. Propagation models for rain are valid from 1 to 1000 GHz, assume line-of-sight (LOS) conditions, and disregard terrain, the curvature of the Earth, and other obstacles.

Creation

Create a Rain object by using the propagationModel function.

Properties

RainRate — Rain rate
16 (default) | nonnegative scalar

Rain rate, specified as a nonnegative scalar in millimeters per hour (mm/h).

Data Types: double

Tilt — Polarization tilt angle of signal
0 (default) | scalar

Polarization tilt angle of the signal, specified as a scalar in degrees.

Data Types: double

Object Functions

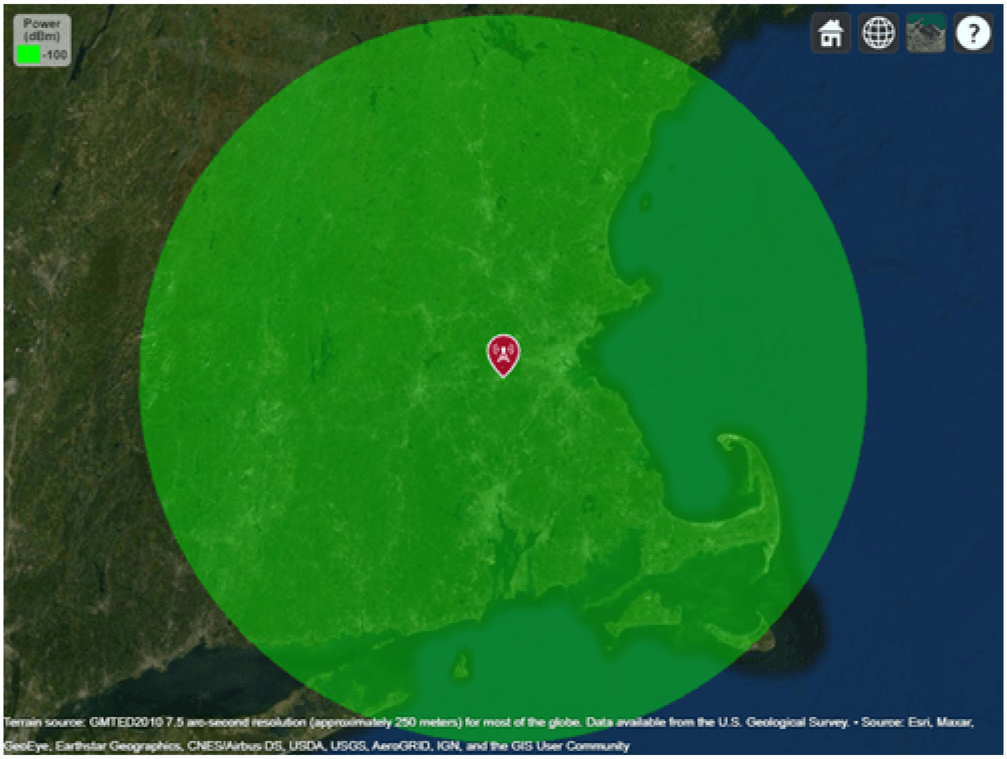
pathloss	Path loss of radio wave propagation
range	Range of radio wave propagation
add	Add propagation models

Examples

Model Coverage in Heavy Rain

Display the coverage area for a transmitter in heavy rain. Specify the rain rate as 50 millimeters per hour.

```
pm = propagationModel("rain","RainRate",50);
tx = txsite("Name","Apple Hill","Latitude",42.3001,"Longitude",-71.3604);
coverage(tx,pm)
```


Power (dBm)
🟩 -100
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2017b

References

[1] International Telecommunications Union Radiocommunication Sector. *Specific attenuation model for rain for use in prediction methods*. Recommendation P.838-3. ITU-R, approved March 8, 2005. https://www.itu.int/rec/R-REC-P.838-3-200503-I/en.

[2] Seybold, John S. *Introduction to RF Propagation*. Hoboken, N.J: Wiley, 2005.

See Also

Functions

`propagationModel` | `coverage` | `rainpl`

Objects

`FreeSpace` | `Gas` | `Fog` | `CloseIn` | `LongleyRice` | `TIREM` | `RayTracing`

Topics

"Choose a Propagation Model"
"Visualize Antenna Coverage Map and Communication Links"
"Urban Link and Coverage Analysis Using Ray Tracing"
"Planning a 5G Fixed Wireless Access Link over Terrain"

RayTracing

Ray tracing propagation model

Description

Ray tracing models compute propagation paths using 3-D environment geometry [1][2] . Represent a ray tracing model by using a `RayTracing` object.

Ray tracing models:

- Are valid from 100 MHz to 100 GHz.
- Compute multiple propagation paths. Other propagation models compute only single propagation paths.
- Support 3-D outdoor and indoor environments.
- Determine the path loss and phase shift of each ray using electromagnetic analysis, including tracing the horizontal and vertical polarizations of a signal through the propagation path. The path loss includes free-space loss and reflection losses. For each reflection, the model calculates losses on the horizontal and vertical polarizations by using the Fresnel equation, the incident angle, and the relative permittivity and conductivity of the surface material [3][4] at the specified frequency.

You can create ray tracing models that use either the shooting and bouncing rays (SBR) method on page 1-891 or the image method on page 1-892.

Creation

Create a `RayTracing` object by using the `propagationModel` function.

Properties

Ray Tracing

`Method` — Ray tracing method
`"sbr"` (default) | `"image"`

Ray tracing method, specified as one of these values:

- `"sbr"` — Use the shooting and bouncing rays (SBR) method on page 1-891, which supports up to 10 path reflections. The SBR method calculates an approximate number of propagation paths with exact geometric accuracy. The SBR method is generally faster than the image method. The model calculates path loss from free-space loss plus reflection losses due to material and antenna polarizations.
- `"image"` — Use the image method on page 1-892, which supports up to 2 path reflections. The image method calculates an exact number of propagation paths with exact geometric accuracy. The model calculates path loss from free-space loss plus reflection losses due to material and antenna polarizations.

Specify the maximum number of path reflections by using the `MaxNumReflections` property.

When both the image and SBR methods find the same path, the points along the path are the same within a tolerance of machine precision for single-precision floating-point values. For more information about differences between the image and SBR methods, see “Choose a Propagation Model”.

Data Types: `char` | `string`

`AngularSeparation` — Average number of degrees between launched rays

`"medium"` (default) | `"high"` | `"low"` | numeric scalar in degrees in the range [0.05, 10]

Average number of degrees between launched rays, specified as `"high"`, `"medium"`, `"low"`, or a numeric scalar in degrees in the range [0.05, 10]. If you specify a numeric value, then the ray tracing algorithm might use a lower value than the value you specify.

This table describes the behavior of the `"high"`, `"medium"`, and `"low"` options.

Option	Approximate Numeric Equivalent	Range of Numeric Values	Number of Launched Rays
`"high"`	1.0781	[0.9912, 1.1845]	40,962
`"medium"`	0.5391	[0.4956, 0.5923]	163,842
`"low"`	0.2695	[0.2478, 0.2961]	655,362

To improve the accuracy of the number of paths found by the SBR method, decrease the value of `AngularSeparation`. Decreasing the value of `AngularSeparation` can increase the amount of time MATLAB requires to perform the analysis.

When you first use a given value of `AngularSeparation` in a MATLAB session, MATLAB caches the geodesic sphere associated with that value for the duration of the session. As a result, the first use of that value of `AngularSeparation` takes longer than subsequent uses within the same session. For more information about geodesic spheres, see “Shooting and Bouncing Rays Method” on page 1-891.

Tips

When creating coverage maps using the `coverage` function, you can improve the results by choosing a lower angular separation.

Dependencies

To enable this argument, you must specify the `Method` property as `"sbr"`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`MaxNumReflections` — Maximum number of path reflections

2 (default) | integer in the range [0,10]

Maximum number of path reflections to search for using ray tracing, specified as an integer. Supported values depend on the value of the `Method` property.

- When `Method` is `"image"`, supported values are `0`, `1`, and `2`.
- When `Method` is `"sbr"`, supported values are in the range [0, 10].

Data Types: `double`

`CoordinateSystem` — Coordinate system of map and site location

`"geographic"` (default) | `"cartesian"`

Coordinate system of the site location, specified as `"geographic"` or `"cartesian"`. If you specify `"geographic"`, define material types by using the `BuildingsMaterial` and `TerrainMaterial` properties. If you specify `"cartesian"`, define material types by using the `SurfaceMaterial` property.

Data Types: `string` | `char`

Buildings Material

`BuildingsMaterial` — Surface material of geographic buildings

`"concrete"` (default) | `"perfect-reflector"` | `"brick"` | `"wood"` | `"glass"` | `"metal"` | `"custom"`

Surface material of geographic buildings, specified as one of these values: `"perfect-reflector"`, `"concrete"`, `"brick"`, `"wood"`, `"glass"`, `"metal"`, or `"custom"`. The model uses the material type to calculate reflection loss where propagation paths reflect off of building surfaces. For more information, see “ITU Permittivity and Conductivity Values for Common Materials” on page 1-893.

When `BuildingsMaterial` is `"custom"`, specify the material permittivity and conductivity by using the `BuildingsMaterialPermittivity` and `BuildingsMaterialConductivity` properties.

Dependencies

To enable `BuildingsMaterial`, you must set `CoordinateSystem` to `"geographic"`.

Data Types: `char` | `string`

`BuildingsMaterialPermittivity` — Relative permittivity of surface materials of buildings

`5.31` (default) | nonnegative scalar

Relative permittivity of the surface materials of the buildings, specified as a nonnegative scalar. Relative permittivity is expressed as a ratio of absolute material permittivity to the permittivity of vacuum. The model uses this value to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To enable `BuildingsMaterialPermittivity`, you must set `CoordinateSystem` to `"geographic"` and `BuildingsMaterial` to `"custom"`.

Data Types: `double`

`BuildingsMaterialConductivity` — Conductivity of surface materials of buildings

`0.0548` (default) | nonnegative scalar

Conductivity of the surface materials of the buildings, specified as a nonnegative scalar in siemens per meter (S/m). The model uses this value to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To enable `BuildingsMaterialConductivity`, you must set `CoordinateSystem` to `"geographic"` and `BuildingsMaterial` to `"custom"`.

Data Types: double

Terrain Material

TerrainMaterial — Surface material of geographic terrain
"concrete" (default) | "perfect-reflector" | "brick" | "water" | "vegetation" | "loam" | "custom"

Surface material of the geographic terrain, specified as one of these values: "perfect-reflector", "concrete", "brick", "water", "vegetation", "loam", or "custom". The model uses the material type to calculate reflection loss where propagation paths reflect off of terrain surfaces. For more information, see "ITU Permittivity and Conductivity Values for Common Materials" on page 1-893.

When TerrainMaterial is "custom", specify the material permittivity and conductivity by using the TerrainMaterialPermittivity and TerrainMaterialConductivity properties.

Dependencies

To enable TerrainMaterial, you must set CoordinateSystem to "geographic".

Data Types: char | string

TerrainMaterialPermittivity — Relative permittivity of terrain materials
5.31 (default) | nonnegative scalar

Relative permittivity of the terrain material, specified as a nonnegative scalar. Relative permittivity is expressed as a ratio of absolute material permittivity to the permittivity of vacuum. The model uses this value to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To enable TerrainMaterialPermittivity, you must set CoordinateSystem to "geographic" and TerrainMaterial to "custom".

Data Types: double

TerrainMaterialConductivity — Conductivity of terrain materials
0.0548 (default) | nonnegative scalar

Conductivity of the terrain material, specified as a nonnegative scalar in siemens per meter (S/m). The model uses this value to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To enable TerrainMaterialConductivity, you must set CoordinateSystem to "geographic" and set TerrainMaterial to "custom".

Data Types: double

Surface Material

SurfaceMaterial — Surface material of Cartesian map surface
"plasterboard" (default) | "perfect-reflector" | "ceilingboard" | "chipboard" | "floorboard" | "concrete" | "brick" | "wood" | "glass" | "metal" | "water" | "vegetation" | "loam" | "custom"

Surface material of Cartesian map surface, specified as one of these values: "plasterboard","perfect-reflector", "ceilingboard", "chipboard", "floorboard", "concrete", "brick", "wood", "glass", "metal", "water", "vegetation", "loam", or "custom". The model uses the material type to calculate reflection loss where propagation paths reflect off of surfaces. For more information, see “ITU Permittivity and Conductivity Values for Common Materials” on page 1-893.

When SurfaceMaterial is "custom", specify the material permittivity and conductivity by using the SurfaceMaterialPermittivity and SurfaceMaterialConductivity properties.

Dependencies

To enable SurfaceMaterial, you must set CoordinateSystem to "cartesian".

Data Types: char | string

SurfaceMaterialPermittivity — Relative permittivity of surface materials

2.94 (default) | nonnegative scalar

Relative permittivity of the surface material, specified as a nonnegative scalar. Relative permittivity is expressed as a ratio of absolute material permittivity to the permittivity of vacuum. The model uses this value to calculate path loss due to reflection. The default value corresponds to plaster board at 1.9 GHz.

Dependencies

To enable SurfaceMaterialPermittivity, you must set CoordinateSystem to "cartesian" and SurfaceMaterial to "custom".

Data Types: double

SurfaceMaterialConductivity — Conductivity of surface materials

0.0183 (default) | nonnegative scalar

Conductivity of the surface material, specified as a nonnegative scalar in siemens per meter (S/m). The model uses this value to calculate path loss due to reflection. The default value corresponds to plaster board at 1.9 GHz.

Dependencies

To enable SurfaceMaterialConductivity, you must set CoordinateSystem to "cartesian" and set SurfaceMaterial to "custom".

Data Types: double

Object Functions

pathloss	Path loss of radio wave propagation
add	Add propagation models

Examples

Model Propagation Paths Using SBR and Image Methods

Show reflected propagation paths in Chicago by using the SBR and image methods.

Create a Site Viewer with buildings in Chicago. For more information about the osm file, see [1] on page 1-890.

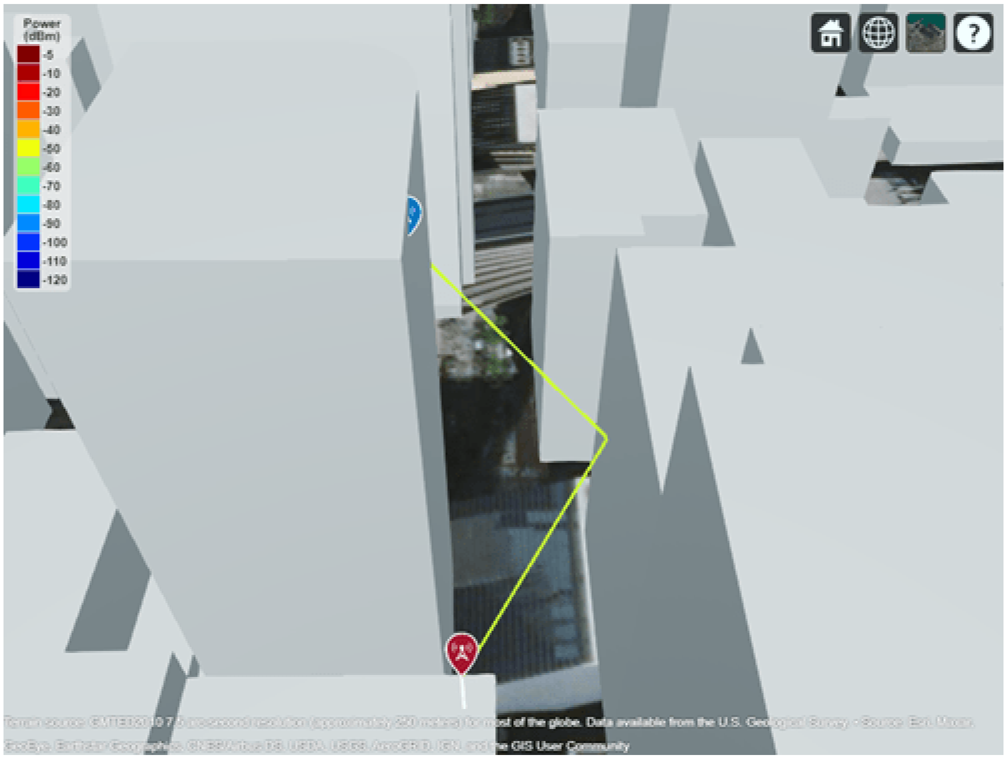
```
viewer = siteviewer("Buildings","chicago.osm");
```

Create a transmitter site on a building and a receiver site near another building.

```
tx = txsite("Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
show(tx)
rx = rxsite("Latitude",41.8813452, ...
    "Longitude",-87.629771, ...
    "AntennaHeight",30);
show(rx)
```

Create a ray tracing model. Use the image method and calculate paths with up to one reflection. Then, display the propagation paths.

```
pm = propagationModel("raytracing","Method","image", ...
    "MaxNumReflections",1);
raytrace(tx,rx,pm)
```

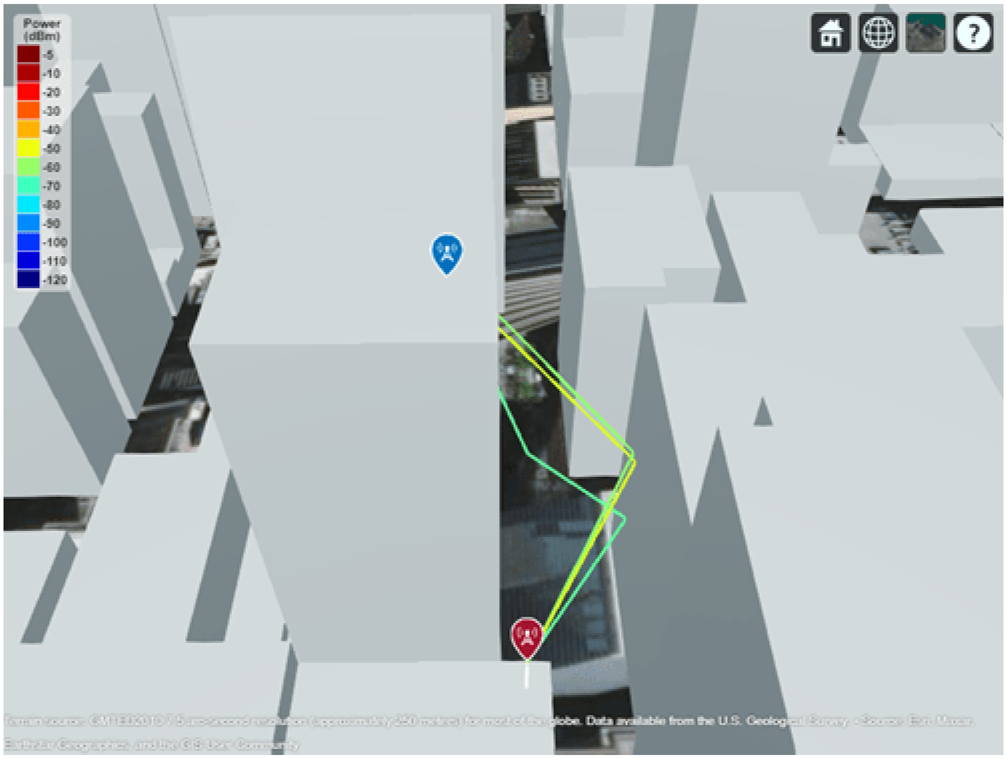

Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

For this ray tracing model, there is one propagation path from the transmitter to the receiver.

Update the ray tracing model to use the SBR method and to calculate paths with up to two reflections. Display the propagation paths.

```
pm.Method = "sbr";
pm.MaxNumReflections = 2;
```

```
clearMap(viewer)
raytrace(tx,rx,pm)
```


Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

The updated ray tracing model shows three propagation paths from the transmitter to the receiver.

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Model Coverage Using Ray Tracing

Create a Site Viewer with buildings in Chicago. For more information about the `.osm` file, see [1] on page 1-891.

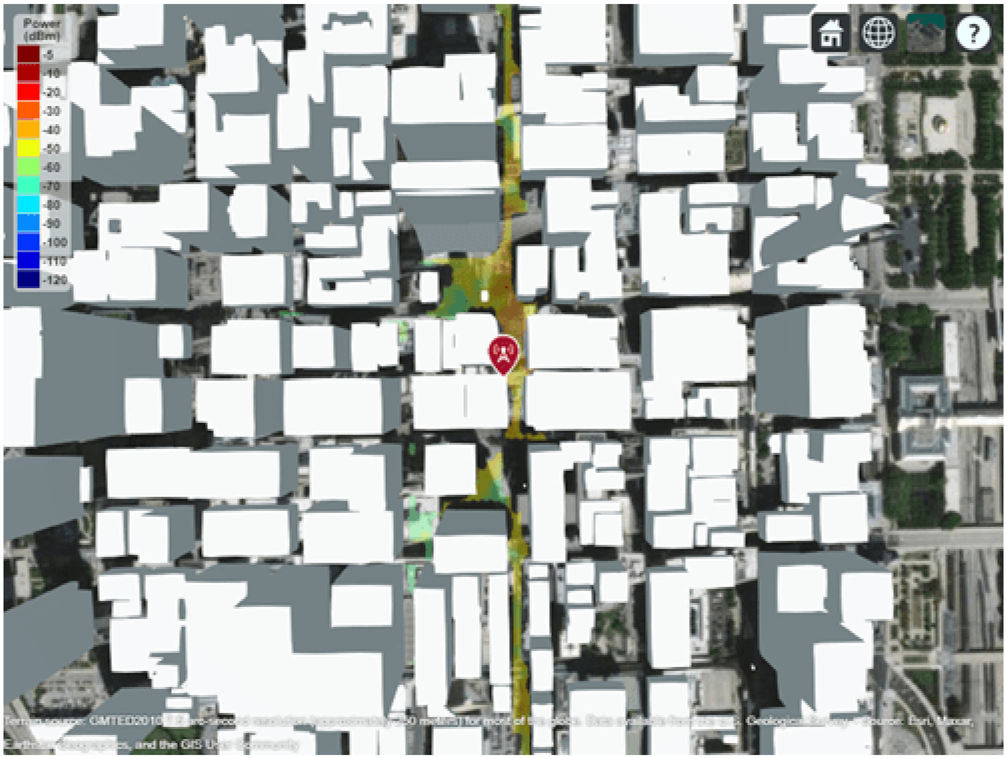
```
viewer = siteviewer("Buildings","chicago.osm");
```

Create a transmitter site on a building and a receiver site near another building.

```
tx = txsite("Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
show(tx)
```

Create a ray tracing model. By default, ray tracing models use the SBR method. Set the maximum number of reflections to 2. Then, display the coverage map.

```
pm = propagationModel("raytracing","Method","sbr", ...
    "MaxNumReflections",2);
coverage(tx,pm,"SignalStrengths",-100:5)
```


Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Basemap source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The `.osm` file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

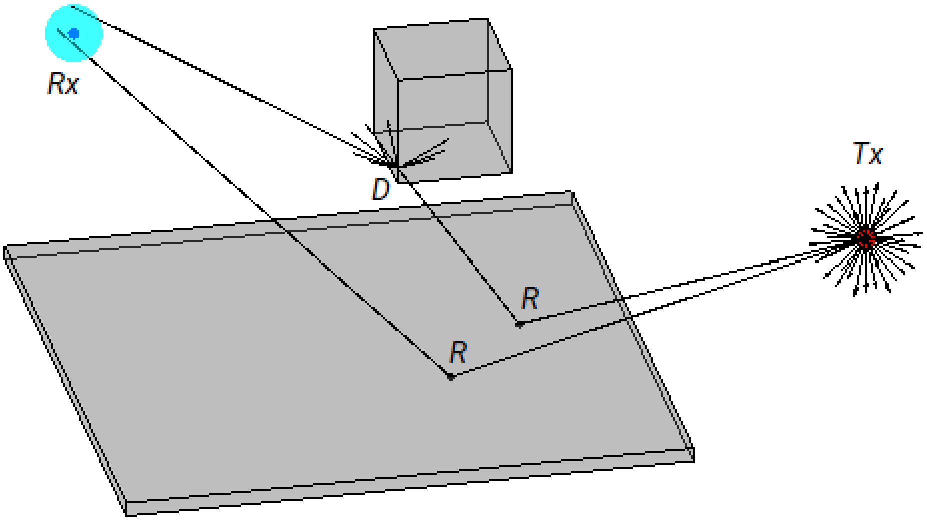
More About

Shooting and Bouncing Rays Method

The shooting and bouncing rays (SBR) method finds an approximate number of propagation paths with exact geometric accuracy. You can use this method to find paths with up to 10 path reflections.

The computational complexity of the SBR method increases linearly with the number of reflections. As a result, the SBR method is generally faster than the image method.

This figure illustrates the SBR method for calculating propagation paths from a transmitter, *Tx*, to a receiver, *Rx*.



The SBR method launches many rays from a geodesic sphere centered at *Tx*. The geodesic sphere enables the model to launch rays that are approximately uniformly spaced.

Then, the method traces every ray from *Tx* and can model different types of interactions between the rays and surrounding objects, such as reflections, diffractions, refractions, and scattering. Note that the current implementation of the SBR method considers only reflections.

- When a ray hits a flat surface, shown as *R*, the ray reflects based on the law of reflection.
- When a ray hits an edge, shown as *D*, the ray spawns many diffracted rays based on the law of diffraction [5][6]. Each diffracted ray has the same angle with the diffracting edge as the incident ray. The diffraction point then becomes a new launching point and the SBR method traces the diffracted rays in the same way as the rays launched from *Tx*. A continuum of diffracted rays forms a cone around the diffracting edge, which is commonly known as a Keller cone [6]. The current implementation of the SBR method does not consider edge diffractions.

For each launched ray, the SBR method surrounds *Rx* with a sphere, called a reception sphere, with a radius that is proportional to the distance the ray travels and the average number of degrees between the launched rays. If the ray intersects the sphere, then the model considers the ray a valid path from *Tx* to *Rx*. The SBR method corrects the valid paths so that the paths have exact geometric accuracy.

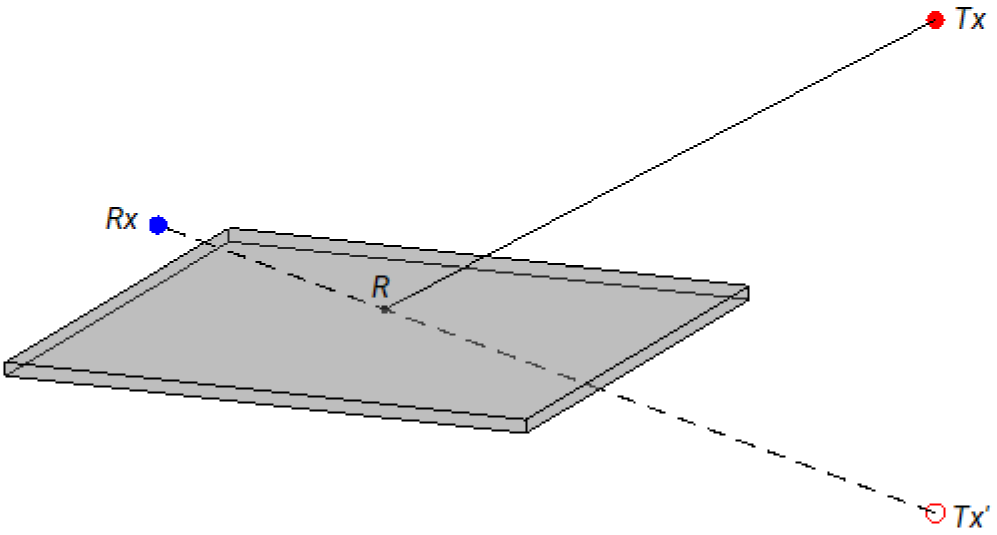
When you increase the number of rays by decreasing the number of degrees between rays, the reception sphere becomes smaller. As a result, in some cases, launching more rays results in fewer or different paths. This situation is more likely to occur with custom 3-D scenarios created from STL files or triangulation objects than with scenarios that are automatically generated from OpenStreetMap® buildings and terrain data.

The SBR method finds paths using single-precision floating-point computations.

Image Method

The image method finds an exact number of propagation paths with exact geometric accuracy. You can use this method to find paths with up to 2 path reflections. The computational complexity of the image method increases exponentially with the number of reflections.

This figure illustrates the image method for calculating the propagation path of a single reflection ray for the same transmitter and receiver as the SBR method. The image method locates the image of *Tx* with respect to a planar reflection surface, *Tx*'. Then, the method connects *Tx*' and *Rx* with a line segment. If the line segment intersects the planar reflection surface, shown as *R* in the figure, then a valid path from *Tx* to *Rx* exists. The method determines paths with multiple reflections by recursively extending these steps. The image method finds paths using single-precision floating-point computations.



ITU Permittivity and Conductivity Values for Common Materials

ITU-R P.2040-1 [3] and ITU-R P.527-5 [4] present methods, equations, and values used to calculate real relative permittivity, conductivity, and complex relative permittivity for common materials.

- For information about the values computed for building materials specified in ITU-R P.2040-1, see `buildingMaterialPermittivity`.
- For information about the values computed for terrain materials specified in ITU-R P.527-5, see `earthSurfacePermittivity`.

Version History

Introduced in R2017b

Customize spacing of launched rays for ray tracing with SBR method

When performing ray tracing using the SBR method, you can customize the spacing of launched rays by specifying the `AngularSeparation` property of the `RayTracing` object as a numeric value in degrees. In previous releases, the `AngularSeparation` property supported only the options `"high"`, `"medium"`, and `"low"`.

SBR method calculates propagation paths with exact geometric accuracy

Behavior changed in R2022b

When you find propagation paths using the SBR method, MATLAB corrects the results so that the geometric accuracy of each path is exact. In previous releases, the paths have approximate geometric accuracy.

Default modeling method is shooting and bouncing rays method
Behavior changed in R2021b

Starting in R2021b, when you create a propagation model using the syntax `propagationModel("raytracing")`, MATLAB returns a `RayTracing` model with the `Method` value set to `"sbr"` and two reflections (instead of `"image"` and one reflection, as in previous releases).

To create ray tracing propagation models that use the image method, use the syntax `propagationModel("raytracing","Method","image")`.

References

[1] Yun, Zhengqing, and Magdy F. Iskander. “Ray Tracing for Radio Propagation Modeling: Principles and Applications.” *IEEE Access* 3 (2015): 1089–1100. https://doi.org/10.1109/ACCESS.2015.2453991.

[2] Schaubach, K.R., N.J. Davis, and T.S. Rappaport. “A Ray Tracing Method for Predicting Path Loss and Delay Spread in Microcellular Environments.” In *[1992 Proceedings] Vehicular Technology Society 42nd VTS Conference - Frontiers of Technology*, 932–35. Denver, CO, USA: IEEE, 1992. https://doi.org/10.1109/VETEC.1992.245274.

[3] International Telecommunications Union Radiocommunication Sector. *Effects of building materials and structures on radiowave propagation above about 100MHz.* Recommendation P.2040-1. ITU-R, approved July 29, 2015. https://www.itu.int/rec/R-REC-P.2040-1-201507-I/en.

[4] International Telecommunications Union Radiocommunication Sector. *Electrical characteristics of the surface of the Earth*. Recommendation P.527-5. ITU-R, approved August 14, 2019. https://www.itu.int/rec/R-REC-P.527-5-201908-I/en.

[5] International Telecommunications Union Radiocommunication Sector. *Propagation by diffraction*. Recommendation P.526-15. ITU-R, approved October 21, 2019. https://www.itu.int/rec/R-REC-P.526-15-201910-I/en.

[6] Keller, Joseph B. “Geometrical Theory of Diffraction.” *Journal of the Optical Society of America* 52, no. 2 (February 1, 1962): 116. https://doi.org/10.1364/JOSA.52.000116.

See Also

Functions
`propagationModel` | `raytrace` | `coverage` | `sigstrength` | `buildingMaterialPermittivity` | `earthSurfacePermittivity`

Objects
`FreeSpace` | `Rain` | `Gas` | `Fog` | `CloseIn` | `LongleyRice` | `TIREM`

Topics
“Choose a Propagation Model”
“Ray Tracing for Wireless Communications”

“Urban Link and Coverage Analysis Using Ray Tracing”

TIREM

TIREM propagation model

Description

Model the behavior of electromagnetic radiation from a point of transmission as it travels over irregular terrain, including buildings, by using the Terrain Integrated Rough Earth Model™ (TIREM™) model. Represent the TIREM model by using a `TIREM` object.

The TIREM model:

- Is valid from 1 MHz to 1000 GHz.
- Calculates path loss from free-space loss, terrain and obstacle diffraction, ground reflection, atmospheric refraction, and tropospheric scatter.
- Provides path loss estimates by combining physics with empirical data.

`TIREM` objects require access to an external TIREM library. For more information, see “Access TIREM Software”.

Creation

Create a `TIREM` object by using the `propagationModel` function.

Properties

`AntennaPolarization` — Polarization of transmitter and receiver antennas
`"horizontal"` (default) | `"vertical"`

Polarization of transmitter and receiver antennas, specified as `"horizontal"` or `"vertical"`. The object assumes both antennas have the same polarization. The model uses this value to calculate path loss due to ground reflection.

Data Types: `char` | `string`

`GroundConductivity` — Conductivity of ground
`0.005` (default) | numeric scalar in the range [0.0005, 100]

Conductivity of the ground, specified as a numeric scalar in siemens per meter (S/m) in the range [0.0005, 100]. The model uses this value to calculate path loss due to ground reflection. The default value corresponds to average ground.

Data Types: `double`

`GroundPermittivity` — Relative permittivity of ground
`15` (default) | numeric scalar in the range [1, 100]

Relative permittivity of the ground, specified as a numeric scalar in the range [1, 100]. Relative permittivity is expressed as a ratio of absolute material permittivity to the permittivity of vacuum. The model uses this value to calculate the path loss due to ground reflection. The default value corresponds to average ground.

Data Types: `double`

`AtmosphericRefractivity` — Atmospheric refractivity near ground

`301` (default) | numeric scalar in the range [250, 400]

Atmospheric refractivity near the ground, specified as a numeric scalar in N-units on page 1-898 in the range [250, 400]. The model uses this value to calculate the path loss due to refraction through the atmosphere and tropospheric scatter. The default value corresponds to average atmospheric conditions.

Data Types: `double`

`Humidity` — Absolute air humidity near ground

`9` (default) | numeric scalar in the range [0, 110]

Absolute air humidity near the ground, specified as a numeric scalar in grams per cubic meter (`g/m`^{3}) in the range [0, 110]. You can use this value to calculate path loss due to atmospheric absorption. The default value corresponds to the absolute humidity of air at 15 degrees Celsius and 70 percent relative humidity.

Data Types: `double`

Object Functions

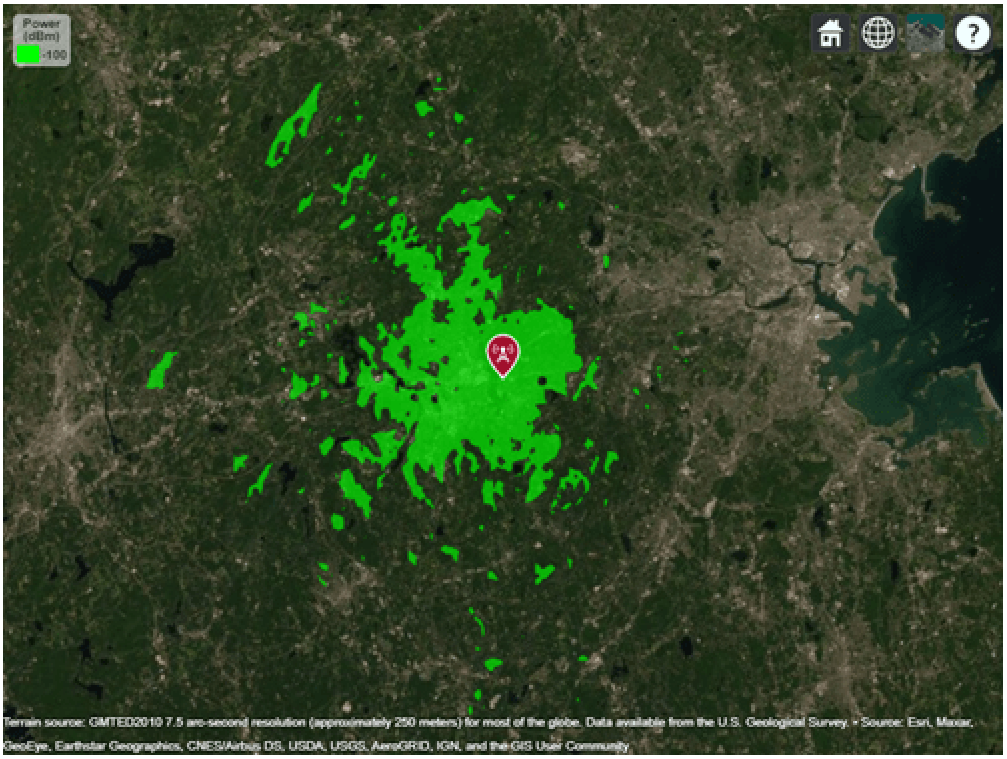
pathloss	Path loss of radio wave propagation
add	Add propagation models

Examples

Model Coverage Using TIREM™

Display the coverage area for a transmitter using the TIREM model.

```
pm = propagationModel("tirem");
tx = txsite("Name","Apple Hill","Latitude",42.3001,"Longitude",-71.3604);
coverage(tx,pm)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

More About

N-Units

The refractive index of air, n, is related to the dielectric constants of the gas constituents of an air mixture. The numerical value of n is only slightly larger than 1. To make the calculation more convenient, you can use N units, which are given by the formula: $N = (n - 1) \times 10^6$.

Version History

Introduced in R2017b

References

[1] Seybold, John S. *Introduction to RF Propagation*. Hoboken, N.J: Wiley, 2005.

See Also

Functions
`propagationModel` | `coverage`

Objects
`FreeSpace` | `Rain` | `Gas` | `Fog` | `CloseIn` | `LongleyRice` | `RayTracing`

Topics
“Choose a Propagation Model”

“Planning Radar Network Coverage over Terrain”

solver

Access FMM solver for electromagnetic analysis

Description

Use the `solver` object to access the fast multipole method (FMM) solver settings in large antenna structures, such as egg crate, installed antenna, and curved reflectors, for electromagnetic (EM) analysis.

Creation

Syntax

`s = solver(obj)`

Description

`s = solver(obj)` returns the solver used in the antenna defined in `obj` for EM analysis. For more information on EM analysis, see "Antenna and Array Analysis". After you create a `solver` object, you can modify the object properties using dot notation.

Note You must set the `SolverType` property to `'FMM'` in the egg crate, installed antenna, or curved reflectors antenna objects to access the `solver` object.

Input Arguments

`obj` — Large antenna
`eggCrate` object | `installedAntenna` object | curved reflector objects

Large antenna, specified as one of the following:

Antenna Name	Antenna Objects
Egg crate	`eggCrate`
Installed antenna	`installedAntenna`
Curved reflectors	• `cassegrain` • `cassegrainOffset` • `gregorian` • `gregorianOffset` • `reflectorParabolic` • `reflectorSpherical`

Example: `s = solver(obj)`

Properties

`Iterations` — Maximum number of iterations

`100` (default) | positive scalar

Maximum number of iterations needed by the FMM solver to achieve convergence, specified as a positive scalar. During EM analysis, if convergence is achieved with fewer iterations than the maximum number of iterations specified using this property, the FMM solver terminates the EM analysis and displays the EM analysis solutions.

Example: `obj.Iterations = 150;`

`RelativeResidual` — Residual error allowed in EM solution

`1.0000e-4` (default) | positive scalar

Residual error allowed in EM solution, specified as a positive scalar.

Example: `obj.RelativeResidual = 1.5e-3;`

`Precision` — FMM solver precision

`2.0000e-4` (default) | positive scalar

FMM solver precision, specified as a positive scalar.

Example: `obj.Precision = 2.0000e-5;`

Object Functions

convergence	Calculate and plot convergence of FMM solver

Examples

Calculate and Plot Convergence of FMM Solver

Design a default parabolic reflector antenna.

```
m = reflectorParabolic;
```

Set the solver type of the parabolic reflector antenna to FMM.

```
m.SolverType = 'FMM';
```

Calculate the impedance of the parabolic reflector antenna at 10 GHz.

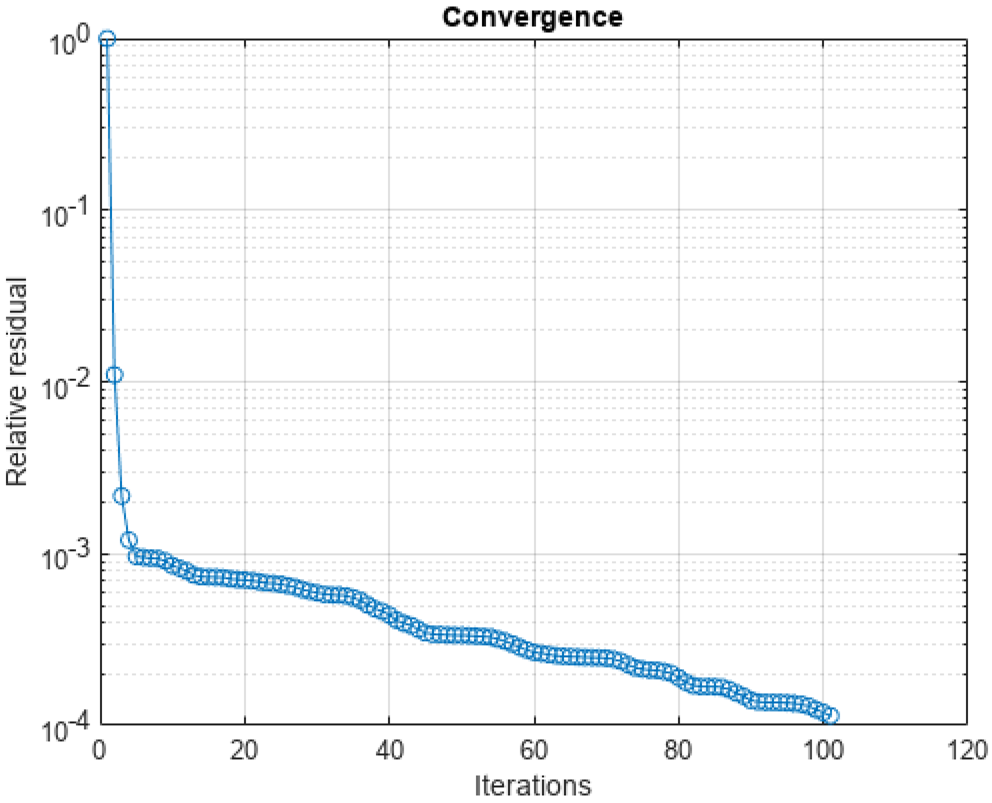
```
Z = impedance(m,10e9);
```

Access the FMM solver and set the relative residual to 1e-3.

```
s = solver(m);
s.RelativeResidual = 1e-3;
```

Calculate and plot the convergence of the FMM solver for the parabolic reflector antenna.

```
convergence(s)
```



Version History

Introduced in R2021b

See Also

“Fast Multipole Method for Large Structures”

Topics

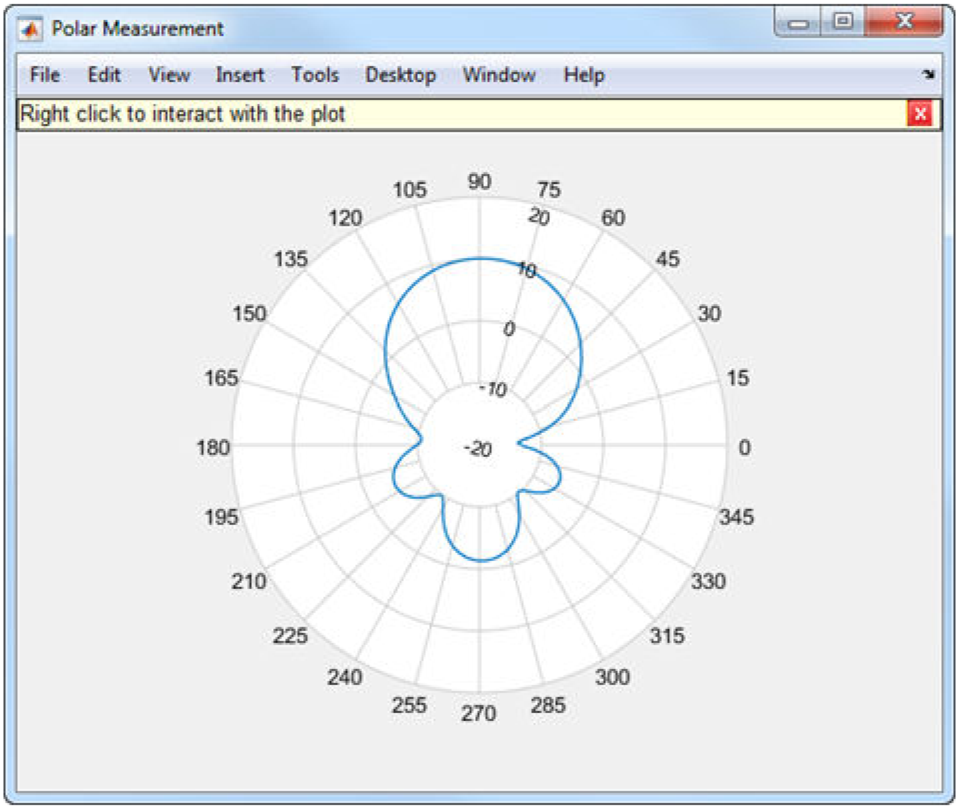
“Analysis of Electrically Large Structures Using Hybrid MoM and FMM”

polarpattern

Interactive plot of radiation patterns in polar format

Description

The `polarpattern` object creates an interactive plot of antenna or array radiation patterns in polar format with uniformly spaced angles. You can also plot other types of polar data. Use this plot for interactive data visualization or measurement. To change the properties, zoom in, or add more data to the plot, right-click or scroll or drag the **Polar Measurement** window.



Creation

Syntax

```
polarpattern
polarpattern(data)
polarpattern(angle,magnitude)
polarpattern( ___ ,Name,Value)
polarpattern(ax, ___ )
p = polarpattern( ___ )
```

```
p = polarpattern('gco')
```

Description

`polarpattern` creates an empty polar plot. You can add plots of antenna or array radiation patterns and other types of data to the plot by importing saved polari objects from MAT-files.

`polarpattern(data)` creates a polar plot with real magnitude values in the vector `data` with angles uniformly spaced on the unit circle starting at `0` degrees. Magnitudes may be negative when dB data units are used. For a matrix `data`, columns of `data` are independent datasets. For *N*-`data` arrays, dimensions 2 and greater are independent datasets. For complex values, magnitude and angle are derived from `data`.

`polarpattern(angle,magnitude)` creates a polar plot for a set of angles and corresponding magnitudes. You can also create polar plots from multiple sets of angle vectors in degrees and corresponding sets of magnitude using the syntax: `polarpattern(angle1, magnitude1,..., angleN, magnitudeN)`.

`polarpattern( ___ ,Name,Value)` creates a polar plot, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding property value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Unspecified properties retain their default values. To list all the property names and values, use `details(p)`. You can use the properties to extract data about the radiation pattern from the polar plot. For example, `p = polarpattern(data,'Peaks',3)` identifies and displays the three highest peaks in the pattern data.

For a list of properties, see PolarPattern.

`polarpattern(ax, ___ )` creates a polar plot using axes object, `ax` instead of the current axes object.

`p = polarpattern( ___ )` creates a polari object using any combination of input arguments from the previous syntaxes. Use this syntax to customize the plot or add measurements.

`p = polarpattern('gco')` creates a polar plot object from the polar pattern in the current figure.

Input Arguments

data — Antenna or array data

real length-*M* vector | real *M*-by-*N* matrix | real multidimensional array | complex vector or matrix

Antenna or array data, specified as one of these options

- A real length-*M* vector, containing *M* magnitude values with angles their defined as $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees.
- A real *M*-by-*N* matrix, containing *M* magnitude values in a dataset and *N* such independent data sets. Each column of the matrix has angles in degrees from the vector $\frac{(0:M-1)}{M} \times 360^{\circ}$.
- A real multidimensional array. Arrays with `2` or more dimensions contain independent data sets.
- A complex vector or matrix, that contains Cartesian coordinates (*x*, *y*) of each point. *x* contains the real part of the `data` and *y* contains the imaginary part of the `data`.

When the data is in a logarithmic form, such as dB, magnitude values can be negative. In this case, `polarpattern` plots the smallest magnitude values at the origin of the polar plot and largest magnitude values at the maximum radius.

Data Types: `double`

`angle` — Set of angles
vector

Set of angles in degrees, specified as a vector.

Data Types: `double`

`magnitude` — Set of magnitude values
vector | matrix

Set of magnitude values, specified as a vector or a matrix. If you specify this input as a matrix, each column is an independent set of magnitude values and corresponds to the same set of angles in the same column of the angle input .

Data Types: `double`

ax — Axes of polar plot
`axes` object

Axes of the polar plot, specified as an axes object.

Output Arguments

p — polari object
`polari` object

Stores a polari object with a set of properties. Use p to modify properties of the plot after creation. For a list of all the properties, see PolarPattern Properties.

Example: `P = polarplot(V)`

Object Functions

add	Add data to polar plot
addCursor	Add cursor to polar plot angle
animate	Replace existing data with new data for animation
createLabels	Create legend labels for polar plot
findLobes	Main, back, and side lobe data
replace	Replace polar plot data with new data
showPeaksTable	Show or hide peak marker table
showSpan	Show or hide angle span between two markers

Examples

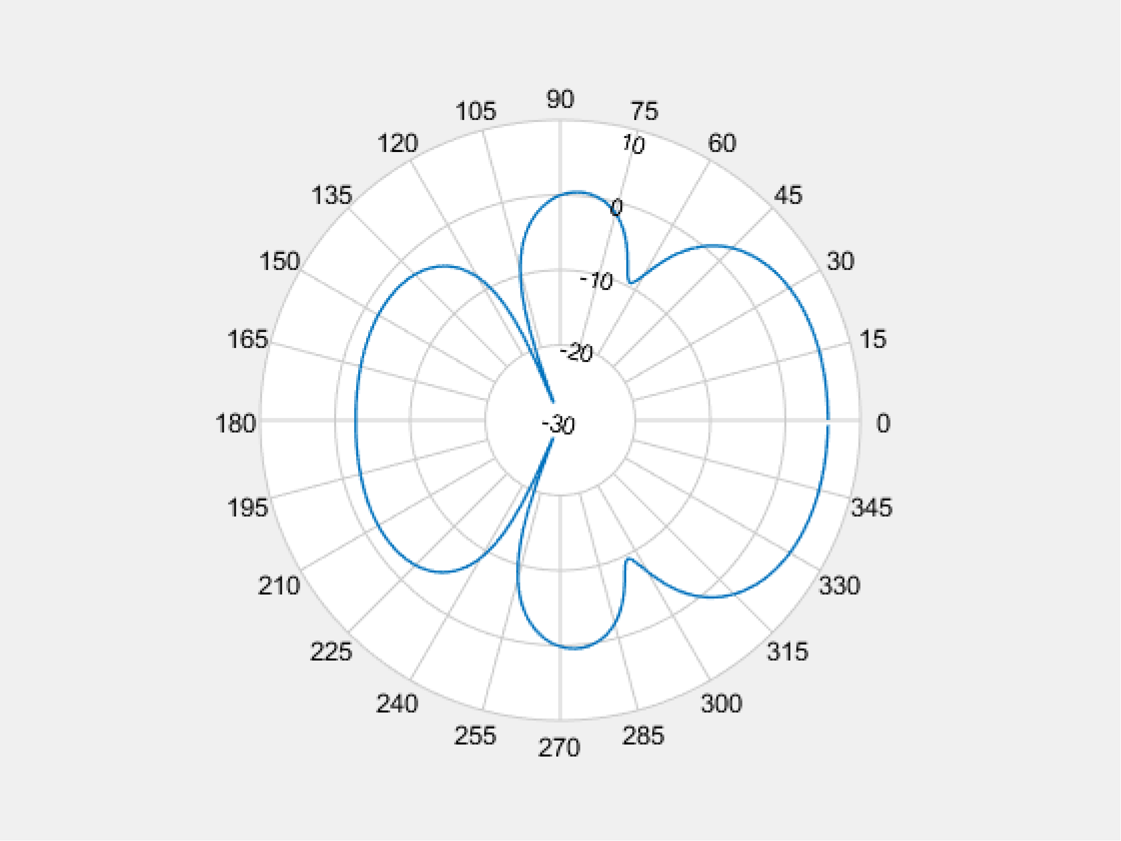
Polar Pattern for Vivaldi Antenna

Create a default Vivaldi antenna and calculate the directivity at 1.5 GHz.

```
v = vivaldi;
V = pattern(v,1.5e9,0,0:1:360);
```

Plot the polar pattern of the calculated directivity.

```
P = polarpattern(V);
```



Polar Pattern of Cavity Antenna

Create a default cavity antenna. Calculate the directivity of the antenna and write the data to `cavity.pln` using the `msiwrite` function.

```
c = cavity;
msiwrite(c,2.8e9,'cavity','Name','Cavity Antenna Specifications');
```

Read the cavity specification file into `Horizontal`, `Vertical`, and `Optional` structures using the `msiread` function.

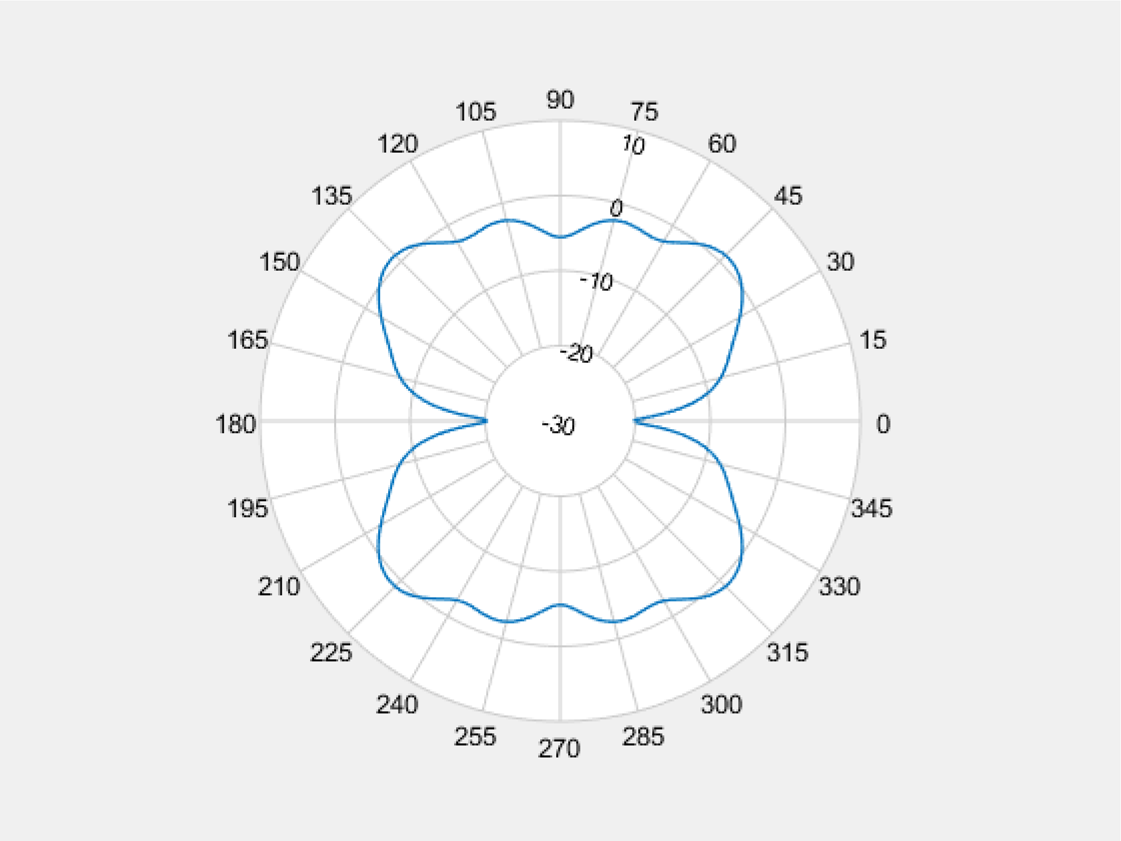
```
[Horizontal,Vertical,Optional] = msiread('cavity.pln')
```

```
Horizontal = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: [360x1 double]
           Elevation: 0
           Frequency: 2.8000e+09
               Slice: 'Elevation'
```

```
Vertical = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: 0
           Elevation: [360x1 double]
           Frequency: 2.8000e+09
               Slice: 'Azimuth'


Optional = struct with fields:
         name: 'Cavity Antenna Specifications'
    frequency: 2.8000e+09
         gain: [1x1 struct]
```

Plot the polar pattern of the cavity at azimuth angles.

```
P = polarpattern(Horizontal.Azimuth,Horizontal.Magnitude);
```



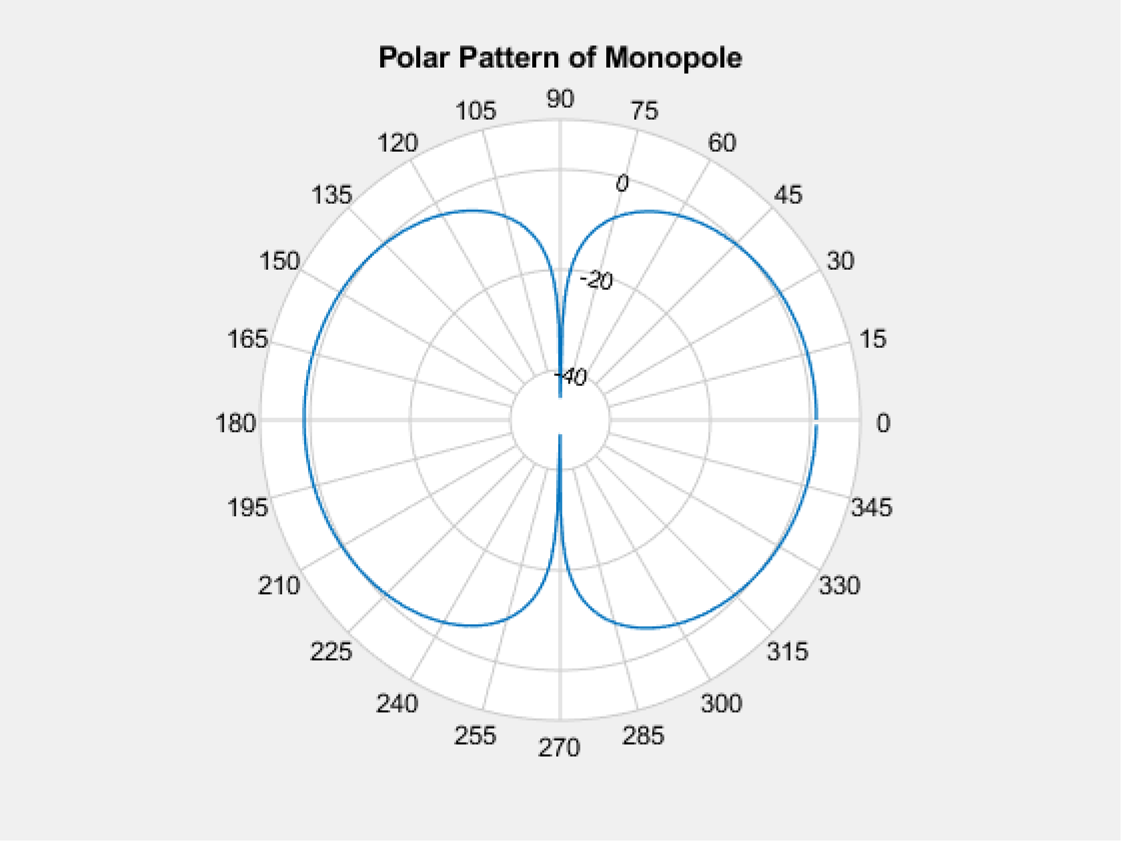
Add Title to Polar Plot

Create a default monopole antenna and calculate the directivity at 75 MHz.

```
m = monopole;
M = pattern(m,75e6,0,0:1:360);
```

Plot the polar pattern of the antenna.

```
P = polarpattern(M,'TitleTop','Polar Pattern of Monopole');
```



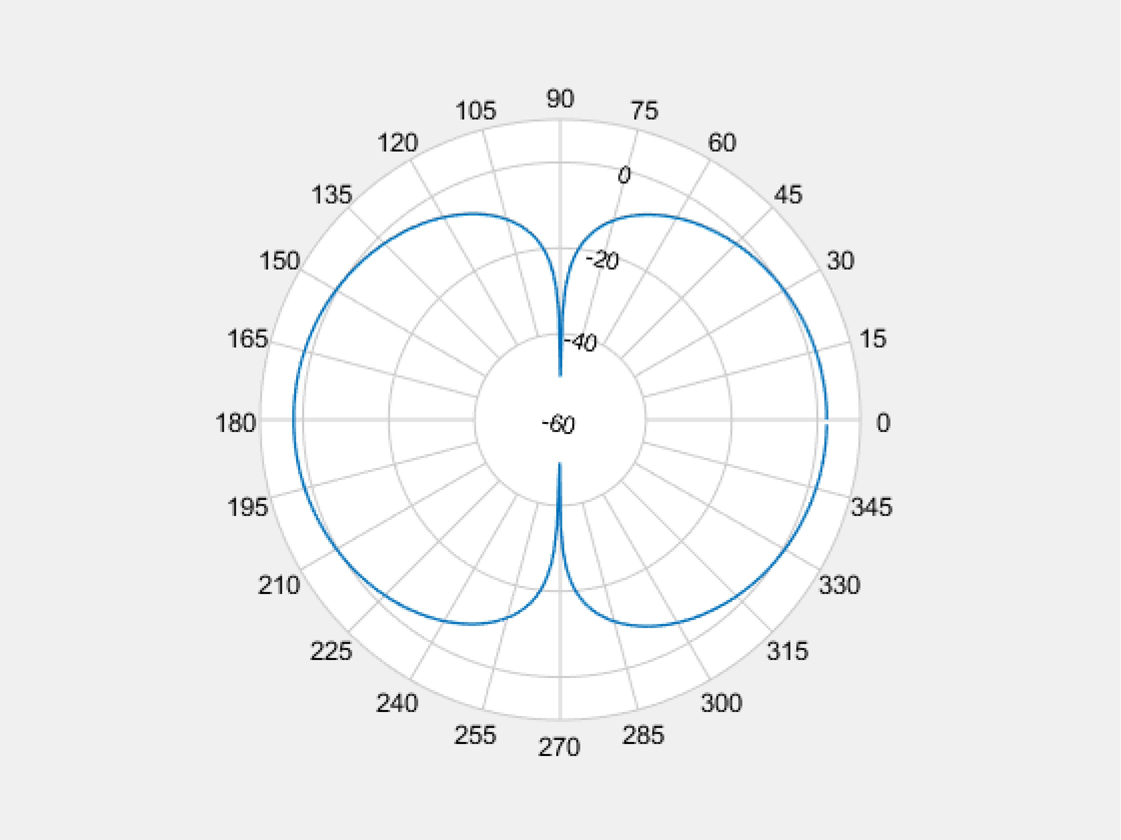
Polar Pattern Properties

Create a default dipole antenna and calculate the directivity at 75 MHz.

```
d = dipole;
D = pattern(d,75e6,0,0:1:360);
```

Plot the polar pattern of the antenna and display the properties of the plot.

```
P = polarpattern(D);
```



```
details(P)
```

```
  internal.polari handle with properties:

                   Interactive: 1
                  LegendLabels: ''
                AntennaMetrics: 0
                     CleanData: 1
                     AngleData: [361x1 double]
                 MagnitudeData: [361x1 double]
                 IntensityData: []
                  AngleMarkers: [0x1 struct]
                 CursorMarkers: [0x1 struct]
                   PeakMarkers: [0x1 struct]
                 ActiveDataset: 1
               AngleLimVisible: 0
                 LegendVisible: 0
                          Span: 0
                      TitleTop: ''
                   TitleBottom: ''
                         Peaks: []
                      FontSize: 10
                  MagnitudeLim: [-60 10]
            MagnitudeAxisAngle: 75
                 MagnitudeTick: [-60 -40 -20 0]
       MagnitudeTickLabelColor: 'k'
                      AngleLim: [0 360]
                AngleTickLabel: {1x24 cell}
```

```
              AngleTickLabelColor: 'k'
       TitleTopFontSizeMultiplier: 1.1000
    TitleBottomFontSizeMultiplier: 0.9000
               TitleTopFontWeight: 'bold'
            TitleBottomFontWeight: 'normal'
          TitleTopTextInterpreter: 'none'
       TitleBottomTextInterpreter: 'none'
                   TitleTopOffset: 0.1500
                TitleBottomOffset: 0.1500
                         ToolTips: 1
               MagnitudeLimBounds: [-Inf Inf]
      MagnitudeFontSizeMultiplier: 0.9000
          AngleFontSizeMultiplier: 1
                       AngleAtTop: 90
                   AngleDirection: 'ccw'
                  AngleResolution: 15
           AngleTickLabelRotation: 0
             AngleTickLabelFormat: '360'
          AngleTickLabelColorMode: 'contrast'
                     PeaksOptions: {}
            AngleTickLabelVisible: 1
                            Style: 'line'
                        DataUnits: 'dB'
                     DisplayUnits: 'dB'
                    NormalizeData: 0
                 ConnectEndpoints: 0
              DisconnectAngleGaps: 0
                        EdgeColor: 'k'
                        LineStyle: '-'
                        LineWidth: 1
                         FontName: 'Helvetica'
                     FontSizeMode: 'auto'
              GridForegroundColor: [0.8000 0.8000 0.8000]
              GridBackgroundColor: 'w'
                 DrawGridToOrigin: 0
                     GridOverData: 0
               GridAutoRefinement: 0
                        GridWidth: 0.5000
                      GridVisible: 1
                         ClipData: 1
                  TemporaryCursor: 1
                 MagnitudeLimMode: 'auto'
           MagnitudeAxisAngleMode: 'auto'
                MagnitudeTickMode: 'auto'
      MagnitudeTickLabelColorMode: 'contrast'
        MagnitudeTickLabelVisible: 1
                   MagnitudeUnits: ''
                   IntensityUnits: ''
                           Marker: 'none'
                       MarkerSize: 6
                           Parent: [1x1 Figure]
                         NextPlot: 'replace'
                       ColorOrder: [7x3 double]
                  ColorOrderIndex: 1
                     SectorsColor: [16x3 double]
                     SectorsAlpha: 0.5000
                             View: 'full'
                    ZeroAngleLine: 0
```

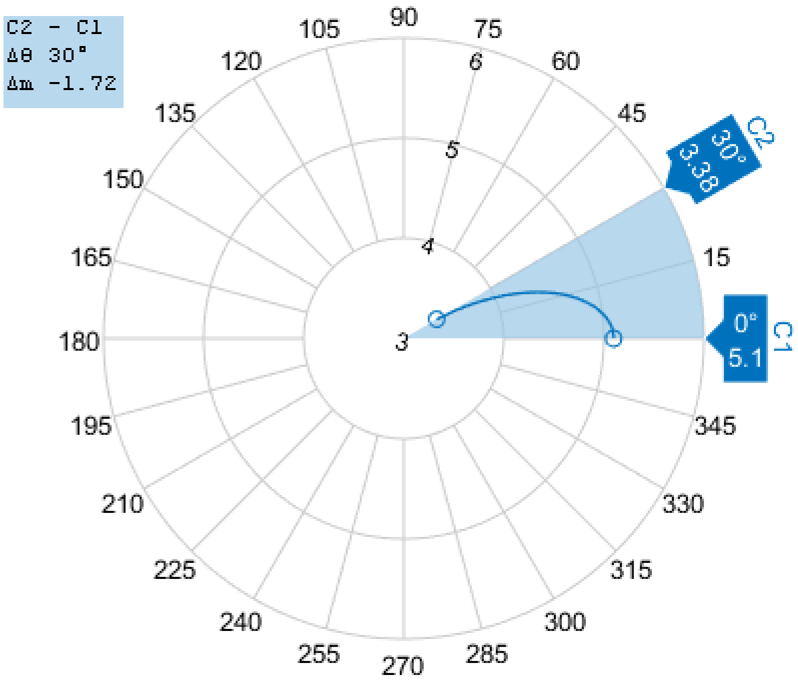
Remove -inf and NaN Values in Antenna Polar Pattern

Remove -inf and NaN values from monopole antenna polar pattern data by using the CleanData and AntennaMetrics properties of a polari object. Use CleanData for partial data with -inf and NaN values.

```
m = monopole;
m.GroundPlaneLength = inf;
```

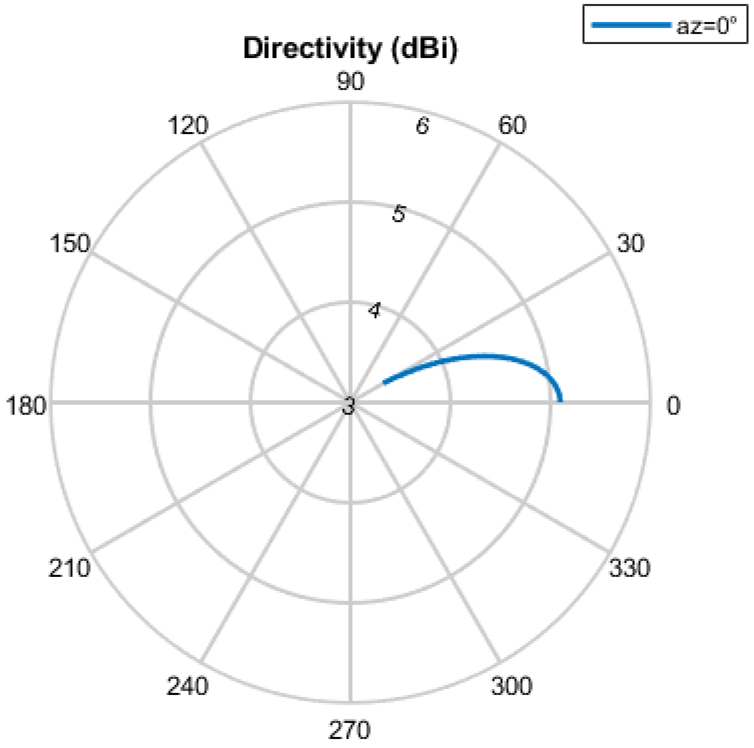
Plot the beamwidth of the antenna at 70 MHz.

```
figure;
beamwidth(m,70e6,0,-50:30)
```



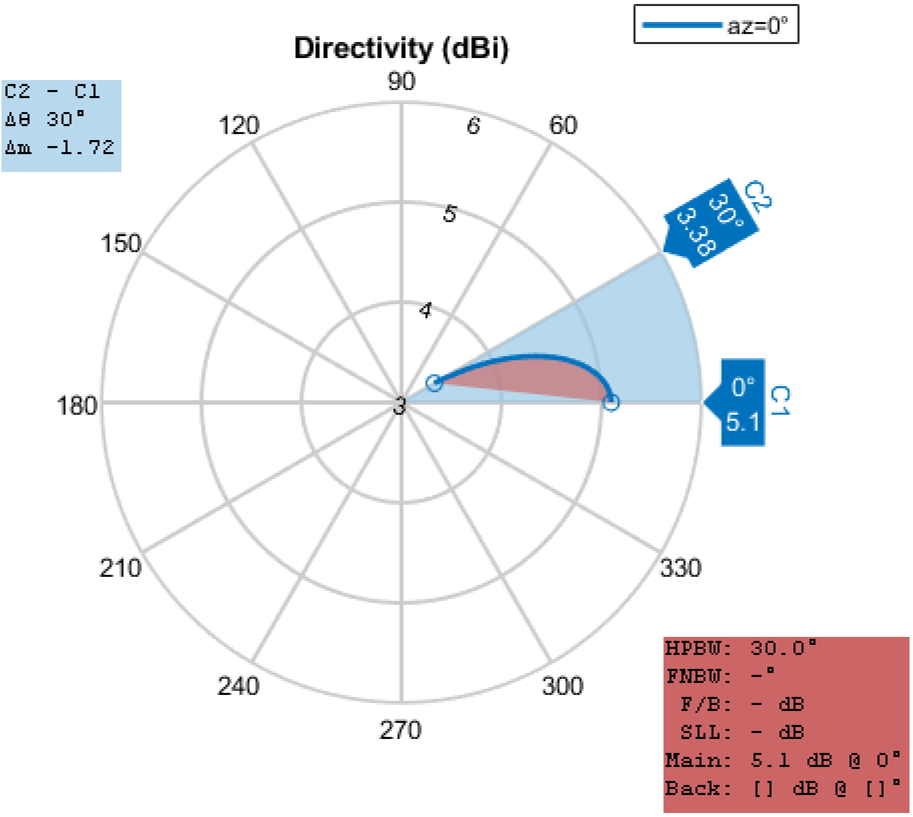
Plot the radiation pattern of the antenna at 70 MHz.

```
figure;
pattern(m,70e6,0,-50:30);
```



Use `polarpattern` to view the antenna metrics of the radiation pattern.

```
P = polarpattern('gco');
P.CleanData = 1;
P.AntennaMetrics = 1;
```

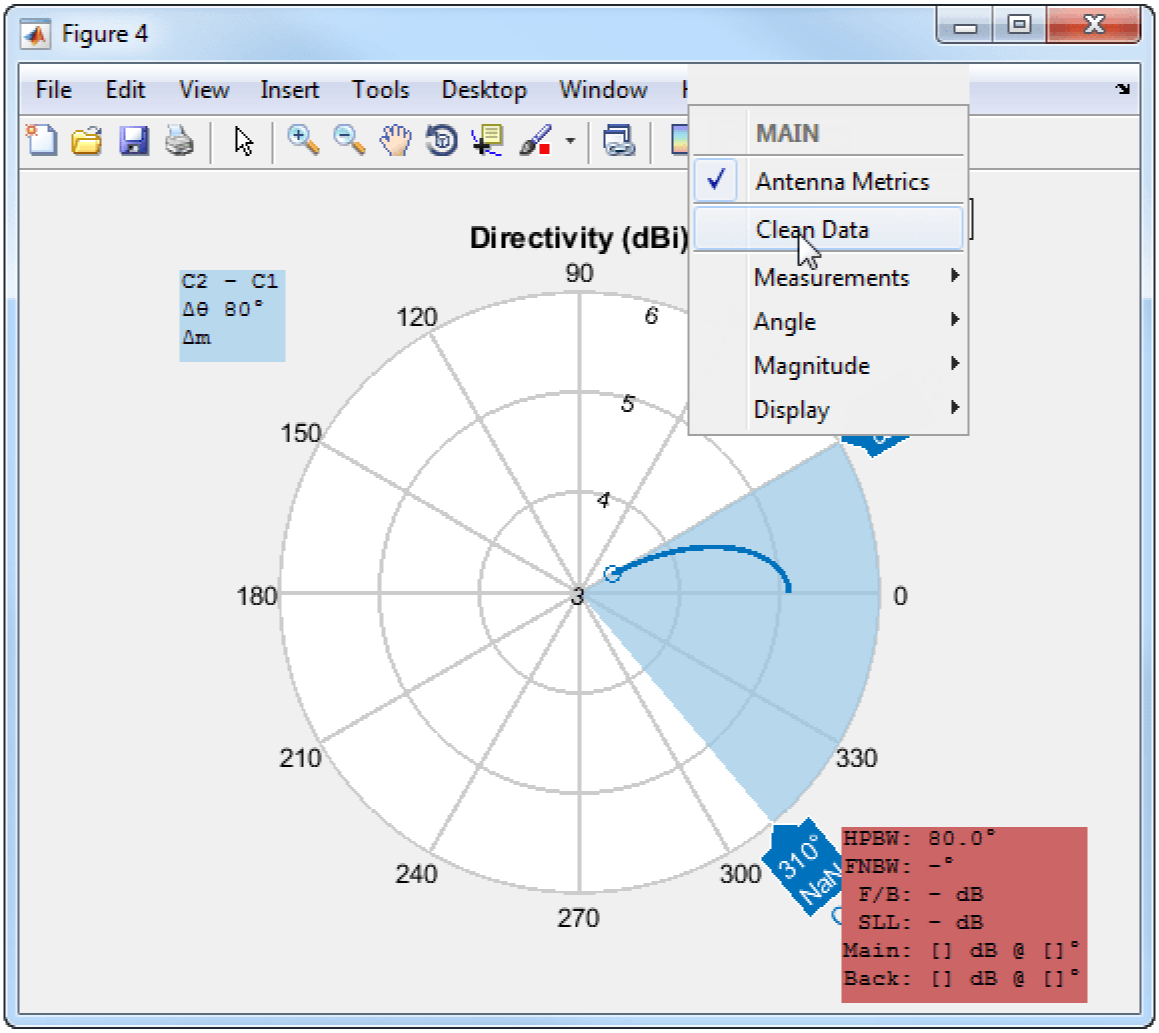


```
C2 - C1
Δθ 30°
Δm -1.72
```

```
HPBW:  30.0°
FNBW:  -°
 F/B:  - dB
 SLL:  - dB
Main:  5.1 dB @ 0°
Back:  [] dB @ []°
```

Compare the `beamwidth` plot and the `polarpattern` plot. The Antenna Metrics does not represent the beamwidth correctly.

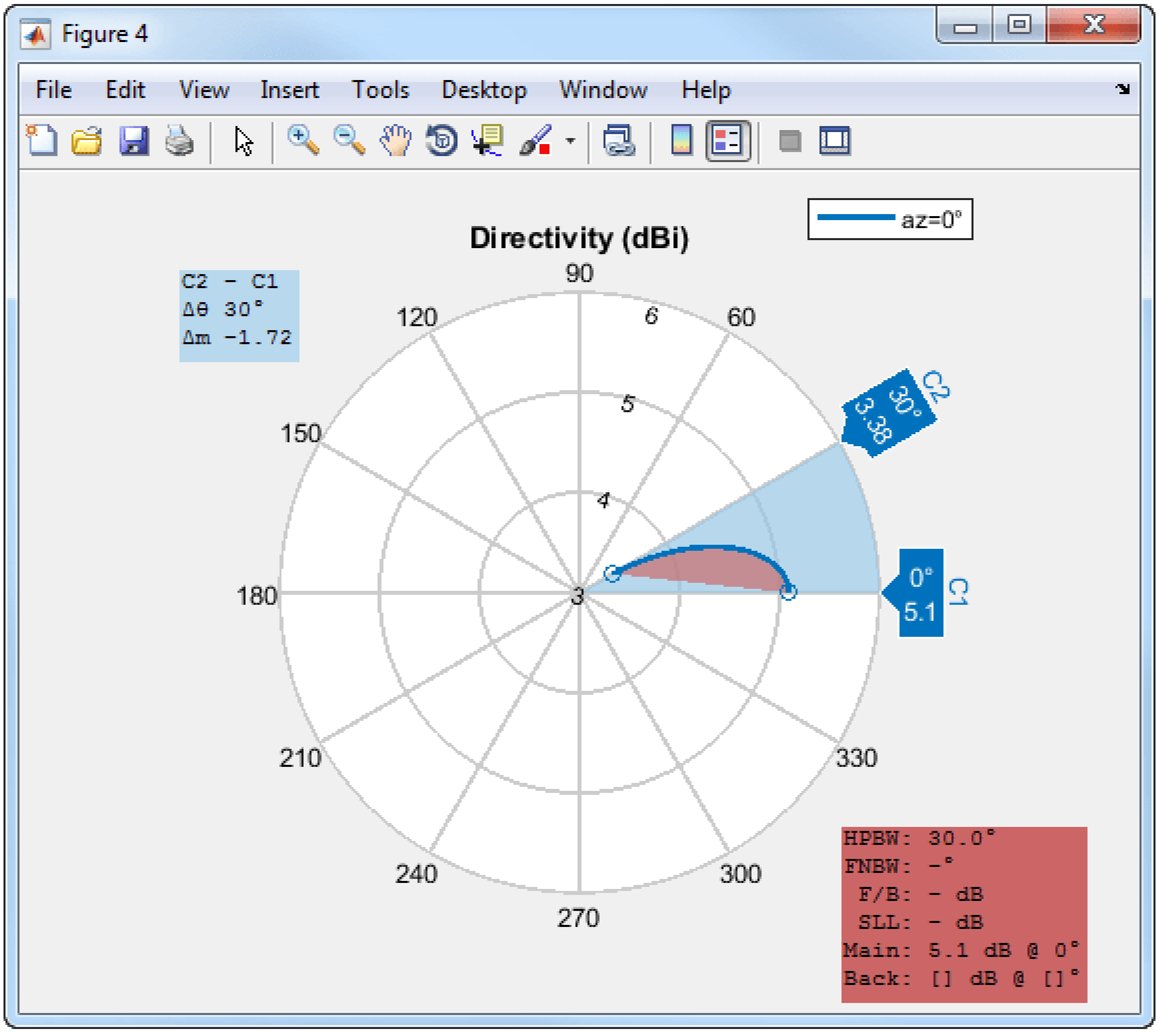
You can also clean the data by right clicking on the plot and selecting **Clean Data.**



```
C2 - C1
Δθ 80°
Δm
```

```
HPBW: 80.0°
FNBW: -°
 F/B: - dB
 SLL: - dB
Main: [] dB @ []°
Back: [] dB @ []°
```

After you clean the data, the `polarpattern` plot calculation matches the `beamwidth` plot calculation.



```
C2 - C1
Δθ 30°
Δm -1.72
```

```
HPBW: 30.0°
FNBW: -°
 F/B: - dB
 SLL: - dB
Main: 5.1 dB @ 0°
Back: [] dB @ []°
```

Version History

Introduced in R2016a

See Also

PolarPattern Properties

Topics

"Interact with Polar Plot"

2

Apps

Antenna Designer

Design, visualize, and analyze antennas

Description

The **Antenna Designer** app lets you design, visualize, and analyze antennas in the Antenna Toolbox library interactively.

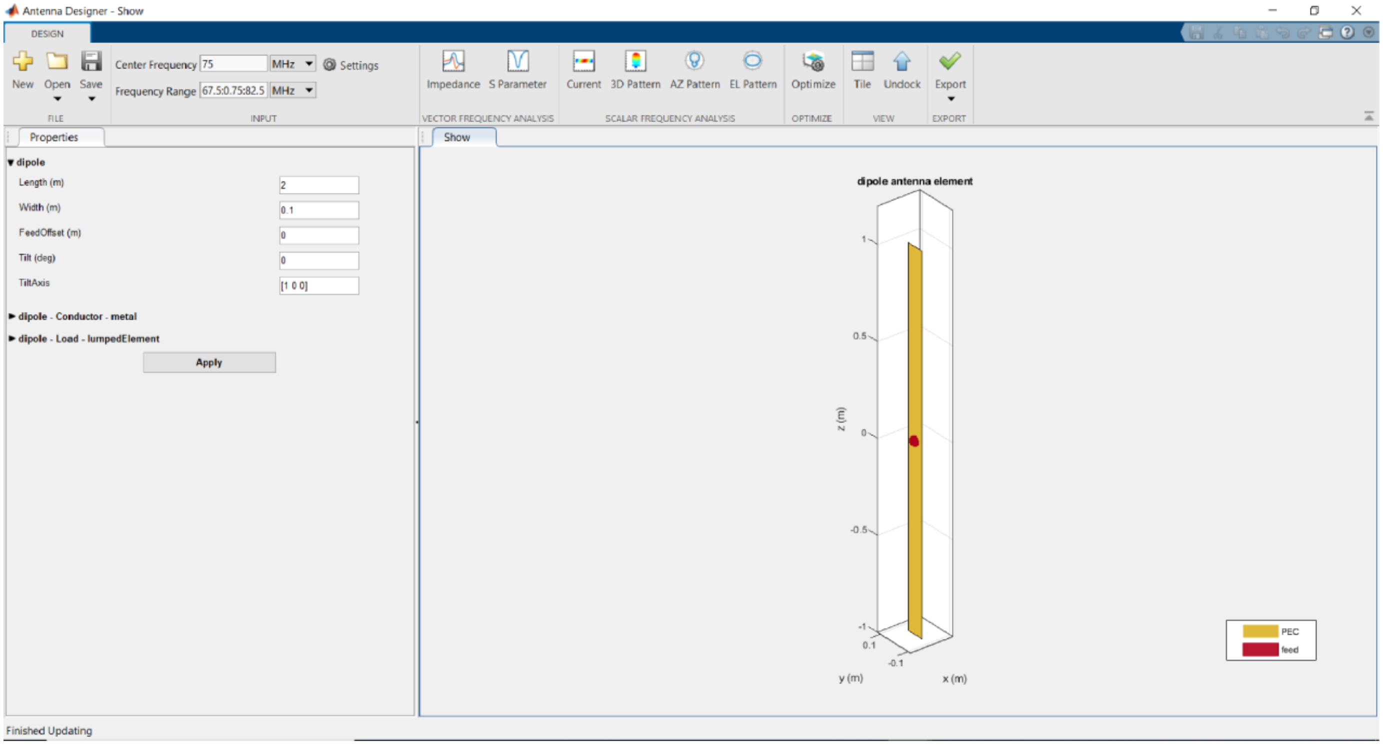
Using this app, you can:

- Select antennas based on general properties or antenna performance.
- Select backing structures from the gallery of backing structures.
- Visualize antennas based on frequency and frequency range.
- Analyze antennas based on radiation pattern, polarization, and bandwidth.
- Export selected and designed antennas as a variable to the MATLAB workspace, as either script or a variable. The exported MATLAB script has two sections: `Antenna Properties` and `Antenna Analysis`.
- Save and load an existing antenna .mat file to the app and analyze the antenna.
- Optimize antennas for various analysis results under given constraints using SADEA or Surrogate optimization methods.

Note

- To use **Parallel Computing** for `SADEA` optimizer, you need the Parallel Computing Toolbox™.

To use the Surrogate optimization algorithm, you need the Global Optimization Toolbox.



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

▾ dipole

Length (m) 2

Width (m) 0.1

FeedOffset (m) 0

Tilt (deg) 0

TiltAxis [1 0 0]

▸ dipole - Conductor - metal

▸ dipole - Load - lumpedElement

Finished Updating

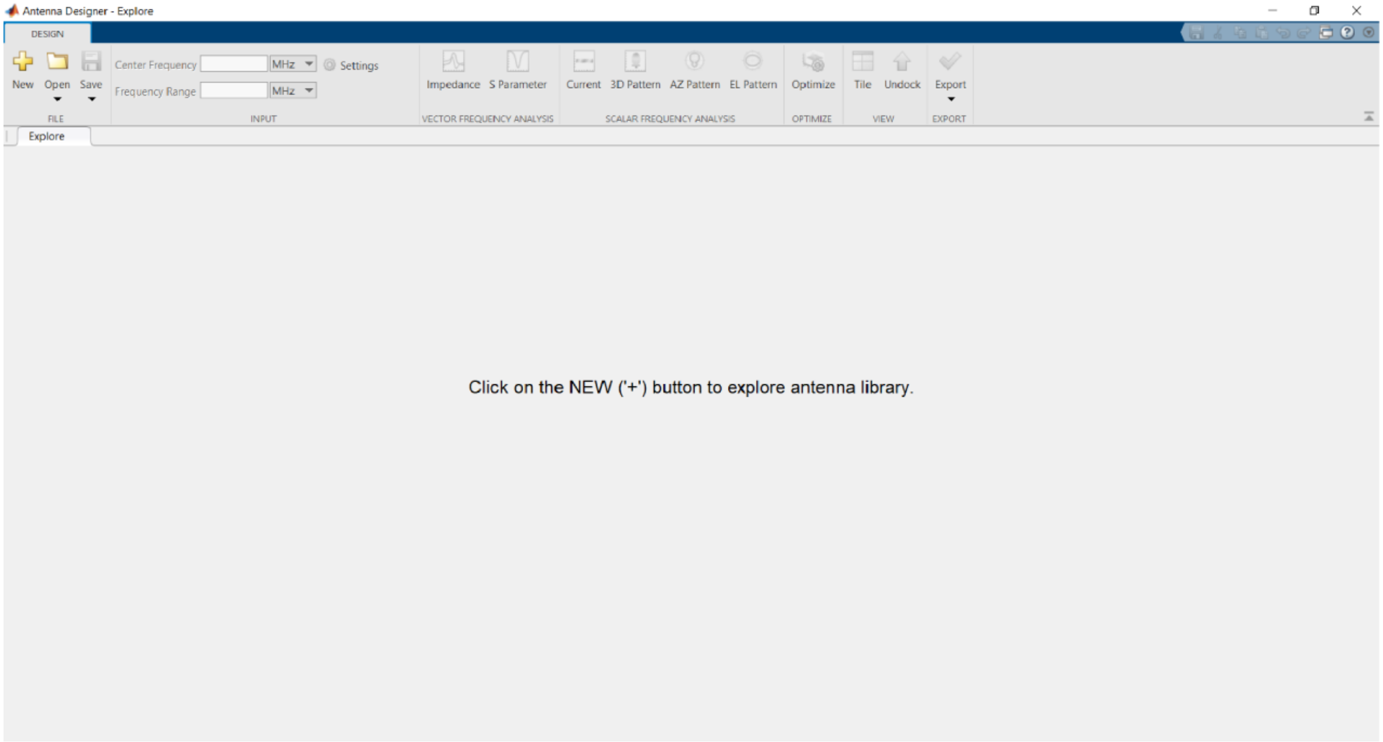
Open the Antenna Designer App

- MATLAB Toolstrip: In the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `antennaDesigner`.

Examples

Antenna Designer Canvas

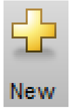
The **Antenna Designer** opens a blank canvas.



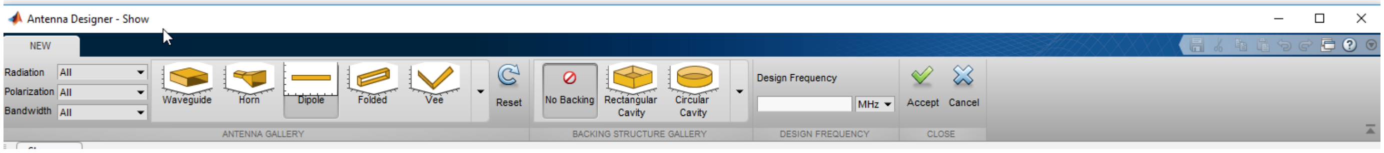
Click on the NEW ('+') button to explore antenna library.

1 **Select and Visualize Antenna**

- Click

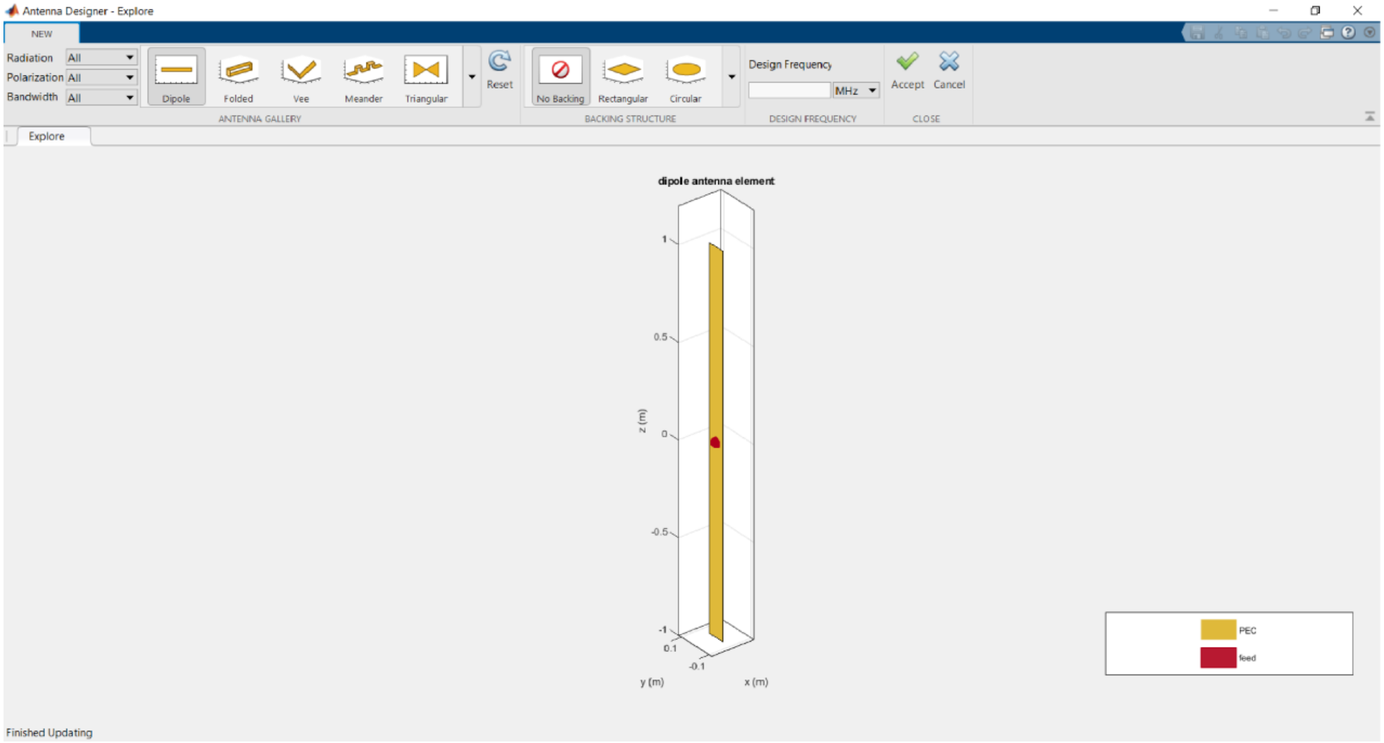
  in the canvas toolstrip to choose the antenna you want to analyze.
- The default antenna is a dipole antenna.



  You can filter the antennas based on `Radiation` pattern, `Polarization`, and `Bandwidth`.
- Using the toolstrip you can also add **Cavity** backing, or **Reflector** backing to the antennas.
- You can also specify the **Design Frequency** of the antenna. Setting this value scales the antenna to resonate at the specified frequency. You can also tune the antenna using **Antenna Properties** tab during analysis.
- Use **Reset**, to go back to default settings.
- Use **Accept**, to analyze the antenna characteristics.
- Use **Cancel**, to start over.

2 Antenna Gallery

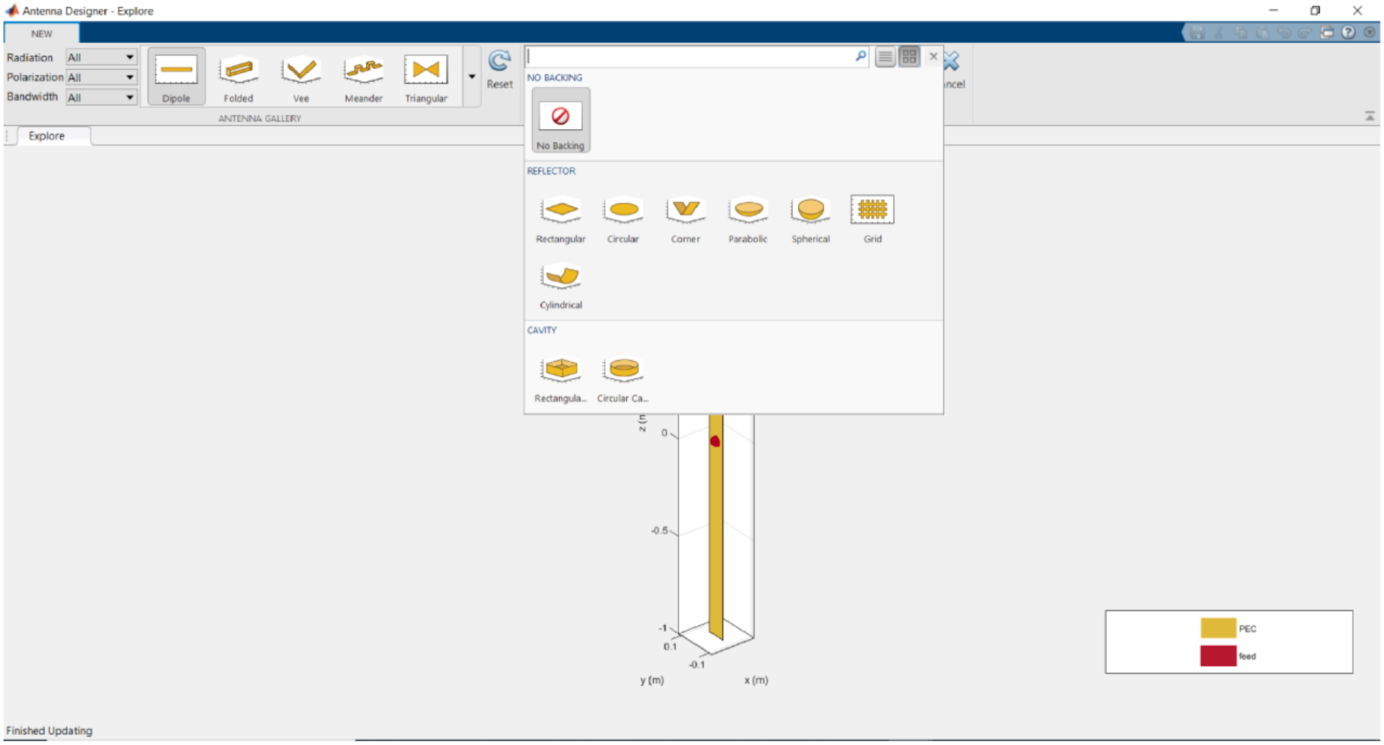
- You can choose your antennas from the **ANTENNA GALLERY**.



- When you filter antennas based on `Radiation` pattern, `Polarization`, or `Bandwidth`, the antenna gallery greys out the antennas that do not belong to the chosen filter.

3 Back Structure Gallery

- You can choose your antenna backing structures from the **BACKING STRUCTURE GALLERY**.

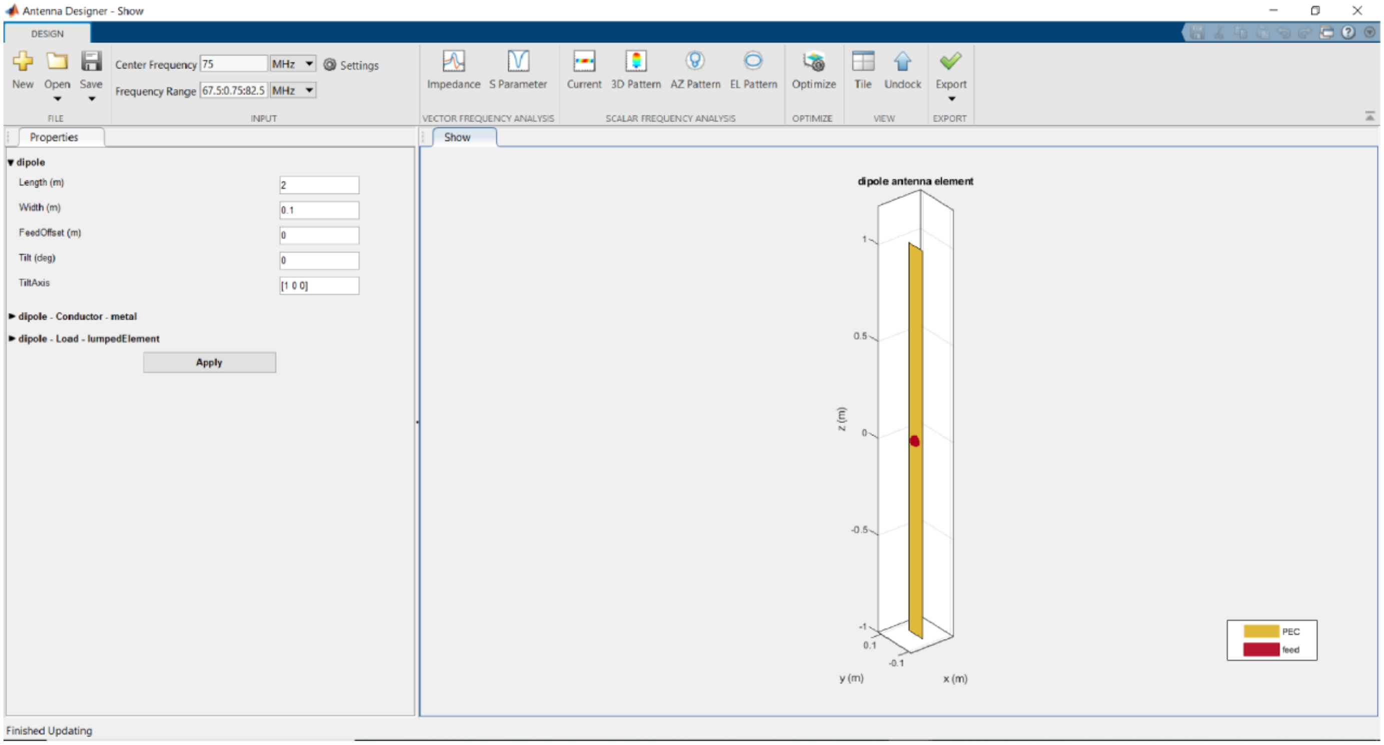


Antenna Designer - Explore

Radiation All
Polarization All
Bandwidth All

Finished Updating

4 Analyze Antenna



Antenna Designer - Show

Center Frequency 75 MHz
Frequency Range 67.5:0.75:82.5 MHz

Properties

▾ dipole

Length (m)	2
Width (m)	0.1
FeedOffset (m)	0
Tilt (deg)	0
TiltAxis	[1 0 0]

▸ dipole - Conductor - metal
▸ dipole - Load - lumpedElement

Apply

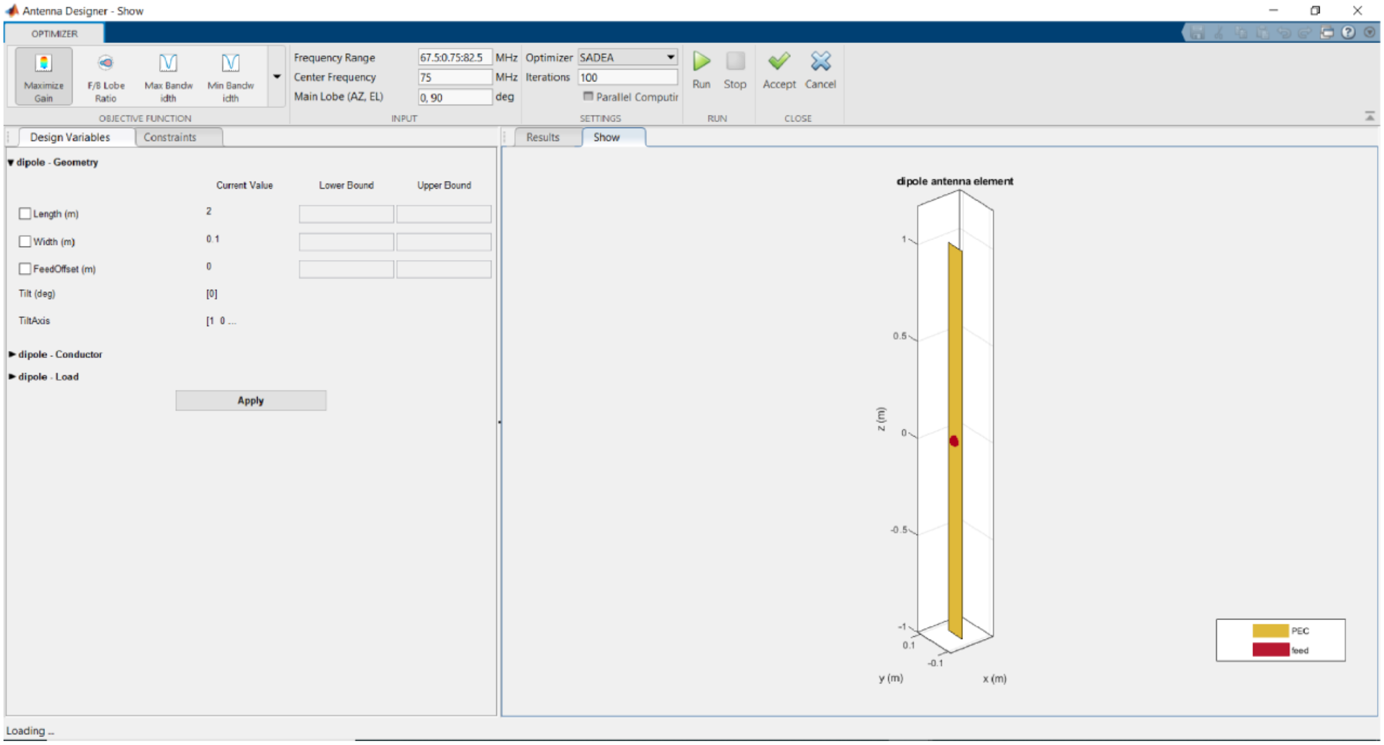
Finished Updating

- You can plot the **Impedance** and **S Parameter** of the antenna based on the specified **Frequency Range** in Hz.

- You can visualize the **Current** distribution on the antenna based on the specified **Frequency** in Hz.
- You can visualize the **3D Pattern**, **AZ Pattern**, **EL Pattern** of the antenna based on the specified frequency. Here AZ stands for azimuth and EL stands for elevation.
- Use **Export** to view your antenna in MATLAB workspace or MATLAB script.
- Manually change the antenna properties using the **Antenna Properties** tab. In this tab, you can change the geometrical properties of the antenna, add a dielectric substrate and metal conductor parameters to the antenna, and change the value and location of the load.

5 Optimize Antenna

- Click on **Optimize** to open the optimizer canvas of the antenna designer app.


Antenna Designer - Show

Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 100
□ Parallel Computir

▾ dipole - Geometry

	Current Value	Lower Bound	Upper Bound
□ Length (m)	2		
□ Width (m)	0.1		
□ FeedOffset (m)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		

▸ dipole - Conductor
▸ dipole - Load

Loading ...

Use the **OBJECTIVE FUNCTION**to choose the main goal of optimizing the antenna.

- Use the **Design Variables** to input the variables The variables are then changed by the optimizer depending on the lower and upper bounds.
- Use **Constraints** functions to restrict a desired analysis function value on the antenna.
- Use the `Optimizer` to choose between SADEA or `Surrogate Opt`.

Note

- To use **Parallel Computing** for SADEA optimizer, you need the Parallel Computing Toolbox.

To use the Surrogate optimization algorithm, you need the Global Optimization Toolbox.

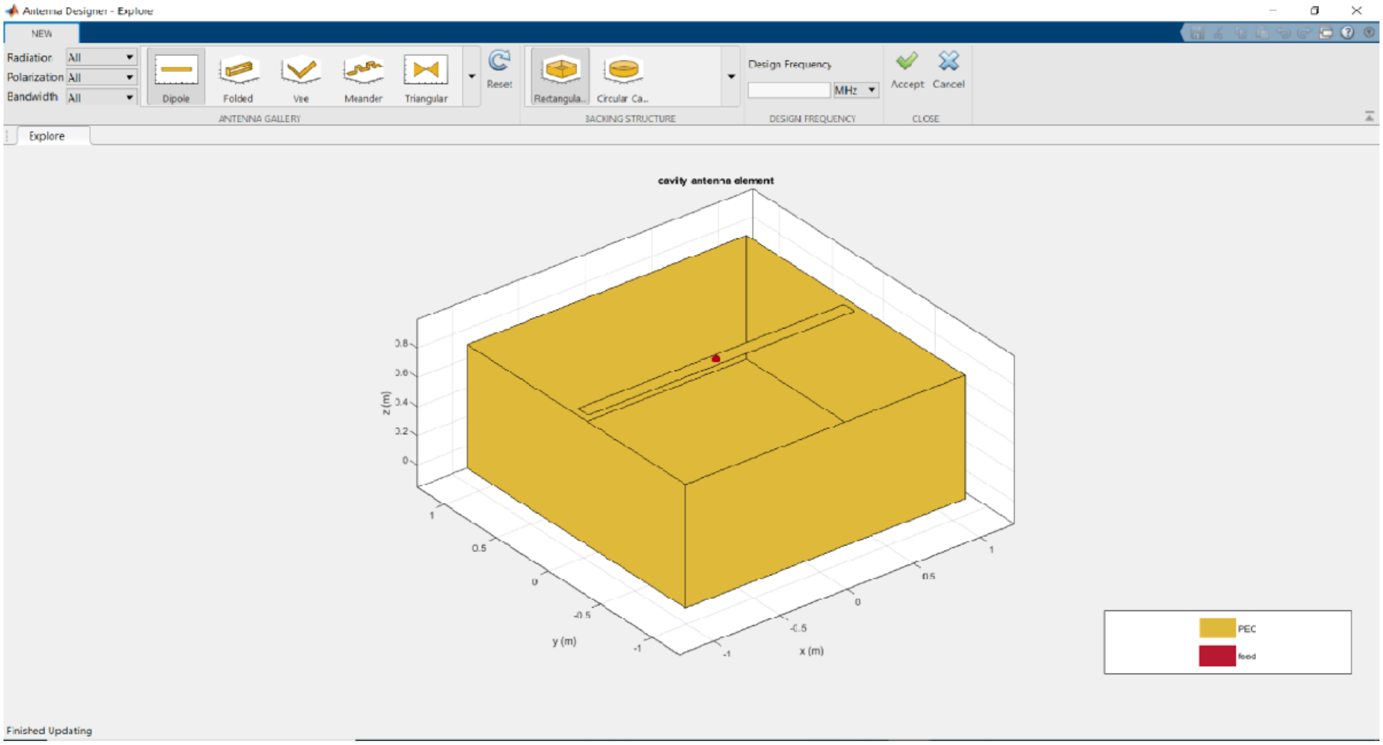
- After adding the required values, click **Run** to start the optimization.

Plot Radiation Pattern of Cavity-Backed Dipole

Use the **Antenna Designer** app to plot the radiation pattern of a cavity-backed dipole antenna.

Open the app and click **New** to show the default dipole antenna.

From the **BACKING STRUCTURE GALLERY,** click **Rectangular Cavity** to create a cavity-backed dipole antenna.

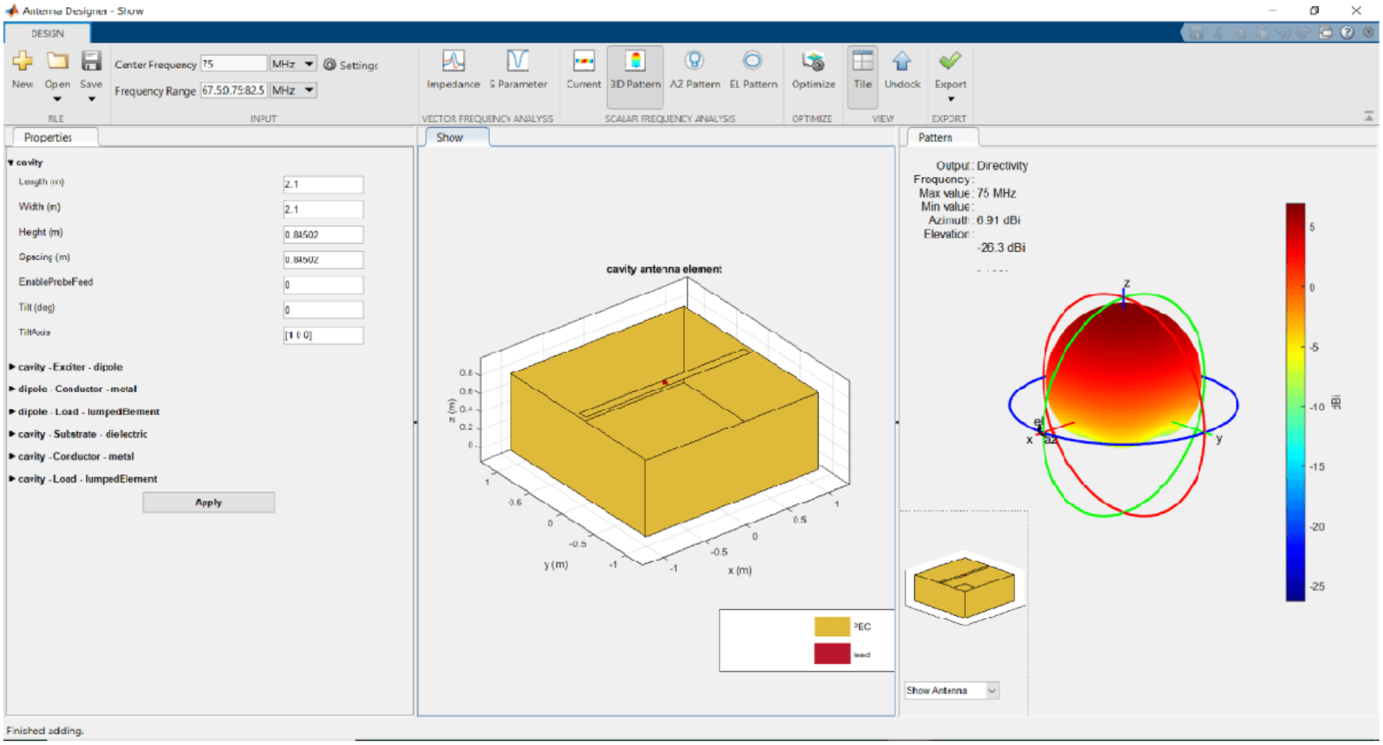


Radiation All
Polarization All
Bandwidth All

Finished Updating

Click **Accept.**

In **SCALAR FREQUENCY ANALYSIS**, click **3D Pattern** to calculate the radiation pattern of the cavity-backed dipole. The default frequency used is 75 MHz. Click **Tile** to view both the antenna and the radiation pattern.



Antenna Designer - Show

Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

Properties

▾ cavity

Length (m) 2.1

Width (m) 2.1

Height (m) 0.84502

Spacing (m) 0.84502

EnableProbeFeed 0

Tilt (deg) 0

TiltAxis [1 0 0]

▸ cavity - Exciter - dipole

▸ dipole - Conductor - metal

▸ dipole - Load - lumpedElement

▸ cavity - Substrate - dielectric

▸ cavity - Conductor - metal

▸ cavity - Load - lumpedElement

Output: Directivity
Frequency: 75 MHz
Max value: 6.91 dBi
Min value: -26.3 dBi
Azimuth:
Elevation:

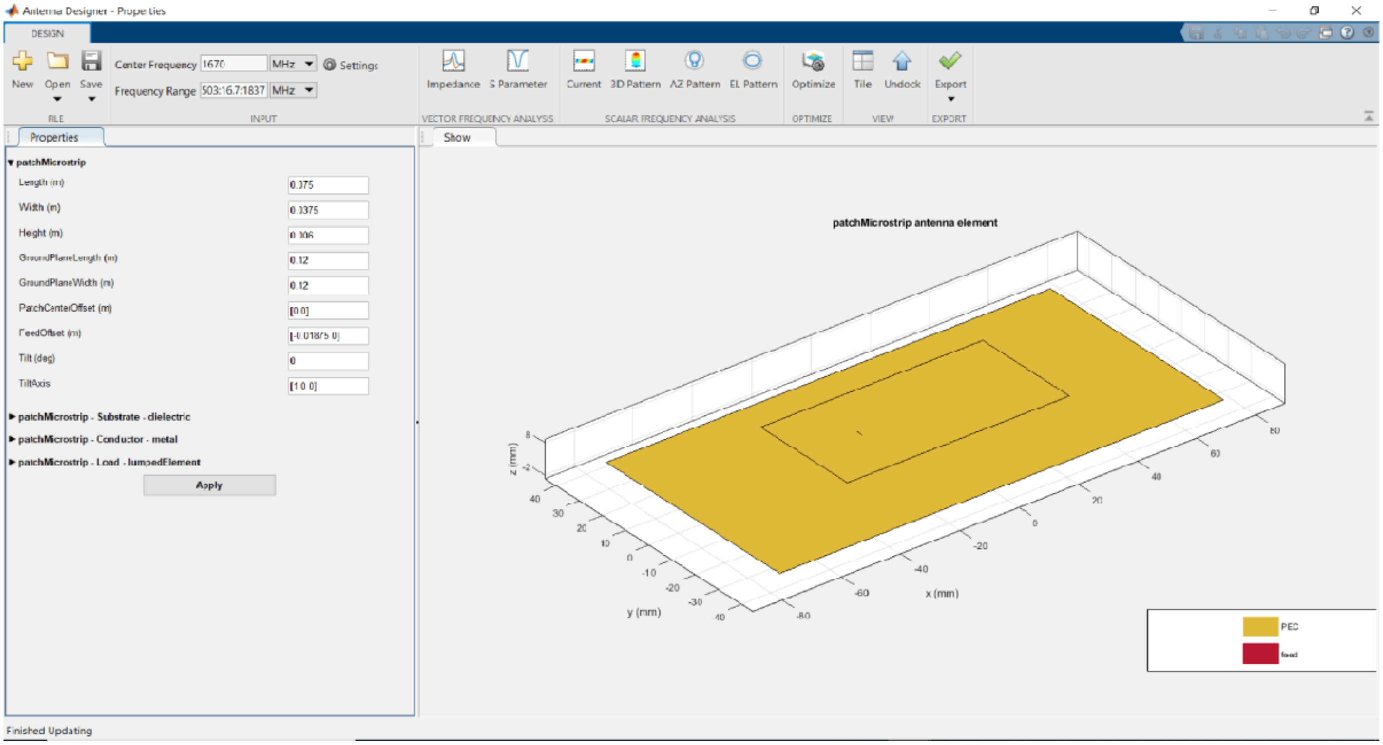
Finished adding.

Analyze Patch Microstrip Antenna Having Dielectric Substrate

Use the **Antenna Designer** app to plot the radiation pattern of a patch microstrip antenna with a dielectric substrate.

Open the app and click **New**. In the **ANTENNA GALLERY** section, under **PATCH FAMILY**, click `Microstrip.` Click **Accept**.

On the **Antenna Properties** tab, change the groundplane length and groundplane width to 0.120 m. Click **Apply** to see the changes.



Center Frequency 1670 MHz

Frequency Range 503:16.7:1837 MHz

▾ patchMicrostrip

Length (m)	0.075
Width (m)	0.0375
Height (m)	0.006
GroundPlaneLength (m)	0.12
GroundPlaneWidth (m)	0.12
PatchCenterOffset (m)	[0 0]
FeedOffset (m)	[-0.01875 0]
Tilt (deg)	0
TiltAxis	[1 0 0]

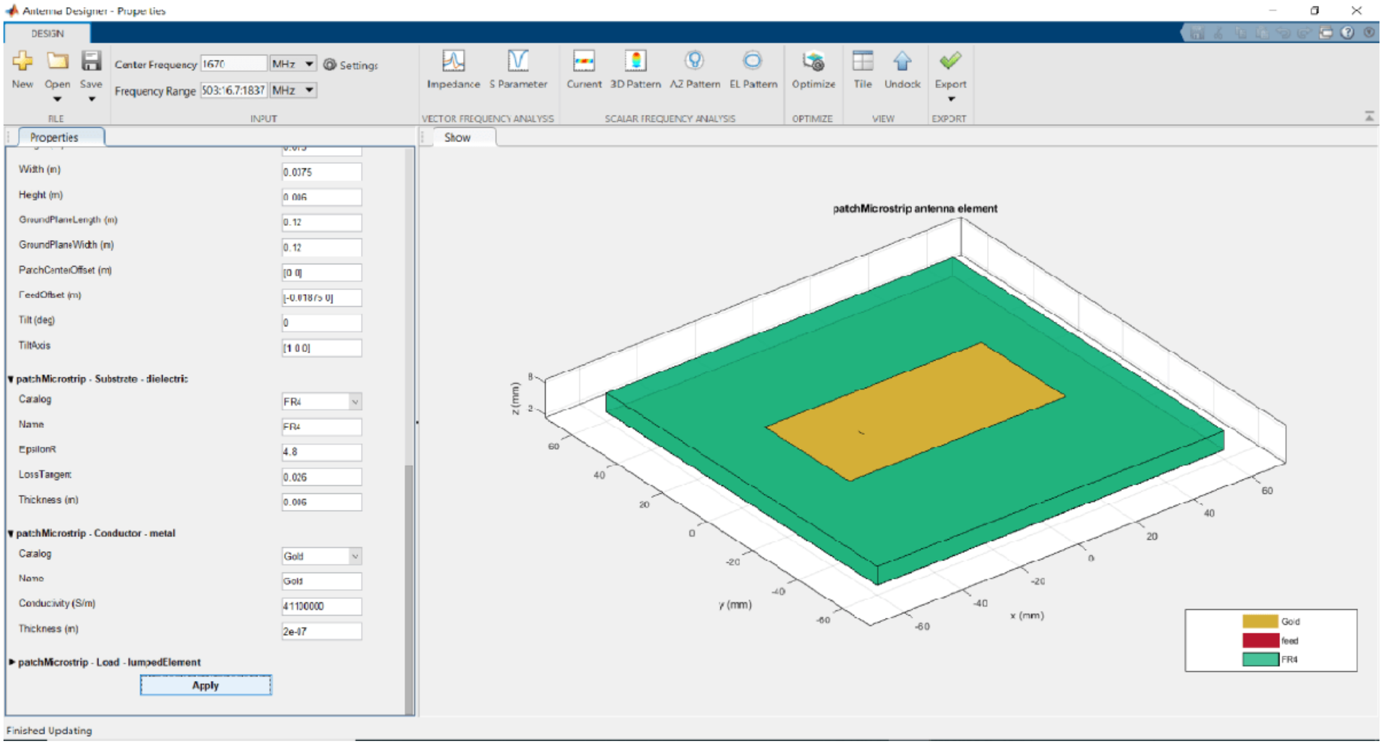
▸ patchMicrostrip - Substrate - dielectric

▸ patchMicrostrip - Conductor - metal

▸ patchMicrostrip - Load - lumpedElement

Finished Updating

Add an FR4 dielectric as a substrate to the patch microstrip antenna. To add the dielectric, open the **Substrate** section and select required dielectric from the `Dielectric Catalog drop-down.` Set the substrate **Name** to FR4, **EpsilonR** to 4.8000, and **Loss Tangent** to 0.0260. Also add `Gold` metal as a conductor to the patch microstrip antenna. To add the metal, open the **Conductor** section and select required metal from the `Metal Catalog drop-down.` You can also use custom dielectric or metal materials by setting fields: `Name`, `EpsilonR`, `Loss Tangent`, `Conductivity` and `Thickness`. Click **Apply** to see the antenna.



Center Frequency 1670 MHz

Frequency Range 503:16.7:1837 MHz

Width (m) 0.0375

Height (m) 0.006

GroundPlaneLength (m) 0.12

GroundPlaneWidth (m) 0.12

PatchCenterOffset (m) [0 0]

FeedOffset (m) [-0.01875 0]

Tilt (deg) 0

TiltAxis [1 0 0]

▾ patchMicrostrip - Substrate - dielectric

Catalog FR4

Name FR4

EpsilonR 4.8

LossTangent 0.026

Thickness (m) 0.006

▾ patchMicrostrip - Conductor - metal

Catalog Gold

Name Gold

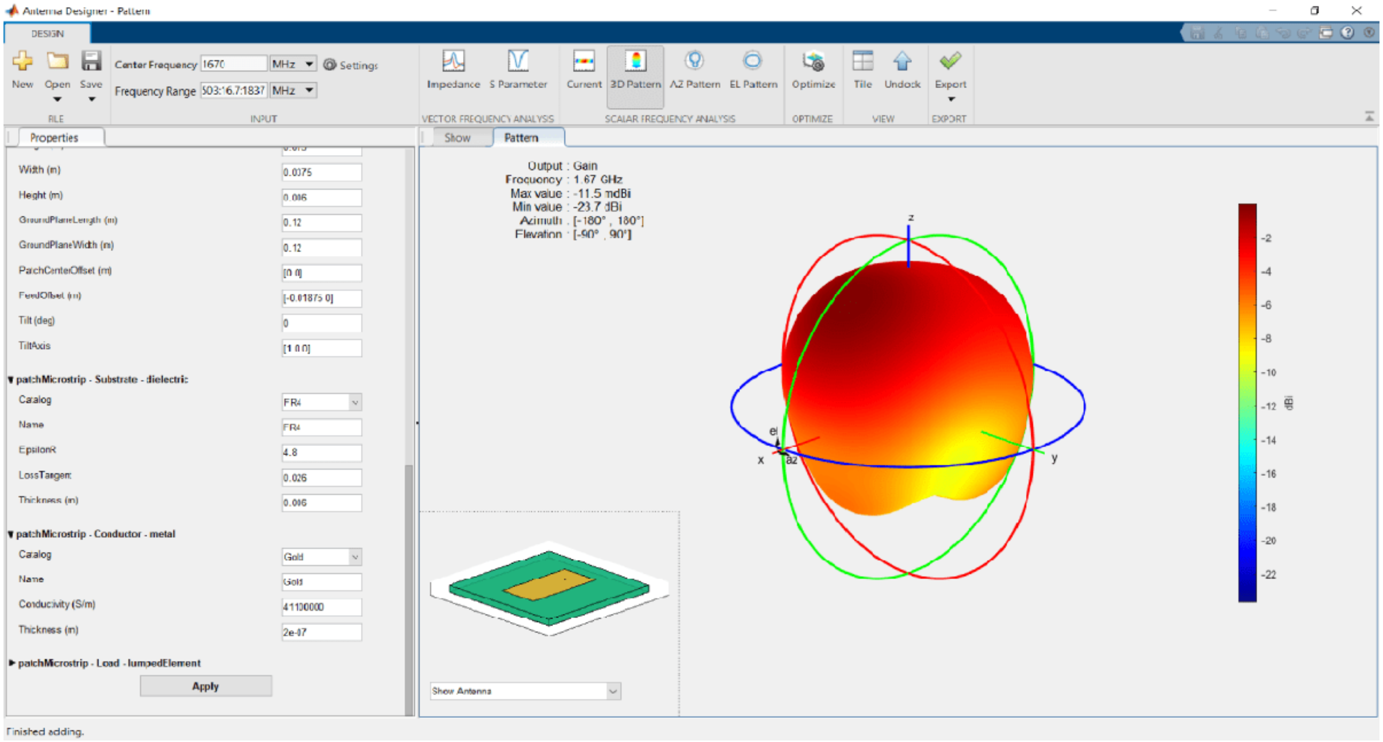
Conductivity (S/m) 41100000

Thickness (m) 2e-07

▸ patchMicrostrip - Load - lumpedElement

Finished Updating

Click **3D Pattern** to plot the radiation pattern of the antenna at the default frequency of 1.67 GHz.



Center Frequency 1670 MHz

Frequency Range 503:16.7:1837 MHz

Width (m) 0.0375

Height (m) 0.006

GroundPlaneLength (m) 0.12

GroundPlaneWidth (m) 0.12

PatchCenterOffset (m) [0 0]

FeedOffset (m) [-0.01875 0]

Tilt (deg) 0

TiltAxis [1 0 0]

▾ patchMicrostrip - Substrate - dielectric

Catalog FR4

Name FR4

EpsilonR 4.8

LossTangent 0.026

Thickness (m) 0.006

▾ patchMicrostrip - Conductor - metal

Catalog Gold

Name Gold

Conductivity (S/m) 41100000

Thickness (m) 2e-07

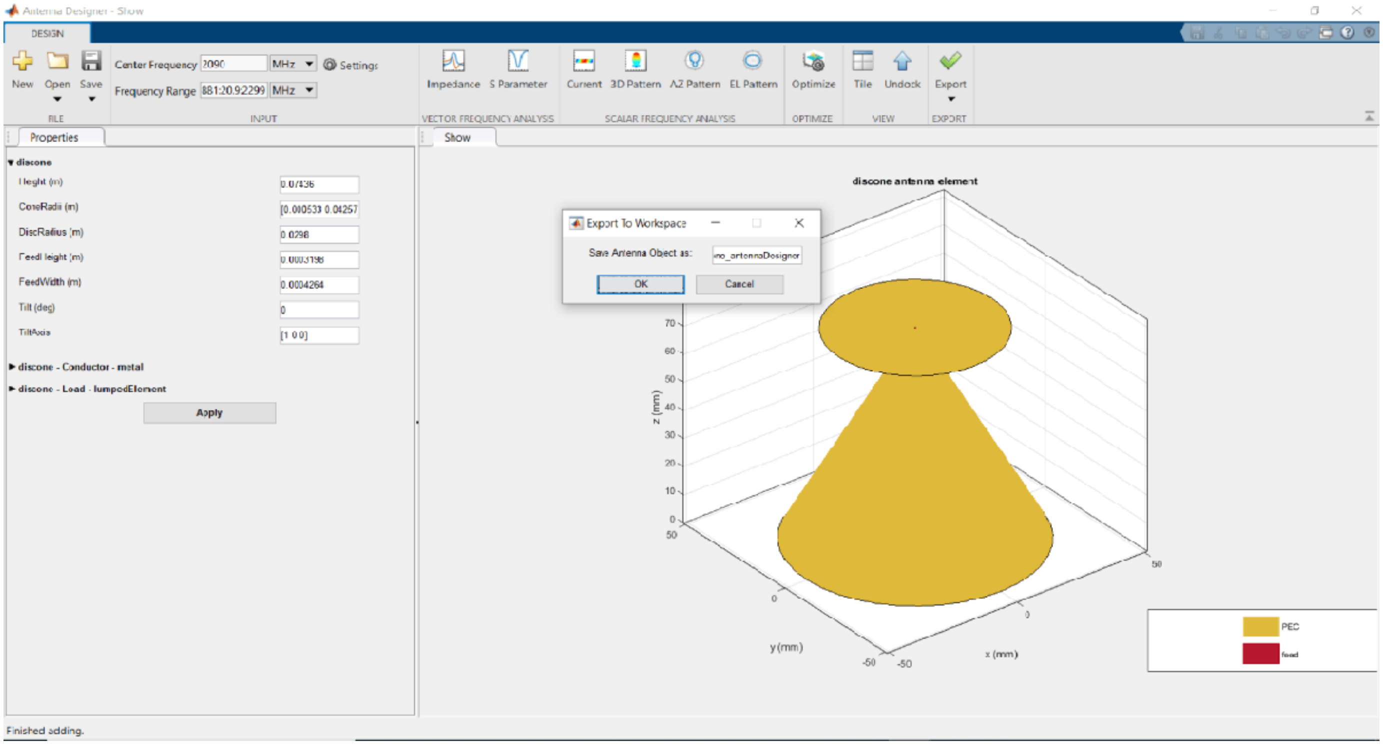
▸ patchMicrostrip - Load - lumpedElement

```
   Output : Gain
Frequency : 1.67 GHz
Max value : -11.5 mdBi
Min value : -23.7 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Finished adding.

Export, Save, Load and Analyze Discone Antenna

Create and export a discone antenna using Antenna Designer app.



Antenna Designer - Show

Center Frequency 2090 MHz

Frequency Range 881:20.92299 MHz

discone

Height (m) 0.07436

ConeRadii (m) [0.000533 0.04257

DiscRadius (m) 0.0298

FeedHeight (m) 0.0003198

FeedWidth (m) 0.0004264

Tilt (deg) 0

TiltAxis [1 0 0]

Export To Workspace

Save Antenna Object as: no_antennaDesigner

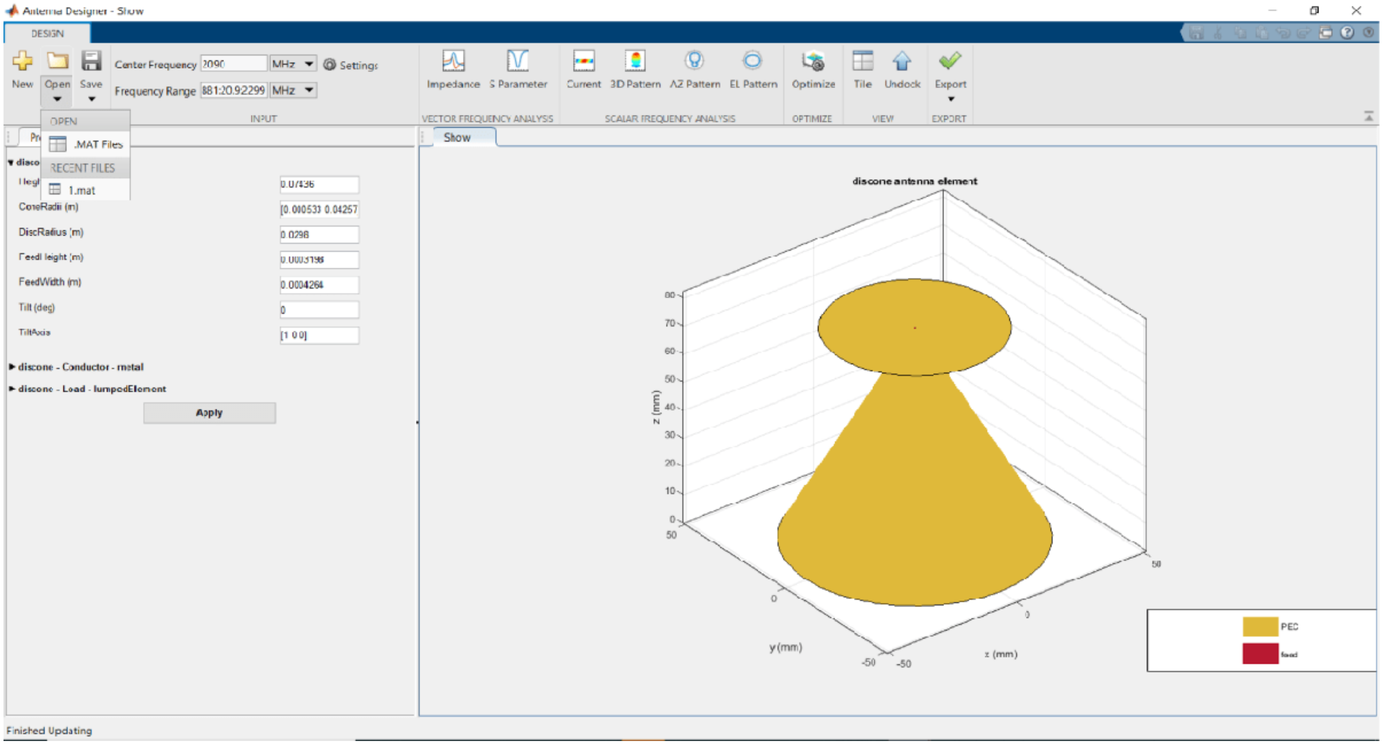
Finished adding.

In the MATLAB workspace, you will see the exported antenna. This is in the form of a .mat file.

Change the parameters of the antenna to the below given values at the MATLAB command line and save the .mat file again to a known folder.

```
Rd=55e-3;                  % Radius of disc
Rc1=72.1e-3;               % Broad Radius of cone
Rc2=1.875e-3;              % Narrow Radius of cone
Hc=160e-3;                 % Vertical height of cone
Fw=1e-3;                   % Feed Width
S=1.75e-3;                 % Spacing between cone and disc
```

Open the updated .mat file of the discone antenna using the open antenna designer app.



Antenna Designer - Show

Center Frequency 2090 MHz

Frequency Range 881:20.92299 MHz

Heigl 0.07436

CoreRadii (m) [0.000533 0.04257

DiscRadius (m) 0.0298

FeedHeight (m) 0.0003198

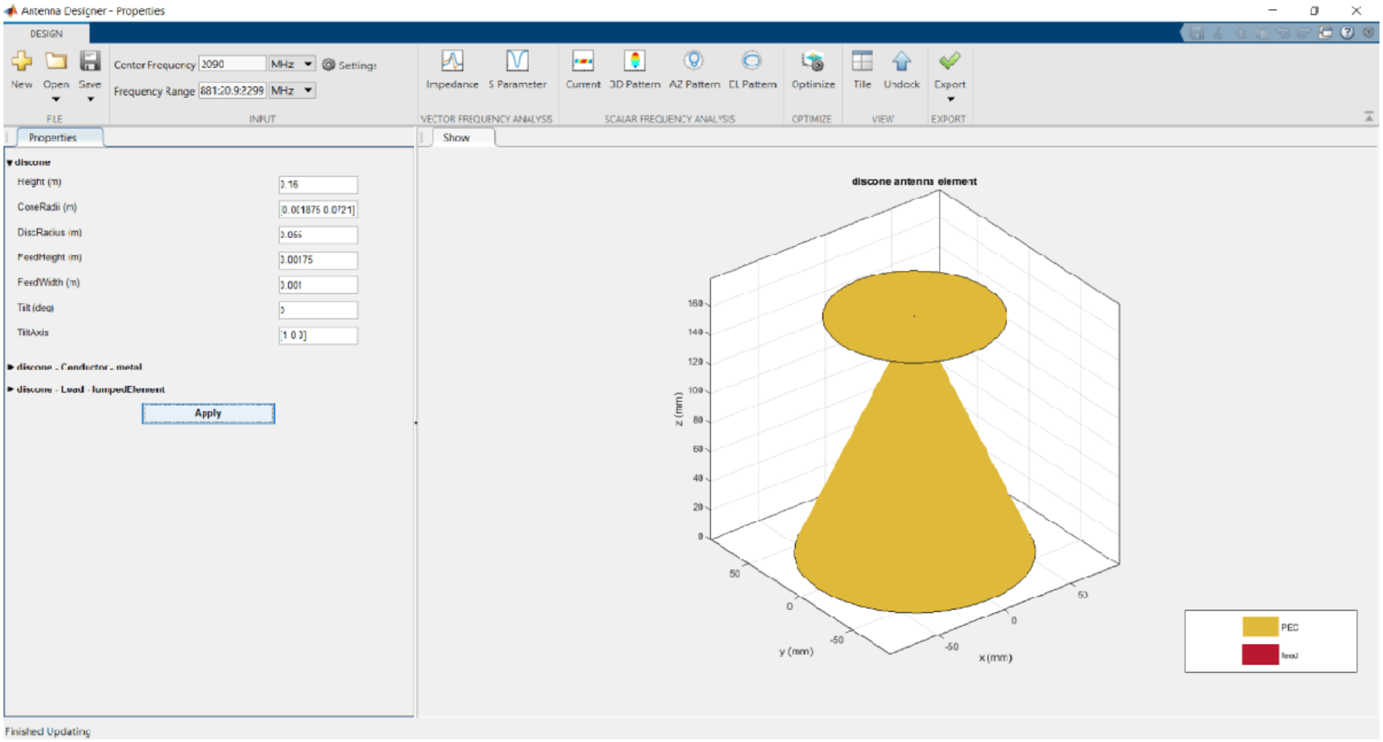
FeedWidth (m) 0.0004264

Tilt (deg) 0

TiltAxis [1 0 0]

Finished Updating

The app will overwrite the previous discone antenna design and open the updated discone antenna.



Antenna Designer - Properties

Center Frequency 2090 MHz

Frequency Range 881:20.9:2299 MHz

discone

Height (m) 0.16

CoreRadii (m) [0.001875 0.0721]

DiscRadius (m) 0.066

FeedHeight (m) 0.00175

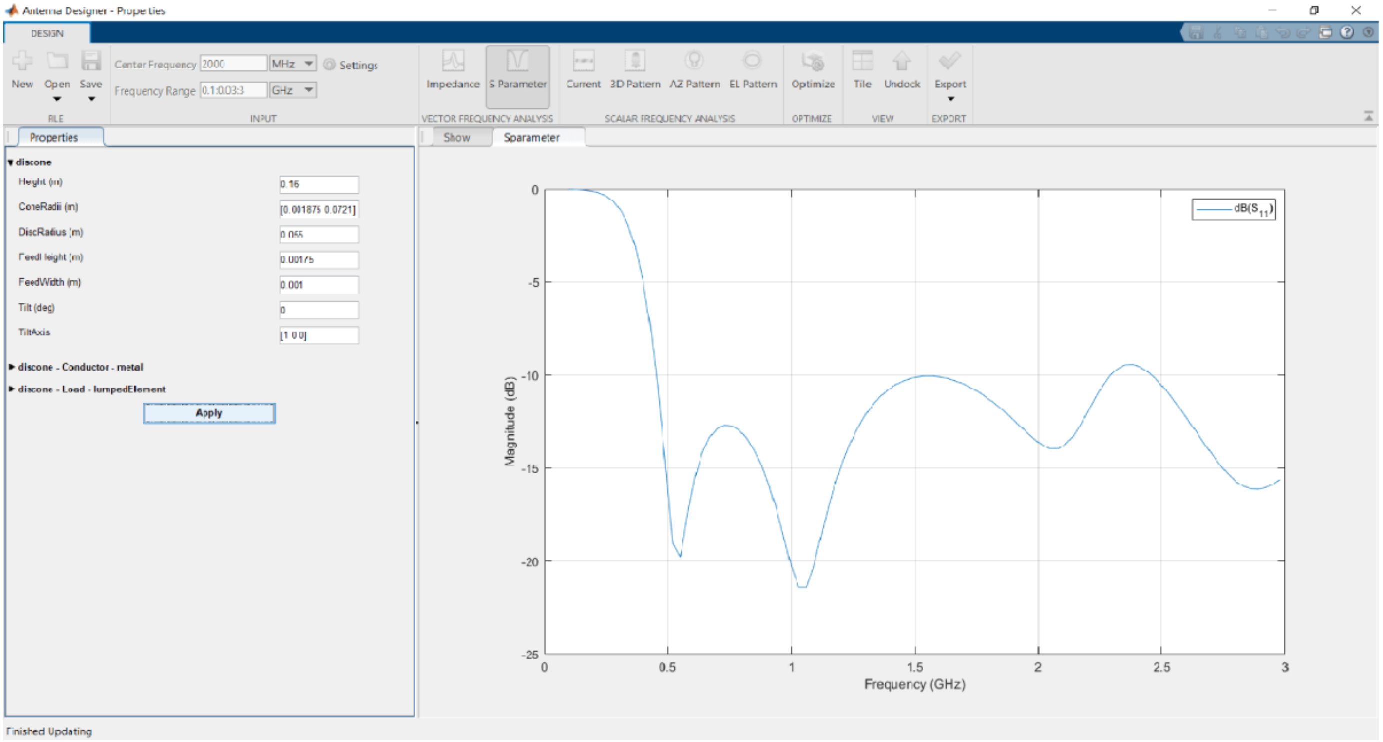
FeedWidth (m) 0.001

Tilt (deg) 0

TiltAxis [1 0 0]

Finished Updating

Calculate the S-parameter of the antenna at the specified frequency range.



Center Frequency 2000 MHz

Frequency Range 0.1:0.03:3 GHz

▾ discone

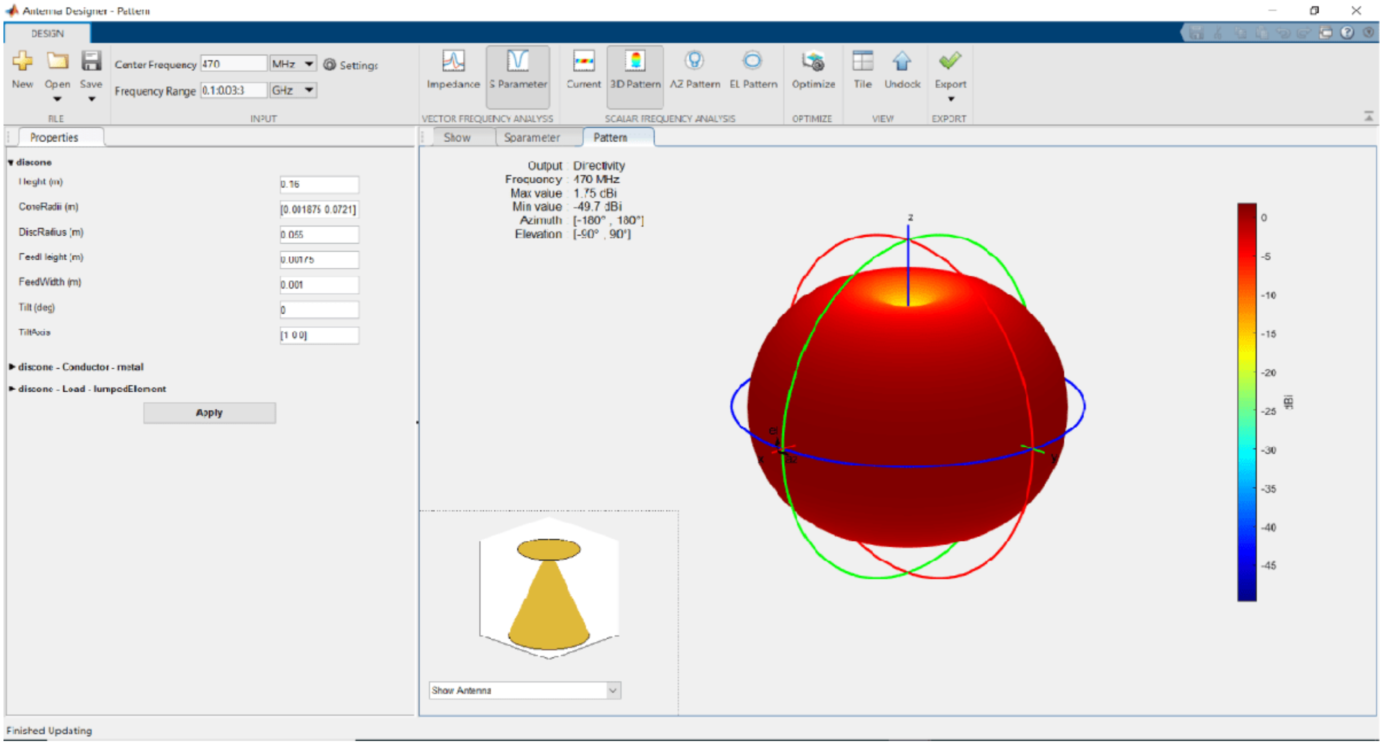
Height (m)	0.16
ConeRadii (m)	[0.001875 0.0721]
DiscRadius (m)	0.055
FeedHeight (m)	0.00175
FeedWidth (m)	0.001
Tilt (deg)	0
TiltAxis	[1 0 0]

▸ discone - Conductor - metal

▸ discone - Load - lumpedElement

Finished Updating

Plot the radiation pattern of the antenna at the specified frequency.



Center Frequency 470 MHz

Frequency Range 0.1:0.03:3 GHz

▾ discone

Height (m)	0.16
ConeRadii (m)	[0.001875 0.0721]
DiscRadius (m)	0.055
FeedHeight (m)	0.00175
FeedWidth (m)	0.001
Tilt (deg)	0
TiltAxis	[1 0 0]

▸ discone - Conductor - metal

▸ discone - Load - lumpedElement

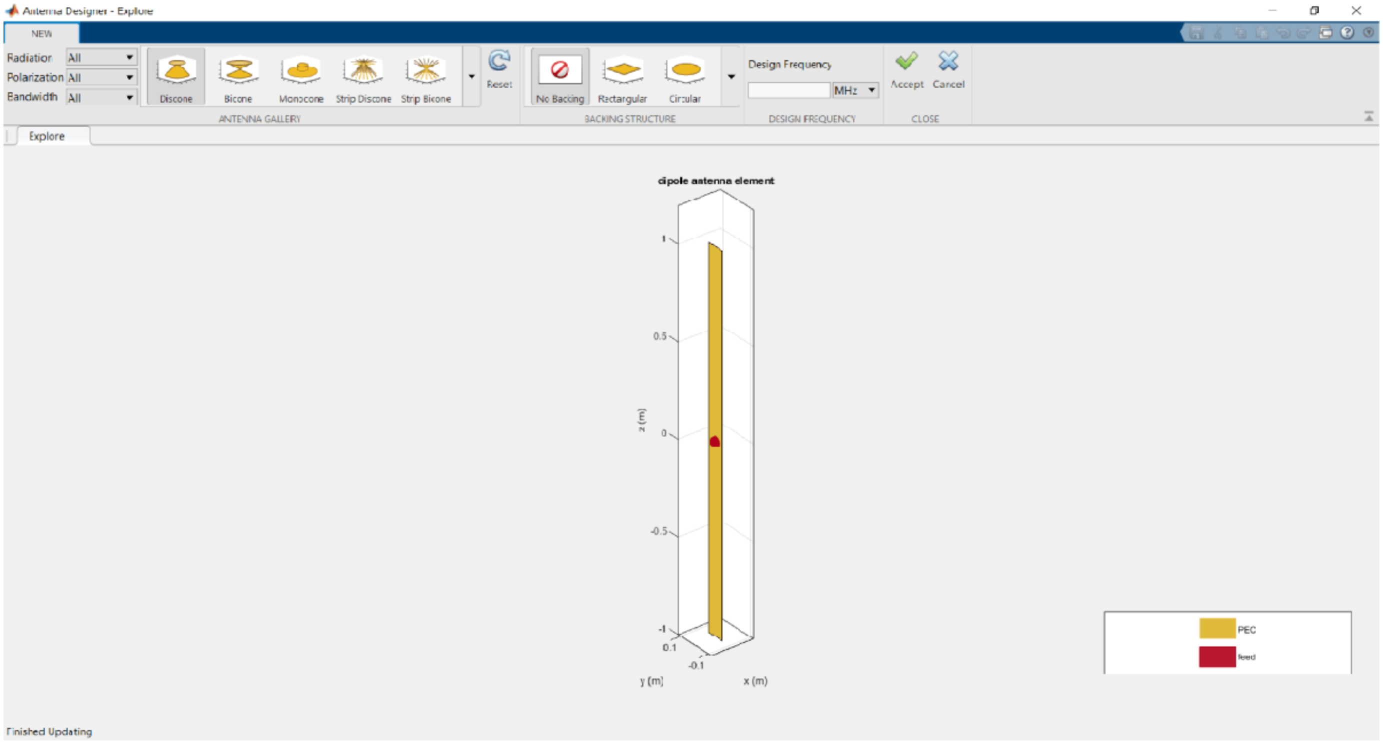
```
   Output : Directivity
Frequency : 470 MHz
Max value : 1.75 dBi
Min value : -49.7 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Finished Updating

Minimize Area of Dipole Antenna to Optimize Gain

Minimize the occupied area of a dipole antenna such that gain of the antenna is greater than 4 dBi.

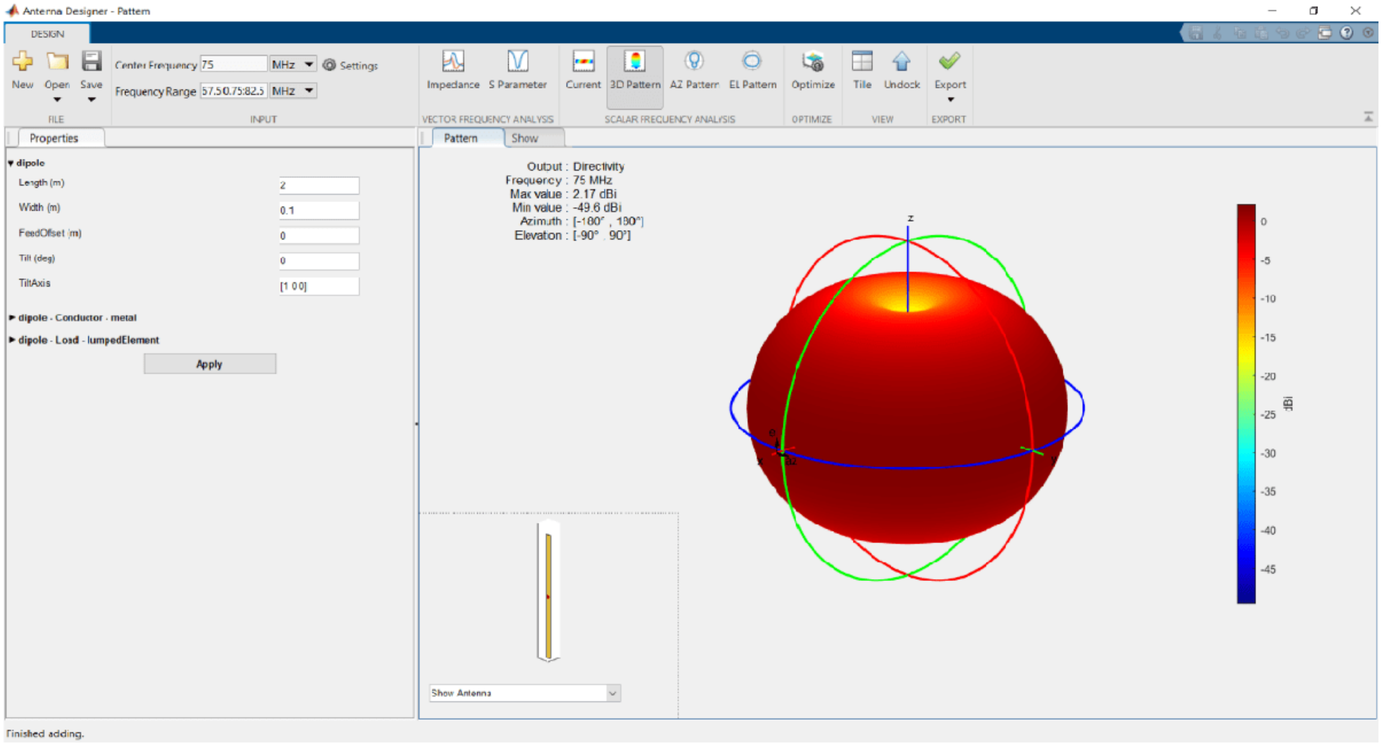
Open Antenna Designer app and accept the default dipole antenna.



Radiation All
Polarization All
Bandwidth All

Finished Updating

Analyze the pattern of the antenna. Notice that the Max value for directivity in the plot is 2.17 dBi.



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

dipole

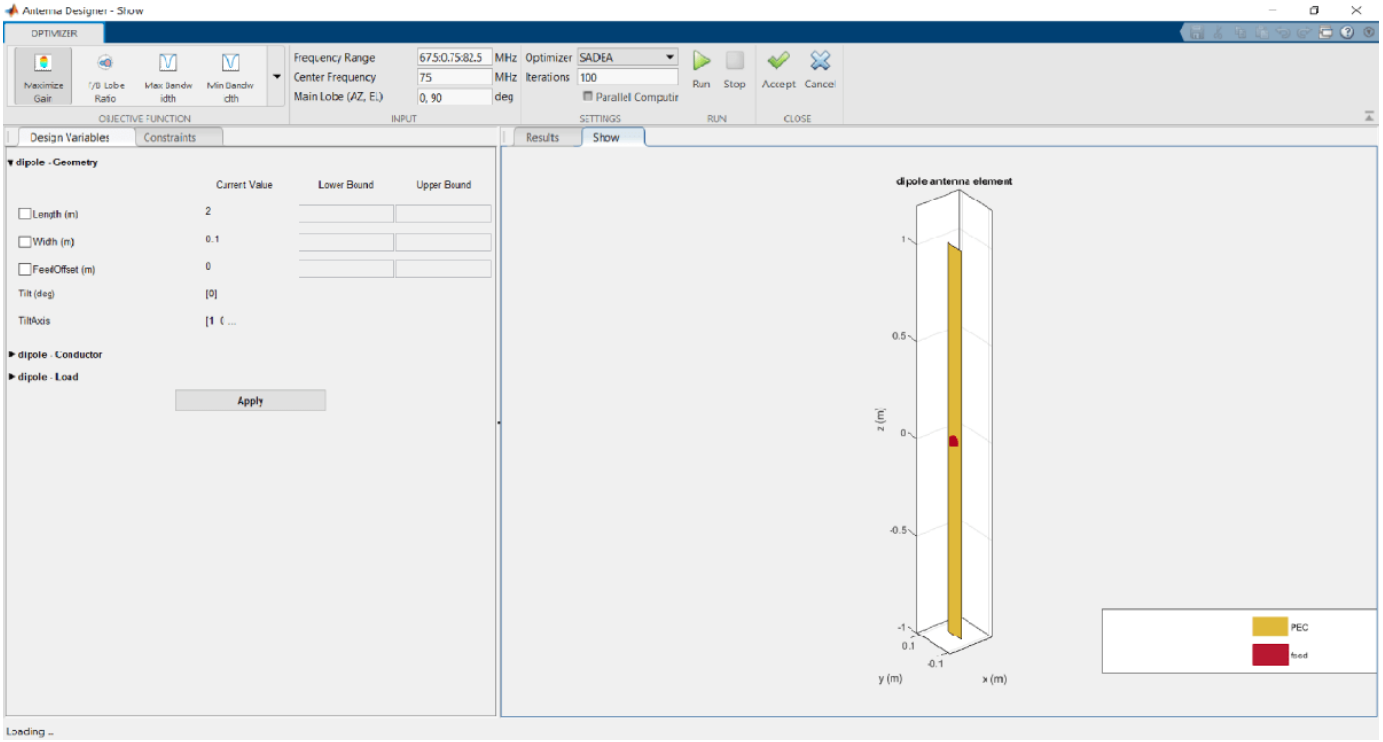
Length (m)	2
Width (m)	0.1
FeedOffset (m)	0
Tilt (deg)	0
TiltAxis	[1 0 0]

```
   Output : Directivity
Frequency : 75 MHz
Max value : 2.17 dBi
Min value : -49.6 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Finished adding.

Optimize Dipole Antenna

Click on **Optimize** to open the `Optimizer` canvas of the Antenna Designer app.



Frequency Range 67.5:0.75:82.5 MHz

Center Frequency 75 MHz

Main Lobe (AZ, EL) 0, 90 deg

Optimizer SADEA

Iterations 100

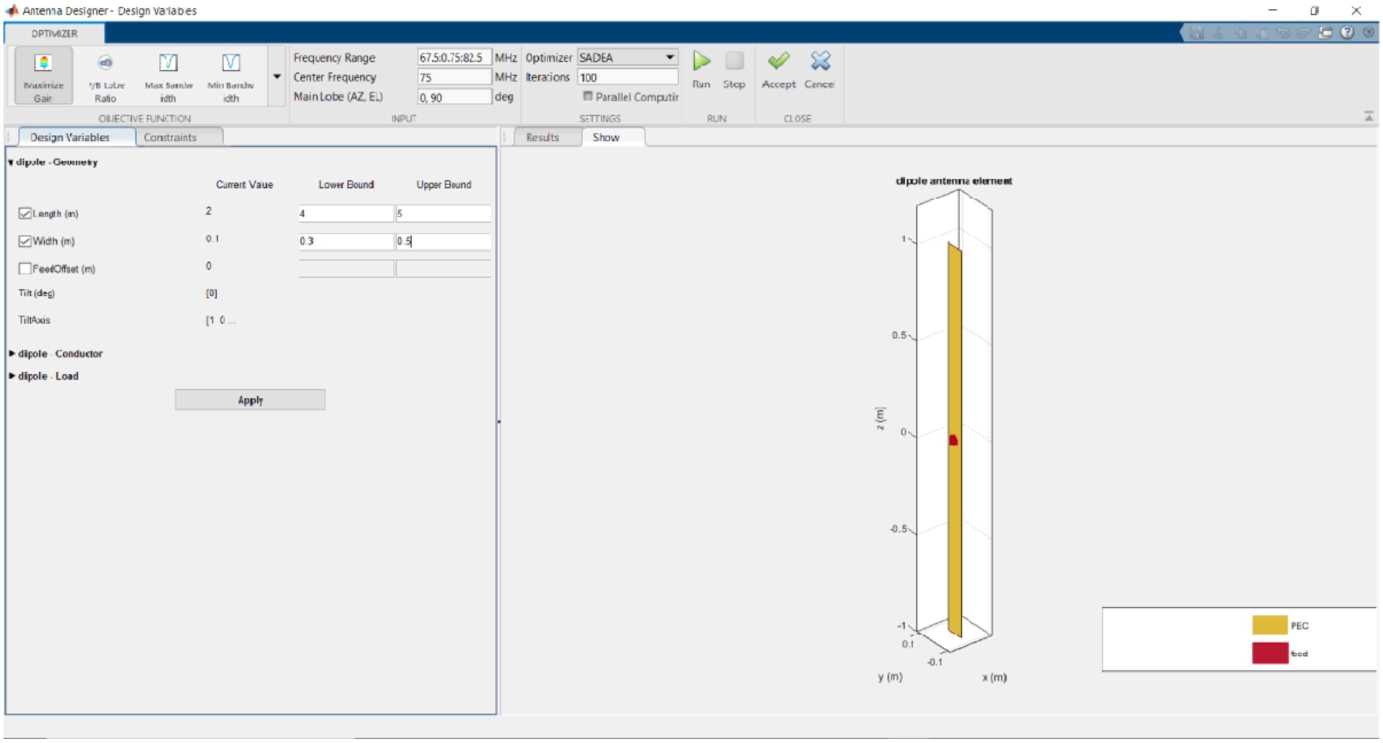
□ Parallel Computing

dipole - Geometry

	Current Value	Lower Bound	Upper Bound
□ Length (m)	2		
□ Width (m)	0.1		
□ FeedOffset (m)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		

Loading ...

From the **OBJECTIVE FUNCTION** drop down choose, **Minimize Area**. Enter the bounds for the length and the width of the antenna in the `Design Variables` tab. Click **Apply.**



Antenna Designer - Design Variables

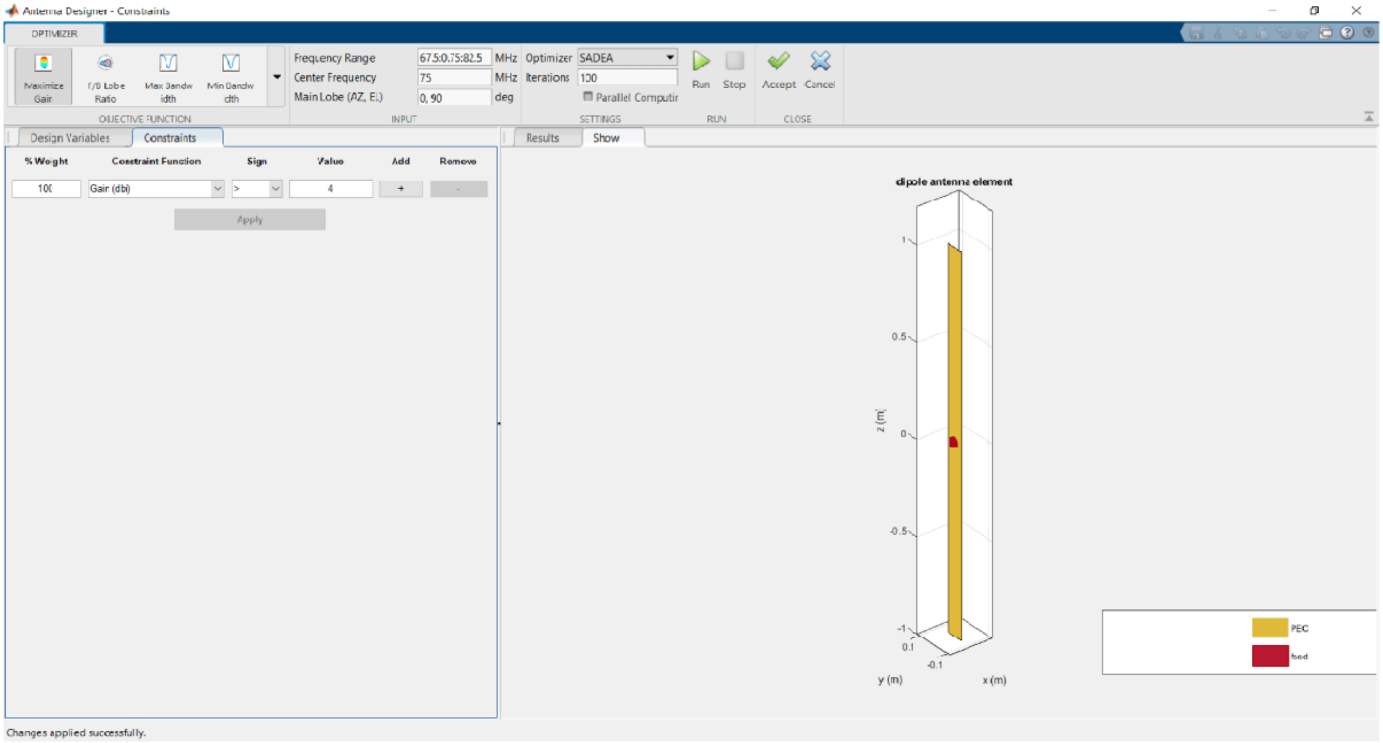
Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 100
□ Parallel Computir

▾ dipole - Geometry

	Current Vaue	Lower Bound	Upper Bound
☑ Length (m)	2	4	5
☑ Width (m)	0.1	0.3	0.5
□ FeedOffset (m)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		

▸ dipole - Conductor
▸ dipole - Load

Enter the constraints in the `Constraints` tab. Click **Apply**.



Antenna Designer - Constraints

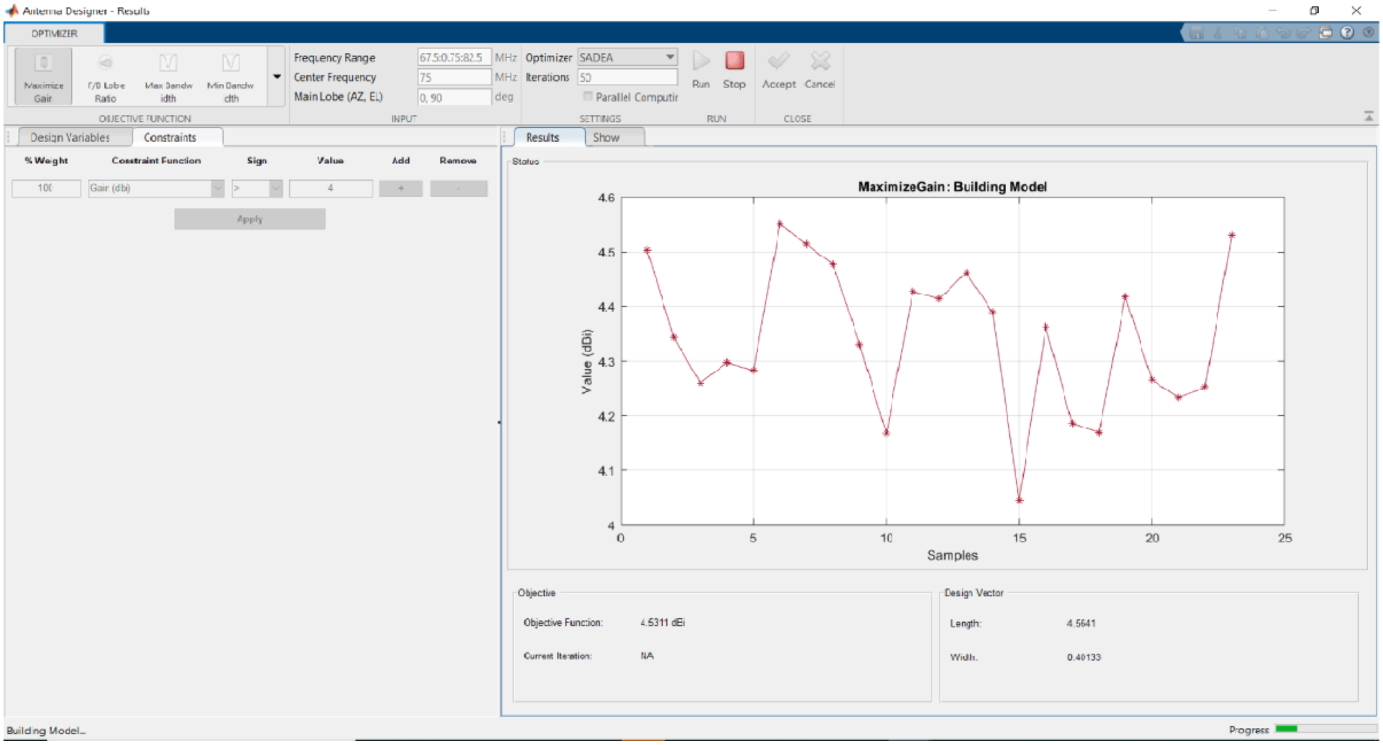
Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 100
□ Parallel Computir

% Weight	Constraint Function	Sign	Value	Add	Remove
100	Gain (dbi)	>	4	+	-

Changes applied successfully.

Set the number of iterations to 50. Click **Run**.

First the optimizer builds the model.



Frequency Range 67.5:0.75:82.5 MHz

Center Frequency 75 MHz

Main Lobe (AZ, EL) 0, 90 deg

Optimizer SADEA

Iterations 50

□ Parallel Computir

% Weight	Constraint Function	Sign	Value	Add	Remove
10(	Gain (dbi)	>	4	+	-

Apply

Objective Function: 4.5311 dBi

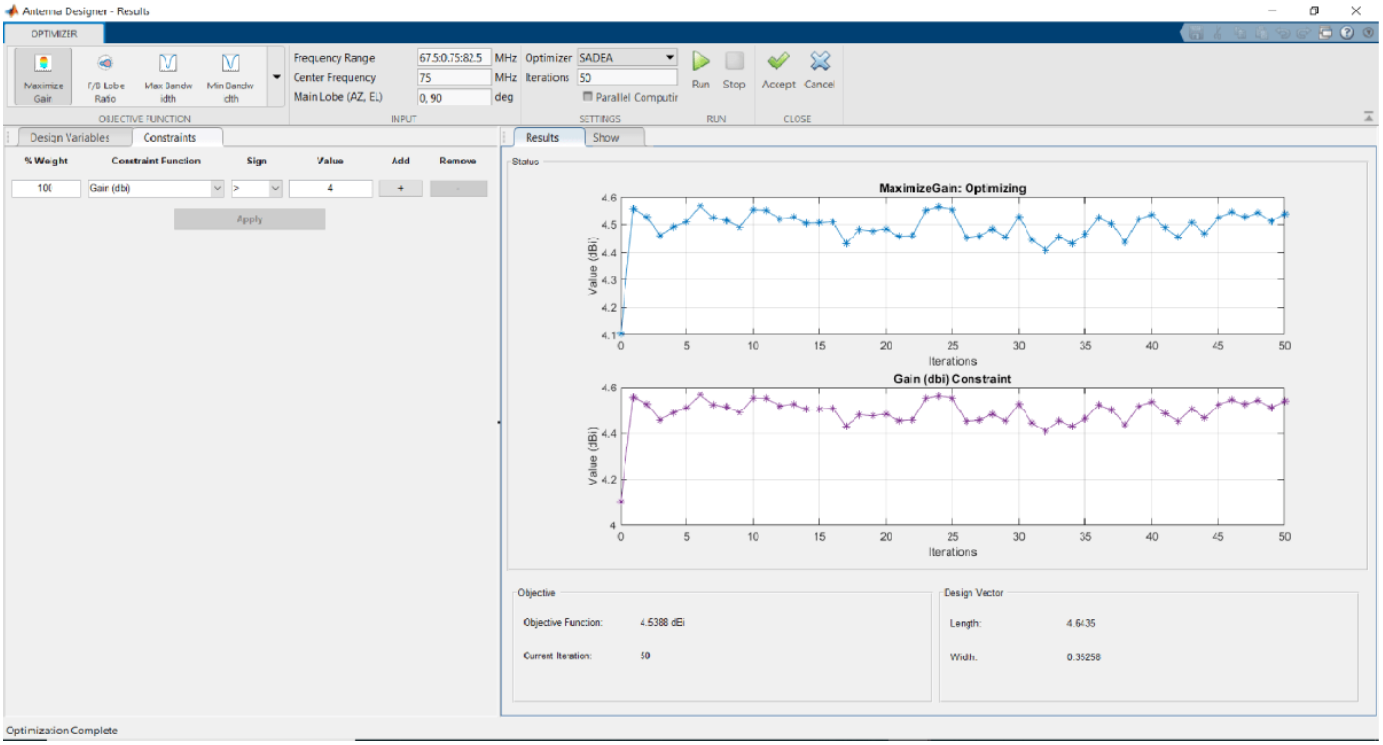
Current Iteration: NA

Length: 4.5641

Width: 0.40133

Building Model...

Then starts the optimization based on the objective function and the constraints.



Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 50
□ Parallel Computir

% Weight	Constraint Function	Sign	Value	Add	Remove
100	Gain (dbi)	>	4	+	-

Objective

Objective Function: 4.5388 dBi

Current Iteration: 50

Design Vector

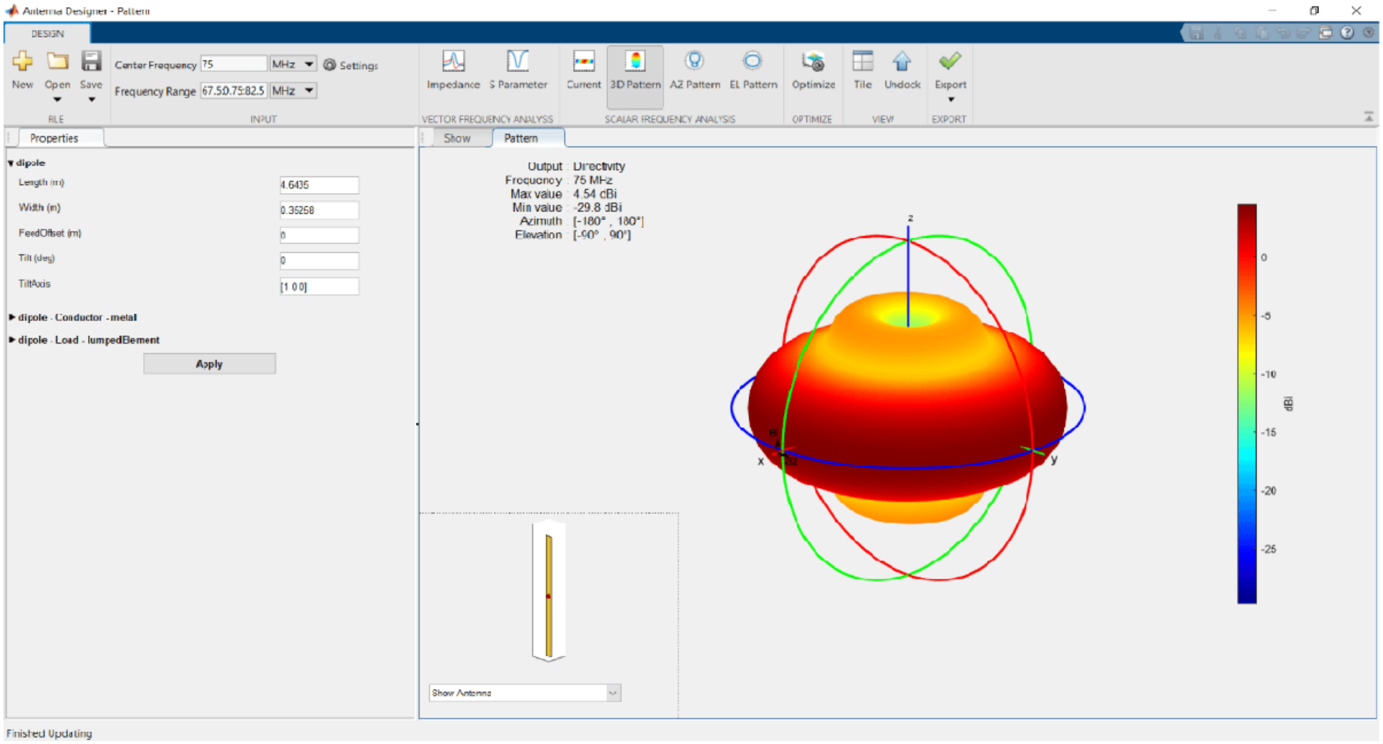
Length: 4.6435

Width: 0.35258

Optimization Complete

Click **Accept**.

Analyze the antenna again for the 3D pattern. See that the Max value of the directivity is now 4.54 dBi.



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

▾ dipole

Length (m)	4.6435
Width (m)	0.35258
FeedOffset (m)	0
Tilt (deg)	0
TiltAxis	[1 0 0]

▸ dipole - Conductor - metal

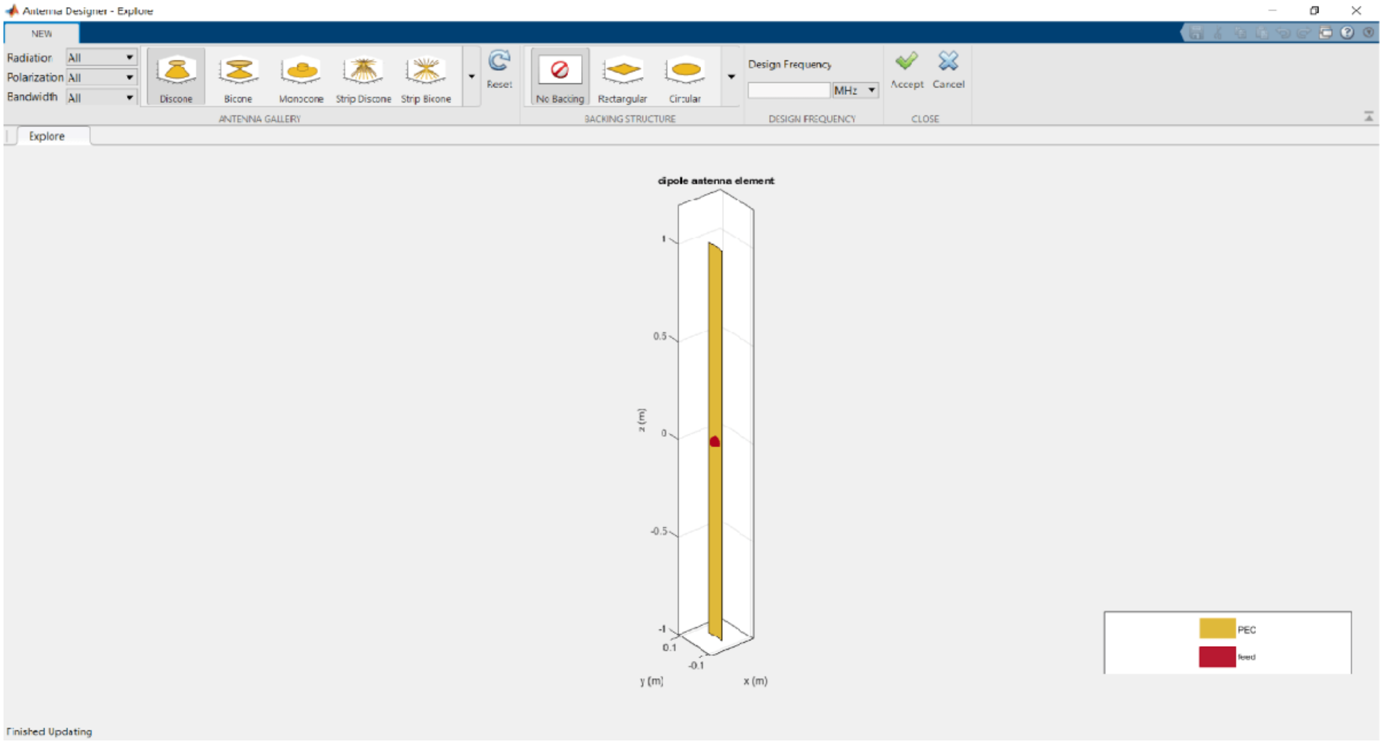
▸ dipole - Load - lumpedElement

```
   Output : Directivity
Frequency : 75 MHz
Max value : 4.54 dBi
Min value : -29.8 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Minimize Area of Dipole Antenna to Optimize Gain Using Surrogate Optimization Method

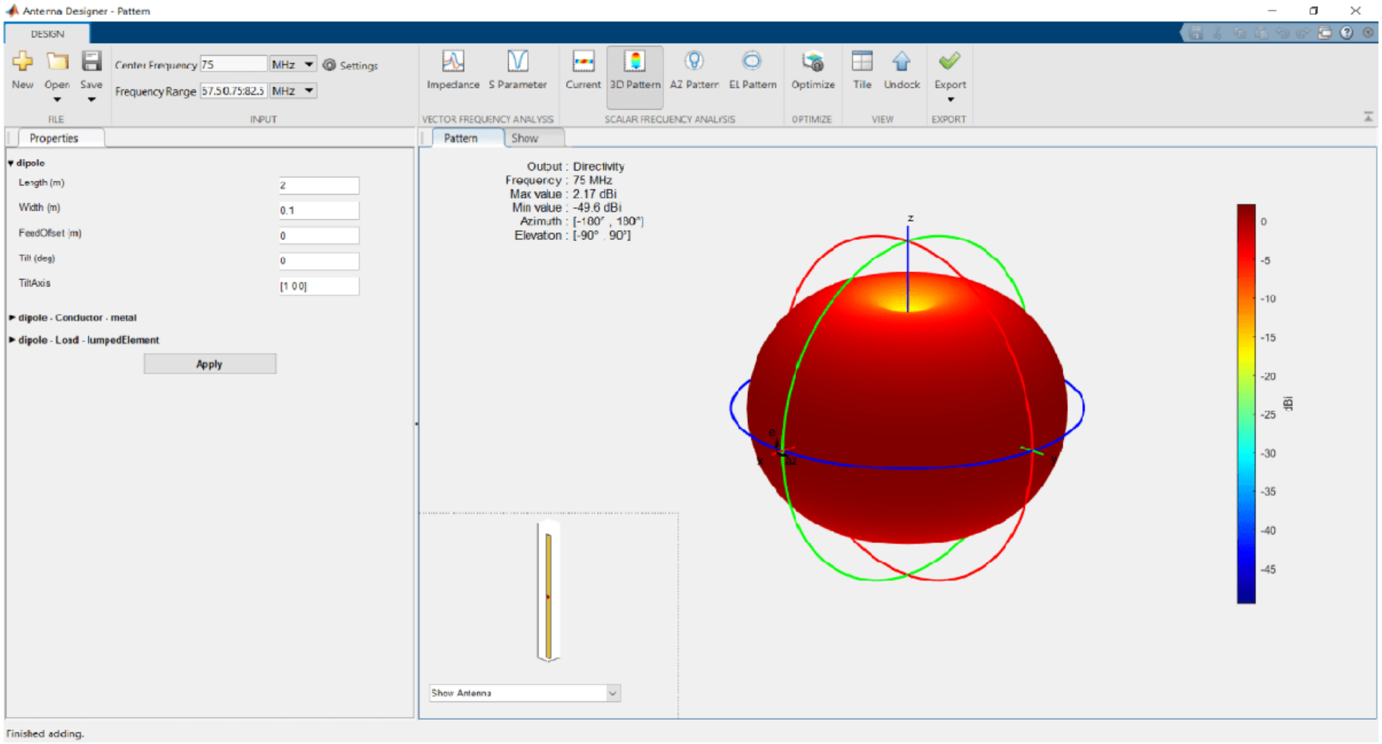
Minimize the occupied area of a dipole antenna such that gain of the antenna is greater than 4 dBi.

Open Antenna Designer app and accept the default dipole antenna.



Finished Updating

Analyze the pattern of the antenna. Notice that the Max value for directivity in the plot is 2.17 dBi.



Center Frequency 75 MHz
Frequency Range 57.5:0.75:82.5 MHz

▾ dipole

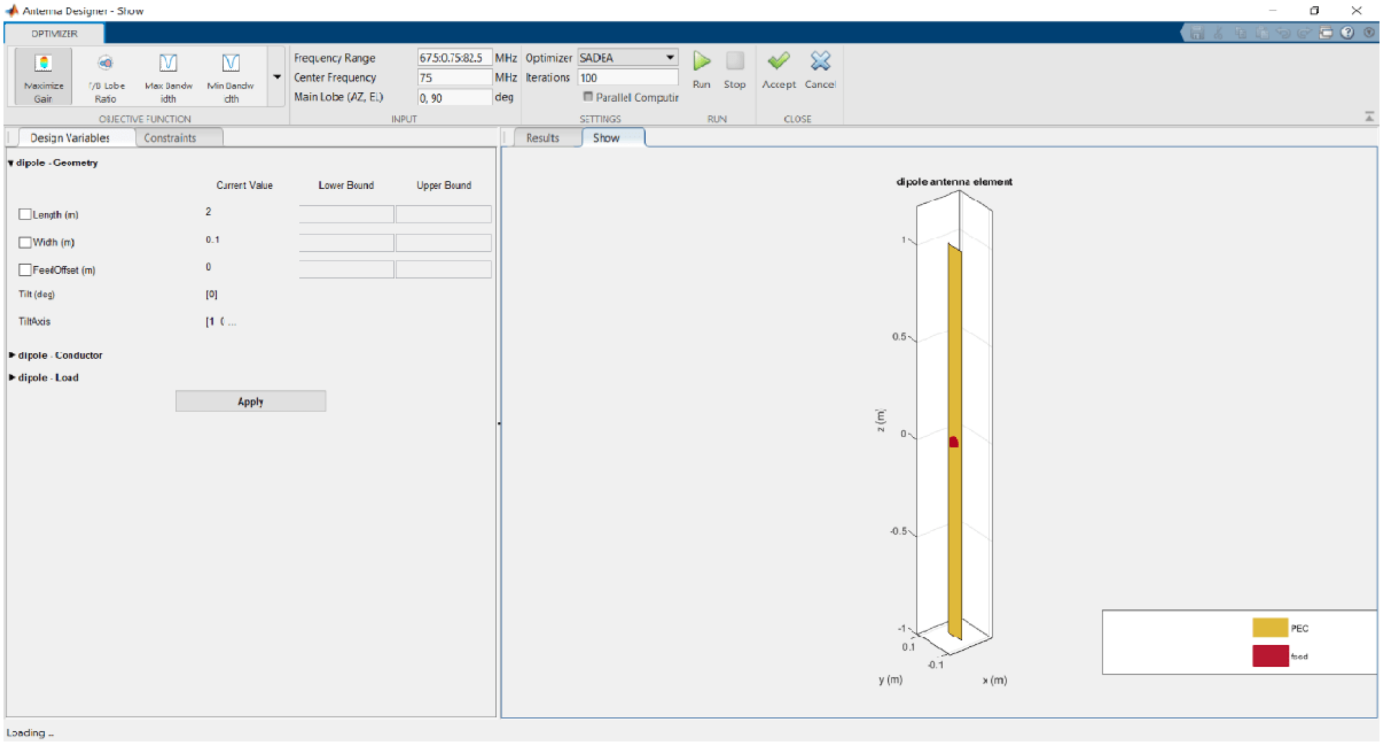
Length (m) 2
Width (m) 0.1
FeedOffset (m) 0
Tilt (deg) 0
TiltAxis [1 0 0]

```
Output : Directivity
Frequency : 75 MHz
Max value : 2.17 dBi
Min value : -49.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Finished adding.

Optimize Dipole Antenna

Click on **Optimize** to open the `Optimizer` canvas of the Antenna Designer app.



Frequency Range 675:0.75:82.5 MHz

Center Frequency 75 MHz

Main Lobe (AZ, EL) 0, 90 deg

Optimizer SADEA

Iterations 100

□ Parallel Computir

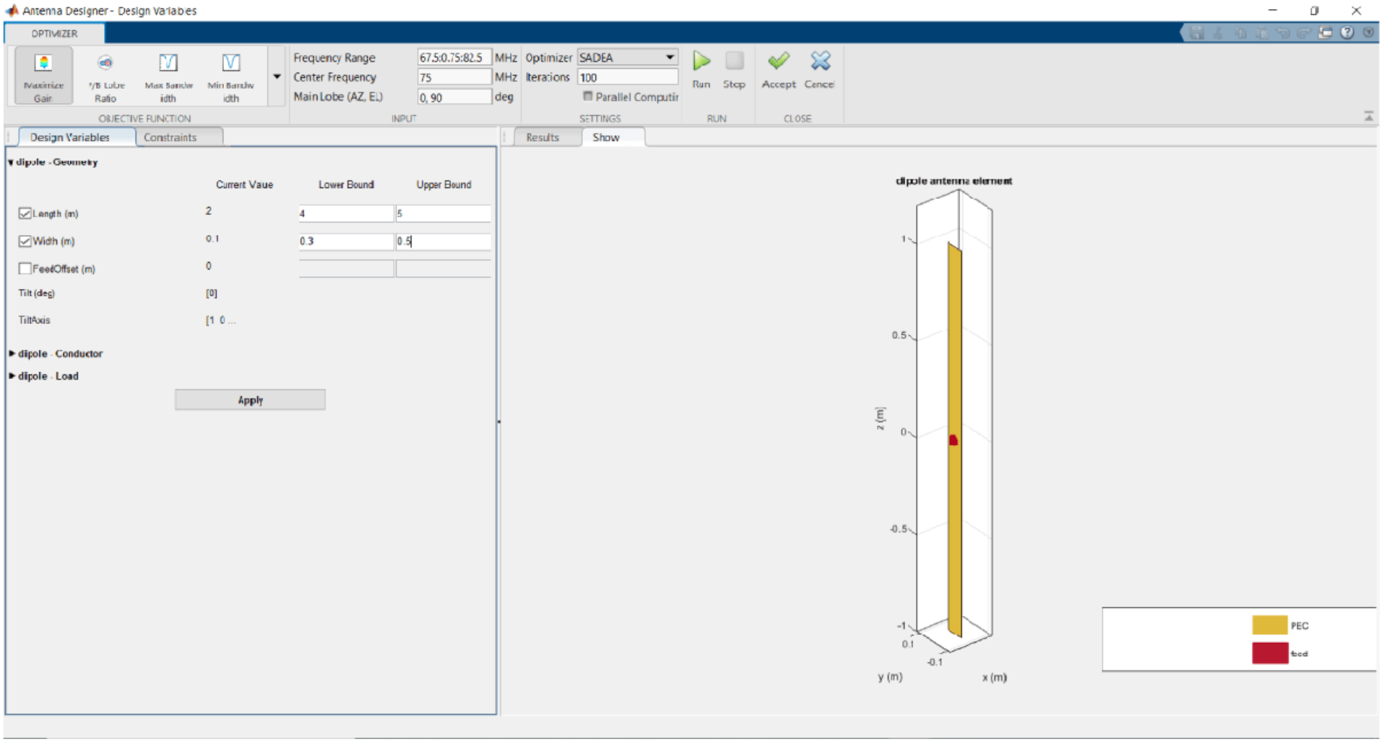
▾ dipole - Geometry

	Current Value	Lower Bound	Upper Bound
□ Length (m)	2		
□ Width (m)	0.1		
□ FeedOffset (m)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		

▸ dipole - Conductor

▸ dipole - Load

From the **OBJECTIVE FUNCTION** drop down choose, **Minimize Area**. Enter the bounds for the length and the width of the antenna in the `Design Variables` tab. Click **Apply.**


Antenna Designer - Design Variables

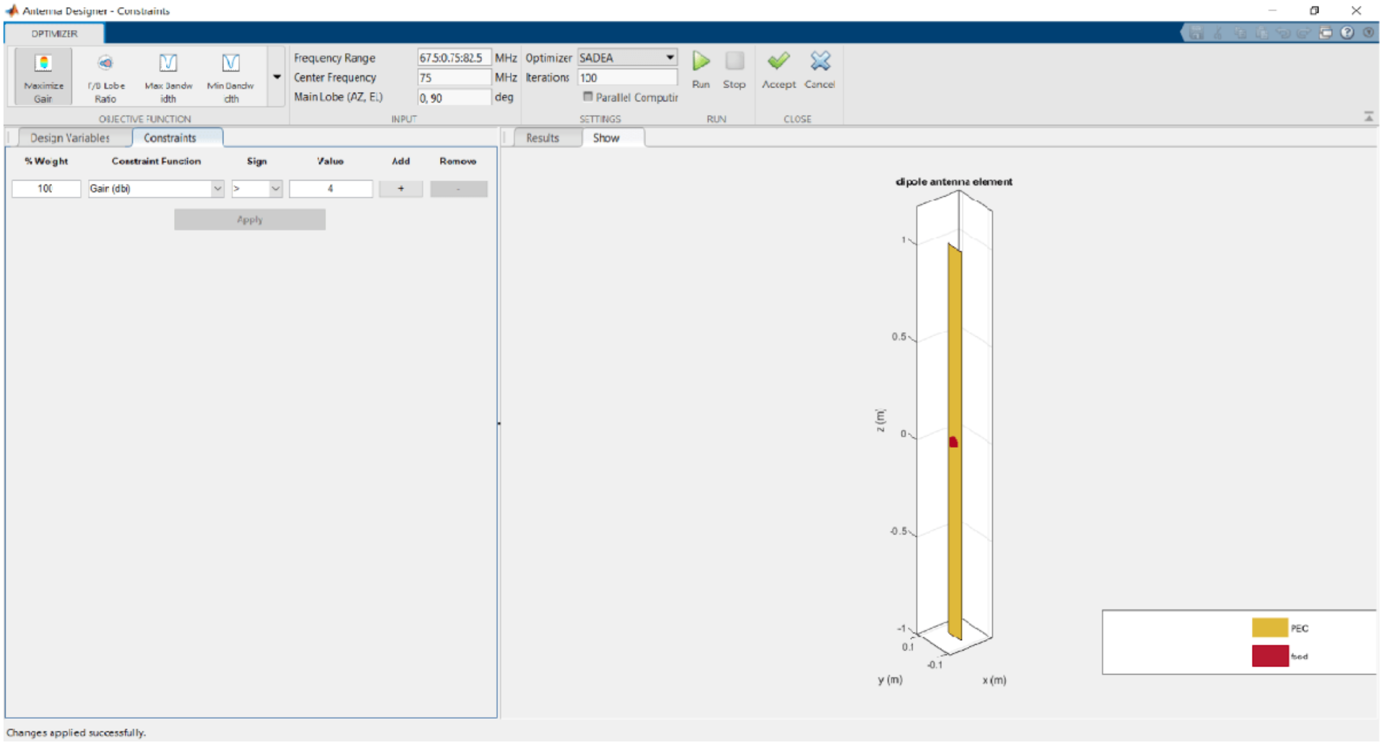
Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 100
□ Parallel Computir

▾ dipole - Geometry

	Current Value	Lower Bound	Upper Bound
☑ Length (m)	2	4	5
☑ Width (m)	0.1	0.3	0.5
□ FeedOffset (m)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		

▸ dipole - Conductor
▸ dipole - Load

Enter the constraints in the `Constraints` tab. Click **Apply**.

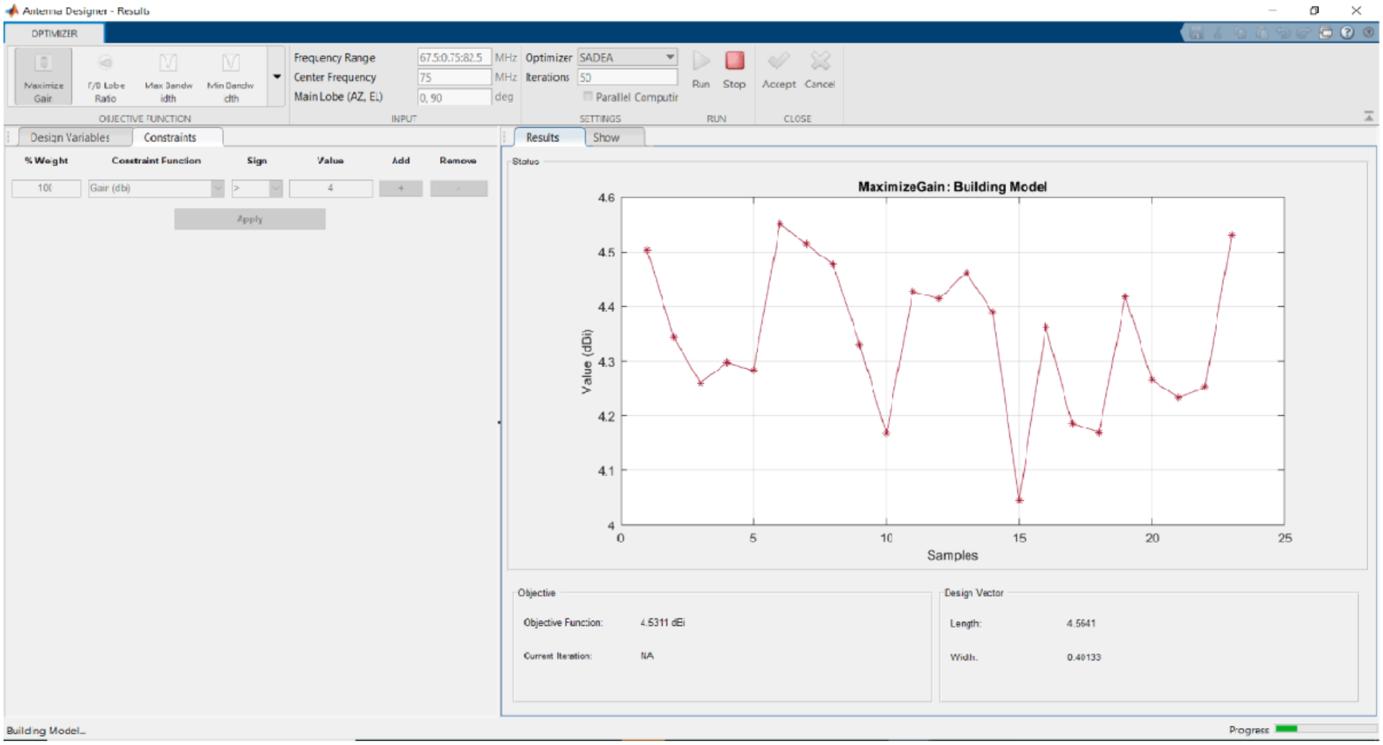

Antenna Designer - Constraints

Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 100
□ Parallel Computir

% Weight	Constraint Function	Sign	Value	Add	Remove
100	Gain (dbi)	>	4	+	-

Changes applied successfully.

Change the `Optimizer` from **SADEA** to **Surrogate Opt**. The number iterations is always 200 and **Parallel Computing** is greyed out.



Antenna Designer - Results

Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 50

% Weight	Constraint Function	Sign	Value
10(	Gain (dbi)	>	4

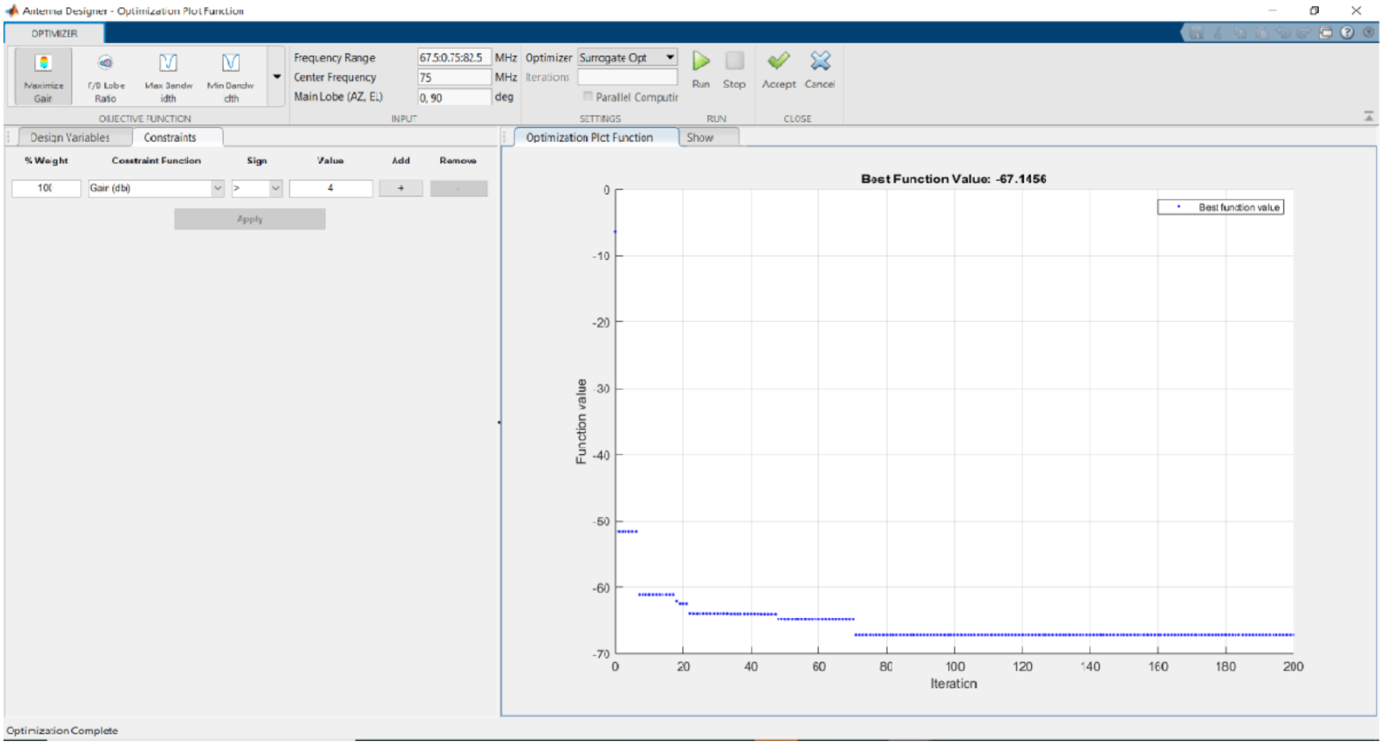
Objective
Objective Function: 4.5311 dBi
Current Iteration: NA

Design Vector
Length: 4.5641
Width: 0.40133

Building Model...

Click **Run**.

The optimization starts based on the objective function and the constraints.



Antenna Designer - Optimization Plot Function

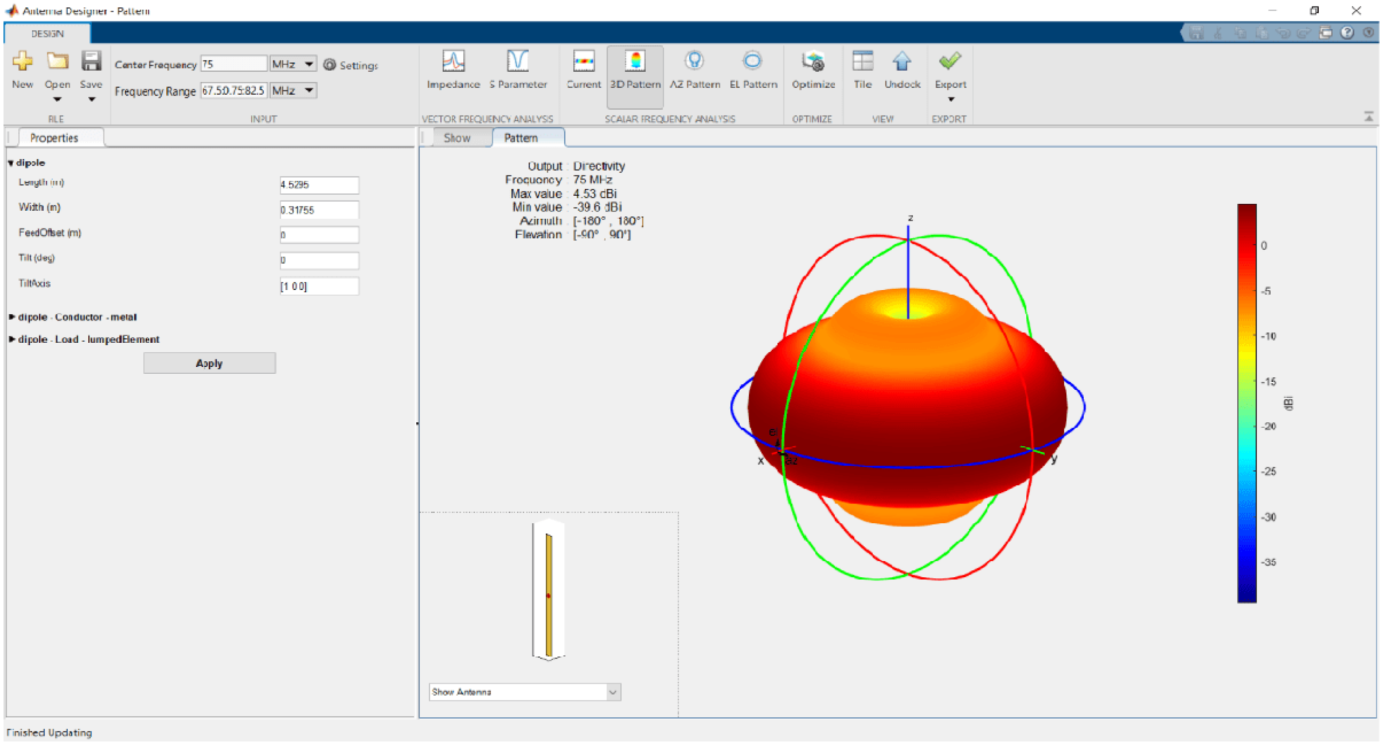
Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer Surrogate Opt

% Weight	Constraint Function	Sign	Value
10(	Gain (dbi)	>	4

Optimization Complete

Click **Accept**.

Analyze the antenna again for the 3D pattern. See that the Max value of the directivity is now 4.53 dBi.



Center Frequency 75 MHz
Frequency Range 67.5:0.75:82.5 MHz

▾ dipole

Length (m)	4.5295
Width (m)	0.31755
FeedOffset (m)	0
Tilt (deg)	0
TiltAxis	[1 0 0]

```
   Output : Directivity
Frequency : 75 MHz
Max value : 4.53 dBi
Min value : -39.6 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

- “Design and Analysis Using Antenna Designer”
- “Maximizing Gain and Improving Impedance Bandwidth of E-Patch Antenna”

Programmatic Use

`antennaDesigner` opens the **Antenna Designer** app, enabling you to design, analyze, and optimize antennas present in the Antenna Toolbox library.

Version History

Introduced in R2017a

See Also

Topics

“Design and Analysis Using Antenna Designer”
“Maximizing Gain and Improving Impedance Bandwidth of E-Patch Antenna”
“Antenna Optimization Algorithm”

Antenna Array Designer

Design, visualize, and analyze arrays

Description

The **Array Designer** app lets you design, visualize, and analyze arrays in the Antenna Toolbox library interactively.

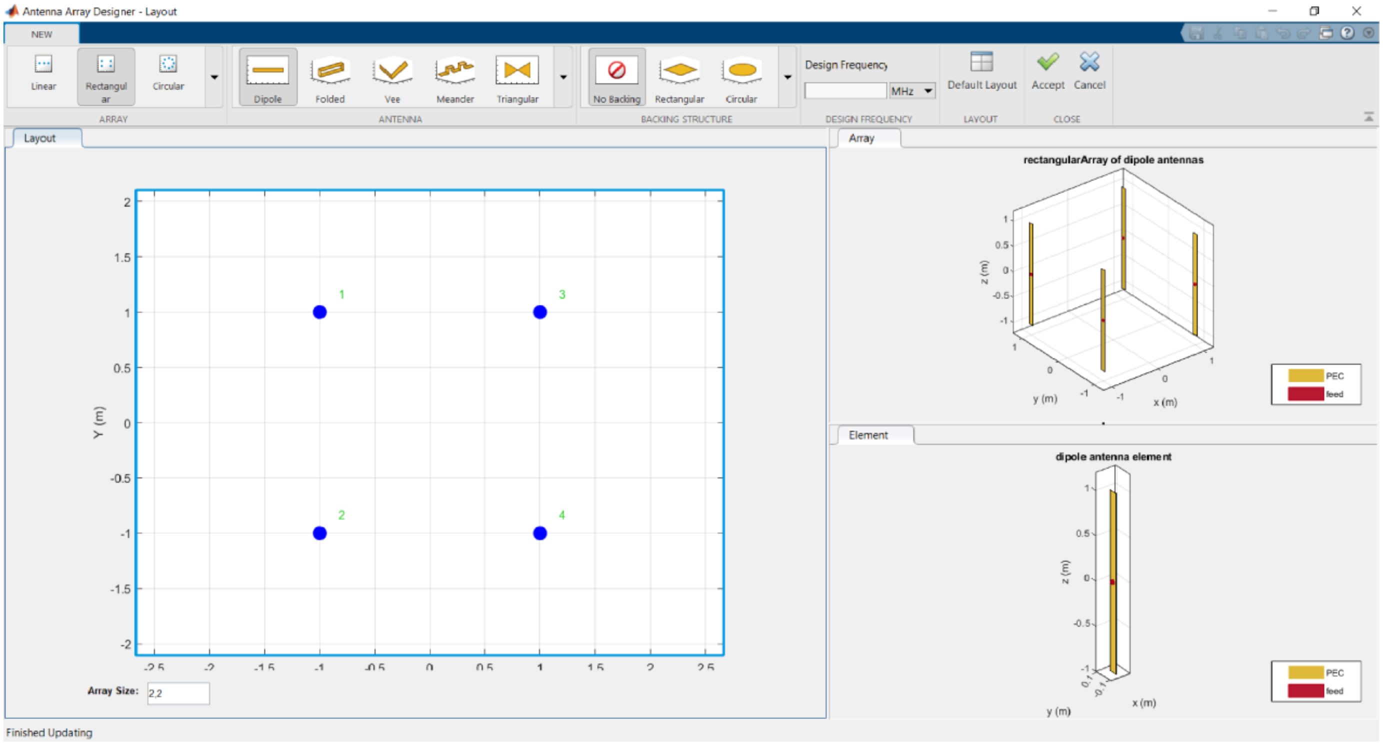
Using this app, you can:

- Show different array configurations and layouts defining element spacing.
- Compare different array types and responses.
- Pick array configuration to meet specific peek gain, directivity, desired coverage, pattern, port parameters.
- Change the spacing between the elements and see the effect on the performance of the array.
- Visualize the effect of mutual coupling at the port and in the far-field.
- Optimize arrays for various analysis results under given constraints using SADEA or Surrogate optimization methods.

Note

- To use **Parallel Computing** for SADEA optimizer, you need the Parallel Computing Toolbox.

To use the Surrogate optimization algorithm, you need the Global Optimization Toolbox.



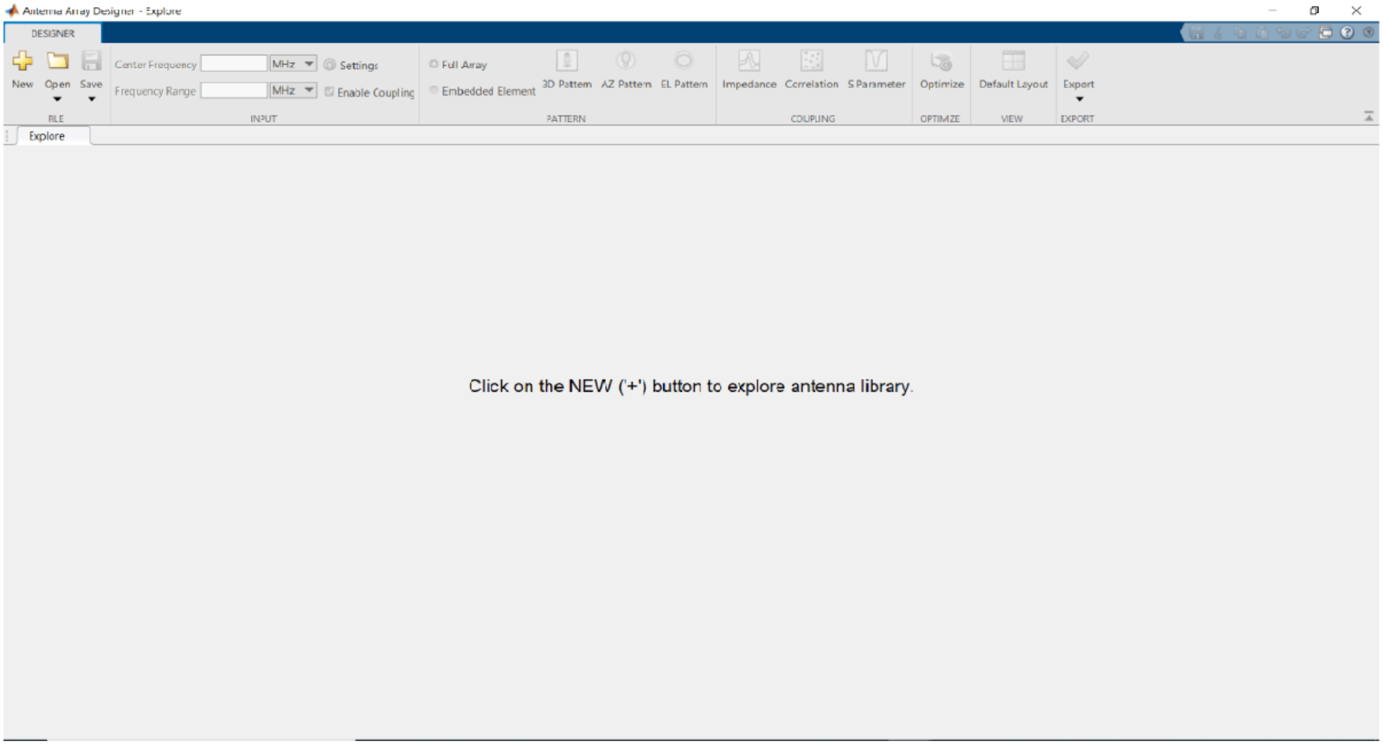
Open the Antenna Array Designer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `antennaArrayDesigner`.

Examples

Antenna Array Designer Canvas

The antenna array designer app opens a new blank canvas.

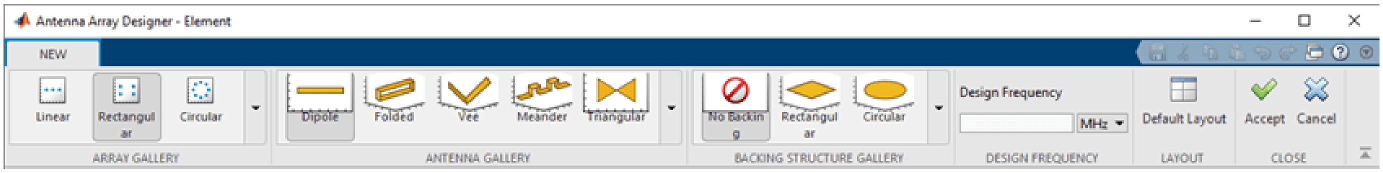


Click on the NEW ('+') button to explore antenna library.

Select and Visualize an Array

Click New in the canvas toolstrip to choose the type of array you want to analyze.

The default is a rectangular array with dipole antennas.



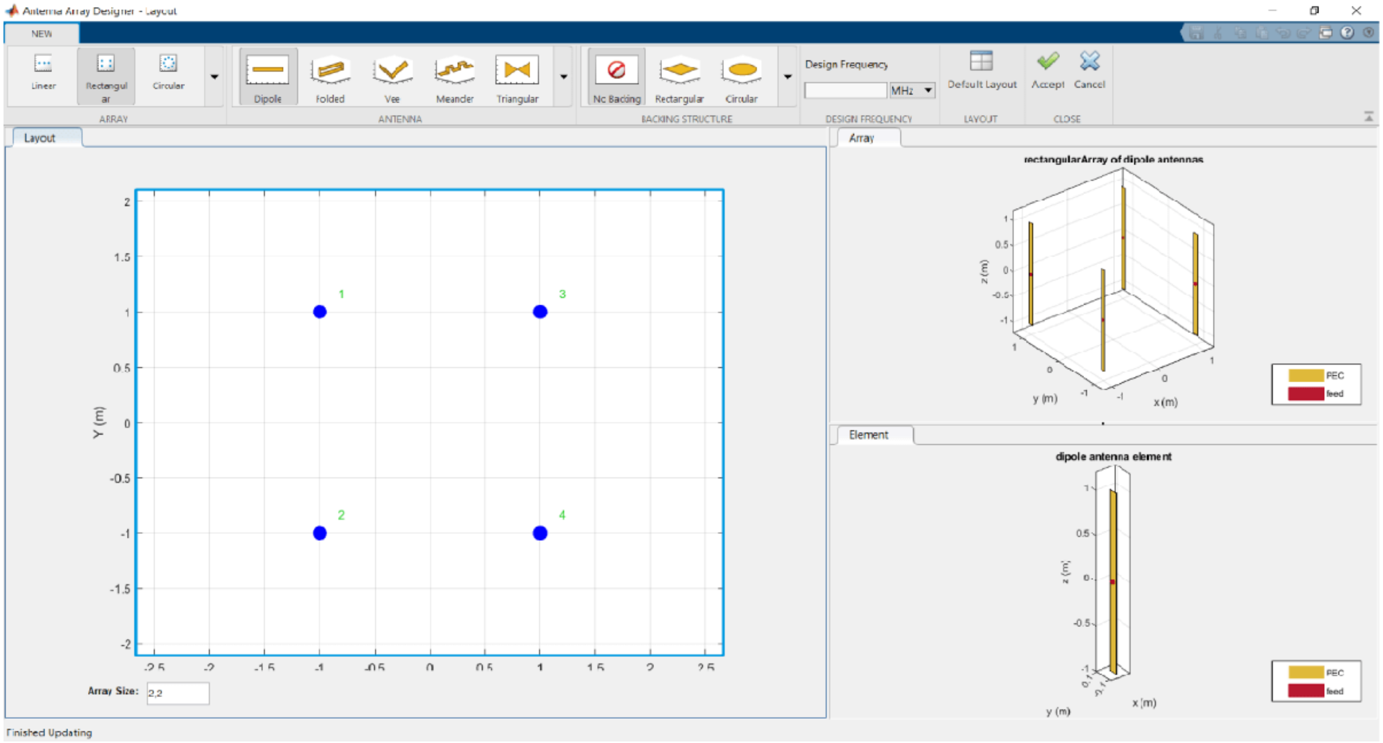
Using the toolstrip, you can choose different types of array layouts, antennas, and backing structures.

You can also specify the **Design Frequency** of the antenna or array. Setting this value scales the individual array elements to resonate at the specified frequency and places the elements at optimal location in the array to avoid interferences.

Click **Accept** to analyze the array characteristics.

Galleries

You can select an `Array Type` from the **Array Gallery**, and you can choose from different antennas from the **Antenna Gallery**.



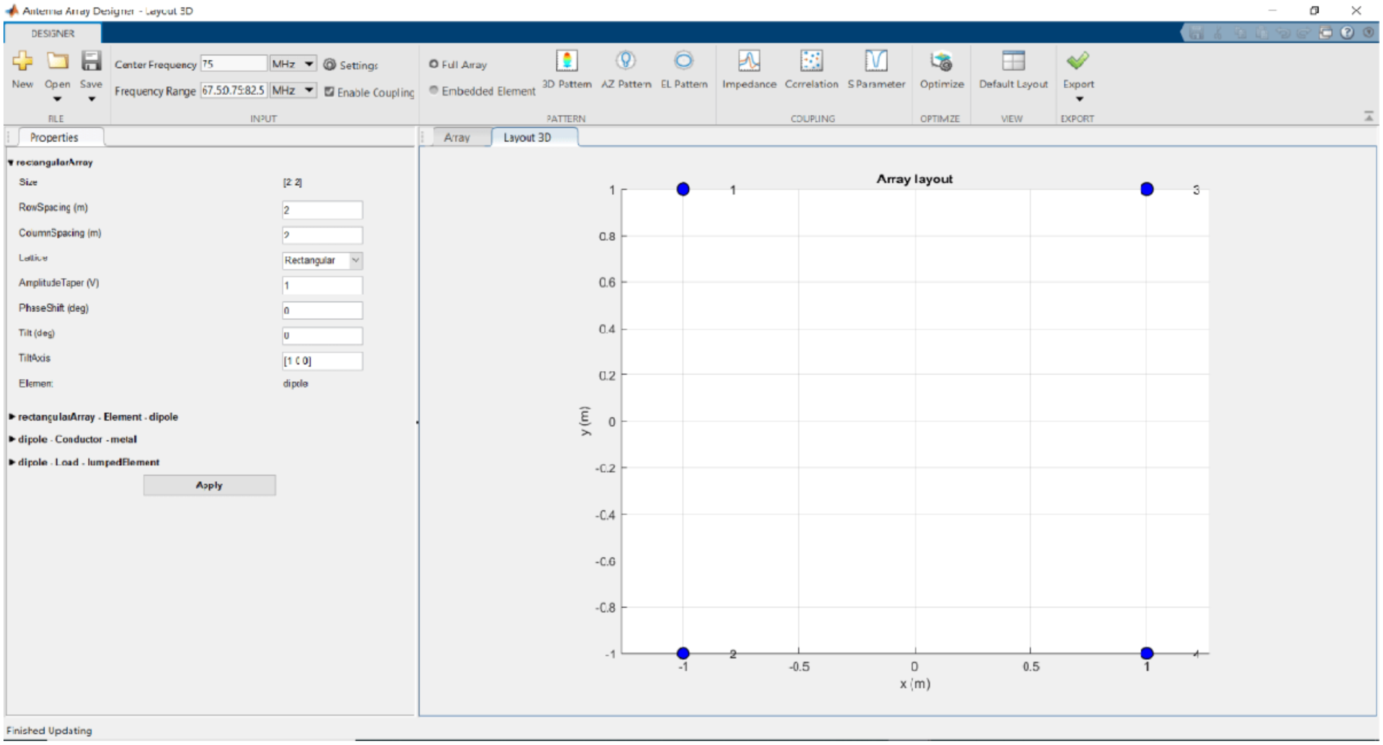
Array Size: 2,2

Finished Updating

You can choose different types of antennas from the **Antenna Gallery**.

You can also choose different types of backing structures for your antenna array elements from the **Backing Structure Gallery**.

Analyze Array



Antenna Array Designer - Layout 3D

Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☑ Enable Coupling

○ Full Array

○ Embedded Element

Properties

▾ rectangularArray

Size	[2 2]
RowSpacing (m)	2
ColumnSpacing (m)	2
Lattice	Rectangular
AmplitudeTaper (V)	1
PhaseShift (deg)	0
Tilt (deg)	0
TiltAxis	[1 0 0]
Element	dipole

▸ rectangularArray - Element - dipole

▸ dipole - Conductor - metal

▸ dipole - Load - lumpedElement

Finished Updating

Once you have clicked **Accept** on a design, you can specify the **Frequency Range** in the Input pane. Then plot the impedance, correlation, or S-parameters of the array using the corresponding buttons in the **Coupling** pane.

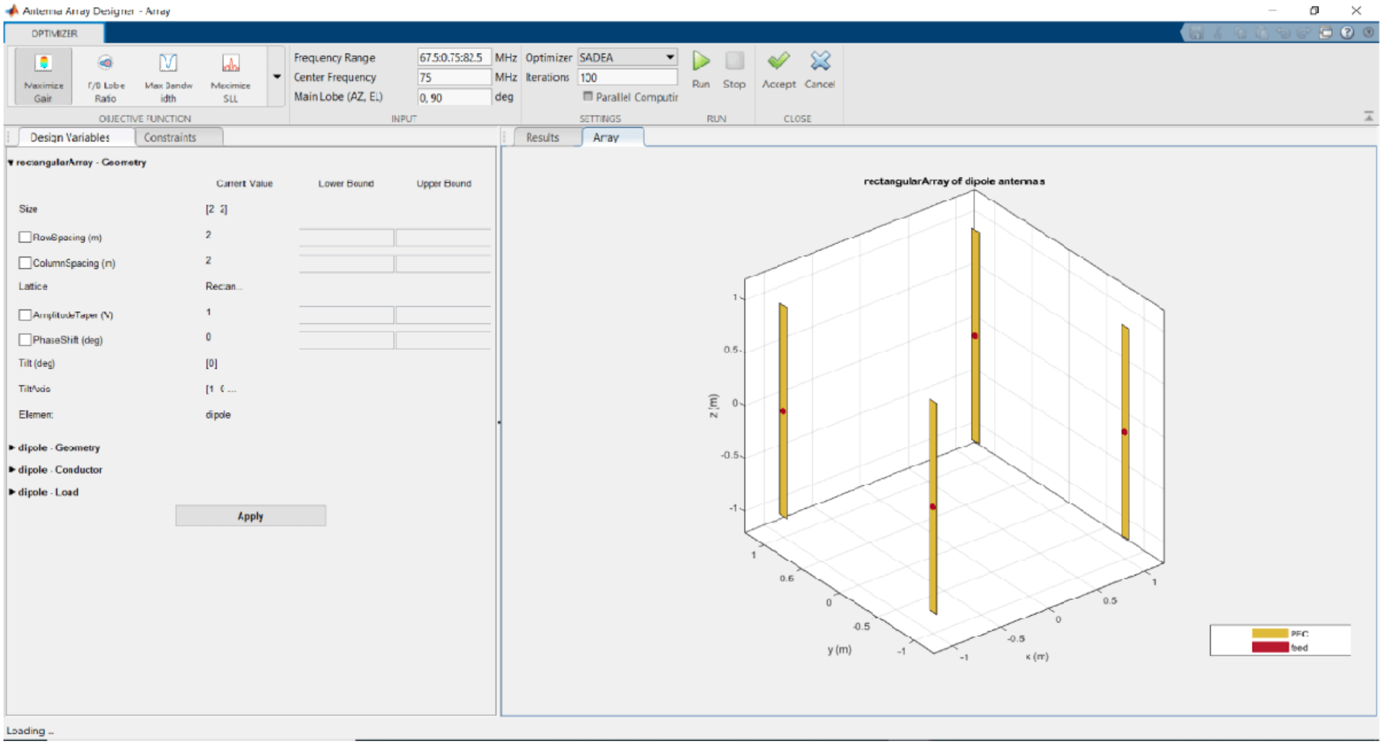
You can visualize the 3-D Pattern, AZ Pattern, or EL Pattern of the full array or an embedded element using the corresponding buttons in the **Pattern** pane. You can also add dielectric substrates to the individual elements or change the value and location of the load using the **Properties** pane.

Use **Properties** to manually change the properties of the array or its individual elements.

Use **Export** to view your array in MATLAB workspace or MATLAB script.

Optimize Array

Click on **Optimize** to open the optimizer canvas of the antenna array designer app.



Frequency Range 67.5:0.75:82.5 MHz
Center Frequency 75 MHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 100
□ Parallel Computir

▾ rectangularArray - Geometry

	Current Value	Lower Bound	Upper Bound
Size	[2 2]		
□ RowSpacing (m)	2		
□ ColumnSpacing (m)	2		
Lattice	Rectan...		
□ AmplitudeTaper (V)	1		
□ PhaseShift (deg)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		
Element	dipole		

▸ dipole - Geometry
▸ dipole - Conductor
▸ dipole - Load

Use the **OBJECTIVE FUNCTION** to choose the main goal of optimizing the array

Use the **Design Variables** to input the variables. The variables are then changed by the optimizer depending on the lower and upper bounds.

Use **Constraints** to restrict a desired analysis function value on the antenna.

Use the Optimizer to choose between `SADEA` or `Surrogate Opt`.

Note: To use the Surrogate optimization algorithm, you need the Global Optimization Toolbox.

After adding the required values, click **Run** to start the optimization.

Linear Dipole Array and Maximum Directivity

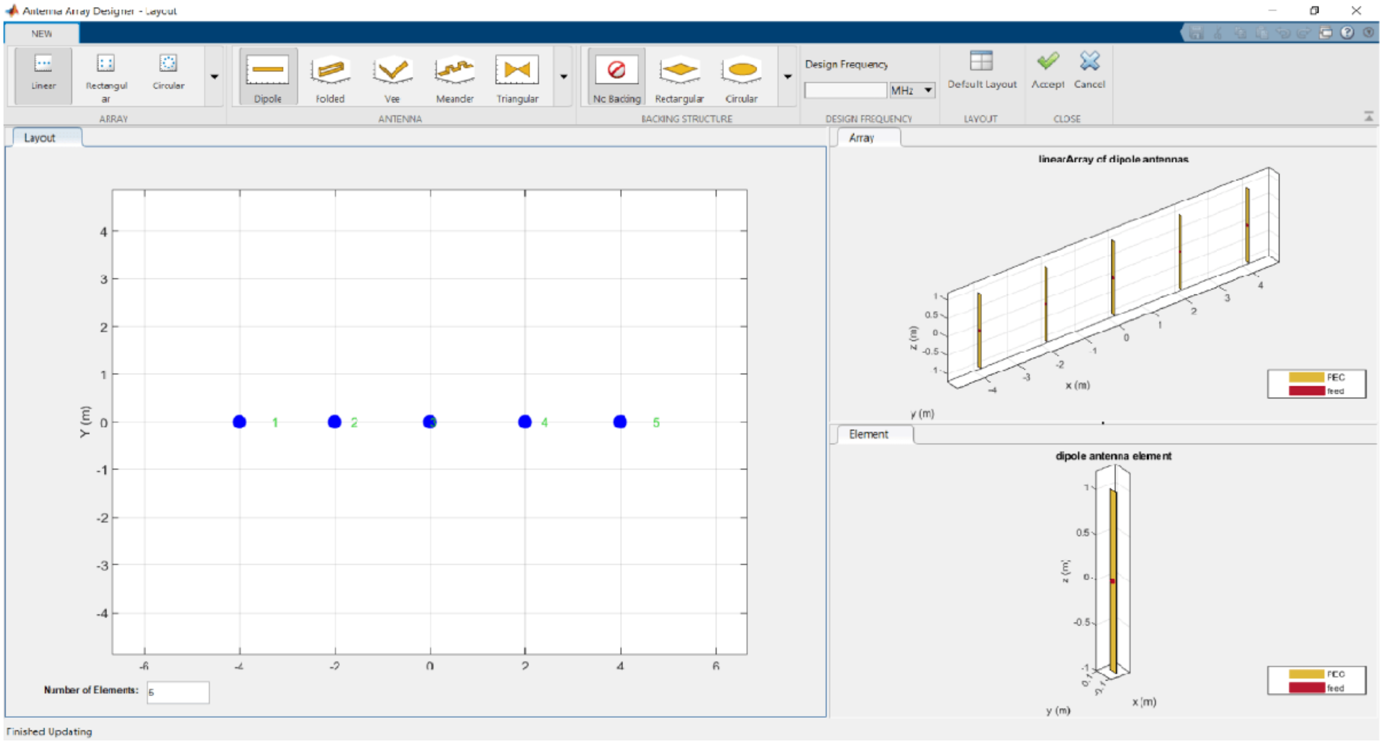
Open the **Antenna Array Designer** app.

```
antennaArrayDesigner
```

Click on the NEW ('+') button to explore antenna library.

Click on **New** and from the **Array Type** pane, click **Linear**.

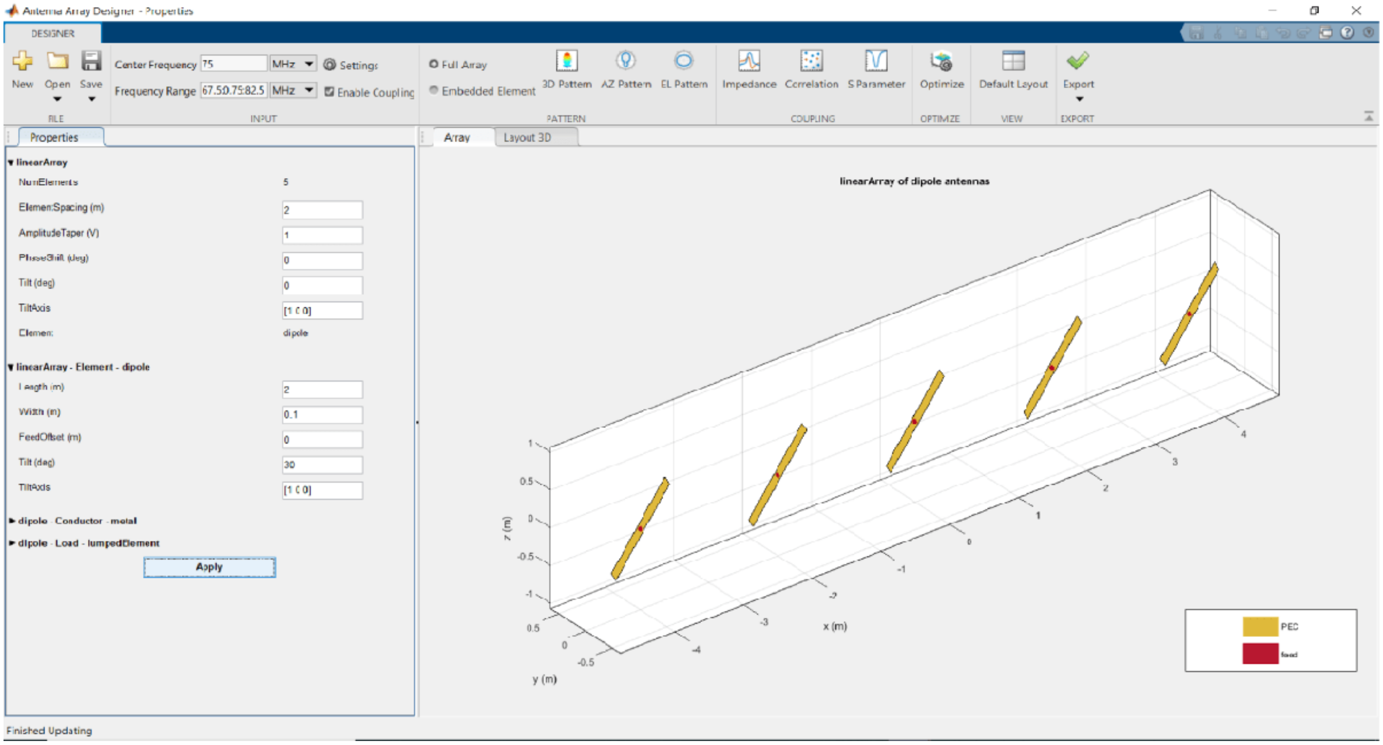
In the bottom left corner, change **Number of Elements** to 5. Click **Accept**.



Number of Elements: 5

Finished Updating

In the **Properties** pane, expand **dipole-Geometry** and change the **Tilt(deg)** to 30. This changes the tilt of each dipole element in the array to 30 degrees. Click on **Array** tab to view the array.



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☑ Enable Coupling

○ Full Array

● Embedded Element

▾ linearArray

NumElements 5

ElementSpacing (m) 2

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 0

TiltAxis [1 0 0]

Element dipole

▾ linearArray - Element - dipole

Length (m) 2

Width (m) 0.1

FeedOffset (m) 0

Tilt (deg) 30

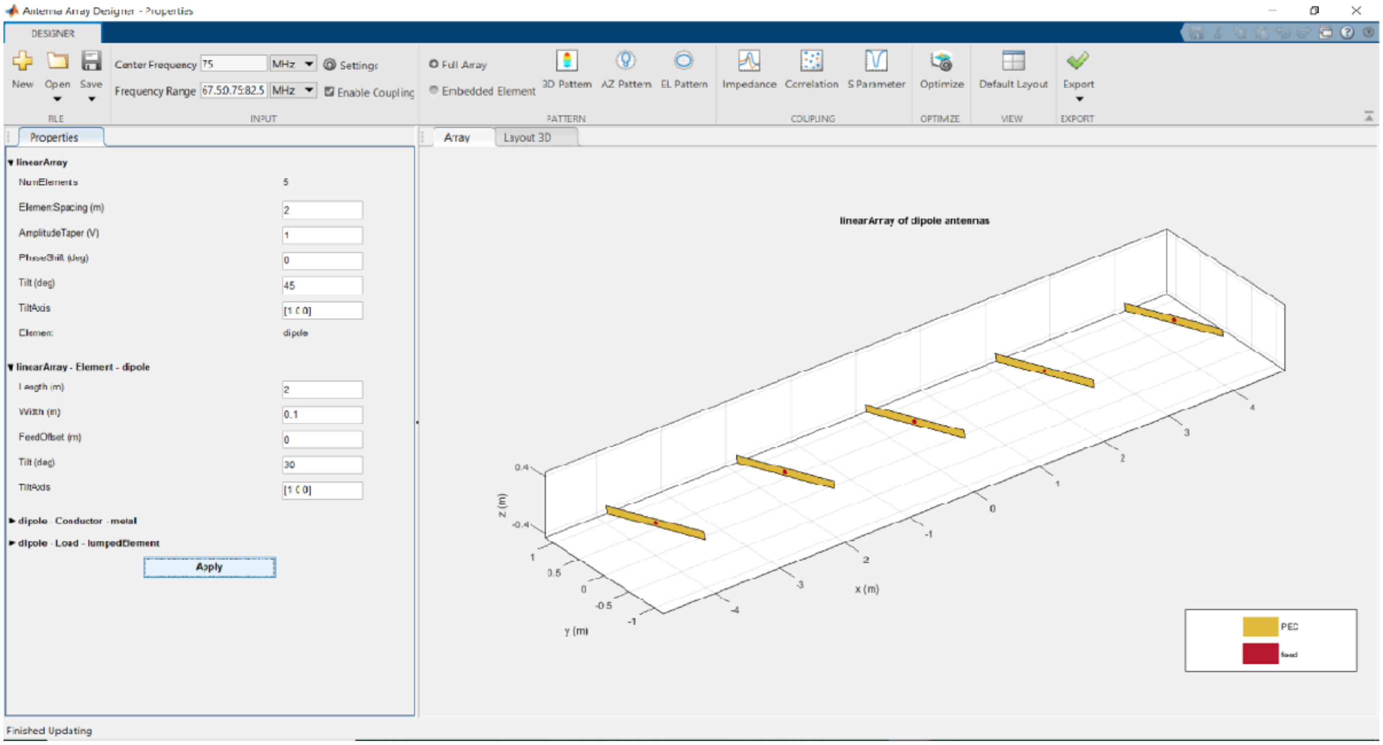
TiltAxis [1 0 0]

▸ dipole - Conductor - metal

▸ dipole - Load - lumpedElement

Finished Updating

In the **Properties** pane, expand **linear-Geometry** and change the **Tilt(deg)** to 45. This changes the tilt of the entire array to 45 degrees.



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☑ Enable Coupling

▼ linearArray

NumElements 5

ElementSpacing (m) 2

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 45

TiltAxis [1 0 0]

Element dipole

▼ linearArray - Element - dipole

Length (m) 2

Width (m) 0.1

FeedOffset (m) 0

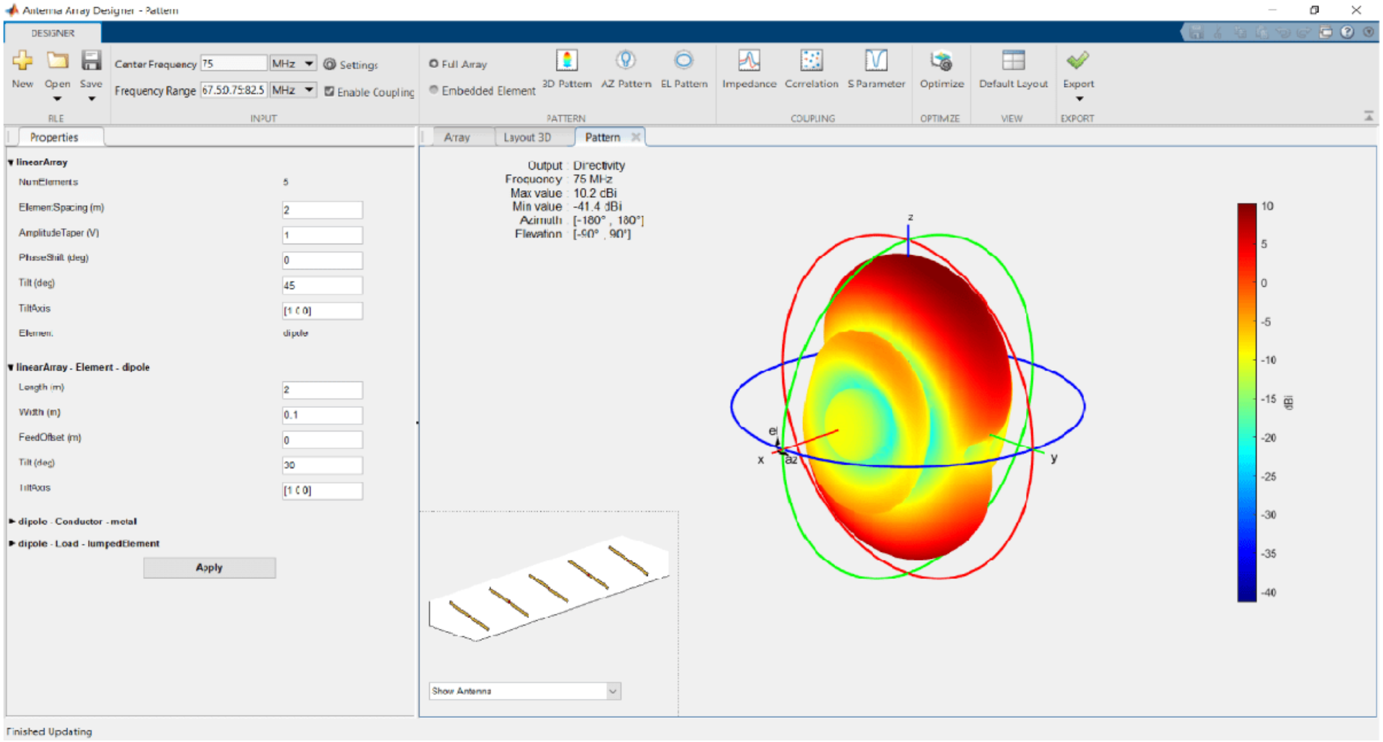
Tilt (deg) 30

TiltAxis [1 0 0]

► dipole - Conductor - metal

► dipole - Load - lumpedElement

On the **Input** pane, change the **Center Frequency** of the array to 60 MHz. Click **3D Pattern** in the **Pattern** pane to plot the radiation pattern. Observe the maximum directivity of the array.



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☑ Enable Coupling

▾ linearArray

NumElements 5

ElementSpacing (m) 2

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 45

TiltAxis [1 0 0]

Element dipole

▾ linearArray - Element - dipole

Length (m) 2

Width (m) 0.1

FeedOffset (m) 0

Tilt (deg) 30

TiltAxis [1 0 0]

▸ dipole - Conductor - metal

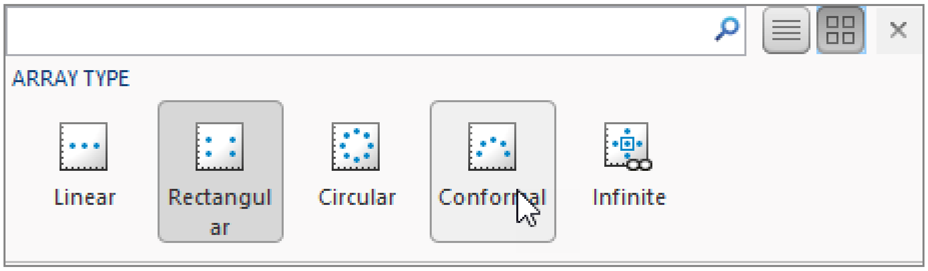
▸ dipole - Load - lumpedElement

```
   Output : Directivity
Frequency : 75 MHz
Max value : 10.2 dBi
Min value : -41.4 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

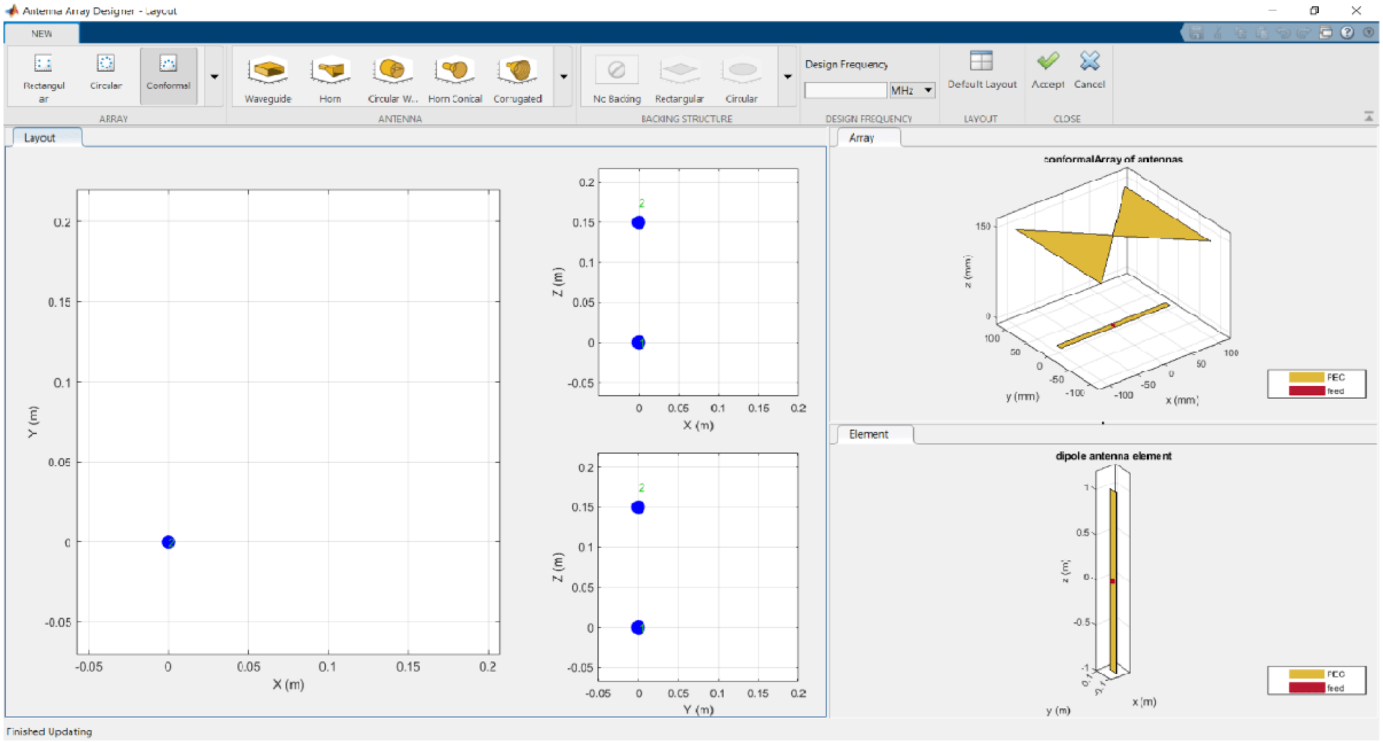
Finished Updating

Conformal Array Design and Analysis

Open **Antenna Array Designer** app. In the **Array Gallery** pane, click **Conformal**.



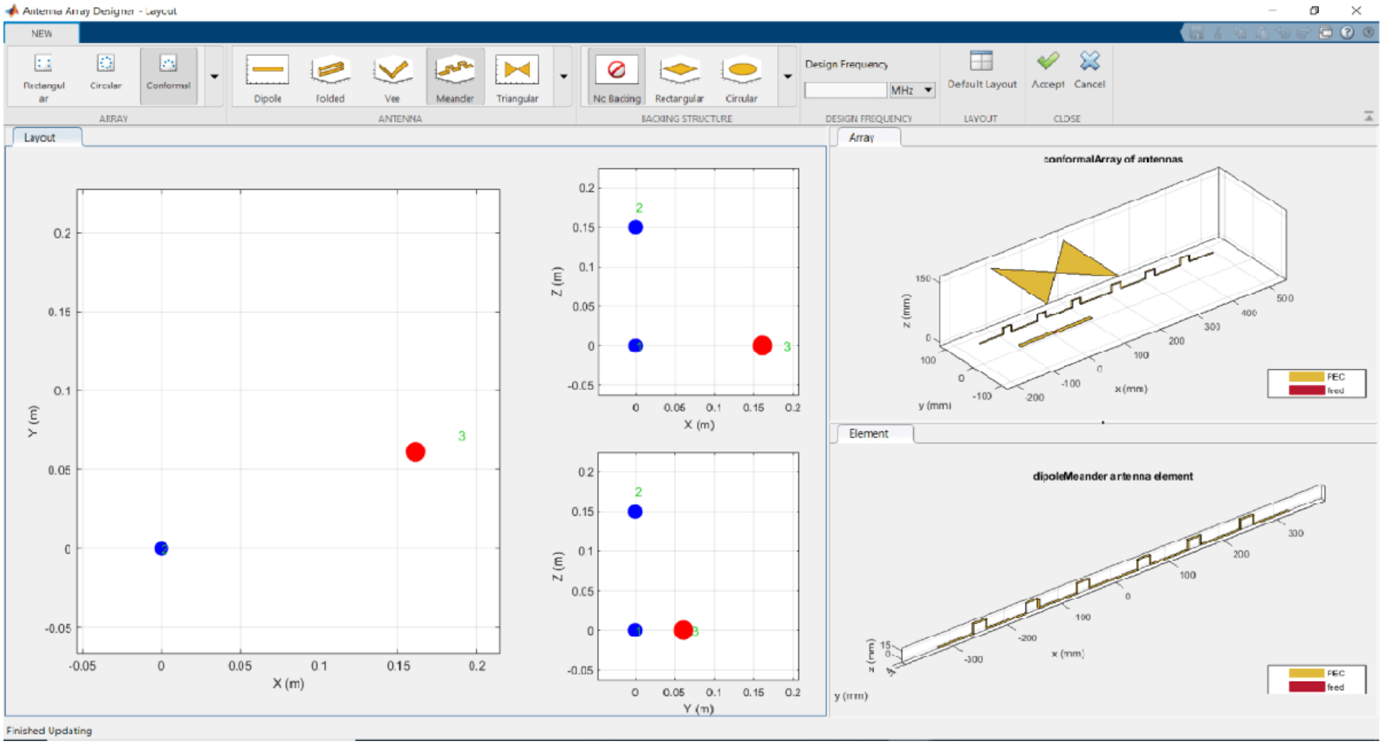
The default conformal array consists of a dipole antenna and a bowtie antenna.



You can view each element separately by clicking on the element in the **Layout** window.

Meander Antenna with Rectangular Backing

Add a meander dipole antenna with rectangular backing. From the **ANTENNA GALLERY**, click **Meander** to create a meander dipole antenna. Move the antenna by dragging the antenna in the **Layout** window.

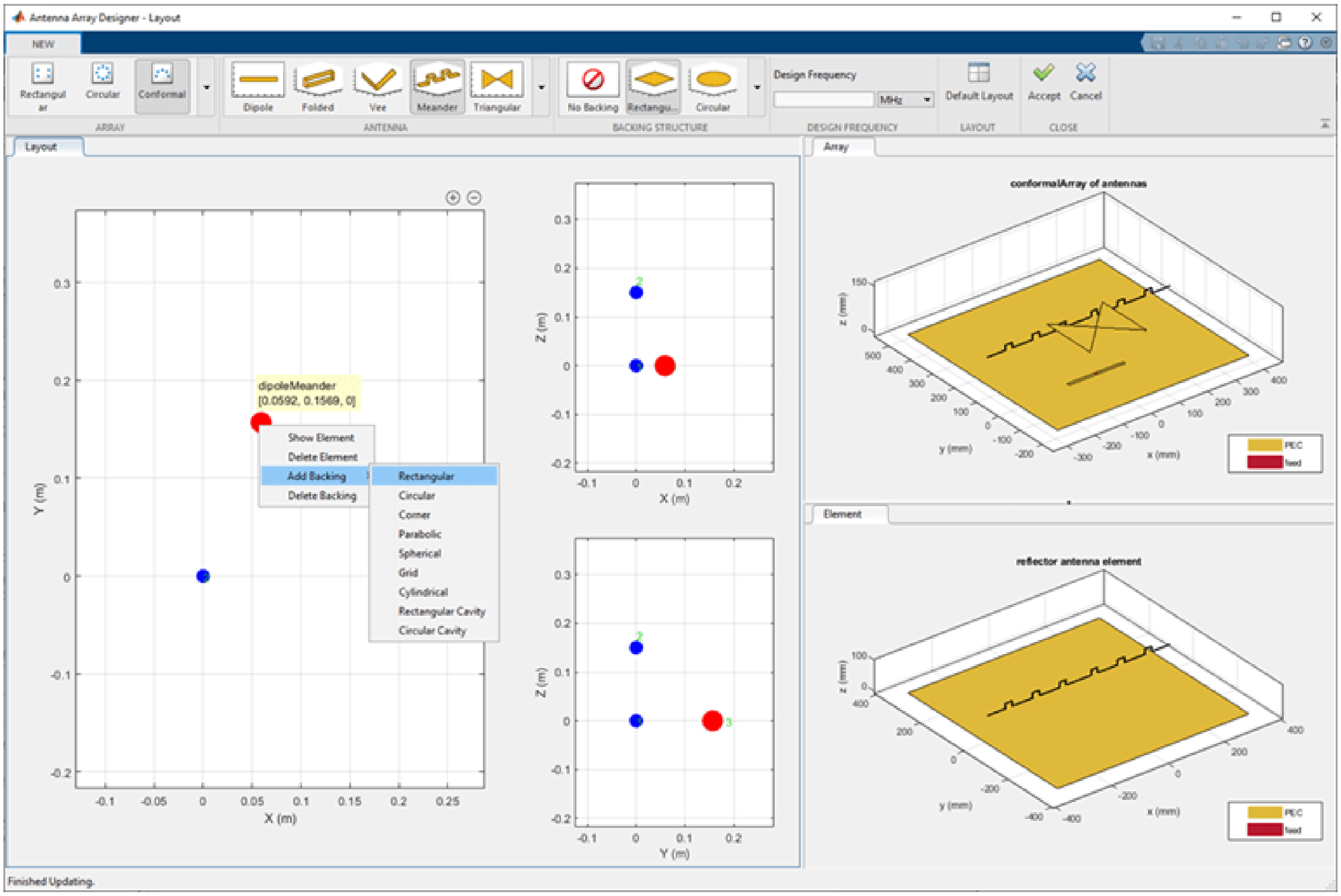


To add the rectangular backing:

- Choose the meander dipole antenna from the **Layout** window and then click **Rectangular** in the **BACKING STRUCTURE GALLERY** pane.

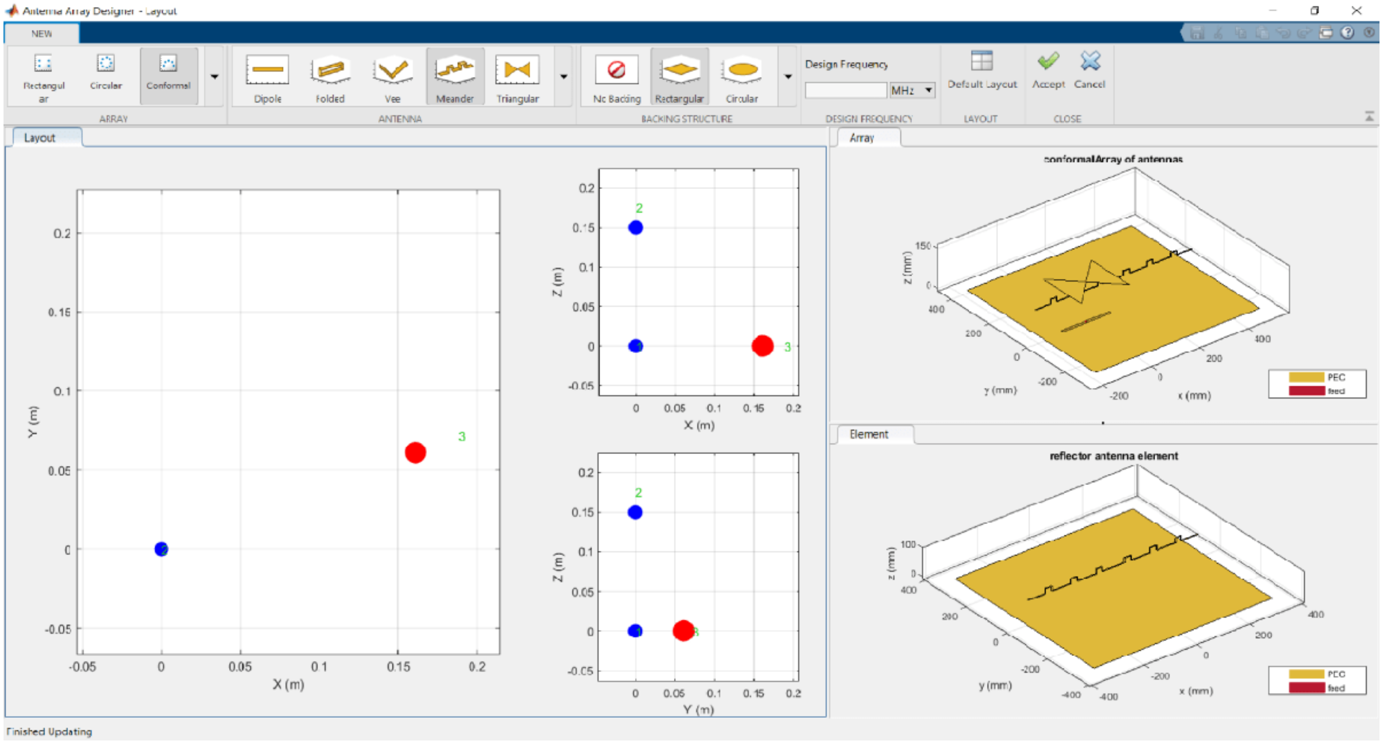
or

- Right click on the antenna in the **Layout** window and select **Add Backing > Rectangular Reflector**.



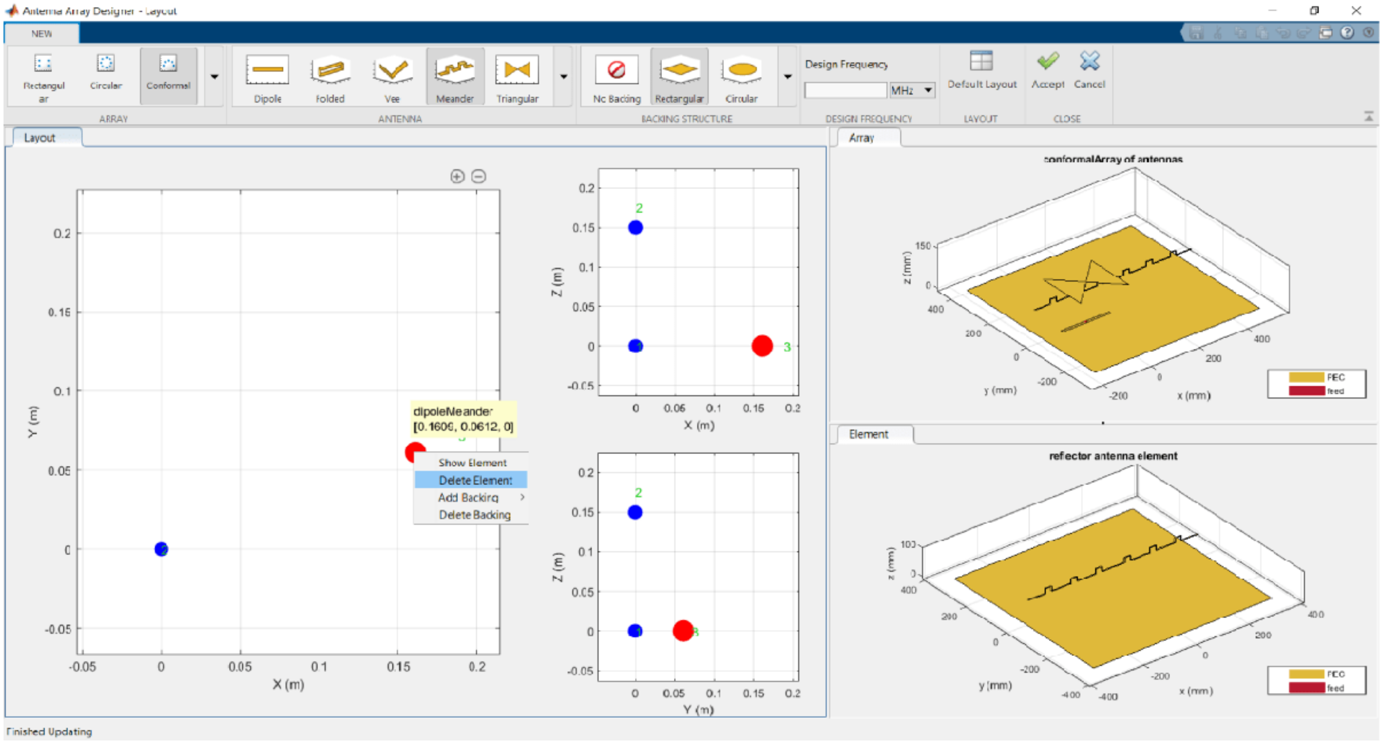
dipoleMeander
[0.0592, 0.1569, 0]

Finished Updating.

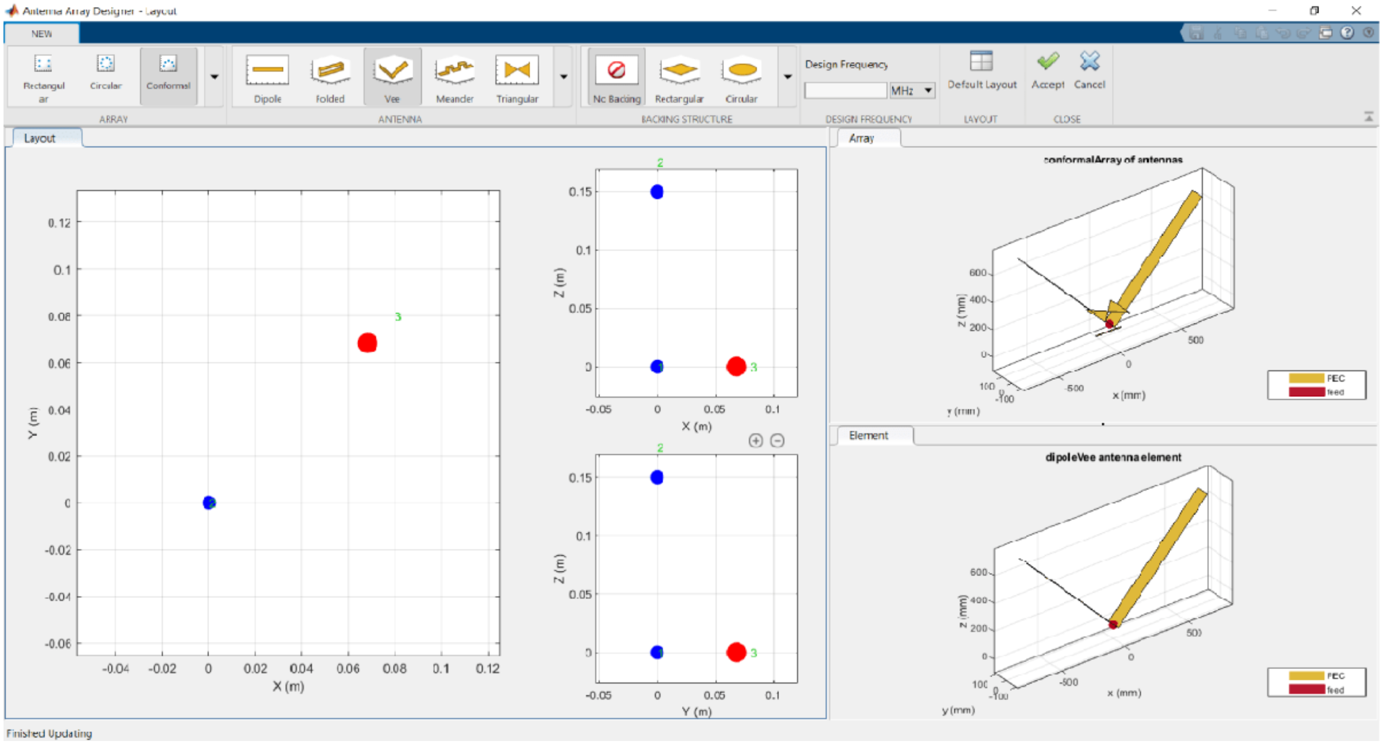


Delete Meander and Add V-Dipole

To delete the meander dipole antenna, right click from the **Layout** window, and select **Delete**.



Click **Vee** from the **Antenna Gallery** to add a V-dipole antenna.



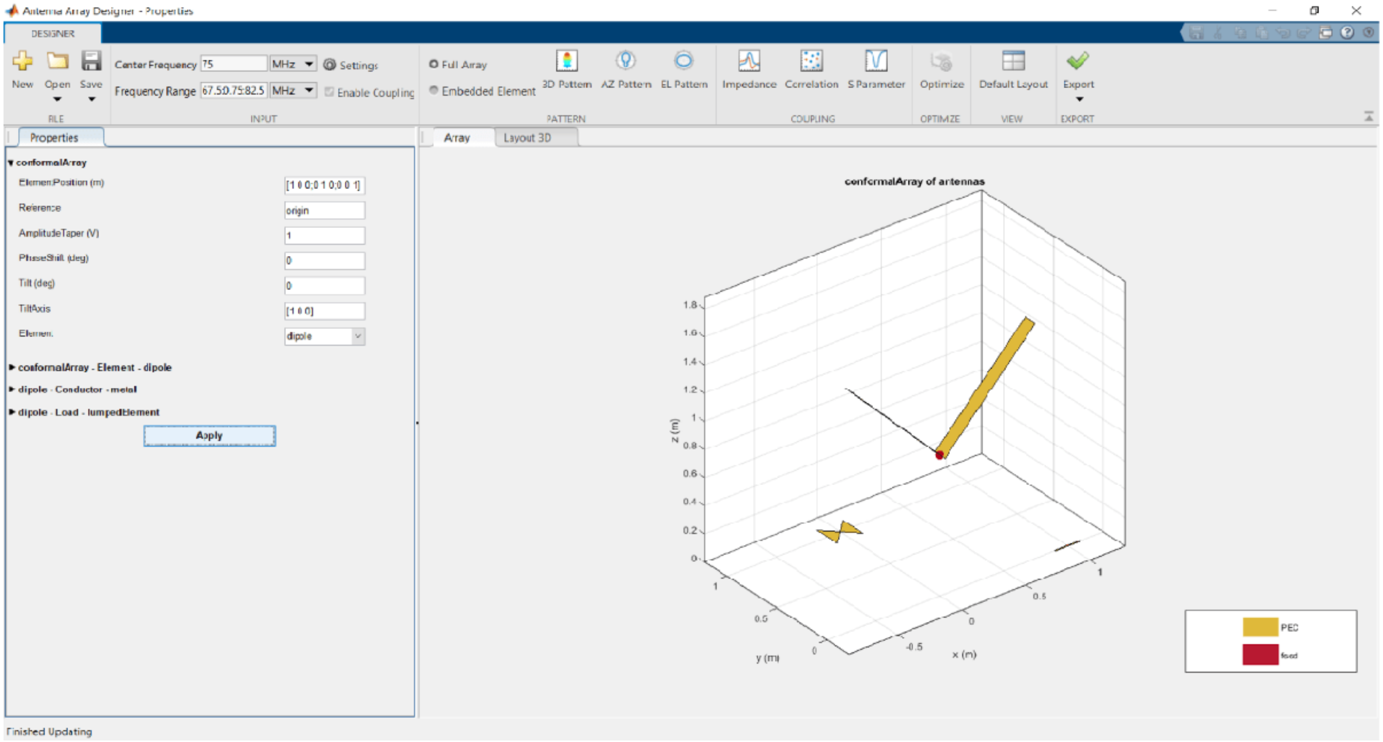
Click **Accept**.

Antenna Placement

Place the antennas at the following locations in the X-Y-Z plane:

- Element 1 - dipole - [1 0 0]
- Element 2 - bowtie - [0 1 0]
- Element 3 - V-dipole - [0 0 1]

In the **Properties** pane, expand **conformalArray - Geometry** and change the values of **ElementPosition(m)** to [1 0 0;0 1 0;0 0 1]. Click **Apply**.



Antenna Array Designer - Properties

Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

conformalArray

ElementPosition (m) [1 0 0;0 1 0;0 0 1]

Reference origin

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 0

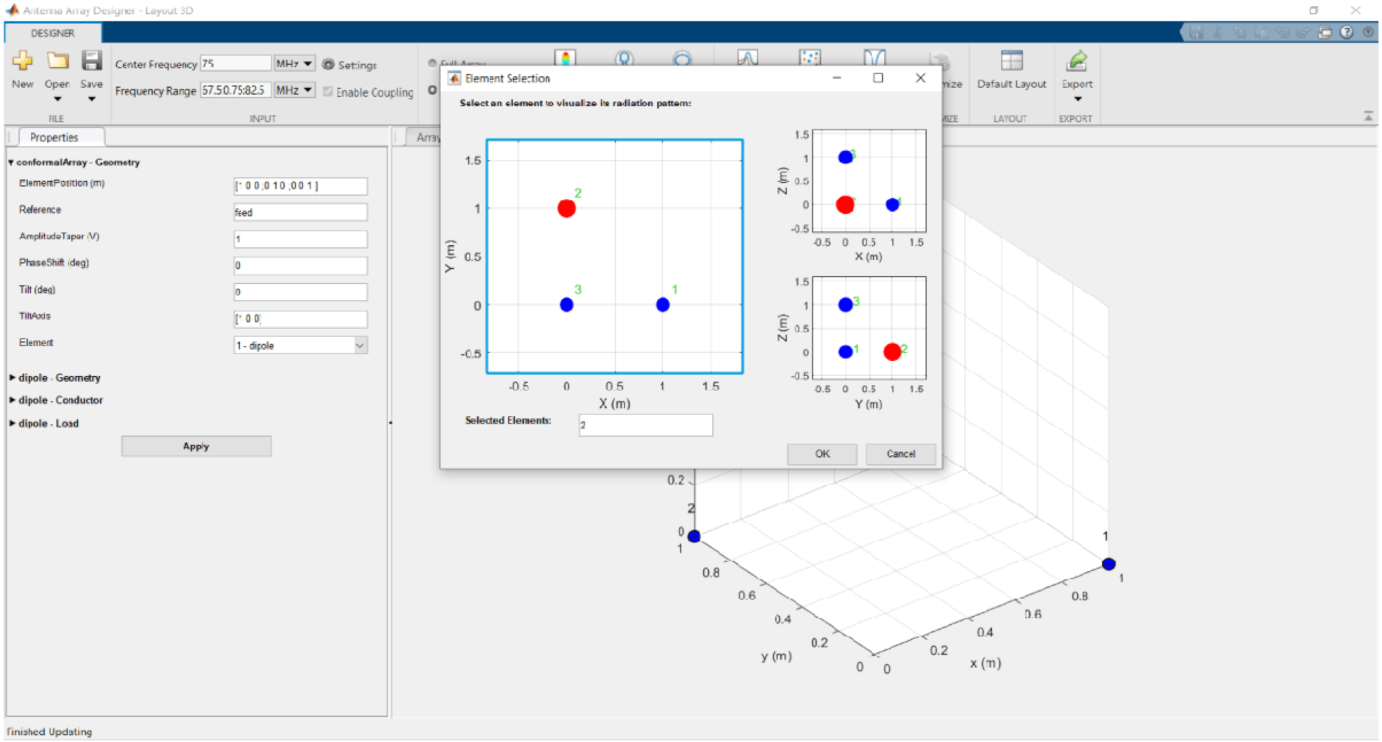
TiltAxis [1 0 0]

Element dipole

Finished Updating

Embedded Element Pattern and Half-Power Beam Width (HPBW)

Show the embedded element pattern in the azimuth plane for element 2. Choose **Embedded Element** in the **PATTERN** pane. Click **AZ Pattern**. From the element selection window, click element 2 and then **OK**.



Antenna Array Designer - Layout 3D

Center Frequency 75 MHz

Frequency Range 57.50.75:82.5 MHz

☑ Enable Coupling

conformalArray - Geometry

ElementPosition (m) [1 0 0;0 1 0 ;0 0 1]

Reference feed

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 0

TiltAxis [1 0 0]

Element 1 - dipole

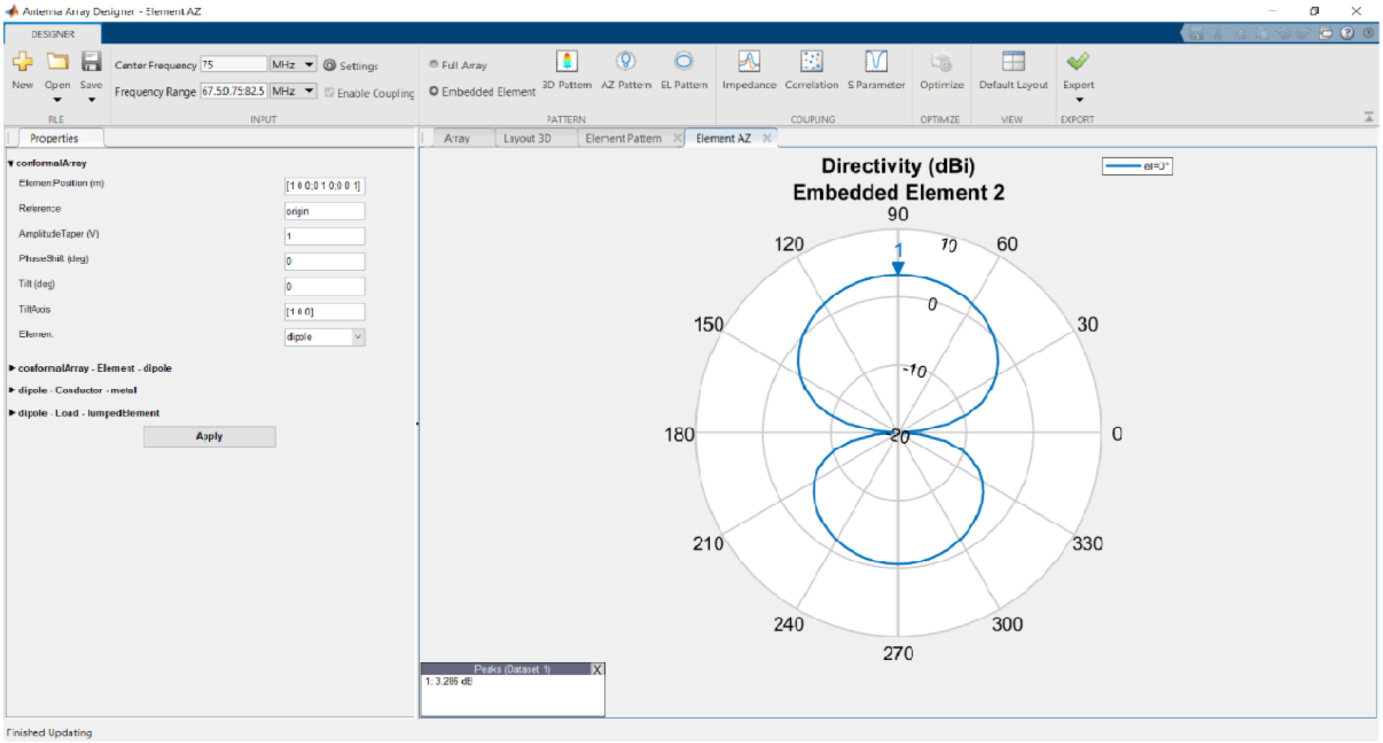
Element Selection

Select an element to visualize its radiation pattern:

Selected Elements: 2

Finished Updating

To view the HPBW, right click on the azimuth pattern and select **Measurements > Antenna Metrics**.



Antenna Array Designer - Element AZ

Center Frequency 75 MHz

Frequency Range 67.50.75:82.5 MHz

☑ Enable Coupling

○ Full Array

● Embedded Element

conformalArray

ElementPosition (m) [1 0 0;0 1 0;0 0 1]

Reference origin

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 0

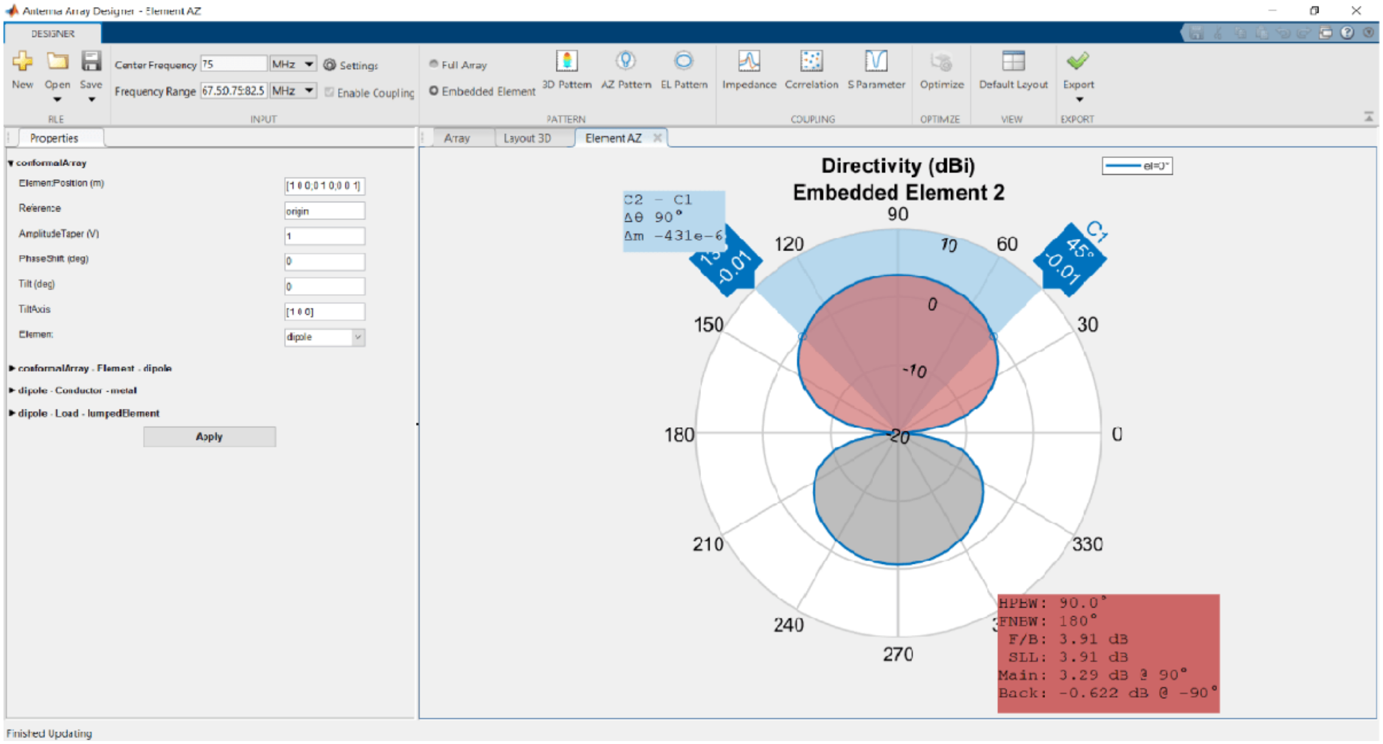
TiltAxis [1 0 0]

Element dipole

Peaks (Dataset 1)

1: 3.286 dB

Finished Updating



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☑ Enable Coupling

○ Full Array

● Embedded Element

▾ conformalArray

ElementPosition (m)	[1 0 0;0 1 0;0 0 1]
Reference	origin
AmplitudeTaper (V)	1
PhaseShift (deg)	0
Tilt (deg)	0
TiltAxis	[1 0 0]
Element	dipole

▸ conformalArray - Element - dipole

▸ dipole - Conductor - metal

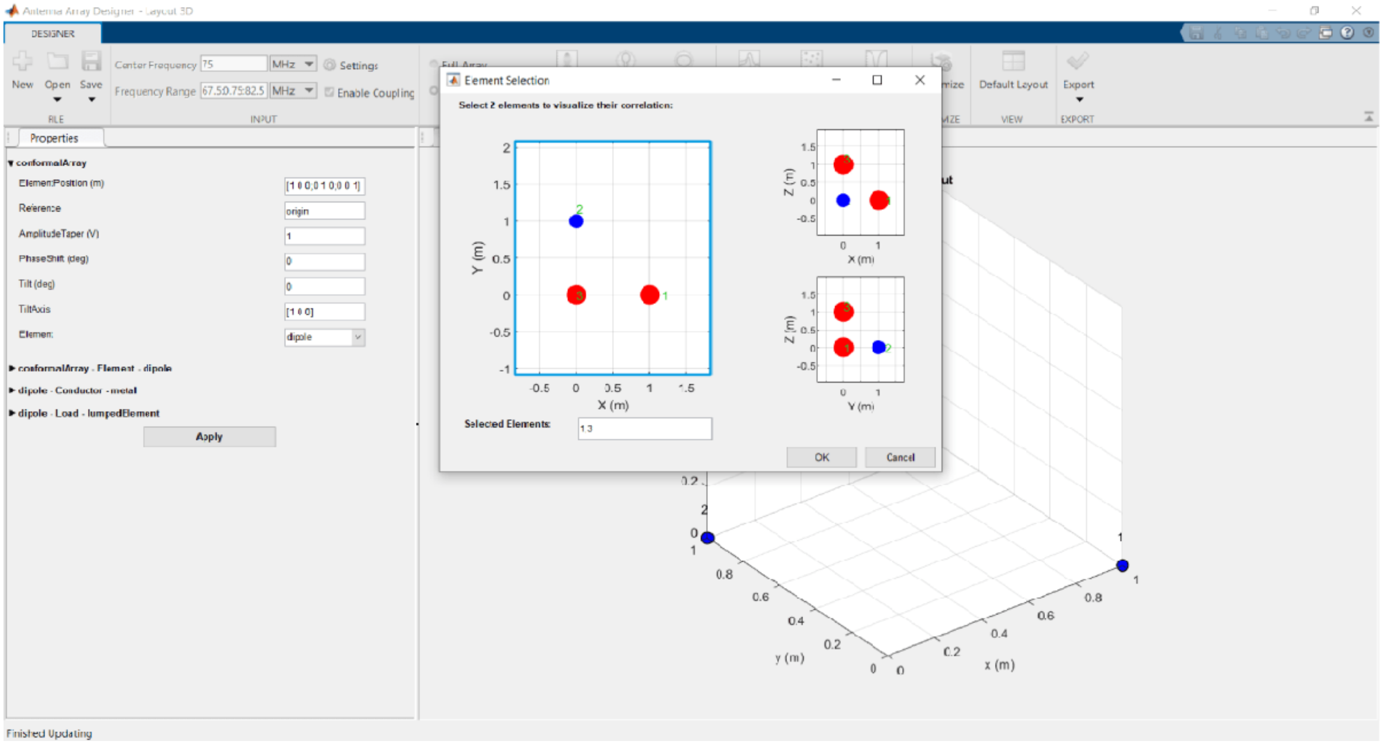
▸ dipole - Load - lumpedElement

```
C2 - C1
Δθ 90°
Δm -431e-6
```

```
HPBW: 90.0°
FNBW: 180°
 F/B: 3.91 dB
 SLL: 3.91 dB
Main: 3.29 dB @ 90°
Back: -0.622 dB @ -90°
```

Coupling Between Elements

To observe the coupling between elements 1 and 3, make sure that the **Enable Coupling** is selected in the **INPUT** pane. In the **COUPLING**, click **`Correlation`**. From the element selection window, click 1 and 3.



Antenna Array Designer - Layout 3D

Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☑ Enable Coupling

Properties

▼ conformalArray

ElementPosition (m)	[1 0 0;0 1 0;0 0 1]
Reference	origin
AmplitudeTaper (V)	1
PhaseShift (deg)	0
Tilt (deg)	0
TiltAxis	[1 0 0]
Element	dipole

▶ conformalArray - Element - dipole

▶ dipole - Conductor - metal

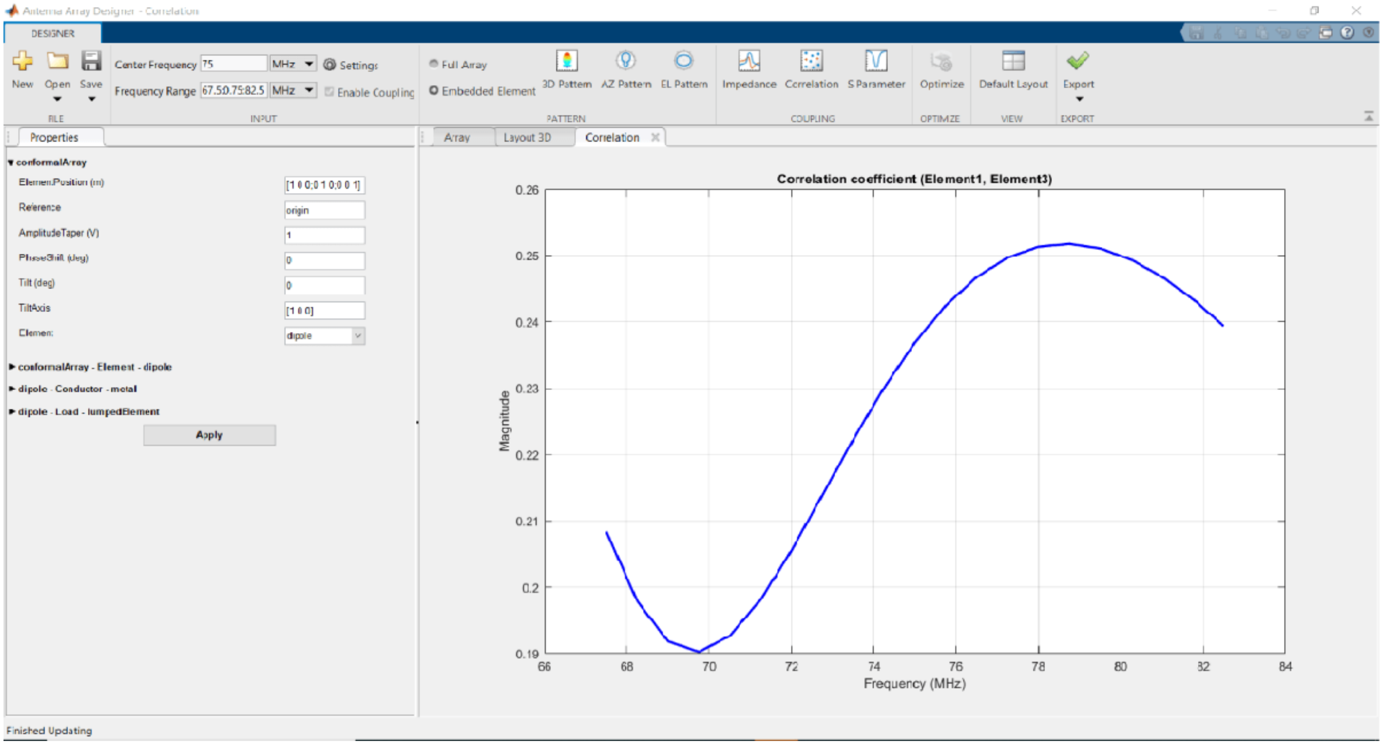
▶ dipole - Load - lumpedElement

Element Selection

Select 2 elements to visualize their correlation:

Selected Elements: 1 3

Finished Updating



Antenna Array Designer - Correlation

Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☐ Enable Coupling

○ Full Array

● Embedded Element

Properties

▼ conformalArray

ElementPosition (m)	[1 0 0;0 1 0;0 0 1]
Reference	origin
AmplitudeTaper (V)	1
PhaseShift (deg)	0
Tilt (deg)	0
TiltAxis	[1 0 0]
Element	dipole

▶ conformalArray - Element - dipole

▶ dipole - Conductor - metal

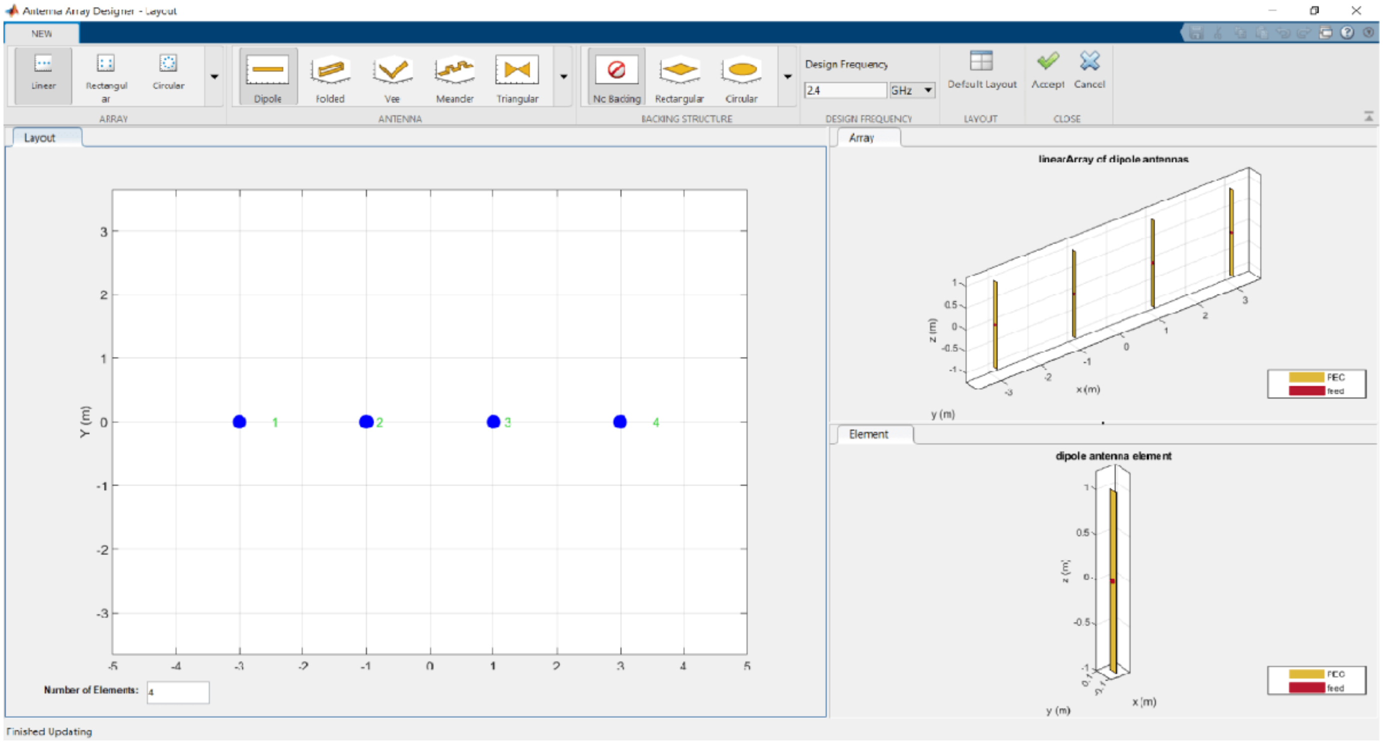
▶ dipole - Load - lumpedElement

Finished Updating

Optimizing Linear Dipole Antenna Array

Open the **Antenna Array Designer** app. In the Array Gallery section, select the array type as **Linear.**

Select **Dipole** from **Antenna Gallery**. Select **No Backing** under the **Backing Structure Gallery**. Specify the design frequency as 2.4 GHz. In **Layout** pane, specify the **Number of Elements** as 4 and click **Accept** under the **Close** section.


Antenna Array Designer - Layout

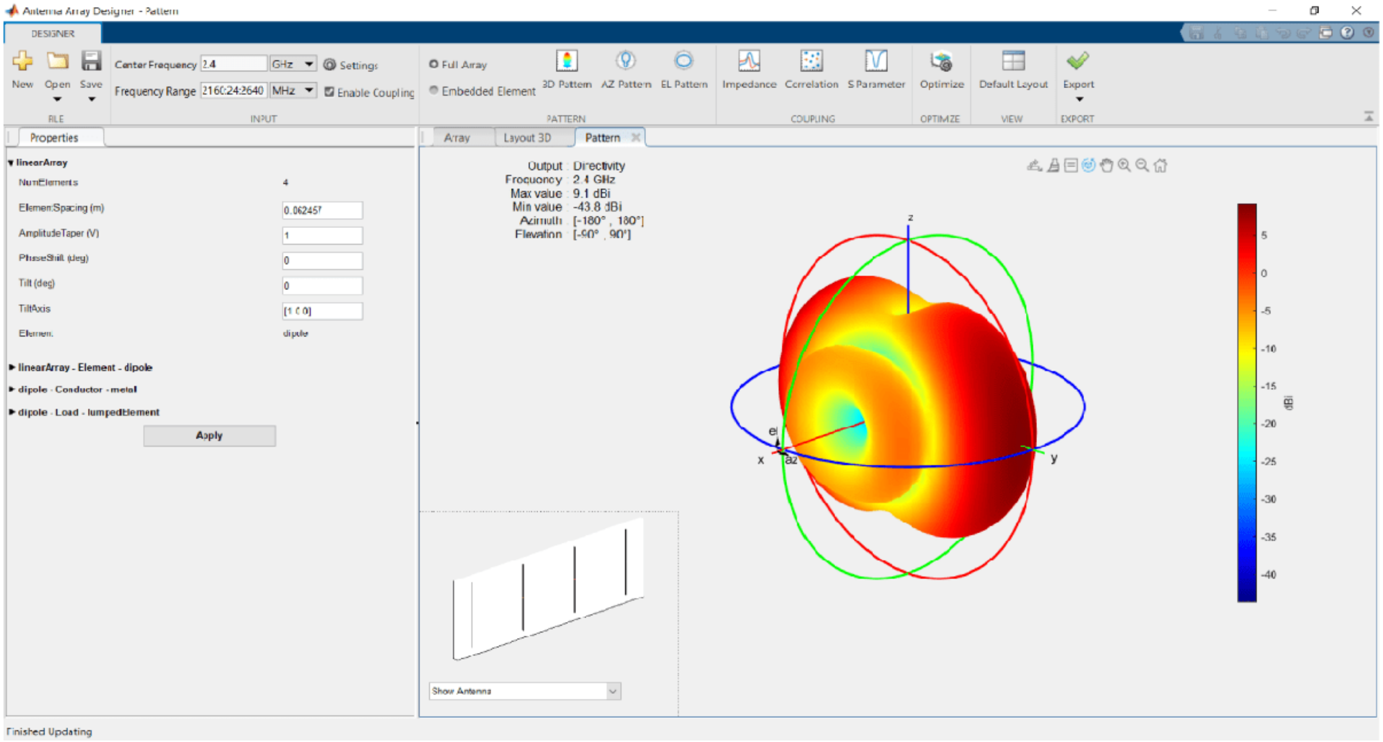
Design Frequency: 2.4 GHz

Number of Elements: 4

Finished Updating

Select **3D Pattern** under **Pattern** section to calculate the 3-D radiation pattern.

The gain is 9.1 dBi. Click **Optimize** on the app toolstrip to optimize this array.



Antenna Array Designer - Pattern

Center Frequency 2.4 GHz

Frequency Range 2160:24:2640 MHz

☑ Enable Coupling

○ Full Array

○ Embedded Element

linearArray

NumElements 4

ElementSpacing (m) 0.062457

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 0

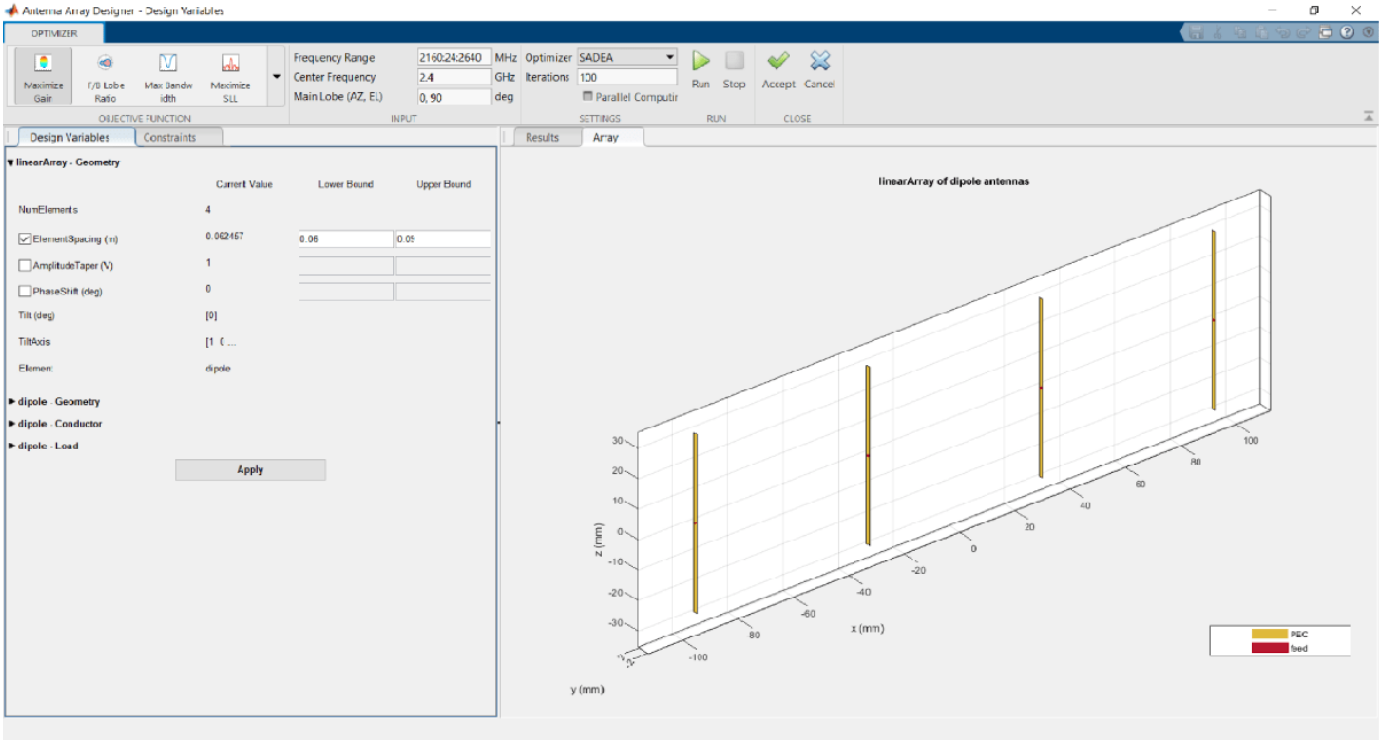
TiltAxis [1 0 0]

Element dipole

Output : Directivity
Frequency : 2.4 GHz
Max value : 9.1 dBi
Min value : -43.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Finished Updating

On the Optimizer tab, click **Maximize Gain** in the **Objective Function** section. In the **Design Variables pane**, select the variables you want to optimize. In this example, select the **Element Spacing** variable and set the lower and upper bounds to 0.06 and 0.09.



Antenna Array Designer - Design Variables

Frequency Range 2160:24:2640 MHz

Center Frequency 2.4 GHz

Main Lobe (AZ, EL) 0, 90 deg

Optimizer SADEA

Iterations 100

□ Parallel Computing

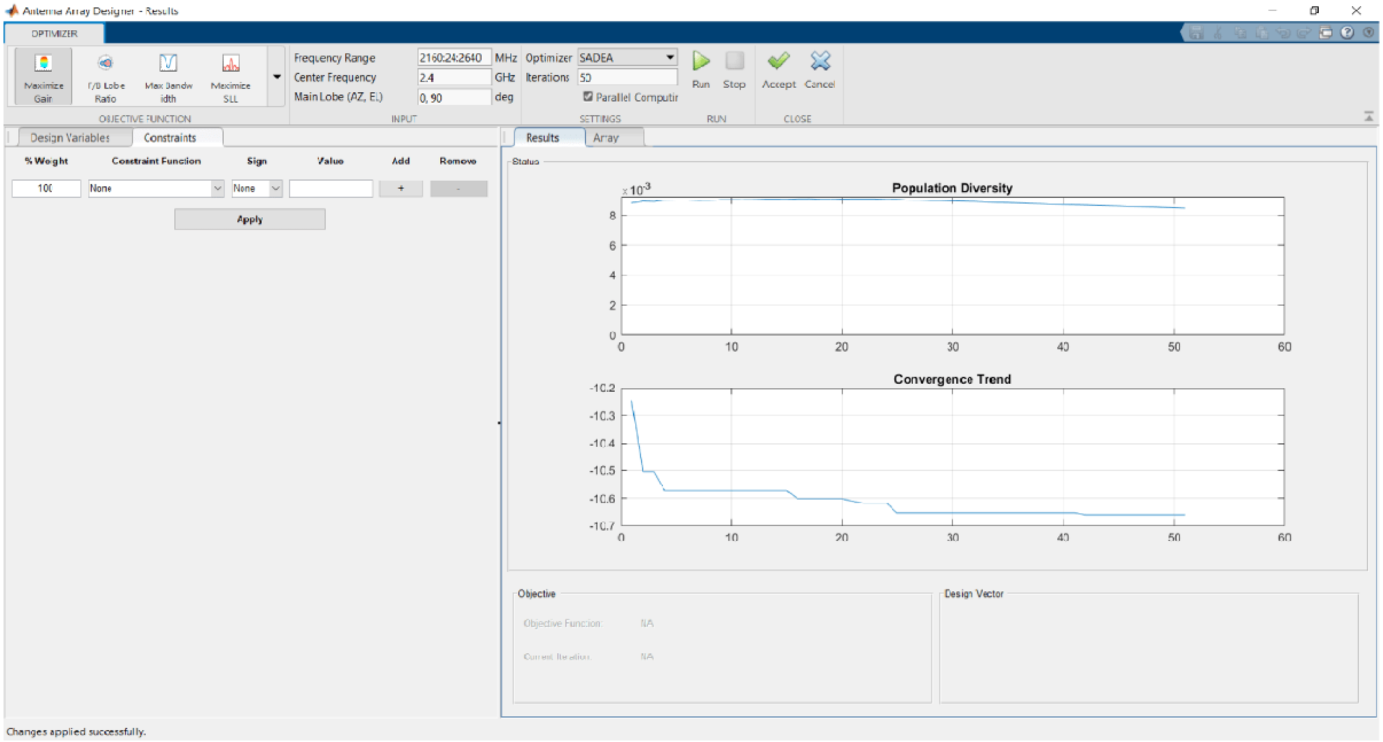
linearArray - Geometry

	Current Value	Lower Bound	Upper Bound
NumElements	4		
☑ ElementSpacing (m)	0.062457	0.06	0.09
□ AmplitudeTaper (V)	1		
□ PhaseShift (deg)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		
Element	dipole		

Click the **Constraints** pane. In this example, there are no **Constraint Functions**. If your application requires constraints, chose one or more constraint functions from the dropdown.

Click **Apply** to apply the design variables in this example. In the **Settings** section, set the number of iterations to `50`, select **Parallel Computing** if you have Parallel Computing Toolbox™, and click **Run.**

Once the simulation is complete, the optimization results are displayed in the **Results** pane



Antenna Array Designer - Results

Frequency Range 2160:24:2640 MHz

Center Frequency 2.4 GHz

Main Lobe (AZ, EL) 0, 90 deg

Optimizer SADEA

Iterations 50

☑ Parallel Computir

% Weight	Constraint Function	Sign	Value	Add	Remove
100	None	None		+	-

Objective

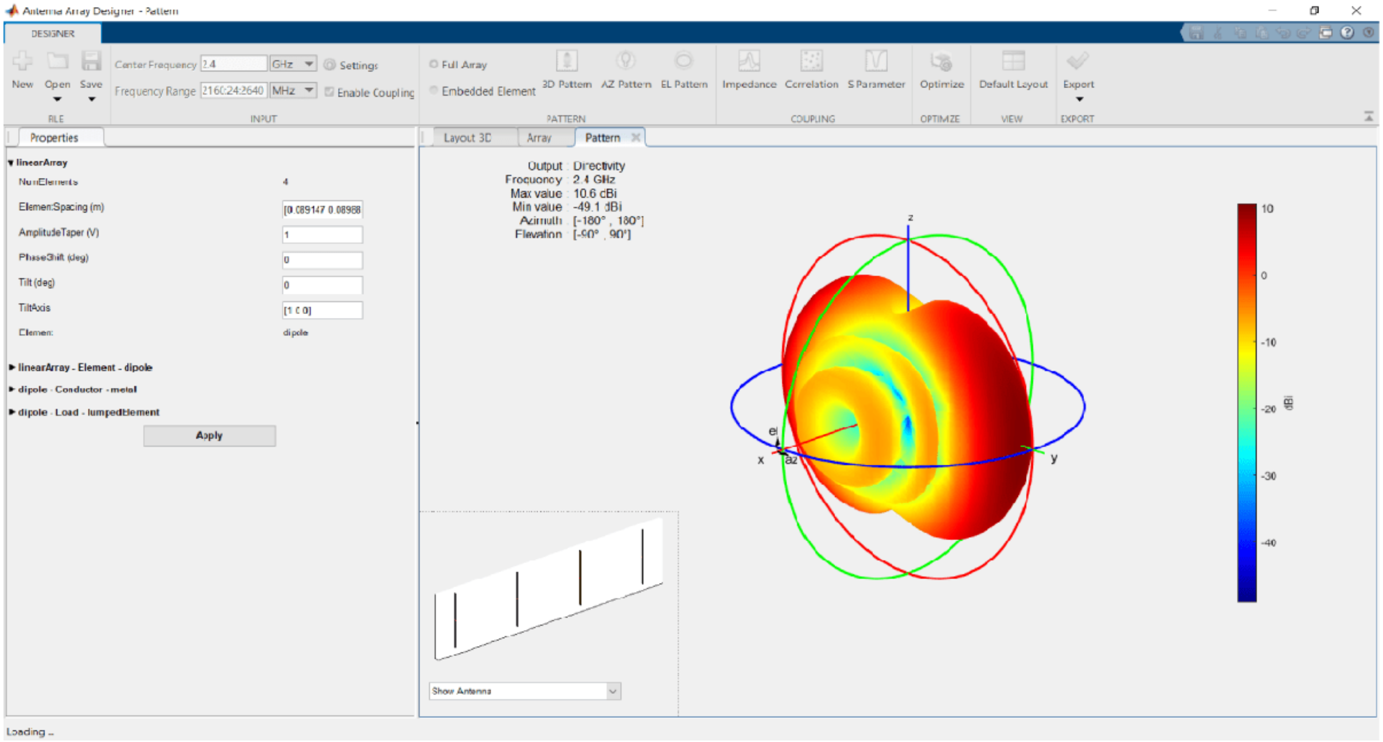
Objective Function: NA

Current Iteration: NA

Design Vector

Changes applied successfully.

Click **Accept**. In the **Pattern** section, plot the **3D Pattern** again. The gain has now increased to 10.7 dBi.



Center Frequency 2.4 GHz

Frequency Range 2160:24:2640 MHz

☑ Enable Coupling

linearArray

NumElements	4
ElementSpacing (m)	[0.089147 0.08988
AmplitudeTaper (V)	1
PhaseShift (deg)	0
Tilt (deg)	0
TiltAxis	[1 0 0]
Element	dipole

linearArray - Element - dipole

dipole - Conductor - metal

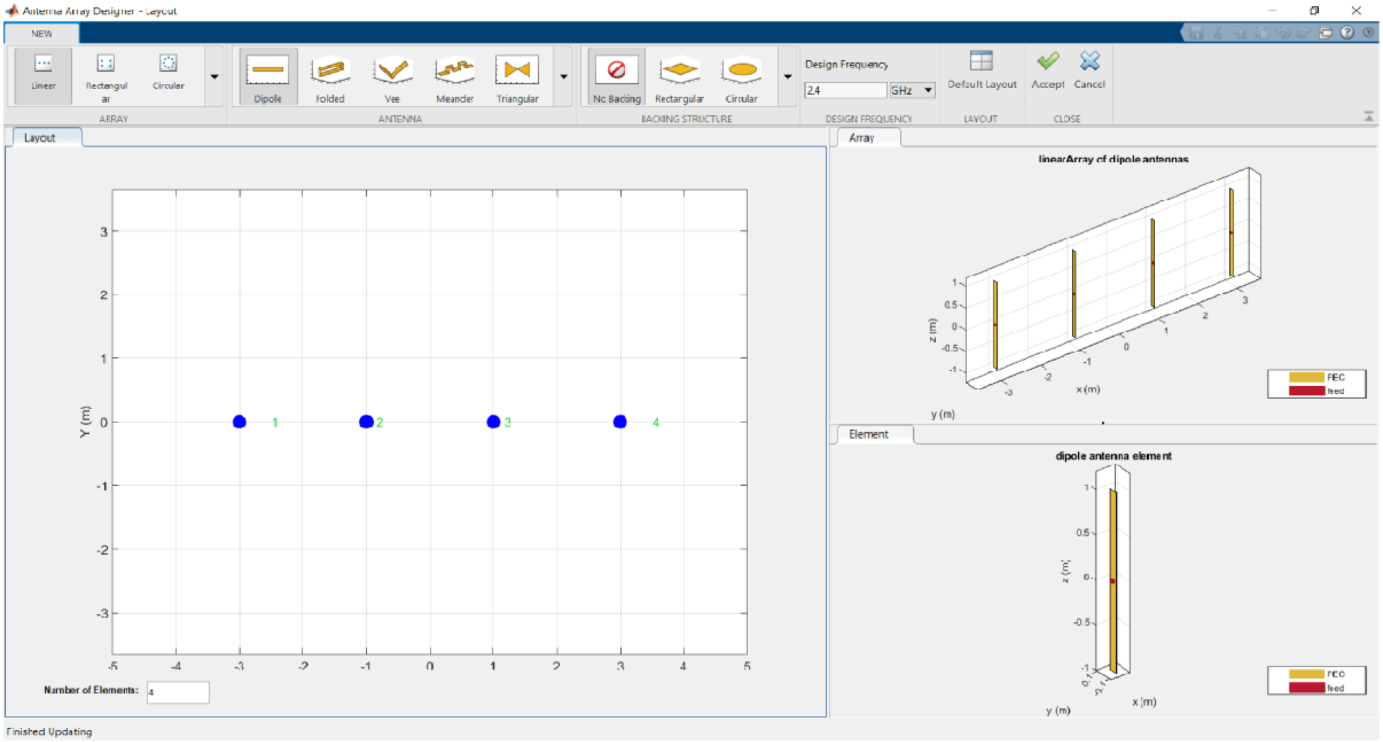
dipole - Load - lumpedElement

```
   Output : Directivity
Frequency : 2.4 GHz
Max value : 10.6 dBi
Min value : -49.1 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Optimizing Linear Dipole Antenna Array Using Surrogate Optimization

Open the **Antenna Array Designer** app. In the Array Gallery section, select the array type as **Linear.**

Select **Dipole** from **Antenna Gallery**. Select **No Backing** under the **Backing Structure Gallery**. Specify the design frequency as 2.4 GHz. In **Layout** pane, specify the **Number of Elements** as 4 and click **Accept** under the **Close** section.

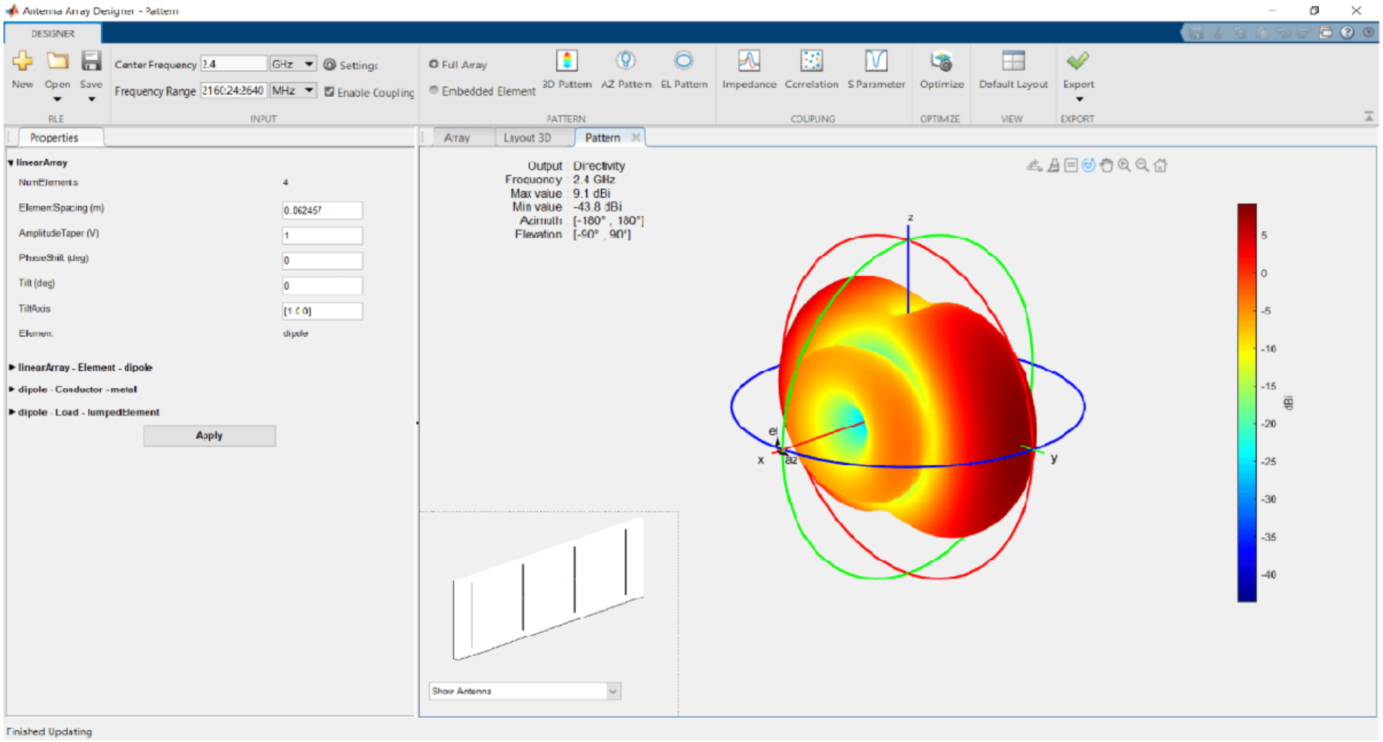


Design Frequency 2.4 GHz

Number of Elements: 4

Select **3D Pattern** under **Pattern** section to calculate the 3-D radiation pattern.

The gain is 9.1 dBi. Click **Optimize** on the app toolstrip to optimize this array.



Center Frequency 2.4 GHz

Frequency Range 2160:24:2640 MHz

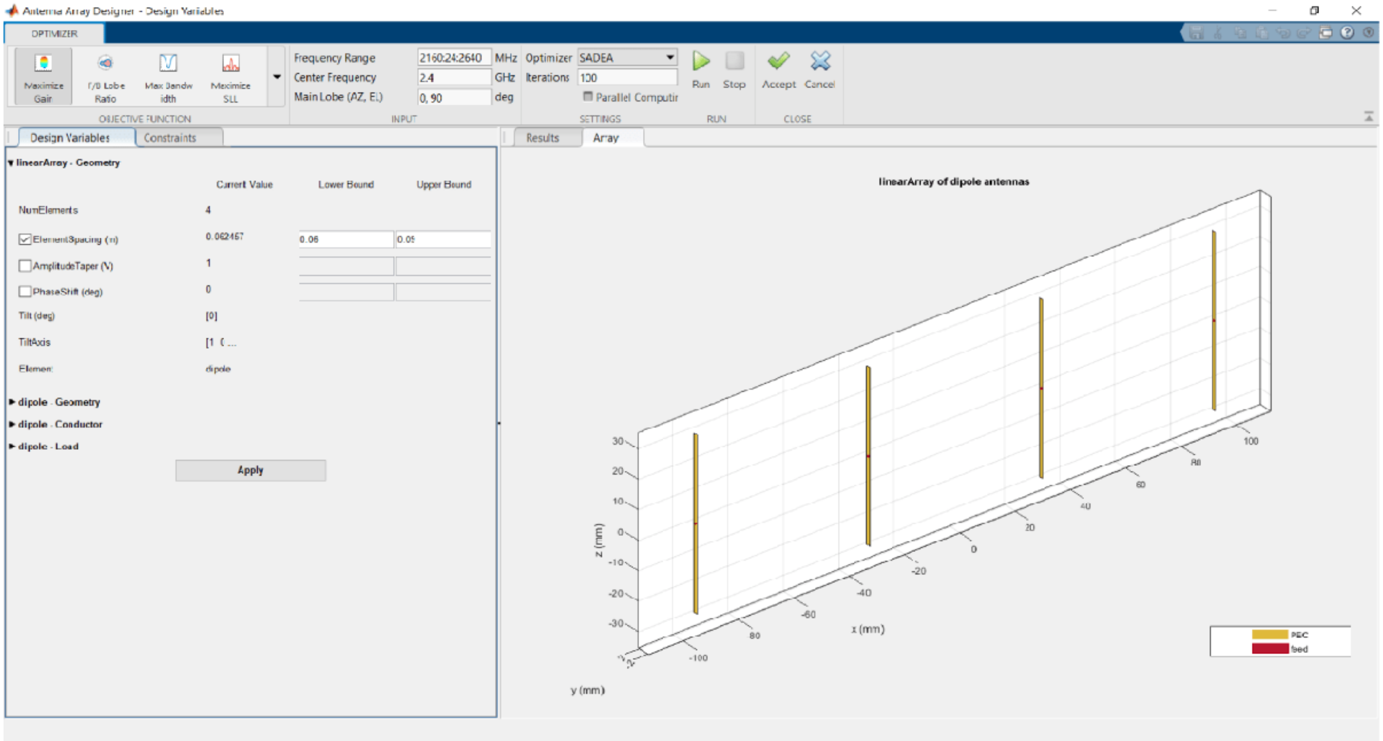
☑ Enable Coupling

▾ linearArray

NumElements	4
ElementSpacing (m)	0.062457
AmplitudeTaper (V)	1
PhaseShift (deg)	0
Tilt (deg)	0
TiltAxis	[1 0 0]
Element	dipole

Output	Directivity
Frequency	2.4 GHz
Max value	9.1 dBi
Min value	-43.8 dBi
Azimuth	[-180°, 180°]
Elevation	[-90°, 90°]

On the Optimizer tab, click **Maximize Gain** in the **Objective Function** section. In the **Design Variables pane**, select the variables you want to optimize. In this example, select the **Element Spacing** variable and set the lower and upper bounds to 0.06 and 0.09.



Frequency Range 2160:24:2640 MHz
Center Frequency 2.4 GHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer SADEA
Iterations 100
□ Parallel Computir

▾ linearArray - Geometry

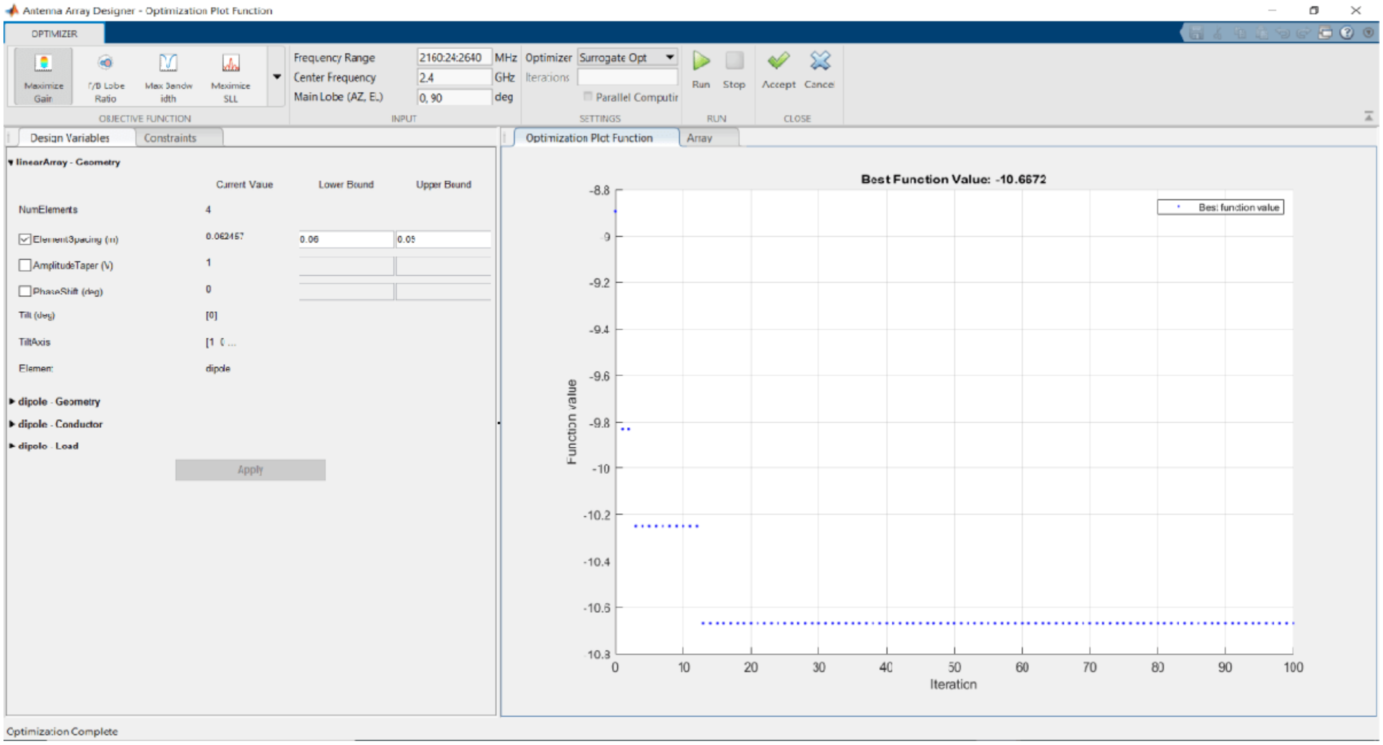
	Current Value	Lower Bound	Upper Bound
NumElements	4		
☑ ElementSpacing (m)	0.062457	0.06	0.09
□ AmplitudeTaper (V)	1		
□ PhaseShift (deg)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		
Element	dipole		

▸ dipole - Geometry
▸ dipole - Conductor
▸ dipole - Load

Click the **Constraints** pane. In this example, there are no **Constraint Functions**. If your application requires constraints, chose one or more constraint functions from the dropdown.

Click **Apply** to apply the design variables in this example. In the **Settings** section, from the **Optimizer** drop down select `Surrogate Opt`. **Parallel Computing** is greyed out.

Once the simulation is complete, the optimization results are displayed in the **Results** pane.



Frequency Range 2160:24:2640 MHz
Center Frequency 2.4 GHz
Main Lobe (AZ, EL) 0, 90 deg
Optimizer Surrogate Opt

linearArray - Geometry

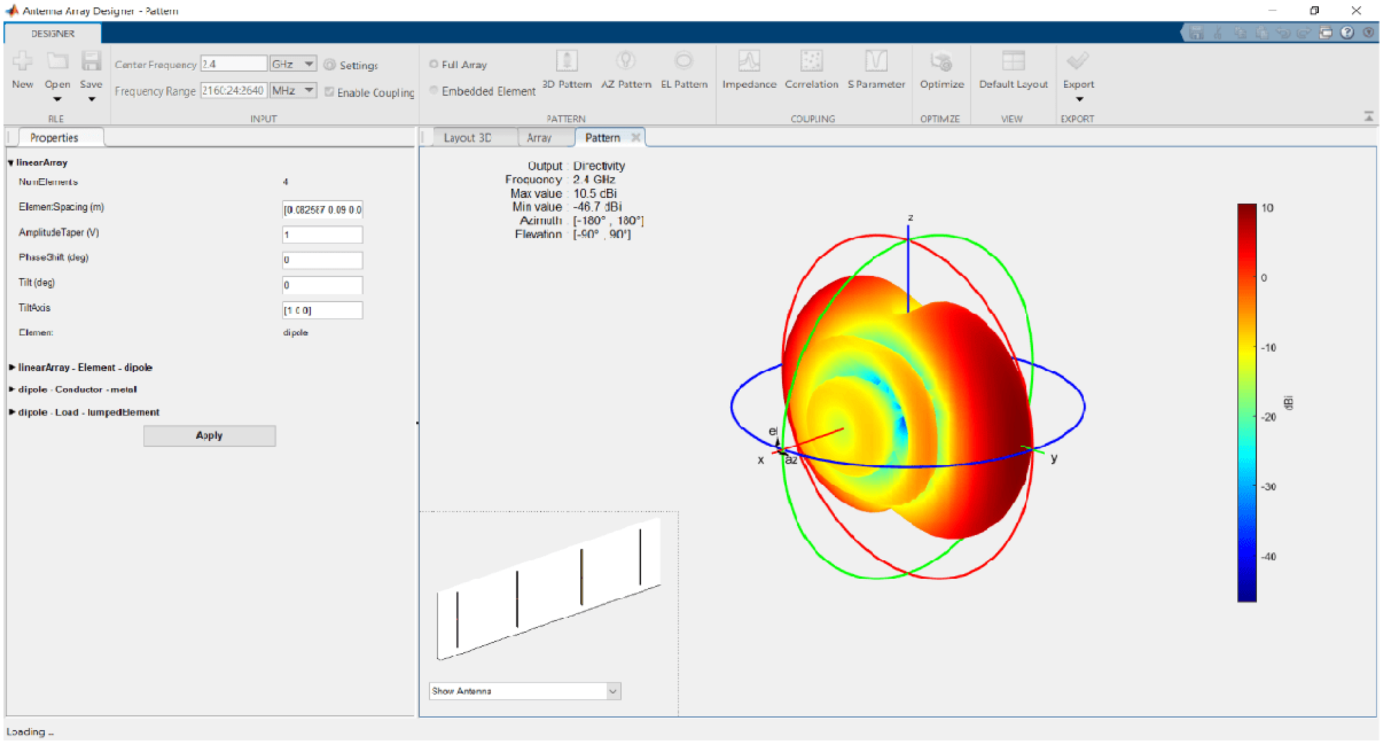
	Current Value	Lower Bound	Upper Bound
NumElements	4		
☑ ElementSpacing (m)	0.062457	0.06	0.09
□ AmplitudeTaper (V)	1		
□ PhaseShift (deg)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		
Element	dipole		

dipole - Geometry
dipole - Conductor
dipole - Load

Best Function Value: -10.6672

Optimization Complete

Click **Accept**. In the **Pattern** section, plot the **3D Pattern** again. The gain has now increased to 10.5 dBi.



Center Frequency 2.4 GHz
Frequency Range 2160:24:2640 MHz

linearArray

NumElements	4
ElementSpacing (m)	[0.082587 0.09 0.0
AmplitudeTaper (V)	1
PhaseShift (deg)	0
Tilt (deg)	0
TiltAxis	[1 0 0]
Element	dipole

linearArray - Element - dipole
dipole - Conductor - metal
dipole - Load - lumpedElement

Output Directivity
Frequency 2.4 GHz
Max value 10.5 dBi
Min value -46.7 dBi
Azimuth [-180°, 180°]
Elevation [-90°, 90°]

Loading ...

- “Design and Analysis Using Antenna Array Designer”
- “Optimization of Antenna Array Elements Using Antenna Array Designer App”

Programmatic Use

`antennaArrayDesigner` opens the **Array Designer** app, enabling you to design and analyze antenna arrays using the Antenna Toolbox library.

Version History

Introduced in R2019b

See Also

Topics

“Design and Analysis Using Antenna Array Designer”
“Optimization of Antenna Array Elements Using Antenna Array Designer App”
“Antenna Optimization Algorithm”

PCB Antenna Designer

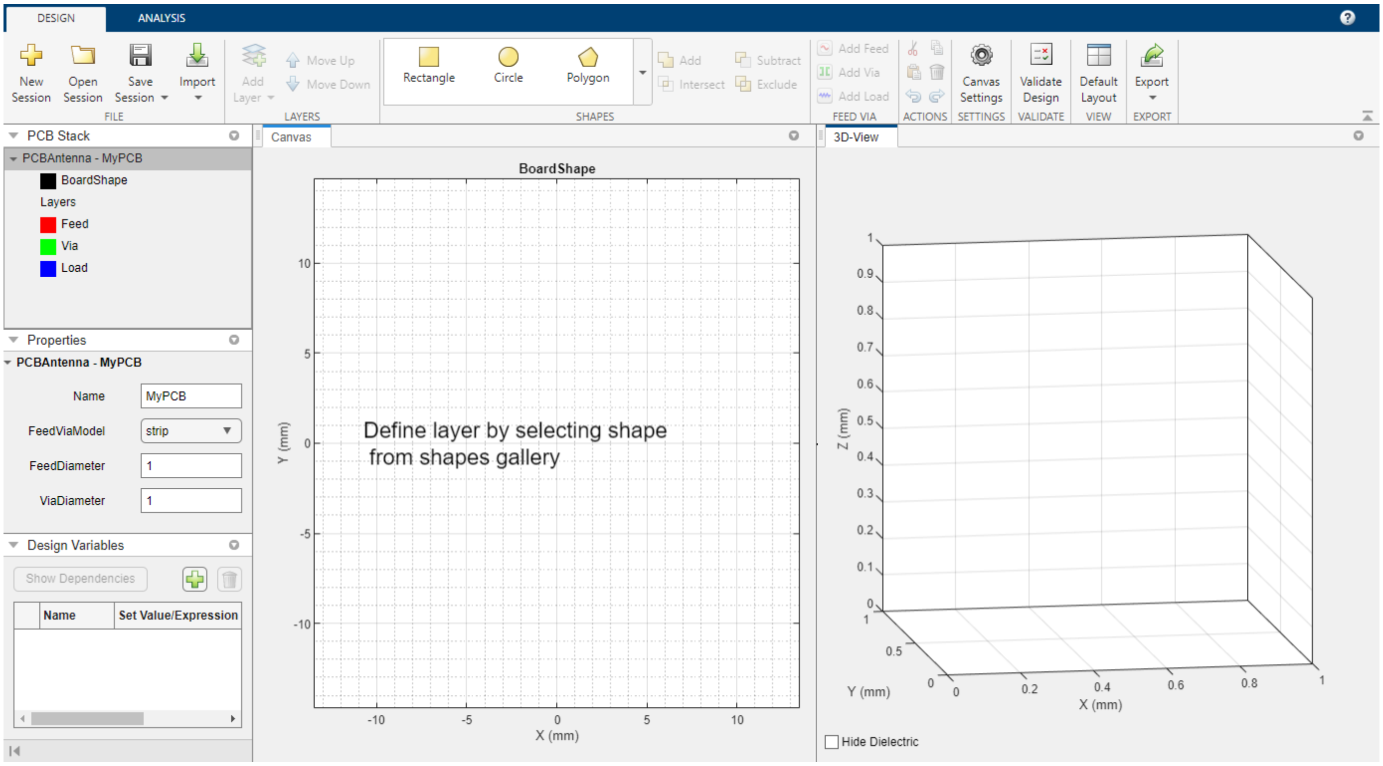
Design, analyze, optimize, and export single or multifeed PCB antennas

Description

The **PCB Antenna Designer** app lets you design and visualize single or multifeed PCB antennas.

Using this app, you can:

- Create single-layer, multilayer metal, or metal-dielectric substrate PCB antennas
- Create an arbitrary number of feeds and vias in a PCB antenna
- Create shapes and perform Boolean operations
- Validate PCB antenna design
- Perform vector frequency analysis: impedance and S-parameters over a frequency range
- Perform scalar frequency analysis: current distribution, 3-D pattern and azimuth and elevation patterns
- Create variables for the object properties and optimize the design using these variables
- Generate mesh and estimate memory requirements
- Export your design to MATLAB® work space or script or, as Gerber files



Name: MyPCB

FeedViaModel: strip

FeedDiameter: 1

ViaDiameter: 1

Define layer by selecting shape from shapes gallery

Open the PCB Antenna Designer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the **PCB Antenna Designer** app icon.
- MATLAB command prompt: Enter `pcbAntennaDesigner`.

Examples

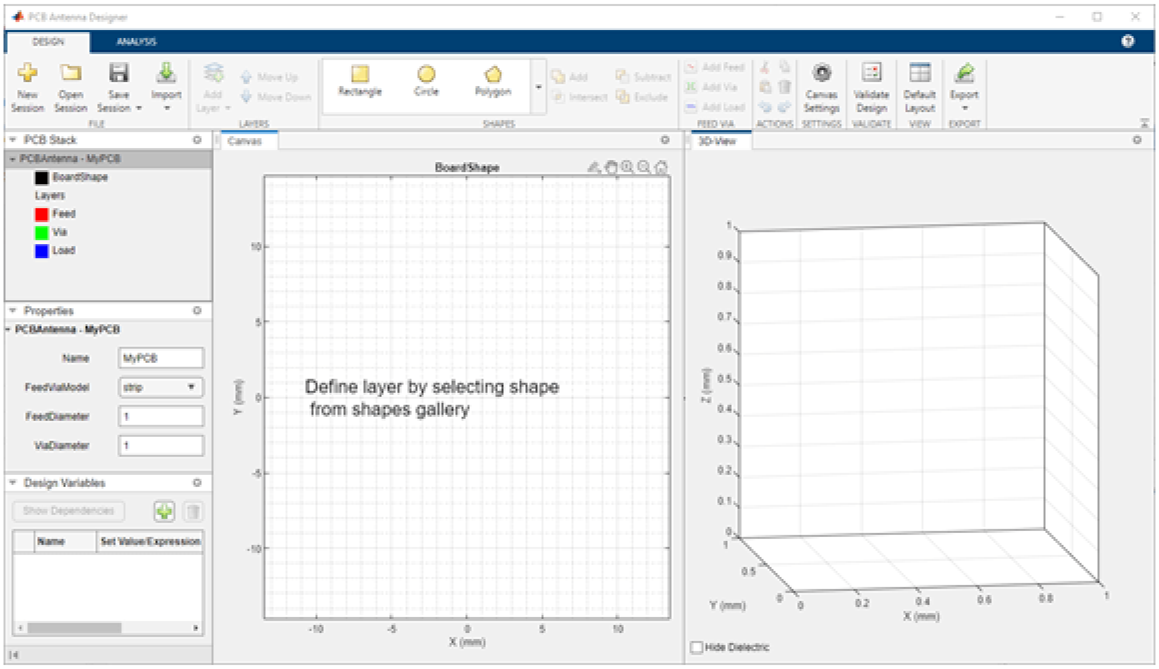
Design and Analyze X-Band Custom PCB Patch Antenna

Type this command at the command line to open the **PCB Antenna Designer** app.

```
pcbAntennaDesigner
```

On the **Design** tab click **New Session** to start a new session and open a blank canvas.

The default units for the canvas are in millimeters (mm). The app uses global units and you can change the units using the **Canvas Settings** button.



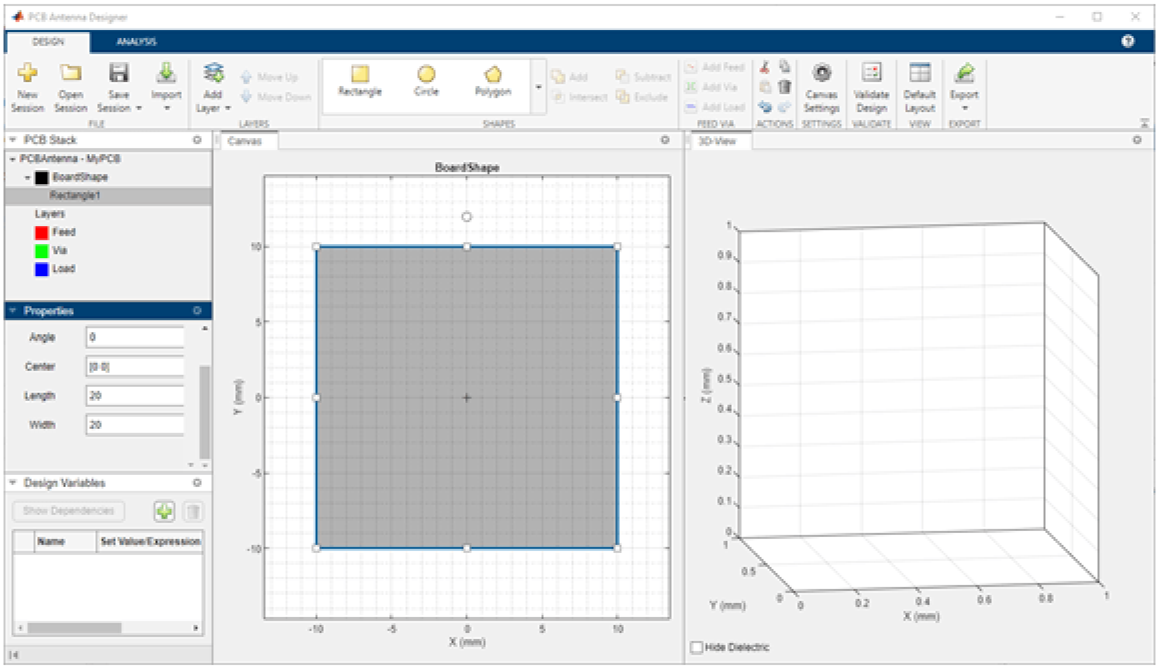
Name MyPCB
FeedViaModel strip
FeedDiameter 1
ViaDiameter 1

Define layer by selecting shape from shapes gallery

Define Board Shape

Select **Rectangle** from the **Shapes** section on the toolbar. Drag the shape on the canvas to create a rectangle and change the properties of `Rectangle1` in the **Properties** pane to the following:

- **Center** — `[0,0]`
- **Length** — `20`
- **Width** — `20`

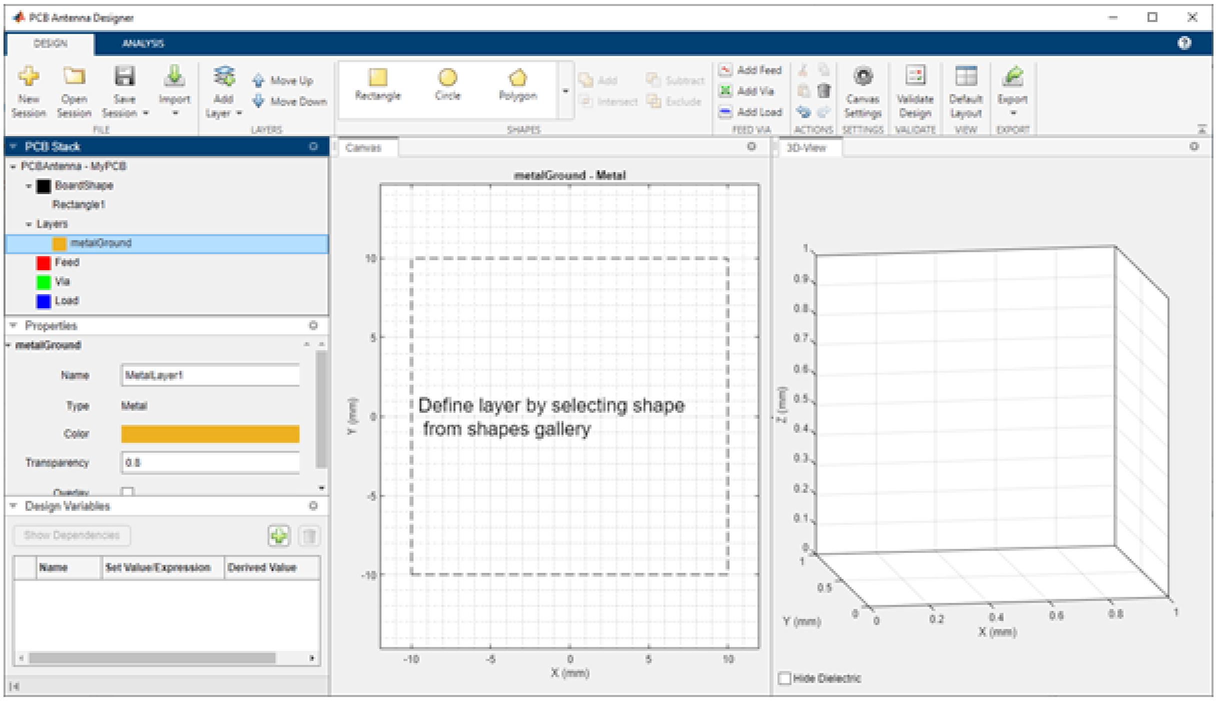


Angle 0
Center [0 0]
Length 20
Width 20

Add Ground Plane

Click **Add Layer** on the toolbar and then select **Metal Layer**. The shape of the board is represented as a dotted line on the canvas. Rename this layer to metalGround and then change the default color to [0.93 0.69 0.13].

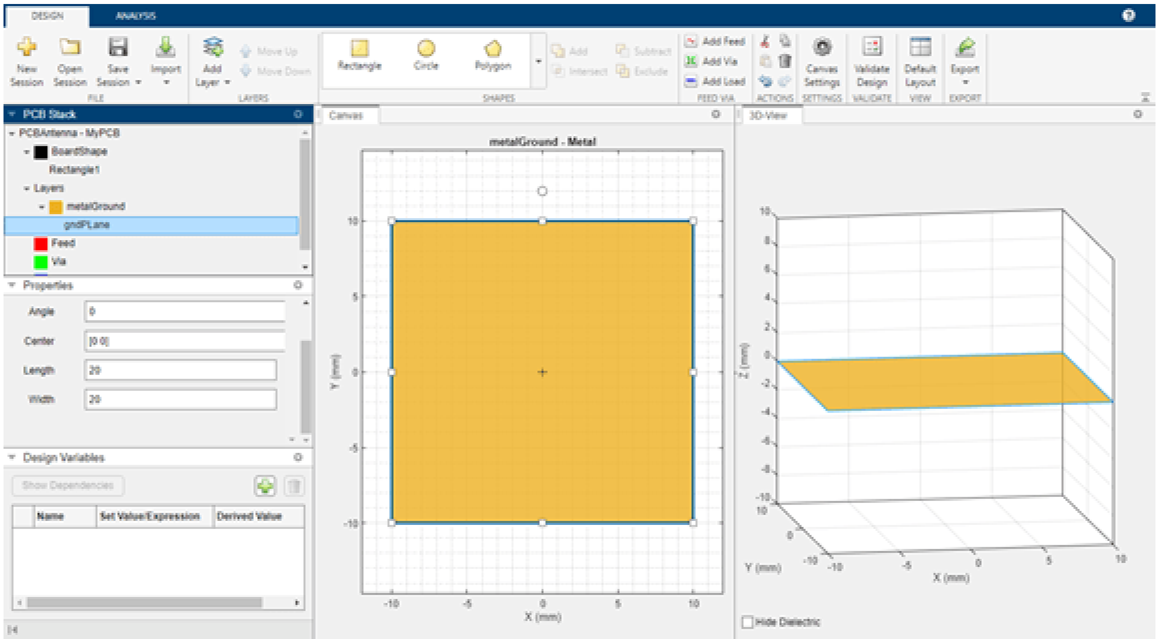
The default metal type is PEC. To change the metal type, select Layers and then select a metal from the **Type** drop-down list. Use the default metal type for this example.



Name MetalLayer1
Type Metal
Transparency 0.8

Select **Rectangle** from the **Shapes** section and drag the shape on the canvas to create a rectangular ground plane. Set the properties of the ground plane to the following

- **Name** — `gndplane`
- **Center** — `[0,0]`
- **Length** — `20`
- **Width** — `20`



Angle 0

Center [0 0]

Length 20

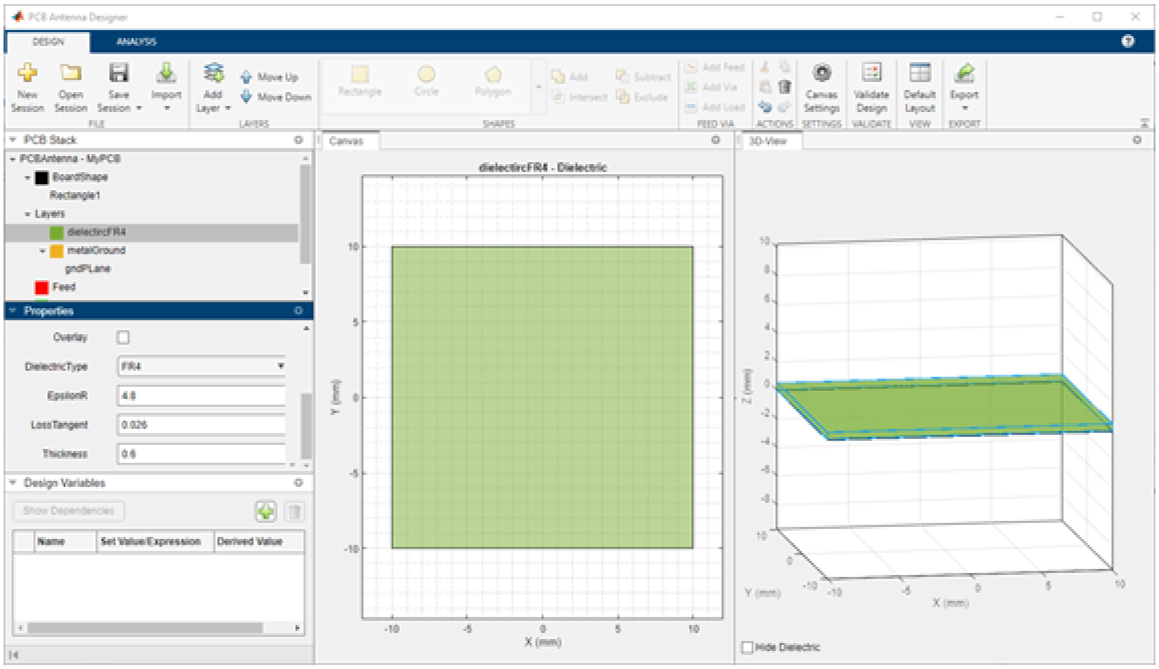
Width 20

Name	Set Value/Expression	Derived Value

Add Dielectric Layer

Click **Add Layer** and then select **Dielectric Layer** to create a dielectric layer. A dielectric layer is created with the dimensions of the board shape. Set the properties of the layer to the following:

- **Name** — `dielectricFR4`
- **DielectricType** — FR4
- **Color** — `[0.47 0.67 0.19]`
- **Transparency** — `0.5`
- **Thickness** — `0.6`



Overlay □

DielectricType FR4

EpsilonR 4.8

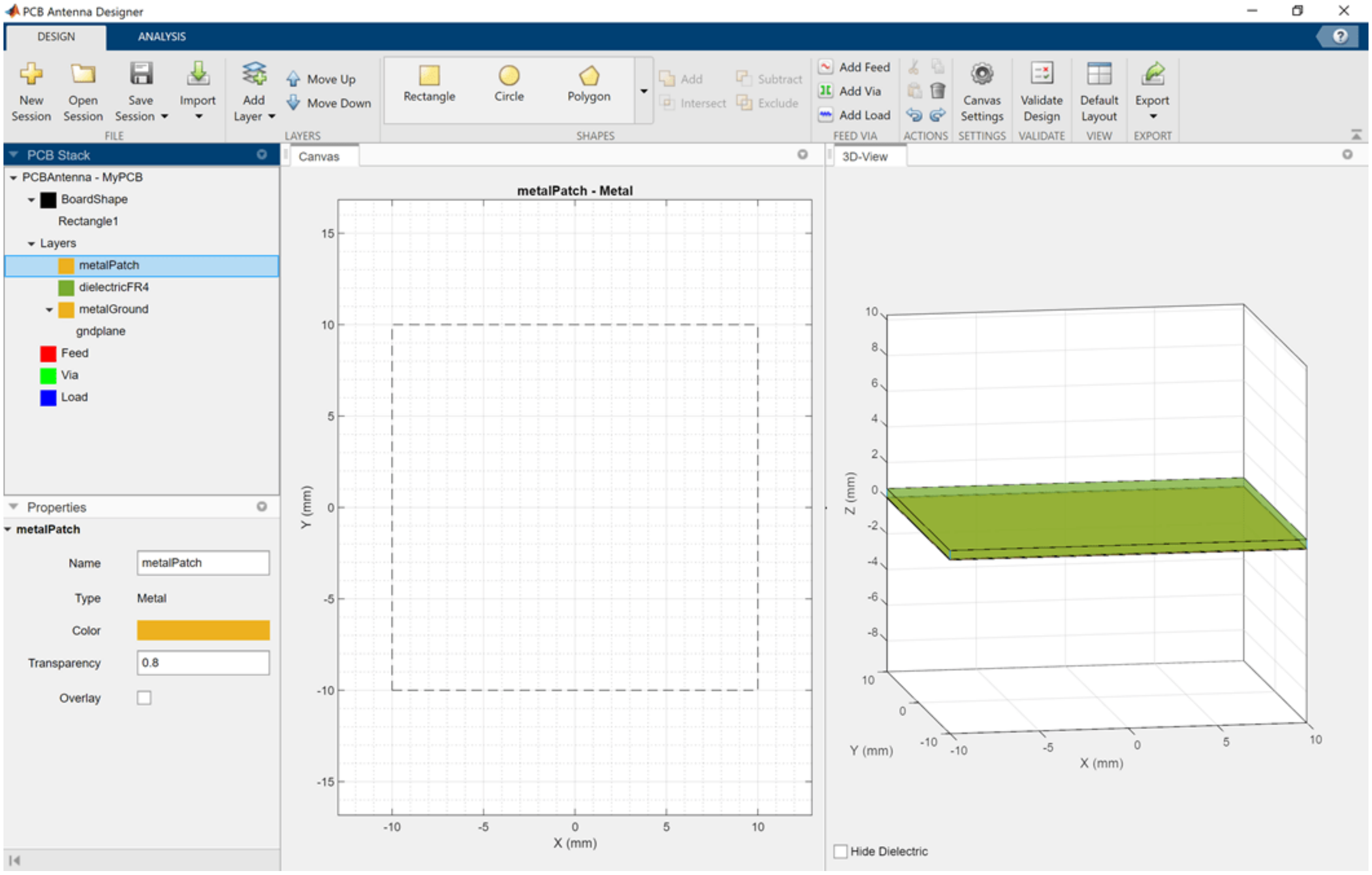
LossTangent 0.026

Thickness 0.6

□ Hide Dielectric

Create Polygon Patch

Click **Add Layer** and then select **Metal Layer**. Rename the metal layer to `metalPatch`.



metalPatch

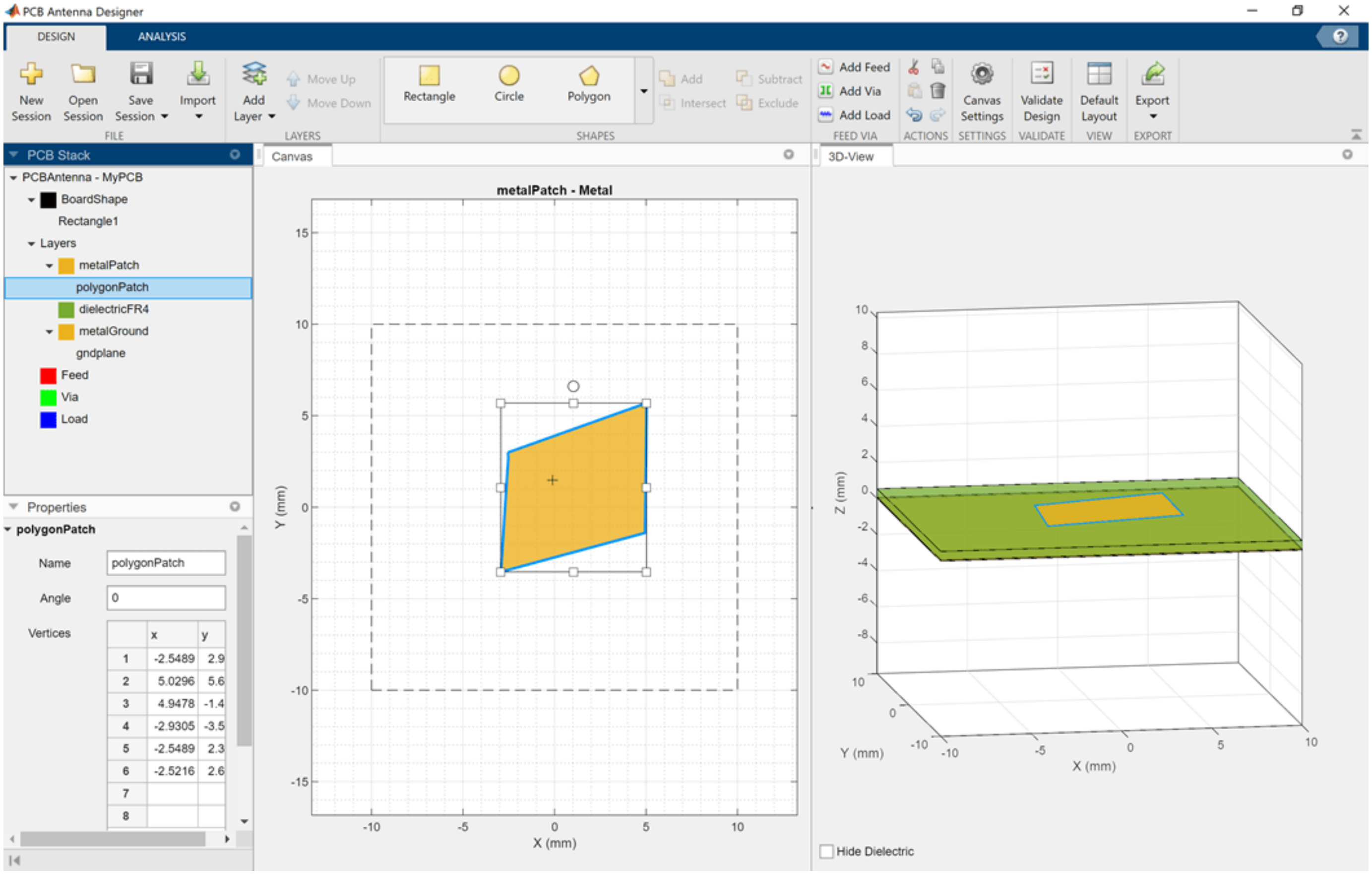
Name	metalPatch
Type	Metal
Color	🟧
Transparency	0.8
Overlay	□

Select **Polygon** from the **Shapes** section. Select the vertices on the canvas to create a polygon patch. Set the properties of the polygon to the following:

- **Name** — `polygonPatch`
- **Angle** — `0`

Under **Vertices** set the following values:

- `x1 = –2.5489 and y1 = 2.9902,`
- `x2 = 5.0296 and y2 = 5.6928,`
- `x3 = 4.9478 and y3 = –1.4050,`
- `x4 = –2.9305 and y4 = –3.5343,`
- `x5 = –2.5489 and y5 = 2.3623 and`
- `x6 = –2.5216 and y1 = 2.6899`



polygonPatch

Name: polygonPatch

Angle: 0

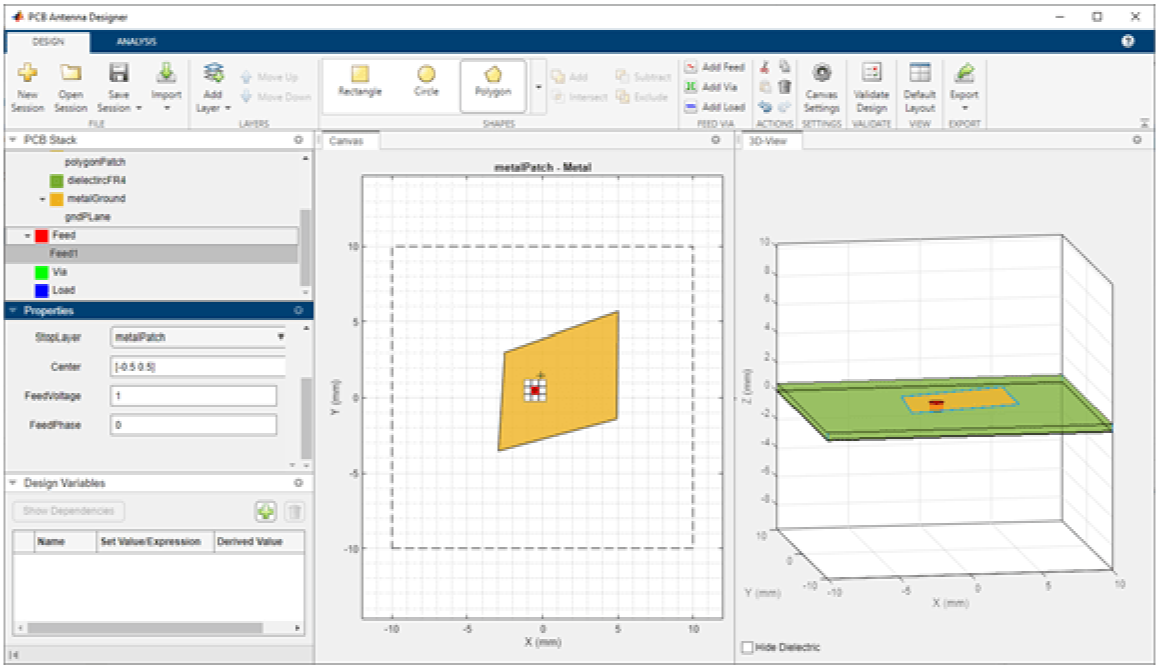
Vertices

	x	y
1	-2.5489	2.9
2	5.0296	5.6
3	4.9478	-1.4
4	-2.9305	-3.5
5	-2.5489	2.3
6	-2.5216	2.6
7		
8		

Add Feed

Click **Add Feed** from the **Feed Via** section on the toolbar. Set the properties of the feed to the following:

- **Name** — Feed1
- **StartLayer** — metalGround
- **StopLayer** — metalPatch
- **Center** — [- 0.5 0.5]



StopLayer: metalPatch

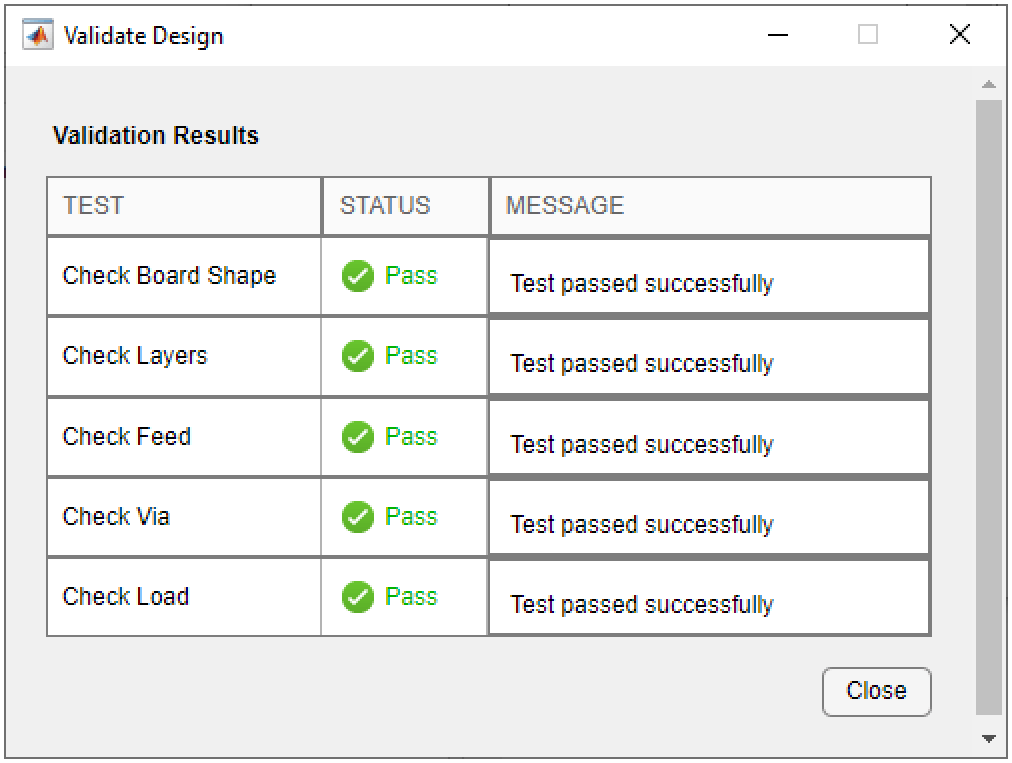
Center: [-0.5 0.5]

FeedVoltage: 1

FeedPhase: 0

Validate Your Design

Click **Validate Design** on the toolbar to validate your board shape, layers, feed, via, and load.

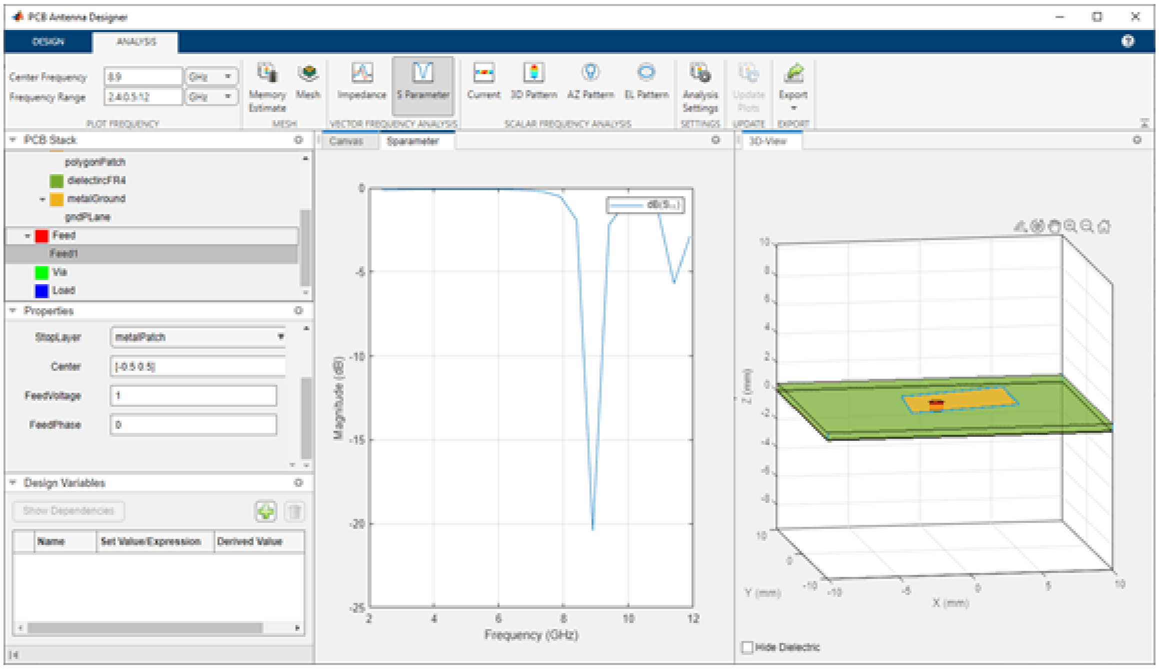


Validation Results

TEST	STATUS	MESSAGE
Check Board Shape	✓ Pass	Test passed successfully
Check Layers	✓ Pass	Test passed successfully
Check Feed	✓ Pass	Test passed successfully
Check Via	✓ Pass	Test passed successfully
Check Load	✓ Pass	Test passed successfully

Analyze Your Design

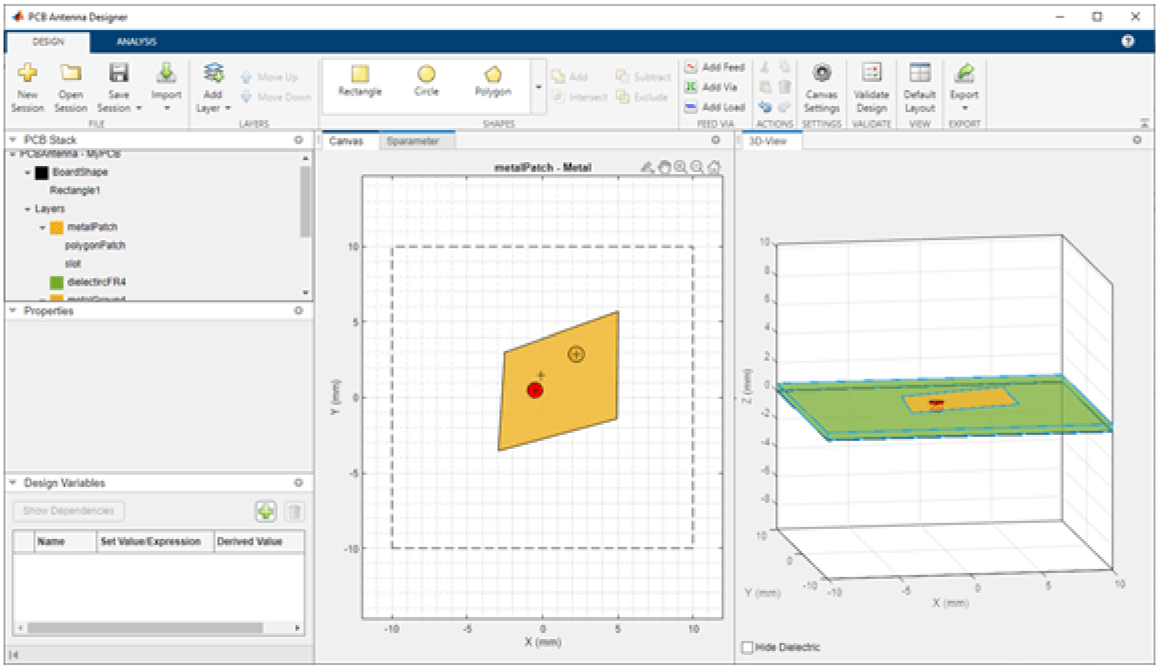
In the **Analysis** tab, set **Center Frequency** to `8.9` GHz and **Frequency Range** to `2:4:0.5:12`. Click **S-parameters** under the **Vector Frequency Analysis** section to plot the reflection coefficient of the PCB patch antenna. The PCB patch antenna resonates at 8.9 GHz.


Center Frequency 8.9 GHz
Frequency Range 2:4:0.5:12 GHz

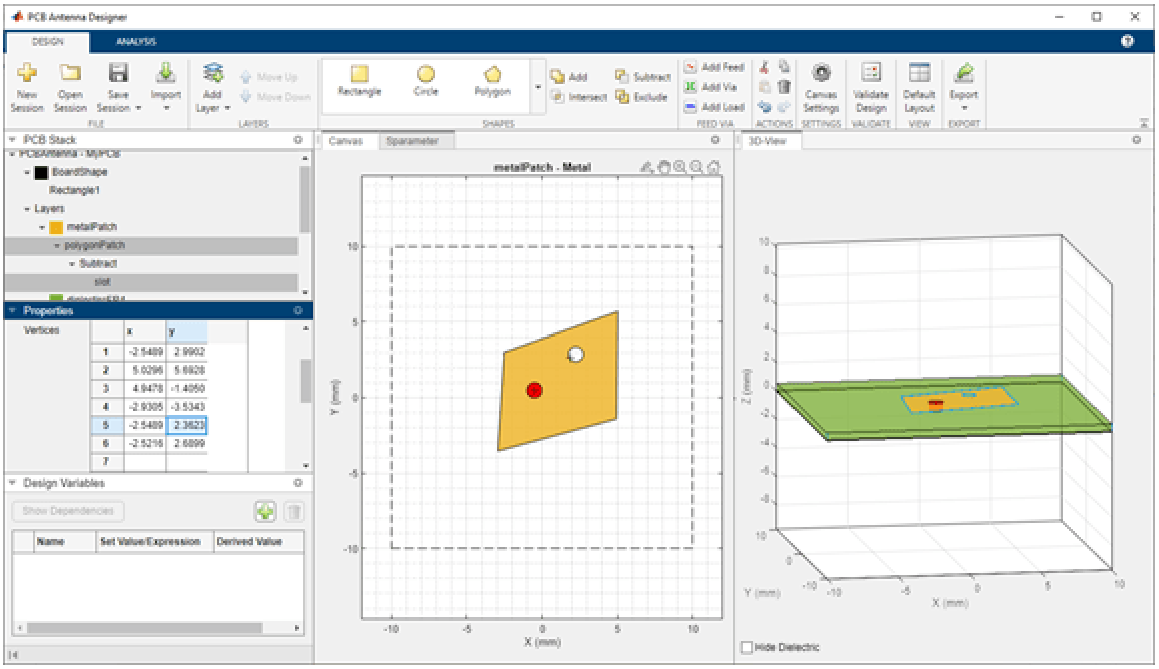
StopLayer metalPatch
Center [-0.5 0.5]
FeedVoltage 1
FeedPhase 0

This examples employs adding a slot to shift the resonant frequency. To add a circular slot, in the **Design** tab, select `metalPatch` and then select the **Circle** in the `Shapes`. Set the properties of the circle to the following:

- **Name** — `slot`
- **Center** — `[2.24675 2.88105]`
- **Radius** — `0.5`



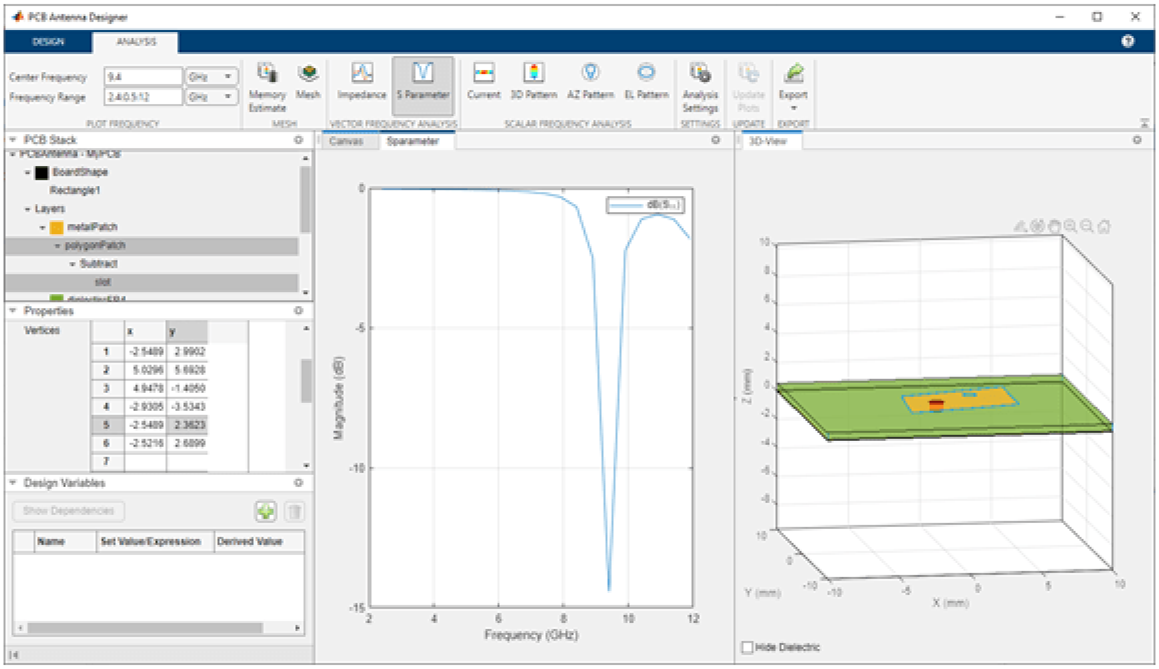
Select the `polygonPatch` and the `slot` you created in the previous step and then select **Subtract** on the toolbar.



Vertices

	x	y
1	-2.5489	2.9902
2	5.0296	5.6928
3	4.9478	-1.4050
4	-2.9305	-3.5343
5	-2.5489	2.3623
6	-2.5216	2.6899
7		

In the **Analysis** tab, set **Center Frequency** to `9.4` GHz and click **Update Plots**. The PCB patch antenna now resonates at 9.4 GHz.



PCB Antenna Designer

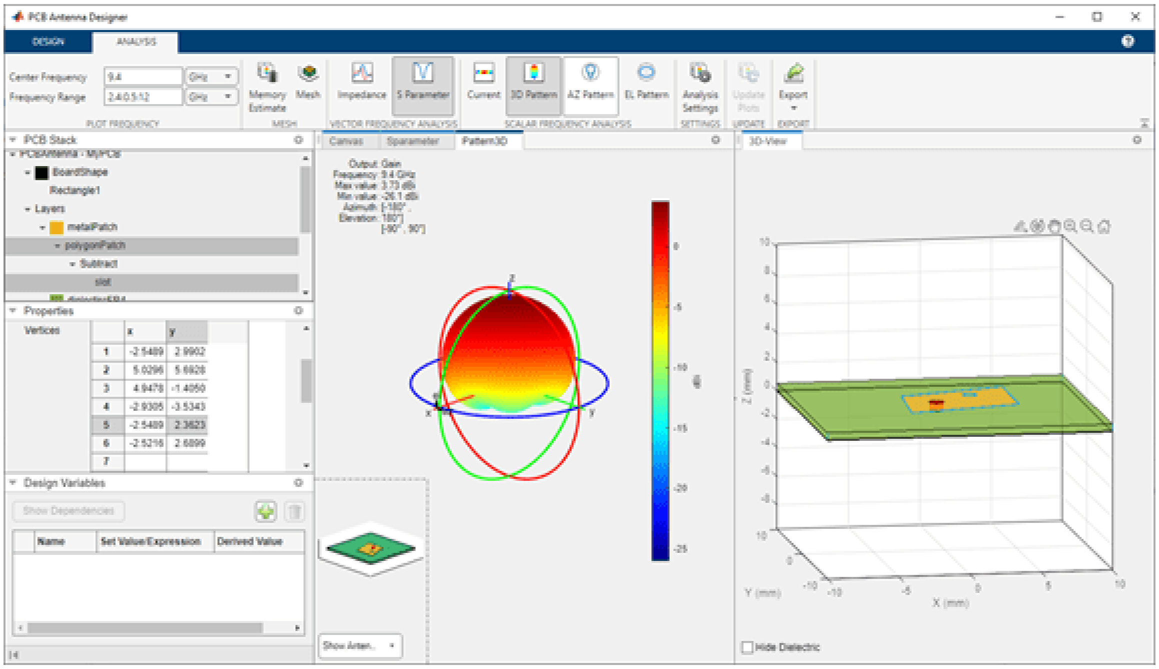
Center Frequency 9.4 GHz

Frequency Range 2:0.5:12 GHz

Vertices

	x	y
1	-2.5489	2.9902
2	5.0296	5.6828
3	4.9478	-1.4050
4	-2.9305	-3.5343
5	-2.5489	2.3623
6	-2.5216	2.6899
7		

Click **3D Pattern** in the **Scalar Frequency Analysis** section to plot the 3-D radiation pattern. The directivity of the PCB patch antenna is 3.69 dBi.



PCB Antenna Designer

Center Frequency 9.4 GHz

Frequency Range 2:0.5:12 GHz

Output: Gain
Frequency: 9.4 GHz
Max value: 3.73 dBi
Min value: -26.1 dBi
Azimuth: [-180°, 180°]
Elevation: [-90°, 90°]

Vertices

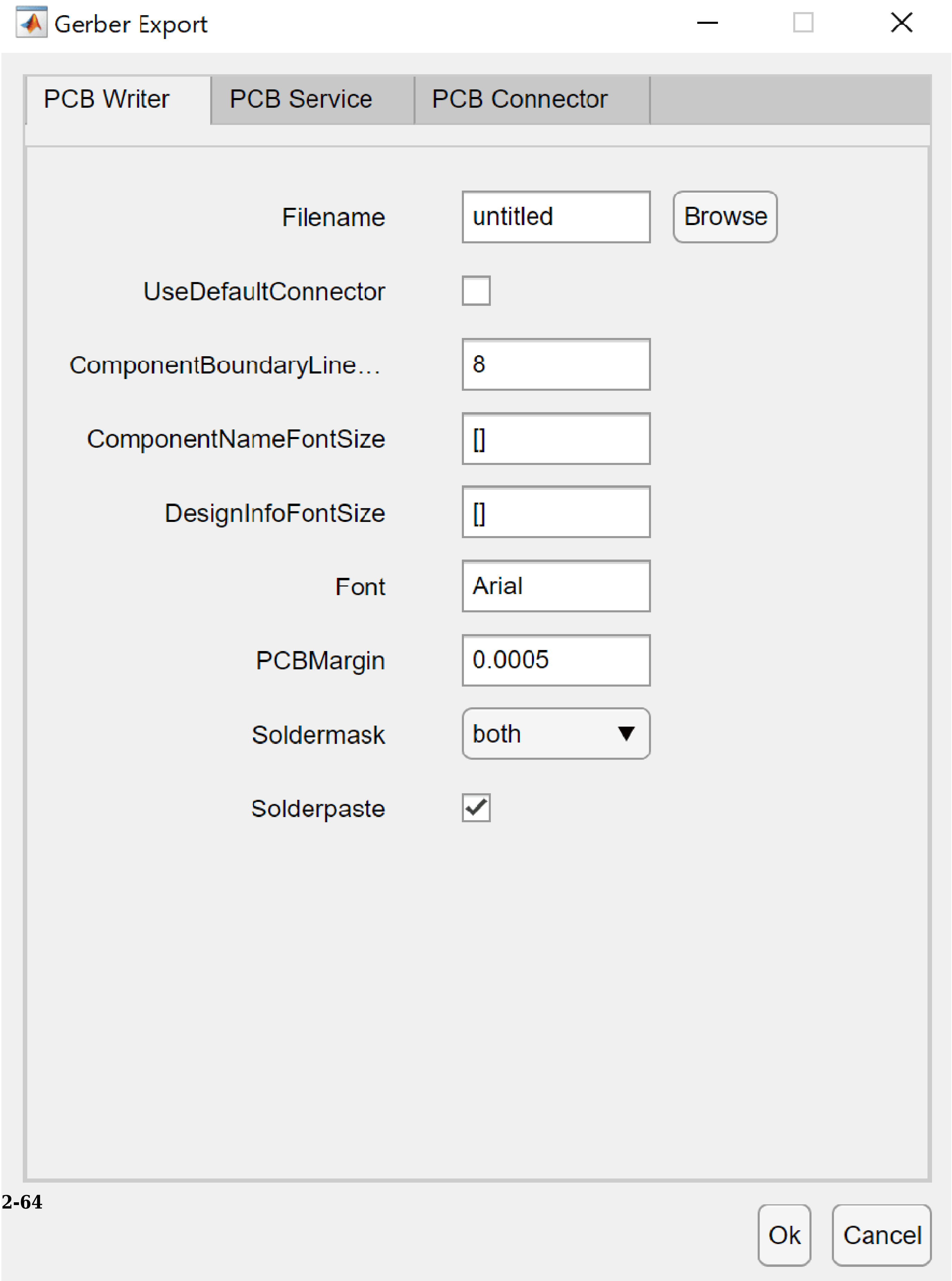
	x	y
1	-2.5489	2.9902
2	5.0296	5.6828
3	4.9478	-1.4050
4	-2.9305	-3.5343
5	-2.5489	2.3623
6	-2.5216	2.6899
7		

Export PCB Patch Antenna to Gerber Files

This example uses a PCB patch antenna design from the “Design and Analyze X-Band Custom PCB Patch Antenna” on page 2-53 example.

Follow the Design and Analyze X-Band Custom PCB Patch Antenna example to design a PCB patch antenna.

In the **Analysis** tab, click **Export** and then select **Export as Gerber File**. In the Gerber Export dialog, select **Browse** in the **PCB Writer** tab to select the directory in which you want the Gerber files to be stored. Clear the **UseDefaultConnector**.



Gerber Export

Filename: untitled

UseDefaultConnector: □

ComponentBoundaryLine...: 8

ComponentNameFontSize: []

DesignInfoFontSize: []

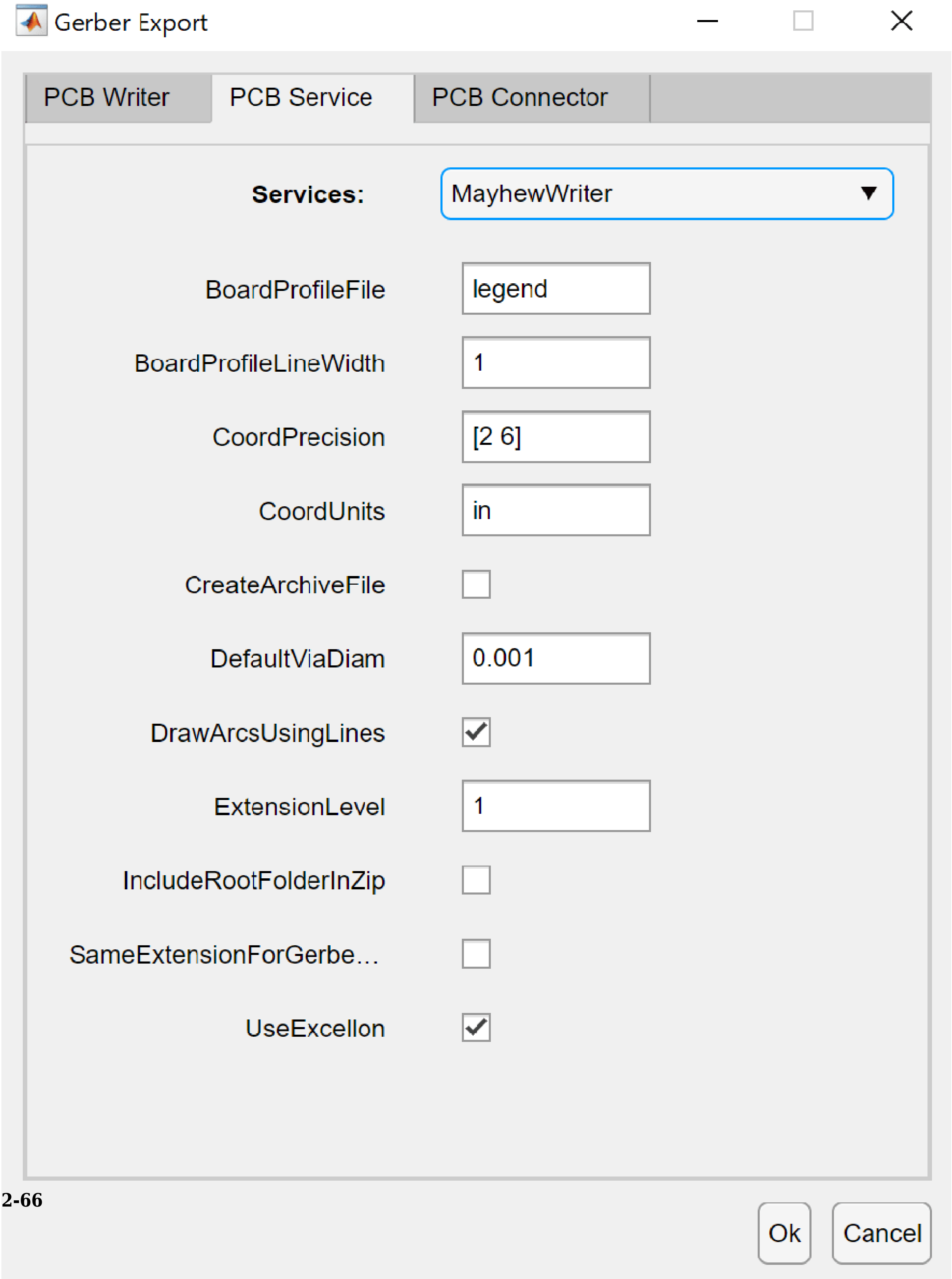
Font: Arial

PCBMargin: 0.0005

Soldermask: both ▼

Solderpaste: ☑

In the **PCB Services** tab, under **Services**, select `MayhewWriter`. The PCB writer uses the selected service to create the Gerber files.



Services: MayhewWriter ▼

BoardProfileFile: legend

BoardProfileLineWidth: 1

CoordPrecision: [2 6]

CoordUnits: in

CreateArchiveFile ☐

DefaultViaDiam: 0.001

DrawArcsUsingLines ☑

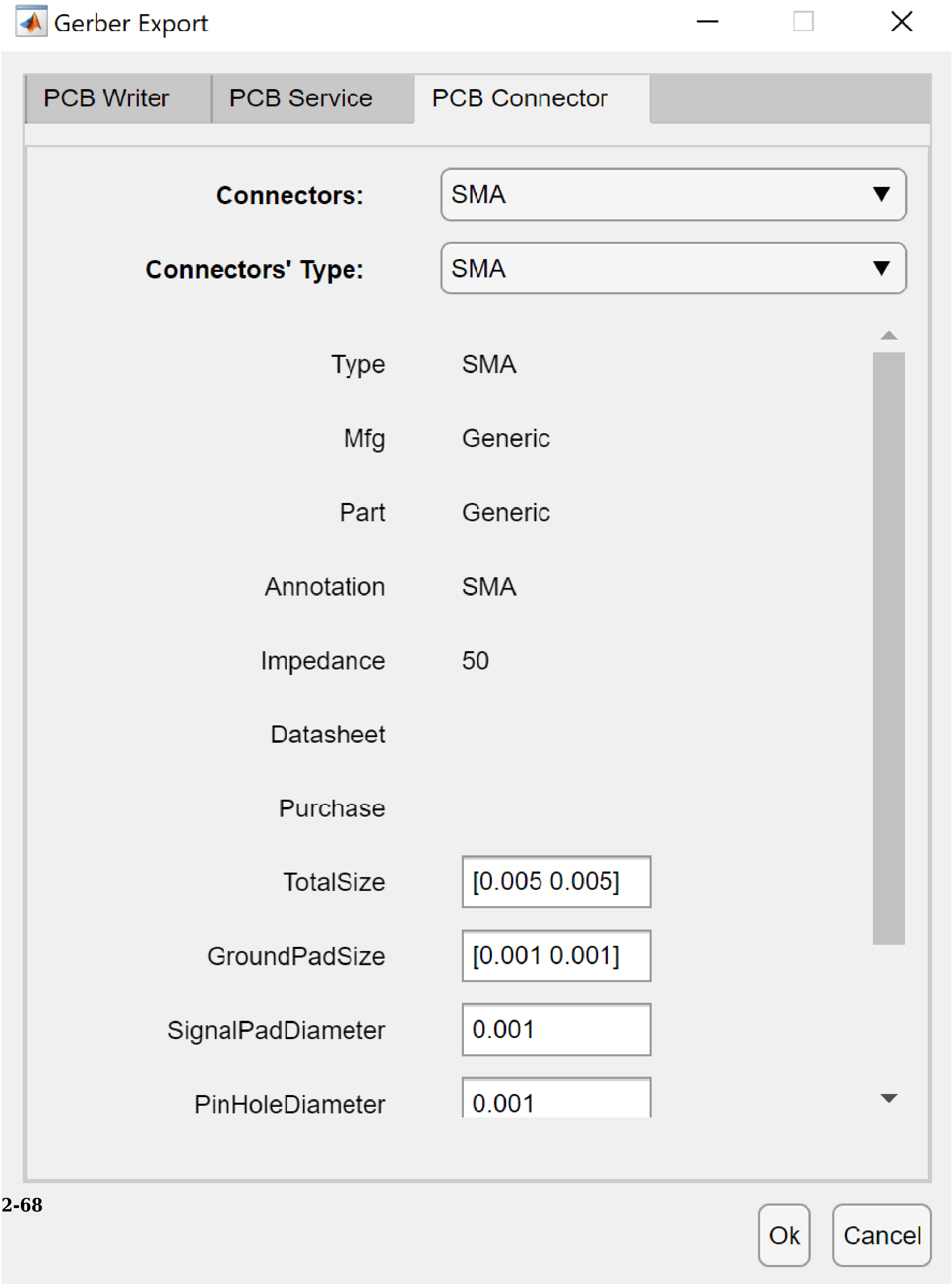
ExtensionLevel: 1

IncludeRootFolderInZip ☐

SameExtensionForGerbe... ☐

UseExcellon ☑

In the **PCB Connector** tab, set **Connectors** and **Connectors' Type** to `SMA`.



Connectors: SMA

Connectors' Type: SMA

Type SMA

Mfg Generic

Part Generic

Annotation SMA

Impedance 50

Datasheet

Purchase

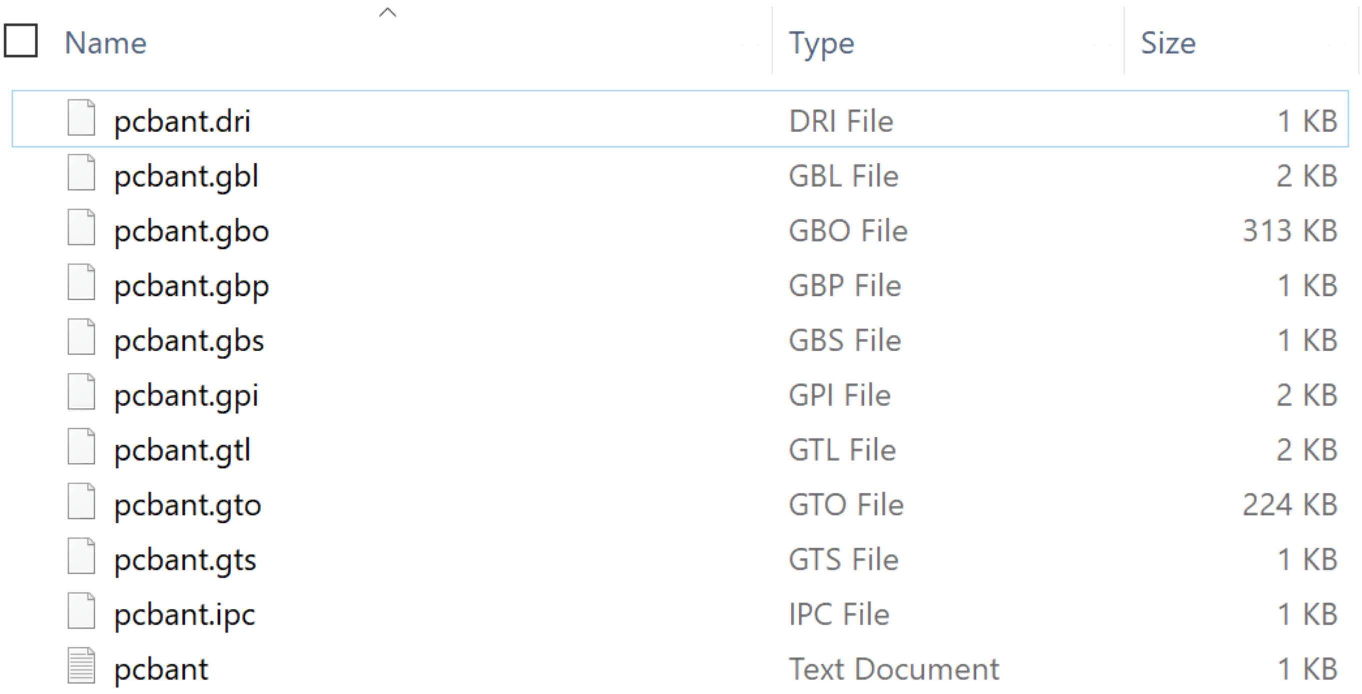
TotalSize [0.005 0.005]

GroundPadSize [0.001 0.001]

SignalPadDiameter 0.001

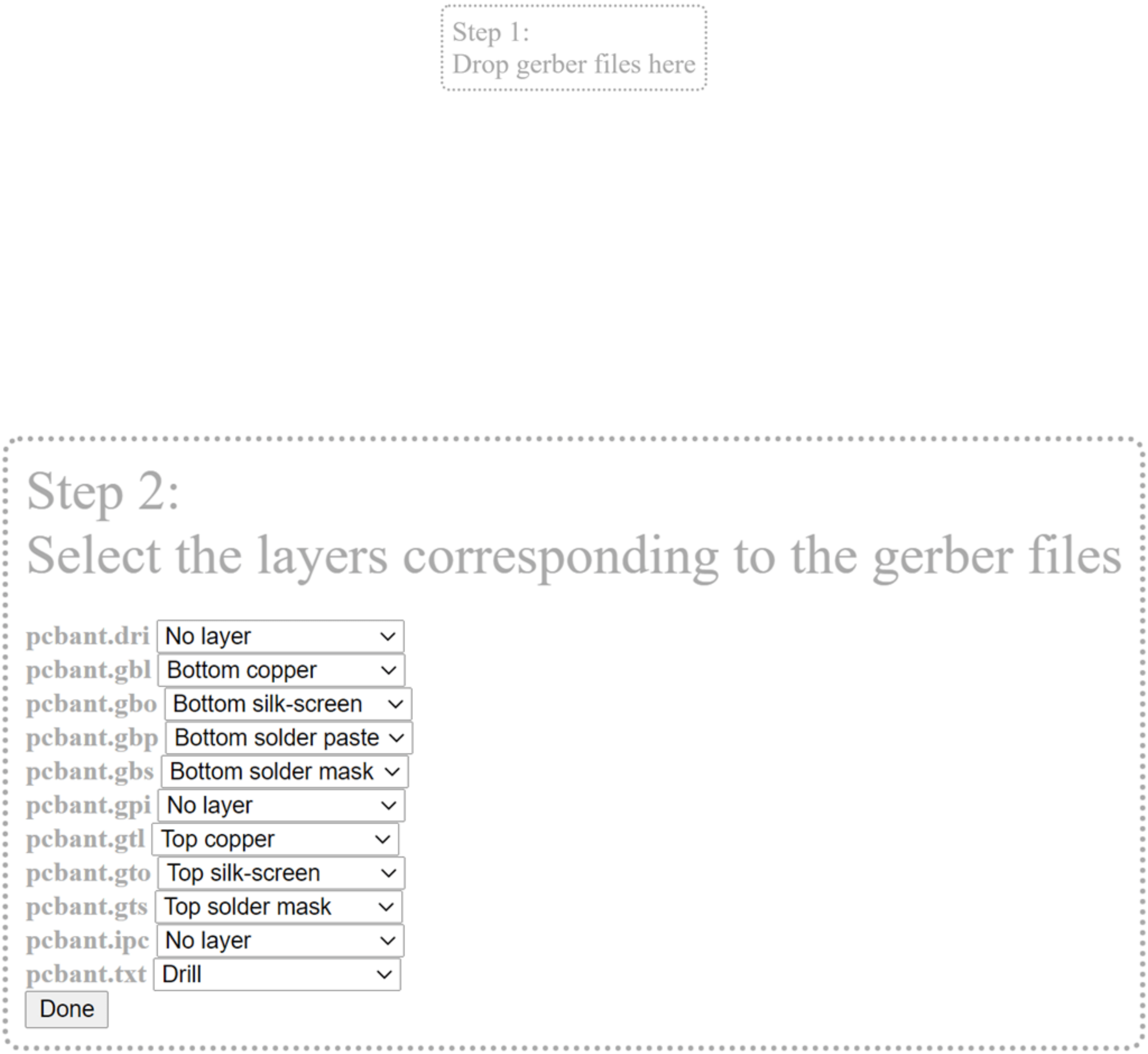
PinHoleDiameter 0.001

Select **OK** to generate the Gerber files. The generated Gerber files are stored in the directory you selected in the **PCB Writer** tab.



Name	Type	Size
pcbant.dri	DRI File	1 KB
pcbant.gbl	GBL File	2 KB
pcbant.gbo	GBO File	313 KB
pcbant.gbp	GBP File	1 KB
pcbant.gbs	GBS File	1 KB
pcbant.gpi	GPI File	2 KB
pcbant.gtl	GTL File	2 KB
pcbant.gto	GTO File	224 KB
pcbant.gts	GTS File	1 KB
pcbant.ipc	IPC File	1 KB
pcbant	Text Document	1 KB

Open *mayhewlabs.com/webGerber* from your browser. Drag and drop the Gerber files from the directory and select **Done**.

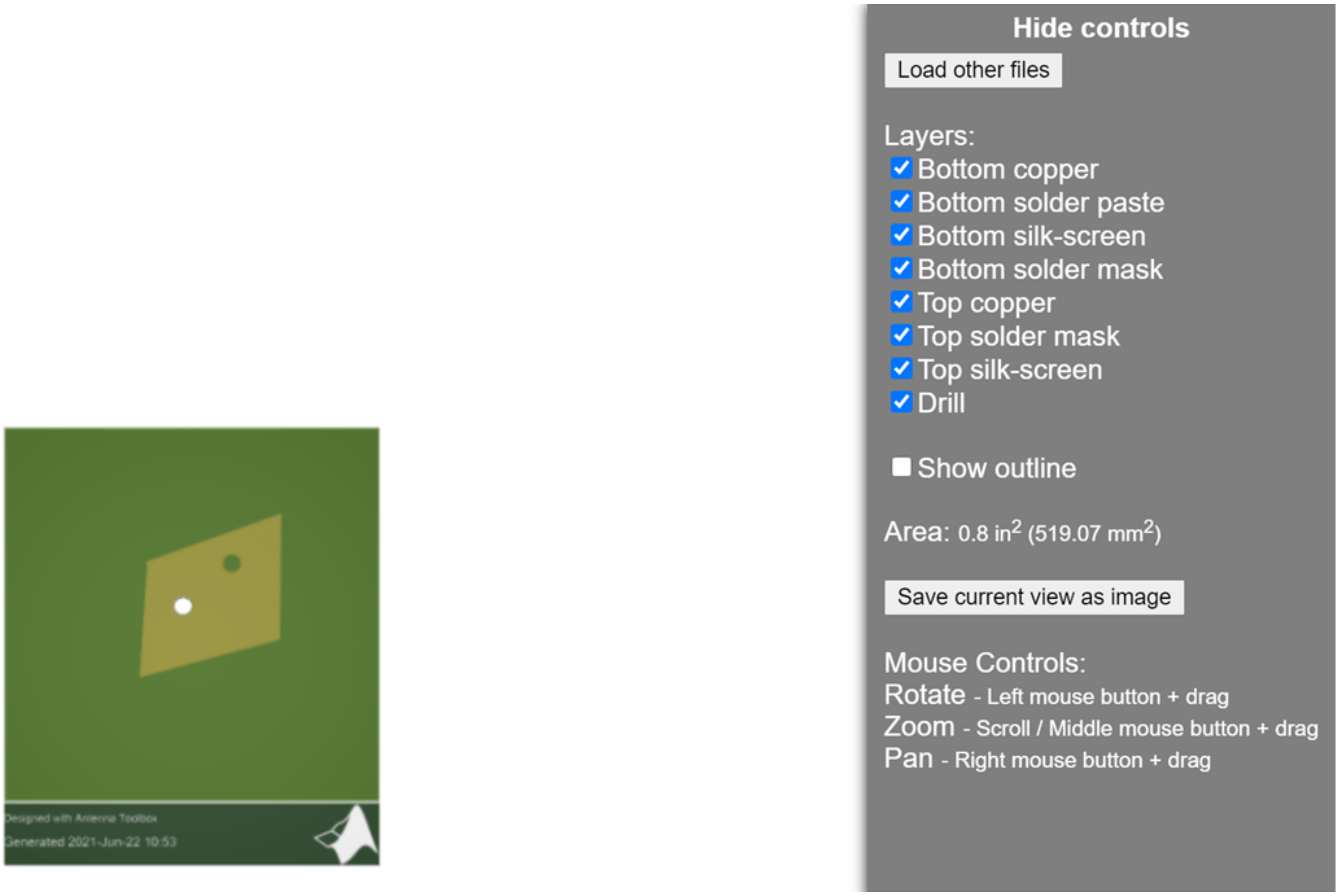


Step 1:
Drop gerber files here

Step 2:
Select the layers corresponding to the gerber files

pcbant.dri No layer
pcbant.gbl Bottom copper
pcbant.gbo Bottom silk-screen
pcbant.gbp Bottom solder paste
pcbant.gbs Bottom solder mask
pcbant.gpi No layer
pcbant.gtl Top copper
pcbant.gto Top silk-screen
pcbant.gts Top solder mask
pcbant.ipc No layer
pcbant.txt Drill

The Gerber files are displayed in the Gerber viewer. You can use the Gerber files in the fabrication of your PCB patch antennas.



Hide controls

Load other files

Layers:

☑ Bottom copper
☑ Bottom solder paste
☑ Bottom silk-screen
☑ Bottom solder mask
☑ Top copper
☑ Top solder mask
☑ Top silk-screen
☑ Drill

□ Show outline

Area: 0.8 in^2 (519.07 mm^2)

Save current view as image

Mouse Controls:
Rotate - Left mouse button + drag
Zoom - Scroll / Middle mouse button + drag
Pan - Right mouse button + drag

You can also export your design to the MATLAB workspace or MATLAB script by selecting **Export to MATLAB Workspace** or **Export to MATLAB Script** under the **Export** in the **Design** or **Analysis** tab.

- “Design and Analysis Using PCB Antenna Designer”

Programmatic Use

`pcbAntennaDesigner` opens the **PCB Antenna Designer** app to design and visualize a single or multifeed PCB antennas.

Version History

Introduced in R2021b

See Also

Apps
Antenna Designer | Antenna Array Designer

Functions
`pcbStack`

Topics
"Design and Analysis Using PCB Antenna Designer"

3

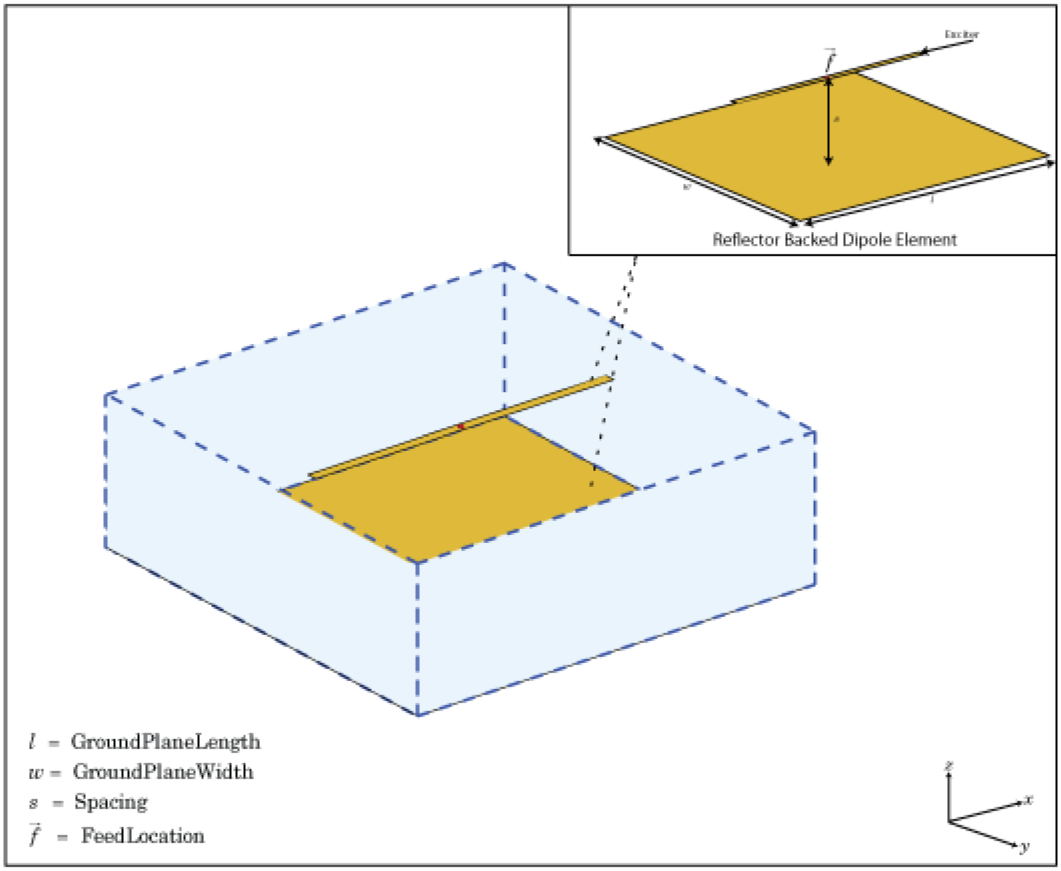
Array Objects

infiniteArray

Create infinite array of 2-D custom antenna in X-Y plane

Description

The `infiniteArray` object is an infinite antenna array in the X-Y plane. Infinite array models a single antenna element called the *unit cell*. Ground plane of the antenna specifies the boundaries of the unit cell. Antenna without a ground plane requires a reflector. The infinite array has a reflector-backed dipole as the default exciter antenna element. The default dimensions are chosen for an operating frequency of 1 GHz.



Creation

Description

`infa = infiniteArray` creates an infinite antenna array in the *xy*-plane.

`infa = infiniteArray(Name=Value)` sets “Properties” on page 3-3 using one or more name-value arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1=Value1`, `...`, `NameN=ValueN`. Properties you do not specify retain their default values.

Properties

`Element` — Type of individual antenna elements in unit cell

reflector-backed dipole antenna (default) | antenna object | `pcbStack` object

Type of individual antenna elements in a unit cell, specified as an antenna object or a `pcbStack` object. Back an antenna without a groundplane by a reflector. The ground plane size specifies the unit cell boundaries. Add vias to the infinite array by setting this property to a `pcbStack` object.

Note You cannot set this property to a `pcbStack` object for these specifications:

- Disconnected array of metal-dielectric structures
- Antenna with multiple dielectric layers, edge feed, and different dimensions of the ground plane and dielectric substrate
- Antenna that you create with the**PCB Antenna Designer** app

Example: `Element=reflector` creates an infinite array of reflector antennas.

`ScanAzimuth` — Scan direction in azimuth plane

`0` (default) | scalar

Scan direction in azimuth plane, specified as a scalar in degrees.

Example: `ScanAzimuth=25`

Data Types: `double`

`ScanElevation` — Scan direction in elevation plane

`0` (default) | scalar

Scan direction in elevation plane, specified as a scalar in degrees.

Example: `ScanElevation=80`

Data Types: `double`

`RemoveGround` — Remove ground plane

`false` or `0` (default) | `true` or `1`

Remove the ground plane of the reflector with air substrate, specified as a numeric or logical `1` (`true`) or `0` (`false`). When you specify `1`, the object removes the ground plane. When you specify `0`, the object does not remove the ground plane

Example: `RemoveGround=1`

Data Types: `logical`

Object Functions

numSummationTerms	Change number of summation terms for calculating periodic Green's function

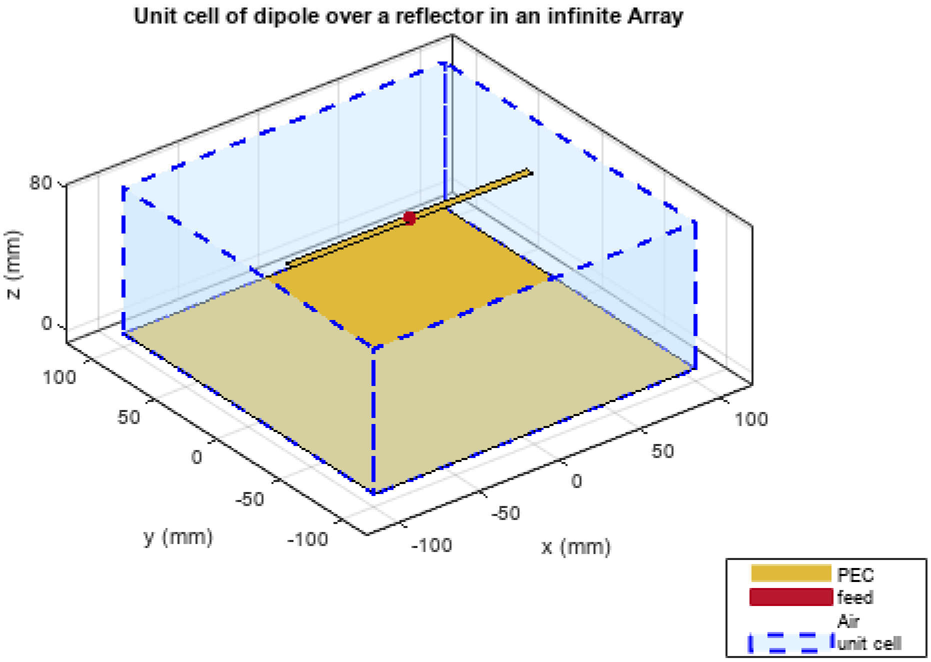
show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
layout	Display array or PCB stack layout
mesh	Mesh properties of metal, dielectric antenna, or array structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

Examples

Infinite Array of Reflector-Backed Dipoles

Create an infinite array of reflector-backed dipole as the unit cell. Scan the array at boresight. Visualize the unit cell.

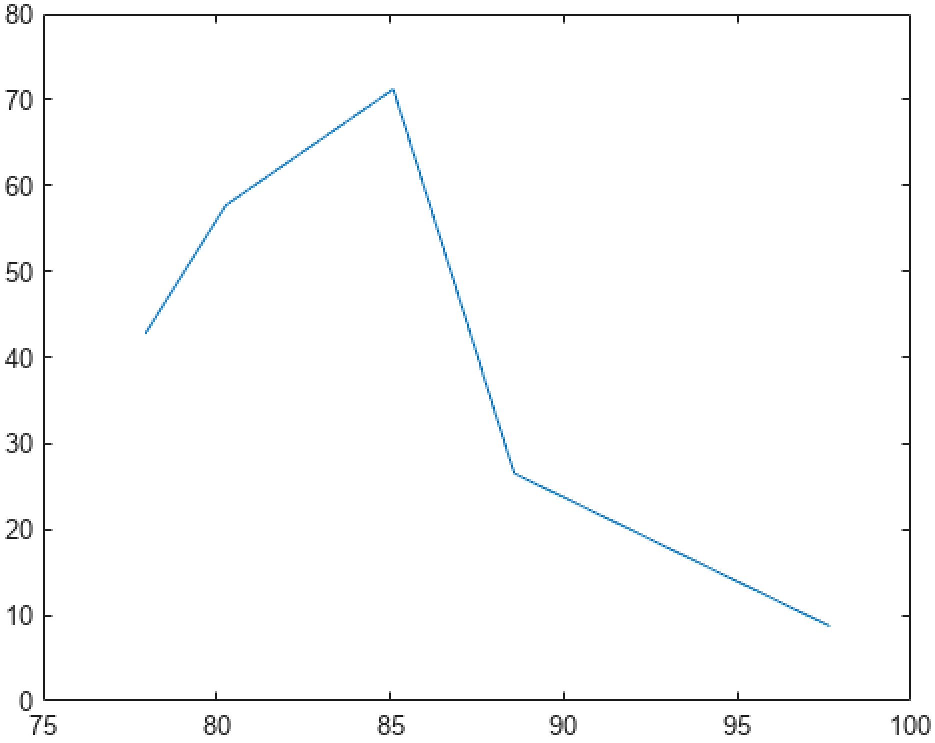
```
infa = infiniteArray(Element=reflector,ScanAzimuth=0, ...
    ScanElevation=90);
show(infa)
```



Scan Impedance of Infinite Array

Calculate the scan impedance of an infinite array at 1 GHz. To calculate the impedance, scan the infinite array from boresight to horizon in the elevation plane.

```
infa = infiniteArray;
theta0deg = linspace(0,90,5);
zscan = nan(1,numel(theta0deg));
    for j = 1:numel(theta0deg)
      infa.ScanElevation = theta0deg(j);
      zscan(1,j) = impedance(infa,1e9);
    end
 plot(zscan)
```



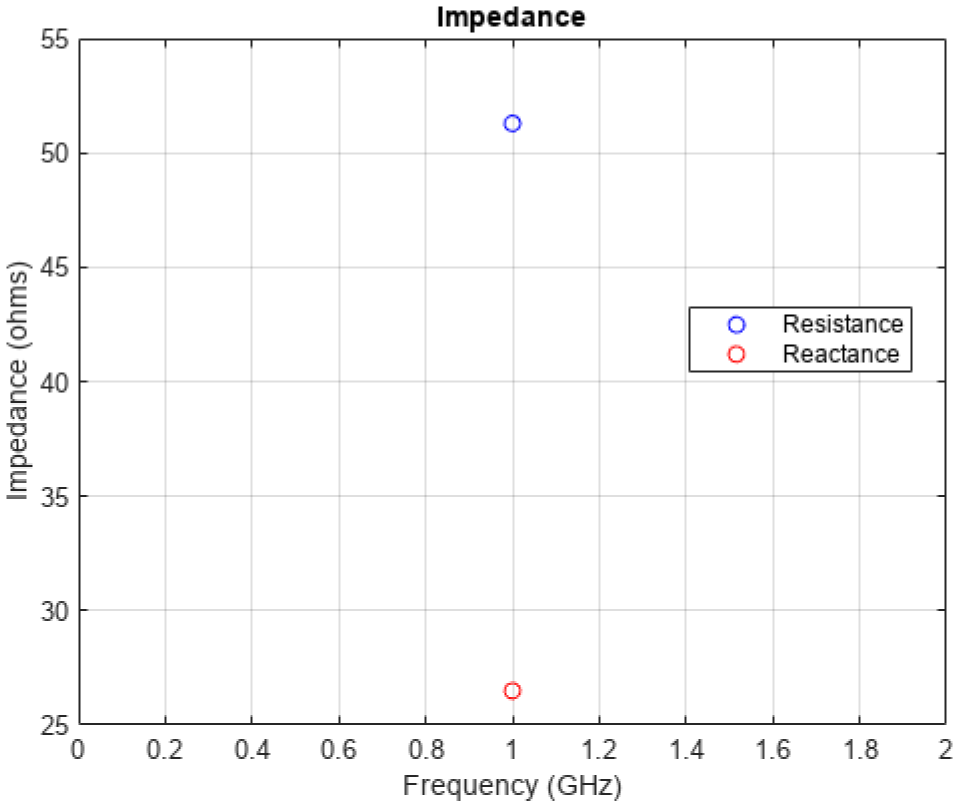
Calculate and Plot Scan Impedance at Bore Sight of Metal Infinite Array

Calculate the scan impedance at bore sight of a metal infinite array without ground plane at a frequency of 1 GHz.

```
h = infiniteArray(Element=reflector,ScanAzimuth=0, ...
    ScanElevation=90,RemoveGround=1);
zin = impedance(h,1e9)
```

```
zin = 51.2867 + 26.4937i
```

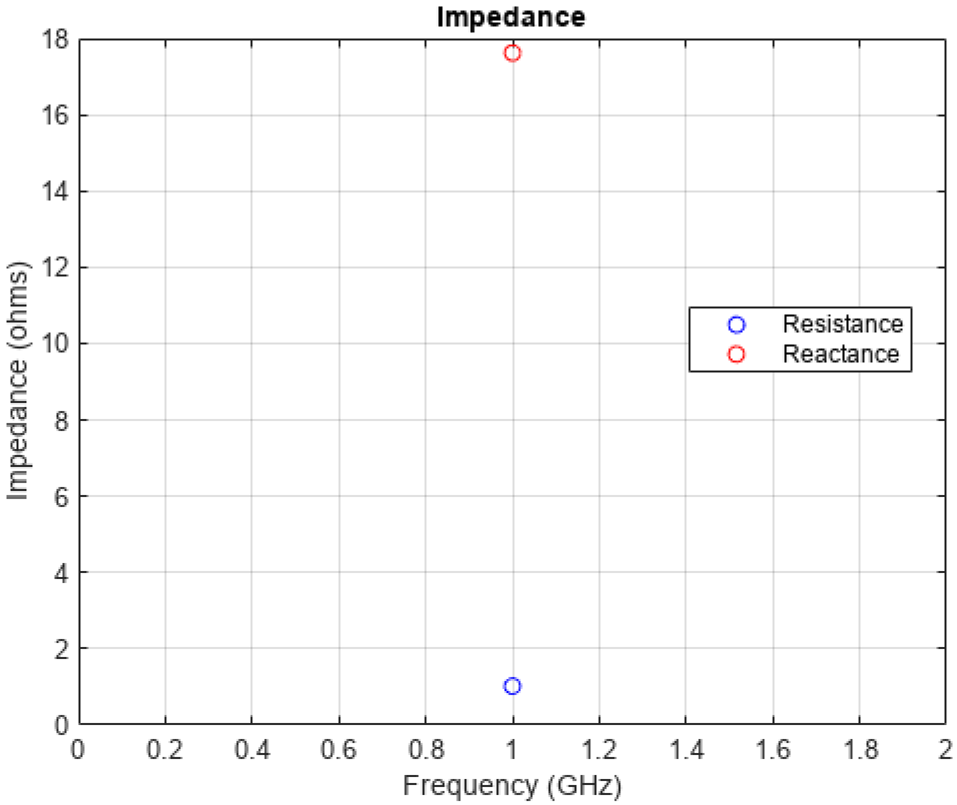
```
impedance(h,1e9)
```



Plot Scan Impedance at Boresight of Metal-Dielectric Infinite Array

Plot the scan impedance at the boresight of a metal-dielectric infinite array at a frequency of 1 GHz.

```
ant = patchMicrostrip(Substrate=dielectric('Teflon'));
h = infiniteArray(Element=ant,ScanAzimuth=0,ScanElevation=90);
impedance(h,1e9)
```



Add Via to Infinite Array

This example shows how to add vias to an infinite array using a `pcbStack` object.

Create Infinite Array of pcbStack Objects

Create a `pcbStack` object with an FR4 dielectric substrate. Define the feed dimension and location. Define the via dimension and location. Create an infinite array of this `pcbStack` object.

```
f = 1e9;
lambda = 3e8/f;
p = pcbStack;
d = dielectric('FR4');
p.BoardThickness = d.Thickness;
p.Layers = {p.Layers{1} d p.Layers{2}};
p.FeedLocations = [0.02 0 1 3];
p.FeedDiameter = 1e-3;
p.ViaLocations = [0 0 1 3];
p.ViaDiameter = 1e-3;
ant = infiniteArray(Element=p)
```

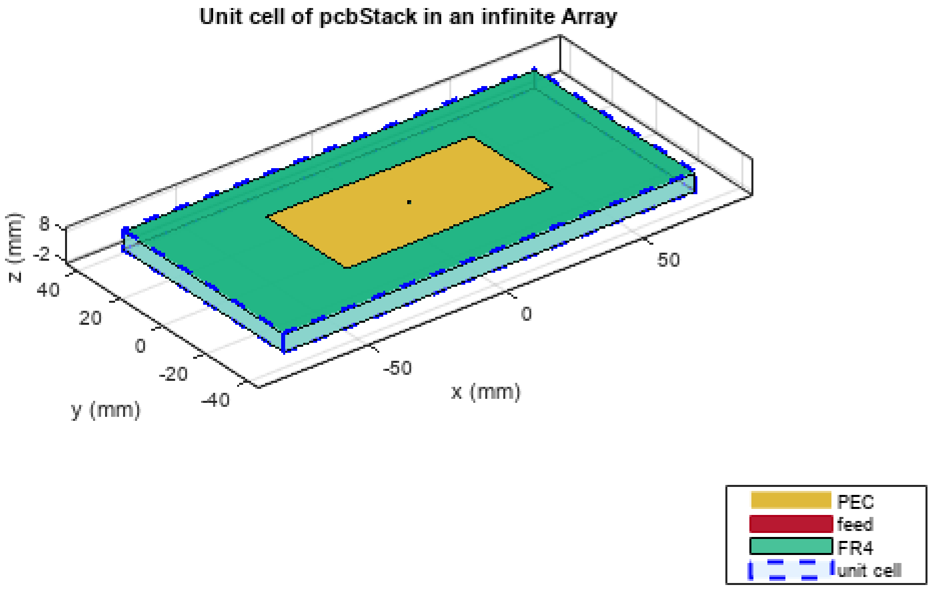
```
ant = 
  infiniteArray with properties:

          Element: [1x1 pcbStack]
```

```
      ScanAzimuth: 0
    ScanElevation: 90
     RemoveGround: 0
```

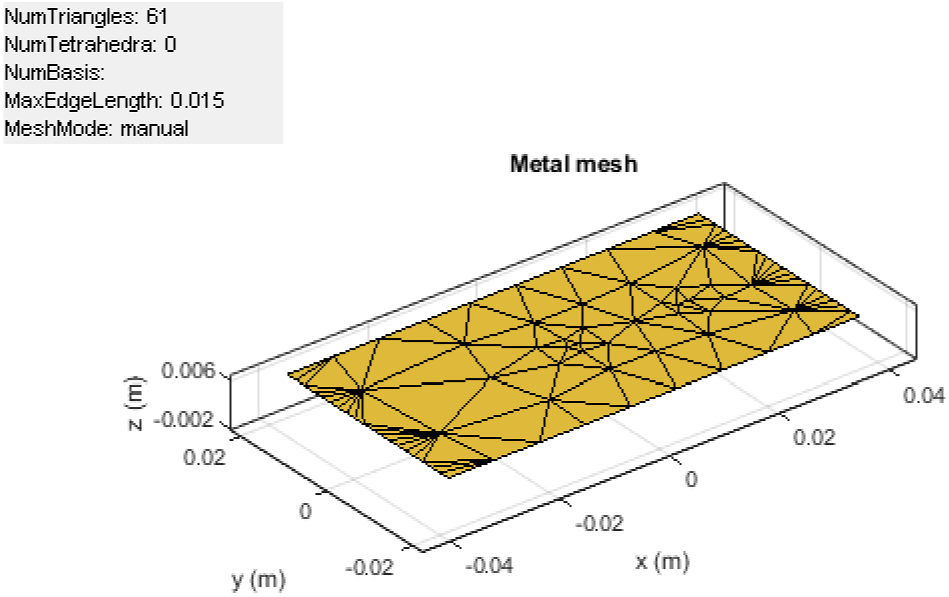
View Array and Elevation Pattern

View and mesh the array. Plot the elevation pattern of the array at 1 GHz.

```
figure;
show(ant)
```

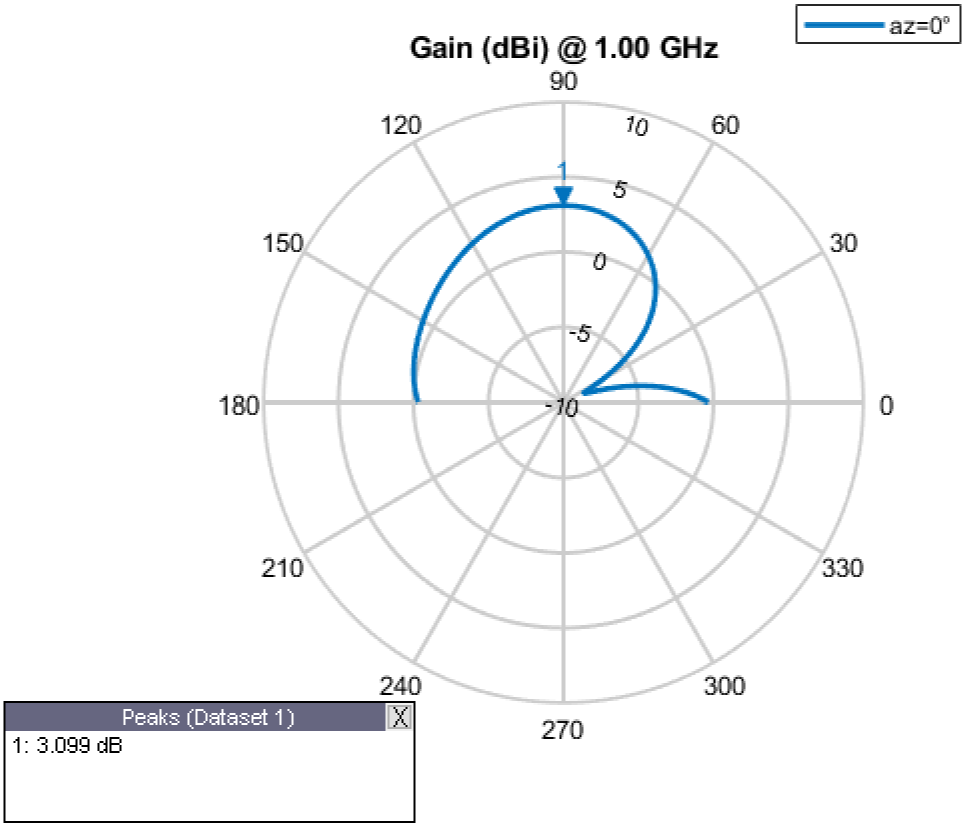


```
figure;
mesh(ant,MaxEdgeLength=lambda/20)
```



NumTriangles: 61
NumTetrahedra: 0
NumBasis:
MaxEdgeLength: 0.015
MeshMode: manual

```
figure;
patternElevation(ant,f)
```



Version History

Introduced in R2015b

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

See Also

`linearArray` | `rectangularArray` | `conformalArray` | `circularArray` | “Infinite Arrays”

Topics

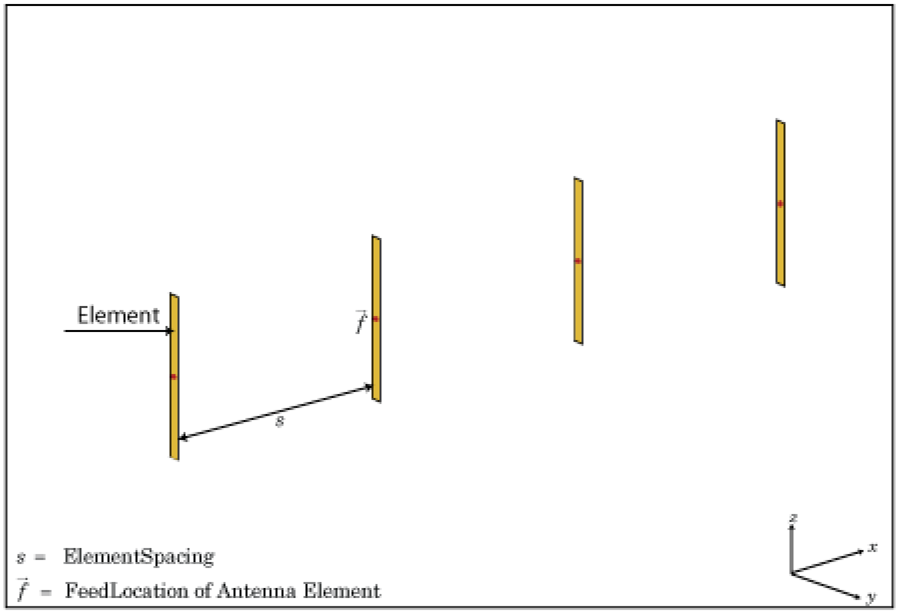
“Rotate Antennas and Arrays”

linearArray

Create linear antenna array

Description

The `linearArray` class creates a linear antenna array in the X-Y plane. By default, the linear array is a two-element dipole array. The dipoles are center fed. Each dipole resonates at 70 MHz when isolated.



Creation

Syntax

```
array = linearArray
array = linearArray(Name,Value)
```

Description

`array = linearArray` creates a linear antenna array in the X-Y plane.

`array = linearArray(Name,Value)` class to create a linear antenna array, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`,`...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Output Arguments

`array` — Linear array
`linearArray` object

Linear array, returned as an `linearArray` object.

Properties

`Element` — Individual antenna elements, or array elements in array
dipole (default) | antenna object | array object

Individual antenna elements or array elements, specified as an antenna or array object.

Example: `'Element',monopole`

`NumElements` — Number of antenna elements in array
2 (default) | scalar

Number of antenna elements in array, specified as a scalar.

Example: `'NumElements'`,4

`'ElementSpacing'` — Spacing between antenna elements
2 (default) | scalar | vector

Spacing between antenna elements, specified as a scalar or vector in meters. By default, the dipole elements are spaced `2 m` apart.

Example: `'ElementSpacing',3`

Data Types: `double`

`AmplitudeTaper` — Excitation amplitude of antenna elements
1 (default) | scalar | vector

Excitation amplitude of antenna elements, specified as a scalar or vector. Set the property value to `0` to model dead elements. This value corresponds to the excitation voltages for the elements in the array.

Example: `'AmplitudeTaper',3`

Data Types: `double`

`Phaseshift` — Phase shift for antenna elements
0 (default) | scalar | vector

Phase shift for antenna elements, specified as a scalar or vector in degrees. This value corresponds to the excitation voltages for the elements in the array.

Example: `'PhaseShift',[3 3 0 0]`

Data Types: `double`

`Tilt` — Tilt angle of array
`0` (default) | scalar | vector

Tilt angle of the array specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`,

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the array at 90 degrees about the two axes, defined by vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of array

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the array, specified as:

- Three-element vectors of Cartesian coordinates in meters. In this case, each vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the array rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis='Z'`

Data Types: `double`

Object Functions

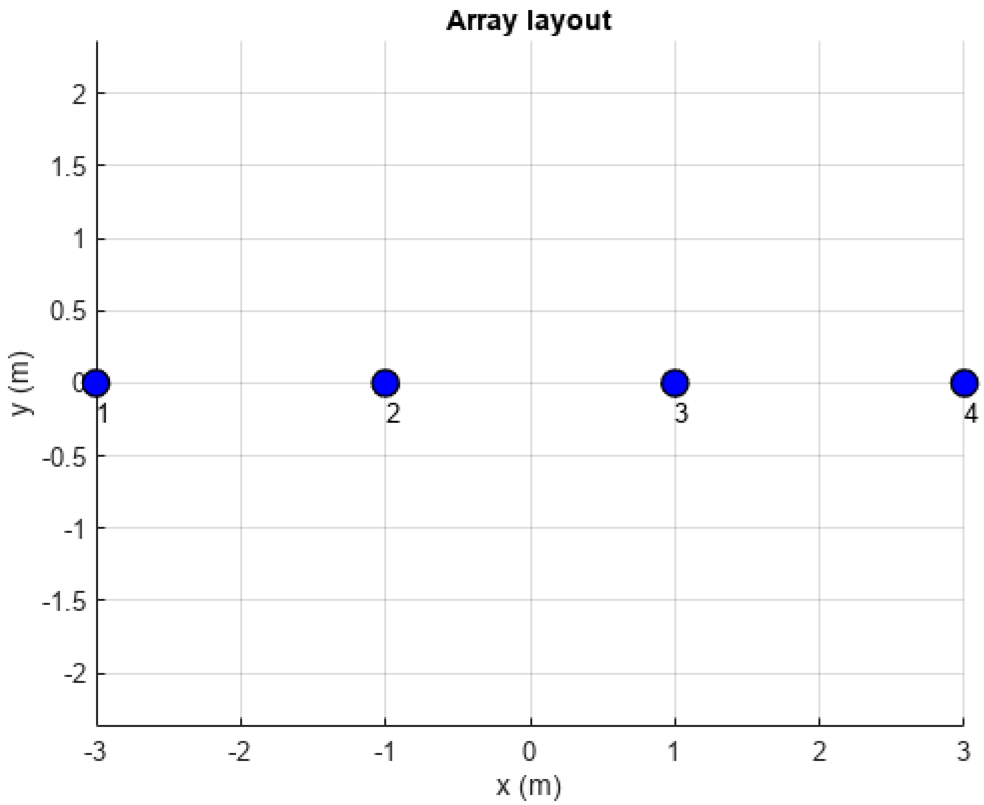
show	Display antenna, array structures or shapes
info	Display information about antenna or array
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
layout	Display array or PCB stack layout
mesh	Mesh properties of metal, dielectric antenna, or array structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

Examples

Create and Plot Layout of Linear Array

Create a linear array of four dipoles and plot the layout of the array.

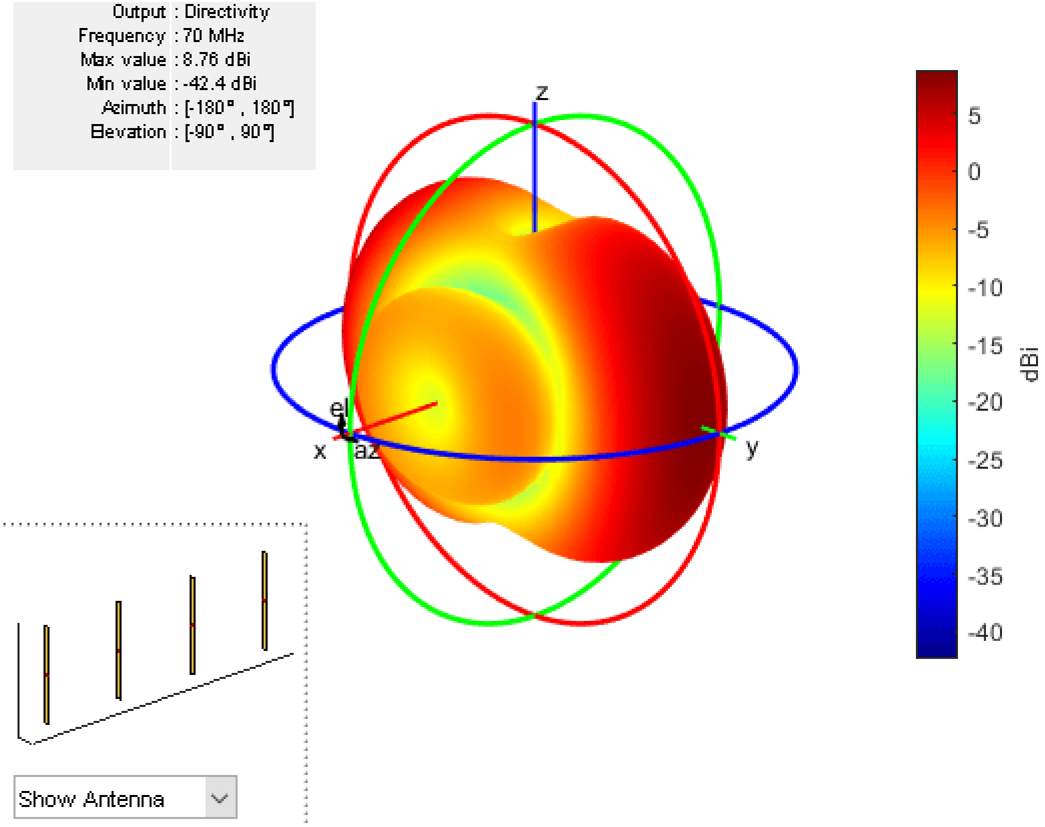
```
la = linearArray;
la.NumElements = 4;
layout(la);
```



Radiation Pattern of Linear Array

Plot the radiation pattern of a four element linear array of dipoles at a frequency 70MHz.

```
la = linearArray('NumElements',4);
pattern(la,70e6);
```


Output : Directivity
Frequency : 70 MHz
Max value : 8.76 dBi
Min value : -42.4 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

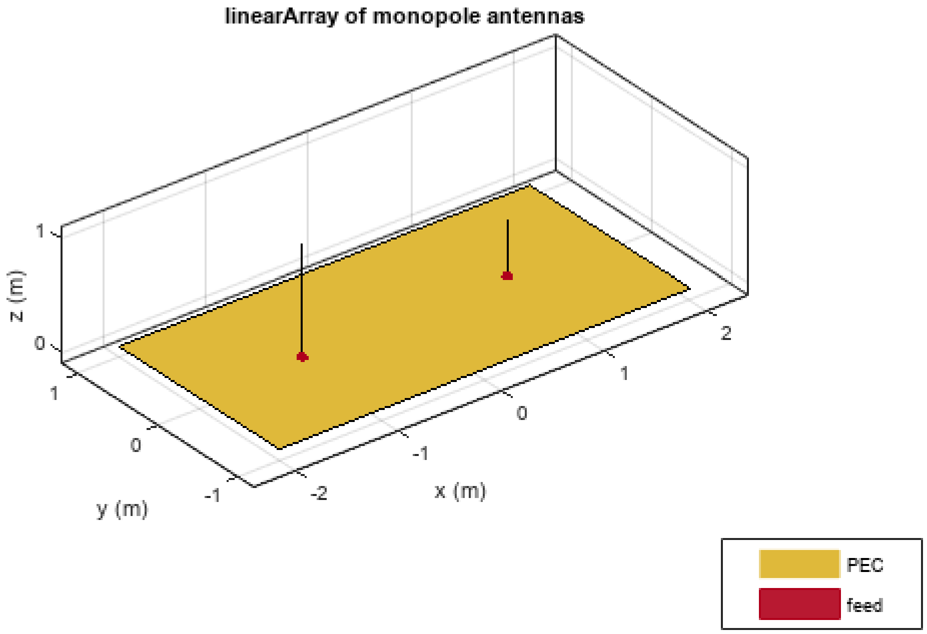
Linear Array Using Groundplane Antennas

Create a linear array of two monopoles.

```
m1 = monopole;
m2 = monopole('Height',0.5);
mla = linearArray
```

```
mla = 
  linearArray with properties:

           Element: [1x1 dipole]
       NumElements: 2
    ElementSpacing: 2
     AmplitudeTaper: 1
        PhaseShift: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

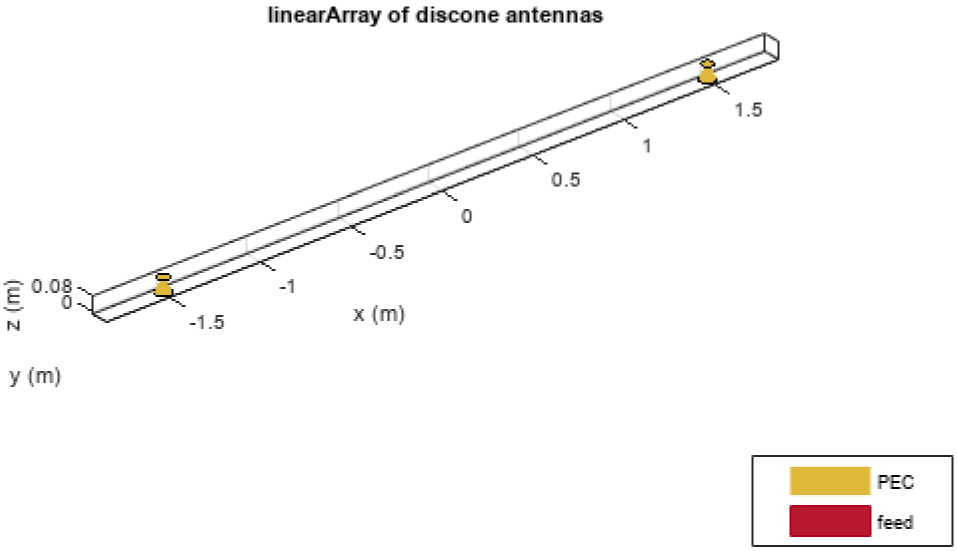
```
mla.Element = [m1,m2];
show(mla);
```



Rectangular Array of Linear Array

Create an array of discones with element spacing of 3 m.

```
la = linearArray('Element',discone);
la.ElementSpacing = 3;
show(la)
```

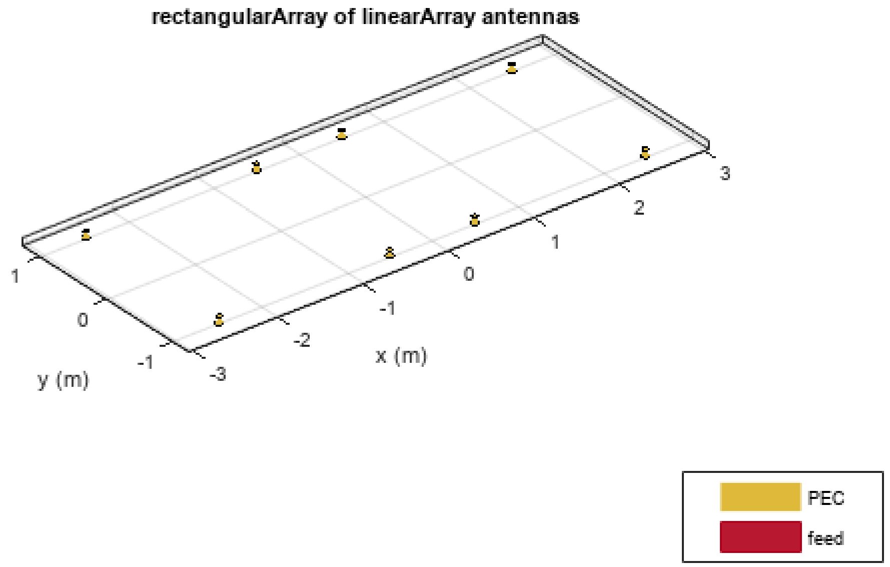


Create a rectangular of the linear array.

```
ra = rectangularArray("Element",la)
```

```
ra = 
  rectangularArray with properties:

           Element: [1x1 linearArray]
              Size: [2 2]
        RowSpacing: 2
     ColumnSpacing: 2
           Lattice: 'Rectangular'
    AmplitudeTaper: 1
        PhaseShift: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

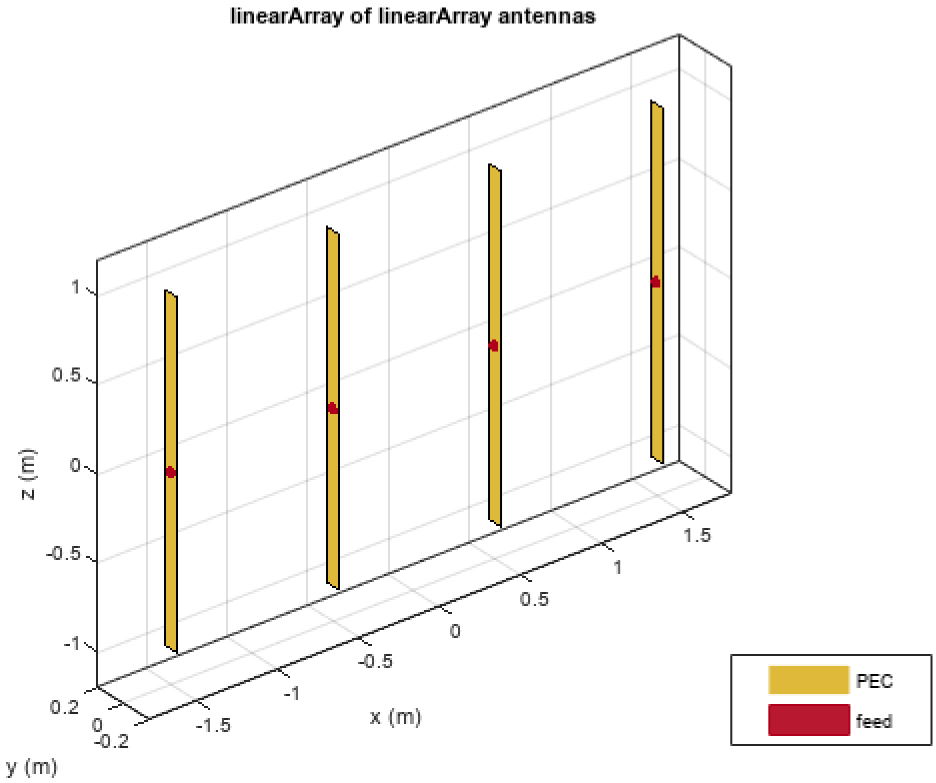
```
show(ra)
```



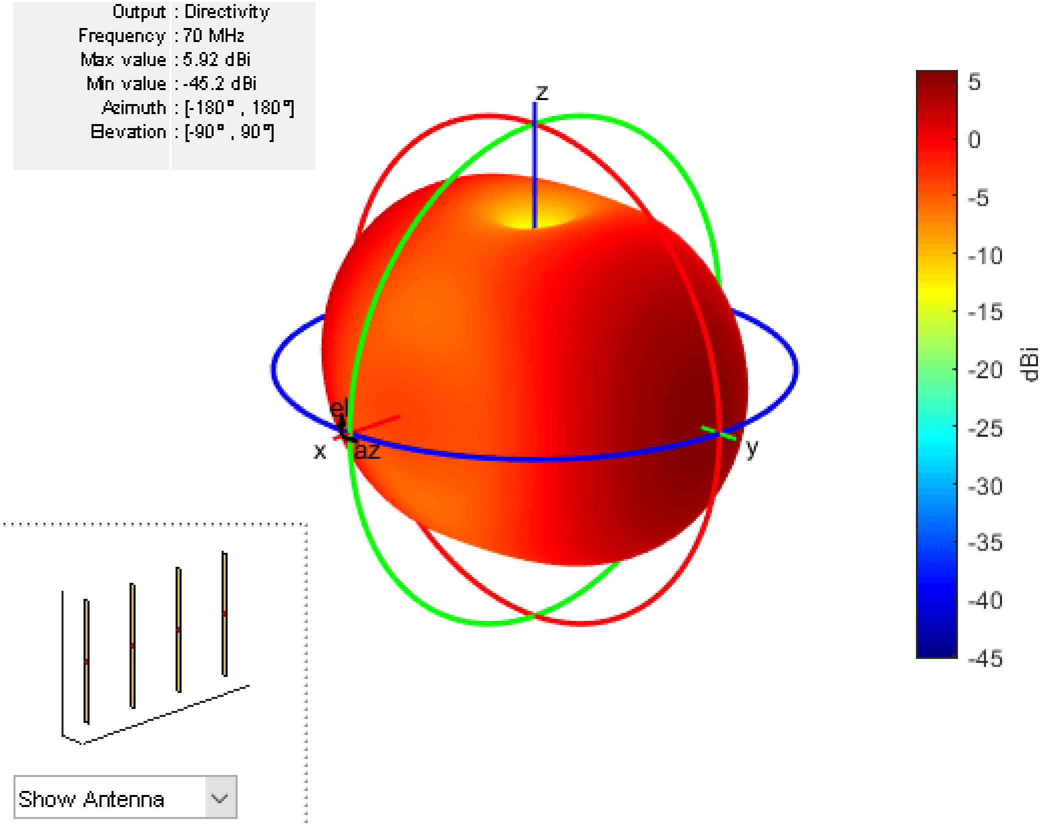
Pattern of Linear Array of Linear Array

Create a linear array and plot the pattern.

```
la=linearArray('Element',linearArray('ElementSpacing',1));
show(la)
```



```
pattern(la,70e6);
```



Output : Directivity
Frequency : 70 MHz
Max value : 5.92 dBi
Min value : -45.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`rectangularArray` | `conformalArray` | `infiniteArray` | `circularArray`

Topics

"Rotate Antennas and Arrays"

conformalArray

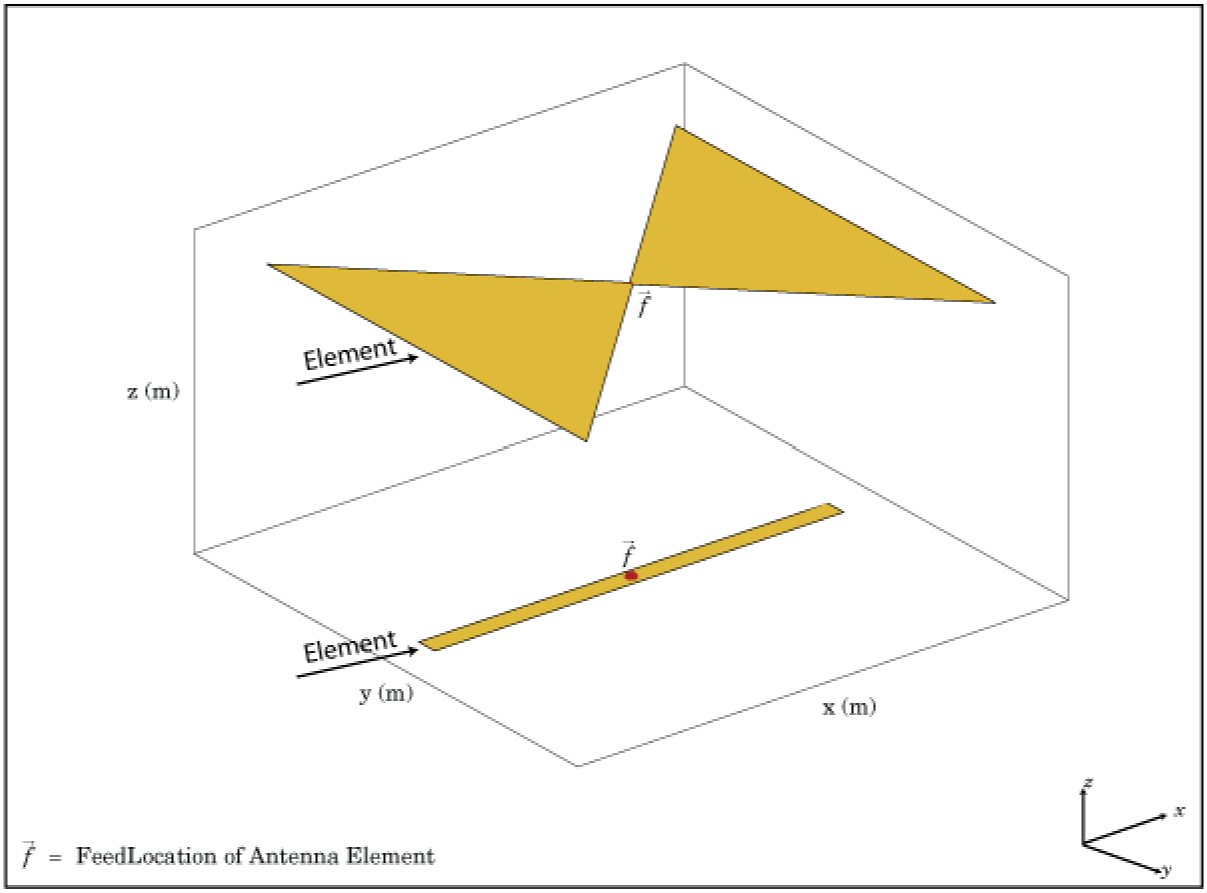
Create conformal antenna array

Description

The `conformalArray` class creates an antenna array using any element from the antenna or array library. You can also specify an array of any arbitrary geometry, such as a circular array, a nonplanar array, an array with nonuniform geometry, or a conformal array of arrays.

Conformal arrays are used in:

- Direction-finding systems that use circular arrays or stacked circular arrays
- Aircraft systems due to surface irregularities or mechanical stress



Creation

Syntax

```
array = conformalArray
array = conformalArray(Name,Value)
```

Description

`array = conformalArray` creates a conformal antenna array using the default antenna element, shape, and antenna positions.

`array = conformalArray(Name,Value)` creates a conformal antenna array with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain default values.

Properties

`ElementPosition` — Position of feed or origin
`[0 0 0; 0 0 0.1500]` (default) | *M*-by-3 real matrix

Position of the feed or origin for each antenna element, specified as an *M*-by-3 real matrix. *M* is the number of element positions. By default, *M* is `2`. To specify additional antenna elements, add additional element positions in the conformal array.

Example: `'ElementPosition',[0.1 0.1 0.1; -0.1 -0.1 -0.1;0.2 0.2]`

Data Types: `double`

`Element` — Individual antenna or array elements in array
scalar | array of objects | cell array of objects

Individual antenna or array elements in the array, specified as one of the following values:

- A scalar
- An array of objects
- A cell array of objects

By default, a conformal array has two antenna elements, the dipole and the bowtie. To specify additional antenna or array elements, add additional element positions in the conformal array. You can add both balanced and unbalanced antennas to the same conformal array.

Example: `m = monopole; h = conformalArray('Element', [m,m])`. Creates a conformal array consisting of two monopoles antenna elements.

Example: `la = linearArray; ra = rectangularArray; h = conformalArray('Element', {la,ra})`. Creates a conformal array consisting of a linear array and a rectangular array.

Data Types: `cell`

`Reference` — Position reference for antenna element
`'feed'` (default) | `'origin'`

Position reference for the antenna element, specified as either `'origin'` or `'feed'`. For more information, see “Position Reference” on page 3-40.

Example: `'Reference','origin'`

Data Types: `char` | `string`

`AmplitudeTaper` — Excitation amplitude of antenna elements

1 (default) | scalar | nonnegative vector

Excitation amplitude of the antenna elements, specified as a scalar or a nonnegative vector. To model dead elements, set the property value to `0`.

Example: `'AmplitudeTaper',3`

Example: `'AmplitudeTaper',[3 0]`. Creates a two-element conformal array, where `3` and `0` are the excitations amplitudes of two elements.

Data Types: `double`

`PhaseShift` — Phase shift for antenna elements

0 (default) | scalar | real vector

Phase shift for antenna elements, specified as a scalar or a real vector in degrees.

Example: `'PhaseShift',[-45 -45 45 45]`

Data Types: `double`

`Tilt` — Tilt angle of array

`0` (default) | scalar | vector

Tilt angle of the array specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`,

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the array at 90 degrees about the two axes, defined by vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of array

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the array, specified as:

- Three-element vectors of Cartesian coordinates in meters. In this case, each vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the array rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis='Z'`

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
doa	Direction of arrival of signal
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
layout	Display array or PCB stack layout
mesh	Mesh properties of metal, dielectric antenna, or array structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

Examples

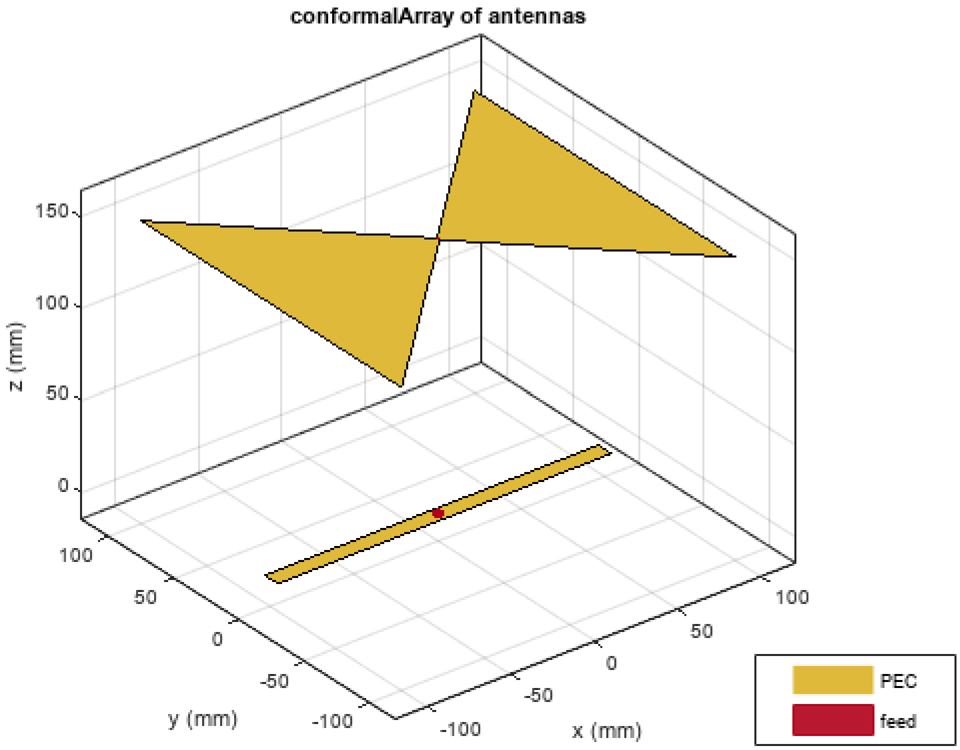
Default Conformal Array

Create a default conformal array.

```
c = conformalArray
```

```
c = 
  conformalArray with properties:

            Element: {[1x1 dipole]  [1x1 bowtieTriangular]}
    ElementPosition: [2x3 double]
          Reference: 'feed'
     AmplitudeTaper: 1
         PhaseShift: 0
               Tilt: 0
           TiltAxis: [1 0 0]
```

```
show(c)
```



Circular Array of Dipoles

Define the radius and the number of elements for the array.

```
r = 2;
N = 12;
```

Create an array of 12 dipoles.

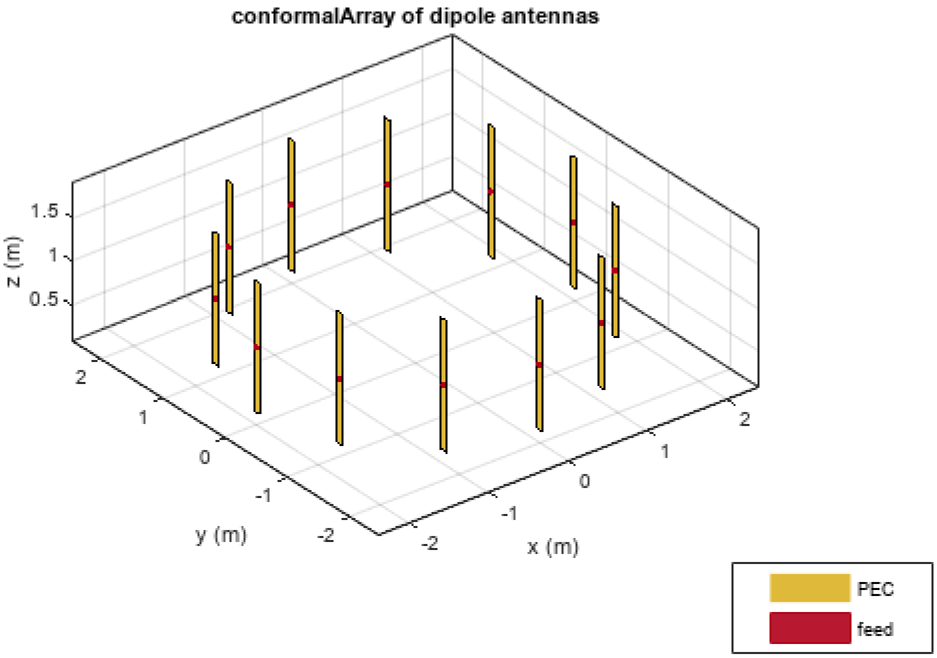
```
elem = repmat(dipole('Length',1.5),1,N);
```

Define the x,y,z values for the element positions in the array.

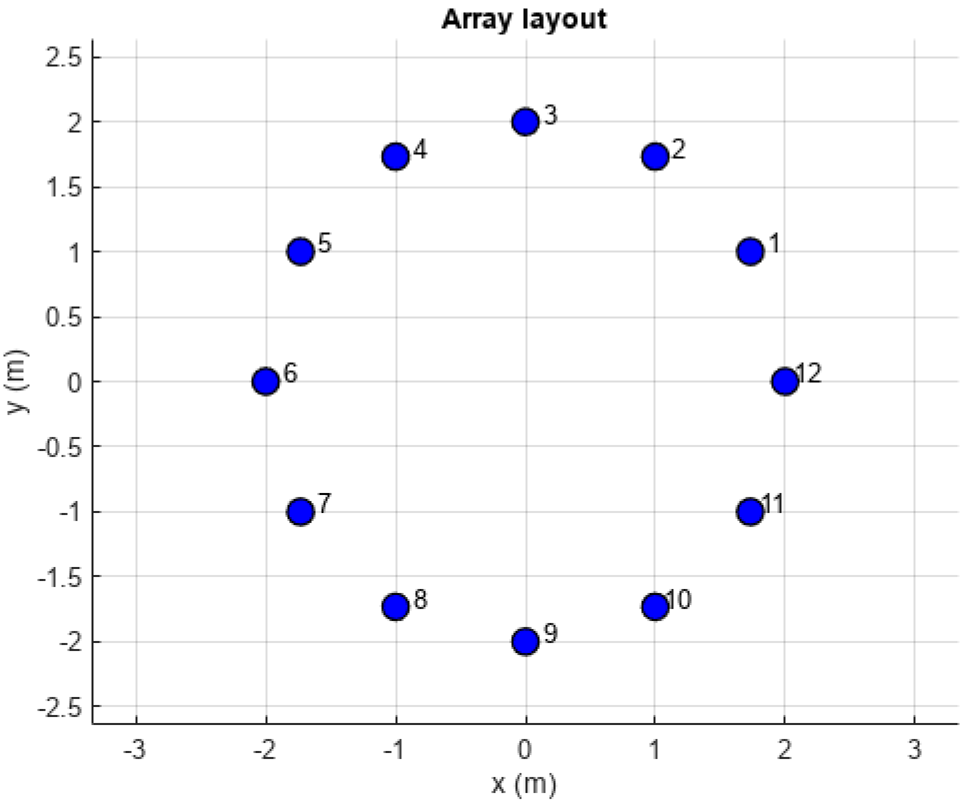
```
del_th = 360/N;
th = del_th:del_th:360;
x = r.*cosd(th);
y = r.*sind(th);
z = ones(1,N);
pos = [x;y;z];
```

Create a circular array using the defined dipoles and then visualize it. Display the layout of the array.

```
c = conformalArray('Element',elem,'ElementPosition',pos');
show(c)
```

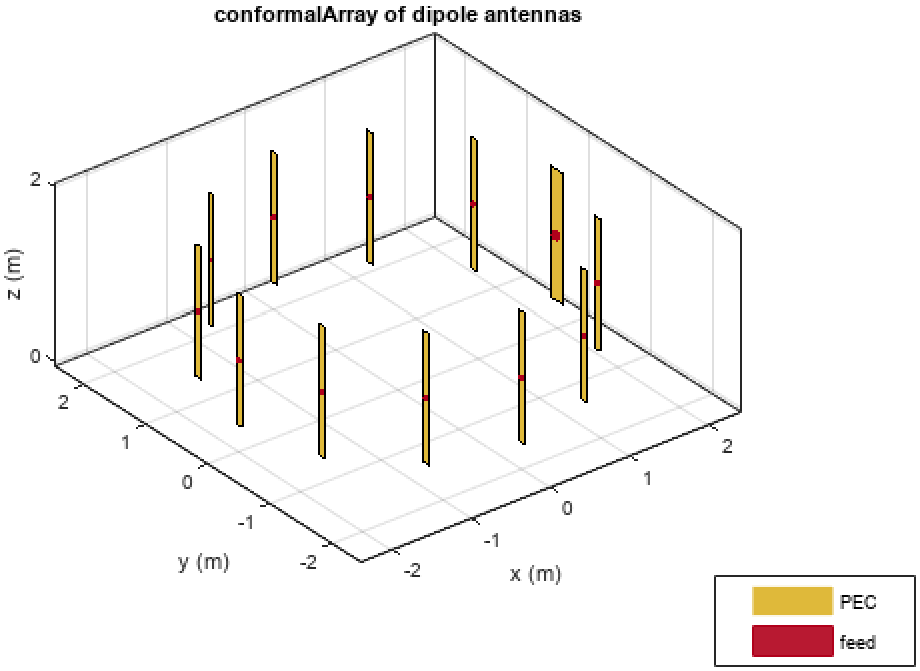


```
figure
layout(c)
```



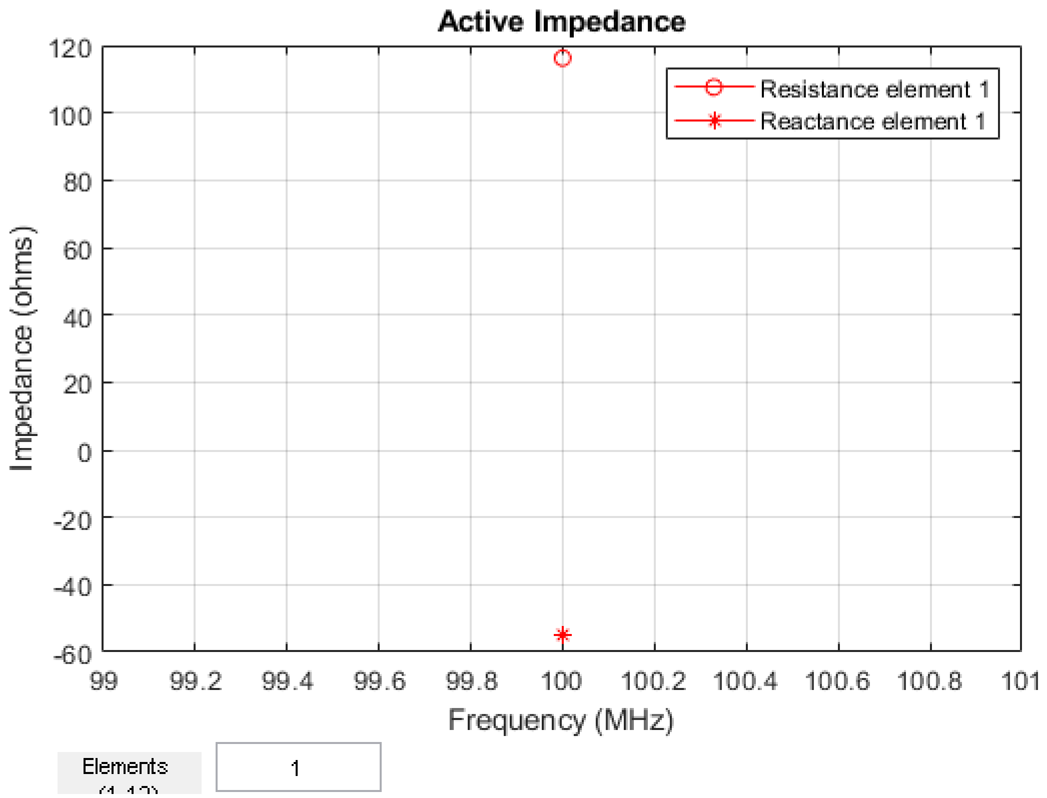
Change the width of the fourth and the twelfth element of the circular array. Visualize the new arrangement.

```
c.Element(4).Width = 0.05;
c.Element(12).Width = 0.2;
figure
show(c)
```

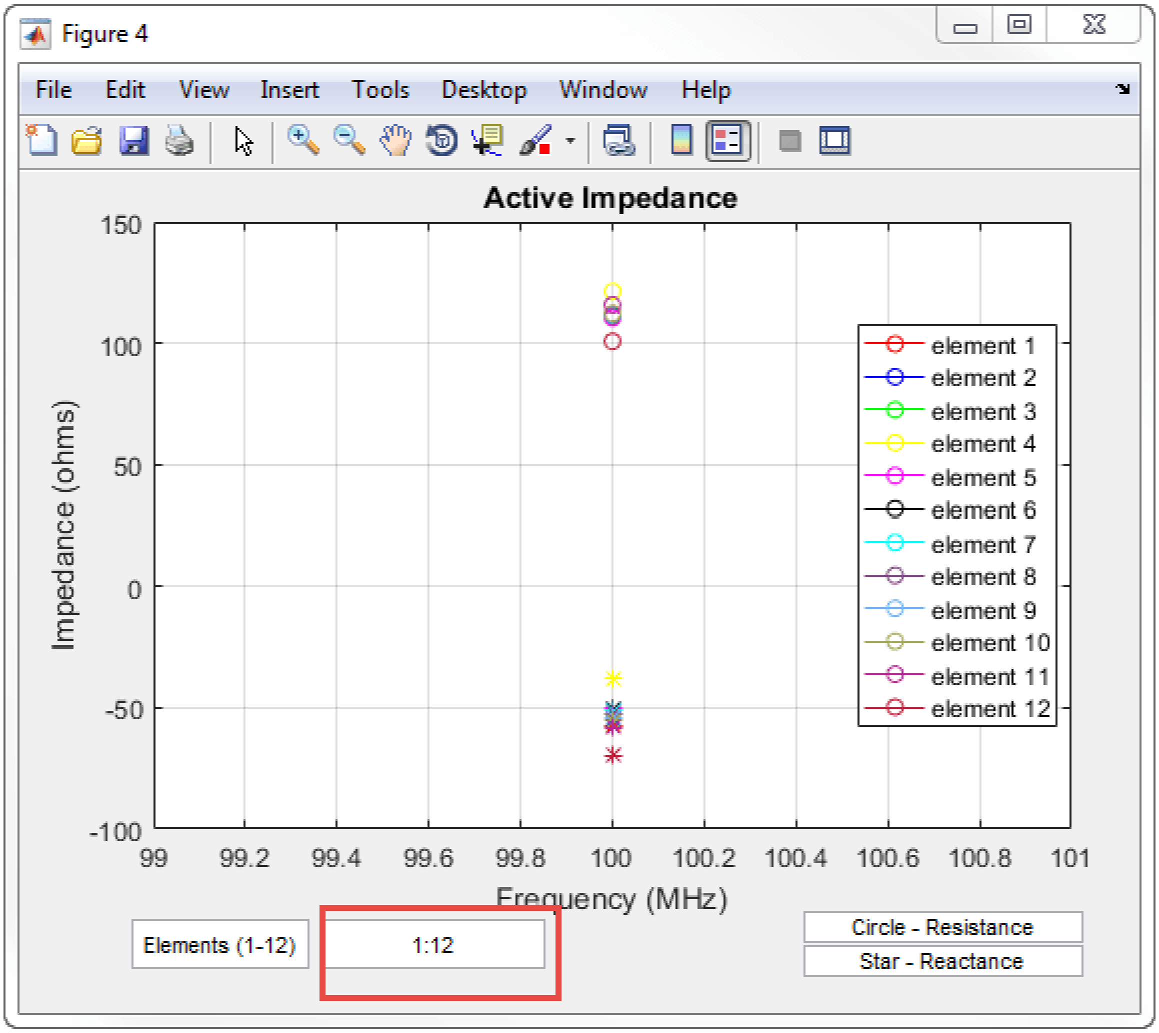


Calculate and plot the impedance of the circular array at 100 MHz. The plot shows the impedance of the first element in the array.

```
figure
impedance(c,100e6)
```



To view the impedance of all the elements in the array change the value from **1** to **1:12** as shown in the figure.



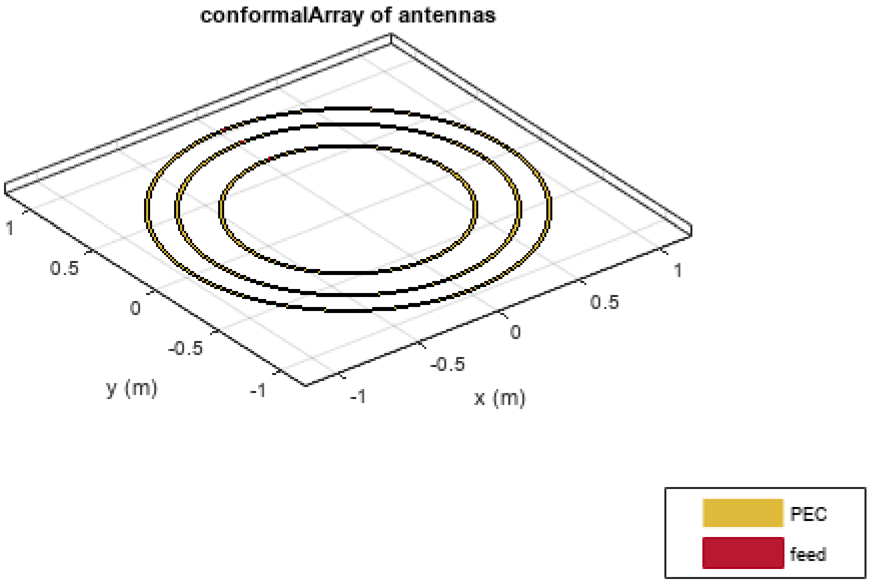
Radiation Pattern of Concentric Array of Circular Loop Antennas

Define three circular loop antennas of radii 0.6366 m (default), 0.85 m, and 1 m, respectively.

```
l1 = loopCircular;
l2 = loopCircular('Radius',0.85);
l3 = loopCircular('Radius',1);
```

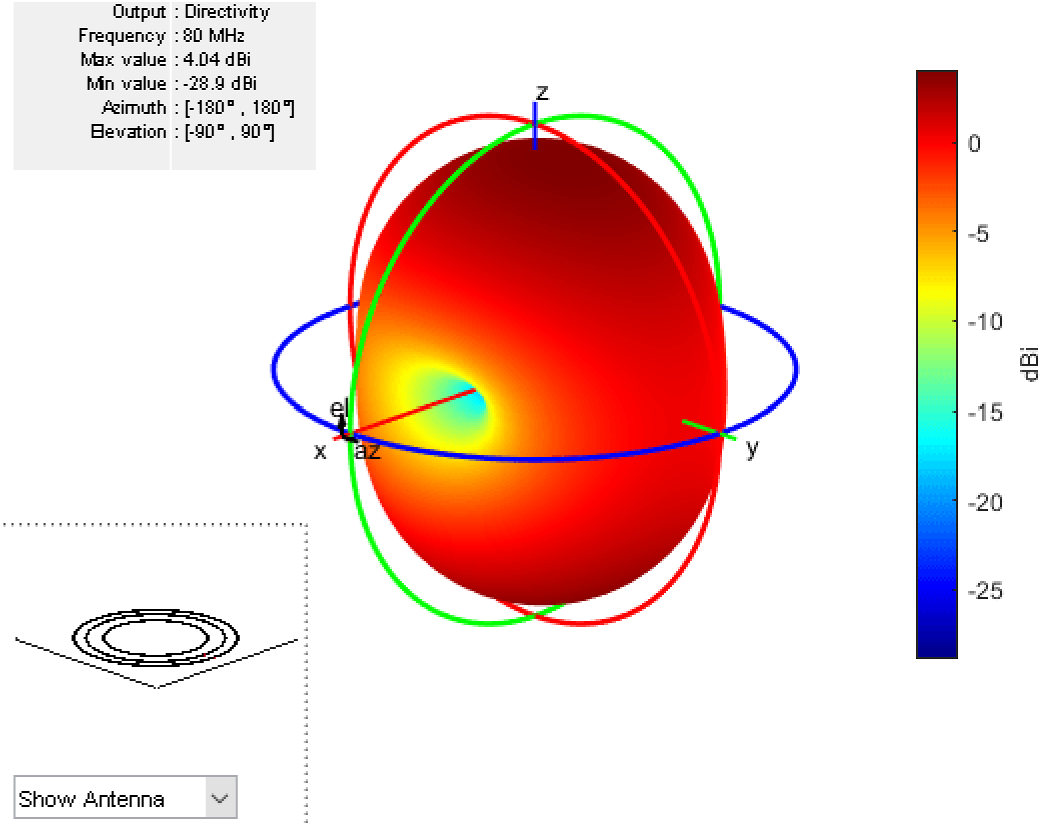
Create a concentric array that uses the origin of circular loop antennas as its position reference.

```
c = conformalArray('Element',{l1,l2,l3},'ElementPosition',[0 0 0;0 0 0;...
    0 0 0],'Reference','origin');
show(c)
```



Visualize the radiation pattern of the array at 80 MHz.

```
pattern(c,80e6)
```



Conformal Array Using Infinite Ground Plane Antenna

Create a dipole antenna to use in the reflector and the conformal array.

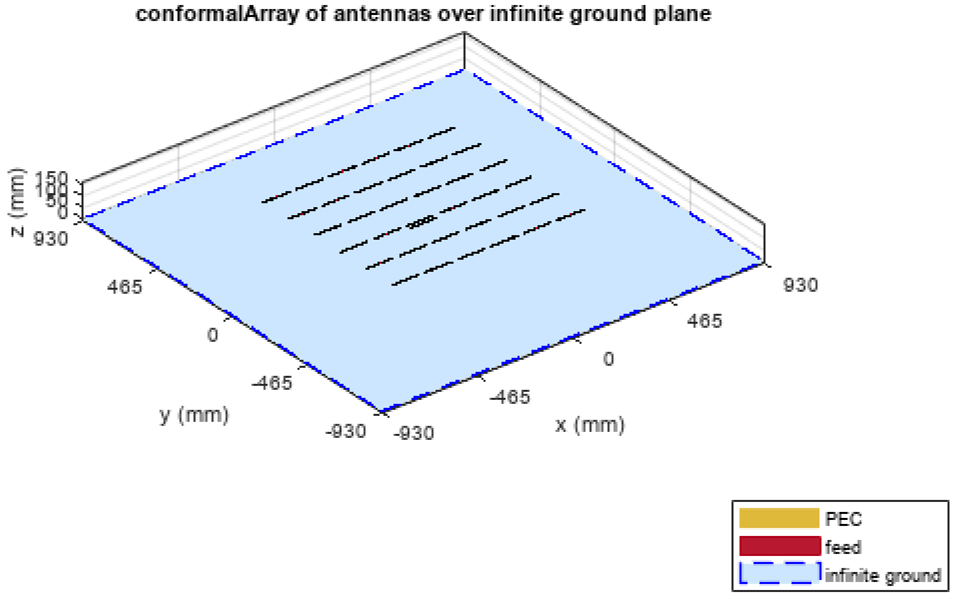
```
d = dipole('Length',0.13,'Width',5e-3,'Tilt',90,'TiltAxis','Y');
```

Create an infinite groundplane reflector antenna using the dipole as exciter.

```
rf = reflector('Exciter',d,'Spacing',0.15/2,'GroundPlaneLength',inf);
```

Create a conformal array using 36 dipole antennas and one infinite groundplane reflector antenna. View the array.

```
x = linspace(-0.4,0.4,6);
y = linspace(-0.4,0.4,6);
[X,Y] = meshgrid(x,y);
pos = [X(:) Y(:) 0.15*ones(numel(X),1)];
for i = 1:36
    element{i} = d;
end
element{37} = rf;
lwa = conformalArray('Element',element,'ElementPosition',[pos;0 0 0.15/2]);
show(lwa)
```

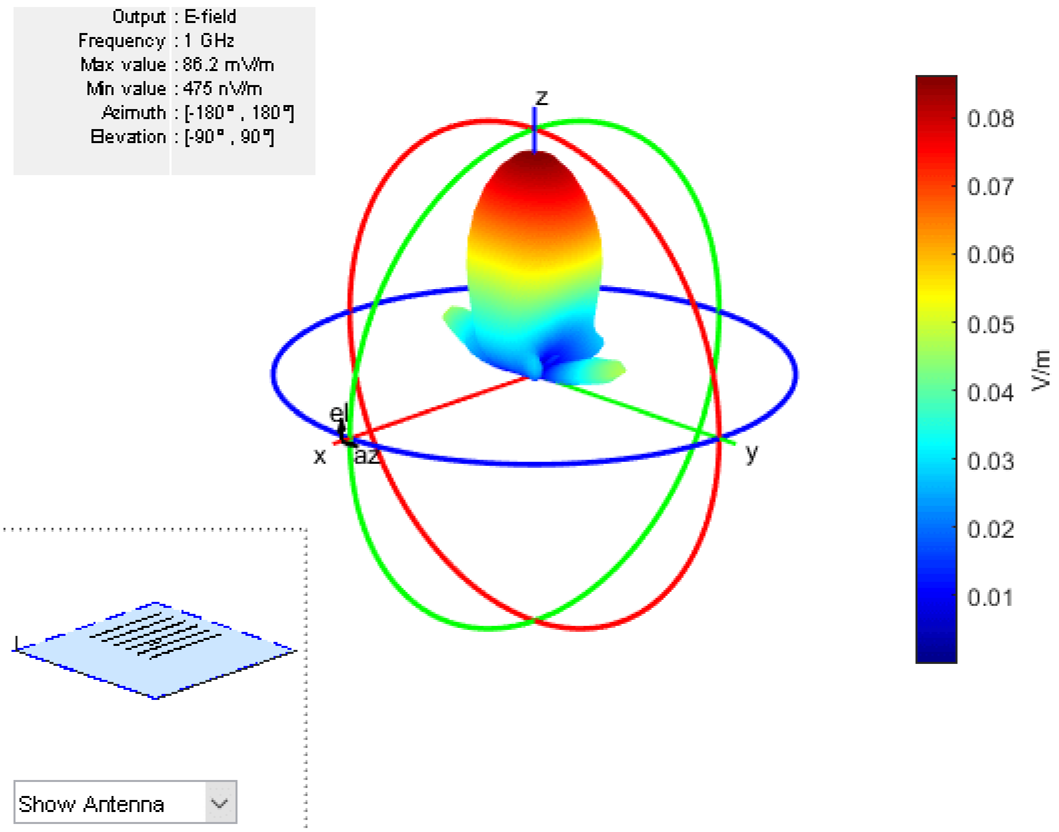


Drive only the reflector antenna with an amplitude of 1.

```
V = zeros(1,37);
V(end) = 1;
lwa.AmplitudeTaper = V;
```

Compute the radiation pattern of the conformal array.

```
figure
pattern(lwa,1e9,'Type','efield')
```



Conformal Array Using Dielectric Antennas

Create two patch microstrip antennas using dielectric substrate FR4. Tilt the second patch microstrip antenna by 180 degrees.

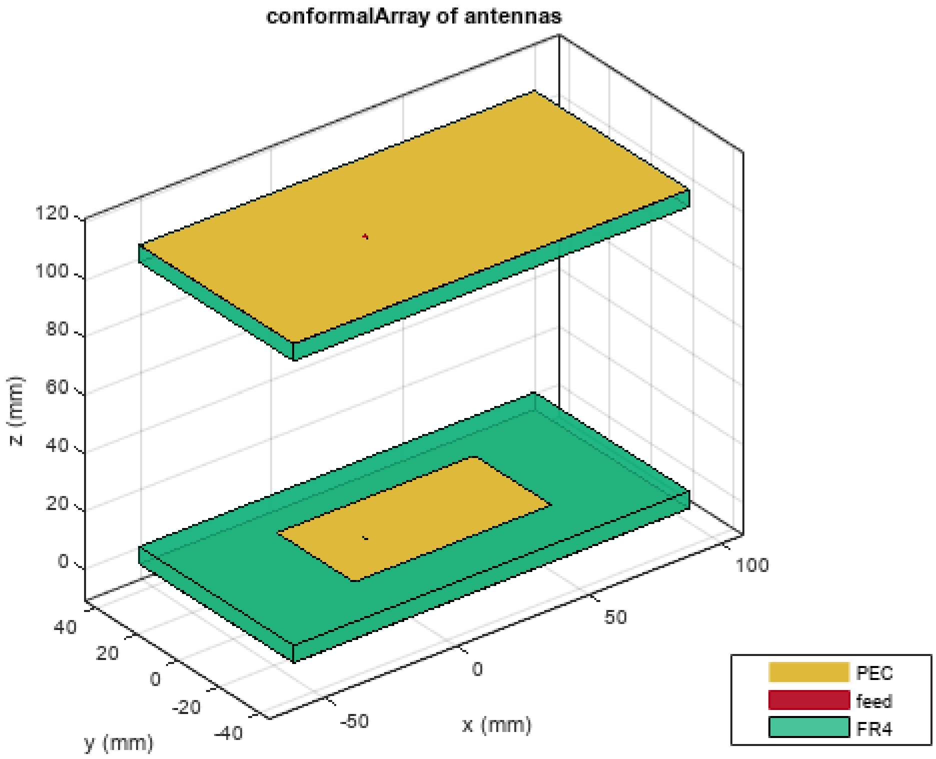
```
d = dielectric('FR4');
p1 = patchMicrostrip('Substrate',d);
p2 = patchMicrostrip('Substrate',d,'Tilt',180);
```

Create and view a conformal array using the two patch microstrip antennas placed 11 cm apart.

```
c = conformalArray('ElementPosition',[0 0 0;0 0 0.1100],'Element',{p1,p2})
```

```
c = 
  conformalArray with properties:

            Element: {[1x1 patchMicrostrip]  [1x1 patchMicrostrip]}
    ElementPosition: [2x3 double]
          Reference: 'feed'
      AmplitudeTaper: 1
         PhaseShift: 0
               Tilt: 0
           TiltAxis: [1 0 0]
```

```
show(c)
```



Conformal Array Using Balanced and Unbalanced Antennas

Create a conformal array using dipole and monopole antennas.

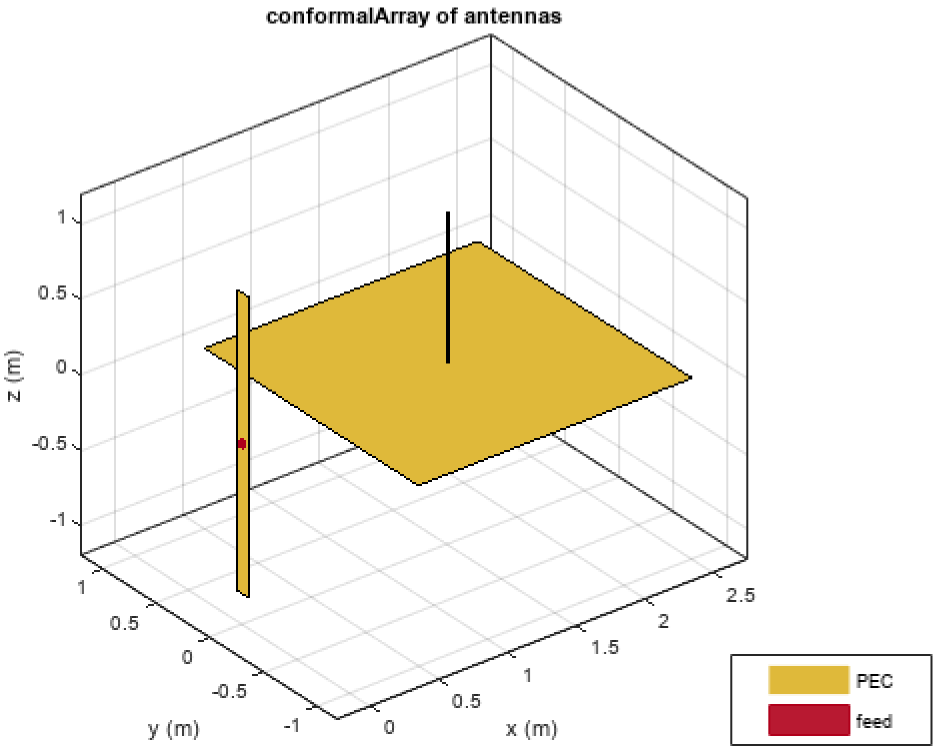
```
c = conformalArray('Element', {dipole, monopole})
```

```
c = 
  conformalArray with properties:

            Element: {[1x1 dipole]  [1x1 monopole]}
    ElementPosition: [2x3 double]
          Reference: 'feed'
     AmplitudeTaper: 1
         PhaseShift: 0
               Tilt: 0
           TiltAxis: [1 0 0]
```

```
c.ElementPosition = [0 0 0; 1.5 0 0];
```

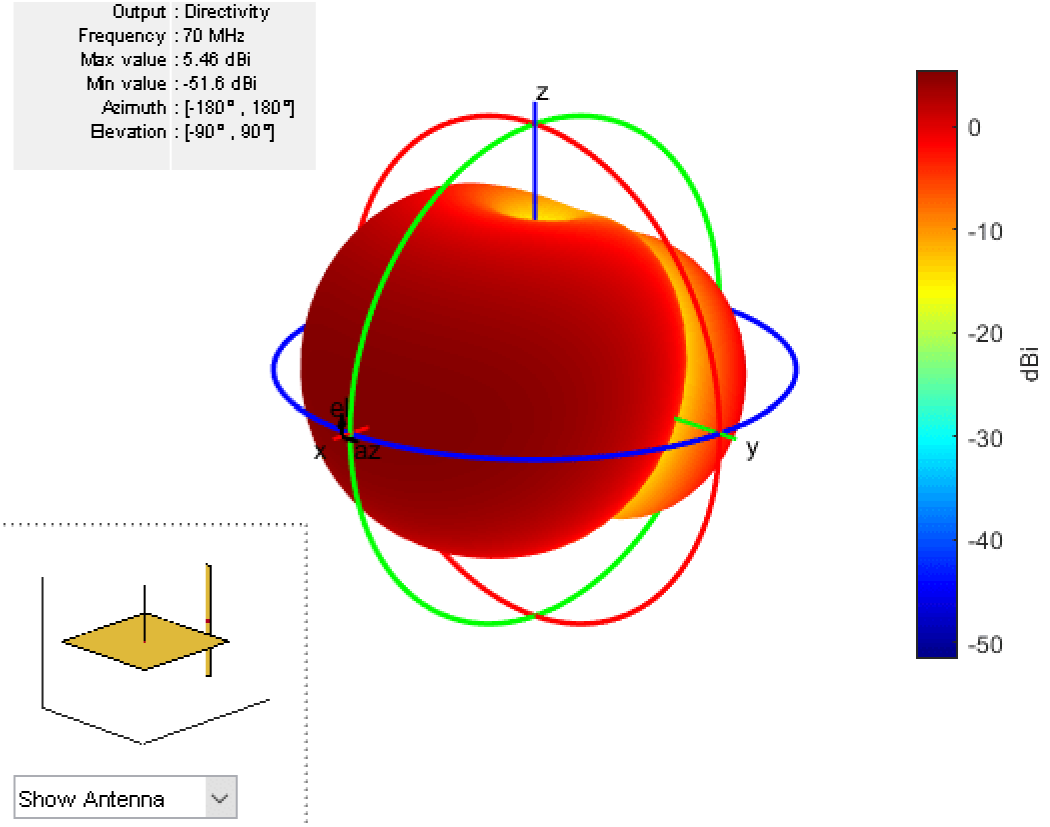
Visualize the array.

```
figure;
show(c);
```



Plot the radiation pattern of the array at 70 MHz.

```
pattern(c, 70e6)
```



```
     Output : Directivity
  Frequency : 70 MHz
  Max value : 5.46 dBi
  Min value : -51.6 dBi
    Azimuth : [-180° , 180°]
  Elevation : [-90° , 90°]
```

Subarrays of Linear Arrays

Create a subarray of linear arrays at different locations.

```
la = linearArray('ElementSpacing',1)
```

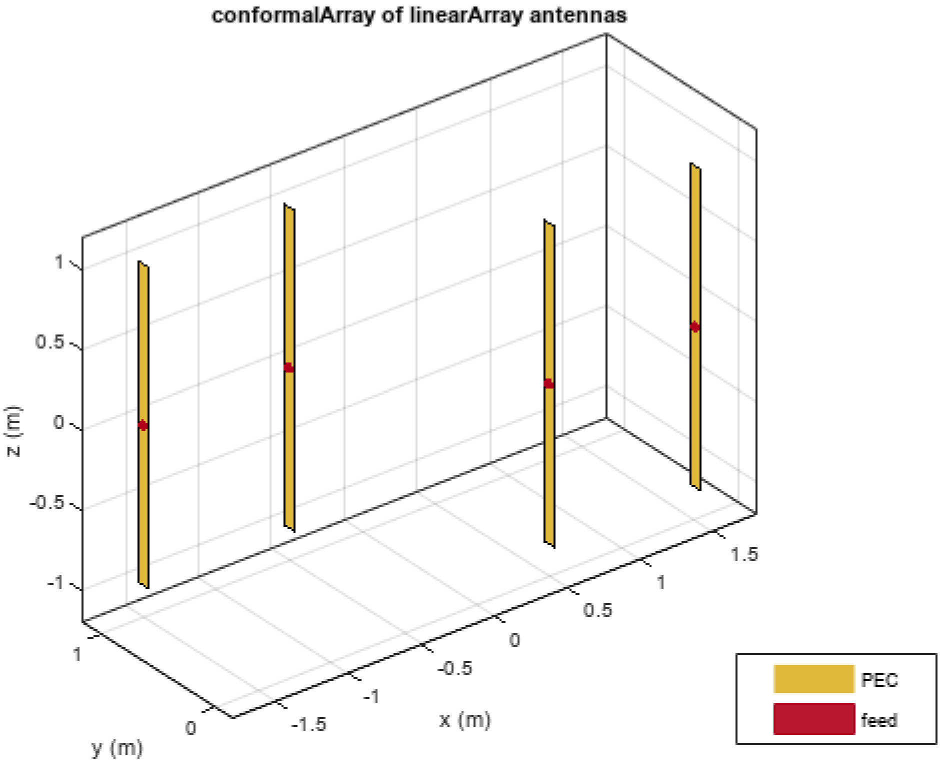
```
la = 
  linearArray with properties:

           Element: [1x1 dipole]
       NumElements: 2
    ElementSpacing: 1
     AmplitudeTaper: 1
        PhaseShift: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

```
subArr = conformalArray('Element',[la la],'ElementPosition',[1 0 0;-1 1 0])
```

```
subArr = 
  conformalArray with properties:

            Element: [1x2 linearArray]
    ElementPosition: [2x3 double]
```

```
       Reference: 'feed'
   AmplitudeTaper: 1
      PhaseShift: 0
           Tilt: 0
        TiltAxis: [1 0 0]
```

```
show(subArr)
```



Conformal Array of Subarrays and Antennas

Create a linear array of dipoles with and element spacing of 1m.

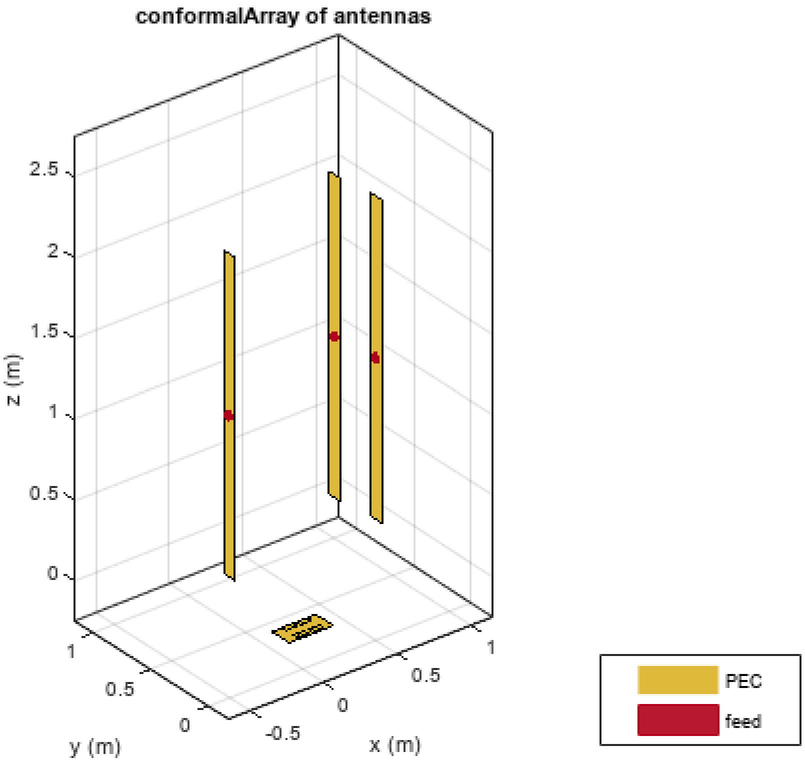
```
la = linearArray('ElementSpacing',1);
```

Create a rectangular array of microstrip patch antennas.

```
ra = rectangularArray('Element',patchMicrostrip,'RowSpacing',0.1,'ColumnSpacing',0.1);
```

Create a subarray containing the above linear and rectangular arrays with changes in amplitude taper and phase shift values.

```
subArr = conformalArray('Element',{la ra dipole},'ElementPosition',[0 0 1.5;0 0 0;1 1 1],...
    'AmplitudeTaper',[3 0.3 0.03],'PhaseShift',[90 180 120]);
show(subArr)
```

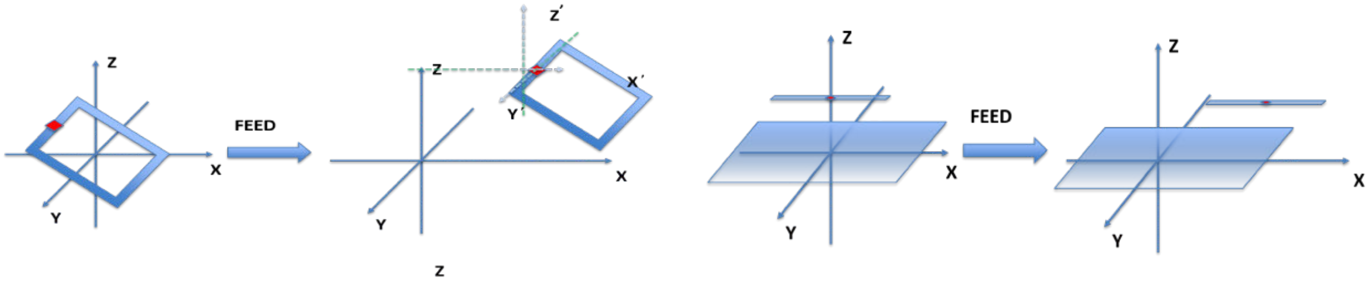


More About

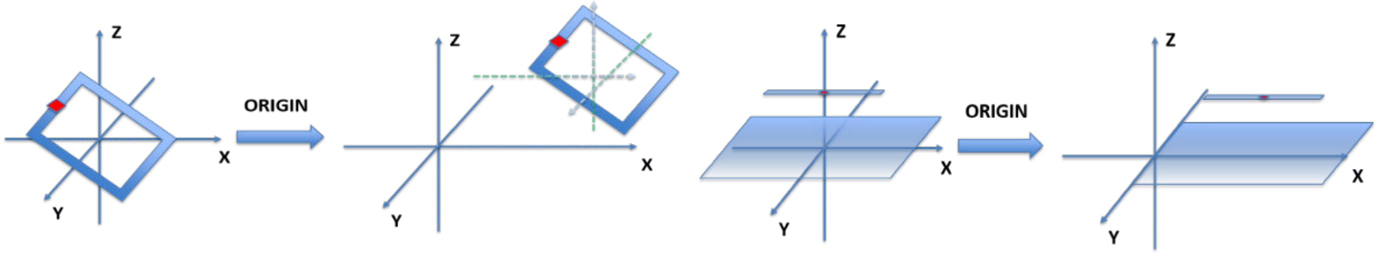
Position Reference

`'Reference'` property of `conformalArray` class defines the position reference of an antenna element in 3–D space. You can position the antenna by specifying the `Reference` property as `feed` or `origin`.

Choosing `feed` as the position reference moves the antenna element with so that the new feed location is at the specified coordinates. The loop rectangle antenna and reflector-backed antenna show the new position with respect to feed:



Choosing `origin` as the position reference moves the antenna element so that new antenna origin is at the specified coordinates. The loop rectangle antenna and reflector-backed antenna show the new position with respect to origin:



Version History

Introduced in R2016a

References

[1] Balanis, Constantine A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: John Wiley and Sons, 2005.

See Also

`linearArray` | `rectangularArray` | `circularArray` | `infiniteArray`

Topics

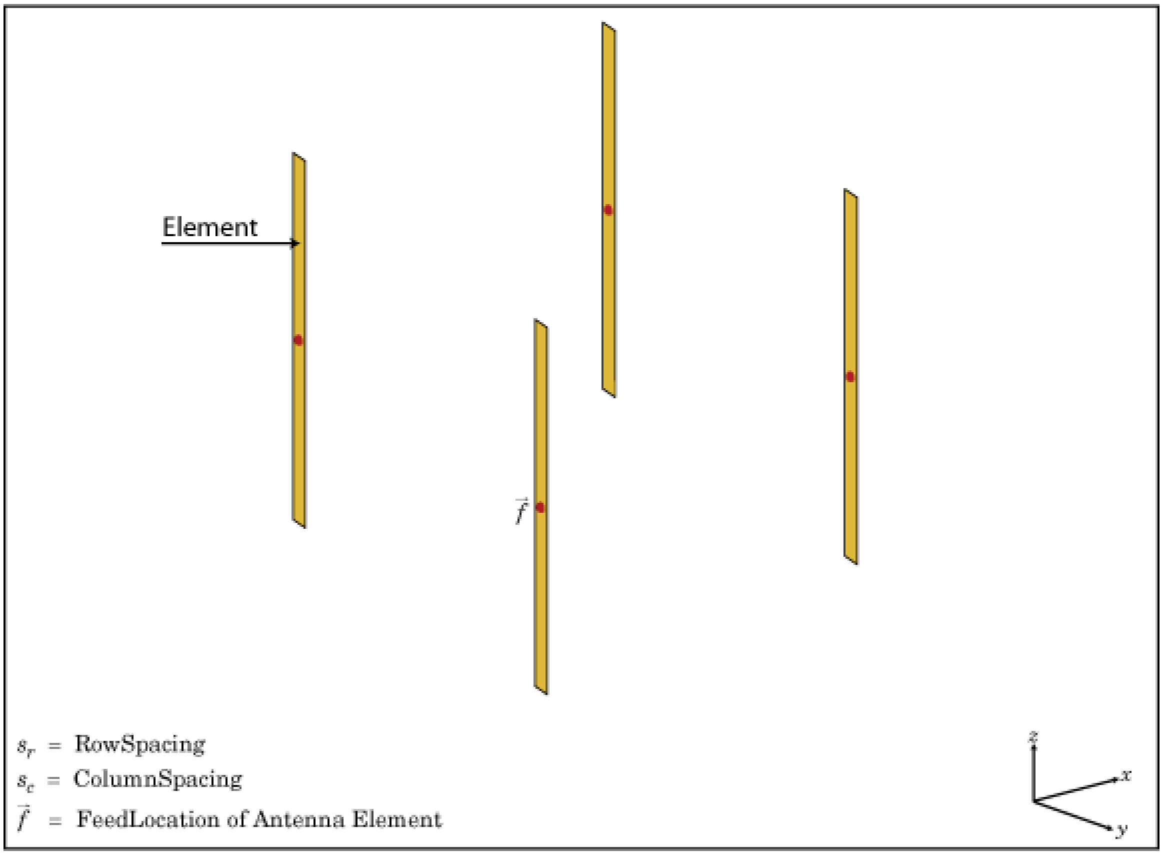
“Rotate Antennas and Arrays”

rectangularArray

Create rectangular antenna array

Description

The `rectangularArray` class creates a rectangular antenna array in the X-Y plane. By default, the rectangular array is a four-element dipole array in a 2 x 2 rectangular lattice. The dipoles are center-fed. Each dipole resonates at 70 MHz when isolated.



Creation

Syntax

```
array = rectangularArray
array = rectangularArray(Name,Value)
```

Description

`array = rectangularArray` creates a rectangular antenna array in the X-Y plane.

`array = rectangularArray(Name,Value)` creates a rectangular antenna array, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain default values.

Output Arguments

`array` — Rectangular array
`rectangularArray` object

Rectangular array, returned as an `rectangularArray` object.

Properties

`Element` — Antenna elements or linear arrays
dipole (default) | antenna object | array object

Antenna elements or linear arrays, specified as an antenna or array object.

Example: `'Element',monopole`

`Size` — Number of antenna elements in row and column of array
`[2 2]` (default) | two-element vector

Number of antenna elements in row and column of array, specified as a two-element vector.

Example: `'Size',[4 4]`

`RowSpacing` — Row spacing between two antenna elements
2 (default) | scalar | vector

Row spacing between two antenna elements, specified as a scalar or vector in meters. By default, the antenna elements are spaced `2m` apart.

Example: `'RowSpacing'`,[5 6]

Data Types: `double`

`ColumnSpacing` — Column spacing between two antenna elements
2 (default) | scalar | vector

Column spacing between two antenna elements, specified as a scalar or vector in meters. By default, the antenna elements are spaced `2m` apart.

Example: `'ColumnSpacing',[3 4]`

Data Types: `double`

`Lattice` — Antenna elements spatial arrangement
`'Rectangular'` (default) | `"Triangular"`

Antenna elements spatial arrangement, specified as a text input.

Example: `'Lattice'`,`"Triangular"`

Data Types: `char` | `string`

`AmplitudeTaper` — Excitation amplitude of antenna elements

1 (default) | scalar | vector

Excitation amplitude of antenna elements, specified as a scalar or vector. Set the property value to `0` to model dead elements.

Example: `'AmplitudeTaper',3`

Data Types: `double`

`PhaseShift` — Phase shift for antenna elements

0 (default) | scalar | vector

Phase shift for antenna elements, specified as a scalar or vector in degrees.

Example: `'PhaseShift',[3 3 0 0]`

Data Types: `double`

`Tilt` — Tilt angle of array

`0` (default) | scalar | vector

Tilt angle of the array specified as a scalar or vector with each element unit in degrees. For more information, see "Rotate Antennas and Arrays".

Example: `Tilt=90`,

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the array at 90 degrees about the two axes, defined by vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of array

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the array, specified as:

- Three-element vectors of Cartesian coordinates in meters. In this case, each vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the array rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see "Rotate Antennas and Arrays".

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis='Z'`

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
beamwidth	Beamwidth of antenna

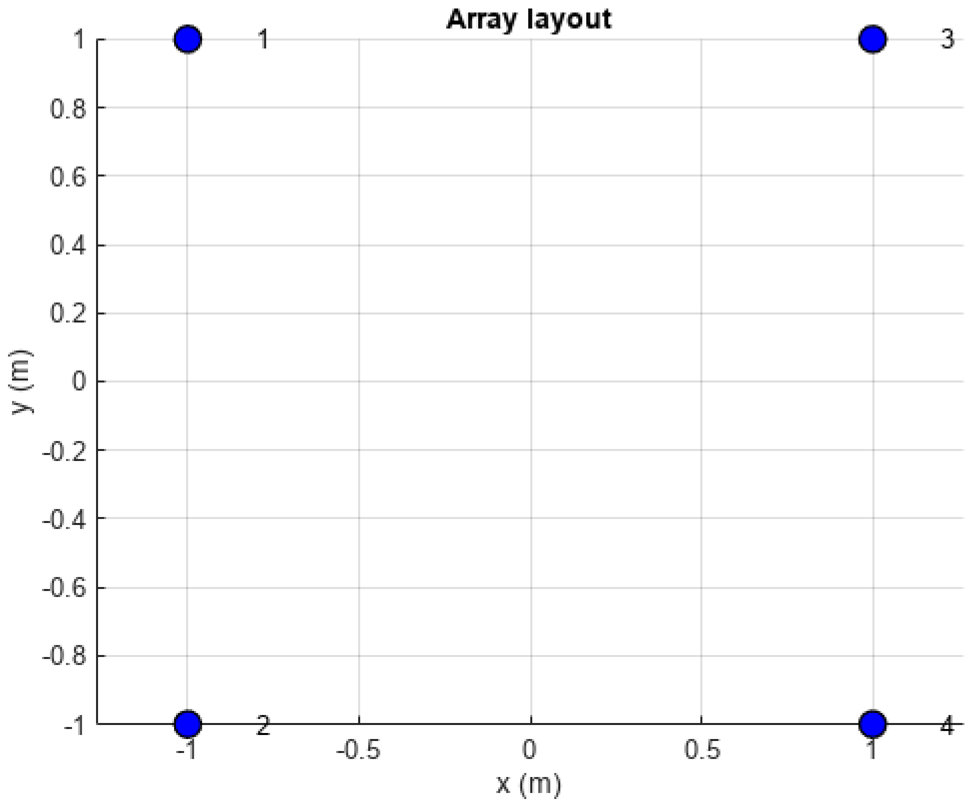
charge	Charge distribution on antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
layout	Display array or PCB stack layout
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
optimize	Optimize antenna or array using SADEA optimizer
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

Examples

Create and Plot Layout of Rectangular Array

Create and plot the layout of a rectangular array of four dipoles.

```
ra = rectangularArray;
ra.Size = [2 2];
layout(ra);
```



Calculate Scan Impedance of Rectangular Array

Calculate the scan impedance of a 2x2 rectangular array of dipoles at 70 MHz.

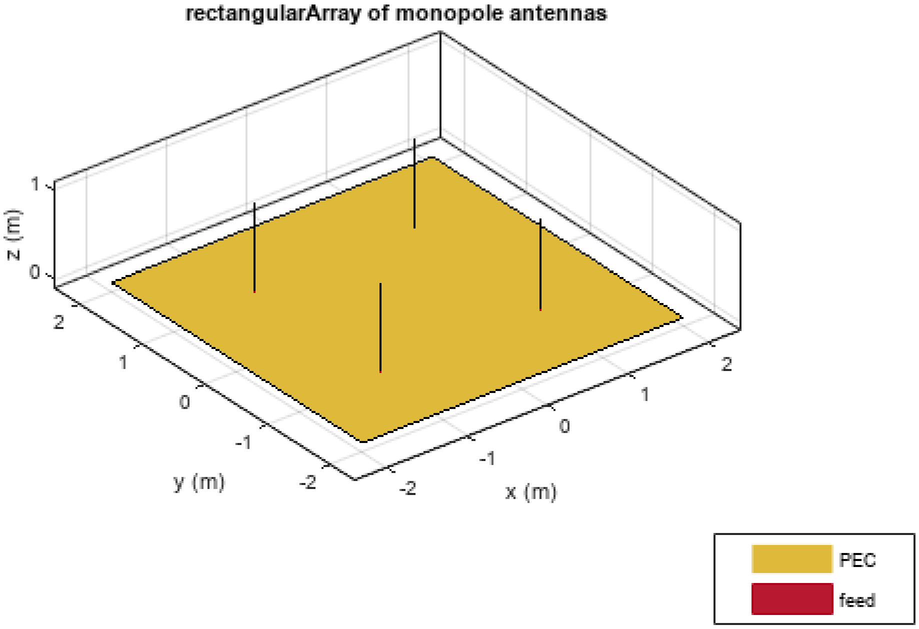
```
h = rectangularArray('Size',[2 2]);
Z = impedance(h,70e6)
```

```
Z = 1×4 complex

  26.7086 -56.9363i  26.6918 -56.9498i  26.7086 -56.9363i  26.6918 -56.9498i
```

Rectangular Array Using Groundplane Antennas

Create a rectangular array of monopoles.

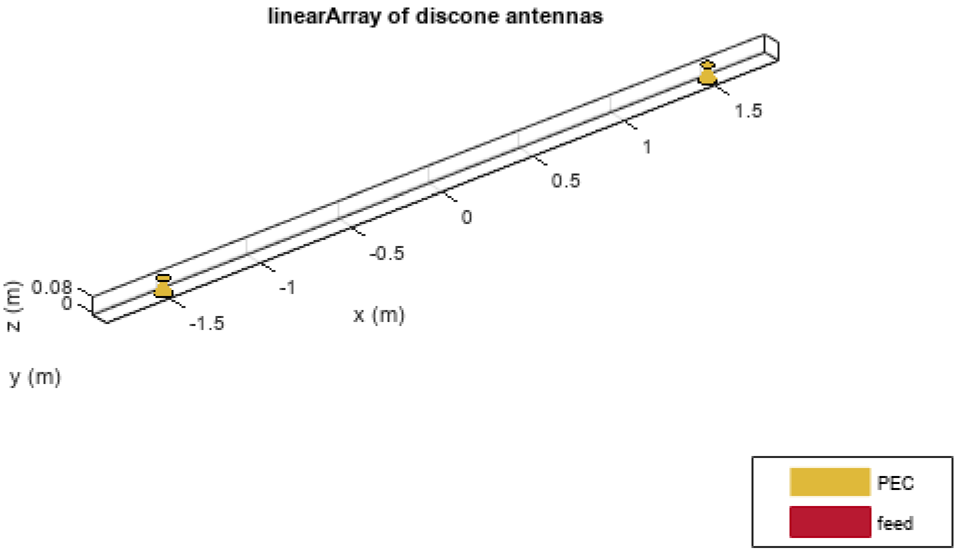
```
m1 = monopole;
mra = rectangularArray('Element',m1);
show(mra);
```



Rectangular Array of Linear Array

Create an array of discones with element spacing of 3 m.

```
la = linearArray('Element',discone);
la.ElementSpacing = 3;
show(la)
```

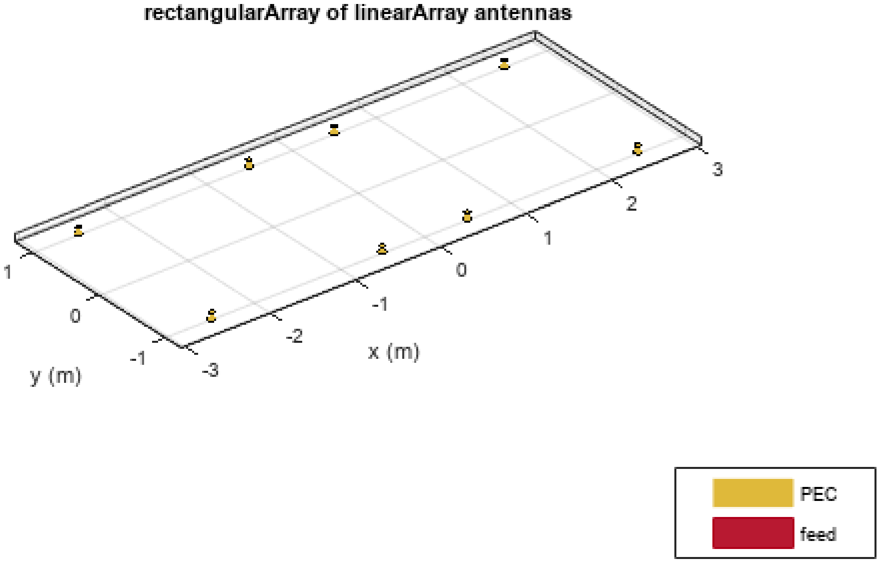


Create a rectangular of the linear array.

```
ra = rectangularArray("Element",la)
```

```
ra = 
  rectangularArray with properties:

           Element: [1x1 linearArray]
              Size: [2 2]
        RowSpacing: 2
     ColumnSpacing: 2
           Lattice: 'Rectangular'
    AmplitudeTaper: 1
        PhaseShift: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

```
show(ra)
```



Version History

Introduced in R2015a

References

[1] Balanis, C.A. *Antenna Theory. Analysis and Design*, 3rd Ed. New York: Wiley, 2005.

See Also

`linearArray` | `conformalArray` | `infiniteArray` | `circularArray`

Topics

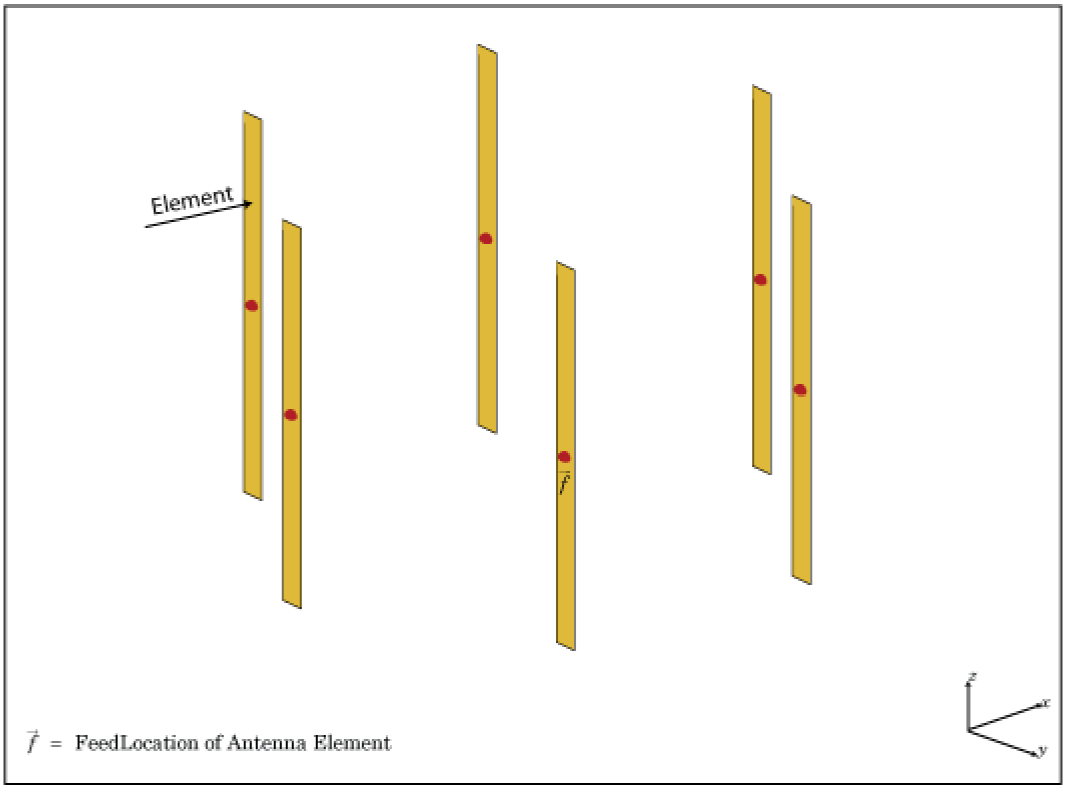
"Rotate Antennas and Arrays"

circularArray

Create circular antenna array

Description

The `circularArray` object is a circular antenna array. Circular array finds application in direction of arrival (DoA) systems. You can use circular arrays to perform 2-D scanning, while lowering element counts. These arrays also have the ability for 360-degree scanning. The individual elements in the circular array are part of the same array environment. This property reduces the impact of edge effects and other coupling variation.



Creation

Syntax

```
array = circularArray
array = circularArray(Name,Value)
```

Description

`array = circularArray` creates a circular antenna array in the X-Y plane.

`array = circularArray(Name,Value)` class to create a circular antenna array, with additional properties specified by one, or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1,...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Element` — Individual antenna type

`dipole` (default) | vector of objects

Individual antenna type, specified as a vector of objects. This property supports scalar expansion.

Example: `'Element',[monopole,monopole]`

Data Types: `char` | `string`

`NumElements` — Number of elements in array

`6` (default) | positive scalar integer

Number of elements in the array,specified as a positive scalar integer. The elements in the array are arranged along the X-axis.

Example: `'NumElements',4`

Data Types: `double`

`Radius` — Radius of array

`1` (default) | positive scalar integer

Radius of array, specified as a positive scalar integer in meters.

Example: `'Radius',0.4`

Data Types: `double`

`AngleOffset` — Starting angle offset for first element in array

`0` (default) | real scalar

Starting angle offset for first element in array, specified as a real scalar in degrees.

Example: `'AngleOffset',8`

Data Types: `double`

`AmplitudeTaper` — Excitation amplitude for antenna elements in array

`1` (default) | real positive vector of size `'Element'`

Excitation amplitude for antenna elements in the array, specified as a real positive vector of size `'Element'`.

Example: `'AmplitudeTaper',[0 1]`

Data Types: `double`

`PhaseShift` — Phase shift for each element in array

`0` (default) | real vector of size `'Element'` in degrees

Phase shift for each element in the array, specified as a real vector of size `'Element'` in degrees.

Example: `'PhaseShift',[0 2]`

Data Types: `double`

`Tilt` — Tilt angle of array

`0` (default) | scalar | vector

Tilt angle of the array specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`,

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the array at 90 degrees about the two axes, defined by vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of array

`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the array, specified as:

- Three-element vectors of Cartesian coordinates in meters. In this case, each vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the array rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis='Z'`

Data Types: `double`

Analysis Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
layout	Display array or PCB stack layout
show	Display antenna, array structures or shapes
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on antenna or array surface
design	Design prototype antenna or arrays for resonance around specified frequency
efficiency	Radiation efficiency of antenna
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
optimize	Optimize antenna or array using SADEA optimizer

pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

Examples

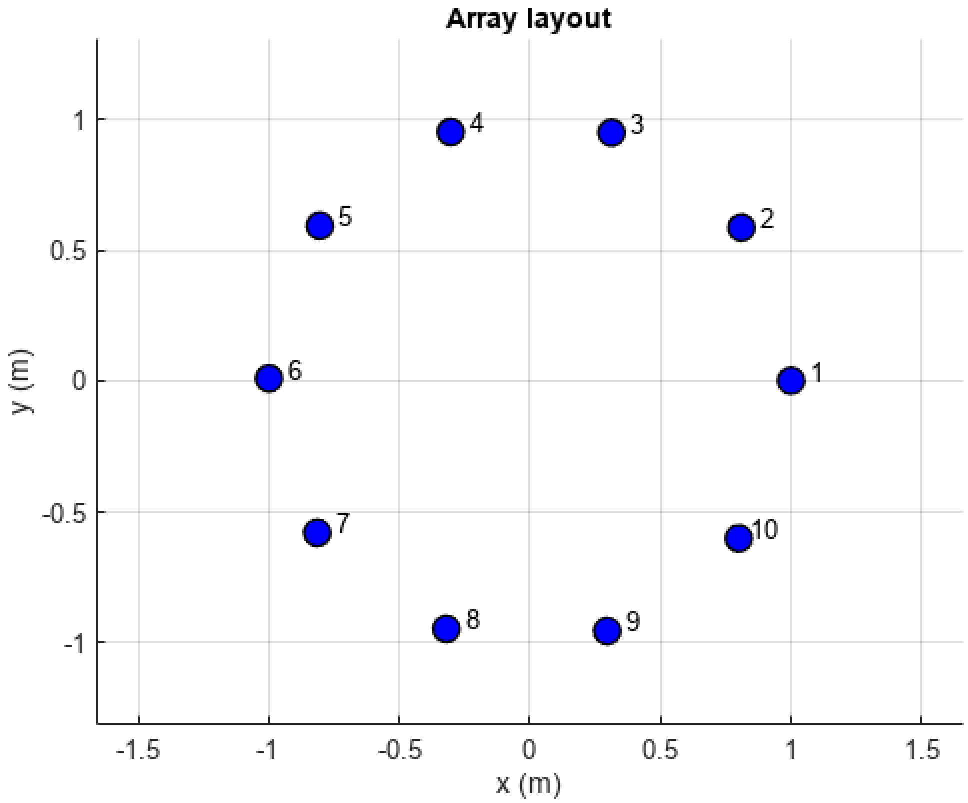
Plot Elevation Pattern of Circular Antenna Array

Create a circular antenna array using 10 antenna elements. View the layout of the antenna elements in the array.

```
ca = circularArray('NumElements',10)
```

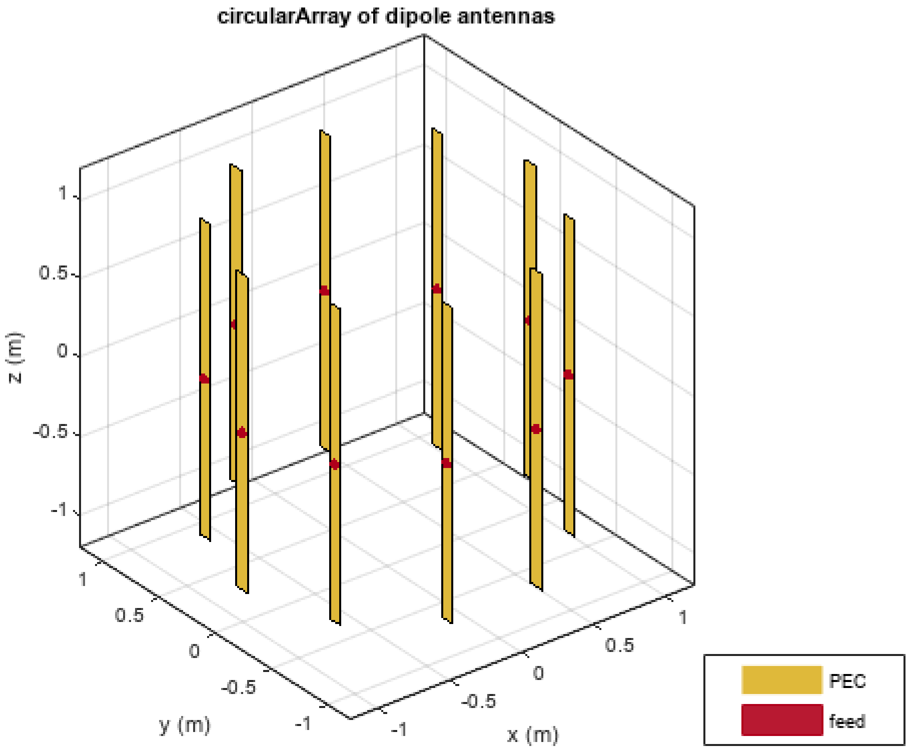
```
ca = 
  circularArray with properties:

           Element: [1x1 dipole]
       NumElements: 10
            Radius: 1
       AngleOffset: 0
    AmplitudeTaper: 1
        PhaseShift: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

```
figure;
layout(ca)
```



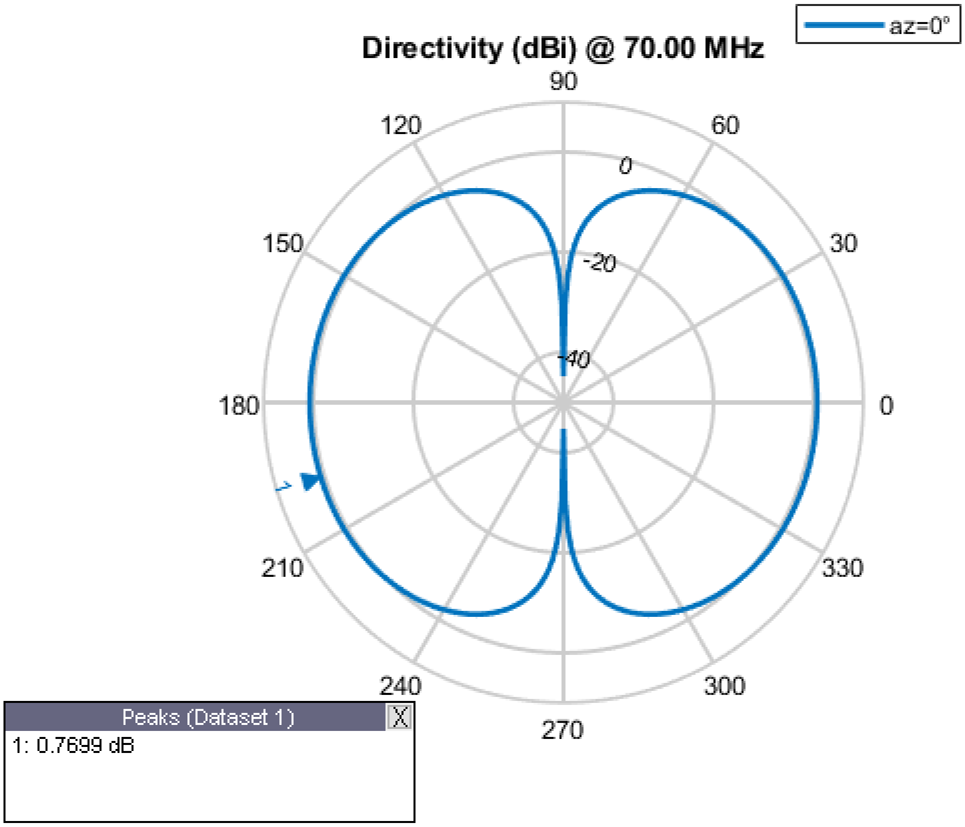
Display the array.

```
figure;
show(ca)
```



Plot the elevation pattern of the circular array at a frequency of 70 MHz.

```
figure;
patternElevation(ca,70e6)
```



Version History

Introduced in R2016b

See Also

`linearArray` | `rectangularArray` | `conformalArray` | `infiniteArray`

Topics

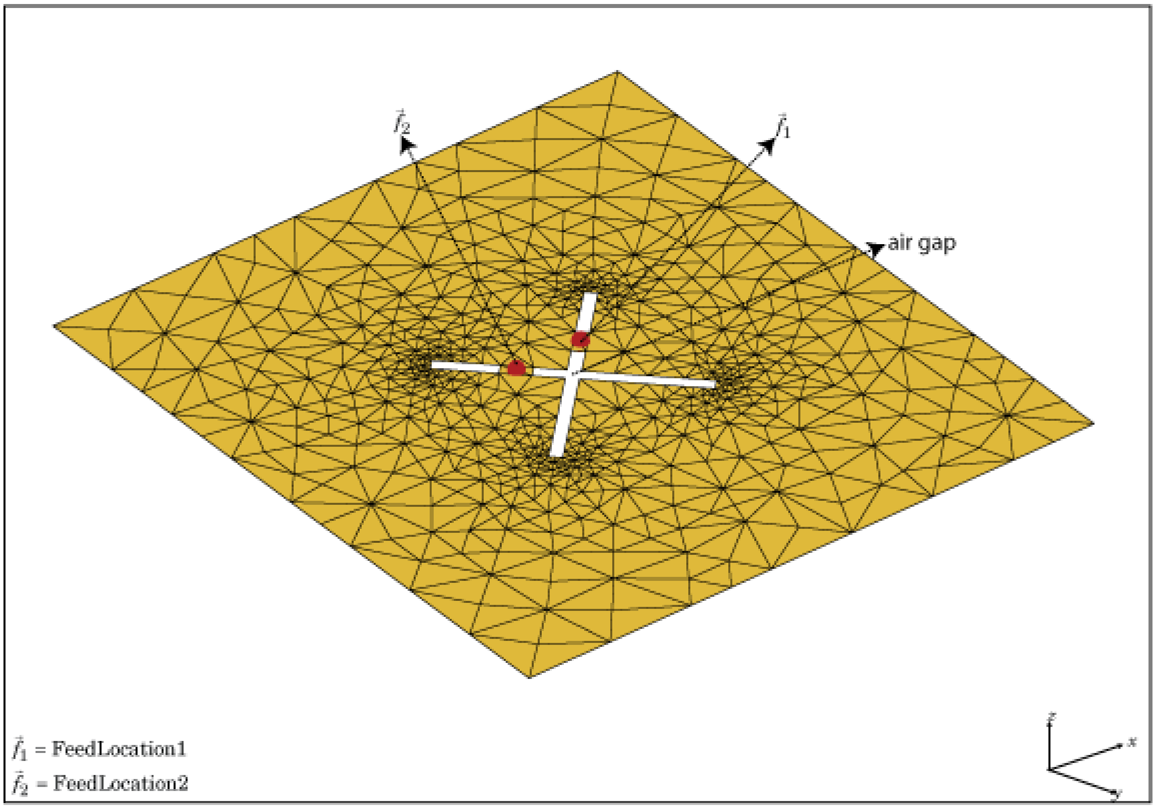
"Rotate Antennas and Arrays"

customArrayMesh

Create 2-D custom mesh antenna array on X-Y plane

Description

The `customArrayMesh` object creates an array represented by a 2-D custom mesh on the X-Y plane. You can provide an arbitrary array mesh to the Antenna Toolbox and analyze this mesh as a custom array for port and field characteristics.



Creation

Description

`customarray = customArrayMesh(points,triangles,numfeeds)` creates a 2-D array represented by a custom mesh, based on the specified points and triangles.

Input Arguments

`points` — Points in custom mesh
`2`-by-*`N`* or `3`-by-*`N`* matrix of Cartesian coordinates in meters

Points in custom mesh, specified as a `2-by-N` or `3-by-N` matrix of Cartesian coordinates in meters. *N* is the number of points. In case you specify a `3-by-N` integer matrix, the Z-coordinate must be zero or a constant value. This value sets the `'Points'` property in the custom array mesh.

Example: `load planarmesh.mat`; `c = customArrayMesh(p,t,4)`. Creates a custom array mesh from the points, `p`, extracted from the `planarmesh.mat` file.

Data Types: `double`

`triangles` — Triangles in mesh
`4-by-M` matrix

Triangles in the mesh, specified as a `4-by-M` matrix. *M* is the number of triangles. The first three rows are indices to the points matrix and represent the vertices of each triangle. The fourth row is a domain number useful for identifying separate parts of an array. This value sets the `'Triangles'` property in the custom array mesh.

Example: `load planarmesh.mat`; `c = customArrayMesh(p,t,4)`. Creates a custom array mesh from the triangles, `t`, extracted from the `planarmesh.mat` file.

Data Types: `double`

`numfeeds` — Number of feeding points in array
2 (default) | scalar

Number of feeding points in array, specified as a scalar. By default, the number of feed points are 2.

Example: `load planarmesh.mat`; `c = customArrayMesh(p,t,4)`. Creates a custom array mesh requiring 4 feed points.

Data Types: `double`

Properties

`Points` — Points in custom mesh
`2-by-N` or `3-by-N` matrix of Cartesian coordinates

Points in a custom mesh, specified as a `2-by-N` or `3-by-N` matrix of Cartesian coordinates in meters. *N* is the number of points.

Data Types: `double`

`Triangles` — Triangles in mesh
`4-by-M` matrix

Triangles in the mesh, specified as a `4-by-M` matrix. *M* is the number of triangles.

Data Types: `double`

`NumFeeds` — Number of feeding points
scalar

Number of feeding points in the array, specified as a scalar.

Data Types: `double`

`FeedLocation` — Feed location of array
cartesian coordinates

Feed locations of array, specified as Cartesian coordinates in meters. Feed location is a read-only property. To create a feed for the 2–D custom mesh, use the `createFeed` method.

Data Types: `double`

`AmplitudeTaper` — Excitation amplitude of antenna elements

1 (default) | scalar | non-negative vector

Excitation amplitude of antenna elements, specified as a scalar or a non-negative vector. Set the property value to `0` to model dead elements.

Example: `'AmplitudeTaper'`,3

Data Types: `double`

`PhaseShift` — Phase shift for antenna elements

0 (default) | scalar | real vector

Phase shift for antenna elements, specified as a scalar or a real vector in degrees.

Example: `'PhaseShift'`,[`3 3 0 0`]. Creates a custom array mesh of four antennas with phase shifts specified.

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
createFeed	Create feed locations for custom array
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects

Examples

Custom Array Mesh Impedance.

Load a custom mesh and create an array.

```
load planarmesh.mat;
c = customArrayMesh(p,t,2);
```

Create feeds for the custom array mesh.

```
createFeed(c,[0.07,0.01],[0.05,0.05], [-0.07,0.01],[-0.05,0.05])
```

Calculate the impedance of the array.

```
Z = impedance(c,1e9)
```

```
Z = 1×2 complex

  64.3925 - 7.8288i  58.9598 -11.3558i
```

Version History

Introduced in R2015b

References

[1] Balanis, C.A. *Antenna Theory: Analysis and Design*. 3rd Ed. New York: Wiley, 2005.

See Also

`linearArray` | `rectangularArray` | `conformalArray`

Topics

“Rotate Antennas and Arrays”

customArrayGeometry

Create array represented by 2-D custom geometry

Description

The `customArrayGeometry` object is an array represented by a 2-D custom geometry on the X-Y plane. You can use the `customArrayGeometry` to import a 2-D custom geometry, define feeds to create an array element, and analyze the custom array.

Creation

Syntax

```
array = customArrayGeometry
array = customArrayGeometry(Name,Value)
```

Description

`array = customArrayGeometry` creates a custom array represented by 2-D geometry on the X-Y plane, based on the specified boundary.

`array = customArrayGeometry(Name,Value)` creates a 2-D array geometry, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Output Arguments

`array` — Custom array geometry
`customArrayGeometry` object

Custom array geometry, returned as an `customArrayGeometry` object.

Properties

`Boundary` — Boundary information in Cartesian coordinates
cell array

Boundary information in Cartesian coordinates, specified as a cell array in meters.

Data Types: `double`

`Operation` — Boolean operation performed on boundary list
`'P1'` (default) | character vector

Boolean operation performed on the boundary list, specified as a character vector. operation set is; [+, -, *].

Example: `'Operation','P1-P2'`

Data Types: `double`

`FeedLocation` — Array element feed location in Cartesian coordinates
`[0 0 0]` (default) | three-element vector

Array element feed location in Cartesian coordinates, specified as a three-element vector. The three elements represent the X, Y, and Z coordinates respectively.

Example: `'FeedLocation', [0 0.2 0]`

Data Types: `double`

`FeedWidth` — Width of feed for array elements
`0.0100` (default) | scalar

Width of feed for array elements, specified as a scalar in meters.

Example: `'FeedWidth',0.05`

Data Types: `double`

`AmplitudeTaper` — Excitation amplitude for array elements
`1` (default) | non-negative scalar | vector of non-negative scalars

Excitation amplitude for array elements, specified as a non-negative scalar or vector of non-negative scalars. Set property value to `0` to model dead elements.

Example: `'AmplitudeTaper',3`

Data Types: `double`

`PhaseShift` — Phase shift for array elements
`0` (default) | real scalar | real vector

Phase shift for array elements, specified as a real scalar in degrees or a real vector in degrees.

Example: `'PhaseShift'`,`[3 3 0 0]` specified the phase shift for custom array containing four elements.

Data Types: `double`

`Tilt` — Tilt angle of array
`0` (default) | scalar | vector

Tilt angle of the array specified as a scalar or vector with each element unit in degrees. For more information, see “Rotate Antennas and Arrays”.

Example: `Tilt=90`,

Example: `Tilt=[90 90]`,`TiltAxis=[0 1 0;0 1 1]` tilts the array at 90 degrees about the two axes, defined by vectors.

Data Types: `double`

`TiltAxis` — Tilt axis of array
`[1 0 0]` (default) | three-element vector of Cartesian coordinates | two three-element vectors of Cartesian coordinates | `'X'` | `'Y'` | `'Z'`

Tilt axis of the array, specified as:

- Three-element vectors of Cartesian coordinates in meters. In this case, each vector starts at the origin and lies along the specified points on the X-, Y-, and Z-axes.
- Two points in space, each specified as three-element vectors of Cartesian coordinates. In this case, the array rotates around the line joining the two points in space.
- A string input describing simple rotations around one of the principal axes, 'X', 'Y', or 'Z'.

For more information, see “Rotate Antennas and Arrays”.

Example: `TiltAxis=[0 1 0]`

Example: `TiltAxis=[0 0 0;0 1 0]`

Example: `TiltAxis='Z'`

Data Types: `double`

Object Functions

show	Display antenna, array structures or shapes
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on antenna or array surface
current	Current distribution on antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal, dielectric antenna, or array structure
meshconfig	Change mesh mode of antenna structure
pattern	Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array
patternAzimuth	Azimuth pattern of antenna or array
patternElevation	Elevation pattern of antenna or array
returnLoss	Return loss of antenna; scan return loss of array
sparameters	Calculate S-parameter for antenna and antenna array objects
show	Display antenna, array structures or shapes
vswr	Voltage standing wave ratio of antenna

Examples

Create Custom Slot Antenna Array

Create and visualize a custom array using `pcbStack`. Plot the impedance and current distribution of the array.

Create a ground plane with a length of 0.6 m and a width of 0.5 m.

```
Lp  = 0.6;
Wp  = 0.5;
p1  = antenna.Rectangle('Length',Lp,'Width',Wp);
```

Create slots on the ground plane with a length 0.05 m and a width of 0.4 m.

Add strips of 0.05 m by 0.01 m in between the slots for basing the feed point.

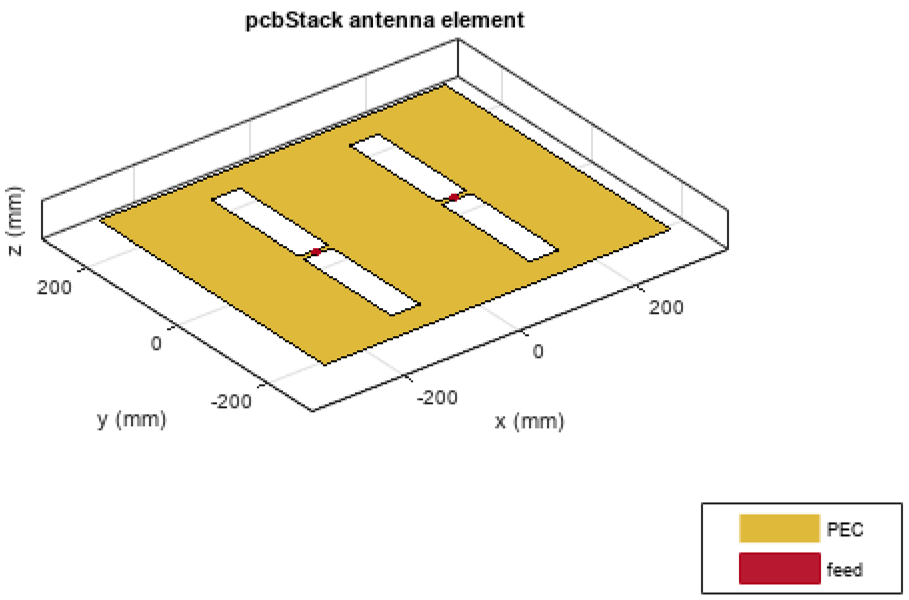
```
Ls  = 0.05;
Ws  = 0.4;
offset = 0.12;
p2 = antenna.Rectangle('Length',Ls,'Width',Ws,'Center',[-offset 0]);
p3 = antenna.Rectangle('Length',Ls,'Width',Ws,'Center',[offset 0]);
Wf  = 0.01;
p4   = antenna.Rectangle('Length',Ls,'Width',Wf,'Center',[-offset 0]);
p5   = antenna.Rectangle('Length',Ls,'Width',Wf,'Center',[offset 0]);
```

Create an array using the slotted ground plane. Create a feed in between the slots on the ground plane.

```
carray = pcbStack;
shape = p1-p2-p3+p4+p5';
carray.BoardShape = antenna.Rectangle('Length',0.6,'Width',0.5);
carray.Layers = {shape};
carray.FeedLocations = [-offset 0 1 ; offset 0 1];
carray.FeedDiameter= 0.01;
```

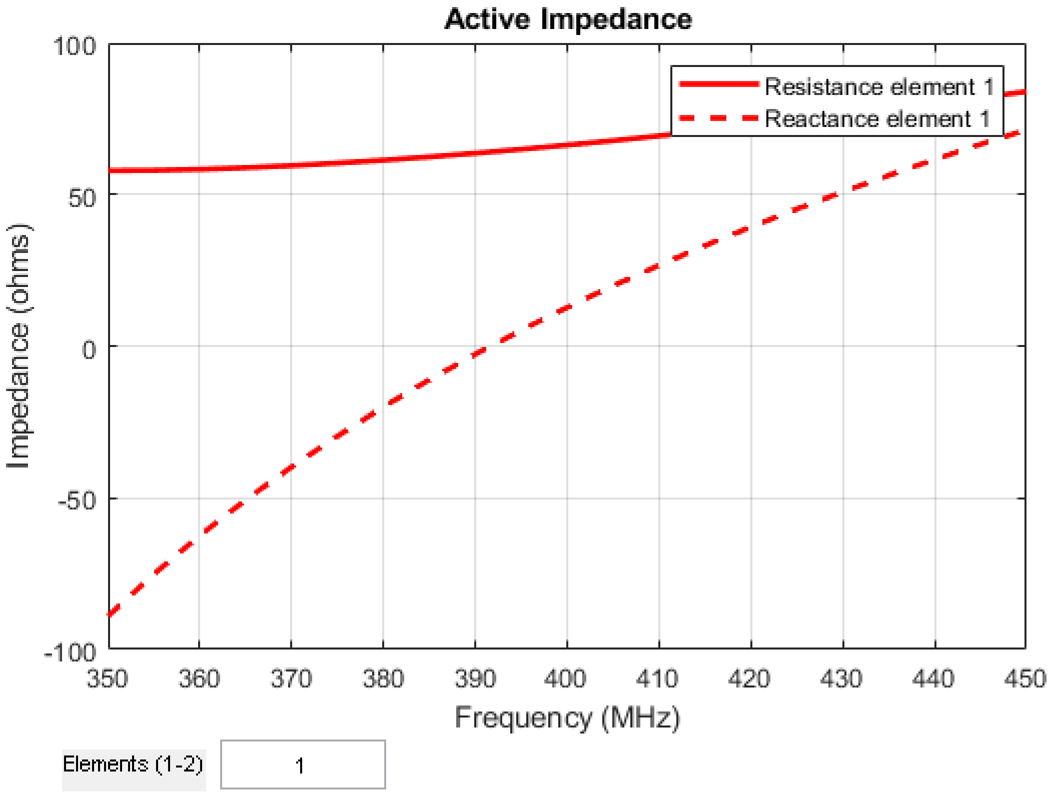
Visualize the array.

```
figure;
show(carray)
```



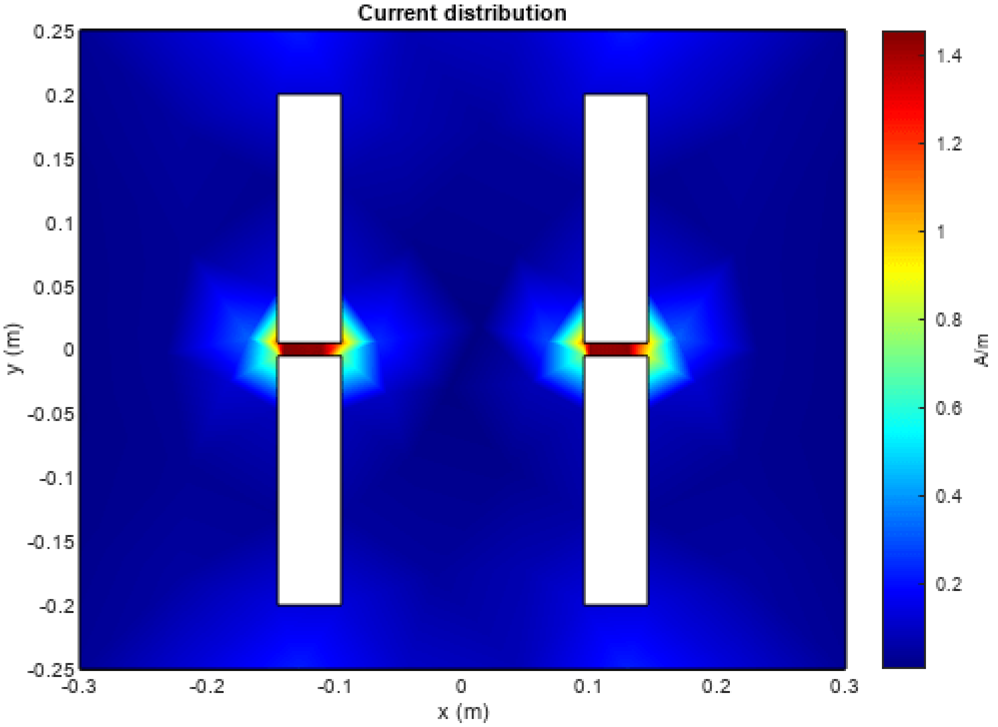
Calculate the impedance of the array using the frequency range of 350 MHz to 450 MHz.

```
figure;
impedance(carray, 350e6:5e6:450e6)
```



Visualize the current distribution of the array at 410 MHz.

```
figure;
current(carray, 410e6)
```



Version History

Introduced in R2017a

References

[1] Balanis, C. A. *Antenna Theory. Analysis and Design*. 3rd Ed. Hoboken, NJ: John Wiley & Sons, 2005.

See Also

Topics

"Rotate Antennas and Arrays"

4

Methods

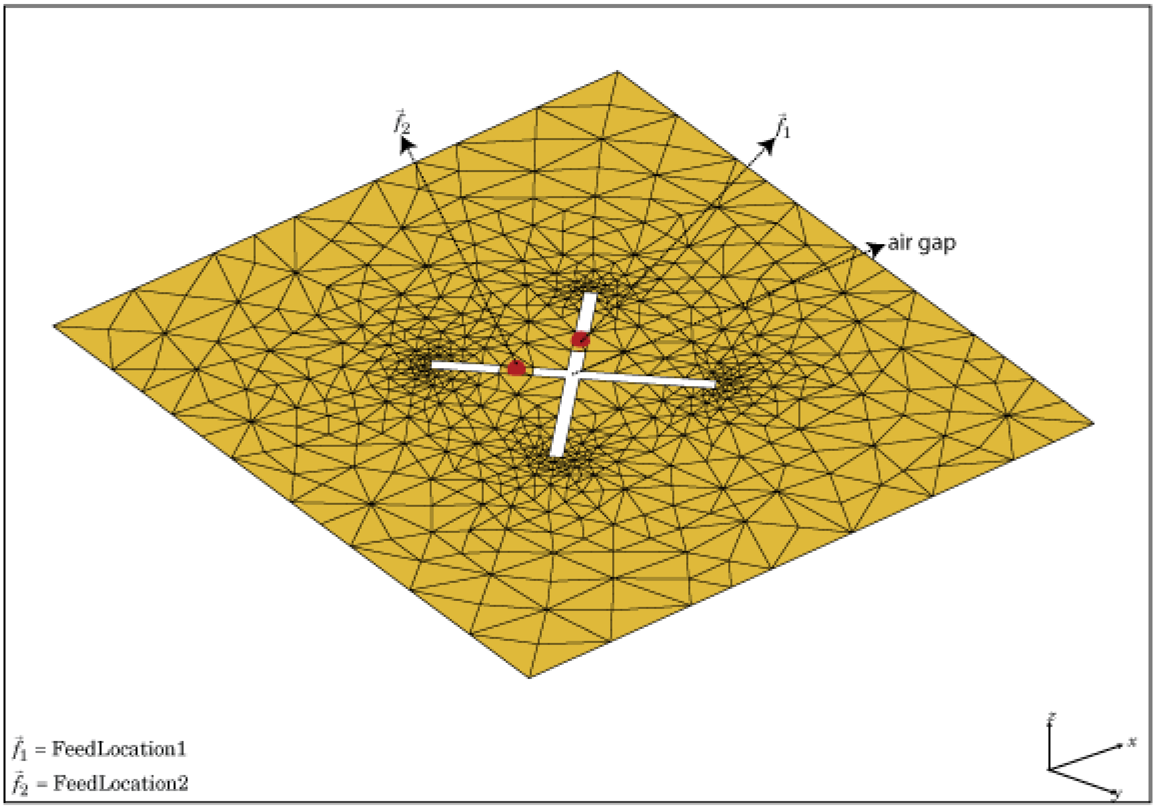
createFeed

Create feed locations for custom array

Syntax

```
createFeed(array)
createFeed(array,point1a,point1b,point2a,point2b,.....)
```

Description



`createFeed(array)` plots a custom array mesh in a figure window. From the figure window, you can specify feed locations by clicking on the mesh and create a custom array. To specify a region for the feed point, select two pairs of points, inside triangles on either side of the air gap.

`createFeed(array,point1a,point1b,point2a,point2b,.....)` creates the feed across the triangle edges identified by pairs of points (`point1a` and `point1b`, `point2a`, and `point2b`). After creating the feed, feed location is highlighted when you plot the resulting array mesh.

Input Arguments

`array` — Custom array mesh

scalar

Custom mesh array, specified as a scalar.

point1a,point1b — Point pairs to identify feed region
Cartesian coordinates in meters

Point pairs to identify feed region, specified as Cartesian coordinates in meters. Specify the points in the format [x_1, y_1], [x_2, y_2].

Example: `createFeed(c,[0.07,0.01],[0.05,0.05],[-0.07,0.01],[-0.05,0.05])`. Creates two pairs of feedpoints for a custom array mesh at the x-y coordinates specified.

Examples

Two-Feed Custom Array Mesh Using GUI

Create a custom array with two feeds.

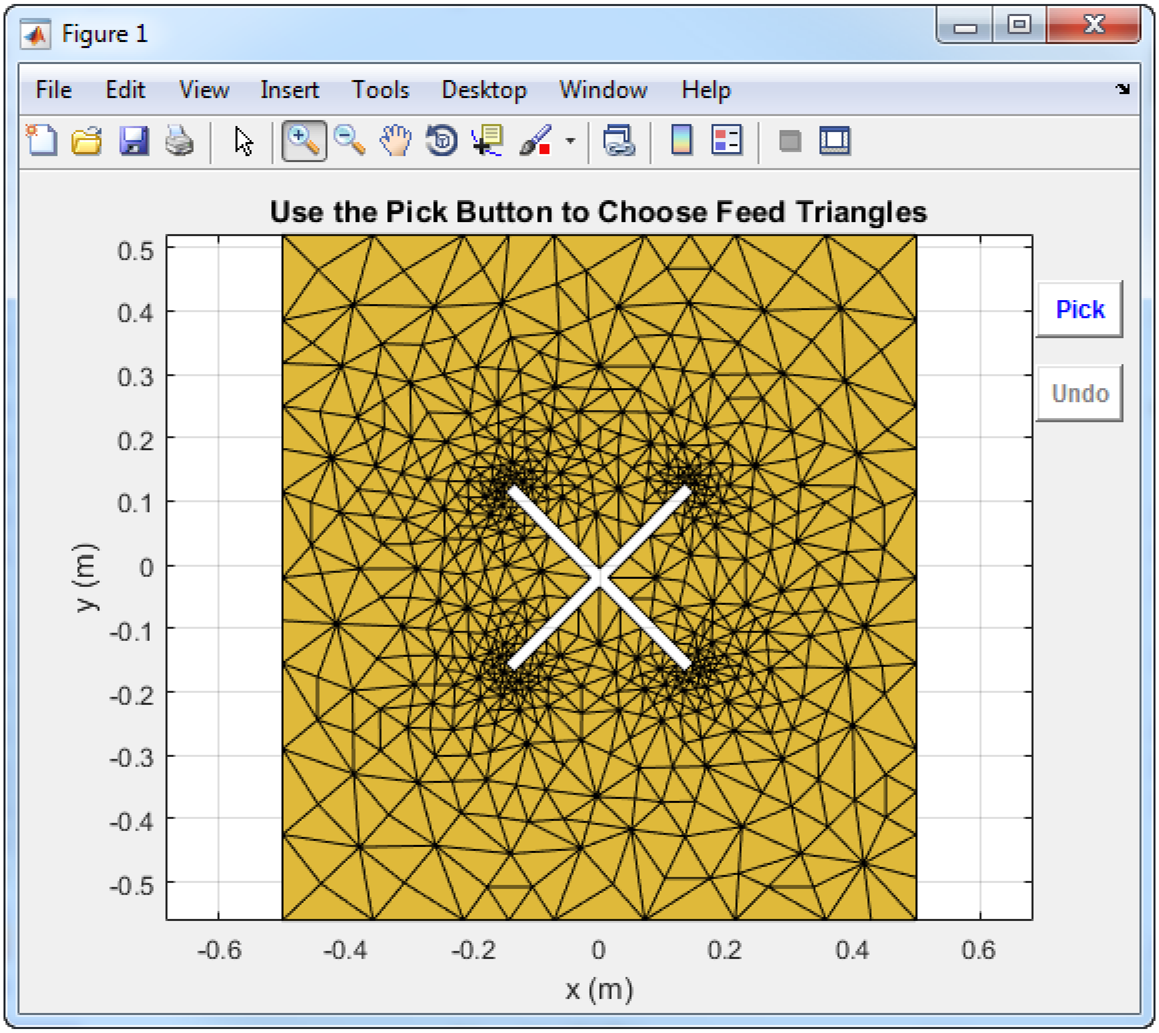
Load a 2-D custom mesh. Create a custom array using the points and triangles.

```
load planarmesh.mat;
c = customArrayMesh(p,t,2);
```

```
c = 
  customArrayMesh with properties:

            Points: [3x658 double]
         Triangles: [4x1219 double]
          NumFeeds: 2
      FeedLocation: []
    AmplitudeTaper: 1
        PhaseShift: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

Use the `createFeed` function to view the array mesh structure. In this array mesh view, you see **Pick** and **Undo** buttons. The **Pick** button is highlighted.

```
createFeed(c)
```



Click **Pick** to display the cross hairs. For an array with two feeds, select two pairs (four points) in the mesh. To specify a feed-region for the, zoom in and select two points each, one inside each triangle on either side of the air gap. Select the points using the cross hairs.

- Select the first triangle for feedpoint 1.

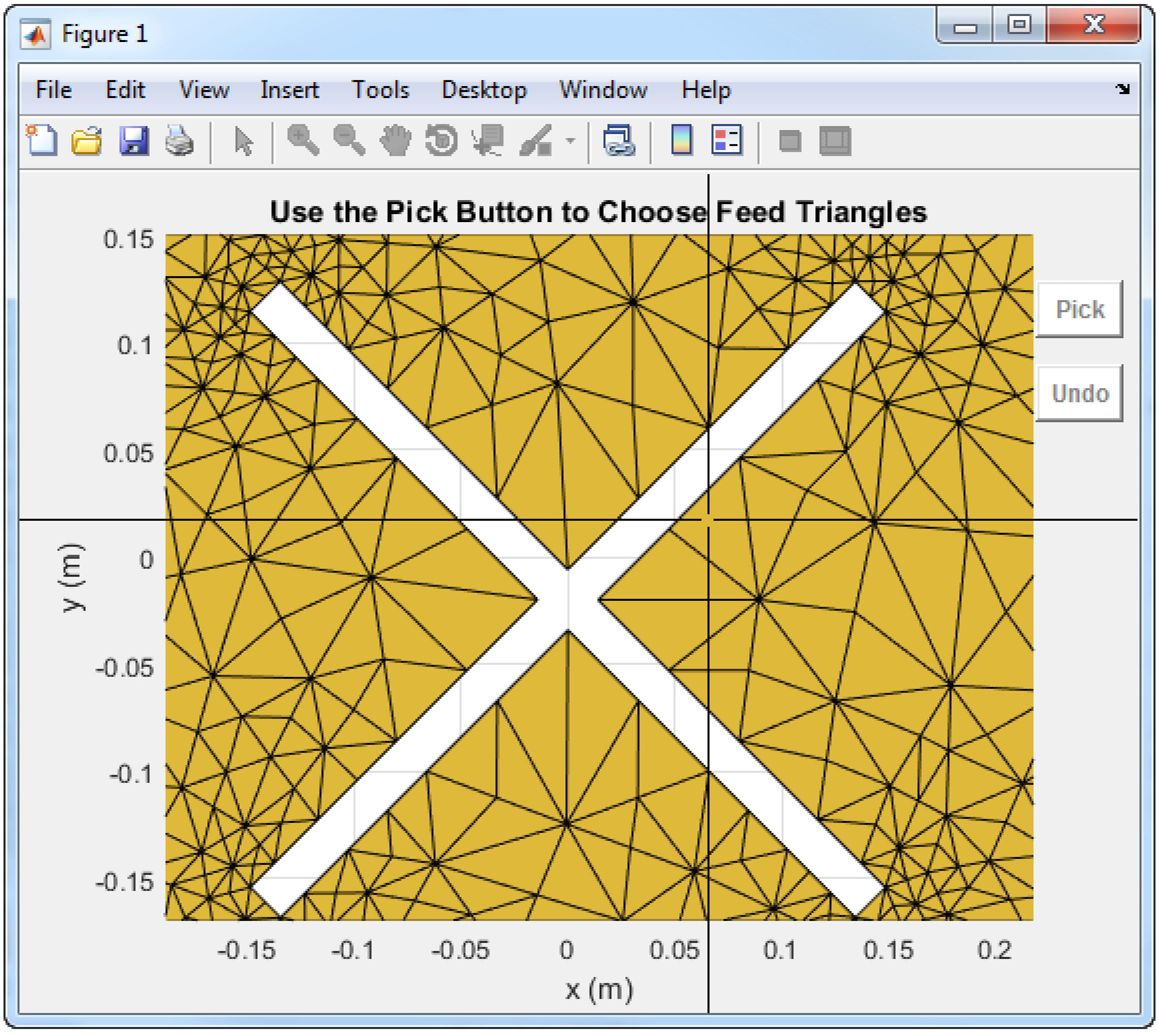


Figure 1
File Edit View Insert Tools Desktop Window Help
Use the Pick Button to Choose Feed Triangles
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
y (m)
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
x (m)
Pick
Undo

- Select the second triangle on the other side of the air gap for feedpoint 1.

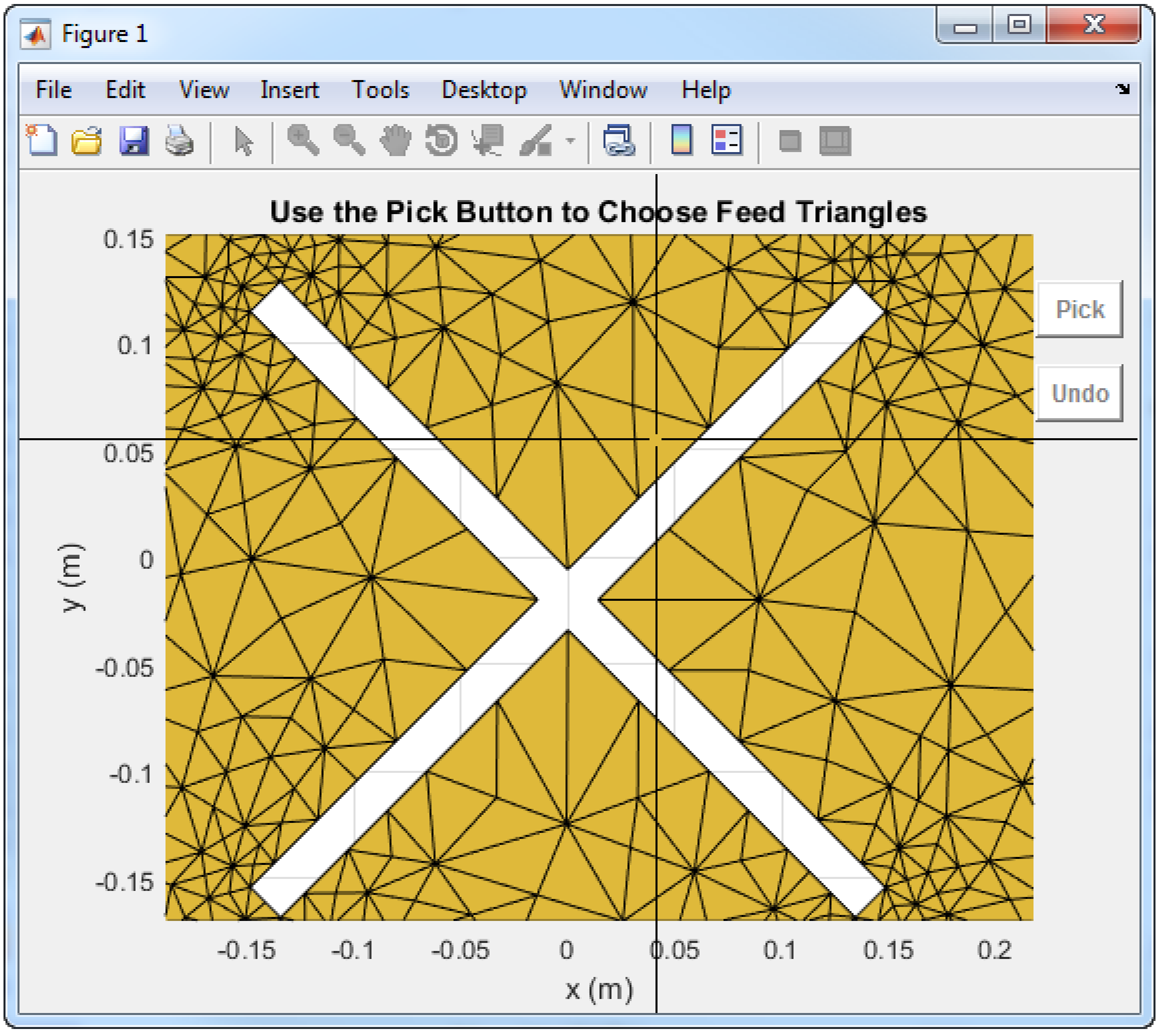
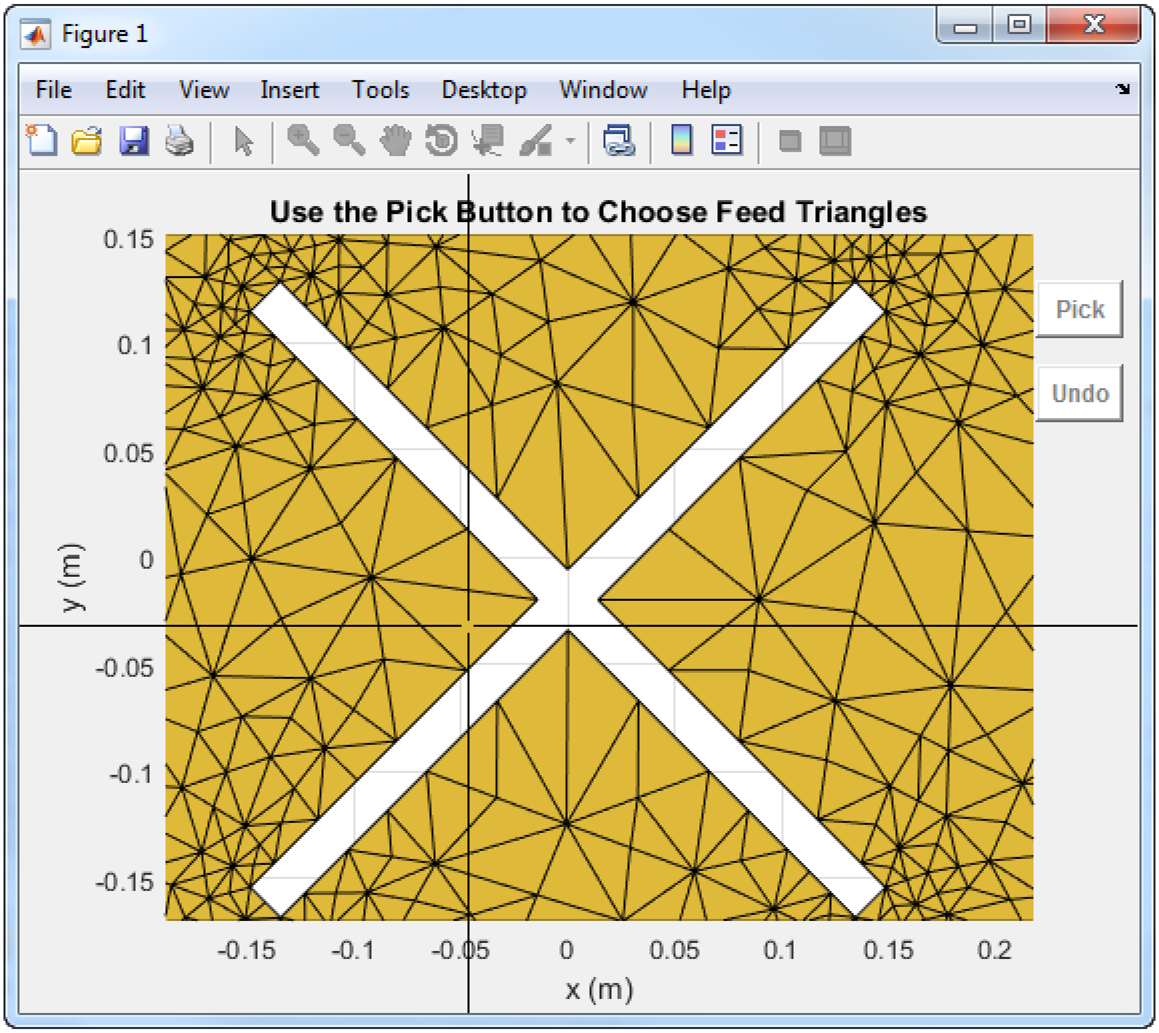
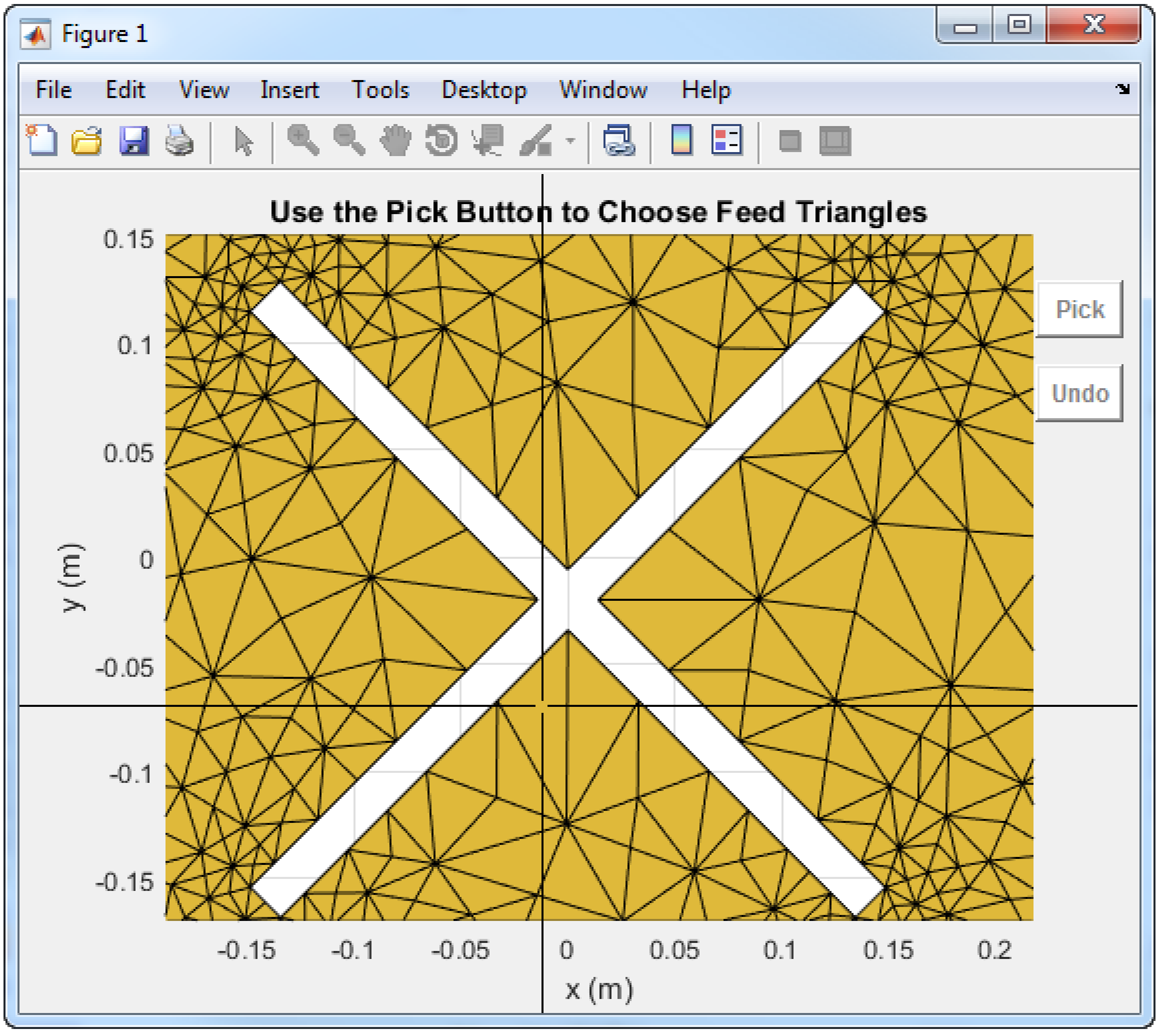


Figure 1
File Edit View Insert Tools Desktop Window Help
Use the Pick Button to Choose Feed Triangles
Pick
Undo
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
y (m)
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
x (m)

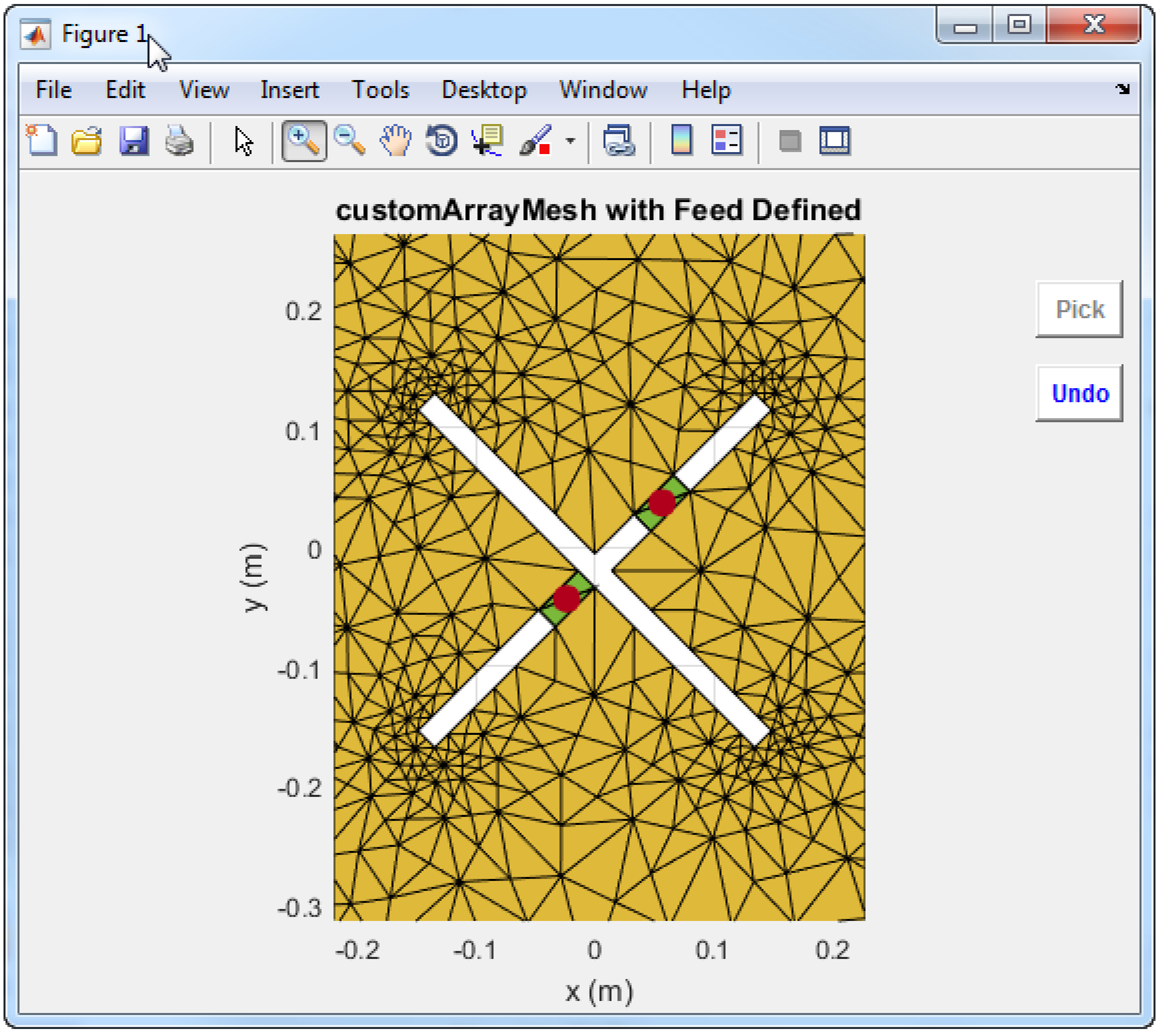
- Select first triangle for feedpoint 2.



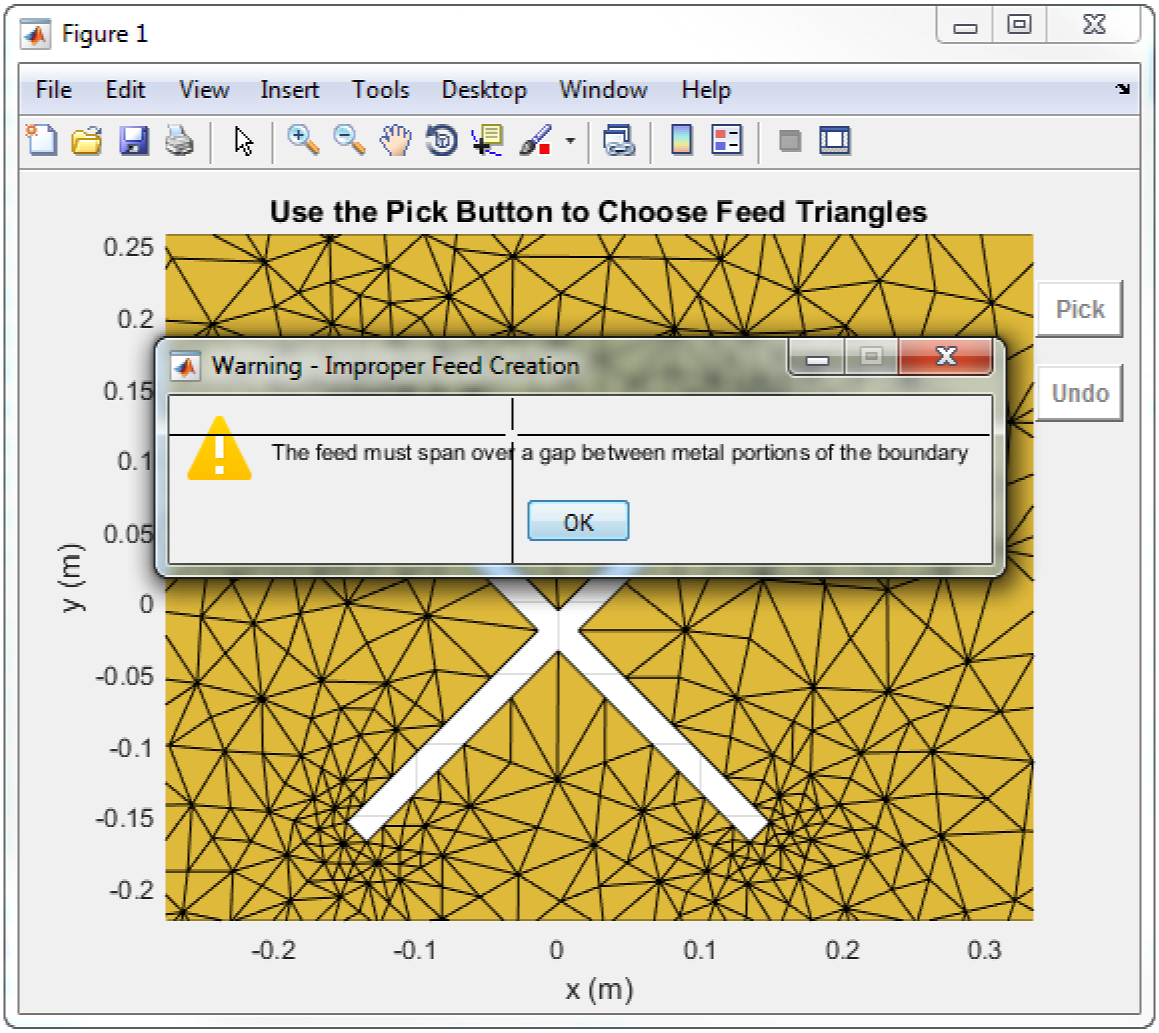
- Select the second triangle on the other side of the air gap for feedpoint 2.



Selecting the fourth triangle creates and displays the array feeds.



You must select the two triangles on either side of the air gap. Otherwise, the function displays an error message.

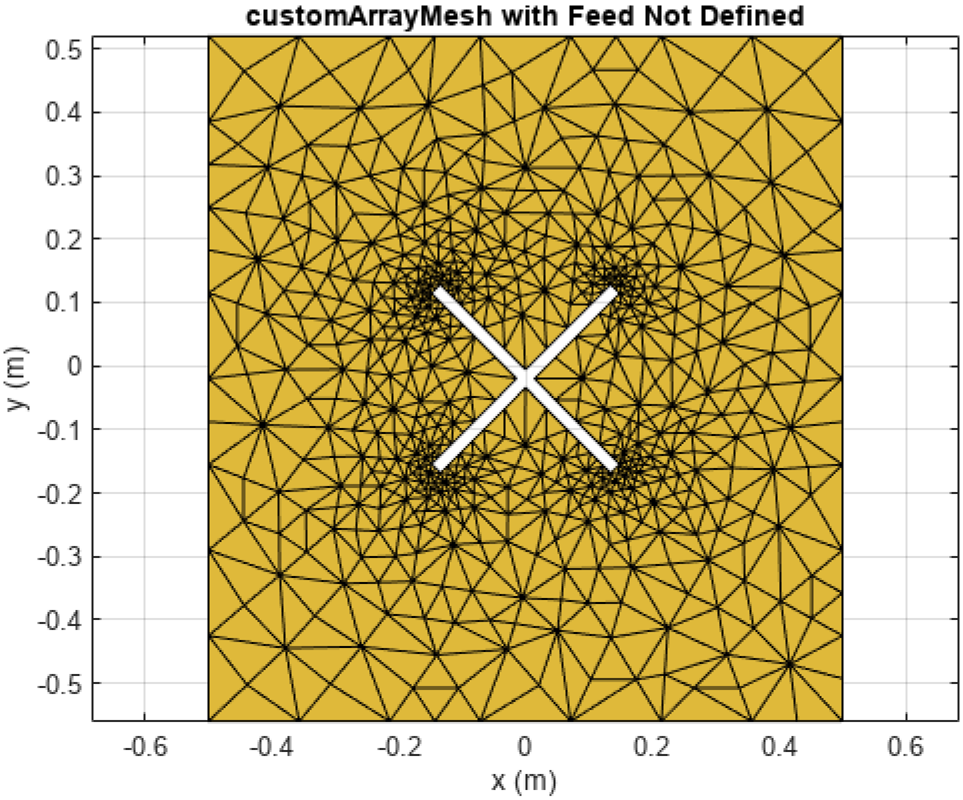


The feed must span over a gap between metal portions of the boundary

Create Feed for Custom Array Mesh

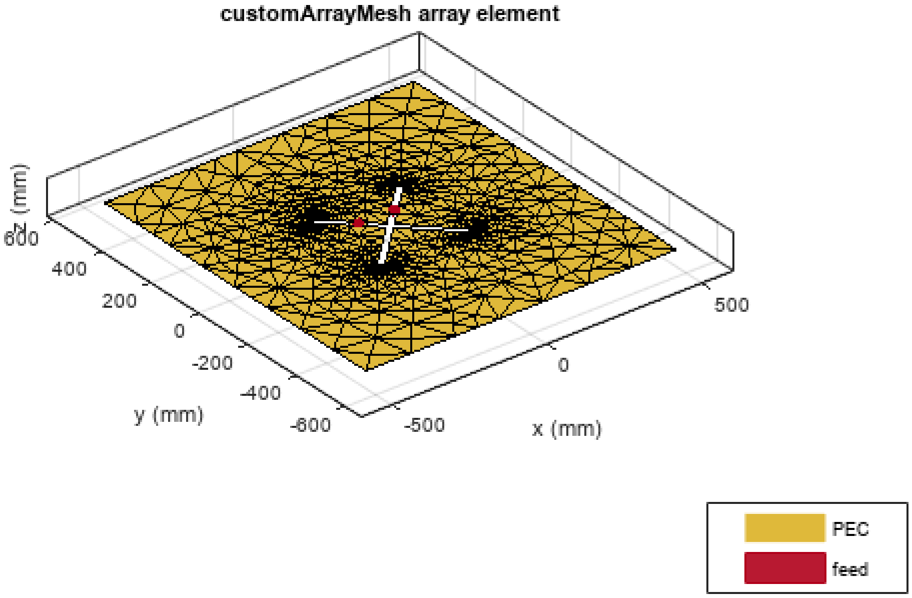
Load a custom mesh and create an array.

```
load planarmesh.mat;
c = customArrayMesh(p,t,2);
show(c)
```



Create feeds for the custom array mesh.

```
createFeed(c,[0.07,0.01],[0.05,0.05], [-0.07,0.01],[-0.05,0.05]);
show(c)
```



Version History

Introduced in R2016a

See Also

`returnLoss` | `sparameters`

impedance

Input impedance of antenna; scan impedance of array

Syntax

```
impedance(antenna,frequency)
z = impedance(antenna,frequency)
```

```
impedance(array,frequency,elementnumber)
z = impedance(array,frequency,elementnumber)
```

Description

`impedance(antenna,frequency)` calculates the input impedance of an antenna object and plots the resistance and reactance over a specified frequency.

`z = impedance(antenna,frequency)` returns the impedance of the antenna object, over a specified frequency.

`impedance(array,frequency,elementnumber)` calculates and plots the scan impedance of a specified antenna element in an array.

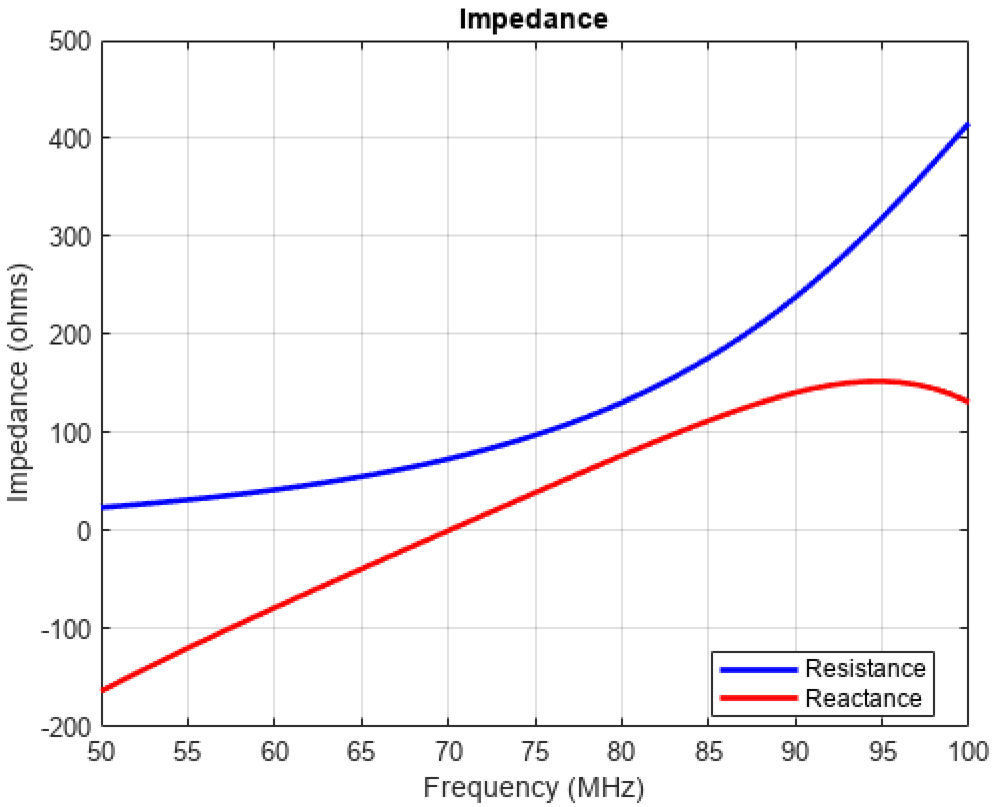
`z = impedance(array,frequency,elementnumber)` returns the scan impedance of a specified antenna element in an array.

Examples

Calculate and Plot Impedance of Antenna

Calculate and plot the impedance of a planar dipole antenna over a frequency range of 50MHz - 100MHz.

```
h = dipole;
impedance (h,50e6:1e6:100e6);
```



Calculate Scan Impedance of Array

Calculate scan impedance of default linear array over a frequency range of 50MHz to 100MHz.

```
h = linearArray;
z = impedance(h,50e6:1e6:100e6)
```

```
z = 51×2 complex
10² ×

   0.2976 - 1.7632i   0.2976 - 1.7632i
   0.3089 - 1.6801i   0.3089 - 1.6801i
   0.3204 - 1.5989i   0.3204 - 1.5989i
   0.3322 - 1.5193i   0.3322 - 1.5193i
   0.3441 - 1.4411i   0.3441 - 1.4411i
   0.3564 - 1.3643i   0.3564 - 1.3643i
   0.3689 - 1.2887i   0.3689 - 1.2887i
   0.3816 - 1.2142i   0.3816 - 1.2142i
   0.3947 - 1.1406i   0.3947 - 1.1406i
   0.4081 - 1.0678i   0.4081 - 1.0678i
       ⋮
```

Input Arguments

`antenna` — Antenna or array object

scalar

Antenna object, specified as a scalar.

`array` — Array object

scalar

Array object, specified as a scalar.

`frequency` — Frequency range used to calculate impedance

scalar | vector

Frequency range to calculate impedance, specified as a scalar in hertz or a vector with each element unit in Hz.

Example: 50e6:1e6:100e6

Data Types: `double`

`elementnumber` — Antenna element number in array

scalar

Antenna element number in array, specified as a scalar.

Example: 1

Data Types: `double`

Output Arguments

`z` — Input impedance of antenna or scan impedance of array

complex number in ohms

Input impedance of antenna or scan impedance of array, returned as a complex number in ohms. The real part of the complex number indicates the resistance. The imaginary part of the complex number indicates the reactance.

Note Antenna Toolbox caches the impedance values while running for the first time so that the subsequent runs are faster.

Version History

Introduced in R2015a

See Also

`returnLoss`

rfparam

Extract vector of network parameters

Syntax

```
n_ij = rfparam(hnet,i,j)
```

Description

`n_ij = rfparam(hnet,i,j)` extracts the network parameter vector (*i*,*j*) from the network parameter object, `hnet`.

Examples

Create Data Vector From S-Parameter Object

Read in the file default.s2p into an sparameters object and get the S21 value.

```
S = sparameters('default.s2p');
s21 = rfparam(S,2,1)
```

```
s21 = 191×1 complex

  -0.6857 + 1.7827i
  -0.6560 + 1.7980i
  -0.6262 + 1.8131i
  -0.5963 + 1.8278i
  -0.5664 + 1.8422i
  -0.5363 + 1.8563i
  -0.5062 + 1.8700i
  -0.4760 + 1.8835i
  -0.4457 + 1.8966i
  -0.4152 + 1.9094i
      ⋮
```

Input Arguments

hnet — Network parameters
network parameter object

Network parameters, specified as an RF Toolbox™ network parameter object.

i — Row index
positive integer

Row index of data to extract, specified as a positive integer.

j — Column index
positive integer

Column index of data to extract, specified as a positive integer.

Output Arguments

`n_ij` — Network parameters (*i, j*)
vector

Network parameters (*i, j*), returned as a vector. The `i` and `j` input arguments determine which parameters the function returns.

Example: `S_21 = rfparam(hs,2,1)`

Version History

Introduced before R2006a

See Also

`sparameters` | `rfinterp1` | `rfplot`

rfplot

Plot S-parameter data

Syntax

```
rfplot(s_obj)
rfplot(s_obj,i,j)
rfplot(s_obj,[i1:in],[j1:jn])
rfplot(s_obj,{[i1 j1];...;[in jn]})
rfplot( ___ ,LineSpec)
rfplot( ___ ,plotflag)
rfplot(s_obj,'diag')
rfplot(s_obj,part)
rfplot(s_obj,part,k)
rfplot(ax, ___ )
hline = rfplot( ___ )
[hline,haxes] = rfplot(filter,frequencies)
```

Description

`rfplot(s_obj)` plots the magnitude in decibels versus frequency of all S-parameters (S_{11}, S_{12} ... S_{NN}) on the current axes.

`rfplot(s_obj,i,j)` plots the magnitude of $S_{i\,j}$ in decibels on the current axis.

`rfplot(s_obj,[i1:in],[j1:jn])` plots the magnitude of multiple S-parameters in decibels on the current axis.

`rfplot(s_obj,{[i1 j1];...;[in jn]})` plots the magnitude of specific S-parameters in decibels on the current axis.

`rfplot( ___ ,LineSpec)` plots S-parameters using the line parameters specified in `LineSpec`.

`rfplot( ___ ,plotflag)`plots S-parameters according to the type specified in `plotflag`.

`rfplot(s_obj,'diag')` plots the magnitude of $S_{i\,i}$ reflection coefficients or the diagonal elements of the S-parameter matrix `'diag'` on the current axis.

`rfplot(s_obj,part)` plots the upper or lower triangular portion of the S-parameters matrix on the current axis.

`rfplot(s_obj,part,k)` plots the elements on, above, or below the `k`th diagonal of the S-parameters matrix. For more information, see the `tril` and `triu` functions.

`rfplot(ax, ___ )` plots the S-parameters on the axes specified in `ax` instead of the current axes. Specify `ax` as the first input argument followed by any of the input argument combinations in the previous syntaxes. Return the current axes using the `gca` function.

`hline = rfplot( ___ )` plots the S-parameters and returns a column vector of line handles in `hline`.

`[hline,haxes] = rfplot(filter,frequencies)` plots the magnitude response of the S-parameters of the RF filter.

Examples

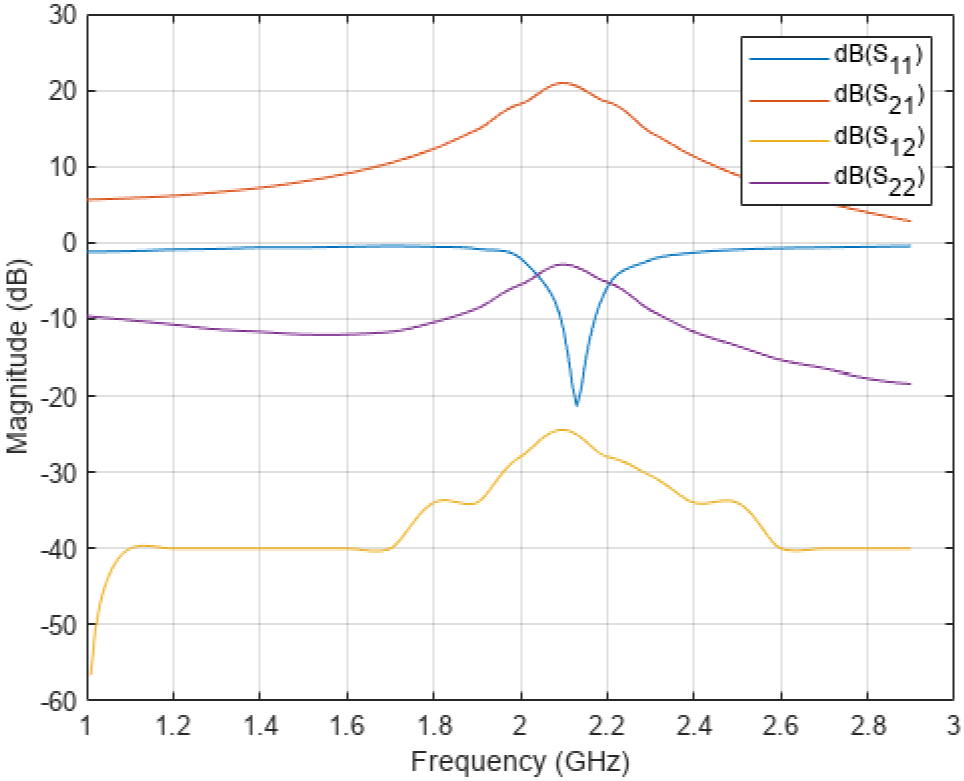
Plot S-Parameter Data

Use the `sparameters` function to create a set S-parameters.

```
hs = sparameters('default.s2p');
```

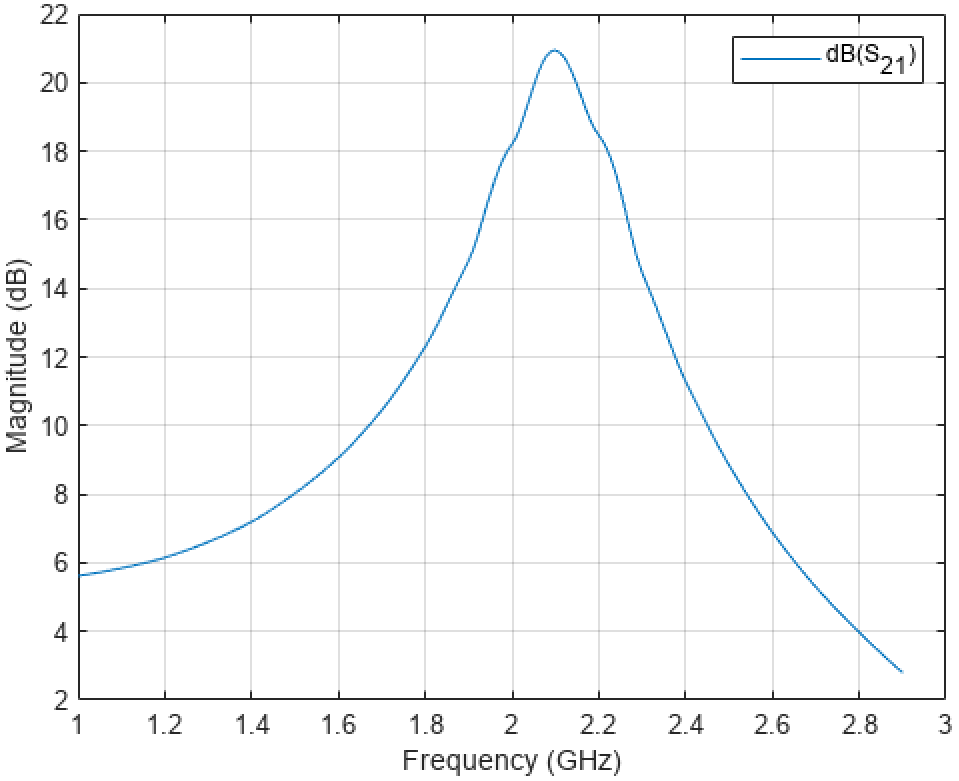
Plot all the S-parameters.

```
rfplot(hs)
```



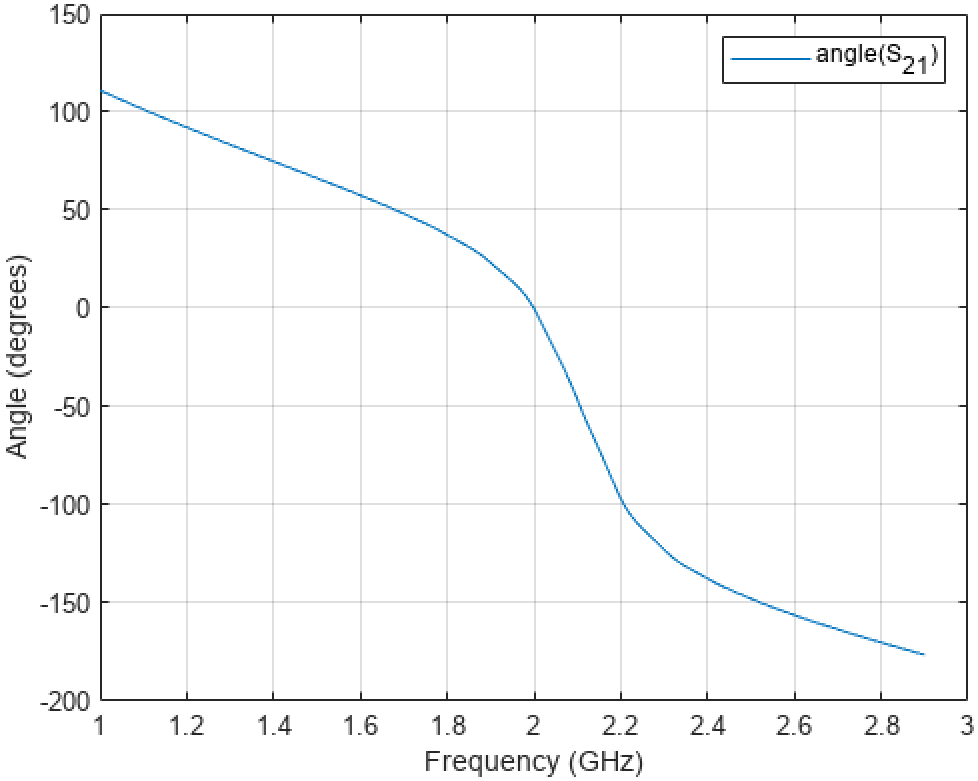
Plot S21.

```
rfplot(hs,2,1);
```



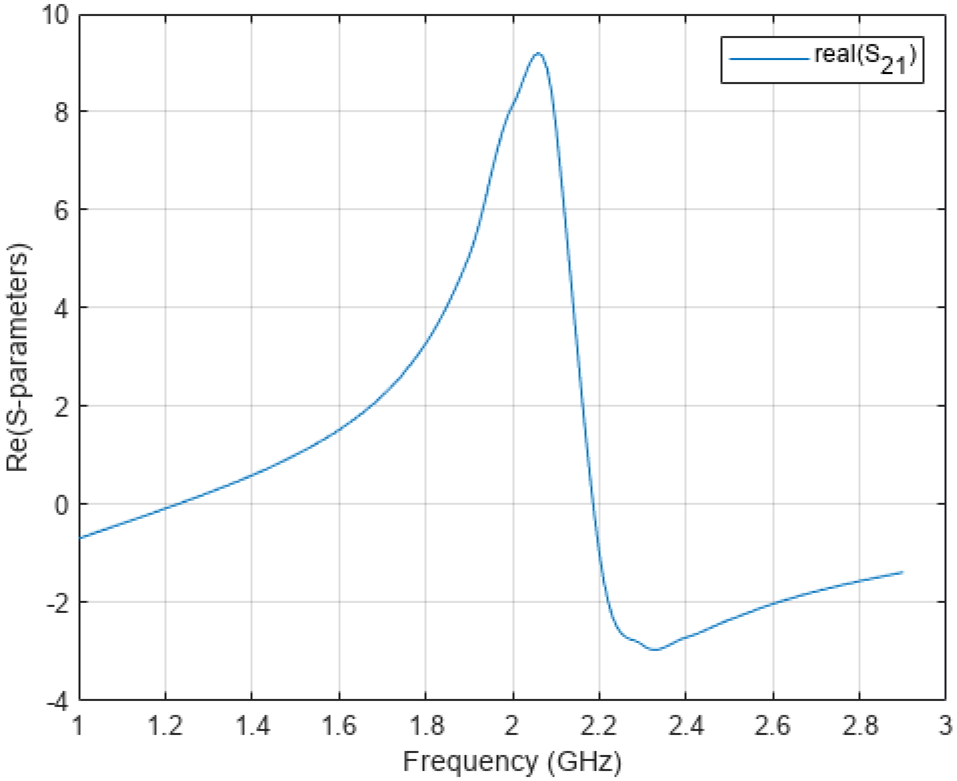
Plot the angle of S21 in degrees.

```
rfplot(hs,2,1,'angle')
```



Plot the real part of S21.

```
rfplot(hs,2,1,'real')
```



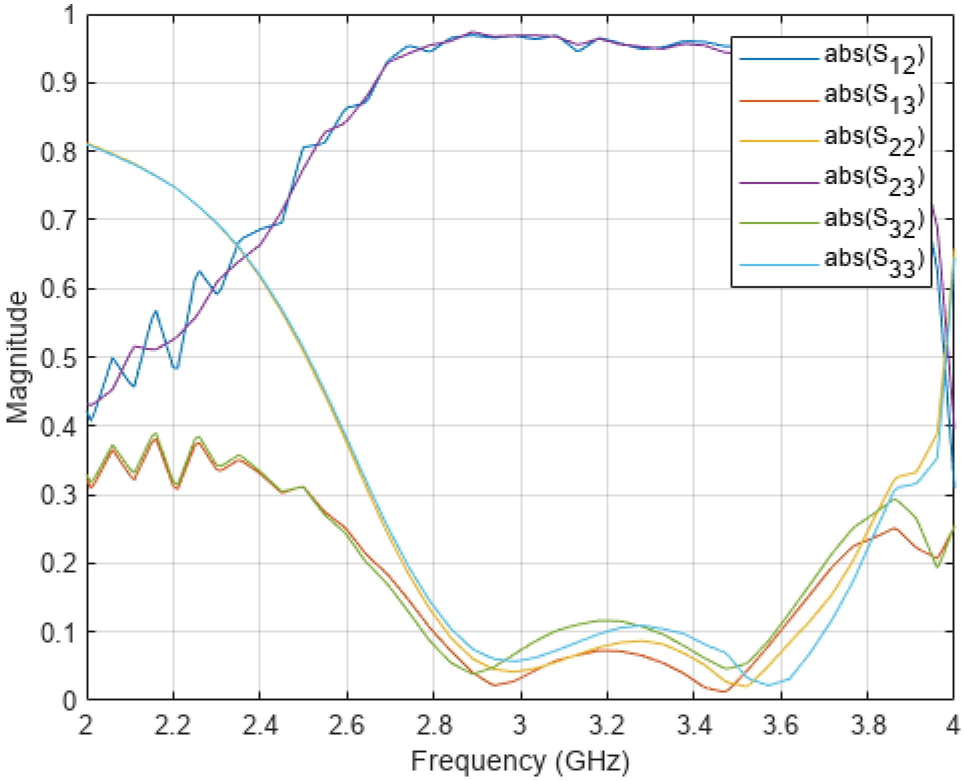
Plot S-Parameters in Specified Range

Create an S-parameter object from a three-port Touchstone file.

```
sobj = sparameters('default.s3p');
```

Plot S12, S13, S22, S23, S32, and S33.

```
rfplot(sobj,[1:3],[2:3],'abs')
```



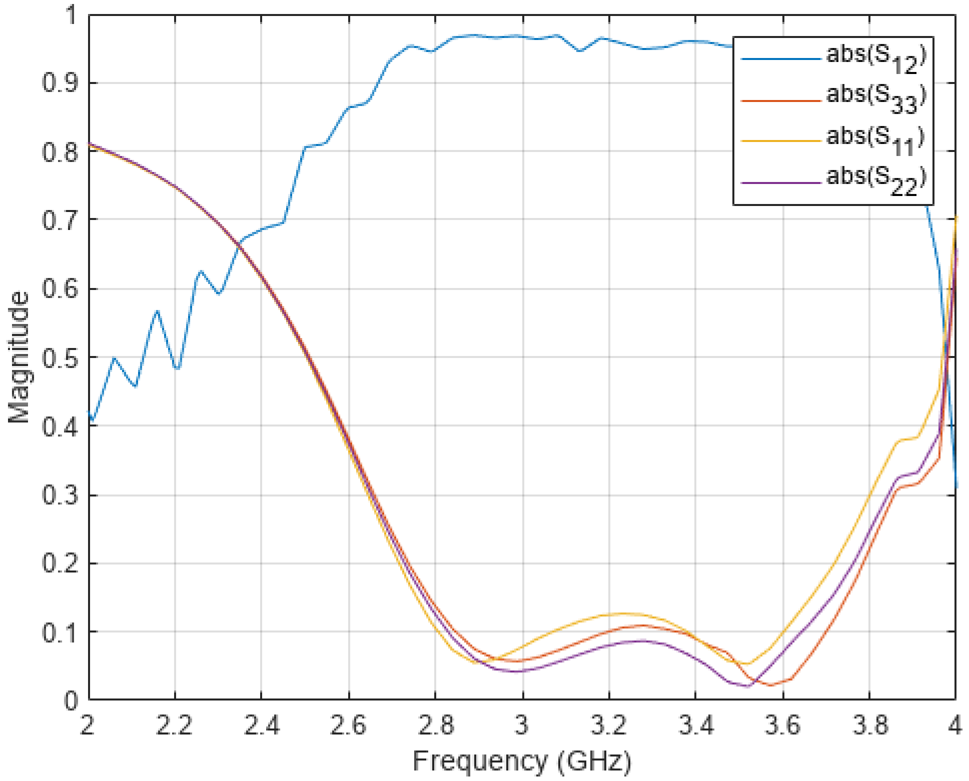
Plot Specific S-Parameters

Create an S-parameter object from a three-port Touchstone file.

```
sobj = sparameters('default.s3p');
```

Plot S12, S33, S11, and S22.

```
rfplot(sobj,{[1 2]; [3 3]; [1 1]; [2 2]},'abs')
```



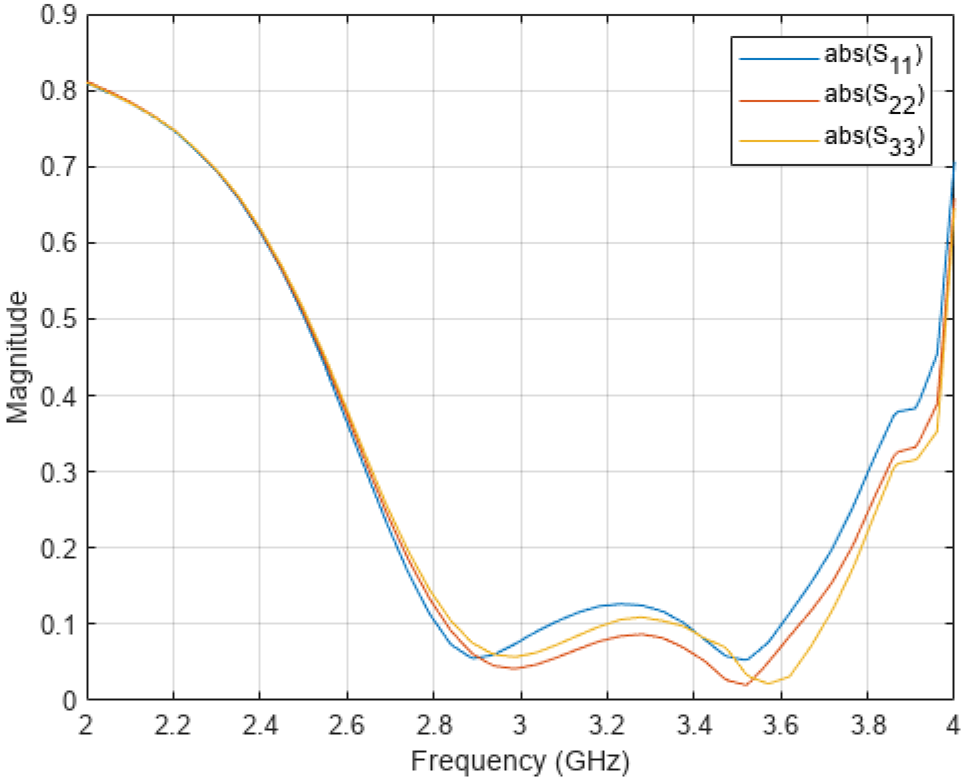
Plot Reflection and Transmission Coefficients

Create an S-parameter object from a three-port Touchstone file.

```
sobj = sparameters('default.s3p');
```

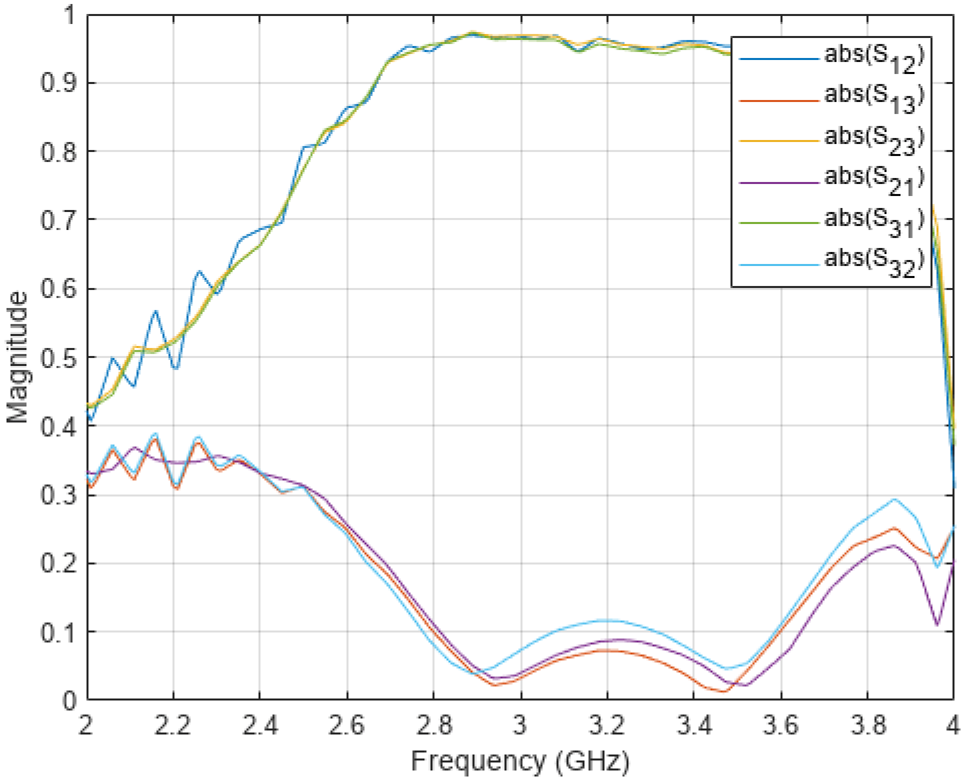
Plot the reflection coefficients of the S-parameters.

```
rfplot(sobj,'diag','abs')
```



Plot the transmission coefficients of the S-parameters.

```
rfplot(sobj,'triu',1,'abs')
hold on
rfplot(sobj,'tril',-1,'abs')
```



Input Arguments

`s_obj` — S-parameters

network parameter object

S-parameters, specified as RF Toolbox network parameter object. To create this type of object, use the `sparameters` function.

`i` — Row index

scalar | vector | cell array

Row index of the data to plot, specified as a scalar, vector, or cell array.

Type of Plot	How to Specify Indices
Single parameter	Specify `i` and `j` as scalars. `rfplot(s_obj,[1,2])`

Set of parameters	Specify `i` and `j` as vectors.
	`rfplot(s_obj,[1:3],[2:3])`
	`rfplot(s_obj,[1,2],[2,3])`

Type of Plot	How to Specify Indices
Specific parameters	Specify a cell array of `i` and `j` scalars. `rfplot(s_obj, {[1 2];[2 3]})`

`j` — Column index

scalar | vector | cell array

Column index of the data to plot, specified as a scalar, vector, or cell array.

Type of Plot	How to Specify Indices
Single parameter	Specify `i` and `j` as scalars. `rfplot(s_obj,[1,2])`

Set of parameters	Specify `i` and `j` as vectors.
	`rfplot(s_obj,[1:3],[2:3])`
	`rfplot(s_obj,[1,2],[2,3])`

Specific parameters	Specify a cell array of `i` and `j` scalars. `rfplot(s_obj, {[1 2];[2 3]})`

`LineSpec` — Line style, marker, and color

character vector | string

Line style, marker, and color, specified as a character vector or a string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, if you omit the line style and specify the marker, then the plot shows only the marker and no line.

Example: `'--or'`creates a dashed line in red with circular markers

Line Style	Description	Resulting Line
`'-'`	Solid line	————
`'--'`	Dashed line	- — — — —
`':'`	Dotted line	………………
`'-.'`	Dash-dotted line	—·—·—·—·-

Marker	Description	Resulting Marker
`'o'`	Circle	○
`'+'`	Plus sign	+
`'*'`	Asterisk	✳

Marker	Description	Resulting Marker
'.'	Point	•
'x'	Cross	×
'_'	Horizontal line	—
'|'	Vertical line	|
's'	Square	□
'd'	Diamond	◇
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	🟥
'green'	'g'	[0 1 0]	🟩
'blue'	'b'	[0 0 1]	🟦
'cyan'	'c'	[0 1 1]	🟦
'magenta'	'm'	[1 0 1]	🟪
'yellow'	'y'	[1 1 0]	🟨
'black'	'k'	[0 0 0]	⬛
'white'	'w'	[1 1 1]	⬜

plotflag — Plot types

'db' (default) | 'real' | 'imag' | 'abs' | 'angle'

Plot types, specified as either 'db', 'real', 'imag', 'abs', or 'angle'.

Example: 'angle'

filter — RF filter

rffilter object | lcladder object

RF filter, specified as an rffilter object or an lcladder object.

frequencies — Frequencies to plot magnitude response

vector

Frequencies to plot magnitude response, specified as a vector.

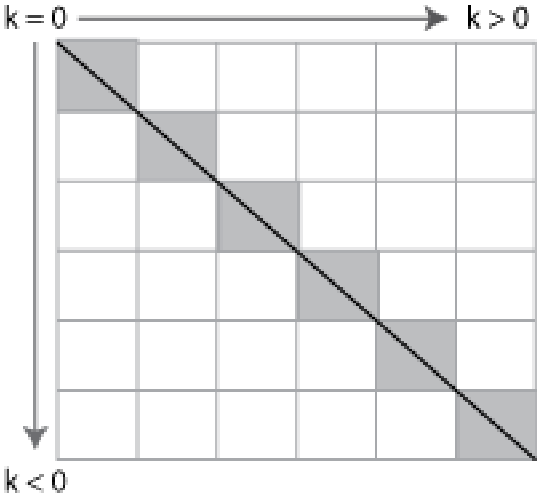
`part` — Portion of S-parameters matrix
`'triu'` | `'tril'`

Portion of the S-parameters matrix, specified as `'triu'` or `'tril'`. Specify `triu` to plot the “Upper Triangular” (RF Toolbox) portion of the matrix and `tril` to plot the “Lower Triangular” (RF Toolbox) portion.

k — Diagonals to include
`0` (default) | scalar

Diagonals to include, specified as a scalar.

- `k = 0` specifies the main diagonal.
- `k > 0` specifies a diagonal above the main diagonal.
- `k < 0` specifies a diagonal below the main diagonal.



ax — Axes object
`axes` object | `uiaxes` object

Axes object, specified as an `axes` or a `uiaxes` object.

Output Arguments

`hline` — Line
line handle

Line containing the S-parameter plot, returned as a line handle.

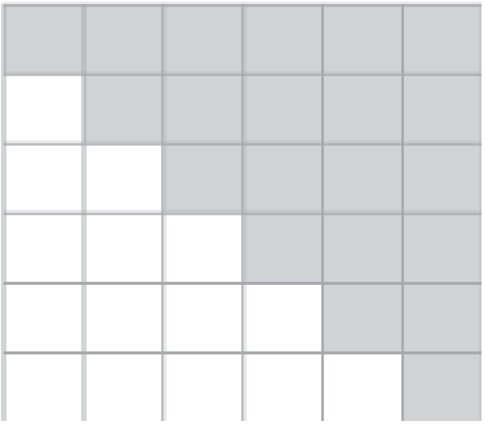
haxes — Axes
axes handle

Axes of the `rfplot`, returned as an axes handle.

More About

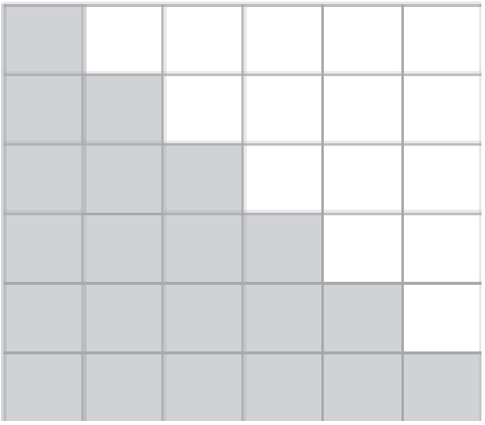
Upper Triangular

The upper triangular portion of a matrix includes the main diagonal and all elements above it. The shaded elements in this graphic depict the upper triangular portion of a 6-by-6 matrix.



Lower Triangular

The lower triangular portion of a matrix includes the main diagonal and all elements below it. The shaded elements in this graphic depict the lower triangular portion of a 6-by-6 matrix.



Version History

Introduced before R2006a

Plot S-parameter data with multiple indices

Use the `'diag'`, `part`, and `k` input arguments in the `rfplot` function to plot S-parameter data with multiple indices.

See Also

`sparameters` | `sparameters` | `setrfplot`

show

Display antenna, array structures or shapes

Syntax

`show(object)`

`show(shape)`

Description

`show(object)` displays the structure of an antenna or array object.

`show(shape)` displays shape as filled region using patches.

Examples

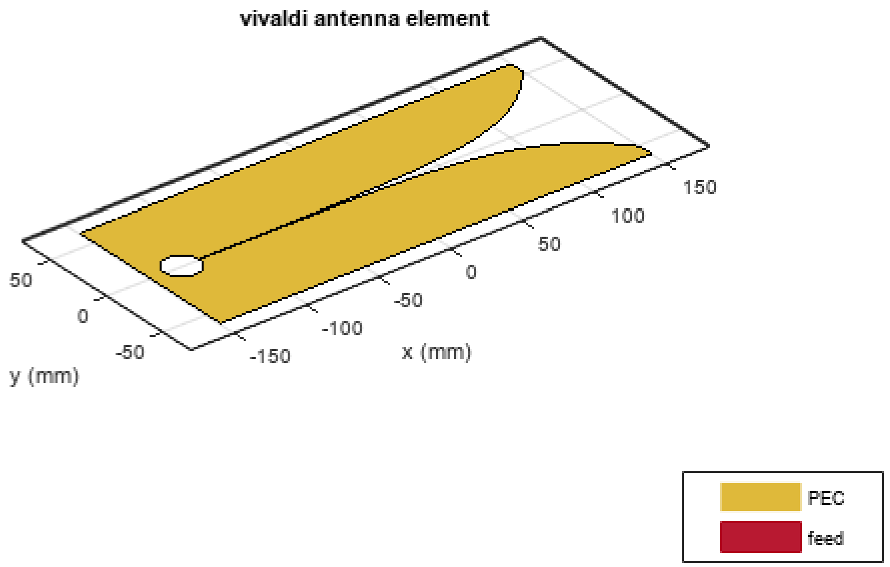
Display Antenna Structure

This example shows how to create a vivaldi antenna and display the antenna structure.

```
h = vivaldi
```

```
h = 
  vivaldi with properties:

             TaperLength: 0.2430
           ApertureWidth: 0.1050
             OpeningRate: 25
           SlotLineWidth: 5.0000e-04
          CavityDiameter: 0.0240
    CavityToTaperSpacing: 0.0230
       GroundPlaneLength: 0.3000
        GroundPlaneWidth: 0.1250
              FeedOffset: -0.1045
               Conductor: [1x1 metal]
                    Tilt: 0
                TiltAxis: [1 0 0]
                    Load: [1x1 lumpedElement]
```

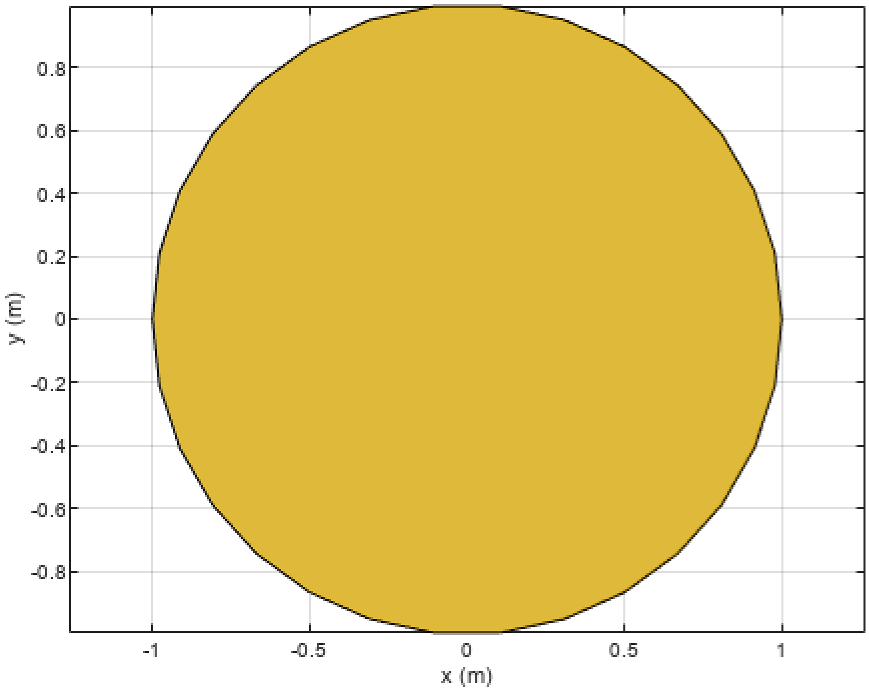
```
show(h)
```



Show Circle Shape

Create a circular shape and visualize the filled regions.

```
c  = antenna.Circle;
show(c);
```



Input Arguments

`object` — Antenna or array object
scalar

Antenna or array object, specified as a scalar.

`shape` — Shape created using custom elements and shape objects
object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object. You can create the shapes using `antenna.Circle`, `antenna.Polygon`, or `antenna.Rectangle`.

Example: `c = antenna.Rectangle; show(c)`

Version History

Introduced in R2015a

See Also

`layout` | `mesh` | `plot`

returnLoss

Return loss of antenna; scan return loss of array

Syntax

```
returnLoss(antenna,frequency)
returnLoss(antenna,frequency,z0)
rl = returnLoss(antenna,frequency,z0)

returnLoss(array,frequency,z0,elementnumber)
rl = returnLoss(array,frequency,z0,elementnumber)
```

Description

`returnLoss(antenna,frequency)` calculates and plots the return loss of an antenna, over a specified frequency and at a reference impedance of 50 ohm.

`returnLoss(antenna,frequency,z0)` calculates and plots the return loss of an antenna, over a specified frequency and a given reference impedance, `z0`.

`rl = returnLoss(antenna,frequency,z0)` returns the return loss of an antenna.

`returnLoss(array,frequency,z0,elementnumber)` calculates and plots the scan return loss of a specified antenna element in an array.

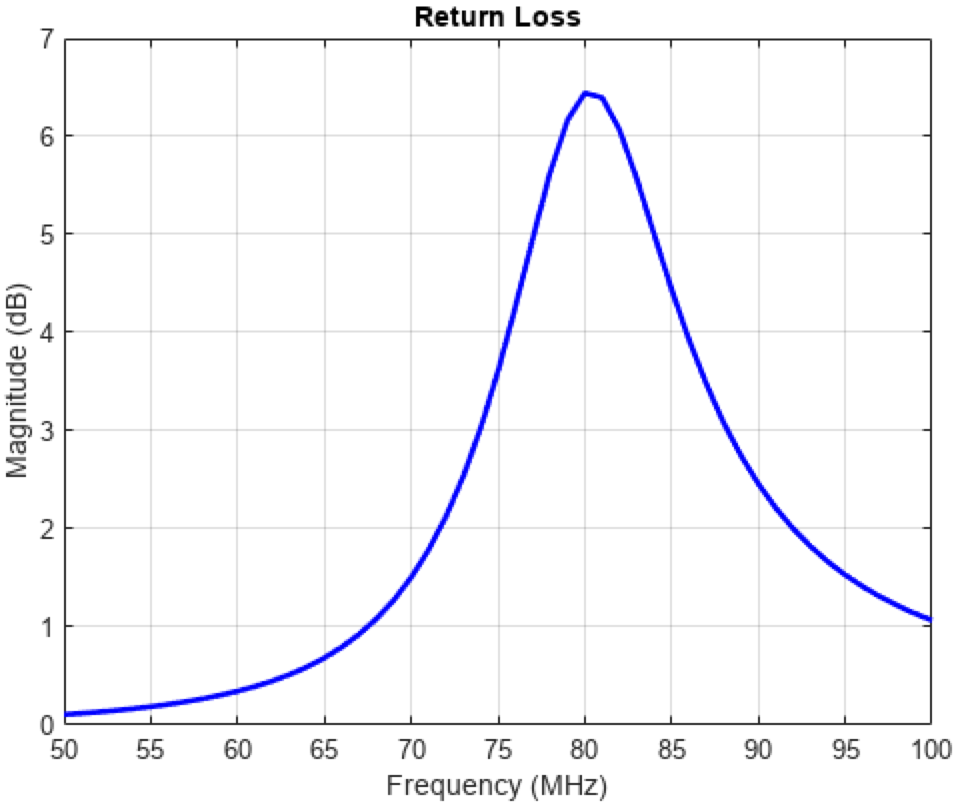
`rl = returnLoss(array,frequency,z0,elementnumber)` returns the scan return loss of a specified antenna element in an array.

Examples

Calculate and Plot Return Loss of Antenna

This example shows how to calculate and plot the return loss of a circular loop antenna over a frequency range of 50MHz-100MHz.

```
h = loopCircular;
returnLoss (h, 50e6:1e6:100e6);
```



Input Arguments

`antenna` — Antenna object
scalar

Antenna object, specified as a scalar.

`array` — array object
scalar

Array object, specified as a scalar.

`frequency` — Frequency range used to calculate return loss
vector in Hz

Frequency range used to calculate return loss, specified as a vector in Hz.

Example: 50e6:1e6:100e6

Data Types: `double`

`z0` — Reference impedance
50 (default) | scalar in ohms

Reference impedance, specified as a scalar in ohms.

Example: 40

Data Types: `double`

`elementnumber` — Antenna element number in array

scalar

Antenna element number in array, specified as a scalar.

Example: 1

Data Types: `double`

Output Arguments

`rl` — Return loss of antenna object or scan return loss of array object

vector in dB

Return loss of antenna object or scan return loss of array object, returned as a vector in dB. The return loss is calculated using the formula

$$RL = -20\log_{10}\left|\frac{(Z - Z_0)}{(Z + Z_0)}\right|$$

where,

- Z = input impedance of antenna or scan impedance of array
- Z_0 = reference impedance

Version History

Introduced in R2015a

See Also

`impedance` | `EHfields` | `sparameters`

pattern

Radiation pattern and phase of antenna or array; Embedded pattern of antenna element in array

Syntax

```
pattern(object,frequency)
pattern(object,frequency,azimuth,elevation)
pattern( ___ ,Name,Value)

[pat,azimuth,elevation] = pattern(object,frequency,azimuth,elevation)
[pat,azimuth,elevation] = pattern( ___ ,Name,Value)
```

Description

`pattern(object,frequency)` plots the 3-D radiation pattern of the antenna or array object over a specified frequency. By default, in Antenna Toolbox, the far-field radius is set to `100λ`. For a detailed explanation of field calculation of antennas, see "Field Calculation in Antennas".

`pattern(object,frequency,azimuth,elevation)` plots the radiation pattern of the antenna or array object using the specified `azimuth` and `elevation` angles.

`pattern( ___ ,Name,Value)` uses additional options specified by one or more Name, Value pair arguments. You can use any of the input arguments from previous syntaxes.

Use the `'ElementNumber'` and `'Termination'` property to calculate the embedded pattern of the antenna element in an array connected to a voltage source. The voltage source model consists of an ideal voltage source of 1 volt in series with a source impedance. The embedded pattern includes the effect of mutual coupling due to the other antenna elements in the array.

`[pat,azimuth,elevation] = pattern(object,frequency,azimuth,elevation)` returns the pattern value, `pat`, value of an antenna or array object at specified frequency. `azimuth` and `elevation` are the angles at which the pattern function calculates the directivity.

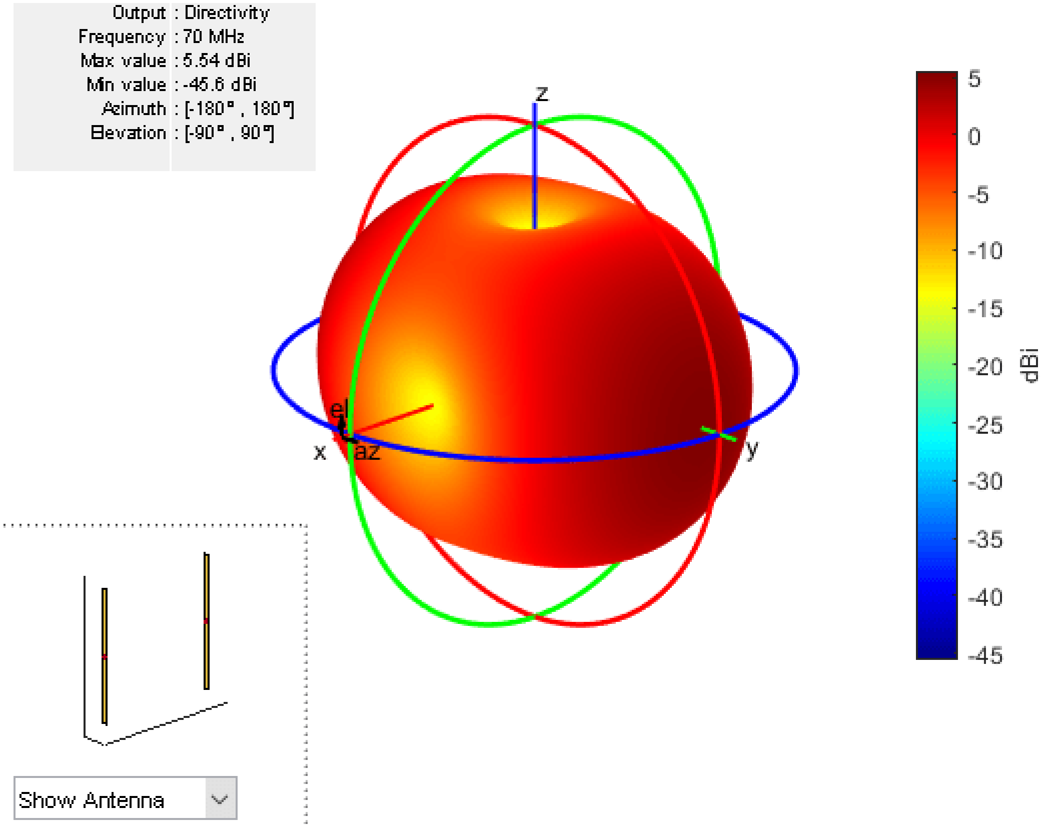
`[pat,azimuth,elevation] = pattern( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Calculate Radiation Pattern of Array

Calculate radiation pattern of default linear array for a frequency of 70 MHZ.

```
l = linearArray;
pattern(l,70e6)
```

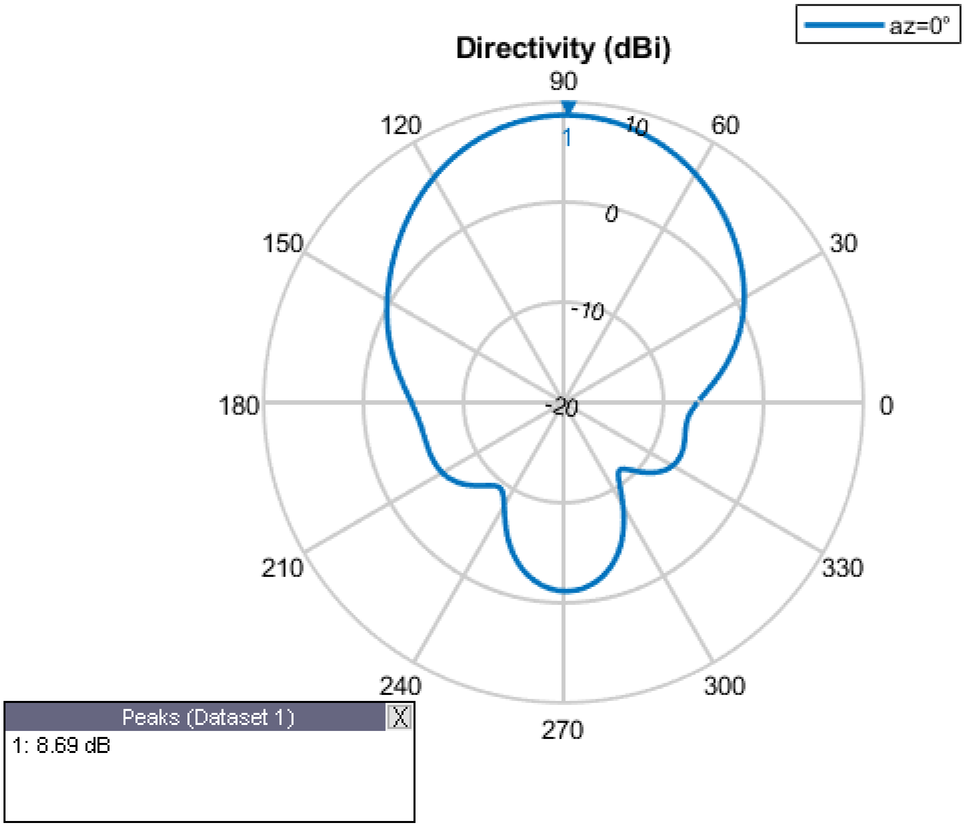


Output : Directivity
Frequency : 70 MHz
Max value : 5.54 dBi
Min value : -45.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Radiation Pattern of Helix in X-Z Plane

Plot the radiation pattern of a helix antenna in xz-plane.

```
h = helix;
pattern (h, 2e9, 0, 1:1:360);
```



```
[pat,azimuth,elevation] = pattern (h, 2e9, 0, 1:1:360);
```

Compute the maximum and the minimum value of the radiation pattern and the elevation angle.

```
pattern_max = max(max(pat))
```

```
pattern_max = 8.6905
```

```
pattern_min = min(min(pat))
```

```
pattern_min = -11.2357
```

```
elevation_max = max(elevation)
```

```
elevation_max = 360
```

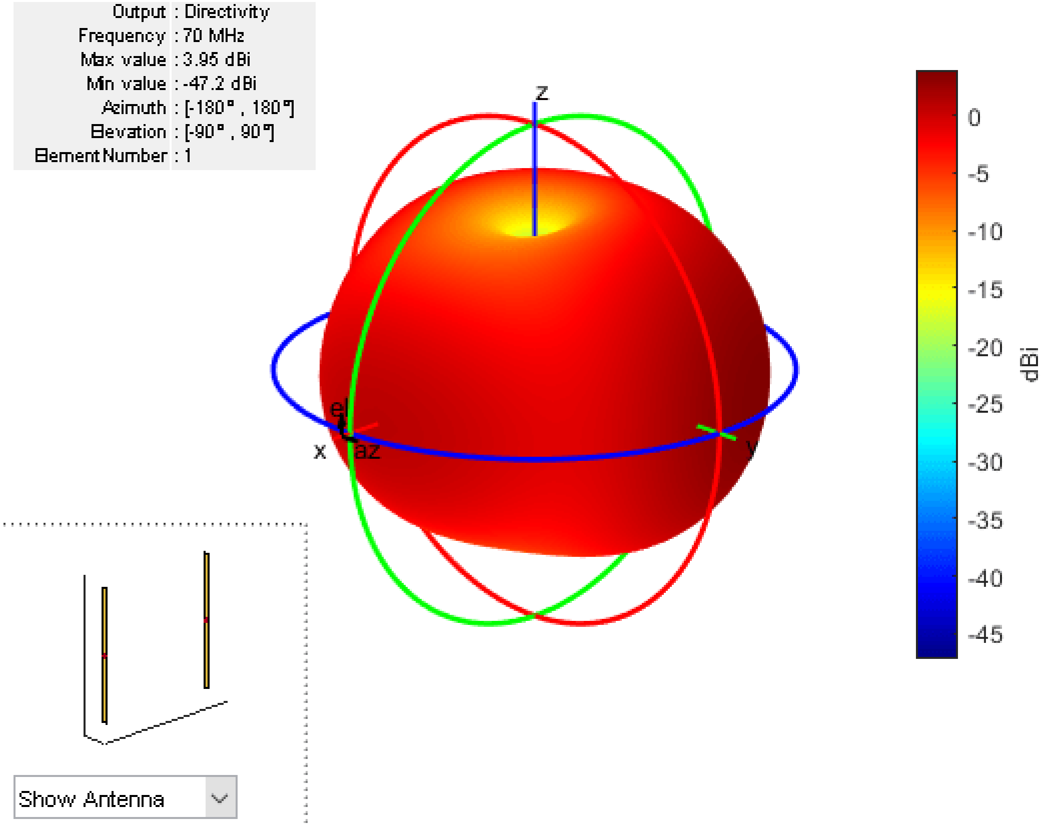
```
elevation_min = min(elevation)
```

```
elevation_min = 1
```

Embedded Element Pattern of Linear Array

Calculate the embedded element pattern of a linear array. Excite the first antenna element in the array. Terminate all the other antenna elements using a 50-ohm resistance.

```
l = linearArray;
pattern(l, 70e6,'ElementNumber', 1,'Termination', 50);
```



Output : Directivity
Frequency : 70 MHz
Max value : 3.95 dBi
Min value : -47.2 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
Element Number : 1

Directivity Value of Helix Antenna.

Calculate the directivity of a helix antenna.

```
h = helix;
D = pattern(h, 2e9, 0, 1:1:360);
```

Showing the first five directivity values.

```
Dnew = D(1:5)
```

```
Dnew = 5×1

   -6.4066
   -6.1929
   -5.9655
   -5.7262
   -5.4770
```

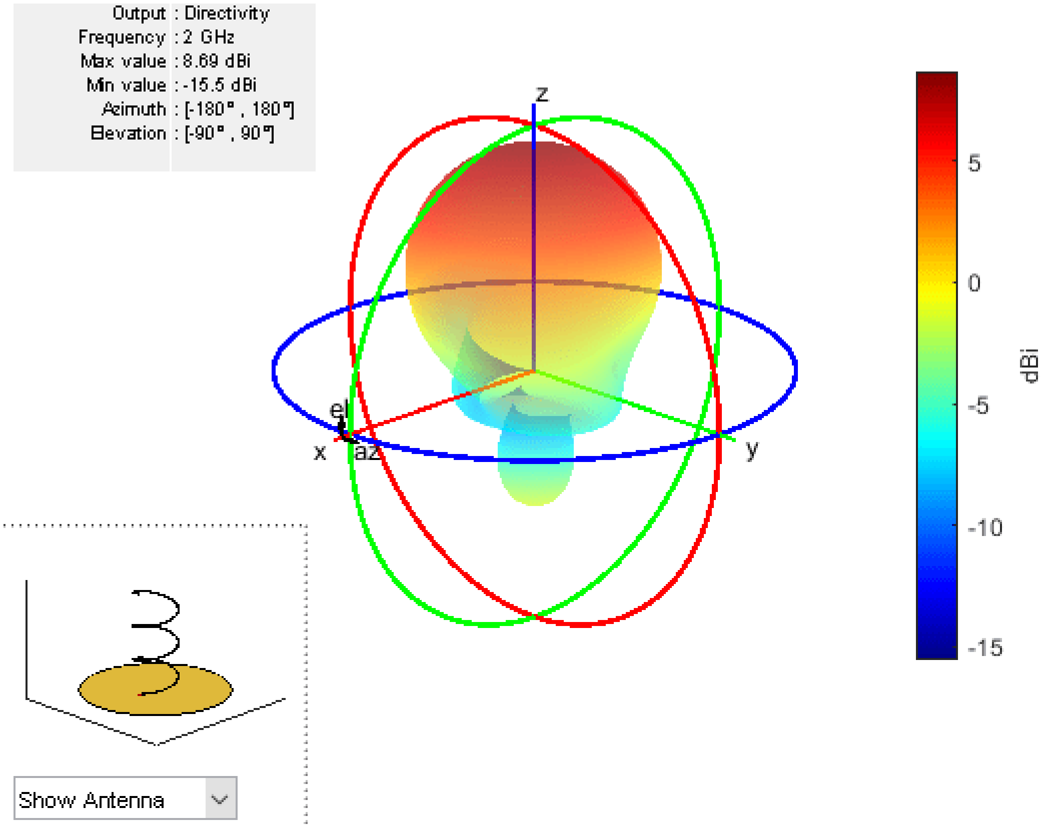
Radiation Pattern of Helix Antenna

Plot the radiation pattern of a helix antenna with transparency specified as 0.5.

```
p = PatternPlotOptions
```

```
p = 
  PatternPlotOptions with properties:

      Transparency: 1
         SizeRatio: 0.9000
    MagnitudeScale: []
     AntennaOffset: [0 0 0]
```

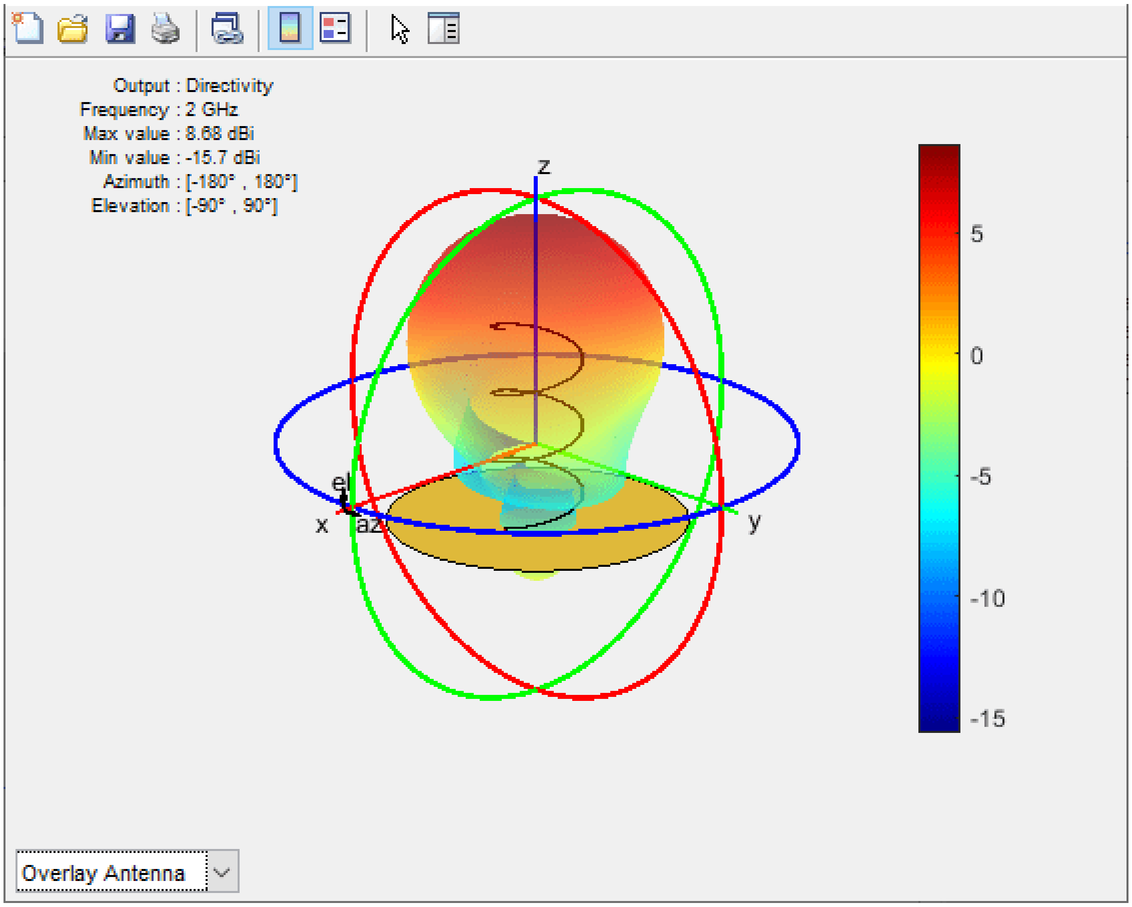
```
p.Transparency = 0.5;
ant = helix;
pattern(ant,2e9,'patternOptions',p)
```



```
   Output : Directivity
Frequency : 2 GHz
Max value : 8.69 dBi
Min value : -15.5 dBi
  Azimuth : [-180°, 180°]
Elevation : [-90°, 90°]
```

To understand the effect of Transparency, chose `Overlay Antenna` in the radiation pattern plot.

This option overlays the helix antenna on the radiation pattern.

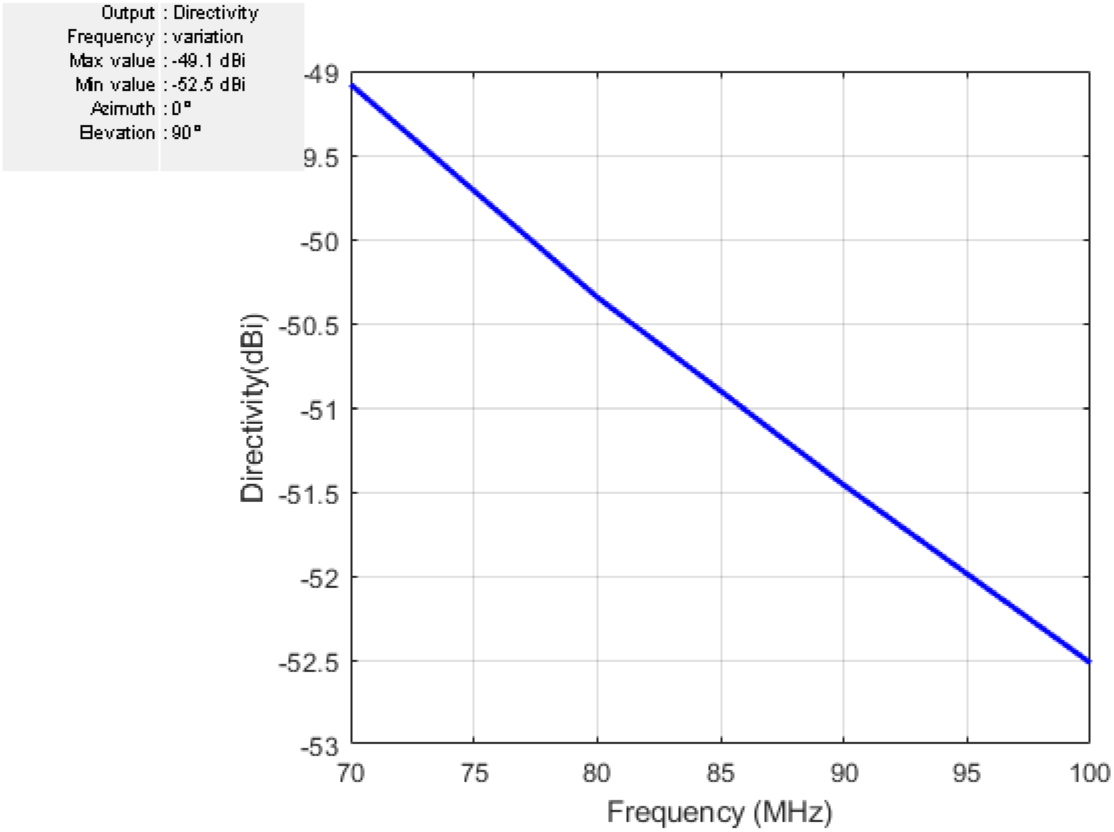


Output : Directivity
Frequency : 2 GHz
Max value : 8.68 dBi
Min value : -15.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Radiation Pattern of Dipole Antenna

Plot radiation pattern of dipole antenna in rectangular cartesian co-ordinate system.

```
pattern(dipole, 70e6:10e6:100e6, 0, 90, 'CoordinateSystem', 'rectangular')
```



Directivity values of dipole antenna

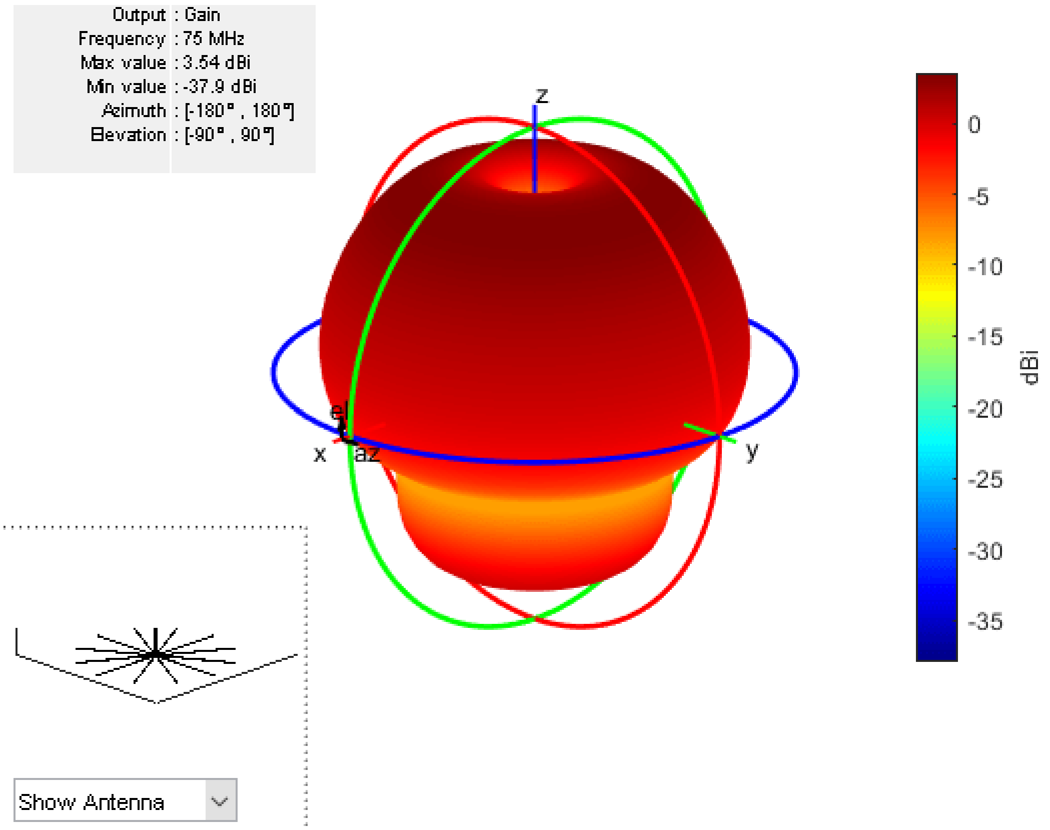
```
D = pattern(dipole, 70e6:10e6:100e6, 0, 90, 'CoordinateSystem', 'rectangular')
```

```
D = 4×1

  -49.0784
  -50.3449
  -51.4599
  -52.5182
```

Compare Gain and Realized Gain of Radial Monopole Antenna

Visualize gain plot of radial monopole antenna.

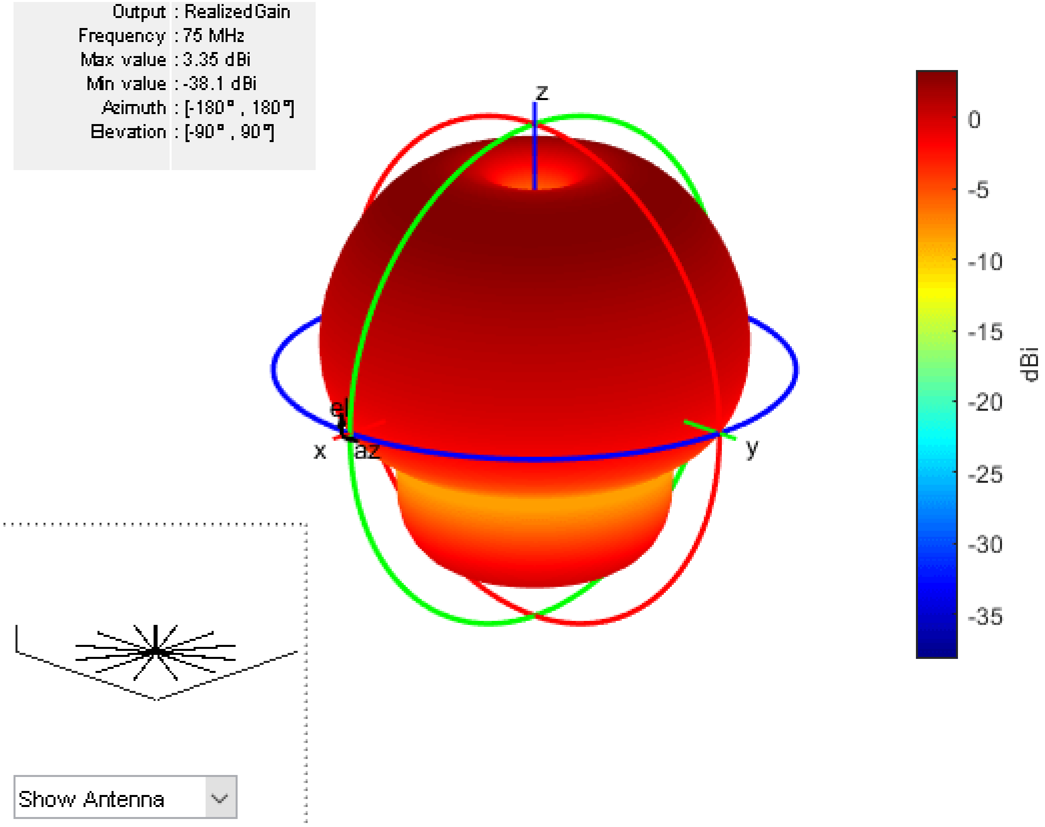
```
pattern(monopoleRadial,75e6,'Type','gain')
```



Output : Gain
Frequency : 75 MHz
Max value : 3.54 dBi
Min value : -37.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Visualize gain plot of radial monopole antenna.

```
pattern(monopoleRadial,75e6,'Type','realizedgain')
```



Output : RealizedGain
Frequency : 75 MHz
Max value : 3.35 dBi
Min value : -38.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Input Arguments

`object` — Antenna or array element

object

Antenna or array element, specified as an object.

`frequency` — Frequency to calculate or plot antenna or array radiation pattern

scalar | vector

Frequency to calculate or plot the antenna or array radiation pattern, specified as a scalar or a vector with each element in Hz. The vector frequencies support rectangular coordinate system.

Example: 70e6

Data Types: `double`

`azimuth` — Azimuth angles and spacing between angles

`–180:5:180` (default) | vector

Azimuth angles and spacing between the angles to visualize the radiation pattern, specified as a vector in degrees. If the coordinate system is set to uv, then the U values are specified in this parameter. The values of U are between `-1` to `1`.

Example: 90

Data Types: double

elevation — Elevation angles and spacing between angles

–90:5:90 (default) | vector

Elevation angles and spacing between the angles to visualize the radiation pattern, specified as a vector in degrees. If the coordinate system is set to uv, then the V values are specified in this parameter. The values of V are between -1 to 1.

Example: 0:1:360

Data Types: double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value pair 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: 'CoordinateSystem', 'uv'

CoordinateSystem — Coordinate system to visualize radiation pattern

'polar' (default) | 'rectangular' | 'uv'

Coordinate system to visualize the radiation pattern, specified as the comma-separated pair consisting of 'CoordinateSystem' and one of these values: 'polar', 'rectangular', 'uv'.

Example: 'CoordinateSystem', 'polar'

Data Types: char

Type — Quantity to plot

'directivity' | 'gain' | 'realizedgain' | 'efield' | 'power' | 'powerdb' | 'phase'

Quantity to plot, specified as a comma-separated pair consisting of 'Type' and one of these values:

- directivity – Directivity in dBi
- gain – Gain in dBi

Note The antenna Gain and Directivity are measured at a distance of 100*lambda.

- realizedgain – Realized gain in dBi
- efield – Electric field in volt/meter
- power – Power in watts
- powerdb – Power in dB
- phase – Phase in degrees.

Note Type can only be set to phase when Polarization is provided.

The default value is 'directivity' for a lossless antenna and 'gain' for a lossy antenna. You cannot plot the 'directivity' of a lossy antenna. For a detailed explanation on types and polarization, see “Field Calculation in Antennas”

Example: 'Type', 'efield'

Data Types: `char`

`Normalize` — Normalize field pattern

`false` (default) | `true` | boolean

Normalize field pattern, specified as the comma-separated pair consisting of `'Normalize'` and either `true` or `false`.

Example: `'Normalize'`, `false`

Data Types: `logical`

`PlotStyle` — 2-D pattern display style when frequency is vector

`'overlay'` (default) | `'waterfall'`

2-D pattern display style when frequency is a vector, specified as the comma-separated pair consisting of `'PlotStyle'` and one of these values:

- `'overlay'` – Overlay frequency data in a 2-D line plot
- `'waterfall'` – Plot frequency data in a waterfall plot

You can use this property when using `pattern` function with no output arguments.

Example: `'PlotStyle'`, `'waterfall'`

Data Types: `char`

`Polarization` — Field polarization

`'H'` | `'V'` | `'RHCP'` | `'LHCP'`

Field polarization, specified as the comma-separated pair consisting of `'Polarization'` and one of these values:

- `'H'` – Horizontal polarization
- `'V'` – Vertical polarization
- `'RHCP'` – Right-hand circular polarization
- `'LHCP'` – Left-hand circular polarization

By default, you can visualize a combined polarization.

Example: `'Polarization'`, `'RHCP'`

Data Types: `char`

`ElementNumber` — Antenna element in array

scalar

Antenna element in array, specified as the comma-separated pair consisting of `'ElementNumber'` and scalar. This antenna element is connected to the voltage source.

Note Use this property to calculate the embedded pattern of an array.

Example: `'ElementNumber'`,1

Data Types: `double`

`Termination` — Impedance value for array element termination
50 (default) | scalar

Impedance value for array element termination, specified as the comma-separated pair consisting of `'Termination'` and scalar. The impedance value terminates other antenna elements of an array while calculating the embedded pattern of the antenna connected to the voltage source.

Note Use this property to calculate the embedded pattern of an array.

Example: `'Termination'`,40

Data Types: `double`

`PatternOptions` — Parameter to change pattern plot properties
`PatternPlotOptions` object (default) | scalar

Parameter to change pattern plot properties, specified as the comma-separated pair consisting of `'PatternOptions'` and a `PatternPlotOptions` output. The properties that you can vary are:

- `Transparency`
- `SizeRatio`
- `AntennaOffset`
- `MagnitudeScale`

Example: `p = PatternPlotOptions('Transparency',0.1);` Create a pattern plot option with a transparency of 0.1. `ant = helix;pattern(ant,2e9,'PatternOptions',p)`; Use this pattern plot option to visualize the pattern of a helix antenna.

Data Types: `double`

Output Arguments

`pat` — Radiation pattern of antenna or array or embedded pattern of array
matrix

Radiation pattern of antenna or array or embedded pattern of array, returned as a matrix of number of elevation values by number of azimuth values. The pattern is one of the following:

- `directivity` – Directivity in dBi (lossless antenna or array)
- `gain` – Gain in dBi (lossy antenna or array)
- `realizedgain` – Realized gain in dBi (lossy antenna or array)
- `efield` – Electric field in volt/meter
- `power` – Power in watts
- `powerdb` – Power in dB

Matrix size is number of elevation values multiplied by number of azimuth values.

`azimuth` — Azimuth angles of calculated radiation pattern
vector in degrees

Azimuth angles to calculate the radiation pattern, returned as a vector in degrees.

`elevation` — Elevation angles of calculate radiation pattern

vector in degrees

Elevation angles to calculate the radiation pattern, returned as a vector in degrees.

Version History

Introduced in R2015a

References

[1] Makarov, Sergey N. *Antenna and EM Modeling in MATLAB*. Chapter3, Sec 3.4 3.8. Wiley Inter-Science.

[2] Balanis, C.A. *Antenna Theory, Analysis and Design*, Chapter 2, sec 2.3-2.6, Wiley.

See Also

`EHfields` | `PatternPlotOptions` | `patternFromSlices` | `patternElevation` | `patternAzimuth`

Topics

"Field Analysis of Monopole Antenna"
"Field Calculation in Antennas"
"Radiation Pattern"
"Far-field Terminologies"

patternAzimuth

Azimuth pattern of antenna or array

Syntax

```
patternAzimuth(object,frequency,elevation)
patternAzimuth(object,frequency,elevation,Name,Value)
```

```
directivity = patternAzimuth(object,frequency,elevation)
directivity = patternAzimuth(object,frequency,elevation,'Azimuth')
```

Description

`patternAzimuth(object,frequency,elevation)` plots the 2-D radiation pattern of the antenna or array object over a specified frequency. Elevation values defaults to zero if not specified.

`patternAzimuth(object,frequency,elevation,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

`directivity = patternAzimuth(object,frequency,elevation)` returns the directivity of the antenna or array object over a specified frequency. Elevation values defaults to zero if not specified.

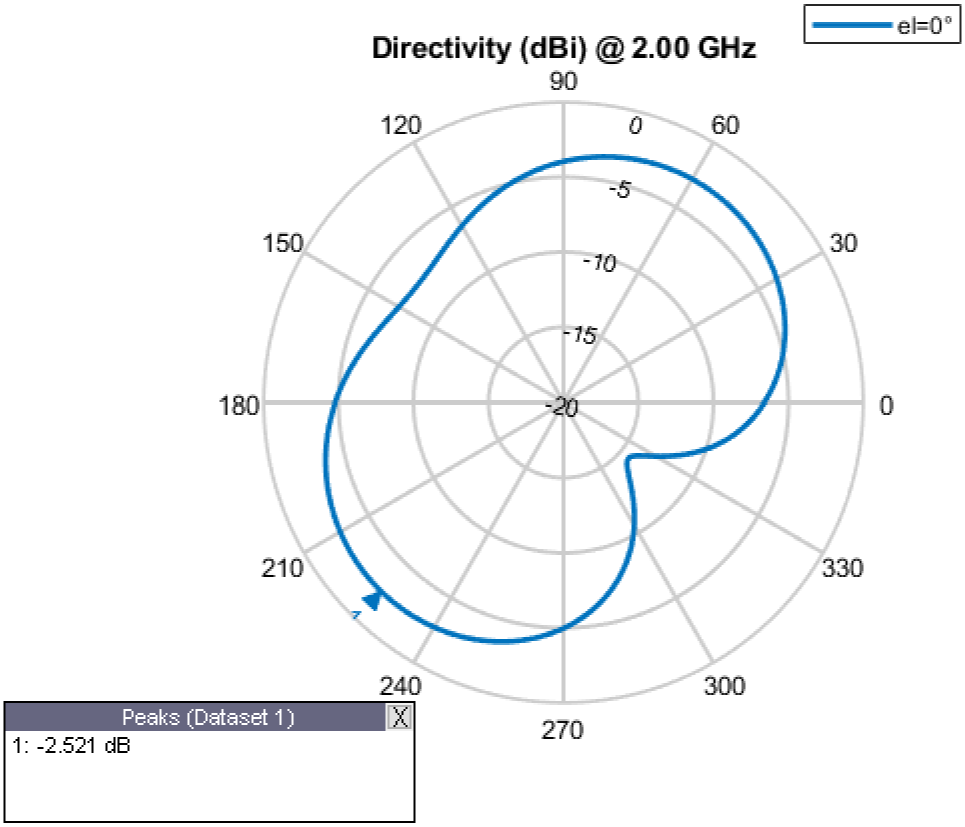
`directivity = patternAzimuth(object,frequency,elevation,'Azimuth')` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Azimuth Radiation Pattern of Helix Antenna

Calculate and plot the azimuth radiation pattern of the helix antenna at 2 GHz.

```
h = helix;
patternAzimuth(h,2e9);
```



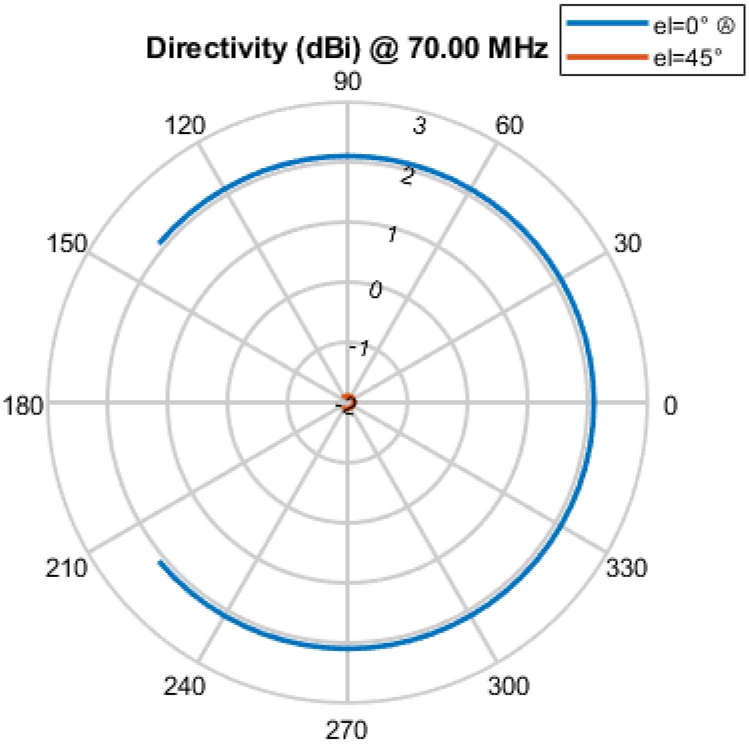
Peaks (Dataset 1)

1: -2.521 dB

Azimuth Radiation Pattern of Dipole Antenna

Calculate and plot the azimuth radiation pattern of the dipole antenna at 70 MHz at elevation values of 0 and 45.

```
d = dipole;
patternAzimuth(d,70e6,[0 45],'Azimuth',-140:5:140);
```



Input Arguments

`object` — antenna or array object

scalar

Antenna or array object, specified as a scalar.

`frequency` — Frequency used to calculate charge distribution

scalar in Hz

Frequency used to calculate charge distribution, specified as a scalar in Hz.

Example: 70e6

Data Types: `double`

`elevation` — Elevation angle values

vector in degrees

Elevation angle values, specified as a vector in degrees.

Example: `[0 45]`

Data Types: `double`

`'Azimuth'` — Azimuth angles of antenna

–180:1:180 (default) | vector in degrees

Azimuth angles of antenna, specified as the comma-separated pair consisting of `'Azimuth'` and a vector in degrees.

Example: `'Azimuth'`,2:2:340

Data Types: `double`

Output Arguments

`directivity` — Antenna or array directivity

matrix in `dBi`

Antenna or array directivity, returned as a matrix in `dBi`. The matrix size is the product of number of elevation values and number of azimuth values.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation` | `polarpattern`

patternMultiply

Radiation pattern of array using pattern multiplication

Syntax

```
patternMultiply(array,frequency)
patternMultiply(array,frequency,azimuth)
patternMultiply(array,frequency,azimuth, elevation)
patternMultiply( ___ ,Name,Value)
```

```
[fieldval,azimuth,elevation] = patternMultiply(array,frequency)
[fieldval,azimuth,elevation] = patternMultiply(array,frequency,azimuth)
[fieldval,azimuth,elevation] = patternMultiply(array,frequency,azimuth,
elevation)
[fieldval,azimuth,elevation] = patternMultiply( ___ ,Name,Value)
```

Description

`patternMultiply(array,frequency)` plots the 3-D radiation pattern of the array object over a specified frequency. `patternMultiply` calculates the full array pattern without taking the effect of mutual coupling between the different array elements.

`patternMultiply(array,frequency,azimuth)` plots the radiation pattern of the array object for the given azimuth angles. Elevation angles retain default values.

`patternMultiply(array,frequency,azimuth, elevation)` plots the radiation pattern of the array object for the given azimuth and elevation angles.

`patternMultiply( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. Specify name-value pair arguments after all other input arguments.

`[fieldval,azimuth,elevation] = patternMultiply(array,frequency)` returns the field value such as the directivity of the lossless array in dBi or gain of the lossy array in dBi at the specified frequency. The size of the field value matrix is (number of elevation values) x (number of azimuth values).

`[fieldval,azimuth,elevation] = patternMultiply(array,frequency,azimuth)` returns the field value at the specified azimuth angles. Elevation angles retain default values.

`[fieldval,azimuth,elevation] = patternMultiply(array,frequency,azimuth, elevation)` returns the field value at the specified azimuth angles, and elevation angles.

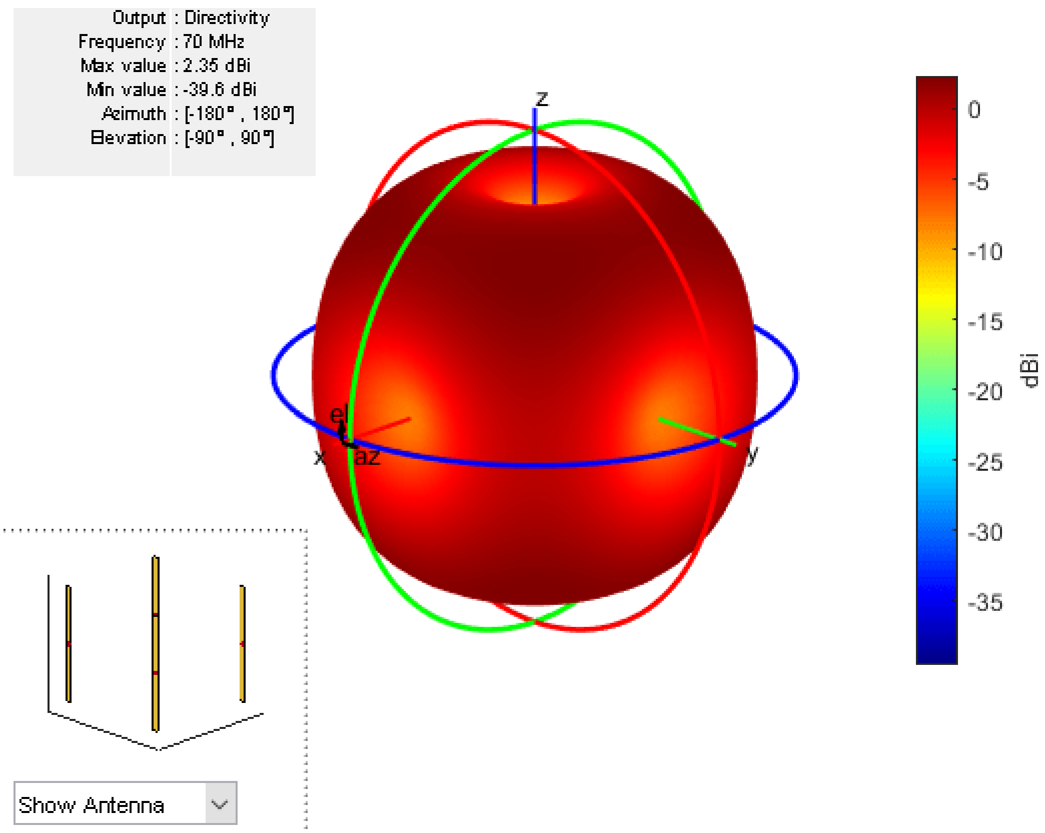
`[fieldval,azimuth,elevation] = patternMultiply( ___ ,Name,Value)` returns the field value using additional options specified by one or more `Name,Value` pair arguments. Specify name-value pair arguments after all other input arguments.

Examples

Radiation Pattern of Rectangular Array

Plot the radiation pattern of a default rectangular array at 70 MHz. Pattern multiplication does not take into consideration the effect of mutual coupling in array elements.

```
h = rectangularArray;
patternMultiply(h,70e6);
```

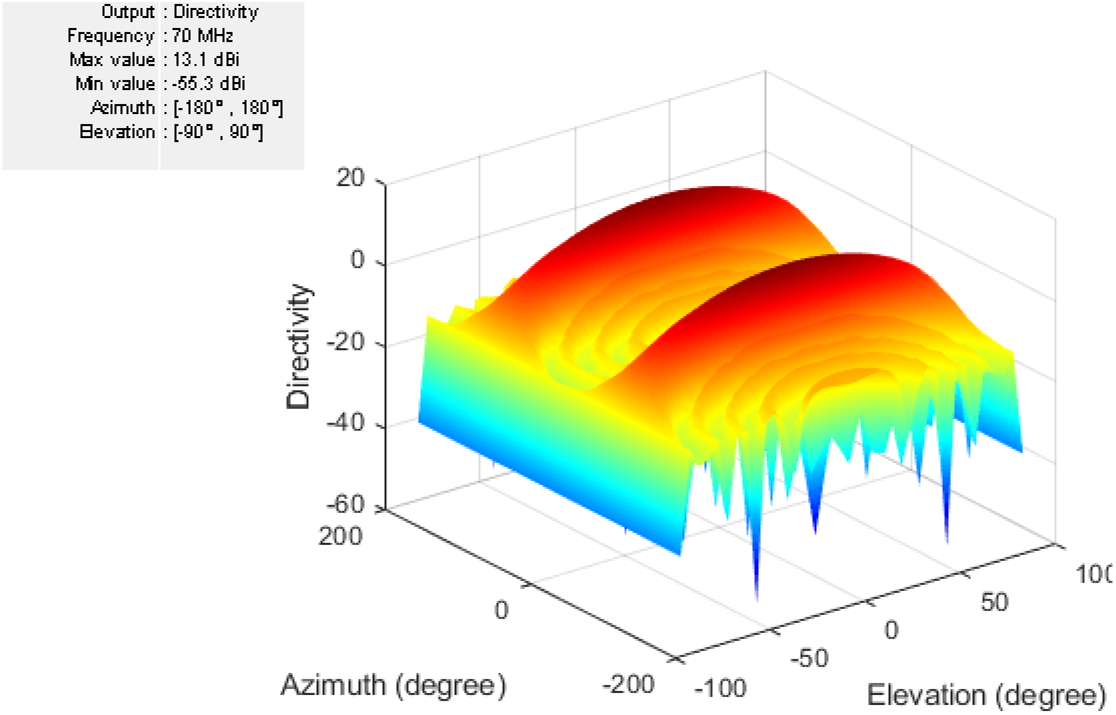


```
   Output : Directivity
Frequency : 70 MHz
Max value : 2.35 dBi
Min value : -39.6 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Radiation Pattern of Linear Array in Rectangular Coordinates

Plot the radiation pattern of a 10-element linear array at 70 MHz. Visualize the pattern using the rectangular coordinate system.

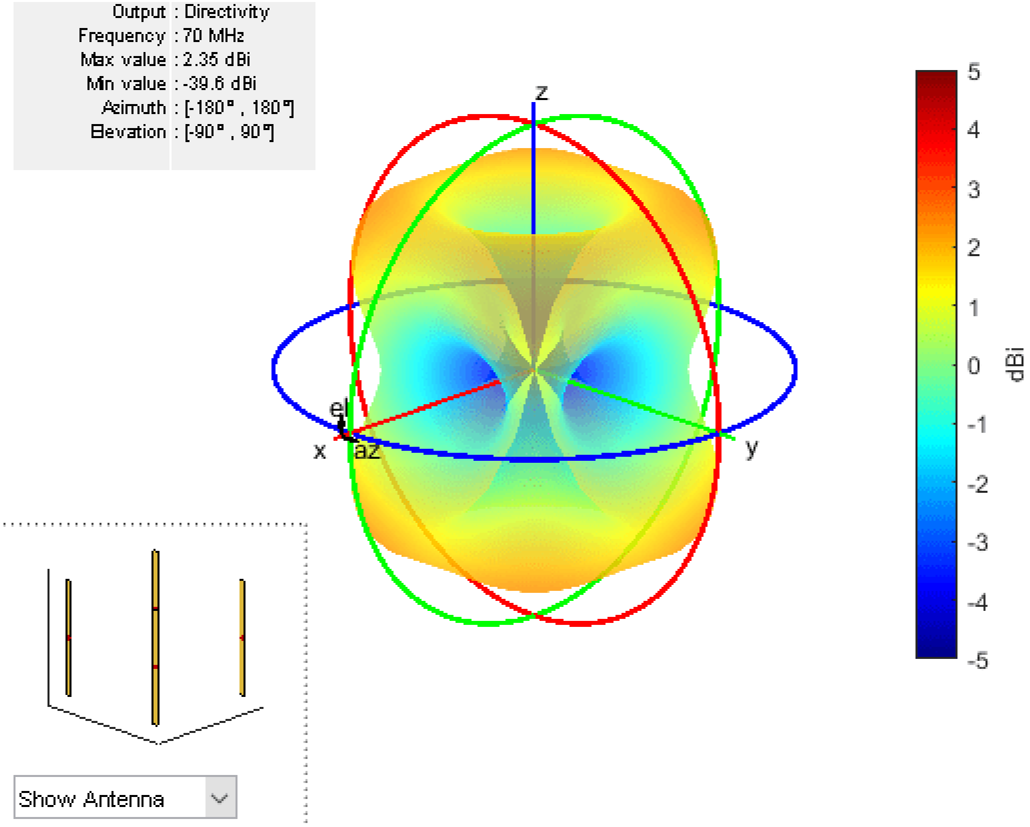
```
l = linearArray('NumElements',10);
patternMultiply(l,70e6,'CoordinateSystem','rectangular');
```



Rectangular Array Using Pattern Options

Plot the radiation pattern of a rectangular array with PatternOptions using a transparency of 0.6 and a magnitude scale of [-5 5].

```
h = rectangularArray;
p = PatternPlotOptions;
p.Transparency = 0.6;
p.MagnitudeScale = [-5 5];
patternMultiply(h, 70e6,'PatternOptions',p);
```



Output : Directivity
Frequency : 70 MHz
Max value : 2.35 dBi
Min value : -39.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Input Arguments

`array` — Input antenna array

object

Array object, specified as an object.

Example: `r = rectangularArray; patternMultiply(r,70e6)`. Plot the pattern of a rectangular array.

`frequency` — Frequency used to calculate array pattern

scalar in Hz

Frequency used to calculate array pattern, specified as a scalar in Hz.

Example: `70e6`

Data Types: `double`

`azimuth` — Azimuth angle of antenna

`–180:5:180` (default) | vector in degrees

Azimuth angle of the antenna, specified as a vector in degrees.

Example: `–90:5:90`

Data Types: `double`

`elevation` — Elevation angle of antenna

`–90:5:90` (default) | vector in degrees

Elevation angle of the antenna, specified as a vector in degrees.

Example: `0:1:360`

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` pair 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: `'CoordinateSystem'`, `rectangular`

`CoordinateSystem` — Coordinate system of radiation pattern

`'polar'` (default) | `'rectangular'` | `'uv'`

Coordinate system of radiation pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of these values: `'polar'`, `'rectangular'`, `'uv'`.

Example: `'CoordinateSystem'`, `'polar'`

Data Types: `char`

`Type` — Value to plot

`'directivity'` (default) | `'gain'` | `'efield'` | `'power'` | `'powerdb'`

Value to plot, specified as a comma-separated pair consisting of `'Type'` and one of these values:

- `'directivity'` – Radiation intensity in a given direction of antenna in dB
- `'gain'` – Radiation intensity in a given direction of antenna, when the antenna has a lossy substrate in dB
- `'efield'` – Electric field of antenna in volt/meter
- `'power'` – Antenna power in watts
- `'powerdb'` – Antenna power in dB

Example: `'Type'`, `'efield'`

Data Types: `char`

`Normalize` — Normalize field pattern

`true` (default) | `false` | boolean

Normalize field pattern, specified as the comma-separated pair consisting of `'Normalize'` and either `true` or `false`. For directivity patterns, this property is not applicable.

Example: `'Normalize'`, `false`

Data Types: `double`

`Polarization` — Field polarization

`'combined'` (default) | `'H'` | `'V'` | `'RHCP'` | `'LHCP'`

Field polarization, specified as the comma-separated pair consisting of `'Polarization'` and one of these values:

- `'combined'`
- `'H'` – Horizontal polarization
- `'V'` – Vertical polarization
- `'RHCP'` – Right-hand circular polarization
- `'LHCP'` – Left-hand circular polarization

By default, you can visualize a combined polarization.

Example: `'Polarization'`, `'RHCP'`

Data Types: `char`

`PatternOptions` — Parameter to change pattern plot properties
`PatternPlotOptions` object (default) | scalar

Parameter to change pattern plot properties, specified as the comma-separated pair consisting of `'PatternOptions'` and a `PatternPlotOptions` output. The properties that you can vary are:

- `Transparency`
- `SizeRatio`
- `AntennaOffset`
- `MagnitudeScale`

Example: `p = PatternPlotOptions('Transparency',0.1);` Create a pattern plot option with a transparency of 0.1. `antennaarray = linearArray;patternMultiply(antennaarray,75e6,'PatternOptions',p)`; Use this pattern plot option to visualize the pattern of a helix antenna.

Data Types: `double`

Output Arguments

`fieldval` — Array directivity or gain
matrix in `dBi`

Array directivity or gain, returned as a matrix in `dBi`. The matrix size is the product of the number of elevation values and azimuth values.

`azimuth` — Azimuth angles
vector in degrees

Azimuth angle used to calculate field values, returned as a vector in degrees.

`elevation` — Elevation angles
vector in degrees

Elevation angles used to calculate field values, returned as a vector in degrees.

Version History

Introduced in R2017a

See Also

`pattern` | `patternElevation`

Topics

"Antenna Toolbox Coordinate System"

patternElevation

Elevation pattern of antenna or array

Syntax

```
patternElevation(object,frequency,azimuth)
patternElevation(object,frequency,azimuth,Name,Value)
```

```
directivity = patternElevation(object,frequency,azimuth)
directivity = patternElevation(object,frequency,azimuth,'Elevation')
```

Description

`patternElevation(object,frequency,azimuth)` plots the 2-D radiation pattern of the antenna or array object over a specified frequency. Azimuth values defaults to zero if not specified.

`patternElevation(object,frequency,azimuth,Name,Value)` uses additional options specified by one or more `Name, Value` pair arguments.

`directivity = patternElevation(object,frequency,azimuth)` returns the directivity of the antenna or array object at specified frequency. Azimuth values defaults to zero if not specified.

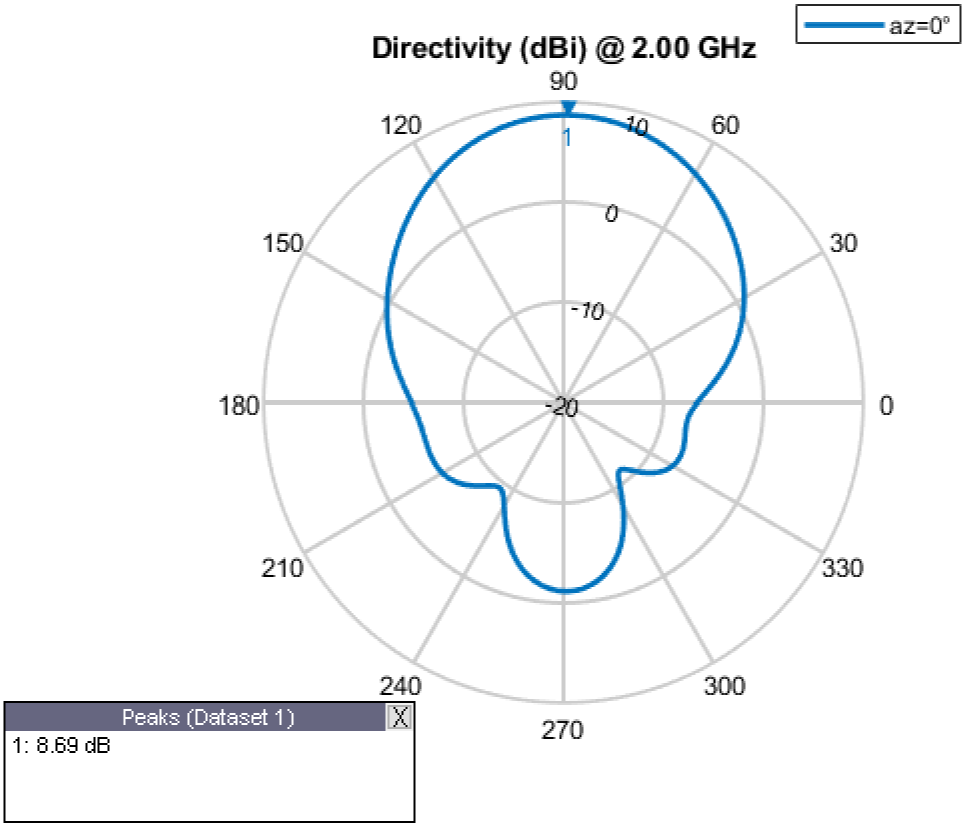
`directivity = patternElevation(object,frequency,azimuth,'Elevation')` uses additional options specified by one or more `Name, Value` pair arguments.

Examples

Elevation Radiation Pattern of Helix

Calculate and plot the elevation pattern of the helix antenna at 2 GHz.

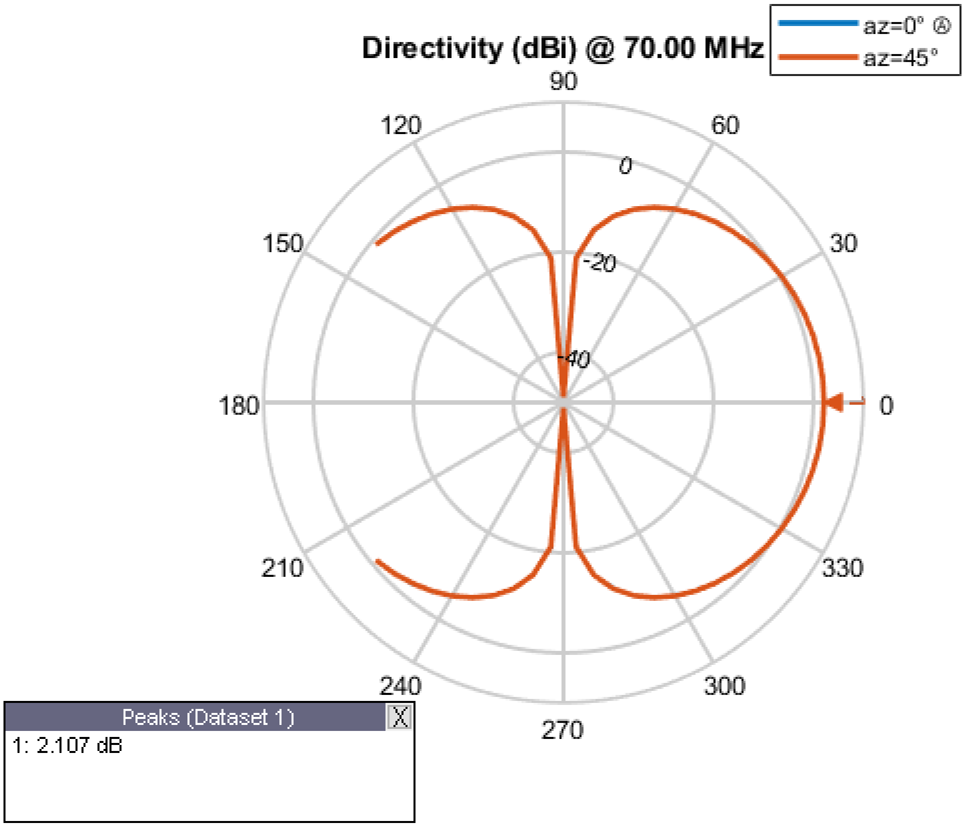
```
h = helix;
patternElevation (h, 2e9);
```



Elevation Radiation Pattern of Dipole Antenna

Calculate and plot the elevation radiation pattern of the dipole antenna at 70 MHz at elevation values of 0 and 45.

```
d = dipole;
patternElevation(d,70e6,[0 45],'Elevation',-140:5:140);
```



Input Arguments

`object` — Antenna or array object

scalar

Antenna or array object, specified as a scalar.

`frequency` — Frequency used to calculate charge distribution

scalar in Hz

Frequency used to calculate charge distribution, specified as a scalar in Hz.

Example: 70e6

Data Types: `double`

`azimuth` — Azimuth angle values

vector in degrees

Azimuth angle values, specified as a vector in degrees.

Example: `[0 45]`

Data Types: `double`

`'Elevation'` — Elevation angles of antenna

–90:1:90 (default) | vector in degrees

Elevation angles of antenna, specified the comma-separated pair consisting of `'Elevation'` and a vector in degrees.

Example: `'Elevation'`, 0:1:360

Data Types: `double`

Output Arguments

`directivity` — Antenna or array directivity

matrix in `dBi`

Antenna or array directivity, returned as a matrix in `dBi`. The matrix size is the product of number of elevation values and number of azimuth values.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth` | `polarpattern`

current

Current distribution on antenna or array surface

Syntax

```
current(object,frequency)

i = current(object,frequency)
[i,p] = current(object,frequency)

current(object,frequency,'dielectric')
[ ___ ] = current(object,frequency,'dielectric')
[ ___ ] = current( ___ ,Name=Value)
```

Description

`current(object,frequency)` calculates and plots the absolute value of the current on the surface of the specified antenna or array `object` at the specified frequency.

`i = current(object,frequency)` returns the components of the antenna or array surface current at the specified frequency in a Cartesian coordinate system.

`[i,p] = current(object,frequency)` also returns the point at which the function performs the current calculation.

`current(object,frequency,'dielectric')` calculates and plots the absolute value of the current at the specified frequency on the dielectric face of the antenna or array.

`[ ___ ] = current(object,frequency,'dielectric')` returns the components of the current on the dielectric surface of the antenna or array at the specified frequency in a Cartesian coordinate system.

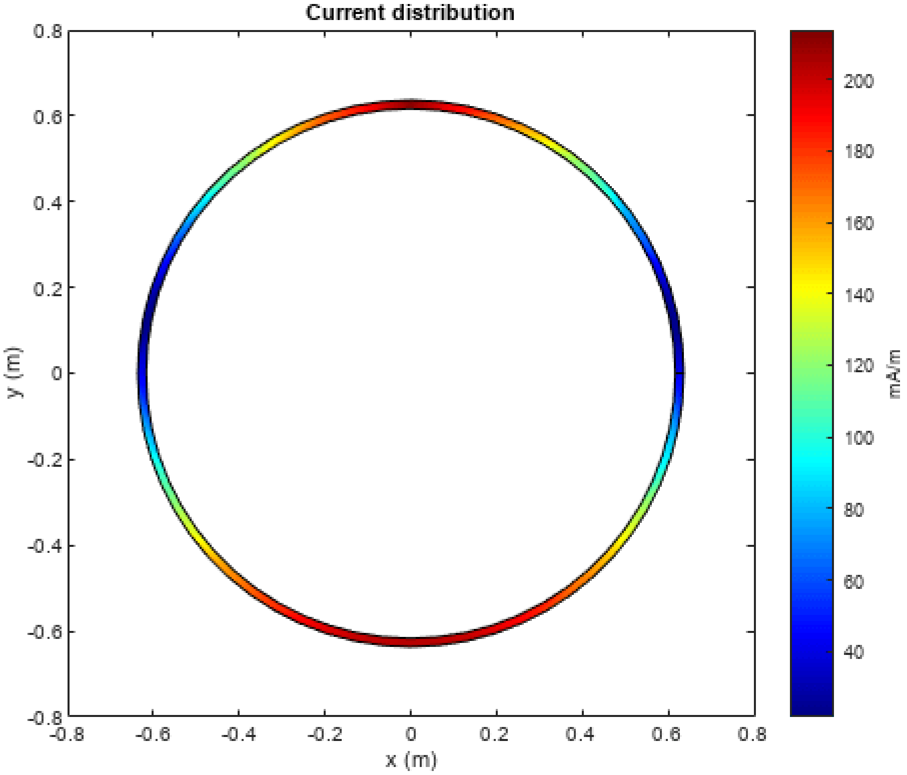
`[ ___ ] = current( ___ ,Name=Value)` specifies additional options using one or more name-value arguments.

Examples

Calculate and Plot Current Distribution on Antenna Surface

Calculate and plot the current distribution for a circular loop antenna at 70MHz frequency.

```
h = loopCircular;
current(h,70e6);
```



Calculate Current Distribution of Array

Calculate the current distribution of a default rectangular array at 70MHz frequency.

```
h = rectangularArray;
i = current(h,70e6)
```

```
i = 3×160 complex

   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0012 + 0.0024i  -0.0012 - 0.0023i   0.0031 + 0.0053i  -0.0021 - 0.0035i   0.0017 + 0.0030i
   0.0447 + 0.0790i   0.0494 + 0.0883i   0.0031 + 0.0053i   0.0135 + 0.0228i   0.0173 + 0.0294i
```

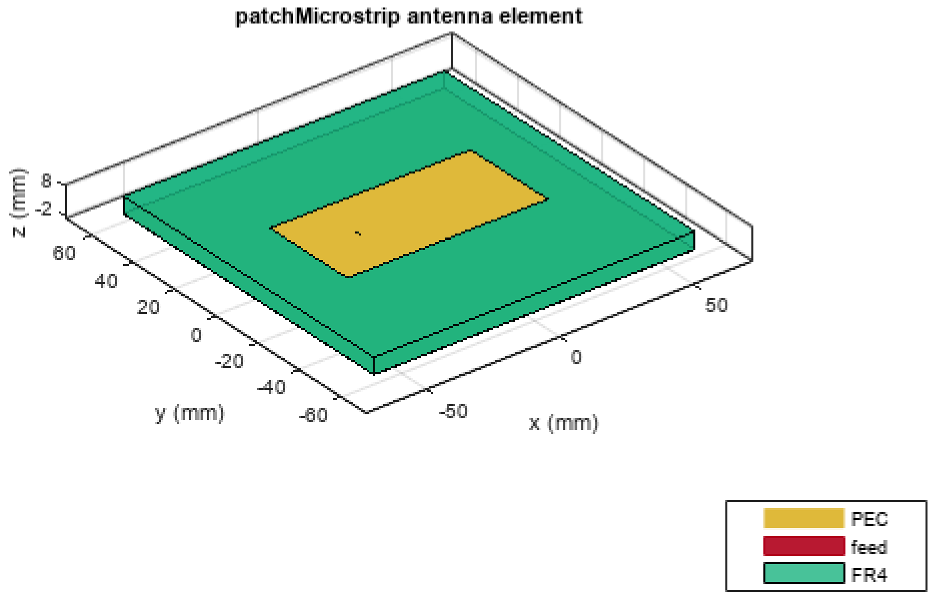
Current Distribution on Microstrip Patch Antenna

Create a microstrip patch antenna with a FR4 dielectric substrate.

```
d = dielectric('FR4');
pm = patchMicrostrip(Length=75e-3,Width=37e-3,...
        GroundPlaneLength=120e-3,GroundPlaneWidth=120e-3,...
        Substrate=d)
```

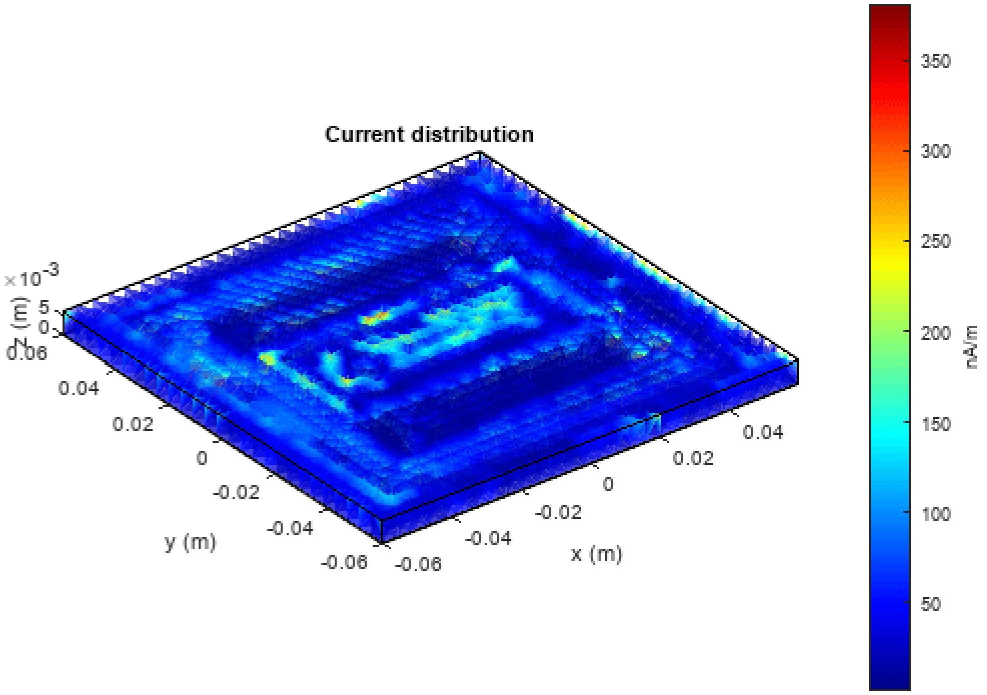
```
pm = 
  patchMicrostrip with properties:

                Length: 0.0750
                 Width: 0.0370
                Height: 0.0060
             Substrate: [1x1 dielectric]
     GroundPlaneLength: 0.1200
      GroundPlaneWidth: 0.1200
     PatchCenterOffset: [0 0]
            FeedOffset: [-0.0187 0]
             Conductor: [1x1 metal]
                  Tilt: 0
             TiltAxis: [1 0 0]
                  Load: [1x1 lumpedElement]
```

```
show(pm)
```



Plot the current distribution on the antenna at a frequency of 1.67 GHz.

```
figure
current(pm,1.67e9,'dielectric')
```



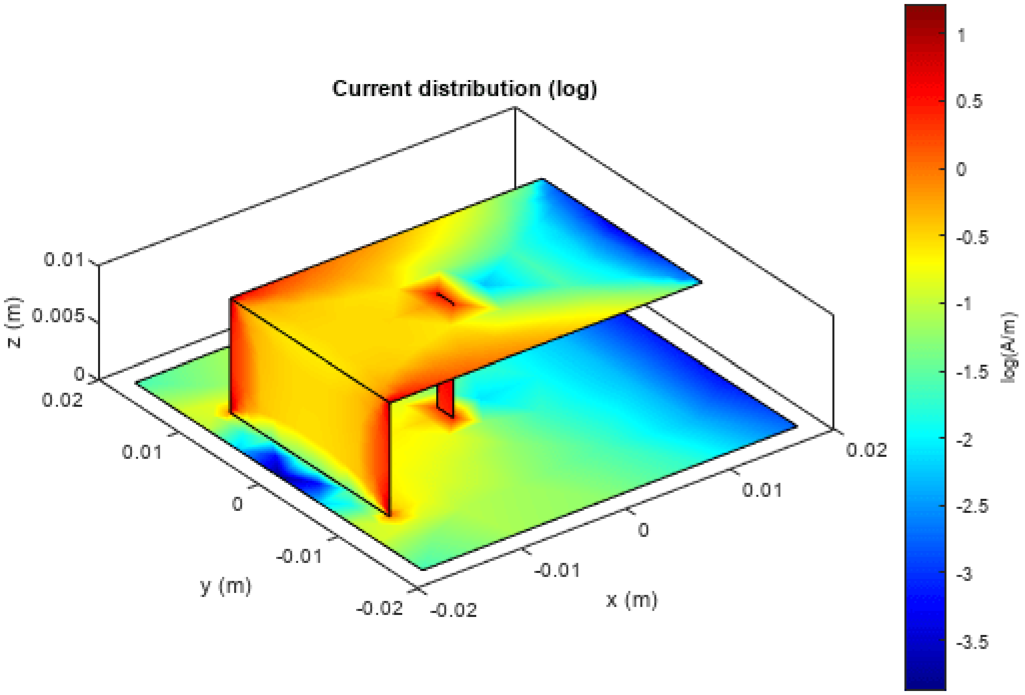
Logarithmic Current Distribution on Antenna Surface

Create a default pifa (planar inverted F antenna).

```
ant = pifa;
```

Visualize the current distribution on the pifa antenna using log function scale.

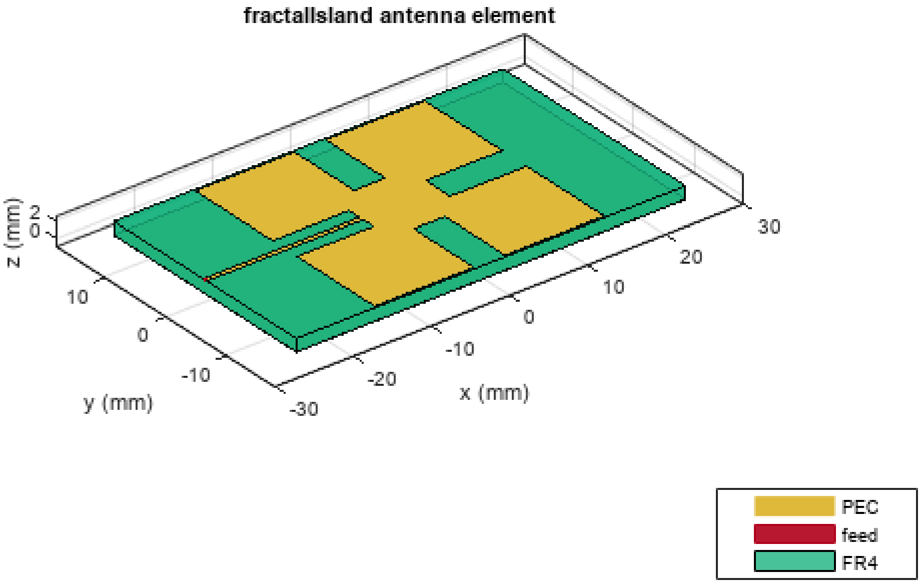
```
current(ant,1.75e9,Scale="log")
```



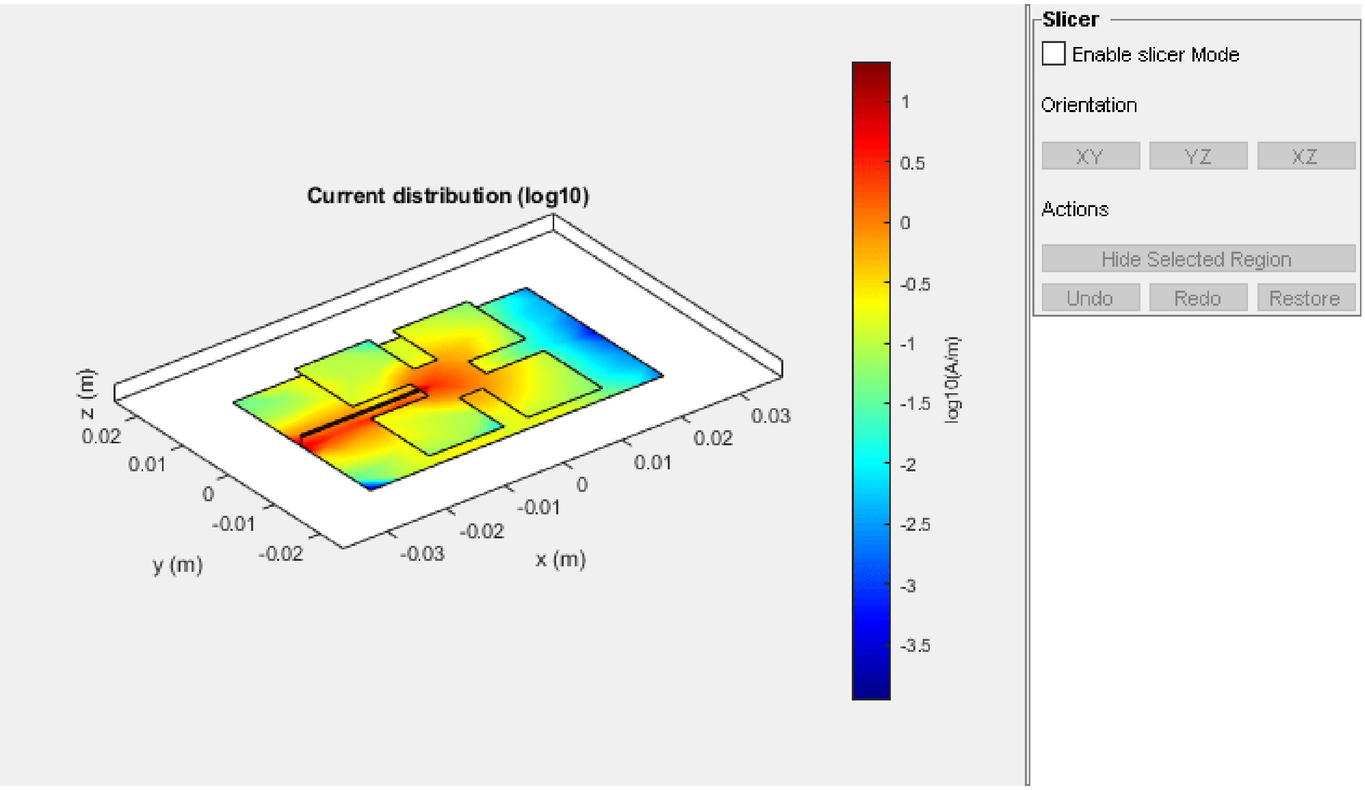
View Sliced Current Distribution Plot of Antenna

Create Minkowski's island fractal antenna with an FR4 dielectric substrate. Plot the current distribution of the antenna at 1 GHz with the `'Slicer'` argument set to `'on'`.

```
ant = fractalIsland(Substrate=dielectric('FR4'));
show(ant)
```

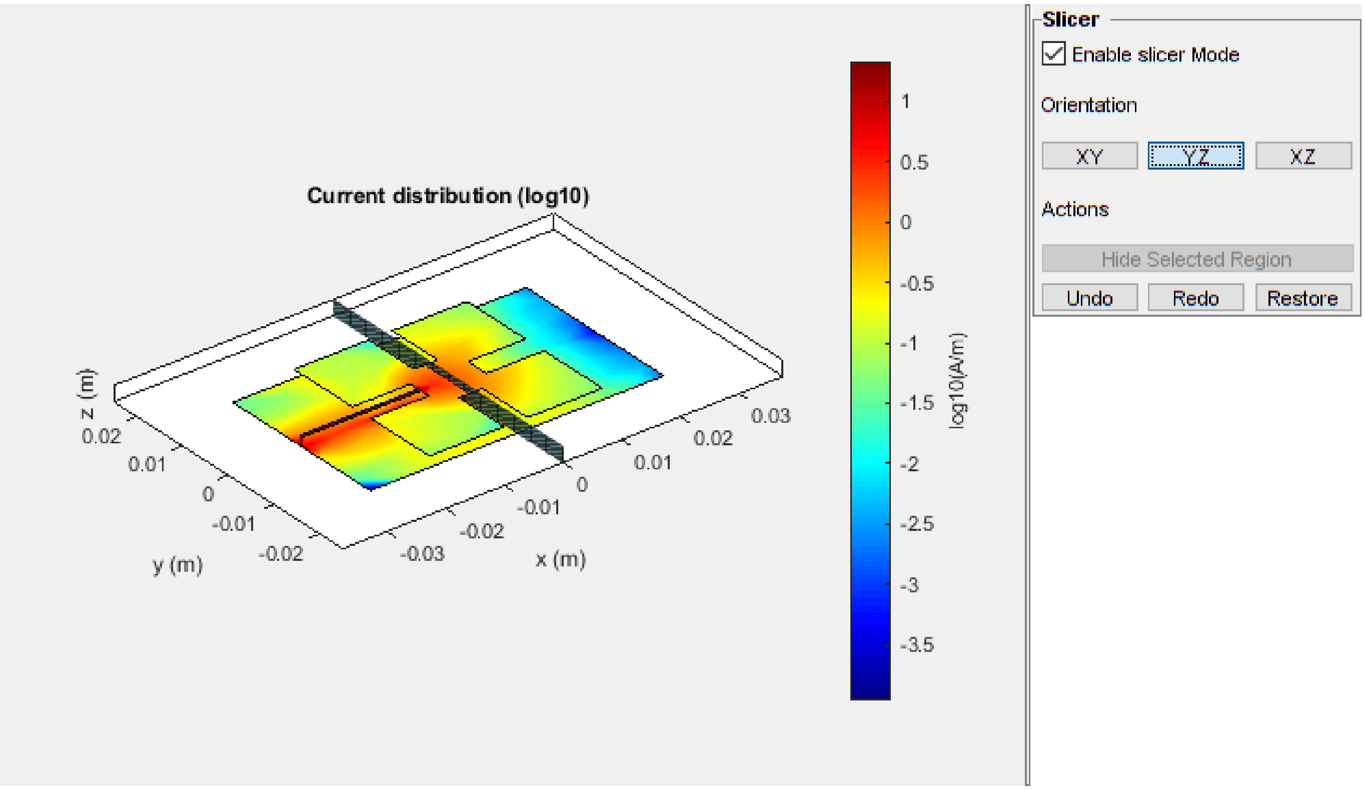


```
current(ant,1e9,Slicer="on",Scale="log10")
```



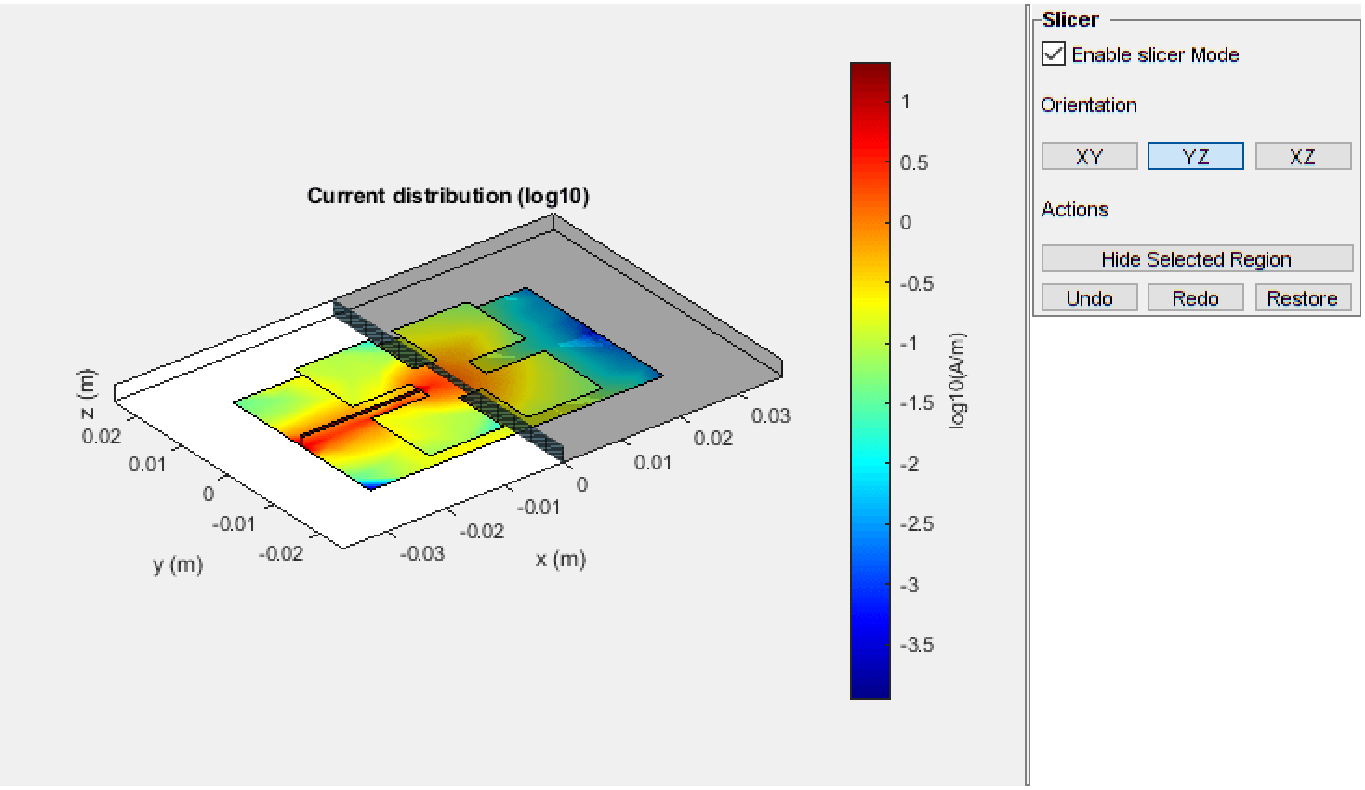
☐ Enable slicer Mode

Select **Enable slicer Mode**. Then select a plane for the slice under **Orientation**.



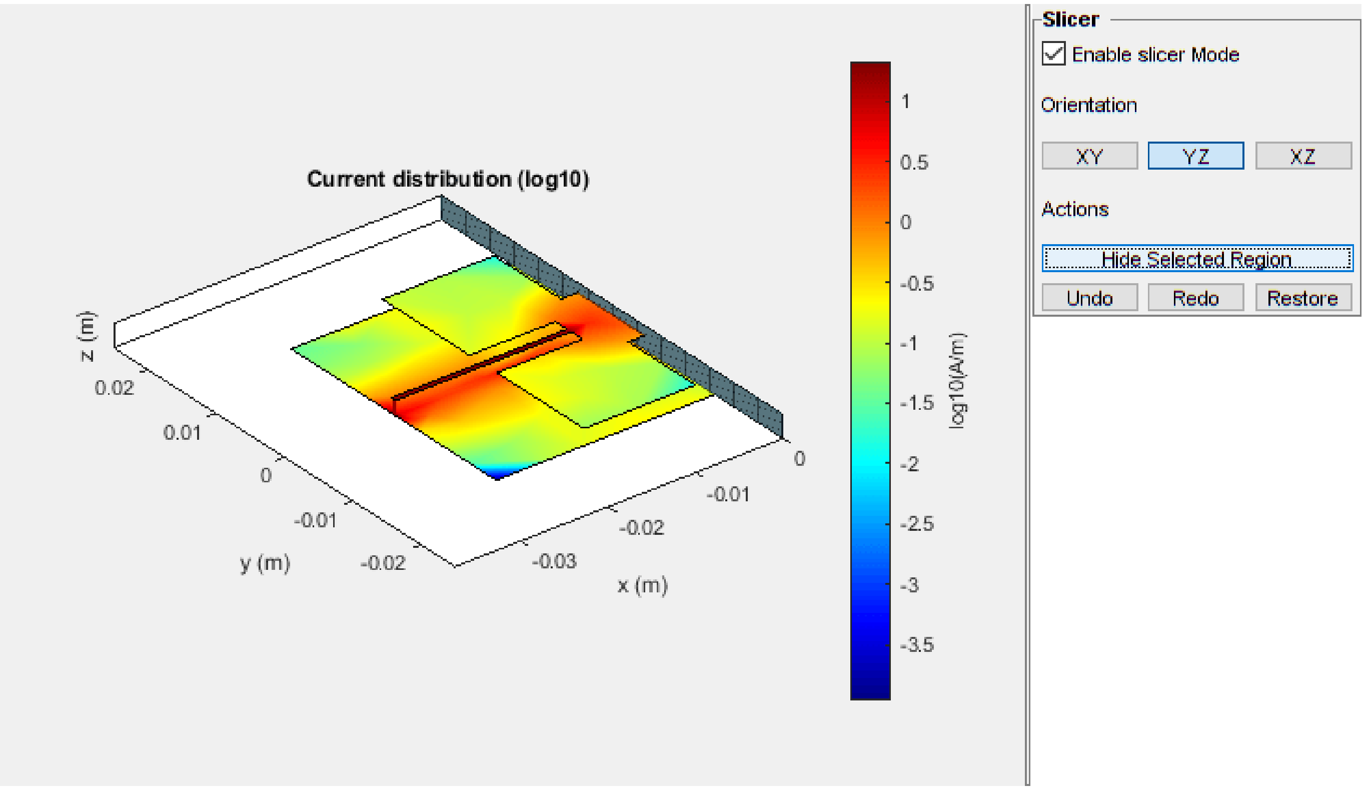
☑ Enable slicer Mode

Click on the plot and select a region to hide.



☑ Enable slicer Mode

Click **Hide Selected Region** to view the desired slice.



Input Arguments

`object` — Antenna or array object

scalar

Antenna or array object, specified as a scalar.

`frequency` — Frequency used to calculate current distribution

scalar in Hz

Frequency to calculate current distribution, specified as a scalar in Hz.

Example: 70e6

Data Types: `double`

Name-Value Arguments

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

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

Example: `Scale="log10"`

`Region` — Layer to plot current distribution

`"metal"` (default) | `"dielectric"`

Layer to plot the current distribution, specified as either `"metal"` or `"dielectric"`. Use this argument to choose between metal and dielectric layers and plot the current distribution at the chosen layer.

Example: `Region="dielectric"`

Data Types: `string`

`Scale` — Scale to visualize current distribution
`"linear"` (default) | `"log"` | `"log10"`

Scale of the current distribution plot to visualize the current distribution on the surface of the antenna, specified as a string or a logarithmic function. The default scale is `"linear"`.

Example: `Scale="log10"`

Data Types: `string` | `function`

`Slicer` — Option to enable or disable plot interactivity
`0` (default) | `1` | `false` | `true` | `"off"` | `"on"`

Option to enable or disable plot interactivity, specified as `"on"` or `"off"`, or as numeric or logical `1(true)` or `0(false)`. Set this argument to `1` or `"on"` to open the plot with the slicer panel, and to slice and view the desired cross section of the plot along the *xy*-, *yz*-, and *xz*- planes. Set this argument to `0` or `"off"` to open the plot without the slicer panel.

Data Types: `string` | `logical`

`Type` — Type of current distribution plot
`"absolute"` (default) | `"real"` | `"imaginary"`

Type of the current distribution plot, specified as `"absolute"`, `"real"`, or `"imaginary"`. The default type is `"absolute"`. Use this name-value argument to choose between absolute, real, and imaginary values of the current against which to plot the current distribution.

Example: `Type="imaginary"`

Data Types: `string`

`Direction` — Option to display direction of current vector
`"off"` (default) | `"on"`

Option to display the direction of the current vector, specified as `"on"` or `"off"`. To display the direction of the current vector on the current distribution plot, specify this argument as `"on"`. Otherwise, specify this argument as `"off"`.

Example: `Direction="on"`

Data Types: `string`

Output Arguments

`i` — Components of current distribution in Cartesian coordinate system
3-by-*n* complex matrix in A/m

x, *y*, *z* components of the current distribution, returned as a 3-by-*n* complex matrix in A/m. The value of the current is calculated on every triangle mesh or every dielectric tetrahedron face on the surface of an antenna or array.

p — Cartesian coordinates representing center of each triangle in mesh
3-by-*n* real matrix

Cartesian coordinates representing the center of each triangle in the mesh, returned as a 3-by-*n* real matrix.

Version History

Introduced in R2015a

See Also

`charge` | `axialRatio`

charge

Charge distribution on antenna or array surface

Syntax

```
charge(object,frequency)

c = charge(object,frequency)
[c,p] = charge(object,frequency)

charge(object,frequency,'dielectric')
c = charge(object,frequency,'dielectric')
c = charge( ___ ,Name,Value)
```

Description

`charge(object,frequency)` calculates and plots the absolute value of the charge on the surface of an antenna or array object surface at a specified frequency.

`c = charge(object,frequency)` returns a vector of charges in C/m on the surface of an antenna or array object, at a specified frequency.

`[c,p] = charge(object,frequency)` returns a vector of charges in C/m on the surface of an antenna or array object, at a specified frequency and at the point at which the charge calculation was performed.

`charge(object,frequency,'dielectric')` calculates and plots the absolute value of charge at a specified frequency value on the dielectric face of the antenna or array.

`c = charge(object,frequency,'dielectric')` returns the *x*, *y*, *z* components of the charge on the dielectric surface of an antenna or array object, at a specified frequency.

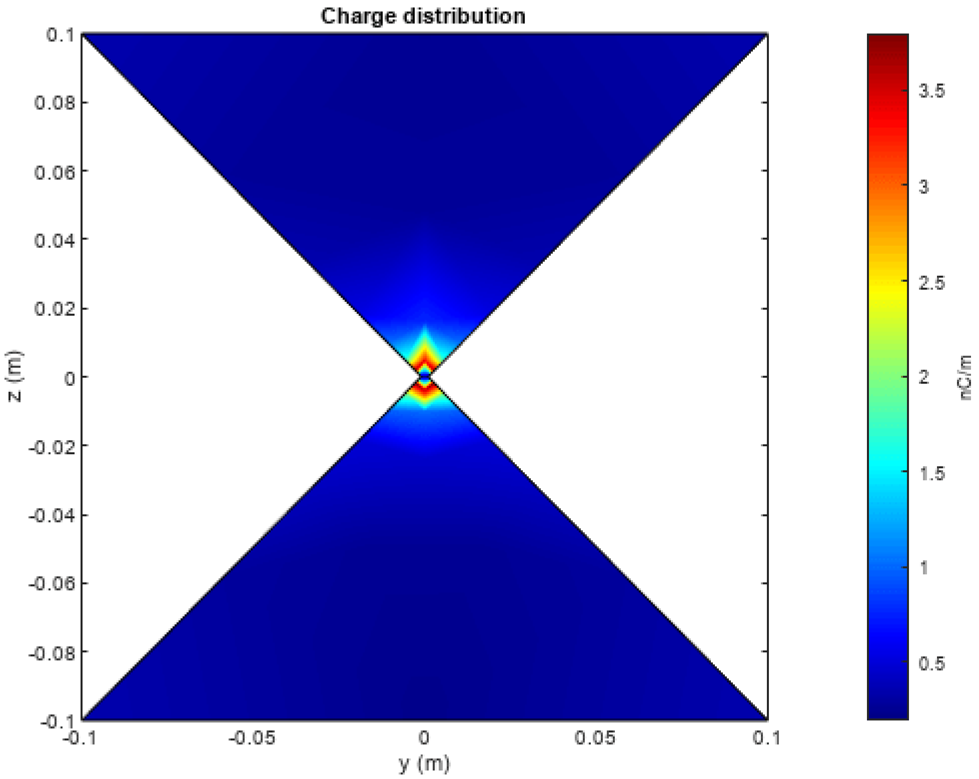
`c = charge( ___ ,Name,Value)` calculates the charge on the surface of an antenna using one or more name-value pairs.

Examples

Calculate and Plot Charge Distribution on Antenna Surface

Calculate and plot the charge distribution on a bowtieTriangular antenna at 70MHz frequency.

```
h = bowtieTriangular;
charge (h, 70e6);
```



Calculate Charge Distribution of Array

Calculate charge distribution of linear array at 70 MHz frequency.

```
h = linearArray;
h.NumElements = 4;
C = charge(h,70e6);
```

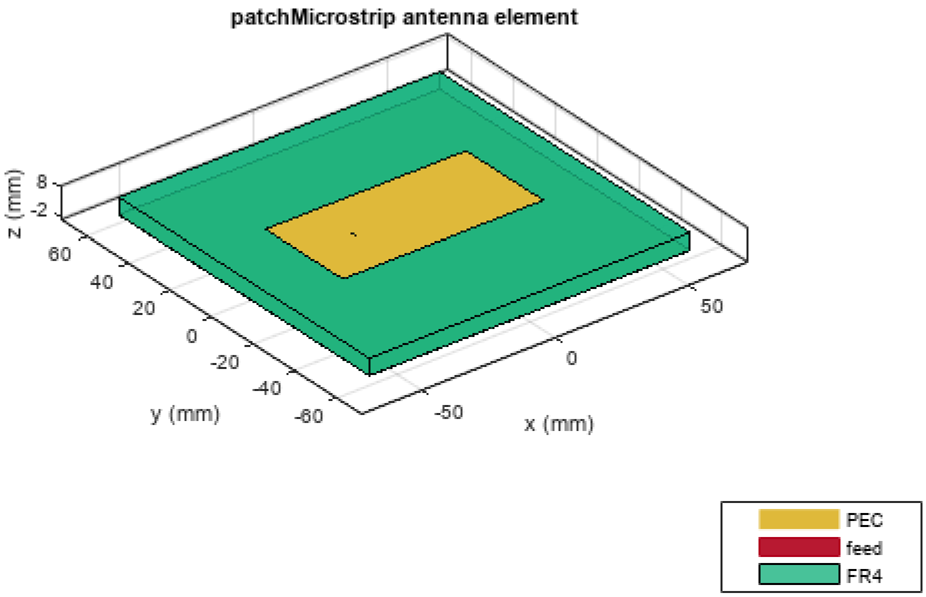
Charge Distribution On Microstrip Patch Antenna

Create a microstrip patch antenna using **'FR4'** as the dielectric substrate.

```
d = dielectric('FR4');
pm = patchMicrostrip('Length',75e-3, 'Width',37e-3,                 ...
        'GroundPlaneLength',120e-3, 'GroundPlaneWidth',120e-3, ...
        'Substrate',d)
```

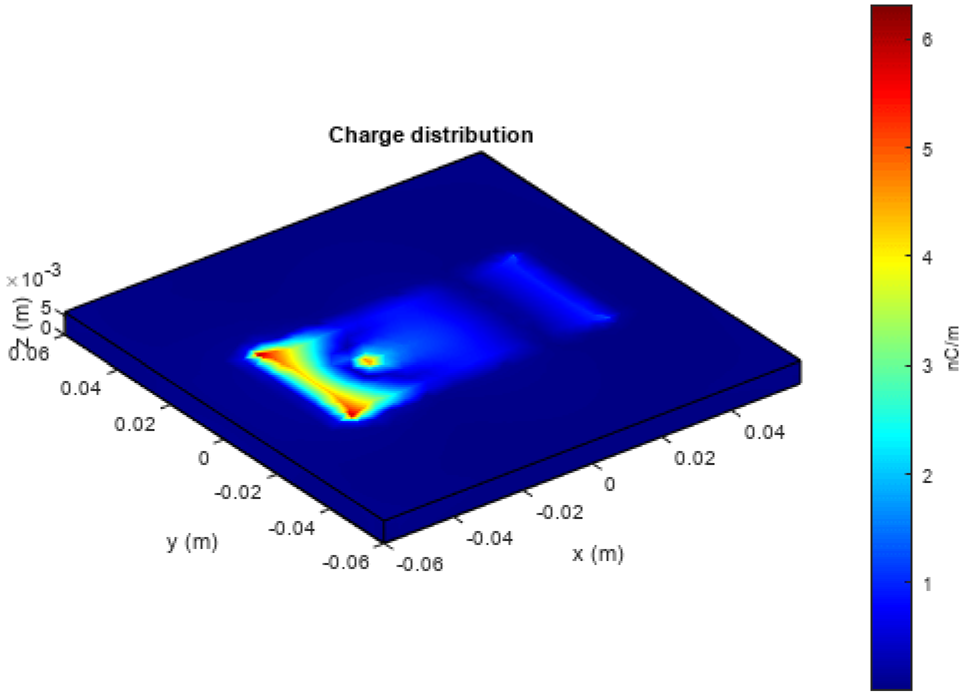
```
pm = 
  patchMicrostrip with properties:

               Length: 0.0750
                Width: 0.0370
               Height: 0.0060
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1200
     GroundPlaneWidth: 0.1200
    PatchCenterOffset: [0 0]
           FeedOffset: [-0.0187 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(pm)
```



Plot the charge distribution on the antenna at a frequency of 1.67 GHz.

```
figure
charge(pm,1.67e9,'dielectric')
```



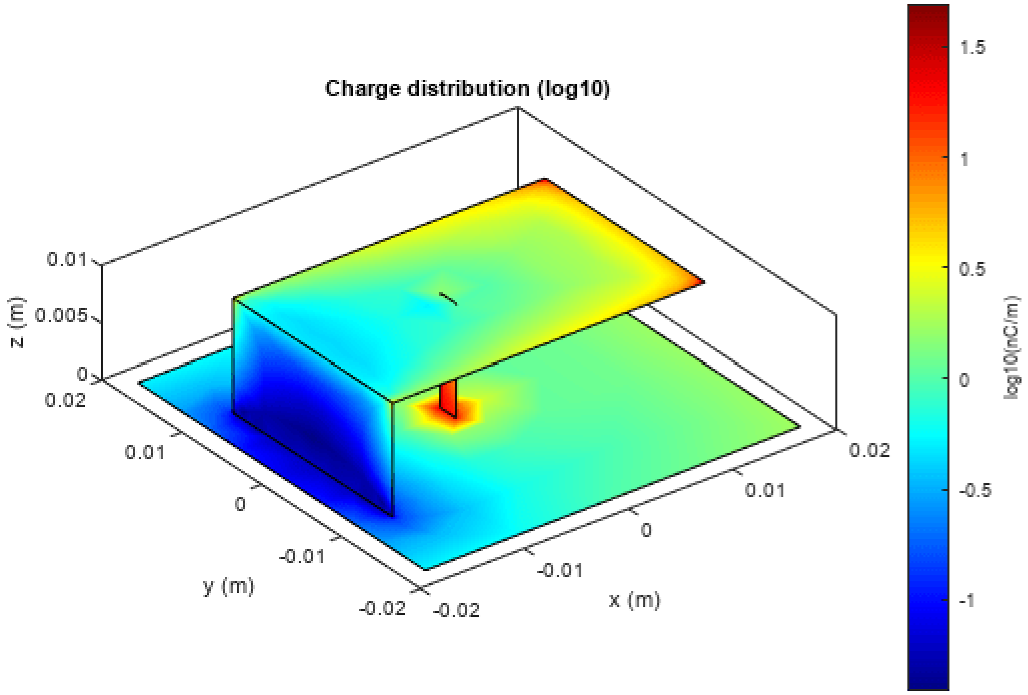
Logarithmic Charge Distribution on Antenna Surface

Create a default pifa (planar inverted F antenna).

```
ant = pifa;
```

Visualize the charge distribution on the pifa antenna in log10 scale.

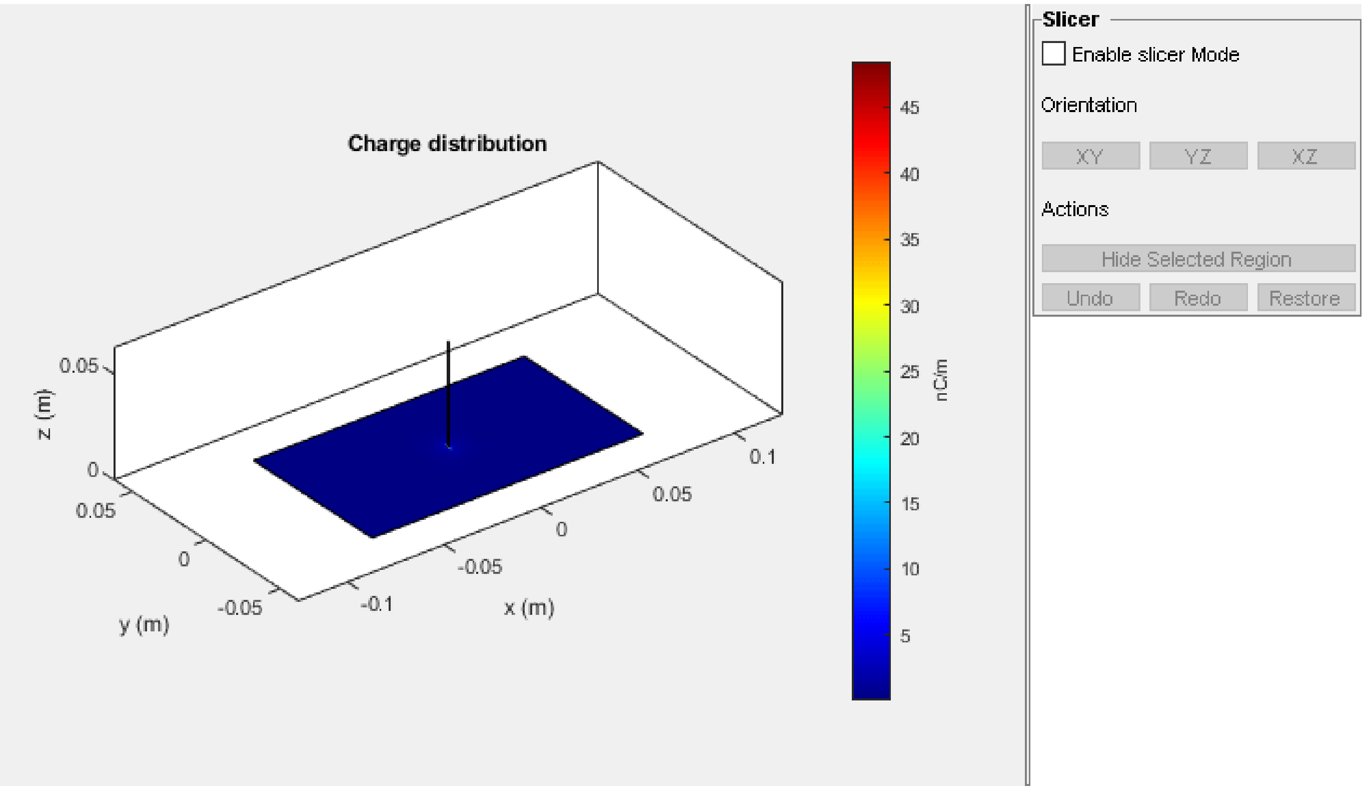
```
charge(ant,1.75e9,'Scale','log10')
```



View Sliced Charge Distribution Plot of Antenna

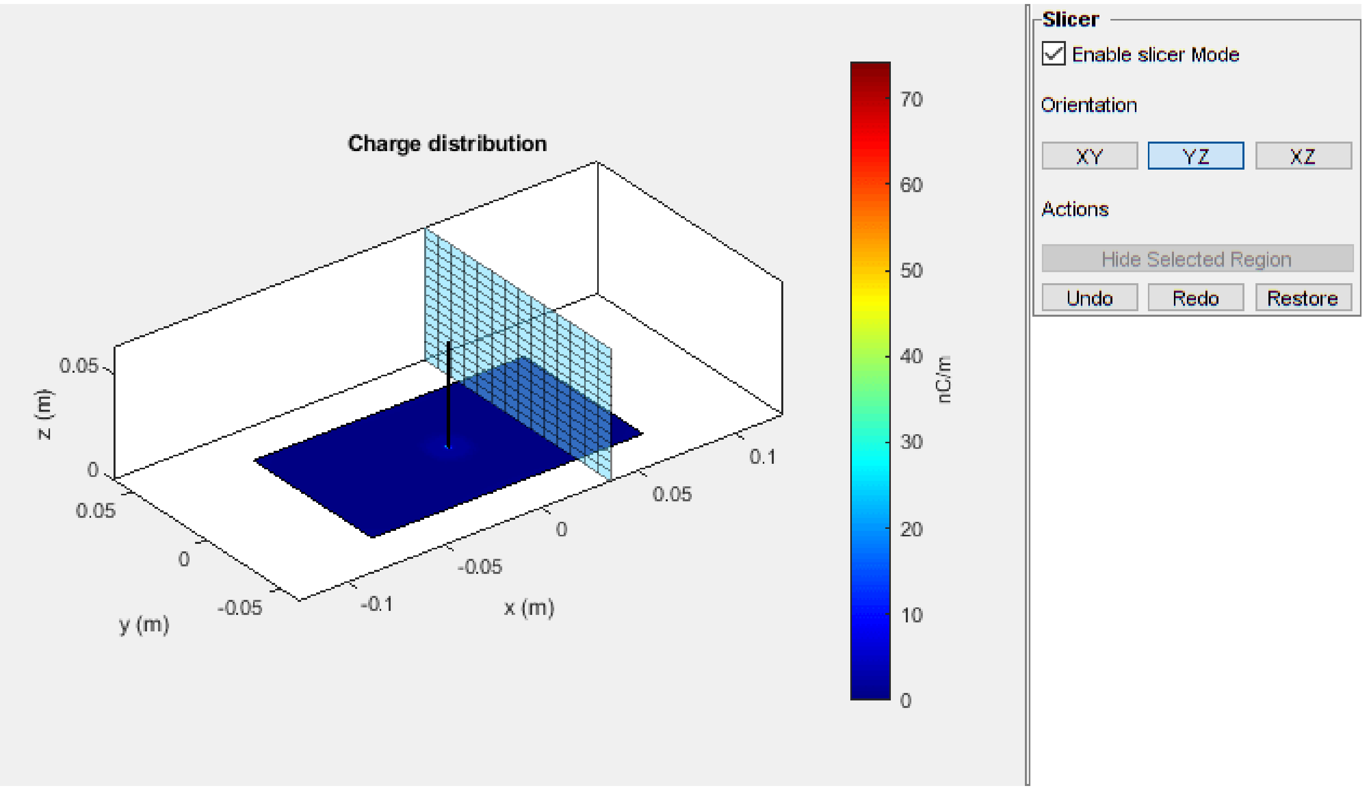
Create a default cylindrical dielectric resonator antenna. Plot the charge distribution of the antenna at 1GHz with the 'Slicer' argument set to 'on'.

```
ant = draCylindrical('ResonatorRadius',0.01);
charge(ant,1e9,'Slicer','on')
```



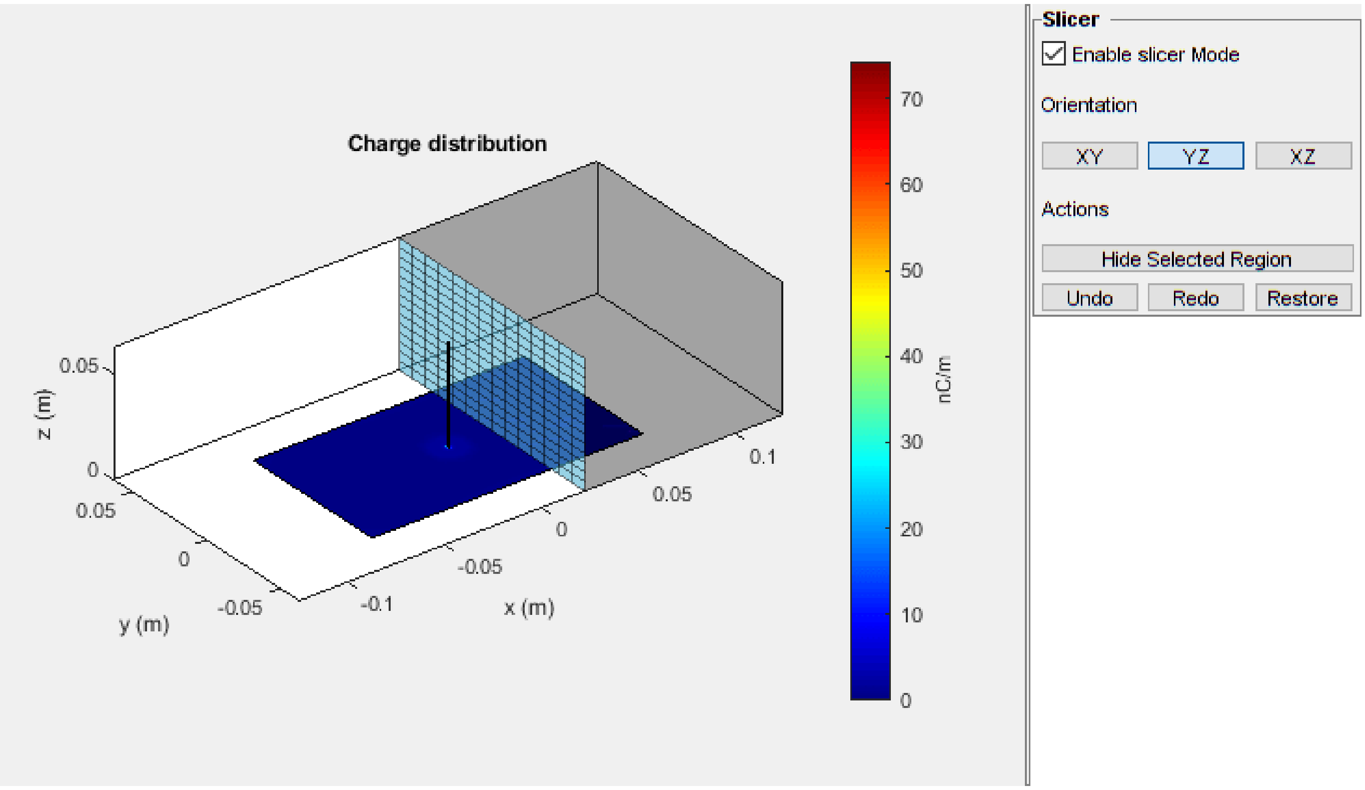
☐ Enable slicer Mode

Select **Enable slicer Mode**. Then select a plane for the slice under **Orientation**.

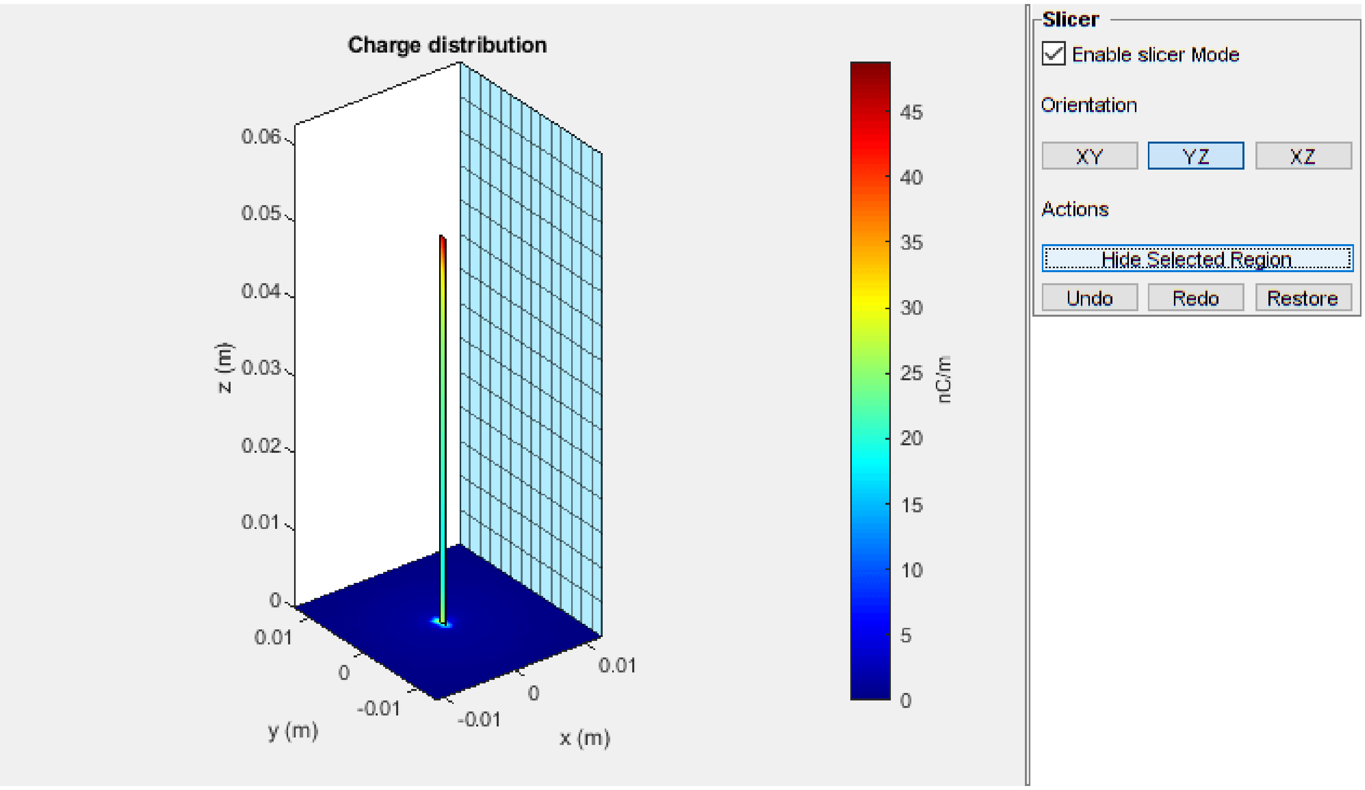


☑ Enable slicer Mode

Click on the plot and select a region to hide.



Click **Hide Selected Region** to view the desired slice.



Input Arguments

`object` — Antenna or array object

scalar

Antenna or array object, specified as a scalar.

`frequency` — Frequency used to calculate charge distribution

scalar in Hz

Frequency used to calculate charge distribution, specified as a scalar in Hz.

Example: 70e6

Data Types: `double`

Name-Value Arguments

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

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

Example: `'Scale','log10'`

`Scale` — Scale to visualize charge distribution

`'linear'` (default) | `'log'` | `'log10'`

Scale of the charge distribution plot to visualize the charge distribution on the surface of the antenna, specified as a string or a logarithmic function. The default scale is `'linear'`.

Example: `'Scale','log10'`

Data Types: `string` | `function`

`Slicer` — Option to enable or disable plot interactivity
`0` (default) | `1` | `'off'` | `'on'`

Option to enable or disable plot interactivity, specified as `'on'` or `'off'`, or as numeric or logical `1(true)` or `0(false)`. Set this argument to `1` or `'on'` to open the plot with the slicer panel, and to slice and view the desired cross section of the plot along the *xy*-, *yz*-, and *xz*- planes. Set this argument to `0` or `'off'` to open the plot without the slicer panel.

Example: `'Slicer','on'`

Data Types: `string` | `logical`

Output Arguments

c — Complex charges
1-by-*n* vector in C/m

Complex charges, returned as a 1-by-*n* vector in C/m. This value is calculated on every triangle mesh or every dielectric tetrahedron face on the surface of an antenna or array.

p — Cartesian coordinates representing center of each triangle in mesh
3-by-*n* real matrix

Cartesian coordinates representing the center of each triangle in the mesh, returned as a 3-by-*n* real matrix.

Version History
Introduced in R2015a

See Also
`current` | `EHfields`

design

Design prototype antenna or arrays for resonance around specified frequency

Syntax

```
hant = design(antenna,frequency)

harray = design(array,frequency)
harray = design(array,frequency,elements)

harray = design(conformalarray,frequency)
harray = design(conformalarray,frequency,elements)

harray = design(infinitearray,frequency)
harray = design(infinitearray,frequency,elements)

ha = design(planewaveexcitation,frequency)
ha = design(planewaveexcitation,frequency,direction)
```

Description

`hant = design(antenna,frequency)` designs an `antenna` object from the antenna library that resonates at the specified `frequency`.

`harray = design(array,frequency)` designs an `array` of dipoles that operates at the specified frequency. The spacing between elements is half a wavelength.

`harray = design(array,frequency,elements)` designs an array of elements for operation at the specified frequency. The spacing between elements is half a wavelength, if possible. If the object cannot achieve half-wavelength spacing, it uses the element size to calculate the separation between elements and evenly distributes the elements on a sphere with a radius proportional to the largest element in `element`.

`harray = design(conformalarray,frequency)` designs a conformal array of dipole and bowtie elements at the specified frequency. The object places the elements in the locations specified by the default `conformalArray` object. If the object cannot place elements at the specified positions due to the intersection of elements, it uses the element size to calculate the spacing between elements and distributes the elements on a sphere with a radius proportional to the largest element in the `Elements`property of the `conformalArray` object.

`harray = design(conformalarray,frequency,elements)` designs a conformal array of the specified elements at the specified frequency.

`harray = design(infinitearray,frequency)` designs an infinite array with a reflector element at the specified frequency.

`harray = design(infinitearray,frequency,elements)` designs an infinite array of the specified elements at the specified frequency.

Note

- Antennas or arrays that you design with the `design` function resonate around the design frequency with an error tolerance of 10–15%. To reduce this tolerance and optimize your antenna or array design, use the `optimize` function.
- The `desgin` function uses air as its default substrate.

`ha = design(planewaveexcitation,frequency)` creates a plane-wave excitation environment and calculates the required orientation and polarization of the receiver antenna element to capture the maximum power from the incident plane-wave at the specified frequency.

`ha = design(planewaveexcitation,frequency,direction)` creates a plane-wave excitation environment with the receiver antenna element orientation `direction` and calculates the required polarization of this antenna to capture the maximum power from the incident plane-wave at the specified frequency.

Examples

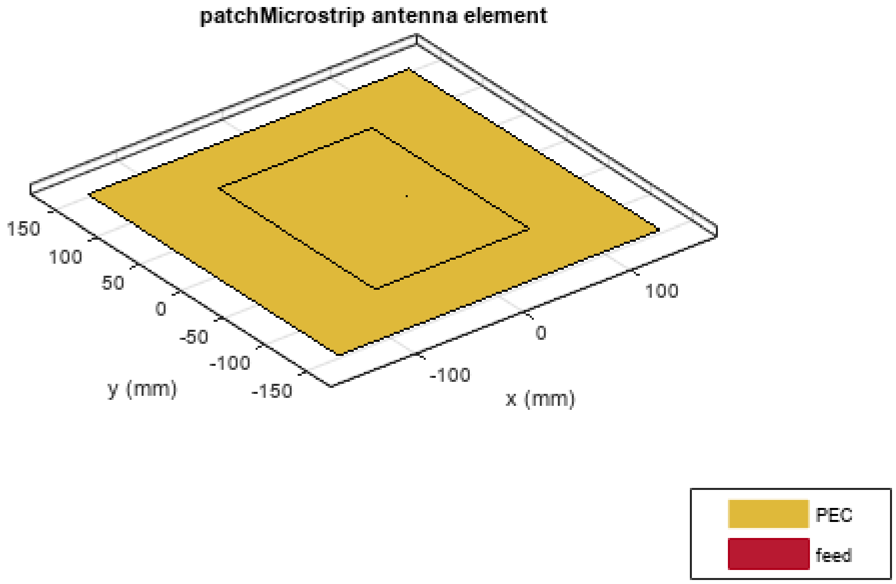
Prototype Antenna Design

Design a prototype microstrip patch antenna that resonates at a frequency of 1 GHz.

```
p = design(patchMicrostrip,1e9)
```

```
p = 
  patchMicrostrip with properties:

               Length: 0.1439
                Width: 0.1874
               Height: 0.0030
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.2998
     GroundPlaneWidth: 0.2998
    PatchCenterOffset: [0 0]
           FeedOffset: [0.0303 0]
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(p)
```



Calculate the impedance of the above antenna at the same frequency.

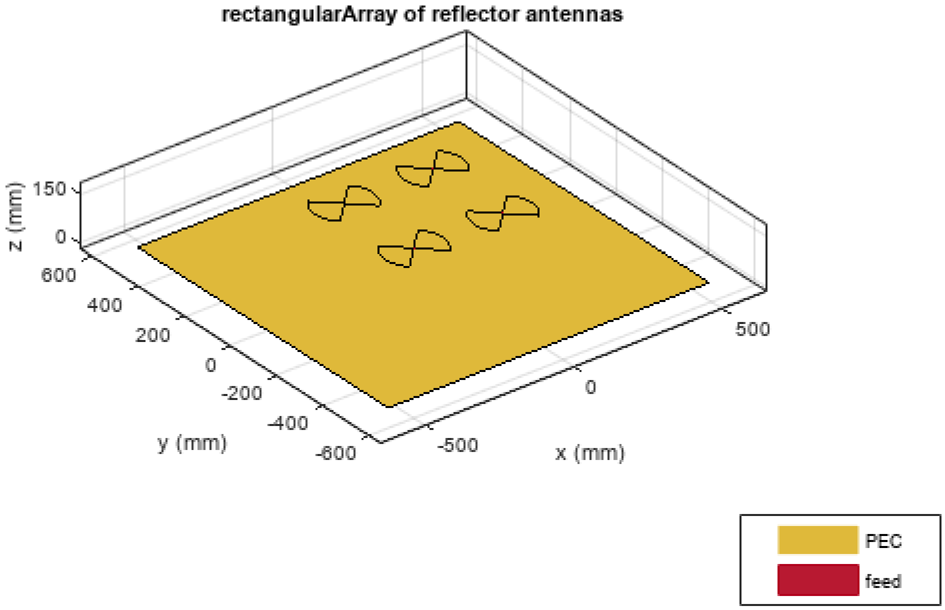
```
Z = impedance(p,1e9)
```

```
Z = 40.3562 - 12.3908i
```

Rectangular Array of Reflector Backed Rounded Bowtie Antennas

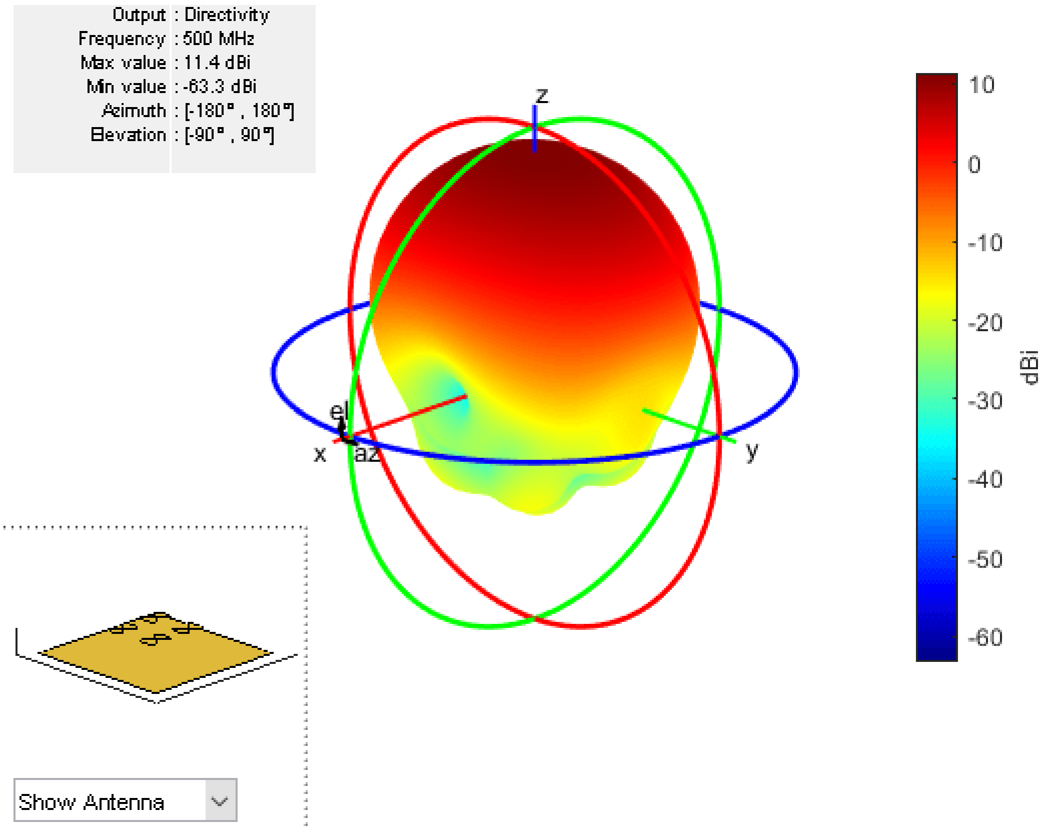
Design a rectangular array of reflector backed rounded bowtie antennas to operate at 500 MHz.

```
b = bowtieRounded('Tilt',90,'TiltAxis',[0 1 0]);
r = reflector('Exciter',b);
ra = design(rectangularArray,500e6,r);
show(ra)
```



Plot the radiation pattern of the rectangular array at 500 MHz.

```
pattern(ra,500e6)
```



```
Output : Directivity
Frequency : 500 MHz
Max value : 11.4 dBi
Min value : -63.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Design Conformal Array of Four Elements

Create a default conformal array.

```
confarraydef = conformalArray
```

```
confarraydef = 
  conformalArray with properties:

            Element: {[1x1 dipole]  [1x1 bowtieTriangular]}
    ElementPosition: [2x3 double]
          Reference: 'feed'
     AmplitudeTaper: 1
         PhaseShift: 0
               Tilt: 0
           TiltAxis: [1 0 0]
```

Design a conformal array using a dipole antenna, folded dipole antenna, meander dipole antenna, and a monopole antenna at 1 GHz.

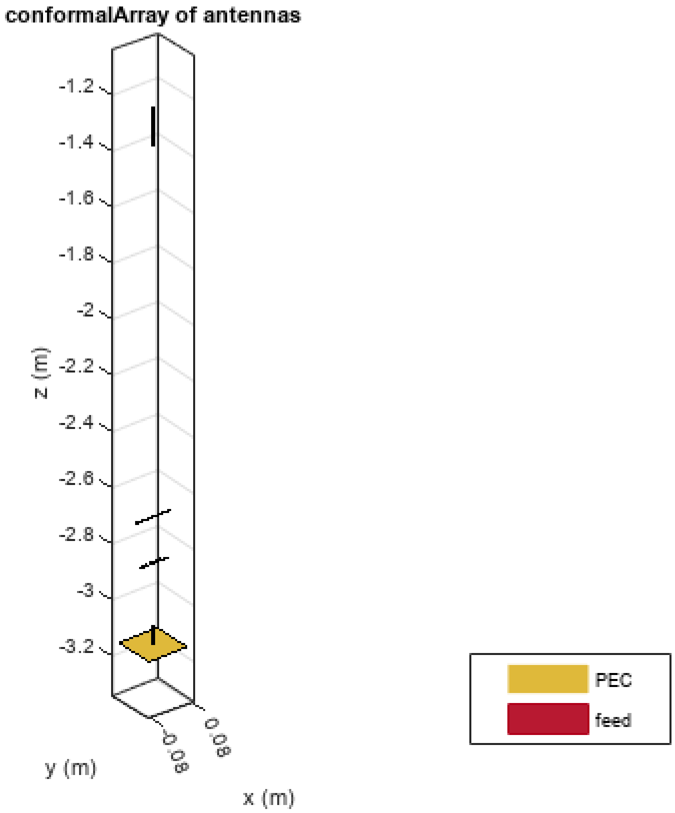
```
desC = design(confarraydef,1e9,{dipole, dipoleFolded, dipoleMeander, monopole})
```

```
desC = 
  conformalArray with properties:
          Element: {1x4 cell}
  ElementPosition: [4x3 double]
        Reference: 'feed'
    AmplitudeTaper: 1
      PhaseShift: 0
            Tilt: 0
        TiltAxis: [1 0 0]
```

```
desC.ElementPosition
```

```
ans = 4×3

         0         0   -1.3016
         0         0   -2.6939
         0         0   -2.8594
         0         0   -3.1498
```

```
show(desC)
```



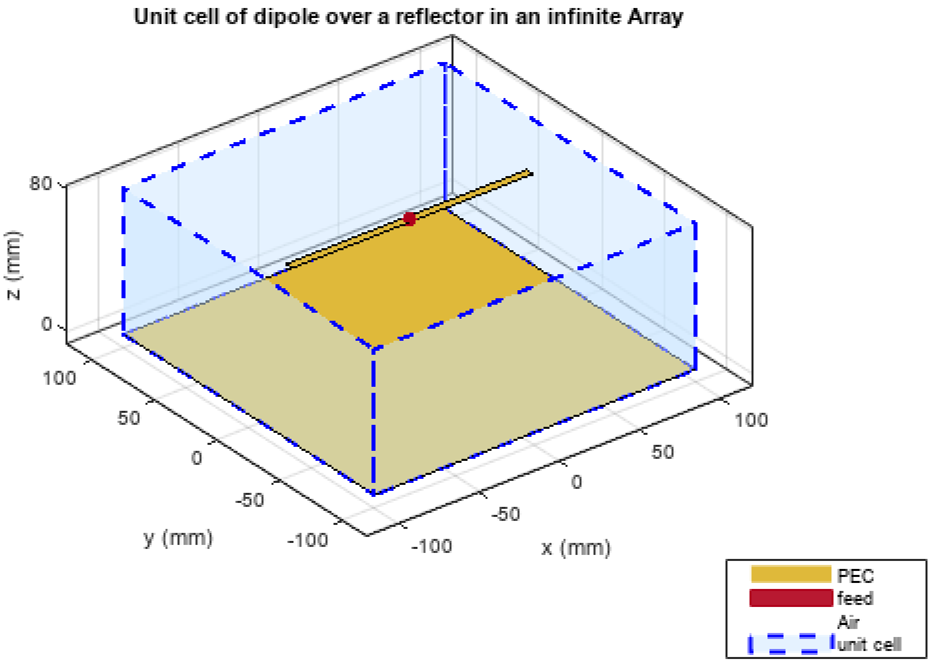
Design Infinite Array Using Specified Frequency and Antenna

Create an infinite array.

```
infarrayV1 = infiniteArray
```

```
infarrayV1 = 
  infiniteArray with properties:

          Element: [1x1 reflector]
      ScanAzimuth: 0
    ScanElevation: 90
     RemoveGround: 0
```

```
show(infarrayV1)
```

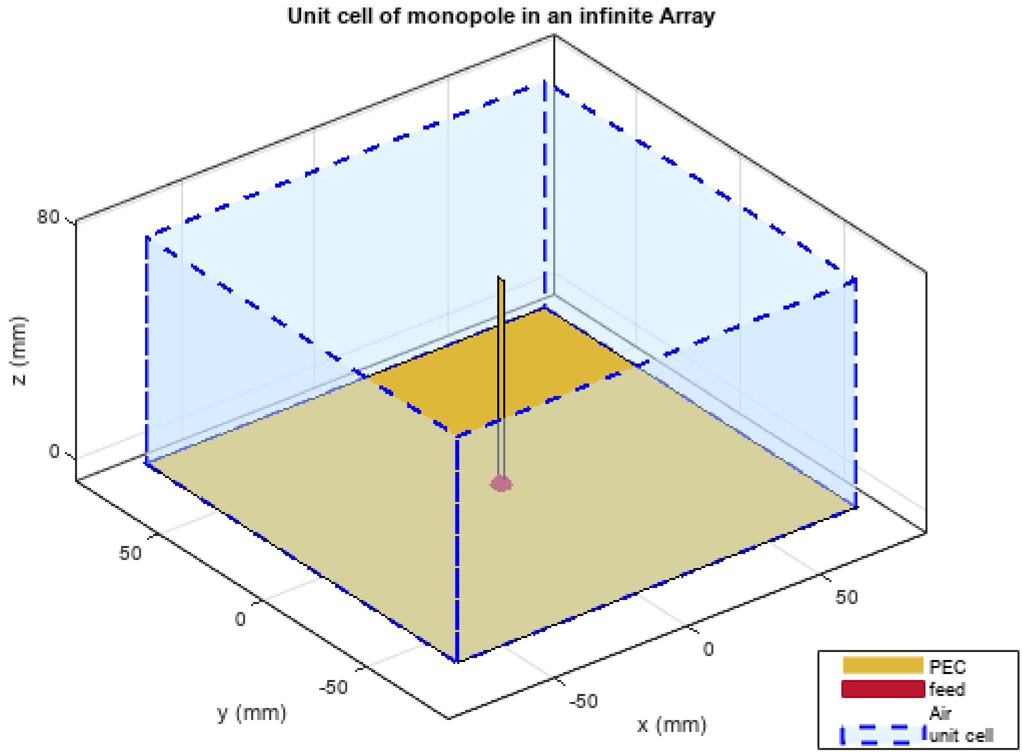


Design the above array using a monopole antenna and at 1 GHz frequency.

```
infarrayV2 = design(infarrayV1,1e9,monopole)
```

```
infarrayV2 = 
  infiniteArray with properties:

          Element: [1x1 monopole]
      ScanAzimuth: 0
    ScanElevation: 90
     RemoveGround: 0
```

```
show(infarrayV2)
```



Input Arguments

antenna — Antenna object

scalar

Antenna object from antenna library, specified as a scalar.

Example: `dipole`

array — Array object

`linearArray` | `rectangularArray` | `circularArray`

Array object from antenna library, specified as a `linearArray`, `rectangularArray`, or `circularArray` object.

Example: `r = reflector;ra = design(rectangularArray,500e6,r);` Designs a rectangular array of reflectors operating at a frequency of 500 MHz.

conformalarray — Conformal array object

`conformalArray` object

Conformal array object, specified as a `conformalArray` object.

You can position elements in a conformal array in three ways:

- Case 1: Points lie on a line.
- Case 2: Points lie on a plane.
- Case 3: Points lie in 3-D space.

Example: `c = conformalArray;ca = design(c,50e6,{dipole,dipoleFolded, dipoleJ, bowtieTriangular,dipole,dipole,dipole,dipole,dipole});` Designs a conformal array of specified elements operating at a frequency of 50 MHz.

`infinitearray` — Infinite array object
scalar

Infinite array, specified as an `infiniteArray` object.

Example: `i = infiniteArray;ia = design(i,1e9,monopole);` Designs an infinite array with a monopole antenna element operating at a frequency of 1 GHz.

`planewaveexcitation` — Plane-wave excitation environment
`planeWaveExcitation` object

Plane-wave excitation environment, specified as a `planeWaveExcitation` object.

Example: `design(planeWaveExcitation,1e9);` Creates a plane-wave excitation environment and calculates the orientation and polarization of the receiver antenna required to capture the maximum power from the incident plane-wave at 1 GHz.

`frequency` — Resonant frequency of antenna
real positive scalar

Resonant frequency of the antenna, specified as a real positive scalar.

Example: `55e6`

Data Types: `double`

`elements` — Antenna object in array
single antenna element | cell array

Antenna object from the antenna library used in the array, specified as a single antenna element or a cell array in conformal array. For more information on element positions for conformal array, see `conformalarray`.

Example: `r = reflector;ra = design(rectangularArray,500e6,r);` Designs a rectangular array of reflectors operating at a frequency of 500 MHz.

Example: `c = conformalArray;ca = design(c,50e6,{dipole,dipoleFolded, dipoleJ, bowtieTriangular,dipole,dipole,dipole,dipole,dipole});` Designs a conformal array of specified elements operating at a frequency of 50 MHz.

`direction` — Orientation of receiver antenna element
1-by-3 vector of Cartesian coordinates | 1-by-2 vector of azimuth and elevation angles

Orientation of the receiver antenna element, specified as a 1-by-3 vector of Cartesian coordinates or a 1-by-2 vector of azimuth and elevation angles. When you specify the Cartesian coordinates of a point, the function calculates the direction by joining a line from the origin to this point.

Example: `design(planeWaveExcitation,1e9,[45 45])`

Output Arguments

`hant` — Antenna object operating at specified reference frequency
antenna object

Antenna object operating at the specified reference frequency, returned as an antenna object.

`harray` — Array object operating at specified reference frequency and specified elements
array object

Array object operating at the specified reference frequency and specified elements, returned as an array object.

ha — Plane-wave excitation environment
`planeWaveExcitation` object

Plane-wave excitation environment, returned as a `planeWaveExcitation` object. This output contains the receiver antenna orientation and polarization required to capture the maximum power from the incident plane-wave.

Tips

- Use the `mesh` function to refine your antenna or array prototypes.

Version History

Introduced in R2016b

See Also

`show` | `impedance`

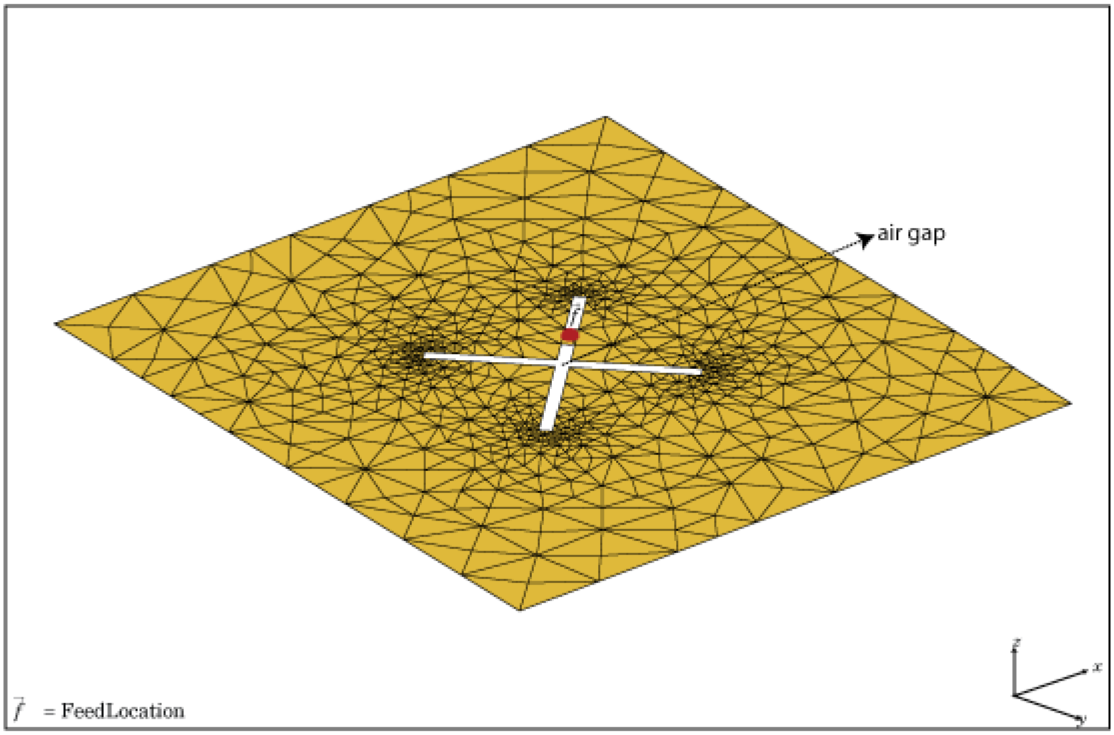
createFeed

Create feed location for custom antenna

Syntax

```
createFeed(antenna)
createFeed(antenna,point1,point2)
```

Description



`createFeed(antenna)` plots a custom antenna mesh in a figure window. From the figure window, you can specify a feed location for the mesh and create a custom antenna. To specify a region for the feed point, select two points, inside triangles on either side of the air gap or inside triangles that share a common edge.

`createFeed(antenna,point1,point2)` creates the feed across the triangle edges identified by `point1` and `point2`. After the feed is created, when you plot the resulting antenna mesh the feed location is highlighted.

Input Arguments

antenna — Custom antenna mesh
scalar

Custom mesh antenna, specified as a scalar.

point1,point2 — Points to identify feed region

Cartesian coordinates in meters

Points to identify feed region, specified as Cartesian coordinates in meters. Specify the points in the format [x_1, y_1], [x_2, y_2].

Example: `createFeed(c,[0.07,0.01],[0.05,0.05]);`

Examples

Create Feed for Custom Mesh Antenna Using Air Gap between Triangles

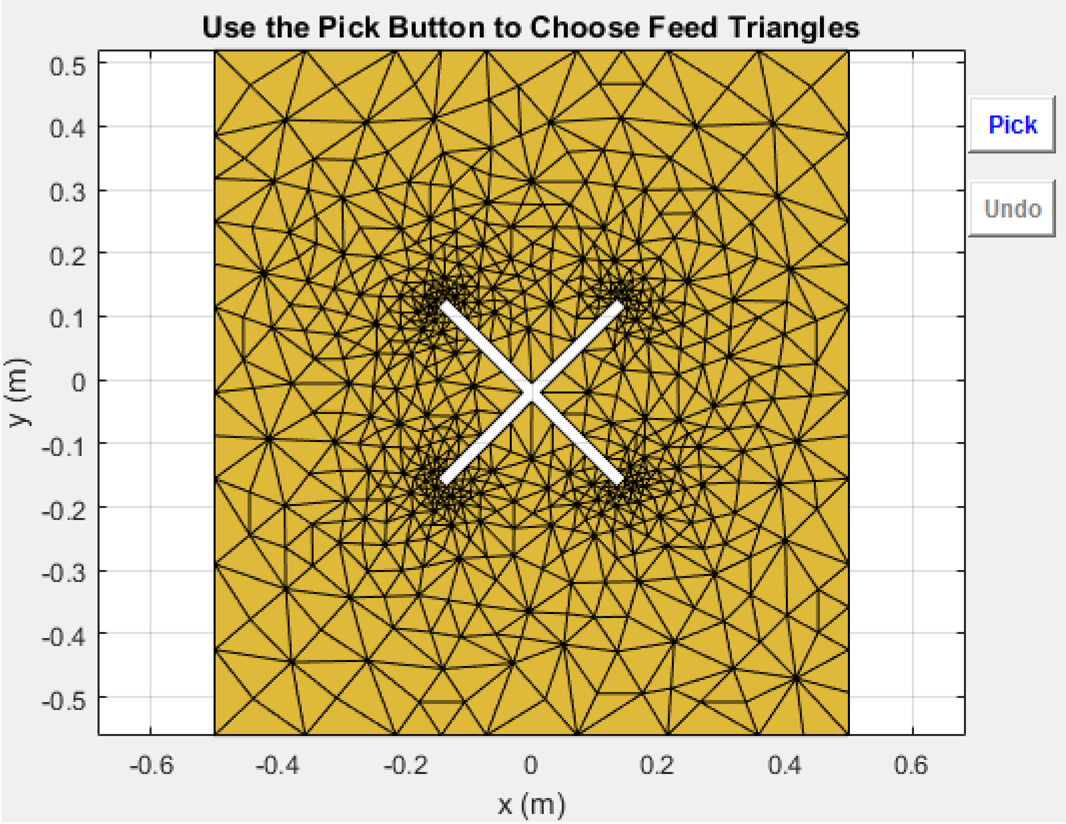
Load a 2-D custom mesh. Create a custom antenna using the points and triangles.

```
load planarmesh.mat
c = customAntennaMesh(p,t)
```

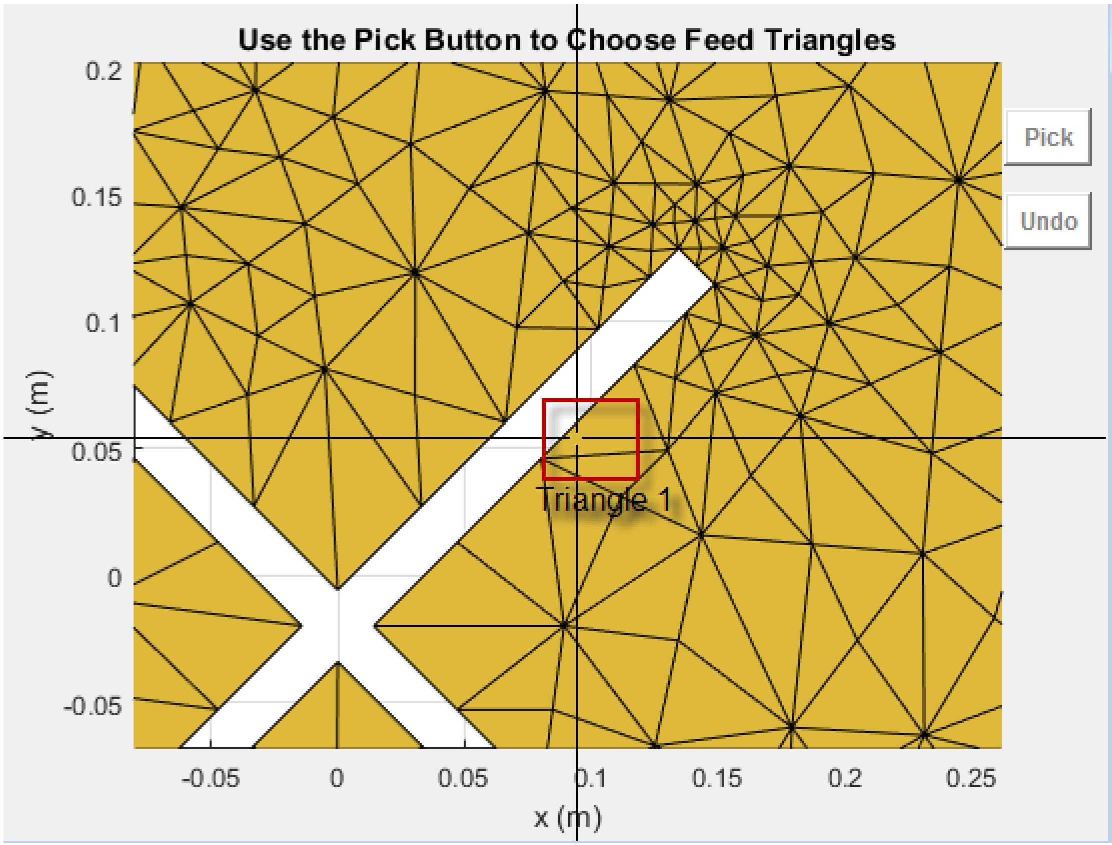
```
c = 

  customAntennaMesh with properties:

          Points: [3x658 double]
       Triangles: [4x1219 double]
    FeedLocation: []
            Tilt: 0
        TiltAxis: [1 0 0]
```

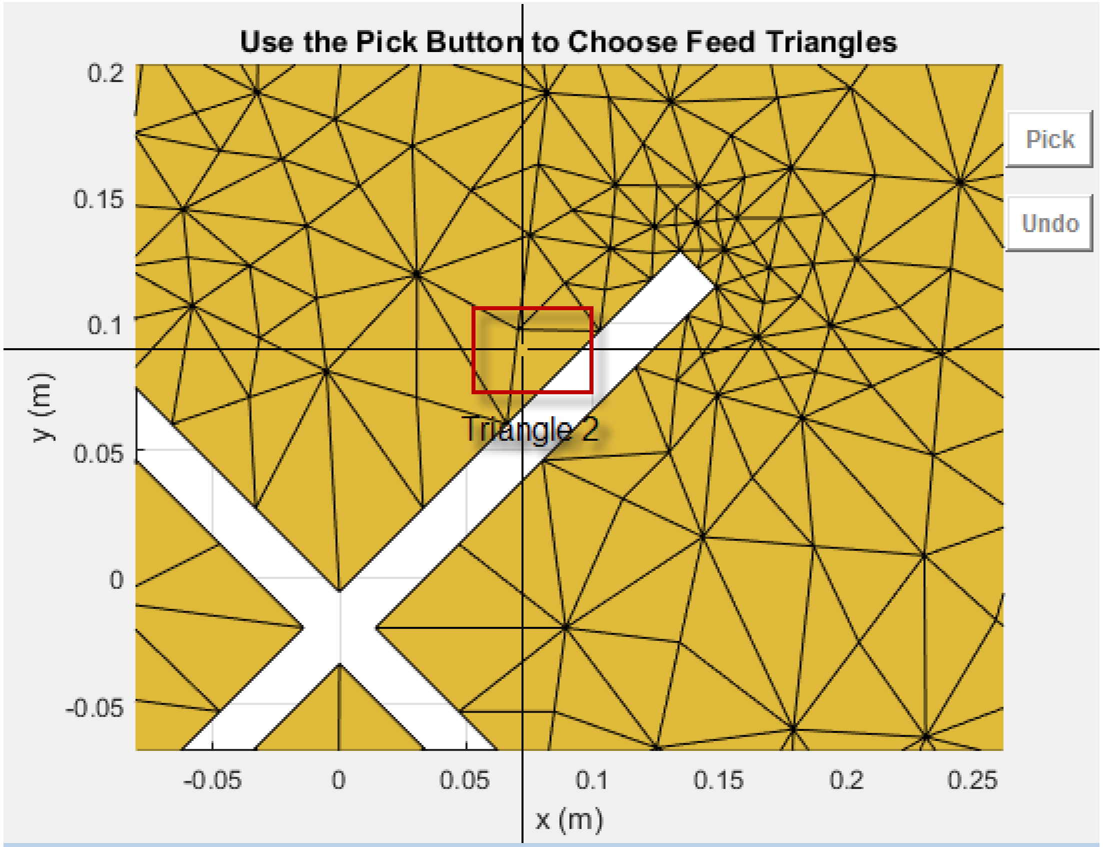
Use the `createFeed` function to view the antenna mesh structure. In this antenna mesh view, you see **Pick** and **Undo** buttons. The **Pick** button is highlighted.

```
createFeed(c)
```

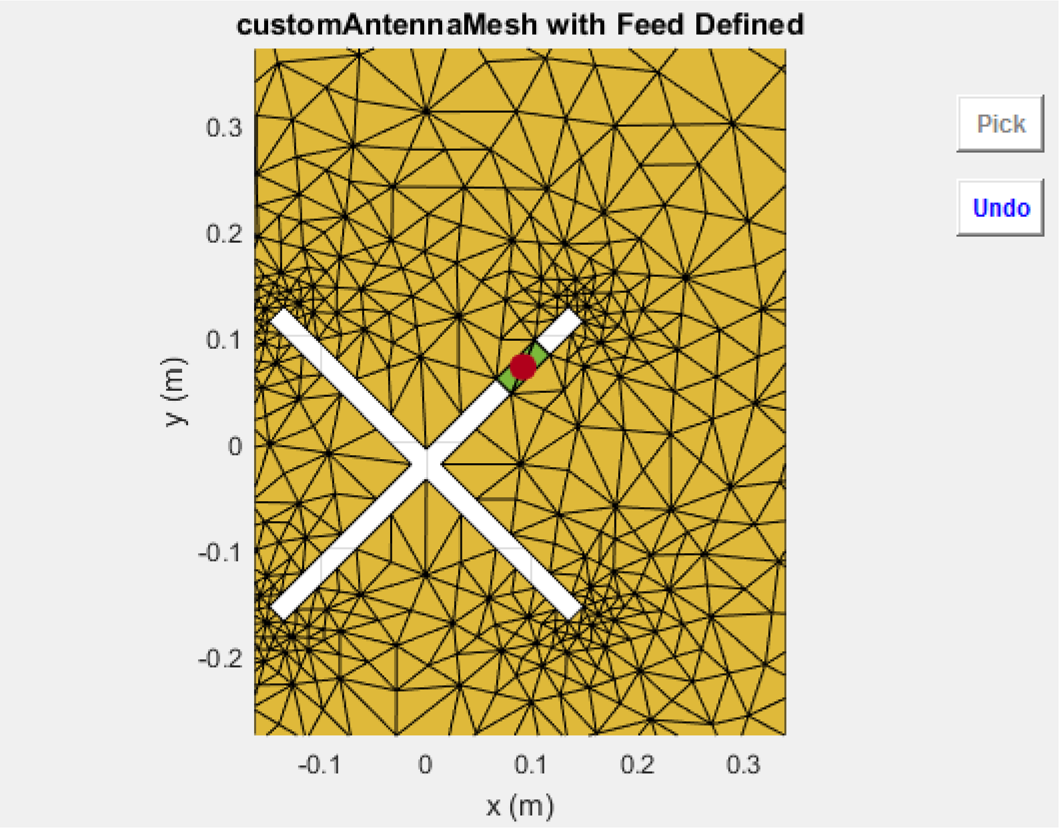


Click **Pick** to display the cross-hairs. To specify a region for the feed point, zoom in and select two points, one inside each triangle on either side of the air gap. Select the points using the cross-hairs.


Use the Pick Button to Choose Feed Triangles
Pick
Undo
Triangle 1
0.2
0.15
0.1
0.05
0
-0.05
y (m)
-0.05
0
0.05
0.1
0.15
0.2
0.25
x (m)



Selecting the second triangle creates and displays the antenna feed.



Create Feed for Custom Mesh Antenna Using Triangles Sharing Edge

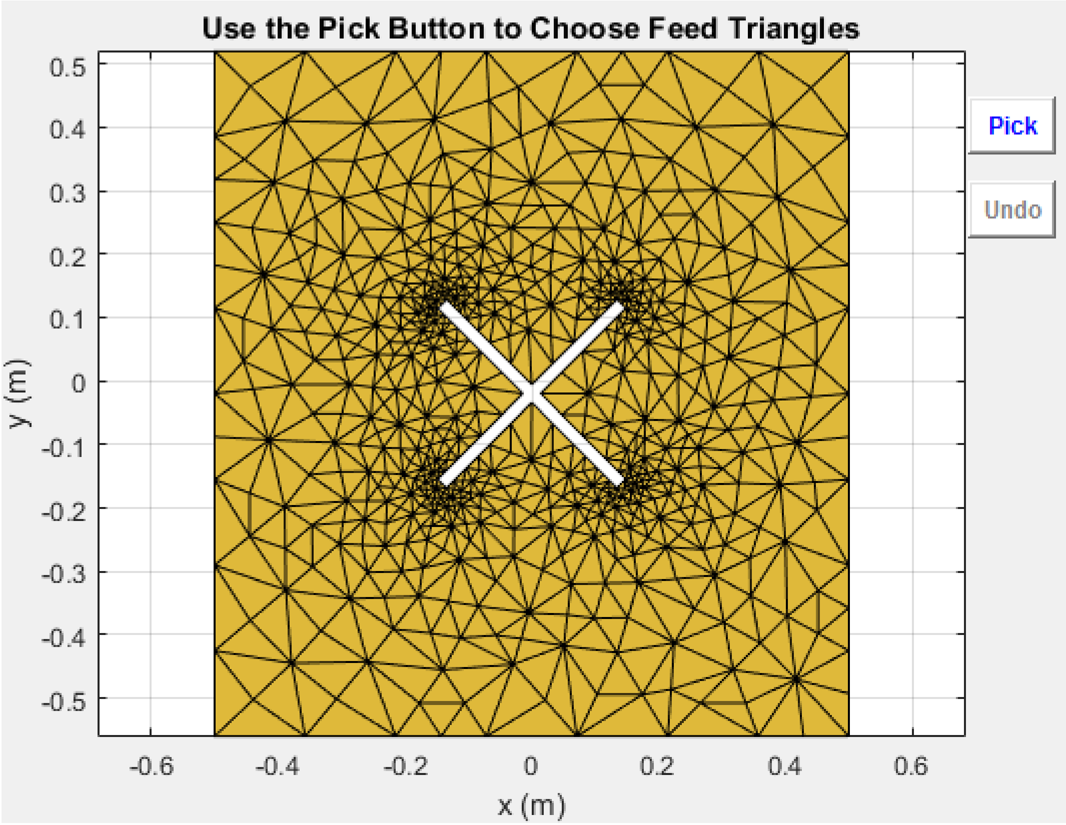
Load a 2-D custom mesh. Create a custom antenna using the points and triangles.

```
load planarmesh.mat
c = customAntennaMesh(p,t)
```

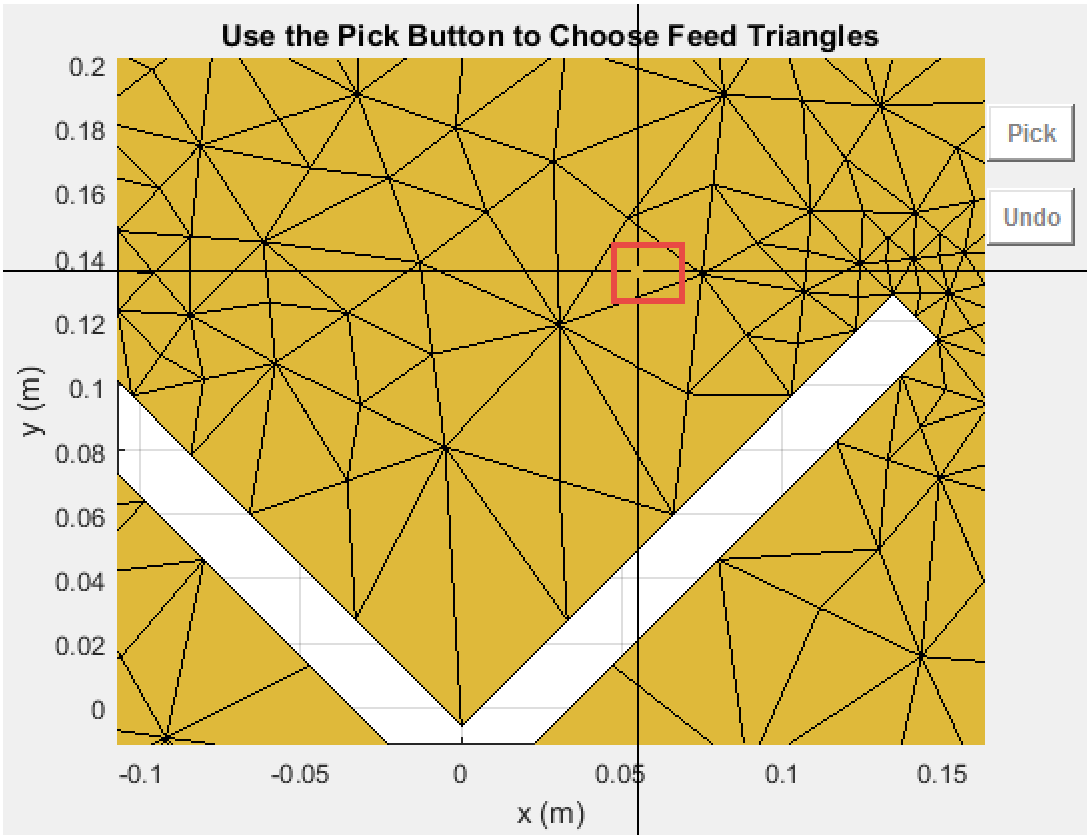
```
c = 

  customAntennaMesh with properties:

         Points: [3x658 double]
      Triangles: [4x1219 double]
   FeedLocation: []
           Tilt: 0
      TiltAxis: [1 0 0]
```

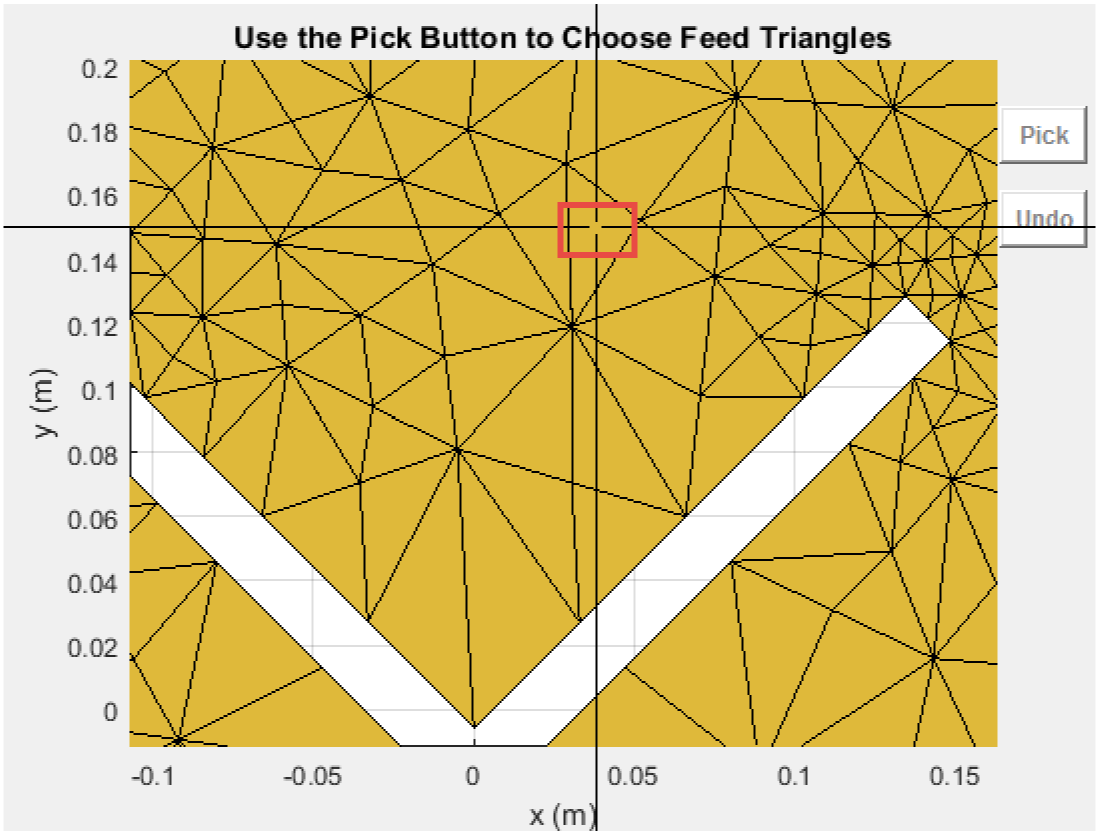
Use the `createFeed` function to view the antenna mesh structure. In this antenna mesh view, you see **Pick** and **Undo** buttons. The **Pick** button is highlighted.

```
createFeed(c)
```

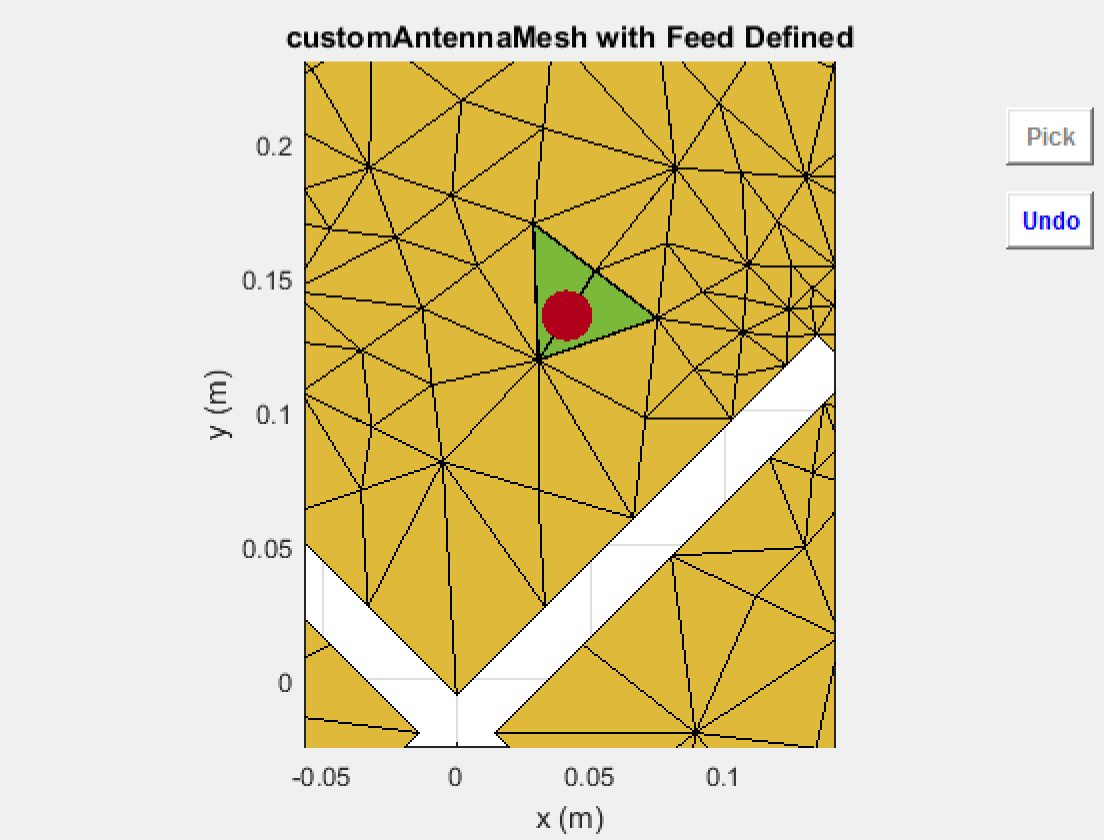


Click **Pick** to display the cross-hairs. To specify a region for the feed point, zoom in and select two points, one inside each triangle sharing an edge. Select the points using the cross-hairs.


Use the Pick Button to Choose Feed Triangles
Pick
Undo
y (m)
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
x (m)
-0.1
-0.05
0
0.05
0.1
0.15



Selecting the second triangle creates and displays the antenna feed.



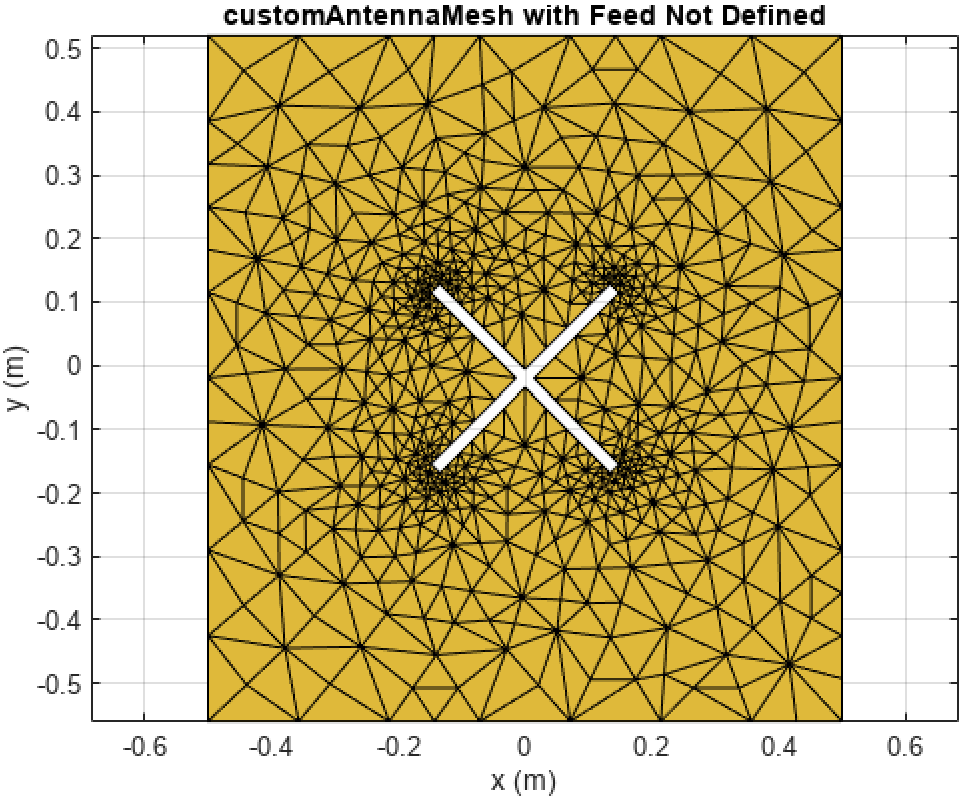
Create Feed for Custom Antenna Mesh

Load a 2-D custom mesh using the planarmesh.mat. Create a custom antenna using the points and triangles.

```
load planarmesh.mat
c = customAntennaMesh(p,t)
```

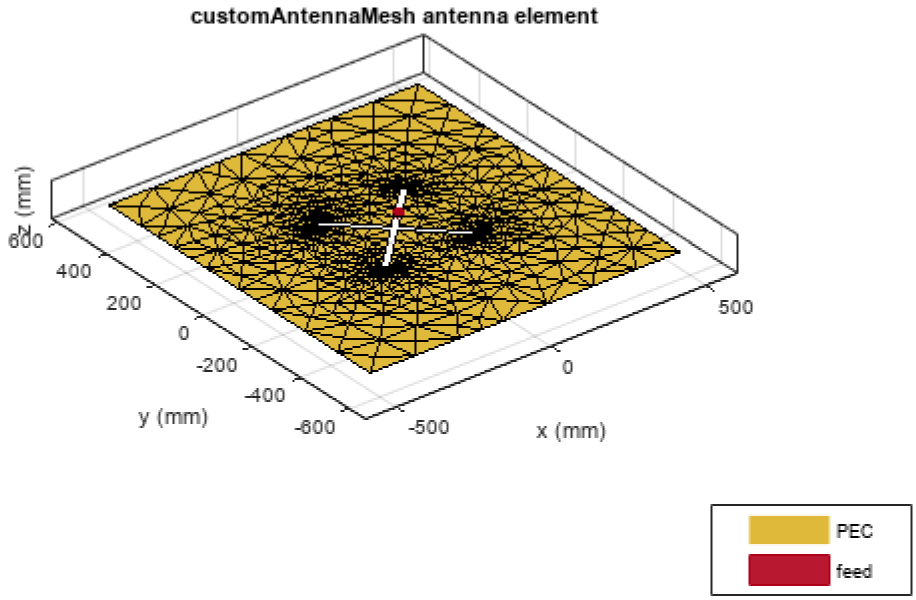
```
c = 
  customAntennaMesh with properties:

          Points: [3x658 double]
       Triangles: [4x1219 double]
    FeedLocation: []
       Conductor: [1x1 metal]
            Tilt: 0
        TiltAxis: [1 0 0]
            Load: [1x1 lumpedElement]
```

```
show (c)
```



Create the feed for the custom antenna across the points (0.07,0.01) and (0.05,0.05) meters respectively.

```
createFeed(c,[0.07,0.01],[0.05,0.05])
show(c)
```



Version History

Introduced in R2015b

See Also

`returnLoss` | `sparameters`

EHfields

Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays

Syntax

```
[e,h] = EHfields(object,frequency)
EHfields(object,frequency)
```

```
[e,h] = EHfields(object,frequency,points)
EHfields(object, frequency, points)
```

```
EHfields( ___ ,Name,Value)
```

Description

`[e,h] = EHfields(object,frequency)` calculates the *x*, *y*, and *z* components of the electric and magnetic fields of an antenna or array object at a specified frequency. The fields are calculated at points on the surface of a sphere whose radius is twice that of the radius of a sphere completely enclosing the antenna or array structure.

`EHfields(object,frequency)` plots the absolute values of the electric and magnetic field vectors along with corresponding signed complex angles at the specified frequency. The multiplication factor with absolute field value is +1 for positive and -1 for negative complex angle. The fields are calculated at points on the surface of a sphere whose radius is twice that of the radius of a sphere completely enclosing the antenna or array structure.

`[e,h] = EHfields(object,frequency,points)` calculates the *x*, *y*, and *z* components of electric field and magnetic field of an antenna or array object. These fields are calculated at specified points in the space and at a specified frequency. Specify the points in the Cartesian coordinate system.

`EHfields(object, frequency, points)` plots the absolute values of the electric and magnetic field vectors along with corresponding signed complex angles at specified frequency values and at specified points in space. The multiplication factor with absolute field value is +1 for positive and -1 for negative complex angle. Specify the points in the Cartesian coordinate system.

`EHfields( ___ ,Name,Value)` plots the electric and magnetic field vectors using one or more name-value arguments in addition to any of the input argument combinations in previous syntaxes. For example, `'ViewField','E'` displays only the electric field.

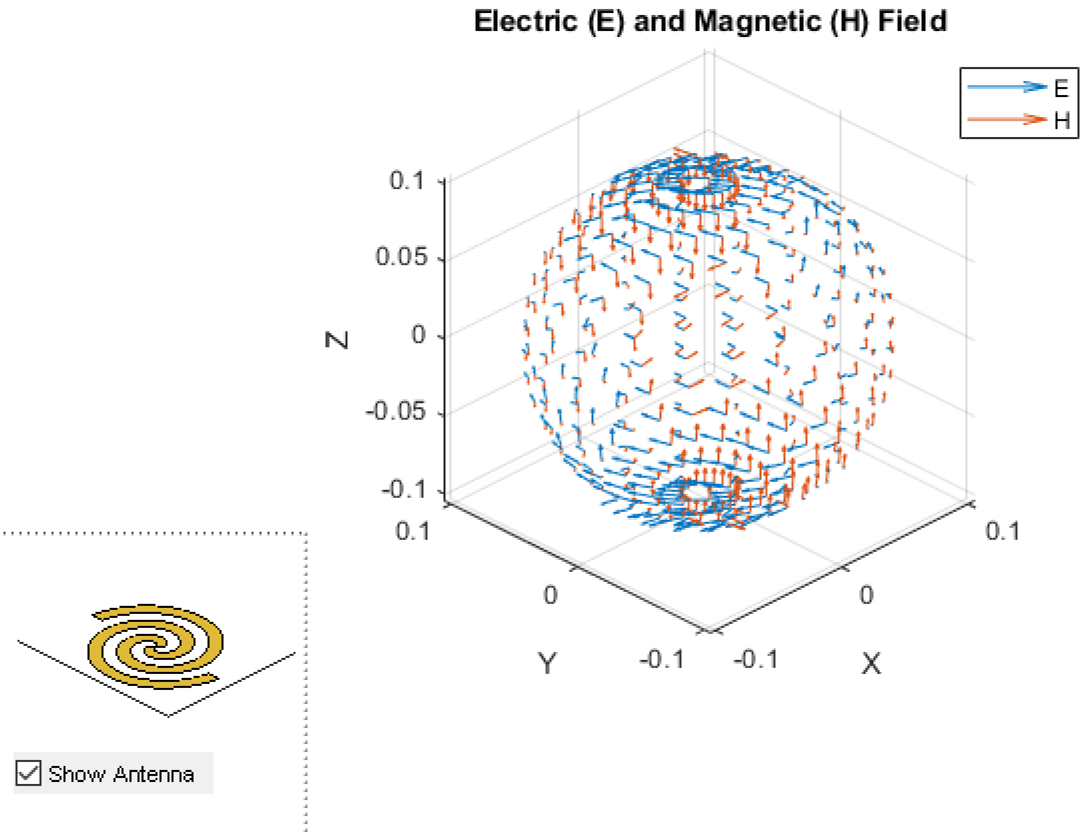
Use the `'ElementNumber'` and `'Termination'` properties to calculate the embedded electric and magnetic fields of the antenna element in an array connected to a voltage source. The voltage source model consists of an ideal voltage source of 1 volt. The embedded pattern includes the effect of mutual coupling due to the other antenna elements in the array.

Examples

Plot E and H Fields of Antenna

Plot electric and magnetic fields of a default Archimedean spiral antenna.

```
h = spiralArchimedean;
EHfields(h,4e9)
```



Calculate EH Fields of Antenna

Calculate electric and magnetic fields at a point 1m along the z-axis from an Archimedean spiral antenna.

```
h = spiralArchimedean;
[e,h] = EHfields(h,4e9,[0;0;1])
```

```
e = 3×1 complex

  -0.4145 - 0.2545i
  -0.3036 + 0.4095i
   0.0000 - 0.0000i


h = 3×1 complex

   0.0008 - 0.0011i
  -0.0011 - 0.0007i
  -0.0000 + 0.0000i
```

Calculate EH Fields in Spherical Coordinates

Calculate electric and magnetic fields at a point 1 m along the z-axis from an Archimedean spiral antenna. Return the field as azimuth, elevation, and radius values in a spherical coordinate system.

```
h = spiralArchimedean;
[e,h] = EHfields(h,4e9,[0;0;1],'CoordinateSystem','spherical')
```

```
e = 3×1 complex

  -0.3036 + 0.4095i
   0.4145 + 0.2545i
   0.0000 - 0.0000i


h = 3×1 complex

  -0.0011 - 0.0007i
  -0.0008 + 0.0011i
  -0.0000 + 0.0000i
```

Plot Electric and Magnetic Field Vector of Antenna

Create an Archimedean spiral antenna. Plot electric and magnetic field vector at the z = 1cm plane from the antenna.

```
h = spiralArchimedean;
```

Define points on a rectangular grid in the X-Y plane.

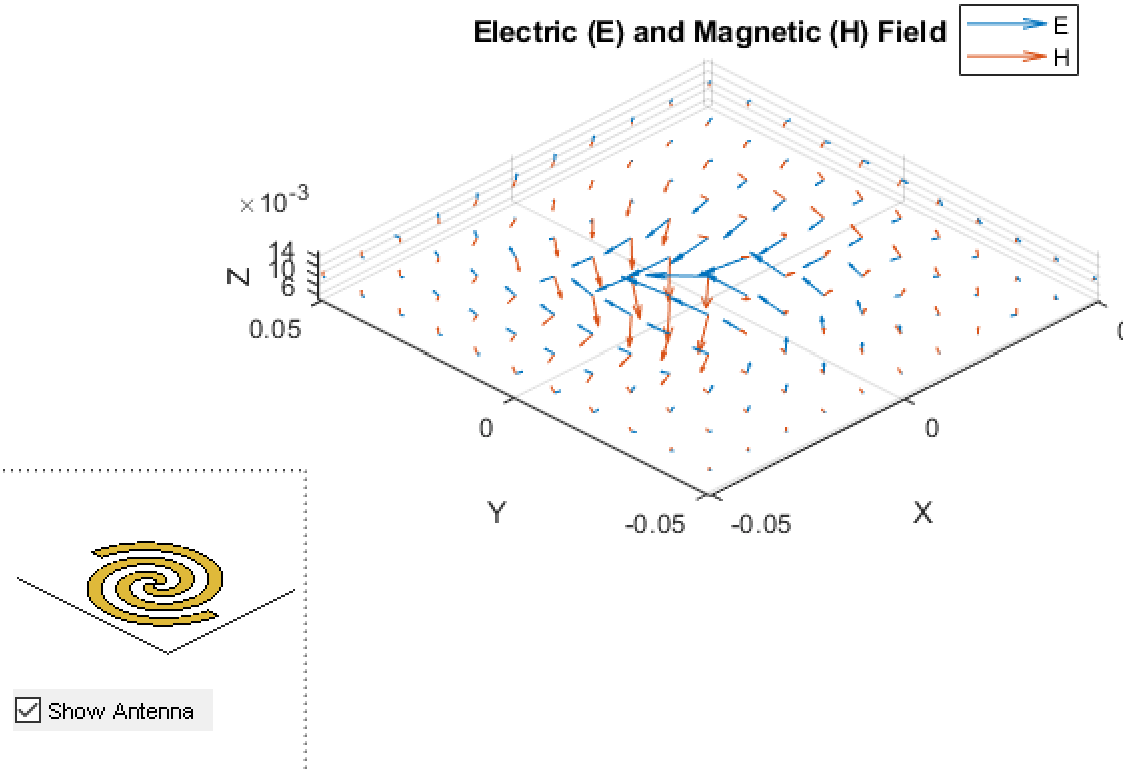
```
[X,Y] = meshgrid(-.05:.01:.05,-.05:.01:.05);
```

Add a z-offset of 0.01.

```
p = [X(:)';Y(:)';.01*ones(1,prod(size(X)))];
```

Plot electric and magnetic field vector at the z = 1cm plane. from the antenna

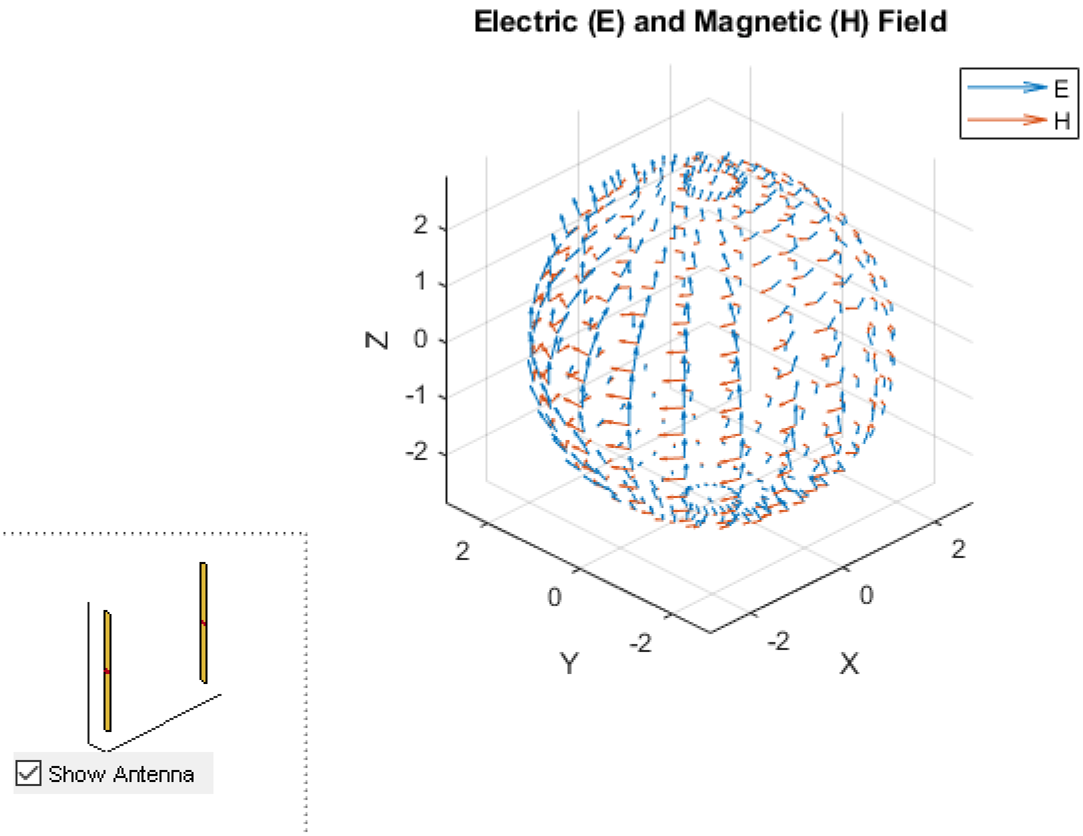
```
EHfields (h,4e9,p)
```



Embedded Vector Fields of Linear Array

Plot the embedded vector fields of a linear array when the first element is excited and all the other antenna elements are terminated using 50-ohm resistance.

```
l = linearArray;
EHfields(l, 70e6, 'ElementNumber', 1, 'Termination', 50);
```



Calculate the Electric and Magnetic Fields at Multiple Frequencies at Single Point in Space

Calculate electric and magnetic fields of a dipole antenna.

```
[E H] = EHfields(dipole, 70e6:2e6:80e6, [0 0 0]')
```

```
E = 3×6 complex

   0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i
   0.0002 + 0.0033i   0.0008 + 0.0030i   0.0011 + 0.0025i   0.0012 + 0.0022i   0.0013 + 0.0019i
  -1.5534 - 1.9685i  -1.8714 - 1.6824i  -2.0190 - 1.3839i  -2.0605 - 1.1326i  -2.0479 - 0.9370i


H = 3×6 complex
10^-3 ×

  -0.6007 + 0.0834i  -0.5184 + 0.1850i  -0.4301 + 0.2353i  -0.3547 + 0.2531i  -0.2955 + 0.2539i
   0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

Calculate Electric and Magnetic Field of Helix Antenna

Calculate the electric and magnetic fields of a helix antenna with a vertically polarized electric field.

```
EHfields(helix,1e9,'Polarization','V')
```

```
ans = 1×441 complex

   0.0251 - 0.0286i  -0.0313 + 0.0450i  -0.0387 + 0.0590i  -0.0482 + 0.0704i  -0.0601 + 0.0786i
```

Input Arguments

`object` — Antenna or array object
object

Antenna or array object, specified as an object.

Example: `h = spiralArchimedean`

`frequency` — Frequency used to calculate electric and magnetic fields
scalar | vector in Hz

Frequency used to calculate electric and magnetic fields, specified as a scalar or a vector in Hz.

Example: 70e6

Data Types: `double`

`points` — Cartesian coordinates of points in space
3-by-*p* complex matrix

Cartesian coordinates of points in space, specified as a 3-by-*p* complex matrix. *p* is the number of points at which to calculate the `E-H` field.

Example: `[0;0;1]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` 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: `'ScaleFields',[2 0.5]` specifies scalar values of the electric and magnetic fields

`ScaleFields` — Value by which to scale electric and magnetic fields
two-element vector

Value by which to scale the electric and magnetic fields, specified as the comma-separated pair consisting of `'ScaleFields'` and a two-element vector. The first element scales the E field and the second element scales the H-field. A value of `2` doubles the relative length of either field. A value of `0.5` to halves the length of either field. A value of `0` plots either field without automatic scaling.

Example: `'ScaleFields',[2 0.5]`

Data Types: `double`

`ViewField` — Field to display
`'E'` | `'H'`

Field to display, specified as the comma-separated pair consisting of `'ViewField'` and `'E'` or `'H'`. `'E'` displays the electric field and `'H'` displays the magnetic field.

Example: `'ViewField','E'`

Data Types: `char`

`ElementNumber` — Antenna element in array
scalar

Antenna element in array, specified as the comma-separated pair consisting of `'ElementNumber'` and scalar This antenna element is connected to the voltage source.

Note Use this property to calculate the embedded pattern of an array.

Example: `'ElementNumber',1`

Data Types: `double`

`Termination` — Impedance value for array element termination
50 (default) | scalar

Impedance value for array element termination, specified as the comma-separated pair consisting of `'Termination'` and scalar. The impedance value terminates other antenna elements of an array while calculating the embedded pattern of the antenna connected to the voltage source.

Note Use this property to calculate the embedded pattern of an array.

Example: `'Termination',40`

Data Types: `double`

`CoordinateSystem` — Coordinate system
`'rectangular'` (default) | `'spherical'`

Coordinate system to calculate and plot the electric and magnetic fields of an antenna or array, specified as the comma-separated pair consisting of `'CoordinateSystem'` and `'rectangular'` or `'spherical'`.

When you specify both the `'CoordinateSystem'` and the `'Polarization'` name-value arguments, the `EHfields` function ignores the value of `'CoordinateSystem'`. The output is the same in both coordinate systems.

Example: `'CoordinateSystem','spherical'`

Data Types: `char`

`Polarization` — Polarization of electric field
`'combined'` (default) | `'H'` | `'V'`

Polarization of the electric field, specified as the comma-separated pair consisting of `'Polarization'` and `'H'` for horizontal polarization and `'V'` for vertical polarization. The default value is set to `'combined'`. The magnetic field has the opposite polarization.

When you specify `'Polarization'` in a syntax without output arguments, the `EHfields` function returns the component of the electric field with polarization as a 1-by-*p* vector in the variable `ans`. The function does not plot the values when the polarization is set to either `'H'` or `'V'`.

Example: `'Polarization','H'`

Data Types: `char`

Output Arguments

e — Components of electrical field
3-by-*p* complex matrix in V/m

x, *y*, *z* components of electrical field in the rectangular coordinate system or `azimuth`, `elevation`, `radial` components in the spherical coordinate system, returned as 3-by-*p* complex matrix in V/m. The dimension *p* is the number of points in space at which the electric and magnetic fields are computed.

h — *x*, *y*, *z* components of magnetic field
3-by-*p* complex matrix in A/m

x, *y*, *z* components of magnetic field in the rectangular coordinate system or `azimuth`, `elevation`, `radial` components in the spherical coordinate system, returned as a 3-by-*p* complex matrix in A/m. The dimension *p* is the number of points in space at which the electric and magnetic fields are computed.

Version History

Introduced in R2015a

See Also

`beamwidth` | `axialRatio`

Topics

"Antenna Toolbox Coordinate System"

axialRatio

Axial ratio of antenna

Syntax

```
axialRatio(antenna,frequency,azimuth,elevation)
ar = axialRatio(antenna,frequency,azimuth,elevation)
```

Description

`axialRatio(antenna,frequency,azimuth,elevation)` plots axial ratio of an antenna over a specified frequency, and in the direction specified by `azimuth` and `elevation`. Any one among frequency, azimuth, or elevation values must be scalar. If only one of the values are scalar, the plot is 3-D. If two values are scalar, the plot is 2-D.

`ar = axialRatio(antenna,frequency,azimuth,elevation)` returns the axial ratio of an antenna, over the specified frequency, and in the direction specified by `azimuth` and `elevation`.

Examples

Calculate Axial Ratio of Antenna

Calculate the axial ratio of an equiangular spiral antenna at azimuth=0 and elevation=0.

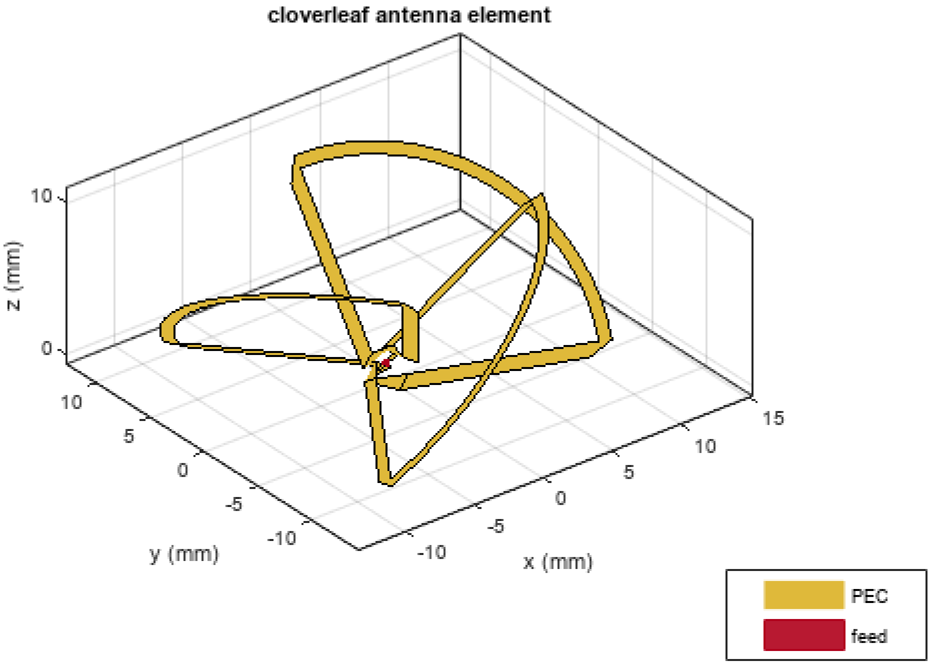
```
s  = spiralEquiangular;
ar = axialRatio(s,3e9,0,0)
```

```
ar = Inf
```

Axial Ratio of Cloverleaf Antenna

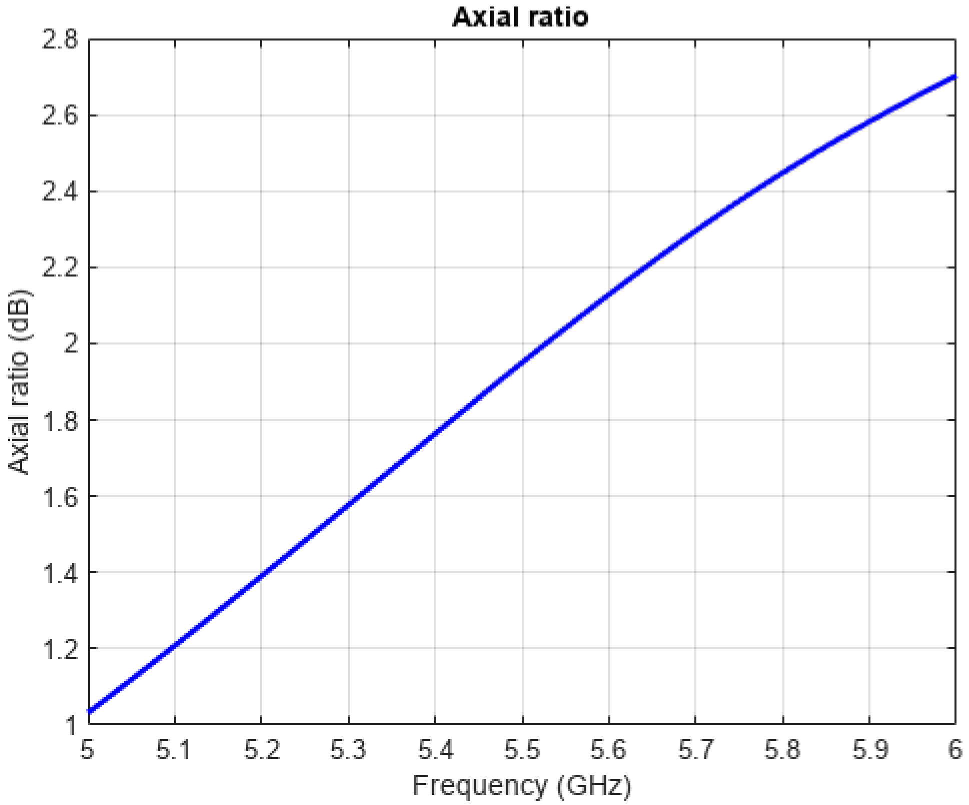
Create a cloverleaf antenna.

```
cl = cloverleaf;
show(cl);
```



Plot the axial ratio of the antenna from 5 GHz to 6 GHz.

```
freq = linspace(5e9,6e9,101);
axialRatio(cl,freq,0,0);
```



The axial ratio plot shows that the antenna supports circular polarization over the entire frequency range.

Input Arguments

`antenna` — Antenna element
object

Antenna object, specified as an object.

`frequency` — Frequency used to calculate axial ratio
scalar | vector

Frequency used to calculate axial ratio, specified as a scalar or vector with each element in Hz.

Example: 70e6

Data Types: `double`

`azimuth` — Azimuth angle of antenna
scalar | vector

Azimuth angle of antenna, specified as a scalar or vector with each element in degrees.

Example: 0

Data Types: `double`

`elevation` — Elevation angle of antenna

scalar | vector

Elevation angle of antenna, specified as a scalar or vector with each element in degrees.

Example: 0

Data Types: `double`

Output Arguments

`ar` — Axial ratio of antenna

scalar in dB

Axial ratio of antenna, returned as a scalar in dB.

Version History

Introduced in R2015a

See Also

`pattern` | `beamwidth`

beamwidth

Beamwidth of antenna

Syntax

```
beamwidth(antenna,frequency,azimuth,elevation)
bw = beamwidth(antenna,frequency,azimuth,elevation,dBdown)
```

```
[bw,angles] = beamwidth(____)
```

Description

`beamwidth(antenna,frequency,azimuth,elevation)` plots the beamwidth of the input antenna at a specified frequency. The beamwidth is the angular separation at which the magnitude of the directivity pattern decreases by a certain value from the peak of the main beam. The directivity decreases in the direction specified by azimuth and elevation angles of the antenna.

Note

- `beamwidth` plots only one beamwidth for symmetrical patterns.
- `beamwidth` might not interpret the data well for partial angle data.

`bw = beamwidth(antenna,frequency,azimuth,elevation,dBdown)` returns the beamwidth of the antenna at a specified `dBdown` value from the peak of the radiation pattern main beam.

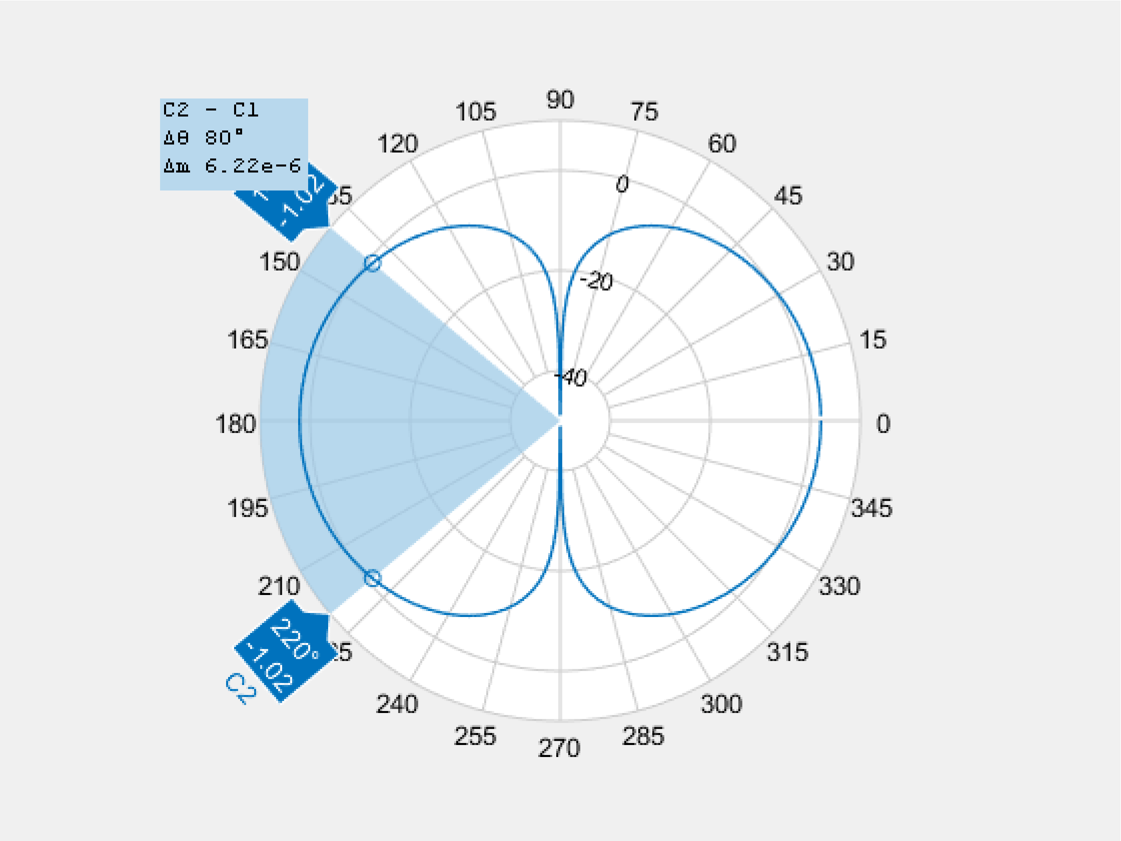
`[bw,angles] = beamwidth(____)` returns the beamwidth and angles (points in a plane) using any input arguments from previous syntaxes.

Examples

Plot Beamwidth of Dipole Antenna

Plot the beamwidth for a dipole antenna at azimuth=0 and elevation=1:1:360 (x-z plane)

```
d  = dipole;
beamwidth(d,70e6,0,1:1:360);
```



Calculate Beamwidth and Angles of Antenna

Calculate the beamwidth of a helix antenna and the angles of the beamwidth. The antenna has an azimuth angle of 1:1:360 degrees, an elevation angle of 0 degrees on the X-Y plane, and a dB down value of 5 dB.

```
hx = helix;
[bw,angles] = beamwidth(hx,2e9,1:1:360,0,5)
```

```
bw = 144
```

```
angles = 1×2

   143   287
```

Plot Beamwidth of Antenna with Symetric Patterns

Create a `fractalGasket` antenna object.

```
fg = fractalGasket("NumIterations",4,"TiltAxis",[0 1 0],'Tilt',90);
```

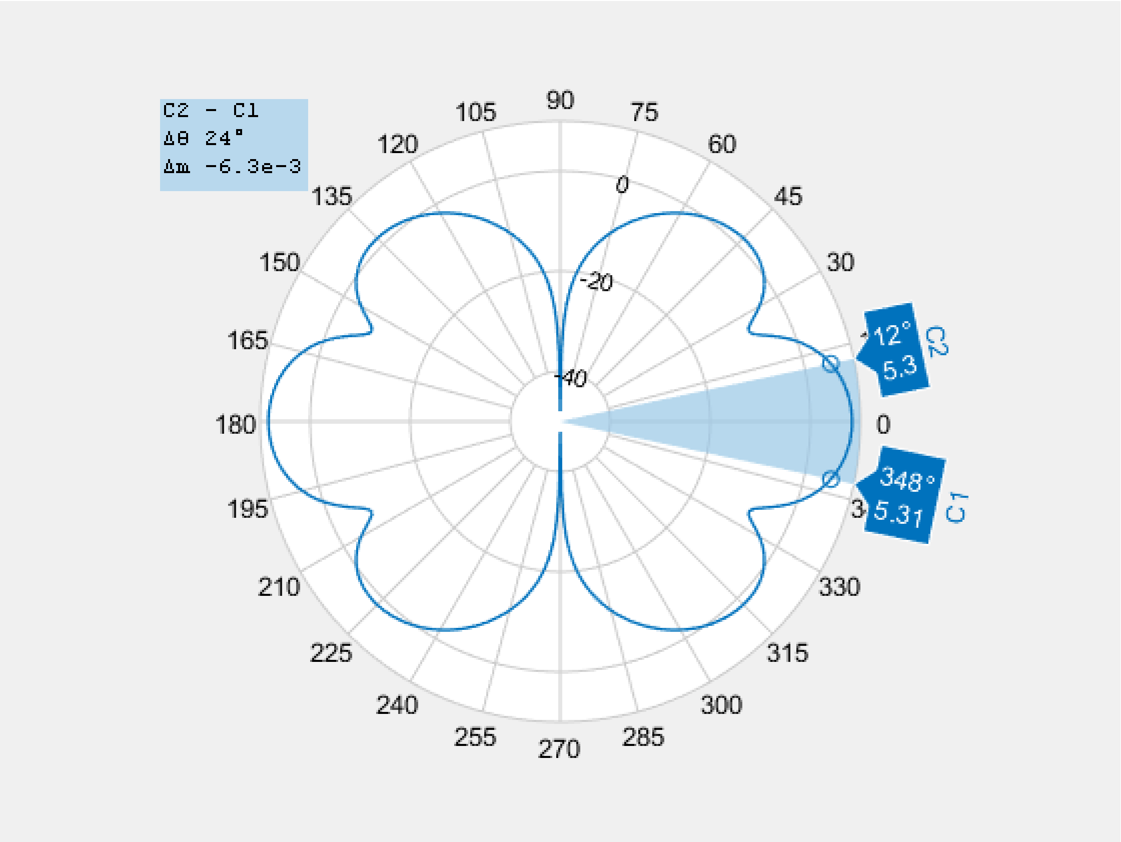
Calculate beamwidth and angle of a `fractalGasket`.

```
[bw,ang] = beamwidth(fg,1.3e9,0,0:1:360) % bw is a 2-by-1 vector.
```

```
bw = 2×1

   24.0000
   24.0000


ang = 2×2

   348    12
   168   192
```

Plot beamwidth.

```
beamwidth(fg,1.3e9,0,0:1:360)
```



Plot Second Beamwidth Solution

Get the `polarpattern` handle.

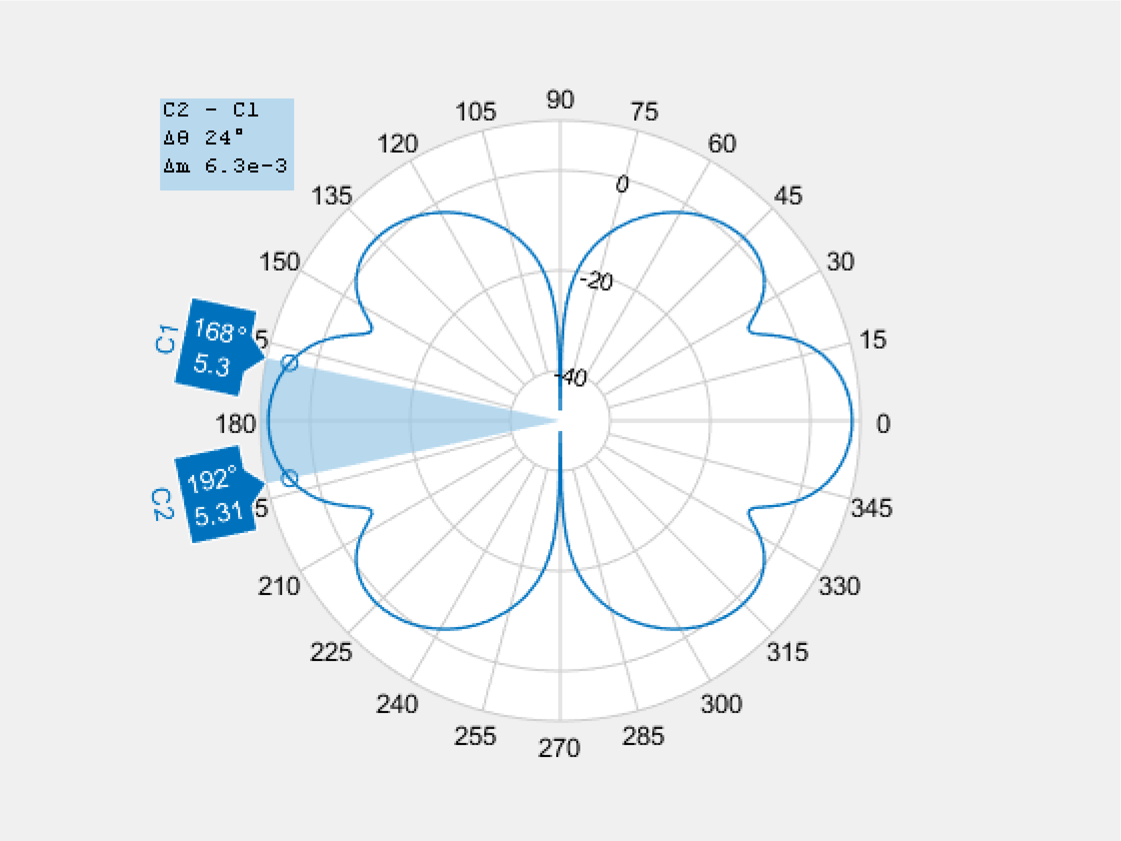
```
P = polarpattern('gco');
```

Hide the beamwidth span and remove the cursor C1 and C2. All the cursors can also be removed using the function `removeCursors`.

```
showSpan(P,0);
removeCursors(P,1);
removeCursors(P,2);
```

Add the cursors at other side of the plot and second beamwidth solution is displayed.

```
addCursor(P,ang(2,:));
showSpan(P,1);
```



Input Arguments

antenna — Antenna object
scalar

Antenna object, specified as a scalar.

`frequency` — Frequency used to calculate beamwidth
scalar in Hz

Frequency to calculate beamwidth, specified as a scalar in Hz.

Example: 70e6

Data Types: `double`

`azimuth` — Azimuth angle of antenna
scalar in degrees | vector in degrees

Azimuth angle of the antenna, specified as a scalar or vector in degrees. If the elevation angle is specified as a vector, then the azimuth angle must be a scalar.

Example: 3

Data Types: `double`

`elevation` — Elevation angle of antenna
scalar in degrees | vector in degrees

Elevation angle of the antenna, specified as a scalar or vector in degrees. If the azimuth angle is specified as a vector, then the elevation angle must be a scalar.

Example: 1:1:360

Data Types: `double`

`dBdown` — Power point from peak of main beam of antenna
`3` (default) | scalar in dB

Power point from peak of main beam of antenna, specified as a scalar in dB.

Example: 5

Data Types: `double`

Output Arguments

`bw` — Beamwidth of antenna
scalar | 2-by-1 vector

Beamwidth of antenna, returned as a scalar in degrees or a 2-by-1 vector with each element unit in degrees.

`angles` — Points on plane
vector in degrees

Points on plane used to measure beamwidth, returned as a vector with each element unit in degrees.

Version History

Introduced in R2015a

See Also

`pattern` | `axialRatio`

mesh

Mesh properties of metal, dielectric antenna, or array structure

Syntax

```
mesh(object)
mesh(shape)
mesh(object,Name=Value)
meshdata = mesh( ___ ,Name=Value)
```

Description

`mesh(object)` plots the mesh that is used to analyze the specified antenna or array element.

`mesh(shape)` plots the mesh for the specified shapes.

`mesh(object,Name=Value)` specifies additional options using name-value arguments.

`meshdata = mesh( ___ ,Name=Value)` returns the mesh as a structure that contains the properties used to analyze the antenna or array. Use this syntax to determine the number of basis functions in the output.

Examples

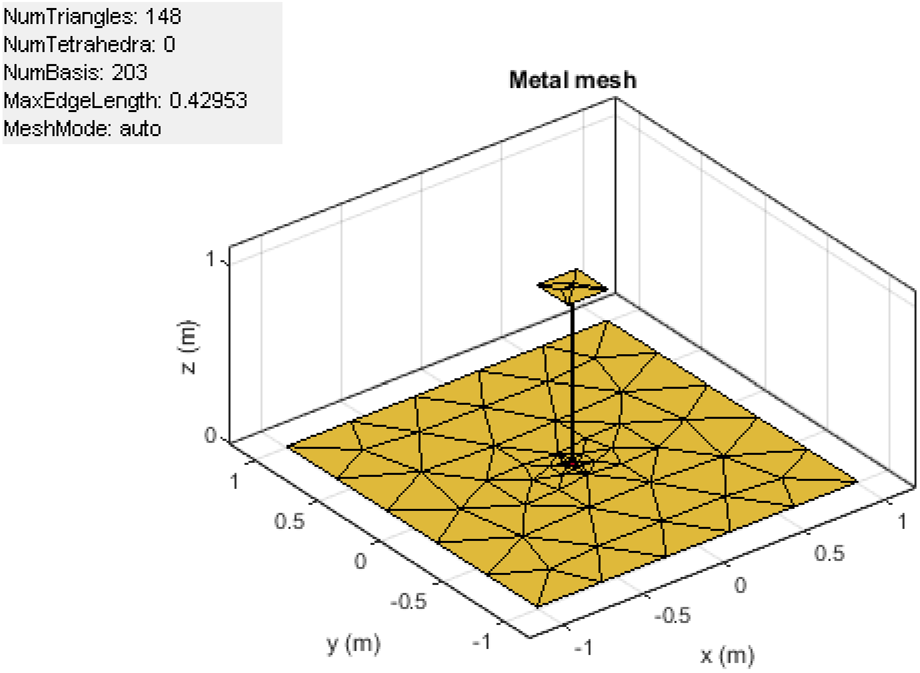
View Mesh Structure of Antenna

Create and view the mesh structure of a top-hat monopole antenna with Maximum edge length of 0.1 m.

```
h = monopoleTopHat;
i = impedance(h,75e6)
```

```
i = 2.4127e+02 + 5.8816e+02i
```

```
mesh(h)
```



```
m = mesh(h)
```

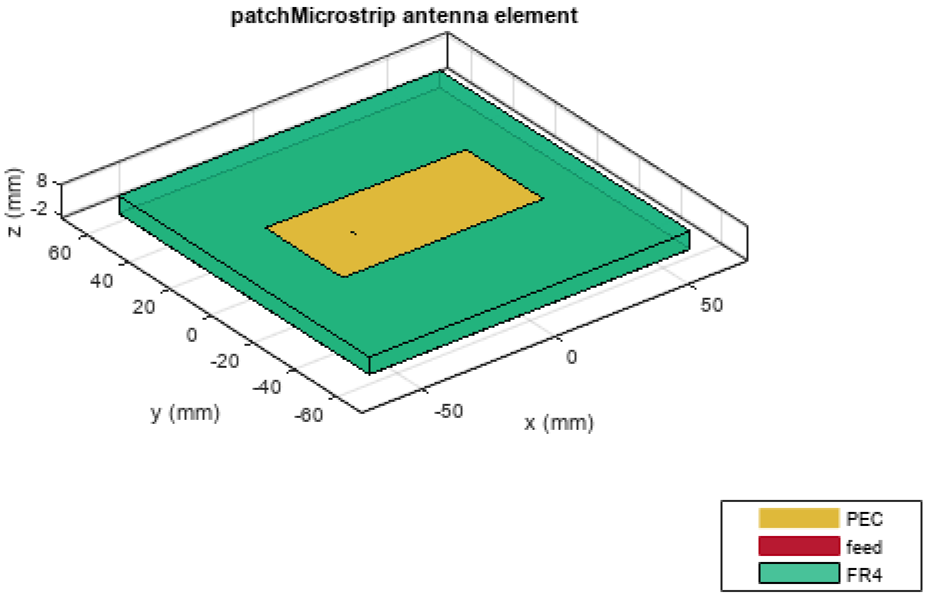
```
m = struct with fields:
     NumTriangles: 148
    NumTetrahedra: 0
         NumBasis: 203
    MaxEdgeLength: 0.4295
    MinEdgeLength: 0.3221
       GrowthRate: 0.9500
         MeshMode: 'auto'
```

Mesh Microstrip Patch Metal-Dielectric Antenna

Radiation Pattern of Microstrip Patch Antenna

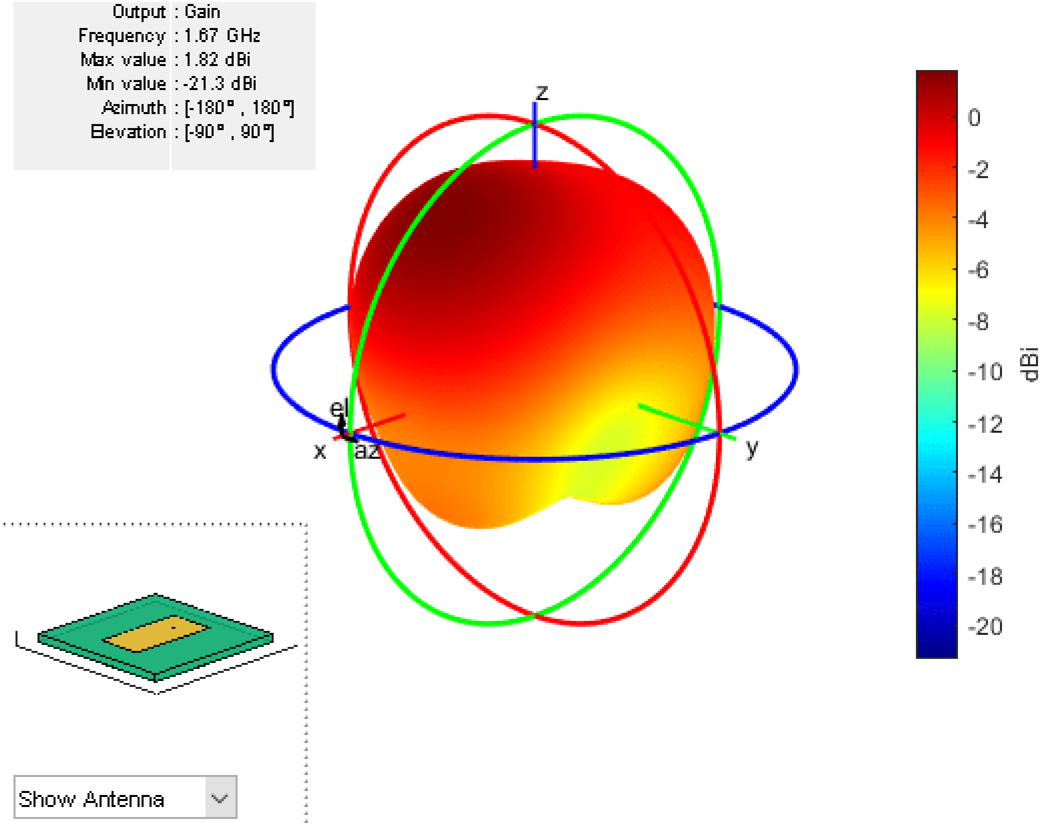
Create a microstrip patch antenna using **'FR4'** as the dielectric substrate.

```
d = dielectric('FR4');
pm = patchMicrostrip(Length=75e-3,Width=37e-3,...
        GroundPlaneLength=120e-3,GroundPlaneWidth=120e-3,...
        Substrate=d);
show(pm)
```



Plot the radiation pattern of the antenna at a frequency of 1.67 GHz.

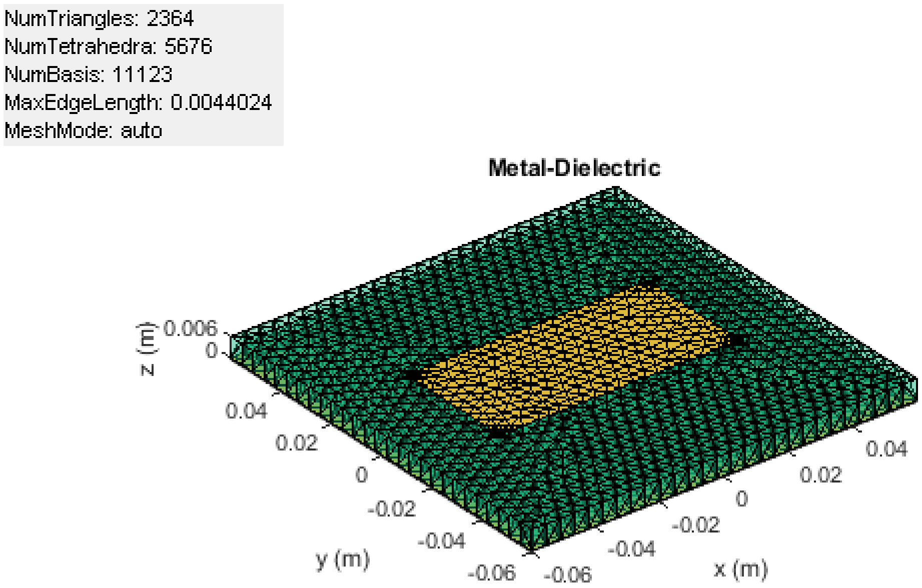
```
figure
pattern(pm,1.67e9)
```



```
   Output : Gain
Frequency : 1.67 GHz
Max value : 1.82 dBi
Min value : -21.3 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Mesh the whole antenna.

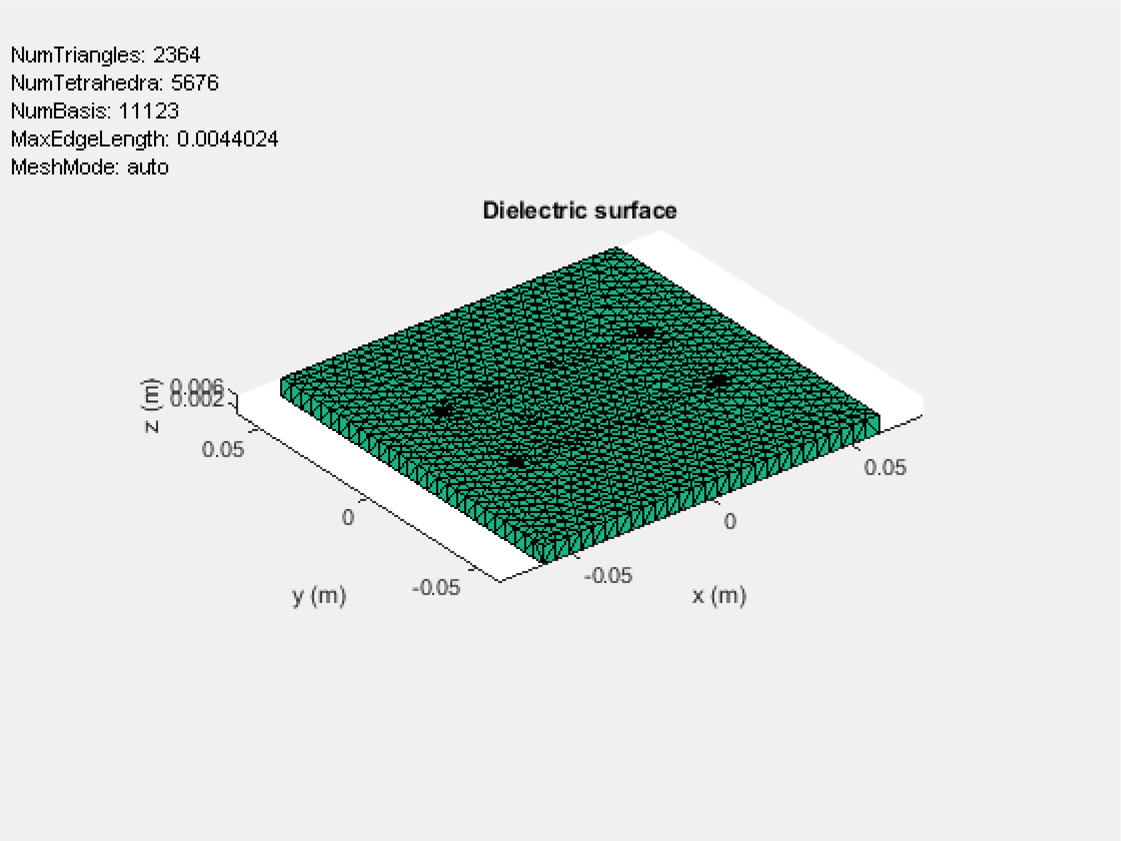
```
figure
mesh(pm)
```



```
NumTriangles: 2364
NumTetrahedra: 5676
NumBasis: 11123
MaxEdgeLength: 0.0044024
MeshMode: auto
```

Mesh only the dielectric surface of the antenna.

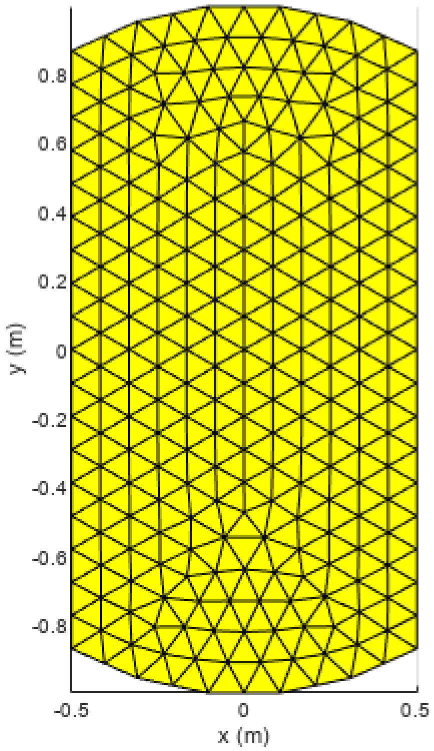
```
figure
mesh(pm,View='dielectric surface')
```



Mesh Arbitrary Shape

Create a rectangular and circular shape, intersect them and mesh at a wavelength of 2 m.

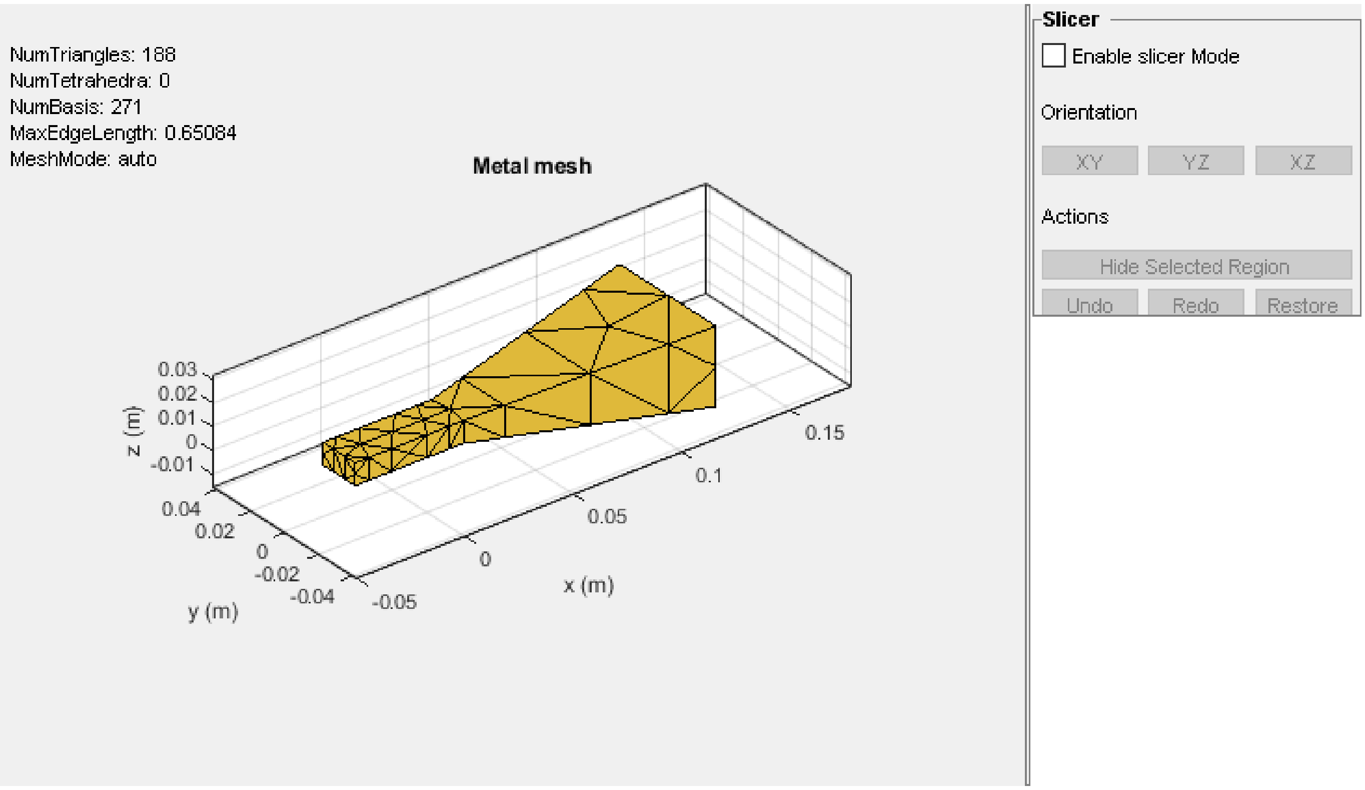
```
r  = antenna.Rectangle;
c  = antenna.Circle;
p = r&c;
mesh(p,2);
```



View Sliced Mesh Plot of Antenna

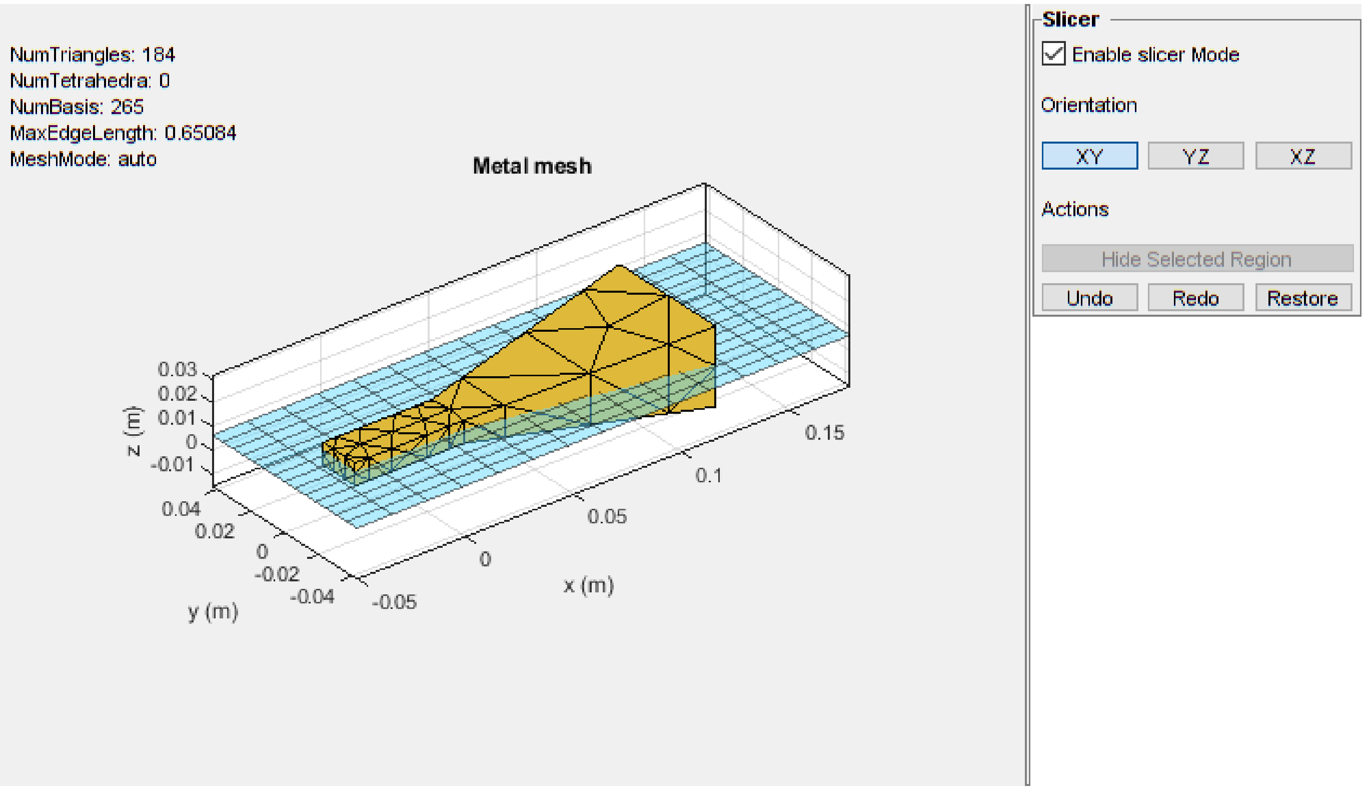
Create a default horn antenna. Mesh the structure with the `'Slicer'` argument set to `'on'`.

```
ant = horn;
z = impedance(ant,70e6);
mesh(ant,Slicer='on')
```



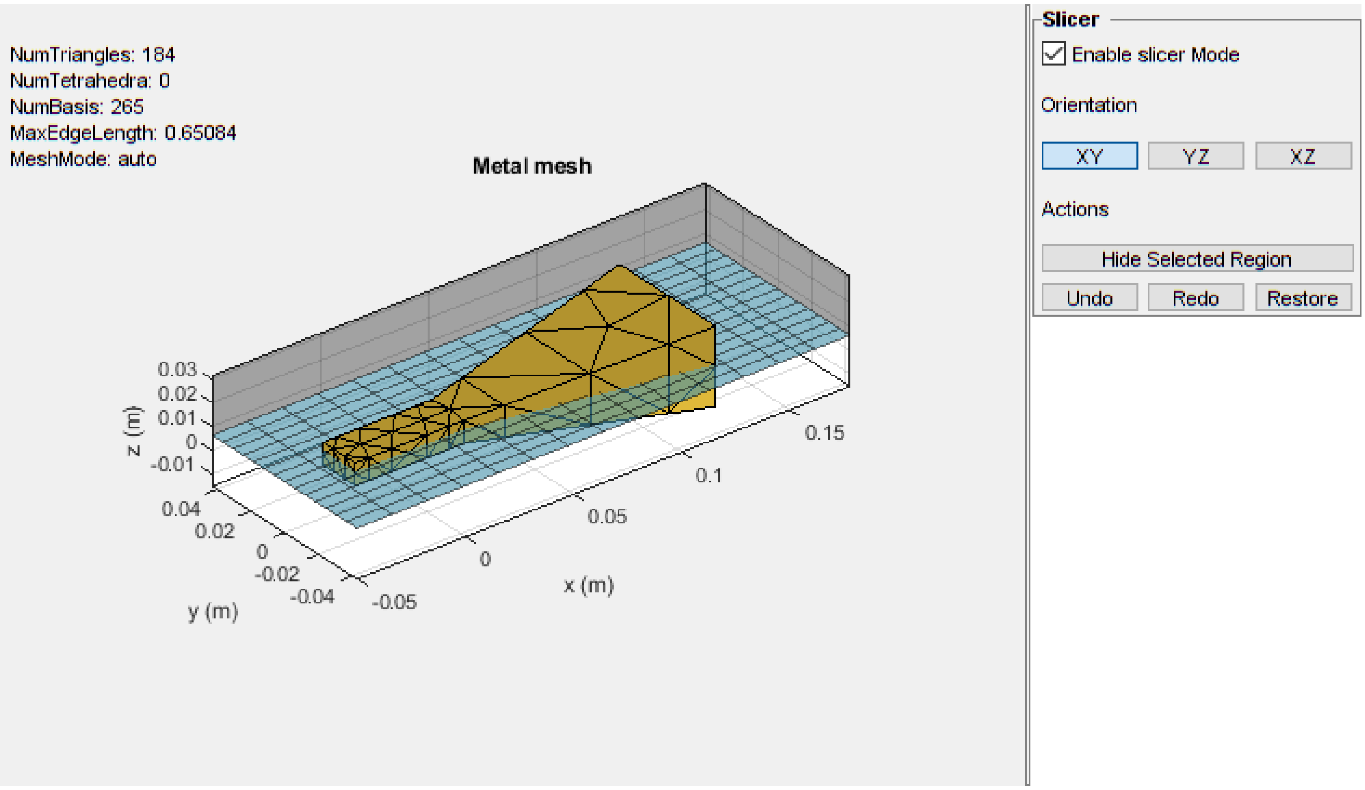
```
NumTriangles: 188
NumTetrahedra: 0
NumBasis: 271
MaxEdgeLength: 0.65084
MeshMode: auto
```

Select **Enable slicer Mode**. Then select a plane for the slice under **Orientation**.



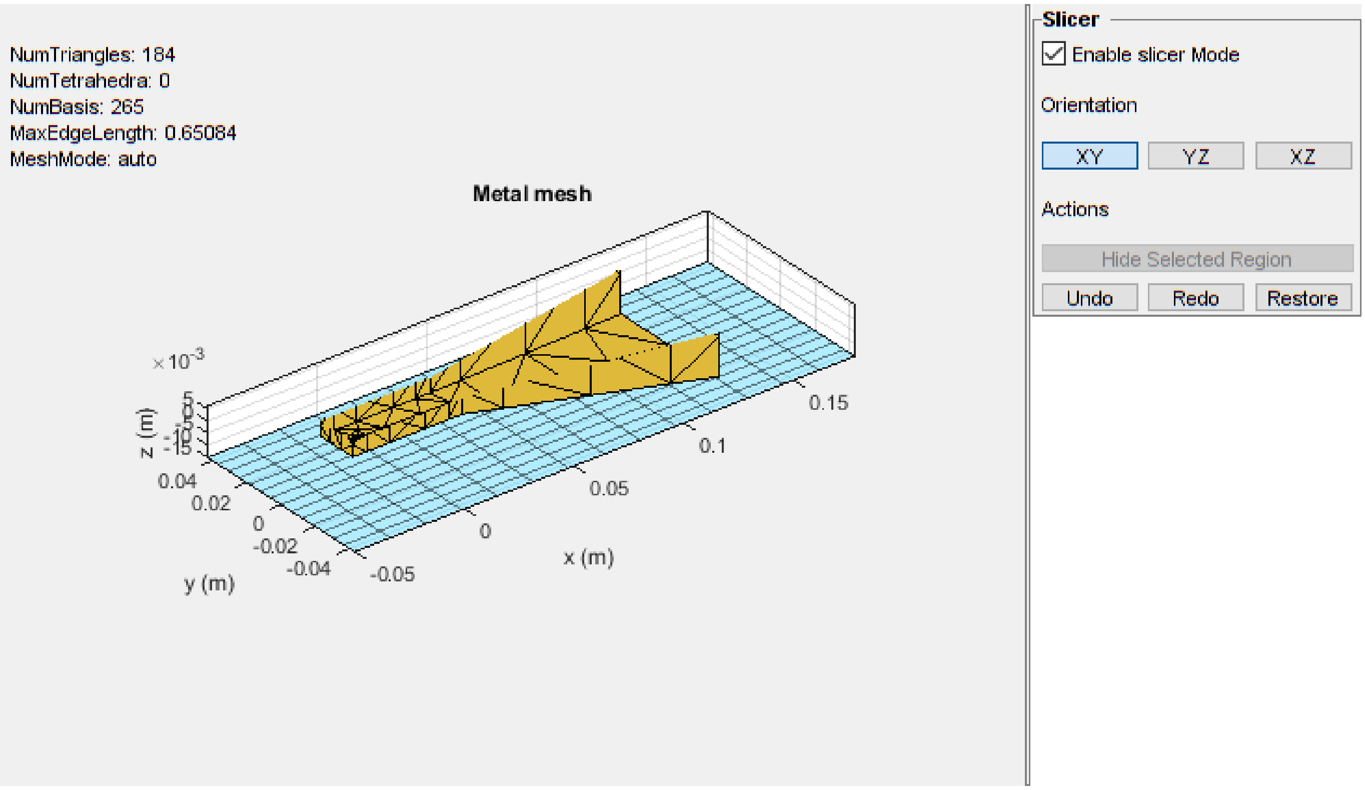
```
NumTriangles: 184
NumTetrahedra: 0
NumBasis: 265
MaxEdgeLength: 0.65084
MeshMode: auto
```

Click on the plot and select a region to hide.



NumTriangles: 184
NumTetrahedra: 0
NumBasis: 265
MaxEdgeLength: 0.65084
MeshMode: auto

Click **Hide Selected Region** to view the desired slice.



```
NumTriangles: 184
NumTetrahedra: 0
NumBasis: 265
MaxEdgeLength: 0.65084
MeshMode: auto
```

Input Arguments

`object` — Antenna or array element

antenna or array object

Antenna or array element, specified as an object.

Example: `ant = dipole; mesh(ant)`

`shape` — Shape created using custom elements and shape objects

shape object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object. You can create the shapes using `antenna.Circle`, `antenna.Polygon`, or `antenna.Rectangle`.

Example: `c = antenna.Rectangle; mesh(c)`

Name-Value Arguments

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

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

Example: `MaxEdgeLength=0.1`

`MaxEdgeLength` — Maximum edge length of triangles in mesh

positive scalar

Maximum edge length of triangles in mesh, specified as a comma-separated pair consisting of `'MaxEdgeLength'` and a scalar in meters.

Example: `MaxEdgeLength=0.1`

Data Types: `double`

`MinEdgeLength` — Minimum edge length of triangles in mesh

positive scalar

Smallest edge length of the triangles in the mesh, specified as a positive scalar in meters.

Example: `MinEdgeLength=0.1`

Data Types: `double`

`GrowthRate` — Mesh growth rate

scalar in the range [0, 1]

Gradation in the triangle sizes of the mesh, specified as a scalar in the range [0, 1].

Example: `GrowthRate=0.7` The value `0.7` states that the growth rate of the mesh is 70 percent.

Data Types: `double`

`View` — Option to customize mesh view of antenna or array element

`'all'` (default) | `'metal'` | `'dielectric surface'` | `'dielectric volume'`

Customize mesh view of antenna or array element, specified as a comma-separated pair consisting of `'View'` and `'all'`, `'metal'`,`'dielectric surface'`, or `'dielectric volume'`.

You choose `'dielectric surface'` to view the boundary triangle mesh of the dielectric. You choose `'dielectric volume'` to view the tetrahedral volume mesh of the dielectric.

Example: `View='metal'`

Data Types: `char`

`Slicer` — Option to enable or disable plot interactivity

`0` (default) | `1` | `'off'` | `'on'`

Option to enable or disable plot interactivity, specified as `'on'` or `'off'`, or as numeric or logical `1(true)` or `0(false)`. Set this argument to `1` or `'on'` to open the plot with the slicer panel, and to slice and view the desired cross section of the plot along the *xy*-, *yz*-, and *xz*- planes. Set this argument to `0` or `'off'` to open the plot without the slicer panel.

Example: `Slicer='on'`

Data Types: `string` | `logical`

Version History

Introduced in R2015a

See Also

`show` | `meshconfig` | `plot`

layout

Display array or PCB stack layout

Syntax

```
layout(array)
layout(pcbstack)
```

Description

`layout(array)` displays the layout of the array object. The circles in the layout corresponds to antenna feed points within the array.

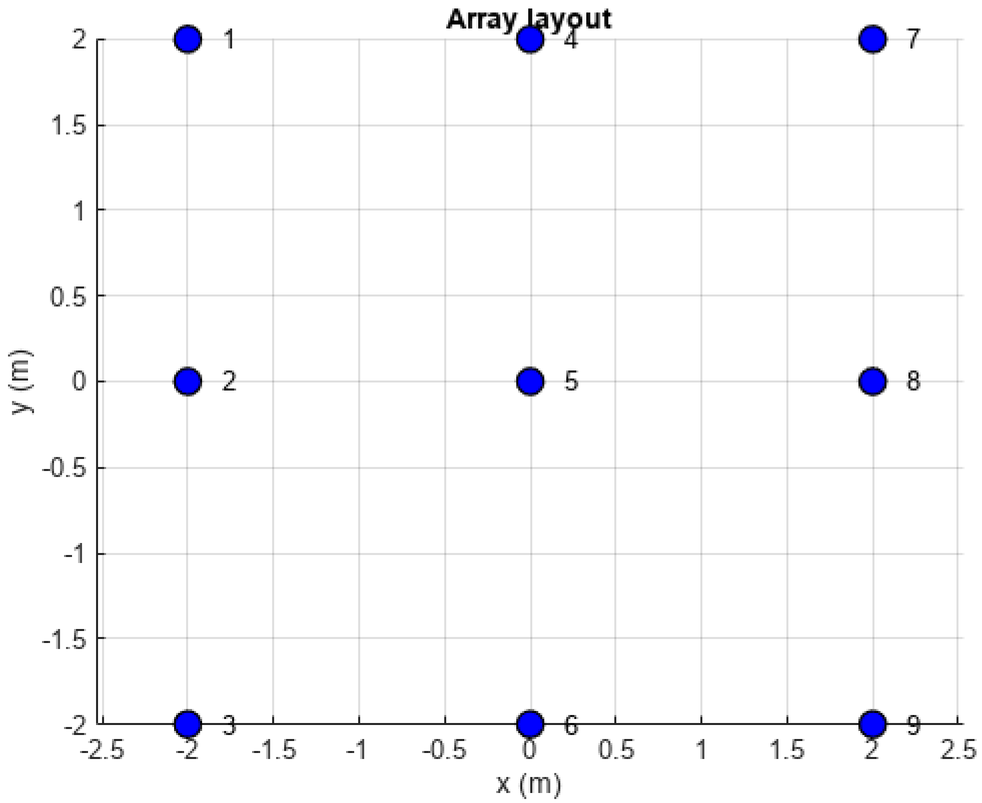
`layout(pcbstack)` displays the layout of the PCB stack object. The circles in the layout corresponds to antenna feed points on the PCB.

Examples

Display Array Layout on X-Y Plane

Create and view a 3x3 rectangular array layout on the X-Y plane.

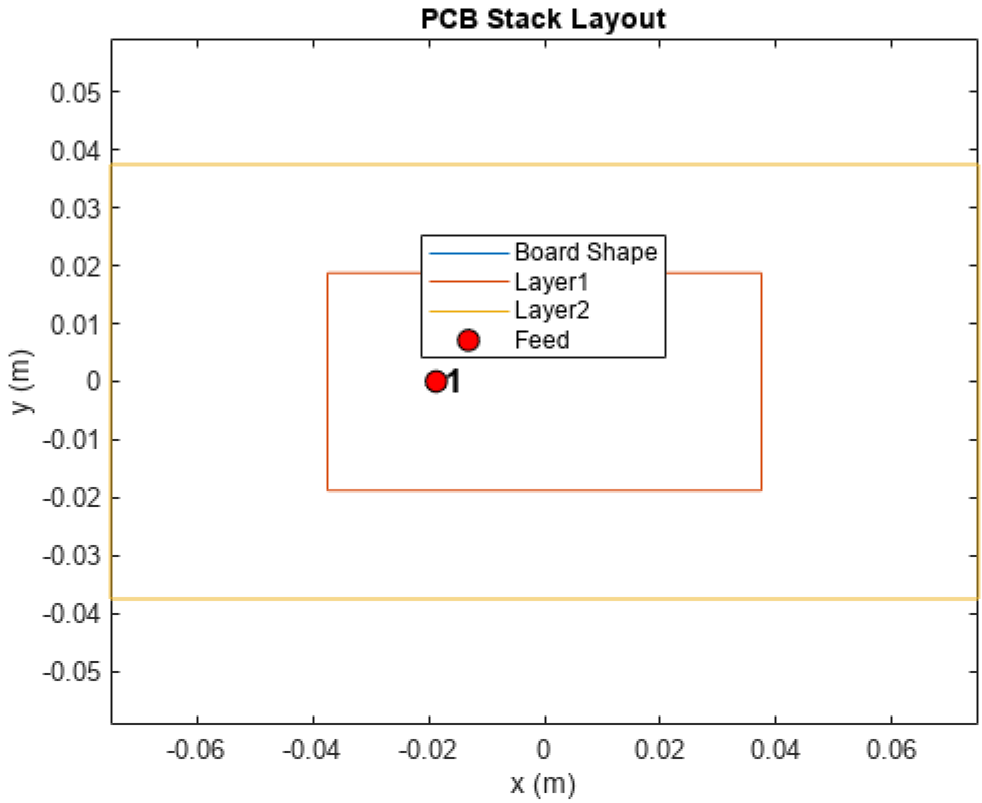
```
h = rectangularArray('Size',[3 3]);
layout(h)
```



Display PCB Stack Layout

Default PCB stack layout.

```
p = pcbStack;
layout(p)
```



Input Arguments

array — Array object
scalar

Array object, specified as a scalar.

pcbstack — PCB stack
pcbStack object

PCB stack, specified as a pcbStack object.

Version History

Introduced in R2015a

See Also

pcbStack | show

lumpedElement

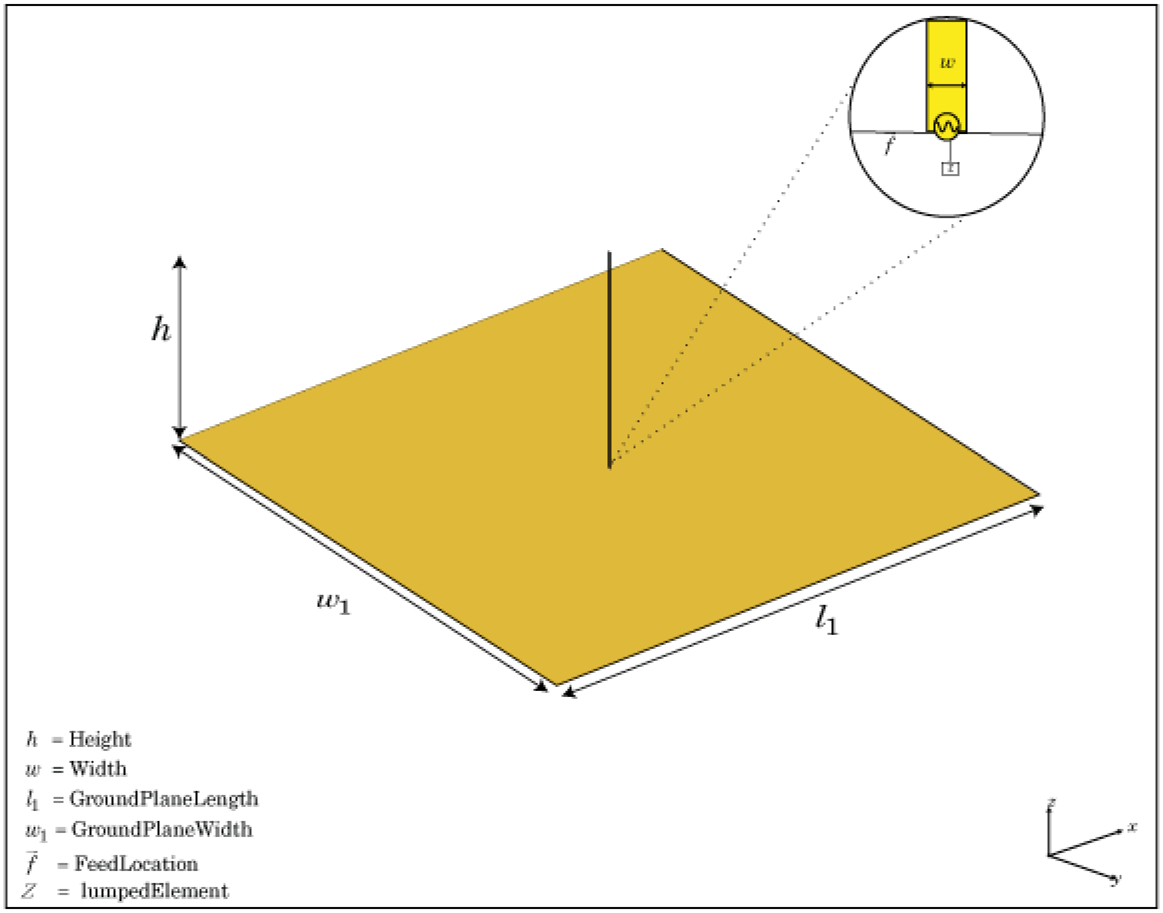
Lumped element circuit to load antenna

Syntax

```
le = lumpedElement
le = lumpedElement(Name=Value)
```

Description

`le = lumpedElement` creates a lumped element circuit. The default value is an empty `lumpedElement` object.



When you load an antenna using a lumped resistor, capacitor, or inductor, the electrical properties of the antennas changes. These lumped elements are typically added to the antenna feed. You can use lumped elements to increase the bandwidth of the antenna without increasing the size of the antenna.

`le = lumpedElement(Name=Value)` returns the lumped element circuit based on the additional options specified by one or more name-value arguments.

Examples

Antenna Using Frequency Independent Load

Create a resistor with 50 Ohms of impedance. Any pure resistive load has a nonvariable impedance when the frequency changes.

```
le = lumpedElement(Impedance=50);
```

Create a dipole antenna. Calculate the impedance of the antenna without loading the antenna.

```
d = dipole;
i1 = impedance(d,70e6)
```

```
i1 = 73.1597 + 0.1659i
```

Load the antenna using a frequency-independent resistor. Calculate the impedance of the antenna.

```
d.Load = le;
i1e1 = impedance(d,70e6)
```

```
i1e1 = 1.2316e+02 + 1.6594e-01i
```

Change the frequency to 85 MHz and calculate the impedance of the antenna.

```
ile2 = impedance(d,85e6)
```

```
ile2 = 2.2378e+02 + 1.1282e+02i
```

Antenna with Two Loads at Arbitrary Locations

Create a dipole antenna using one load at the antenna feed and one load at a location above the antenna feed.

Create a dipole antenna.

```
d = dipole;
```

Create two lumped elements to load the dipole antenna.

One lumped element of impedance, `50 Ohms`, loads the antenna at the feed.

```
l1 = lumpedElement(Impedance=complex(50, -20),Location='feed');
```

The second lumped element of complex impedance, `50+ j*20 Ohms`, loads the antenna at the top. Locate the load half distance from the feed.

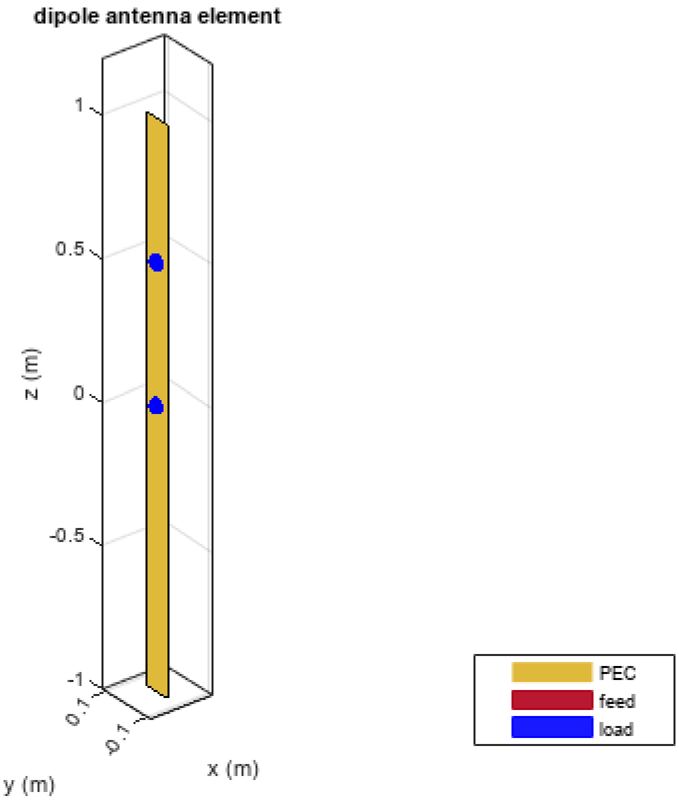
```
l2 = lumpedElement(Impedance=complex(50, -20), Location=[0 0 0.5]);
```

Add the two loads to the dipole antenna.

```
 d.Load = [l1, l2];
```

View the dipole antenna.

```
show(d);
```



Input Arguments

Name-Value Arguments

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

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

Example: `Frequency=2e9`

Impedance — Complex impedance of circuit
real or complex vector of Z-parameters in ohms

Complex impedance of circuit, specified as the comma-separated pair consisting of `'Impedance'` and a real or complex vector of z-parameters in ohms.

Example: `Impedance=complex(75,30)` specifies a complex impedance of `75+i30`.

Data Types: `double`

`Frequency` — Frequency of operation
real vector in Hz

Frequency of operation, specified as the comma-separated pair consisting of `'Frequency'` and a real vector in Hz.

Example: `Frequency=[10e6,20e6,30e6]`

Data Types: `double`

`Location` — Location of load
`[0 0 0]` (default) | Cartesian coordinates

Location of load, specified as the comma-separated pair consisting of `'Location'` and Cartesian coordinates.

Example: `Location=[0 0 0.5]`

Data Types: `double`

Output Arguments

`le` — Lumped element
`lumpedElement` object

Lumped element, returned as a `lumpedElement` object. The real part of the complex number indicates the resistance. The imaginary part of the complex number indicates the reactance.

Version History

Introduced in R2016b

See Also

`dielectric`

vswr

Voltage standing wave ratio of antenna

Syntax

```
vswr(antenna,frequency,z0)
vswrant = vswr(antenna,frequency,z0)
```

Description

`vswr(antenna,frequency,z0)` calculates and plots the voltage standing wave ratio of an antenna, over specified frequency range, and given reference impedance, `z0`.

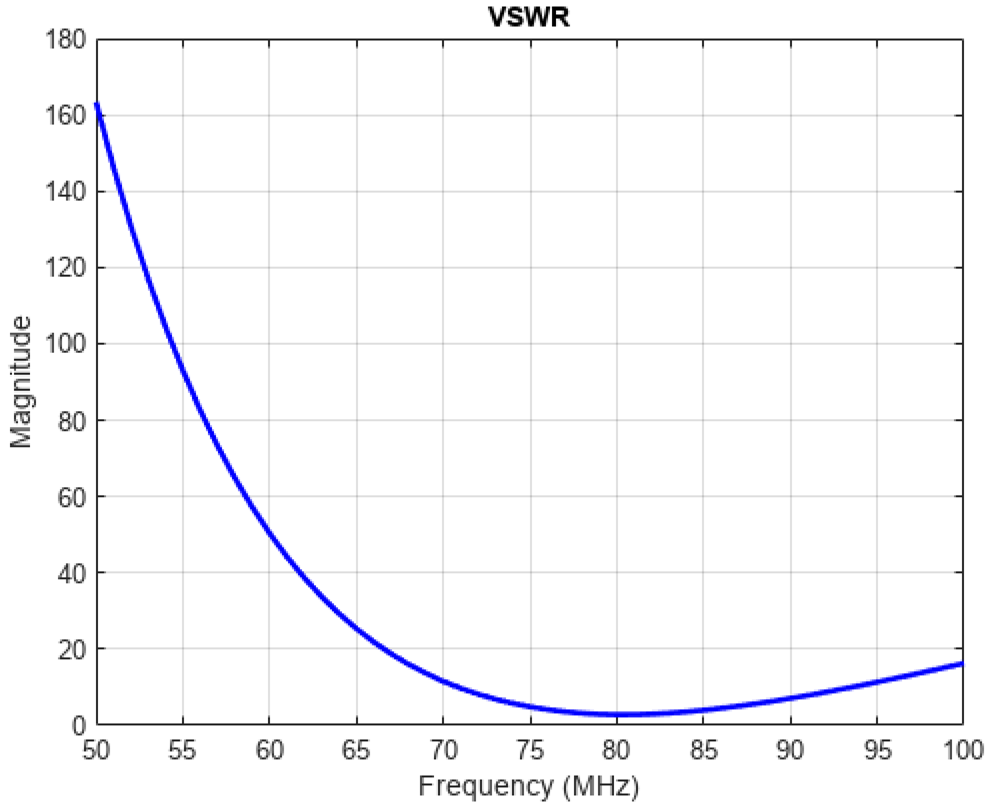
`vswrant = vswr(antenna,frequency,z0)` returns the VSWR of the antenna.

Examples

Plot VSWR of Antenna

Plot vswr (voltage standing wave ratio) of a circular loop antenna.

```
h = loopCircular;
vswr(h,50e6:1e6:100e6,50)
```



Calculate VSWR of Antenna

Calculate vswr (voltage standing wave ratio) of a helix antenna.

```
h = helix;
hvswr = vswr(h,2e9:1e9:4e9,50)
```

```
hvswr = 1×3

    3.5981    6.7141    3.3519
```

Input Arguments

antenna — Antenna object
scalar

Antenna object, specified as a scalar.

frequency — Frequency range used to calculate VSWR
vector in Hz

Frequency range used to calculate VSWR, specified as a vector in Hz. The minimum value of frequency must be 1 kHz.

Example: 50e6:1e6:100e6

Data Types: `double`

`z0` — Reference impedance

50 (default) | scalar

Reference impedance, specified as a scalar in ohms.

Output Arguments

`vswrant` — Voltage standing wave ratio

vector in dB

Voltage standing wave ratio, returned as a vector in dB.

Version History

Introduced in R2015a

See Also

`impedance`

correlation

Correlation coefficient between two antennas in array

Syntax

```
correlation(array,frequency,elem1,elem2,z0)
rho = correlation(array,frequency,elem1,elem2,z0)
```

Description

`correlation(array,frequency,elem1,elem2,z0)` calculates and plots the correlation coefficient between two antenna elements, `elem1` and `elem2` of an array. The correlation values are calculated for a specified frequency and impedance and for a specified impedance `z0`.

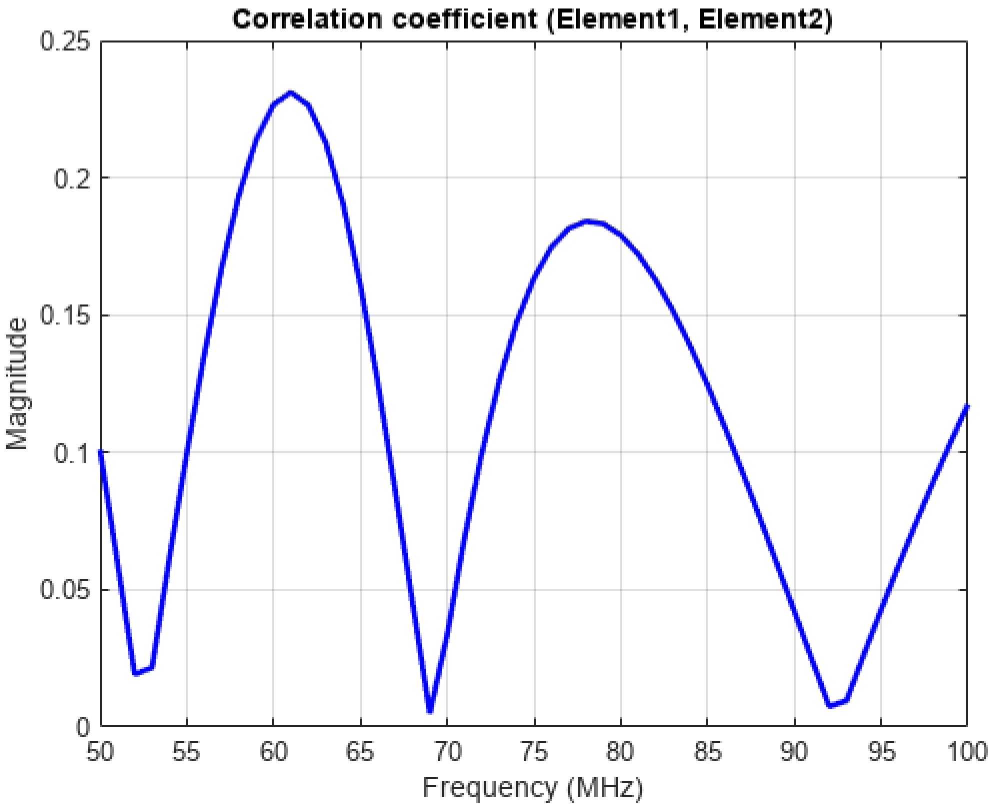
`rho = correlation(array,frequency,elem1,elem2,z0)` returns the correlation coefficient between two antenna elements, `elem1` and `elem2` of an array.

Examples

Plot Correlation of Array

Plot the correlation between 1 and 2 antenna elements in a default linear array over a frequency range of 50MHz to 100MHz.

```
h = linearArray;
correlation (h,50e6:1e6:100e6,1,2);
```



Calculate Correlation Coefficient of Array

Calculate correlation coefficient of default rectangular array at a frequency range of 50MHz to 100MHz.

```
h = rectangularArray;
rho = correlation (h, 50e6:1e6:100e6, 1, 2)
```

```
rho = 51×1

    0.1414
    0.1120
    0.0823
    0.0521
    0.0212
    0.0105
    0.0432
    0.0765
    0.1096
    0.1410
      ⋮
```

Input Arguments

`array` — Array object

scalar

Array object, specified as a scalar.

`frequency` — Frequency range used to calculate correlation

vector in Hz

Frequency range used to calculate correlation, specified as a vector in Hz.

Example: 50e6:1e6:100e6

Data Types: `double`

`elem1,elem2` — Antenna elements in an array

scalar

Antenna elements in an array, specified as a scalar.

`z0` — Reference impedance

50 (default) | scalar in ohms

Reference impedance, specified as a scalar in ohms.

Example: 70

Data Types: `double`

Output Arguments

`rho` — Correlation coefficient between two antenna elements of an array

vector

Correlation coefficient between two antenna elements of an array, returned as a vector.

Version History

Introduced in R2015a

References

[1] S. Blanch, J. Romeu, and I. Corbella. *Exact representation of antenna system diversity performance from input parameter description*. Electron. Lett., vol. 39, pp. 705-707, May 2003.

See Also

`impedance` | `returnLoss` | `sparameters`

cylinder2strip

Cylinder equivalent width approximation

Syntax

```
w = cylinder2strip(r)
```

Description

`w = cylinder2strip(r)` calculates the equivalent width of a strip approximation for a cylinder cross section.

Examples

Calculate Cylinder to Strip Approximation

Calculate the width of the strip approximation to a cylinder of radius 20 mm.

```
w = cylinder2strip(20e-3)
```

```
w = 0.0800
```

Input Arguments

r — Cylindrical cross-section radius
scalar in meters | vector in meters

Cylindrical cross-section radius, specified as a scalar or vector in meters.

Example: 20e-3

Data Types: `double`

Output Arguments

w — Equivalent width of strip
scalar | vector

Equivalent width of strip, returned as a scalar or vector.

Data Types: `double`

Version History

Introduced in R2015a

See Also

`helixpitch2spacing`

helixpitch2spacing

Spacing between turns of helix

Syntax

```
s = helixpitch2spacing(a,r)
```

Description

`s = helixpitch2spacing(a,r)` calculates the spacing between the turns of a helix antenna given the pitch angle, `a`, and the radius of the helix, `r`.

Examples

Calculate Spacing Between Helix Turns

Calculate spacing for helix with pitch varying from 12 degrees to 14 degrees in steps of 0.5 and 20 mm radius.

```
s = helixpitch2spacing(12:0.5:14,20e-3)
```

```
s = 1×5

    0.0267    0.0279    0.0290    0.0302    0.0313
```

Calculate Spacing for Helix with Varying Pitch

Calculate spacing for helix with pitch varying from 12 degrees to 14 degrees in steps of 0.5 and radius 20 mm.

```
s = helixpitch2spacing(12:0.5:14,20e-3)
```

```
s = 1×5

    0.0267    0.0279    0.0290    0.0302    0.0313
```

Calculate Spacing of Helix Antenna with Varying Radius

Calculate the spacing of a helix that has a pitch of 12 degrees and a radius that varies from 20 mm to 22 mm in steps of 0.5 mm.

```
s = helixpitch2spacing(12,20e-3:0.5e-3:22e-3)
```

```
s = 1×5
```

```
    0.0267    0.0274    0.0280    0.0287    0.0294
```

Calculate Spacing of Helix with Varying Pitch and Radius

Calculate spacing for helix with pitch varying from 12 degrees to 14 degrees in steps of 0.5 and radius varying from 20mm to 22mm in steps of 0.5.

```
s = helixpitch2spacing(12:0.5:14,20e-3:0.5e-3:22e-3)
```

```
s = 1×5

    0.0267    0.0286    0.0305    0.0324    0.0345
```

Input Arguments

a — Pitch angle of helix
scalar in meters | vector in meters

Pitch angle of helix, specified as a scalar or vector in meters.

Example: 12:0.5:14

r — Radius of helix
scalar in meters | vector in meters

Radius of helix, specified as a scalar or vector in meters.

Example: 20e-3

Note If the pitch angle and radius are both vectors, then their lengths must be equal.

Output Arguments

s — Spacing between helix turns
scalar in meters | vector in meters

Spacing between helix turns, returned as a scalar or vector in meters.

Version History

Introduced in R2015a

See Also

`cylinder2strip`

meshconfig

Change mesh mode of antenna structure

Syntax

```
meshconfig(antenna,mode)
m = meshconfig(antenna,mode)
```

Description

`meshconfig(antenna,mode)` changes the meshing mode of the antenna according to the text input mode.

`m = meshconfig(antenna,mode)` changes the meshing mode of the antenna according to the text input mode.

Examples

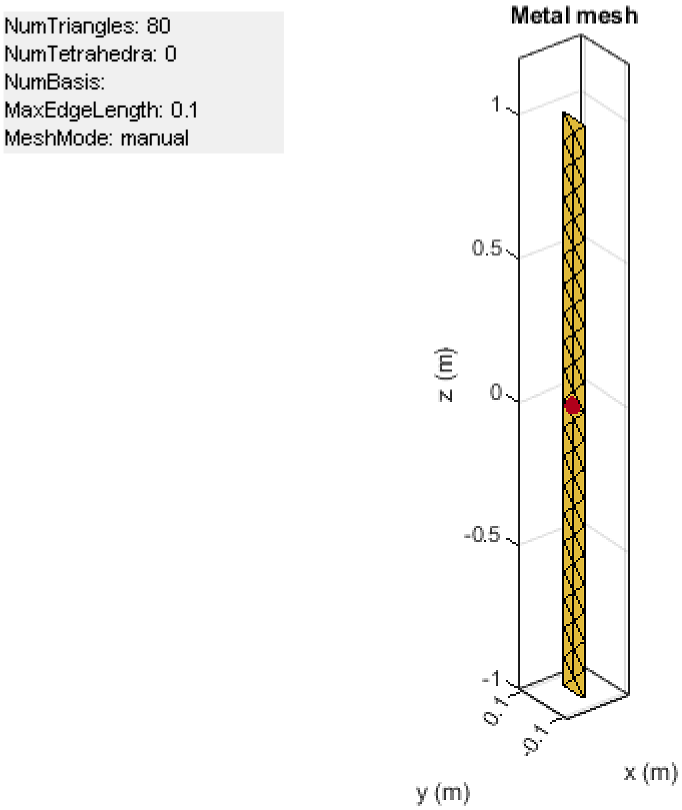
Change Mesh Configuration of Antenna

Change the mesh configuration of a dipole antenna from auto (default) to manual mode.

```
h = dipole;
meshconfig(h,'manual')
```

```
ans = struct with fields:
     NumTriangles: 0
    NumTetrahedra: 0
         NumBasis: []
    MaxEdgeLength: []
    MinEdgeLength: []
       GrowthRate: []
         MeshMode: 'manual'
```

```
mesh(h,'MaxEdgeLength',0.1)
```



NumTriangles: 80
NumTetrahedra: 0
NumBasis:
MaxEdgeLength: 0.1
MeshMode: manual

Input Arguments

antenna — Antenna object
scalar

Antenna object, specified as a scalar.

mode — Meshing mode
`'auto'` (default) | `'manual'`

Meshing mode, specified as `'auto'` or `'manual'`.

Data Types: `char`

Version History

Introduced in R2015a

See Also

`show` | `mesh`

numSummationTerms

Change number of summation terms for calculating periodic Green's function

Syntax

```
numSummationTerms(array,num)
```

Description

`numSummationTerms(array,num)` changes the number of summation terms used to calculate periodic Green's function of the infinite array. This method calculates $2 * num + 1$ of the periodic Green's function. The summation is carried out from `-num` to `+num`. A higher number of terms results in better accuracy but increases the overall computation time.

Input Arguments

`array` — Infinite array
scalar

Infinite array, specified as a scalar.

`num` — Number to calculate summation terms
10 (default) | scalar

Number to calculate summation terms, specified as a scalar. The summation is carried out from `-num` to `+num`.

Example: `50`

Examples

Change Number of Summation Terms in Infinite Array

Create an infinite array with the scan elevation at 45 degrees. Calculate the scan impedance. By default, the number of summation terms used is 21.

```
h = infiniteArray('ScanElevation',45);
s = impedance(h,1e9)
```

```
s = 85.0891 + 71.2268i
```

Change the number of summation terms to 51. Calculate the scan impedance again.

```
numSummationTerms(h,25)
s = impedance(h,1e9)
```

```
s = 85.2350 + 71.2606i
```

Change the number of terms to 101. Increasing the number of summation terms results in a more accurate scan impedance. However, the time required to calculate the scan impedance increases.

```
numSummationTerms(h,50)
s = impedance(h,1e9)
```

```
s = 85.2802 + 71.2652i
```

Version History

Introduced in R2015b

See Also

`pattern` | `beamwidth`

Topics

"Infinite Arrays"

feedCurrent

Calculate current at feed for antenna or array

Syntax

```
i = feedCurrent(obj,frequency)
```

Description

`i = feedCurrent(obj,frequency)` calculates and returns the complex current in Ampere at the feed for an antenna or array object at a specified frequency. The feed current when multiplied by the antenna impedance gives the voltage across the antenna.

Examples

Feed Current of Monopole Antenna Excited By Plane Wave

Excite a monopole antenna using plane wave. Calculate the feed current at 75 MHz.

```
h = planeWaveExcitation('Element',monopole, 'Direction',[1 0 0])
```

```
h = 
  planeWaveExcitation with properties:

         Element: [1x1 monopole]
       Direction: [1 0 0]
    Polarization: [0 0 1]
      SolverType: 'MoM'
```

```
cur = feedCurrent(h,75e6)
```

```
cur = -0.0138 + 0.0135i
```

Feed Current of Rounded-Bowtie Antenna

Calculate the feed current of a rounded-bowtie designed for operation at 2.4 GHz.

```
b  = design(bowtieRounded,2.4e9);
If = feedCurrent(b,2.4e9)
```

```
If = 0.0297 - 0.0003i
```

Feed Current of Dipole Antenna

Calculate the feed current of a dipole antenna designed for operation at 70 MHz and 75 MHz.

```
feedCurrent(dipole, [75e6, 70e6])
```

```
ans = 1×2 complex

   0.0137 + 0.0000i   0.0089 - 0.0036i
```

Input Arguments

`obj` — Antenna or array object

object

Antenna or array object, specified as an object.

`frequency` — Frequency to calculate feed current

scalar | vector in Hz

Frequency to calculate feed current, specified as a scalar integer in Hz or a vector with each element specified in Hz.

Output Argument

`i` — Complex current

1-by-*n* vector in A

Complex current, returned as a 1-by-*n* vector in A. This value is calculated at the feed point of an antenna or array.

Version History

Introduced in R2017a

See Also

`current`

fieldsCustom

Plot electric or magnetic fields of antenna

Syntax

```
fieldsCustom(fields,points)
fieldsCustom(fields,points,scalefield)
qobj = fieldsCustom( ___ )
```

```
fieldsCustom(axeshandle, ___ )
```

Description

`fieldsCustom(fields,points)` plots electric or magnetic field vectors, `fields`, at specified points in space, `points`, in the current axes.

`fieldsCustom(fields,points,scalefield)` scales the field arrows by a scalar value, `scalefield`.

`qobj = fieldsCustom( ___ )` returns the quiver object, using either of the previous syntaxes.

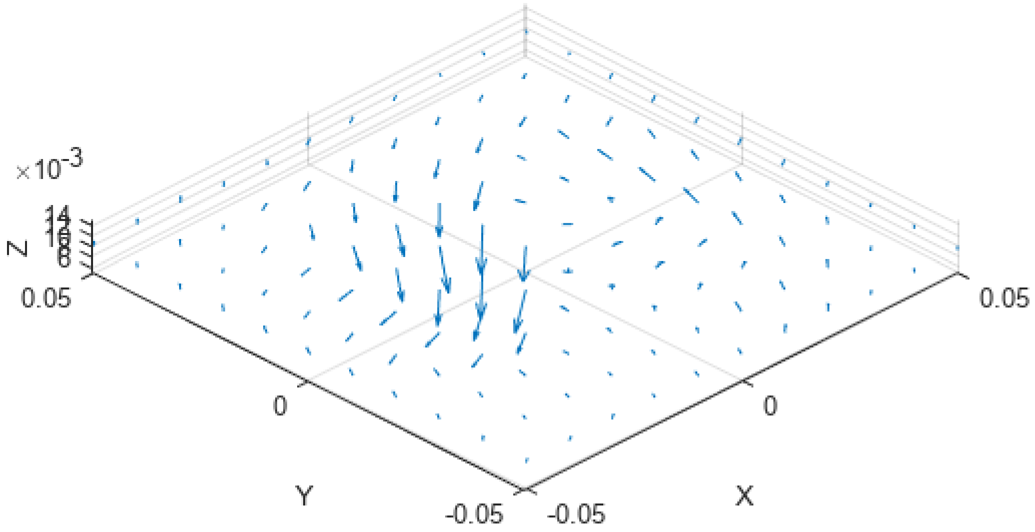
`fieldsCustom(axeshandle, ___ )` plots into the axes specified by `axeshandle` instead of the current axes.

Examples

Visualize Magnetic Field of Antenna Using `fieldsCustom`

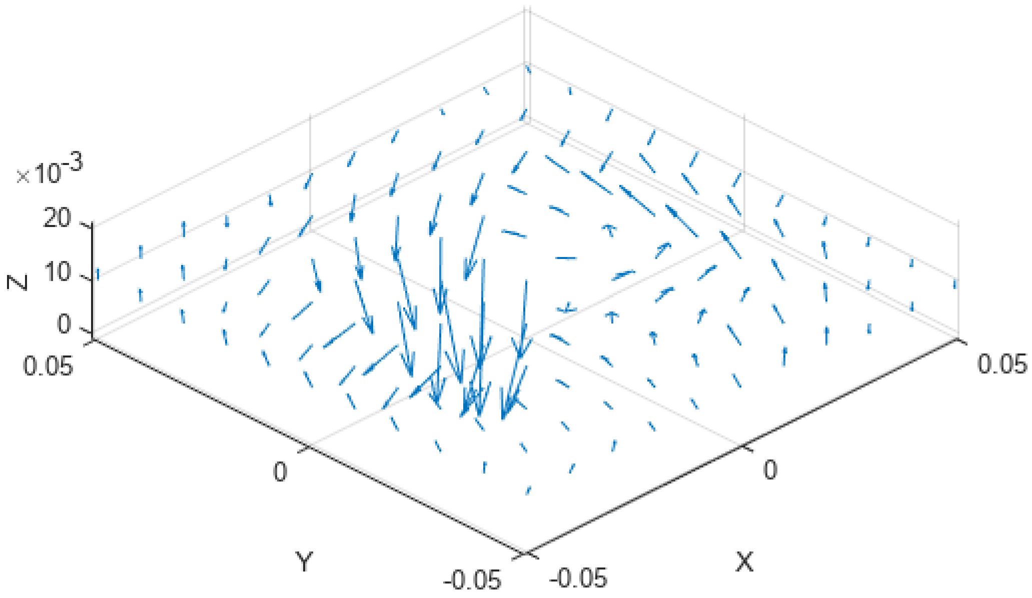
Load and visualize the magnetic field data available in the file '`fielddata.mat`'.

```
load fielddata
fieldsCustom(H,p)
```



Scale the magnetic field arrows by a factor of 2.

```
figure
fieldsCustom(H,p,2)
```



Input Arguments

fields — Electric or magnetic field vectors

3-by-*p* complex matrix

Electric or magnetic field vectors, specified as a 3-by-*p* complex matrix. *p* is the number of points in space.

Data Types: `double`

points — x, y, z coordinates in space

3-by-*p* real matrix

x, *y*, *z* coordinates in space, specified as a 3-by-*p* real matrix. *p* is the number of points in space.

Data Types: `double`

axeshandle — Axes object

object

Axes object, specified as an object.

Data Types: `char`

scalefield — Value by which to scale field arrows

0.9 (default) | scalar

Value by which to scale the field arrows, specified as a scalar. A value of `2` doubles the relative length of the field arrows. A value of `0.5` halves the length of the field arrows. A value of `0` plots the field arrows without automatic scaling.

Example: 2

Data Types: `double`

Output Arguments

`qobj` — Electric or magnetic field plot

quiver object

Electric or magnetic field plot, returned as quiver object.

Version History

Introduced in R2016a

See Also

`pattern` | `EHfields` | `patternCustom`

patternCustom

Plot radiation pattern using spherical coordinate system (phi and theta angles)

Syntax

```
patternCustom(magE,theta,phi)
patternCustom(magE,theta,phi,Name,Value)
hplot = patternCustom( ___ )
```

Description

`patternCustom(magE,theta,phi)` plots the 3-D radiation pattern of an antenna with magnitude `magE` over the specified `phi` and `theta` angle vectors. Using `patternCustom` on multiple datasets plots multiple radiation patterns in the same figure.

`patternCustom(magE,theta,phi,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

`hplot = patternCustom( ___ )` returns handles of the lines or surface in the figure window. This syntax accepts any combination of arguments from the previous syntaxes

Examples

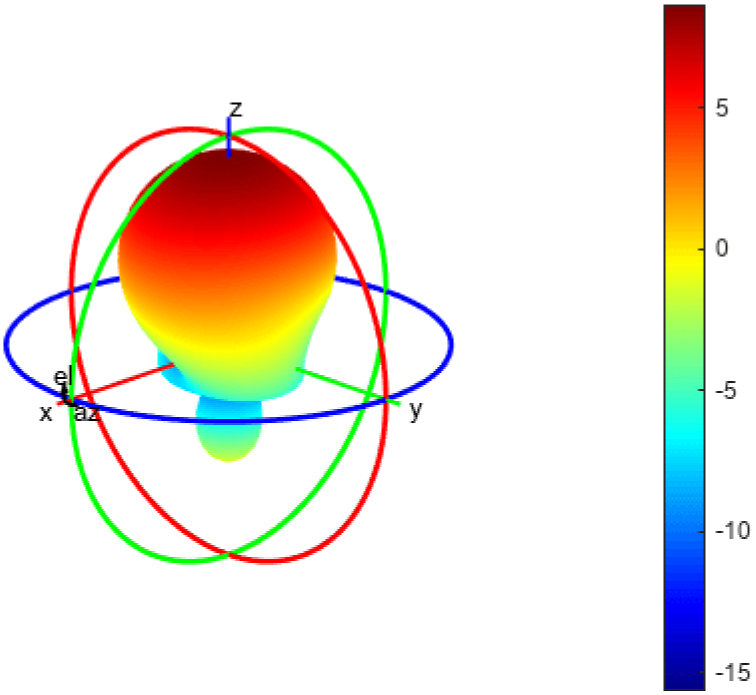
Visualize Radiation Pattern From Antenna Data File

Consider a helix antenna data file in .csv format. This file contains the magnitude of the antenna directivity in phi and theta angles. Read the file .

```
helixdata = csvread('antennadata_test.csv',1,0);
```

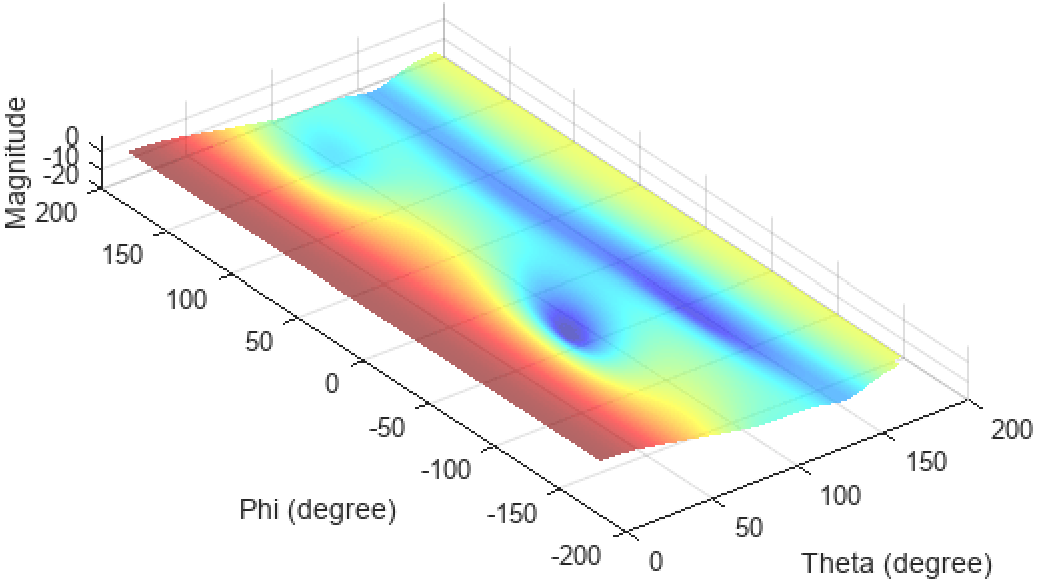
Use patternCustom to extract the magnitude of directivity, and the phi, and theta angle values. Plot the 3-D polar radiation pattern.

```
patternCustom(helixdata(:,3),helixdata(:,2),helixdata(:,1));
```



Use the same data to plot the 3-D rectangular radiation pattern using pattern plot options.

```
p = PatternPlotOptions('Transparency',0.6);
figure
patternCustom(helixdata(:,3),helixdata(:,2),helixdata(:,1),...
 'CoordinateSystem','rectangular','PatternOptions',p);
```



Visualize 2-D Radiation Patterns of Helix Directivity

Calculate the magnitude, azimuth, and elevation angles of a helix's directivity at 2 GHz.

```
h = helix;
[D,az,el] = pattern(h,2e9);
```

Here, `az` = the angle from the positive *x*-axis to the directivity vector's orthogonal projection onto the *xy* plane, moving in the direction towards the y-axis.

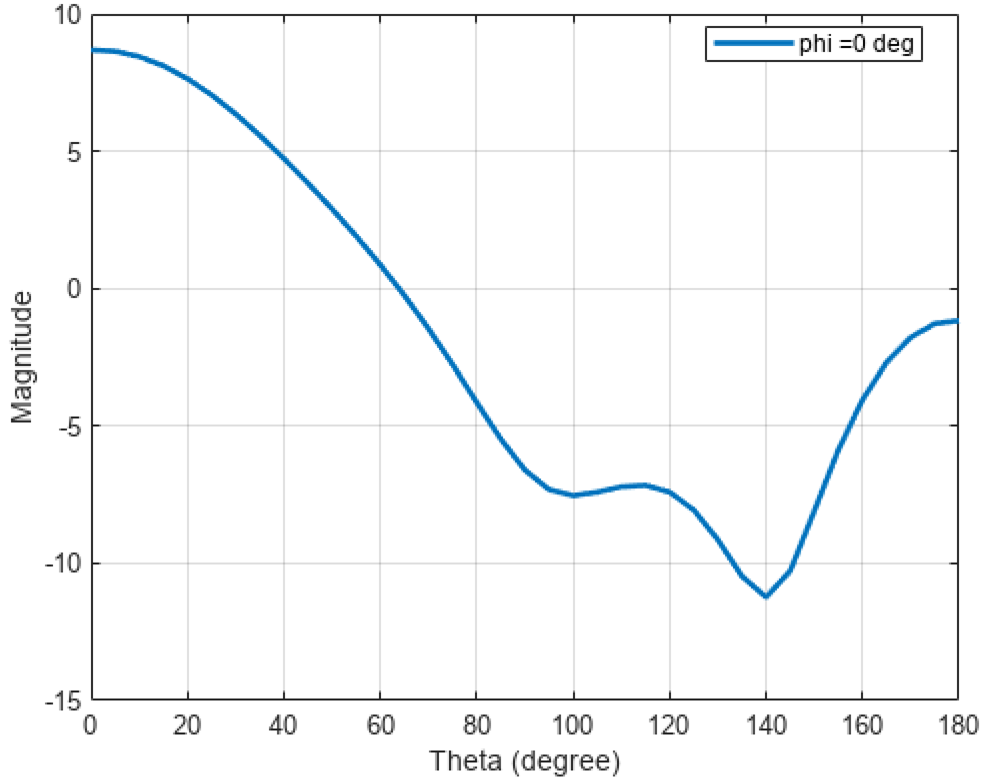
`el` = the angle from the directivity vector's orthogonal projection onto the *xy* plane to the vector, moving in the direction towards the z-axis.

Extract theta and phi angles of the directivity magnitude.

```
phi = az';
theta = (90-el);
MagE = D';
```

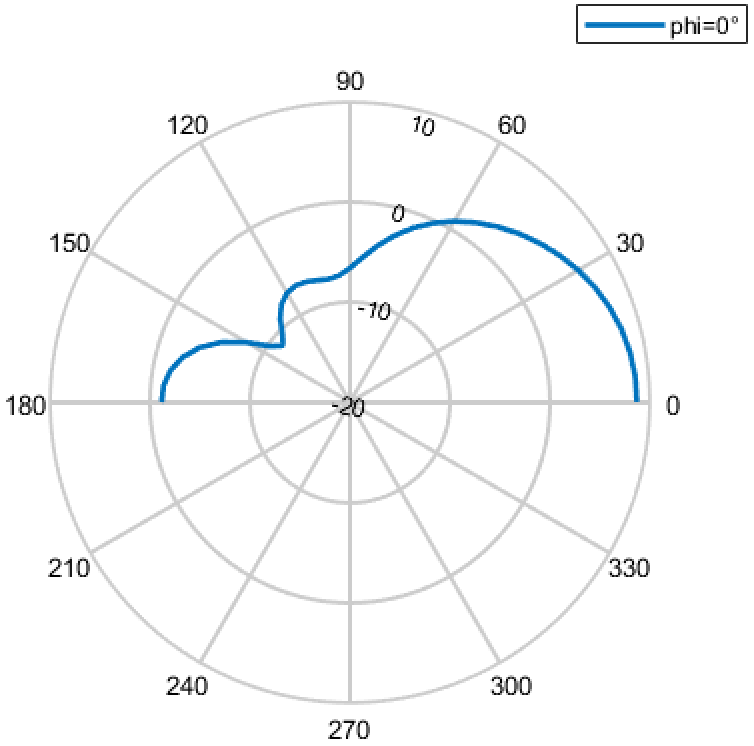
Plot 2-D phi slice of the antenna in rectangular coordinates.

```
figure;
patternCustom(MagE,theta,phi,'CoordinateSystem','rectangular',...
    'Slice','phi','SliceValue',0);
```



Plot 2-D phi slice of the antenna in polar coordinates.

```
figure;
patternCustom(MagE, theta, phi,'CoordinateSystem','polar',...
    'Slice','phi','SliceValue',0);
```



Input Arguments

`magE` — Magnitude of plotted quantity

real vector | matrix

Magnitude of plotted quantity, specified as one of the following:

- A *N*-by-1 real vector . *N* is the same size as the `phi` and `theta` angle vectors.
- A *M*-by-*R* matrix. The matrix should be the same size as `phi x theta`.

where `theta` and `phi` angles are in the spherical coordinate system specified as a vector.

Data quantities plotted include directivity, E-fields, H-fields, or power of an antenna or array object.

Data Types: `double`

`theta` — Theta angles in spherical coordinates

vector in degrees

Theta angles in spherical coordinates, specified as a vector in degrees. If azimuth and elevation values are given, theta angle values are 90 degrees minus elevation.

For more information, “Antenna Toolbox Coordinate System”.

Data Types: `double`

`phi` — Phi angles in spherical coordinates

vector in degrees

Phi angles in spherical coordinates, specified as a vector in degrees. If azimuth and elevation values are given, phi angle values are same as azimuth values.

For more information, “Antenna Toolbox Coordinate System”.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'CoordinateSystem'`,`'rectangular'`

`CoordinateSystem` — Coordinate system of radiation pattern

`'polar'` (default) | `'rectangular'`

Coordinate system of radiation pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of these values: `'polar'`, `'rectangular'`.

Example: `'CoordinateSystem'`,`'polar'`

Data Types: `char`

`Slice` — Plane to visualize 2-D data

`'theta'` | `'phi'`

Plane to visualize 2-D data, specified as a comma-separated pair consisting of `'Slice'` and `'theta'` or `'phi'`.

Example: `'Slice'`,`'phi'`

Data Types: `char`

`SliceValue` — Angle values for slice

scalar | vector

Angle values for slice, specified as a comma-separated pair consisting of `'SliceValue'` and a scalar or a vector.

`PatternOptions` — Parameter to change pattern plot properties

`PatternPlotOptions` object (default) | scalar

Parameter to change pattern plot properties, specified as the comma-separated pair consisting of `'PatternOptions'` and a `PatternPlotOptions` output. The properties that you can vary are:

- `Transparency`
- `MagnitudeScale`

Other properties used in the `'PatternOptions'` for the inset figure are ignored in `patternCustom`.

Example: `p = PatternPlotOptions('Transparency',0.1);` Create a pattern plot option with a transparency of 0.1. `helixdata = csvread('antennadata_test.csv',1,0);patternCustom(helixdata(:,3),helixdata(:,2),helixdata(:,1),'CoordinateSystem','rectangular','PatternOptions',p);` Use this pattern plot option to visualize the pattern of a helix antenna.

Data Types: `double`

Output Arguments

`hplot` — Lines or surfaces in figure window
object handle

Lines or surfaces in figure window, returned as object handle.

Version History

Introduced in R2016a

See Also

`pattern` | `EHfields` | `fieldsCustom` | `polarpattern`

Topics
"Antenna Toolbox Coordinate System"

msiread

Read MSI planet antenna file

Syntax

```
msiread(fname)
[horizontal] = msiread(fname)
[horizontal,vertical] = msiread(fname)
[horizontal,vertical,optional] = msiread(fname)
```

Description

`msiread(fname)` reads an MSI planet antenna file in `.pln`, or `.msi` formats.

`[horizontal] = msiread(fname)` reads the file and returns a structure containing horizontal gain data.

`[horizontal,vertical] = msiread(fname)` reads the file and returns structures containing horizontal and vertical gain data.

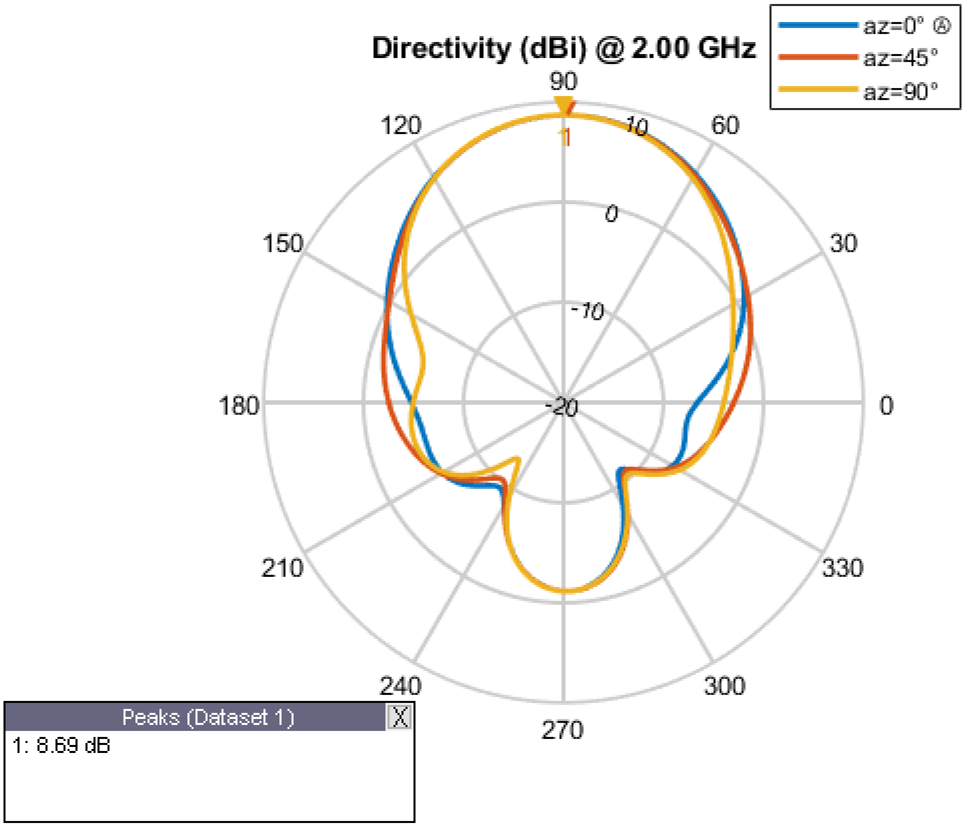
`[horizontal,vertical,optional] = msiread(fname)` reads the file and returns structures containing horizontal gain data, vertical gain data, and all additional data in the file.

Examples

Write and Read MSI Antenna Data File

Create a helix antenna and plot the elevation pattern at 2 GHz.

```
h = helix;
patternElevation(h,2e9,[0 45 90],'Elevation',0:1:360);
```



Write the elevation pattern of the helix antenna in an MSI Planet Antenna file.

```
msiwrite(h,2e9,'helix','Name','Helix Antenna Specifications')
```

The msiwrite function saves a file named `helix.pln` to the default MATLAB™ folder.

```
NAME Helix Antenna Specifications
FREQUENCY 2000.0
GAIN 8.74 dBi
HORIZONTAL 360
0.00 13.56
1.00 13.48
2.00 13.39
3.00 13.30
4.00 13.22
5.00 13.13
```

Read the MSI antenna data file created.

```
msiread helix.pln
```

```
ans = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: [360x1 double]
           Elevation: 0
           Frequency: 2.0000e+09
```

```
               Slice: 'Elevation'
```

Read Horizontal, Vertical and Optional Data from Antenna File

Read horizontal, vertical and optional data from the antenna data file **Test_file_demo.pln**.

```
[Horizontal,Vertical,Optional] = msiread('Test_file_demo.pln')
```

```
Horizontal = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBd'
             Azimuth: [360x1 double]
           Elevation: 0
           Frequency: 659000000
               Slice: 'Elevation'


Vertical = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBd'
             Azimuth: 0
           Elevation: [360x1 double]
           Frequency: 659000000
               Slice: 'Azimuth'


Optional = struct with fields:
             name: 'Sample.pln'
             make: 'Sample 4DR-16-2HW'
        frequency: 659000000
          h_width: 180
          v_width: 7.3000
    front_to_back: 34
             gain: [1x1 struct]
             tilt: 'MECHANICAL'
     polarization: 'POL_H'
          comment: 'Ch-45 0 deg dt'
     scaling_mode: 'AUTOMATIC'
```

Input Arguments

`fname` — Name of MSI file

character vector

Name of MSI file, specified as a character vector. The files must be a `.pln` or `.msi` format.

Output Arguments

horizontal — Horizontal gain data

structure

Horizontal gain data, returned as a structure containing the following fields:

- PhysicalQuantity — Quantity specified in the MSI file, returned as one of the values: 'E-field', 'H-field', 'directivity', 'power', 'powerdB', or 'Gain'.
- Magnitude — Magnitude values of the quantity specified in the MSI file, returned as a real vector of size *N*—by—1 where *N* is same size as theta and phi angles.
- Units — Units of the quantity specified in the MSI file, returned as one of the values: 'dBi', 'dB', 'V/m', 'watts', or 'dBd'.
- Azimuth — Azimuth angles specified in the MSI file, returned as a scalar or a vector in degrees.
- Elevation — Elevation angles specified in the MSI file, returned as a scalar or a vector in degrees.
- Frequency — Frequency specified in the MSI file, returned as a scalar or a vector in Hertz.
- Slice — Type of data set variation, returned as text. The variations are 'Azimuth' or 'Elevation'.

vertical — Vertical gain data

structure

Vertical gain data, returned as a structure containing the following fields:

- PhysicalQuantity — Quantity specified in the MSI file, returned as one of the values: 'E-field', 'H-field', 'directivity', 'power', 'powerdB', or 'Gain'.
- Magnitude — Magnitude values of the quantity specified in the MSI file, returned as a real vector of size *N*—by—1 where *N* is same size as theta and phi angles.
- Units — Units of the quantity specified in the MSI file, returned as one of the values: 'dBi', 'dB', 'V/m', 'watts', or 'dBd'.
- Azimuth — Azimuth angles specified in the MSI file, returned as a scalar or a vector in degrees.
- Elevation — Elevation angles specified in the MSI file, returned as a scalar or a vector in degrees.
- Frequency — Frequency specified in the MSI file, returned as a scalar or a vector in Hertz.
- Slice — Type of data set variation, returned as text. The variations are Azimuth or Elevation.

optional — Additional data

structure

Additional data, returned as a structure containing (but not limited to): Name, Make, Frequency, H_width, V_width, Front_to_back, Gain, Tilt, Polarization, Comment.

Version History

Introduced in R2016a

See Also

msiwrite

Topics
“Read, Visualize and Write MSI Planet Antenna Files”

msiwrite

Write data in MSI planet antenna file format

Syntax

```
msiwrite(fname,dataslice1,dataslice2)
msiwrite(fname,dataslice1,dataslice2,optional)
```

```
msiwrite(objname,frequency,fname)
msiwrite(objname,frequency,fname,Name,Value)
```

Description

`msiwrite(fname,dataslice1,dataslice2)` writes the data from structures `dataSlice1` and `dataSlice2` to an MSI planet antenna file called `fname`.

`msiwrite(fname,dataslice1,dataslice2,optional)` writes the data from structures `dataSlice1`, `dataSlice2`, and `optional` to an MSI planet antenna file called `fname`.

`msiwrite(objname,frequency,fname)` writes calculated data of an antenna or array object at a specified frequency to an MSI planet antenna file called `fname`.

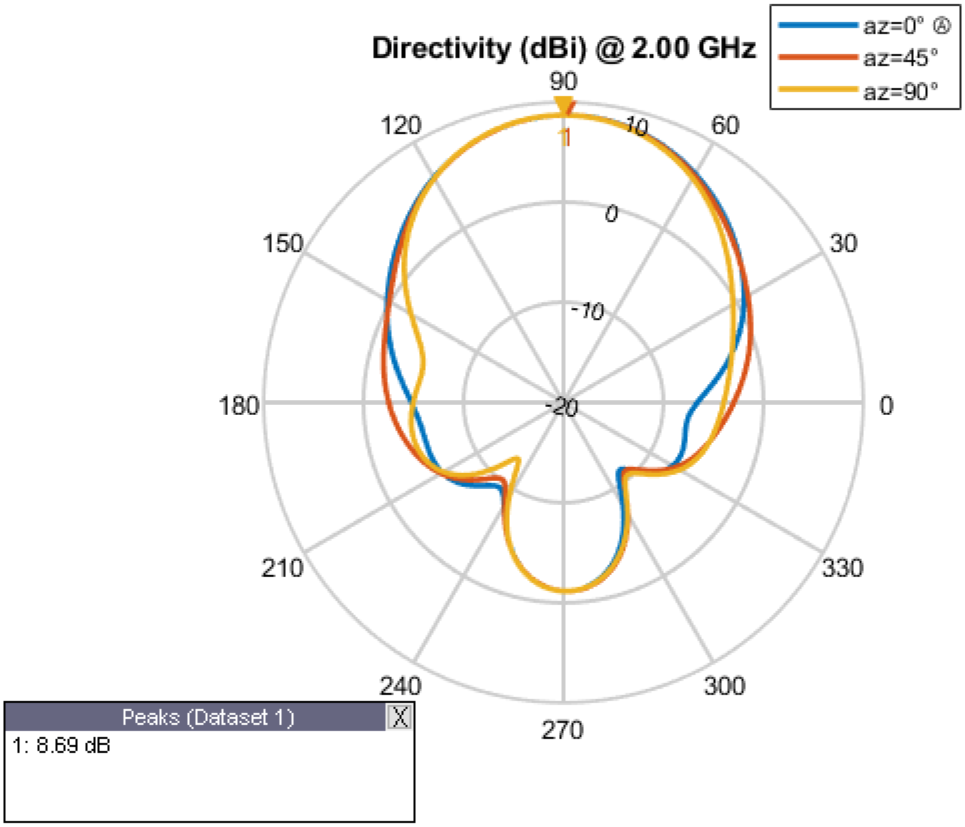
`msiwrite(objname,frequency,fname,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Write and Read MSI Antenna Data File

Create a helix antenna and plot the elevation pattern at 2 GHz.

```
h = helix;
patternElevation(h,2e9,[0 45 90],'Elevation',0:1:360);
```



Write the elevation pattern of the helix antenna in an MSI Planet Antenna file.

```
msiwrite(h,2e9,'helix','Name','Helix Antenna Specifications')
```

The msiwrite function saves a file named `helix.pln` to the default MATLAB™ folder.

```
NAME Helix Antenna Specifications
FREQUENCY 2000.0
GAIN 8.74 dBi
HORIZONTAL 360
0.00 13.56
1.00 13.48
2.00 13.39
3.00 13.30
4.00 13.22
5.00 13.13
```

Read the MSI antenna data file created.

```
msiread helix.pln
```

```
ans = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: [360x1 double]
           Elevation: 0
           Frequency: 2.0000e+09
```

```
            Slice: 'Elevation'
```

Input Arguments

fname — Name of MSI file

`.pln` (default) | character vector

Name of MSI file, specified as a character vector. By default, `msiwrite` writes the MSI planet antenna file that has a `.pln` format.

dataslice1 — Horizontal or vertical gain data

structure

Horizontal or vertical gain data, specified as a structure containing the following fields:

- `PhysicalQuantity` — Measured quantity in the MSI file: `E-field`, `H-field`, `directivity`, `power`, `powerdB`, or, `gain`.
- `Magnitude` — Magnitude values of the measured quantity.
- `Units` — Units of the measured quantity.
- `Azimuth` — Azimuth angles.
- `Elevation` — Elevation angles.
- `Frequency` — Frequency of operation.
- `Slice` — Type of data set variation: `Azimuth`, or `Elevation`.

dataslice2 — Horizontal or vertical gain data

structure

Horizontal or vertical gain data, specified as a structure containing the following fields:

- `PhysicalQuantity` — Measured quantity in the MSI file: `E-field`, `H-field`, `directivity`, `power`, `powerdB`, or, `gain`.
- `Magnitude` — Magnitude values of the measure quantity.
- `Units` — Units of the measured quantity.
- `Azimuth` — Azimuth angles.
- `Elevation` — Elevation angles.
- `Frequency` — Frequency of operation.
- `Slice` — Type of data set variation: `Azimuth`, or `Elevation`.

optional — Additional data

structure

Additional data, specified as a structure containing the following fields: `Name`, `Make`, `Frequency`, `H_width`, `V_width`, `Front_to_back`, `Gain`, `Tilt`, `Polarization`, `Comment`.

objname — Antenna or array object

antenna or array

Antenna or array object, specified as an antenna or array.

`frequency` — Frequency of operation of antenna or array object

positive numeric scalar

Frequency of operation of antenna or array object, specified as a positive numeric scalar.

Name-Value Pair Arguments

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

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

Example: `'Comment'`, `'horn antenna'`

`Name` — Title of file

character vector

Title of file in the first line, specified as the comma-separated pair consisting of `'Name'` and a character vector.

Example: `'Name'`, `'Designed Helix Antenna in MATLAB'`

Data Types: `char`

`Comment` — Comments about antenna or array data file

character array

Comments about an antenna or array data file, specified as the comma-separated pair consisting of `'Comment'` and a character array.

Example: `'Comment'`, `'This antenna is for space simulations.'`

Data Types: `char`

Version History

Introduced in R2016a

See Also

`msiread`

Topics

"Read, Visualize and Write MSI Planet Antenna Files"

dielectric

Dielectric material for use as substrate

Syntax

```
d = dielectric(material)
d = dielectric(Name,Value)
```

Description

`d = dielectric(material)` returns dielectric materials for use as a substrate in antenna elements.

`d = dielectric(Name,Value)` returns dielectric materials, based on the properties specified by one or more `Name,Value` pair arguments.

Examples

PIFA Antenna with Dielectric Substrate

Use a Teflon dielectric material as a substrate for a PIFA antenna. View the antenna.

```
d = dielectric('Teflon')
```

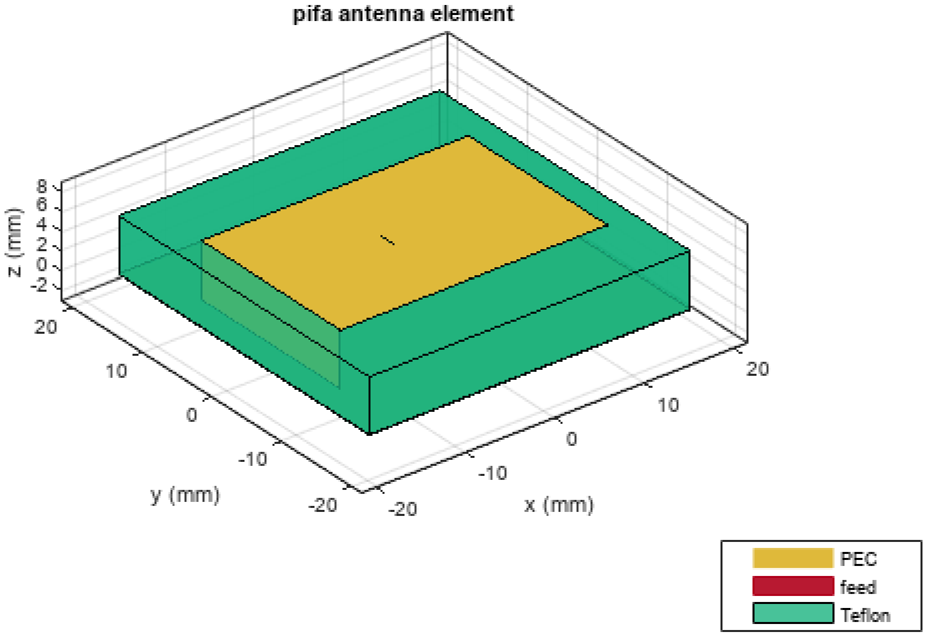
```
d = 
  dielectric with properties:

           Name: 'Teflon'
       EpsilonR: 2.1000
    LossTangent: 2.0000e-04
      Thickness: 0.0060

For more materials see catalog
```

```
p = pifa('Height',0.0060,'Substrate',d)
```

```
p = 
  pifa with properties:

               Length: 0.0300
                Width: 0.0200
               Height: 0.0060
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.0360
     GroundPlaneWidth: 0.0360
    PatchCenterOffset: [0 0]
        ShortPinWidth: 0.0200
           FeedOffset: [-0.0020 0]
            Conductor: [1x1 metal]
                 Tilt: 0
```

```
              TiltAxis: [1 0 0]
                  Load: [1x1 lumpedElement]
```

```
show(p)
```



Custom Dielectric Properties

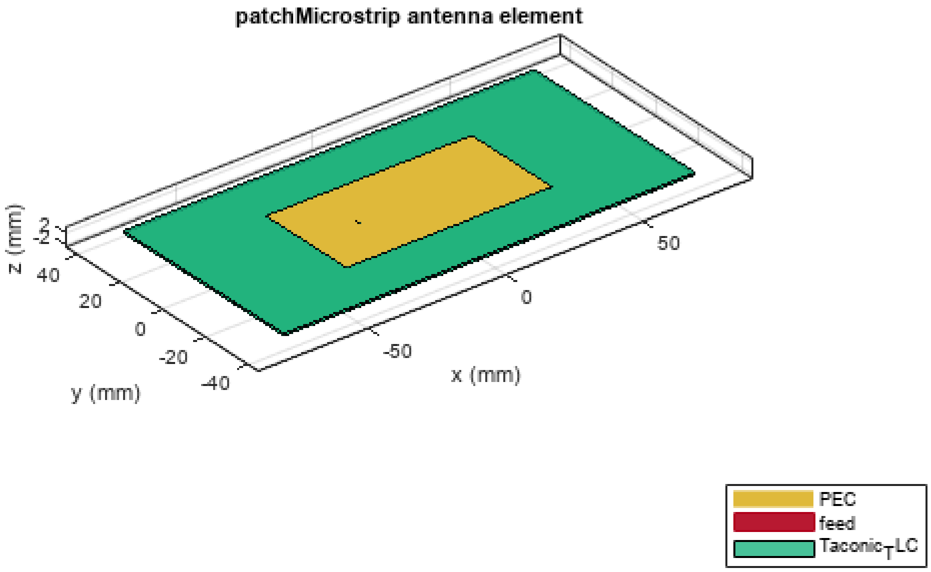
Create a patch microstrip antenna using a substrate with a relative permittivity of 2.70, a loss tangent of 0.002 and a thickness of 0.0008 m. View the antenna.

```
t = dielectric('Name','Taconic_TLC','EpsilonR',2.70,'LossTangent',0.002,...
    'Thickness',0.0008);
p = patchMicrostrip('Height',0.0008,'Substrate',t)
```

```
p = 
  patchMicrostrip with properties:

               Length: 0.0750
                Width: 0.0375
               Height: 8.0000e-04
            Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1500
     GroundPlaneWidth: 0.0750
    PatchCenterOffset: [0 0]
```

```
        FeedOffset: [-0.0187 0]
         Conductor: [1x1 metal]
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1x1 lumpedElement]
```

```
show(p)
```



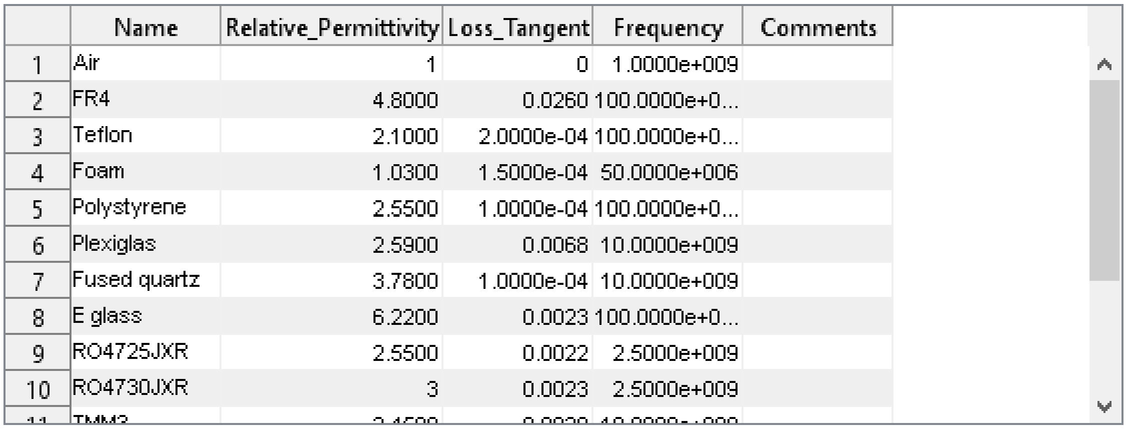
Patch Antenna with Air Gap between Groundplane and Dielectric

Create a microstrip patch antenna.

```
p = patchMicrostrip;
```

For property values of air and teflon dielectrics, refer Dielectric Catalog.

```
openDielectricCatalog
```



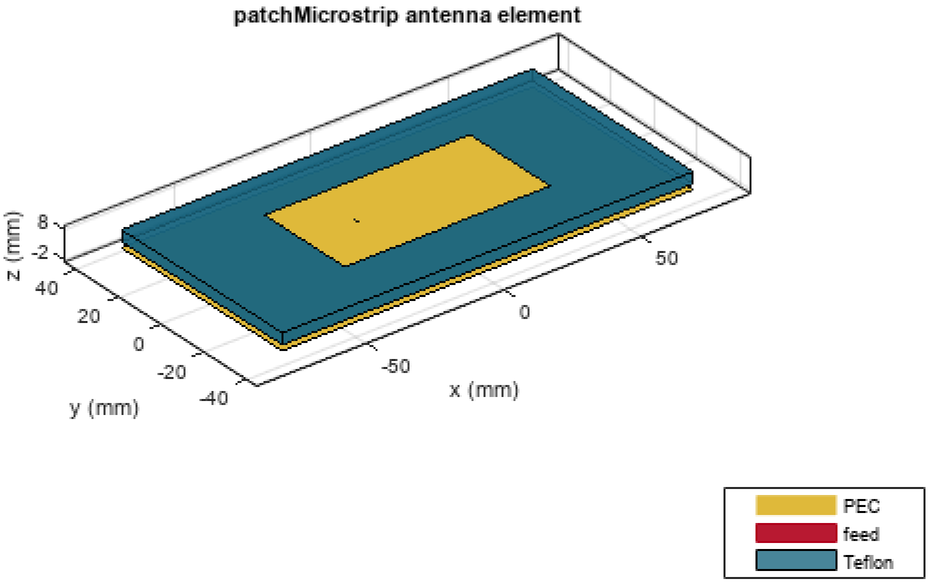
	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+009	
2	FR4	4.8000	0.0260	100.0000e+0...	
3	Teflon	2.1000	2.0000e-04	100.0000e+0...	
4	Foam	1.0300	1.5000e-04	50.0000e+006	
5	Polystyrene	2.5500	1.0000e-04	100.0000e+0...	
6	Plexiglas	2.5900	0.0068	10.0000e+009	
7	Fused quartz	3.7800	1.0000e-04	10.0000e+009	
8	E glass	6.2200	0.0023	100.0000e+0...	
9	RO4725JXR	2.5500	0.0022	2.5000e+009	
10	RO4730JXR	3	0.0023	2.5000e+009	
11	TMM3	3.4500	0.0020	10.0000e+009	

Use Teflon as a dielectric substrate. There is an air gap between the patch groundplane and the dielectric.

```
sub = dielectric('Name',{'Air','Teflon'},'EpsilonR',[1 2.1],...
    'Thickness',[.002 .004],'LossTangent',[0 2e-04]);
```

Add the substrate to the patch antenna.

```
p.Substrate = sub;
figure
show(p)
```



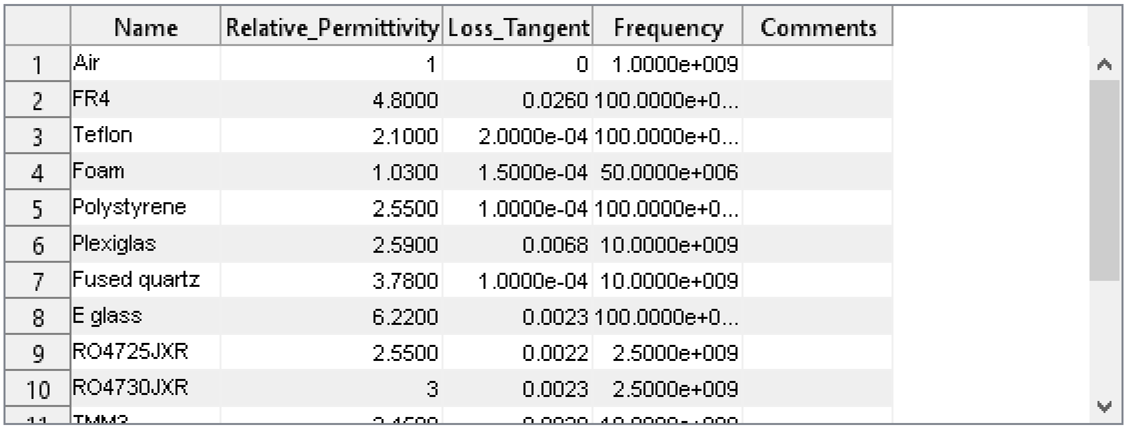
Three Layer Dielectric Substrate between Patch and Ground Plane

Create a microstrip patch antenna.

```
p = patchMicrostrip;
```

For dielectric properties, use the Dielectric Catalog.

```
openDielectricCatalog
```



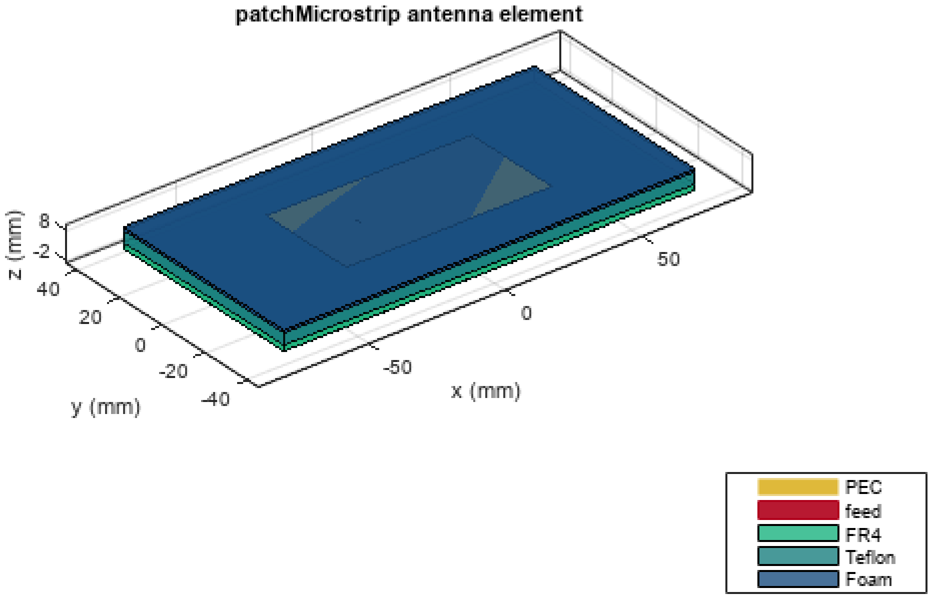
	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+009	
2	FR4	4.8000	0.0260	100.0000e+0...	
3	Teflon	2.1000	2.0000e-04	100.0000e+0...	
4	Foam	1.0300	1.5000e-04	50.0000e+006	
5	Polystyrene	2.5500	1.0000e-04	100.0000e+0...	
6	Plexiglas	2.5900	0.0068	10.0000e+009	
7	Fused quartz	3.7800	1.0000e-04	10.0000e+009	
8	E glass	6.2200	0.0023	100.0000e+0...	
9	RO4725JXR	2.5500	0.0022	2.5000e+009	
10	RO4730JXR	3	0.0023	2.5000e+009	
11	TMM3	3.4500	0.0020	10.0000e+009	

Use FR4, Teflon and Foam as the three layers of the substrate.

```
sub = dielectric('Name',{'FR4','Teflon','Foam'},'EpsilonR',...
    [4.80 2.10 1.03],'Thickness',[0.002 0.004 0.001],...
    'LossTangent',[0.0260 2e-04 1.5e-04]);
```

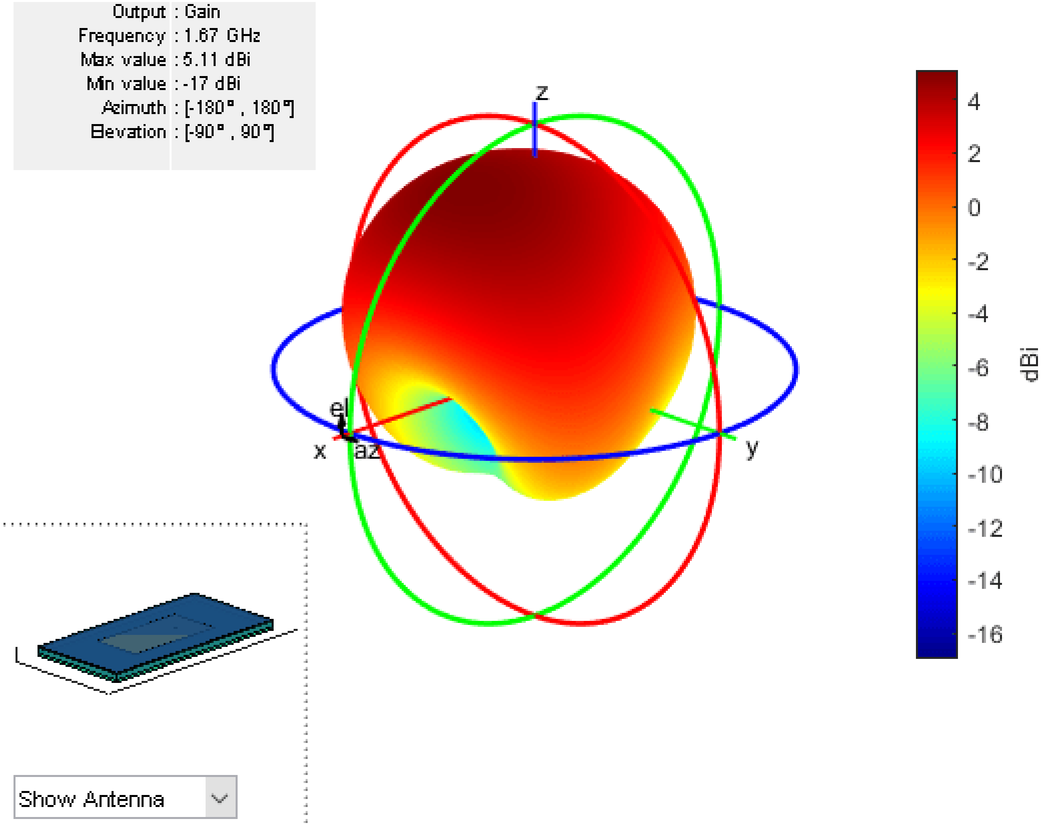
Add the three layer substrate to the patch antenna.

```
p.Substrate = sub;
figure
show(p)
```



Plot the radiation pattern of the antenna.

```
figure
pattern(p,1.67e9)
```



```
Output : Gain
Frequency : 1.67 GHz
Max value : 5.11 dBi
Min value : -17 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Infinite Reflector Backed Dielectric Substrate Antenna

Design a dipole antenna backed by a dielectric substrate and an infinite reflector.

Create a dipole antenna of length, 0.15 m, and width, 0.015 m.

```
d = dipole('Length',0.15,'Width',0.015, 'Tilt',90,'TiltAxis',[0 1 0]);
```

Create a reflector using the dipole antenna as an exciter and the dielectric, `teflon` as the substrate.

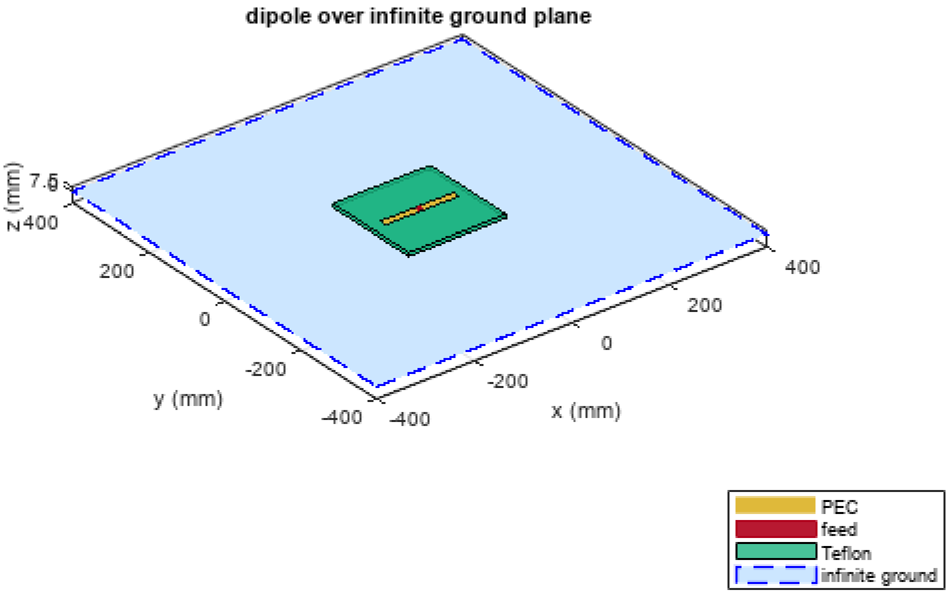
```
t = dielectric('Teflon')
```

```
t = 
  dielectric with properties:

           Name: 'Teflon'
       EpsilonR: 2.1000
    LossTangent: 2.0000e-04
      Thickness: 0.0060

For more materials see catalog
```

```
rf = reflector('Exciter',d,'Spacing',7.5e-3,'Substrate',t);
```

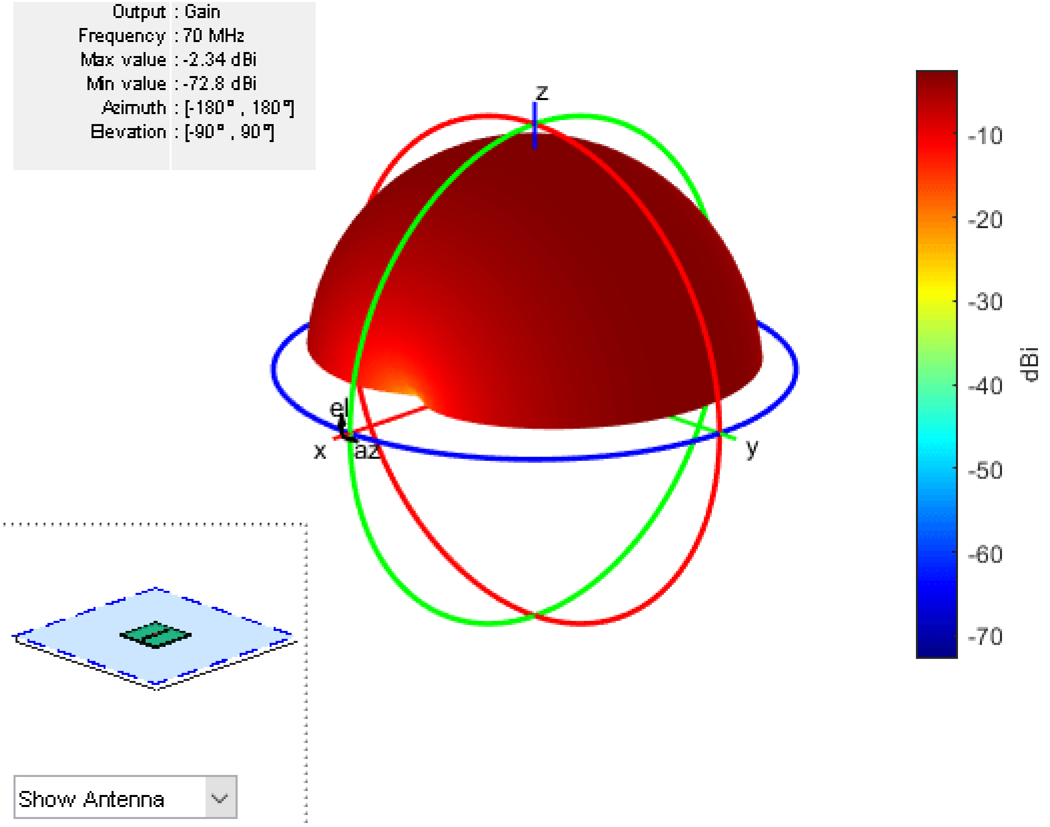
Set the groundplane length of the reflector to `inf`. View the structure.

```
rf.GroundPlaneLength = inf;
show(rf)
```



Calculate the radiation pattern of the antenna at 70 MHz.

```
pattern(rf,70e6)
```



Antenna On Dielectric Substrate - Compare Gain Values

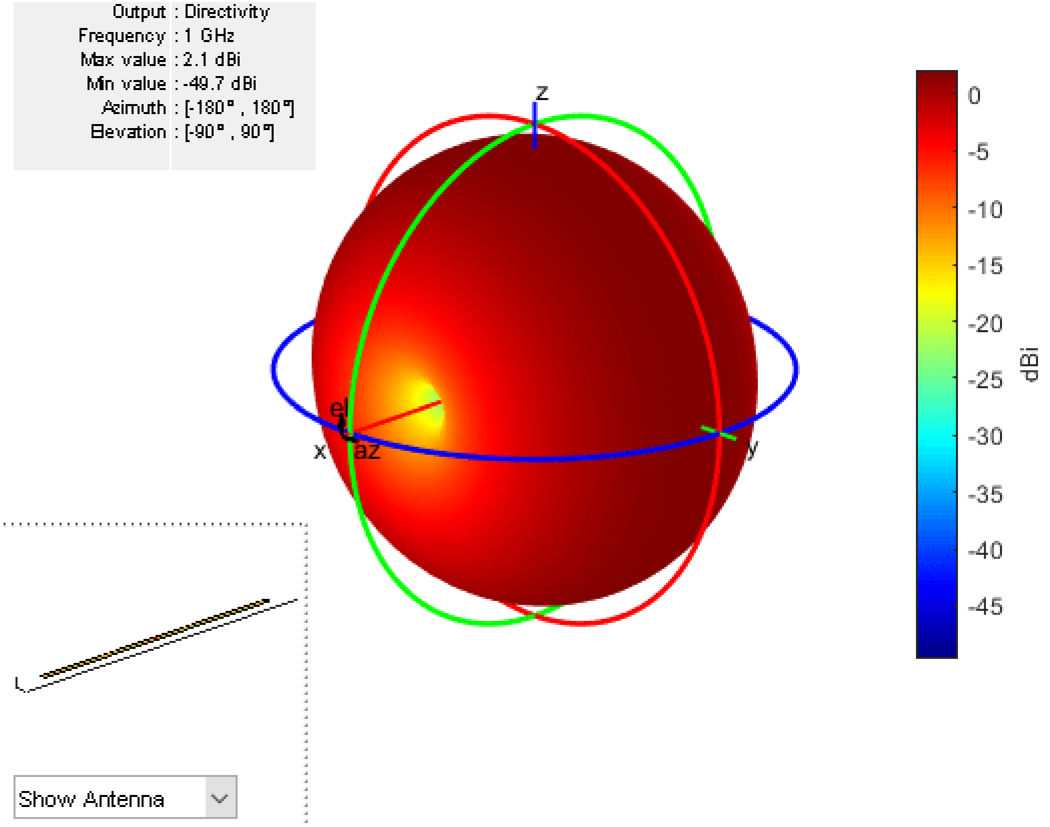
Compare the gain values of a dipole antenna in free space and dipole antenna on a substrate.

Design a dipole antenna at a frequency of 1 GHz.

```
d = design(dipole,1e9);
l_by_w = d.Length/d.Width;
d.Tilt = 90;
d.TiltAxis = [0 1 0];
```

Plot the radiation pattern of the dipole in free space at 1 GHz.

```
figure
pattern(d,1e9);
```



Use FR4 as the dielectric substrate.

```
t = dielectric('FR4')
```

```
t = 
  dielectric with properties:

         Name: 'FR4'
     EpsilonR: 4.8000
  LossTangent: 0.0260
    Thickness: 0.0060

For more materials see catalog
```

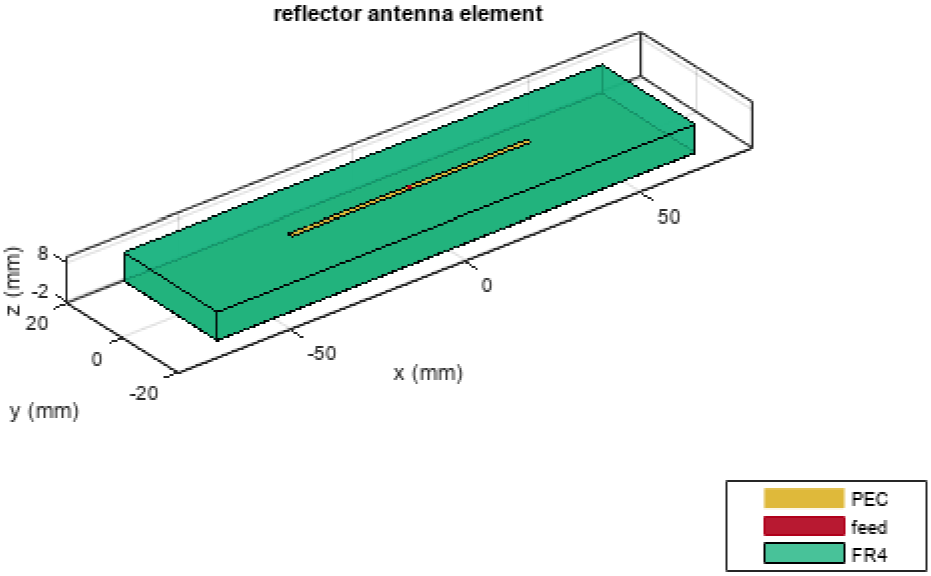
```
eps_r = t.EpsilonR;
lambda_0 = physconst('lightspeed')/1e9;
lambda_d = lambda_0/sqrt(eps_r);
```

Adjust the length of the dipole based on the wavelength.

```
d.Length = lambda_d/2;
d.Width = d.Length/l_by_w;
```

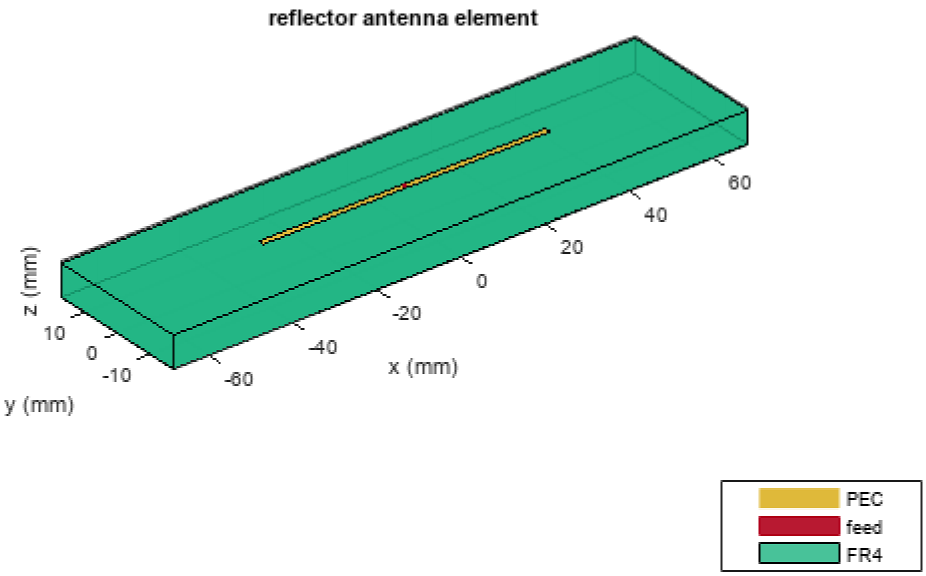
Design a reflector at 1 GHz with the dipole as the exciter and FR4 as the substrate.

```
rf = reflector('Exciter',d,'Spacing',7.5e-3,'Substrate',t);
rf.GroundPlaneLength = lambda_d;
rf.GroundPlaneWidth = lambda_d/4;
figure
show(rf)
```



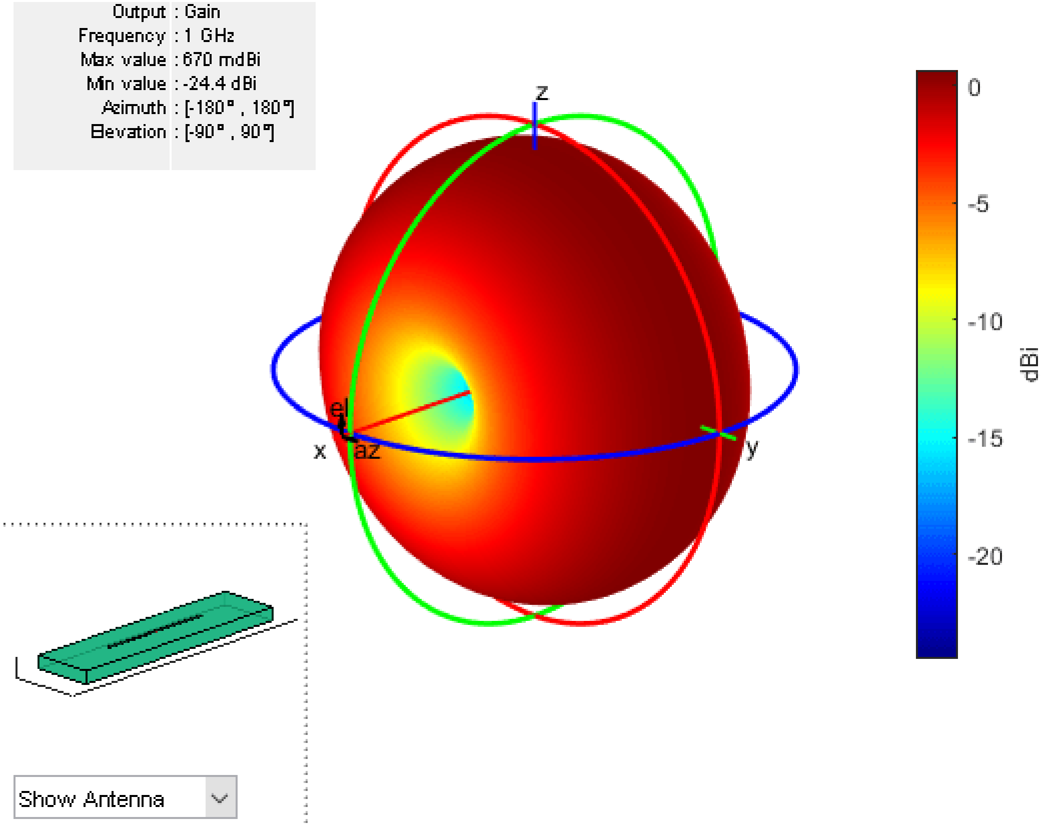
Remove the groundplane for plotting the gain of the dipole on the substrate.

```
rf.GroundPlaneLength = 0;
show(rf)
```



Plot the radiation pattern of the dipole on the substrate at 1 GHz.

```
figure
pattern(rf,1e9);
```



```
   Output : Gain
Frequency : 1 GHz
Max value : 670 mdBi
Min value : -24.4 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Compare the gain values.

- Gain of the dipole in free space = 2.11 dBi
- Gain of the dipole on substrate = 1.93 dBi

Input Arguments

`material` — Material from dielectric catalog

`'Air'` (default) | single dielectric | multiple dielectrics

Material from the dielectric catalog, specified as one or more dielectrics from the `DielectricCatalog` object with predefined properties. You can specify multiple dielectric layers and create an array of dielectrics.

Example: `'FR4'`

Example: `'FR4','Teflon'`

Example: [dielectric(`'FR4'`) dielectric(`'Teflon'`)]

Data Types: `char`

Name-Value Pair Arguments

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

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

Example: `'Name'`,`'Air'`

`Name` — Name of dielectric material
character vector

Name of the dielectric material you want to specify in the output, specified as the comma-separated pair consisting of `'Name'` and a character vector.

Example: `'Name'`,`'Taconic_TLC'`

Data Types: `char`

`EpsilonR` — Relative permittivity of dielectric material
`1` | vector

Relative permittivity of the dielectric material, specified as the comma-separated pair consisting of `'EpsilonR'` and vector.

Example: `'EpsilonR'`,`4.8000`

Data Types: `double`

`LossTangent` — Loss in dielectric material
`0` (default) | vector

Loss in the dielectric material, specified as the comma-separated pair consisting of `'LossTangent'` and vector.

Note In Antenna Toolbox, the upper limit to loss tangent value is 0.03.

Example: `'LossTangent'`,`0.0260`

Data Types: `double`

`Thickness` — Thickness of dielectric material
`0.0060` (default) | vector in meters

Thickness of the dielectric material along default z-axis, specified as the comma-separated pair consisting of `'Thickness'` and vector in meters. This property applies only when you call the function with no output arguments.

Example: `'Thickness'`, `0.05`

Data Types: `double`

Output Arguments

d — Dielectric material
object

Dielectric material, returned as an object. You can use the dielectric material object to add dielectric material to an antenna.

Version History

Introduced in R2016a

See Also

`DielectricCatalog`

Topics

"Antenna Toolbox Limitations"

DielectricCatalog

Catalog of dielectric materials

Syntax

```
dc = DielectricCatalog
```

Description

`dc = DielectricCatalog` creates an object for the dielectric catalog.

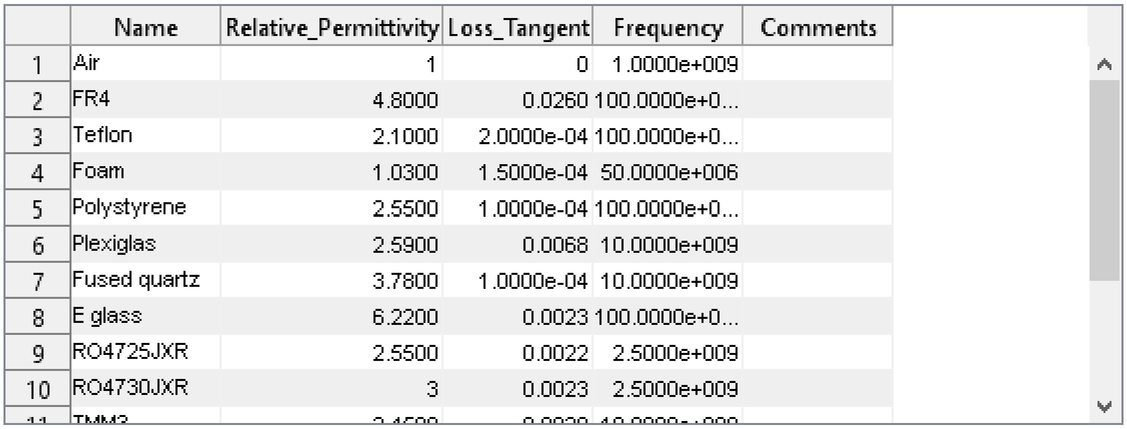
- To open the dielectric catalog, use `open(dc)`
- To know the properties of a dielectric material from the dielectric catalog, use `s = find(dc, name)`.

Examples

Use Dielectric Catalog Element in Cavity

Open the dielectric catalog.

```
dc = DielectricCatalog;
open(dc)
```



	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+009	
2	FR4	4.8000	0.0260	100.0000e+0...	
3	Teflon	2.1000	2.0000e-04	100.0000e+0...	
4	Foam	1.0300	1.5000e-04	50.0000e+006	
5	Polystyrene	2.5500	1.0000e-04	100.0000e+0...	
6	Plexiglas	2.5900	0.0068	10.0000e+009	
7	Fused quartz	3.7800	1.0000e-04	10.0000e+009	
8	E glass	6.2200	0.0023	100.0000e+0...	
9	RO4725JXR	2.5500	0.0022	2.5000e+009	
10	RO4730JXR	3	0.0023	2.5000e+009	
11	TMM3	3.4500	0.0020	10.0000e+009	

List the properties of the dielectric material `Foam`.

```
s = find(dc,'Foam')
```

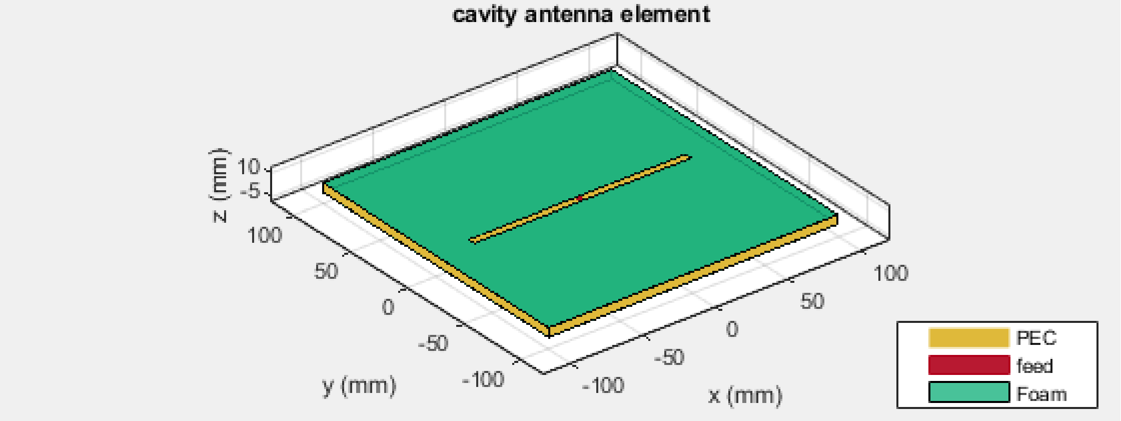
```
s = struct with fields:
                     Name: 'Foam'
    Relative_Permittivity: 1.0300
             Loss_Tangent: 1.5000e-04
                Frequency: 50000000
                 Comments: ''
```

Use the material Foam as a dielectric in a cavity antenna of height and spacing, 0.0060 m.

```
d = dielectric('Foam');
c = cavity('Height',0.0060,'Spacing',0.0060,'Substrate',d)
```

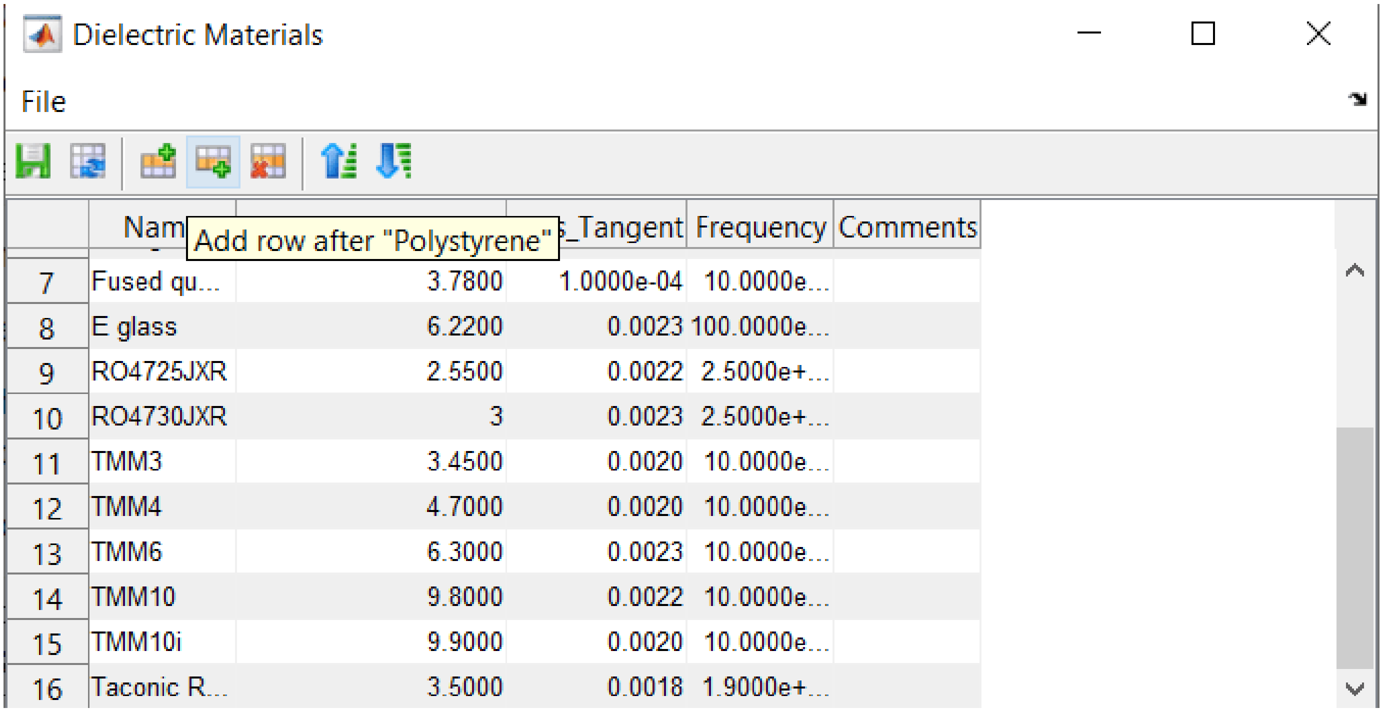
```
c = 
  cavity with properties:

            Exciter: [1x1 dipole]
          Substrate: [1x1 dielectric]
             Length: 0.2000
              Width: 0.2000
             Height: 0.0060
            Spacing: 0.0060
    EnableProbeFeed: 0
          Conductor: [1x1 metal]
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show (c)
```



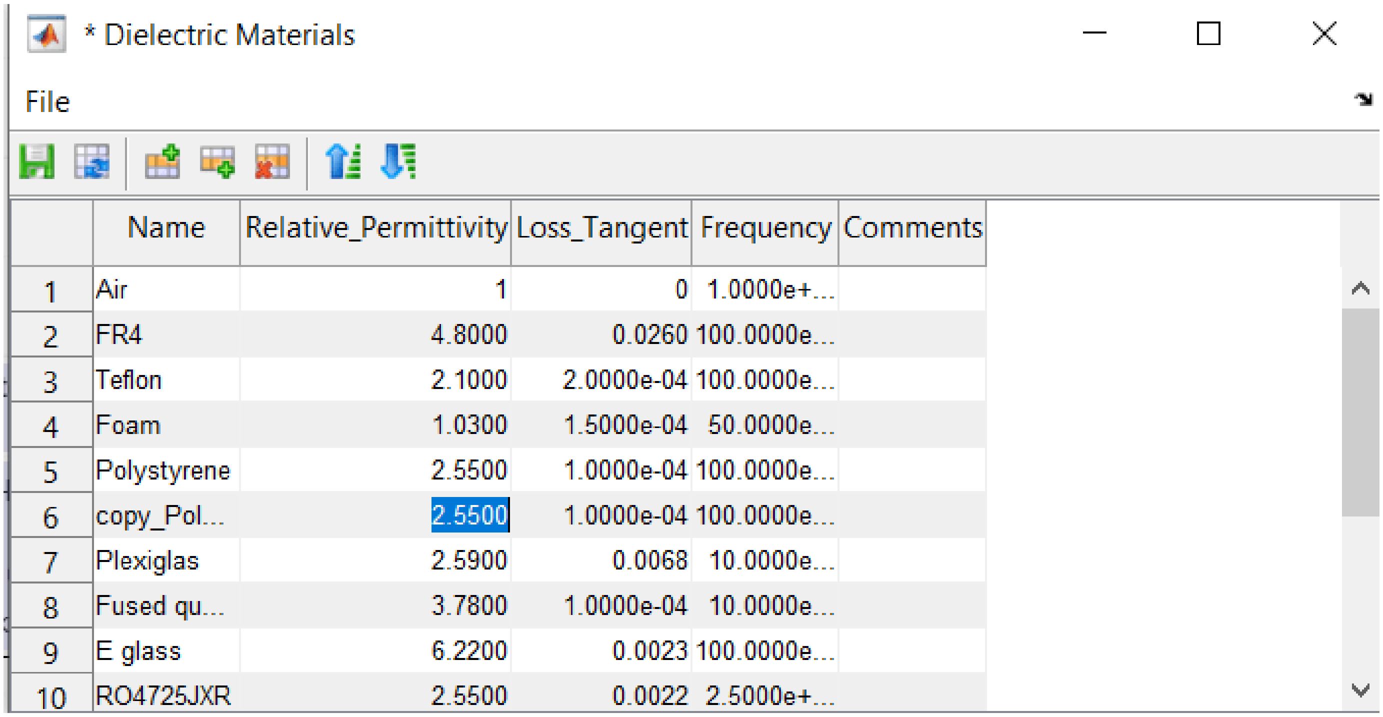
Addition of Custom `dielectric` Material in Dielectric Catalog

Open the dielectric catalog by using `openDielectric Catalog` function.To add a new material to the dielectric catalog, click on the row addition icon.



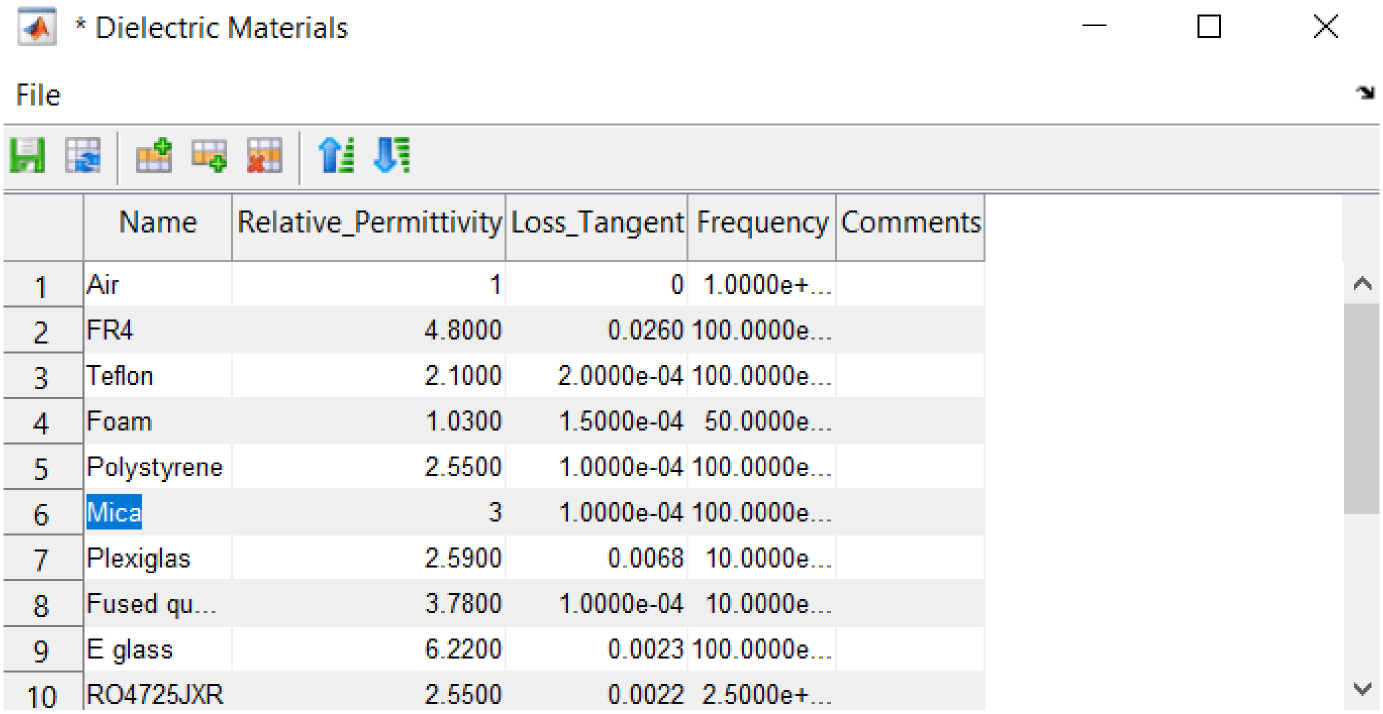
	Nam		_Tangent	Frequency	Comments
7	Fused qu...	3.7800	1.0000e-04	10.0000e...	
8	E glass	6.2200	0.0023	100.0000e...	
9	RO4725JXR	2.5500	0.0022	2.5000e+...	
10	RO4730JXR	3	0.0023	2.5000e+...	
11	TMM3	3.4500	0.0020	10.0000e...	
12	TMM4	4.7000	0.0020	10.0000e...	
13	TMM6	6.3000	0.0023	10.0000e...	
14	TMM10	9.8000	0.0022	10.0000e...	
15	TMM10i	9.9000	0.0020	10.0000e...	
16	Taconic R...	3.5000	0.0018	1.9000e+...	

A duplicate record of the dielectric material appears below the selected row.



	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+...	
2	FR4	4.8000	0.0260	100.0000e...	
3	Teflon	2.1000	2.0000e-04	100.0000e...	
4	Foam	1.0300	1.5000e-04	50.0000e...	
5	Polystyrene	2.5500	1.0000e-04	100.0000e...	
6	copy_Pol...	2.5500	1.0000e-04	100.0000e...	
7	Plexiglas	2.5900	0.0068	10.0000e...	
8	Fused qu...	3.7800	1.0000e-04	10.0000e...	
9	E glass	6.2200	0.0023	100.0000e...	
10	RO4725JXR	2.5500	0.0022	2.5000e+...	

You can change the record by setting a desired name, permittivity, frequency and thickness of the material to customize.



	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+...	
2	FR4	4.8000	0.0260	100.0000e...	
3	Teflon	2.1000	2.0000e-04	100.0000e...	
4	Foam	1.0300	1.5000e-04	50.0000e...	
5	Polystyrene	2.5500	1.0000e-04	100.0000e...	
6	Mica	3	1.0000e-04	100.0000e...	
7	Plexiglas	2.5900	0.0068	10.0000e...	
8	Fused qu...	3.7800	1.0000e-04	10.0000e...	
9	E glass	6.2200	0.0023	100.0000e...	
10	RO4725JXR	2.5500	0.0022	2.5000e+...	

You can access the new added dielectric material by using `dielectric` object.

Input Arguments

name — Name of dielectric material
`'Air'` (default) | character vector

Name of a dielectric material from the dielectric catalog, specified as a character vector.

Example: `'FR4'`

Data Types: `char`

dc — Dielectric catalog
object

Dielectric catalog, specified as an object.

Data Types: `char`

Output Arguments

dc — Dielectric catalog
object

Dielectric catalog, returned as an object.

s — Parameters of dielectric material
structure

Parameters of a dielectric material from the dielectric catalog, returned as a structure.

Version History

Introduced in R2016a

See Also

`dielectric`

hornangle2size

Equivalent flare width and flare height from flare angles

Syntax

`[flarewidth,flareheight]= hornangle2size(width,height,flarelength,angleE, angleH)`

Description

`[flarewidth,flareheight]= hornangle2size(width,height,flarelength,angleE, angleH)` calculates the equivalent `flarewidth` and `flareheight` for a rectangular horn antenna from its flare angles, `angleE`, and `angleH`.

Examples

Calculate Flare Width and Flare Height of Horn Antenna

Calculate the flare width and the flare height of a horn antenna with

- Width of the waveguide = 0.0229 m
- Height of the waveguide = 0.0102 m
- Flare length of the horn = 0.2729 m
- Flare angle in the E-plane = 12.2442 degrees
- Flare angle in the H-plane = 14.4712 degrees

```
width = 0.0229;
height = 0.0102;
flarelength = 0.2729;
angleE = 12.2442;
angleH = 14.4712;
[flarewidth,flareheight] = hornangle2size(width,height,flarelength,...
                           angleE,angleH)
```

```
flarewidth = 0.1638
```

```
flareheight = 0.1286
```

Input Arguments

`width` — Rectangular waveguide width
scalar in meters

Rectangular waveguide width, specified a scalar in meters.

Data Types: `double`

`height` — Rectangular waveguide height

scalar in meters

Rectangular waveguide height, specified a scalar in meters.

Data Types: `double`

`flarelength` — Flare length of horn

scalar in meters

Flare length of horn, specified as a scalar in meters.

Data Types: `double`

`angleE` — Flare angle in E-plane

scalar in degrees

Flare angle in E-plane of the horn, specified as a scalar in degrees.

Data Types: `double`

`angleH` — Flare angle in H-plane

scalar in meters

Flare angle in H-plane of the horn, specified as a scalar in degrees.

Data Types: `double`

Output Arguments

`flarewidth` — Flare width of horn

scalar in meters

Flare width of horn, returned as a scalar in meters.

Data Types: `double`

`flareheight` — Flare height of horn

scalar in meters

Flare height of horn, returned as a scalar in meters.

Data Types: `double`

Version History

Introduced in R2016a

See Also

`horn`

add

Add data to polar plot

Syntax

```
add(p,d)
add(p,angle,magnitude)
```

Description

`add(p,d)` adds new antenna data to the polar plot, `p` based on the real amplitude values, `data`.

`add(p,angle,magnitude)` adds data sets of `angle` vectors and corresponding `magnitude` matrices to polar plot `p`.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`data` — Antenna or array data
real length-*M* vector | real *M*-by-*N* matrix | real *N-D* array | complex vector or matrix

Antenna or array data, specified as one of the following:

- A real length-*M* vector, where *M* contains the magnitude values with angles assumed to be $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees.
- A real *M*-by-*N* matrix, where *M* contains the magnitude values and *N* contains the independent data sets. Each column in the matrix has angles taken from the vector $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees. The set of each angle can vary for each column.
- A real *N-D* array, where *N* is the number of dimensions. Arrays with dimensions `2` and greater are independent data sets.
- A complex vector or matrix, where `data` contains Cartesian coordinates (*(x,y)* of each point. *x* contains the real part of `data` and *y* contains the imaginary part of `data`.

When data is in a logarithmic form such as dB, magnitude values can be negative. In this case,`polarpattern` plots the lowest magnitude values at the origin of the polar plot and highest magnitude values at the maximum radius.

`angle` — Set of angles
vector in degrees

Set of angles, specified as a vector in degrees.

`magnitude` — Set of magnitude values
vector | matrix

Set of magnitude values, specified as a vector or a matrix. For a matrix of magnitude values, each column is an independent set of magnitude values and corresponds to the same set of angles.

Examples

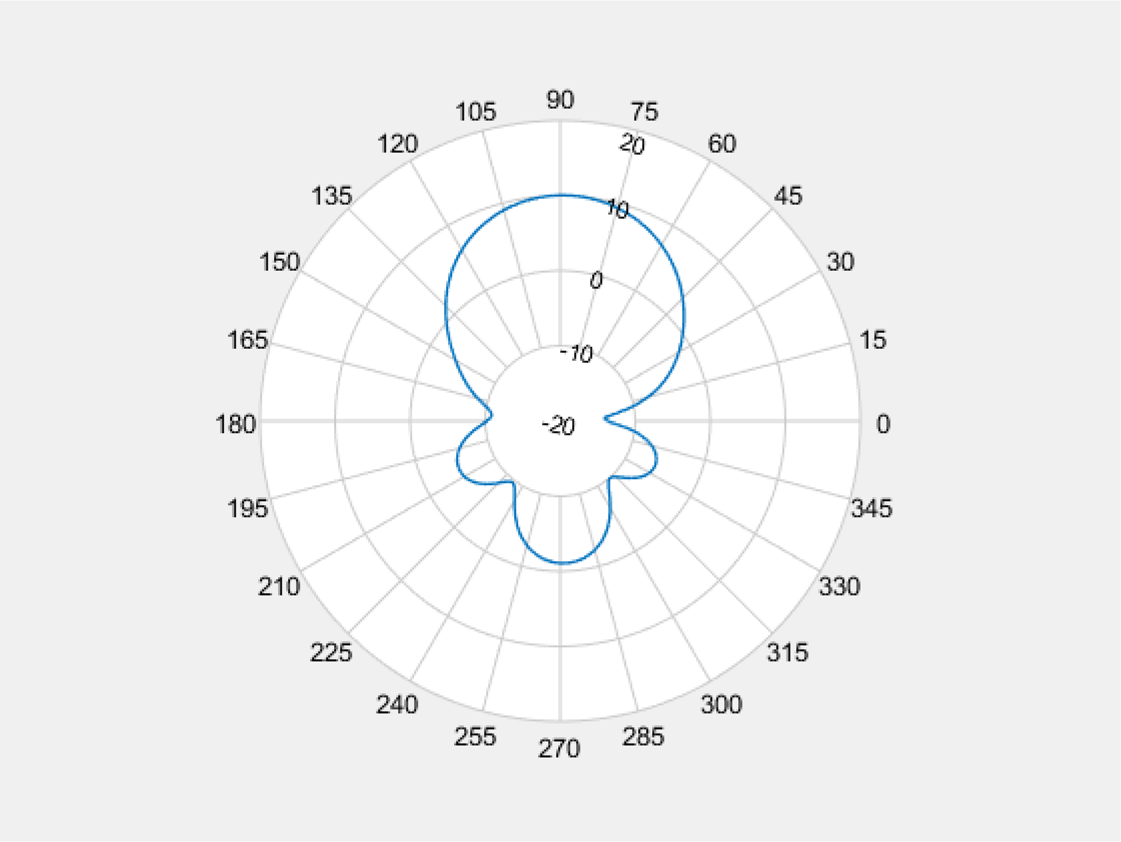
Add Data To Polar Plot

Create a helix antenna that has 28 mm radius, 1.2 mm width, and 4 turns. Calculate the directivity of the antenna at 1.8 GHz.

```
hx = helix('Radius',28e-3,'Width',1.2e-3,'Turns',4);
H = pattern(hx, 1.8e9,0,0:1:360);
```

Plot the polar pattern.

```
P = polarpattern(H);
```

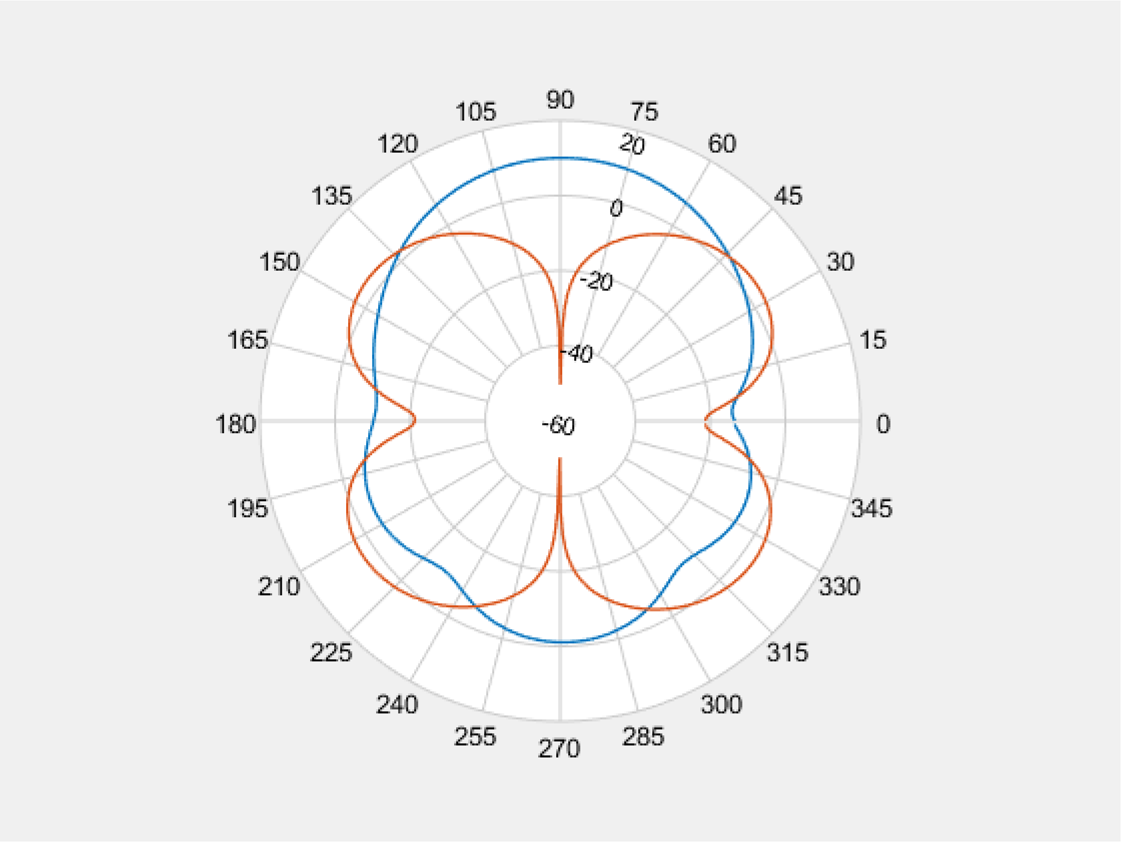


Create a dipole antenna and calculate the directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

Add the directivity of the dipole to the existing polar plot of helix antenna.

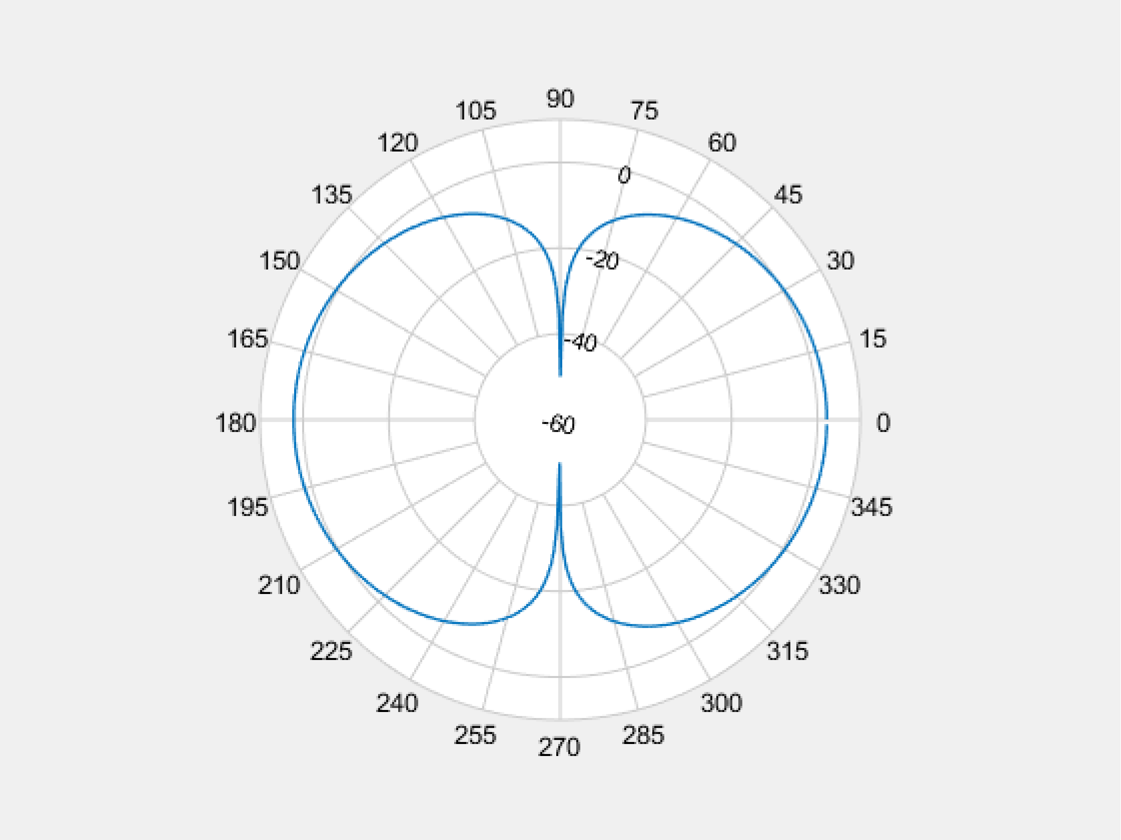
```
add(P,D);
```



Add Angle and Magnitude Data to Polar Pattern

Create a dipole and plot the polar pattern of its directivity at a frequency of 75 MHz.

```
d = dipole;
D = pattern(d,75e6,0,0:1:360);
P = polarpattern(D);
```



Create a cavity antenna. Calculate the directivity of the antenna at 1 GHz. Write the directivity of the antenna to `cavity.pln` using the `msiwrite` function.

```
c = cavity;
msiwrite(c,1e9,'cavity','Name','Cavity Antenna Specifications');
```

Read the data from `cavity.pln` to `Horizontal`, `Vertical`, and `Optional` structures using the `msiread` function.

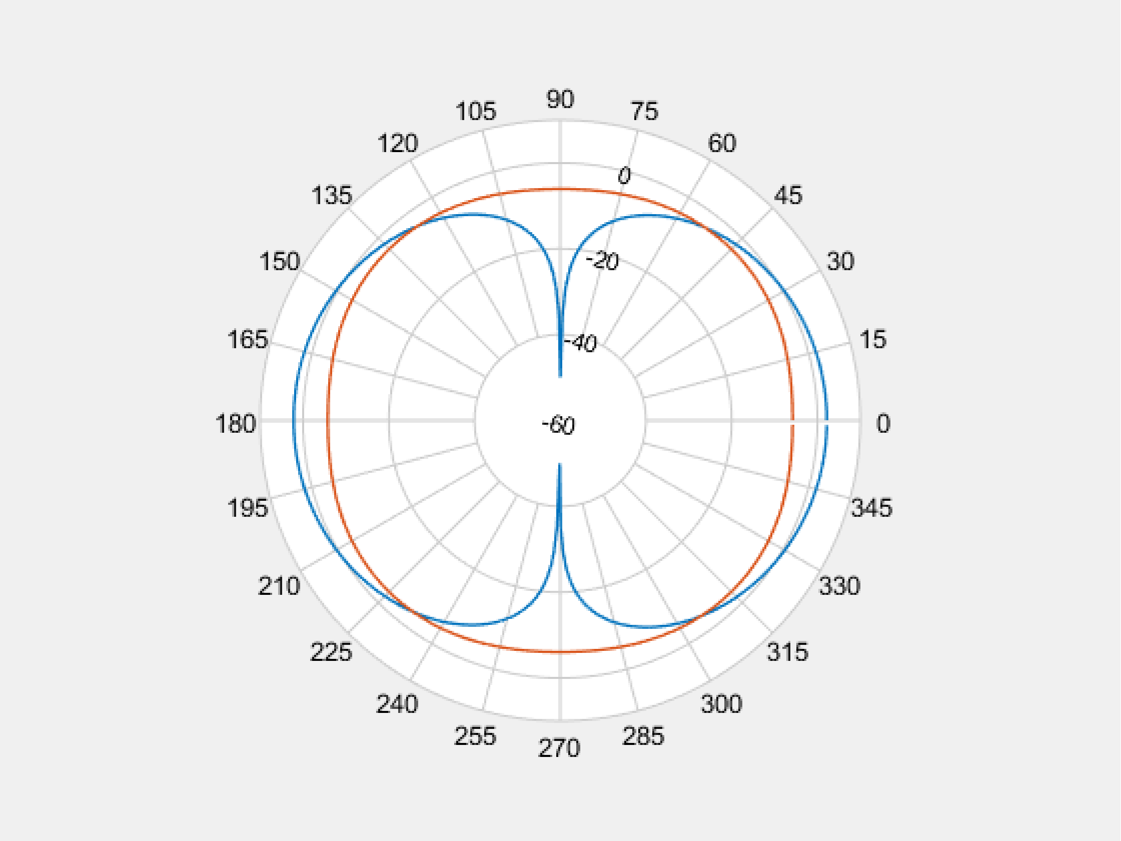
```
[Horizontal,Vertical,Optional] = msiread('cavity.pln')
```

```
Horizontal = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: [360x1 double]
           Elevation: 0
           Frequency: 1.0000e+09
               Slice: 'Elevation'


Vertical = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: 0
           Elevation: [360x1 double]
           Frequency: 1.0000e+09
```

```
                Slice: 'Azimuth'


Optional = struct with fields:
         name: 'Cavity Antenna Specifications'
    frequency: 1.0000e+09
         gain: [1x1 struct]
```

Add horizontal directivity data of the cavity antenna to the existing polar pattern of the dipole

```
add(P,Horizontal.Azimuth,Horizontal.Magnitude);
```



Version History

Introduced in R2016a

See Also

addCursor | animate | createLabels | findLobes | replace | showPeaksTable | showSpan

addCursor

Add cursor to polar plot angle

Syntax

```
addCursor(p,angle)
addCursor(p,angle,index)
id = addCursor(___)
```

Description

`addCursor(p,angle)` adds a cursor to the active polar plot, `p`, at the data point closest to the specified `angle`. Angle units are in degrees.

The first cursor added is called `'C1'`, the second `'C2'`, and so on.

`addCursor(p,angle,index)` adds a cursor at a specified data set `index`. `index` can be a vector of indices.

`id = addCursor(___)` returns a cell array with one ID for each cursor created. You can specify any of the arguments from the previous syntaxes.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`angle` — Angle values
scalar in degrees | vector in degrees

Angle values at which the cursor is added, specified as a scalar or a vector in degrees.

`index` — Data set index
scalar | vector

Data set index, specified as a scalar or a vector.

Examples

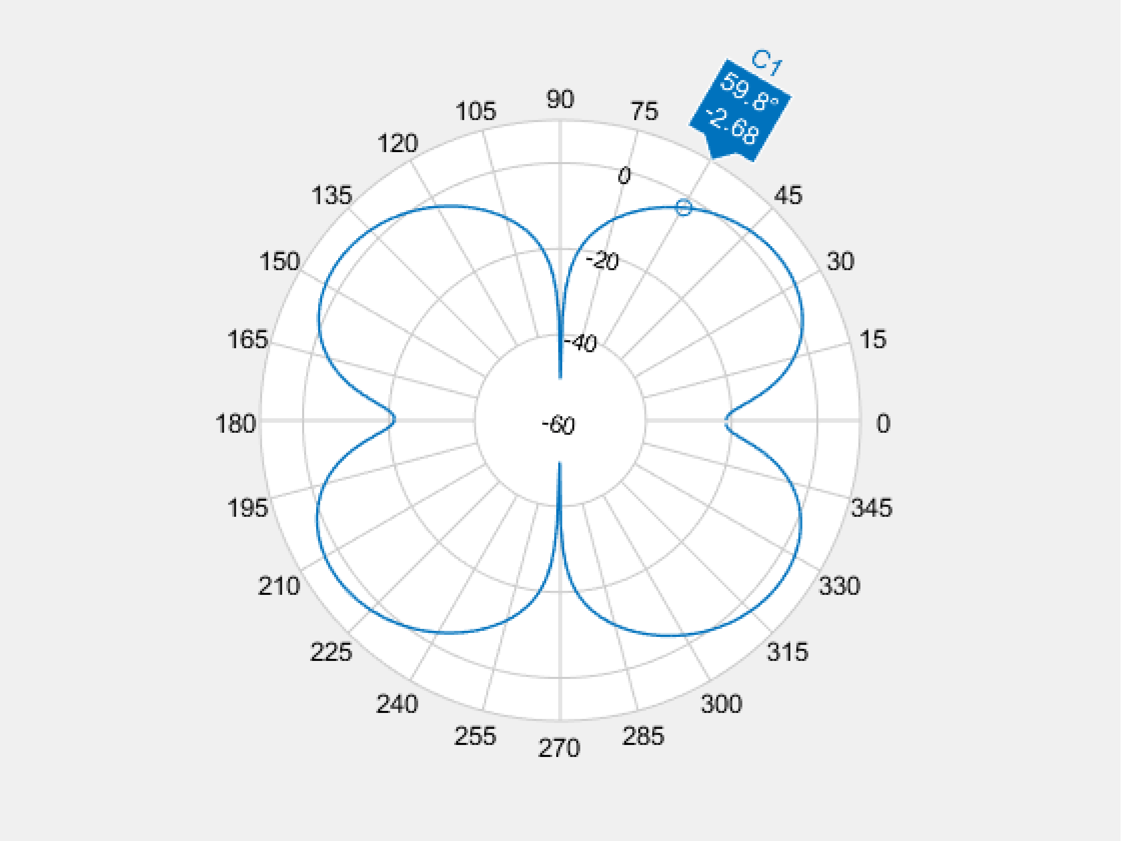
Add Cursor to Plot

Create a dipole antenna and calculate its directivity at a frequency of 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

Add a cursor to the polar plot at approximately 60 degrees. To place the cursor at 60 degrees, move it there by placing the pointer on the cursor and dragging.

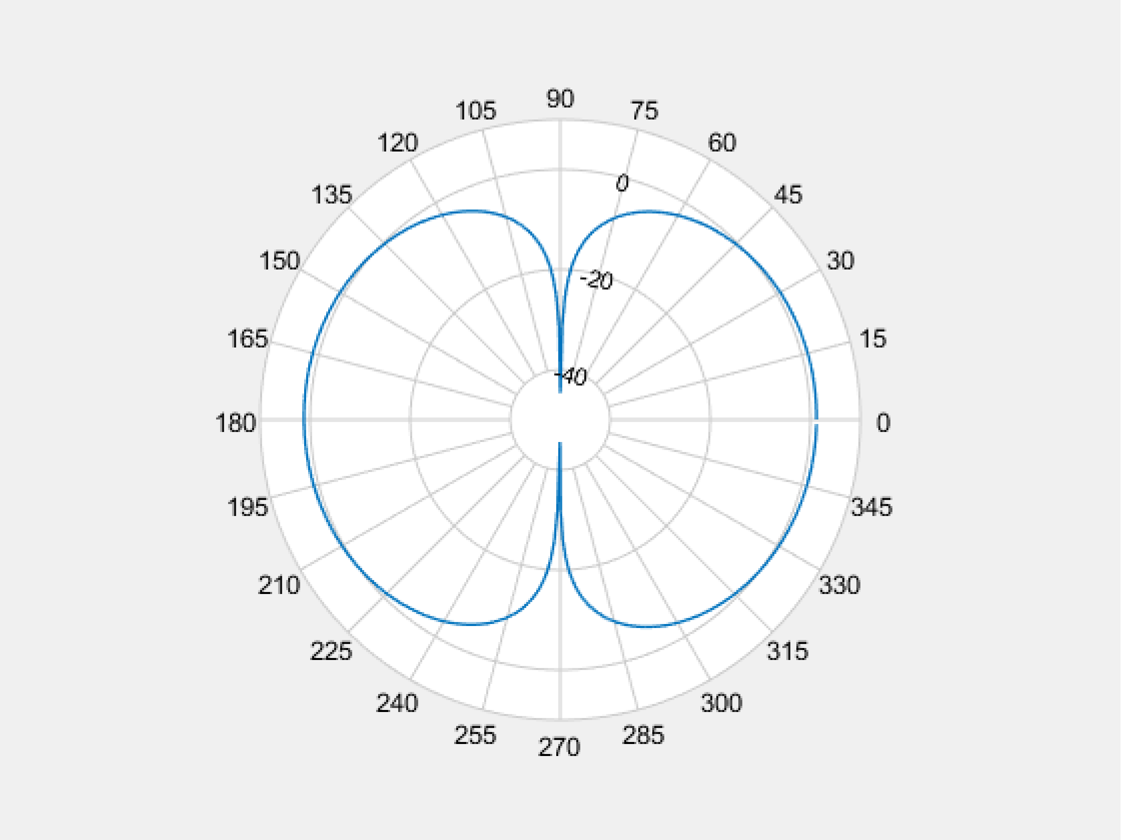
```
p = polarpattern(D);
addCursor(p,60);
```



Add Cursors to Two Data Sets

Create a top-hat monopole and plot its directivity at a frequency of 75 MHz.

```
m = monopoleTopHat;
M = pattern(m,75e6,0,0:1:360);
P = polarpattern(M);
```

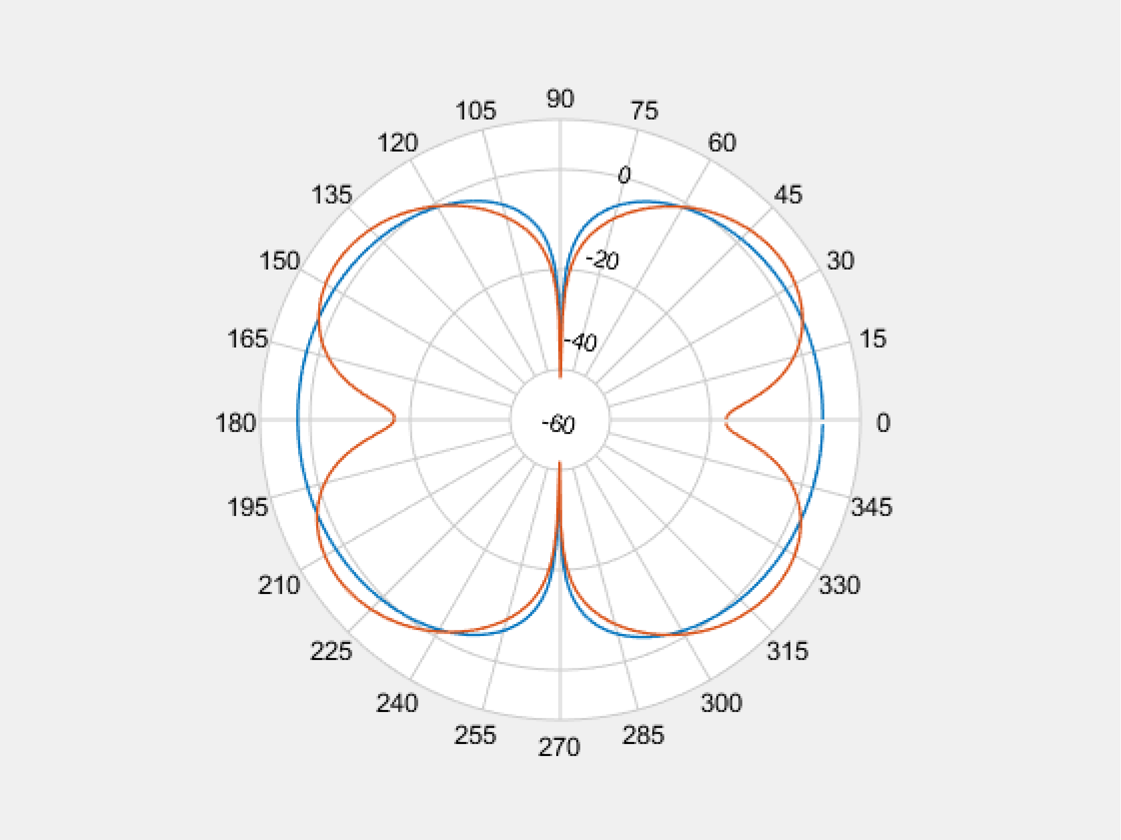


Create a dipole antenna and calculate its directivity at a frequency of 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

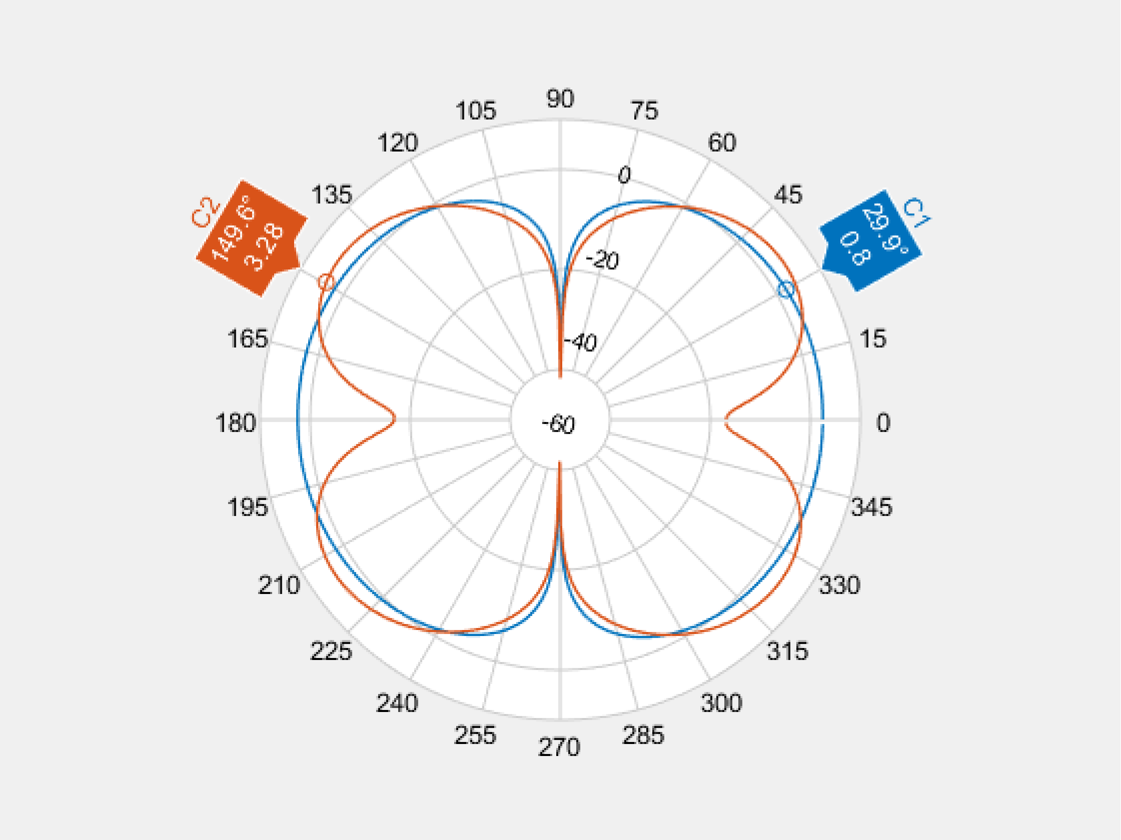
Add the directivity pattern of the dipole to the polar plot of the top-hat monopole.

```
add(P,D);
```



Add a cursor at approximately 30 degrees to the top-hat monopole polar pattern (data set 1) and at approximately 150 degrees to the dipole polar pattern (data set 2).

```
addCursor(P,[30 150],[1 2]);
```



Version History

Introduced in R2016a

See Also

`add` | `animate` | `createLabels` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

animate

Replace existing data with new data for animation

Syntax

```
animate(p,data)
animate(p,angle,magnitude)
```

Description

`animate(p,data)` removes all the current data from polar plot, `p` and adds new data, based on real amplitude values, `data`.

`animate(p,angle,magnitude)` removes all the current data polar plot, `p` and adds new data sets of angle vectors and corresponding magnitude matrices.

Input Arguments

p — Polar plot

scalar handle

Polar plot, specified as a scalar handle.

`data` — Antenna or array data

real length-*M* vector | real *M*-by-*N* matrix | real *N-D* array | complex vector or matrix

Antenna or array data, specified as one of the following:

- A real length-*M* vector, where *M* contains the magnitude values with angles assumed to be $\frac{(0:M-1)}{M} \times 360^\circ$ degrees.
- A real *M*-by-*N* matrix, where *M* contains the magnitude values and *N* contains the independent data sets. Each column in the matrix has angles taken from the vector $\frac{(0:M-1)}{M} \times 360^\circ$ degrees. The set of each angle can vary for each column.
- A real *N-D* array, where *N* is the number of dimensions. Arrays with dimensions `2` and greater are independent data sets.
- A complex vector or matrix, where `data` contains Cartesian coordinates (*(x,y)* of each point. *x* contains the real part of `data` and *y* contains the imaginary part of `data`.

When data is in a logarithmic form such as dB, magnitude values can be negative. In this case,`polarpattern` plots the lowest magnitude values at the origin of the polar plot and highest magnitude values at the maximum radius.

`angle` — Set of angles

vector in degrees

Set of angles, specified as a vector in degrees.

`magnitude` — Set of magnitude values
vector | matrix

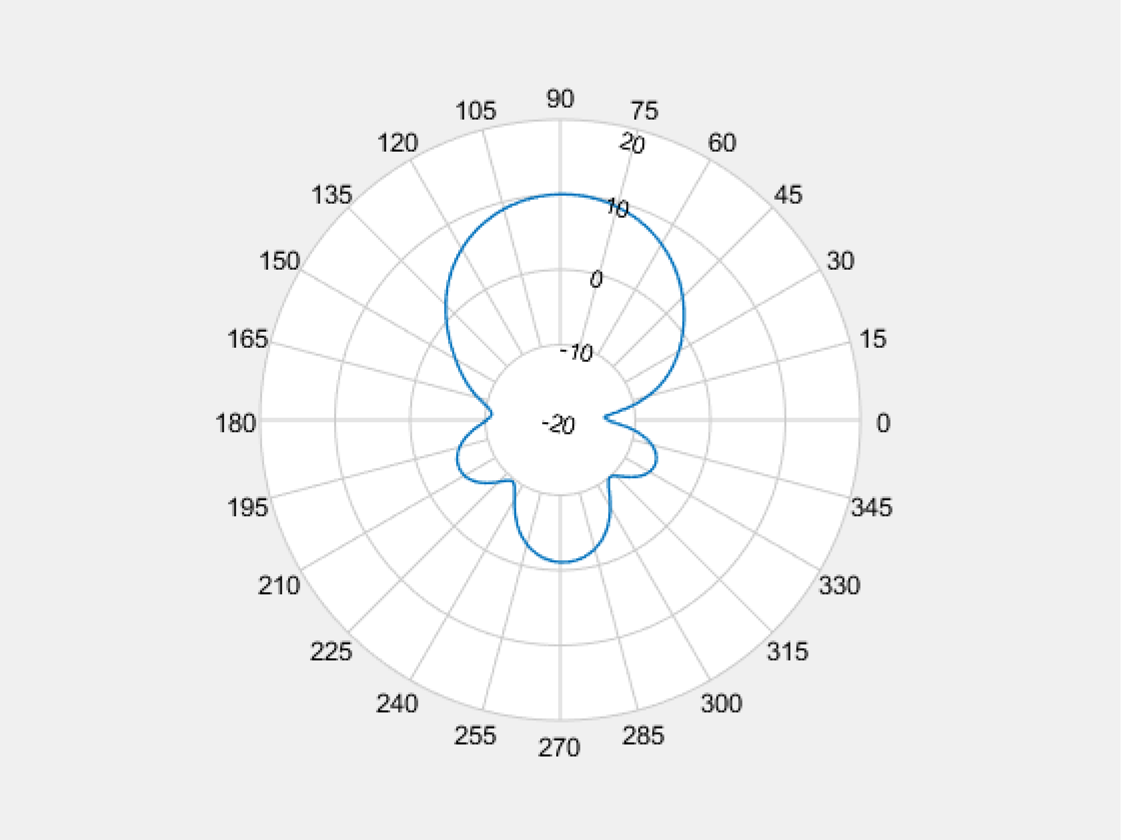
Set of magnitude values, specified as a vector or a matrix. For a matrix of magnitude values, each column is an independent set of magnitude values and corresponds to the same set of angles.

Examples

Replace Existing Polar Plot Data For Animation

Create a helix antenna that has a 28 mm radius, a 1.2 mm width, and 4 turns. Plot the directivity of the antenna at 1.8 GHz.

```
hx = helix('Radius',28e-3,'Width',1.2e-3,'Turns',4);
H = pattern(hx, 1.8e9,0,0:1:360);
P = polarpattern(H);
```

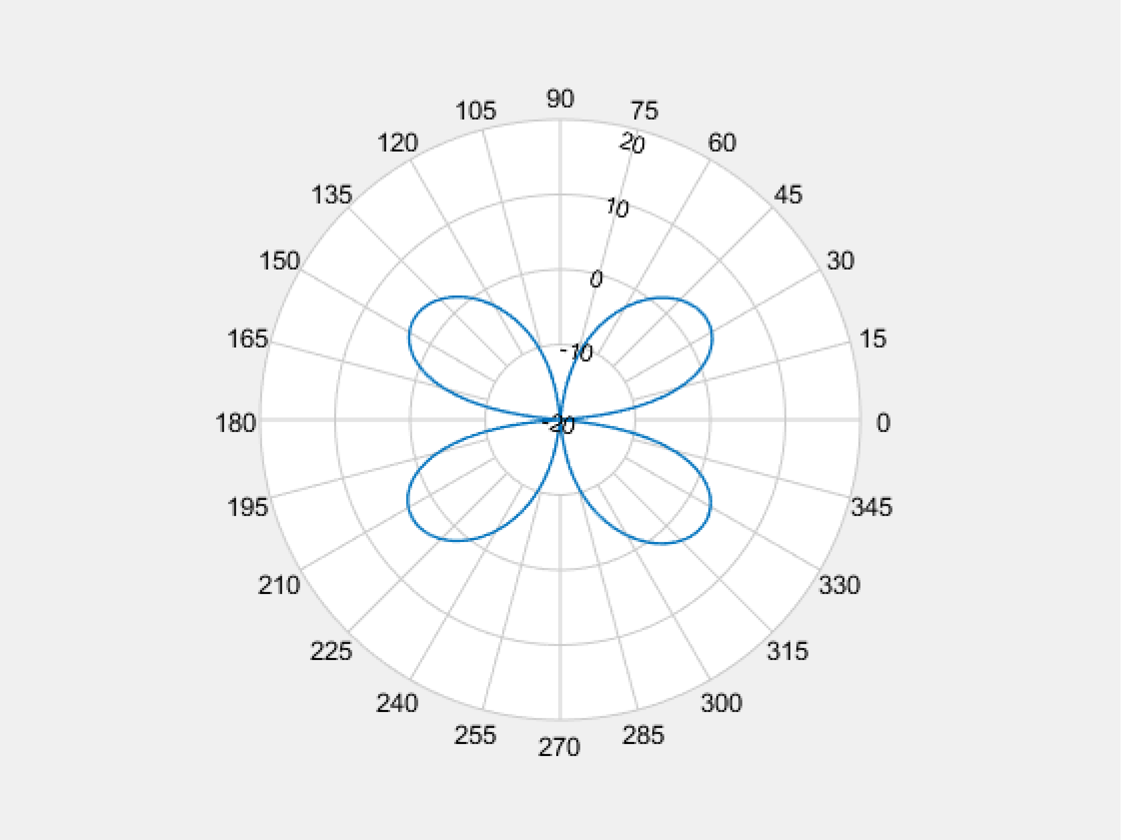


Create a dipole antenna and calculate its directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

Replace the existing polar plot of the helix antenna with the directivity of the dipole using the `animate` method.

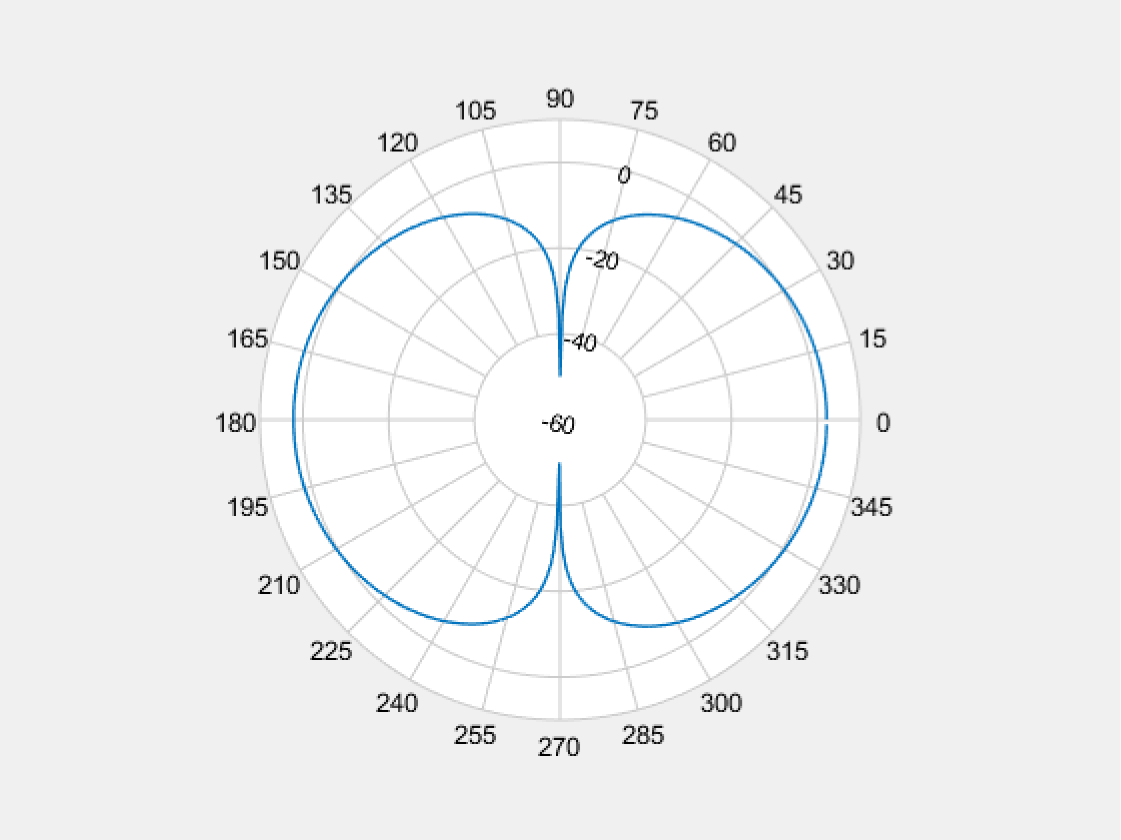
```
animate(P,D);
```



Animate Using Cavity Data

Create a default dipole antenna and plot the polar pattern of its directivity at 1 GHz.

```
d = dipole;
D = pattern(d,75e6,0,0:1:360);
P = polarpattern(D);
```



Create a default cavity antenna. Calculate the directivity of the antenna and write the data to `cavity.pln` using the `msiwrite` function.

```
c = cavity;
msiwrite(c,2.8e9,'cavity','Name','Cavity Antenna Specifications');
```

Read the cavity specifications file into `Horizontal`, `Vertical` and `Optional` structures using the `msiread` function.

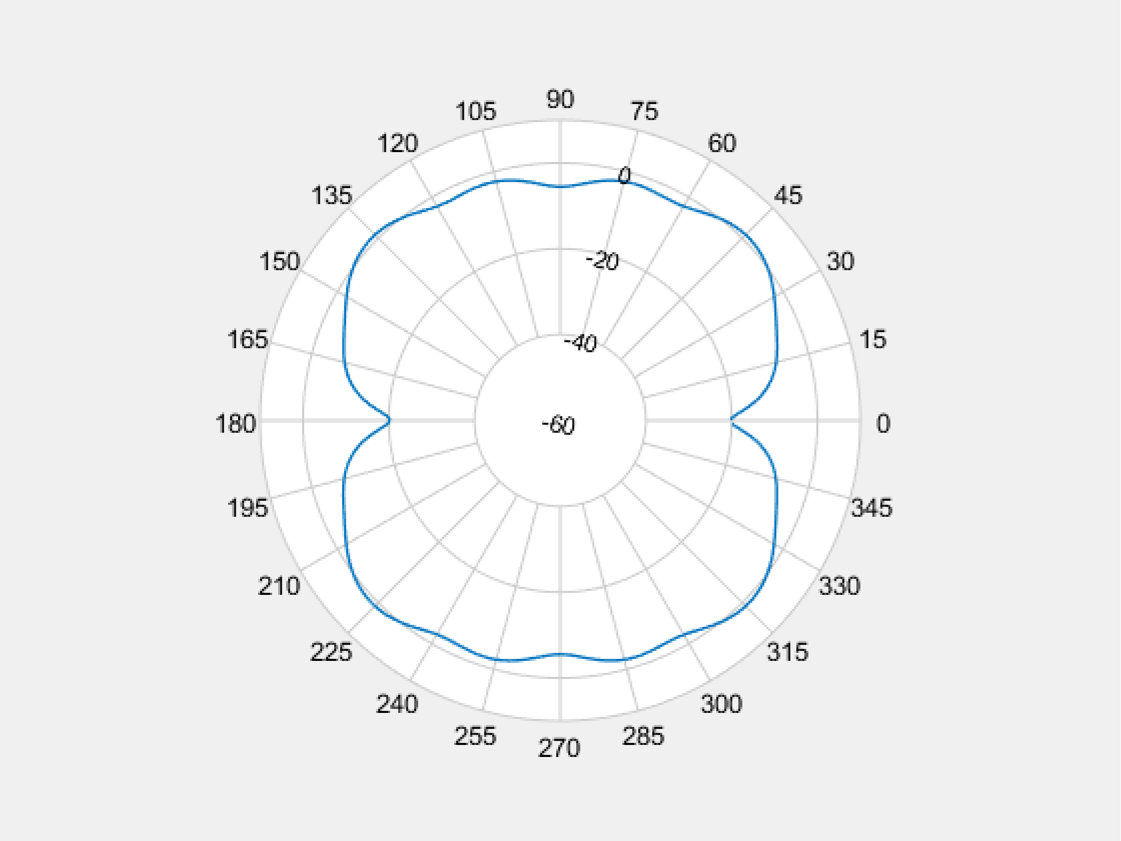
```
[Horizontal,Vertical,optional]= msiread('cavity.pln')
```

```
Horizontal = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: [360x1 double]
           Elevation: 0
           Frequency: 2.8000e+09
               Slice: 'Elevation'


Vertical = struct with fields:
    PhysicalQuantity: 'Gain'
           Magnitude: [360x1 double]
               Units: 'dBi'
             Azimuth: 0
           Elevation: [360x1 double]
           Frequency: 2.8000e+09
```

```
              Slice: 'Azimuth'


optional = struct with fields:
         name: 'Cavity Antenna Specifications'
    frequency: 2.8000e+09
         gain: [1x1 struct]
```

Replace data from the dipole antenna with data from cavity antenna.

```
animate(P,Horizontal.Azimuth,Horizontal.Magnitude);
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `createLabels` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

createLabels

Create legend labels for polar plot

Syntax

`createLabels(p,format,array)`

Description

`createLabels(p,format,array)` adds the specified `format` label to each `array` of the polar plot `p`. The labels are stored as a cell array in the `LegendLabels` property of `p`.

Input Arguments

p — Polar plot

scalar handle

Polar plot, specified as a scalar handle.

`format` — Format for legend label

cell array

Format for legend label added to the polar plot, specified as a cell array. For more information on legend label format see, `legend`.

Data Types: `char`

`array` — Values to apply to `format`

array

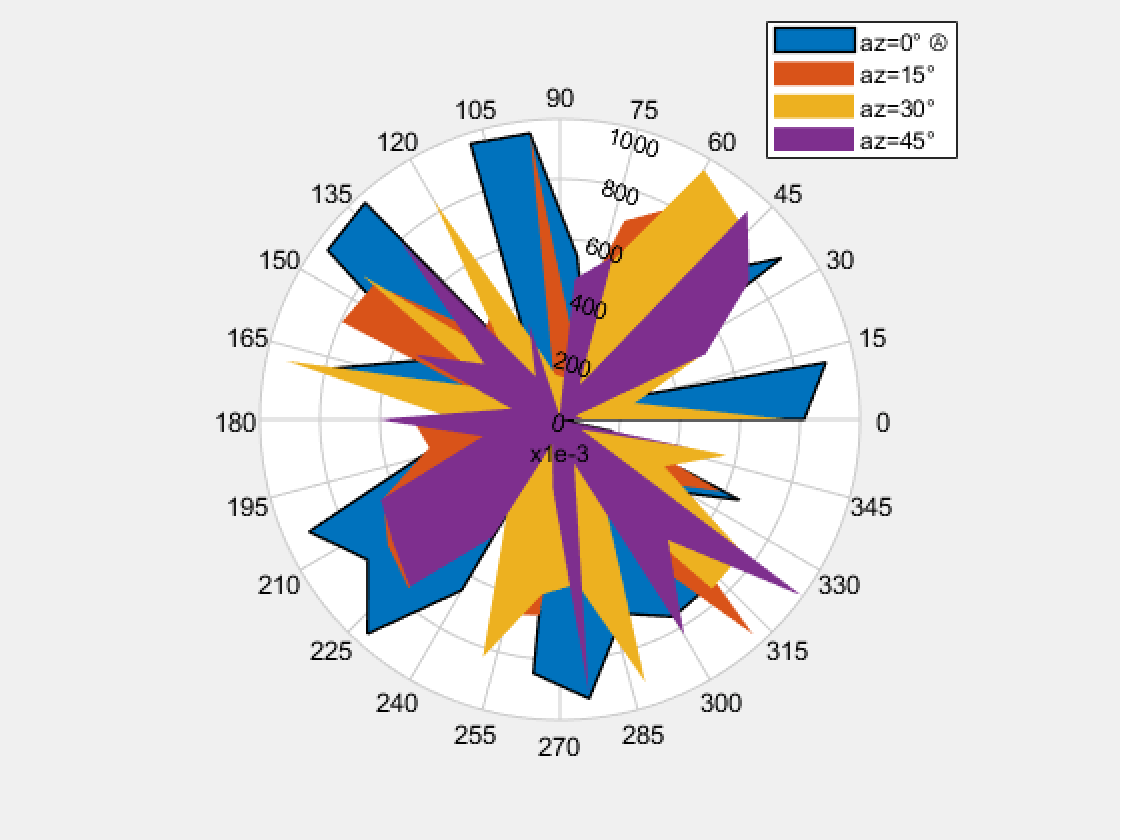
Values to apply to `format` , specified as an array. The values can be an array of angles or array of magnitude.

Examples

Add Legend Label to Polar Plot

Create a polar plot of unique values. Generate a legend label for this plot.

```
p = polarpattern(rand(30,4),'Style','filled');
createLabels(p,'az=%d#deg',0:15:45)
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

findLobes

Main, back, and side lobe data

Syntax

```
L = findLobes(p)
L = findLobes(p,index)
```

Description

`L = findLobes(p)` returns a structure, `L`, defining the main, back, and side lobes of the antenna or array radiation pattern in the specified polar plot, `p`.

`L = findLobes(p,index)` returns the radiation pattern lobes from the data set specified in `index`.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`index` — Index of data set
scalar

Index of data set, specified as a scalar.

Output Arguments

L — Main,back, and side lobe data
structure

Main,back, and side lobe data, returned as a structure.

Examples

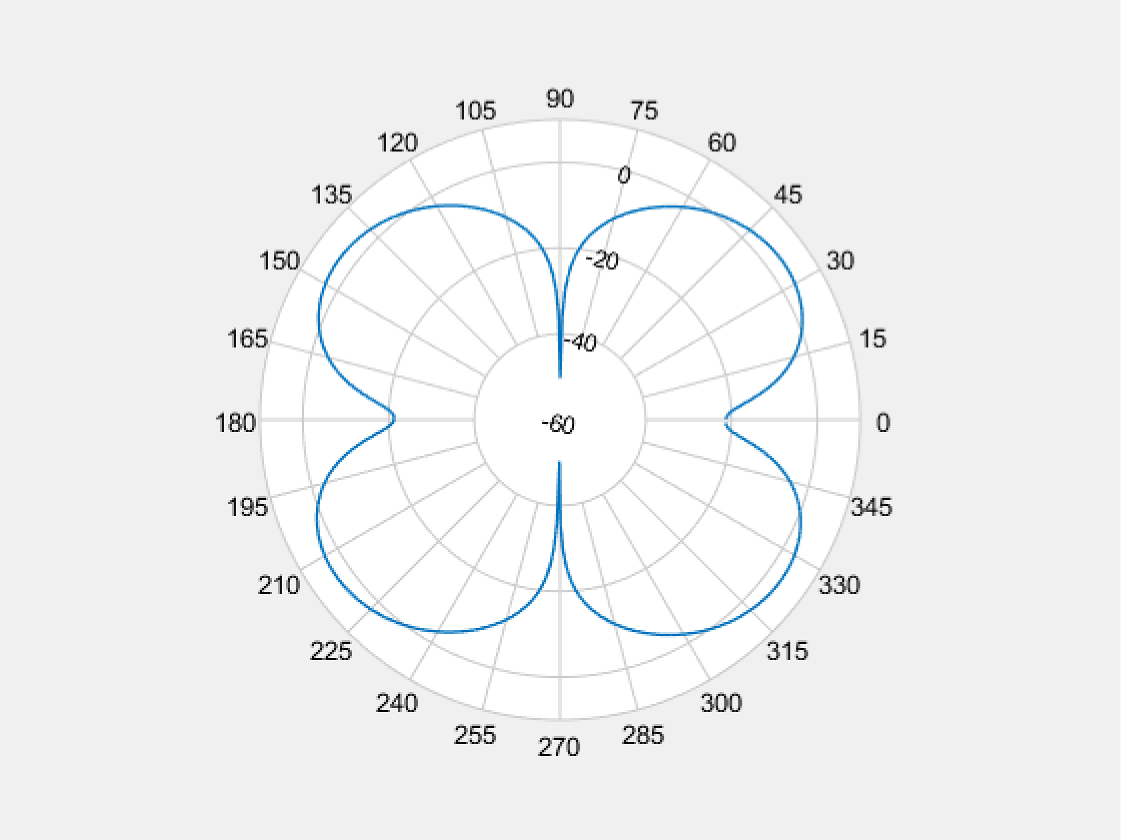
Find Main, Back, and Side Lobes

Create a dipole antenna and calculate its directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

Create a polar plot of the dipole directivity. Find the main, back, and side lobes of the dipole antenna.

```
p = polarpattern(D);
```



```
L = findLobes(p)
```

```
L = struct with fields:
     mainLobe: [1x1 struct]
     backLobe: [1x1 struct]
    sideLobes: [1x1 struct]
           FB: 0.0129
          SLL: 0
         HPBW: 30.9141
         FNBW: 89.7507
        FBIdx: [216 35.5000]
       SLLIdx: [216 326]
      HPBWIdx: [202 233]
      HPBWAng: [200.4432 231.3573]
      FNBWIdx: [181 271]
```

Inspect main, back, and side lobe data.

```
MainLobe = L.mainLobe
```

```
MainLobe = struct with fields:
        index: 216
    magnitude: 3.6685
        angle: 214.4044
       extent: [181 271]
```

```
BackLobe = L.backLobe
BackLobe = struct with fields:
    magnitude: 3.6556
        angle: 34.4044
       extent: [361 91]
        index: 35.5000
```

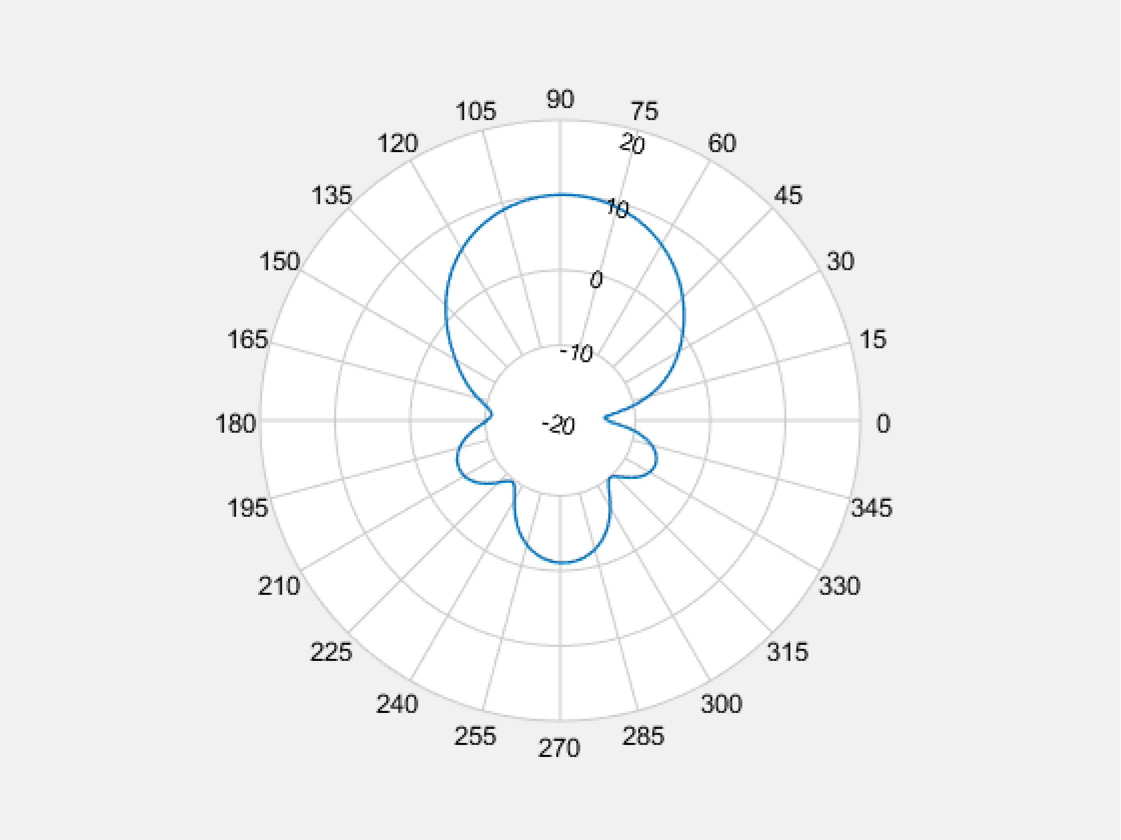
```
SideLobe = L.sideLobes
```

```
SideLobe = struct with fields:
        index: 326
    magnitude: 3.6685
        angle: 324.0997
       extent: [2x2 double]
```

Find Lobes in Two Data Sets

Create a helix antenna that has a 28 mm radius, a 1.2 mm width, and 4 turns. Calculate and plot the directivity of the antenna at 1.8 GHz.

```
hx = helix('Radius',28e-3,'Width',1.2e-3,'Turns',4);
H = pattern(hx, 1.8e9,0,0:1:360);
P = polarpattern(H);
```

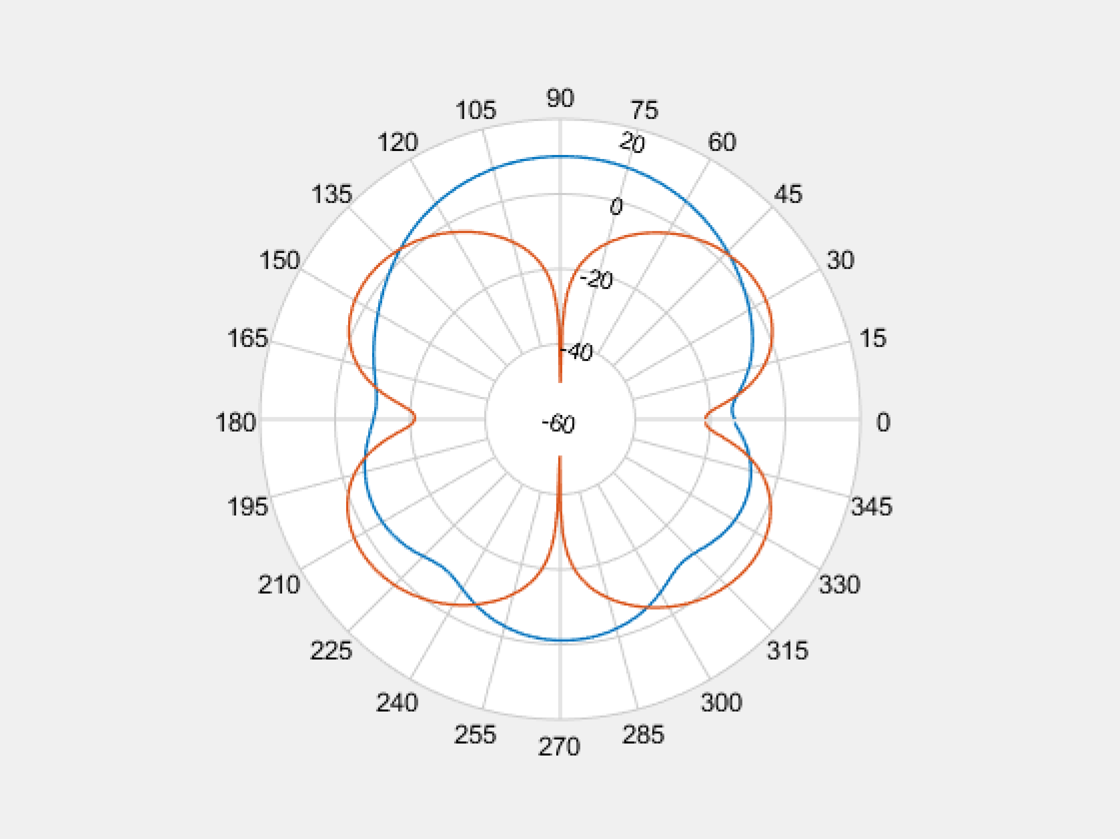


Create a dipole antenna and calculate the directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

Add the directivity of the dipole to the existing polar plot.

```
add(P,D);
```



Find the main, back, and side lobes of helix antenna.

```
L = findLobes(P,1)
```

```
L = struct with fields:
     mainLobe: [1x1 struct]
     backLobe: [1x1 struct]
    sideLobes: [1x1 struct]
           FB: 11.1683
          SLL: 11.1166
         HPBW: 56.8421
         FNBW: 172.5208
        FBIdx: [90 270.5000]
       SLLIdx: [90 273]
      HPBWIdx: [61 118]
      HPBWAng: [59.8338 116.6759]
      FNBWIdx: [4 177]
```

Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `replace` | `showPeaksTable` | `showSpan`

replace

Replace polar plot data with new data

Syntax

```
replace(p,data)
replace(p,angle,magnitude)
```

Description

`replace(p,data)` removes all data from polar plot, `p` and adds new data based on real amplitude values, `data`.

`replace(p,angle,magnitude)` removes all the current data and adds new data sets of angle vectors and corresponding magnitude matrices to the polar plot, `p`.

Input Arguments

p — Polar plot

scalar handle

Polar plot, specified as a scalar handle.

`data` — Antenna or array data

real length-*M* vector | real *M*-by-*N* matrix | real *N-D* array | complex vector or matrix

Antenna or array data, specified as one of the following:

- A real length-*M* vector, where *M* contains the magnitude values with angles assumed to be $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees.
- A real *M*-by-*N* matrix, where *M* contains the magnitude values and *N* contains the independent data sets. Each column in the matrix has angles taken from the vector $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees. The set of each angle can vary for each column.
- A real *N-D* array, where *N* is the number of dimensions. Arrays with dimensions `2` and greater are independent data sets.
- A complex vector or matrix, where `data` contains Cartesian coordinates (*(x,y)* of each point. *x* contains the real part of `data` and *y* contains the imaginary part of `data`.

When data is in a logarithmic form such as dB, magnitude values can be negative. In this case,`polarpattern` plots the lowest magnitude values at the origin of the polar plot and highest magnitude values at the maximum radius.

`angle` — Set of angles

vector in degrees

Set of angles, specified as a vector in degrees.

`magnitude` — Set of magnitude values
vector | matrix

Set of magnitude values, specified as a vector or a matrix. For a matrix of magnitude values, each column is an independent set of magnitude values and corresponds to the same set of angles.

Examples

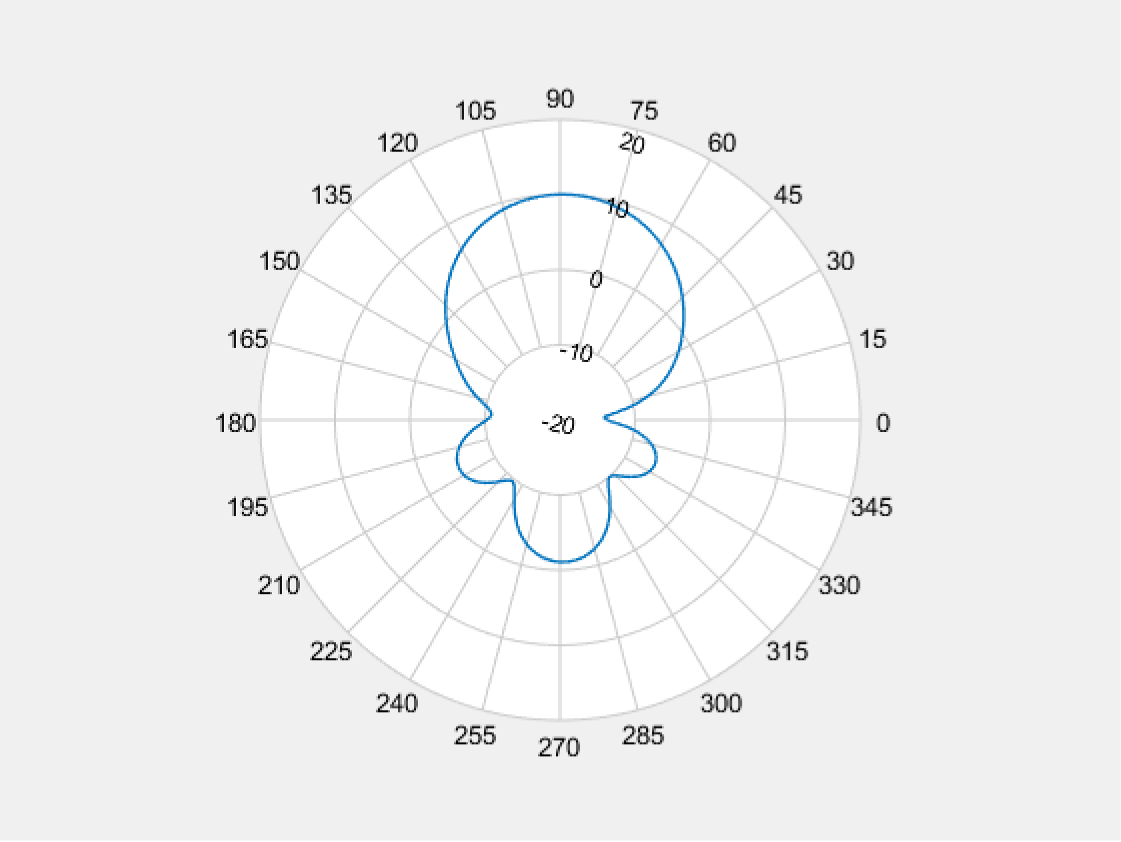
Replace Polar Plot Data with New Data

Create a helix antenna that has a 28 mm radius, a 1.2 mm width, and 4 turns. Calculate the directivity of the antenna at 1.8 GHz.

```
hx = helix('Radius',28e-3,'Width',1.2e-3,'Turns',4);
H = pattern(hx, 1.8e9,0,0:1:360);
```

Plot the polar pattern.

```
P = polarpattern(H);
```

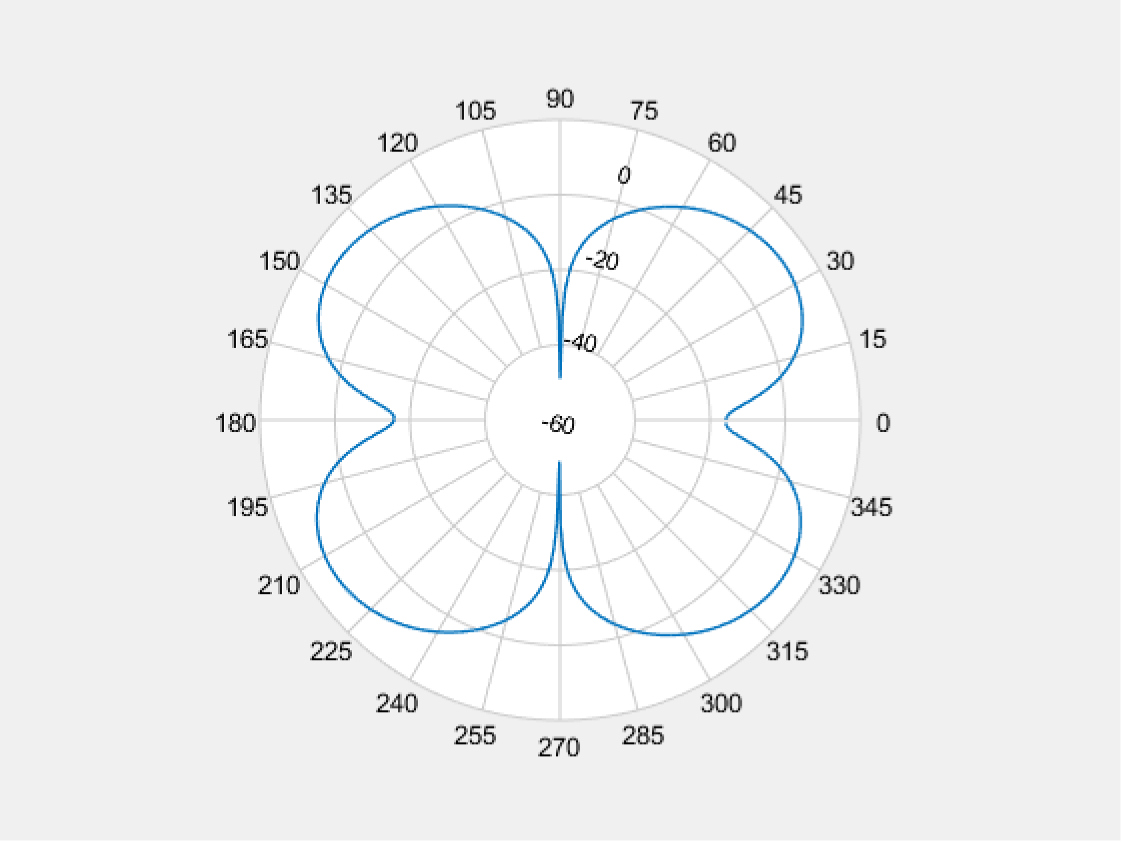


Create a dipole antenna and calculate its directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

Replace the existing polar plot of the helix antenna with the directivity of the dipole.

```
replace(P,D);
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `findLobes` | `showPeaksTable` | `showSpan`

showPeaksTable

Show or hide peak marker table

Syntax

```
showPeaksTable(p,vis)
```

Description

`showPeaksTable(p,vis)` shows or hides a table of the peak values. By default, the peak values table is visible.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`vis` — Show or hide peaks table
`0` | `1`

Show or hide peaks table, specified as `0` or `1`.

Examples

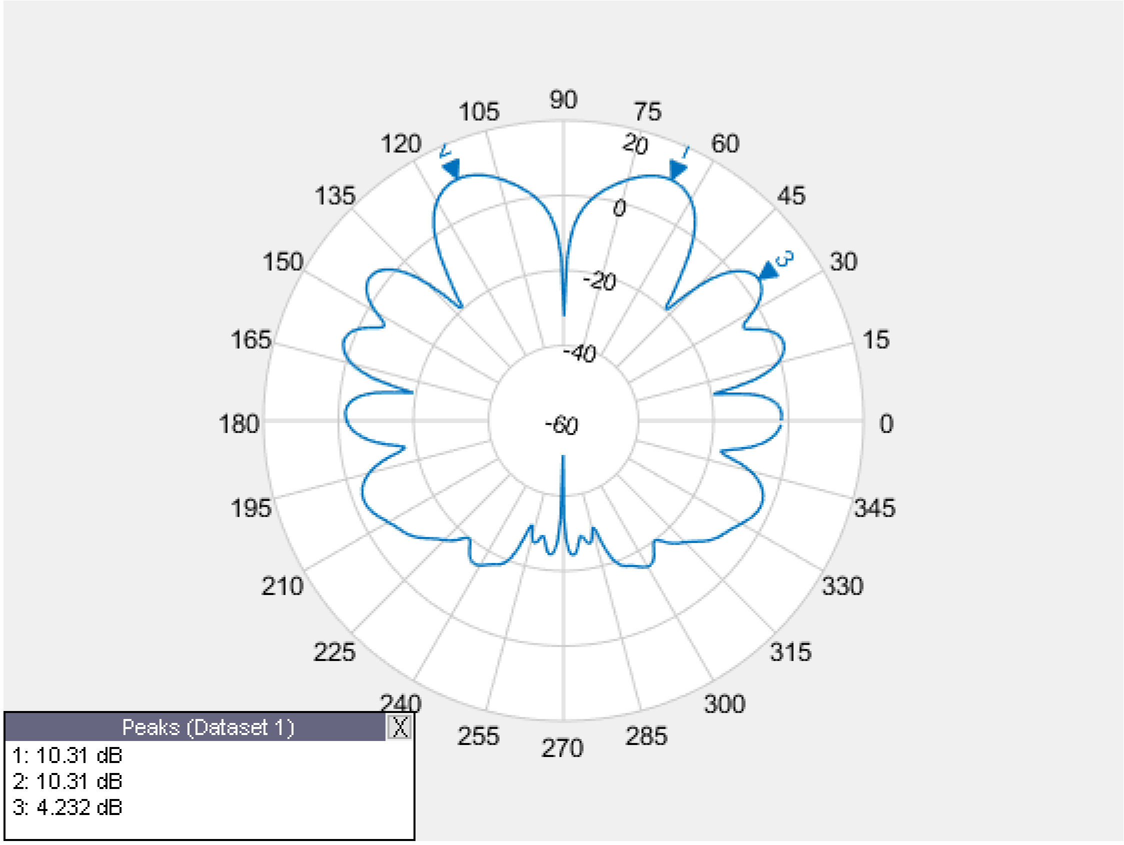
Peaks of Antenna in Polar Pattern

Create a monopole antenna and calculate the directivity at 1 GHz.

```
m = monopole;
M = pattern(m,1e9,0,0:1:360);
```

Plot the polar pattern and show three peaks of the antenna. When creating a `polarpattern` plot, if you specify the Peaks property, the peaks table is displayed by default.

```
P = polarpattern(M,'Peaks',3);
```

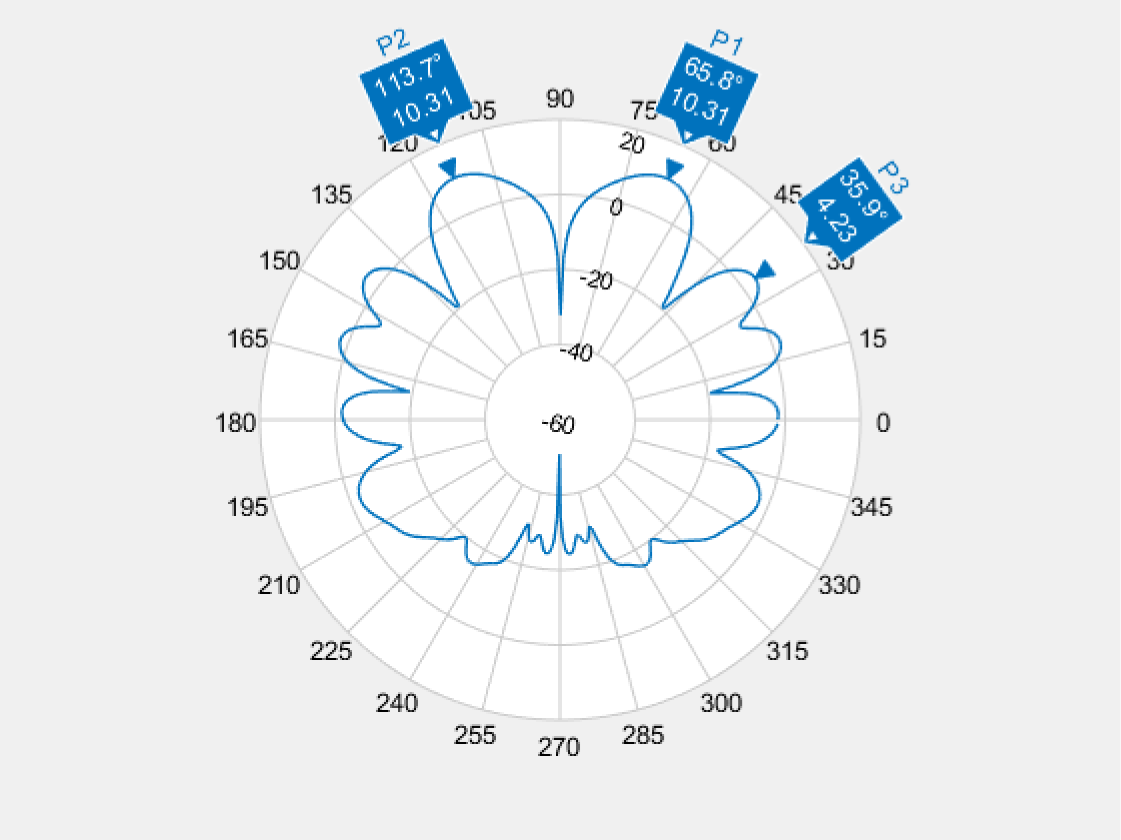


Peaks (Dataset 1) X

1: 10.31 dB
2: 10.31 dB
3: 4.232 dB

Hide the table. When the peaks table is hidden, the peak markers display the peak values.

```
showPeaksTable(P,0);
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `findLobes` | `replace` | `showSpan`

showSpan

Show or hide angle span between two markers

Syntax

```
showSpan(p,id1,id2)
showSpan(p,id1,id2,true)
showSpan(p,vis)
showSpan(p)
d = showSpan( ___ )
```

Description

`showSpan(p,id1,id2)` displays the angle span between two angle markers, `id1` and `id2`. The angle span is calculated counterclockwise.

`showSpan(p,id1,id2,true)` automatically reorders the angle markers such that the initial angle span is less than or equal to 180° counterclockwise.

`showSpan(p,vis)` sets angle span visibility by setting `vis` to `true` or `false`.

`showSpan(p)` toggles the angle span display on and off.

`d = showSpan( ___ )` returns angle span details in a structure, `d` using any of the previous syntaxes.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`id1,id2` — Cursor or peak marker identifiers
character vector

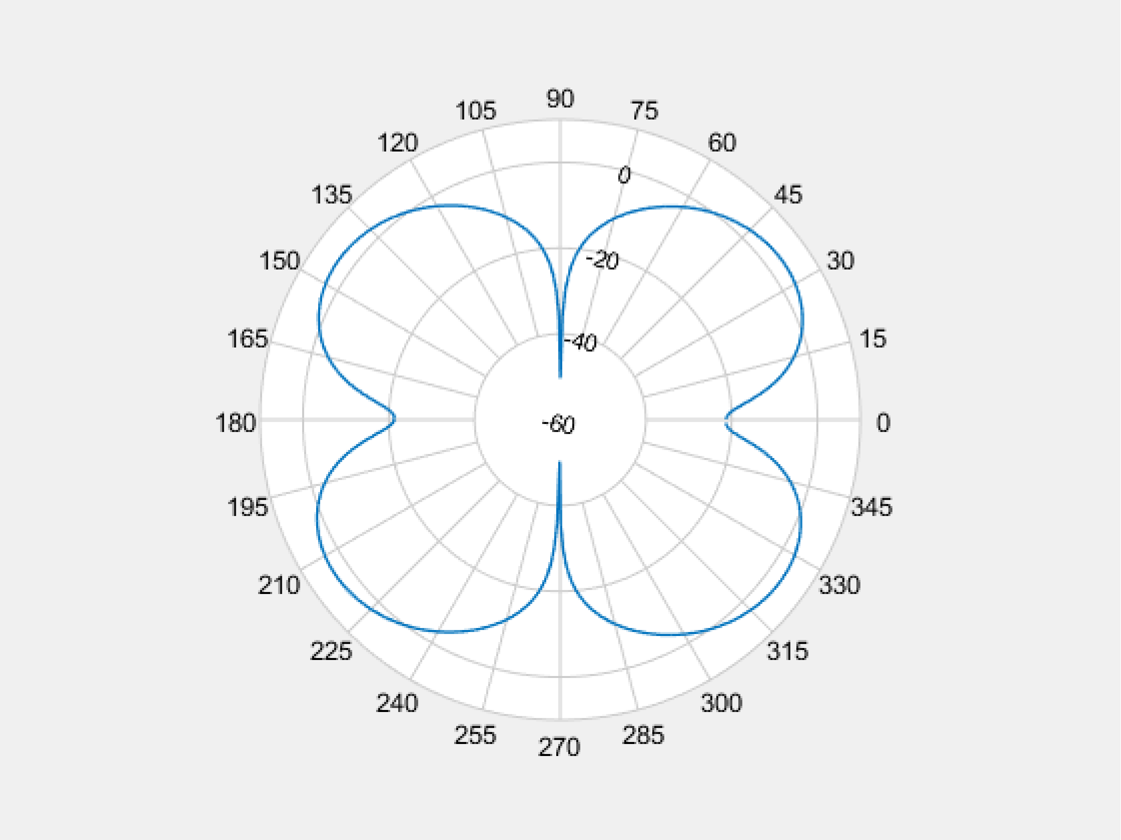
Cursor or peak marker identifiers, specified as character vector. Adding cursors to the polar plot creates cursor marker identifiers. Adding peaks to the polar plot creates peak marker identifiers.

Example: `showspan(p,'C1','C2')`. Displays the angle span between cursors, `C1` and `C2` in polar plot, `p`.

ExamplesShow Angle Span

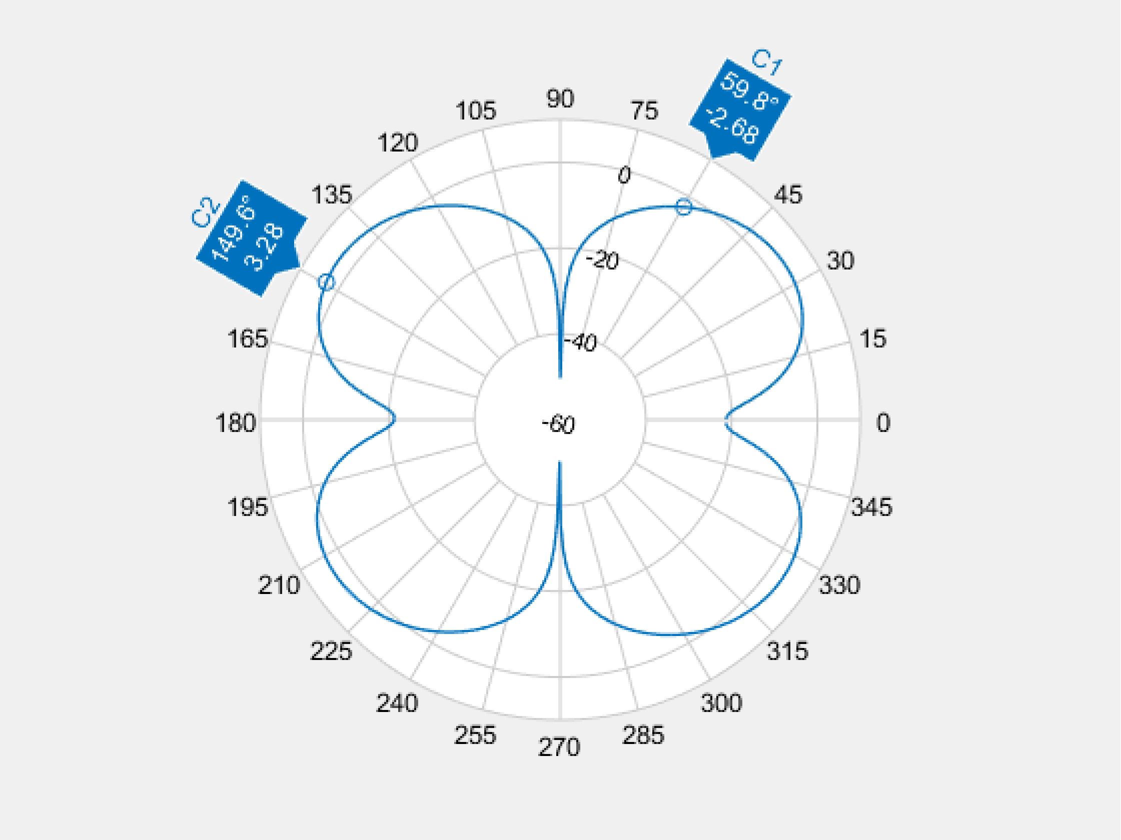
Create a dipole antenna and plot the directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
p = polarpattern(D);
```



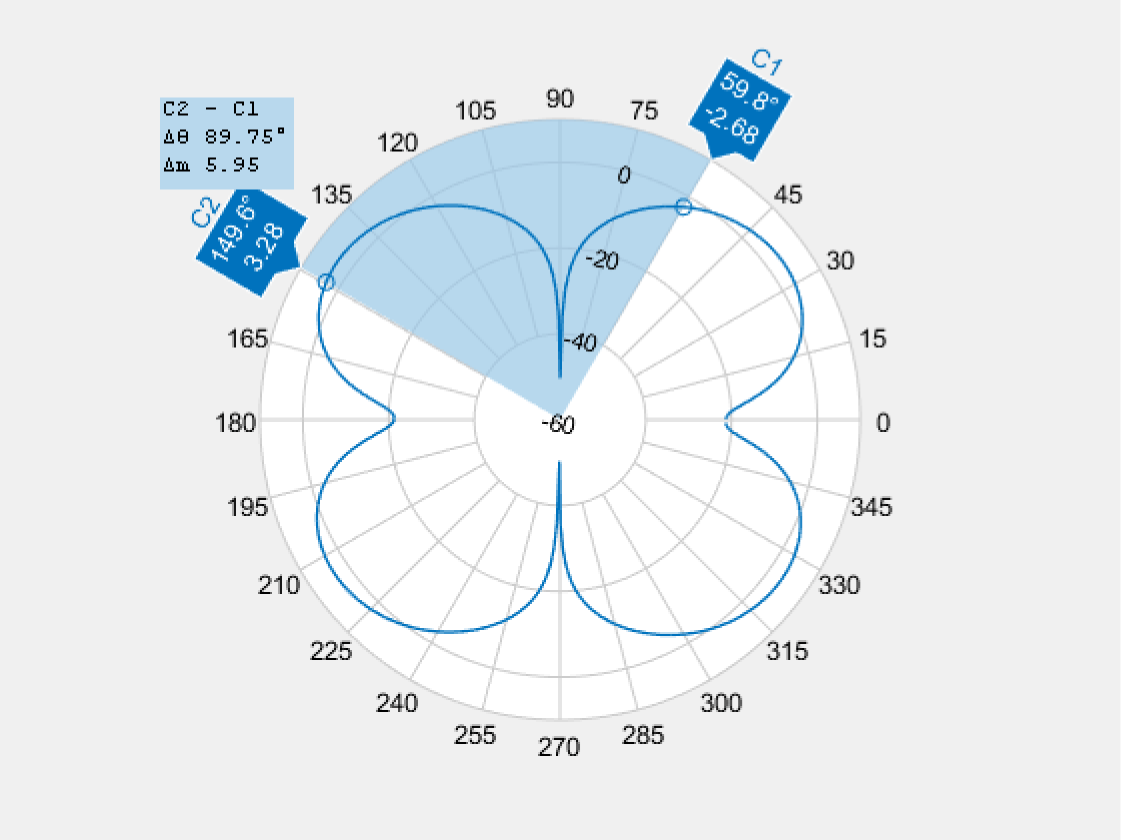
Add cursors to the polar plot at approximately 60 and 150 degrees.

```
addCursor(p,[60 150]);
```



Show the angle span between the two angles.

```
showSpan(p,'C1','C2');
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `findLobes` | `replace` | `showPeaksTable`

arrayFactor

Array factor in dB

Syntax

```
arrayFactor(object,frequency)
arrayFactor(object,frequency,azimuth,elevation)
arrayFactor( ___ ,Name,Value)
```

```
[af] = arrayFactor(object,frequency)
[af,azimuth,elevation] = arrayFactor( ___ )
[af,azimuth,elevation] = arrayFactor( ___ ,Name,Value)
```

Description

`arrayFactor(object,frequency)` plots the 3-D array factor over the specified frequency value in dB.

`arrayFactor(object,frequency,azimuth,elevation)` plots the array factor over the specified frequency, azimuth, and elevation values.

`arrayFactor( ___ ,Name,Value)` plots the array factor using additional options specified by one or more `Name,Value` pair arguments. Specify name-value pair arguments after all other input arguments.

`[af] = arrayFactor(object,frequency)` returns the 3-D array factor over the specified frequency value.

`[af,azimuth,elevation] = arrayFactor( ___ )` returns the array factor at the specified frequency, azimuth, and elevation values.

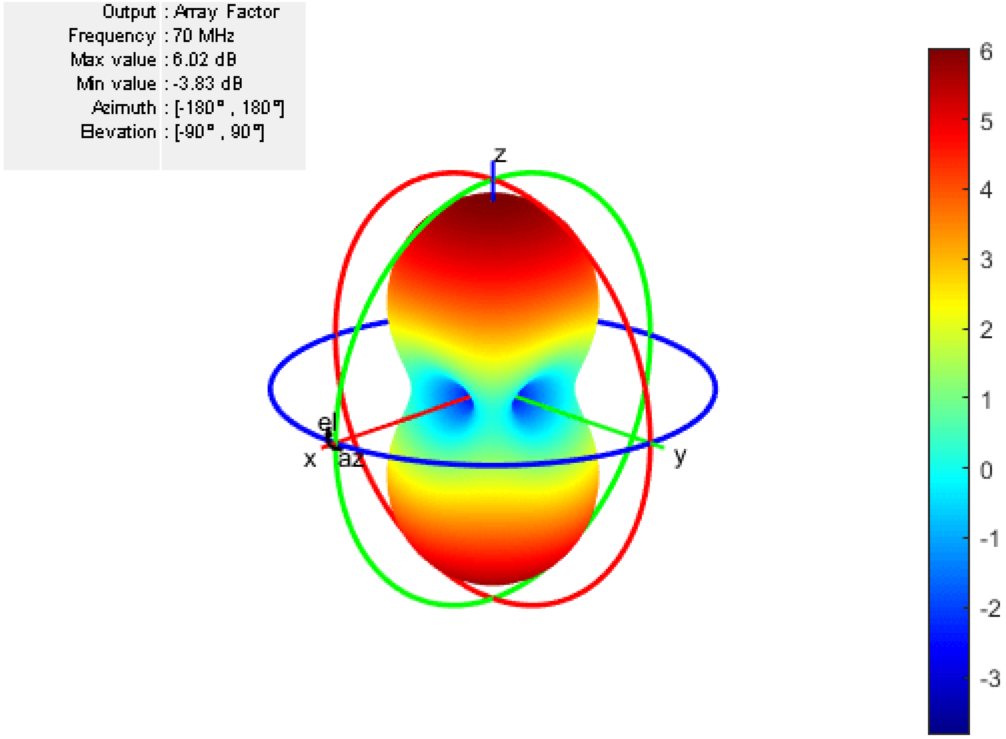
`[af,azimuth,elevation] = arrayFactor( ___ ,Name,Value)` returns the array factor using additional options specified by one or more `Name,Value` pair arguments. Specify name-value pair arguments after all other input arguments.

Examples

Plot Array Factor

Plot the array factor of a default rectangular array at a frequency of 70 MHz.

```
ra = rectangularArray;
arrayFactor(ra,70e6);
```



Input Arguments

`object` — Input antenna array

object

Input antenna array object, specified as an object.

Example: `r = rectangularArray; arrayFactor (r,70e6)`. Calculates the array factor of a rectangular array.

`frequency` — Frequency value used to calculate array factor

scalar in Hz

Frequency value used to calculate array factor, specified as a scalar in Hz.

Example: `70e6`

Data Types: `double`

`azimuth` — Azimuth angle of antenna

`–180:5:180` (default) | vector in degrees

Azimuth angle of the antenna, specified as a vector in degrees.

Example: `–90:5:90`

Data Types: `double`

elevation — Elevation angle of antenna
–90:5:90 (default) | vector in degrees

Elevation angle of the antenna, specified as a vector in degrees.

Example: 0:1:360

Data Types: double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value pair 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: 'CoordinateSystem', rectangular

CoordinateSystem — Coordinate system of array factor
'polar' (default) | 'rectangular' | 'uv'

Coordinate system of array factor, specified as the comma-separated pair consisting of 'CoordinateSystem' and one of these values: 'polar', 'rectangular', 'uv'.

Example: 'CoordinateSystem', 'polar'

Data Types: char

PatternOptions — Parameter to change pattern plot properties
PatternPlotOptions object (default) | scalar

Parameter to change pattern plot properties, specified as the comma-separated pair consisting of 'PatternOptions' and a PatternPlotOptions output. The properties that you can vary are:

- Transparency
- MagnitudeScale

Other properties used in the 'PatternOptions' for the inset figure are ignored in arrayFactor.

Example: p = PatternPlotOptions('Transparency',0.1); Create a pattern plot option with a transparency of 0.1. antarray = rectangularArray;arrayFactor(antarray,70e6,'PatternOptions',p); Use this pattern plot option to visualize the pattern of a helix antenna.

Data Types: double

Output Arguments

af — Array factor
matrix in dB

Array factor, returned as a matrix in dB. The matrix size is the product of number of elevation values and number of azimuth values.

azimuth — Azimuth values
vector in degrees

Azimuth values used to calculate the array factor, returned as a vector in degrees.

`elevation` — Elevation values

vector in degrees

Elevation values used to calculate the array factor, returned as a vector in degrees.

Version History

Introduced in R2017a

See Also

`pattern` | `patternMultiply` | `feedCurrent`

add

Boolean unite operation on two shapes

Syntax

```
c = add(shape1,shape2)
```

Description

`c = add(shape1,shape2)` unites `shape1` and `shape2` using the add operation. You can also use the **+** to add the two shapes together.

Examples

Add Two Circles

Create and view a default circle.

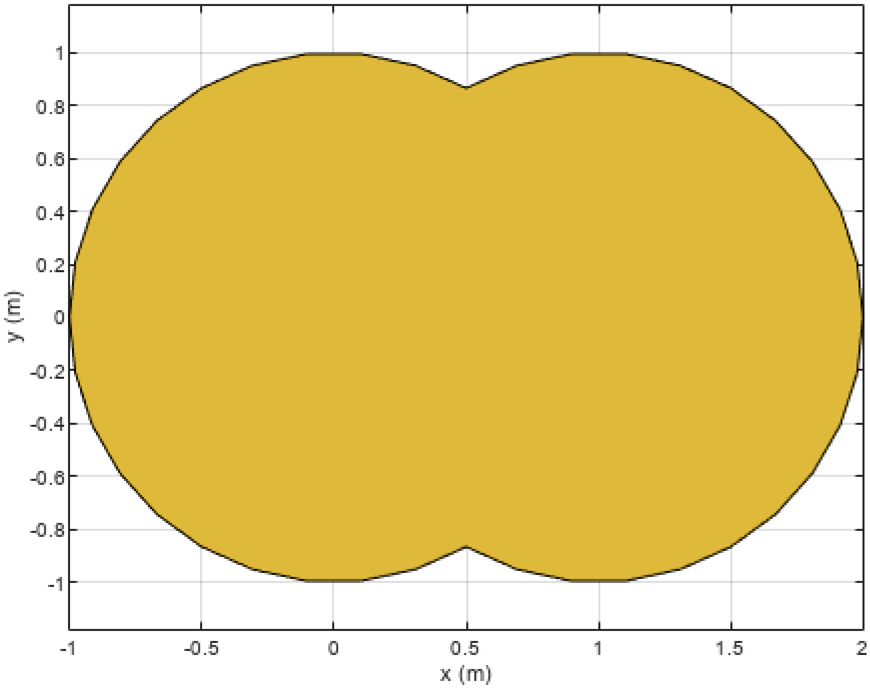
```
circle1 = antenna.Circle;
```

Create a circle with a radius of 1 m. The center of the circle is at [1 0].

```
circle2 = antenna.Circle('Center',[1 0],'Radius',1);
```

Add the two circles.

```
add(circle1,circle2)
```



Add Two Shapes

Create a circle with a radius of 1 m. The center of the circle is at [1 0].

```
circle1 = antenna.Circle('Center',[1 0],'Radius',1);
```

Create a rectangle with a length of 2 m and a width of 4 m centered at the origin.

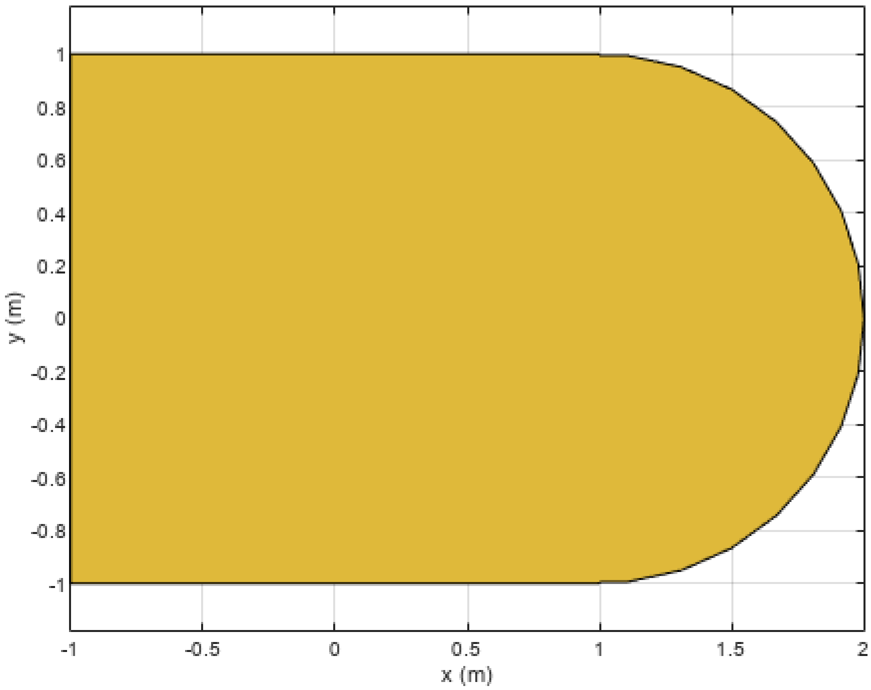
```
rect1 = antenna.Rectangle('Length',2,'Width',2);
```

Add the two shapes together using the + function.

```
polygon1 = circle1+rect1
```

```
polygon1 = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [21x3 double]
```

```
show(polygon1)
```



Input Arguments

`shape1,shape2` — Shapes created using custom elements and shape objects
object

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `c = add(rectangle1, rectangle2)`, where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`area` | `intersect` | `subtract` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

area

Calculate area of shape in square meters

Syntax

```
a = area(shape)
```

Description

`a = area(shape)` calculate area of the shape in units sq.m.

Examples

Create Notched Rectangle

Create a rectangle with a length of 0.15 m, and a width of 0.15 m.

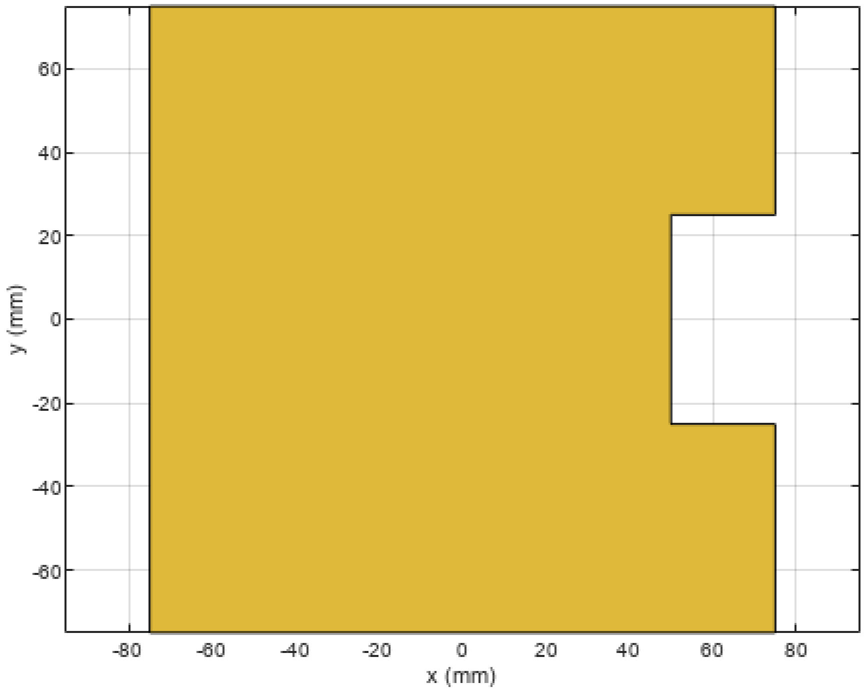
```
r  = antenna.Rectangle('Length',0.15,'Width',0.15);
```

Create a second rectangle with a length of 0.05 m, and a width of 0.05 m. Set the center of the second rectangle at half the length of the first rectangle r.

```
n = antenna.Rectangle('Center',[0.075,0],'Length',0.05,'Width',0.05);
```

Create and view a notched rectangle by subtracting n from r.

```
rn  = r-n;
show(rn)
```



Calculate the area of the notched rectangle.

```
area(rn)
```

```
ans = 0.0212
```

Input Arguments

shape — Shape created using custom elements and shape objects
object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `c = area(rectangle)` where rectangle is created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

intersect

Boolean intersection operation on two shapes

Syntax

```
c = intersect(shape1,shape2)
```

Description

`c = intersect(shape1,shape2)` intersect `shape1` and `shape2` using the intersect operation. You can also use the & to intersect the two shapes.

Examples

Intersect Rectangle and Circle

Create a default rectangle.

```
r = antenna.Rectangle;
```

Create a default circle.

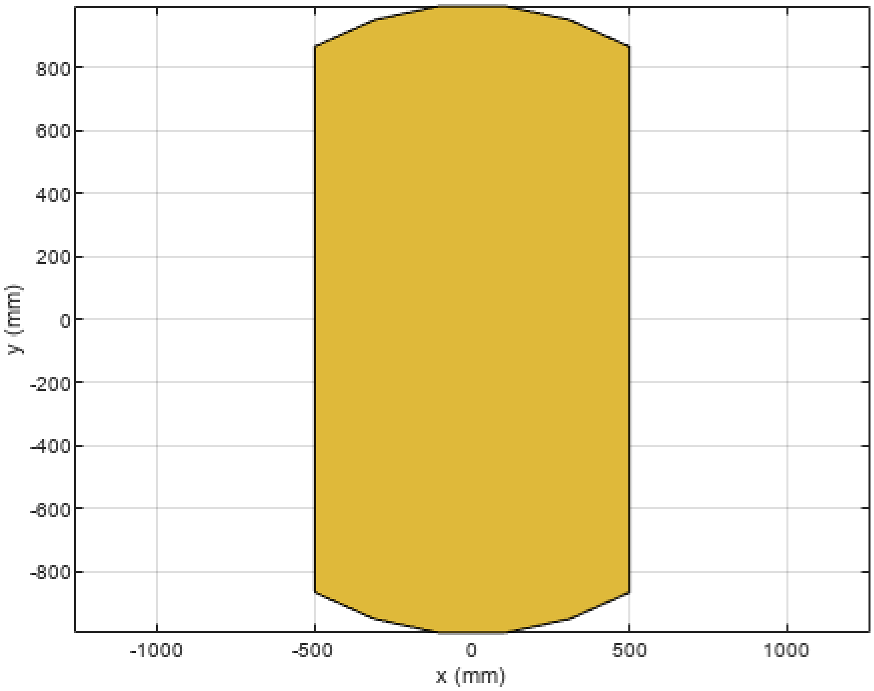
```
c = antenna.Circle;
```

Use `intersect` to combine the shared surfaces of the rectangle and the circle.

```
rc = intersect(r,c)
```

```
rc = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [12x3 double]
```

```
show(rc)
axis equal
```



Input Arguments

`shape1,shape2` — Shapes created using custom elements and shape objects

object

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `c = intersect(rectangle1, rectangle2)` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `area` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot`

rotate

Rotate shape about axis and angle

Syntax

```
rotate(shape,angle,axis1,axis2)
c = rotate(shape,angle,axis1,axis2)
```

Description

`rotate(shape,angle,axis1,axis2)` rotate shape about an axes object and angle.

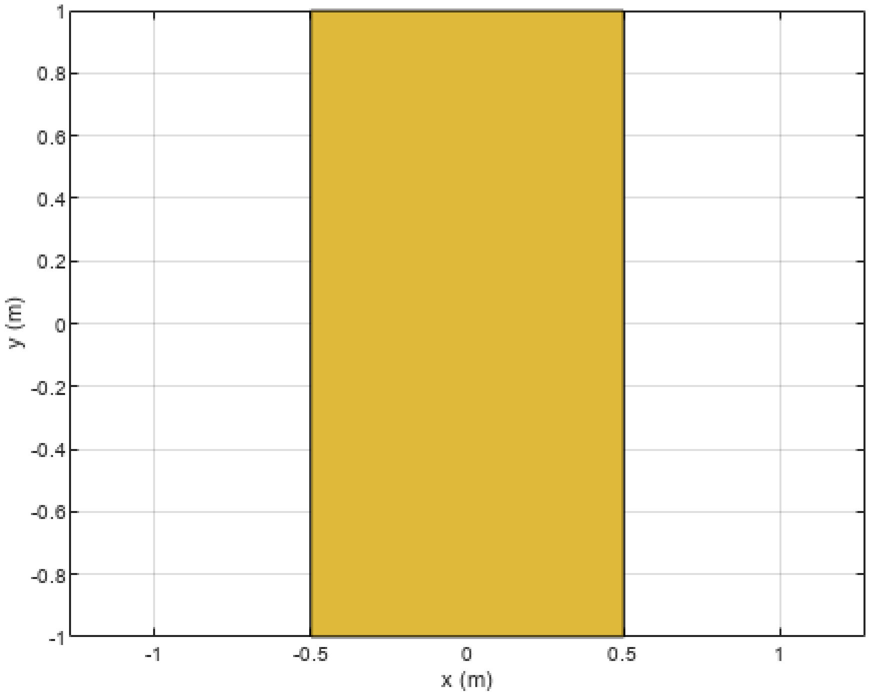
`c = rotate(shape,angle,axis1,axis2)` rotate shape about an axes object and angle.

Examples

Rotate Rectangle

Create a rectangle shape.

```
r = antenna.Rectangle;
show(r)
axis equal
```

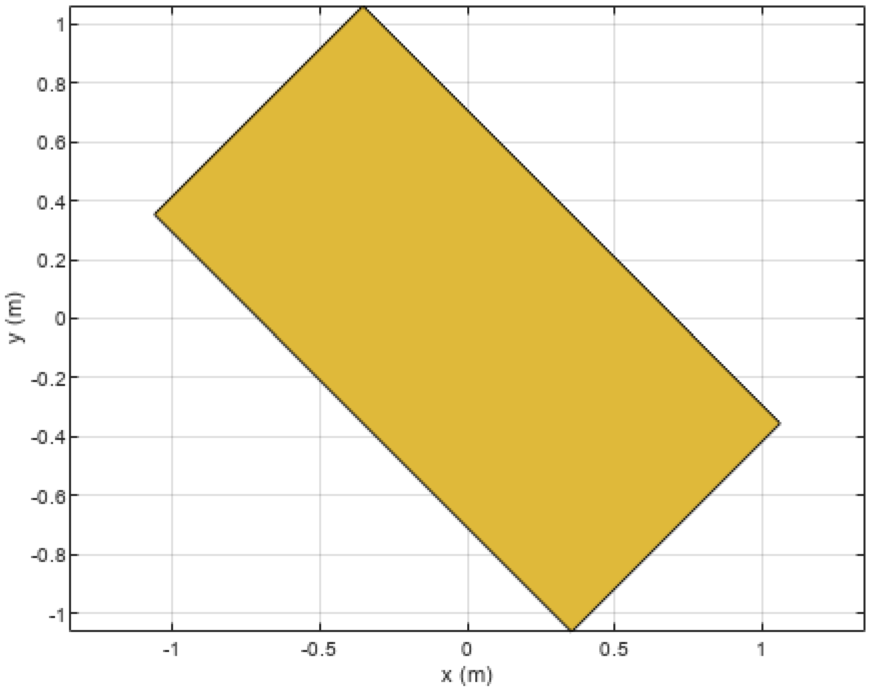


Rotate the rectangle at 45 degrees about the Z-axis.

```
r1 = rotate(r,45,[0 0 0],[0 0 1])
```

```
r1 = 
  Rectangle with properties:

         Name: 'myrectangle'
       Center: [0 0]
       Length: 1
        Width: 2
    NumPoints: 2
```

```
show(r1)
```



Input Arguments

shape — Shape created using custom elements and shape objects
object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `area(rectangle)` where rectangle is created using `antenna.Rectangle` object.

axis1,axis2 — Axis of rotation
two three-element vector of Cartesian coordinates in meters

Axis of rotation, specified as two unique three-element vectors of Cartesian coordinates in meters.

Example: `rotate(rectangle,45,[0 0 0], [0 0 1])` where rectangle is created using `antenna.Rectangle` object.

Data Types: `double`

angle — Angle of rotation
scalar

Angle of rotation, specified as a scalar in degrees.

Example: `rotate(rectangle,45,[0 0 1], [0 0 0])` rotates the rectangle around X-axis by 45 degrees.

Data Types: `double`

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `area` | `intersect` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

subtract

Boolean subtraction operation on two shapes

Syntax

```
c = subtract(shape1,shape2)
```

Description

`c = subtract(shape1,shape2)` subtracts `shape1` and `shape2` using the subtract operation. You can also use the `-` to subtract the two shapes.

Examples

Create Notched Rectangle

Create a rectangle with a length of 0.15 m, and a width of 0.15 m.

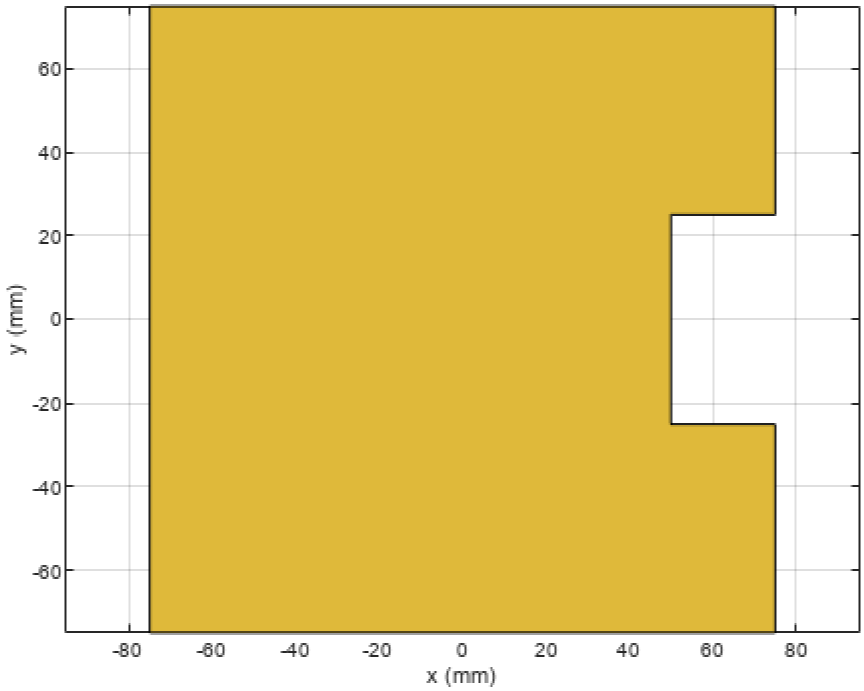
```
r  = antenna.Rectangle('Length',0.15,'Width',0.15);
```

Create a second rectangle with a length of 0.05 m, and a width of 0.05 m. Set the center of the second rectangle at half the length of the first rectangle r.

```
n = antenna.Rectangle('Center',[0.075,0],'Length',0.05,'Width',0.05);
```

Create and view a notched rectangle by subtracting n from r.

```
rn  = r-n;
show(rn)
```



Calculate the area of the notched rectangle.

```
area(rn)
```

```
ans = 0.0212
```

Input Arguments

shape1,shape2 — Shapes created using custom elements and shape objects
object

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `c = subtract(rectangle1, rectangle2)` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`add` | `area` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

gerberWrite

Generate Gerber files

Syntax

```
gerberWrite(designobject)
gerberWrite(designobject,rfconnector)
gerberWrite(designobject,writer)
gerberWrite(designobject,writer,rfconnector)
[a,g] = gerberWrite(designobject,writer,rfconnector)
```

Description

`gerberWrite(designobject)` creates a Gerber file from PCB specification files, such as `PCBWriter` object or `pcbStack` object.

Note To create associated files, run some kind of antenna analysis functions such as `show`, `pattern` etc. before running the `gerberWrite` function.

`gerberWrite(designobject,rfconnector)` creates Gerber file using specified RF connector.

`gerberWrite(designobject,writer)` creates a Gerber file using specified PCB writer services.

`gerberWrite(designobject,writer,rfconnector)` creates a Gerber file using specified PCB writer and connector services.

`[a,g] = gerberWrite(designobject,writer,rfconnector)` creates a Gerber file using specified PCB writer and connector services.

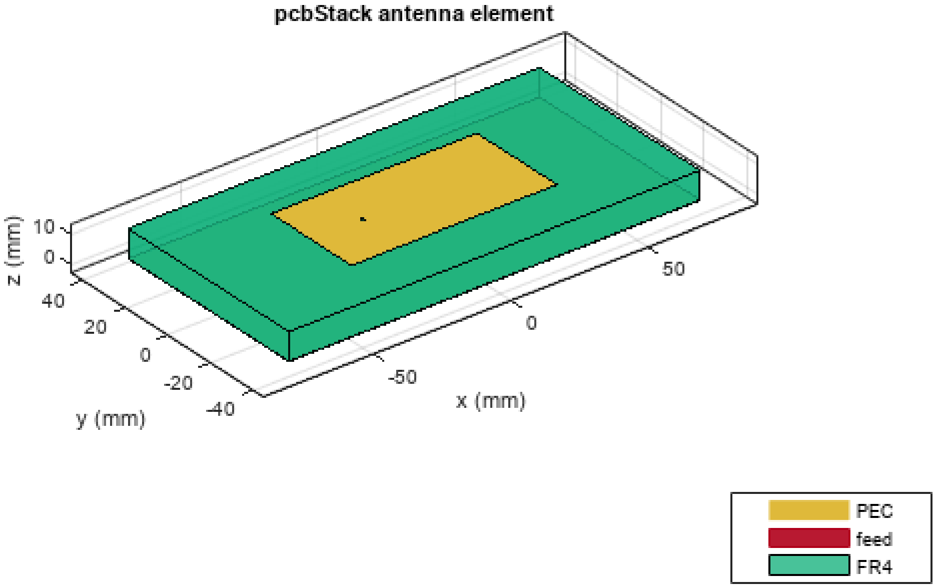
Note You can only use output arguments if the `designobject` is a `pcbStack` object.

Examples

Generate Antenna Gerber Files from PCB Stack

Create a patch antenna with FR4 as a dielectric material using `pcbStack` object.

```
p = pcbStack;
d = dielectric('FR4');
d.Thickness = p.BoardThickness;
p.Layers = {p.Layers{1},d,p.Layers{2}};
p.FeedLocations(3:4) = [1 3];
show(p)
```



Use a Cinch SMA for feeding the antenna. Use the Mayhew Labs PCB viewer as the 3-D viewer. Change the file name of the Mayhew Writer services to `antenna_design_file`.

```
C = PCBConnectors.SMA_Cinch;
W = PCBServices.MayhewWriter;
W.Filename = 'antenna_design_file';
```

Generate the Gerber-format files.

```
[A,g] = gerberWrite(p,W,C)
```

```
A = 
  PCBWriter with properties:

                      Design: [1x1 struct]
                      Writer: [1x1 PCBServices.MayhewWriter]
                   Connector: [1x1 PCBConnectors.SMA_Cinch]
         UseDefaultConnector: 0
    ComponentBoundaryLineWidth: 8
       ComponentNameFontSize: []
          DesignInfoFontSize: []
                        Font: 'Arial'
                   PCBMargin: 5.0000e-04
                  Soldermask: 'both'
                 Solderpaste: 1

   See info for details
```

```
g = 
'C:\TEMP\Bdoc22b_2054784_6060\ibB18F8B\26\tp90bc2890\antenna-ex85477975\antenna_design_file'
```

Show Antenna PCB Design Using Mayhew Manufacturing Service

Create a coplanar inverted F antenna.

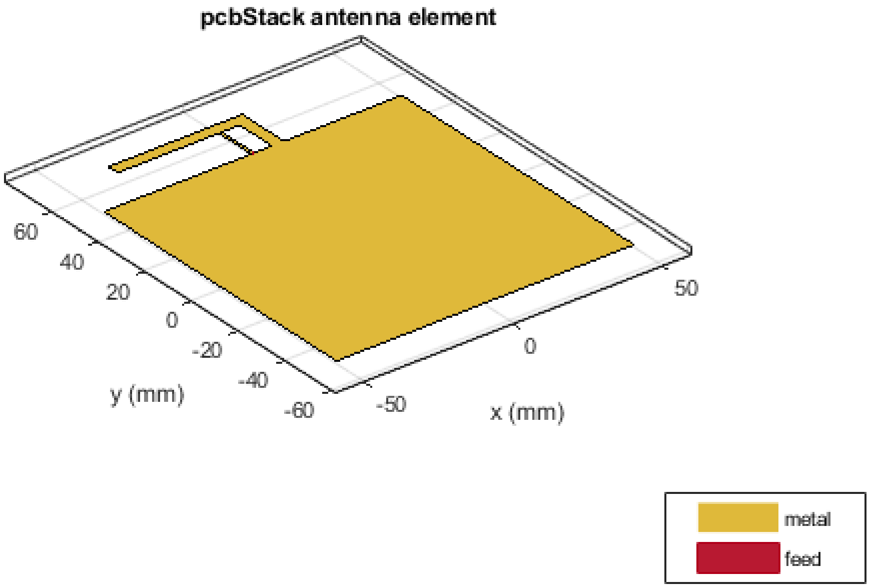
```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Use this antenna in creating a `pcbStack` object.

```
p = pcbStack(fco)
```

```
p = 
  pcbStack with properties:

              Name: 'Coplanar Inverted-F'
          Revision: 'v1.0'
        BoardShape: [1×1 antenna.Rectangle]
    BoardThickness: 0.0013
            Layers: {[1×1 antenna.Polygon]}
     FeedLocations: [0 0.0500 1]
      FeedDiameter: 5.0000e-04
      ViaLocations: []
       ViaDiameter: []
      FeedViaModel: 'strip'
       FeedVoltage: 1
         FeedPhase: 0
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1×1 lumpedElement]
```

```
figure
show(p)
```



Use an SMA_Cinch as an RF connector and Mayhew Writer as a 3-D viewer.

```
c = PCBConnectors.SMA_Cinch
```

```
c = 
  SMA_Cinch with properties:

                  Type: 'SMA'
                   Mfg: 'Cinch'
                  Part: '142-0711-202'
            Annotation: 'SMA'
             Impedance: 50
             Datasheet: 'https://belfuse.com/resources/Johnson/drawings/dr-142-0711-202.pdf'
              Purchase: 'https://www.digikey.com/product-detail/en/cinch-connectivity-solutions
             TotalSize: [0.0071 0.0071]
         GroundPadSize: [0.0024 0.0024]
     SignalPadDiameter: 0.0017
       PinHoleDiameter: 0.0013
         IsolationRing: 0.0041
   VerticalGroundStrips: 1

   Cinch 142-0711-202 (Example Purchase)
```

```
s = PCBServices.MayhewWriter
```

```
s = 
  MayhewWriter with properties:
```

```
               BoardProfileFile: 'legend'
          BoardProfileLineWidth: 1
                 CoordPrecision: [2 6]
                     CoordUnits: 'in'
              CreateArchiveFile: 0
                 DefaultViaDiam: 3.0000e-04
             DrawArcsUsingLines: 1
                 ExtensionLevel: 1
                       Filename: 'untitled'
                          Files: {}
         IncludeRootFolderInZip: 0
                   PostWriteFcn: @(obj)sendTo(obj)
    SameExtensionForGerberFiles: 0
                    UseExcellon: 1
```

Create an antenna design file using `PCBWriter` .

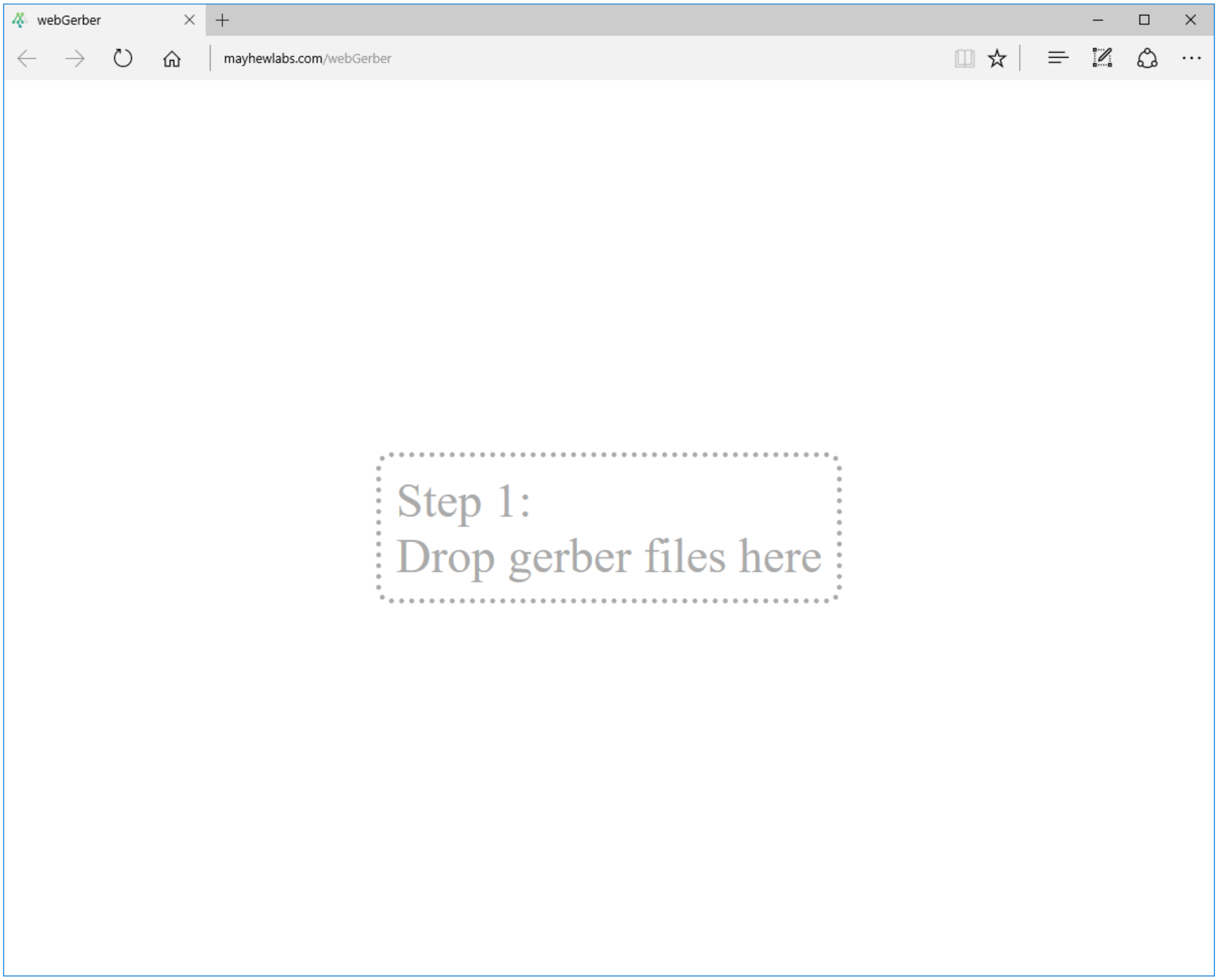
```
PW = PCBWriter(p,s,c)
```

```
PW = 
  PCBWriter with properties:

                        Design: [1×1 struct]
                        Writer: [1×1 PCBServices.MayhewWriter]
                     Connector: [1×1 PCBConnectors.SMA_Cinch]
           UseDefaultConnector: 0
    ComponentBoundaryLineWidth: 8
         ComponentNameFontSize: []
            DesignInfoFontSize: []
                          Font: 'Arial'
                     PCBMargin: 5.0000e-04
                    Soldermask: 'both'
                   Solderpaste: 1

   See info for details
```

Use the gerberWrite method to create gerber files from the antenna design files. The files generated are then send to the Mayhew writer manufacturing service.

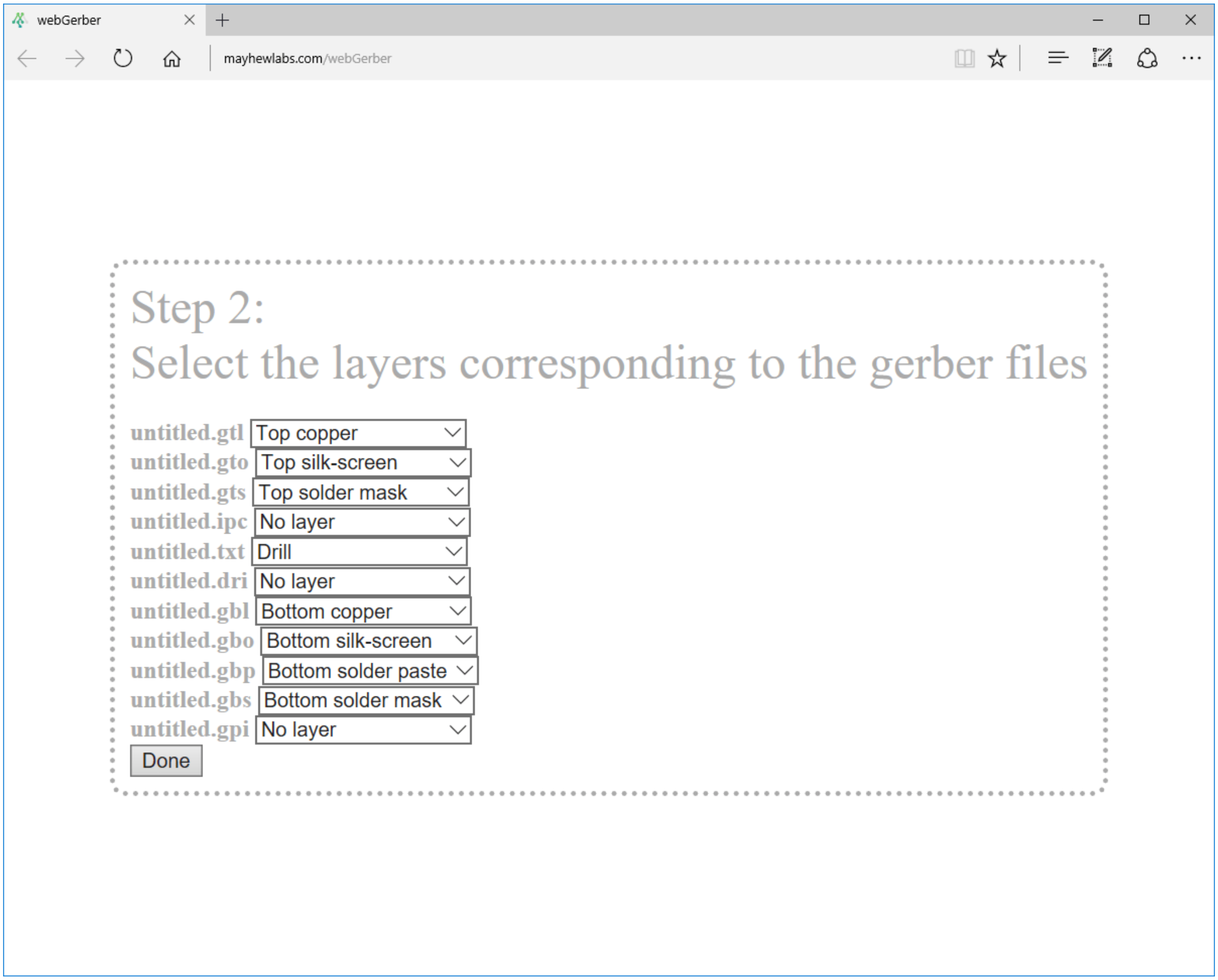
```
gerberWrite(PW)
```

By default, the folder containing the gerber files is called "untitled" and is located in your MATLAB folder. Running this example automatically opens up the Mayhew Labs PCB manufacturing service in your internet browser.



Step 1:
Drop gerber files here

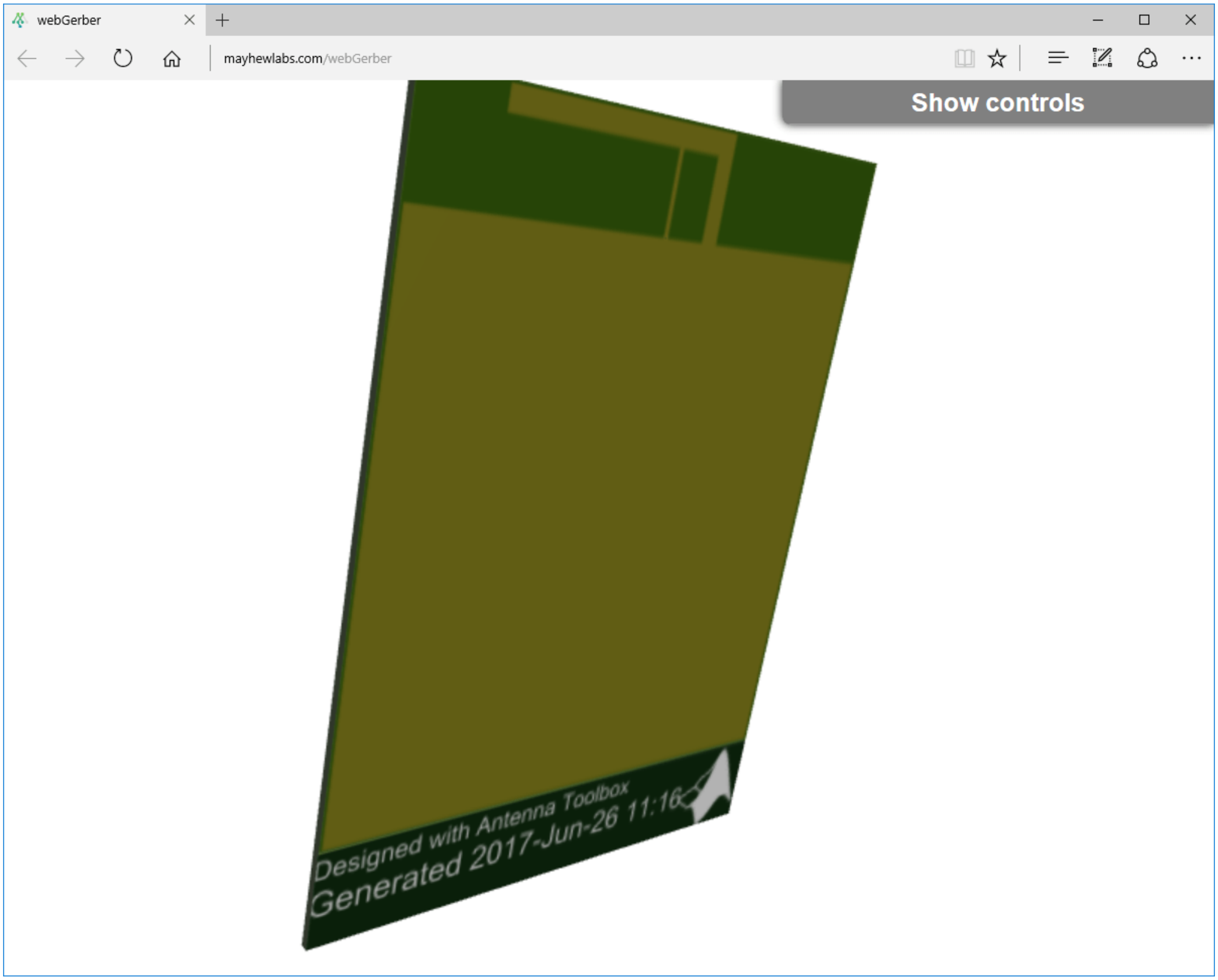
Drag and drop all your files from the "untitled" folder.



Step 2:
Select the layers corresponding to the gerber files

untitled.gtl	Top copper
untitled.gto	Top silk-screen
untitled.gts	Top solder mask
untitled.ipc	No layer
untitled.txt	Drill
untitled.dri	No layer
untitled.gbl	Bottom copper
untitled.gbo	Bottom silk-screen
untitled.gbp	Bottom solder paste
untitled.gbs	Bottom solder mask
untitled.gpi	No layer

Click **Done** to view your Antenna PCB.



Gerber Files of Antennas with Multiple Feeds

Design a patch antenna.

```
p = design(patchMicrostrip,3.5e9);
p.Width = p.Length;
p.Substrate = dielectric('FR4');
```

Create a stack representation of the patch antenna.

```
pb = pcbStack(p);
```

```
pb.FeedLocations = [pb.FeedLocations;-.007 0 1 3;0 .007 1 3;0 -.007 1 3];
```

Pick a connector for the feed locations.

```
C = SMA_Cinchcustom1;
```

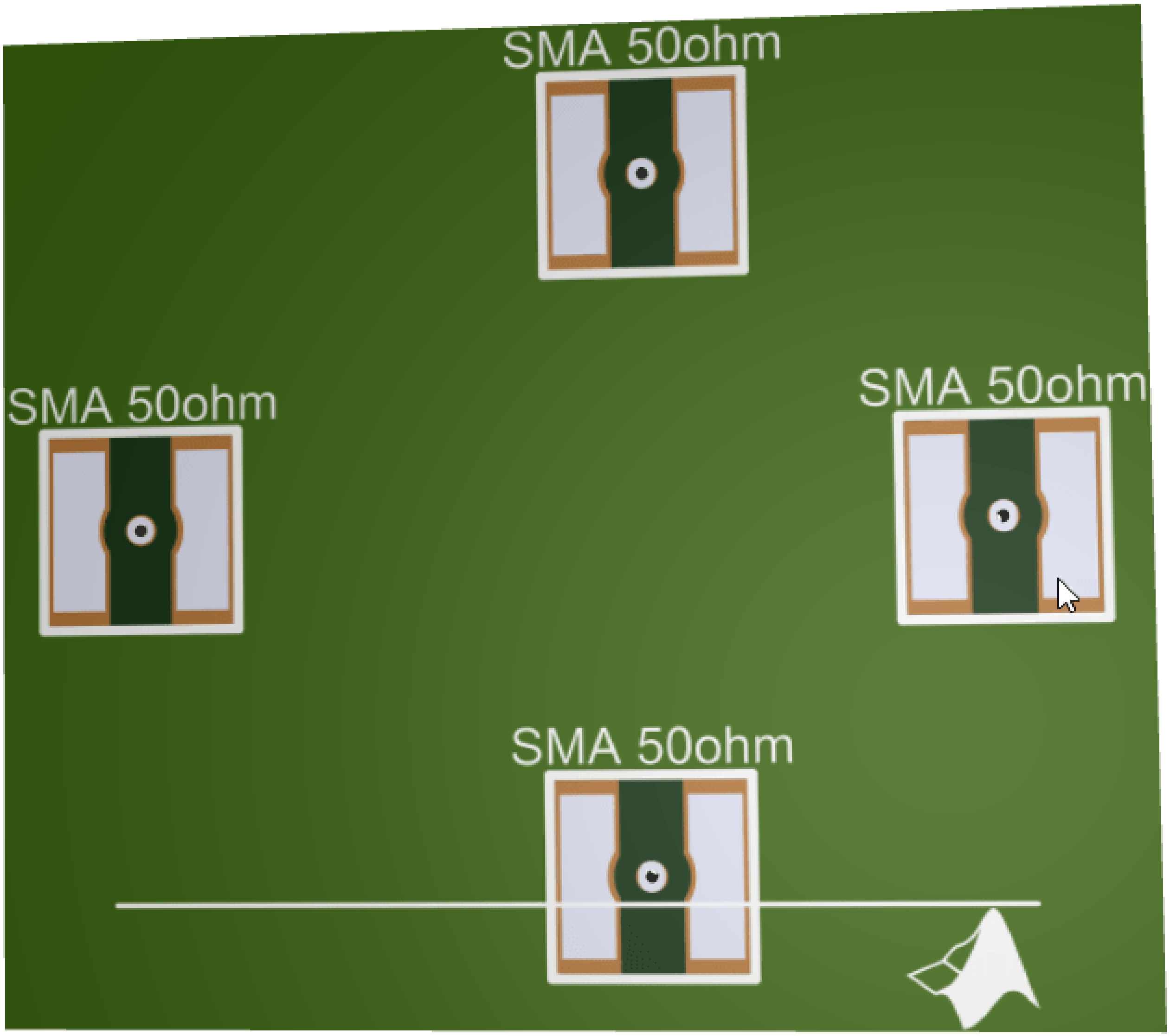
Pick a manufacturing service.

```
Wr = PCBServices.MayhewWriter;
```

Create a Gerber file and generate it.

```
A = PCBWriter(pb,Wr,C);
gerberWrite(A)
```

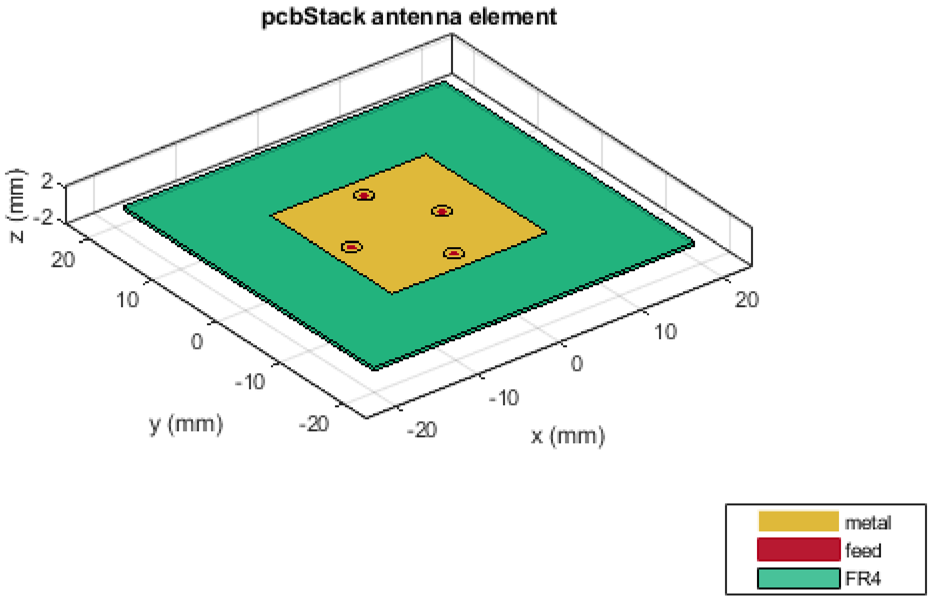
```
Warning: No metal specified for PCB
```



Gerber File Generation Using Multiple Connectors

Create a probe-fed microstrip patch antenna with four ports.

```
p = design(patchMicrostrip('Substrate',dielectric('FR4')),3.5e9);
p.Width = p.Length;
pb = pcbStack(p);
pb.FeedLocations = [pb.FeedLocations;-.007 0 1 3;0 .007 1 3;0 -.007 1 3];
figure
show(pb)
```



Pick a manufacturing service.

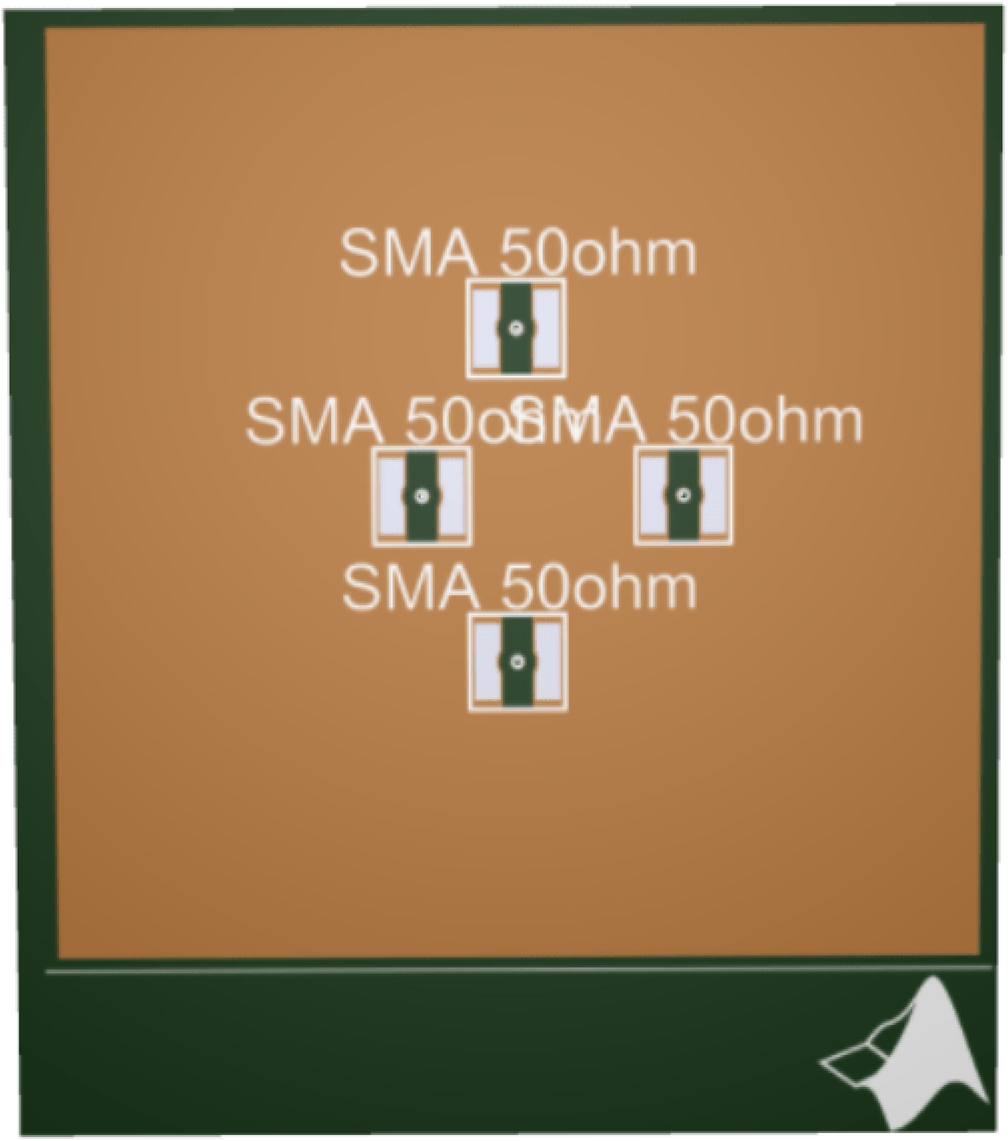
```
Wr = PCBServices.MayhewWriter;
Wr.Filename = 'Microstrip antenna-4ports';
```

Pick a connector for the feed locations.

```
C = SMA_Cinchcustom1;
```

Create a Gerber file and generate it.

```
A = PCBWriter(pb,Wr,C);
A.Soldermask = 'neither';
gerberWrite(A)
```

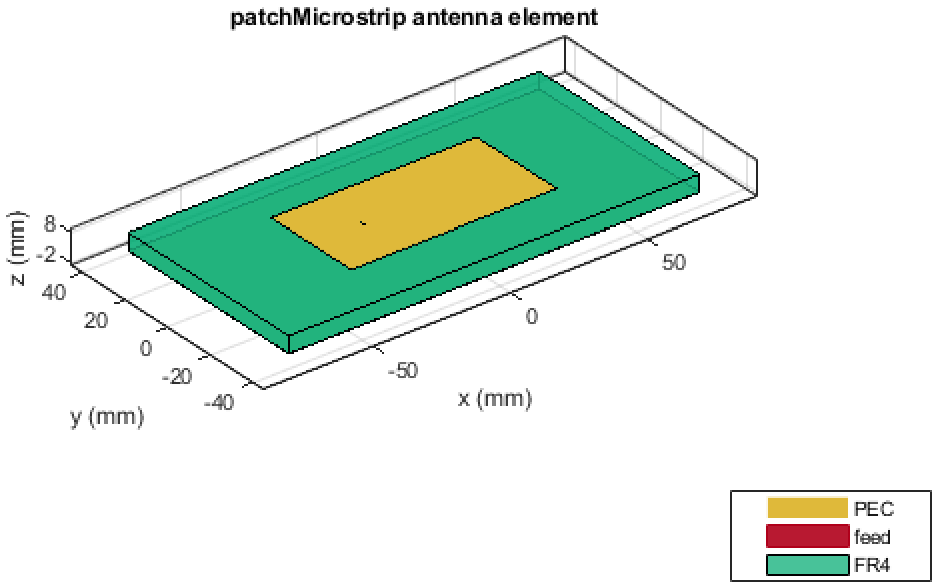


Gerber Files with No Connectors

Generate Gerber files with no connectors.

Create a microstrip antenna with FR4 dielectric substrate.

```
p = patchMicrostrip('Substrate',dielectric('FR4'));
show(p)
```



Create a PCB stack of this antenna.

```
pb = pcbStack(p);
```

Create an antenna design file using `PCBWriter` object.

```
s = PCBServices.MayhewWriter;
s.Filename = 'patchM';
PW = PCBWriter(pb,s);
```

Set the default connector to false and write the antenna to a Gerber file.

```
PW.UseDefaultConnector = 0;
gerberWrite(PW)
```

Input Arguments

`designobject` — Antenna design geometry file

`pcbStack` object | `PCBWriter` object

Antenna design geometry file, specified as a `pcbStack` object or `PCBWriter` object.

Example: `p1 = pcbStack` creates a PCB stack object.`p1` `gerberWrite(p1)` creates a Gerber file using `p1`.

Example: `p1 = pcbStack` creates a PCB stack object.`p1` `a = PCBWriter(p1)`, creates a `PCBWriter` object, `a`. `gerberWrite(a)`, creates a Gerber file using `a`.

`rfconnector` — RF connector type

`PCBConnector` object

RF connector type, specified as a `PCBConnector` object.

Example: `c = PCBConnectors.SMA_Cinch;gerberWrite(p1,c)` uses SMA_Cinch RF connector at the feedpoint.

`writer` — PCB service

`PCBServices` object

PCB service, specified as a `PCBServices` object.

Example: `s =PCBServices.MayhewWriter;gerberWrite(p1,s)` uses Mayhew Labs PCB service to create and view the PCB design.

Output Arguments

Note You can only use output arguments if the `designobject` is a `pcbStack` object.

a — PCBWriter object

object

`PCBWriter` object that generated the Gerber files, returned as an object.

g — Path to generated Gerber files folder

character vector

Path to generated Gerber files folder, returned as character vector.

Version History

Introduced in R2017b

See Also

`PCBServices` | `PCBConnectors`

openFolder

Open file browser to generated Gerber file folder

Syntax

```
openFolder(pcbWriterobject)
```

Description

`openFolder(pcbWriterobject)` opens the parent folder to the PCB writer Gerber design files. You use this function once the Gerber files are generated from the PCB Writer object using the `gerberWrite` function.

Examples

Location of Gerber Files

Create a coplanar inverted F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Use this antenna in creating a pcb stack object.

```
p = pcbStack(fco);
```

Use a SMA_Cinch as an RF connector and Mayhew Writer as a manufacturing service.

```
c = PCBConnectors.SMA_Cinch;
s = PCBServices.MayhewWriter;
```

Create an antenna design file using PCBWriter.

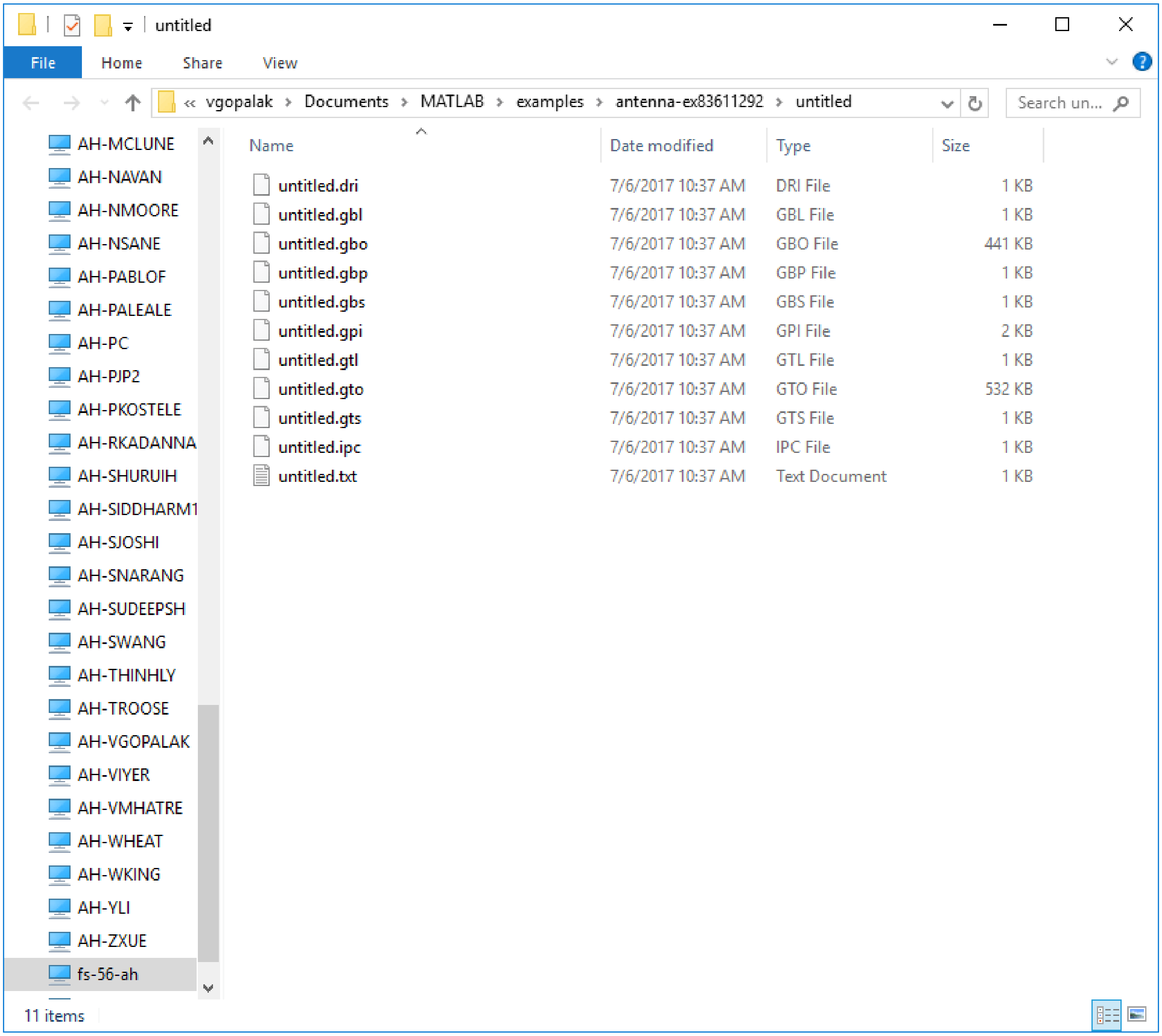
```
PW = PCBWriter(p,s,c);
```

Use the gerberWrite method to create Gerber files from the antenna design files.

```
gerberWrite(PW)
```

Open the folder that contains the Gerber files.

```
openFolder(PW)
```



Name	Date modified	Type	Size
untitled.dri	7/6/2017 10:37 AM	DRI File	1 KB
untitled.gbl	7/6/2017 10:37 AM	GBL File	1 KB
untitled.gbo	7/6/2017 10:37 AM	GBO File	441 KB
untitled.gbp	7/6/2017 10:37 AM	GBP File	1 KB
untitled.gbs	7/6/2017 10:37 AM	GBS File	1 KB
untitled.gpi	7/6/2017 10:37 AM	GPI File	2 KB
untitled.gtl	7/6/2017 10:37 AM	GTL File	1 KB
untitled.gto	7/6/2017 10:37 AM	GTO File	532 KB
untitled.gts	7/6/2017 10:37 AM	GTS File	1 KB
untitled.ipc	7/6/2017 10:37 AM	IPC File	1 KB
untitled.txt	7/6/2017 10:37 AM	Text Document	1 KB

Input Arguments

`pcbWriterobject` — Antenna design files

`PCBWriter` object

Antenna design files specified as a `PCBWriter` object.

Example: `p1 = pcbStack` creates a PCB stack object.`p1 a = PCBWriter(p1)`.

Version History

Introduced in R2017b

See Also

`info` | `sendTo` | `gerberWrite`

info

Display information about antenna or array

Syntax

```
info(antenna)
info(array)
```

Description

`info(antenna)` displays information about antenna element. as a structure:

- `isSolved` – Logical specifying if an antenna is solved.
- `isMeshed` – Logical specifying if an antenna is meshed.
- `MeshingMode` – String specifying the meshing mode.
- `HasSubstrate` – Logical specifying if an antenna uses a substrate.
- `HasLoad` – Logical specifying if an antenna has a load
- `PortFrequency` – Scalar or vector of frequencies used for port analysis.
- `FieldFrequency` – Scalar or vector of frequencies used for field analysis.
- `MemoryEstimate` – Approximate memory requirement for solving the antenna.

`info(array)` displays information about array element as a structure:.

- `isSolved` – Logical specifying if an array is solved.
- `isMeshed` – Logical specifying if an array is meshed.
- `MeshingMode` – String specifying the meshing mode.
- `HasSubstrate` – Logical specifying if an array uses a substrate.
- `HasLoad` – Logical specifying if an array has a load
- `PortFrequency` – Scalar or vector of frequencies used for port analysis.
- `FieldFrequency` – Scalar or vector of frequencies used for field analysis.
- `MemoryEstimate` – Approximate memory requirement for solving the array.

Examples

Antenna Information

Create a dipole antenna and calculate the impedance at 70 MHz.

```
d = dipole;
Z = impedance(d,70e6)
```

```
Z = 73.1597 + 0.1659i
```

Display all the information about the dipole antenna.

```
info(d)
```

```
ans = struct with fields:
          IsSolved: "true"
          IsMeshed: "true"
       MeshingMode: "auto"
      HasSubstrate: "false"
           HasLoad: "false"
     PortFrequency: 70000000
    FieldFrequency: []
    MemoryEstimate: "740 MB"
```

Input Arguments

`antenna` — Antenna element

antenna object

Antenna element, specified as an antenna object.

Example: `d = dipole`;

`array` — Array element

array object

Array element, specified as an array object.

Example: `d = dipole`;

Version History

Introduced in R2017b

See Also

`show`

sendTo

Export generated Gerber Files to service provider

Syntax

```
sendTo(pcbWriterobject)
```

Description

`sendTo(pcbWriterobject)` opens the manufacturing service browser page on your default web browser and opens the folder containing the Gerber files.

For example, if the manufacturing service is MayhewWriter, then `sendTo` action opens the Mayhew Labs online PCB viewer in your default web browser. This function also opens the folder containing the Gerber files. This simplifies use of the service, enabling you to drag and drop the files to the website and view the design.

Examples

Open Manufacturing Service Website

Create a coplanar inverted F antenna.

```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

Use this antenna in creating a pcb stack object.

```
p = pcbStack(fco);
```

Use a SMA_Cinch as an RF connector and Mayhew Writer as a manufacturing service.

```
c = PCBConnectors.SMA_Cinch;
s = PCBServices.MayhewWriter;
```

Create an antenna design file using PCBWriter.

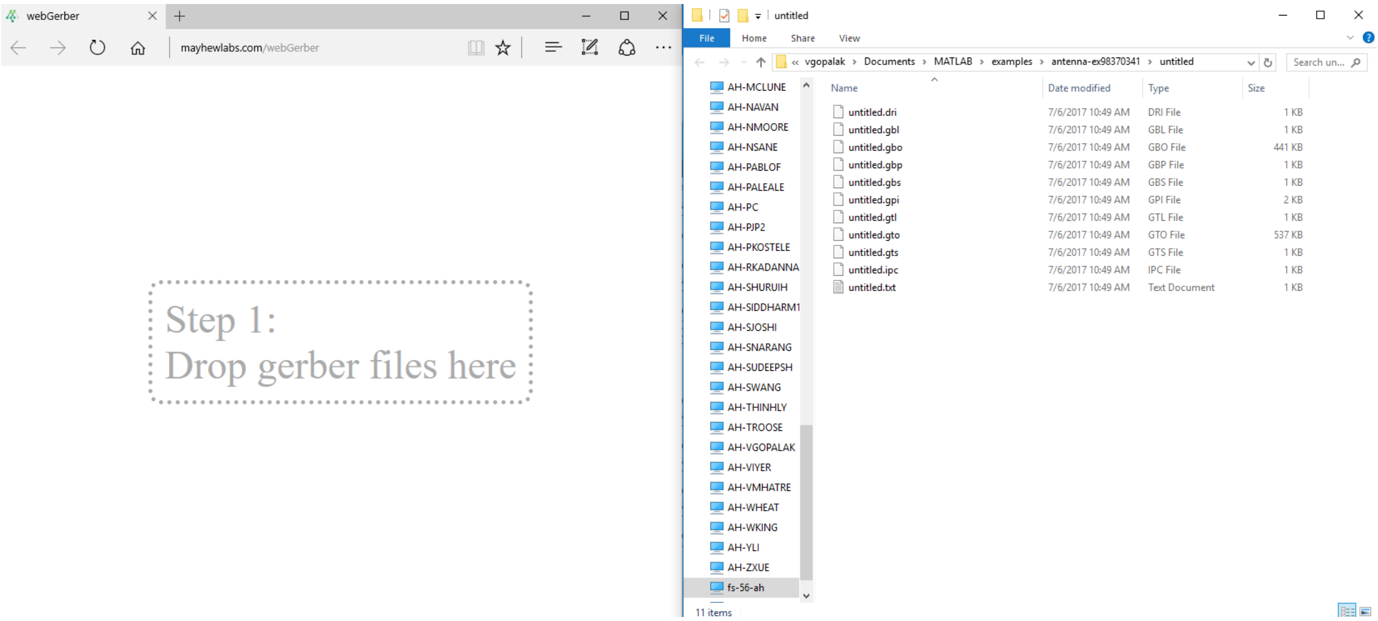
```
PW = PCBWriter(p,s,c);
```

Use the gerberWrite method to create Gerber files from the antenna design files.

```
gerberWrite(PW)
```

Open the manufacturing service website to send the Gerber files.

```
sendTo(PW)
```



Name	Date modified	Type	Size
untitled.dri	7/6/2017 10:49 AM	DRI File	1 KB
untitled.gbl	7/6/2017 10:49 AM	GBL File	1 KB
untitled.gbo	7/6/2017 10:49 AM	GBO File	441 KB
untitled.gbp	7/6/2017 10:49 AM	GBP File	1 KB
untitled.gbs	7/6/2017 10:49 AM	GBS File	1 KB
untitled.gpi	7/6/2017 10:49 AM	GPI File	2 KB
untitled.gtl	7/6/2017 10:49 AM	GTL File	1 KB
untitled.gto	7/6/2017 10:49 AM	GTO File	537 KB
untitled.gts	7/6/2017 10:49 AM	GTS File	1 KB
untitled.ipc	7/6/2017 10:49 AM	IPC File	1 KB
untitled.txt	7/6/2017 10:49 AM	Text Document	1 KB

Input Arguments

`pcbWriterobject` — Antenna design files

`PCBWriter` object

Antenna design files, specified as a `PCBWriter` object.

Example: `p1 = pcbStack` creates a PCB stack object.`p1 a = PCBWriter(p1)`.

Version History

Introduced in R2017b

See Also

`gerberWrite` | `info` | `sendTo`

getLowPassLocs

Feeding location to operate birdcage as lowpass coil

Syntax

```
getLowPassLocs(birdcageantenna)
```

Description

`getLowPassLocs(birdcageantenna)` returns all the feed locations on the birdcage to operate the antenna as a lowpass coil. The feeding locations are located in the center of the rungs. Use this function to find the `FeedLocations` property value for `birdcage`.

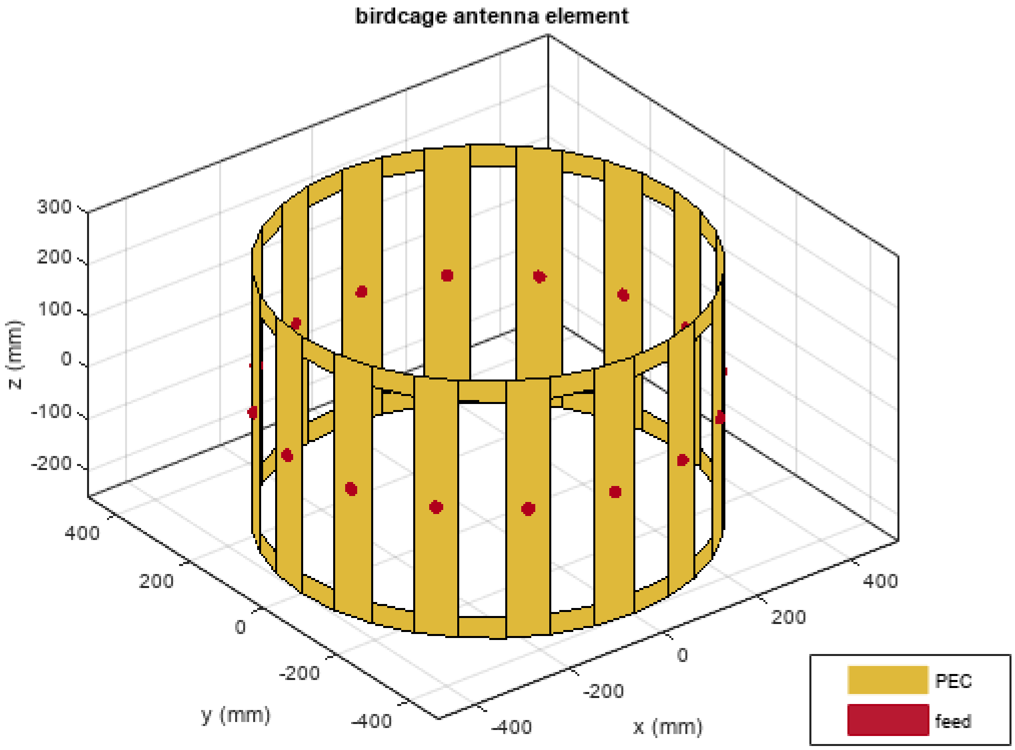
Examples

Birdcage as Lowpass Coil

```
b = birdcage;
b.FeedLocations = getLowPassLocs(b)
```

```
b = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: []
    FeedLocations: [16x3 double]
      FeedVoltage: 1
        FeedPhase: 0
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

```
show(b)
```



Input Arguments

`birdcageantenna` — Birdcage antenna
object

Birdcage antenna, specified as an object.

Example: `b = birdcage` `b.FeedLocations = getLowPassLocs(b)`

Version History

Introduced in R2017b

See Also

getHighPassLocs

Feeding location to operate birdcage as highpass coil

Syntax

```
getHighPassLocs(birdcageantenna)
```

Description

`getHighPassLocs(birdcageantenna)` returns all the feed locations on the birdcage to operate the antenna as a highpass coil. The feeding locations are along the circumference on the upper and lower coils of the birdcage. Use this function to find the `FeedLocations` property value for `birdcage`.

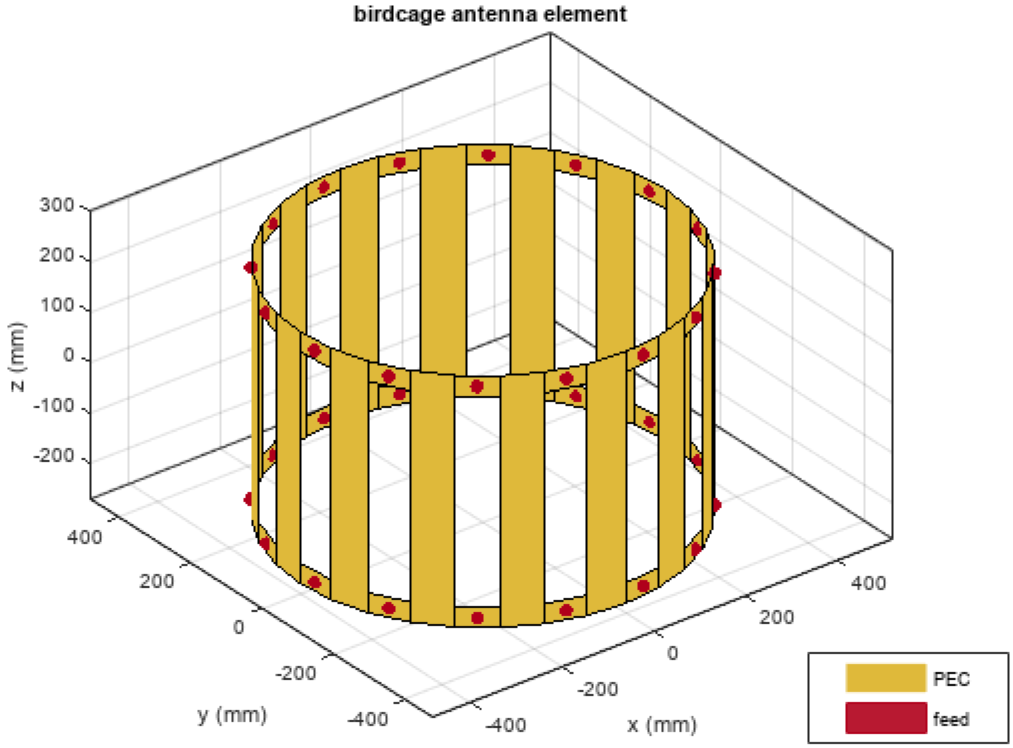
Examples

Birdcage as Highpass Coil

```
b = birdcage;
b.FeedLocations = getHighPassLocs(b)
```

```
b = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: []
    FeedLocations: [32x3 double]
      FeedVoltage: 1
        FeedPhase: 0
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

```
show(b)
```



Input Arguments

`birdcageantenna` — Birdcage antenna
object

Birdcage antenna, specified as an object.

Example: `b = birdcage b.FeedLocations = getHighPassLocs(b)`

Version History

Introduced in R2017b

See Also

rotateX

Rotate shape about *x*-axis and angle

Syntax

```
rotateX(shape,angle)
c =c rotateX(shape,angle)
```

Description

`rotateX(shape,angle)` rotate shape about an axes object and angle.

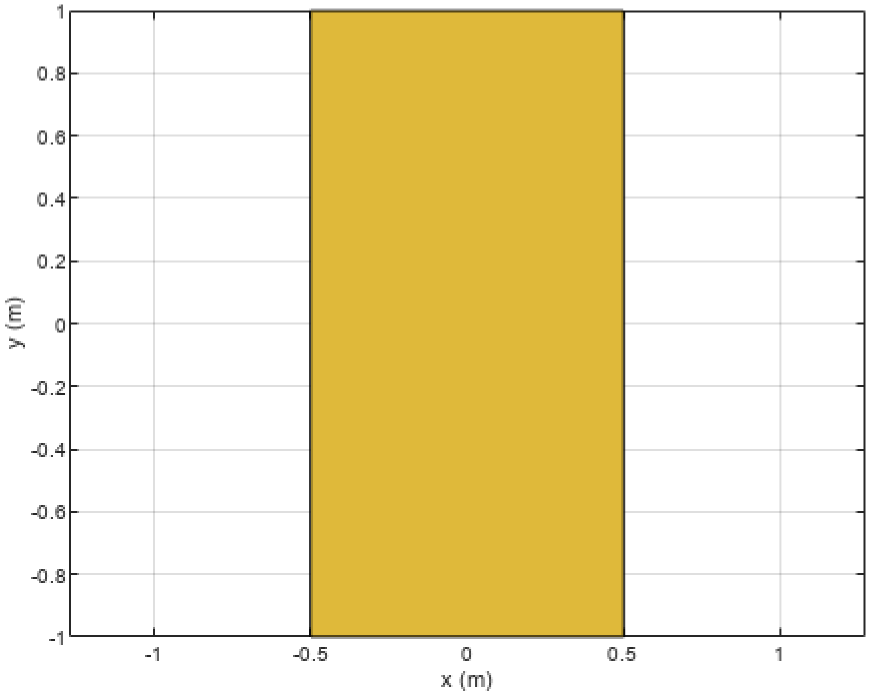
`c =c rotateX(shape,angle)` rotate shape about an axes object and angle.

Examples

Rotate Rectangle About X-Axis

Create a rectangle shape.

```
r = antenna.Rectangle;
show(r)
axis equal
```

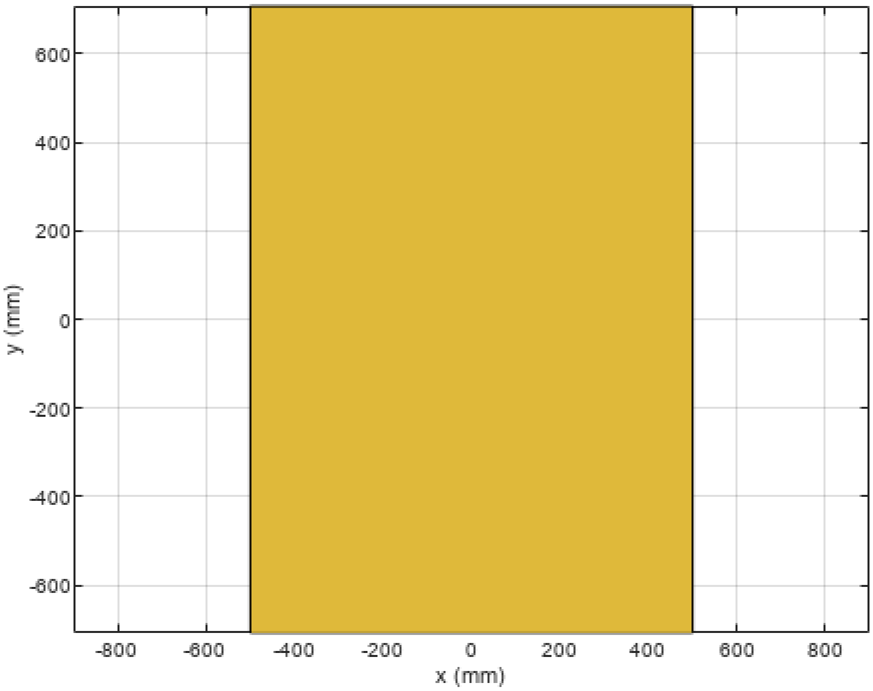


Rotate the rectangle at 45 degrees about the x-axis.

```
r1 = rotateX(r,45)
```

```
r1 = 
  Rectangle with properties:

         Name: 'myrectangle'
       Center: [0 0]
       Length: 1
        Width: 2
    NumPoints: 2
```

```
show(r1)
axis equal
```



Input Arguments

shape — Shape created using custom elements and shape objects

object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `area(rectangle)` where rectangle is created using `antenna.Rectangle` object.

angle — Angle of rotation

scalar

Angle of rotation, specified as a scalar in degrees.

Example: `rotateX(rectangle,45)` rotates the rectangle around X-axis by 45 degrees.

Data Types: `double`

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `area` | `intersect` | `rotate` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

rotateY

Rotate shape about *y*-axis and angle

Syntax

```
rotateY(shape,angle)
c = rotateY(shape,angle)
```

Description

`rotateY(shape,angle)` rotate shape about the *y*-axis and angle.

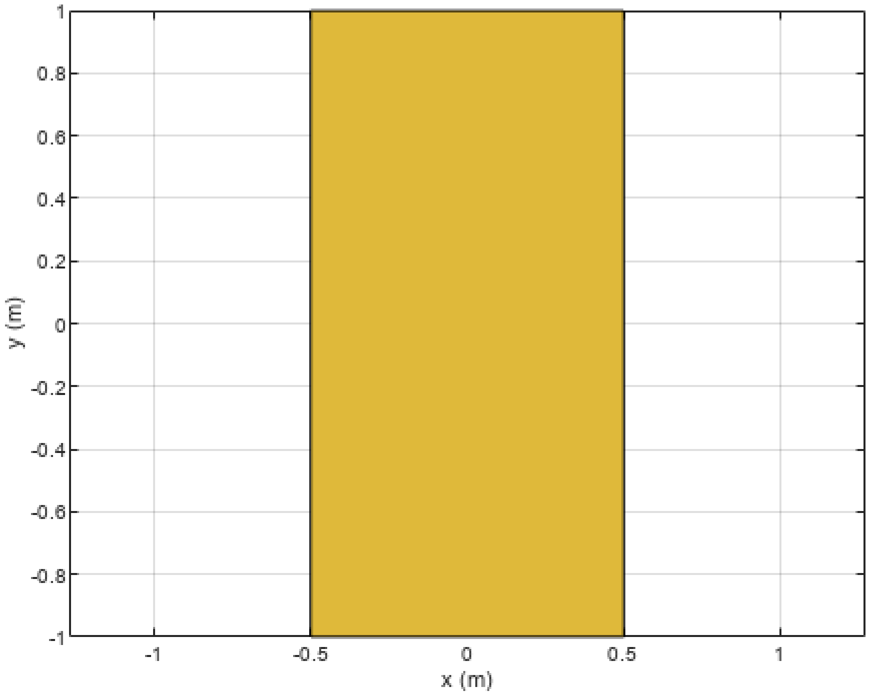
`c = rotateY(shape,angle)` rotate shape about the *y*-axis and angle.

Examples

Rotate Rectangle About Y-Axis

Create a rectangle shape.

```
r = antenna.Rectangle;
show(r)
axis equal
```

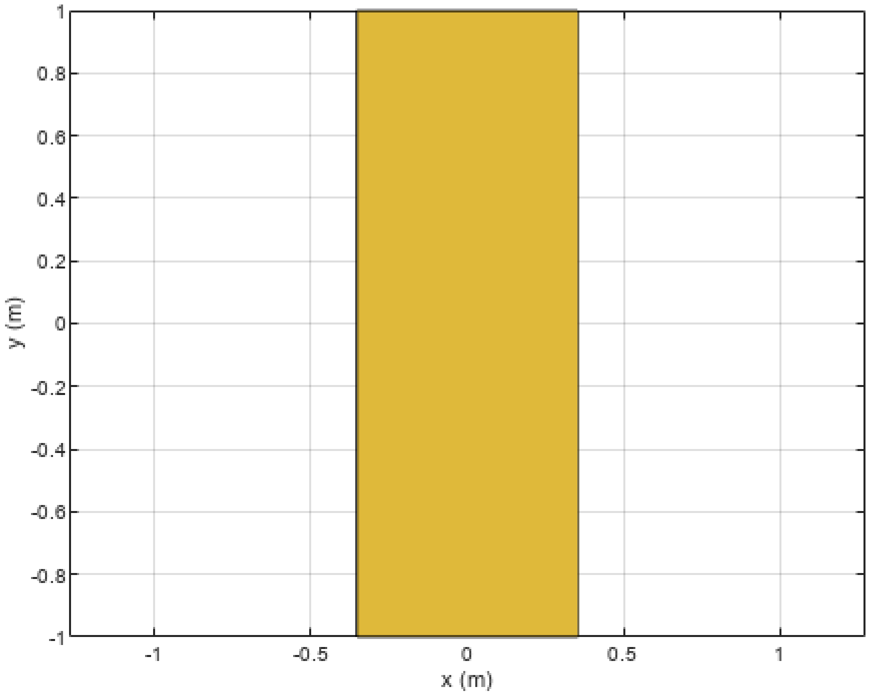


Rotate the rectangle at 45 degrees about the Y-axis.

```
r1 = rotateY(r,45)
```

```
r1 = 
  Rectangle with properties:

         Name: 'myrectangle'
       Center: [0 0]
       Length: 1
        Width: 2
    NumPoints: 2
```

```
show(r1)
axis equal
```



Input Arguments

`shape` — Shape created using custom elements and shape objects

object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `rotateY(rectangle)` where rectangle is created using `antenna.Rectangle` object.

`angle` — Angle of rotation

scalar

Angle of rotation, specified as a scalar in degrees.

Example: `rotateY(rectangle,45)` rotates the rectangle around *y*-axis by 45 degrees.

Data Types: `double`

Version History

Introduced in R2017a

See Also

add | subtract | area | intersect | rotate | rotateX | rotateZ | translate | show | mesh | plot | scale

rotateZ

Rotate shape about *z*-axis and angle

Syntax

```
rotateZ(shape,angle)
c = rotateZ(shape,angle)
```

Description

`rotateZ(shape,angle)` rotate shape about the *z*-axis and angle.

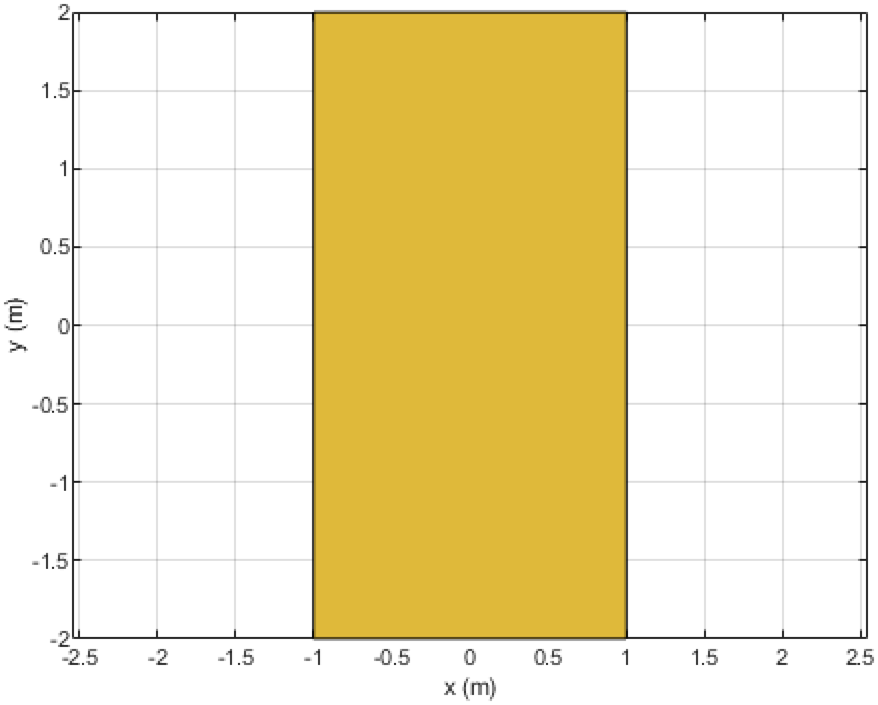
`c = rotateZ(shape,angle)` rotate shape about the *z*-axis and angle.

Examples

Create and Rotate Rectangle Using Specified Properties

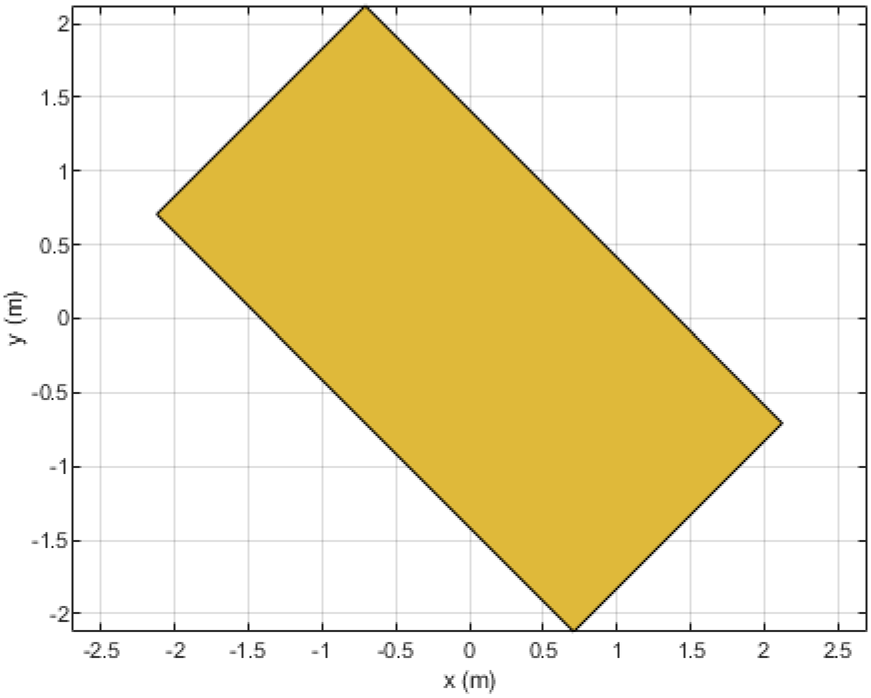
Create and view a rectangle with a length of 2 m and a width of 4 m.

```
r2 = antenna.Rectangle('Length',2,'Width',4);
show(r2)
axis equal
```



Rotate the rectangle.

```
rotateZ(r2,45);
show(r2)
```



Input Arguments

shape — Shape created using custom elements and shape objects
object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `rotateZ(rectangle)` where rectangle is created using `antenna.Rectangle` object.

angle — Angle of rotation
scalar

Angle of rotation, specified as a scalar in degrees.

Example: `rotateZ(rectangle,45)` rotates the rectangle around *z*-axis by 45 degrees.

Data Types: `double`

Version History
Introduced in R2017a

See Also
`add` | `subtract` | `area` | `intersect` | `rotate` | `rotateX` | `rotateY` | `translate` | `show` | `mesh` | `plot` | `scale`

translate

Move shape to new location

Syntax

```
c = translate(shape,locationpoints)
```

Description

`c = translate(shape,locationpoints)` moves the shape to a new specified location using a translation vector.

Examples

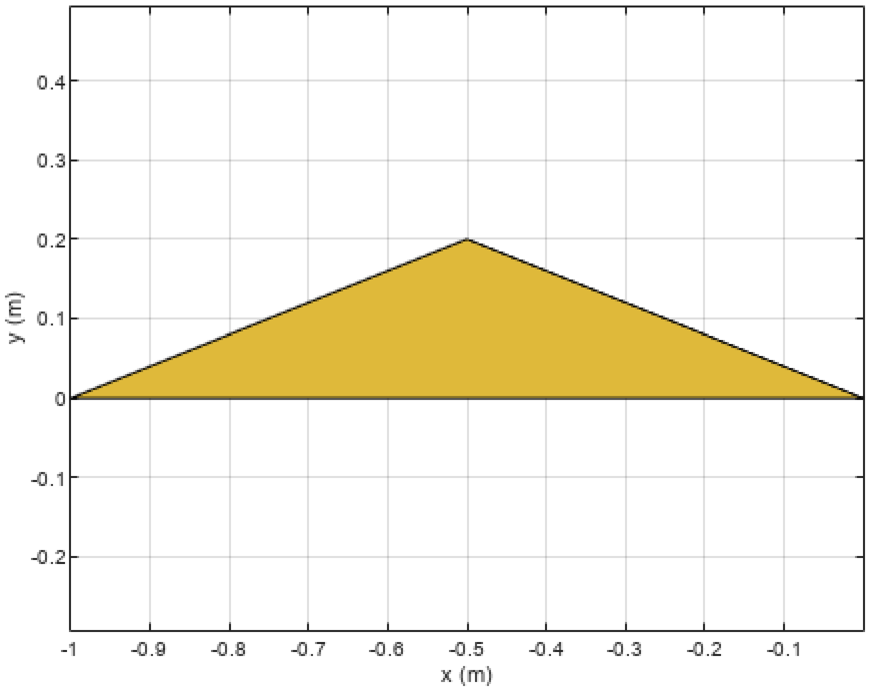
Create and Transform Polygon

Create a polygon using `antenna.Polygon` with vertices at `[-1 0 0;-0.5 0.2 0;0 0 0]` and view it.

```
p = antenna.Polygon('Vertices', [-1 0 0;-0.5 0.2 0;0 0 0])
```

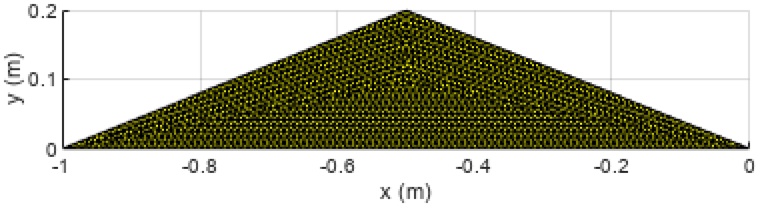
```
p = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [3x3 double]
```

```
show(p)
axis equal
```



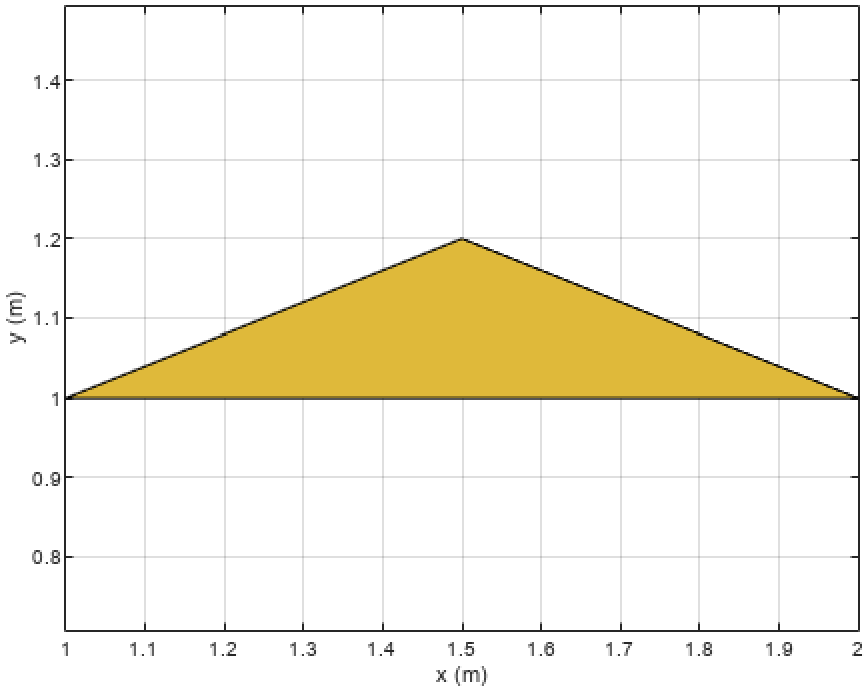
Mesh the polygon and view it.

```
mesh(p,0.2)
```



Move the polygon to a new location on the X-Y plane.

```
translate(p,[2,1,0])
axis equal
```



Input Arguments

`shape` — Shape created using custom elements and shape objects

object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `c = translate(rectangle1,[2 1 0])` where rectangle1 is created using `antenna.Rectangle` object.

`locationpoints` — Translation vector

vector

Translation vector, specified as a vector.

Data Types: `double`

Version History

Introduced in R2017a

See Also

add | subtract | area | intersect | rotate | rotateX | rotateY | rotateZ | show | mesh | plot | scale

plot

Plot boundary of shape

Syntax

```
p = plot(shape,varargin)
```

Description

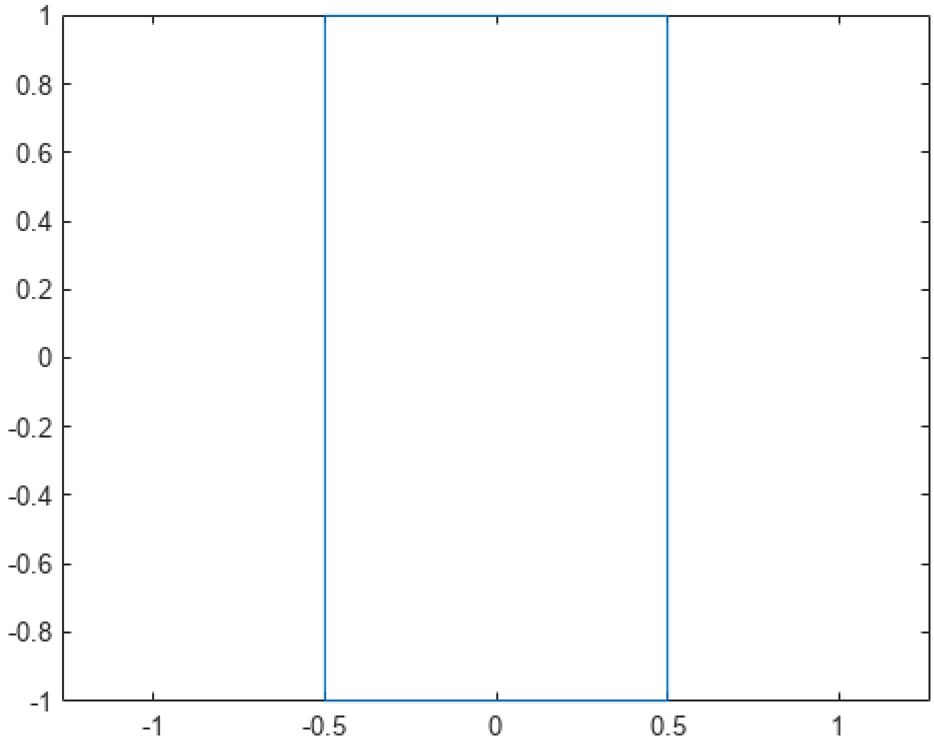
`p = plot(shape,varargin)` plots the boundary of the shape and returns the line.

Examples

Plot Rectangle Shape

Create a rectangular shape and plot it.

```
r = antenna.Rectangle;
p = plot(r);
```



Input Arguments

shape — Shape created using custom elements and shape objects
object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `plot(rectangle)` where rectangle is created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`show` | `mesh`

scale

Change the size of the shape by a fixed amount

Syntax

```
c = scale(shape,scaling)
```

Description

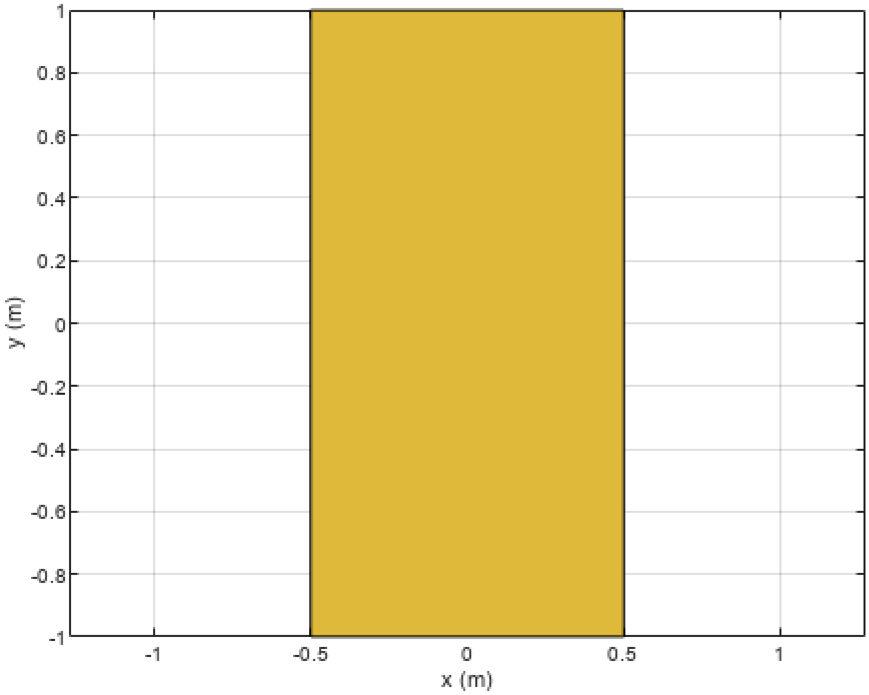
`c = scale(shape,scaling)` scales the shape by a constant factor

Examples

Scale Rectangle Shape

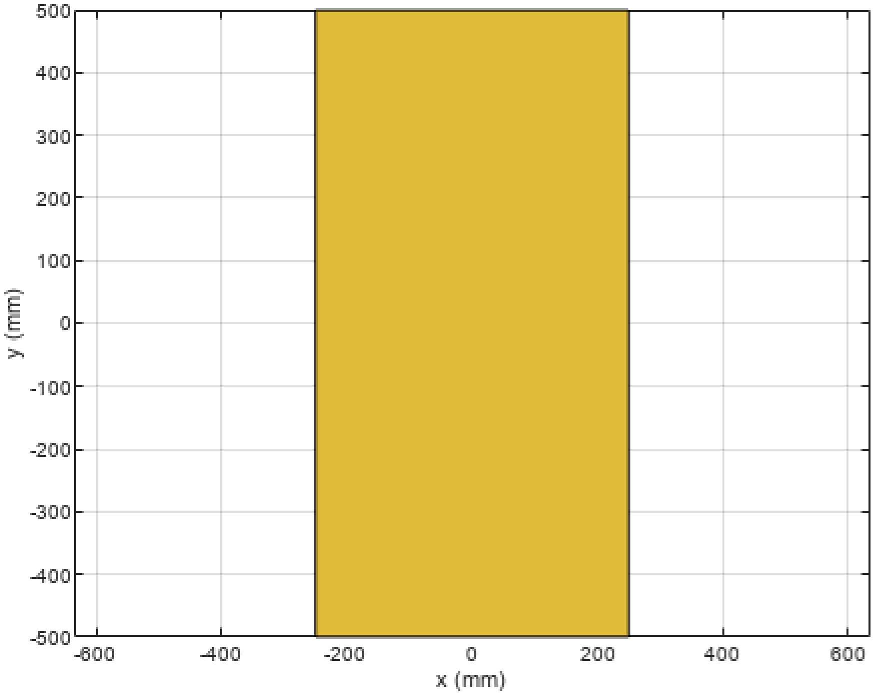
Create a rectangular shape.

```
r  = antenna.Rectangle;
show(r)
axis equal
```



Shrink the rectangle by 50%.

```
scale(r,0.5);
```



Input Arguments

shape — Shape created using custom elements and shape objects

object

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `c = scale(rectangle1,0.5)` where rectangle1 is created using `antenna.Rectangle` object.

`scaling` — Constant factor to change shape size

scalar

Constant factor to change shape size, specified as a scalar.

Data Types: `double`

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `area` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `show` | `mesh` | `plot`

plus

Shape1 + Shape2

Syntax

```
c = plus(shape1,shape2)
```

Description

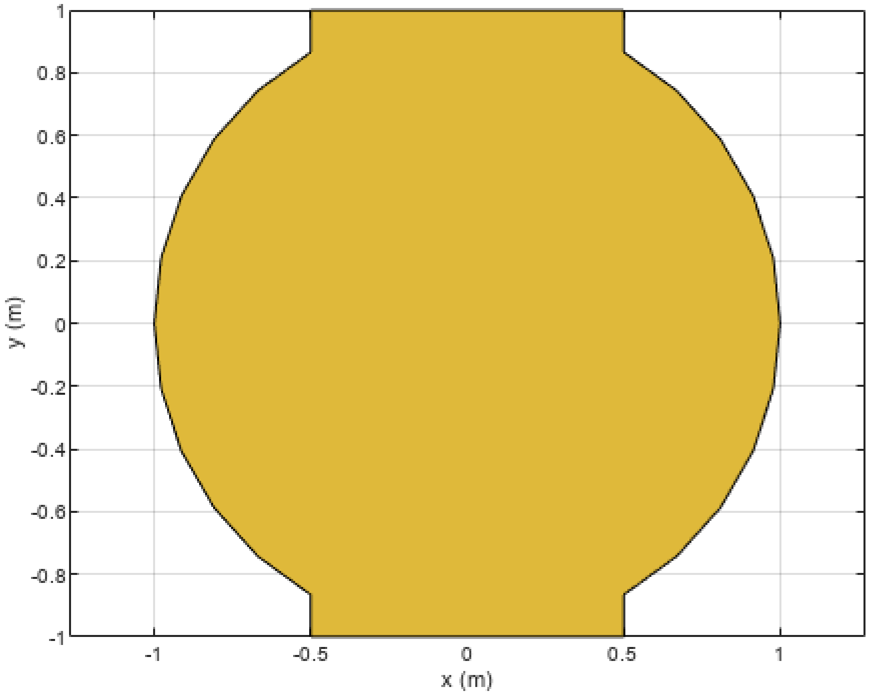
`c = plus(shape1,shape2)` calls the syntax `shape1 + shape2` to unite two shapes.

Examples

Unite Rectangle and Circle

Create a rectangular and circular shape and unite them.

```
r  = antenna.Rectangle;
c  = antenna.Circle;
r+c;
```



Input Arguments

`shape1,shape2` — Shapes created using custom elements and shape objects

object

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `rectangle1+rectangle2` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `area` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

minus

Shape1 - Shape2

Syntax

```
c = minus(shape1,shape2)
```

Description

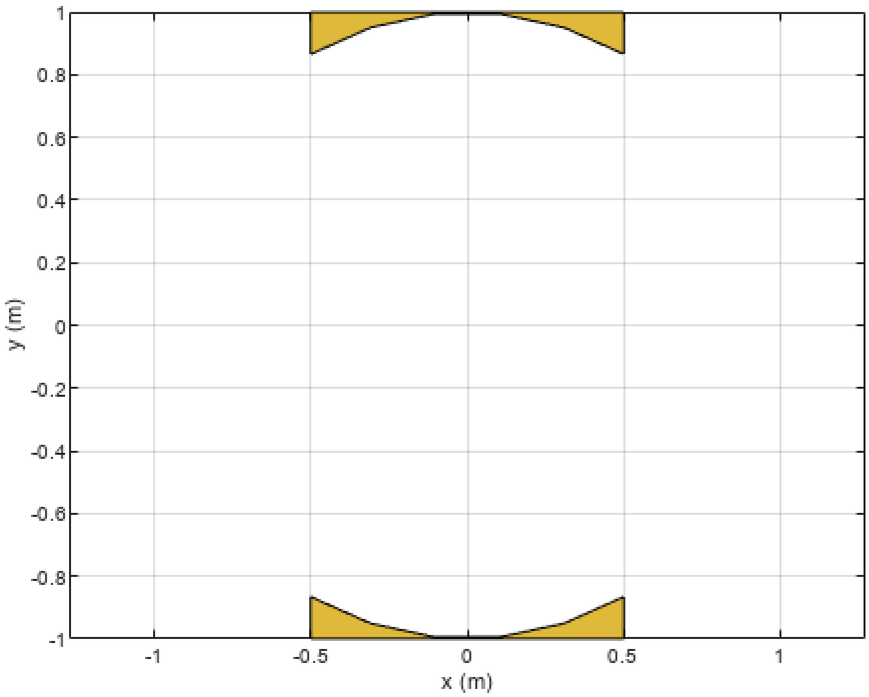
`c = minus(shape1,shape2)` calls the syntax `shape1 - shape2` to subtract two shapes.

Examples

Subtract Rectangle and Circle

Create a rectangular and circular shape and subtract them.

```
r  = antenna.Rectangle;
c  = antenna.Circle;
r-c;
```



Input Arguments

`shape1,shape2` — Shapes created using custom elements and shape objects
object

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `rectangle1-rectangle2` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `area` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

and

Shape1 & Shape2

Syntax

`c = and(shape1,shape2)`

Description

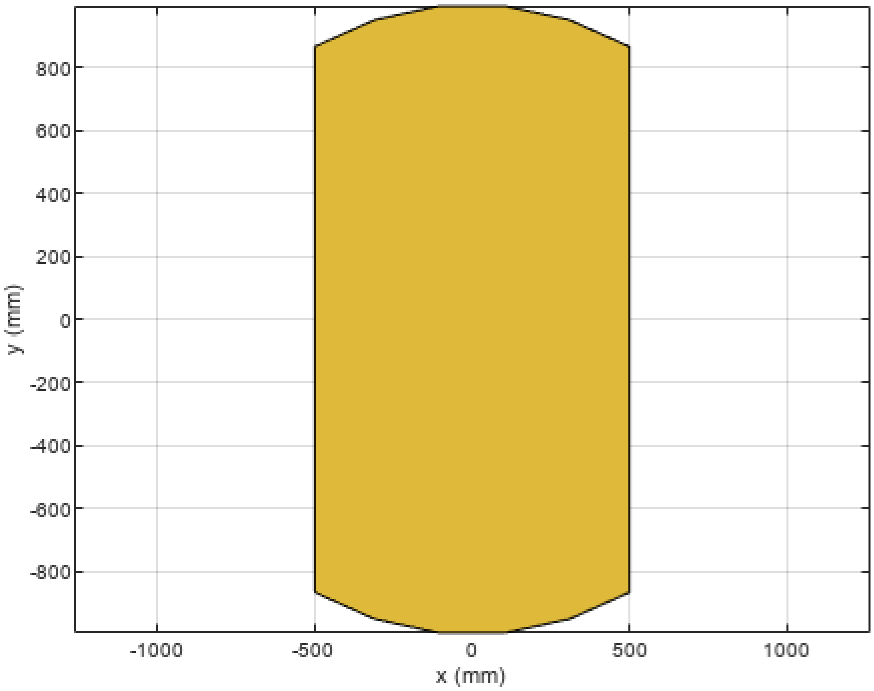
`c = and(shape1,shape2)` calls the syntax `shape1 & shape2` to intersect two shapes.

Examples

Intersect Rectangle and Circle

Create a rectangular and circular shape and intersect them.

```
r  = antenna.Rectangle;
c  = antenna.Circle;
r&c;
```



Input Arguments

`shape1,shape2` — Shapes created using custom elements and shape objects

object

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object.

Example: `rectangle1&rectangle2` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

Version History

Introduced in R2017a

See Also

`add` | `subtract` | `area` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

add

Add additional data to existing Smith chart

Syntax

```
add(plot,data)
add(plot,frequency,data)
```

Description

`add(plot,data)` adds data to an existing Smith chart.

`add(plot,frequency,data)` adds data to an existing Smith chart based on multiple data sets containing frequencies corresponding to columns of data matrix.

Examples

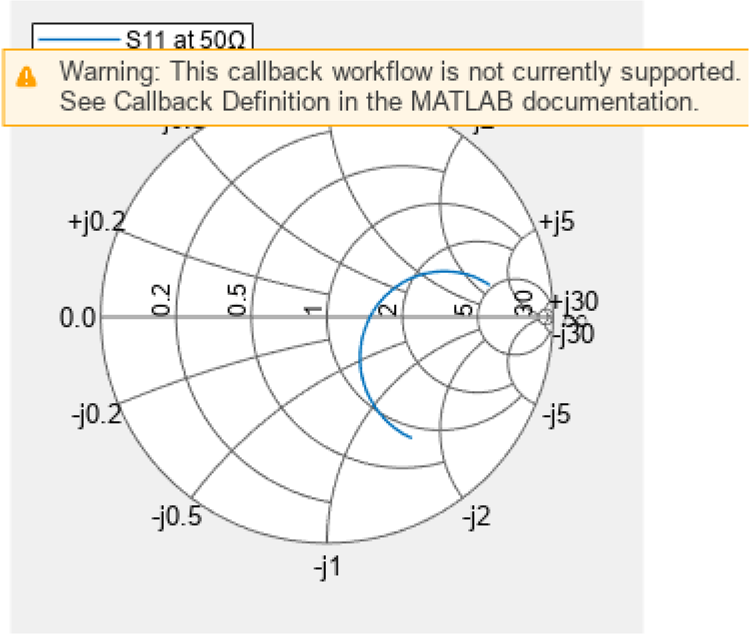
Add S-Parameter Data to Existing Smith Plot

Plot the reflection coefficients of a dipole antenna.

Create a strip dipole antenna on the Y-Z plane. Calculate the complex s-parameters of the dipole antenna from 60 MHz to 90 MHz, with an interval of 150 kHz.

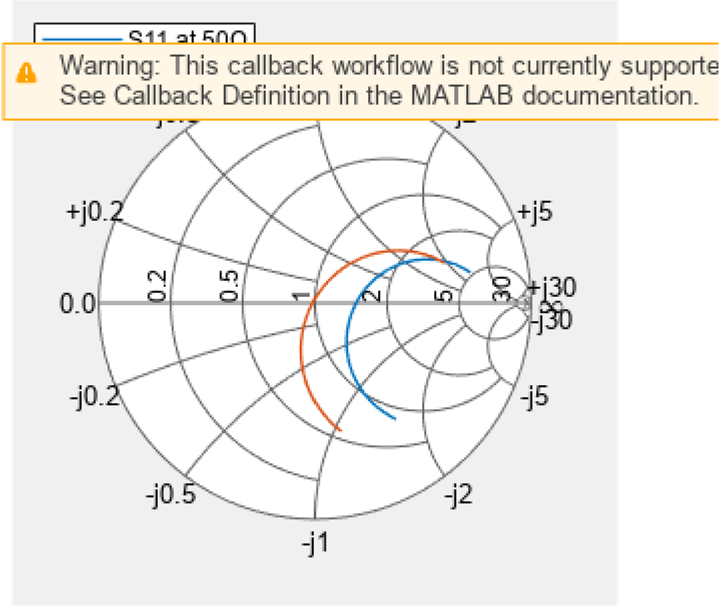
Plot S11 on a Smith plot for a reference impedance of 50 ohm.

```
d = dipole;
freq = linspace(60e6, 90e6, 200);
s_50 = sparameters(d, freq,50);
hg = smithplot(s_50,[1,1]);
hg.LegendLabels = {"S11 at 50#ohm"};
```



Find S11 for a new impedance of 75 ohm. Add new S11 to the existing Smith plot.

```
s_75 = sparameters(d, freq, 75);
gamma = rfparam(s_75,1,1);
add(hg, gamma);
hg.LegendLabels = {"S11 at 50#ohm","S11 at 75#ohm"};
```



Input Arguments

plot — Smith chart
function handle

Smith chart handle, specified as a function handle. If the handle of the Smith chart is not retained during creation, it is obtained by using the command `p = smithplot('gco')`.

Data Types: `double`

`data` — Input data

complex vector | complex matrix

Input data, specified as a complex vector or complex matrix.

For a matrix *D*, the columns of *D* are independent data sets. For *N*-by-*D* arrays, dimensions 2 and greater are independent data sets.

Data Types: `double`
Complex Number Support: Yes

`frequency` — Frequency data

real vector

Frequency data, specified as a real vector.

Data Types: `double`

Version History

Introduced in R2017b

See Also

`smithplot` | `replace`

replace

Remove current data and add new data to Smith chart

Syntax

```
replace(plot,data)
replace(plot,frequency,data)
```

Description

`replace(plot,data)` removes all current data from a Smith chart, `plot`, and adds new data to the Smith chart.

`replace(plot,frequency,data)` removes all current data and adds new data to the Smith chart based on multiple data sets containing frequencies corresponding to columns of the data matrix.

Examples

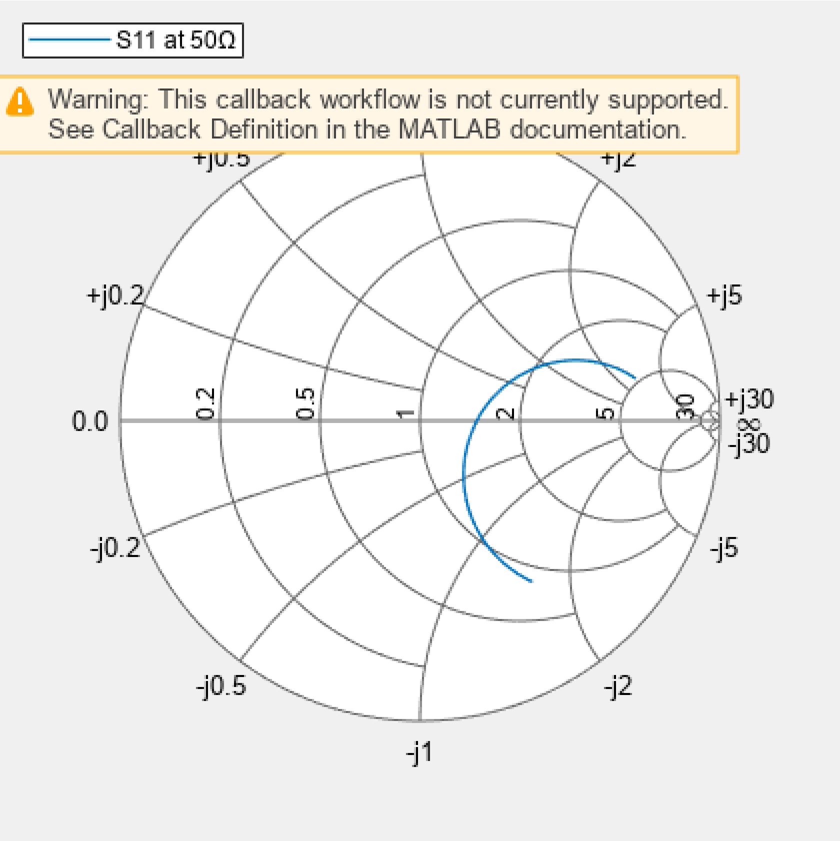
Replace S-Parameter Data on Existing Smith Chart

Plot the reflection coefficients of a dipole antenna.

Create a strip dipole antenna on the Y-Z plane. Calculate the complex S-parameters of the dipole antenna from 60 MHz to 90 MHz, with an interval of 150 kHz.

Plot S11 on a Smith chart for a reference impedance of 50 ohm.

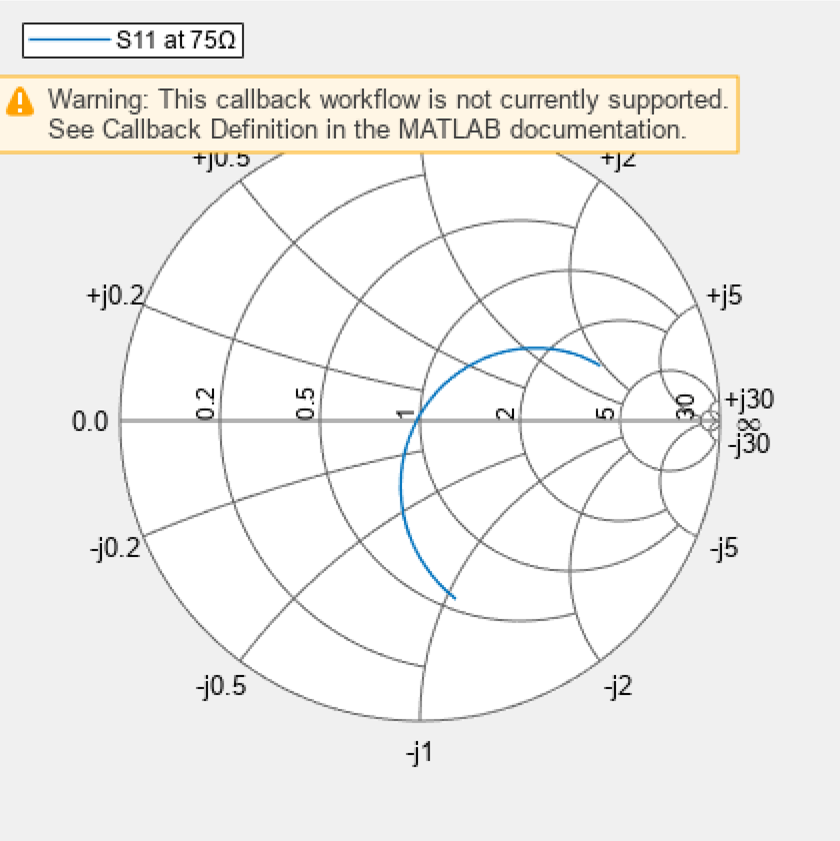
```
d = dipole;
freq = linspace(60e6,90e6,200);
s_50 = sparameters(d,freq,50);
hg = smithplot(s_50,[1,1]);
hg.LegendLabels = 'S11 at 50#ohm';
```



Warning: This callback workflow is not currently supported. See Callback Definition in the MATLAB documentation.

Find S11 for a new impedance of 75 ohm. Replace the old S11 by the new S11 on the existing Smith chart.

```
s_75 = sparameters(d,freq,75);
gamma = rfparam(s_75,1,1);
replace(hg,gamma);
hg.LegendLabels = 'S11 at 75#ohm';
```



Input Arguments

`plot` — Smith plot

plot handle

Smith chart handle, specified as a plot handle. If the handle of the Smith chart is not retained during creation, use `p = smithplot('gco')`.

`data` — Input data

complex vector | complex matrix

Input data, specified as a complex vector or complex matrix.

For a matrix *D*, the columns of *D* are independent datasets. For *N*-by-*D* arrays, dimensions 2 and greater are independent datasets.

Data Types: `double`
Complex Number Support: Yes

`frequency` — Frequency data

real vector

Frequency data, specified as a real vector.

Data Types: `double`

Version History

Introduced in R2017b

See Also

`add` | `smithplot`

smithplot

Plot measurement data on Smith chart

Syntax

```
smithplot(data)
smithplot(frequency,data)
smithplot(ax,___)
smithplot(hnet)
smithplot(hnet,i,j)
smithplot(hnet,[i_1,j_1;i_2,j_2;....,i_n,j_n])
s = smithplot(___)
s = smithplot('gco')
smithplot(___,Name,Value)
```

Description

`smithplot(data)` creates a Smith chart based on input data values.

Note The Smith chart is commonly used to display the relationship between a reflection coefficient, typically given as S11 or S22, and a normalized impedance.

`smithplot(frequency,data)` creates a Smith chart based on frequency and data values.

`smithplot(ax,___)` creates a Smith chart with a user defined axes handle, `ax`, instead of the current axes handle. Axes handles are not supported for network parameter objects. This parameter can be used with either of the two previous syntaxes.

`smithplot(hnet)` plots all the network parameter objects in `hnet`.

`smithplot(hnet,i,j)` plots the (*i*, *j*)th parameter of `hnet`. `hnet` is a network parameter object.

`smithplot(hnet,[i_1,j_1;i_2,j_2;....,i_n,j_n])` plots multiple parameters (i_1, j_1, i_2, j_2, ..., i_n, j_n) of `hnet`. `hnet` is a network parameter object.

`s = smithplot(___)` returns a Smith chart object handle so you can customize the plot and add measurements.

`s = smithplot('gco')` returns a Smith chart object handle of the current plot. This syntax is useful when the function handle, `p` was not returned or retained.

`smithplot(___,Name,Value)` creates a Smith chart with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding property value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

For list of properties, see SmithPlot Properties.

Note The property `'Parent'` might be used to control the location where Smith chart gets plotted. Target can be figure, UI figure, UI panel, etc.

Examples

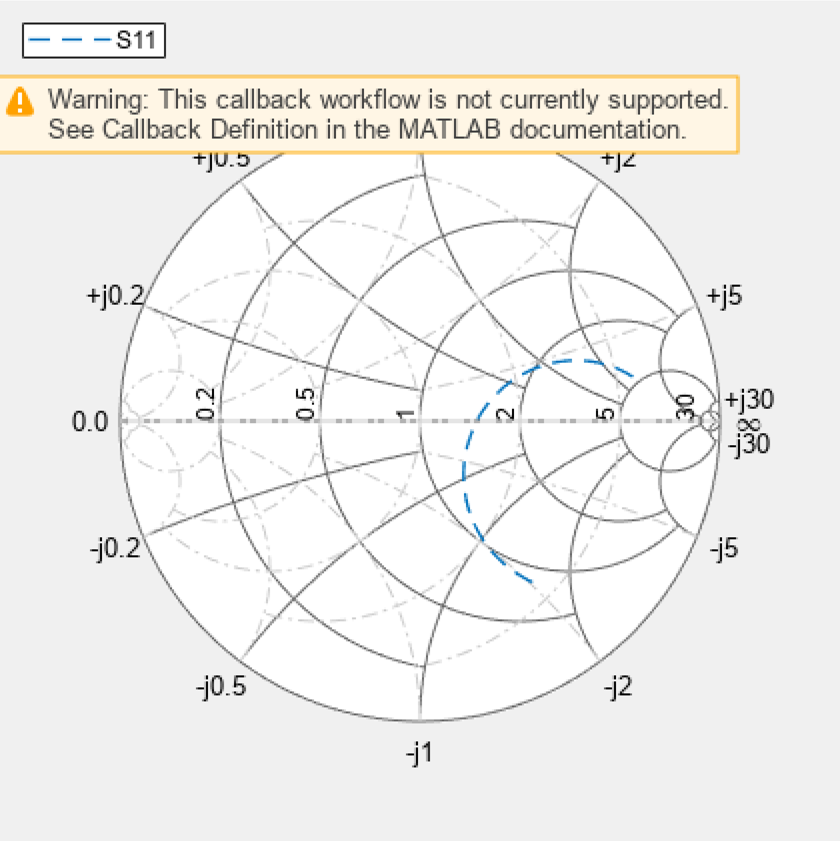
Plot the Reflection Coefficient of Dipole Antenna

Create a strip dipole antenna on the Y-Z plane. Calculate the complex S-parameters of the dipole antenna from 60 MHz to 90 MHz, with an interval of 150 kHz.

```
d = dipole;
freq = linspace(60e6,90e6,200);
s = sparameters(d,freq);
```

Plot the S11 on a Smith chart.

```
hg = smithplot(s,1,1,'GridType','ZY');
hg.LineStyle = '--'
```



Warning: This callback workflow is not currently supported.
See Callback Definition in the MATLAB documentation.

```
hg = 
  smithplot with properties:

         Data: [200x1 double]
    Frequency: [200x1 double]
  Show all properties, methods
```

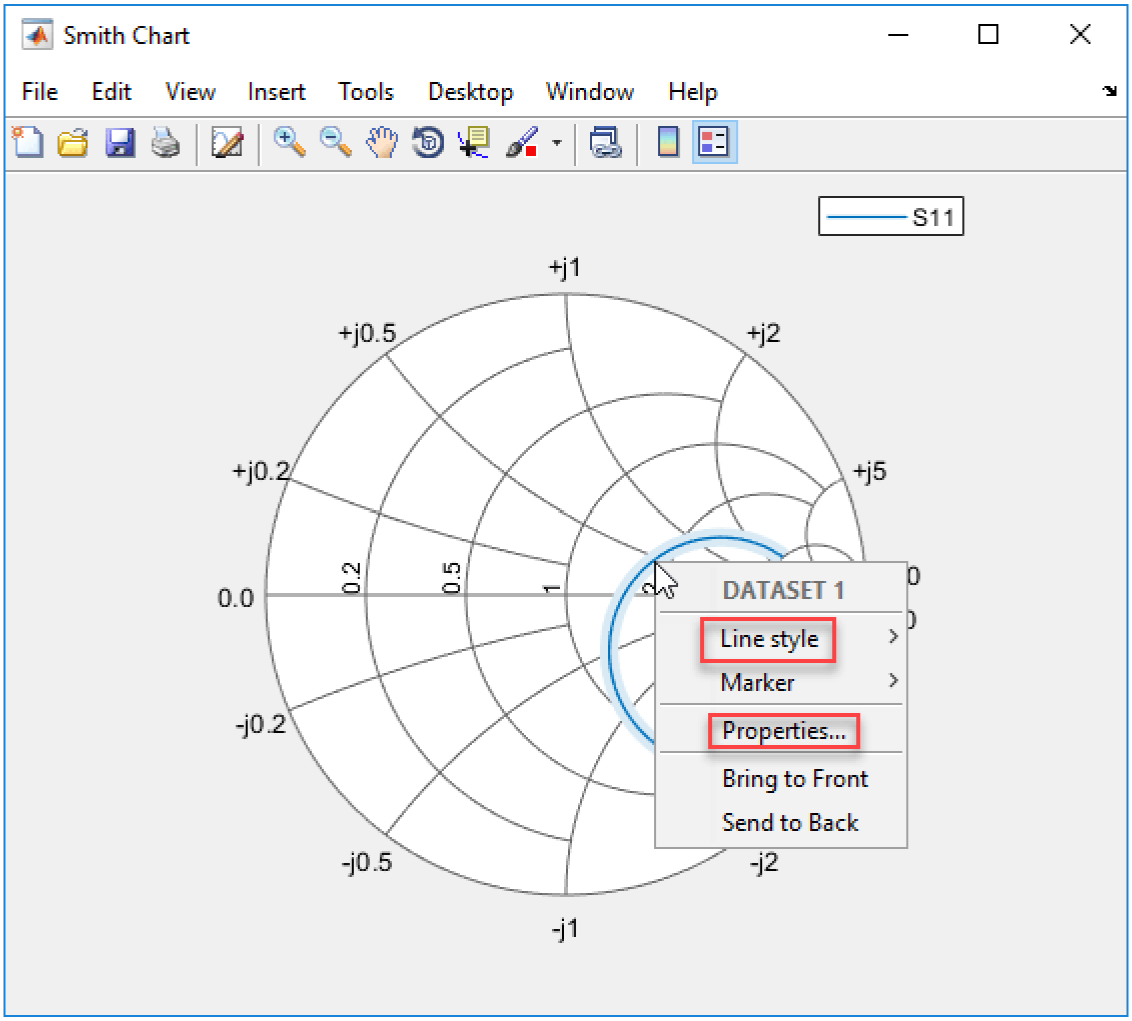
Smith Plot Interactive Menu

Use the Smith chart interactive menu for changing line and marker styles.

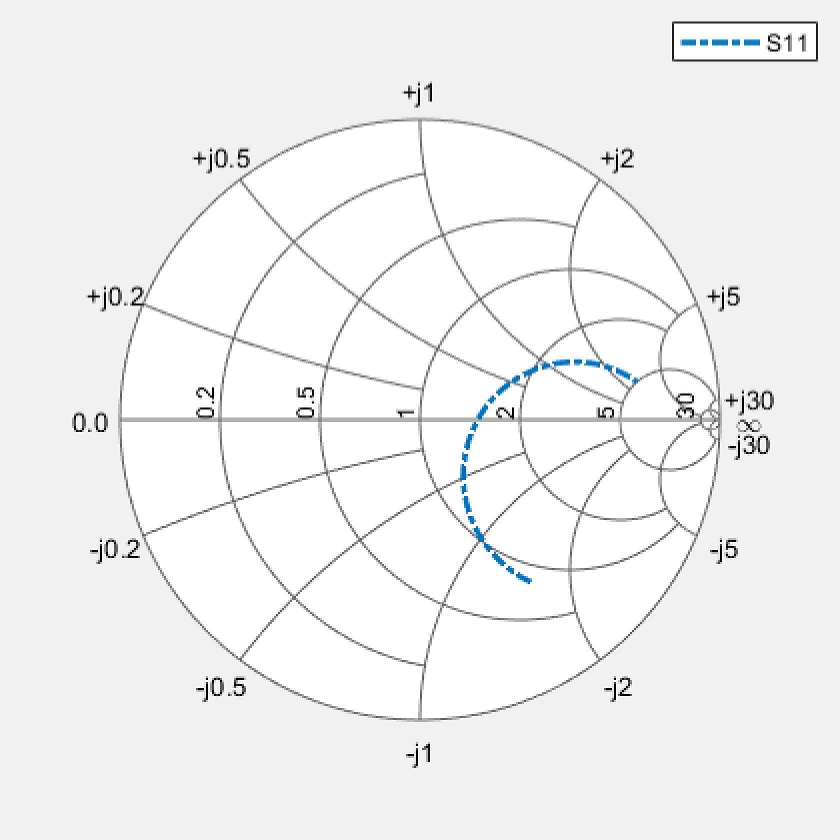
Plot the Smith chart of S-parameters of dipole `d`.

```
smithplot(s)
```

Right click on the S11 line to reveal interactive menu, `DATASET 1`. Set **Line style** to '-.' and **Properties...>Line Width** to '2' to change the line style and width of S11 line on the Smith chart.



You can see the changes you made on the Smith chart.



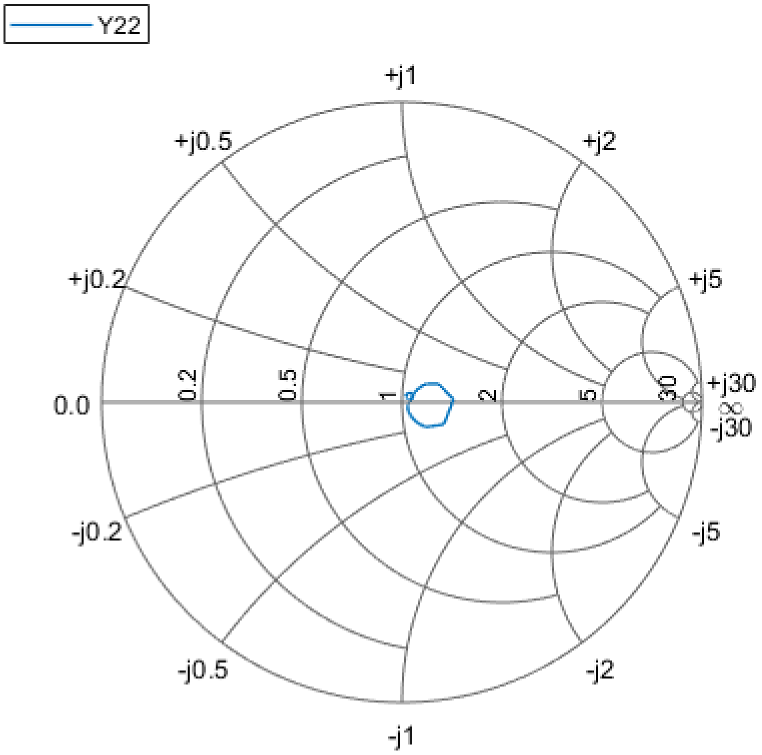
Add Z-Parameter Data to Existing Smith Plot

Read the S-parameter data.

```
amp = read(rfckt.amplifier,'default.s2p');
Sa = sparameters(amp);
```

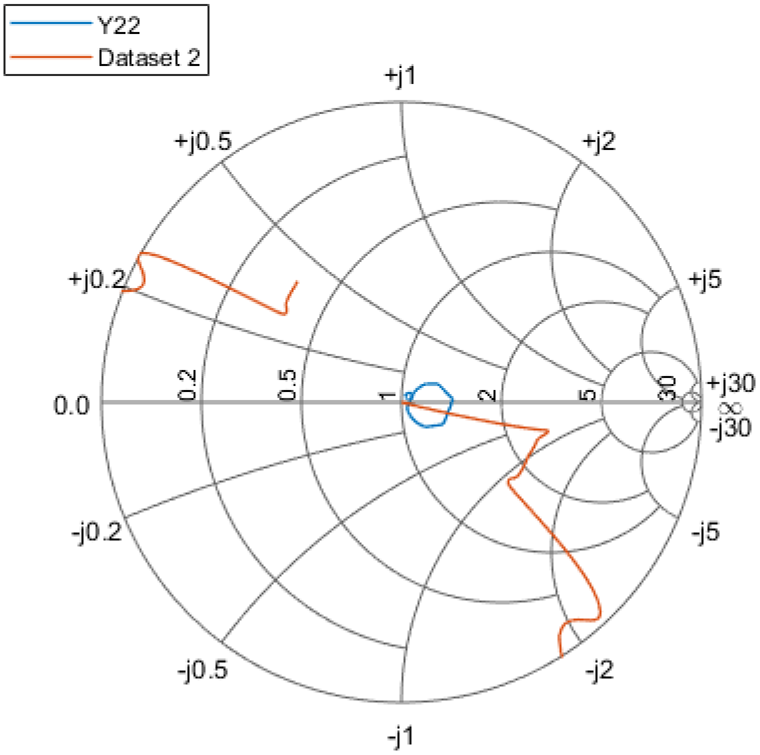
Convert the S-parameter data to Y-parameter and plot it on a Smith plot.

```
Ya = yparameters(Sa);
smithplot(Ya,2,2)
```



Add Z-parameter data to the same plot.

```
Za = zparameters(Sa);
Z12 = rfparam(Za,1,2);
Freq = Za.Frequencies;
s = smithplot('gco');
add(s, Freq, Z12);
```



Input Arguments

`data` — Input data
complex vector | complex matrix

Input data, specified as a complex vector or complex matrix.

For a matrix *D*, the columns of *D* are independent data sets. For *N*-by-*D* arrays, dimensions 2 and greater are independent data sets.

Data Types: `double`
Complex Number Support: Yes

`frequency` — Frequency data
real vector

Frequency data, specified as a real vector.

Data Types: `double`

`hnet` — Input objects
network parameter object (Antenna Toolbox)

Input objects, specified as a network parameter object.

Data Types: `double`

`rfbudgetobj` — RF budget object
`rfbudget` object

RF budget object, specified as a `rfbudget` object.

Output Arguments

`s` — Smith chart object handle
object

Smith chart object handle. You can use the handle to customize the plot and add measurements using MATLAB commands.

Tips

- To list all the property `Name,Value` pairs in `smithplot`, use `details(s)`. You can use the properties to extract any data from the Smith chart. For example, `s = smithplot(data,'GridType','Z')` displays the impedance data grid from the Smith chart.
- For a list of properties of `smithplot`, see SmithPlot Properties (RF Toolbox).
- You can use the `smithplot` interactive menu to change the line and marker styles.

Version History

Introduced in R2017b

See Also

`replace` | `add`

phaseShift

Calculate phase shift values for arrays or multi-feed PCB stack

Syntax

```
ps = phaseShift(array,frequency,angle)
ps = phaseShift(pcb,frequency,angle)
```

Description

`ps = phaseShift(array,frequency,angle)` calculates the phase shift values of an array operating at a specified frequency to scan the beam at the given angle. The velocity of light is assumed to be that in free space.

`ps = phaseShift(pcb,frequency,angle)` calculates the phase shift values of a multi-feed PCB stack at a specified frequency and angle.

Examples

Scan Main Beam of 3-by-3 Rectangular Array of Reflector-Backed Dipoles

Create a 3-by-3 rectangular array of reflector-backed dipoles at an operating frequency of 1.8 GHz, and scan the main beam at 30 degrees along the azimuth and 45 degrees along the elevation.

```
a = design(rectangularArray('Size',[3 3]),1.8e9,reflector);
ps = phaseShift(a,1.8e9,[30;45])
```

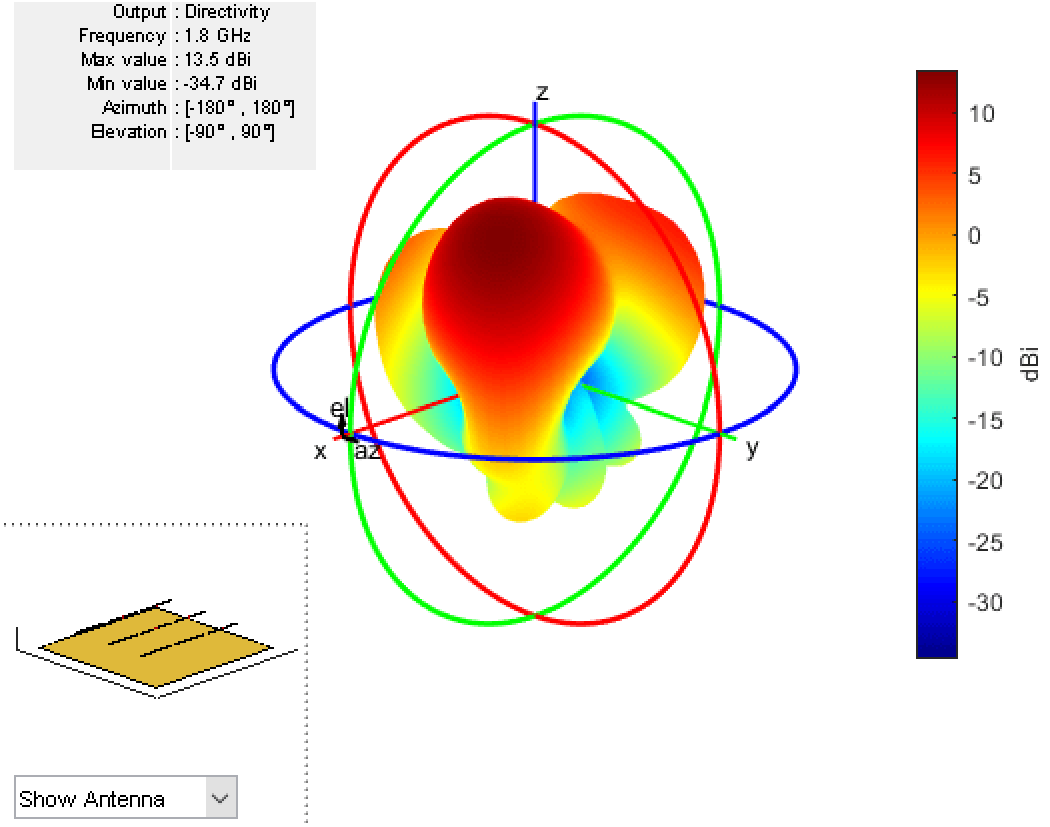
```
ps = 9×1

  350.5337
   54.1733
  117.8129
  240.3066
  303.9462
    7.5858
  130.0796
  193.7192
  257.3588
```

```
a.PhaseShift = ps
```

```
a = 
  rectangularArray with properties:

           Element: [1x1 reflector]
              Size: [3 3]
        RowSpacing: 0.0833
     ColumnSpacing: 0.0833
           Lattice: 'Rectangular'
    AmplitudeTaper: 1
        PhaseShift: [9x1 double]
              Tilt: 0
          TiltAxis: [1 0 0]
```

Calculate the radiation pattern of the array.

```
pattern(a,1.8e9)
```



```
   Output : Directivity
Frequency : 1.8 GHz
Max value : 13.5 dBi
Min value : -34.7 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

Input Arguments

array — Antenna array
array object

Antenna array from the Antenna Toolbox array library, specified as an array object.

Example: `r = rectangularArray; phaseShift (r,70e6,[60;40])`. Calculates the phase shift of the rectangular array.

pcb — Multi-feed PCB stack
`pcbStack` object

Multi-feed PCB stack, specified as a `pcbStack` object.

Example: `fco = invertedFcoplanar; pcb = pcbStack(fco); phaseShift (pcb,70e6,[60;40])` Calculates the phase shift of the coplanar inverted F antenna PCB.

`frequency` — Frequency value to calculate phase shift
scalar

Frequency value used to calculate the phase shift, specified as a scalar in Hz.

Example: `70e6`

Data Types: `double`

`angle` — Azimuth and elevation angle pair
2-element vector

Azimuth and elevation angle pair to scan the array toward, specified as a 2-element vector in degrees.

Example: `[35;40]`

Data Types: `double`

Output Arguments

`ps` — Phase shift values
1-by-*N* vector

Phase shift values, returned as a 1-by-*N* vector in degrees. Phase shift value calculation does not consider mutual coupling.

Version History

Introduced in R2018b

See Also

`pattern` | `patternMultiply` | `feedCurrent`

patternFromSlices

Reconstruct approximate 3-D radiation pattern from two orthogonal slices

Syntax

```
patternFromSlices(vertislice,theta,horizslice,phi)
patternFromSlices(vertislice,theta,horizslice)
patternFromSlices(vertislice,theta)
[pat3D,thetaout,phiout] = patternFromSlices( ___ )
[ ___ ] = patternFromSlices( ___ ,Name,Value)
```

Description

`patternFromSlices(vertislice,theta,horizslice,phi)` plots the approximate 3-D pattern reconstructed from the input data containing the 2-D pattern along the vertical and horizontal plane as well as the polar and azimuthal angles in the spherical coordinates.

`patternFromSlices(vertislice,theta,horizslice)` plots the approximate 3-D pattern with the horizontal slice provided as a real-valued scalar. The syntax assumes that the antenna is omnidirectional with symmetry about the Z-axis.

`patternFromSlices(vertislice,theta)` plots the approximate 3-D pattern reconstructed from only vertical pattern data, assuming of azimuthal omni directionality and that horizontal pattern data is equal to maximum value of vertical pattern data.

`[pat3D,thetaout,phiout] = patternFromSlices( ___ )` returns the reconstructed pattern as a matrix with the vectors of phi and theta.

`[ ___ ] = patternFromSlices( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to any of the input argument combinations in previous syntaxes. For example, you can specify the customization and tuning options to the pattern reconstruction method.

Examples

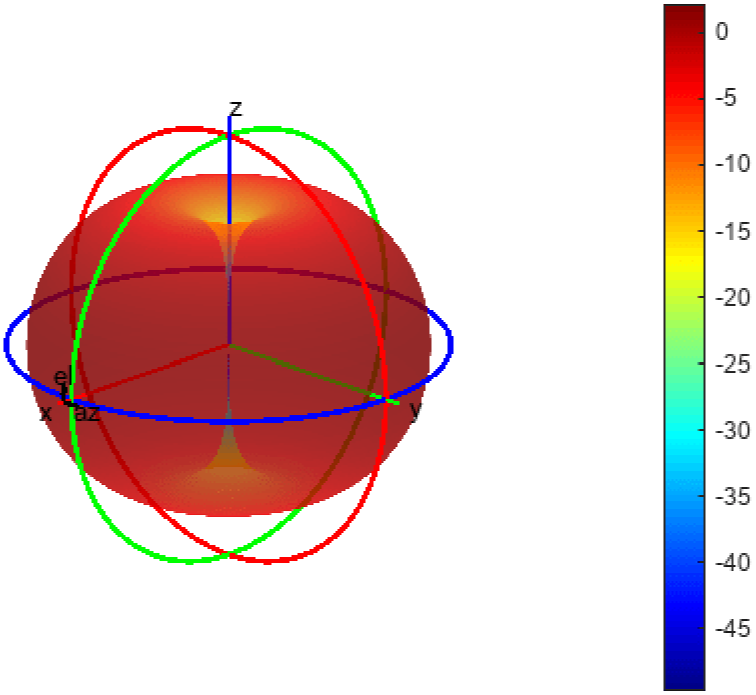
Reconstruct Pattern of Dipole Antenna from 2-D Slices

Load the `MAT` file containing the data of the dipole pattern.

```
load dipoleAntennaSlices.mat
```

Reconstruct the pattern from the data provided using the `CrossWeighted` algorithm and pattern plot options.

```
p = PatternPlotOptions('Transparency',0.6);
patternFromSlices(vertSlice,theta,horizSlice,phi,'Method','CrossWeighted','PatternOptions',p)
```



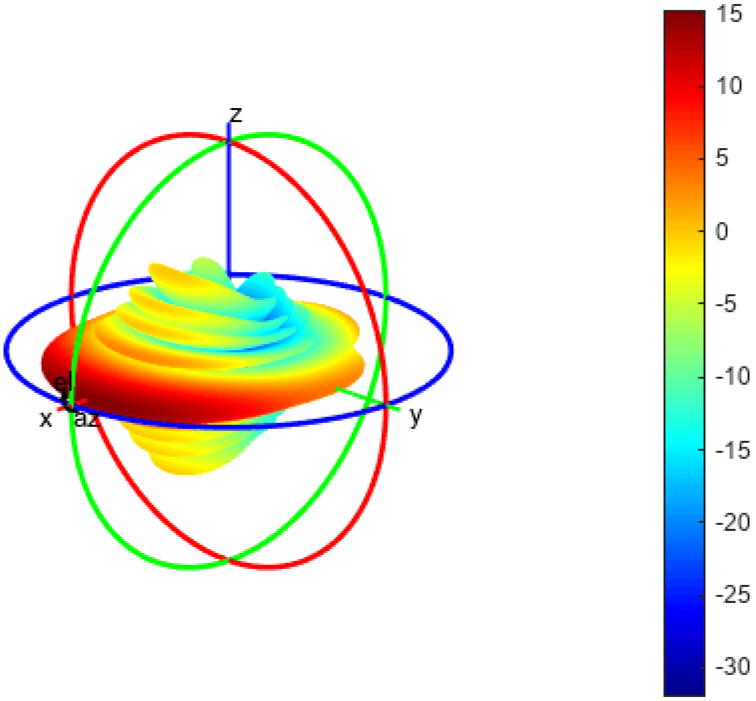
Reconstruct Pattern of Sector Antenna from 2-D Slices

Load the MAT file containing the data of the sector antenna pattern.

```
load sectorAntennaSlices.mat
```

Reconstruct the pattern from the data provided using the Summing algorithm.

```
 patternFromSlices(vertSlice,theta,horizSlice,phi,'Method','Summing')
```



```
[pat3D,thetaout,phiout] = patternFromSlices(vertSlice,theta,horizSlice,phi,'Method','Summing');
pat3D = pat3D(1:5)
```

```
pat3D = 1×5

  -23.2025  -23.2071  -23.2224  -23.2485  -23.2854
```

```
thetaout = thetaout(1:5)
```

```
thetaout = 1×5

   180   179   178   177   176
```

```
phiout = phiout(1:5)
```

```
phiout = 1×5

  -180  -179  -178  -177  -176
```

Input Arguments

Required Input Arguments

`vertislice` — 2-D pattern slice data along vertical or elevation plane

real-valued vector

2-D pattern slice data along the vertical or the elevation plane, specified as a real-valued vector with each element unit in dBi. This parameter need not be normalized. `numel(vertislice)` must be equal to `numel(theta)`.

Data Types: `double`

`theta` — Polar or inclination angles in spherical coordinates

real-valued vector | `theta` < 180

Polar or inclination angles in spherical coordinates, specified as a real-valued vector with each element unit in degrees.

Note

$$\theta = 90 - el$$

el is the elevation angle.

Data Types: `double`

Optional Input Arguments

`horizslice` — 2-D pattern slice data along horizontal or azimuthal plane

real-valued scalar | real-valued vector

2-D pattern slice data along the horizontal or the azimuthal plane, specified as a real-valued scalar in dBi or a real-valued vector with each element unit in dBi.

- If the value is a vector, then `numel(horizslice)` must be equal to `numel(phi)`.
- If the value is a scalar, then the antenna is omnidirectional if the scalar value is used for all angles in the azimuthal plane.
- If no value is provided, then the antenna is omnidirectional and the default value (for the entire azimuthal slice) is set equal to the maximum directivity or gain of the elevation slice.

Data Types: `double`

`phi` — Azimuthal angles in spherical coordinates

real-valued vector

Azimuthal angles in the spherical coordinates, specified as a real-valued vector with each element unit in degrees. If this argument is not provided:

- The antenna is assumed omnidirectional with symmetry about the Z-axis or azimuthal symmetry.
- The default values used are: `phi = 0:5:360`.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` pair 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: `'Method'`, `'Summing'`

`Method` — Approximate interpolation algorithm to perform reconstruction

`'Summing'` (default) | `'CrossWeighted'`

Approximate interpolation algorithm to perform reconstruction, specified as the comma-separated pair consisting of `'Method'` and `'Summing'` or `'CrossWeighted'`.

Example: `'Method'`, `'CrossWeighted'`

Data Types: `char`

`CrossWeightedNormalization` — Normalization parameter for cross-weighted summing method

2 | real-valued positive scalar

Normalization parameter for cross-weighted summing method, specified as a comma-separated pair consisting of `'CrossWeightedNormalization'` and a real-valued positive scalar. As this parameter increases, the pattern reconstruction becomes a pessimistic approximation of the estimated directivity or gain. As this parameter decreases, the pattern reconstruction becomes an optimistic approximation of the estimated directivity or gain.

Example: `'CrossWeightedNormalization'`,2

Data Types: `double`

`PatternOptions` — Parameter to change pattern plot properties

`PatternPlotOptions` object (default) | scalar

Parameter to change pattern plot properties, specified as the comma-separated pair consisting of `'PatternOptions'` and a `PatternPlotOptions` output. The properties that you can vary are:

- `Transparency`
- `MagnitudeScale`

Other properties used in the `'PatternOptions'` for the inset figure are ignored in `patternCustom`.

Data Types: `double`

Output Arguments

`pat3D` — Matrix of reconstructed 3-D pattern

N-by-*M* real-valued array

Matrix of reconstructed 3-D pattern, returned as an *N*-by-*M* real-valued array. The number of rows in the matrix corresponds to the number of phi elements in dBi. The number of columns in the matrix corresponds to the number of theta elements in dBi.

`thetaout` — Polar inclination angle

M-element real-valued vector

Polar inclination angle, returned as an *M*-element real-valued vector in degrees. The returned value is for the subset of input data for the chosen reconstructed method.

phiout — Azimuthal angle
N-element real-valued vector

Azimuthal angle, returned as an *N*-element real-valued vector in degrees. The returned value is for the subset of input data for the chosen reconstructed method.

Limitations

- The summing method does not always reliably approximate the 3-D pattern in the backplane. It works effectively for the azimuthally omnidirectional patterns, as the front plane and the backplane are symmetrical about the z-axis.
- The cross-weighted method can be used to approximate the 3-D pattern in both the front plane and backplane, but the accuracy or robustness is typically best for the main radiation lobe in the front plane.
- Both the summing and the cross-weighted methods do not utilize vertical pattern slice data from the backplane (that is `theta` ≥ 180°). If you provide backplane vertical slice and theta data, the `patternFromSlices` function discards it. However, the `patternFromSlices` function uses all the horizontal pattern and front plane vertical pattern slice data.

More About

Summing

The summing approximation or interpolation algorithm performs:

$G(\phi, \theta) = G_H(\phi) + G_V(\theta)$

where, $G_H(\phi)$ and $G_V(\theta)$ are the normalized 2-D pattern cut data in dBi.

Cross-Weighted

The cross-weighted approximation or interpolation algorithm performs:

$$G_H(\phi, \theta) = \frac{G_H(\phi) \bullet w_1 + G_V(\theta) \bullet w_2}{\sqrt[k]{w_1^k + w_2^k}}$$

where,

- $\begin{cases} w_1(\phi, \theta) = \mathrm{vert}(\theta) \bullet [1 - \mathrm{hor}(\phi)] \\ w_2(\phi, \theta) = \mathrm{hor}(\phi) \bullet [1 - \mathrm{vert}(\theta)] \end{cases}$
- $G_H(\phi)$ and $G_V(\theta)$ are normalized 2-D pattern cut data in dBi.
- `hor(ϕ)` and `vert(θ)` are normalized in linear units.
- *k* is the normalization parameter.

Version History

Introduced in R2019a

References

[1] Makarov, Sergey N. *Antenna and Em Modeling in MATLAB*. Chapter3, Sec 3.4 3.8. Wiley Inter-Science.

[2] Balanis, C.A. *Antenna Theory, Analysis and Design*, Chapter 2, sec 2.3-2.6, Wiley.

[3] T. G. Vasiliadis, A. G. Dimitriou and G. D. Sergiadis, "A novel technique for the approximation of 3-D antenna radiation patterns," in *IEEE Transactions on Antennas and Propagation*, July 2005, vol. 53, no. 7: pp. 2212-2219.

[4] N. R. Leonor, R. F. S. Caldeirinha, M. G. Sánchez and T. R. Fernandes, "A Three-Dimensional Directive Antenna Pattern Interpolation Method," in *IEEE Antennas and Wireless Propagation Letters*, 2016, vol. 15, pp. 881-884.

See Also

`pattern` | `patternAzimuth` | `patternElevation` | `patternCustom`

Topics

"3D Reconstruction of Radiation Pattern From 2D Orthogonal Slices"

PatternPlotOptions

Creates option list to customize 3-D radiation pattern for pattern overlay option

Syntax

```
patternplot = PatternPlotOptions
patternplot = PatternPlotOptions(Name,Value)
```

Description

`patternplot = PatternPlotOptions` creates an option list for a 3-D radiation pattern for pattern overlay option.

`patternplot = PatternPlotOptions(Name,Value)` returns a pattern plot option list based on the specified properties. Properties not specified retain their default values.

Examples

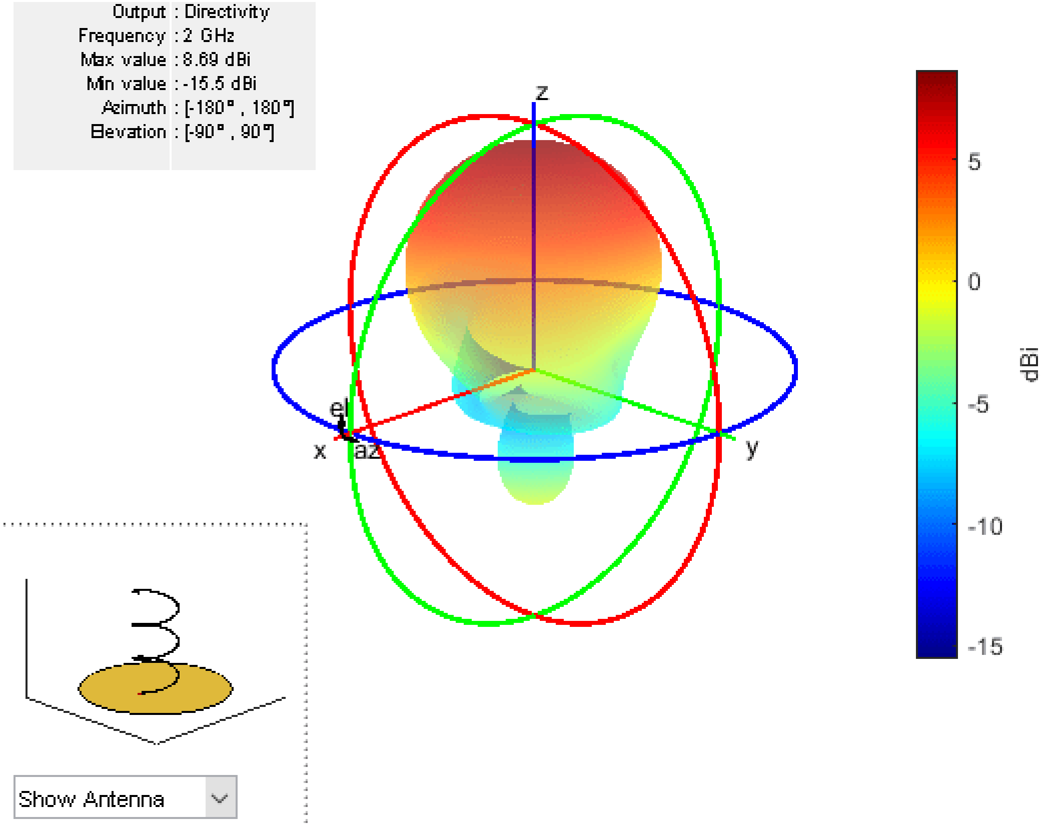
Radiation Pattern of Helix Antenna

Plot the radiation pattern of a helix antenna with transparency specified as 0.5.

```
p = PatternPlotOptions
```

```
p = 
  PatternPlotOptions with properties:

      Transparency: 1
         SizeRatio: 0.9000
    MagnitudeScale: []
     AntennaOffset: [0 0 0]
```

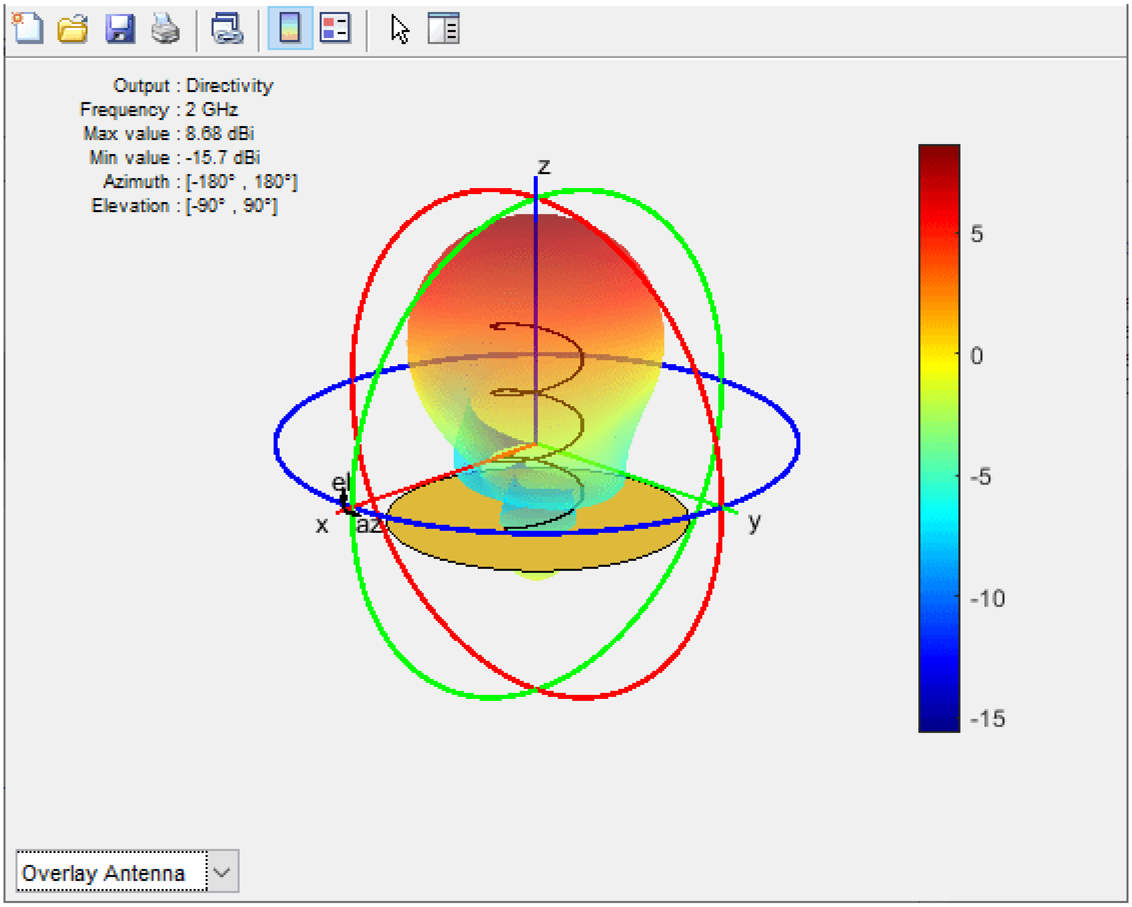
```
p.Transparency = 0.5;
ant = helix;
pattern(ant,2e9,'patternOptions',p)
```



```
   Output : Directivity
Frequency : 2 GHz
Max value : 8.69 dBi
Min value : -15.5 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

To understand the effect of Transparency, chose `Overlay Antenna` in the radiation pattern plot.

This option overlays the helix antenna on the radiation pattern.



Output : Directivity
Frequency : 2 GHz
Max value : 8.68 dBi
Min value : -15.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` pair 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: `'Transparency'`, `0.1`

`Transparency` — Transparency of 3-D radiation pattern

`0.8000` (default) | scalar

Transparency of the 3-D radiation pattern, specified as the comma-separated pair consisting of `'Transparency'` and a scalar value between `0` and `1`.

Example: `'Transparency'`, `0.5`

Example: `patternplot.Transparency = 0.5`

Data Types: `double`

`SizeRatio` — Relative size of antenna to radiation pattern

`0.9000` (default) | positive scalar

Relative size of the antenna to the radiation pattern, specified as the comma-separated pair of `'SizeRatio'` and a positive scalar.

Example: `'SizeRatio'`, `1`

Example: `patternplot.SizeRatio = 1`

Data Types: `double`

`AntennaOffset` — Position of antenna with pattern center as origin

`[0 0 0]` (default) | three-element vector

Position of the antenna with the pattern center as the origin, specified as the comma-separated pair consisting of `'AntennaOffset'` and a three-element vector of [x, y, z] coordinates.

Example: `'AntennaOffset'`, `[1,0,0]`

Example: `patternplot.AntennaOffset = [1,0,0]`

Data Types: `double`

`MagnitudeScale` — Scale of radiation pattern

two-element vector

Scale of the radiation pattern, specified as the comma-separated pair consisting of `'MagnitudeScale'` and a two-element vector of minimum magnitude and maximum magnitude. If this property is empty, the radiation pattern plot is of the full range magnitude.

Example: `'MagnitudeScale'`, `[0,1]`

Example: `patternplot.MagnitudeScale = [0,1]`

Data Types: `double`

Version History

Introduced in R2019a

See Also

`pattern` | `patternAzimuth` | `patternElevation` | `patternCustom`

stlwrite

Write mesh to STL file

Syntax

```
stlwrite(objname,filename)
```

Description

`stlwrite(objname,filename)` writes the triangles in the mesh for an antenna or array object to an STL file in text format using the specified file name.

Examples

Platform from STL of DipoleHelix Antenna

Create a `DipoleHelix` antenna object at 2 GHz and compute the impedance.

```
w = design(dipoleHelix,2e9);
Z = impedance(w,2e9);
```

Create an STL file for `DipoleHelix` antenna object .

```
stlwrite(w,'dipoleHelix_2GHz.stl')
```

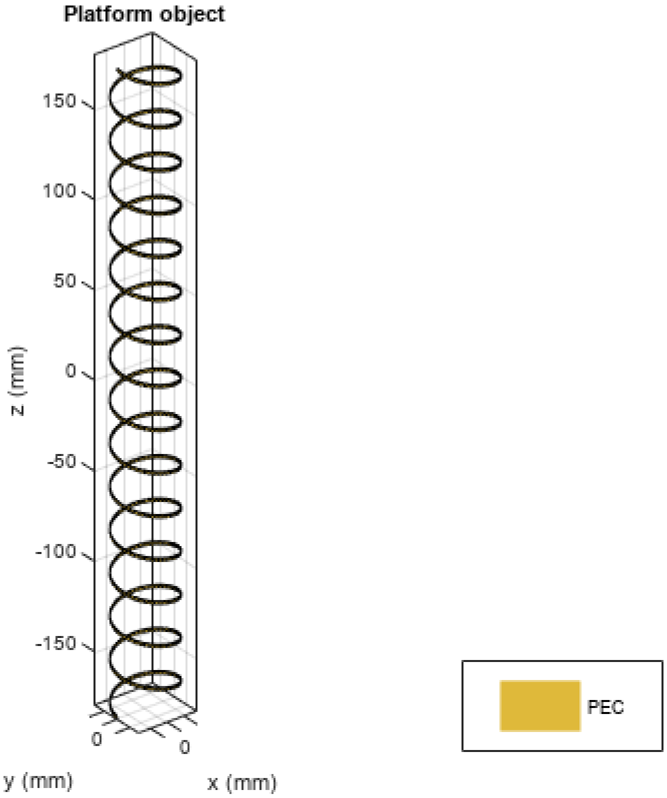
You will see the `dipoleHelix_2GHz.stl` file in your current folder.

Load `dipoleHelix_2GHz.stl` and visualize the platform.

```
plat = platform('FileName','dipoleHelix_2GHz.stl','Units','m')
```

```
plat = 
  platform with properties:

         FileName: 'dipoleHelix_2GHz.stl'
            Units: 'm'
    UseFileAsMesh: 0
             Tilt: 0
         TiltAxis: [1 0 0]
```

```
show(plat)
```



Input Arguments

`objname` — Antenna or array object
antenna or array

Antenna or array object, specified as an antenna or array.

`filename` — Name of STL file
character vector

Name of STL file, specified as a character vector in STL format.

Version History

Introduced in R2019a

See Also

`meshconfig` | `show` | `platform`

rcs

Calculate and plot radar cross section (RCS) of platform, antenna, or array

Syntax

```
rcs(object,frequency)
rcs(object,frequency,azimuth,elevation)
rcs( ___ ,Name,Value)

[rcsval,azimuth,elevation] = rcs(object,frequency)
[rcsval,azimuth,elevation] = rcs( ___ ,Name,Value)
```

Description

`rcs(object,frequency)` plots the monostatic RCS of the platform, antenna, or array object over a specified frequency. To learn more about RCS, see “What Is RCS?” on page 4-343.

`rcs(object,frequency,azimuth,elevation)` plots the monostatic RCS for the specified azimuth and elevation angles.

`rcs( ___ ,Name,Value)` plots the RCS with additional properties specified using one or more Name, Value pair arguments. This parameter can be used with any of the input arguments from the previous syntaxes.

`[rcsval,azimuth,elevation] = rcs(object,frequency)` returns the RCS value of a platform, antenna, or array object at the specified frequency. `azimuth` and `elevation` are vectors over which the RCS value is calculated.

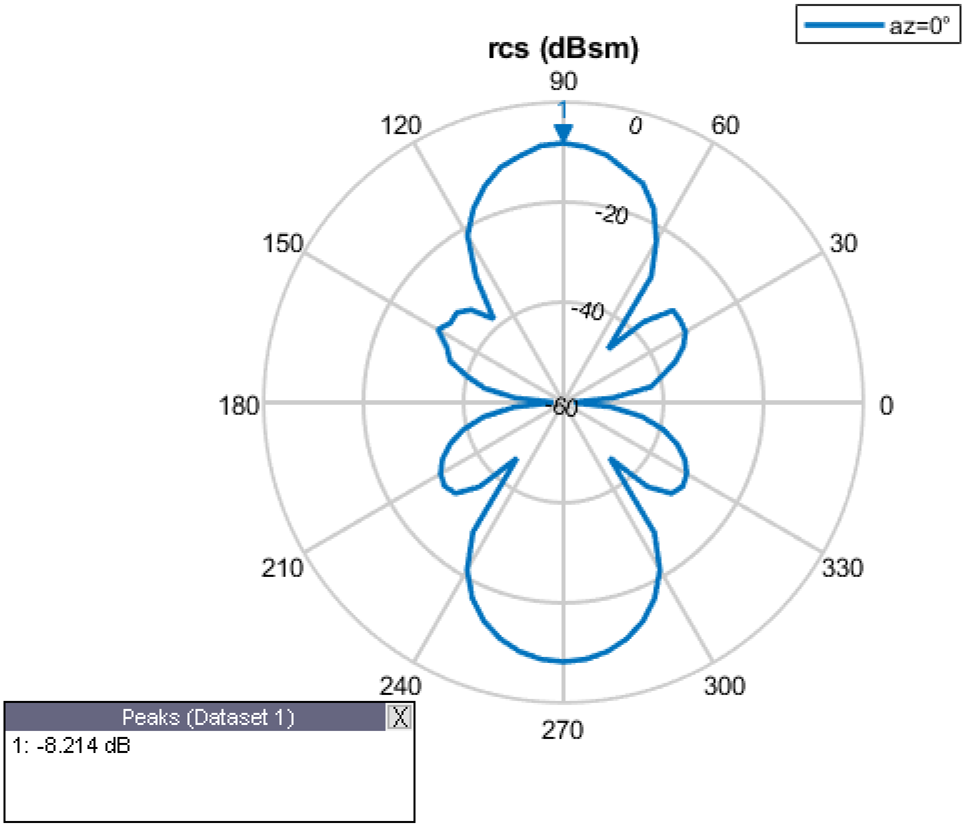
`[rcsval,azimuth,elevation] = rcs( ___ ,Name,Value)` returns the RCS value with additional properties specified using one or more Name, Value pair arguments. This parameter can be used with any of the input arguments from the previous syntaxes.

Examples

RCS of Helix

Create a default helix antenna and plot the RCS at 2 GHz.

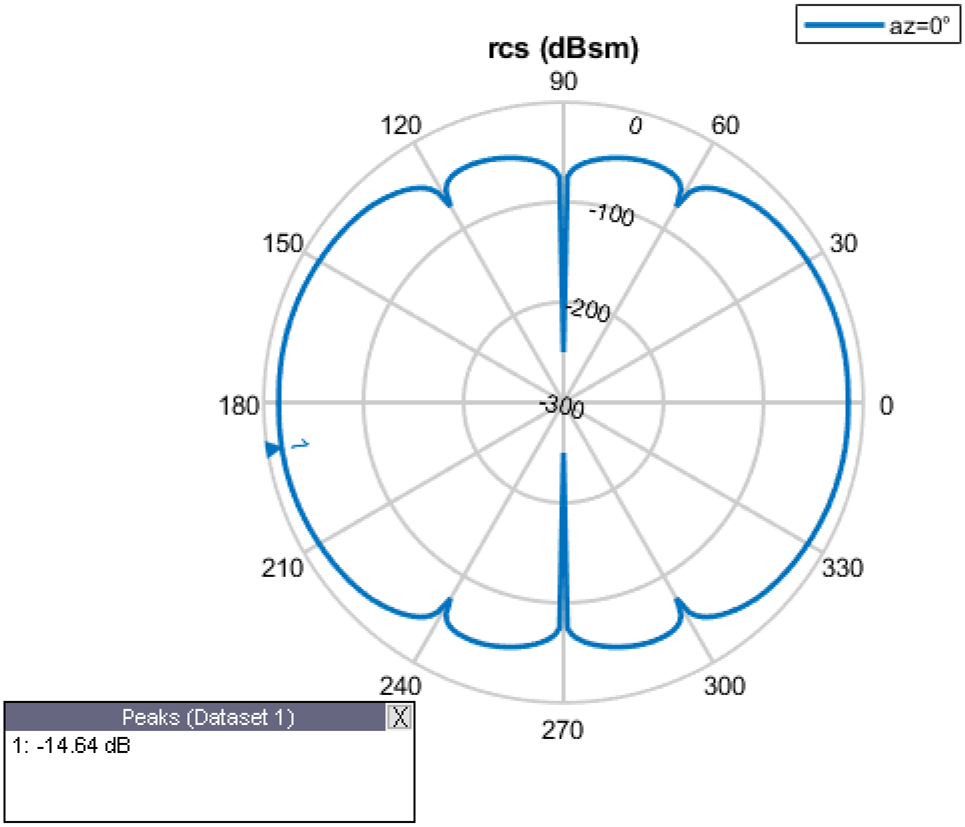
```
ant = helix;
rcs(ant,2e9)
```



RCS of Linear Array

Create a default linear array and plot the RCS at 75 MHz in the elevation pane.

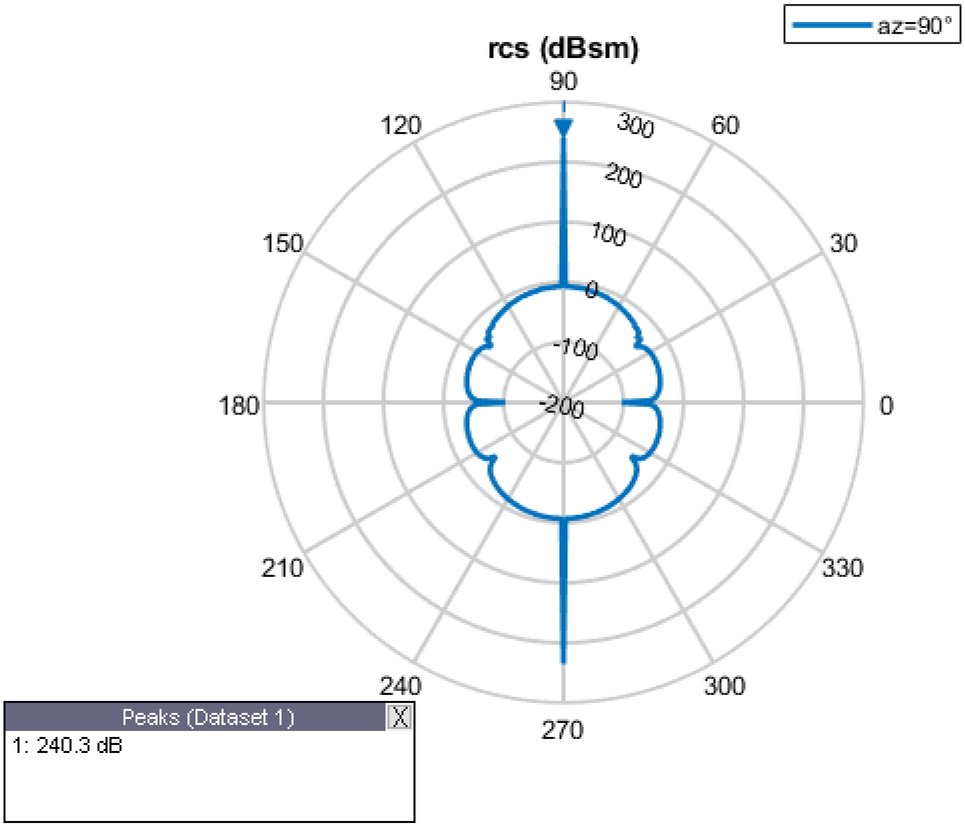
```
array = linearArray;
rcs(array,75e6,0,0:1:360)
```



RCS of Reflector-Backed Dipole

Create a reflector-backed dipole and plot the RCS at 1 GHz in the elevation plane at 90 degree azimuth.

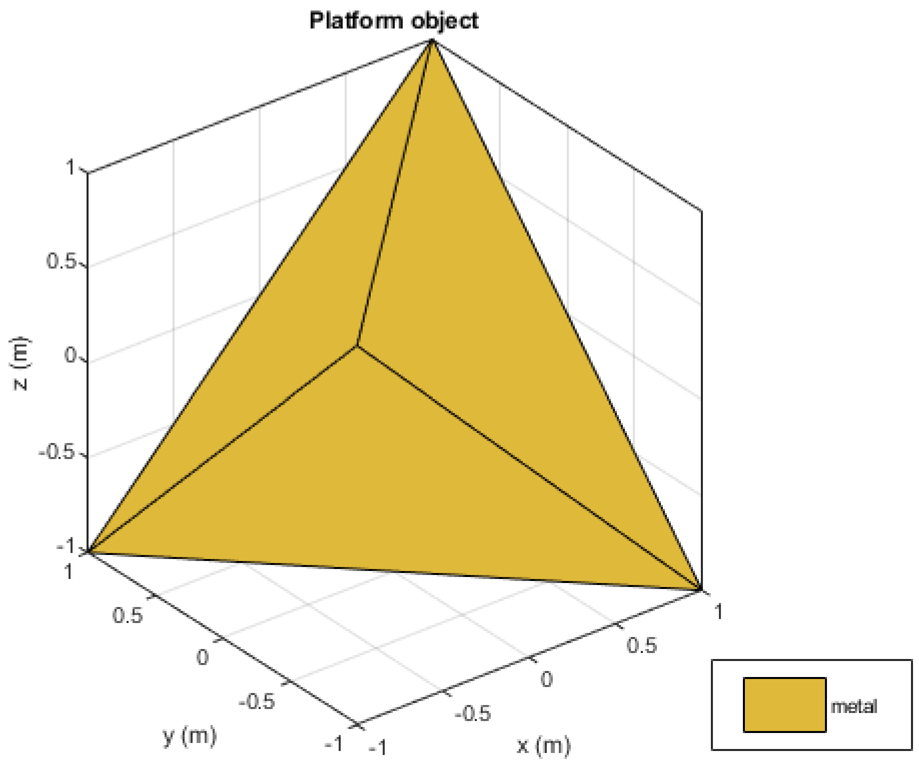
```
ant = reflector;
rcs(ant,1e9,90,0:1:360)
```



RCS of Tetrahedron Platform

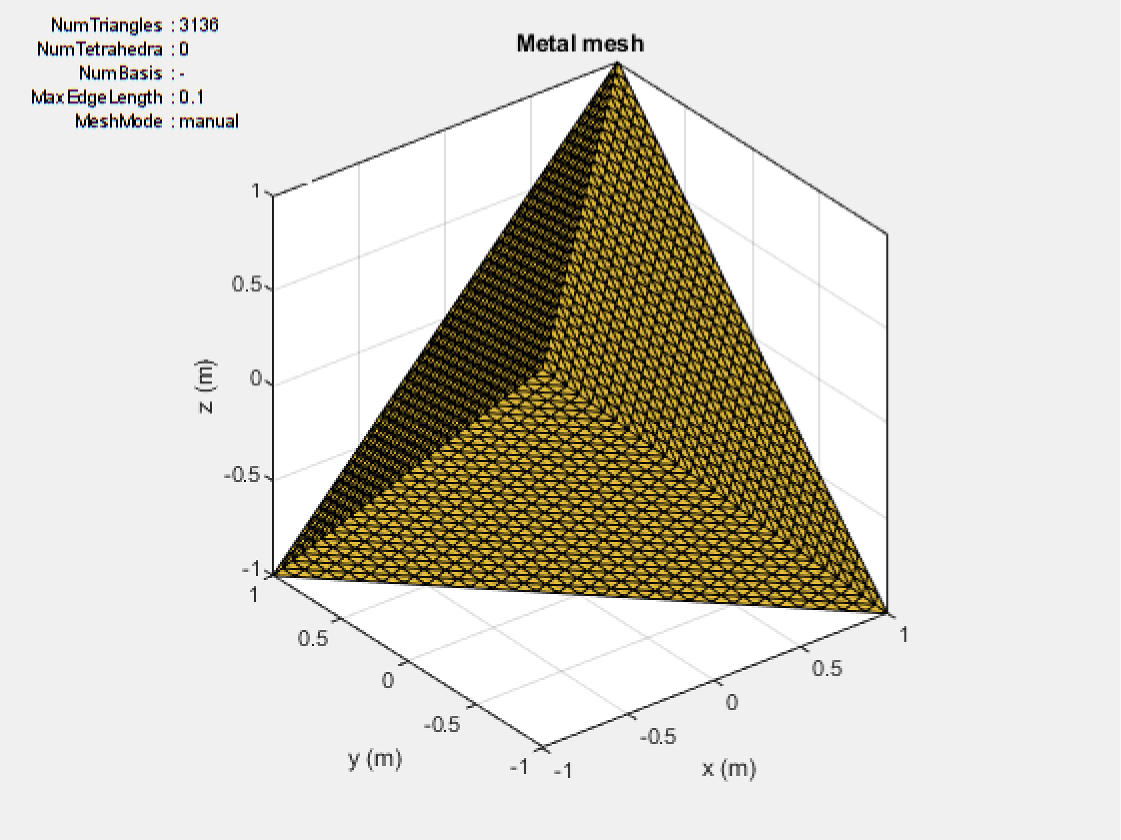
Create a tetrahedron platform from an STL file.

```
p = platform;
p.FileName = 'tetrahedra.stl';
p.Units = 'm';
figure
show(p)
```



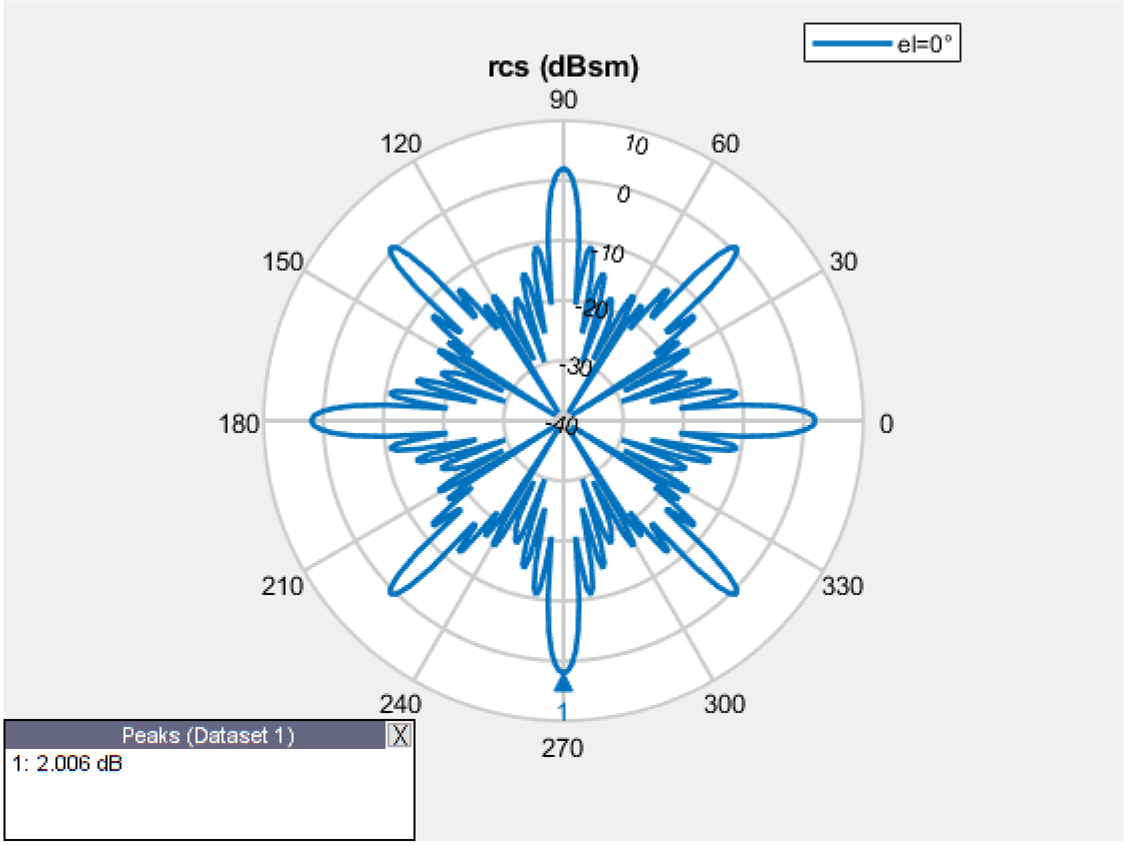
Mesh the platform with edge length of 0.1

```
figure
mesh(p,'MaxEdgeLength',0.1)
```



Sweep over the elevation with a vertically polarized E-field. Plot the RCS at 700 MHz in the azimuth plane.

```
az = 0:1:360;
el = 0;
figure
rcs(p,700e6,az,el)
```



Peaks (Dataset 1)

1: 2.006 dB

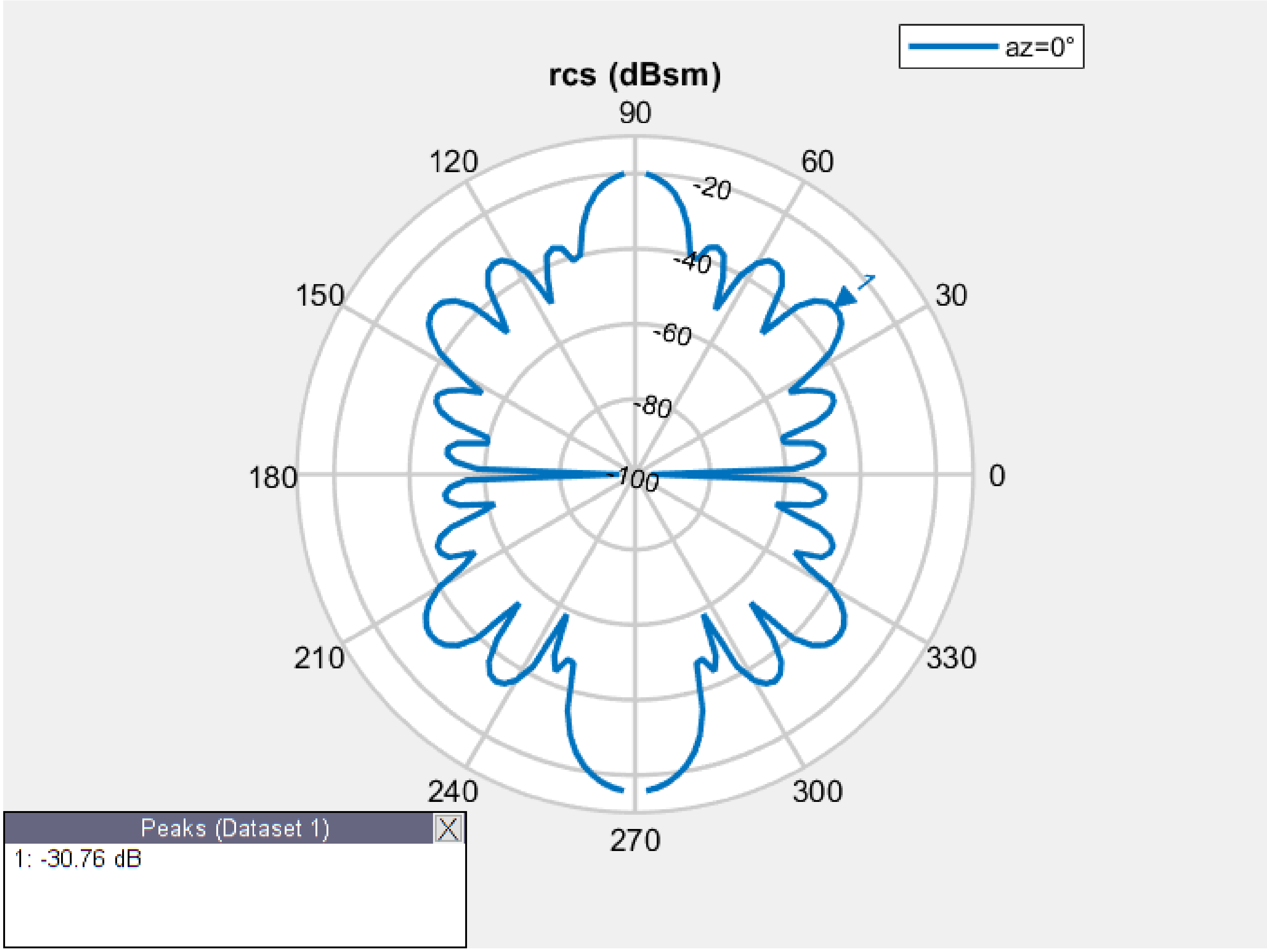
RCS of Corner Reflector

Create a corner reflector-backed antenna.

```
f = 2e9;
c = design(reflectorCorner,750e6);
```

Plot the RCS in the elevation plane.

```
figure
rcs(c,f,0,0:2:360)
```

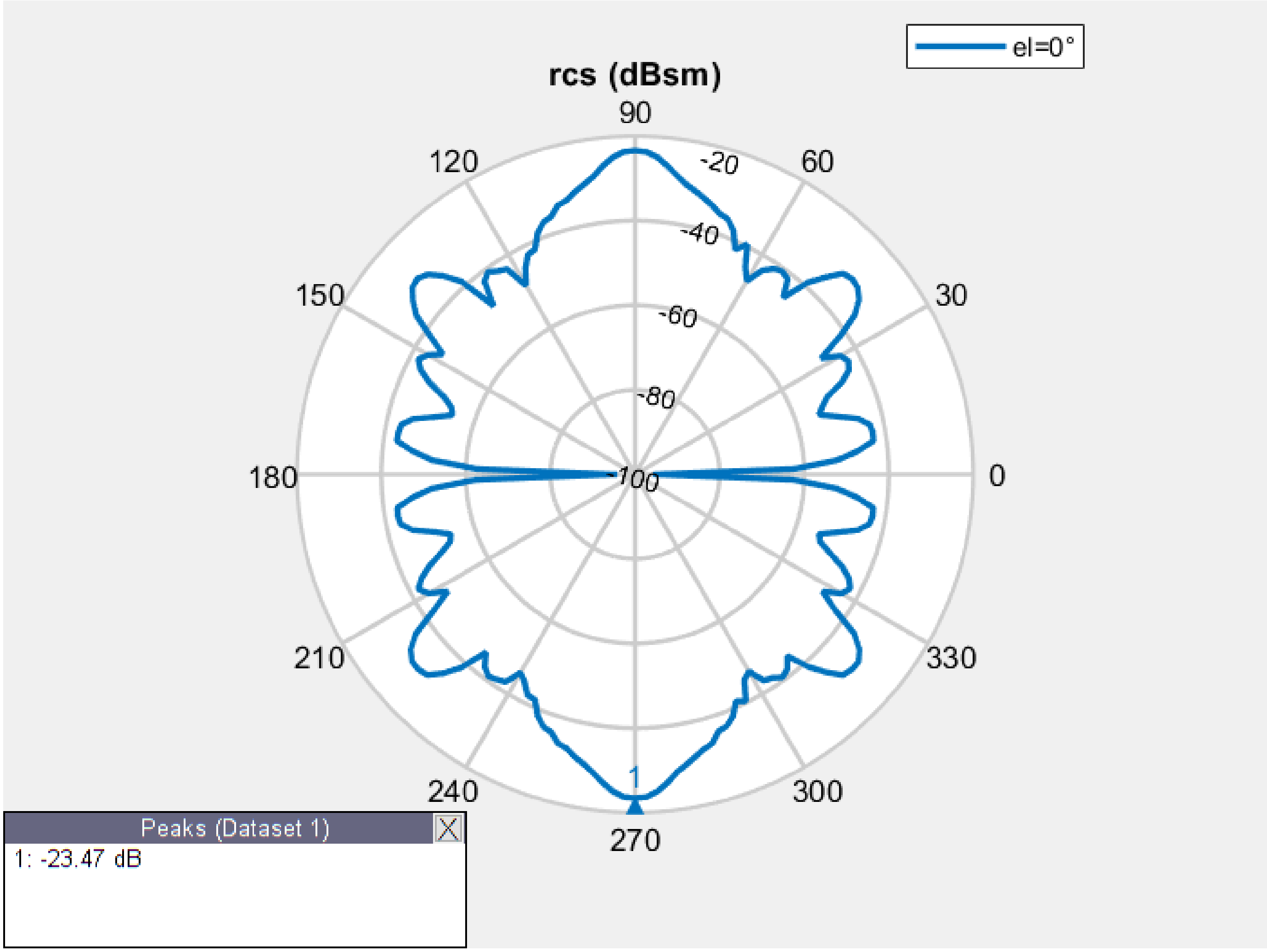


Peaks (Dataset 1)

1: -30.76 dB

Plot the RCS in the azimuth plane.

```
figure
rcs(c,f,0:2:360,0)
```



Bistatic RCS of Offset Cassegrain Antenna

Calculate bistatic RCS for a default offset cassegrain antenna at a frequency of 14 GHz.

```
S = rcs(cassegrainOffset,14e9,'TransmitAngle',[30;60],'ReceiveAngle',[30;45])
```

```
S = -1.2868
```

Input Arguments

object — Platform, antenna, or array element
object

Platform, antenna or array element, specified as an object.

frequency — Analysis frequency
real-valued scalar

Analysis frequency, specified as a real-valued scalar in Hz.

Example: `70e6`

Data Types: `double`

`azimuth` — Azimuth angles

`0` (default) | *N*-element real vector

Azimuth angles at which to visualize the RCS, specified as an *N*-element real vector in degrees.

Example: `90`

Data Types: `double`

`elevation` — Elevation angles

`0:5:360` (default) | *M*-element real vector

Elevation angles at which to visualize the RCS, specified as an *M*-element real vector in degrees.

Example: `0:1:360`

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` pair 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: `'CoordinateSystem','polar'`

`CoordinateSystem` — Coordinate system in which to visualize RCS

`'polar'` (default) | `'rectangular'`

Coordinate system in which to visualize the RCS, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of these values: `'polar'` or `'rectangular'`.

Example: `'CoordinateSystem','rectangular'`

Data Types: `char`

`Scale` — Scale at which to visualize or compute RCS

`'log'` (default) | `'linear'`

Scale at which to visualize or compute the RCS, specified as the comma-separated pair consisting of `'Scale'` and `'log'` or `'linear'`. When you choose `'log'`, the RCS is calculated and plotted in dBsm.

Example: `'Scale','linear'`

Data Types: `char`

`Polarization` — Transmit and receive wave polarization

`'VV'` (default) | `'HH'` | `'HV'` | `'VH'`

Transmit and receive wave polarization, specified as the comma-separated pair consisting of `'Polarization'` and one of these values:

- 'HH' – Horizontal polarized field is transmitted and received.
- 'VV' – Vertical polarized field is transmitted and received.
- 'VH' – Vertical polarized field is transmitted, and horizontal polarized field is received.
- 'HV' – Horizontal polarized field is transmitted, and vertical polarized field is received.

Example: 'Polarization','VV'

Data Types: char

EnableGPU — Use GPU to perform RCS calculations
0 (default) | 1

Use GPU to perform RCS calculations, specified as the comma-separated pair consisting of 'EnableGPU' and 0 to disable GPU or 1 to enable GPU.

Example: 'EnableGPU',1

Data Types: logical

TransmitAngle — Transmit wave angle
2-by-*N* real matrix

Transmit wave angle, specified as the comma-separated pair consisting of 'TransmitAngle' and a 2-by-*N* real matrix representing azimuth and elevation pairs, with each element unit in degrees.

Example: 'TransmitAngle',[30;60]

Data Types: double

ReceiveAngle — Receive wave angle
2-by-*M* real matrix

Receive wave angle, specified as the comma-separated pair consisting of 'ReceiveAngle' and a 2-by-*M* real matrix representing azimuth and elevation pairs, with each element unit in degrees.

Example: 'ReceiveAngle',[30;60]

Data Types: double

Solver — Solver for RCS analysis
'PO' (default) | 'MoM' | 'FMM'

Solver for RCS analysis, specified as the comma-separated pair consisting of 'Solver' and 'PO' (Physical Optics) or 'MoM' (Method of Moments) or 'FMM' (Fast Multipole Method).

Example: 'Solver', 'MOM'

Data Types: char

Type — Output type
'Magnitude' (default) | 'Complex'

Output type, specified as the comma-separated pair consisting of 'Type' and 'Magnitude' or 'Complex'.

Note Plotting rcs will error if the 'Type' is 'Complex'

Example: `'Type','Complex'`

Data Types: `char`

Output Arguments

`rcsval` — RCS value of platform, antenna, or array object

N-by-*M* real-valued array

RCS value of the platform, antenna, or array object, returned as an *N*-by-*M* real-valued array in dBsm. The size of the array is equal to the number of azimuth values (*N*) multiplied by the number of elevation values (*M*).

`azimuth` — Azimuth angles of calculated RCS pattern

N-element real-valued vector

Azimuth angles of the calculated RCS value, returned as an *N*-element real-valued vector in degrees.

`elevation` — Elevation angles of calculated RCS pattern

M-element real-valued vector

Elevation angles of the calculated RCS pattern, returned as an *M*-element real-valued vector in degrees.

More About

What Is RCS?

Radar Cross Section (RCS) is the measure of scattering cross section of an object interrogated by a plane wave. The assumption of a plane wave implies that the structure is in the far field of the radiator, which is typically a part of the radar system. RCS is a function of the object's shape, the frequency of the radar, the angle of interrogation of the wave, and the object's material parameters. RCS can also be measured in logarithmic units of dBsm, which is dB relative to a 1 m^2 reference area.

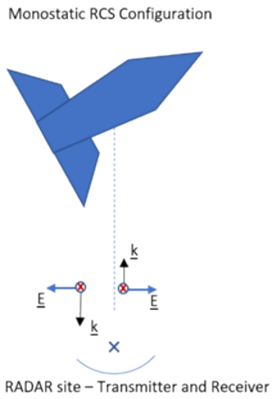
RCS is calculated using two typical configurations:

- Monostatic
- Bistatic

By default, the `rcs` function calculates a monostatic RCS. To calculate a bistatic RCS, restrict the `'TransmitAngle'` to 2-by-1.

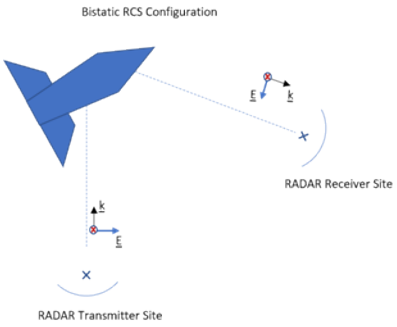
Monostatic RCS

The monostatic RCS configuration is characterized by a radar system that transmits a signal and receives the backscattered signal from the object being interrogated at the same site. The source of the transmitted electromagnetic waves and the receiving system for the scattered wave are co-located.


Monostatic RCS Configuration
k
E
E
k
RADAR site – Transmitter and Receiver

Bistatic RCS

In the bistatic RCS configuration, the radar system consists of a fixed radar transmitting site and a fixed or mobile receiving site captures the backscattered waveform from the object.



RCS Calculation

RCS is calculated in both a scalar form and a matrix form. Equations for both forms include electric (E) and magnetic (H) field quantities calculated or measured in the far field of the scattering object.

Scalar Form

In the scalar form of RCS, σ is defined as a ratio of the squared backscattered-field to the squared incident field, given by the equation:

$$\sigma = \lim_{r \to \infty} 4\pi r^2 \frac{\left|E^{\mathrm{s}}\right|^2}{\left|E^{\mathrm{i}}\right|^2}$$

where E^s and E^i represent the scattered and incident electric fields at a specific point in 3-D space.

Matrix Form

The matrix form of the RCS decomposes the incident and the scattered fields into horizontal and vertical polarizations and then computes the ratios of the various combinations between the scattered and incident fields, given by the equation:

$$\begin{pmatrix} \sigma_{HH} & \sigma_{HV} \\ \sigma_{VH} & \sigma_{VV} \end{pmatrix} = \lim_{r \to \infty} 4\pi r^2 \begin{pmatrix} \dfrac{\left|E_H^{\mathrm{s}}\right|^2}{\left|E_H^{\mathrm{i}}\right|^2} & \dfrac{\left|E_H^{\mathrm{s}}\right|^2}{\left|E_V^{\mathrm{i}}\right|^2} \\ \dfrac{\left|E_V^{\mathrm{s}}\right|^2}{\left|E_H^{\mathrm{i}}\right|^2} & \dfrac{\left|E_V^{\mathrm{s}}\right|^2}{\left|E_V^{\mathrm{i}}\right|^2} \end{pmatrix}$$

where ${E^s}_H$ and ${E^i}_H$ represent the horizontal polarized components of the scattered and incident electric fields at a given point in 3-D space. ${E^s}_V$ and ${E^i}_V$ represent the vertical polarized components of the scattered and incident electric fields at a given point in 3-D space.

Version History

Introduced in R2019b

References

[1] Gurel, L., H. Bagrci, J. C. Castelli, A. Cheraly, F. Tardivel. "Validation Through Comparison: Measurement and Calculation of the Bistatic Radar Cross Section of a Stealth Target." *Radio Science*. Vol. 38, Number 3, 2003, pp.12-1 - 12-8.

[2] Rao, S.M., D. R. Wilton, A. W. Glisson. "Electromagnetic Scattering by Surfaces of Arbitrary Shape." *IEEE Trans. Antennas and Propagation*. Vol. AP-30, Number 3, 1982, pp.409-418.

[3] Jakobus, U., F. M. Landstorfer. "Improved PO-MM Formulation for Scattering from Three-Dimensional Perfectly Conducting Bodies of Arbitrary Shape.." *IEEE Trans. Antennas and Propagation*. Vol. AP-43, Number 2, 1995, pp.162-169.

See Also

`patternAzimuth` | `patternElevation`

rectspirallength2turns

Calculate number of turns for specified arm length in rectangular spiral antenna

Syntax

`Nturns = rectspirallength2turns(ant,reqtotalarmlength)`

Description

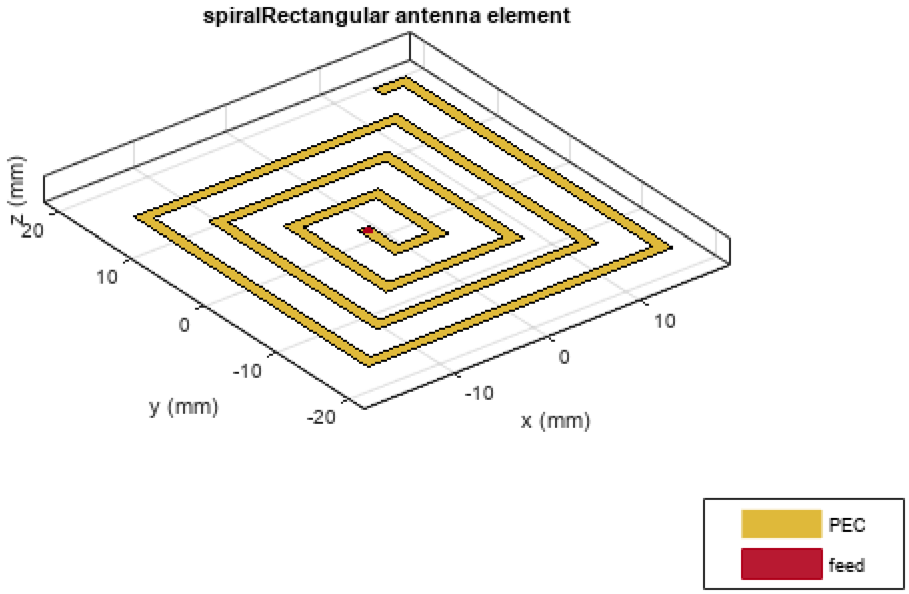
`Nturns = rectspirallength2turns(ant,reqtotalarmlength)` calculates the equivalent number of turns for a specified total arm length in a rectangular spiral antenna.

Examples

Rectangular Spiral Antenna with Specified Arm Length

Create a single arm rectangular spiral antenna with a total arm length of 291 mm.

```
ant = spiralRectangular('NumArms',1,'NumTurns',3,'InitialLength',4.5e-3,...
              'InitialWidth',4.5e-3,'Spacing',3.3e-3,'StripWidth',1.2e-3);
nT = rectspirallength2turns(ant,291e-3);
ant.NumTurns = nT;
figure;
show(ant);
```



Input Arguments

`ant` — Rectangular spiral antenna

`spiralRectangular` object

Rectangular spiral antenna, specified as a `spiralRectangular` object.

`reqtotalarmlength` — Total length of arm

scalar in meters

Total length of the rectangular spiral antenna arm, specified as a scalar in meters. In case of dual arm, the input takes the length of any one of the arms.

Example: 33e-3

Output Arguments

`Nturns` — Equivalent number of turns

scalar

Equivalent number of turns for a specified total arm length, returned as a scalar.

Version History

Introduced in R2020a

See Also

`spiralRectangular`

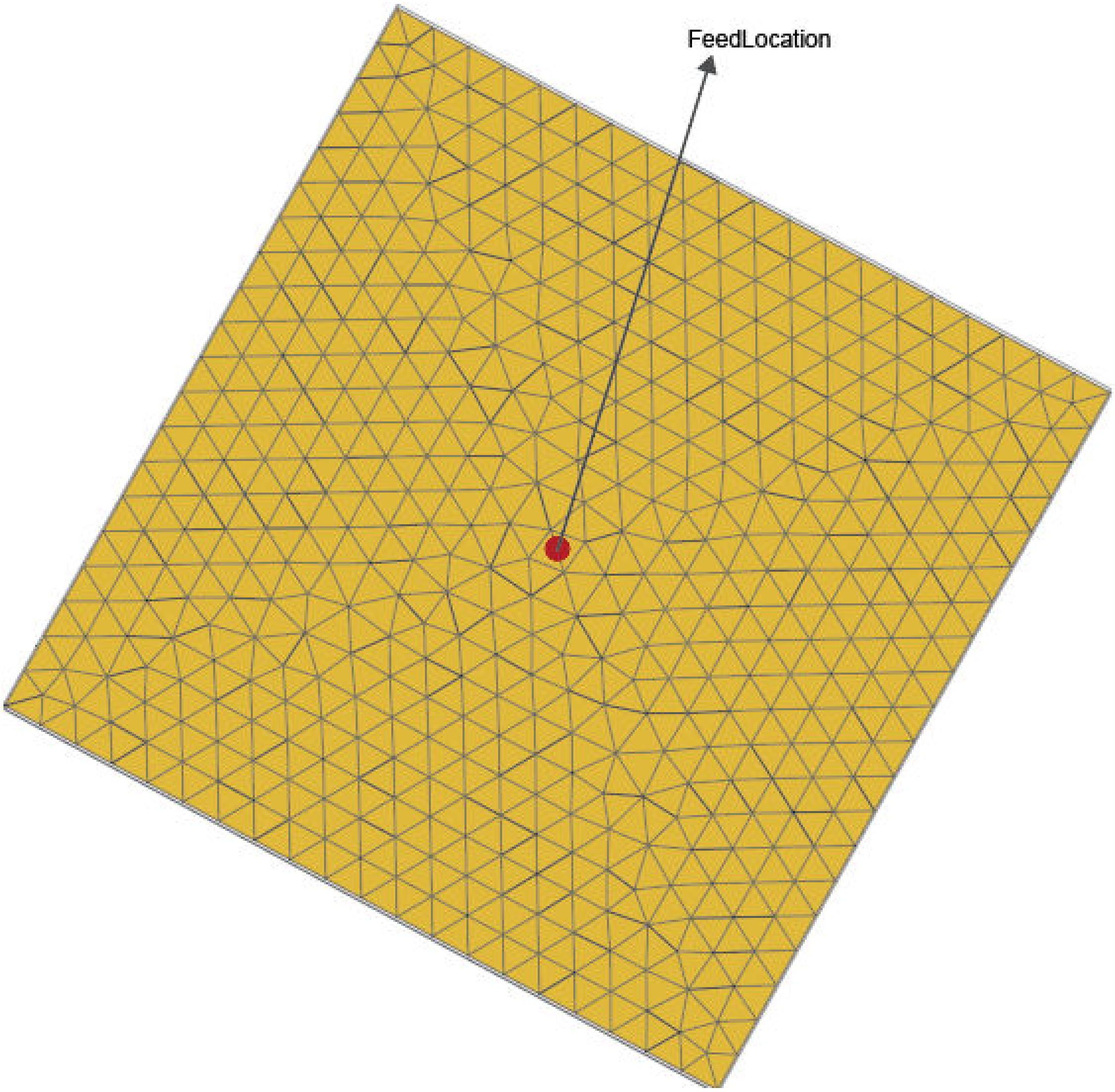
createFeed

Create feed location for `customAntennaStl` object

Syntax

```
createFeed(antenna,FeedLocation,NumEdges)
createFeed(antenna)
```

Description



`createFeed(antenna,FeedLocation,NumEdges)` creates antenna feed for a `customAntennaStl`object using the feed location defined in `FeedLocation`and the number of edges specified in NumEdges. The antenna feed is created along the triangular edges defined in FeedLocation.

createFeed(antenna) opens a UI figure window from which you can interactively create the antenna feed for a customAntennaStlobject. The figure window has two panes: **Slice Antenna** and **Add Feed** .

Examples

Create Feed for customAntennaStl Object

Create antenna feed for a customAntennaStl object using the command-line interface. First create a customAntennaStl object with default properties.

```
ant = customAntennaStl
```

```
ant = 
  customAntennaStl with properties:

          FileName: []
             Units: 'm'
      FeedLocation: []
    AmplitudeTaper: 1
        PhaseShift: 0
     UseFileAsMesh: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

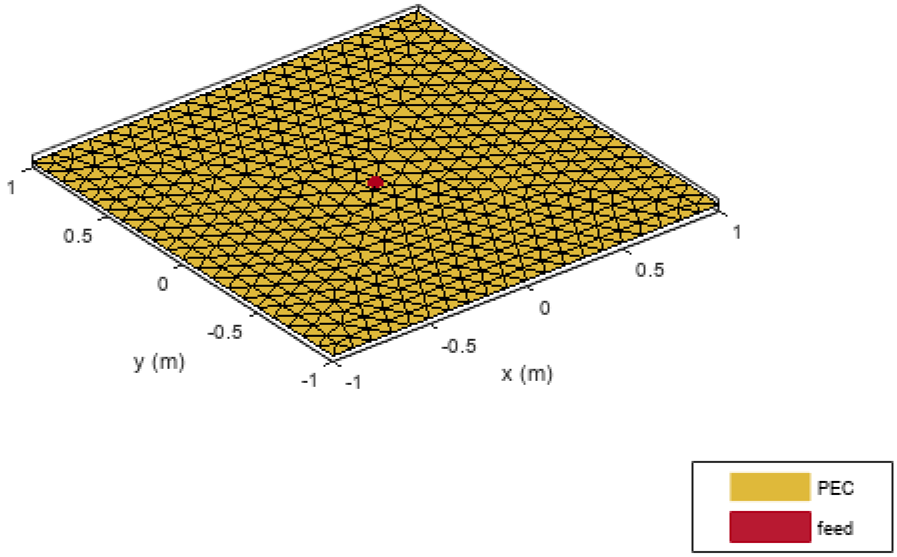
Specify the file name of the STL file to determine the antenna structure.

```
ant.FileName ='plateMesh.stl'
```

```
ant = 
  customAntennaStl with properties:

          FileName: 'plateMesh.stl'
             Units: 'm'
      FeedLocation: []
    AmplitudeTaper: 1
        PhaseShift: 0
     UseFileAsMesh: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

Specify FeedLocation and NumEdges and display the antenna structure.

```
ant.createFeed([0,0,0], 1)
show (ant)
```



Create Feed Using UI Figure Window

Create a `customAntennaStl` object with default properties.

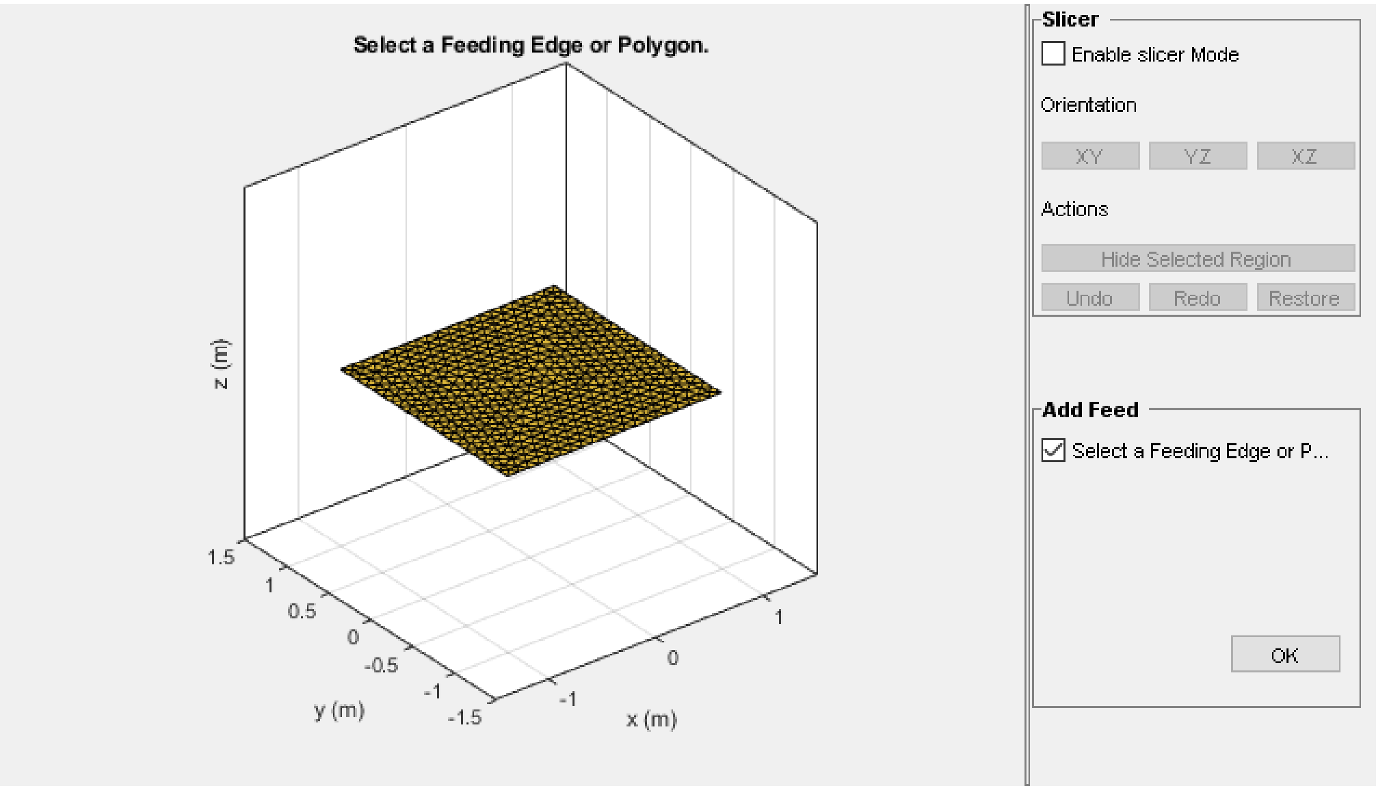
```
ant= customAntennaStl;
```

Import the STL file.

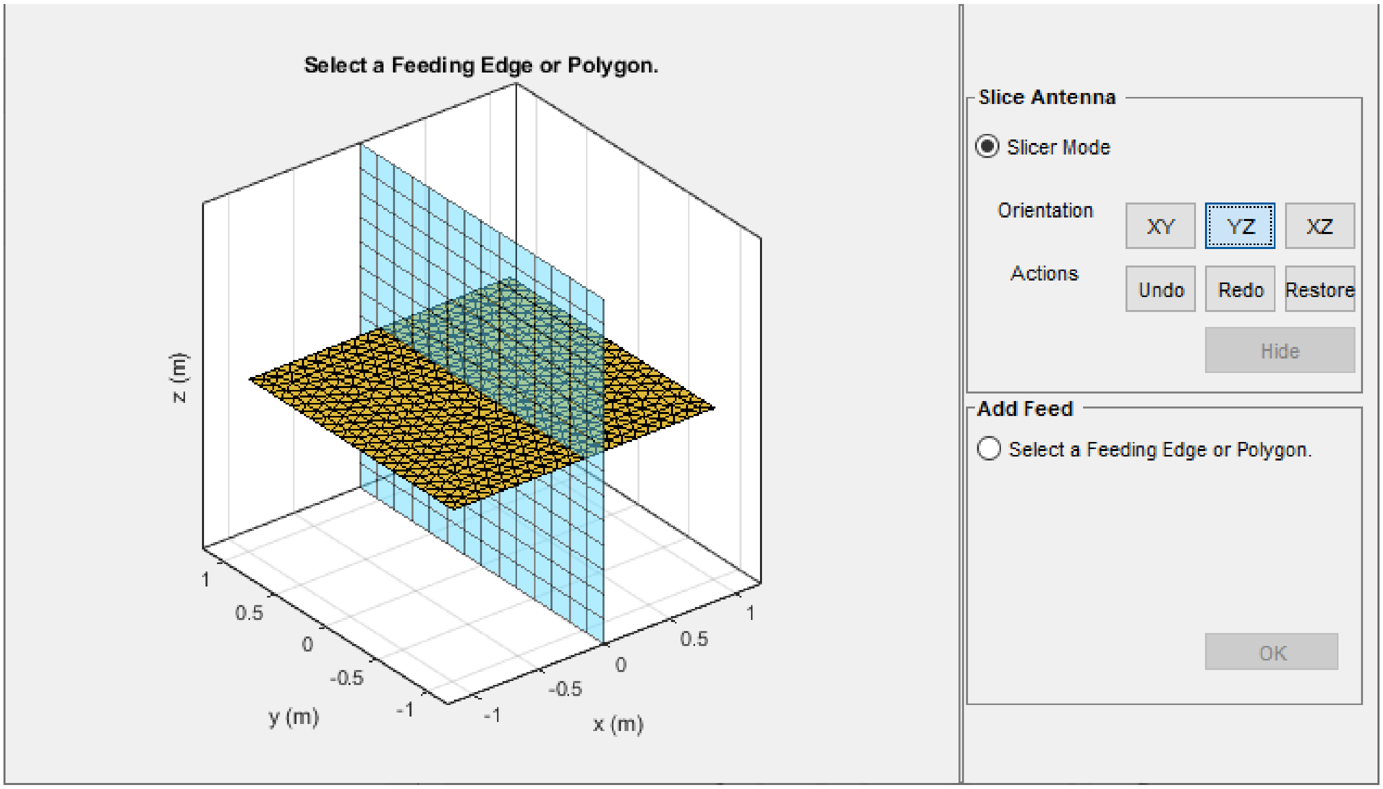
```
ant.FileName = 'plateMesh.stl';
```

Open the UI figure window.

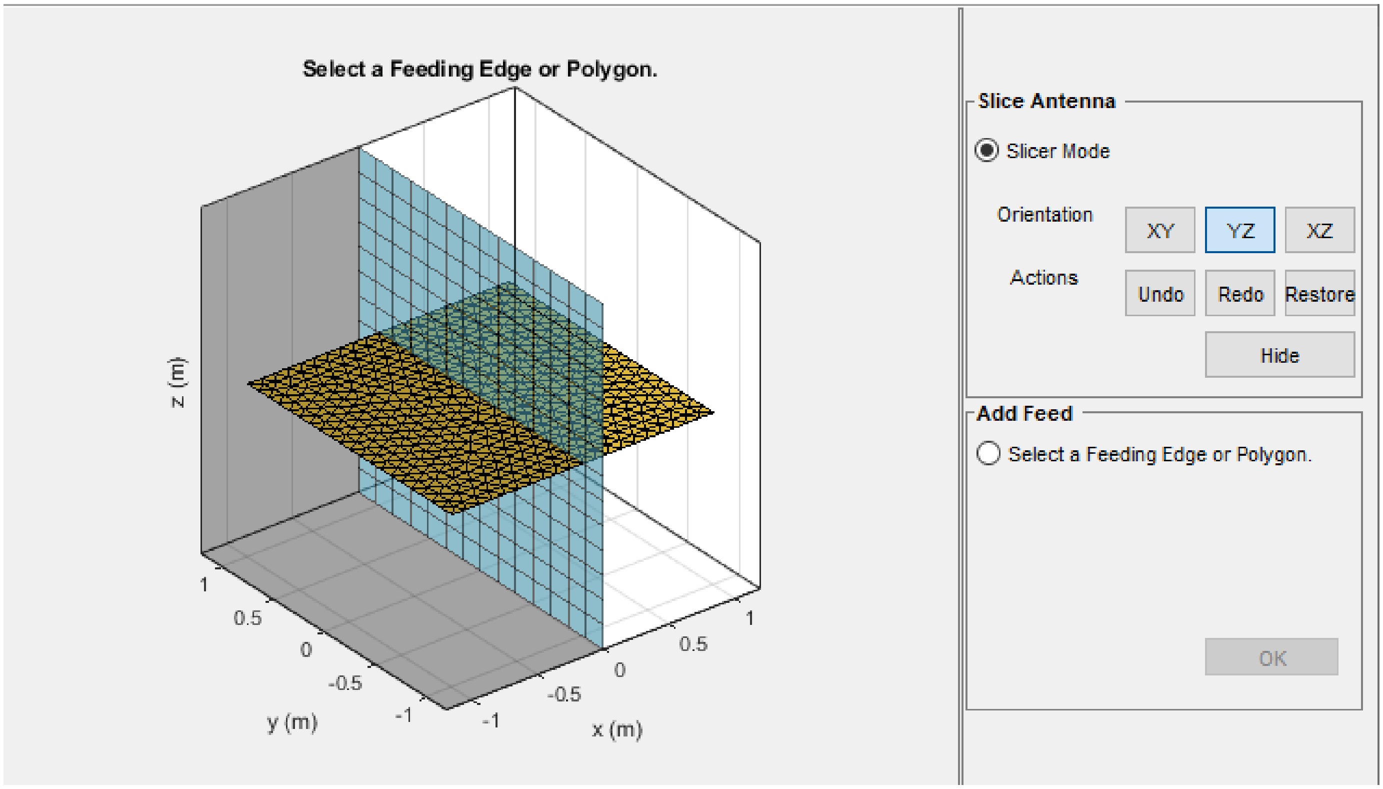
```
createFeed(ant);
```



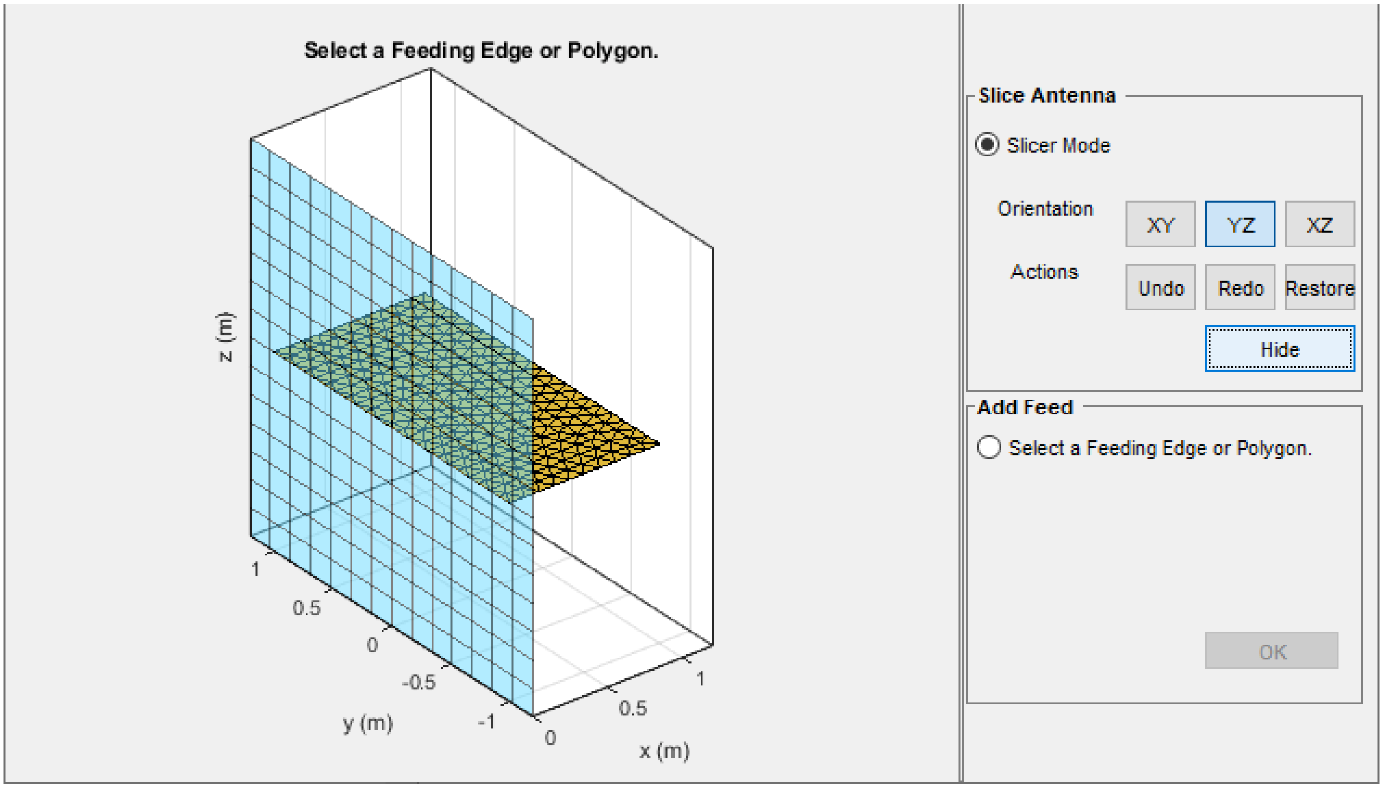
The UI figure window consists of two panes, **Slice Antenna** and the **Add Feed** pane. Select the **Slicer Mode**, then click **YZ** to select that as the plane along which to slice your antenna.



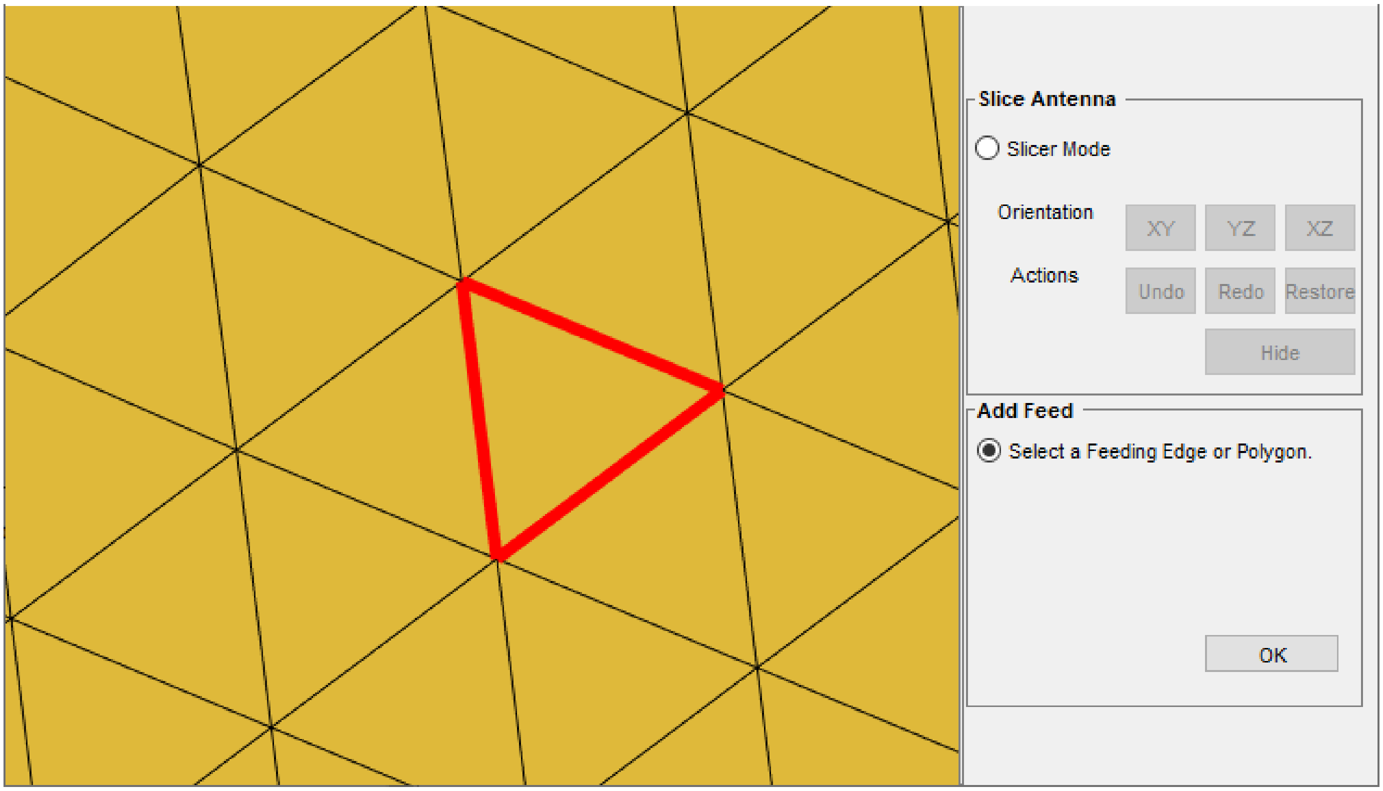
Select the region you want to hide and then click **Hide** to hide the selected region.



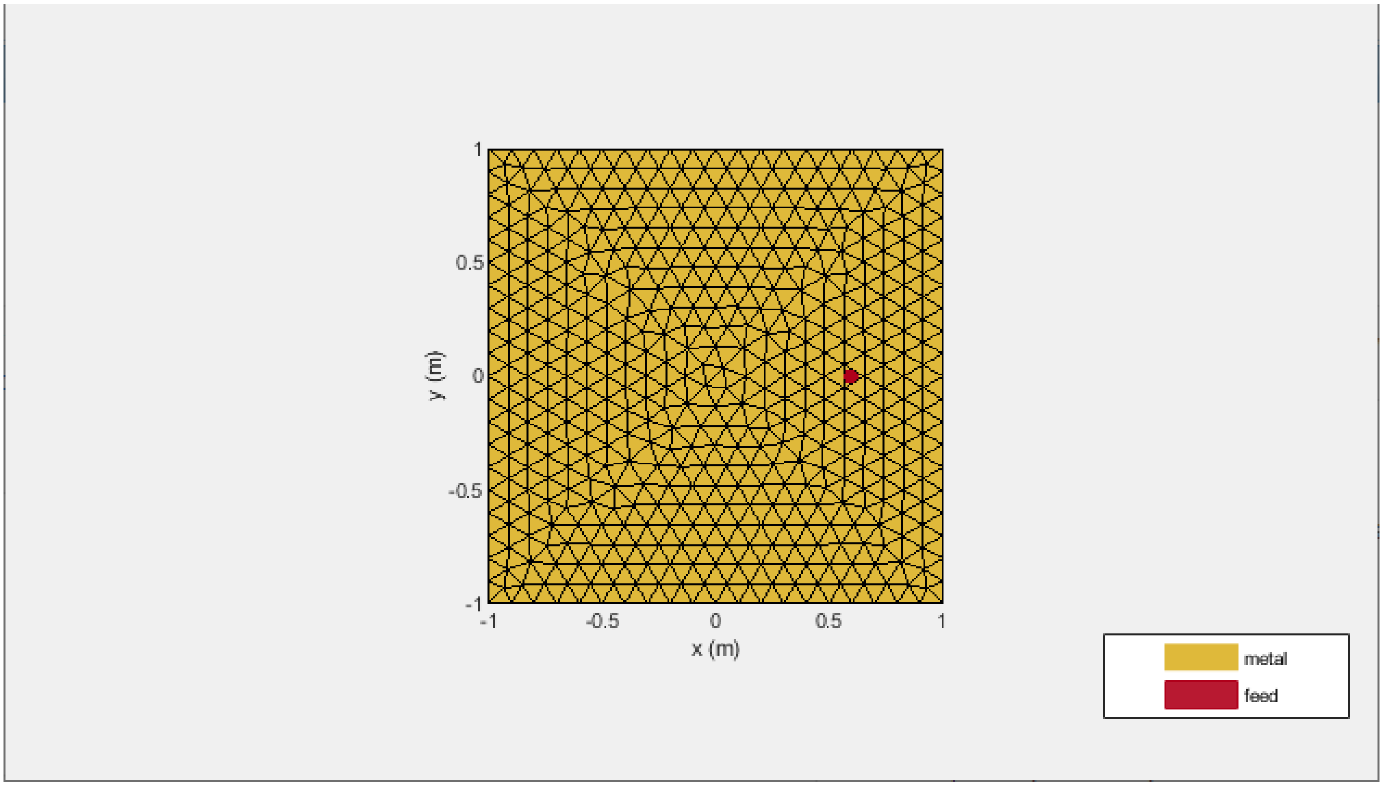
Repeat the process until you reach the region of interest.



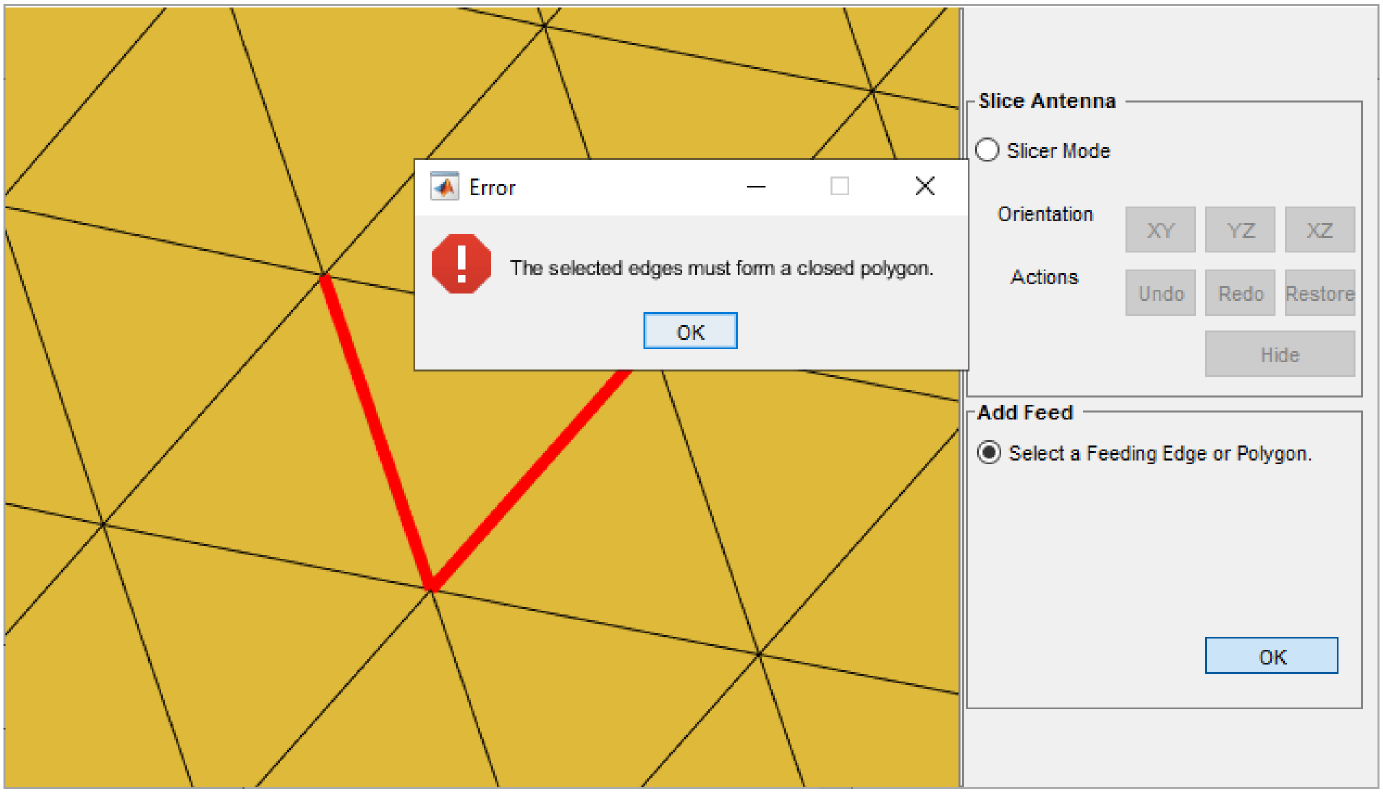
Select **Select a Feeding Edge or Polygon** under the **Add Feed** pane to select the edges to form a closed polygon. Click **OK** to define the selected edges as the feeding edges.



The feed location is displayed.



The selected edges must be connected to other edges, else UI figure window will display an error.



Error

The selected edges must form a closed polygon.

Input Arguments

antenna — Custom antenna

`customAntennaStl` object

Custom antenna stl object, specified as object.

NumEdges — Number of feeding edges

positive real scalar

Number of feeding edges, specified as a positive real scalar. You can also select the feeding edges using the UI figure window.

FeedLocation — Points to identify feed region

[ ] (default) | three-element vector

Points to identify antenna feed location, specified as Cartesian coordinates in meters. The three elements of the vector are the X-, Y-, and Z-coordinates, respectively.

Example: `createFeed(c,[0.07,0.01,0.02],1);`

Version History

Introduced in R2020a

See Also

customAntennaStl | returnLoss | sparameters

strip2cylinder

Calculates equivalent radius approximation for strip

Syntax

```
r = strip2cylinder(w)
```

Description

`r = strip2cylinder(w)` calculates the equivalent radius for a cylindrical approximation to a strip cross section.

Examples

Radius Approximation of Cylinder from Strip Width

Calculate the equivalent radius of a cylinder based on a strip of width 80 mm.

```
r1 = strip2cylinder(80e-3)
```

```
r1 = 0.0200
```

Calculate the equivalent cylindrical cross-sections radii using the strips of widths 80 mm, 88 mm, and 96 mm.

```
r2 = strip2cylinder([80e-3 88e-3 96e-3])
```

```
r2 = 1×3

    0.0200    0.0220    0.0240
```

Input Arguments

w — Width of strip
scalar | vector

Width of strip, specified as a scalar in meters or a vector with each element unit in meters.

Output Arguments

r — Equivalent cylindrical cross-section radius
scalar | vector

Equivalent cylindrical cross-section radius, returned as a scalar in meters or a vector with each element unit in meters.

Example: 20e-3

Version History

Introduced in R2020a

See Also

`cylinder2strip`

numGridsToSpacing

Calculate grid spacing in grid for reflectorGrid object

Syntax

numGridsToSpacing(antenna,numGrids,GridWidth)

Description

numGridsToSpacing(antenna,numGrids,GridWidth) calculates the spacing between cells in the grid given the number of grid cells, numGrids, and the width of the grid cells, GridWidth.

Examples

Calculate Grid Spacing in Grid Reflector-Backed Antenna

Calculate the spacing between grid cells in the reflectorGrid antenna object with given number of grid cells as 4 and the grid width of 0.026 m.

```
ant = reflectorGrid;
numGridsToSpacing(ant,4,0.026)
```

```
ans = 0.0240
```

Input Arguments

antenna — Grid reflector-backed antenna
reflectorGrid object

Grid reflector-backed antenna, specified as a reflectorGrid object.

numGrids — Number of grid cells
positive scalar

Number of grid cells in the reflector, specified as a positive scalar.

GridWidth — Width of grid cells
0.022 (default) | positive scalar

Width of each grid cell of the grid reflector-backed antenna, specified as a positive scalar in meters.

Example: numGridsToSpacing(ant,4,0.3);

Version History

Introduced in R2020b

See Also

`reflectorGrid`

optimize

Optimize antenna or array using SADEA optimizer

Syntax

```
optimizedelement = optimize(element,frequency,objectivefunction,
propertynames,bounds)
optimizedelement = optimize( ___ ,Name,Value)
```

Description

`optimizedelement = optimize(element,frequency,objectivefunction, propertynames,bounds)` optimizes the antenna or the array at the specified frequency using the specified objective function and the antenna or array properties and their bounds.

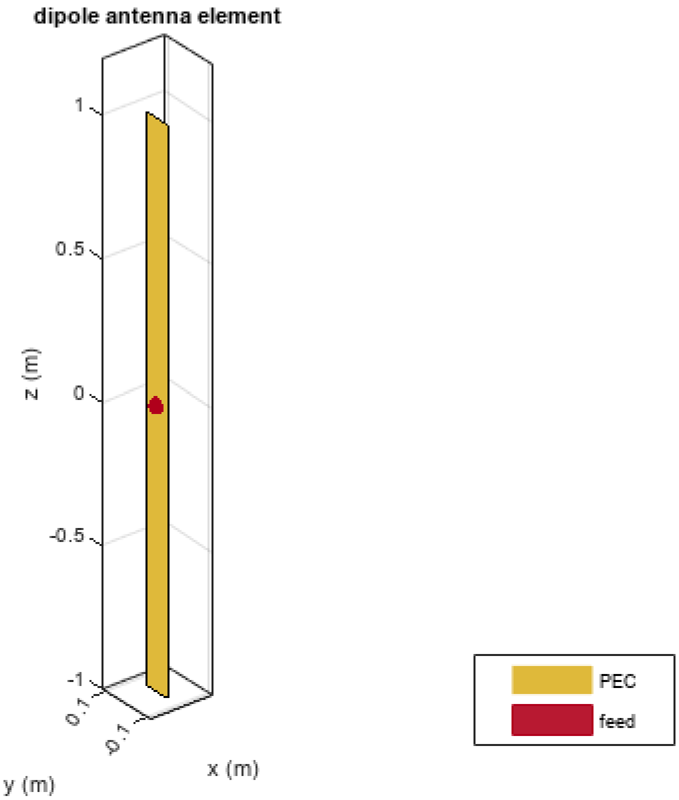
`optimizedelement = optimize( ___ ,Name,Value)` optimizes the antenna or the array using additional name value pairs.

Examples

Maximize Gain of Dipole Antenna

Create and view a default dipole antenna.

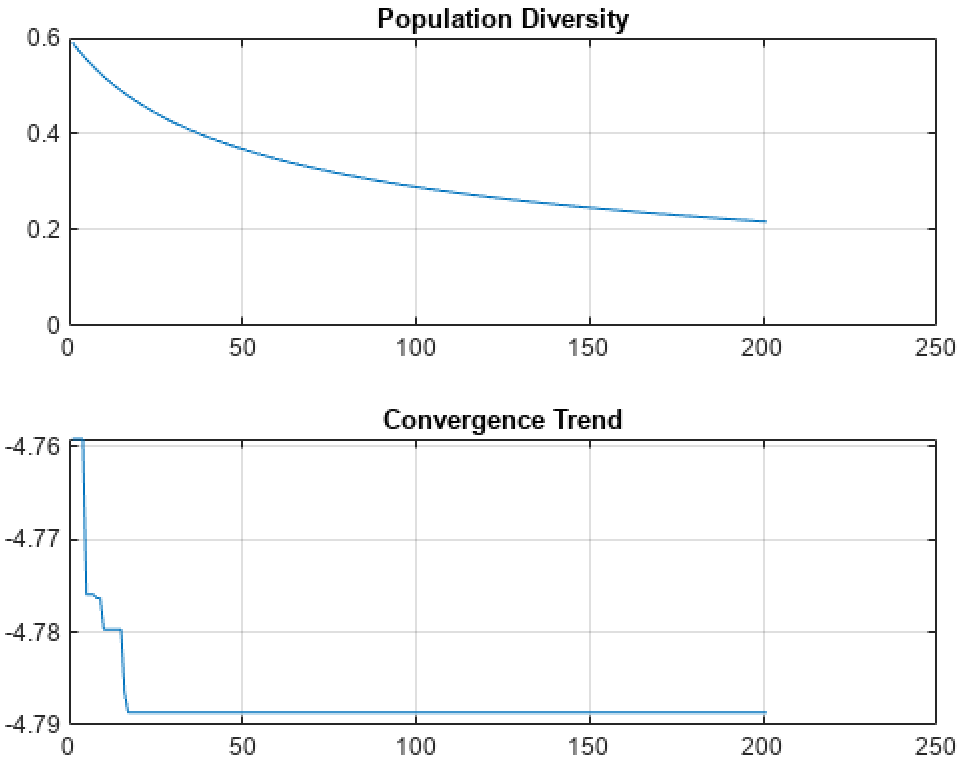
```
ant = dipole;
show(ant)
```



Maximize the gain of the antenna by changing the antenna length from 3 m to 7 m and the width from 0.11 m to 0.13 m.

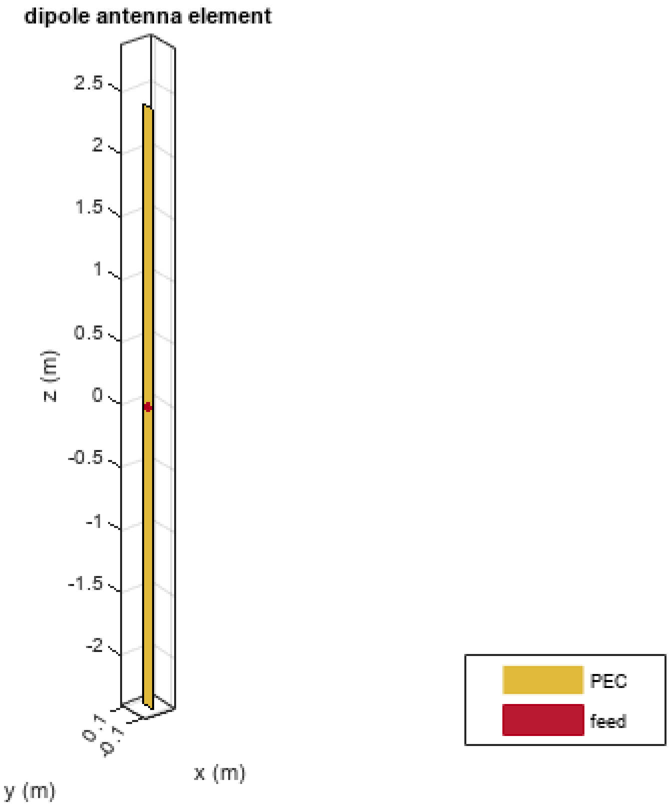
Optimize the antenna at a frequency of 75 MHz.

```
optAnt = optimize(ant, 75e6, 'maximizeGain', ...
                 {'Length', 'Width'}, {3 0.11; 7 0.13})
```



```
optAnt = 
  dipole with properties:

        Length: 4.7864
         Width: 0.1102
    FeedOffset: 0
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
show(optAnt)
```



Input Arguments

`element` — Antenna or array element

object

Antenna or array element, specified as an antenna object from the “Antenna Catalog” or array object from the “Array Catalog”.

`frequency` — Frequency of antenna or array analysis during optimization

nonnegative scalar

Frequency of the antenna or array analysis during optimization, specified as a nonnegative scalar in hertz.

Data Types: `double`

`objectivefunction` — Objective of antenna or array optimization

`'maximizeGain'` | `'fronttoBackLobeRatio'` | `'maximizeBandwidth'` | `'minimizeBandwidth'` | `'maximizeSLL'` | `'minimizeArea'`

Objective of antenna or array optimization, specified as one of the following:

- `'maximizeGain'` — Maximize the gain of the given antenna or array element
- `'fronttoBackRatio'` — Increase the front-lobe-to-back-lobe ratio of the antenna or array element

- `'maximizeBandwidth'` — Maximize the operation bandwidth of the antenna or array element. Use this objective function for optimizing antennas or arrays for wideband applications.
- `'minimizeBandwidth'` — Minimize the operation bandwidth of the antenna or array element. Use this objective function for optimizing antennas or arrays for narrowband applications.
- `'maximizeSLL'` — Maximize the ratio between the front lobe and the first side lobes of the antenna or array pattern.
- `'minimizeArea'` — Minimizes the maximum area occupied by the antenna or the array element. If the dimension of the element in the array is smaller than the aperture, the objective function minimizes the array aperture.

Data Types: `string` | `char`

`propertynames` — Properties of optimizing antenna or array

cell array character vectors

Properties of optimizing antenna or array, specified as a cell array of character vectors. The property names are selected as the design variables in optimization.

Data Types: `cell`

bounds — Lower and upper bounds of design variables

two-row cell array

Lower and upper bounds of design variables, specified as a two-row cell array.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `optAnt = optimize(ant, 75e6, 'maximizeGain',{'Length', 'Width'}, {3 0.11; 7 0.13})`

`Constraints` — Optimization constraints

cell array of strings or character vectors

Antenna or array optimization constraints, specified as the comma-separated pair consisting of `'Constraints'` and a cell array of strings or character vectors. Each character vector or string must be of the form: (analysis function) (inequality sign) (value). You can specify any of the following analysis functions:

- `'Area'` in meter square
- `'Volume'` in meter cube
- `'S11'` in dB
- `'Gain'` in dBi
- `'F/B'` in dBi
- `'SLL'` in dBi

The inequality signs `'<'` or `'>'` and the values specifies the analysis function limits. For example, `Area < 0.03` indicates that the area of the optimizing antenna must be lesser than 0.03 square meter.

Example: `'Constraints',{Area<0.03}`

Data Types: `char` | `string`

`Weights` — Weight or penalty of each constraint function

vector of positive integers in the range (1,100)

Weight or penalty of each constraint function, specified as the comma-separated pair consisting of `'Weights'` and a vector of positive integers in the range (1,100). If the penalty is set to high, a higher priority is given to the constraint function in case of multiple constraint optimization. All constraint functions are weighted equally by default.

Example: `'Weights',8`

Data Types: `double`

`FrequencyRange` — Range of frequencies for vector frequency analysis

vector of nonnegative numbers

Range of frequencies for vector frequency analysis like S-parameters, specified as the comma-separated pair consisting of `'FrequencyRange'` and a vector of nonnegative numbers with each element unit in hertz.

The default frequency range is obtained from the center frequency considering a bandwidth of less than 10 percent.

Example: `'FrequencyRange',linspace(1e9,2e9,10)`

Data Types: `double`

`ReferenceImpedance` — Reference impedance of optimizing antenna or array

`50` (default) | scalar

Reference impedance of antenna or array being optimized, specified as the comma-separated pair consisting of `'ReferenceImpedance'` and a scalar in ohms

Example: `'ReferenceImpedance',50`

Data Types: `double`

`MainLobeDirection` — Azimuth and elevation of main lobe

`[0,90]` (default) | two-element vector

Azimuth and elevation of main lobe of antenna or array being optimized, specified as the comma-separated pair consisting of `'MainLobeDirection'` and a two-element vector with each element unit in degrees. The first element represents azimuth and the second element represents elevation.

Example: `'MainLobeDirection',[20 30]`

Data Types: `double`

`Iterations` — Number of iterations to run optimizer

`200` (default) | positive scalar

Number of iterations to run the optimizer after you build the model, specified as the comma-separated pair consisting of `'Iterations'` and a positive scalar.

Example: 'Iterations',40

Data Types: double

UseParallel — Use Parallel Computing Toolbox during optimization
false (default) | true

Use Parallel Computing Toolbox during optimization, specified as the comma-separated pair consisting of 'UseParallel' and true or false.

Example: 'UseParallel',true

Data Types: logical

EnableCoupling — Enable mutual coupling of elements in arrays during optimization
true (default) | false

Enable mutual coupling of elements in an array during optimization, specified as the comma-separated pair consisting of 'EnableCoupling' and true or false.

Example: 'EnableCoupling',false

Data Types: logical

EnableLog — Enable printing iteration number and value of convergence on command line
false (default) | true

Enable printing iteration number and value of convergence on the command line, specified as the comma-separated pair consisting of 'EnableLog' and true or false.

Example: 'EnableLog',true

Data Types: logical

Output Arguments

optimizedelement — Optimized antenna or array element
antenna or array object

Optimized antenna or array element, returned as an antenna or array object.

Version History

Introduced in R2020b

See Also

numCorrugationsToPitch

Calculate pitch for specified corrugations

Syntax

```
Pitch = numCorrugationsToPitch(antenna,corrugations)
```

Description

`Pitch = numCorrugationsToPitch(antenna,corrugations)` returns the pitch value for the specified number of corrugations. You can calculate the pitch for rectangular or conical corrugated horn antennas.

Examples

Calculate Pitch for Corrugated Horn Antenna

Calculate the pitch for a default corrugated horn antenna with 6 corrugations.

```
Pitch = numCorrugationsToPitch(hornCorrugated,6)
```

```
Pitch = 0.0045
```

Calculate Pitch for Conical Corrugated Horn Antenna

Calculate the pitch for a default conical corrugated horn antenna with 8 corrugations.

```
Pitch = numCorrugationsToPitch(hornConicalCorrugated,8)
```

```
Pitch = 0.0107
```

Input Arguments

antenna — Corrugated-horn antenna
`hornCorrugated` object | `hornConicalCorrugated` object

Corrugated horn antenna, specified as either a `hornCorrugated` or a `hornConicalCorrugated`object.

`corrugations` — Number of corrugations
positive scalar

Number of corrugations used in calculating the pitch, specified as a positive scalar.

Output Arguments

`Pitch` — Distance between two successive corrugations

positive scalar

Distance between two successive corrugations, specified as a positive scalar in meters.

Version History

Introduced in R2020b

See Also

`hornConicalCorrugated` | `hornCorrugated`

gerberRead

Create `PCBReader` object with specified Gerber and drill files

Syntax

```
P = gerberRead(T)
P = gerberRead([],B)
P = gerberRead(T,B)
P = gerberRead(T,B,D)
```

Description

`P = gerberRead(T)` creates a `PCBReader` object with the top layer Gerber file specified in `T`.

Note The `PCBReader` object reads RS-274X Gerber files. It does not support RS-274D Gerber files.

`P = gerberRead([],B)` creates a `PCBReader` object with the bottom layer Gerber file specified in `B`.

`P = gerberRead(T,B)` creates a `PCBReader` object with the specified top and bottom layer Gerber files.

`P = gerberRead(T,B,D)` creates a `PCBReader` object with the specified top and bottom layer Gerber files and the drill file specified in `D` .

Examples

Import and View Top Layer Gerber File

Use the `gerberRead` function to import a top layer Gerber file.

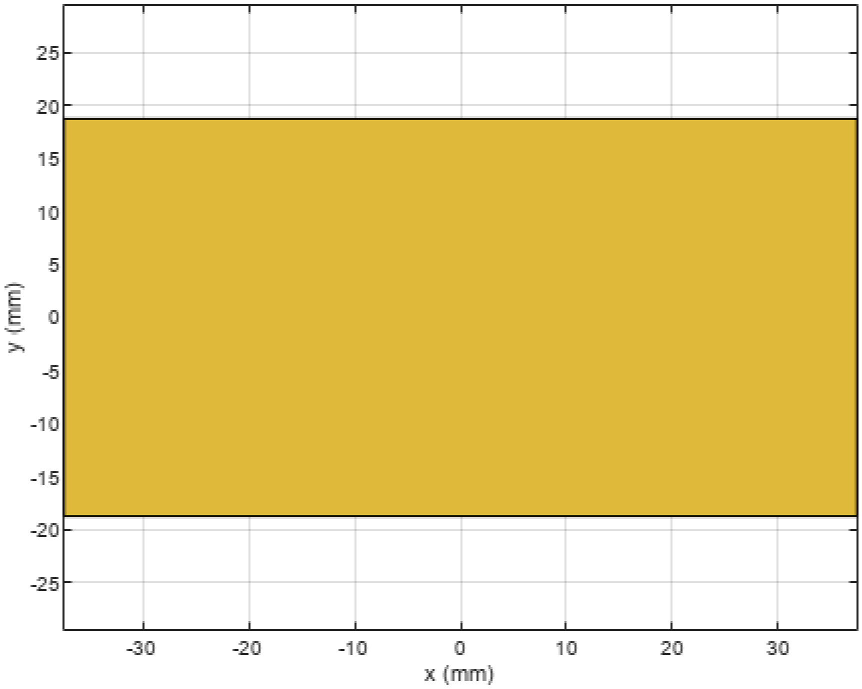
```
P = gerberRead('antenna_design_file.gtl');
```

Extract the metal layer from the file using the `shapes` function.

```
s = shapes(P);
```

View the top metal layer.

```
show(s)
```



Create PCBReader Object with Bottom Layer Gerber File

Import a bottom layer Gerber file to layer 4 of the stack.

```
P = gerberRead([],'UWBVivaldi.gbl');
P.StackUp
```

```
ans = 
  stackUp with properties:

    NumLayers: 5
       Layer1: [1x1 dielectric]
       Layer2: []
       Layer3: [1x1 dielectric]
       Layer4: 'UWBVivaldi.gbl'
       Layer5: [1x1 dielectric]
```

Create Antenna Model Using Top and Bottom Layer Gerber Files

Use the `gerberRead` function to import top and bottom layer Gerber files.

```
P = gerberRead('antenna_design_file.gtl','antenna_design_file.gbl');
```

Display the stack.

```
P.StackUp
```

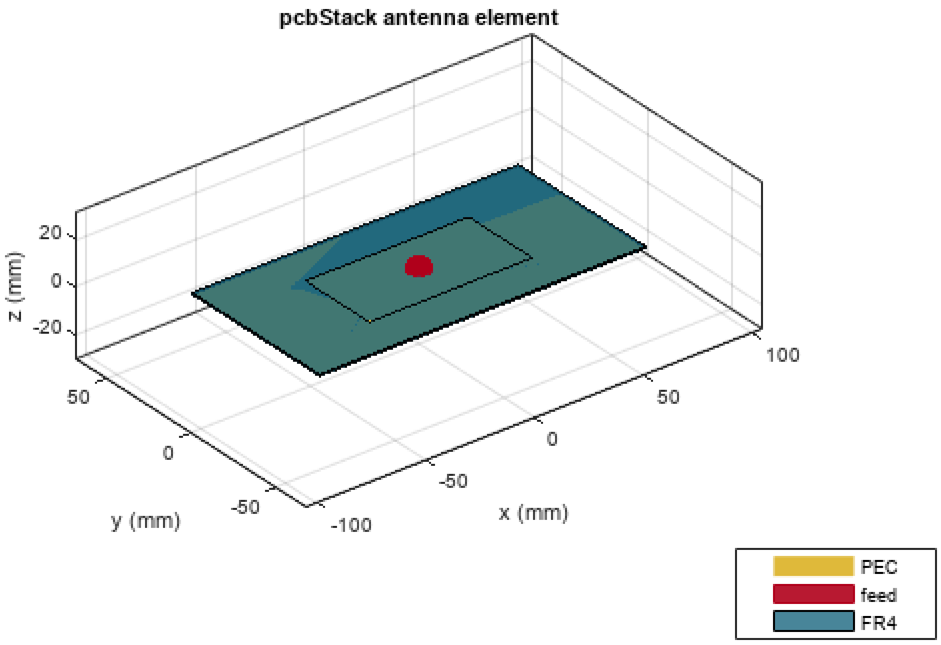
```
ans = 
  stackUp with properties:

    NumLayers: 5
       Layer1: [1x1 dielectric]
       Layer2: 'antenna_design_file.gtl'
       Layer3: [1x1 dielectric]
       Layer4: 'antenna_design_file.gbl'
       Layer5: [1x1 dielectric]
```

Modify the third layer in the stack, which is the dielectric layer between the top and bottom metal layers.

```
S = P.StackUp;
S.Layer3 = dielectric('Name','FR4','EpsilonR', 4.4, 'Thickness', 0.8e-3);
P.StackUp = S;
```

Create the antenna model by calling the `pcbStack` object on the PCB reader.

```
pb2 = pcbStack(P);
figure
show(pb2)
```



Input Arguments

T — Top layer Gerber file

character vector | string scalar

Top layer Gerber file, specified as a character vector or string scalar. The file should be saved as a GTL file.

Example: `gerberRead('Filetop.gtl');`

B — Bottom layer Gerber layer

character vector | string scalar

Bottom layer Gerber file, specified as a character vector or string scalar. The file should be saved as a GBL file.

Example: `gerberRead([],'FileBottom.gbl');`

D — Drill file

character vector | string scalar

Drill file, specified a character vector or string scalar. You can specify either a DRL or a TXT file.

Example: `gerberRead('Filetop.gtl','FileBottom.gbl','FileDrill.txt');`

Output Arguments

P — Read files

`PCBReader` object

Read Gerber and drill files, returned as a `PCBReader` object.

Limitations

Limitations of the `gerberRead` function while reading a gtl or gbl file are:

Command	Description	Comments
AM	Aperture Macro	Not supported
AB	Aperture Block	Not supported
SR	Step and Repeat	Not supported
TF	File Attributes	Command is ignored and no error is thrown
TA	Aperture Attributes	Command is ignored and no error is thrown
TO	Object Attributes	Command is ignored and no error is thrown
TD	Delete Attributes	Command is ignored and no error is thrown

Cut-ins are not supported.

Version History

Introduced in R2020b

See Also

`shapes` | `PCBReader` | `PCBServices` | `PCBConnectors` | `PCBWriter`

Topics

“Create Antenna Model from Gerber Files”

coneangle2size

Calculates equivalent cone height, broad radius, and narrow radius for cone

Syntax

```
conedimensions = coneangle2size(slantheight,halfconeangle,Name,Value)
```

Description

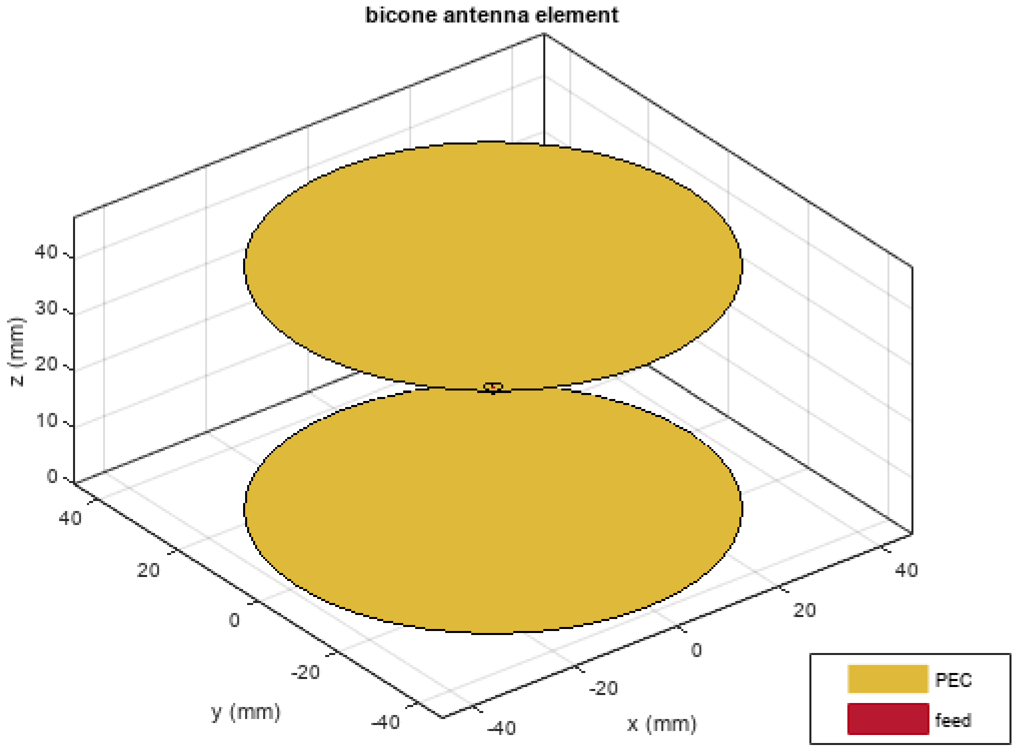
`conedimensions = coneangle2size(slantheight,halfconeangle,Name,Value)` calculates the equivalent cone height,broad radius, and narrow radius for a cone from its half cone angle,slant length, and either feedwidth or narrow radius

Examples

Dimensions of Bicone Antenna Using Feedwidth

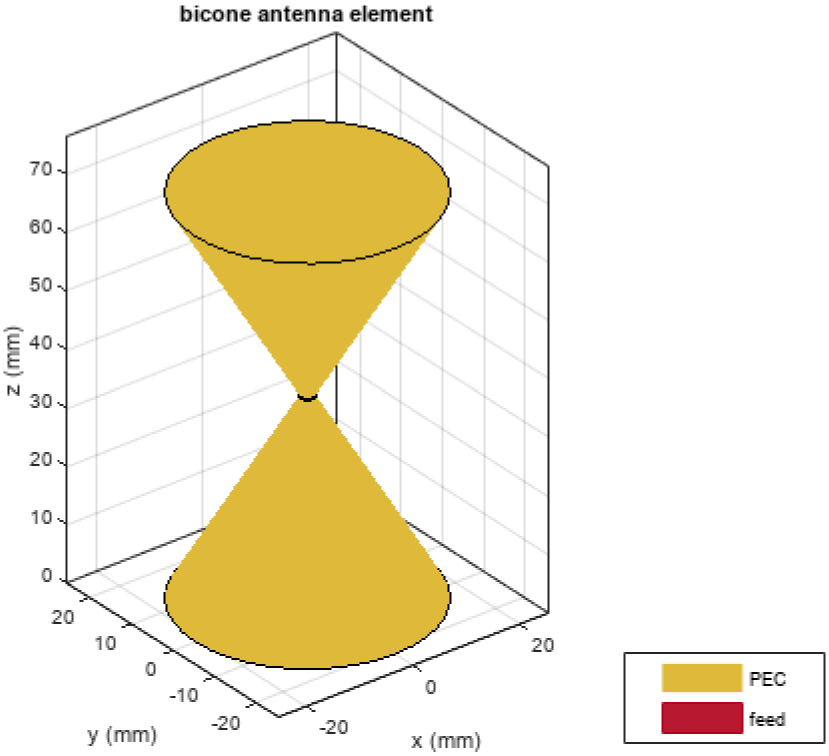
Calculate the cone height, the broad radius, and the narrow radius of the cone in a bicone antenna using a half cone angle of 30 degrees, slant length of 0.0400 m, feed width of 0.001 m.

```
ant = bicone('FeedHeight',0.3e-3,'FeedWidth',0.5e-3);
show(ant)
```



```
m = coneangle2size(30,40e-3,'FeedWidth',1e-3)

m = struct with fields:
    NarrowRadius: 0.0013
     BroadRadius: 0.0213
      ConeHeight: 0.0346
```

```
ant.ConeHeight = m.ConeHeight;
ant.BroadRadius = m.BroadRadius;
ant.NarrowRadius = m.NarrowRadius;
show(ant)
```



Input Arguments

`slantheight` — Length from base of cone to point on circle with narrow radius
positive scalar

Length from the base of the cone to point on the circle with the narrow radius, specified as a positive scalar in meters.

Data Types: `double`

`halfconeangle` — Half of cone angle
positive scalar

Half of the cone angle, specified as a positive scalar in degrees. This value must be between 5 and 85 degrees.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'FeedWidth' = 0.02`

`FeedWidth` — Width of feed

positive scalar

Width of the feed, specified as the comma-separated pair consisting of `'FeedWidth'` and a positive scalar in meters.

Data Types: `double`

`NarrowRadius` — Radius at apex of cone

positive scalar

Radius at the apex of the cone, specified as the comma-separated pair consisting of `'NarrowRadius'` and a positive scalar in meters.

Data Types: `double`

Version History

Introduced in R2020a

See Also

`bicone` | `discone` | `monocone` | `biconeStrip` | `disconeStrip`

shapes

Extract and modify metal layers from `PCBReader` object

Syntax

```
shapes(B)
```

Description

`shapes(B)` extracts and modifies the individual metal layers from a `PCBReader` object.

Examples

Extract and Modify Metal Layer

Use the `gerberRead` function to import top-layer Gerber file.

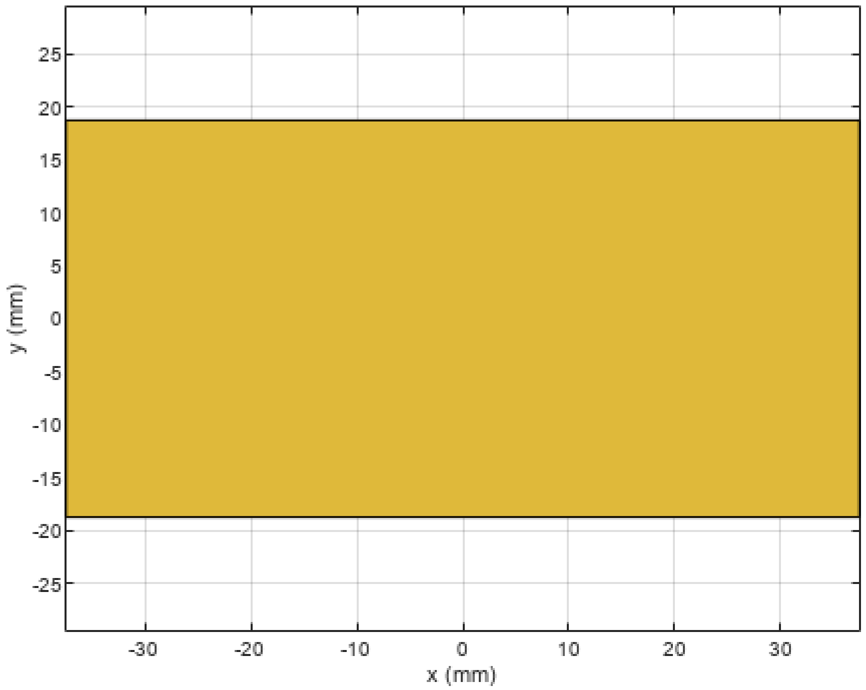
```
P = gerberRead('antenna_design_file.gtl');
```

Extract the metal layer from the file using the `shapes` function.

```
S = shapes(P);
```

View the metal layer.

```
figure
show(S)
```

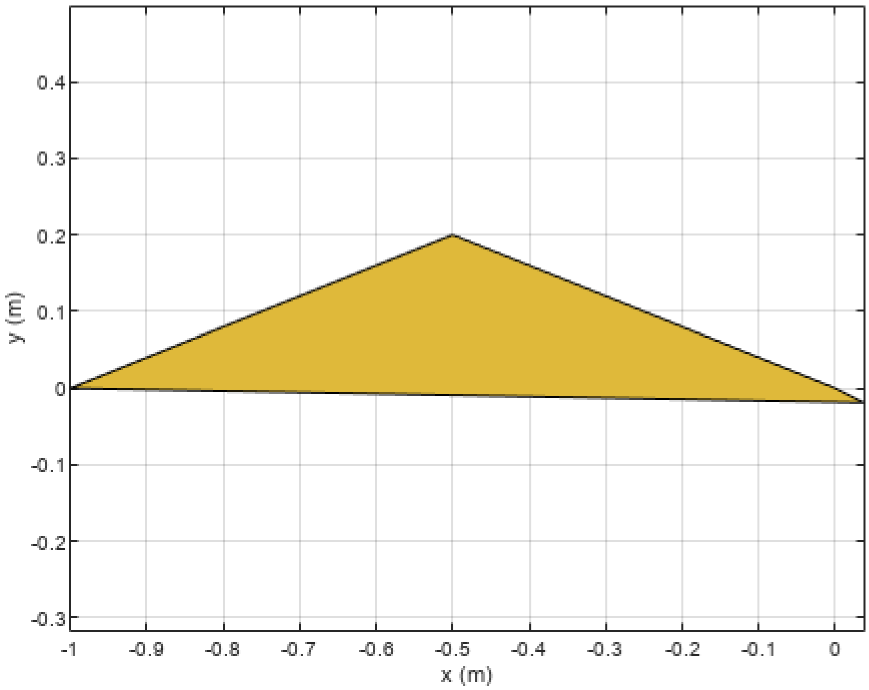


Modify the metal layer.

```
S.Vertices = [-1 0 0;-0.5 0.2 0;0 0 0;0.0375 -0.0188 0];
```

View the modified metal layer.

```
show(S)
```



Extract Metal from Two-Layer Design PCBReader Object

Create a `PCBReader` object.

```
B = PCBReader;
```

Import a two-layer design.

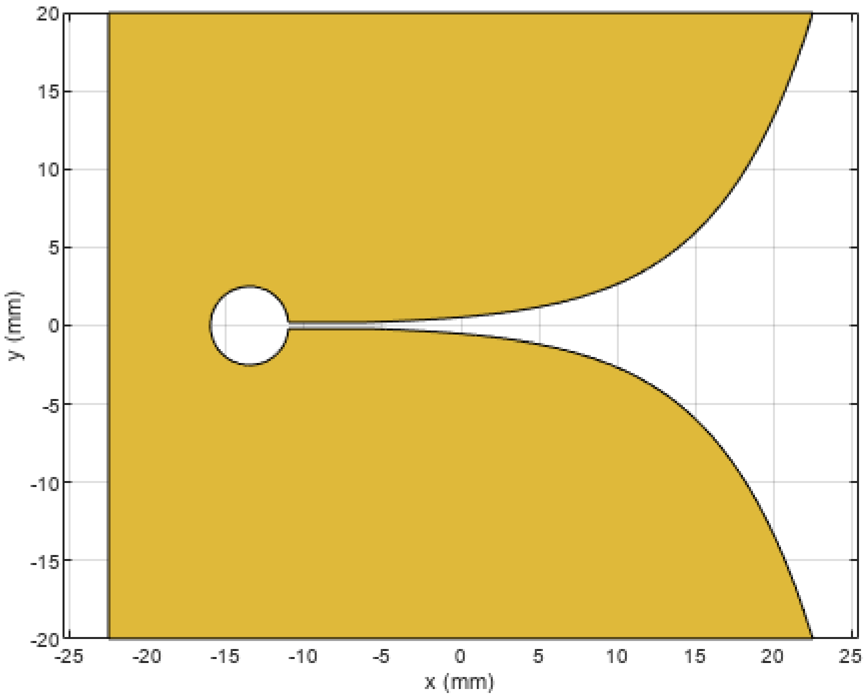
```
st = B.StackUp;
st.Layer2 = 'UWBVivaldi.gtl';
st.Layer4 = 'UWBVivaldi.gbl';
B.StackUp = st;
```

Extract shapes from the metal layers.

```
S = shapes(B);
```

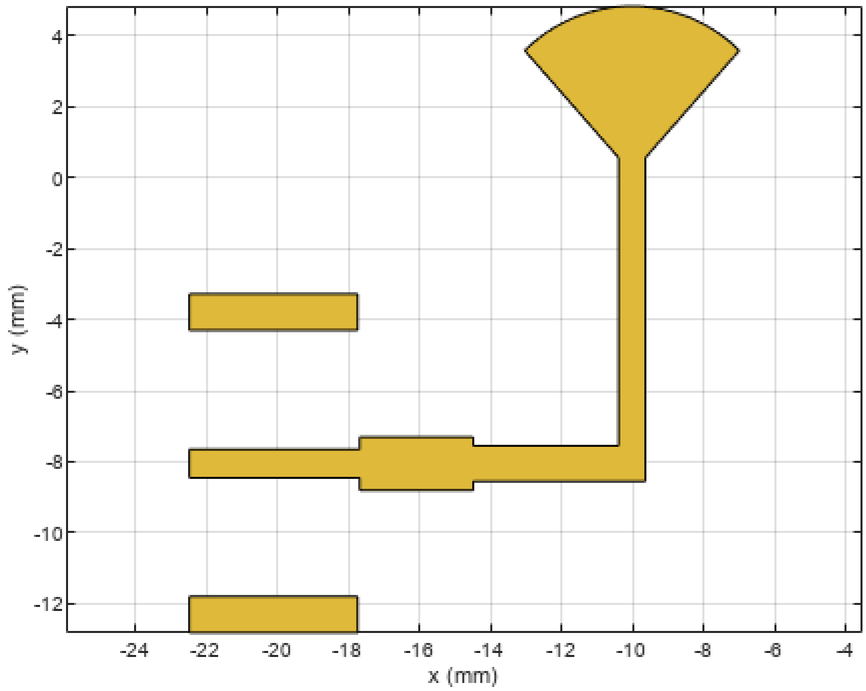
View the top-layer Gerber file.

```
figure
show(S(1))
```



View the bottom-layer Gerber file.

```
figure
show(S(2))
```



Input Arguments

B — PCB reader
PCBReader object

PCB reader, specified as a `PCBReader` object.

Example: `B = gerberRead('antenna_desgin_file.gbl')`

Version History

Introduced in R2020b

See Also

`PCBReader` | `gerberRead` | `removeHoles` | `removeSlivers`

removeSlivers

Remove sliver outliers from boundary of shape

Syntax

```
s = removeSlivers(shapeobject,slivertol)
```

Description

`s = removeSlivers(shapeobject,slivertol)` removes sliver outliers from boundary of shape.

Examples

Remove Slivers from Rectangle Shape for Antenna

Create two rectangle shapes. Change the length and the center of orientation of the second rectangle to the values shown.

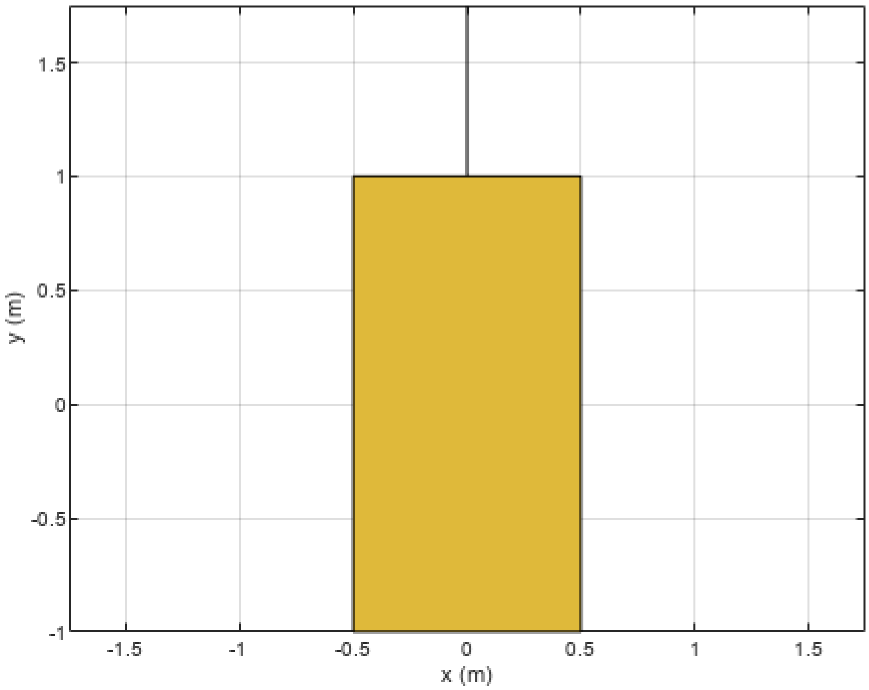
```
rect1 = antenna.Rectangle;
rect2 = antenna.Rectangle;
rect2.Length = 1e-7;
rect2.Center = [0,0.75];
```

Add rectangle 1 and rectangle 2.

```
rect3 = rect1 + rect2
```

```
rect3 = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [8x3 double]
```

```
show(rect3)
```

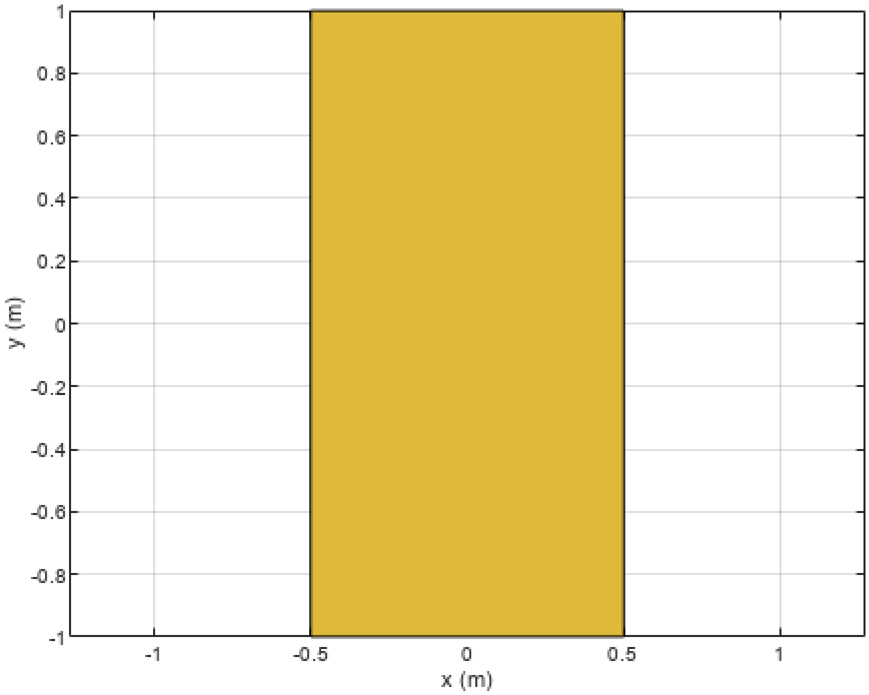


Remove slivers.

```
rect4 = removeSlivers(rect3,1e-6)
```

```
rect4 = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [4x3 double]
```

```
show(rect4)
```



Input Arguments

`shapeobject` — Antenna shape object with sliver outlier

`antenna.Circle` | `antenna.Recatangle` | `antenna.Polygon`

Antenna shape with sliver outlier, specified as `antenna.Circle`, `antenna.Rectangle`, `antenna.Polygon` objects, `antenna.Ellipse`, or the `shapes` function.

Data Types: `function`

`slivertol` — Sliver tolerance

nonnegative scalar

Sliver tolerance, specified as a nonnegative scalar.

Data Types: `double`

Version History

Introduced in R2020b

See Also

`removeHoles`

removeHoles

Remove holes from shape

Syntax

```
s = removeHoles(shapeobject,holetol)
```

Description

`s = removeHoles(shapeobject,holetol)` removes holes with area less than the provided tolerance from the shape.

Examples

Remove Holes from Rectangular Shape of Antenna

Create two rectangle shapes. Change the length and the center of orientation of the second rectangle to the values shown.

```
rect1 = antenna.Rectangle;
rect2 = antenna.Rectangle;
rect2.Length = 1e-7;
rect2.Width = 0.5
```

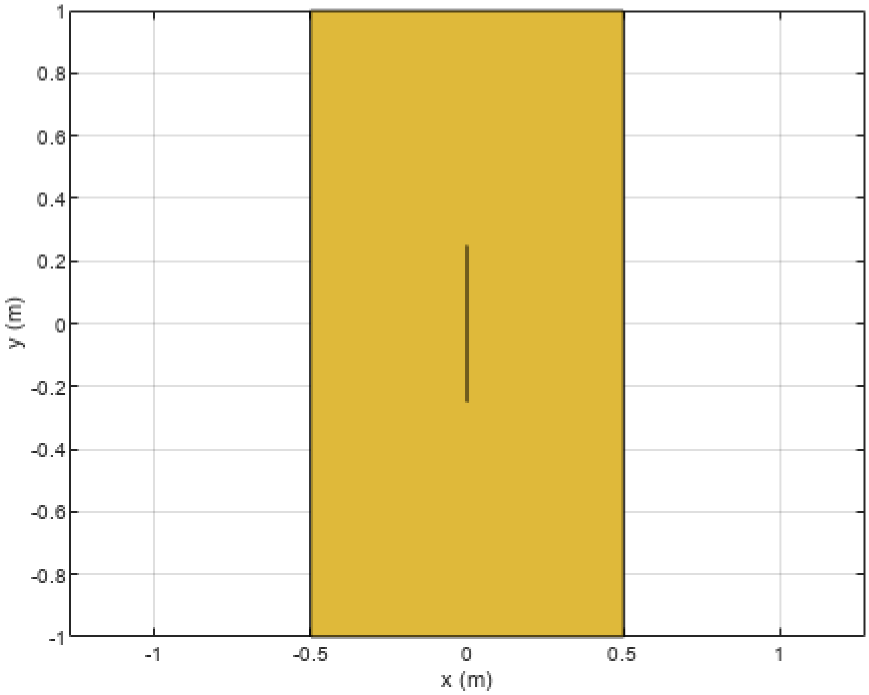
```
rect2 = 
  Rectangle with properties:

         Name: 'myrectangle'
       Center: [0 0]
       Length: 1.0000e-07
        Width: 0.5000
    NumPoints: 2
```

Subtract rectangle 1 from rectangle 2

```
rect3 = rect1-rect2
```

```
rect3 = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [9x3 double]
```

```
show(rect3)
```

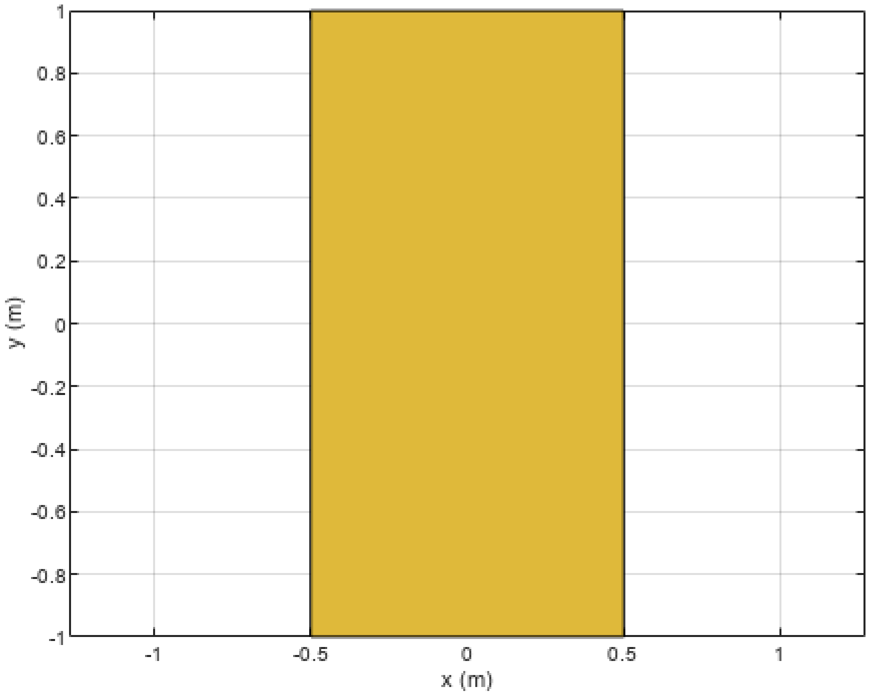


Remove holes from the new rectangle.

```
rect4 = removeHoles(rect3,1e-6)
```

```
rect4 = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [4x3 double]
```

```
show(rect4)
```



Input Arguments

`shapeobject` — Antenna shape object with sliver outlier
`antenna.Circle` | `antenna.Recatangle` | `antenna.Polygon` | `antenna.Ellipse`

Antenna shape with sliver outlier, specified as `antenna.Circle`, `antenna.Rectangle`, `antenna.Polygon` objects, `antenna.Ellipse`, or the `shapes` function.

Data Types: `function`

`holetol` — Hole tolerance
nonnegative scalar

Hole tolerance, specified as a nonnegative scalar.

Data Types: `double`

Version History

Introduced in R2020b

See Also

`removeSlivers`

metal

Conductor material

Syntax

```
m = metal(material)
m = metal(Name,Value)
```

Description

`m = metal(material)` returns the metal used as a conductor in the antenna elements. You can specify a material from the `MetalCatalog`. The default value for material is perfect electric conductor (PEC).

`m = metal(Name,Value)` returns the metal, based on the properties specified by one or more "Name-Value Pair Arguments" on page 4-395.

Examples

Monocone Antenna with Steel Conductor

Use steel as the as a conductor for a monocone antenna.

```
m = metal('Steel')
```

```
m = 
  metal with properties:

            Name: 'Steel'
    Conductivity: 6990000
       Thickness: 6.8000e-04

For more materials see catalog
```

Create a monocone antenna using the `monocone` antenna object.

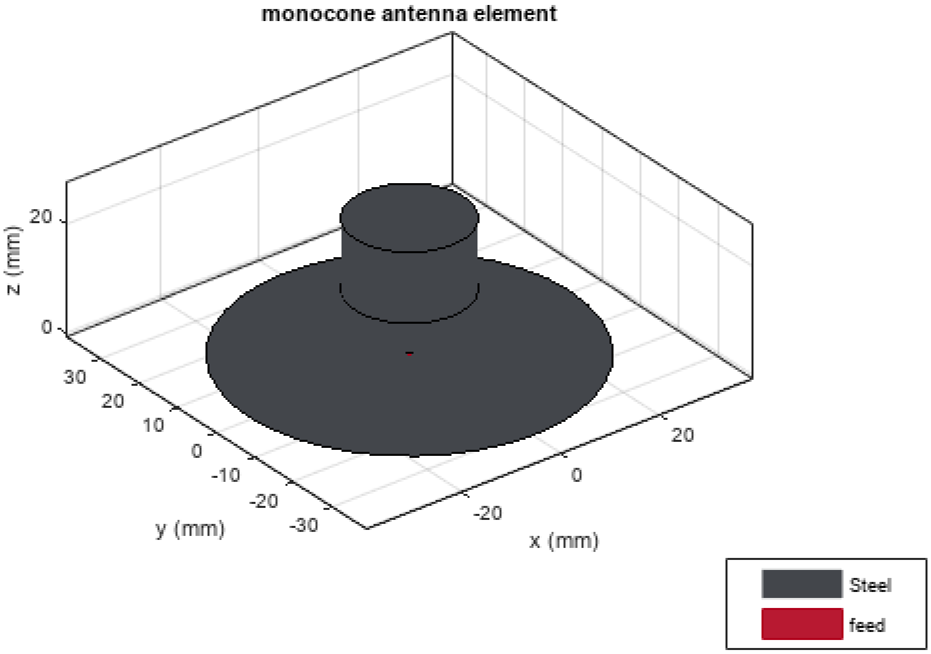
```
ant = monocone('Conductor',m)
```

```
ant = 
  monocone with properties:

                Radii: [5.0000e-04 0.0110 0.0110]
    GroundPlaneRadius: 0.0325
           ConeHeight: 0.0115
               Height: 0.0250
           FeedHeight: 5.0000e-04
            FeedWidth: 5.0000e-04
            Conductor: [1x1 metal]
                 Tilt: 0
             TiltAxis: [1 0 0]
```

```
                 Load: [1x1 lumpedElement]
```

View the antenna using the `show` function.

```
show(ant)
```



Customize Metal Properties

Create an annealed copper conductor with the conductivity of 5.8001e07 S/m and thickness of 1e-04 m.

```
m = metal('Name','Annealed Copper','Conductivity',5.8001e07,'Thickness',1e-04)
```

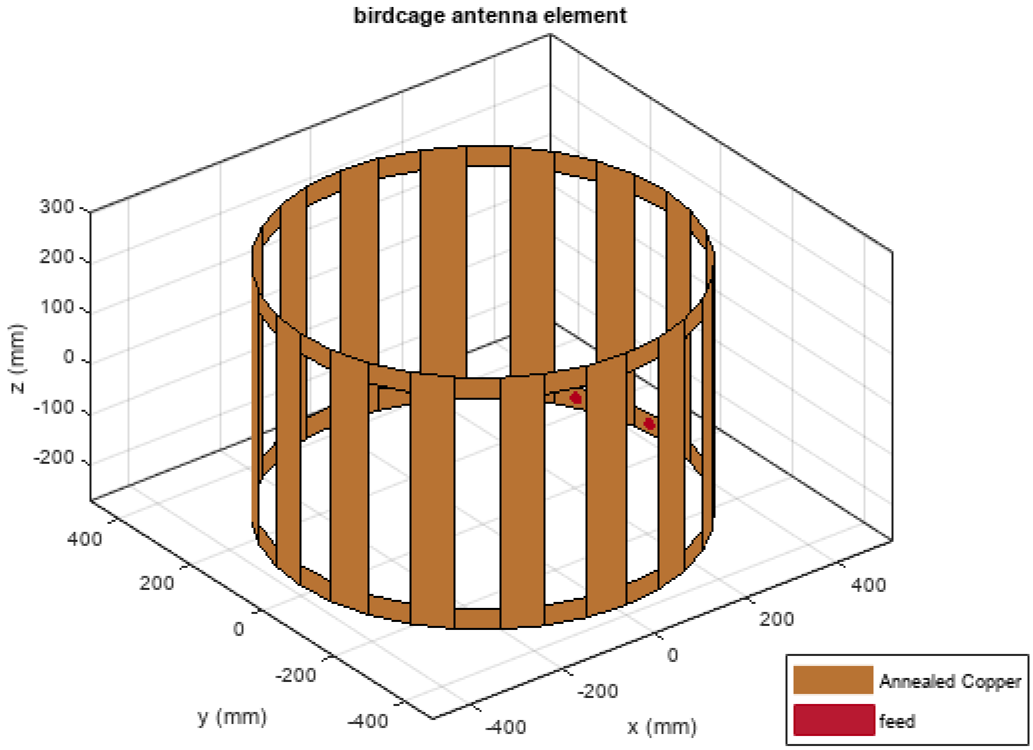
```
m = 
  metal with properties:

            Name: 'Annealed Copper'
    Conductivity: 58001000
       Thickness: 1.0000e-04

For more materials see catalog
```

Create a birdcage antenna using the annealed copper conductor.

```
ant = birdcage('Conductor',m)

ant = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: []
    FeedLocations: [2x3 double]
      FeedVoltage: 1
        FeedPhase: 0
        Conductor: [1x1 metal]
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

View the antenna using `show` function.

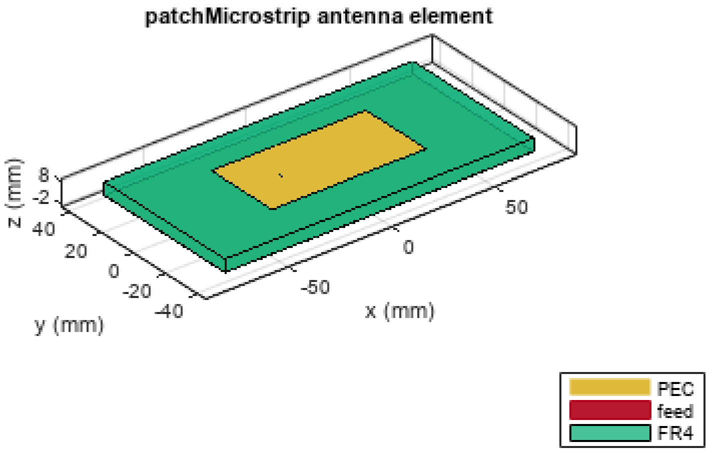
```
show(ant)
```



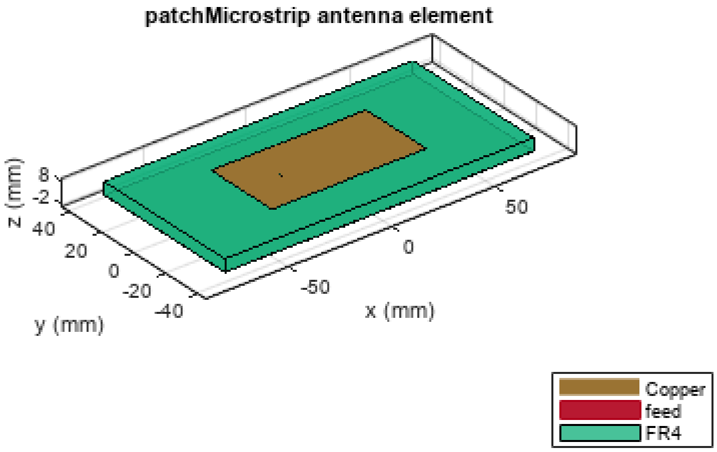
Compare Impedance Values of Microstrip Patch Antennas with Different Metal Patches

Create and visualize the patch microstrip antennas of PEC, copper, silver, and aluminium metal patches.

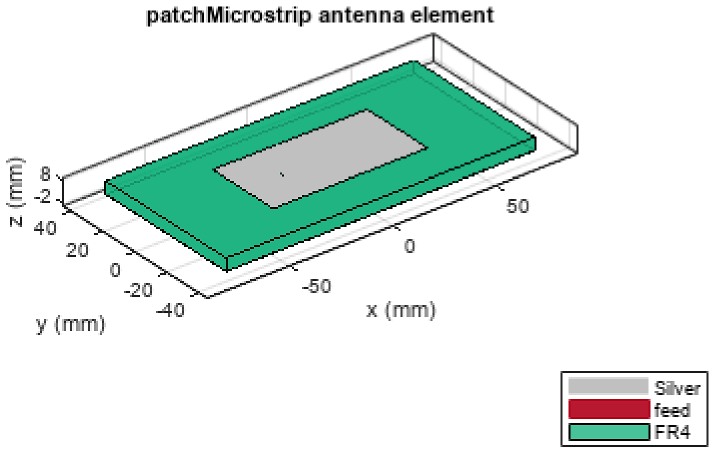
```
ant_patchPEC = patchMicrostrip('Substrate',dielectric('FR4'),'Conductor',metal('PEC'));
show(ant_patchPEC)
```



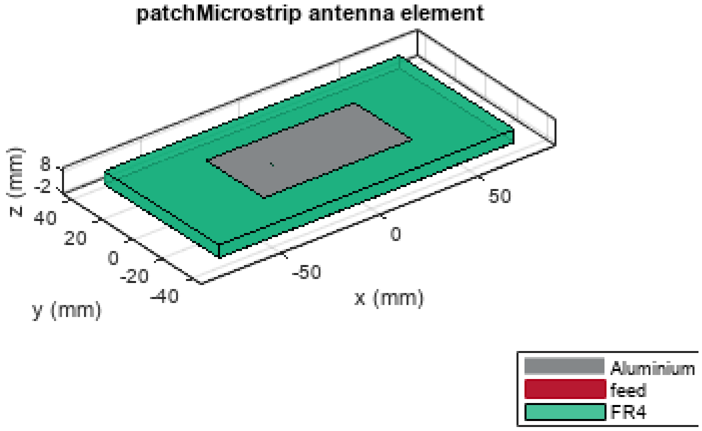
```
ant_patchCopper = patchMicrostrip('Substrate',dielectric('FR4'),'Conductor',metal('Copper'));
show(ant_patchCopper)
```



```
ant_patchSilver = patchMicrostrip('Substrate',dielectric('FR4'),'Conductor',metal('Silver'));
show(ant_patchSilver)
```



```
ant_patchAluminium = patchMicrostrip('Substrate',dielectric('FR4'),'Conductor',metal('Aluminium')
show(ant_patchAluminium)
```



Compare the impedance values at a frequency of 1.2 GHz.

```
Z_patchPEC = impedance(ant_patchPEC,1.2e09)
```

```
Z_patchPEC = 1.4798 + 8.4297i
```

```
Z_patchCopper = impedance(ant_patchCopper,1.2e09)
Z_patchCopper = 1.5195 + 8.4720i
```

```
Z_patchSilver = impedance(ant_patchSilver,1.2e09)
```

```
Z_patchSilver = 1.8449 + 8.4447i
```

```
Z_patchAluminium = impedance(ant_patchAluminium,1.2e09)
```

```
Z_patchAluminium = 1.5297 + 8.4829i
```

Input Arguments

material — Material from metal catalog

'PEC' (default) | character vector

Material from the dielectric catalog, specified as a metal from the MetalCatalog. The default material is PEC, which has infinite conductivity and zero thickness.

Example: 'Iron'

Data Types: char

Name-Value Pair Arguments

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

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

Example: 'Name','Iron'

Name — Name of metal material

character vector

Name of the metal material you want to use as a conductor, specified as the comma-separated pair consisting of 'Name' and a character vector.

Example: 'Name','Tungsten'

Data Types: char

Conductivity — Conductivity of metal material

Inf (default) | scalar

Conductivity of the metal material, specified as a scalar in Siemens per meters(S/m). If you set 'Conductivity' to 'Inf', you must set 'Thickness' to '0'.

Example: 'Conductivity',4.8e06

Data Types: double

Note In Antenna Toolbox, the minimum value of conductivity value is 1e05 S/m.

Thickness — Thickness of metal

0 (default) | scalar

Thickness of the metal material along the default *z*-axis, specified as a scalar in meters.

Example: `'Thickness',0.26e-6`

Data Types: `double`

Note In Antenna Toolbox, the upper limit to thickness value is 1e-03 m.

Output Arguments

`m` — Conductor metal
metal object

Conductor metal, returned as a metal object. You can create an antenna using the `metal` object.

Version History

Introduced in R2021a

See Also

`MetalCatalog` | `dielectric` | `DielectricCatalog`

Topics
"Antenna Toolbox Limitations"

MetalCatalog

Catalog of metals

Syntax

```
mc = MetalCatalog
```

Description

`mc = MetalCatalog` creates an object handle for the metal catalog.

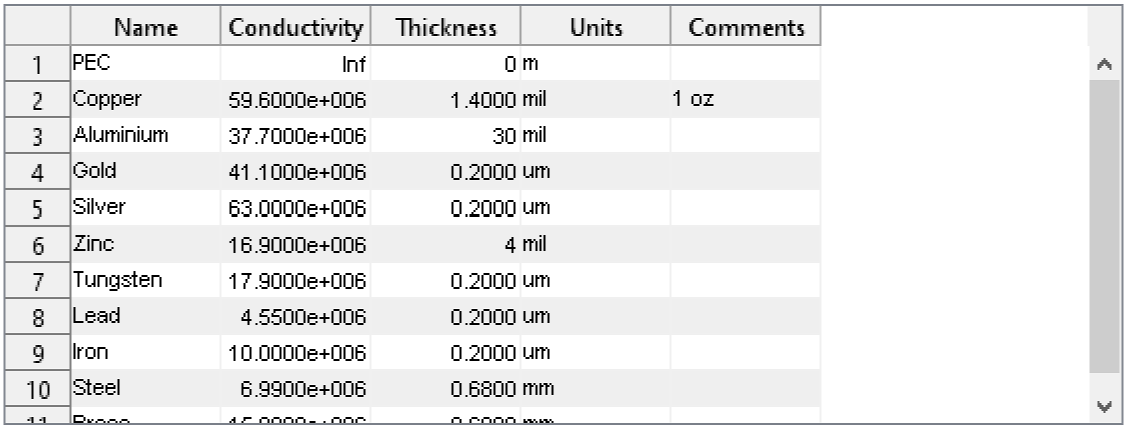
- To open the metal catalog, use `open(mc)`
- To see the properties of a metal from the metal catalog, use `s = find(mc,name)`.

Examples

Use Metal Catalog to Design Corrugated Horn Antenna

Open the metal catalog.

```
mc = MetalCatalog;
open(mc)
```



	Name	Conductivity	Thickness	Units	Comments
1	PEC	Inf	0	m	
2	Copper	59.6000e+006	1.4000	mil	1 oz
3	Aluminium	37.7000e+006	30	mil	
4	Gold	41.1000e+006	0.2000	um	
5	Silver	63.0000e+006	0.2000	um	
6	Zinc	16.9000e+006	4	mil	
7	Tungsten	17.9000e+006	0.2000	um	
8	Lead	4.5500e+006	0.2000	um	
9	Iron	10.0000e+006	0.2000	um	
10	Steel	6.9900e+006	0.6800	mm	
11	Brass	15.9000e+006	0.6800	mm	

List the properties of the metal material `Brass`.

```
s = find(mc,'Brass')
```

```
s = struct with fields:
            Name: 'Brass'
    Conductivity: 15900000
       Thickness: 0.6800
           Units: 'mm'
        Comments: ''
```

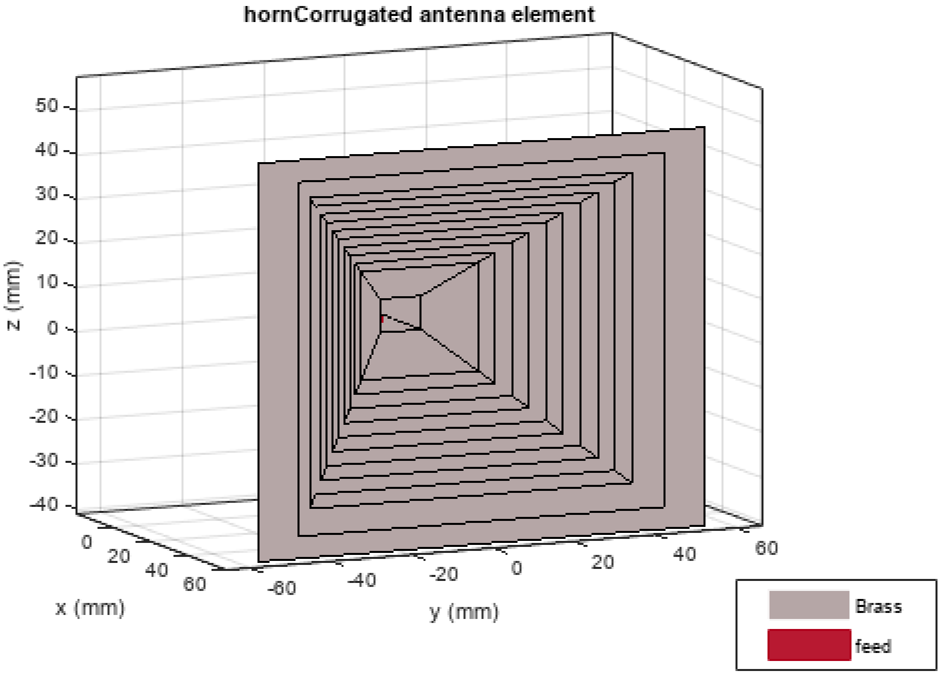
Use the material `Brass` as a metal in a corrugated horn antenna.

```
m = metal('Brass');
ant = hornCorrugated('Conductor',m)
```

```
ant = 
  hornCorrugated with properties:

               FlareLength: 0.0428
                FlareWidth: 0.0900
               FlareHeight: 0.0800
                    Length: 0.0229
                     Width: 0.0102
                    Height: 0.0075
                 FeedWidth: 8.0000e-05
                FeedHeight: 0.0037
                FeedOffset: [-0.0020 0]
    FirstCorrugateDistance: 0.0160
            CorrugateDepth: [0.0050 0.0100]
            CorrugateWidth: 0.0030
                     Pitch: 0.0060
                 Conductor: [1x1 metal]
                      Tilt: 0
                  TiltAxis: [1 0 0]
                      Load: [1x1 lumpedElement]
```

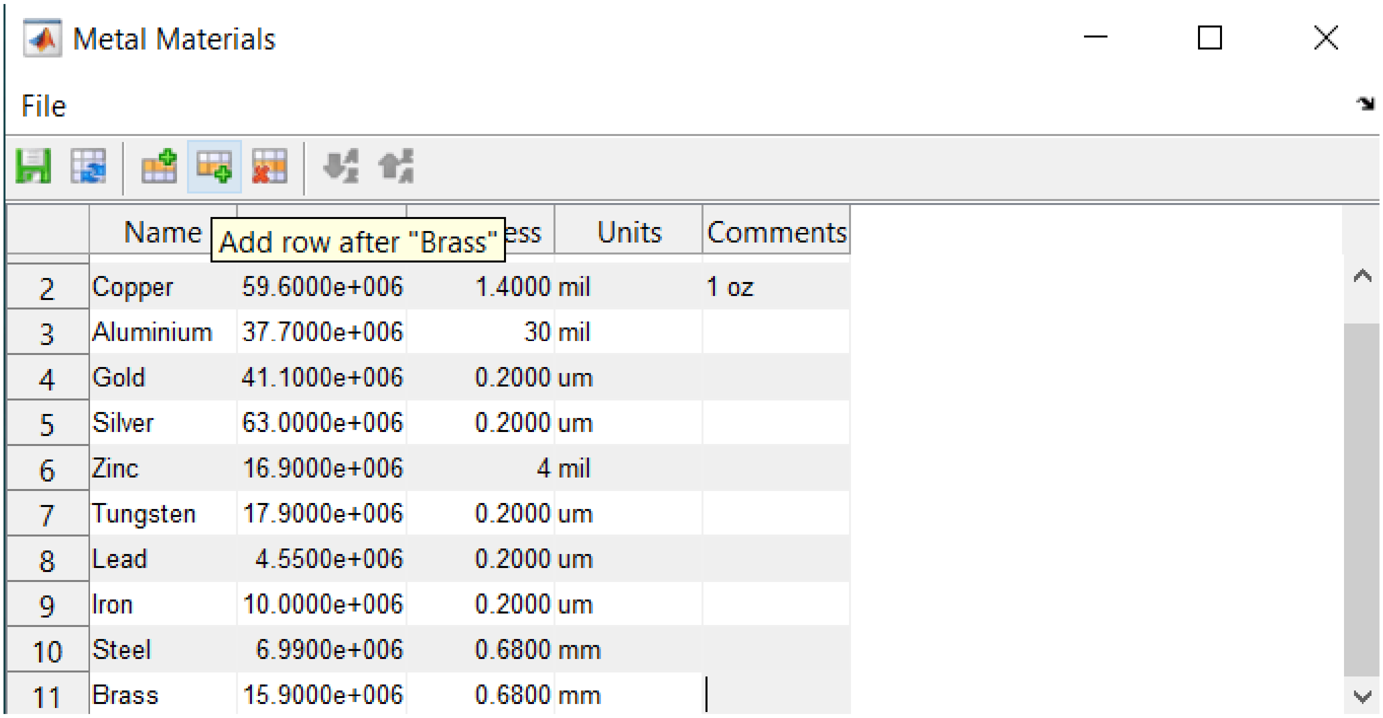
View the antenna using show function.

```
figure;
show(ant)
```



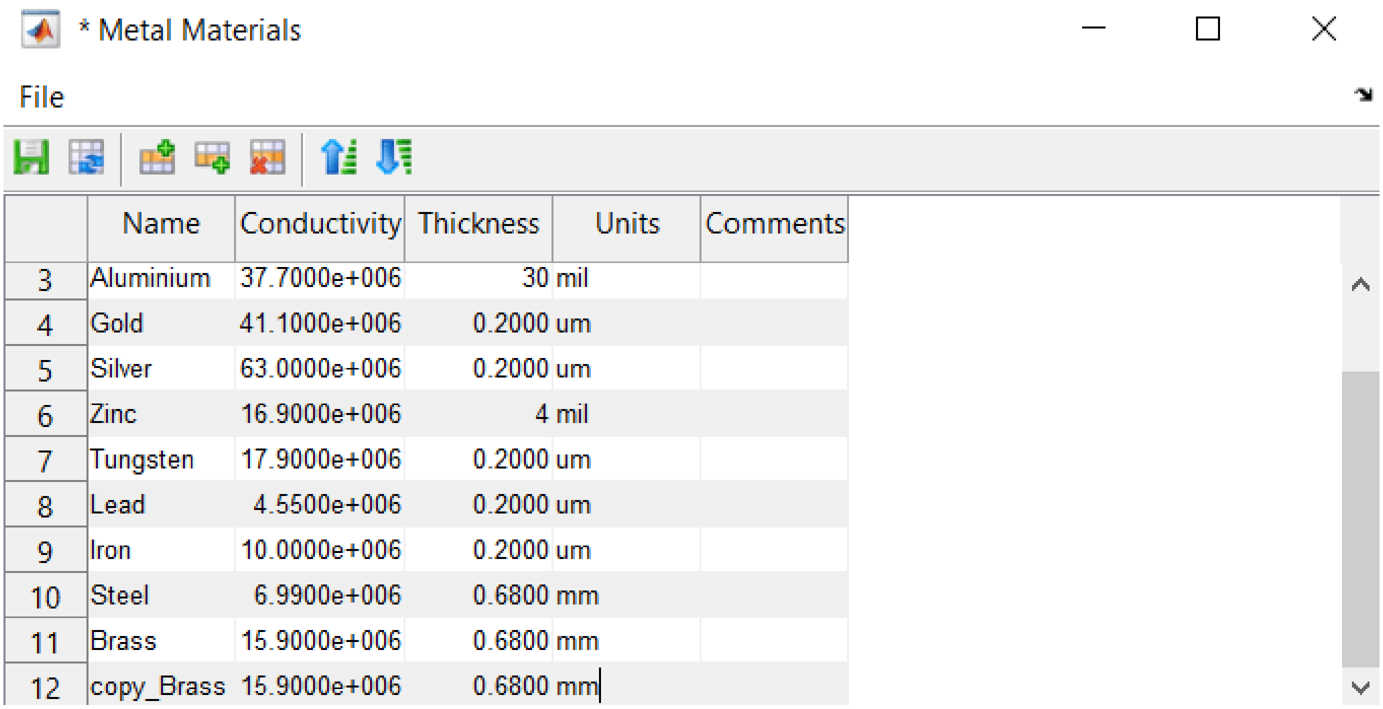
Addition of Custom `metal` in Metal Catalog

Open the metal catalog using `MetalCatalog` function. To add a new material to the metal catalog, click on the row addition icon.



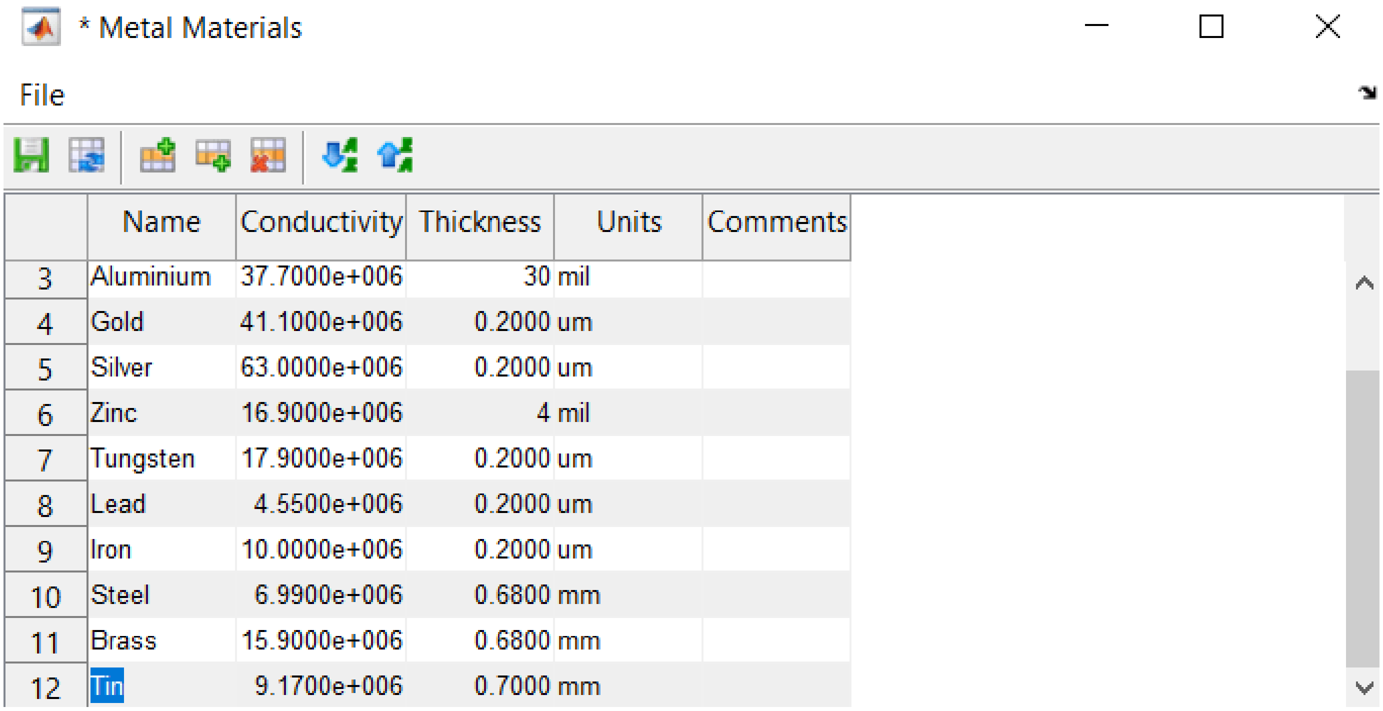
	Name		ess	Units	Comments
2	Copper	59.6000e+006	1.4000	mil	1 oz
3	Aluminium	37.7000e+006	30	mil	
4	Gold	41.1000e+006	0.2000	um	
5	Silver	63.0000e+006	0.2000	um	
6	Zinc	16.9000e+006	4	mil	
7	Tungsten	17.9000e+006	0.2000	um	
8	Lead	4.5500e+006	0.2000	um	
9	Iron	10.0000e+006	0.2000	um	
10	Steel	6.9900e+006	0.6800	mm	
11	Brass	15.9000e+006	0.6800	mm	

A duplicate record of the metal material appears below the selected row.



	Name	Conductivity	Thickness	Units	Comments
3	Aluminium	37.7000e+006	30	mil	
4	Gold	41.1000e+006	0.2000	um	
5	Silver	63.0000e+006	0.2000	um	
6	Zinc	16.9000e+006	4	mil	
7	Tungsten	17.9000e+006	0.2000	um	
8	Lead	4.5500e+006	0.2000	um	
9	Iron	10.0000e+006	0.2000	um	
10	Steel	6.9900e+006	0.6800	mm	
11	Brass	15.9000e+006	0.6800	mm	
12	copy_Brass	15.9000e+006	0.6800	mm	

You can change the record by setting a desired name, conductivity and thickness of the material to customize.



	Name	Conductivity	Thickness	Units	Comments
3	Aluminium	37.7000e+006	30	mil	
4	Gold	41.1000e+006	0.2000	um	
5	Silver	63.0000e+006	0.2000	um	
6	Zinc	16.9000e+006	4	mil	
7	Tungsten	17.9000e+006	0.2000	um	
8	Lead	4.5500e+006	0.2000	um	
9	Iron	10.0000e+006	0.2000	um	
10	Steel	6.9900e+006	0.6800	mm	
11	Brass	15.9000e+006	0.6800	mm	
12	Tin	9.1700e+006	0.7000	mm	

You can access the new added metal material by using `metal` object.

Input Arguments

`name` — Name of metal

`'PEC'` (default) | character vector

Name of the metal from the metal catalog, specified as a character vector.

Example: `'Copper'`

Data Types: `char`

`mc` — Metal catalog

`metal` object

Metal catalog, specified as an object.

Data Types: `char`

Output Arguments

`mc` — Metal catalog

object

Metal catalog, returned as an object.

`s` — Parameters of metal

structure

Parameters of the specified metal from the metal catalog, returned as a structure.

Version History

Introduced in R2021a

See Also

`metal` | `dielectric` | `DielectricCatalog`

efficiency

Radiation efficiency of antenna

Syntax

```
E = efficiency(antenna,frequency)
efficiency(antenna,frequency)
```

Description

`E = efficiency(antenna,frequency)` returns the radiation efficiency of the antenna over the specified frequency.

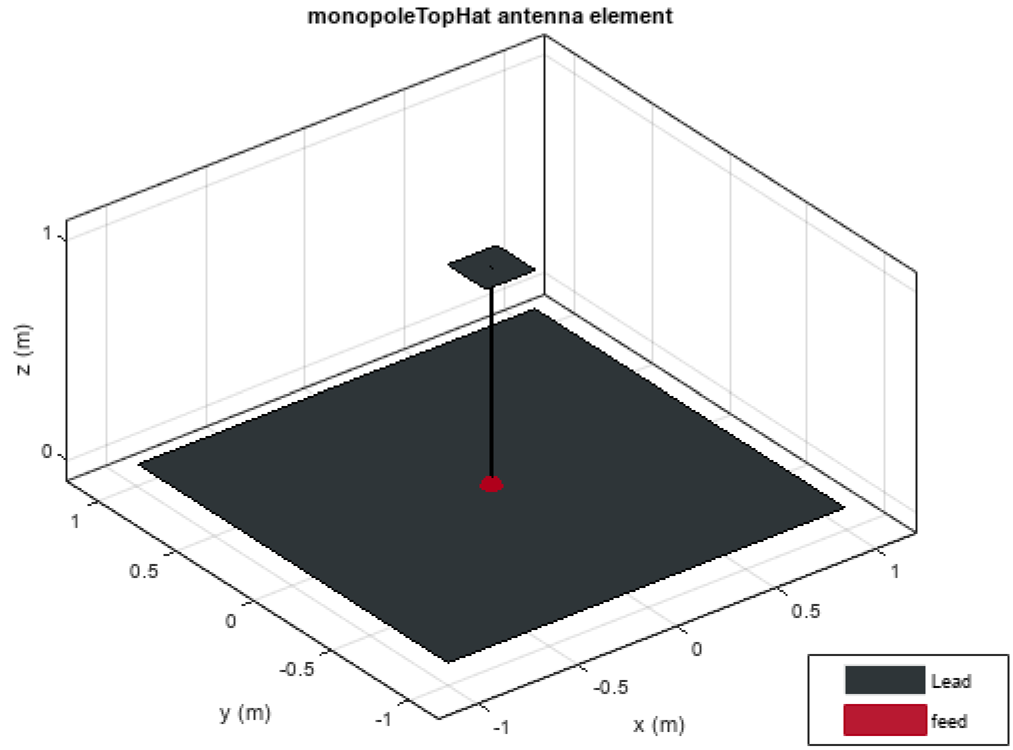
`efficiency(antenna,frequency)` plots the radiation efficiency of the antenna over the specified frequency.

Examples

Radiation Efficiency of Top-Hat Monopole Antenna

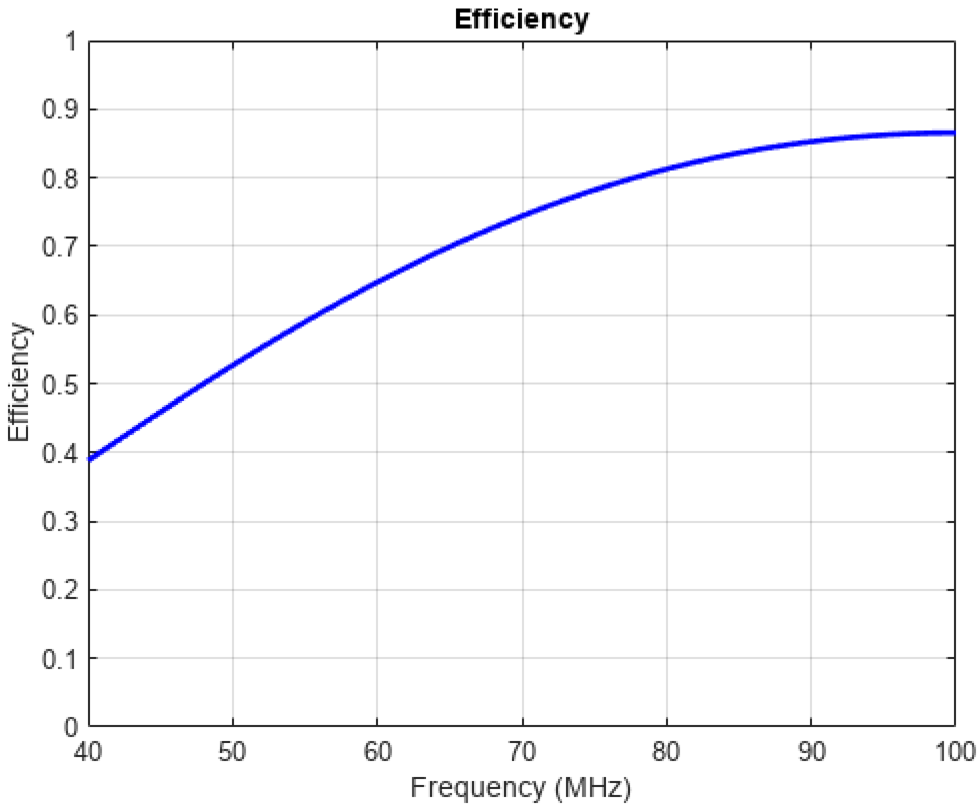
Create a top-hat monopole antenna.

```
m = metal('Lead');
ant = monopoleTopHat('Conductor',m);
show(ant)
```



Plot the radiation efficiency of the antenna over a frequency range of 40-100 MHz.

```
efficiency(ant,linspace(40e6,100e6,41))
```



Compare Radiation Efficiency of Antennas with Different Metal Patches

Create and visualize patch microstrip antennas with PEC, copper, silver, and aluminium metal patches.

```
ant_patchPEC = patchMicrostrip('Substrate',dielectric('Air'),'Conductor',metal('PEC'));
ant_patchCopper = patchMicrostrip('Substrate',dielectric('FR4'),'Conductor',metal('Copper'));
ant_patchSilver = patchMicrostrip('Substrate',dielectric('FR4'),'Conductor',metal('Silver'));
ant_patchAluminium = patchMicrostrip('Substrate',dielectric('FR4'),'Conductor',metal('Aluminium'
```

Compare the radiation efficiency of these antennas at the frequency of 1 GHz.

```
Eff_patchPEC = efficiency(ant_patchPEC,1e09)
```

```
Eff_patchPEC = 1
```

```
Eff_patchCopper = efficiency(ant_patchCopper,1e09)
```

```
Eff_patchCopper = 0.4922
```

```
Eff_patchSilver = efficiency(ant_patchSilver,1e09)
```

```
Eff_patchSilver = 0.4353
```

```
Eff_patchAluminium = efficiency(ant_patchAluminium,1e09)
Eff_patchAluminium = 0.4903
```

Input Arguments

antenna — Input antenna
antenna object

Input antenna, specified as an antenna object. The antenna object that you specify should be a single-feed structure.

frequency — Frequency range used to calculate radiation efficiency
scalar | vector

Frequency range to calculate the radiation efficiency, specified as a scalar in hertz or a vector with each element unit in Hz.

Example: 50e6:1e6:100e6

Data Types: double

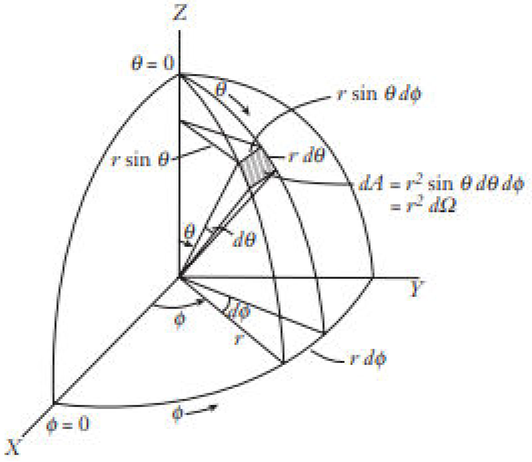
Output Arguments

E — Radiation efficiency of antenna
scalar | vector

Radiation efficiency of the antenna, returned as a scalar or a vector with values in the range [0,1].

More About

Radiation Efficiency



The time-average power density radiated by an antenna is:

$$p(\theta, \varphi, r) = \frac{1}{2} Re.[E \times H^{*}]$$

where, r = radius of the sphere,

θ, φ = spherical coordinate angles,

E, H = electric and magnetic field intensity respectively.

The total power radiated by antenna is:

$$P_{rad} = \oint p(\theta,\varphi,r)\,dS = \int_{\theta=0}^{\pi}\int_{\varphi=0}^{2\pi} p(\theta,\varphi,r) r^2\, sin\theta\, d\theta\, d\varphi$$
$$= \int_{\theta=0}^{\pi}\int_{\varphi=0}^{2\pi} U(\theta,\varphi,r)\, d\Omega$$

where, U = radiation intensity (W/ unit solid angle),

dΩ = element of solid angle.

$$\eta_r = \frac{P_{rad}}{P_{in}}:$$

The radiation efficiency is given as:

where, P_{in} = total input power.

Version History

Introduced in R2021a

See Also

`returnLoss` | `sparameters`

array

Create array of PCB stack objects

Syntax

```
pcbArr = array(pcbObj,'linear')
pcbArr = array(pcbObj,'rectangular')
pcbArr = array(pcbObj,'circular')
pcbArr = array( ___ ,Name,Value)
```

Description

`pcbArr = array(pcbObj,'linear')` creates a default linear array of the input PCB stack object `pcbObj`. The default linear array has with two elements and element spacing of 2 meters.

`pcbArr = array(pcbObj,'rectangular')` creates a default rectangular array of the input PCB stack object. The default rectangular array is a 2-by-2 array with row and column spacing of 2 meters.

`pcbArr = array(pcbObj,'circular')` creates a default circular array of the input PCB stack object. The default circular array has six elements with a radius of 1 meter and an offset angle of 0 degrees.

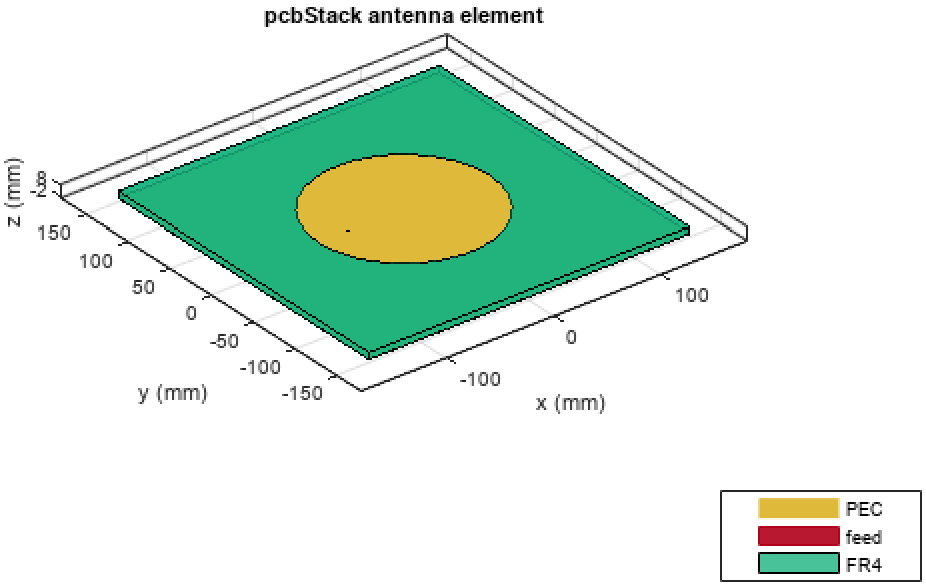
`pcbArr = array( ___ ,Name,Value)` updates the array using one or more name-value pairs. For example, `p = array(pcbObj,'linear','NumElements',5)` creates a linear array of `pcbObj` PCB stack object with five elements.

Examples

Create Linear Array of Circular Patch Microstrip Antenna

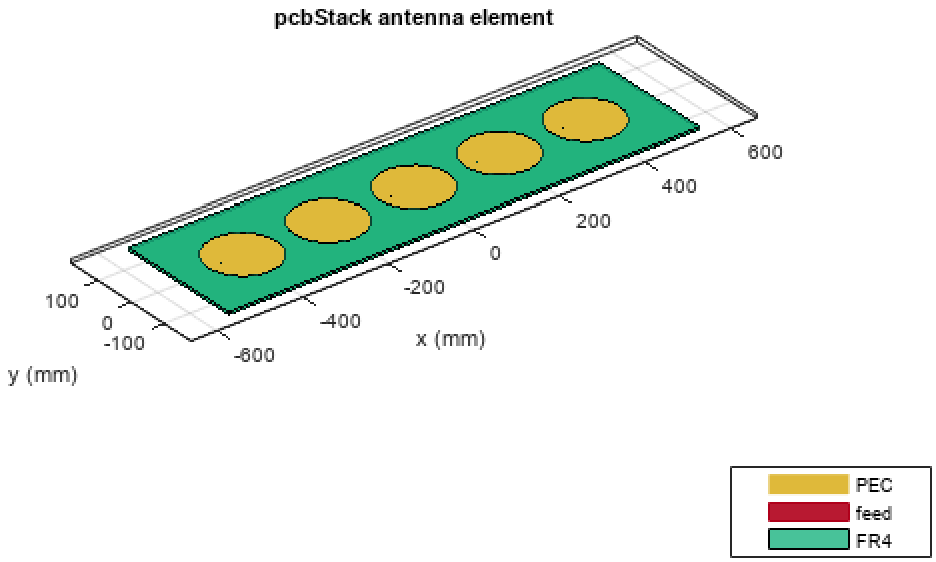
Create a PCB antenna from a circular patch microstrip antenna using the `pcbStack` object.

```
ant = patchMicrostripCircular('Substrate',dielectric('FR4'));
pcbAnt = pcbStack(ant);
show(pcbAnt)
```



Create a linear array of the antenna using the `array` function.

```
pcbArr = array(pcbAnt,'linear','NumElements',5,'ElementSpacing',0.2);
show(pcbArr)
```



Create Rectangular Array of PCB Antenna

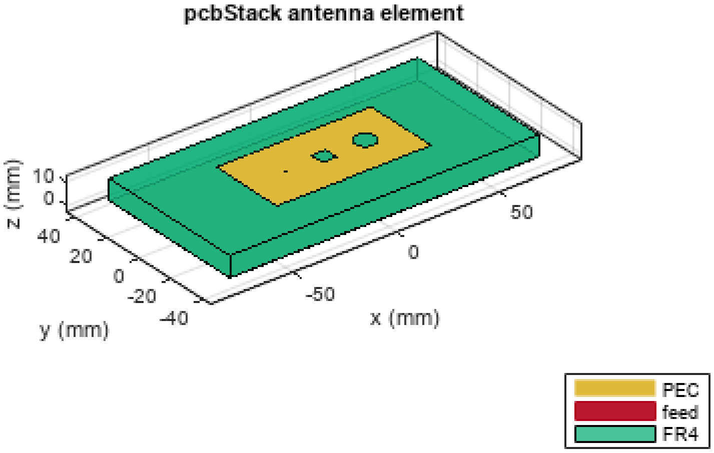
Create a PCB antenna using pcbStack object

```
pcbAnt = pcbStack
```

```
pcbAnt = 
  pcbStack with properties:

              Name: 'MyPCB'
          Revision: 'v1.0'
        BoardShape: [1x1 antenna.Rectangle]
    BoardThickness: 0.0100
            Layers: {[1x1 antenna.Rectangle]  [1x1 antenna.Rectangle]}
     FeedLocations: [-0.0187 0 1 2]
      FeedDiameter: 1.0000e-03
      ViaLocations: []
       ViaDiameter: []
      FeedViaModel: 'strip'
       FeedVoltage: 1
         FeedPhase: 0
         Conductor: [1x1 metal]
              Tilt: 0
          TiltAxis: [1 0 0]
```

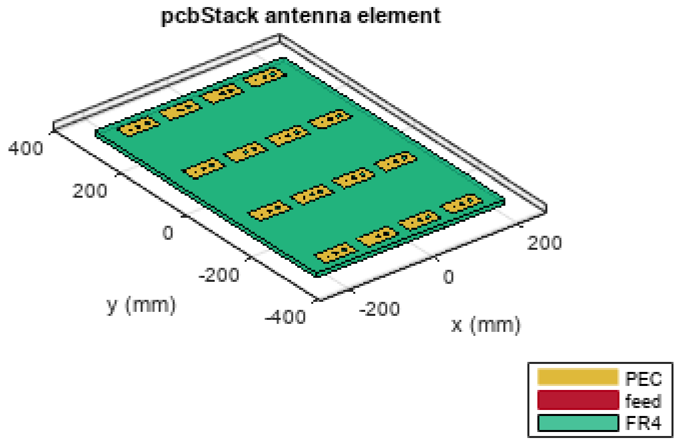
```
              Load: [1x1 lumpedElement]
```

```
pcbAnt.Layers{1} = pcbAnt.Layers{1} - antenna.Rectangle('Length',7e-3,'Width',7e-3);
pcbAnt.Layers{1} = pcbAnt.Layers{1} - antenna.Circle('Radius',5e-3,'Center',[20e-3,0]);
d = dielectric('FR4');
d.Thickness = pcbAnt.BoardThickness;
pcbAnt.Layers{2} = d;
pcbAnt.Layers{3} = antenna.Rectangle('Length',0.15,'Width',0.075);
pcbAnt.FeedLocations(end) = 3;
show(pcbAnt)
```



Create a 4x4 rectangular array of the PCB antenna.

```
pcbArr = array(pcbAnt,'rectangular','Size',[4 4],'ColumnSpacing',0.1,'RowSpacing',0.2);
show(pcbArr)
```



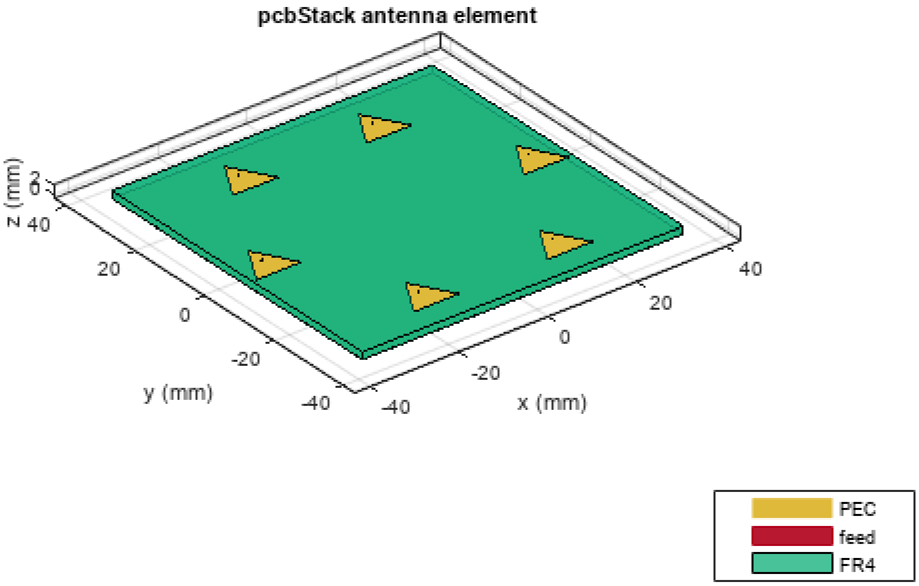
Create Circular Array of Triangular Patch Microstrip Antenna

Create a PCB antenna from a triangular patch microstrip antenna using the `pcbStack` object.

```
ant = patchMicrostripTriangular('Substrate',dielectric('FR4'));
pcbAnt = pcbStack(ant);
```

Create a circular array of the PCB antenna.

```
pcbArr = array(pcbAnt,'circular','Radius',0.03);
show(pcbArr)
```



Input Arguments

pcbObj — PCB antenna

pcbStack object

PCB antenna, specified as a pcbStack object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value pair 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: 'NumElements',4

Linear Array

NumElements — Number of antenna elements in array

2 (default) | positive integer

Number of the antenna elements in the array, specified as a positive integer.

Example: 'NumElements',4

`ElementSpacing` — Spacing between antenna elements

2 (default) | positive scalar | positive vector

Spacing between the antenna elements, specified as a positive scalar or positive vector in meters. By default, the dipole elements are spaced 2 meters apart. Use scalar for uniform and vector for non uniform spacing between the antenna elements.

Example: `'ElementSpacing',3`

Data Types: `double`

Rectangular Array

`Size` — Number of antenna elements in row and column of array

`[2 2]` (default) | two-element vector

Number of the antenna elements in the row and column of the array, specified as a two-element vector.

Example: `'Size',[4 4]`

`RowSpacing` — Row spacing between two antenna elements

2 (default) | positive scalar | positive vector

Row spacing between the two antenna elements, specified as a scalar or vector in meters. By default, the antenna elements are spaced 2 meters apart. Use scalar for uniform and vector for non uniform spacing between the antenna elements.

Example: `'RowSpacing',0.1`

Data Types: `double`

`ColumnSpacing` — Column spacing between two antenna elements

2 (default) | positive scalar | positive vector

Column spacing between the two antenna elements, specified as a positive scalar or positive vector in meters. By default, the antenna elements are spaced 2 meters apart. Use scalar for uniform and vector for non uniform spacing between the antenna elements.

Example: `'ColumnSpacing',0.1`

Data Types: `double`

Circular Array

`NumElements` — Number of elements in array

6 (default) | positive integer

Number of elements in the array, specified as a positive integer. The elements in the array are arranged along the X-axis.

Example: `'NumElements',4`

Data Types: `double`

`Radius` — Radius of array

1 (default) | positive scalar

Radius of the array, specified as a positive scalar in meters.

Example: `'Radius',0.4`

Data Types: `double`

`AngleOffset` — Offset angle for first element in array
`0` (default) | real scalar

Offset angle for the first element in the array, specified as a real scalar in degrees.

Example: `'AngleOffset',8`

Data Types: `double`

Output Arguments

`pcbArr` — Array of PCB antenna elements
`pcbStack` object

Array of the PCB antenna elements, returned as a `pcbStack` object.

Version History

Introduced in R2021a

See Also

`pcbStack` | `linearArray` | `rectangularArray` | `circularArray`

sparameters

Calculate S-parameter for antenna and antenna array objects

Syntax

```
sobj = sparameters(antenna,freq)
sobj = sparameters(antenna,freq,Z0)

sobj = sparameters(array,freq)
sobj = sparameters(array,freq,Z0)

sobj = sparameters(filename)

sobj = sparameters(data,freq)
sobj = sparameters(data,freq,Z0)

sobj = sparameters(netparamobj)
sobj = sparameters(netparamobj,Z0)
```

Description

`sobj = sparameters(antenna,freq)` calculates the complex s-parameters for an `antenna` object over specified frequency values.

`sobj = sparameters(antenna,freq,Z0)` calculates the complex s-parameters for an `antenna` object over specified frequency values and for a given reference impedance, `Z0`.

`sobj = sparameters(array,freq)` calculates the complex s-parameters for an `array` object over specified frequency values .

`sobj = sparameters(array,freq,Z0)` calculates the complex s-parameters for an `array` object over specified frequency values and for a given reference impedance, `Z0`.

`sobj = sparameters(filename)` creates an S-parameter object `sobj` by importing data from the Touchstone file specified by `filename`.

`sobj = sparameters(data,freq)` creates an S-parameter object from the S-parameter data, `data`, and frequencies, `freq`.

`sobj = sparameters(data,freq,Z0)` creates an S-parameter object from the S-parameter data, `data`, and frequencies, `freq`, with a given reference impedance `Z0`.

`sobj = sparameters(netparamobj)` converts the network parameter object, `netparamobj`, to S-parameter object with the default reference impedance.

`sobj = sparameters(netparamobj,Z0)` converts the network parameter object, `netparamobj`, to S-parameter object with a given reference impedance, `Z0`.

Examples

Calculate S-Parameter Matrix For Antenna

Calculate the complex s-parameters for a default dipole at 70MHz frequency.

```
 h = dipole
```

```
h = 
  dipole with properties:

        Length: 2
         Width: 0.1000
    FeedOffset: 0
     Conductor: [1x1 metal]
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
 sparameters (h, 70e6)
```

```
ans = 
  sparameters: S-parameters object

       NumPorts: 1
    Frequencies: 70000000
     Parameters: 0.1880 + 0.0011i
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

Calculate S-parameter Matrix For Array

Calculate the complex s-parameters for a default rectangular array at 70MHz frequency.

```
h = rectangularArray;
sparameters(h,70e6)
```

```
ans = 
  sparameters: S-parameters object

       NumPorts: 4
    Frequencies: 70000000
     Parameters: [4x4 double]
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

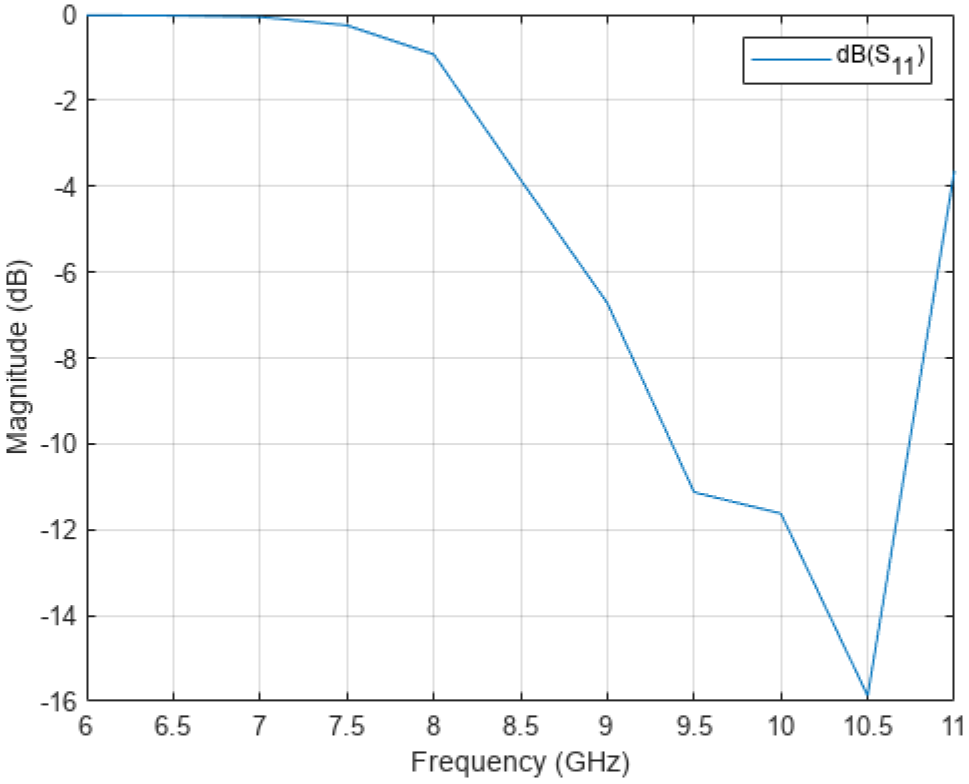
Extract and Plot S-Parameters data from Touchstone File

Extract S-parameters from `corrugatedconicalhorn.s2p` touchstone file .

```
sobj = sparameters('corrugatedconicalhorn.s2p');
display(sobj)

sobj = 
  sparameters: S-parameters object

       NumPorts: 1
    Frequencies: [11x1 double]
     Parameters: [1x1x11 double]
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

Plot the S-parameters data using `rfplot` function.

```
rfplot(sobj)
```



Input Arguments

antenna — Antenna object
scalar handle

Antenna object, specified as a scalar handle.

array — Array object
scalar handle

Array object, specified as a scalar handle.

freq — S-parameter frequencies
vector of positive real numbers

S-parameter frequencies, specified as a vector of positive real numbers, sorted from smallest to largest.

Z0 — Reference impedance
50 (default) | positive real scalar

Reference impedance in ohms, specified as a positive real scalar. You cannot specify `Z0` if you are importing data from a file. The argument Z0 is optional and is stored in the `Impedance` property.

data — S-parameter data
array of complex numbers

S-parameter data, specified as an array of complex numbers, of size *N*-by-*N*-by-*K* where K represents number of frequency points.

netparamobj — Network parameter object
network parameter object

Network parameter object. The network parameter objects are of the type: `sparameters`, `yparameters`, `zparameters`, `gparameters`, `hparameters`, `abcdparameters`, and `tparameters`.

Example: `S1 = sparameters(Y1,100)` . `Y1` is a parameter object. This example converts Y-parameters to S-parameters at `100` ohms.

filename — Touchstone data file
character vector | string scalar

Touchstone data file, specified as a character vector, that contains network parameter data. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `sobj = sparameters('defaultbandpass.s2p');`

Output Arguments

sobj — S-parameter data
S-parameter object

S-parameter data, returned as an object. `disp(sobj)` returns the properties of the object:

- `NumPorts` — Number of ports, specified as an integer. The function calculates this value automatically when you create the object.
- `Frequencies` — S-parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — S-parameter data, specified as an *N*-by-*N*-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.

- `Impedance` — Reference impedance in ohms, specified as a positive real scalar. The function sets this property from the `filename` or `Z0` input arguments. If no reference impedance is provided, the function uses a default value of `50`.

Version History

Introduced in R2015a

See Also

`sparameters` | `vswr` | `returnLoss` | `impedance` | `feedCurrent` | `efficiency` | `correlation`

convergence

Calculate and plot convergence of FMM solver

Syntax

```
convergence(hsolver)
```

Description

`convergence(hsolver)` calculates and plots convergence of the fast multipole method (FMM) solver defined in `hsolver`. The function calculates the convergence over the iterations and relative residual specified in the `hsolver` object.

Examples

Calculate and Plot Convergence of FMM Solver

Design a default parabolic reflector antenna.

```
m = reflectorParabolic;
```

Set the solver type of the parabolic reflector antenna to FMM.

```
m.SolverType = 'FMM';
```

Calculate the impedance of the parabolic reflector antenna at 10 GHz.

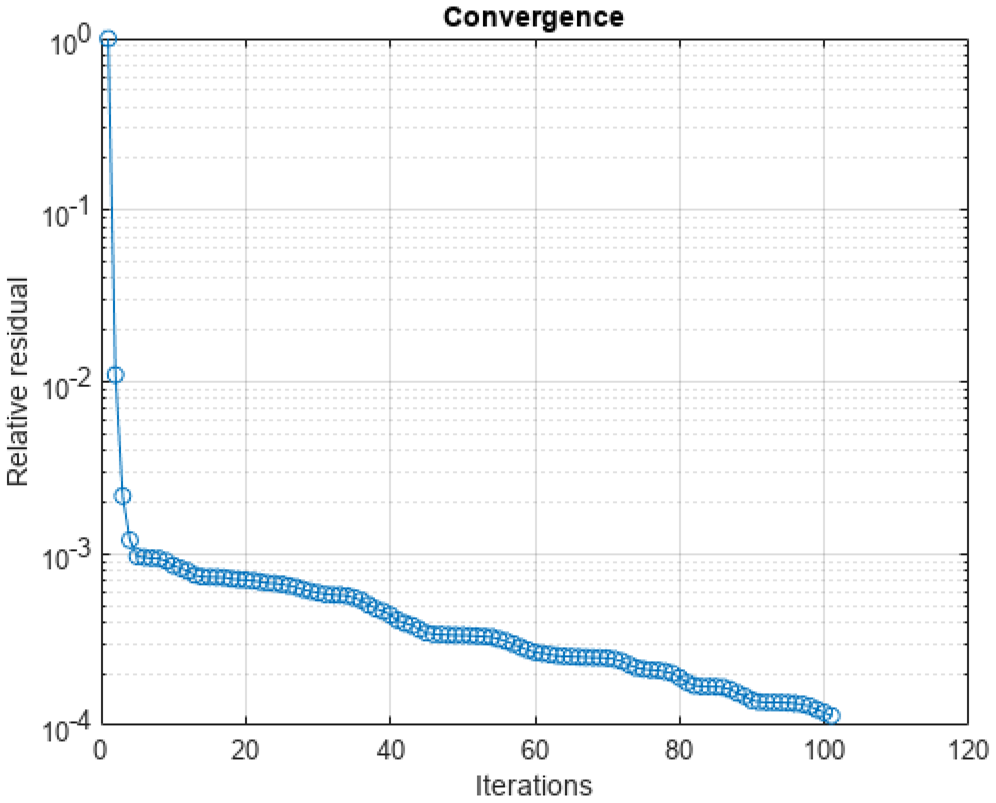
```
Z = impedance(m,10e9);
```

Access the FMM solver and set the relative residual to 1e-3.

```
s = solver(m);
s.RelativeResidual = 1e-3;
```

Calculate and plot the convergence of the FMM solver for the parabolic reflector antenna.

```
convergence(s)
```



Input Arguments

`hsolver` — FMM solver
`solver` object

Fast multipole method (FMM) solver, specified as a `solver` object.

Version History

Introduced in R2021b

See Also

`solver`

Topics
"Analysis of Electrically Large Structures Using Hybrid MoM and FMM"

doa

Direction of arrival of signal

Syntax

```
[phiArrival,thetaArrival] = doa(obj,freq)
```

Description

`[phiArrival,thetaArrival] = doa(obj,freq)` calculates the direction of arrival of a signal when the transmission source is either a single-feed antenna defined using a `conformalArray` object or an incident plane-wave defined using a `planeWaveExcitation` object.

Examples

Calculate Direction of Arrival Using Conformal Array Object

Visualize a transmitter-receiver system as a conformal array. Use a `conformalArray` object with a dipole antenna as the transmitter element and an array of dipoles as the receiver element. Specify an operating fequency of 70 MHz.

```
f = 70e6;
lambda = 3e8/f;
```

Specify the location coordinates of the transmitter and the receiver.

```
txCenterOffset = [200,50,370];
rxCenterOffset = [0,0,0];
```

Create the first and second element of the conformal array.

```
elem1 = design(dipole,f);
elem1.Tilt = 90;
elem2 = linearArray('Element',elem1,'NumElements',4,'ElementSpacing',0.4*lambda);
```

Create and view the conformal array.

```
c = conformalArray;
c.Element = {elem1 elem2};
c.ElementPosition = [txCenterOffset;rxCenterOffset];
```

Calculate the direction of arrival.

```
[phiArrival,thetaArrival] = doa(c,f)
```

```
phiArrival = 14.0362
```

```
thetaArrival = 27.9882
```

Calculate Direction of Arrival Using Plane-Wave Excitation Object

Calculate the direction of arrival of a signal using the plane-wave excitation object. Use a linear array of dipoles operating at 2.4 GHz as the receiver element and set the azimuth and elevation reference angles.

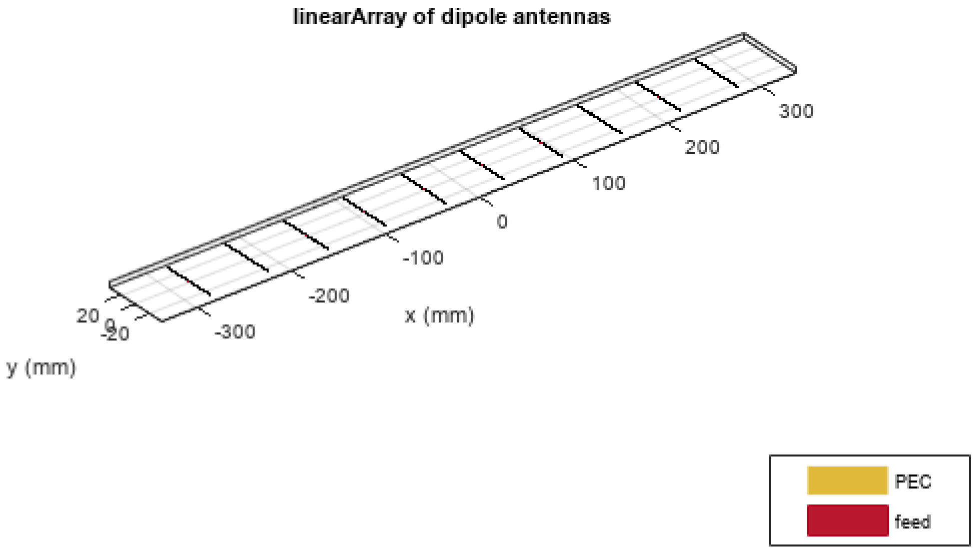
```
azimuthRef = 10;
elevationRef = 30;
f = 2.4e9;
lambda = 3e8/f;
```

Design a dipole antenna using the specified frequency.

```
d = design(dipole,f);
d.Tilt = 90;
```

Create and view a linear array.

```
ant = linearArray('Element',d,'NumElements',10,'ElementSpacing',0.5*lambda);
show(ant)
```



Create a plane-wave excitation object using the dipole array.

```
dir = [sind(elevationRef)*cosd(azimuthRef) sind(elevationRef)*sind(azimuthRef) -cosd(elevationRe
pol = [cosd(elevationRef)*cosd(azimuthRef) cosd(elevationRef)*sind(azimuthRef) sind(elevationRef
pw = planeWaveExcitation('Element',ant,'Direction',dir,'Polarization',pol);
```

Calculate the direction of arrival.

```
[phiArrival,thetaArrival] = doa(pw,f)
```

```
phiArrival = 10

thetaArrival = 29.8870
```

Input Arguments

obj — Transmitter-receiver system

conformalArray object | planeWaveExcitation object

Transmitter-receiver system, specified as a conformalArray object or a planeWaveExcitation object.

To define the transmitter-receiver system as a conformal array of two elements, specify this input as a conformalArray object. The first element of the conformal array is a single-feed antenna, and is the transmission signal source. The second element of the conformal array is the receiver array.

To define the transmitter-receiver system as a plane-wave excitation, specify this input as a planeWaveExcitation object. The plane-wave defined using the Direction and Polarization properties of the planeWaveExcitation object is the incident signal. The linear or rectangular array defined using the Element property of the planeWaveExcitation object is the receiver.

Note The doa function accepts only linear or rectangular array configurations for the receiver array of the conformalArray and planeWaveExcitation objects. The number of elements along each axis of the receiver array must be even, and the center of the receiver array must be at the origin.

freq — Frequency of incident signal

nonnegative scalar | vector with nonnegative entries

Frequency of the incident signal in Hz. to use for the direction of arrival calculations, specified as a nonnegative scalar or a vector with nonnegative entries.

Example: f = 70e6

Data Types: double

Output Arguments

phiArrival — Azimuth angle of arrival of signal

scalar | vector

Azimuth angle of arrival of the signal in degrees, returned as a scalar or vector. The function calculates this output with reference to the center of the receiver array.

Data Types: double

thetaArrival — Elevation angle of arrival of signal

scalar | vector

Elevation angle of arrival of the signal in degrees, returned as a scalar or vector. The function calculates this output with reference to the center of the receiver array.

Data Types: `double`

Algorithms

To estimate the direction of arrival, the `doa` function uses the fundamentals of RF propagation from the transmission to the reception of the signal. The phase of a far-field received signal $E_R(r)$ at an individual antenna is:

$$E_R(r)\,\alpha\, e^{j\beta r} \quad (1)$$

where β is the propagation constant of the received signal and r is the distance between the transmit and receive elements.

For a given transmit antenna, the position of the antenna element is [x_{tx}, y_{tx}, z_{tx}], where

$$x_{tx} = \rho\cos(\varphi)\sin(\theta) \quad (2)$$

$$y_{tx} = \rho\sin(\varphi)\sin(\theta) \quad (3)$$

$$z_{tx} = \rho\cos(\theta) \quad (4)$$

where ρ is the distance between the transmit element and the geometrical midpoint of the receive antenna array. θ is the elevation angle or the angular position of the transmit element from the z-axis and φ is the azimuth angle that lies in the xy-plane.

Assume that:

- The receiver array is a homogenous linear or rectangular array with identical elements.
- The separation between two adjacent elements of the receiver array is uniform. For a linear array, the element spacing is d_{space}. For a rectangular array, the row spacing is d_{row} and the column spacing is d_{col}.
- The receiver array has an equal number of elements along each axis. The number of elements along each axis of the array is even. For a linear array, the number of elements is N_{elem}. For a rectangular array, the number of elements in a row is N_{row} and the number of elements in a column is N_{col}.

For a uniform linear array (ULA) where the receive array elements lie along the x-axis, the position of the n^{th} elements is

$$x_{pos_n} = \frac{d_{space}}{2}(N_{elem} + 1 - 2n) \quad (5)$$

where N_{elem} is the number of elements in the ULA.

For the uniform rectangular array (URA), which lies in the xy-plane, the position of the receive array element in the n^{th} row and m^{th} column is [x_{pos_n} y_{pos_n}], where

$$x_{pos_n} = \frac{d_{col}}{2}(N_{elem} + 1 - 2n) \quad (6)$$

$$y_{pos_n} = \frac{d_{row}}{2}(M_{elem} + 1 - 2m) \quad (7)$$

Consider a single-antenna transmitter located at (x_{tx}, y_{tx}, z_{tx}). This equation determines the far-field separation r between the receiver array element and the transmitter for a ULA:

$$r_n = \sqrt{(x_{tx} - x_{pos_n})^2 + (y_{tx})^2 + (z_{tx})^2} \ \ \text{(ULA)} \ \ (8)$$

.

This equation determines the far-field separation r between the transmitter and the element in the m^{th} column and n^{th} row of a URA:

$$r_{m,n} = \sqrt{(x_{tx} - x_{pos_n})^2 + (y_{tx} - y_{pos_n})^2 + (z_{tx})^2} \ \ \text{(URA)} \ \ (9)$$

The function uses these equations to calculate the phase of the electric field by doing a full-wave electromagnetic simulation and uses this phase to calculate the direction of arrival in terms of the elevation and azimuth angles.

Version History

Introduced in R2022a

See Also

`conformalArray` | `planeWaveExcitation`

Properties

PolarPattern Properties

Control appearance and behavior of polar plot

Description

Polar pattern properties control the appearance and behavior of the polar pattern object. By changing property values, you can modify certain aspects of the polar plot. To change the default properties use:

```
p = polarpattern(____,Name,Value)
```

To view all the properties of the polar pattern object use:

```
details(p)
```

You can also interact with the polar plot to change the properties. For more information, see “Interact with Polar Plot”.

Properties

Antenna Metrics

'AntennaMetrics' — Show antenna metric
0 (default) | 1

Show antenna metrics, specified as a comma-separated pair consisting of 'AntennaMetrics' and 0 or 1. Antenna metric displays main, back, and side lobes of antenna/array pattern passed as input.

Data Types: logical

'Peaks' — Maximum number of peaks to compute for each data set
positive integer | vector of integers

Maximum number of peaks to compute for each data set, specified as a comma-separated pair consisting of 'Peaks' and a positive scalar or vector of integers.

Data Types: double

Angle Properties

'AngleAtTop' — Angle at top of polar plot
90 (default) | scalar in degrees

Angle at the top of the polar plot, specified as a comma-separated pair consisting of 'AngleAtTop' and a scalar in degrees.

Data Types: double

'AngleLim' — Visible polar angle span
[0 360] (default) | 1-by-2 vector of real values

Visible polar angle span, specified as a comma-separated pair consisting of `'AngleLim'` and a 1-by-2 vector of real values.

Data Types: `double`

`'AngleLimVisible'` — Show interactive angle limit cursors
`0` (default) | `1`

Show interactive angle limit cursors, specified as a comma-separated pair consisting of `'AngleLimVisible'` and `0` or `1`.

Data Types: `logical`

`'AngleDirection'` — Direction of increasing angle
`'ccw'` (default) | `'cw'`

Direction of increasing angle, specified as a comma-separated pair consisting of `'AngleDirection'` and `'ccw'` (counterclockwise) or `'cw'` (clockwise).

Data Types: `char`

`'AngleResolution'` — Number of degrees between radial lines
`15` (default) | scalar in degrees

Number of degrees between radial lines depicting angles in the polar plot, specified as a comma-separated pair consisting of `'AngleResolution'` and a scalar in degrees.

Data Types: `double`

`'AngleTickLabelRotation'` — Rotate angle tick labels
`0` (default) | `1`

Rotate angle tick labels, specified as a comma-separated pair consisting of `'AngleTickLabelRotation'` and `0` or `1`.

Data Types: `logical`

`'AngleTickLabelVisible'` — Show angle tick labels
`1` (default) | `0`

Show angle tick labels, specified as a comma-separated pair consisting of `'AngleTickLabelVisible'` and `0` or `1`.

Data Types: `logical`

`'AngleTickLabelFormat'` — Format for angle tick labels
`360` (default) | `180`

Format for angle tick labels, specified as a comma-separated pair consisting of `'AngleTickLabelFormat'` and `360` degrees or `180` degrees.

Data Types: `double`

`'AngleFontSizeMultiplier'` — Scale factor of angle tick font
`1` (default) | numeric value greater than zero

Scale factor of angle tick font, specified as a comma-separated pair consisting of `'AngleFontSizeMultiplier'` and a numeric value greater than zero.

Data Types: `double`

`'Span'` — Show angle span measurement

`0` (default) | `1`

Show angle span measurement, specified as a comma-separated pair consisting of `'Span'` and `0` or `1`.

Data Types: `logical`

`'ZeroAngleLine'` — Highlight radial line at zero degrees

`0` (default) | `1`

Highlight radial line at zero degrees, specified as a comma-separated pair consisting of `'ZeroAngleLine'` and `0` or `1`.

Data Types: `logical`

`'DisconnectAngleGaps'` — Show gaps in line plots with nonuniform angle spacing

`1` (default) | `0`

Show gaps in line plots with nonuniform angle spacing, specified as a comma-separated pair consisting of `'DisconnectAngleGaps'` and `0` or `1`.

Data Types: `logical`

Magnitude Properties

`'MagnitudeAxisAngle'` — Angle of magnitude tick label radial line

`75` (default) | real scalar in degrees

Angle of magnitude tick label radial line, specified as a comma-separated pair consisting of `'MagnitudeAxisAngle'` and real scalar in degrees.

Data Types: `double`

`'MagnitudeTick'` — Magnitude ticks

`[0 0.2 0.4 0.6 0.8]` (default) | 1-by-N vector

Magnitude ticks, specified as a comma-separated pair consisting of `'MagnitudeTick'` and a `1`-by-`N` vector, where `N` is the number of magnitude ticks.

Data Types: `double`

`'MagnitudeTickLabelVisible'` — Show magnitude tick labels

`1` (default) | `0`

Show magnitude tick labels, specified as a comma-separated pair consisting of `'MagnitudeTickLabelVisible'` and `0` or `1`.

Data Types: `logical`

`'MagnitudeLim'` — Minimum and maximum magnitude limits

`[0 1]` (default) | two-element vector of real values

Minimum and maximum magnitude limits, specified as a comma-separated pair consisting of `'MagnitudeLim'` and a two-element vector of real values.

Data Types: `double`

'MagnitudeLimMode' — Determine magnitude dynamic range

'auto' (default) | 'manual'

Determine magnitude dynamic range, specified as a comma-separated pair consisting of 'MagnitudeLimMode' and 'auto' or 'manual'.

Data Types: char

'MagnitudeAxisAngleMode' — Determine angle for magnitude tick labels

'auto' (default) | 'manual'

Determine angle for magnitude tick labels, specified as a comma-separated pair consisting of 'MagnitudeAxisAngleMode' and 'auto' or 'manual'.

Data Types: char

'MagnitudeTickMode' — Determine magnitude tick locations

'auto' (default) | 'manual'

Determine magnitude tick locations, specified as a comma-separated pair consisting of 'MagnitudeTickMode' and 'auto' or 'manual'.

Data Types: char

'MagnitudeUnits' — Magnitude units

'dB' | 'dBLoss'

Magnitude units, specified as a comma-separated pair consisting of 'MagnitudeUnits' and 'db' or 'dBLoss'.

Data Types: char

'MagnitudeFontSizeMultiplier' — Scale factor of magnitude tick font

0.9000 (default) | numeric value greater than zero

Scale factor of magnitude tick font, specified as a comma-separated pair consisting of 'MagnitudeFontSizeMultiplier' and a numeric value greater than zero.

Data Types: double

Miscellaneous Properties

'View' — View section of Smith plot

'full' (default) | 'top' | 'bottom' | 'left' | 'right' | 'top-left' | 'top-right' | 'bottom-left' | 'bottom-right' | string scalar | character vector

View section of Smith plot, specified as a string scalar or character vector. Smith plot can be viewed by setting View property to one of property values in this table.

View Property Effect

Property Value	Effect
'full'	Full Smith plot is viewed.
'top'	Top-half of the Smith plot is viewed.
'bottom'	Bottom-half of the Smith plot is viewed.
'left'	Left-half of the Smith plot is viewed.
'right'	Right-half of the Smith plot is viewed.
'top-left'	Top-left of the Smith plot is viewed.
'top-right'	Top-right of the Smith plot is viewed.
'bottom-left'	Bottom-left of the Smith plot is viewed.
'bottom-right'	Bottom-right of the Smith plot is viewed.

Data Types: `char` | `string`

'NormalizeData' — Normalize each data trace to maximum value

`0` (default) | `1`

Normalize each data trace to maximum value, specified as a comma-separated pair consisting of `'NormalizeData'` and `0` or `1`.

Data Types: `logical`

'ConnectEndpoints' — Connect first and last angles

`0` (default) | `1`

Connect first and last angles, specified as a comma-separated pair consisting of `'ConnectEndpoints'` and `0` or `1`.

Data Types: `logical`

'Style' — Style of polar plot display

`'line'` (default) | `'filled'`

Style of polar plot display, specified as a comma-separated pair consisting of `'Style'` and `'line'` or `'filled'`.

Data Types: `char`

'TemporaryCursor' — Create temporary cursor

`0` (default) | `1`

Create a temporary cursor, specified as a comma-separated pair consisting of `'TemporaryCursor'` and `0` or `1`.

Data Types: `logical`

'ToolTips' — Show tool tips

`1` (default) | `0`

Show tool tips when you hover over a polar plot element, specified as a comma-separated pair consisting of `'ToolTips'` and `0` or `1`.

Data Types: `logical`

'ClipData' — Clip data to outer circle

0 (default) | 1

Clip data to outer circle, specified as a comma-separated pair consisting of 'ClipData' and 0 or 1.

Data Types: logical

'CleanData' — Cleans data

0 (default) | 1

Cleans data removing any Inf or NaN from the data. The property is specified as a comma-separated pair consisting of 'CleanData' and 0 or 1.

Data Types: logical

'NextPlot' — Directive on how to add next plot

'replace' (default) | 'new' | 'add'

Directive on how to add next plot, specified as a comma-separated pair consisting of 'NextPlot' and one of the values in the table:

Property Value	Effect
'new'	Creates a figure and uses it as the current figure.
'add'	Adds new graphics objects without clearing or resetting the current figure.
'replace'	Removes all axes objects and resets figure properties to their defaults before adding new graphics objects.

Legend and Title Properties

'LegendLabels' — Data tables for legend annotation

character vector | cell array of character vectors

Data tables for legend annotation, specified as a comma-separated pair consisting of 'LegendLabels' and a character vector or cell array of character vectors. Ⓐ denotes the active line for interactive operation.

Data Types: char

'LegendVisible' — Show legend label

0 (default) | 1

Show legend label, specified as a comma-separated pair consisting of 'LegendVisible' and 0 or 1.

Data Types: logical

'TitleTop' — Title to display above the polar plot

character vector

Title to display above the polar plot, specified as a comma-separated pair consisting of 'TitleTop' and a character vector.

Data Types: char

'TitleBottom' — Title to display below the polar plot
character vector

Title to display below the polar plot, specified as a comma-separated pair consisting of 'TitleBottom' and a character vector.

Data Types: char

'TitleTopOffset' — Offset between top title and angle ticks
0.1500 (default) | scalar

Offset between top title and angle ticks, specified as a comma-separated pair consisting of 'TitleTopOffset' and a scalar. The value must be in the range [-0.5,0.5].

Data Types: double

'TitleBottomOffset' — Offset between bottom title and angle ticks
0.1500 (default) | scalar

Offset between bottom title and angle ticks, specified as a comma-separated pair consisting of 'TitleBottomOffset' and a scalar. The value must be in the range [-0.5,0.5].

Data Types: double

'TitleTopFontSizeMultiplier' — Scale factor of top title font
1.1000 (default) | numeric value greater than zero

Scale factor of top title font, specified as a comma-separated pair consisting of 'TitleTopFontSizeMultiplier' and a numeric value greater than zero.

Data Types: double

'TitleBottomFontSizeMultiplier' — Scale factor of bottom title font
0.9000 (default) | numeric value greater than zero

Scale factor of bottom title font, specified as a comma-separated pair consisting of 'TitleBottomFontSizeMultiplier' and a numeric value greater than zero.

Data Types: double

'TitleTopFontWeight' — Thickness of top title font
'bold' (default) | 'normal'

Thickness of top title font, specified as a comma-separated pair consisting of 'TitleTopFontWeight' and 'bold' or 'normal.

Data Types: char

'TitleBottomFontWeight' — Thickness of bottom title font
'normal' (default) | 'bold'

Thickness of bottom title font, specified as a comma-separated pair consisting of 'TitleBottomFontWeight' and 'bold' or 'normal.

Data Types: char

'TitleTopTextInterpreter' — Interpretation of top title characters
'none' (default) | 'tex' | 'latex'

Interpretation of top title characters, specified as a comma-separated pair consisting of `'TitleTopTextInterpreter'` and:

- `'tex'` — Interpret using a subset of TeX markup
- `'latex'` — Interpret using LaTeX markup
- `'none'` — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the `TickLabelInterpreter` property is set to `'tex'`, which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or text within curly braces `{}`.

Modifier	Description	Example
`^{ }`	Superscript	`'text^{superscript}'`
`_{ }`	Subscript	`'text_{subscript}'`
`\bf`	Bold font	`'\bf text'`
`\it`	Italic font	`'\it text'`
`\sl`	Oblique font (rarely available)	`'\sl text'`
`\rm`	Normal font	`'\rm text'`
`\fontname{specifier}`	Set `specifier` as the name of a font family to change the font style. You can use this modifier with other modifiers.	`'\fontname{Courier} text'`
`\fontsize{specifier}`	Set `specifier` as a scalar numeric value to change the font size.	`'\fontsize{15} text'`
`\color{specifier}`	Set `specifier` as one of these colors: `red`, `green`, `yellow`, `magenta`, `blue`, `black`, `white`, `gray`, `darkGreen`, `orange`, or `lightBlue`.	`'\color{magenta} text'`
`\color[rgb]{specifier}`	Set `specifier` as a three-element RGB triplet to change the font color.	`'\color[rgb]{0,0.5,0.5} text'`

LaTeX Markup

To use LaTeX markup, set the `TickLabelInterpreter` property to `'latex'`. The displayed text uses the default LaTeX font style. The `FontName`, `FontWeight`, and `FontAngle` properties do not have an effect. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multiline text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

'TitleBottomTextInterpreter' — Interpretation of bottom title characters

'none' (default) | 'tex' | 'latex'

Interpretation of bottom title characters, specified as a comma-separated pair consisting of 'TitleBottomTextInterpreter' and:

- 'tex' — Interpret using a subset of TeX markup
- 'latex' — Interpret using LaTeX markup
- 'none' — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the TickLabelInterpreter property is set to 'tex', which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or the text within the curly braces {}.

Modifier	Description	Example
^{ }	Superscript	'text^{superscript}'
_{ }	Subscript	'text_{subscript}'
\bf	Bold font	'\bf text'
\it	Italic font	'\it text'
\sl	Oblique font (rarely available)	'\sl text'
\rm	Normal font	'\rm text'
\fontname{specifier}	Set specifier as the name of a font family to change the font style. You can use this modifier with other modifiers.	'\fontname{Courier} text'
\fontsize{specifier}	Set specifier as a scalar numeric value to change the font size.	'\fontsize{15} text'
\color{specifier}	Set specifier as one of these colors: red, green, yellow, magenta, blue, black, white, gray, darkGreen, orange, or lightBlue.	'\color{magenta} text'
\color[rgb]{specifier}	Set specifier as a three-element RGB triplet to change the font color.	'\color[rgb]{0,0.5,0.5} text'

LaTeX Markup

To use LaTeX markup, set the TickLabelInterpreter property to 'latex'. The displayed text uses the default LaTeX font style. The FontName, FontWeight, and FontAngle properties do not have an effect. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multiline text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

Grid Properties

`'GridOverData'` — Draw grid over data plots

`0` (default) | `1`

Draw grid over data plots, specified as a comma-separated pair consisting of `'GridOverData'` and `0` or `1`.

Data Types: `logical`

`'DrawGridToOrigin'` — Draw radial lines within innermost circle

`0` (default) | `1`

Draw radial lines within innermost circle of the polar plot, specified as a comma-separated pair consisting of `'DrawGridToOrigin'` and `0` or `1`.

Data Types: `logical`

`'GridAutoRefinement'` — Increase angle resolution

`0` (default) | `1`

Increase angle resolution in the polar plot, specified as a comma-separated pair consisting of `'GridAutoRefinement'` and `0` or `1`. This property increases angle resolution by doubling the number of radial lines outside each magnitude.

Data Types: `logical`

`'GridWidth'` — Width of grid lines

`0.5000` (default) | positive scalar

Width of grid lines, specified as a comma-separated pair consisting of `'GridWidth'` and a positive scalar.

Data Types: `double`

`'GridVisible'` — Show grid lines

`1` (default) | `0`

Show grid lines, including magnitude circles and angle radii, specified as a comma-separated pair consisting of `'GridVisible'` and `0` or `1`.

Data Types: `logical`

`'GridForegroundColor'` — Color of foreground grid lines

`[0.8000 0.8000 0.8000]` (default) | `'none'` | character vector of color names

Color of foreground grid lines, specified as a comma-separated pair consisting of `'GridForegroundColor'` and an RGB triplet, character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.

- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	🟥
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	🟩
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	🟦
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	🟦
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	🟪
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	🟨
`"black"`	`"k"`	`[0 0 0]`	`"#000000"`	⬛
`"white"`	`"w"`	`[1 1 1]`	`"#FFFFFF"`	⬜

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`"#0072BD"`	🟦
`[0.8500 0.3250 0.0980]`	`"#D95319"`	🟧
`[0.9290 0.6940 0.1250]`	`"#EDB120"`	🟨
`[0.4940 0.1840 0.5560]`	`"#7E2F8E"`	🟪
`[0.4660 0.6740 0.1880]`	`"#77AC30"`	🟩
`[0.3010 0.7450 0.9330]`	`"#4DBEEE"`	🟦
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	🟥

Data Types: `double` | `char`

`'GridBackGroundColor'` — Color of background grid lines

`'w'` (default) | character vector of color names | `'none'`

Color of background grid lines, specified as a comma-separated pair consisting of `'GridBackGroundColor'` and an RGB triplet, character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	🟥
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	🟩
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	🟦
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	🟦
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	🟪
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	🟨
`"black"`	`"k"`	`[0 0 0]`	`"#000000"`	⬛
`"white"`	`"w"`	`[1 1 1]`	`"#FFFFFF"`	⬜

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`"#0072BD"`	🟦
`[0.8500 0.3250 0.0980]`	`"#D95319"`	🟧
`[0.9290 0.6940 0.1250]`	`"#EDB120"`	🟨
`[0.4940 0.1840 0.5560]`	`"#7E2F8E"`	🟪
`[0.4660 0.6740 0.1880]`	`"#77AC30"`	🟩
`[0.3010 0.7450 0.9330]`	`"#4DBEEE"`	🟦
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	🟥

Data Types: `double` | `char`

Marker, Color, Line, and Font Properties

`'Marker'` — Marker symbol

`'none'` (default) | character vector of symbols

Marker symbol, specified as a comma-separated pair consisting of `'Marker'` and either `'none'` or one of the symbols in this table. By default, a line does not have markers. Add markers at selected points along the line by specifying a marker.

Marker	Description	Resulting Marker
`"o"`	Circle	○
`"+"`	Plus sign	+
`"*"`	Asterisk	✳
`"."`	Point	•

Marker	Description	Resulting Marker
`"x"`	Cross	×
`"_"`	Horizontal line	—
`"|"`	Vertical line	|
`"square"`	Square	□
`"diamond"`	Diamond	◇
`"^"`	Upward-pointing triangle	△
`"v"`	Downward-pointing triangle	▽
`">"`	Right-pointing triangle	▷
`"<"`	Left-pointing triangle	◁
`"pentagram"`	Pentagram	☆
`"hexagram"`	Hexagram	✡
`"none"`	No markers	Not applicable

`'MarkerSize'` — Marker size

6 (default) | positive value

Marker size, specified as a comma-separated pair consisting of `'MarkerSize'` and a positive value in point units.

Data Types: `double`

`'ColorOrder'` — Colors to use for multiline plots

seven predefined colors (default) | three-column matrix of RGB triplets

Colors to use for multi-line plots, specified as a comma-separated pair consisting of `'ColorOrder'` and a three-column matrix of RGB triplets. Each row of the matrix defines one color in the color order.

Data Types: `double`

`'ColorOrderIndex'` — Next color to use in color order

1 (default) | positive integer

Next color to use in color order, specified as a comma-separated pair consisting of `'ColorOrderIndex'` and a positive integer. New plots added to the axes use colors based on the current value of the color order index.

Data Types: `double`

`'EdgeColor'` — Color of data lines

`'k'` (default) | RGB triplet vector

Color of data lines, specified as a comma-separated pair consisting of `'EdgeColor'` and a character vector of color names or RGB triplet vector.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	🟥
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	🟩
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	🟦
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	🟦
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	🟪
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	🟨
`"black"`	`"k"`	`[0 0 0]`	`"#000000"`	⬛
`"white"`	`"w"`	`[1 1 1]`	`"#FFFFFF"`	⬜

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`"#0072BD"`	🟦
`[0.8500 0.3250 0.0980]`	`"#D95319"`	🟧
`[0.9290 0.6940 0.1250]`	`"#EDB120"`	🟨
`[0.4940 0.1840 0.5560]`	`"#7E2F8E"`	🟪
`[0.4660 0.6740 0.1880]`	`"#77AC30"`	🟩
`[0.3010 0.7450 0.9330]`	`"#4DBEEE"`	🟦
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	🟥

Data Types: `double` | `char`

`'LineStyle'` — Line style of the plot

`'-'` (default) | `'--'` | `':'` | `'-.'` | `'none'`

Line style of the plot, specified as a comma-separated pair consisting of `'LineStyle'` and one of the symbols in the table:

Symbol	Line Style	Resulting Line
`'-'`	Solid line	————
`'--'`	Dashed line	- — — — —
`':'`	Dotted line	··················
`'-.'`	Dash-dotted line	—·—·—·—·-
`'none'`	No line	No line

`'LineWidth'` — Line width of plot
`1` (default) | positive scalar | positive vector

Line width of the plot, specified as a comma-separated pair consisting of `'LineWidth'` and a positive scalar or vector.

`'FontSize'` — Font size of text in plot
`10` (default) | positive scalar

Font size of text in the plot, specified as a comma-separated pair consisting of `'FontSize'` and a positive scalar.

`'FontSizeAutoMode'` — Set font size
`'auto'` (default) | `'manual'`

Set font size, specified as a comma-separated pair consisting of `'FontSizeAutoMode'` and `'auto'` or `'manual'`.

Data Types: `char`

See Also

“Interact with Polar Plot”

6

RF Propagation Objects and Methods

siteviewer

Create Site Viewer

Description

Display transmitter sites, receiver sites, and RF propagation visualizations by using a `siteviewer` object. By default, Site Viewer displays a 3-D view of the globe. When you display sites on the globe, they are referenced to geographic coordinates. You can customize the globe using custom terrain, high-zoom-level or custom basemaps, and buildings.

You can also import and view 3-D models represented by standard tessellation language (STL) files or `triangulation` objects. When you display sites on a 3-D model, they are referenced to Cartesian coordinates.

Site Viewer requires hardware graphics support for WebGL™.

Creation

Syntax

```
viewer = siteviewer
viewer = siteviewer(Name,Value)
```

Description

`viewer = siteviewer` creates a Site Viewer.

`viewer = siteviewer(Name,Value)` specifies Site Viewer properties using name-value arguments. For example, import and view a 3-D model file by using the `SceneModel` name-value argument.

Properties

Site Viewer

`Name` — Caption to display on map viewer window
`'Site Viewer'` (default) | character vector | string scalar

Caption to display on map viewer window, specified as a character vector or a string scalar.

Data Types: `char` | `string`

`Position` — Size and location of map viewer window in pixels
four-element integer-valued vector

Size and location of map viewer window in pixels, specified as a four-element integer-valued vector in the form `[left bottom width height]`. The default value depends on the screen resolution such that the window lies in the center of the screen with a width of 800 pixels and a height of 600 pixels.

Data Types: `double`

`CoordinateSystem` — Coordinate reference system

`'geographic'` (default) | `'cartesian'`

This property is read-only.

Coordinate reference system, specified as `'geographic'` or `'cartesian'`. The value of `CoordinateSystem` depends on how you create the Site Viewer.

- By default, the value of `CoordinateSystem` is `'geographic'`and visualizations are referenced to the WGS84 reference ellipsoid.
- When you create the Site Viewer by specifying the `SceneModel` argument, the value of `CoordinateSystem` is `'cartesian'` and coordinates are referenced to Cartesian coordinates.

When `CoordinateSystem` is `'geographic'`, you can view the latitude and longitude coordinates for a location by right-clicking the map and selecting **Show Location**. To remove the location, right-click and select **Remove Location**.

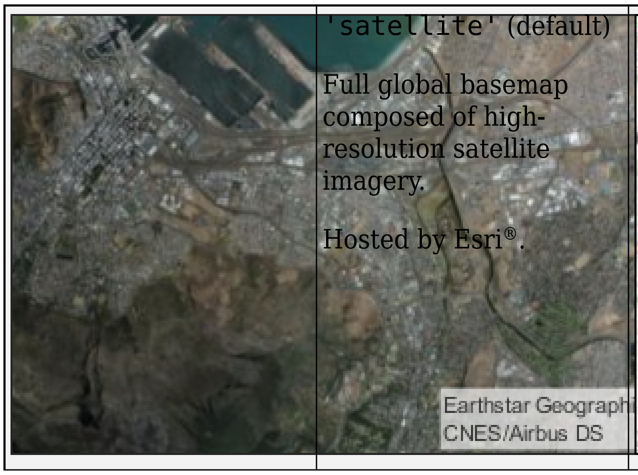
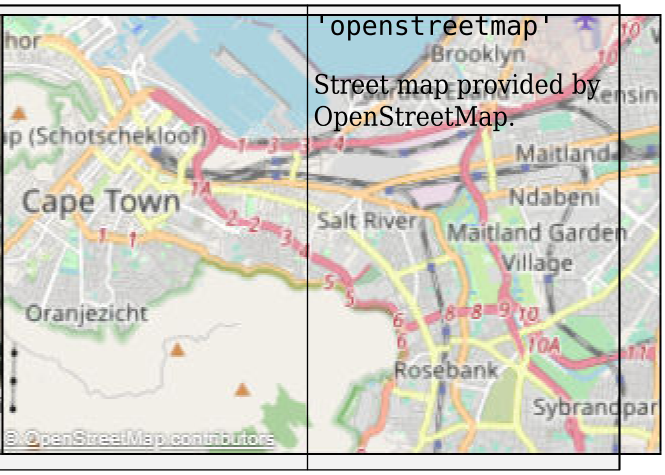
Data Types: `char` | `string`

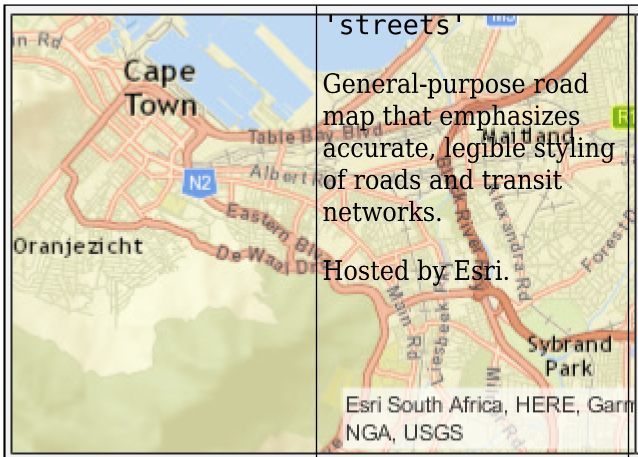
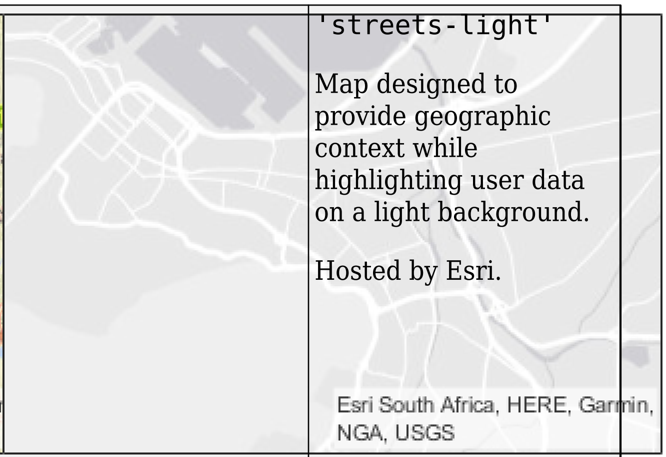
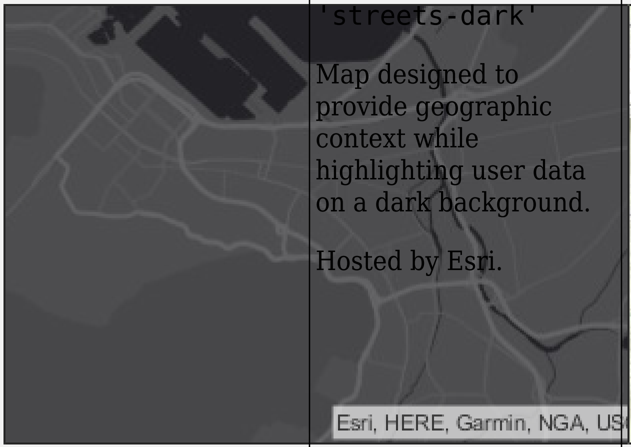
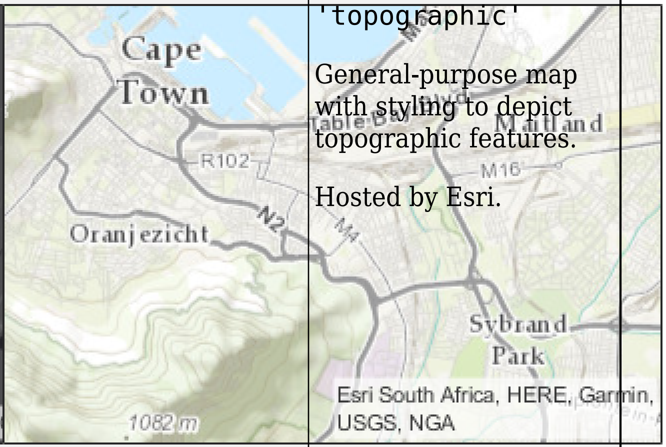
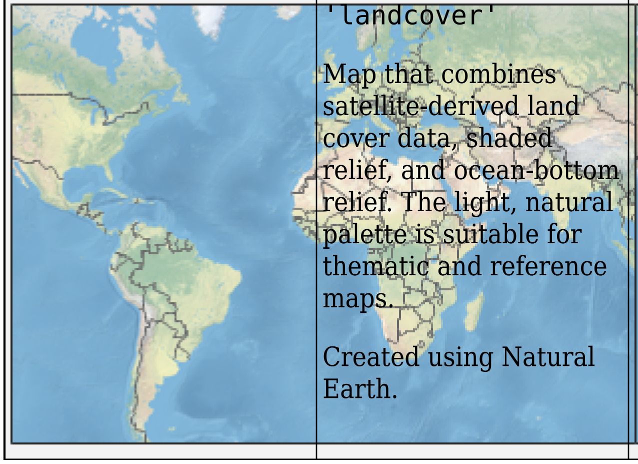
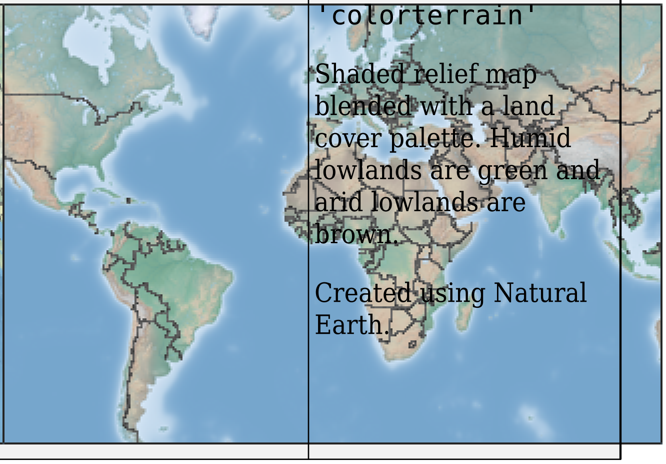
Geographic Coordinate System

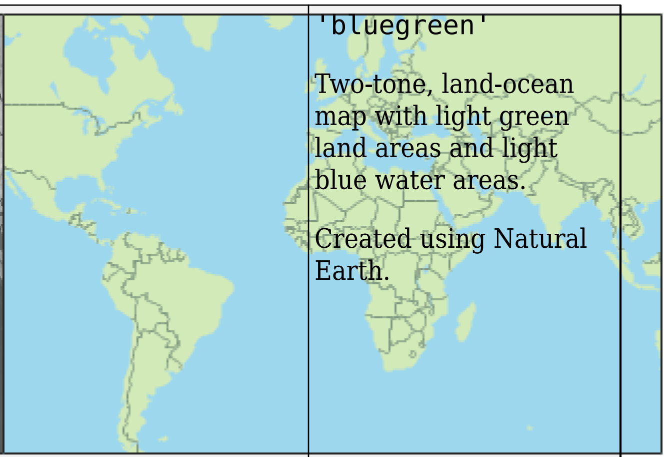
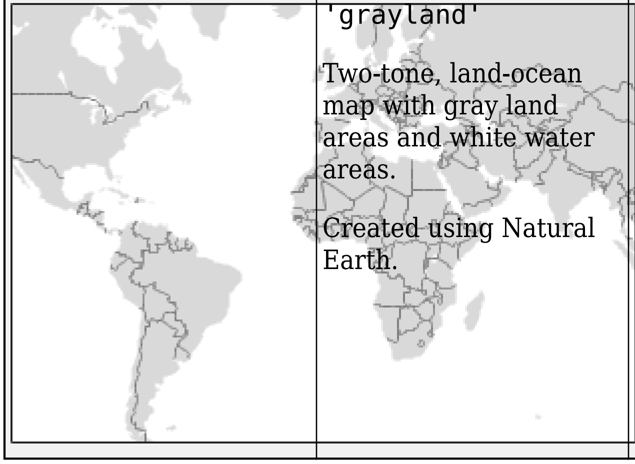
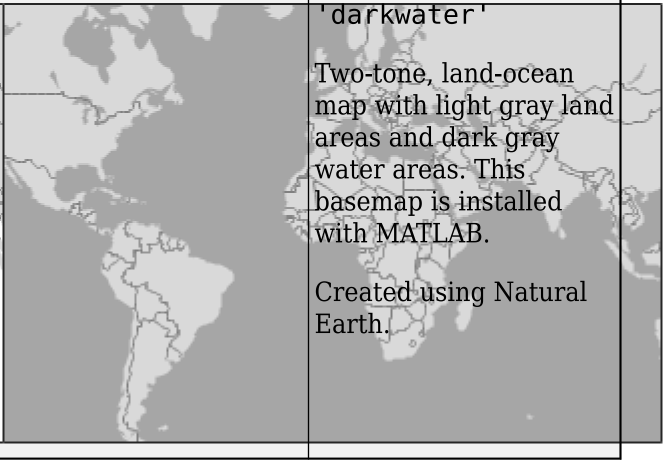
`Basemap` — Map imagery used to visualize sites

`'satellite'` (default) | `'openstreetmap'` | `'streets'` | `'streets-light'` | `'streets-dark'` | `'topographic'` | ...

Map imagery used to visualize sites, specified as one of the basemap names in this table or as a custom basemap defined using the `addCustomBasemap` function.

	`'satellite'` (default) Full global basemap composed of high-resolution satellite imagery. Hosted by Esri®.	 Cape Town (Schotschekloof) Oranjezicht Brooklyn Maitland Ndabeni Salt River Maitland Garden Village Rosebank Sybrandpar © OpenStreetMap contributors	`'openstreetmap'` Street map provided by OpenStreetMap.

 Cape Town Oranjezicht N2 Table Bay Blvd Albert Rd Eastern Blvd De Waal Dr Maitland Main Rd Sybrand Park Esri South Africa, HERE, Garmin, NGA, USGS	`'streets'` General-purpose road map that emphasizes accurate, legible styling of roads and transit networks. Hosted by Esri.		`'streets-light'` Map designed to provide geographic context while highlighting user data on a light background. Hosted by Esri.
	`'streets-dark'` Map designed to provide geographic context while highlighting user data on a dark background. Hosted by Esri.	 Cape Town Oranjezicht R102 N2 M16 M4 Maitland Sybrand Park 1082 m Esri South Africa, HERE, Garmin, USGS, NGA	`'topographic'` General-purpose map with styling to depict topographic features. Hosted by Esri.
	`'landcover'` Map that combines satellite-derived land cover data, shaded relief, and ocean-bottom relief. The light, natural palette is suitable for thematic and reference maps. Created using Natural Earth.		`'colorterrain'` Shaded relief map blended with a land cover palette. Humid lowlands are green and arid lowlands are brown. Created using Natural Earth.

	`'grayterrain'` Terrain map in shades of gray. Shaded relief emphasizes both high mountains and micro-terrain found in lowlands. Created using Natural Earth.		`'bluegreen'` Two-tone, land-ocean map with light green land areas and light blue water areas. Created using Natural Earth.
	`'grayland'` Two-tone, land-ocean map with gray land areas and white water areas. Created using Natural Earth.		`'darkwater'` Two-tone, land-ocean map with light gray land areas and dark gray water areas. This basemap is installed with MATLAB. Created using Natural Earth.

The basemaps hosted by Esri update periodically. As a result, you might see differences in your visualizations over time.

Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks®.

This property applies only when `CoordinateSystem` is `'geographic'`.

Data Types: `char` | `string`

`Terrain` — Data on which to visualize sites and perform terrain calculations

`'gmted2010'` (default) | `'none'` | character vector | scalar

Data on which to visualize sites and perform terrain calculations, specified as a character vector or a scalar previously added using `addCustomTerrain` or one of the following options:

- `'none'` — Terrain elevation is `0` everywhere.
- `'gmted2010'` — USGS GMTED2010 terrain data. This option requires an internet connection.

This property applies only when `CoordinateSystem` is `'geographic'`.

This property is read-only after you create the Site Viewer.

For limitations, see “Limitations” on page 6-18.

Data Types: `char` | `string`

`Buildings` — Name of OpenStreetMap (.osm) file to use as buildings data
string scalar | character vector

Name of the `OpenStreetMap` (.osm) file to use as buildings data, specified as a string scalar or a character vector. The file must be in the current directory, in a directory on the MATLAB path. You can also use a full or relative path to the file to specify the data. By default, this value is empty.

This property applies only when `CoordinateSystem` is `'geographic'`.

This property is read-only after you create the Site Viewer.

For limitations, see "Limitations" on page 6-18.

Data Types: `char` | `string`

Cartesian Coordinate System

`SceneModel` — Name of 3-D model file or triangulation
character vector | string scalar | `triangulation` object

Name of the 3-D model file or triangulation, specified as a string scalar, a character vector, or a `triangulation` object.

When `SceneModel` is the name of a 3-D model file, you must specify an STL file with extension `.stl`. The form of `SceneModel` depends on the location of your file.

- If the file is in your current folder or in a folder on the MATLAB path, then specify the name of the file, such as `'myFile.stl'`.
- If the file is not in the current folder or in a folder on the MATLAB path, then specify the full or relative path name, such as `'C:\myfolder\myFile.stl'` or `'dataDir\myFile.stl'`.

This property applies only when `CoordinateSystem` is `'cartesian'`.

This property is read-only after you create the Site Viewer.

Data Types: `char` | `string`

`Transparency` — Transparency of model
scalar in the range `[0,1]`

Transparency of the model, specified as a scalar in the range `[0,1]`, where `0` is transparent and `1` is opaque. The default is `0.1` when `ShowEdges` is `1` (`true`), and `1` otherwise.

This property applies only when `CoordinateSystem` is `'cartesian'`.

Data Types: `double`

`ShowOrigin` — Option to show origin
`true` or `1` (default) | `false` or `0`

Option to show the origin of the model, specified as numeric or logical `1` (`true`) or `0` (`false`). The *x*-axis appears in red, the *y*-axis appears in green, and the *z*-axis appears in blue. The *z*-axis of the scene points up.

This property applies only when `CoordinateSystem` is `'cartesian'`.

Data Types: `logical`

ShowEdges — Option to show edges of model
`true` or `1` (default) | `false` or `0`

Option to show the edges of the model using black lines, specified as numeric or logical `1` (`true`) or `0` (`false`). Site Viewer defines edges as two adjacent facets with normals that differ by more than two degrees.

This property applies only when `CoordinateSystem` is `'cartesian'`.

This property is read-only after you create the Site Viewer.

Data Types: `logical`

Object Functions

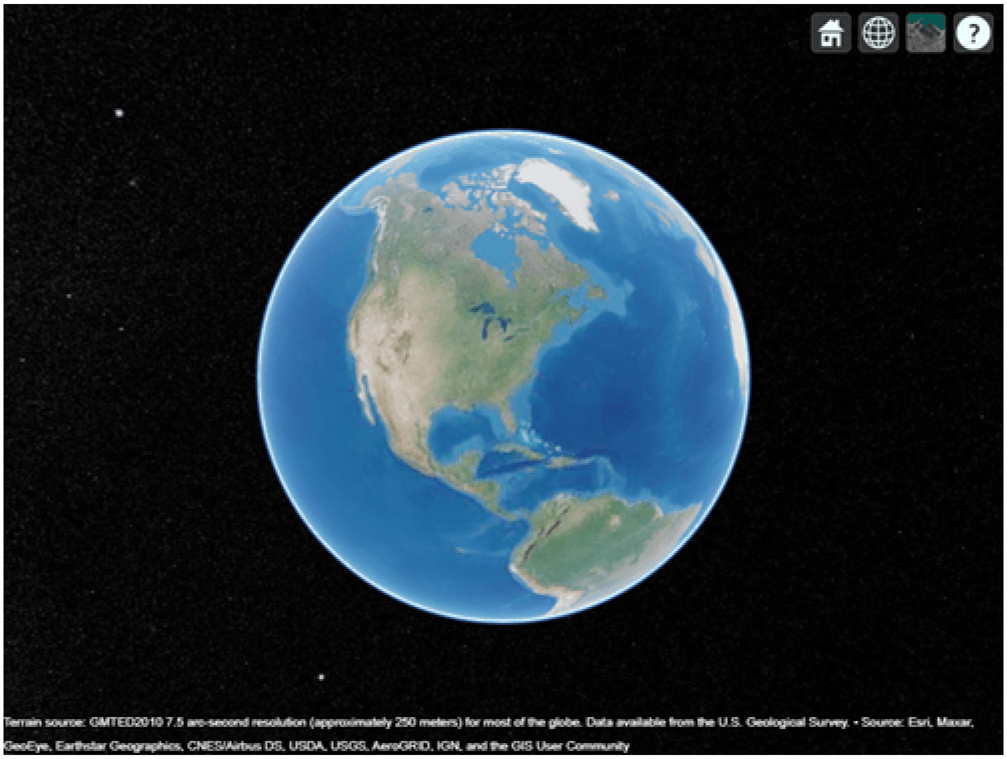
clearMap	Clear plots
close	Close Site Viewer

Examples

Default Site Viewer Map Display

Create a default Site Viewer.

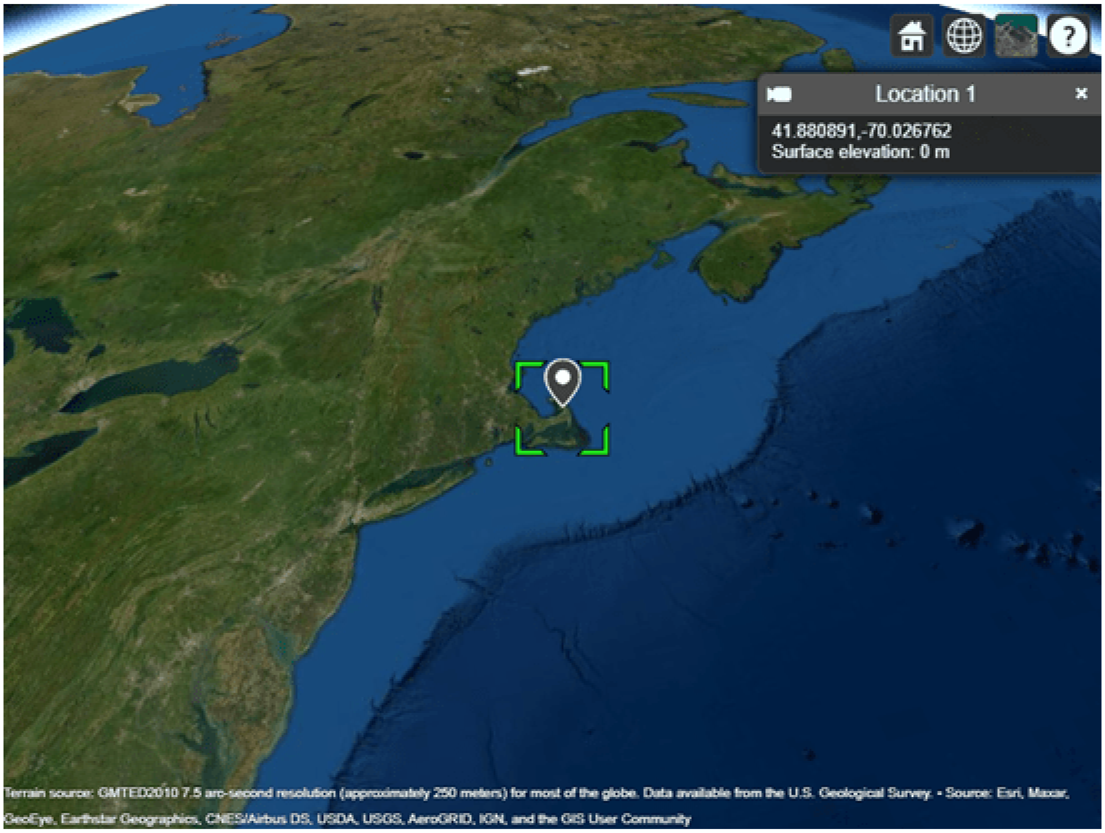
```
viewer = siteviewer;
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Pan by left-clicking and dragging, zoom by right-clicking and dragging or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking or dragging. View the coordinates for a location by right-clicking and selecting **Show location**.

For this example, navigate to a region containing New England and view the coordinates for a location near Cape Cod.



Location 1

41.880891,-70.026762
Surface elevation: 0 m

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

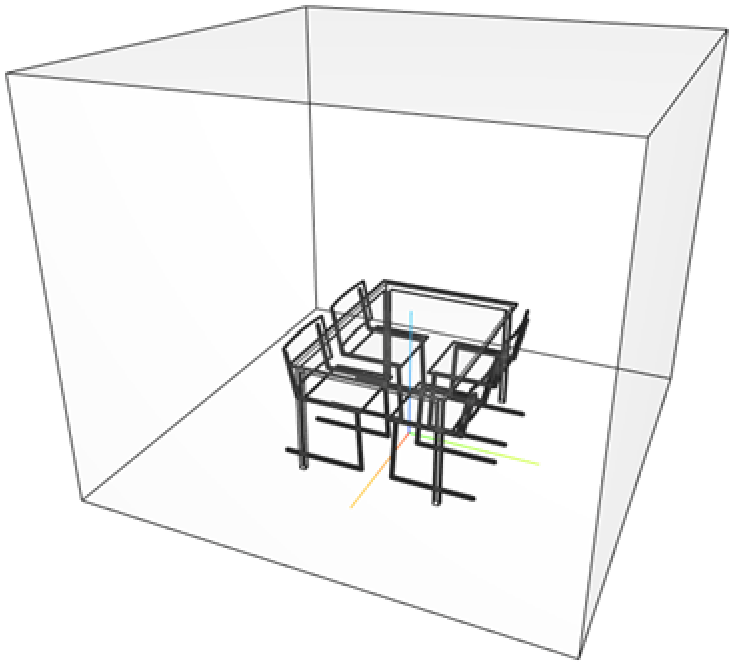
A gray marker appears at the location you selected. Remove the marker by right-clicking the location and selecting **Remove location.**

Site Viewer with 3-D Model

Import and view an STL file. The file models a small conference room with one table and four chairs.

```
viewer = siteviewer("SceneModel","conferenceroom.stl");
```

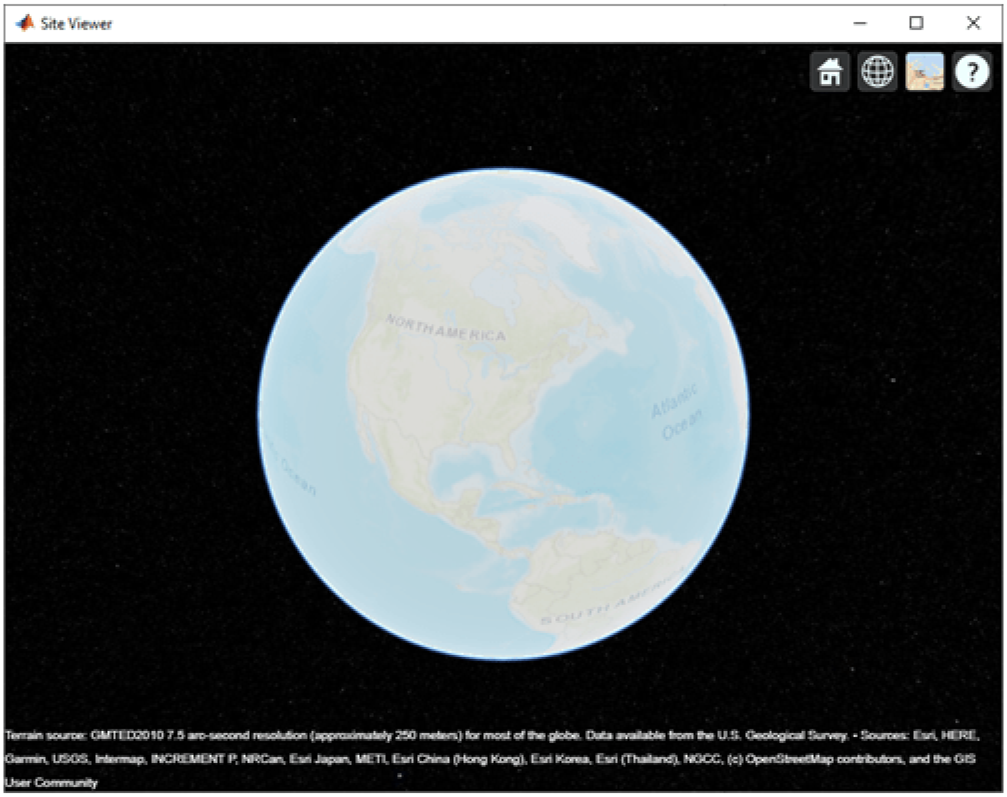
Pan by left-clicking, zoom by right-clicking or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking and dragging.



View Transmitter Site On Site Viewer

Launch a Site Viewer with `streets` basemap.

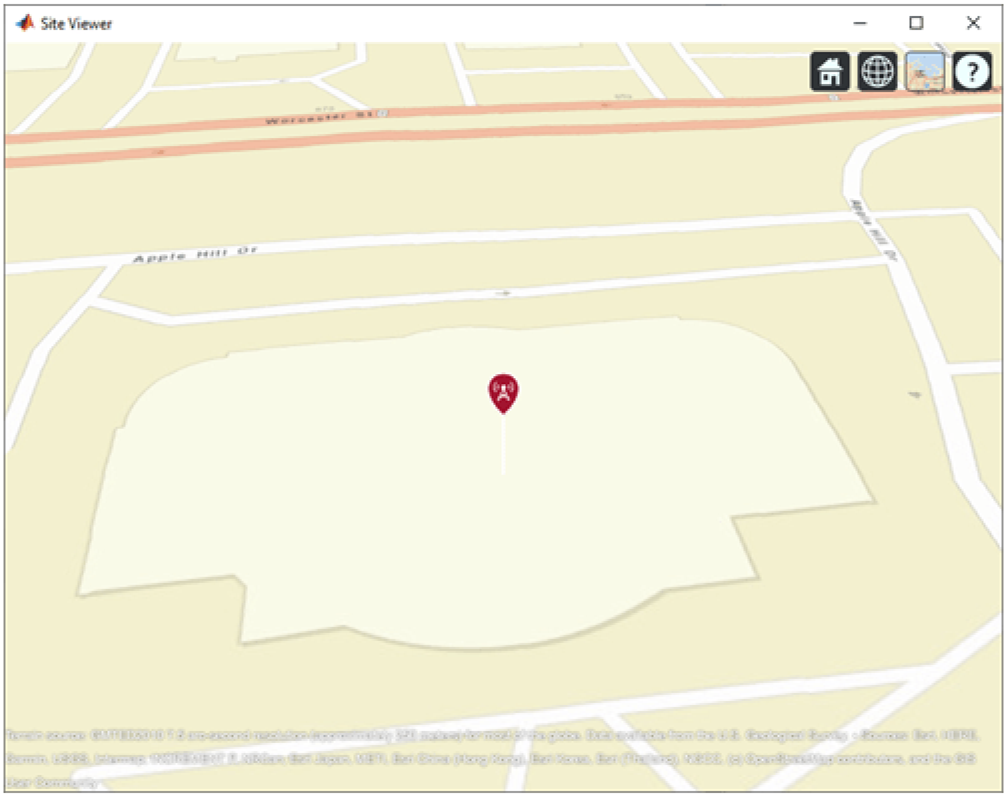
```
viewer = siteviewer("Basemap","streets");
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, HERE, Garmin, USGS, Intermap, INCREMENT P, NRCan, Esri Japan, METI, Esri China (Hong Kong), Esri Korea, Esri (Thailand), NGCC, (c) OpenStreetMap contributors, and the GIS User Community

View a transmitter site on this map.

```
tx = txsite;
show(tx)
```



Compare Coverage Maps

Launch two Site Viewer windows. One Site Viewer window uses the terrain model and the other window does not use the terrain model.

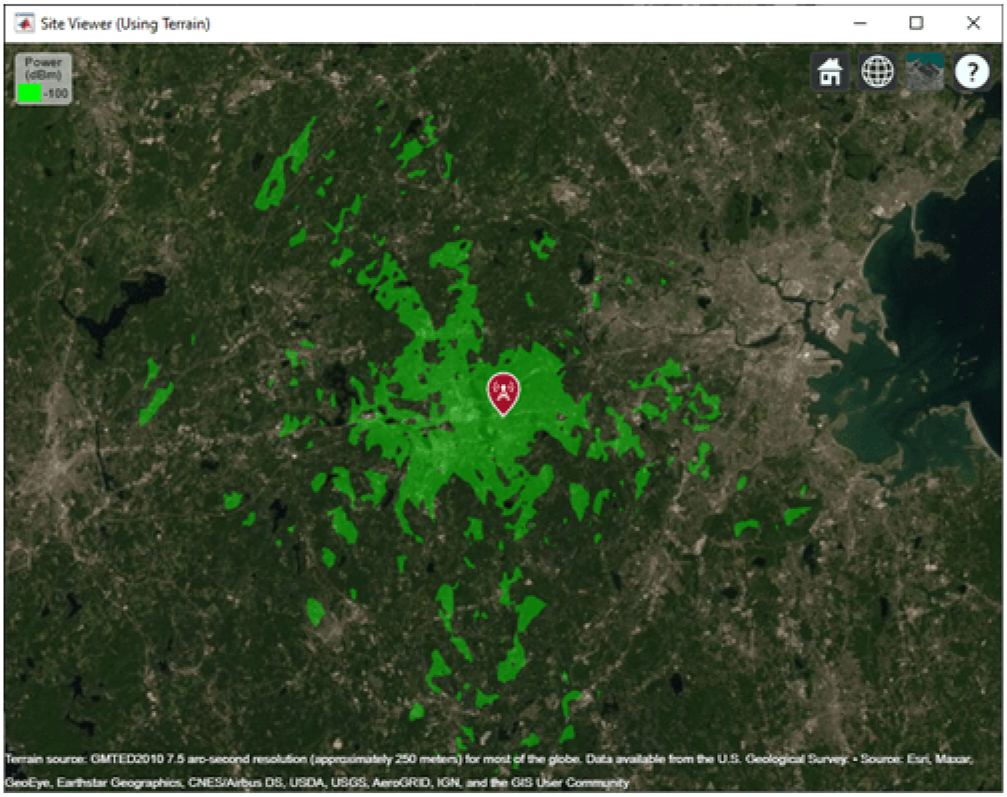
```
viewer1 = siteviewer("Terrain","gmted2010","Name","Site Viewer (Using Terrain)");
viewer2 = siteviewer("Terrain","none","Name","Site Viewer (No Terrain)");
```

Create a transmitter site.

```
tx = txsite;
```

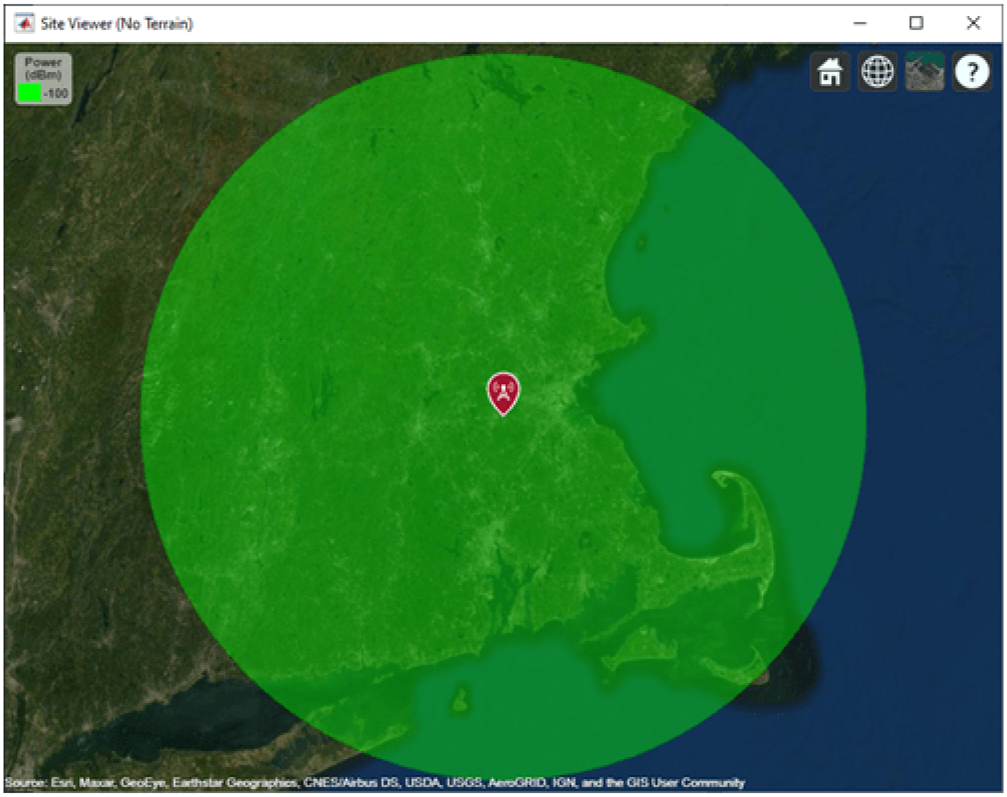
Generate a coverage map on each window. The map with terrain uses the Longley-Rice propagation model by default.

```
coverage(tx,"Map",viewer1)
```


Site Viewer (Using Terrain)
Power (dBm)
🟩 -100
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

The map without terrain uses the free-space model by default.

```
coverage(tx,"Map",viewer2)
```

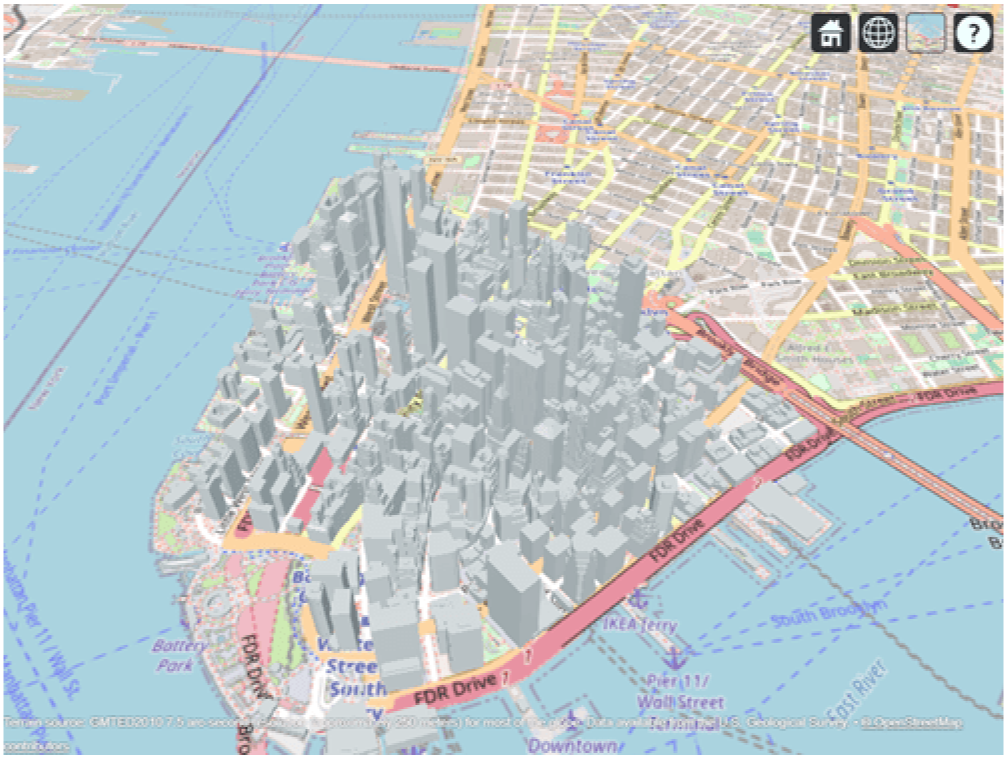


Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Site Viewer with Buildings

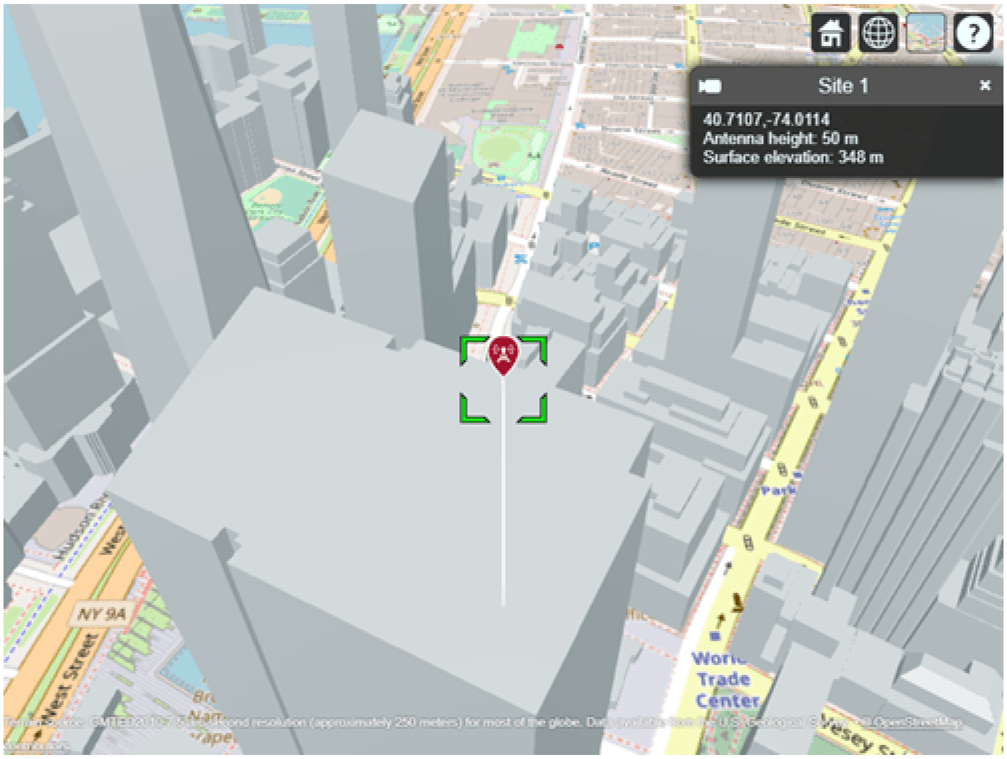
Launch Site Viewer with a basemap and buildings file for Manhattan. For more information about the osm file, see [1] on page 6-15.

```
viewer = siteviewer("Basemap","openstreetmap",...
       "Buildings","manhattan.osm");
```


FDR Drive
FDR Drive
Battery Park
IKEA ferry
Pier 11/ Wall Street
South Brooklyn
East River
Downtown
Brooklyn Bridge
Terrain source: GMTED2010 7.5 arc-second ... Data available from the U.S. Geological Survey • © OpenStreetMap contributors

Show a transmitter site on a building.

```
tx = txsite("Latitude",40.7107,...
       "Longitude",-74.0114,...
       "AntennaHeight",50);
show(tx)
```



Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Add and Remove a Custom Basemap

Add a custom basemap to view locations on an OpenTopoMap® basemap, then remove the custom basemap from `siteviewer`.

Initialize simulation variables to:

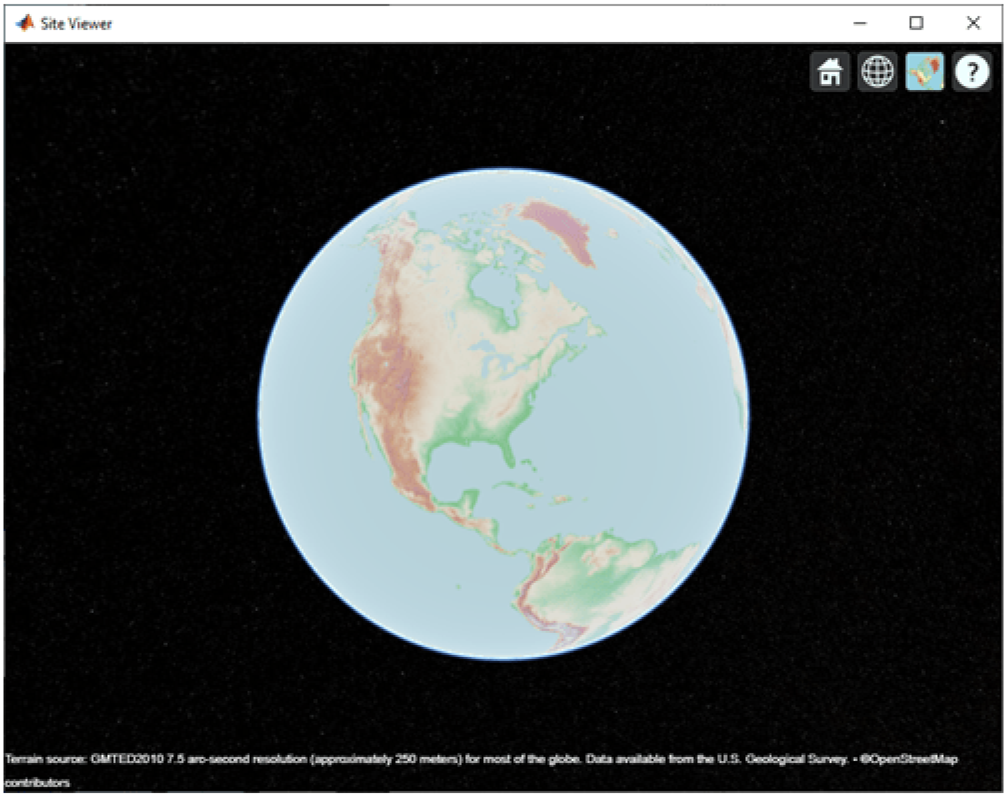
- Define the name that you will use to specify your custom basemap.
- Specify the website that provides the map data. The first character of the URL indicates which server to use to get the data. For load balancing, the provider has three servers that you can use: a, b, or c.
- Create an attribution to display on the map that gives credit to the provider of the map data. Web map providers might define specific requirements for the attribution.
- Define a display name for the custom map.

```
name = 'opentopomap';
url = 'a.tile.opentopomap.org';
copyright = char(uint8(169));
```

```
attribution = copyright + "OpenStreetMap contributors";
displayName = 'Open Topo Map';
```

Use `addCustomBasemap` to load the custom basemap, and then create a `siteviewer` object that loads the custom basemap.

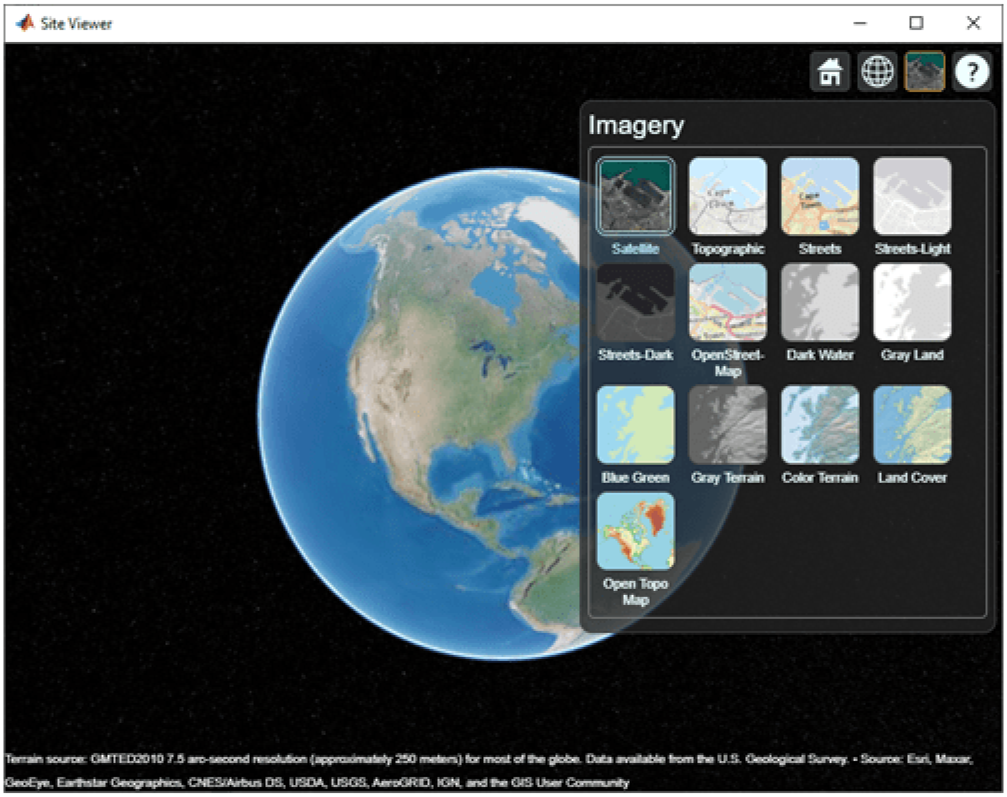
```
addCustomBasemap(name,url,'Attribution',attribution','DisplayName',displayName)
viewer = siteviewer('Basemap',name);
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • ©OpenStreetMap contributors

After a custom basemap is added to `siteviewer`, the custom map is available for future calls to `siteviewer`. Note the `'Open Topo Map'` icon in the `Imagery` tab.

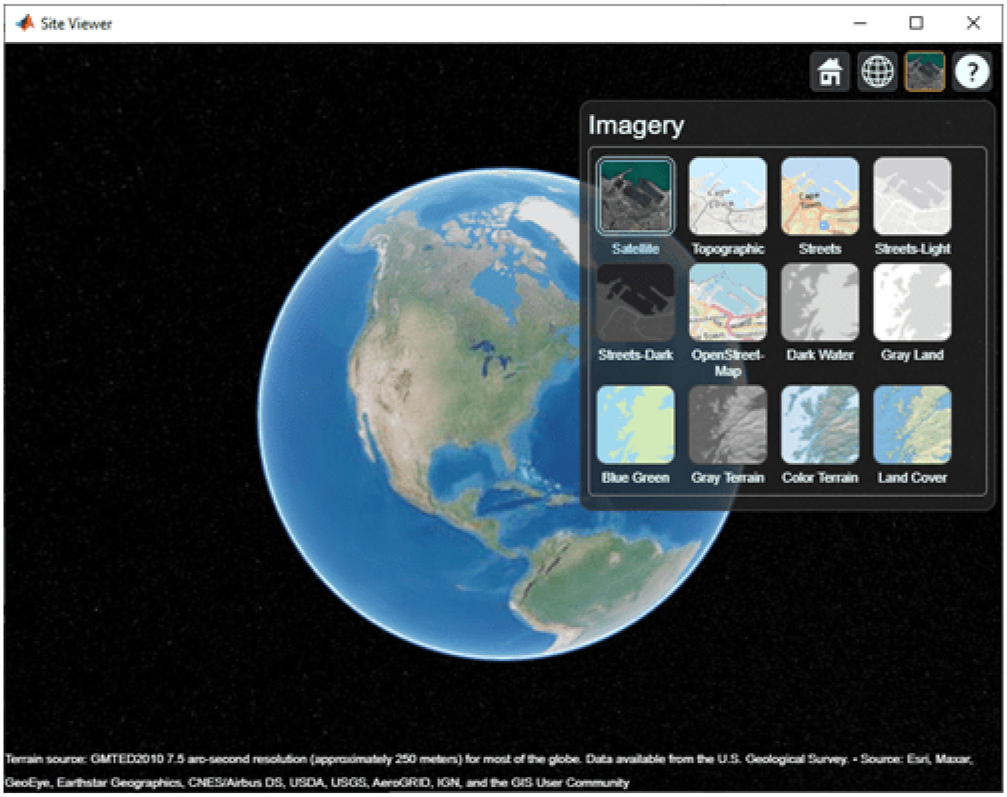
```
siteviewer;
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Use `removeCustomBasemap` to remove the custom basemap from future calls to `siteviewer`. Note the `'Open Topo Map'` icon is no longer available in the `Imagery` tab.

```
removeCustomBasemap(name)
siteviewer;
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Limitations

Terrain

- Default terrain access requires an internet connection. If no internet connection exists, then Site Viewer automatically uses `'none'` in the property `Terrain`.
- Custom DTED terrain files for use with `addCustomTerrain` must be acquired outside of MATLAB for example by using USGS EarthExplorer.
- When using custom terrain, analysis is restricted to the terrain region. For example, an error occurs if you are trying to show a transmitter or receiver site outside of the region.

Buildings

- OpenStreetMap files obtained from https://www.openstreetmap.org represent crowd-sourced map data, and the completeness and accuracy of the buildings data may vary depending on the map location.
- When downloading data from https://www.openstreetmap.org, select an export area larger than the desired area to ensure that all expected building features are fully captured. Building features at the edge of the selected export area may be missing.
- Building geometry and features are interpreted from the file according to the recommendations of OpenStreetMap for 3-D buildings.

MATLAB Online

- In MATLAB Online™, if you refresh the URL, then the Site Viewer window remains open but the visualizations disappear.

More About

Site Viewer Navigation

You can interactively navigate Site Viewer by using your mouse.

- Pan by left-clicking and dragging.
- Zoom by scrolling or by right-clicking and dragging.
- Tilt and rotate by holding **Ctrl** and dragging or by middle-clicking and dragging.

When `CoordinateSystem` is `'geographic'`, you can restore the default view by selecting **Restore View** from the toolbar.

Dimension Picker

When `CoordinateSystem` is `'geographic'`, you can choose between three view options by using the **Dimension Picker** in the toolbar.

- **3-D View** — A smooth globe. This is the default view.
- **2-D View** — A flat map in the Mercator projection.
- **Columbus View** — A flat map in the Mercator projection that supports tilt and rotation.

Some interactions are not supported for **2-D View** and **Columbus View**.

Version History

Introduced in R2019a

See Also

Functions

`addCustomTerrain` | `addCustomBasemap` | `removeCustomTerrain` | `removeCustomBasemap`

Objects

`txsite` | `rxsite`

Topics

“RF Propagation and Visualization”

txsite

Create RF transmitter site

Description

Use the `txsite` object to create a radio frequency transmitter site.

A transmitter consists of an RF circuit and an antenna, where the RF circuit excites the antenna with a signal and power. Key characteristics of a transmitter include its output power, operating frequency, and antenna radiation pattern.

Creation

Syntax

```
tx = txsite
tx = txsite(coordsys)
tx = txsite( ___ ,Name,Value)
```

Description

`tx = txsite` creates a radio frequency transmitter site.

`tx = txsite(coordsys)` creates a transmitter site with the specified coordinate system. You can specify either `'geographic'` or `'cartesian'` coordinate system.

`tx = txsite( ___ ,Name,Value)` sets properties using one or more name-value pairs. For example, `tx = txsite('Name','TX Site')` creates a transmitter site with the name `TX Site`. Enclose each property name in quotes.

You can create multiple transmitter sites by using `Name`, `Latitude`, and `Longitude` properties. For example: `names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"]; lats = [42.3467,42.3598,42.3763]; lons = [-71.0972,-71.0545,-71.0611];`. The `CoordinateSystem` property must be a string scalar or a character vector.

Properties

Name — Site name
character vector | string | row or column vector

Site name, specified as a character vector or string or as a row or column vector of *N* elements. Specifying name as a row or column vector creates multiple sites.

Example: `'Name','Site 2'`

Example: `tx.Name = 'Fenway Park'`

Example: `names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"]; tx = txsite('Name',names)`

Data Types: `char` | `string`

`CoordinateSystem` — Coordinate system used to site location
`'geographic'` (default) | `'cartesian'`

Coordinate system used to the site location, specified as `'geographic'` or `'cartesian'`. If you specify `'geographic'`, the site location is defined using the `Latitude`, `Longitude`, and `AntennaHeight` properties. If you specify `'cartesian'`, the site location is defined using the `AntennaPosition` property.

Example: `'CoordinateSystem','cartesian'`

Example: `tx.CoordinateSystem = 'cartesian'`

`Latitude` — Site latitude coordinates
`42.3001` (default) | numeric scalar in the range `[-90 90]` | row or column vector

Site latitude coordinates, specified as a numeric scalar in the range of `-90` to `90`, or as a row or column vector of *N* elements in the range `[-90 90]`. Specifying latitude as a row or column vector creates multiple sites. The coordinates are defined using the world geodesic system of 1984 (WGS-84) reference ellipsoid. Latitude specifies north-south position.

Example: `'Latitude',45.098`

Example: `tx.Latitude = 45.098`

Example: `latitude = [42.3467,42.3598,42.3763]; tx = txsite('Latitude',latitude)`

Dependencies

To enable this property, set `CoordinateSystem` to `'geographic'`.

`Longitude` — Site longitude coordinates
`-71.3504` (default) | numeric scalar in the range `[-180 180]` | row or column vector

Site longitude coordinates, specified as a numeric scalar in the range `[-180 180]`or as a row or column vector of *N* elements in the range `[-180 180]`. Specifying longitude as a row or column vector creates multiple sites. The coordinates are defined using the world geodesic system of 1984 (WGS-84) reference ellipsoid. Longitude specifies the east-west the position.

Example: `'Longitude',-68.890`

Example: `tx.Longitude = -71.0972`

Example: `longitude = [-71.0972,-71.0545,-71.0611]; tx = txsite('Longitude',longitude)`

Dependencies

To enable this property, set the `CoordinateSystem` to `'geographic'`.

`Antenna` — Antenna element or array
`'isotropic'` (default) | object | row vector

Antenna element or array specified as one of these options.

- `'isotropic'` to model an antenna that radiates uniformly in all directions.
- An antenna element from the “Antenna Catalog” or array elements from the “Array Catalog”.

> **Note** When using antenna elements, please use the `design` function to design the antenna at the required receive frequency. Then add this antenna element to the transmitter site.

- If you have Communications Toolbox™, an `arrayConfig` object.
- If you have Phased Array System Toolbox™, any antenna object in “Antennas, Microphones, and Sonar Transducers” (Phased Array System Toolbox) or any array object in “Array Geometries and Analysis” (Phased Array System Toolbox).

Example: `'Antenna'`,`monopole`

Example: `tx.Antenna = monopole`

`AntennaAngle` — Antenna X-axis angle
`0` (default) | numeric scalar | 2-by-1 vector | 2-by-*N* matrix

Antenna X-axis angle defined with reference to a local Cartesian coordinate system, specified as a numeric scalar representing an azimuth angle in degrees or as a 2-by-1 vector or a 2-by-*N* matrix representing both azimuth and elevation angles with each element unit in degrees.

The azimuth angle is measured counterclockwise from the east along the antenna X-axis (for geographical sites) or from the global X-axis around the global Z-axis (for Cartesian sites). Specify the azimuth angle between `-180` to `180`. degrees.

The elevation angle is measured from antenna X-axis along the horizontal or XY plane. Specify the elevation angle between`-90` to `90` degrees.

Example: `'AntennaAngle'`,`25`

Example: `tx.AntennaAngle = [25,-80]`

`AntennaHeight` — Antenna height above surface
`10` (default) | nonnegative numeric scalar | row vector

Antenna height from the ground or building surface, specified as a nonnegative numeric scalar in meters. Maximum value for this property is 6,371,000 m.

If the site location coincides with the building location, the antenna height is measured from the top of the building to the center of the antenna. Otherwise,the height is measured from ground elevation to the center of the antenna.

Example: `'AntennaHeight'`,`25`

Example: `tx.AntennaHeight = 15`

Dependencies

To enable this property, set `CoordinateSystem` to `'geographic'`.

Data Types:

`AntennaPosition` — Position of antenna center
`[0;0;0]` (default) | 3-by-1 vector

Position of the antenna center, specified as a 3-by-1 vector representing X-, Y-, and Z-axis Cartesian coordinates with each element in meters.

Example: `'AntennaPosition'`,`[0;2;4]`

Example: `tx.AntennaPosition = [0;2;4]`

Dependencies

To enable this property, set `CoordinateSystem` to `'cartesian'`.

Data Types:

`SystemLoss` — System loss

`0` (default) | nonnegative scalar | row vector

System loss, specified as a nonnegative scalar in dB.

System loss includes transmission line loss and any other miscellaneous system losses.

Example: `'SystemLoss'`,`10`

Example: `txsite.SystemLoss = 10`

Data Types:

`TransmitterFrequency` — Transmitter operating frequency

`1.9000e+09` (default) | positive scalar | row vector

Transmitter operating frequency, specified as a positive scalar in Hz. in the range `[1e3 200e9`.

Example: `'TransmitterFrequency'`,`30e9`

Example: `txsite.TransmitterFrequency = 30e9`

Data Types: `double`

`TransmitterPower` — Signal power at transmitter output

`10` (default) | positive scalar

Signal power at transmitter output, specified as a positive scalar in watts. The transmitter out is connected to the antenna.

Example: `'TransmitterPower'`,`30`

Example: `txsite.TransmitterPower = 30`

Data Types: `double`

Object Functions

show	Show site in Site Viewer
hide	Hide site from Site Viewer
distance	Distance between sites
angle	Angle between sites
elevation	Elevation of site
location	Coordinates at distance and angle from site
los	Display or compute line-of-sight (LOS) visibility status
coverage	Display or compute coverage map
sinr	Display or compute signal-to-interference-plus-noise (SINR) ratio
pattern	Display antenna radiation pattern in Site Viewer

Examples

Default Transmitter Site

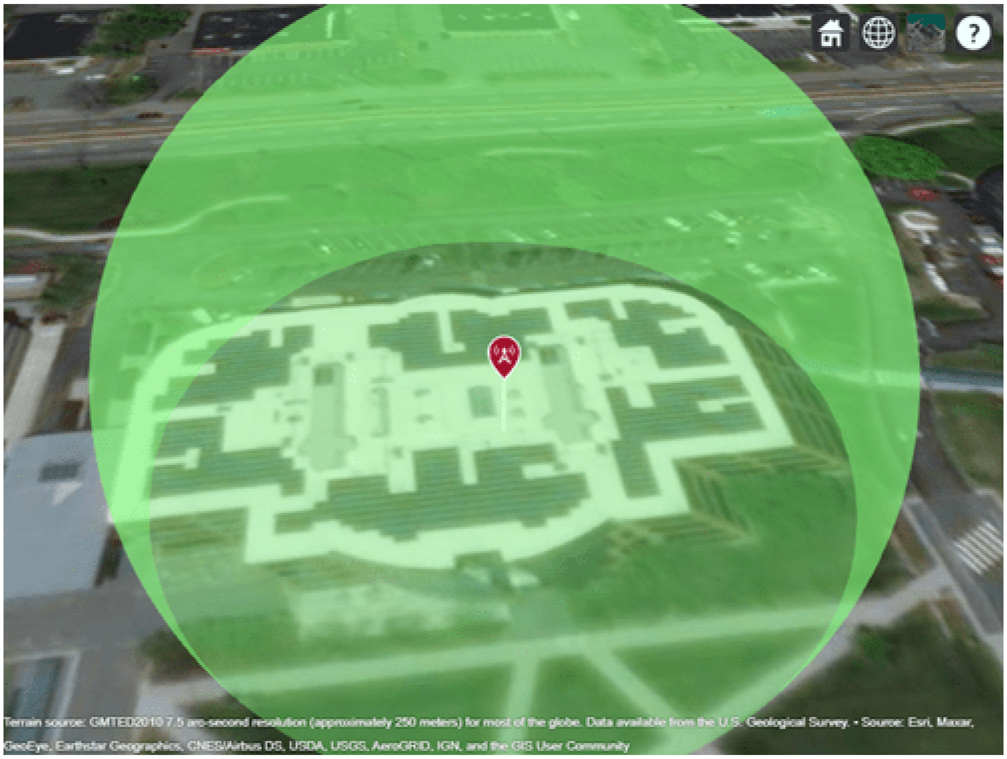
Create a transmitter site at a latitude of `42.3001` degrees and a longitude of `-71.3504` degrees.

```
tx = txsite("Name","MathWorks Apple Hill", ...
    "Latitude",42.3001,"Longitude",-71.3504)
```

```
tx = 
  txsite with properties:

                   Name: 'MathWorks Apple Hill'
       CoordinateSystem: 'geographic'
               Latitude: 42.3001
              Longitude: -71.3504
                Antenna: 'isotropic'
           AntennaAngle: 0
          AntennaHeight: 10
             SystemLoss: 0
    TransmitterFrequency: 1.9000e+09
       TransmitterPower: 10
```

View the coverage of the site.

```
pattern(tx)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

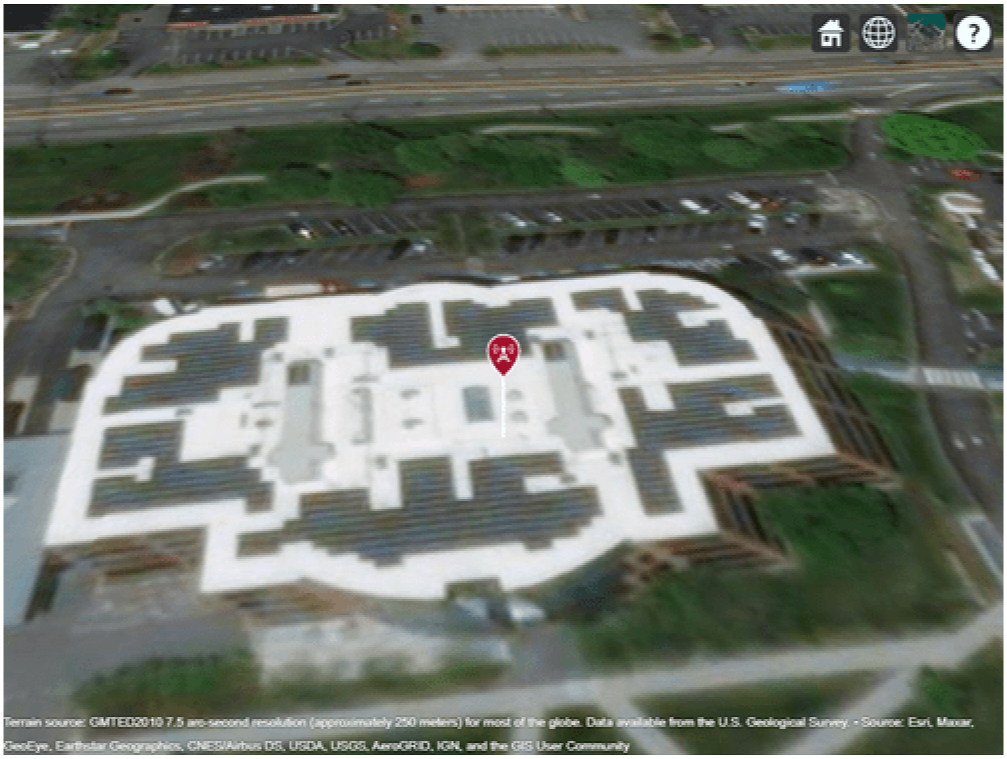
Transmitter Site Using Dipole Antenna

Create and view a transmitter site using a dipole antenna at a latitude of `42.3001` degrees and a longitude of `-71.3504` degrees.

```
fq = 2.5e9;
tx = txsite("Name","MathWorks Apple Hill", ...
    "Latitude",42.3001,"Longitude",-71.3504, ...
    "Antenna",design(dipole,fq),"TransmitterFrequency",fq)
```

```
tx = 
  txsite with properties:

                    Name: 'MathWorks Apple Hill'
        CoordinateSystem: 'geographic'
                Latitude: 42.3001
               Longitude: -71.3504
                 Antenna: [1×1 dipole]
            AntennaAngle: 0
           AntennaHeight: 10
              SystemLoss: 0
    TransmitterFrequency: 2.5000e+09
        TransmitterPower: 10
```

```
show(tx)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Transmitter Array Using Dipole Antenna

Specify the names, latitudes, and longitudes of three transmitter locations.

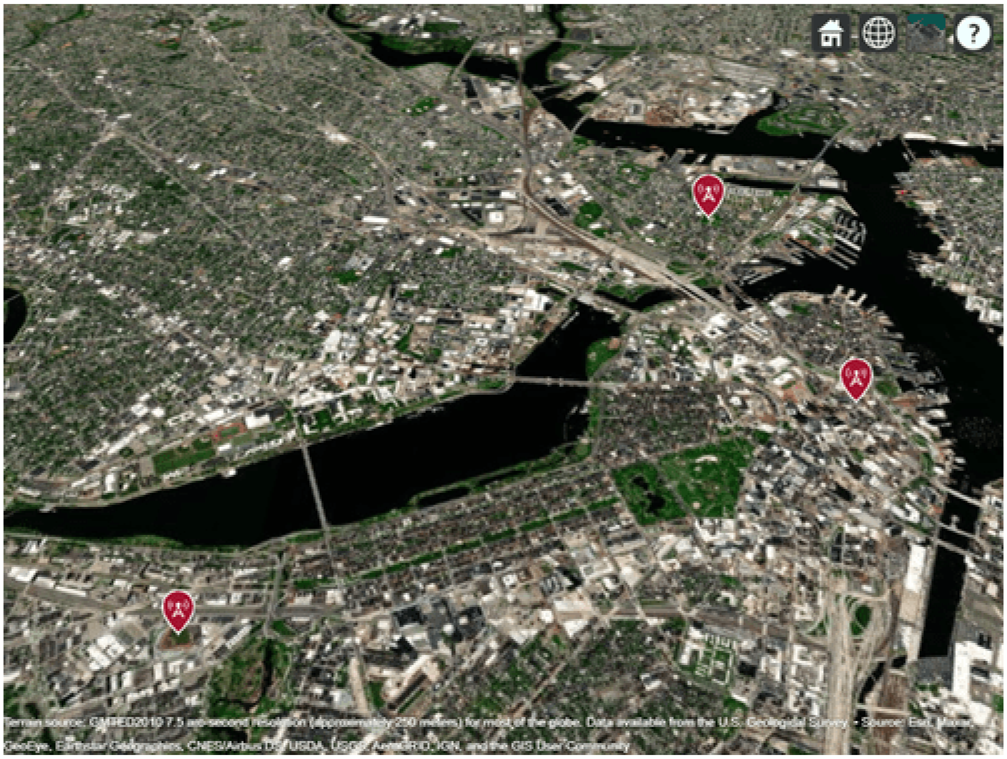
```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

Define the frequency of the transmitters.

```
fq = 2.5e9;
```

Create and view the transmitter array using a dipole antenna.

```
txs = txsite("Name",names,...
"Antenna",dipole,"Latitude",lats,...
"Longitude",lons, ...
"TransmitterFrequency",fq);
show(txs)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2017b

See Also

siteviewer | rxsite

rxsite

Create RF receiver site

Description

Use the `rxsite` object to create a radio frequency receiver site.

A receiver consists of an RF circuit and an antenna, where the antenna intercepts radio waves and converts them to a current that is decoded by the RF circuit (e.g. demodulated) into a signal. Key characteristics of a receiver include its sensitivity and its antenna radiation pattern.

Creation

Syntax

```
rx = rxsite
rx = rxsite(coordsys)
rx = rxsite(Name,Value)
```

Description

`rx = rxsite` creates a radio frequency receiver site.

`rx = rxsite(coordsys)` creates a receiver site with coordinate system set to `'geographic'` or `'cartesian'`.

`rx = rxsite(Name,Value)` sets properties using one or more name-value pairs. For example, `rx = rxsite('Name','RX Site')` creates a receiver site with name `RX Site`. Enclose each property name in quotes.

Create a 1-by-*N* array of receiver sites by specifying a property value as an array of *N* columns. Other property values must be specified with either 1 or *N* columns. The `Name`, `Latitude`, and `Longitude` properties may be specified as either a row vector or column vector with *N* elements. The `CoordinateSystem` property must be a string scalar or a character vector.

Properties

Name — Site name
character vector | string | row or column vector

Site name, specified as a character vector or as a row or column vector or as a string.

Example: `'Name','Site 3'`

Example: `rx.Name = 'Site 3'`

Example: If you want to assign multiple values then - `names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"]; rx = rxsite('Name',names)`

Data Types: `char` | `string`

`CoordinateSystem` — Coordinate system of site location

`'geographic'` (default) | `'cartesian'`

Coordinate system of the site location, specified as `'geographic'` or `'cartesian'`. If this property is `'geographic'`, the site location is defined using the properties `Latitude`, `Longitude`, and `AntennaHeight`. If this property is `'cartesian'`, the site location is defined using `AntennaPosition`.

Example: `'CoordinateSystem','cartesian'`

Example: `tx.CoordinateSystem = 'cartesian'`

`Latitude` — Site latitude coordinates

`42.3021` (default) | numeric scalar | row or column vector

Site latitude coordinates, specified as a numeric scalar or a row or column vector in the range of range `-90` to `90`. Coordinates are defined using Earth ellipsoid model WGS-84. Latitude is the north/south angle.

Example: `'Latitude',45.098`

Example: `rx.Latitude = 45.098`

Example: If you want to assign multiple values then - `latitude = [42.3467,42.3598,42.3763]; rx = rxsite('Latitude',latitude)`

Dependencies

To use this property, `CoordinateSystem` must be set to `'geographic'`.

`Longitude` — Site longitude coordinates

`-71.3764` (default) | numeric scalar | row or column vector

Site longitude coordinates, specified as a numeric scalar or a row or column vector. Coordinates are defined using Earth ellipsoid model WGS-84. Longitude is the east/west angle.

Example: `'Longitude',-68.890`

Example: `rx.Longitude = -68.890`

Example: If you want to assign multiple values then - `longitude = [-71.0972,-71.0545,-71.0611]; rx = rxsite('Longitude',longitude)`

Dependencies

To use this property, `CoordinateSystem` must be set to `'geographic'`.

`Antenna` — Antenna element or array

`'isotropic'` (default) | object | row vector

Antenna element or array specified as one of these:

- `'isotropic'` to model an antenna that radiates uniformly in all directions.
- An antenna element from the “Antenna Catalog” or array elements from the “Array Catalog”.

> **Note** When using antenna elements, please use the `design` function to design the antenna at the required receive frequency. Then add this antenna element to the receiver site.

- If you have Communications Toolbox, an `arrayConfig` object.
- If you have Phased Array System Toolbox, any antenna object in “Antennas, Microphones, and Sonar Transducers” (Phased Array System Toolbox) or any array object in “Array Geometries and Analysis” (Phased Array System Toolbox).

Example: `'Antenna'`,`monopole`

Example: `rx.Antenna = monopole`

`AntennaAngle` — Angle of antenna local X-axis

`0` (default) | numeric scalar | 2-by-1 vector | 2-by-*N* matrix

Angle of antenna local Cartesian coordinate system X-axis, specified as a numeric scalar representing azimuth angle in degrees or a 2-by-1 vector representing both azimuth and elevation angles with each element unit in degrees.

The azimuth angle is measured counterclockwise to the antenna X-axis, either from the east (for geographical sites) or from the global X-axis around the global Z-axis (for Cartesian sites).

The elevation angle is measured from the horizontal plane or X-Y plane to the antenna X-axis in the range `-90` to `90` degrees.

Example: `'AntennaAngle'`,`25`

Example: `tx.AntennaAngle = [25,-80]`

`AntennaHeight` — Antenna height above surface

`1` (default) | non-negative numeric scalar | row vector

Antenna height from the ground or building surface, specified as a non-negative numeric scalar in meters. Maximum value for this property is 6,371,000 m.

If the site coincides with the building, the height is measured from the top of the building to the center of the antenna. Otherwise,the height is measured from ground elevation to the center of the antenna.

Example: `'AntennaHeight'`,`25`

Example: `rx.AntennaHeight = 15`

Dependencies

To use this property, `CoordinateSystem` must be set to `'geographic'`.

Data Types:

`AntennaPosition` — Position of antenna center

`[0;0;0]` (default) | 3-by-1 vector

Position of the antenna center, specified as a 3-by-1 vector representing [*x;y;z*] Cartesian coordinates with each element in meters.

Example: `'AntennaPosition'`,`[0;2;4]`

Example: `tx.AntennaPosition = [0;2;4]`

Dependencies

To use this property, choose `CoordinateSystem` must be set to `'cartesian'`.

Data Types:

`SystemLoss` — System loss
`0` (default) | nonnegative numeric scalar | row vector

System loss, specified as a non-negative numeric scalar or a row vector in dB.

System loss includes transmission line loss and any other miscellaneous system losses.

Example: `'SystemLoss',10`

Example: `rx.SystemLoss = 10`

Data Types:

`ReceiverSensitivity` — Minimum received power to detect signal
`-100` (default) | numeric scalar | row vector

Minimum received power to detect the signal, specified as a numeric scalar or a row vector in dBm.

Example: `'ReceiverSensitivity',-80`

Example: `rx.ReceiverSensitivity = -80`

Data Types: `double`

Object Functions

show	Show site in Site Viewer
hide	Hide site from Site Viewer
distance	Distance between sites
angle	Angle between sites
elevation	Elevation of site
location	Coordinates at distance and angle from site
sigstrength	Received signal strength
los	Display or compute line-of-sight (LOS) visibility status
link	Display or compute communication link status
pattern	Display antenna radiation pattern in Site Viewer

Examples

Default Receiver Site

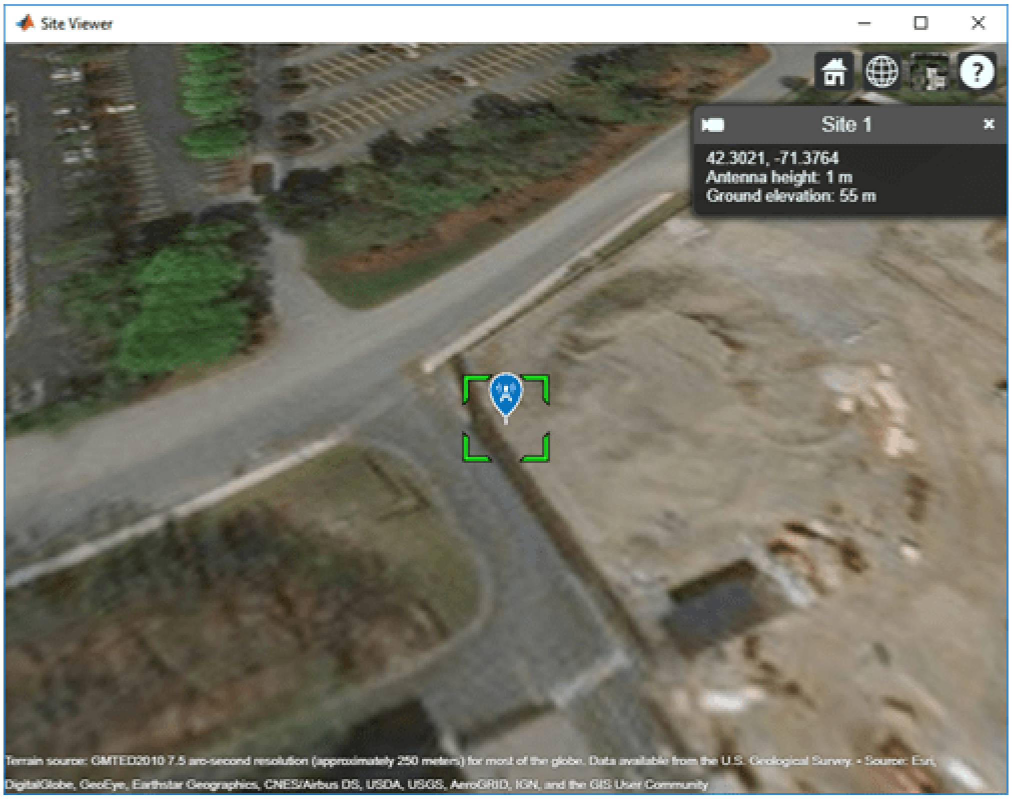
Create and show the default receiver site.

```
rx = rxsite
```

```
rx = 
  rxsite with properties:

                   Name: 'Site 2'
               Latitude: 42.3021
              Longitude: -71.3764
                Antenna: 'isotropic'
           AntennaAngle: 0
          AntennaHeight: 1
             SystemLoss: 0
```

```
    ReceiverSensitivity: -100
```

```
show(rx)
```



42.3021, -71.3764
Antenna height: 1 m
Ground elevation: 55 m

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Receiver Array Site and Coverage Using Dipole Antenna

Create and show a 1-by-3 receiver site array using dipole antenna.

Define names and locations of the sites around Boston.

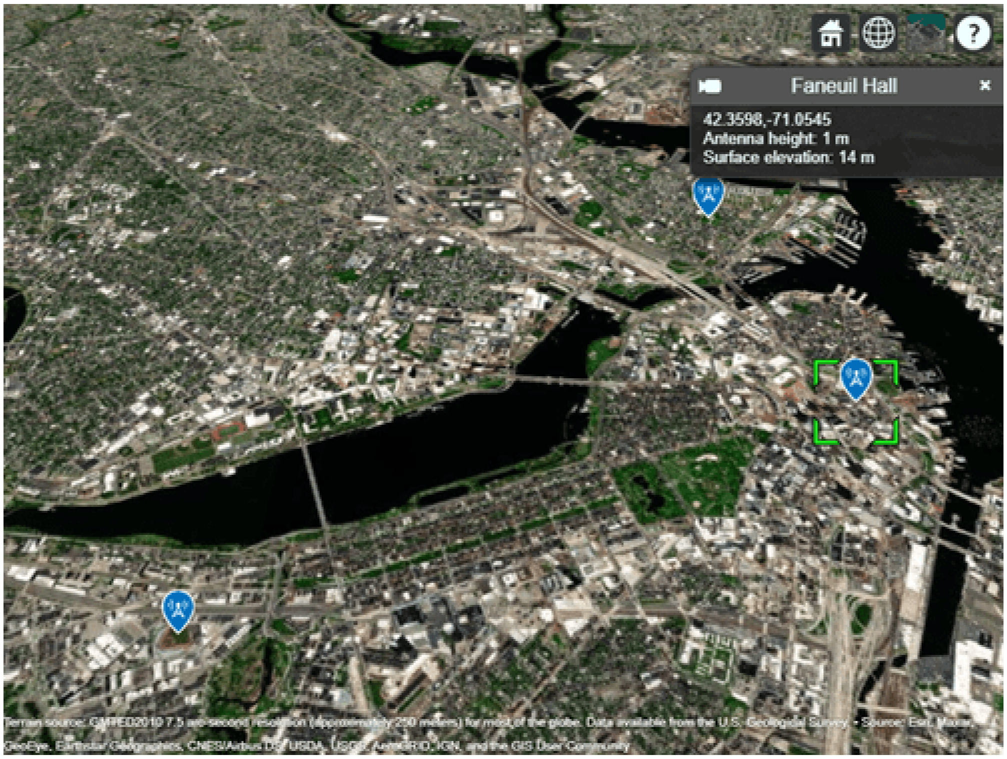
```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

Define the sensitivity of the receivers.

```
sens = -90;
```

Create and show the receiver site array.

```
rxs = rxsite("Name",names, ...
      "Antenna",dipole,"Latitude",lats, ...
      "Longitude",lons, ...
      "ReceiverSensitivity",sens);
show(rxs)
```



Faneuil Hall

42.3598,-71.0545
Antenna height: 1 m
Surface elevation: 14 m

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2017b

See Also

`txsite` | `siteviewer`

propagationData

Create RF propagation data from measurements

Description

Use the `propagationData` object to import and visualize geolocated propagation data. The measurement data can be path loss data, signal strength measurements, signal-to-noise-ratio (SNR) data, or cellular information.

Creation

Syntax

```
pd = propagationData(filename)
pd = propagationData(table)

pd = propagationData(latitude,longitude,varname,varvalue)
pd = propagationData(___,Name,Value)
```

Description

`pd = propagationData(filename)` creates a propagation data object by reading data from a file specified by `filename`.

`pd = propagationData(table)` creates a propagation data container object from a table object specified by `table`.

`pd = propagationData(latitude,longitude,varname,varvalue)` creates a propagation data container object using `latitude` and `longitude` coordinates with data specified using `varname` and `varvalue`.

`pd = propagationData(___,Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes.

Input Arguments

`filename` — Name of file containing propagation data
character vector | string scalar

Name of the file containing propagation data, specified as a character vector or a string scalar. The file must be in the current directory, in a directory on the MATLAB path, or be specified using a full or relative path. The file must be compatible with the `readtable` function. Call the `readtable` function if customized parameters are required to import the file and then pass the `table` object to the `propagationData` object.

Propagation data in the file must have one variable corresponding to the latitude values, one variable corresponding to longitude values, and at least one variable containing numeric data.

Data Types: `string` | `char`

table — Table containing propagation data
table object

Table containing propagation data, specified as a table object.

Propagation data in the file must have one variable corresponding to the latitude values, one variable corresponding to longitude values, and at least one variable containing numeric data.

Data Types: table

latitude — Latitude coordinate values
vector

Latitude coordinate values, specified as a vector in decimal degrees with reference to Earth's ellipsoid model WGS-84. The latitude coordinates must be in the range [-90 90].

Data Types: double

longitude — Longitude coordinate values
vector

Longitude coordinate values, specified as a vector in decimal degrees with reference to earth's ellipsoid. model WGS-84.

Data Types: double

varname — Variable name
character vector | string scalar

Variable name, specified as a character vector or a string scalar. This variable name must correspond to the variable with numeric data other than latitude or longitude. The variable name and the corresponding values are stored as a column in the Data property table object.

Data Types: string | char

varvalue — Variable values
numeric vector

Variable values, specified as a numeric vector. The numeric vectors must be the same size as latitude and longitude. The variable name and corresponding values are stored as a column in the Data property table object.

Data Types: double

Output Arguments

pd — Propagation data
propagationData object

Propagation data, returned as a propagationData object.

Properties

Name — Propagation data name
'Propagation Data' (default) | character vector | string scalar

Propagation data name, specified as a character vector or string scalar.

Example: `'Name','propdata'`

Example: `pd.Name = 'propdata'`

Data Types: `char` | `string`

`Data` — Propagation data table

scalar table object

This property is read-only.

Propagation data table, specified as a scalar table object containing a column corresponding to latitude coordinates, a column corresponding to longitude coordinates, and one or more columns corresponding to associated propagation data.

Data Types: `table`

`DataVariableName` — Name of data variable to plot

character vector | string scalar

Name of the data variable to plot, specified as a character vector or string scalar corresponding to a variable name in the `Data` table used to create propagation data container object. The variable name must correspond to a variable with numeric data and cannot correspond to the latitude or longitude variables. The default value for this property is the name of the first numeric data variable name in the `Data` table that is not a latitude or longitude variable.

Data Types: `char` | `string`

Object Functions

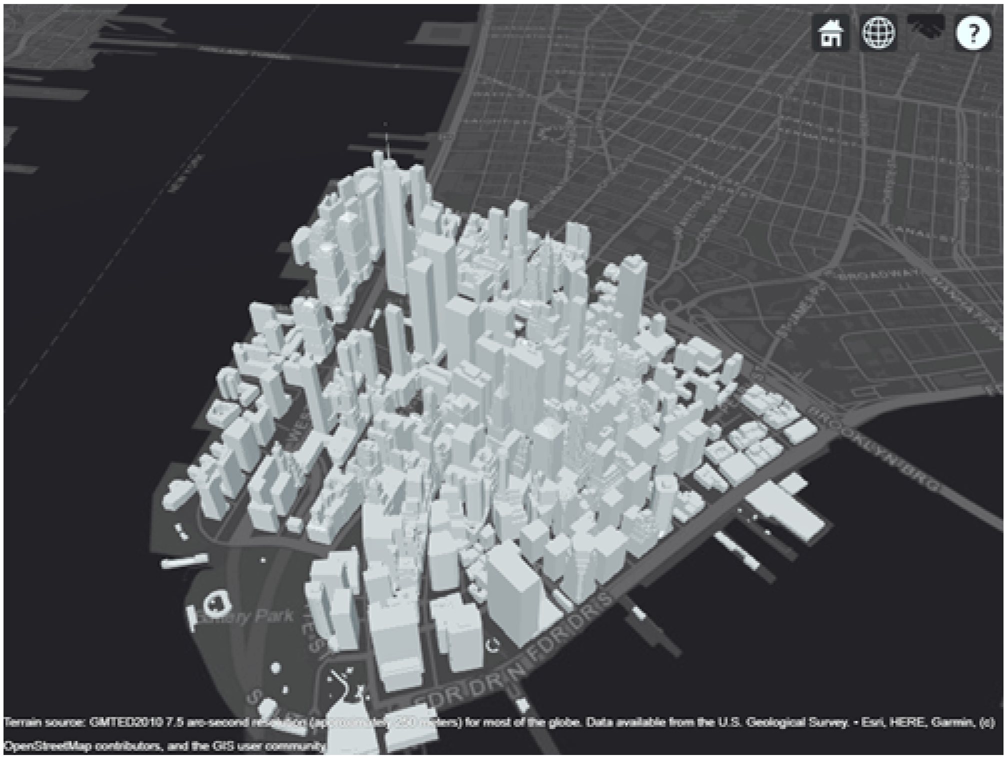
plot	Display RF propagation data in Site Viewer
contour	Display contour map of RF propagation data in Site Viewer
location	Coordinates of RF propagation data
getDataVariable	Get data variable values
interp	Interpolate RF propagation data

Examples

Compute Signal Strength Data in Urban Environment

Launch Site Viewer with basemaps and building files for Manhattan. For more information about the osm file, see [1] on page 6-38.

```
viewer = siteviewer("Basemap","streets_dark",...
        "Buildings","manhattan.osm");
```


Battery Park
FDR DR
BROOKLYN BRG
BROADWAY
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Esri, HERE, Garmin, (c) OpenStreetMap contributors, and the GIS user community

Show a transmitter site on a building.

```
tx = txsite("Latitude",40.7107,...
        "Longitude",-74.0114,...
        "AntennaHeight",80);
show(tx)
```

Washington Market Park
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Esri, HERE, Garmin, (c) OpenStreetMap contributors, and the GIS user community

Create receiver sites along nearby streets.

```
latitude = [linspace(40.7088, 40.71416, 50), ...
        linspace(40.71416, 40.715505, 25), ...
        linspace(40.715505, 40.7133, 25), ...
        linspace(40.7133, 40.7143, 25)]';
longitude = [linspace(-74.0108, -74.00627, 50), ...
        linspace(-74.00627 ,-74.0092, 25), ...
        linspace(-74.0092, -74.0110, 25), ...
        linspace(-74.0110, -74.0132, 25)]';
rxs = rxsite("Latitude", latitude, "Longitude", longitude);
```

Compute signal strength at each receiver location.

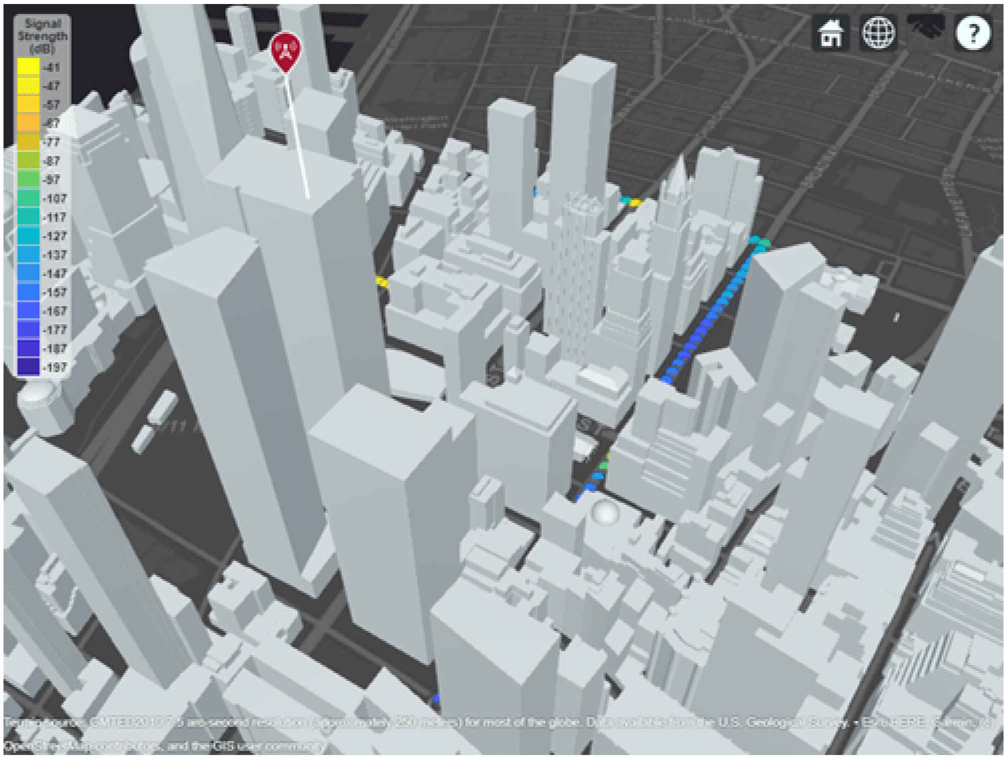
```
signalStrength = sigstrength(rxs, tx)';
```

Create a `propagationData` object to hold computed signal strength data.

```
tbl = table(latitude, longitude, signalStrength);
pd = propagationData(tbl);
```

Plot the signal strength data on a map as colored points.

```
legendTitle = "Signal" + newline + "Strength" + newline + "(dB)";
plot(pd, "LegendTitle", legendTitle, "Colormap", parula);
```


Signal Strength (dB)
-41
-47
-57
-67
-77
-87
-97
-107
-117
-127
-137
-147
-157
-167
-177
-187
-197
Terrain Sources: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Esri, HERE, Garmin, (c) OpenStreetMap contributors, and the GIS user community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

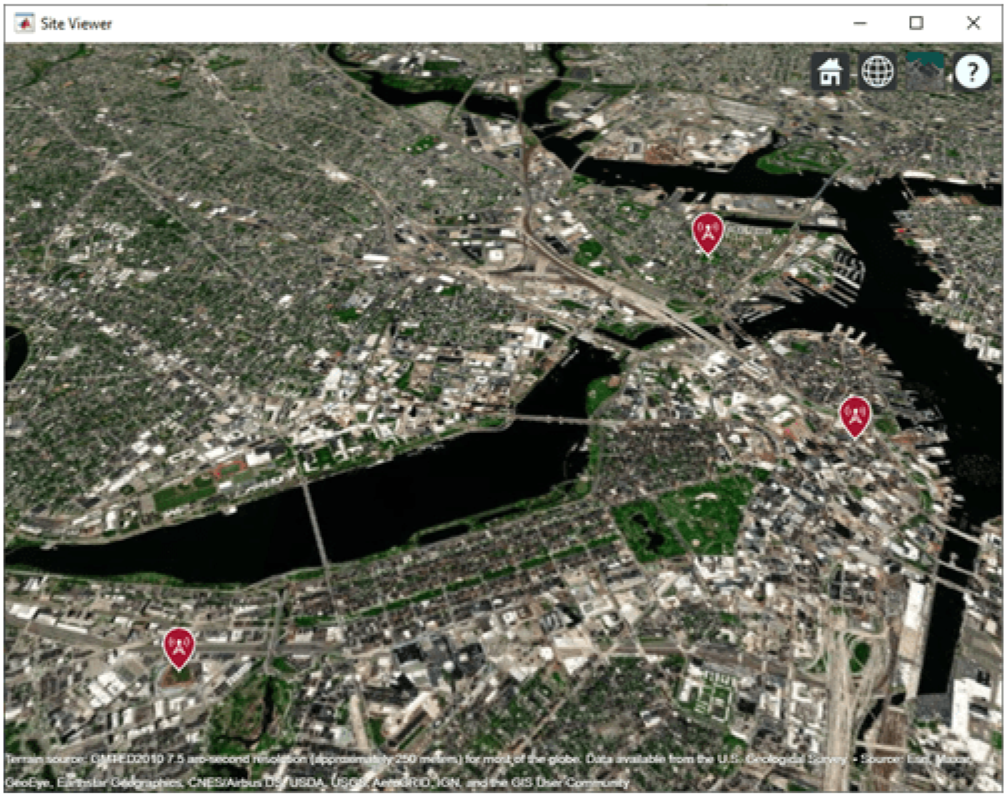
Capacity Map Using SINR Data

Define names and locations of sites around Boston.

```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

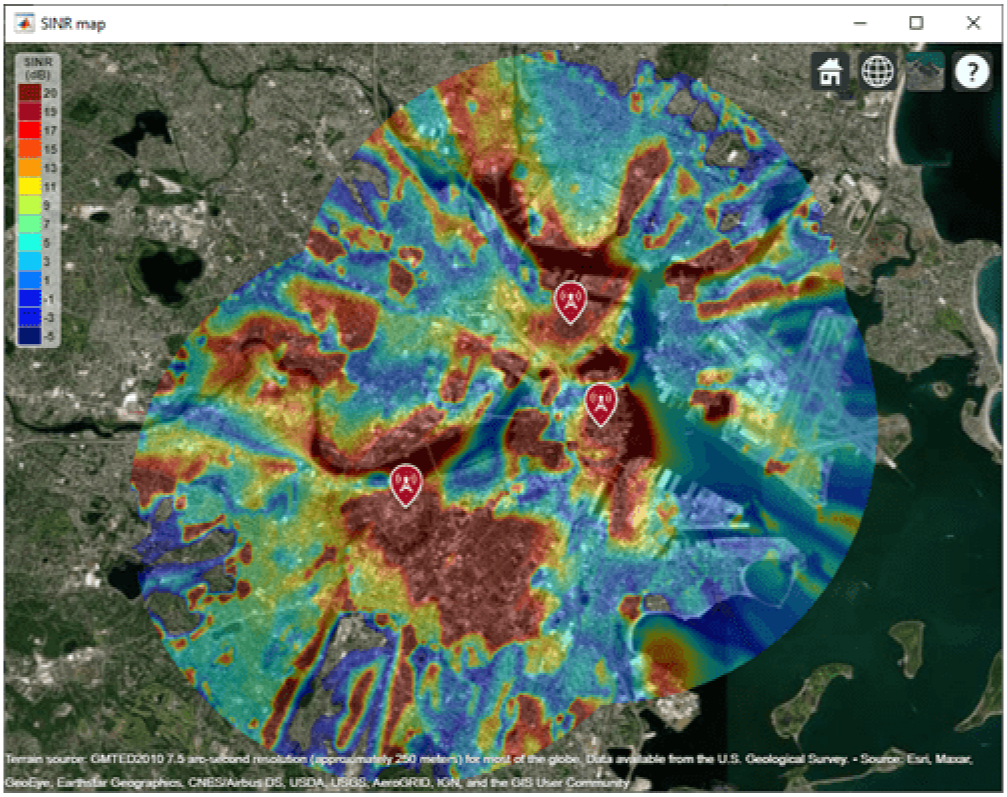
Create an array of transmitter sites.

```
txs = txsite("Name",names,...
       "Latitude",lats,...
       "Longitude",lons, ...
       "TransmitterFrequency",2.5e9);
show(txs)
```


Site Viewer
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create a signal-to-interference-plus-noise-ratio (SINR) map, where signal source for each location is selected as the transmitter site with the strongest signal.

```
sv1 = siteviewer("Name","SINR map");
sinr(txs,"MaxRange",5000)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Return SINR propagation data.

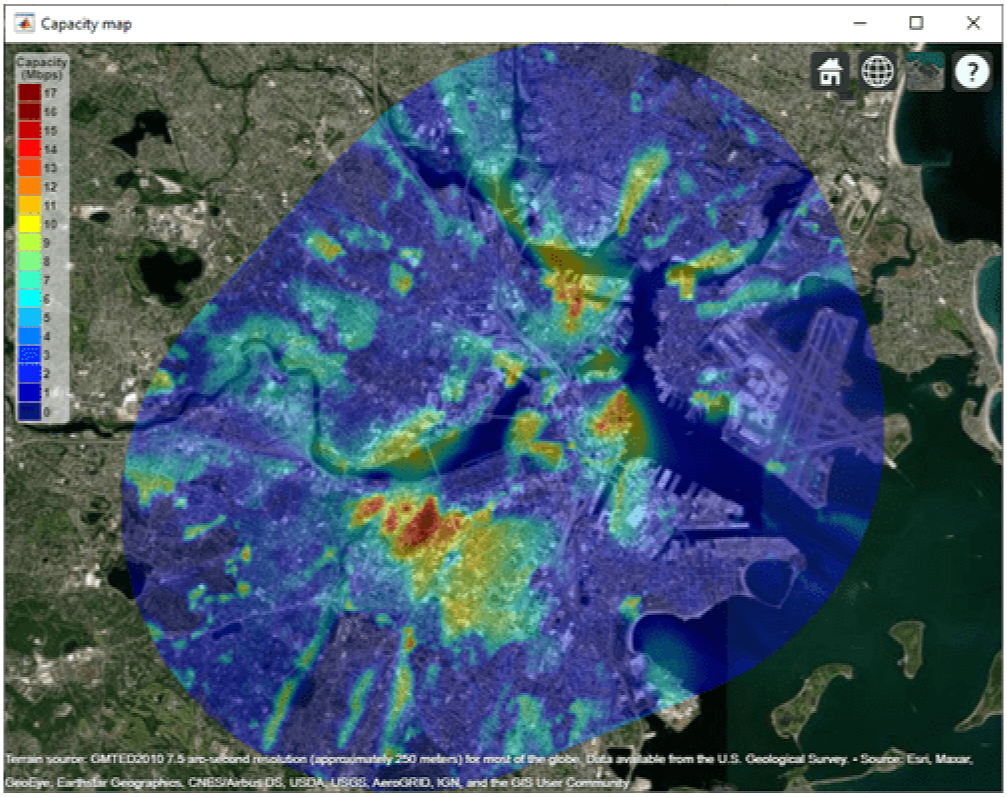
```
pd = sinr(txs,"MaxRange",5000);
[sinrDb,lats,lons] = getDataVariable(pd,"SINR");
```

Compute capacity using the Shannon-Hartley theorem.

```
bw = 1e6; % Bandwidth is 1 MHz
sinrRatio = 10.^(sinrDb./10); % Convert from dB to power ratio
capacity = bw*log2(1+sinrRatio)/1e6; % Unit: Mbps
```

Create new propagation data for the capacity map and display the contour plot.

```
pdCapacity = propagationData(lats,lons,"Capacity",capacity);
sv2 = siteviewer("Name","Capacity map");
legendTitle = "Capacity" + newline + "(Mbps)";
contour(pdCapacity,"LegendTitle",legendTitle);
```


Capacity map
Capacity (Mbps)
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Version History

Introduced in R2020a

See Also

`txsite` | `siteviewer` | `rxsite` | `readtable`

fogpl

RF signal attenuation due to fog and clouds

Syntax

```
L = fogpl(R,freq,T,den)
```

Description

`L = fogpl(R,freq,T,den)` returns attenuation, `L`, when signals propagate in fog or clouds. `R` represents the signal path length. `freq` represents the signal carrier frequency, `T` is the ambient temperature, and `den` specifies the liquid water density in the fog or cloud.

The `fogpl` function applies the International Telecommunication Union (ITU) cloud and fog attenuation model to calculate path loss of signals propagating through clouds and fog. See [1] (Phased Array System Toolbox). Fog and clouds are the same atmospheric phenomenon, differing only by height above ground. Both environments are parametrized by their liquid water density. Other model parameters include signal frequency and temperature. This function applies to cases when the signal path is contained entirely in a uniform fog or cloud environment. The liquid water density does not vary along the signal path. The attenuation model applies only for frequencies at 10–1000 GHz.

Examples

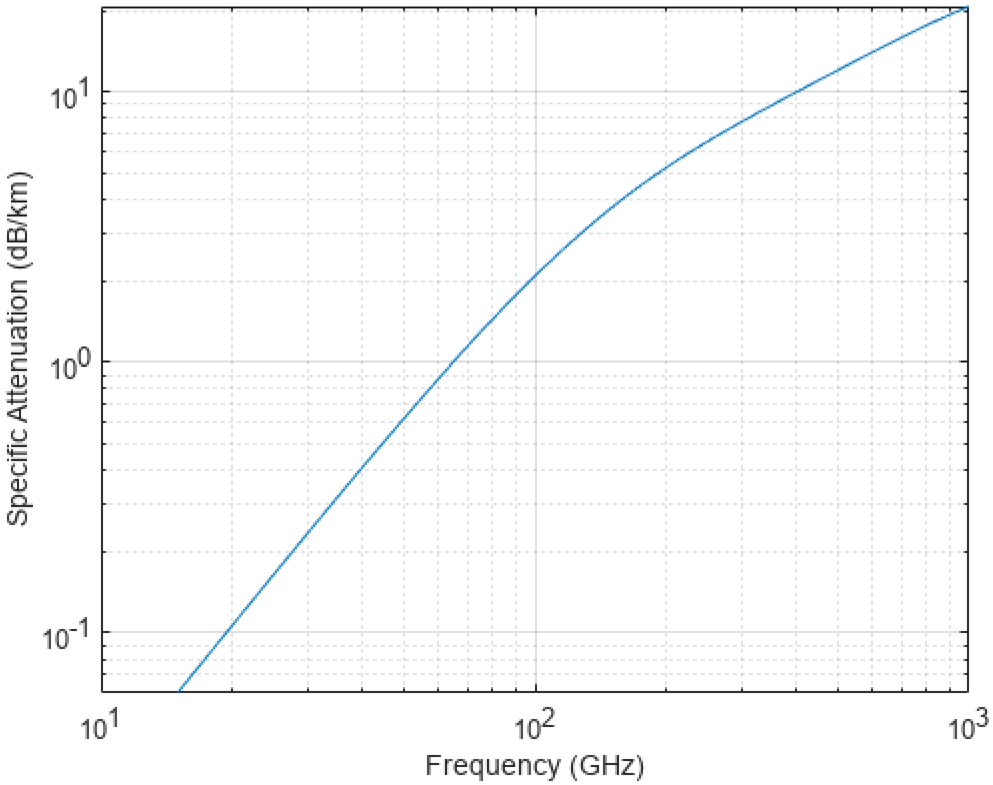
Attenuation in Cumulus Clouds

Compute the attenuation of signals propagating through a cloud that is 1 km long at 1000 meters altitude. Compute the attenuation for frequencies from 15 to 1000 GHz. A typical value for the cloud liquid water density is 0.5 g/m^3. Assume the atmospheric temperature at 1000 meters is 20˚C.

```
R = 1000.0;
freq = [15:5:1000]*1e9;
T = 20.0;
lwd = 0.5;
L = fogpl(R,freq,T,lwd);
```

Plot the specific attenuation as a function of frequency. Specific attenuation is the attenuation or loss per kilometer.

```
loglog(freq/1e9,L)
grid
xlabel('Frequency (GHz)')
ylabel('Specific Attenuation (dB/km)')
```



Input Arguments

R — Signal path length

positive real-valued scalar | *M*-by-1 nonnegative real-valued vector | 1-by-*M* nonnegative real-valued vector

Signal path length, specified as a scalar or as an *M*-by-1 or 1-by-*M* vector of nonnegative real-values. Total attenuation is the specific attenuation multiplied by the path length. Units are meters.

Example: `[1300.0,1400.0]`

`freq` — Signal frequency

positive real-valued scalar | *N*-by-1 nonnegative real-valued column vector | 1-by-*N* nonnegative real-valued row vector

Signal frequency, specified as a positive real-valued scalar or as an *N*-by-1 nonnegative real-valued vector or 1-by-*N* nonnegative real-valued vector. Frequencies must lie in the range 10–1000 GHz. Units are in Hz.

Example: `[14.0e9,15.0e9]`

T — Ambient temperature

real-valued scalar

Ambient temperature in fog or cloud, specified as a real-valued scalar. Units are in degrees Celsius.

Example: `-10.0`

den — Liquid water density
nonnegative real-valued scalar

Liquid water density, specified as a nonnegative real-valued scalar. Units are g/m^3. Typical values for liquid water density in fog range from approximately 0.05 g/m^3 for medium fog to approximately 0.5 g/m^3 for thick fog. For medium fog, visibility is about 300 meters. For heavy fog, visibility is about 50 meters. Cumulus cloud liquid water density is typically 0.5 g/m^3.

Example: `0.01`

Output Arguments

L — Signal attenuation
real-valued *M*-by-*N* matrix

Signal attenuation, returned as a real-valued *M*-by-*N* matrix. Each matrix row represents a different path where *M* is the number of paths. Each column represents a different frequency where *N* is the number of frequencies. Units are in dB.

More About

Fog and Cloud Attenuation Model

This model calculates the attenuation of signals that propagate through fog or clouds.

Fog and cloud attenuation are the same atmospheric phenomenon. The ITU model, *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog* is used. The model computes the specific attenuation (attenuation per kilometer), of a signal as a function of liquid water density, signal frequency, and temperature. The model applies to polarized and nonpolarized fields. The formula for specific attenuation at each frequency is

$$\gamma_c = K_l(f)M,$$

where M is the liquid water density in gm/m^3. The quantity $K_l(f)$ is the specific attenuation coefficient and depends on frequency. The cloud and fog attenuation model is valid for frequencies 10–1000 GHz. Units for the specific attenuation coefficient are (dB/km)/(g/m^3).

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length R. Total attenuation is $L_c = R\gamma_c$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply narrowband attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

References

[1] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog*. 2013.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Does not support variable-size inputs.

See Also

fspl

Free space path loss

Syntax

```
L = fspl(R,lambda)
```

Description

`L = fspl(R,lambda)` returns the free space path loss in decibels for a waveform with wavelength `lambda` propagated over a distance of `R` meters. The minimum value of `L` is zero, indicating no path loss.

Examples

Calculate Free-Space Path Loss

Calculate the free-space path loss (in dB) of a 10 GHz radar signal over a distance of 10 km.

```
fc = 10.0e9;
lambda = physconst('LightSpeed')/fc;
R = 10e3;
L = fspl(R,lambda)
```

```
L = 132.4478
```

Input Arguments

R — Propagation distance of signal
real-valued 1-by-*M* or *M*-by-1 vector

Units are in meters.

lambda — Speed of propagation divided by the signal frequency
real-valued 1-by-*N* or *N*-by-1 vector

Wavelength units are meters.

Output Arguments

L — Path loss in decibels
M-by-*N* non-negative matrix. A value of zero signifies no path loss.

When `lambda` is a scalar, `L` has the same dimensions as `R`.

More About

Free Space Path Loss

The free-space path loss, *L*, in decibels is:

$$L = 20\log_{10}(\frac{4\pi R}{\lambda})$$

This formula assumes that the target is in the far-field of the transmitting element or array. In the near-field, the free-space path loss formula is not valid and can result in a loss smaller than 0 dB, equivalent to a signal gain. For this reason, the loss is set to 0 dB for range values $R \le \lambda/4\pi$.

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Does not support variable-size inputs.

See Also

gaspl

RF signal attenuation due to atmospheric gases

Syntax

```
L = gaspl(range,freq,T,P,den)
```

Description

`L = gaspl(range,freq,T,P,den)` returns the attenuation, `L`, of signals propagating through the atmosphere.

- `range` represents the signal path length.
- `freq` represents the signal carrier frequency.
- `T` represents the ambient temperature.
- `P` represents the atmospheric pressure.
- `den` represents the atmospheric water vapor density.

The `gaspl` function applies the International Telecommunication Union (ITU) atmospheric gas attenuation model [1] to calculate path loss for signals primarily due to oxygen and water vapor. The model computes attenuation as a function of ambient temperature, pressure, water vapor density, and signal frequency.

The function requires that the signal path is contained entirely in a homogeneous environment – temperature `T`, atmospheric pressure `P`, and water vapor density `den` do not vary along the signal path. You can account for the variation of atmospheric parameters with height using the `tropopl` and `atmositu` functions in the Radar Toolbox.

The attenuation model applies only for frequencies at 1–1000 GHz.

Examples

Atmospheric Gas Attenuation Spectrum

Compute the attenuation spectrum from 1 to 1000 GHz for an atmospheric pressure of 101.300 kPa and a temperature of 15°C. Plot the spectrum for a water vapor density of 7.5 g/m^3 and then plot the spectrum for dry air (zero water vapor density).

Set the attenuation frequencies.

```
freq = [1:1000]*1e9;
```

Assume a 1 km path distance.

```
R = 1000.0;
```

Compute the attenuation for air containing water vapor.

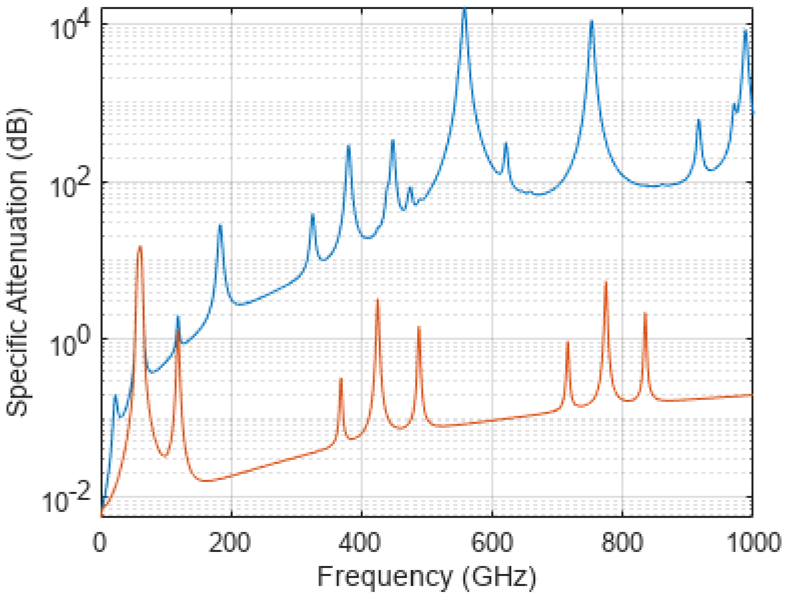
```
T = 15;
P = 101300.0;
W = 7.5;
L = gaspl(R,freq,T,P,W);
```

Compute the attenuation for dry air.

```
L0 = gaspl(R,freq,T,P,0.0);
```

Plot the attenuations.

```
semilogy(freq/1e9,L)
hold on
semilogy(freq/1e9,L0)
grid
xlabel('Frequency (GHz)')
ylabel('Specific Attenuation (dB)')
hold off
```



Plot Attenuation Due to Atmospheric Gases and Free Space

First, plot the specific attenuation of atmospheric gases for frequencies from 1 GHz to 1000 GHz. Assume a sea-level dry air pressure of 101.325e5 kPa and a water vapor density of 7.5 g/m^3. The air temperature is 20˚C. Specific attenuation is defined as dB loss per kilometer. Then, plot the actual attenuation at 10 GHz for a span of ranges.

Plot Specific Atmospheric Gas Attenuation

Set the atmosphere temperature, pressure, water vapor density.

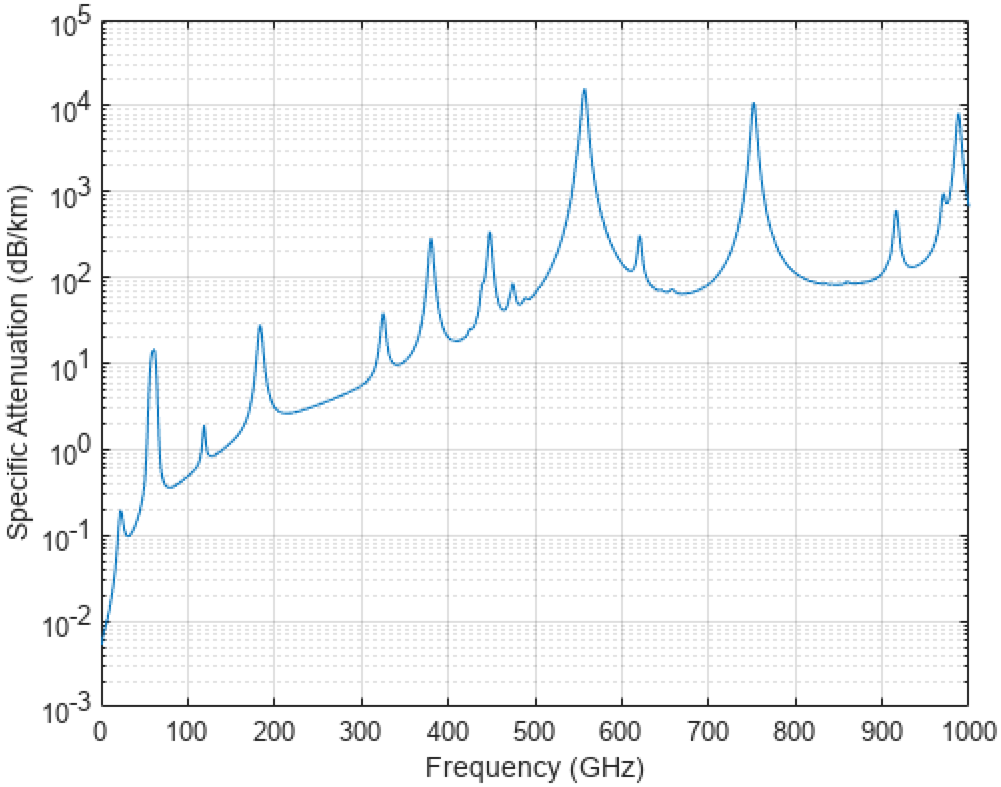
```
T = 20.0;
Patm = 101.325e3;
rho_wv = 7.5;
```

Set the propagation distance, speed of light, and frequencies.

```
km = 1000.0;
c = physconst('LightSpeed');
freqs = [1:1000]*1e9;
```

Compute and plot the atmospheric gas loss.

```
loss = gaspl(km,freqs,T,Patm,rho_wv);
semilogy(freqs/1e9,loss)
grid on
xlabel('Frequency (GHz)')
ylabel('Specific Attenuation (dB/km)')
```

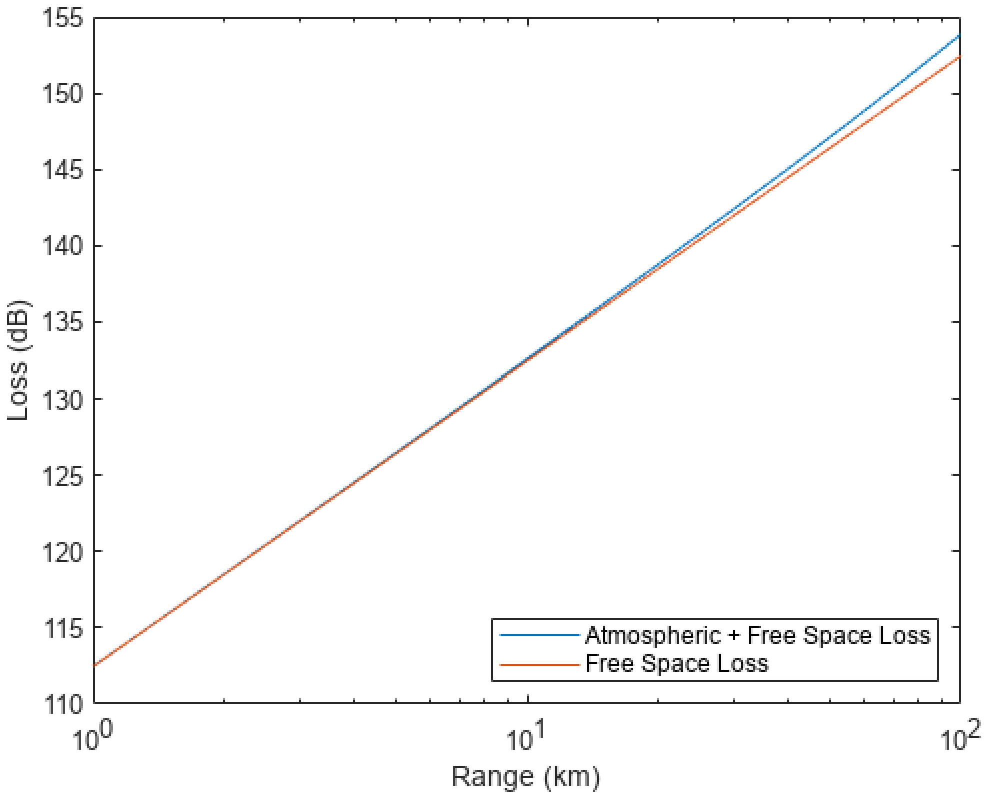


Plot Actual Atmospheric and Free Space Attenuation

Compute both free space loss and atmospheric gas loss at 10 GHz for ranges from 1 to 100 km. The frequency corresponds to an *X*-band radar. Then, plot the free space loss and the total (atmospheric + free space) loss.

```
ranges = [1:100]*1000;
freq_xband = 10e9;
loss_gas = gaspl(ranges,freq_xband,T,Patm,rho_wv);
lambda = c/freq_xband;
```

```
loss_fsp = fspl(ranges,lambda);
semilogx(ranges/1000,loss_gas + loss_fsp.',ranges/1000,loss_fsp)
legend('Atmospheric + Free Space Loss','Free Space Loss','Location','SouthEast')
xlabel('Range (km)')
ylabel('Loss (dB)')
```



Input Arguments

`range` — Signal path length

nonnegative real-valued scalar | *M*-by-1 nonnegative real-valued column vector | 1-by-*M* nonnegative real-valued row vector

Signal path length used to compute attenuation, specified as a nonnegative real-valued scalar or vector. You can specify multiple path lengths simultaneously. Units are in meters.

Example: `[13000.0,14000.0]`

`freq` — Signal frequency

positive real-valued scalar | *N*-by-1 nonnegative real-valued column vector | 1-by-*N* nonnegative real-valued row vector

Signal frequency, specified as a positive real-valued scalar, or as an *N*-by-1 nonnegative real-valued vector or 1-by-*N* nonnegative real-valued vector. You can specify multiple frequencies simultaneously. Frequencies must lie in the range 1–1000 GHz. Units are in hertz.

Example: `[1.4e9,2.0e9]`

T — Ambient temperature
real-valued scalar

Ambient temperature, specified as a real-valued scalar. Units are in degrees Celsius.

Example: `-10.0`

P — Dry air pressure
positive real-valued scalar

Dry air pressure, specified as a positive real-valued scalar. Units are in Pa. One standard atmosphere at sea level is 101325 Pa.

Example: `101300.0`

den — Water vapor density
nonnegative real-valued scalar

Water vapor density or absolute humidity, specified as a nonnegative real-valued scalar. Units are g/m^3. The maximum water vapor density of air at 30° C is approximately 30.0 g/m^3. The maximum water vapor density of air at 0°C is approximately 5.0 g/m^3.

Example: `4.0`

Output Arguments

L — Signal attenuation
real-valued *M*-by-*N* matrix

Signal attenuation, returned as a real-valued *M*-by-*N* matrix. Each matrix row represents a different path where *M* is the number of paths. Each column represents a different frequency where *N* is the number of frequencies. Units are in dB.

More About

Atmospheric Gas Attenuation Model

This model calculates the attenuation of signals that propagate through atmospheric gases.

Electromagnetic signals attenuate when they propagate through the atmosphere. This effect is due primarily to the absorption resonance lines of oxygen and water vapor, with smaller contributions coming from nitrogen gas. The model also includes a continuous absorption spectrum below 10 GHz. The ITU model *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* is used. The model computes the specific attenuation (attenuation per kilometer) as a function of temperature, pressure, water vapor density, and signal frequency. The atmospheric gas model is valid for frequencies from 1–1000 GHz and applies to polarized and nonpolarized fields.

The formula for specific attenuation at each frequency is

$$\gamma = \gamma_o(f) + \gamma_w(f) = 0.1820 f N''(f) \, .$$

The quantity *N"()* is the imaginary part of the complex atmospheric refractivity and consists of a spectral line component and a continuous component:

$$N''(f) = \sum_i S_i F_i + N''_D(f)$$

The spectral component consists of a sum of discrete spectrum terms composed of a localized frequency bandwidth function, *F(f)*$_i$, multiplied by a spectral line strength, S_i. For atmospheric oxygen, each spectral line strength is

$$S_i = a_1 \times 10^{-7}\left(\frac{300}{T}\right)^3 \exp\left[a_2(1-\left(\frac{300}{T}\right)\right]P\,.$$

For atmospheric water vapor, each spectral line strength is

$$S_i = b_1 \times 10^{-1}\left(\frac{300}{T}\right)^{3.5} \exp\left[b_2(1-\left(\frac{300}{T}\right)\right]W\,.$$

P is the dry air pressure, W is the water vapor partial pressure, and T is the ambient temperature. Pressure units are in hectoPascals (hPa) and temperature is in degrees Kelvin. The water vapor partial pressure, W, is related to the water vapor density, ρ, by

$$W = \frac{\rho T}{216.7}\,.$$

The total atmospheric pressure is $P + W$.

For each oxygen line, S_i depends on two parameters, a_1 and a_2. Similarly, each water vapor line depends on two parameters, b_1 and b_2. The ITU documentation cited at the end of this section contains tabulations of these parameters as functions of frequency.

The localized frequency bandwidth functions $F_i(f)$ are complicated functions of frequency described in the ITU references cited below. The functions depend on empirical model parameters that are also tabulated in the reference.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length, R. Then, the total attenuation is $L_g = R(\gamma_o + \gamma_w)$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

References

[1] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* 2013.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Does not support variable-size inputs.

See Also

rainpl

RF signal attenuation due to rainfall

Syntax

```
L = rainpl(range,freq,rainrate)
L = rainpl(range,freq,rainrate,elev)
L = rainpl(range,freq,rainrate,elev,tau)
L = rainpl(range,freq,rainrate,elev,tau,pct)
```

Description

`L = rainpl(range,freq,rainrate)` returns the signal attenuation, `L`, due to rainfall. In this syntax, attenuation is a function of signal path length, `range`, signal frequency, `freq`, and rain rate, `rainrate`. The path elevation angle and polarization tilt angles are assumed to zero.

The `rainpl` function applies the International Telecommunication Union (ITU) rainfall attenuation model to calculate path loss of signals propagating in a region of rainfall [1]. The function applies when the signal path is contained entirely in a uniform rainfall environment. Rain rate does not vary along the signal path. The attenuation model applies only for frequencies at 1–1000 GHz.

`L = rainpl(range,freq,rainrate,elev)` also specifies the elevation angle, `elev`, of the propagation path.

`L = rainpl(range,freq,rainrate,elev,tau)` also specifies the polarization tilt angle, `tau`, of the signal.

`L = rainpl(range,freq,rainrate,elev,tau,pct)` also specifies the specified percentage of time, `pct`. `pct` is a scalar in the range of 0.001–1, inclusive. The attenuation, `L`, is computed from a power law using the long-term statistical 0.01% rain rate (in mm/h).

Examples

Signal Attenuation Due to Rainfall

Compute the signal attenuation due to rainfall for a 20 GHz signal over a distance of 10 km in light and heavy rain.

Propagate the signal in a light rainfall of 1 mm/hr.

```
rr = 1.0;
L = rainpl(10000,20.0e9,rr)
```

```
L = 1.3009
```

Propagate the signal in a heavy rainfall of 10 mm/hr.

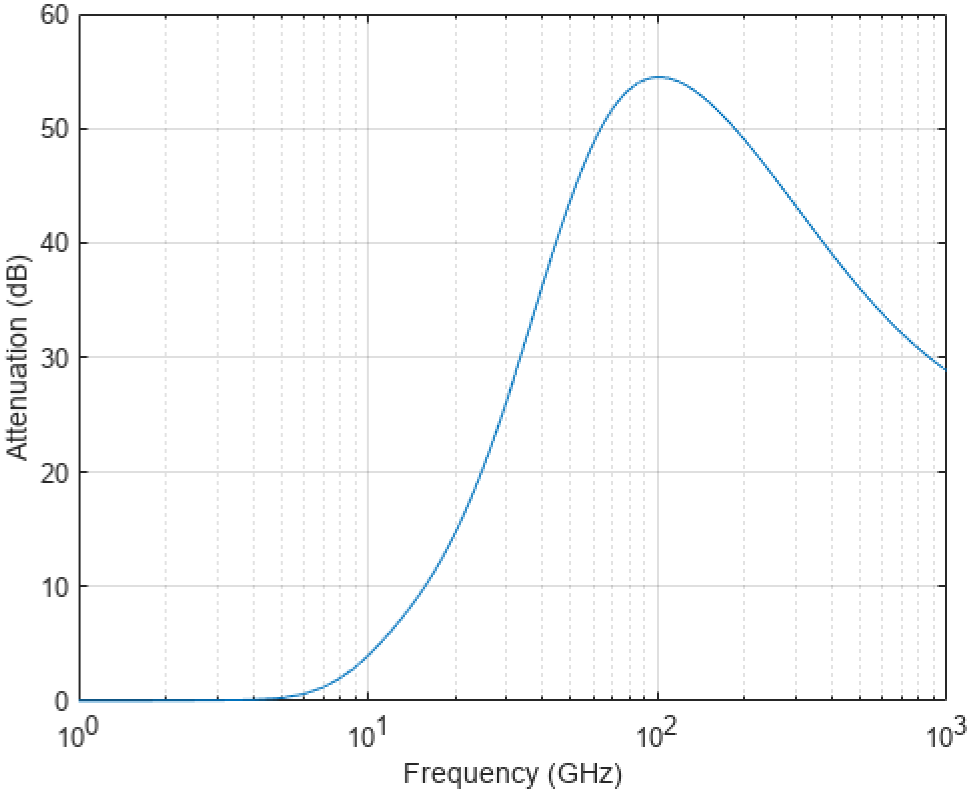
```
rr = 10.0;
L = rainpl(10000,20.0e9,rr)
```

```
L = 8.1584
```

Signal Attenuation Due to Rainfall as Function of Frequency

Plot the signal attenuation due to a 20 mm/hr statistical rainfall for signals in the frequency range from 1 to 1000 GHz. The path distance is 10 km.

```
rr = 20.0;
freq = [1:1000]*1e9;
L = rainpl(10000,freq,rr);
semilogx(freq/1e9,L)
grid
xlabel('Frequency (GHz)')
ylabel('Attenuation (dB)')
```



Signal Attenuation Due to Rainfall as Function of Elevation Angle

Compute the signal attenuation due to heavy rain as a function of elevation angle. Elevation angles vary from 0 to 90 degrees. Assume a path distance of 100 km and a signal frequency of 100 GHz.

Set the rain rate to 10 mm/hr.

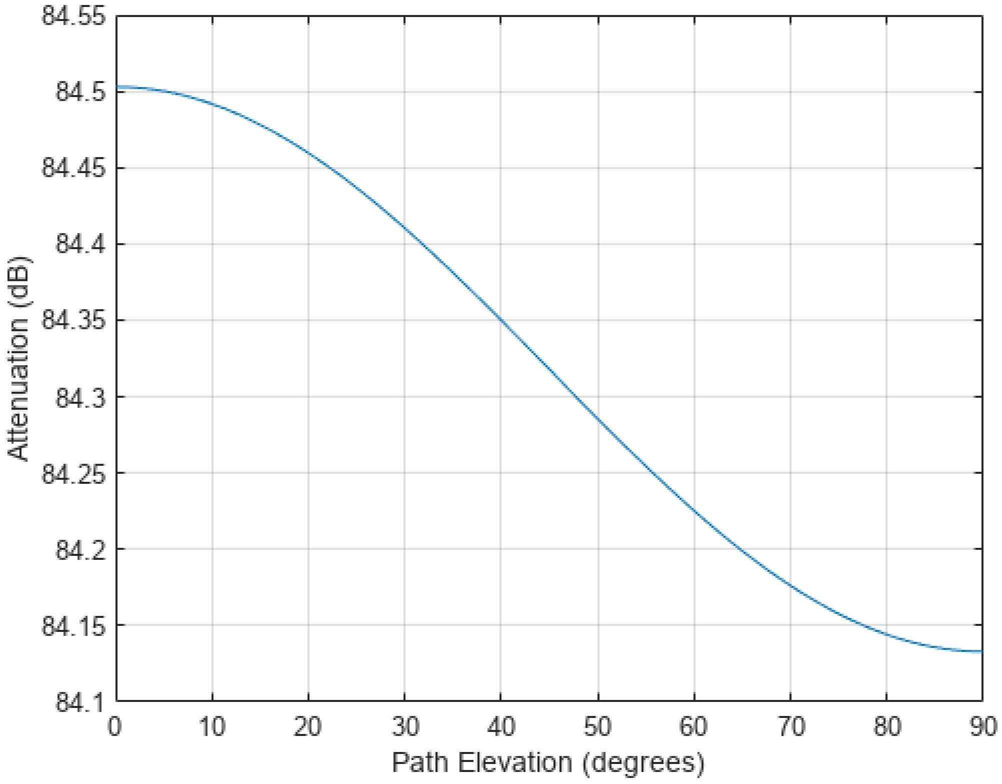
```
rr = 10.0;
```

Set the elevation angles, frequency, range.

```
elev = [0:1:90];
freq = 100.0e9;
rng = 100000.0*ones(size(elev));
```

Compute and plot the loss.

```
L = rainpl(rng,freq,rr,elev);
plot(elev,L)
grid
xlabel('Path Elevation (degrees)')
ylabel('Attenuation (dB)')
```



Signal Attenuation Due to Rainfall as Function of Polarization

Compute the signal attenuation due to heavy rainfall as a function of the polarization tilt angle. Assume a path distance of 100 km, a signal frequency of 100 GHz, and a path elevation angle of 0 degrees. Set the rainfall rate to 10 mm/hour. Plot the signal attenuation versus polarization tilt angle.

Set the polarization tilt angle to vary from -90 to 90 degrees.

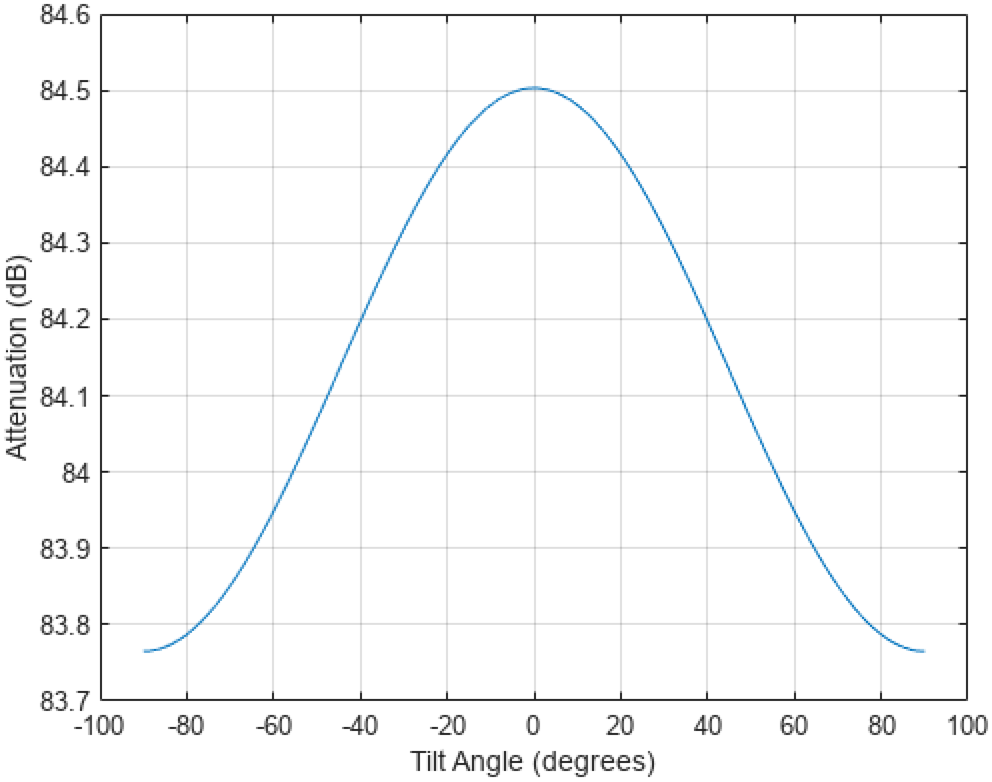
```
tau = -90:90;
```

Set the elevation angle, frequency, path distance, and rain rate.

```
elev = 0;
freq = 100.0e9;
rng = 100e3*ones(size(tau));
rr = 10.0;
```

Compute and plot the attenuation.

```
L = rainpl(rng,freq,rr,elev,tau);
plot(tau,L)
grid
xlabel('Tilt Angle (degrees)')
ylabel('Attenuation (dB)')
```



Input Arguments

range — Signal path length

nonnegative real-valued scalar | nonnegative real-valued *M*-by-1 column vector | nonnegative real-valued 1-by-*M* row vector

Signal path length, specified as a nonnegative real-valued scalar, or as a *M*-by-1 or 1-by-*M* vector. Units are in meters.

Example: `[13000.0,14000.0]`

freq — Signal frequency

positive real-valued scalar | nonnegative real-valued *N*-by-1 column vector | nonnegative real-valued 1-by-*N* row vector

Signal frequency, specified as a positive real-valued scalar, or as a nonnegative *N*-by-1 or 1-by-*N* vector. Frequencies must lie in the range 1–1000 GHz.

Example: `[1400.0e6,2.0e9]`

rainrate — Long-term statistical rain rate

nonnegative real-valued scalar

Long-term statistical rain rate, specified as a nonnegative real-valued scalar. The long-term statistical rain rate is the rain rate that is exceeded 0.01% of the time. You can adjust the percent of time using the `pct` argument. Units are in mm/hr.

Example: `1.5`

elev — Signal path elevation angle

0.0 (default) | real-valued scalar | real-valued *M*-by-1 column vector | real-valued 1-by-*M* row vector

Signal path elevation angle, specified as a real-valued scalar, or as an *M*-by-1 or 1-by- *M* vector. Units are in degrees between –90° and 90°. If `elev` is a scalar, all propagation paths have the same elevation angle. If `elev` is a vector, its length must match the dimension of `range` and each element in `elev` corresponds to a propagation range in `range`.

Example: `[0,45]`

tau — Tilt angle of polarization ellipse

0.0 (default) | real-valued scalar | real-valued *M*-by-1 column vector | real-valued 1-by-*M* row vector

Tilt angle of the signal polarization ellipse, specified as a real-valued scalar, or as an *M*-by-1 or 1-by-*M* vector. Units are in degrees between –90° and 90°. If `tau` is a scalar, all signals have the same tilt angle. If `tau` is a vector, its length must match the dimension of `range`. In that case, each element in `tau` corresponds to a propagation path in `range`.

The tilt angle is defined as the angle between the semi-major axis of the polarization ellipse and the *x*-axis. Because the ellipse is symmetrical, a tilt angle of 100° corresponds to the same polarization state as a tilt angle of -80°. Thus, the tilt angle need only be specified between ±90°.

Example: `[45,30]`

pct — Exceedance percentage of rainfall

`0.01` (default) | positive scalar between 0.001 and 1

Exceedance percentage of rainfall, specified as a positive scalar between 0.001 and 1. The long-term statistical rain rate is the rain rate that is exceeded `pct` of the time. Units are dimensionless.

Data Types: `double`

Output Arguments

L — Signal attenuation

real-valued *M*-by-*N* matrix

Signal attenuation, returned as a real-valued *M*-by-*N* matrix. Each matrix row represents a different path where *M* is the number of paths. Each column represents a different frequency where *N* is the number of frequencies. Units are in dB.

More About

Rainfall Attenuation Model

This model calculates the attenuation of signals that propagate through regions of rainfall. Rain attenuation is a dominant fading mechanism and can vary from location-to-location and from year-to-year.

Electromagnetic signals are attenuated when propagating through a region of rainfall. Rainfall attenuation is computed according to the ITU rainfall model *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The model computes the specific attenuation (attenuation per kilometer) of a signal as a function of rainfall rate, signal frequency, polarization, and path elevation angle. The specific attenuation, γ_R, is modeled as a power law with respect to rain rate

$$\gamma_R = kR^{\alpha},$$

where R is rain rate. Units are in mm/hr. The parameter k and exponent α depend on the frequency, the polarization state, and the elevation angle of the signal path. The specific attenuation model is valid for frequencies from 1–1000 GHz.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the an effective propagation distance, d_{eff}. Then, the total attenuation is $L = d_{\mathrm{eff}}\gamma_R$.

The effective distance is the geometric distance, d, multiplied by a scale factor

$$r = \frac{1}{0.477d^{0.633}R_{0.01}^{0.073\alpha}f^{0.123} - 10.579(1 - \exp(-0.024d))}$$

where f is the frequency. The article *Recommendation ITU-R P.530-17 (12/2017): Propagation data and prediction methods required for the design of terrestrial line-of-sight systems* presents a complete discussion for computing attenuation.

The rain rate, R, used in these computations is the long-term statistical rain rate, $R_{0.01}$. This is the rain rate that is exceeded 0.01% of the time. The calculation of the statistical rain rate is discussed in *Recommendation ITU-R P.837-7 (06/2017): Characteristics of precipitation for propagation modelling*. This article also explains how to compute the attenuation for other percentages from the 0.01% value.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

References

[1] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. 2005.

[2] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.530-17: Propagation data and prediction methods required for the design of terrestrial line-of-sight systems*. 2017.

[3] *Recommendation ITU-R P.837-7: Characteristics of precipitation for propagation modelling*

[4] Seybold, J. *Introduction to RF Propagation*. New York: Wiley & Sons, 2005.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Does not support variable-size inputs.

See Also

addCustomTerrain

Add custom terrain data

Syntax

```
addCustomTerrain(terrainName,files)
addCustomTerrain( ___ ,Name,Value)
```

Description

`addCustomTerrain(terrainName,files)` adds the terrain data specified with a user-defined `terrainName` and `files`. You can use this function to add custom terrain data in Site Viewer and other RF propagation functions. You can access the custom terrain data in the current and future sessions of MATLAB until you call `removeCustomTerrain`.

Note In Antenna Toolbox, `addCustomTerrain` function converts terrain elevation data from orthometric to ellipsoidal for visualization and when performing Euclidean distance or angle calculations between locations for example for free space path loss.

`addCustomTerrain( ___ ,Name,Value)` adds custom terrain data with additional options specified by one or more name-value pairs.

Examples

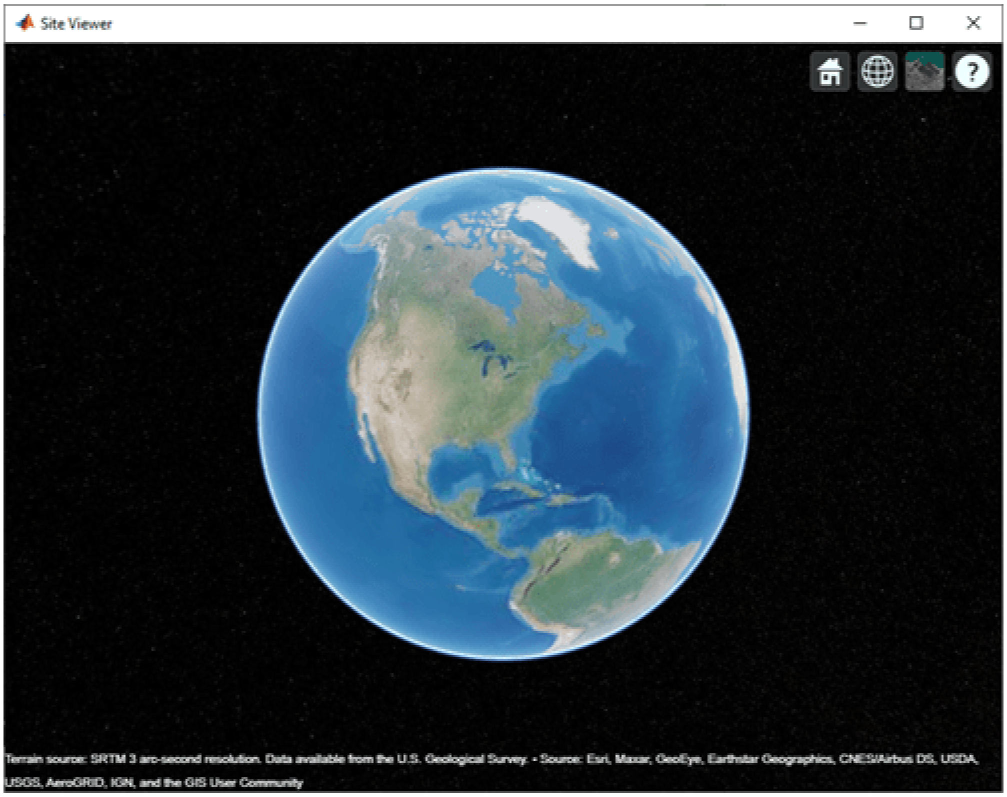
Site Viewer Maps Using Custom Terrain

Add terrain for a region around Boulder, CO. The DTED file was downloaded from the "SRTM Void Filled" data set available from the U.S. Geological Survey.

```
dtedfile = "n39_w106_3arc_v2.dt1";
attribution = "SRTM 3 arc-second resolution. Data available " + ...
   "from the U.S. Geological Survey.";
addCustomTerrain("southboulder",dtedfile,"Attribution",attribution)
```

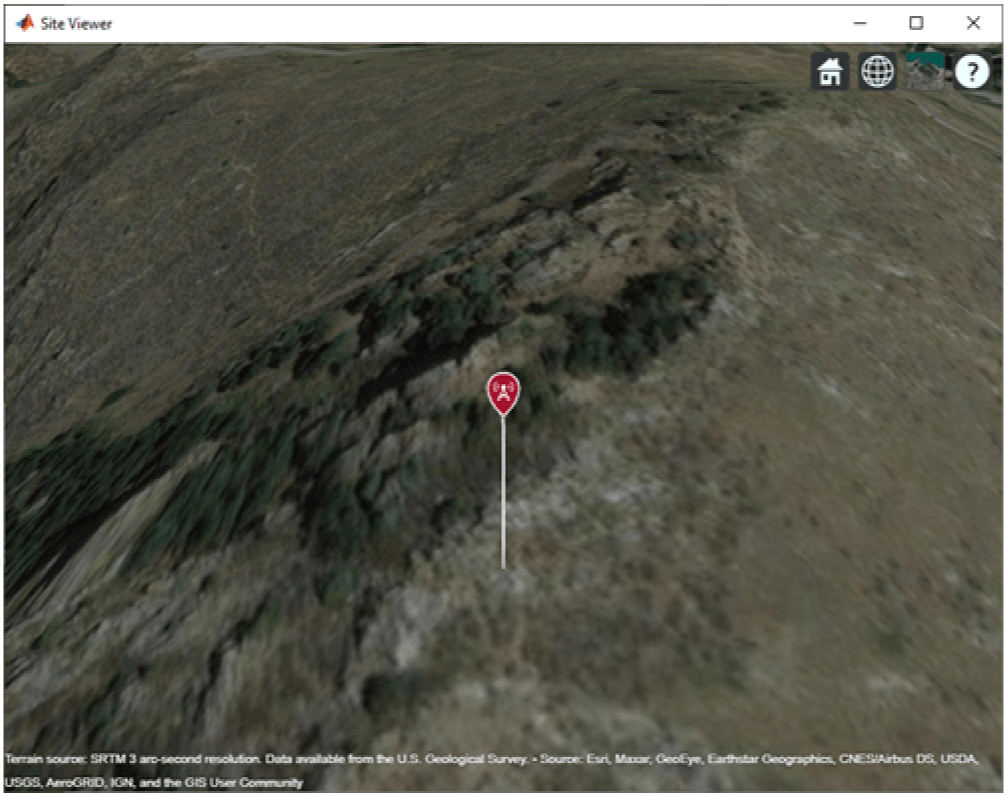
Use the custom terrain name in Site Viewer.

```
viewer = siteviewer("Terrain","southboulder");
```


Site Viewer
Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

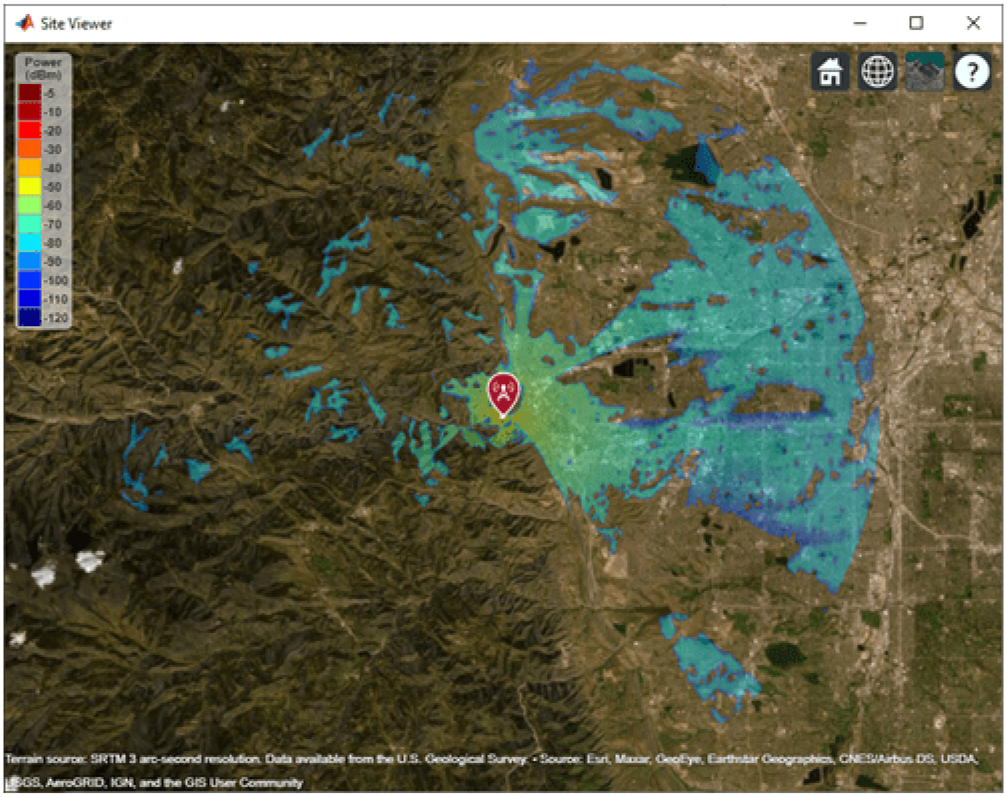
Create a site with the terrain region.

```
mtzion = txsite("Name","Mount Zion", ...
   "Latitude",39.74356, ...
   "Longitude",-105.24193, ...
   "AntennaHeight", 30);
show(mtzion)
```


Site Viewer
Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create a coverage map of the area within 20 km of the transmitter site.

```
coverage(mtzion, ...
   "MaxRange",20000, ...
   "SignalStrengths",-100:-5)
```



Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Remove the custom terrain.

```
close(viewer)
removeCustomTerrain("southboulder")
```

Input Arguments

terrainName — User-defined identifier for terrain data

string scalar | character vector

User-defined identifier for terrain data, specified as a string scalar or a character vector.

Data Types: `char` | `string`

files — Names of DTED files to read

string scalar | character vector | string vector | cell array of character vectors

Names of DTED files to read, specified as a string scalar, a character vector, a string vector, or a cell array of character vectors.

- To add custom terrain from one DTED file, specify `files` as a string scalar or a character vector.
- To add custom terrain from multiple DTED files, specify `files` as a string vector or a cell array of character vectors. If you specify multiple files that do not cover a complete rectangular geographic region, you must set the `FillMissing` name-value argument to `true`.

The form of each element of `files` depends on the location of the file.

- If the file is in your current folder or in a folder on the MATLAB path, then specify the name of the file, such as `"myFile.dt1"`.
- If the file is not in the current folder or in a folder on the MATLAB path, then specify the full or relative path name, such as `"C:\myfolder\myFile.dt1"` or `"dataDir\myFile.dt1"`.

Data Types: `char` | `string` | `cell`

Name-Value Arguments

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

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

Example: `'FillMissing',true`

`Attribution` — Attribution of custom terrain data
character vector | string scalar

Attribution of custom terrain data, specified as a character vector or a string scalar. The attribution is displayed on the Site Viewer map. By default, the value is empty.

Data Types: `char` | `string`

`FillMissing` — Fill data of missing files with value 0
`false` (default) | `true`

Fill data of missing files with value 0, specified as `true` or `false`. Missing file values are required to complete a rectangular geographic region with the input `files`.

Data Types: `logical`

`WriteLocation` — Name of folder to write extracted terrain files to
character vector | string scalar

Name of folder to write extracted terrain files to, specified as a character vector or a string scalar. The folder must exist and have write permissions. By default, `addCustomTerrain` writes extracted terrain files to a temporary folder that it generates using the `tempname` function.

Data Types: `char` | `string`

Tips

- You can find and download DTED files by using EarthExplorer, a data portal provided by the US Geological Survey (USGS). From the list of data sets, search for DTED files by selecting **Digital Elevation**, **SRTM**, and then **SRTM 1 Arc-Second Global** and **SRTM Void Filled**.

Version History

Introduced in R2019a

See Also

`removeCustomTerrain` | `siteviewer`

angle

Angle between sites

Syntax

```
[az,el] = angle(site1,site2)
[az,el] = angle(site1,site2,path)
[az,el] = angle(___,Name,Value)
```

Description

`[az,el] = angle(site1,site2)` returns the azimuth and elevation angles between `site1` and `site2`.

`[az,el] = angle(site1,site2,path)` returns the angles using a specified path type, either a Euclidean or great circle path.

`[az,el] = angle(___,Name,Value)` returns the azimuth and elevation angles with additional options specified by name-value arguments.

Examples

Angle Between Sites

Create transmitter and receiver sites.

```
tx = txsite('Name','MathWorks','Latitude',42.3001,'Longitude',-71.3504);
rx = rxsite('Name','Fenway Park','Latitude',42.3467,'Longitude',-71.0972);
```

Get the azimuth and elevation angles between the sites.

```
[az,el] = angle(tx,rx)
```

```
az = 14.0142

el = -0.2816
```

Get the azimuth angle between sites in degrees clockwise from north.

```
azFromEast = angle(tx,rx); % Unit: degrees counter-clockwise from east
azFromNorth = -azFromEast + 90 % Convert angle to clockwise from north
```

```
azFromNorth = 75.9858
```

Angle Between Sites When Path is Great Circle

Create transmitter and receiver sites.

```
tx = txsite('Name','MathWorks','Latitude',42.3001,'Longitude',-71.3504);
rx = rxsite('Name','Fenway Park','Latitude',42.3467,'Longitude',-71.0972);
```

Get the azimuth and elevation angles between the sites.

```
[az,el] = angle(tx,rx,'greatcircle')
```

```
az = 14.0635
```

```
el = 0
```

Input Arguments

`site1,site2` — Transmitter or receiver site

`txsite` object | `rxsite` object

Transmitter or receiver site, specified as a `txsite` or `rxsite` object. You can use array inputs to specify multiple sites.

`path` — Measurement path type

`'euclidean'` | `'greatcircle'`

Measurement path type, specified as one of the following:

- `'euclidean'` — Use the shortest path through space connecting the antenna center positions of the sites.
- `'greatcircle'` — Use the shortest path on the surface of the earth connecting the latitude and longitude locations of the sites. This path uses a spherical Earth model.

Data Types: `char`

Name-Value Pair Arguments

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

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

Example: `'Map','siteviewer1'`

`Map` — Map for visualization or surface data

`siteviewer` object | `triangulation` object | string scalar | character vector

Map for visualization or surface data, specified as a `siteviewer` object, a `triangulation` object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
`"geographic"`	- A `siteviewer` object[a]. - A terrain name, if the function is called with an output argument. Valid terrain names are `"none"`, `"gmted2010"`, or the name of the custom terrain data added using `addCustomTerrain`.	- The current `siteviewer` object or a new `siteviewer` object if none are open. - `"gmted2010"`, if the function is called with an output.
`"cartesian"`	- `"none"`. - A `siteviewer` object. - The name of an STL file. - A `triangulation` object.	- `"none"`.

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: `char` | `string`

Output Arguments

`az` — Azimuth angle between sites

M-by-*N* arrays

Azimuth angle between `site1` and `site2`, returned as *M*-by-*N* arrays in degrees. *M* is the number of sites in `site1` and *N* is the number of sites in `site2`. The azimuth angle is expressed in degrees counter-clockwise from the east (for geographic sites), or from the global x-axis around the global z-axis (for Cartesian sites), ranging from -180 to 180 degrees.

`el` — Elevation angle between sites

M-by-*N* arrays

Elevation angle between `site1` and `site2`, returned as *M*-by-*N* arrays in degrees. *M* is the number of sites in `site2` and *N* is the number of sites in `site1`. The elevation angle is expressed in degrees from the horizontal (or X-Y) plane, ranging from -90 to 90 degrees.

When you specify the path type as `'greatcircle'`, the elevation angle is always zero.

Version History

Introduced in R2017b

See Also

`distance`

clearMap

Clear plots

Syntax

```
clearMap(viewer)
```

Description

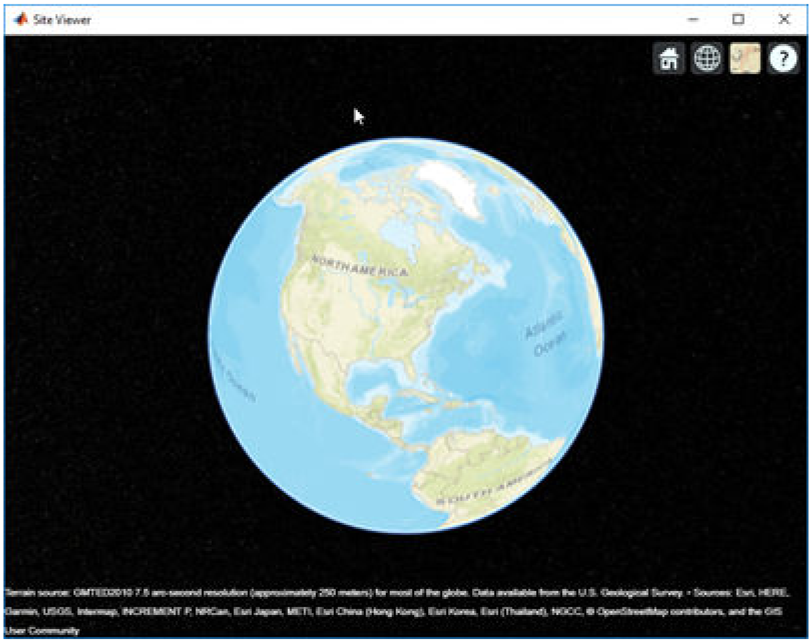
`clearMap(viewer)` removes all plots from the specified Site Viewer.

Examples

View Transmitter Site On Site Viewer

1. Launch a Site Viewer with streets basemap.

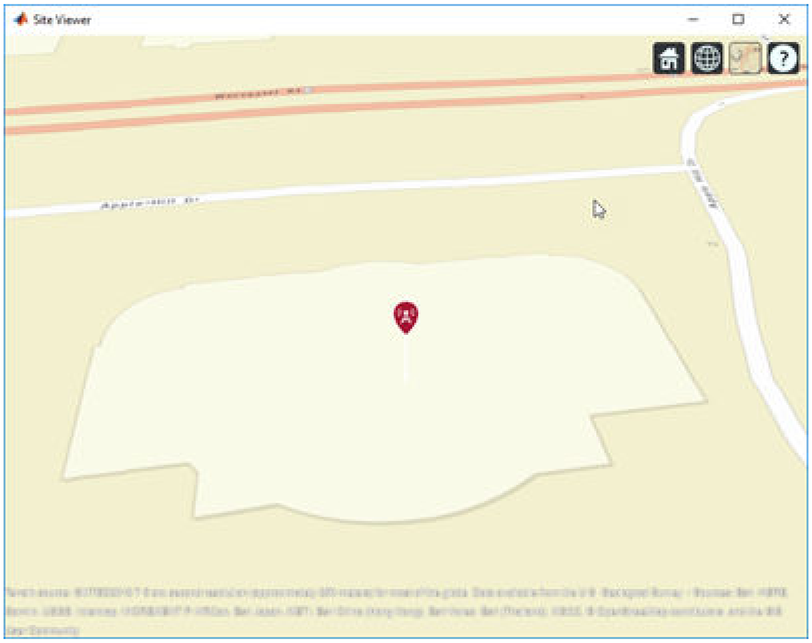
```
viewer = siteviewer("Basemap","streets");
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, HERE, Garmin, USGS, Intermap, INCREMENT P, NRCan, Esri Japan, METI, Esri China (Hong Kong), Esri Korea, Esri (Thailand), NGCC, © OpenStreetMap contributors, and the GIS User Community

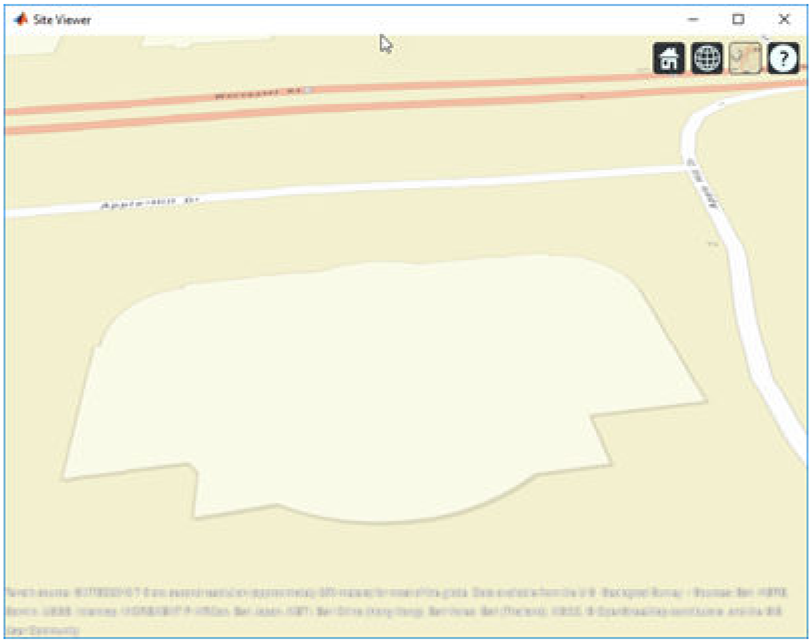
2. View a transmitter site on this map.

```
tx = txsite;
show(tx)
```



3 Clear the map.

```
t = timer('TimerFcn',@(~,~)disp('Fired.'),'StartDelay',3);
start(t)
wait(t)
clearMap(viewer)
```



Input Arguments

viewer — Map viewer for visualizing transmitter or receiver sites

siteviewer object

Map viewer for visualizing transmitter or receiver sites, specified as a `siteviewer` object.[1]

Version History

Introduced in R2019a

See Also

`close` | `siteviewer`

[1] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

close

Close Site Viewer

Syntax

```
close(viewer)
```

Description

`close(viewer)` closes the Site Viewer window and deletes the handle.

Examples

Compare Coverage Maps

Launch two Site Viewer windows. One Site Viewer window uses the terrain model and the other window does not use the terrain model.

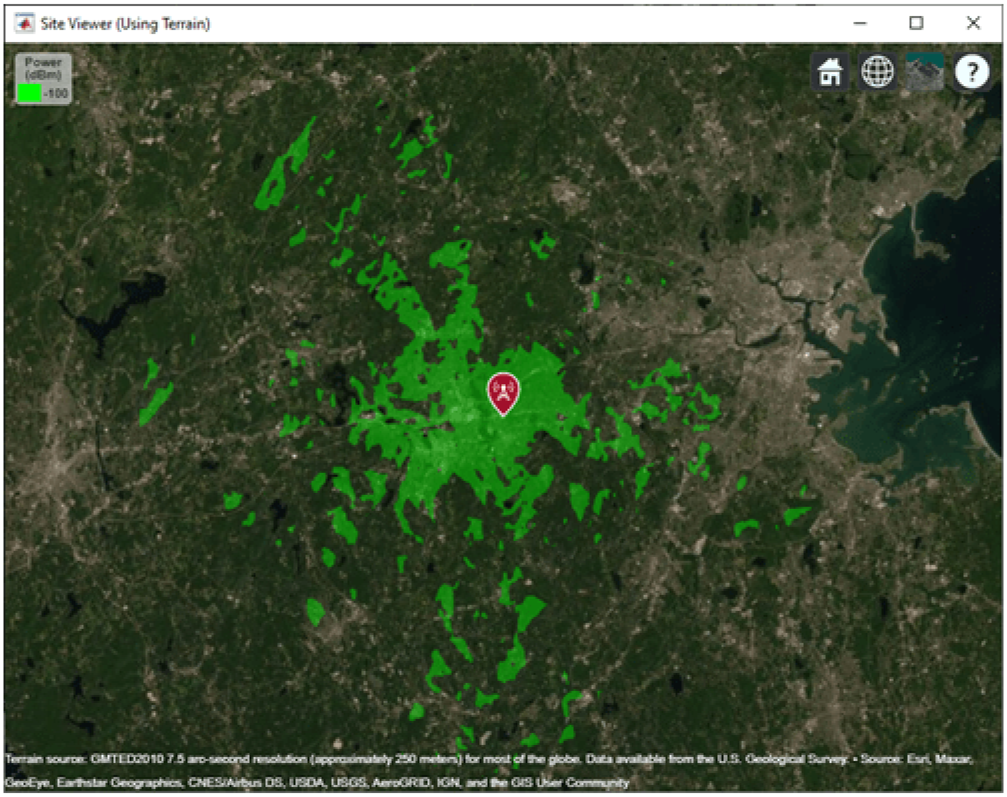
```
viewer1 = siteviewer("Terrain","gmted2010","Name","Site Viewer (Using Terrain)");
viewer2 = siteviewer("Terrain","none","Name","Site Viewer (No Terrain)");
```

Create a transmitter site.

```
tx = txsite;
```

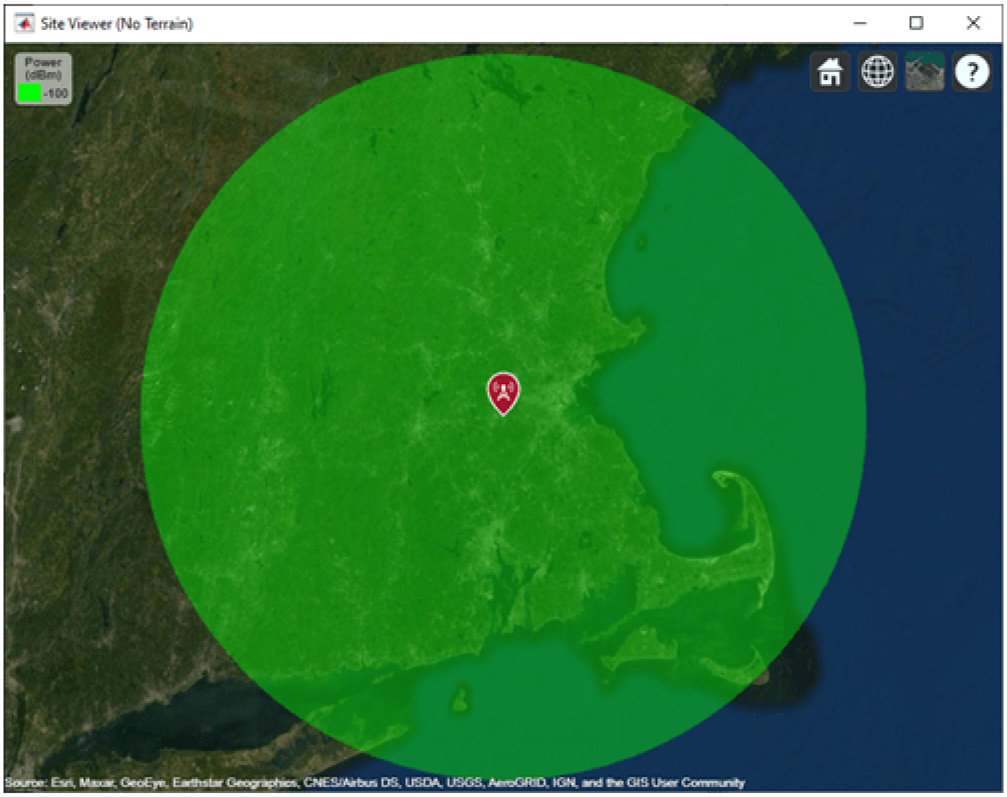
Generate a coverage map on each window. The map with terrain uses the Longley-Rice propagation model by default.

```
coverage(tx,"Map",viewer1)
```


Site Viewer (Using Terrain)
Power (dBm)
-100
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

The map without terrain uses the free-space model by default.

```
coverage(tx,"Map",viewer2)
```


Site Viewer (No Terrain)
Power (dBm)
🟩 -100
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

`viewer` — Map viewer for visualizing transmitter or receiver sites

`siteviewer` object

Map viewer for visualizing transmitter or receiver sites, specified as a `siteviewer` object.[2]

Version History

Introduced in R2019a

See Also

`clearMap` | `siteviewer`

[2] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

coverage

Display or compute coverage map

Syntax

```
coverage(txs)
coverage(txs,propmodel)
coverage(txs,rx)
coverage(txs,rx,propmodel)
coverage( ___ ,Name,Value, ___ )
pd = coverage(txs, ___ )
```

Description

`coverage(txs)` displays the coverage map for the specified transmitter site in the current Site Viewer. Each colored contour of the map defines an area where the corresponding signal strength is transmitted to the mobile receiver.

Note This function only supports antenna sites with `CoordinateSystem` property set to `"geographic"`.

`coverage(txs,propmodel)` displays the coverage map based on the specified propagation model. The default propagation model is `"longley-rice"` when terrain is in use and `"freespace"` when terrain is not used.

`coverage(txs,rx)` displays the coverage map based on the receiver site properties.

`coverage(txs,rx,propmodel)` displays the coverage map based on the receiver site properties and specified propagation model.

`coverage( ___ ,Name,Value, ___ )` displays the coverage map using additional options specified by the `Name,Value` pairs.

`pd = coverage(txs, ___ )` returns computed coverage data in the propagation data object, `pd`. No plot is displayed and any graphical only name-value pairs are ignored.

Examples

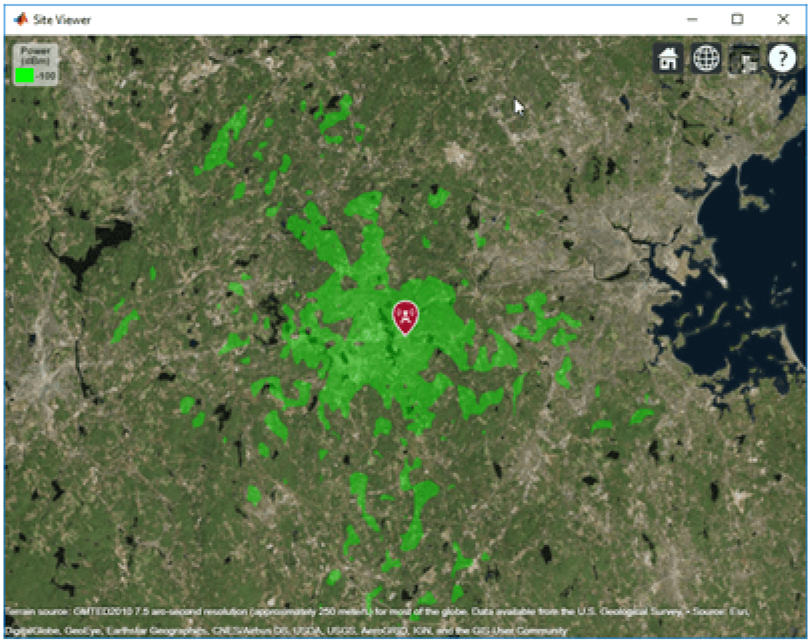
Coverage Map of Transmitter

Create a transmitter site at MathWorks headquarters.

```
tx = txsite('Name','MathWorks', ...
        'Latitude', 42.3001, ...
        'Longitude', -71.3503);
```

Show the coverage map.

```
coverage(tx)
```


Site Viewer
Power (dBm)
🟩 -100
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Coverage Map Using Transmitter and Receiver

Create a transmitter site at MathWorks headquarters.

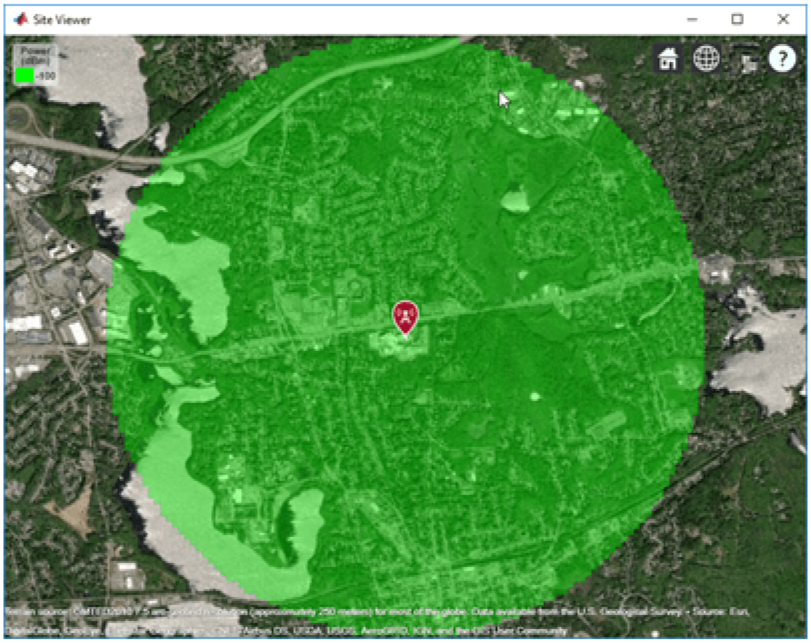
```
tx = txsite('Name','MathWorks', ...
        'Latitude', 42.3001, ...
        'Longitude', -71.3503);
```

Create a receiver site at Fenway Park with an antenna height of 1.2 m and system loss of 10 dB.

```
rx = rxsite('Name','Fenway Park', ...
       'Latitude',42.3467, ...
       'Longitude',-71.0972,'AntennaHeight',1.2,'SystemLoss',10);
```

Calculate the coverage area of the transmitter using a close-in propagation model.

```
coverage(tx,rx,'PropagationModel','closein')
```


Site Viewer
Power (dBm)
-100
Data available from the U.S. Geological Survey • Source: Esri,
Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

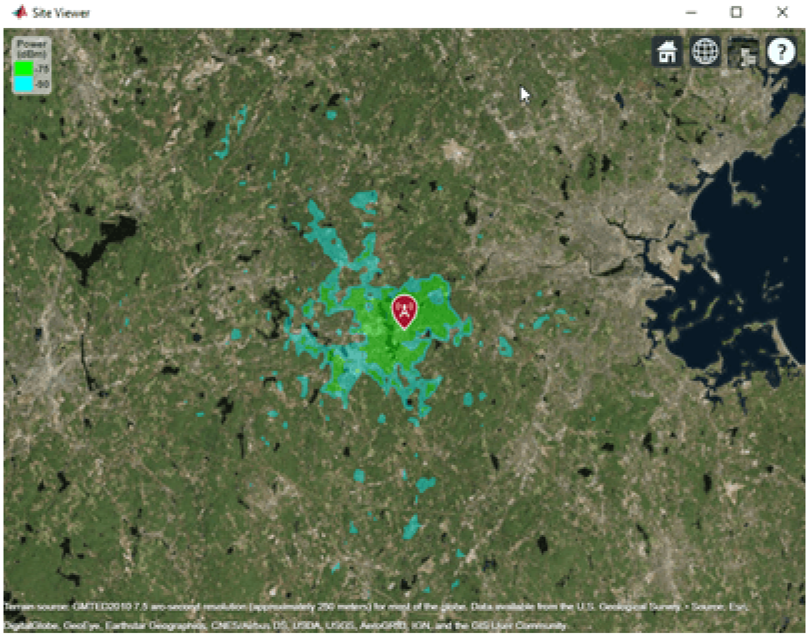
Coverage Map for Strong and Weak Signals

Define strong and weak signal strengths with corresponding colors.

```
strongSignal = -75;
strongSignalColor = "green";
weakSignal = -90;
weakSignalColor = "cyan";
```

Create a transmitter site and display the coverage map.

```
tx = txsite('Name','MathWorks', ...
    'Latitude',42.3001, ...
    'Longitude',-71.3503);
coverage(tx, ...
    'SignalStrengths',[strongSignal,weakSignal], ...
    'Colors', [strongSignalColor,weakSignalColor])
```


Site Viewer
Power (dBm)
-75
-90
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Coverage Map of Directional Antenna in Rain

Define a Yagi-Uda antenna designed for a transmitter frequency of 4.5 GHz. Tilt the antenna to direct radiation in the XY-plane (i.e., geographic azimuth).

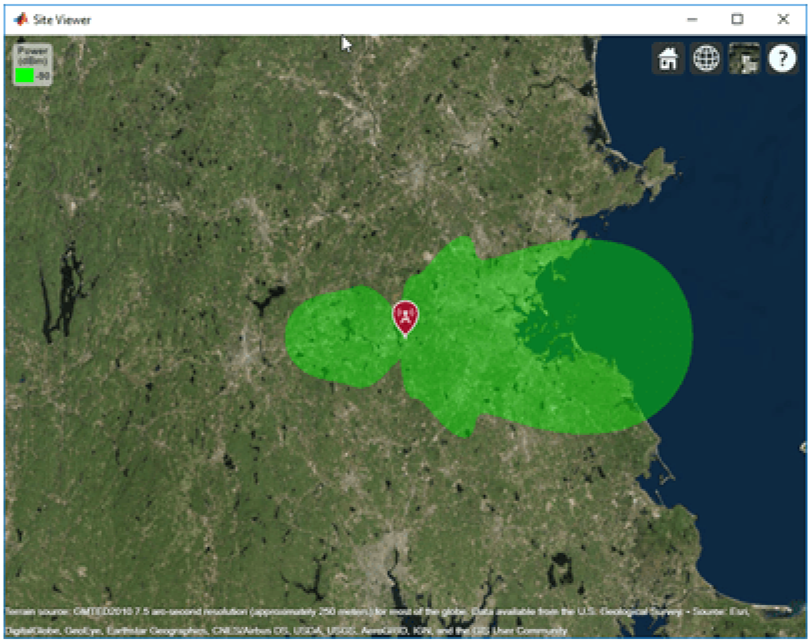
```
fq = 4.5e9;
y = design(yagiUda,fq);
y.Tilt = 90;
y.TiltAxis = 'y';
```

Create a transmitter site with this directional antenna.

```
tx = txsite('Name','MathWorks', ...
    'Latitude',42.3001,'Longitude',-71.3503, ...
    'Antenna',y,'AntennaHeight',60, ...
    'TransmitterFrequency',fq,'TransmitterPower',10);
```

Display the coverage map using the rain propagation model. The map pattern points east, which corresponds to default antenna angle value of 0 degrees.

```
coverage(tx,'rain','SignalStrengths',-90)
```


Site Viewer
Power (dBm)
🟩 -80
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Combined Coverage Map of Multiple Transmitters

Define the names and the locations of sites around Boston.

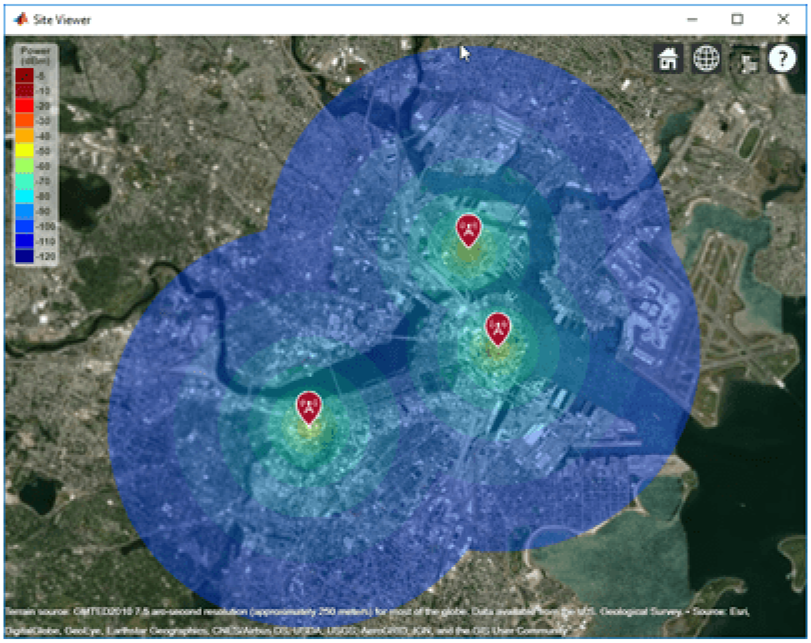
```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

Create the transmitter site array.

```
txs = txsite('Name', names,...
       'Latitude',lats,...
       'Longitude',lons, ...
       'TransmitterFrequency',2.5e9);
```

Display the combined coverage map for multiple signal strengths, using close-in propagation model.

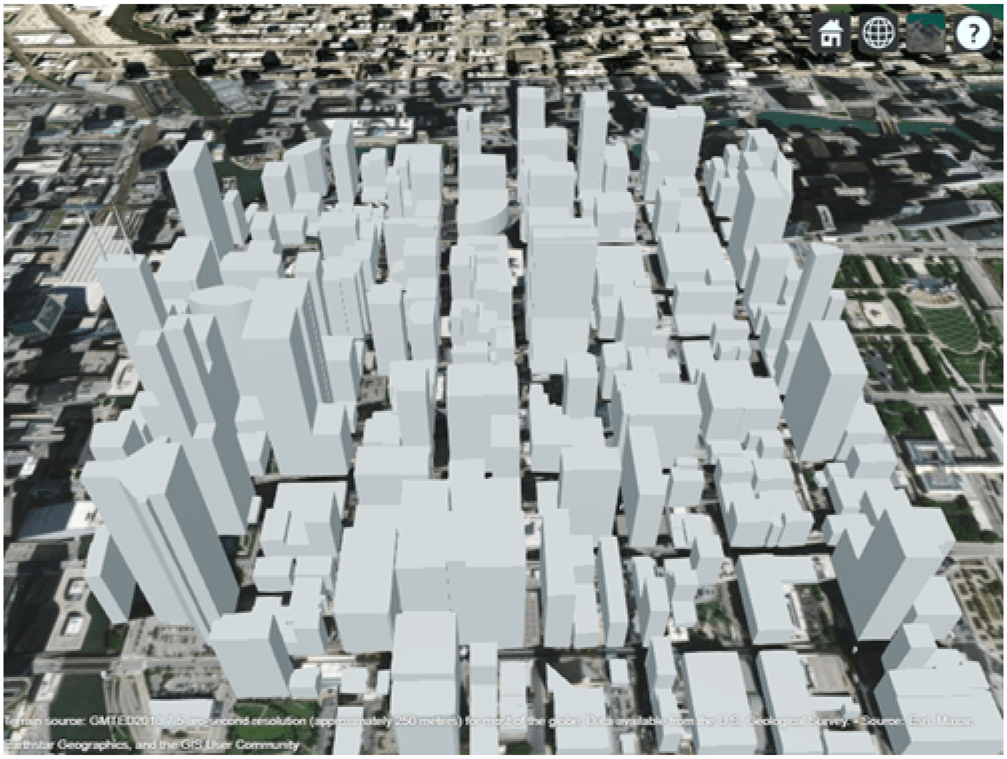
```
coverage(txs,'close-in','SignalStrengths',-100:5:-60)
```


Site Viewer
Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Coverage Map Using Longley-Rice and Ray Tracing Method

Launch Site Viewer using buildings in Chicago. For more information about the osm file, see [1] on page 6-85.

```
viewer = siteviewer("Buildings","chicago.osm");
```



Terrain source: GMTED2010 7.5-arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Create a transmitter site on the building.

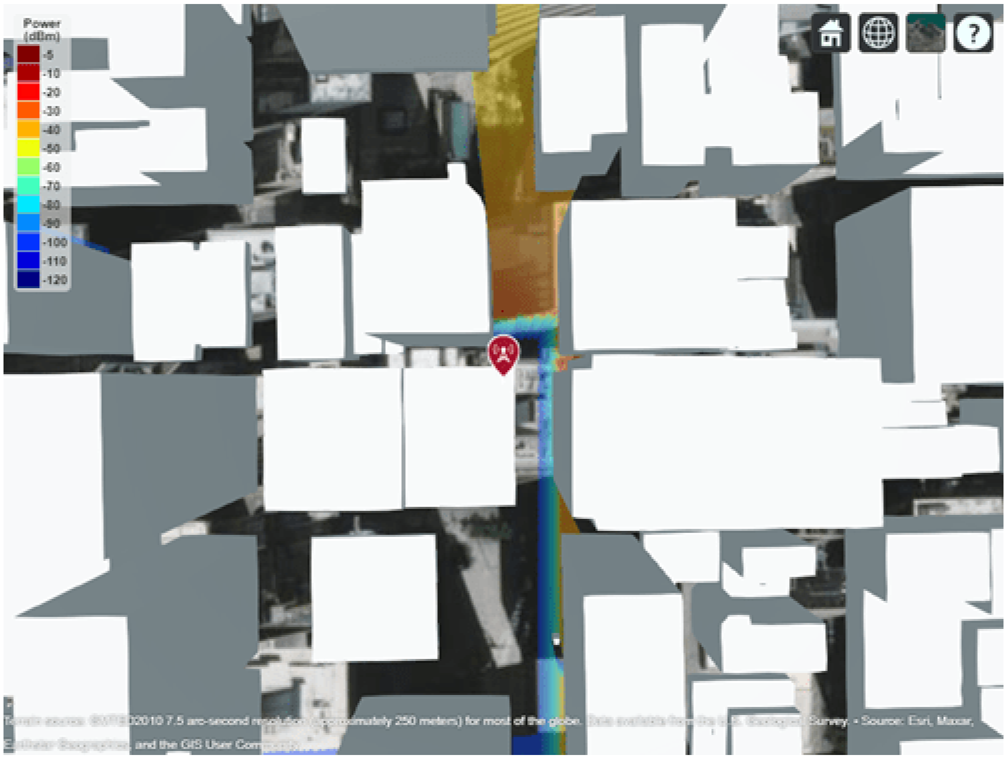
```
tx = txsite("Latitude",41.8800, ...
   "Longitude",-87.6295, ...
   "TransmitterFrequency",2.5e9);
show(tx)
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Coverage Map Using Longley-Rice Propagation Model

Create a coverage map of the city using the Longley-Rice propagation model.

```
coverage(tx,"SignalStrengths",-100:-5,"MaxRange",250,"Resolution",1)
```

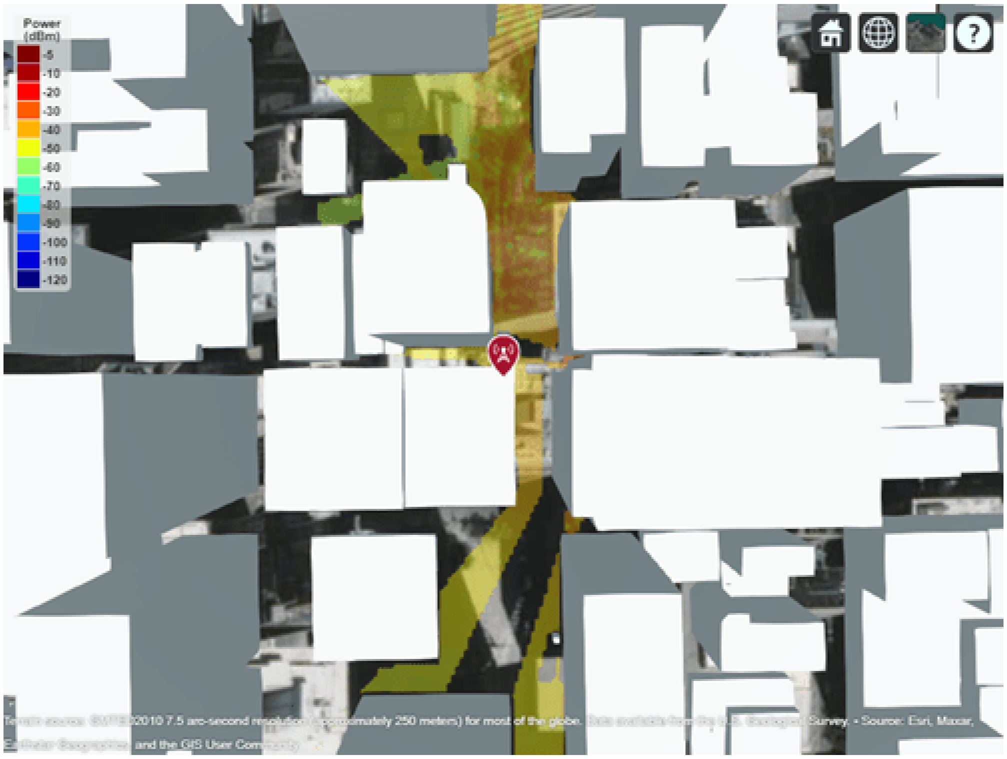

Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Longley-Rice models over-the-rooftops propagation along vertical slices and obstructions tend to dominate the coverage region.

Coverage Map Using Ray Tracing Propagation Model and Image Method

Create a coverage map of the city by using a ray tracing propagation model that uses the image method. Calculate line-of-sight and single-reflection propagation paths.

```
pmImage = propagationModel("raytracing","Method","image", ...
    "MaxNumReflections",1);
coverage(tx,pmImage,"SignalStrengths",-100:-5, ...
    "MaxRange",250,"Resolution",2)
```

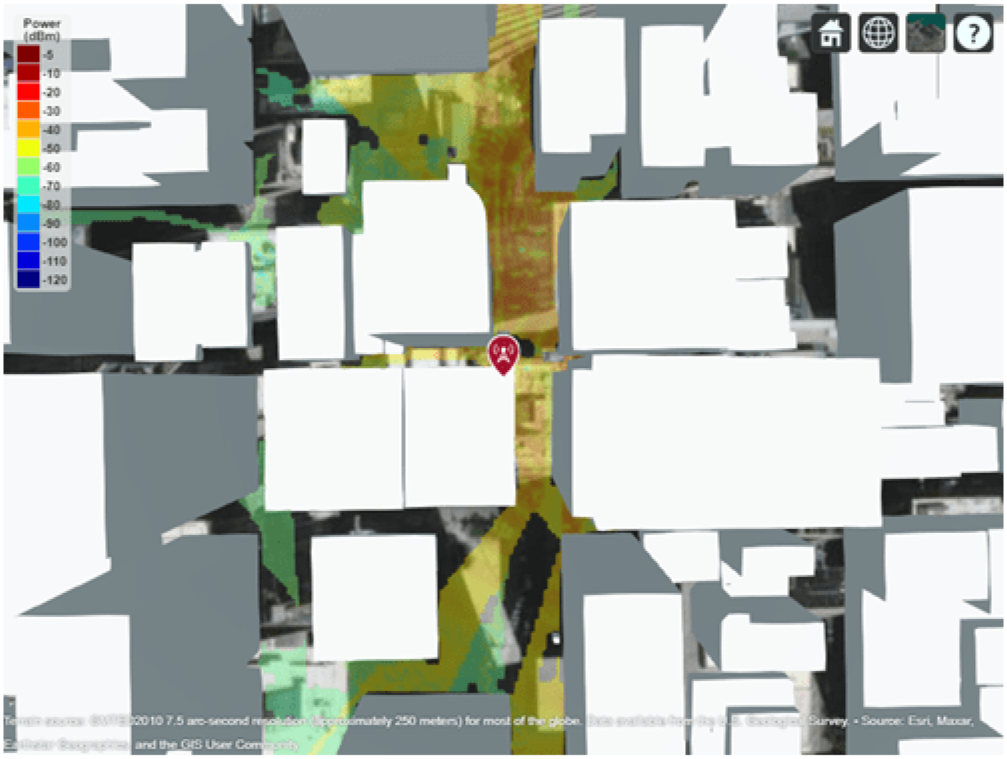

Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

This coverage map shows new regions that are in service due to reflected propagation paths.

Coverage Map Using Ray Tracing Propagation Model and SBR Method

Create a coverage map of the city by using a ray tracing propagation model that uses the shooting and bouncing rays (SBR) method, which is generally faster than the image method. Increase the maximum number of path reflections to 2.

```
pmSBR = propagationModel("raytracing","Method","sbr", ...
    "MaxNumReflections",2);
coverage(tx,pmSBR,"SignalStrengths",-100:-5, ...
    "MaxRange",250,"Resolution",2)
```


Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: SRTM2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

This coverage map shows new regions that are in service due to the additional reflected propagation paths.

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Input Arguments

txs — Transmitter sites

txsite object | array of txsite objects

Transmitter site, specified as a `txsite` object. Use array inputs to specify multiple sites.

This function only supports plotting antenna sites when `CoordinateSystem` property is set to `"geographic"`.

rx — Receiver site

rxsite object

Receiver site, specified as a `rxsite` object.

This function only supports plotting antenna sites when `CoordinateSystem` property is set to `"geographic"`.

propmodel — Propagation model to use for path loss calculations

"longley-rice" (default) | "freespace" | "close-in" | "rain" | "gas" | "fog" | "raytracing" | propagation model created with propagationModel

Propagation model to use for the path loss calculations, specified as one of these options:

- "freespace" — Free space propagation model
- "rain" — Rain propagation model
- "gas" — Gas propagation model
- "fog" — Fog propagation model
- "close-in" — Close-in propagation model
- "longley-rice" — Longley-Rice propagation model
- "tirem" — TIREM propagation model
- "raytracing" — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the propagationModel function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
"geographic"	• "longley-rice" when you use a terrain. • "freespace" when you do not use a terrain.
"cartesian"	• "freespace" when Map is set to none. • "raytracing" when Map is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including "longley-rice" and "tirem", are only supported for sites with a CoordinateSystem value of "geographic".

You can also specify the propagation model by using the PropagationModel name-value pair argument.

Name-Value Pair Arguments

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

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

Example: "Type","power"

Type — Type of signal strength to compute

"power" (default) | "efield"

Type of signal strength to compute, specified as one of these options:

- "power" — The signal strengths in SignalStrengths is in power units (dBm) of the signal at the mobile receiver input.
- "efield"— The signal strength in SignalStrengths is in electric field strength units (dBµV/m) of signal wave incident on the antenna.

Data Types: char

SignalStrengths — Signal strengths to display on coverage map

numeric vector

Signal strengths to display on coverage map, specified as a numeric vector.

Each strength uses different colored filled contour on the map. The default value is -100 dBm if Type is "power" and 40 dBµV/m if Type is "efield".

Data Types: double

PropagationModel — Propagation model to use for path loss calculations

"freespace" | "close-in" | "rain" | "gas" | "fog" | "longley-rice" | "raytracing" | propagation model created with propagationModel

Propagation model to use for the path loss calculations, specified as one of these options:

- "freespace" — Free space propagation model
- "rain" — Rain propagation model
- "gas" — Gas propagation model
- "fog" — Fog propagation model
- "close-in" — Close-in propagation model
- "longley-rice" — Longley-Rice propagation model
- "tirem" — TIREM propagation model
- "raytracing" — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the propagationModel function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
"geographic"	• "longley-rice" when you use a terrain. • "freespace" when you do not use a terrain.
"cartesian"	• "freespace" when Map is set to none. • "raytracing" when Map is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including "longley-rice" and "tirem", are only supported for sites with a CoordinateSystem value of "geographic".

Data Types: char | string

MaxRange — Maximum range of coverage map from each transmitter site

numeric scalar

Maximum range of coverage map from each transmitter site, specified as a positive numeric scalar in meters representing great circle distance. MaxRange defines the region of interest on the map to plot. The default value is automatically computed based on the type of propagation model.

Type of Propagation Model	MaxRange
Atmospheric or empirical	Range of minimum value in SignalStrengths.
Terrain	30 km or distance to the furthest building.
Ray tracing	500 m

For more information about the types of propagation models, see “Choose a Propagation Model”.

Data Types: double

Resolution — Resolution of coverage map

"auto" (default) | numeric scalar

Resolution of coverage map, specified as "auto" or a numeric scalar in meters.

The resolution of "auto" computes the maximum value scaled to MaxRange. Decreasing the resolution increases the quality of the coverage map and the time required to create it.

Data Types: char | double

ReceiverGain — Mobile receiver gain

2.1 (default) | numeric scalar

Mobile receiver gain, specified as a numeric scalar in dB. The receiver gain value includes the mobile receiver antenna gain and system loss.

The receiver gain computes received signal strength when Type is "power".

If receiver site argument rx is passed to coverage, the default value is the maximum gain of the receiver antenna with the system loss subtracted. Otherwise the default value is 2.1.

Data Types: char | double

ReceiverAntennaHeight — Mobile receiver antenna height above ground elevation

1 (default) | numeric scalar

Mobile receiver antenna height above ground elevation, specified as a numeric scalar in meters.

If receiver site argument rx is passed to coverage, the default value is the AntennaHeight of the receiver. Otherwise the default value is 1.

Data Types: double

Colors — Colors of filled contours on coverage map

M-by-3 array of RGB triplets | array of strings | cell array of character vectors

Colors of filled contours on coverage map, specified as one of these options:

- An *M*-by-3 array of RGB triplets whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- An array of strings such as `["red" "green" "blue"]` or `["r" "g" "b"]`.
- A cell array of character vectors such as `{'red','green','blue'}` or `{'r','g','b'}`.

Colors are assigned element-wise to `SignalStrengths` values for coloring the corresponding filled contours.

`Colors` cannot be used with `ColorLimits` or `ColorMap`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	🟥
`"green"`	`"g"`	`[0 1 0]`	🟩
`"blue"`	`"b"`	`[0 0 1]`	🟦
`"cyan"`	`"c"`	`[0 1 1]`	🟦
`"magenta"`	`"m"`	`[1 0 1]`	🟪
`"yellow"`	`"y"`	`[1 1 0]`	🟨
`"black"`	`"k"`	`[0 0 0]`	⬛
`"white"`	`"w"`	`[1 1 1]`	⬜

Data Types: `char` | `string` | `double`

`ColorLimits` — Color limits for colormap

two-element vector

Color limits for colormap, specified as a two-element vector of type `[min max]`.

The color limits indicate the signal level values that map to the first and last colors on the colormap.

The default value is `[-120 -5]` if the `Type` is `"power"` and `[20 135]` if `Type` is `"efield"`.

`ColorLimits` cannot be used with `Colors`.

Data Types: `double`

`ColorMap` — Colormap filled contours for coverage map

`"jet"` (default) | predefined color map | *M*-by-3 array of RGB triplets

Colormap filled contours on coverage map, specified as a predefined colormap or *M*-by-3 array of RGB triplets, where *M* defines individual colors.

`ColorMap` cannot be used with `Colors`.

Data Types: `char` | `double`

`ShowLegend` — Show signal strength color legend on map

`true` (default) | `false`

Show signal strength color legend on map, specified as `true` or `false`.

Data Types: `logical`

`Transparency` — Transparency of coverage map
`0.4` (default) | numeric scalar

Transparency of coverage map, specified as a numeric scalar in the range `0` to `1`. `0` is transparent and `1` is opaque.

Data Types: `double`

`Map` — Map for visualization of surface data
`siteviewer` object

Map for visualization of surface data, specified as a `siteviewer` object.[3]

Data Types: `char` | `string`

Output Arguments

pd — Coverage data
`propagationData` object

Coverage data, returned as a `propagationData` object consisting of *Latitude* and *Longitude*, and a signal strength variable corresponding to the plot type. Name of the `propagationData` is `"Coverage Data"`.

Version History

Introduced in R2017b

Ray tracing functions consider multipath interference

Behavior changed in R2022b

When calculating received power using ray tracing models, the `coverage` function now incorporates multipath interference by using a phasor sum. In previous releases, the function used a power sum. As a result, the calculations in R2022b are more accurate than in previous releases.

`"raytracing"` propagation models use SBR method

Behavior changed in R2021b

Starting in R2021b, when you use the `coverage` function and specify the `propmodel` argument or `PropagationModel` name-value argument as `"raytracing"`, the function uses the shooting and bouncing rays (SBR) method and calculates up to two reflections. In previous releases, the `coverage` function uses the image method and calculates up to one reflection.

To display or compute coverage maps using the image method instead, create a propagation model by using the `propagationModel` function. Then, use the `coverage` function with the propagation model as input. This example shows how to update your code.

```
pm = propagationModel("raytracing","Method","image");
coverage(txs,pm)
```

[3] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

For information about the SBR and image methods, see “Choose a Propagation Model”.

Starting in R2021b, all RF Propagation functions use the SBR method by default and calculate up to two reflections. For more information, see “Default modeling method is shooting and bouncing rays method” on page 6-129.

See Also

`link` | `sigstrength` | `sinr` | `propagationModel`

distance

Distance between sites

Syntax

```
d = distance(site1,site2)
d = distance(site1,site2,path)
d = distance( ___ ,Name,Value)
```

Description

`d = distance(site1,site2)` returns the distance in meters between `site1` and `site2`.

`d = distance(site1,site2,path)` returns the distance using a specified path type, either a Euclidean or great circle path.

`d = distance( ___ ,Name,Value)` returns the distance with additional options specified by name-value arguments.

Examples

Distance Between Transmitter and Receiver Site

Create transmitter and receiver sites.

```
tx = txsite('Name','MathWorks','Latitude',42.3001,'Longitude',-71.3504);
rx = rxsite('Name','Fenway Park','Latitude',42.3467,'Longitude',-71.0972);
```

Get the Euclidean distance in km between the sites.

```
dme = distance(tx,rx)
```

```
dme = 2.1504e+04
```

```
dkm = dme / 1000
```

```
dkm = 21.5037
```

Get the great circle distance between the two sites.

```
dmg = distance(tx,rx,'greatcircle')
```

```
dmg = 2.1451e+04
```

Input Arguments

`site1,site2` — Transmitter or receiver site
`txsite` object | `rxsite` object

Transmitter or receiver site, specified as a `txsite` or `rxsite` object. You can use array inputs to specify multiple sites.

`path` — Measurement path type
`'euclidean'` | `'greatcircle'`

Measurement path type, specified as one of the following:

- `'euclidean'` — Use the shortest path through space that connects the antenna center positions of the sites.
- `'greatcircle'` — Use the shortest path on the surface of the earth that connects the latitude and longitude locations of the sites. This path uses a spherical Earth model.

Data Types: `char`

Name-Value Pair Arguments

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

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

Example: `'Map','siteviewer1'`

`Map` — Map for visualization or surface data
`siteviewer` object | `triangulation` object | string scalar | character vector

Map for visualization or surface data, specified as a `siteviewer` object, a `triangulation` object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
`"geographic"`	• A `siteviewer` object[a]. • A terrain name, if the function is called with an output argument. Valid terrain names are `"none"`, `"gmted2010"`, or the name of the custom terrain data added using `addCustomTerrain`.	• The current `siteviewer` object or a new `siteviewer` object if none are open. • `"gmted2010"`, if the function is called with an output.
`"cartesian"`	• `"none"`. • A `siteviewer` object. • The name of an STL file. • A `triangulation` object.	• `"none"`.

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: `char` | `string`

Output Arguments

d — Distance between sites
M-by-*N* numeric array

Distance between sites, returned as an *M*-by-*N* numeric array in meters, where *M* is the number of sites in `site2` and *N* is the number of sites in `site1`.

Version History

Introduced in R2017b

See Also

`angle`

elevation

Elevation of site

Syntax

```
z = elevation(site)
z = elevation( ___ ,Name,Value)
```

Description

`z = elevation(site)` returns the ground or building surface elevation of antenna site in meters. Elevation is measured relative to mean sea level using earth gravitational model, EGM-96. If the site coincides with a building, elevation is measured at the top of the building. Otherwise, elevation is measured at the ground.

This function only supports antenna sites with a `CoordinateSystem` property value of `'geographic'`.

`z = elevation( ___ ,Name,Value)` returns the ground elevation of the antenna in meters with additional options specified by name-value arguments.

Examples

Elevation at Mount Washington

Compute and display the elevation at Mount Washington in meters.

```
mtwash = txsite('Name','Mt Washington','Latitude',44.2706, ...
       'Longitude',-71.3033);
z = elevation(mtwash)
```

```
z = 1.8704e+03
```

Input Arguments

site — Transmitter or receiver site

`txsite` or `rxsite` object | array of `txsite` or `rxsite` objects

Transmitter or receiver site, specified as a `txsite` or `rxsite` object or an array of `txsite` or `rxsite` objects.

Name-Value Pair Arguments

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

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

Example: `'Map','siteviewer1'`

`Map` — Map for visualization or surface data
`siteviewer` object | string scalar | character vector

Map for visualization or surface data, specified as one of the following:

- A `siteviewer` object.[4]
- A terrain name if the function is called with an output argument. Valid terrain names are `'none'`, `'gmted2010'`, or the name of the custom terrain data added using `addCustomTerrain`.

The default value is:

- The current `siteviewer` object or a new `siteviewer` object if none are open.
- `'gmted2010'` if called with an output.

Data Types: `char` | `string`

Output Arguments

`z` — Ground or building surface elevation of antenna site
M-by-1 matrix

Ground or building surface elevation of the antenna site, returned as an *M*-by-`1` matrix with each element unit in meters. *M* is the number of sites in `site`.

Version History

Introduced in R2018b

See Also

Functions
`distance` | `angle`

Objects
`txsite` | `rxsite`

[4] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

hide

Hide site from Site Viewer

Syntax

```
hide(site)
hide(___,Name,Value)
```

Description

`hide(site)` hides the location of the specified antenna site from the current Site Viewer.

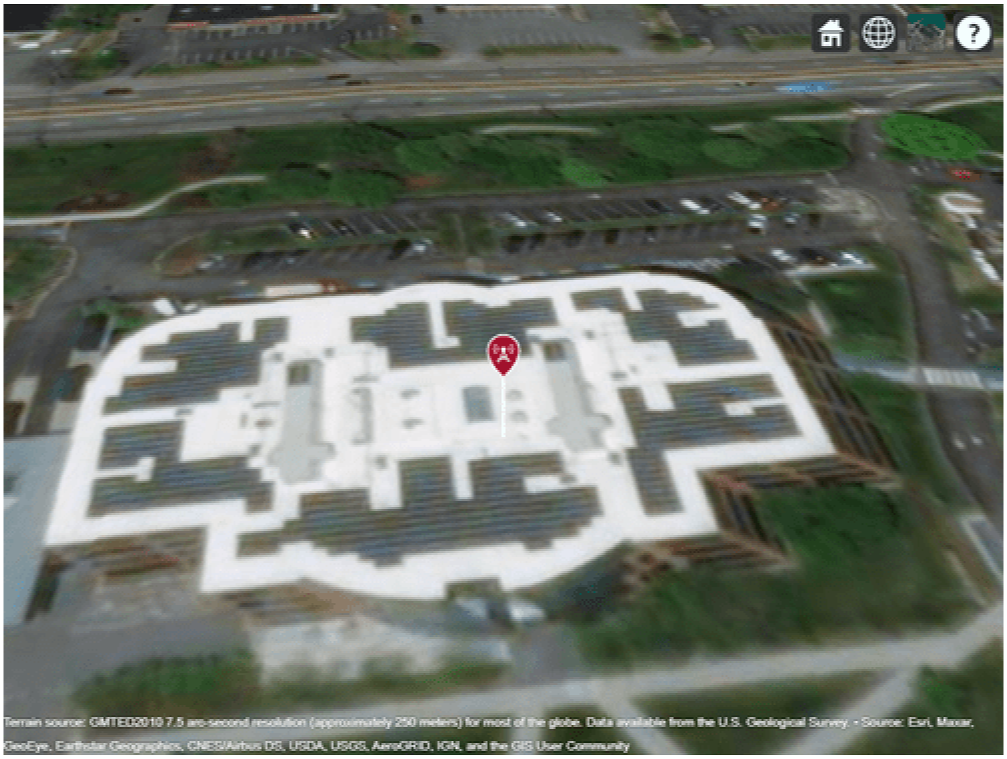
`hide(___,Name,Value)` hides the site with additional options specified by one or more name-value pairs.

Examples

Show and Hide Transmitter Site

Create and show a transmitter site.

```
tx = txsite('Name','MathWorks Apple Hill',...
       'Latitude',42.3001, ...
       'Longitude',-71.3504);
show(tx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Hide the transmitter site.

```
hide(tx)
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Show and Hide Sites with Cartesian Coordinates

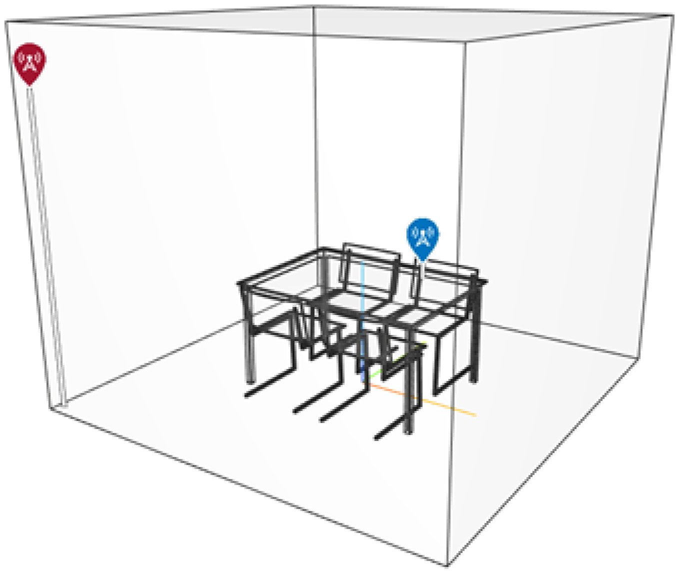
Import and view an STL file. The file models a small conference room with one table and four chairs.

```
viewer = siteviewer('SceneModel','conferenceroom.stl');
```

Create a transmitter site near the upper corner of the room and a receiver site above the table. Specify the position using Cartesian coordinates in meters. Then, visualize the sites.

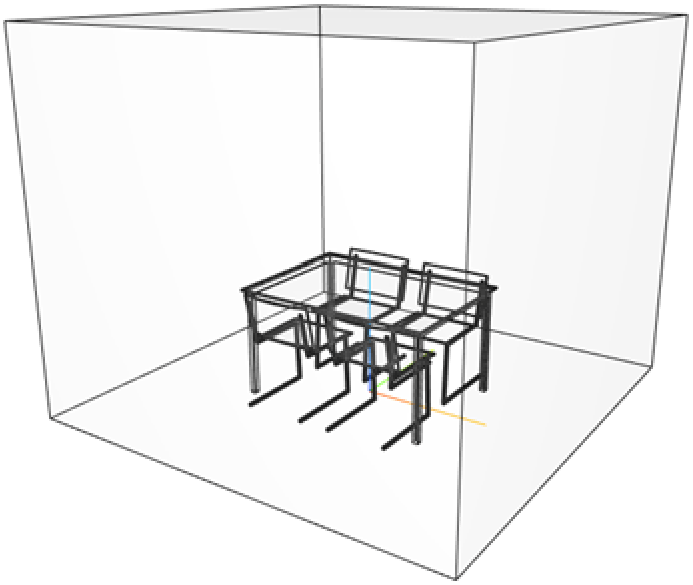
```
tx = txsite('cartesian', ...
    'AntennaPosition',[-1.46; -1.42; 2.1]);
rx = rxsite('cartesian', ...
    'AntennaPosition',[0.3; 0.3; 0.85]);

show(tx)
show(rx)
```

Pan by left-clicking, zoom by right-clicking or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking and dragging.



Hide the sites.

```
hide(tx)
hide(rx)
```



Input Arguments

`site` — Transmitter or receiver site
`txsite` or `rxsite` object | array of `txsite` or `rxsite` objects

Transmitter or receiver site, specified as a `txsite` or `rxsite` object or an array of `txsite` or `rxsite` objects.

Name-Value Pair Arguments

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

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

Example: `'Map','siteviewer1'`

`Map` — Map for visualization of surface data
`siteviewer` object

Map for visualization of surface data, specified as a `siteviewer` object.[5]

Data Types: `char` | `string`

Version History

Introduced in R2017b

See Also

`show`

[5] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

link

Display or compute communication link status

Syntax

```
link(rx,tx)
link(rx,tx,propmodel)
link(___,Name,Value)
status = link(___)
```

Description

`link(rx,tx)` displays a one-way point-to-point communication link between a receiver site and transmitter site in the current Site Viewer. The plot is color coded to identify the link success status.

`link(rx,tx,propmodel)` displays the communication link based on the specified propagation model.

`link(___,Name,Value)` displays a communication link using additional options specified by `Name,Value` pairs.

`status = link(___)` returns the success status of the communication link as `true` or `false`.

Examples

Communication Link Between Geographic Transmitter and Receiver

Create a transmitter site.

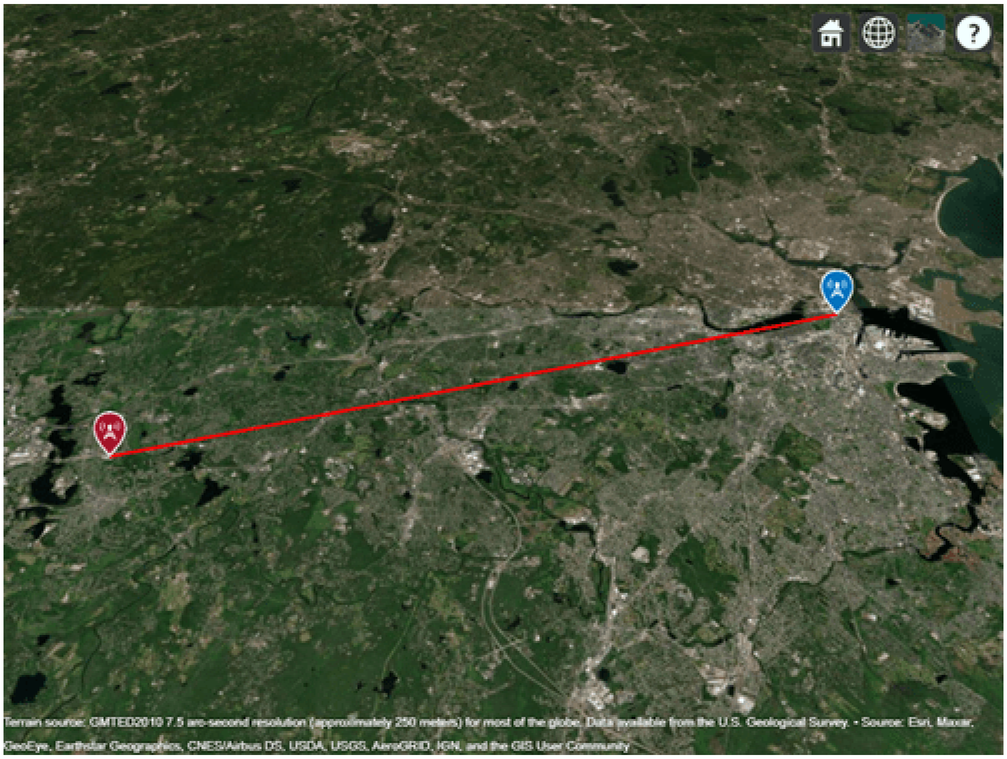
```
tx = txsite("Name","MathWorks", ...
        "Latitude",42.3001, ...
        "Longitude",-71.3503);
```

Create a receiver site with a sensitivity defined in dBm.

```
 rx = rxsite("Name","Boston", ...
        "Latitude",42.3601, ...
        "Longitude",-71.0589, ...
        "ReceiverSensitivity",-90);
```

Plot the communication link between the transmitter and the receiver.

```
link(rx,tx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Communication Link Between Cartesian Transmitter and Receiver

Import and view an STL file. The file models a small conference room with one table and four chairs.

```
viewer = siteviewer('SceneModel','conferenceroom.stl');
```

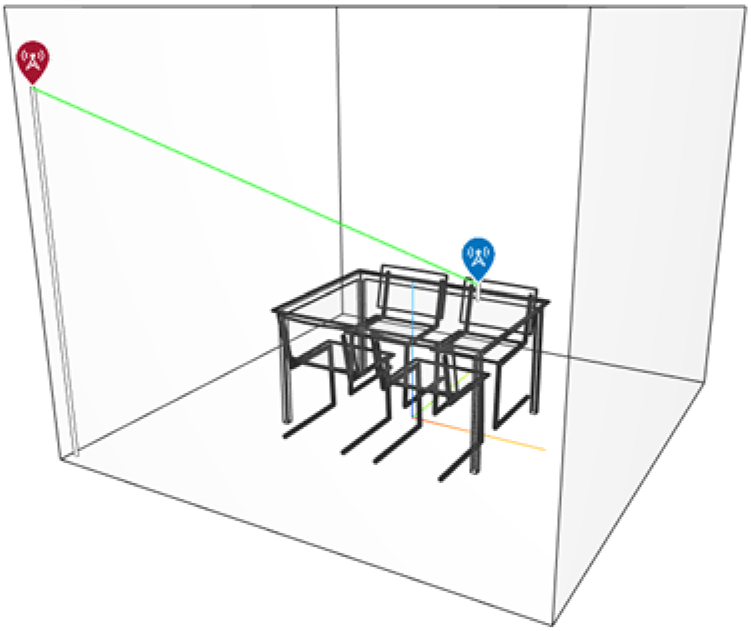
Create a transmitter site near the upper corner of the room and a receiver site above the table. Specify the position using Cartesian coordinates in meters.

```
tx = txsite('cartesian', ...
    'AntennaPosition',[-1.46; -1.42; 2.1]);
rx = rxsite('cartesian', ...
    'AntennaPosition',[0.3; 0.3; 0.85]);
```

Plot the communication link between the transmitter and the receiver.

```
link(rx,tx)
```

Pan by left-clicking, zoom by right-clicking or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking and dragging.



Input Arguments

rx — Receiver site
`rxsite` object | array of `rxsite` objects

Receiver site, specified as a `rxsite` object. You can use array inputs to specify multiple sites.

tx — Transmitter site
`txsite` object | array of `txsite` objects

Transmitter site, specified as a `txsite` object. You can use array inputs to specify multiple sites.

propmodel — Propagation model to use for path loss calculations
`"longley-rice"` (default) | `"freespace"` | `"close-in"` | `"rain"` | `"gas"` | `"fog"` | `"raytracing"` | propagation model created with `propagationModel`

Propagation model to use for the path loss calculations, specified as one of these options:

- `"freespace"` — Free space propagation model
- `"rain"` — Rain propagation model
- `"gas"` — Gas propagation model
- `"fog"` — Fog propagation model
- `"close-in"` — Close-in propagation model
- `"longley-rice"` — Longley-Rice propagation model
- `"tirem"` — TIREM propagation model

- "raytracing" — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the propagationModel function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
"geographic"	• "longley-rice" when you use a terrain. • "freespace" when you do not use a terrain.
"cartesian"	• "freespace" when Map is set to none. • "raytracing" when Map is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including "longley-rice" and "tirem", are only supported for sites with a CoordinateSystem value of "geographic".

You can also specify the propagation model by using the PropagationModel name-value pair argument.

Name-Value Pair Arguments

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

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

Example: "Type","power"

PropagationModel — Propagation model to use for path loss calculations

"freespace" | "close-in" | "rain" | "gas" | "fog" | "longley-rice" | "raytracing" | propagation model created with propagationModel

Propagation model to use for the path loss calculations, specified as one of these options:

- "freespace" — Free space propagation model
- "rain" — Rain propagation model
- "gas" — Gas propagation model
- "fog" — Fog propagation model
- "close-in" — Close-in propagation model
- "longley-rice" — Longley-Rice propagation model
- "tirem" — TIREM propagation model
- "raytracing" — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the propagationModel function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
"geographic"	• "longley-rice" when you use a terrain. • "freespace" when you do not use a terrain.
"cartesian"	• "freespace" when Map is set to none. • "raytracing" when Map is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including "longley-rice" and "tirem", are only supported for sites with a CoordinateSystem value of "geographic".

Data Types: char | string

SuccessColor — Color of successful links

"green" (default) | RGB triplet | character vector | string scalar

Color of successful links, specified as one of these options:

- An RGB triplet whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A character vector such as "red" or "r".
- A string scalar such as "red" or "r".

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
"red"	"r"	[1 0 0]	🟥
"green"	"g"	[0 1 0]	🟩
"blue"	"b"	[0 0 1]	🟦
"cyan"	"c"	[0 1 1]	🟦
"magenta"	"m"	[1 0 1]	🟪
"yellow"	"y"	[1 1 0]	🟨
"black"	"k"	[0 0 0]	⬛
"white"	"w"	[1 1 1]	⬜

Data Types: char | string | double

FailColor — Color of unsuccessful links

"red" (default) | RGB triplet | character vector | string scalar

Color of unsuccessful links, specified as one of these options:

- An RGB triplet whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A character vector such as "red" or "r".

- A string scalar such as `"red"` or `"r"`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	🟥
`"green"`	`"g"`	`[0 1 0]`	🟩
`"blue"`	`"b"`	`[0 0 1]`	🟦
`"cyan"`	`"c"`	`[0 1 1]`	🟦
`"magenta"`	`"m"`	`[1 0 1]`	🟪
`"yellow"`	`"y"`	`[1 1 0]`	🟨
`"black"`	`"k"`	`[0 0 0]`	⬛
`"white"`	`"w"`	`[1 1 1]`	⬜

Data Types: `char` | `string` | `double`

Map — Map for visualization or surface data

`siteviewer` object | `triangulation` object | string scalar | character vector

Map for visualization or surface data, specified as a `siteviewer` object, a `triangulation` object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
`"geographic"`	• A `siteviewer` object[a]. • A terrain name, if the function is called with an output argument. Valid terrain names are `"none"`, `"gmted2010"`, or the name of the custom terrain data added using `addCustomTerrain`.	• The current `siteviewer` object or a new `siteviewer` object if none are open. • `"gmted2010"`, if the function is called with an output.
`"cartesian"`	• `"none"`. • A `siteviewer` object. • The name of an STL file. • A `triangulation` object.	• `"none"`.

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: `char` | `string`

Output Arguments

`status` — Success status of communication link

M-by-*N* array

Success status of communication links, returned as an *M*-by-*N* arrays. *M* is the number of transmitter sites and *N* is the number of receiver sites.

Version History

Introduced in R2017b

Ray tracing functions consider multipath interference

Behavior changed in R2022b

When calculating received power using ray tracing models, the `link` function now incorporates multipath interference by using a phasor sum. In previous releases, the function used a power sum. As a result, the calculations in R2022b are more accurate than in previous releases.

`"raytracing"` propagation models use SBR method

Behavior changed in R2021b

Starting in R2021b, when you use the `link` function and specify the `propmodel` argument or `PropagationModel` name-value argument as `"raytracing"`, the function uses the shooting and bouncing rays (SBR) method and calculates up to two reflections. In previous releases, the `link` function uses the image method and calculates up to one reflection.

To display or compute communication link status using the image method instead, create a propagation model by using the `propagationModel` function. Then, use the `link` function with the propagation model as input. This example shows how to update your code.

```
pm = propagationModel("raytracing","Method","image");
link(rx,tx,pm)
```

For information about the SBR and image methods, see “Choose a Propagation Model”.

Starting in R2021b, all RF Propagation functions use the SBR method by default and calculate up to two reflections. For more information, see “Default modeling method is shooting and bouncing rays method” on page 6-129.

See Also

`sigstrength` | `coverage` | `sinr` | `los` | `propagationModel`

location

Coordinates at distance and angle from site

Syntax

```
sitelocation = location(site)
[lat,lon] = location(site)
[ ___ ] = location(site,distance,azimuth)
```

Description

`sitelocation = location(site)` returns the site location of the antenna.

`[lat,lon] = location(site)` returns the latitude and longitude of the antenna site.

This syntax only supports antenna sites with a `CoordinateSystem` property value of `'geographic'`.

`[ ___ ] = location(site,distance,azimuth)` returns the new location achieved by moving the antenna site by the distance specified in the direction of the azimuth angle. The location is calculated by moving along a great circle path using a spherical Earth model.

This syntax only supports antenna sites with a `CoordinateSystem` property value of `'geographic'`.

Examples

Location of Antenna Site

Create a site 1 km north of a given site.

Create the first transmitter site.

```
tx = txsite('Name','MathWorks', ...
       'Latitude',42.3001, ...
       'Longitude',-71.3504);
```

Calculate the location 1 km north of the first site.

```
[lat,lon] = location(tx,1000,90)
```

```
lat = 42.3091
```

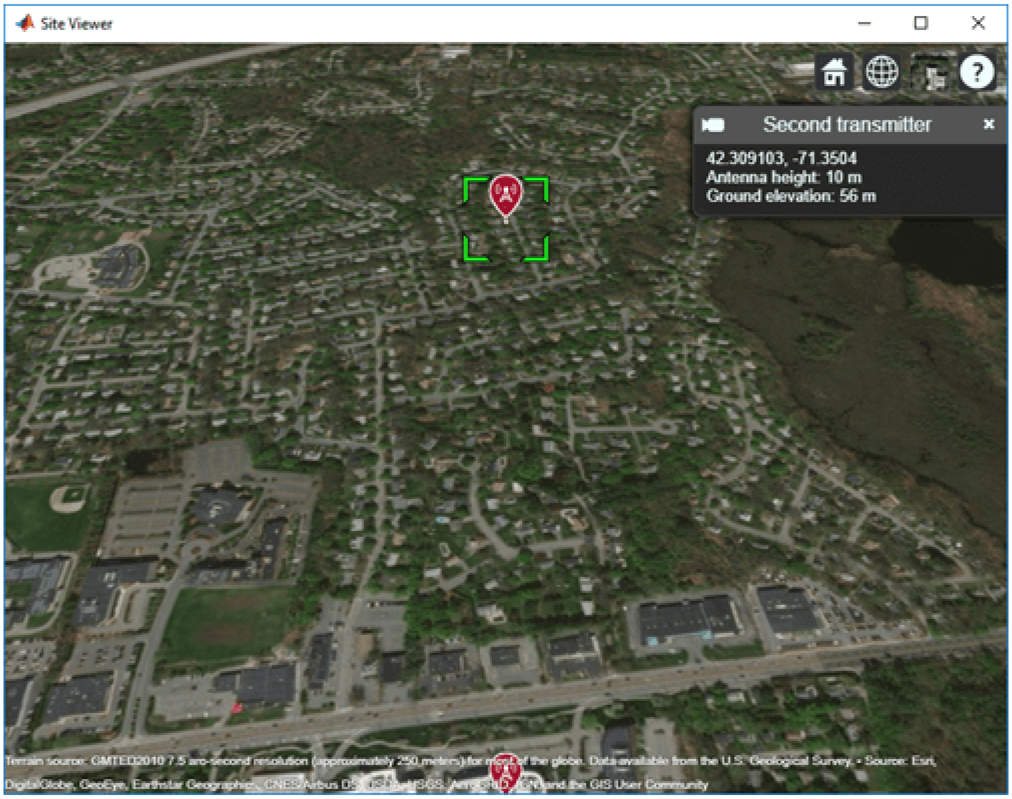
```
lon = -71.3504
```

Create a second transmitter site at the location specified by `lat` and `lon`.

```
tx2 = txsite('Name','Second transmitter', ...
       'Latitude',lat, ...
       'Longitude',lon);
```

Show the two transmitter sites.

```
show([tx,tx2])
```



Second transmitter

42.309103, -71.3504
Antenna height: 10 m
Ground elevation: 56 m

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

`site` — Antenna site

scalar | array

Antenna site, specified as a scalar or an array. It is either a txsite or a rxsite object. For more information, see `txsite`, and `rxsite`

Note If `distance` or `azimuth` is a vector, then site must be a scalar.

`distance` — Distance to move antenna site

scalar | vector

Distance to move antenna site, specified as a scalar or vector in meters.

`azimuth` — Azimuth angle

scalar | vector

Azimuth angle, specified as a scalar or vector in degrees. Azimuth angle is measured counterclockwise from due east.

Output Arguments

`sitelocation` — Location of antenna site

M-by-2 matrix

Location of antenna site, returned as an *M*-by-`2` matrix with each element unit in degrees. *M* is the number of sites in sites. The location value includes the latitude and longitude of the antenna site.

If the antenna site has the `CoordinateSystem` property set to `'geographic'`, *L* is a 1-by-2 vector in degrees latitude and longitude. The output longitude wrapped so that values are in the range `[-180 180]`. If SITE has `CoordinateSystem` set to `'cartesian'`, *L* is a 1-by-3 vector.

`lat` — Latitude of one or more antenna sites

M-by-1 vector

Latitude of one or more antenna sites, returned as an *M*-by-`1` vector with each element unit in degrees. *M* is the number of sites in `site`.

`lon` — Longitude of one or more antenna sites

M-by-1 matrix

Longitude of one or more antenna sites, returned as an *M*-by-`1` matrix with each element unit in degrees. *M* is the number of sites in `site`. The output is wrapped so that the values are in the range `[-180 180]`.

Version History

Introduced in R2018a

See Also

`distance` | `angle` | `txsite` | `rxsite`

los

Display or compute line-of-sight (LOS) visibility status

Syntax

```
los(site1,site2)
los(site1,site2,Name,Value)
vis = los(site1,site2,Name,Value)
```

Description

`los(site1,site2)` displays the line-of-sight (LOS) visibility from site 1 to site 2 in the current Site Viewer. The plot is color coded to identify the visibility of the points along the line.

`los(site1,site2,Name,Value)` sets properties using one or more name-value pairs. For example, `los(site1,site2,'ObstructedColor','red')` displays the LOS in red to show blocked visibility.

`vis = los(site1,site2,Name,Value)` returns the status of the LOS visibility.

Examples

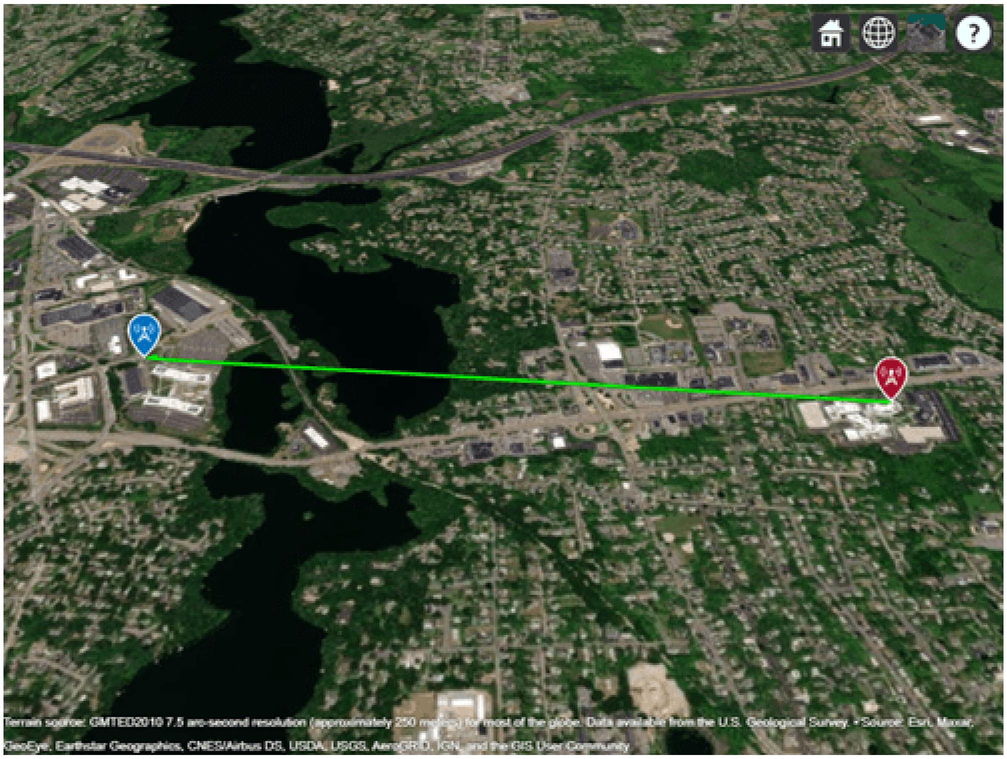
LOS from a Transmitter Site to a Receiver Site

Create a transmitter site with an antenna of height 30 m and a receiver site at ground level.

```
tx = txsite("Name","MathWorks Apple Hill",...
          "Latitude",42.3001,"Longitude",-71.3504,"AntennaHeight",30);
rx = rxsite("Name","MathWorks Lakeside", ...
           "Latitude",42.3021,"Longitude",-71.3764);
```

Plot the LOS between the two sites.

```
los(tx,rx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

LOS from a Transmitter Site to Two Receiver Sites

Create a transmitter site with an antenna of height 30 m and two receiver sites with antennas at ground level.

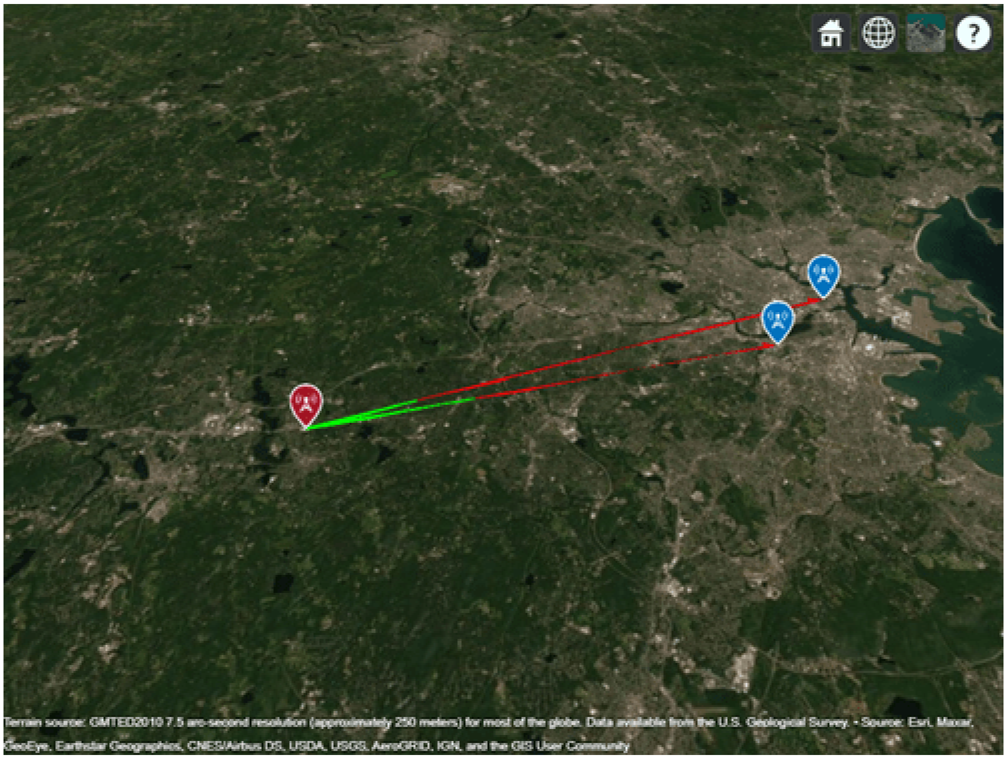
```
tx = txsite("Name","MathWorks Apple Hill",...
          "Latitude",42.3001,"Longitude",-71.3504,"AntennaHeight",30);

names = ["Fenway Park","Bunker Hill Monument"];
lats = [42.3467,42.3763];
lons = [-71.0972,-71.0611];
```

Create the receiver site array.

```
rxs = rxsite("Name", names,...
      "Latitude",lats,...
      "Longitude",lons);
```

Plot the LOSs to the receiver sites. The red portion of the LOS represents obstructed visibility.

```
los(tx,rxs)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

LOS Between Cartesian Sites

Import and view an STL file. The file models a small conference room with one table and four chairs.

```
viewer = siteviewer("SceneModel","conferenceroom.stl");
```

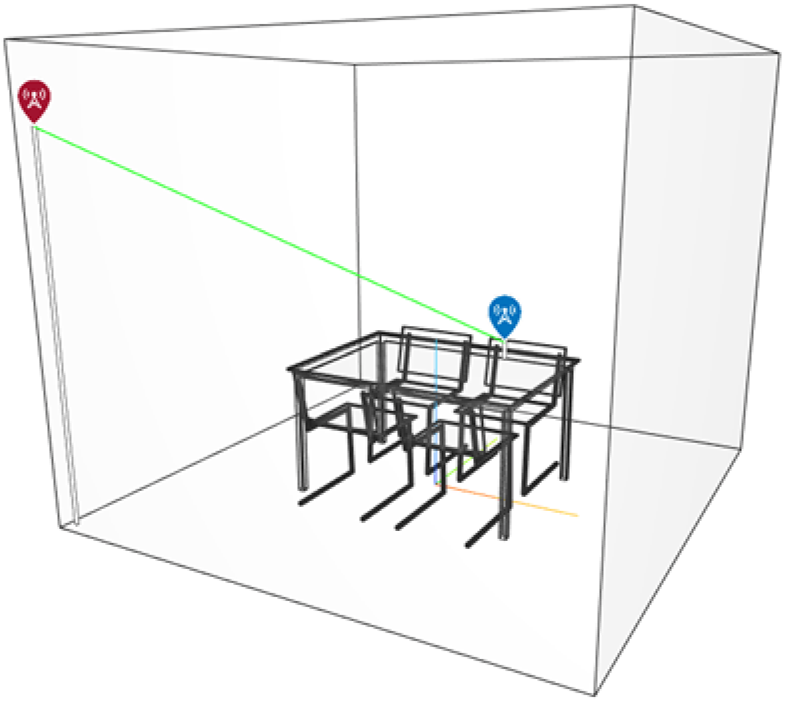
Create a transmitter site near the upper corner of the room and a receiver site above the table. Specify the position using Cartesian coordinates in meters.

```
tx = txsite("cartesian", ...
    "AntennaPosition",[-1.46; -1.42; 2.1]);
rx = rxsite("cartesian", ...
    "AntennaPosition",[0.3; 0.3; 0.85]);
```

Plot the LOS between the transmitter and the receiver.

```
los(rx,tx)
```

Pan by left-clicking, zoom by right-clicking or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking and dragging.

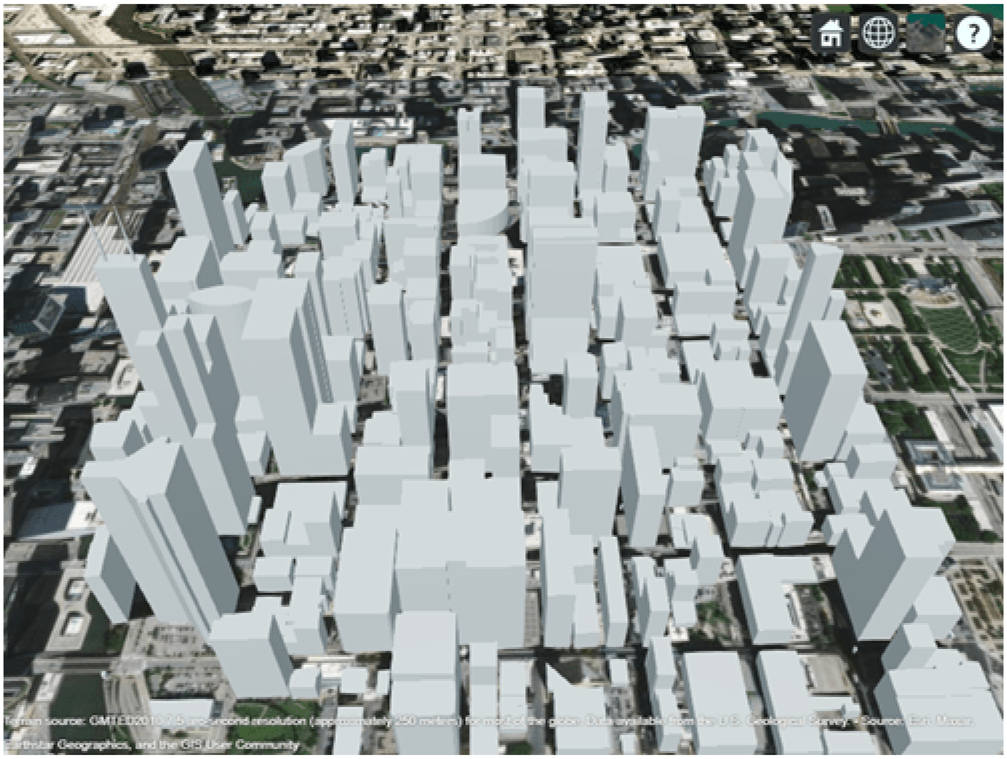


Plot Propagation Rays Between Sites in Chicago

Return ray tracing results in `comm.Ray` objects and plot the ray propagation paths after relaunching the Site Viewer map.

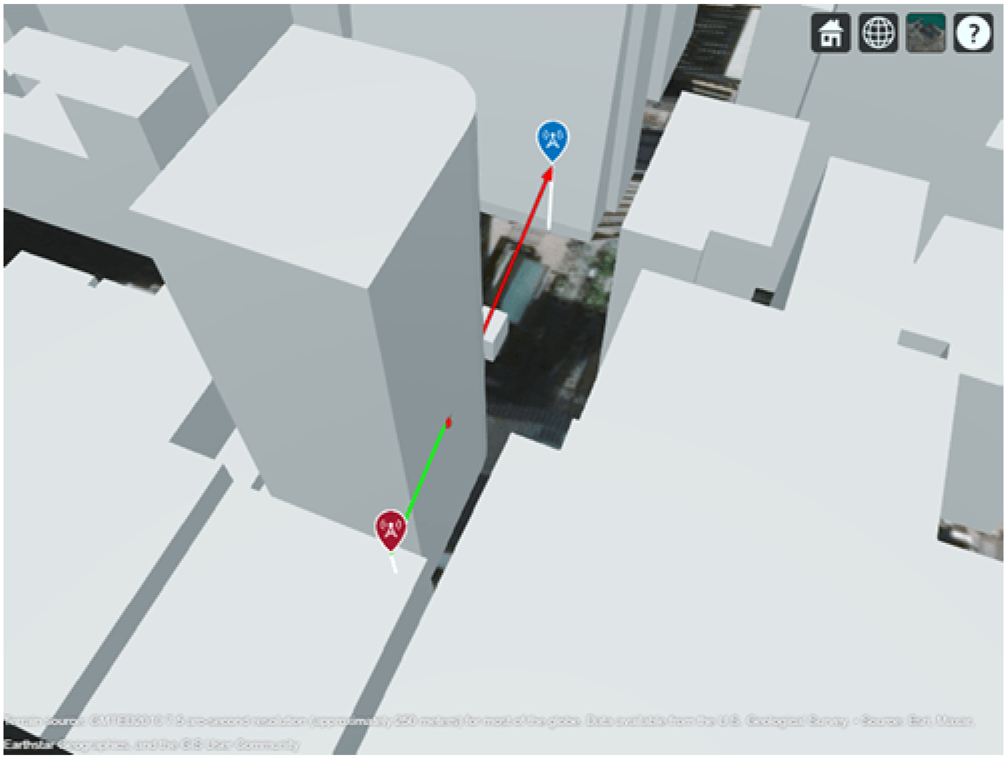
Create a Site Viewer map, loading building data for Chicago. For more information about the osm file, see [1] on page 6-118.

```
viewer = siteviewer("Buildings","chicago.osm");
```


Terrain source: GMTED2010 7.5-arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Create a transmitter site on one building and a receiver site on another building. Use the `los` function to show the line of sight path between the transmitter and receiver sites.

```
tx = txsite( ...
    "Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
rx = rxsite( ...
    "Latitude",41.881352, ...
    "Longitude",-87.629771, ...
    "AntennaHeight",30);
los(tx,rx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Sources: Esri, Maxar,
Earthstar Geographics, and the GIS User Community

Perform ray tracing for up to two reflections. For the configuration defined, ray tracing returns a cell array containing the ray objects. Close the Site Viewer map.

```
pm = propagationModel( ...
    "raytracing", ...
    "Method","sbr", ...
    "MaxNumReflections",2);
rays = raytrace(tx,rx,pm)
```

```
rays = 1×1 cell array
    {1×3 comm.Ray}
```

```
rays{1}(1,1)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×1 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 92.7739
             PhaseShift: 1.2933

   Read-only properties:
       PropagationDelay: 5.7088e-07
    PropagationDistance: 171.1462
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 1
```

```
rays{1}(1,2)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 100.8574
             PhaseShift: 2.9398

   Read-only properties:
       PropagationDelay: 5.9259e-07
    PropagationDistance: 177.6532
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
rays{1}(1,3)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 106.3302
             PhaseShift: 4.6994

   Read-only properties:
       PropagationDelay: 6.3790e-07
    PropagationDistance: 191.2374
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
close(viewer)
```

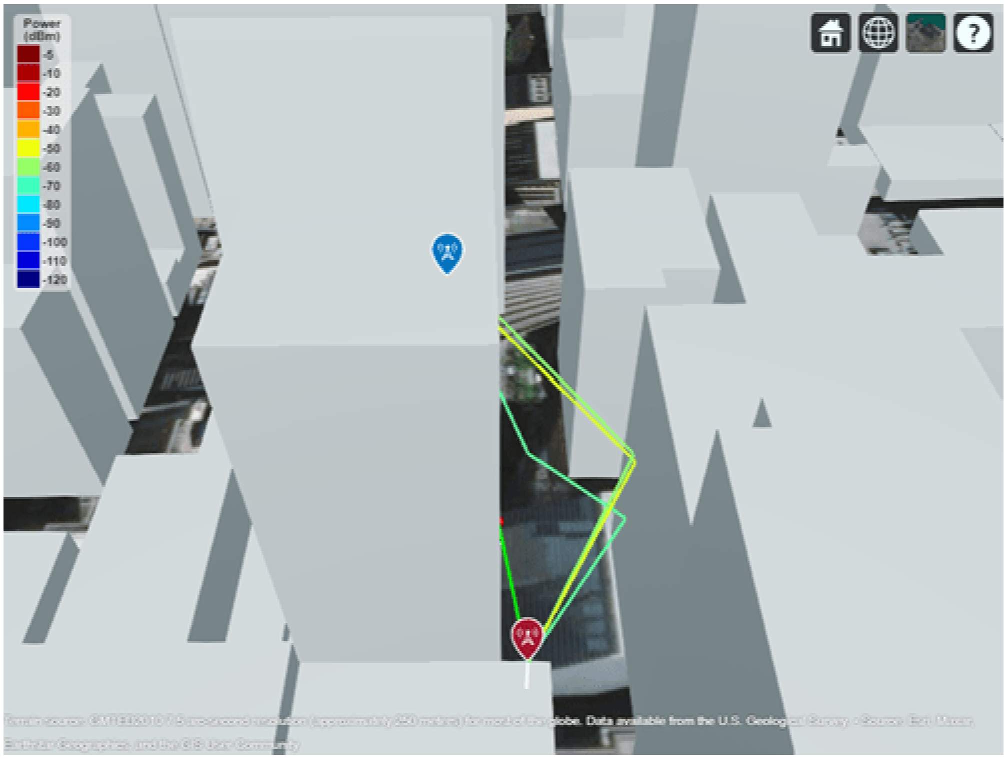
You can plot the rays without performing ray tracing again. Create another Site Viewer map with the same buildings. Show the transmitter and receiver sites. Using the previously returned cell array of ray objects, plot the reflected rays between the transmitter site and the receiver site. The plot

function can plot the path for ray objects collectively or individually. For example, to plot rays for the only second ray object, specify `rays{1}(1,2)`. This figure plot all paths for all the ray objects.

```
siteviewer("Buildings","chicago.osm")
```

```
ans = 
  siteviewer with properties:

                Name: 'Site Viewer'
            Position: [560 240 800 600]
    CoordinateSystem: "geographic"
             Basemap: 'satellite'
             Terrain: 'gmted2010'
           Buildings: 'chicago.osm'
```

```
los(tx,rx)
plot(rays{:},"Type","power", ...
    "TransmitterSite",tx,"ReceiverSite",rx)
```


Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Input Arguments

`site1` — Source antenna site

`txsite` object | `rxsite` object

Source antenna site, specified as a `txsite` object or a `rxsite` object. Site 1 must be a single site object.

`site2` — Target antenna site

`txsite` object | `rxsite` object | vector of `txsite` or `rxsite` objects

Target antenna site, specified as a `txsite` object or a `rxsite` object. Site 2 can be a single site object or a vector of multiple site objects.

Name-Value Pair Arguments

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

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

Example: `'ObstructedColor','blue'`

`VisibleColor` — Plot color for successful visibility

`'green'` (default) | RGB triplet | character vector | string scalar

Plot color for successful visibility, specified as one of the following:

- An RGB triplet whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A character vector such as `'red'` or `'r'`.
- A string scalar such as `"red"` or `"r"`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	🟥
`"green"`	`"g"`	`[0 1 0]`	🟩
`"blue"`	`"b"`	`[0 0 1]`	🟦
`"cyan"`	`"c"`	`[0 1 1]`	🟦
`"magenta"`	`"m"`	`[1 0 1]`	🟪
`"yellow"`	`"y"`	`[1 1 0]`	🟨
`"black"`	`"k"`	`[0 0 0]`	⬛
`"white"`	`"w"`	`[1 1 1]`	⬜

ObstructedColor — Plot color for blocked visibility

'red' (default) | RGB triplet | character vector | string scalar

Plot color for blocked visibility, specified as one of the following:

- An RGB triplet whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A character vector such as 'red' or 'r'.
- A string scalar such as "red" or "r".

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
"red"	"r"	[1 0 0]	🟥
"green"	"g"	[0 1 0]	🟩
"blue"	"b"	[0 0 1]	🟦
"cyan"	"c"	[0 1 1]	🟦
"magenta"	"m"	[1 0 1]	🟪
"yellow"	"y"	[1 1 0]	🟨
"black"	"k"	[0 0 0]	⬛
"white"	"w"	[1 1 1]	⬜

Resolution — Sampling distance between two sites

'auto' (default) | numeric scalar

Resolution of sample locations used to compute line-of-sight visibility, specified as 'auto' or a numeric scalar expressed in meters. Resolution defines the distance between samples on the great circle path using a spherical Earth model. If Resolution is 'auto', the function computes a value based on the distance between the sites.

Map — Map for visualization or surface data

siteviewer object | triangulation object | string scalar | character vector

Map for visualization or surface data, specified as a siteviewer object, a triangulation object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
"geographic"	• A siteviewer object[a]. • A terrain name, if the function is called with an output argument. Valid terrain names are "none", "gmted2010", or the name of the custom terrain data added using addCustomTerrain.	• The current siteviewer object or a new siteviewer object if none are open. • "gmted2010", if the function is called with an output.

Coordinate System	Valid map values	Default map value
"cartesian"	• "none". • A siteviewer object. • The name of an STL file. • A triangulation object.	• "none".

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: char | string

Output Arguments

vis — Status of LOS visibility

true or 1 | false or 0 | *n*-by 1 logical array

Status of LOS visibility, returned as logical 1 (true) or 0 (false). If there are multiple target sites, the function returns a logical array of *n*-by-1.

Version History

Introduced in R2018a

See Also

distance | angle | link

pathloss

Package: rfprop

Path loss of radio wave propagation

Syntax

```
pl = pathloss(propmodel,rx,tx)
pl = pathloss( ___ ,Name,Value)
[pl,info] = pathloss( ___ )
```

Description

pl = pathloss(propmodel,rx,tx) returns the path loss of radio wave propagation at the receiver site from the transmitter site.

pl = pathloss(___ ,Name,Value) returns the path loss using additional options specified by Name,Value pairs.

[pl,info] = pathloss(___) returns the path loss and the information about the propagation paths.

Examples

Path Loss of Receiver In Heavy Rain

Specify the transmitter and the receiver sites.

```
tx = txsite('Name','MathWorks Apple Hill', ...
    'Latitude',42.3001,'Longitude',-71.3504, ...
    'TransmitterFrequency', 2.5e9);

rx = rxsite('Name','Fenway Park', ...
    'Latitude',42.3467,'Longitude',-71.0972);
```

Create the propagation model for heavy rainfall rate.

```
pm = propagationModel('rain','RainRate',50)
```

```
pm = 
  Rain with properties:

    RainRate: 50
        Tilt: 0
```

Calculate the pathloss at the receiver using the rain propagation model.

```
pl = pathloss(pm,rx,tx)
```

```
pl = 127.3208
```

Input Arguments

propmodel — Propagation model

propagation model object

Propagation model, specified as a propagation model object. Use the propagationModel function.

Data Types: object

rx — Receiver site

rxsite object

Receiver site, specified as a rxsite object. You can use array inputs to specify multiple sites.

Data Types: char

tx — Transmitter site

txsite object

Transmitter site, specified as a txsite object. You can use array inputs to specify multiple sites.

Data Types: char

Name-Value Pair Arguments

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

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

Example: 'Map','none'

Map — Map for visualization or surface data

siteviewer object | triangulation object | string scalar | character vector

Map for visualization or surface data, specified as a siteviewer object, a triangulation object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
"geographic"	- A siteviewer object[a]. - A terrain name, if the function is called with an output argument. Valid terrain names are "none", "gmted2010", or the name of the custom terrain data added using addCustomTerrain.	- The current siteviewer object or a new siteviewer object if none are open. - "gmted2010", if the function is called with an output.

Coordinate System	Valid map values	Default map value
"cartesian"	• "none". • A siteviewer object. • The name of an STL file. • A triangulation object.	• "none".

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: char | string

Output Arguments

pl — Path loss

scalar | *M*-by-*N* arrays

Path loss, returned as a scalar or *M*-by-*N* cell arrays containing a row vector of path loss values in decibels. *M* is the number of TX sites and *N* is the number of RX sites.

Path loss is computed along the shortest path shortest path through space connecting the transmitter and receiver antenna centers.

For terrain propagation models, path loss is computed using terrain elevation profile that is computed at sample locations on the great circle path between the transmitter and the receiver. If Map is a siteviewer object with buildings specified, the terrain elevation is adjusted to include the height of the buildings.

info — Information corresponding to each propagation path

M-by-*N* structure array | *M*-by-*N* cell array containing vector of structures in each cell

Information corresponding to each propagation path, returned as a *M*-by-*N* cell array containing vector of structures in each cell for ray tracing propagation models and *M*-by-*N* structure array for all other propagation models. The field and values for the structures are:

- PropagationDistance — Total distance of propagation path returned as a double scalar in meters.
- AngleOfDeparture — Angle of departure of signal from transmitter site antenna returned as a 2-by-1 double vector of azimuth and elevation angles in degrees.
- AngleOfArrival — Angle of arrival of signal at receiver site antenna returned as a 2-by-1 double vector of azimuth and elevation angles in degrees.
- NumReflections — Number of reflections undergone by signal along propagation path, returned specified as 0, 1, or 2. This field and value is only for ray tracing propagation models.

Angle values in this structure are defined using the local East-North-Up coordinate system of the antenna when CoordinateSystem is set to geographic. Angle values in this structure are defined using global Cartesian coordinate system when CoordinateSystem is set to cartesian. Azimuth angle is measured either from east (when 'geographic') or from the global x-axis around the global z-axis (when 'cartesian'). Elevation angle is measured from the horizontal (or X-Y) plane to the x-axis of the antenna in the range -90 to 90.

Version History

Introduced in R2017b

Ray tracing models using SBR method find paths with exact geometric accuracy

Behavior changed in R2022b

Ray tracing models that find propagation paths by using the shooting and bouncing rays (SBR) method correct the results so that the geometric accuracy of each path is exact, using single-precision floating-point computations. In previous releases, the paths have approximate geometric accuracy.

As a result, when you use a ray tracing model as input to the `pathloss` function, the function can return different results than in previous releases.

See Also

`propagationModel` | `range`

propagationModel

Create RF propagation model

Syntax

```
pm = propagationModel(modelname)
pm = propagationModel( ___ ,Name,Value)
```

Description

`pm = propagationModel(modelname)` creates an RF propagation model for the specified model.

`pm = propagationModel( ___ ,Name,Value)` specifies options using name-value arguments. For example, `pm = propagationModel('rain','RainRate',96)` creates a rain propagation model with a rain rate of 96 mm/h.

Examples

Signal Strength of Receiver in Heavy Rain

Specify transmitter and receiver sites.

```
tx = txsite('Name','MathWorks Apple Hill',...
       'Latitude',42.3001, ...
       'Longitude',-71.3504, ...
       'TransmitterFrequency', 2.5e9);

rx = rxsite('Name','Fenway Park',...
       'Latitude',42.3467, ...
       'Longitude',-71.0972);
```

Create the propagation model for a heavy rainfall rate.

```
pm = propagationModel('rain','RainRate',50)
```

```
pm = 
  Rain with properties:

    RainRate: 50
        Tilt: 0
```

Calculate the signal strength at the receiver using the rain propagation model.

```
ss = sigstrength(rx,tx,pm)
```

```
ss = -87.1559
```

Longley-Rice Propagation Model

Create a default transmitter site.

```
tx = txsite;
```

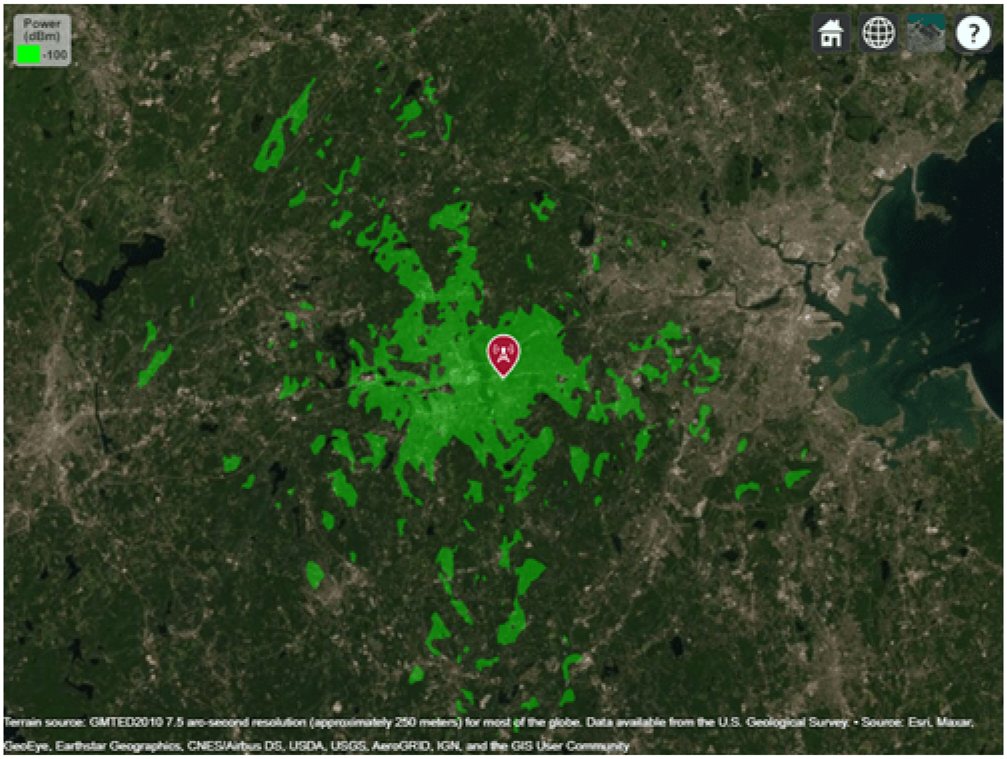
Create a Longley-Rice propagation model by using the `propagationModel` function.

```
pm = propagationModel("longley-rice","TimeVariabilityTolerance",0.7)
```

```
pm = 
  LongleyRice with properties:

              AntennaPolarization: 'horizontal'
               GroundConductivity: 0.0050
               GroundPermittivity: 15
          AtmosphericRefractivity: 301
                      ClimateZone: 'continental-temperate'
         TimeVariabilityTolerance: 0.7000
    SituationVariabilityTolerance: 0.5000
```

Find the coverage of the transmitter site by using the defined propagation model.

```
coverage(tx,"PropagationModel",pm)
```


Power (dBm)
🟩 -100
?
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

`modelname` — Name of propagation model

`'freespace'` | `'rain'` | `'gas'` | `'fog'` | `'close-in'` |`'longley-rice'` | `'tirem'` | `'raytracing'`

Name of propagation model, specified as one of these:

- `'freespace'` — Free space propagation model.
- `'rain'` — Rain propagation model. For more information, see [3].
- `'gas'` — Gas propagation model. For more information, see [6].
- `'fog'` — Fog propagation model. For more information, see [2].
- `'close-in'` — Close-in propagation model typically used in urban macro-cell scenarios. For more information, see [1].

  Note The close-in model implements a statistical path loss model and can be configured for different scenarios. The default values correspond to an urban macro-cell scenario in a non-line-of-sight (NLOS) environment.
- `'longley-rice'` — Longley-Rice propagation model. This model is also known as Irregular Terrain Model (ITM). You can use this model to calculate point-to-point path loss between sites over an irregular terrain, including buildings. Path loss is calculated from free-space loss, terrain diffraction, ground reflection, refraction through atmosphere, tropospheric scatter, and atmospheric absorption. For more information and list of limitations, see [4].

  Note The Longley-Rice model implements the point-to-point mode of the model, which uses terrain data to predict the loss between two points.
- `'tirem'` — Terrain Integrated Rough Earth Model (TIREM). You can use this model to calculate point-to-point path loss between sites over an irregular terrain, including buildings. Path loss is calculated from free-space loss, terrain diffraction, ground reflection, refraction through atmosphere, tropospheric scatter, and atmospheric absorption. This model needs access to an external TIREM library. The actual model is valid from 1 MHZ to 1000 GHz. But with Antenna Toolbox elements and arrays the frequency range is limited to 200 GHz.
- `'raytracing'` — A multipath propagation model that uses ray tracing analysis to compute propagation paths and corresponding path losses. Path loss is calculated from free-space loss, reflection loss due to material, and antenna polarization loss. You can perform ray tracing analysis using the shooting and bouncing rays (SBR) method or the image method. Specify a method using the `'Method'` property. The SBR method includes effects from surface reflections but does not include effects from diffraction, refraction, or scattering. The image method considers surface reflection only. Both ray tracing methods are valid for a frequency range of 100 MHz to 100 GHz. For information about differences between the image and SBR methods, see “Choose a Propagation Model”. Use the `raytrace` function to compute and plot the propagation paths between the sites.

Data Types: `char`

Name-Value Pair Arguments

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

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

Example: `propagationModel("rain","RainRate",50)` sets the rate of rainfall in the rain propagation model to 50 millimeters per hour.

Each type of propagation model object supports a different set of properties. For a full list of the properties and their descriptions for a propagation model type, see the associated object page.

Type of Propagation Model	Object Page
`'freespace'`	`FreeSpace`
`'rain'`	`Rain`
`'gas'`	`Gas`
`'fog'`	`Fog`
`'close-in'`	`CloseIn`
`'longley-rice'`	`LongleyRice`
`'tirem'`	`TIREM`
`'raytracing'`	`RayTracing`

Output Arguments

pm — Propagation model
`FreeSpace` object | `Rain` object | `Gas` object | `Fog` object | `CloseIn` object | ...

Propagation model, returned as a `FreeSpace`, `Rain`, `Gas`, `Fog`, `CloseIn`, `LongleyRice`, `TIREM`, or `RayTracing` object.

Version History

Introduced in R2017b

Default modeling method is shooting and bouncing rays method

Behavior changed in R2021b

Starting in R2021b, when you create a propagation model using the syntax `propagationModel('raytracing')`, MATLAB returns a `RayTracing` model with the `Method` value set to `'sbr'` and two reflections (instead of `'image'` and one reflection as in previous releases).

To create ray tracing propagation models that use the image method, use the syntax `propagationModel('raytracing','Method','image')`.

`propagationModel('raytracing-image-method')` syntax will be removed in a future release

Warns starting in R2022a

The `propagationModel('raytracing-image-method')` syntax will be removed in a future release.

Use the `propagationModel('raytracing','Method','image')` syntax to use the image ray tracing method. Alternatively, using the `propagationModel('raytracing')` syntax sets the ray tracing method to SBR.

References

[1] Sun, Shu, Theodore S. Rappaport, Timothy A. Thomas, Amitava Ghosh, Huan C. Nguyen, Istvan Z. Kovacs, Ignacio Rodriguez, Ozge Koymen, and Andrzej Partyka. “Investigation of Prediction Accuracy, Sensitivity, and Parameter Stability of Large-Scale Propagation Path Loss Models for 5G Wireless Communications.” *IEEE Transactions on Vehicular Technology* 65, no. 5 (May 2016): 2843–60. https://doi.org/10.1109/TVT.2016.2543139.

[2] International Telecommunications Union Radiocommunication Sector. *Attenuation due to clouds and fog*. Recommendation P.840-6. ITU-R, approved September 30, 2013. https://www.itu.int/rec/R-REC-P.840-6-201309-S/en.

[3] International Telecommunications Union Radiocommunication Sector. *Specific attenuation model for rain for use in prediction methods*. Recommendation P.838-3. ITU-R, approved March 8, 2005. https://www.itu.int/rec/R-REC-P.838-3-200503-I/en.

[4] Hufford, George A., Anita G. Longley, and William A.Kissick. *A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode*. NTIA Report 82-100. National Telecommunications and Information Administration, April 1, 1982.

[5] Seybold, John S. *Introduction to RF Propagation*. Hoboken, N.J: Wiley, 2005.

[6] International Telecommunications Union Radiocommunication Sector. *Attenuation by atmospheric gases*. Recommendation P.676-11. ITU-R, approved September 30, 2016. https://www.itu.int/rec/R-REC-P.676-11-201609-S/en.

[7] International Telecommunications Union Radiocommunication Sector. *Effects of building materials and structures on radiowave propagation above about 100MHz.* Recommendation P.2040-1. ITU-R, approved July 29, 2015. https://www.itu.int/rec/R-REC-P.2040-1-201507-I/en.

[8] International Telecommunications Union Radiocommunication Sector. *Electrical characteristics of the surface of the Earth*. Recommendation P.527-5. ITU-R, approved August 14, 2019. https://www.itu.int/rec/R-REC-P.527-5-201908-I/en.

[9] Yun, Zhengqing, and Magdy F. Iskander. “Ray Tracing for Radio Propagation Modeling: Principles and Applications.” *IEEE Access* 3 (2015): 1089–1100. https://doi.org/10.1109/ACCESS.2015.2453991.

[10] Schaubach, K.R., N.J. Davis, and T.S. Rappaport. “A Ray Tracing Method for Predicting Path Loss and Delay Spread in Microcellular Environments.” In *[1992 Proceedings] Vehicular Technology Society 42nd VTS Conference - Frontiers of Technology*, 932–35. Denver, CO, USA: IEEE, 1992. https://doi.org/10.1109/VETEC.1992.245274.

See Also

Functions

`sigstrength` | `coverage` | `link` | `sinr` | `range` | `los` | `pathloss` | `tirempl` | `tiremSetup` | `raytrace`

Topics

"Access TIREM Software"
"Choose a Propagation Model"

range

Package: `rfprop`

Range of radio wave propagation

Syntax

```
r = range(propmodel,tx,pl)
```

Description

`r = range(propmodel,tx,pl)`returns the range of radio wave propagation from the transmitter site.

Examples

Range of Transmitter In Heavy Rain

Specify transmitter and receiver sites.

```
tx = txsite('Name','MathWorks Apple Hill',...
       'Latitude',42.3001, ...
       'Longitude',-71.3504, ...
       'TransmitterFrequency', 2.5e9);

rx = rxsite('Name','Fenway Park',...
       'Latitude',42.3467, ...
       'Longitude',-71.0972);
```

Create the propagation model for heavy rainfall rate.

```
pm = propagationModel('rain','RainRate',50)
```

```
pm = 
  Rain with properties:

    RainRate: 50
        Tilt: 0
```

Calculate the range of transmitter using the rain propagation model and a path loss of 127 dB.

```
r = range(pm,tx,127)
```

```
r = 2.0747e+04
```

Input Arguments

`propmodel` — Propagation model
propagation model object

Propagation model, specified as a propagation model object. Use the `propagationModel` function.

Data Types: `object`

`tx` — Transmitter site
`txsite` object

Transmitter site, specified as a `txsite` object. You can use array inputs to specify multiple sites.

Data Types: `char`

`pl` — Path loss
scalar

Path loss, specified as a scalar in decibels.

Data Types: `double`

Output Arguments

`r` — range
scalar | *M*-by-1 arrays

Range, returned as a scalar or *M*-by-1 array with each element in meters. *M* is the number of TX sites.

Range is the maximum distance for which the path loss does not exceed the value of the specified `pl`.

Version History

Introduced in R2017b

See Also

`propagationModel` | `pathloss`

removeCustomTerrain

Remove custom terrain data

Syntax

```
removeCustomTerrain(terrainName)
```

Description

`removeCustomTerrain(terrainName)` removes the custom terrain data specified by the user-defined `terrainName`. You can use this function to remove terrain data that is no longer needed. The terrain data to be removed must have been previously added using `addCustomTerrain`.

Examples

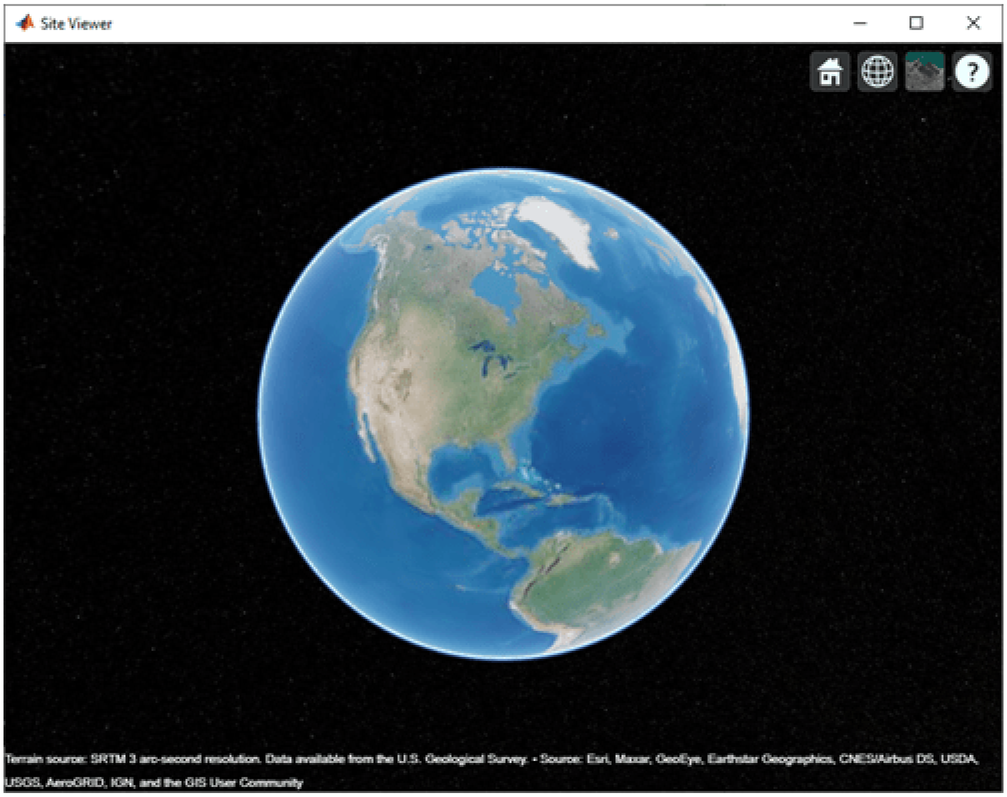
Site Viewer Maps Using Custom Terrain

Add terrain for a region around Boulder, CO. The DTED file was downloaded from the "SRTM Void Filled" data set available from the U.S. Geological Survey.

```
dtedfile = "n39_w106_3arc_v2.dt1";
attribution = "SRTM 3 arc-second resolution. Data available " + ...
   "from the U.S. Geological Survey.";
addCustomTerrain("southboulder",dtedfile,"Attribution",attribution)
```

Use the custom terrain name in Site Viewer.

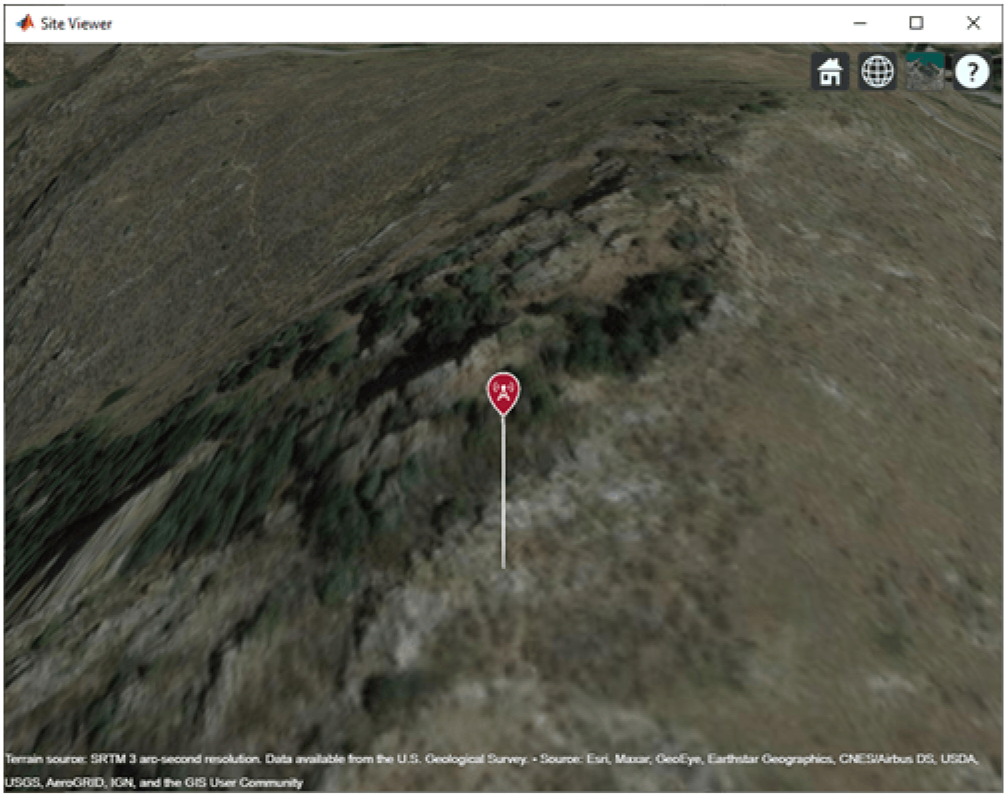
```
viewer = siteviewer("Terrain","southboulder");
```



Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create a site with the terrain region.

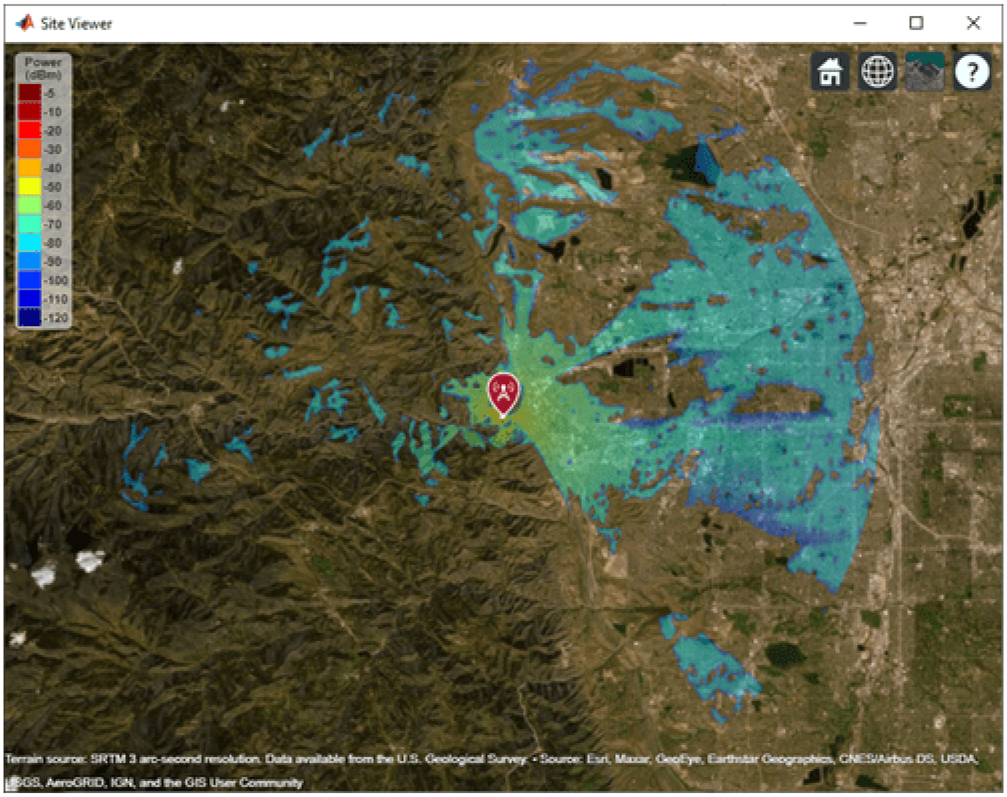
```
mtzion = txsite("Name","Mount Zion", ...
   "Latitude",39.74356, ...
   "Longitude",-105.24193, ...
   "AntennaHeight", 30);
show(mtzion)
```



Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create a coverage map of the area within 20 km of the transmitter site.

```
coverage(mtzion, ...
   "MaxRange",20000, ...
   "SignalStrengths",-100:-5)
```


Site Viewer
Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Remove the custom terrain.

```
close(viewer)
removeCustomTerrain("southboulder")
```

Input Arguments

`terrainName` — User-defined identifier for terrain data

string scalar | character vector

User-defined identifier for terrain data previously added using `addCustomTerrain`, specified as a string scalar or a character vector.

Data Types: `char` | `string`

Version History

Introduced in R2019a

See Also

`addCustomTerrain` | `siteviewer`

pattern

Display antenna radiation pattern in Site Viewer

Syntax

```
pattern(tx)
pattern(rx,frequency)
pattern( ___ ,Name,Value)
```

Description

`pattern(tx)` displays the 3-D antenna radiation pattern for the transmitter site `txsite` in the current Site Viewer. Signal gain value (dBi) in a particular direction determines the color of the pattern.

`pattern(rx,frequency)` displays the 3-D radiation pattern for the receiver site `rxsite` for the specified `frequency`.

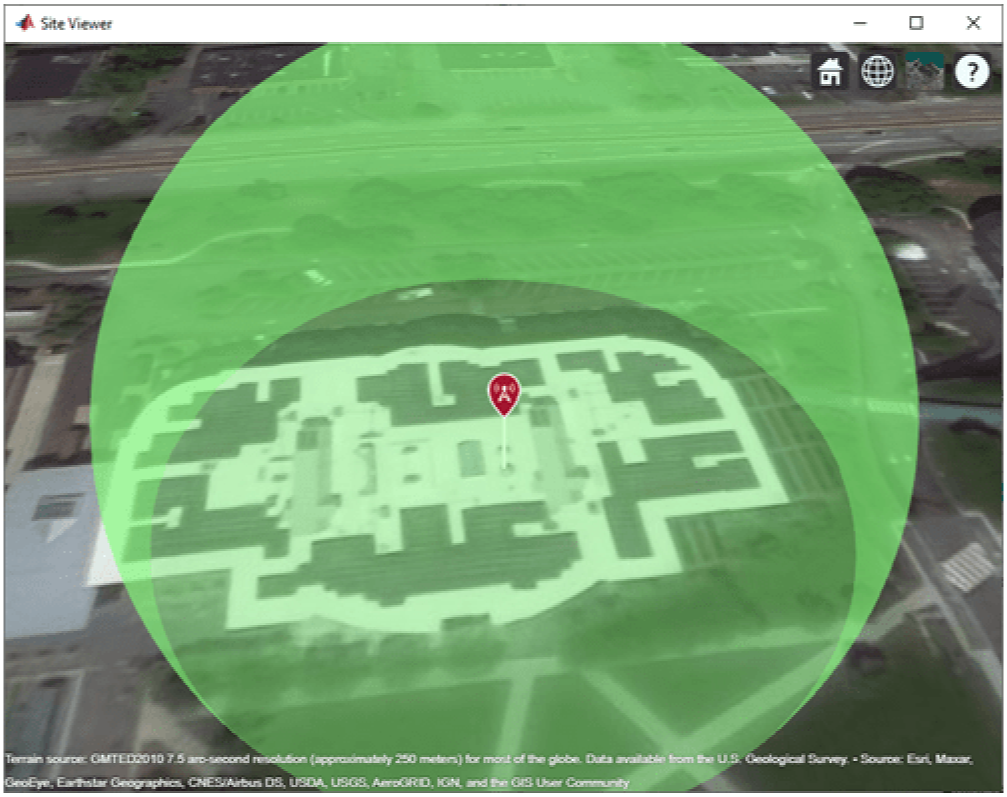
`pattern( ___ ,Name,Value)` displays the 3-D radiation pattern with additional options specified by name-value pair arguments.

Examples

Single Transmitter Site Pattern

Define and visualize the radiation pattern of a single transmitter site.

```
tx = txsite;
pattern(tx)
```


Site Viewer
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

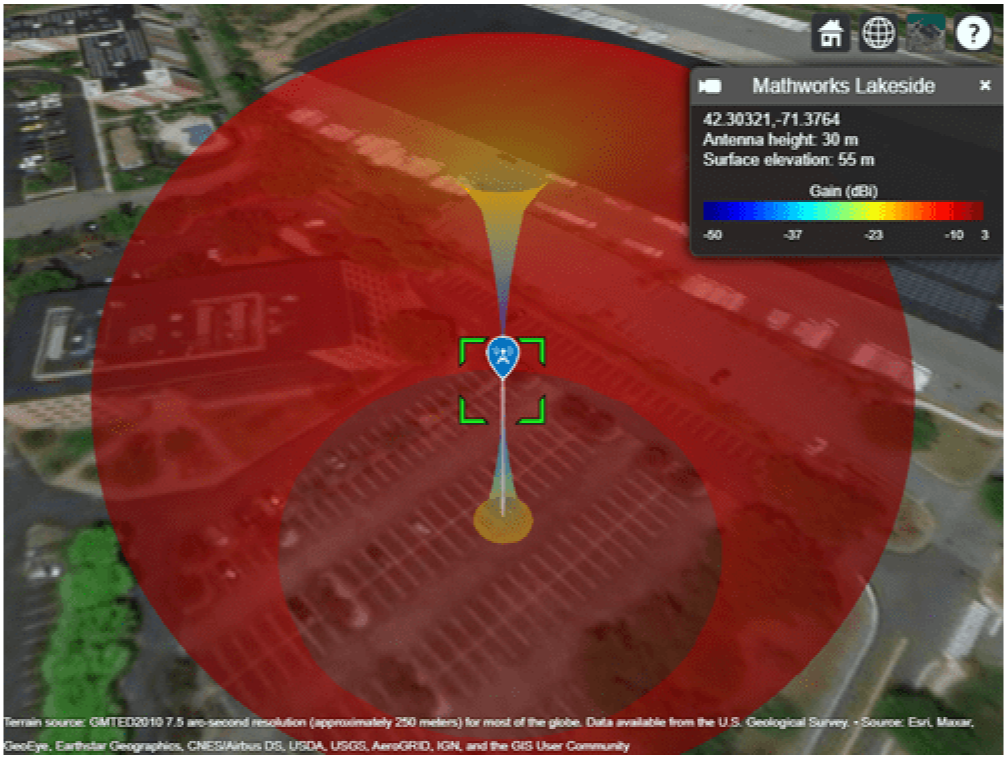
Single Receiver Site Pattern

Design a receiver site using a dipole antenna at a height of 30 meters.

```
d = dipole;
rx = rxsite("Name","Mathworks Lakeside", ...
    "Latitude",42.30321,"Longitude",-71.3764, ...
    "Antenna",d,"AntennaHeight",30);
```

Visualize the pattern of the receiver site at 75 MHz.

```
pattern(rx,75e6)
```


Mathworks Lakeside
42.30321,-71.3764
Antenna height: 30 m
Surface elevation: 55 m
Gain (dBi)
-50
-37
-23
-10
3
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Pattern for Directional Transmitter and Receiver

Create a directional antenna.

```
yagiAntenna = design(yagiUda,4.5e9);
yagiAntenna.Tilt = 90;
yagiAntenna.TiltAxis = 'y';
```

Create transmitter and receiver sites at a frequency of 4.5 GHz. Use the Yagi antenna as the transmitter antenna. Design a dipole at 4.5 GHz and use this as the receiver antenna.

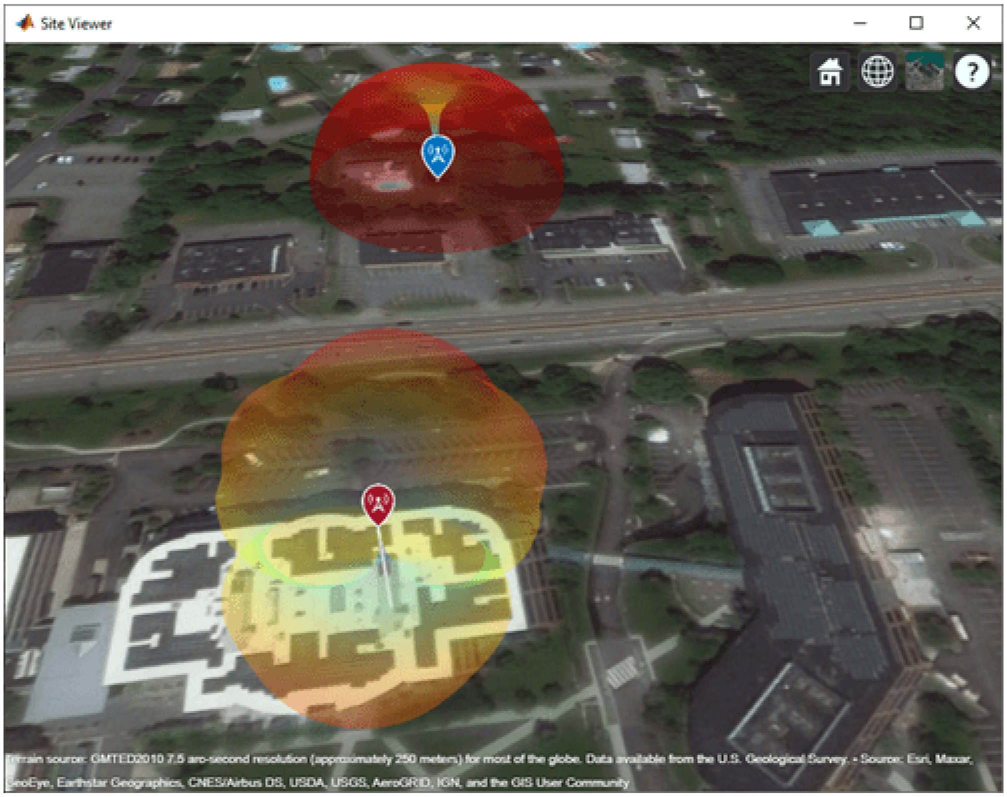
```
fq = 4.5e9;
tx = txsite('Name','MathWorks','Latitude',42.3001,'Longitude',-71.3503, ...
      'Antenna',yagiAntenna,'AntennaAngle',90,'AntennaHeight',30, ...
      'TransmitterFrequency',fq,'TransmitterPower',10);
rx = rxsite('Antenna',design(dipole,fq));
```

Position the receiver 200 meters from the transmitter.

```
[lat,lon] = location(tx,200,90);
rx.Latitude = lat;
rx.Longitude = lon;
```

Display both transmitter and receiver patterns. Zoom out so you can see both of the patterns.

```
pattern(tx,'Transparency',0.2)
pattern(rx,fq)
```


Site Viewer
errain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar,
GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Pattern for Cartesian Transmitter

Import and view an STL file. The file models a small conference room with one table and four chairs.

```
viewer = siteviewer("SceneModel","conferenceroom.stl");
```

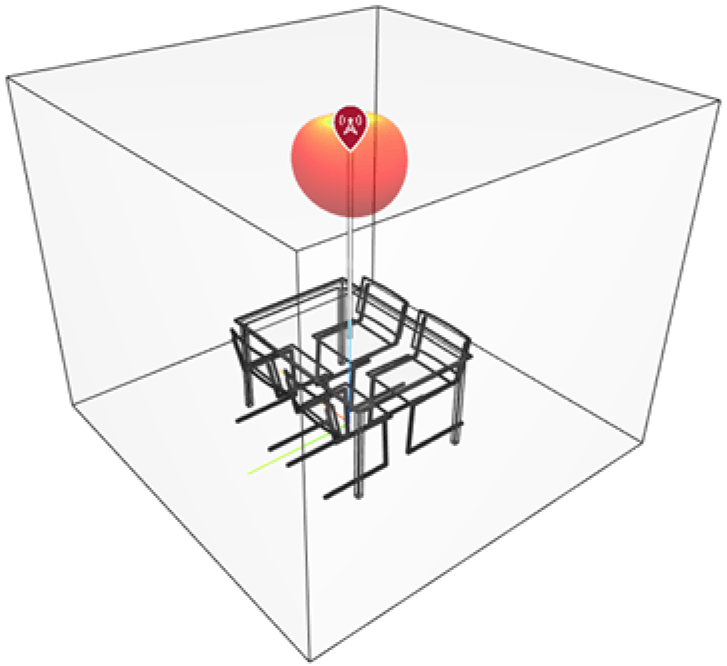
Design an inverted-F antenna mounted over a rectangular ground plane that resonates at 2.4 GHz. Create a transmitter site that uses the antenna. Specify the position using Cartesian coordinates in meters.

```
ant = design(invertedF,2.4e9);
ant.Tilt = 180;

tx = txsite("cartesian", ...
    "AntennaPosition",[0; 0; 2.1], ...
    "Antenna",ant);
```

Visualize the pattern of the site. Specify the size of the pattern plot as `0.4` meters.

```
pattern(tx,"Transparency",0.6,"Size",0.4)
```

Pan by left-clicking, zoom by right-clicking or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking and dragging.



Input Arguments

tx — Transmitter site

txsite object

Transmitter site, specified as a txsite object.

rx — Receiver site

rxsite object

Receiver site, specified as a rxsite object.

frequency — Frequency to calculate radiation pattern

positive scalar

Frequency to calculate radiation pattern, specified as a positive scalar.

Name-Value Pair Arguments

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

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

Example: 'Size',2

Size — Size of pattern plot

'auto' (default) | numerical scalar

Size of the pattern plot, specified as a numerical scalar in meters. This parameter represents the distance between the antenna position and the point on the plot with the highest gain.

The default value depends on the `CoordinateSystem` property of the `siteviewer` object. When `CoordinateSystem` is `'geographic'`, the default size is 50 meters. When `CoordinateSystem` is `'cartesian'`, the default size is approximately 1/6 of the scene model size.

Data Types: `double`

`Transparency` — Transparency of pattern plot

`0.4` (default) | real number in the range of [0,1]

Transparency of the pattern plot, specified as a real number in the range of [0,1], where `0` is completely transparent and `1` is completely opaque.

Data Types: `double`

`Colormap` — Colormap for coloring of pattern plot

`'jet(256)'` (default) | predefined colormap name | *M*-by-3 array of RGB triplets

Colormap for coloring of the pattern plot, specified as a predefined colormap name or an *M*-by-3 array of RGB (red, blue, green) triplets that define *M* individual colors.

Data Types: `double`

`Resolution` — Resolution of 3-D pattern

`'high'` (default) | `'low'` | `'medium'`

Resolution of 3-D map, specified as `'low'`, `'medium'`, or `'high'`. This property controls the visual quality and the time taken to plot the pattern where the value of `'low'` corresponds to the fastest and the least detailed pattern.

Data Types: `double`

`Map` — Map for visualization of surface data

`siteviewer` object

Map for visualization of surface data, specified as a `siteviewer` object.[6]

Data Types: `char` | `string`

Version History

Introduced in R2018b

See Also

`coverage`

[6] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

show

Show site in Site Viewer

Syntax

```
show(site)
show(site,Name,Value)
```

Description

`show(site)` displays the location of the specified transmitter or receiver site using a marker in the current Site Viewer.

`show(site,Name,Value)` displays `site` with additional options specified by one or more name-value arguments.

Examples

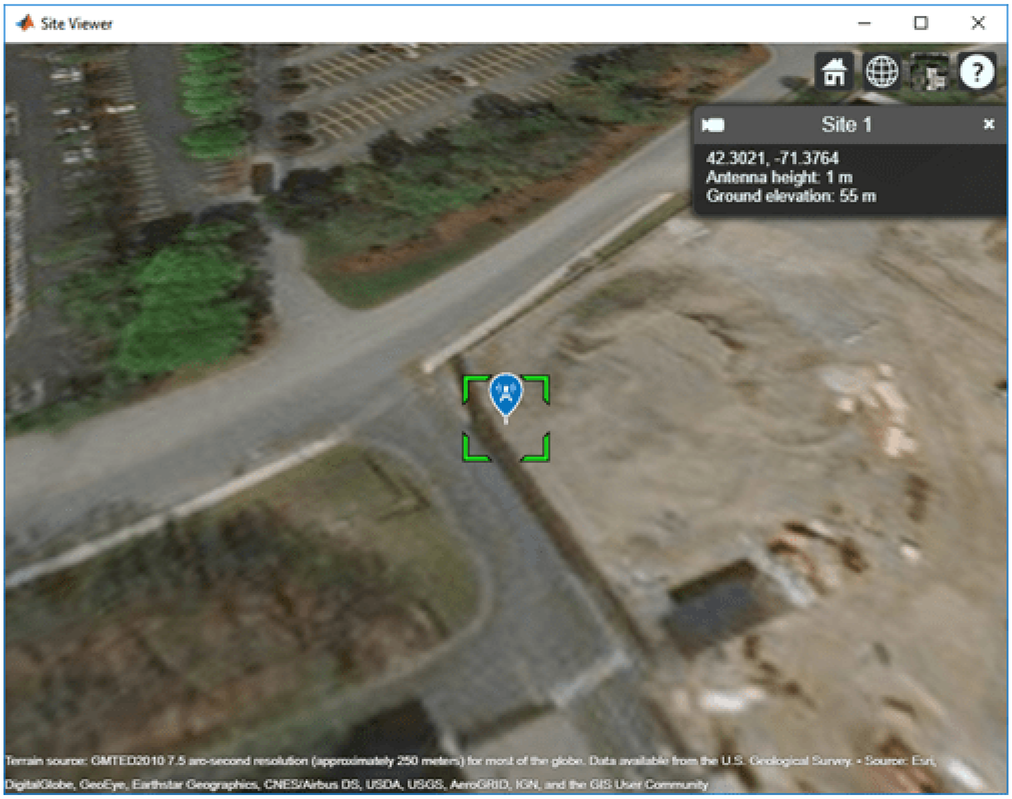
Default Receiver Site

Create and show the default receiver site.

```
rx = rxsite
```

```
rx = 
  rxsite with properties:

                   Name: 'Site 2'
               Latitude: 42.3021
              Longitude: -71.3764
                Antenna: 'isotropic'
           AntennaAngle: 0
          AntennaHeight: 1
             SystemLoss: 0
    ReceiverSensitivity: -100
```

```
show(rx)
```



Site 1

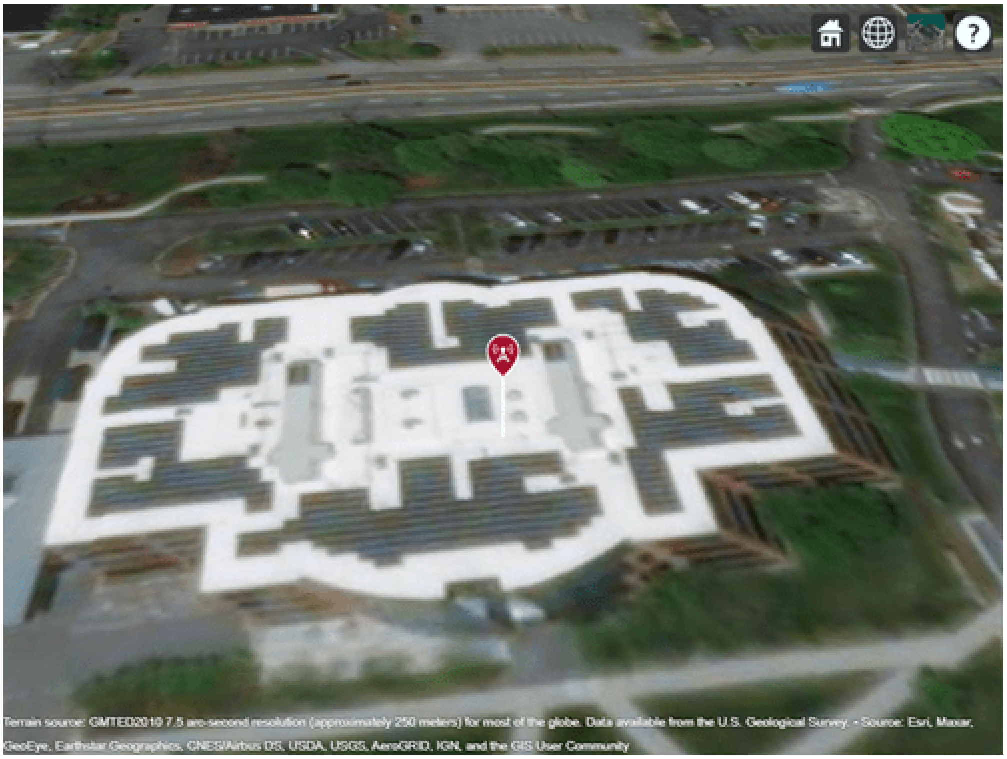
42.3021, -71.3764
Antenna height: 1 m
Ground elevation: 55 m

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Show and Hide Transmitter Site

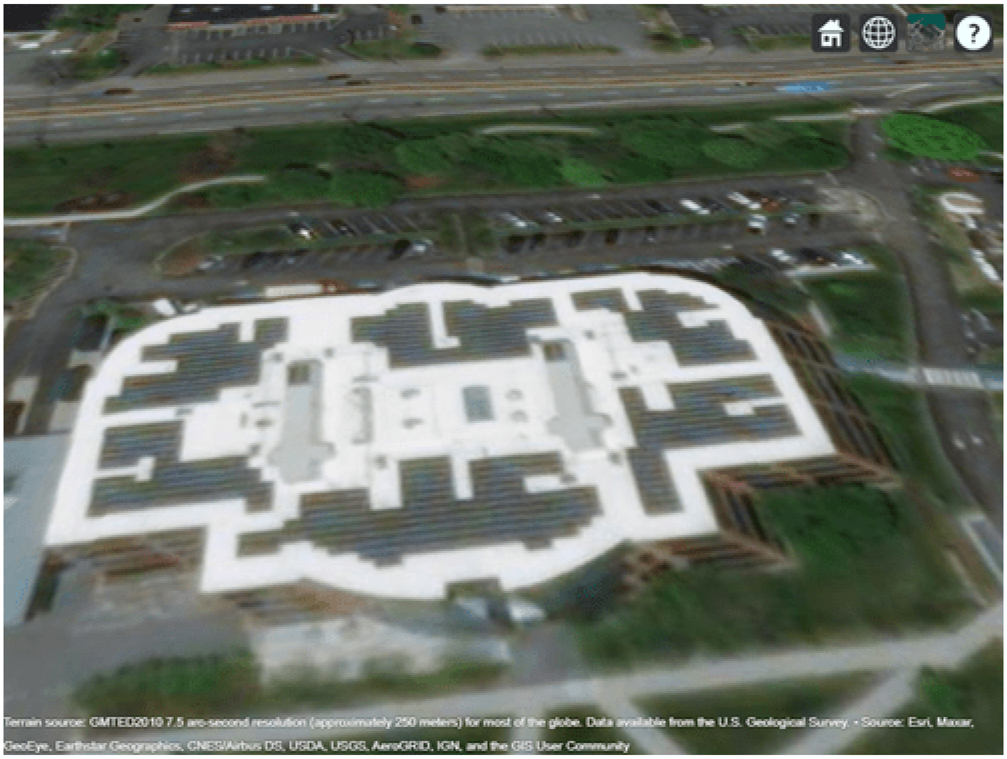
Create and show a transmitter site.

```
tx = txsite('Name','MathWorks Apple Hill',...
       'Latitude',42.3001, ...
       'Longitude',-71.3504);
show(tx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Hide the transmitter site.

```
hide(tx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Show and Hide Sites with Cartesian Coordinates

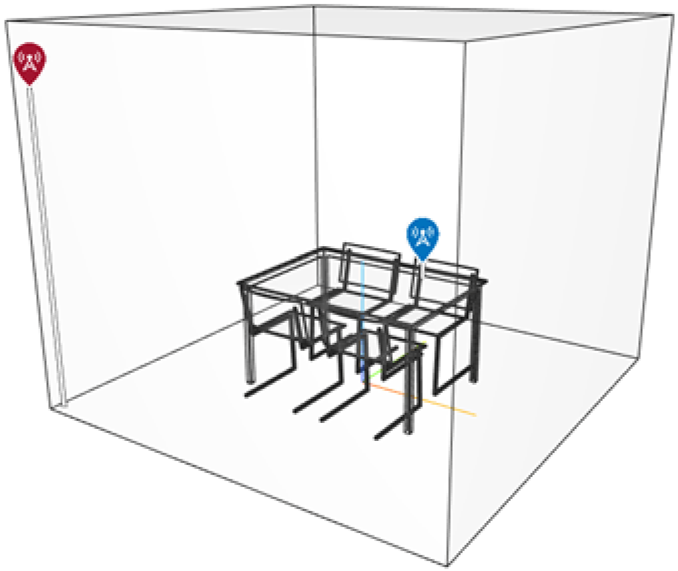
Import and view an STL file. The file models a small conference room with one table and four chairs.

```
viewer = siteviewer('SceneModel','conferenceroom.stl');
```

Create a transmitter site near the upper corner of the room and a receiver site above the table. Specify the position using Cartesian coordinates in meters. Then, visualize the sites.

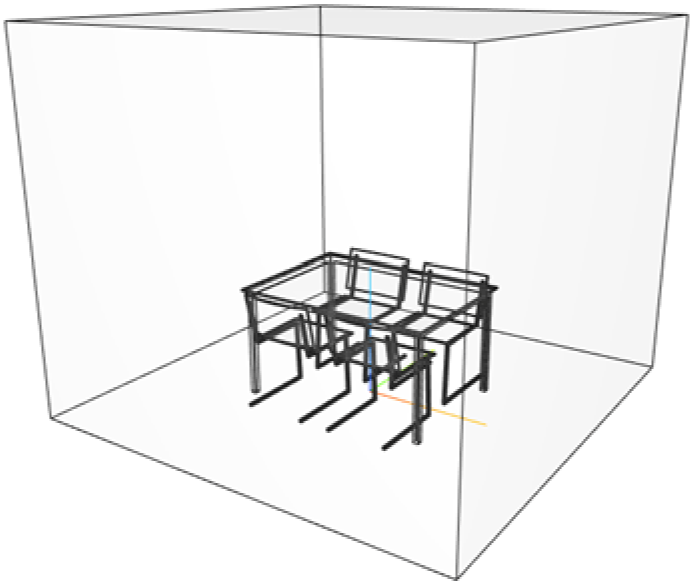
```
tx = txsite('cartesian', ...
    'AntennaPosition',[-1.46; -1.42; 2.1]);
rx = rxsite('cartesian', ...
    'AntennaPosition',[0.3; 0.3; 0.85]);

show(tx)
show(rx)
```

Pan by left-clicking, zoom by right-clicking or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking and dragging.



Hide the sites.

```
hide(tx)
hide(rx)
```



Input Arguments

site — Transmitter or receiver site
txsite or rxsite object | array of txsite or rxsite objects

Transmitter or receiver site, specified as a txsite or rxsite object or an array of txsite or rxsite objects.

Name-Value Pair Arguments

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

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

Example: 'ClusterMarkers',true

Icon — Image file
character vector

Image file, specified as a character vector.

Data Types: char

IconSize — Width and height of icon
36-by-36 (default) | 1-by-2 vector of positive numeric values

Width and height of the icon, specified as a 1-by-2 vector of positive numeric values in pixels.

`IconAlignment` — Vertical position of icon relative to site

'top' (default) | `'center'` | `'bottom'`

Vertical position of icon relative to site, specified as:

- `'bottom` - Aligns the icon below the site antenna position.
- `'center'` - Aligns the center of the icon to the site antenna position.
- `'top'` - Aligns the icon above the site antenna position.

`ClusterMarkers` — Combine nearby markers into groups or clusters

true | false

Combine nearby markers into groups or clusters, specified as true or false.

Data Types: `char`

`Map` — Map for visualization of surface data

`siteviewer` object

Map for visualization of surface data, specified as a `siteviewer` object.[7]

Data Types: `char` | `string`

`ShowAntennaHeight` — Option to show line from site to surface

`true` or `1` (default) | `false` or `0`

Option to show a white line from the site down to the nearest surface, specified as numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

Version History

Introduced in R2017b

See Also

`hide`

[7] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

sigstrength

Received signal strength

Syntax

```
ss = sigstrength(rx,tx)
ss = sigstrength(rx,tx,propmodel)
ss = sigstrength( ___ ,Name,Value)
```

Description

`ss = sigstrength(rx,tx)` returns the signal strength in power units (dBm) at the receiver site due to the transmitter site.

`ss = sigstrength(rx,tx,propmodel)` returns the signal strength at the receiver site using the specified propagation model. Specifying a propagation model is the same as specifying the `PropagationModel` name-value argument.

`ss = sigstrength( ___ ,Name,Value)` specifies options using name-value arguments, in addition to any combination of arguments from the previous syntaxes. For example, `"Type","efield"` returns the signal strength in electric field strength units (dBμV/m).

Examples

Received Power and Link Margin at Receiver

Create a transmitter site.

```
tx = txsite('Name','Fenway Park', ...
        'Latitude', 42.3467, ...
        'Longitude', -71.0972);
```

Create a receiver site with sensitivity defined (in dBm).

```
 rx = rxsite('Name','Bunker Hill Monument', ...
        'Latitude', 42.3763, ...
        'Longitude', -71.0611, ...
        'ReceiverSensitivity', -90);
```

Calculate the received power and link margin. Link margin is the difference between the receiver's sensitivity and the received power.

```
ss = sigstrength(rx,tx)
```

```
ss = -71.1414
```

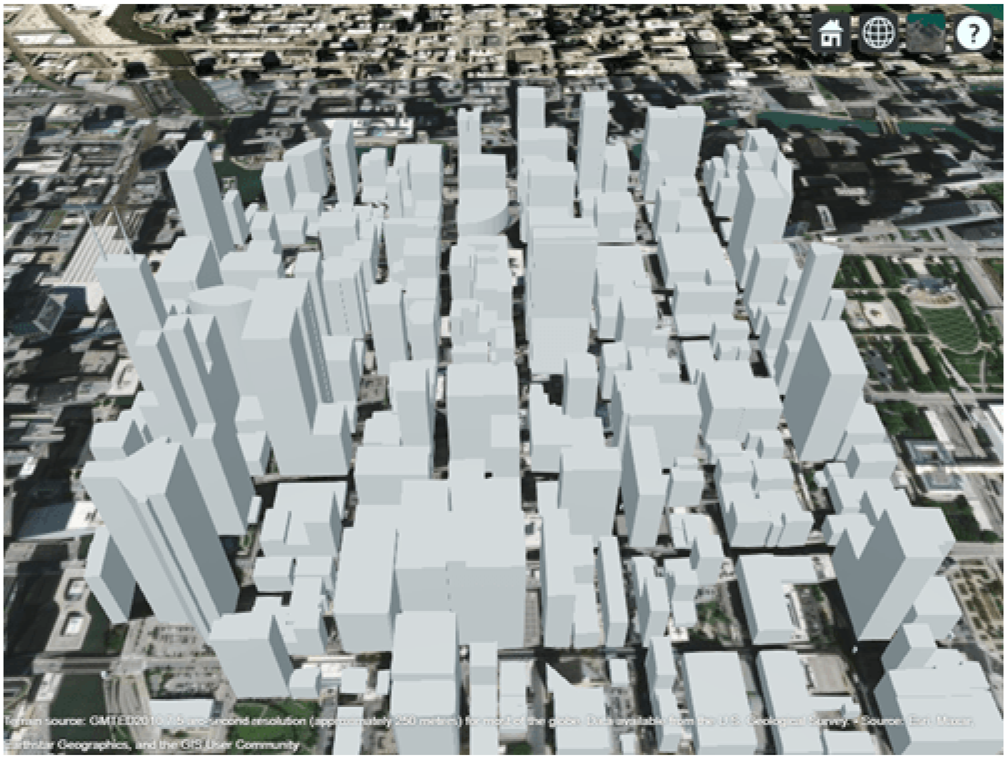
```
margin = abs(rx.ReceiverSensitivity - ss)
```

```
margin = 18.8586
```

Signal Strength Using Ray Tracing Propagation Model

Launch Site Viewer with buildings in Chicago. For more information about the osm file, see [1] on page 6-153.

```
viewer = siteviewer("Buildings","chicago.osm");
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Create a transmitter site on a building.

```
tx = txsite("Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
```

Create a receiver site near another building.

```
rx = rxsite("Latitude",41.881352, ...
    "Longitude",-87.629771, ...
    "AntennaHeight",30);
```

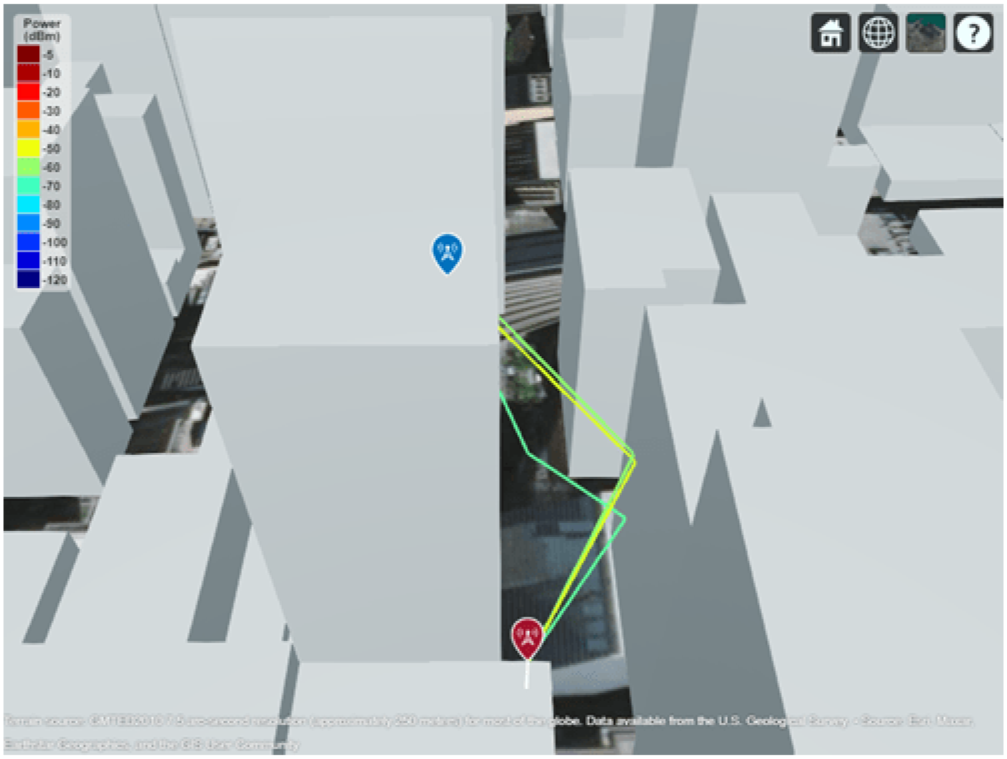
Compute the signal strength by using a ray tracing propagation model. By default, the ray tracing model uses the SBR method, and performs line-of-sight and two-reflection analysis.

```
pm = propagationModel("raytracing");
ssTwoReflections = sigstrength(rx,tx,pm)
```

```
ssTwoReflections = -54.3015
```

Plot the propagation paths for SBR with up to two reflections.

```
raytrace(tx,rx,pm)
```


Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Compute signal strength with analysis up to two reflections, where total received power is the cumulative power of all propagation paths

```
pm.MaxNumReflections = 5;
ssFiveReflections = sigstrength(rx,tx,pm)
```

```
ssFiveReflections = -53.3889
```

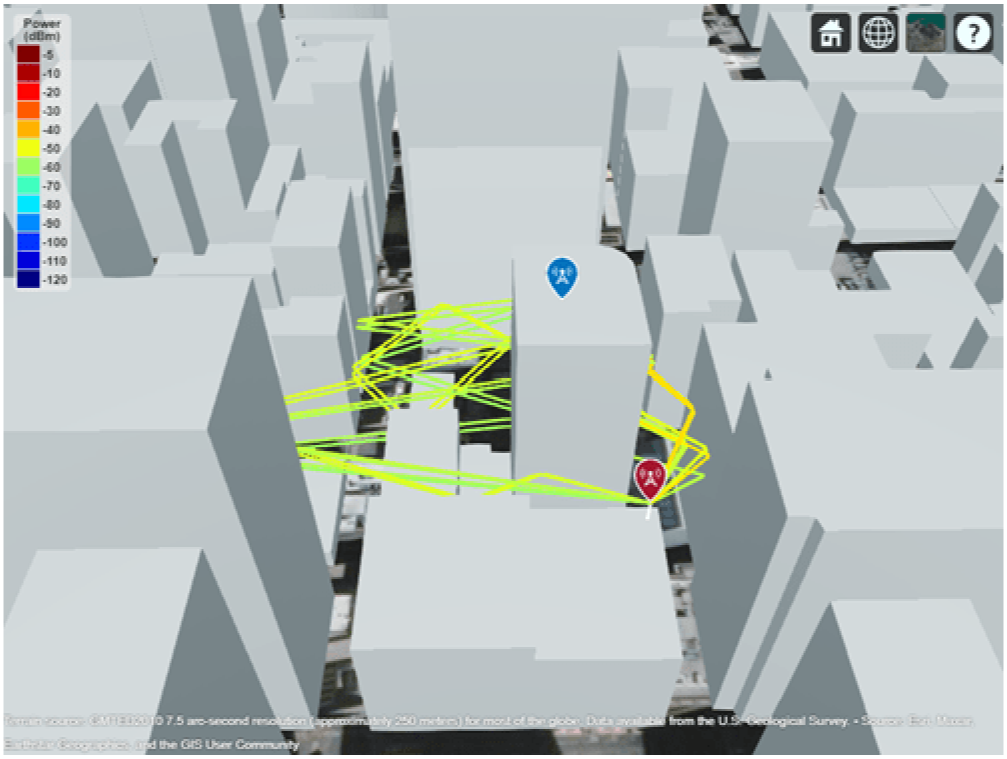
Observe the effect of material by replacing default concrete material with perfect reflector.

```
pm.BuildingsMaterial = "perfect-reflector";
ssPerfect = sigstrength(rx,tx,pm)
```

```
ssPerfect = -39.6703
```

Plot the propagation paths for SBR with up to five reflections.

```
raytrace(tx,rx,pm)
```


Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Input Arguments

`rx` — Receiver site

`rxsite` object | array of `rxsite` objects

Receiver site, specified as a `rxsite` object. You can use array inputs to specify multiple sites.

`tx` — Transmitter site

`txsite` object | array of `txsite` objects

Transmitter site, specified as a `txsite` object. You can use array inputs to specify multiple sites.

`propmodel` — Propagation model to use for path loss calculations

`"longley-rice"` (default) | `"freespace"` | `"close-in"` | `"rain"` | `"gas"` | `"fog"` | `"raytracing"` | propagation model created with `propagationModel`

Propagation model to use for the path loss calculations, specified as one of these options:

- `"freespace"` — Free space propagation model
- `"rain"` — Rain propagation model

- "gas" — Gas propagation model
- "fog" — Fog propagation model
- "close-in" — Close-in propagation model
- "longley-rice" — Longley-Rice propagation model
- "tirem" — TIREM propagation model
- "raytracing" — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the propagationModel function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
"geographic"	• "longley-rice" when you use a terrain. • "freespace" when you do not use a terrain.
"cartesian"	• "freespace" when Map is set to none. • "raytracing" when Map is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including "longley-rice" and "tirem", are only supported for sites with a CoordinateSystem value of "geographic".

You can also specify the propagation model by using the PropagationModel name-value pair argument.

Name-Value Pair Arguments

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

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

Example: "Type","power"

Type — Type of signal strength to compute

"power" (default) | "efield"

Type of signal strength to compute, specified as one of these options:

- "power" — The signal strength is in power units (dBm) of the signal at the mobile receiver input.
- "efield"— The signal strength is in electric field strength units (dBμV/m) of signal wave incident on the antenna.

Data Types: char | string

PropagationModel — Propagation model to use for path loss calculations

"freespace" | "close-in" | "rain" | "gas" | "fog" | "longley-rice" | "raytracing" | propagation model created with propagationModel

Propagation model to use for the path loss calculations, specified as one of these options:

- `"freespace"` — Free space propagation model
- `"rain"` — Rain propagation model
- `"gas"` — Gas propagation model
- `"fog"` — Fog propagation model
- `"close-in"` — Close-in propagation model
- `"longley-rice"` — Longley-Rice propagation model
- `"tirem"` — TIREM propagation model
- `"raytracing"` — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the `propagationModel` function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
`"geographic"`	• `"longley-rice"` when you use a terrain. • `"freespace"` when you do not use a terrain.
`"cartesian"`	• `"freespace"` when `Map` is set to none. • `"raytracing"` when `Map` is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including `"longley-rice"` and `"tirem"`, are only supported for sites with a `CoordinateSystem` value of `"geographic"`.

Data Types: `char` | `string`

`Map` — Map for visualization or surface data

`siteviewer` object | `triangulation` object | string scalar | character vector

Map for visualization or surface data, specified as a `siteviewer` object, a `triangulation` object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
`"geographic"`	• A `siteviewer` object[a]. • A terrain name, if the function is called with an output argument. Valid terrain names are `"none"`, `"gmted2010"`, or the name of the custom terrain data added using `addCustomTerrain`.	• The current `siteviewer` object or a new `siteviewer` object if none are open. • `"gmted2010"`, if the function is called with an output.

Coordinate System	Valid map values	Default map value
"cartesian"	• "none". • A siteviewer object. • The name of an STL file. • A triangulation object.	• "none".

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: char | string

Output Arguments

ss — Signal strength

M-by-*N* array

Signal strength, returned as *M*-by-*N* array, where *M* is the number of transmitter sites and *N* is the number of receiver sites.

The units of ss depend on the value of the Type name-value argument.

- When you specify Type as "power", then ss is in power units (dBm) of the signal at the mobile receiver input.
- When you specify Type as "efield", then ss is in electric field strength units (dBμV/m) of signal wave incident on the antenna.

Version History

Introduced in R2017b

Ray tracing functions consider multipath interference

Behavior changed in R2022b

When calculating received power using ray tracing models, the sigstrength function now considers multipath interference by using a phasor sum. In previous releases, the function used a power sum. As a result, the calculations in R2022b are more accurate than in previous releases.

"raytracing" propagation models use SBR method

Behavior changed in R2021b

Starting in R2021b, when you use the sigstrength function and specify the propmodel argument or PropagationModel name-value argument as "raytracing", the function uses the shooting and bouncing rays (SBR) method and calculates up to two reflections. In previous releases, the sigstrength function uses the image method and calculates up to one reflection.

To calculate received signal strength using the image method instead, create a propagation model by using the propagationModel function. Then, use the sigstrength function with the propagation model as input. This example shows how to update your code.

```
pm = propagationModel("raytracing","Method","image");
ss = sigstrength(rx,tx,pm)
```

For information about the SBR and image methods, see “Choose a Propagation Model”.

Starting in R2021b, all RF Propagation functions use the SBR method by default and calculate up to two reflections. For more information, see “Default modeling method is shooting and bouncing rays method” on page 6-129.

See Also

`link` | `sinr` | `propagationModel`

sinr

Display or compute signal-to-interference-plus-noise (SINR) ratio

Syntax

```
sinr(txs)
sinr(txs,propmodel)
sinr( ___ ,Name,Value)
pd = sinr(txs, ___ )
r = sinr(rxs,txs, ___ )
```

Description

`sinr(txs)` displays the signal-to-interference-plus-noise ratio (SINR) for transmitter sites `txs` in the current Site Viewer. The map contours are generated using SINR values computed for receiver site locations on the map. For each location, the signal source is the transmitter site in TXS with the greatest signal strength. The remaining transmitter sites in `txs` with the same transmitter frequency act as sources of interference. If `txs` is scalar or there are no sources of interference the resultant map displays signal-to-noise ratio (SNR).

This function only supports plotting for antenna sites with a `CoordinateSystem` property value of `"geographic"`.

`sinr(txs,propmodel)` displays the SINR map with the propagation model set to the value in `propmodel`.

`sinr( ___ ,Name,Value)` sets properties using one or more name-value pairs, in addition to the input arguments in previous syntaxes. For example, `sinr(txs,"MaxRange",8000)` sets the range from the site location at 8000 meters to include in the SINR map region.

`pd = sinr(txs, ___ )` returns computed SINR data in the propagation data object, `pd`. No plot is displayed and any graphical only name-value pairs are ignored.

`r = sinr(rxs,txs, ___ )` returns the `sinr` in dB computed at the receiver sites due to the transmitter sites.

Examples

SINR Map for Multiple Transmitters

Define names and location of sites in Boston.

```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

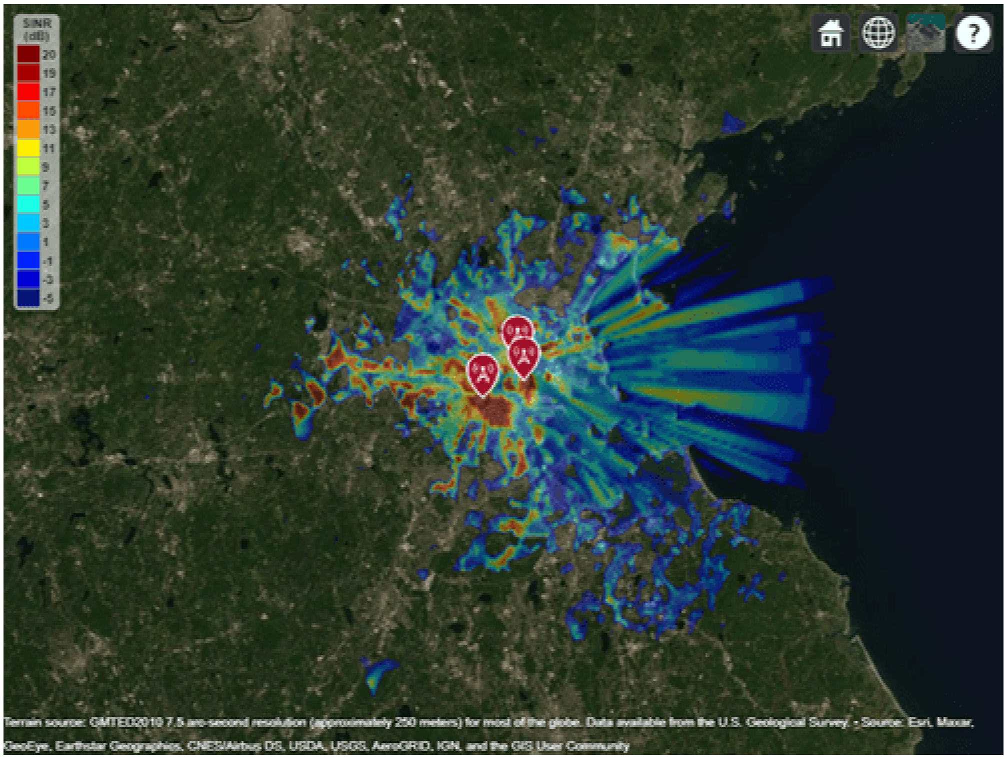
Create a transmitter site array.

```
txs = txsite("Name", names,...
       "Latitude",lats,...
```

```
        "Longitude",lons, ...
        "TransmitterFrequency",2.5e9);
```

Display the SINR map, where signal source for each location is selected as the transmitter site with the strongest signal.

```
sinr(txs)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

txs — Transmitter sites

txsite object | array of txsite objects

Transmitter site, specified as a `txsite` object. Use array inputs to specify multiple sites.

This function only supports plotting antenna sites when `CoordinateSystem` property is set to `"geographic"`.

rxs — Receiver sites

rxsite object | array of rxsite objects

Receiver site, specified as a `rxsite` object. Use array inputs to specify multiple sites.

This function only supports plotting antenna sites when `CoordinateSystem` property is set to `"geographic"`.

propmodel — Propagation model to use for path loss calculations

"longley-rice" (default) | "freespace" | "close-in" | "rain" | "gas" | "fog" | "raytracing" | propagation model created with propagationModel

Propagation model to use for the path loss calculations, specified as one of these options:

- `"freespace"` — Free space propagation model
- `"rain"` — Rain propagation model
- `"gas"` — Gas propagation model
- `"fog"` — Fog propagation model
- `"close-in"` — Close-in propagation model
- `"longley-rice"` — Longley-Rice propagation model
- `"tirem"` — TIREM propagation model
- `"raytracing"` — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the `propagationModel` function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
`"geographic"`	• `"longley-rice"` when you use a terrain. • `"freespace"` when you do not use a terrain.
`"cartesian"`	• `"freespace"` when `Map` is set to none. • `"raytracing"` when `Map` is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including `"longley-rice"` and `"tirem"`, are only supported for sites with a `CoordinateSystem` value of `"geographic"`.

You can also specify the propagation model by using the `PropagationModel` name-value pair argument.

Name-Value Pair Arguments

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

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

Example: `"MaxRange",8000`

General

`SignalSource` — Signal source of interest

`"strongest"` (default) | transmitter site object

Signal source of interest, specified as the comma-separated pair consisting of `SignalSource` and `"strongest"` or as a transmitter site object. When the signal source of interest is `"strongest"`, the transmitter with the greatest signal strength is chosen as the signal source of interest for that location. When computing `sinr`, `SignalSource` can be a `txsite` array with equal number of elements `rxs` where each transmitter site element defines the signal source for the corresponding receiver site.

`PropagationModel` — Propagation model to use for path loss calculations

`"freespace"` | `"close-in"` | `"rain"` | `"gas"` | `"fog"` | `"longley-rice"` | `"raytracing"` | propagation model created with `propagationModel`

Propagation model to use for the path loss calculations, specified as one of these options:

- `"freespace"` — Free space propagation model
- `"rain"` — Rain propagation model
- `"gas"` — Gas propagation model
- `"fog"` — Fog propagation model
- `"close-in"` — Close-in propagation model
- `"longley-rice"` — Longley-Rice propagation model
- `"tirem"` — TIREM propagation model
- `"raytracing"` — Ray tracing propagation model that uses the shooting and bouncing rays (SBR) method. When you specify a ray tracing model as input, the function incorporates multipath interference by using a phasor sum.
- A propagation model created with the `propagationModel` function

The default value depends on the coordinate system used by the input sites.

Coordinate System	Default propagation model value
`"geographic"`	• `"longley-rice"` when you use a terrain. • `"freespace"` when you do not use a terrain.
`"cartesian"`	• `"freespace"` when `Map` is set to none. • `"raytracing"` when `Map` is set to the name of an STL file or a triangulation object. The default ray tracing model uses the shooting and bouncing rays (SBR) method.

Terrain propagation models, including `"longley-rice"` and `"tirem"`, are only supported for sites with a `CoordinateSystem` value of `"geographic"`.

Data Types: `char` | `string`

`ReceiverNoisePower` — Total noise power at receiver

`-107` (default) | scalar

Total noise power at receiver, specified as a scalar in dBm. The default value assumes that the receiver bandwidth is 1 MHz and receiver noise figure is 7 dB.

$$N = -174 + 10*\log(B) + F$$

where,

- N = Receiver noise in dBm
- B = Receiver bandwidth in Hz

- *F* = Noise figure in dB

ReceiverGain — Receiver gain

2.1 (default) | scalar

Mobile receiver gain, specified as a scalar in dB. The receiver gain values include the antenna gain and the system loss. If you call the function using an output argument, the default value is computed using rxs.

ReceiverAntennaHeight — Receiver antenna height

1 (default) | scalar

Receiver antenna height above the ground, specified as a scalar in meters. If you call the function using an output argument, the default value is computed using rxs.

Map — Map for visualization or surface data

siteviewer object | triangulation object | string scalar | character vector

Map for visualization or surface data, specified as a siteviewer object, a triangulation object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
"geographic"	• A siteviewer object[a]. • A terrain name, if the function is called with an output argument. Valid terrain names are "none", "gmted2010", or the name of the custom terrain data added using addCustomTerrain.	• The current siteviewer object or a new siteviewer object if none are open. • "gmted2010", if the function is called with an output.
"cartesian"	• "none". • A siteviewer object. • The name of an STL file. • A triangulation object.	• "none".

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: char | string

For Plotting SINR

Values — Values of SINR for display

[-5:20] (default) | numeric vector

Values of SINR for display, specified as a numeric vector. Each value is displayed as a different colored, filled on the contour map. The contour colors are derived using Colormap and ColorLimits.

MaxRange — Maximum range of coverage map from each transmitter site

numeric scalar

Maximum range of coverage map from each transmitter site, specified as a positive numeric scalar in meters representing great circle distance. MaxRange defines the region of interest on the map to plot. The default value is automatically computed based on the type of propagation model.

Type of Propagation Model	MaxRange
Atmospheric or empirical	30 km
Terrain	30 km or distance to the furthest building.
Ray tracing	500 m

For more information about the types of propagation models, see "Choose a Propagation Model".

Data Types: double

Resolution — Resolution of receiver site locations used to compute SINR values

"auto" (default) | numeric scalar

Resolution of receiver site locations used to compute SINR values, specified as "auto" or a numeric scalar in meters. The resolution defines the maximum distance between the locations. If the resolution is "auto", sinr computes a value scaled to MaxRange. Decreasing the resolution increases the quality of the SINR map and the time required to create it.

Colormap — Colormap for coloring filled contours

"jet" (default) | *M*-by-3 array of RGB triplets

Colormap for coloring filled contours, specified as an *M*-by-3 array of RGB triplets, where *M* is the number of individual colors.

ColorLimits — Color limits for color maps

[-5 20] (default) | two-element vector

Color limits for color maps, specified as a two-element vector of the form [min max]. The color limits indicate the SINR values that map to the first and last colors in the colormap.

ShowLegend — Show signal strength color legend on map

"true" (default) | "false"

Show signal strength color legend on map, specified as "true" or "false".

Transparency — Transparency of SINR map

0.4 (default) | numeric scalar

Transparency of SINR map, specified as a numeric scalar in the range [0, 1]. If the value is zero, the map is completely transparent. If the value is one, the map is completely opaque.

Output Arguments

r — Signal to interference plus noise ratio at the receiver

numeric vector (default)

Signal to interference plus noise ratio at the receiver due to the transmitter sites, returned as a numeric vector. The vector length is equal to the number of receiver sites.

Data Types: double

pd — SINR data
propagationData object

SINR data, returned as a `propagationData` object consisting of *Latitude* and *Longitude*, and a signal strength variable corresponding to the plot type. Name of the `propagationData` is `"SINR Data"`.

Version History

Introduced in R2018a

Ray tracing functions consider multipath interference

Behavior changed in R2022b

When calculating received power using ray tracing models, the `sinr` function now incorporates multipath interference by using a phasor sum. In previous releases, the function used a power sum. As a result, the calculations in R2022b are more accurate than in previous releases.

"raytracing" propagation models use SBR method

Behavior changed in R2021b

Starting in R2021b, when you use the `sinr` function and specify the `propmodel` argument or `PropagationModel` name-value argument as `"raytracing"`, the function uses the shooting and bouncing rays (SBR) method and calculates up to two reflections. In previous releases, the `sinr` function uses the image method and calculates up to one reflection.

To display or compute the SINR using the image method instead, create a propagation model by using the `propagationModel` function. Then, use the `sinr` function with the propagation model as input. This example shows how to update your code.

```
pm = propagationModel("raytracing","Method","image");
sinr(txs,pm)
```

For information about the SBR and image methods, see “Choose a Propagation Model”.

Starting in R2021b, all RF Propagation functions use the SBR method by default and calculate up to two reflections. For more information, see “Default modeling method is shooting and bouncing rays method” on page 6-129.

See Also

`coverage` | `propagationModel`

tirempl

Path loss using Terrain Integrated Rough Earth Model (TIREM)

Syntax

```
pl = tirempl(r,z,f)
pl = tirempl(r,z,f,Name,Value)
[pl,output] = tirempl( ___ )
```

Description

`pl = tirempl(r,z,f)` returns the path loss in dB for a signal with frequency `f` when it is propagated over terrain. You can specify terrain using numeric vectors for distance `r` and elevation `z` along the great circle path between the transmitter and the receiver. The Terrain Integrated Rough Earth Model (TIREM) model combines physics with empirical data to provide path loss estimates. The TIREM model is valid from 1 MHz to 1000 GHz.

Note `tirempl` requires access to the external TIREM library. Use `tiremSetup` to set up access.

`pl = tirempl(r,z,f,Name,Value)` returns the path loss in dB with additional options specified by name-value pairs.

`[pl,output] = tirempl( ___ )` returns the path loss, `pl`, and the output structure containing the information on the TIREM analysis.

Examples

Path Loss Over Flat Terrain

Calculate the path loss over flat terrain. Define the terrain profile for distances up to 10 km with step size of 100 m.

```
freq = 28e9;
r = 0:100:10000;
z = zeros(1,numel(r));
    Lterrain1 = tirempl(r,z,freq,...
       'TransmitterAntennaHeight',5, ...
       'ReceiverAntennaHeight',5)
```

```
Lterrain1 =

  142.6089
```

Input Arguments

r — Distances
numeric vector

Distances along the great circle path between the transmitter and the receiver, specified as a numeric vector with each value in meters. The number of distance values must be equal to the number of elevation values.

Data Types: `double`

`z` — Elevation

numeric vector

Elevation values corresponding to the distance values along the great circle path between the transmitter and the receiver, specified as a numeric vector with each value in meters. The number of elevation values must be equal to the number of distance values.

Data Types: `double`

`f` — Frequency of propagated signal

scalar | numeric vector

Frequency of the propagated signal, specified as a scalar or numeric vector with each element unit in Hz.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'TransmitterAntennaHeight',50`

`TransmitterAntennaHeight` — Transmitter antenna height above ground

`10` (default) | numeric scalar

Transmitter antenna height above the ground, specified as a numeric scalar in the range of 0 to 30000. The height is measured from ground elevation to the center of the antenna.

Data Types: `double`

`ReceiverAntennaHeight` — Receiver antenna height above ground

`1` (default) | numeric scalar

Receiver antenna height above the ground, specified as a numeric scalar in the range of 0 to 30000. The height is measured from ground elevation to the center of the antenna.

Data Types: `double`

`AntennaPolarization` — Polarization of transmitter and receiver antennas

`'horizontal'` (default) | `'vertical'`

Polarization of the transmitter and the receiver antennas, specified as `'horizontal'` or `'vertical'`.

Data Types: `string` | `char`

`GroundConductivity` — Conductivity of ground

`0.005` (default) | numeric scalar

Conductivity of the ground, specified as a numeric scalar in the range of 0.00005 to 100 in Siemens per meter. This value is used to calculate the path loss due to ground reflection. The default value corresponds to the average ground conductivity.

Data Types: `double`

`GroundPermittivity` — Relative permittivity of ground

`15` (default) | numeric scalar

Relative permittivity of the ground, specified as a numeric scalar in the range of 1 to100. Relative permittivity is the ratio of absolute material permittivity to the permittivity of vacuum. This value is used to calculate the path loss due to ground reflection. The default value corresponds to the average ground permittivity.

Data Types: `double`

`AtmosphericRefractivity` — Atmospheric refractivity near ground

`301` (default) | numeric scalar

Atmospheric refractivity near the ground, specified as a numeric scalar in N-units in the range of 250 to 400. This value is used to calculate the path loss due to atmospheric refraction and tropospheric scatter. The default value corresponds to average atmospheric conditions.

Data Types: `double`

`Humidity` — Absolute air humidity near ground

`9` (default) | numeric scalar

Absolute air humidity near the ground, specified as a numeric scalar in g/m^^{3} in the range of 50 to 110. This value is used to calculate path loss due to atmospheric absorption. The default value corresponds to the absolute humidity of air at 15 degrees Celsius and 70 percent relative humidity.

Data Types: `double`

Output Arguments

`pl` — Path loss

scalar | 1-by-*N* vector

Path loss, returned as a scalar or 1-by-*N* vector with each element unit in decibels. *N* is the number of frequencies defined in the input `f`.

Path loss is calculated from free-space loss, terrain diffraction, ground reflection, refraction through the atmosphere, tropospheric scatter, and atmospheric absorption.

`output` — Information of TIREM analysis

structure

Information of TIREM analysis, returned as a structure. Each field of the structure represents an output from TIREM analysis.

Version History

Introduced in R2019a

See Also

`propagationModel` | `tiremSetup`

Topics

"Access TIREM Software"

tiremSetup

Set up access to Terrain Integrated Rough Earth Model (TIREM)

Syntax

```
tiremSetup
tiremSetup(libfolder)
libfolder = tiremSetup
```

Description

`tiremSetup` opens a dialog to select the Terrain Integrated Rough Earth Model (TIREM) library folder. The TIREM library folder must contain the `tirem3` shared library, where the full library name is platform dependent. For more information, see "Platform dependent library names" on page 6-169.

`tiremSetup(libfolder)` sets the TIREM library folder to `libfolder`.

`libfolder = tiremSetup` returns the current TIREM library folder.

Input Arguments

`libfolder` — Name of TIREM library folder

character vector

Name of the TIREM library folder, specified as a character vector.

Data Types: `char` | `string`

Output Arguments

`libfolder` — Current TIREM library folder

character vector | string scalar

Current TIREM library folder, returned as a character vector or a string scalar. If TIREM access has not been setup, `libfolder` is empty.

More About

Platform dependent library names

Platform	Shared library name
`Windows`	`libtirem3.dll` or `tirem3.dll`
`Linux`	`libtirem3.so`
`Mac`	`libtirem3.dylib`

Version History

Introduced in R2019a

See Also

`propagationModel` | `tirempl`

Topics

"Access TIREM Software"

raytrace

Display or compute RF propagation rays

Syntax

```
raytrace(tx,rx)
raytrace(tx,rx,propmodel)
raytrace( ___ ,Name,Value)
rays = raytrace( ___ )
```

Description

The `raytrace` function plots or computes propagation paths by using ray tracing with surface geometry defined by the `'Map'` property. Each plotted propagation path is color-coded according to the received power (dBm) or path loss (dB) along the path. The ray tracing analysis includes surface reflections but does not include effects from diffraction, refraction, or scattering. Operational frequency for this function is from 100 MHz to 100 GHz. For more information, see “Choose a Propagation Model”.

`raytrace(tx,rx)` displays the propagation paths from the transmitter site (`tx`) to the receiver site (`rx`) in the current Site Viewer by using the shooting and bouncing rays (SBR) method with up to two reflections.

`raytrace(tx,rx,propmodel)` displays the propagation paths from the transmitter site (`tx`) to the receiver site (`rx`) based on the specified propagation model. To input building and terrain materials to calculate path loss, create a `'raytracing'` propagation model using the `propagationModel` function and set the properties to specify building materials.

`raytrace( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in previous syntaxes.

`rays = raytrace( ___ )` returns the propagation paths in `rays`.

Examples

Obstructed and Reflected Paths Using Ray Tracing

Show reflected propagation paths in Chicago using the ray tracing analysis with the SBR method

Launch Site Viewer with buildings in Chicago. For more information about the osm file, see [1] on page 6-173.

```
viewer = siteviewer("Buildings","chicago.osm");
```

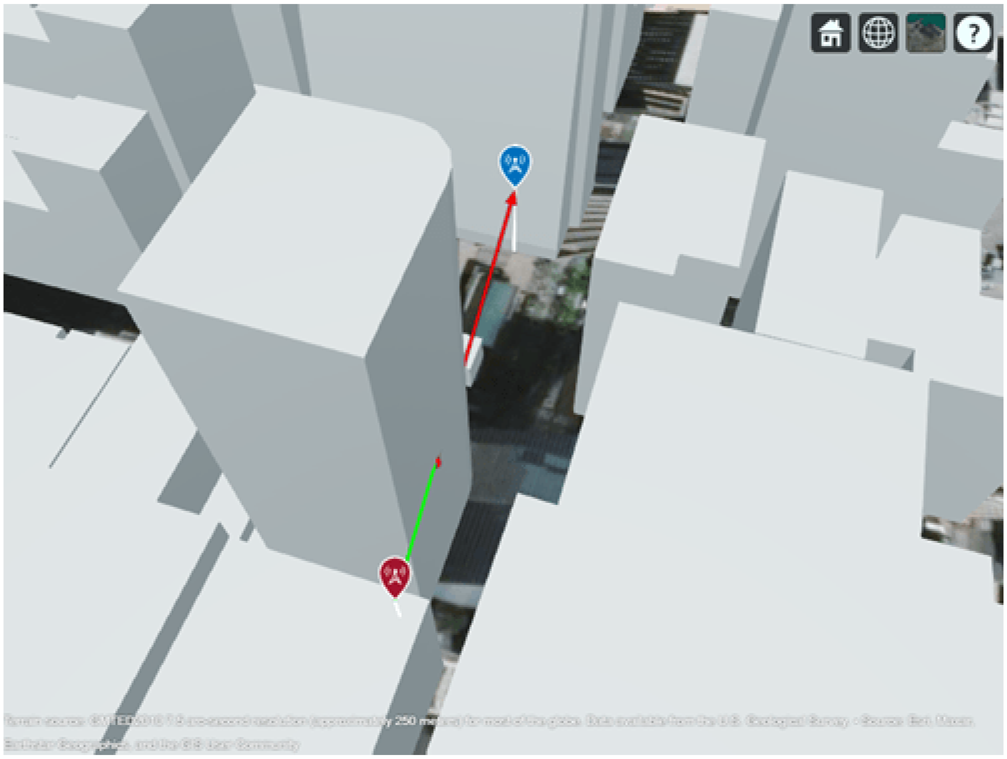
Create a transmitter site on a building and a receiver site near another building.

```
tx = txsite("Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
```

```
show(tx)
rx = rxsite("Latitude",41.8813452, ...
    "Longitude",-87.629771, ...
    "AntennaHeight",30);
show(rx)
```

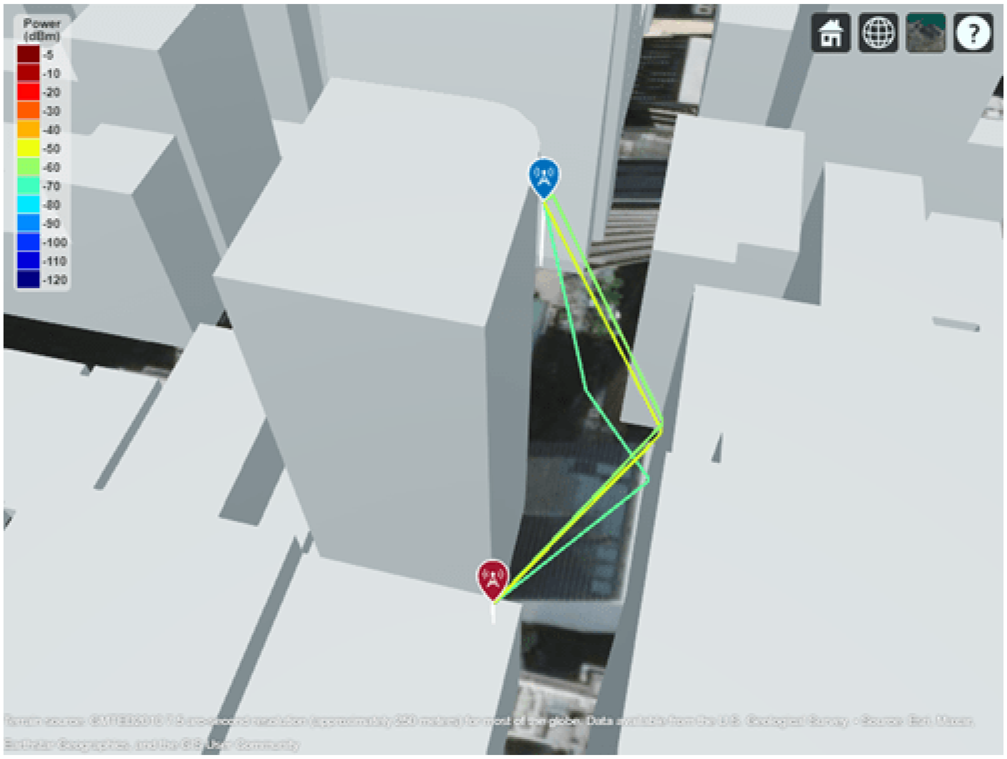
Show obstruction to line-of-sight.

```
los(tx,rx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Show reflected propagation path using ray tracing with up to two reflections.

```
raytrace(tx,rx)
```


Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

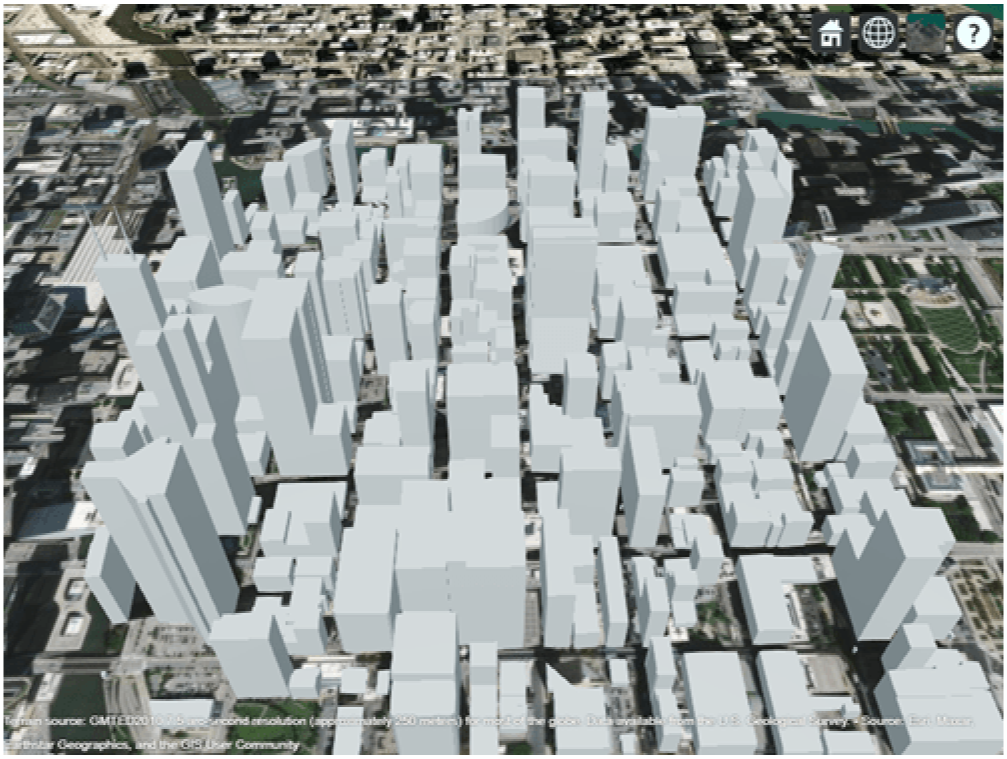
Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Signal Strength Using Ray Tracing Propagation Model

Launch Site Viewer with buildings in Chicago. For more information about the osm file, see [1] on page 6-176.

```
viewer = siteviewer("Buildings","chicago.osm");
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Create a transmitter site on a building.

```
tx = txsite("Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
```

Create a receiver site near another building.

```
rx = rxsite("Latitude",41.881352, ...
    "Longitude",-87.629771, ...
    "AntennaHeight",30);
```

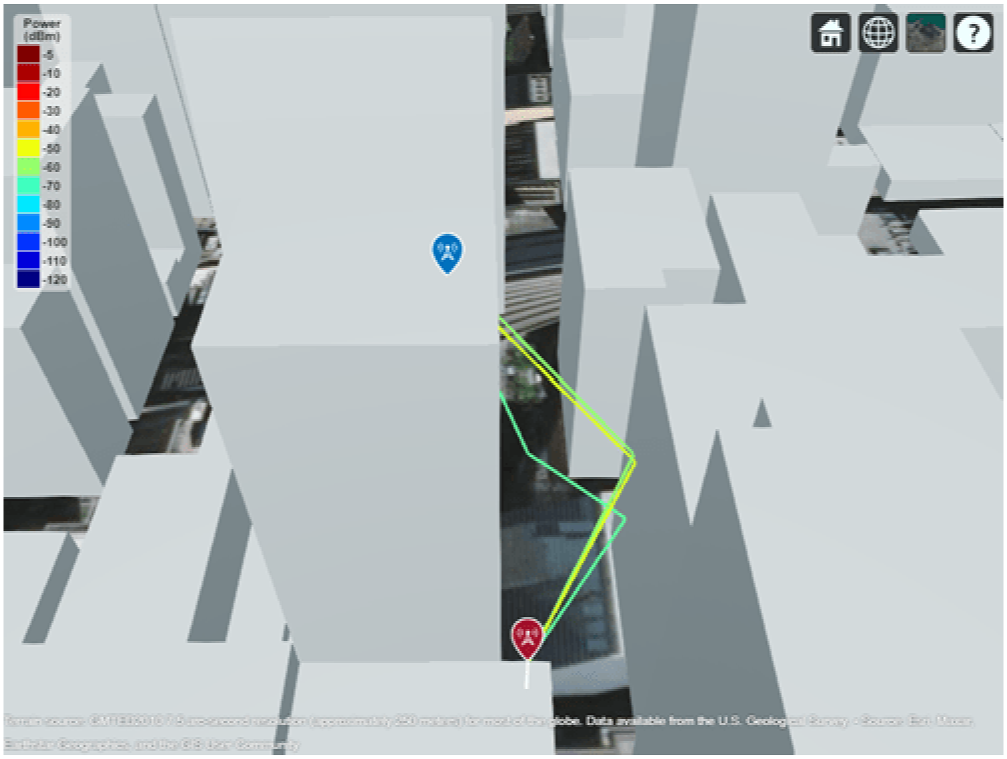
Compute the signal strength by using a ray tracing propagation model. By default, the ray tracing model uses the SBR method, and performs line-of-sight and two-reflection analysis.

```
pm = propagationModel("raytracing");
ssTwoReflections = sigstrength(rx,tx,pm)
```

```
ssTwoReflections = -54.3015
```

Plot the propagation paths for SBR with up to two reflections.

```
raytrace(tx,rx,pm)
```


Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Compute signal strength with analysis up to two reflections, where total received power is the cumulative power of all propagation paths

```
pm.MaxNumReflections = 5;
ssFiveReflections = sigstrength(rx,tx,pm)
```

```
ssFiveReflections = -53.3889
```

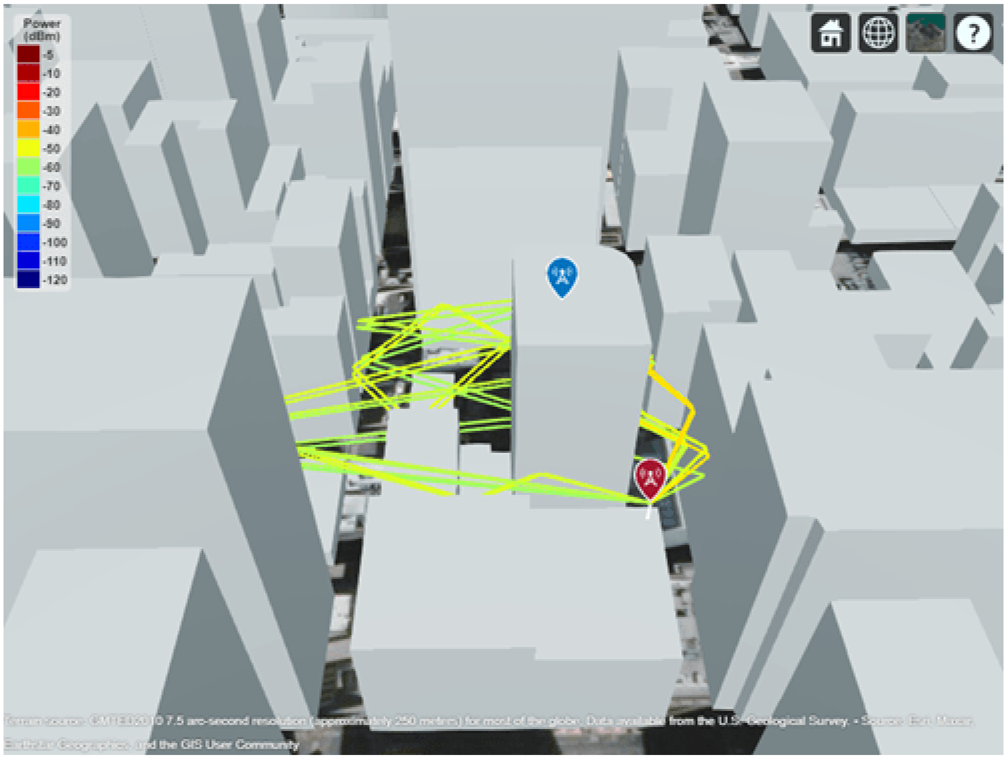
Observe the effect of material by replacing default concrete material with perfect reflector.

```
pm.BuildingsMaterial = "perfect-reflector";
ssPerfect = sigstrength(rx,tx,pm)
```

```
ssPerfect = -39.6703
```

Plot the propagation paths for SBR with up to five reflections.

```
raytrace(tx,rx,pm)
```


Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
?
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Path Loss Due to Material Reflection and Atmosphere

Calculate path loss due to material reflection and atmosphere in Hong Kong. Configure a ray tracing model to use the shooting and bouncing rays (SBR) method with up to 5 reflections.

Launch Site Viewer with buildings in Hong Kong. For more information about the osm file, see [1] on page 6-180.

```
viewer = siteviewer("Buildings","hongkong.osm");
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Define transmitter and receiver sites to model a small cell scenario in a dense urban environment.

```
tx = txsite("Name","Small cell transmitter", ...
    "Latitude",22.2789, ...
    "Longitude",114.1625, ...
    "AntennaHeight",10, ...
    "TransmitterPower",5, ...
    "TransmitterFrequency",28e9);
rx = rxsite("Name","Small cell receiver", ...
    "Latitude",22.2799, ...
    "Longitude",114.1617, ...
    "AntennaHeight",1);
```

Create a ray tracing propagation model for perfect reflection with up to 5 reflections. Specify the ray tracing method as shooting and bouncing rays (SBR).

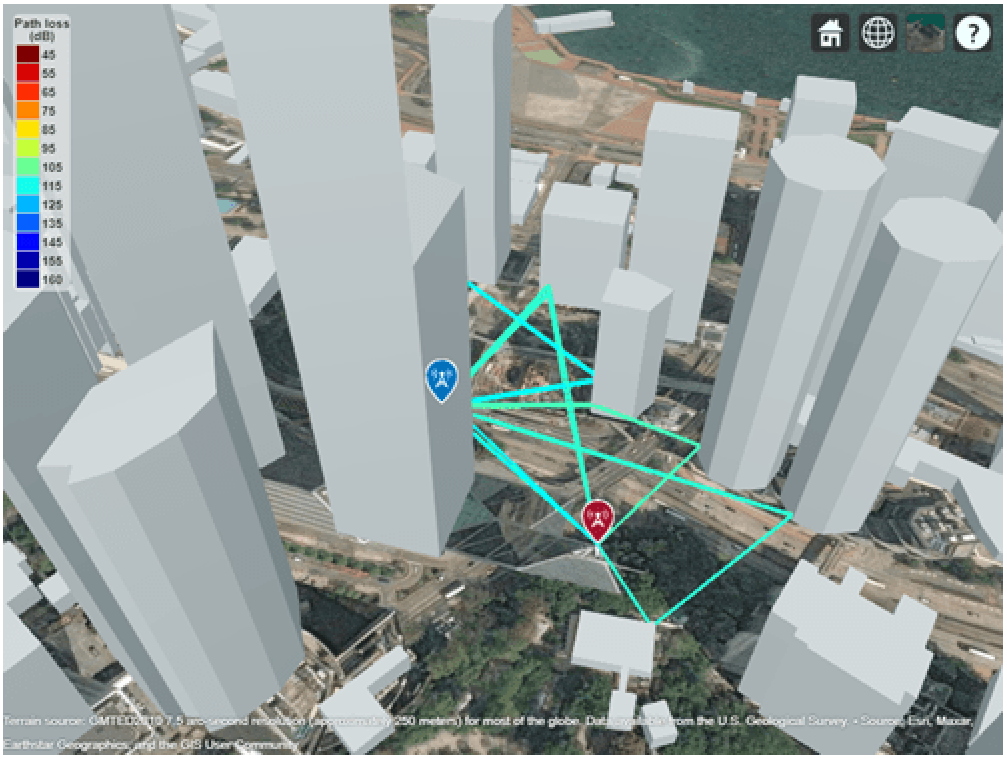
```
pm = propagationModel("raytracing", ...
    "Method","sbr", ...
    "AngularSeparation","low", ...
    "MaxNumReflections",5, ...
    "BuildingsMaterial","perfect-reflector", ...
    "TerrainMaterial","perfect-reflector");
```

Visualize the propagation paths and compute the corresponding path losses.

```
raytrace(tx,rx,pm,"Type","pathloss")
raysPerfect = raytrace(tx,rx,pm,"Type","pathloss");
plPerfect = [raysPerfect{1}.PathLoss]
```

```
plPerfect = 1×13
```

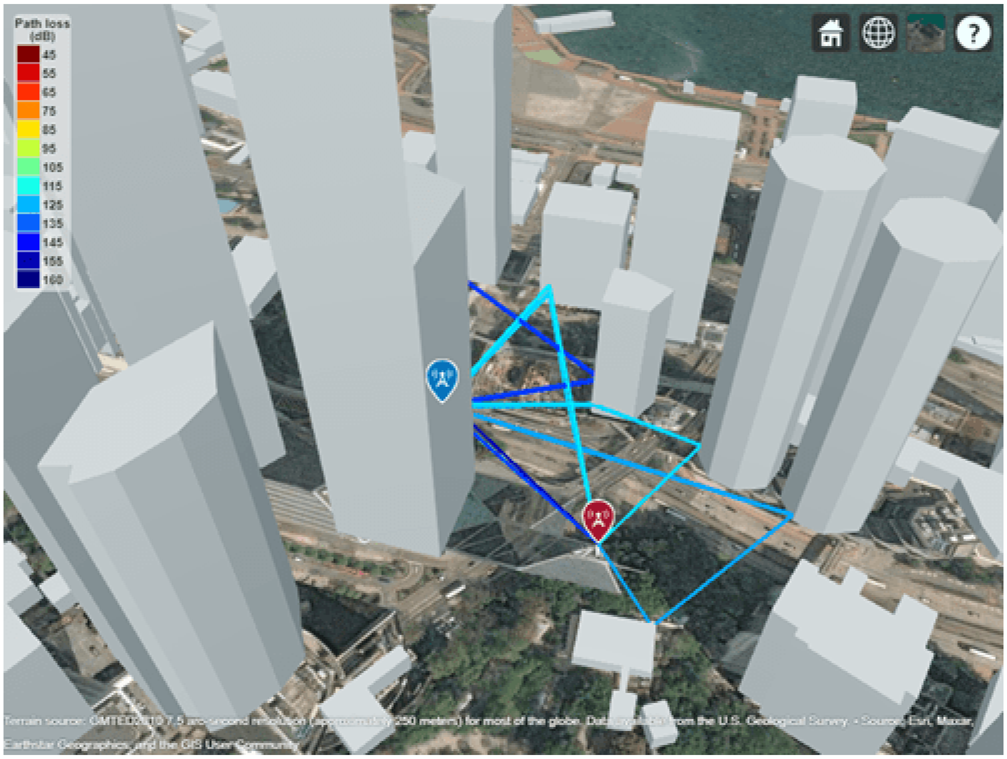
```
  104.2656  103.5720  112.0095  109.3152  111.2814  112.0011  112.4436  108.1516  111.2827  111.3
```


Path loss
(dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Recompute and visualize the propagation paths after configuring material reflection loss by setting building and terrain material types in the propagation model. The first value is unchanged because it corresponds to the line-of-sight propagation path.

```
pm.BuildingsMaterial = "glass";
pm.TerrainMaterial = "concrete";
raytrace(tx,rx,pm,"Type","pathloss")
raysMtrls = raytrace(tx,rx,pm,"Type","pathloss");
plMtrls = [raysMtrls{1}.PathLoss]
```

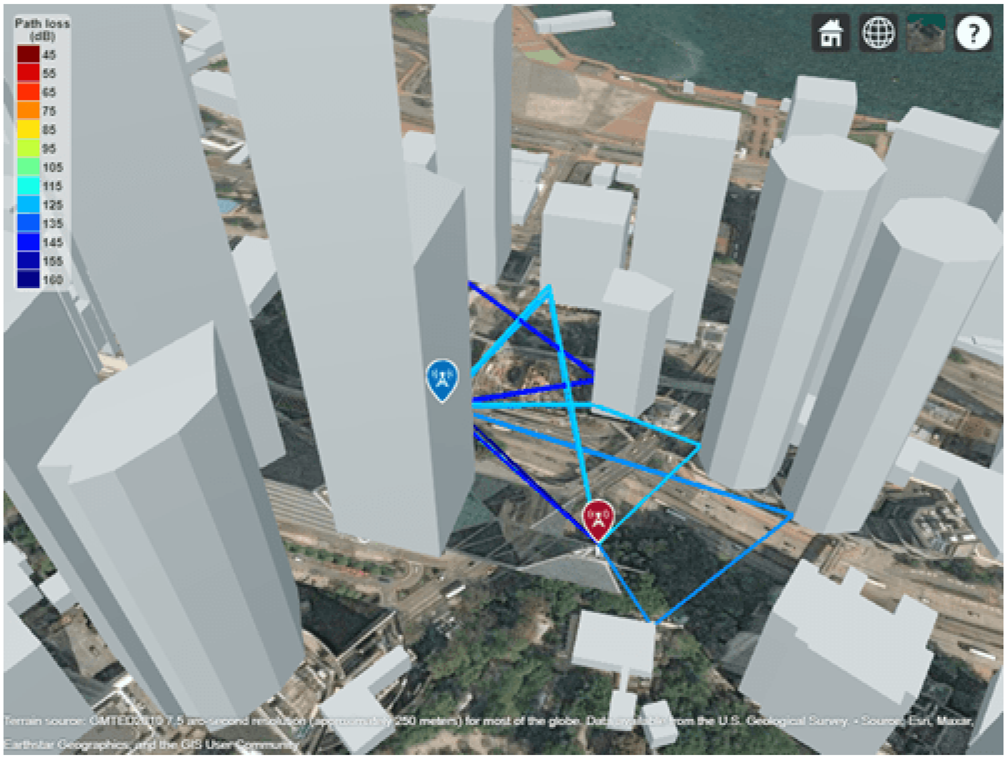
```
plMtrls = 1×13

  104.2656  106.1294  119.2408  121.2477  122.4096  121.5561  126.9482  124.1615  122.8182  127.5
```


Path loss (dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Recompute and visualize the propagation paths with atmospheric loss by adding atmospheric propagation models.

```
pm = pm + propagationModel("rain") + propagationModel("gas");
raytrace(tx,rx,pm,"Type","pathloss")
raysAtmospheric = raytrace(tx,rx,pm,"Type","pathloss");
plAtmospheric = [raysAtmospheric{1}.PathLoss]
```

```
plAtmospheric = 1×13

  105.3245   107.1891   121.8260   123.1432   124.9966   124.1453   129.6661   126.0578   125.4086   130.
```


Path loss (dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Visualize Ray Tracing in Conference Room

This example shows how to:

- Scale an STL file so that the model uses units of meters.
- View the scaled model in Site Viewer.
- Use ray tracing to calculate and display propagation paths from a transmitter to a receiver.

While Cartesian `txsite` and `rxsite` objects require position coordinates in meters, STL files might use other units. If your STL file does not use meters, you must scale the model before importing it into Site Viewer.

Read an STL file as a `triangulation` object. The file models a small conference room with one table and four chairs.

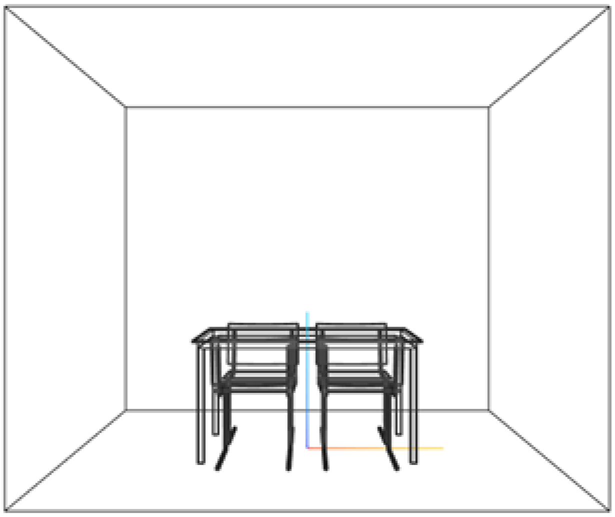
```
TR = stlread("conferenceroom.stl");
```

Scale the coordinates and create a new `triangulation` object. For this example, assume that the conversion factor from the STL units to meters is `0.9`.

```
scale = 0.9;
scaledPts = TR.Points * scale;
TR_scaled = triangulation(TR.ConnectivityList,scaledPts);
```

View the new `triangulation` object using Site Viewer. Alternatively, you can save the new `triangulation` object as an STL file by using the `stlwrite` function.

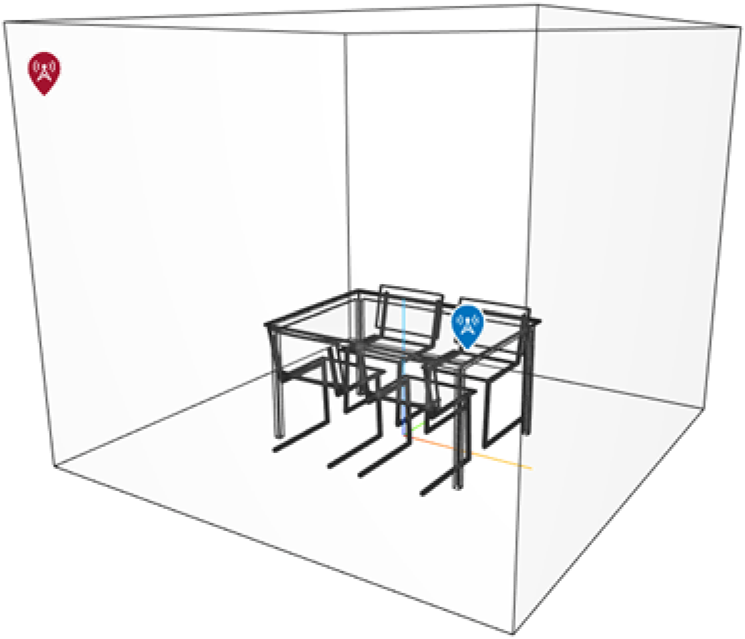
```
viewer = siteviewer("SceneModel",TR_scaled);
```



Create and display a transmitter site close to the wall and a receiver site under the table. Specify the position using Cartesian coordinates in meters.

```
tx = txsite("cartesian", ...
    "AntennaPosition",[-1.25; -1.25; 1.9], ...
    "TransmitterFrequency",2.8e9);
show(tx,"ShowAntennaHeight",false)

rx = rxsite("cartesian", ...
    "AntennaPosition",[0.3; 0.2; 0.5]);
show(rx,"ShowAntennaHeight",false)
```

Pan by left-clicking, zoom by right-clicking or by using the scroll wheel, and rotate the visualization by clicking the middle button and dragging or by pressing **Ctrl** and left-clicking and dragging.



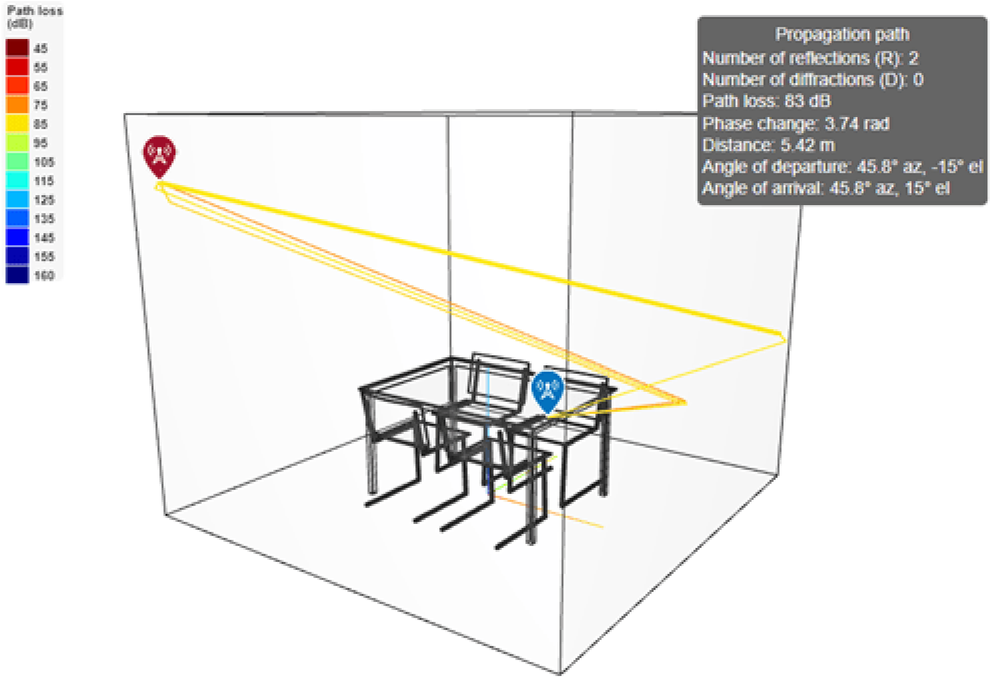
Create a ray tracing propagation model for Cartesian coordinates. Specify the ray tracing method as shooting and bouncing rays (SBR). Calculate rays that have up to 2 reflections. Set the surface material to wood.

```
pm = propagationModel("raytracing", ...
    "CoordinateSystem","cartesian", ...
    "Method","sbr", ...
    "MaxNumReflections",2, ...
    "SurfaceMaterial","wood");
```

Calculate the propagation paths and return the result as a `comm.Ray` object. Extract and plot the rays.

```
r = raytrace(tx,rx,pm);
r = r{1};
plot(r)
```

View information about a ray by clicking on it.


Path loss (dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Propagation path
Number of reflections (R): 2
Number of diffractions (D): 0
Path loss: 83 dB
Phase change: 3.74 rad
Distance: 5.42 m
Angle of departure: 45.8° az, -15° el
Angle of arrival: 45.8° az, 15° el

Input Arguments

`tx` — Transmitter site

`txsite` object | array of `txsite` objects

Transmitter site, specified as a `txsite` object or an array of `txsite` objects. If the receiver sites are specified as arrays, then the propagation paths are plotted from each transmitter to each receiver site.

`rx` — Receiver site

`rxsite` object | array of `rxsite` objects

Receiver site, specified as a `rxsite` object or an array of `rxsite` objects. If the transmitter sites are specified as arrays, then the propagation paths are plotted from each transmitter to each receiver site.

`propmodel` — Propagation model

character vector | string | ray tracing propagation model created with `propagationModel`

Propagation model, specified as a character vector, a string, or a ray tracing propagation model created with the `propagationModel` function. The default is `'raytracing'`, a ray tracing propagation model that uses the SBR method with the maximum number of reflections set to 2.

To specify a ray tracing propagation model that calculates different numbers of reflections, create a `RayTracing` object by using the `propagationModel` function and set the `MaxNumReflections` property.

Name-Value Pair Arguments

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

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

Example: `'Type','power'`

Type — Type of quantity to plot

`'power'` (default) | `'pathloss'`

Type of quantity to plot, specified as the comma-separated pair consisting of `'Type'` and `'power'` in dBm or `'pathloss'` in dB.

When you specify `'power'`, each path is color-coded according to the received power along the path. When you specify `'pathloss'`, each path is color-coded according to the path loss along the path.

Friis equation is used to calculate the received power:

$$P_{rx} = P_{tx} + G_{tx} + G_{rx} - L - L_{tx} - L_{rx}$$

where:

- P_{rx} is the received power along the path.
- P_{tx} is the transmit power defined in tx.TransmitterPower.
- G_{tx} is the antenna gain of tx in the direction of the angle-of-departure (AoD).
- G_{rx} is the antenna gain of rx in the direction of the angle-of-arrival (AoA).
- L is the path loss calculated along the path.
- L_{tx} is the system loss of the transmitter defined in tx.SystemLoss.
- L_{rx} is the system loss of the receiver defined in rx.SystemLoss.

Data Types: `char`

`PropagationModel` — Type of propagation model for ray tracing analysis

`'raytracing'` (default) | ray tracing propagation model created with `propagationModel`

Type of propagation model for ray tracing analysis, specified as the comma-separated pair consisting of `'PropagationModel'` and `'raytracing'` or a ray tracing propagation model created with the `propagationModel` function. If you specify `'raytracing'`, then the `raytrace` function calculates propagation paths by using the SBR method with up to 2 reflections for the ray tracing propagation model object configuration

To perform ray tracing analysis using the image method instead, specify a propagation model created using the `propagationModel` function. This code shows how to create a propagation model that uses the image method.

```
pm = propagationModel('raytracing','Method','image');
```

For information about differences between the image and SBR methods, see “Choose a Propagation Model”.

Data Types: `char`

ColorLimits — Color limits for colormap

two-element numeric row vector

Color limits for colormap, specified as the comma-separated pair consisting of `'ColorLimits'` and a two-element numeric row vector of the form [min max]. The units and default values of the color limits depend on the value of the `'Type'` parameter:

- `'power'`– Units are in dBm, and the default value is `[-120 -5]`.
- `'pathloss'`– Units are in dB, and the default value is `[45 160]`.

The color limits indicate the values that map to the first and last colors in the colormap. Propagation paths with values below the minimum color limit are not plotted.

Data Types: `double`

Colormap — Colormap for coloring propagation paths

`'jet'` (default) | predefined color map name | *M*-by-3 array of RGB

Colormap for coloring propagation paths, specified as the comma-separated pair consisting of `'Colormap'` and a predefined color map name or an *M*-by-3 array of RGB (red, blue, green) triplets that define *M* individual colors.

Data Types: `char` | `double`

ShowLegend — Show color legend on map

`true` (default) | `false`

Show color legend on map, specified as the comma-separated pair consisting of `'ShowLegend'` and `true` or `false`.

Data Types: `logical`

Map — Map for visualization or surface data

`siteviewer` object | `triangulation` object | string scalar | character vector

Map for visualization or surface data, specified as a `siteviewer` object, a `triangulation` object, a string scalar, or a character vector. Valid and default values depend on the coordinate system.

Coordinate System	Valid map values	Default map value
`"geographic"`	• A `siteviewer` object[a]. • A terrain name, if the function is called with an output argument. Valid terrain names are `"none"`, `"gmted2010"`, or the name of the custom terrain data added using `addCustomTerrain`.	• The current `siteviewer` object or a new `siteviewer` object if none are open. • `"gmted2010"`, if the function is called with an output.

Coordinate System	Valid map values	Default map value
"cartesian"	- "none". - A siteviewer object. - The name of an STL file. - A triangulation object.	- "none".

[a] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: char | string

Output Arguments

rays — Ray configuration object

M-by-*N* cell array

Ray configuration, returned as a *M*-by-*N* cell array where *M* is the number of transmitter sites and *N* is the number of receiver sites. Each cell element is a row vector of comm.Ray objects representing all the rays found between the corresponding transmitter site and receiver site. Within each row vector, the comm.Ray objects with the same transmitter to receiver interactions types are grouped together, groups are sorted alphabetically and then by ascending number of reflections. In each group, the rays are ordered by increasing propagation distance.

Version History

Introduced in R2019b

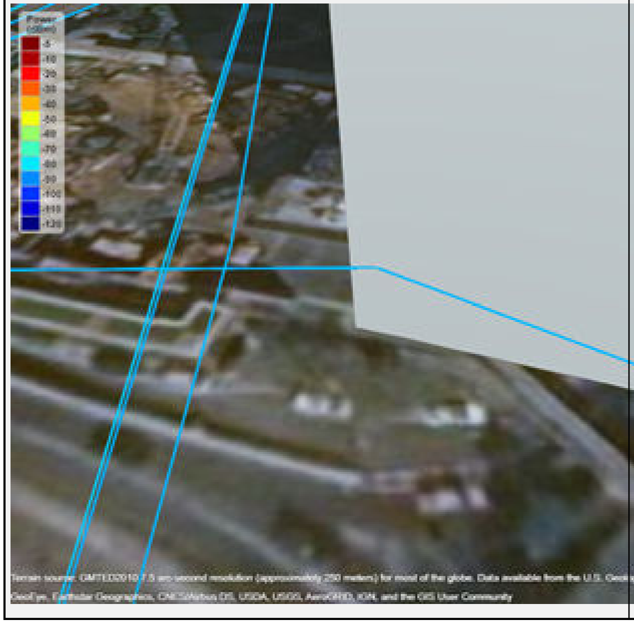
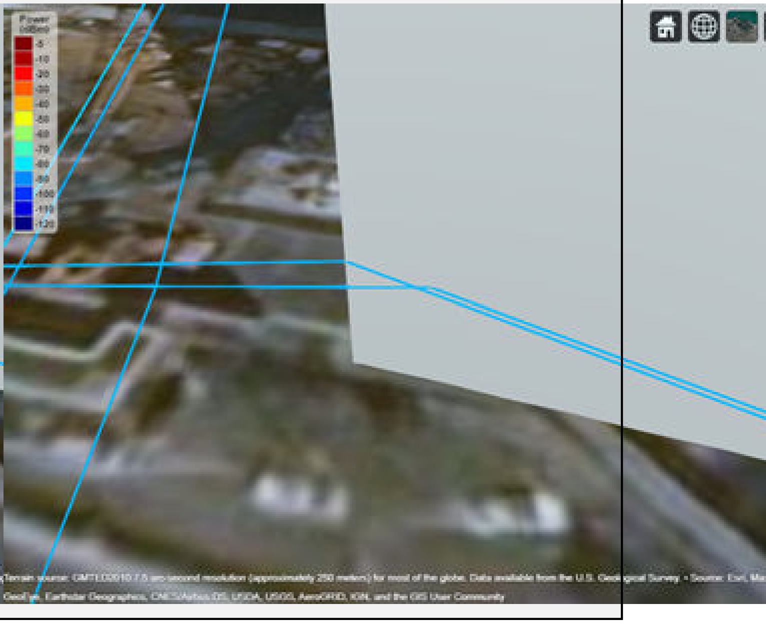
SBR method finds paths with exact geometric accuracy

Behavior changed in R2022b

When you find propagation paths using the SBR method, MATLAB corrects the results so that the geometric accuracy of each path is exact, using single-precision floating-point computations. In previous releases, the paths have approximate geometric accuracy.

For example, this code finds propagation paths between a transmitter and receiver by using the default SBR method and returns the paths as comm.Ray objects. In R2022b, the raytrace function finds seven propagation paths. In earlier releases, the function approximates eight propagation paths, one of which is a duplicate path.

```
viewer = siteviewer(Buildings="hongkong.osm");

tx = txsite(Latitude=22.2789,Longitude=114.1625,AntennaHeight=10, ...
   TransmitterPower=5,TransmitterFrequency=28e9);
rx = rxsite(Latitude=22.2799,Longitude=114.1617,AntennaHeight=1);

rSBR = raytrace(tx,rx)
raytrace(tx,rx)
```

R2022b	R2022a
rSBR = 1×1 cell array {1×7 comm.Ray} 	rSBR = 1×1 cell array {1×8 comm.Ray} 

Paths calculated using the SBR method in R2022b more closely align with paths calculated using the image method. The image method finds all possible paths with exact geometric accuracy. For example, this code uses the image method to find propagation paths between the same transmitter and receiver.

```
viewer = siteviewer(Buildings="hongkong.osm");

tx = txsite(Latitude=22.2789,Longitude=114.1625, ...
   AntennaHeight=10,TransmitterPower=5, ...
   TransmitterFrequency=28e9);
rx = rxsite(Latitude=22.2799,Longitude=114.1617, ...
   AntennaHeight=1);

pm = propagationModel("raytracing",Method="image",MaxNumReflections=2);

rImage = raytrace(tx,rx,pm)
```

```
rImage =

  1×1 cell array

    {1×7 comm.Ray}
```

In this case, the SBR method finds the same number of propagation paths as the image method. In general, the SBR method finds a subset of the paths found by the image method. When both the image and SBR methods find the same path, the points along the path are the same within a tolerance of machine precision for single-precision floating-point values.

This code compares the path losses, within a tolerance of 0.0001, calculated by the SBR and image methods.

```
abs([rSBR{1}.PathLoss]-[rImage{1}.PathLoss]) < 0.0001
```

```
ans =

  1×7 logical array

   1   1   1   1   1   1   1
```

The path losses are the same within the specified tolerance.

As a result, the raytrace function can return different results in R2022b compared to previous releases.

- The function can return a different number of comm.Ray objects because it discards invalid or duplicate paths.
- The function can return different comm.Ray objects because it calculates exact paths rather than approximate paths.

raytrace function uses SBR method

Behavior changed in R2021b

Starting in R2021b, the raytrace function uses the shooting and bouncing rays (SBR) method and calculates up to two reflections by default. In previous releases, the raytrace function uses the image method and calculates up to one reflection.

To display or compute RF propagation rays using the image method instead, create a propagation model by using the propagationModel function. Then, use the raytrace function with the propagation model as input. This example shows how to update your code.

```
pm = propagationModel('raytracing','Method','image');
raytrace(tx,rx,pm)
```

For information about the SBR and image methods, see “Choose a Propagation Model”.

Starting in R2021b, all RF Propagation functions use the SBR method by default and calculate up to two reflections. For more information, see “Default modeling method is shooting and bouncing rays method” on page 6-129.

NumReflections name-value argument will be removed

Warns starting in R2022a

The NumReflections name-value argument will be removed in a future release. The NumReflections name-value argument now only applies for the image ray tracing method. Instead, create a propagation model by using the propagationModel function with its MaxNumReflections name-value argument. Then, use the raytrace function with the propagation model as an input. This example shows the recommended workflow.

```
pm = propagationModel('raytracing', ...
    'Method','image','MaxNumReflections',2);
rays = raytrace(tx,rx,pm);
```

See Also

Functions
`propagationModel` | `los` | `sigstrength`

Objects
`siteviewer` | `rxsite` | `txsite`

Topics
"Choose a Propagation Model"

addCustomBasemap

Add custom basemap

Syntax

```
addCustomBasemap(basemapName,URL)
addCustomBasemap( ___ ,Name,Value)
```

Description

`addCustomBasemap(basemapName,URL)` adds the custom basemap specified by `URL` to the list of basemaps available for use with mapping functions. `basemapName` is the name you choose to call the custom basemap. Added basemaps remain available for use in future MATLAB sessions.

`addCustomBasemap( ___ ,Name,Value)` specifies name-value arguments that set additional parameters of the basemap.

Examples

Add and Remove a Custom Basemap

Add a custom basemap to view locations on an OpenTopoMap® basemap, then remove the custom basemap from `siteviewer`.

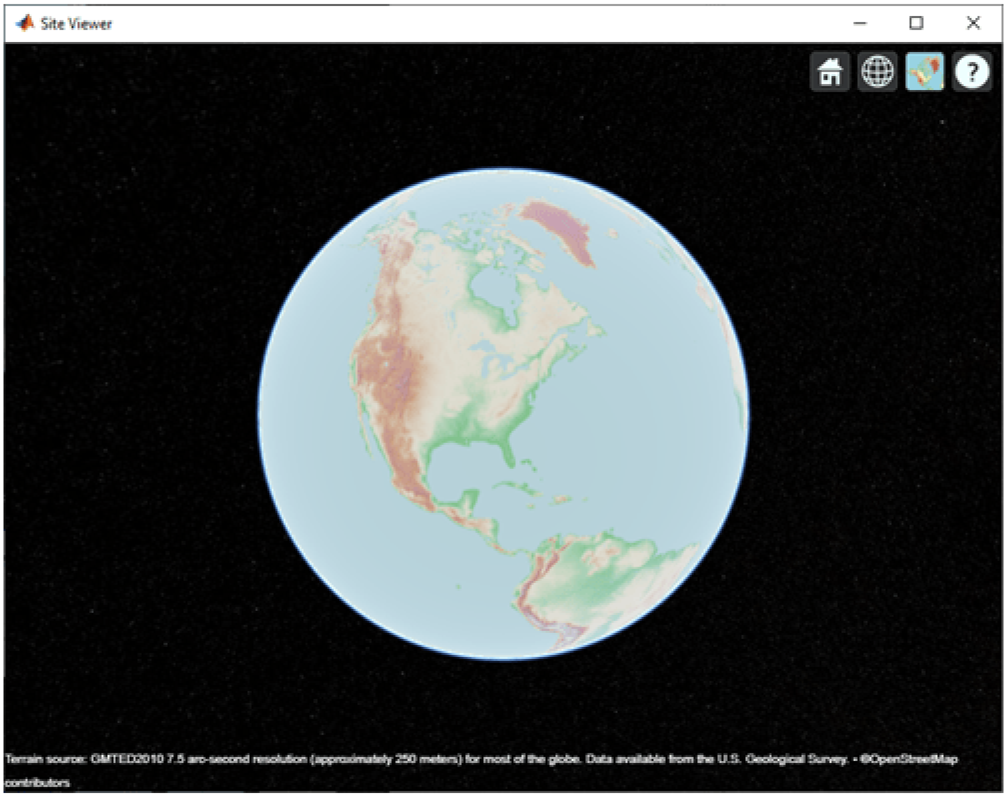
Initialize simulation variables to:

- Define the name that you will use to specify your custom basemap.
- Specify the website that provides the map data. The first character of the URL indicates which server to use to get the data. For load balancing, the provider has three servers that you can use: a, b, or c.
- Create an attribution to display on the map that gives credit to the provider of the map data. Web map providers might define specific requirements for the attribution.
- Define a display name for the custom map.

```
name = 'opentopomap';
url = 'a.tile.opentopomap.org';
copyright = char(uint8(169));
attribution = copyright + "OpenStreetMap contributors";
displayName = 'Open Topo Map';
```

Use `addCustomBasemap` to load the custom basemap, and then create a `siteviewer` object that loads the custom basemap.

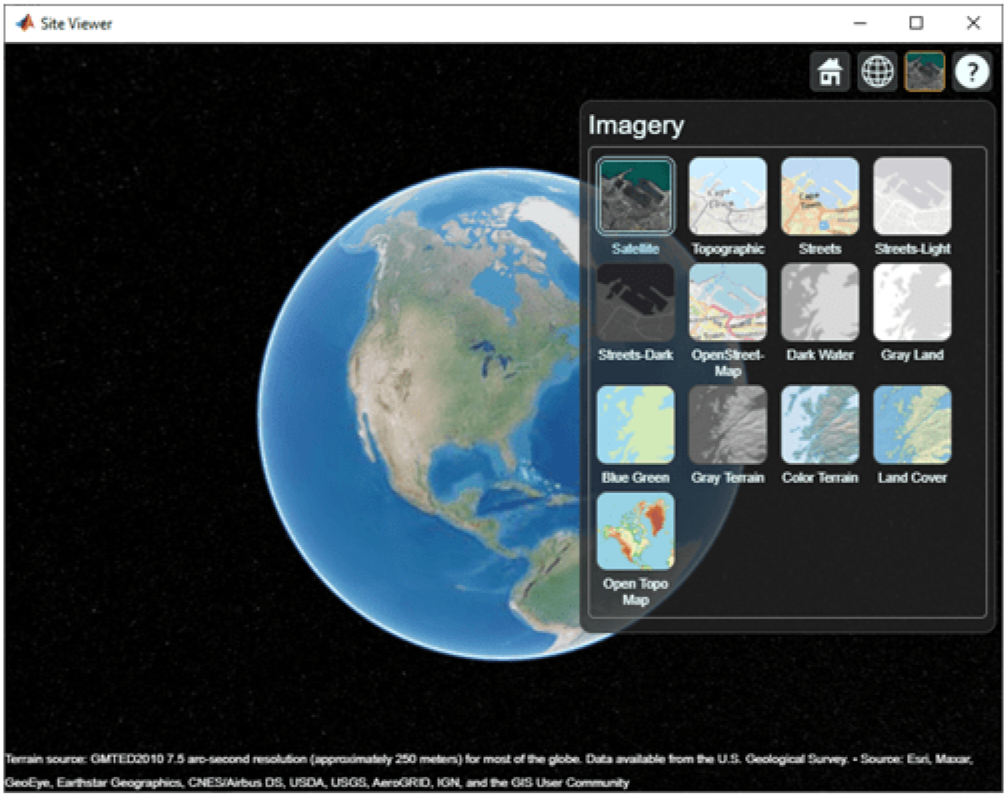
```
addCustomBasemap(name,url,'Attribution',attribution','DisplayName',displayName)
viewer = siteviewer('Basemap',name);
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • ©OpenStreetMap contributors

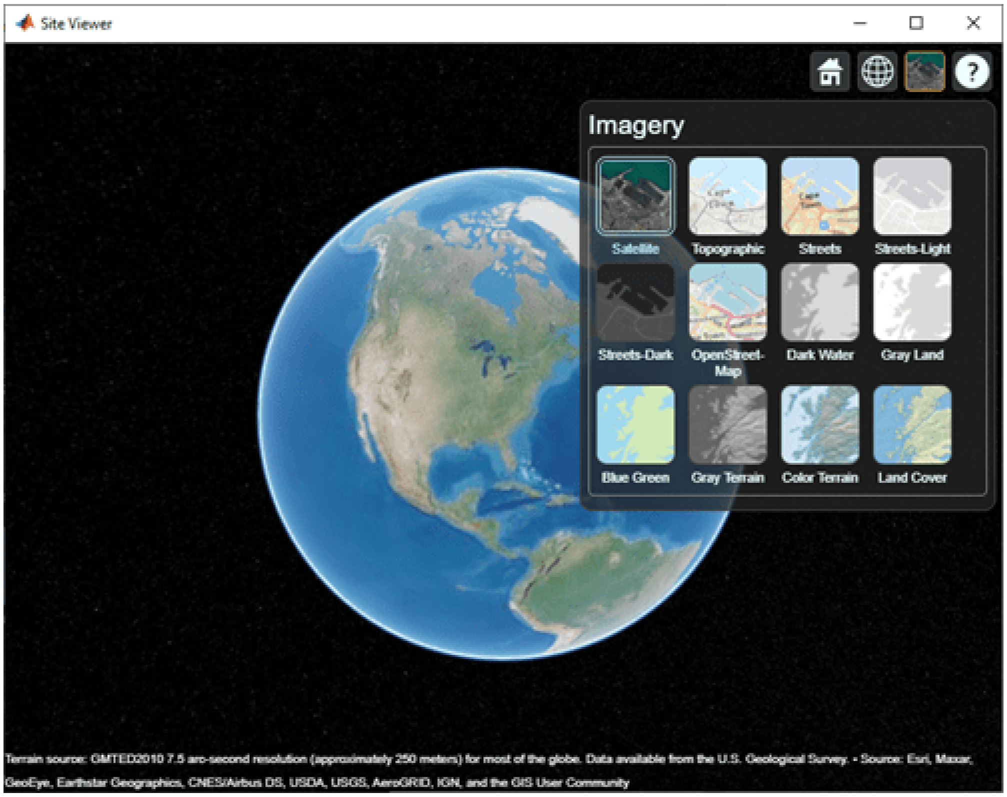
After a custom basemap is added to `siteviewer`, the custom map is available for future calls to `siteviewer`. Note the `'Open Topo Map'` icon in the `Imagery` tab.

```
siteviewer;
```


Site Viewer
Imagery
Satellite
Topographic
Streets
Streets-Light
Streets-Dark
OpenStreet-Map
Dark Water
Gray Land
Blue Green
Gray Terrain
Color Terrain
Land Cover
Open Topo Map
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Use `removeCustomBasemap` to remove the custom basemap from future calls to `siteviewer`. Note the `'Open Topo Map'` icon is no longer available in the `Imagery` tab.

```
removeCustomBasemap(name)
siteviewer;
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

basemapName — Name used to identify basemap programmatically

string scalar | character vector

Name used to identify basemap programmatically, specified as a string scalar or character vector.

Example: `'openstreetmap'`

Data Types: `string` | `char`

URL — Parameterized map URL

string scalar | character vector

Parameterized map URL, specified as a string scalar or character vector. A parameterized URL is an index of the map tiles, formatted as `${z}/${x}/${y}.png` or `{z}/{x}/{y}.png`, where:

- `${z}` or `{z}` is the tile zoom level.
- `${x}` or `{x}` is the tile column index.
- `${y}` or `{y}` is the tile row index.

Example: `'https://hostname/${z}/${x}/${y}.png'`

Data Types: `string` | `char`

Name-Value Arguments

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

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

Example: `addCustomBasemap(basemapName,URL,"Attribution",attribution)`

`Attribution` — Attribution of custom basemap

string scalar | string array | character vector | cell array of character vectors

Attribution of the custom basemap, specified as a string scalar, string array, character vector, or cell array of character vectors. To create a multiline attribution, specify a string array or nonscalar cell array of character vectors.

When you create a custom basemap from a URL, the default attribution is `'Tiles courtesy of` *`DOMAIN_NAME_OF_URL`*`'`, where *`DOMAIN_NAME_OF_URL`* is the domain name from the `URL` input argument. If the host is `'localhost'`, or if `URL` contains only IP numbers, specify the attribution as an empty string (`""`).

Example: `"Credit: U.S. Geological Survey"`

Data Types: `string` | `char` | `cell`

`DisplayName` — Display name of custom basemap

string scalar | character vector

Display name of the custom basemap, specified as a string scalar or character vector.

Example: `"OpenStreetMap"`

Data Types: `string` | `char`

`MaxZoomLevel` — Maximum zoom level of basemap

`18` (default) | integer in range [0, 25]

Maximum zoom level of the basemap, specified as an integer in the range [0, 25].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`IsDeployable` — Map is deployable using MATLAB Compiler™

`false` or `0` (default) | `true` or `1`

Map is deployable using MATLAB Compiler, specified as a numeric or logical `0` (`false`) or `1` (`true`).

Data Types: `logical`

Limitations

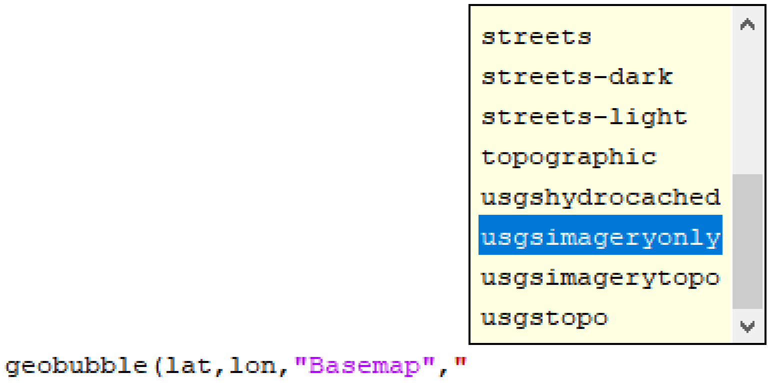
The `addCustomBasemap` function does not support adding custom basemaps from vector map tiles.

Tips

- You can find tiled web maps from various vendors, such as OpenStreetMap, the USGS National Map, Mapbox, DigitalGlobe, Esri ArcGIS Online, the Geospatial Information Authority of Japan

(GSI), and HERE Technologies. Abide by the map vendors terms-of-service agreement and include accurate attribution with the maps you use.

- If you have Mapping Toolbox™, you can create custom basemaps from MBTiles files. For more information, see `addCustomBasemap`.
- To access a list of available basemaps, press **Tab** before specifying the basemap in your plotting function. This image shows a sample list of available basemaps, including several custom basemaps from the USGS National Map.



```
geobubble(lat,lon,"Basemap","
```

See Also

`geobasemap` | `geobubble` | `removeCustomBasemap` | `readBasemapImage`

Topics

"Use Basemaps in Offline Environments" (Mapping Toolbox)

removeCustomBasemap

Remove custom basemap

Syntax

`removeCustomBasemap(basemapName)`

Description

`removeCustomBasemap(basemapName)` removes the custom basemap specified by `basemapName` from the list of available basemaps.

Examples

Add and Remove a Custom Basemap

Add a custom basemap to view locations on an OpenTopoMap® basemap, then remove the custom basemap from `siteviewer`.

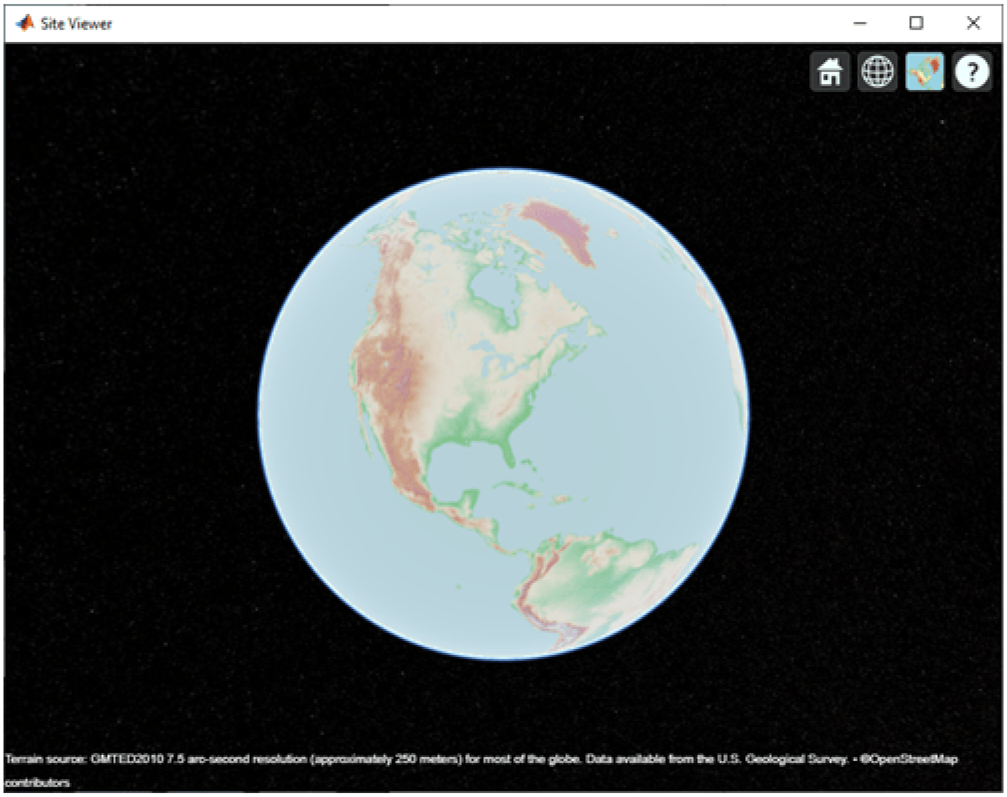
Initialize simulation variables to:

- Define the name that you will use to specify your custom basemap.
- Specify the website that provides the map data. The first character of the URL indicates which server to use to get the data. For load balancing, the provider has three servers that you can use: a, b, or c.
- Create an attribution to display on the map that gives credit to the provider of the map data. Web map providers might define specific requirements for the attribution.
- Define a display name for the custom map.

```
name = 'opentopomap';
url = 'a.tile.opentopomap.org';
copyright = char(uint8(169));
attribution = copyright + "OpenStreetMap contributors";
displayName = 'Open Topo Map';
```

Use `addCustomBasemap` to load the custom basemap, and then create a `siteviewer` object that loads the custom basemap.

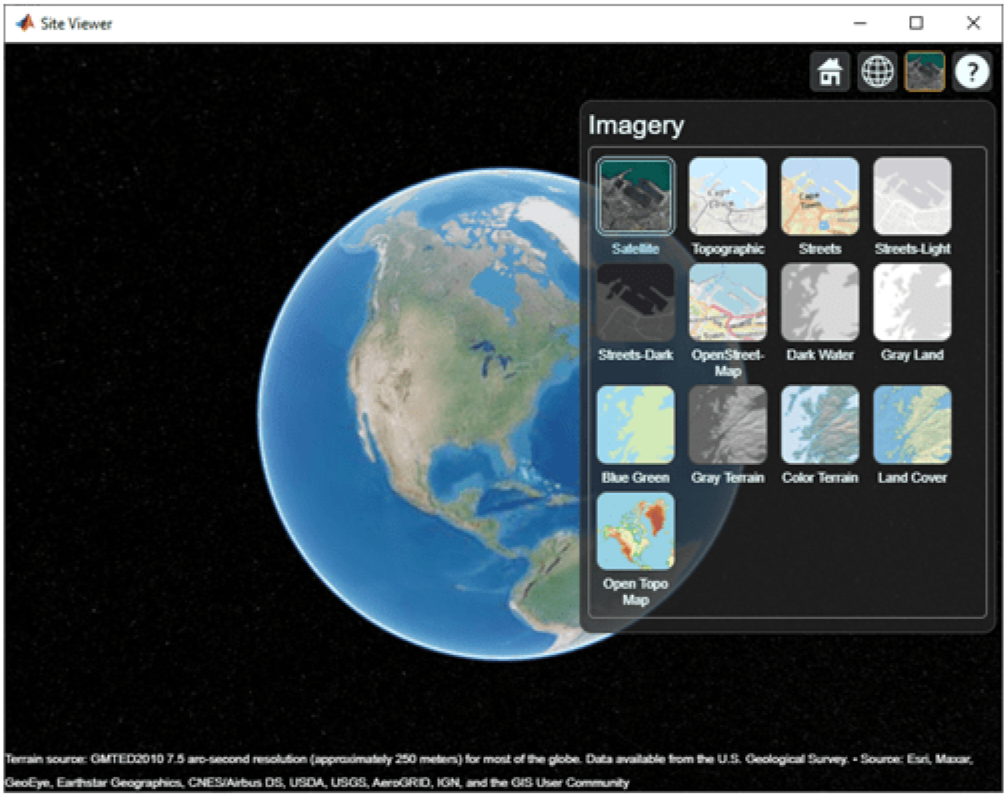
```
addCustomBasemap(name,url,'Attribution',attribution','DisplayName',displayName)
viewer = siteviewer('Basemap',name);
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • ©OpenStreetMap contributors

After a custom basemap is added to `siteviewer`, the custom map is available for future calls to `siteviewer`. Note the `'Open Topo Map'` icon in the `Imagery` tab.

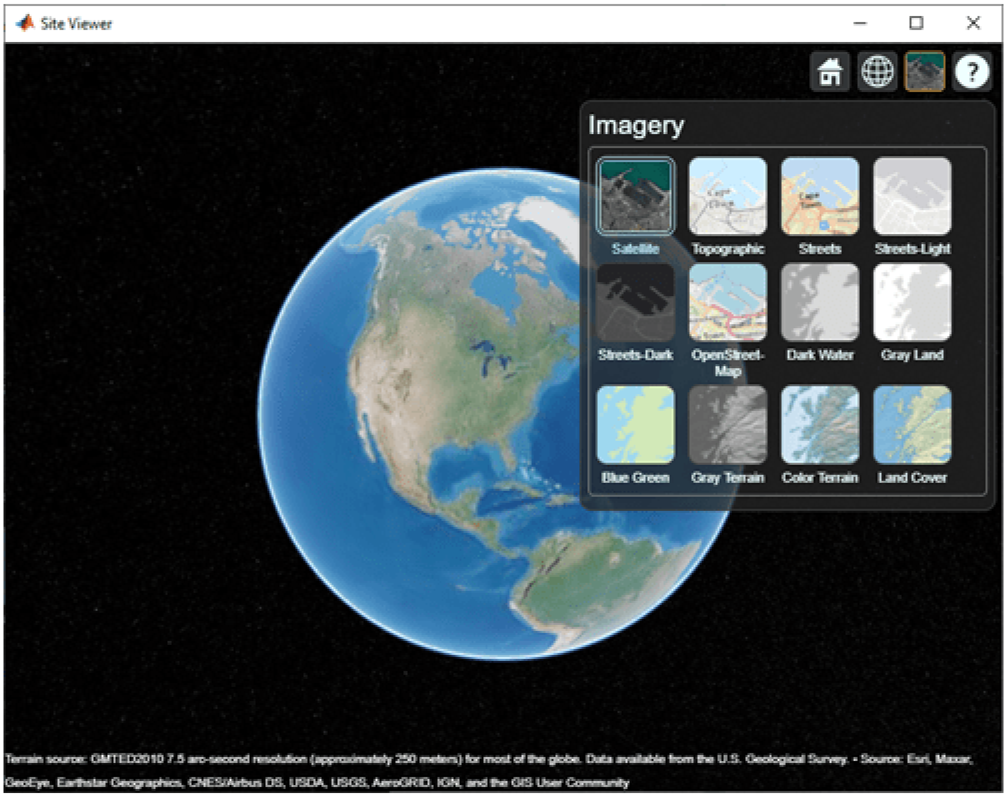
```
siteviewer;
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Use `removeCustomBasemap` to remove the custom basemap from future calls to `siteviewer`. Note the `'Open Topo Map'` icon is no longer available in the `Imagery` tab.

```
removeCustomBasemap(name)
siteviewer;
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

basemapName — Name of custom basemap

string scalar | character vector

Name of the custom basemap to remove, specified as a string scalar or character vector. You define the basemap name when you add the basemap using the `addCustomBasemap` function.

Data Types: `string` | `char`

See Also

`geoaxes` | `geobasemap` | `geobubble` | `geodensityplot` | `geoplot` | `geoscatter` | `addCustomBasemap`

buildingMaterialPermittivity

Permittivity and conductivity of building materials

Syntax

```
[epsilon,sigma,complexepsilon] = buildingMaterialPermittivity(material,fc)
```

Description

`[epsilon,sigma,complexepsilon] = buildingMaterialPermittivity(material,fc)` calculates the relative permittivity, conductivity, and complex relative permittivity for the specified material at the specified frequency. The methods and equations modeled in the `buildingMaterialPermittivity` function are presented in Recommendation ITU-R P.2040 [1].

Examples

Calculate Permittivity of Various Building Materials

Calculate relative permittivity and conductivity at 9 GHz for various building materials as defined by textual classifications in ITU-R P.2040, Table 3.

```
material = ["vacuum";"concrete";"brick";"plasterboard";"wood"; ...
    "glass";"ceiling-board";"chipboard";"floorboard";"metal"];
fc = repmat(9e9,size(material)); % Frequency in Hz
[permittivity,conductivity] = ...
    arrayfun(@(x,y)buildingMaterialPermittivity(x,y),material,fc);
```

Display the results in a table.

```
varNames = ["Material";"Permittivity";"Conductivity"];
table(material,permittivity,conductivity,'VariableNames',varNames)
```

```
ans=10×3 table
       Material        Permittivity    Conductivity
    _______________    ____________    ____________

    "vacuum"                 1                 0
    "concrete"            5.31           0.19305
    "brick"               3.75             0.038
    "plasterboard"        2.94          0.054914
    "wood"                1.99          0.049528
    "glass"               6.27          0.059075
    "ceiling-board"        1.5         0.0064437
    "chipboard"           2.58           0.12044
    "floorboard"          3.66          0.085726
    "metal"                  1             1e+07
```

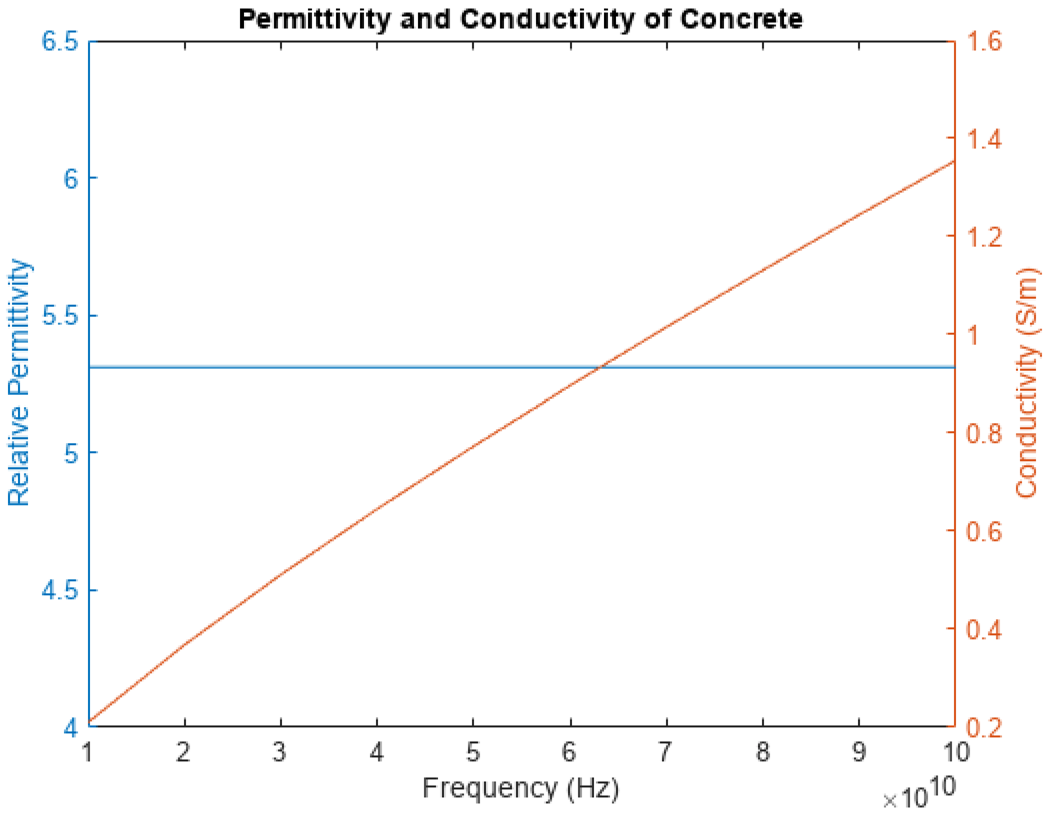
Plot Permittivity and Conductivity of Concrete at Various Frequencies

Calculate the relative permittivity and conductivity for concrete at frequencies specified.

```
fc = ((1:1:10)*10e9); % Frequency in Hz
[permittivity,conductivity] = ...
    arrayfun(@(y)buildingMaterialPermittivity("concrete",y),fc);
```

Plot the relative permittivity and conductivity of concrete across the range of frequencies.

```
figure
yyaxis left
plot(fc,permittivity)
ylabel('Relative Permittivity')
yyaxis right
plot(fc,conductivity)
ylabel('Conductivity (S/m)')
xlabel('Frequency (Hz)')
title('Permittivity and Conductivity of Concrete')
```



Input Arguments

material — Building material
"vacuum" | "concrete" | "brick" | "plasterboard" | ...

Building material, specified as vector of strings, or an equivalent character vector or cell array of character vectors including one or more of these options:

`"vacuum"`	`"glass"`	`"very-dry-ground"`
`"concrete"`	`"ceiling-board"`	`"medium-dry-ground"`
`"brick"`	`"floorboard"`	`"wet-ground"`
`"plasterboard"`	`"chipboard"`	
`"wood"`	`"metal"`	

Example: `["vacuum" "brick"]`

Data Types: `char` | `string`

`fc` — Carrier frequency
positive scalar

Carrier frequency in Hz, specified as a positive scalar.

Note `fc` must be in the range [1e6, 10e6] when the `material` is `"very-dry-ground"`, `"medium-dry-ground"` or `"wet-ground"`.

Data Types: `double`

Output Arguments

`epsilon` — Relative permittivity
nonnegative scalar | nonnegative row vector

Relative permittivity of the building material, returned as a nonnegative scalar or row vector. The output dimension of `epsilon` matches that of the input argument `material`. For more information about the computation for the relative permittivity, see “ITU Building Materials” on page 6-203.

`sigma` — Conductivity
nonnegative scalar | nonnegative row vector

Conductivity, in Siemens/m, of the building material, returned as a nonnegative scalar or row vector. The output dimension of `sigma` matches that of the input argument `material`. For more information about the computation for the conductivity, see “ITU Building Materials” on page 6-203.

`complexepsilon` — Complex relative permittivity
complex scalar | row vector of complex values

Complex relative permittivity of the building material, returned as a complex scalar or row vector of complex values.

The output dimension of `complexepsilon` matches that of the input argument `material`. For more information about the computation for the complex relative permittivity, see “ITU Building Materials” on page 6-203.

More About

ITU Building Materials

Section 3 of ITU-R P.2040-1 [1] presents methods, equations, and values used to calculate real relative permittivity, conductivity, and complex relative permittivity at carrier frequencies up to 100 GHz for common building materials.

The `buildingMaterialPermittivity` function uses equations from ITU-R P.2040-1 to compute these values.

- The real part of the relative permittivity is calculated as
  `epsilon` = af^b.
  The computation of `epsilon` is based on equation (58). *f* is the frequency in GHz. Values for *a* and *b* are specified in Table 3 from ITU-R P.2040-1.
- The conductivity in Siemens/m is calculated as
  `sigma` = cf^d.
  The computation of `sigma` is based on equation (59). *f* is the frequency in GHz. Values for *c* and *d* are specified in Table 3 from ITU-R P.2040-1.
- The complex permittivity is calculated as
  `complexepsilon` = `epsilon` – 1*i* `sigma` / ($2\pi fc\varepsilon_0$).
  The computation of `complexepsilon` is based on Equations (59) and (9b). *f* is the frequency in GHz. *c* is the velocity of light in free space. ε_0 = 8.854187817e-12 Farads/m, where ε_0 is the electric constant for the permittivity of free space.

For cases where the value of *b* or *d* is zero, the corresponding value of `epsilon` or `sigma` is *a* or *c*, respectively and independent of frequency.

The contents of Table 3 from ITU-R P.2040-1 are repeated in this table. The values *a*, *b*, *c*, and *d* are used to calculate relative permittivity and conductivity. Except as noted for the three ground types, the frequency ranges given in the table are not hard limits but are indicative of the measurements used to derive the models. The `buildingMaterialPermittivity` function interpolates or extrapolates relative permittivity and conductivity values for frequencies that fall outside of the noted limits. To compute relative permittivity and conductivity for different types of ground as a function carrier frequencies up to 1000 GHz, see the `earthSurfacePermittivity` function.

Material Class	Real Part of Relative Permittivity		Conductivity (S/m)		Frequency Range (GHz)
	a	***b***	***c***	***d***	
Vacuum (~ air)	1	0	0	0	[0.001, 100]
Concrete	5.31	0	0.0326	0.8095	[1, 100]
Brick	3.75	0	0.038	0	[1, 10]
Plasterboard	2.94	0	0.0116	0.7076	[1, 100]
Wood	1.99	0	0.0047	1.0718	[0.001, 100]
Glass	6.27	0	0.0043	1.1925	[0.1, 100]
Ceiling board	1.50	0	0.0005	1.1634	[1, 100]
Chipboard	2.58	0	0.0217	0.78	[1, 100]
Floorboard	3.66	0	0.0044	1.3515	[50, 100]

Material Class	Real Part of Relative Permittivity		Conductivity (S/m)		Frequency Range (GHz)
	a	***b***	***c***	***d***	
Metal	1	0	10^7	0	[1, 100]
Very dry ground	3	0	0.00015	2.52	[1, 10] only(a)
Medium dry ground	15	– 0.1	0.035	1.63	[1, 10] only(a)
Wet ground	30	– 0.4	0.15	1.30	[1, 10] only(a)
Note (a): For the three ground types (very dry, medium dry, and wet), the noted frequency limits cannot be exceeded.					

Version History

Introduced in R2020a

References

[1] International Telecommunications Union Radiocommunication Sector. *Effects of building materials and structures on radiowave propagation above about 100MHz.* Recommendation P.2040-1. ITU-R, approved July 29, 2015. https://www.itu.int/rec/R-REC-P.2040-1-201507-I/en.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

When you specify multiple reflective materials, you must define each value as a character vector (`char` data type) in a cell array.

See Also

Functions

`earthSurfacePermittivity` | `raytrace` | `raypl` | `propagationModel`

Objects

`comm.Ray`

earthSurfacePermittivity

Permittivity and conductivity of earth surface materials

Syntax

```
[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('pure-water',fc,
temp)
[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('dry-ice',fc,temp)
[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('sea-water',fc,
temp,salinity)
[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('wet-ice',fc,
liqfrac)
```

```
[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('soil',fc,temp,
sandpercent,claypercent,specificgravity,vwc)
[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('soil',___,
bulkdensity)
```

```
[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('vegetation',fc,
temp,gwc)
```

Description

The `earthSurfacePermittivity` function computes electrical characteristics (relative permittivity, conductivity, and complex relative permittivity) of earth surface materials based on the methods and equations presented in ITU-R P.527 [1]. The `earthSurfacePermittivity` function provides various syntaxes to account for characteristics germane to the specified surface material.

`[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('pure-water',fc,temp)` calculates the electrical characteristics for pure water at the specified frequency and temperature. For pure-water, the temperature setting must be greater than 0 ℃.

`[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('dry-ice',fc,temp)` calculates the electrical characteristics for dry-ice at the specified frequency and temperature. For dry-ice, the temperature must be less than or equal to 0 ℃.

`[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('sea-water',fc,temp,salinity)` calculates the electrical characteristics for sea water at the specified frequency, temperature, and salinity. For sea-water, the temperature must be greater than –2 ℃.

`[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('wet-ice',fc,liqfrac)` calculates the electrical characteristics for wet ice at the specified frequency, and liquid water volume fraction. For wet-ice, the temperature is 0 ℃.

`[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('soil',fc,temp,sandpercent,claypercent,specificgravity,vwc)` calculates the electrical characteristics for soil at the specified frequency, temperature, sand percentage, clay percentage, specific gravity, and volumetric water content.

`[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('soil',___,bulkdensity)` sets the soil bulk density in addition to input arguments from the previous syntax.

`[epsilon,sigma,complexepsilon] = earthSurfacePermittivity('vegetation',fc, temp,gwc)` calculates the electrical characteristics for vegetation at the specified frequency, temperature, and gravimetric water content. For vegetation, the temperature must be greater than or equal to –20 ℃.

Examples

Compare Permittivity and Conductivity of Salt-free Sea Water to Pure Water

Compare the relative permittivity and conductivity for salt-free (zero-salinity) sea water to pure water.

Specify a carrier frequency of 9 GHz, temperature of 30℃, and salinity of zero.

```
fc = 9e9; % Carrier frequency in Hz.
temp = 30;
salinity = 0;
```

Compute the relative permittivity and conductivity.

```
[epsilon_pure_water,sigma_pure_water] = earthSurfacePermittivity('pure-water',fc,temp);
[epsilon_sea_water,sigma_sea_water] = earthSurfacePermittivity('sea-water',fc,temp,salinity);
```

Confirm that salt-free sea water and pure water have equal relative permittivity and conductivity.

```
isequal(epsilon_pure_water,epsilon_sea_water)
```

```
ans = logical
   1
```

```
isequal(sigma_pure_water,sigma_sea_water)
```

```
ans = logical
   1
```

Compare Permittivity and Conductivity of Wet Ice to Dry Ice

Compare the relative permittivity and conductivity for wet ice with no liquid water to dry ice at 0℃. Confirm the results differ by a negligible amount.

Specify a carrier frequency of 12 GHz.

```
fc = 12e9; % Carrier frequency in Hz.
```

Calculate the relative permittivity and conductivity for wet ice with zero liquid water by volume.

```
liqfrac = 0;
[epsilon_wet_ice_0,sigma_wet_ice_0] = earthSurfacePermittivity('wet-ice',fc,liqfrac); % Set liqu
```

Calculate the relative permittivity and conductivity for dry ice at 0 ℃.

```
temp = 0;
[epsilon_dry_ice_0,sigma_dry_ice_0] = earthSurfacePermittivity('dry-ice',fc,temp); % Set temperat
```

Compare the relative permittivity and conductivity for wet ice with no liquid to dry ice at 0°C. Confirm that wet ice with no liquid and dry ice at 0°C have essentially equal relative permittivity and conductivity.

```
epsilon_wet_ice_0-epsilon_dry_ice_0
```

```
ans = 8.8818e-16
```

```
sigma_wet_ice_0-sigma_dry_ice_0
```

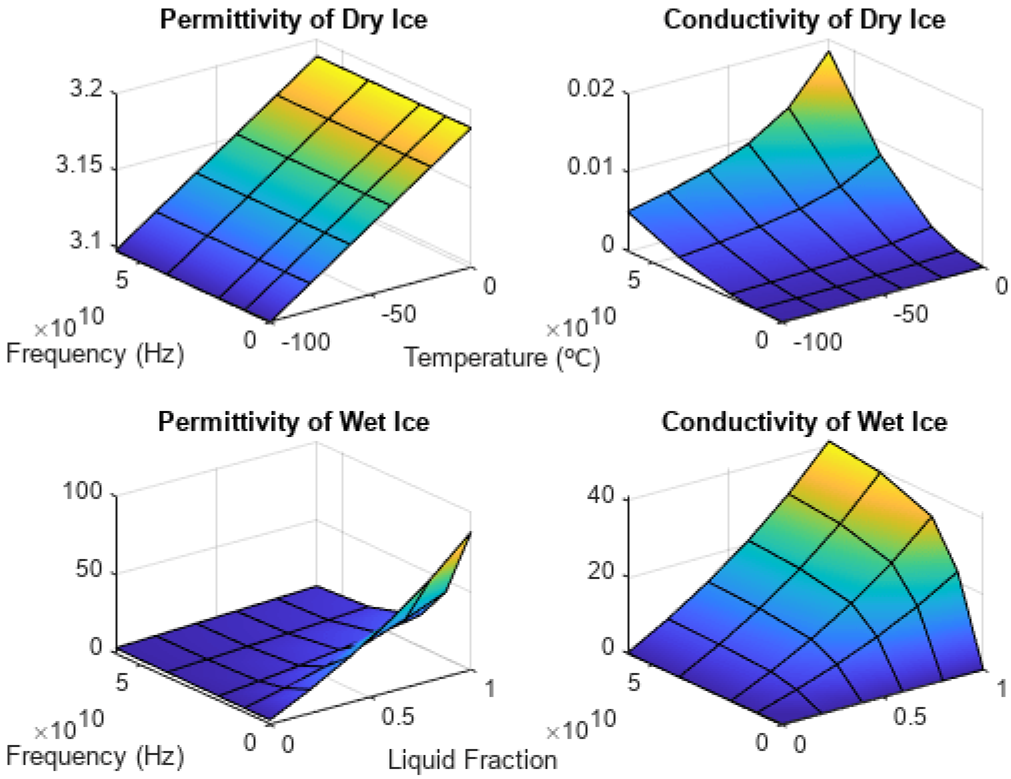
```
ans = -9.2179e-16
```

Plot permittivity and conductivity versus frequency for dry ice and for wet ice. For dry ice, vary the temperature. For wet ice, vary the liquid water volume fraction. Calculate the permittivity and conductivity values by using `arrayfun` to apply the `earthSurfacePermittivity` function to the elements of the arrayed inputs.

```
freq = repmat([0.1,10,20,40,60]*1e9,6,1);
temp = repmat((-100:20:0)',1,5);
liqfrac = repmat((0:0.2:1)',1,5);
[epsilon_dry_ice, sigma_dry_ice] = arrayfun(@(x,y)earthSurfacePermittivity('dry-ice',x,y),freq,te
[epsilon_wet_ice, sigma_wet_ice] = arrayfun(@(x,y)earthSurfacePermittivity('wet-ice',x,y),freq,li
```

Display tiled surface plots across specified ranges.

```
figure
tiledlayout(2,2)
nexttile
surf(temp,freq,epsilon_dry_ice,'FaceColor','interp')
title('Permittivity of Dry Ice')
xlabel('Temperature (℃)')
ylabel('Frequency (Hz)')
nexttile
surf(temp,freq,sigma_dry_ice,'FaceColor','interp')
title('Conductivity of Dry Ice')
nexttile
surf(liqfrac,freq,epsilon_wet_ice,'FaceColor','interp')
title('Permittivity of Wet Ice')
xlabel('Liquid Fraction')
ylabel('Frequency (Hz)')
nexttile
surf(liqfrac,freq,sigma_wet_ice,'FaceColor','interp')
title('Conductivity of Wet Ice')
```



Calculate Permittivity and Conductivity of Various Soil Mixtures

Calculate relative permittivity and conductivity for various soil mixtures as defined by textual classifications in ITU-R P.527, Table 1.

Initialize computation variables for constant values and arrayed values.

```
fc = 28e9; % Frequency in Hz
temp = 23; % Temperature in °C
vwc = 0.5; % Volumetric water content
pSand = [51.52; 41.96; 30.63; 5.02]; % Sand percentage
pClay = [13.42; 8.53; 13.48; 47.38]; % Clay percentage
sg = [2.66; 2.70; 2.59; 2.56]; % Specific gravity
bd = [1.6006; 1.5781; 1.5750; 1.4758]; % Bulk density (g/cm^3)
```

Calculate the relative permittivity and conductivity for these textual classifications: sandy loam, loam, silty loam, and silty clay. Use `arrayfun` to apply the `earthSurfacePermittivity` function to the elements of the arrayed inputs. Tabulate the results.

```
[Permittivity,Conductivity] = arrayfun(@(w,x,y,z)earthSurfacePermittivity( ...
    'soil',fc,temp,w,x,y,vwc,z),pSand,pClay,sg,bd);

pSilt = 100 - (pSand + pClay); % Silt percentage
soilType = ["Sandy Loam";"Loam";"Silty Loam";"Silty Clay"];
```

```
varNames1 = ["Soil Textual Classification";"Sand";"Clay";"Silt";"Specific Gravity";"Bulk Density"
varNames2 = ["Soil Textual Classification";"Permittivity";"Conductivity"];
```

ITU-R P.527, Table 1 specifies the sand percentage, clay percentage, specific gravity, and bulk density for soil mixtures with these soil textual classifications.

```
table(soilType,pSand,pClay,pSilt,sg,bd,'VariableNames',varNames1)
```

```
ans=4×6 table
    Soil Textual Classification    Sand     Clay     Silt     Specific Gravity    Bulk Density
    ___________________________    _____    _____    _____    ________________    ____________

           "Sandy Loam"            51.52    13.42    35.06          2.66             1.6006   
           "Loam"                  41.96     8.53    49.51           2.7             1.5781   
           "Silty Loam"            30.63    13.48    55.89          2.59              1.575   
           "Silty Clay"             5.02    47.38     47.6          2.56             1.4758   
```

The relative permittivity and conductivity for these soil textual classifications are included in this table.

```
table(soilType,Permittivity,Conductivity,'VariableNames',varNames2)
```

```
ans=4×3 table
    Soil Textual Classification    Permittivity    Conductivity
    ___________________________    ____________    ____________

           "Sandy Loam"               15.281            18.2   
           "Loam"                     14.563          16.998   
           "Silty Loam"               13.965          16.011   
           "Silty Clay"               12.861          14.647   
```

Calculate Permittivity and Conductivity of Vegetation

Calculate relative permittivity and conductivity versus frequency for vegetation, varying gravimetric water content and temperature.

Calculate relative permittivity and conductivity for vegetation at specified settings.

```
fc = 10e9; % Frequency in Hz
temp  = 23; % Temperature in °C
gwc = 0.68; % Gravimetric water content
[epsilon_veg,sigma_veg] = ...
    earthSurfacePermittivity('vegetation',fc,temp,gwc)
```

```
epsilon_veg = 20.5757
```

```
sigma_veg = 4.9320
```

Calculate values necessary to plot permittivity and conductivity by using `arrayfun` to apply the `earthSurfacePermittivity` function to the elements of the arrayed inputs.

For a range of temperatures, calculate values to plot permittivity and conductivity versus frequency for vegetation at a 0.68 gravimetric water content.

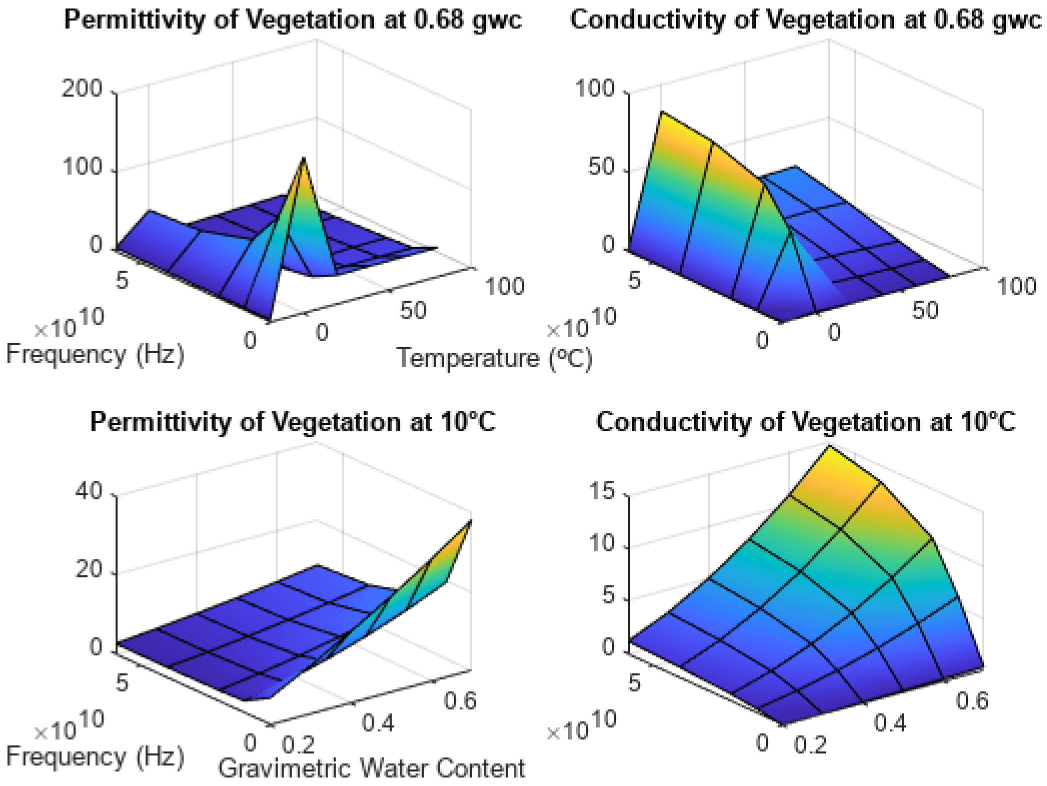
```
fc = repmat([0.1,10,20,40,60]*1e9,6,1);
gwc1 = 0.68;
temp1 = repmat((-20:20:80)',1,5);
[epsilon_veg_gwc,sigma_veg_gwc] = ...
    arrayfun(@(x,y)earthSurfacePermittivity('vegetation',x,y,gwc1),fc,temp1);
```

For a range of gravimetric water contents, calculate values to plot permittivity and conductivity versus frequency for vegetation at 10°C.

```
temp2 = 10;
gwc2 = repmat((0.2:0.1:0.7)',1,5);
[epsilon_veg_tmp, sigma_veg_tmp] = ...
    arrayfun(@(x,z)earthSurfacePermittivity('vegetation',x,temp2,z),fc,gwc2);
```

Display tiled surface plots across specified ranges.

```
figure
tiledlayout(2,2)
nexttile
surf(temp1,fc,epsilon_veg_gwc,'FaceColor','interp')
title('Permittivity of Vegetation at 0.68 gwc')
xlabel('Temperature (℃)')
ylabel('Frequency (Hz)')
nexttile
surf(temp1,fc,sigma_veg_gwc,'FaceColor','interp')
title('Conductivity of Vegetation at 0.68 gwc')
nexttile
surf(gwc2,fc,epsilon_veg_tmp,'FaceColor','interp')
title('Permittivity of Vegetation at 10°C')
xlabel('Gravimetric Water Content')
ylabel('Frequency (Hz)')
nexttile
surf(gwc2,fc,sigma_veg_tmp,'FaceColor','interp')
title('Conductivity of Vegetation at 10°C')
```



Permittivity of Vegetation at 0.68 gwc

Conductivity of Vegetation at 0.68 gwc

Permittivity of Vegetation at 10°C

Conductivity of Vegetation at 10°C

Input Arguments

`fc` — Carrier frequency

scalar in the range (0, 1e12]

Carrier frequency in Hz, specified as a scalar in the range (0, 1e12].

Data Types: `double`

`temp` — Temperature

numeric scalar

Temperature in °C, specified as a numeric scalar. Valid surfaces and associated temperature limits are indicated in this table.

Surface	Valid Temperature (°C)
pure-water	greater than 0
dry-ice	less than or equal to 0
sea-water	greater than or equal to –2
soil	any numeric
vegetation	≥ –20

Note When the surface is wet-ice, the temperature is 0 ℃.

Data Types: `double`

`salinity` — Salinity of sea water
nonnegative scalar

Salinity of the sea water in g/Kg, specified as a nonnegative scalar.

Data Types: `double`

`liqfrac` — Liquid water volume fraction of wet ice
numeric scalar in the range [0, 1]

Liquid water volume fraction of the wet ice, specified as a numeric scalar in the range [0, 1].

Data Types: `double`

`sandpercent` — Sand percentage of soil
numeric scalar in the range [0, 100]

Sand percentage of the soil, specified as a numeric scalar in the range [0, 100]. The sum of `sandpercent` and `claypercent` must be less than or equal to 100.

Data Types: `double`

`claypercent` — Clay percentage of soil
numeric scalar in the range [0, 100]

Clay percentage of the soil, specified as a numeric scalar in the range [0, 100]. The sum of `sandpercent` and `claypercent` must be less than or equal to 100.

Data Types: `double`

`specificgravity` — Specific gravity of soil
nonnegative scalar

Specific gravity of the soil, specified as a nonnegative scalar. The specific gravity is the mass density of the soil sample divided by the mass density of the amount of water in the soil sample.

Data Types: `double`

`vwc` — Volumetric water content of soil
numeric scalar in the range [0, 1]

Volumetric water content of the soil, specified as a numeric scalar in the range [0, 1]. For more information, see “Soil Water Content” on page 6-214.

Data Types: `double`

`bulkdensity` — Bulk density of soil
nonnegative scalar

Bulk density, in g/cm^3, of the soil, specified as a nonnegative scalar. For more information, see “Soil Water Content” on page 6-214.

Data Types: `double`

gwc — Gravimetric water content of vegetation
numeric scalar in the range [0, 0.7]

Gravimetric water content of the vegetation, specified as a numeric scalar in the range [0, 0.7]. For more information, see “Soil Water Content” on page 6-214.

Data Types: `double`

Output Arguments

epsilon — Relative permittivity
nonnegative scalar

Relative permittivity of the earth surface, returned as a nonnegative scalar.

sigma — Conductivity
nonnegative scalar

Conductivity of the earth surface in Siemens per meter (S/m), returned as a nonnegative scalar.

complexepsilon — Complex relative permittivity
complex scalar

Complex relative permittivity of the earth surface, returned as a complex scalar calculated as

`complexepsilon` = `epsilon` – 1*i* `sigma` / ($2\pi fc\varepsilon_0$).

The computation of `complexepsilon` is based on Equations (59) and (9b) in ITU-R P.527 [1]. *f* is the frequency in GHz. *c* is the velocity of light in free space. ε_0 = 8.854187817e-12 Farads/m, where ε_0 is the electric constant for the permittivity of free space.

More About

ITU Terrain Materials

ITU-R P.527 [1] presents methods and equations to calculate complex relative permittivity at carrier frequencies up to 1,000 GHz for these common earth surface materials.

- Water
- Sea Water
- Dry or Wet Ice
- Dry or Wet Soil (combination of sand, clay, and silt)
- Vegetation (above and below freezing)

As described in ITU-R P.527, specific textural classification applies to these mixtures of sand, clay, and silt in soil with associated specific gravities and bulk densities.

Soil Designation Textural Class	Sandy Loam	Loam	Silty Loam	Silty Clay
% Sand	51.52	41.96	30.63	5.02
% Clay	13.42	8.53	13.48	47.38
% Silt	35.06	49.51	55.89	47.60

Soil Designation Textural Class	Sandy Loam	Loam	Silty Loam	Silty Clay
Specific gravity (ρ_s)	2.66	2.70	2.59	2.56
Bulk Density (ρ_b) in g/cm^3	1.6006	1.5781	1.5750	1.4758

Soil Water Content

Soil water content is expressed on a gravimetric or volumetric basis. Gravimetric water content, `gwc`, is the mass of water per mass of dry soil. Volumetric water content, `vwc`, is the volume of liquid water per volume of soil. The bulk density, `bulkdensity`, is the ratio of the dry soil weight to the volume of the soil sample. The relationship between `gwc` and `vwc` is `vwc` = `gwc` □ `bulkdensity`. When bulk density is not specified, the value of `bulkdensity` is computed by using ITU-R P.527, Equation 36:

`bulkdensity` = 1.07256 + 0.078886 ln(*pSand*) + 0.038753 ln(*pClay*) + 0.032732 ln(*pSilt*),

where

- *pSand* = `sandpercent`
- *pClay* = `claypercent`
- *pSilt* = 100 – (`sandpercent` + `claypercent`)

Version History

Introduced in R2020a

References

[1] International Telecommunications Union Radiocommunication Sector. *Electrical characteristics of the surface of the Earth*. Recommendation P.527-5. ITU-R, approved August 14, 2019. https://www.itu.int/rec/R-REC-P.527-5-201908-I/en.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`buildingMaterialPermittivity` | `raytrace` | `raypl` | `propagationModel`

Objects

`comm.Ray`

raypl

Path loss and phase change for RF propagation ray

Syntax

```
[pl,phase] = raypl(ray)
[pl,phase] = raypl(ray,Name,Value)
```

Description

`[pl,phase] = raypl(ray)` returns the path loss `pl` in dB and phase shift `phase` in radians for the RF propagation ray `ray`. The function calculates the path loss and phase shift using free space loss and reflection loss derived from the propagation path, reflection materials, and antenna polarizations.

By default, `raypl` assumes the antennas are unpolarized. You can polarize the antennas by specifying the `TransmitterPolarization` and `ReceiverPolarization` name-value arguments.

For more information about the path loss computations, see “Path Loss Computation” on page 6-223.

`[pl,phase] = raypl(ray,Name,Value)` specifies options using name-value arguments. For example, `"ReflectionMaterials","brick"` specifies the reflection material as brick.

Examples

Reevaluate Path Loss Changing Reflection Materials and Frequency

Change the reflection materials and frequency for a ray and reevaluate the path loss and phase shift.

Launch Site Viewer with buildings in Hong Kong. For more information about the osm file, see [1] on page 6-219. Specify transmitter and receiver sites.

```
viewer = siteviewer("Buildings","hongkong.osm");

tx = txsite("Latitude",22.2789,"Longitude",114.1625, ...
    "AntennaHeight",10,"TransmitterPower",5, ...
    "TransmitterFrequency",28e9);
rx = rxsite("Latitude",22.2799,"Longitude",114.1617, ...
    "AntennaHeight",1);
```

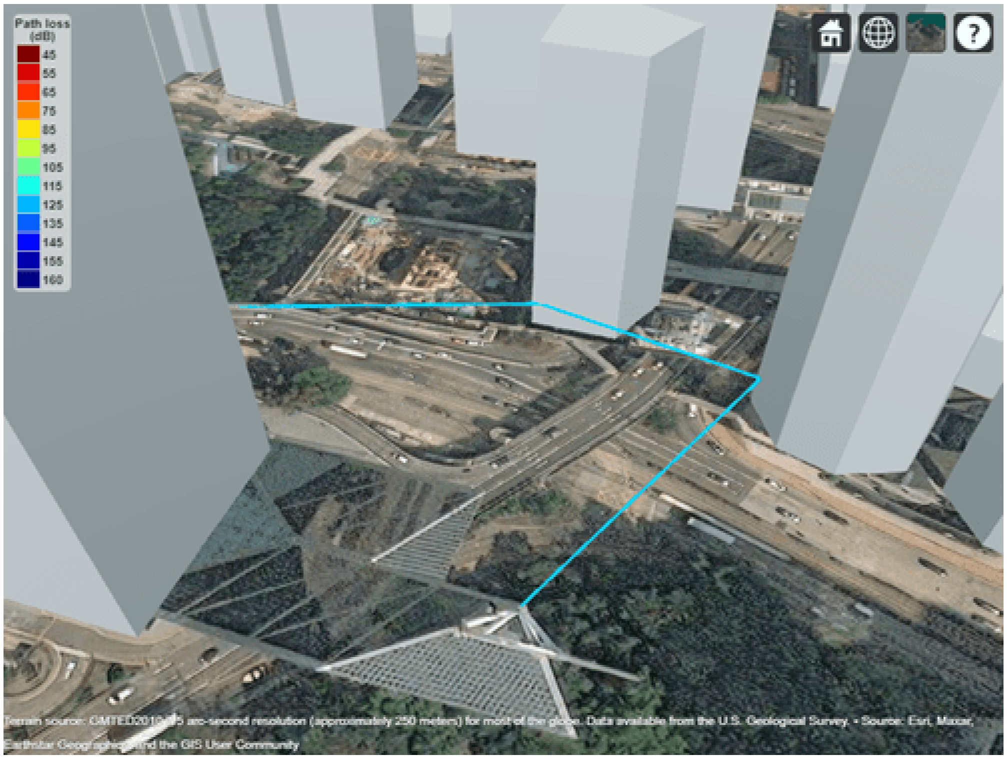
Perform ray tracing between the sites.

```
pm = propagationModel("raytracing", ...
    "Method","image", ...
    "MaxNumReflections",2);
rays = raytrace(tx,rx,pm);
```

Find the first ray with 2-order reflections from the result. Display the ray characteristics. Plot the ray to see the ray reflect off two buildings.

```
ray = rays{1}(find([rays{1}.NumInteractions] == 2,1))
ray = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 121.8178
             PhaseShift: 4.5601

   Read-only properties:
       PropagationDelay: 8.3060e-07
    PropagationDistance: 249.0068
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
plot(ray)
```

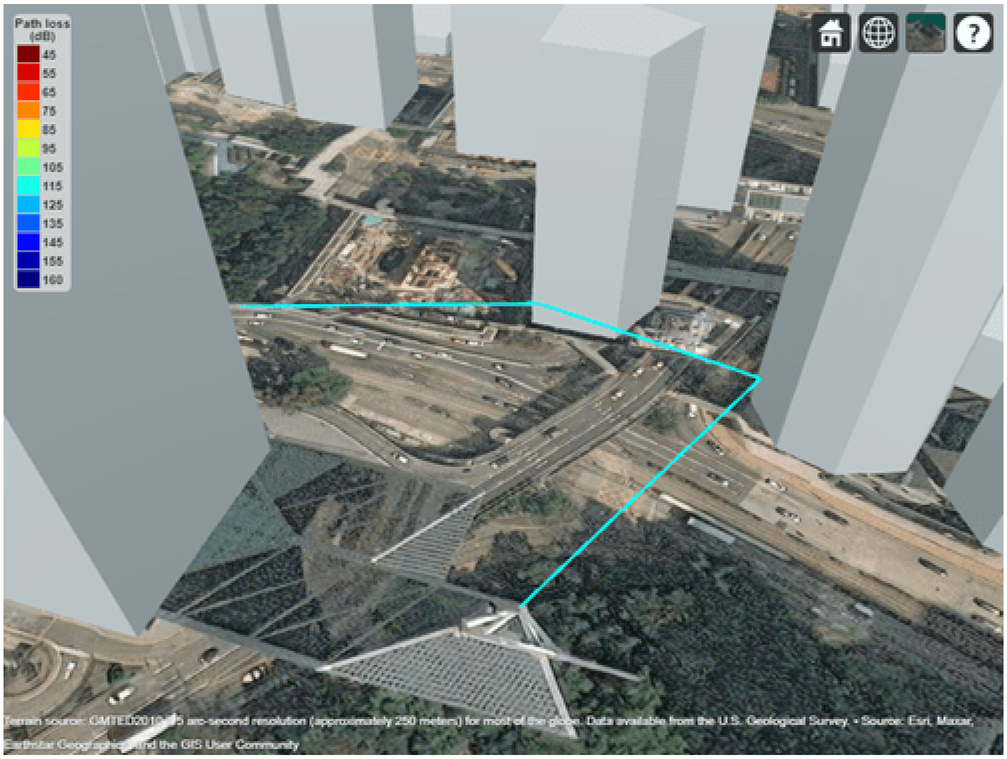

Path loss (dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

By default, all buildings have concrete building material electrical characteristics. Change the material to metal for the second reflection and re-evaluate path loss. Use the `raypl` function to reevaluate the pathloss for the ray. Display the ray path to compare the change in path loss. Replot to show the slight change in color due to the path loss change of the ray.

```
[ray.PathLoss,ray.PhaseShift] = raypl(ray, ...
    "ReflectionMaterials",["concrete","metal"])
```

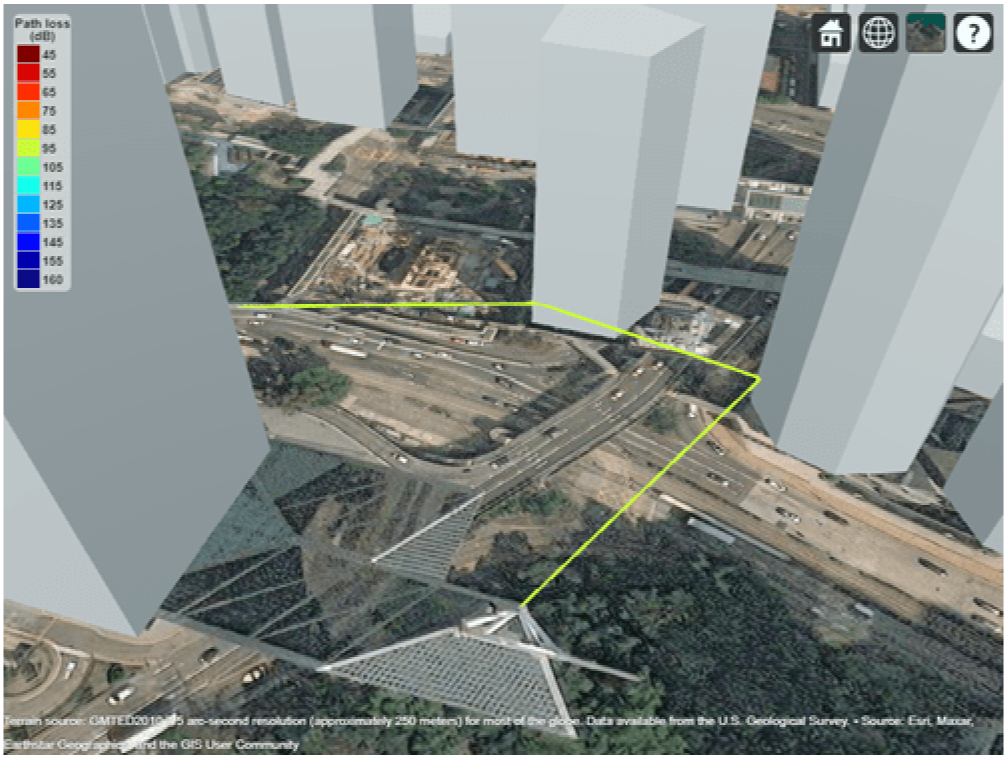
```
ray =
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 117.1214
             PhaseShift: 4.5601

   Read-only properties:
       PropagationDelay: 8.3060e-07
    PropagationDistance: 249.0068
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2


ray =
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 117.1214
             PhaseShift: 4.5601

   Read-only properties:
       PropagationDelay: 8.3060e-07
    PropagationDistance: 249.0068
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
plot(ray)
```


Path loss (dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Change the frequency and reevaluate the path loss and phase shift. Plot the ray again and observe the obvious color change.

```
ray.Frequency = 2e9;
[ray.PathLoss,ray.PhaseShift] = raypl(ray, ...
    "ReflectionMaterials",["concrete","metal"]);
plot(ray)
```


Path loss (dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Input Arguments

ray — RF propagation ray

`comm.Ray` object

RF propagation ray, specified as one `comm.Ray` object. The `PathSpecification` property of the object must be `"Locations"`.

Data Types: `comm.Ray`

Name-Value Pair Arguments

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

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

Example: `raypl(ray,"TransmitterPolarization","H","ReceiverPolarization","H")`, specifies the horizontal polarizations for the transmit and receive antennas for `ray`.

`ReflectionMaterials` — Reflection materials

`"concrete"` (default) | string scalar | 1-by-*NR* string vector | character vector | 1-by-*NR* cell array of character vectors | 2-by-1 numeric vector | 2-by-*NR* numeric matrix

Reflection materials for a non-line-of-sight (NLOS) ray, specified as a string scalar, a 1-by-*NR* string vector, a character vector, a 1-by-*NR* cell array of character vectors, a 2-by-1 numeric vector, or a 2-by-*NR* numeric matrix. *NR* is the number of reflections stored in the `NumReflections` property of `ray`.

When you specify one reflection material, the reflection material applies to all the reflections. When you specify multiple reflection materials, each material applies to the associated reflection point in `ray`.

- To use predefined reflection materials, specify `ReflectionMaterials` as a string scalar, a character vector, a string vector, or a cell array of character vectors. Specify each reflection material as one of these options: `"concrete"`, `"brick"`, `"wood"`, `"glass"`, `"plasterboard"`, `"ceiling-board"`, `"chipboard"`, `"floorboard"`, `"metal"`, `"water"`, `"vegetation"`, `"loam"`, or `"perfect-reflector"`.
- To use custom reflection materials, specify a 2-by-1 numeric vector or a 2-by-*NR* numeric matrix. Each column is of the form `[rp; cv]`, where `rp` is the relative permittivity and `cv` is the conductivity.

For more information, see “ITU Permittivity and Conductivity Values for Common Materials” on page 6-222.

Example: `"ReflectionMaterials",["concrete","water"]`, specifies that a ray with two reflections uses the electrical characteristics of concrete at the first reflection point and water at the second reflection point.

Data Types: `string` | `char` | `double`

`TransmitterPolarization` — Transmit antenna polarization type

`"none"` (default) | `"V"` | `"H"` | `"LHCP"` | `"RHCP"` | normalized 2-by-1 Jones vector

Transmit antenna polarization type, specified as one of these values:

- `"none"` — Unpolarized
- `"V"` — Linearly polarized in the vertical (θ) direction
- `"H"` — Linearly polarized in the horizontal (φ) direction
- `"LHCP"` — Left-hand circular polarized
- `"RHCP"` — Right-hand circular polarized
- A normalized 2-by-1 Jones vector (also called a polarization matrix) of the form `[H;V]`, where `H` is the horizontal component and `V` is the vertical component.

For more information about polarization types and Jones vectors, see “Jones Vector Notation” on page 6-225.

Example: `"TransmitterPolarization","RHCP"` specifies right-hand circular polarization for the transmit antenna.

Data Types: `double` | `char` | `string`

`ReceiverPolarization` — Receive antenna polarization type

`"none"` (default) | `"V"` | `"H"` | `"LHCP"` | `"RHCP"` | normalized 2-by-1 Jones vector

Receive antenna polarization type, specified as one of these values:

- `"none"` — Unpolarized
- `"V"` — Linearly polarized in the vertical (θ) direction
- `"H"` — Linearly polarized in the horizontal (φ) direction
- `"LHCP"` — Left-hand circular polarized
- `"RHCP"` — Right-hand circular polarized
- A normalized 2-by-1 Jones vector (also called a polarization matrix) of the form `[H;V]`, where `H` is the horizontal component and `V` is the vertical component.

For more information about polarization types and Jones vectors, see "Jones Vector Notation" on page 6-225.

Example: `"ReceiverPolarization",[1;0]` specifies horizontal polarization for the receive antenna by using Jones vector notation.

Data Types: `double` | `char` | `string`

`TransmitterAxes` — Orientation of transmit antenna axes

3-by-3 identity matrix (default) | 3-by-3 unitary matrix

Orientation of the transmit antenna axes, specified as a 3-by-3 unitary matrix indicating the rotation from the transmitter local coordinate system (LCS) into the global coordinate system (GCS). When the `CoordinateSystem` property of the `comm.Ray` is set to `"Geographic"`, the GCS orientation is the local East-North-Up (ENU) coordinate system at transmitter. For more information, see "Coordinate System Orientation" on page 6-222.

Example: `"TransmitterAxes",eye(3)`, specifies that the local coordinate system for the transmitter axes is aligned with the global coordinate system. This is the default orientation.

Data Types: `double`

`ReceiverAxes` — Orientation of receive antenna axes

3-by-3 identity matrix (default) | 3-by-3 unitary matrix

Orientation of the receive antenna axes, specified as a 3-by-3 unitary matrix indicating the rotation from the receiver local coordinate system (LCS) into the global coordinate system (GCS). The GCS orientation is the local East-North-Up (ENU) coordinate system at receiver when the .`CoordinateSystem` property of the `comm.Ray` is set to `"Geographic"`. For more information, see "Coordinate System Orientation" on page 6-222.

Example: `"ReceiverAxes",[0 -1 0; 1 0 0; 0 0 1]`, specifies a 90° rotation around the z-axis of the local receiver coordinate system with respect to the global coordinate system.

Data Types: `double`

Output Arguments

`pl` — Path loss

nonnegative scalar

Path loss in dB, returned as a nonnegative scalar.

Data Types: `double`

phase — Phase shift
scalar

Phase shift in radians, returned as a scalar in the range [–π, π] radians. The argument uses the $e^{-i\omega t}$ time convention.

Data Types: `double`

More About

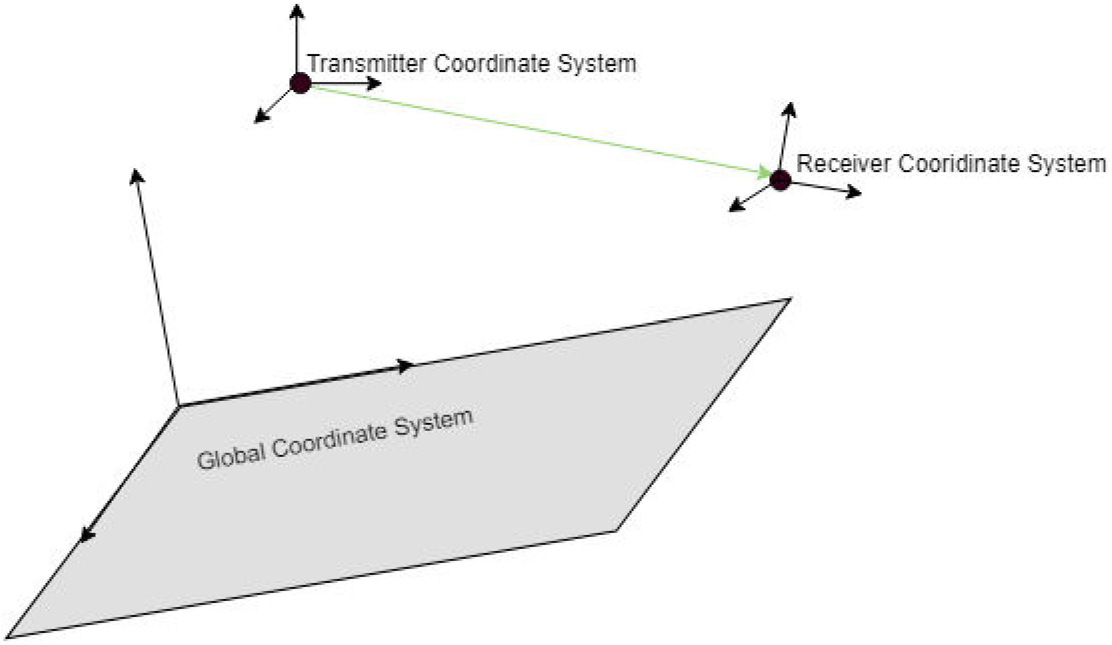
ITU Permittivity and Conductivity Values for Common Materials

ITU-R P.2040-1 [2] (Communications Toolbox) and ITU-R P.527-5 [3] (Communications Toolbox) present methods, equations, and values used to calculate real relative permittivity, conductivity, and complex relative permittivity for the common materials.

- For information about the values computed for building materials specified in ITU-R P.2040-1, see `buildingMaterialPermittivity`.
- For information about the values computed for terrain materials specified in ITU-R P.527-5, see `earthSurfacePermittivity`.

Coordinate System Orientation

This image shows the orientation of the electromagnetic fields in the global coordinate system (GCS) and the local coordinate systems of the transmitter and receiver.

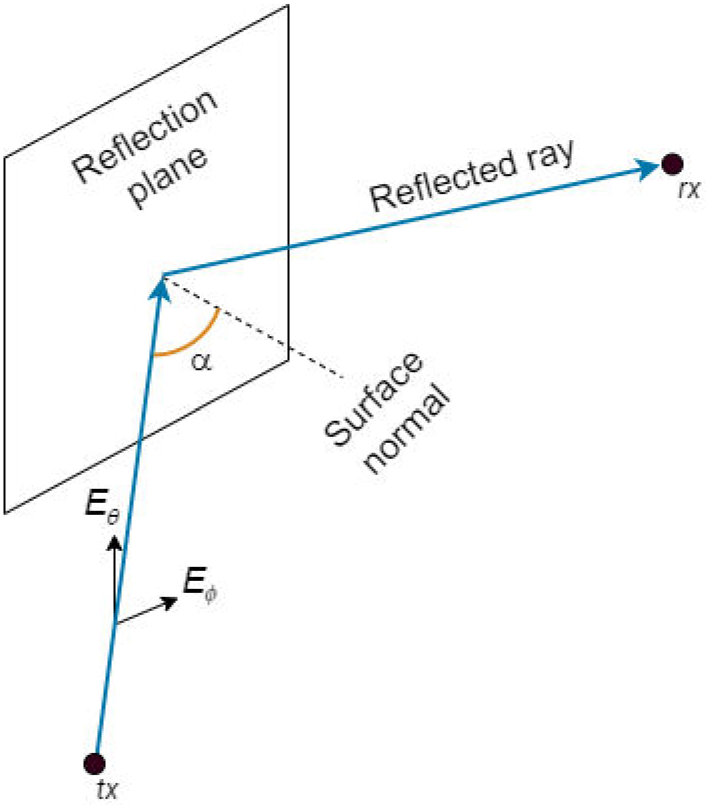


When the `CoordinateSystem` property of the `comm.Ray` is set to `"Geographic"`, the GCS orientation is the local East-North-Up (ENU) coordinate system at observer. The path loss computation accounts for the round-earth differences between ENU coordinates at the transmitter and receiver.

Path Loss Computation

The ray tracing model used by the `raypl` function calculates reflection losses by tracking the horizontal and vertical polarizations of signals through the propagation path. Total power loss is the sum of free space loss and reflection loss.

This image shows a reflection path from a transmitter site *tx* to a receiver site *rx*.



The model determines polarization and reflection loss using these steps.

1 Track the propagation of the ray in 3-D space by calculating the propagation matrix *P*. The matrix is a repeating product, where *i* is the number of reflection points.

$$P = \prod_i P_i$$

For each reflection, calculate P_i by transforming the global coordinates of the incident electromagnetic field into the local coordinates of the reflection plane, multiplying the result by a reflection coefficient matrix, and transforming the coordinates back into the original global coordinate system [1]. The equations for P_i and P_0 are:

$$P_i = [s_{out}\ p_{out}\ k_{out}]_i \begin{bmatrix} R_V(\alpha) & 0 & 0 \\ 0 & R_H(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}_i [s_{in}\ p_{in}\ k_{in}]_i^{-1}$$

$$P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where:

- s, p, and k form a basis for the plane of incidence (the plane created by the incident ray and the surface normal of the reflection plane). s and p are perpendicular and parallel, respectively, to the plane of incidence.
- k_{in} and k_{out} are the directions (in global coordinates) of the incident and exiting rays, respectively.
- s_{in} and s_{out} are the directions (in global coordinates) of the horizontal polarizations for the incident and exiting rays, respectively.
- p_{in} and p_{out} are the directions (in global coordinates) of the vertical polarizations for the incident and exiting rays, respectively.
- R_H and R_V are the Fresnel reflection coefficients for the horizontal and vertical polarizations, respectively. α is the incident angle of the ray and ε_r is the complex relative permittivity of the material.

  $$R_H(\alpha) = \frac{\cos(\alpha) - \sqrt{(\varepsilon_r - \sin^2(\alpha))/\varepsilon_r 2}}{\cos(\alpha) + \sqrt{(\varepsilon_r - \sin^2(\alpha))/\varepsilon_r 2}}$$

  $$R_V(\alpha) = \frac{\cos(\alpha) - \sqrt{\varepsilon_r - \sin^2(\alpha)}}{\cos(\alpha) + \sqrt{\varepsilon_r - \sin^2(\alpha)}}$$

2 Project the propagation matrix P into a 2-by-2 polarization matrix R. The model rotates the coordinate systems for the transmitter and receiver so that they are in global coordinates.

$$R = \begin{bmatrix} H_{in} \cdot H_{rx} & V_{in} \cdot H_{rx} \\ H_{in} \cdot V_{rx} & V_{in} \cdot V_{rx} \end{bmatrix}$$

$$H_{in} = P(V_{tx} \times k_{tx})$$

$$V_{in} = PV_{tx}$$

where:

- H_{rx} and V_{rx} are the directions (in global coordinates) of the horizontal (E_θ) and vertical (E_ϕ) polarizations, respectively, for the receiver.
- H_{in} and V_{in} are the directions (in global coordinates) of the propagated horizontal and vertical polarizations, respectively.
- V_{tx} is the direction (in global coordinates) of the nominal vertical polarization for the ray departing the transmitter.
- k_{tx} is the direction (in global coordinates) of the ray departing the transmitter.

3 Specify the normalized horizontal and vertical polarizations of the electric field at the transmitter and receiver by using the 2-by-1 Jones polarization vectors J_{tx} and J_{rx}, respectively. If either the transmitter or receiver are unpolarized, then the model assumes $J_{tx} = J_{rx} = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

4 Calculate the polarization and reflection loss IL by combining R, J_{tx}, and J_{rx}.

$$IL = -20\log_{10}\left|J_{rx}^{-1}RJ_{tx}\right|$$

Jones Vector Notation

For Jones vector notation, the raypl function describes signal polarization using Jones calculus.

The orthogonal components of Jones vectors are defined for E_θ and E_φ. This table shows the Jones vector corresponding to various antenna polarizations.

Antenna Polarization Type	Corresponding Jones Vector
Linear polarized in the θ direction	$\begin{pmatrix}H\\V\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix}$
Linear polarized in the φ direction	$\begin{pmatrix}H\\V\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}$
Left-hand circular polarized (LHCP)	$\begin{pmatrix}H\\V\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}j\\1\end{pmatrix}$
Right-hand circular polarized (RHCP)	$\begin{pmatrix}H\\V\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}-j\\1\end{pmatrix}$

Version History

Introduced in R2020a

Ray tracing models using SBR method find paths with exact geometric accuracy

Behavior changed in R2022b

When you find propagation paths using the `raytrace` function and a ray tracing model that uses the shooting and bouncing rays (SBR) method, MATLAB corrects the results so that the geometric accuracy of each path is exact, using single-precision floating-point computations. In previous releases, the paths have approximate geometric accuracy.

As a result, when you use rays returned by the `raytrace` function as input to the `raypl` function, the `raypl` function can return different results than in previous releases.

References

[1] Chipman, Russell A., Garam Young, and Wai Sze Tiffany Lam. "Fresnel Equations." In *Polarized Light and Optical Systems*. Optical Sciences and Applications of Light. Boca Raton: Taylor & Francis, CRC Press, 2019.

[2] International Telecommunications Union Radiocommunication Sector. *Effects of building materials and structures on radiowave propagation above about 100MHz.* Recommendation P.2040-1. ITU-R, approved July 29, 2015. https://www.itu.int/rec/R-REC-P.2040-1-201507-I/en.

[3] International Telecommunications Union Radiocommunication Sector. *Attenuation by atmospheric gases*. Recommendation P.676-11. ITU-R, approved September 30, 2016. https://www.itu.int/rec/R-REC-P.676-11-201609-S/en.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

When you specify multiple reflective materials, you must define each value as a character vector (`char` data type) in a cell array.

See Also

Functions

`raytrace` | `buildingMaterialPermittivity` | `earthSurfacePermittivity` | `propagationModel`

Objects

`comm.Ray` | `siteviewer`

location

Coordinates of RF propagation data

Syntax

```
datalocation = location(pd)
[lat,lon] = location(pd)
```

Description

`datalocation = location(pd)` returns the location coordinates of the data points in the propagation data object.

`[lat,lon] = location(pd)` returns the latitude and longitude of the propagation data object

Examples

Transmitter Site Service Areas

Define names and locations of sites around Boston.

```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

Create array of transmitter sites.

```
txs = txsite("Name", names,...
       "Latitude",lats,...
       "Longitude",lons, ...
       "TransmitterFrequency",2.5e9);
```

Compute received power data for each transmitter site.

```
maxr = 20000;
pd1 = coverage(txs(1),"MaxRange",maxr);
pd2 = coverage(txs(2),"MaxRange",maxr);
pd3 = coverage(txs(3),"MaxRange",maxr);
```

Compute rectangle containing locations of all data.

```
locs = [location(pd1); location(pd2); location(pd3)];
[minlatlon, maxlatlon] = bounds(locs);
```

Create grid of locations over rectangle.

```
gridlength = 300;
latv = linspace(minlatlon(1),maxlatlon(1),gridlength);
lonv = linspace(minlatlon(2),maxlatlon(2),gridlength);
[lons,lats] = meshgrid(lonv,latv);
lats = lats(:);
lons = lons(:);
```

Get data for each transmitter at grid locations using interpolation.

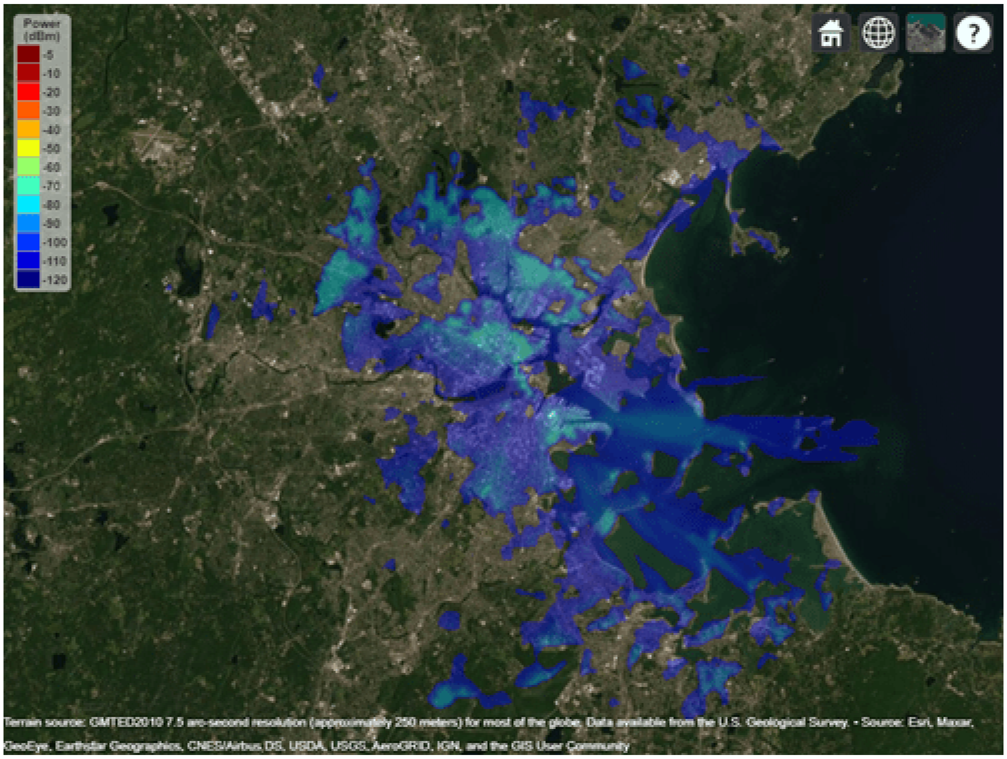
```
v1 = interp(pd1,lats,lons);
v2 = interp(pd2,lats,lons);
v3 = interp(pd3,lats,lons);
```

Create propagation data containing minimum received power values.

```
minReceivedPower = min([v1 v2 v3],[],2,"includenan");
pd = propagationData(lats,lons,"MinReceivedPower",minReceivedPower);
```

Plot minimum received power, which shows the weakest signal received from any transmitter site. The area shown may correspond to the service area of triangulation using the three transmitter sites.

```
sensitivity = -110;
contour(pd,"Levels",sensitivity:-5,"Type","power")
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

pd — Propagation data

propagationData object (default)

Propagation data, specified as a `propagationData` object.

Output Arguments

datalocation — Location coordinates of data points

M-by-2 matrix

Location of antenna site, returned as an *M*-by-2 matrix with each element unit in degrees. *M* is the number of rows in the data table with valid latitude and longitude values. Duplicate locations are not removed.

`lat` — Latitude of data points
M-by-1 vector

Latitude of data points, returned as an *M*-by-`1` vector with each element unit in degrees.

`lon` — Longitude of data points
M-by-1 vector

Longitude of data points, returned as an *M*-by-`1` matrix with each element unit in degrees. The output is wrapped so that the values are in the range `[-180 180]`.

Version History

Introduced in R2020a

See Also

`getDataVariable` | `interp`

plot

Display RF propagation data in Site Viewer

Syntax

```
plot(pd)
plot(___,Name,Value)
```

Description

`plot(pd)` displays propagation data in the current Site Viewer. Each data point is displayed as a circular marker that is colored according to the corresponding value.

`plot(___,Name,Value)` displays the propagation data with additional options specified by name-value pair arguments.

Examples

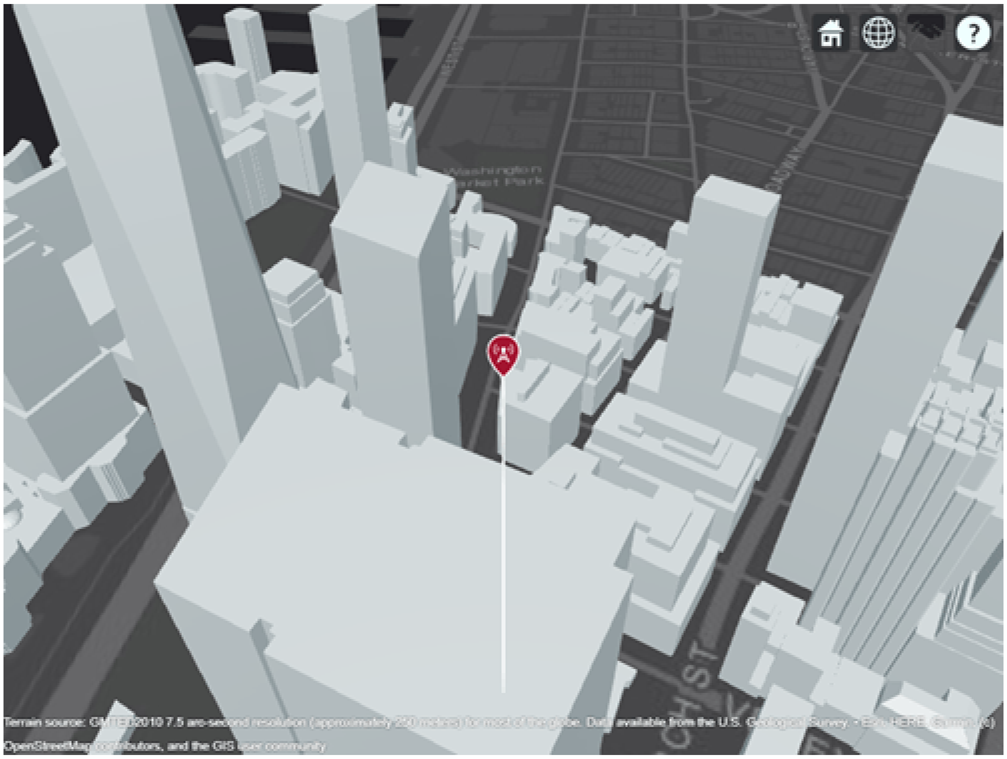
Compute Signal Strength Data in Urban Environment

Launch Site Viewer with basemaps and building files for Manhattan. For more information about the osm file, see [1] on page 6-232.

```
viewer = siteviewer("Basemap","streets_dark",...
        "Buildings","manhattan.osm");
```

Show a transmitter site on a building.

```
tx = txsite("Latitude",40.7107,...
        "Longitude",-74.0114,...
        "AntennaHeight",80);
show(tx)
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Esri, HERE, Garmin, (c) OpenStreetMap contributors, and the GIS user community

Create receiver sites along nearby streets.

```
latitude = [linspace(40.7088, 40.71416, 50), ...
        linspace(40.71416, 40.715505, 25), ...
        linspace(40.715505, 40.7133, 25), ...
        linspace(40.7133, 40.7143, 25)]';
longitude = [linspace(-74.0108, -74.00627, 50), ...
        linspace(-74.00627 ,-74.0092, 25), ...
        linspace(-74.0092, -74.0110, 25), ...
        linspace(-74.0110, -74.0132, 25)]';
rxs = rxsite("Latitude", latitude, "Longitude", longitude);
```

Compute signal strength at each receiver location.

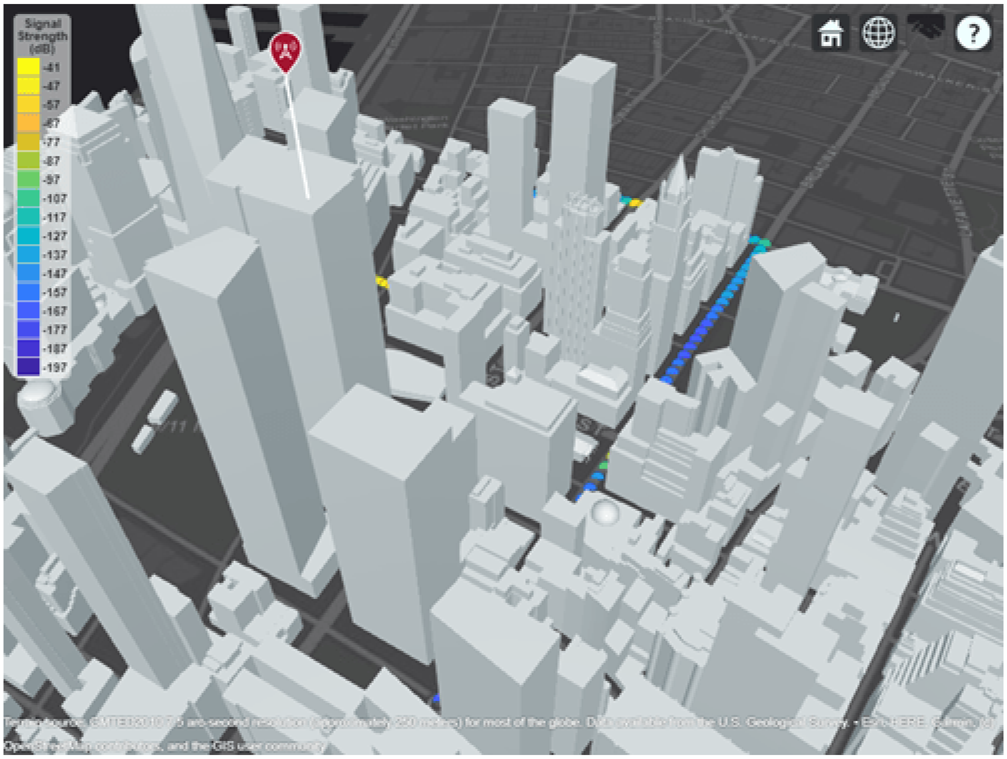
```
signalStrength = sigstrength(rxs, tx)';
```

Create a `propagationData` object to hold computed signal strength data.

```
tbl = table(latitude, longitude, signalStrength);
pd = propagationData(tbl);
```

Plot the signal strength data on a map as colored points.

```
legendTitle = "Signal" + newline + "Strength" + newline + "(dB)";
plot(pd, "LegendTitle", legendTitle, "Colormap", parula);
```



Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Input Arguments

pd — Propagation data

`propagationData` object (default)

Propagation data, specified as a `propagationData` object.

Name-Value Pair Arguments

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

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

Example: `'Type','power'`

DataVariableName — Data variable to plot

`pd.DataVariableName` (default) | character vector | string scalar

Data variable to plot, specified as the comma-separated pair consisting of `'DataVariableName'` and a character vector or a string scalar corresponding to a variable name in the data table used to create the propagation data container object pd. The default value is dynamic and corresponds to the DataVariableName property of the propagationData object.

Data Types: char | string

Type — Type of value to plot

'custom' (default) | 'power' | 'efield' | 'sinr' | 'pathloss'

Type of value to plot, specified as the comma-separated pair consisting of 'Type' and one of the values in the Type column:

Type	ColorLimits	LegendTitle
'custom'	[min(Data) max(Data)]	''
'power'	[-120 -5]	'Power (dBm)'
'efield'	[20 135]	'E-field (dBuV/m)'
'sinr'	[-5 20]	'SINR (dB)'
'pathloss'	[45 160]	'Path loss (dB)'

The default value for Levels is a linearly spaced vector bounded by ColorLimits.

Data Types: char | string

Levels — Data value levels to plot

numeric vector

Data value levels to plot, specified as the comma-separated pair consisting of 'Levels' and a numeric vector. The propagation data is binned according to Levels. The data in each bin is color coded according to the corresponding level. The colors are selected using Colors if specified, or else Colormap and ColorLimits. Data points with values below the minimum level are not included in the plot.

The default value for Levels is a linearly spaced vector bounded by ColorLimits.

Data Types: double

Colors — Colors of data points

M-by-3 array of RGB | array of strings | cell array of character vectors

Colors of the data points, specified as the comma-separated pair consisting of 'Colors' and an *M*-by-3 array of RGB (red, blue, green) or an array of strings, or a cell array of character vectors. Colors are assigned element-wise to values in Levels for coloring the corresponding points. Colors cannot be used with Colormap and ColorLimits.

Data Types: double | char | string

Colormap — Color map for coloring points

'jet(256)' (default) | predefined colormap name | *M*-by-3 array of RGB triplets

Colormap for the coloring points, specified as the comma-separated pair consisting of 'Colormap' and a predefined colormap name or an *M*-by-3 array of RGB (red, blue, green) triplets that define *M* individual colors. Colormap cannot be used with Colors.

Data Types: double | char | string

`ColorLimits` — Color limits for color map

two-element vector

Color limits for the colormap, specified as the comma-separated pair consisting of `'ColorLimits'` and a two-element vector of the form [min max]. The color limits indicate the data level values that map to the first and last colors in the colormap. `ColorLimits` cannot be used with `Colors`.

Data Types: `double`

`MarkerSize` — Size of data markers

`10` (default) | positive numeric scalar

Size of data markers plotted on the map, specified as the comma-separated pair consisting of `'MarkerSize'` and a positive numeric scalar in pixels.

Data Types: `double`

`ShowLegend` — Show color legend on map

`true` (default) | `false`

Show color legend on map, specified as the comma-separated pair consisting of `'ShowLegend'` and `true` or `false`.

Data Types: `logical`

`LegendTitle` — Title of color legend

character vector | string scalar

Title of color legend, specified as the comma-separated pair consisting of `'LegendTitle'` and a character vector or a string scalar.

Data Types: `string` | `char`

`Map` — Map for surface data

`siteviewer` object

Map for surface data, specified as the comma-separated pair consisting of `'Map'` and a `siteviewer` object.[8] The default value is the current Site Viewer or a new Site Viewer, if none is open.

Data Types: `char` | `string`

Version History

Introduced in R2020a

See Also

`contour` | `interp`

[8] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

getDataVariable

Get data variable values

Syntax

```
datavariable = getDataVariable(pd)
[datavariable,lat,lon] = getDataVariable(pd)
[___] = getDataVariable(pd,varname)
```

Description

`datavariable = getDataVariable(pd)` returns the values of the data points in the propagation data object. The data is processed such that the missing values are removed and duplicate location data are replaced with mean values.

`[datavariable,lat,lon] = getDataVariable(pd)` returns the location coordinates of the data points in the propagation data object.

`[___] = getDataVariable(pd,varname)` returns the values of the data points corresponding to the `varname` variable.

Examples

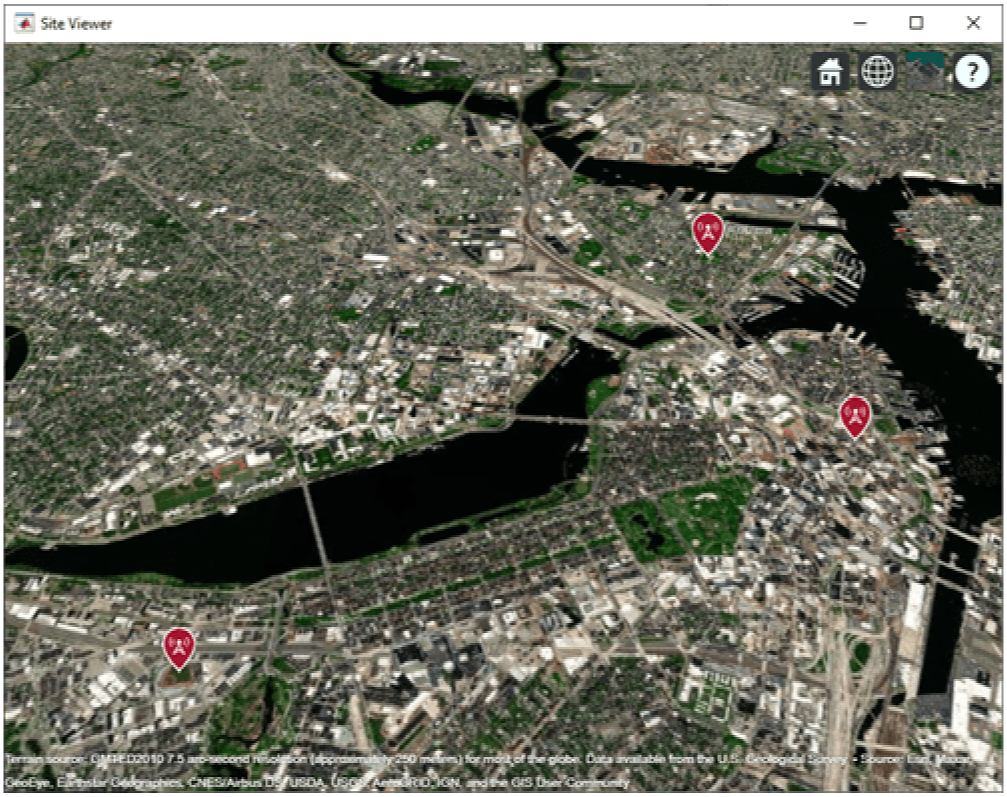
Capacity Map Using SINR Data

Define names and locations of sites around Boston.

```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

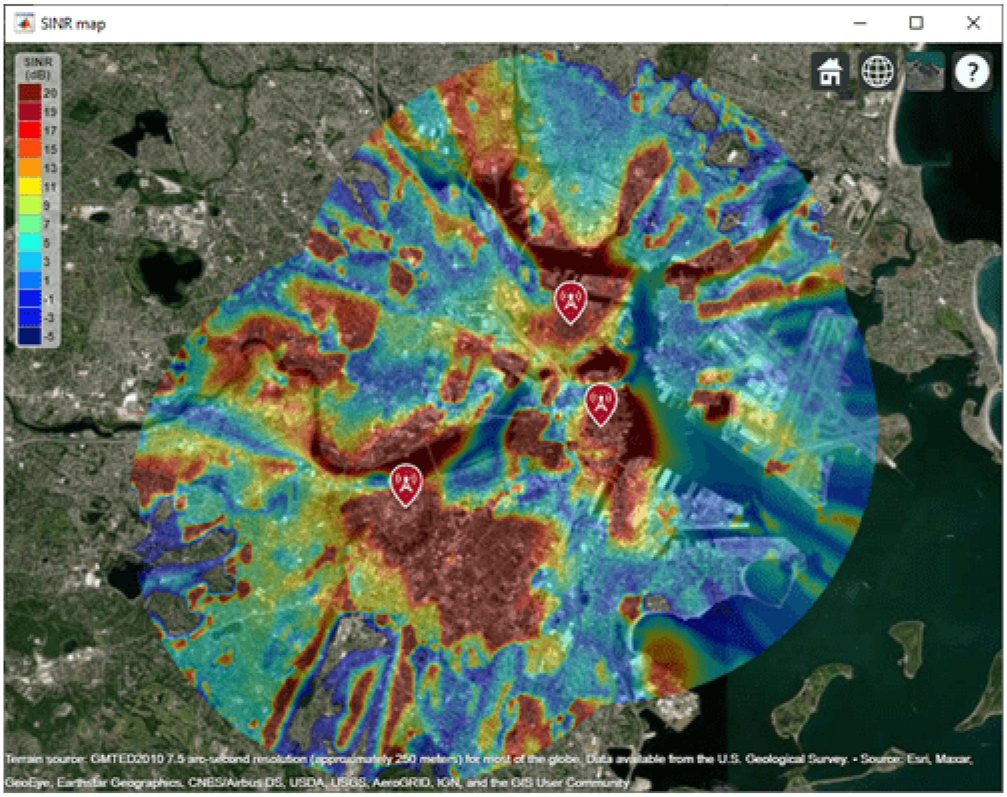
Create an array of transmitter sites.

```
txs = txsite("Name",names,...
       "Latitude",lats,...
       "Longitude",lons, ...
       "TransmitterFrequency",2.5e9);
show(txs)
```


Site Viewer
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create a signal-to-interference-plus-noise-ratio (SINR) map, where signal source for each location is selected as the transmitter site with the strongest signal.

```
sv1 = siteviewer("Name","SINR map");
sinr(txs,"MaxRange",5000)
```


SINR map
SINR (dB)
20
19
17
15
13
11
9
7
5
3
1
-1
-3
-5
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Return SINR propagation data.

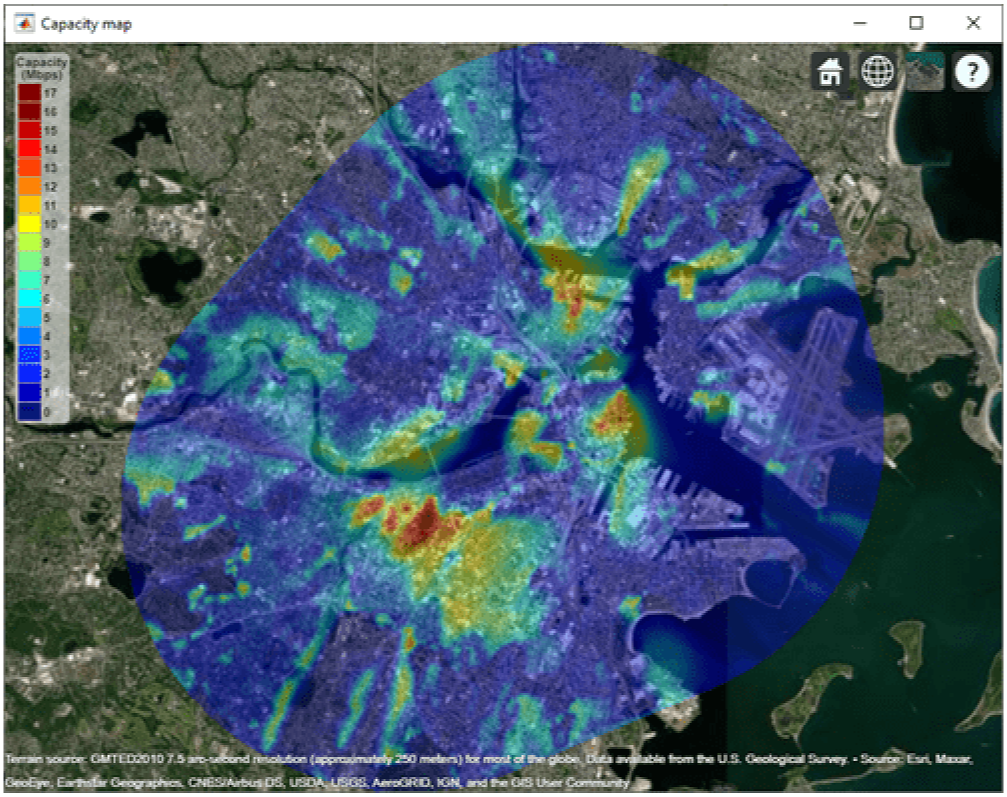
```
pd = sinr(txs,"MaxRange",5000);
[sinrDb,lats,lons] = getDataVariable(pd,"SINR");
```

Compute capacity using the Shannon-Hartley theorem.

```
bw = 1e6; % Bandwidth is 1 MHz
sinrRatio = 10.^(sinrDb./10); % Convert from dB to power ratio
capacity = bw*log2(1+sinrRatio)/1e6; % Unit: Mbps
```

Create new propagation data for the capacity map and display the contour plot.

```
pdCapacity = propagationData(lats,lons,"Capacity",capacity);
sv2 = siteviewer("Name","Capacity map");
legendTitle = "Capacity" + newline + "(Mbps)";
contour(pdCapacity,"LegendTitle",legendTitle);
```


Capacity map
Capacity (Mbps)
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

pd — Propagation data

`propagationData` object (default)

Propagation data, specified as a `propagationData` object.

`varname` — Variable name in data table

character vector | string scalar

Variable name in the data table, specified as a character vector or a string scalar. This variable name must correspond to a variable with numeric data other than the latitude or longitude data.

Output Arguments

`datavariable` — Values of data points

column vector

Values of data points in the propagation data object, returned as a column vector.

`lat` — Latitude of data points

M-by-1 vector

Latitude of data points, returned as an *M*-by-1 vector with each element unit in degrees.

`lon` — Longitude of data points
M-by-1 vector

Longitude of data points, returned as an *M*-by-`1` matrix with each element unit in degrees. The output is wrapped so that the values are in the range `[-180 180]`.

Version History

Introduced in R2020a

See Also

`location` | `interp`

interp

Interpolate RF propagation data

Syntax

```
interpvalue = interp(pd,lat,lon)
interpvalue = interp(pd,Name,Value)
```

Description

`interpvalue = interp(pd,lat,lon)` returns interpolated values from the propagation data for each query point specified in latitude and longitude vectors. The interpolation is performed using a scattered data interpolation method. Values corresponding to query points outside the data region are assigned a `NaN`.

`interpvalue = interp(pd,Name,Value)` returns interpolated values with additional options specified by name-value pair arguments.

Examples

Transmitter Site Service Areas

Define names and locations of sites around Boston.

```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

Create array of transmitter sites.

```
txs = txsite("Name", names,...
       "Latitude",lats,...
       "Longitude",lons, ...
       "TransmitterFrequency",2.5e9);
```

Compute received power data for each transmitter site.

```
maxr = 20000;
pd1 = coverage(txs(1),"MaxRange",maxr);
pd2 = coverage(txs(2),"MaxRange",maxr);
pd3 = coverage(txs(3),"MaxRange",maxr);
```

Compute rectangle containing locations of all data.

```
locs = [location(pd1); location(pd2); location(pd3)];
[minlatlon, maxlatlon] = bounds(locs);
```

Create grid of locations over rectangle.

```
gridlength = 300;
latv = linspace(minlatlon(1),maxlatlon(1),gridlength);
```

```
lonv = linspace(minlatlon(2),maxlatlon(2),gridlength);
[lons,lats] = meshgrid(lonv,latv);
lats = lats(:);
lons = lons(:);
```

Get data for each transmitter at grid locations using interpolation.

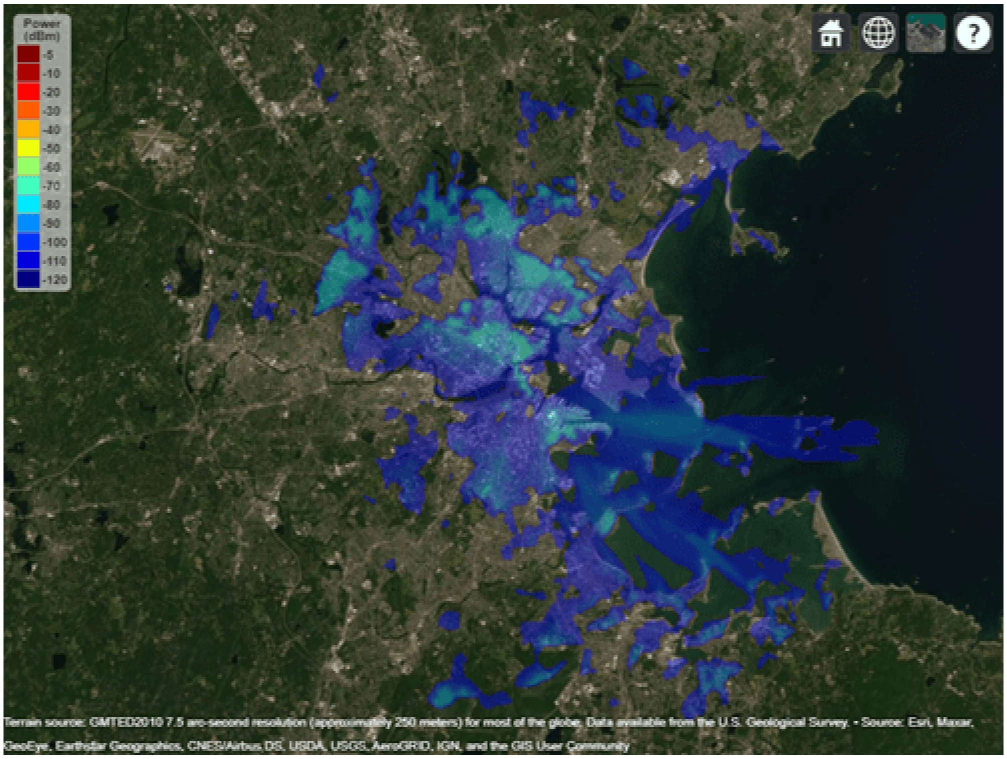
```
v1 = interp(pd1,lats,lons);
v2 = interp(pd2,lats,lons);
v3 = interp(pd3,lats,lons);
```

Create propagation data containing minimum received power values.

```
minReceivedPower = min([v1 v2 v3],[],2,"includenan");
pd = propagationData(lats,lons,"MinReceivedPower",minReceivedPower);
```

Plot minimum received power, which shows the weakest signal received from any transmitter site. The area shown may correspond to the service area of triangulation using the three transmitter sites.

```
sensitivity = -110;
contour(pd,"Levels",sensitivity:-5,"Type","power")
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

pd — Propagation data

propagationData object (default)

Propagation data, specified as a `propagationData` object.

lat — Latitude coordinate values
vector

Latitude coordinate values, specified as a vector in decimal degrees with reference to Earth's ellipsoid. model WGS-84. The latitude coordinates must be in the range `[-90 90]`.

lon — Longitude coordinate values
vector

Longitude coordinate values, specified as a vector in decimal degrees with reference to Earth's ellipsoid. model WGS-84.

Name-Value Pair Arguments

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

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

Example: `'Method','linear'`

DataVariableName — Data variable to interpolate
character vector | string scalar

Data variable to interpolate, specified as the comma-separated pair consisting of `'DataVariableName'` and a character vector or string scalar corresponding to a variable name in the data table used to create the `propagationData` container object. The default value is the `DataVariableName` property in the `propagationData`.

Data Types: `char` | `string`

Method — Method used to interpolate data
`'natural'` (default) | `'nearest'` | `'linear'`

Method used to interpolate data, specified as the comma separated-pair consisting `'Method'` and one of the following:

- `'natural'` - Natural neighbor interpolation
- `'linear'` - Linear interpolation
- `'nearest'` - Nearest neighbor interpolation

Data Types: `char` | `string`

Output Arguments

interpvalue — Interpolated values from propagation data
numeric vector

Interpolated values from the propagation data for each query point specified in latitude and longitude vectors, returned as a numeric vector.

Version History

Introduced in R2020a

See Also

`plot` | `contour` | `location` | `getDataVariable`

contour

Display contour map of RF propagation data in Site Viewer

Syntax

```
contour(pd)
contour(___,Name,Value)
```

Description

`contour(pd)` creates a filled contour plot in the current Site Viewer. Contours are colored according to data values of corresponding locations.

`contour(___,Name,Value)` creates a filled contour map with additional options specified by name-value pair arguments.

Examples

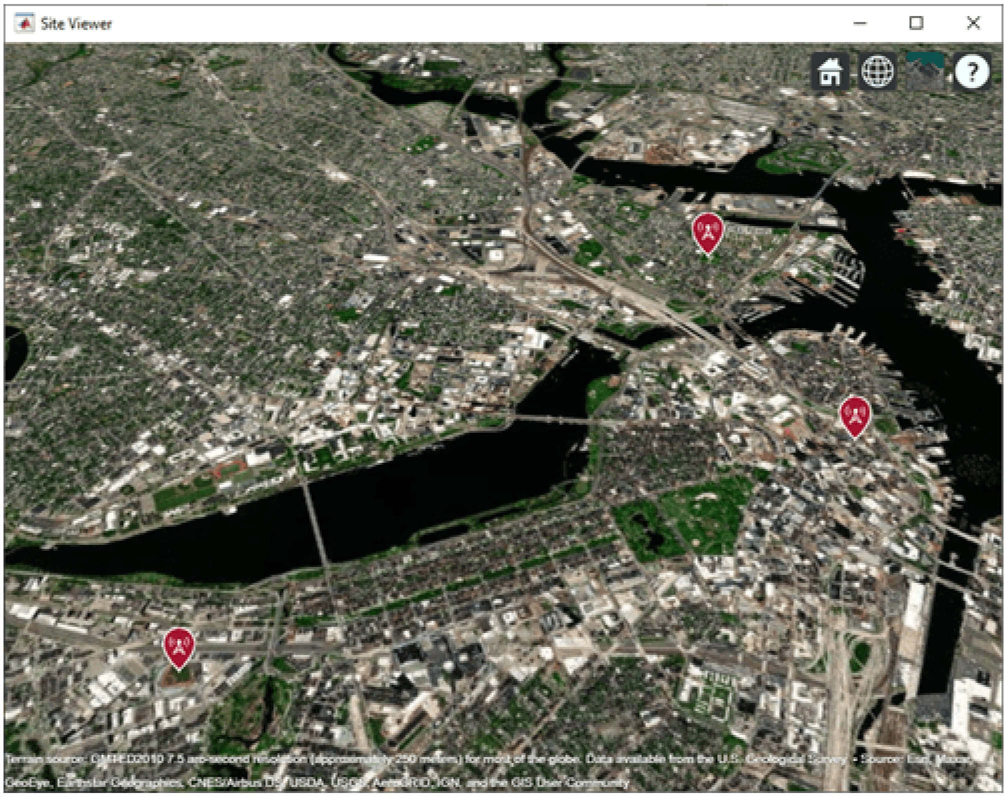
Capacity Map Using SINR Data

Define names and locations of sites around Boston.

```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

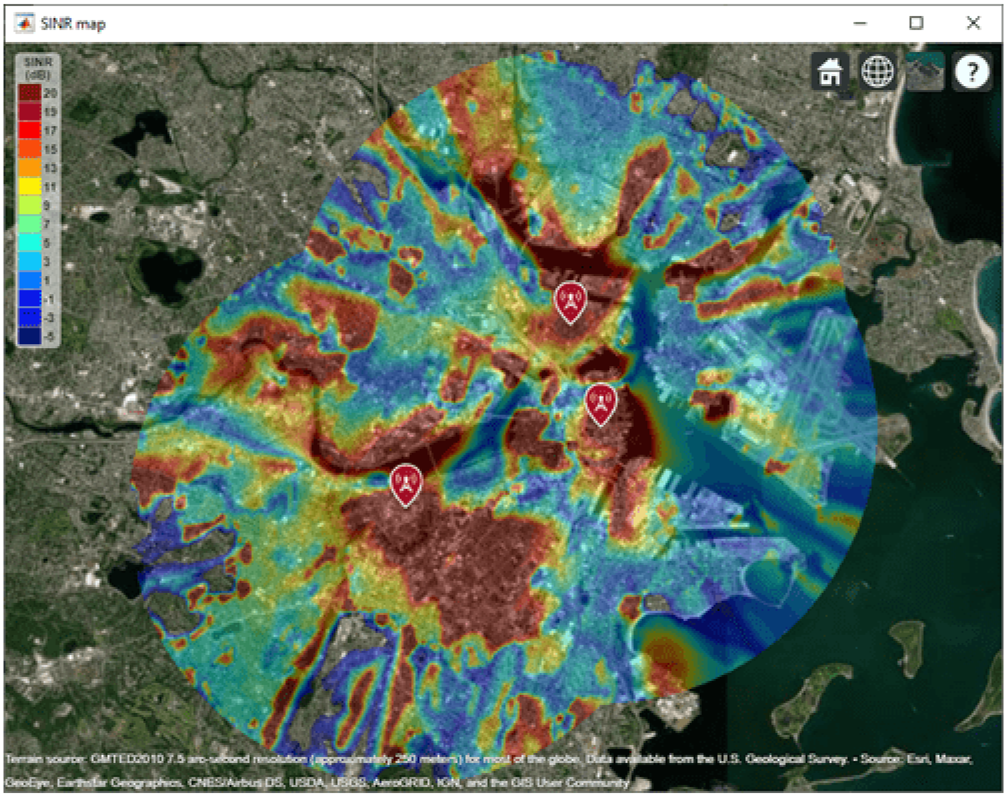
Create an array of transmitter sites.

```
txs = txsite("Name",names,...
       "Latitude",lats,...
       "Longitude",lons, ...
       "TransmitterFrequency",2.5e9);
show(txs)
```


Site Viewer
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create a signal-to-interference-plus-noise-ratio (SINR) map, where signal source for each location is selected as the transmitter site with the strongest signal.

```
sv1 = siteviewer("Name","SINR map");
sinr(txs,"MaxRange",5000)
```


SINR map
SINR
(dB)
20
19
17
15
13
11
9
7
5
3
1
-1
-3
-5
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Return SINR propagation data.

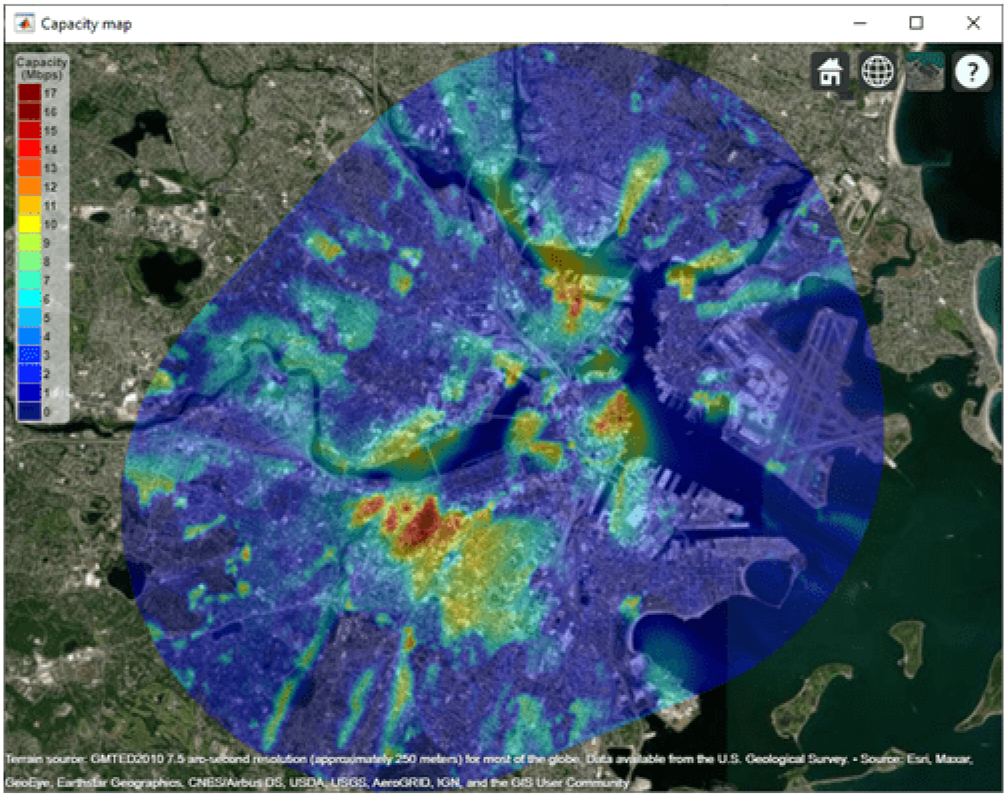
```
pd = sinr(txs,"MaxRange",5000);
[sinrDb,lats,lons] = getDataVariable(pd,"SINR");
```

Compute capacity using the Shannon-Hartley theorem.

```
bw = 1e6; % Bandwidth is 1 MHz
sinrRatio = 10.^(sinrDb./10); % Convert from dB to power ratio
capacity = bw*log2(1+sinrRatio)/1e6; % Unit: Mbps
```

Create new propagation data for the capacity map and display the contour plot.

```
pdCapacity = propagationData(lats,lons,"Capacity",capacity);
sv2 = siteviewer("Name","Capacity map");
legendTitle = "Capacity" + newline + "(Mbps)";
contour(pdCapacity,"LegendTitle",legendTitle);
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

pd — Propagation data

propagationData object (default)

Propagation data, specified as a propagationData object.

Name-Value Pair Arguments

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

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

Example: 'Type','power'

DataVariableName — Data variable to contour map

DataVariableName (default) | character vector | string scalar

Data variable to contour map, specified as the comma-separated pair consisting of 'DataVariableName' and a character vector or a string scalar corresponding to a variable name in the data table used to create the propagation data container object pd.

Data Types: char | string

Type — Type of value to plot

'custom' (default) | 'power' | 'efield' | 'sinr' | 'pathloss'

Type of value to plot, specified as the comma-separated pair consisting of `'Type'` and one of the values in the `'Type'` column:

Type	ColorLimits	LegendTitle
'custom'	[min(Data) max(Data)]	''
'power'	[-120 -5]	'Power (dBm)'
'efield'	[20 135]	'E-field (dBuV/m)'
'sinr'	[-5 20]	'SINR (dB)'
'pathloss'	[45 160]	'Path loss (dB)'

The default value for `Levels` is a linearly spaced vector bounded by `ColorLimits`.

Data Types: `char` | `string`

`Levels` — Data value levels to plot

numeric vector

Data value levels to plot, specified as the comma-separated pair consisting of `'Levels'` and numeric vector. Each level is displayed as a different colored, filled contour on the map. The colors are selected using `Colors` if specified, or else `Colormap` and `ColorLimits`. Data points with values below the minimum level are not included in the plot.

The default value for `Levels` is a linearly spaced vector bounded by `ColorLimits`.

Data Types: `double`

`Colors` — Colors of data points

M-by-3 array of RGB | array of strings | cell array of character vectors

Colors of the filled contours, specified as the comma-separated pair consisting of `'Colors'` and an M-by-3 array of RGB (red, blue, green) or an array of strings, or a cell array of character vectors. Colors are assigned element-wise to values in `Levels` for coloring the corresponding points. Colors cannot be used with `Colormap` and `ColorLimits`.

Data Types: `double` | `char` | `string`

`Colormap` — Color map for coloring points

`'jet(256)'` (default) | predefined colormap name | *M*-by-3 array of RGB triplets

Colormap for the coloring points, specified as the comma-separated pair consisting of `'Colormap'` and a predefined colormap name or an *M*-by-3 array of RGB (red, blue, green) triplets that define *M* individual colors. `Colormap` cannot be used with `Colors`.

Data Types: `double` | `char` | `string`

`ColorLimits` — Color limits for color map

two-element vector

Color limits for the colormap, specified as the comma-separated pair consisting of `'ColorLimits'` and a two-element vector of the form [min max]. The color limits indicate the data level values that map to the first and last colors in the colormap. `ColorLimits` cannot be used with `Colors`.

Data Types: `double`

Transparency — Transparency of contour map

0.4 (default) | numeric scalar in the range of [0,1]

Transparency of the contour plot, specified as a numeric scalar in the range [0,1], where 0 is completely transparent and 1 is completely opaque.

Data Types: double

ShowLegend — Show color legend on map

true (default) | false

Show color legend on map, specified as the comma-separated pair consisting of 'ShowLegend' and true or false.

Data Types: logical

LegendTitle — Title of color legend

character vector | string scalar

Title of color legend, specified as the comma-separated pair consisting of 'LegendTitle' and a character vector or a string scalar.

Data Types: string | char

Map — Map for surface data

siteviewer object

Map for surface data, specified as the comma-separated pair consisting of 'Map' and a siteviewer object.[9] The default value is the current Site Viewer or a new Site Viewer, if none is open.

Data Types: char | string

Version History

Introduced in R2020a

See Also

plot | interp | getDataVariable

[9] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

add

Package: `rfprop`

Add propagation models

Syntax

```
pmc = add(propmodel1,propmodel2)
```

Description

`pmc = add(propmodel1,propmodel2)` adds propagation model objects `propmodel1` and `propmodel2` and returns a composite propagation model object `pmc` which contains `propmodel1` and `propmodel2`.

Note

- The syntax `propmodel1+propmodel2` can be used in place of add.
- A composite propagation model cannot contain more than one propagation model object of the same class.
- A composite propagation model cannot contain more than one propagation model object which includes effects of free-space loss.

Examples

Signal Strength Over Terrain Using Composite Propagation Model

Specify the transmitter and the receiver sites.

```
tx = txsite("Name","Fenway Park", ...
      "Latitude",42.3467, ...
        "Longitude",-71.0972, ...
        "TransmitterFrequency",6e9);
rx = rxsite("Name","Bunker Hill Monument", ...
        "Latitude",42.3763, ...
        "Longitude",-71.0611);
```

Calculate signal strength using default Longley-Rice model.

```
 ss1 = sigstrength(rx,tx)
```

```
ss1 = -80.9353
```

Create composite propagation model with Longley-Rice and specific atmospheric propagation models.

```
pm = propagationModel("longley-rice") + ...
       propagationModel("gas") + propagationModel("rain");
```

Calculate signal strength using composite propagation model.

```
ss2 = sigstrength(rx,tx,pm)
```

```
ss2 = -81.2259
```

Input Arguments

propmodel1 — Propagation model

character vector | string

Propagation model, specified as a character vector or string. You can also use the `propagationModel` function to define this input.

Data Types: `char` | `string`

propmodel2 — Propagation model

character vector | string

Propagation model, specified as a character vector or string. You can also use the `propagationModel` function to define this input.

Data Types: `char` | `string`

Output Arguments

pmc — Composite propagation model

composite `propagationModel` function object

Composite propagation model, composite `propagationModel` function object

The path loss computed by `pmc` is the sum of path losses computed by `propmodel1` and `propmodel2`. If either `propmodel1` or `propmodel2` is a ray tracing model, then `pmc` is also a ray tracing model where path losses from rain, gas, or fog models in the composite are added to the path loss computed for each propagation path.

Version History

Introduced in R2020a

See Also

`propagationModel` | `range`

comm.Ray

Create RF propagation ray

Description

The `comm.Ray` object is a container object for the properties of a propagation ray. The object contains the geometric and electromagnetic information of a radio wave propagating from one point to another point in the space.

Creation

Typically you create `comm.Ray` objects by using the `raytrace` function.

Syntax

```
ray = comm.Ray
ray = comm.Ray(Name,Value)
```

Description

`ray = comm.Ray` creates a ray object that initializes properties for a propagation ray.

`ray = comm.Ray(Name,Value)` sets properties using one or more name-value pair arguments. Enclose each property name in quotes. For example, `comm.Ray('CoordinateSystem','Geographic','TransmitterLocation',[40.730610,-73.935242,0])` specifies the geographic coordinate system and a transmitter located in New York City.

Properties

`PathSpecification` — Propagation path specification method
`'Locations'` (default) | `'Delay and angles'`

Propagation path specification method, specified as one of these values.

- `'Locations'` — The ray object path between waypoints are conveyed as (*x*, *y*, *z*) coordinate points by the `TransmitterLocation`, `ReceiverLocation`, and, if applicable, `ReflectorLocations` properties .
- `'Delay and angles'` — The ray object path between waypoints are conveyed by the `PropagationDelay`, `AngleOfDeparture`, and `AngleOfArrival` properties.

Data Types: `char` | `string`

`CoordinateSystem` — Coordinate system
`'Cartesian'` (default) | `'Geographic'`

Coordinate system, specified as `'Cartesian'` or `'Geographic'`. When you set the `CoordinateSystem` property to `'Geographic'`, the coordinate system is defined relative to the

WGS-84 Earth ellipsoid model and the object defines angles relative to the local East-North-Up (ENU) coordinate system at the transmitter and receiver.

Dependencies

To enable this property, set the `PathSpecification` property to `'Locations'`.

Data Types: `char` | `string`

`SystemScale` — Cartesian coordinate system scale

`1` (default) | positive scalar

Cartesian coordinate system scale in meters, specified as a positive scalar.

Dependencies

To enable this property, set the `PathSpecification` property to `'Locations'` and the `CoordinateSystem` property to `'Cartesian'`.

Data Types: `double`

`TransmitterLocation` — Transmitter location

`[0;0;0]` (default) | three-element numeric column vector

Transmitter location, specified as a three-element numeric column vector of coordinates in one of these forms.

- [*x*; *y*; *z*] — This form applies when you set the `CoordinateSystem` property to `'Cartesian'`. The object does not perform range validation for *x*, *y*, and *z*.
- [*latitude*; *longitude*; *height*] — This form applies when you set the `CoordinateSystem` property to `'Geographic'`. *latitude* must be in the range [–90, 90]. The object does not perform range validation for *longitude* and *height*. *height* is referenced to the ellipsoid defined by the World Geodetic System of 1984 (WGS84).

Dependencies

To enable this property, set the `PathSpecification` property to `'Locations'`.

Data Types: `double`

`ReceiverLocation` — Receiver location

`[10;10;10]` (default) | three-element numeric column vector

Receiver location, specified as a three-element numeric column vector of coordinates in one of these forms.

- [*x*; *y*; *z*] — This form applies when you set the `CoordinateSystem` property to `'Cartesian'`. The object does not perform range validation for *x*, *y*, and *z*.
- [*latitude*; *longitude*; *height*] — This form applies when you set the `CoordinateSystem` property to `'Geographic'`. *latitude* must be in the range [–90, 90]. The object does not perform range validation for *longitude* and *height*. *height* is referenced to the ellipsoid defined by the World Geodetic System of 1984 (WGS84).

Dependencies

To enable this property, set the `PathSpecification` property to `'Locations'`.

Data Types: `double`

LineOfSight — Line of sight

true or 1 (default) | false or 0

Line of sight, specified as a logical value of 1 (true) or 0 (false) to indicate whether the ray is a line-of-sight ray.

Dependencies

To enable this property, set the PathSpecification property to 'Locations'.

Data Types: logical

Interactions — Ray-surface interactions

1-by-N_I structure

Ray-surface interactions along the propagation path, specified as a 1-by-N_I structure containing these fields. N_I is the number of interactions.

Type — Type of ray-surface interaction

'Reflection' (default) | 'Diffraction'

Type of ray-surface interaction, specified as 'Reflection' or 'Diffraction'.

Data Types: char | string

Location — Location

[10;10;0] (default) | 3-by-1 numeric vector

Location, specified as a 3-by-1 numeric vector containing the coordinates of one interaction point on the ray.

- When the CoordinateSystem property is set to 'Cartesian', the form is [*x*; *y*; *z*]. The object does not perform range validation for *x*, *y*, and *z*.
- When the CoordinateSystem property is set to 'Geographic', the form is [*latitude*; *longitude*; *height*] . The *latitude* must be in the range [–90, 90]. The object does not perform range validation for *longitude* and *height*. *height* is referenced to the ellipsoid defined by the World Geodetic System of 1984 (WGS84).

Data Types: double

Dependencies

To enable this property, set the PathSpecification property to 'Locations' and the LineOfSight property to 0 (false).

Data Types: struct

PropagationDelay — Propagation delay

5.7775e-08 | nonnegative scalar

Propagation delay in seconds, specified as a nonnegative scalar. The default value is computed using the default values of the TransmitterLocation and ReceiverLocation properties for a line-of-sight ray.

- When you set the PathSpecification property to 'Locations', this property is read-only and the value is derived from TransmitterLocation, ReceiverLocation and, if applicable, the Interactions.

- When you set the PathSpecification property to 'Delay and angles', this property is configurable.

Data Types: double

PropagationDistance — Propagation distance
17.3205 | nonnegative scalar

This property is read-only.

Propagation distance in meters, specified as a nonnegative scalar. The default value is computed using the default values of the TransmitterLocation and ReceiverLocation properties for a line-of-sight ray.

- When you set the PathSpecification property to 'Locations', the value is derived from TransmitterLocation, ReceiverLocation and, if applicable, the Interactions.
- When you set the PathSpecification property to 'Delay and angles', the value is derived from PropagationDelay.

Data Types: double

AngleOfDeparture — Angle of departure
[45; 35.2644] | numeric vector of the form [*az*; *el*]

Angle of departure in degrees of the ray at the transmitter, specified as a numeric vector of the form [*az*; *el*]. The azimuth angle, *az*, is measured from the positive x-axis counterclockwise and must be in the range (–180, 180]. The elevation angle, *el*, is measured from the x-y plane and must be in the range [–90, 90]. The default value is computed using the default values of the TransmitterLocation and ReceiverLocation properties for a line-of-sight ray.

- When you set the PathSpecification property to 'Delay and angles', this property is configurable.
- When you set the PathSpecification property to 'Locations', this property is read-only and the value is derived from TransmitterLocation, ReceiverLocation and, if applicable, the Interactions.
- When CoordinateSystem is set to 'Geographic', the angles are defined with reference to the local East-North-Up (ENU) coordinate system at transmitter.

Data Types: double

AngleOfArrival — Angle of arrival
[-135; -35.2644] | numeric vector of the form [*az*; *el*]

Angle of arrival in degrees of the ray at the receiver, specified as a numeric vector of the form [*az*; *el*]. The azimuth angle, *az*, is measured from the positive x-axis counterclockwise and must be in the range (–180, 180]. The elevation angle, *el*, is measured from the x-y plane and must be in the range [–90, 90]. The default value is computed using the default values of the TransmitterLocation and ReceiverLocation properties for a line-of-sight ray.

- When you set the PathSpecification property to 'Delay and angles', this property is configurable.
- When you set the PathSpecification property to 'Locations', this property is read-only and the value is derived from TransmitterLocation, ReceiverLocation and, if applicable, the Interactions.

- When `CoordinateSystem` is set to `'Geographic'`, the angles are defined with reference to the local East-North-Up (ENU) coordinate system at receiver.

Data Types: `double`

`NumInteractions` — Number of ray-surface interactions
`0` (default) | nonnegative integer

This property is read-only.

Number of ray-surface interactions for the ray object from the transmitter to the receiver, specified as a nonnegative integer. The value is derived from `LineOfSight` and, if applicable, the `Interactions`.

Dependencies

To enable this property, set the `PathSpecification` property to `'Locations'`.

Data Types: `double`

`Frequency` — Signal frequency
`1.9e+09` (default) | positive scalar

Signal frequency in Hz, specified as a positive scalar.

Data Types: `double`

`PathLossSource` — Path loss source
`'Free space model'` (default) | `'Custom'`

Path loss source, specified as `'Free space model'` or `'Custom'`.

Data Types: `char` | `string`

`PathLoss` — Path loss
`62.7941` | nonnegative scalar

Path loss in dB, specified as a nonnegative scalar. The default value is computed using the default values of the `TransmitterLocation` and `ReceiverLocation` properties for a line-of-sight ray.

- When you set the `PathLossSource` property to `'Free space model'`, the `PathLoss` property is read-only and derived from the `PropagationDistance` and `Frequency` properties by using the free space propagation model.
- When you set the `PathLossSource` property to `'Custom'`, you can set the `PathLoss` property, independent of the geometric properties.

Data Types: `double`

`PhaseShift` — Phase shift
`4.8537` | numeric scalar

Phase shift in radians, specified as a numeric scalar. The default value is computed using the default values of the `TransmitterLocation` and `ReceiverLocation` properties for a line-of-sight ray.

- When you set the `PathLossSource` property to `'Free space model'`, the `PhaseShift` property is read-only and derived from the `PropagationDistance` and `Frequency` properties by using the free space propagation model.

- When you set the PathLossSource property to 'Custom', you can set the PhaseShift property, independent of the geometric properties.

Data Types: double

Object Functions

plot (rays)   Display RF propagation rays in Site Viewer

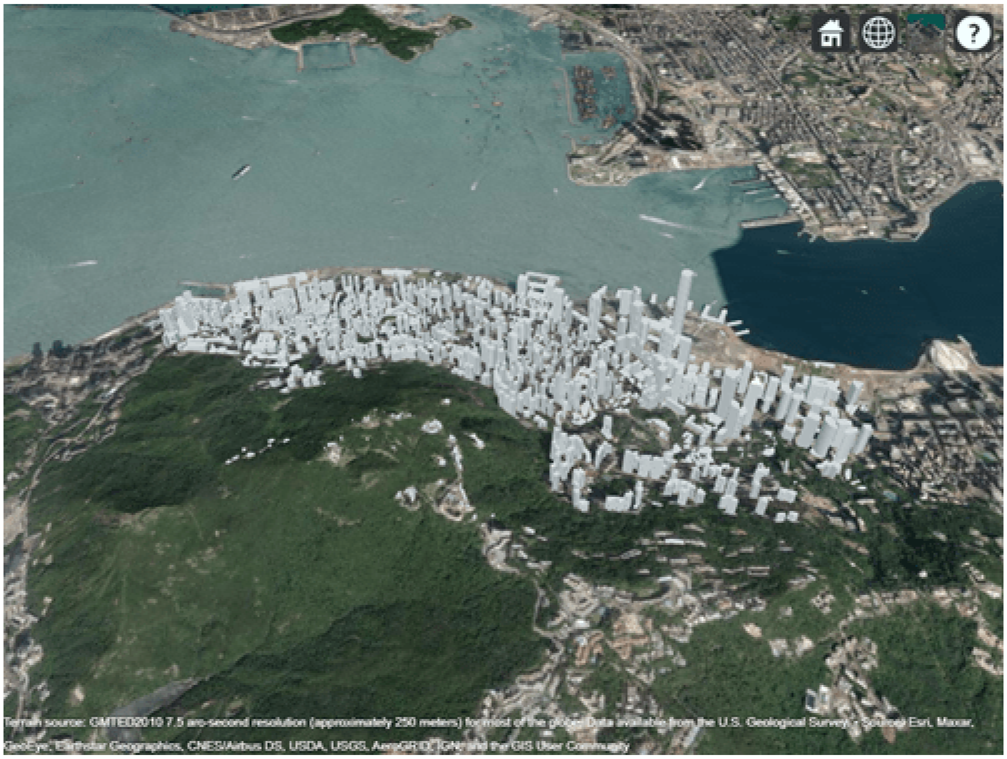
Examples

Perform Ray Tracing Between Two Sites in Hong Kong

Perform ray tracing between two sites in Hong Kong, generating a cell array containing comm.Ray objects. The comm.Ray objects contain the geometric and electromagnetic information for the radio wave propagation paths from the transmitter site to the receiver site.

Create a Site Viewer map, loading building data for Hong Kong. For more information about the osm file, see [1] on page 6-260.

```
viewer = siteviewer("Buildings","hongkong.osm");
```


?
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Sources: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Specify transmitter and receiver sites.

```
tx = txsite("Latitude",22.2789,"Longitude",114.1625, ...
    "AntennaHeight",10,"TransmitterPower",5, ...
    "TransmitterFrequency",28e9);
rx = rxsite("Latitude",22.2799,"Longitude",114.1617, ...
    "AntennaHeight",1);
show(tx)
show(rx)
```

Specify a ray tracing propagation model that uses the SBR method with up to three reflections.

```
pm = propagationModel("raytracing", ...
    "Method","sbr", ...
    "MaxNumReflections",3);
```

Perform ray tracing between the sites, generating `comm.Ray` objects in a cell array. For the specified transmitter and receiver sites, performing ray tracing results in a 1-by-1 cell array containing three ray objects in a row vector.

```
rays = raytrace(tx,rx,pm)
```

```
rays = 1×1 cell array
    {1×11 comm.Ray}
```

Display the properties of the first `comm.Ray` object. The `LineOfSight` property value is `1`, and the `NumInteractions` property value is `0`. This combination indicates that the ray defines a line-of-sight path.

```
rays{1}(1)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 1
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 104.2656
             PhaseShift: 4.6360

   Read-only properties:
       PropagationDelay: 4.6442e-07
    PropagationDistance: 139.2294
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 0
```

Display the properties of the second and third `comm.Ray` objects. The `LineOfSight` property values are `0`, and the `NumInteractions` property values are greater than `0`. This combination indicates that the rays define reflected paths.

```
rays{1}(2)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
```

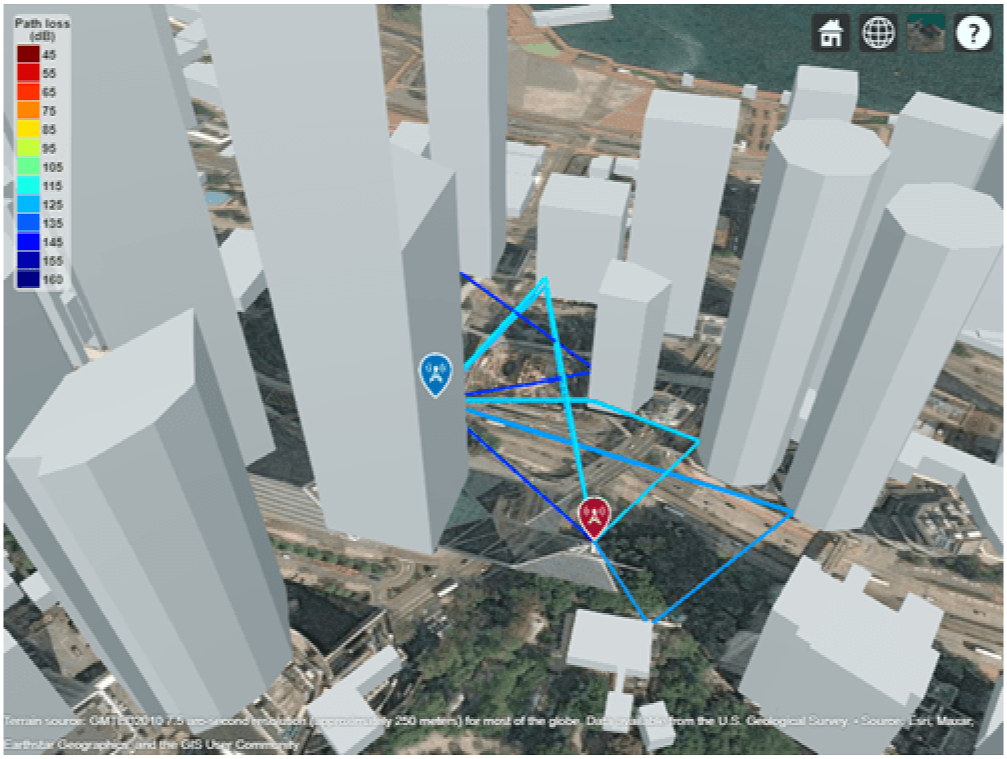
```
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×1 struct]
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 106.1294
             PhaseShift: 0.3952

   Read-only properties:
       PropagationDelay: 4.6490e-07
    PropagationDistance: 139.3720
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 1
```

```
rays{1}(3)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×1 struct]
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 119.9462
             PhaseShift: 0.3965

   Read-only properties:
       PropagationDelay: 1.1327e-06
    PropagationDistance: 339.5692
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 1
```

Visualize ray tracing results.

```
plot(rays{1})
```


Path loss
(dB)
45
55
65
75
85
95
105
115
125
135
145
155
160
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

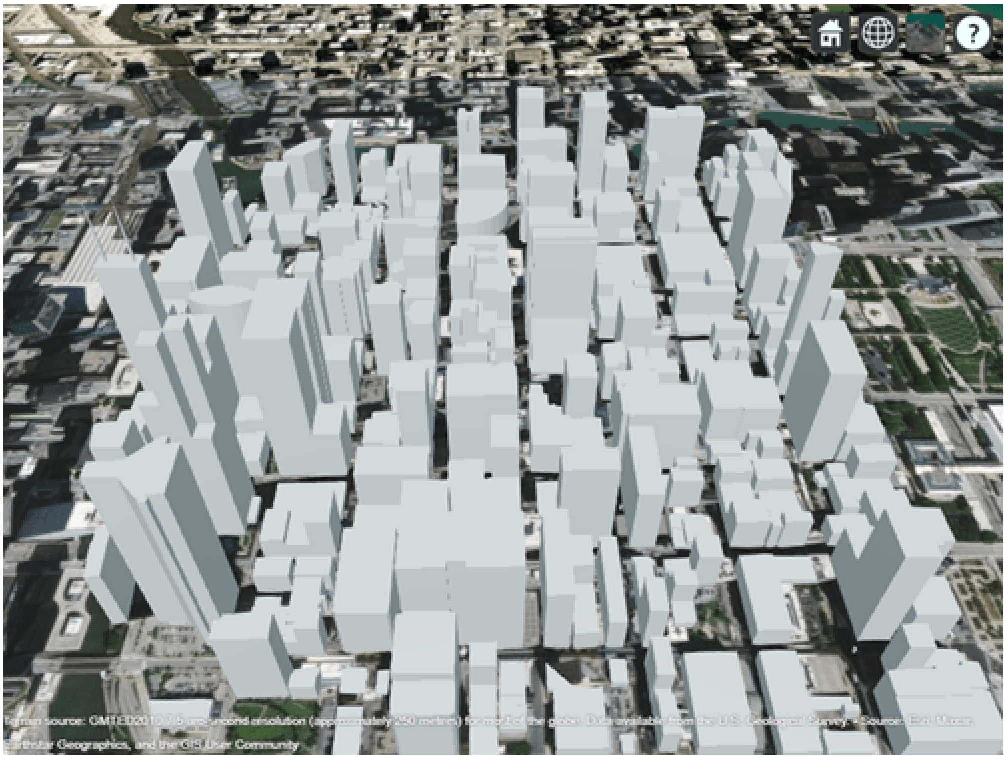
[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Plot Propagation Rays Between Sites in Chicago

Return ray tracing results in `comm.Ray` objects and plot the ray propagation paths after relaunching the Site Viewer map.

Create a Site Viewer map, loading building data for Chicago. For more information about the osm file, see [1] on page 6-264.

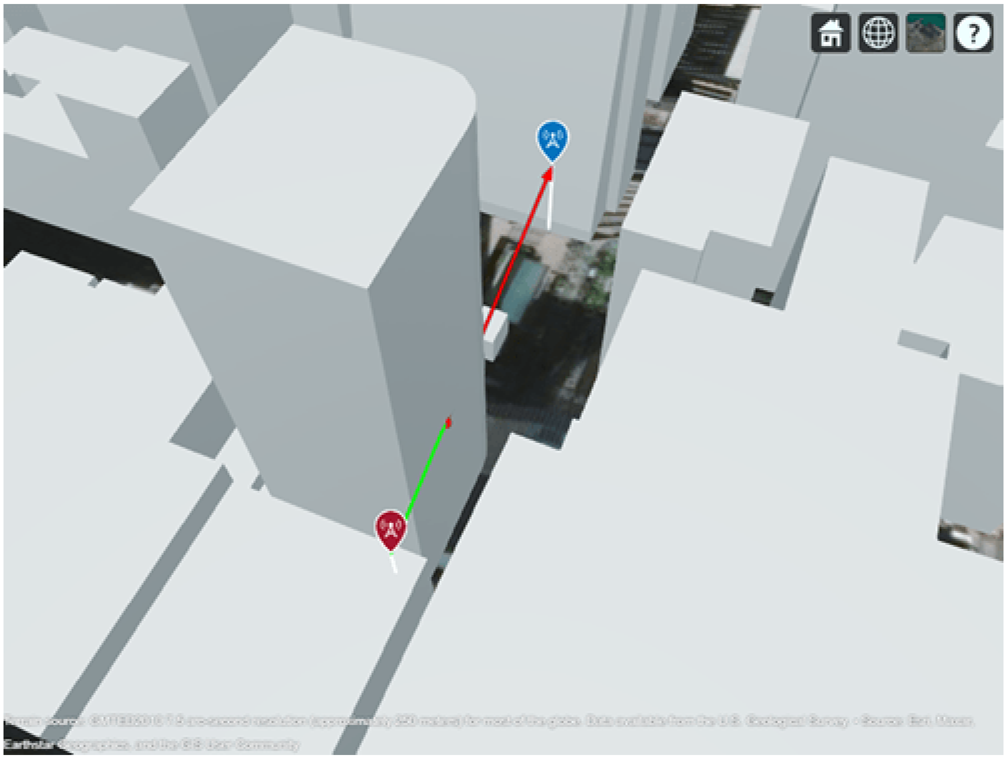
```
viewer = siteviewer("Buildings","chicago.osm");
```



Terrain source: GMTED2010 7.5-arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Create a transmitter site on one building and a receiver site on another building. Use the `los` function to show the line of sight path between the transmitter and receiver sites.

```
tx = txsite( ...
    "Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
rx = rxsite( ...
    "Latitude",41.881352, ...
    "Longitude",-87.629771, ...
    "AntennaHeight",30);
los(tx,rx)
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Sources: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Perform ray tracing for up to two reflections. For the configuration defined, ray tracing returns a cell array containing the ray objects. Close the Site Viewer map.

```
pm = propagationModel( ...
    "raytracing", ...
    "Method","sbr", ...
    "MaxNumReflections",2);
rays = raytrace(tx,rx,pm)
```

```
rays = 1×1 cell array
    {1×3 comm.Ray}
```

```
rays{1}(1,1)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×1 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 92.7739
             PhaseShift: 1.2933

   Read-only properties:
       PropagationDelay: 5.7088e-07
    PropagationDistance: 171.1462
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 1
```

```
rays{1}(1,2)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 100.8574
             PhaseShift: 2.9398

   Read-only properties:
       PropagationDelay: 5.9259e-07
    PropagationDistance: 177.6532
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
rays{1}(1,3)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 106.3302
             PhaseShift: 4.6994

   Read-only properties:
       PropagationDelay: 6.3790e-07
    PropagationDistance: 191.2374
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
close(viewer)
```

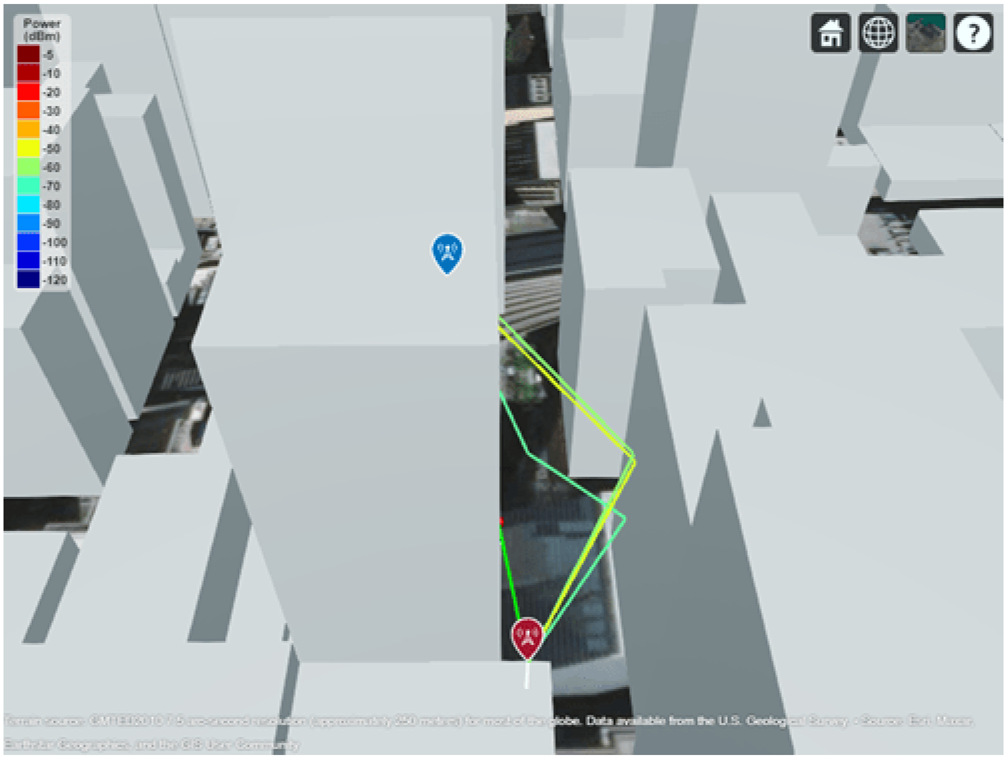
You can plot the rays without performing ray tracing again. Create another Site Viewer map with the same buildings. Show the transmitter and receiver sites. Using the previously returned cell array of ray objects, plot the reflected rays between the transmitter site and the receiver site. The plot

function can plot the path for ray objects collectively or individually. For example, to plot rays for the only second ray object, specify `rays{1}(1,2)`. This figure plot all paths for all the ray objects.

```
siteviewer("Buildings","chicago.osm")
```

```
ans = 
  siteviewer with properties:

                Name: 'Site Viewer'
            Position: [560 240 800 600]
    CoordinateSystem: "geographic"
             Basemap: 'satellite'
             Terrain: 'gmted2010'
           Buildings: 'chicago.osm'
```

```
los(tx,rx)
plot(rays{:},"Type","power", ...
    "TransmitterSite",tx,"ReceiverSite",rx)
```


Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Version History

Introduced in R2020a

`ReflectionLocations` and `NumReflections` properties have been removed

Errors starting in R2021b

The `ReflectionLocations` and `NumReflections` properties have been removed. To accommodate reflections, use the `Interactions` property to replace the `ReflectionLocations` property and use the `NumInteractions` property to replace the `NumReflections` property.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`raytrace` | `raypl` | `buildingMaterialPermittivity` | `earthSurfacePermittivity` | `propagationModel`

Objects

`siteviewer`

Topics

“Choose a Propagation Model”

plot (rays), plot

Package: `comm`

Display RF propagation rays in Site Viewer

Syntax

```
plot(rays)
plot(rays,Name,Value)
```

Description

`plot(rays)` plots the propagation paths for ray objects in the Site Viewer map.

`plot(rays,Name,Value)` plots the propagation paths for ray objects in the Site Viewer map with additional options specified by one or more name-value pair arguments.

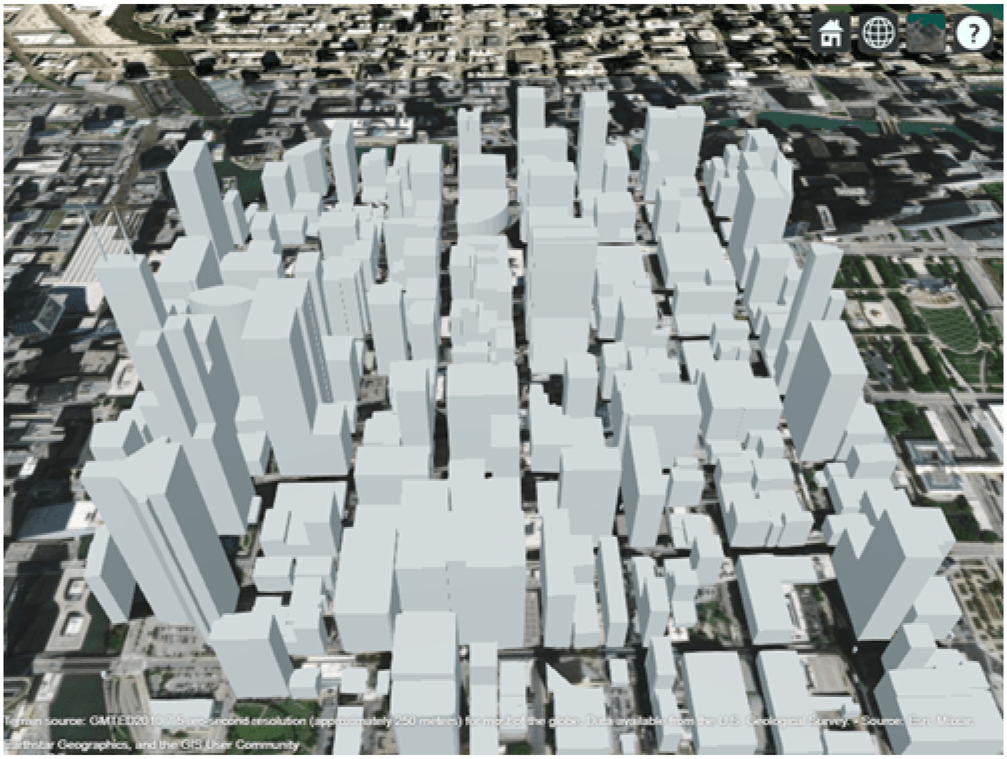
Examples

Plot Propagation Rays Between Sites in Chicago

Return ray tracing results in `comm.Ray` objects and plot the ray propagation paths after relaunching the Site Viewer map.

Create a Site Viewer map, loading building data for Chicago. For more information about the osm file, see [1] on page 6-270.

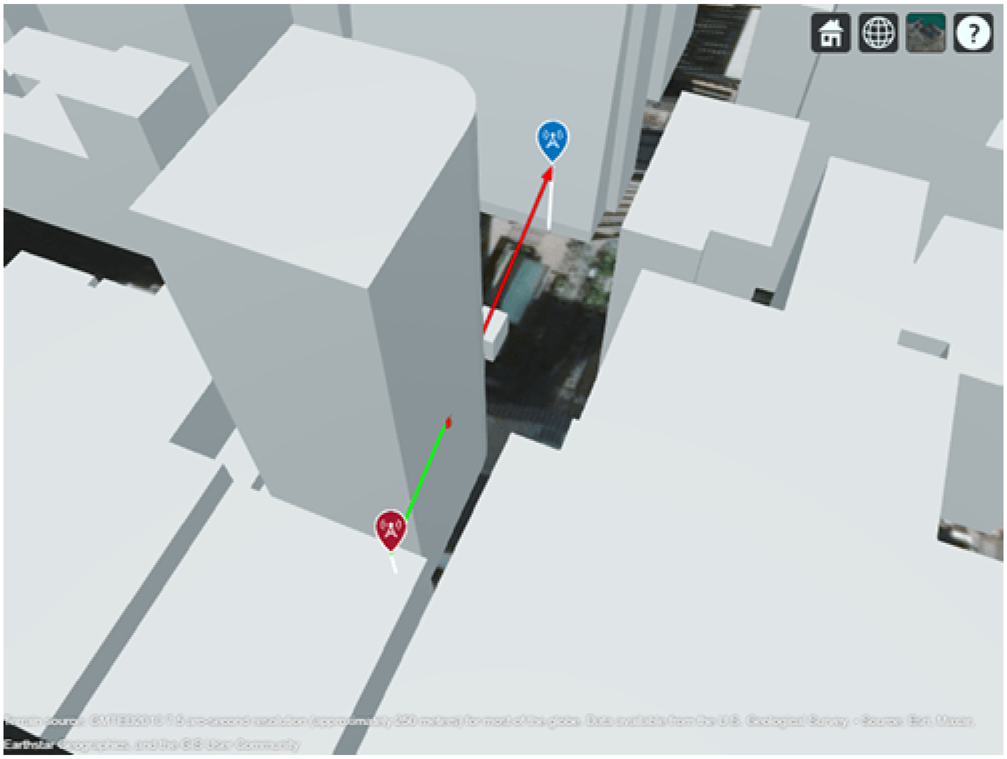
```
viewer = siteviewer("Buildings","chicago.osm");
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Create a transmitter site on one building and a receiver site on another building. Use the `los` function to show the line of sight path between the transmitter and receiver sites.

```
tx = txsite( ...
    "Latitude",41.8800, ...
    "Longitude",-87.6295, ...
    "TransmitterFrequency",2.5e9);
rx = rxsite( ...
    "Latitude",41.881352, ...
    "Longitude",-87.629771, ...
    "AntennaHeight",30);
los(tx,rx)
```



Perform ray tracing for up to two reflections. For the configuration defined, ray tracing returns a cell array containing the ray objects. Close the Site Viewer map.

```
pm = propagationModel( ...
    "raytracing", ...
    "Method","sbr", ...
    "MaxNumReflections",2);
rays = raytrace(tx,rx,pm)
```

```
rays = 1×1 cell array
    {1×3 comm.Ray}
```

```
rays{1}(1,1)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×1 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 92.7739
             PhaseShift: 1.2933

   Read-only properties:
       PropagationDelay: 5.7088e-07
    PropagationDistance: 171.1462
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 1
```

```
rays{1}(1,2)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 100.8574
             PhaseShift: 2.9398

   Read-only properties:
       PropagationDelay: 5.9259e-07
    PropagationDistance: 177.6532
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
rays{1}(1,3)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
           Interactions: [1×2 struct]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 106.3302
             PhaseShift: 4.6994

   Read-only properties:
       PropagationDelay: 6.3790e-07
    PropagationDistance: 191.2374
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
        NumInteractions: 2
```

```
close(viewer)
```

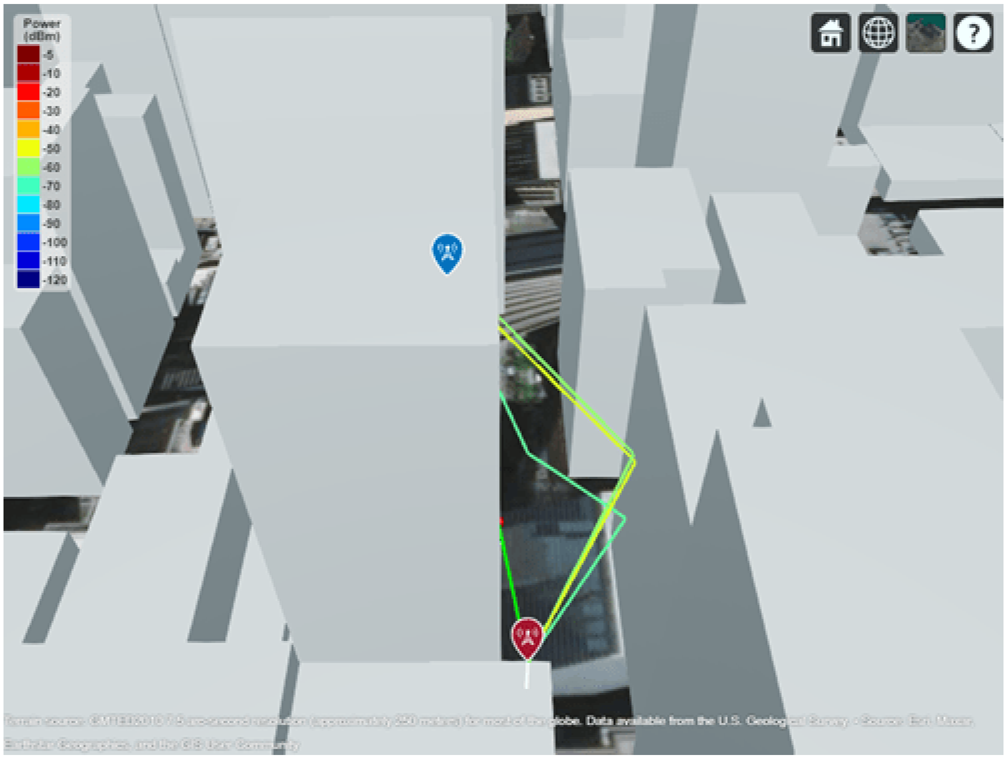
You can plot the rays without performing ray tracing again. Create another Site Viewer map with the same buildings. Show the transmitter and receiver sites. Using the previously returned cell array of ray objects, plot the reflected rays between the transmitter site and the receiver site. The plot

function can plot the path for ray objects collectively or individually. For example, to plot rays for the only second ray object, specify `rays{1}(1,2)`. This figure plot all paths for all the ray objects.

```
siteviewer("Buildings","chicago.osm")
```

```
ans = 
  siteviewer with properties:

                Name: 'Site Viewer'
            Position: [560 240 800 600]
    CoordinateSystem: "geographic"
             Basemap: 'satellite'
             Terrain: 'gmted2010'
           Buildings: 'chicago.osm'
```

```
los(tx,rx)
plot(rays{:},"Type","power", ...
    "TransmitterSite",tx,"ReceiverSite",rx)
```


Power
(dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community

Appendix

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Input Arguments

rays — Ray configuration object

comm.Ray object

Ray configuration, specified as one comm.Rayobject or a vector of comm.Ray objects. Each object must have the PathSpecification property set to "Locations".

Data Types: comm.Ray

Name-Value Pair Arguments

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

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

Example: plot(rays,"Type","pathloss","ColorLimits",[-100 0]) adds the propagation path specified in rays to the current Site Viewer and adjusts the default color limits.

Type — Quantity type to plot

"pathloss" (default) | "power"

Quantity type to plot, specified as "pathloss" or "power". Based on the value specified for Type, the color applied along the path maps to the path loss in dB or the power in dBm of the signal along the path.

Data Types: char | string

TransmitterSite — Transmitter site

txsite object

Transmitter site, specified as a txsite object.

Dependencies

Applies only when Type is set to "power".

Data Types: char

ReceiverSite — Receiver site

rxsite object

Receiver site, specified as an rxsite object.

Dependencies

Applies only when Type is set to "power".

Data Types: char

ColorLimits — Colormap color limits

[-120 -5] or [45 160] (default) | 1-by-2 numeric vector

Color limits for colormap, specified as a 1-by-2 numeric vector, [*min*, *max*], where *min* represents the lower saturation limit and *max* represents the upper saturation limit. The default is [-120 -5] when `Type` is set to `'power'` and [45 160] when `Type` is set to `'pathloss'`.

Data Types: `double`

`Colormap` — Colormap applied to propagation path

`'jet'` (default) | *M*-by-3 numeric array

Colormap applied to propagation path, specified as an *M*-by-3 numeric array of RGB (red,green,blue) triplets that define *M* individual colors.

Data Types: `double` | `char` | `string`

`ShowLegend` — Show color legend on map

`true` (default) | `false`

Show color legend on map, specified as `true` or `false`.

Data Types: `logical`

`Map` — Map for visualization and surface data

`siteviewer` object

Map for visualization and surface data, specified as a `siteviewer` object.[10] The default is the current `siteviewer` object, or if no Site Viewer is open a new `siteviewer` object opens.

Data Types: `siteviewer object`

Version History

Introduced in R2020a

See Also

Functions

`raytrace`

Objects

`comm.Ray` | `siteviewer`

[10] Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.