Antenna Toolbox™

Reference

MATLAB®

R2020a

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)

Contents

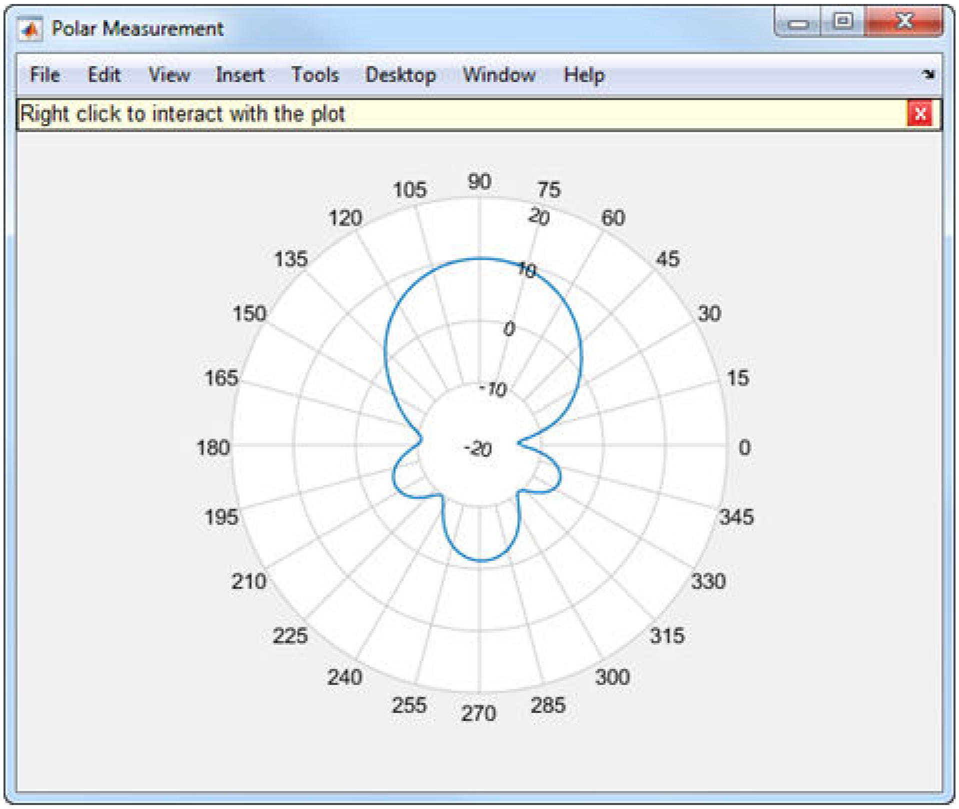
1

Classes

polarpattern class

Interactive plot of radiation patterns in polar format

Description



`polarpattern` class plots antenna or array radiation patterns in interactive polar format. You can also plot other types of polar data. Use these plots when interactive data visualization or measurement is required. Right-click the **Polar Measurement** window to change the properties, zoom in, or add more data to the plot.

Construction

`polarpattern` plots antenna or array radiation patterns and other types of data in polar format. `polarpattern` plots field value data of radiation patterns for visualization and measurement. Right-click the polar plot to interact.

`polarpattern(data)` creates a polar plot with magnitude values in the vector `d`. In this polar plot, angles are uniformly spaced on the unit circle, starting at `0` degrees.

`polarpattern(angle,magnitude)` creates a polar plot from a set of angle vectors and corresponding magnitudes. You can also create polar plots from multiple sets for angle vectors and corresponding sets of magnitude using the syntax: `polarpattern(angle1, magnitude1, angle2, magnitude2...)`.

`p = polarpattern( ___ )` returns an object handle that you can use to customize the plot or add measurements. You can specify any of the arguments from the previous syntaxes.

`p = polarpattern('gco')` returns an object handle from polar pattern in the current figure.

`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`. Properties not specified retain their default values. To list all the property `Name,Value` pairs, use `details(p)`. To list all the property `Name,Value` pairs, use `details(p)`. You can use the properties to extract any data from 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 handle, `ax` instead of the current axes handle.

Input Arguments

`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.
- 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 (`data`) and *y* contains the imaginary (`data`).

When 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.

`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.

Methods

`add`	Add data to existing polar plot
`addCursor`	Add cursor to polar plot angle
`animate`	Replace existing data with new data for animation
`createLabels`	Create legend labels
`findLobes`	Main, back and side lobe data
`replace`	Replace existing data with new data in polar plot
`showPeaksTable`	Show or hide peak marker table
`showSpan`	Show or hide angle span between two markers

Examples

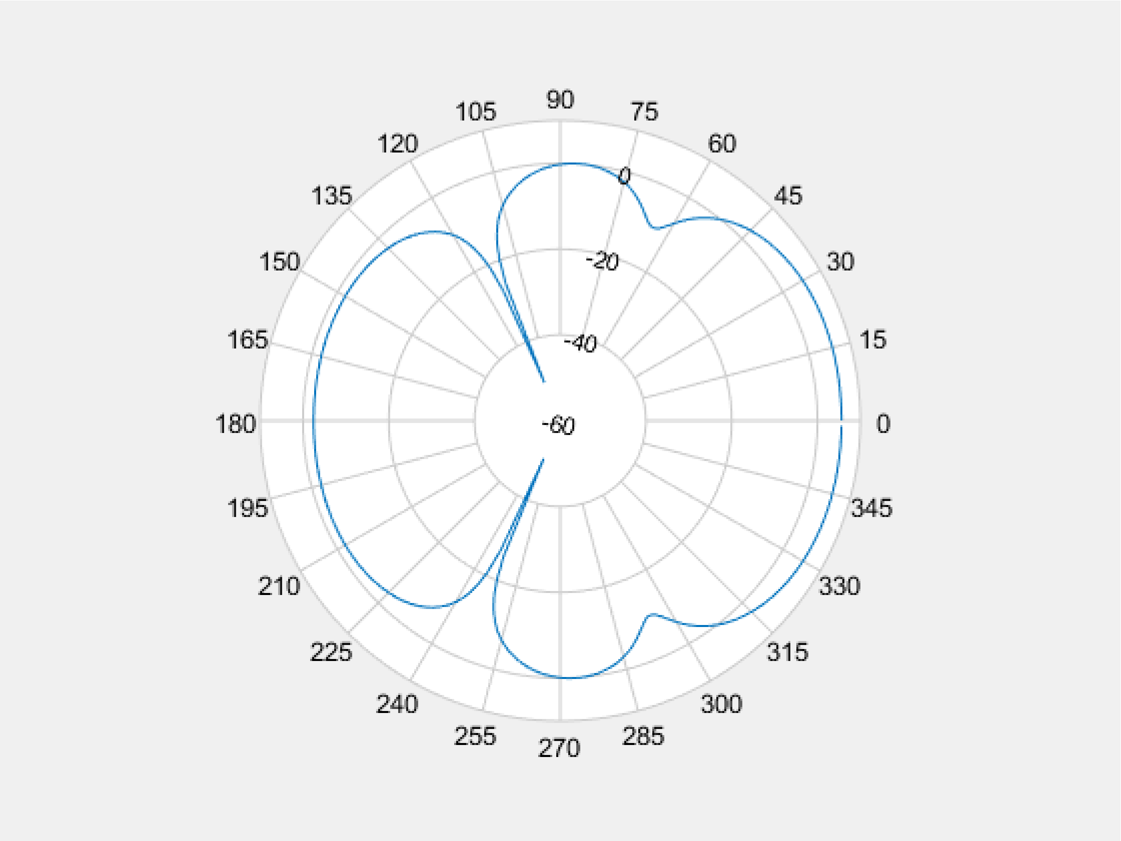
Polar Pattern for Vivaldi Antenna

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.

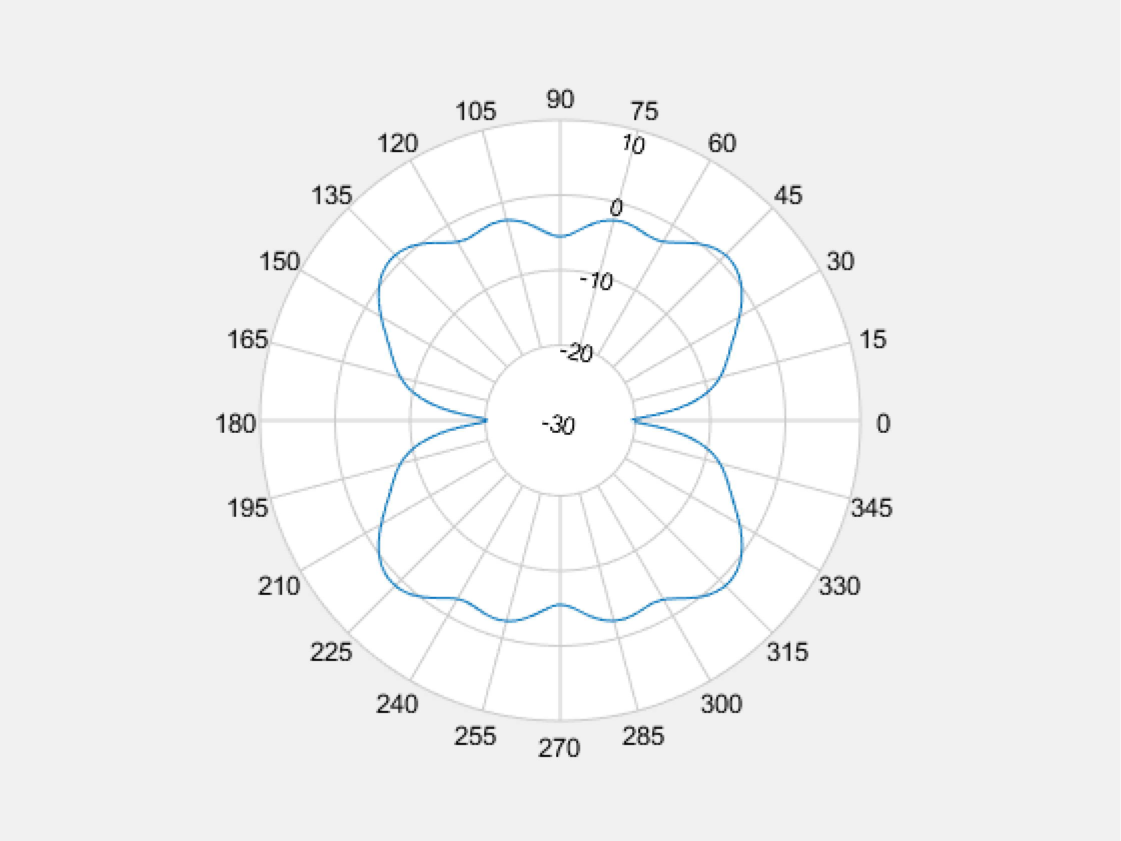
```
[Horizontal,Vertical,Optional] = msiread('cavity.pln')
```

```
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);
```



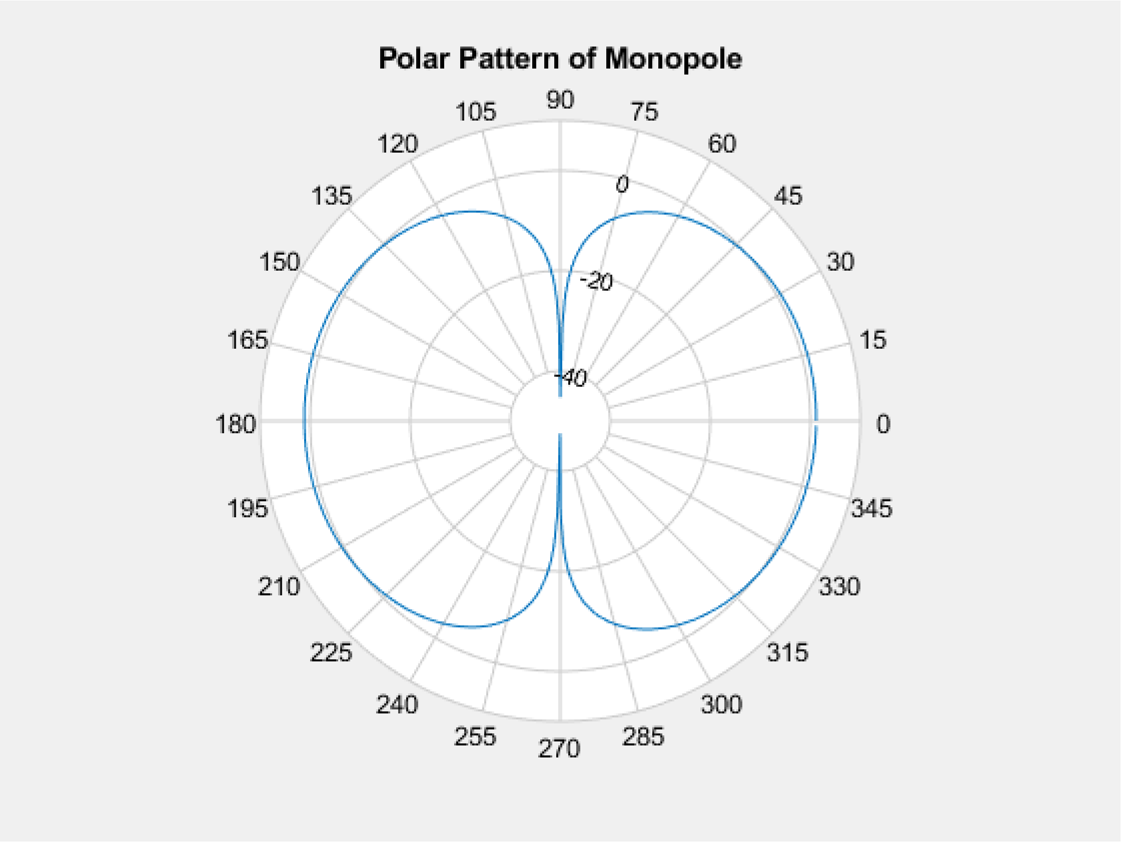
Add Title to Polar Plot

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');
```



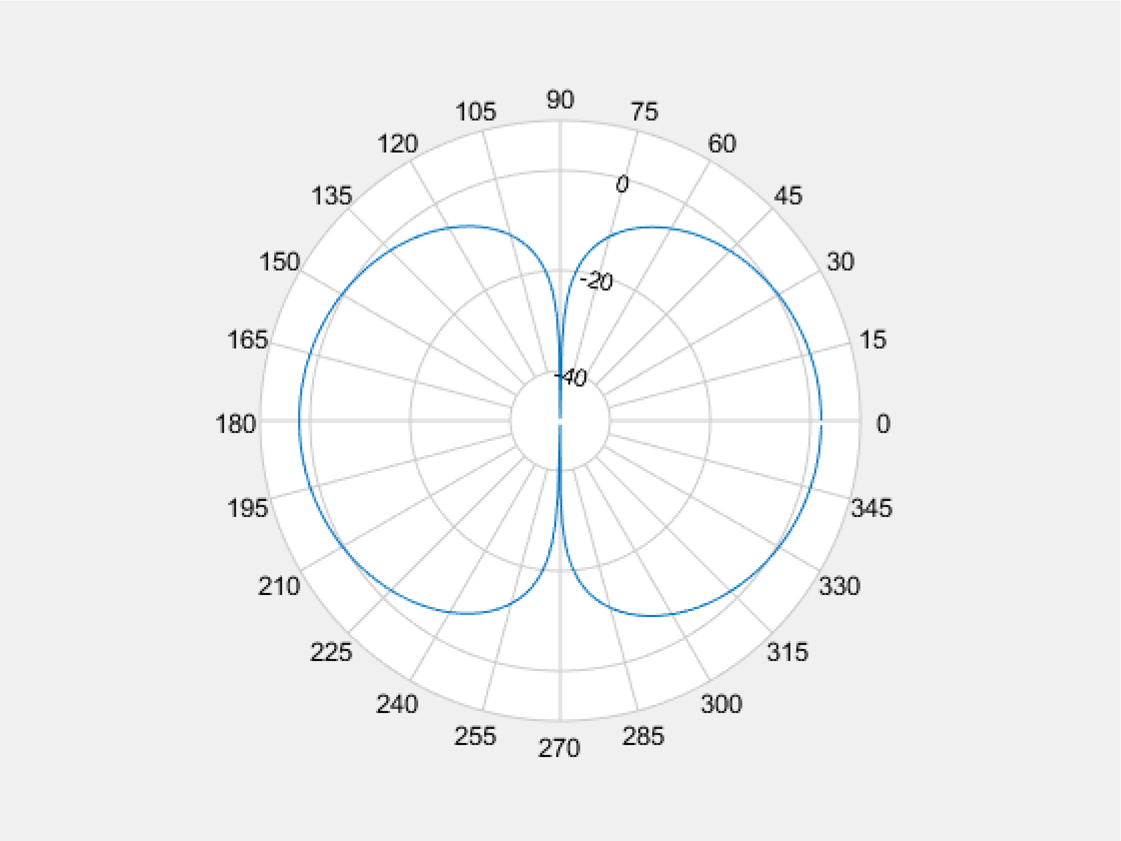
Polar Pattern Properties

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: [-50 10]
               MagnitudeAxisAngle: 75
                    MagnitudeTick: [-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
```

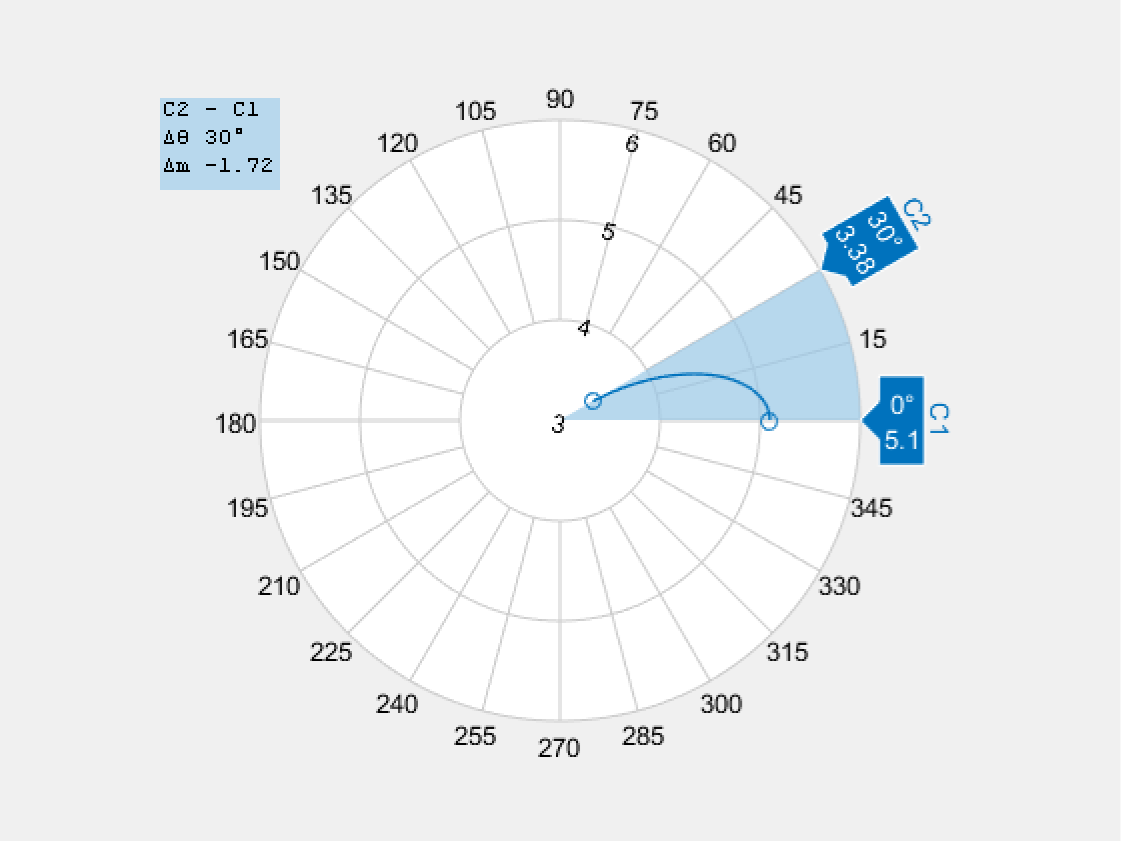
Remove -Inf and NaN Values in Antenna PolarPattern

Use `Clean Data` in `Antenna Metrics` to remove -inf and NaN values in a monopole antenna polar pattern. It is recommended to use `Clean Data` for partial data with -inf and NaN values.

```
m = monopole;
m.GroundPlaneLength = inf;
```

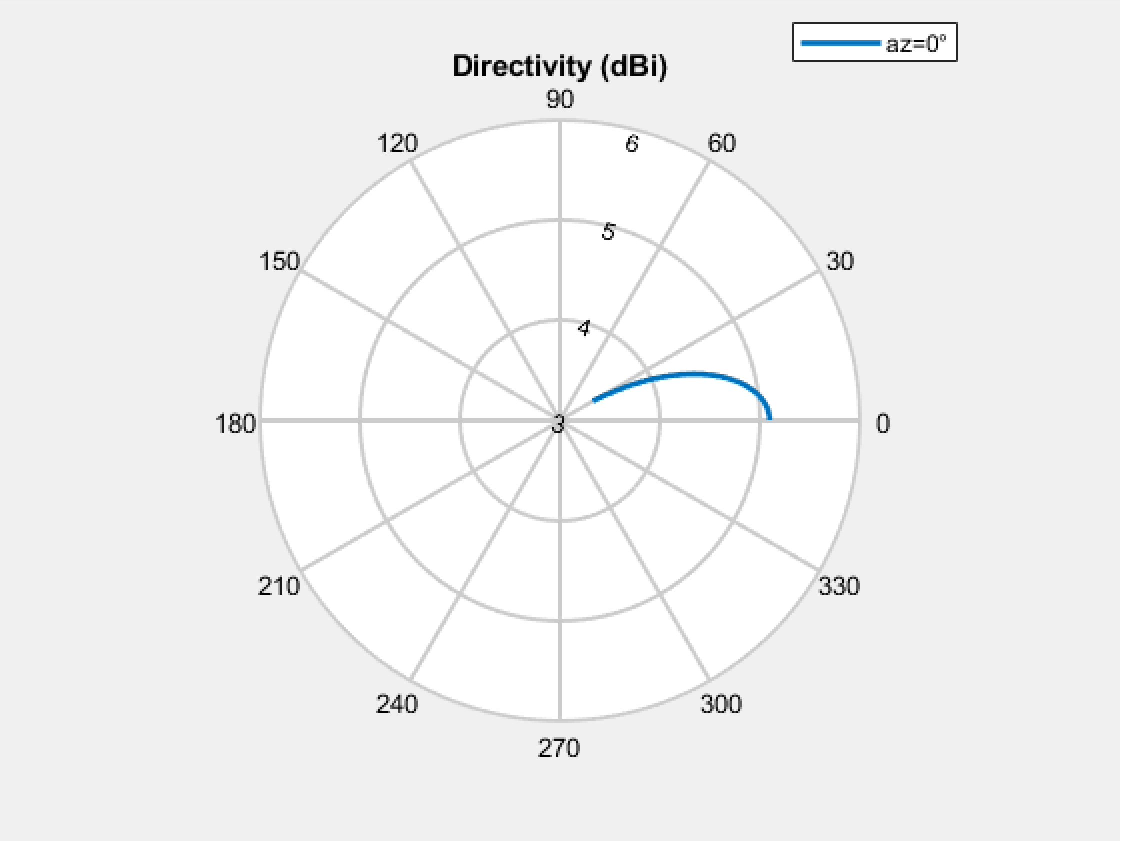
Plot the beamwidth of the antenna at 70 MHz.

```
figure;
beamwidth(m,70e6,0,-50:30)
```



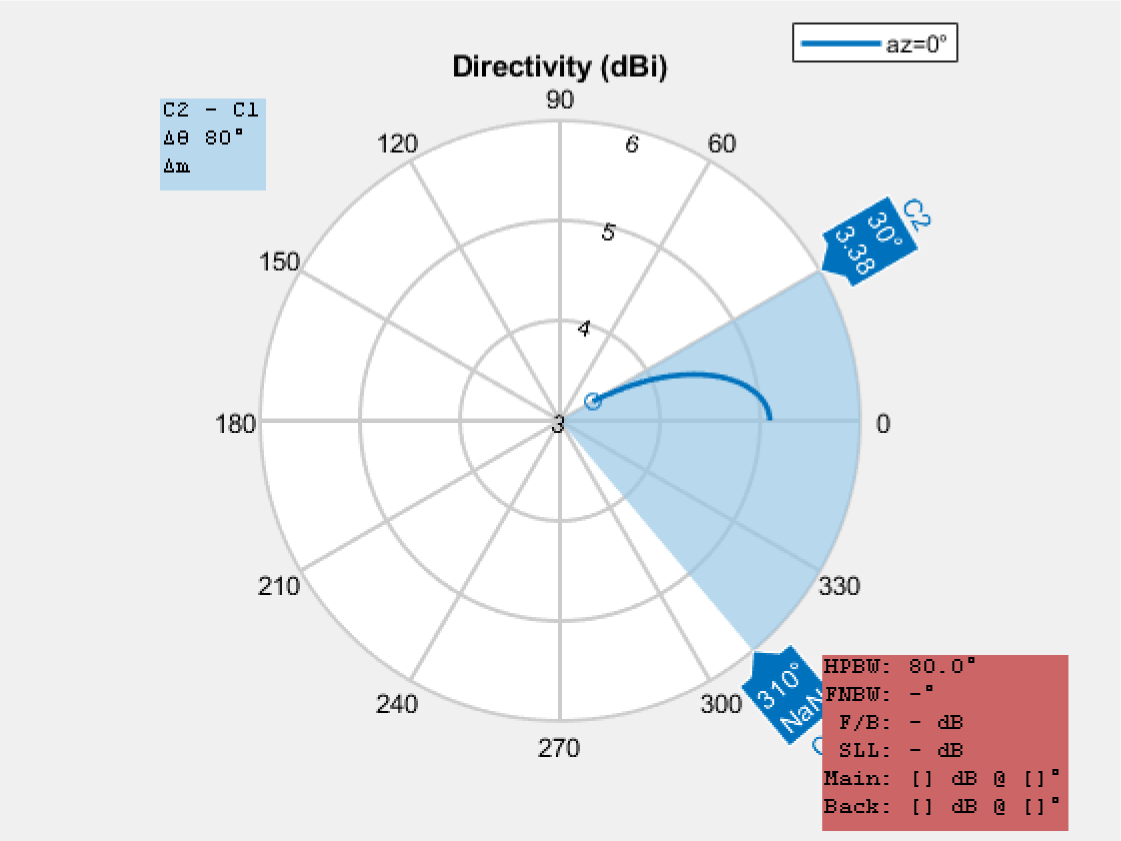
Plot the radiation pattern of the antenna at 70 MHz.

```
figure;
pattern(m,70e6,0,-50:30);
```



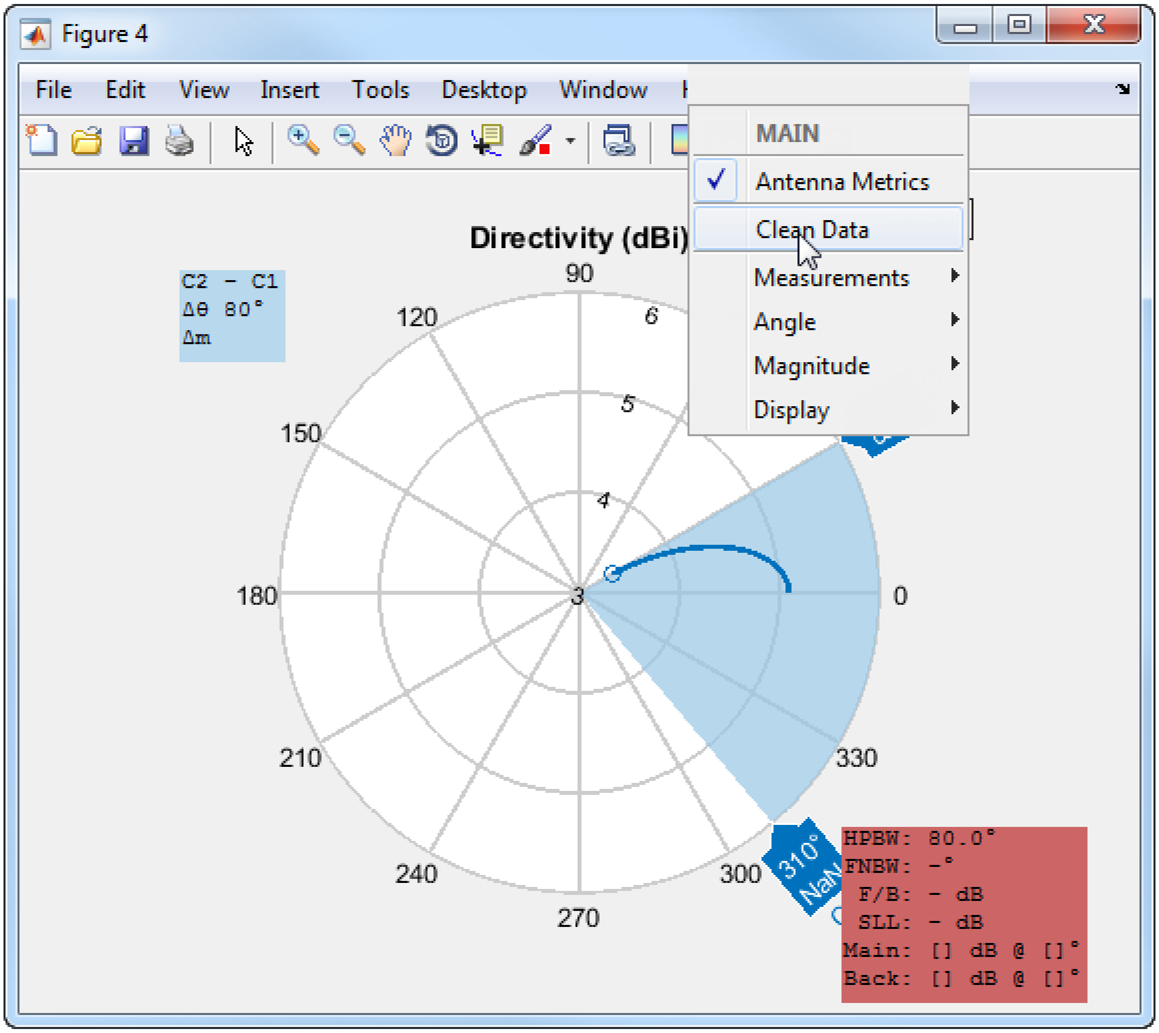
Use `polarpattern` to view antenna metrics of the radiation pattern.

```
P = polarpattern('gco');
P.AntennaMetrics = 1;
```



Compare the beamwidth plot and the polarpattern plot. You will see that Antenna Metrics does not represent the beamwidth correctly.

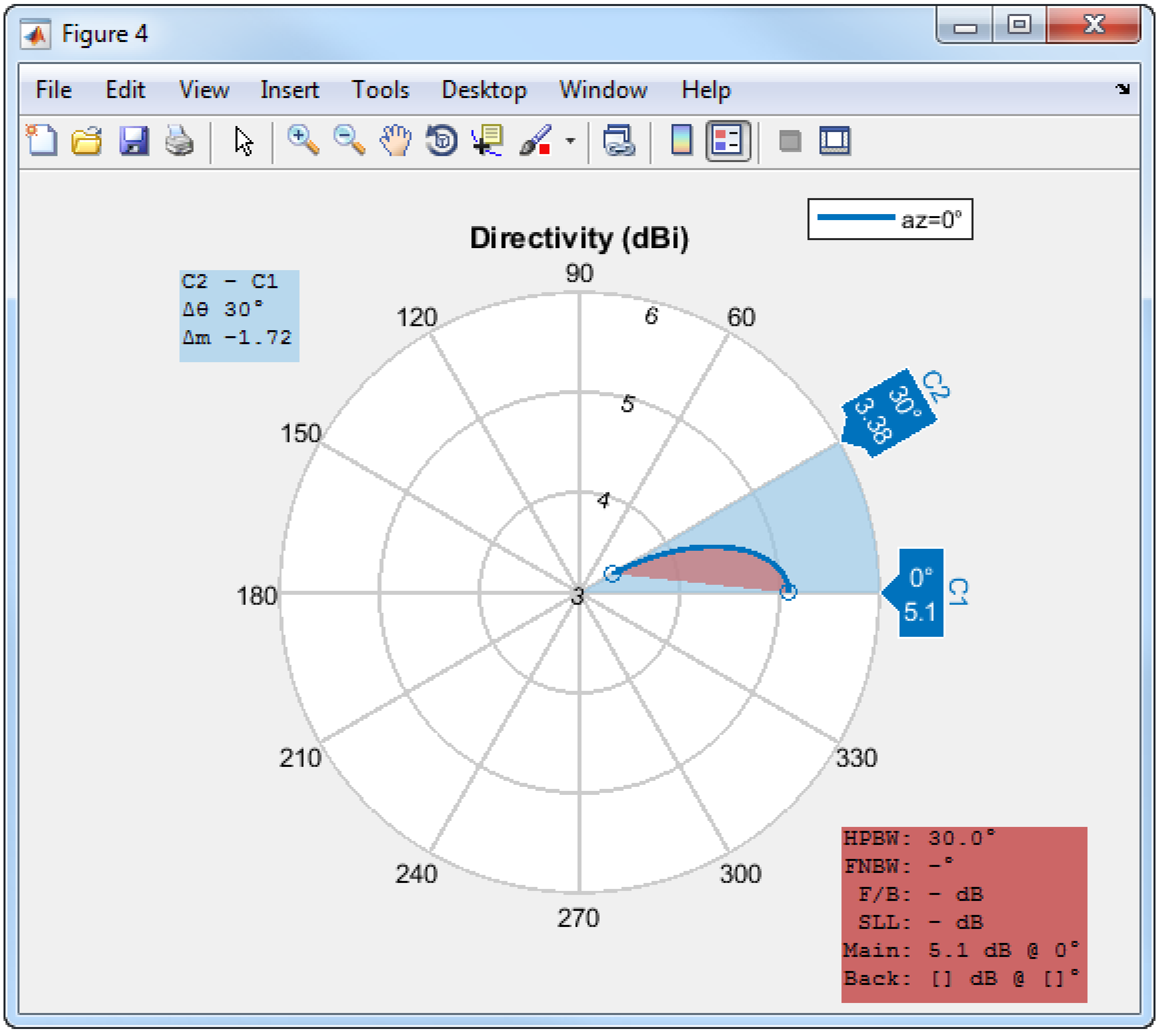
Use `Clean Data` to clean the -inf and NaN values.



```
C2 - C1
Δθ 80°
Δm
```

```
HPBW: 80.0°
FNBW: -°
 F/B: - dB
 SLL: - dB
Main: [] dB @ []°
Back: [] dB @ []°
```

After using `Clean Data`, you see that the polarpattern beamwidth 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 @ []°
```

See Also

Topics

"Interact with Polar Plot"

Introduced in R2016a

2

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.

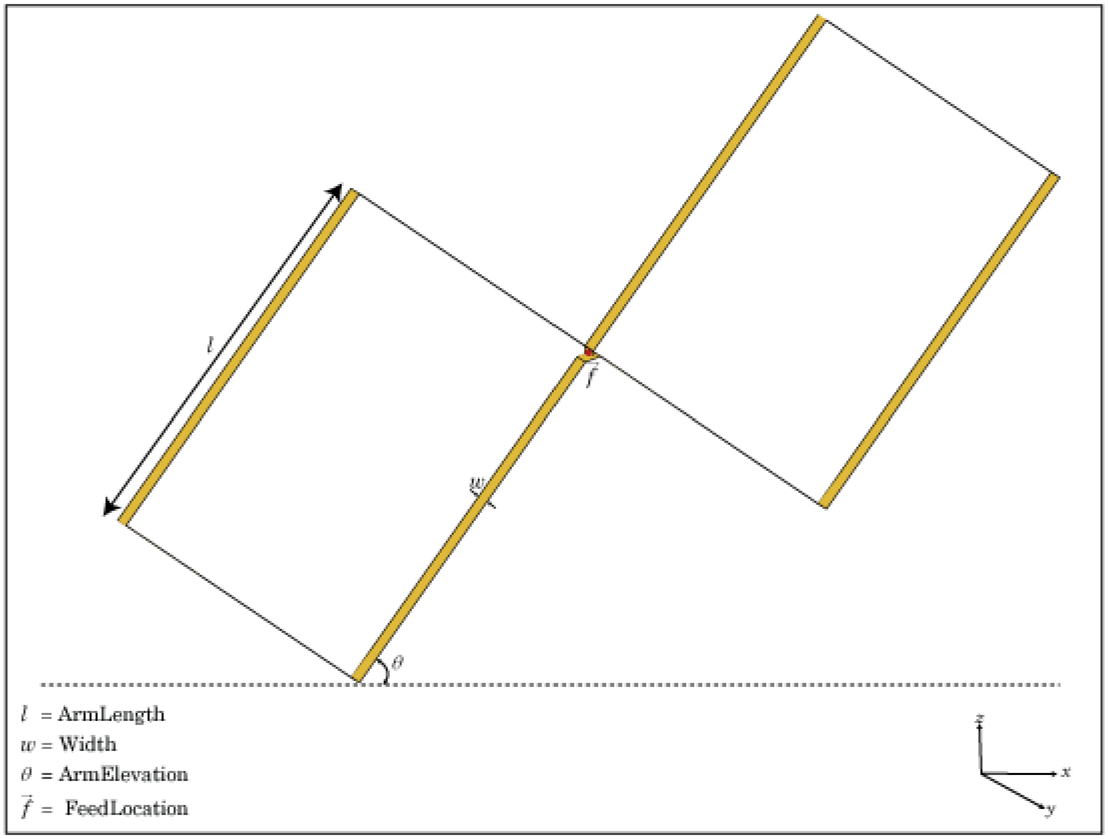
The width of the strip is related to the diameter an equivalent cylinder:

$$w = 2d = 4r$$

, 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 Y-Z 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 X-Y plane
45 (default) | scalar

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

Example: `'ArmElevation'`, `50`

Data Types: `double`

Load — Lumped elements

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

Lumped elements added to the antenna feed, specified 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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array

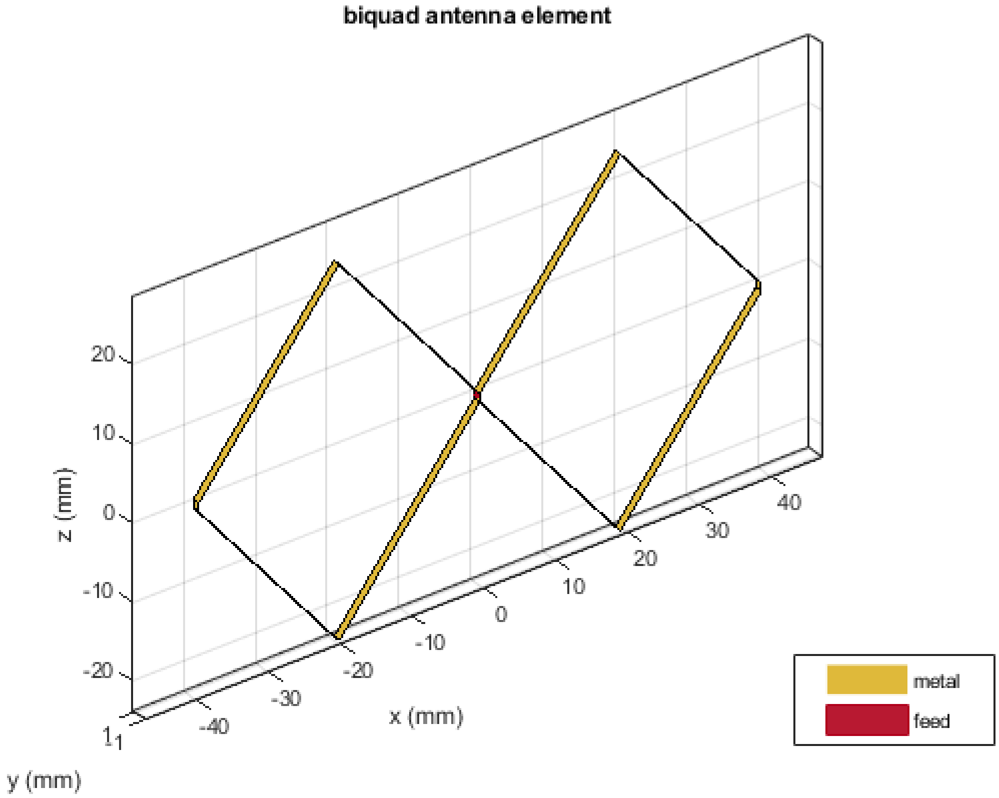
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at specified frequency
EHfields	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 or 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	S-parameter object
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.

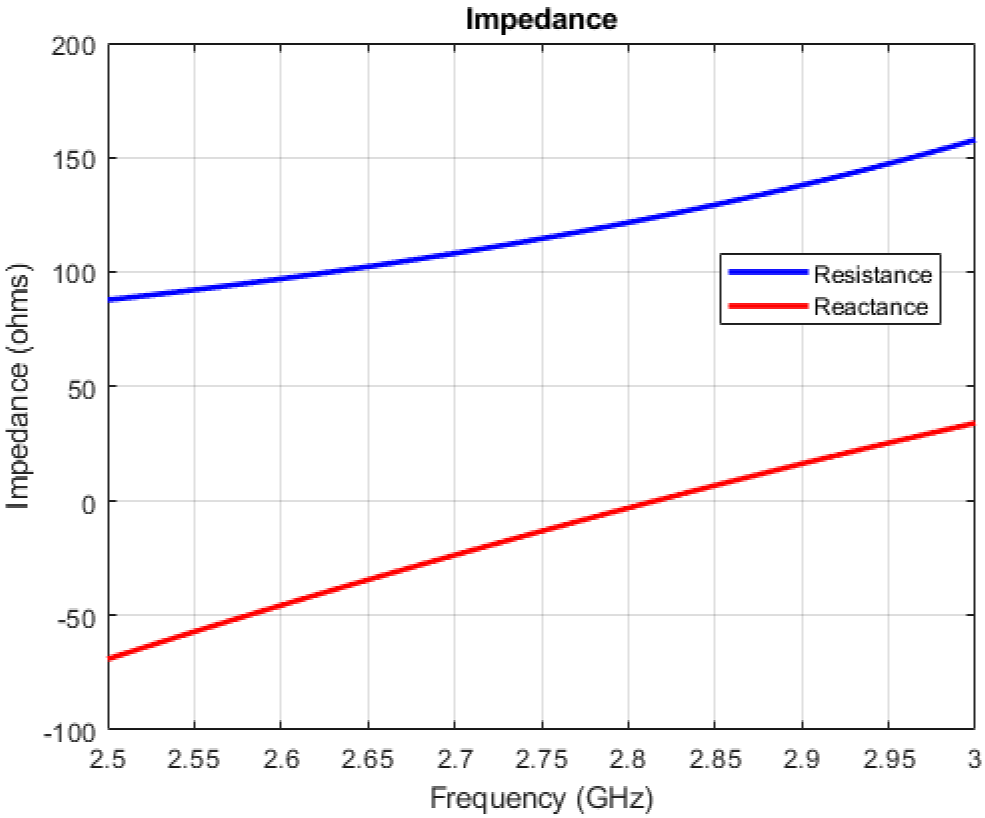
```
bq = biquad('ArmElevation',50);
show(bq)
```



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));
```



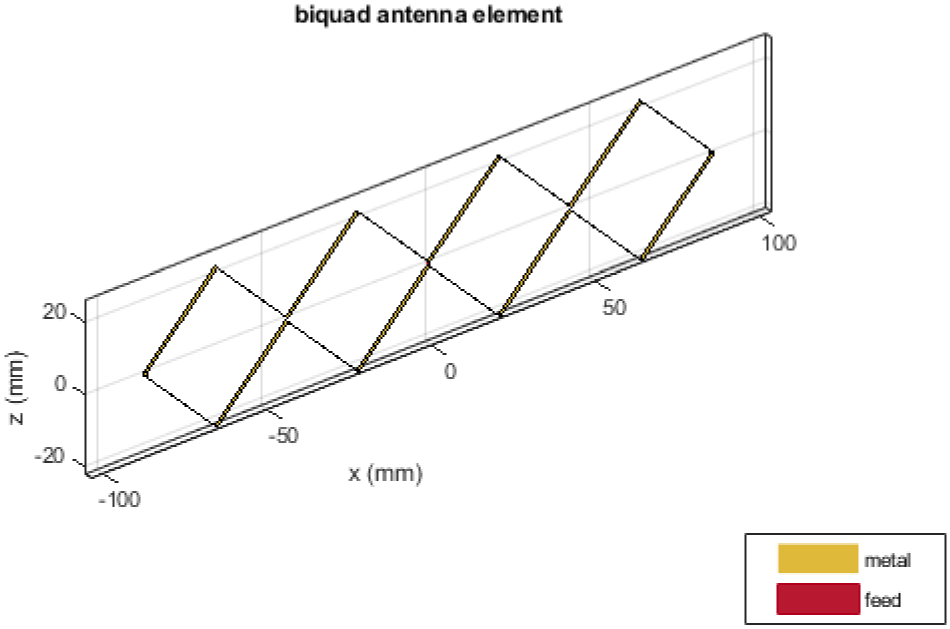
Double Biquad Antenna

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

```
show(ant)
```



See Also

`dipole` | `dipoleFolded` | `loopCircular`

Topics

"Rotate Antennas and Arrays"

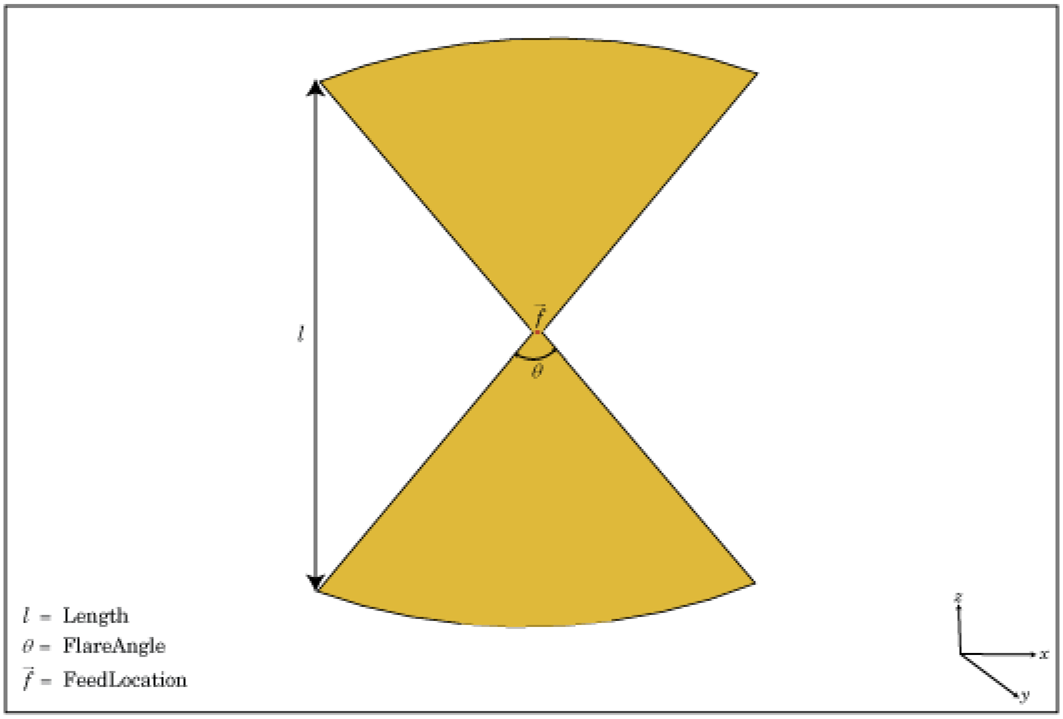
Introduced in R2015b

bowtieRounded

Create rounded bowtie dipole antenna

Description

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

`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: `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: `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 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 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 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'`

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

Object Functions

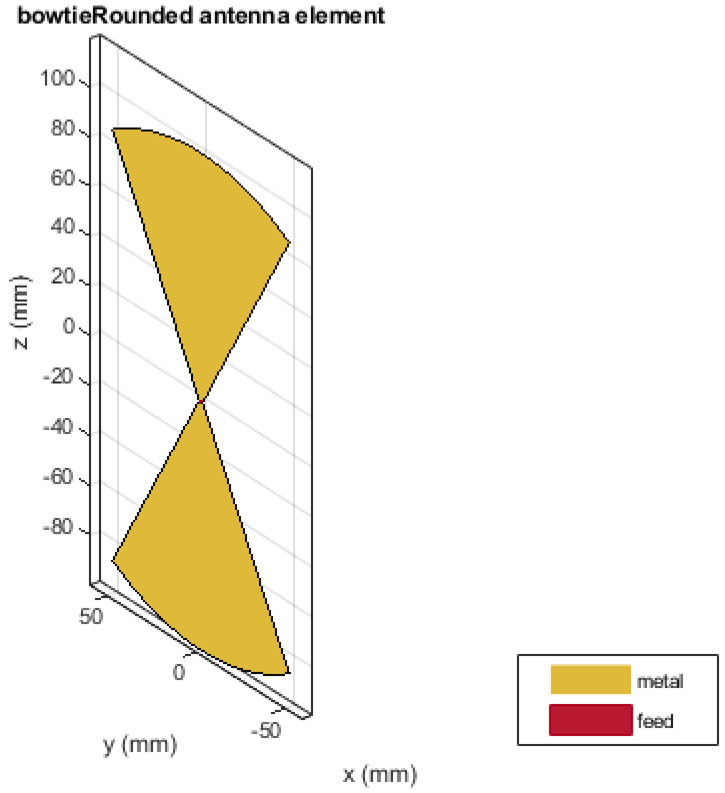
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

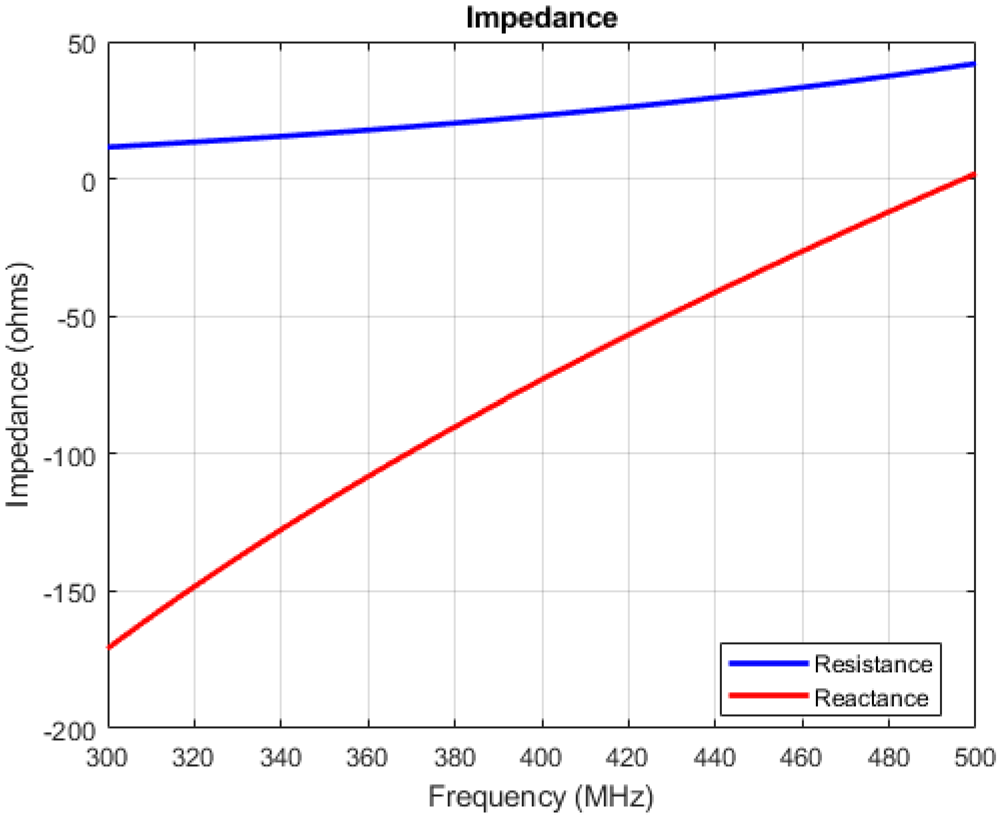
```
b = bowtieRounded('FlareAngle',60);
show(b);
```



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))
```



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”

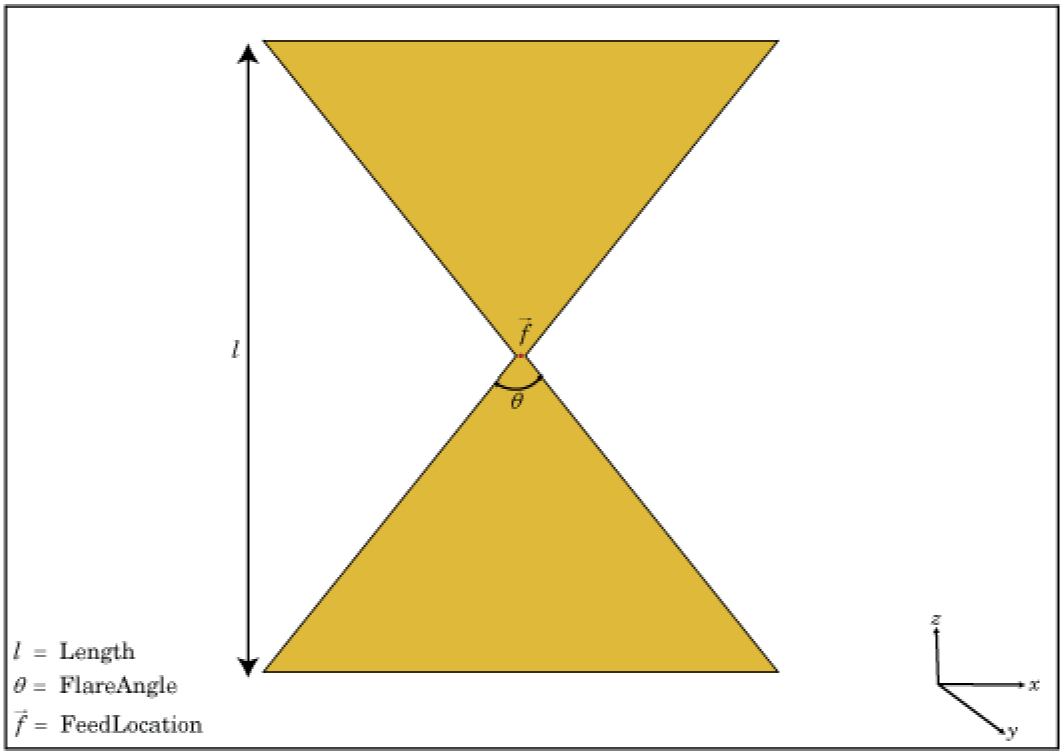
Introduced in R2015a

bowtieTriangular

Create planar bowtie dipole antenna

Description

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

`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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

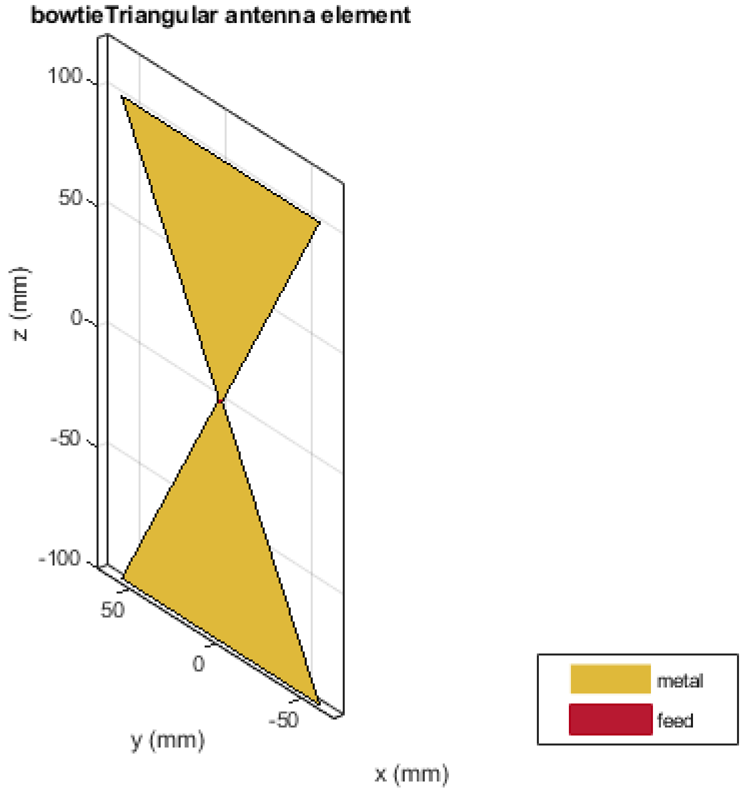
Create and View Center-Fed Planar Bowtie Antenna

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

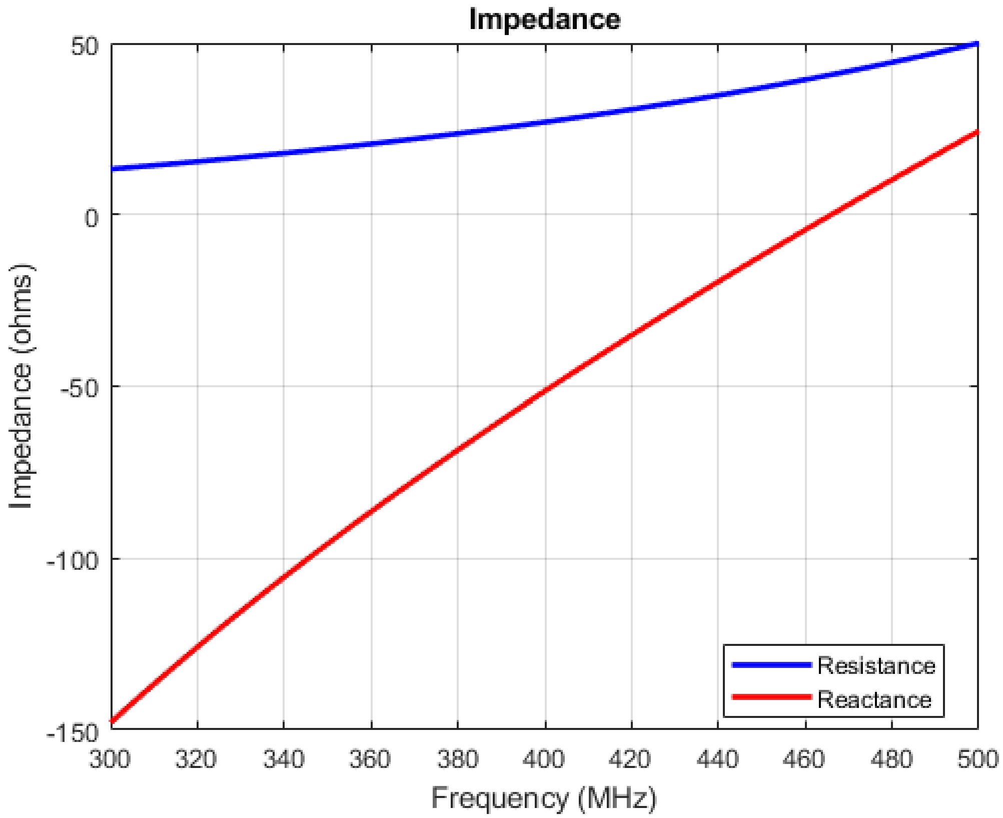
```
show(b)
```



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))
```



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”

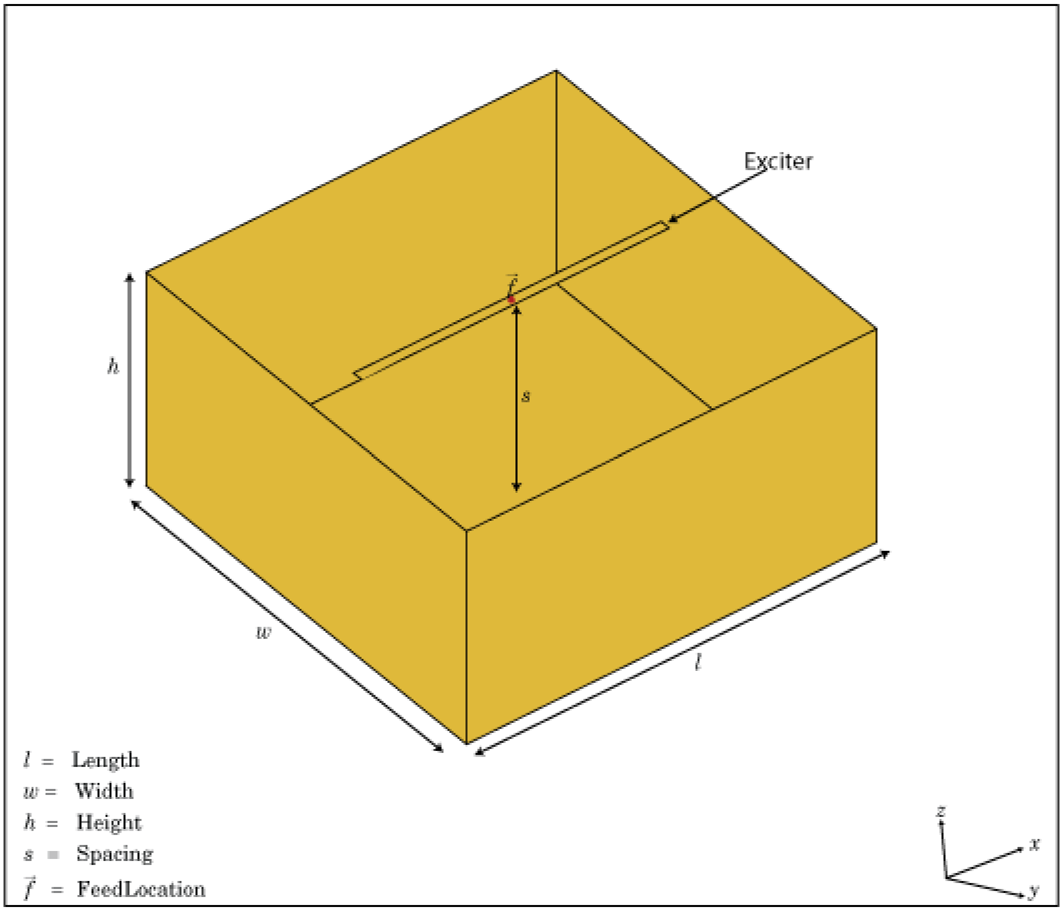
Introduced in R2015a

cavity

Create cavity-backed antenna

Description

The `cavity` object is a cavity-backed antenna located on the X-Y-Z 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 type used as exciter

`dipole` (default) | object

Antenna type used as an exciter, specified as an object. Except reflector and cavity antenna elements, you can use all the single elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',dipole`

Data Types: `char` | `string`

`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'); 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

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: 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: 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 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 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 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'`

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

Object Functions

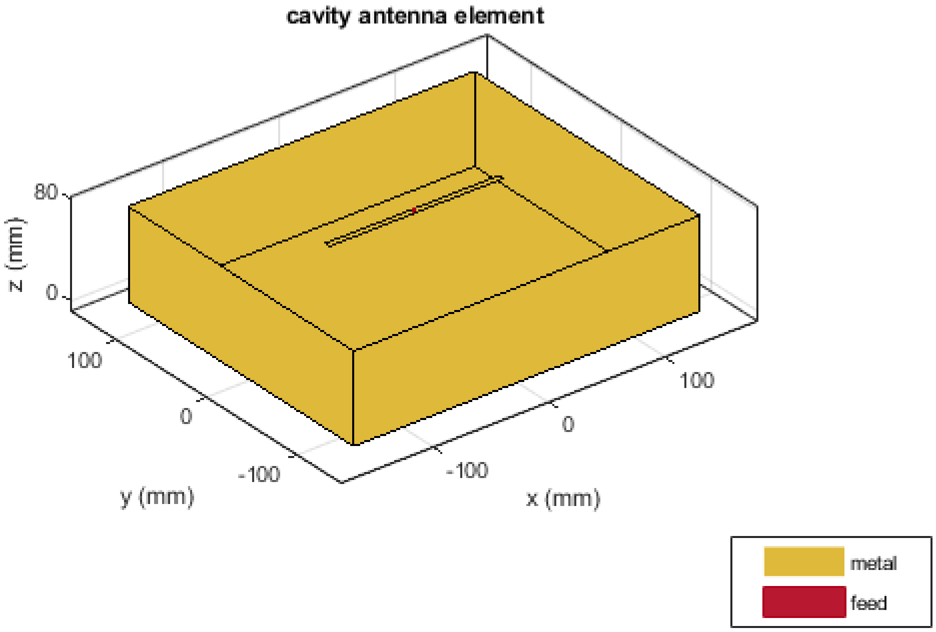
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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)
```



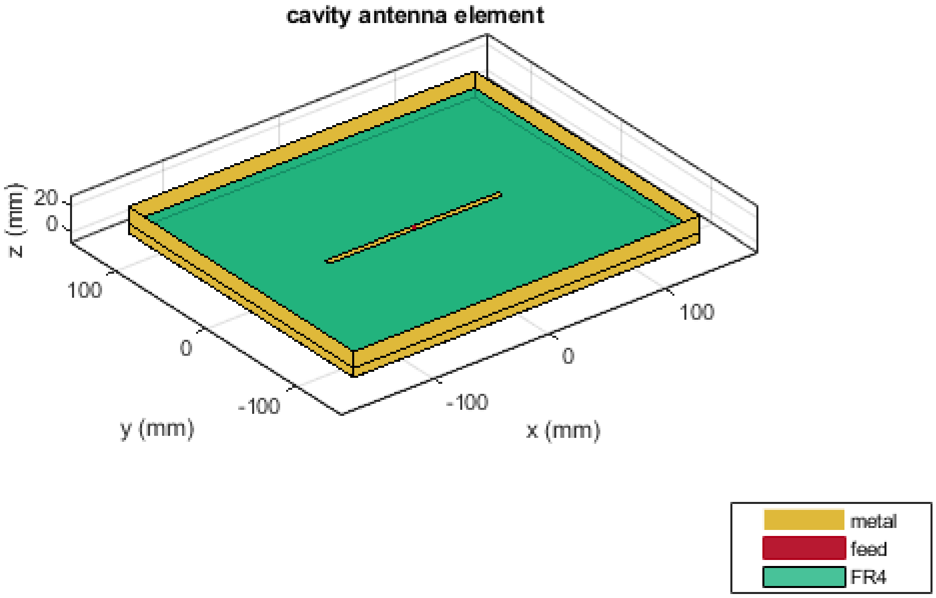
Radiation Pattern of Cavity-Backed Antenna

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)
```

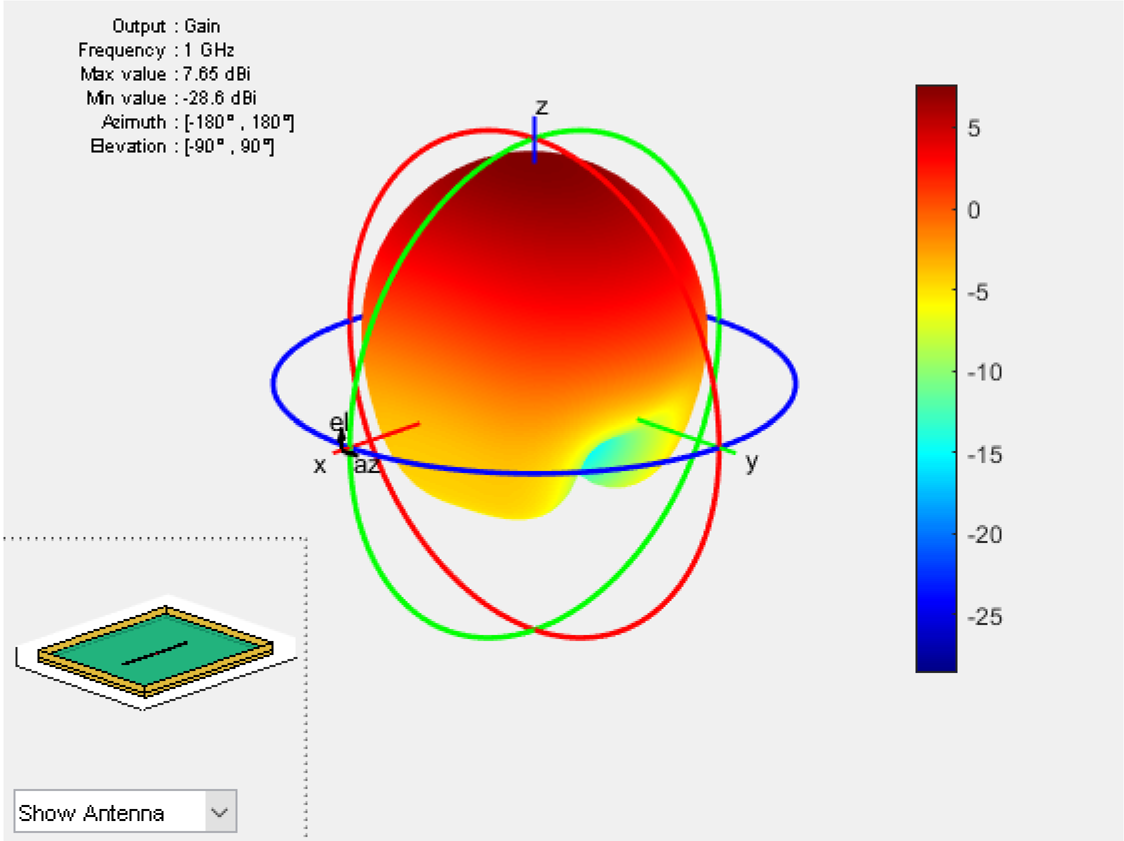
```
c = 
  cavity with properties:

           Exciter: [1x1 dipole]
         Substrate: [1x1 dielectric]
            Length: 0.3000
             Width: 0.2500
            Height: 0.0205
           Spacing: 0.0075
    EnableProbeFeed: 0
              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 : 7.65 dBi
Min value : -28.6 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

References

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

See Also

`reflector` | `spiralArchimedean` | `spiralEquiangular`

Topics

"Rotate Antennas and Arrays"

Introduced in R2015a

dipole

Create strip dipole antenna

Description

The `dipole` object is a strip dipole antenna on the Y-Z plane.

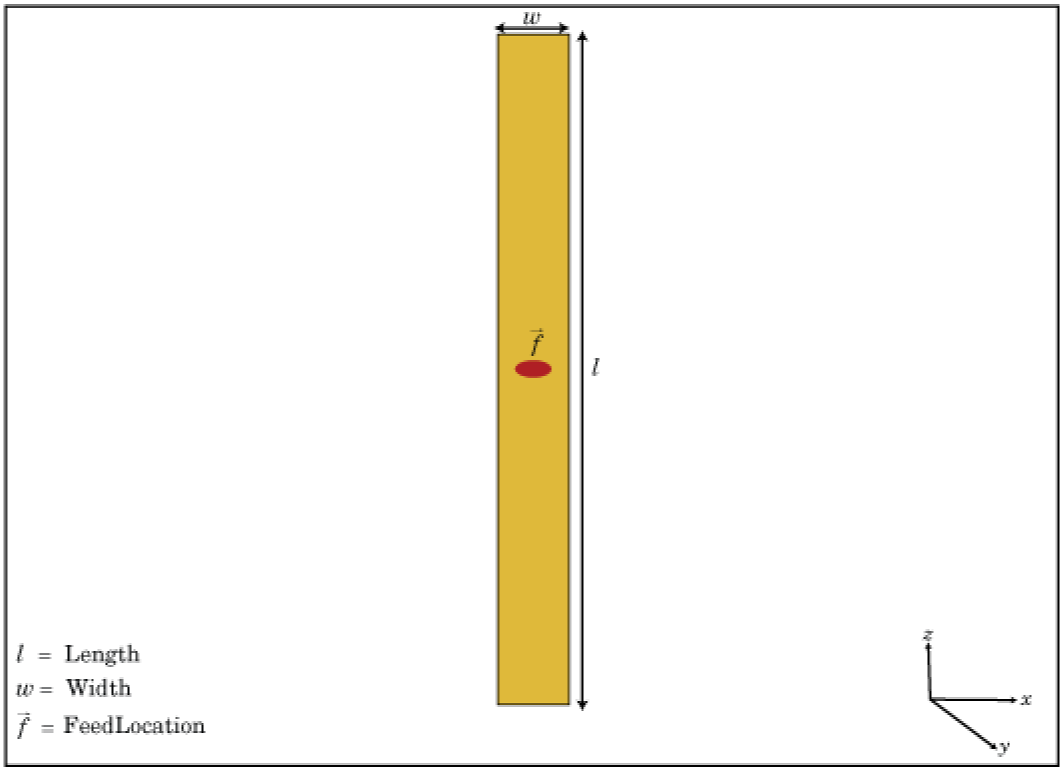
The width of the dipole is related to the diameter of an equivalent cylindrical dipole by the equation

$$w = 2d = 4r$$

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 Y-Z 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 Y-Z plane.

Example: `'FeedOffset',3`

Data Types: `double`

`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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

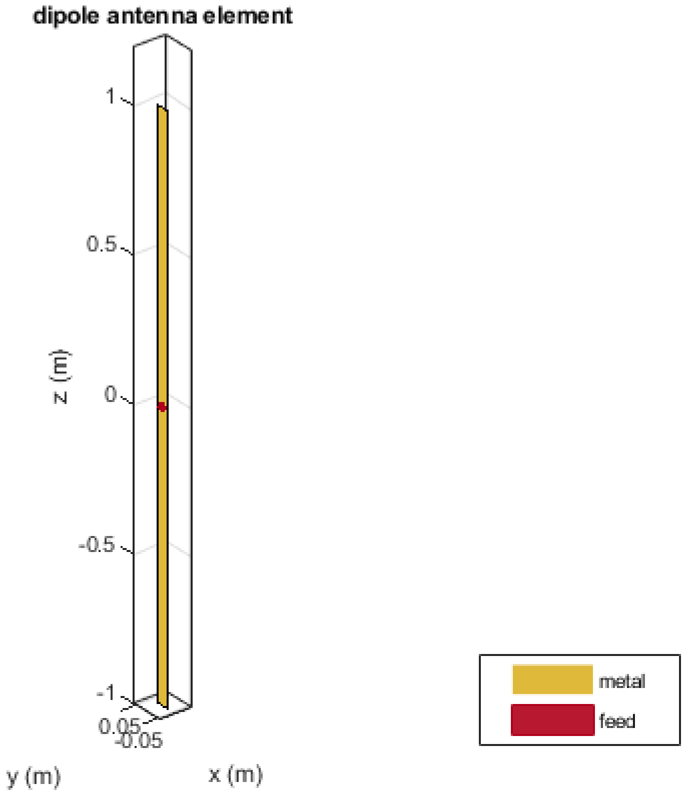
Create and View Dipole Antenna

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

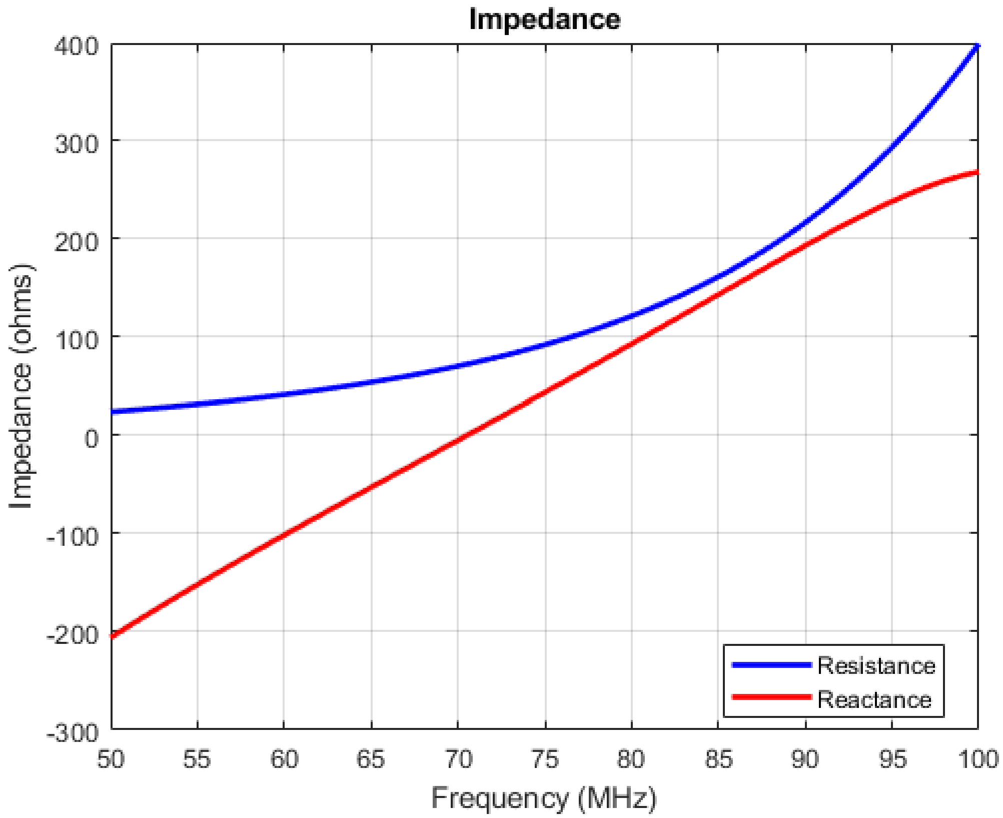
```
show(d)
```



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.

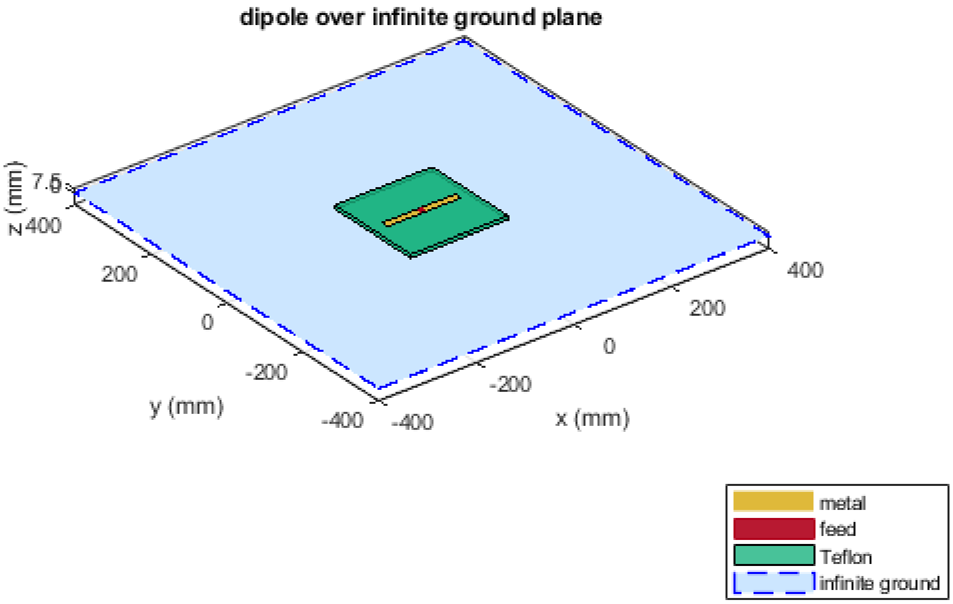
```
t = dielectric('Teflon')
```

```
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);
```

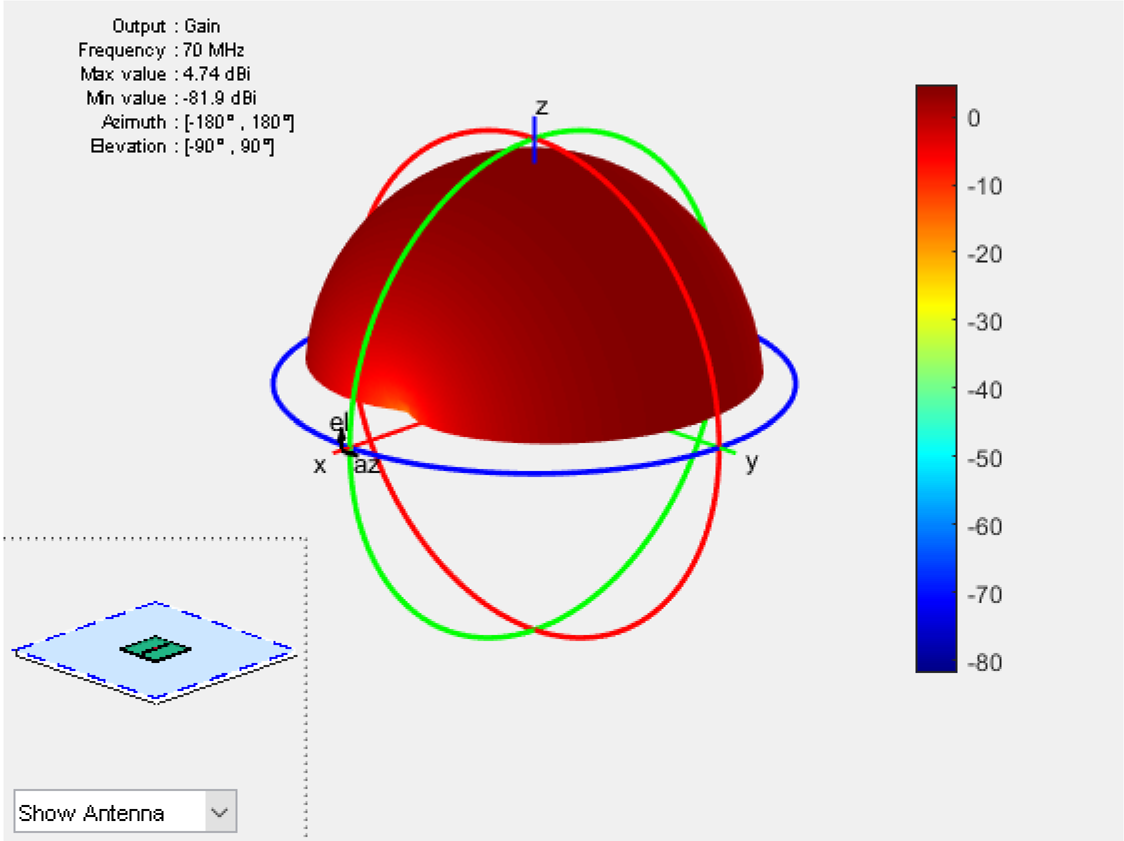
Set the groundplane length of the reflector to `inf`. View the structure.

```
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 : 4.74 dBi
 Min value : -81.9 dBi
   Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

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

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

Topics

“Rotate Antennas and Arrays”

Introduced in R2015a

dipoleFolded

Create folded dipole antenna

Description

The `dipolefolded` object is a folded dipole antenna on the X-Y plane.

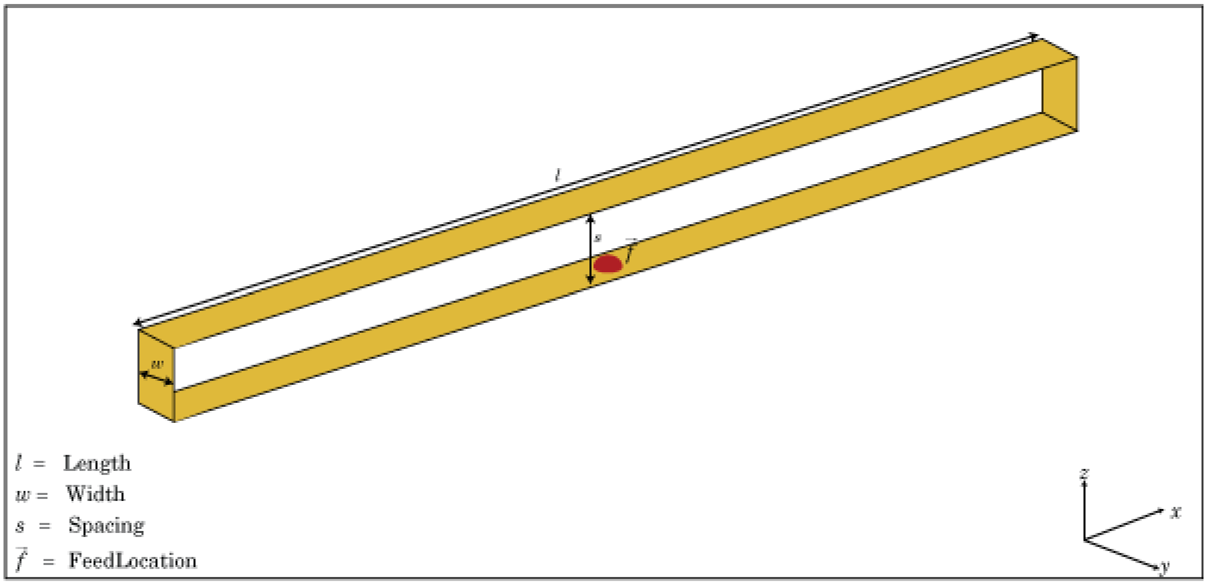
The width of the dipole is related to the diameter of an equivalent cylindrical dipole by the equation

$$w = 2d = 4r$$

, 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 X-Y 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`

`Load` — Lumped elements

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

Lumped elements added to the antenna feed, specified 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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

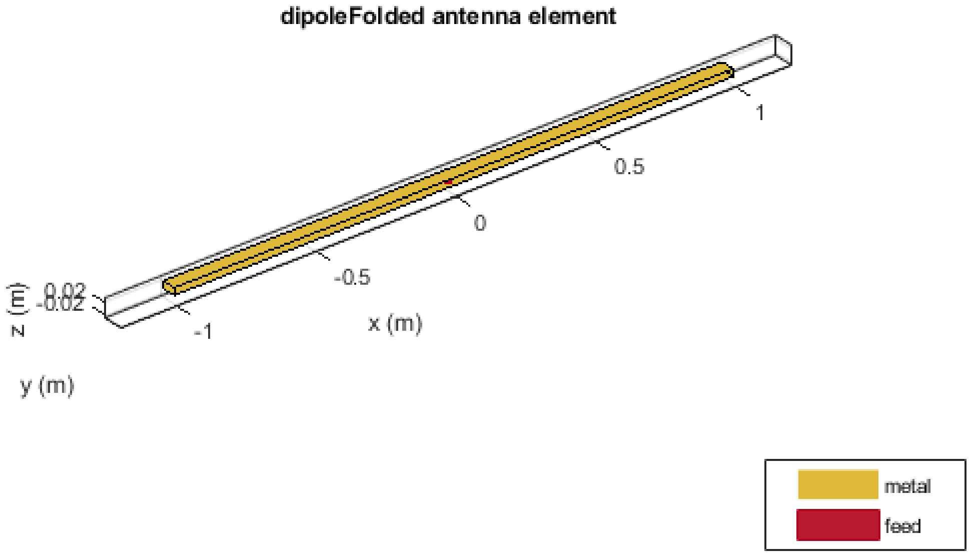
Create and View Folded Dipole Antenna

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

```
show(df)
```



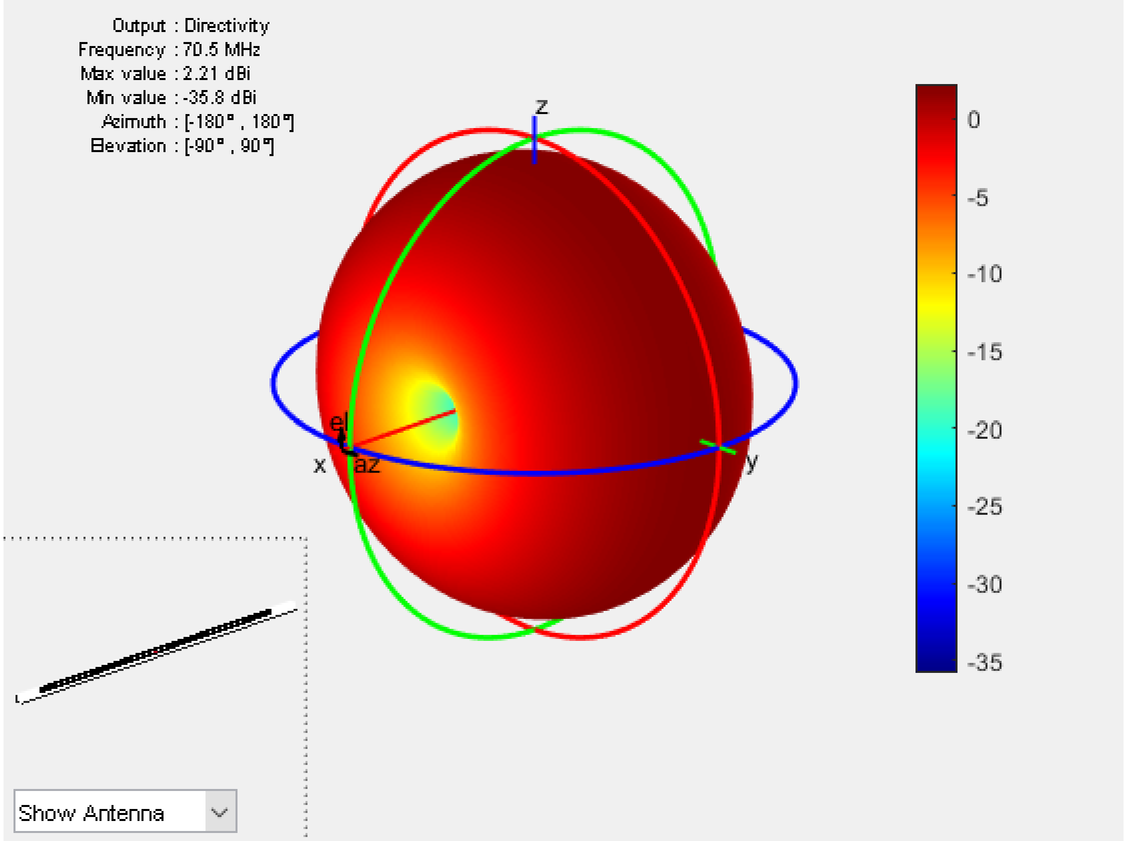
Radiation Pattern of Folded Dipole Antenna

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
        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°]
```

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

Topics

"Rotate Antennas and Arrays"

Introduced in R2015a

dipoleVee

Create V-dipole antenna

Description

The `dipoleVee` object is a planar V-dipole antenna in the X-Y plane.

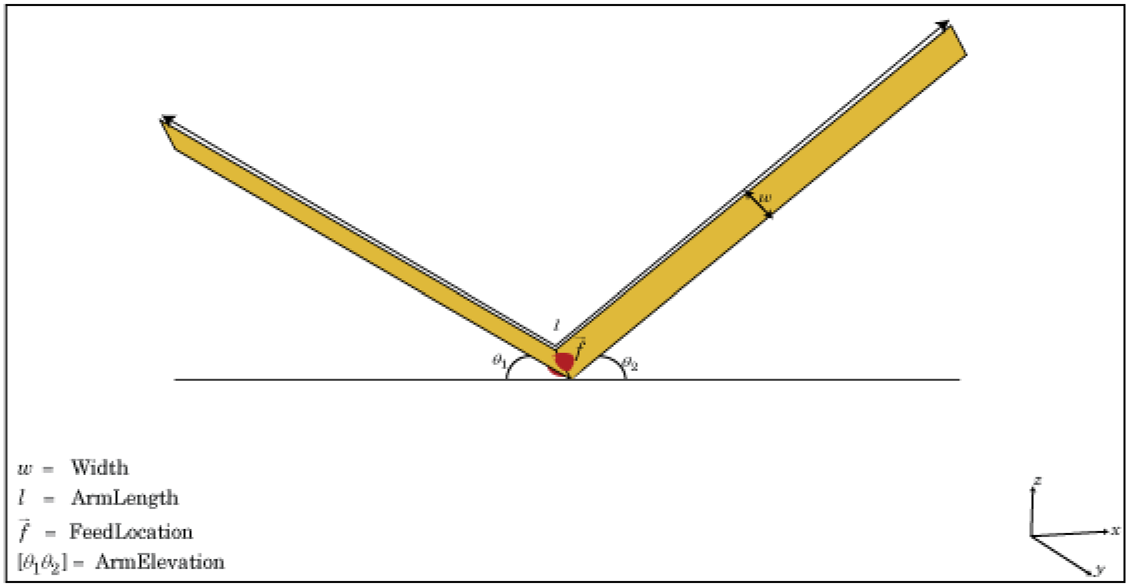
The width of the dipole is related to the circular cross-section by the equation

$$w = 2d = 4r$$

, 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 X-Y 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 X-Y plane

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

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

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

Data Types: `double`

`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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

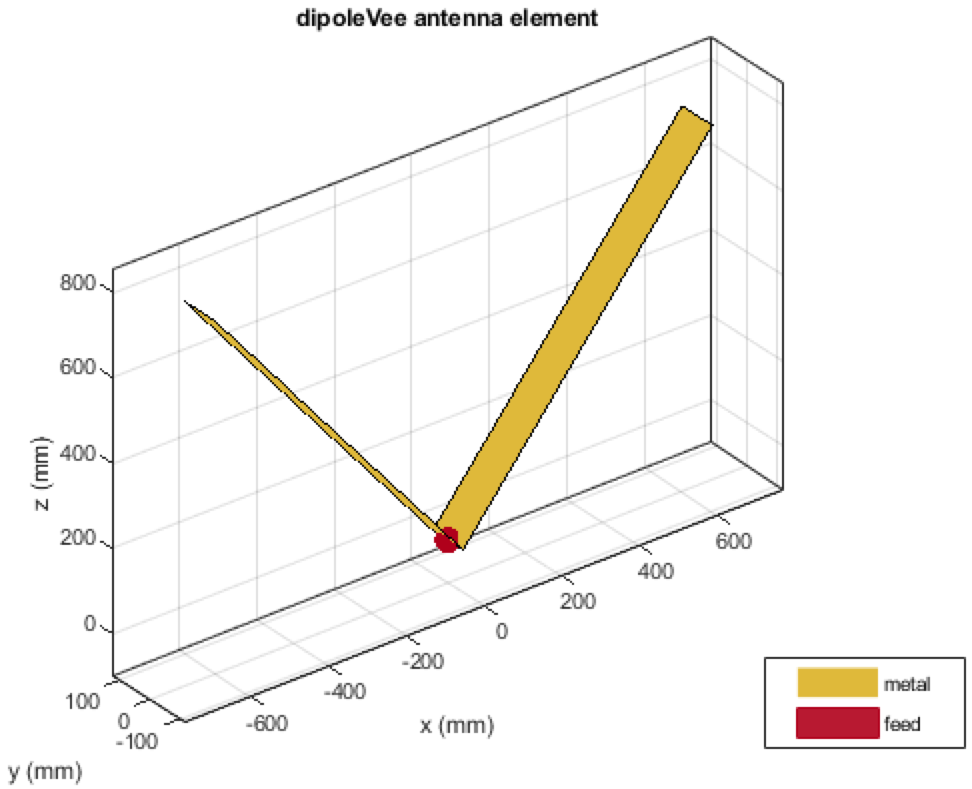
Create V-Dipole Antenna

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

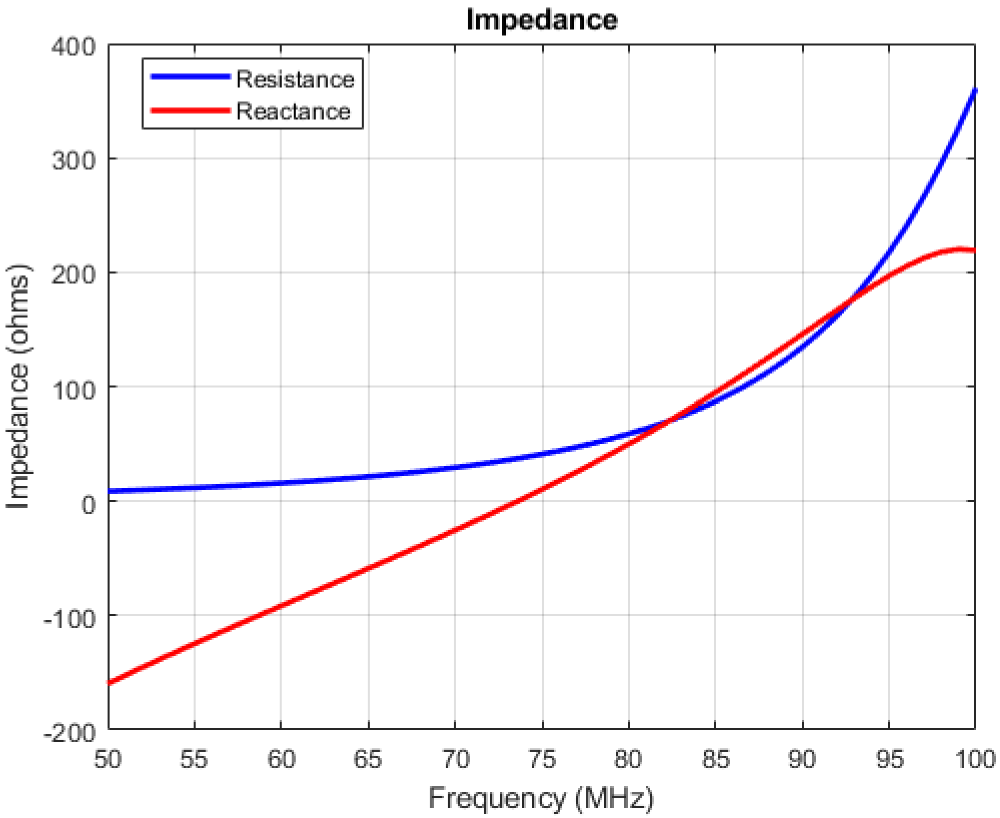
```
show(dv)
```



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))
```



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

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

Topics

"Rotate Antennas and Arrays"

Introduced in R2015a

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.

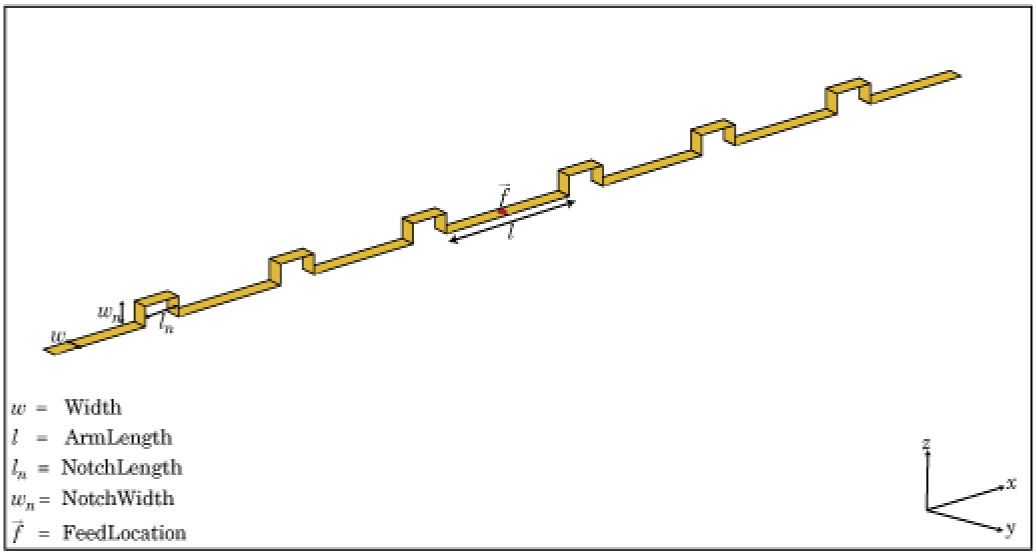
The width of the dipole is related to the diameter of an equivalent cylindrical dipole by the equation

$$w = 2d = 4r$$

, 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 X-Y 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

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: 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: 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 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 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 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'

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

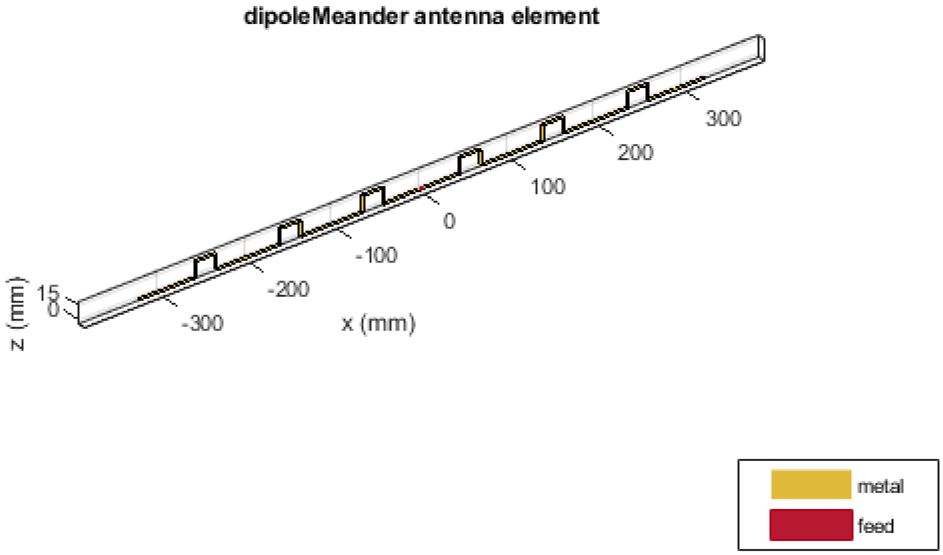
Create and View Meander Dipole Antenna

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

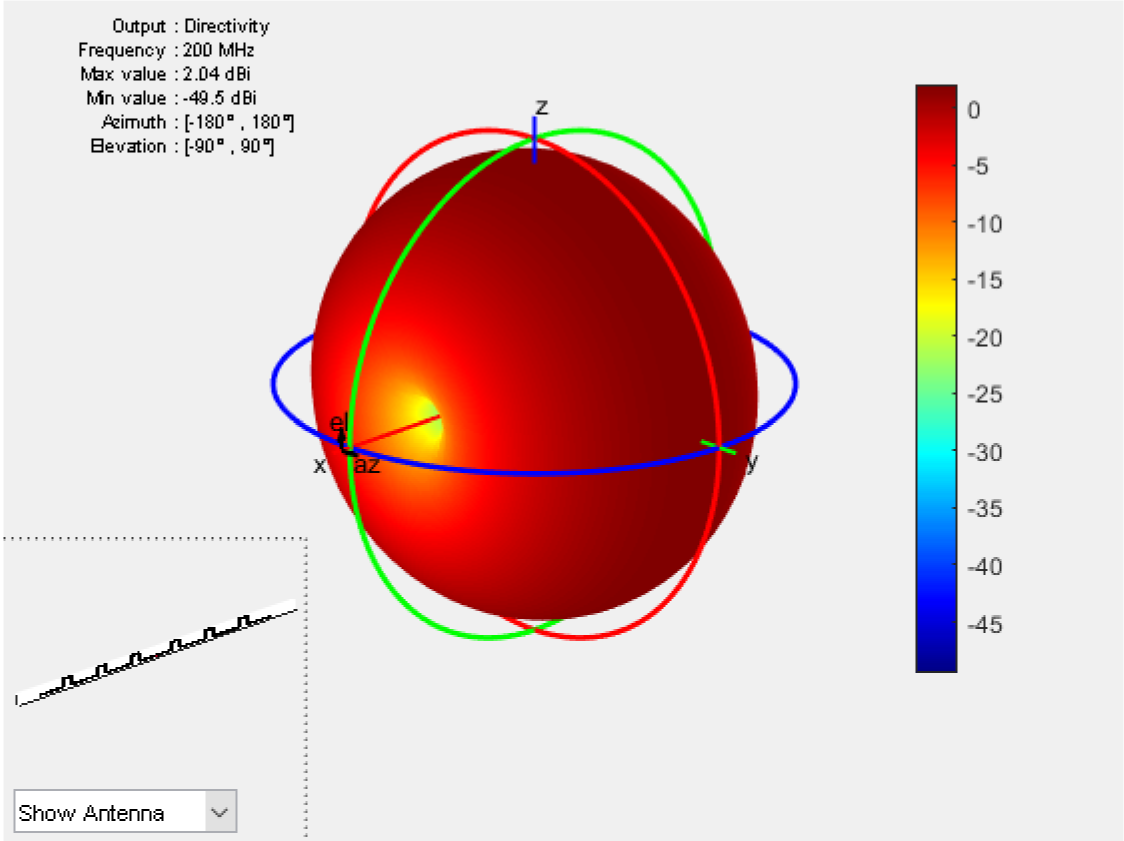
```
show(dm)
```



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°]

References

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

See Also

`dipole` | `dipoleFolded` | `loopCircular`

Topics

“Rotate Antennas and Arrays”

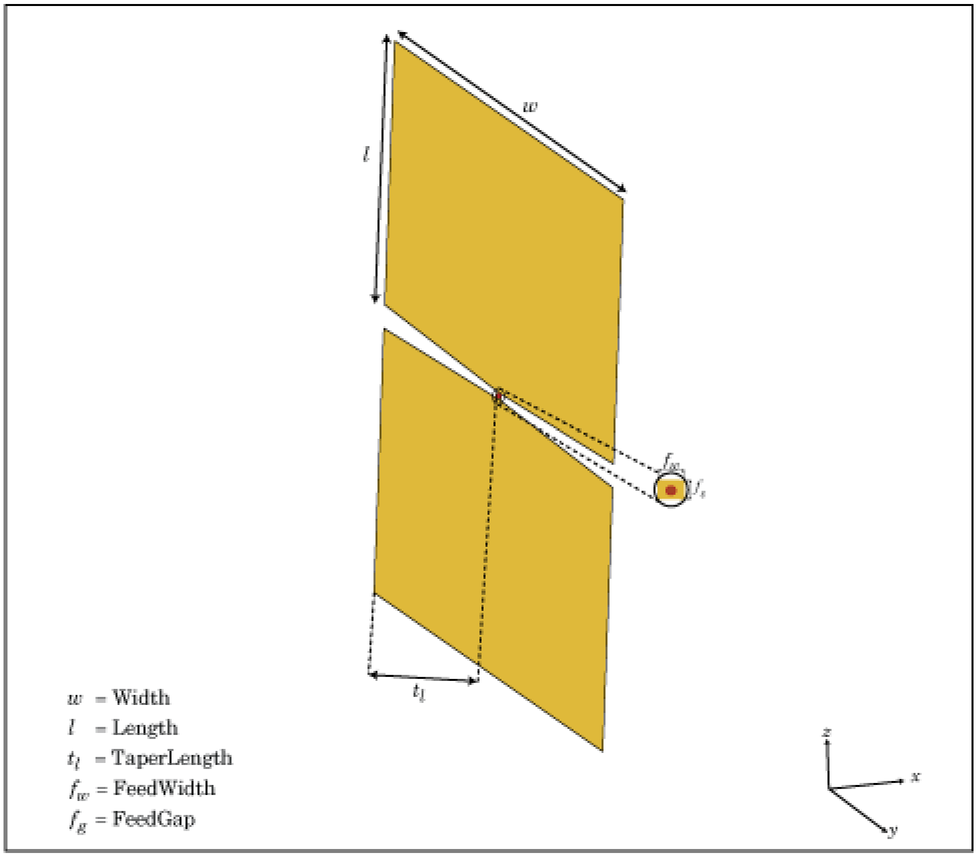
Introduced in R2015a

dipoleBlade

Create blade dipole antenna

Description

The `dipoleBlade` object is a wideband blade dipole antenna oriented along the X-Y 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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

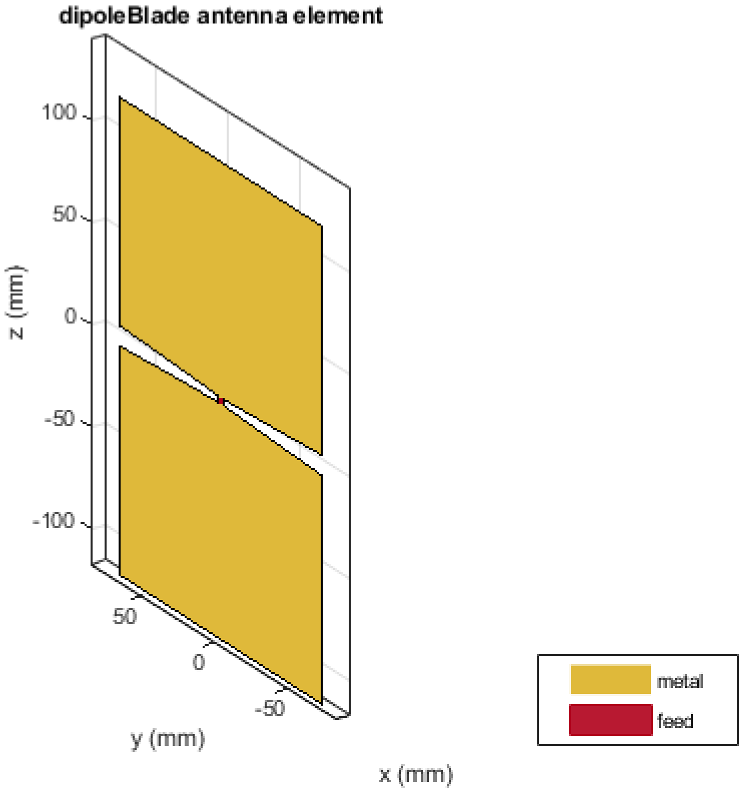
Default Blade Dipole and Radiation Pattern

Create and view a default blade dipole.

```
db = dipoleBlade
```

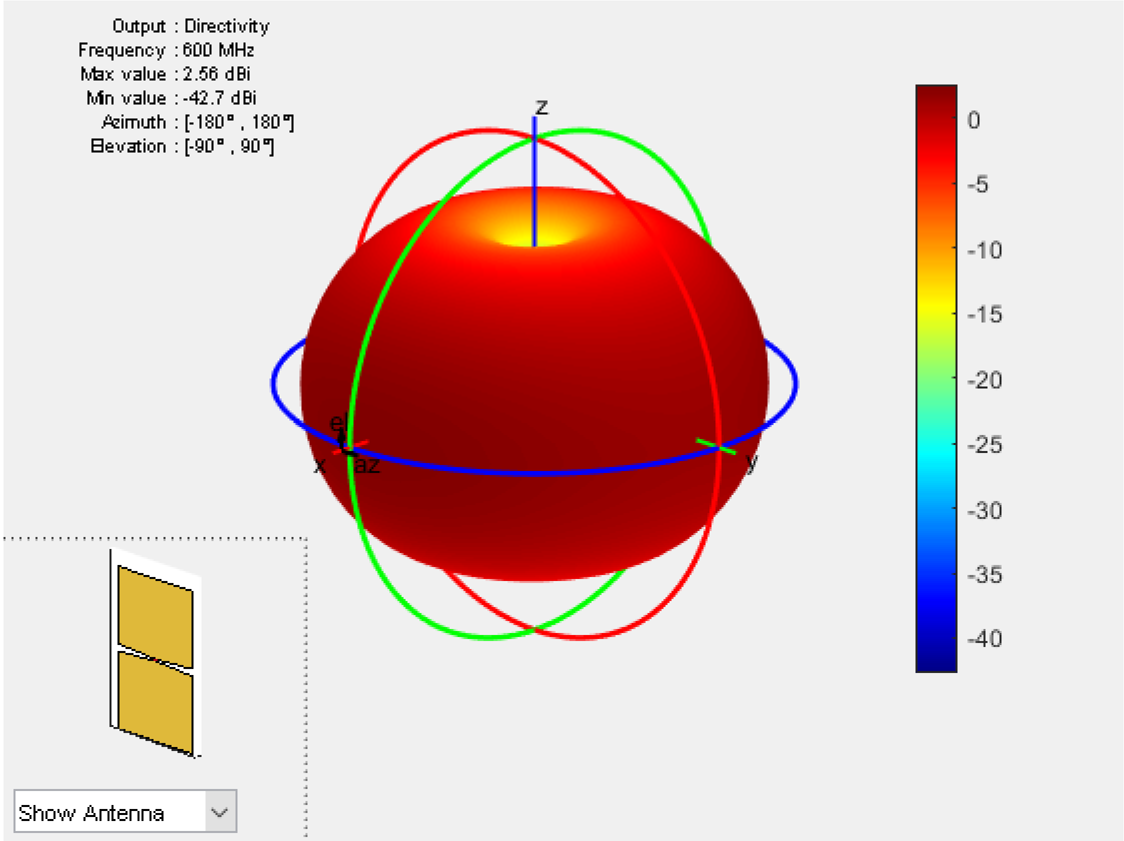
```
db = 
  dipoleBlade with properties:

         Length: 0.1170
    TaperLength: 0.1120
          Width: 0.1400
      FeedWidth: 0.0030
        FeedGap: 0.0030
           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 : -42.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

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

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

Topics

“Rotate Antennas and Arrays”

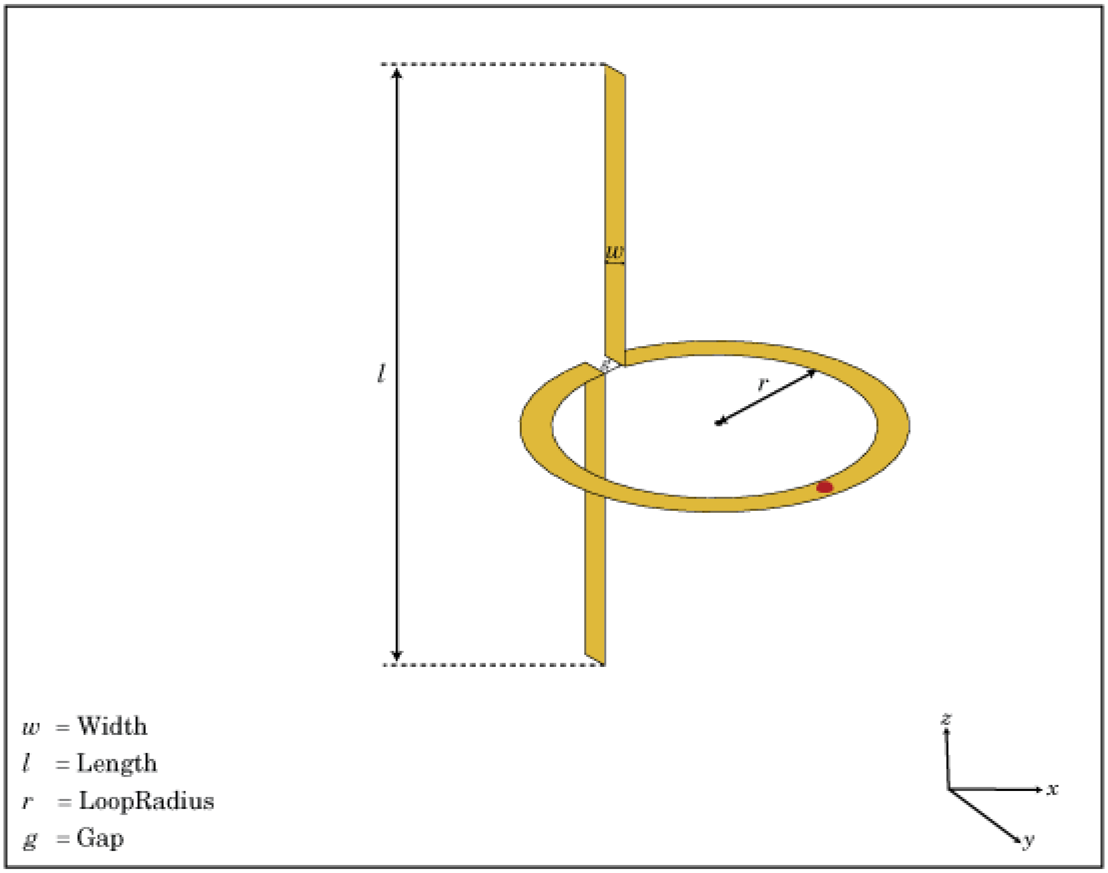
Introduced in R2017a

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 X-Y plane
`0.3100` (default) | scalar

Circular loop radius in X-Y plane, specified as a scalar in meters.

Example: `'LoopRadius',0.500`

Data Types: `double`

`Gap` — Gap of loop in X-Y plane
`0.0400` (default) | scalar

Gap of loop in X-Y plane, specified as a scalar in meters.

Example: `'Gap',0.006`

Data Types: `double`

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

Lumped elements added to the antenna feed, specified as the comma-separated pair consisting of `'Load'` and 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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface

current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

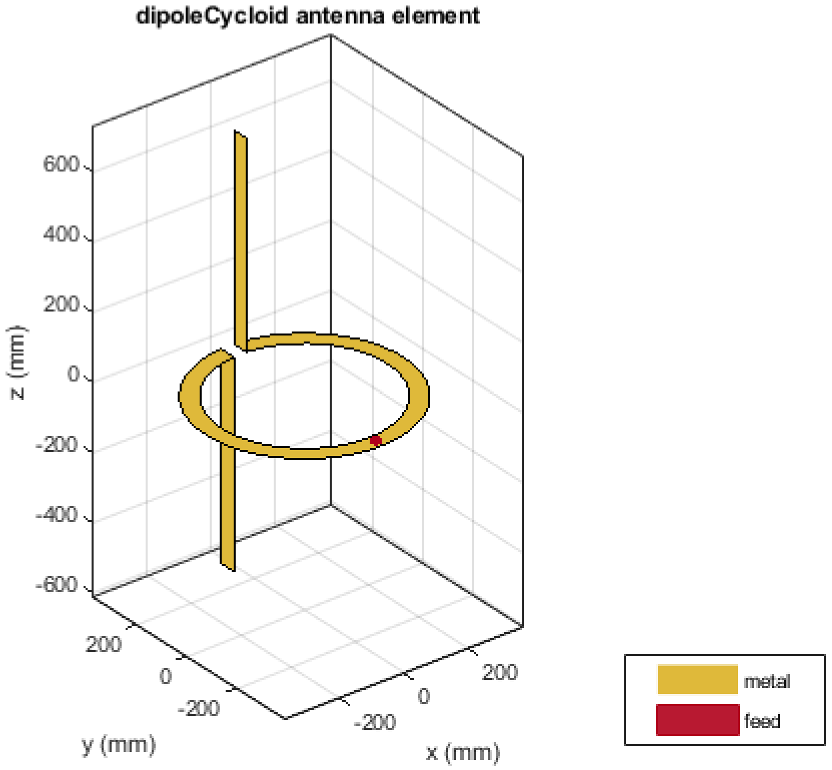
Default Cycloid Dipole

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

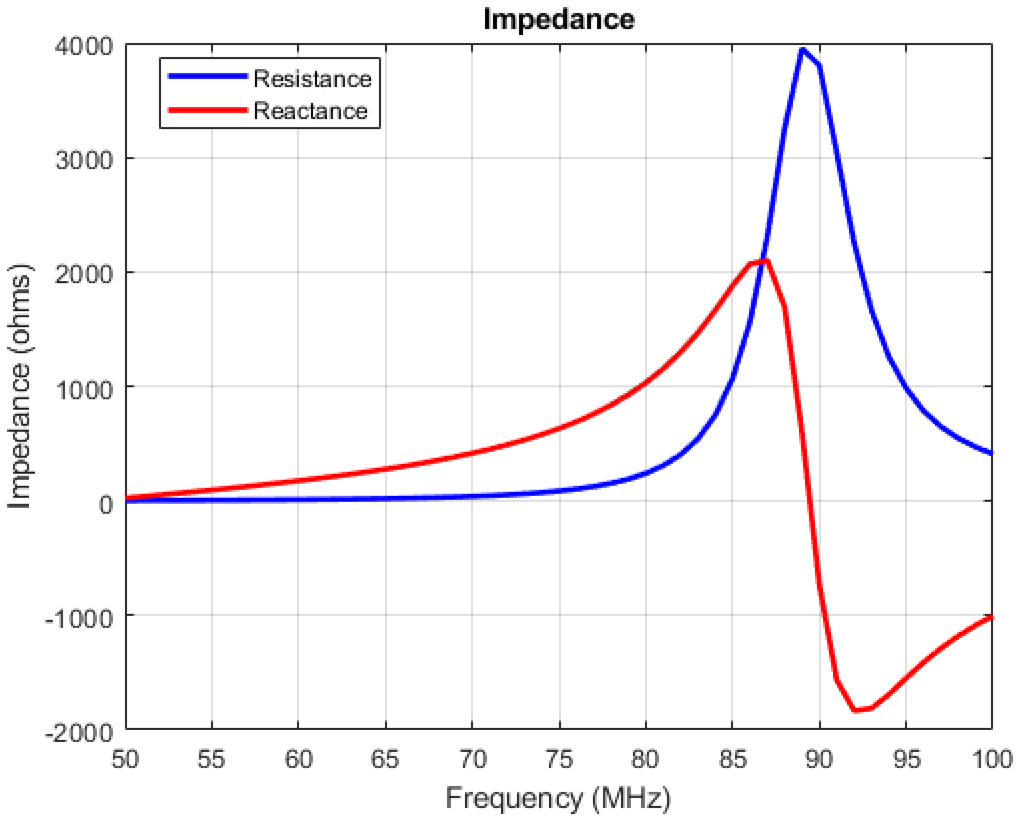
```
show(dc)
```



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))
```



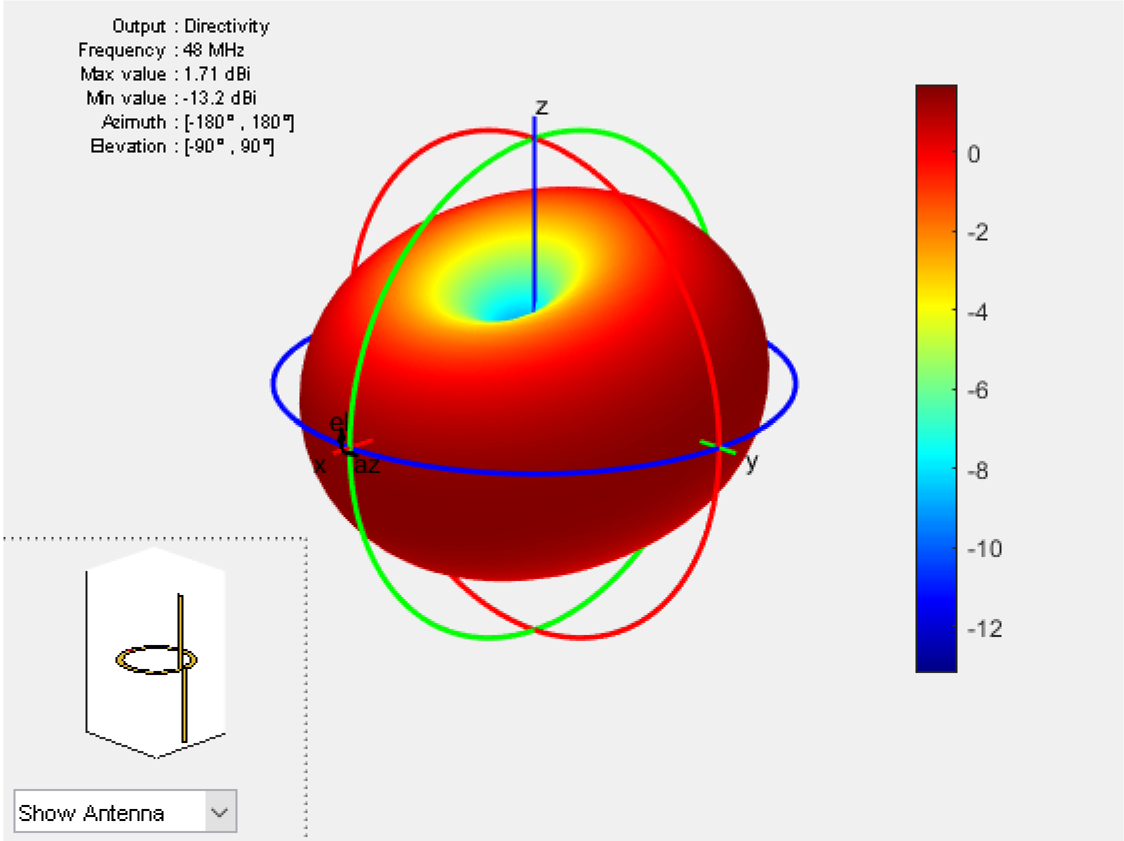
Radiation Pattern of Cycloid Dipole

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

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



```
    Output : Directivity
 Frequency : 48 MHz
 Max value : 1.71 dBi
 Min value : -13.2 dBi
   Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

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

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

Topics

“Rotate Antennas and Arrays”

Introduced in R2017a

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. The circular ground plane is on the X-Y plane. 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:

$x = r\cos(\theta)$

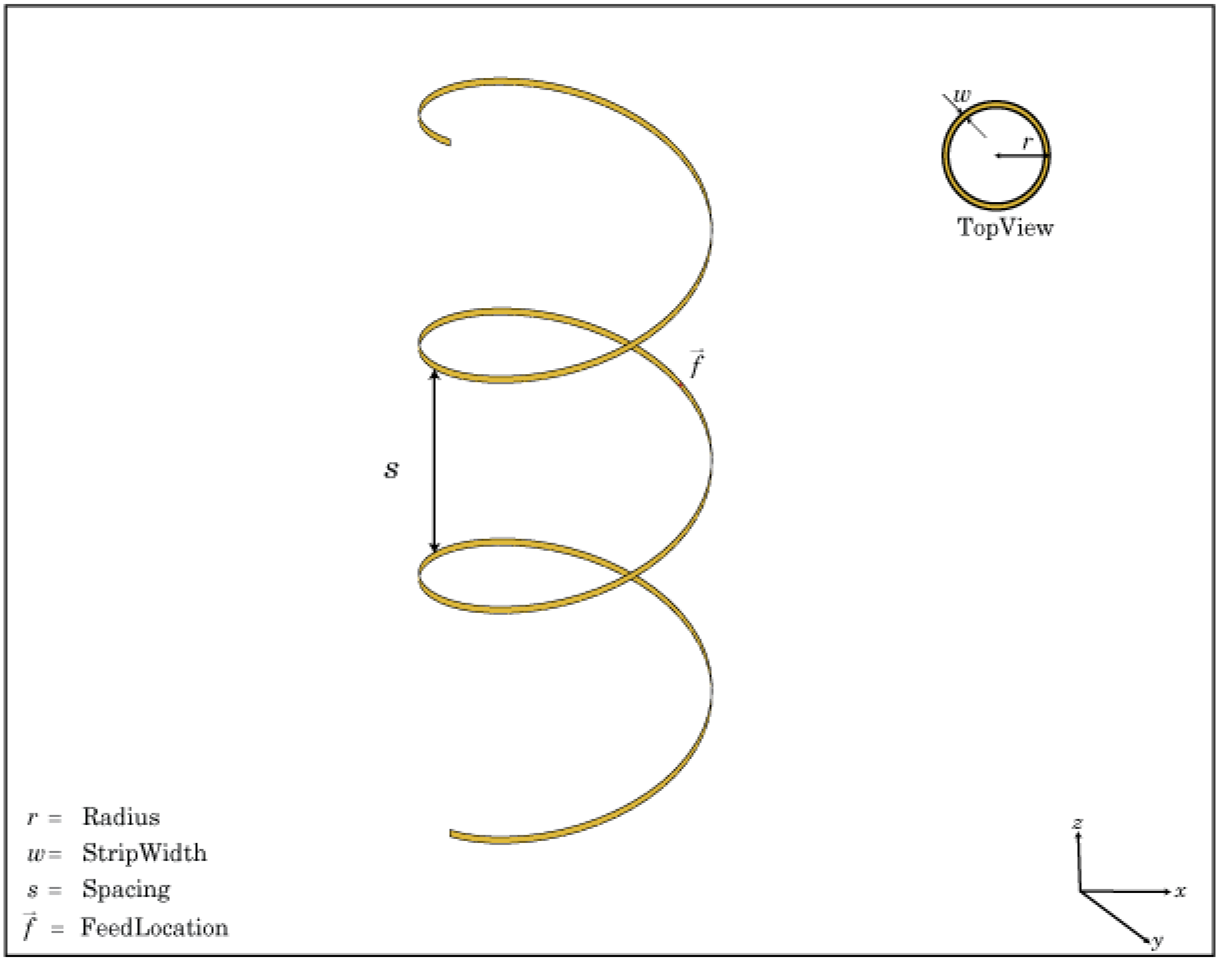
$y = r\sin(\theta)$

$z = S\theta$

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 around 2 GHz.

`dh = dipoleHelix(Name,Value)` creates a helical 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

`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`

`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: `dh.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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface

current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

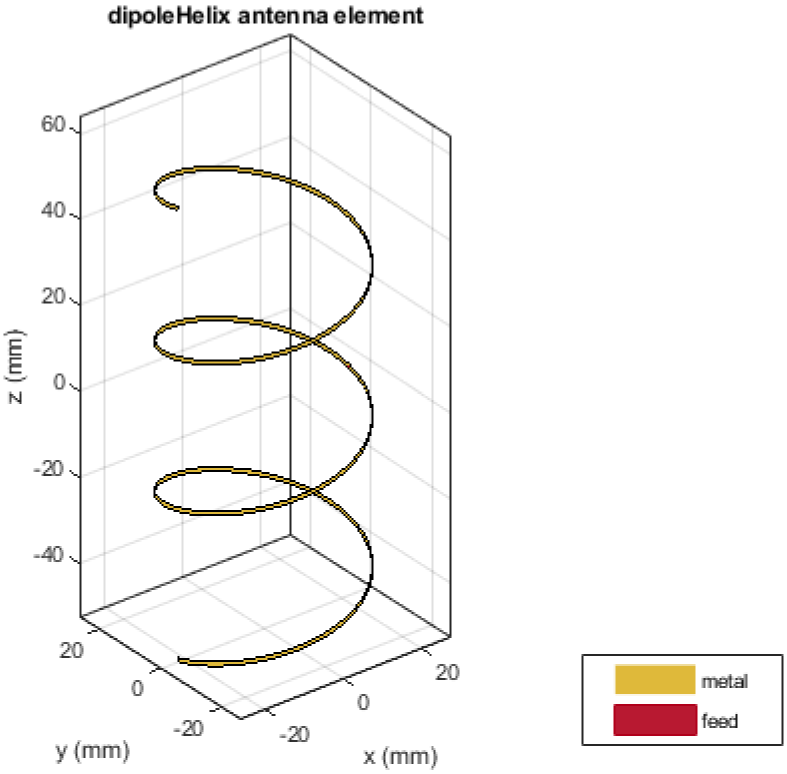
Helical Dipole Antenna

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

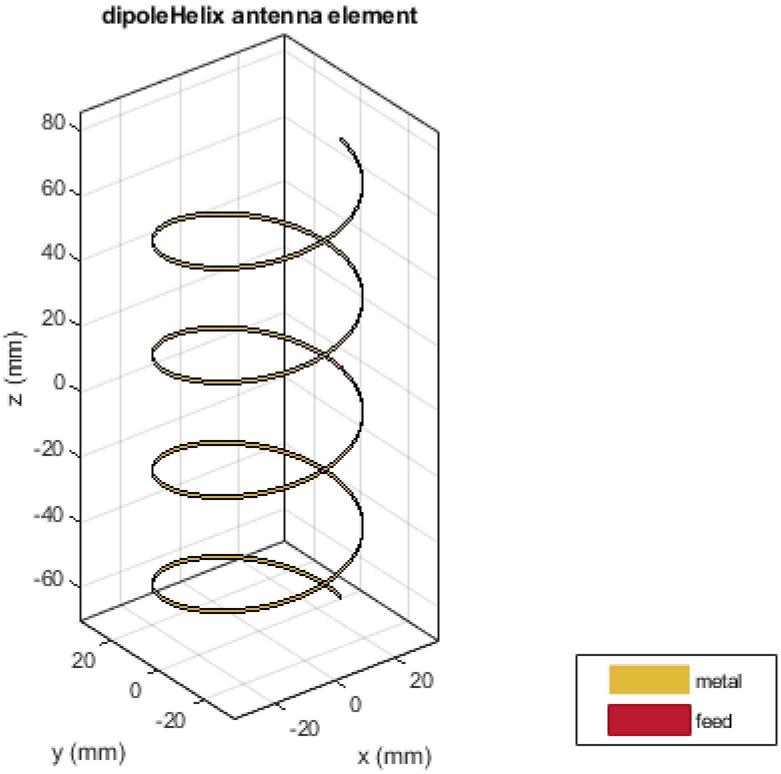
```
show(dh)
```



Radiation Pattern of Helical Dipole

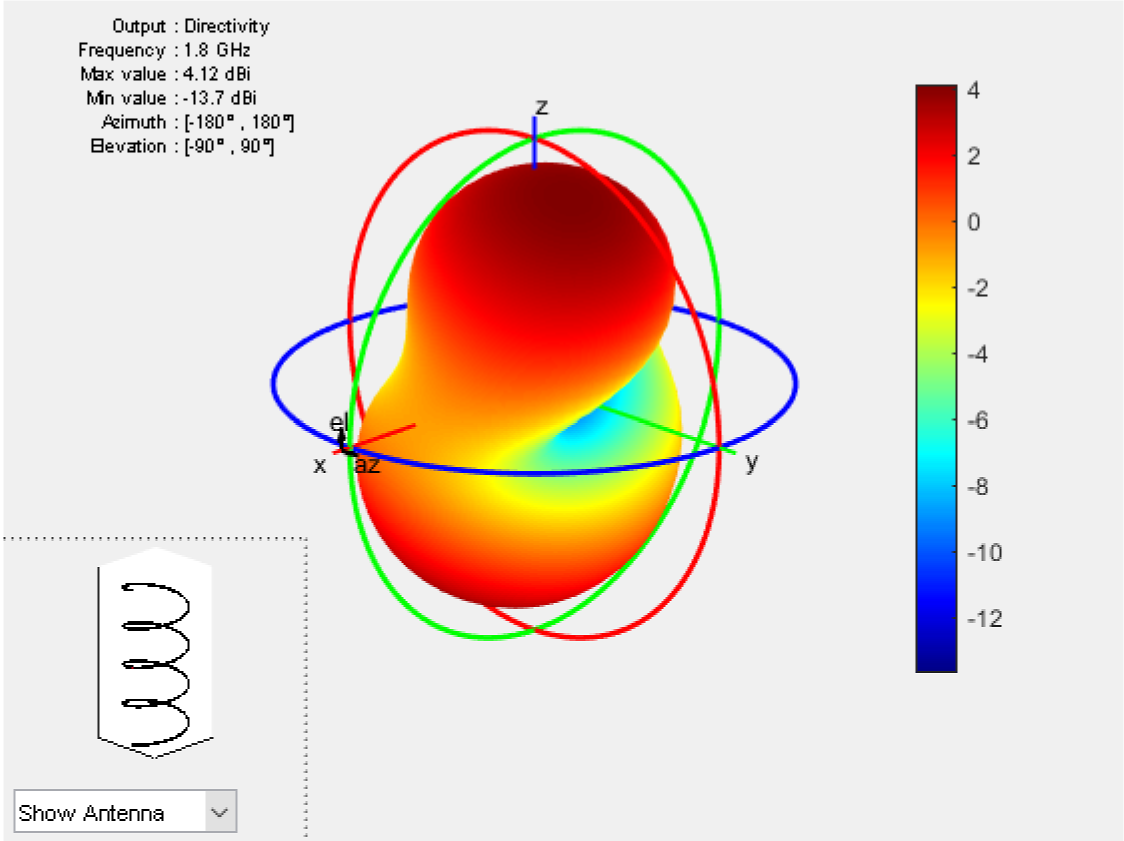
Create a four-turn helical dipole antenna with a turn radius of 28 mm and a strip width of 1.2 mm.

```
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°]

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

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

Topics

"Rotate Antennas and Arrays"

Introduced in R2016b

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.

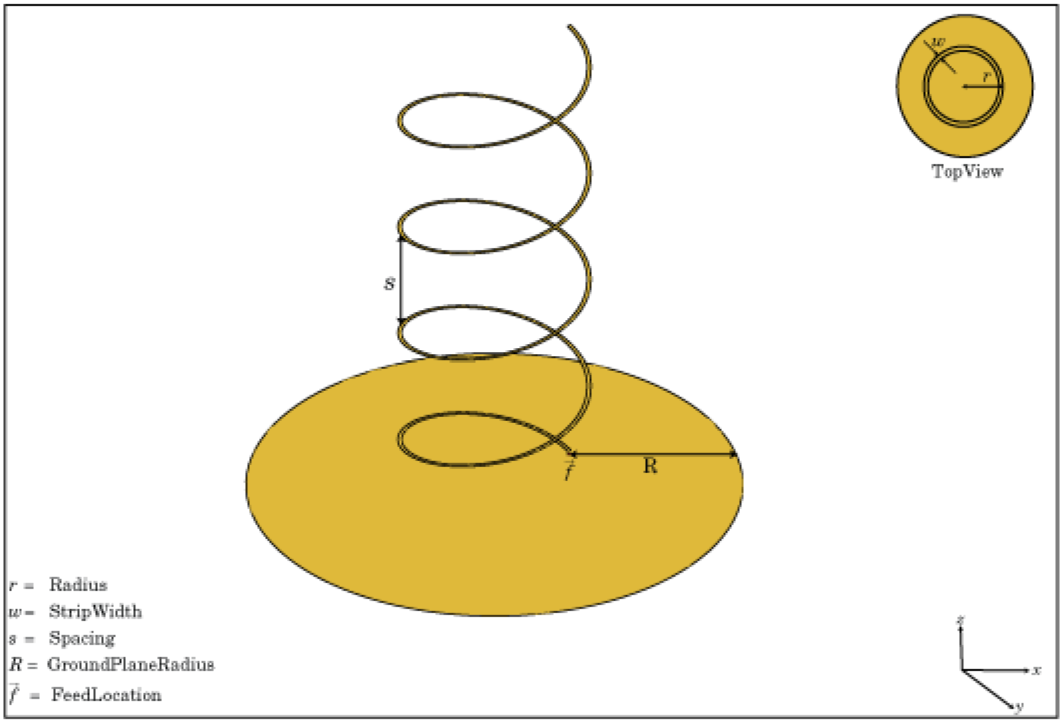
For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default helix antenna is end-fed. The circular ground plane is on the X-Y plane. Commonly, helix antennas are used in axial mode. In this mode, the helix circumference is comparable to the operating 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. B

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

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

Note The default value is chosen to allow backward compatibility.

Data Types: `double`

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 origin. 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)`

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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

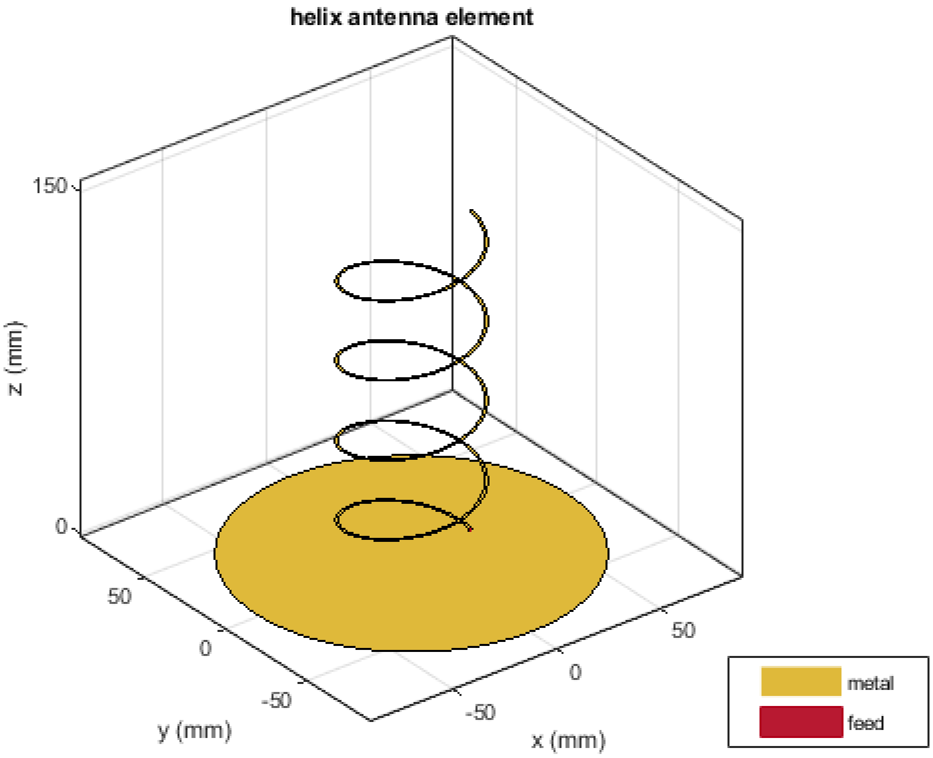
Examples

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

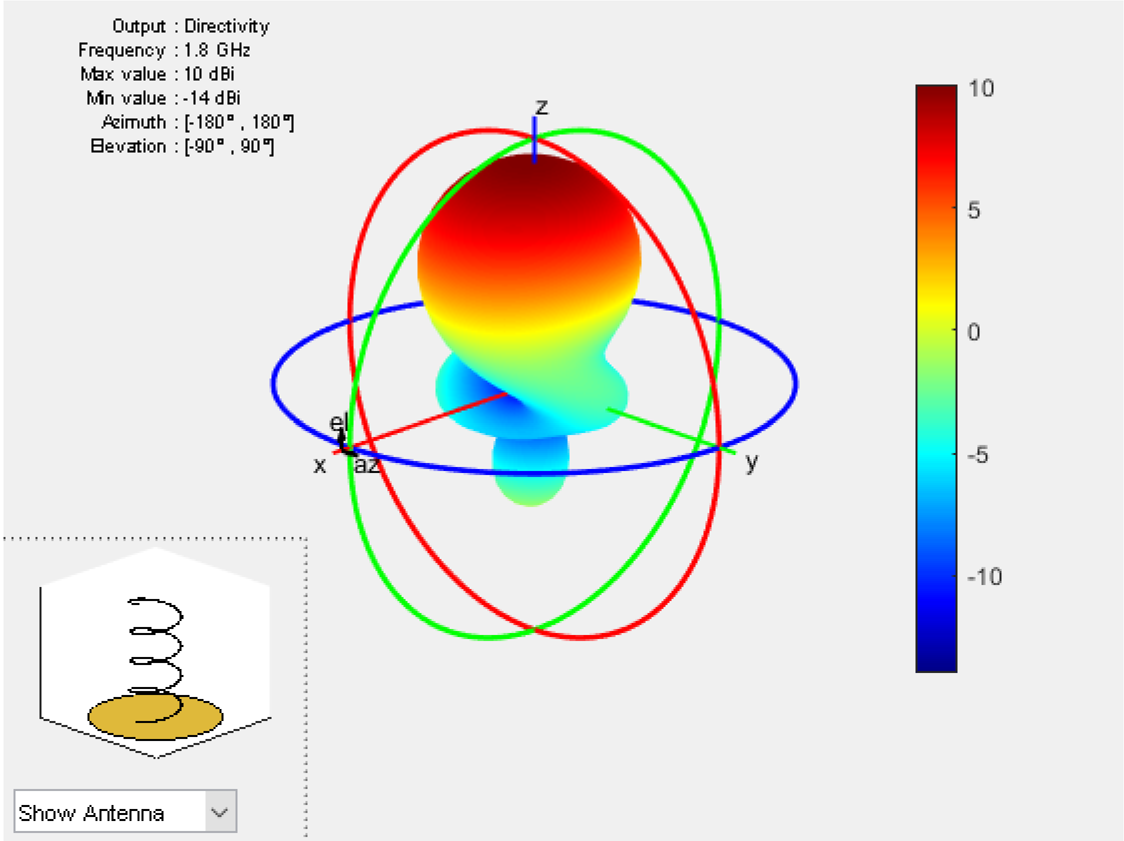
```
show(hx)
```



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

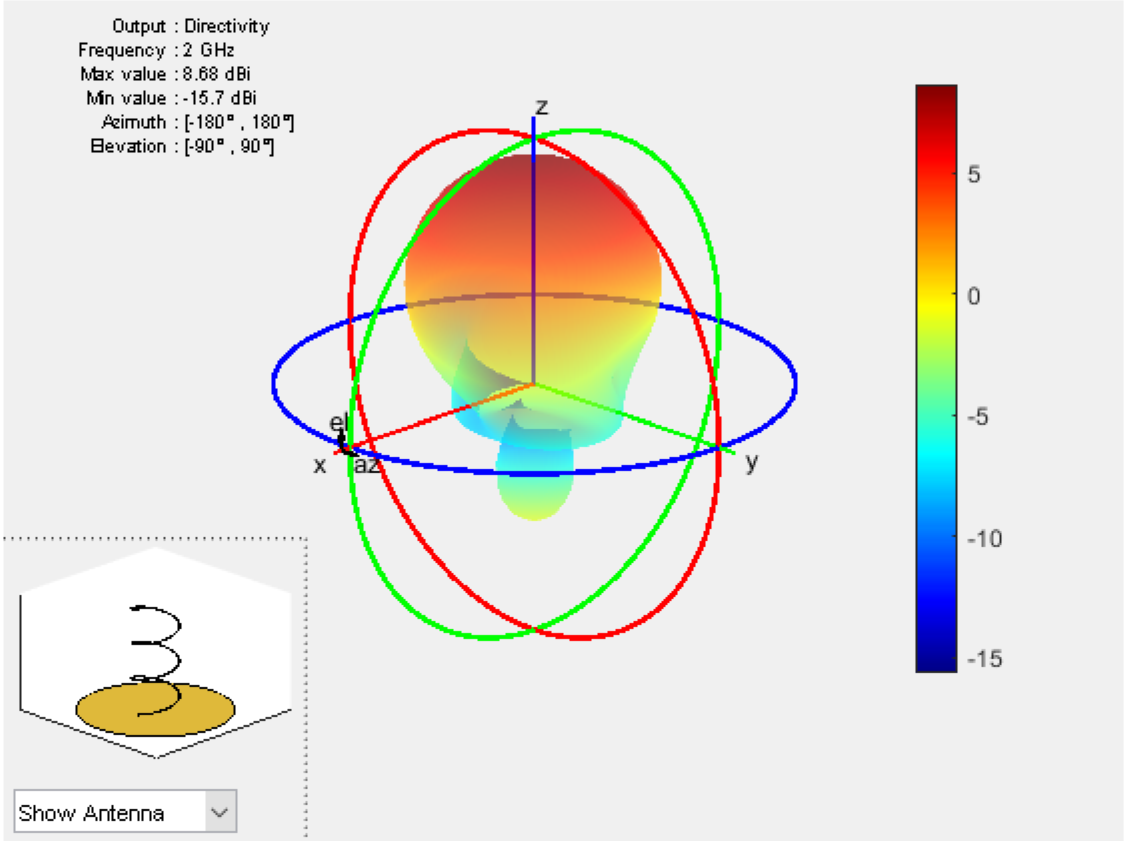
Plot the radiation pattern of a helix antenna with transparency specified as 0.5.

```
p = PatternPlotOptions
```

```
p = 
  PatternPlotOptions with properties:

      Transparency: 1
```

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

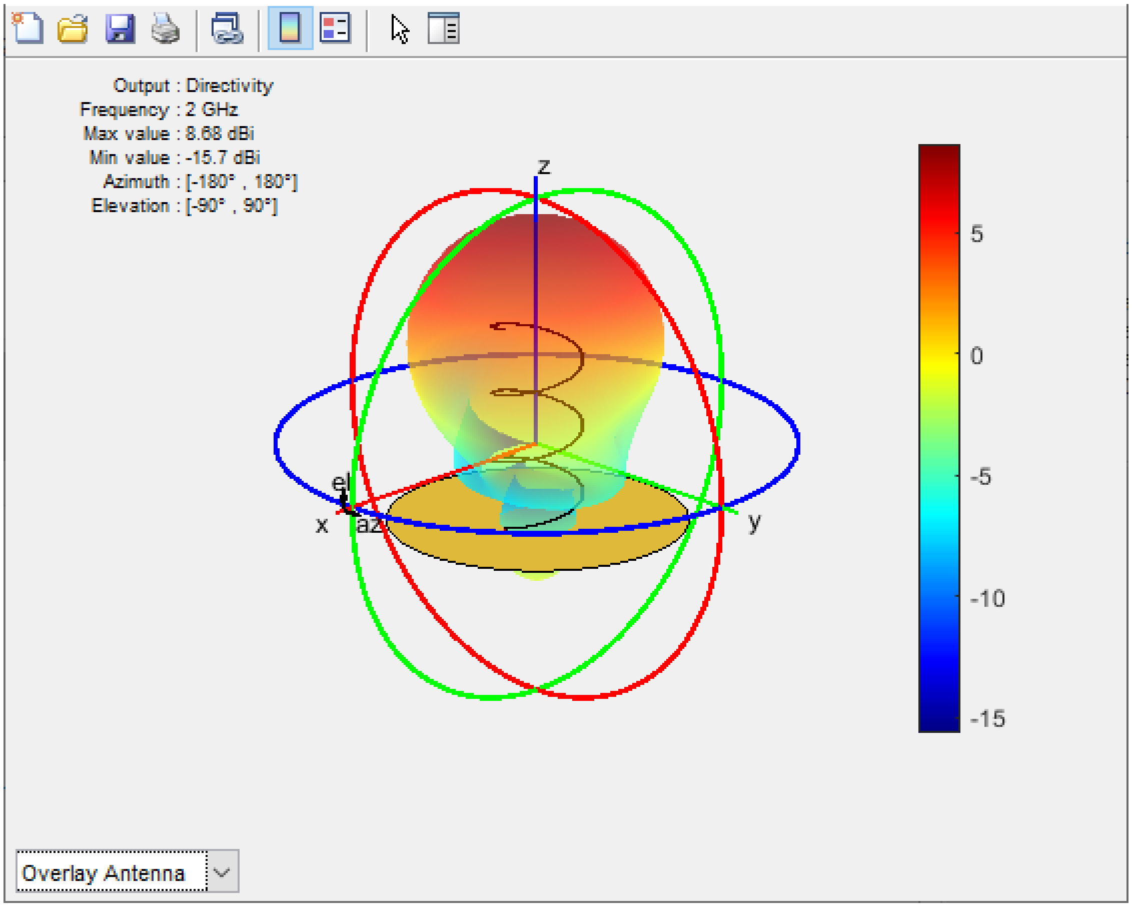
```
p.Transparency = 0.5;
ant = helix;
pattern(ant,2e9,'patternOptions',p)
```



```
   Output : Directivity
Frequency : 2 GHz
Max value : 8.68 dBi
Min value : -15.7 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°]
```

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.

See Also

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

Topics

“Rotate Antennas and Arrays”

Introduced in R2015a

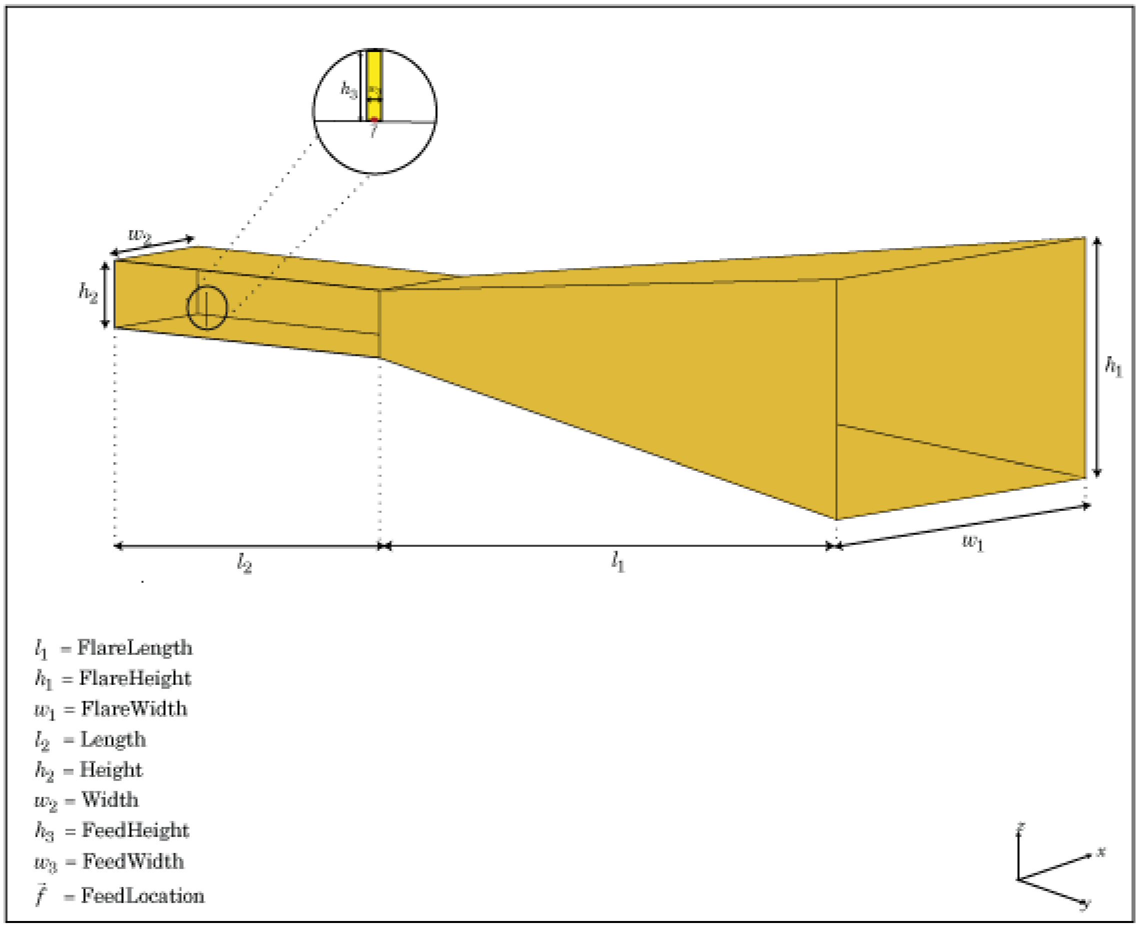
horn

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`

`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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

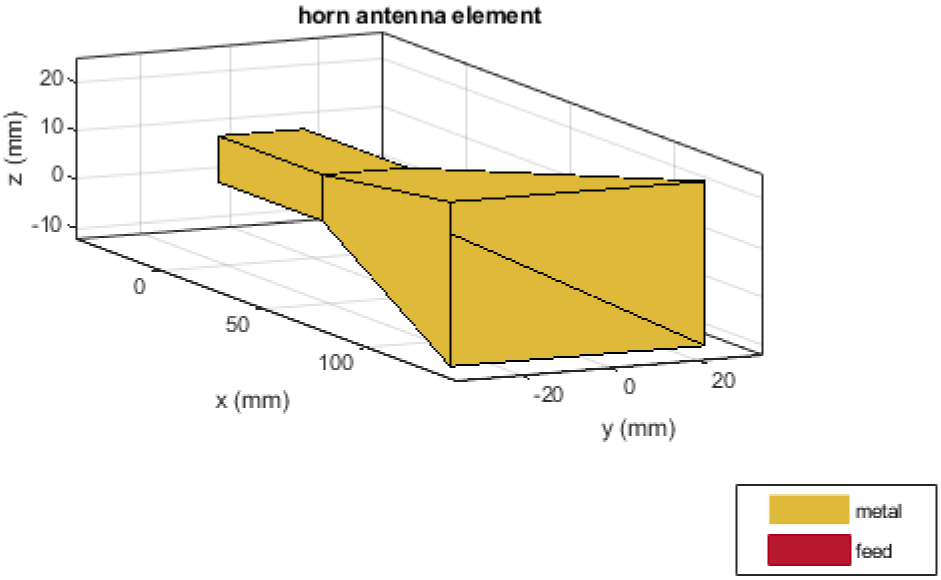
Default Horn Antenna

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

```
show(h)
```



References

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

See Also

`hornangle2size` | `waveguide`

Topics

"Rotate Antennas and Arrays"

Introduced in R2016a

invertedF

Create inverted-F antenna over rectangular ground plane

Description

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

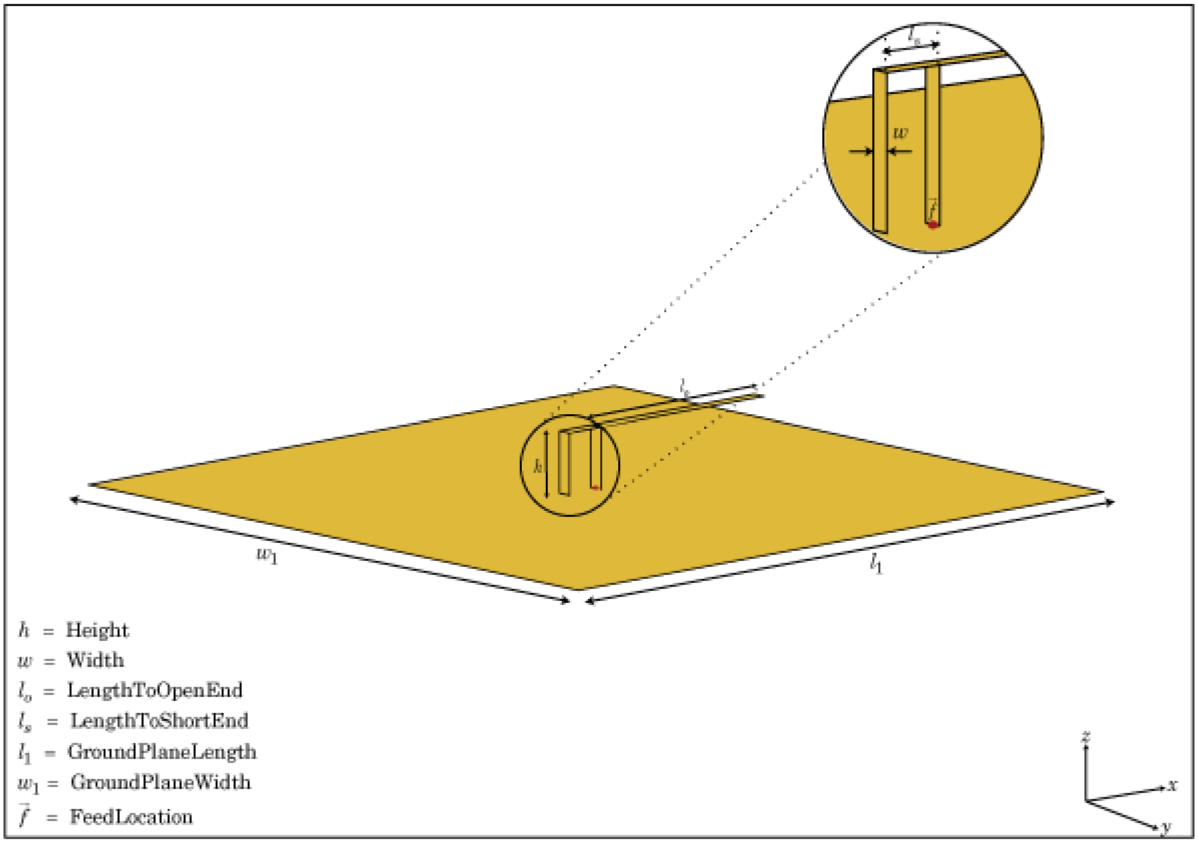
The width of the metal strip is related to the diameter of an equivalent cylinder by the equation

$$w = 2d = 4r$$

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 X-Y 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`

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: `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: `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 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 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 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'`

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

Object Functions

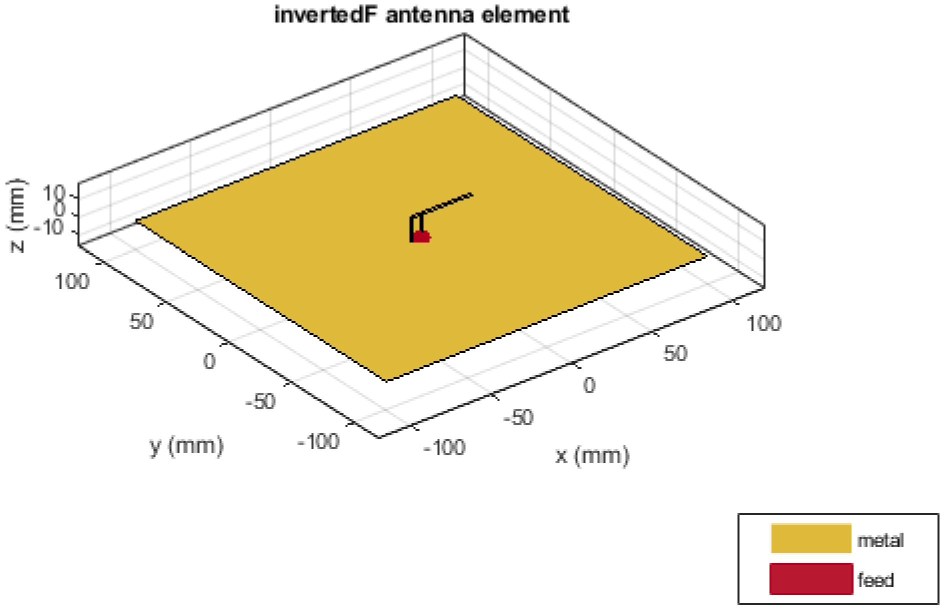
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

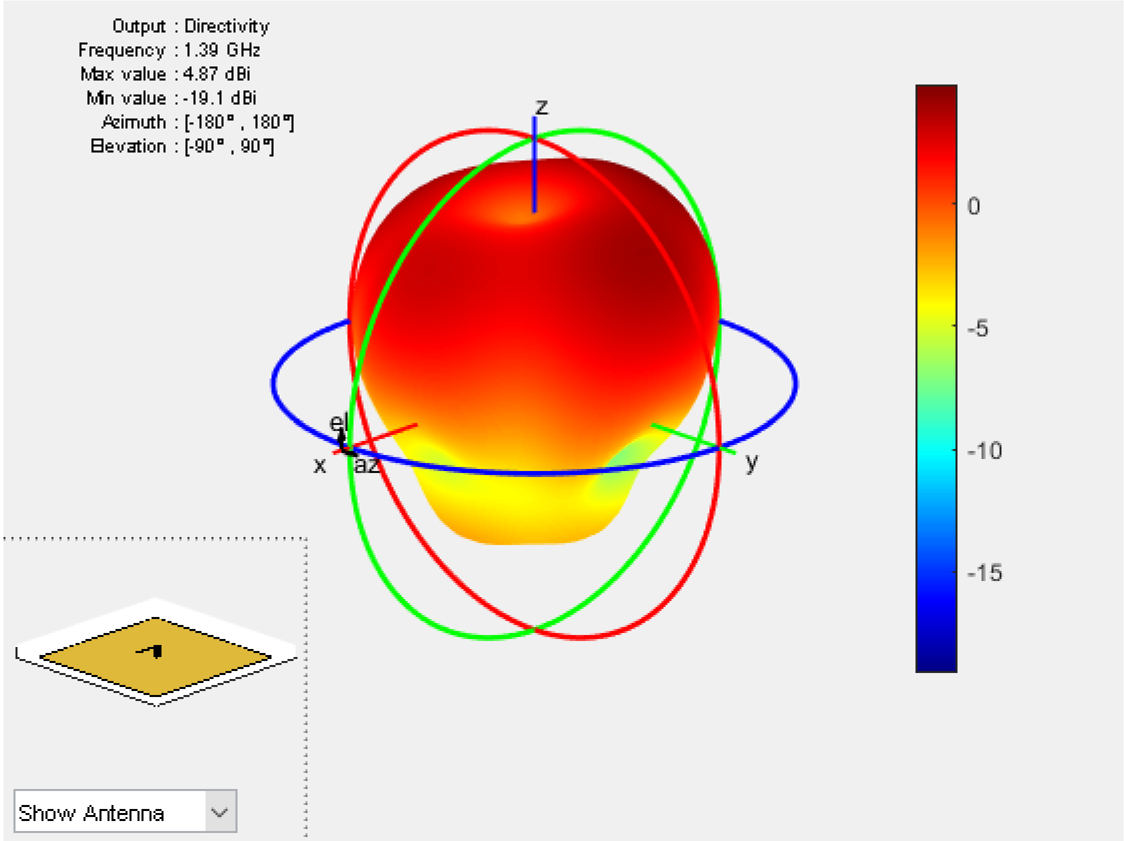
```
f = invertedF('Height',14e-3, 'GroundPlaneLength',200e-3,          ...
                  'GroundPlaneWidth',200e-3);
show(f)
```



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)
```



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

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

Topics

"Rotate Antennas and Arrays"

Introduced in R2015a

invertedL

Create inverted-L antenna over rectangular ground plane

Description

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

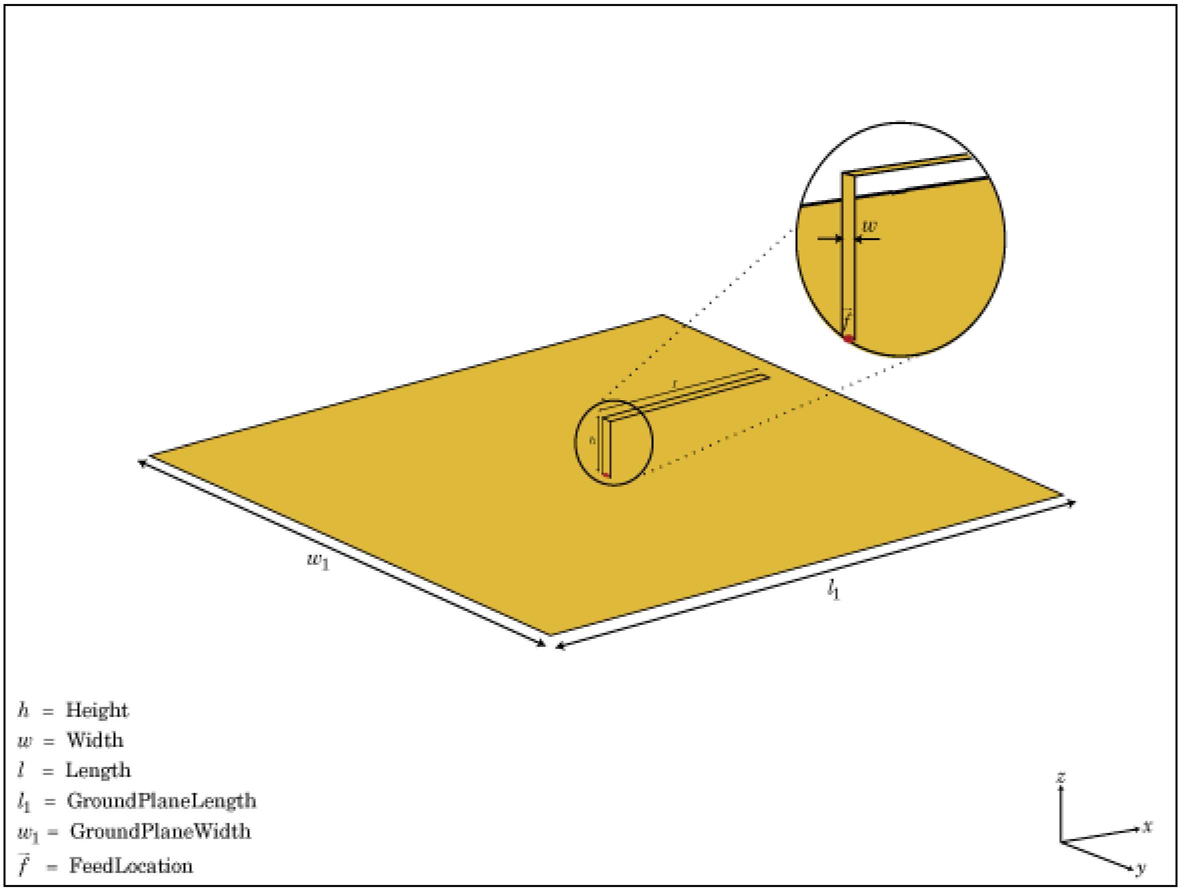
The width of the metal strip is related to the diameter of an equivalent cylinder by the equation

$$w = 2d = 4r$$

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 X-Y 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`

`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: `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: `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 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 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 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'`

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

Object Functions

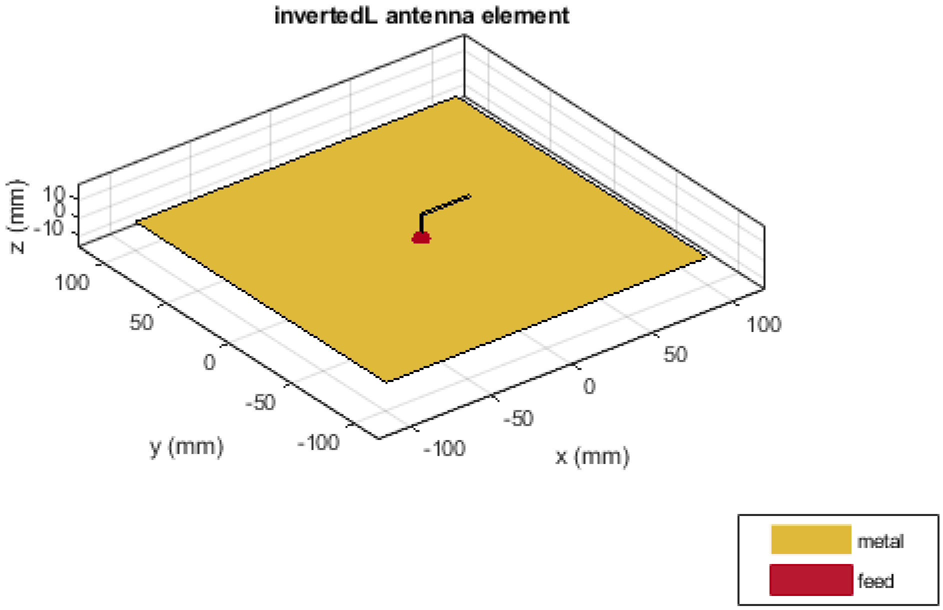
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

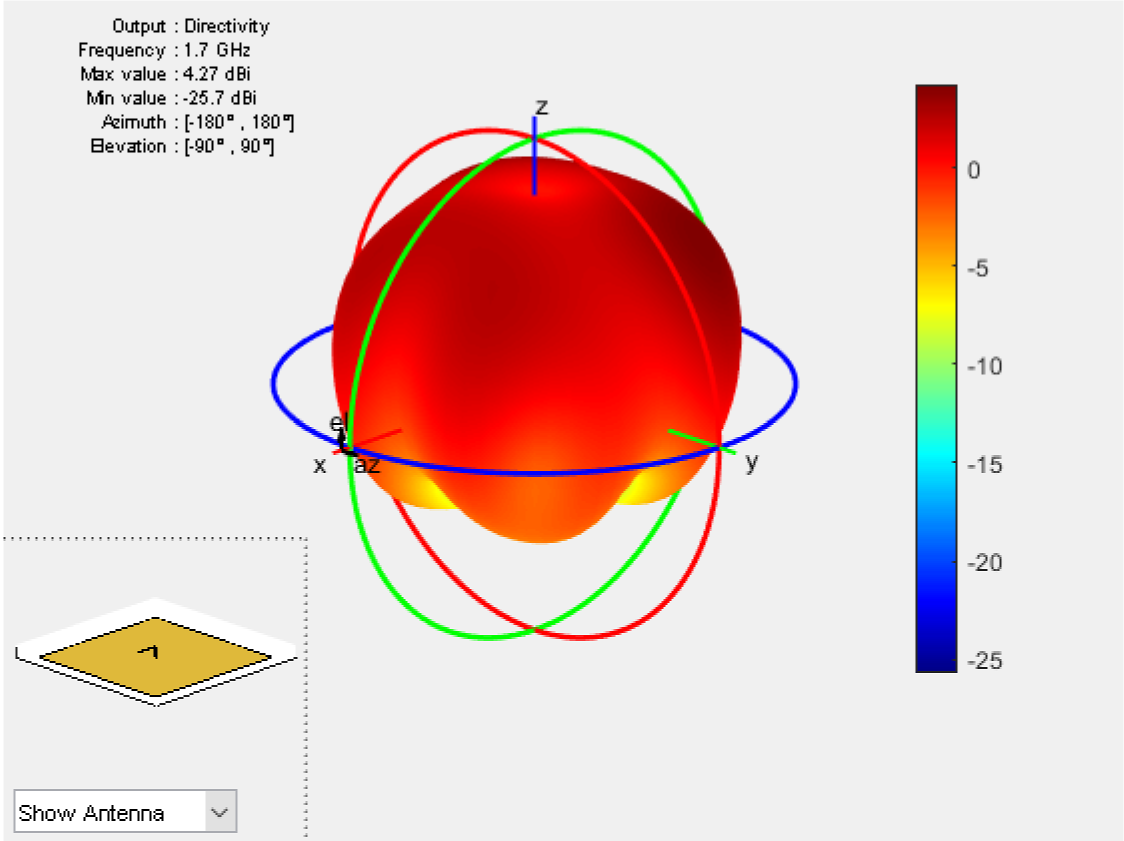
```
il = invertedL('Length',30e-3, 'GroundPlaneLength',200e-3,...
                  'GroundPlaneWidth',200e-3);
show(il)
```



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 : 4.27 dBi
 Min value : -25.7 dBi
   Azimuth : [-180° , 180°]
  Elevation : [-90° , 90°]
```

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

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

Topics

"Rotate Antennas and Arrays"

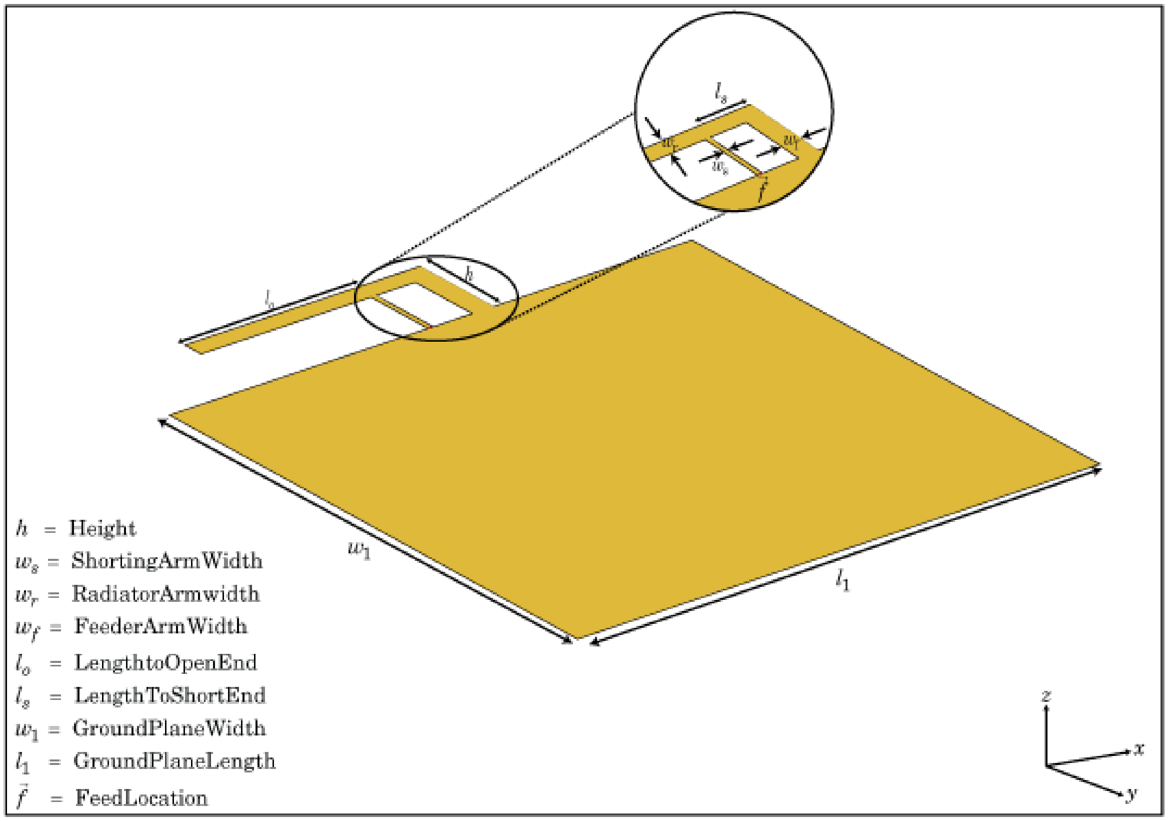
Introduced in R2015a

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 groundplane, specified as a scalar in meters.

Example: 'FeedOffset',0.06

Data Types: double

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: 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: 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 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 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 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'`

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

Analysis Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
show	Display antenna or array structure; display shape as filled patch
vswr	Voltage standing wave ratio of antenna

Examples

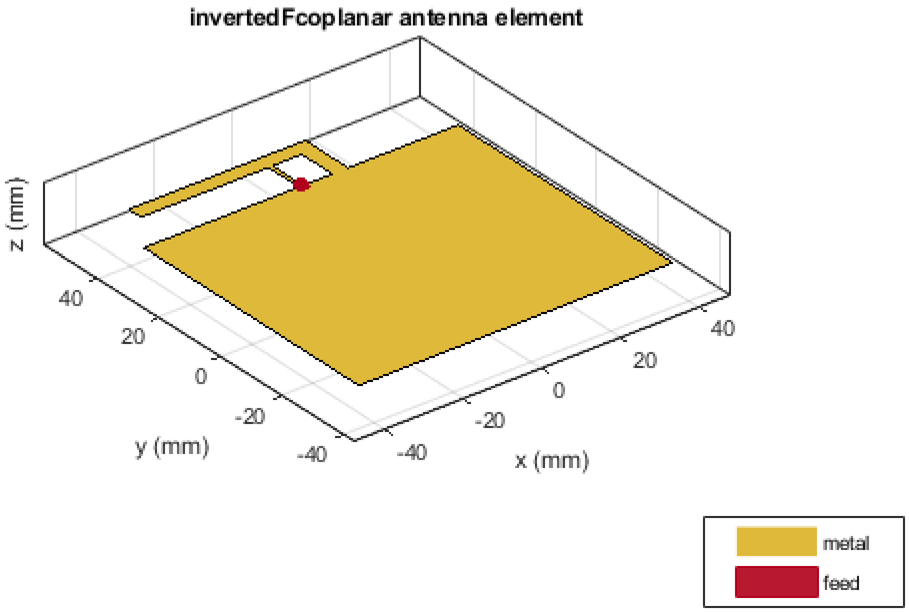
Coplanar Inverted-F Antenna

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

```
show(fco)
```



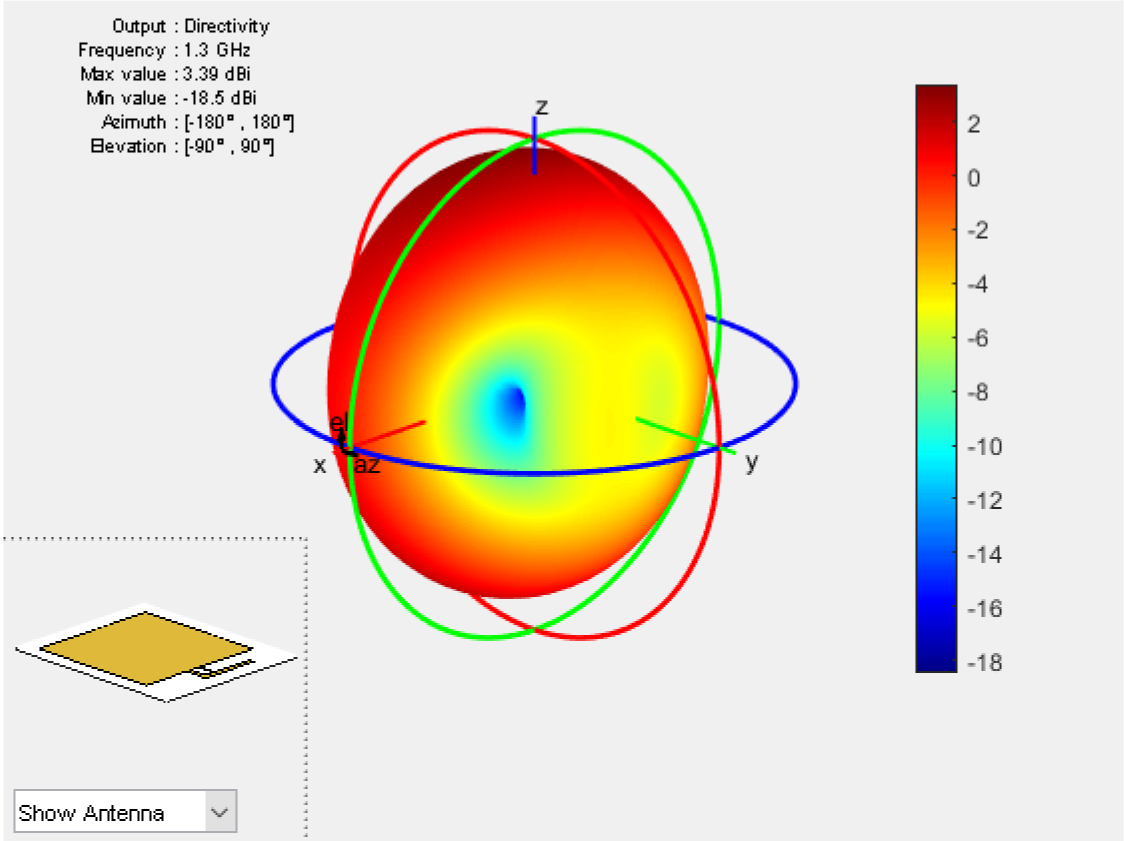
Radiation Pattern of Coplanar Inverted-F Antenna

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 : -18.5 dBi
    Azimuth : [-180° , 180°]
  Elevation : [-90° , 90°]
```

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

invertedF | invertedL | invertedLcoplanar

Topics

"Rotate Antennas and Arrays"

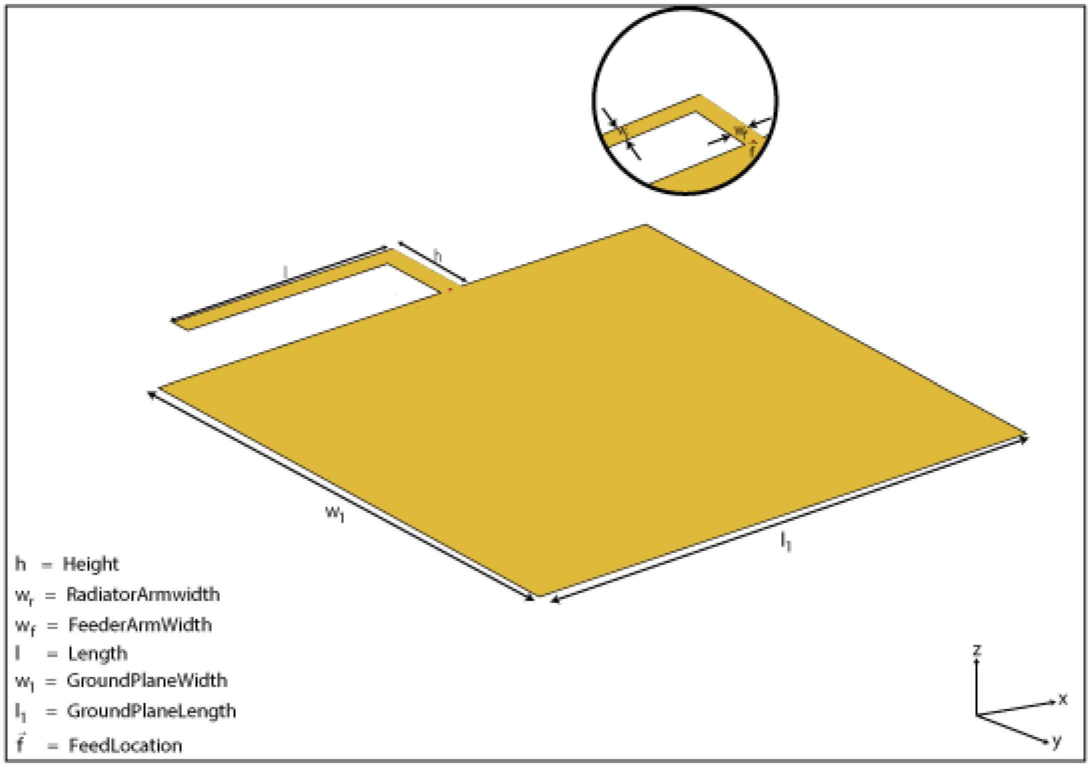
Introduced in R2016b

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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
show	Display antenna or array structure; display shape as filled patch
vswr	Voltage standing wave ratio of antenna

Examples

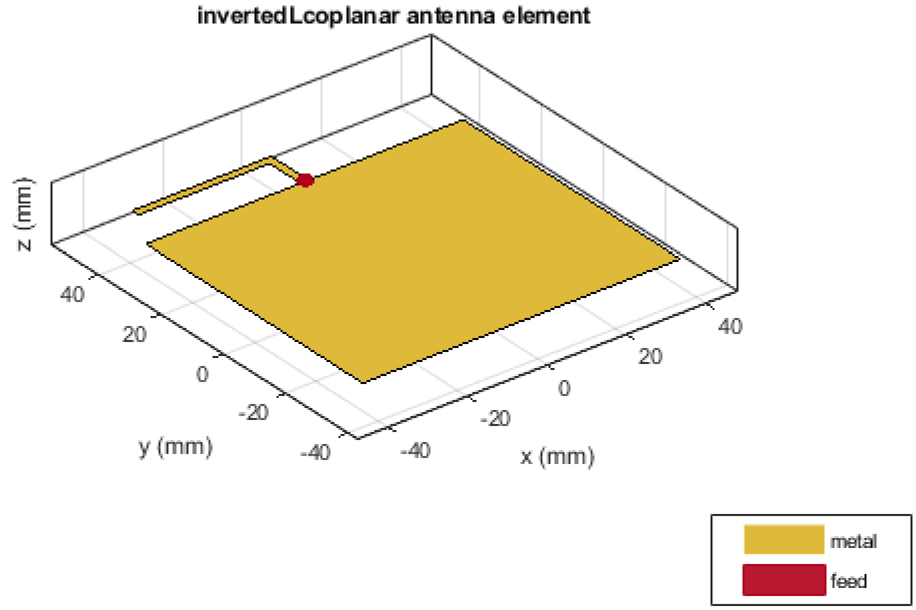
Coplanar Inverted-L Antenna

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

```
show(lco)
```



Impedance of Coplanar Inverted-L Antenna

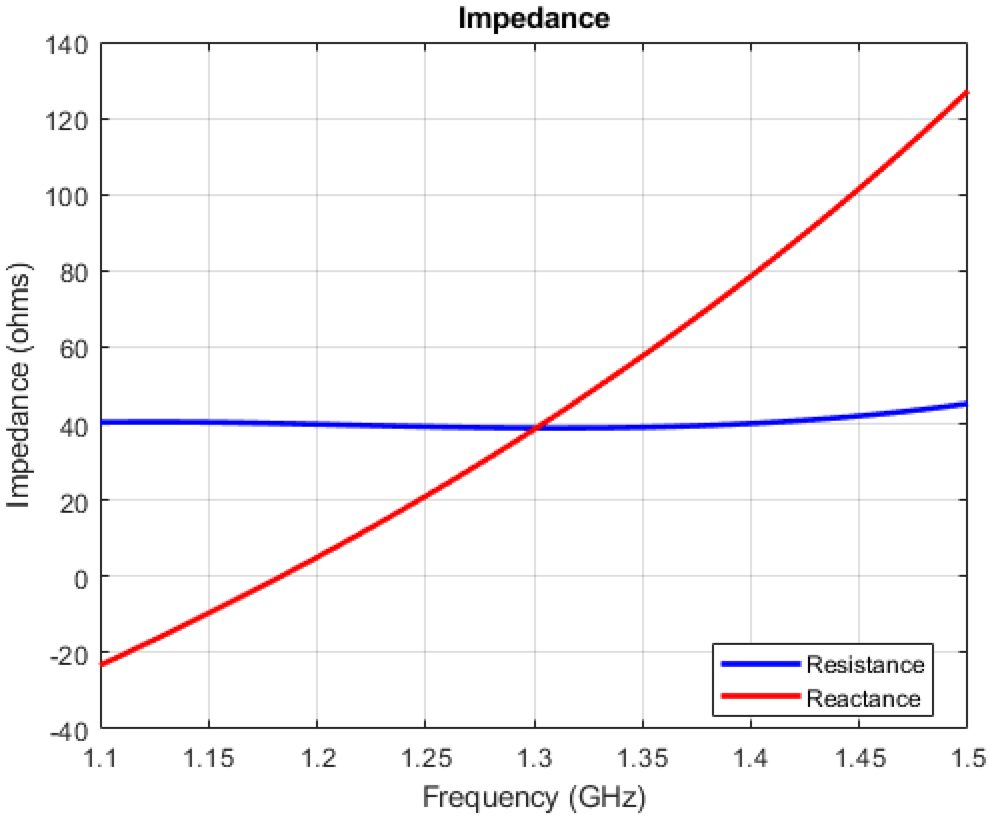
Create a coplanar inverted-L antenna of length 0.050 m, height 0.014 m, ground plane length 0.1 m, and ground plane width 0.1 m.

```
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
                 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);
```



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

`invertedF` | `invertedFcoplanar` | `invertedL`

Topics

"Rotate Antennas and Arrays"

Introduced in R2016b

loopCircular

Create circular loop antenna

Description

The `loopCircular` object is a planar circular loop antenna on the X-Y plane.

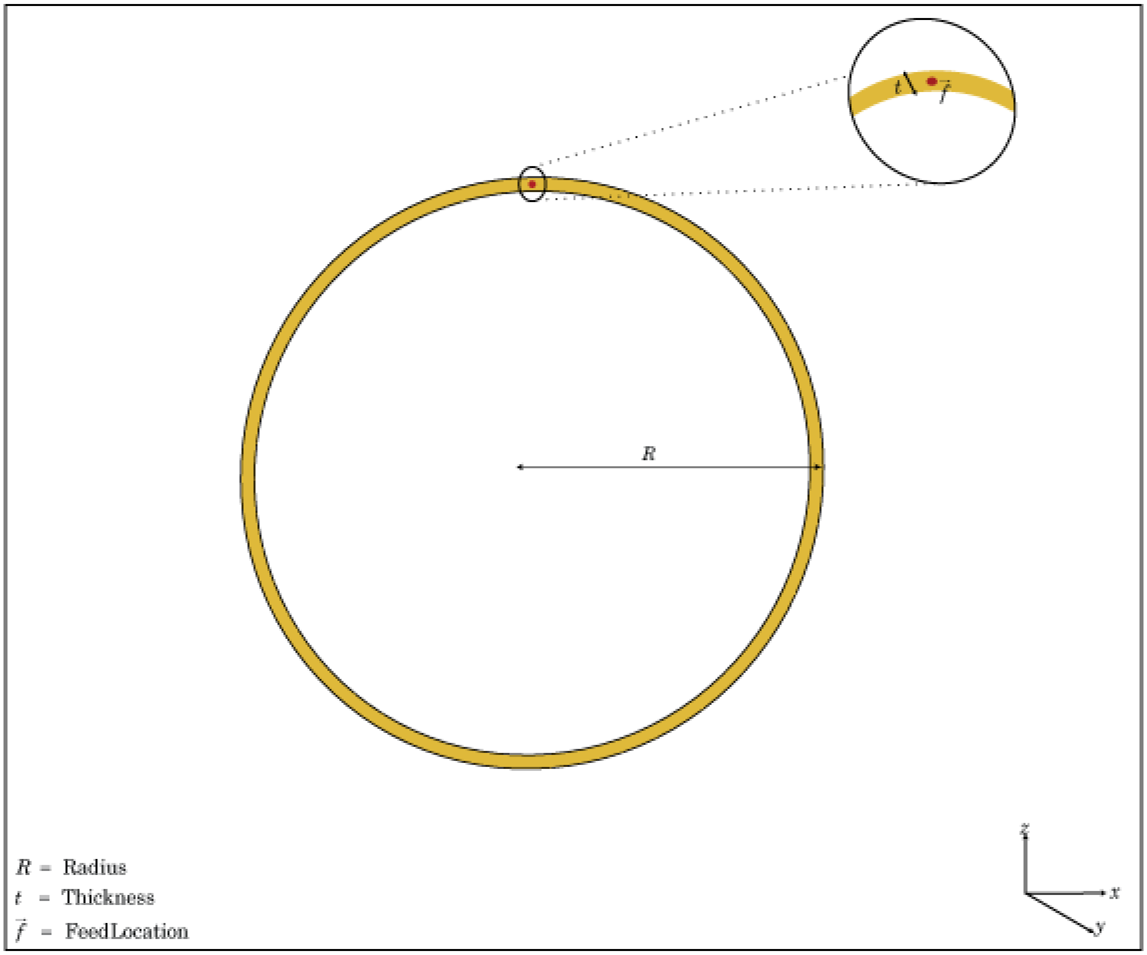
The thickness of the loop is related to the diameter of an equivalent cylinder loop by the equation

$$t = 2d = 4r$$

, 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`

`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: `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: `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 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 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 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'`

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

Object Functions

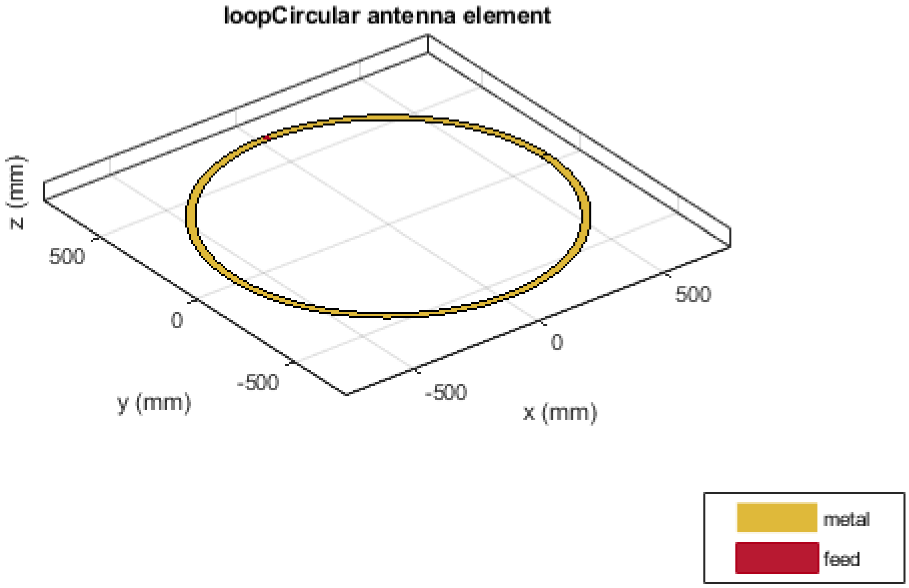
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

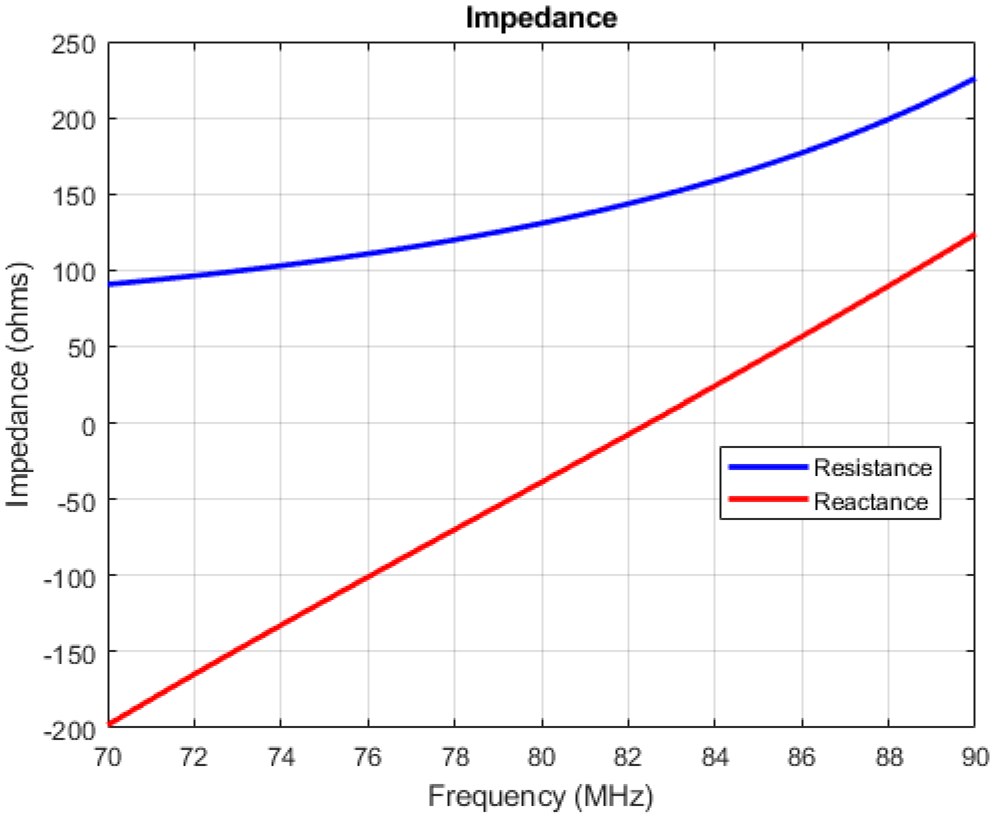
```
c = loopCircular('Radius',0.64,'Thickness',0.03);
show(c)
```



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))
```



References

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

See Also

`dipole` | `loopRectangular` | `slot`

Topics

“Rotate Antennas and Arrays”

Introduced in R2015a

loopRectangular

Create rectangular loop antenna

Description

The `loopRectangular` object is a rectangular loop antenna on the X-Y plane.

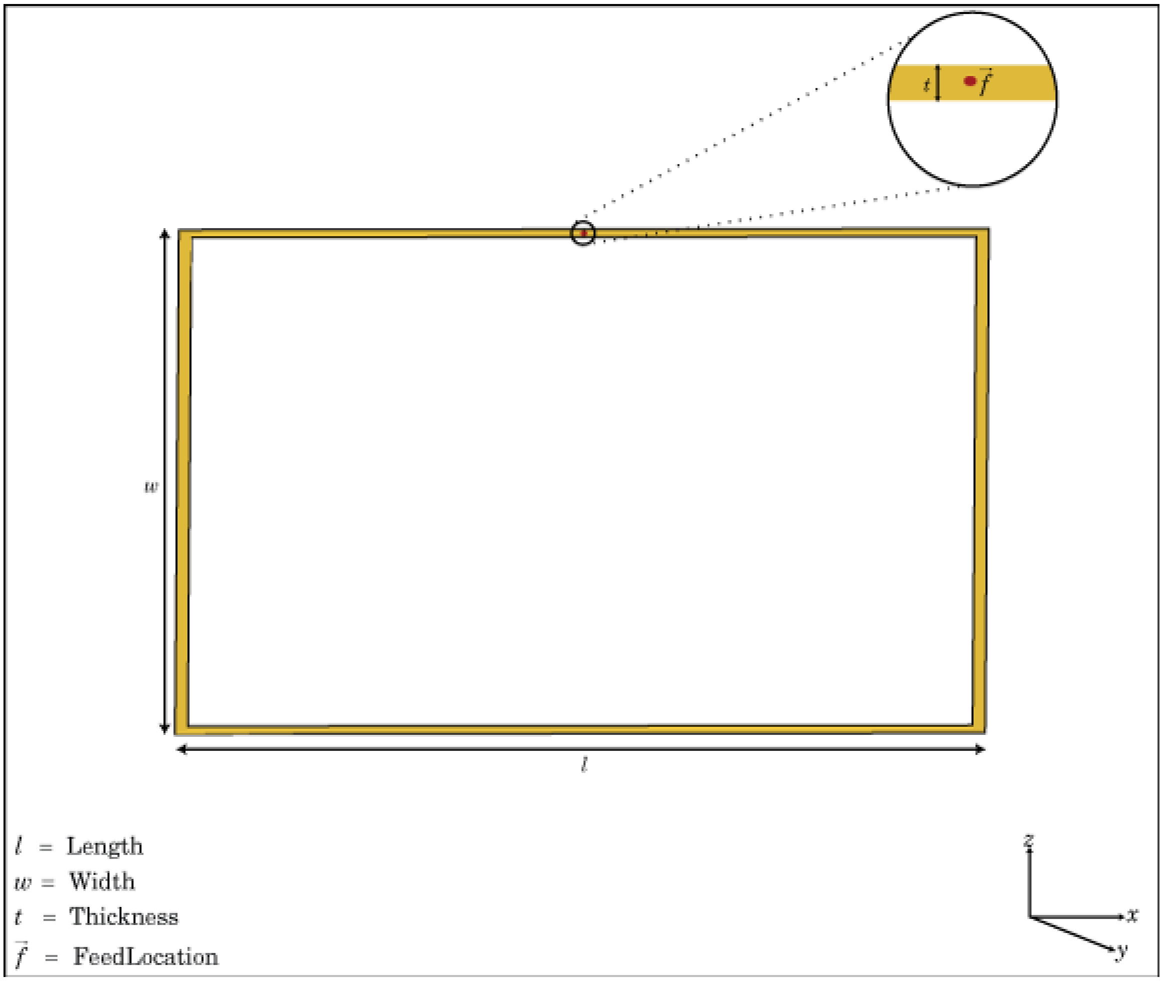
The thickness of the loop is related to the diameter of an equivalent cylinder loop by the equation

$$t = 2d = 4r$$

, 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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

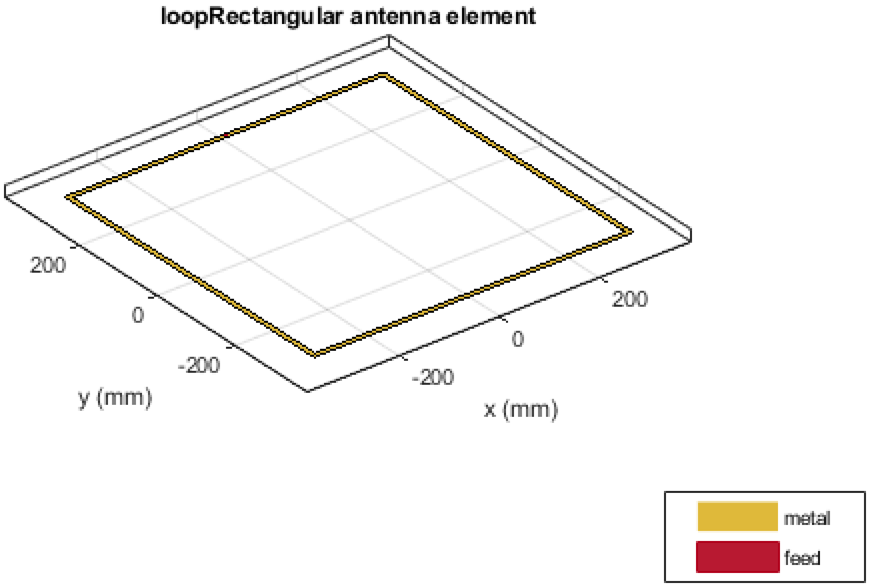
Create and View Rectangular Loop Antenna

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

```
show(r)
```



Impedance of Rectangular Loop Antenna

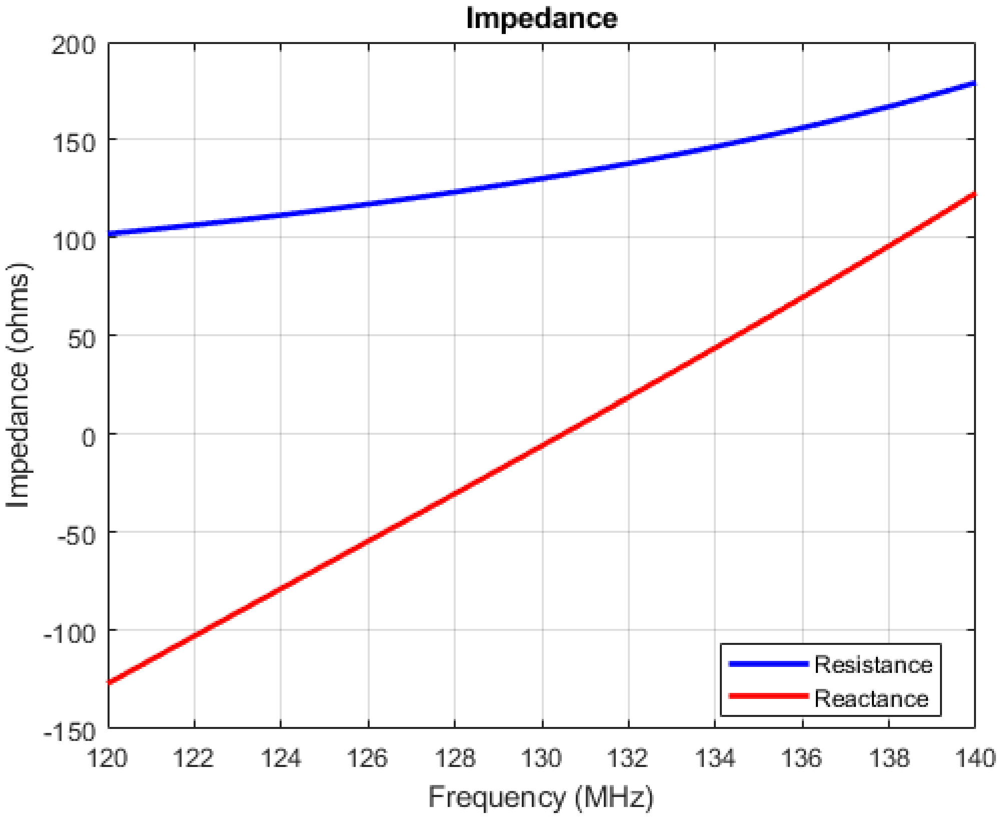
Calculate the impedance of a rectangular loop antenna over a frequency range of 120MHz-140MHz.

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

```
r = 
  loopRectangular with properties:

       Length: 0.6400
        Width: 0.6400
    Thickness: 0.0100
```

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

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



References

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

See Also

`dipole` | `loopCircular` | `monopole`

Topics

"Rotate Antennas and Arrays"

Introduced in R2015a

monopole

Create monopole antenna over rectangular ground plane

Description

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

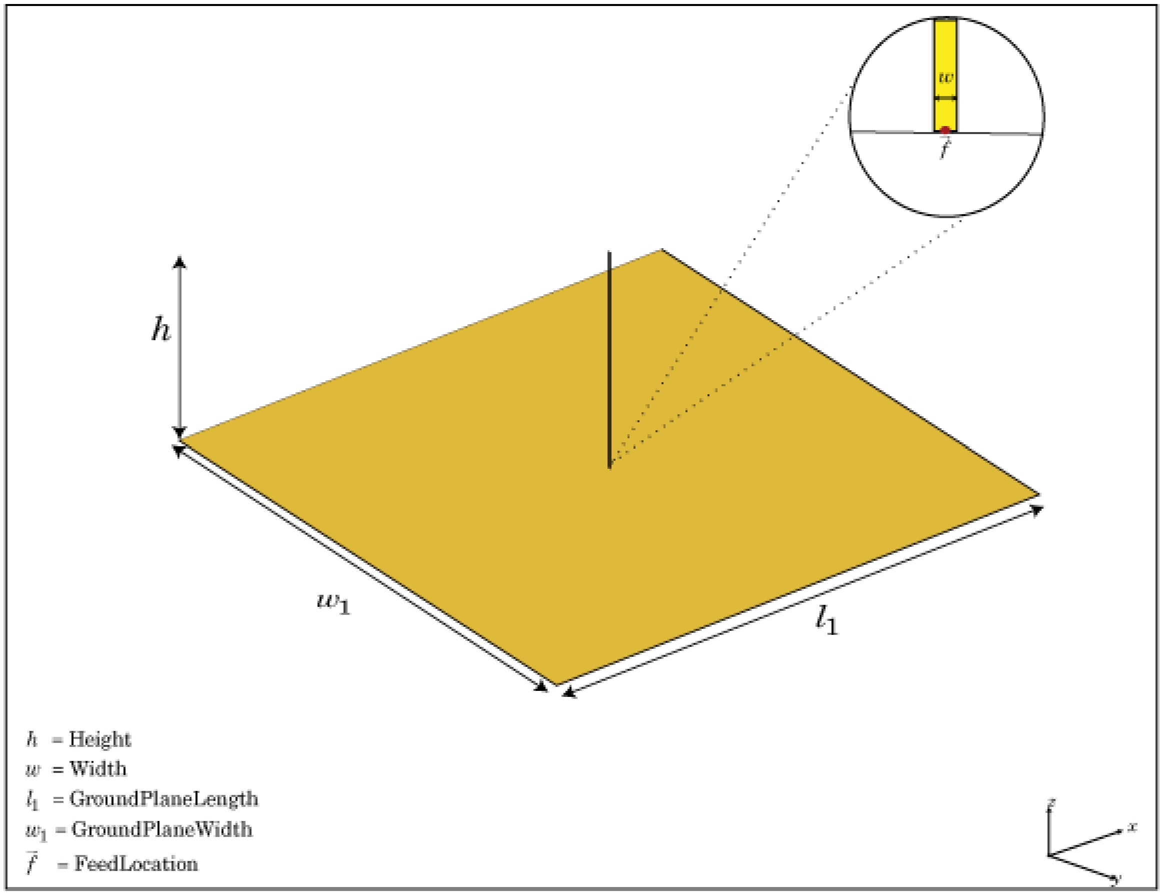
The width of the monopole is related to the diameter of an equivalent cylindrical monopole by the equation

$$w = 2d = 4r$$

, 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 X-Y 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

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: 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: 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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

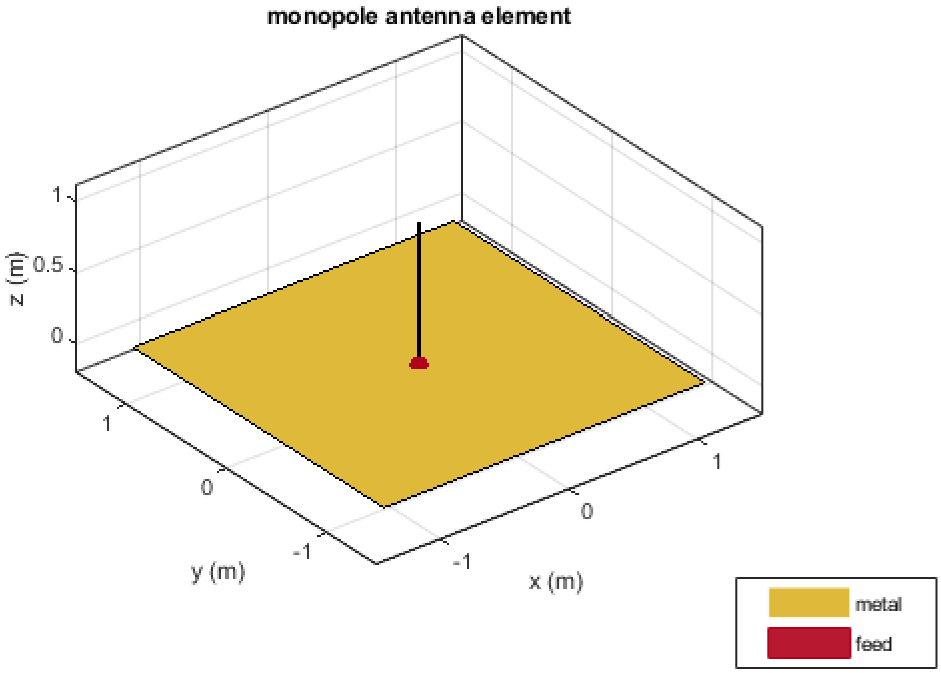
Create and View Monopole Antenna

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

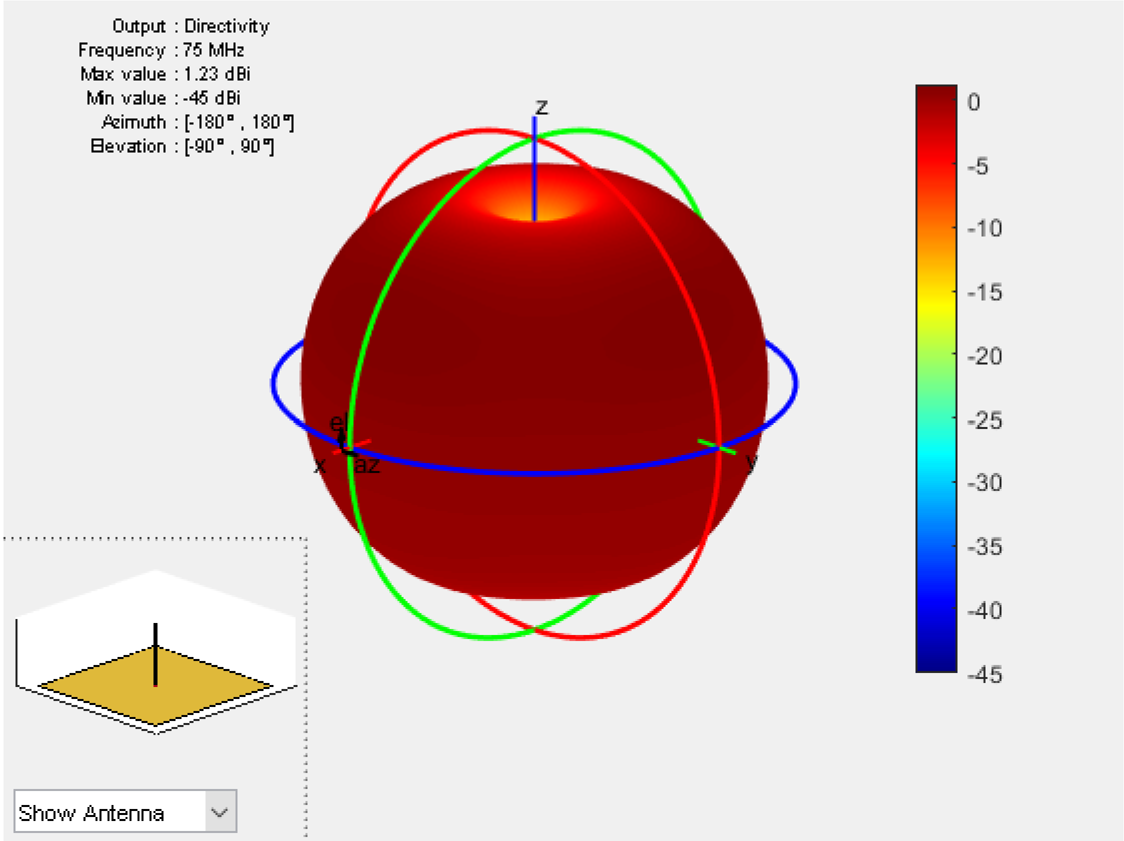
```
show(m)
```



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 : 1.23 dBi
Min value : -45 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

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` | `monopoleTopHat` | `patchMicrostrip`

Topics

"Rotate Antennas and Arrays"

Introduced in R2015a

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.

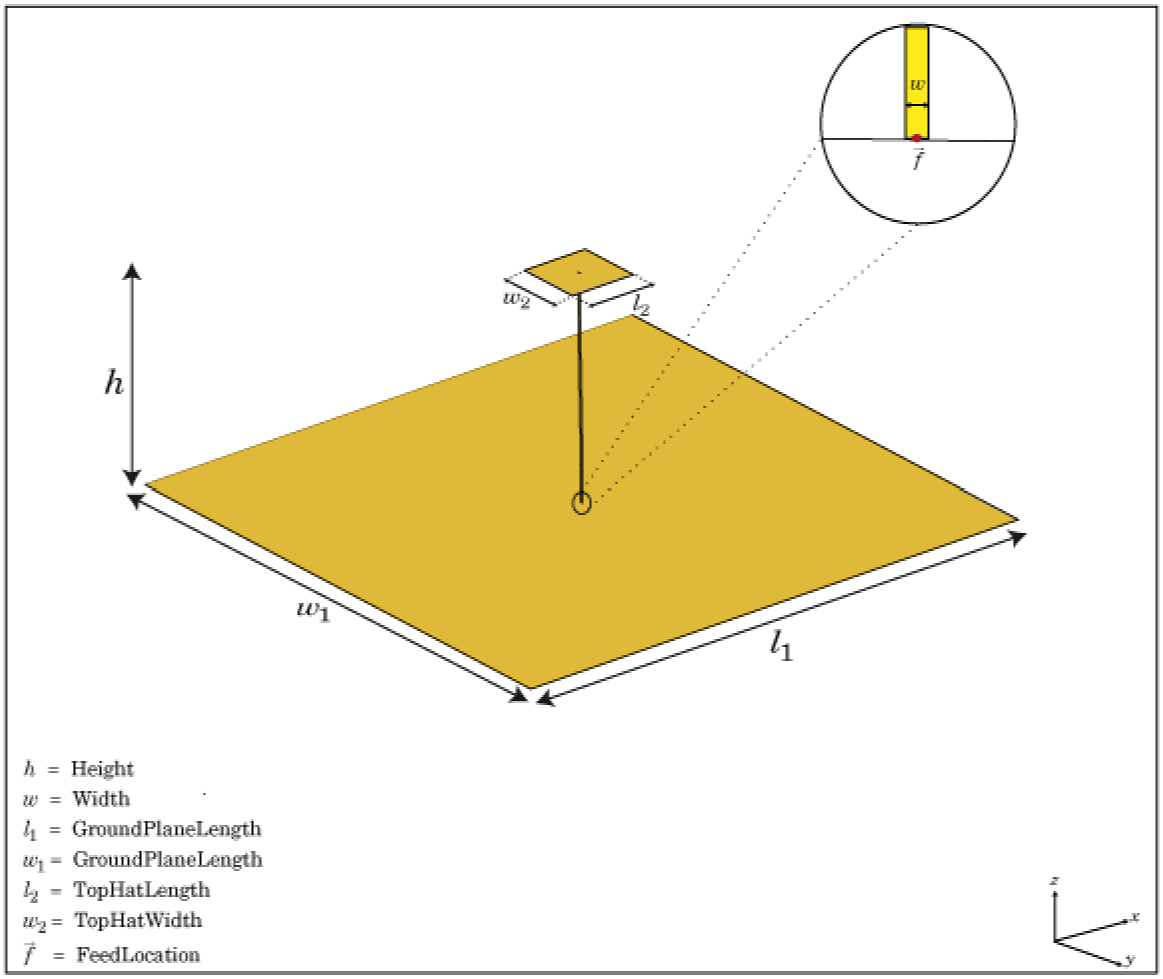
The width of the monopole is related to the diameter of an equivalent cylindrical monopole by the expression

$$w = 2d = 4r$$

,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 X-Y 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 retains 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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

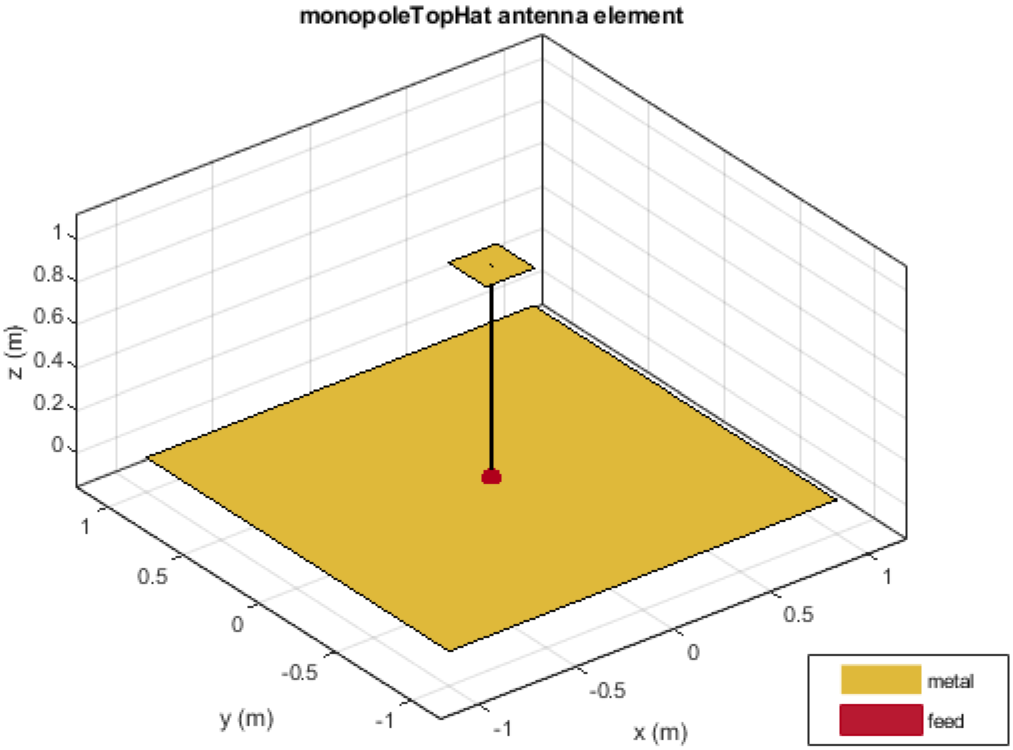
Create and View Top Hat Monopole.

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

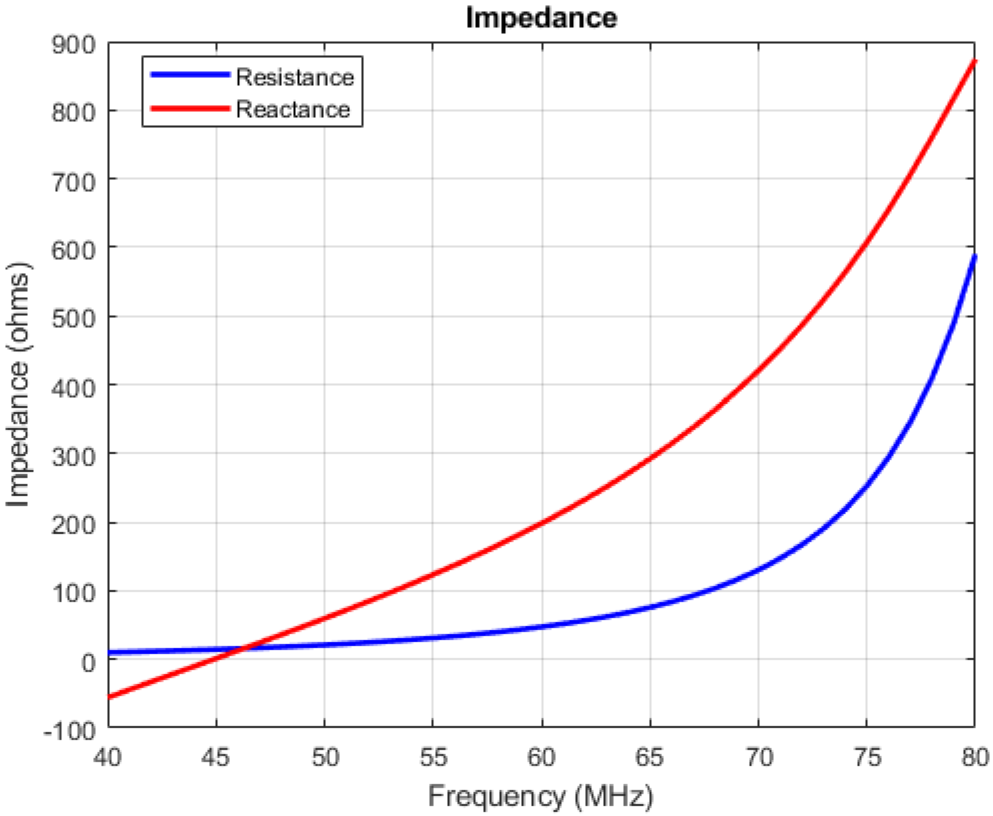
```
show(th)
```



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.

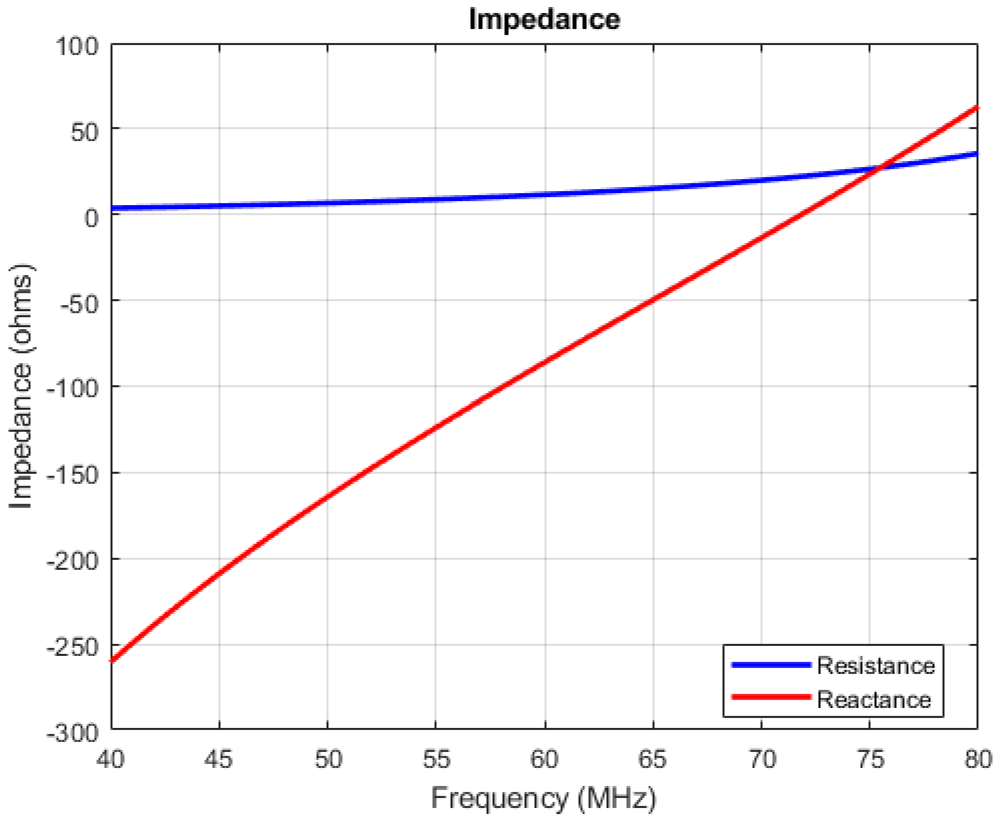
```
th = monopoleTopHat;
impedance(th,linspace(40e6,80e6,41));
```



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));
```



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` | `monopole` | `patchMicrostrip`

Topics

"Rotate Antennas and Arrays"

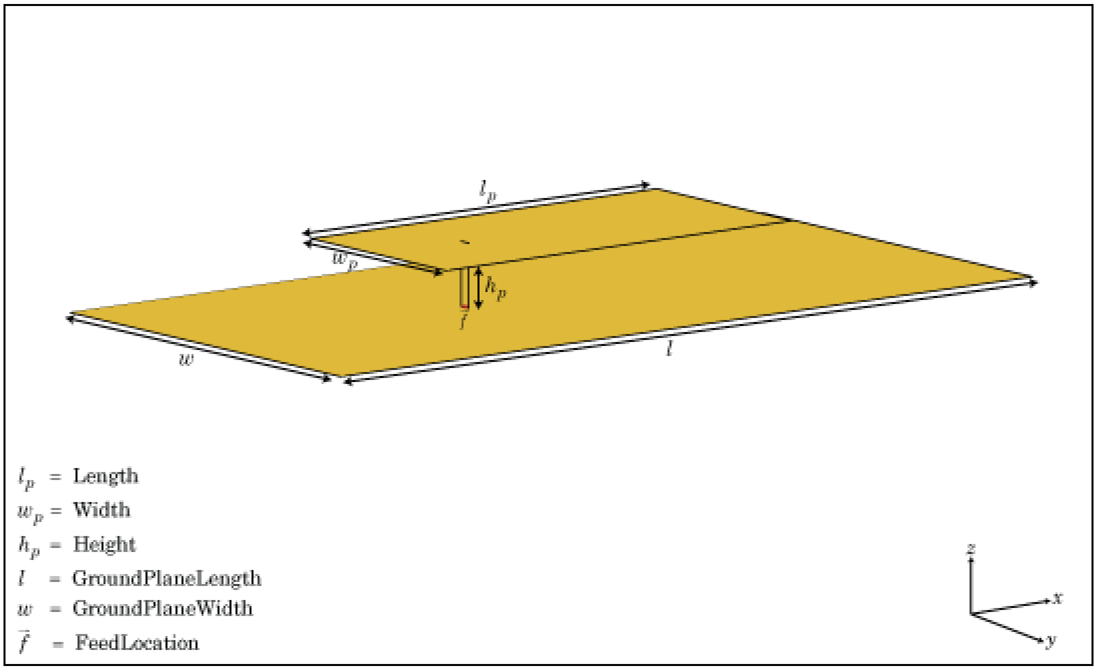
Introduced in R2015a

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 handle

Type of dielectric material used as a substrate, specified as a dielectric material object handle. 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.

Example: `'FeedOffset',[0.01 0.01]`

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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

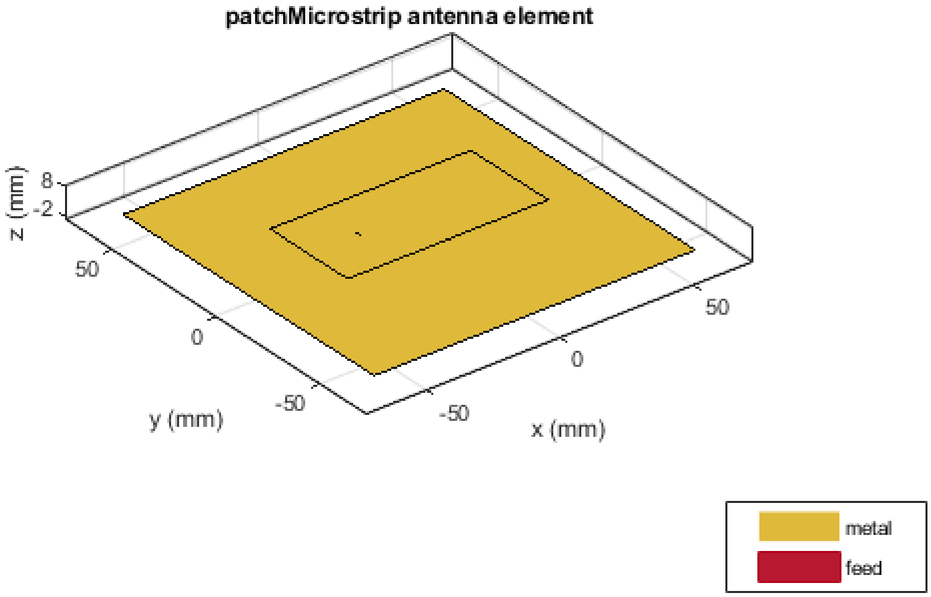
Create and View Microstrip Patch Antenna

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

```
show (pm)
```



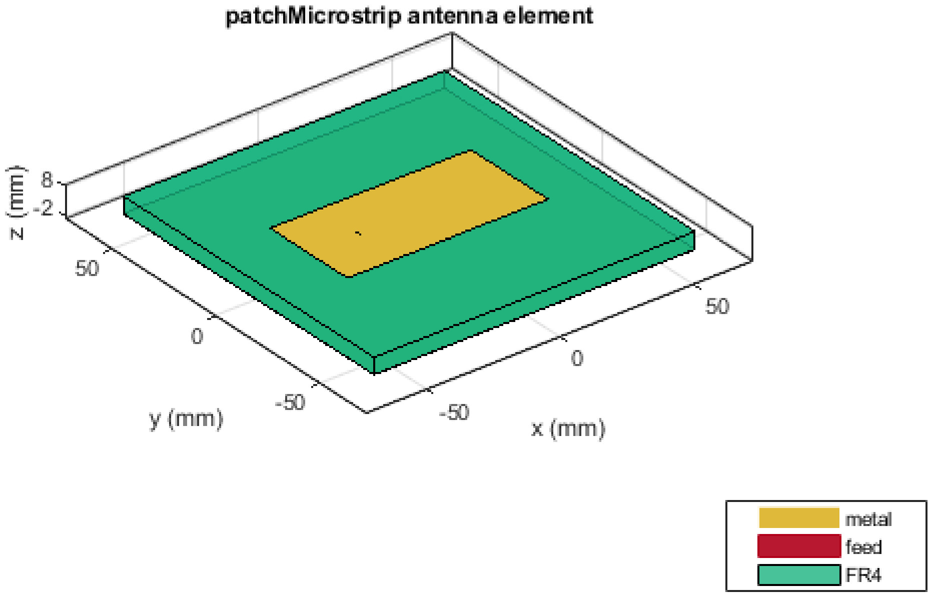
Radiation Pattern of Microstrip Patch Antenna

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)
```

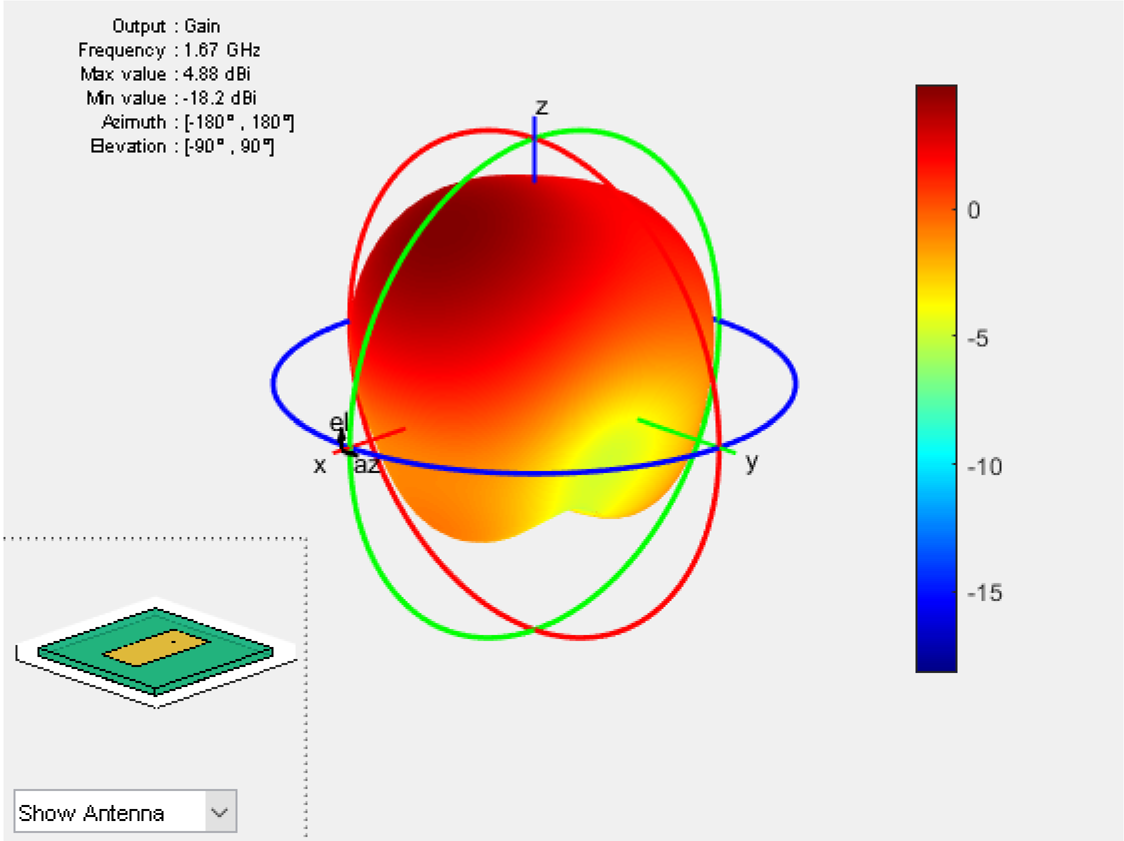
```
pm = 
  patchMicrostrip with properties:

               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]
                 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)
```



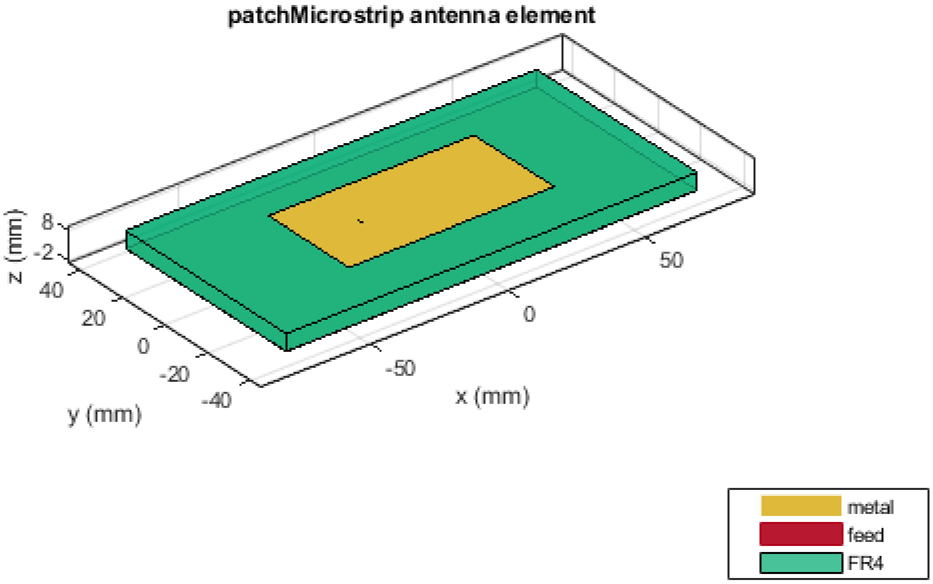
Impedance of Microstrip Patch Antenna

Create a microstrip patch antenna using **'FR4'** as the dielectric substrate.

```
d = dielectric('FR4');
pm = patchMicrostrip('Substrate',d)
```

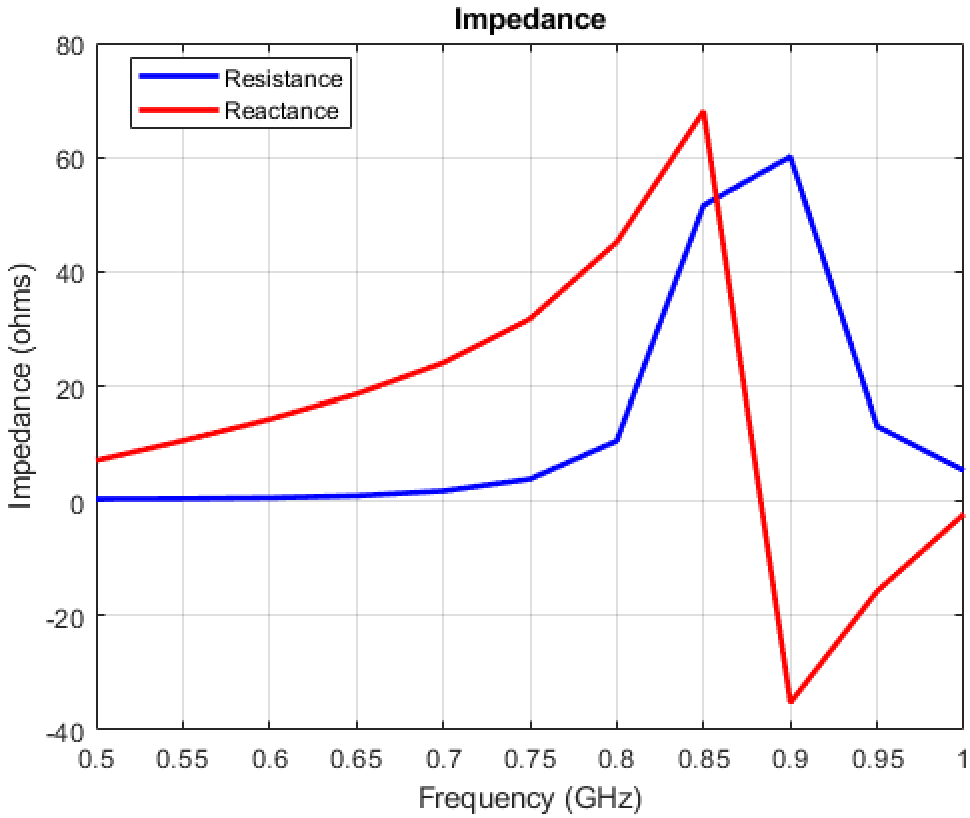
```
pm =
  patchMicrostrip with properties:

              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]
                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));
```



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 Mutual Coupling Between different patches"
"Rotate Antennas and Arrays"

Introduced in R2015a

planeWaveExcitation

Create plane wave excitation environment for antenna or array

Description

The `planeWaveExcitation` object creates an environment where a plane wave excites an antenna or array. Plane wave excitation is a scattering solution that solves the receiving antenna problem. By default, the antenna element is a dipole. The dipole is excited using a plane wave that travels along the positive x-axis having a z-polarization.

Creation

Syntax

```
h = planeWaveExcitation
h = planeWaveExcitation(Name,Value)
```

Description

`h = planeWaveExcitation` creates an environment where a plane wave excites the antenna or array. By default, the plane wave excites a dipole antenna.

`h = planeWaveExcitation(Name,Value)` returns a `planeWaveExcitation` environment, 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` — Antenna or array element
`dipole` (default) | object handle

Antenna or array element, specified as an object handle.

Note For infinite array, support for unit cell analysis is for only transmit scenarios.

Example: `'Element',linearArray`

`Direction` — Incidence of plane wave
`[1 0 0]` (default) | three-element real vector

Incidence of plane wave, specified as a three-element real vector.

Example: `'Direction',[0 0 1]`

Data Types: `double`

`Polarization` — Polarization of incident electric field
`[0 0 1]` (default) | three-element complex vector

Polarization of incident electric field in *x*, *y*, and *z* components, specified as a three-element complex vector in V/m The polarization vector gives the orientation and magnitude of the electric field.

Example: `'Polarization',[0 1 0]`

Data Types: `double`

Object Functions

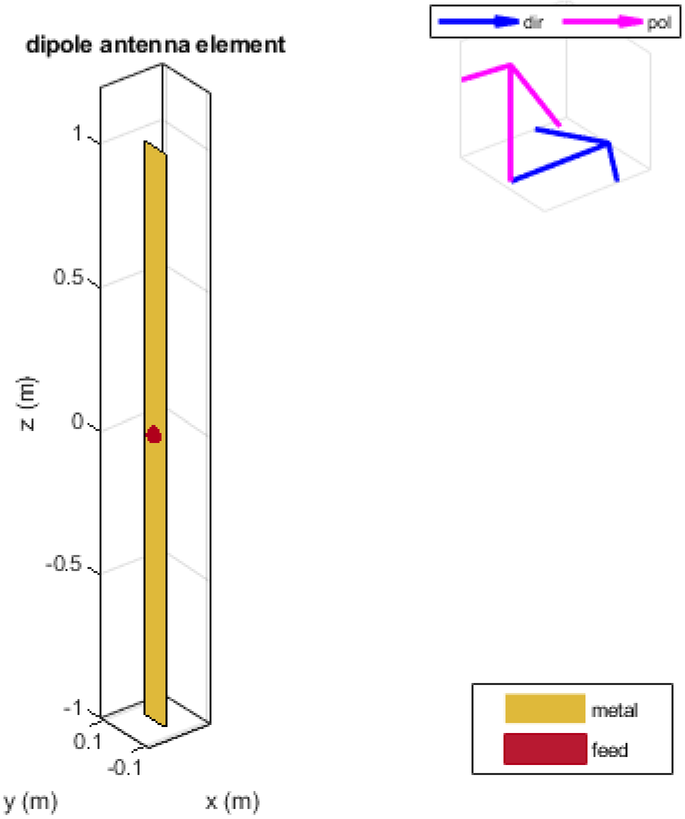
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal or 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 or array structure; display shape as filled patch

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. By default, the direction is along the x-axis. The pink arrow shows polarization of the plane wave. By default, the polarization is perpendicular to the direction of propagation i.e. 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.

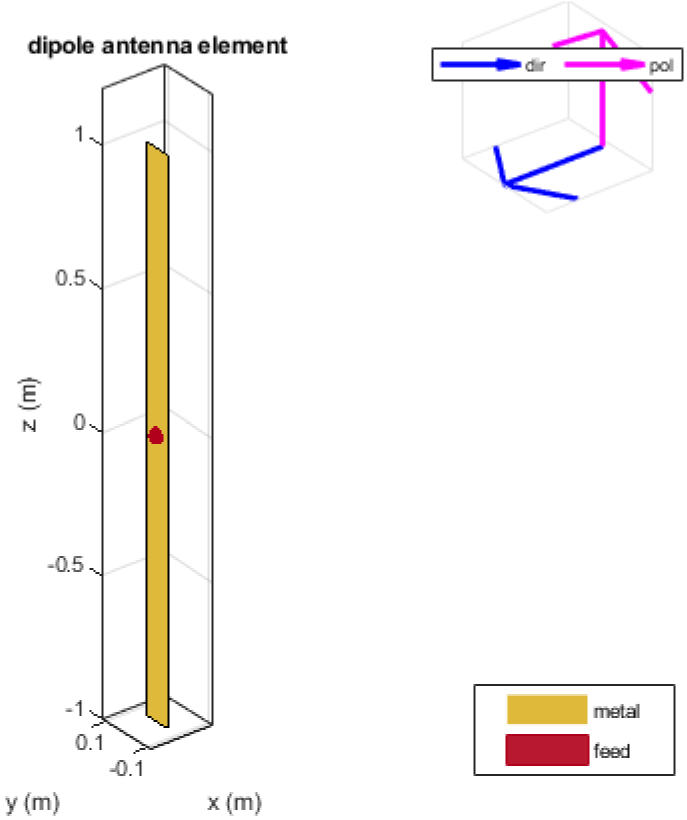
```
h = planeWaveExcitation
cur = feedCurrent(h, 70e6)
```

```
h = 

  planeWaveExcitation with properties:

         Element: [1×1 dipole]
       Direction: [1 0 0]
    Polarization: [0 0 1]


cur = 
   0.0179 - 0.0040i
```

Current Distribution On Antenna

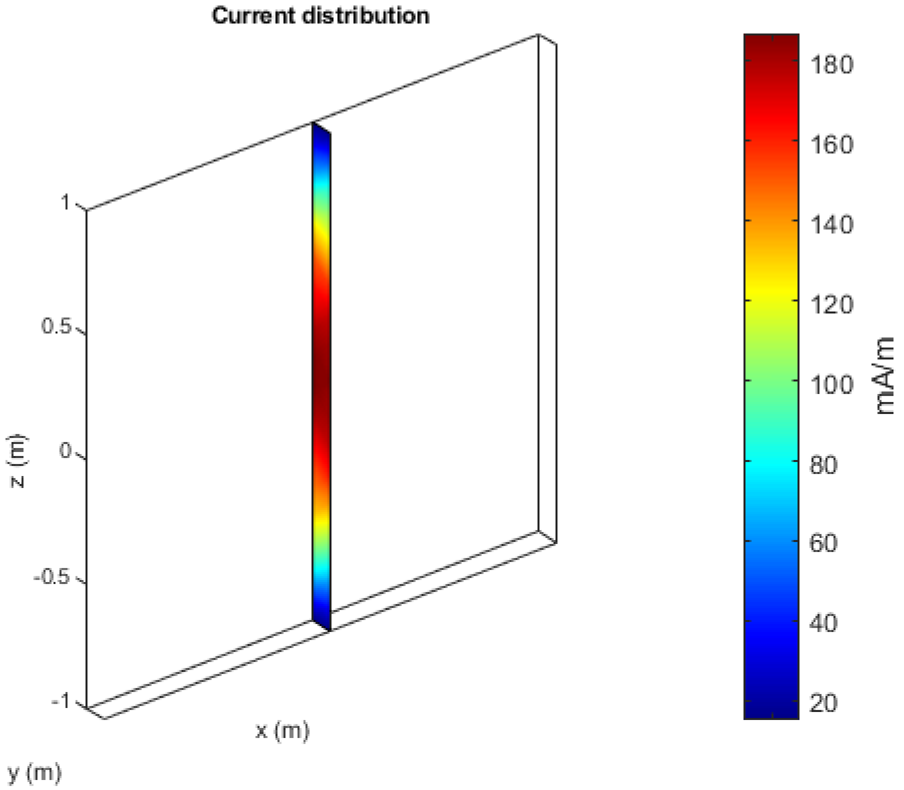
Excite a dipole antenna using a plane wave. The polarization of the wave is along the z-axis and the direction of propagation is along the negative x-axis. View the antenna.

```
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.

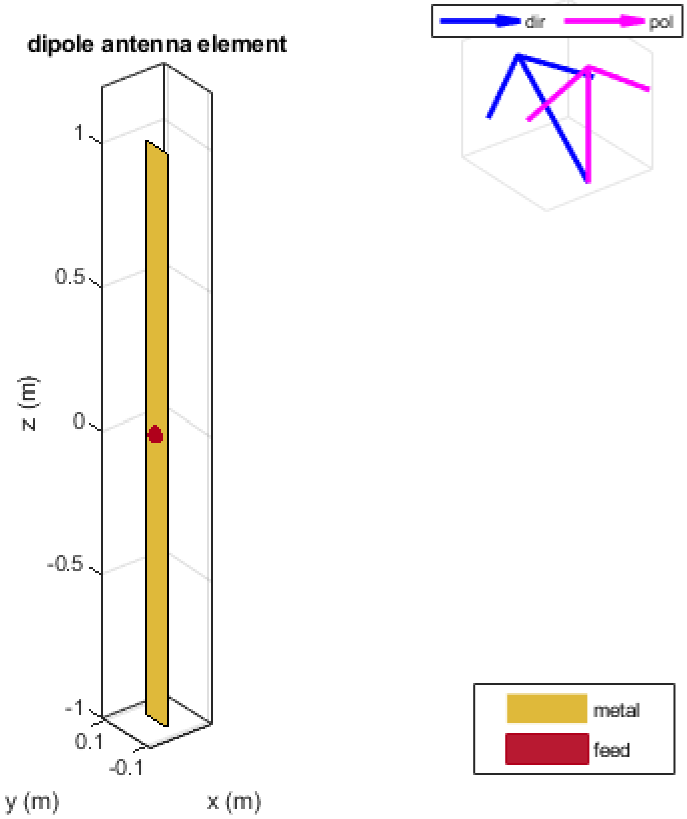
```
current(p, 70e6);
```



Antenna Excited By Plane Wave In Arbitrary Direction

Consider a dipole excited by a plane wave.

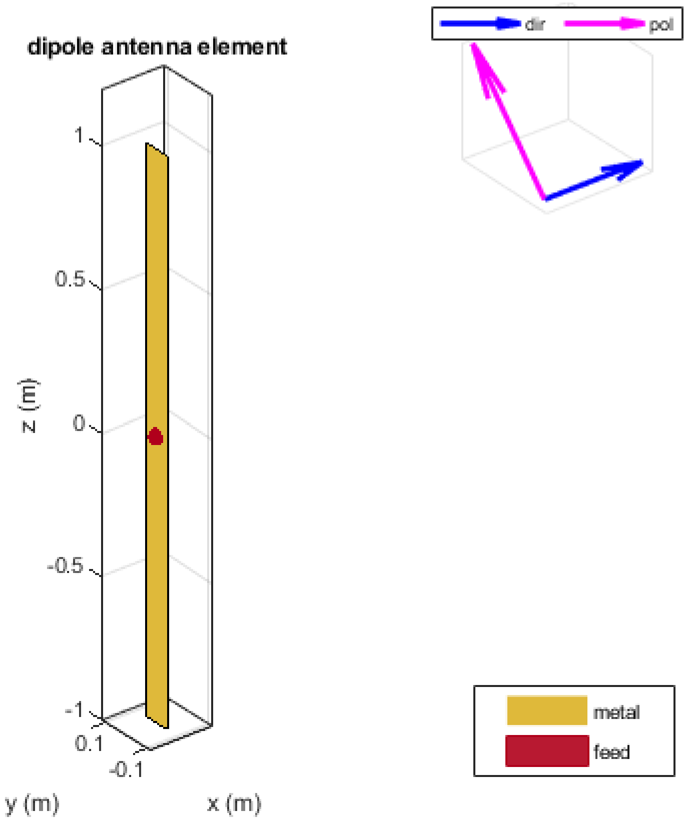
```
p = planeWaveExcitation;
p.Direction = [0 1 1];
show(p)
```



If you use the above option, any analysis of this antenna will error out as the polarization and direction vector are not orthogonal to each other.

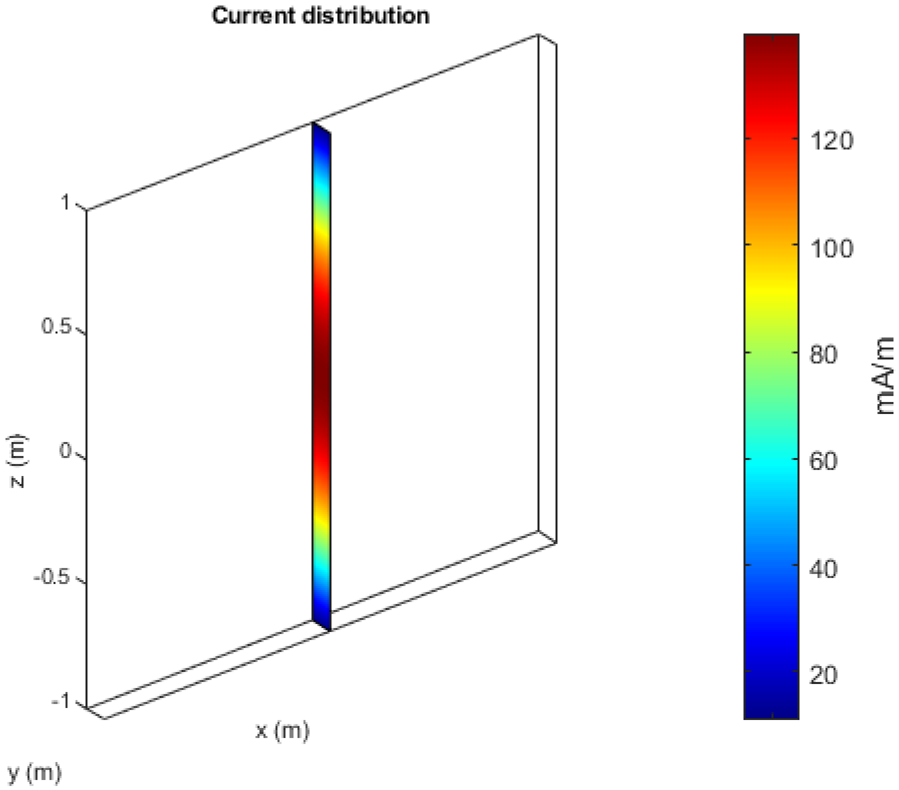
Use the cross-product function to find the appropriate polarization direction of such a wave.

```
p = planeWaveExcitation;
p.Polarization = cross(p.Direction, [0 1 1]);
show(p);
```



Calculate the current distribution of the antenna.

```
current(p,75e6);
```



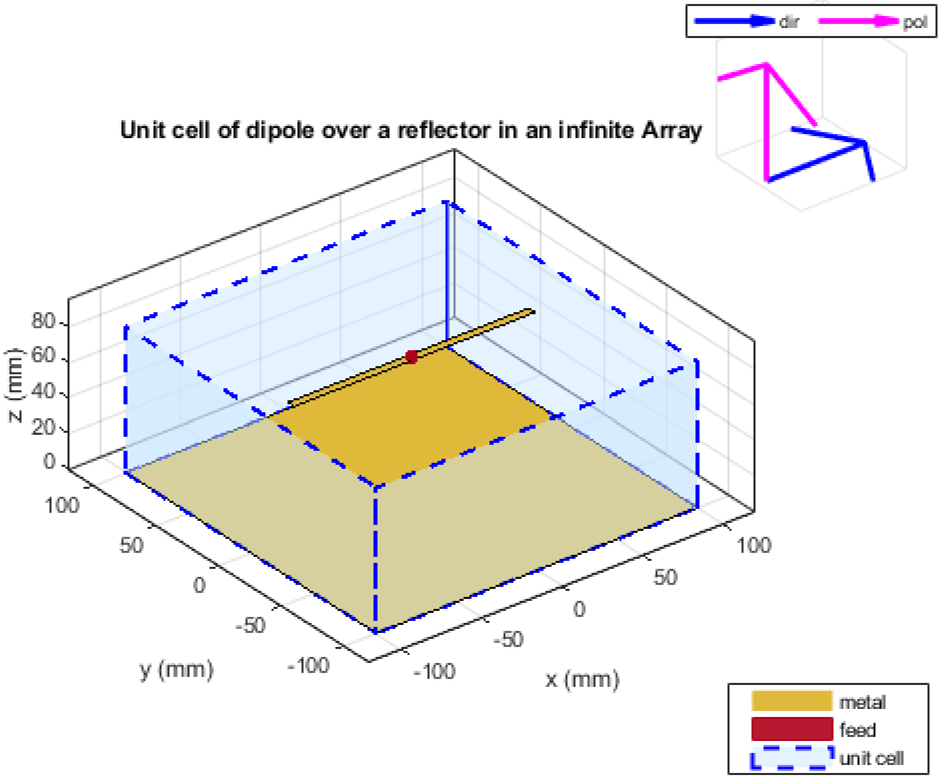
Plane Wave Excitation of Infinite Array

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]
```

```
show(p)
```



References

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

See Also

`dipole` | `linearArray`

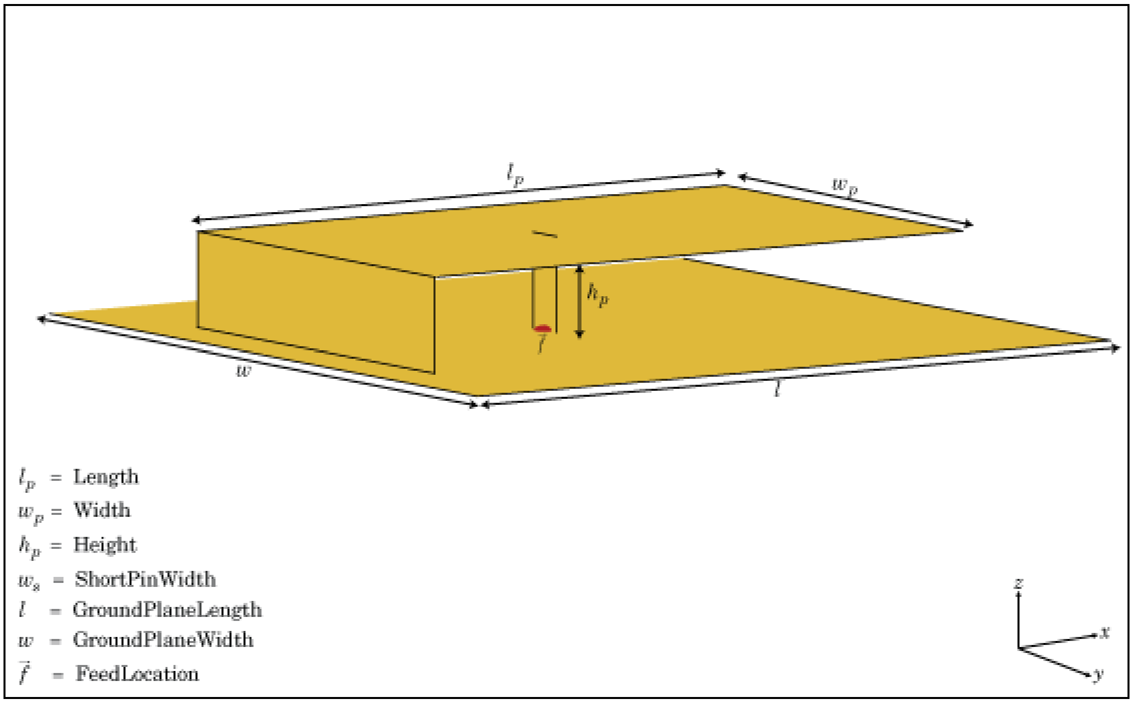
Introduced in R2017a

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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

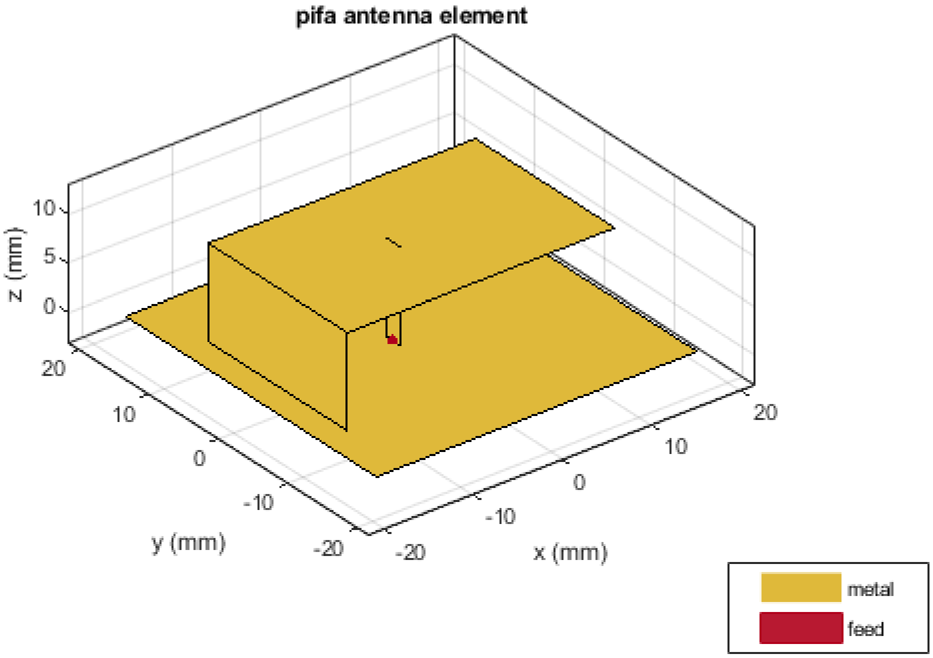
Create and View Planar Inverted-F Antenna(PIFA) Antenna

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

```
show(pf)
```



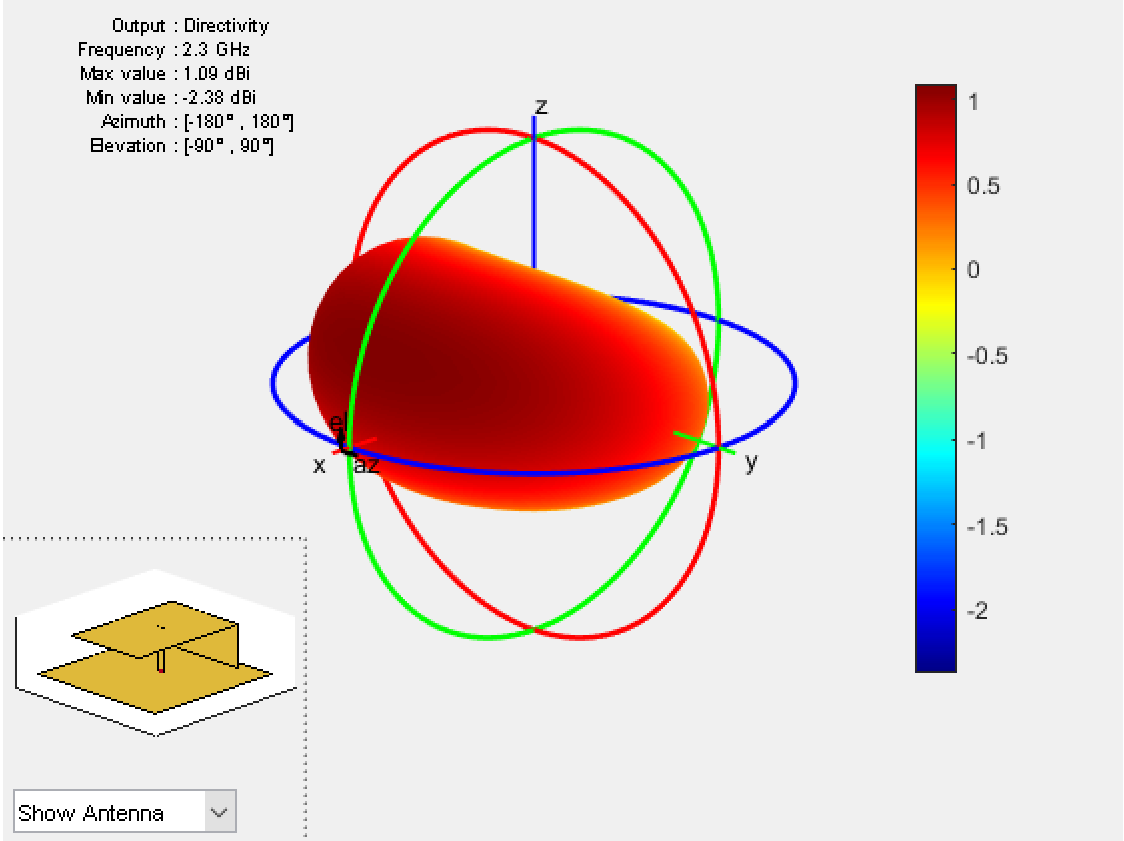
Radiation Pattern of PIFA Antenna

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

```
pattern(pf,2.3e9);
```



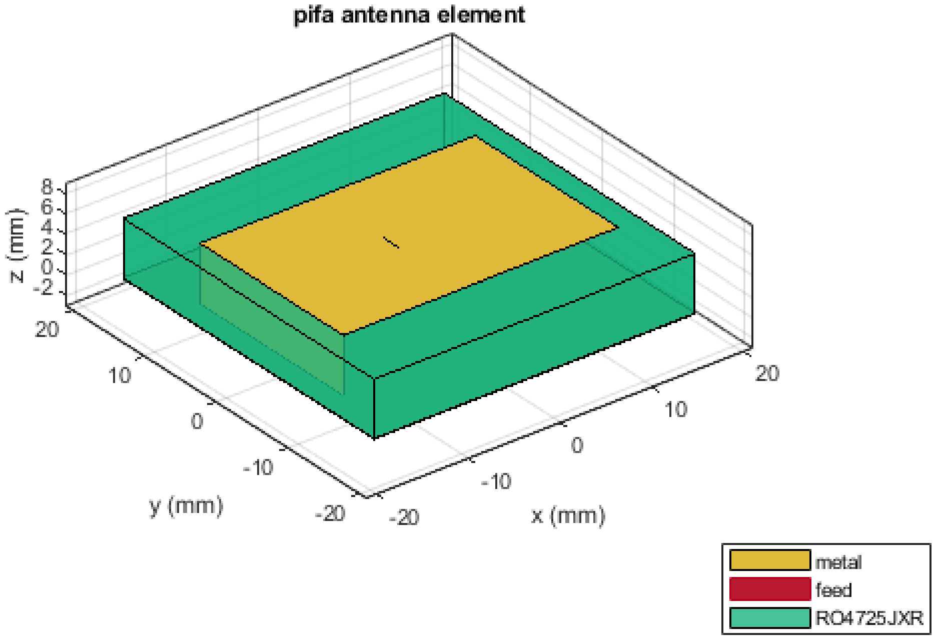
```
   Output : Directivity
Frequency : 2.3 GHz
Max value : 1.09 dBi
 Min value : -2.38 dBi
   Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

Impedance of PIFA Antenna

Create a PIFA antenna using a dielectric substrate **'RO4725JXR'**.

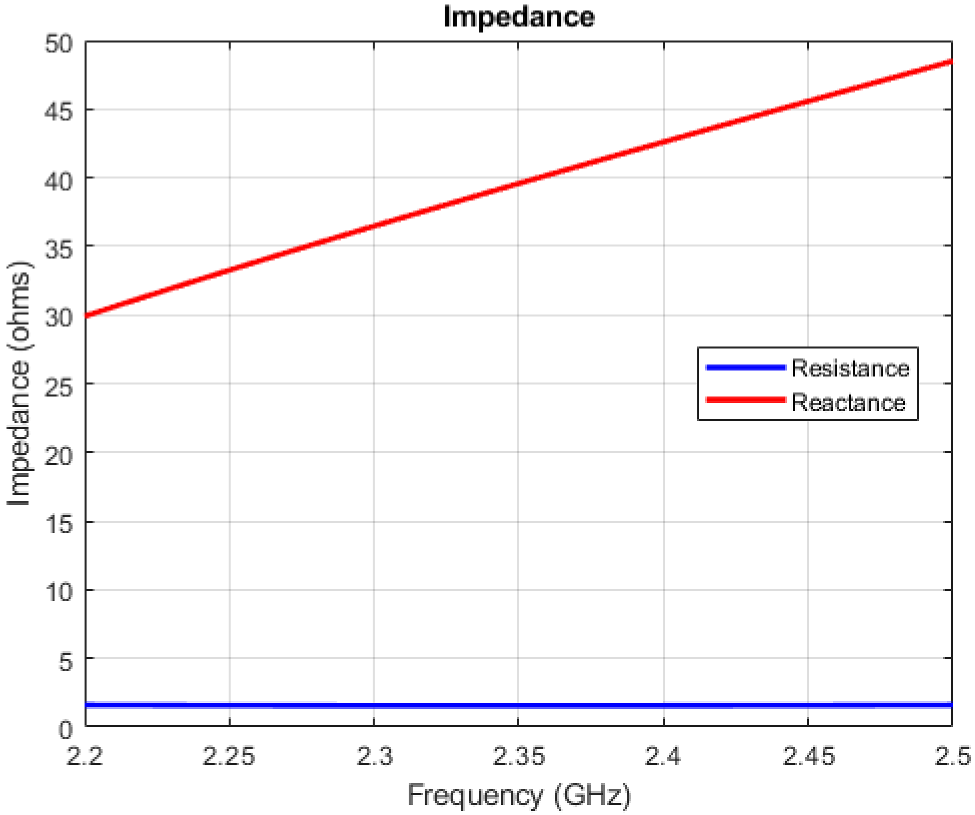
```
d = dielectric('RO4725JXR');
pf = pifa('Length',30e-3, 'Width',20e-3,'Height',0.0060, 'GroundPlaneLength',35e-3, ...
         'GroundPlaneWidth', 35e-3,'Substrate',d)
show(pf)
```

```
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));
```



References

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

See Also

`invertedF` | `invertedL` | `patchMicrostrip`

Topics

"Rotate Antennas and Arrays"

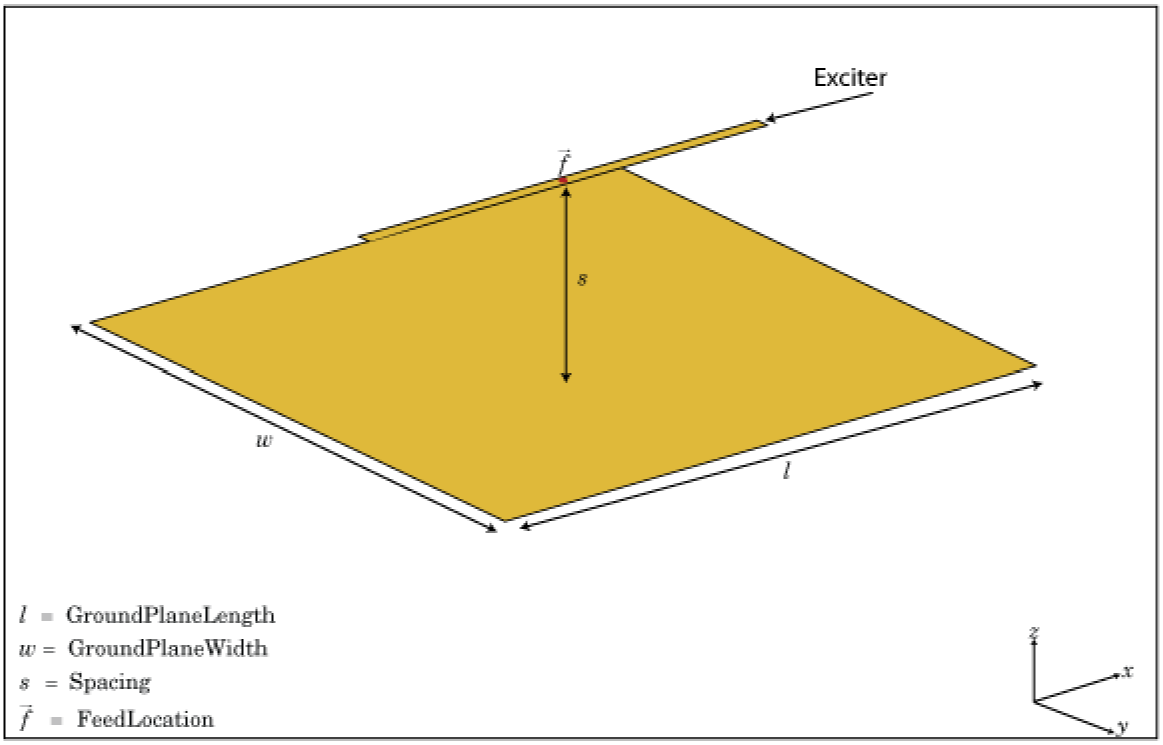
Introduced in R2015a

reflector

Create reflector-backed antenna

Description

The `reflector` object is a reflector-backed antenna on the X-Y-Z 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 type used as exciter

`dipole` (default) | object

Antenna type used as an exciter, specified as an object. Except reflector and cavity antenna elements, you can use all the single elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',dipole`

`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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

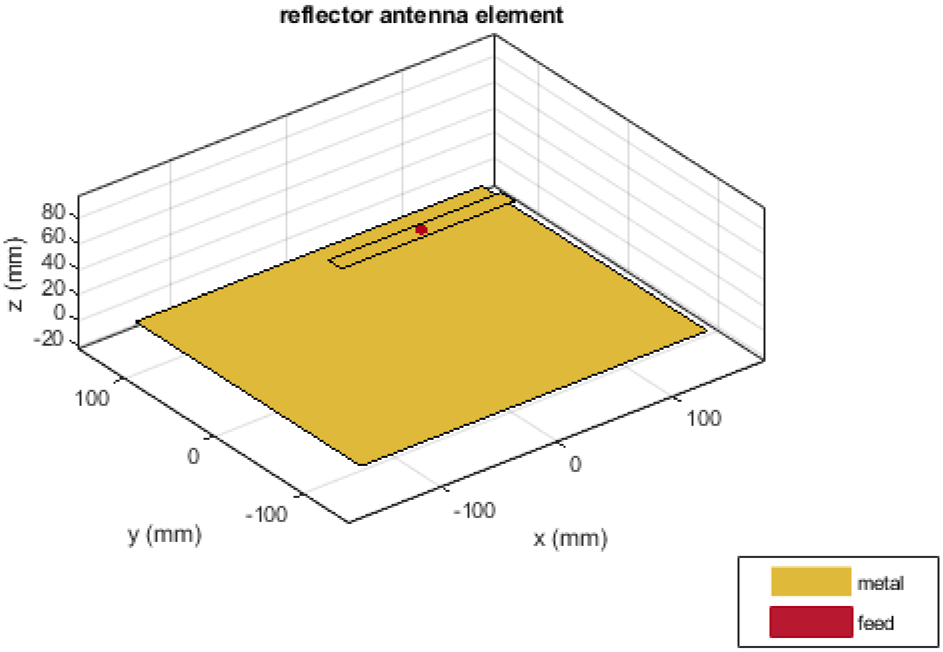
Create and View Reflector-Backed Dipole Antenna

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

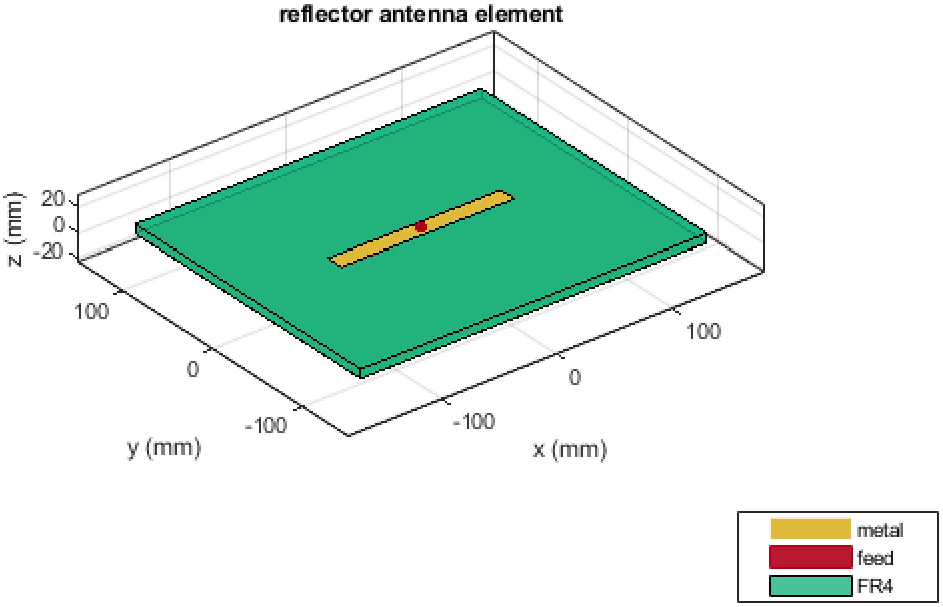
```
show(rf)
```



Radiation Pattern of Reflector Backed Antenna

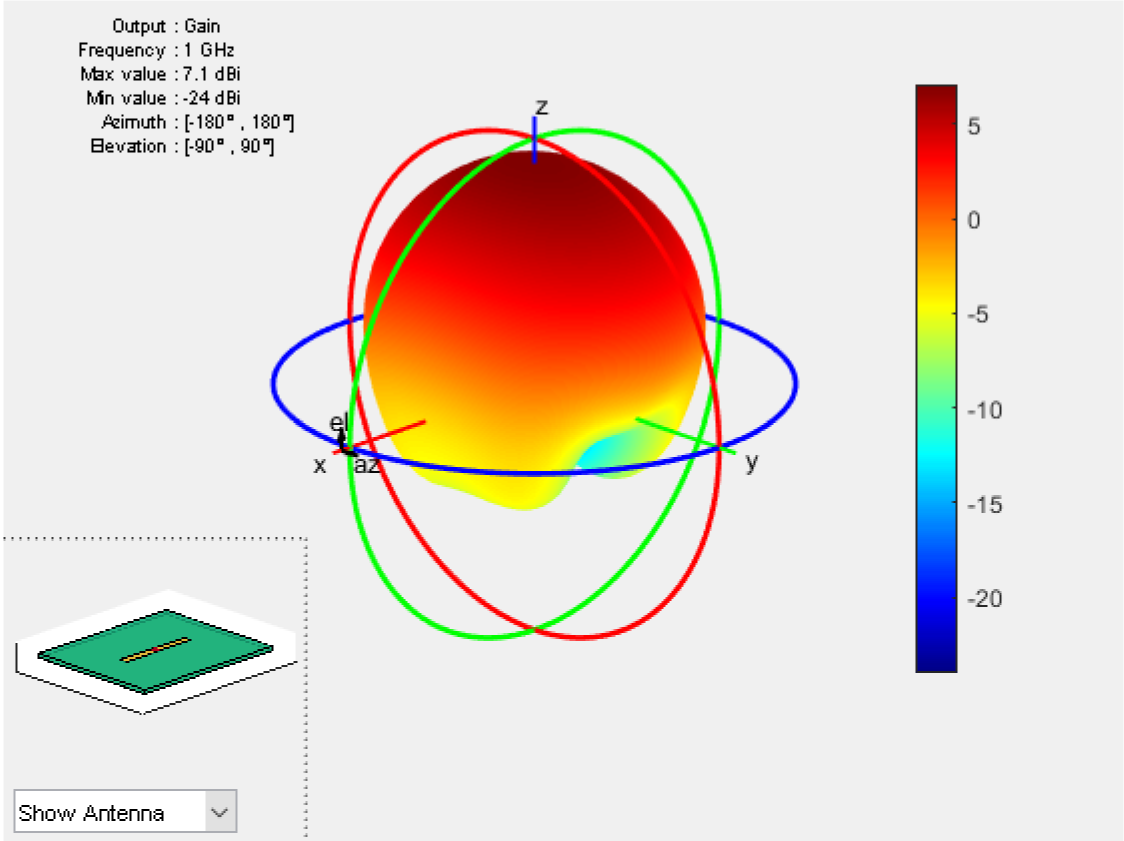
Create a reflector backed dipole antenna using a dielectric substrate **'FR4'**.

```
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)
```



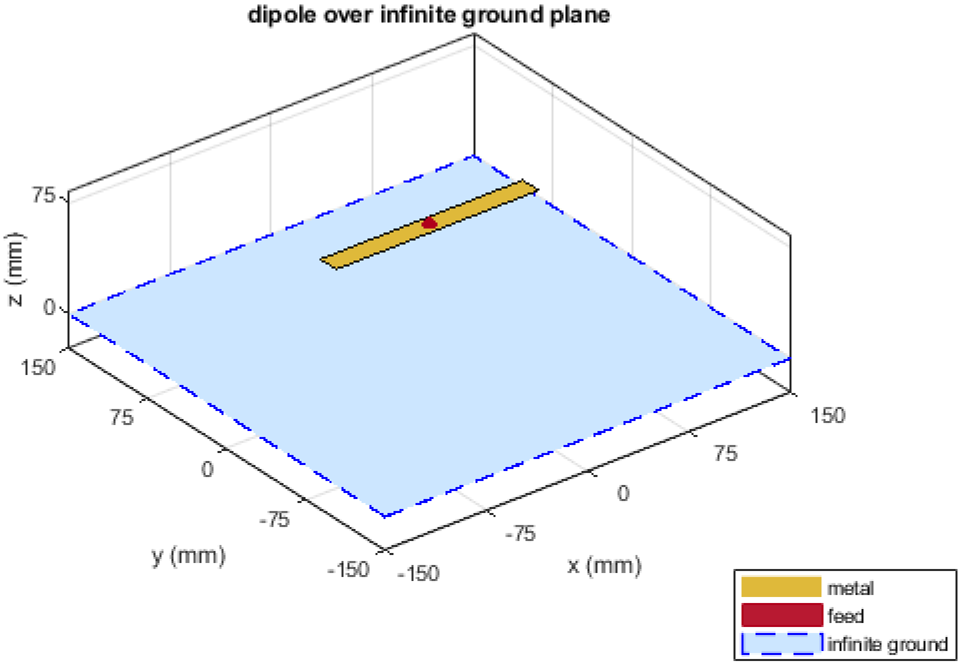
Create Reflector-Backed Antenna Over Infinite Ground Plane

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

```
show(rf)
```



Antenna On Dielectric Substrate - Compare Gain Values

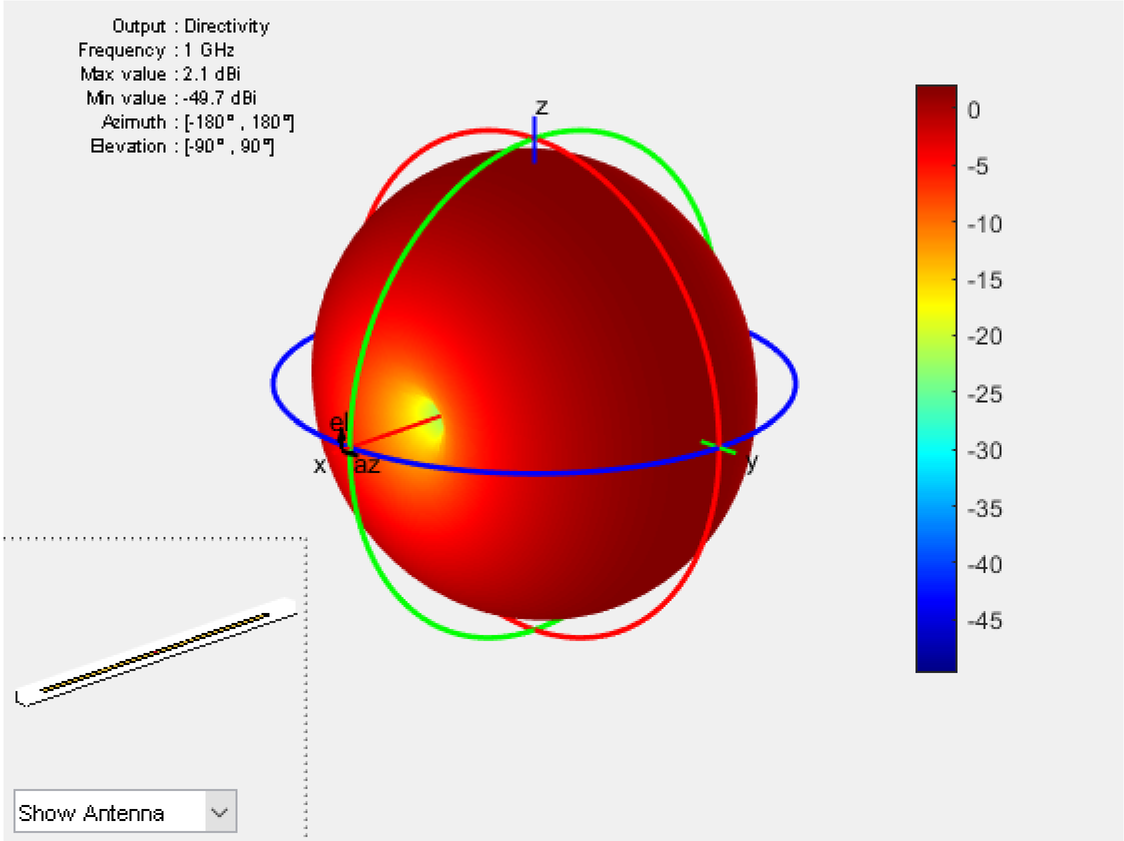
Compare the gain values of a dipole antenna in free space and dipole antenna on a substrate.

Design a dipole antenna at 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 1GHz.

```
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);
```

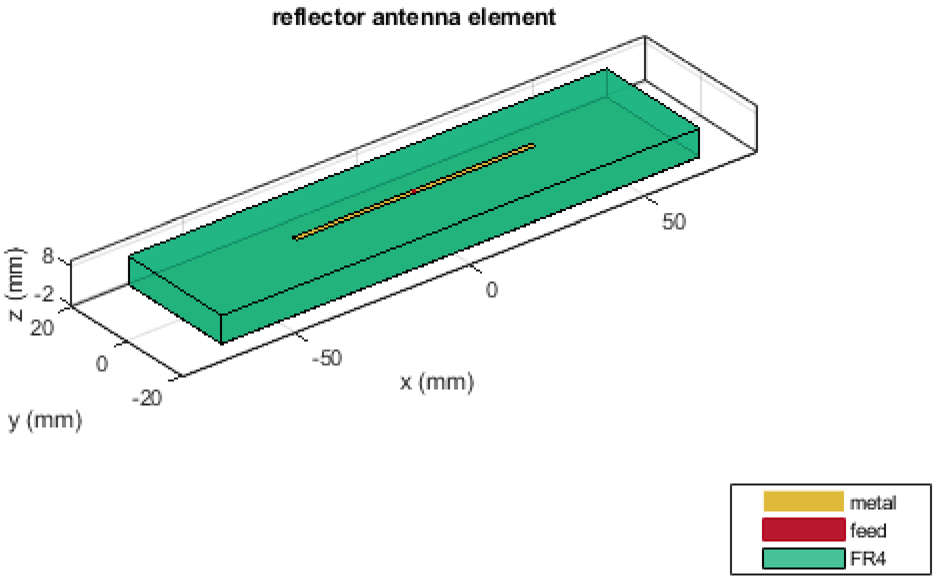
Adjust the length of the dipole based on the wavelength.

```
d.Length = lambda_d/2;
d.Width = d.Length/l_by_w;
```

Design a reflector at 1 GHz with the dipole as the excitor and FR4 as the substrate.

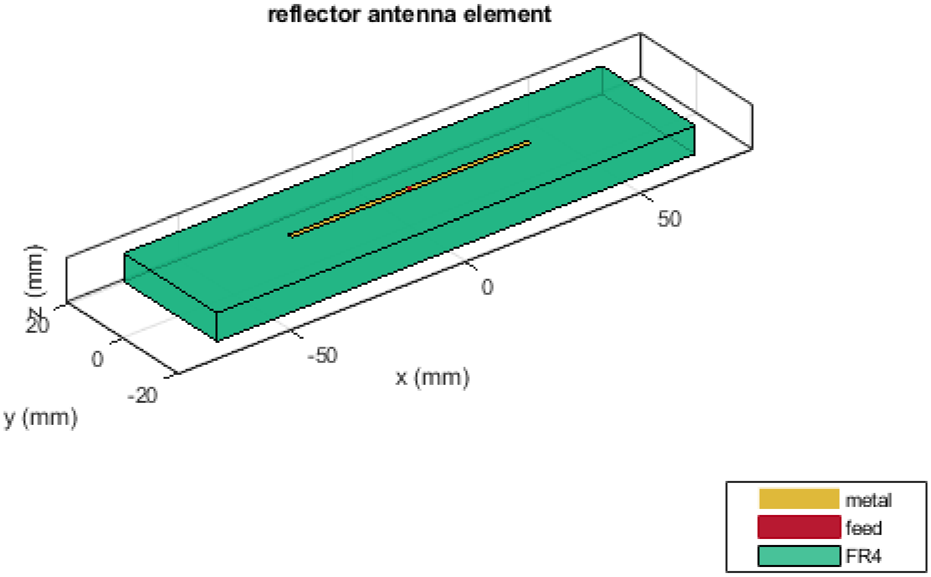
```
rf = design(reflector,1e9);
rf = reflector('Exciter',d,'Spacing',7.5e-3,'Substrate',t);
```

```
rf.GroundPlaneLength = lambda_d;
rf.GroundPlaneWidth = lambda_d/4;
figure
show(rf)
```



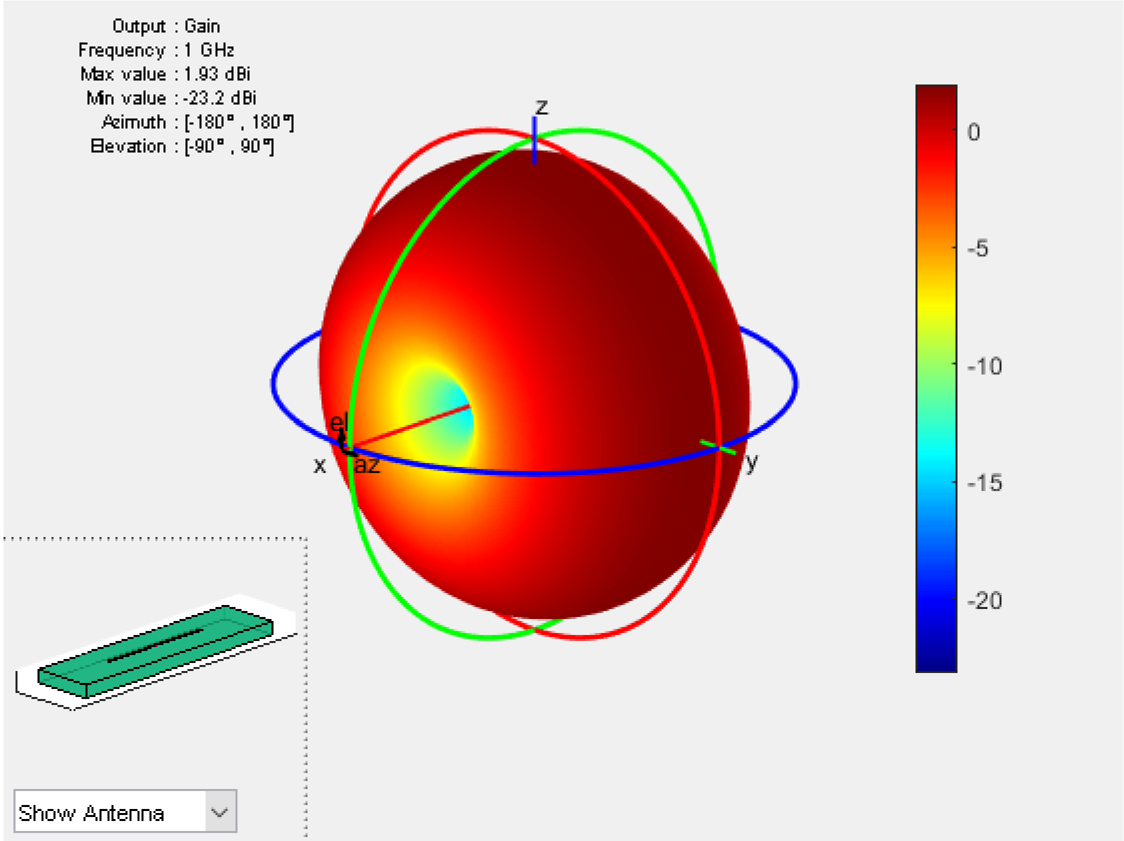
Remove the groundplane for plotting the gain of the dipole on the substrate.

```
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 : 1.93 dBi
Min value : -23.2 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

References

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

See Also

`cavity` | `spiralArchimedean` | `spiralEquiangular`

Topics

“Rotate Antennas and Arrays”

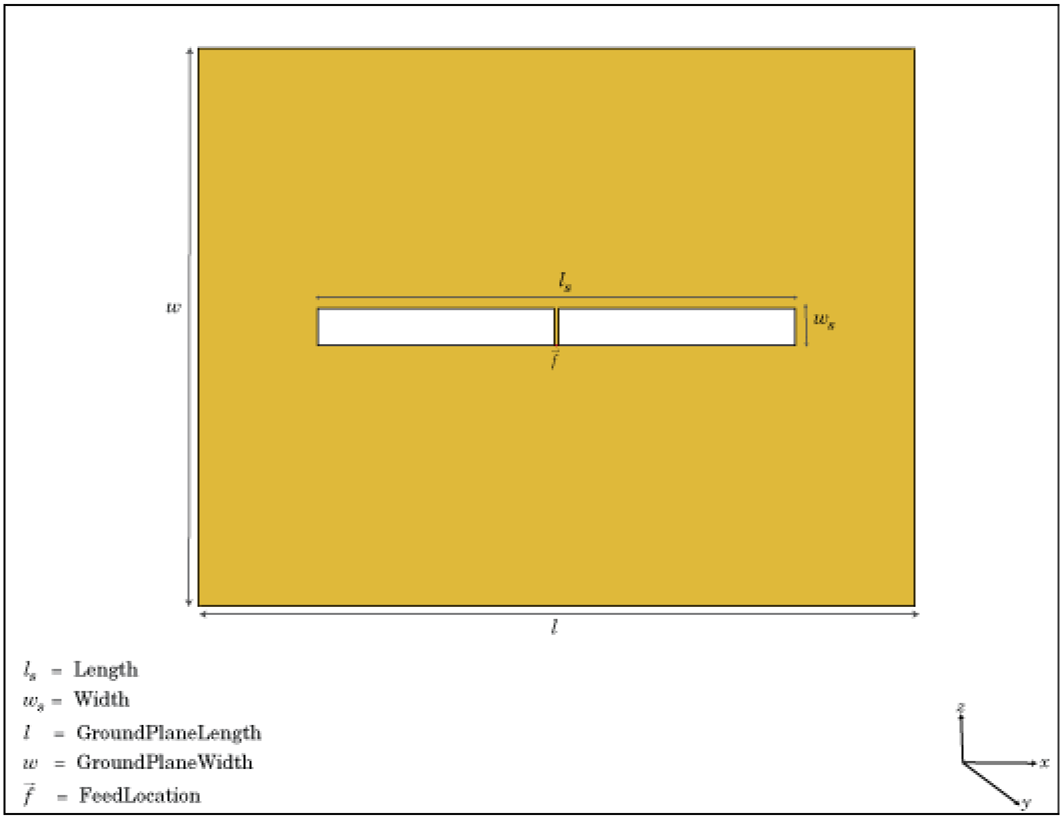
Introduced in R2015a

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`

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: `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: `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 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 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 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'`

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

Object Functions

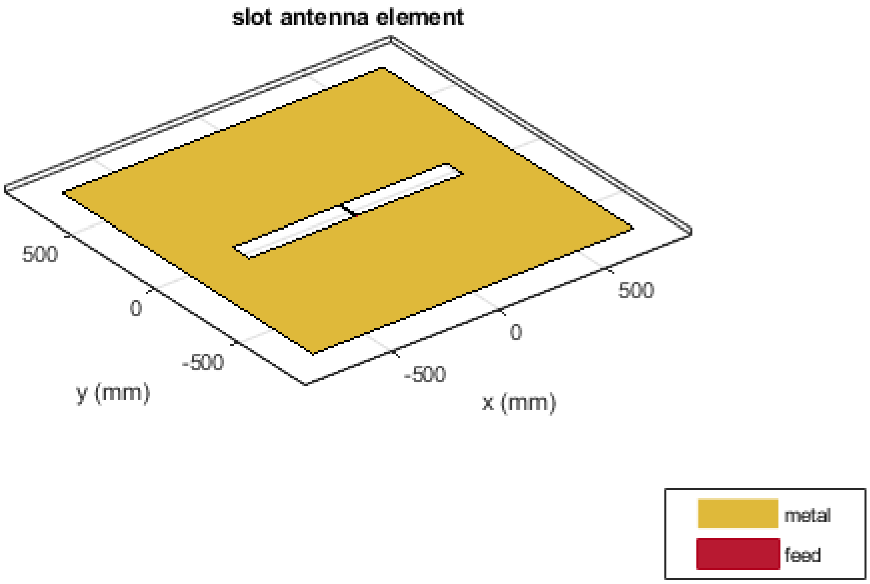
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

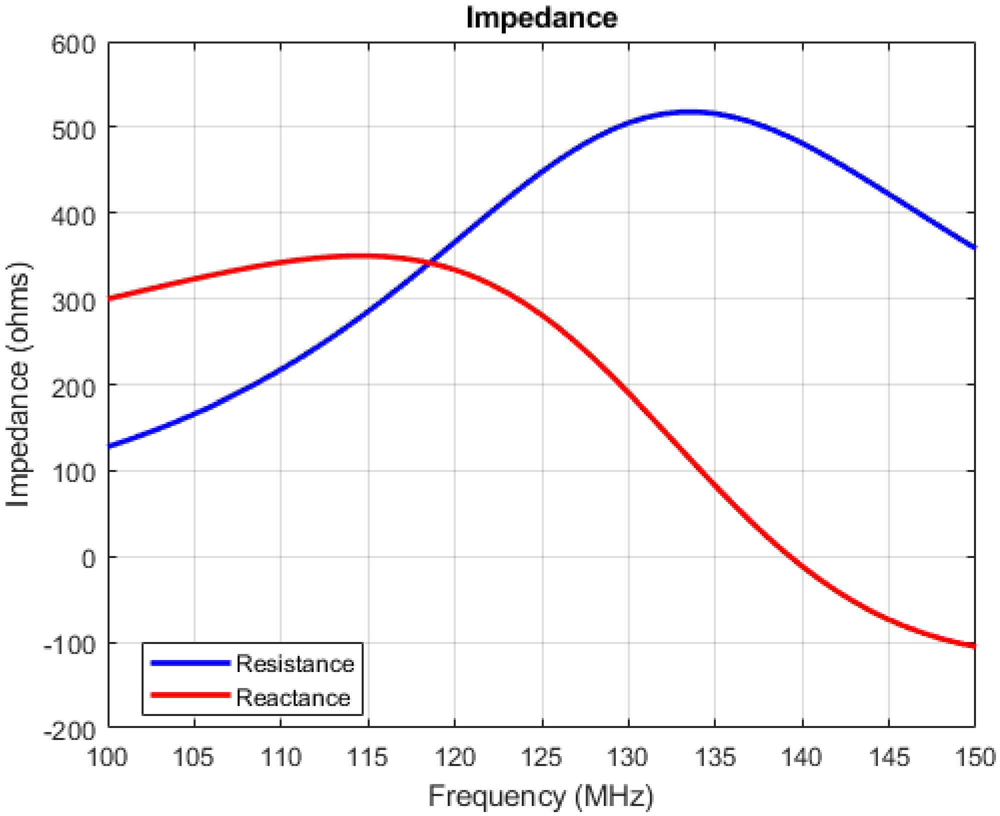
```
s = slot('Length',1,'Width',0.1);
show(s)
```



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));
```



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”

Introduced in R2015a

spiralArchimedean

Create Archimedean spiral antenna

Description

The `spiralArchimedean` object creates a planar Archimedean spiral antenna on the X-Y 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.

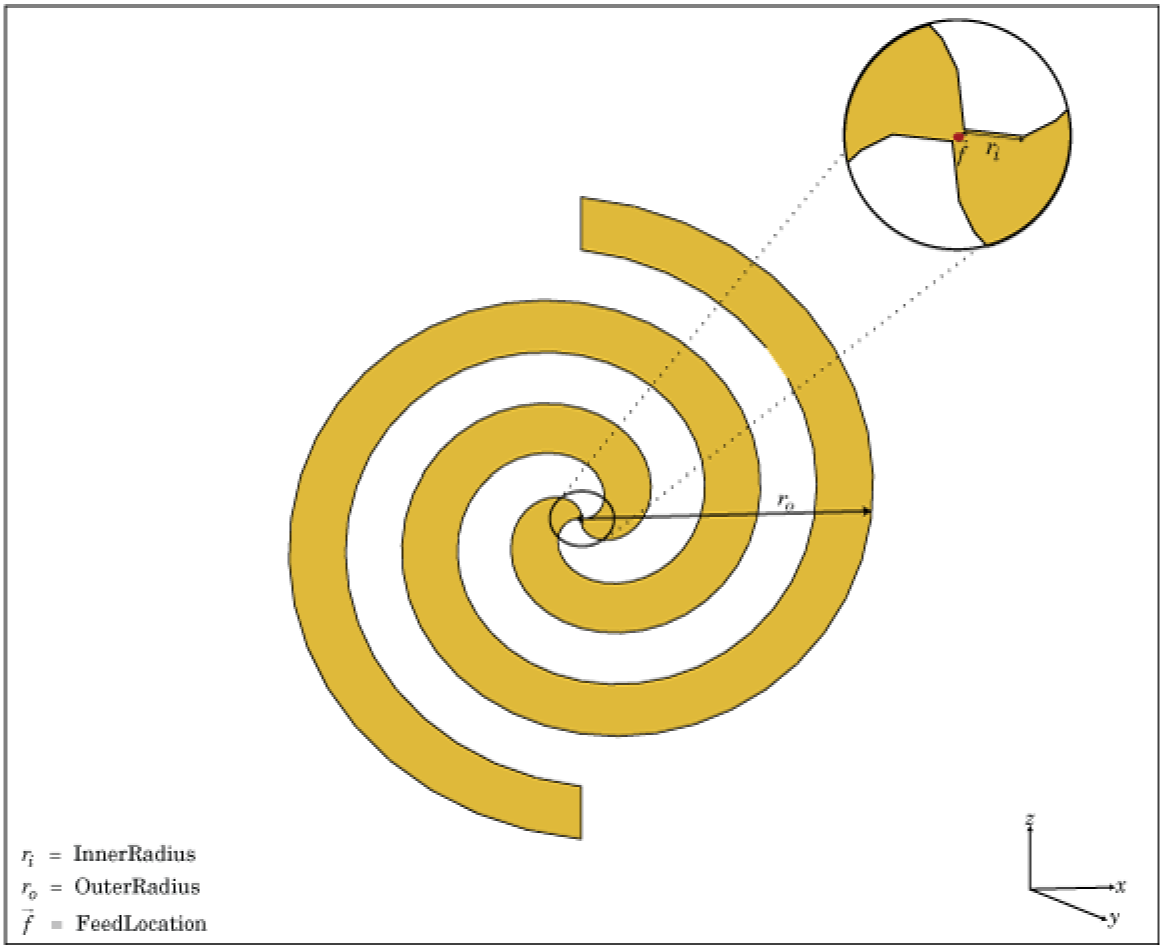
The equation of the Archimedean spiral is:

$$r = r_0 + a\phi$$

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-complimentary 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 X-Y 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`

Load — Lumped elements

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

Lumped elements added to the spiral antenna feed, specified as a lumped element object handle. 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 handle 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: `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 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 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 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'`

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

Object Functions

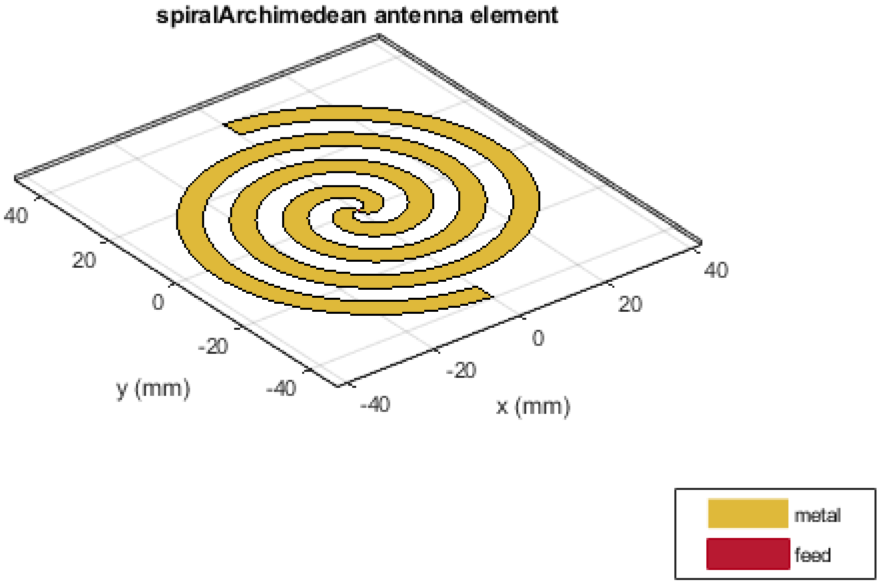
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

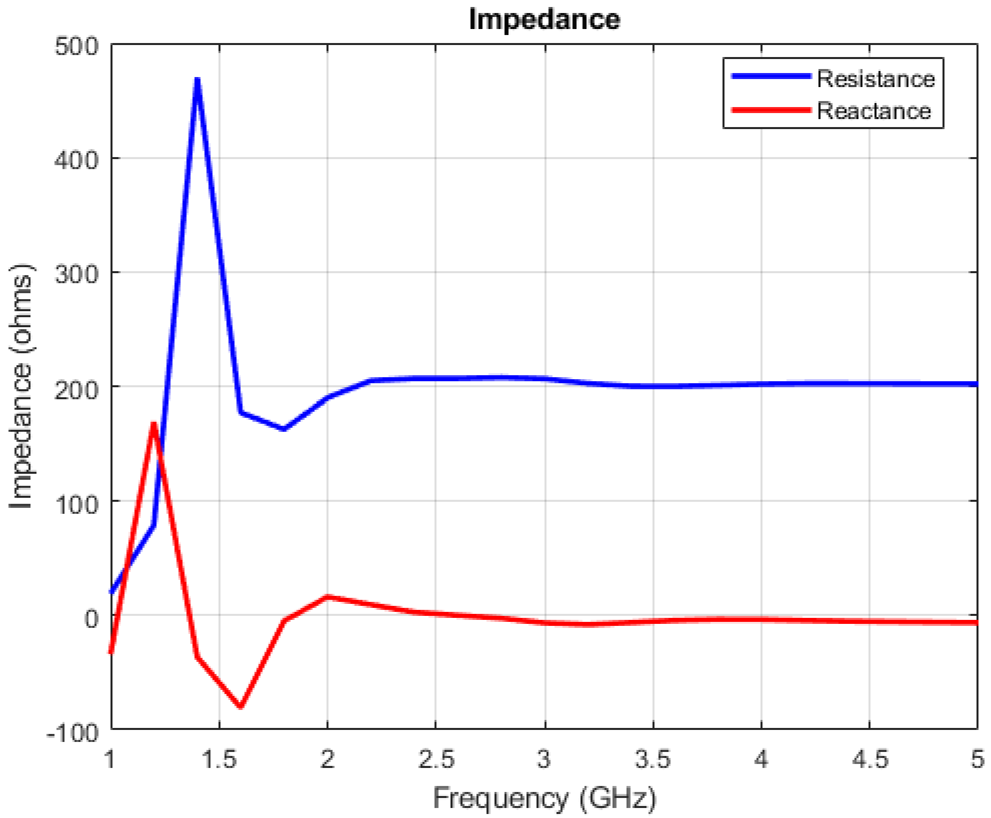
```
sa = spiralArchimedean('Turns',2, 'InnerRadius',1e-3, 'OuterRadius',40e-3);
show(sa)
```



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));
```



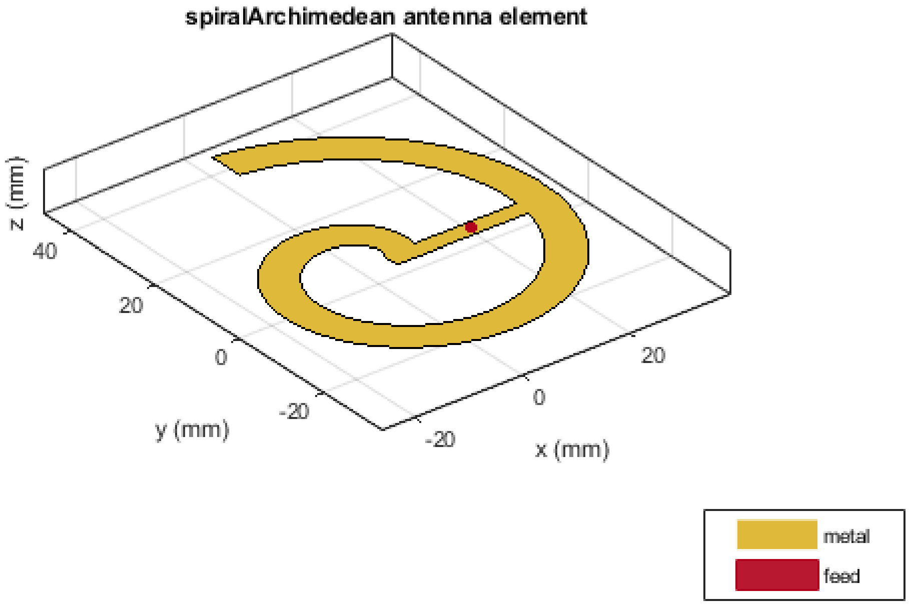
Single-Arm Archimedean Spiral

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

```
show(ant)
```



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

`helix` | `spiralEquiangular` | `yagiUda`

Topics

“Rotate Antennas and Arrays”

Introduced in R2015a

spiralEquiangular

Create equiangular spiral antenna

Description

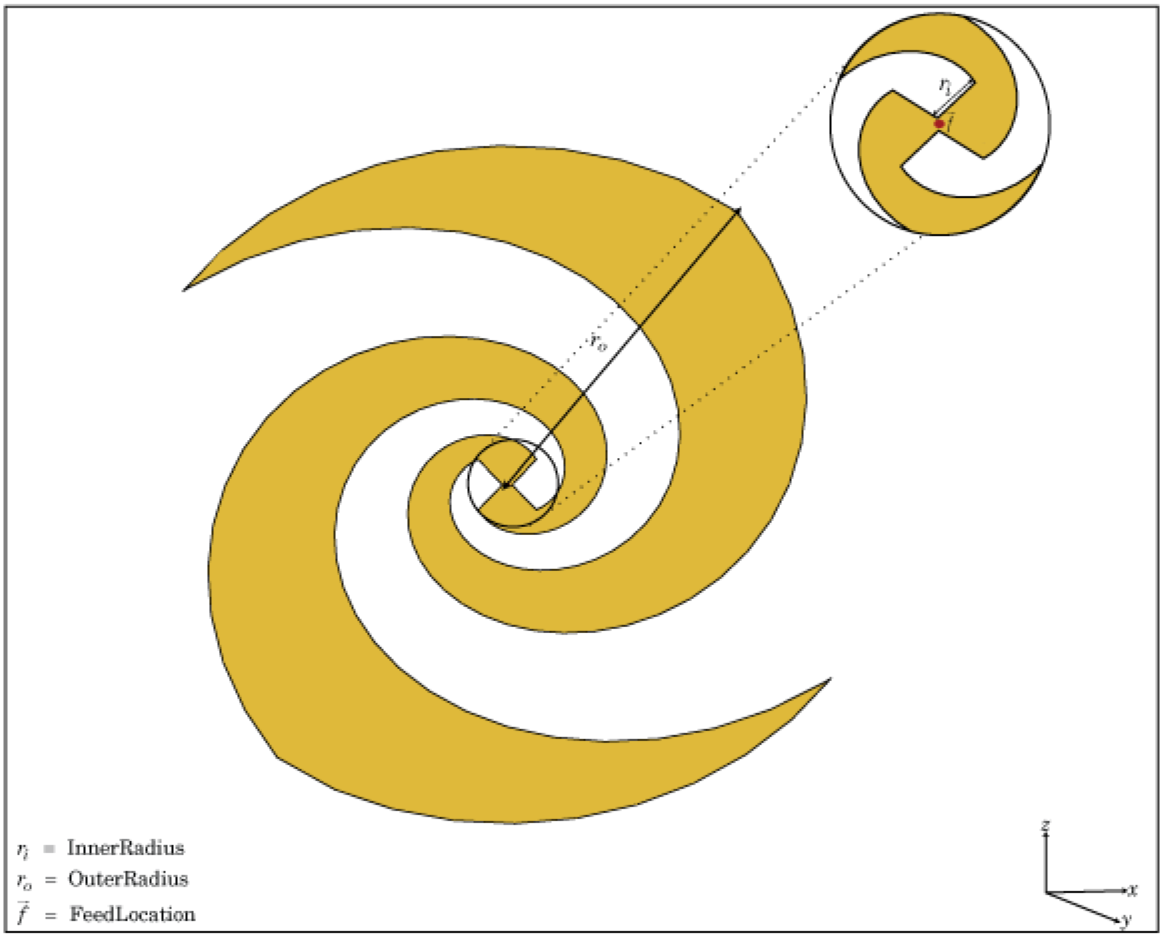
The `spiralEquiangular` object is a planar equiangular spiral antenna on the X-Y plane. The equiangular spiral is always center fed and has two arms. The field characteristics of the antenna are frequency independent. A realizable spiral has finite limits on the feeding region and the outermost point 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 X-Y 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

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: 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: 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 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 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 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'`

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

Object Functions

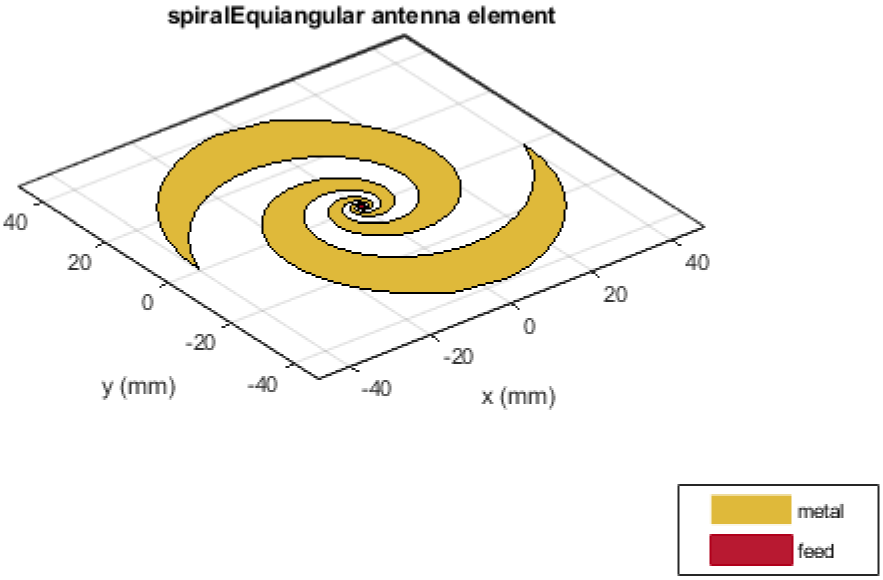
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

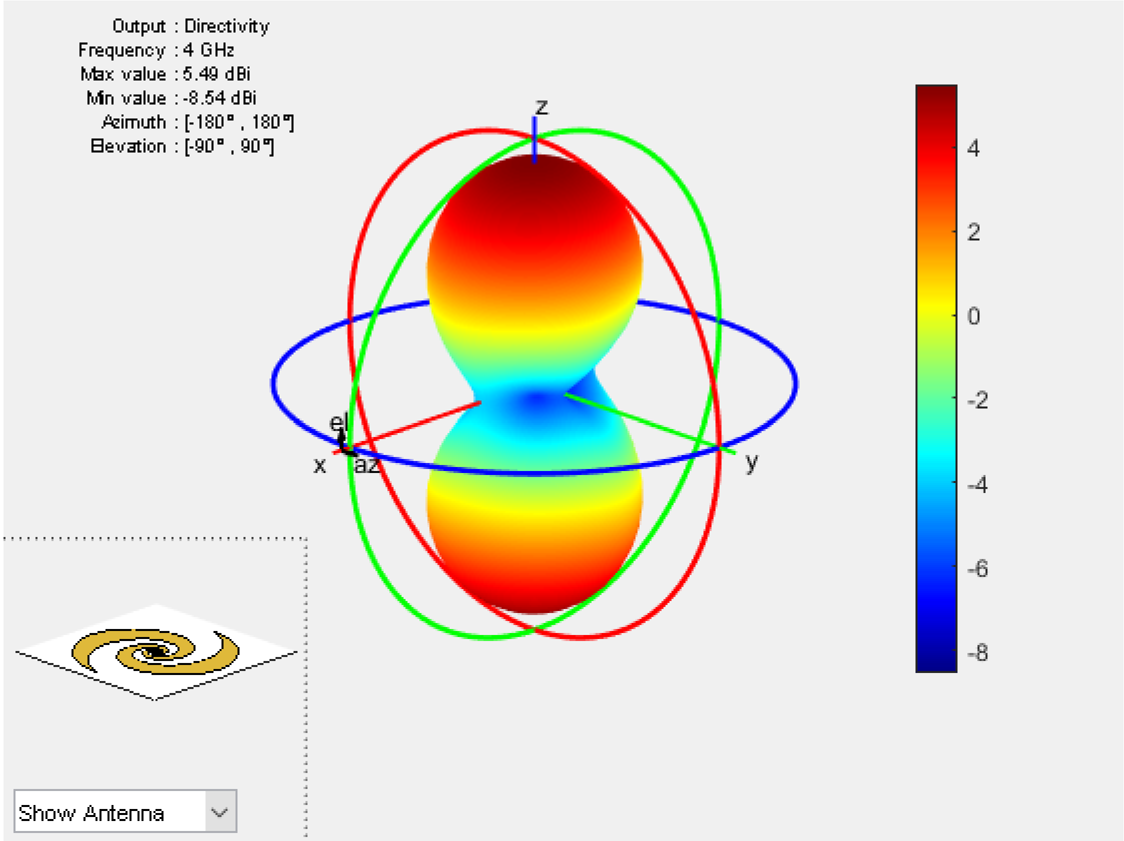
```
se = spiralEquiangular('GrowthRate',0.35, 'InnerRadius',0.65e-3,    ...
                          'OuterRadius',40e-3);
show(se)
```



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 : 5.49 dBi
Min value : -8.54 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

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] Violates, John *Antenna Engineering Handbook*, 4th Ed., McGraw-Hill.

See Also

`cavity` | `spiralArchimedean` | `vivaldi`

Topics

“Rotate Antennas and Arrays”

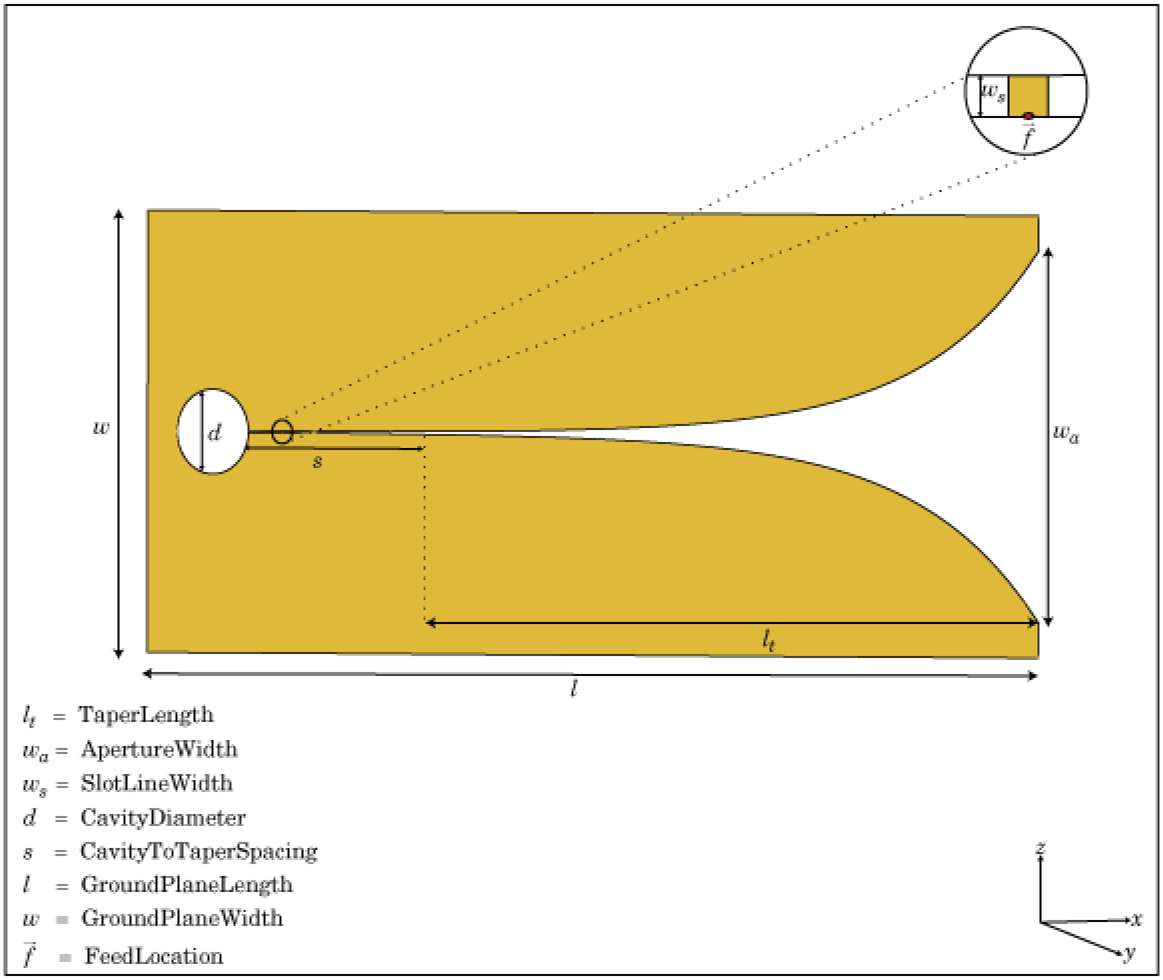
Introduced in R2015a

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 retains 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 (default) | scalar

Distance from feed along x-axis, specified a scalar in meters.

Example: 'FeedOffset',3

Data Types: double

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 origin. For more information, see lumpedElement.

Example: 'Load',lumpedelement. lumpedelement is the object handle 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: 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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

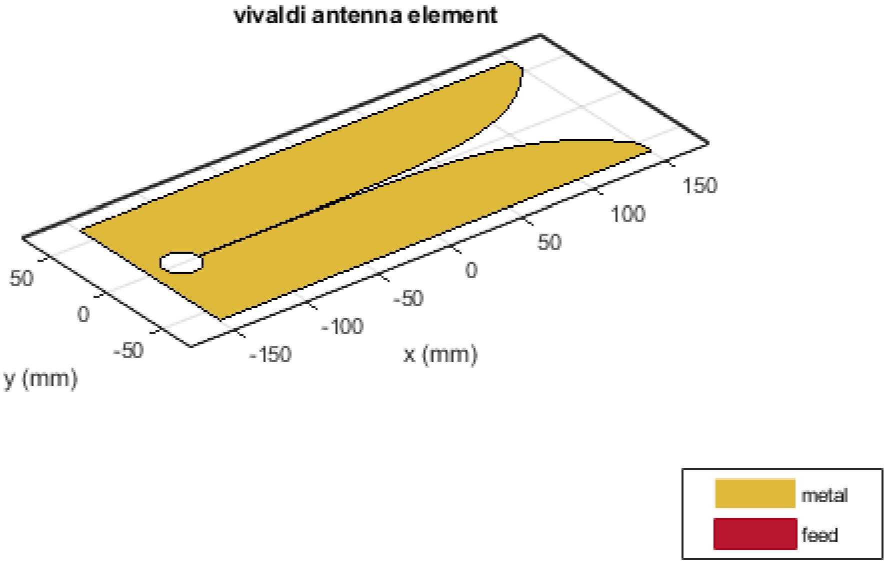
Create and View Vivaldi Antenna

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

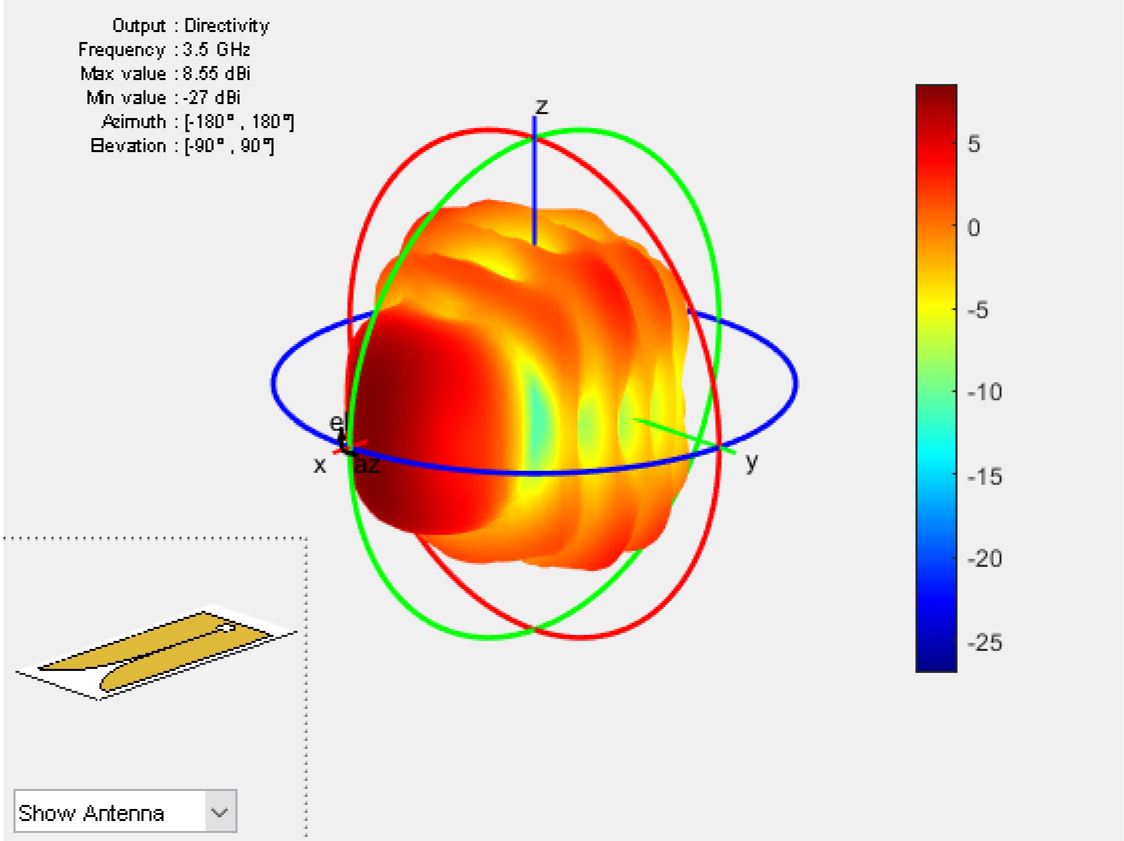
```
show(vi);
```



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 : 8.55 dBi
Min value : -27 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

References

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

See Also

`slot` | `spiralArchimedean` | `yagiUda`

Topics

"Rotate Antennas and Arrays"

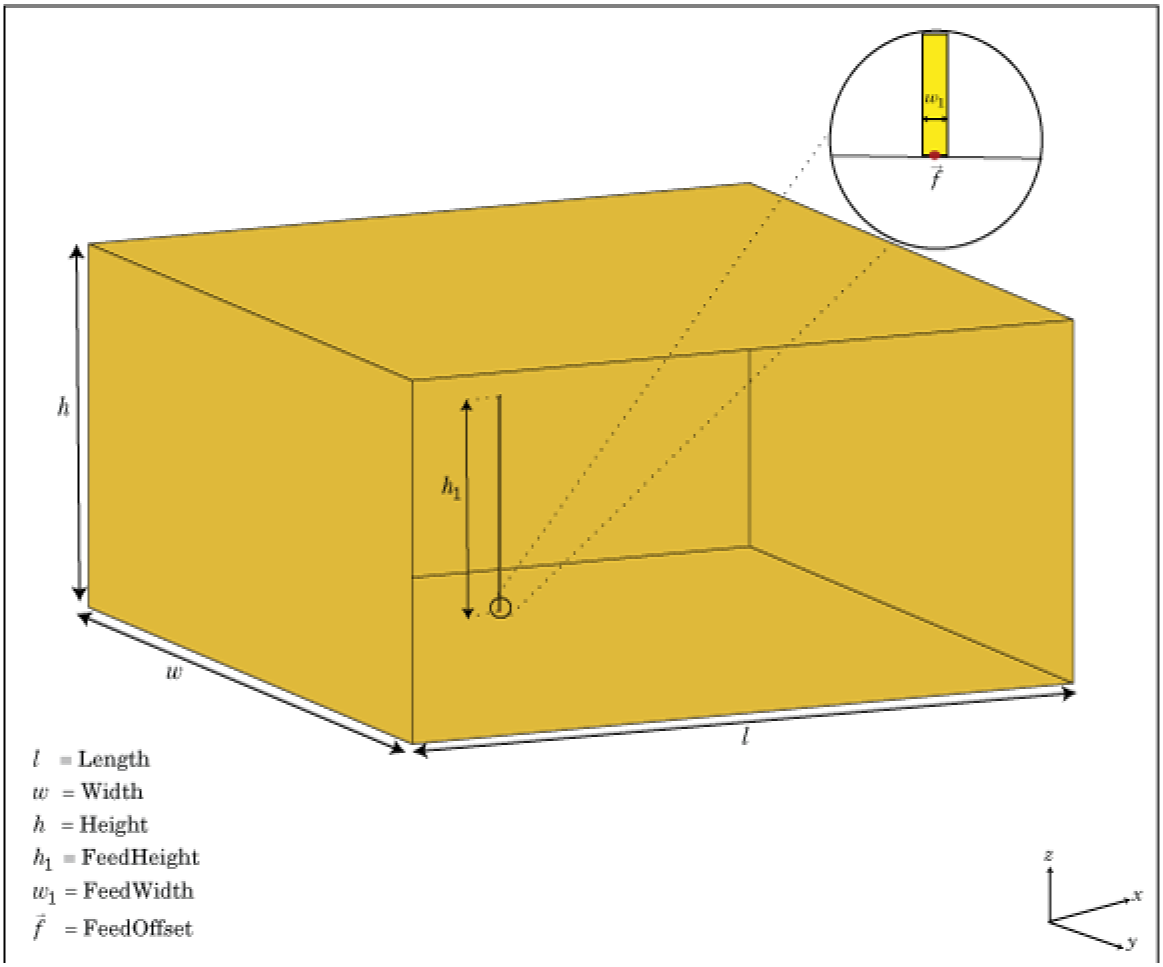
Introduced in R2015a

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 X-Y plane.

Example: 'FeedOffset',[–0.0070 0.01]

Data Types: double

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: 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: 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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

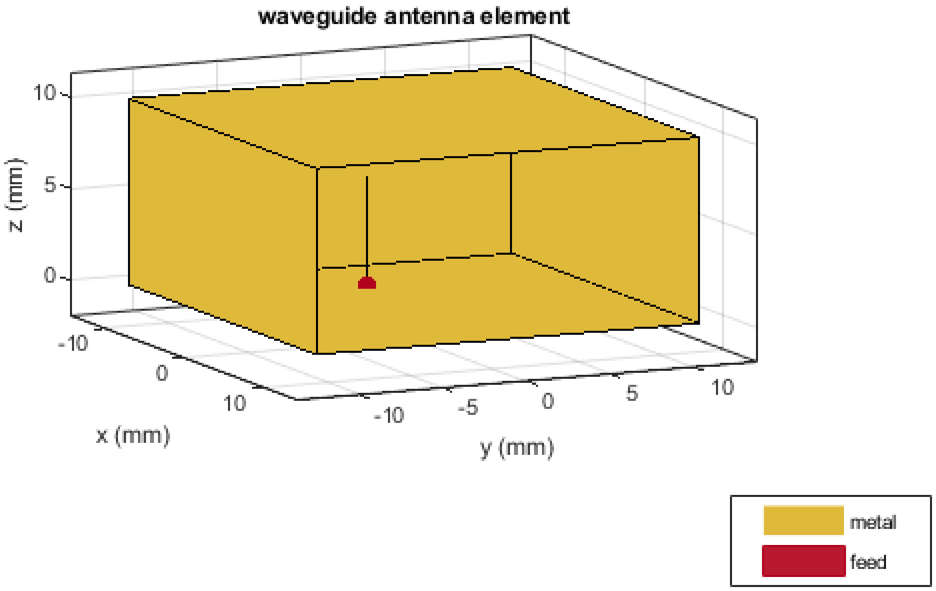
Default Rectangular Waveguide

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

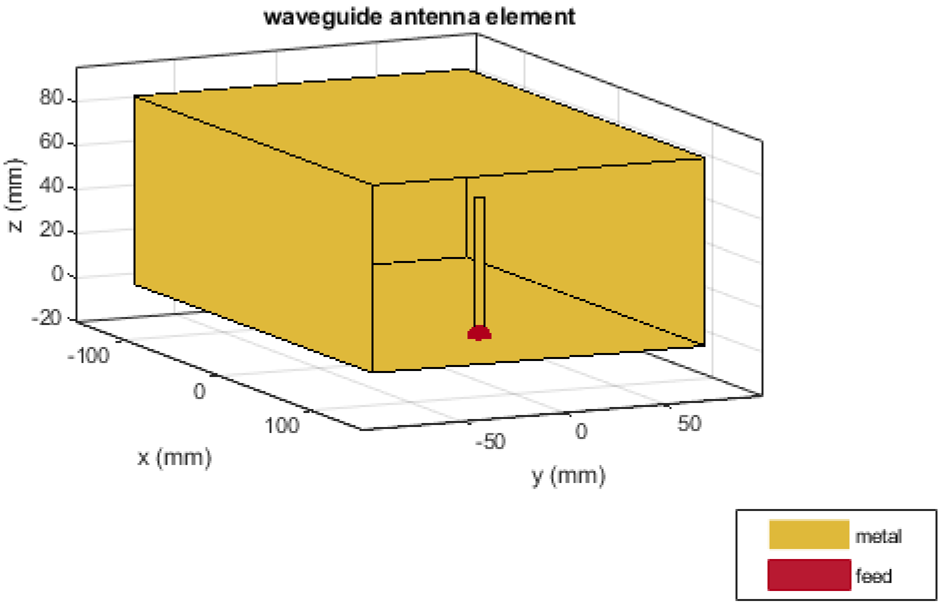
```
show(wg)
```



Radiation Pattern of WR-650 Rectangular Waveguide

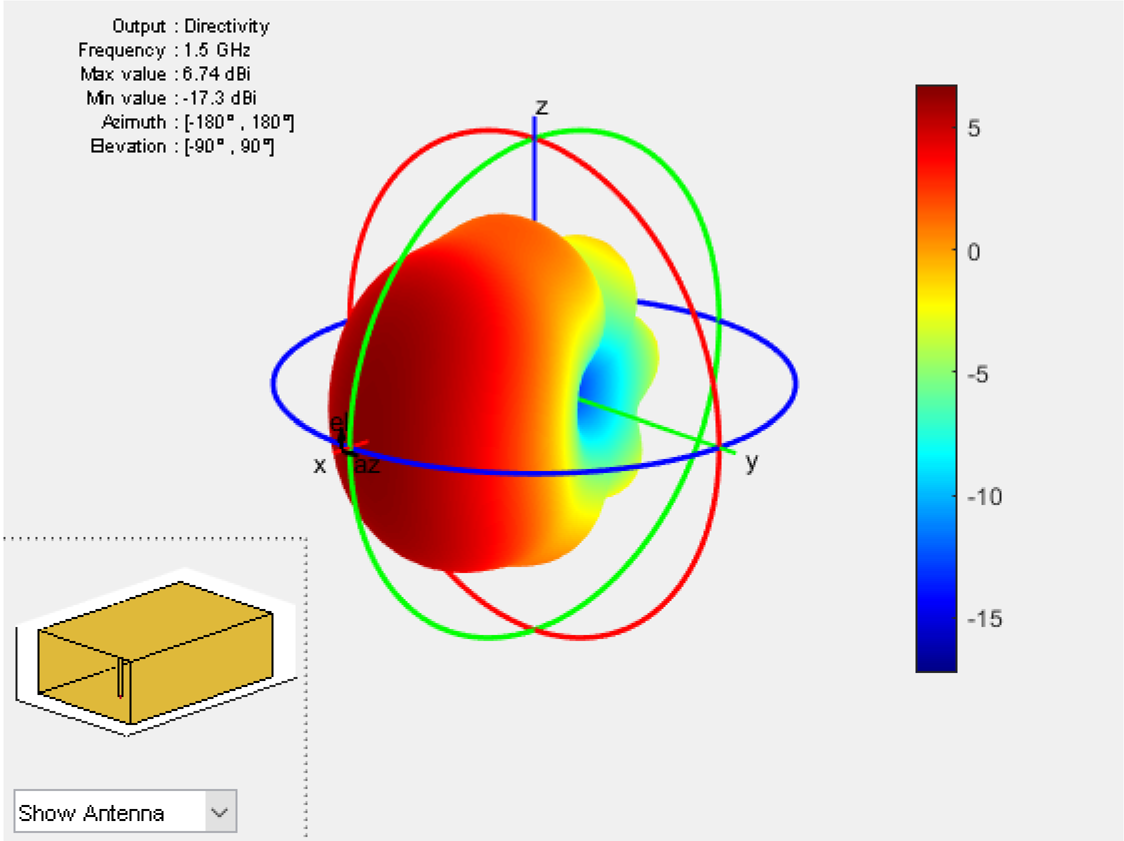
Create a WR-650 rectangular waveguide and display it.

```
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 : 6.74 dBi
 Min value : -17.3 dBi
   Azimuth : [-180° , 180°]
  Elevation : [-90° , 90°]
```

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”

Introduced in R2016a

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.

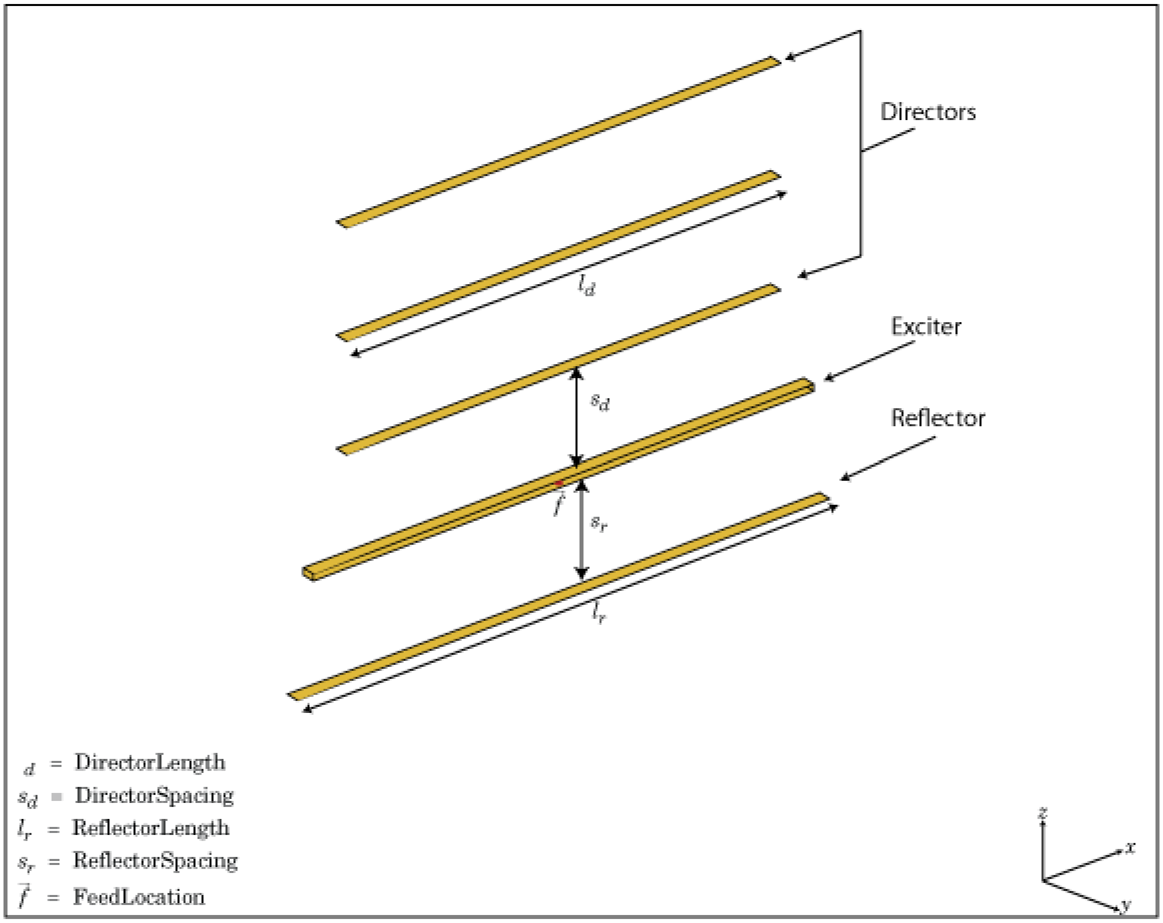
The exciter, reflector, and directors have equal widths and are related to the diameter of an equivalent cylindrical structure by the equation

$$w = 2d = 4r$$

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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array

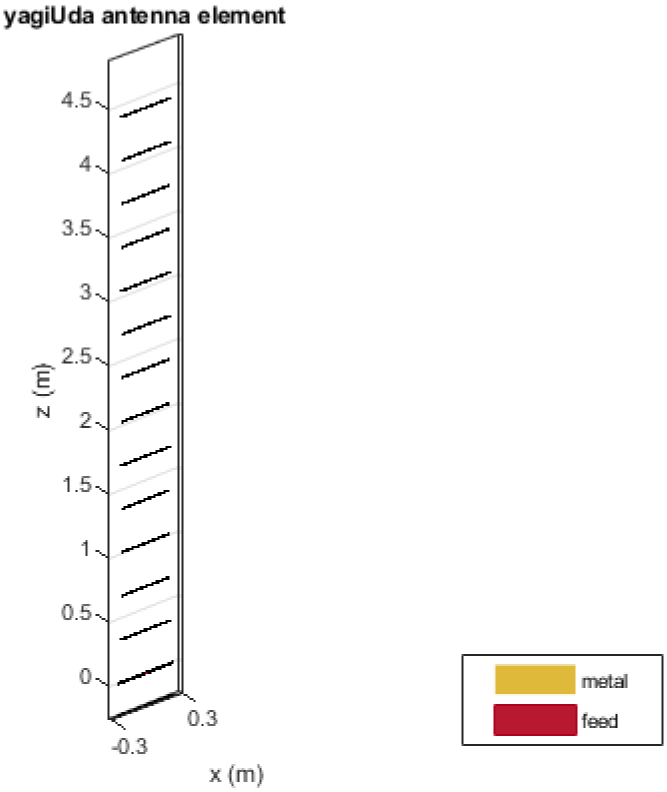
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at specified frequency
EHfields	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 or 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	S-parameter object
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.

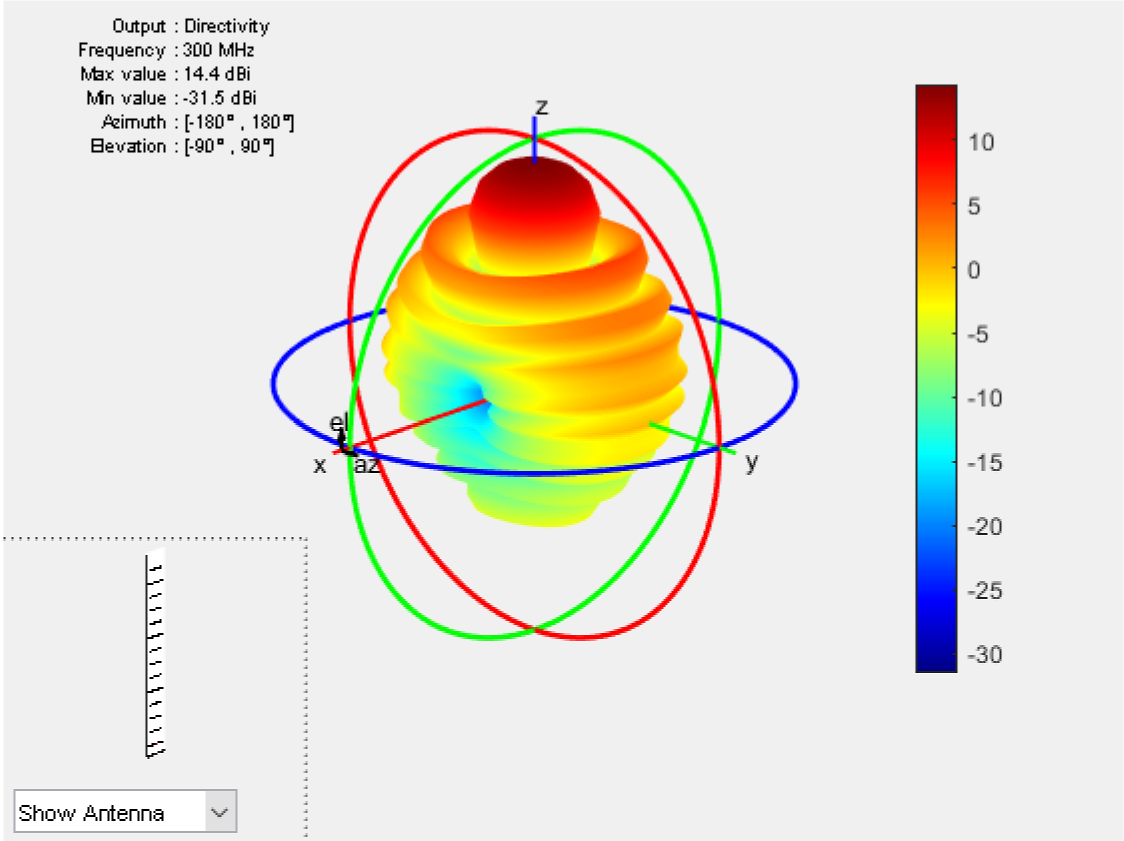
```
y = yagiUda('NumDirectors',13);
show(y)
```



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)
```



Output : Directivity
Frequency : 300 MHz
Max value : 14.4 dBi
Min value : -31.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

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
```

References

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

See Also

`cylinder2strip` | `dipole` | `dipoleFolded` | `slot`

Topics

"Rotate Antennas and Arrays"

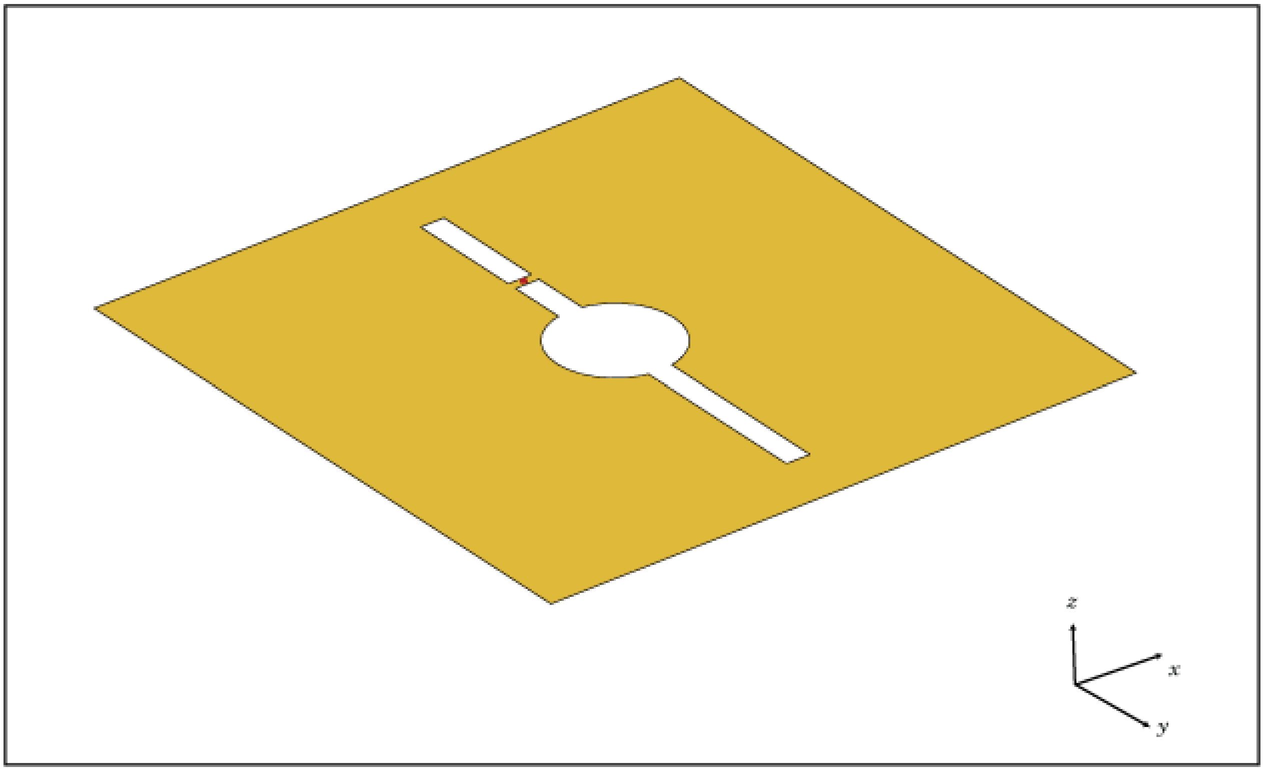
Introduced in R2015a

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 X-Y 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`

Load — Lumped elements

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

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

Example: `'Load'`, lumpedelement. `lumpedelement` is the object handle 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna
rcs	Calculate and plot radar cross section (RCS) of platform, antenna, or array

Examples

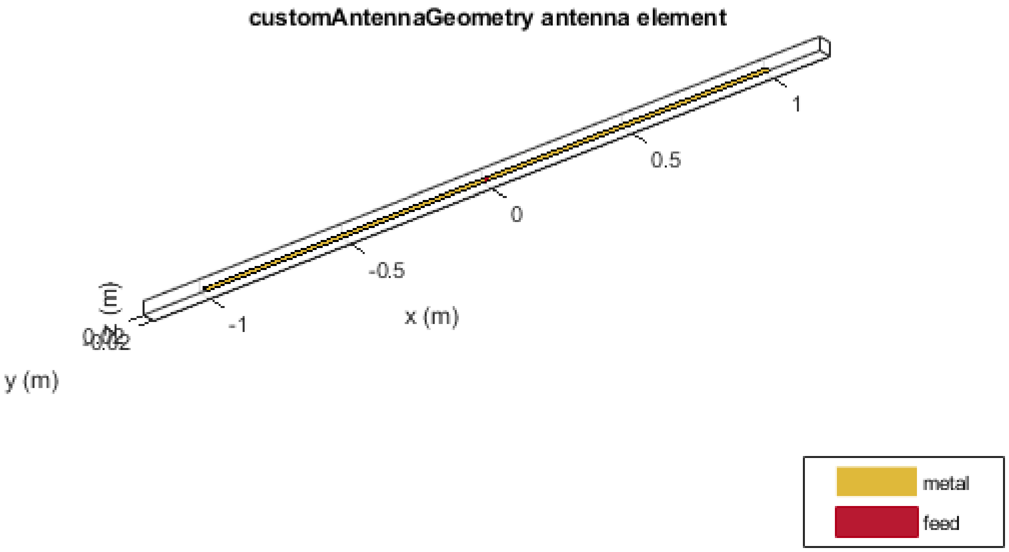
Custom Dipole Antenna

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
            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.

Make three rectangles of 0.5 m x 0.5 m, 0.02 m x 0.4 m and 0.03 m x 0.008 m.

```
pr = em.internal.makerectangle(0.5,0.5);
pr1 = em.internal.makerectangle(0.02,0.4);
pr2 = em.internal.makerectangle(0.03,0.008);
```

Make a circle of radius 0.05 m.

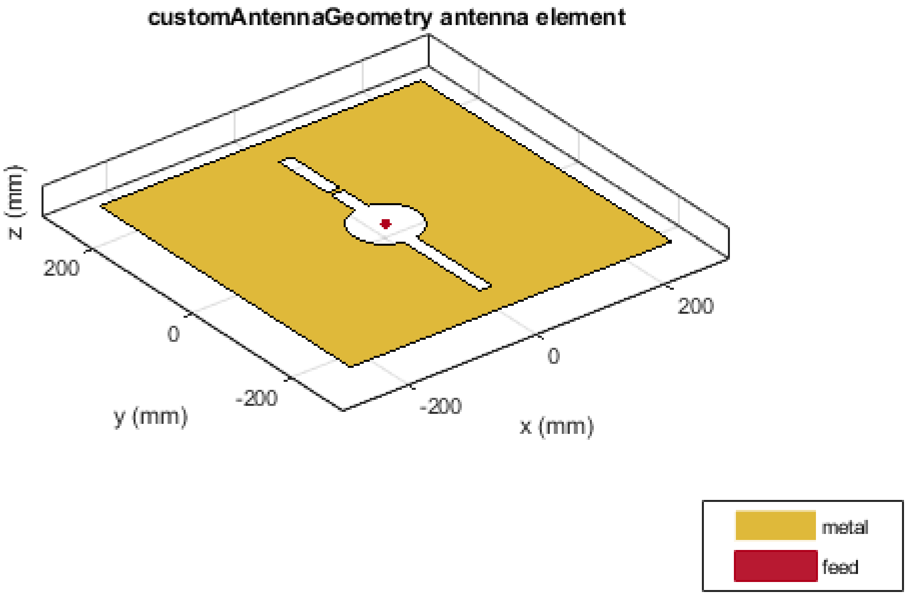
```
ph = em.internal.makecircle(0.05);
```

Translate the third rectangle to the X-Y plane using the coordinates [0 0.1 0].

```
pf = em.internal.translateshape(pr2,[0 0.1 0]);
```

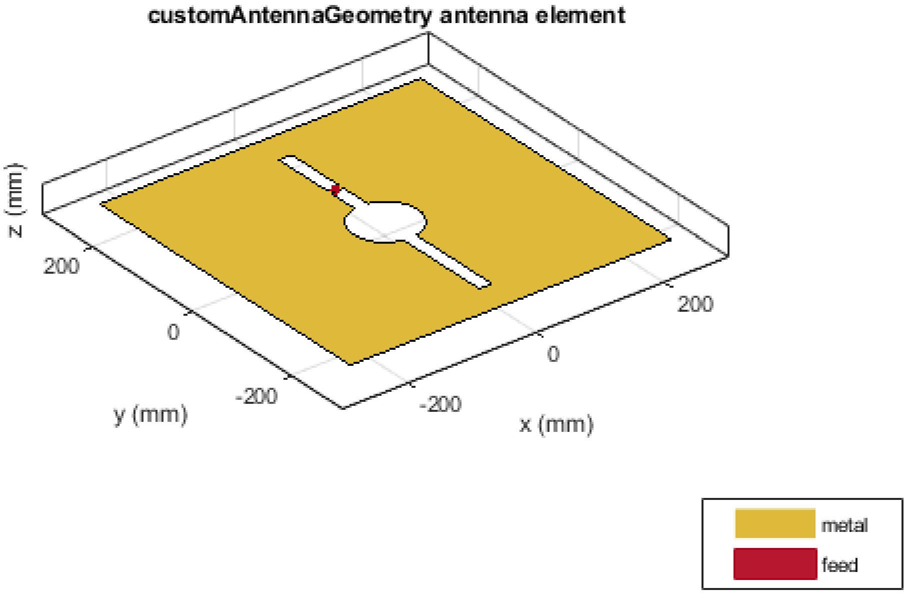
Create a custom slot antenna element using the specified boundary shapes. Transpose pr, ph, pr1, and pf to make sure the boundary inputs are column vector arrays.

```
c = customAntennaGeometry('Boundary',{pr',ph',pr1',pf'},...
    'Operation','P1-P2-P3+P4');
figure;
show(c);
```



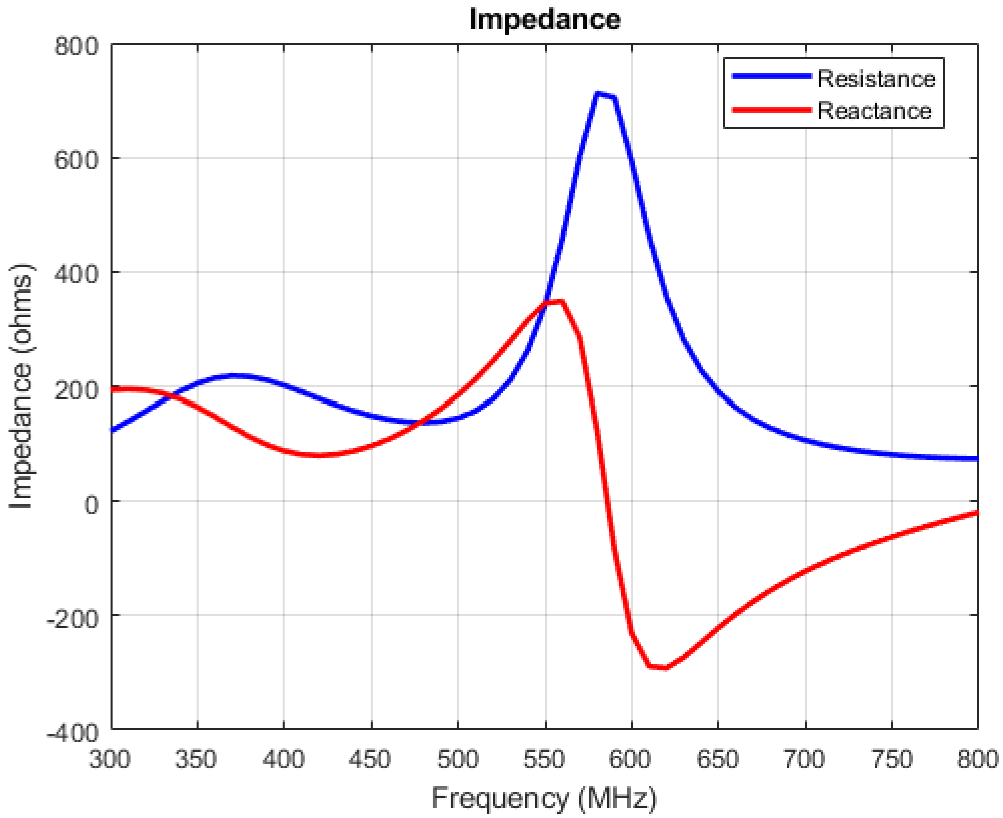
Move the feed location to new coordinates.

```
c.FeedLocation = [0,0.1,0];
figure;
show(c);
```



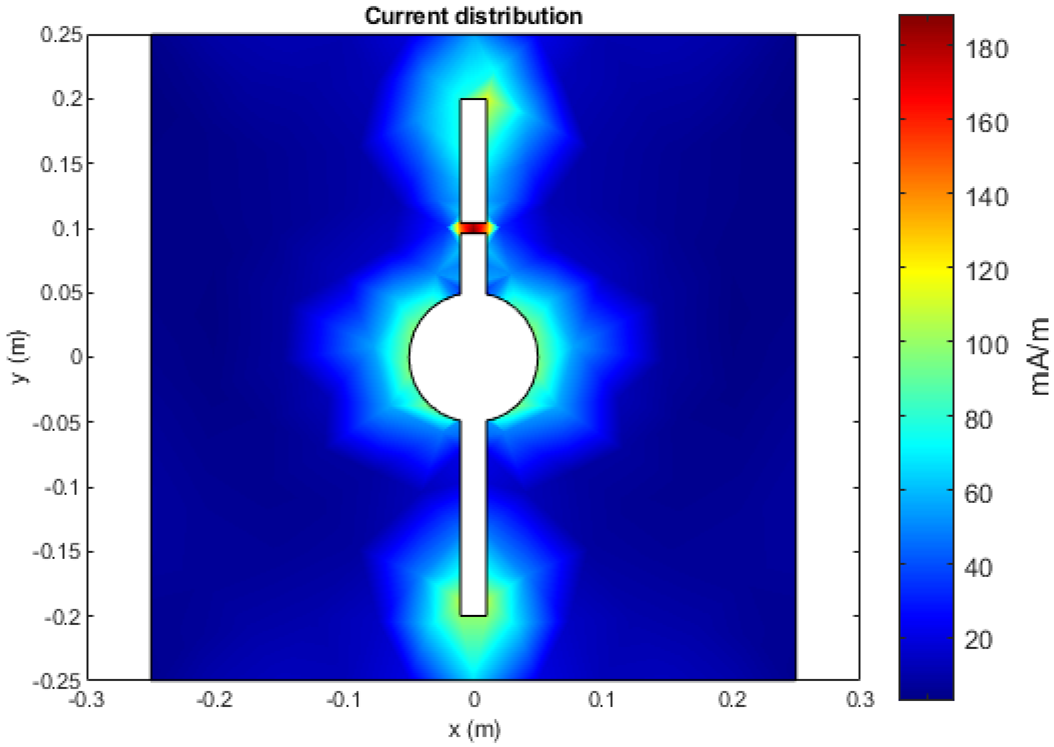
Analyze the impedance of the antenna from 300 MHz to 800 MHz.

```
figure;
impedance(c, linspace(300e6,800e6,51));
```



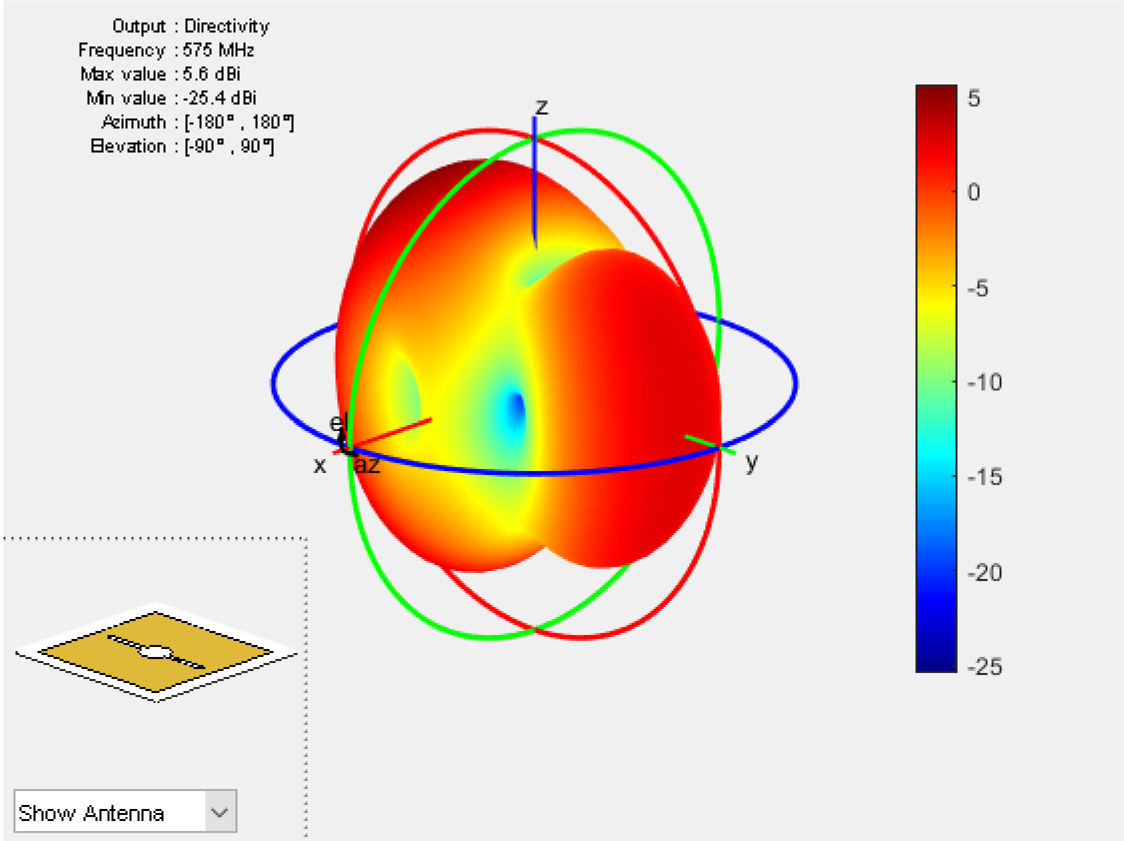
Analyze the current distribution of the antenna at 575 MHz.

```
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 : 5.6 dBi
Min value : -25.4 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

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"

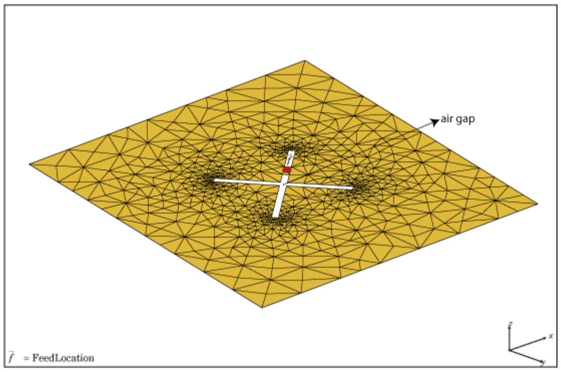
Introduced in R2016b

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 `3x`*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`

`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 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 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 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'`

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

Object Functions

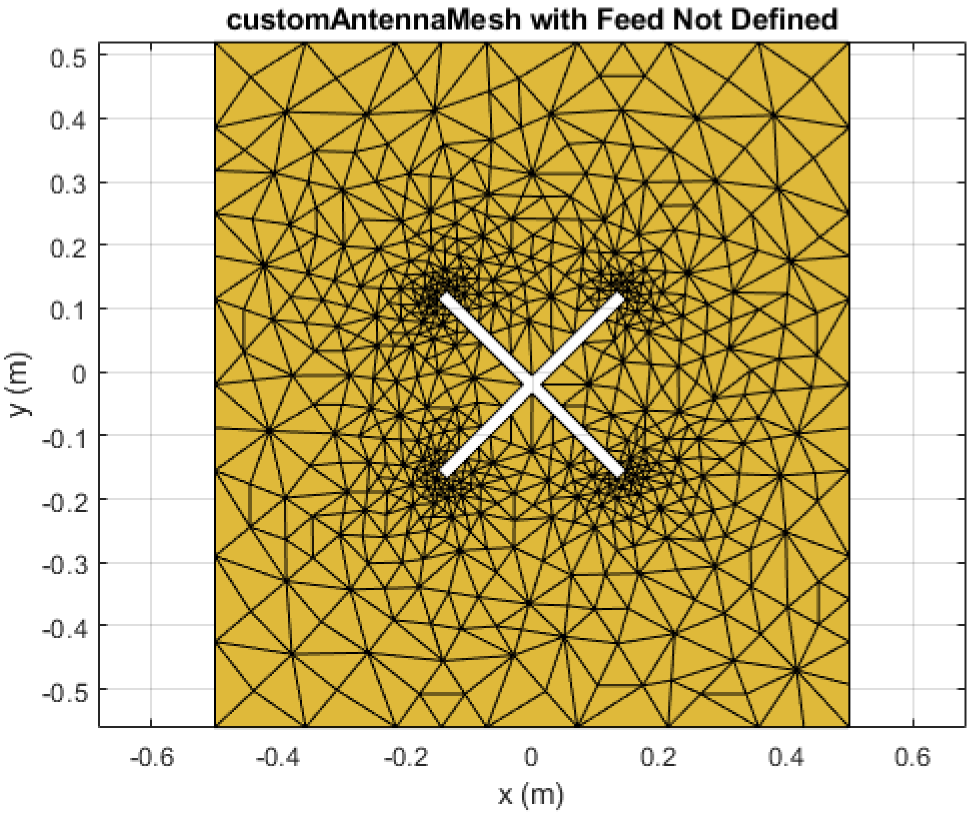
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
createFeed	Create feed location for custom antenna
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric 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 or 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	S-parameter object
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 + 57.2103i
```

References

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

See Also

`cavity` | `reflector`

Topics

"Rotate Antennas and Arrays"

Introduced in R2015b

pcbStack

Single-feed or multifeed PCB antenna

Description

The `pcbStack` object is a single-feed or multi-feed printed circuit board (PCB) antenna. Using the PCB stack, you can create antennas using single-layer or multilayer metal or metal-dielectric substrates. You can also use the PCB stack to create antennas with an arbitrary number of feeds and vias. You can also use Antenna Toolbox catalog antennas to create a PCB antenna.

Note You require a substrate layer to generate a Gerber file. Use the `Substrate` property to create this layer for the PCB antenna.

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)` creates a PCB 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.

`pcbant = pcbStack(ant)` converts any 2-D or 2.5D antenna from the antenna catalog into a PCB antenna for further modeling and analysis.

Properties

Name — Name of PCB antenna
`'MyPCB'` (default) | character vector

Name of PCB antenna, specified a character vector.

Example: `'Name','PCBPatch'`

Data Types: `char`

`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 PC board, specified as a positive scalar.

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: `double`

`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*]

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
`[0 0 0]` (default) | 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 in meters

Electrical shorting pin diameter between metal layers, specified a positive scalar in meters.

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: double

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

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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
show	Display antenna or array structure; display shape as filled patch
vswr	Voltage standing wave ratio of antenna
plot	Plot boundary of shape
layout	Display array or PCB stack layout

Examples

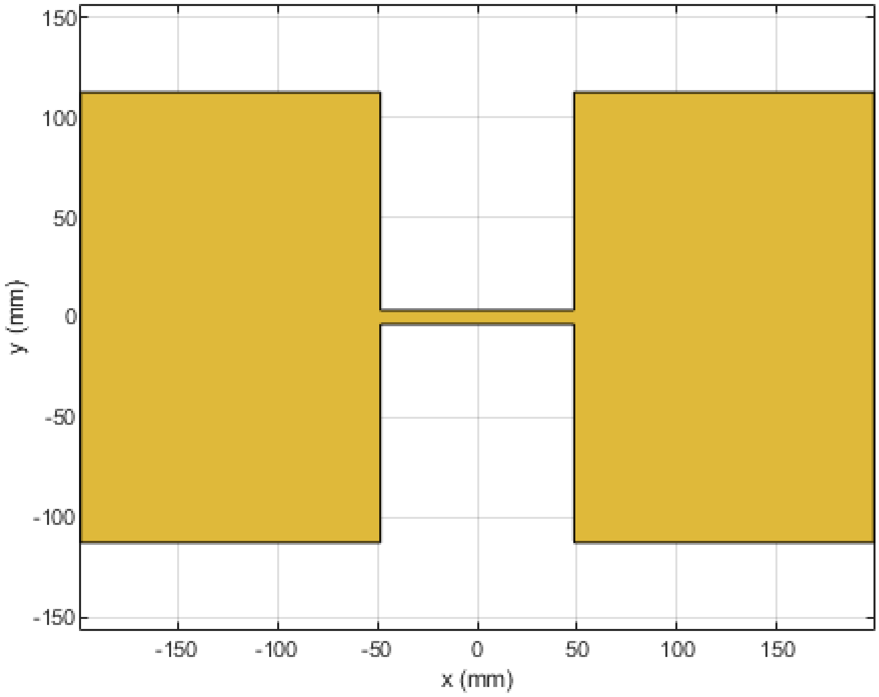
End Loaded Planar Dipole

Setup parameters.

```
vp  = physconst('lightspeed');
f   = 850e6;
lambda = vp./f;
```

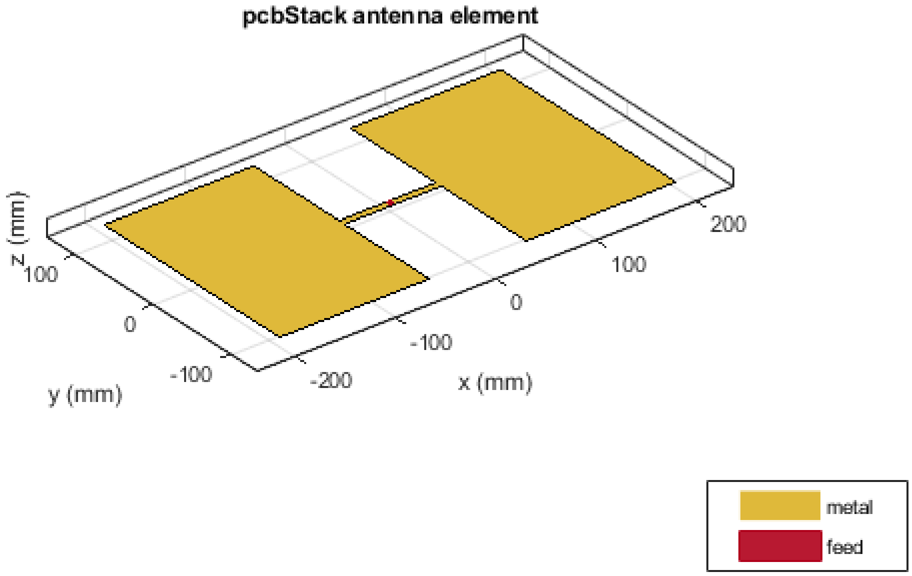
Build a planar dipole with capacitive loading at the ends.

```
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)
```



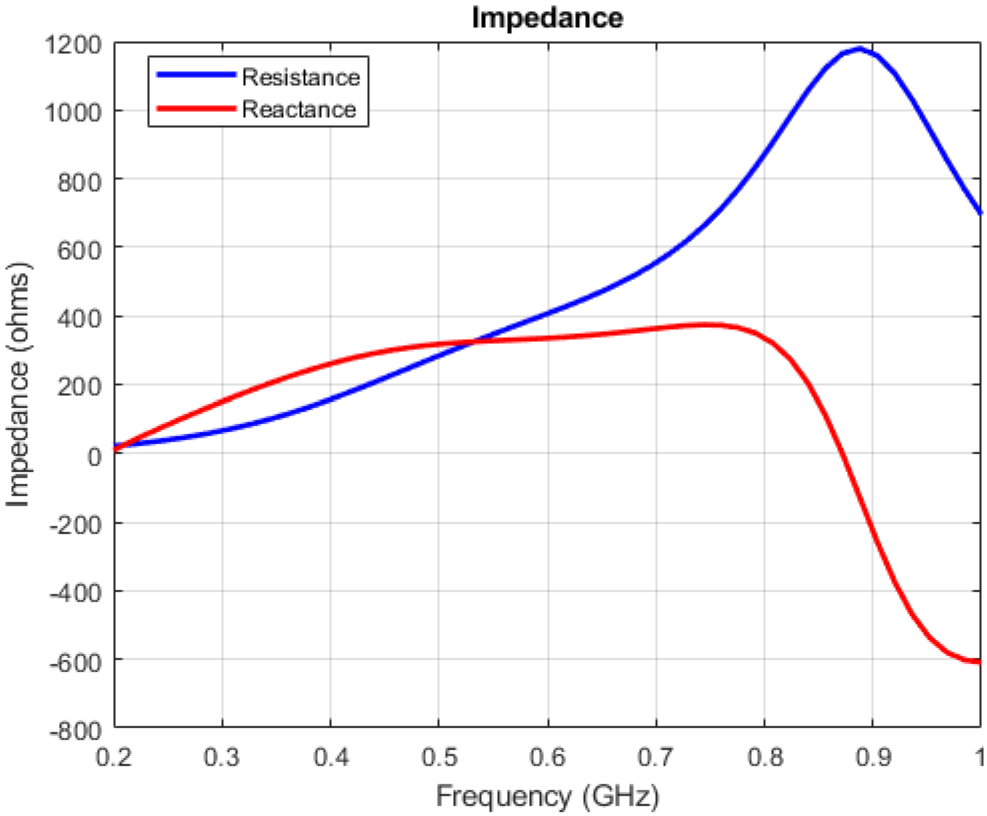
Assign the radiator shape to pcbStack and make the changes to the board shape and feed diameter properties.

```
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.

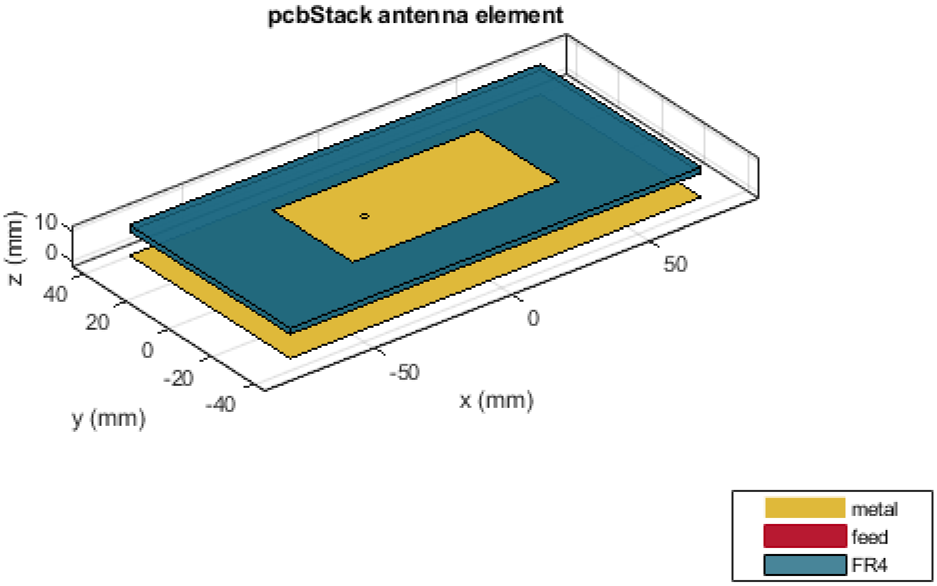
```
figure
impedance(p,linspace(200e6,1e9,51))
```



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.

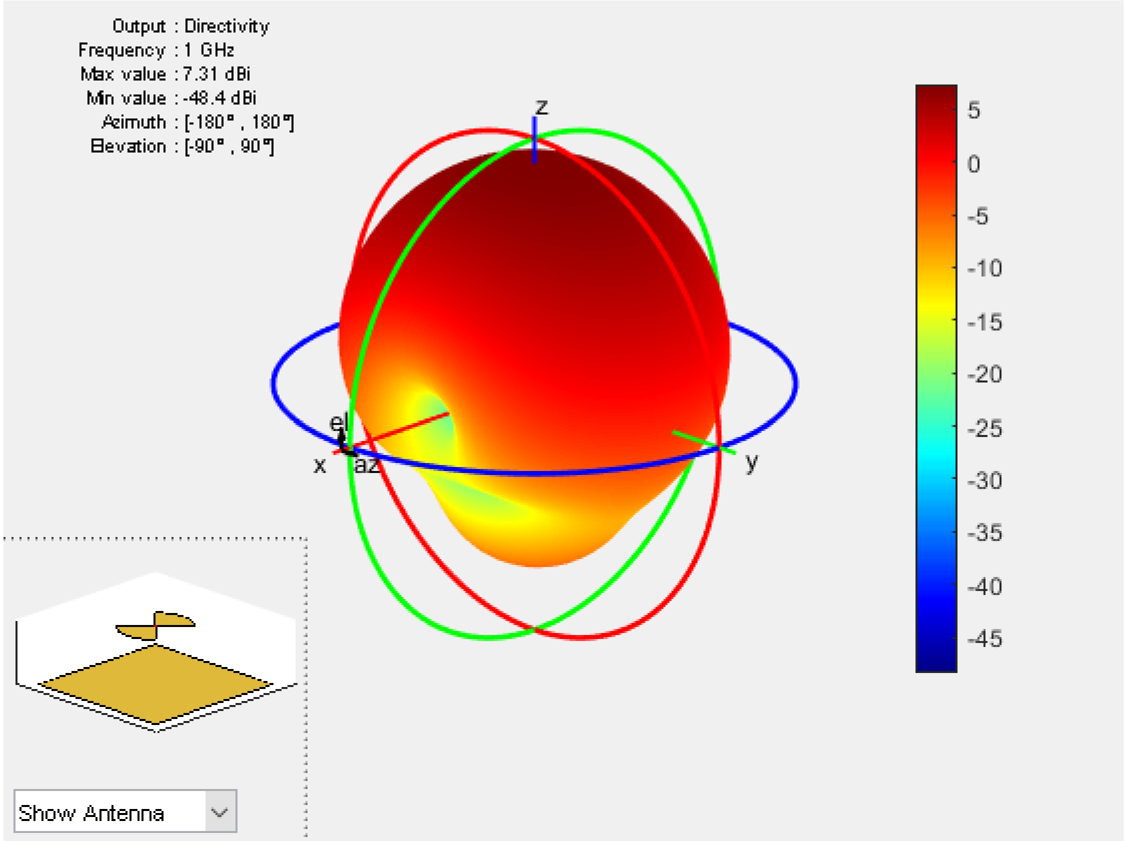
```
b = design(bowtieRounded,1e9);
b.Tilt = 90
```

```
b = 
  bowtieRounded with properties:

        Length: 0.0959
    FlareAngle: 90
          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);
```



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);
```

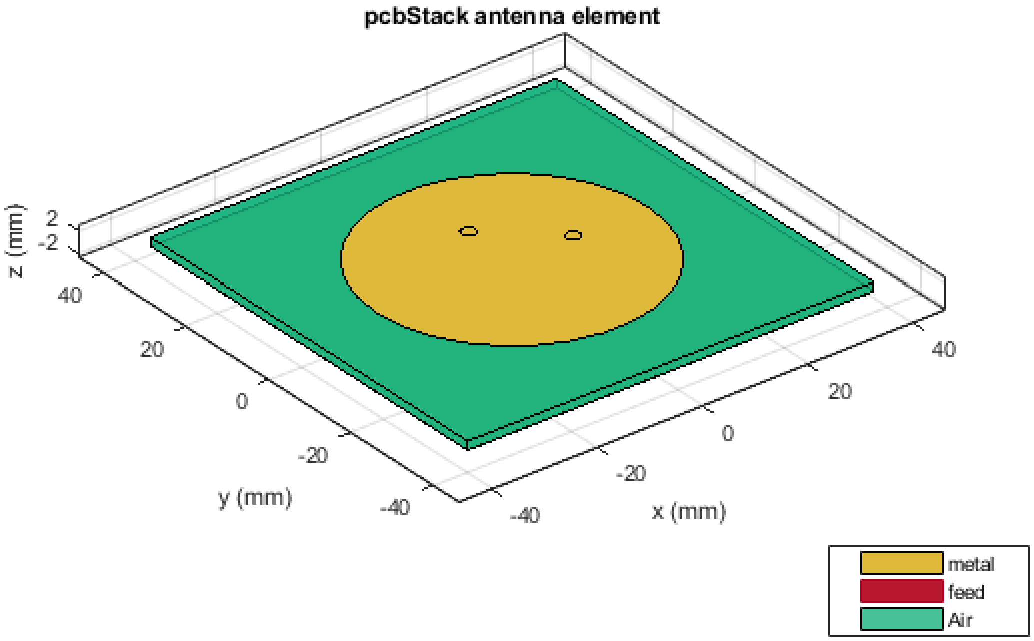
Stack Conversion

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];
```

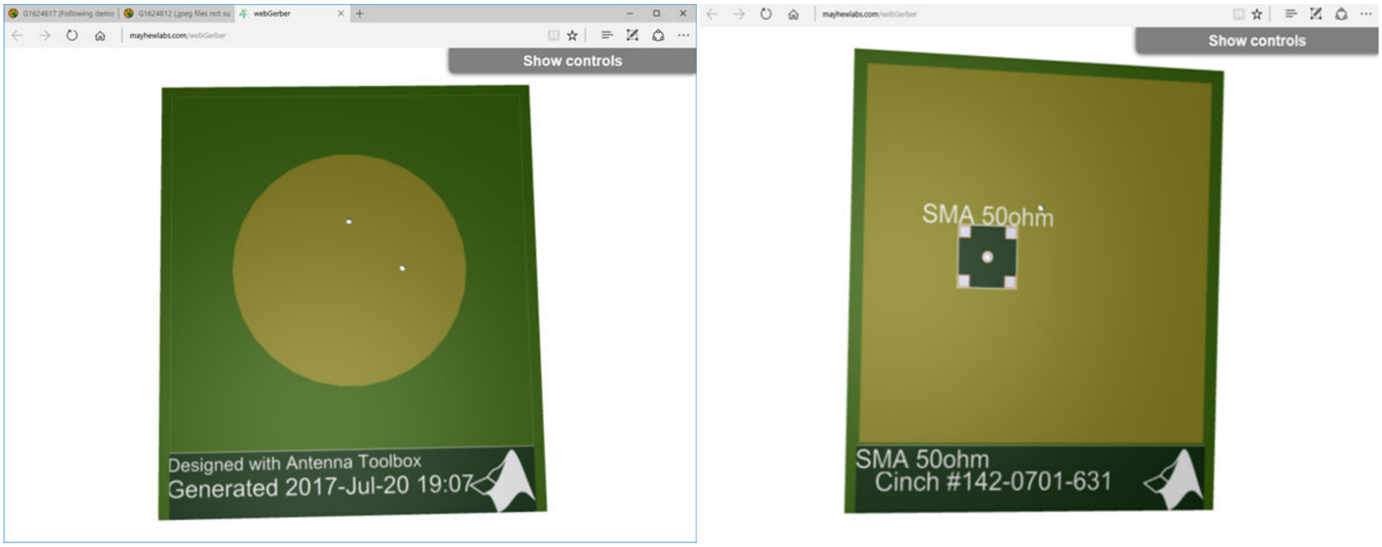
Create a PCB circular microstrip patch using `pcbStack`.

```
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)
```

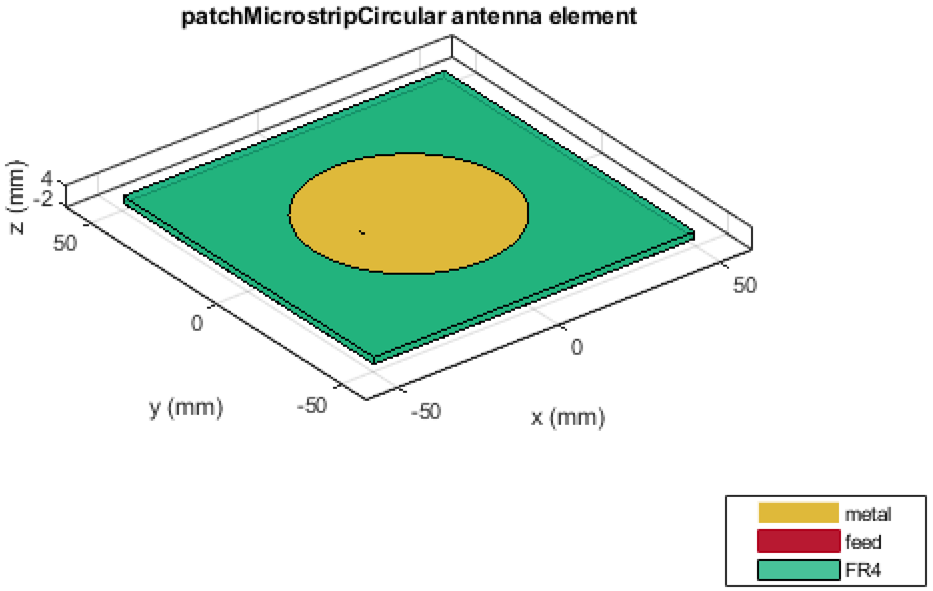
Images using Mayhew Labs 3-D Viewer.



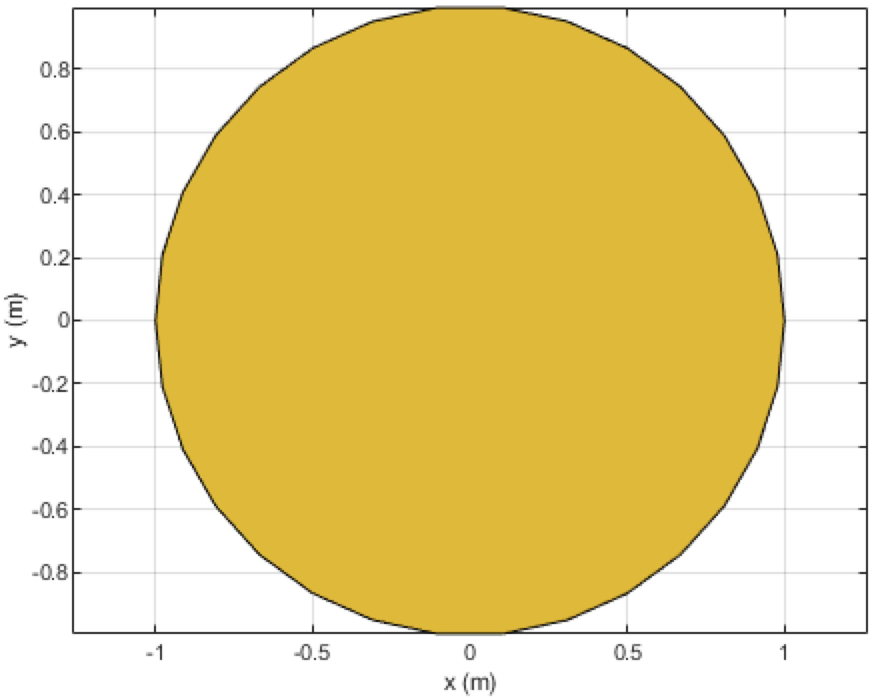
Circular Microstrip Patch Antenna on Polygon Shaped Board

Create a circular microstrip patch antenna.

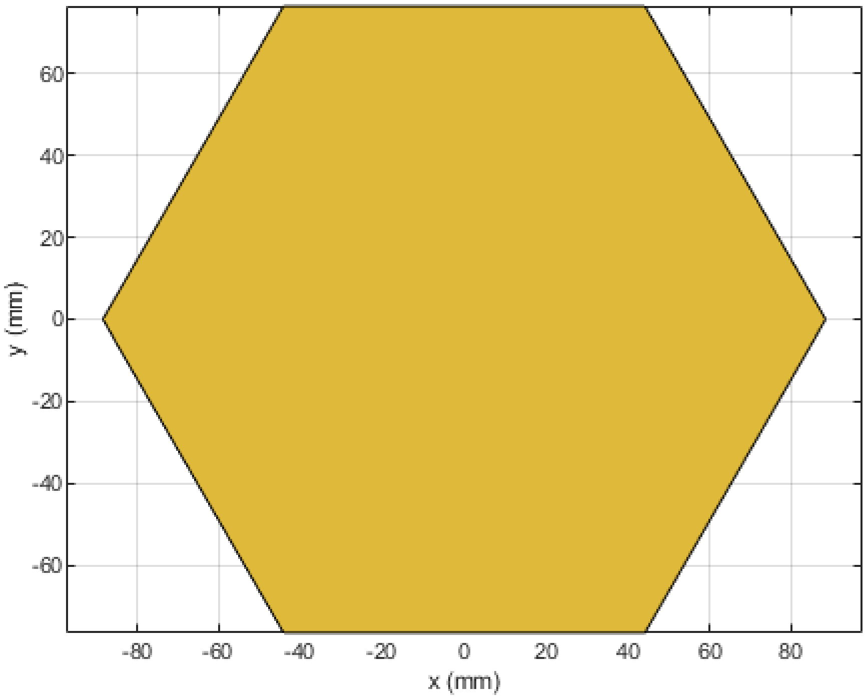
```
ant = design(patchMicrostripCircular,3e9);
ant.Substrate = dielectric( 'FR4' );
show(ant)
```



```
c = antenna.Circle;
show(c)
```

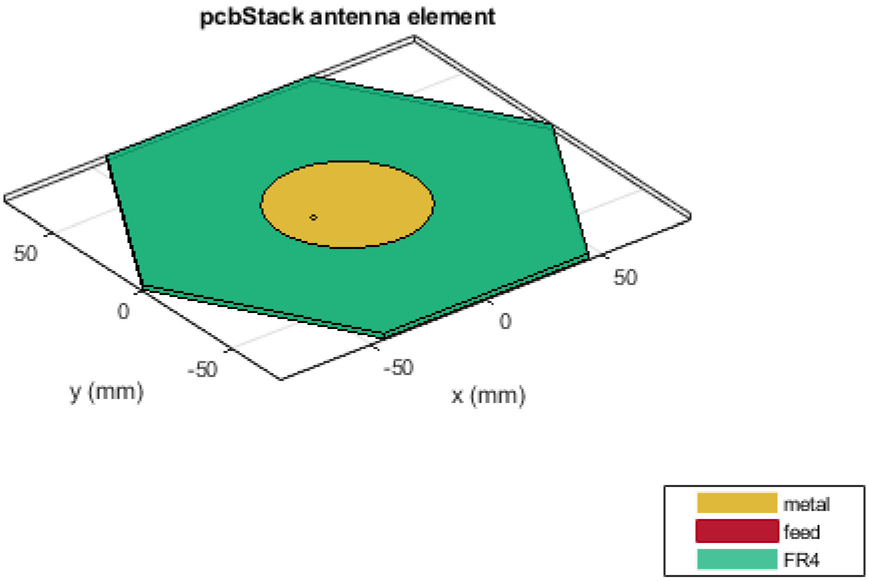


```
c.NumPoints = 6;
c.Radius = 3*ant.Radius;
figure
show(c)
```



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
```



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

`antenna.Circle` | `antenna.Polygon` | `antenna.Rectangle` | `customAntennaMesh` | `customArrayMesh`

Topics

"Design Variations On Microstrip Patch Antenna Using PCB Stack"
"Rotate Antennas and Arrays"

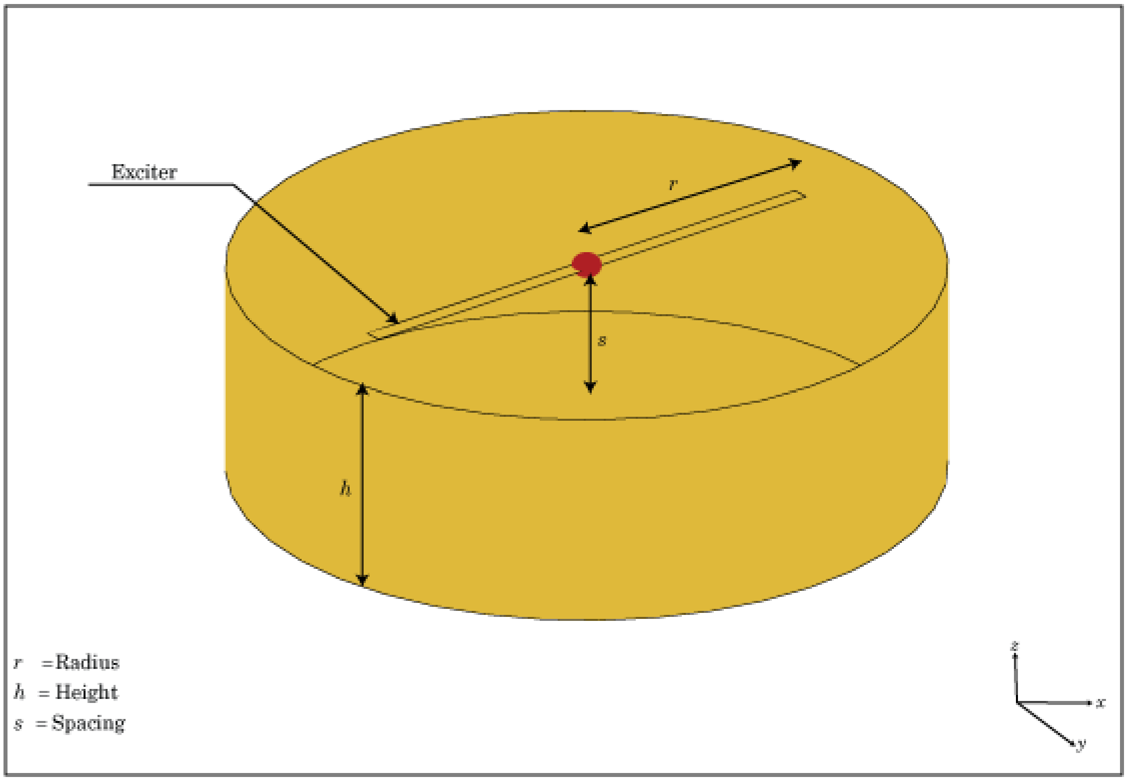
Introduced in R2017a

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 type used as exciter

`dipole` (default) | object

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

Example: `'Exciter',monopole`

Example: `circularcavity.Exciter = monopole`

Data Types: `char` | `string`

`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 groundplane 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`

`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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

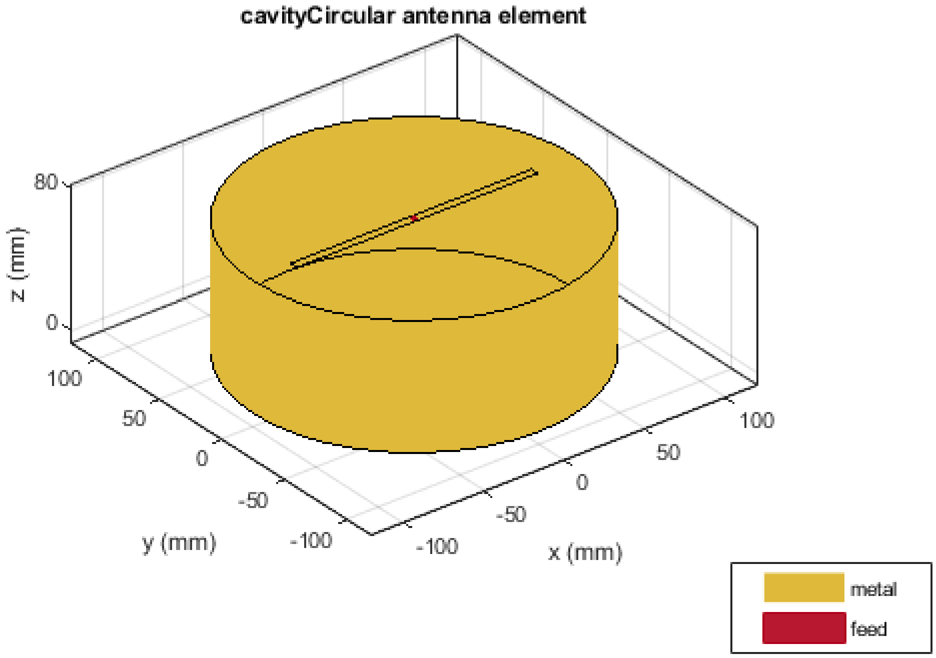
Circular Cavity-Backed Antenna

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

```
show(a)
```



Circular Cavity-Backed Equiangular Spiral

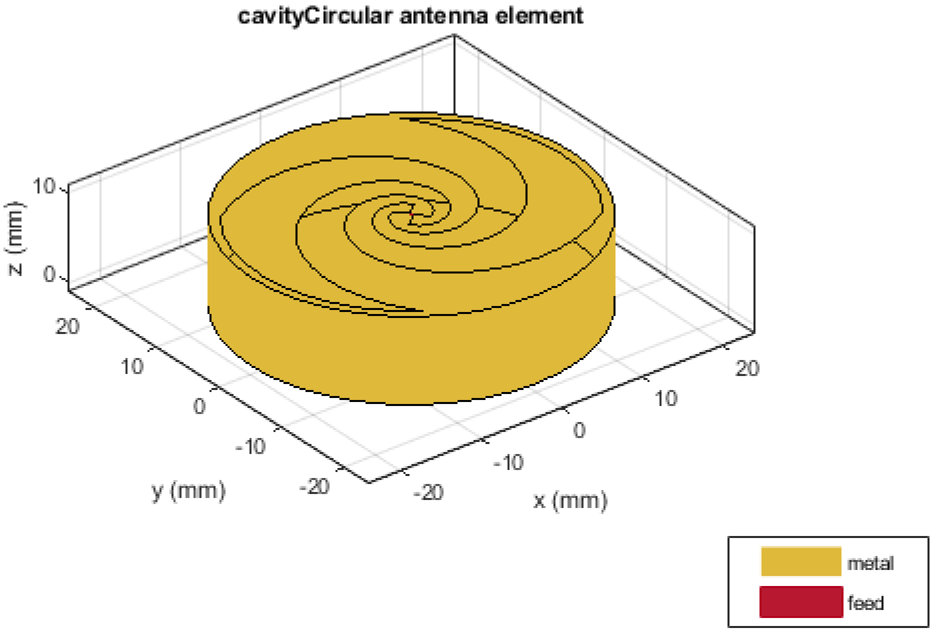
Create and view an equiangular spiral backed by a circular cavity. The cavity dimensions are:

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)
```



See Also

`cavity` | `reflector` | `reflectorCircular`

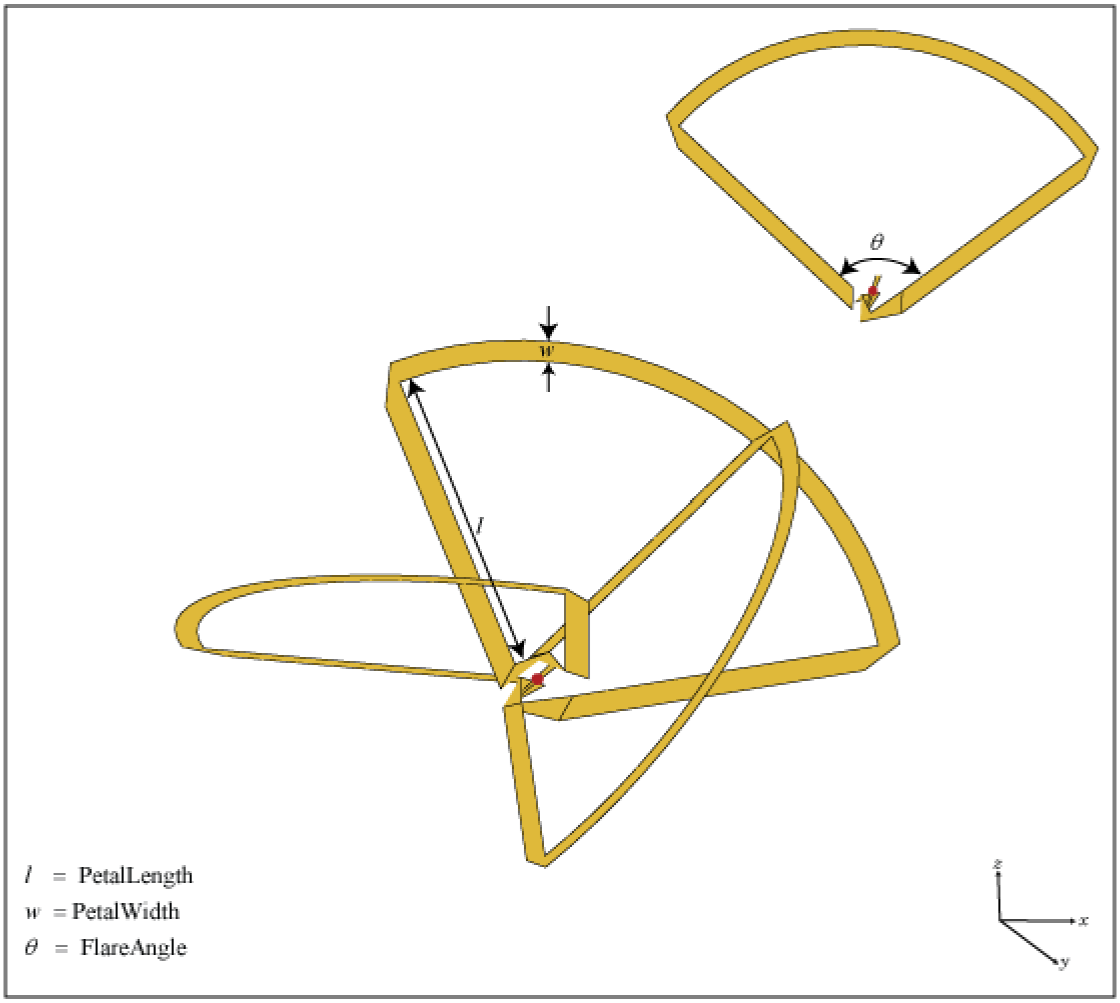
Introduced in R2017b

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`

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, it is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object handle 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array

axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

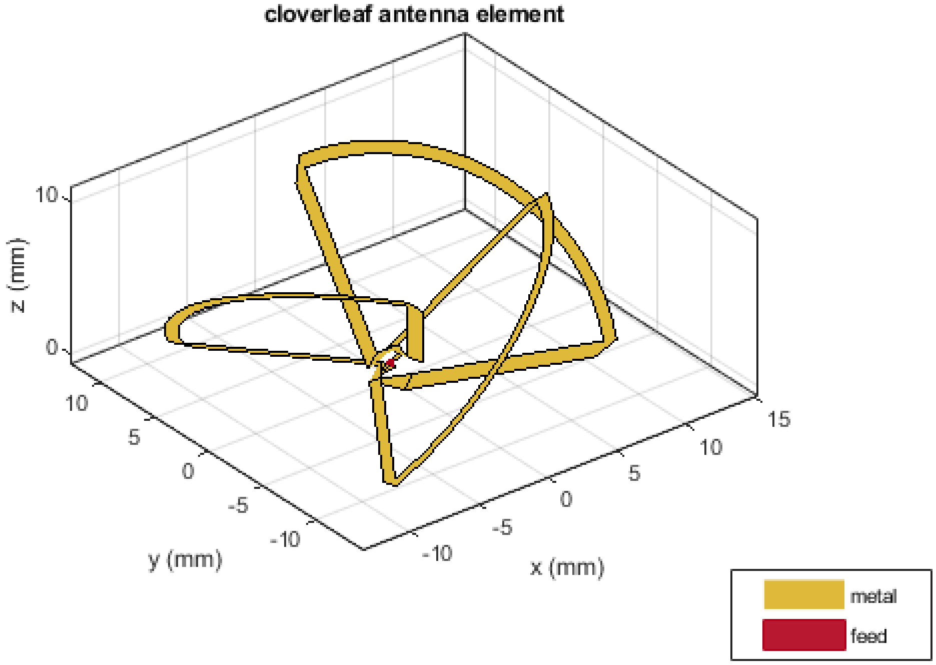
Clover Leaf Antenna

Create and view a default cloverleaf antenna.

```
cl = cloverleaf
```

```
cl = 
  cloverleaf with properties:

      NumPetals: 3
    PetalLength: 0.0515
     PetalWidth: 8.0000e-04
     FlareAngle: 105
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

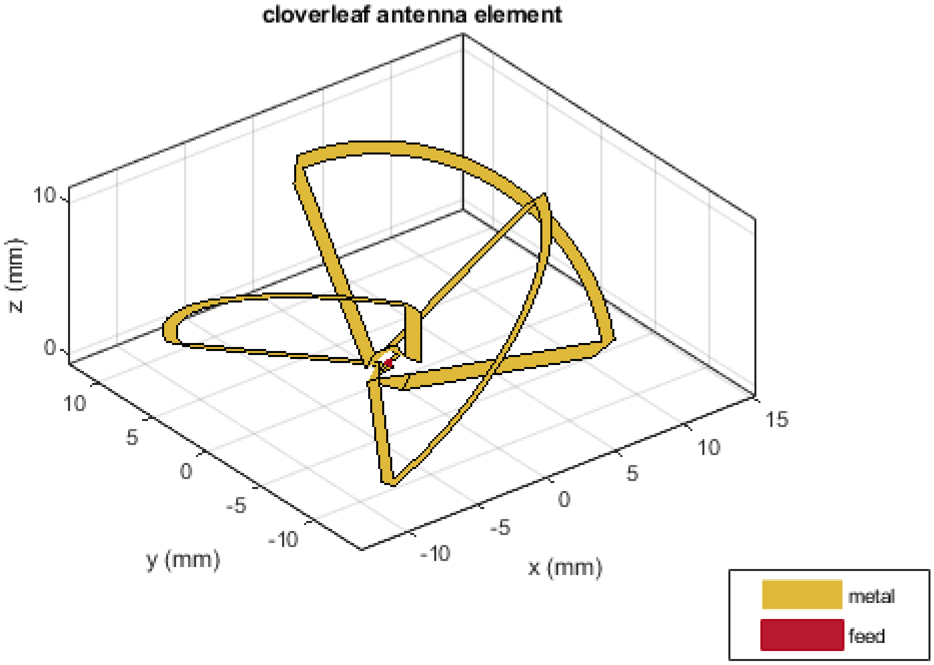
```
show(cl)
```



Axial Ratio of Cloverleaf Antenna

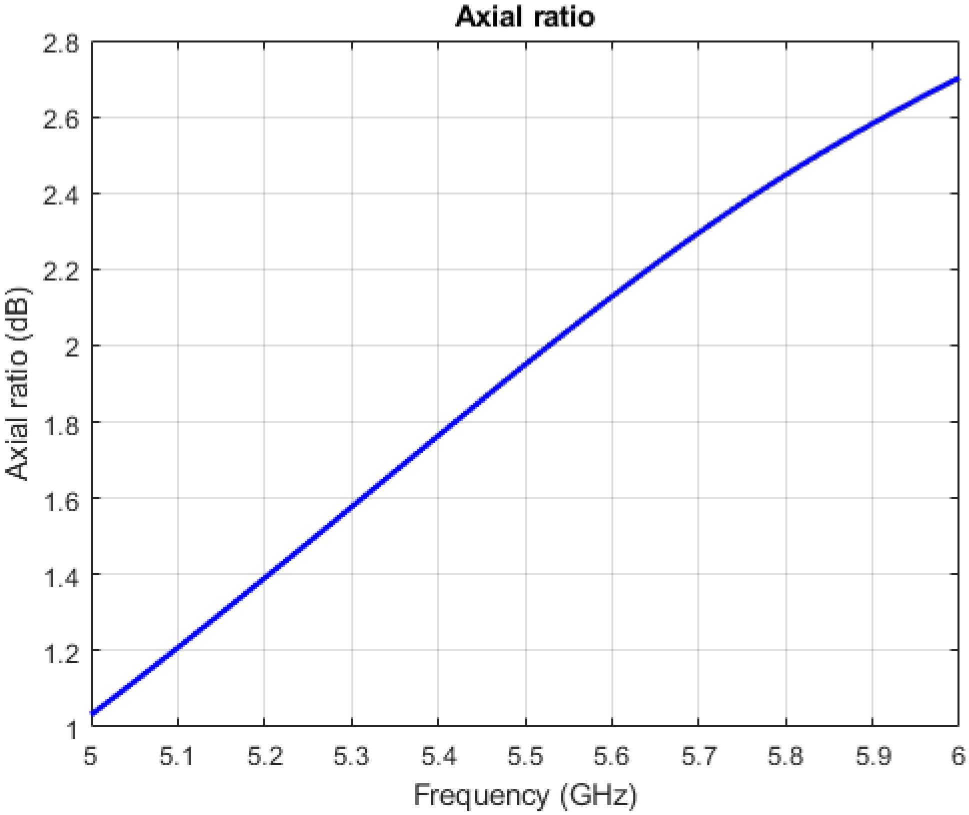
Create a cloverleaf antenna.

```
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);
```



You can see from the axial ratio plot that the antenna supports circular polarization over the entire frequency range.

See Also

`dipole` | `spiralArchimedean`

Introduced in R2017b

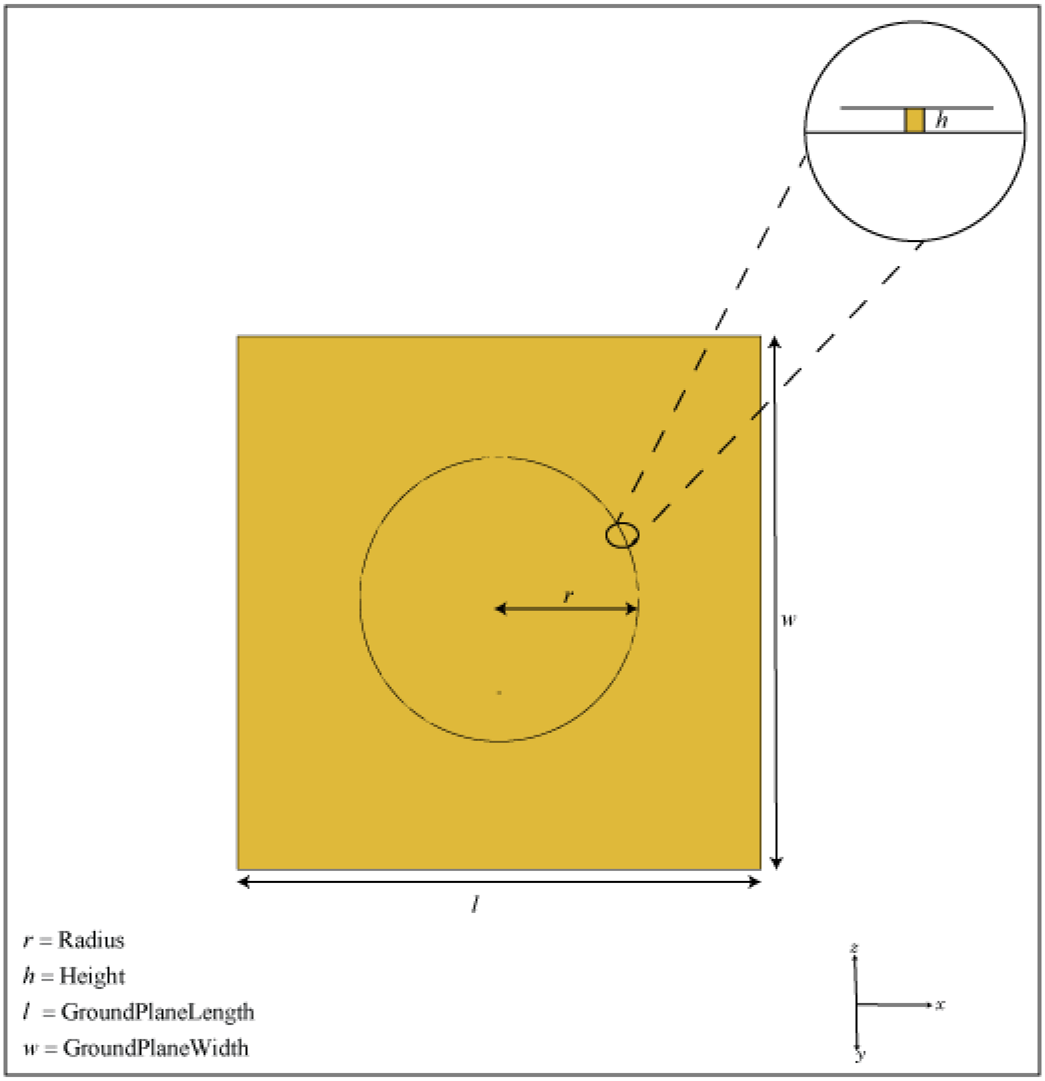
patchMicrostripCircular

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 X-Y 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 handle

Type of dielectric material used as a substrate, specified as a dielectric material object handle. 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`

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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

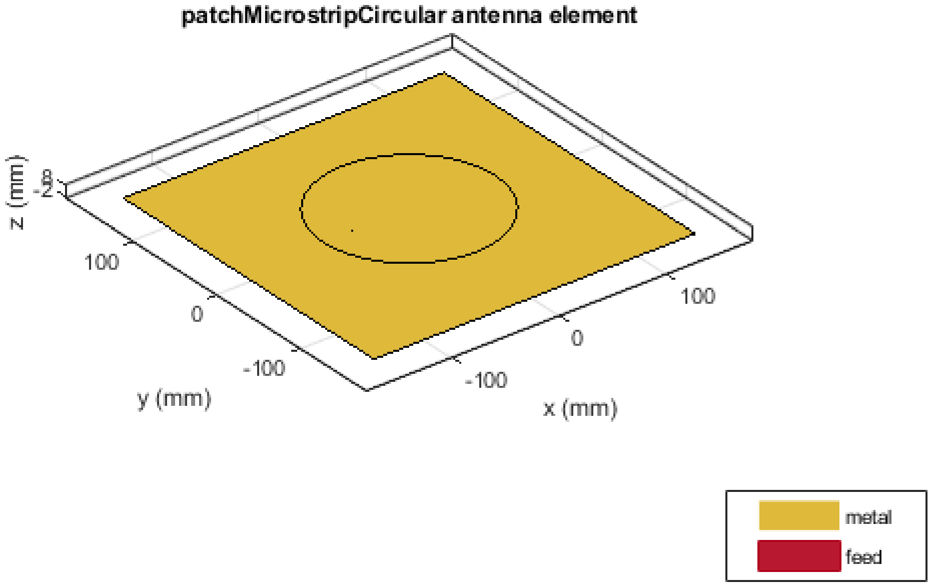
Circular Microstrip Patch

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

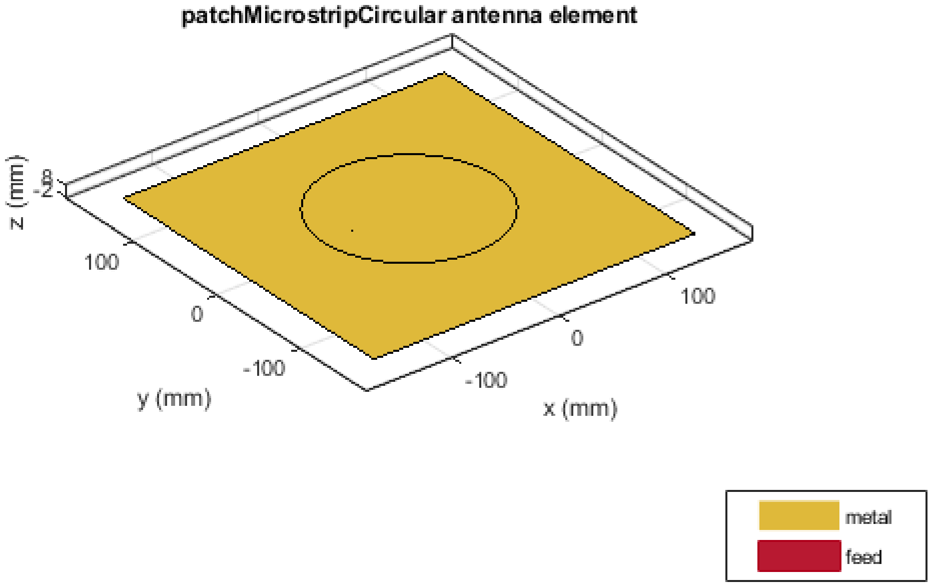
```
show(cp)
```



Radiation Pattern and Impedance of Circular Microstrip Patch

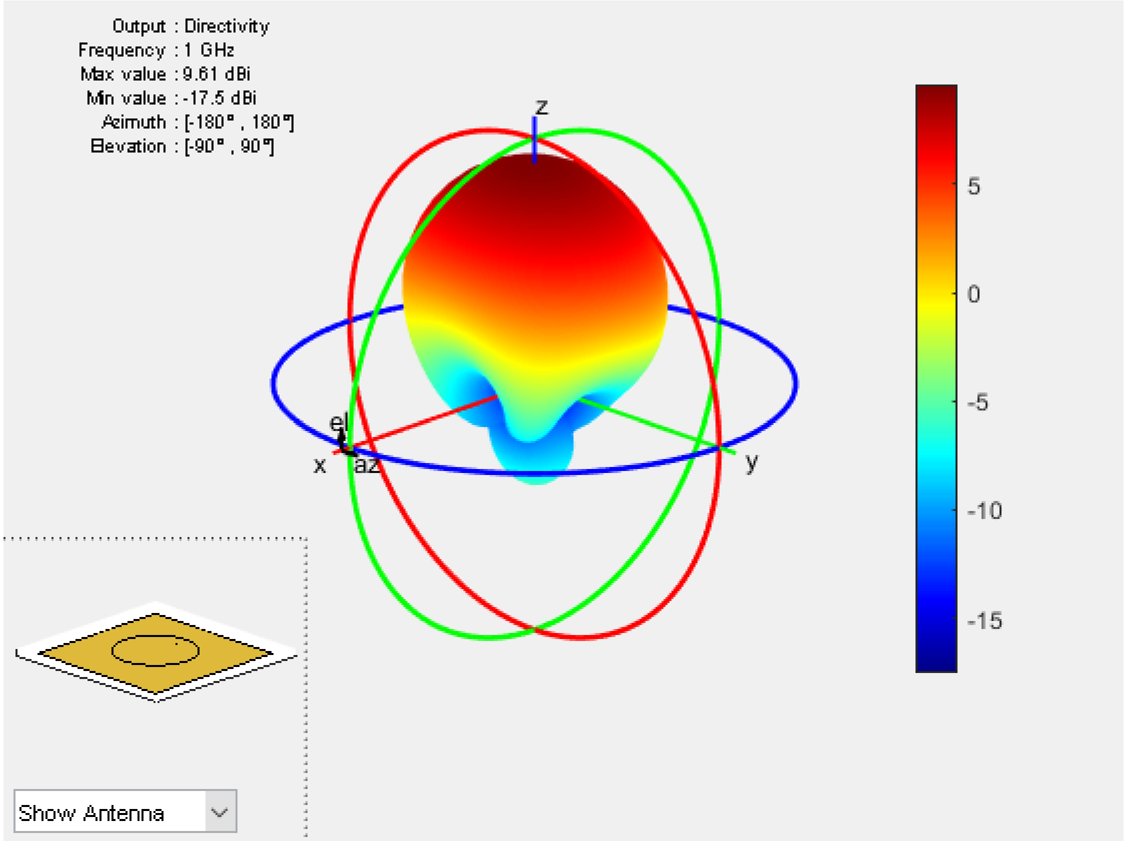
Create a circular patch antenna using given values. Display the antenna.

```
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.

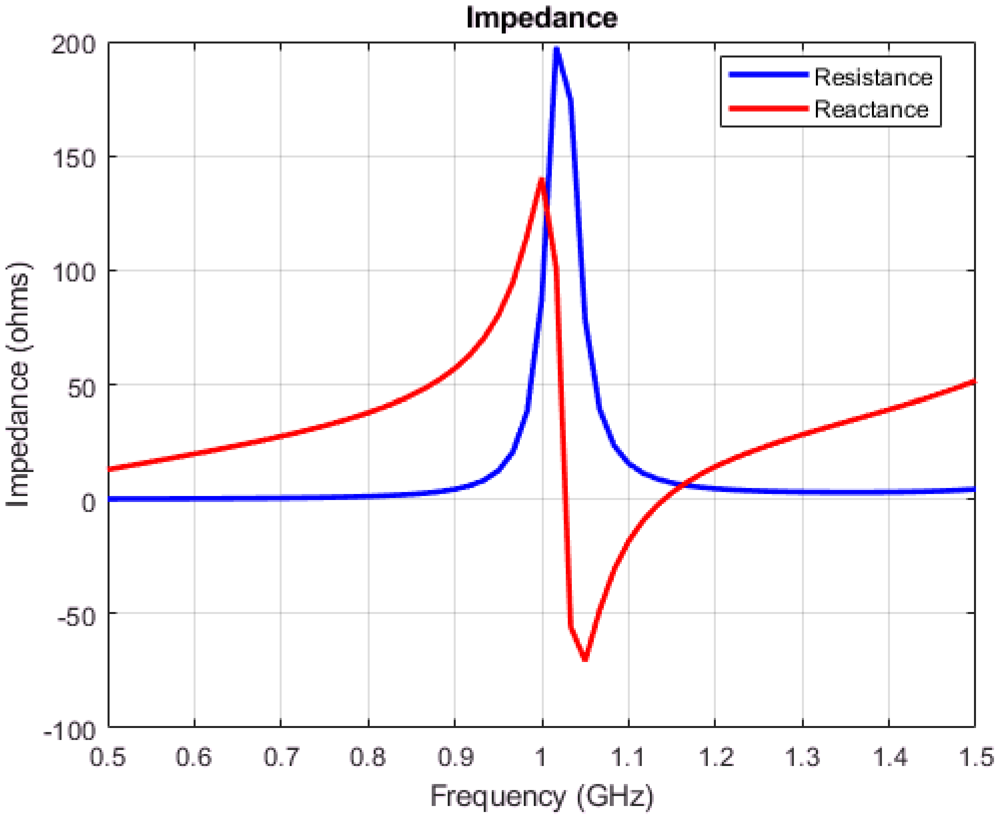
```
pattern(cp,1e9);
```



Output : Directivity
Frequency : 1 GHz
Max value : 9.61 dBi
Min value : -17.5 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);
```



See Also

`patchMicrostrip` | `patchMicrostripInsetfed`

Topics

"ISM Band Patch Microstrip Antennas and Mutual Coupling Between different patches"

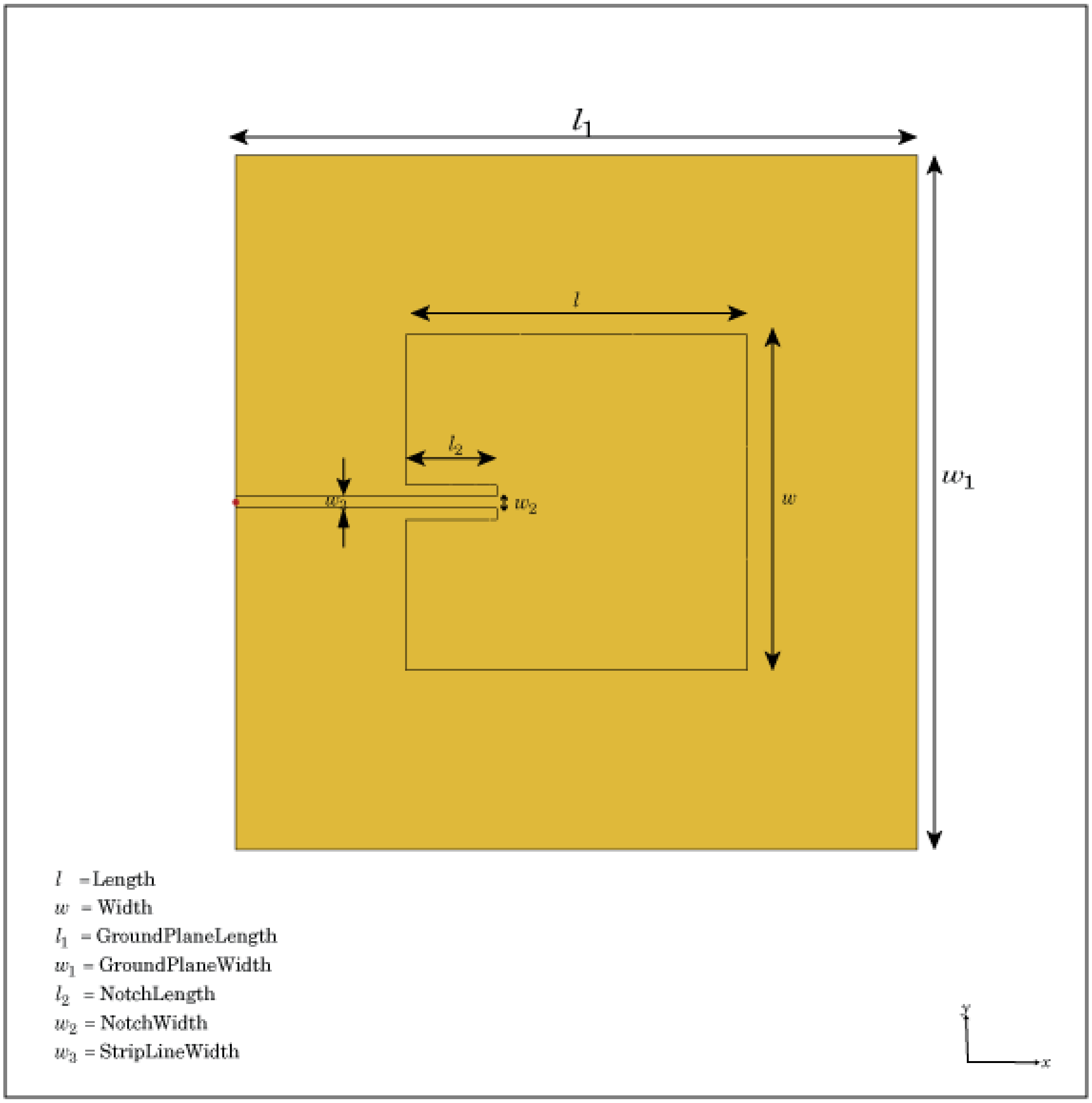
Introduced in R2017b

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 handle

Type of dielectric material used as a substrate, specified as a dielectric material object handle. 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'); 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`

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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

Inset-Fed Microstrip Patch

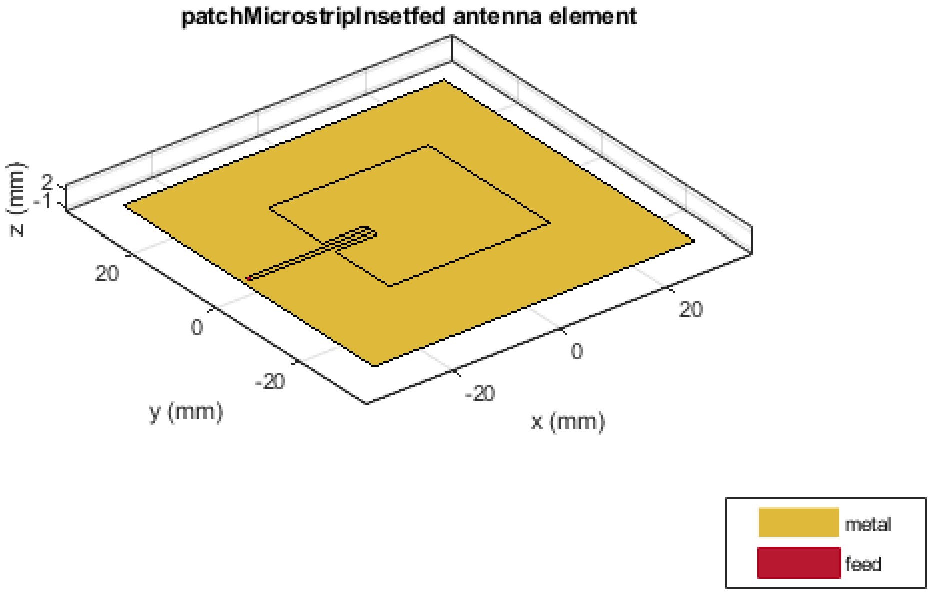
Create and view a default inset-fed microstrip patch.

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

```
show(insetpatch)
```



See Also

patchMicrostrip | patchMicrostripCircular

Topics

"Analysis of an Inset-Feed Patch Antenna on a Dielectric Substrate"

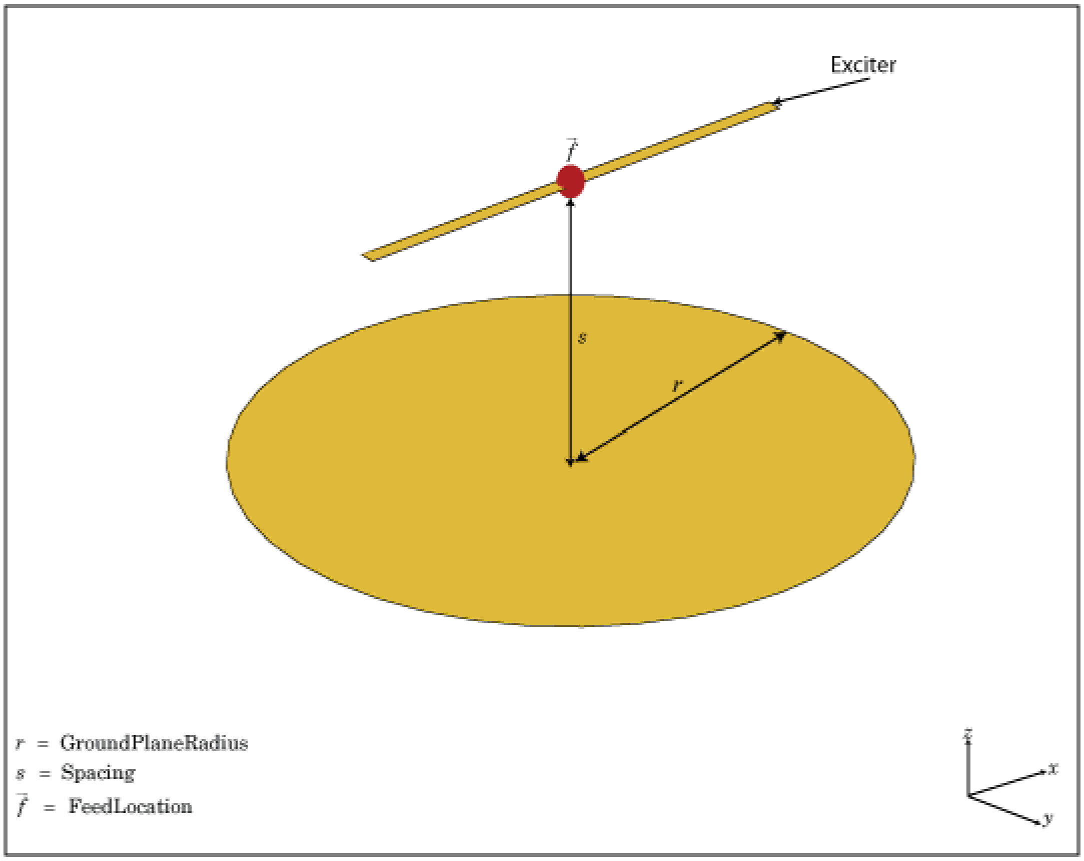
Introduced in R2017b

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 type used as exciter

`dipole` (default) | object

Antenna type used as an exciter, specified as an object. Except for reflector and cavity antenna elements, you can use all the single elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',spiralEquiangular`

Example: `rc.Exciter = spiralEquiangular`

`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`

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 origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object handle 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

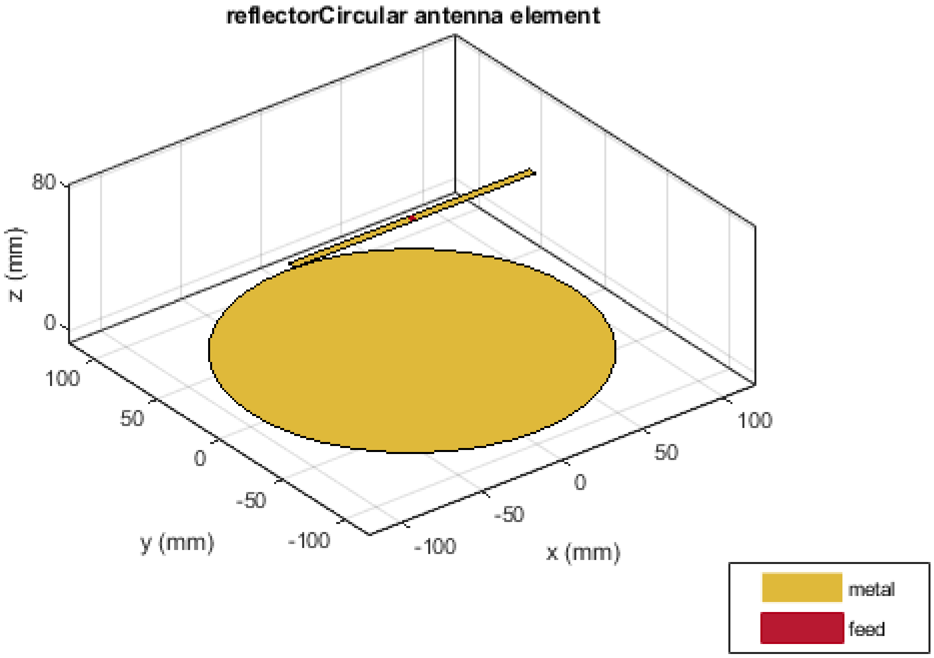
Circular Reflector Backed Antenna

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

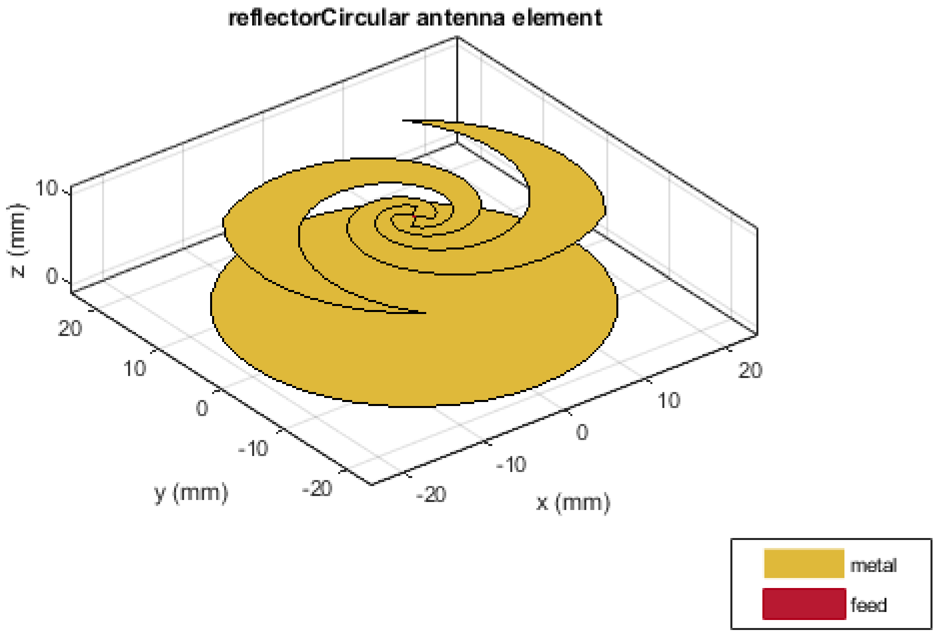
```
show(rc)
```



Radiation Pattern of Circular Reflector Backed Antenna

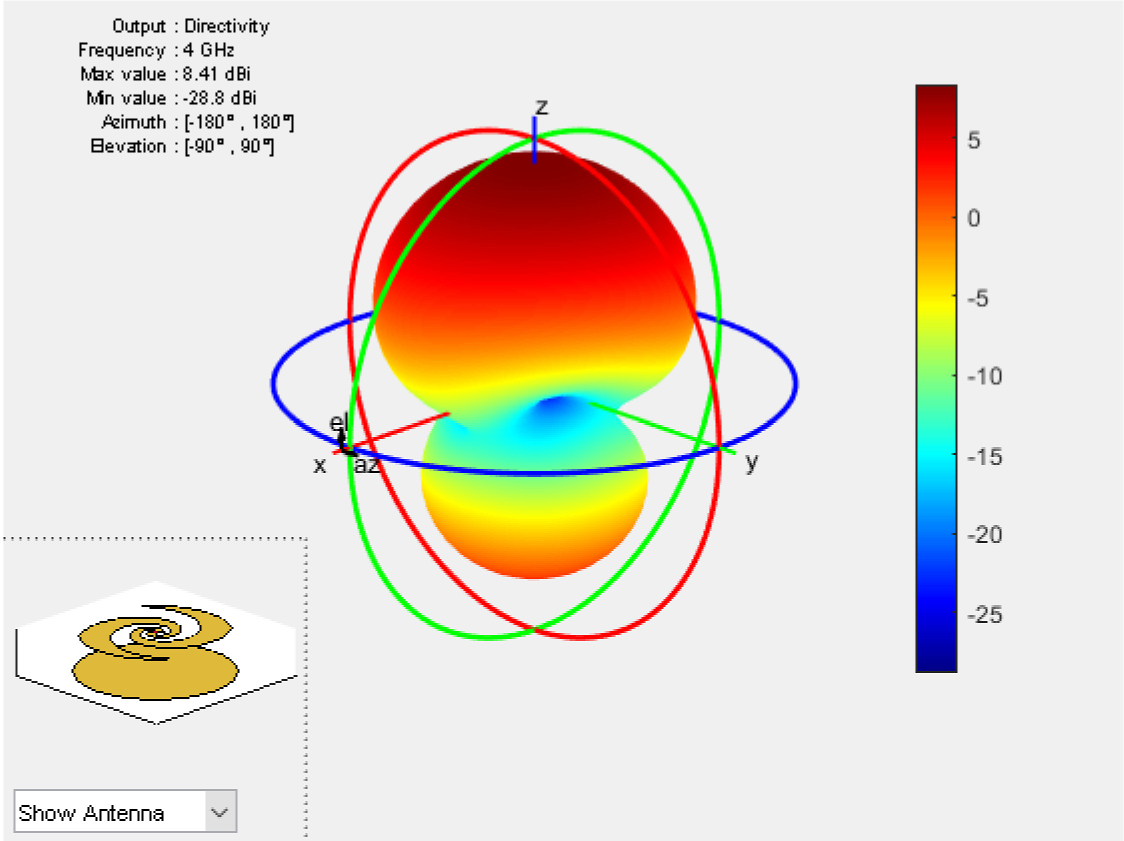
Create an equiangular spiral backed by a circular reflector.

```
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 : 8.41 dBi
Min value : -28.8 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

See Also

`cavity` | `cavityCircular` | `reflector`

Introduced in R2017b

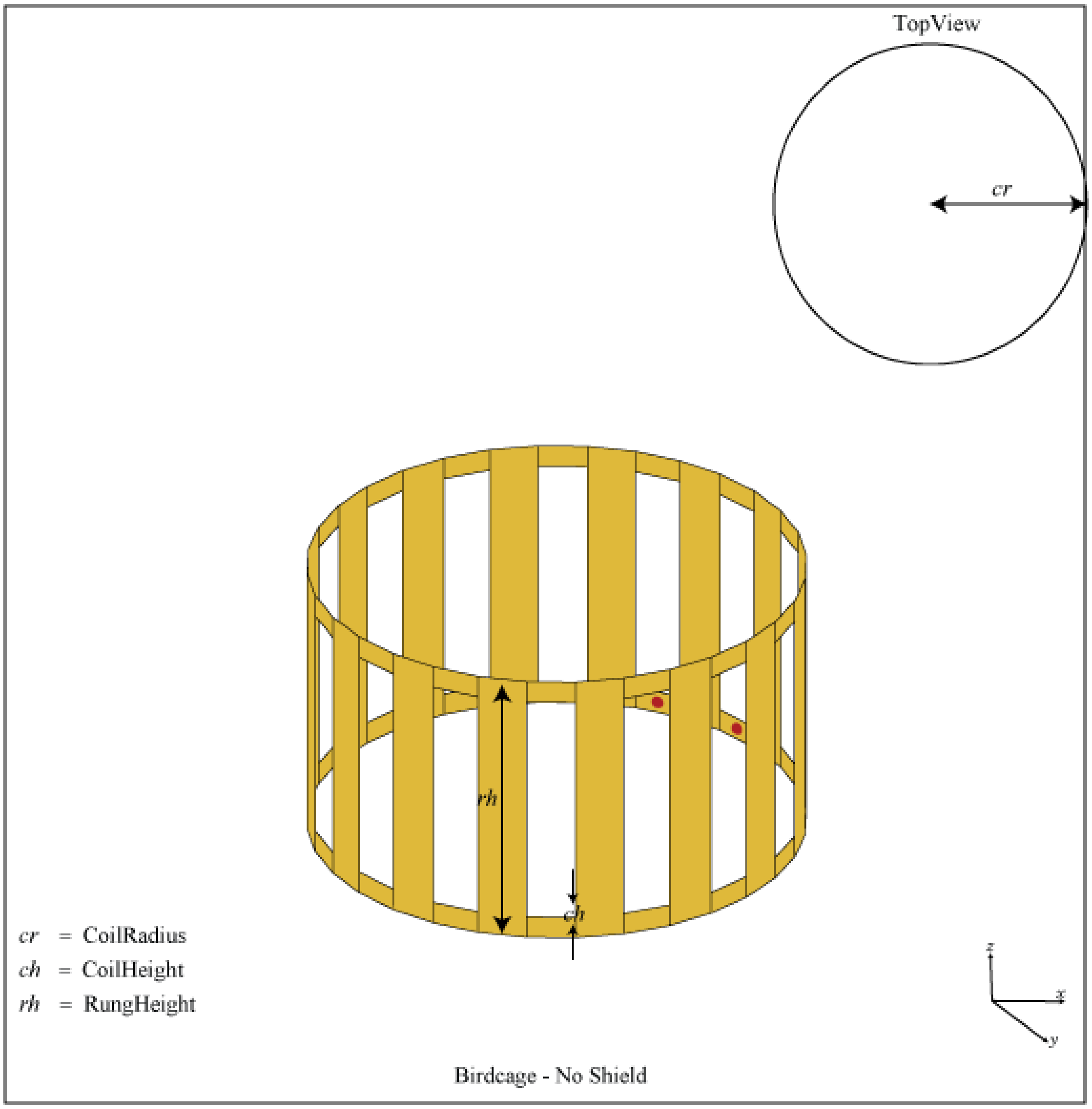
birdcage

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`

`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, it is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object handle 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: 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 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 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 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'

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

Object Functions

getLowPassLocs	Feeding location to operate birdcage as lowpass coil
getHighPassLocs	Feeding location to operate birdcage as highpass coil
show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

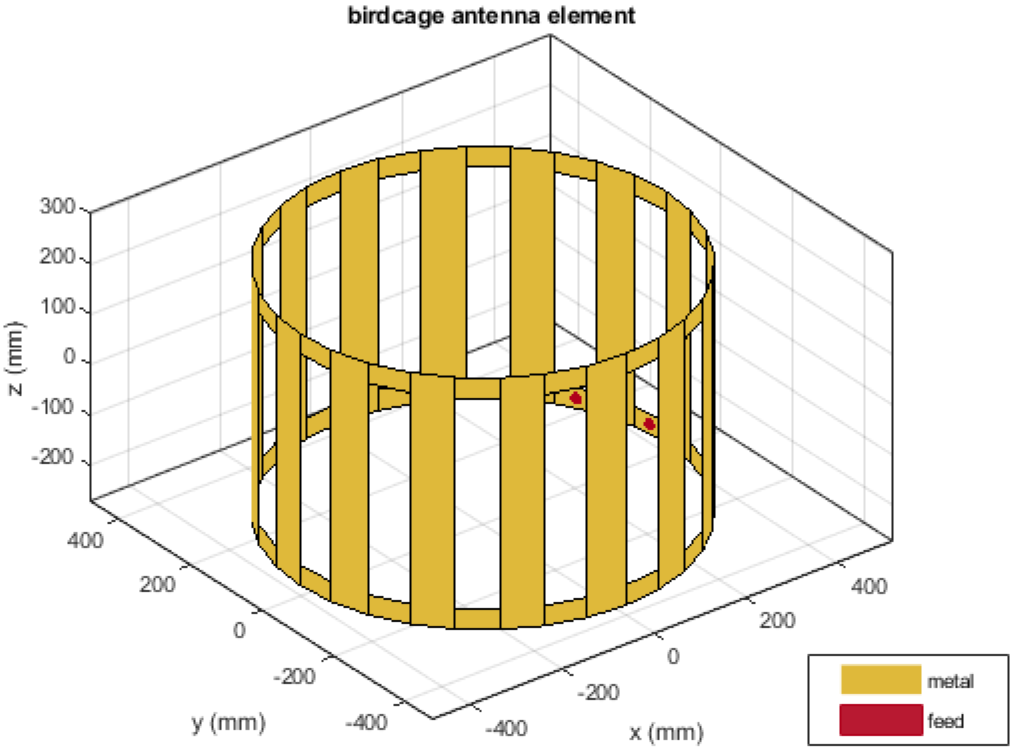
Birdcage Antenna

Create and view a default birdcage antenna.

```
bc = birdcage
```

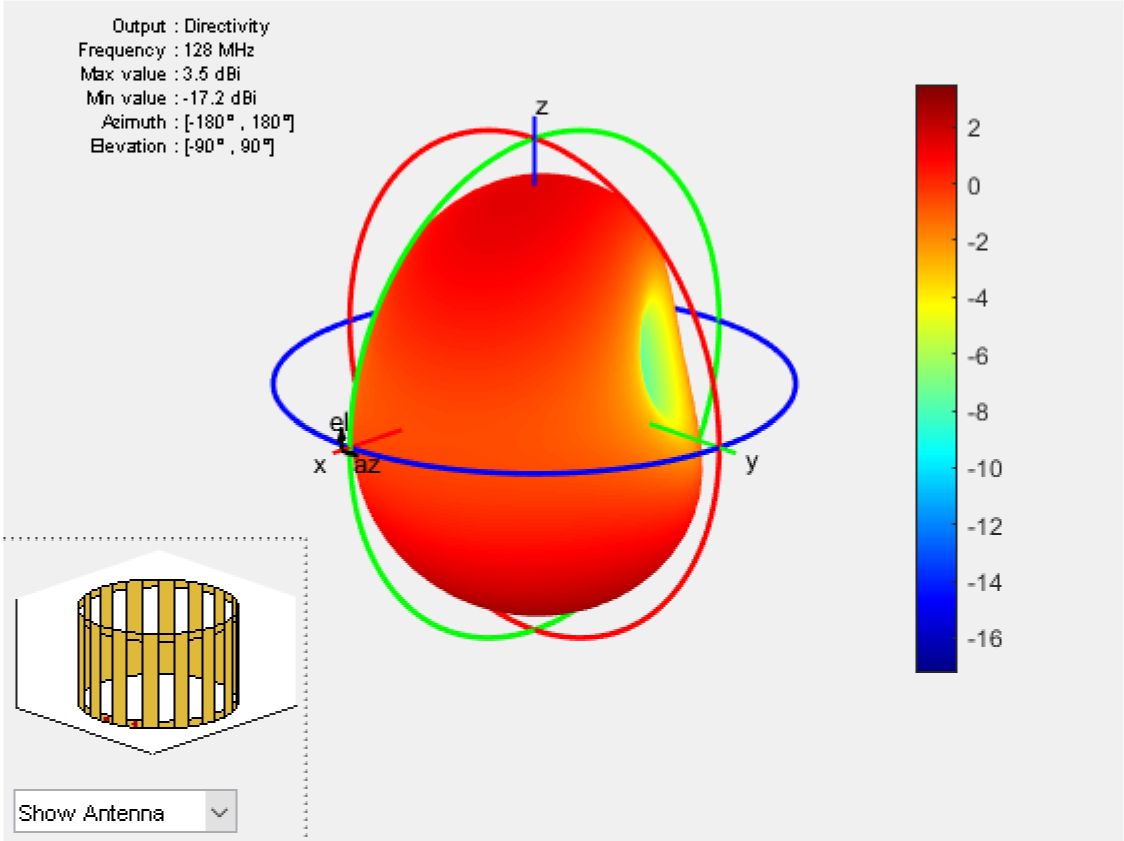
```
bc = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: []
    FeedLocations: [2x3 double]
      FeedVoltage: 1
        FeedPhase: 0
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

```
show(bc);
```



Plot the radiation pattern at 128 MHz.

```
pattern(bc,128e6)
```



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)
```

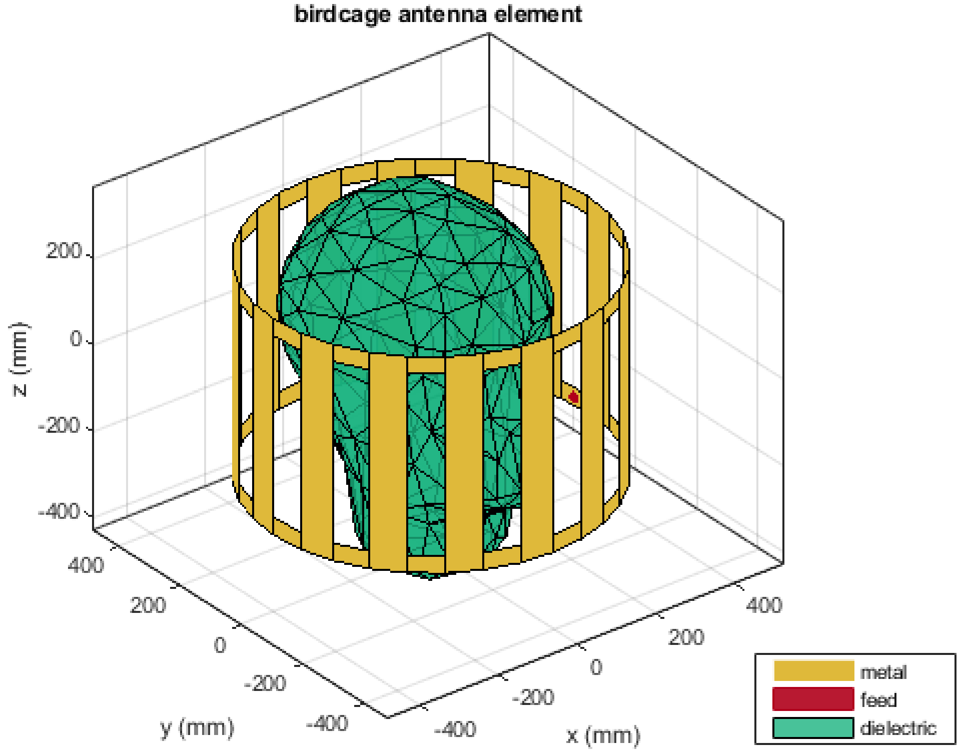
```
humanhead = struct with fields:
         Points: [584x3 double]
     Tetrahedra: [2818x4 double]
       EpsilonR: 10
    LossTangent: 0.0020
```

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

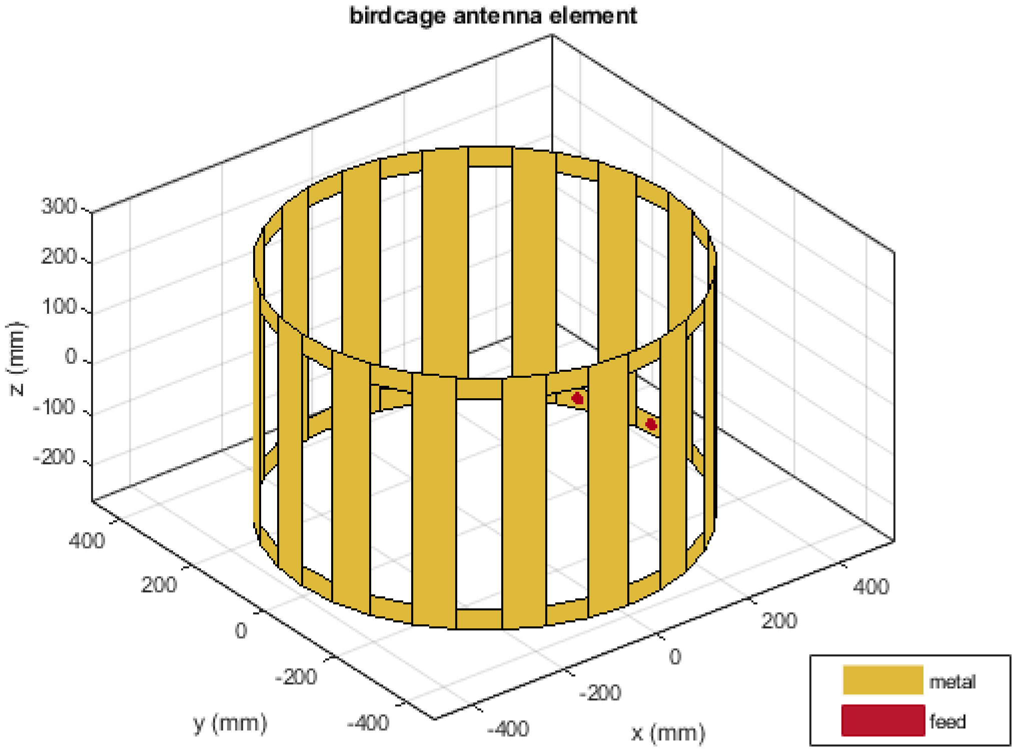
```
show(b)
```



Birdcage In High-Pass Operation

Create a birdcage antenna.

```
b = birdcage;
show(b);
```

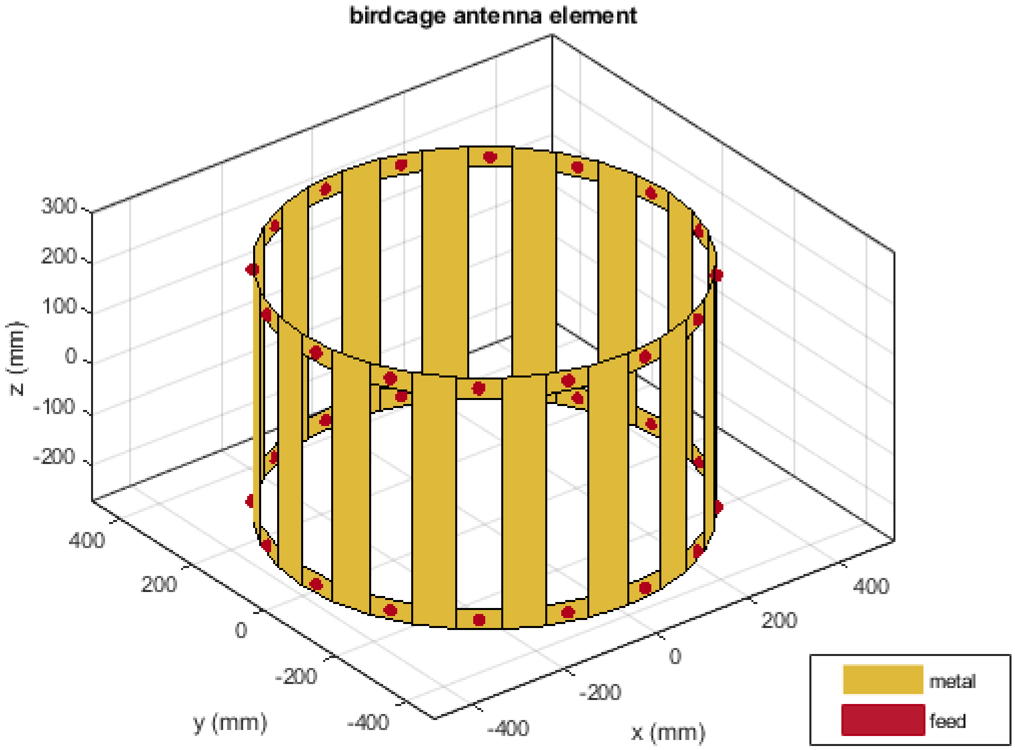


Use the birdcage as a high-pass coil.

```
b.FeedLocations = getHighPassLocs(b)
```

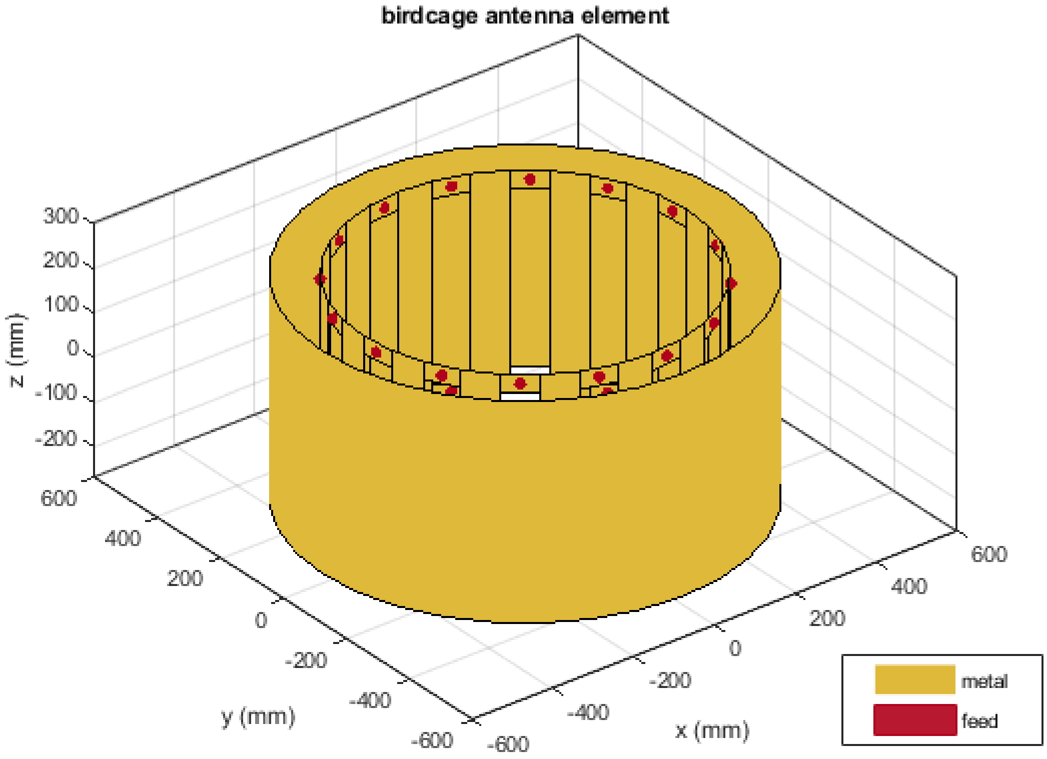
```
b = 
  birdcage with properties:

         NumRungs: 16
       CoilRadius: 0.4000
       CoilHeight: 0.0400
       RungHeight: 0.4600
     ShieldRadius: 0
     ShieldHeight: 0
          Phantom: []
    FeedLocations: [32x3 double]
      FeedVoltage: 1
        FeedPhase: 0
             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) ;
```



See Also

`dipole` | `loopCircular`

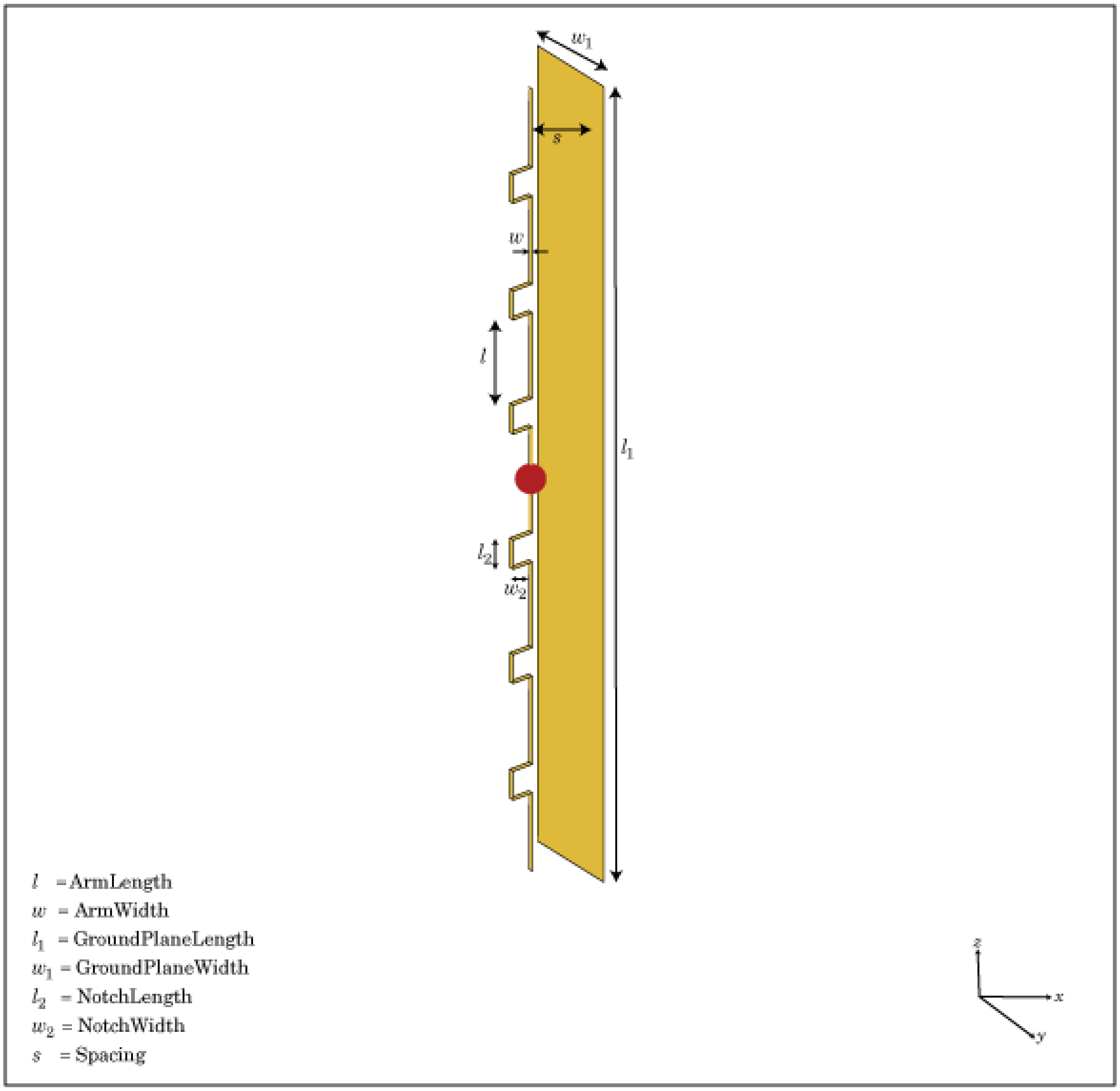
Introduced in R2017b

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`

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, it is at the origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`. `lumpedelement` is the object handle 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

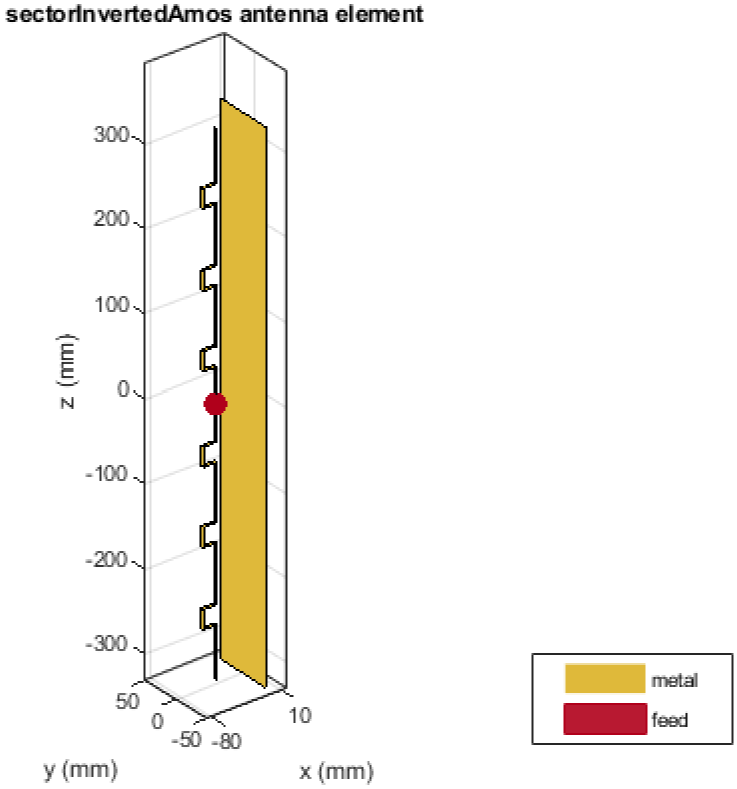
Inverted Amos Sector

Create and view an inverted Amos sector antenna.

```
sectoria = sectorInvertedAmos
```

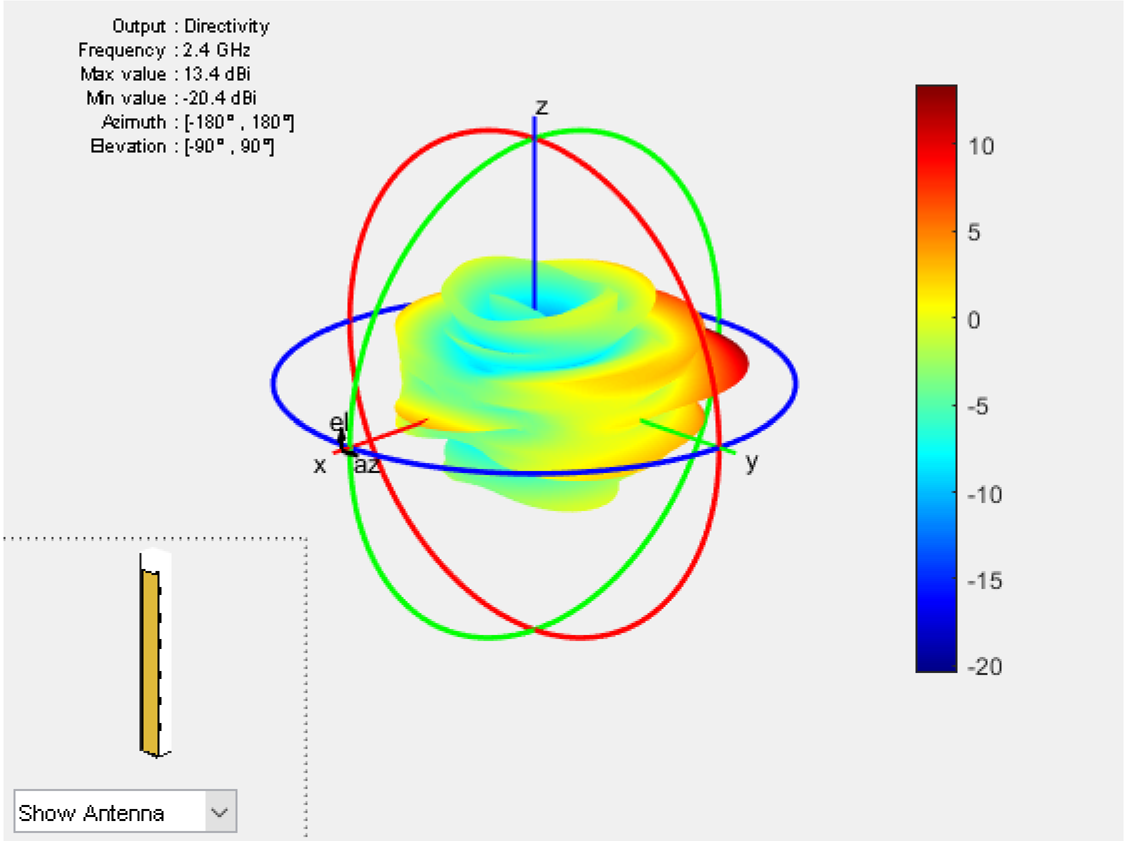
```
sectoria = 
  sectorInvertedAmos with properties:

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

```
show(sectoria)
```



Plot Radiation Pattern at 2.4 GHz

```
pattern(sectoria,2.4e9)
```



See Also

`dipoleMeander` | `reflector`

Introduced in R2017b

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

20 (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 or array structure; display shape as filled patch
mesh	Mesh properties of metal or dielectric antenna or array structure

Examples

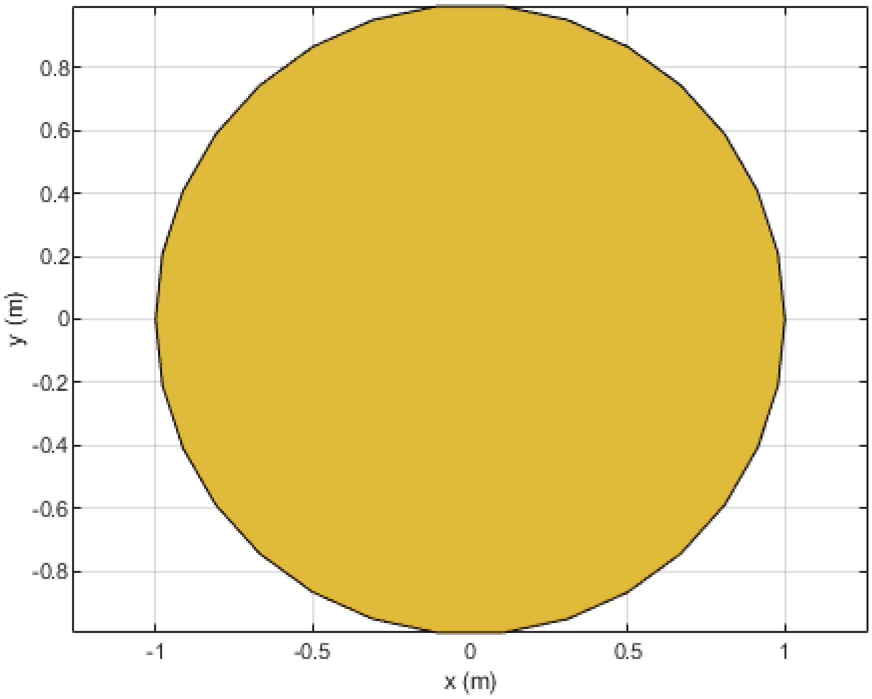
Create Circle with Default Properties

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 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.

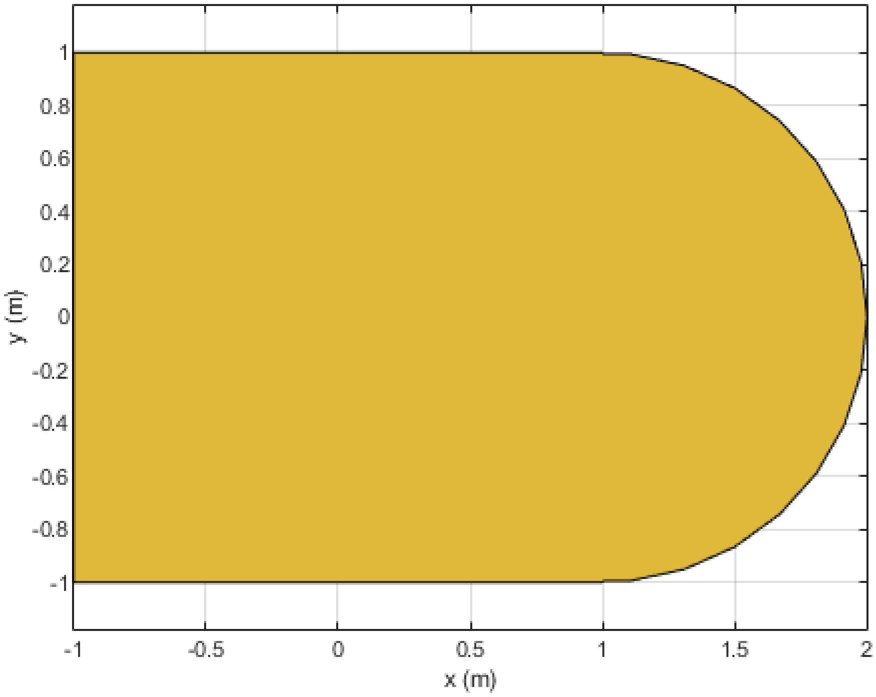
```
rect1 = antenna.Rectangle('Length',2,'Width',2);
```

Add the two shapes together using the + function.

```
polygon1 = circle1+rect1
```

```
polygon1 = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [21x3 double]
```

```
show(polygon1)
```



See Also

antenna.Polygon | antenna.Rectangle

Introduced in R2017a

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
[ ] (default) | *N*-by-3 vector

Cartesian coordinates of polygon vertices, specified as a *N*-by-3 vector 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 or array structure; display shape as filled patch
mesh	Mesh properties of metal or dielectric antenna or array structure

Examples

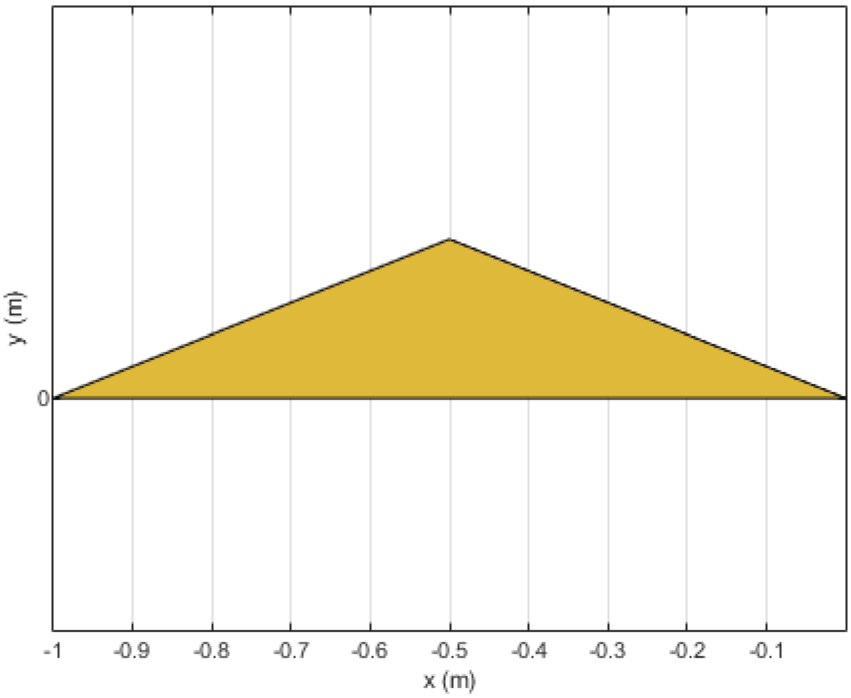
Create and Transform Polygon

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])
```

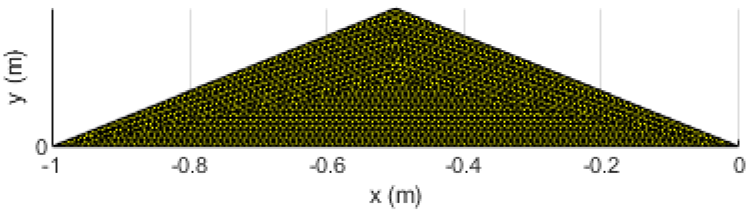
```
p = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [3x3 double]
```

```
show(p)
axis equal
```



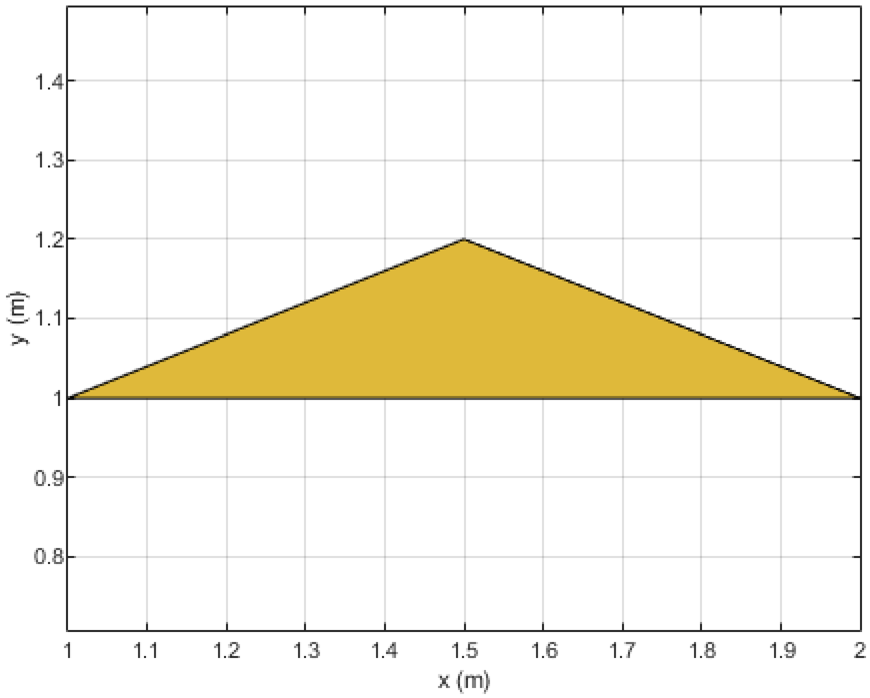
Mesh the polygon and view it.

```
mesh(p,0.2)
```



Move the polygon to a new location on the X-Y plane.

```
translate(p,[2,1,0])
axis equal
```



See Also

`antenna.Circle` | `antenna.Rectangle`

Introduced in R2017a

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 or array structure; display shape as filled patch
mesh	Mesh properties of metal or dielectric antenna or array structure

Examples

Create Rectangle with Default Properties

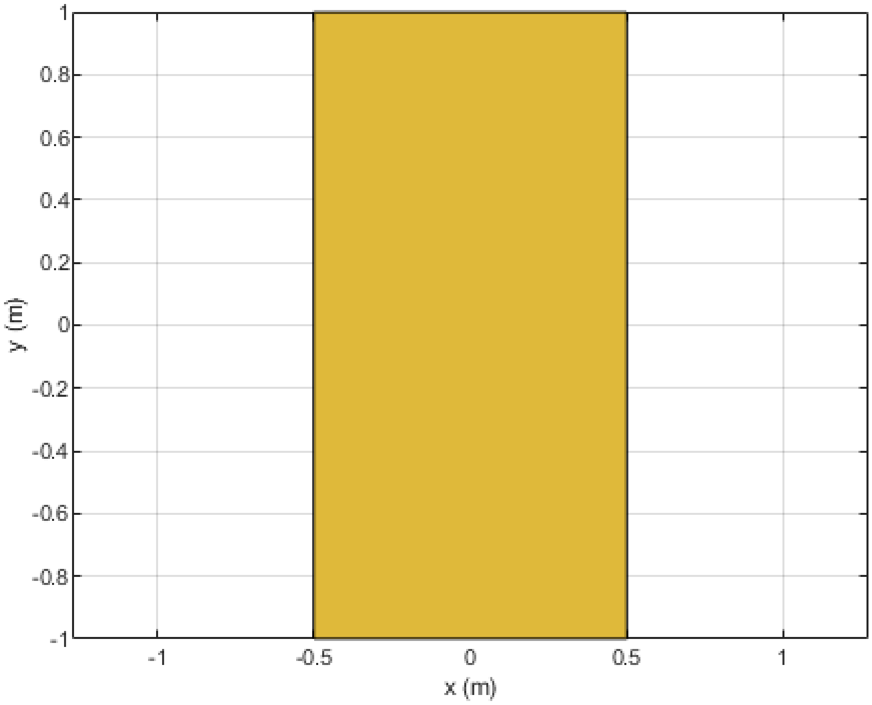
Create a rectangle shape using antenna.Rectangle and view it.

```
r1 = antenna.Rectangle
```

```
r1 = 
  Rectangle with properties:

         Name: 'myrectangle'
       Center: [0 0]
       Length: 1
```

```
        Width: 2
    NumPoints: 2
```

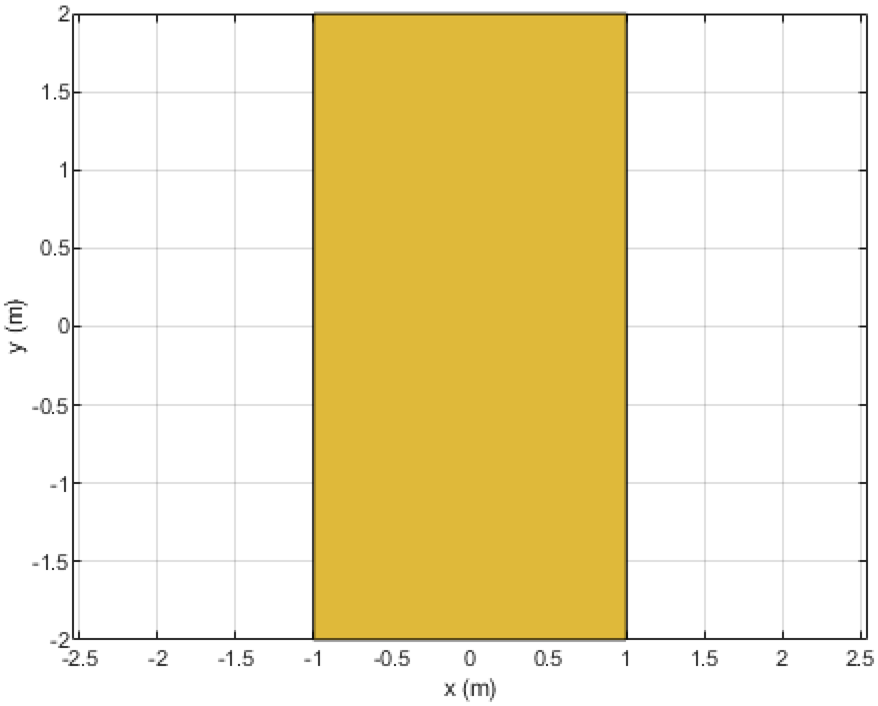
```
show(r1)
```



Create and Rotate Rectangle Using Specified Properties

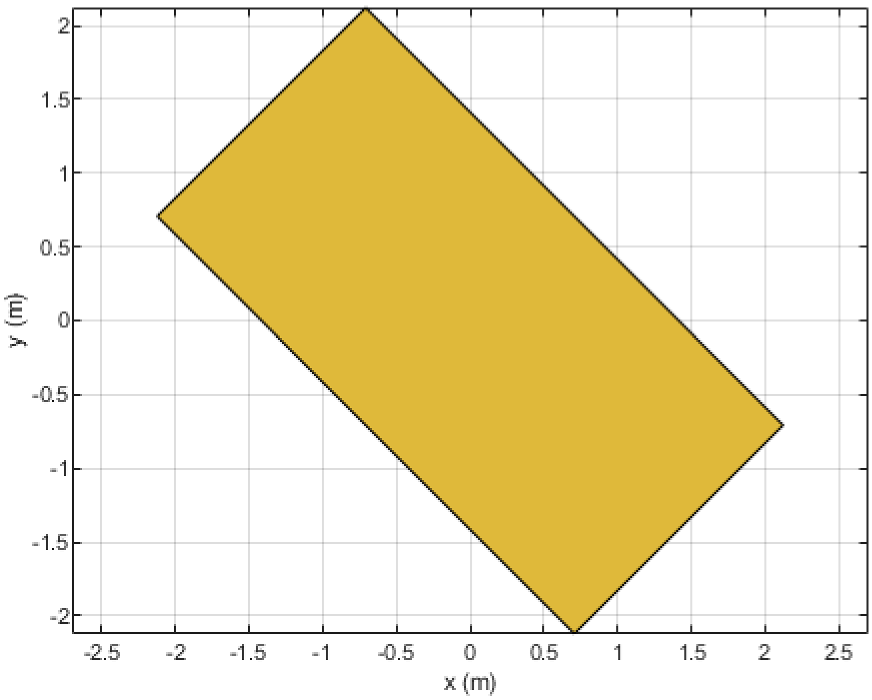
Create and view a rectangle with a length of 2 m and a width of 4 m.

```
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.

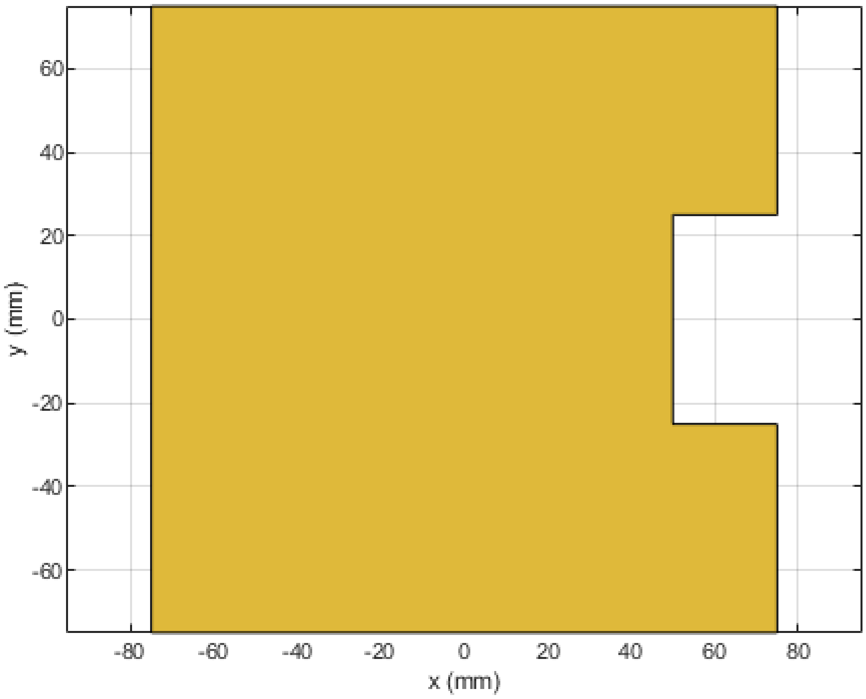
```
r  = antenna.Rectangle('Length',0.15,'Width',0.15);
```

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
```

See Also

`antenna.Circle` | `antenna.Polygon`

Introduced in R2017a

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)
```

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

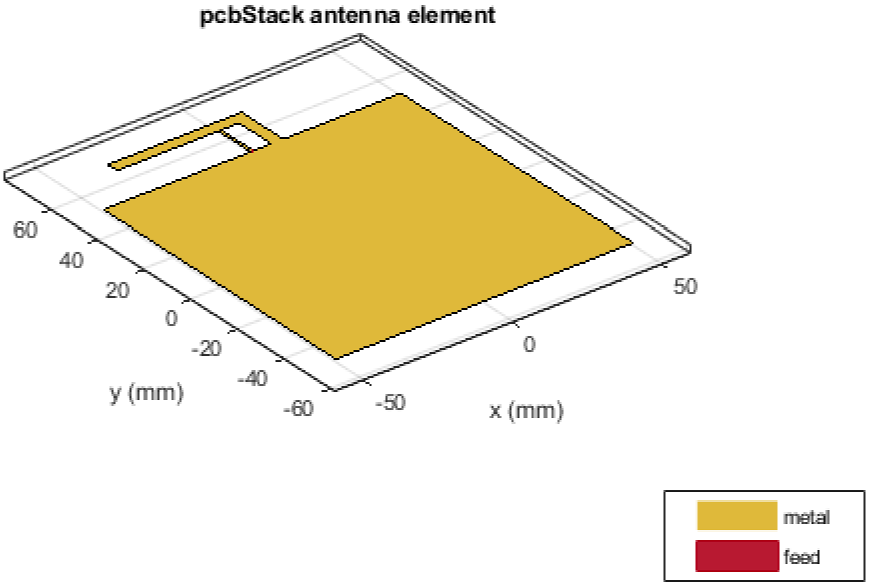
Generate Gerber Format Files From PCB Stack Object

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
```

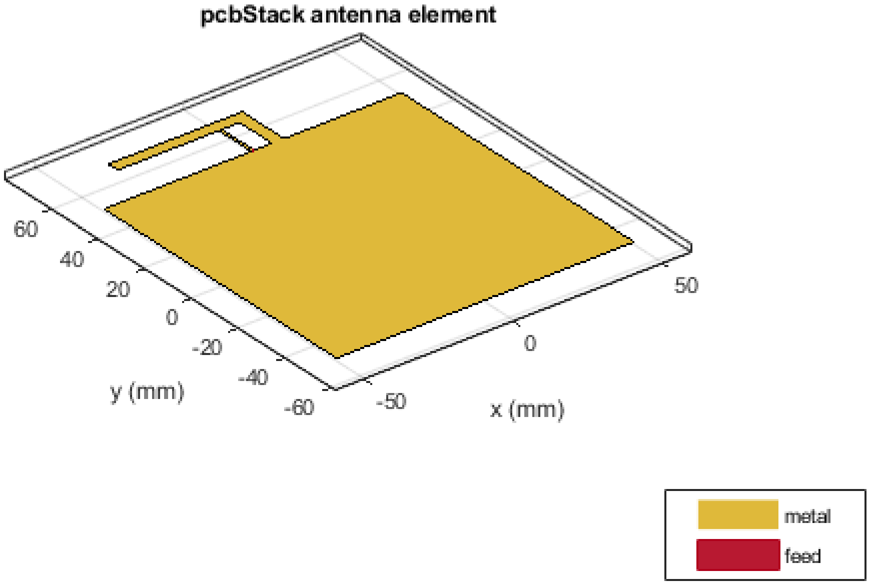
Antenna PCB Design Using SMA Cinch Connector

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
```

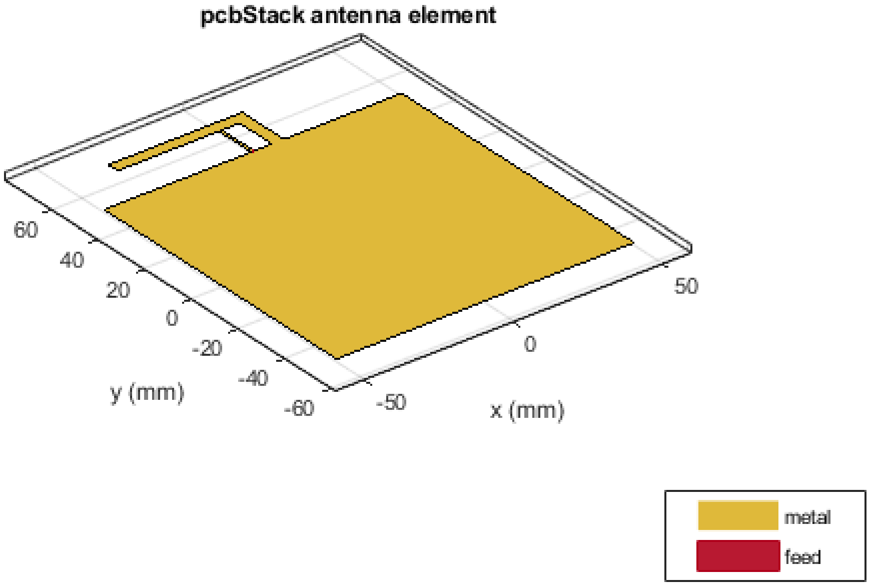
Antenna Design Files Using Advanced Circuits Writer Service

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.

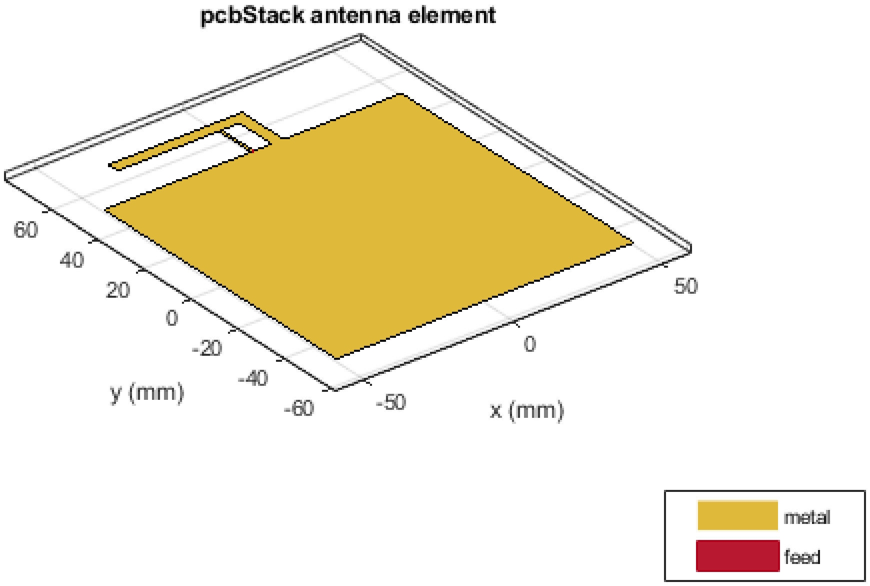
```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

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` .

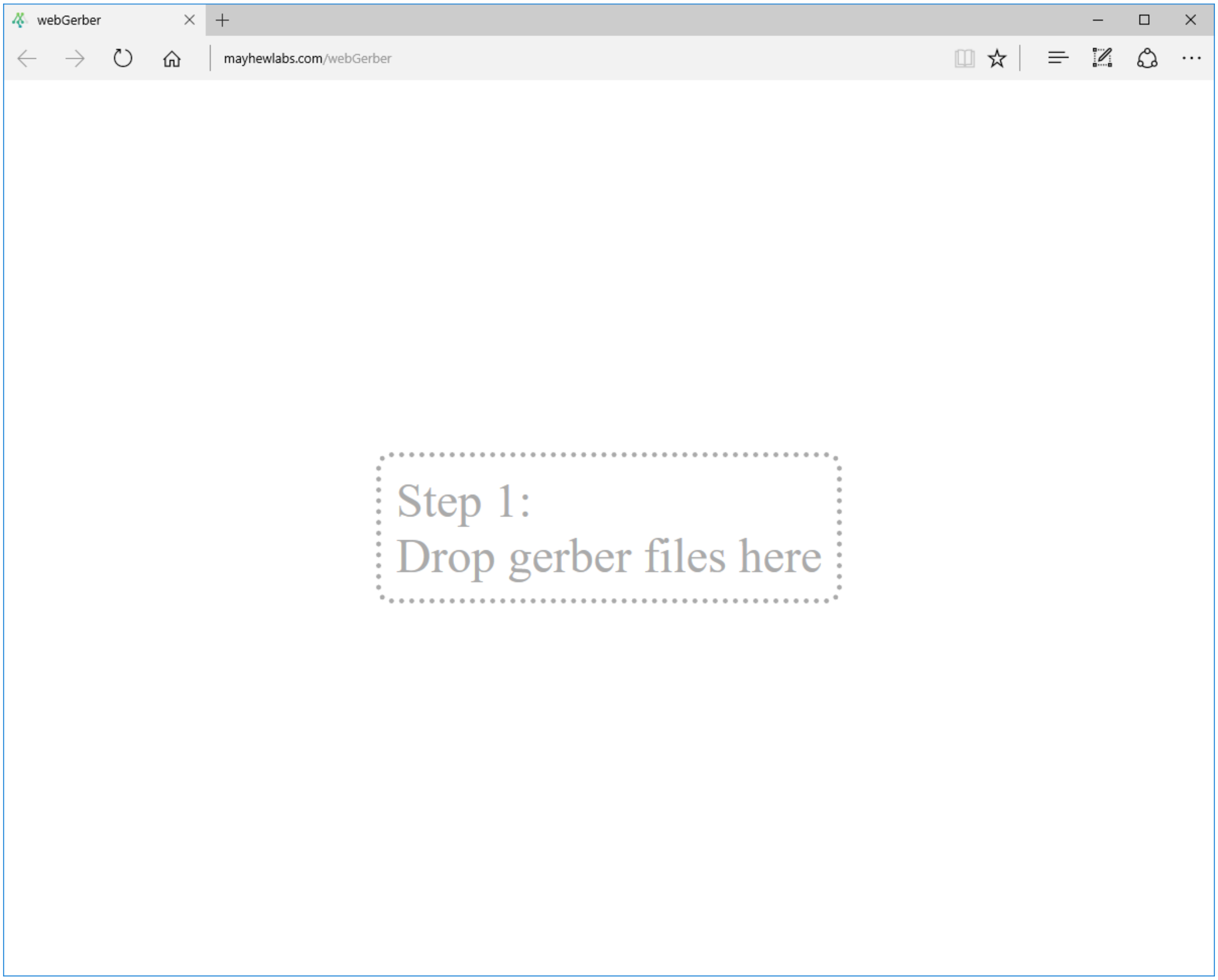
```
PW = PCBWriter(p,s,c)
```

```
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.

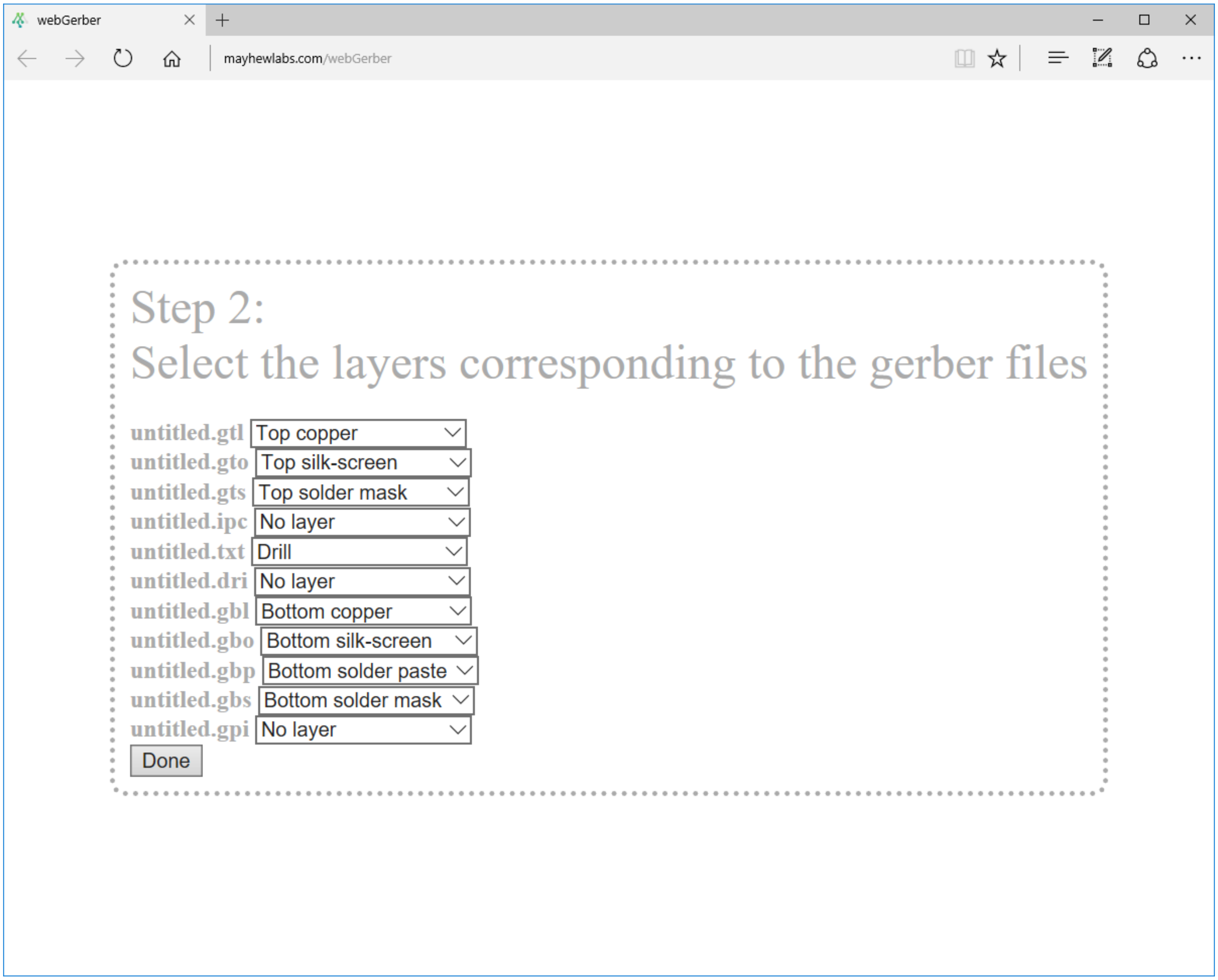
```
gerberWrite(PW)
```

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

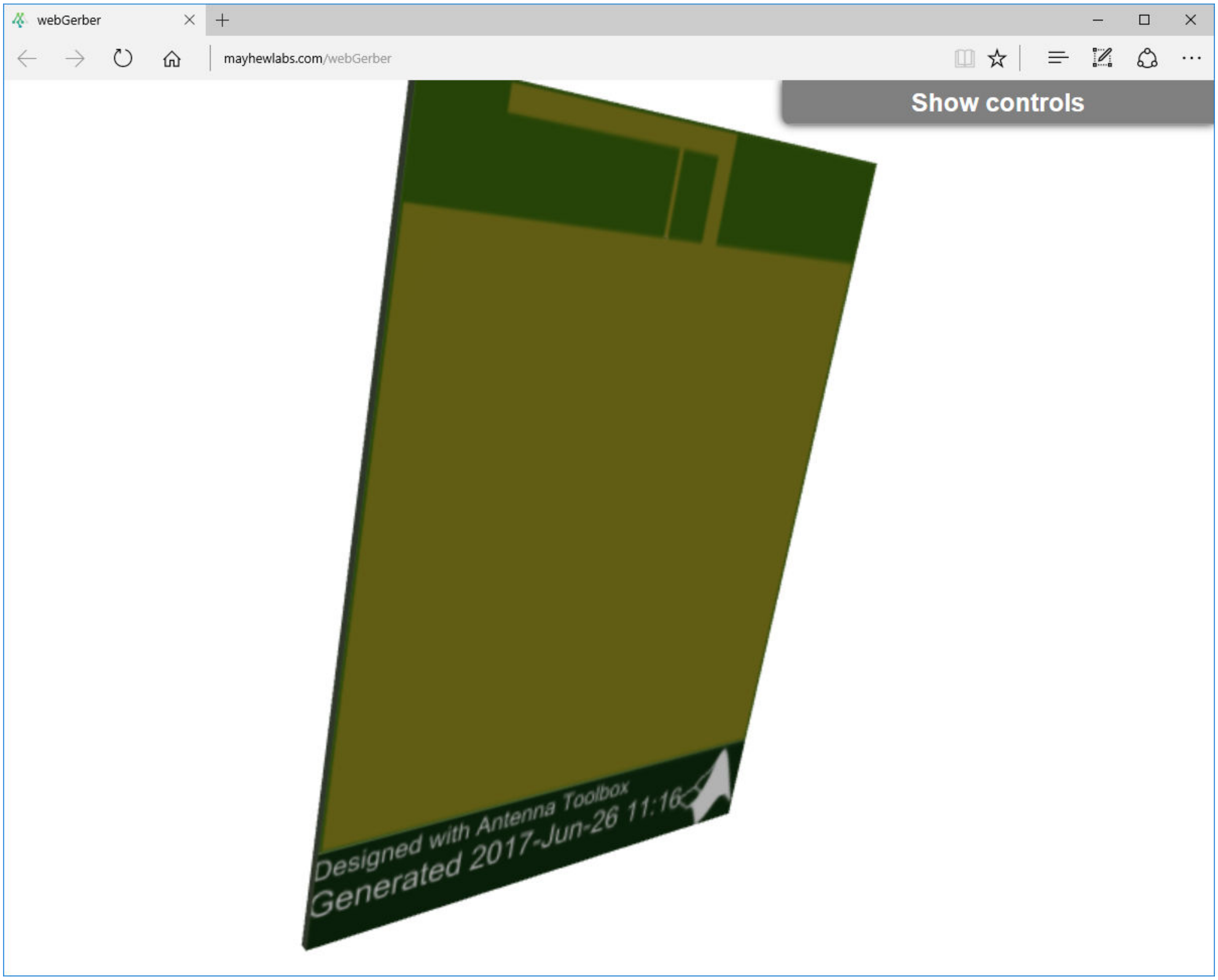
Drag and drop all your files from the "untitled" folder.



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.



See Also

PCBConnectors | PCBServices

Introduced in R2017b

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);
```

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`.

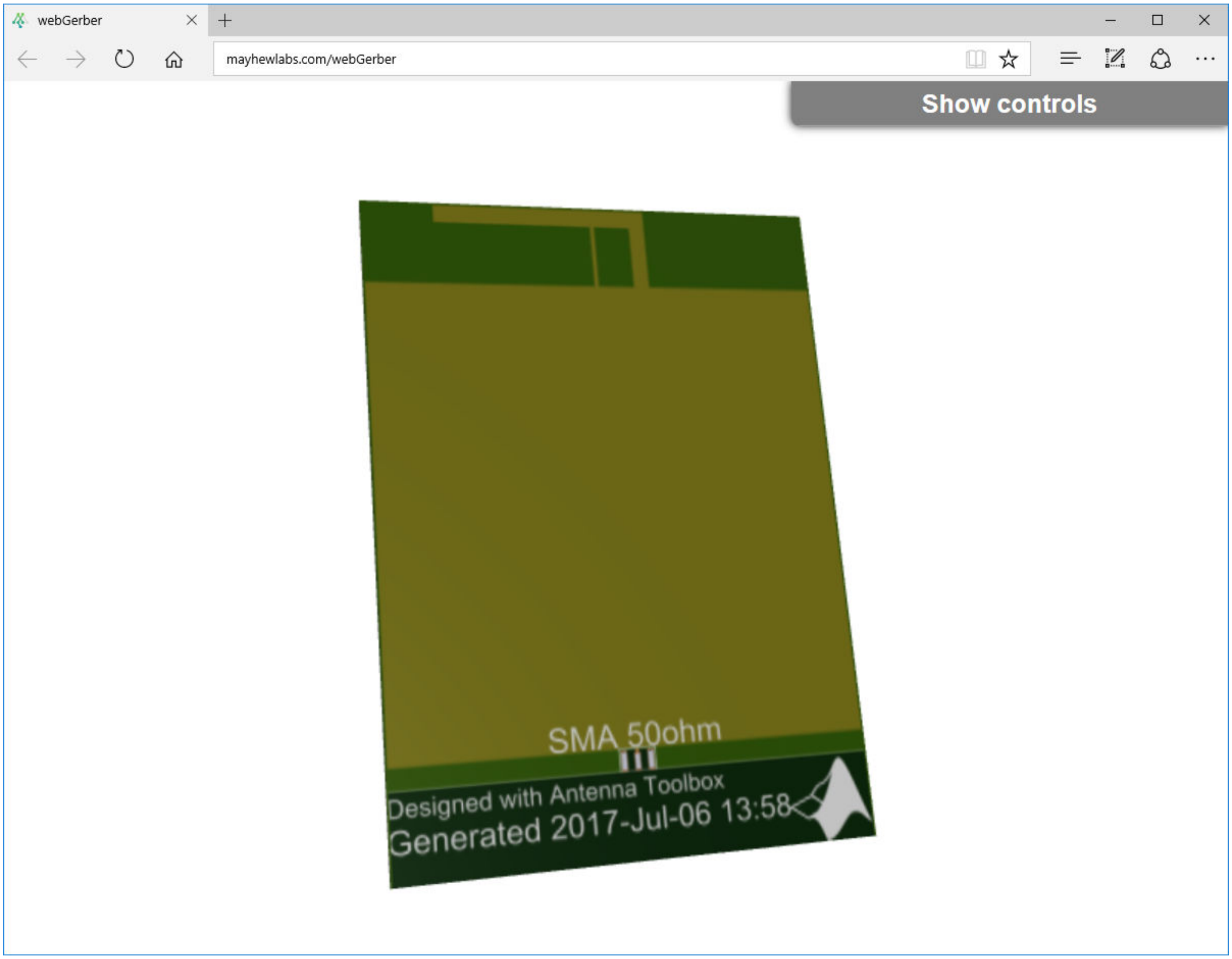
```
PW = PCBWriter(p,s);
```

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**.



See Also

`PCBConnectors` | `PCBWriter` | `gerberWrite`

Introduced in R2017b

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 2-343.

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

Type of connector, specified as a character vector.

Example: 'Coax_RG11'

Data Types: char | string

Mfg — Name of component manufacturer
character vector

Name of component manufacturer, specified as a character vector.

Example: 'Belden'

Data Types: char | string

Part — Manufacturer part number
character vector | string

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

Text added to PCB to identify component, specified as a character vector.

Example: 'RG59U'

Data Types: char | string

Impedance — Connector impedance
50 | positive scalar

Connector impedance, specified as a positive scalar in ohms.

Example: c = PCBConnectors.MMCX_Cinchc.Impedance = 70

Data Types: double

Datasheet — URL for component specifications
character vector

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

URL for purchasing connector, specified as a character vector.

Data Types: `char` | `string`

Common Properties for All Coax Connectors

`SignalPinDiameter` — 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_RG59c.SignalPinDiameter = 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_RG59c.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_RG59c.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_RG59c.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_RG59c.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_RG59c.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_Multicompc.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_Multicompc.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_Multicompc.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_Multicompc.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_Multicompc.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_Multicompc.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.SMAEdgec.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.SMAEdgec.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.SMAEdgec.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.SMAEdgec.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.SMAEdgec.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.SMAEdgec.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.SMAEdgec.EdgeLocation = 'south'`

Data Types: `char`

`EdgeBoardProfile` — 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.SMAEdgec.EdgeBoardProfile = 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.SMAEdgec.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_Lighthorsec.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_Lighthorsec.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_Lighthorsec.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_Lighthorsec.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_Lighthorsec.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);
```

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;
```

```
c = 

  Coax_RG11 with properties:

                  Type: 'Coax'
                   Mfg: 'Belden'
                  Part: 'RG11'
            Annotation: 'RG11'
             Impedance: 75
             Datasheet: 'http://www.belden.com/techdatas/english/8233.pdf'
              Purchase: ''
           PinDiameter: 0.0020
      DielectricDiameter: 0.0072
          ShieldDiameter: 0.0085
        ThermalsDiameter: 0.0100
     ThermalsBridgeWidth: 0.0015
```

```
            AddThermals: 1

   <a href="matlab:web('http://www.belden.com/techdatas/english/8233.pdf','-browser');">Belden RG
```

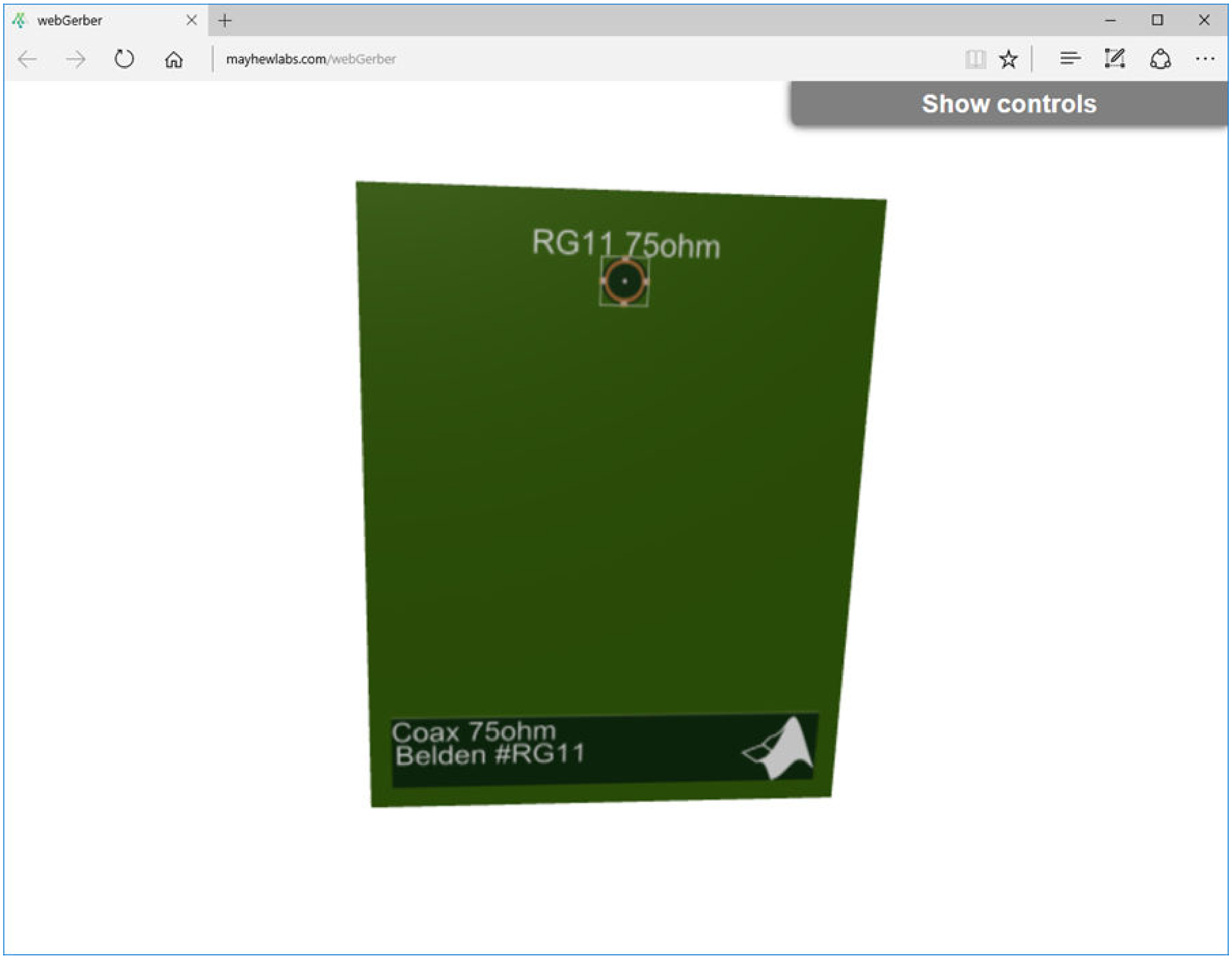
Create an antenna design file using `PCBWriter` .

```
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

```
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  = 'http://www.farnell.com/datasheets/1720451.pdf?_ga=2.164811836.2075200750.1
        Purchase   = 'http://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.CoaxRG11	RG11 coaxial cable direct soldered to PCB pads.
PCBConnectors.CoaxRG58	RG58 coaxial cable direct soldered to PCB pads.
PCBConnectors.CoaxRG59	RG59 coaxial cable direct soldered to PCB pads.
PCBConnectors.CoaxRG174	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

PCB Connectors	Descriptions
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
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

See Also

PCBServices | PCBWriter | gerberWrite

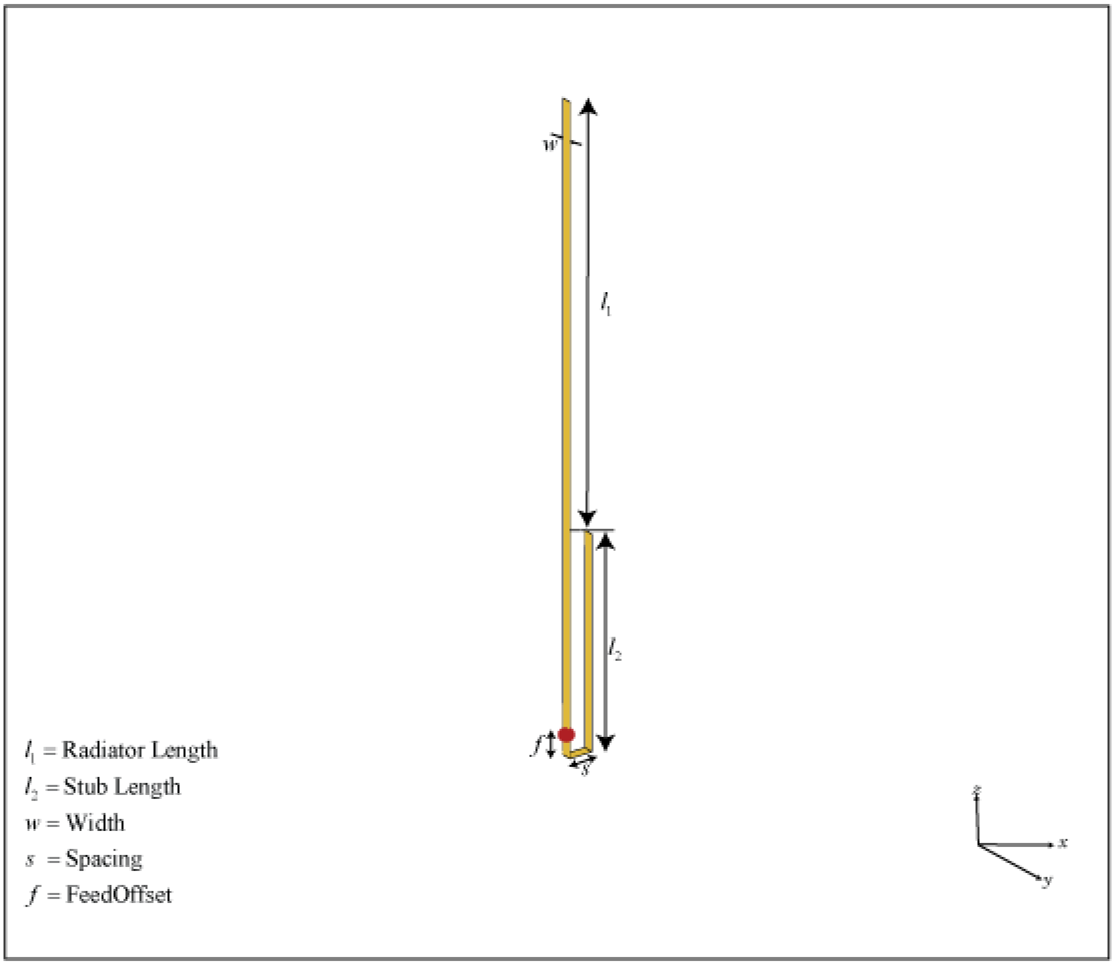
Introduced in R2017b

dipoleJ

Create J-dipole antenna

Description

Use the `dipoleJ` object to create a J-dipole on the Y-Z 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`

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 origin. For more information, see `lumpedElement`.

Example: `'Load',lumpedelement`, where, `lumpedelement` is the object handle 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

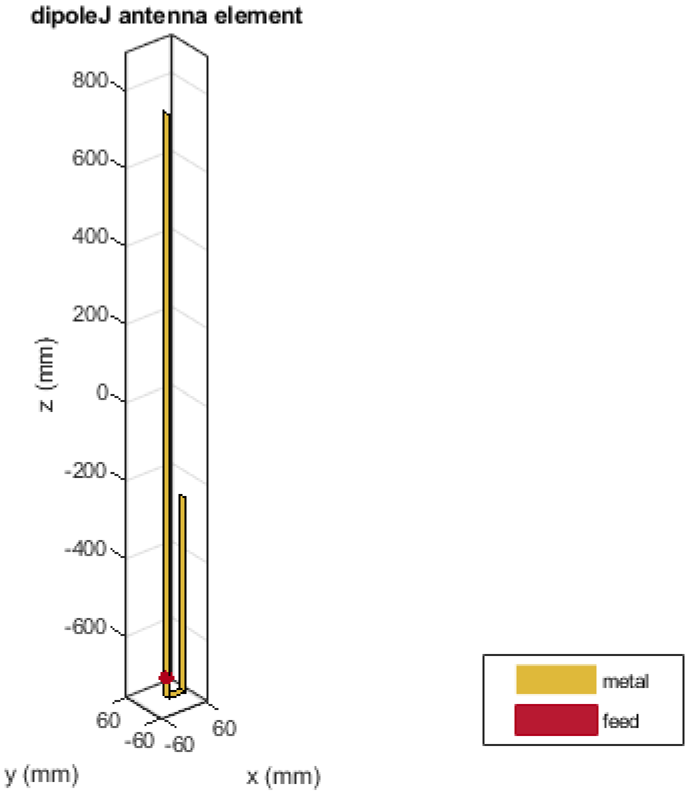
Default J-Dipole Antenna

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

```
show(d)
```



Impedance of J-Dipole Antenna

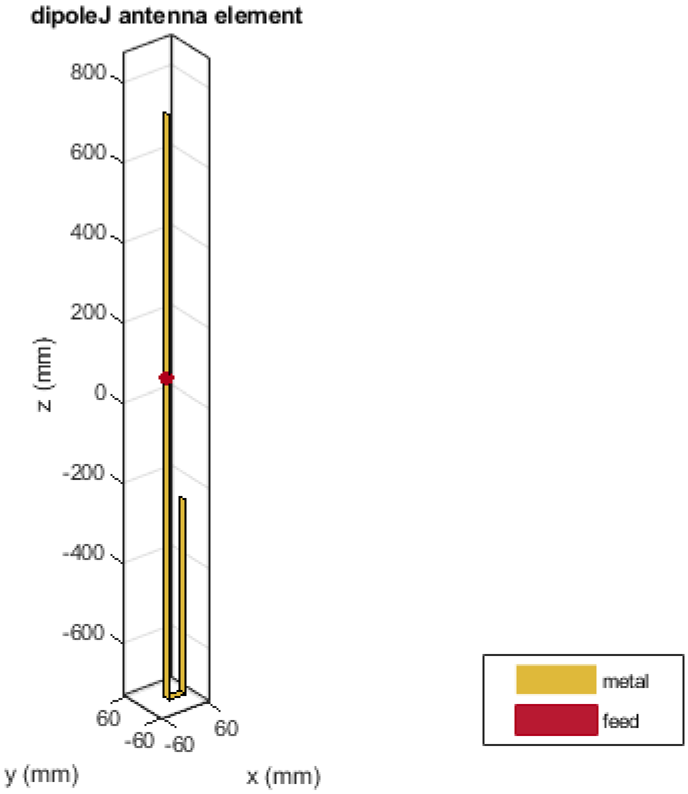
Create and view a J-dipole antenna with the following specifications:

Radiator length = 0.978 m

Stub length = 0.485 m

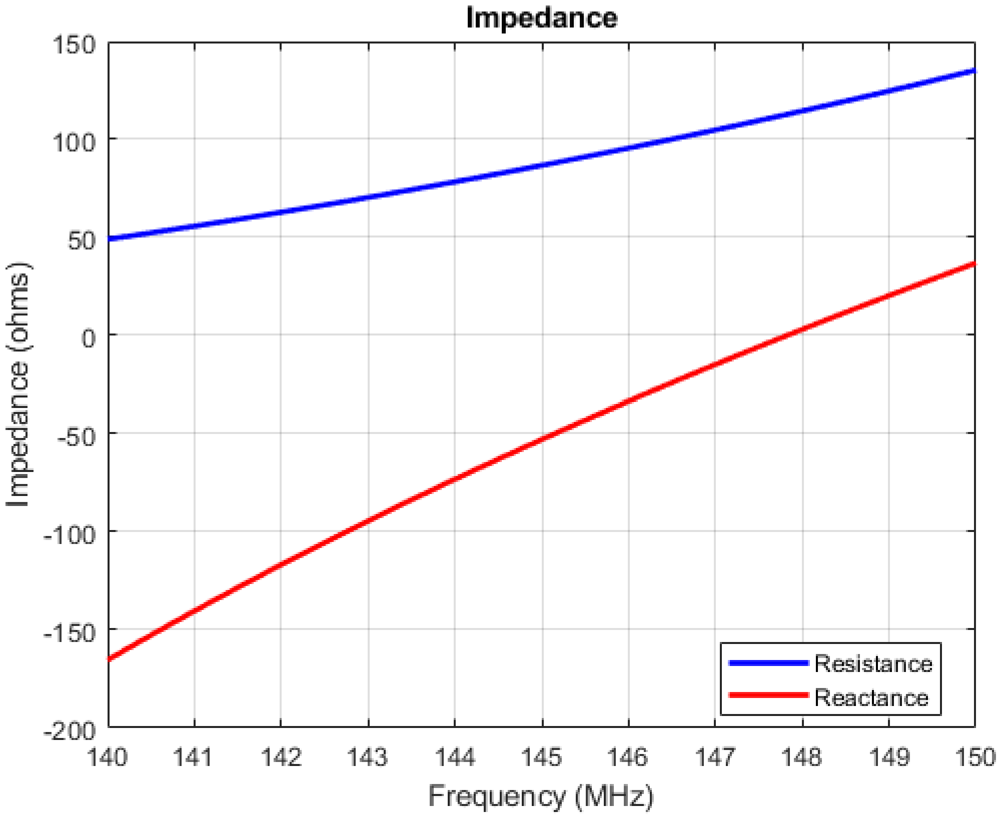
FeedOffset = 0.049 m

```
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));
```



See Also

`dipole` | `dipoleFolded` | `dipoleVee`

Topics

"Rotate Antennas and Arrays"

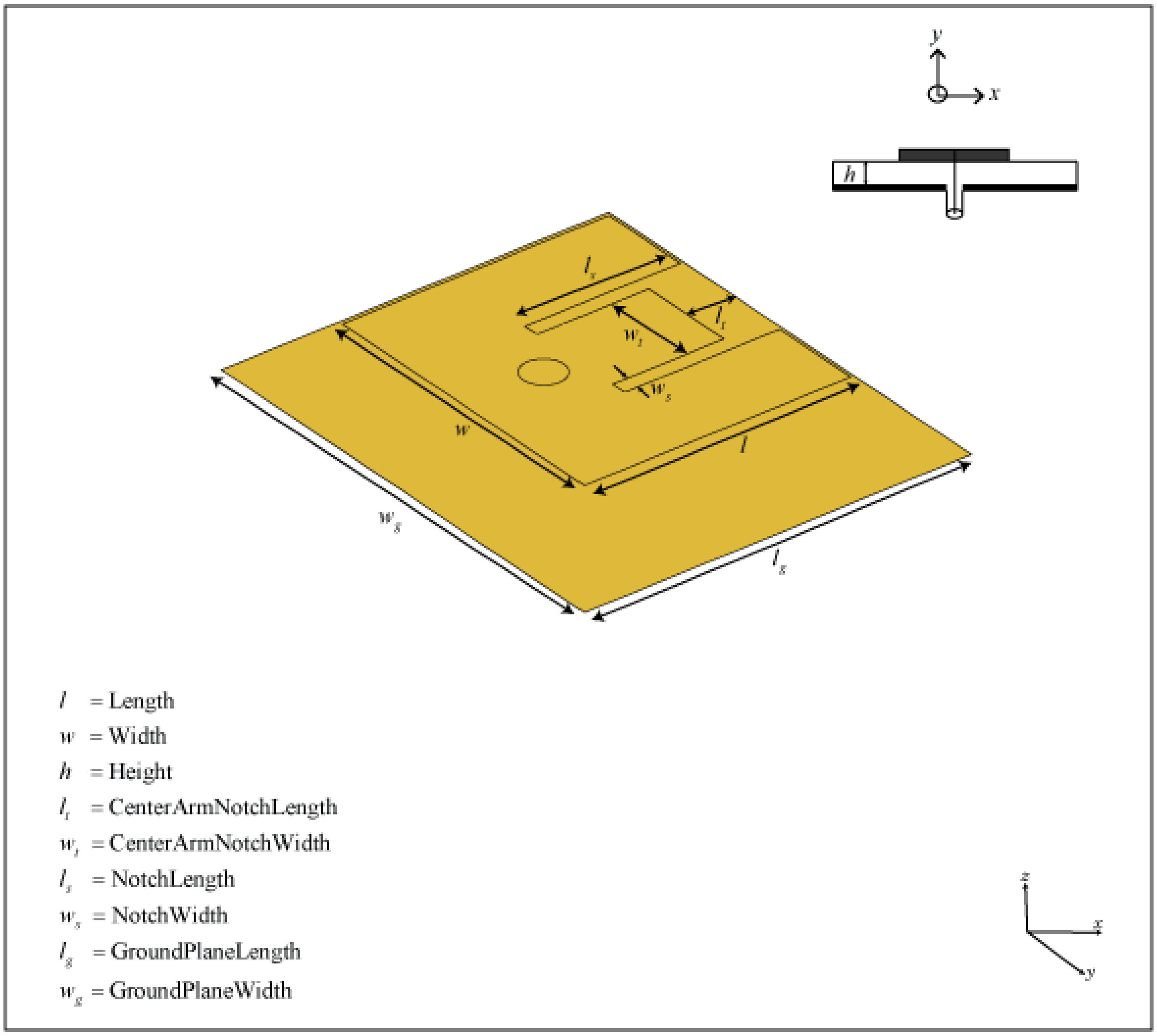
Introduced in R2018a

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`

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 handle 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

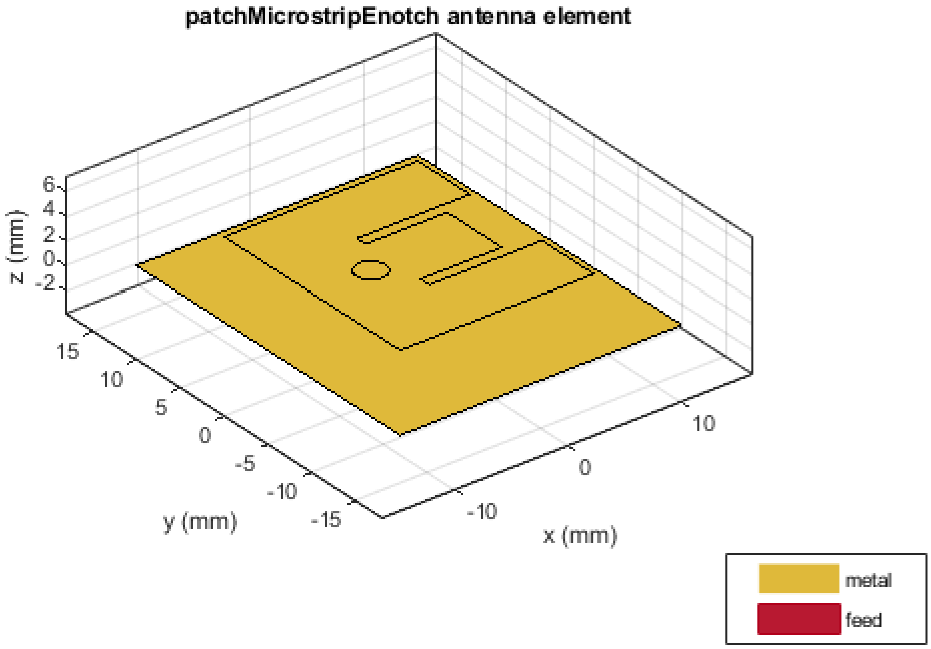
Default E-Shaped Patch Antenna

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

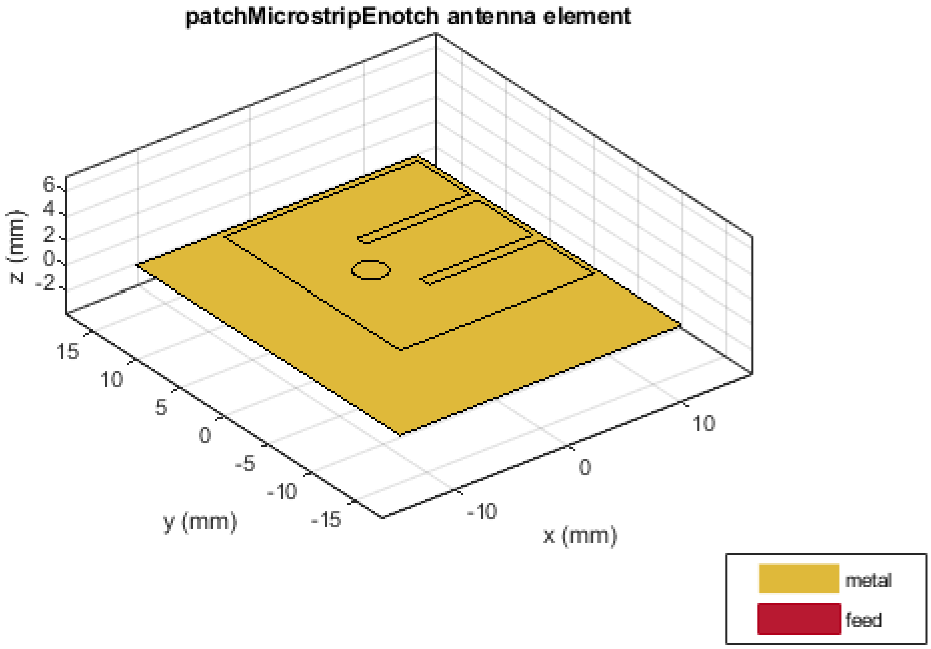
```
show(epatch)
```



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);
```



See Also

`patchMicrostrip` | `patchMicrostripCircular` | `patchMicrostripTriangular`

Topics

"Rotate Antennas and Arrays"

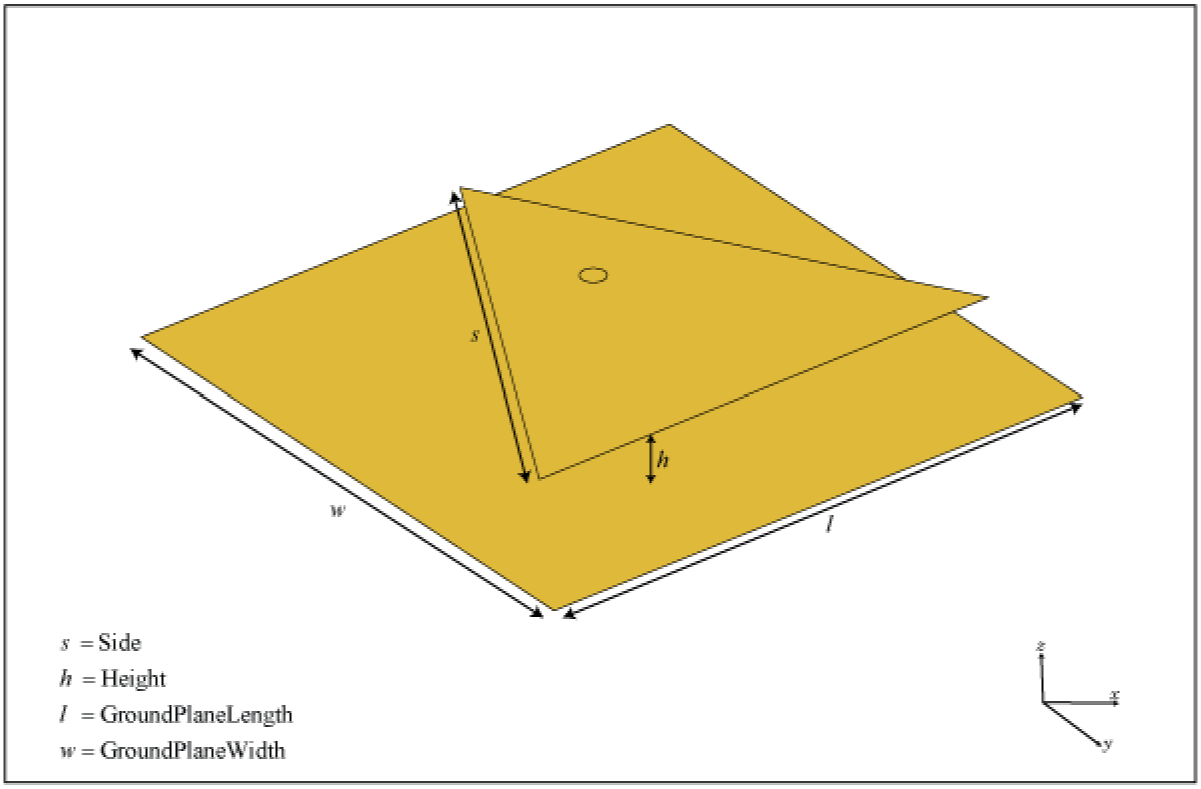
Introduced in R2018a

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

Substrate — Type of dielectric material

'Air' (default) | dielectric function handle

Type of dielectric material used as a substrate, specified as a dielectric material object handle. 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 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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

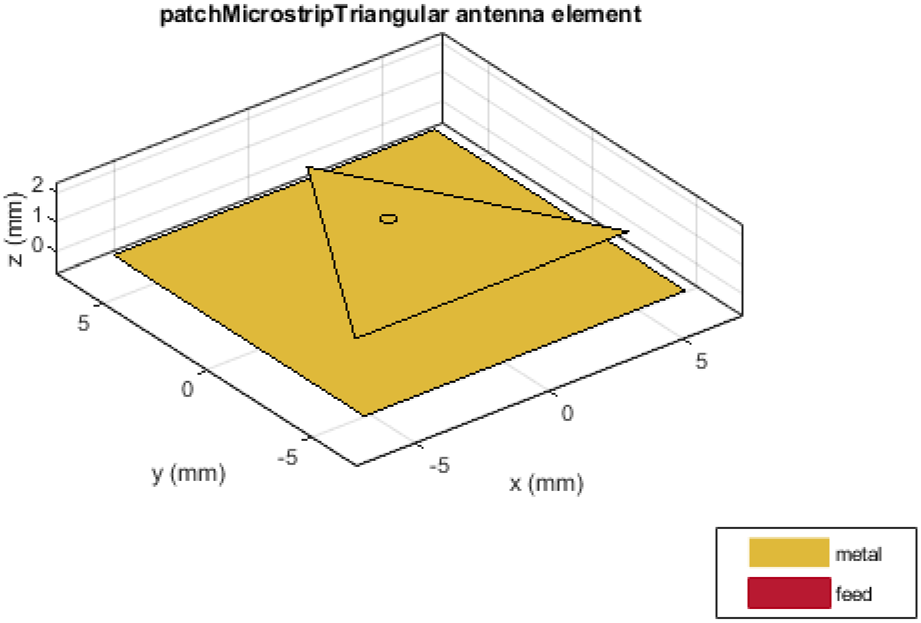
Default Triangular Microstrip Patch and Radiation Pattern

Create and view a default triangular microstrip patch.

```
p = patchMicrostripTriangular
```

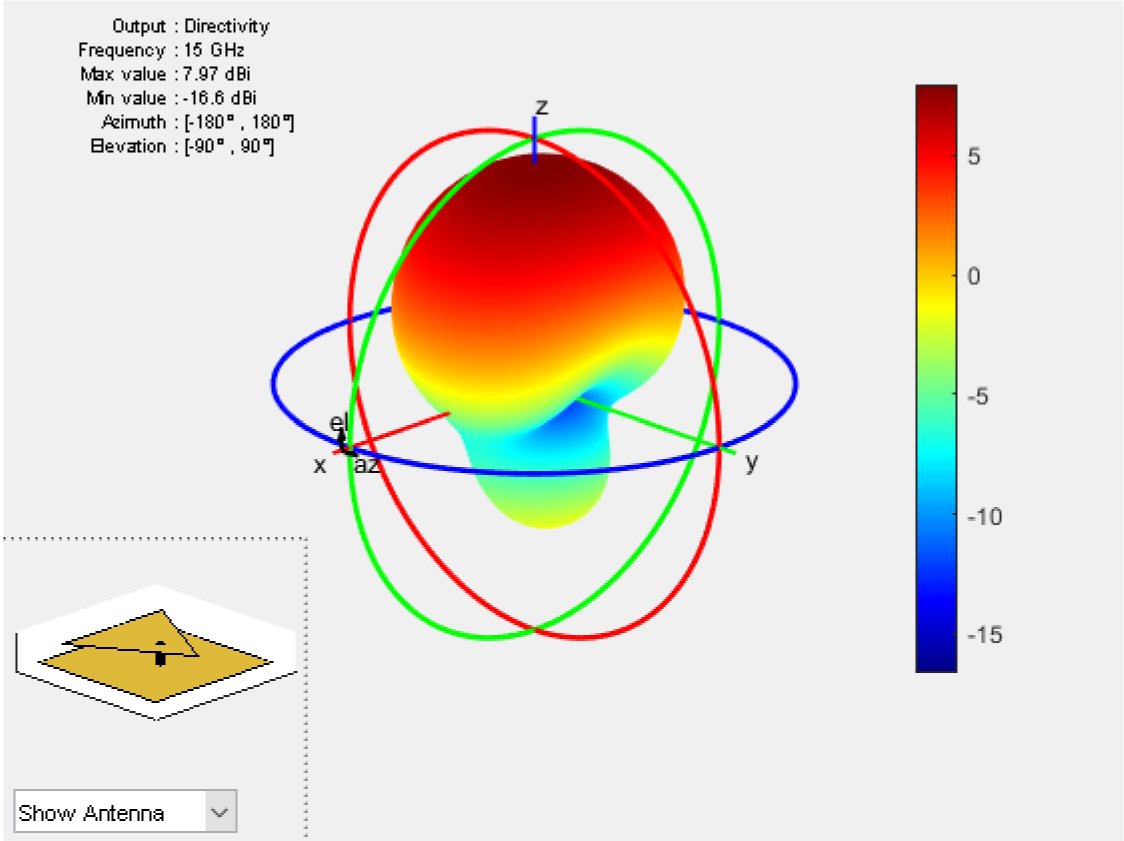
```
p = 
  patchMicrostripTriangular with properties:

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

```
show(p)
```



Plot the radiation pattern at 15 GHz.

```
pattern(p,15e9)
```



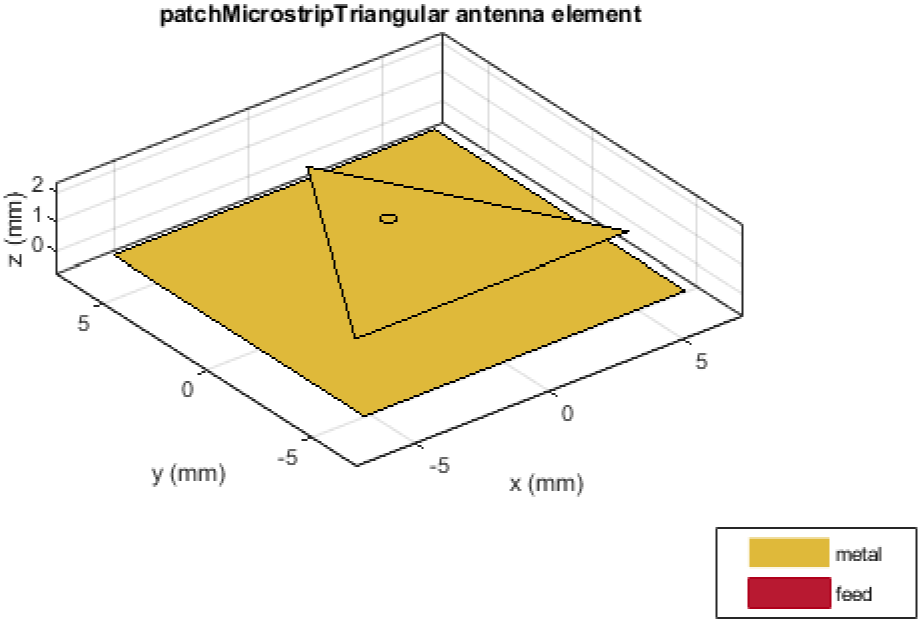
Different Types of Triangular Patch Antennas

Create different types of triangles to use in the patch.

Equilateral Triangle

Create an equilateral triangle patch of side 10.2 mm.

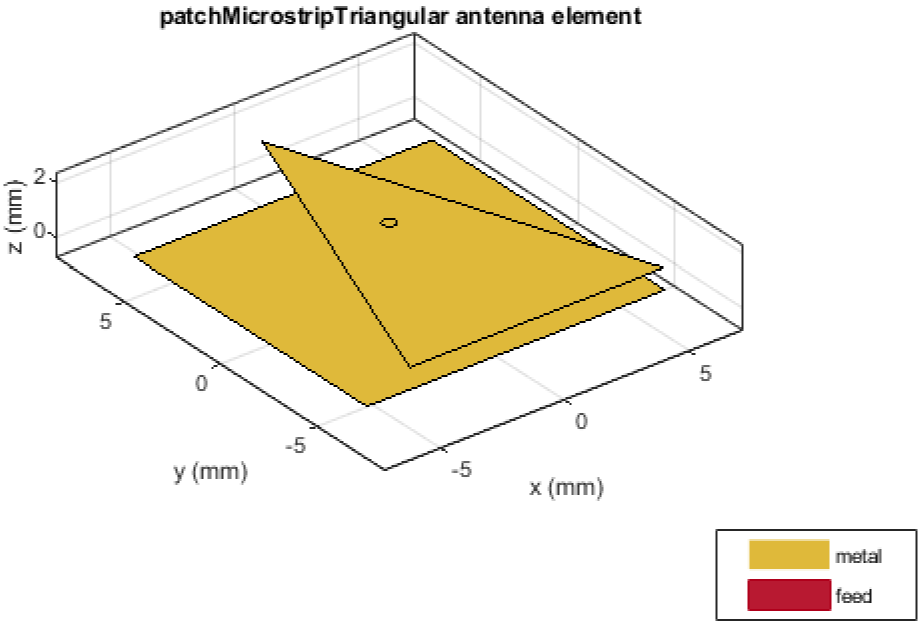
```
ant = patchMicrostripTriangular('Side',10.2e-3);
show(ant);
```



Isosceles Triangle

Create an isosceles triangular patch antenna with sides using the following dimensions: 10.2 mm and 15 mm.

```
ant =  patchMicrostripTriangular('Side',[10.2e-3,15e-3]);
show(ant);
```

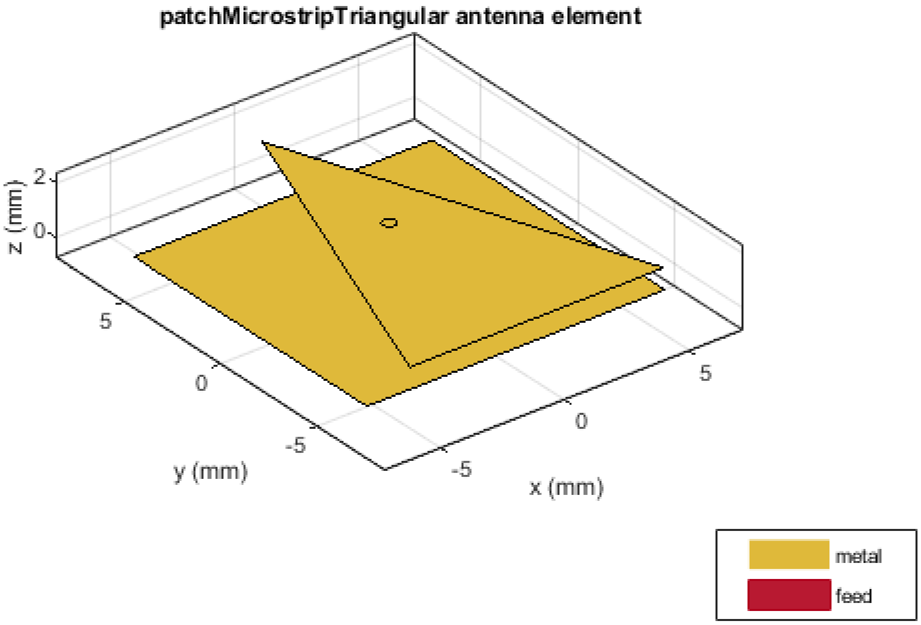


In the above figure, you will see that the first value of the side is chosen as the base of the triangle.

Scalene Triangle

Create a scalene triangular patch antenna with side using the following dimensions: 21 mm, 13 mm, and 20 mm.

```
patchMicrostripTriangular('Side',[21e-3,13e-3,20e-3]);
show(ant);
```

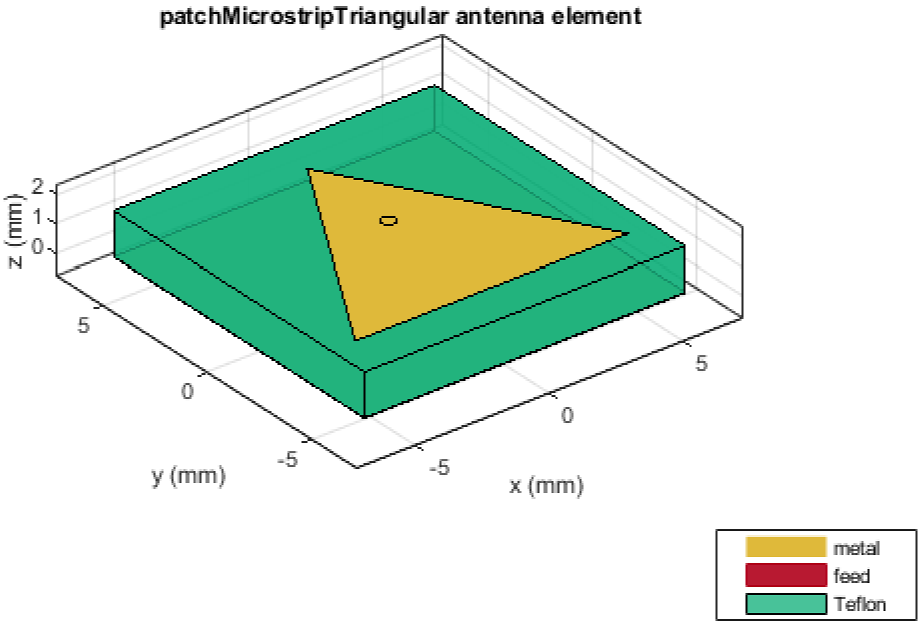


In the above figure, you will see that the first value of the side is chosen as the base of the triangle.

Triangle Patch Using Teflon Substrate and Radiation Pattern

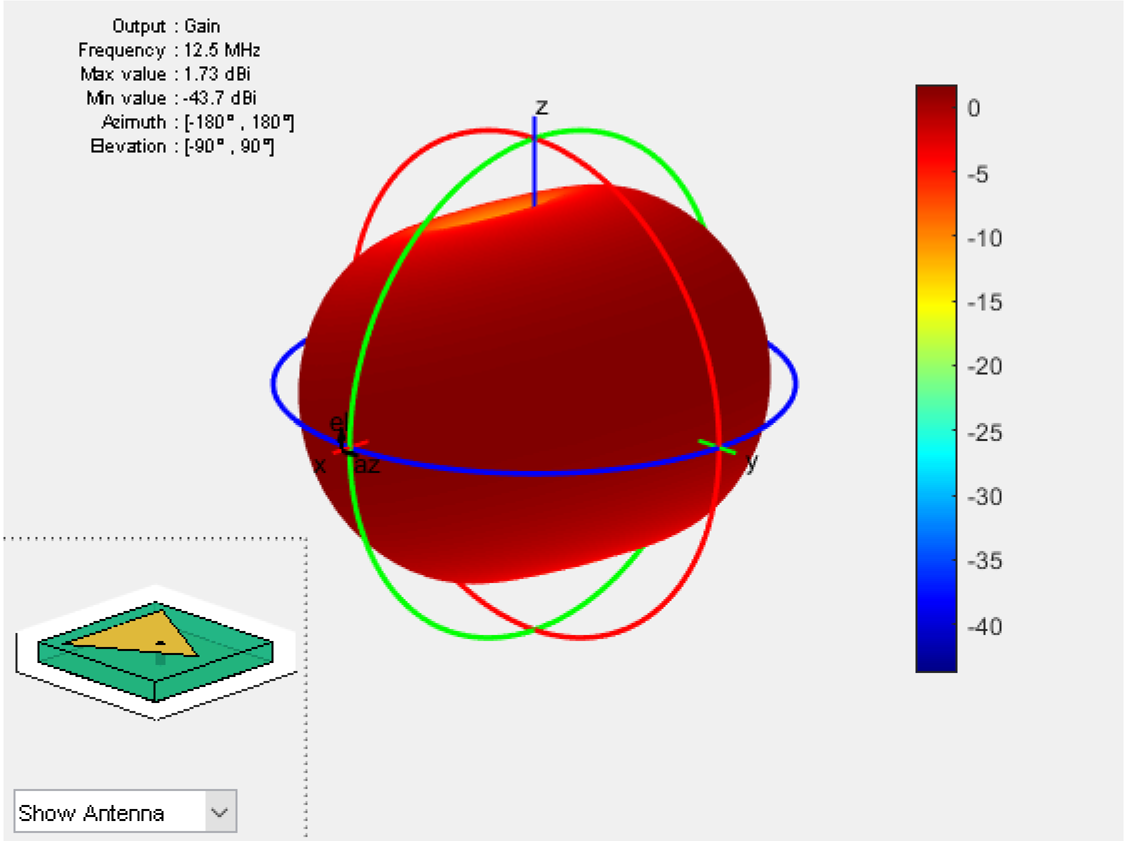
Create and view a triangular microstrip patch using Teflon substrate.

```
d = dielectric('Teflon');
p = patchMicrostripTriangular('Substrate',d);
show(p)
```



Plot the radiation pattern of the antenna.

```
pattern(p,12.5e6)
```



See Also

`patchMicrostrip` | `patchMicrostripCircular` | `patchMicrostripEnotch`

Topics

“ISM Band Patch Microstrip Antennas and Mutual Coupling Between different patches”
“Rotate Antennas and Arrays”

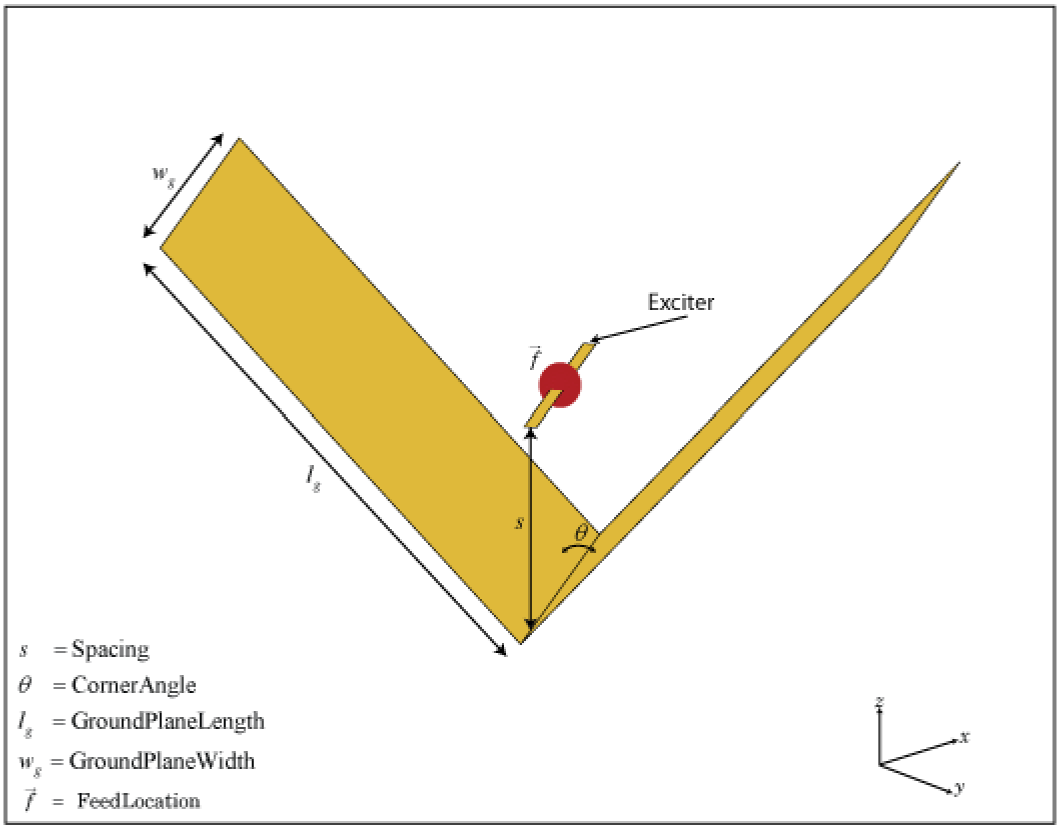
Introduced in R2018a

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 type used as exciter

dipole (default) | antenna object

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

Example: `'Exciter',spiralEquiangular`

Example: `cornerreflector.Exciter = spiralEquiangular`

`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

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 handle 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: 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 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 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 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'

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

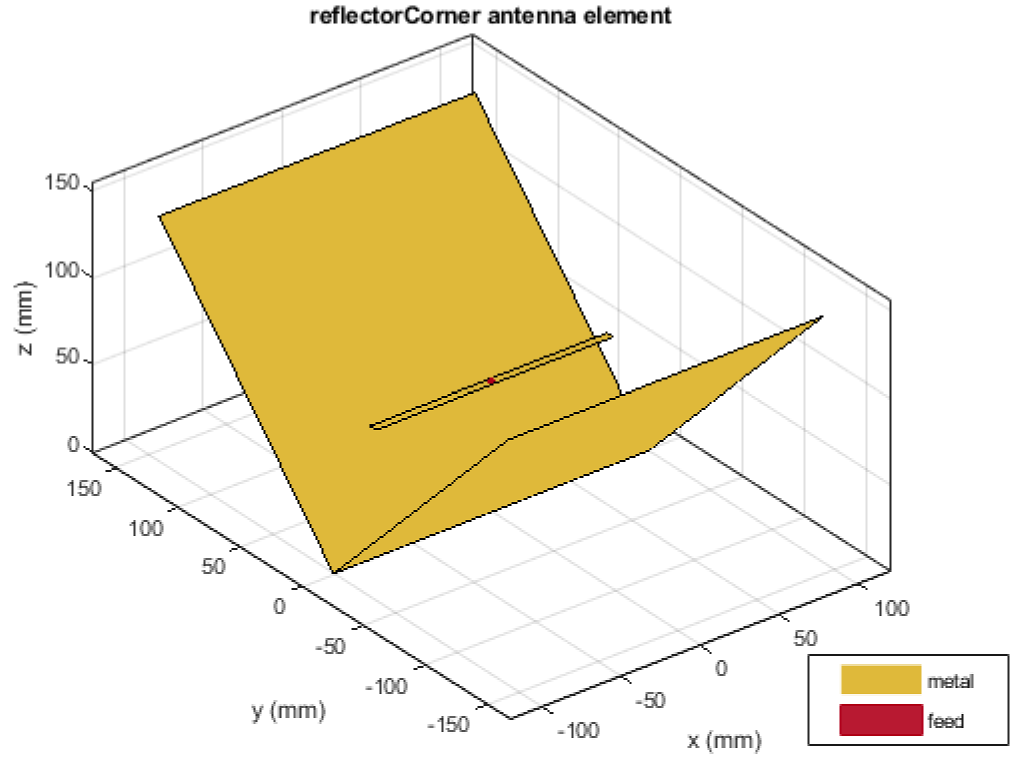
Default Corner Reflector-Backed Antenna and Radiation Pattern

Create and view a corner reflector-backed dipole.

```
cornerreflector = reflectorCorner
```

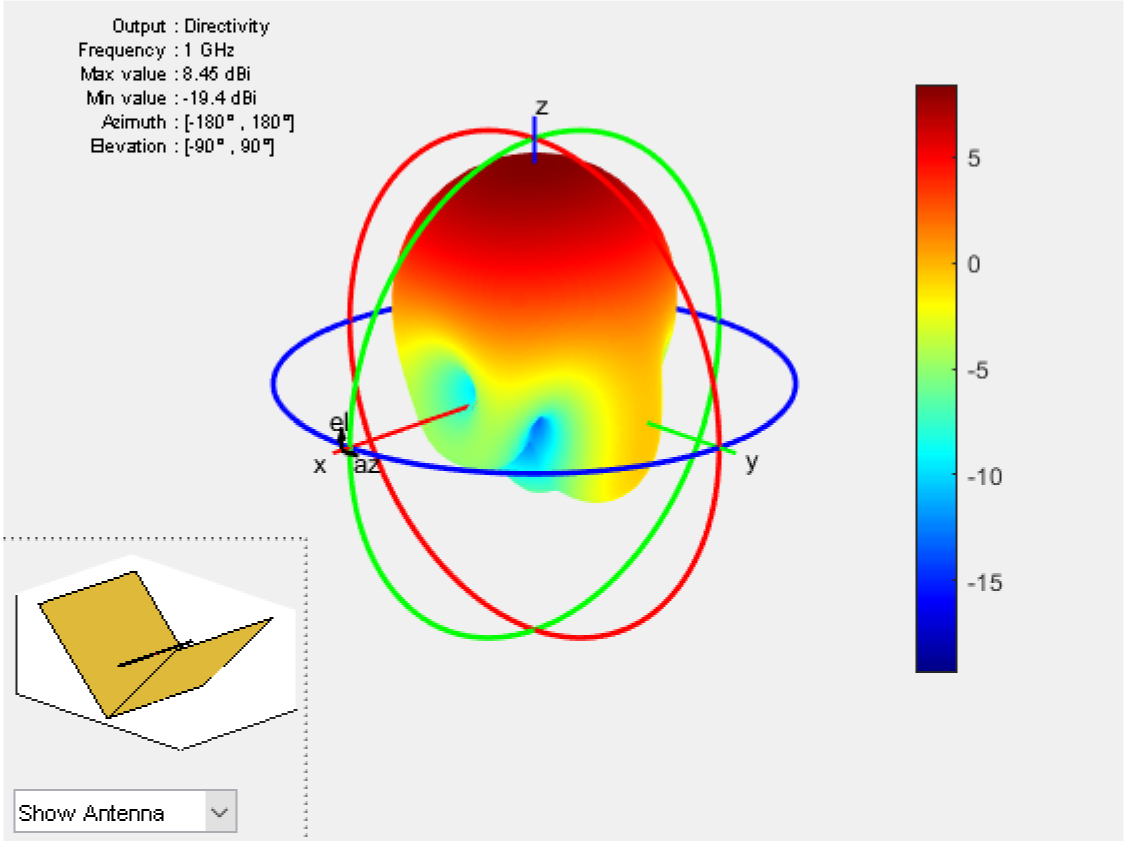
```
cornerreflector = 
  reflectorCorner with properties:

              Exciter: [1x1 dipole]
    GroundPlaneLength: 0.2000
     GroundPlaneWidth: 0.4000
          CornerAngle: 90
              Spacing: 0.0750
                 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 : 8.45 dBi
Min value : -19.4 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

See Also

`reflector` | `reflectorCircular`

Topics

"Rotate Antennas and Arrays"

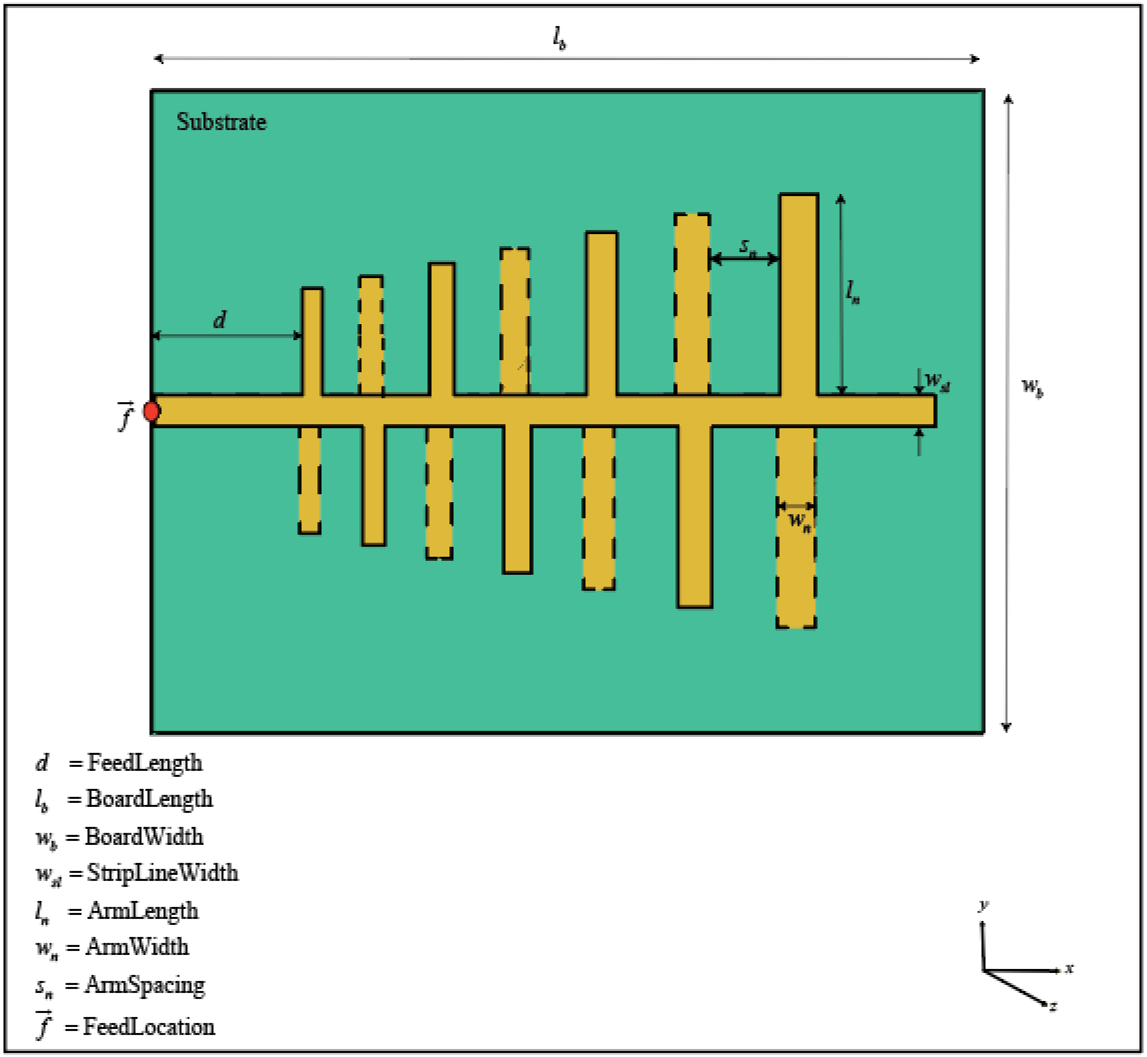
Introduced in R2018a

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`

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 handle 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

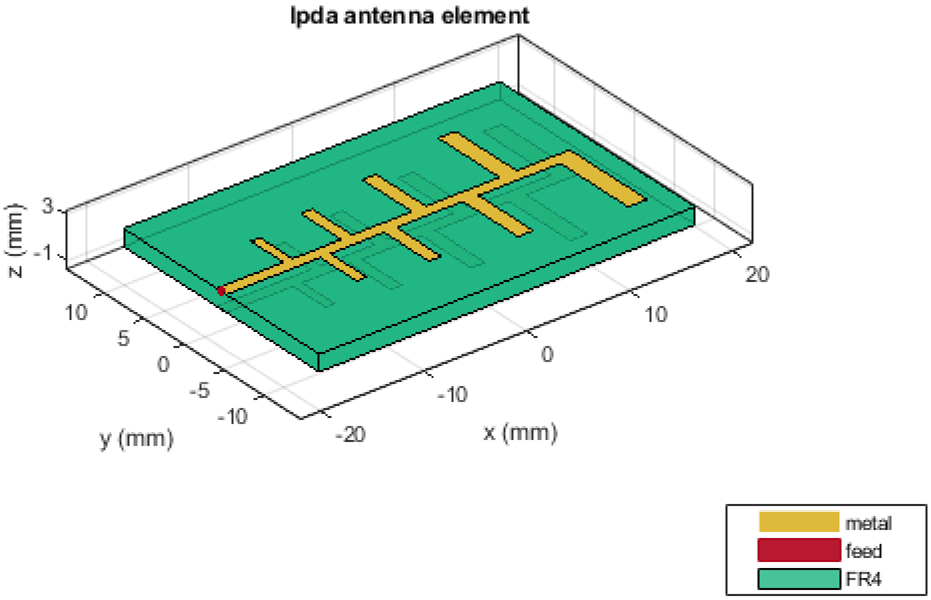
Default Printed Log-Periodic Antenna

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: [1x8 double]
        ArmSpacing: [0.0027 0.0030 0.0033 0.0037 0.0041 0.0046 0.0051]
         Substrate: [1x1 dielectric]
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1x1 lumpedElement]
```

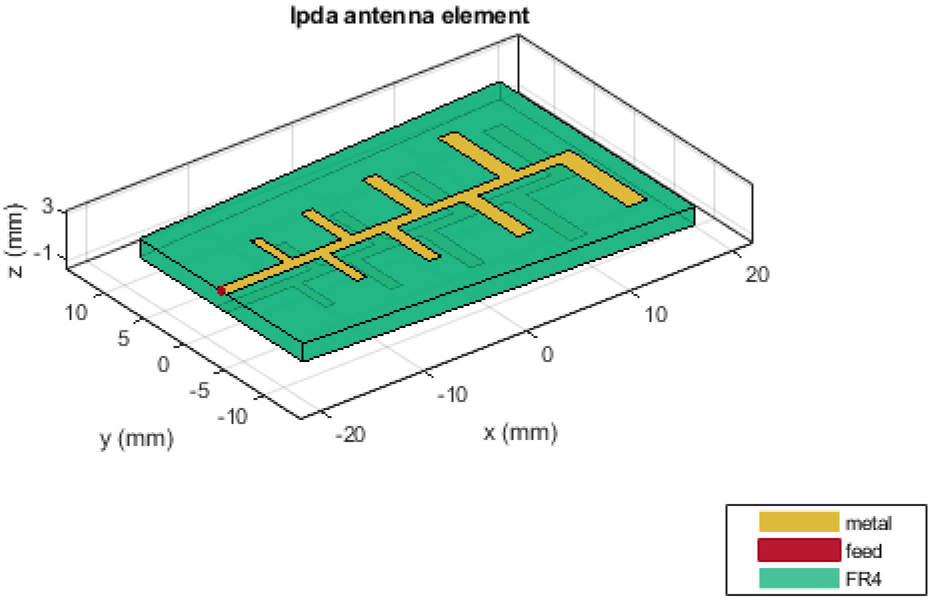
```
show(lpdipole)
```



Create and View Characteristics of Tapered LPDA

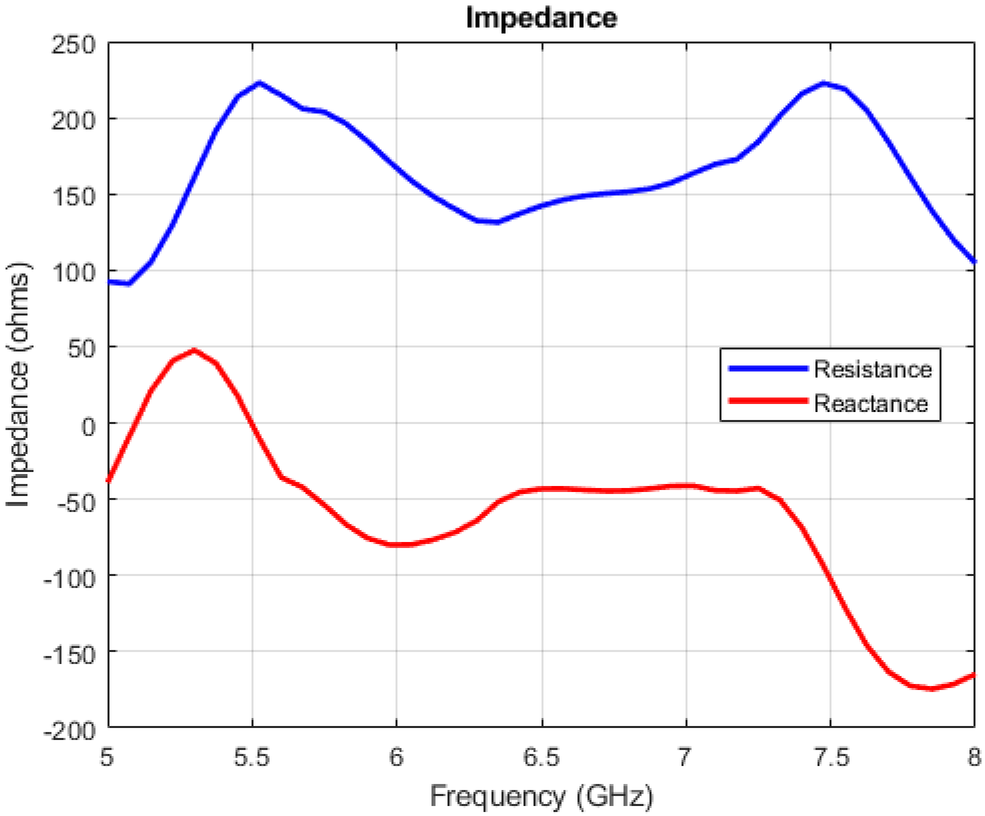
Create a tapered LPDA object and plot impedance over a frequency of 5 - 8GHz. This example also shows how to plot the 3-D radiation pattern of the antenna.

```
lpdipole = lpda('BoardWidth',[20.37e-3 24.37e-3]);
show(lpdipole)
```



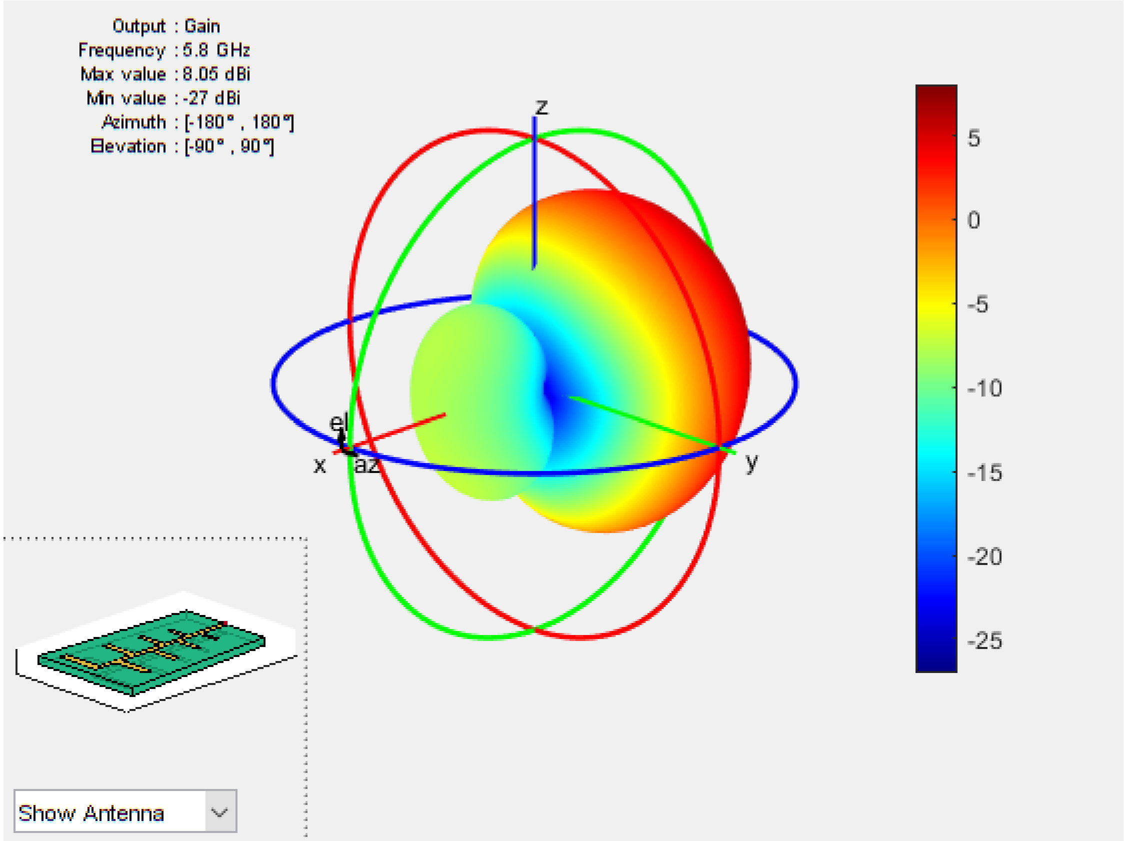
Plot Impedance over the specified frequency range.

```
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 : 8.05 dBi
 Min value : -27 dBi
  Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

See Also

`pcbStack` | `yagiUda`

Topics

“Rotate Antennas and Arrays”

Introduced in R2018a

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.

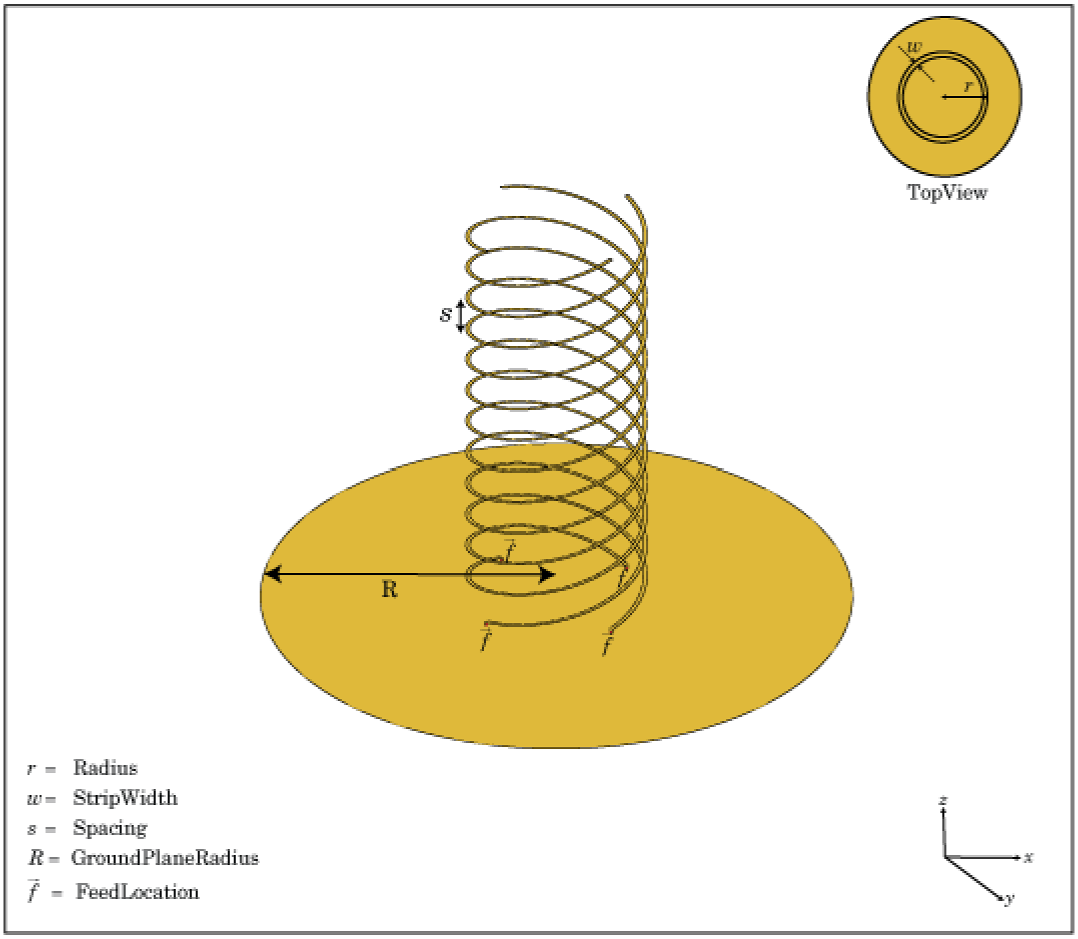
For a given cylinder radius, use the `cylinder2strip` utility function to calculate the equivalent width. The default helix antenna is end-fed. The circular ground plane is on the X-Y plane. Helix antennas are commonly used in axial mode. In this mode, the helix circumference is comparable to the operating 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 X-Y 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.

Output Arguments

ant — Multifilar helix antenna
`helixMultifilar` object

Multifilar helix antenna, returned as a `helixMultifilar` object.

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 X-Y 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`

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 origin. 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)`

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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

Quadrafilar Helix

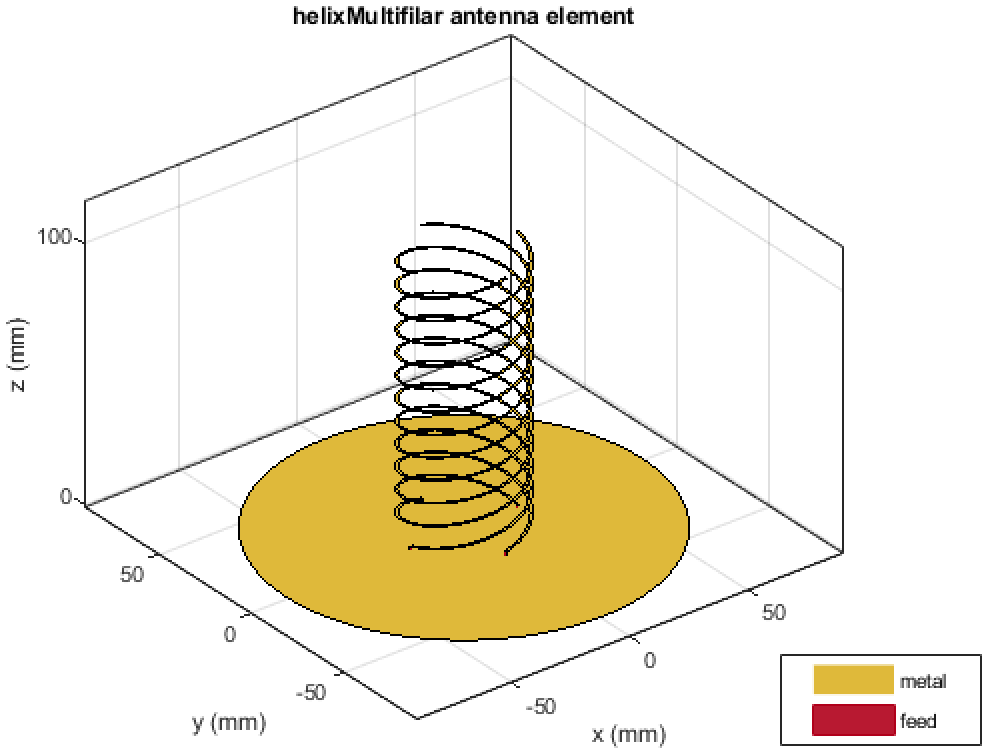
Create and view a Quadrafilar helix antenna.

```
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
                 Tilt: 0
             TiltAxis: [1 0 0]
```

```
                Load: [1x1 lumpedElement]
```

```
show(ant)
```



Bifilar Helix

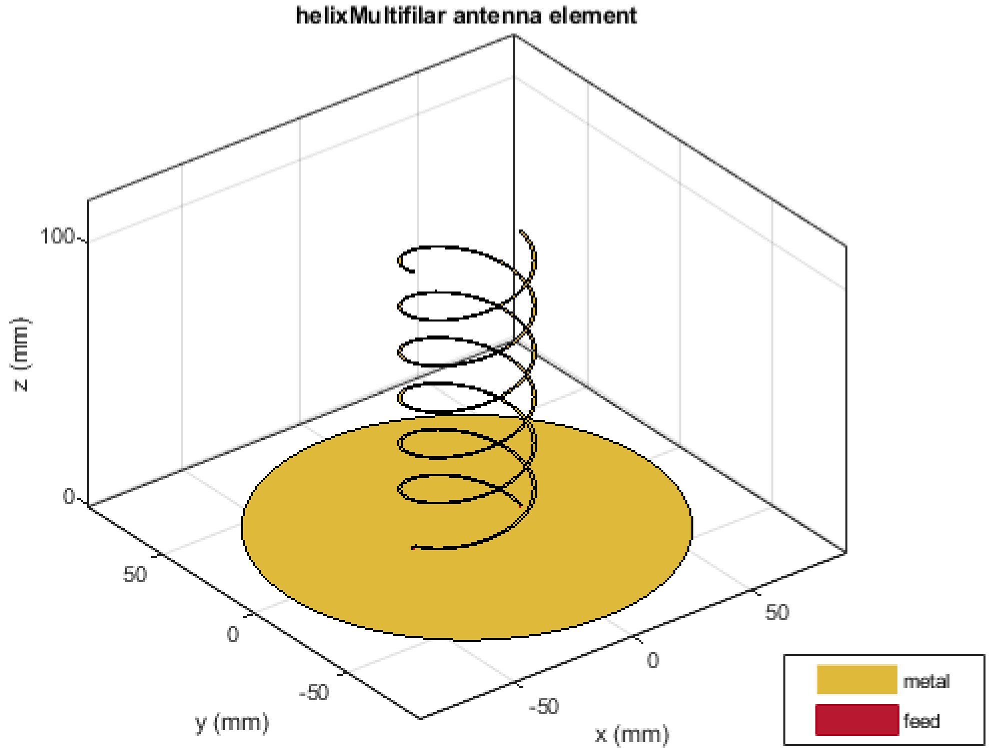
Create and view a bifilar helix antenna.

```
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
```

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

```
show(ant)
```



Radiation Pattern of Conical Multifilar Helix Antenna

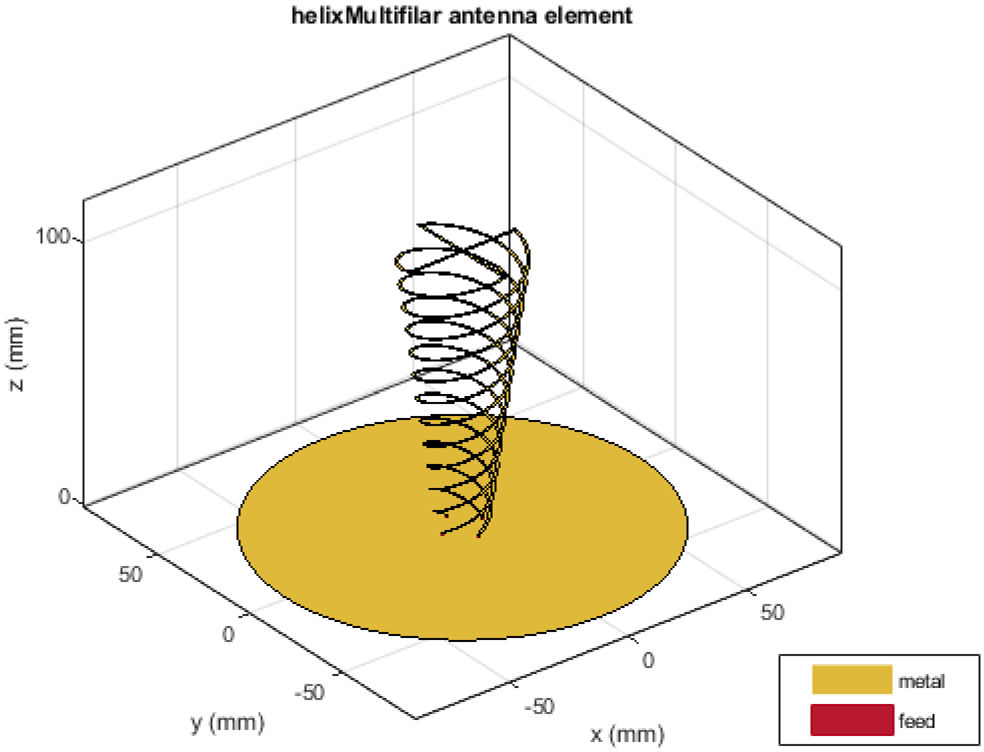
Create and view a conical multifilar helix antenna of radii, 0.0220 m and 0.00800 m respectively.

```
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
```

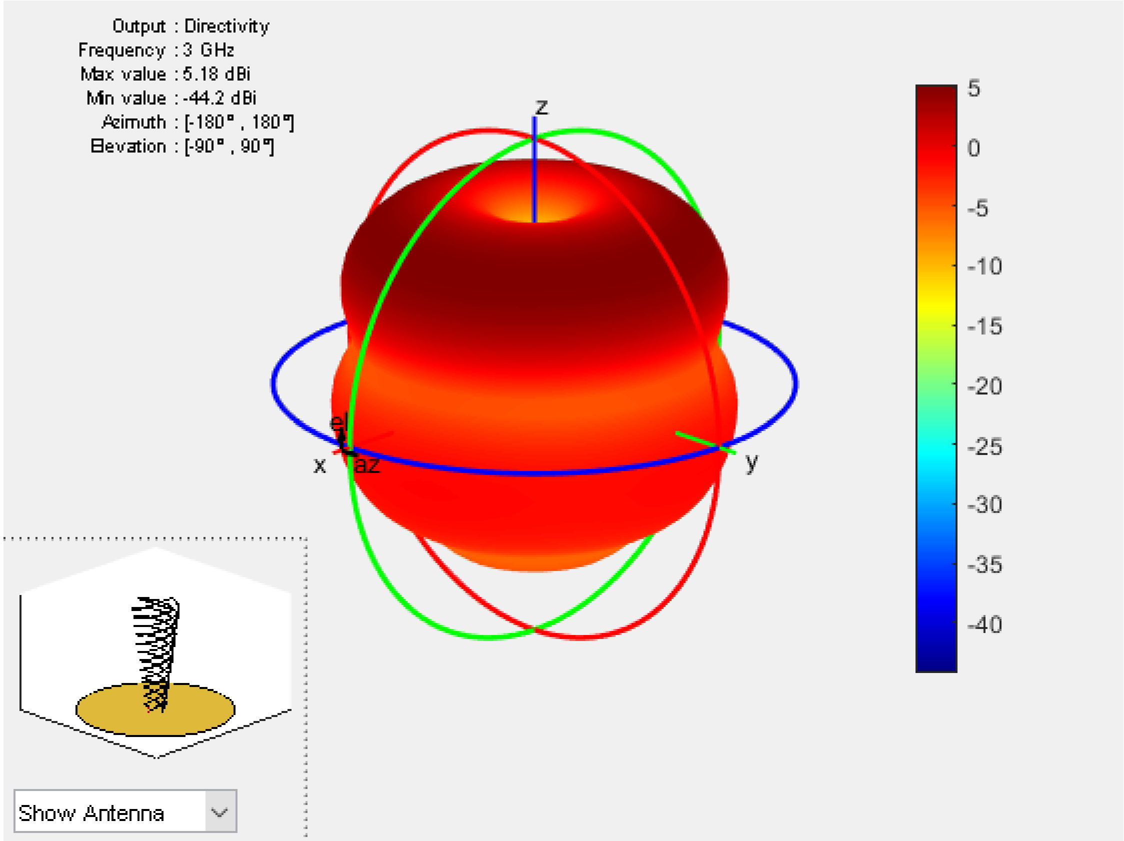
```
          FeedVoltage: 1
            FeedPhase: 0
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(ant)
```

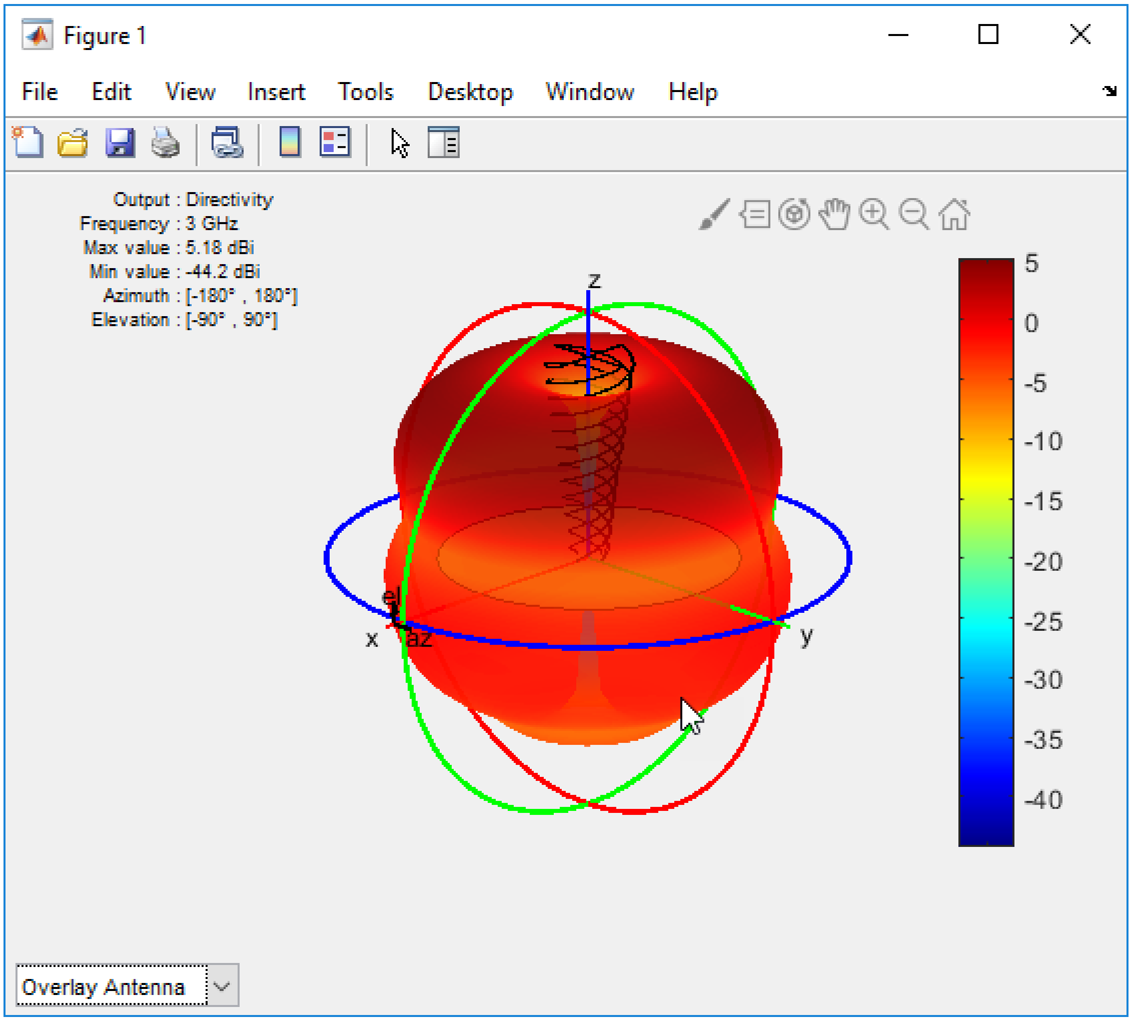


Plot the pattern of the antenna at 3 GHz.

```
pattern(ant,3e9)
```



Overlay the antenna on the pattern.



See Also

`cylinder2strip` | `dipoleHelix` | `dipolehelixMultifilar` | `helix` | `helixpitch2spacing`

Introduced in R2018b

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 X-Y 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

$x = r\cos(\theta)$

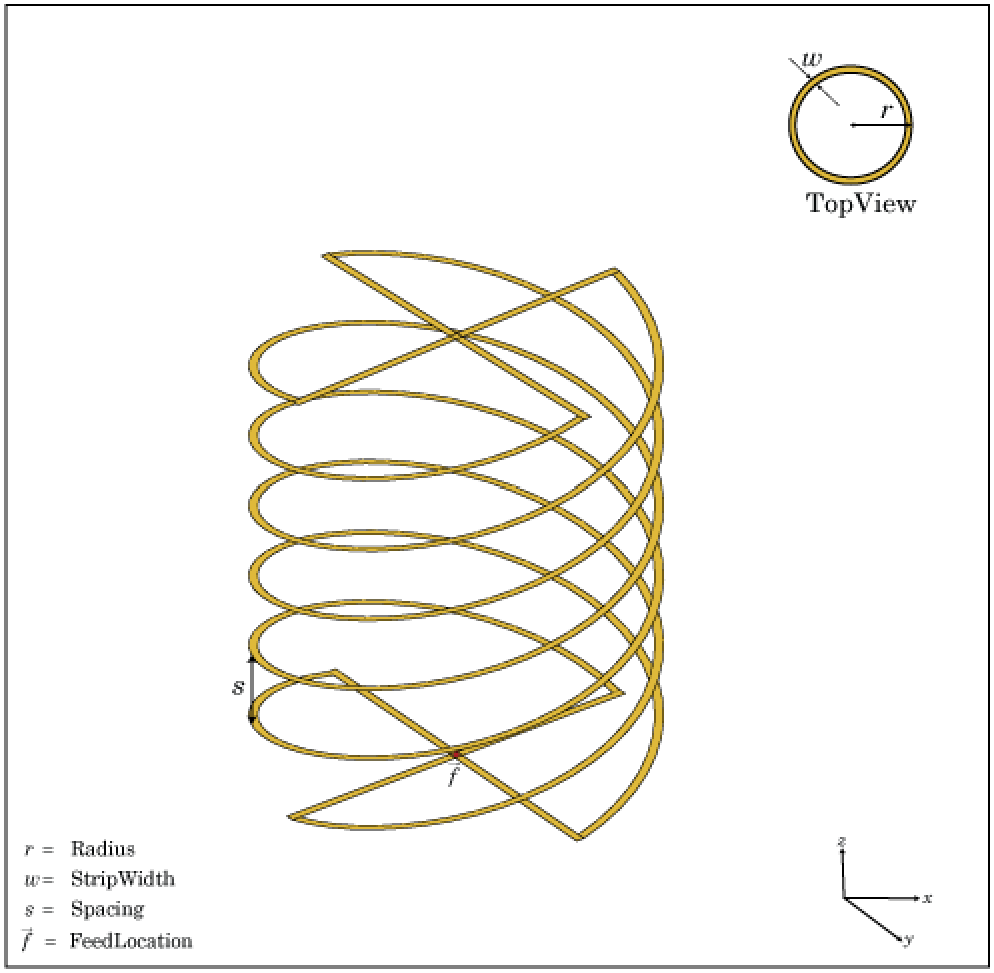
$y = r\sin(\theta)$

$z = S\theta$

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.

Output Arguments

`ant` — Dipole multifilar helix antenna
`dipoleHelixMultifilar` object

Dipole multifilar helix antenna, returned as a `dipoleHelixMultifilar` object.

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`

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 origin. 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)`

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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

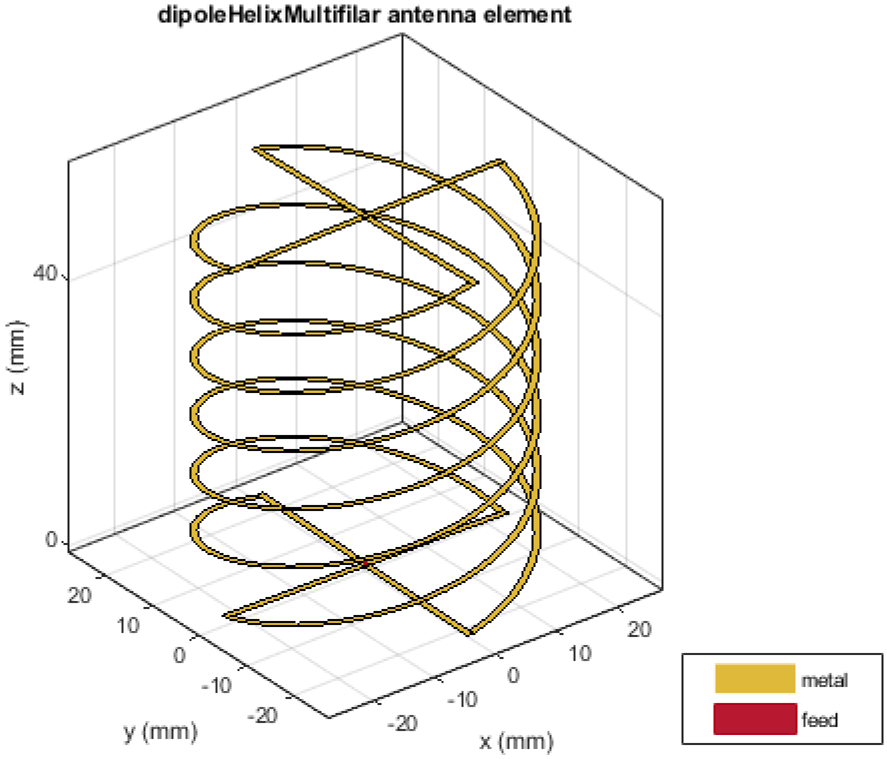
Default Multifilar Helical Dipole Antenna

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

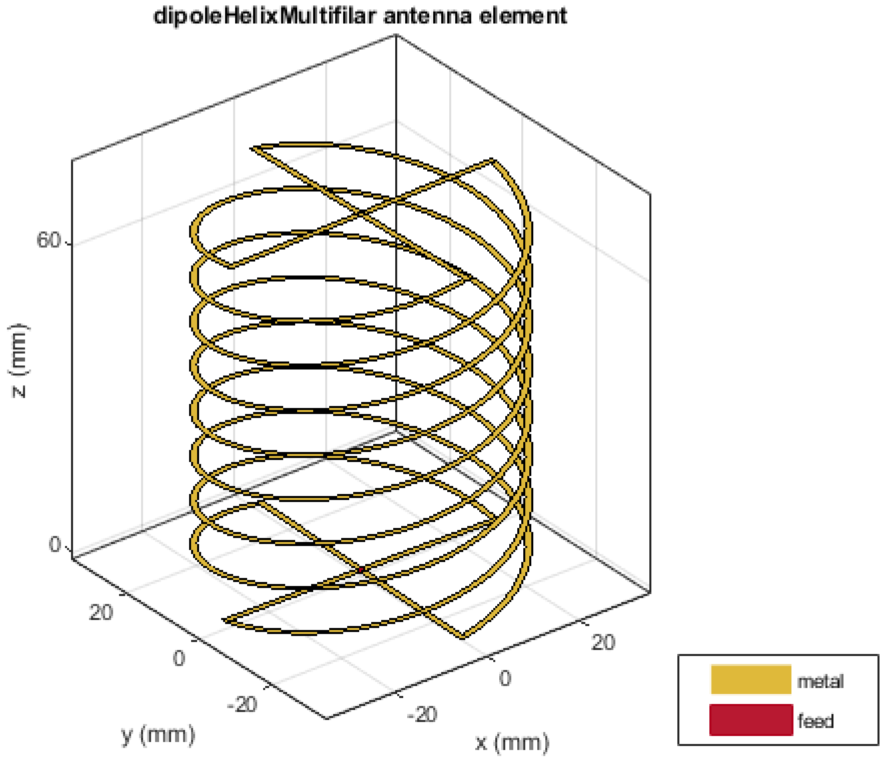
```
show(ant)
```



Quadrafilar Helical Dipole Antenna and Radiation Pattern

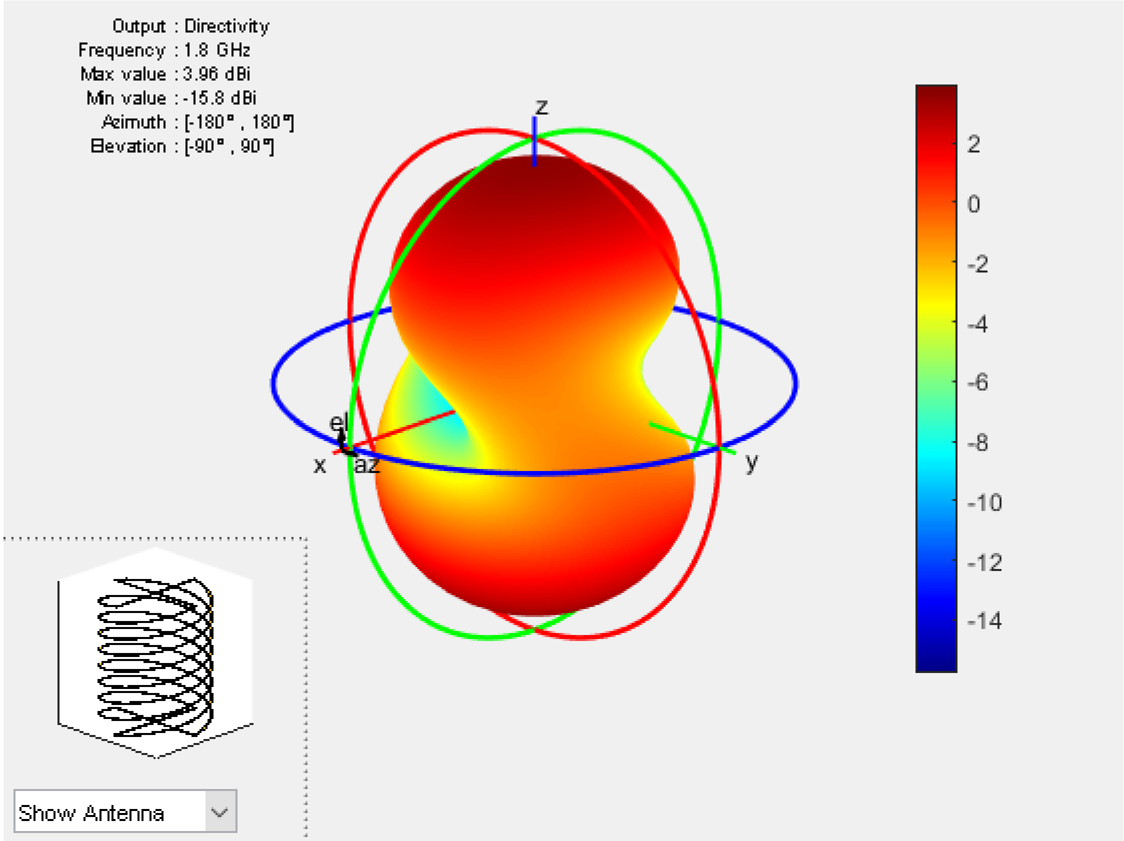
Create and view a quadrafilar helical dipole antenna with turn radius of 28 mm and strip width of 1.2 mm.

```
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°]
```

See Also

`dipoleHelix` | `helix` | `helixMultifilar`

Introduced in R2018b

fractalGasket

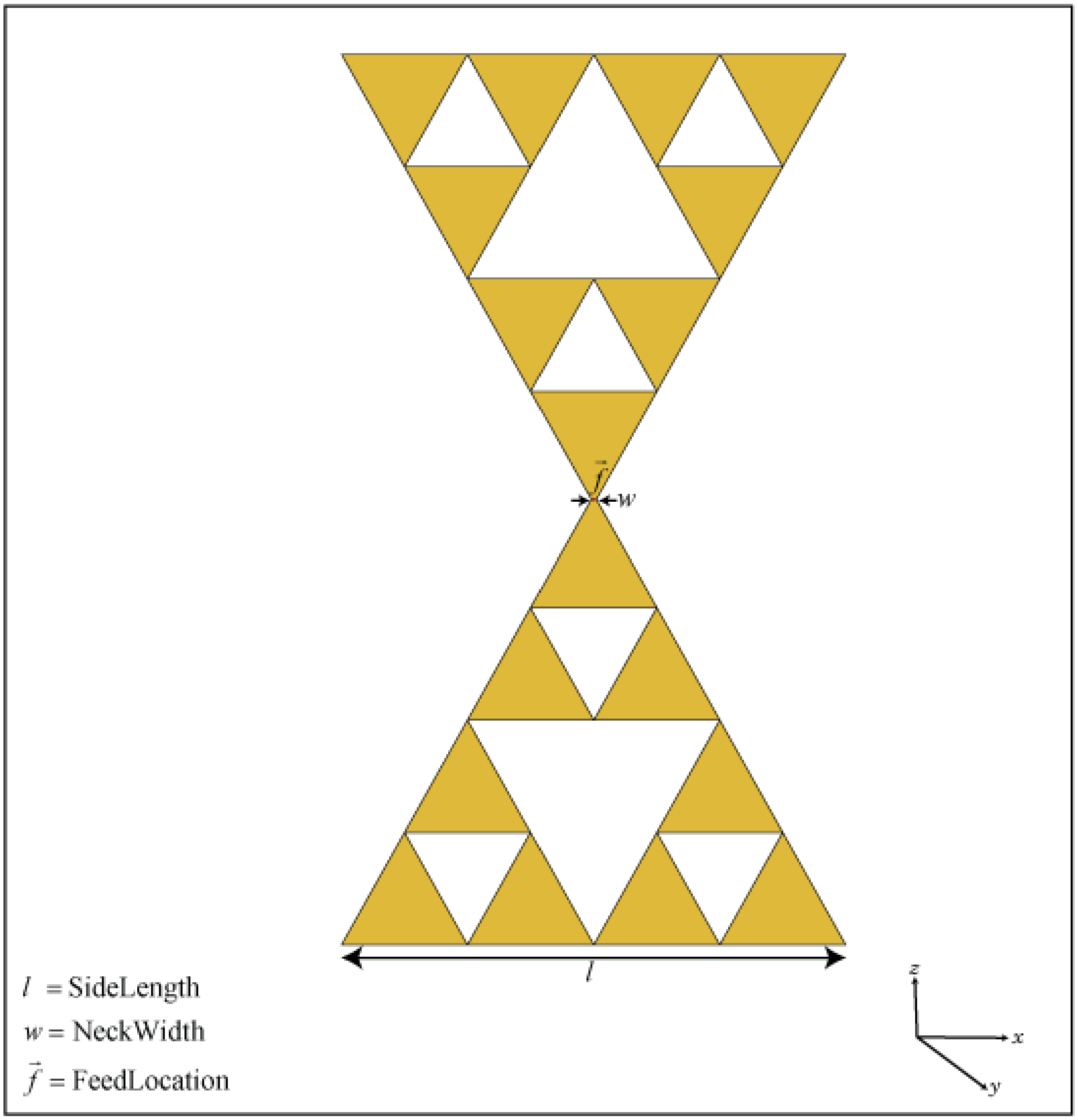
Create Sierpinski's Gasket fractal antenna on X-Y plane

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.

Output Arguments

`ant` — Sierpinski's Gasket fractal antenna
`fractalGasket` object

Sierpinski's Gasket fractal antenna, returned as a `fractalGasket` object.

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`

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 origin. 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface

current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

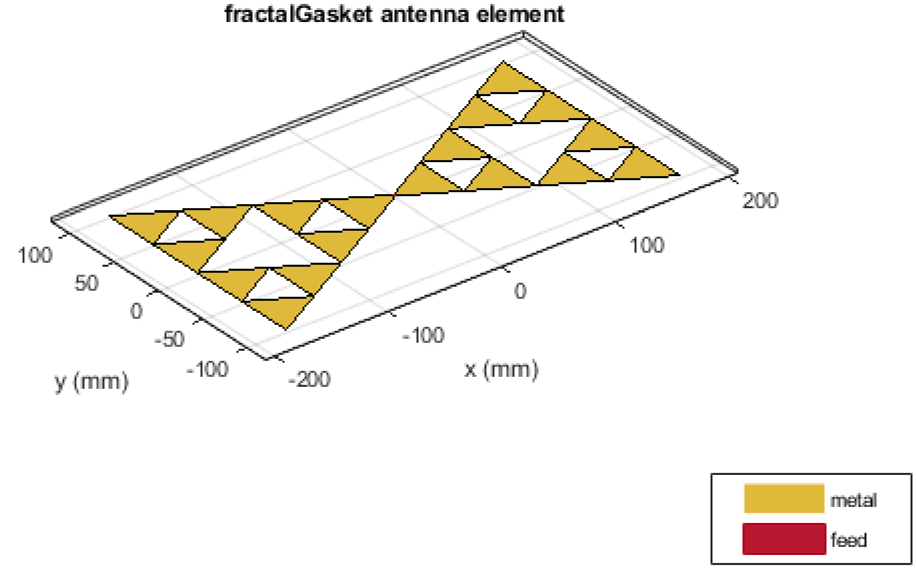
Default Sierpinski's Gasket

Create and view a default fractal Sierpinski's Gasket.

```
ant = fractalGasket
```

```
ant = 
  fractalGasket with properties:

    NumIterations: 2
             Side: 0.2000
        NeckWidth: 0.0020
             Tilt: 0
         TiltAxis: [1 0 0]
             Load: [1x1 lumpedElement]
```

```
show(ant)
```



See Also

`fractalCarpet` | `fractalIsland` | `fractalKoch`

Topics

“Rotate Antennas and Arrays”

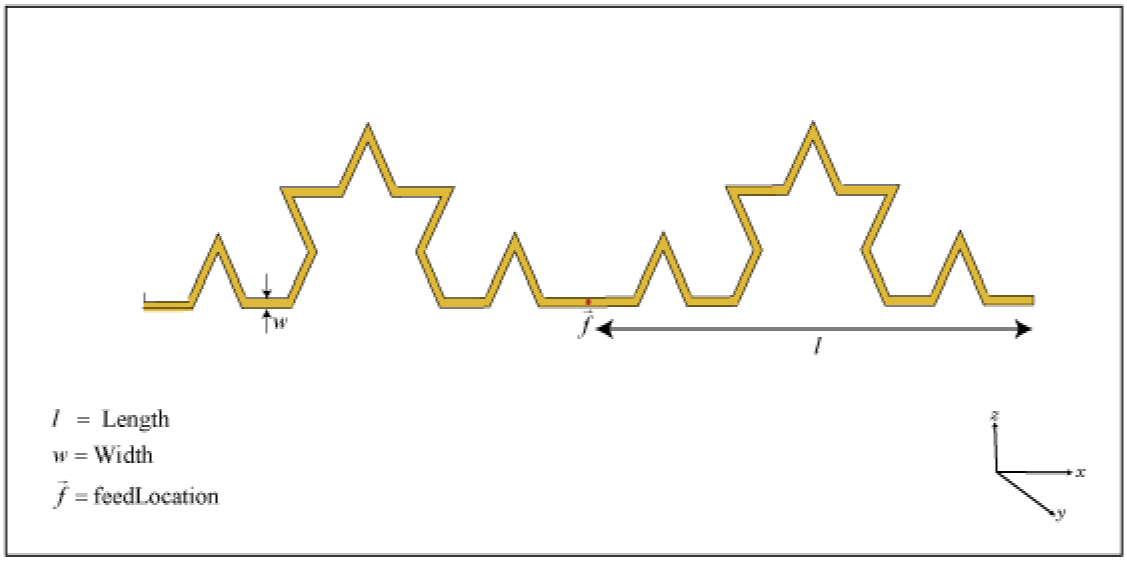
Introduced in R2018b

fractalKoch

Create Koch curve fractal dipole or loop antenna on X-Y plane

Description

The `fractalKoch` object creates a Koch curve fractal dipole or loop antenna on an X-Y 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.

Output Arguments

`ant` — Koch curve fractal antenna
`fractalKoch` object (default)

Koch curve fractal antenna, returned as a `fractalKoch` object.

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`

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 origin. 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface

current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

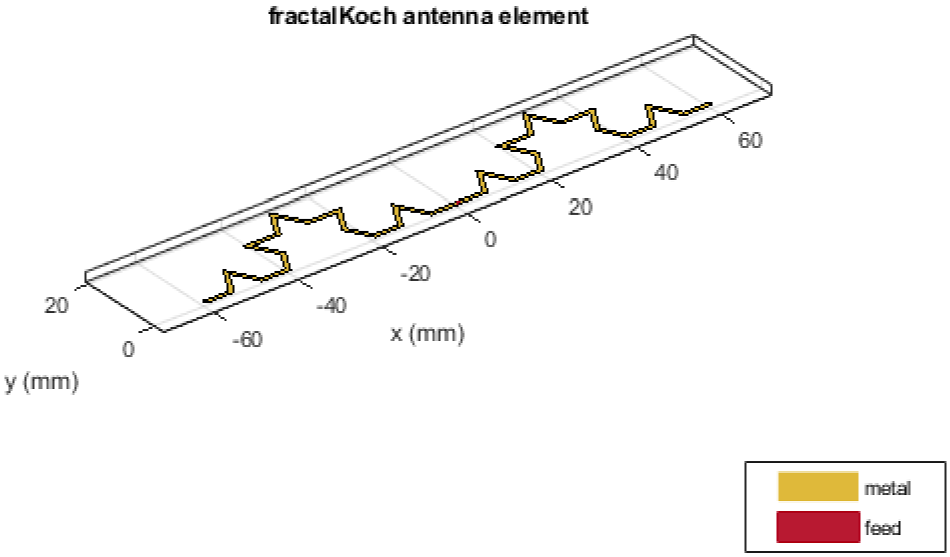
Default Koch Curve Fractal Antenna

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

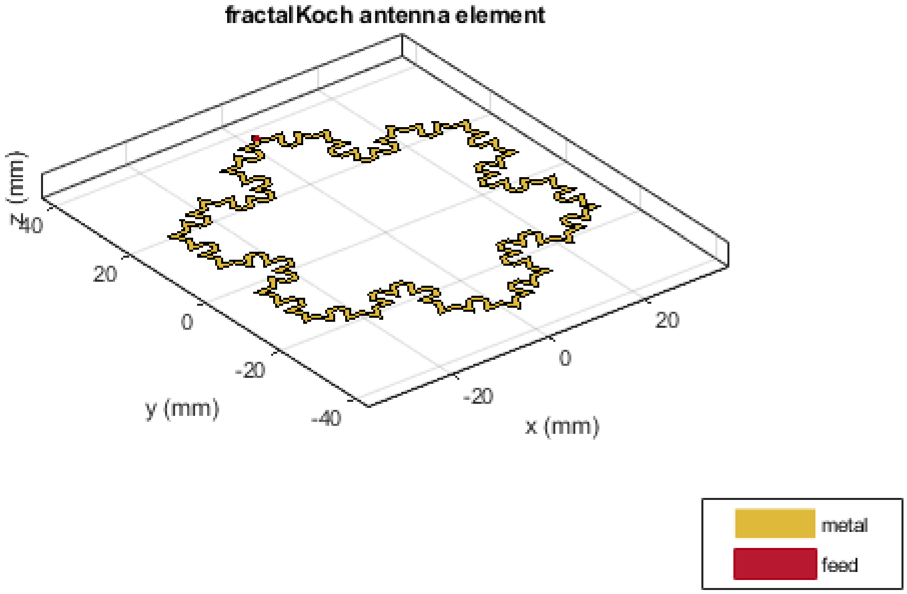
```
show(ant)
```



Koch Loop Fractal Antenna

Create and view a Koch loop fractal antenna with three iterations.

```
ant = fractalKoch('NumIterations',3,'Type','loop');
show(ant)
```



See Also

`fractalCarpet` | `fractalGasket` | `fractalIsland`

Topics

"Rotate Antennas and Arrays"

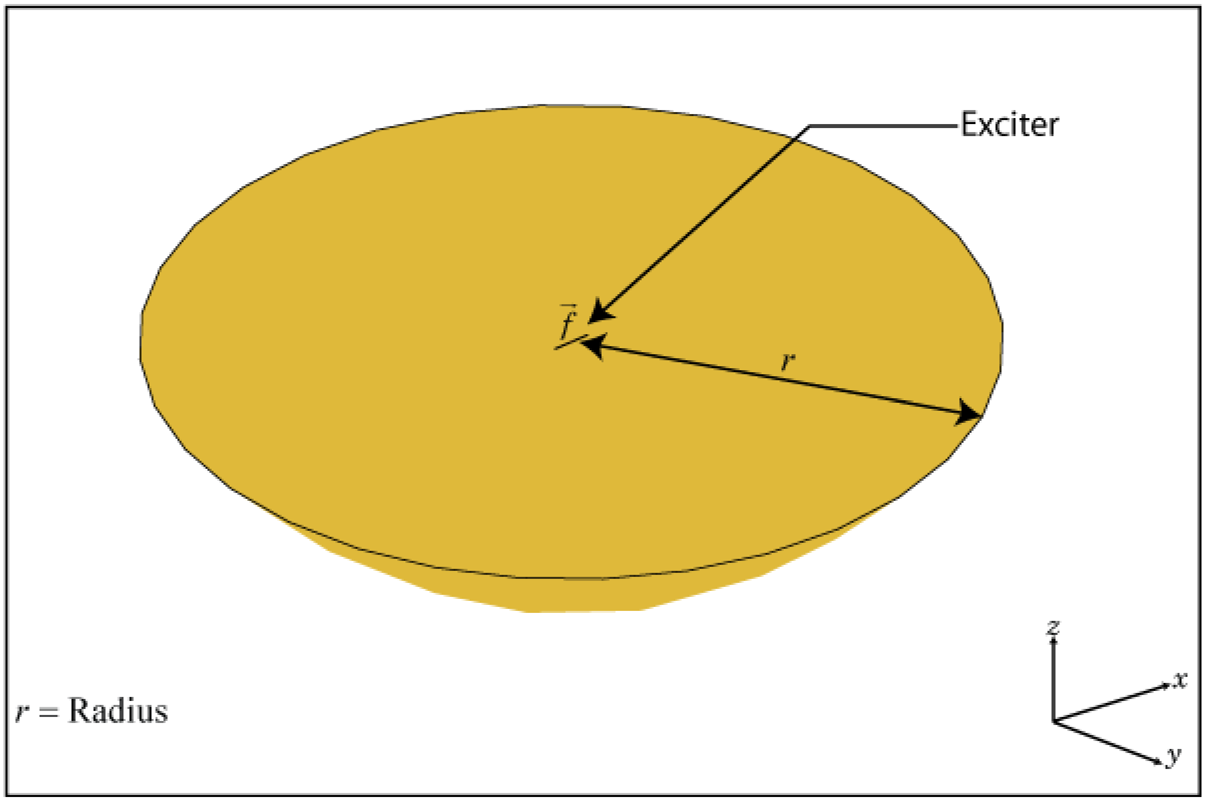
Introduced in R2018b

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.

Output Arguments

`ant` — Parabolic reflector antenna
`reflectorParabolic` object (default)

Parabolic reflector antenna, returned as a `reflectorParabolic` object.

Properties

`Exciter` — Antenna type used as exciter
`dipole` (default) | any single-element antenna 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 single elements in the Antenna Toolbox as an exciter.

Example: `'Exciter',horn`

Example: `ant.Exciter = horn`

`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`

`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: `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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface

design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

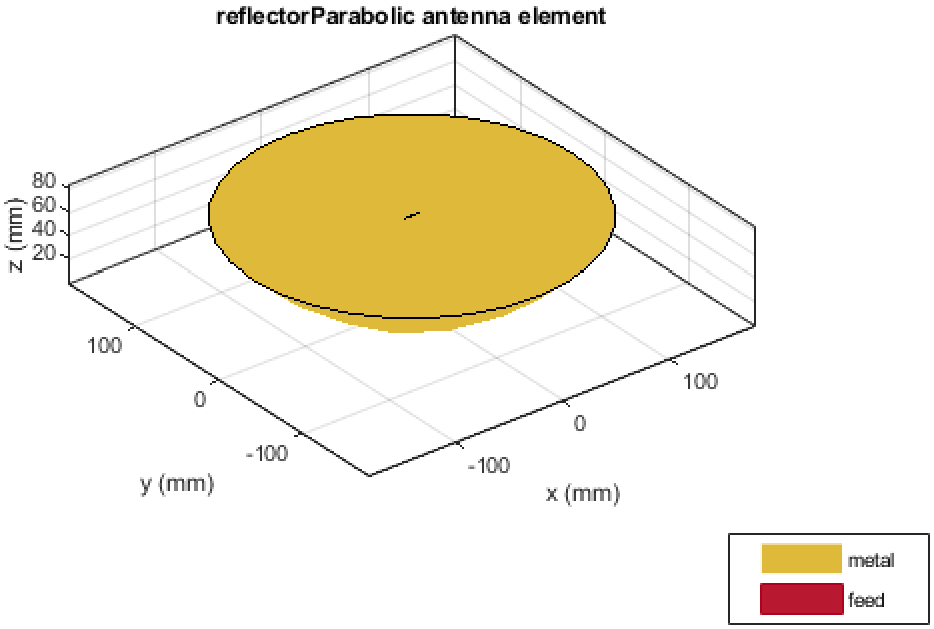
Default Parabolic Reflector and Radiation Pattern

Create and view a default parabolic reflector antenna.

```
ant = reflectorParabolic
```

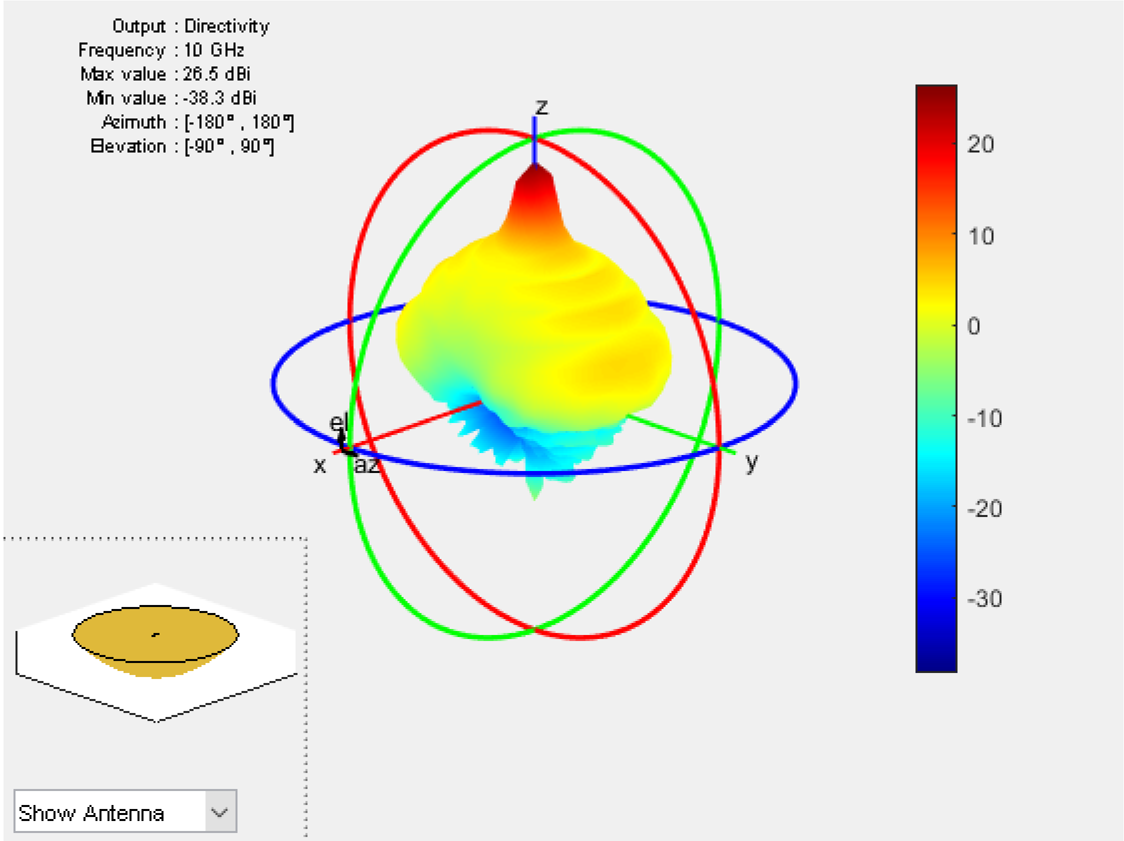
```
ant = 
  reflectorParabolic with properties:

        Exciter: [1x1 dipole]
         Radius: 0.1500
    FocalLength: 0.0750
     FeedOffset: [0 0 0]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the parabolic reflector at 10 GHz.

```
pattern(ant,10e9)
```



Output : Directivity
Frequency : 10 GHz
Max value : 26.5 dBi
Min value : -38.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

See Also

Topics
"Rotate Antennas and Arrays"

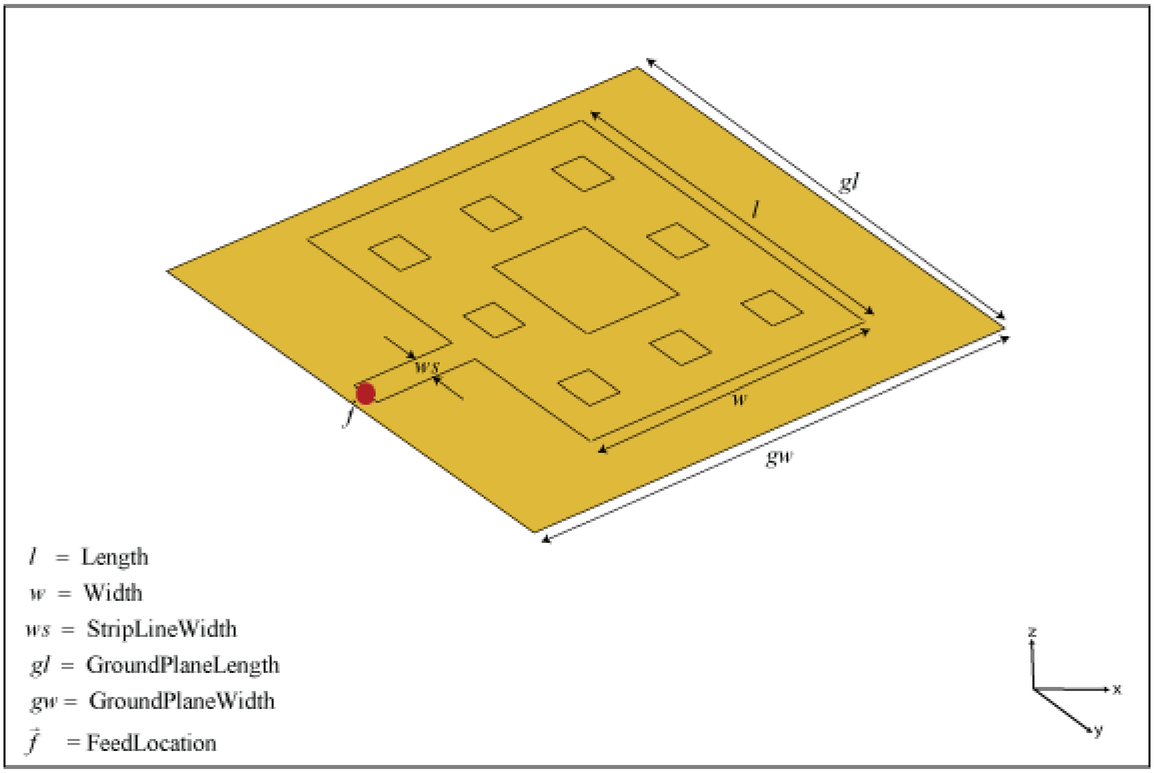
Introduced in R2018b

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.

Output Arguments

`ant` — Sierpinski's carpet fractal antenna
`fractalCarpet` object

Sierpinski's carpet fractal antenna, returned as a `fractalCarpet` object.

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`

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 origin. 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

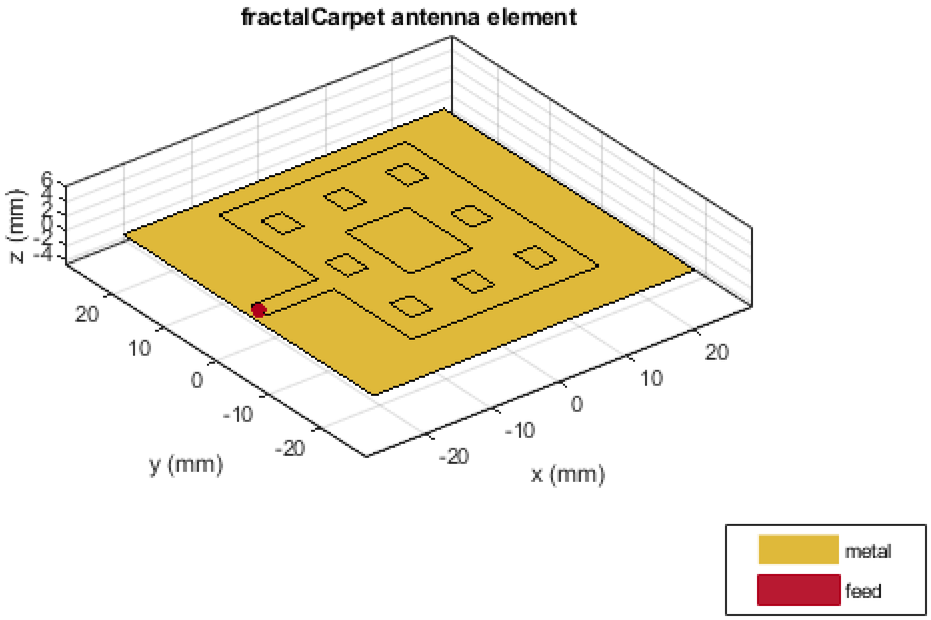
Default Sierpinski's Carpet Antenna

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

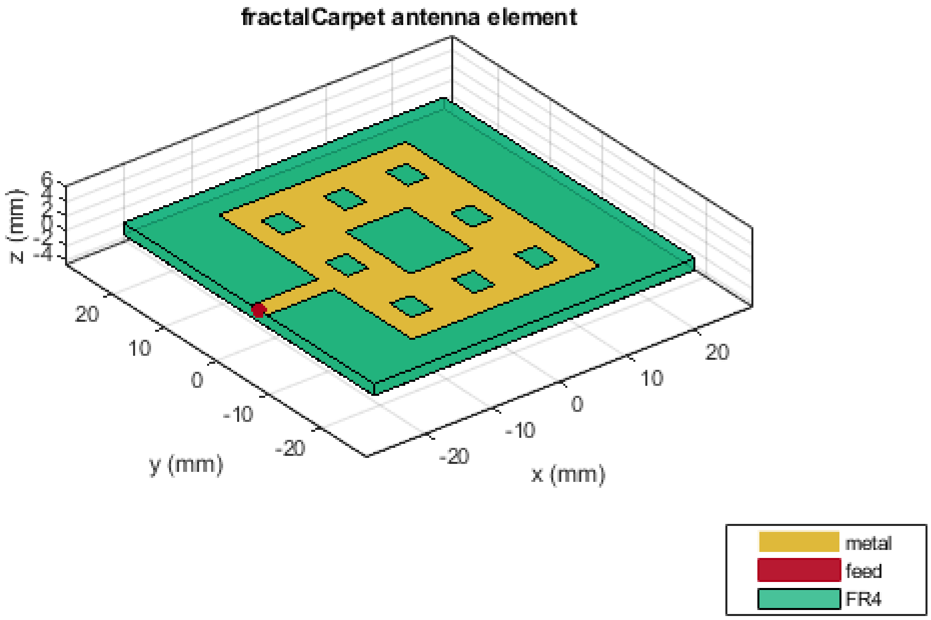
```
show(ant)
```



Radiation Pattern of Sierpinski's Carpet Antenna on FR4 Substrate

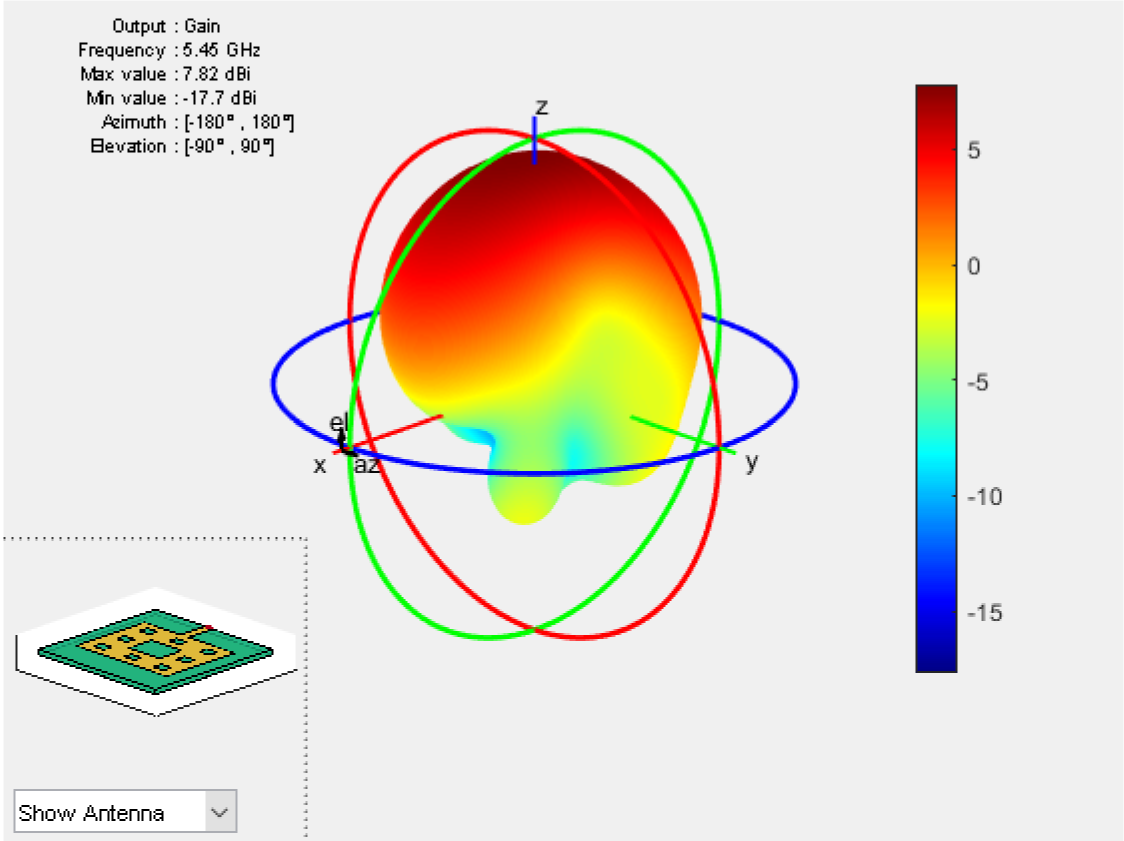
Create and view a Sierpinski's carpet fractal antenna on FR4 substrate.

```
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 : 7.82 dBi
Min value : -17.7 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

See Also

`fractalGasket` | `fractalIsland` | `fractalKoch`

Topics

"Rotate Antennas and Arrays"

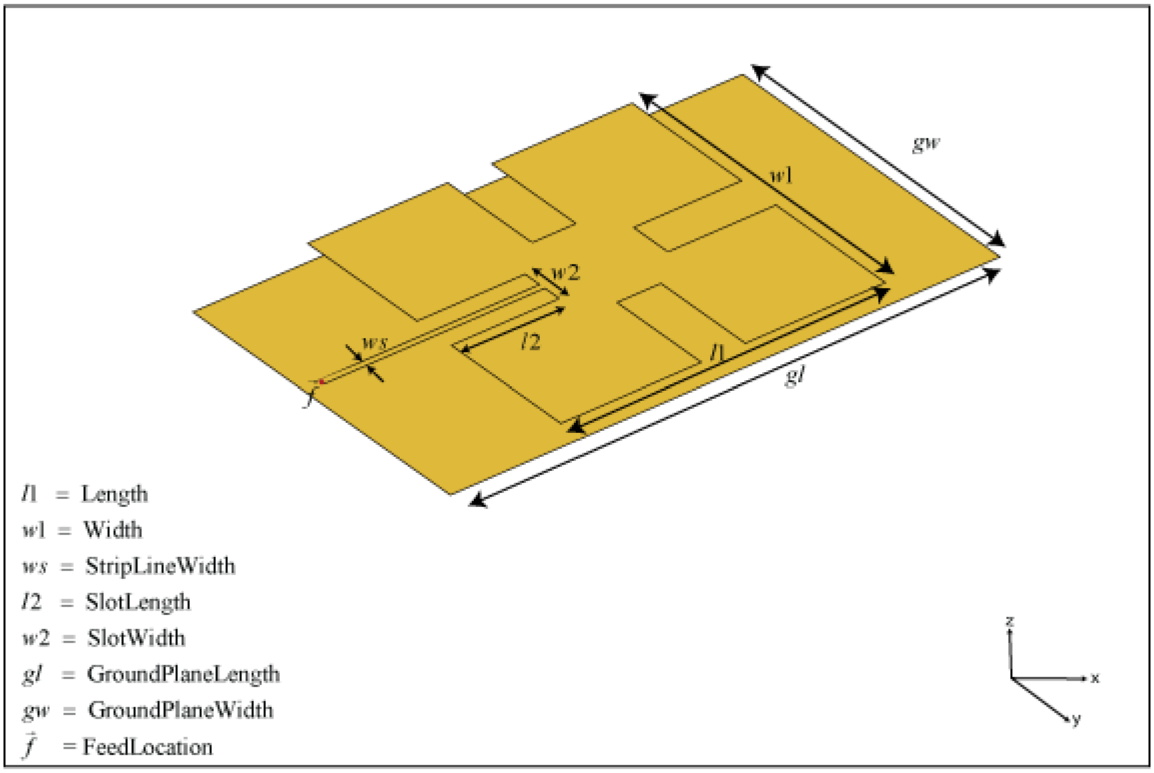
Introduced in R2019a

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.

Output Arguments

`ant` — Minkowski's loop fractal antenna
`fractalIsland` object

Minkowski's loop fractal antenna, returned as a `fractalIsland` object.

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

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 origin. 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: 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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

Default Minkowski's Loop Fractal Antenna

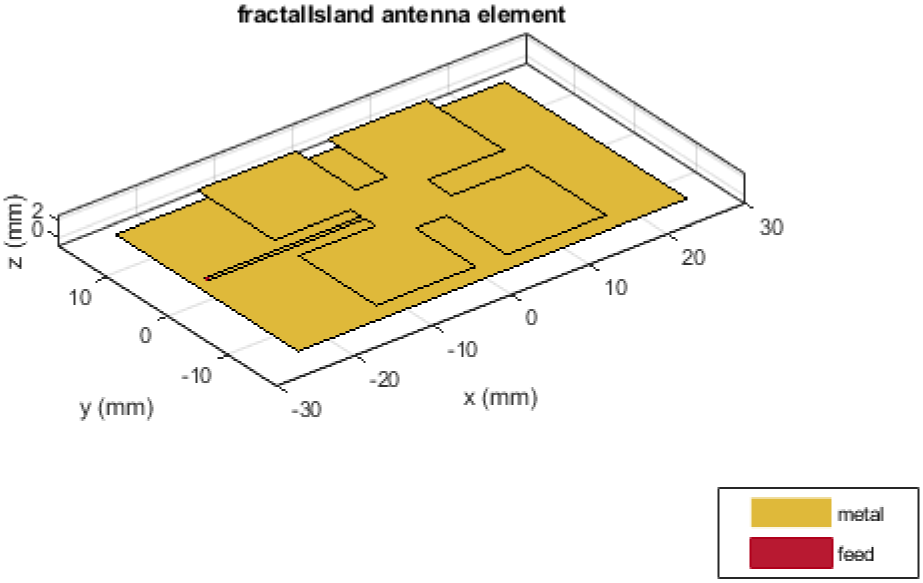
Create and view a Minkowski's loop fractal antenna with default property values.

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

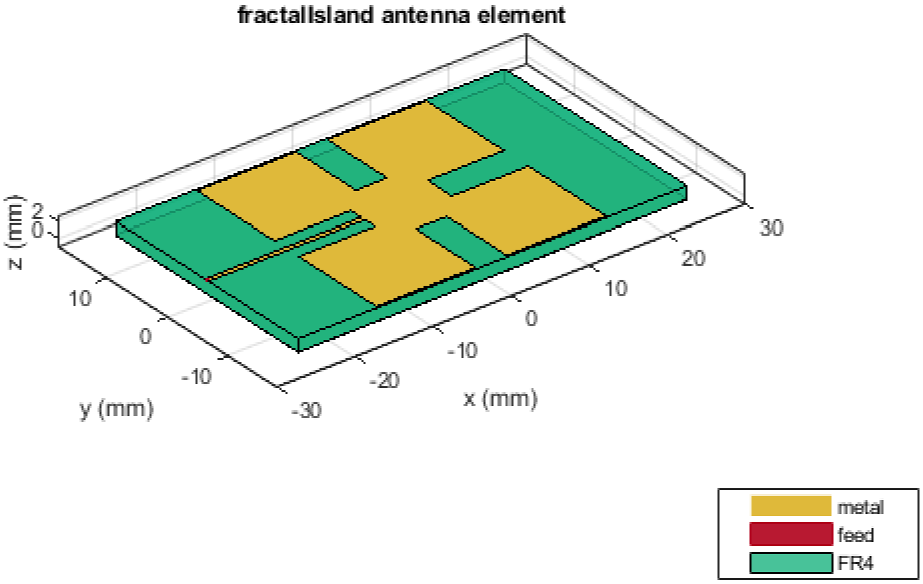
```
show(ant)
```



Radiation Pattern of Minkowski's Loop Fractal Antenna on FR4 Substrate

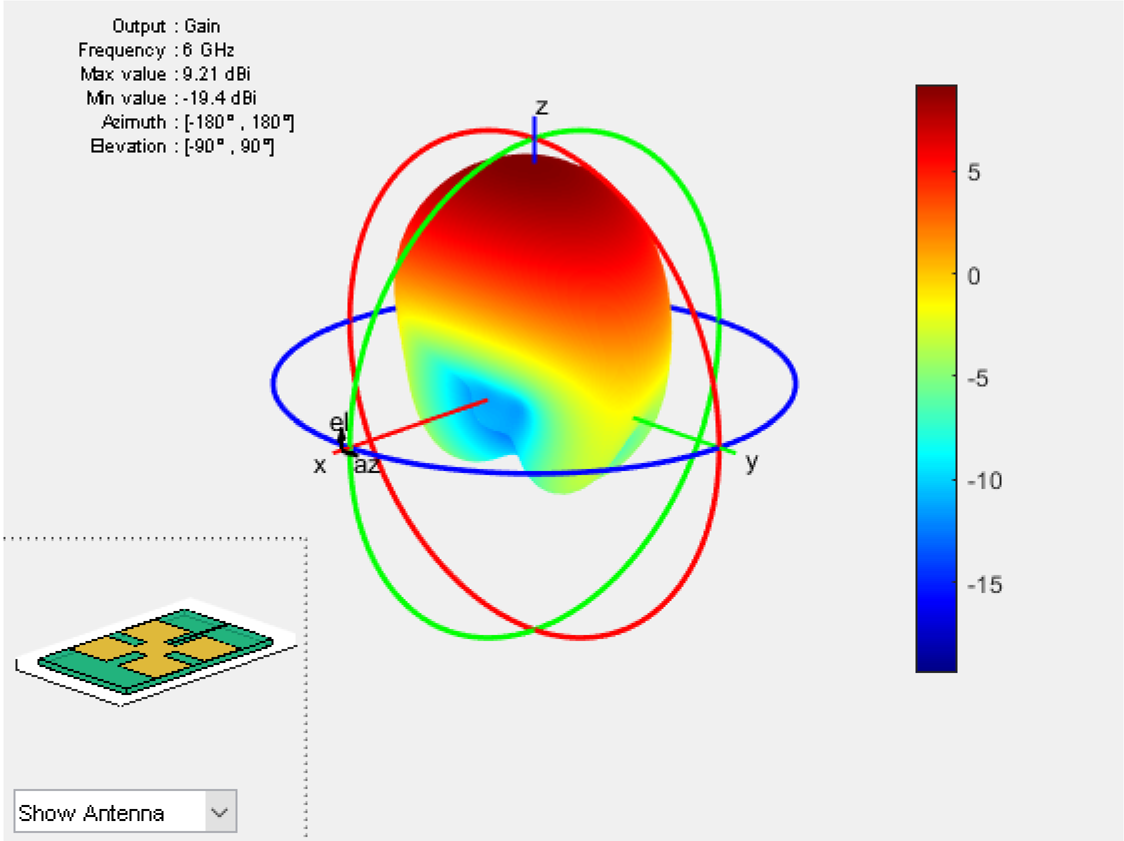
Create and view a Minkowski's loop fractal antenna on FR4 substrate.

```
ant = fractalIsland('Substrate',dielectric('FR4'));
show(ant)
```



Plot the radiation pattern of the antenna at 6 GHz.

```
pattern(ant,6e9)
```



```
   Output : Gain
Frequency : 6 GHz
Max value : 9.21 dBi
Min value : -19.4 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

See Also

`fractalCarpet` | `fractalGasket` | `fractalKoch`

Topics

"Rotate Antennas and Arrays"

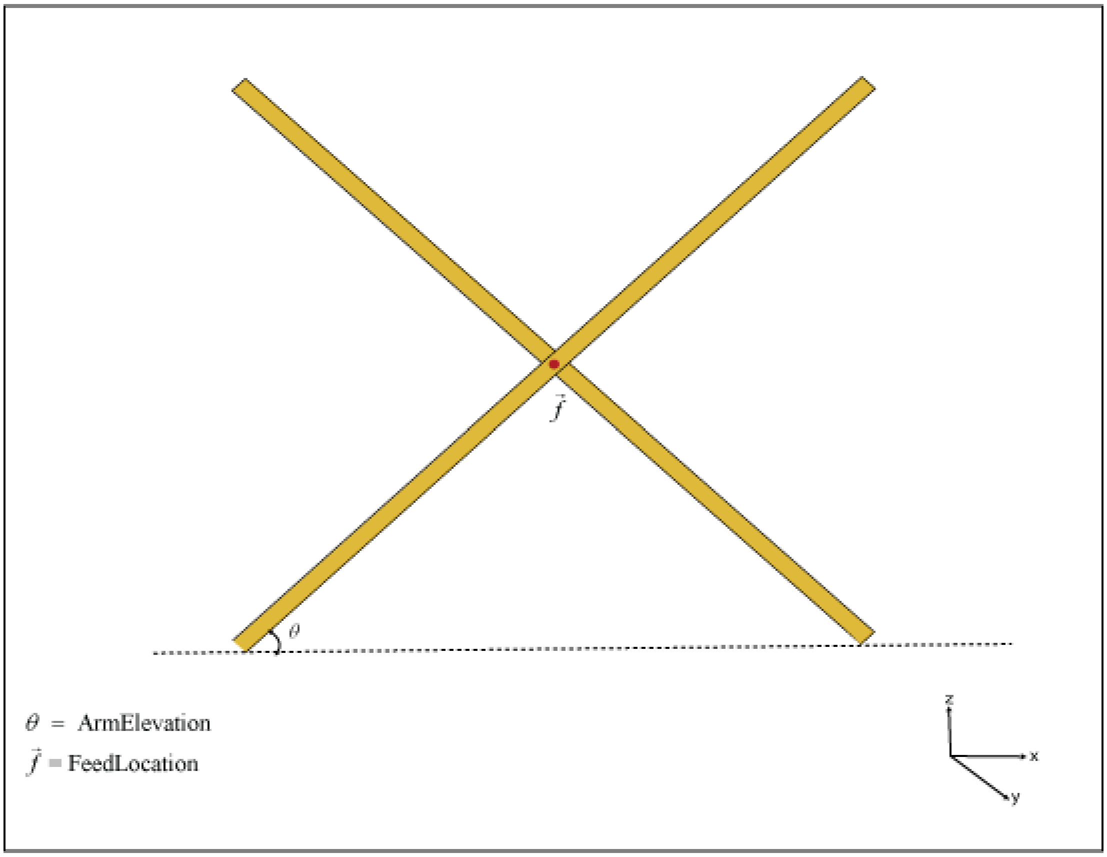
Introduced in R2019a

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.

Output Arguments

`ant` — Turnstile antenna
`dipoleCrossed` object

Turnstile antenna, returned as a `dipoleCrossed` object.

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 in degrees.

Example: `'Tilt',90`

Example: `ant.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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna

charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

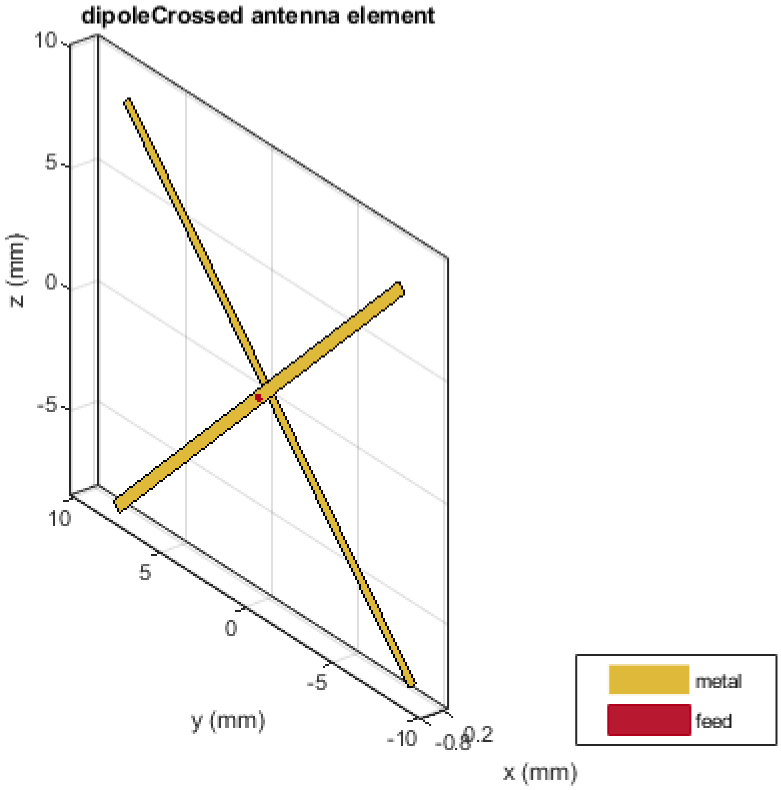
Default Crossed Dipole Antenna

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)
```



See Also

`bowtieRounded` | `bowtieTriangular` | `dipoleBlade`

Topics

"Rotate Antennas and Arrays"

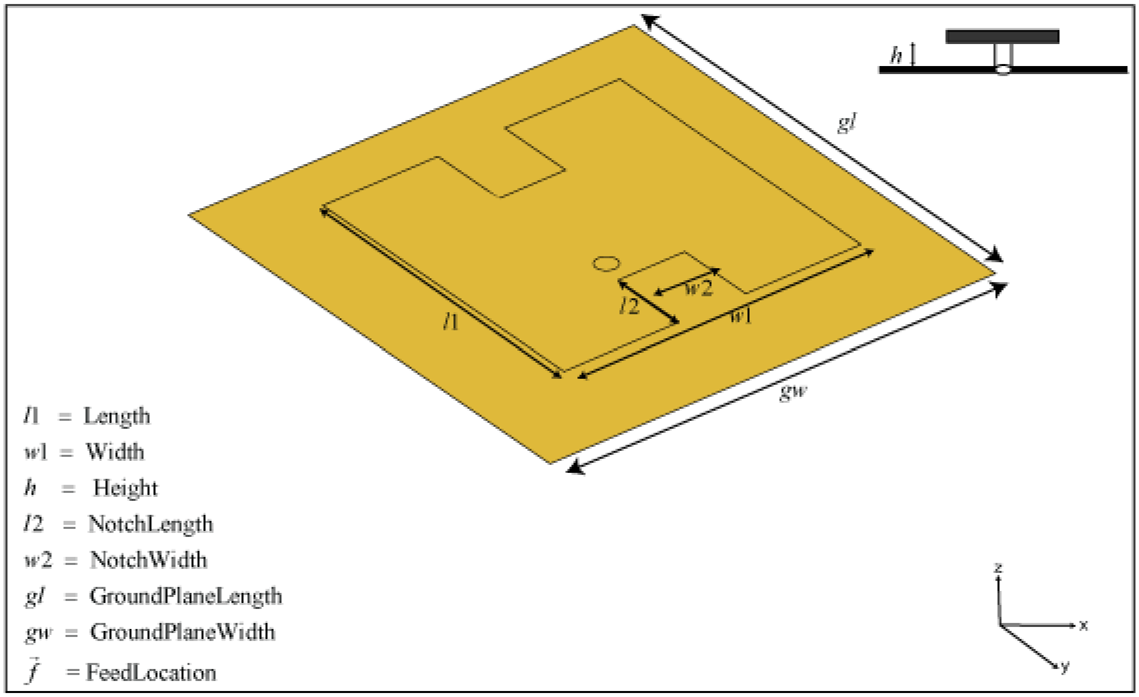
Introduced in R2019a

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.

Output Arguments

`ant` — H-shaped microstrip patch antenna
`patchMicrostripHnotch` object

H-shaped microstrip patch antenna, returned as a `patchMicrostripHnotch` object.

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`

`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 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: `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 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 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 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'`

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

Object Functions

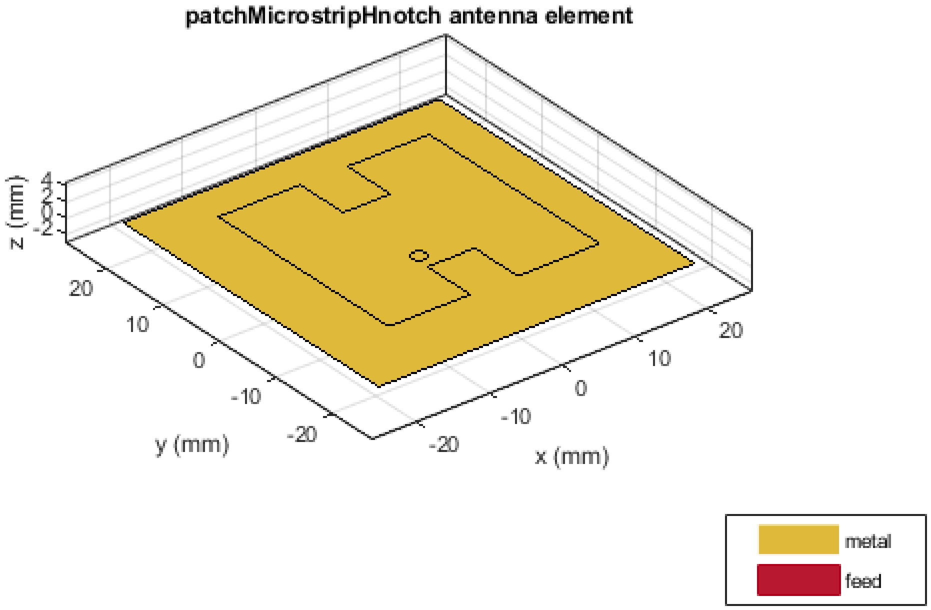
show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
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.

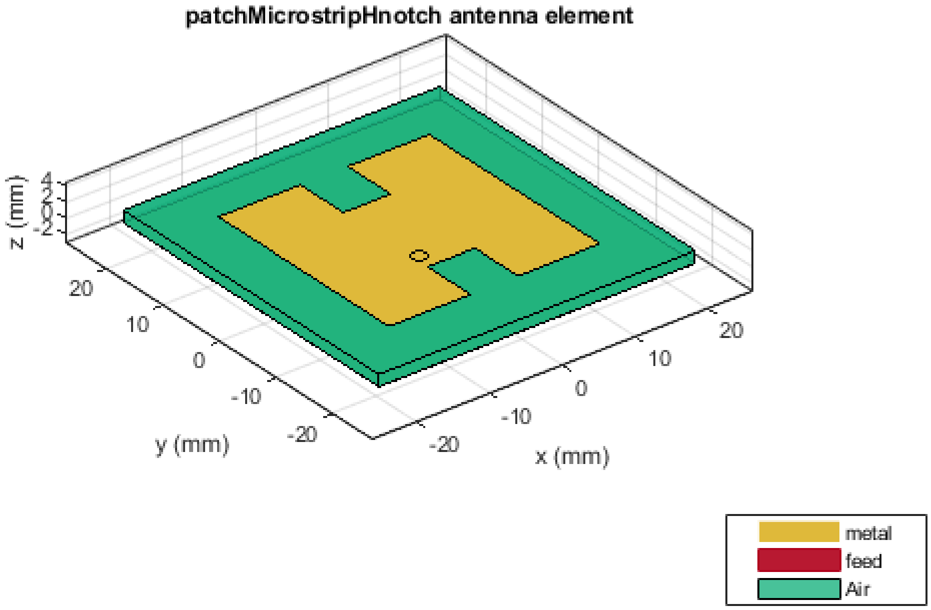
```
ant = patchMicrostripHnotch;
show(ant)
```



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);
```



See Also

`patchMicrostrip` | `patchMicrostripCircular` | `patchMicrostripTriangular`

Topics

"Rotate Antennas and Arrays"

Introduced in R2019a

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.

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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna

charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

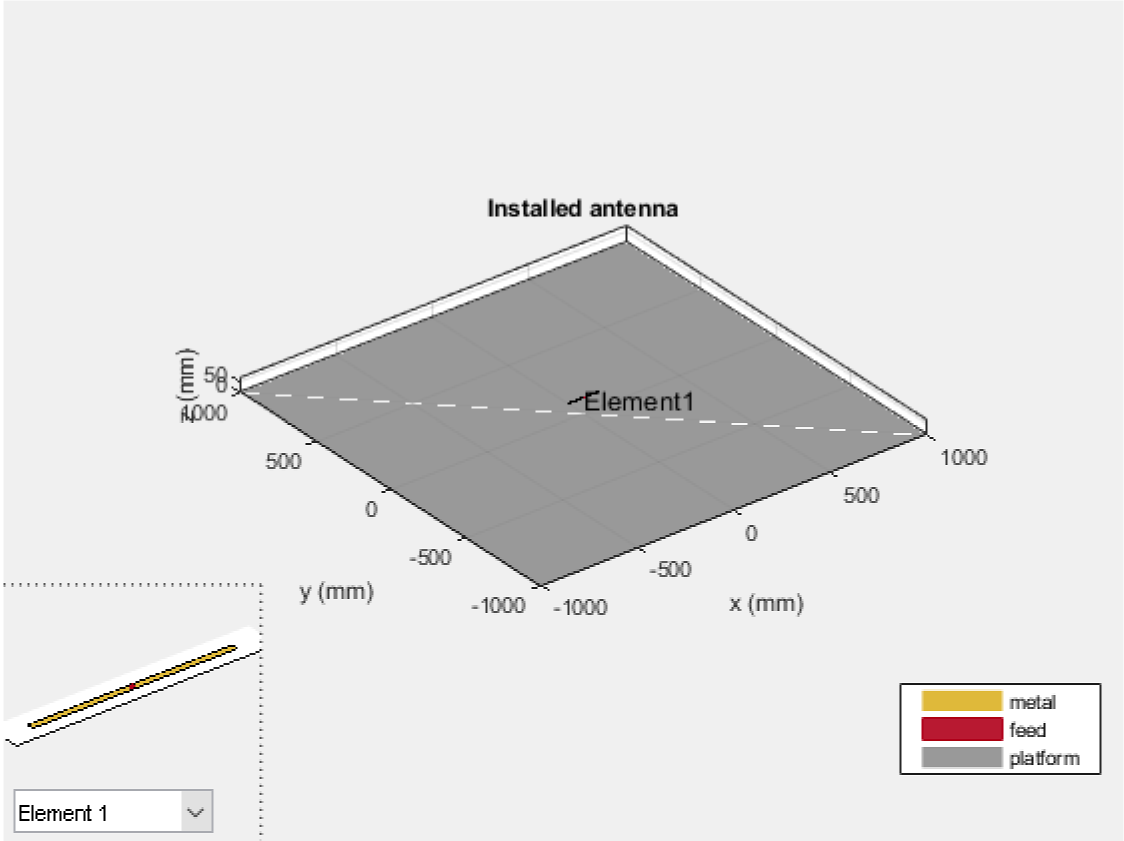
Installed Antenna Setup and Analysis

Create a default installed antenna.

```
ant = installedAntenna
```

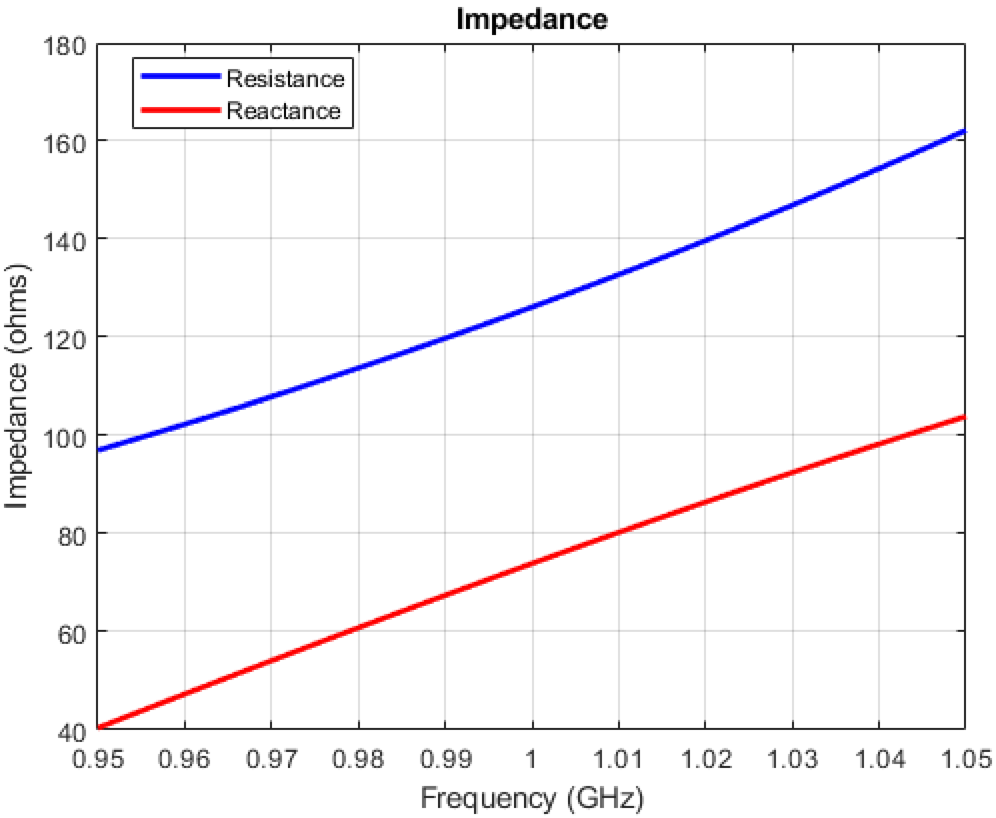
```
ant = 
  installedAntenna with properties:

           Platform: [1x1 platform]
            Element: [1x1 dipole]
    ElementPosition: [0 0 0.0750]
          Reference: 'feed'
        FeedVoltage: 1
          FeedPhase: 0
               Tilt: 0
           TiltAxis: [1 0 0]
```

```
show(ant);
```



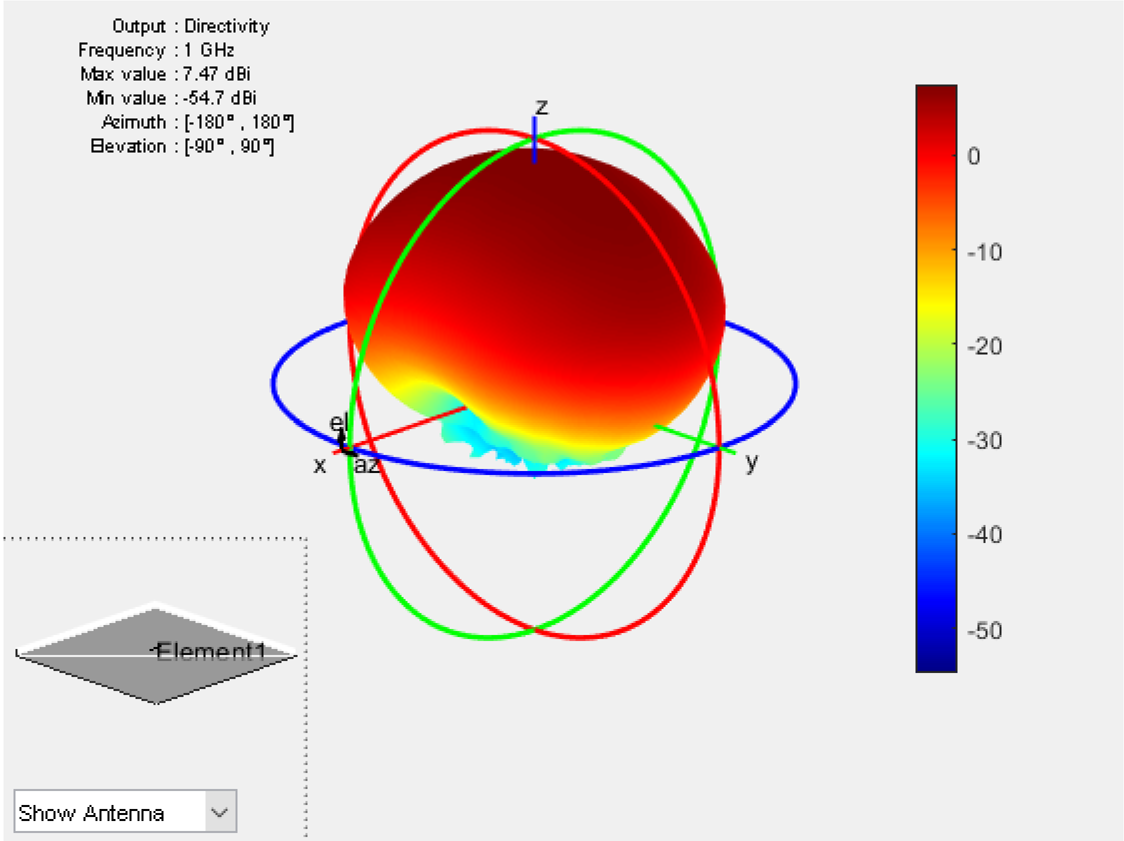
Calculate the impedance of the antenna.

```
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°]
```

See Also

`platform`

Topics

"Rotate Antennas and Arrays"

Introduced in R2019a

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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
mesh	Mesh properties of metal or dielectric antenna or array structure
meshconfig	Change mesh mode of antenna structure
stlwrite	Write mesh to STL file

Examples

Platform from STL of Waveguide Antenna

Create a waveguide antenna for operation at 8 GHz and compute the impedance.

```
w = design(waveguide,8e9);
Z = impedance(w,8e9);
```

Create an STL file for the above antenna.

```
stlwrite(w,'waveguide_8GHz.stl')
```

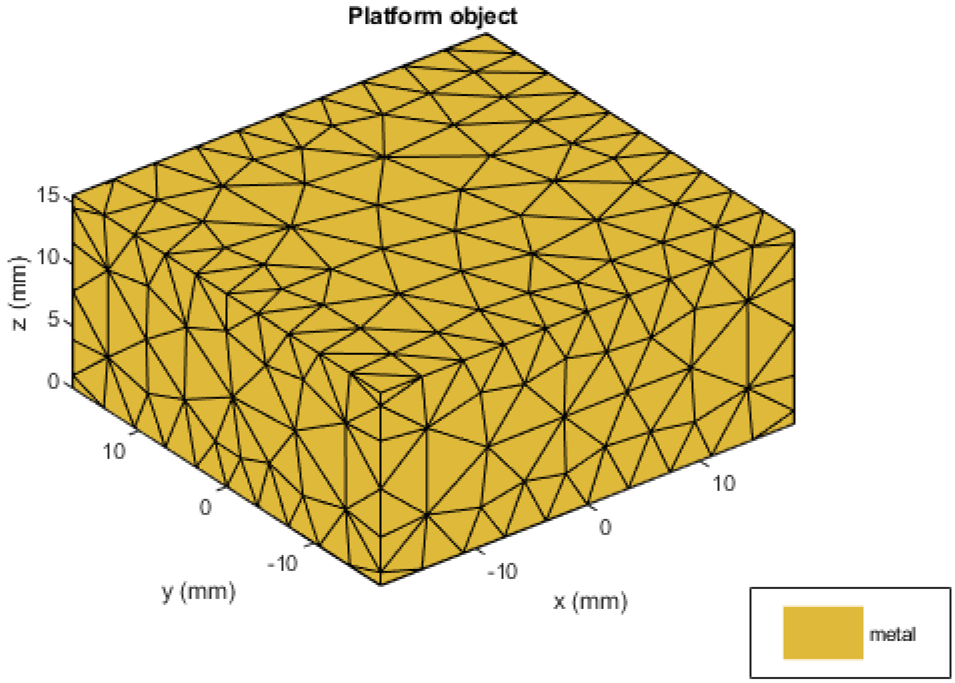
You will see the `waveguide_8GHz.stl` file in your current folder.

Load `waveduide_8GHz.stl` and visualize the platform.

```
plat = platform('FileName','waveguide_8GHz.stl','Units','m')
```

```
plat = 
  platform with properties:

         FileName: 'waveguide_8GHz.stl'
            Units: 'm'
    UseFileAsMesh: 0
             Tilt: 0
        TiltAxis: [1 0 0]
```

```
show(plat)
```



See Also

`installedAntenna` | `stlwrite`

Topics

“Rotate Antennas and Arrays”
“Hybrid MoM-PO Method for Metal Antennas with Large Scatterers”

Introduced in R2019a

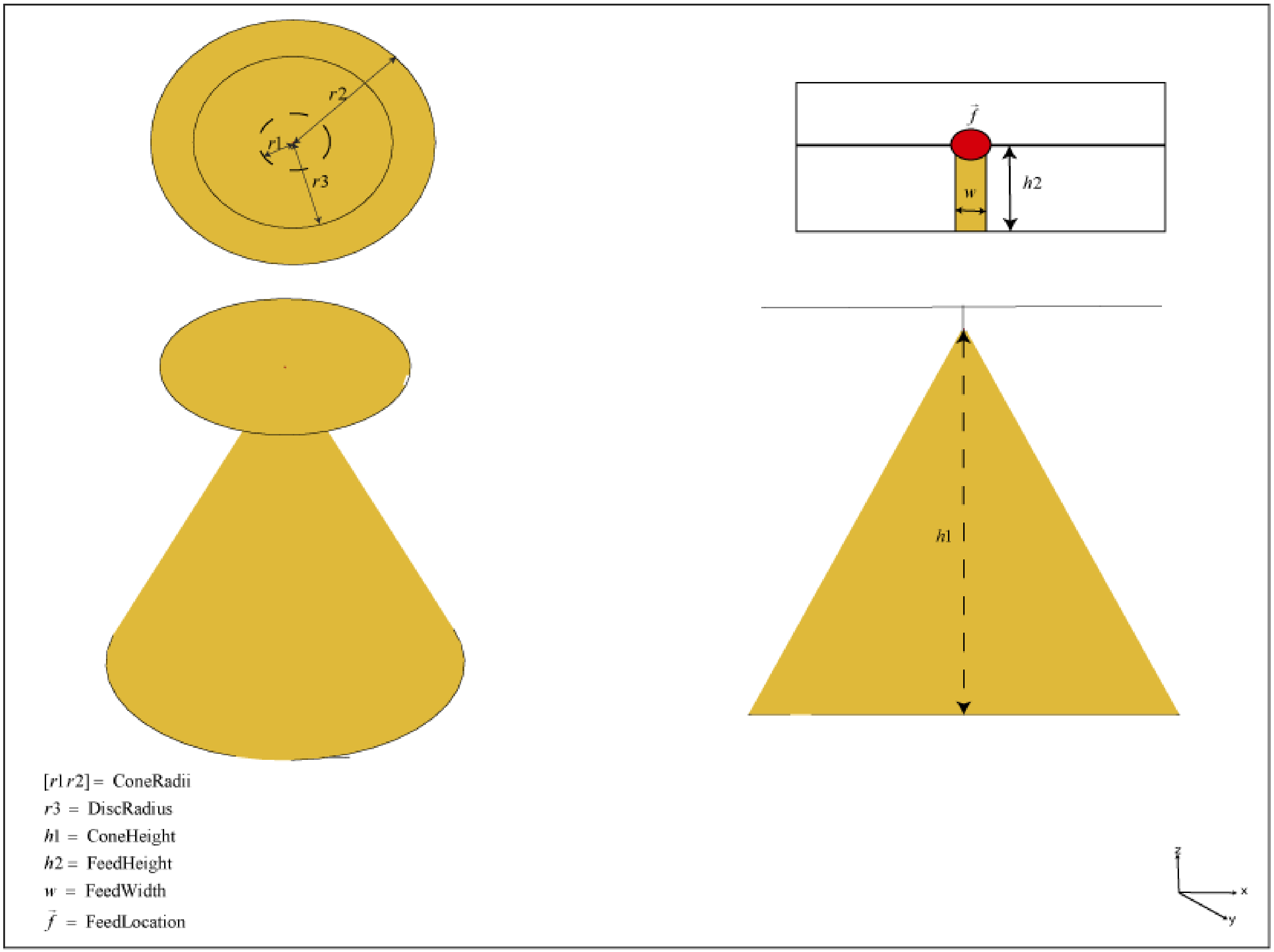
discone

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 2.09 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.

Output Arguments

`ant` — Discone antenna
`discone` object

Discone antenna, returned as a `discone` object.

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` — Diameter of feed

`4.2640e-04` (default) | real-valued scalar

Diameter of the feed, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0050`

Example: `ant.FeedWidth = 0.0050`

Data Types: `double`

`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: `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 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 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 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'`

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

Object Functions

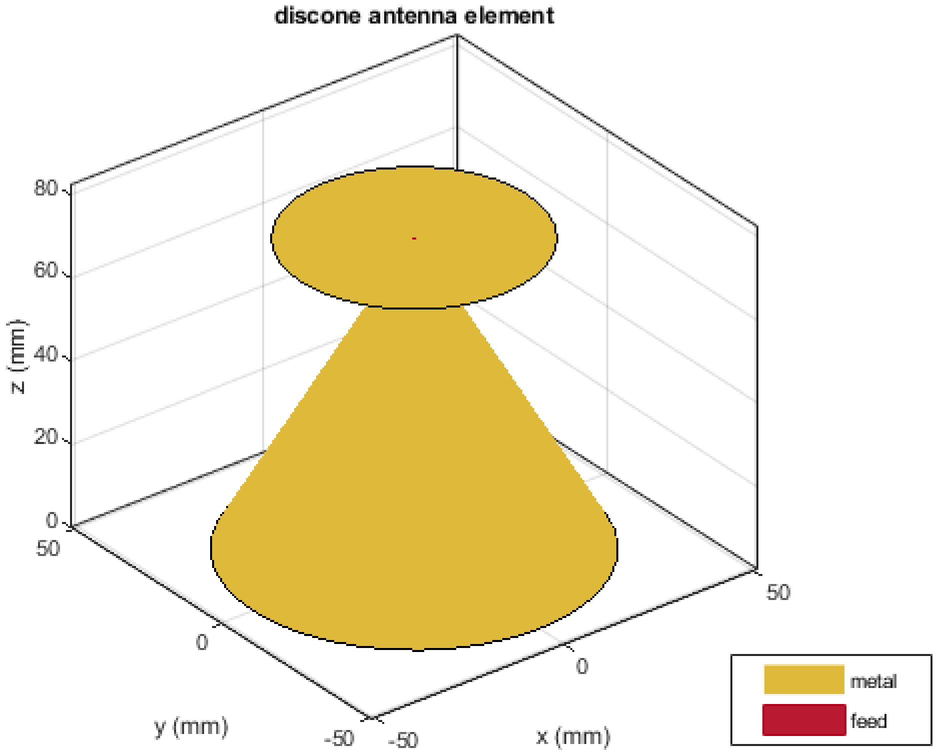
show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

Default Discone Antenna and Radiation Pattern

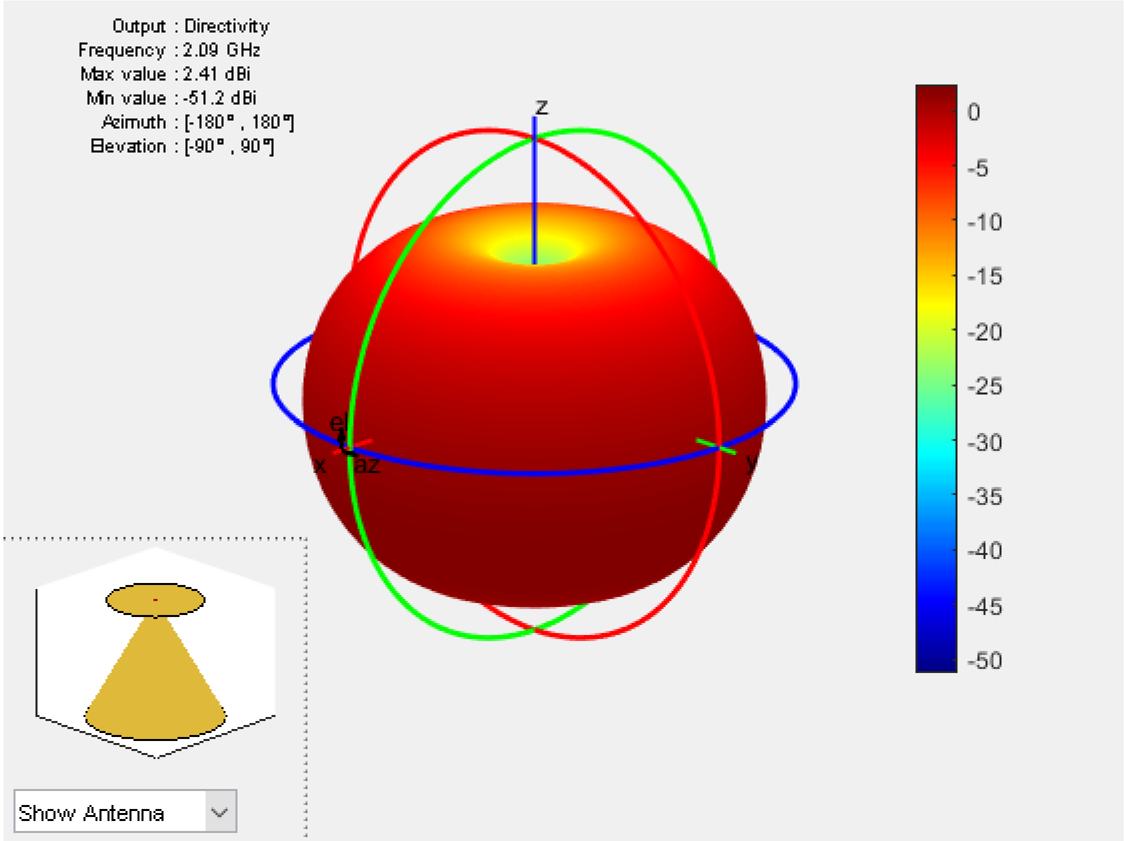
Create and view a default discone antenna.

```
ant = discone;
show(ant)
```



Plot the radiation pattern of the antenna at 2.09 GHz.

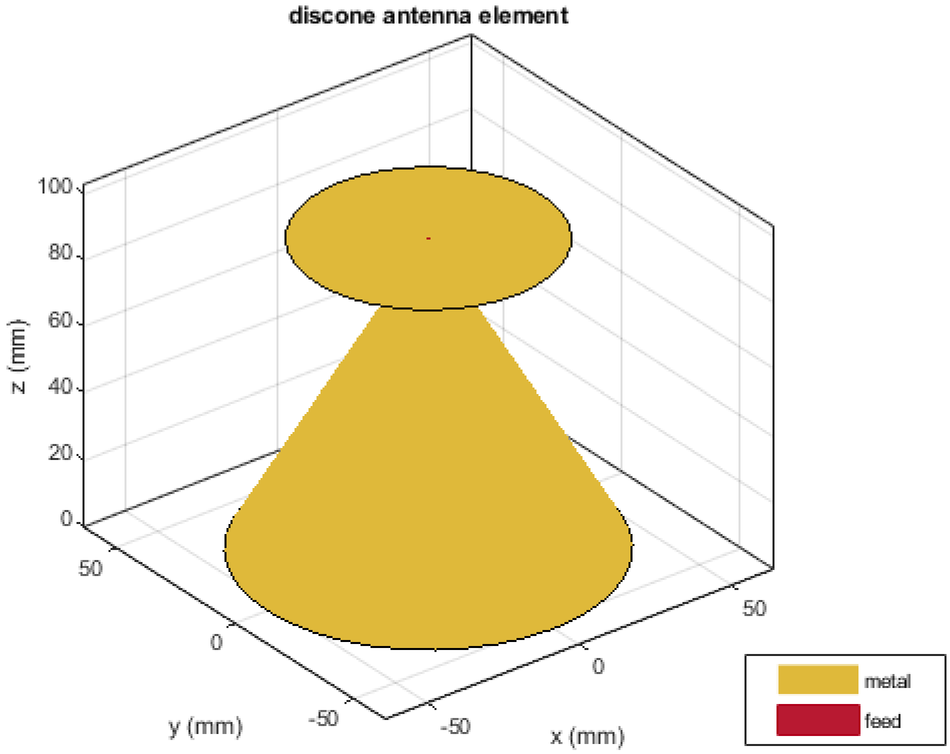
```
pattern(ant,2.09e9)
```



Impedance and Radiation Pattern of Custom Discone Antenna

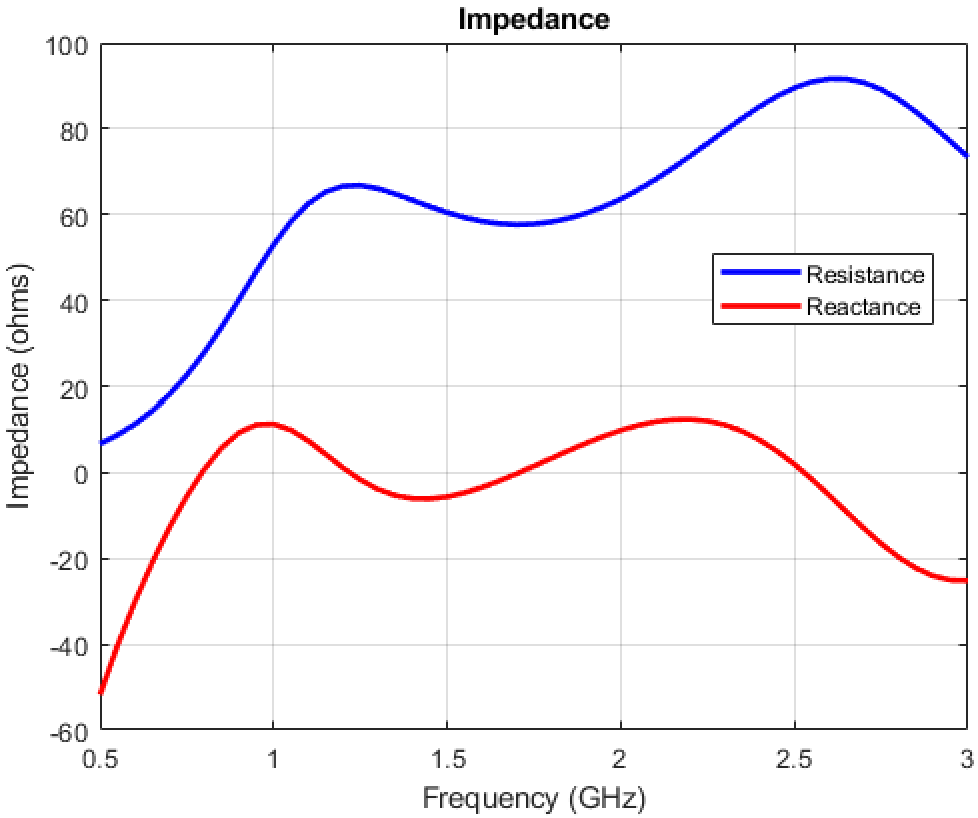
Create and view a discone antenna with specific dimensions.

```
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)
```

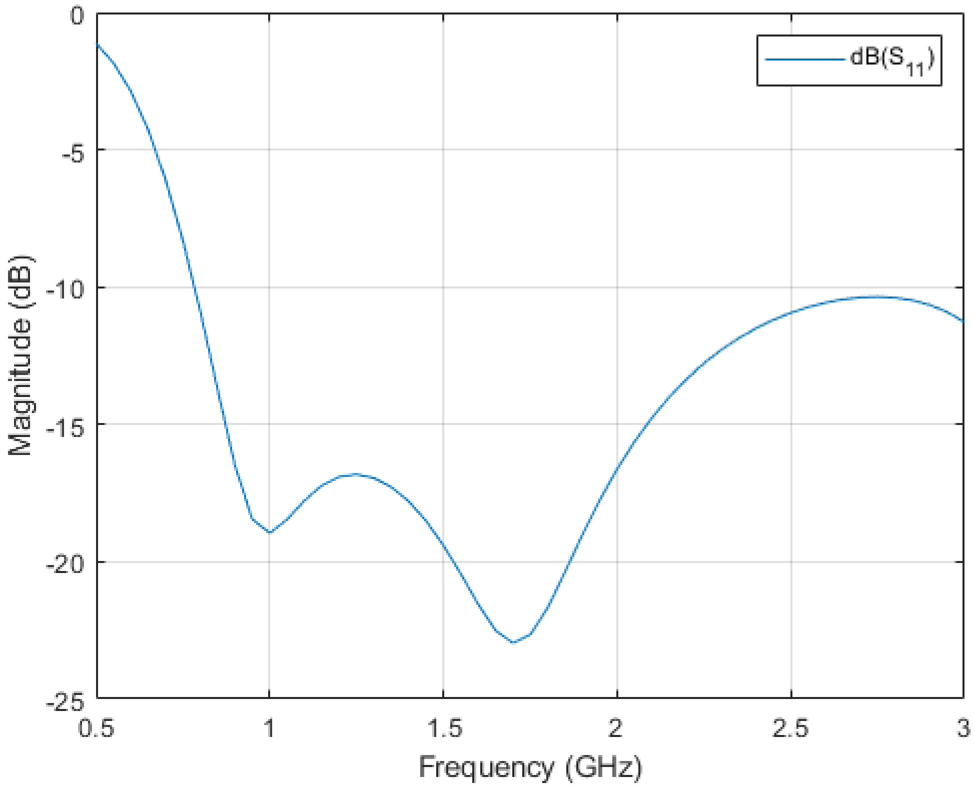


Calculate the impedance of the antenna over the frequency span of 500 MHz to 3 GHz and plot the S-parameters.

```
impedance(ant,linspace(0.5e9,3e9,51));
```

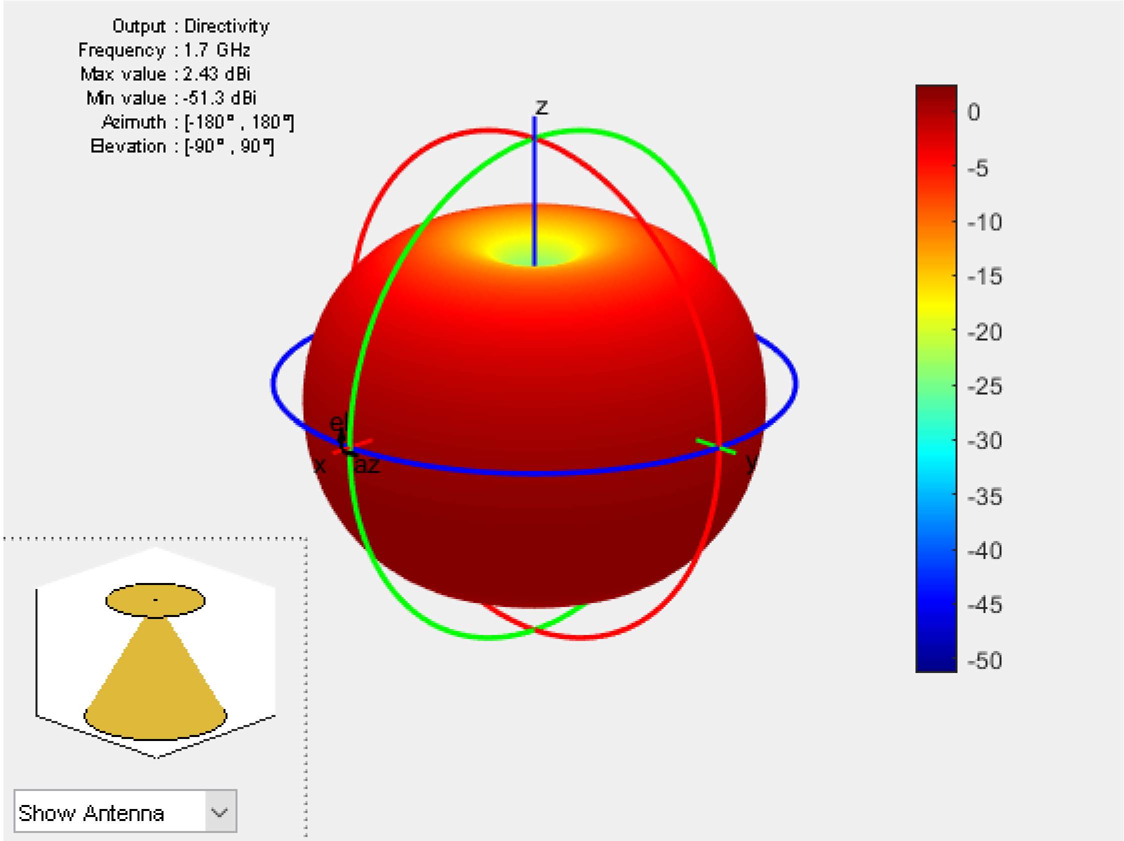


```
s = sparameters(ant,linspace(0.5e9,3e9,51));
figure;
rfplot(s);
```



Plot the radiation pattern of the antenna at 1.7 GHz.

```
pattern(ant,1.7e9);
```



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

`bicone` | `cavityCircular`

Topics

“Rotate Antennas and Arrays”

Introduced in R2019b

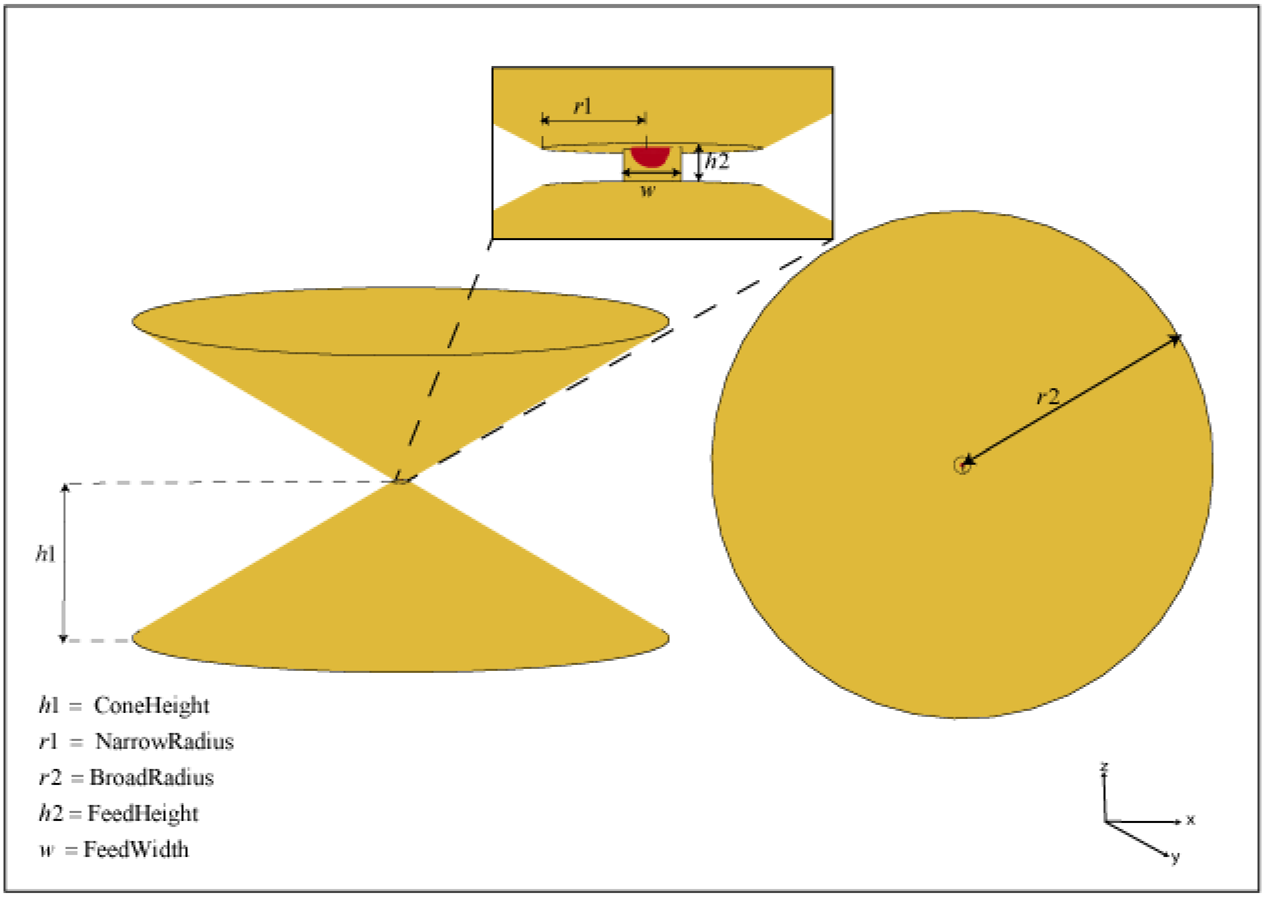
bicone

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.

Output Arguments

`ant` — Bicone antenna
`bicone` object

Bicone antenna, returned as a `bicone` object.

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` — Diameter of feed

`4.2640e-04` (default) | real-valued scalar

Diameter of the feed, specified as a real-valued scalar in meters.

Example: `'FeedWidth',0.0050`

Example: `ant.FeedWidth = 0.0050`

Data Types: `double`

`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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

Default Bicone Antenna and Radiation Pattern

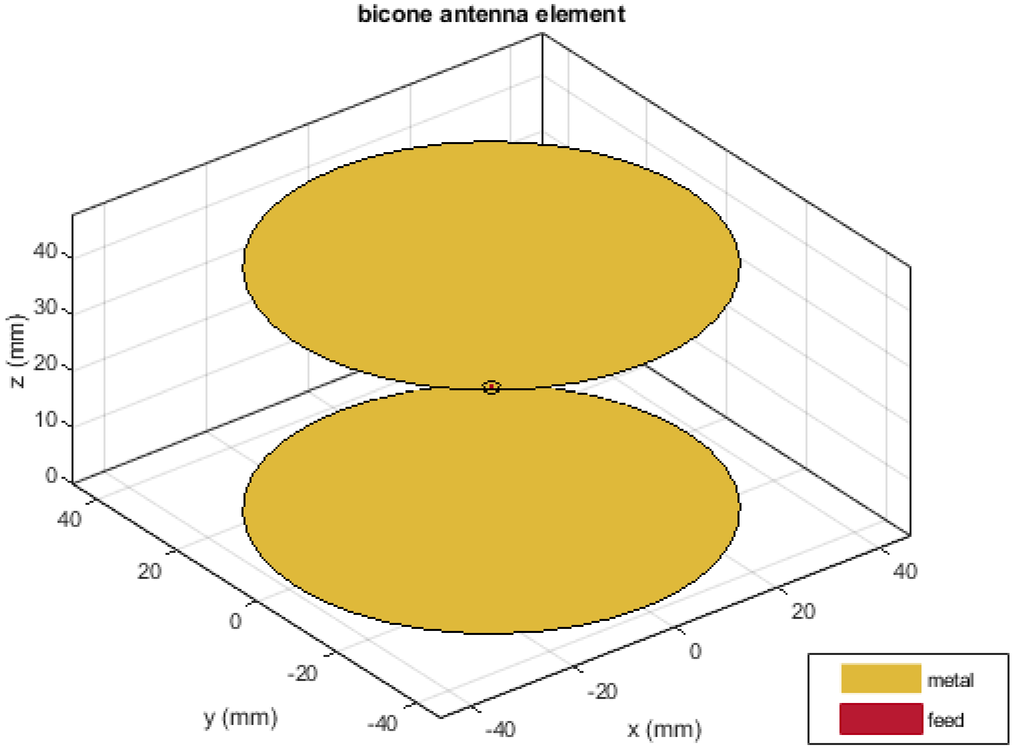
Create and view a default bicone antenna.

```
ant = bicone
```

```
ant = 
  bicone with properties:

      ConeHeight: 0.0215
    NarrowRadius: 0.0013
     BroadRadius: 0.0385
      FeedHeight: 5.0000e-04
       FeedWidth: 1.0000e-03
```

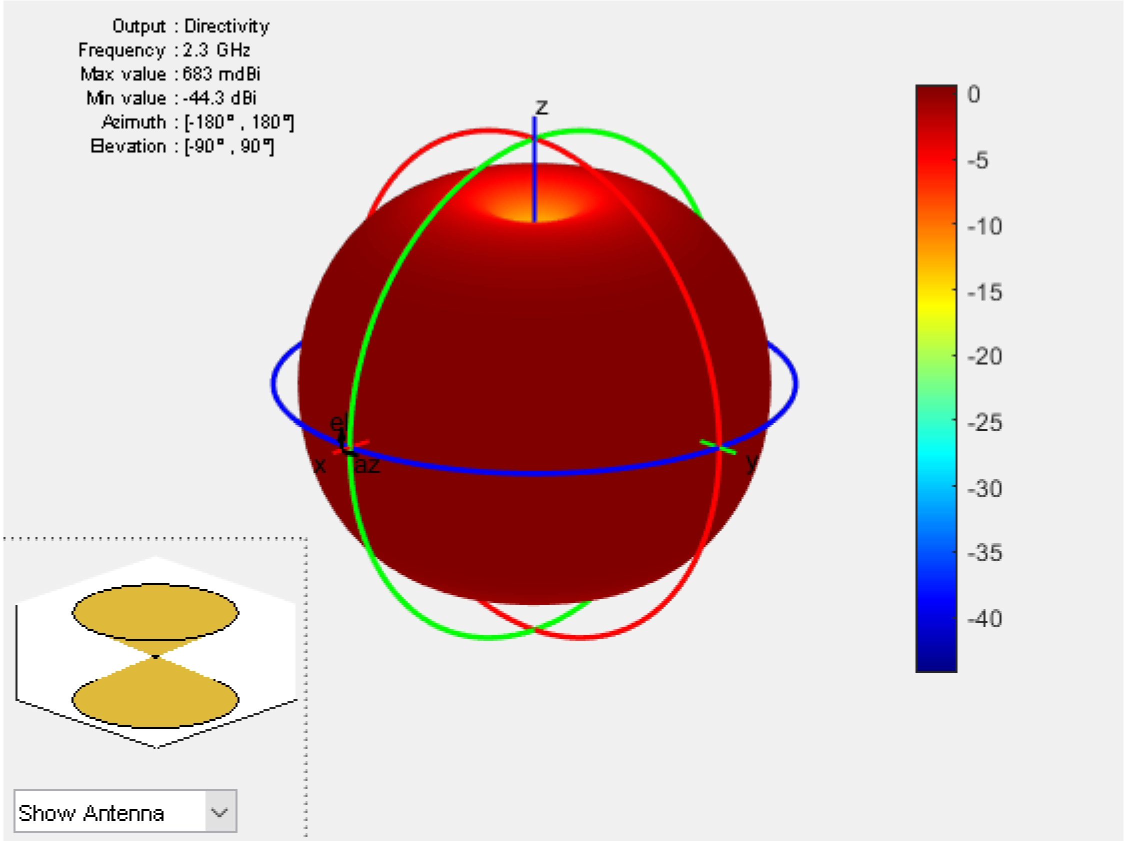
```
        Tilt: 0
    TiltAxis: [1 0 0]
        Load: [1x1 lumpedElement]
```

```
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 : 683 mdBi
Min value : -44.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

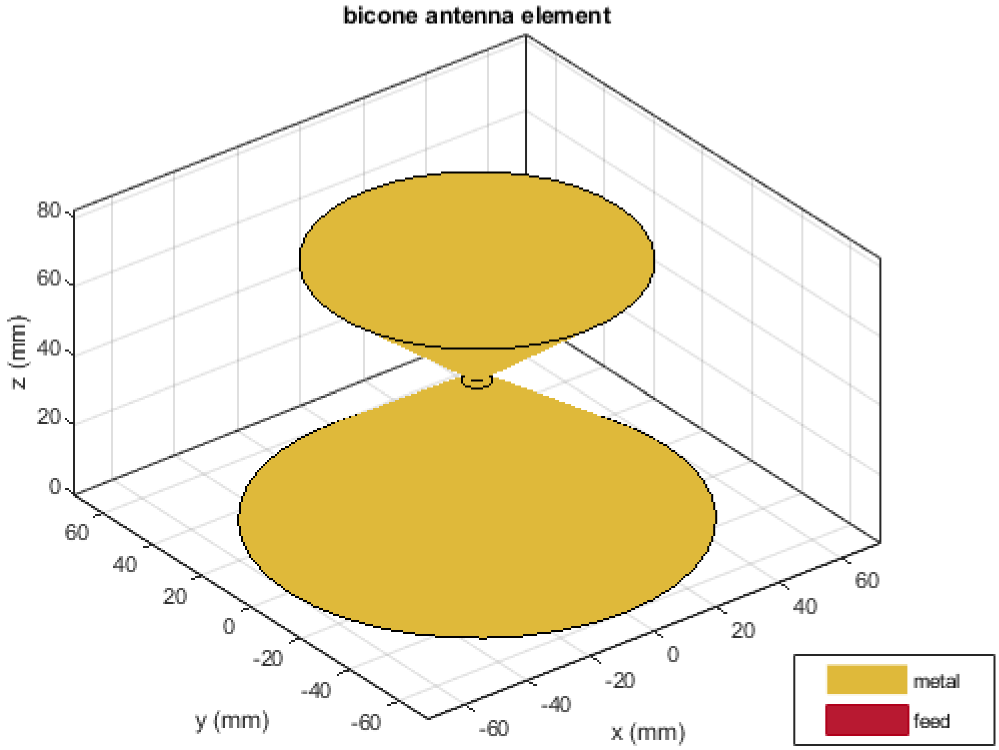
Impedance of Bicone Antenna with Asymmetrical Cones

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)
```

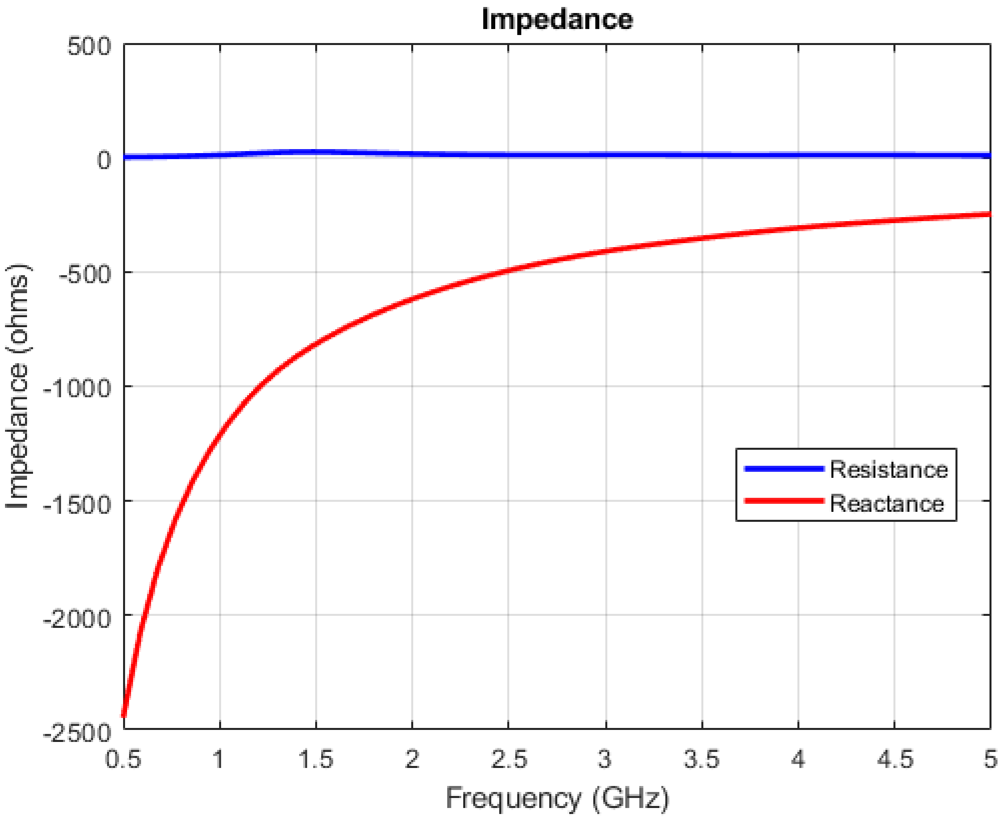
```
ant = 
  bicone with properties:

      ConeHeight: [0.0337 0.0400]
    NarrowRadius: [0.0020 0.0040]
     BroadRadius: [0.0447 0.0600]
      FeedHeight: 1.0000e-03
       FeedWidth: 0.0020
            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));
```



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”

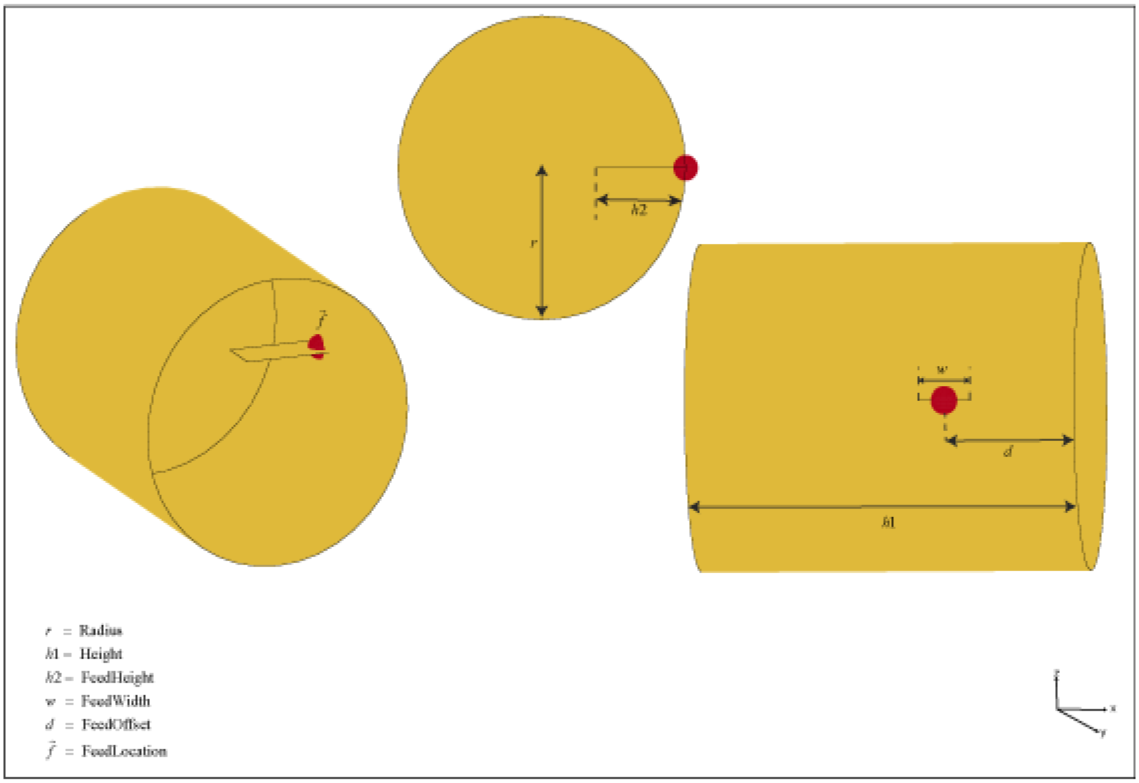
Introduced in R2019b

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.

Output Arguments

`ant` — Circular waveguide
`waveguideCircular` object

Circular waveguide, returned as a `waveguideCircular` object.

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`

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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

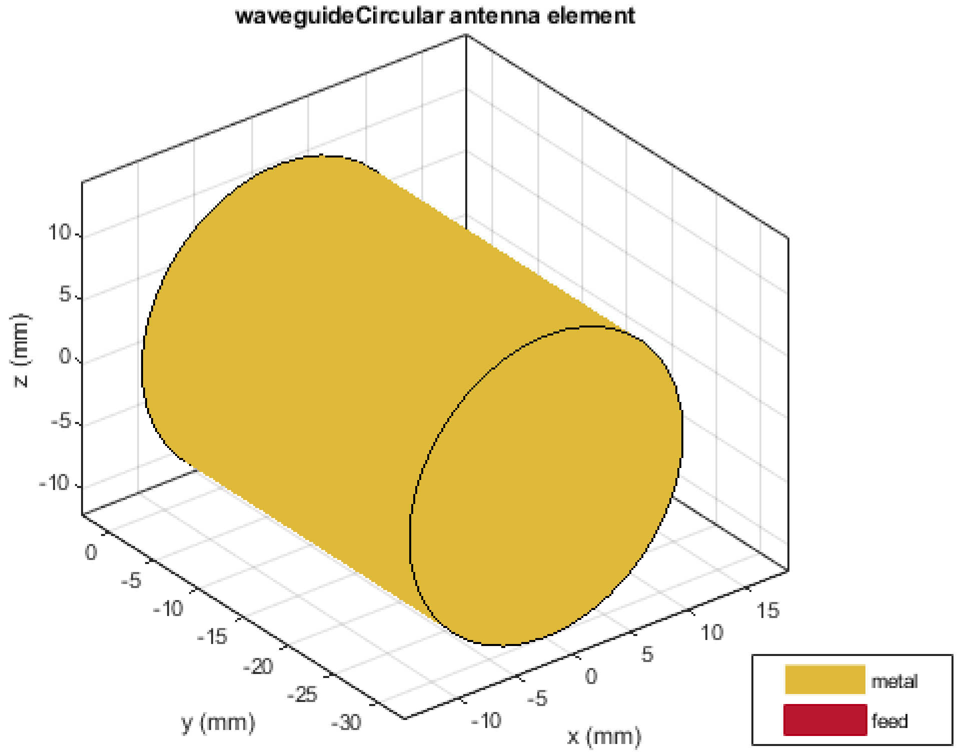
Default Circular Waveguide and Radiation Pattern

Create and view a default circular waveguide.

```
ant = waveguideCircular
```

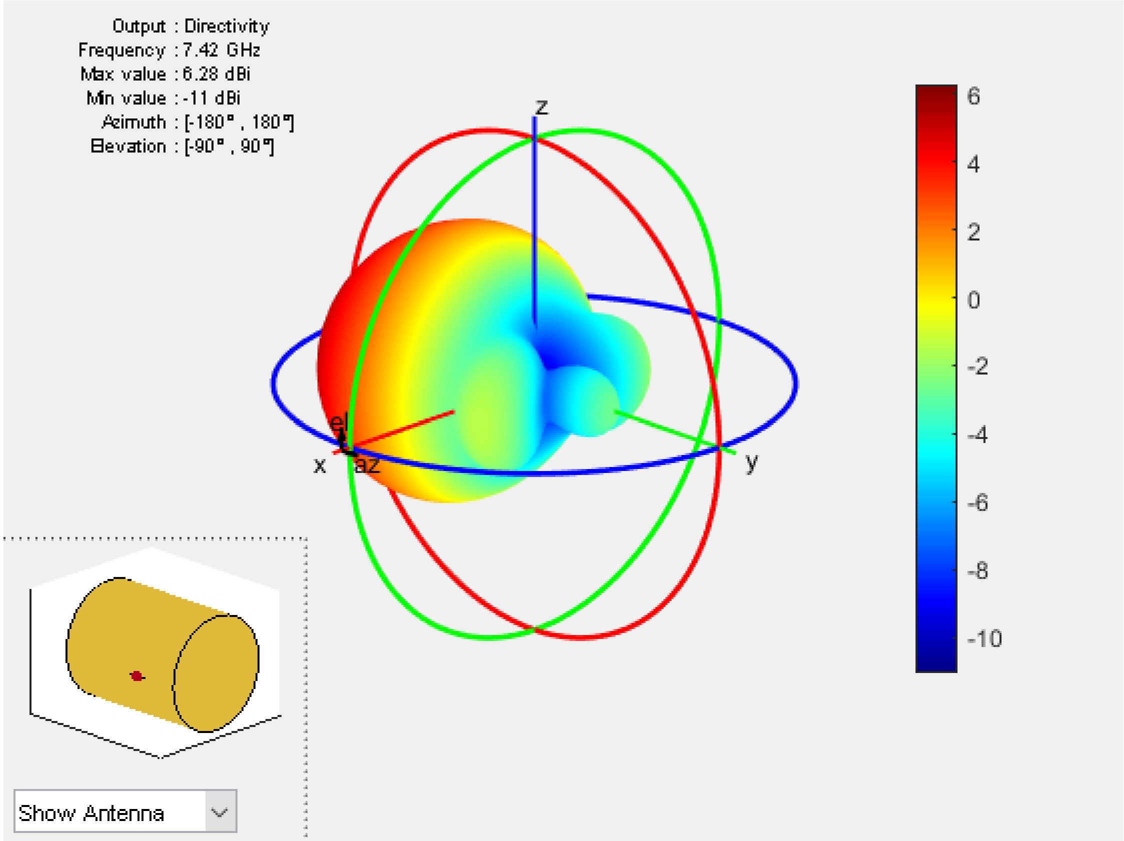
```
ant = 
  waveguideCircular with properties:

        Radius: 0.0120
        Height: 0.0300
    FeedHeight: 0.0075
     FeedWidth: 0.0040
    FeedOffset: 0.0100
          Tilt: 0
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 7.42 GHz.

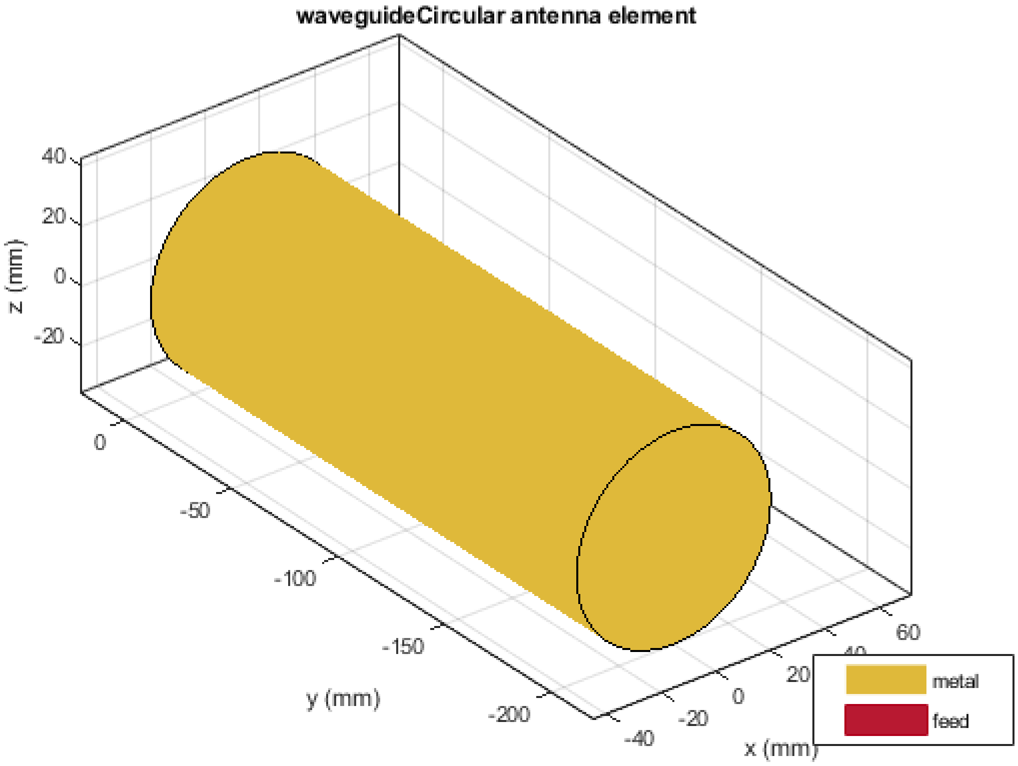
```
pattern(ant,7.42e9)
```


Output : Directivity
Frequency : 7.42 GHz
Max value : 6.28 dBi
Min value : -11 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

S-Parameters and Impedance of Custom Circular Waveguide

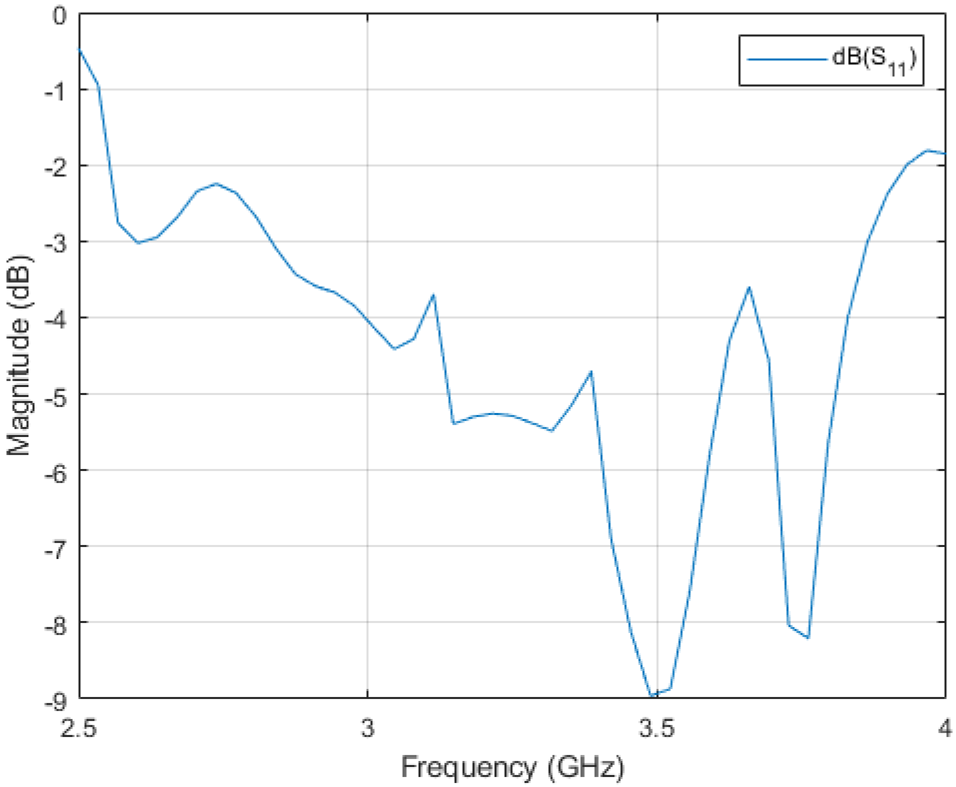
Create a circular waveguide with the following dimensions.

```
ant=waveguideCircular('Radius',35.7e-3,'Height',200e-3,...
       'Feedwidth',26e-3,'FeedHeight',34.71e-3,'FeedOffset', 42.42e-3);
show(ant);
```

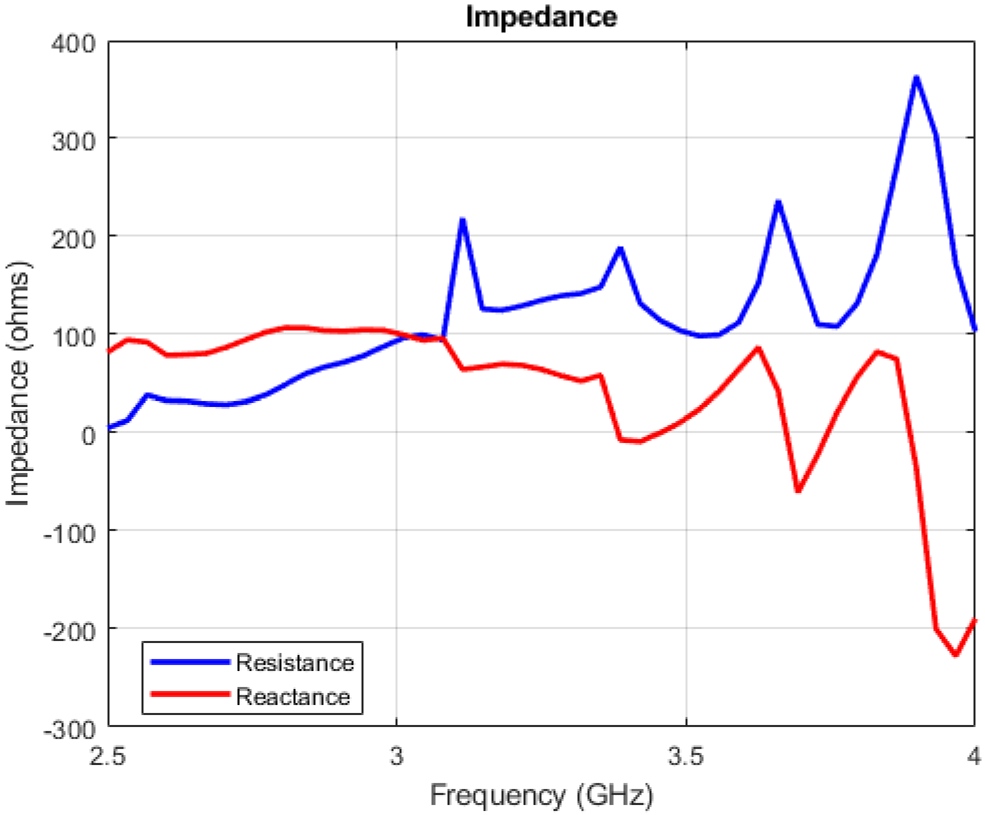


Plot the s-parameters and impedance of the waveguide.

```
s=sparameters(ant,linspace(2.5e9,4e9,45));
rfplot(s);
```



```
figure;
impedance(ant,linspace(2.5e9,4e9,45));
```



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”

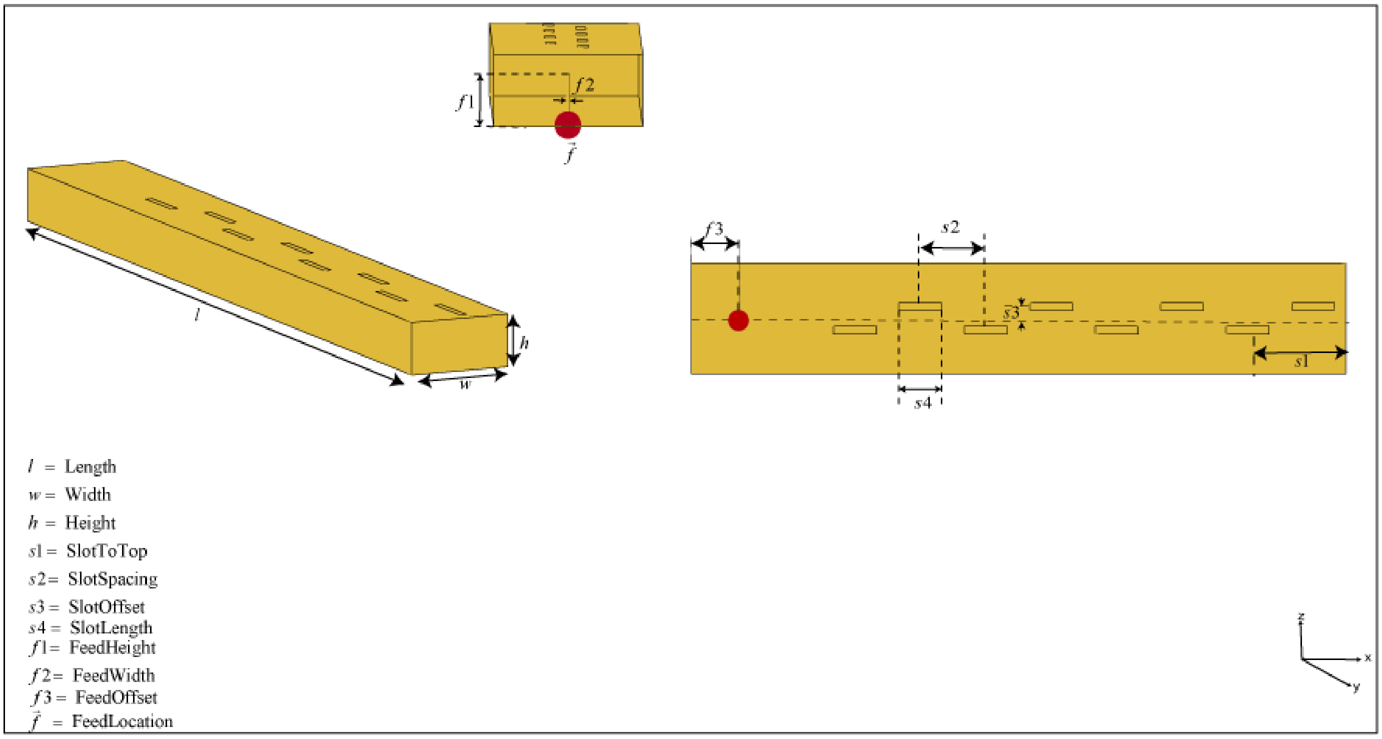
Introduced in R2019b

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 X-Y 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.

Output Arguments

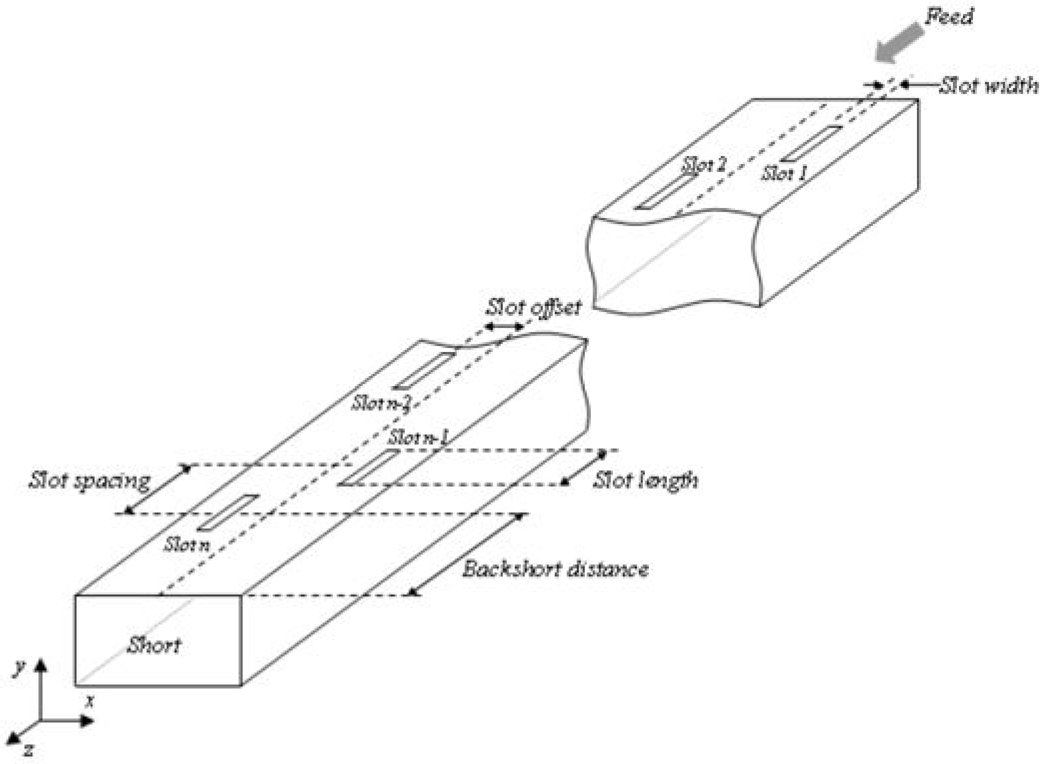
`ant` — Slotted waveguide antenna
`waveguideSlotted` object

Slotted waveguide antenna, returned as a `waveguideSlotted` object.

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.



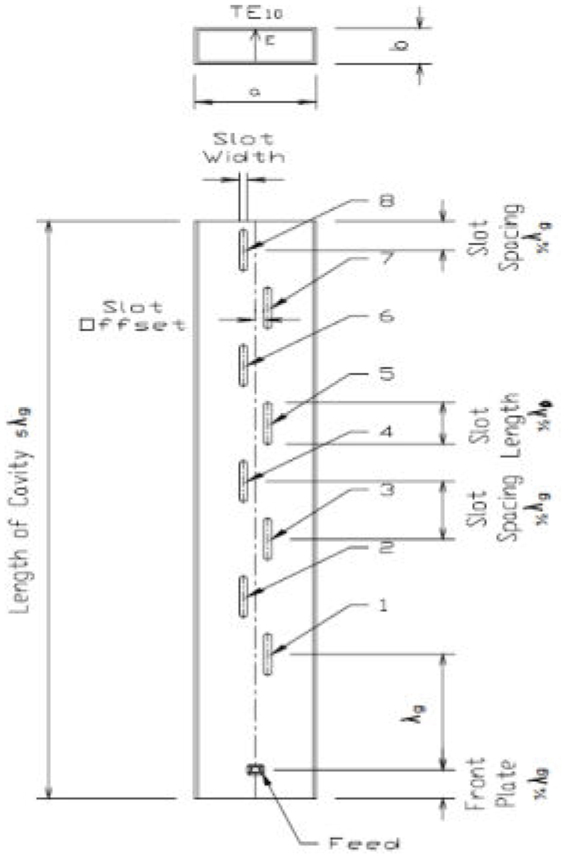
Example: `'Length',0.760`

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

Shape of waveguide slot, specified as one of the following objects: `antenna.Circle`, `antenna.Polygon`, `antenna.Rectangle`.

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

Slot displacement from the centreline of the width of the waveguide to the center of the slot, specified as a real-valued scalar in meters.

Example: `'SlotOffset',0.0560`

Example: `ant.SlotOffset = 0.0560`

Data Types: `double`

`SlotAngle` — Slot angle

`0` (default) | real-valued scalar | two-element vector

Slot angle, specified as a real-valued scalar in degrees or a two-element vector with each element unit in degrees. In slotted waveguide the slots are in pairs. You use a two-element vector when you want one slot in the pair to be tilted at a different angle form the other.

Example: `'SlotAngle',[20 10]`

Example: `ant.SlotOffset = [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`

`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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

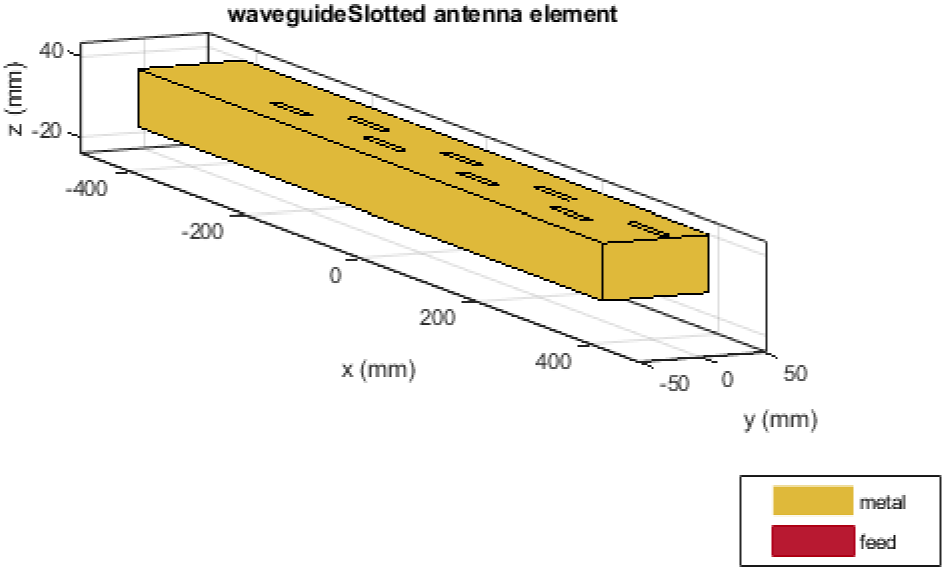
Default Slotted Waveguide Antenna and Radiation Pattern

Create and view a slotted waveguide antenna with default property values.

```
ant = waveguideSlotted
```

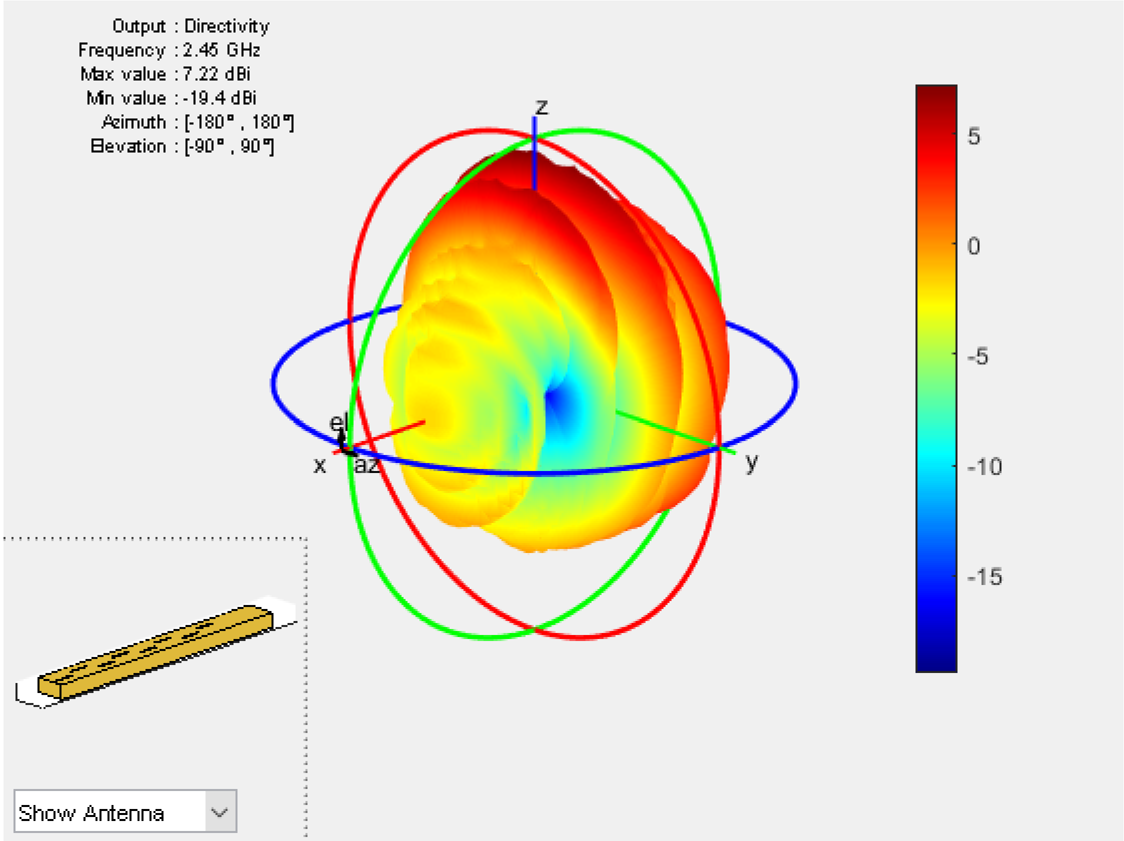
```
ant = 
  waveguideSlotted with properties:

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

```
show(ant)
```



Plot the radiation pattern of the antenna at 2.45 GHz.

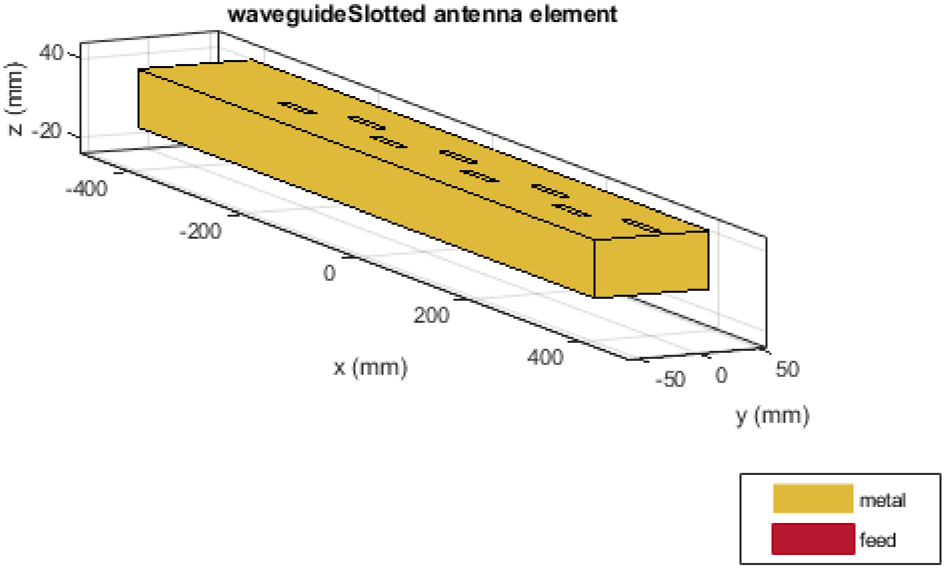
```
pattern(ant, 2.45e9)
```



Impedance and S-Parameters of Custom Slotted Waveguide Antenna

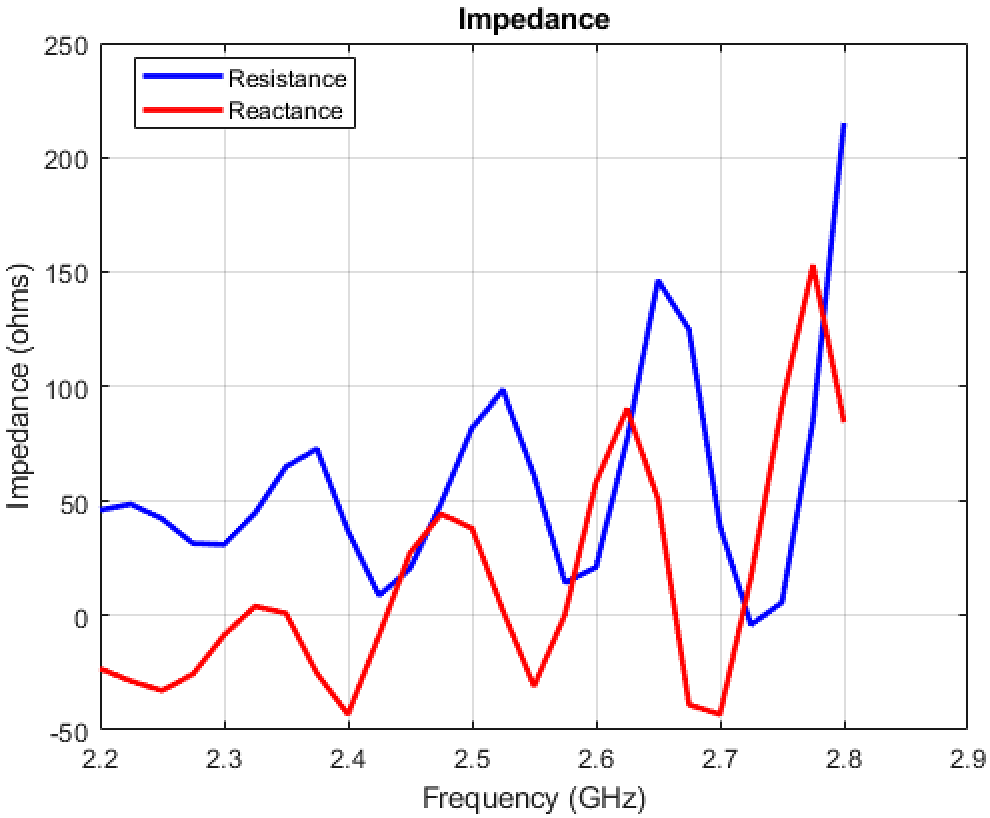
Create a slotted waveguide antenna with the following dimensions.

```
 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)
```

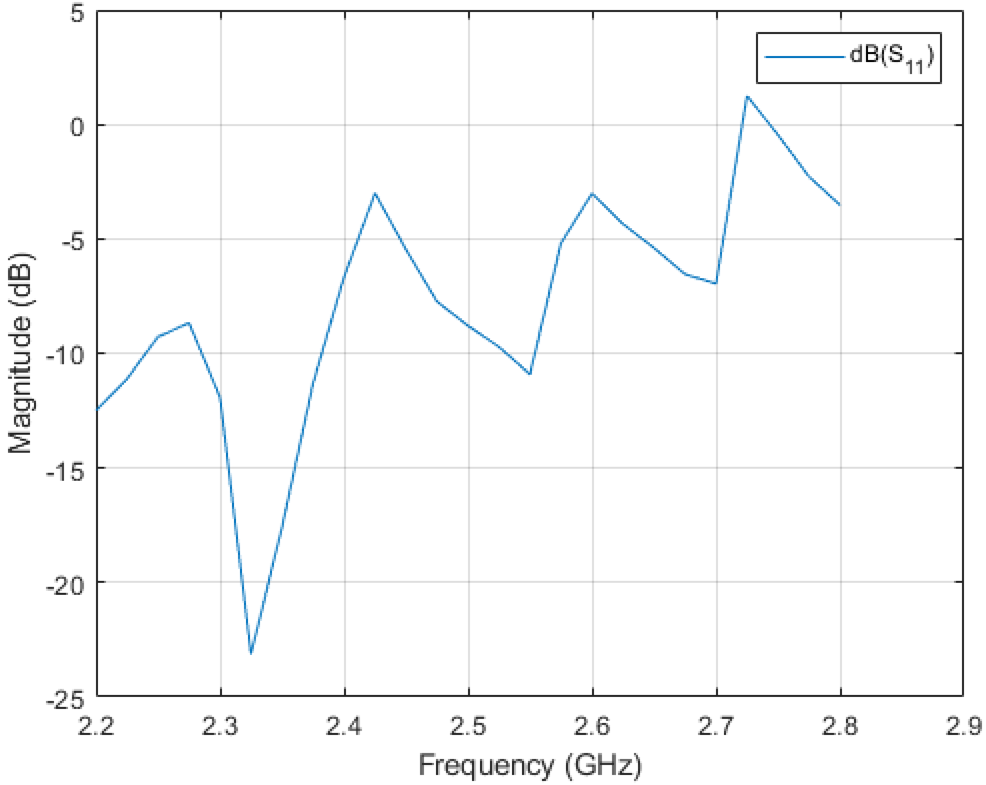


Plot impedance and S-parameters from 2.2 GHz to 2.8 GHz.

```
freq = 2.2e9:0.025e9:2.8e9;
figure;
impedance(ant,freq);
```



```
s = sparameters(ant,freq);
figure;
rfplot(s);
```



References

[1] Perovic, Una. " Investigation of Rectangular, Unidirectional, Horizontally Polarized Waveguide Antenna with Longitudinal Slotted Arrays Operating at 2.45 GHz".

See Also

`cavityCircular` | `waveguide` | `waveguideCircular`

Topics

"Rotate Antennas and Arrays"

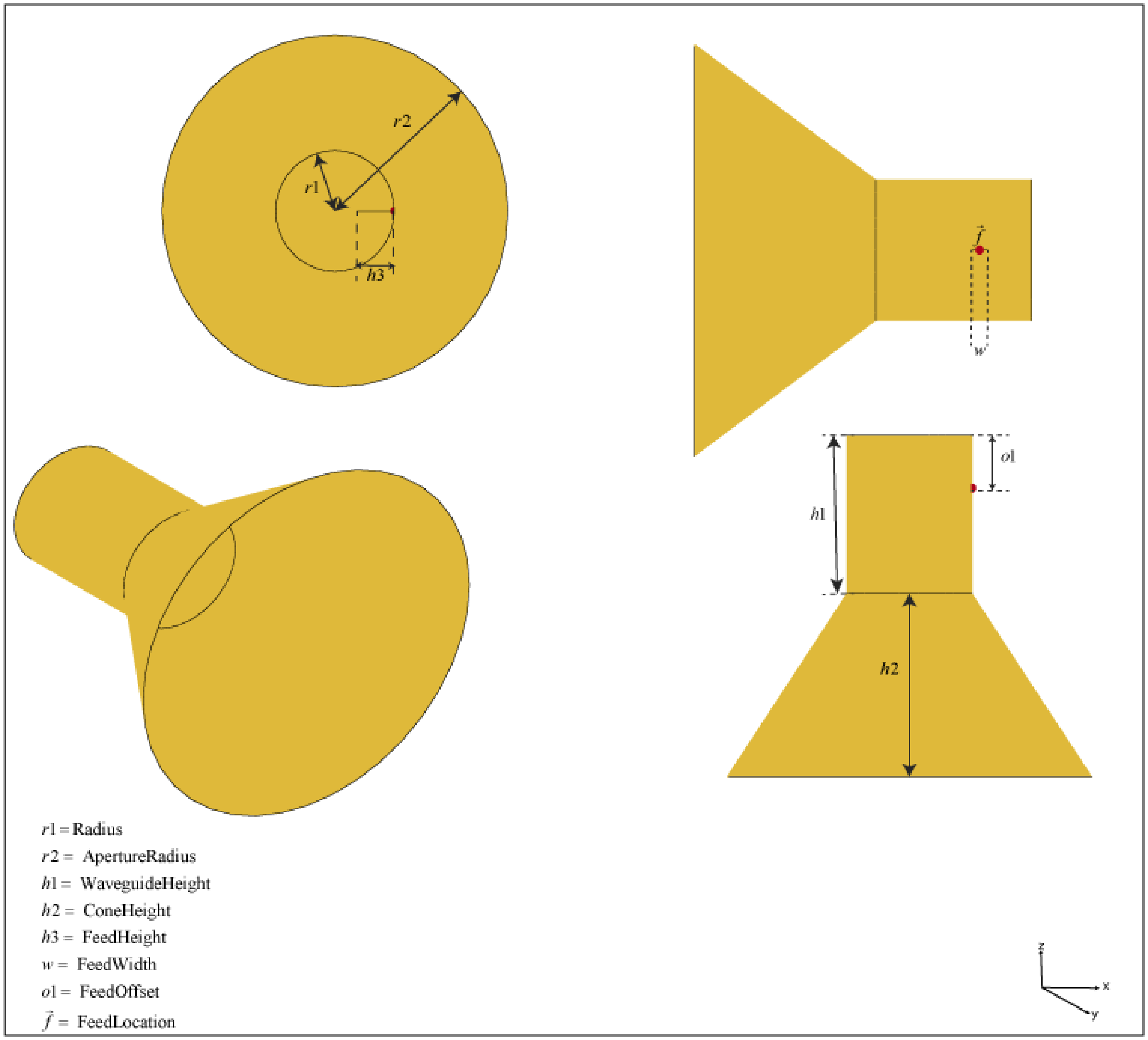
Introduced in R2019b

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.

Output Arguments

`ant` — Conical horn antenna
`hornConical` object

Conical horn antenna, returned as a `hornConical` object.

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

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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object

vswr	Voltage standing wave ratio of antenna

Examples

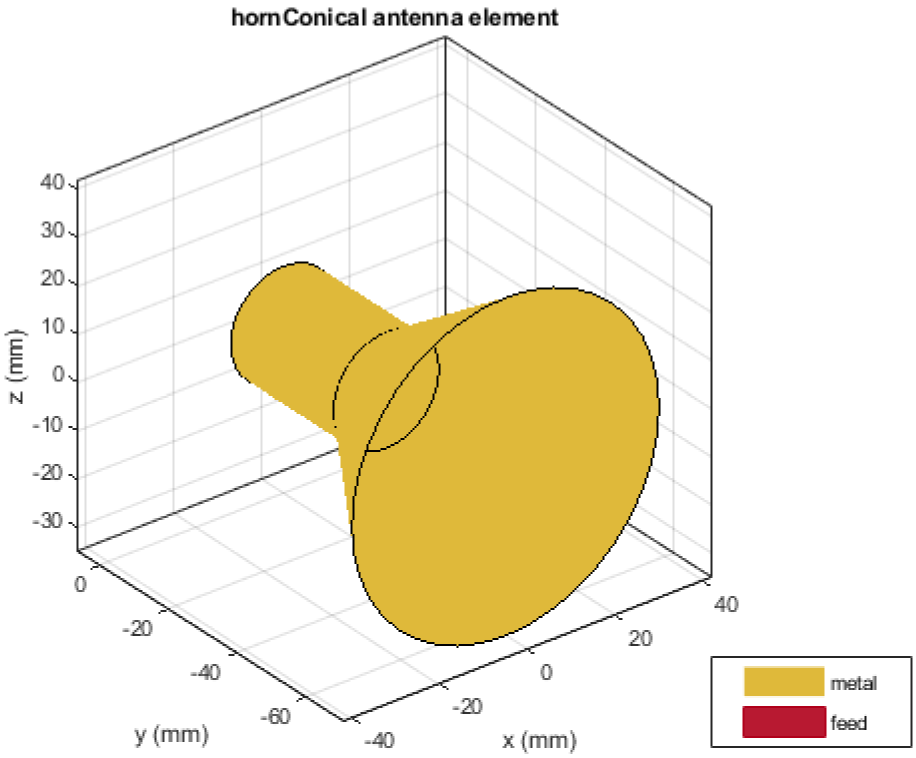
Default Conical Horn and Radiation Pattern

Create and view a default conical horn antenna.

```
ant = hornConical
```

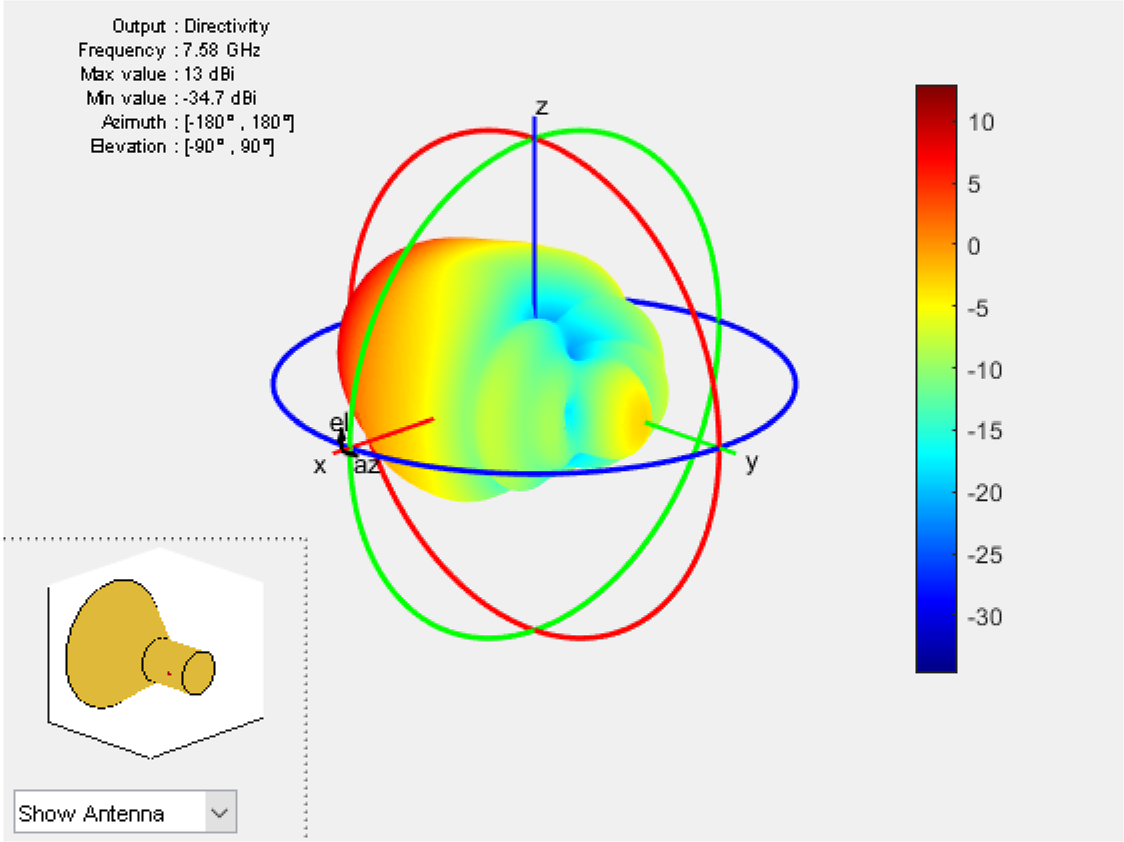
```
ant = 
  hornConical with properties:

             Radius: 0.0120
    WaveguideHeight: 0.0300
         FeedHeight: 0.0075
          FeedWidth: 0.0030
         FeedOffset: 0.0100
         ConeHeight: 0.0348
     ApertureRadius: 0.0350
               Tilt: 0
           TiltAxis: [1 0 0]
               Load: [1x1 lumpedElement]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at 7.58 GHz.

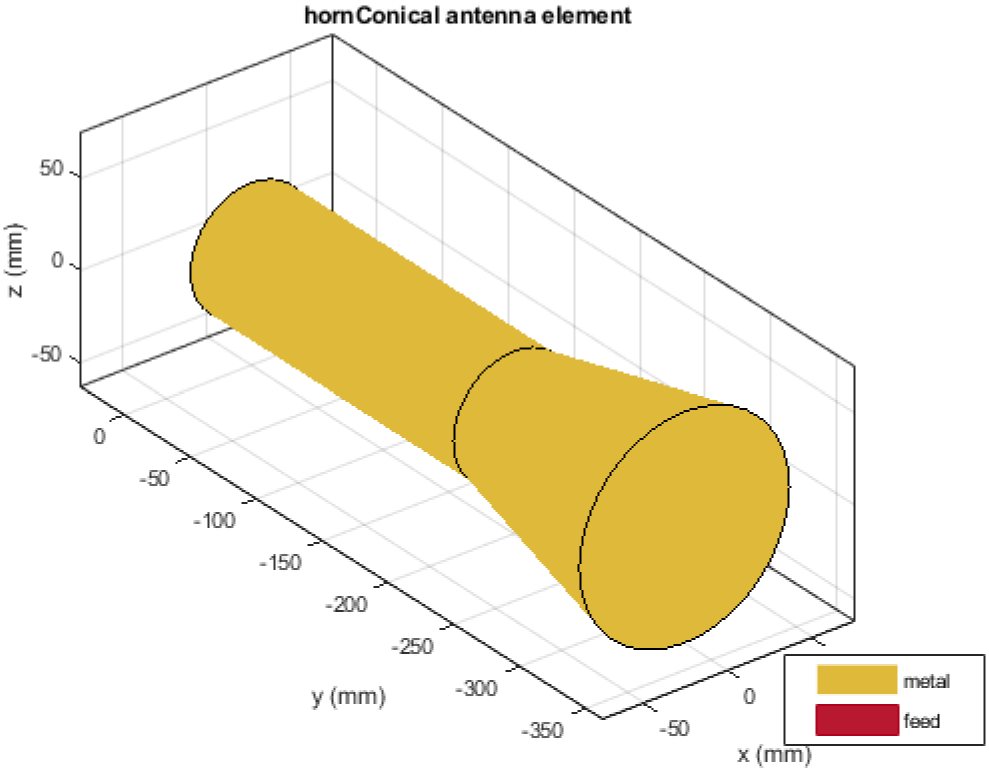
```
pattern(ant,7.58e9)
```



Impedance and S-Parameters of Custom Conical Horn Antenna

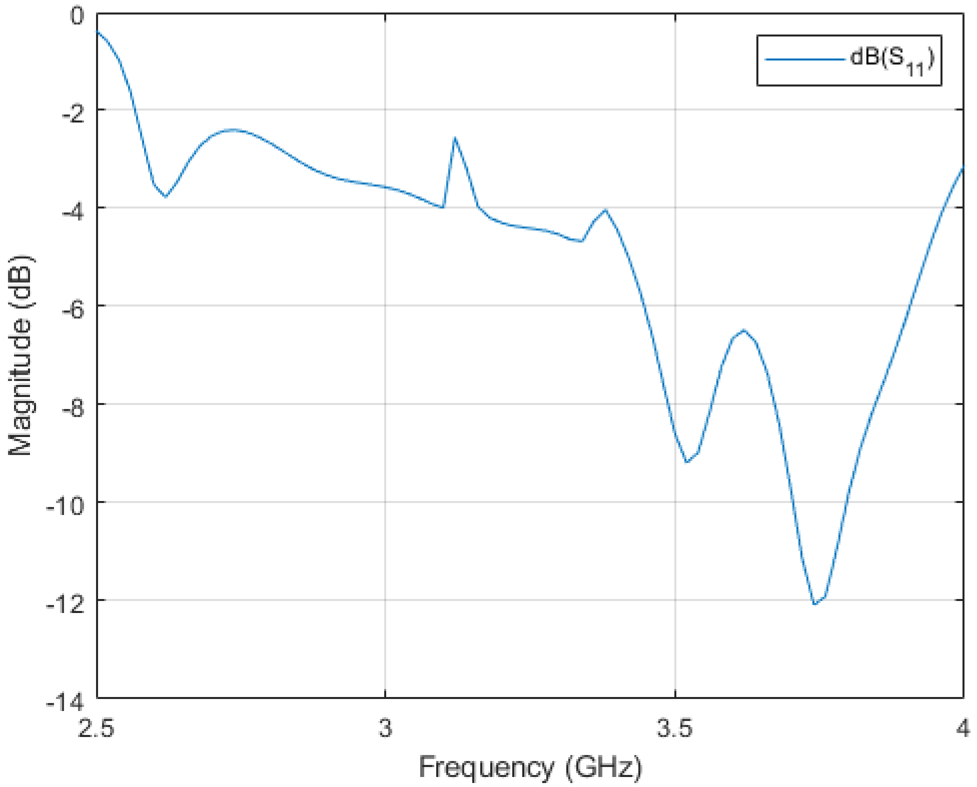
Create a conical horn antenna with the following dimensions.

```
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);
```

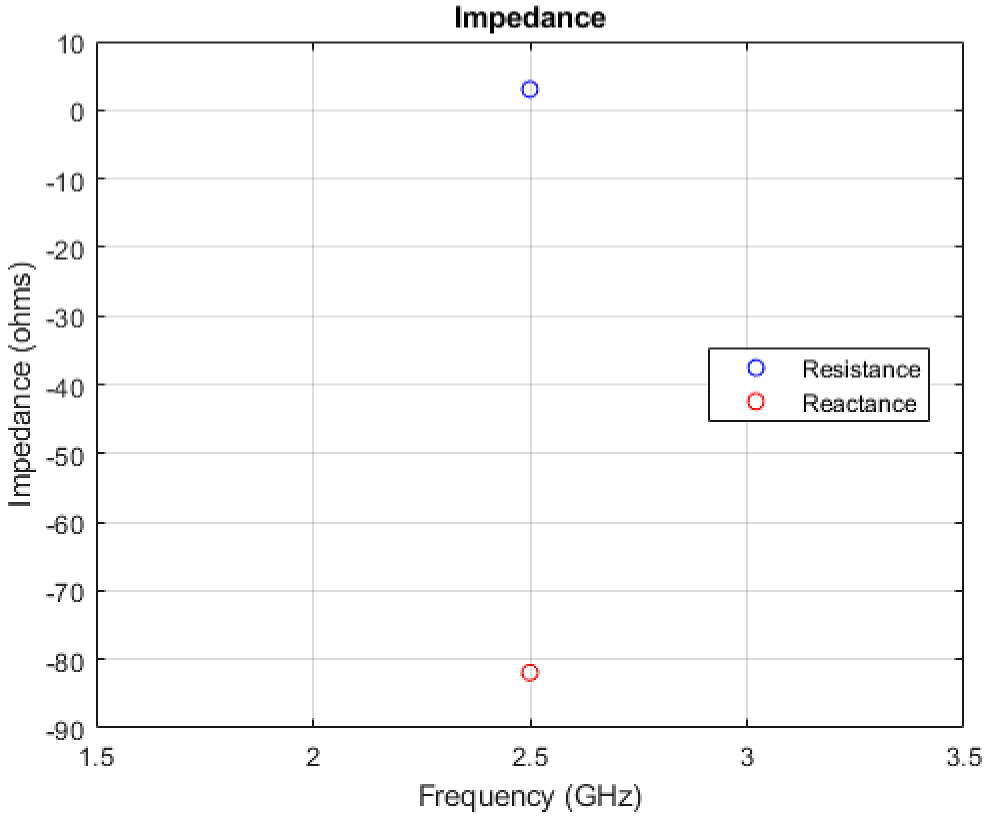


Plot the s-parameters and the impedance of the antenna.

```
s=sparameters(ant,2.5e9:20e6:4e9);
rfplot(s);
```



```
figure;
impedance(ant,2.5e9,20e6:4e9);
```



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` | `horn` | `hornangle2size` | `waveguide`

Topics

"Rotate Antennas and Arrays"

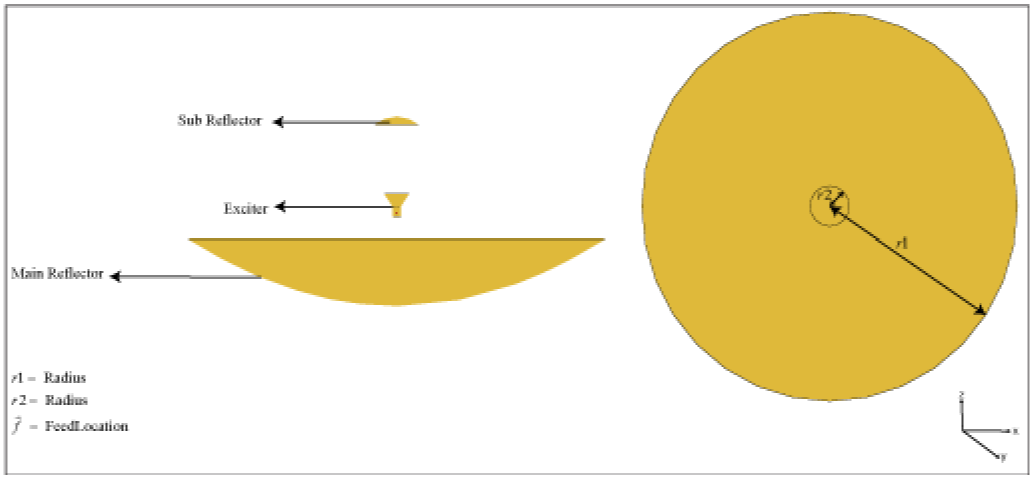
Introduced in R2019b

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 2-515.



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.

Output Arguments

ant — Gregorian antenna
`gregorian` object

Gregorian antenna, returned as a `gregorian` object.

Properties

`Exciter` — Antenna type used as exciter

`hornConical` (default) | antenna object

Antenna type used as exciter, specified as an antenna object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole`

`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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

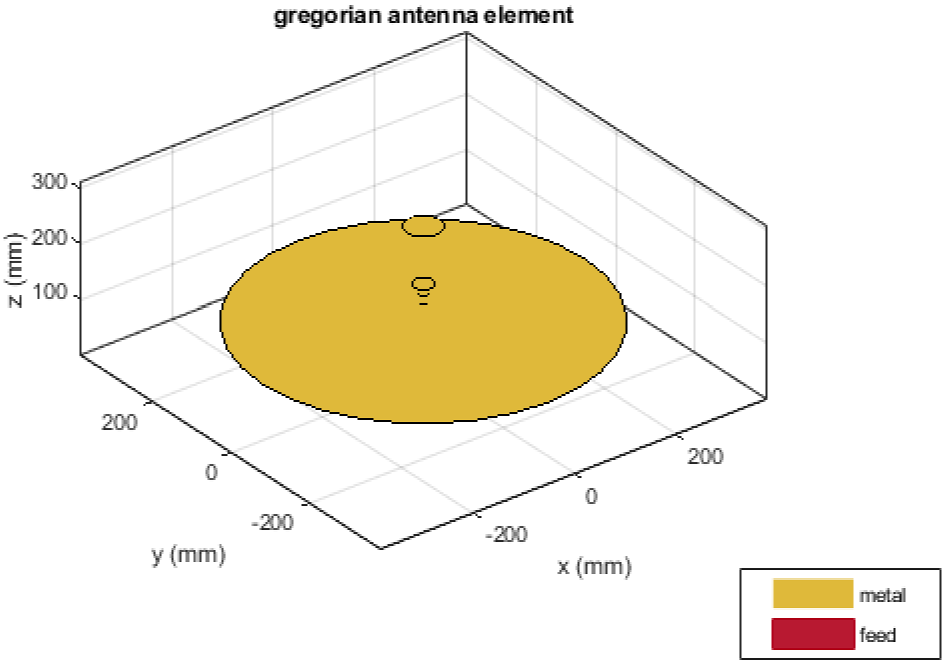
Default Gregorian Antenna and Radiation Pattern

Create and view a default Gregorian antenna.

```
ant = gregorian
```

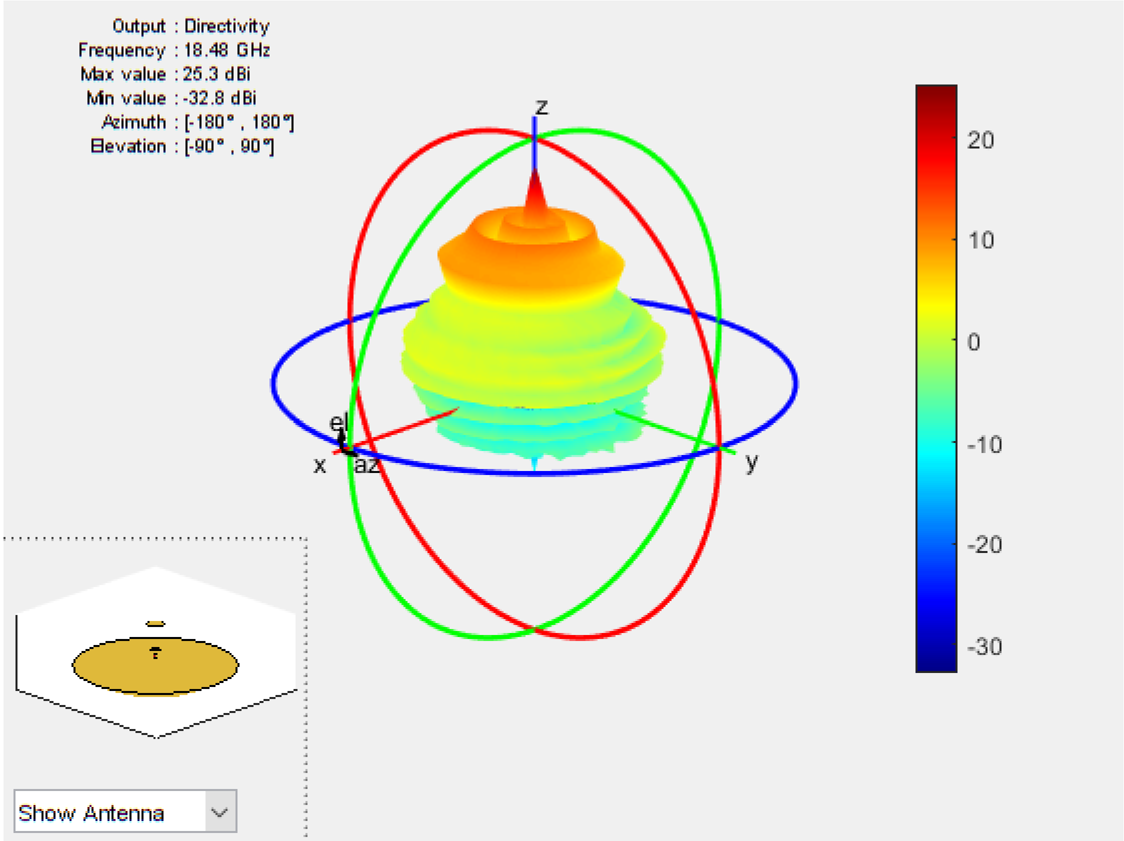
```
ant = 
  gregorian with properties:

        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°]

More About

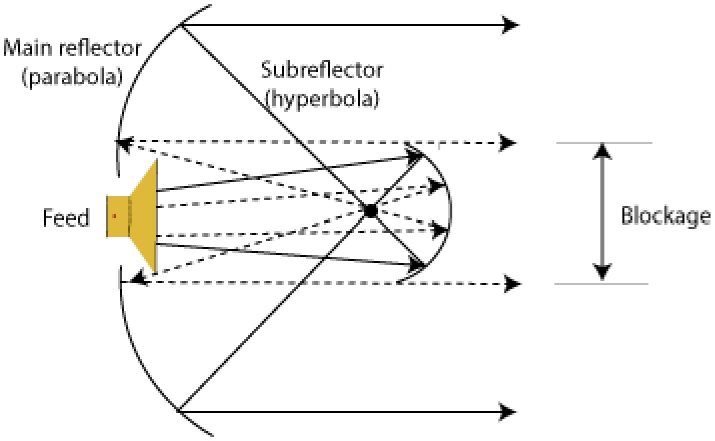
Parabolic Reflector Antennas

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.

See Also

`cassegrain` | `hornConical` | `reflectorParabolic`

Topics

"Rotate Antennas and Arrays"

Introduced in R2019b

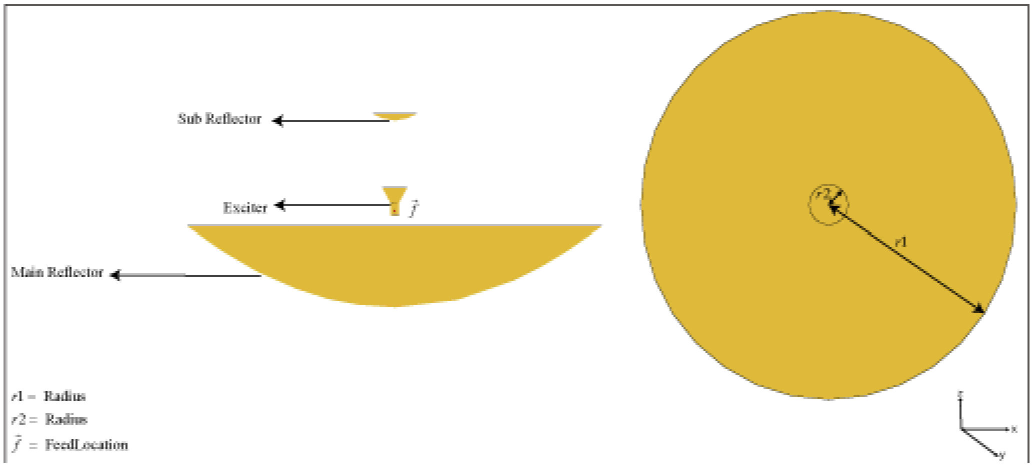
cassegrain

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 2-522.

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.

Output Arguments

ant — Cassegrain antenna
`cassegrain` object

Cassegrain antenna, returned as a `cassegrain` object.

Properties

`Exciter` — Antenna type used as exciter

`hornConical` (default) | antenna object

Antenna type used as exciter, specified as an antenna object.

Example: `'Exciter',dipole`

Example: `ant.Exciter = dipole`

`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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

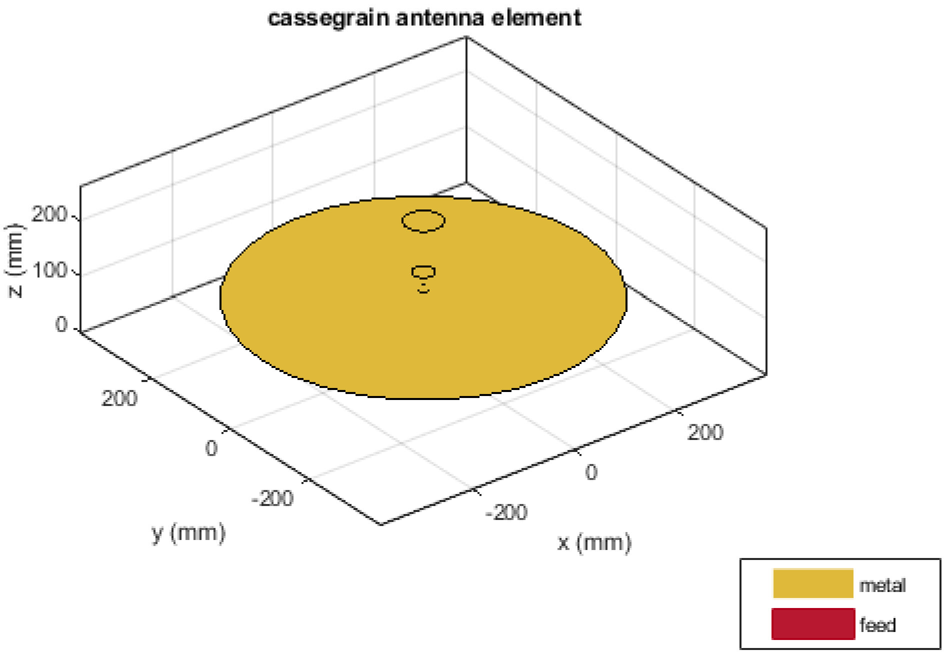
Default Cassegrain Antenna and Radiation Pattern

Create and view a Cassegrain antenna.

```
ant = cassegrain
```

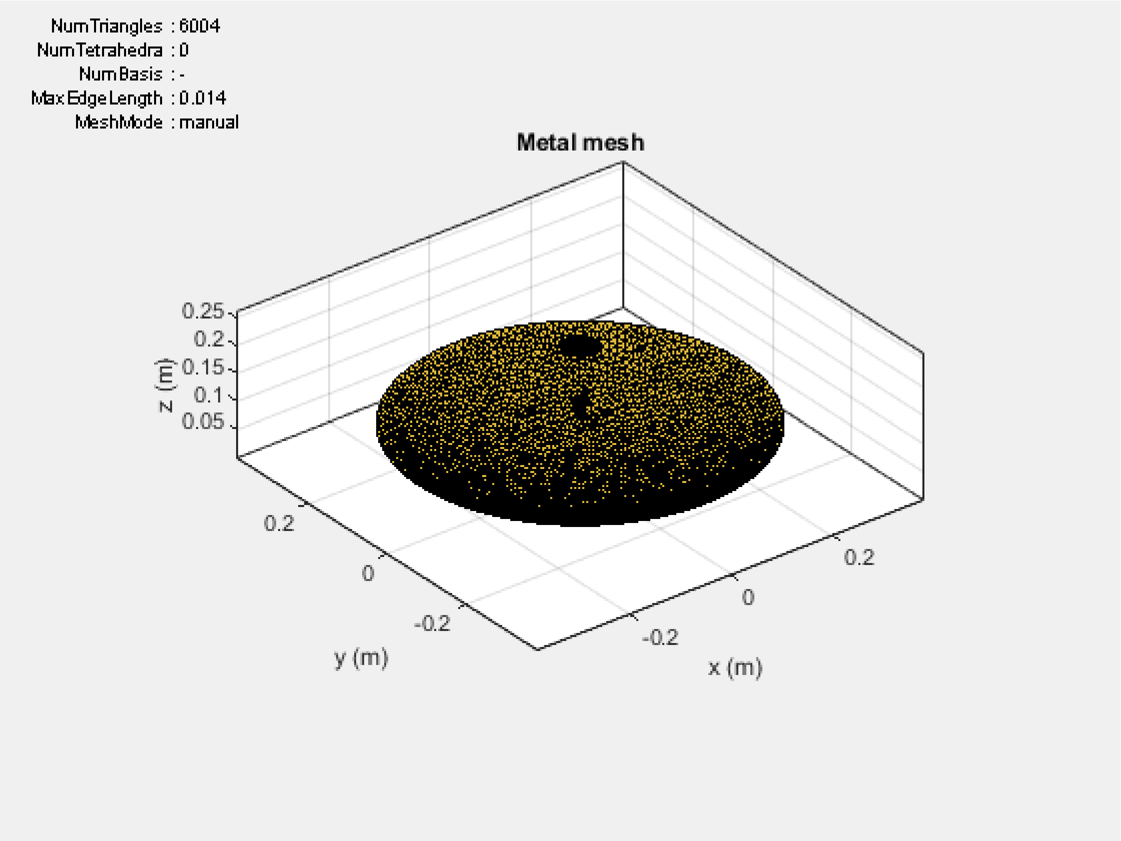
```
ant = 
  cassegrain with properties:

        Exciter: [1x1 hornConical]
         Radius: [0.3175 0.0330]
    FocalLength: [0.2536 0.1416]
           Tilt: 0
       TiltAxis: [1 0 0]
           Load: [1x1 lumpedElement]
```

```
show(ant)
```

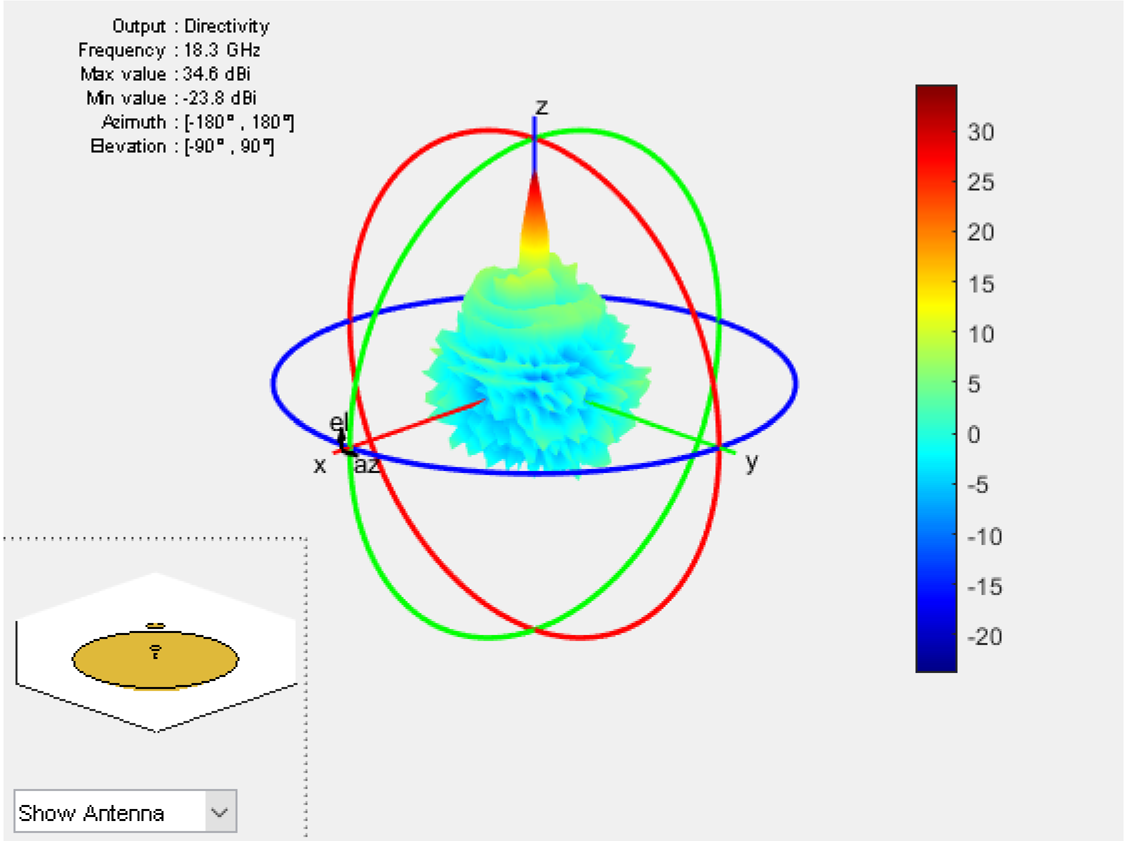


Plot the radiation pattern of the antenna at 18.3 GHz.

```
mesh(ant,'maxEdgeLength',14e-3)
```



```
figure;
pattern(ant,18.3e9)
```



```
   Output : Directivity
Frequency : 18.3 GHz
Max value : 34.6 dBi
Min value : -23.8 dBi
  Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

More About

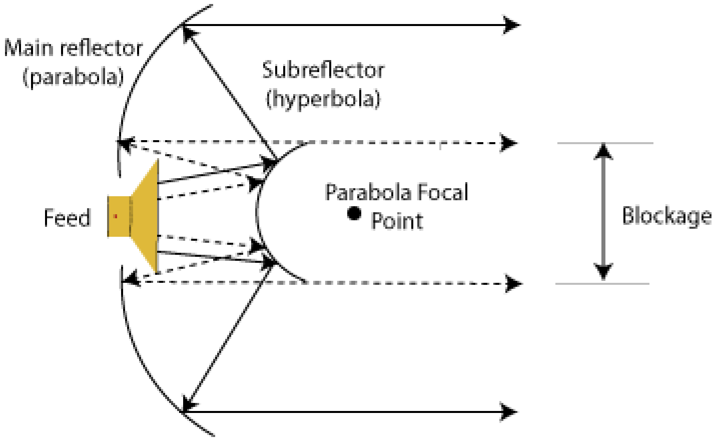
Parabolic Reflector Antennas

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.

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

`gregorian` | `hornConical` | `reflectorParabolic`

Topics
“Rotate Antennas and Arrays”

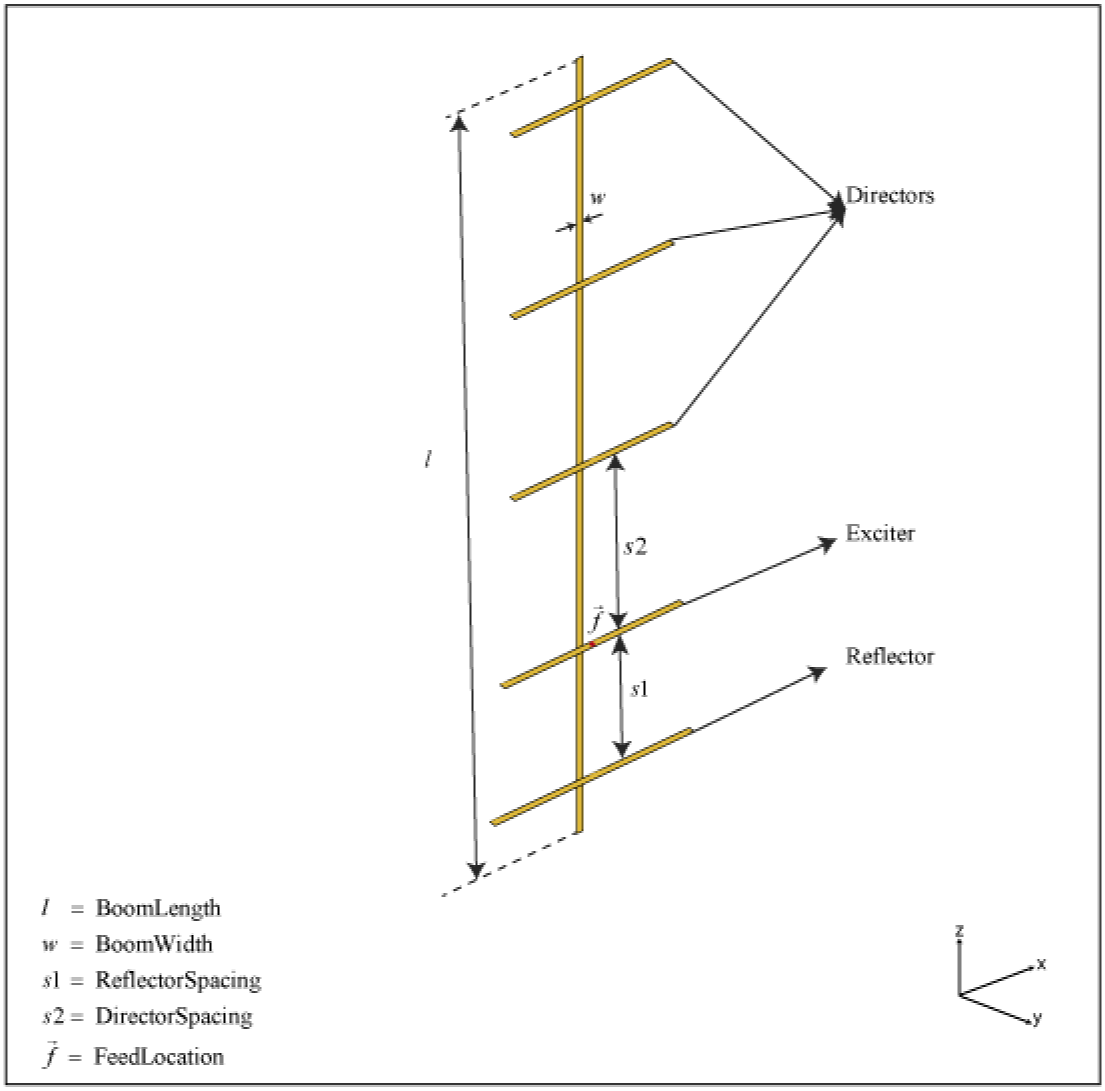
Introduced in R2019b

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.

Output Arguments

`ant` — Yagi-Uda custom array antenna
`quadCustom` object

Yagi-Uda custom array antenna, returned as a `quadCustom` object.

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`

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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch

axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

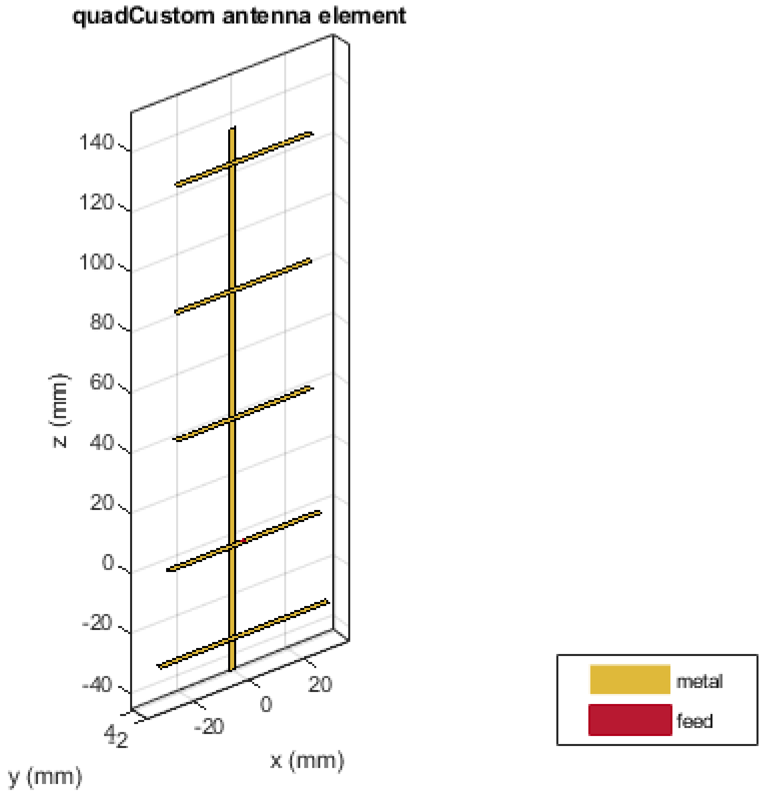
Default Custom Yagi-Uda Array Antenna (quadCustom) and Radiation Pattern

Create and view a custom Yagi-Uda array antenna.

```
ant = quadCustom
```

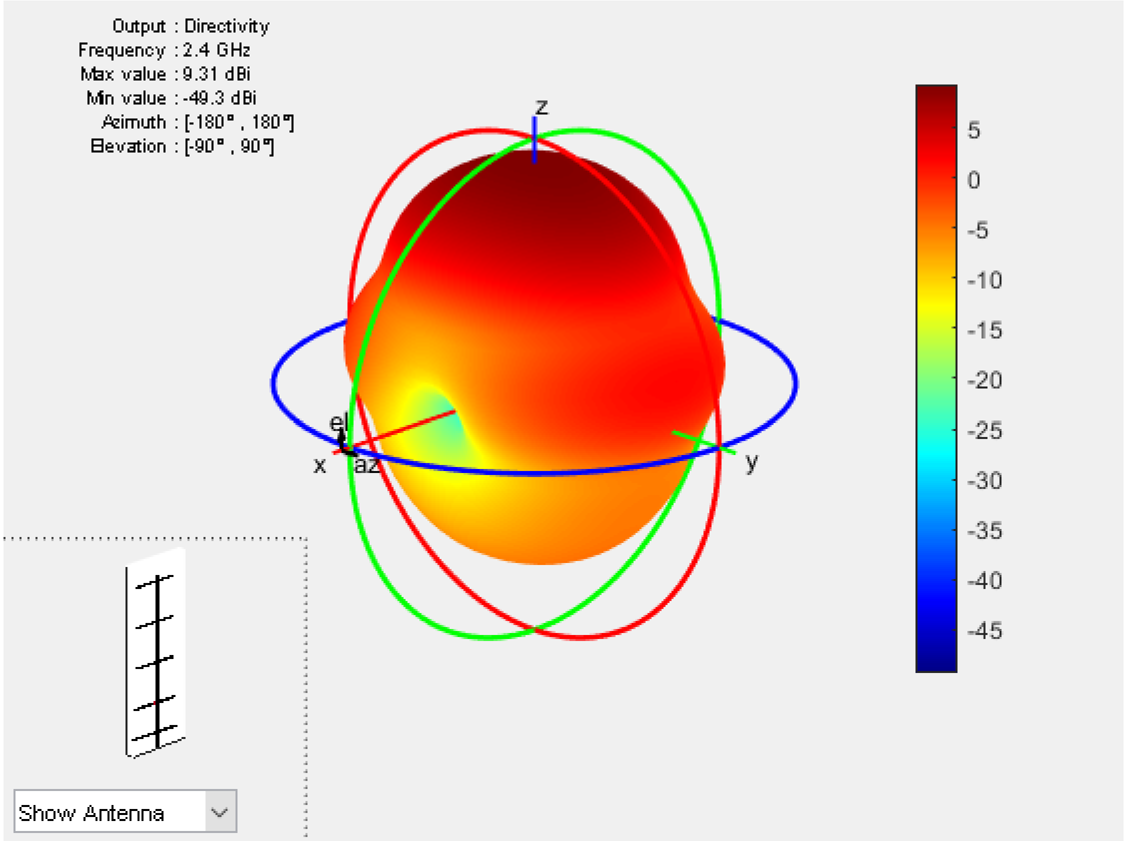
```
ant = 
  quadCustom with properties:

             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]
                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)
```



```
   Output : Directivity
Frequency : 2.4 GHz
Max value : 9.31 dBi
 Min value : -49.3 dBi
   Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

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
          Tilt: 90
      TiltAxis: [1 0 0]
          Load: [1x1 lumpedElement]
```

```
 ant.TiltAxis = [0 1 0]
```

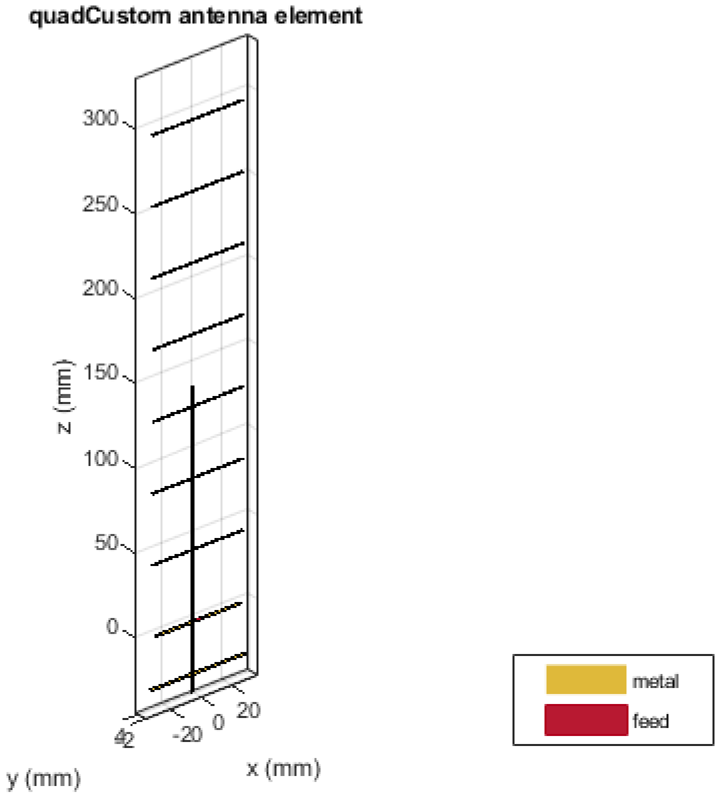
```
ant =
  dipole with properties:

        Length: 0.0587
         Width: 0.0012
```

```
      FeedOffset: 0
            Tilt: 90
        TiltAxis: [0 1 0]
            Load: [1x1 lumpedElement]
```

```
 quad_ant = quadCustom('Director',{ant,ant,ant,ant,ant,ant,ant})
```

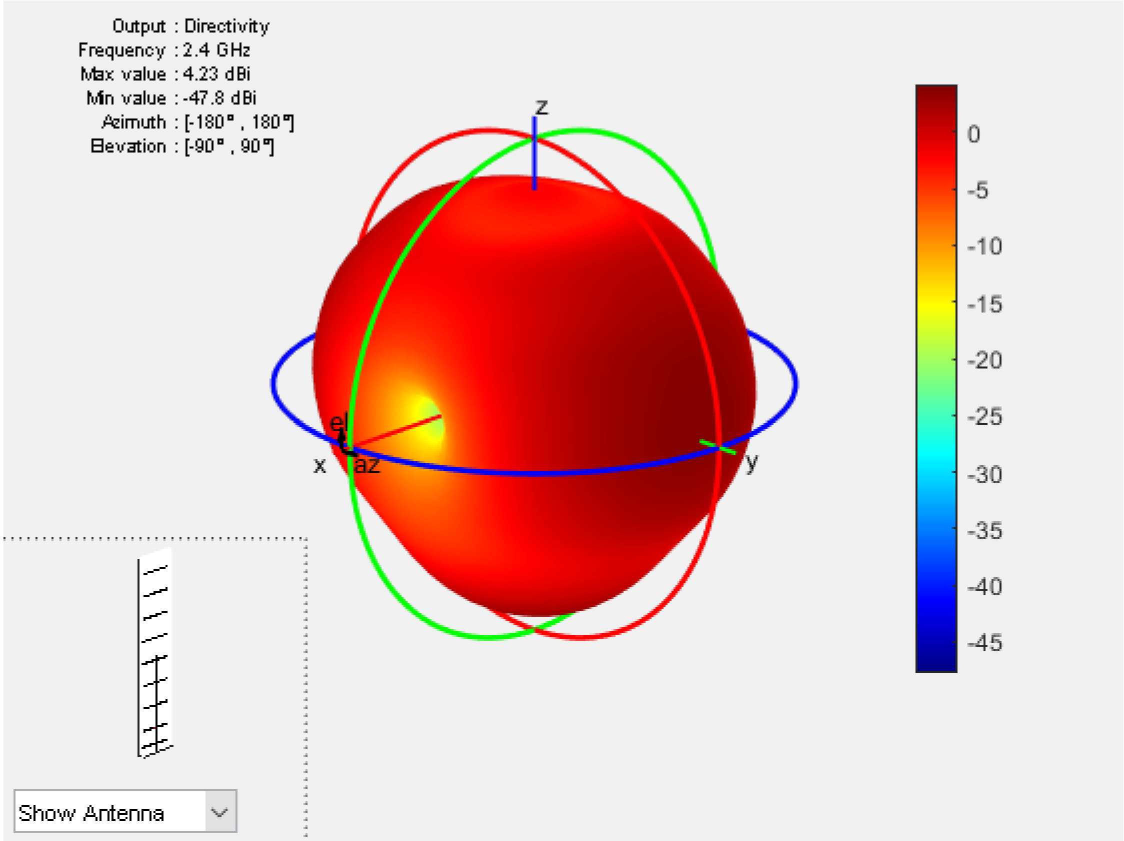
```
quad_ant = 
  quadCustom with properties:

             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]
                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 : 4.23 dBi
Min value : -47.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

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”

Introduced in R2019b

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 or array structure; display shape as filled patch
plot	Plot boundary of shape
mesh	Mesh properties of metal or 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

Examples

Create an Ellipse with Default Properties

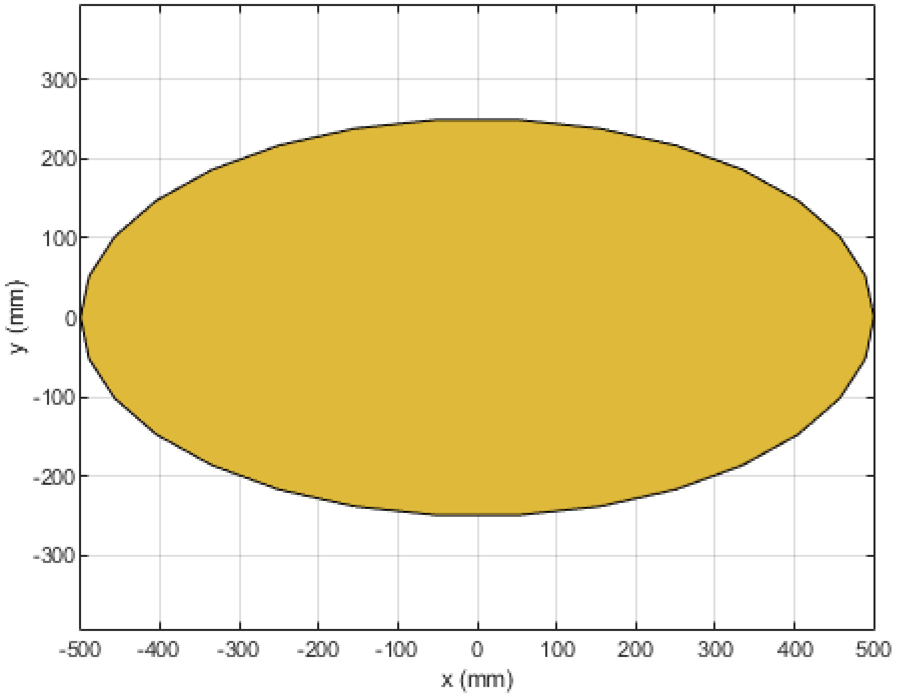
Create ellipse using `antenna.Ellipse.`

```
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.

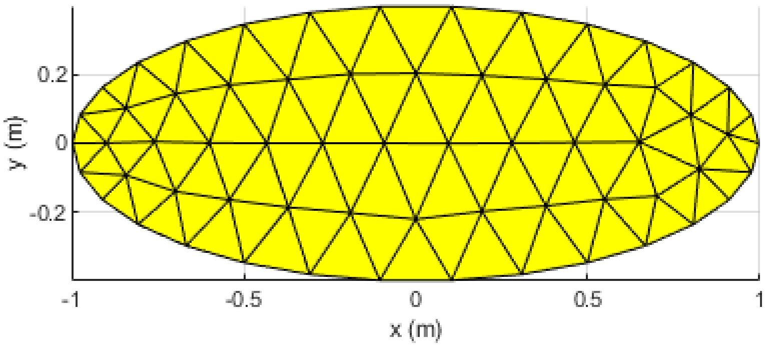
```
e2 = antenna.Ellipse('MajorAxis',2,'MinorAxis',0.8)
```

```
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.

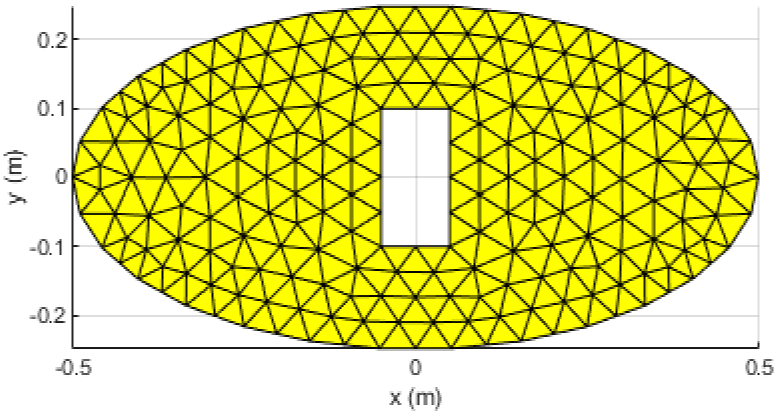
```
 r = antenna.Rectangle('Length',0.1,'Width',0.2);
```

Subtract the two shapes using the `minus` function.

```
 s = e3-r;
```

Mesh the subtracted shape with a maximum edge length of 1 m.

```
 mesh(s,1)
```



See Also

`antenna.Circle` | `antenna.Polygon` | `antenna.Rectangle`

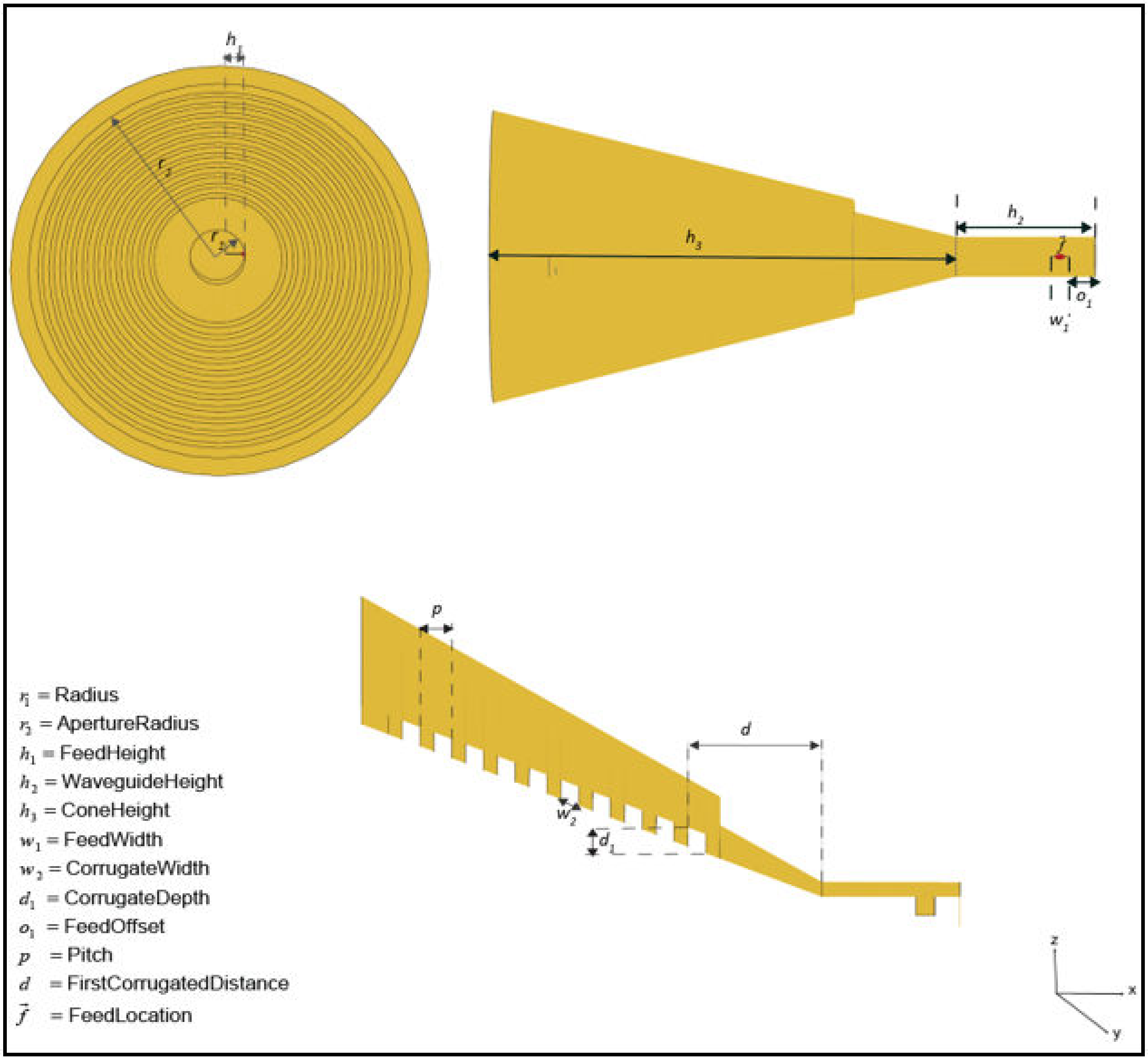
Introduced in R2020a

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 of 9.54 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

Load — Lumped elements

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

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: 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 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 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 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'`

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

Object Functions

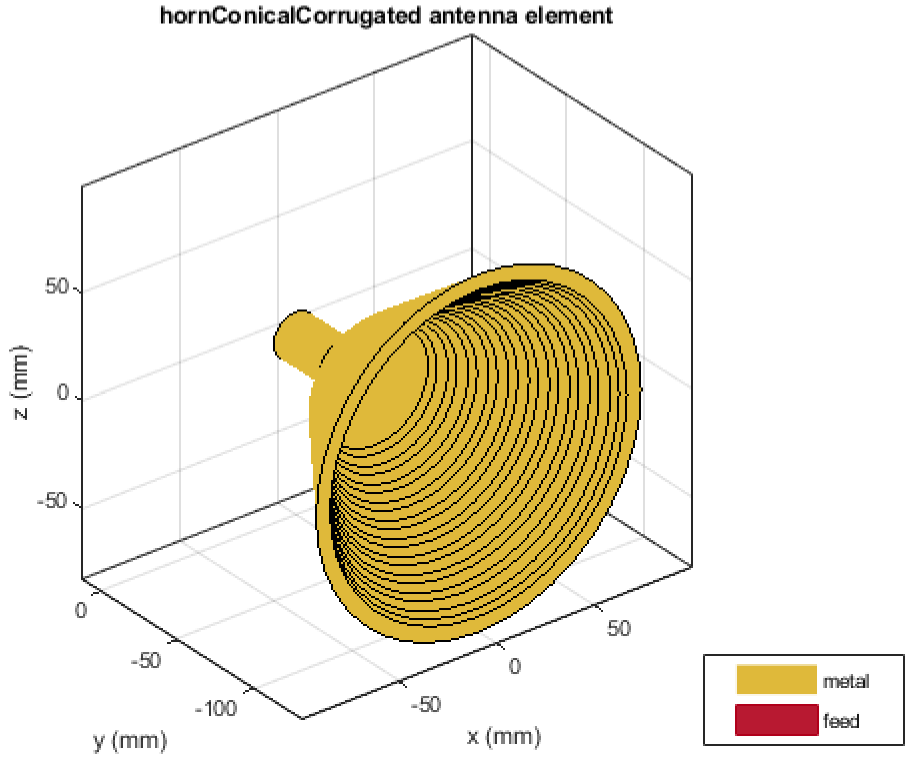
show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

Create a Conical Corrugated-Horn Antenna

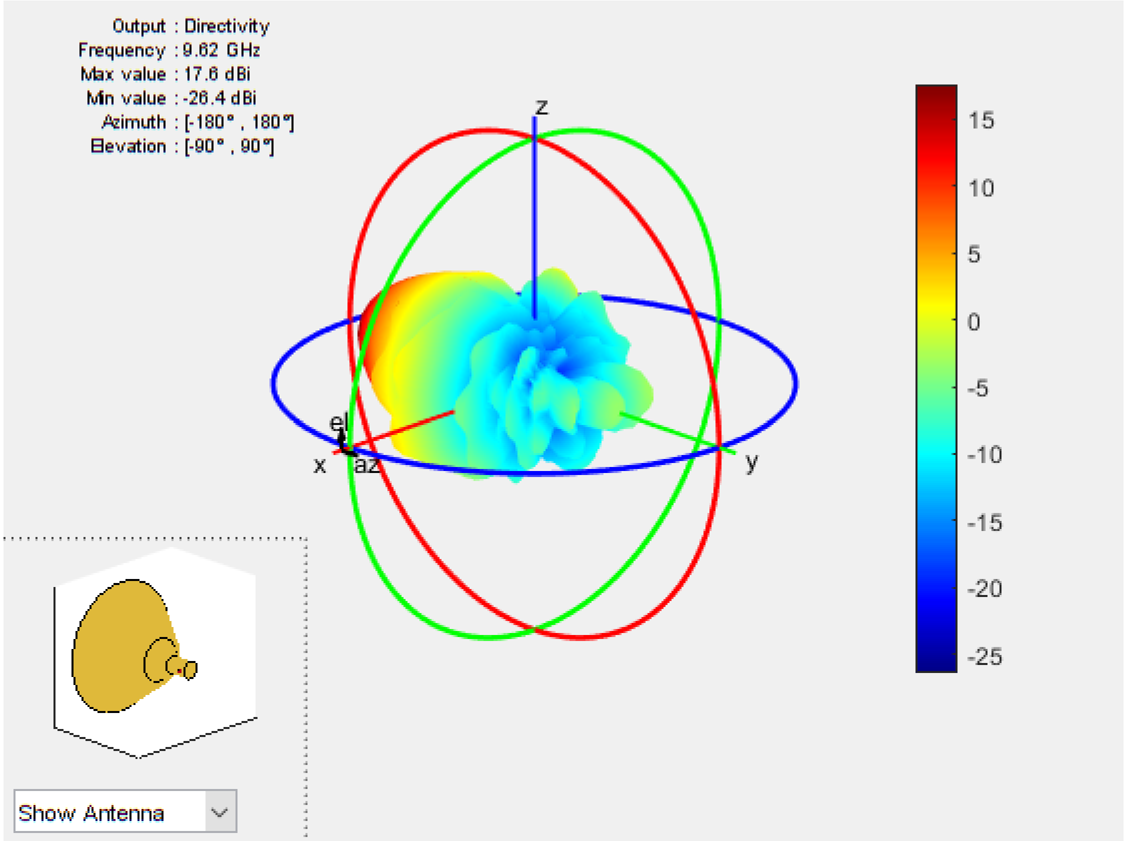
Create a conical corrugated-horn antenna object with the cone height set to 0.09 m

```
ant = hornConicalCorrugated('ConeHeight',0.09);
show(ant)
```



Plot the radiation pattern of the antenna at 9.62 GHz.

```
figure
pattern(ant,9.62e9)
```



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

`cavityCircular` | `horn` | `hornConical` | `hornangle2size` | `waveguide`

Topics

“Rotate Antennas and Arrays”

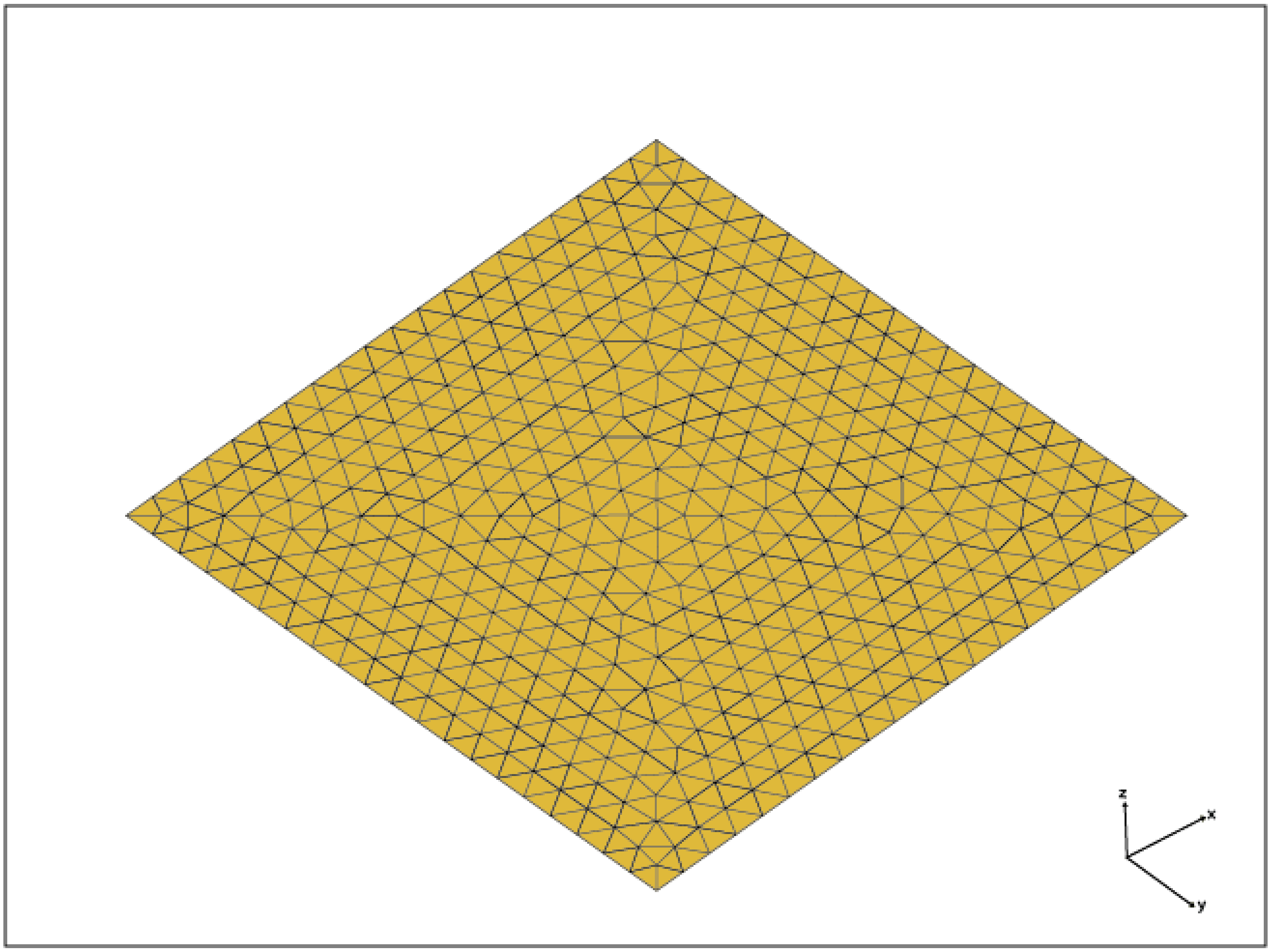
Introduced in R2020a

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

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.

Example: `'FeedLocation', [0 0.2 0]`

Data Types: `three-element real vector`

`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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
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 metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
impedance	Input impedance of antenna; scan impedance of array

mesh	Mesh properties of metal or 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	S-parameter object
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)
```

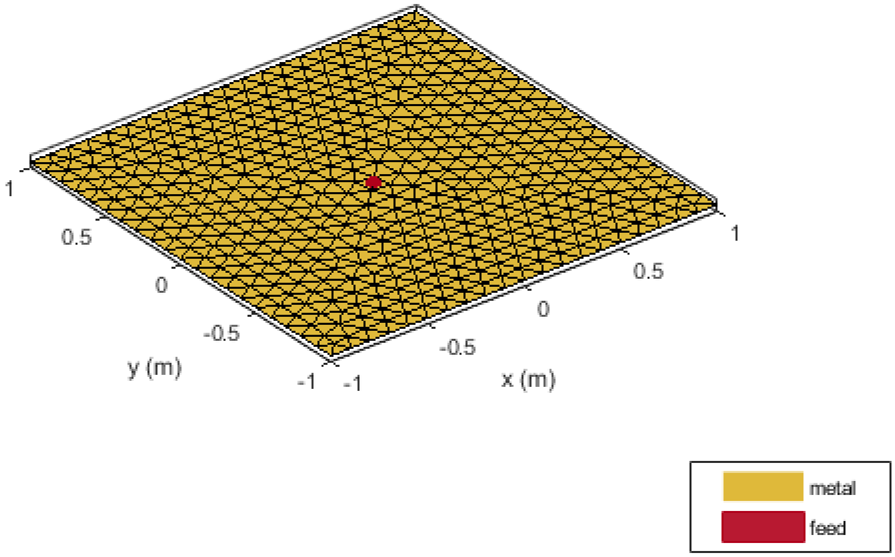
```
Z = 0.0287 + 34.3704i
```

```
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]
```

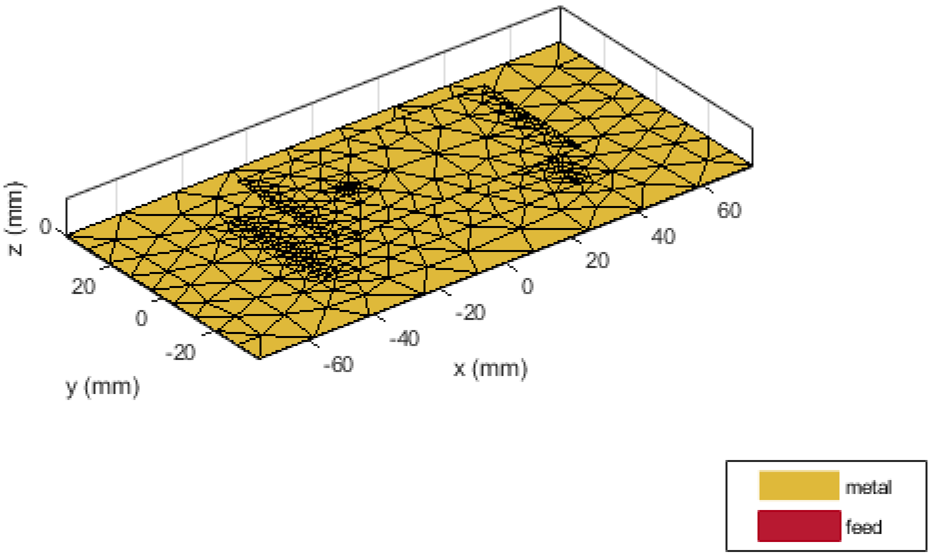
```
ant.FileName ='patchMicrostrip_ColumnFeed.stl'
```

```
ant = 
  customAntennaStl with properties:

         FileName: 'patchMicrostrip_ColumnFeed.stl'
```

```
             Units: 'm'
      FeedLocation: []
    AmplitudeTaper: 1
        PhaseShift: 0
     UseFileAsMesh: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

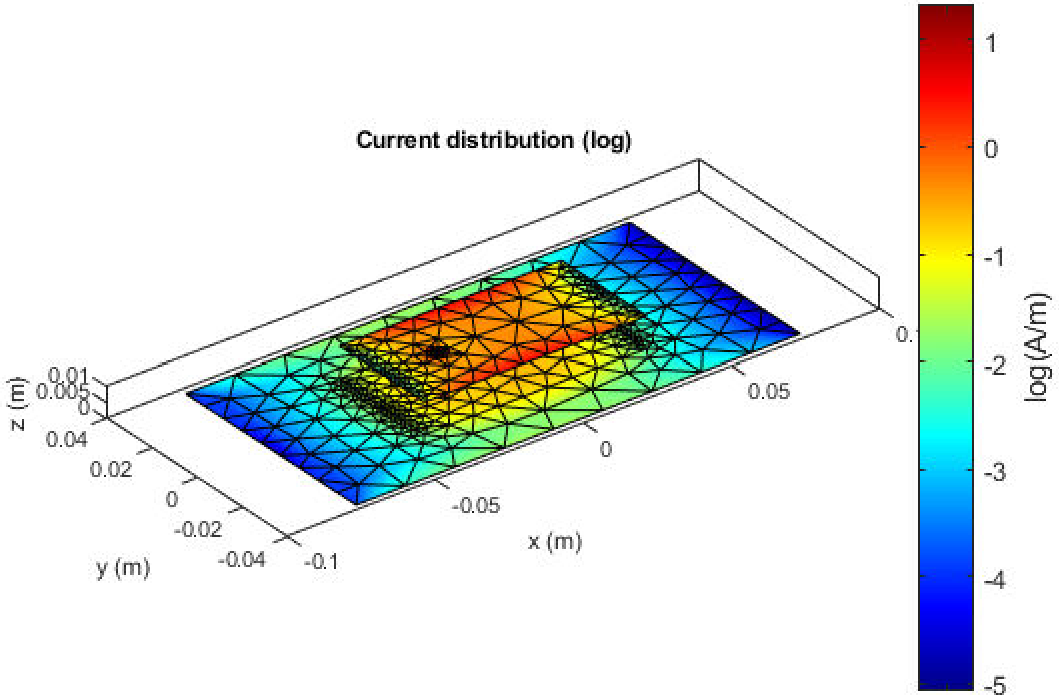
Specify FeedLocation and NumEdges in the createFeed function. The edges are selected based on distance between feed location and midpoints of the edges. Edges can be single feed or a closed polygon.

```
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.

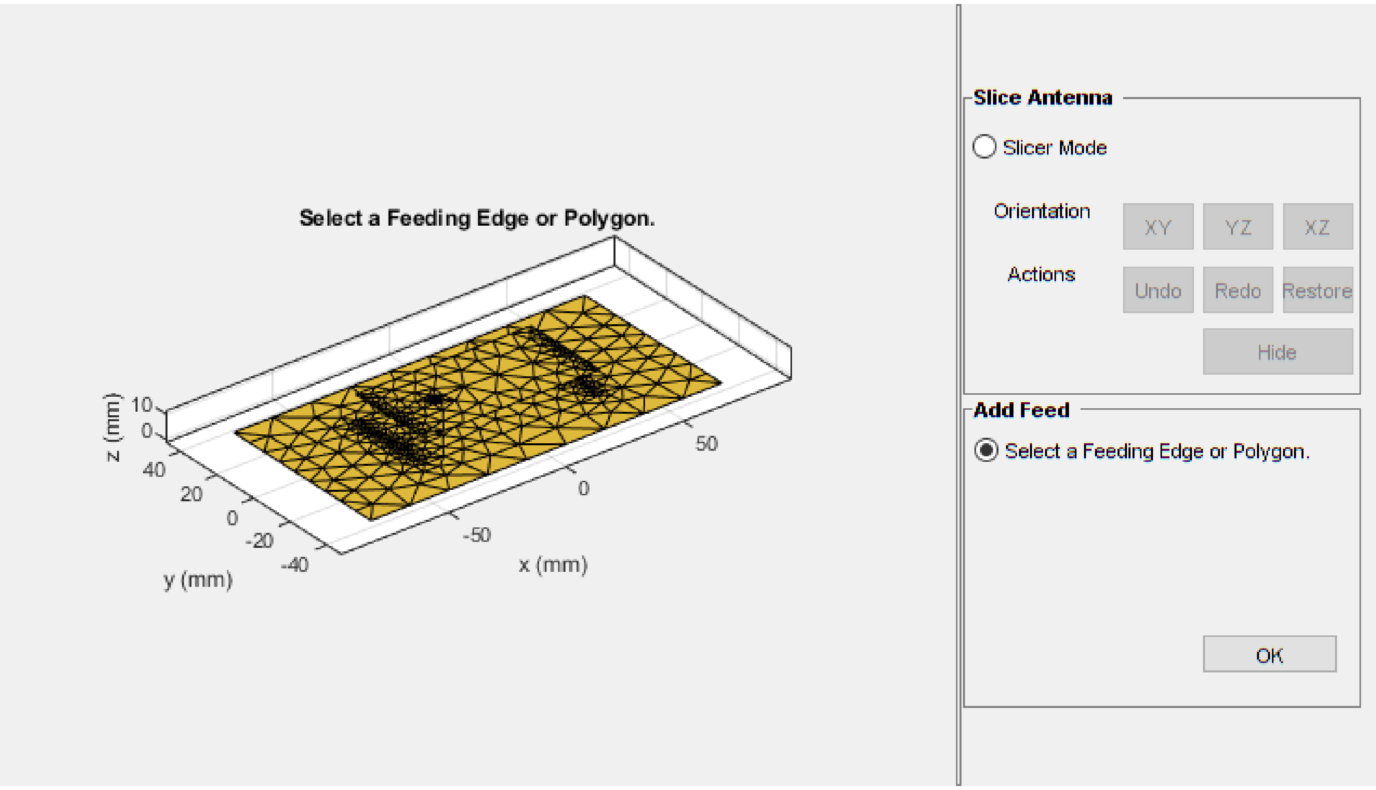
```
ant= customAntennaStl;
```

Import the STL files.

```
ant.FileName = 'patchMicrostrip_ColumnFeed.stl';
```

Create the antenna feed using UI figure window.

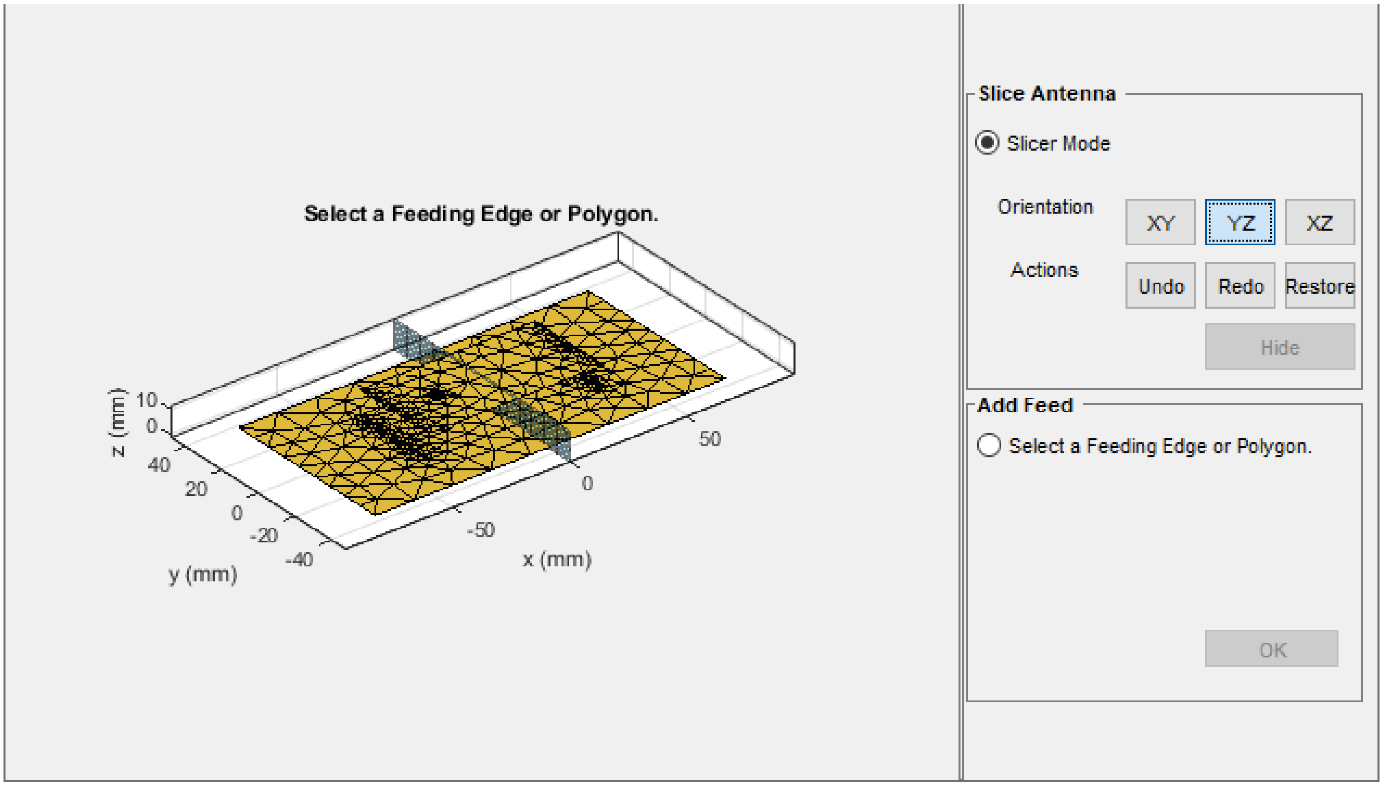
```
createFeed(ant);
```



○ Slicer Mode

● Select a Feeding Edge or Polygon.

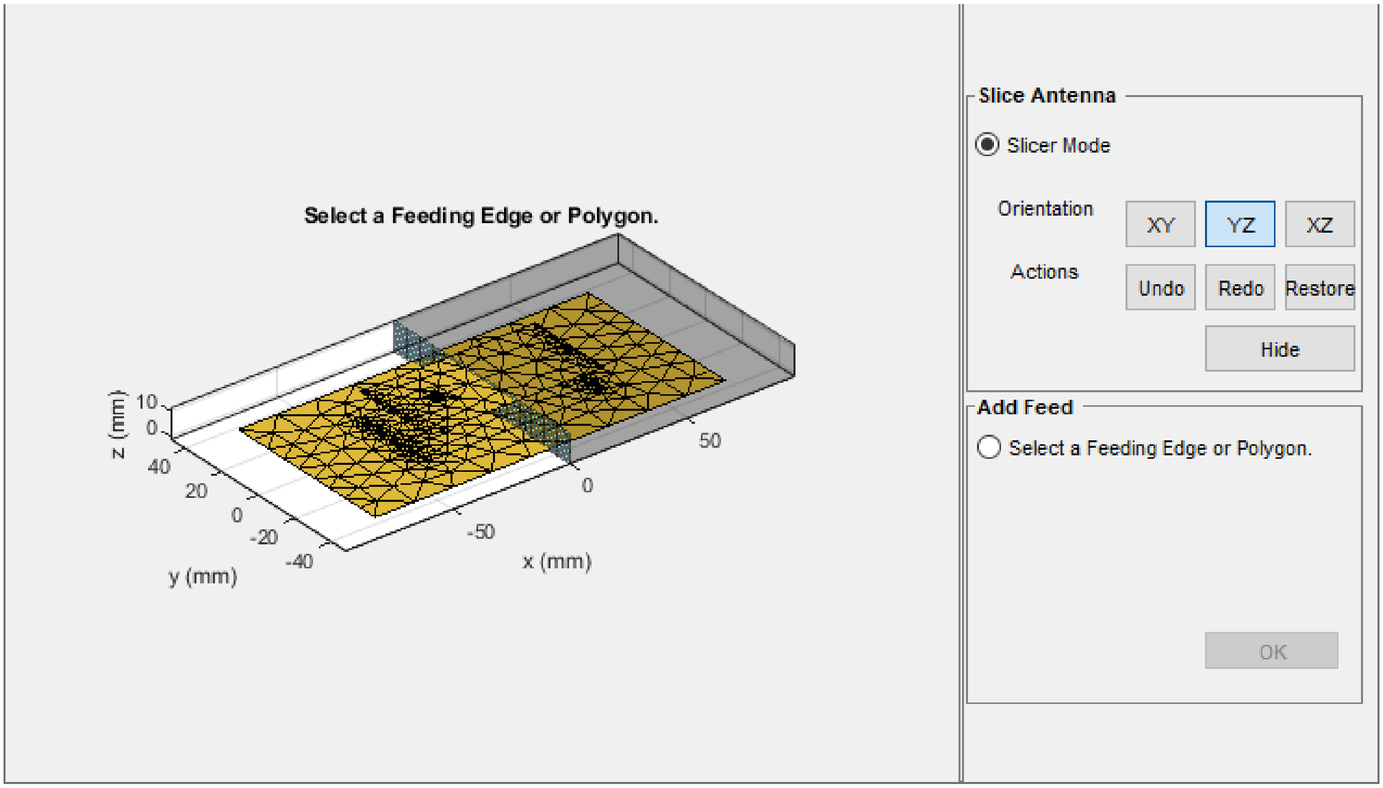
The UI figure window consists of two panes, the **Slice Antenna** panel and the **Add Feed** pane.



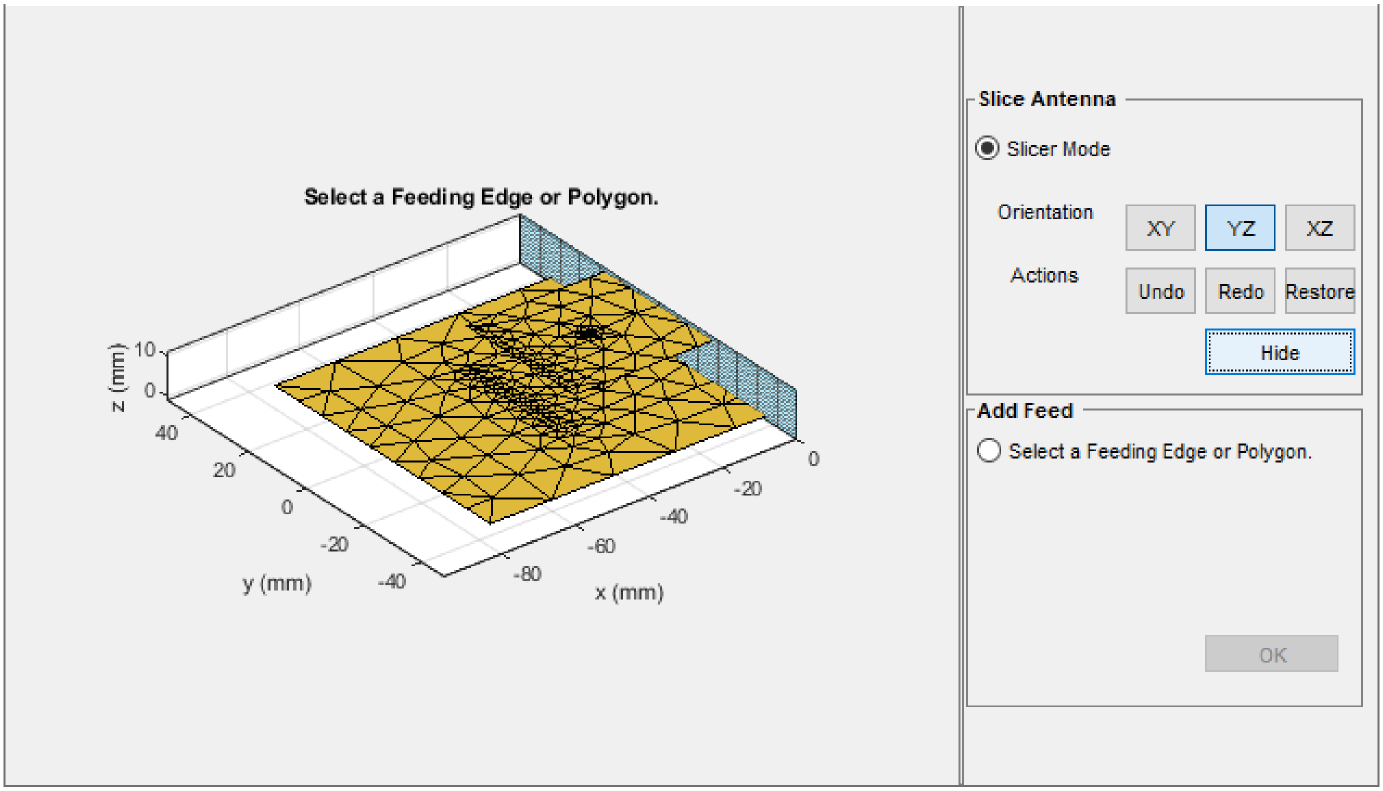
● Slicer Mode

○ Select a Feeding Edge or Polygon.

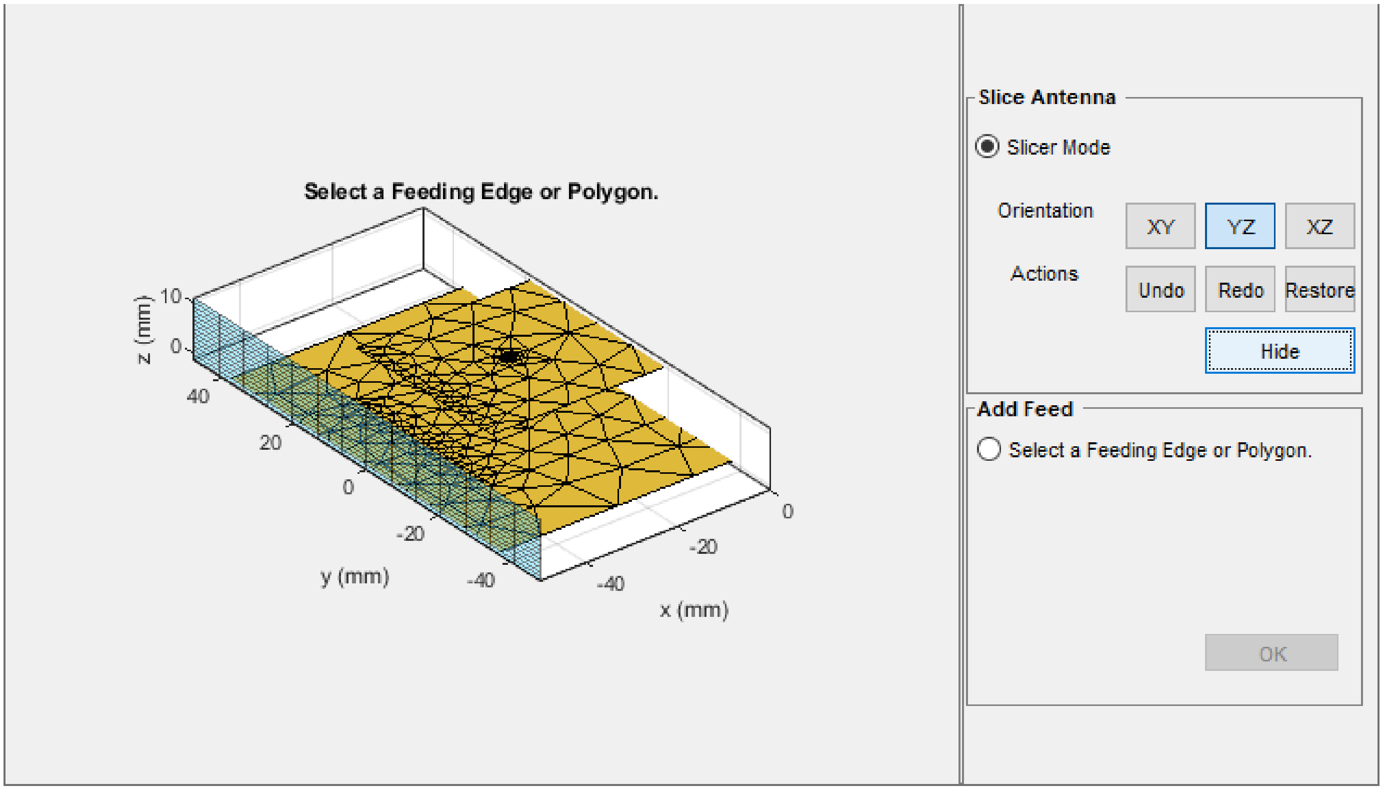
Click the **Slicer Mode,** then click **YZ** to select that as the plane along which to slice your antenna.



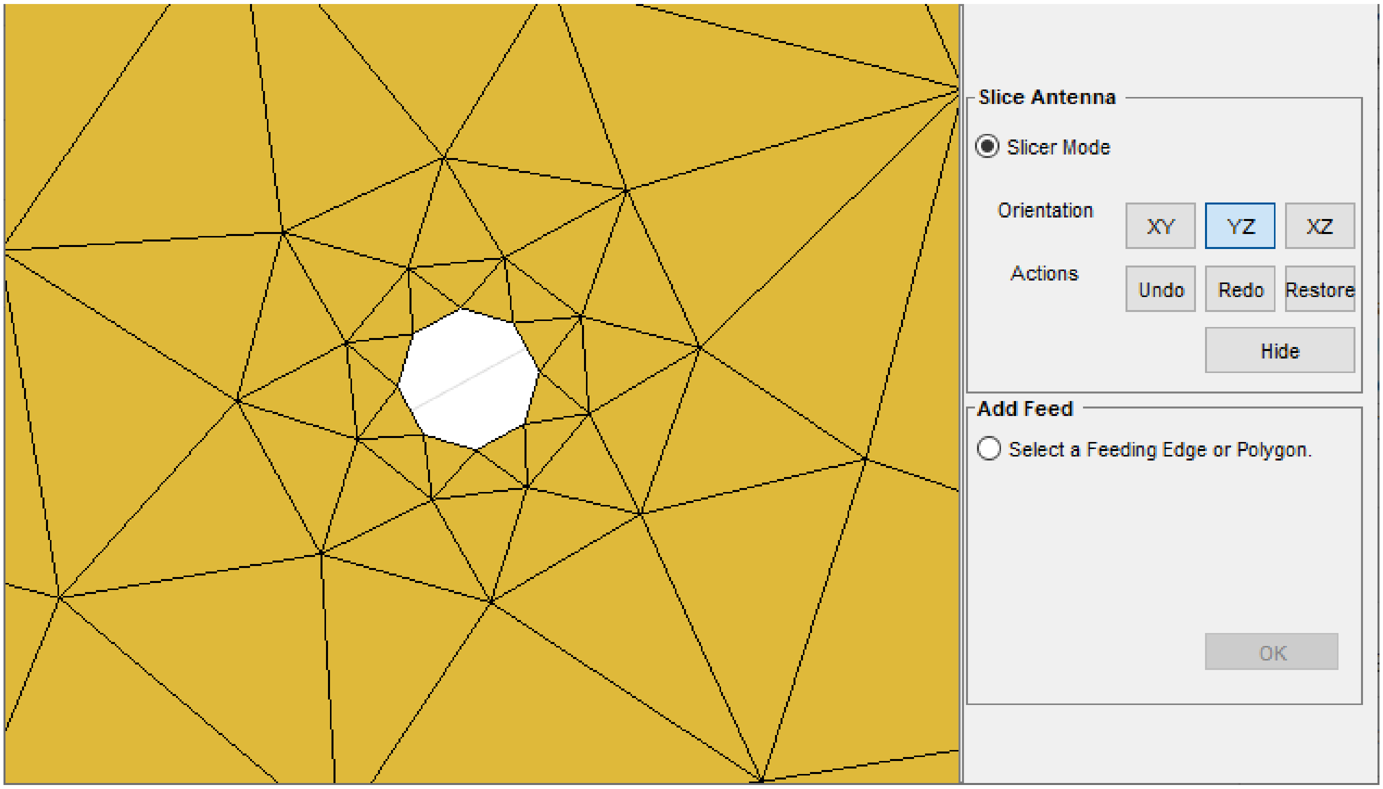
Select the region you want to hide and then click **Hide** to hide the selected region.



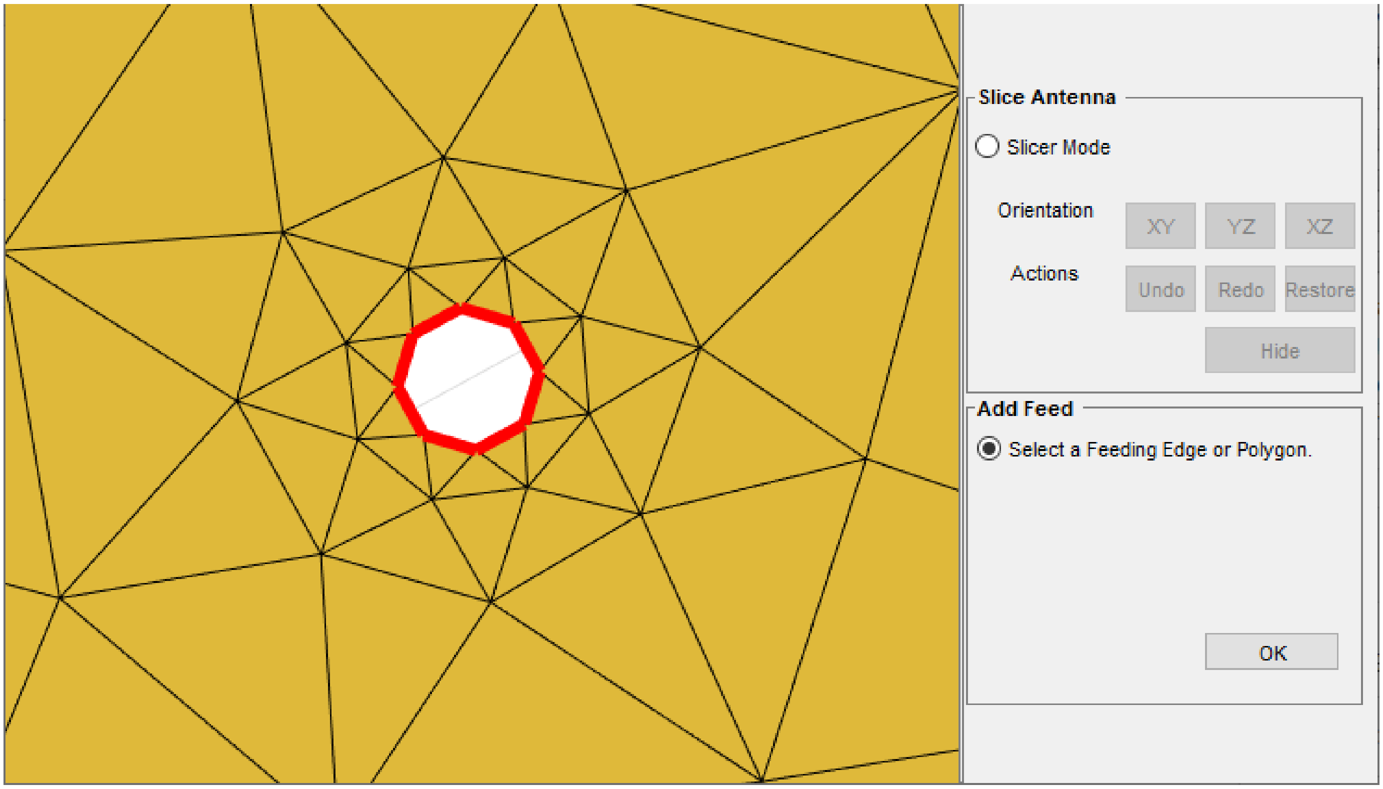
Repeat the process until you reach the region of interest.



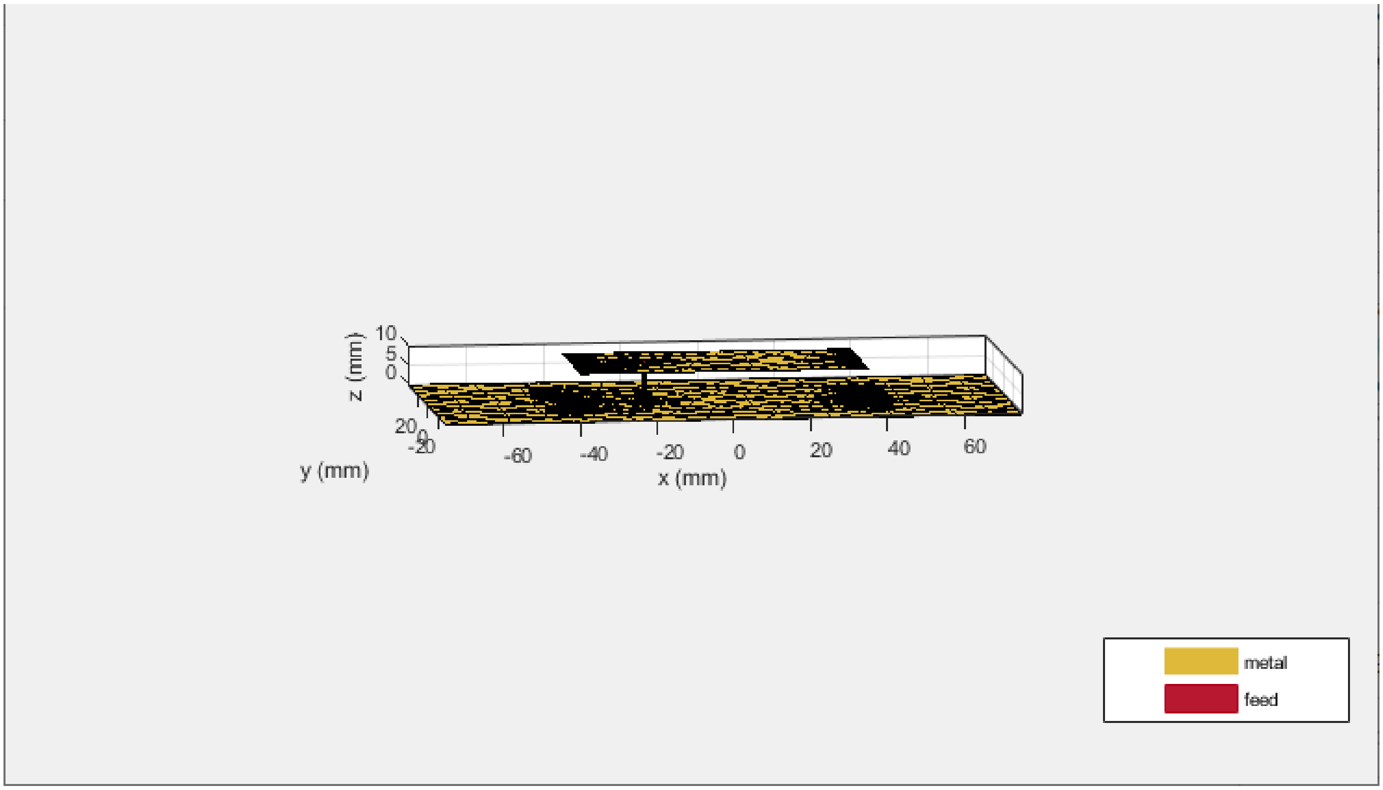
Select **Select a Feeding Edge or Polygon** under the **Add Feed** pane to select the desired feeding edge or feeding polygon.



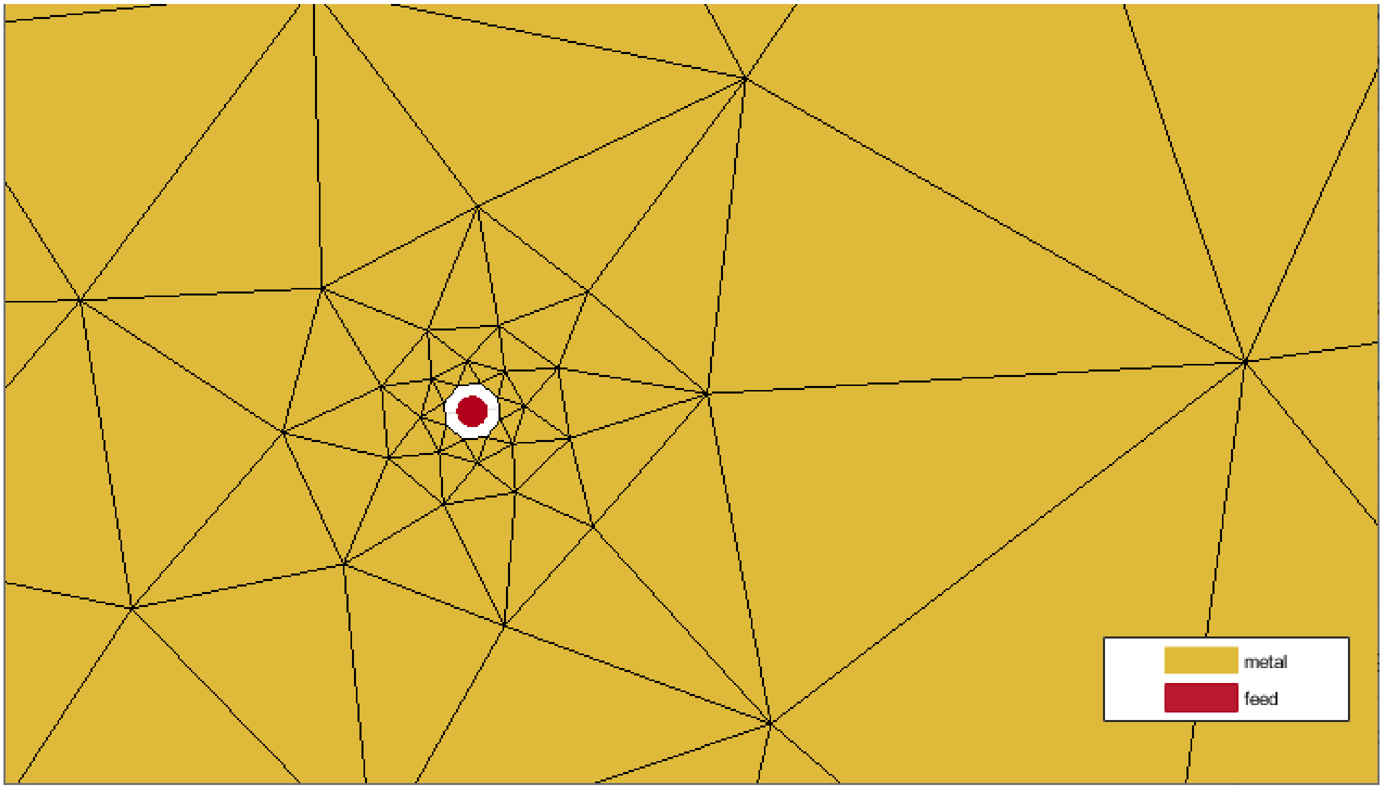
Select the edges of the column that forms a closed polygon. The selected edges must be connected to other edges, else the UI figure window will display an error.



Click **OK** to define the selected edges as feeding edges and the structure with the feed is displayed.



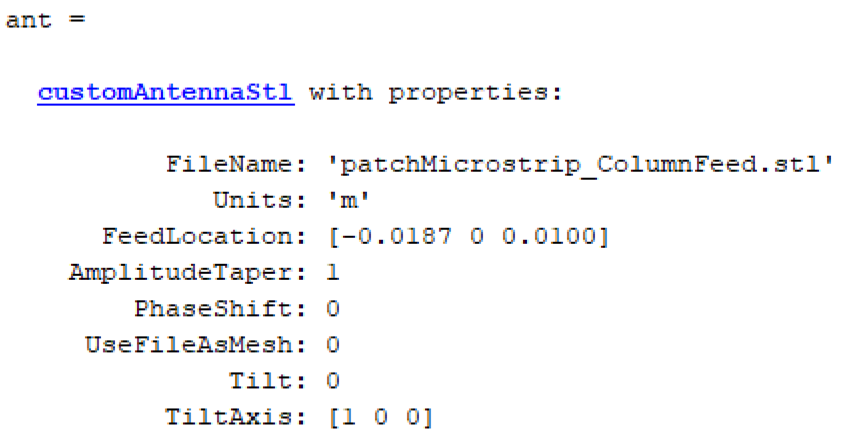
The `FeedLocation` is displayed.



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]
```

References

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

See Also

`customAntennaGeometry` | `customAntennaMesh` | `platform`

Topics

"Rotate Antennas and Arrays"

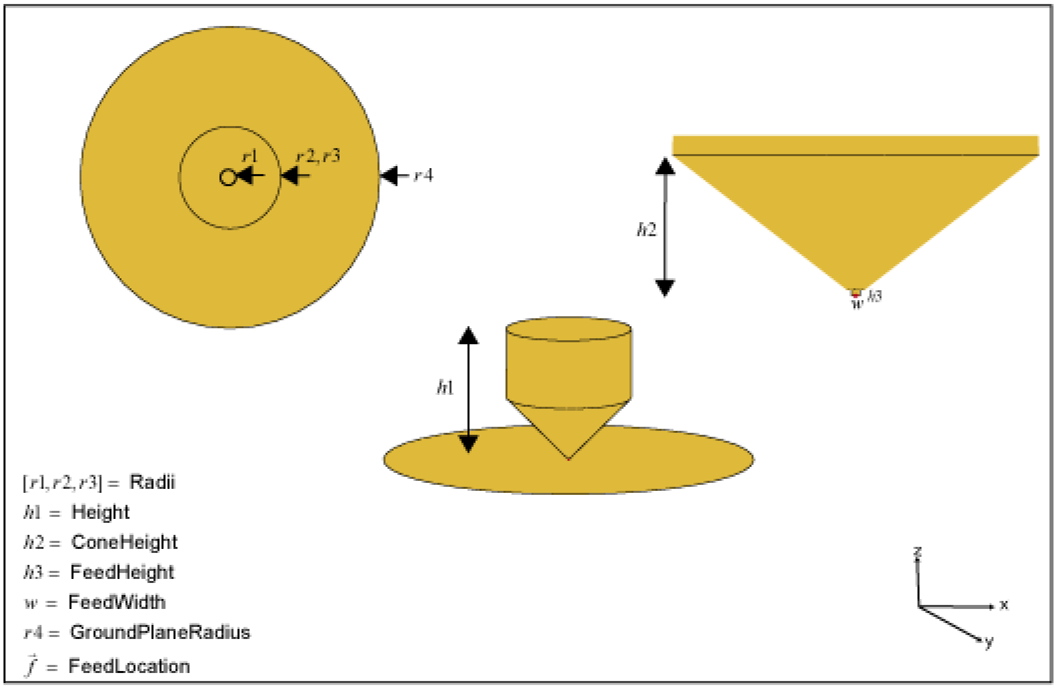
Introduced in R2020a

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 3.94 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.

Output Arguments

`ant` — Monocone antenna
`monocone` object

Monocone antenna, returned as a `monocone` object.

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` — Diameter of feed

`5.0000e-04` (default) | positive scalar

Diameter 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`

`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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

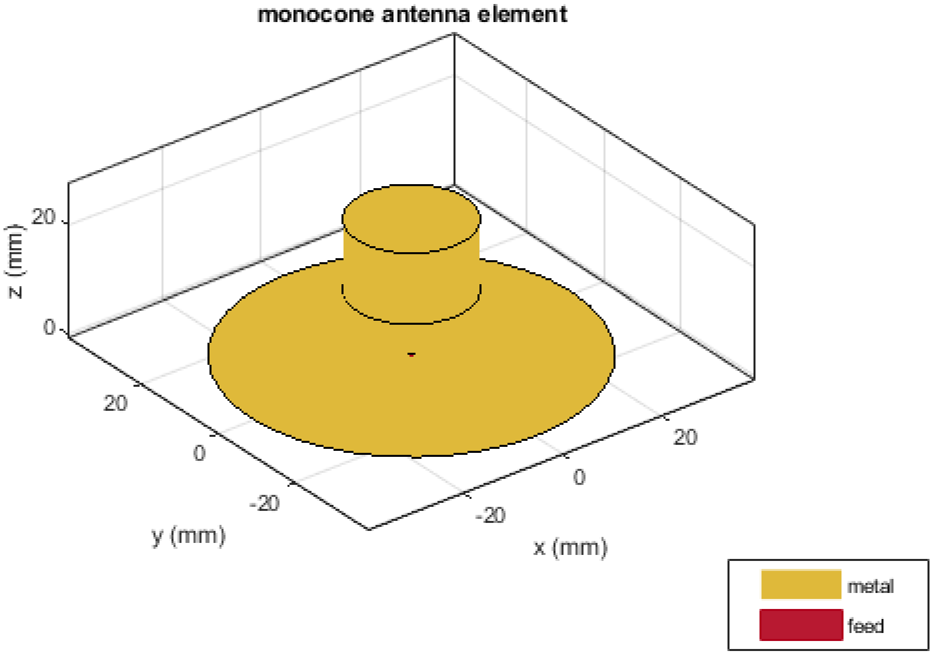
Examples

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

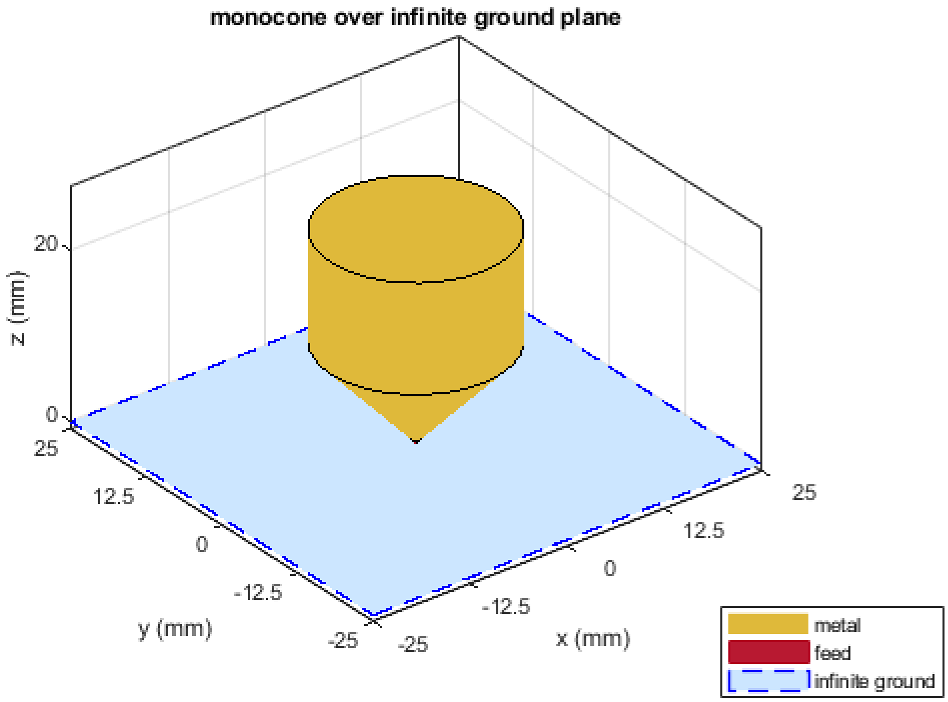
```
show(ant)
```



Monocone Antenna with Infinite Ground Plane

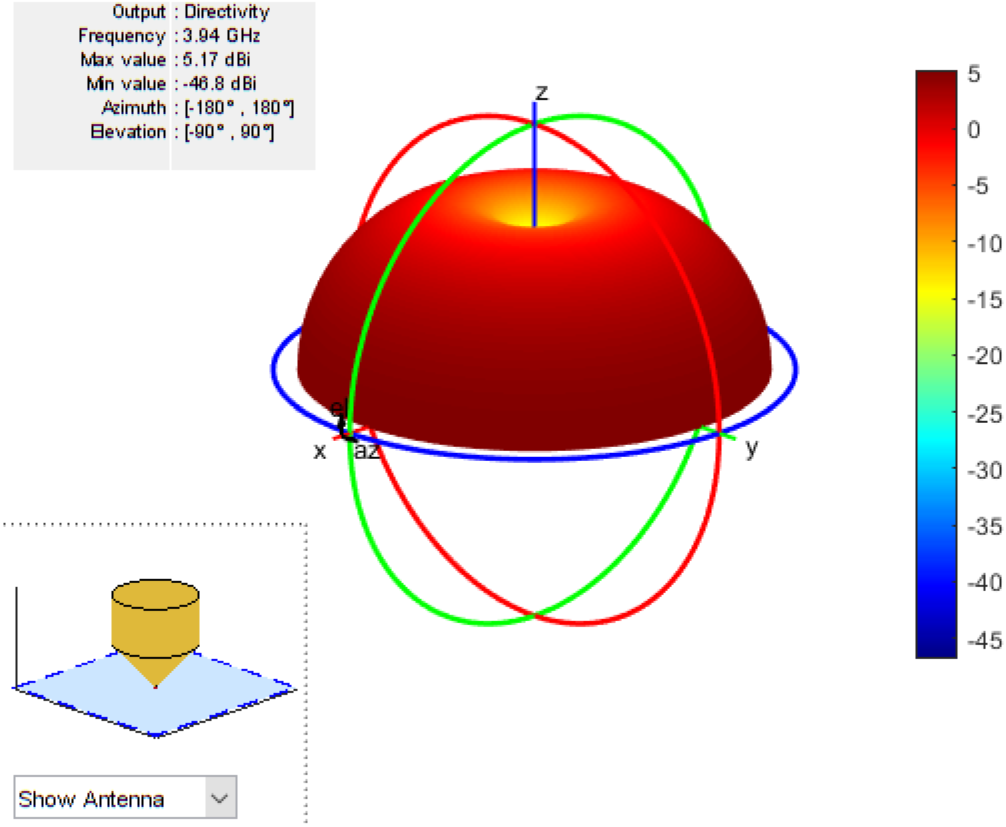
Create a monocone antenna with an infinite ground plane.

```
ant = monocone;
ant.GroundPlaneRadius = inf;
show(ant)
```



Plot the radiation pattern of the monocone antenna for the given frequency.

```
pattern(ant,3.94e9)
```

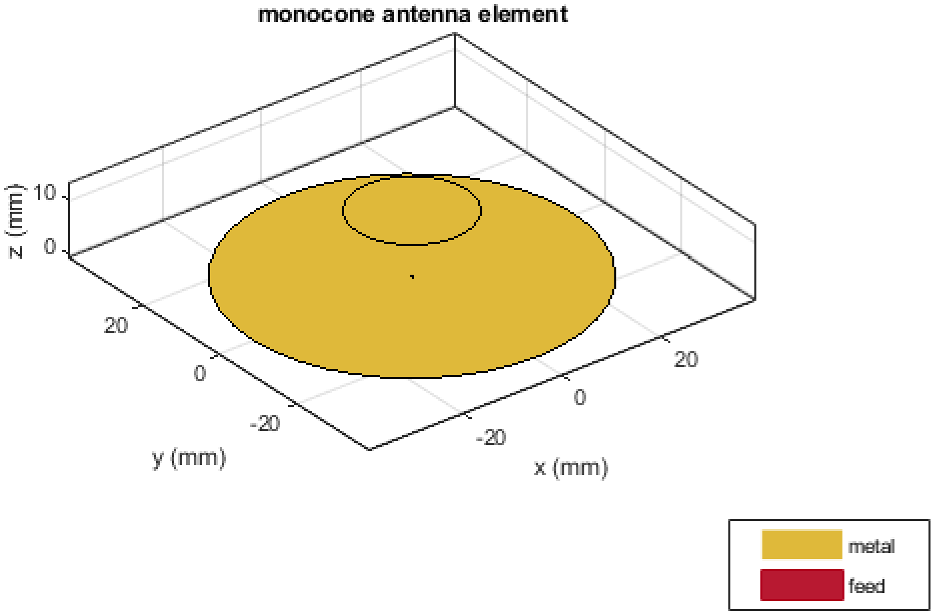


Output : Directivity
Frequency : 3.94 GHz
Max value : 5.17 dBi
Min value : -46.8 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Monocone Antenna Without A Cylinder

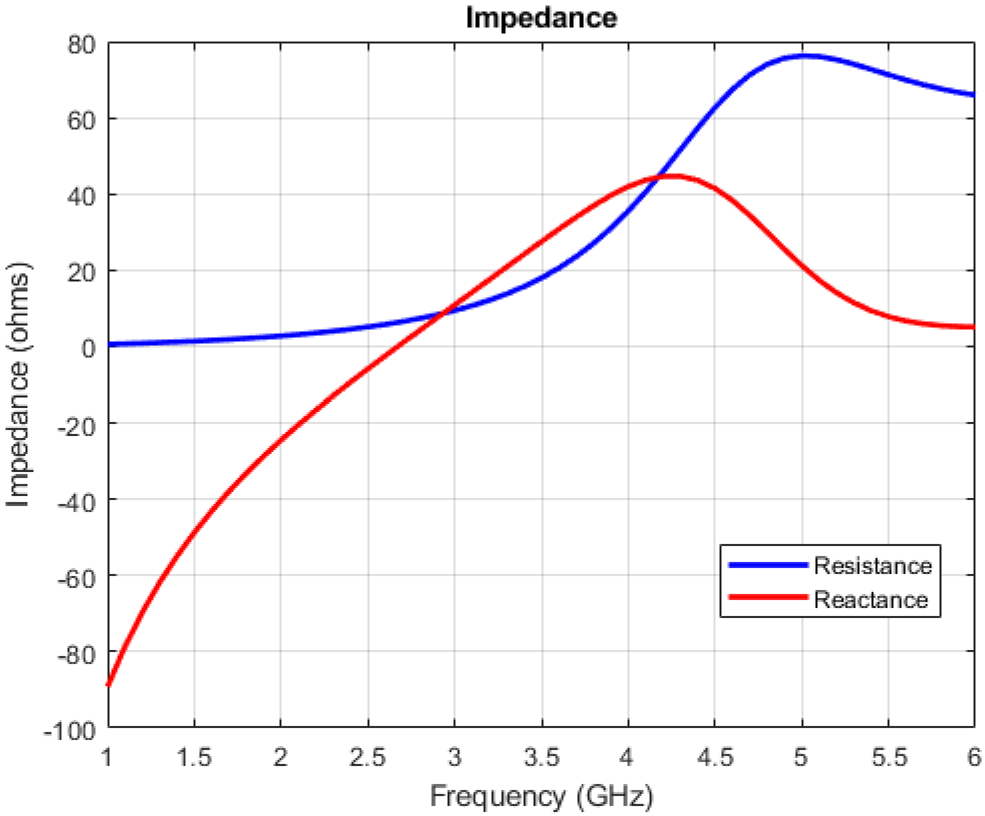
Create a classical monocone antenna by setting the total height of the antenna to equal the sum of cone height and feed height.

```
ant = monocone;
ant.Height = ant.ConeHeight+ant.FeedHeight;
show(ant)
```



Calculate antenna impedance over the given frequency span.

```
impedance(ant,(1e9:0.1e9:6e9))
```



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

`bicone` | `cavityCircular`

Topics

“Rotate Antennas and Arrays”

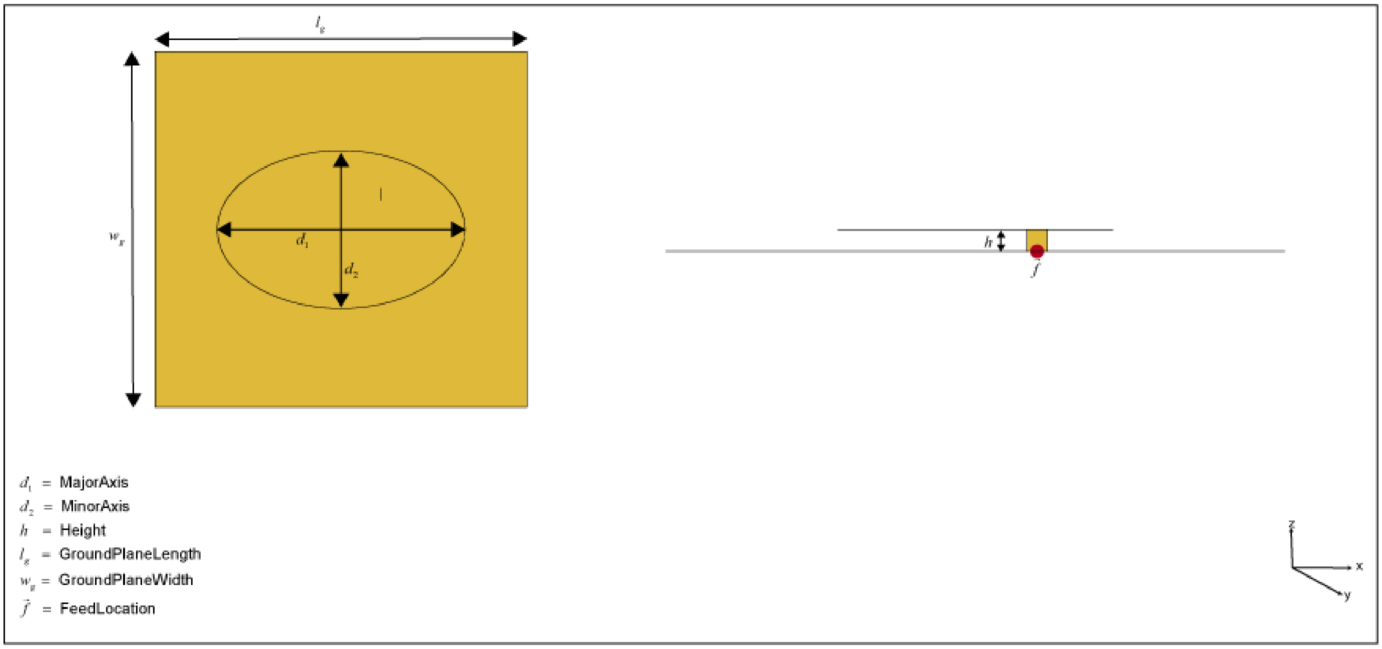
Introduced in R2020a

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.

Output Arguments

`ant` — Elliptical microstrip patch antenna
`patchMicrostripElliptical` object (default)

Elliptical microstrip patch antenna, returned as a `patchMicrostripElliptical` object.

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 handle

Type of dielectric material used as a substrate, specified as a dielectric material object handle. 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`

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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

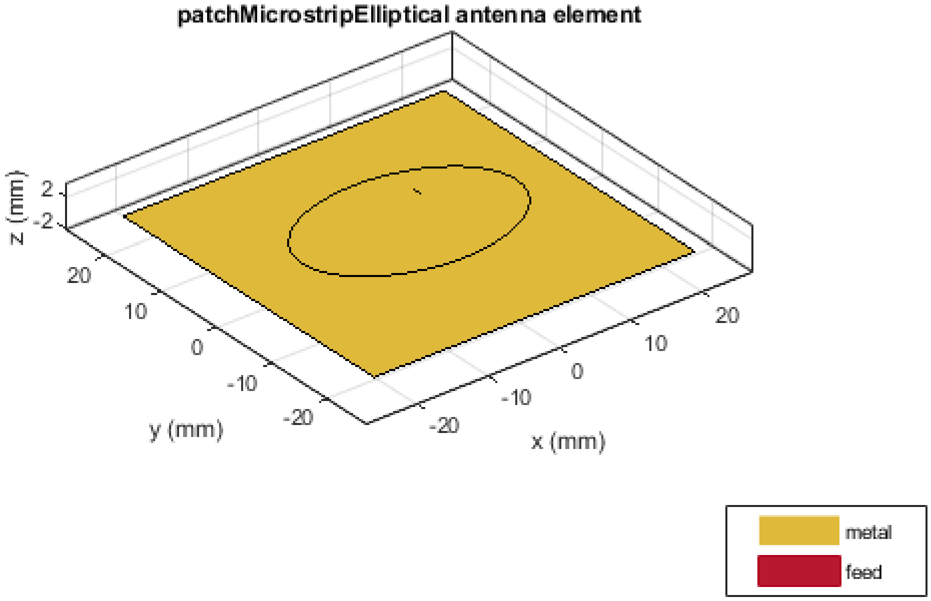
Default Elliptical Microstrip Patch Antenna

Create and view a default elliptical microstrip patch antenna.

```
ant = patchMicrostripElliptical
```

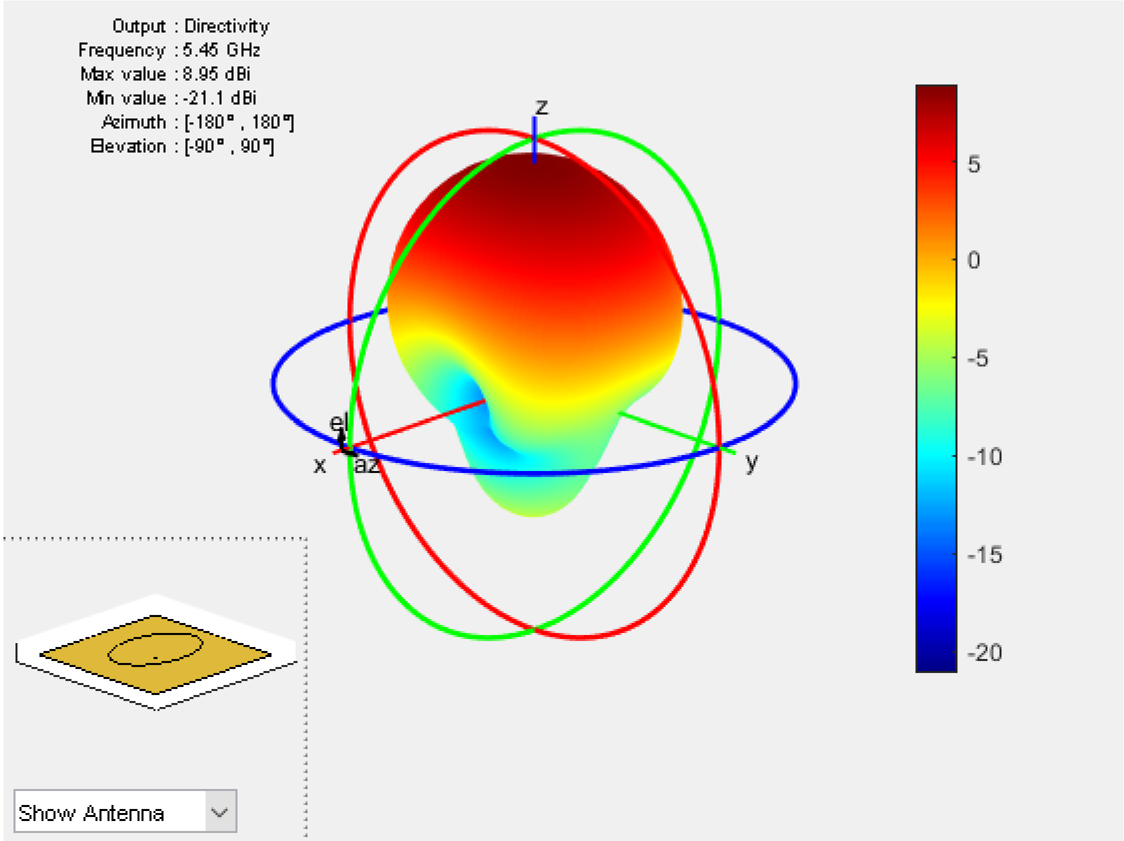
```
ant = 
  patchMicrostripElliptical with properties:

            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]
                 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 : 8.95 dBi
Min value : -21.1 dBi
Azimuth : [-180°, 180°]
Elevation : [-90°, 90°]

See Also

`patchMicrostrip` | `patchMicrostripCircular`

Topics

“ISM Band Patch Microstrip Antennas and Mutual Coupling Between different patches”
“Rotate Antennas and Arrays”

Introduced in R2020a

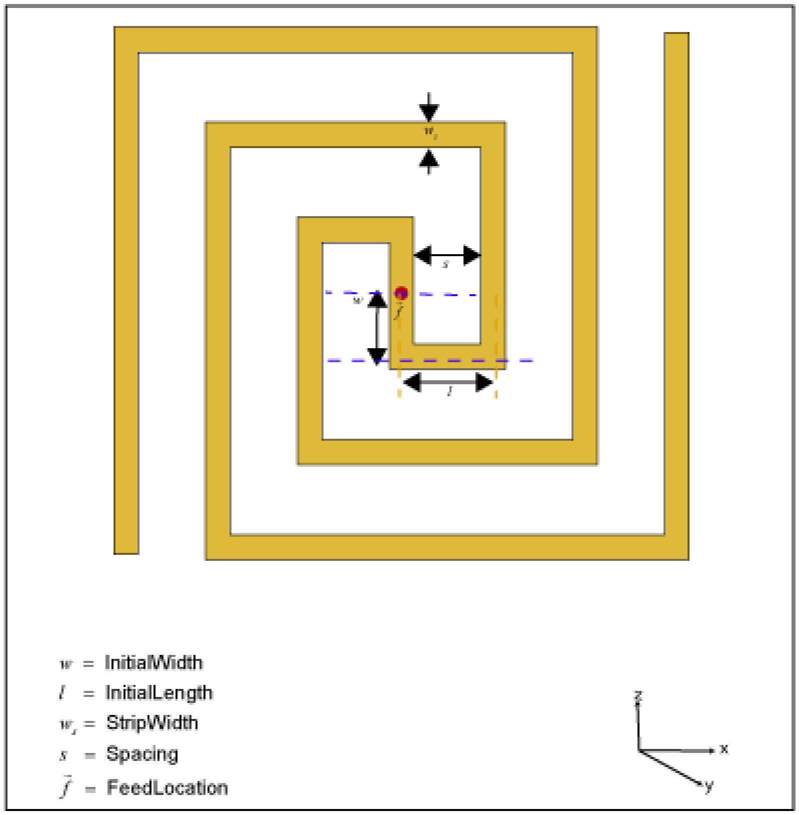
spiralRectangular

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.

Output Arguments

`ant` — Rectangular spiral antenna
`spiralRectangular` object (default)

Rectangular spiral antenna, returned as a `spiralRectangular` object.

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`

`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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

rectspirallength2turns	Calculate number of turns for specified arm length in rectangular spiral antenna

Examples

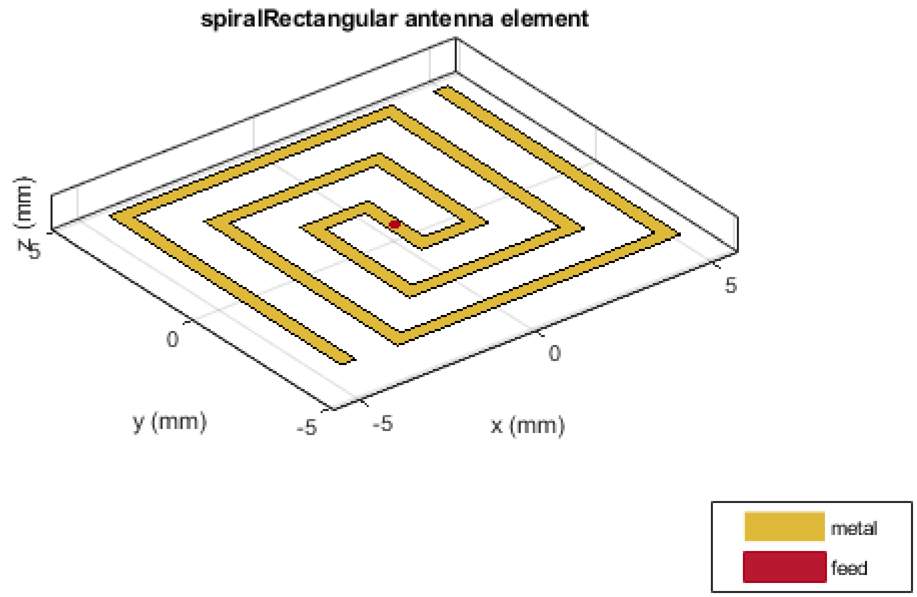
Default Rectangular Spiral Antenna

Create and view a default rectangular spiral antenna.

```
ant = spiralRectangular
```

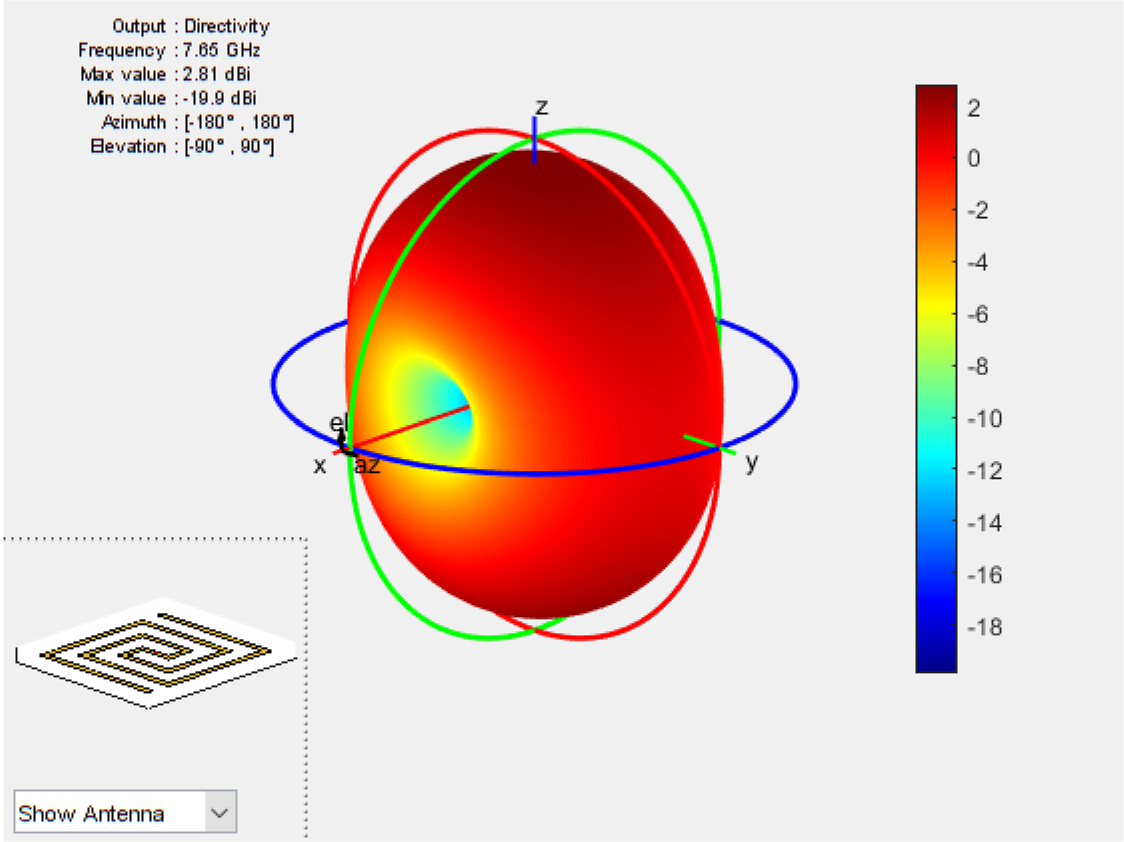
```
ant = 
  spiralRectangular with properties:

             NumArms: 2
            NumTurns: 1.5300
        InitialWidth: 0.0010
       InitialLength: 0.0015
           StripWidth: 4.0500e-04
             Spacing: 0.0011
    WindingDirection: 'CCW'
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1×1 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°]

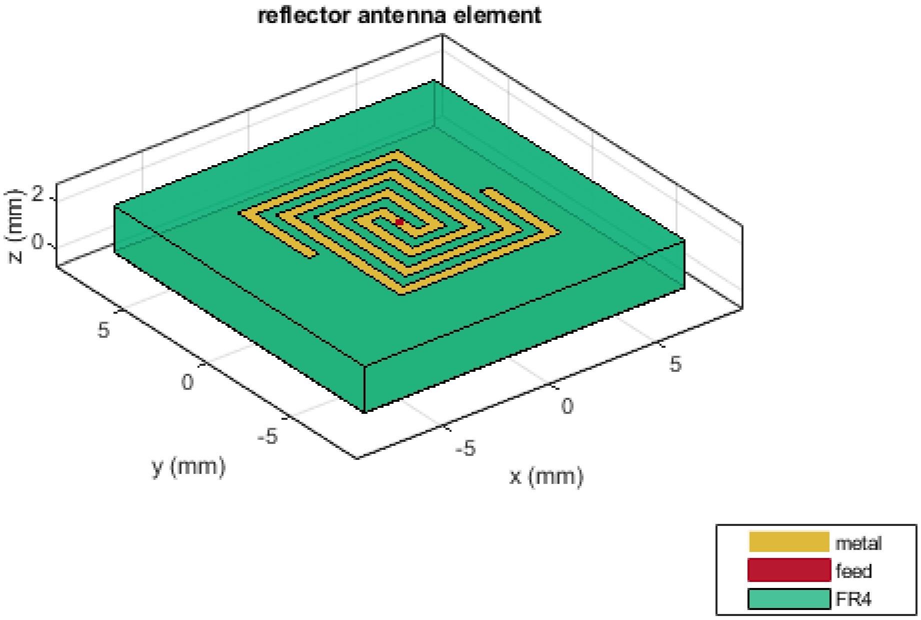
Radiation Pattern of Reflector Backed Rectangular Spiral

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);
```

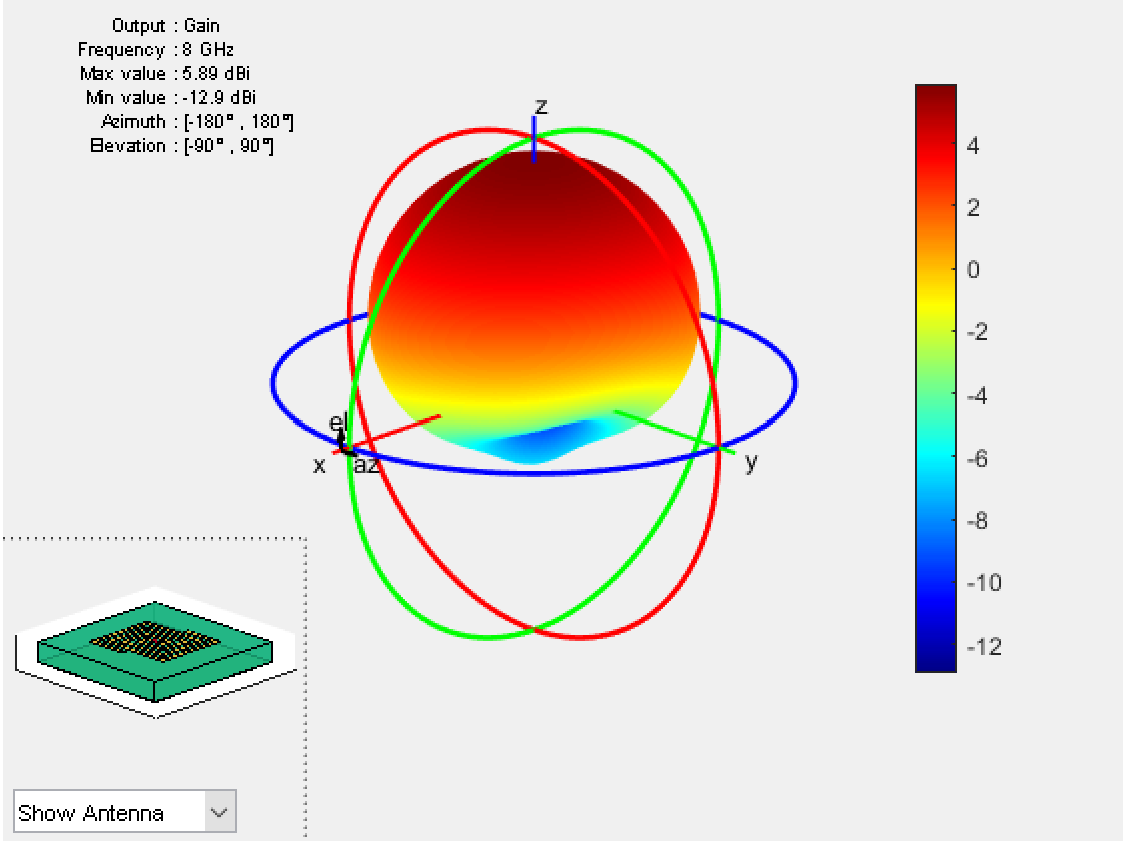
Back the spiral using a reflector antenna object.

```
    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.

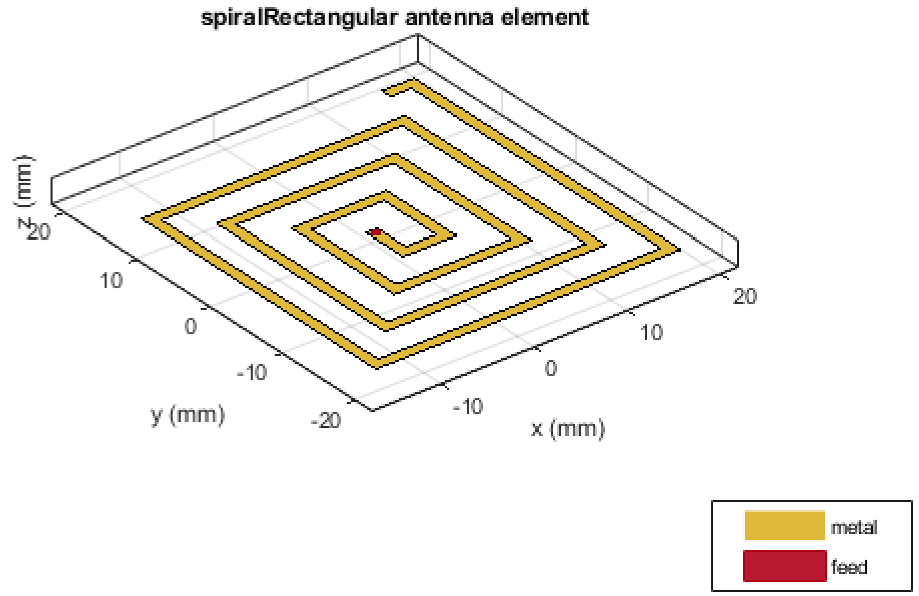
```
figure;
pattern(r,8e9);
```



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);
```



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

`rectspirallength2turns` | `spiralArchimedean` | `spiralEquiangular`

Topics

“Rotate Antennas and Arrays”

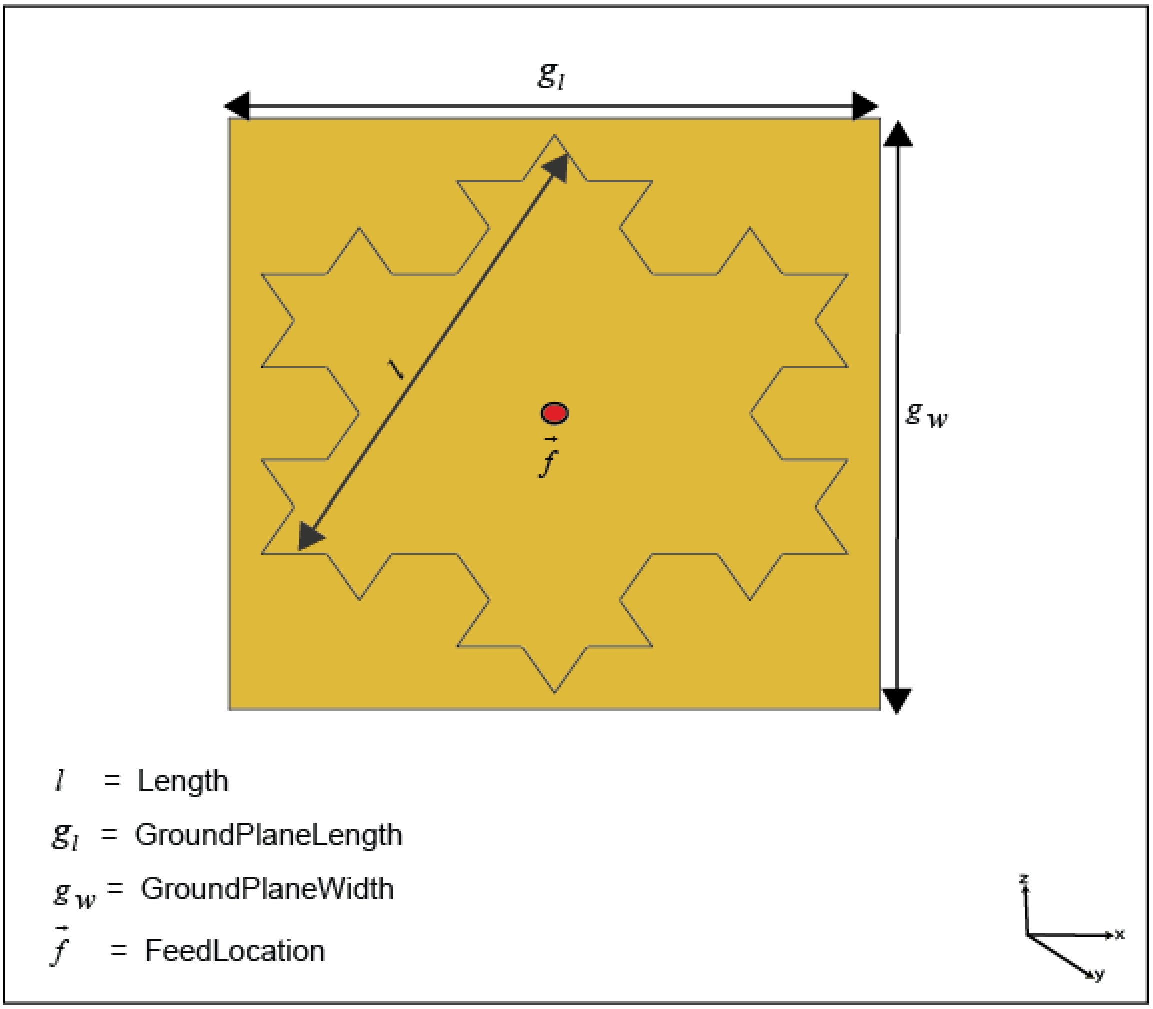
Introduced in R2020a

fractalSnowflake

Create fractal Koch snowflake antenna

Description

The `fractalSnowflake` object creates a Kochs 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`

`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 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 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 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'`

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

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 handle 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 or array structure; display shape as filled patch
impedance	Input impedance of antenna; scan impedance of array
sparameters	S-parameter object
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 metal or dielectric antenna or array surface
charge	Charge distribution on metal or dielectric 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 or dielectric antenna or array structure
design	Design prototype antenna or arrays for resonance at specified frequency

Examples

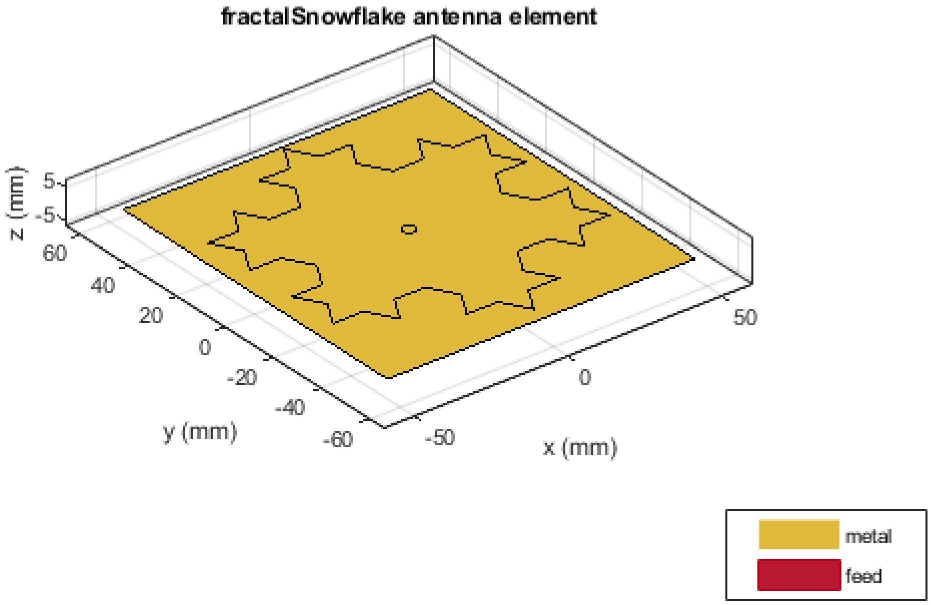
Fractal Snowflake with Default Properties

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: [1×1 dielectric]
     GroundPlaneLength: 0.1000
      GroundPlaneWidth: 0.1100
    FractalCenterOffset: [0 0]
            FeedOffset: [0 0]
          FeedDiameter: 0.0020
                  Tilt: 0
              TiltAxis: [1 0 0]
                  Load: [1×1 lumpedElement]
```

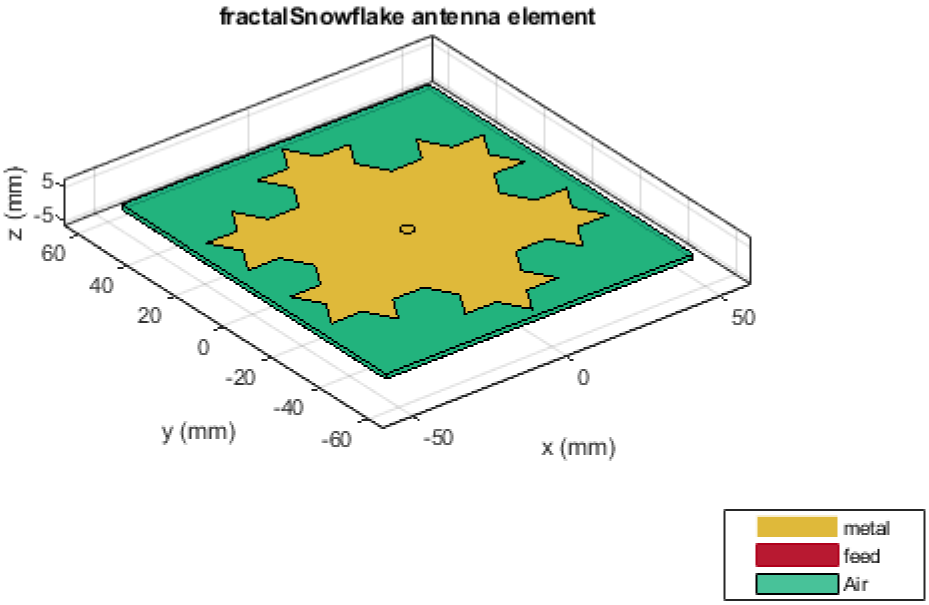
```
show(ant)
```



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.

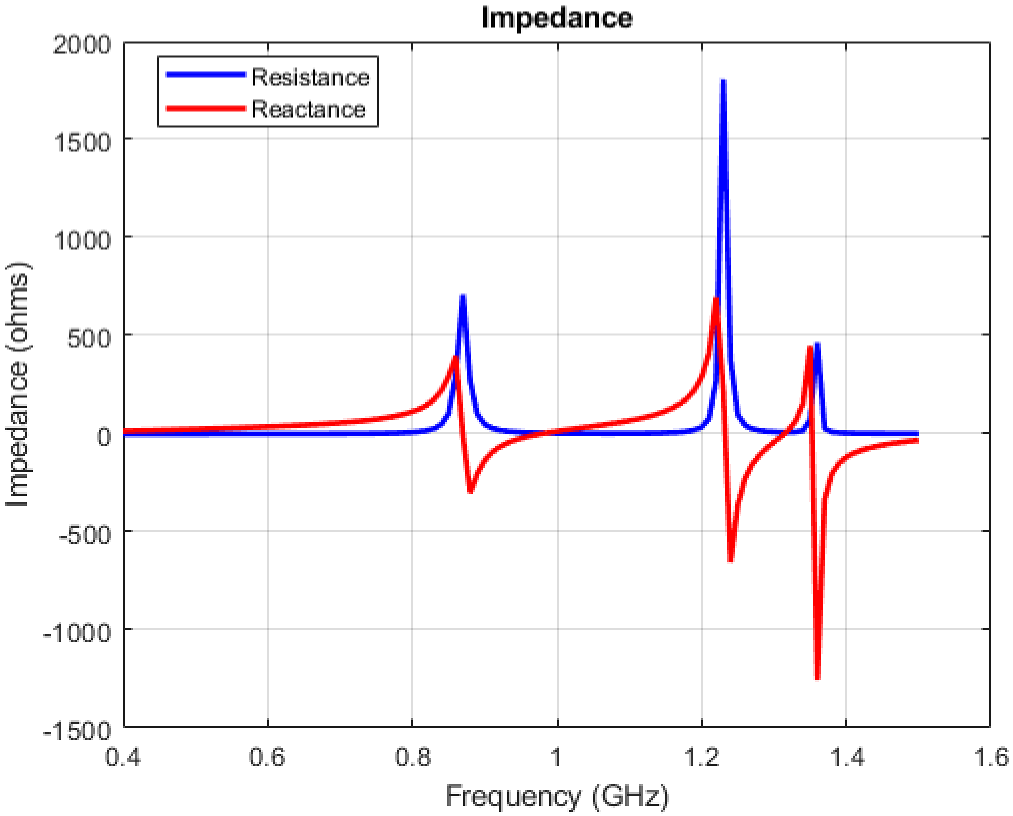
```
ant = fractalSnowflake('Substrate', dielectric('EpsilonR',4,...
          'Thickness',1.5e-3));
   show(ant);
```



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)
```



See Also

`fractalCarpet` | `fractalGasket` | `fractalIsland` | `fractalKoch`

Topics

“Rotate Antennas and Arrays”

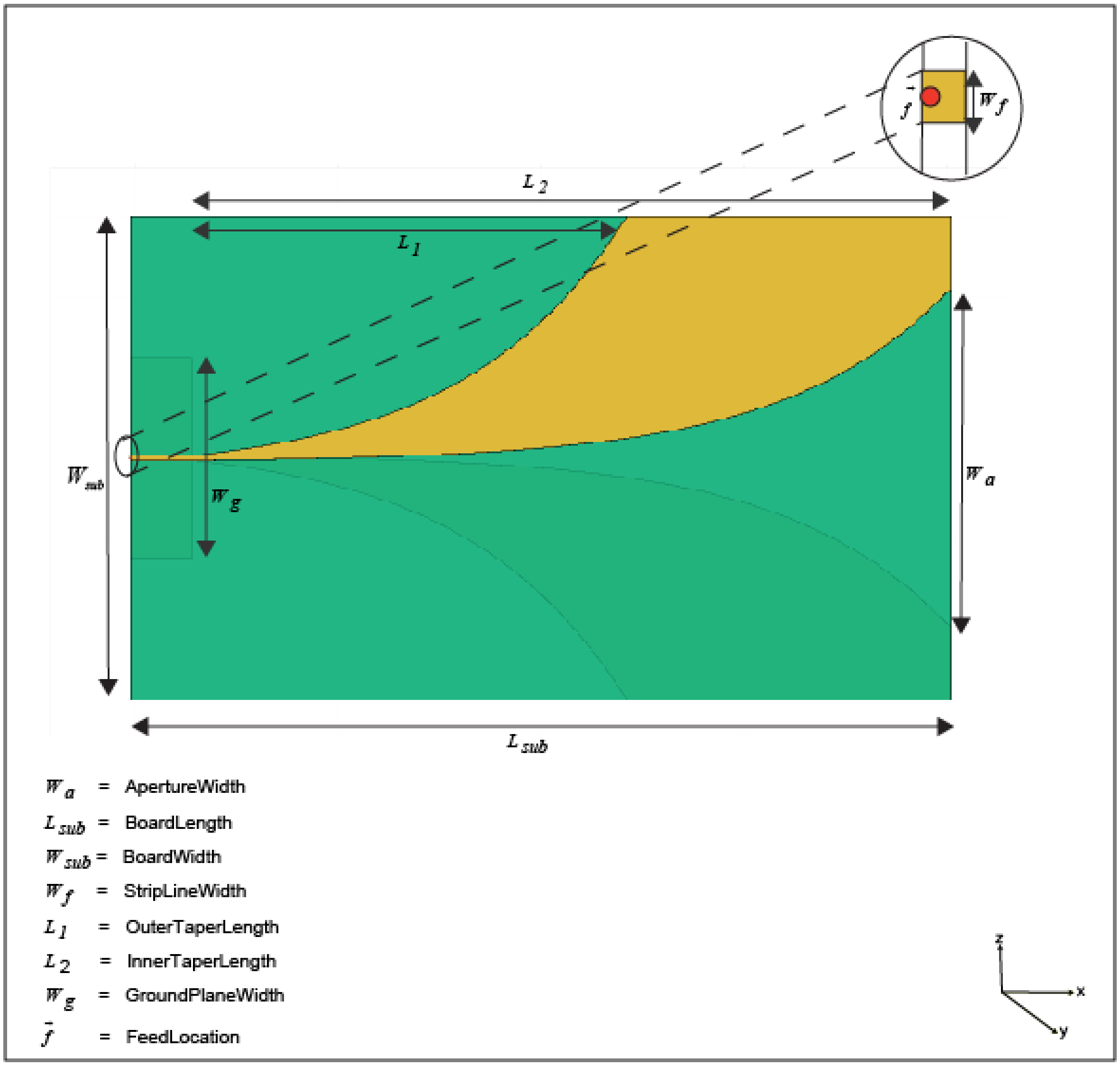
Introduced in R2020a

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 sidelobes 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 openieng 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))`

`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 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 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 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'

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

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: avi.Load = lumpedElement('Impedance',75)

Object Methods

show	Display antenna or array structure; display shape as filled patch
impedance	Input impedance of antenna; scan impedance of array
sparameters	S-parameter object
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
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric 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 or dielectric antenna or array structure
design	Design prototype antenna or arrays for resonance at specified frequency

Examples

Create and View Antipodal Vivaldi Antenna

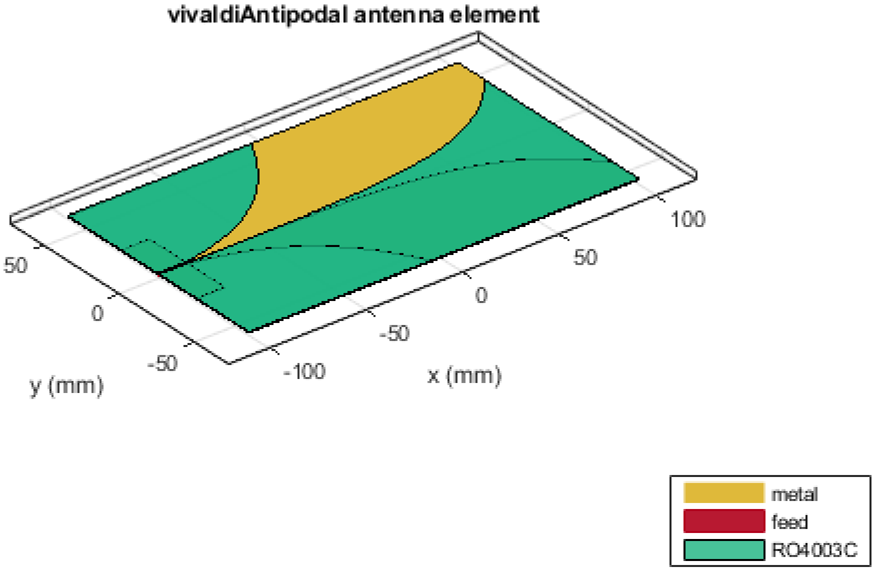
Create an antipodal Vivaldi antenna object with the specified properties.

```
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: [1×1 dielectric]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1×1 lumpedElement]
```

View the antenna.

```
show(avi)
```



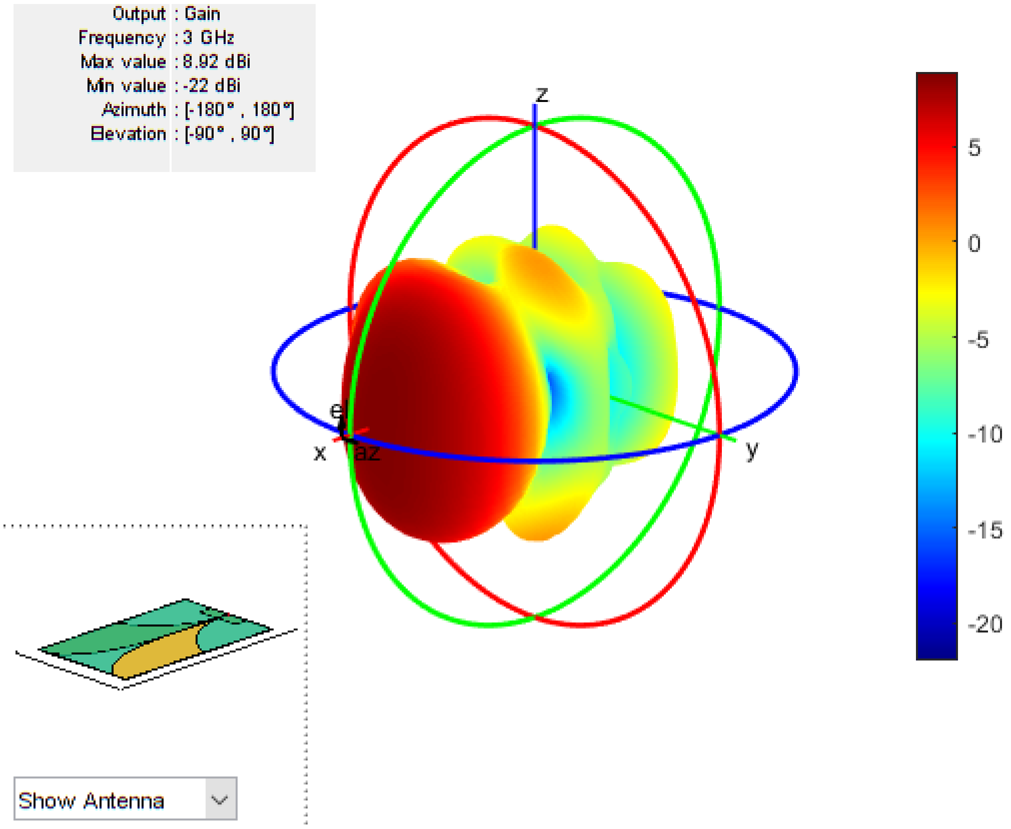
Radiation pattern of Antipodal Vivaldi Antenna

Plot the radiation pattern of the antipodal Vivaldi antenna at 3 GHz

```
avi=vivaldiAntipodal("OpeningRate",30,'Substrate',dielectric('Name','RO4003C','EpsilonR',3.38,'Lo
'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: [1×1 dielectric]
                Tilt: 0
            TiltAxis: [1 0 0]
                Load: [1×1 lumpedElement]
```

```
pattern(avi,3e9)
```



```
Output : Gain
Frequency : 3 GHz
Max value : 8.92 dBi
Min value : -22 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]
```

References

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

See Also

`slot` | `spiralArchimedean` | `vivaldi` | `yagiUda`

Topics

“Rotate Antennas and Arrays”

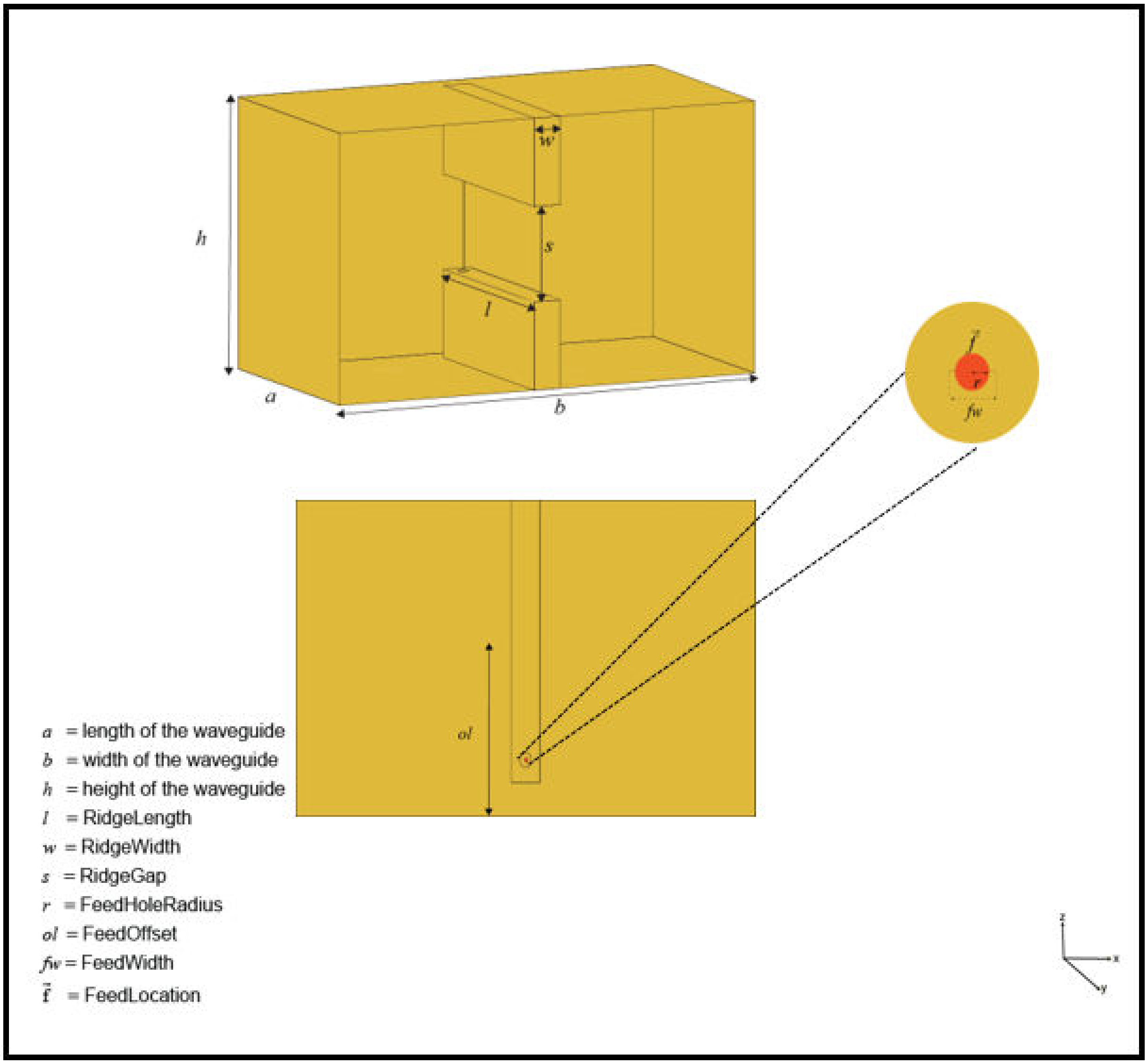
Introduced in R2020a

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`

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 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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch

impedance	Input impedance of antenna; scan impedance of array
sparameters	S-parameter object
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
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at specified frequency
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
mesh	Mesh properties of metal or 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]
              Tilt: 0
          TiltAxis: [1 0 0]
              Load: [1×1 lumpedElement]
```

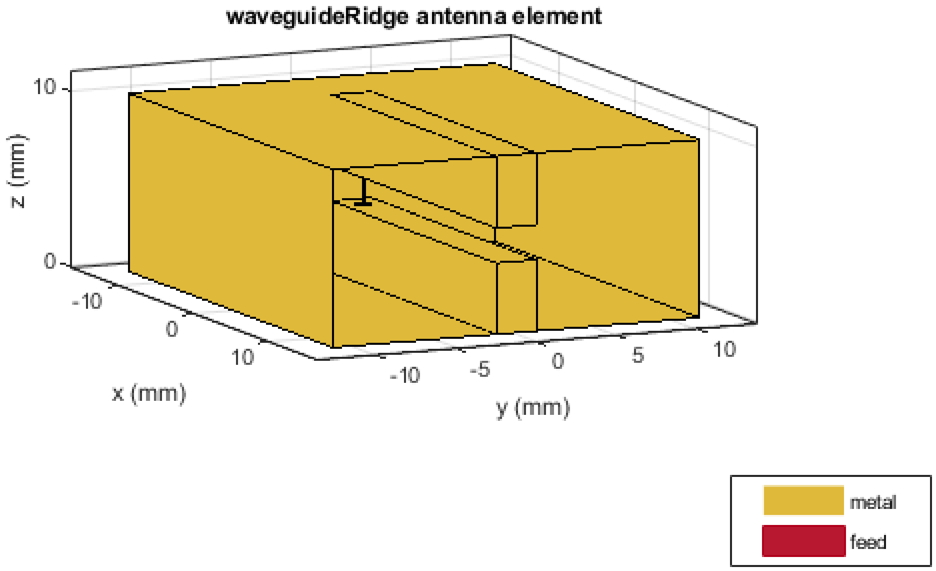
Create a Ridged Waveguide Antenna with Specified Properties

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);
```

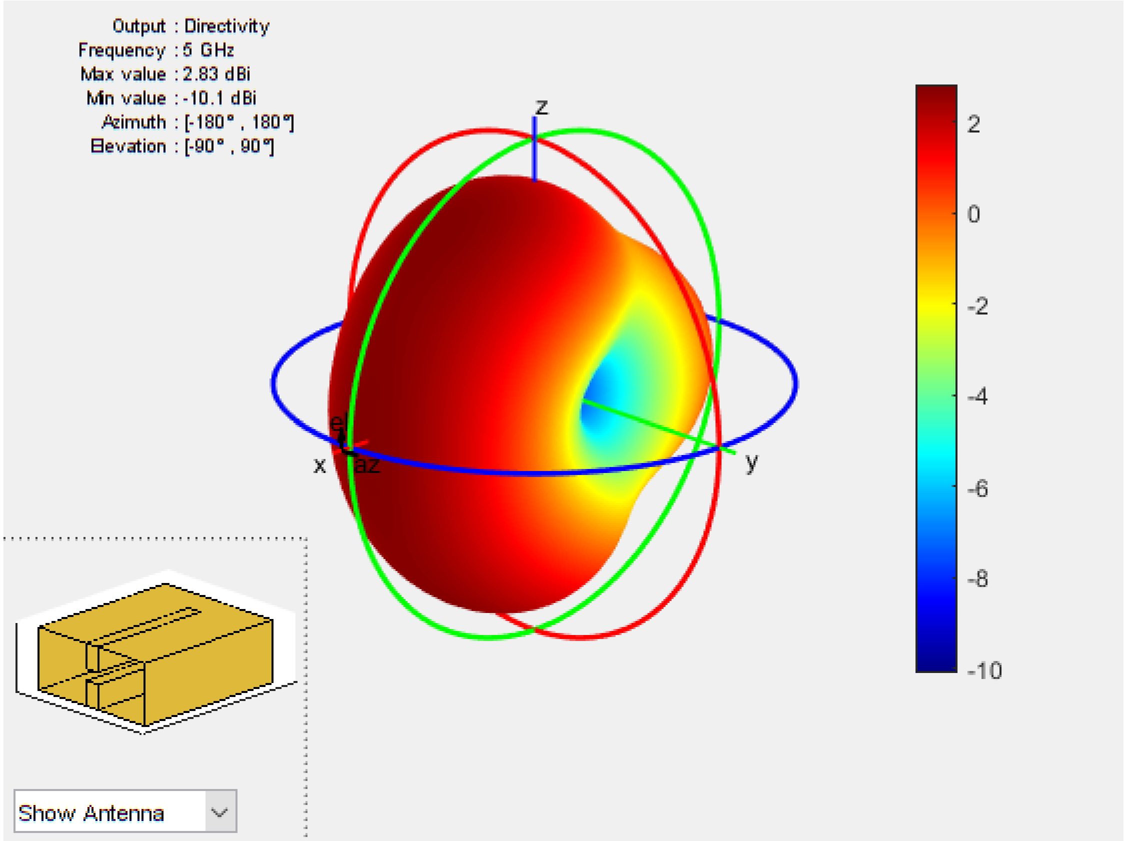
View the `waveguideRidge` antenna using a `show` function.

```
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 : 2.83 dBi
Min value : -10.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

See Also

`cavityCircular` | `waveguide` | `waveguideCircular` | `waveguideSlotted`

Topics

"Rotate Antennas and Arrays"

Introduced in R2019b

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: `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 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 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 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'`

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

Object Functions

show	Display antenna or array structure; display shape as filled patch
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
vswr	Voltage standing wave ratio of antenna

Examples

Default Wire Antenna

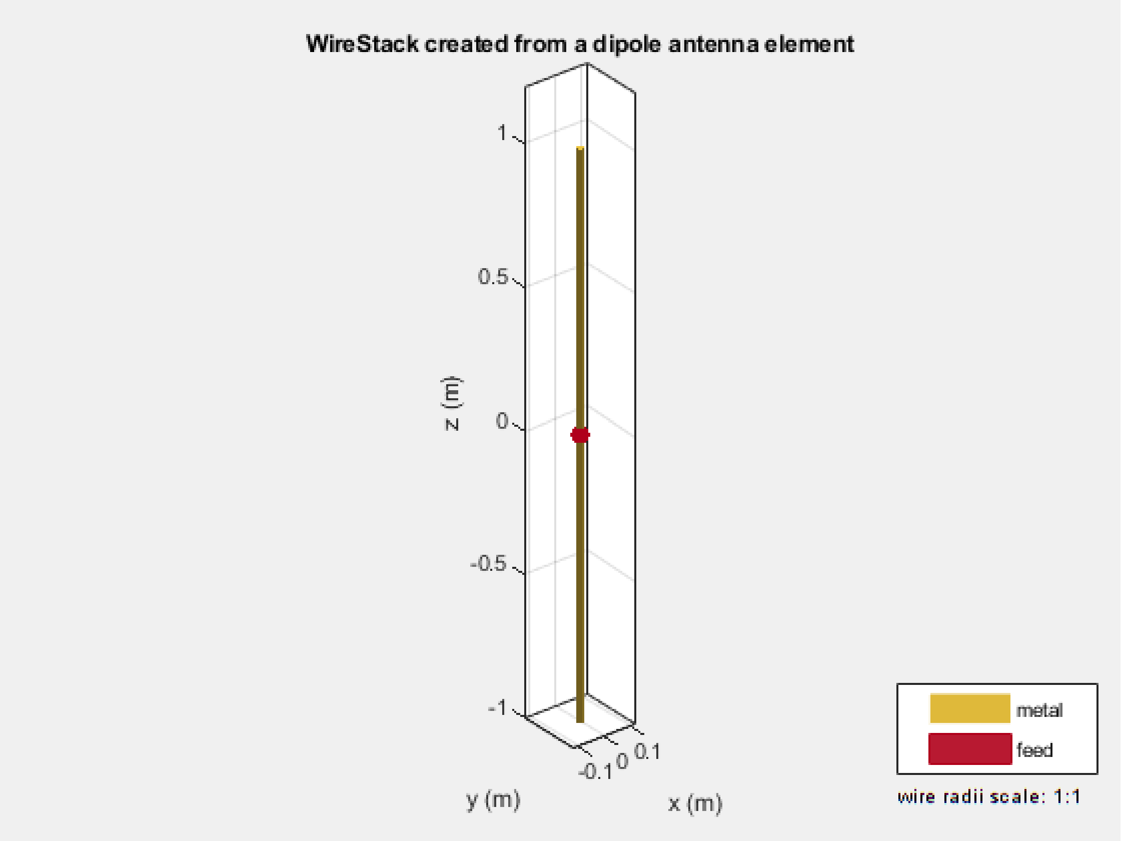
Create and view a default dipole wire antenna.

```
ant = wireStack
```

```
ant = 
  wireStack with properties:
```

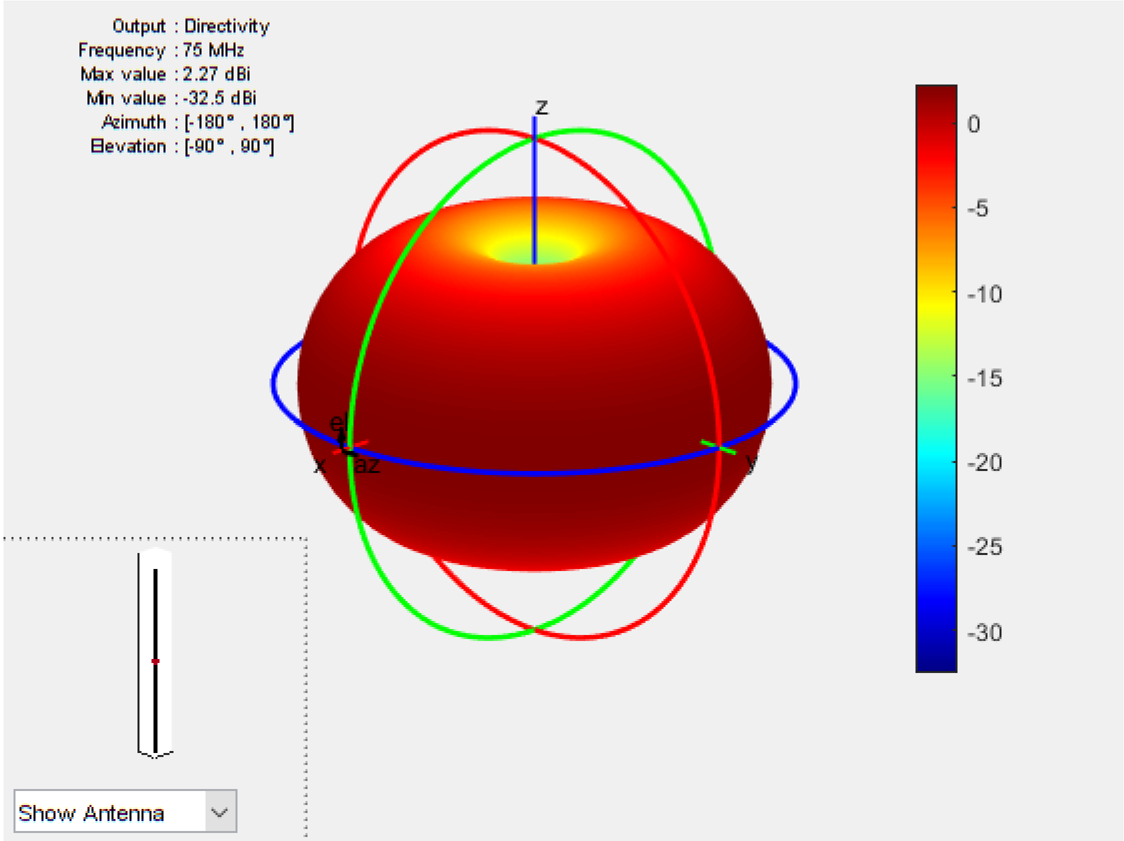
```
            Name: 'Dipole'
    FeedLocation: [0 0 0]
     FeedVoltage: 1
       FeedPhase: 0
            Tilt: 0
        TiltAxis: [1 0 0]
```

```
show(ant)
```



Plot the radiation pattern of the antenna at the specified frequency.

```
pattern(ant,75e6)
```



See Also

dipole | dipoleCrossed | dipoleCycloid | dipoleFolded | dipoleHelix | dipoleJ | dipoleMeander | dipoleVee | loopCircular | loopRectangular | strip2cylinder

Topics

"Rotate Antennas and Arrays"
"Wire Solver"

Introduced in R2020a

3

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.

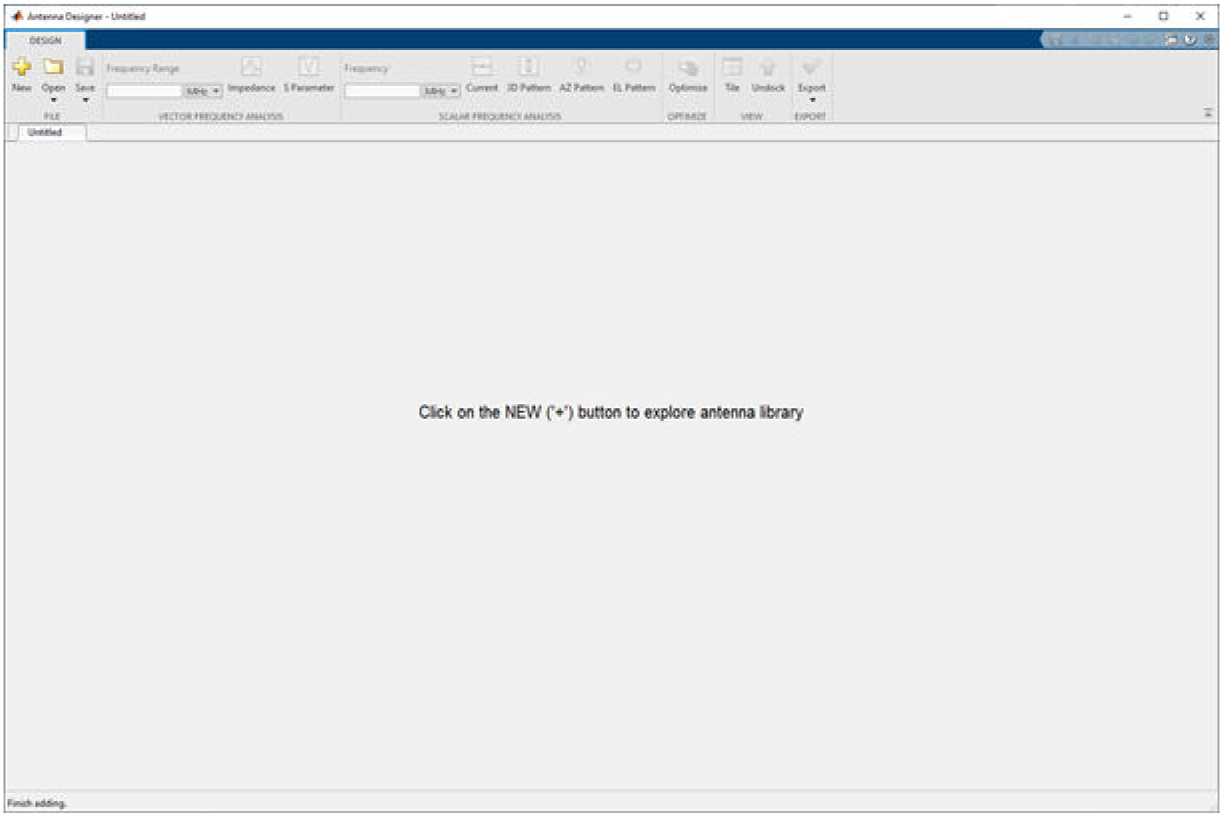
Open the Antenna Designer App

- 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.



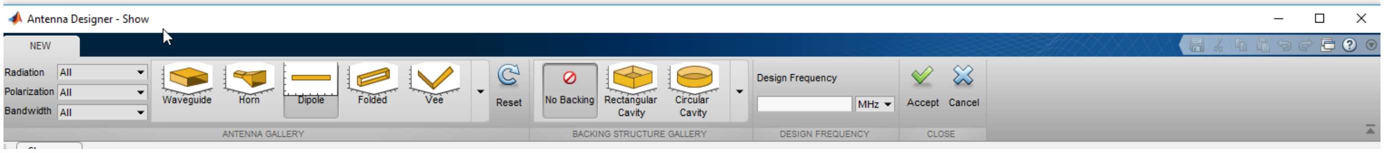
Click on the NEW ('+') button to explore antenna library

Finish adding.

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.



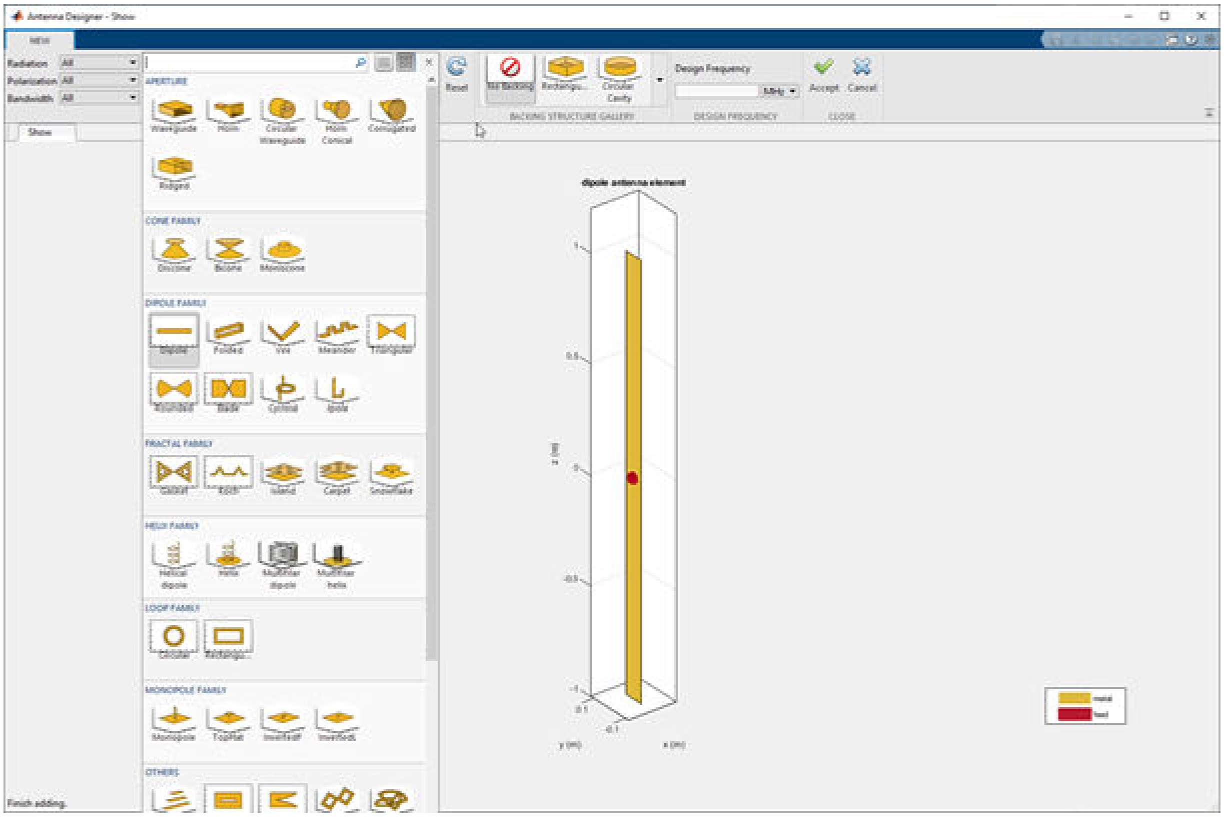
You can filter the antennas based on `Radiation` pattern, `Polarization`, and `Bandwidth`.

- 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**.

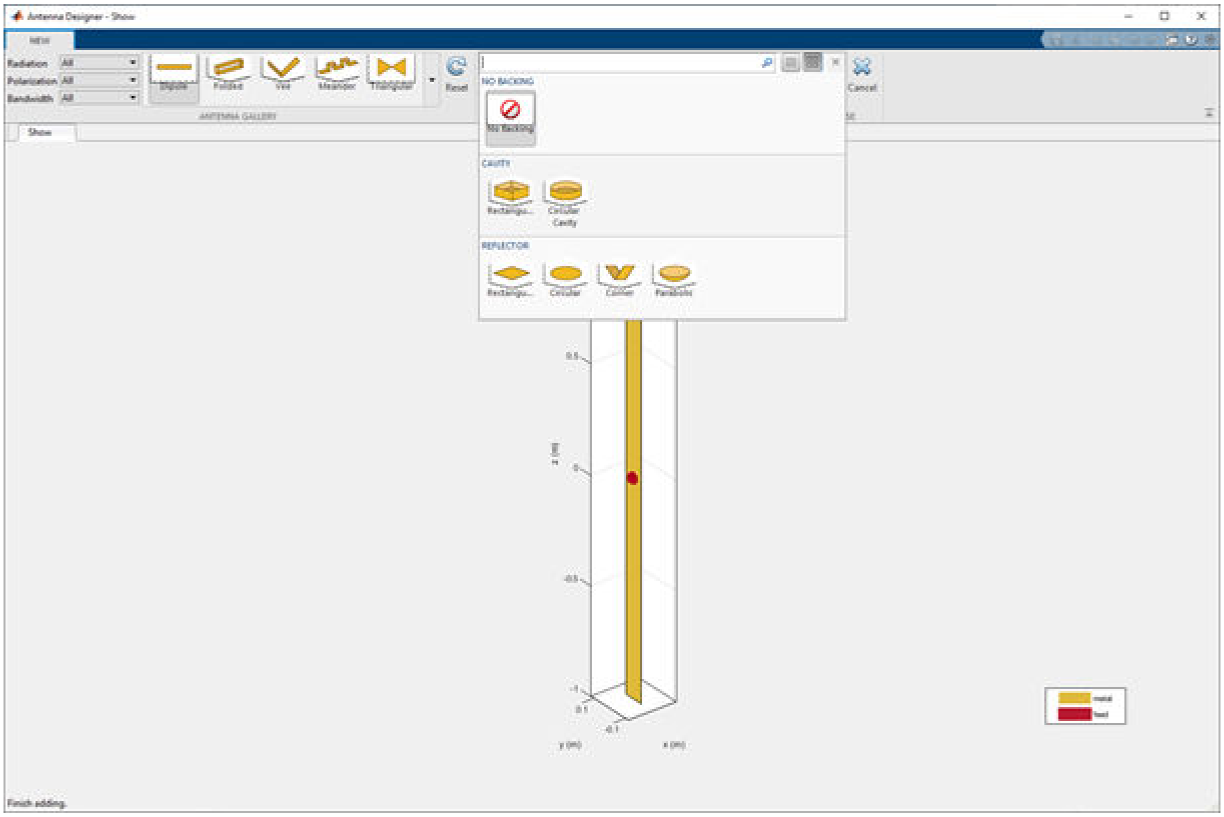


Radiation All
Polarization All
Bandwidth All

- 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**.



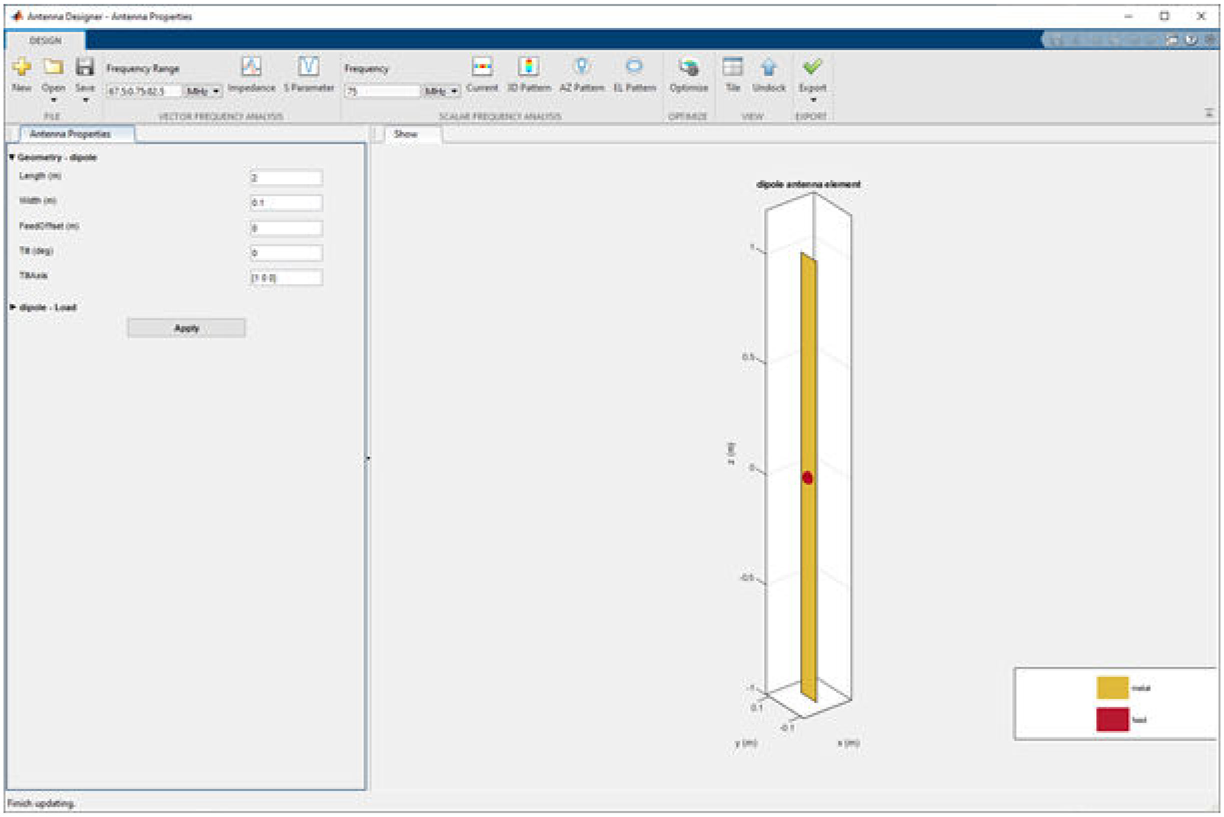
Radiation All

Polarization All

Bandwidth All

Finish adding.

4 Analyze Antenna

- 

  Frequency Range 67.5:0.75:82.5 MHz

  Frequency 75 MHz

  ▼ Geometry - dipole

  Length (m) 2

  Width (m) 0.1

  FeedOffset (m) 0

  Tilt (deg) 0

  TiltAxis [1 0 0]

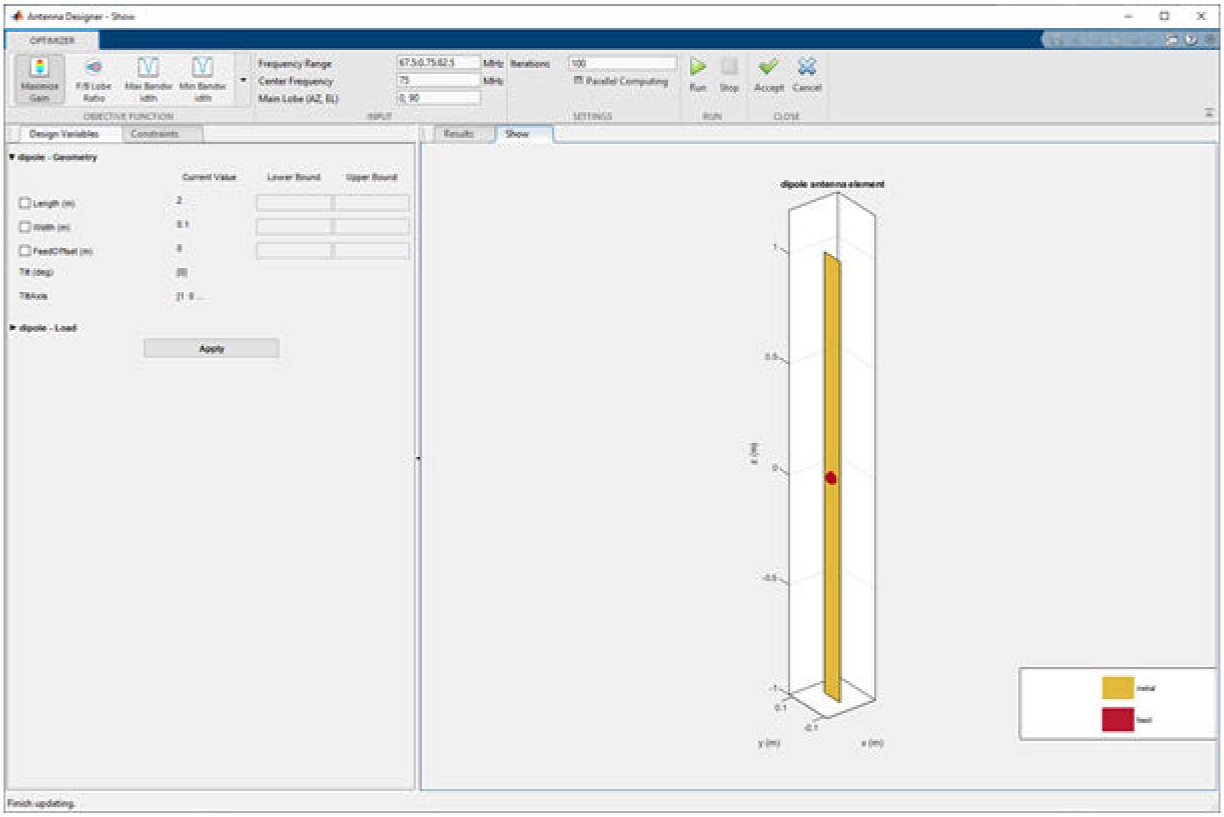
  ▶ dipole - Load

  Finish 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 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.



Frequency Range 67.5:0.75:82.5 MHz

Center Frequency 75 MHz

Main Lobe (AZ, EL) 0, 90

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 ...		

▸ dipole - Load

Use the **OBJECTIVE FUNCTION**to choose the main goal of optimizing the antenna.

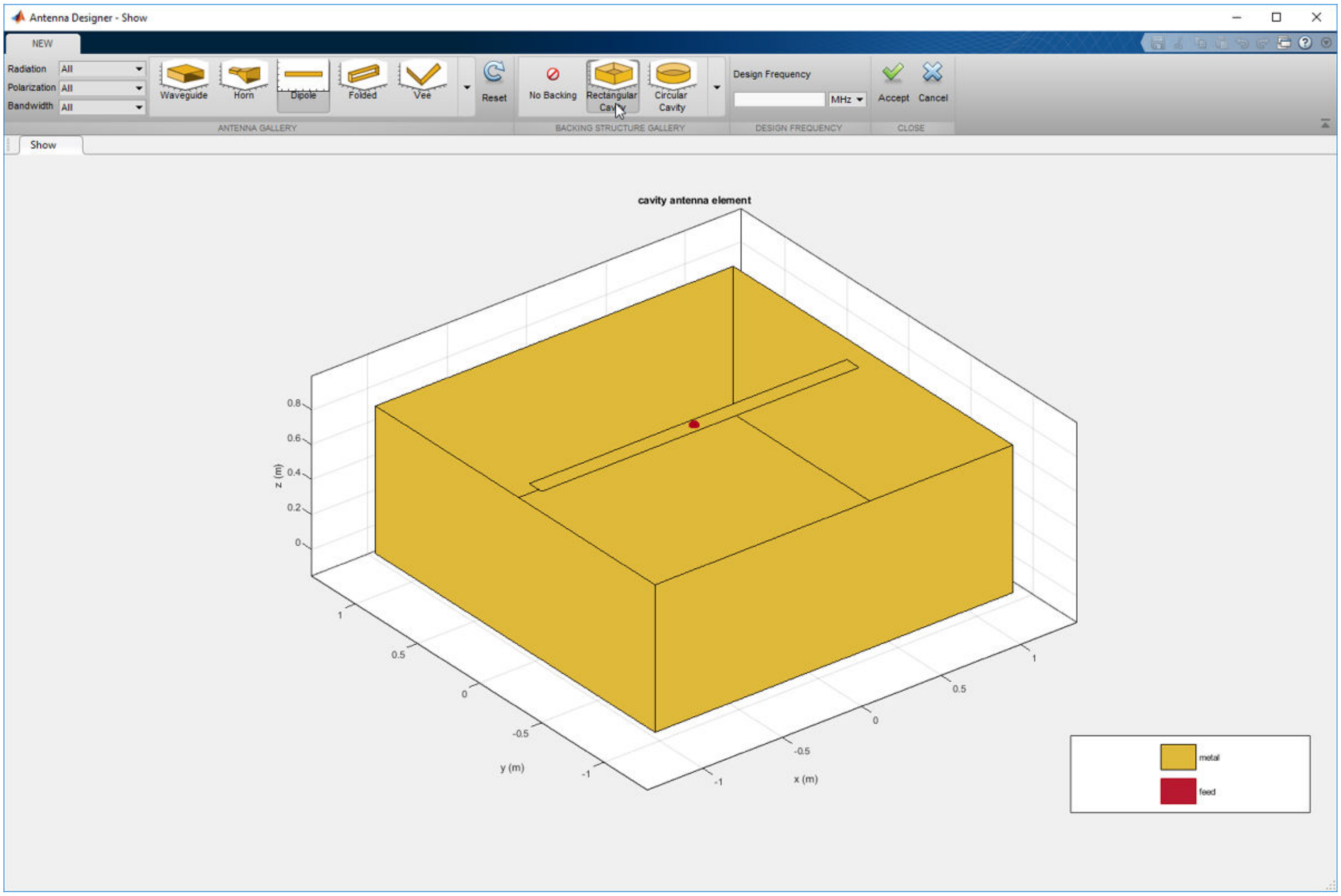
- Use the **Design Variables** to input the variables The variables are then changed by the optimizer depending on the lower and upper bounds.
- Use **Constraints** functions to restrict a desired analysis function value on the antenna.
- After adding the required values, click **Run** to start the optimization.

Plot Radiation Pattern of Cavity-Backed Dipole

Use the **Antenna Designer** app to plot the radiation pattern of a cavity-backed dipole antenna.

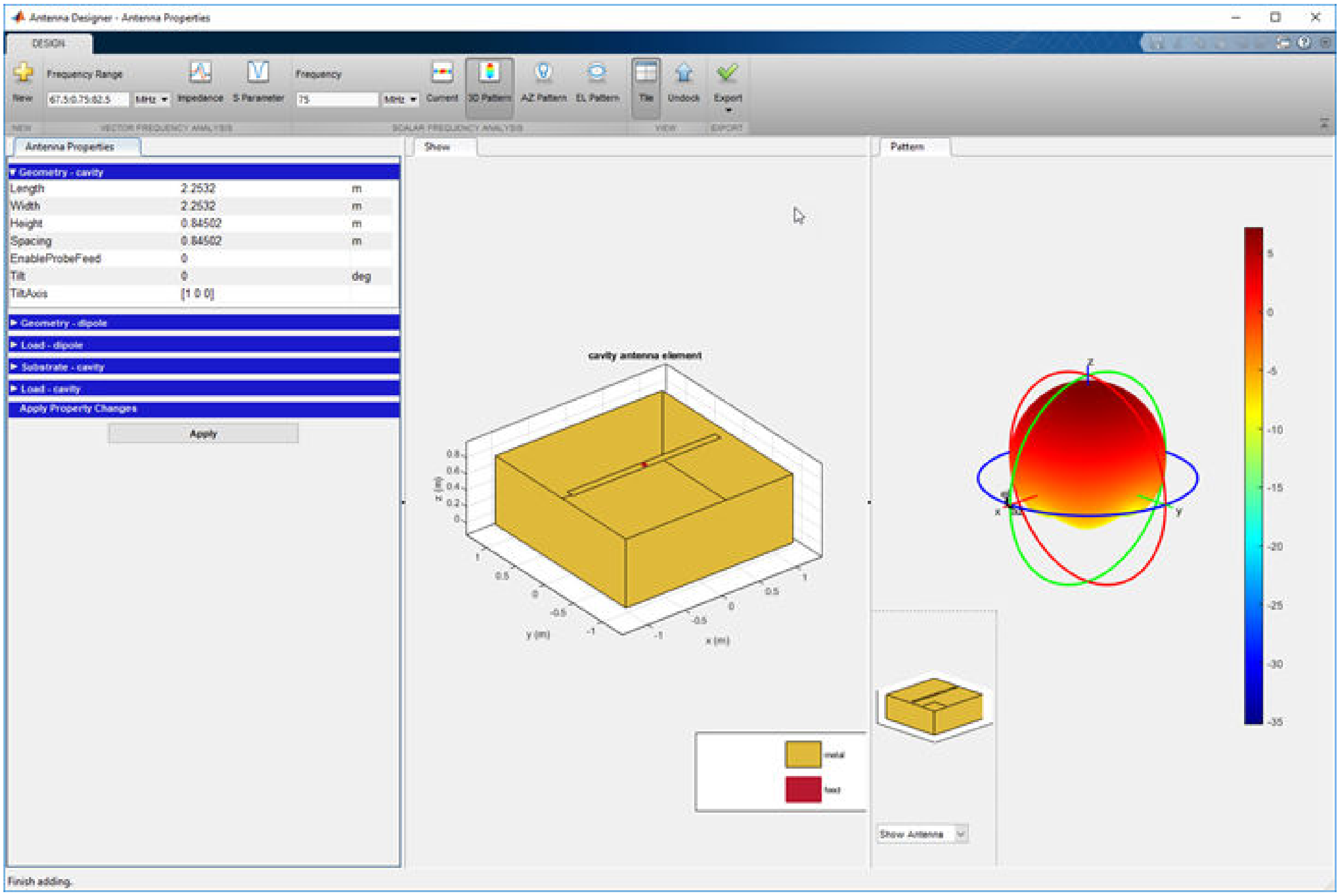
Open the app and click **New** to show the default dipole antenna.

From the **BACKING STRUCTURE GALLERY,** click **Rectangular Cavity** to create a cavity-backed dipole antenna.



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.



Frequency Range: 67.5:0.75:82.5 MHz

Frequency: 75 MHz

▼ Geometry - cavity

Length	2.2532	m
Width	2.2532	m
Height	0.84502	m
Spacing	0.84502	m
EnableProbeFeed	0	
Tilt	0	deg
TiltAxis	[1 0 0]	

► Geometry - dipole

► Load - dipole

► Substrate - cavity

► Load - cavity

Apply Property Changes

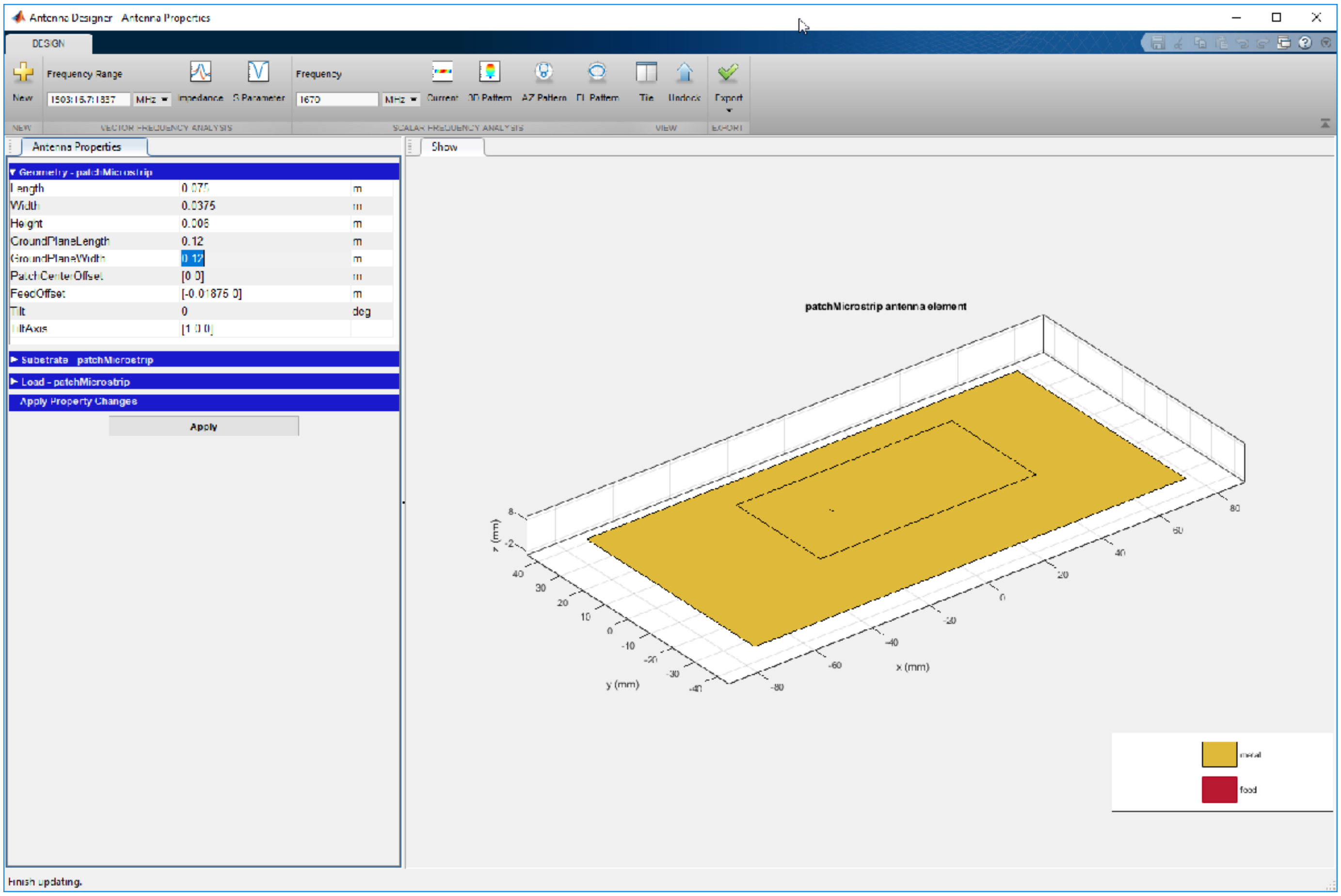
Finish adding.

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.

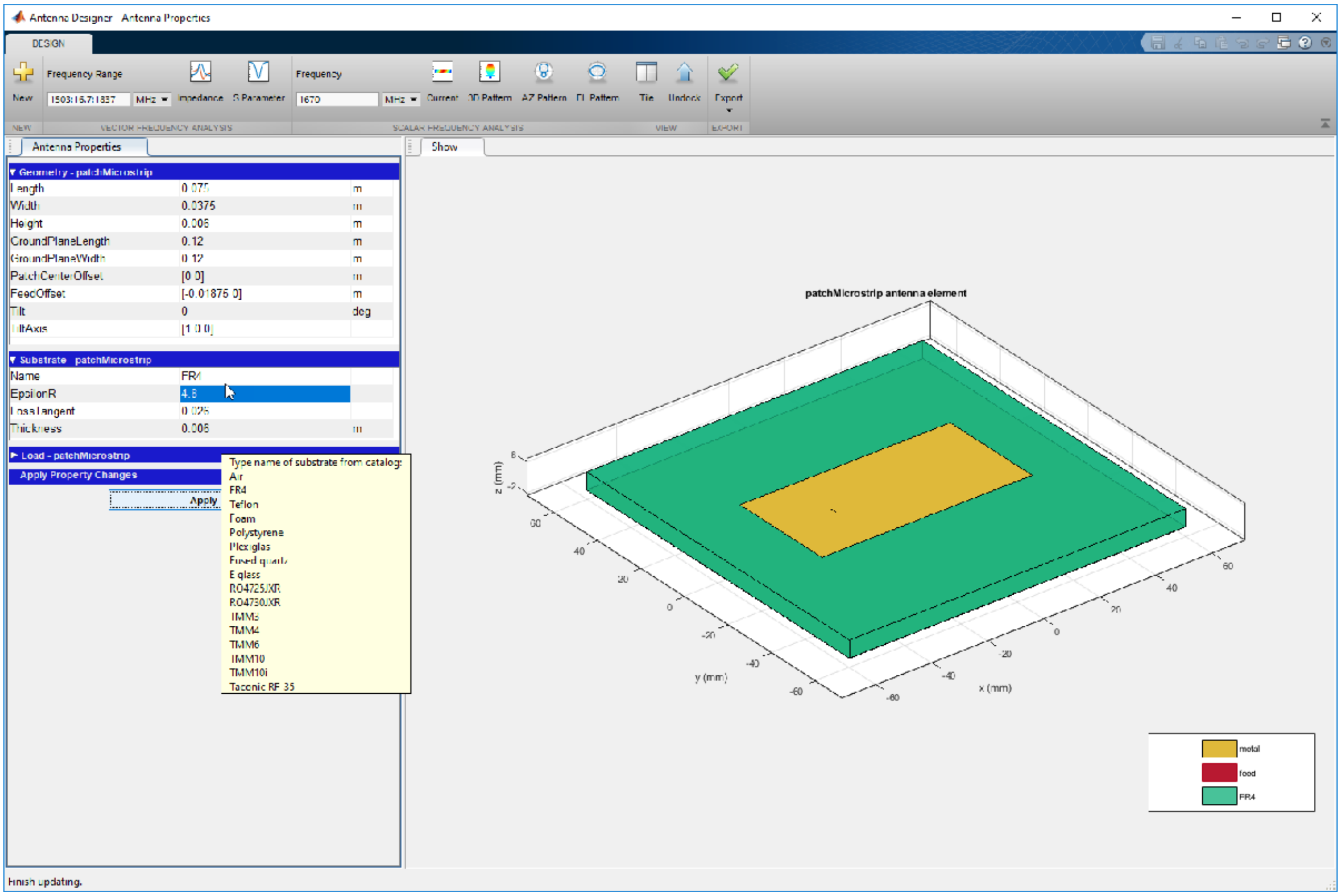
Open the app and click **New**. In the **ANTENNA GALLERY** section, under **PATCH FAMILY**, click `Microstrip.` Click **Accept**.

On the **Antenna Properties** tab, change the groundplane length and groundplane width to 0.120 m. Click **Apply** to see the changes.



Length	0.075	m
Width	0.0375	m
Height	0.006	m
GroundPlaneLength	0.12	m
GroundPlaneWidth	0.12	m
PatchCenterOffset	[0 0]	m
FeedOffset	[-0.01875 0]	m
Tilt	0	deg
TiltAxis	[1 0 0]	

Add an FR4 dielectric as a substrate to the patch microstrip antenna. To add the dielectric, open the **Substrate** section and hover over the **Name** tab to see the `Dielectric Catalog.` Set the substrate **Name** to FR4, **EpsilonR** to 4.8000, and **Loss Tangent** to 0.0260. Click **Apply** to see the antenna.



Antenna Designer   Antenna Properties

Frequency Range: 1503:16.7:1837 MHz

Frequency: 1670 MHz

Antenna Properties

▼ Geometry - patchMicrostrip

Length	0.075	m
Width	0.0375	m
Height	0.006	m
GroundPlaneLength	0.12	m
GroundPlaneWidth	0.12	m
PatchCenterOffset	[0 0]	m
FeedOffset	[-0.01875 0]	m
Tilt	0	deg
TiltAxis	[1 0 0]	

▼ Substrate - patchMicrostrip

Name	FR4	
EpsilonR	4.8	
LossTangent	0.026	
Thickness	0.006	m

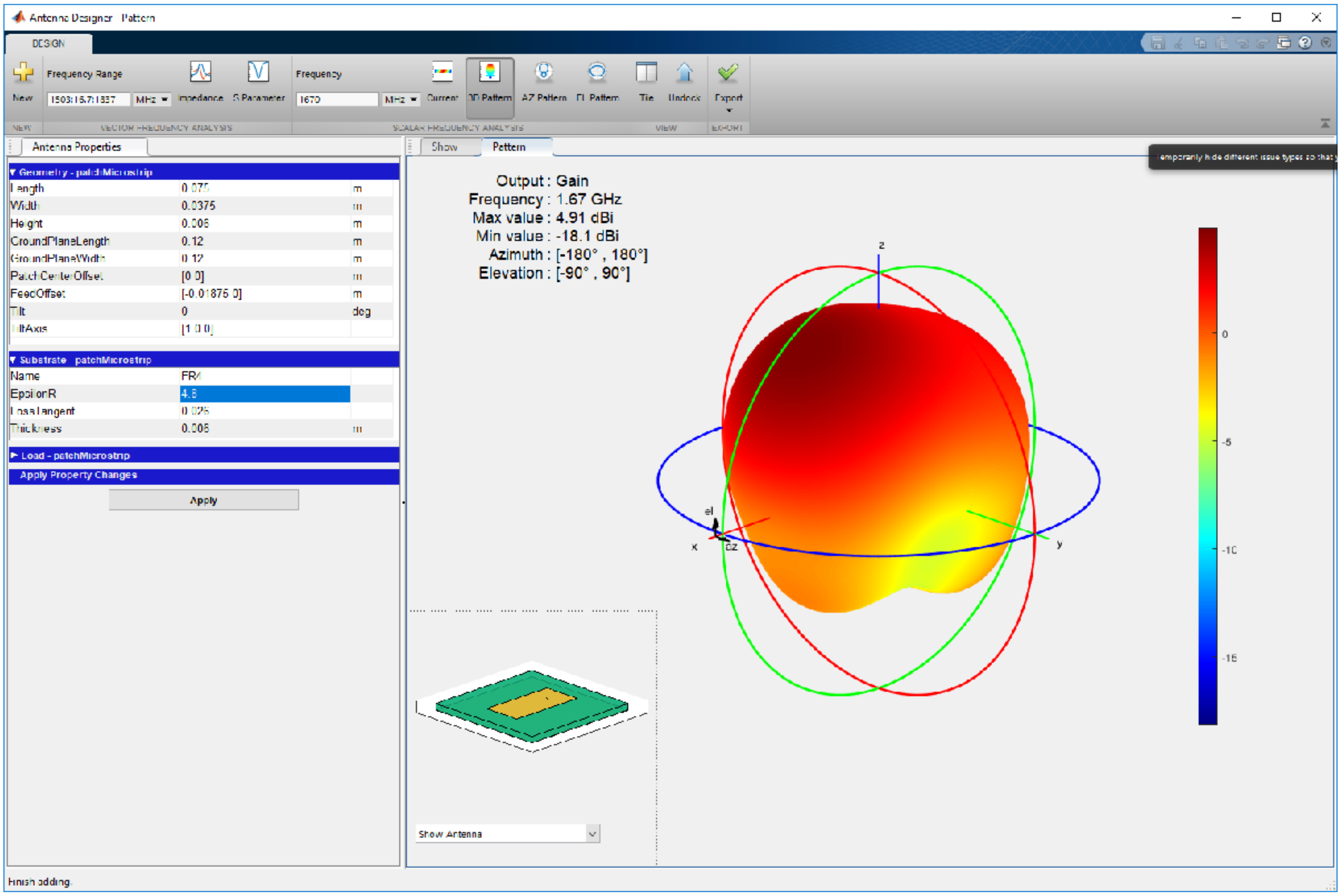
▶ Load - patchMicrostrip

Apply Property Changes

Type name of substrate from catalog:
Air
FR4
Teflon
Foam
Polystyrene
Plexiglas
Fused quartz
E glass
RO4725JXR
RO4730JXR
TMM3
TMM4
TMM6
TMM10
TMM10i
Taconic RF 35

Finish updating.

Click **3D Pattern** to plot the radiation pattern of the antenna at the default frequency of 1.67 GHz.



Frequency Range: 1503:16.7:1837 MHz

Frequency: 1670 MHz

▼ Geometry - patchMicrostrip

Length	0.075	m
Width	0.0375	m
Height	0.006	m
GroundPlaneLength	0.12	m
GroundPlaneWidth	0.12	m
PatchCenterOffset	[0 0]	m
FeedOffset	[-0.01875 0]	m
Tilt	0	deg
TiltAxis	[1 0 0]	

▼ Substrate - patchMicrostrip

Name	FR4	
EpsilonR	4.8	
LossTangent	0.026	
Thickness	0.006	m

▶ Load - patchMicrostrip

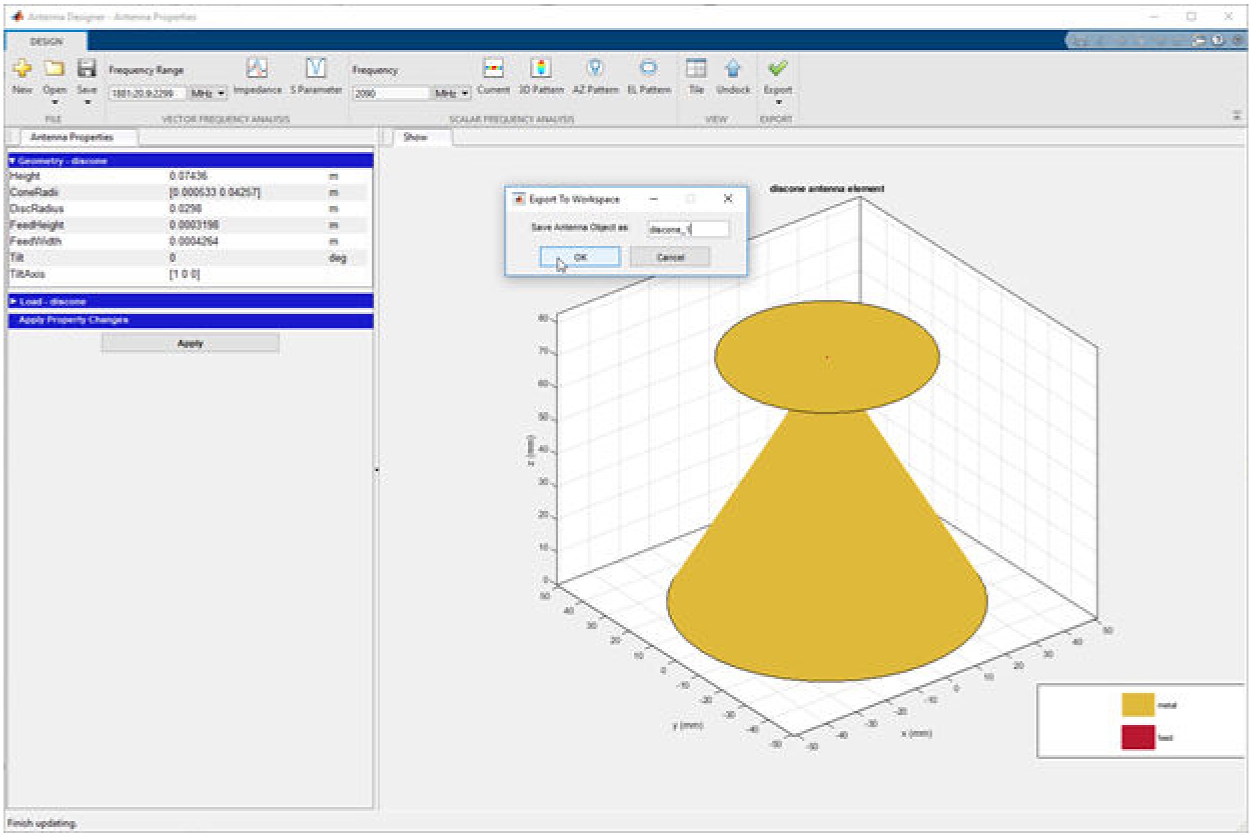
Apply Property Changes

Output : Gain
Frequency : 1.67 GHz
Max value : 4.91 dBi
Min value : -18.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Show Antenna

Export, Save, Load and Analyze Discone Antenna

Create and export a discone antenna using Antenna Designer app.



Frequency Range 1881:20.9:2299 MHz

Frequency 2090 MHz

Geometry - discone

Height	0.07436	m
ConeRadii	[0.000533 0.04257]	m
DiscRadius	0.0298	m
FeedHeight	0.0003198	m
FeedWidth	0.0004264	m
Tilt	0	deg
TiltAxis	[1 0 0]	

Export To Workspace

Save Antenna Object as: discone_1

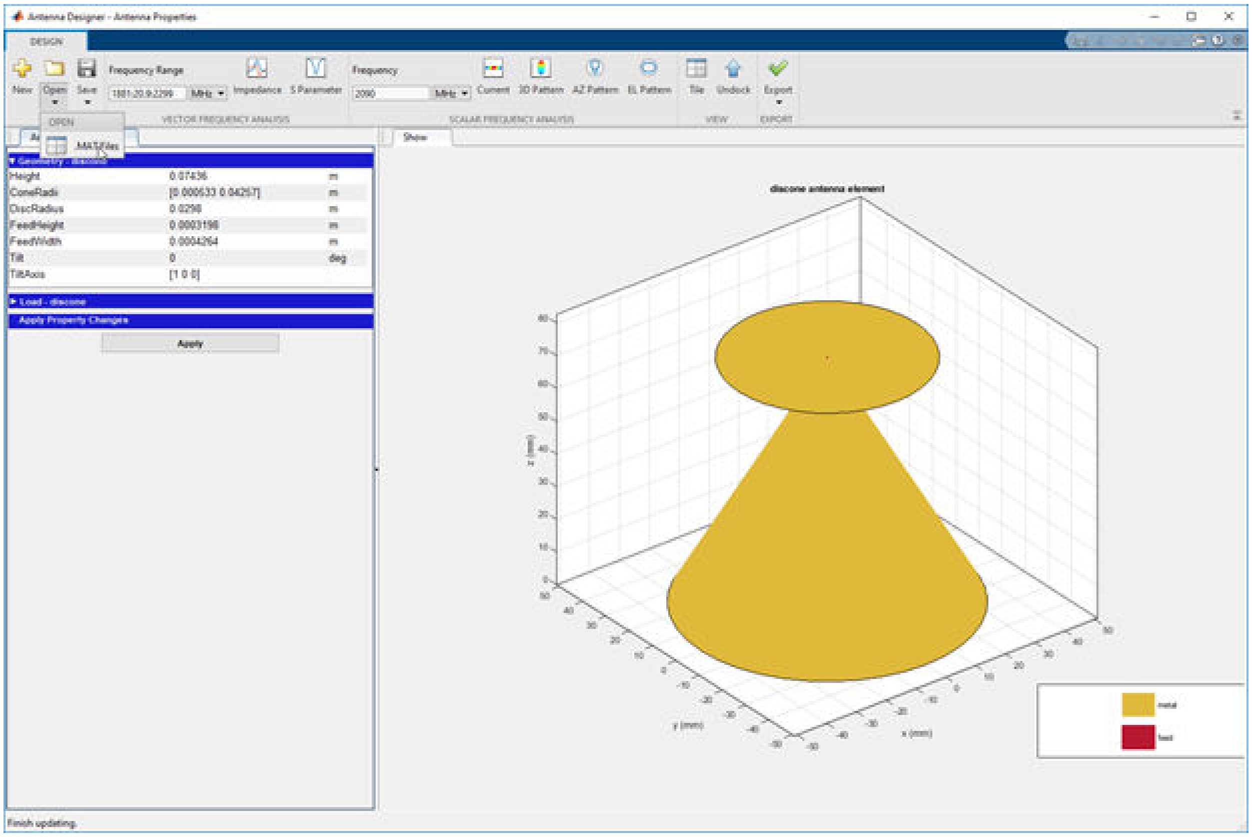
Finish updating.

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.

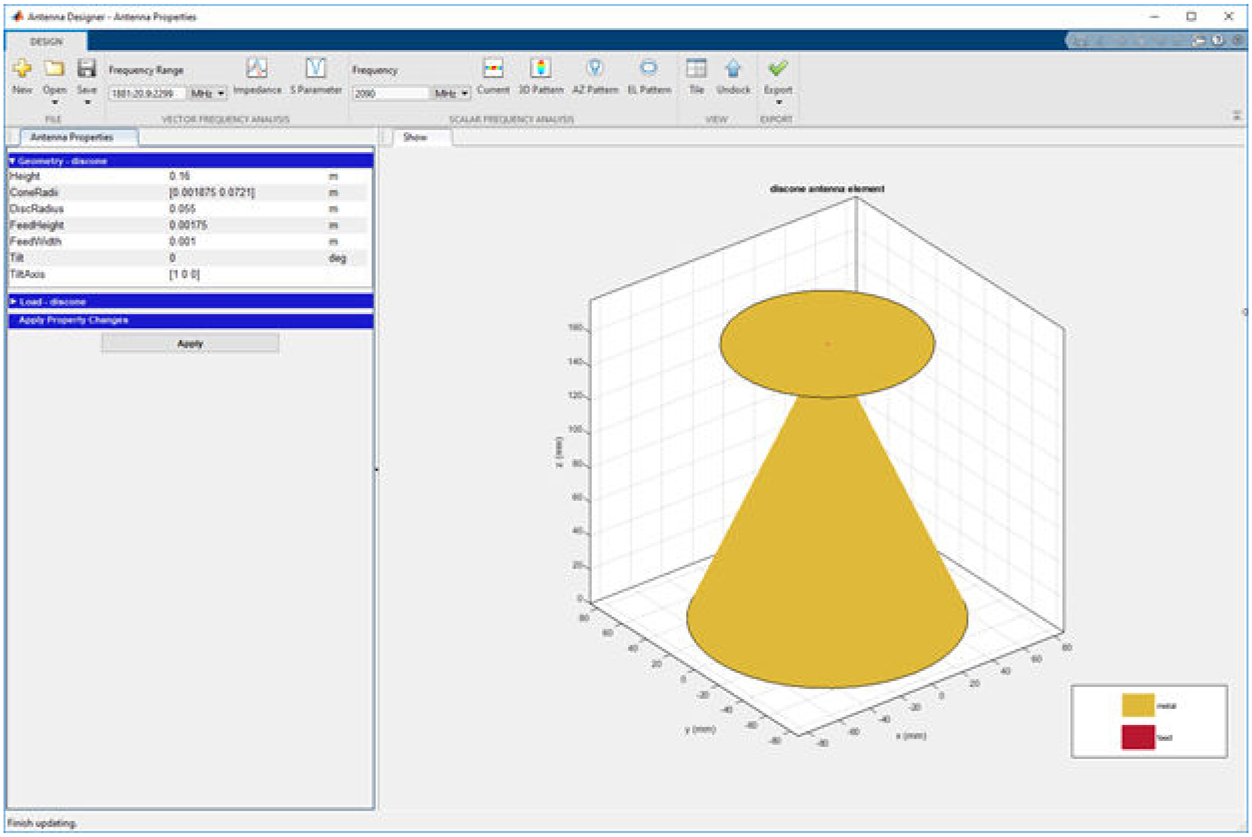


Frequency Range: 1881:20.9:2299 MHz

Frequency: 2090 MHz

Height	0.07436	m
ConeRadii	[0.000533 0.04257]	m
DiscRadius	0.0298	m
FeedHeight	0.0003198	m
FeedWidth	0.0004264	m
Tilt	0	deg
TiltAxis	[1 0 0]	

The app will overwrite the previous discone antenna design and open the updated discone antenna.



Frequency Range: 1801:20.9:2299 MHz

Frequency: 2090 MHz

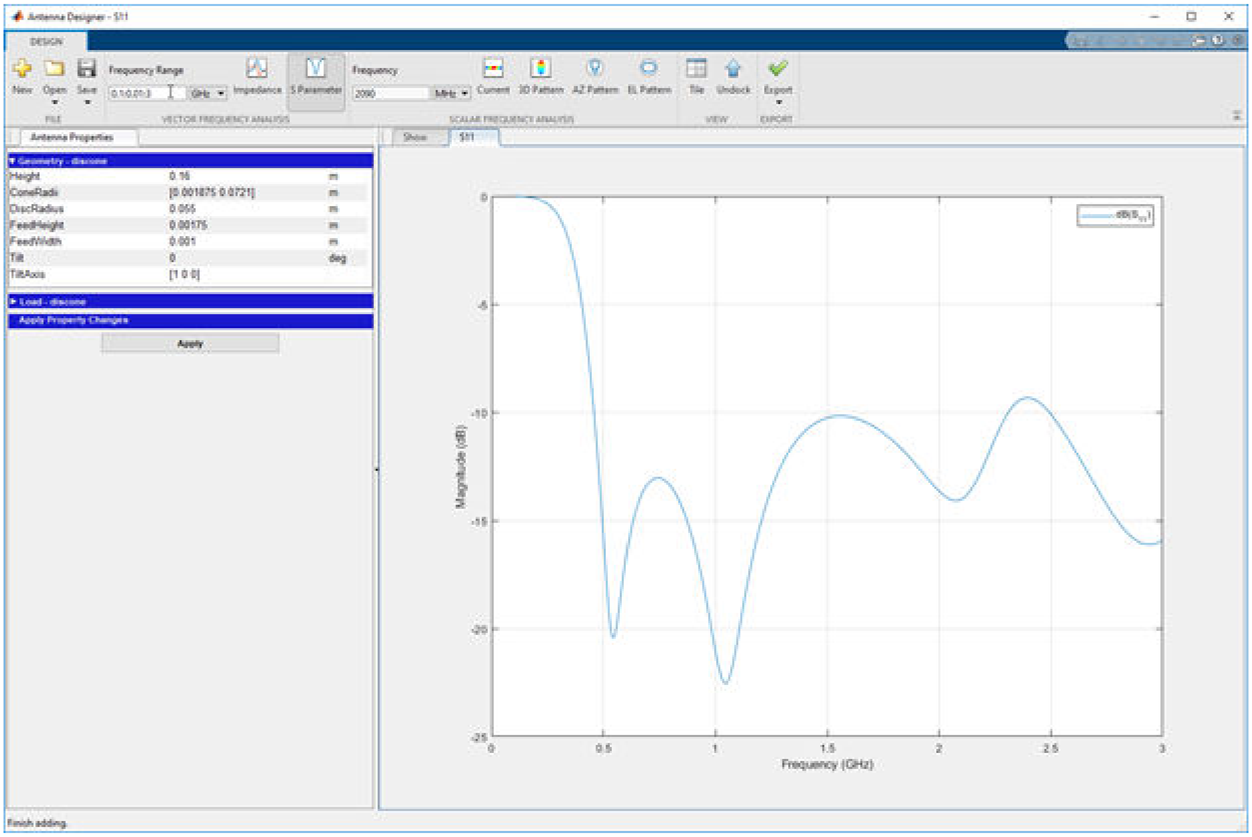
▼ Geometry - discone

Height	0.16	m
ConeRadii	[0.001875 0.0721]	m
DiscRadius	0.055	m
FeedHeight	0.00175	m
FeedWidth	0.001	m
Tilt	0	deg
TiltAxis	[1 0 0]	

▶ Load - discone

Apply Property Changes

Calculate the S-parameter of the antenna at the specified frequency range.



Frequency Range: 0.1:0.01:3 GHz

Frequency: 2090 MHz

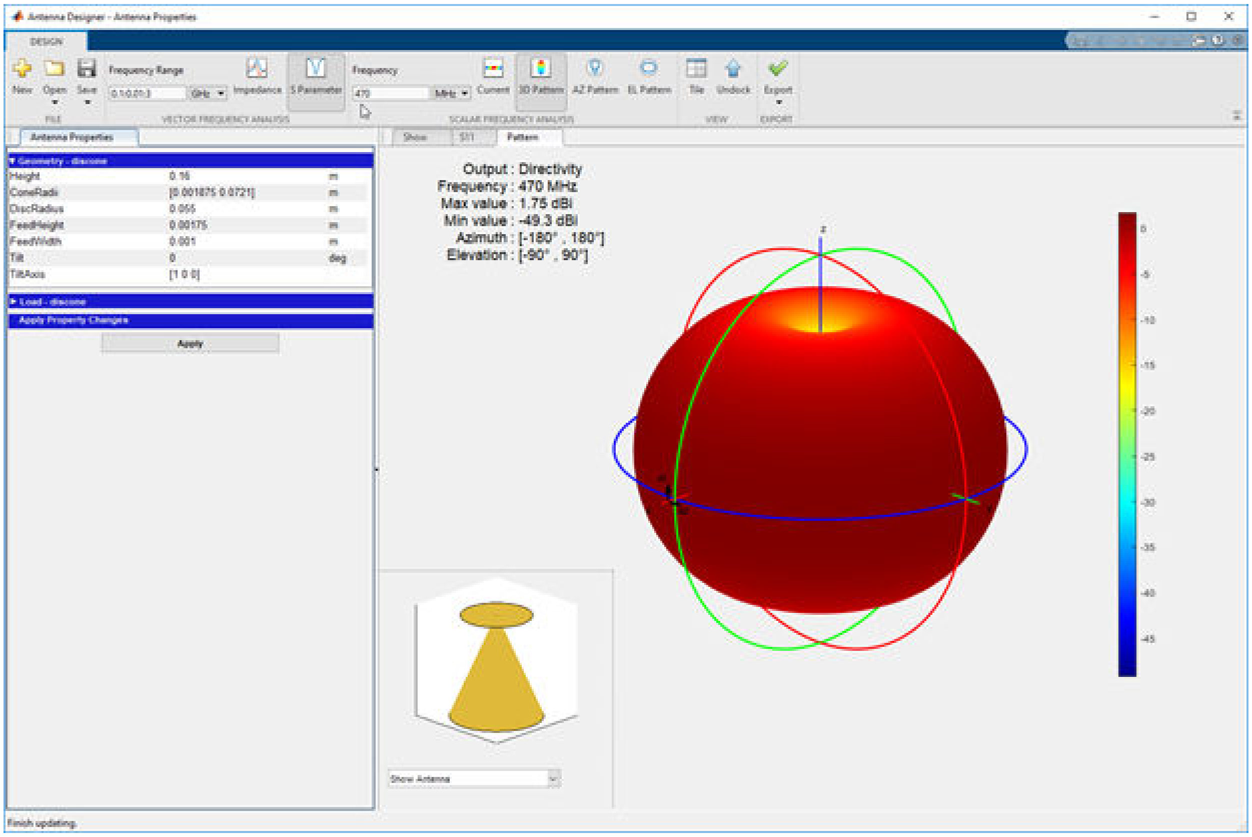
▼ Geometry - discone

Height	0.16	m
ConeRadii	[0.001875 0.0721]	m
DiscRadius	0.055	m
FeedHeight	0.00175	m
FeedWidth	0.001	m
Tilt	0	deg
TiltAxis	[1 0 0]	

▶ Load - discone

Apply Property Changes

Plot the radiation pattern of the antenna at the specified frequency.



Geometry - discone

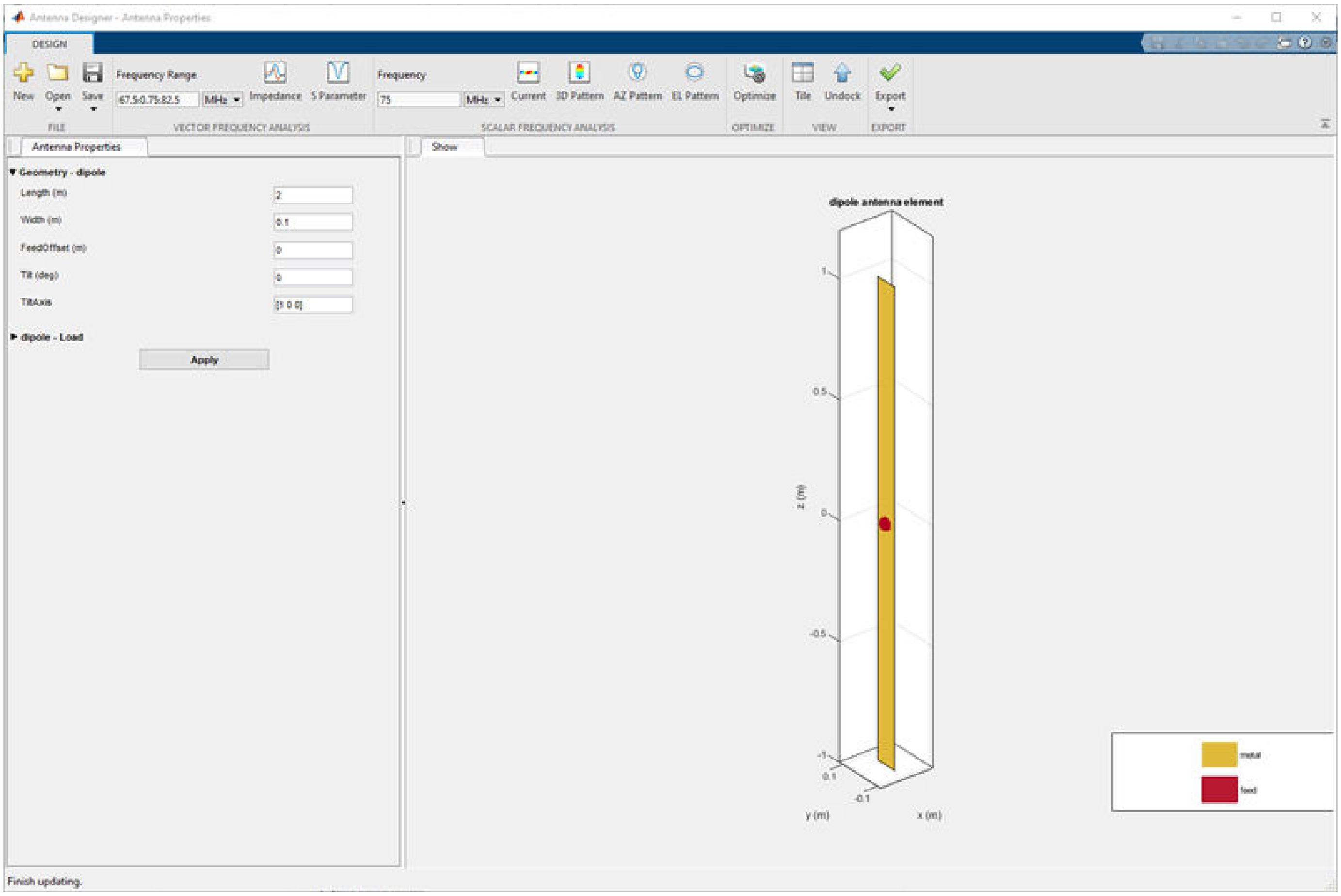
Height	0.16	m
ConeRadii	[0.001875 0.0721]	m
DiscRadius	0.055	m
FeedHeight	0.00175	m
FeedWidth	0.001	m
Tilt	0	deg
TiltAxis	[1 0 0]	

Output : Directivity
Frequency : 470 MHz
Max value : 1.75 dBi
Min value : -49.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

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.



Frequency Range 67.5:0.75:82.5 MHz

Frequency 75 MHz

Geometry - dipole

Length (m) 2

Width (m) 0.1

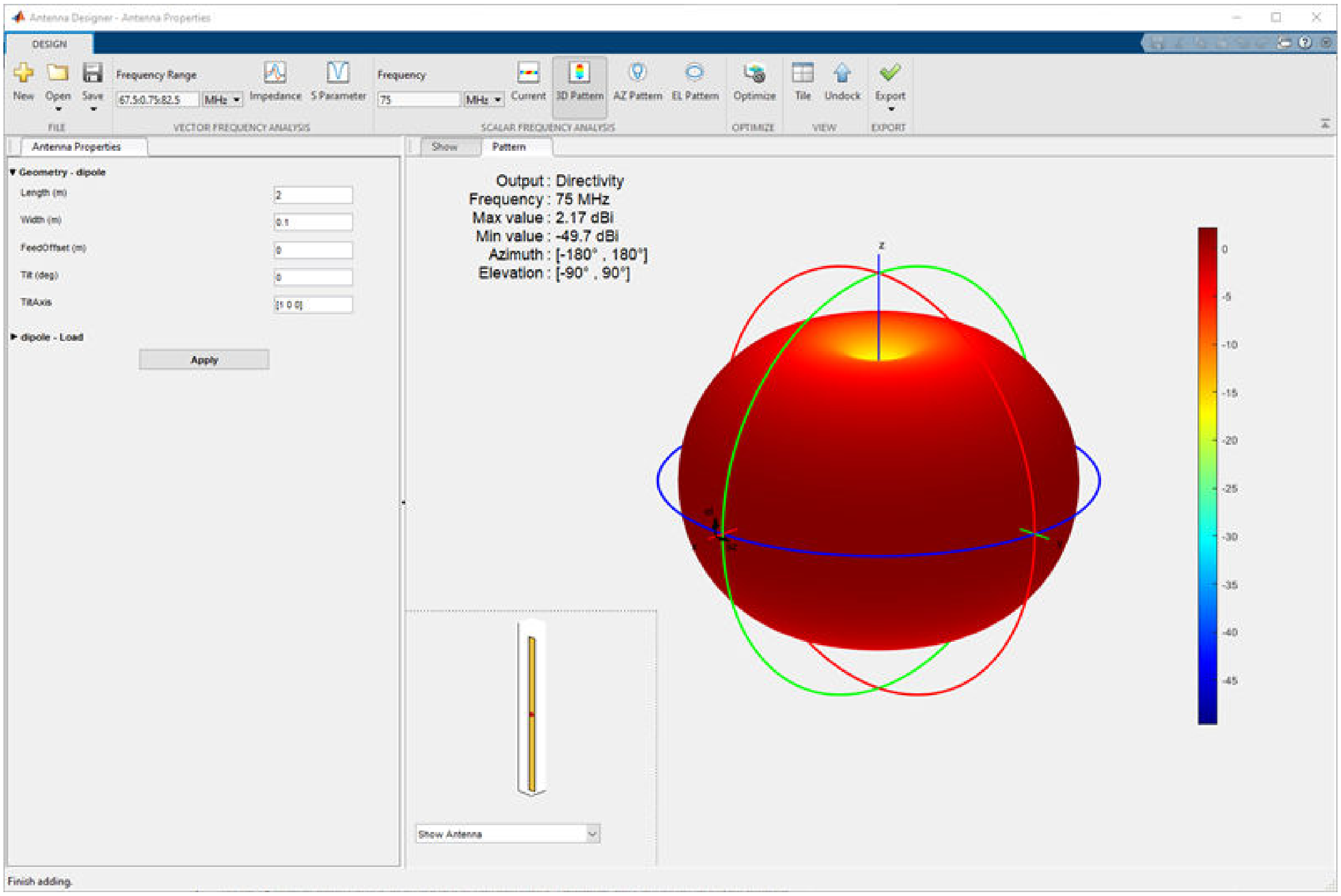
FeedOffset (m) 0

Tilt (deg) 0

TiltAxis [1 0 0]

Finish updating.

Analyze the pattern of the antenna. Notice that the Max value for directivity in the plot is 2.17 dBi.



Frequency Range: 67.5:0.75:82.5 MHz

Frequency: 75 MHz

▼ Geometry - dipole

Length (m): 2

Width (m): 0.1

FeedOffset (m): 0

Tilt (deg): 0

TiltAxis: [1 0 0]

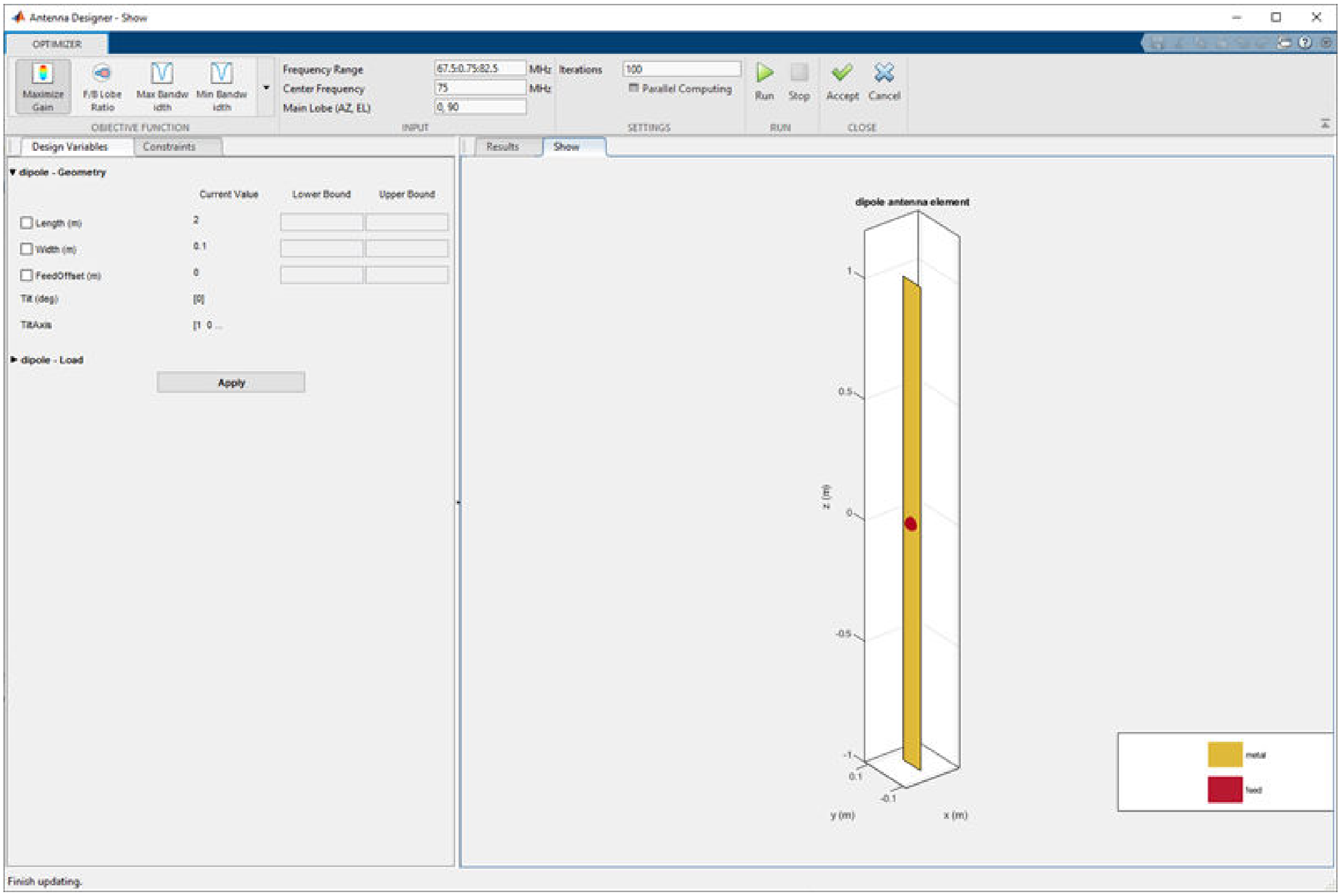
▶ dipole - Load

```
    Output : Directivity
 Frequency : 75 MHz
 Max value : 2.17 dBi
 Min value : -49.7 dBi
   Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

Finish 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

Iterations 100

□ Parallel Computing

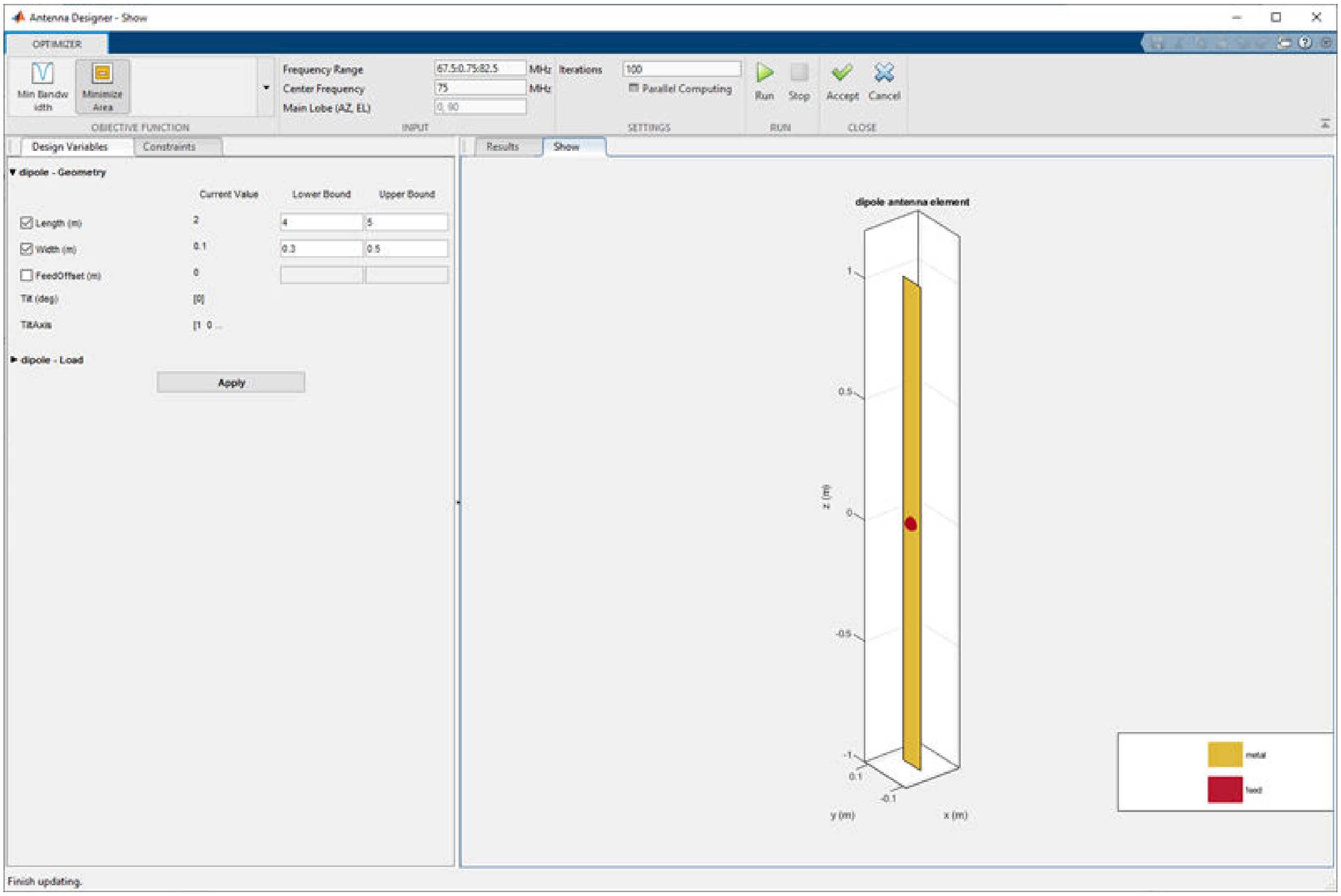
▼ dipole - Geometry

	Current Value	Lower Bound	Upper Bound
□ Length (m)	2		
□ Width (m)	0.1		
□ FeedOffset (m)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		

▶ dipole - Load

Finish updating.

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.**



Frequency Range 67.5:0.75:82.5 MHz

Center Frequency 75 MHz

Main Lobe (AZ, EL) 0, 90

Iterations 100

□ Parallel Computing

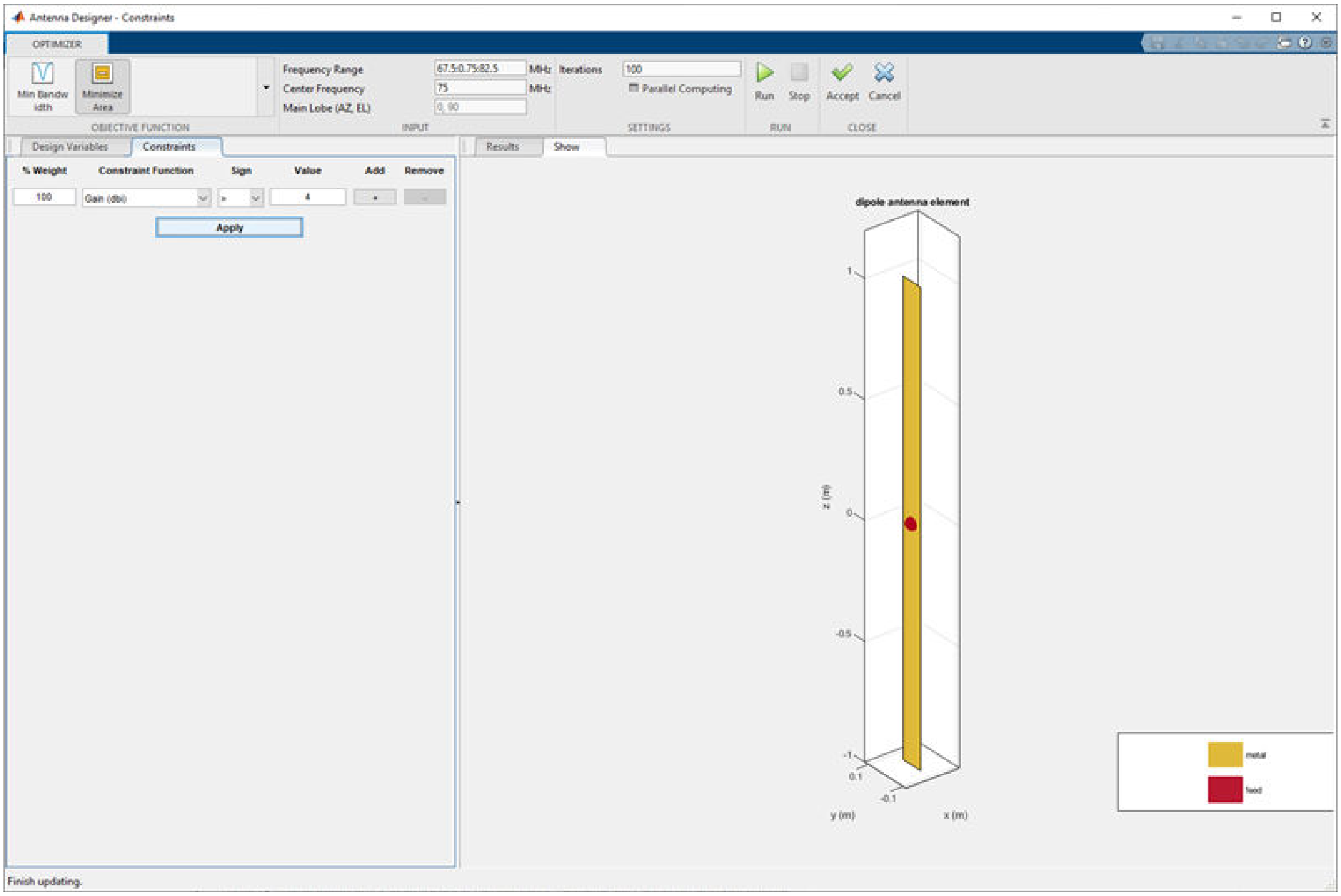
▼ dipole - Geometry

	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 - Load

Finish updating.

Enter the constraints in the `Constraints` tab. Click **Apply**.



Frequency Range 67.5:0.75:82.5 MHz

Center Frequency 75 MHz

Main Lobe (AZ, EL) 0, 90

Iterations 100

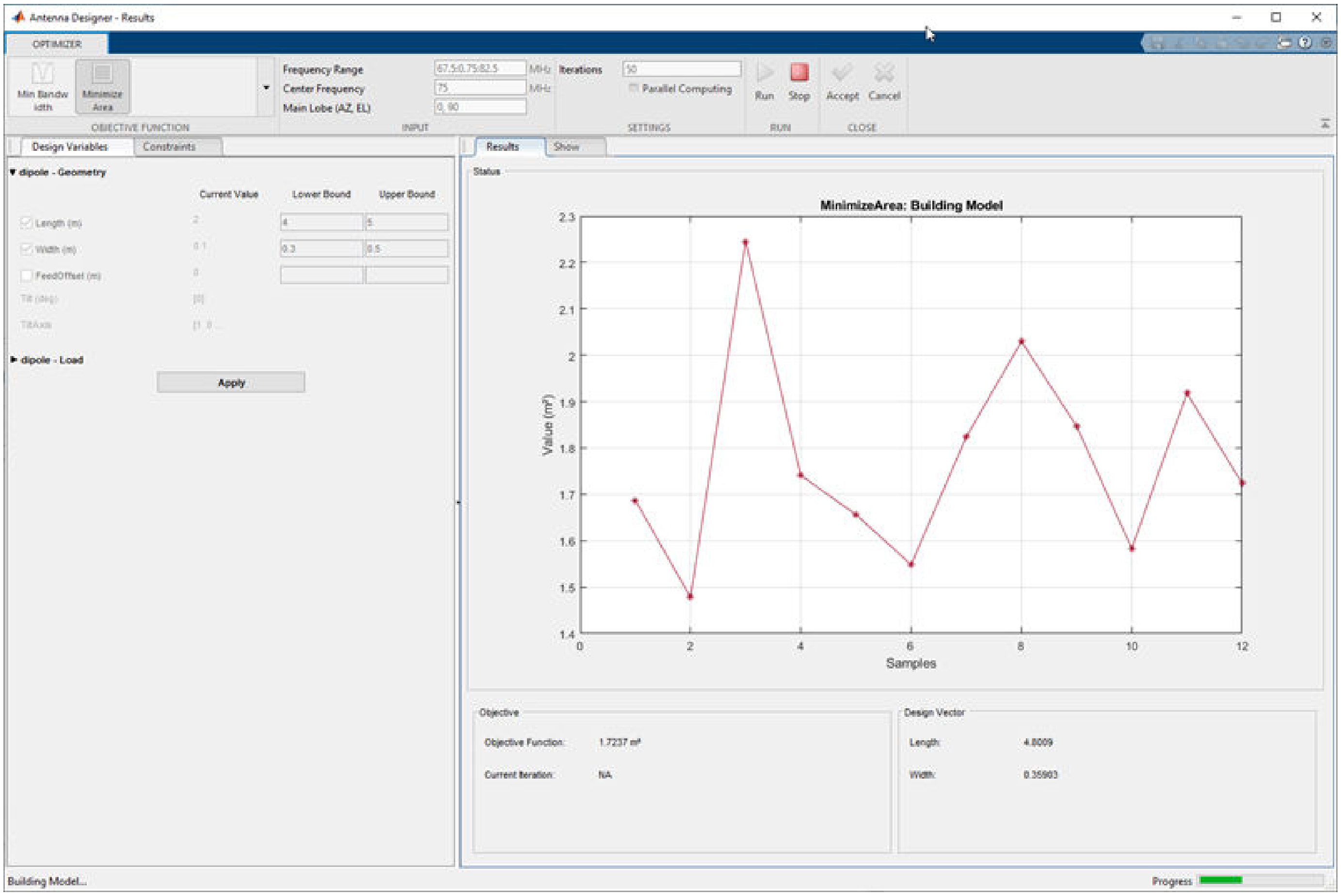
□ Parallel Computing

% Weight	Constraint Function	Sign	Value	Add	Remove
100	Gain (dbi)	>	4	+	-

Finish updating.

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

Iterations 50
□ Parallel Computing

▼ 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 - Load

Objective

Objective Function: 1.7237 m²

Current Iteration: NA

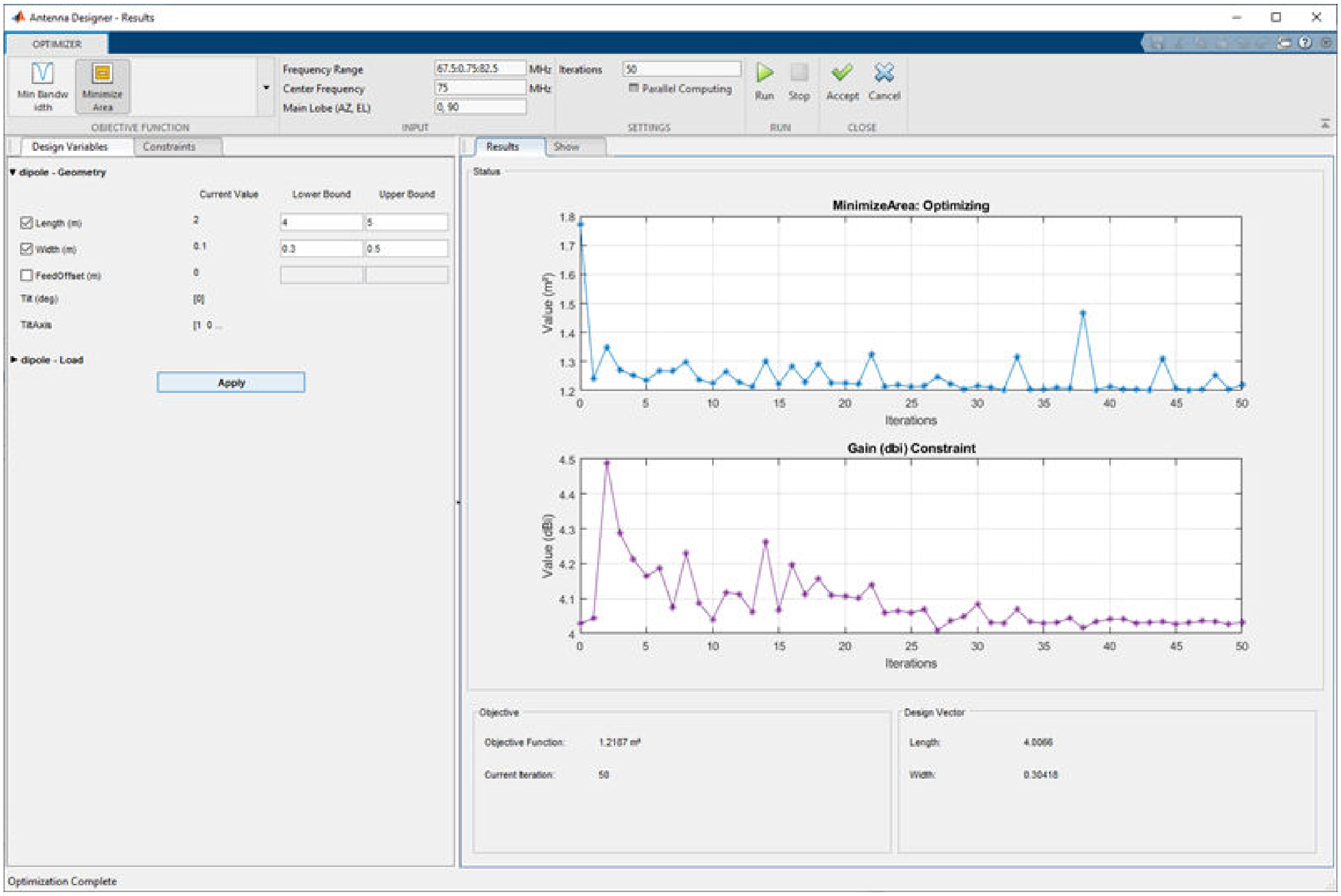
Design Vector

Length: 4.8009

Width: 0.35903

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
Iterations 50
□ Parallel Computing

▼ 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 - Load

Objective

Objective Function: 1.2187 m²

Current Iteration: 50

Design Vector

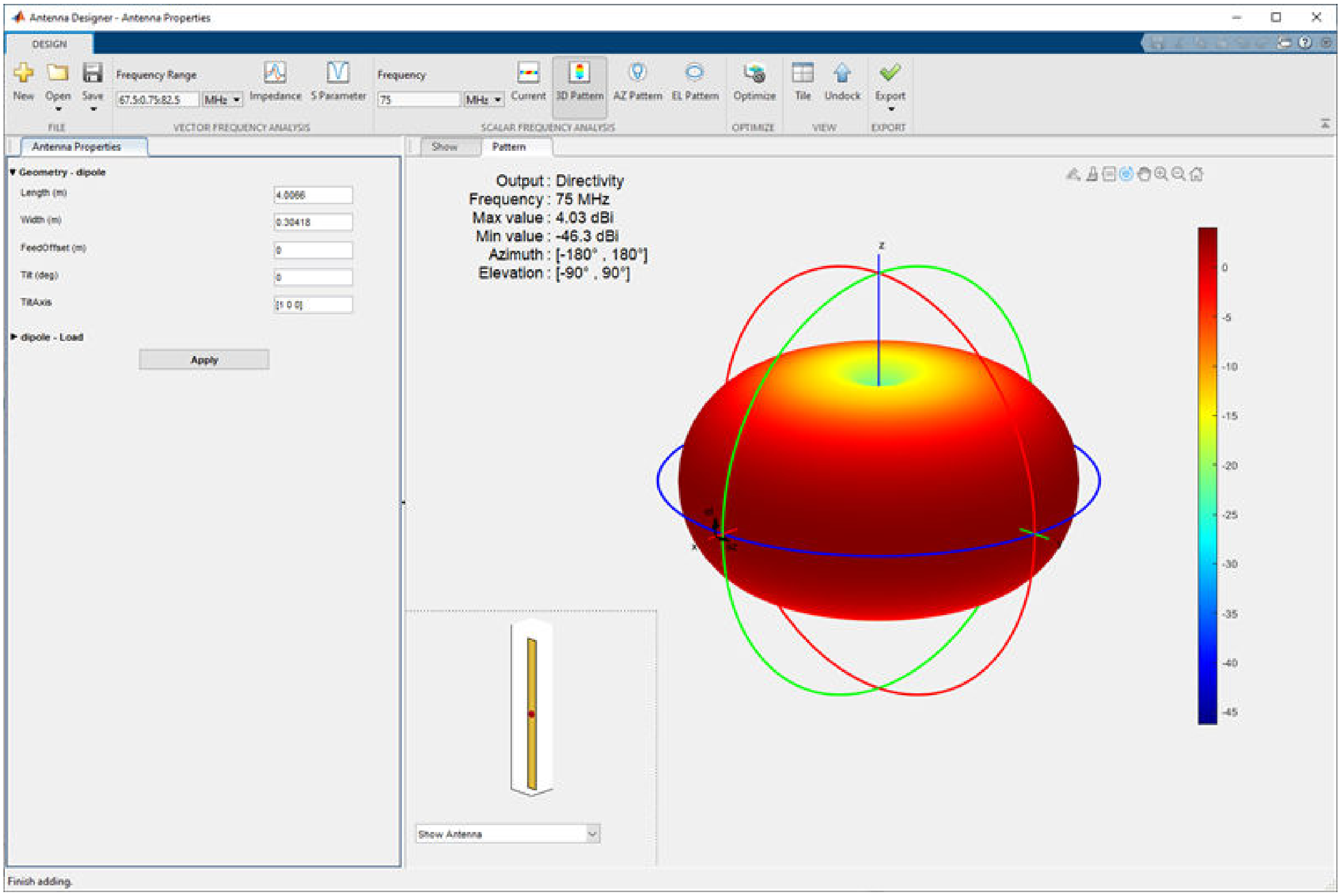
Length: 4.0066

Width: 0.30418

Optimization Complete

Click **Accept**.

Analyze the antenna again for the 3D pattern. See that the Max value of the directivity is now 4.03 dBi.



Antenna Designer - Antenna Properties

Frequency Range: 67.5:0.75:82.5 MHz

Frequency: 75 MHz

Geometry - dipole

Length (m): 4.0066

Width (m): 0.30418

FeedOffset (m): 0

Tilt (deg): 0

TiltAxis: [1 0 0]

Output : Directivity
Frequency : 75 MHz
Max value : 4.03 dBi
Min value : -46.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Finish adding.

- “Design and Analysis Using Antenna Designer App”
- “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.

See Also

Topics

“Design and Analysis Using Antenna Designer App”
“Maximizing Gain and Improving Impedance Bandwidth of E-Patch Antenna”
“Antenna Optimization Algorithm”

Introduced in R2017a

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 the antenna array elements using objective functions, design variables, and constraints.

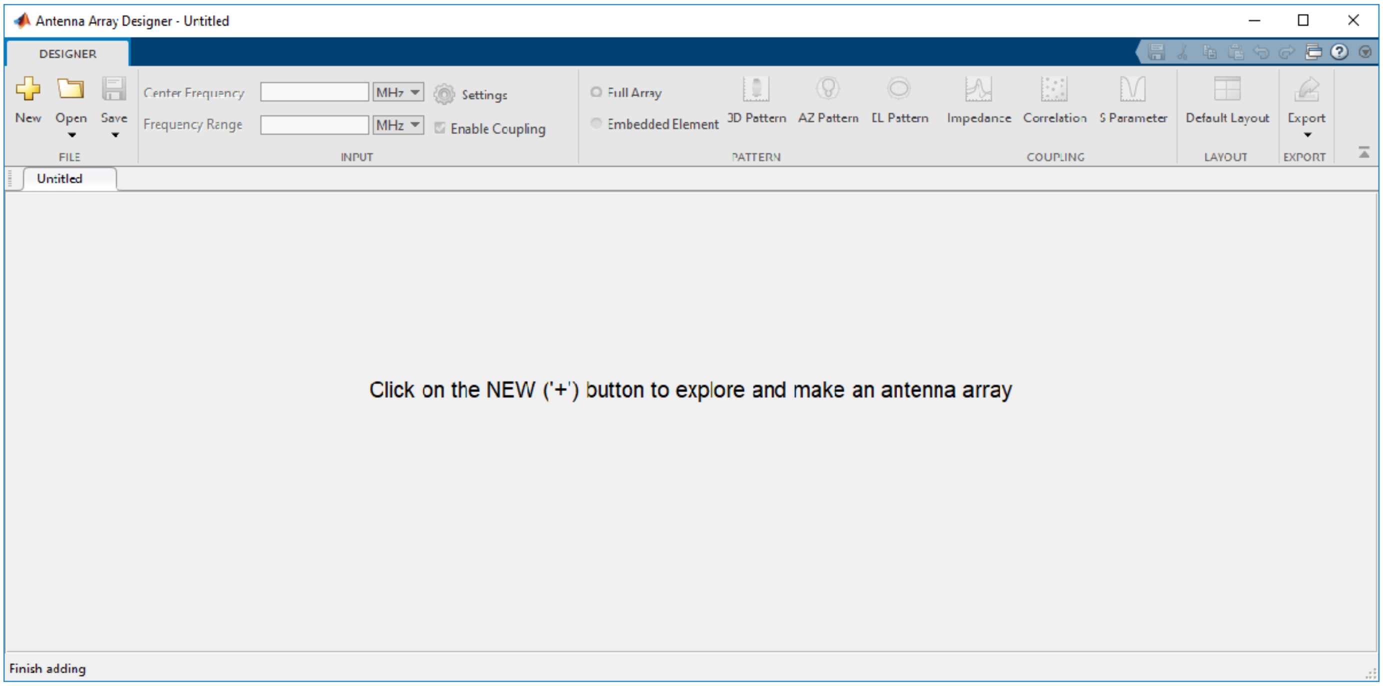
Open the Antenna Array Designer App

- 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:



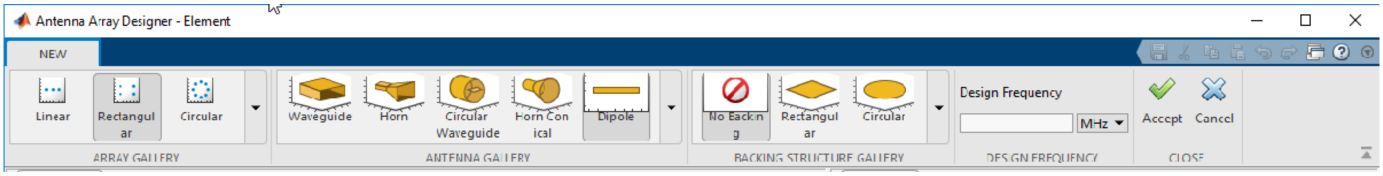
Click on the NEW ('+') button to explore and make an antenna array

Finish adding

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.



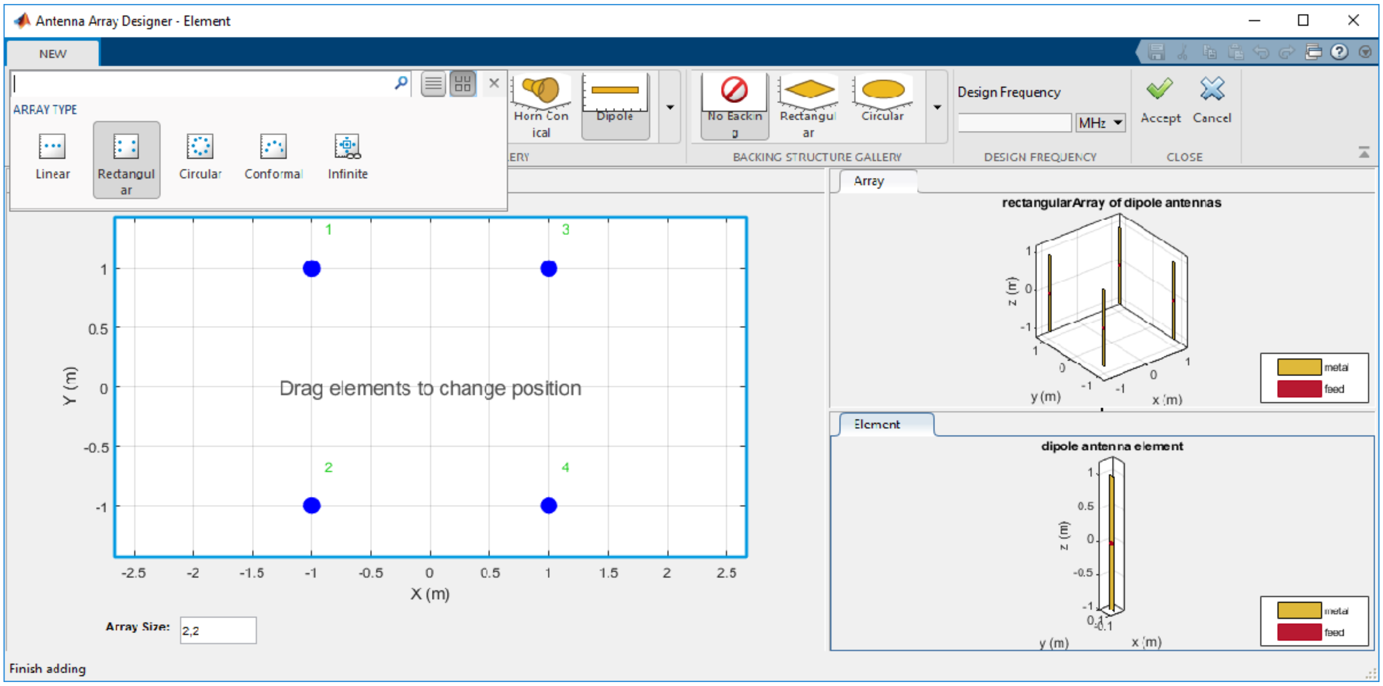
Using the toolstrip, you can choose different types of array layouts, antennas, and backing structures.

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**.



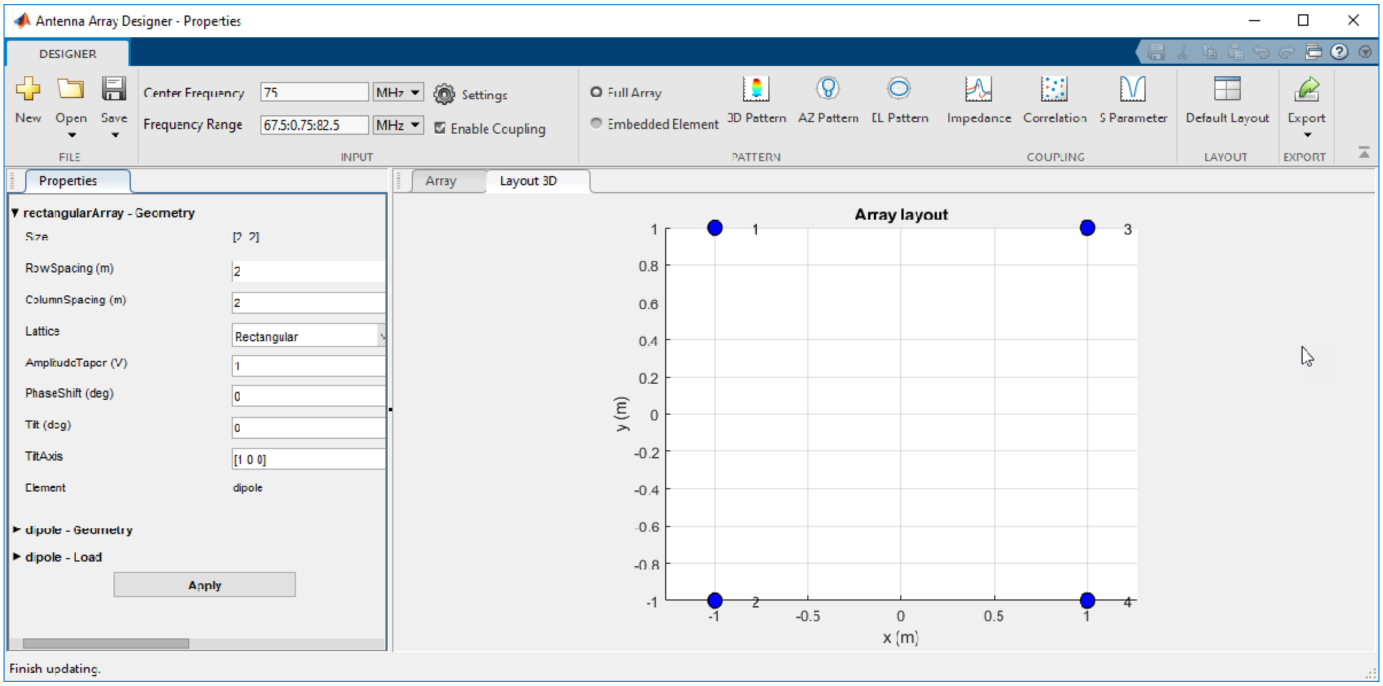
Array Size: 2,2

Finish adding

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



Center Frequency 75 MHz

Frequency Range 67.5:0.75:82.5 MHz

☑ Enable Coupling

▾ rectangularArray - Geometry

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

▸ dipole - Geometry

▸ dipole - Load

Finish 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.

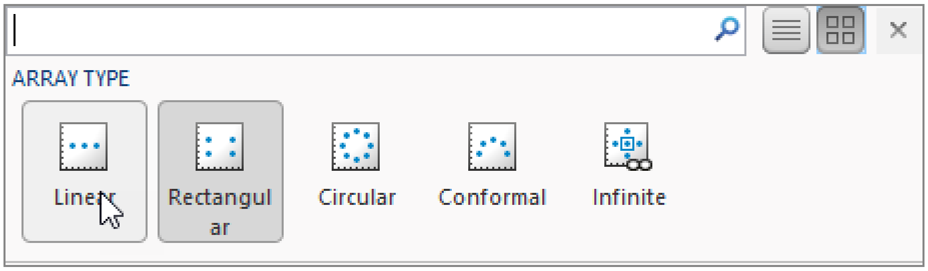
Linear Dipole Array and Maximum Directivity

Open the **Antenna Array Designer** app.

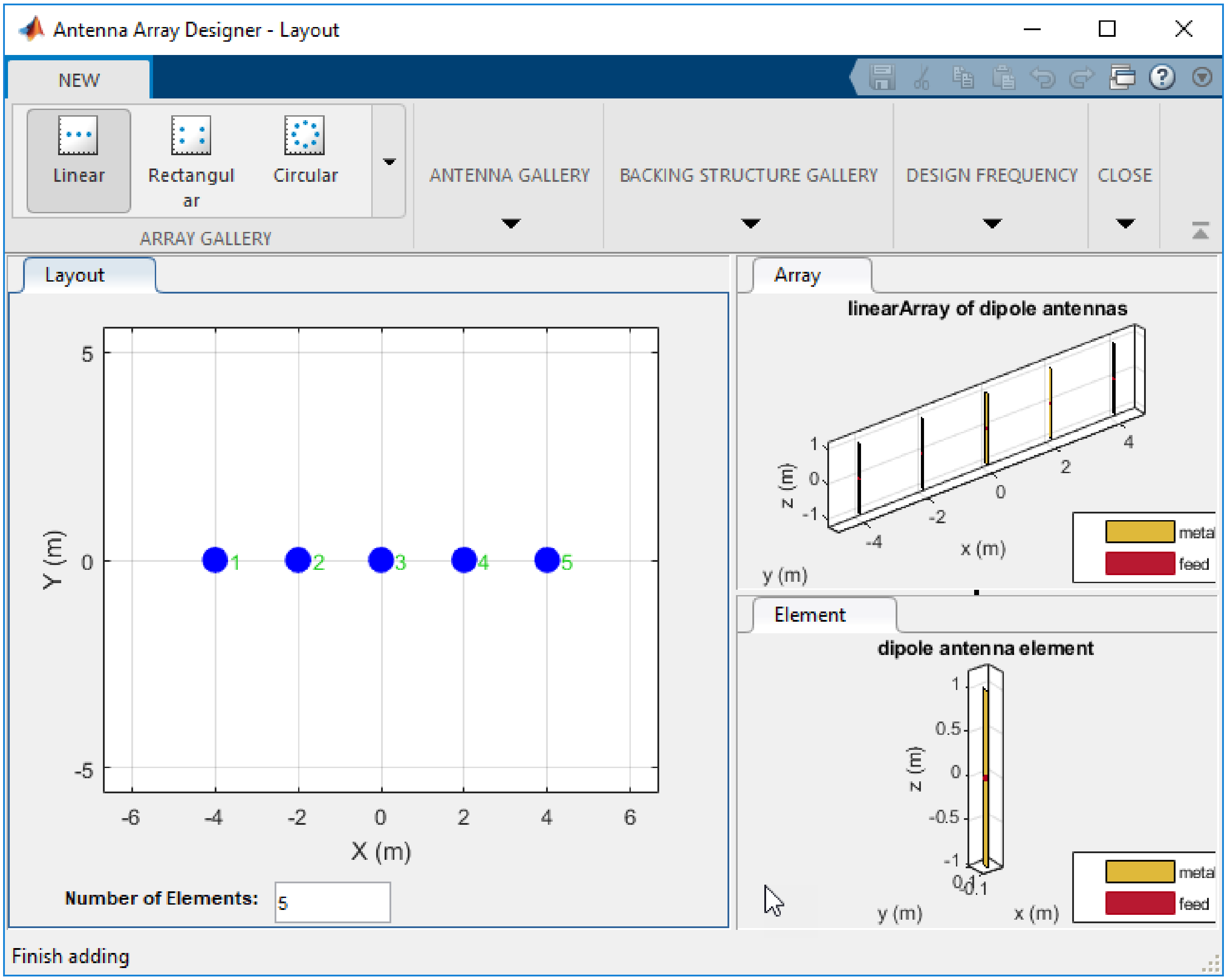
```
antennaArrayDesigner
```

Click on the NEW ('+') button to explore and make an antenna array

Click on **New** and from the **Array Type** pane, click **Linear**.



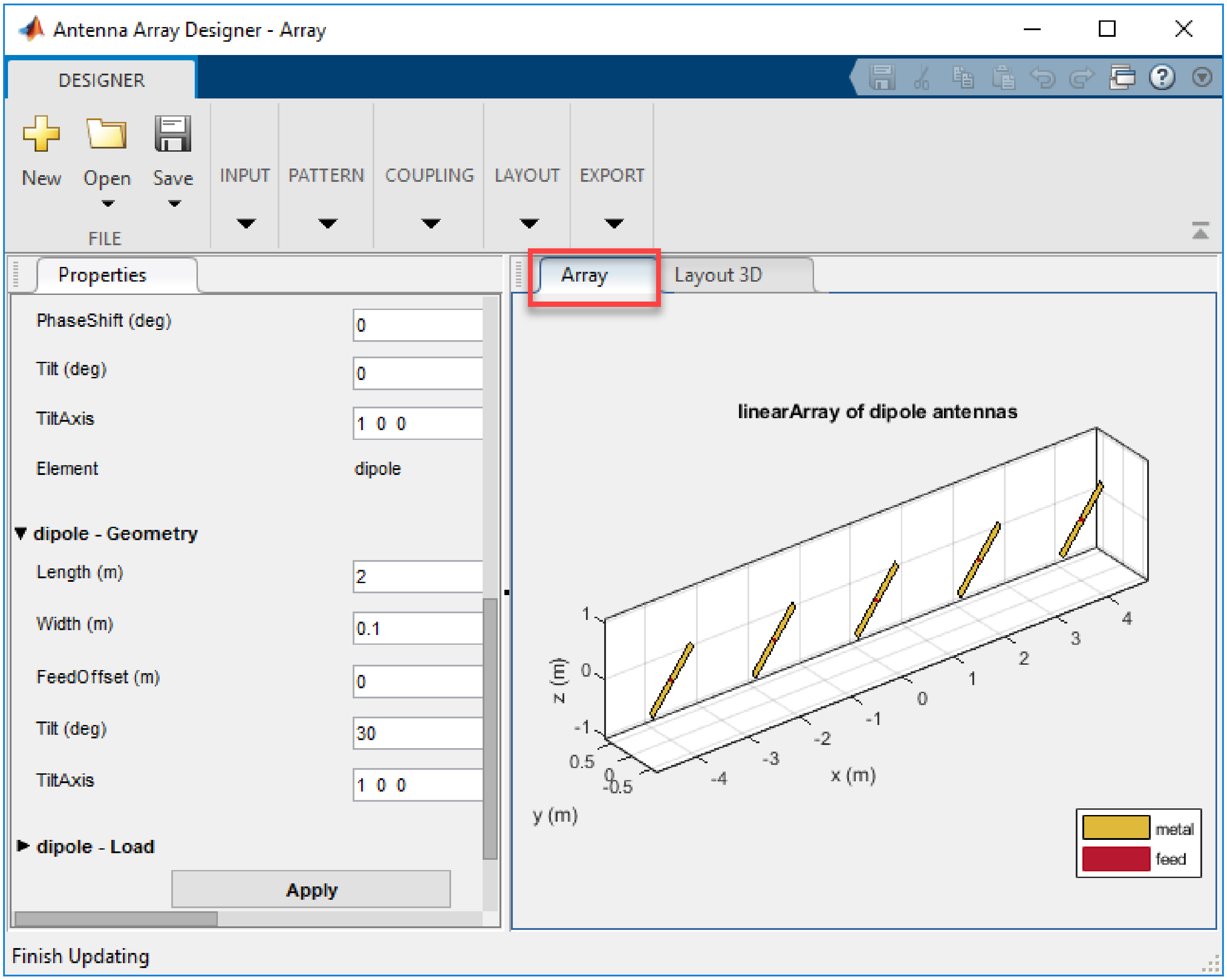
In the bottom left corner, change **Number of Elements** to 5. Click **Accept**.



Number of Elements: 5

Finish adding

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.



PhaseShift (deg) 0

Tilt (deg) 0

TiltAxis 1 0 0

Element dipole

▼ dipole - Geometry

Length (m) 2

Width (m) 0.1

FeedOffset (m) 0

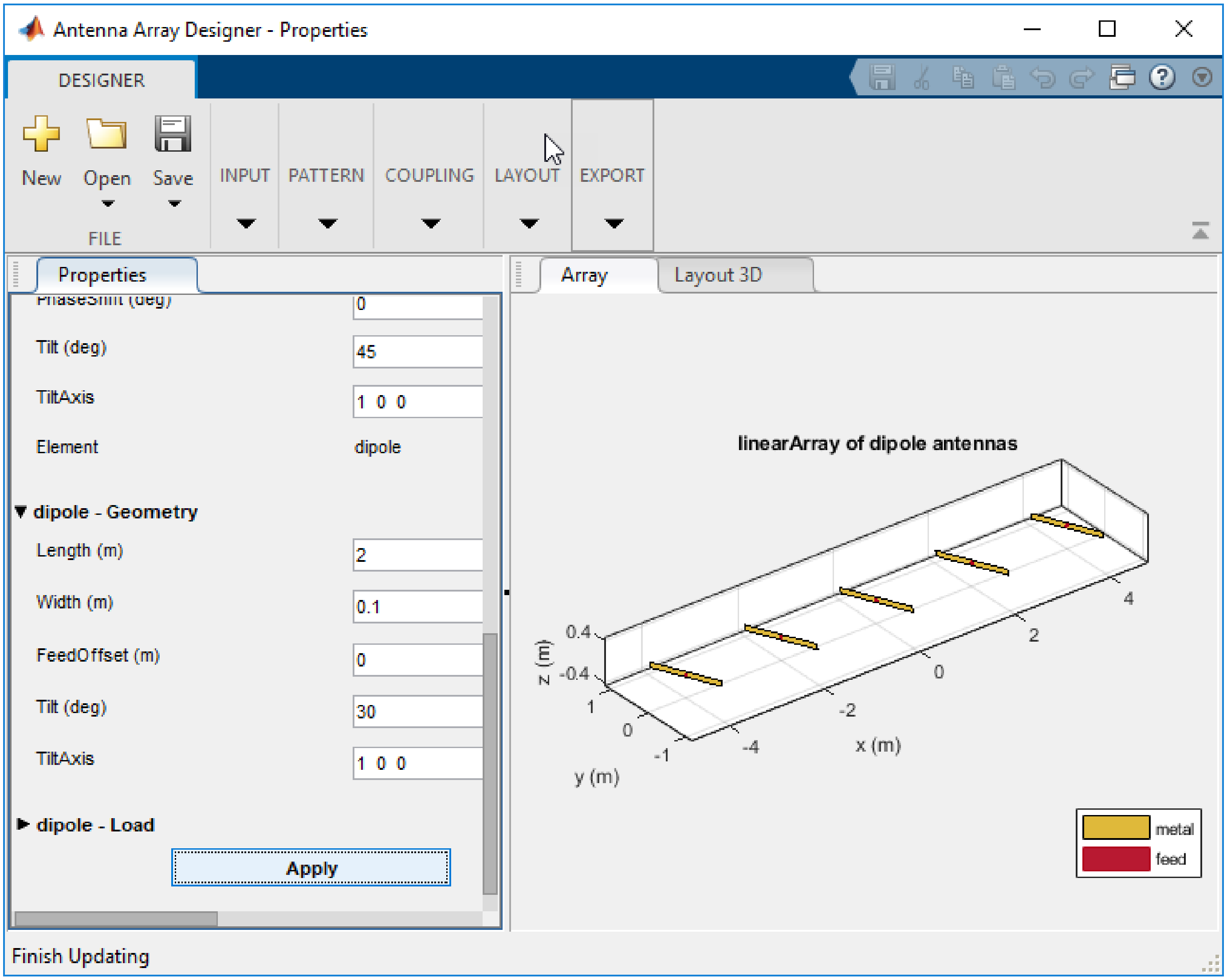
Tilt (deg) 30

TiltAxis 1 0 0

▶ dipole - Load

Finish 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.



PhaseShift (deg) 0

Tilt (deg) 45

TiltAxis 1 0 0

Element dipole

▼ dipole - Geometry

Length (m) 2

Width (m) 0.1

FeedOffset (m) 0

Tilt (deg) 30

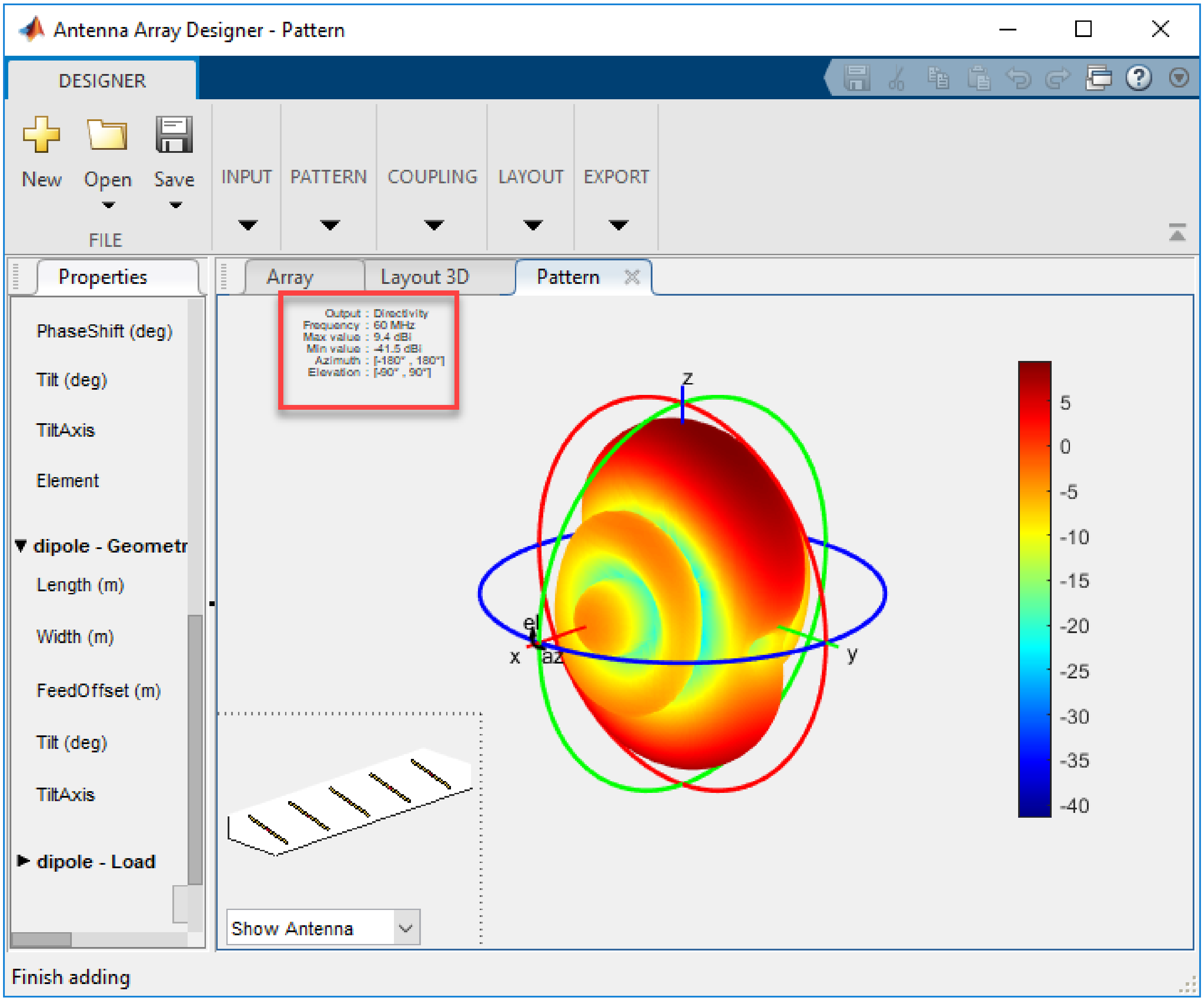
TiltAxis 1 0 0

▶ dipole - Load

Apply

Finish Updating

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.

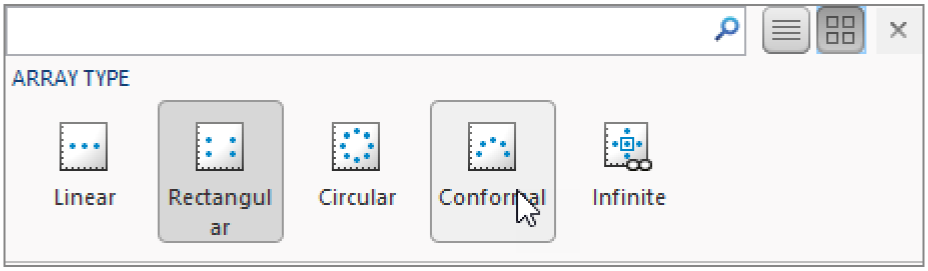


```
    Output : Directivity
 Frequency : 60 MHz
 Max value : 9.4 dBi
 Min value : -41.5 dBi
   Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

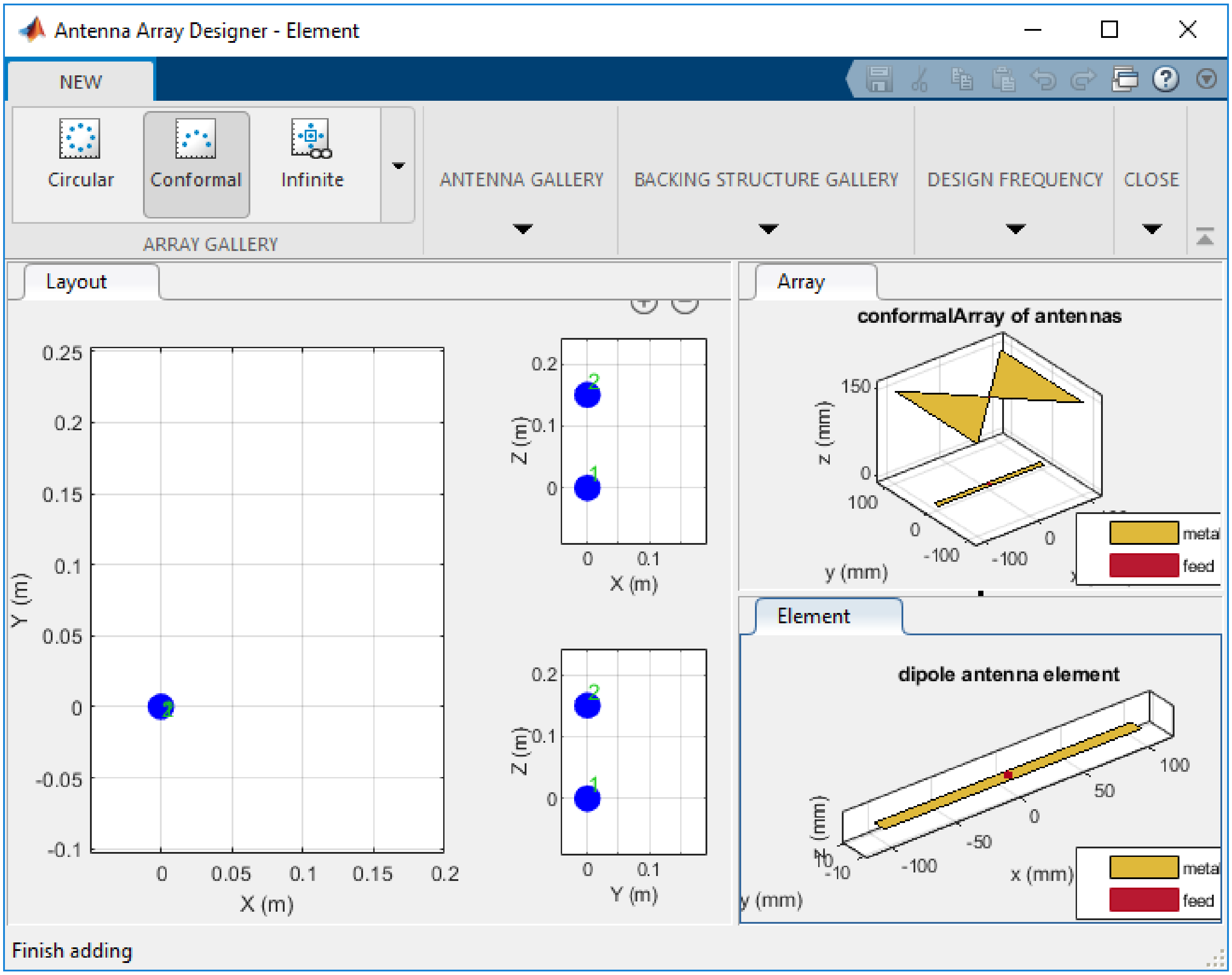
Finish adding

Conformal Array Design and Analysis

Open **Antenna Array Designer** app. In the **Array Gallery** pane, click **Conformal**.



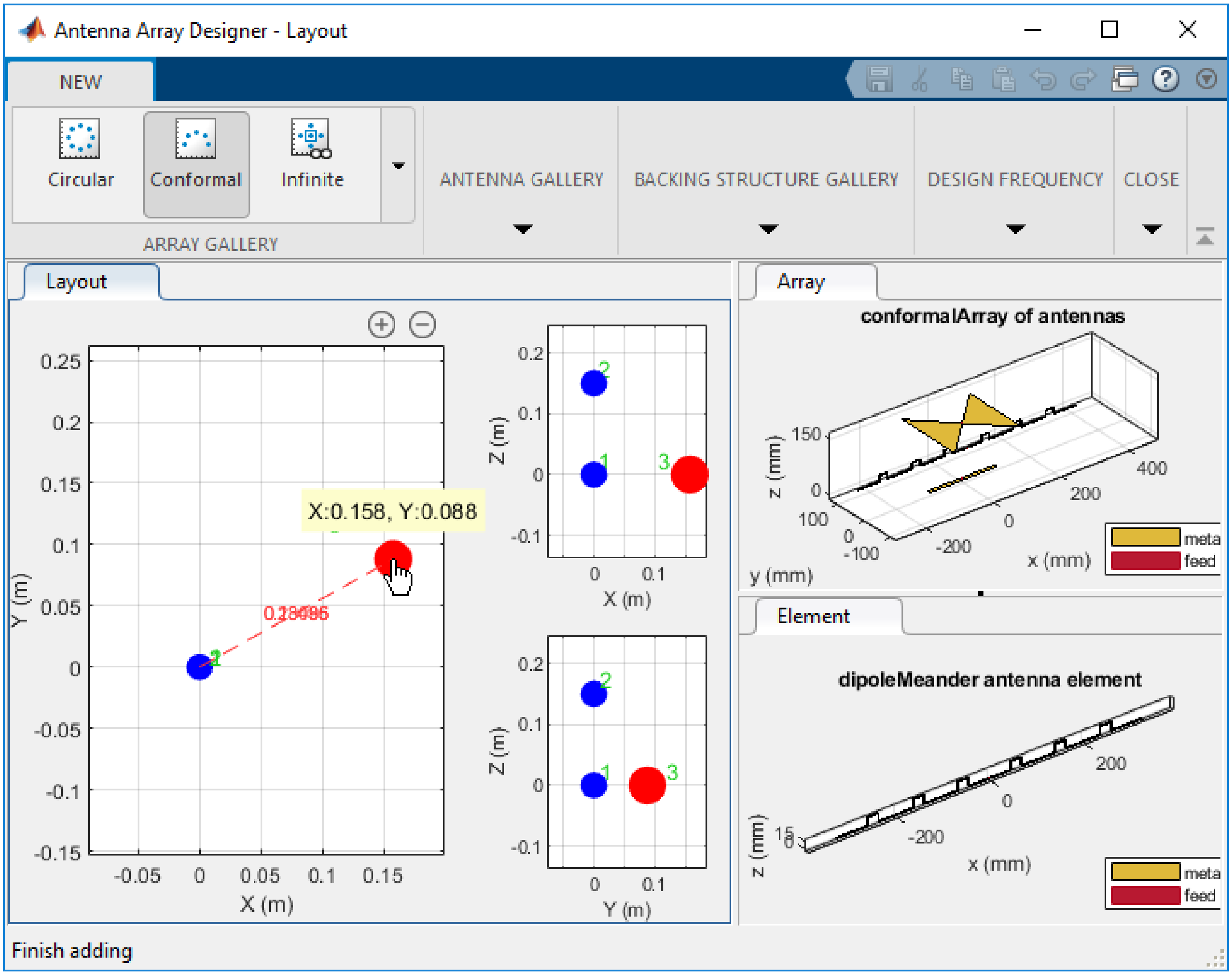
The default conformal array consists of a dipole antenna and a bowtie antenna.



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.

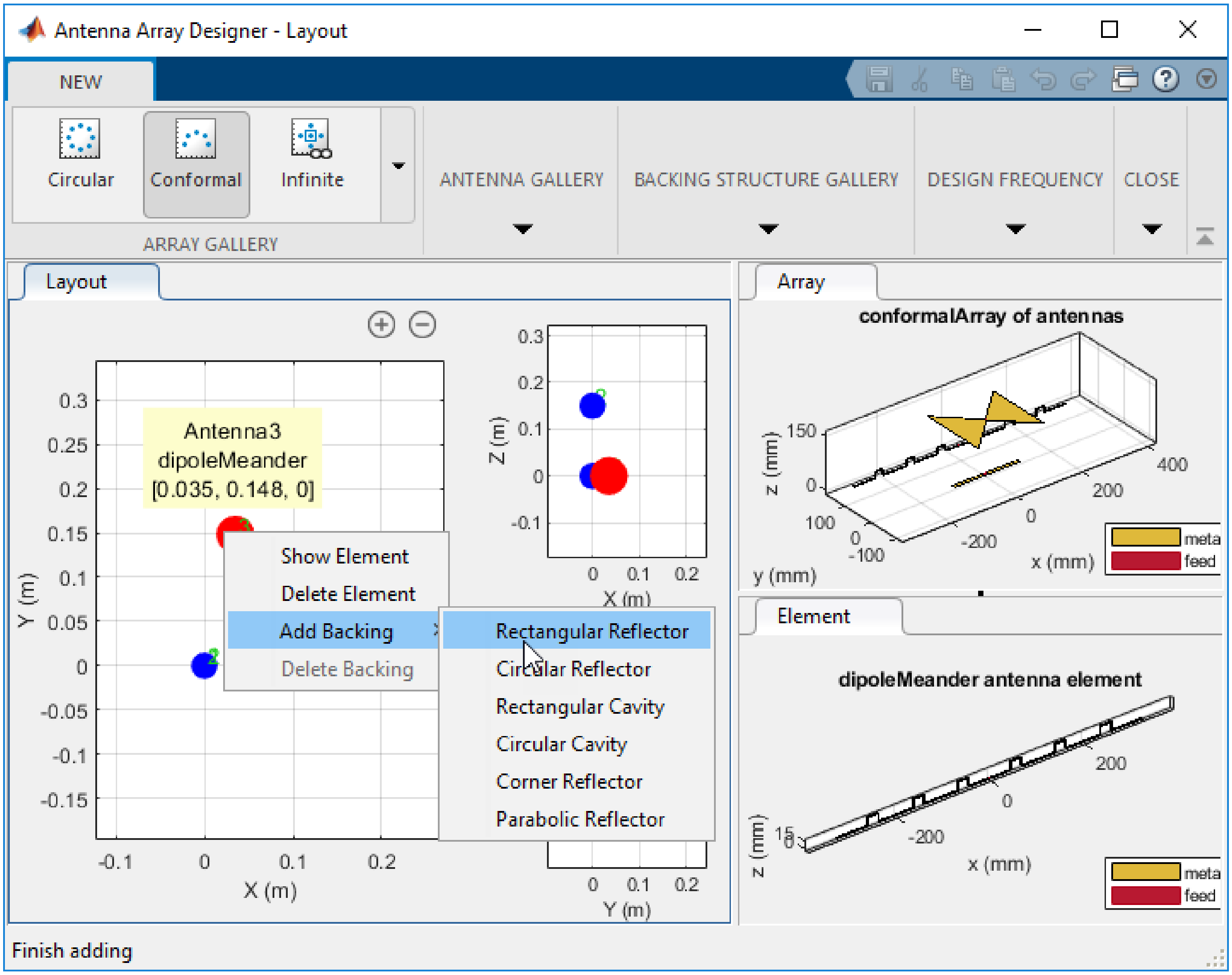


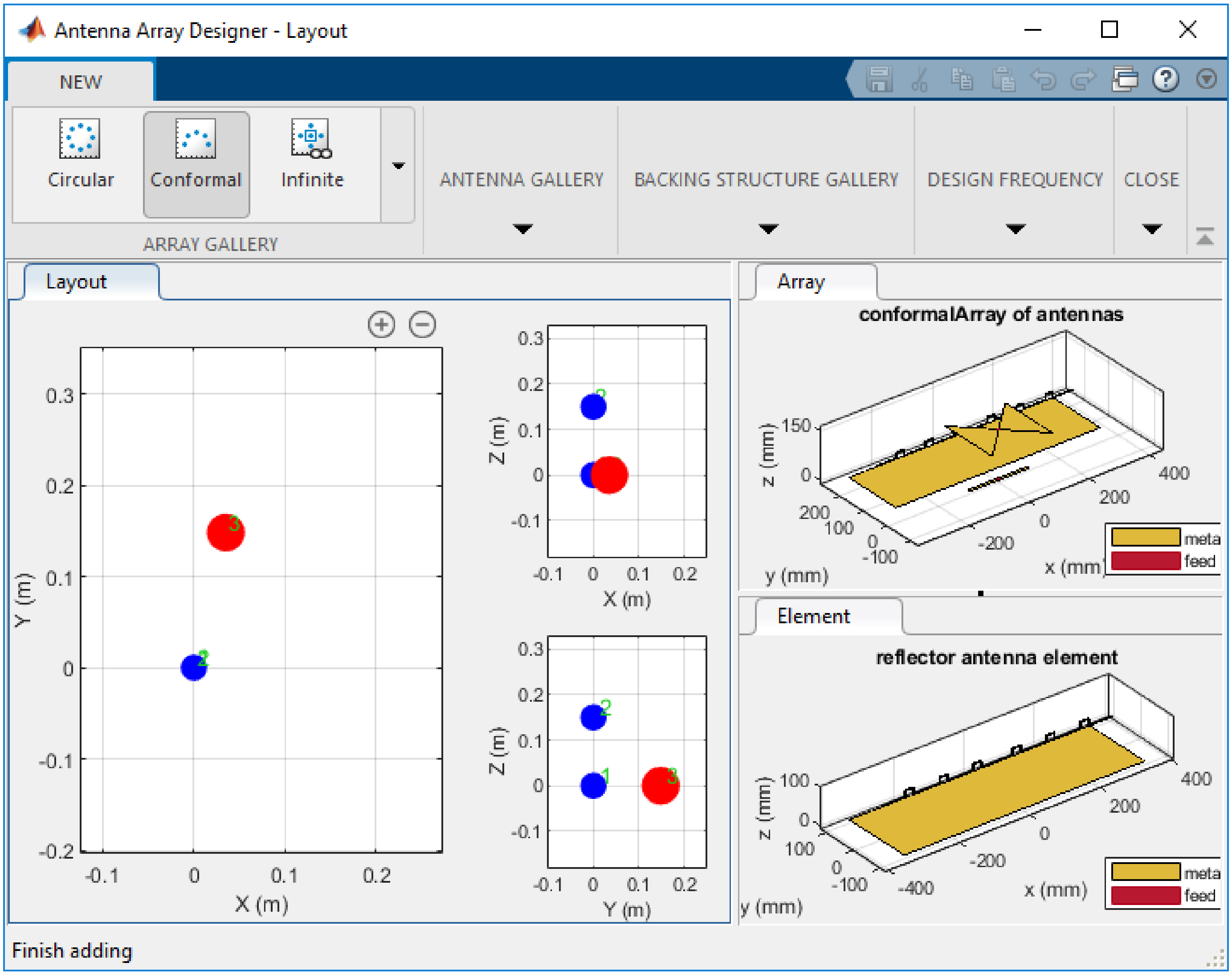
To add the rectangular backing:

- 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**.

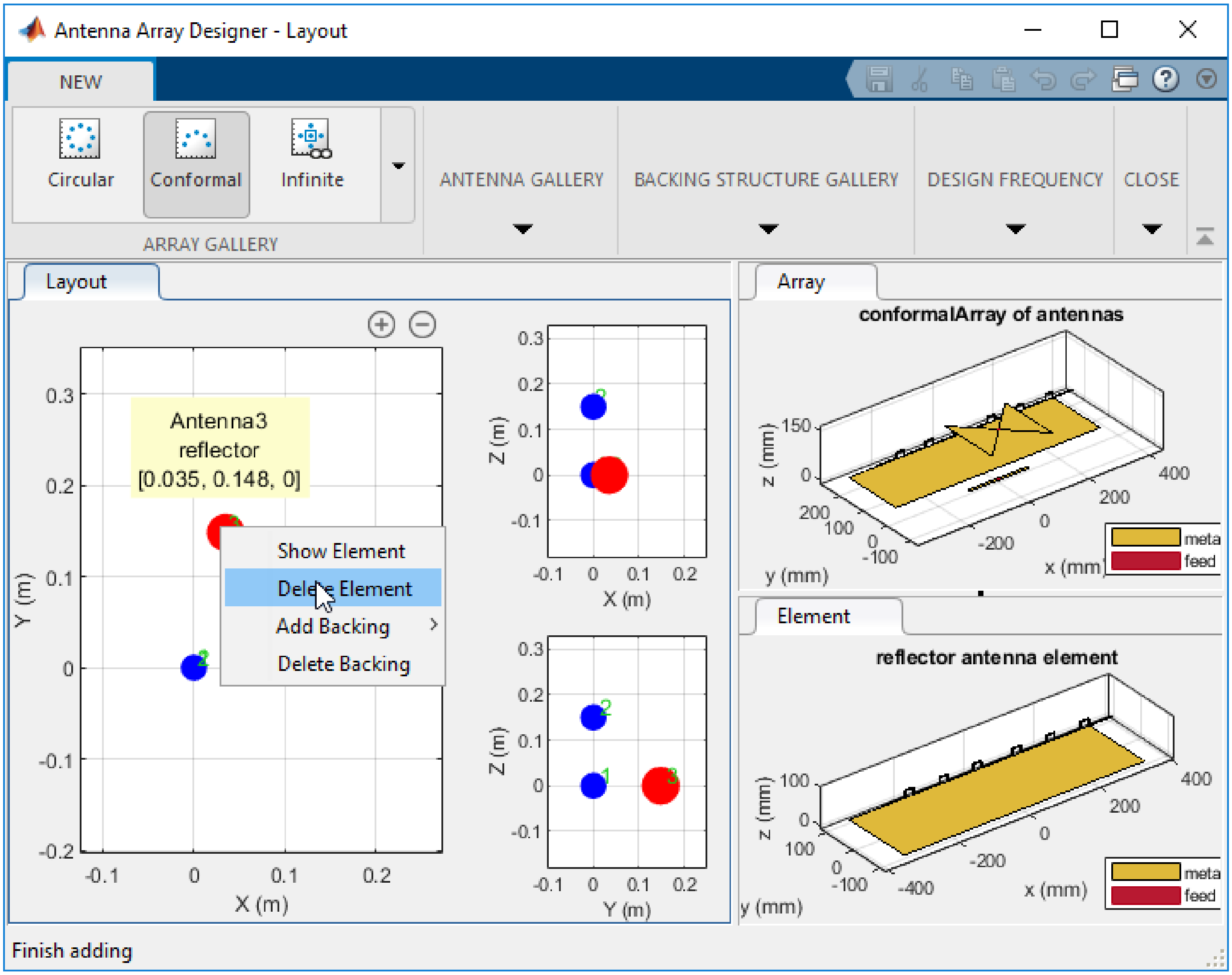

Antenna Array Designer - Layout
NEW
Circular
Conformal
Infinite
ARRAY GALLERY
ANTENNA GALLERY
BACKING STRUCTURE GALLERY
DESIGN FREQUENCY
CLOSE
Layout
Antenna3
dipoleMeander
[0.035, 0.148, 0]
Show Element
Delete Element
Add Backing
Delete Backing
Rectangular Reflector
Circular Reflector
Rectangular Cavity
Circular Cavity
Corner Reflector
Parabolic Reflector
Y (m)
X (m)
Z (m)
Array
conformalArray of antennas
z (mm)
y (mm)
x (mm)
meta
feed
Element
dipoleMeander antenna element
Finish adding



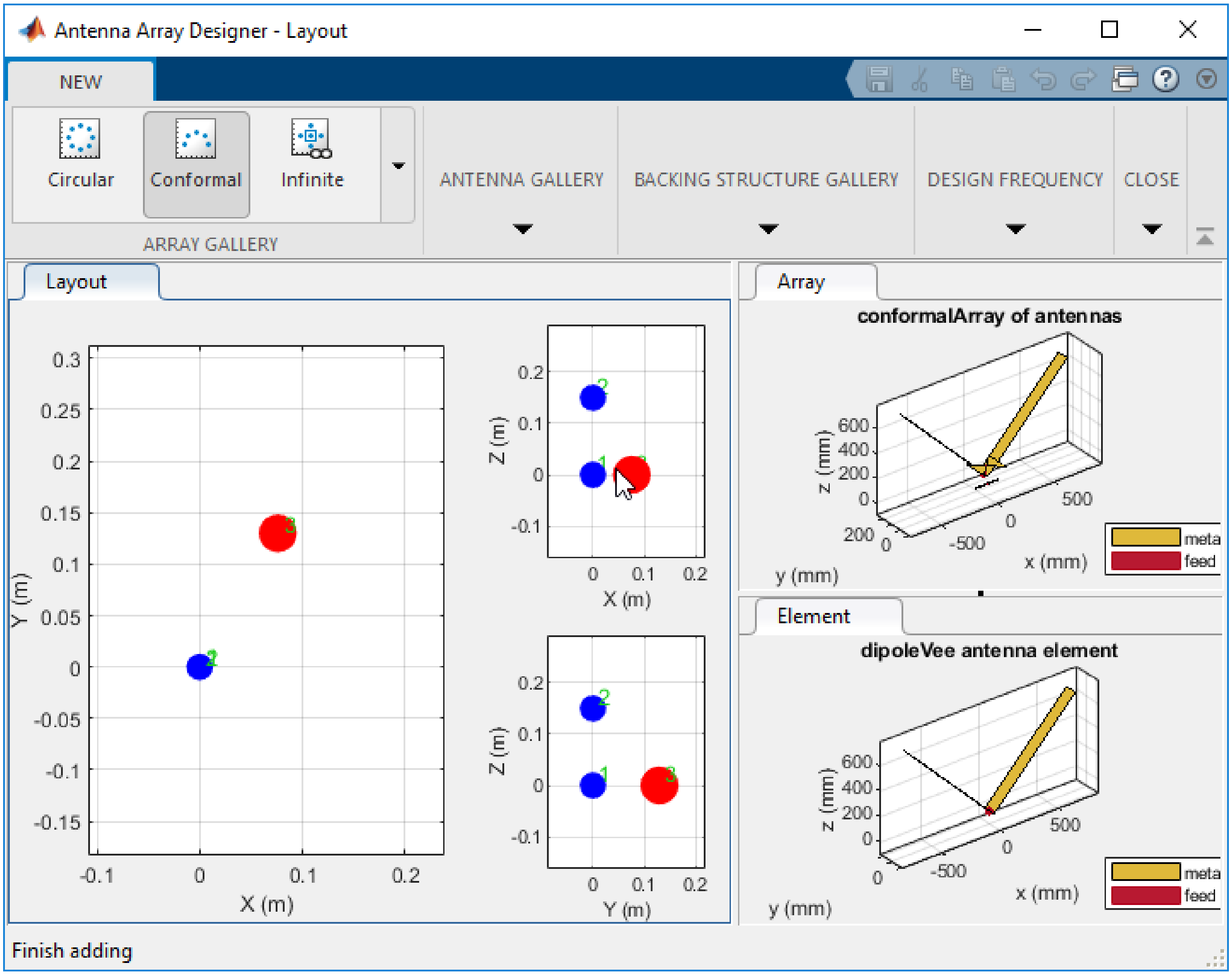
Finish adding

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.



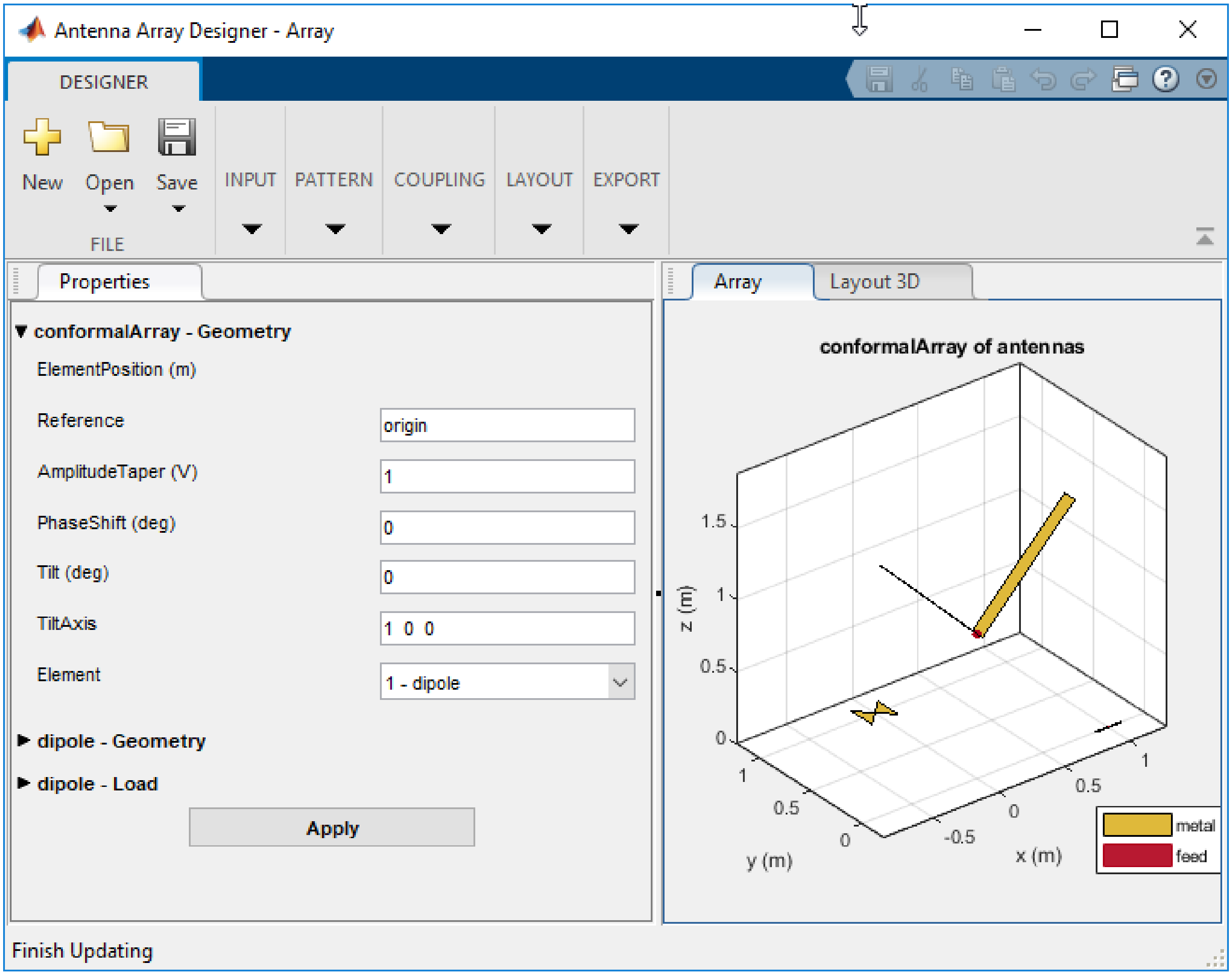
Click **Accept**.

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**.



▼ conformalArray - Geometry

ElementPosition (m)

Reference: origin

AmplitudeTaper (V): 1

PhaseShift (deg): 0

Tilt (deg): 0

TiltAxis: 1 0 0

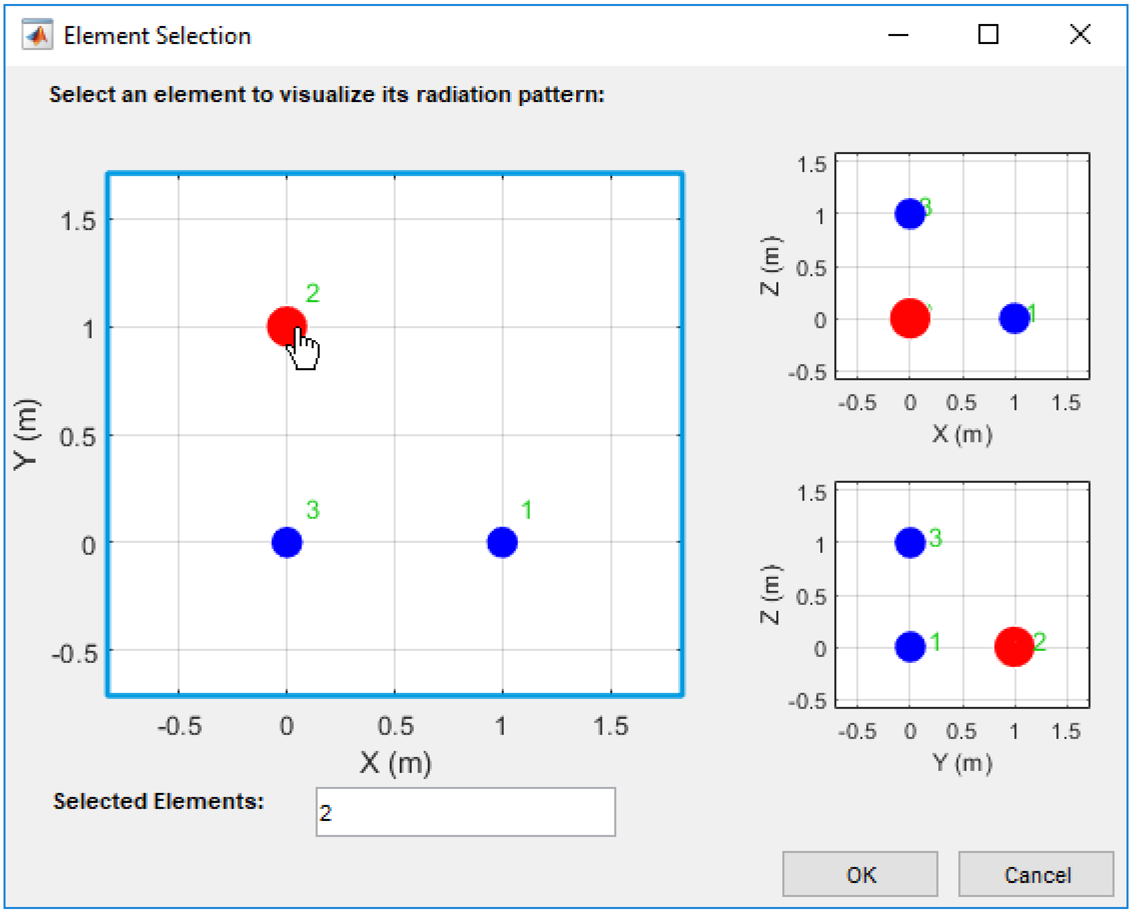
Element: 1 - dipole

▶ dipole - Geometry

▶ dipole - Load

Embedded Element Pattern and Half-Power Beam Width (HPBW)

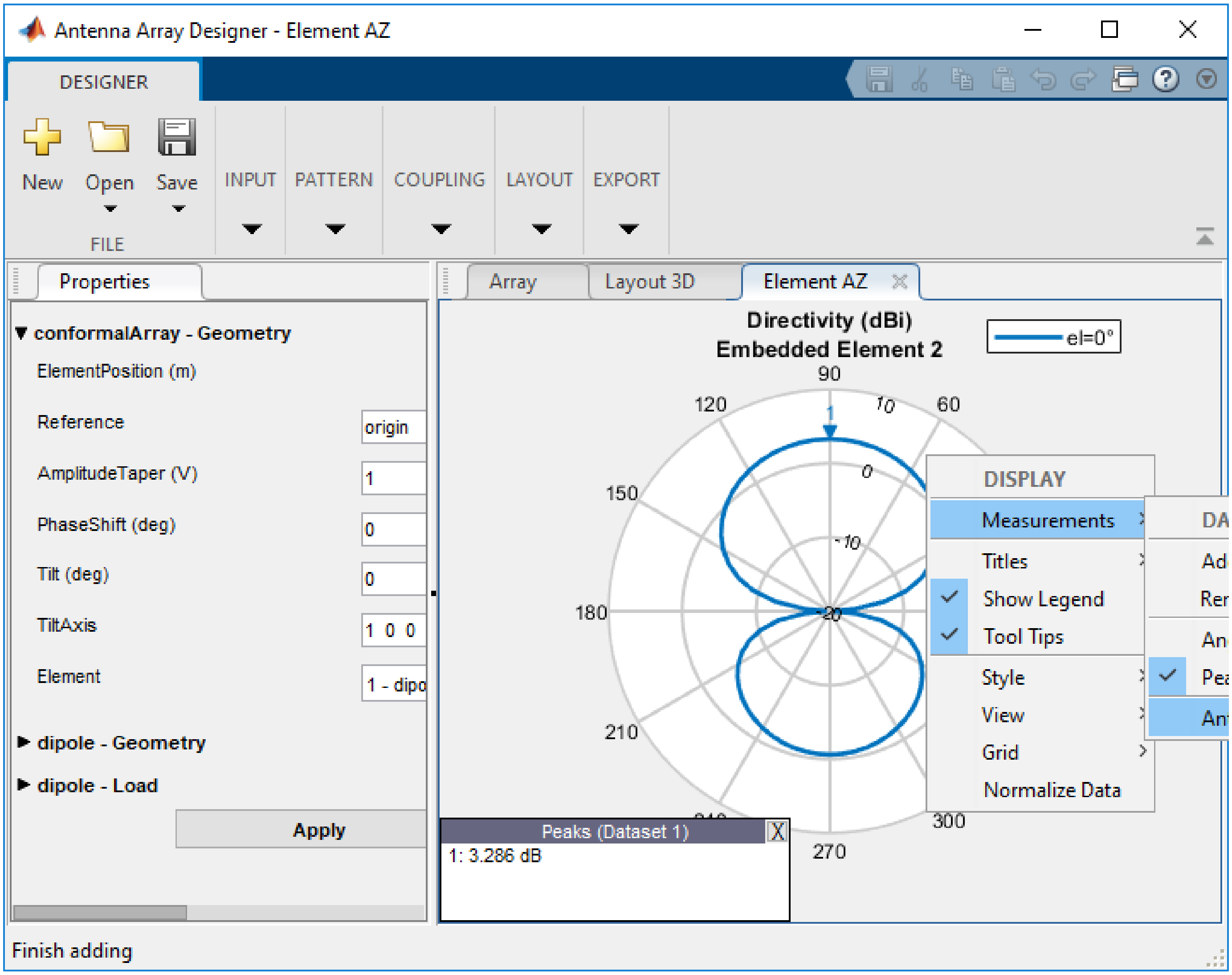
Show the embedded element pattern in the azimuth plane for element 2. Choose **Embedded Element** in the **PATTERN** pane . Click **AZ Pattern**. From the element selection window, click element 2 and then **OK**.



Select an element to visualize its radiation pattern:

Selected Elements: 2

To view the HPBW, right click on the azimuth pattern and select **Measurements > Antenna Metrics**.



Antenna Array Designer - Element AZ

▼ conformalArray - Geometry

ElementPosition (m)

Reference: origin

AmplitudeTaper (V): 1

PhaseShift (deg): 0

Tilt (deg): 0

TiltAxis: 1 0 0

Element: 1 - dipo

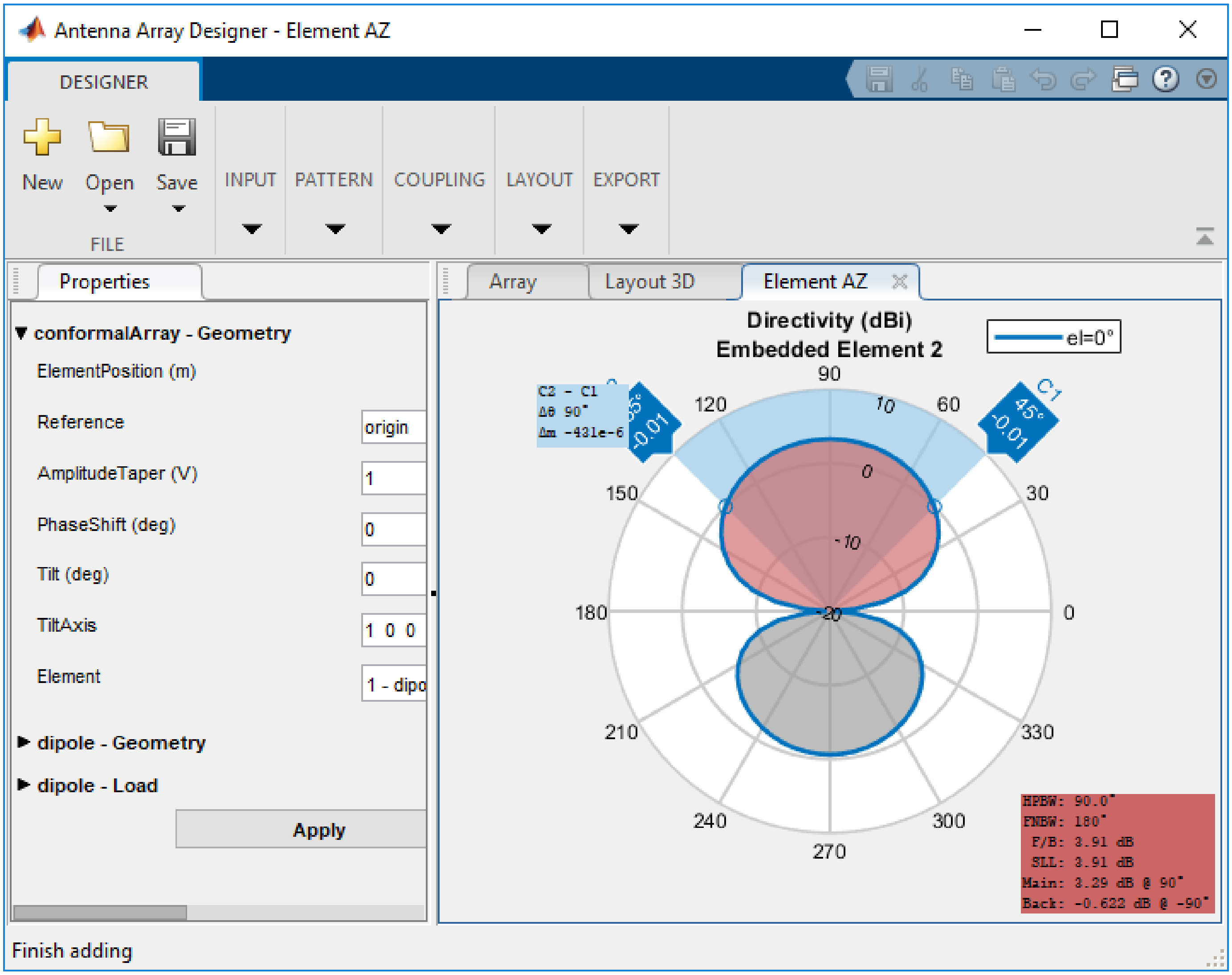
▶ dipole - Geometry

▶ dipole - Load

Peaks (Dataset 1)

1: 3.286 dB

Finish adding



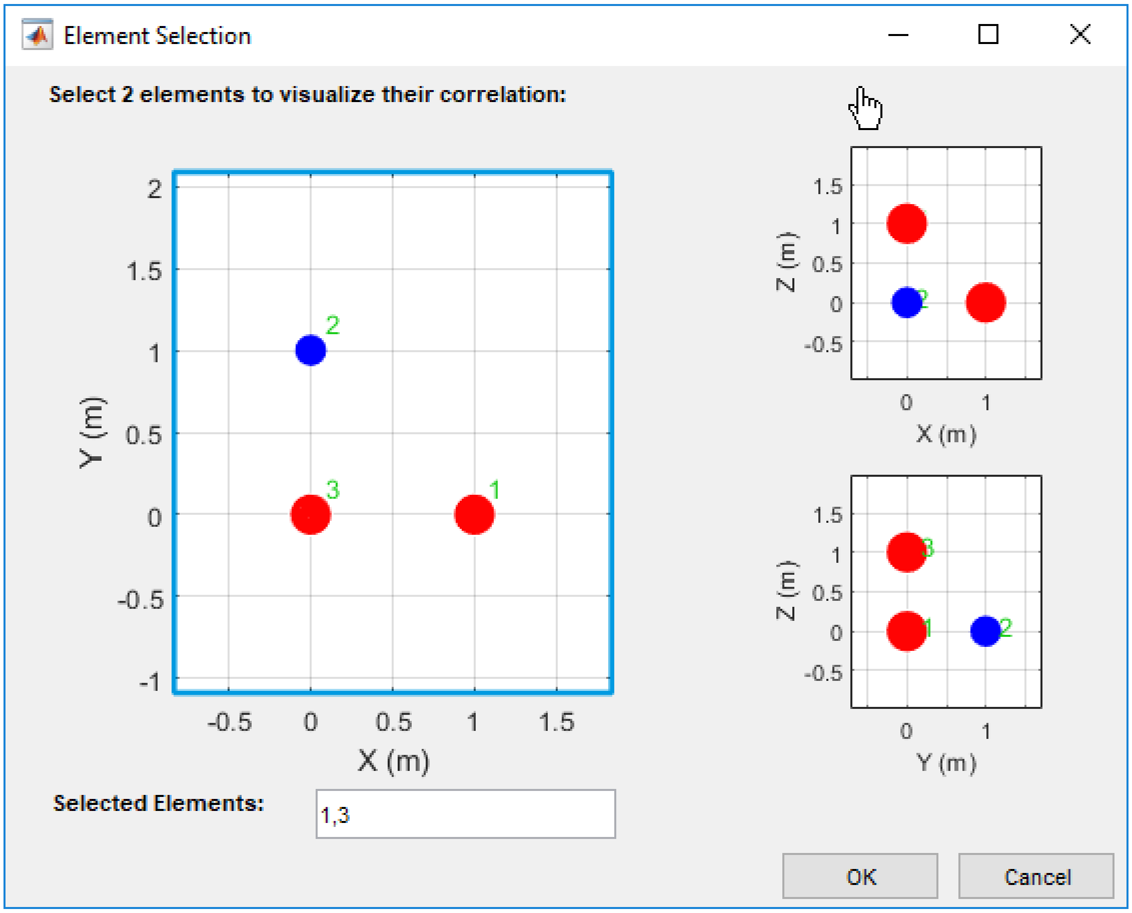
▼ conformalArray - Geometry
ElementPosition (m)
Reference: origin
AmplitudeTaper (V): 1
PhaseShift (deg): 0
Tilt (deg): 0
TiltAxis: 1 0 0
Element: 1 - dipo
▶ dipole - Geometry
▶ dipole - Load

```
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°
```

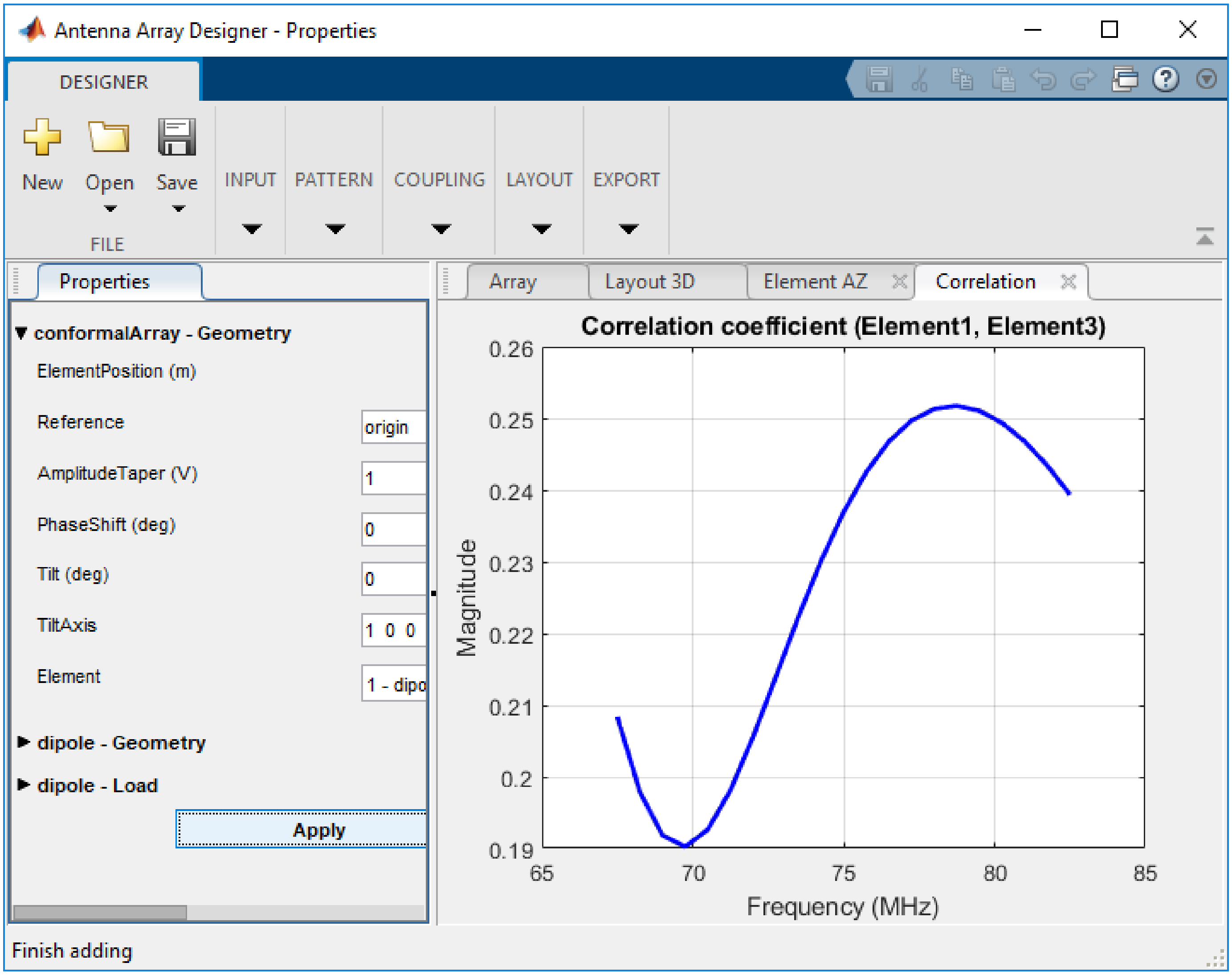
Coupling Between Elements

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.



Select 2 elements to visualize their correlation:

Selected Elements: 1,3



▼ conformalArray - Geometry

ElementPosition (m)

Reference: origin

AmplitudeTaper (V): 1

PhaseShift (deg): 0

Tilt (deg): 0

TiltAxis: 1 0 0

Element: 1 - dipo

▶ dipole - Geometry

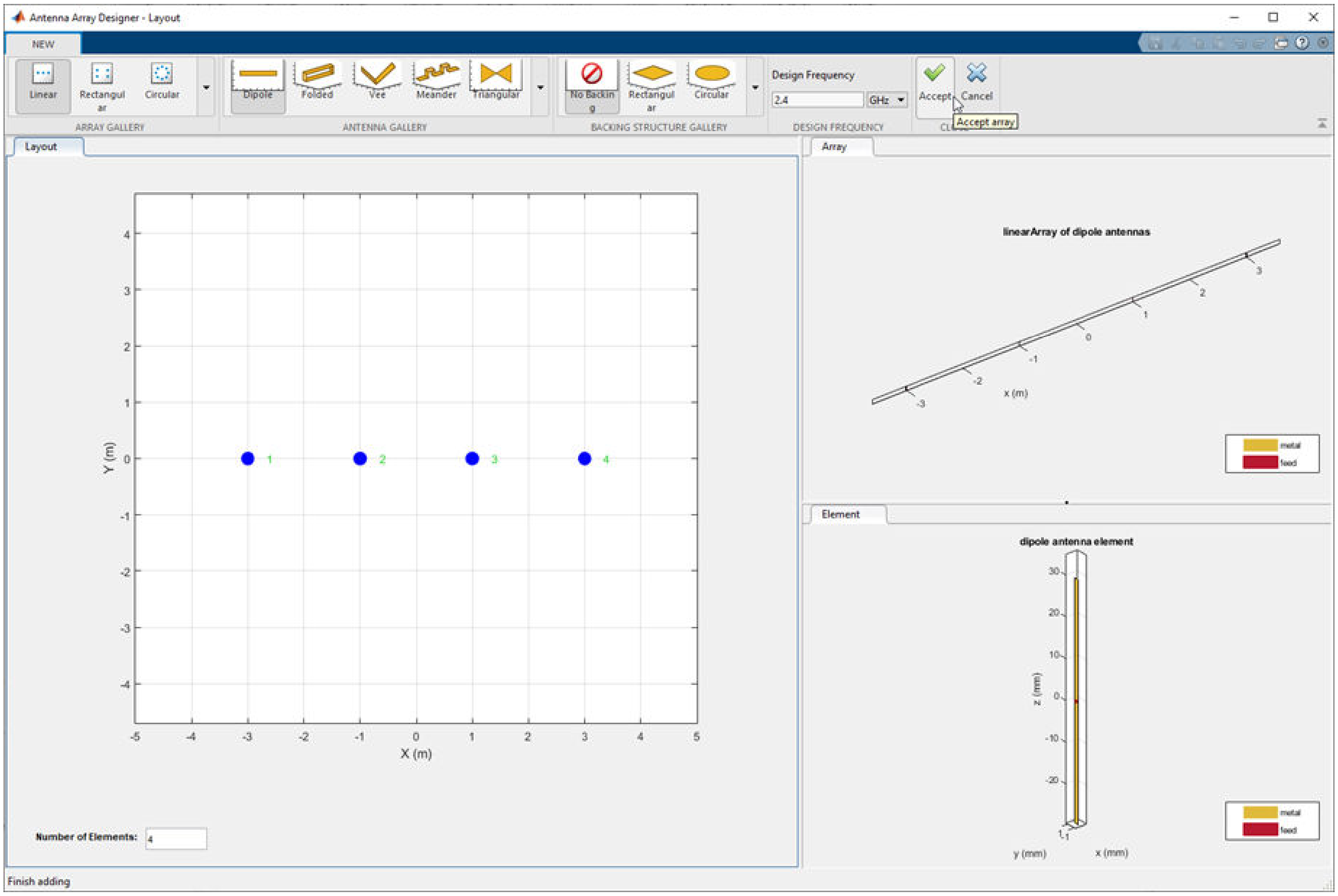
▶ dipole - Load

Finish adding

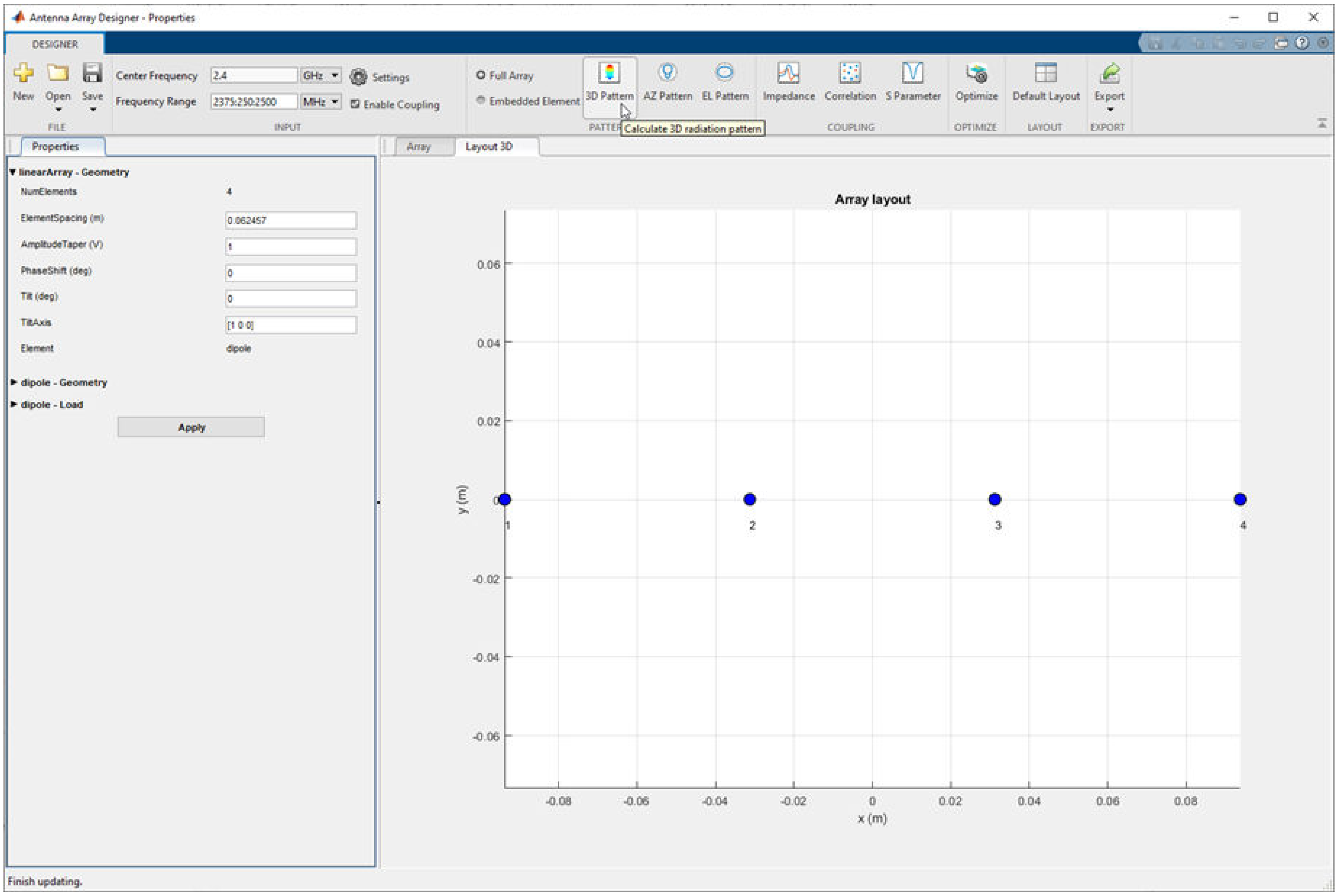
Optimizing Linear Dipole Antenna Array

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.



Select **3D Pattern** under **Pattern** section to calculate the 3-D radiation pattern.



Center Frequency 2.4 GHz

Frequency Range 2375:250:2500 MHz

☑ Enable Coupling

▼ linearArray - Geometry

NumElements 4

ElementSpacing (m) 0.062457

AmplitudeTaper (V) 1

PhaseShift (deg) 0

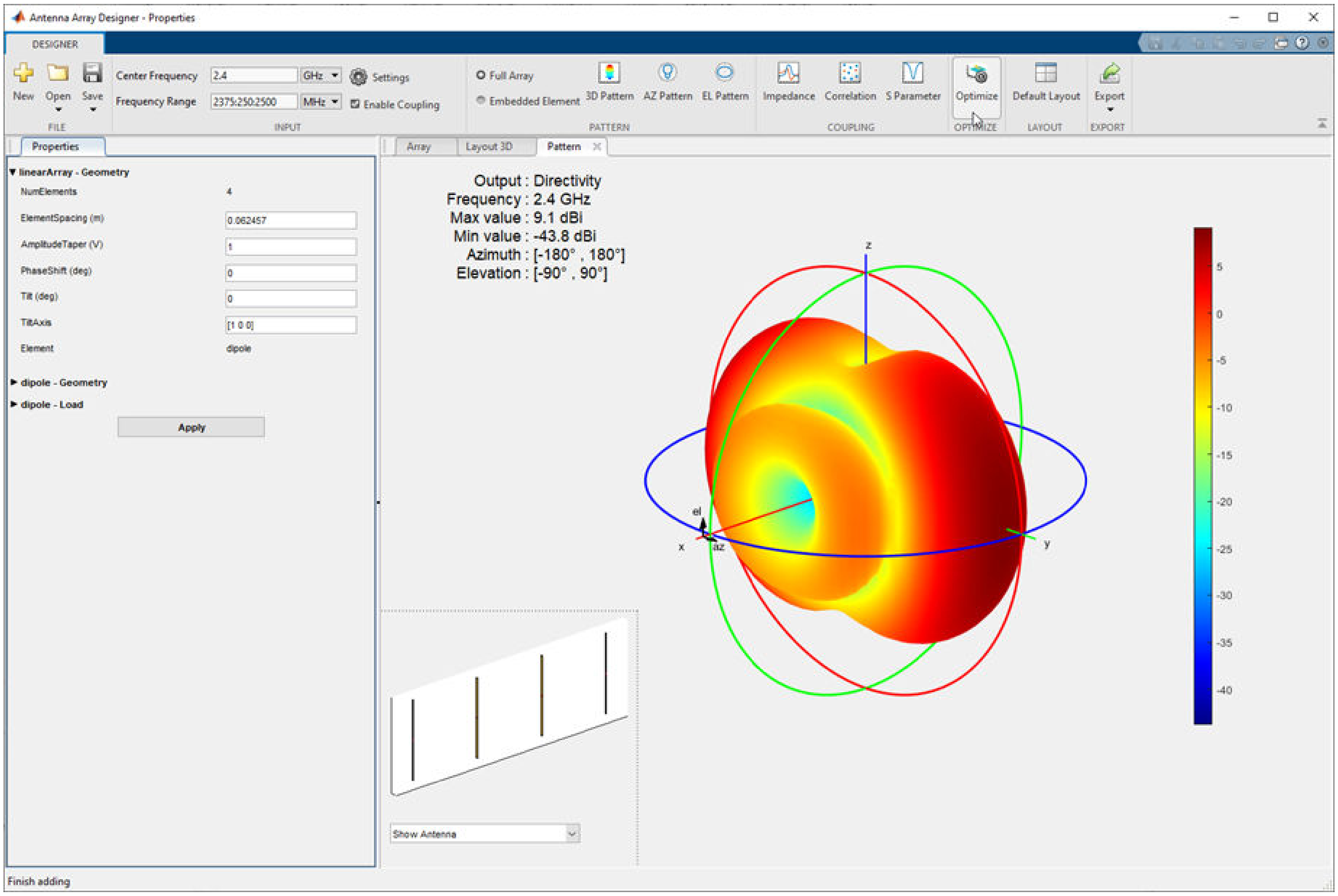
Tilt (deg) 0

TiltAxis [1 0 0]

Element dipole

Finish updating.

The gain is 9.1 dBi. Click **Optimize** on the app toolstrip to optimize this array.



Antenna Array Designer - Properties

Center Frequency 2.4 GHz

Frequency Range 2375:250:2500 MHz

☑ Enable Coupling

○ Full Array

● Embedded Element

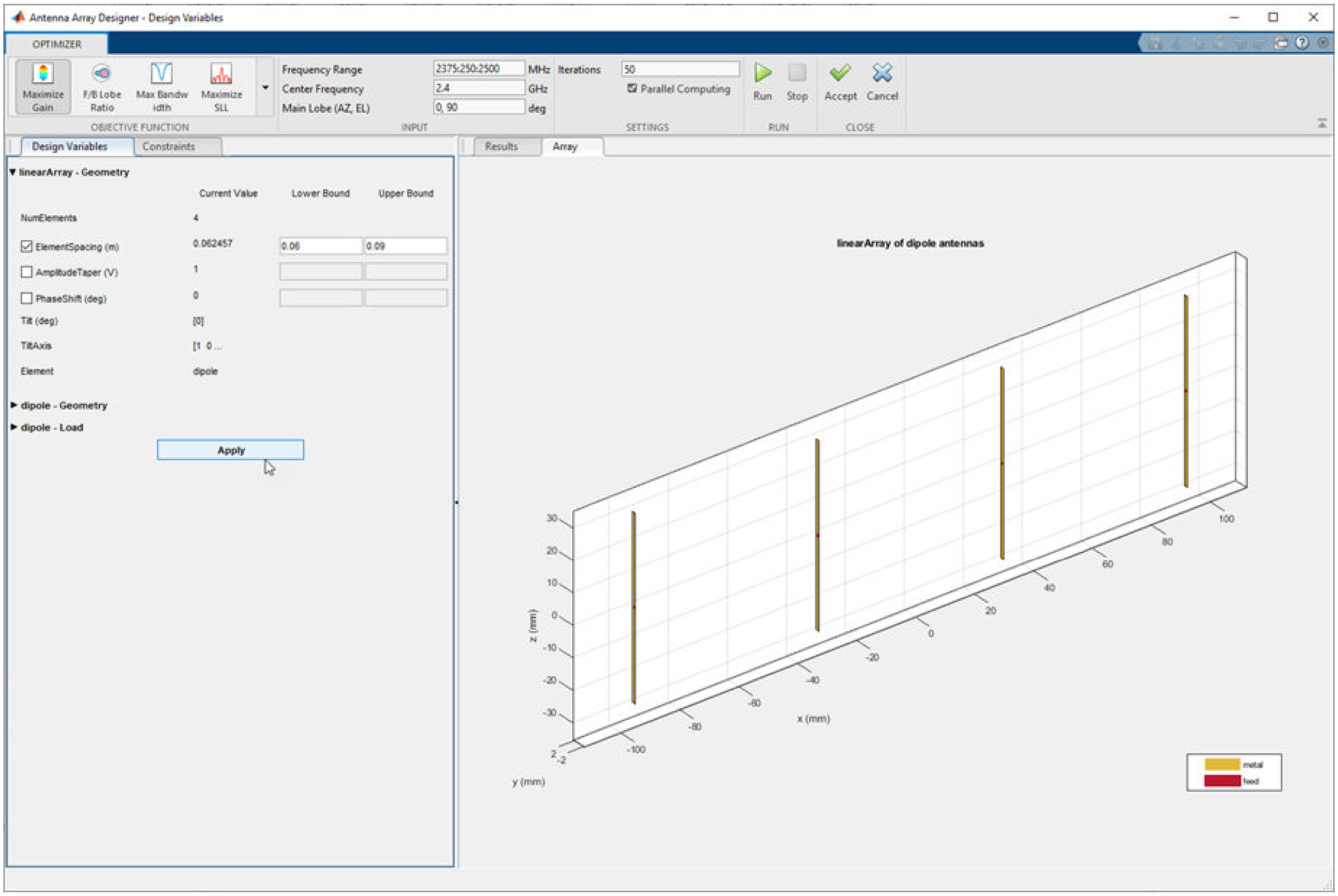
linearArray - Geometry

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°]

Finish adding

On the Optimizer tab, click **Maximize Gain** in the **Objective Function** section. In the **Design Variables pane**, select the variables you want to optimize. For the purpose of this example, select the **Element Spacing** variable, set the lower and upper bounds as 0.06 and 0.09.



Frequency Range 2375:250:2500 MHz
Center Frequency 2.4 GHz
Main Lobe (AZ, EL) 0, 90 deg
Iterations 50
☑ Parallel Computing

▼ linearArray - Geometry

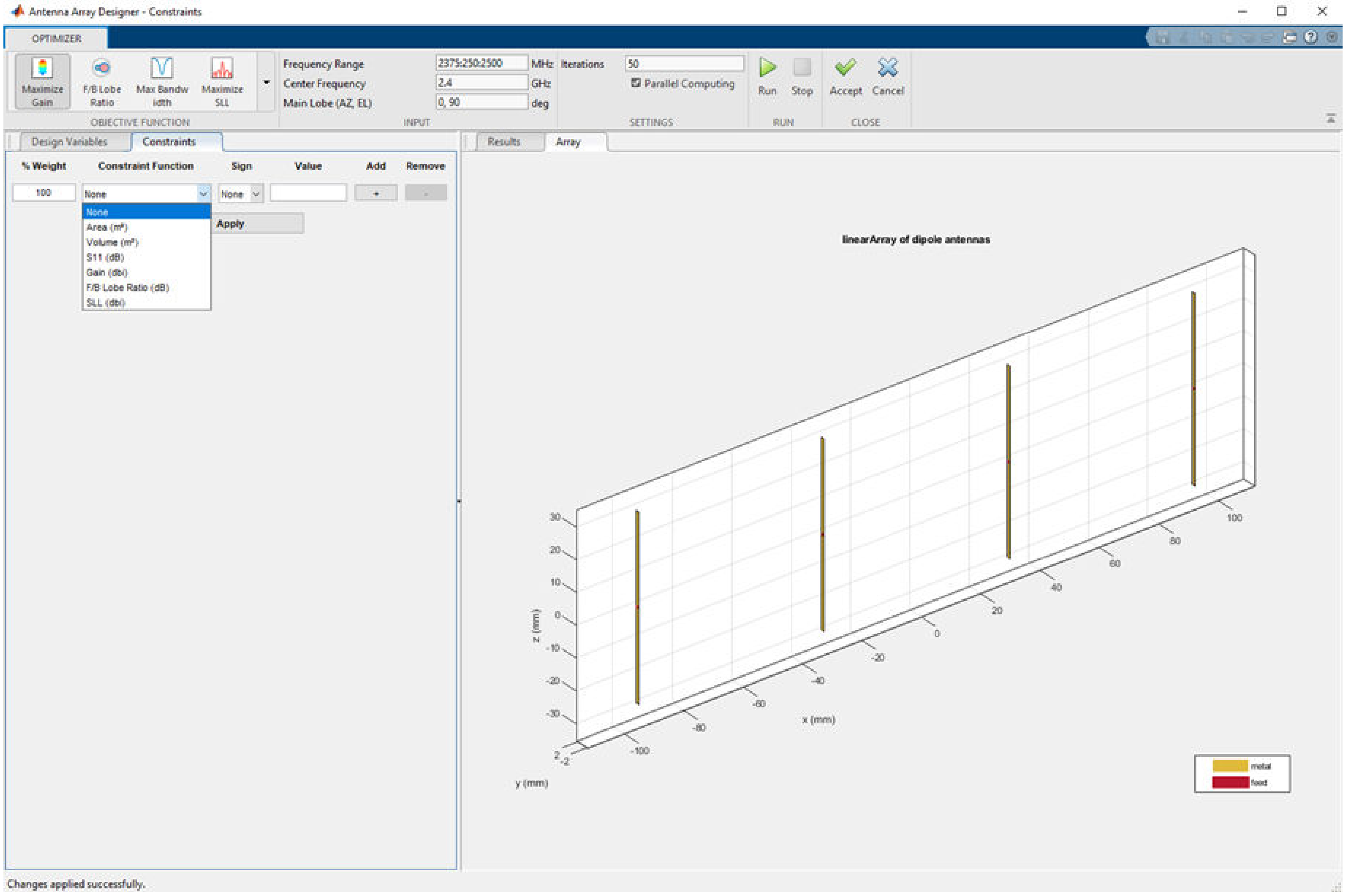
	Current Value	Lower Bound	Upper Bound
NumElements	4		
☑ ElementSpacing (m)	0.062457	0.06	0.09
□ AmplitudeTaper (V)	1		
□ PhaseShift (deg)	0		
Tilt (deg)	[0]		
TiltAxis	[1 0 ...		
Element	dipole		

▶ dipole - Geometry

▶ dipole - Load

Click the **Constraints** pane. In this example, **Constraint Functions** are not selected. If your application requires constraints, chosen one or more constraint functions from the dropdown. You can use the **Add** and **Remove** buttons to add or remove constraints.

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**.

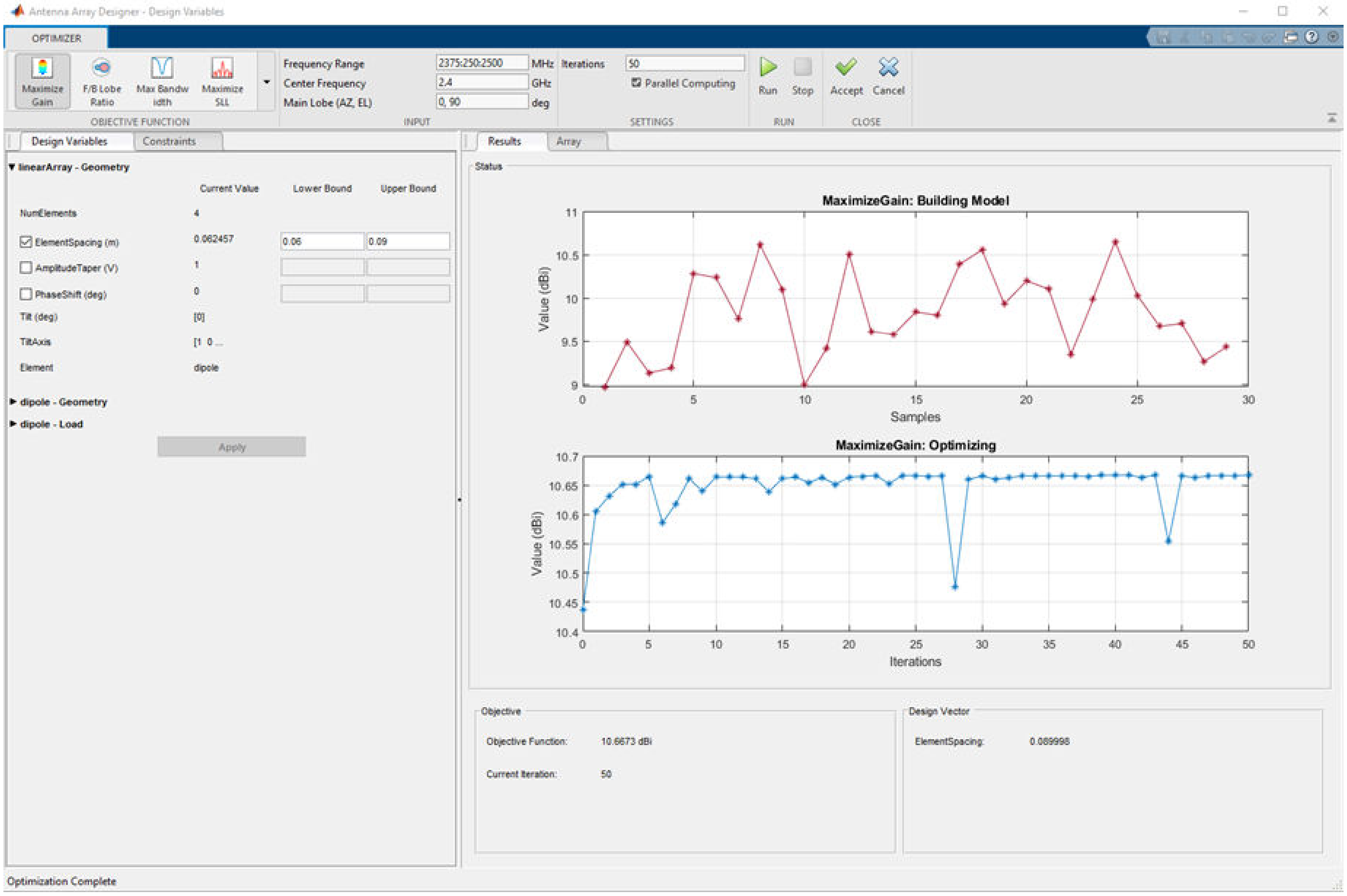


Frequency Range 2375:250:2500 MHz
Center Frequency 2.4 GHz
Main Lobe (AZ, EL) 0, 90 deg
Iterations 50
☑ Parallel Computing

% Weight 100 | Constraint Function None | Sign None

Changes applied successfully.

Once the simulation is complete, the optimization results are displayed in the **Results** pane



Frequency Range 2375:250:2500 MHz
Center Frequency 2.4 GHz
Main Lobe (AZ, EL) 0, 90 deg
Iterations 50
☑ 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		

▶ dipole - Geometry

▶ dipole - Load

Objective

Objective Function: 10.6673 dBi

Current Iteration: 50

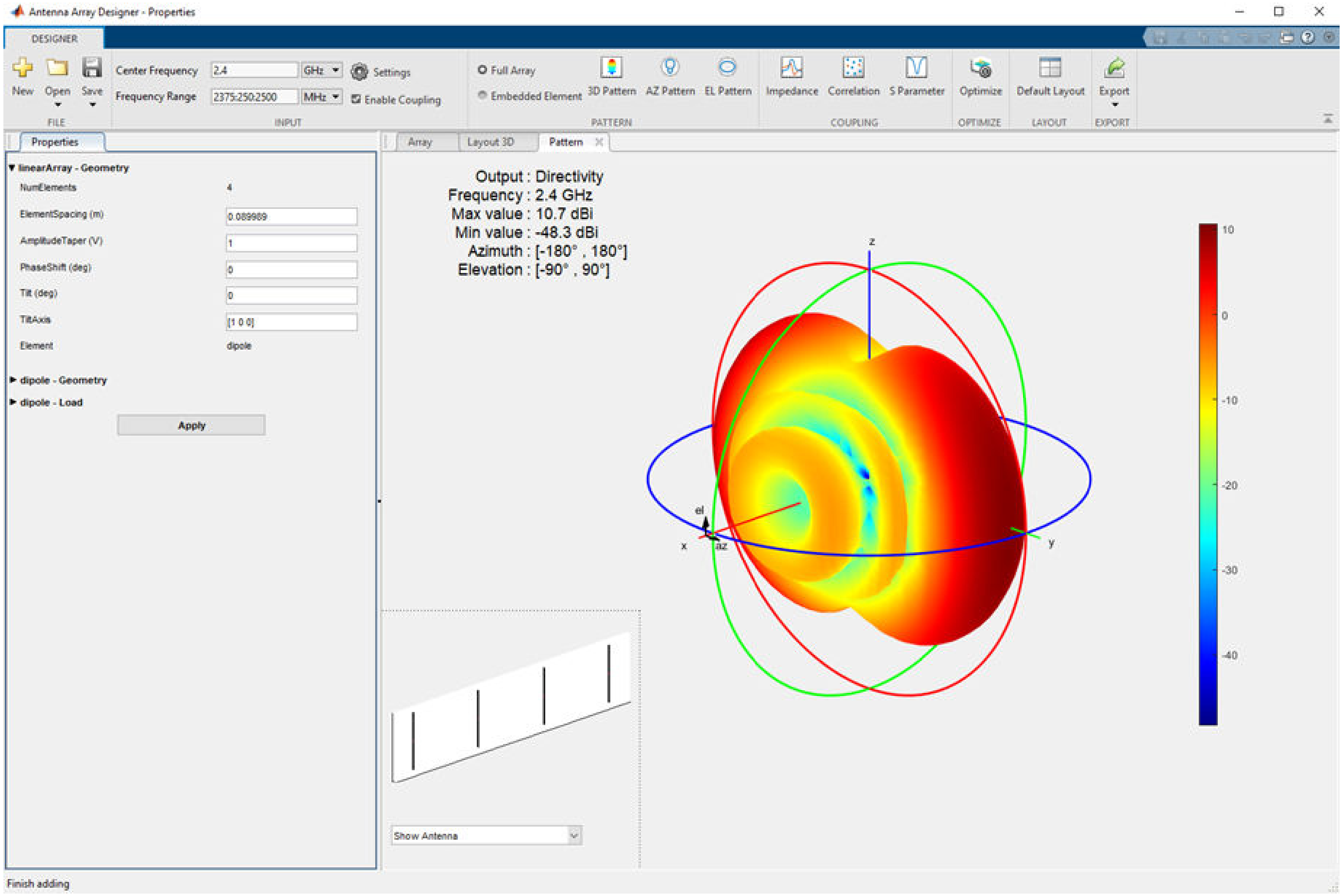
Design Vector

ElementSpacing: 0.089998

Optimization Complete

.

Click **Accept**. In the **Pattern** section, select **3D Pattern** as the pattern to display the optimized array design. The optimized gain has now increased to 10.7 dBi.



Center Frequency 2.4 GHz

Frequency Range 2375:250:2500 MHz

☑ Enable Coupling

○ Full Array

● Embedded Element

linearArray - Geometry

NumElements 4

ElementSpacing (m) 0.089989

AmplitudeTaper (V) 1

PhaseShift (deg) 0

Tilt (deg) 0

TiltAxis [1 0 0]

Element dipole

Output : Directivity
Frequency : 2.4 GHz
Max value : 10.7 dBi
Min value : -48.3 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Finish adding

- “Design and Analysis Using Antenna Array Designer App”
- “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.

See Also

Topics

“Design and Analysis Using Antenna Array Designer App”
“Optimization of Antenna Array Elements Using Antenna Array Designer App”
“Antenna Optimization Algorithm”

Introduced in R2019b

4

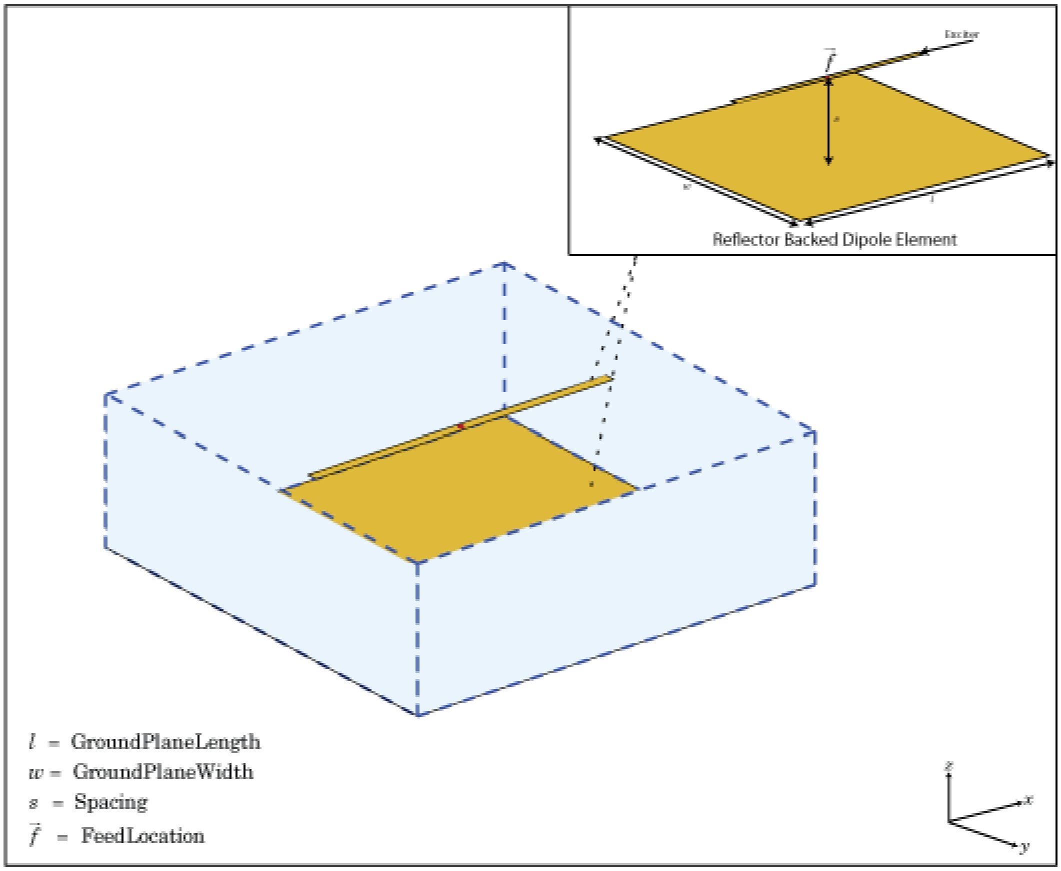
Array Objects

infiniteArray

Create 2-D custom mesh antenna on 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 antennas specifies the boundaries of the unit cell. Antennas without a ground plane require a reflector. By default, the infinite array has reflector-backed dipoles as antenna elements. The default dimensions are chosen for an operating frequency of 1 GHz.



Creation

Description

`infa = infiniteArray` creates an infinite antenna array in the X-Y plane.

`infa = infiniteArray(Name,Value)` creates an infinite 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

`Element` — Type of individual antenna elements in unit cell

reflector-backed dipole (default) | object

Type of individual antenna elements in unit cell, specified as an object. Antenna without a groundplane is backed using a reflector. The ground plane size specifies the unit cell boundaries.

Example: `'Element'`,`reflector`

`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`

Object Functions

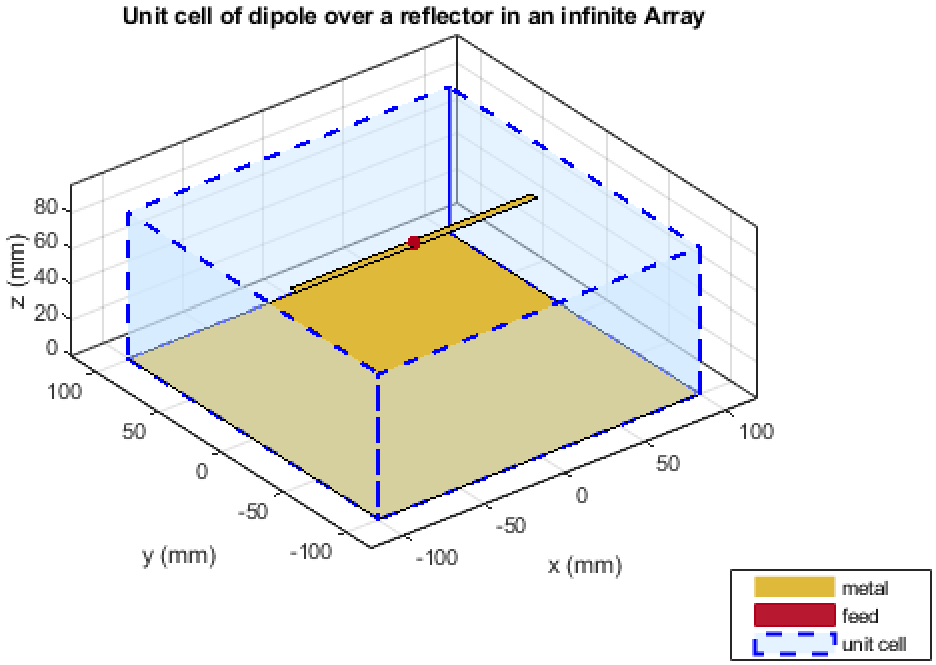
numSummationTerms	Change number of summation terms for calculating periodic Green's function

Examples

Infinite Array of Reflector-Backed Dipoles

Create an infinite array with reflector-backed dipoles as unit cells. Scan the array at boresight. Visualize the unit cell.

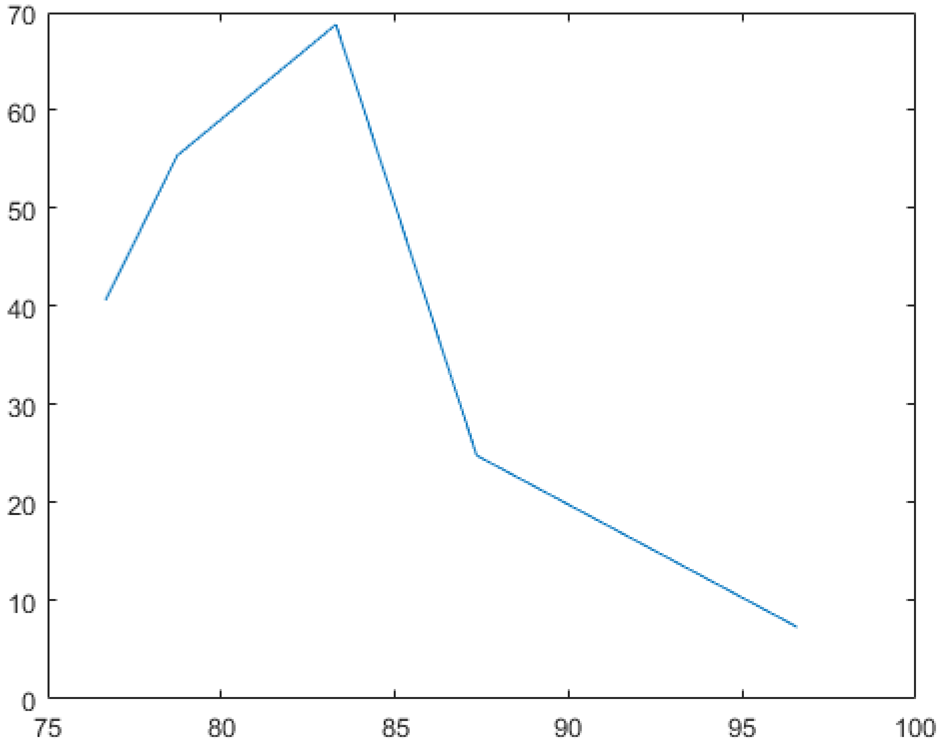
```
infa = infiniteArray('Element',reflector,'ScanAzimuth',0, ...
    'ScanElevation',90);
show(infa)
```



Scan Impedance of Infinite Array

Calculate the scan impedance of an infinite array at 1GHz. 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)
```



References

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

See Also

`circularArray` | `conformalArray` | `linearArray` | `rectangularArray`

Topics

"Rotate Antennas and Arrays"

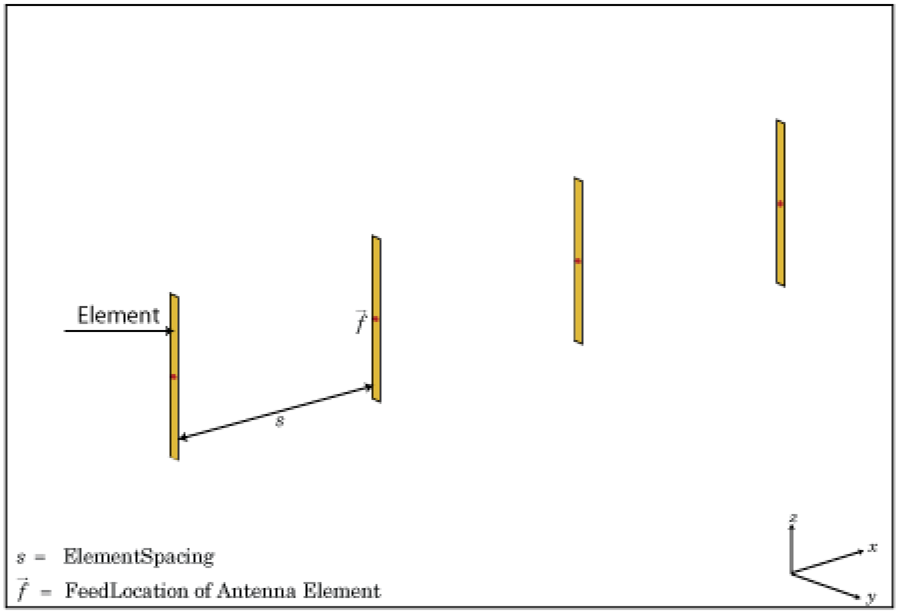
Introduced in R2015b

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 linear arrays
dipole (default) | antenna object | array object

Individual antenna elements or linear arrays, 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: `array.TiltAxis = 'Z'`

Object Functions

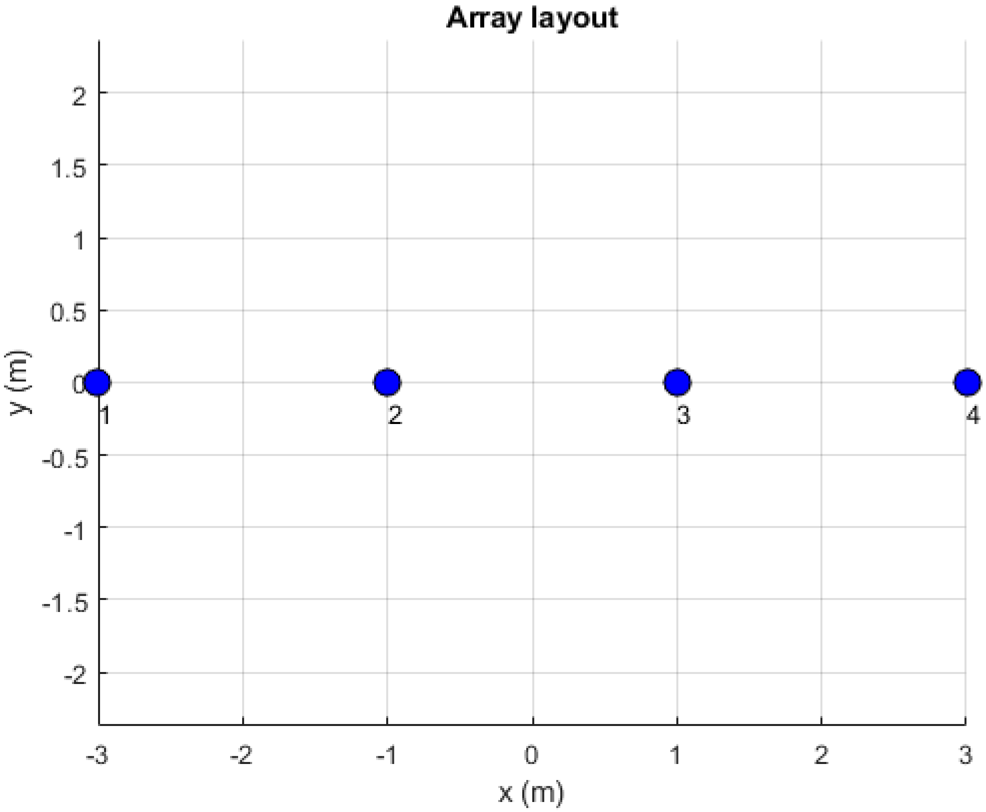
show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on metal or dielectric 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 or 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	S-parameter object

Examples

Create and Plot Layout of Linear Array

Create a linear array of four dipoles and plot the layout of the array.

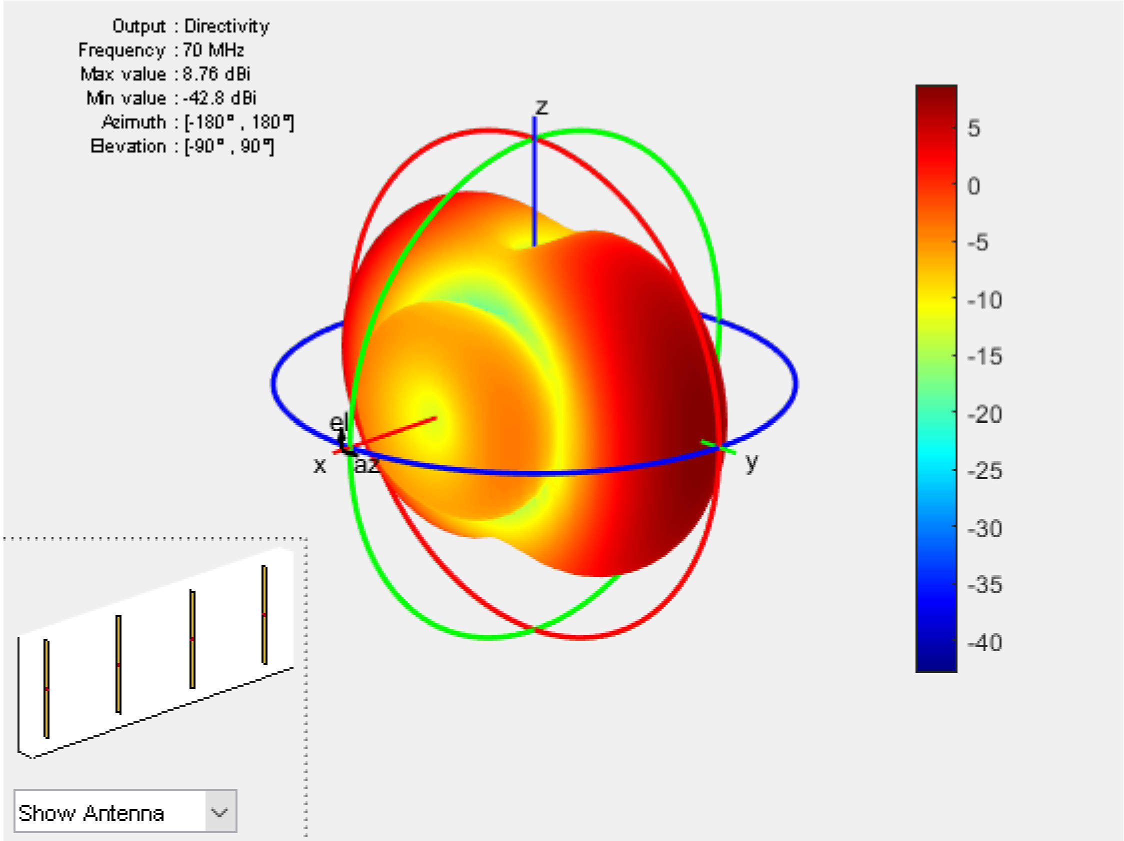
```
la = linearArray;
la.NumElements = 4;
layout(la);
```



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);
```



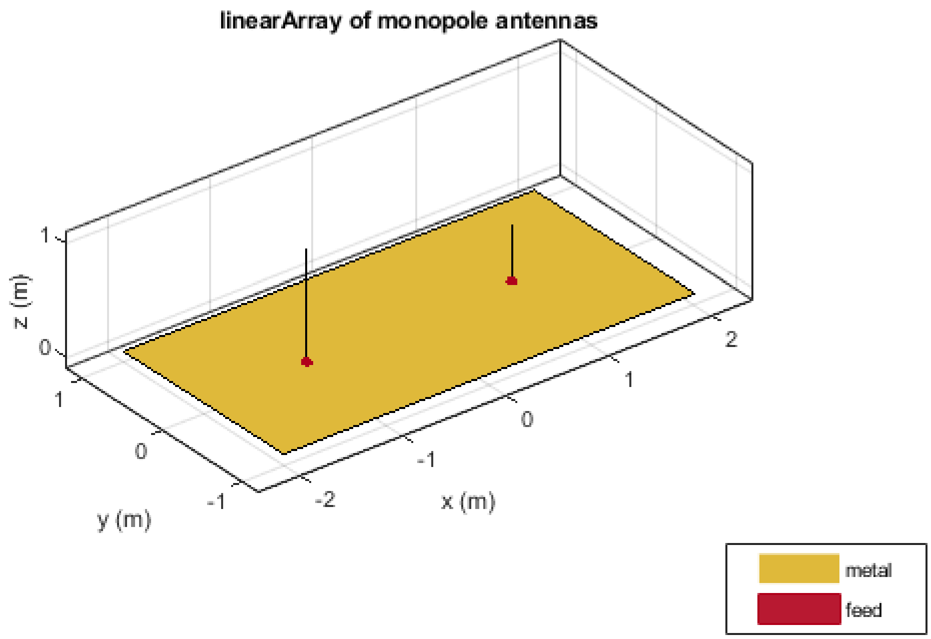
Linear Array Using Groundplane Antennas

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]
```

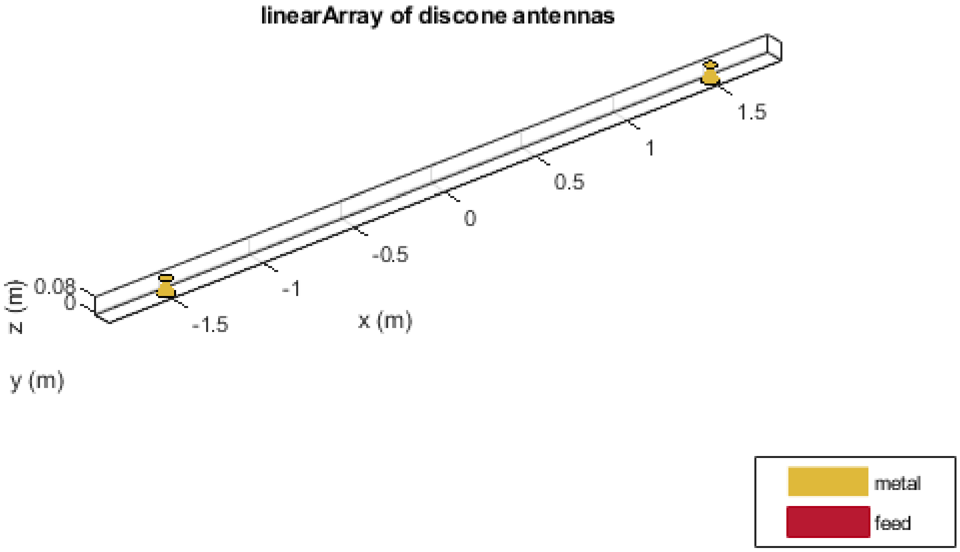
```
mla.Element = [m1,m2];
show(mla);
```



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)
```

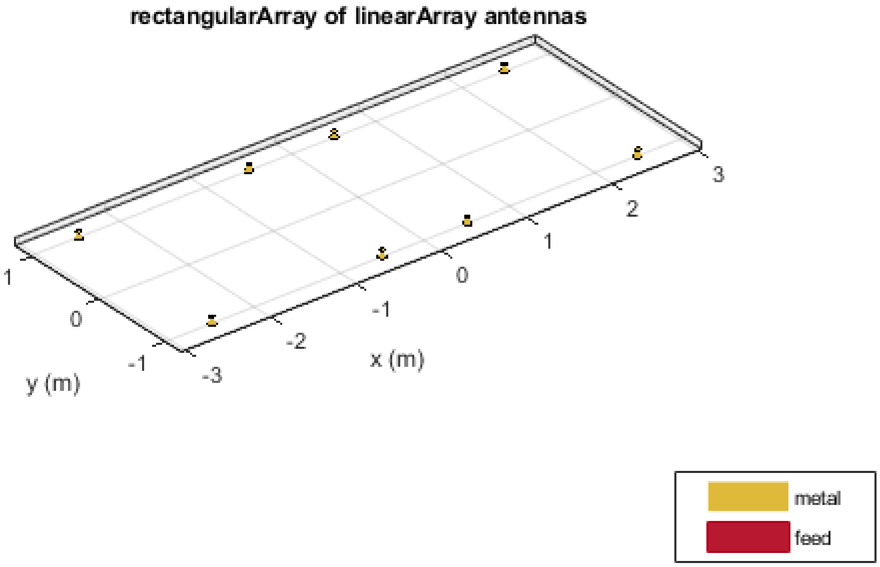


Create a rectangular of the linear array.

```
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]
```

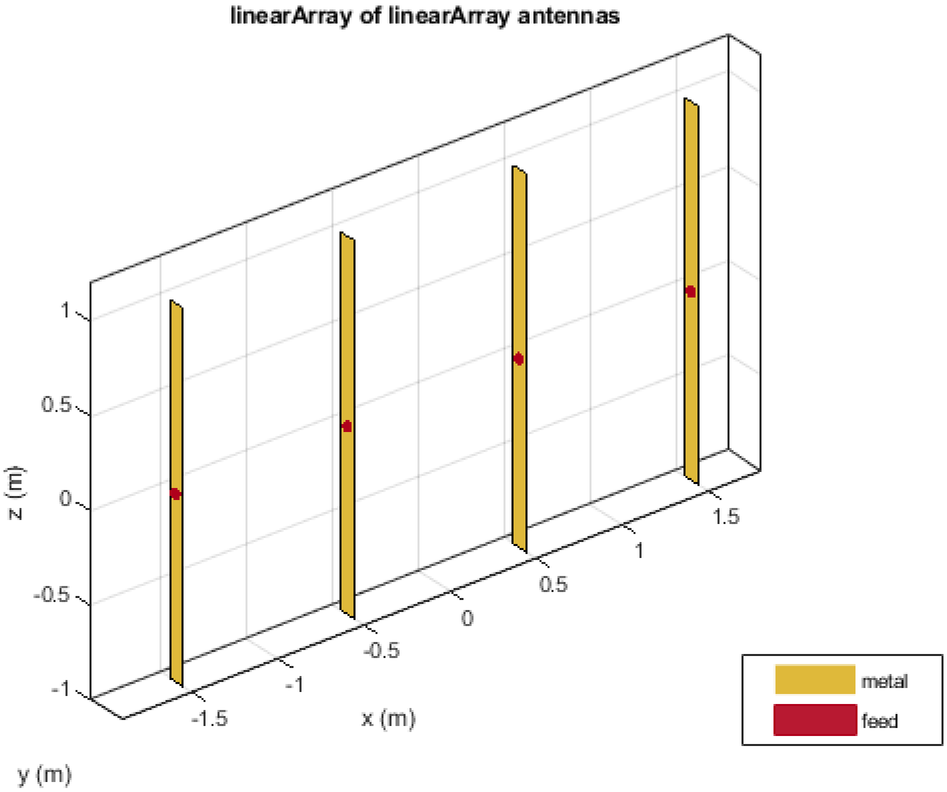
```
show(ra)
```



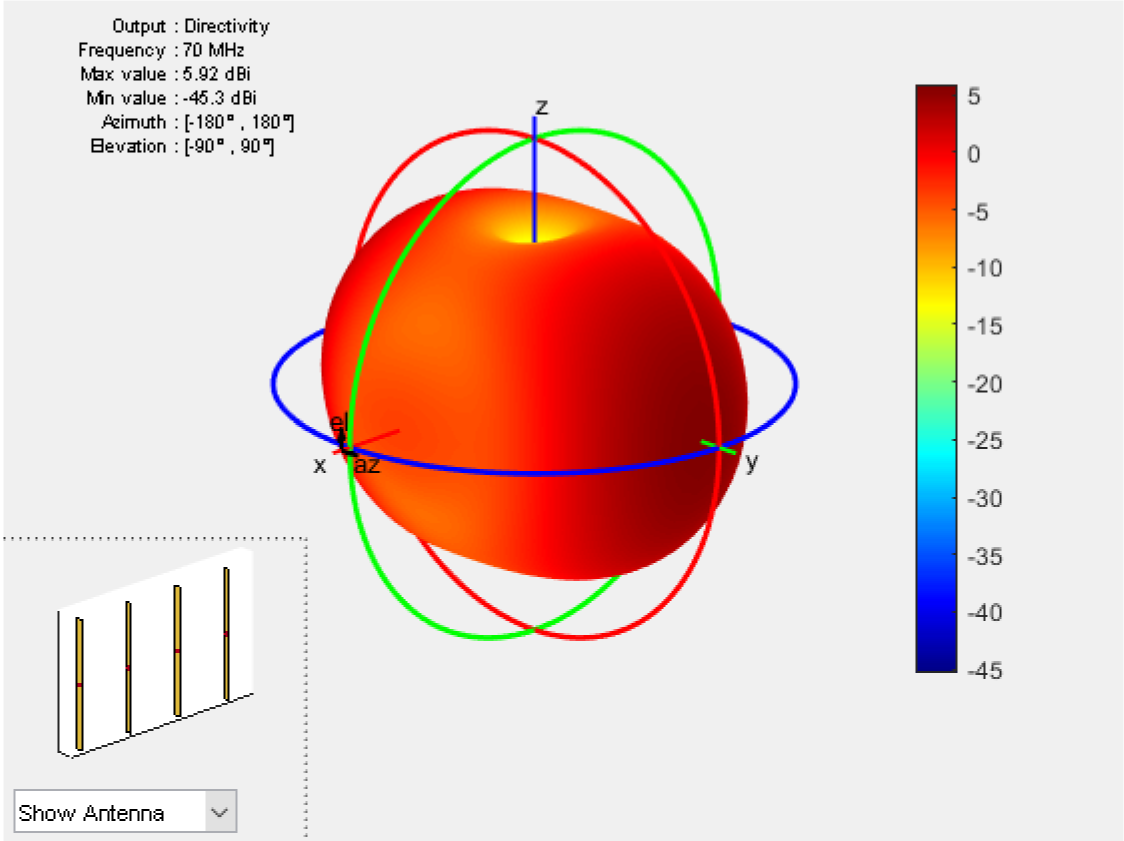
Pattern of Linear Array of Linear Array

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 : -45.3 dBi
  Azimuth : [-180° , 180°]
 Elevation : [-90° , 90°]
```

References

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

See Also

`circularArray` | `conformalArray` | `infiniteArray` | `rectangularArray`

Topics

"Rotate Antennas and Arrays"

Introduced in R2015a

conformalArray

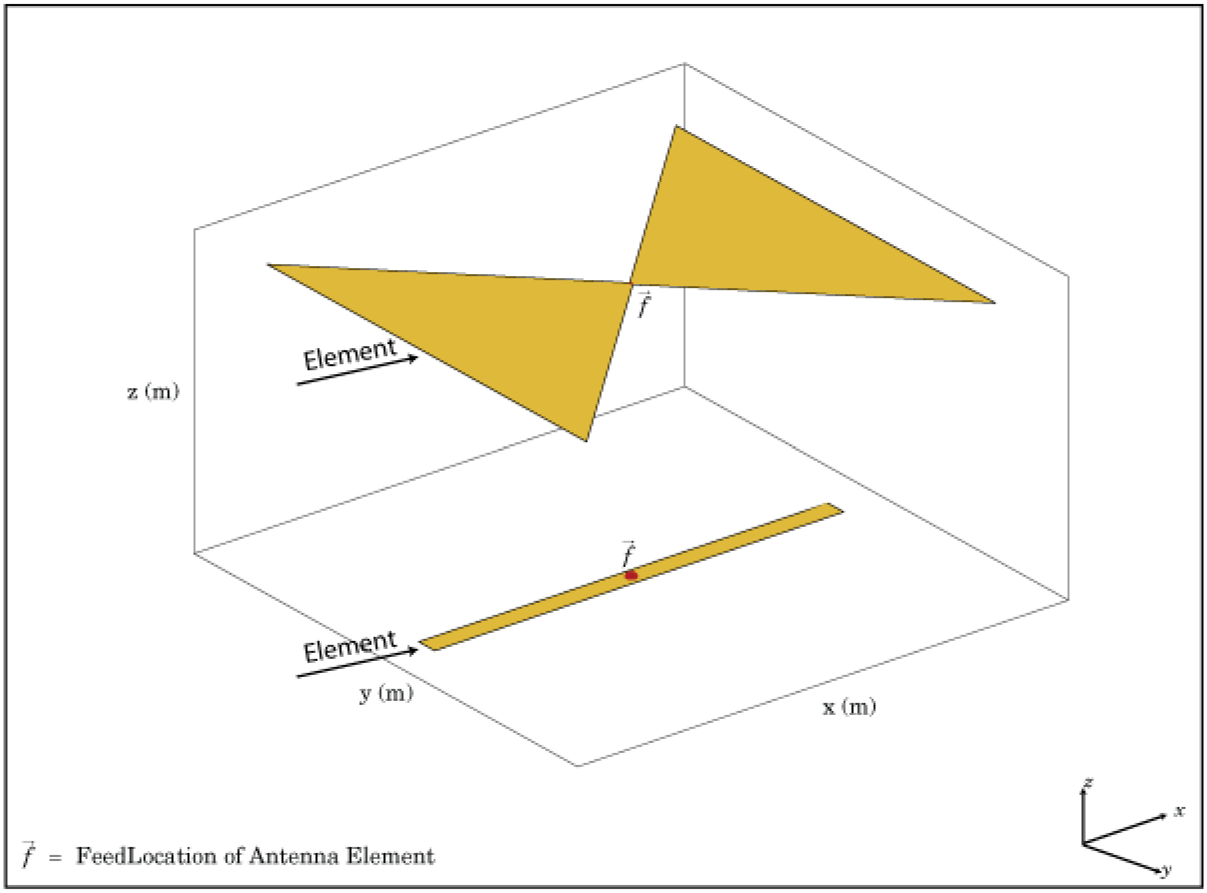
Create conformal antenna array

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.

Output Arguments

`array` — Conformal array
`conformalArray` object

Conformal array, returned as an `conformalArray` object.

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 4-34.

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: `array.TiltAxis = 'Z'`

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on metal or dielectric 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 or 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	S-parameter object

Examples

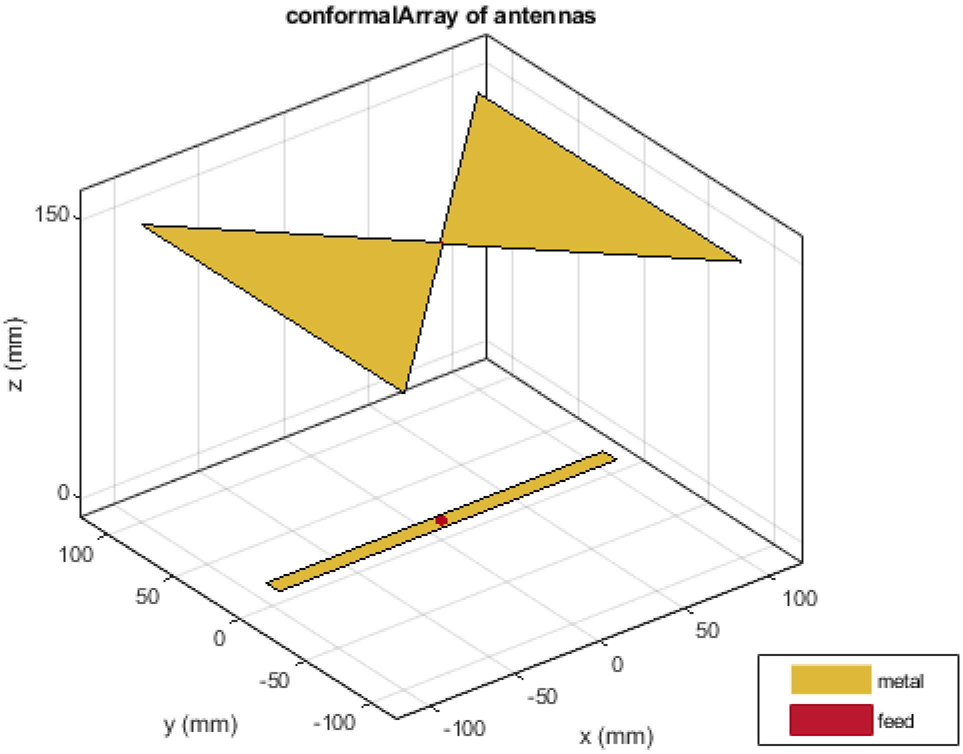
Default Conformal Array

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.

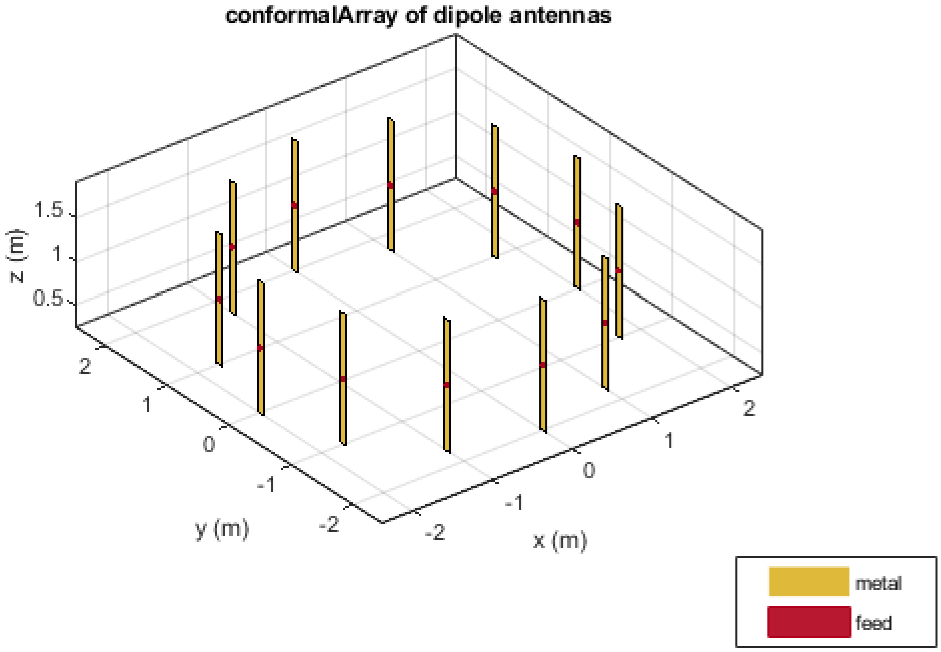
```
elem = repmat(dipole('Length',1.5),1,N);
```

Define the x,y,z values for the element positions in the array.

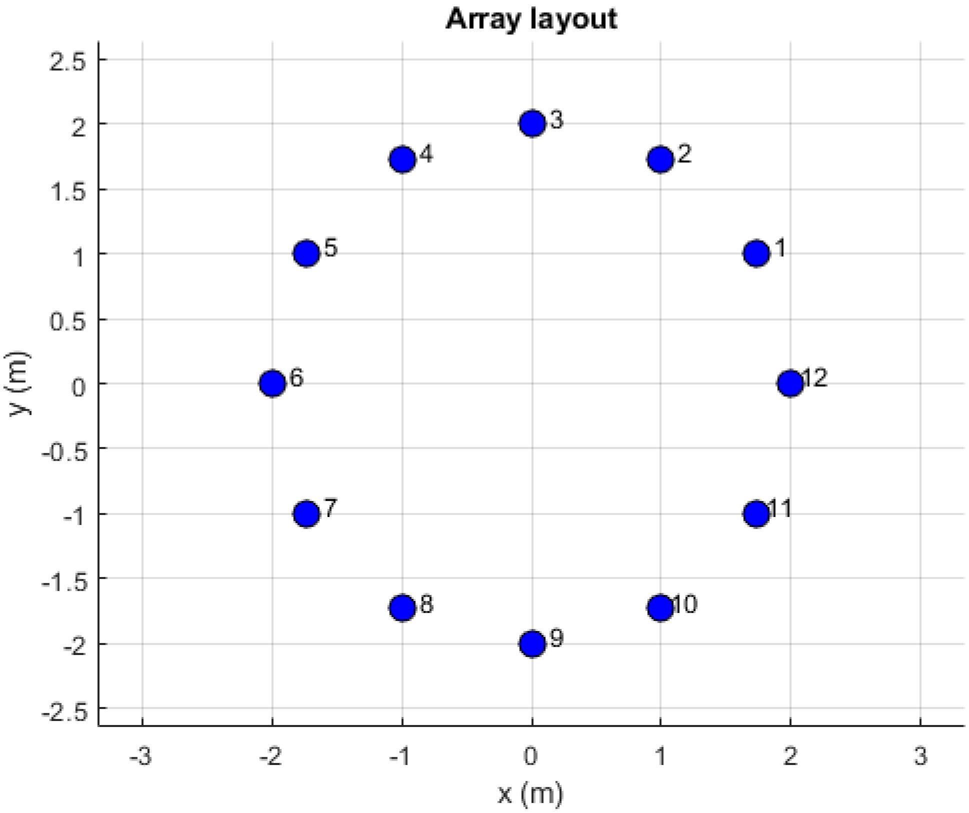
```
del_th = 360/N;
th = del_th:del_th:360;
x = r.*cosd(th);
y = r.*sind(th);
z = ones(1,N);
pos = [x;y;z];
```

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)
```

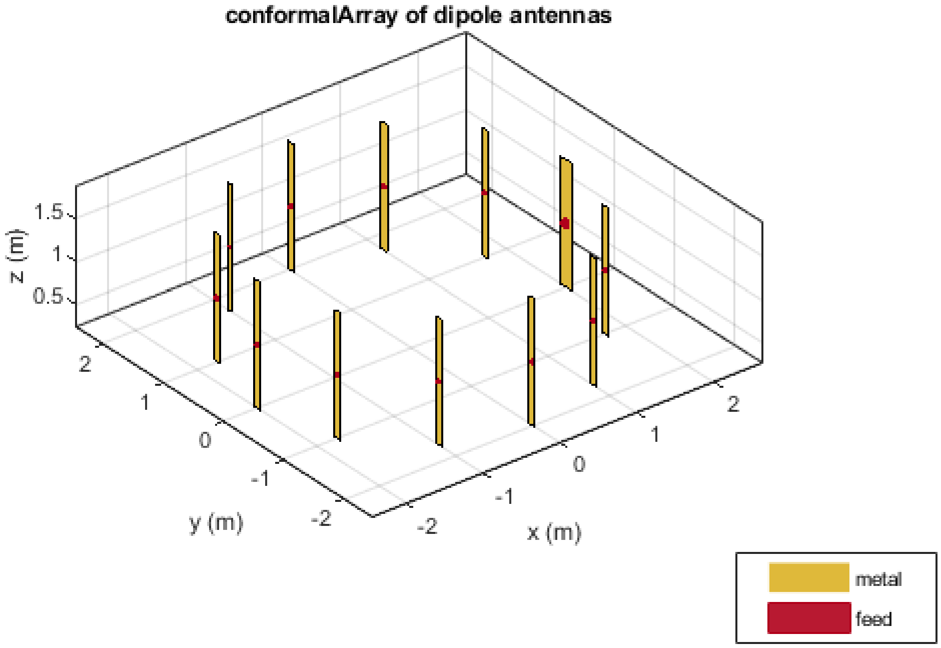


```
figure
layout(c)
```



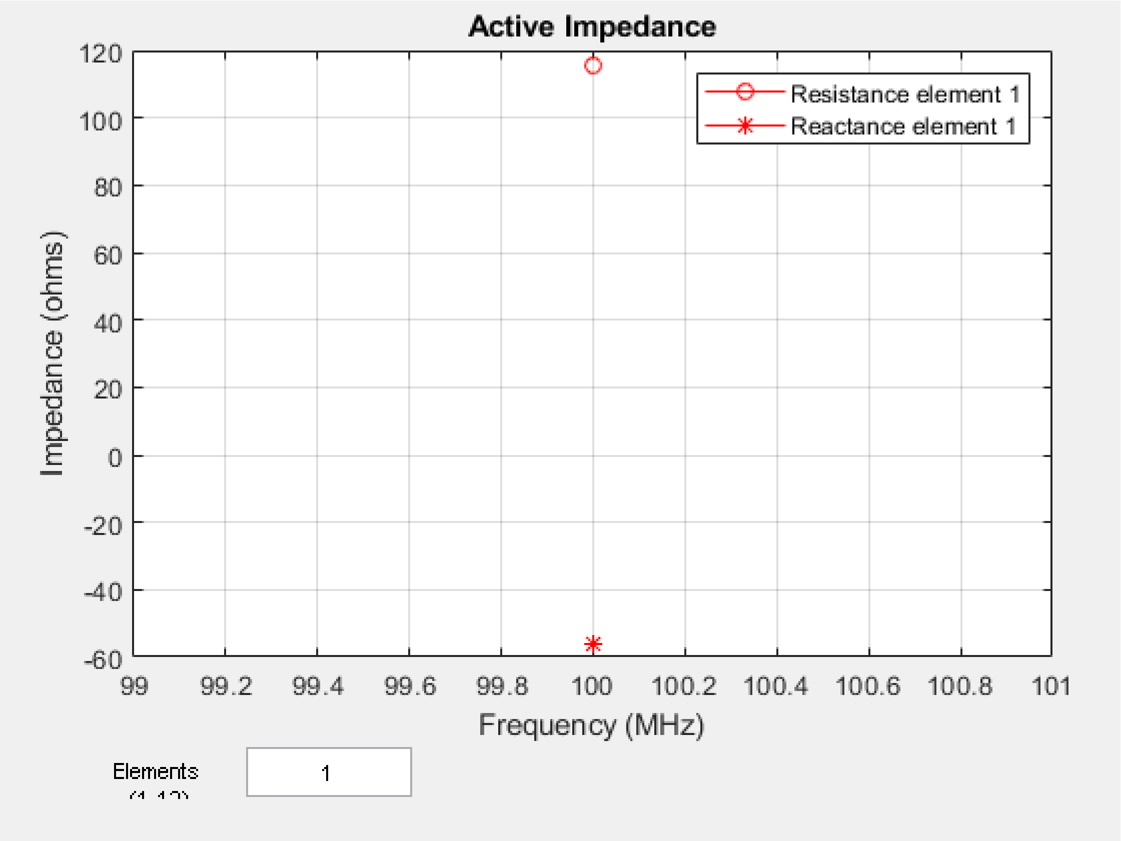
Change the width of the fourth and the twelfth element of the circular array. Visualize the new arrangement.

```
c.Element(4).Width = 0.05;
c.Element(12).Width = 0.2;
figure
show(c)
```



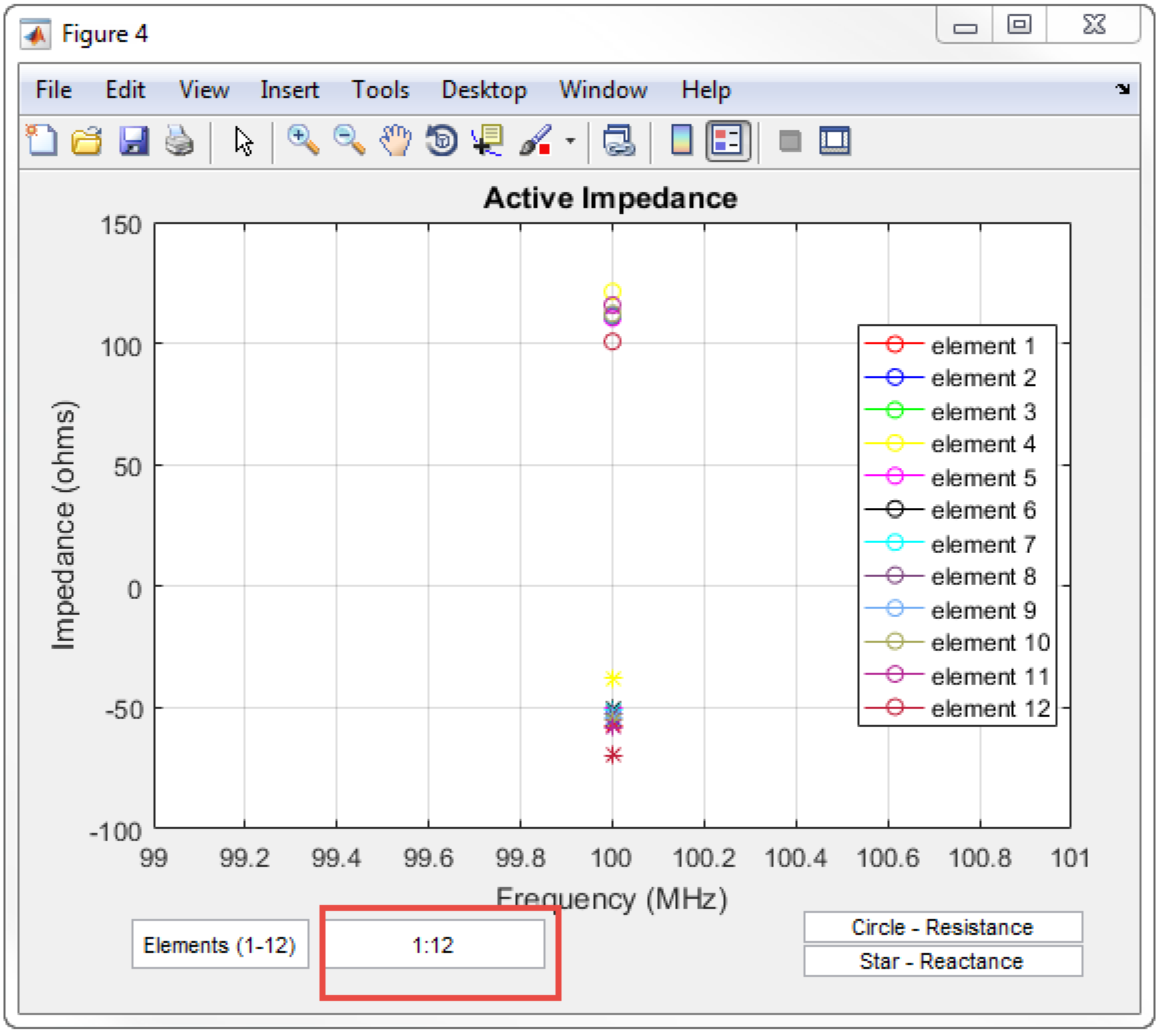
Calculate and plot the impedance of the circular array at 100 MHz. The plot shows the impedance of the first element in the array.

```
figure
impedance(c,100e6)
```



Elements 1

To view the impedance of all the elements in the array change the value from **1** to **1:12** as shown in the figure.



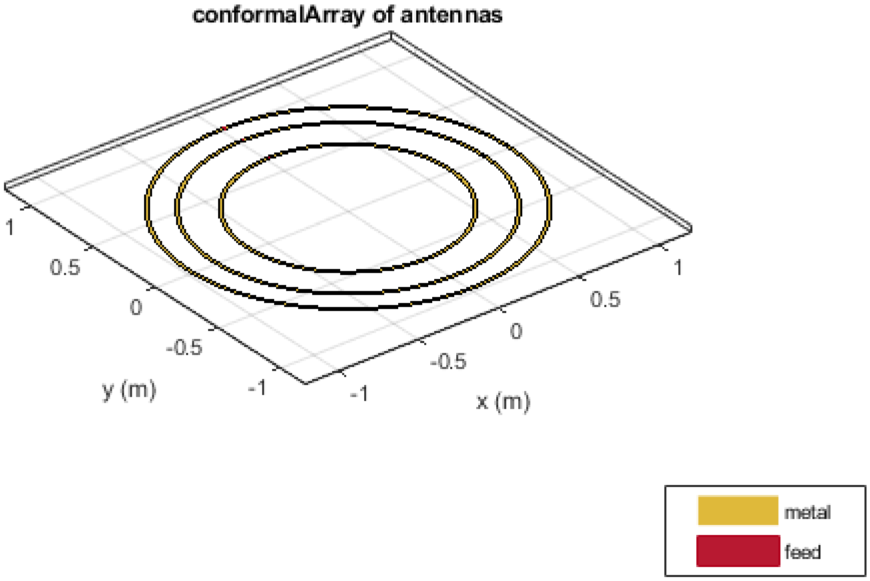
Radiation Pattern of Concentric Array of Circular Loop Antennas

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);
```

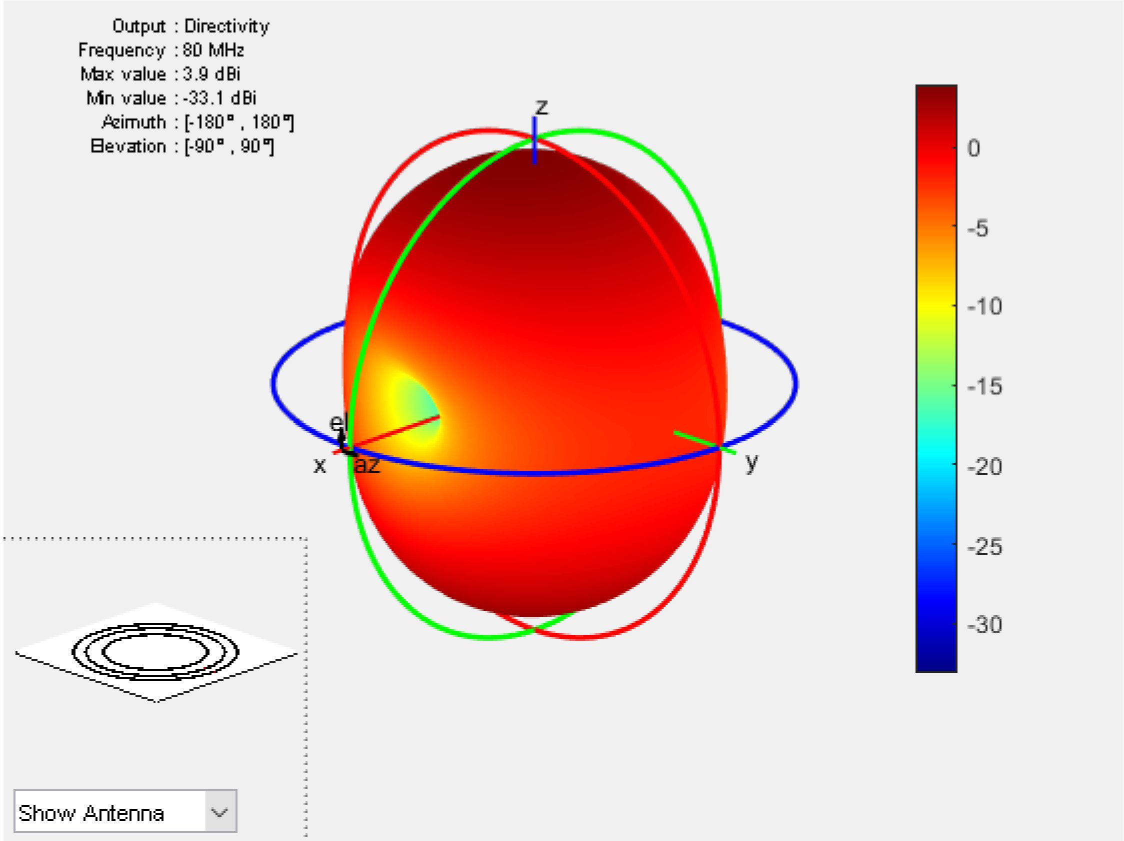
Create a concentric array that uses the origin of circular loop antennas as its position reference.

```
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.

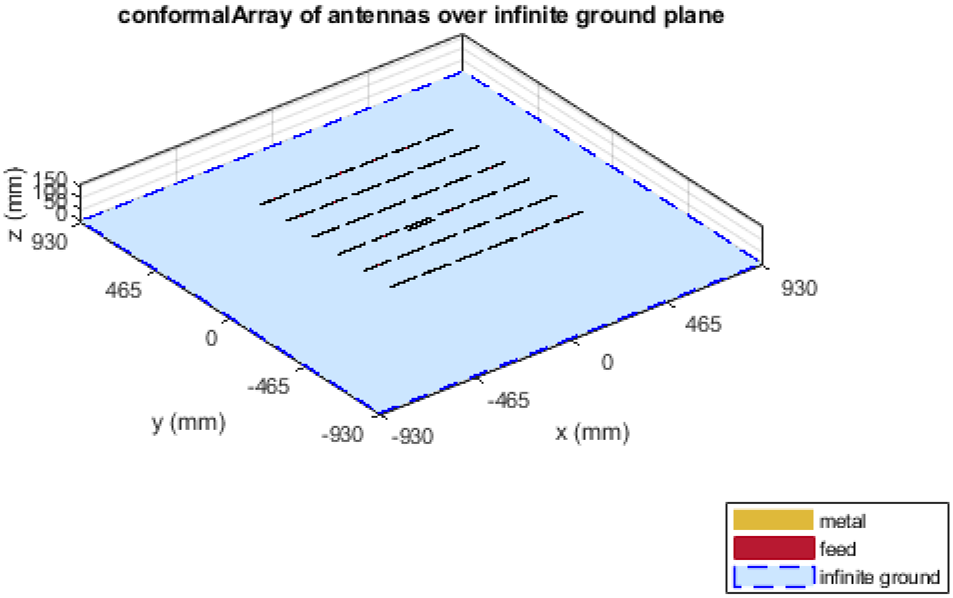
```
d = dipole('Length',0.13,'Width',5e-3,'Tilt',90,'TiltAxis','Y');
```

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)
```

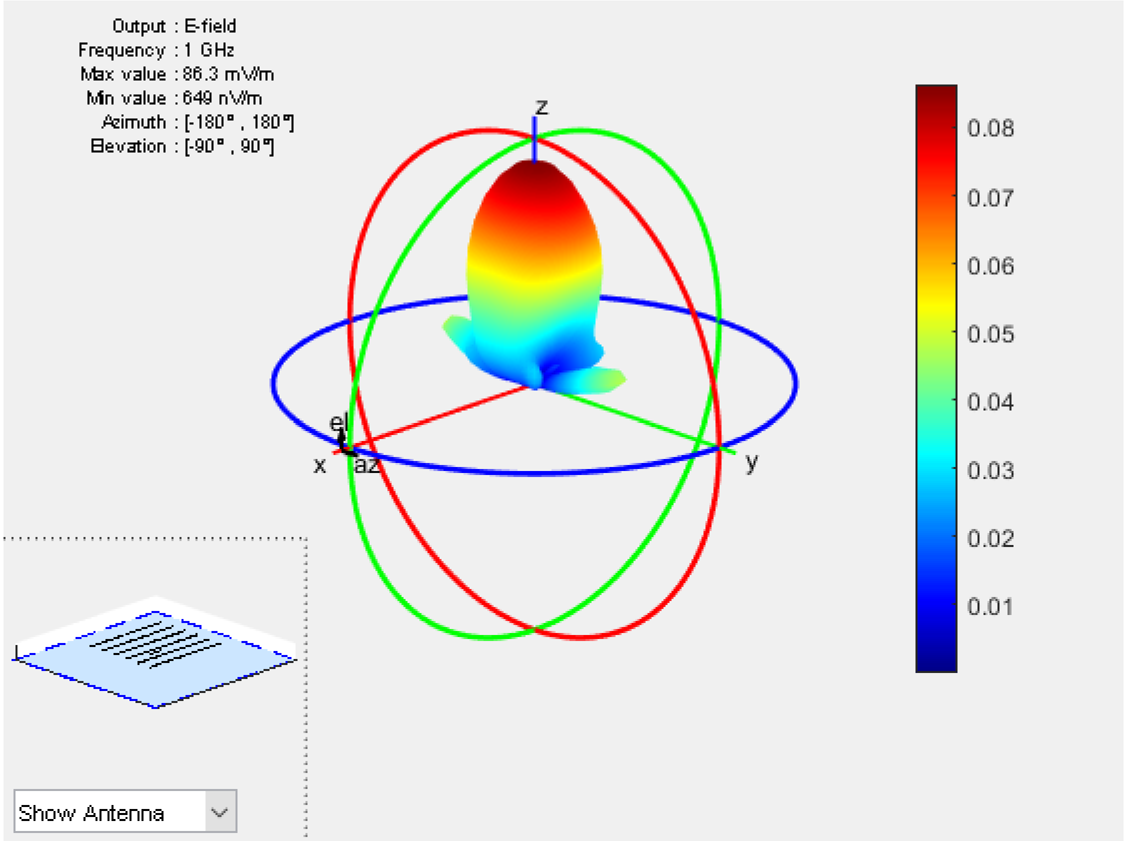


Drive only the reflector antenna with an amplitude of 1.

```
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.

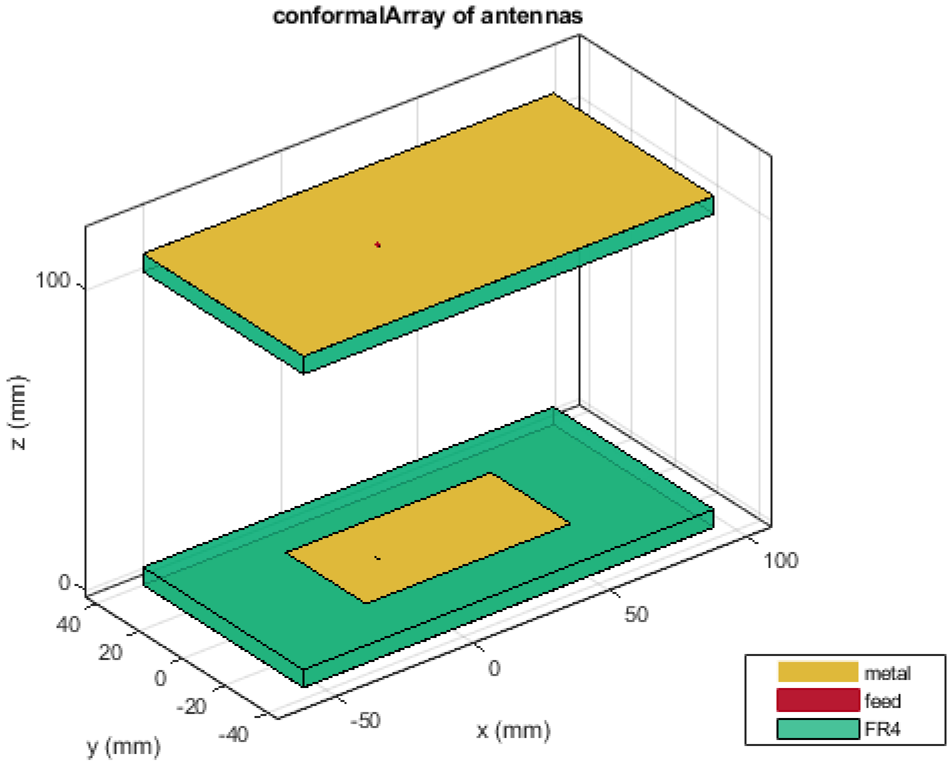
```
d = dielectric('FR4');
p1 = patchMicrostrip('Substrate',d);
p2 = patchMicrostrip('Substrate',d,'Tilt',180);
```

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.

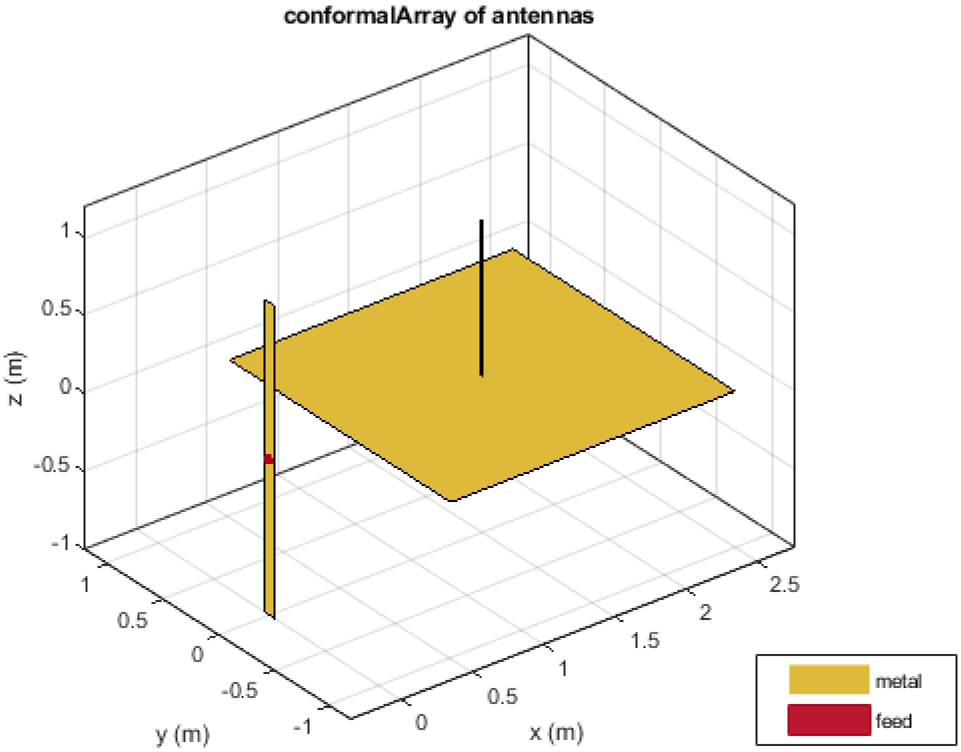
```
c = conformalArray('Element', {dipole, monopole})
```

```
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];
```

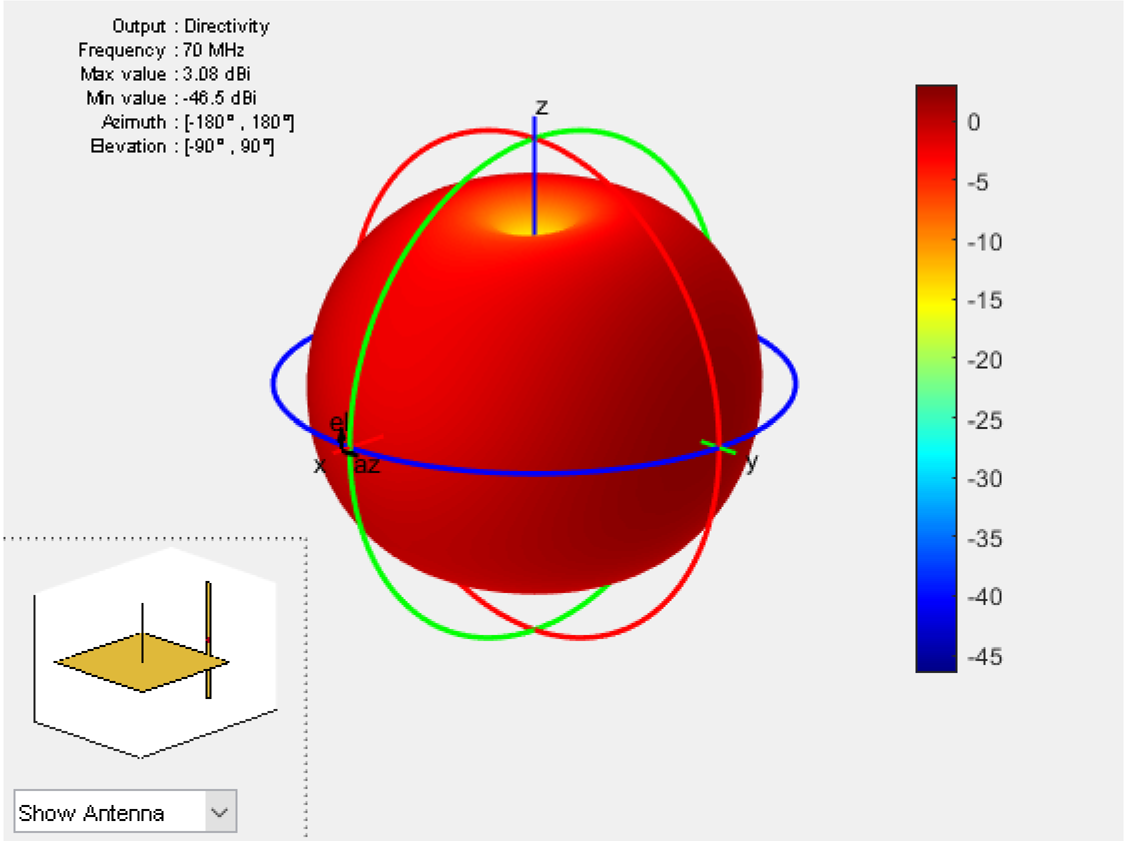
Visualize the array.

```
figure;
show(c);
```



Plot the radiation pattern of the array at 70 MHz.

```
pattern(c, 70e6)
```



Output : Directivity
Frequency : 70 MHz
Max value : 3.08 dBi
Min value : -46.5 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Subarrays of Linear Arrays

Create a subarray of linear arrays at different locations.

```
la = linearArray('ElementSpacing',1)
```

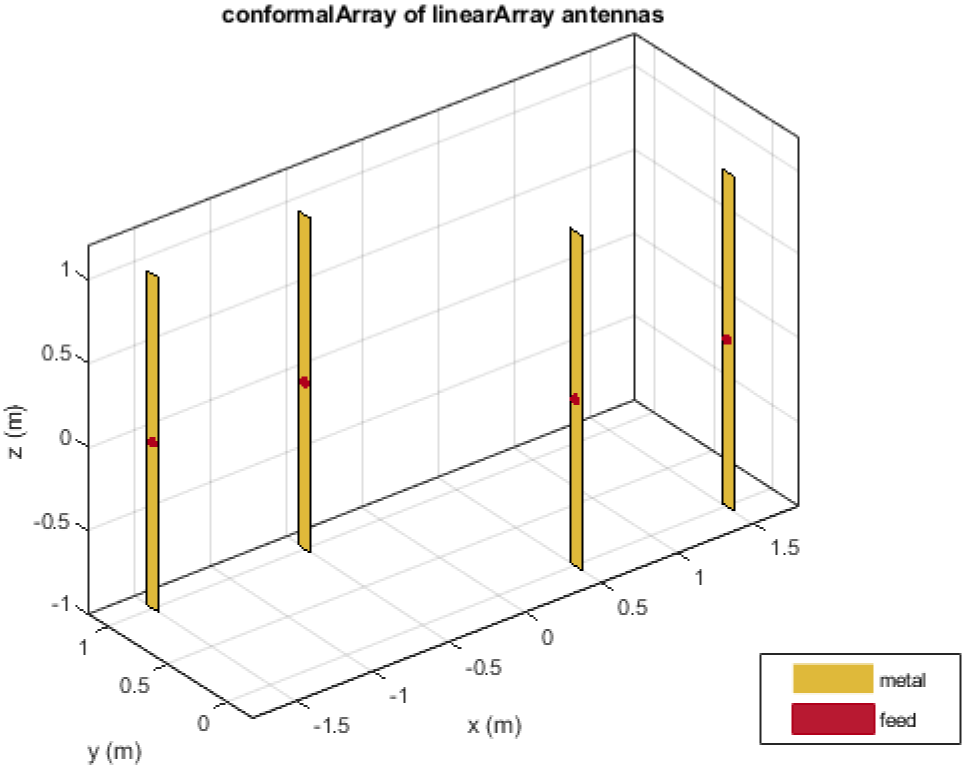
```
la = 
  linearArray with properties:

           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.

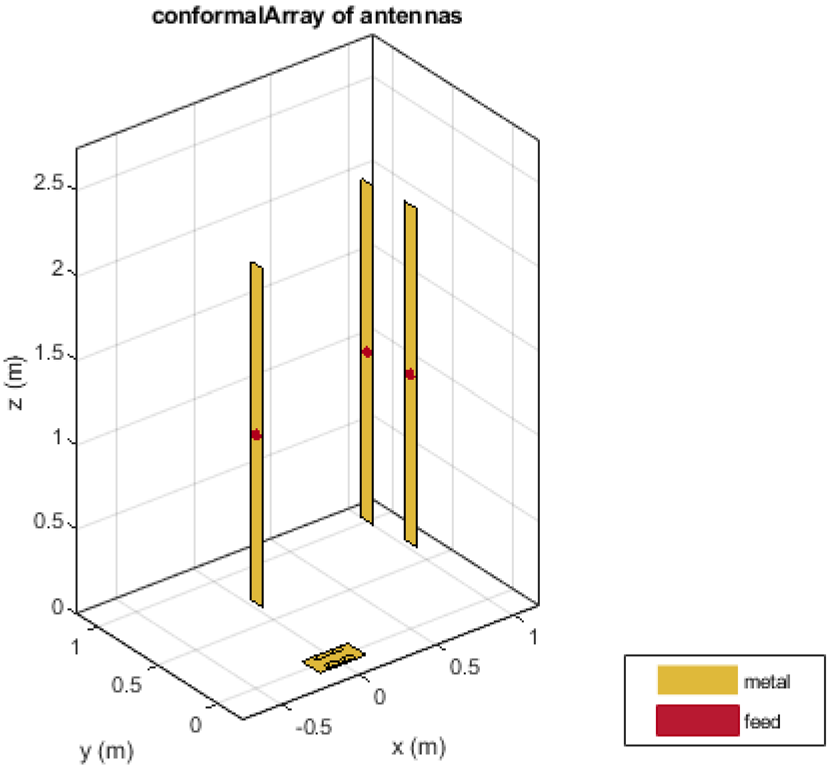
```
la = linearArray('ElementSpacing',1);
```

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)
```

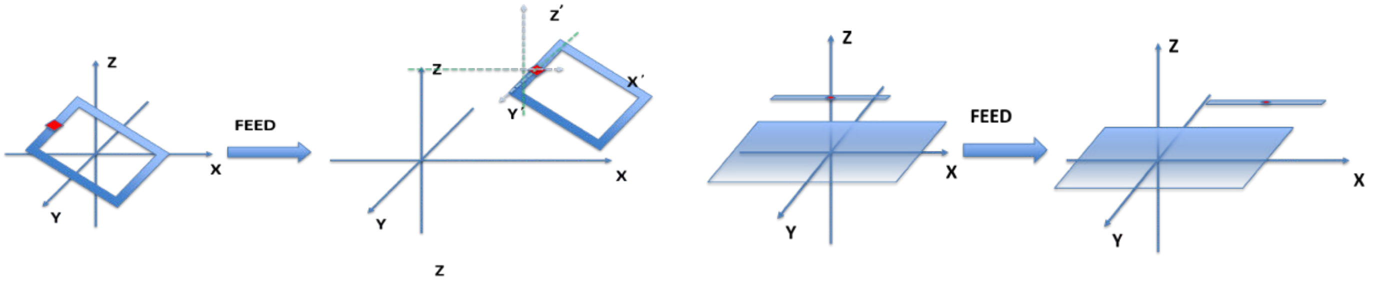


More About

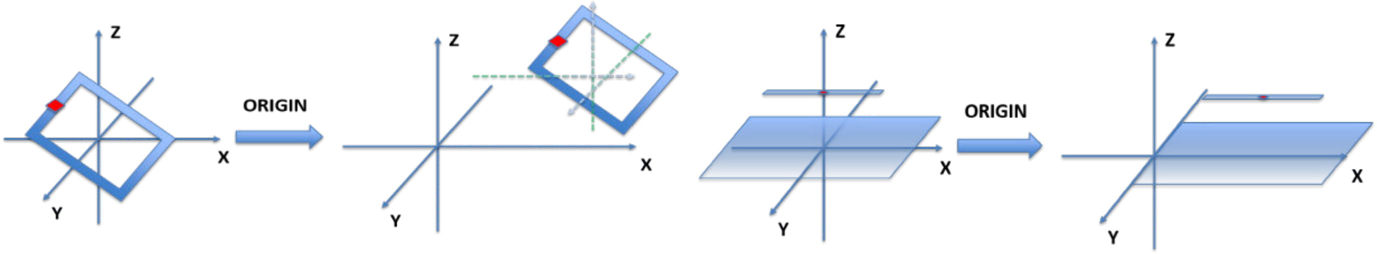
Position Reference

'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:



References

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

See Also

`circularArray` | `infiniteArray` | `linearArray` | `rectangularArray`

Topics

“Rotate Antennas and Arrays”

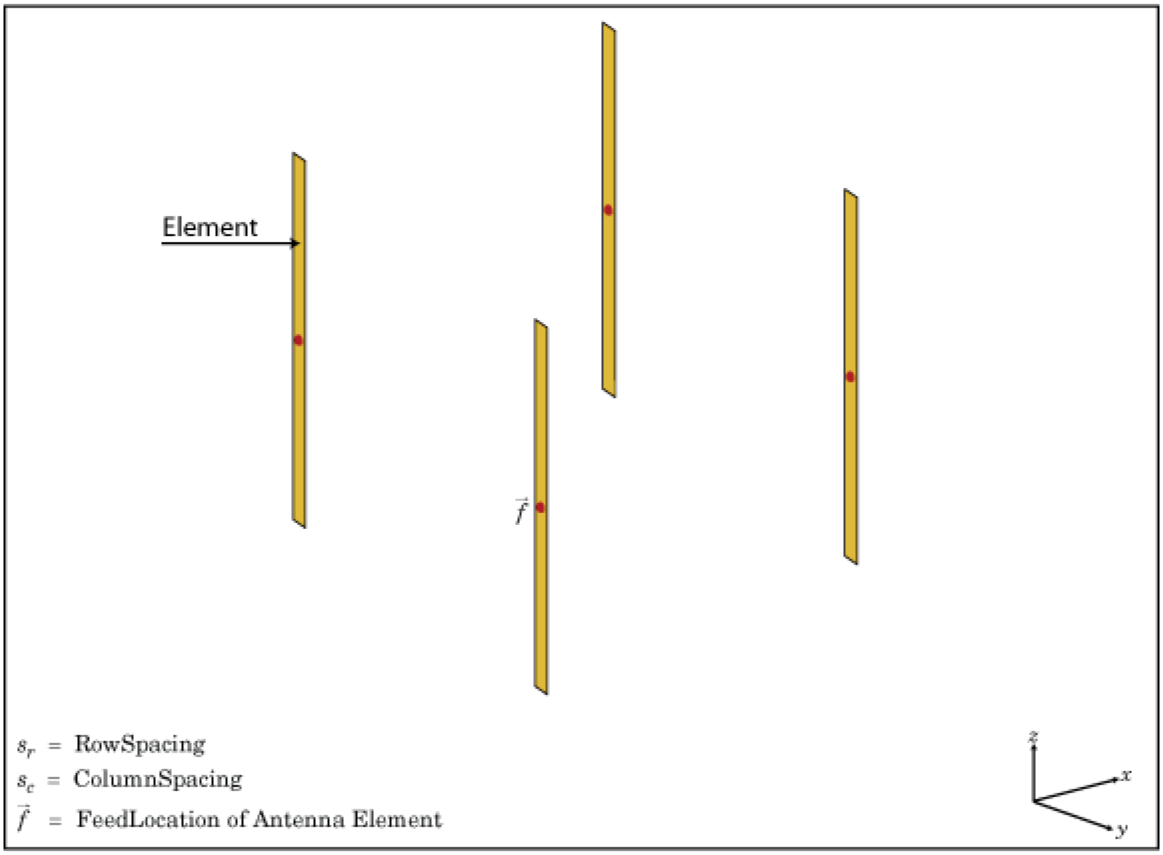
Introduced in R2016a

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: `array.TiltAxis = 'Z'`

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface

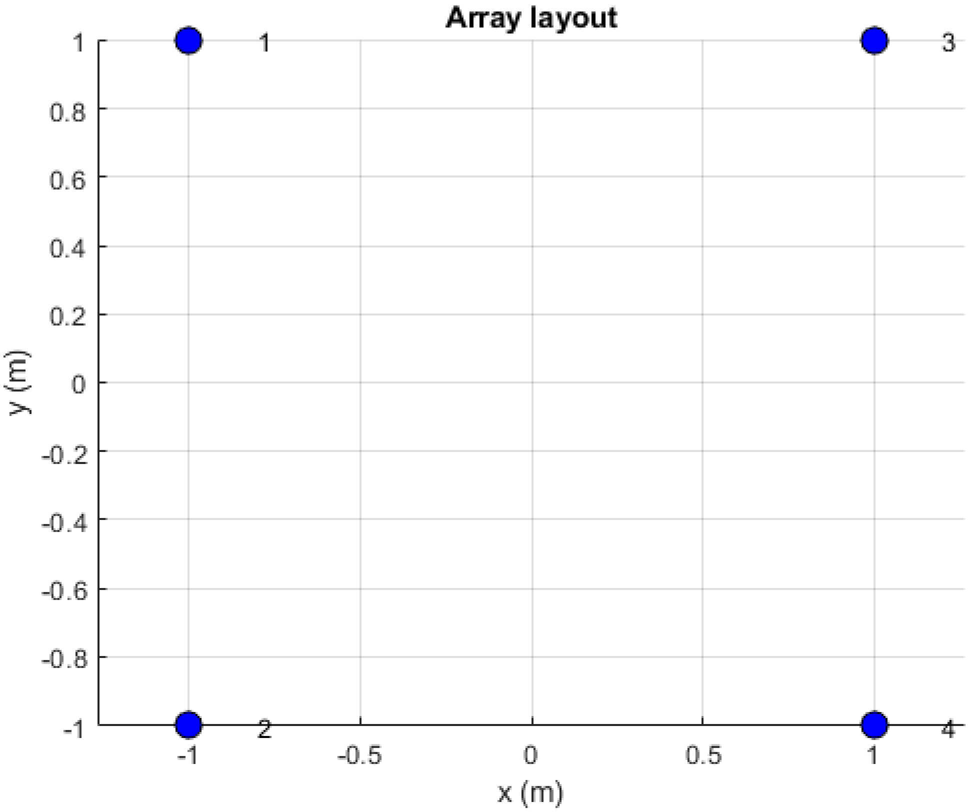
correlation	Correlation coefficient between two antennas in array
current	Current distribution on metal or dielectric antenna or array surface
EHfields	Electric and magnetic fields of antennas; Embedded electric and magnetic fields of antenna element in arrays
impedance	Input impedance of antenna; scan impedance of array
mesh	Mesh properties of metal or 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	S-parameter object

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.

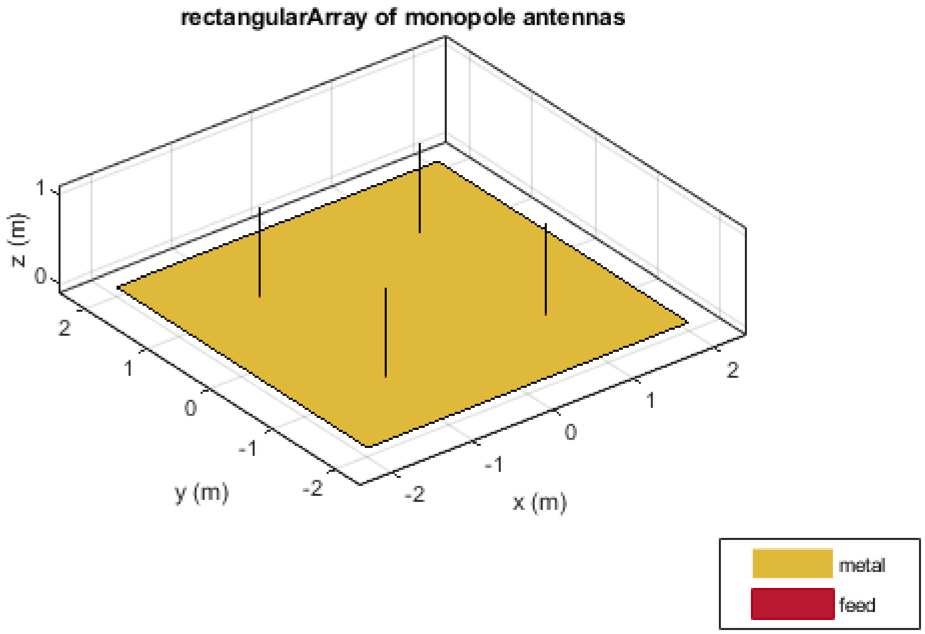
```
h = rectangularArray('Size',[2 2]);
Z = impedance(h,70e6)
```

```
Z = 1×4 complex

  26.2533 -57.2114i  26.2519 -57.2124i  26.2533 -57.2114i  26.2519 -57.2124i
```

Rectangular Array Using Groundplane Antennas

Create a rectangular array of monopoles.

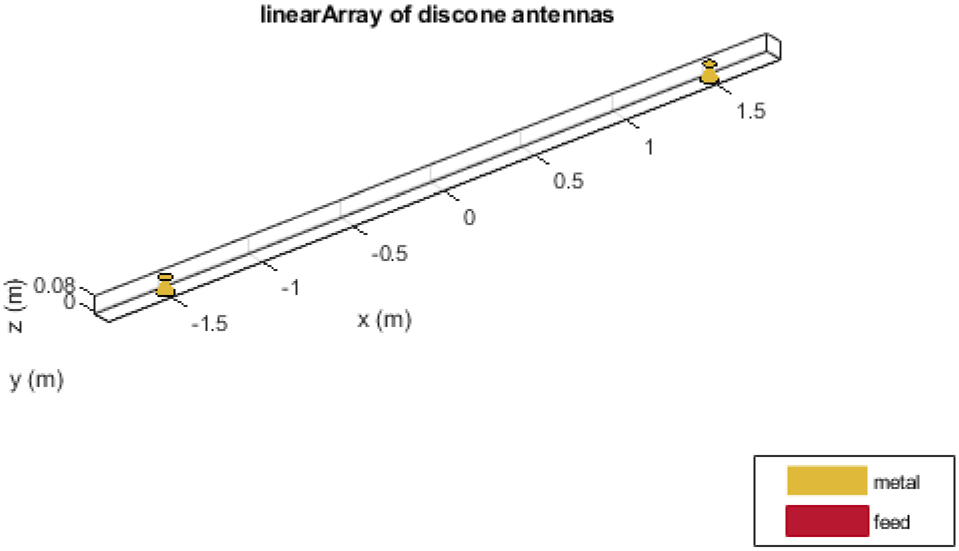
```
m1 = monopole;
mra = rectangularArray('Element',m1);
show(mra);
```



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)
```

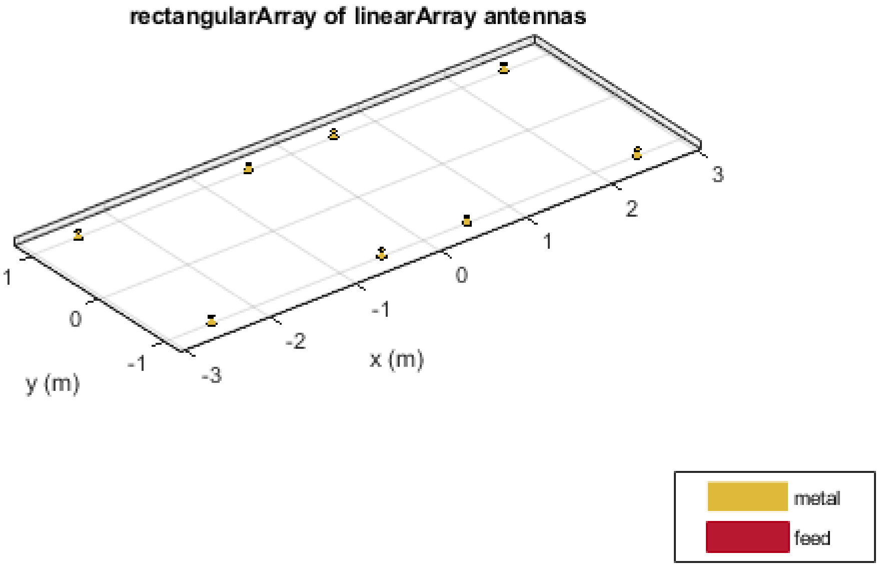


Create a rectangular of the linear array.

```
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)
```



References

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

See Also

`circularArray` | `conformalArray` | `infiniteArray` | `linearArray`

Topics

"Rotate Antennas and Arrays"

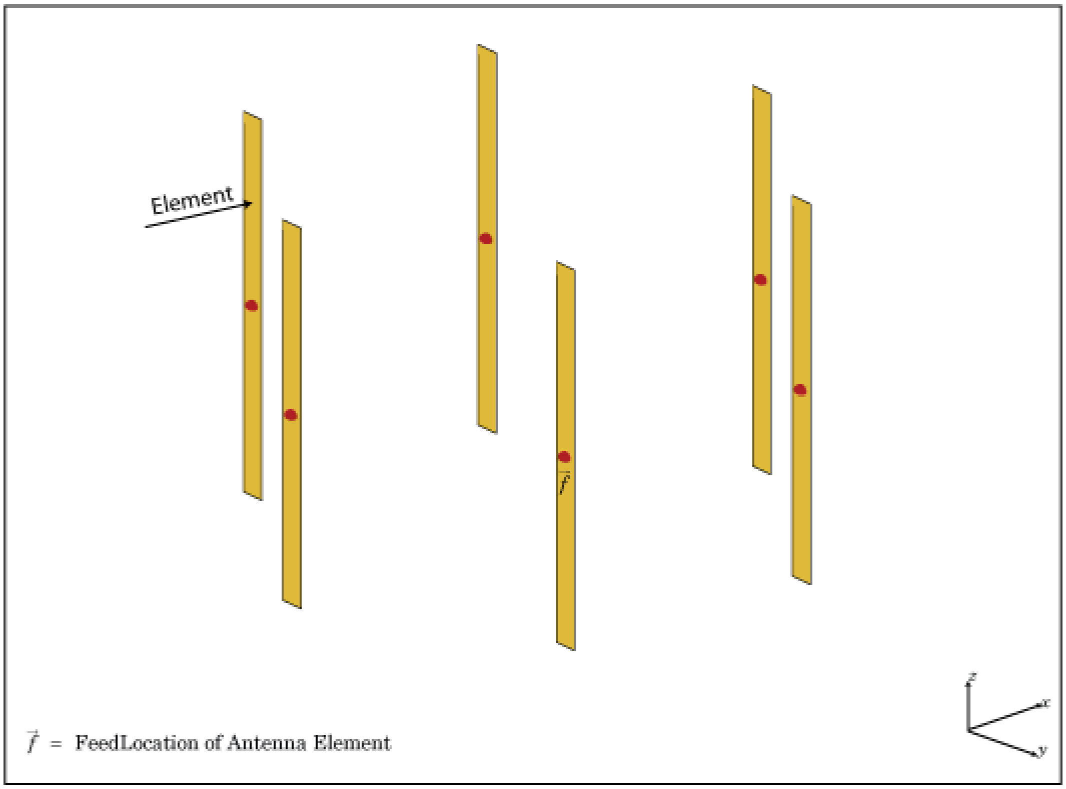
Introduced in R2015a

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: `array.TiltAxis = 'Z'`

Analysis Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
layout	Display array or PCB stack layout
show	Display antenna or array structure; display shape as filled patch
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on metal or dielectric 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 or 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	S-parameter object

Examples

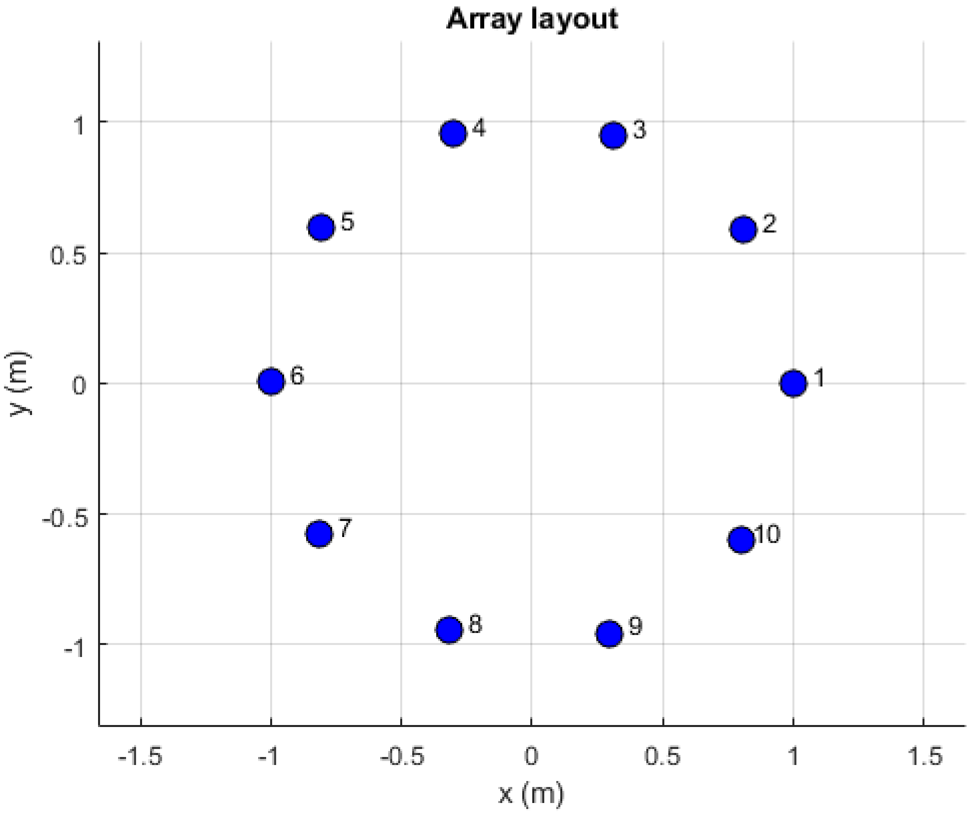
Plot Elevation Pattern of Circular Antenna Array

Create a circular antenna array using 10 antenna elements. View the layout of the antenna elements in the array.

```
ca = circularArray('NumElements',10)
```

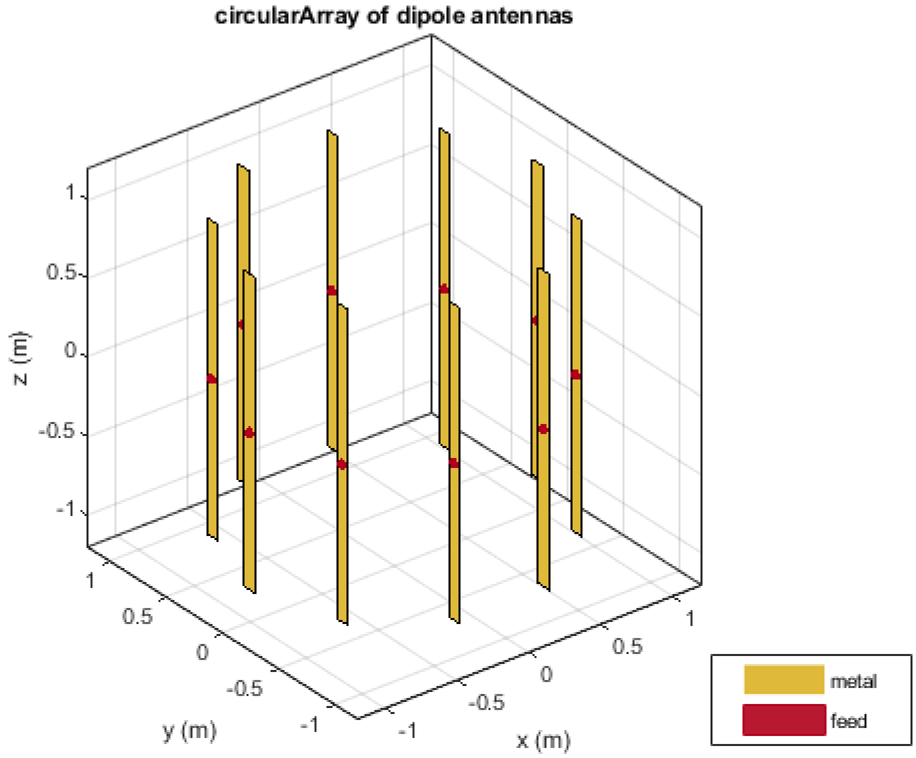
```
ca = 
  circularArray with properties:

           Element: [1x1 dipole]
       NumElements: 10
            Radius: 1
       AngleOffset: 0
    AmplitudeTaper: 1
        PhaseShift: 0
              Tilt: 0
          TiltAxis: [1 0 0]
```

```
figure;
layout(ca)
```



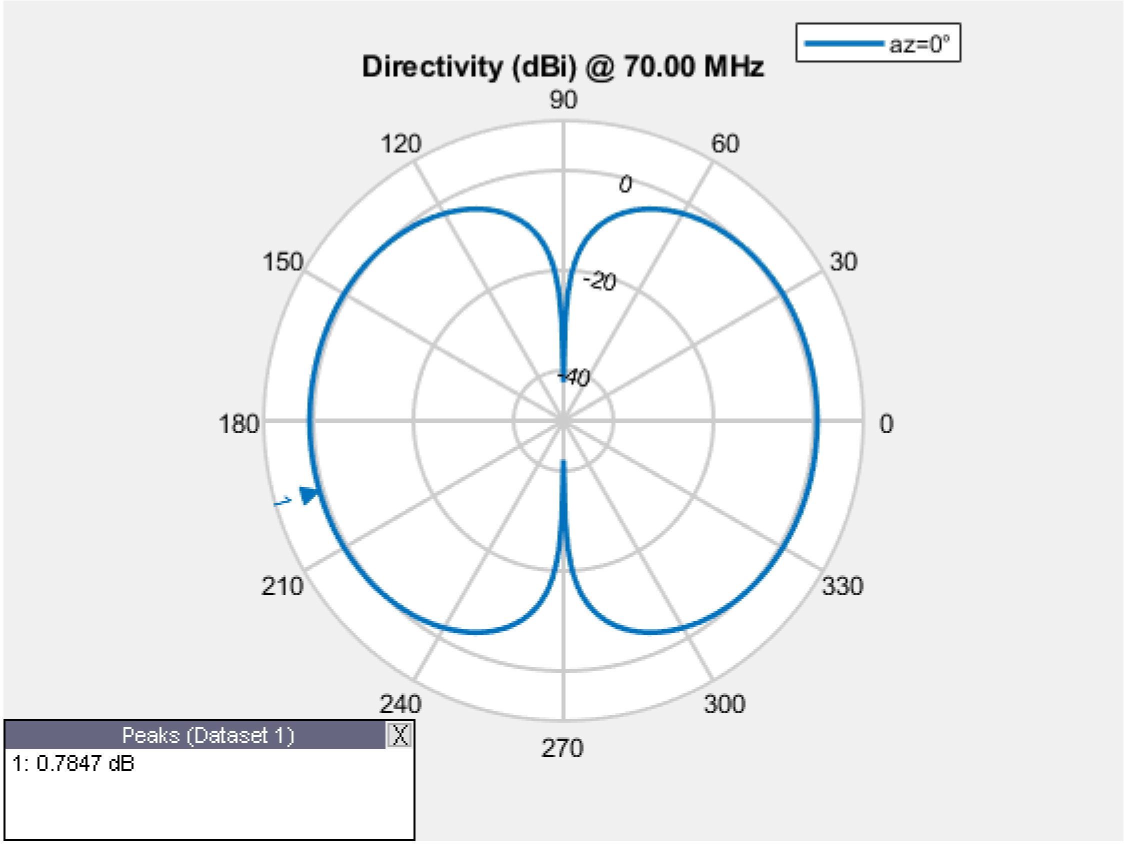
Display the array.

```
figure;
show(ca)
```



Plot the elevation pattern of the circular array at a frequency of 70 MHz.

```
figure;
patternElevation(ca,70e6)
```



1: 0.7847 dB

See Also

`conformalArray` | `infiniteArray` | `linearArray` | `rectangularArray`

Topics

“Rotate Antennas and Arrays”

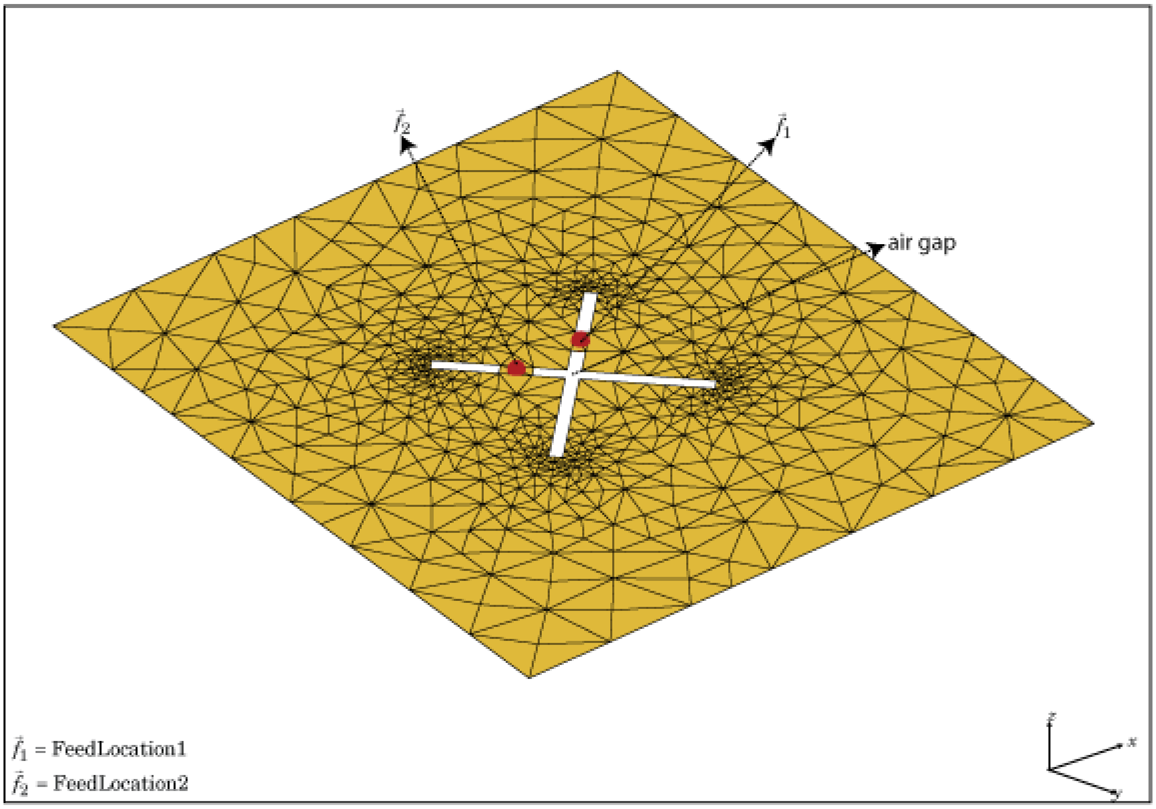
Introduced in R2016b

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 or array structure; display shape as filled patch
info	Display information about antenna or array
createFeed	Create feed locations for custom array
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
correlation	Correlation coefficient between two antennas in array
current	Current distribution on metal or dielectric 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 or 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	S-parameter object

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

  64.3919 - 7.8288i  58.9595 -11.3554i
```

References

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

See Also

`conformalArray` | `linearArray` | `rectangularArray`

Topics

"Rotate Antennas and Arrays"

Introduced in R2015b

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: `array.TiltAxis = 'Z'`

Object Functions

show	Display antenna or array structure; display shape as filled patch
info	Display information about antenna or array
axialRatio	Axial ratio of antenna
beamwidth	Beamwidth of antenna
charge	Charge distribution on metal or dielectric antenna or array surface
current	Current distribution on metal or dielectric antenna or array surface
design	Design prototype antenna or arrays for resonance at 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 or 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	S-parameter object
show	Display antenna or array structure; display shape as filled patch
vswr	Voltage standing wave ratio of antenna

Examples

Create Custom Slot Antenna Array

Create a custom array using `customArrayGeometry`. Visualize it and plot the impedance. Also, visualize the current distribution on 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]   = em.internal.makeplate(Lp,Wp,2,'linear');
```

Create slots on the ground plane with a length 0.05 m and a width of 0.4 m.

```
Ls  = 0.05;
Ws  = 0.4;
```

```
offset = 0.12;
[~,p2]   = em.internal.makeplate(Ls,Ws,2,'linear');
p3 = em.internal.translateshape(p2, [offset, 0, 0]);
p2 = em.internal.translateshape(p2, [-offset, 0, 0]);
```

Create a feed in between the slots on the ground plane.

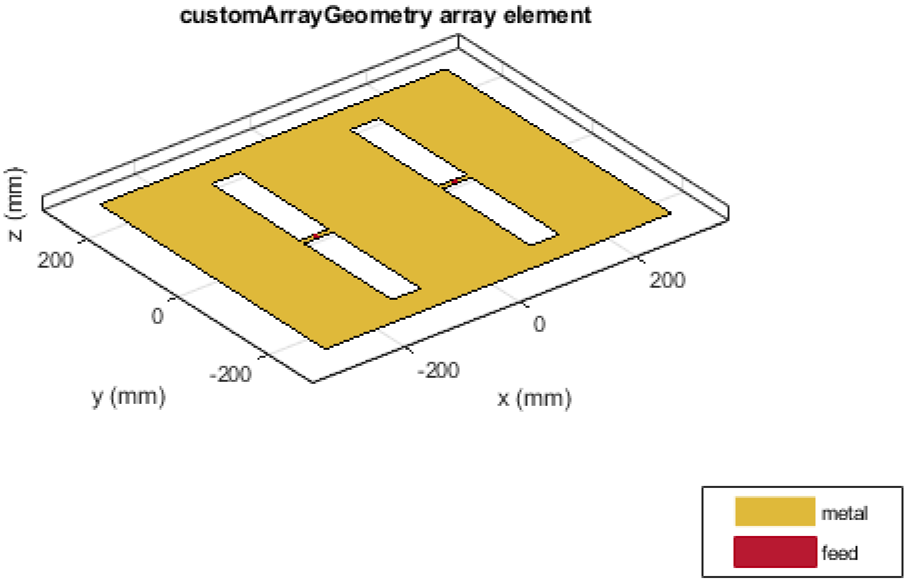
```
Wf  = 0.01;
[~,p4]   = em.internal.makeplate(Ls,Wf,2,'linear');
p5 = em.internal.translateshape(p4, [offset, 0, 0]);
p4 = em.internal.translateshape(p4, [-offset, 0, 0]);
```

Create an array using the slotted ground plane.

```
carray = customArrayGeometry;
carray.Boundary = {p1', p2', p3', p4', p5'};
carray.Operation= 'P1-P2-P3+P4+P5';
carray.NumFeeds = 2;
carray.FeedWidth= [0.01 0.01];
carray.FeedLocation = [-offset,0,0 ; offset,0,0];
```

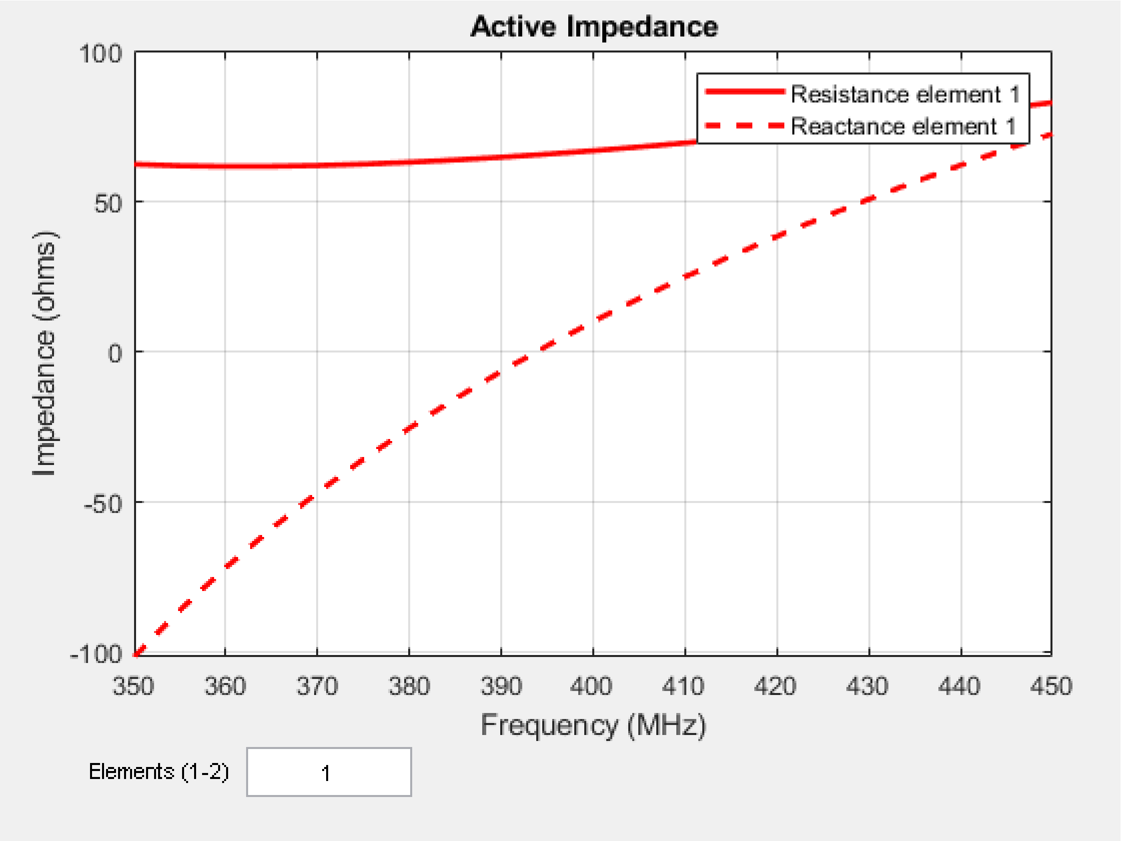
Visualize the array.

```
figure; show(carray);
```



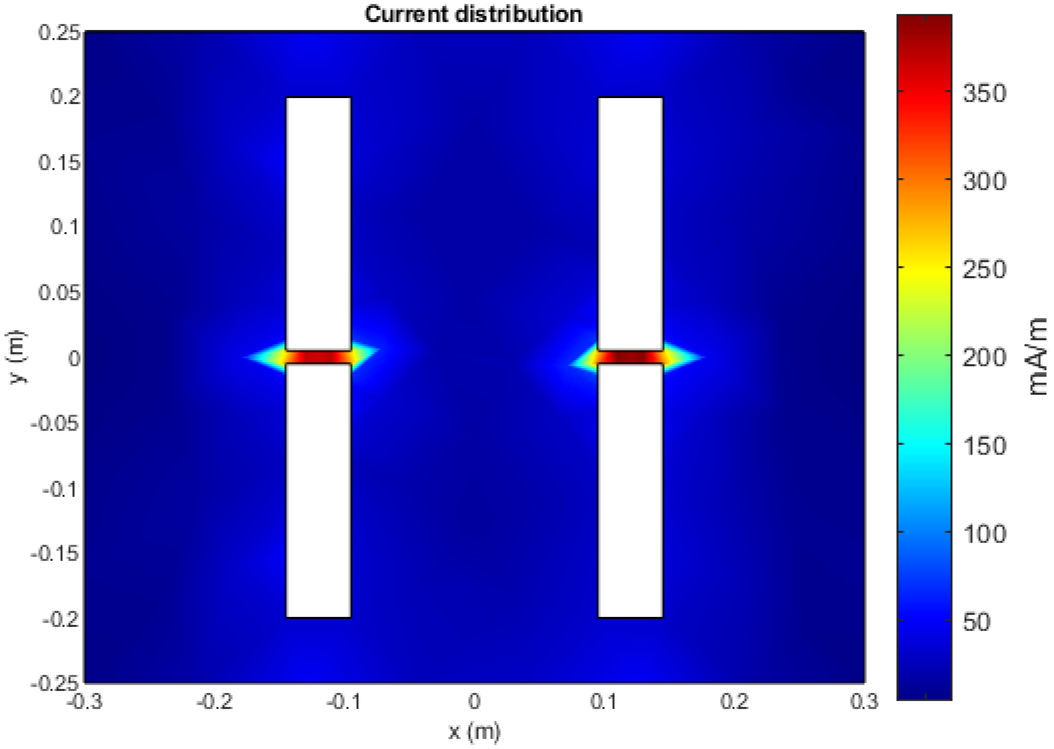
Calculate the impedance of the array using the frequency range of 350 MHz to 450 MHz.

```
figure; impedance(carray, 350e6:5e6:450e6);
```



Visualize the current distribution of the array at 410 MHz.

```
figure; current(carray, 410e6);
```



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"

Introduced in R2017a

5

Methods

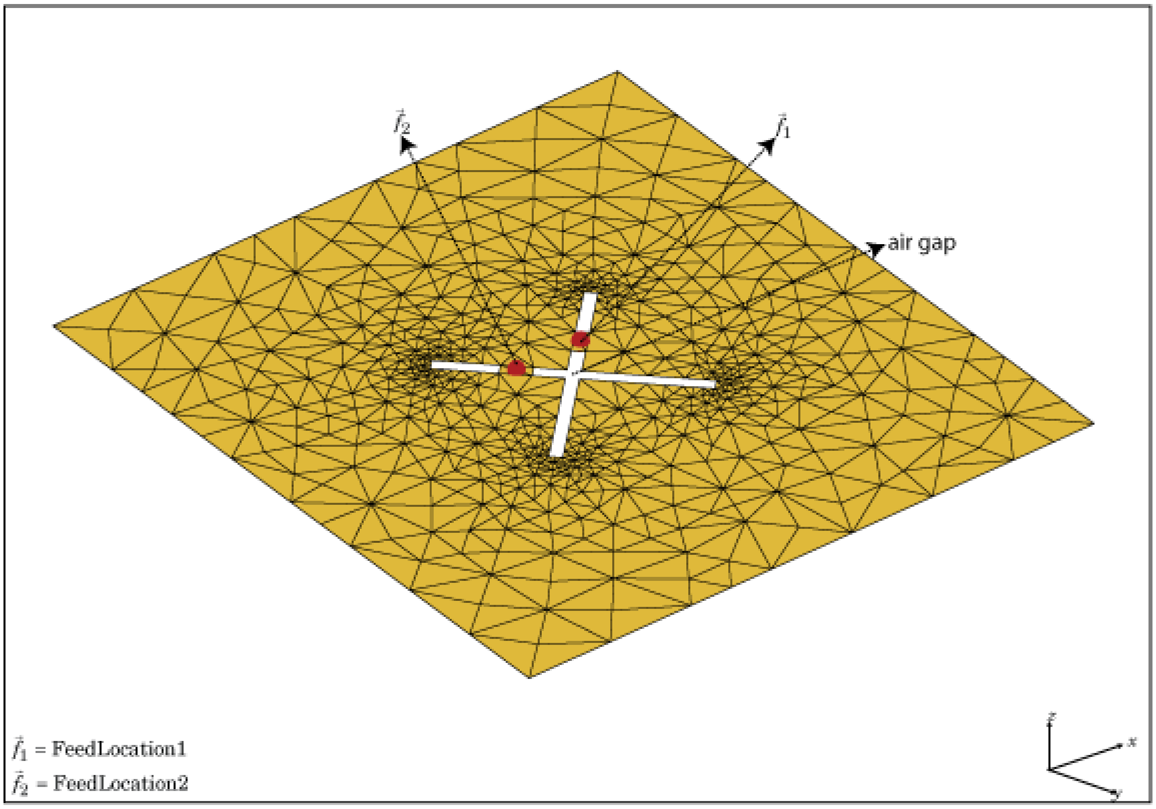
createFeed

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 handle

Custom mesh array, specified as a scalar handle.

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.

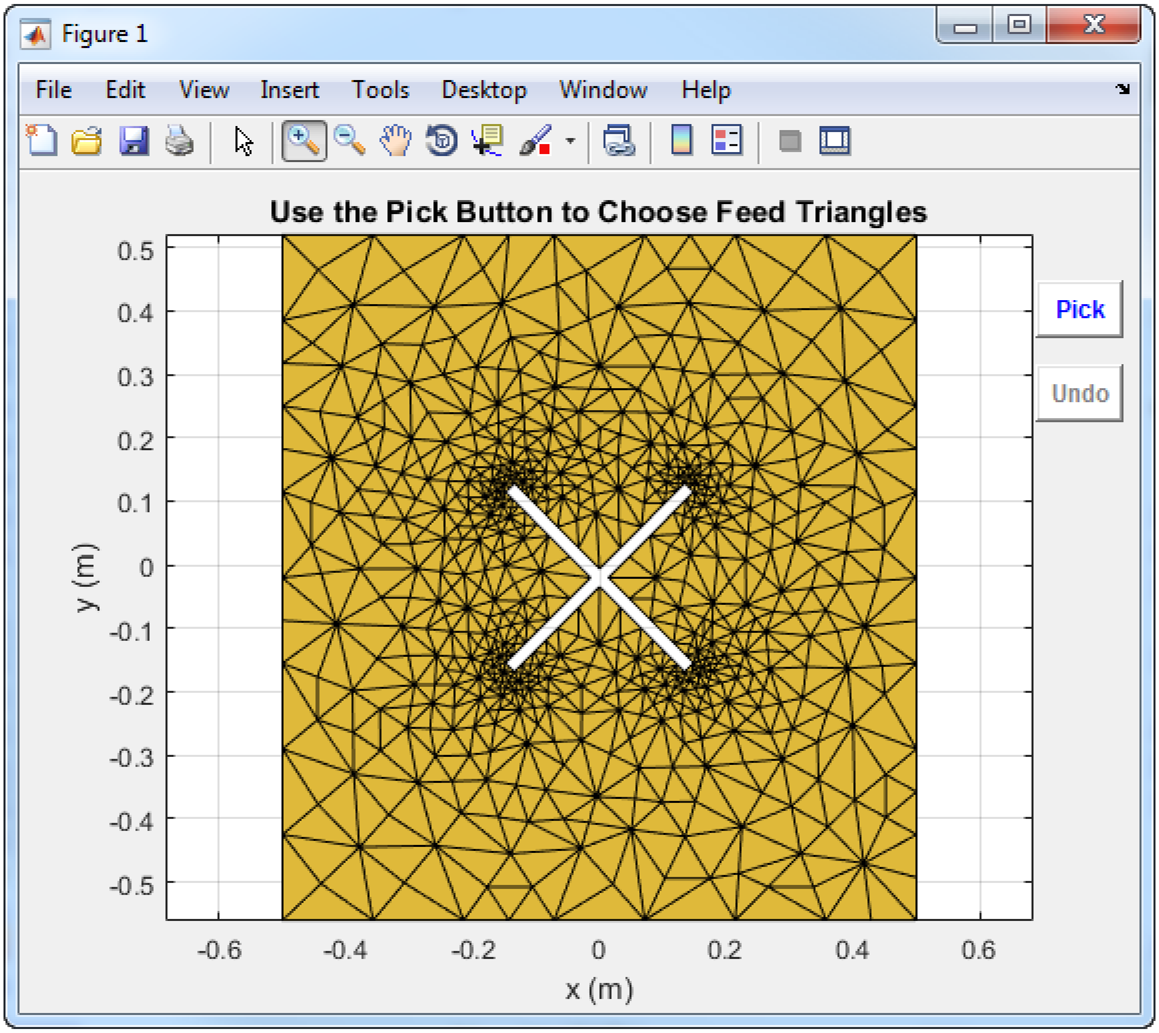
Load a 2-D custom mesh. Create a custom array using the points and triangles.

```
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]
```

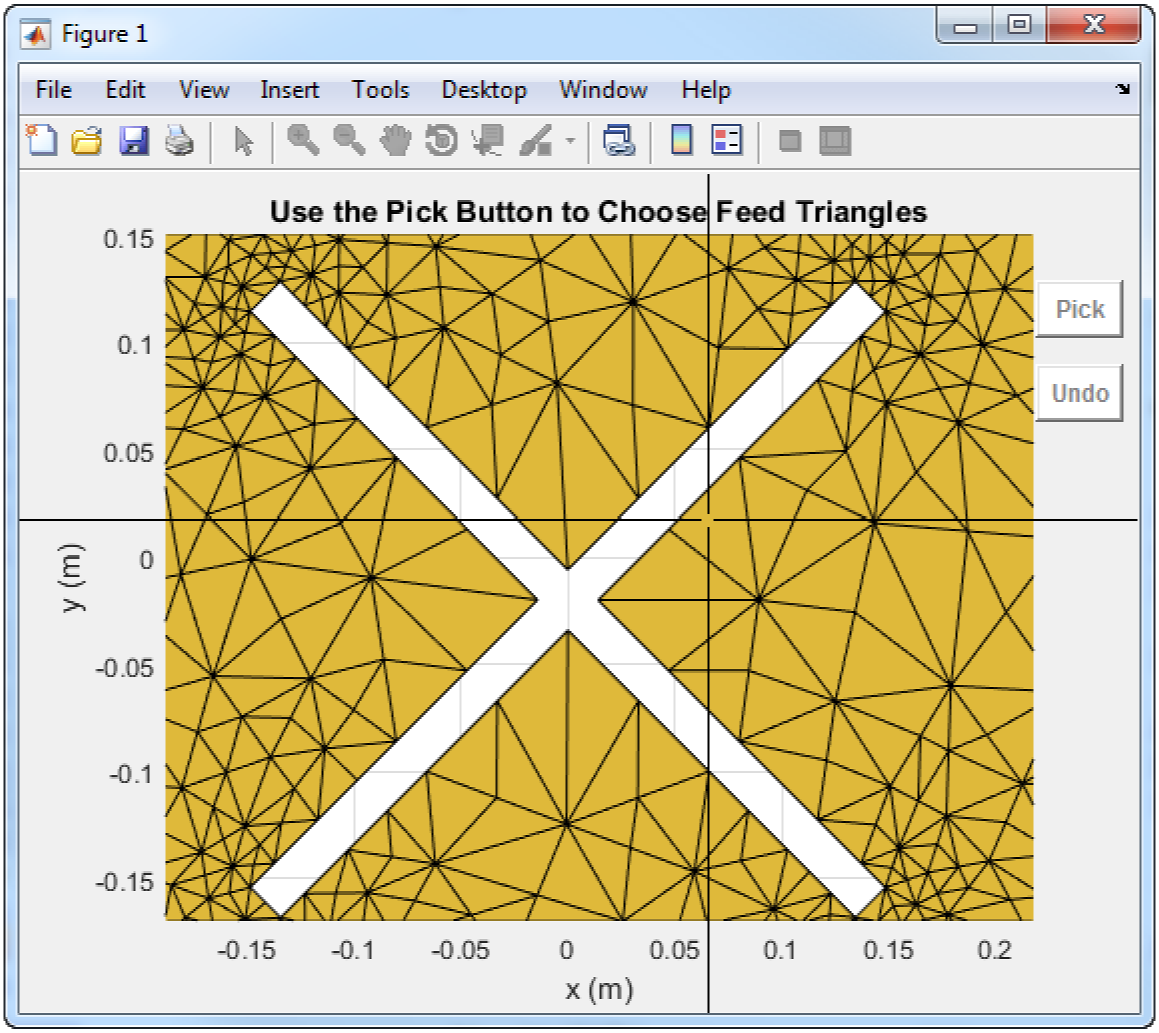
Use the `createFeed` function to view the array mesh structure. In this array mesh view, you see **Pick** and **Undo** buttons. The **Pick** button is highlighted.

```
createFeed(c)
```

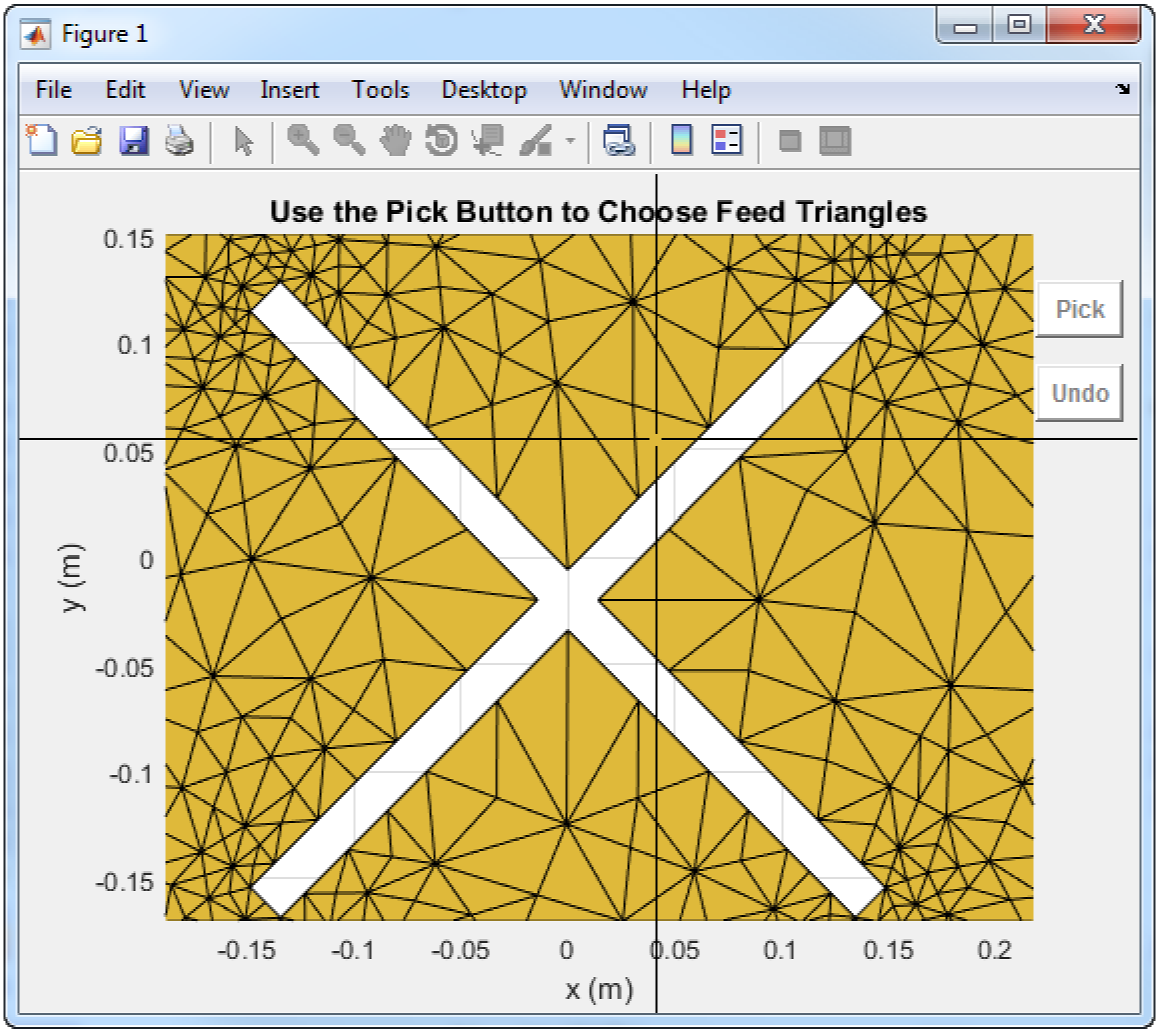


Click **Pick** to display the cross hairs. For an array with two feeds, select two pairs (four points) in the mesh. To specify a feed-region for the, zoom in and select two points each, one inside each triangle on either side of the air gap. Select the points using the cross hairs.

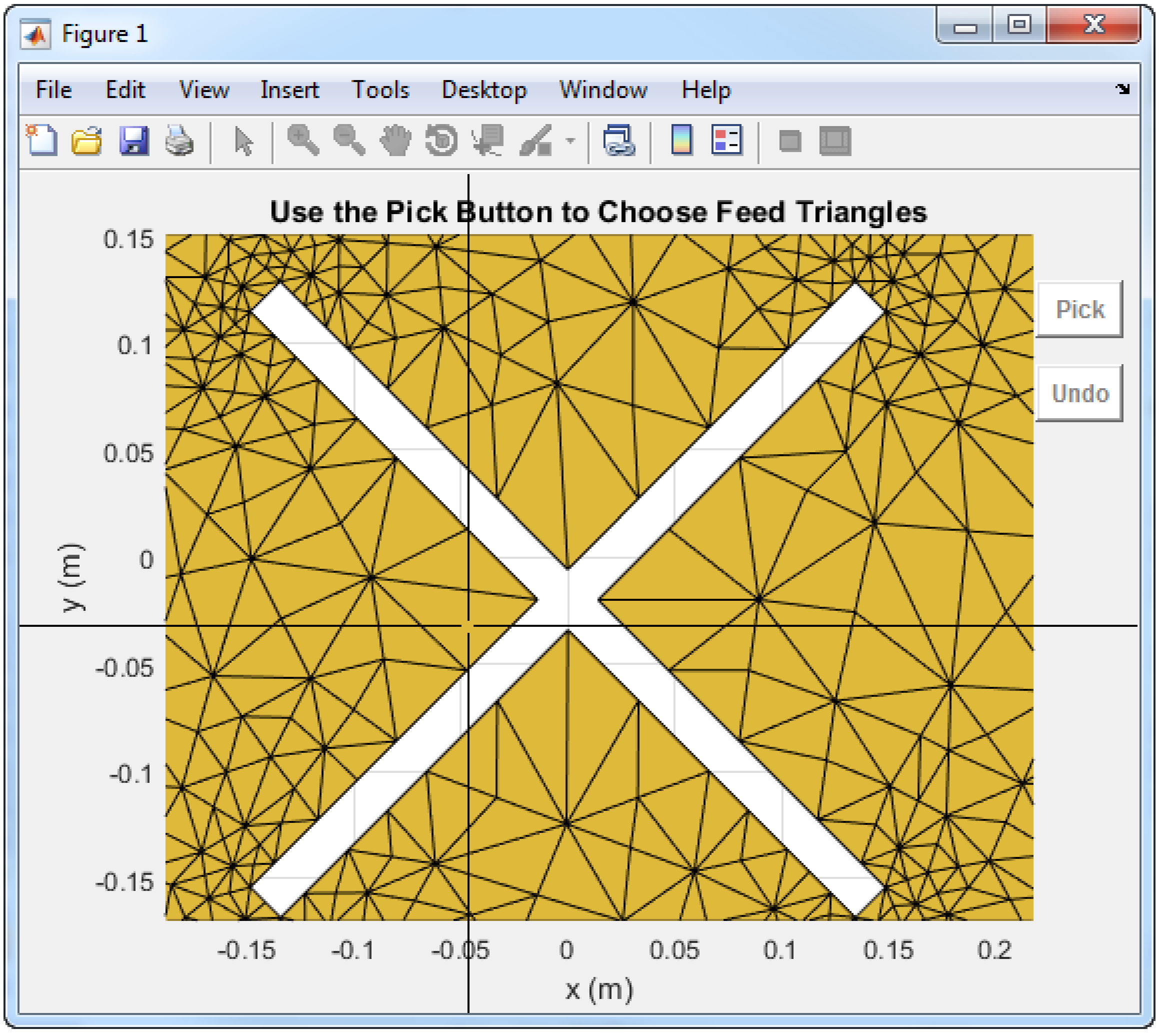
- Select the first triangle for feedpoint 1.



- Select the second triangle on the other side of the air gap for feedpoint 1.



- Select first triangle for feedpoint 2.



- Select the second triangle on the other side of the air gap for feedpoint 2.

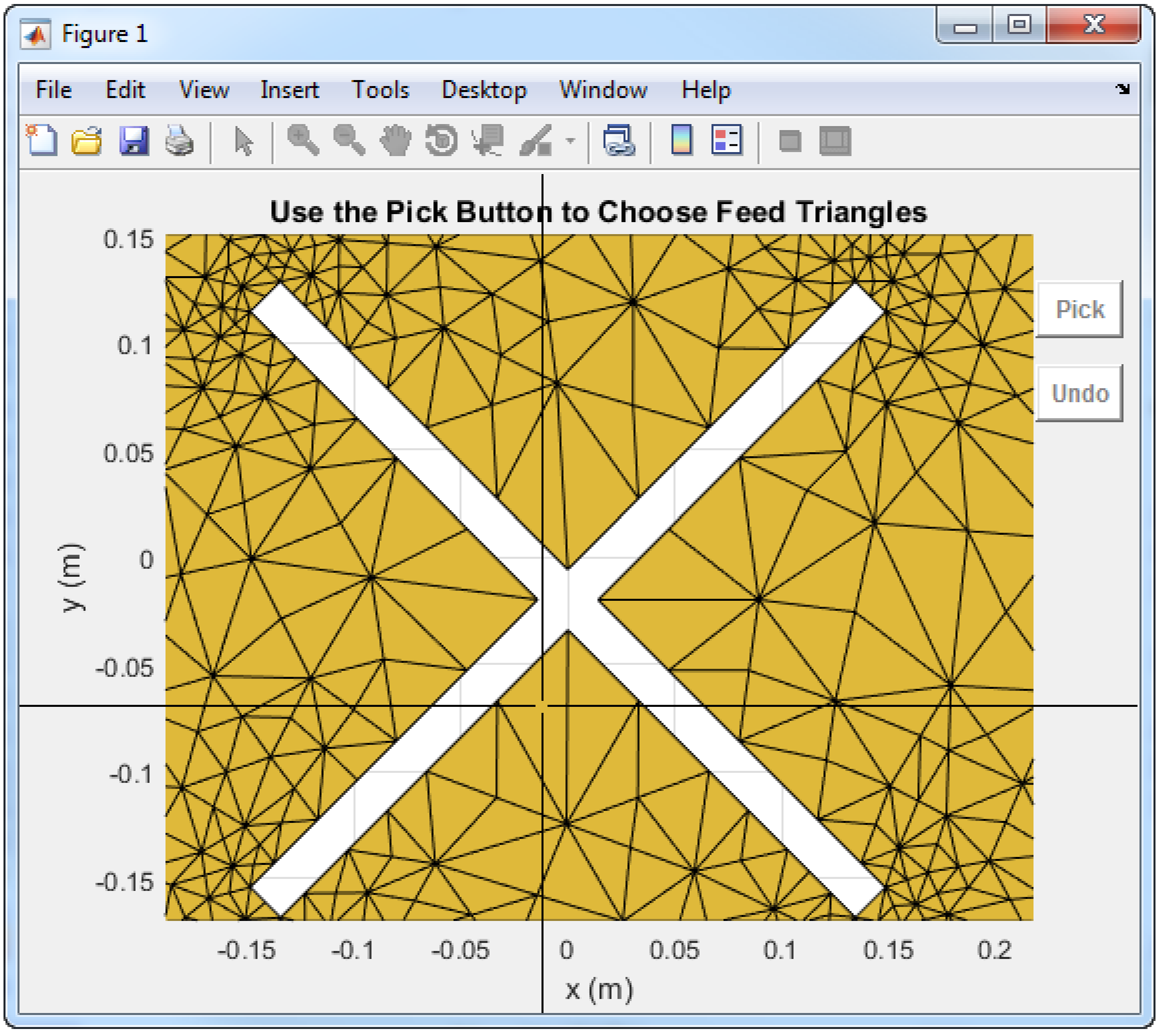
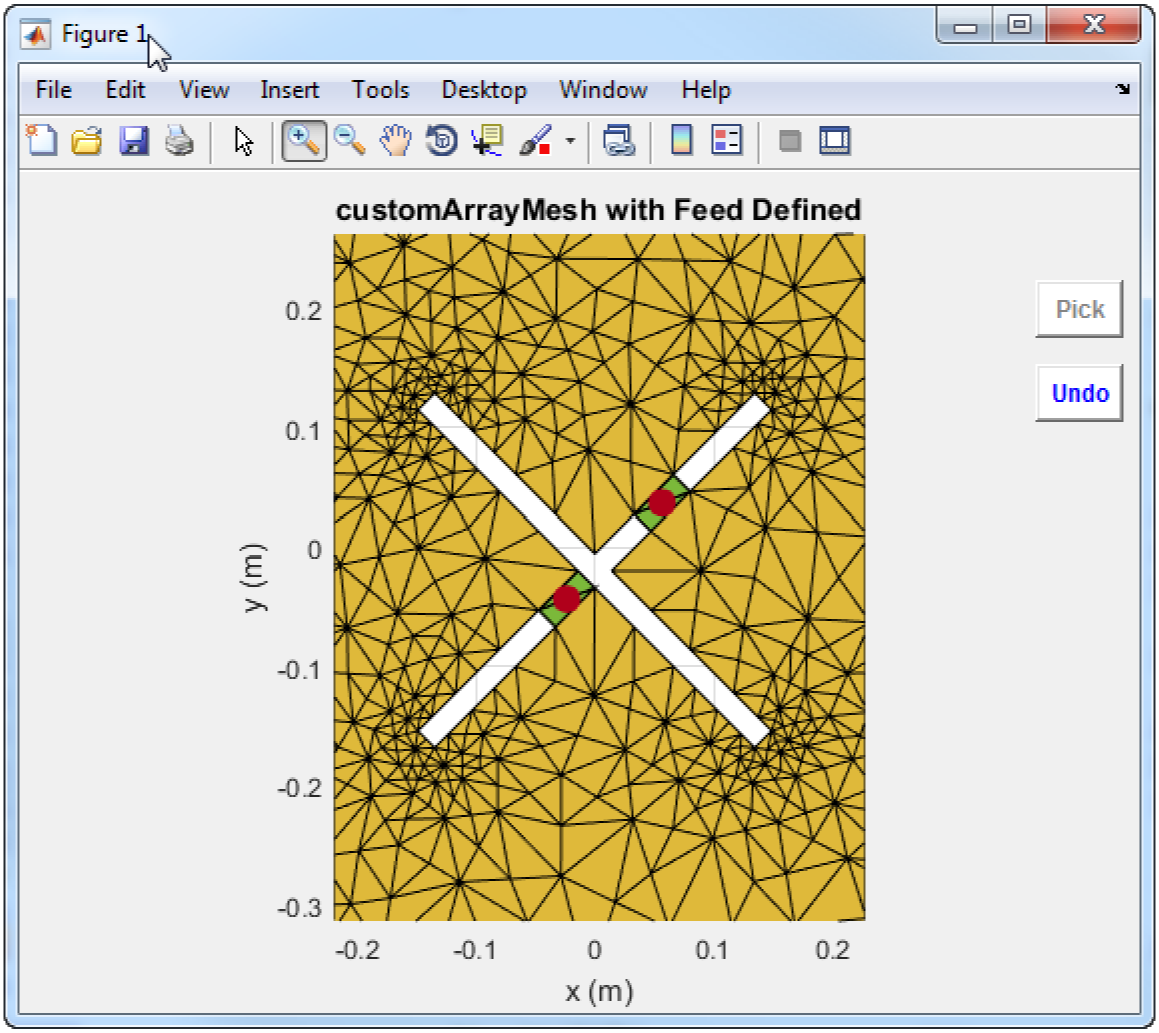


Figure 1
File Edit View Insert Tools Desktop Window Help
Use the Pick Button to Choose Feed Triangles
Pick
Undo
y (m)
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
x (m)

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.

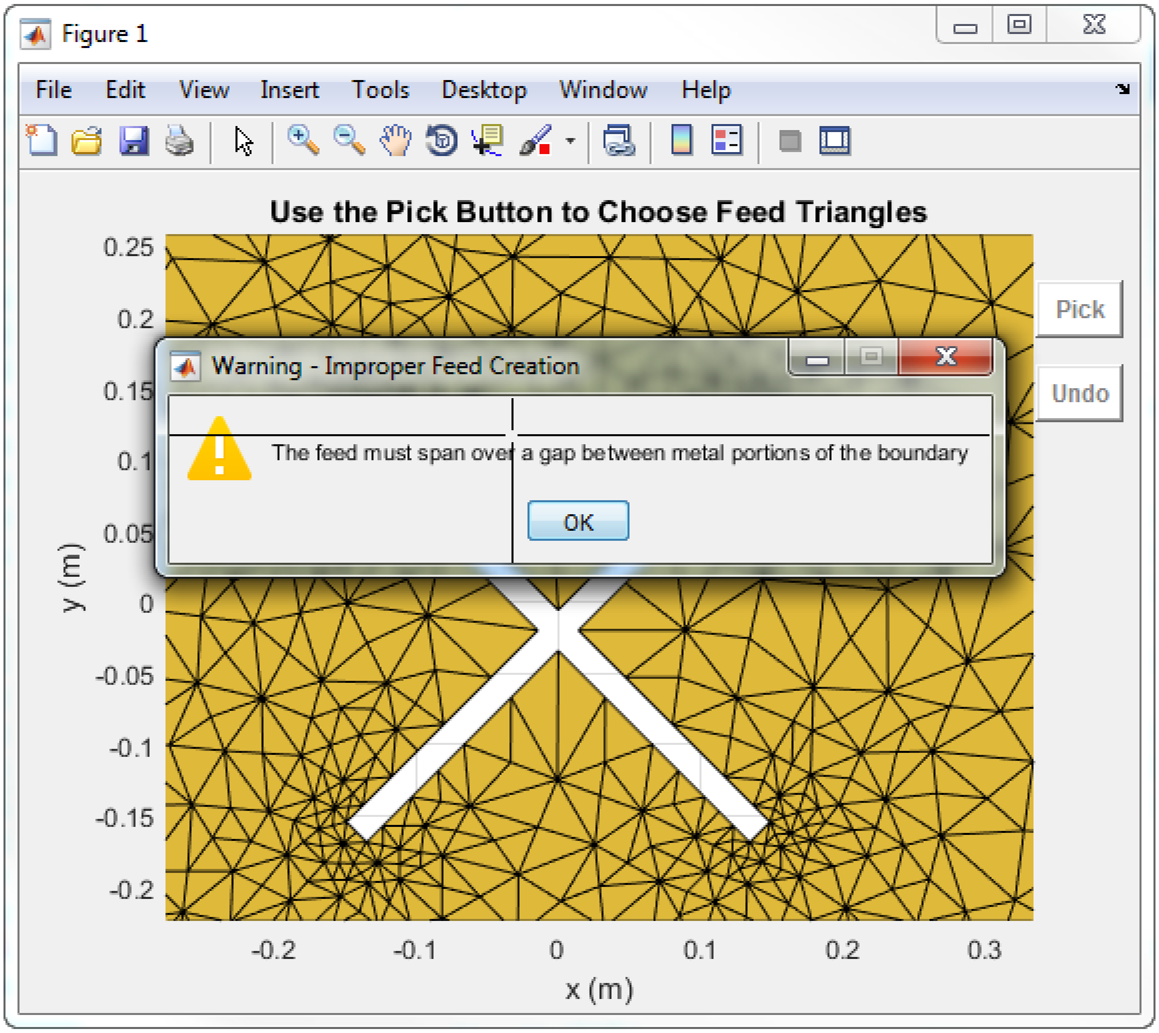


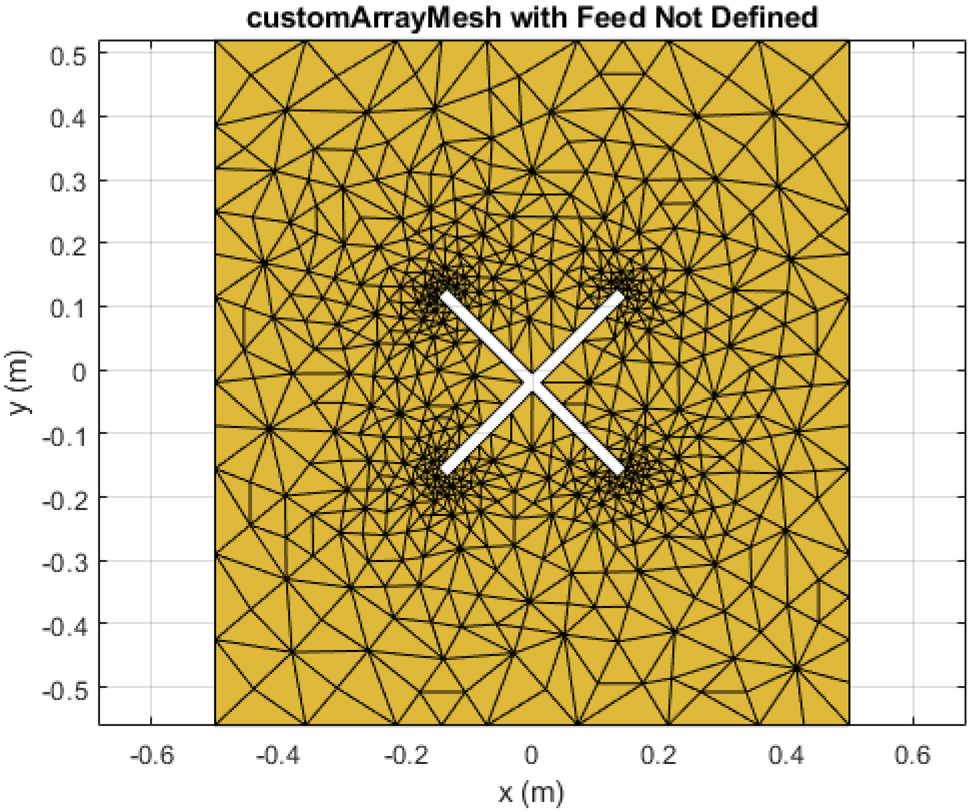
Figure 1
File Edit View Insert Tools Desktop Window Help
Use the Pick Button to Choose Feed Triangles
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
y (m)
-0.2
-0.1
0
0.1
0.2
0.3
x (m)
Pick
Undo
Warning - Improper Feed Creation
OK

The feed must span over a gap between metal portions of the boundary

Create Feed for Custom Array Mesh

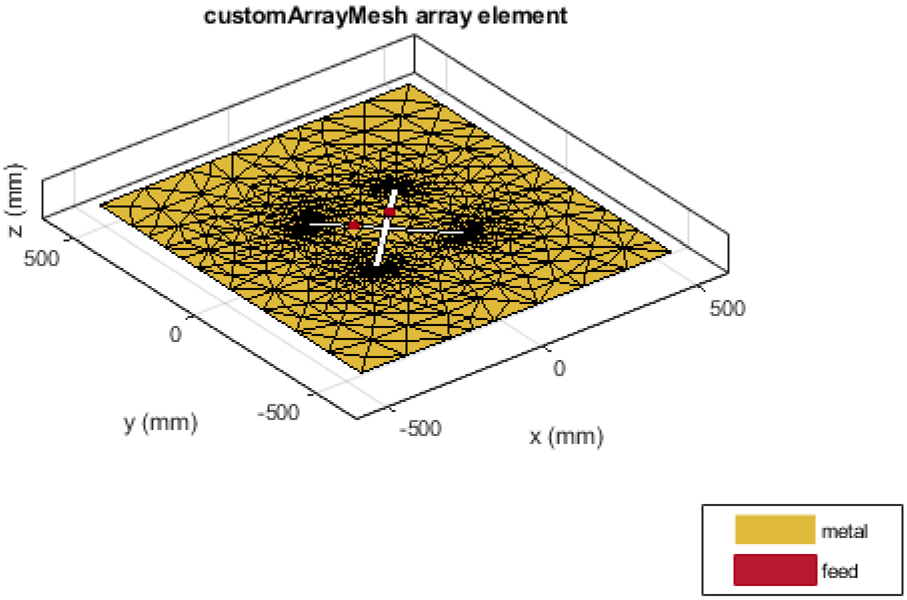
Load a custom mesh and create an array.

```
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)
```



See Also

`returnLoss` | `sparameters`

Introduced in R2016a

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.

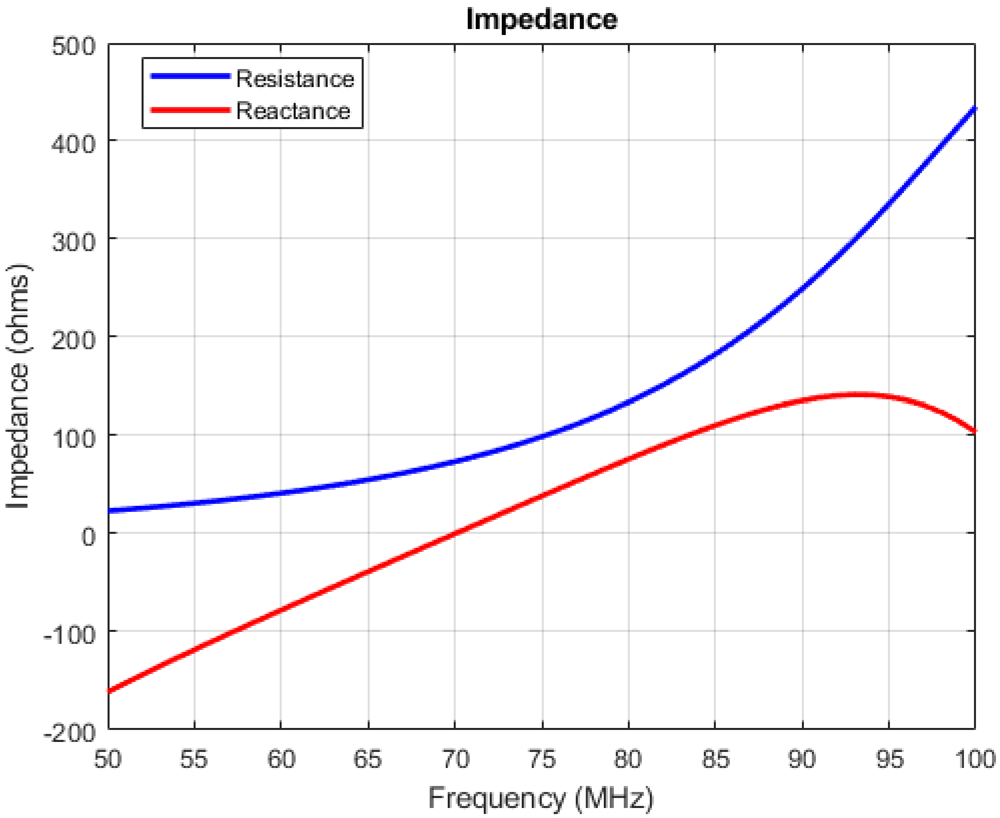
`z = impedance(array,frequency,elementnumber)` returns the scan impedance of a specified antenna element in an array.

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
10^2 ×

   0.2892 - 1.7385i   0.2892 - 1.7385i
   0.3005 - 1.6573i   0.3005 - 1.6573i
   0.3119 - 1.5778i   0.3119 - 1.5778i
   0.3237 - 1.4999i   0.3237 - 1.4999i
   0.3357 - 1.4233i   0.3357 - 1.4233i
   0.3479 - 1.3481i   0.3479 - 1.3481i
   0.3605 - 1.2740i   0.3605 - 1.2740i
   0.3734 - 1.2009i   0.3734 - 1.2009i
   0.3866 - 1.1287i   0.3866 - 1.1287i
   0.4002 - 1.0573i   0.4002 - 1.0573i
      ⋮
```

Input Arguments

`antenna` — Antenna or array object

scalar handle

Antenna object, specified as a scalar handle.

`array` — Array object

scalar handle

Array object, specified as a scalar handle.

`frequency` — Frequency range used to calculate impedance

vector in Hz

Frequency range to calculate impedance, specified as a vector 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.

See Also

`returnLoss`

Introduced in R2015a

sparameters

S-parameter object

Syntax

```
sobj = sparameters(filename)

sobj = sparameters(data,freq)
sobj = sparameters(data,freq, Z0)

sobj = sparameters(filterobj,freq)
sobj = sparameters(filterobj,freq,Z0)

sobj = sparameters(netparamobj)
sobj = sparameters(netparamobj, Z0)

sobj = sparameters(rfdataobj)
sobj = sparameters(rfcktobj)

sobj = sparameters(mnobj)
sobj = sparameters(mnobj,freq)
sobj = sparameters(mnobj,freq,Z0)
sobj = sparameters(mnobj,freq,Z0,circuitindices)

sobj = sparameters(antenna,freq,Z0)
sobj = sparameters(array,freq,Z0)
```

Description

`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(filterobj,freq)` calculates the S-parameters of a filter object, `filterobj` with the default reference impedance.

`sobj = sparameters(filterobj,freq,Z0)` calculates the S-parameters of a filter object, `filterobj` 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`.

`sobj = sparameters(rfdataobj)` extracts network data from `rfdataobj` and converts it into S-parameter object.

`sobj = sparameters(rfcktobj)` extracts network data from `rfcktobj` and converts it into S-parameter object.

`sobj = sparameters(mnobj)` returns the s-parameters of the best created matching network, evaluated at a frequency list constructed from source and load impedance.

`sobj = sparameters(mnobj,freq)` returns the s-parameters of the best created matching network at each specified frequency.

`sobj = sparameters(mnobj,freq,Z0)` returns the s-parameters of the best created matching network at each specified frequency and characteristic impedance, `Z0`.

`sobj = sparameters(mnobj,freq,Z0,circuitindices)` returns an array of S-parameter objects, one object for each circuit indicated in `circuitindices`.

`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,Z0)` calculates the complex s-parameters for an `array` object over specified frequency values and for a given reference impedance, `Z0`.

Examples

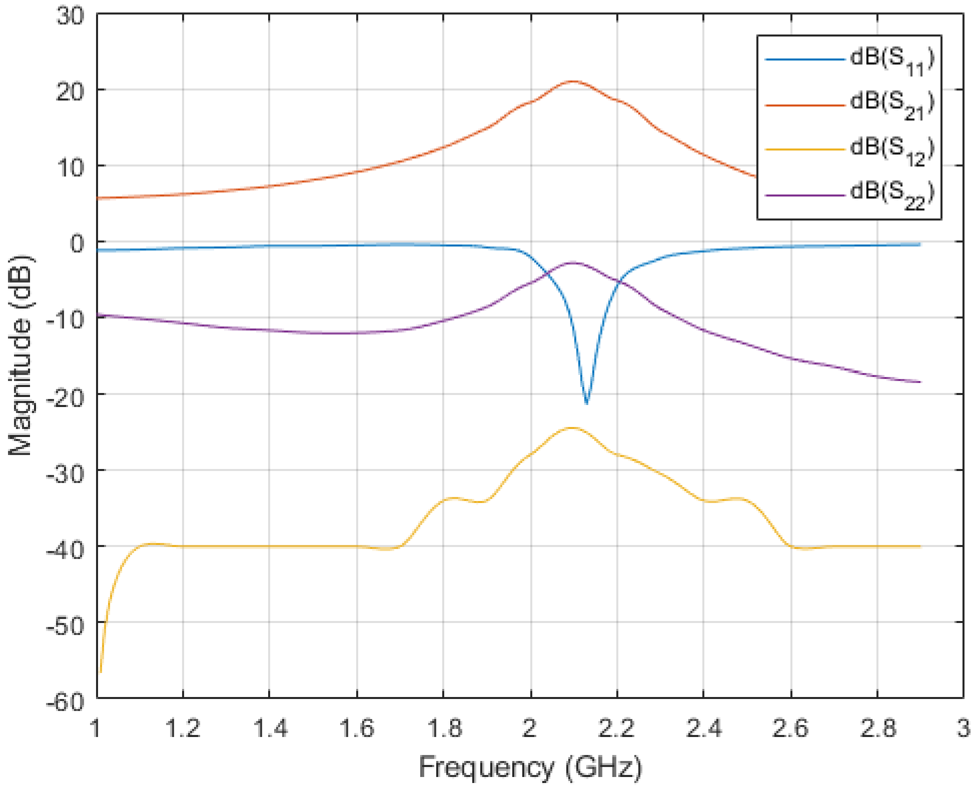
Extract and Plot the S-Parameters of File

Extract S-parameters from file default.s2p and plot it.

```
S = sparameters('default.s2p');
disp(S)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

```
rfplot(S)
```



Calculate the S-Parameters of Circuit Object

Create a resistor element R50 and add it to a circuit object `example2` . Calculate the S-parameters of `example2` .

```
hR1 = resistor(50,'R50');
hckt1 = circuit('example2');
add(hckt1,[1 2],hR1)
setports (hckt1, [1 0],[2 0])
freq = linspace (1e3,2e3,100);
S = sparameters(hckt1,freq,100);
disp(S)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 100

  rfparam(obj,i,j) returns S-parameter Sij
```

Convert RF Data Object to S-parameters

```
file = 'default.s2p';
h = read(rfdata.data, file);
S = sparameters(h)
```

```
S = 
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]
      Impedance: 50.0000 + 0.0000i

  rfparam(obj,i,j) returns S-parameter Sij
```

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

```
 sparameters (h, 70e6)
```

```
ans = 
  sparameters: S-parameters object

       NumPorts: 1
    Frequencies: 70000000
     Parameters: 0.1867 - 0.0080i
      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
```

Calculate the S-Parameters of a Matching Network Circuit

This example shows how to calculate the S-Parameters for a newly created matching network for the auto-generated circuit #2 with a reference impedance of 100 Ohm.

```
n       = matchingnetwork('LoadImpedance',100,'Components',3);
freq    = linspace(n.CenterFrequency-n.Bandwidth/2,n.CenterFrequency+n.Bandwidth/2);
RefZ0   = 100;
ckt_no  = 2;
s       = sparameters(n,freq,RefZ0,ckt_no)
```

```
s = 
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 100

  rfparam(obj,i,j) returns S-parameter Sij
```

Input Arguments

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*.

circuitobj — Circuit object
circuit object

Circuit object. The function uses this input argument to calculate the S-parameters of the circuit object.

filterobj — RF filter
object handle

RF filter, specified as an `rffilter` object.

netparamobj — Network parameter object
network parameter object

Network parameter object. The network parameter objects are of the type: `sparameters`, `yparameters`, `zparameters`, `abcdparameters`, `gparameters`, `hparameters`, 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');`

`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.

`mnobj` — Matching network
`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

`circuitindices` — Index of matching network
scalar

Index of the matching network circuit, specified as a scalar.

Data Types: `double`

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`.

See Also

`circuit` | `correlation` | `impedance` | `rfparam` | `rfplot` | `smithplot` | `yparameters` | `zparameters`

Introduced in R2012a

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)

See Also

rfinterp1 | rfplot | rfplot | sparameters | sparameters

Introduced before R2006a

rfplot

Plot S-parameter data

Syntax

```
rfplot(s_obj)
rfplot(s_obj,i,j)
rfplot( ___ ,lineSpec)
rfplot( ___ ,plotflag)
hline = rfplot( ___ )
[hline,haxes] = rfplot(filter,frequencies)
```

Description

`rfplot(s_obj)` plots the magnitude in dB versus frequency of all S-parameters (S_{11}, S_{12} ... S_{NN}) on the current axis. `s_obj` must be an s-parameter object.

`rfplot(s_obj,i,j)` plots the magnitude of $S_{i,\,j}$, in decibels, versus frequency on the current axis.

`rfplot( ___ ,lineSpec)` plots S-parameters using optional line types, symbols, and colors specified by `linespec`.

`rfplot( ___ ,plotflag)` allows to specify the type of plot by using the `plotflag`.

`hline = rfplot( ___ )` plots the S-parameters and returns the column vector of handles to the line objects, `hline`.

`[hline,haxes] = rfplot(filter,frequencies)` plots the magnitude response of the S-parameters for the rf filter.

Examples

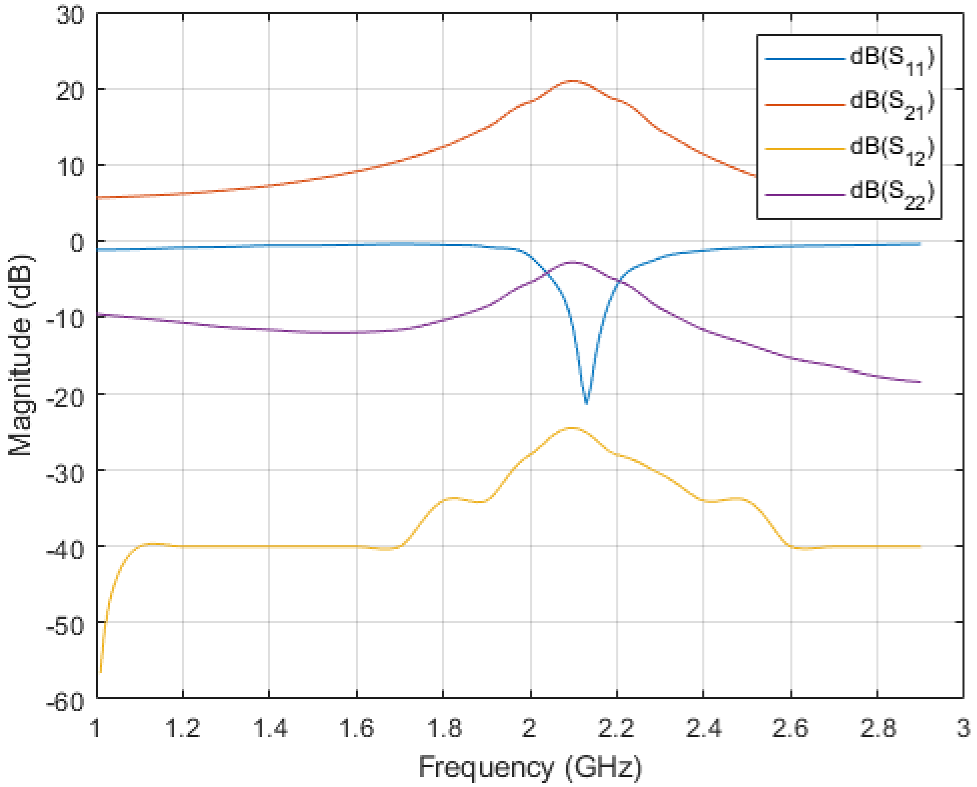
Plot S-Parameter Data Using `rfplot`

Use `sparameters` to create a set S-parameters.

```
hs = sparameters('default.s2p');
```

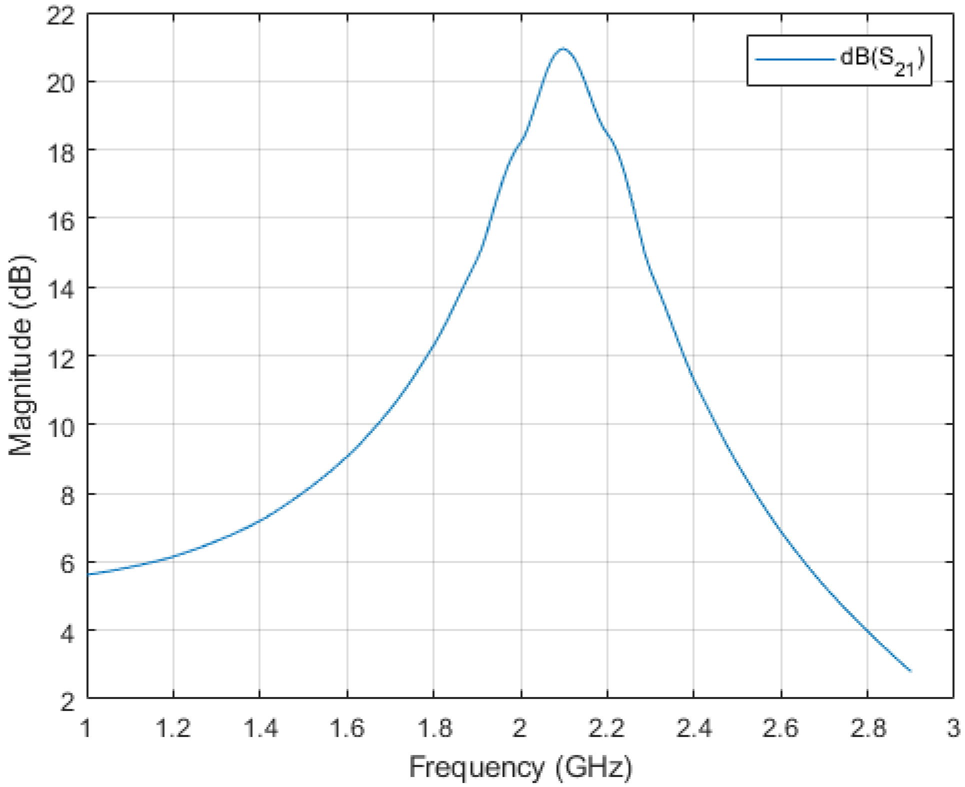
Plot all S-parameters.

```
rfplot(hs)
```



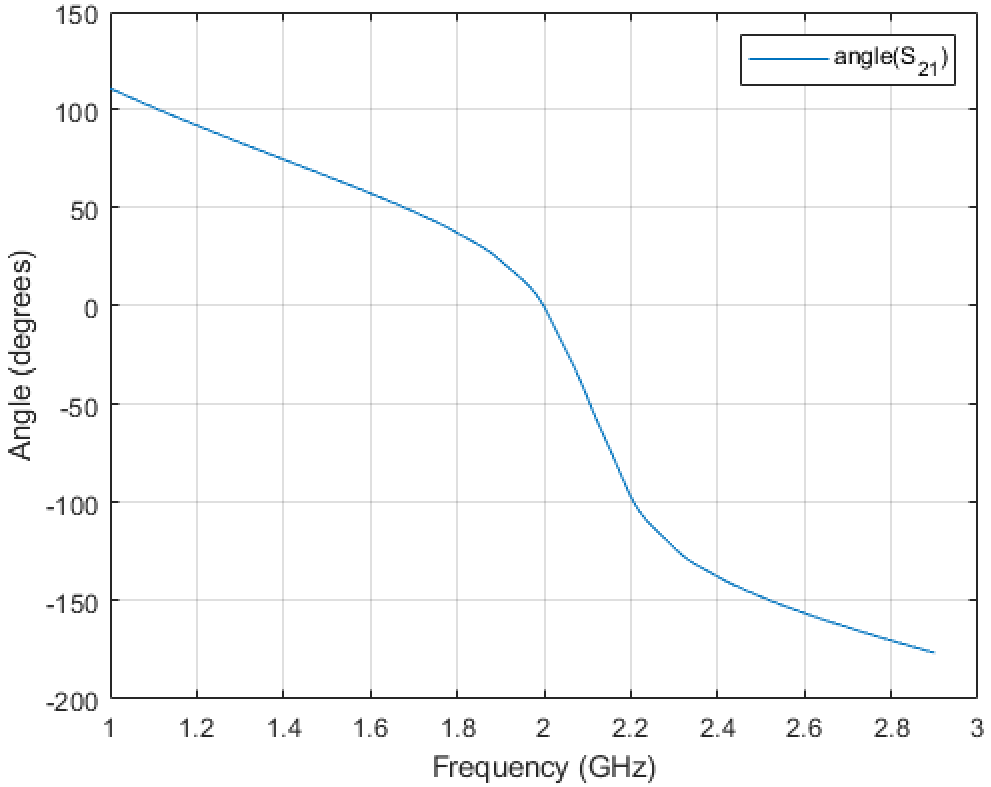
Plot S21.

```
rfplot(hs,2,1)
```



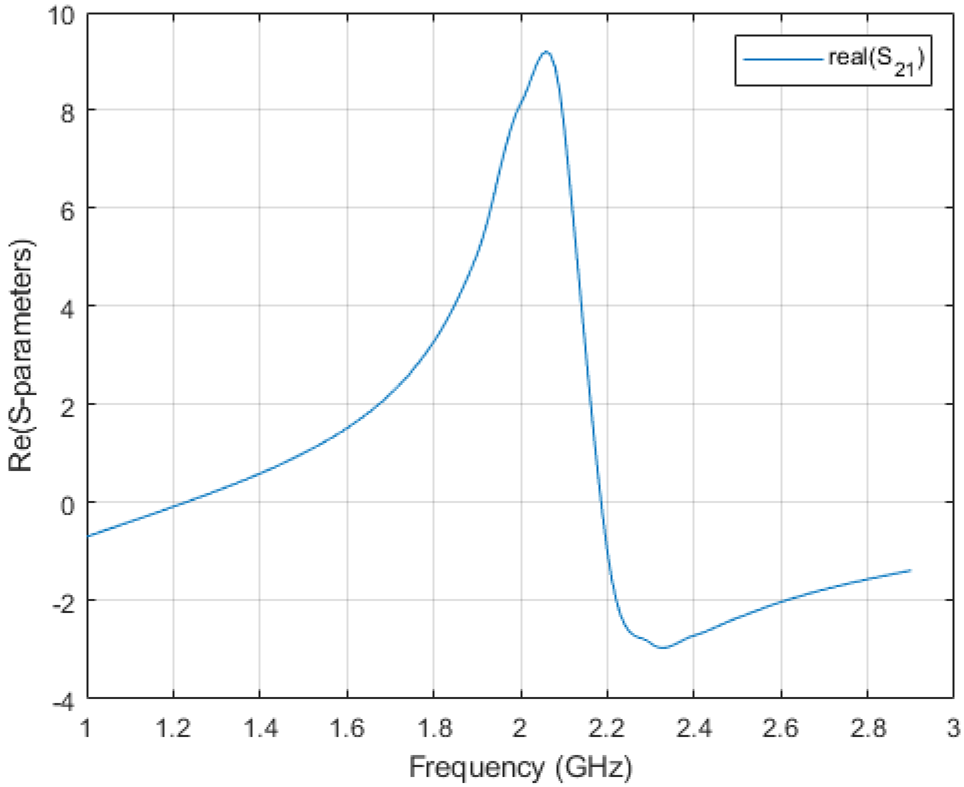
Plot the angle of S21 in degrees.

```
rfplot(hs,2,1,'angle')
```



Plot the real part of S21.

```
rfplot(hs,2,1,'real')
```



Input Arguments

`s_obj` — S-parameters

network parameter object

S-parameters, specified as an RF Toolbox network parameter object. To create this type of object, use the `sparameters` function.

`i` — Row index

positive integer

Row index of data to plot, specified as a positive integer.

`j` — Column index

positive integer

Column index of data to plot, specified as a positive integer.

`lineSpec` — Line specification

character array

Line specification, specified as a text input, that modifies the line types, symbols, and colors of the plot. The function takes text inputs in the same format as `plot` command. For more information on line specification values, see `linespec`.

Example: '-or'

plotflag — Plot types
'db' (default)

Plot types, specified as the following values: 'db', 'real', 'imag', 'abs', 'angle'.

Example: 'angle'

filter — RF filter
rffilter object

RF filter, specified as an rffilter object.

frequencies — Frequencies to plot magnitude response
vector

Frequencies to plot magnitude response, specified as a vector.

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.

See Also

setrfplot | sparameters

Introduced before R2006a

show

Display antenna or array structure; display shape as filled patch

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

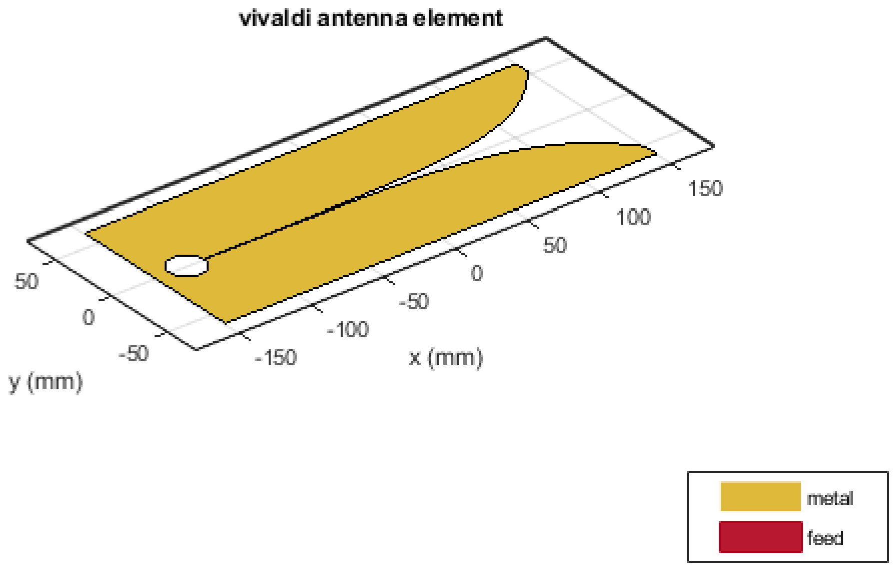
Display Antenna Structure

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

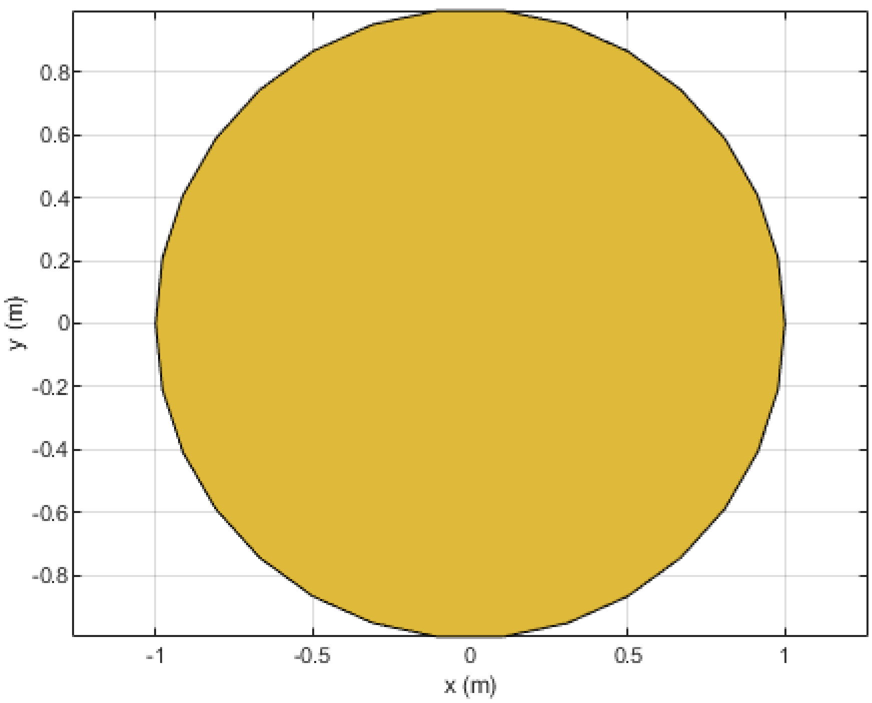
```
show(h)
```



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 handle

Antenna or array object, specified as a scalar handle.

`shape` — Shape created using custom elements and shape objects
object handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle. You can create the shapes using `antenna.Circle`, `antenna.Polygon`, or `antenna.Rectangle`.

Example: `c = antenna.Rectangle; show(c)`

See Also

`layout` | `mesh` | `plot`

Introduced in R2015a

returnLoss

Return loss of antenna; scan return loss of array

Syntax

```
returnLoss(antenna,frequency,z0)
rl = returnLoss(antenna ,frequency, z0)
```

```
returnLoss(array,frequency,elementnumber)
rl = returnLoss(array,frequency,elementnumber)
```

Description

`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,elementnumber)` calculates and plots the scan return loss of a specified antenna element in an array.

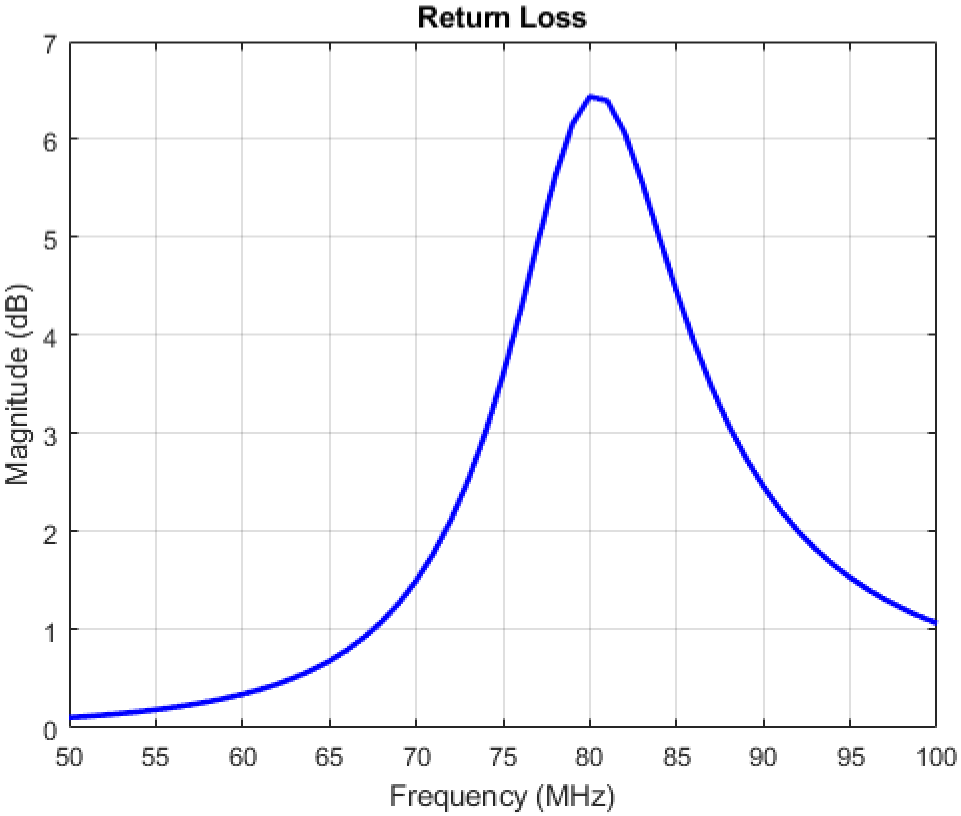
`rl = returnLoss(array,frequency,elementnumber)` returns the scan return loss of a specified antenna element in an array.

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 handle

Antenna object, specified as a scalar handle.

array — array object
scalar handle

Array object, specified as a scalar handle.

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

See Also

`EHfields` | `impedance` | `sparameters`

Introduced in R2015a

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`.

`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.

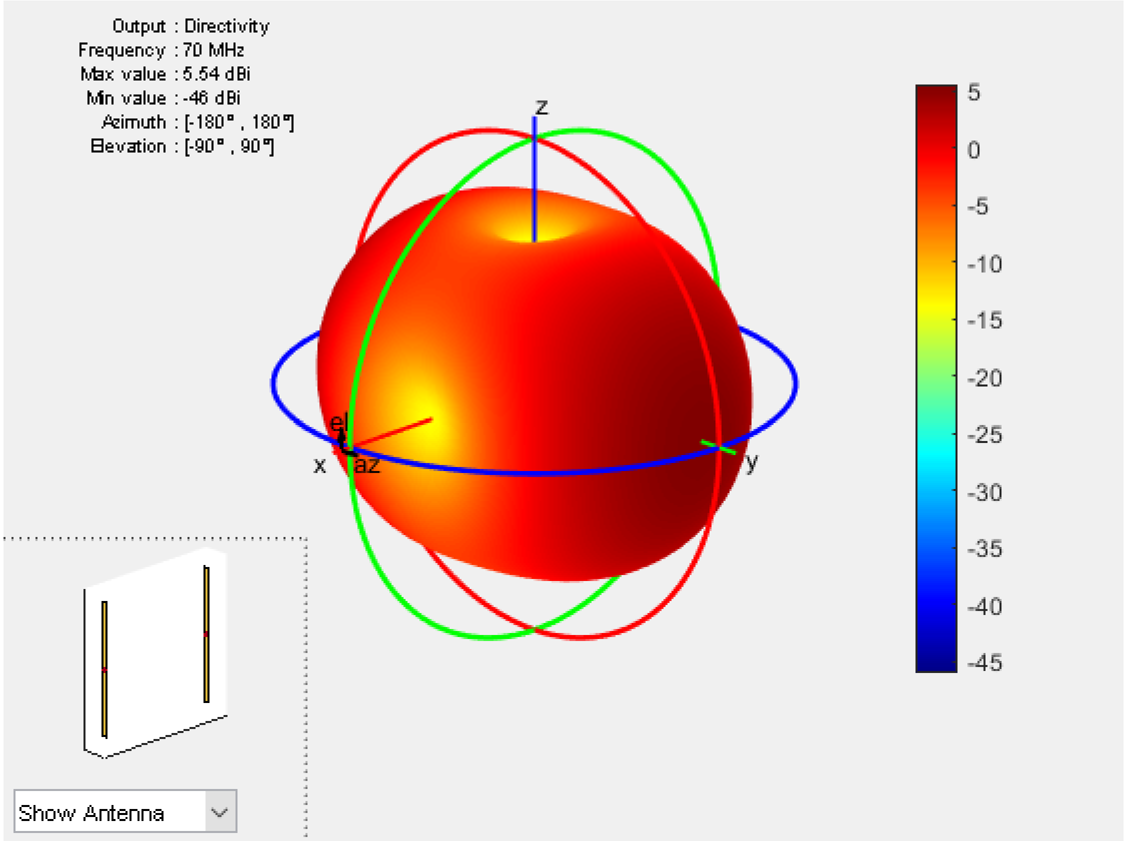
`[pat,azimuth,elevation] = pattern( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Calculate Radiation Pattern of Array

Calculate radiation pattern of default linear array for a frequency of 70 MHZ.

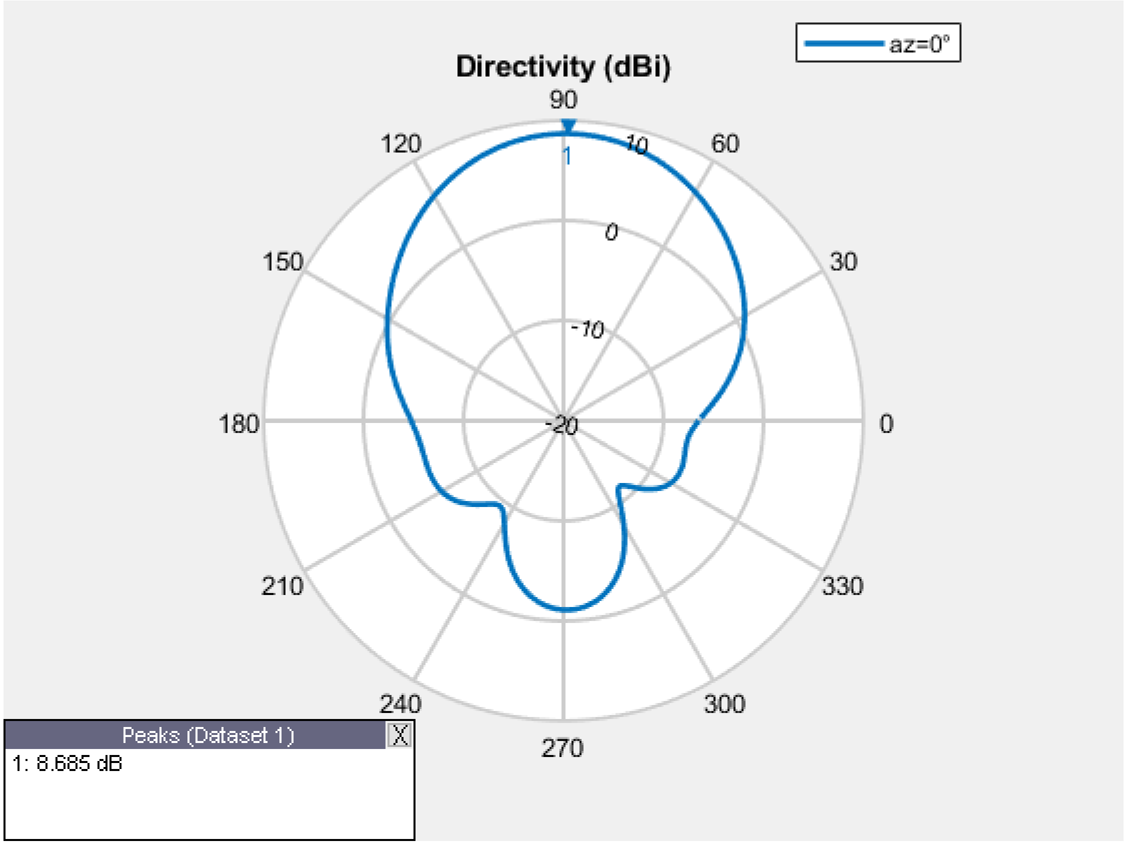
```
l = linearArray;
pattern(l,70e6)
```



Output : Directivity
Frequency : 70 MHz
Max value : 5.54 dBi
Min value : -46 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Radiation Pattern of Helix in X-Z Plane

```
h = helix;
pattern (h, 2e9, 0, 1:1:360);
```



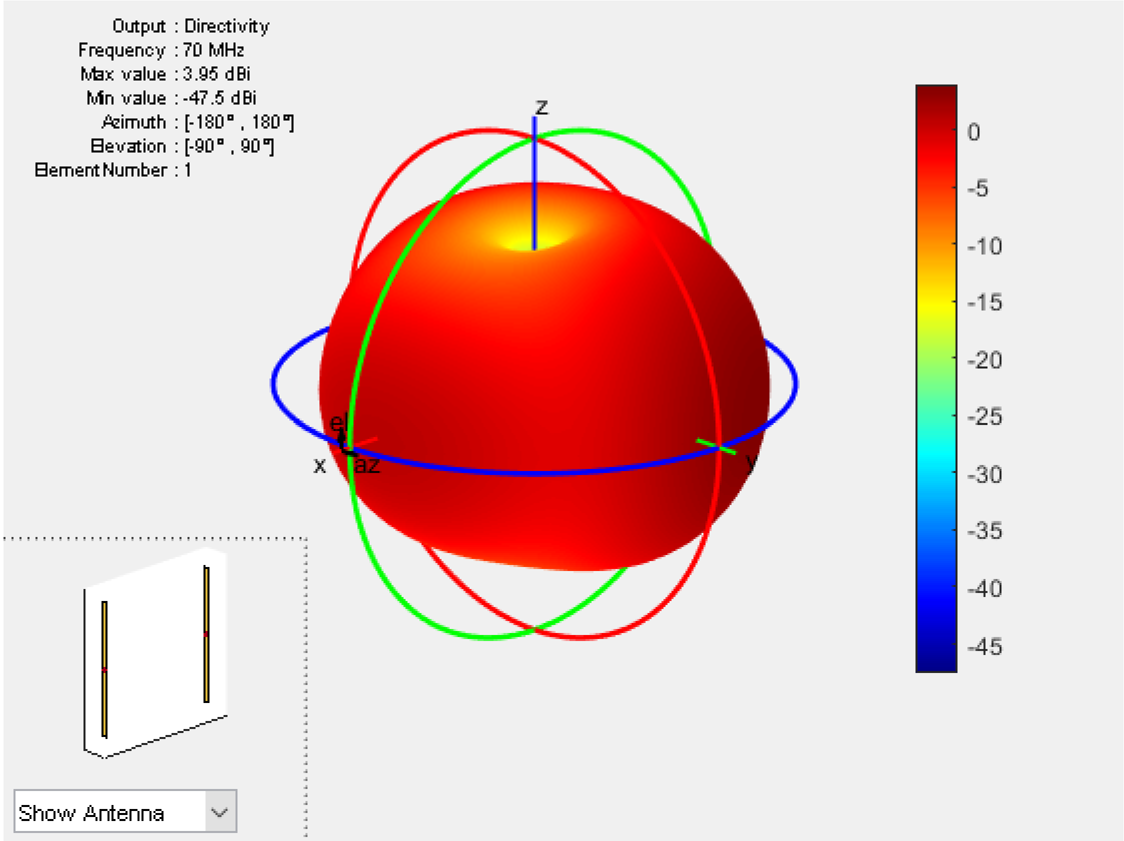
Peaks (Dataset 1)

1: 8.685 dB

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 : -47.5 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

   -6.2750
   -6.0599
   -5.8322
   -5.5935
   -5.3455
```

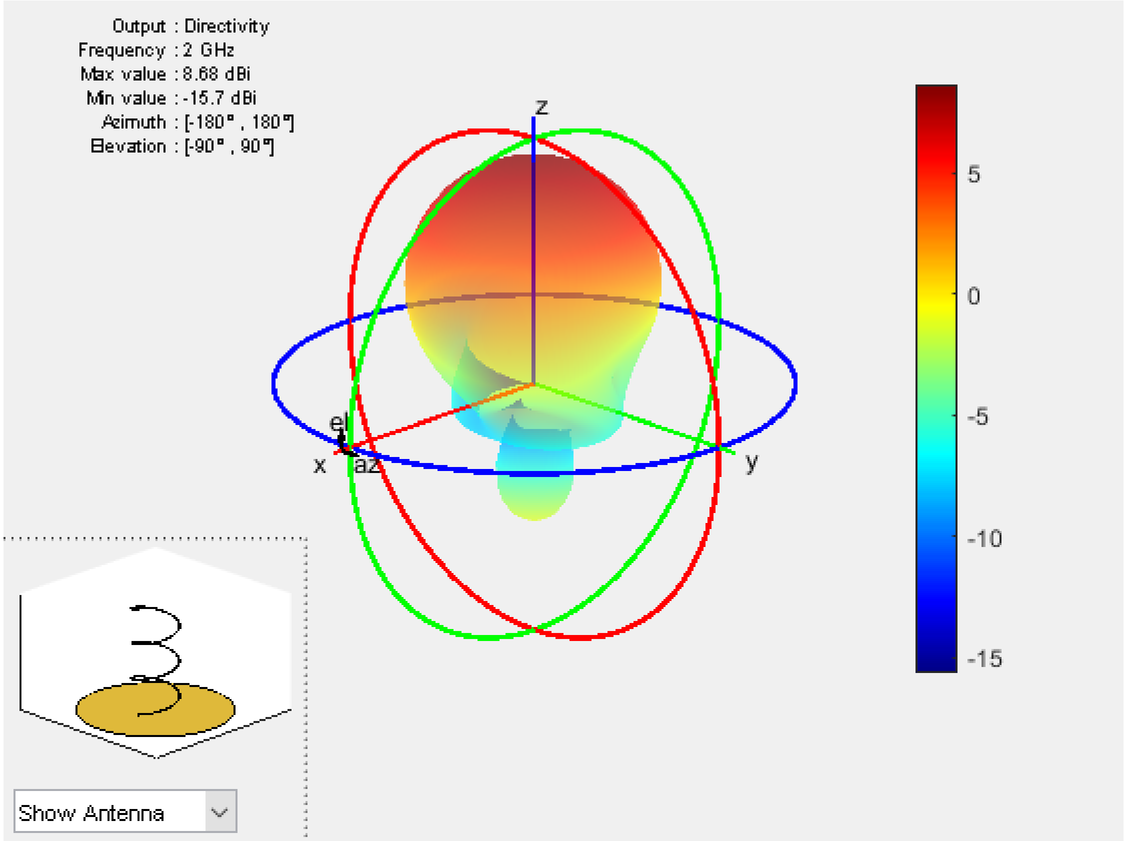
Radiation Pattern of Helix Antenna

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]
```

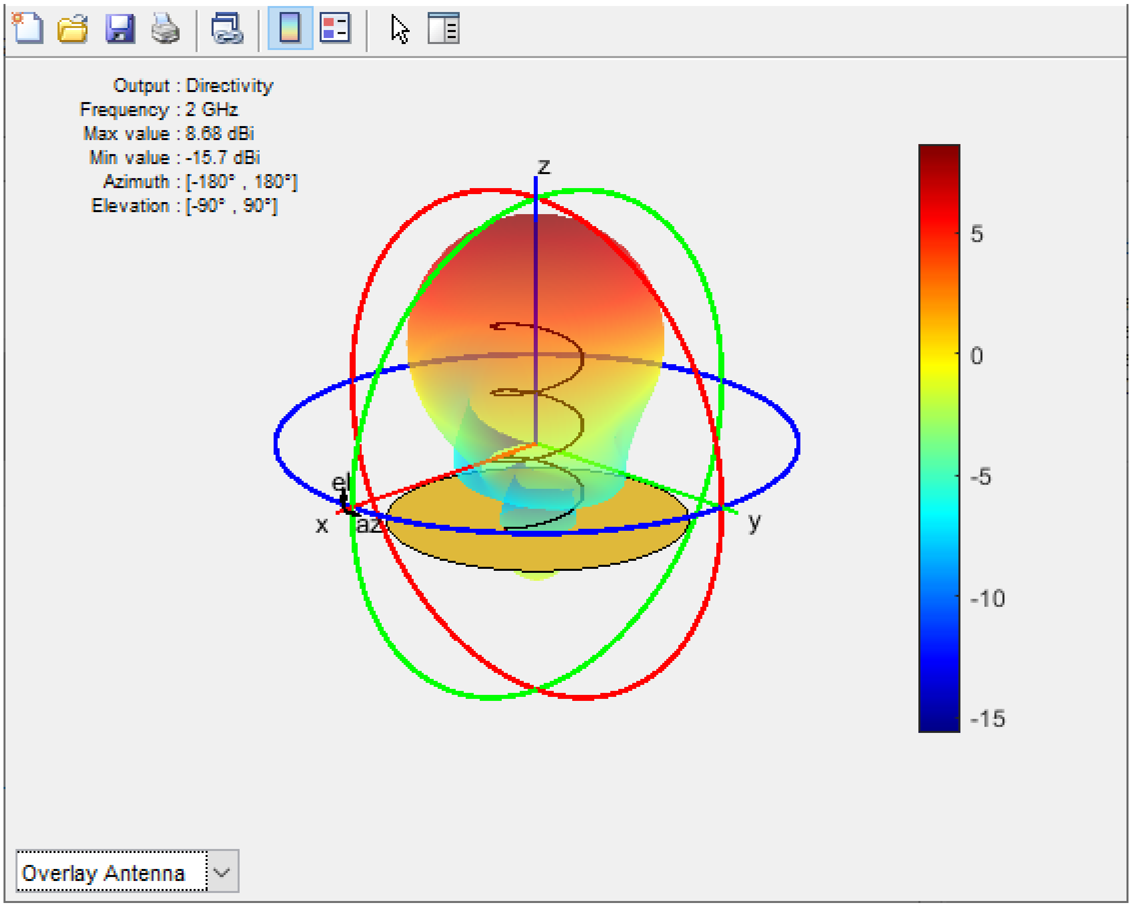
```
p.Transparency = 0.5;
ant = helix;
pattern(ant,2e9,'patternOptions',p)
```



```
       Output : Directivity
    Frequency : 2 GHz
    Max value : 8.68 dBi
    Min value : -15.7 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

`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.

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'` | `'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 (lossless antenna or array)
- `gain` – Gain in dBi (lossy antenna or array)
- `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.

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.

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`
- `AntennaVisibility`
- `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 f 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)
- `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.

More About

Directivity

Directivity is the ability of an antenna to radiate power in a particular direction. It can be defined as ratio of maximum radiation intensity in the desired direction to the average radiation intensity in all other directions. The equation for directivity is:

$$D = \frac{4\pi U(\theta, \phi)}{P_{rad}}$$

where:

- D is the directivity of the antenna
- U is the radiation intensity of the antenna
- P_{rad} is the average radiated power of antenna in all other directions

Antenna directivity is dimensionless and is calculated in decibels compared to the isotropic radiator (dBi).

Gain

The *gain* of an antenna depends on the directivity and efficiency of the antenna. It can be defined as the ratio of maximum radiation intensity in the desired direction to the total power input of the antenna. The equation for gain of an antenna is:

$$G = \frac{4\pi U(\theta, \phi)}{P_{in}}$$

where:

- G is the gain of the antenna
- U is the radiation intensity of the antenna
- P_{in} is the total power input to the antenna

If the efficiency of the antenna in the desired direction is `100%`, then the total power input to the antenna is equal to the total power radiated by the antenna, that is, $P_{in} = P_{rad}$. In this case, the antenna directivity is equal to the antenna gain.

Array Factor and Pattern Multiplication

The basis of the array theory is the *pattern multiplication* theorem. This theorem states that the combined pattern of N identical array elements is expressed as the element pattern times the array factor.

The array factor is calculated using the formula:

$$AF = \sum_{i=0}^{N} V(i) \cdot e^{(k\sin\theta\cos\varphi \cdot x(i) + k\sin\varphi \cdot y(i) + k\cos\theta \cdot z(i))}$$

where:

- *N* is the number of elements in the array.
- *V* is the applied voltage (amplitude and phase) at each element in the array.
- *k* is the wave number.
- theta and phi are the elevation and azimuth angles.
- x, y, and z are the Cartesian coordinates of the feed locations for every antenna element of the array.

Once the array factor is calculated using the above equation, you can calculate the beam pattern of the array as the product of the array factor and the beam pattern of the individual antenna element of the array.

Array pattern = AF individual antenna element pattern*

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` | `current` | `patternFromSlices`

Topics

"Radiation Pattern"

Introduced in R2015a

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.

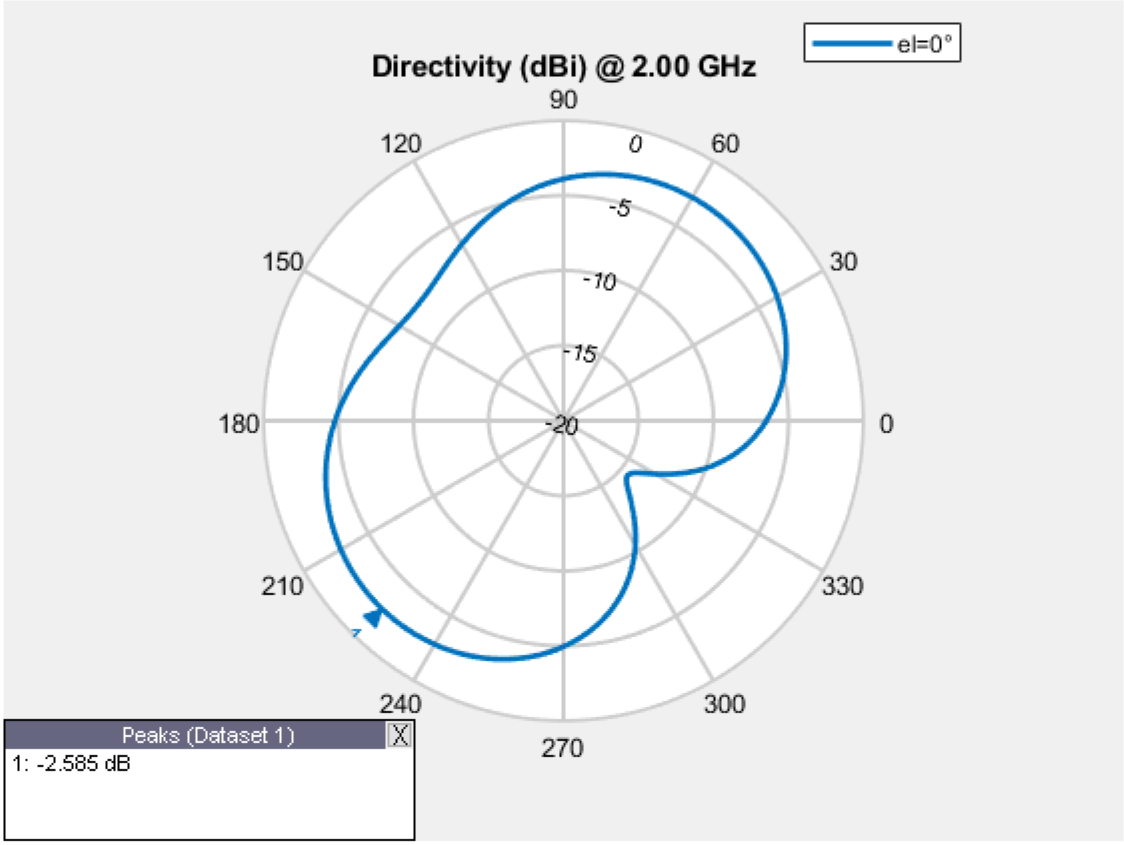
`directivity = patternAzimuth(object,frequency,elevation,'Azimuth')` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Azimuth Radiation Pattern of Helix Antenna

Calculate and plot the azimuth radiation pattern of the helix antenna at 2 GHz.

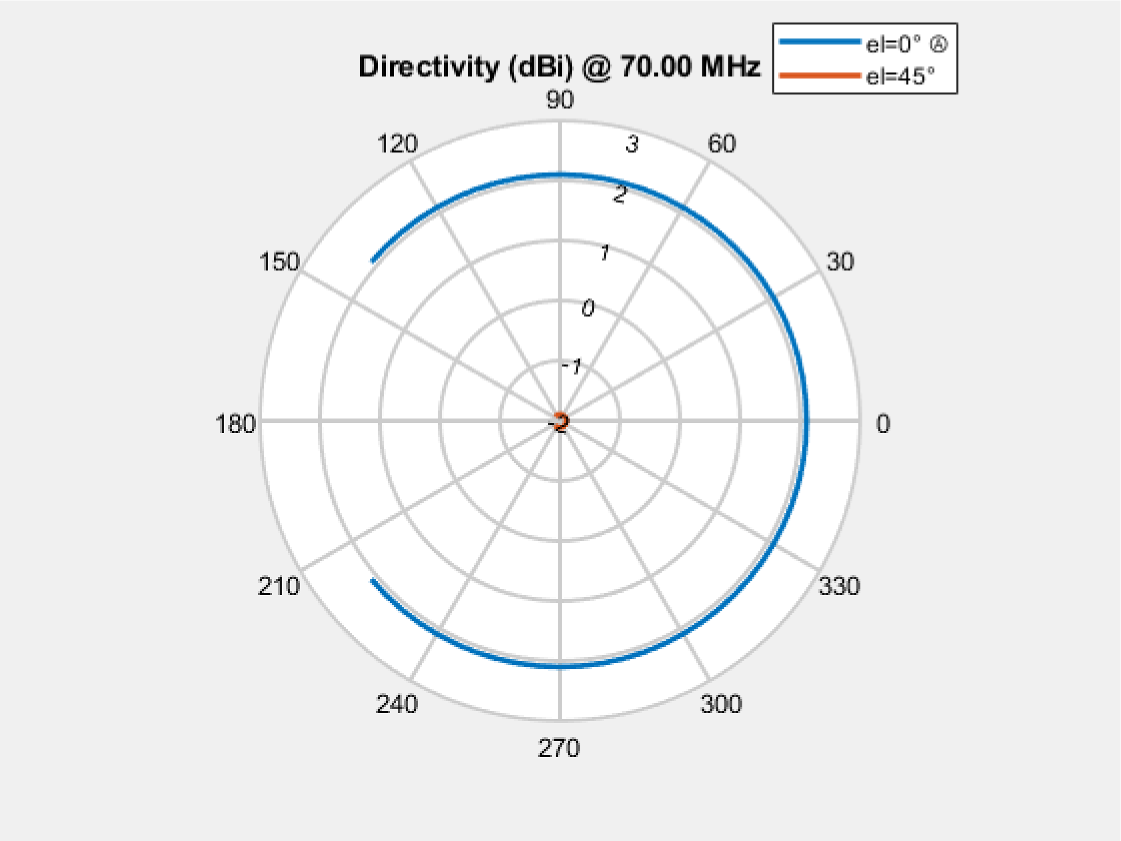
```
h = helix;
patternAzimuth(h,2e9);
```



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 handle

Antenna or array object, specified as a scalar handle.

`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.

See Also

`pattern` | `patternElevation` | `polarpattern`

Introduced in R2015a

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.

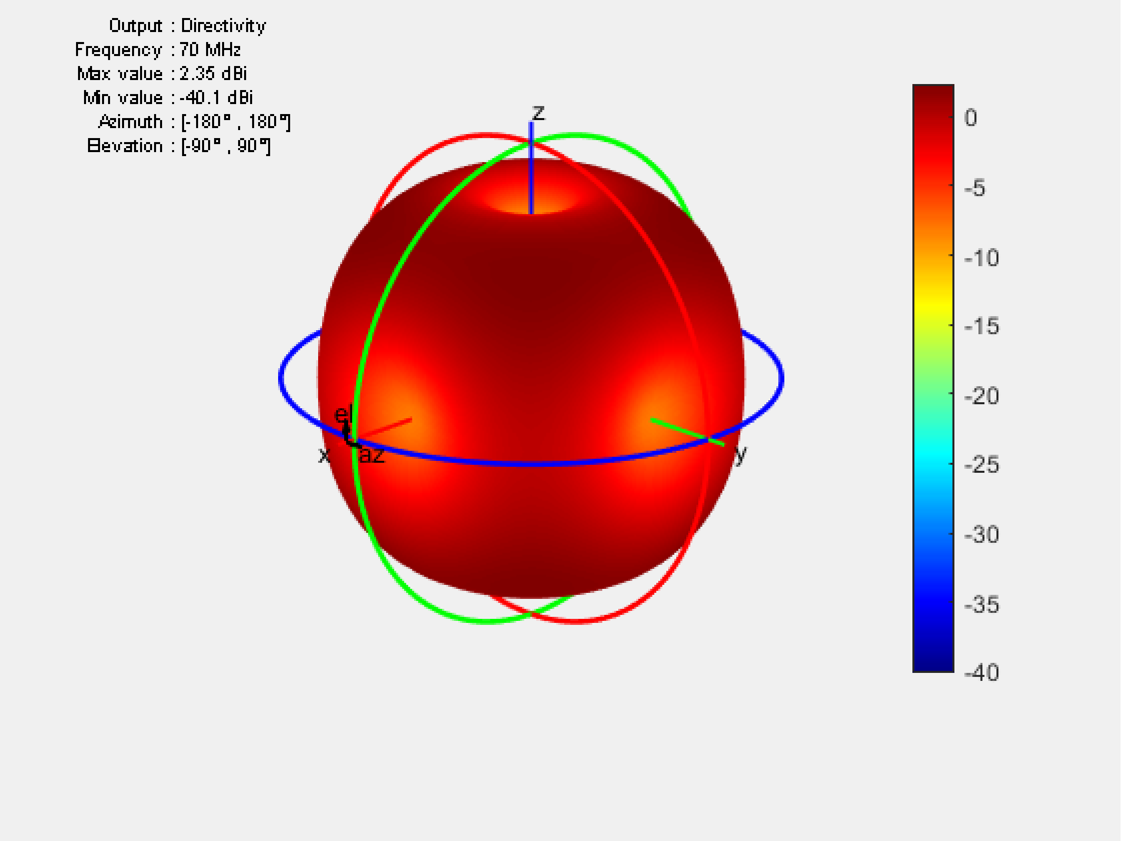
`[fieldval,azimuth,elevation] = patternMultiply( ___ ,Name,Value)` returns the field value using additional options specified by one or more `Name,Value` pair arguments. Specify name-value pair arguments after all other input arguments.

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.

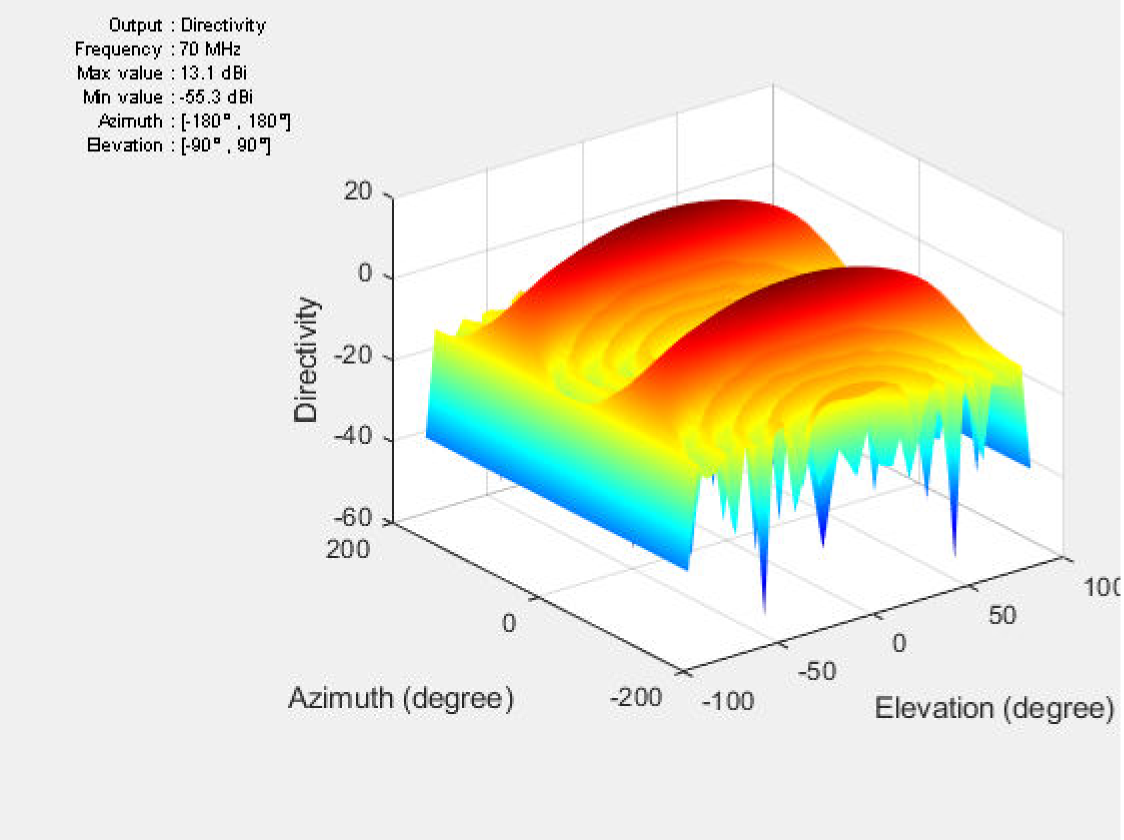
```
h = rectangularArray;
patternMultiply(h,70e6);
```



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');
```



Input Arguments

`array` — Input antenna array

object handle

Array object, specified as an object handle.

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

'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`

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.

See Also

`pattern` | `patternElevation`

Topics
“Antenna Toolbox Coordinate System”

Introduced in R2017a

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.

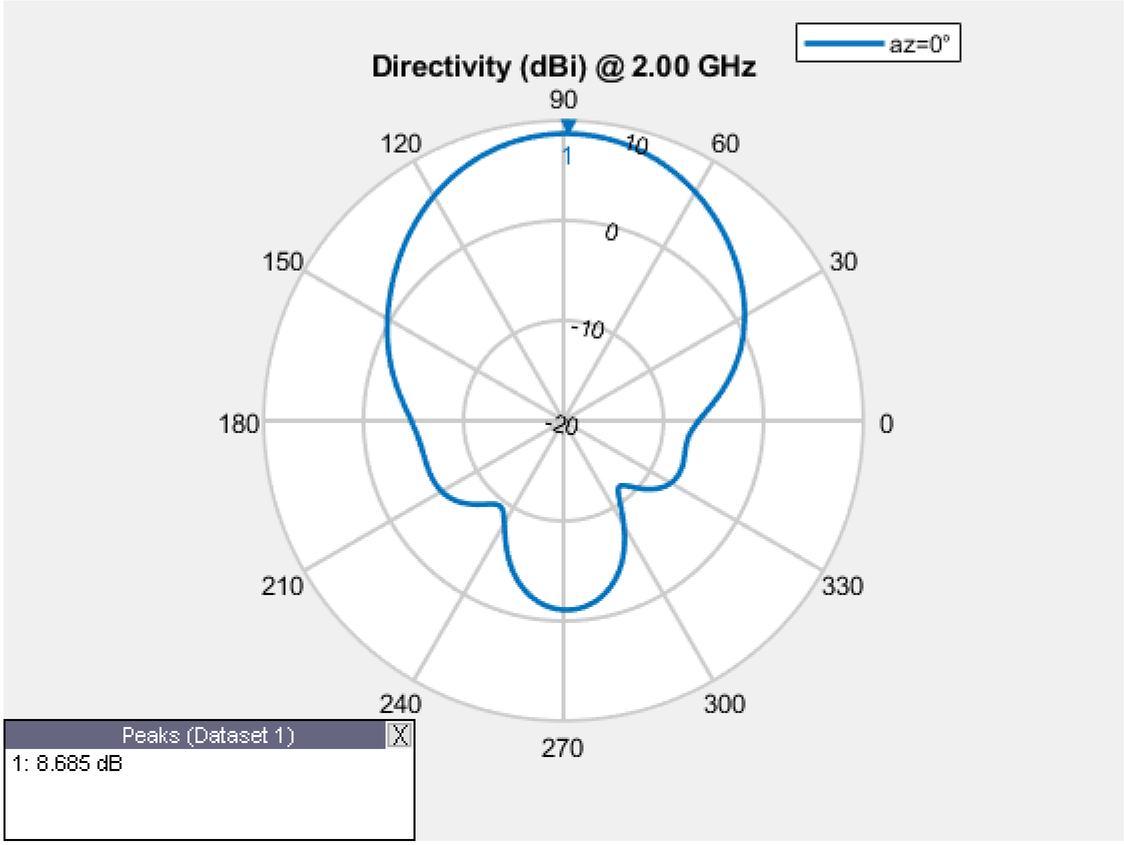
`directivity = patternElevation(object,frequency,azimuth,'Elevation')` uses additional options specified by one or more `Name, Value` pair arguments.

Examples

Elevation Radiation Pattern of Helix

Calculate and plot the elevation pattern of the helix antenna at 2 GHz.

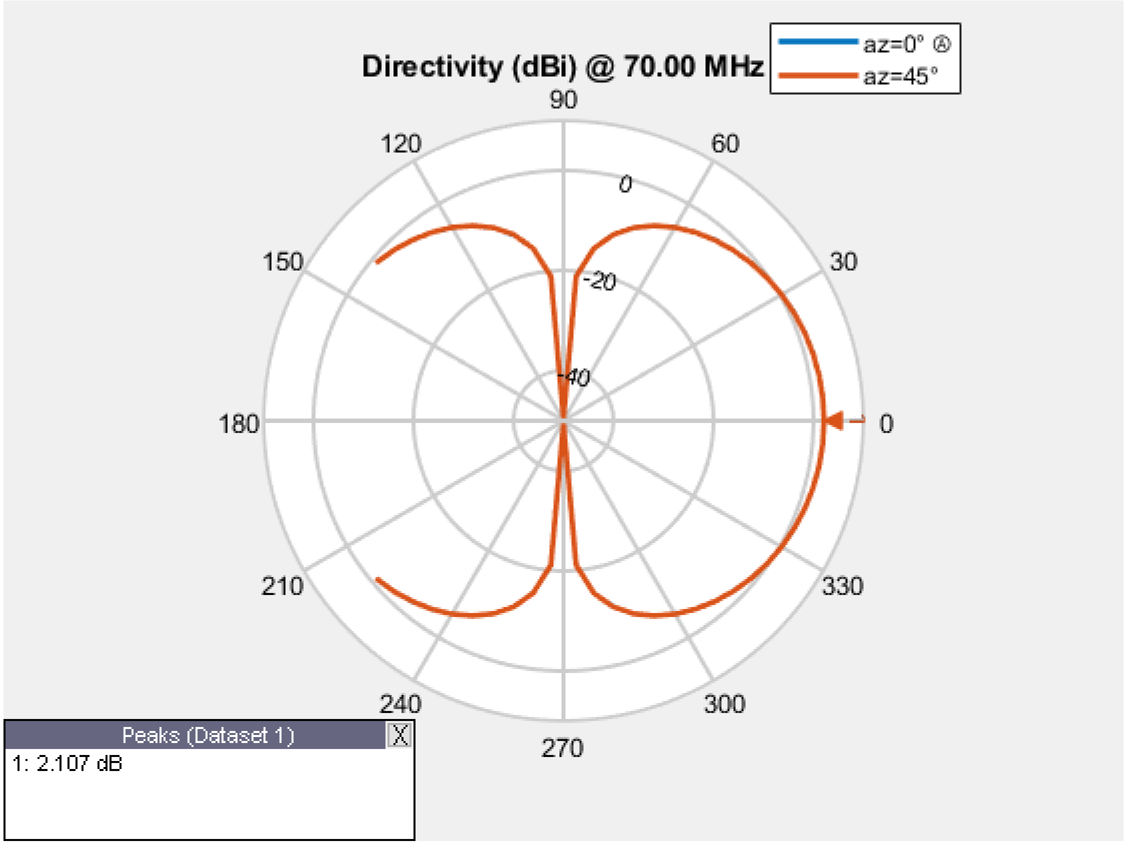
```
h = helix;
patternElevation (h, 2e9);
```



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 handle

Antenna or array object, specified as a scalar handle.

`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.

See Also

`pattern` | `patternAzimuth` | `polarpattern`

Introduced in R2015a

current

Current distribution on metal or dielectric antenna or array surface

Syntax

```
current(object,frequency)

i = current(object,frequency)
[i,p] = charge(object,frequency)

current(object,frequency,'dielectric')
i = current(object,frequency,'dielectric')
i = current( ___ ,Name,Value)
```

Description

`current(object,frequency)` calculates and plots the absolute value of the current on the surface of an antenna or array object, at a specified frequency.

`i = current(object,frequency)` returns the *x*, *y*, *z* components of the current on the surface of an antenna or array object, at a specified frequency.

`[i,p] = charge(object,frequency)` returns the *x*, *y*, *z* components of the current on the surface of an antenna or array object, at a specified frequency and at the point in which the current calculation is performed.

`current(object,frequency,'dielectric')` calculates and plots the absolute value of current at a specified frequency value on the dielectric face of the antenna or array.

`i = current(object,frequency,'dielectric')` returns the *x*, *y*, *z* components of the current on the dielectric surface of an antenna or array object, at a specified frequency.

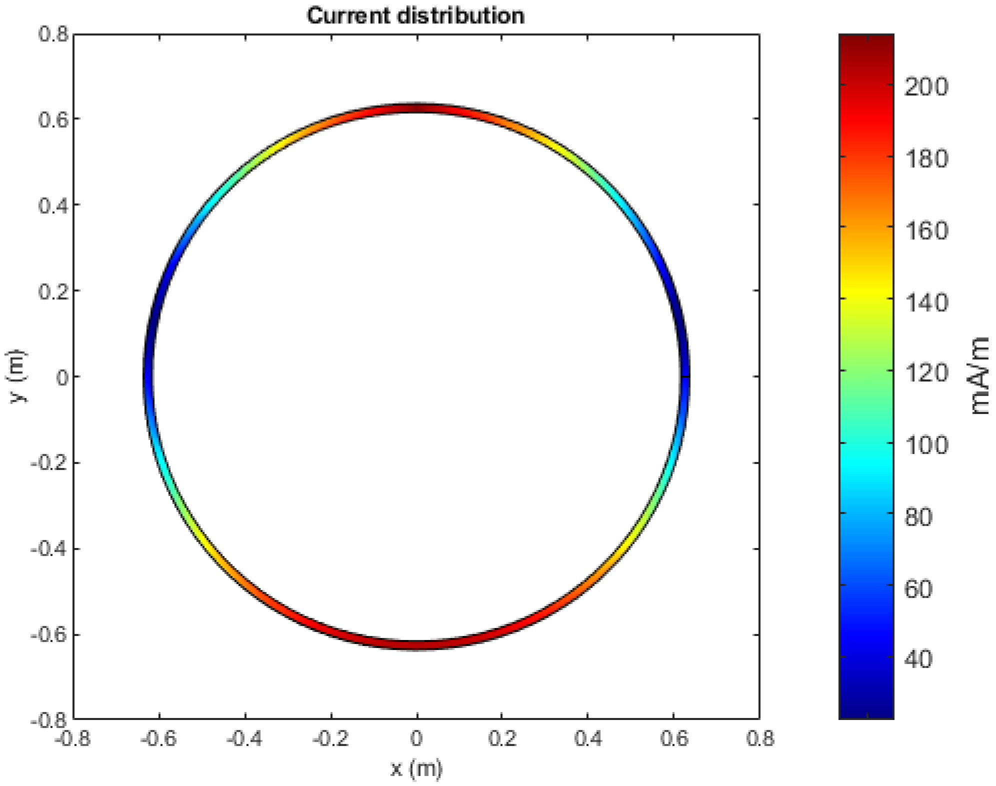
`i = current( ___ ,Name,Value)` calculates the current on the surface of an antenna using additional name-value pairs.

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
   0.0009 + 0.0020i   0.0013 + 0.0025i  -0.0002 - 0.0012i   0.0003 + 0.0013i   0.0004 + 0.0015i
   0.0562 + 0.1041i   0.0428 + 0.0763i   0.0659 + 0.1334i   0.0649 + 0.1280i   0.0641 + 0.1250i
```

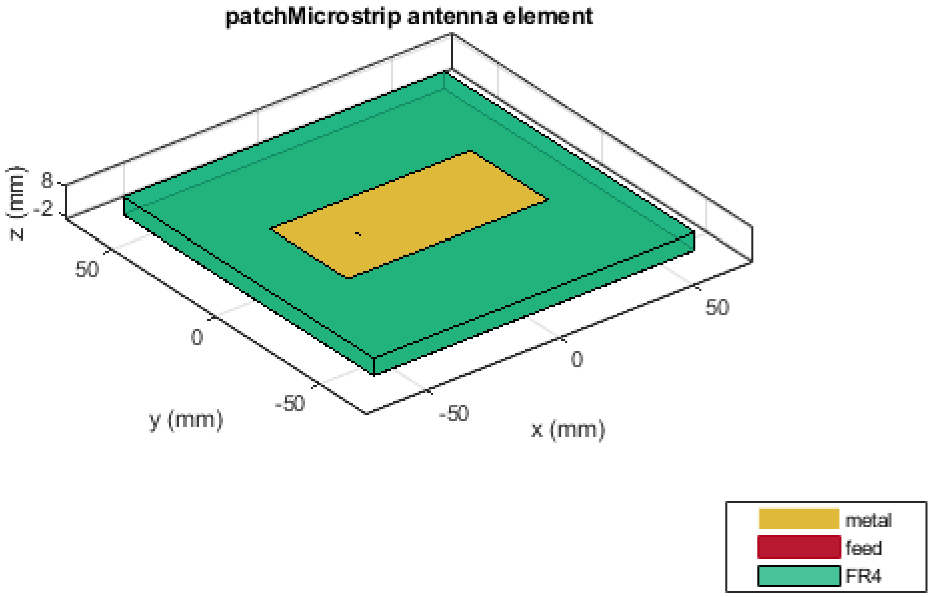
Current Distribution On Microstrip Patch Antenna

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)
```

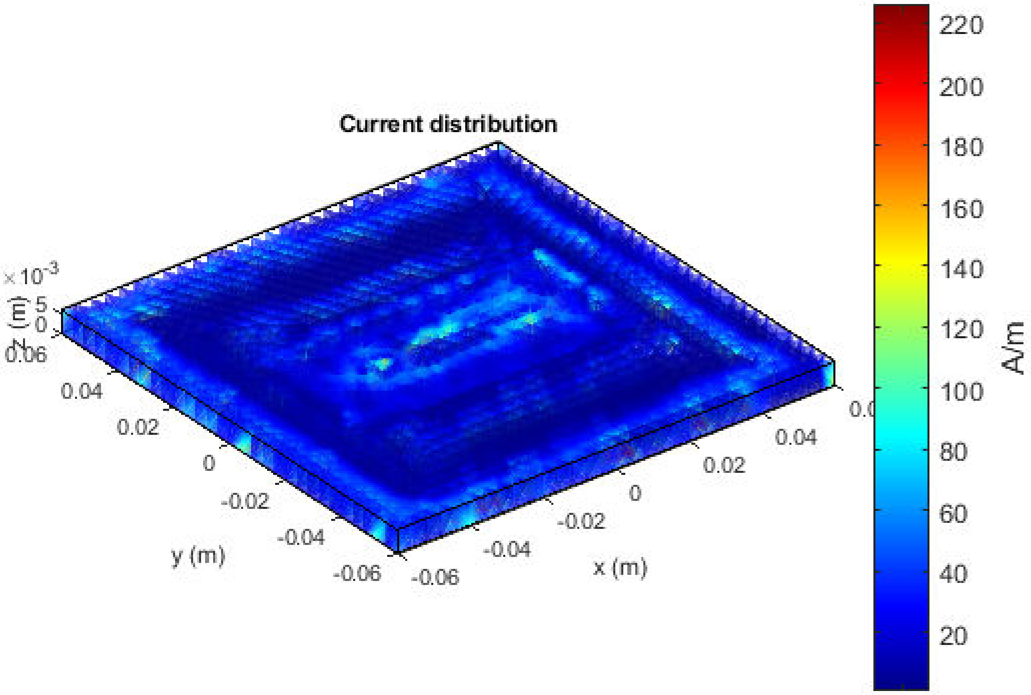
```
pm = 
  patchMicrostrip with properties:

                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]
                  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')
```



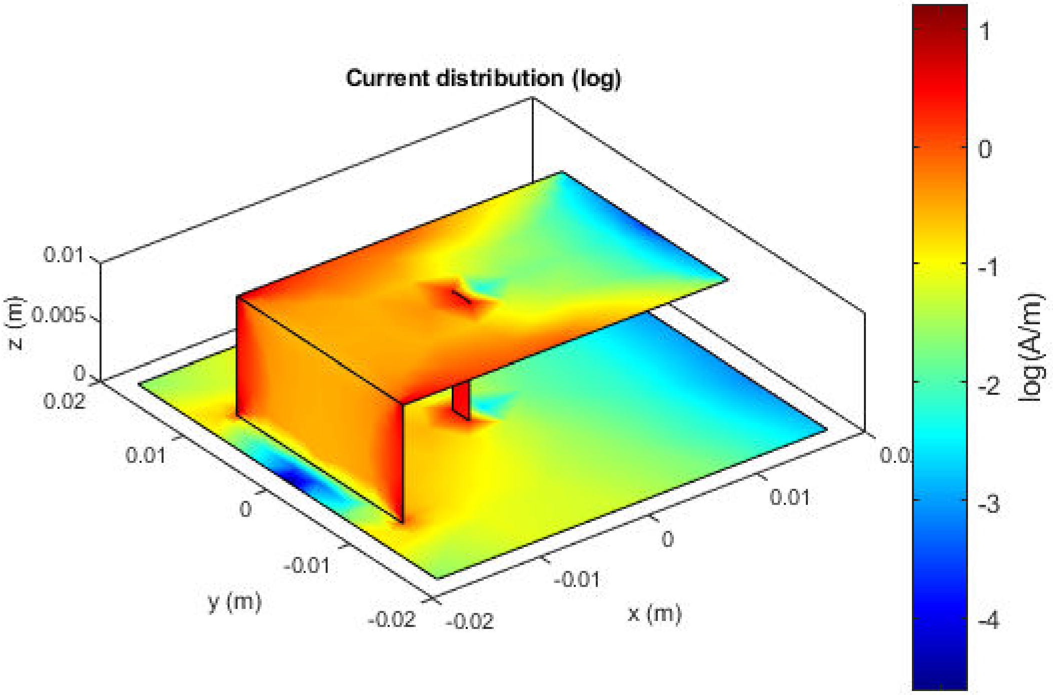
Logarithmic Current Distribution on Antenna Surface

Create a default pifa (planar inverted F antenna).

```
ant = pifa;
```

Visualize the current distribution on the pifa antenna in using log function scale.

```
current(ant,1.75e9,'scale','log')
```



Input Arguments

`object` — Antenna or array object

scalar handle

Antenna or array object, specified as a scalar handle.

`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 Pair Arguments

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

Example: `'scale','log10'`

`scale` — Scale to visualize current distribution

`string` (default) | function handle

Scale to visualize the current distribution on the surface of the antenna, specified as a string or a function handle. The string values are: `'linear'`, `'log'`, `'log10'`. By default, the value is `'linear'`. The function handle can be of any mathematical function such as `log`, `log10`, `cos`, or `sin`.

Data Types: `char` | `function_handle`

Output Arguments

`i` — *x*, *y*, *z* components of current distribution
3-by-*n* complex matrix in A/m

x, *y*, *z* components of 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.

See Also

`axialRatio` | `charge`

Introduced in R2015a

charge

Charge distribution on metal or dielectric 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.

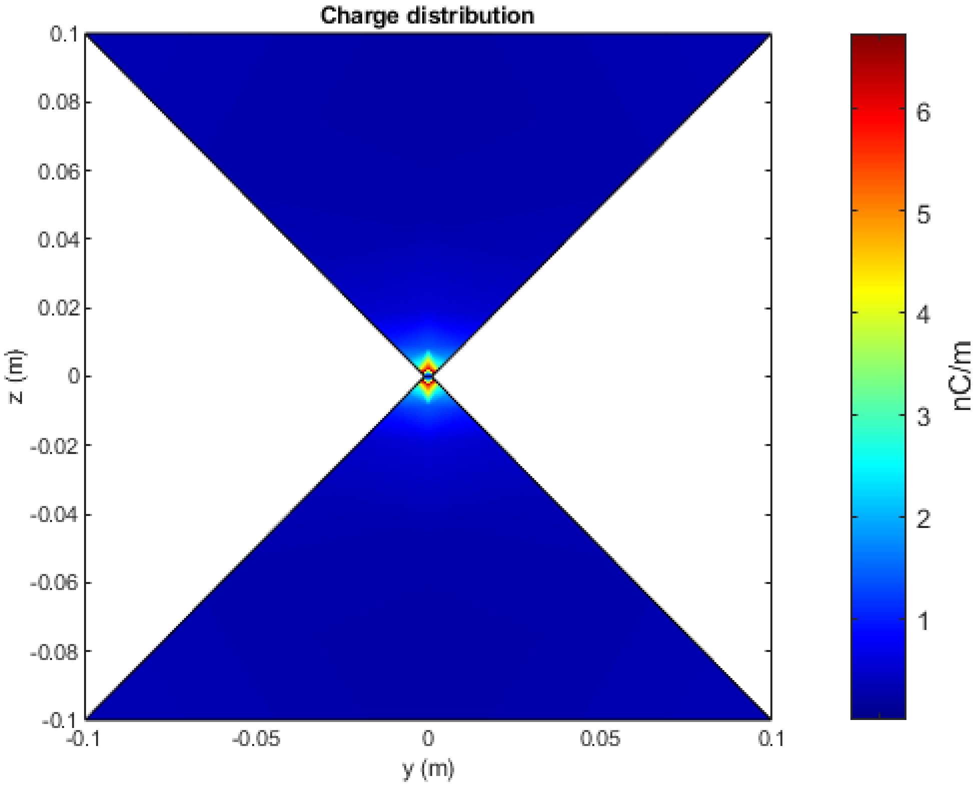
`c = charge( ___ ,Name,Value)` calculates the charge on the surface of an antenna using additional name-value pairs.

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

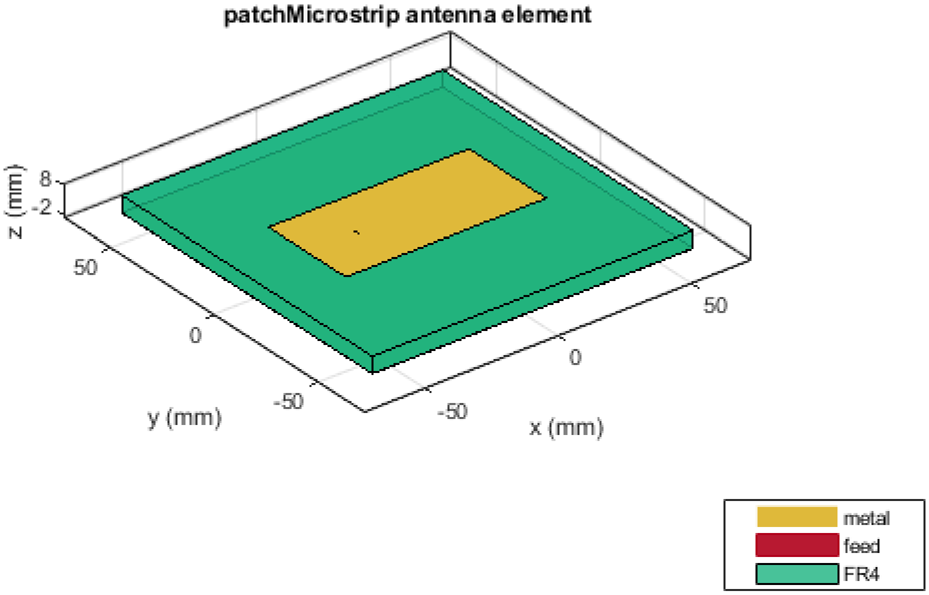
Create a microstrip patch antenna using **'FR4'** as the dielectric substrate.

```
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
```

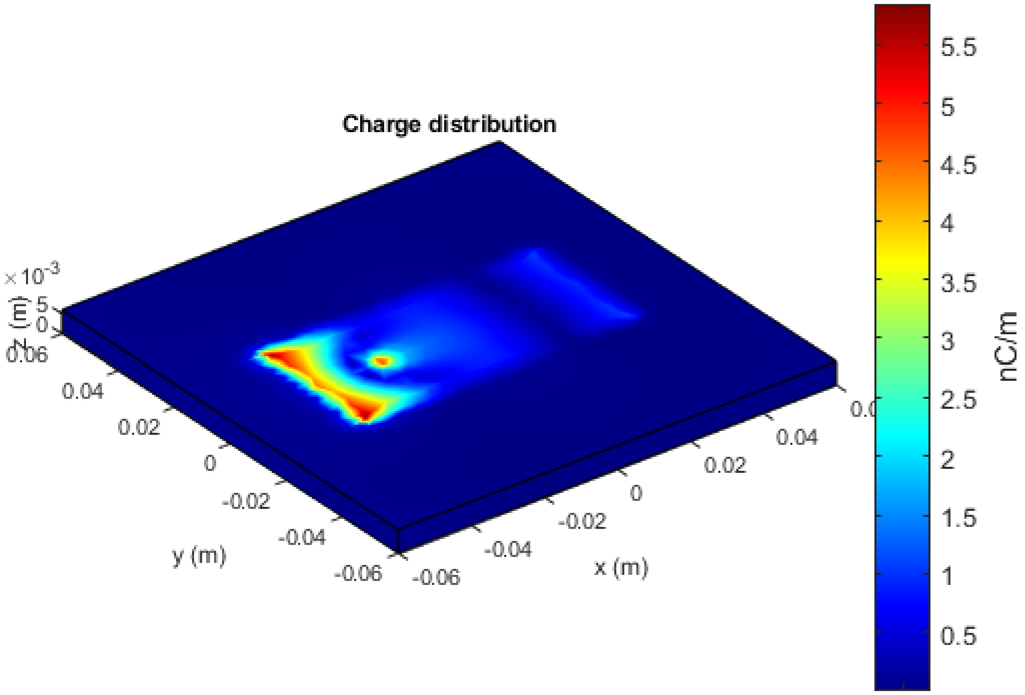
```
              Height: 0.0060
           Substrate: [1x1 dielectric]
    GroundPlaneLength: 0.1200
     GroundPlaneWidth: 0.1200
    PatchCenterOffset: [0 0]
           FeedOffset: [-0.0187 0]
                 Tilt: 0
             TiltAxis: [1 0 0]
                 Load: [1x1 lumpedElement]
```

```
show(pm)
```



Plot the charge distribution on the antenna at a frequency of 1.67 GHz.

```
figure
charge(pm,1.67e9,'dielectric')
```



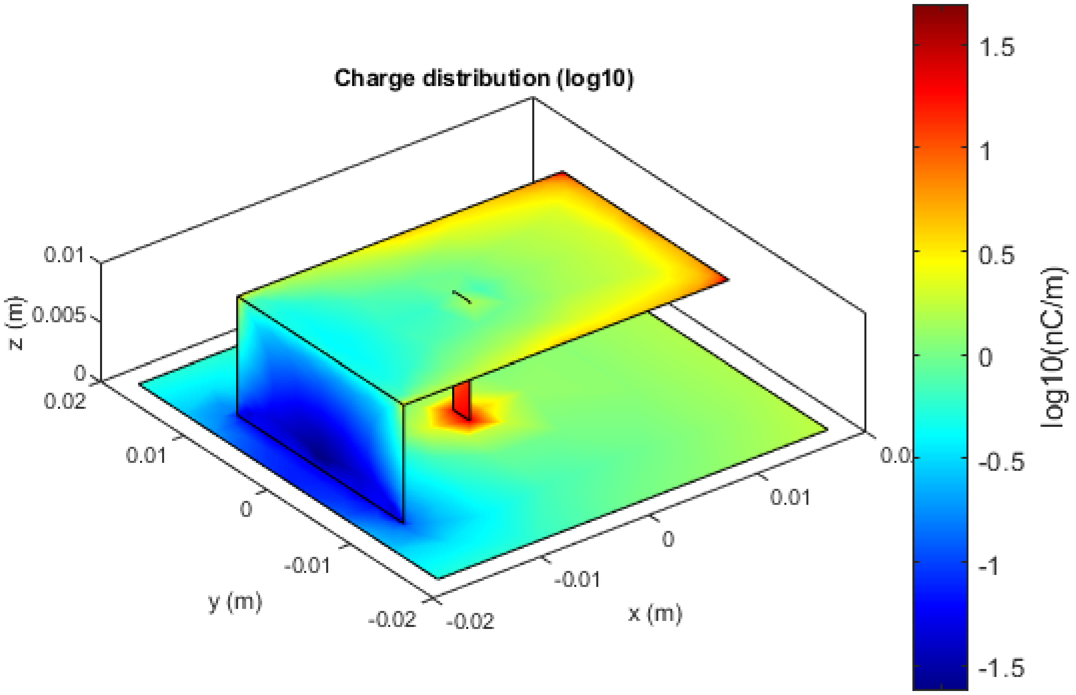
Logarithmic Charge Distribution on Antenna Surface

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')
```



Input Arguments

`object` — Antenna or array object

scalar handle

Antenna or array object, specified as a scalar handle.

`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 Pair Arguments

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

Example: `'scale','log10'`

`scale` — Scale to visualize charge distribution

`string` (default) | function handle

Scale to visualize the charge distribution on the surface of the antenna, specified as a string or a function handle. The string values are: `'linear'`, `'log'`, `'log10'`. By default, the value is `'linear'`. The function handle can be of any mathematical function such as `log`, `log10`, `cos`, or `sin`.

Data Types: `char` | `function_handle`

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.

See Also

`EHfields` | `current`

Introduced in R2015a

design

Design prototype antenna or arrays for resonance at 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)
```

Description

`hant = design(antenna,frequency)` designs any antenna object from the antenna library to resonate at the specified frequency.

`harray = design(array,frequency)` designs an array of dipoles for operation at a specified `frequency`. The elements are separated by half-wavelength.

`harray = design(array,frequency,elements)` designs an array of elements for operation at a specified `frequency`. The elements are separated by half-wavelength, if possible. If you cannot achieve half-wavelength spacing, the element size is used to calculate inter-element separation and the elements are evenly distributed on a sphere 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 elements are placed in the locations specified by default `conformalArray` object. If the required element positions cannot be achieved due to intersection of elements, the element size is used to compute the inter element spacing and the elements are evenly distributed on a sphere of radius proportional to the largest element in the property `Elements`.

`harray = design(conformalarray,frequency,elements)` designs a conformal array of 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 specified elements at the specified frequency.

Examples

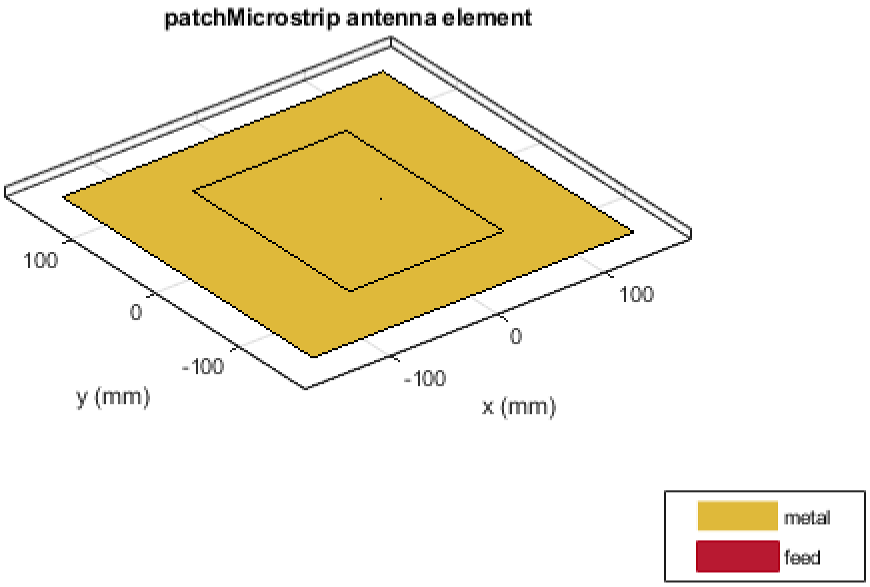
Prototype Antenna Design

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

```
show(p)
```



Calculate the impedance of the above antenna at the same frequency.

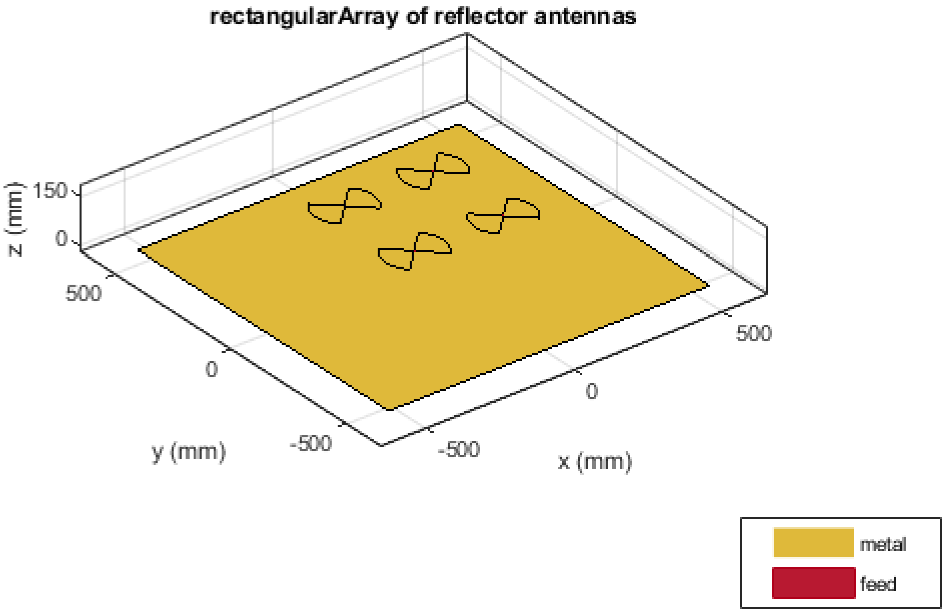
```
Z = impedance(p,1e9)
```

```
Z = 55.8475 - 0.8183i
```

Rectangular Array of Reflector Backed Rounded Bowtie Antennas

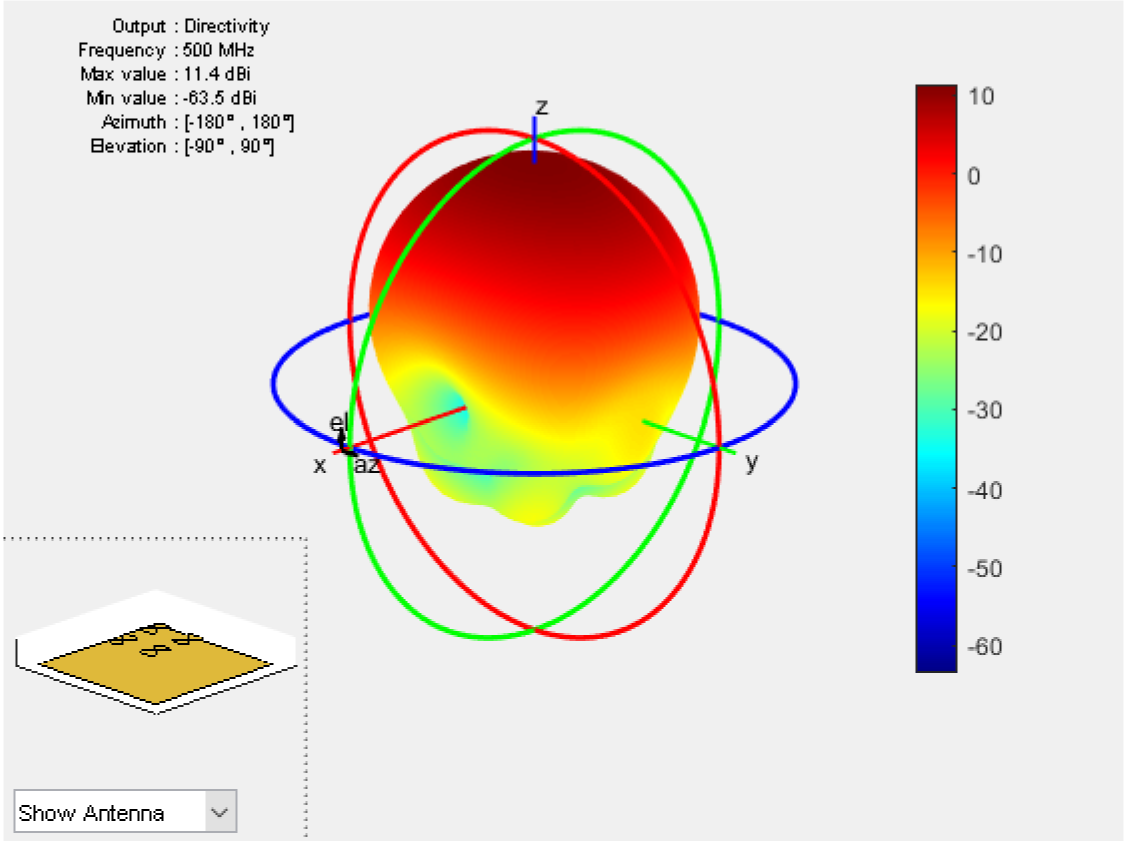
Design a rectangular array of reflector backed rounded bowtie antennas to operate at 500 MHz.

```
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)
```



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})
```

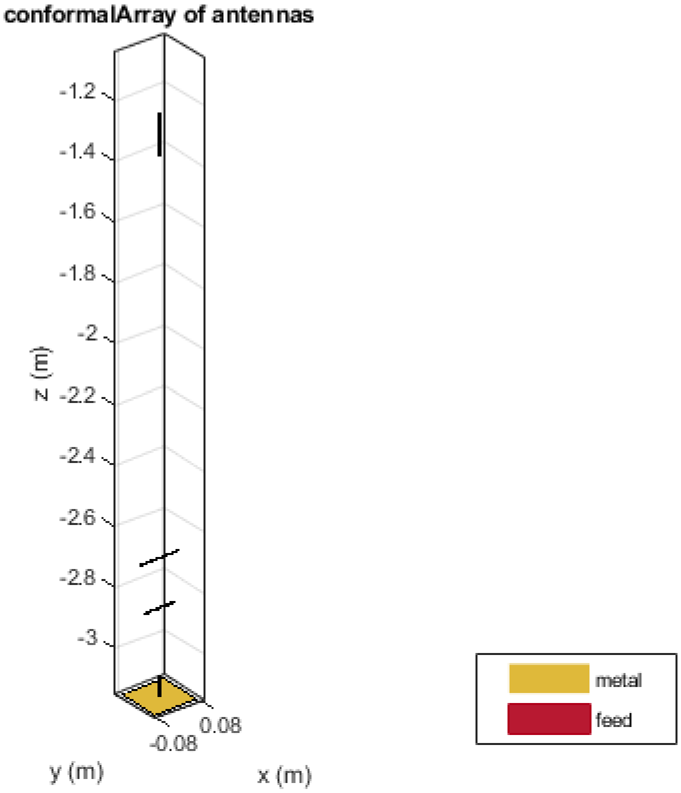
```
desC = 
  conformalArray with properties:
```

```
            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)
```



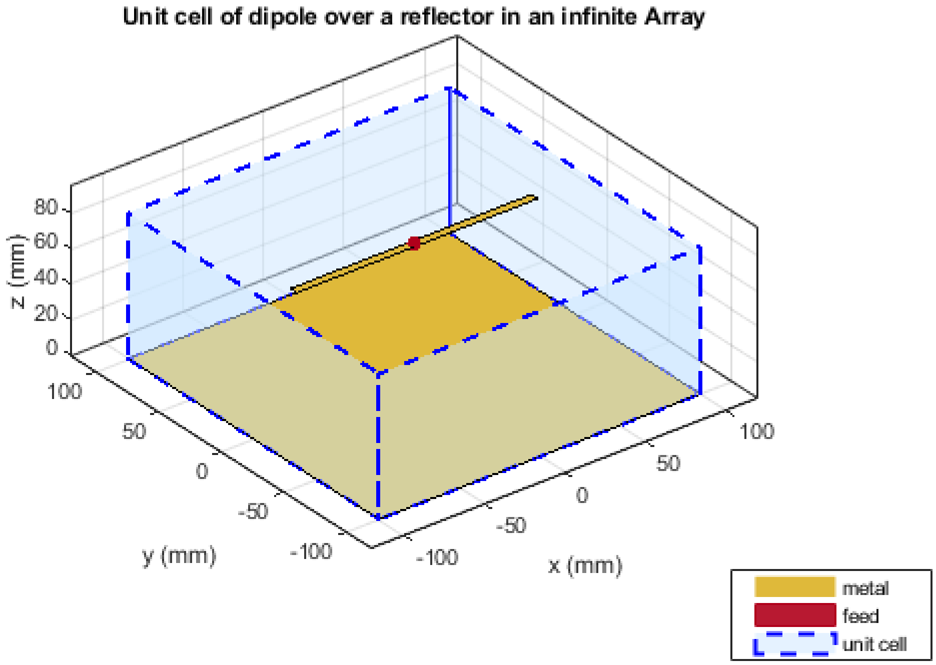
Design Infinite Array Using Specified Frequency and Antenna

Create an infinite array.

```
infarrayV1 = infiniteArray
```

```
infarrayV1 = 
  infiniteArray with properties:

          Element: [1x1 reflector]
      ScanAzimuth: 0
    ScanElevation: 90
```

```
show(infarrayV1)
```

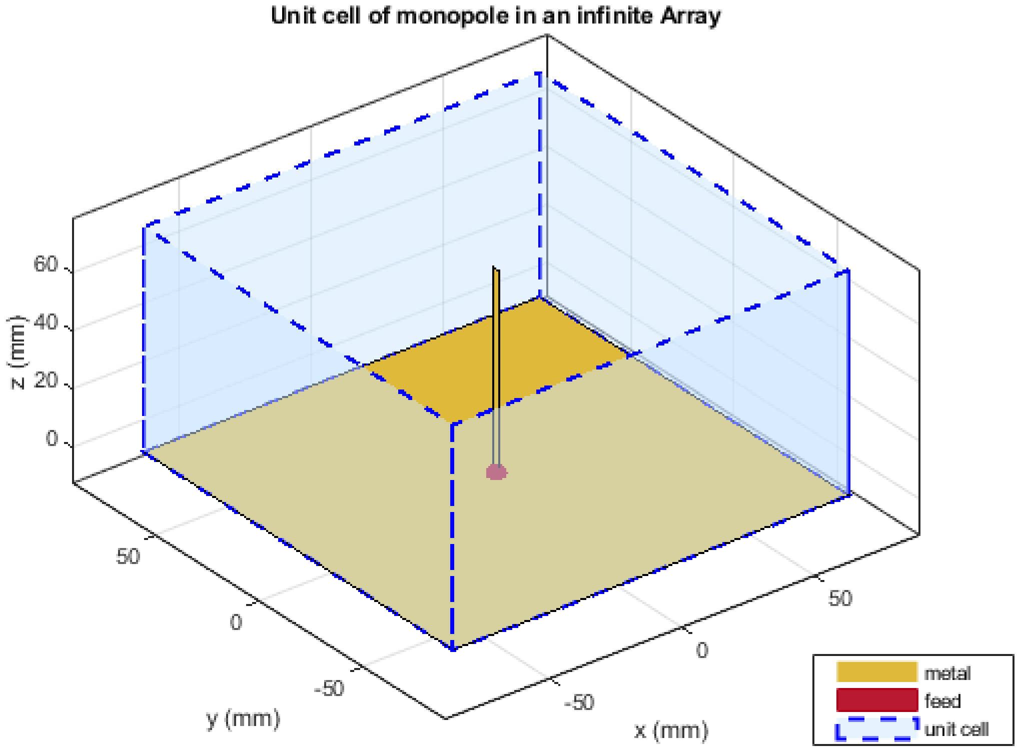


Design the above array using a monopole antenna and at 1 GHz frequency.

```
infarrayV2 = design(infarrayV1,1e9,monopole)
```

```
infarrayV2 = 
  infiniteArray with properties:

          Element: [1x1 monopole]
      ScanAzimuth: 0
    ScanElevation: 90
```

```
show(infarrayV2)
```



Input Arguments

antenna — Antenna object

scalar handle

Antenna object from antenna library, specified as a scalar handle.

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`

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 handle

Infinite array object, specified as a `infiniteArray` object.

Example: `i = infiniteArray;ia = design(1,1e9,monopole);` Designs an infinite array with a monopole antenna element operating at a frequency of 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.

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.

See Also

`show`

Introduced in R2016b

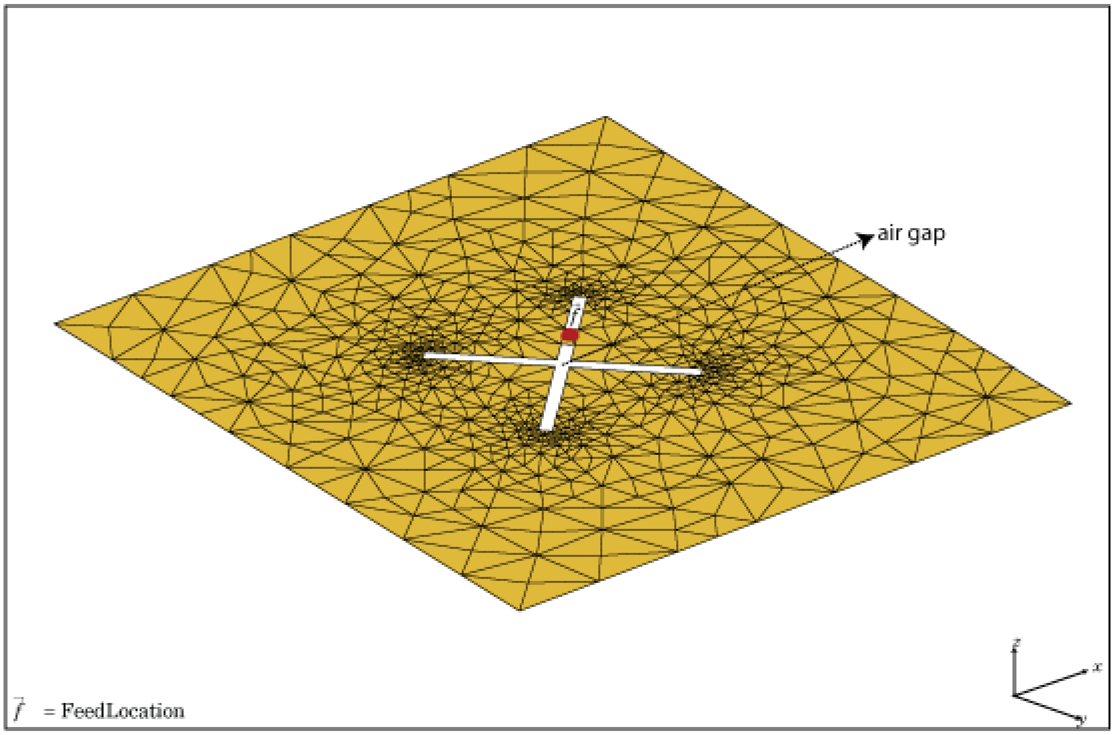
createFeed

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 handle

Custom mesh antenna, specified as a scalar handle.

`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

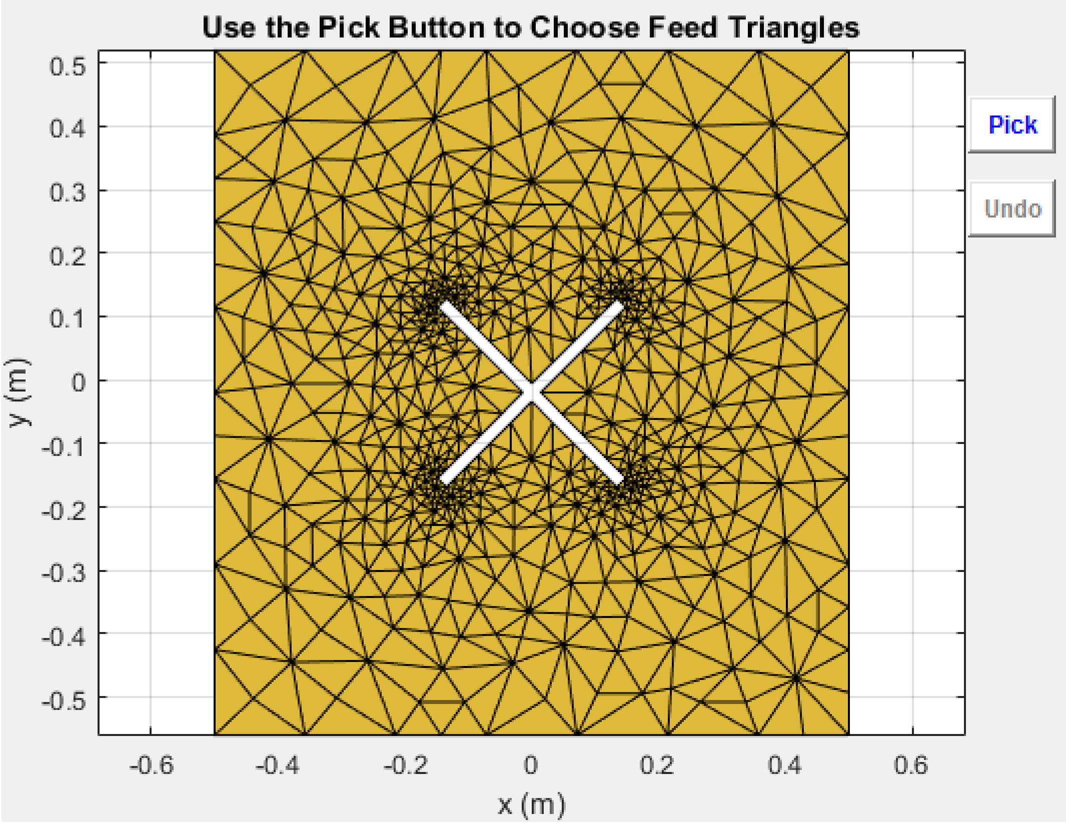
Load a 2-D custom mesh. Create a custom antenna using the points and triangles.

```
load planarmesh.mat
c = customAntennaMesh(p,t)
```

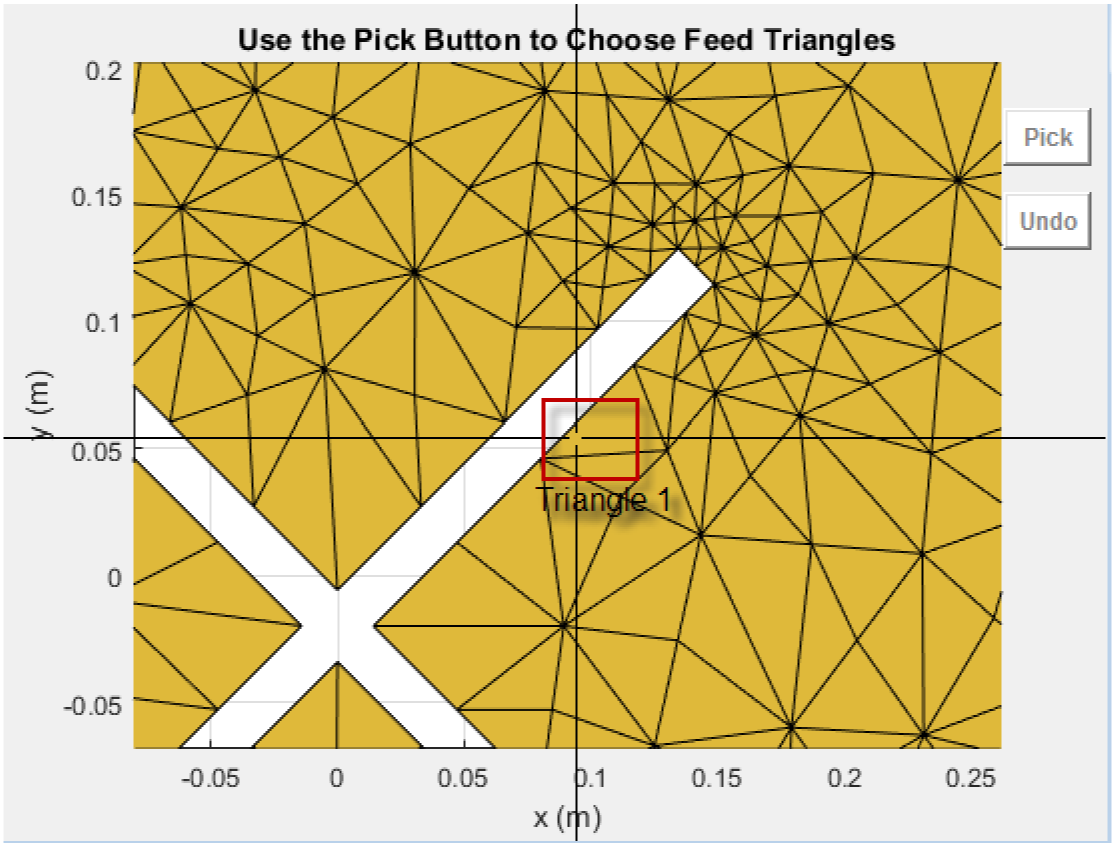
```
c = 

  customAntennaMesh with properties:

          Points: [3x658 double]
       Triangles: [4x1219 double]
    FeedLocation: []
            Tilt: 0
        TiltAxis: [1 0 0]
```

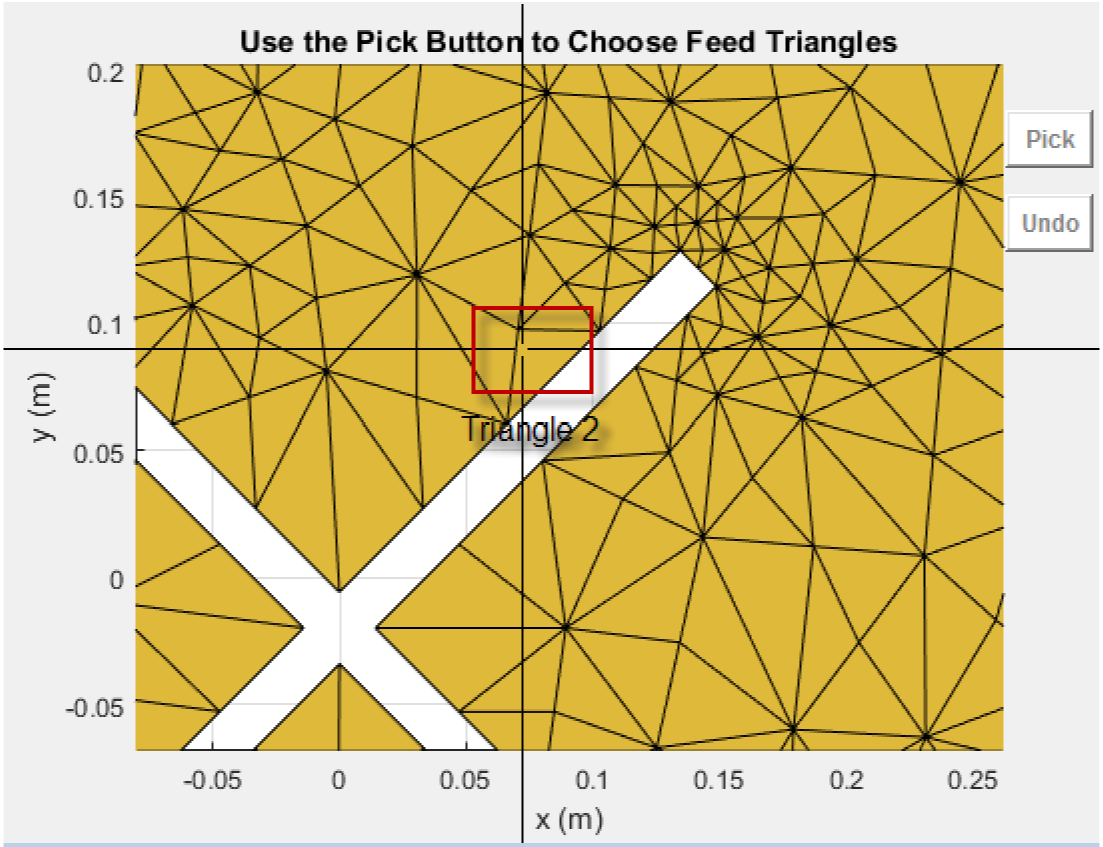
Use the `createFeed` function to view the antenna mesh structure. In this antenna mesh view, you see **Pick** and **Undo** buttons. The **Pick** button is highlighted.

```
createFeed(c)
```

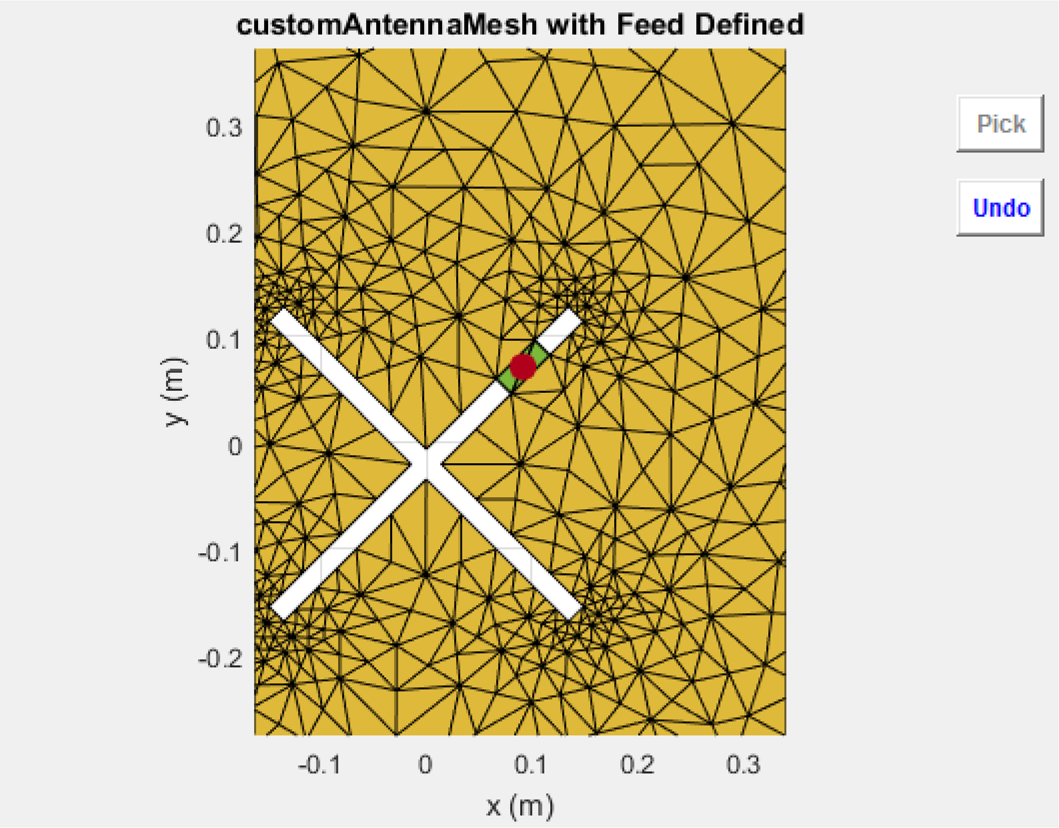


Click **Pick** to display the cross-hairs. To specify a region for the feed point, zoom in and select two points, one inside each triangle on either side of the air gap. Select the points using the cross-hairs.


Use the Pick Button to Choose Feed Triangles
Pick
Undo
Triangle 1
y (m)
x (m)
0.2
0.15
0.1
0.05
0
-0.05
-0.05
0
0.05
0.1
0.15
0.2
0.25



Selecting the second triangle creates and displays the antenna feed.



Create Feed for Custom Mesh Antenna Using Triangles Sharing Edge

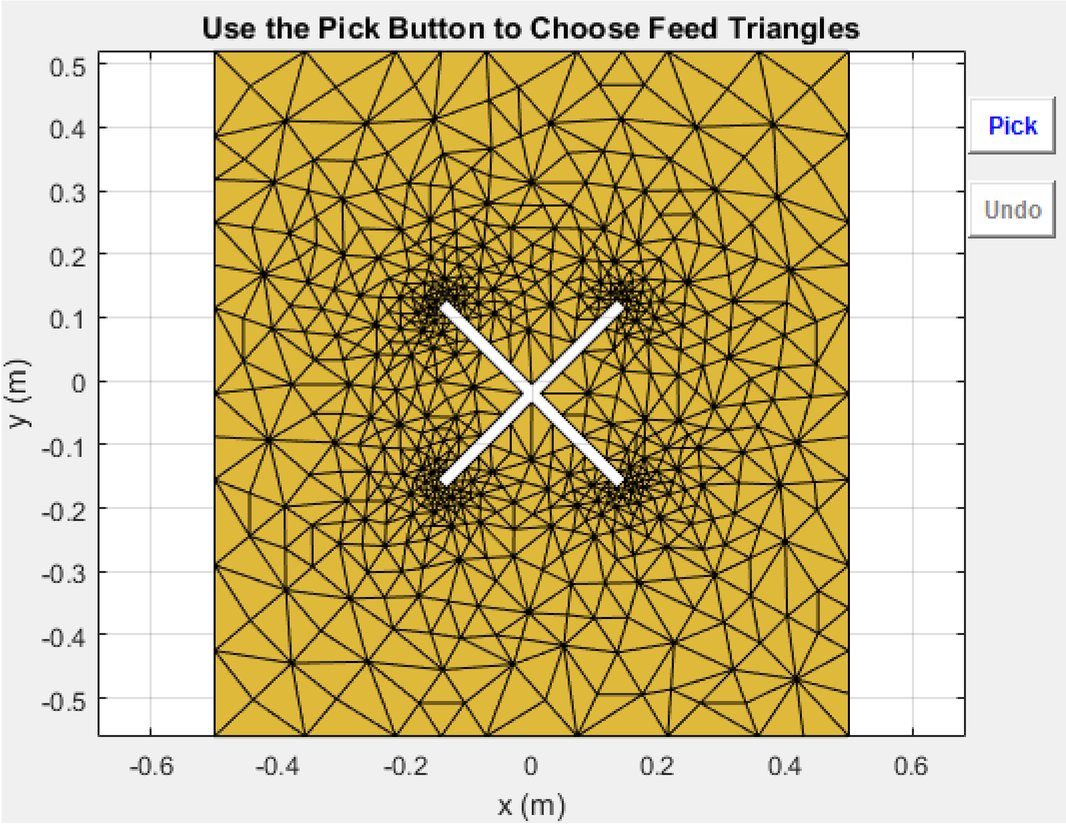
Load a 2-D custom mesh. Create a custom antenna using the points and triangles.

```
load planarmesh.mat
c = customAntennaMesh(p,t)
```

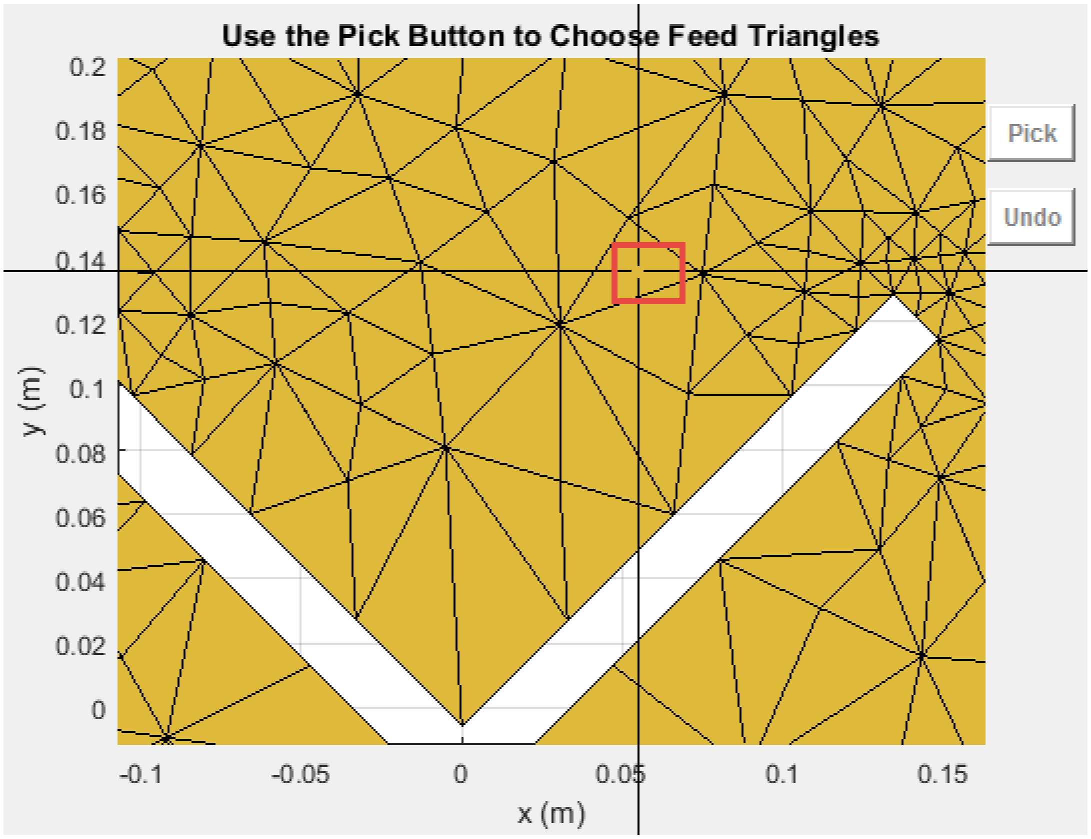
```
c = 

  customAntennaMesh with properties:

          Points: [3x658 double]
       Triangles: [4x1219 double]
    FeedLocation: []
            Tilt: 0
       TiltAxis: [1 0 0]
```

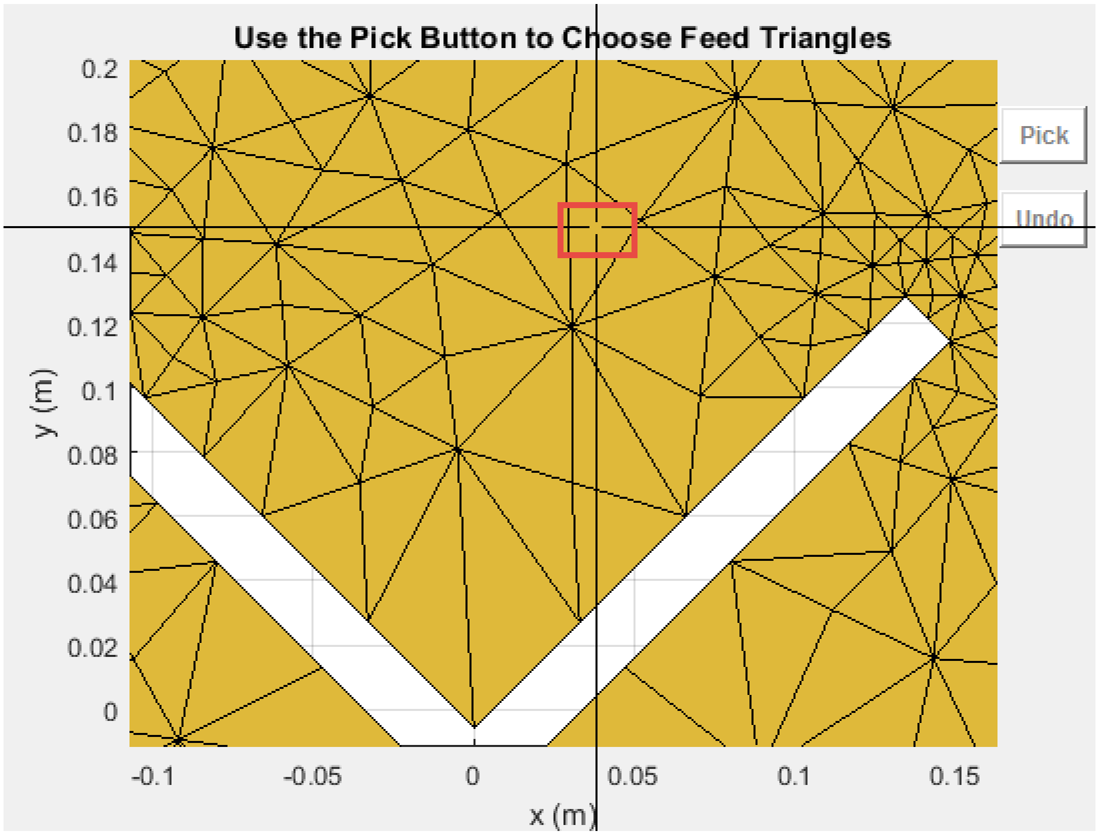
Use the `createFeed` function to view the antenna mesh structure. In this antenna mesh view, you see **Pick** and **Undo** buttons. The **Pick** button is highlighted.

```
createFeed(c)
```

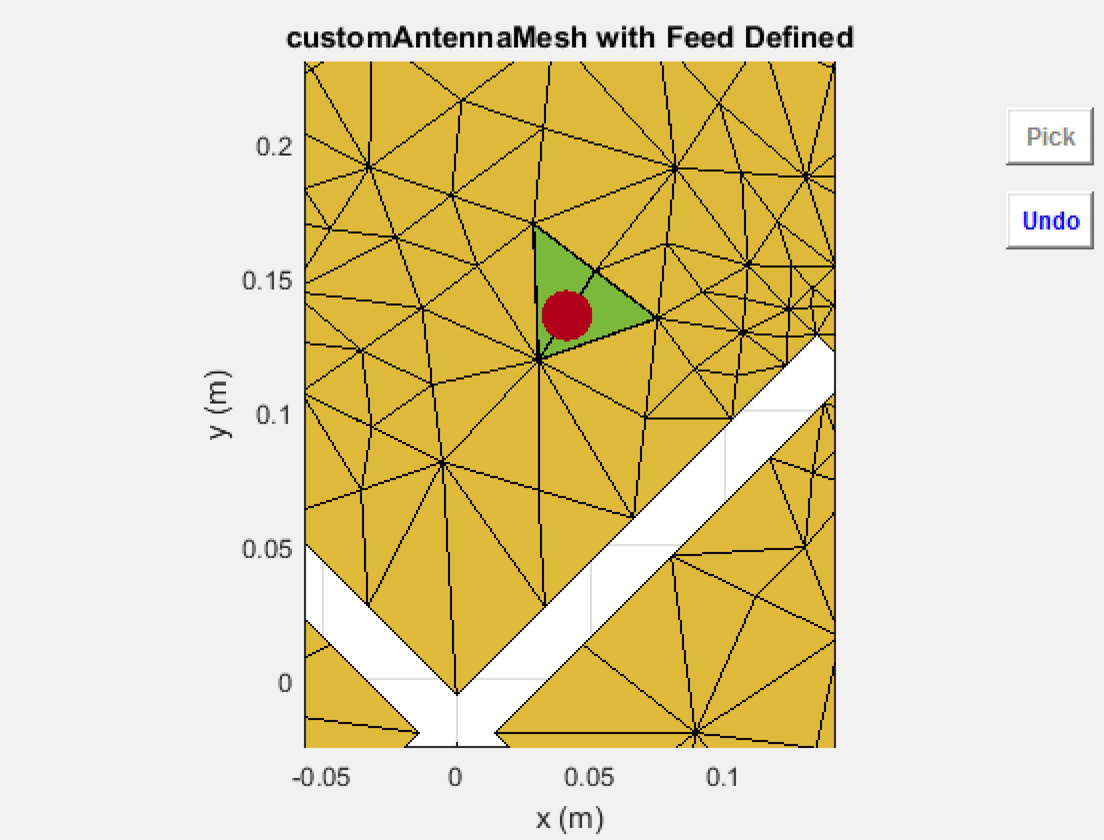


Click **Pick** to display the cross-hairs. To specify a region for the feed point, zoom in and select two points, one inside each triangle sharing an edge. Select the points using the cross-hairs.


Use the Pick Button to Choose Feed Triangles
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
y (m)
-0.1
-0.05
0
0.05
0.1
0.15
x (m)
Pick
Undo



Selecting the second triangle creates and displays the antenna feed.



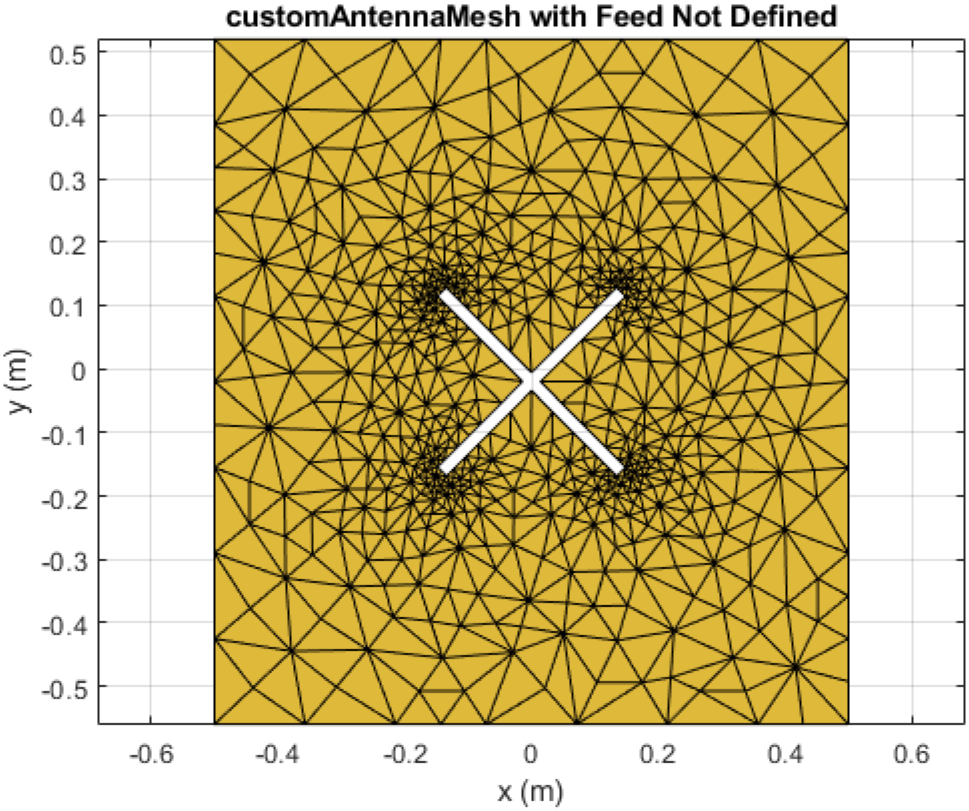
Create Feed for Custom Antenna Mesh

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)
```

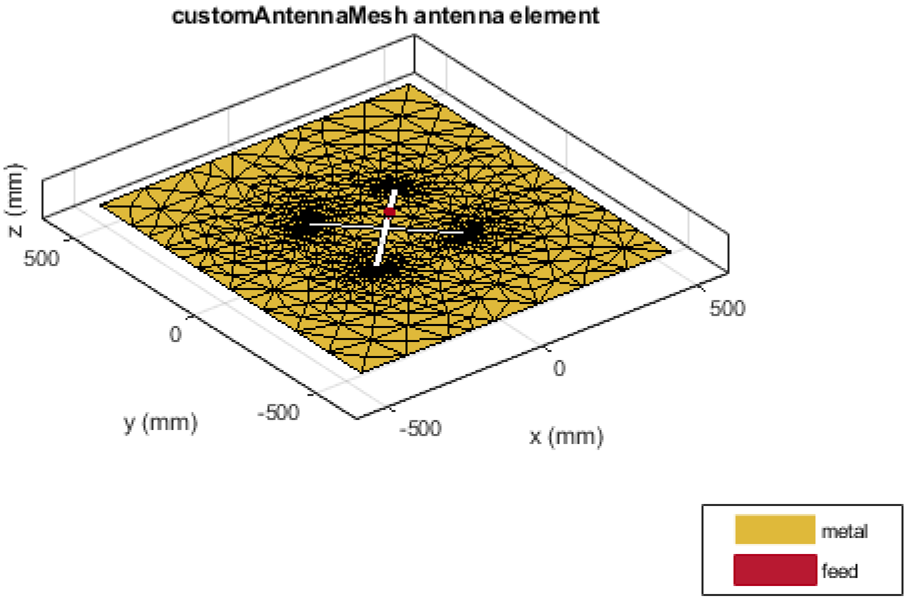
```
c = 
  customAntennaMesh with properties:

         Points: [3x658 double]
      Triangles: [4x1219 double]
   FeedLocation: []
           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)
```



See Also

`returnLoss` | `sparameters`

Introduced in R2015b

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 electric field and magnetic field of an antenna or array object at a specified frequency.

`EHfields(object,frequency)` plots the electric and magnetic field vectors at specified frequency values and at specified points in space.

`[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 space and at a specified frequency.

`EHfields(object, frequency, points)` plots the electric and magnetic field vectors at specified frequency values and at specified points in space.

`EHfields(___,Name,Value)` plots the electric and magnetic field vectors with additional options specified by one or more `Name Value` pair arguments using any of the preceding syntaxes.

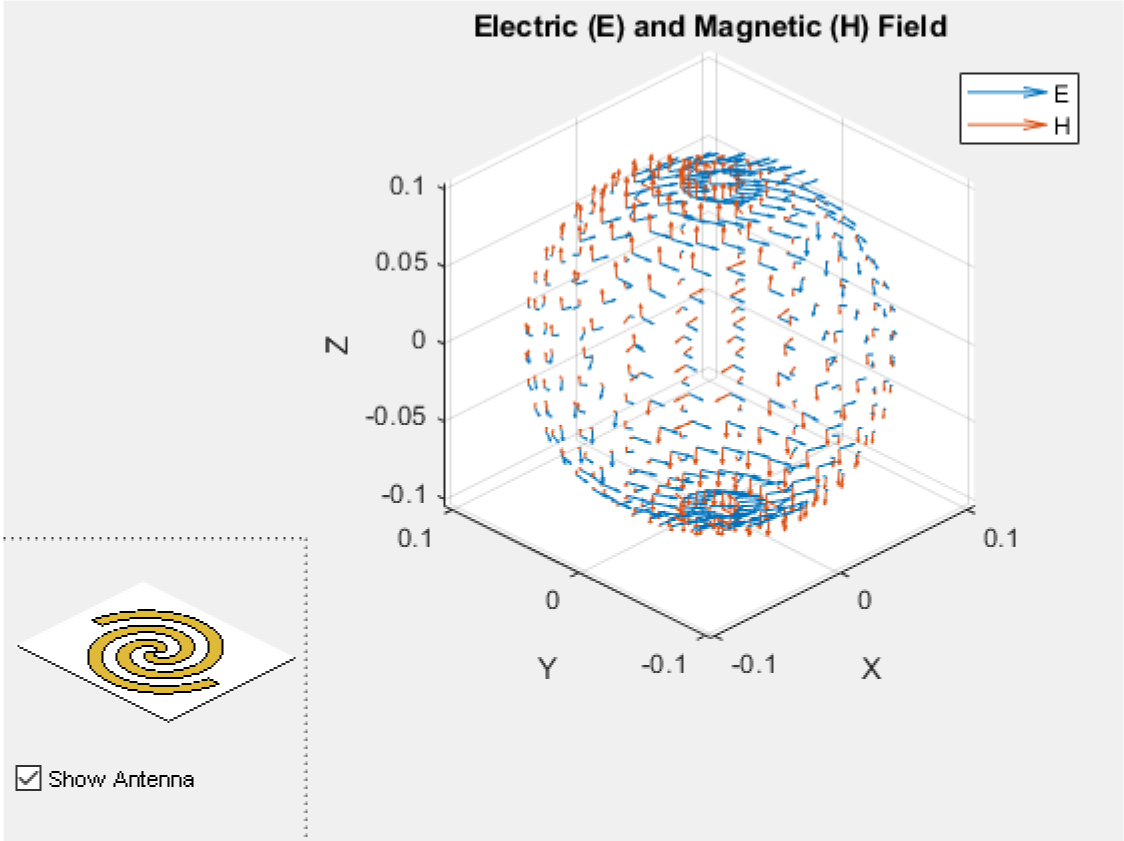
Use the `'ElementNumber'` and `'Termination'` property to calculate the embedded electric and magnetic fields of the antenna element in an array connected to a voltage source. The voltage source model consists of an ideal voltage source of 1 volt in series with a source impedance. The embedded pattern 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

   0.4137 + 0.2557i
   0.3040 - 0.4084i
   0.0000 + 0.0000i


h = 3×1 complex

  -0.0008 + 0.0011i
   0.0011 + 0.0007i
  -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.

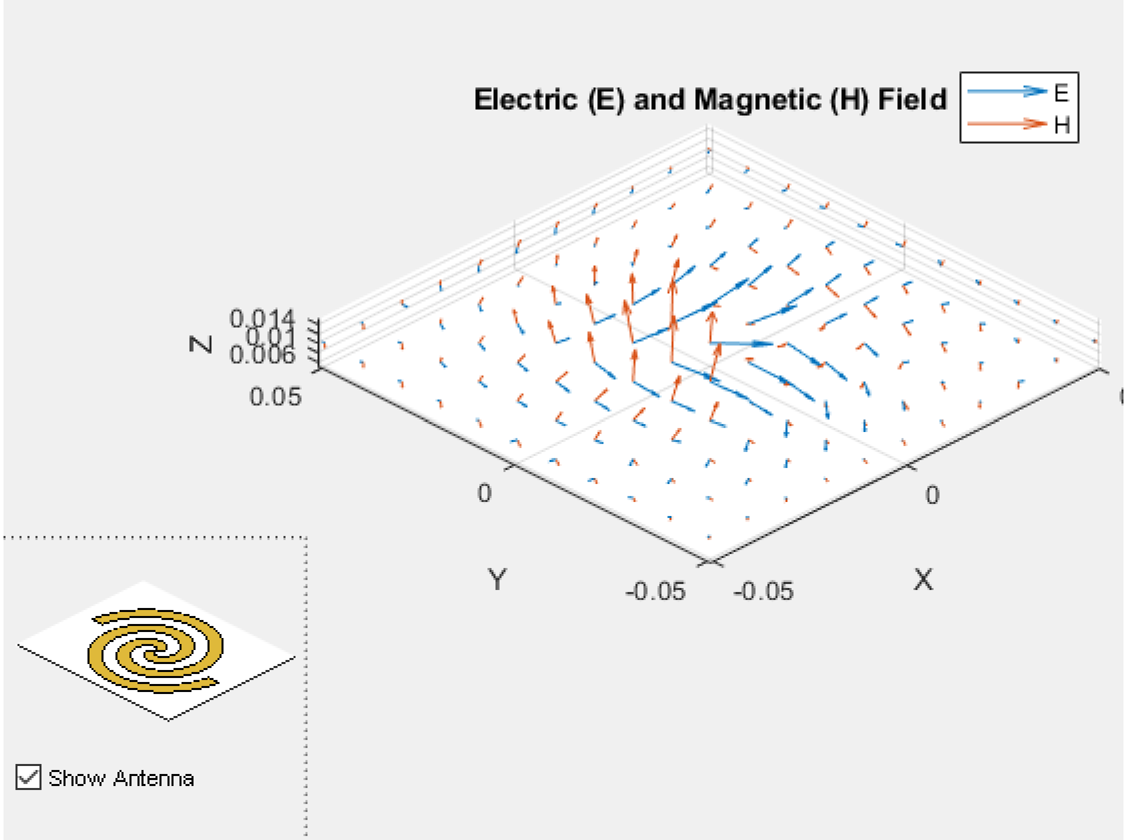
```
[X,Y] = meshgrid(-.05:.01:.05,-.05:.01:.05);
```

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

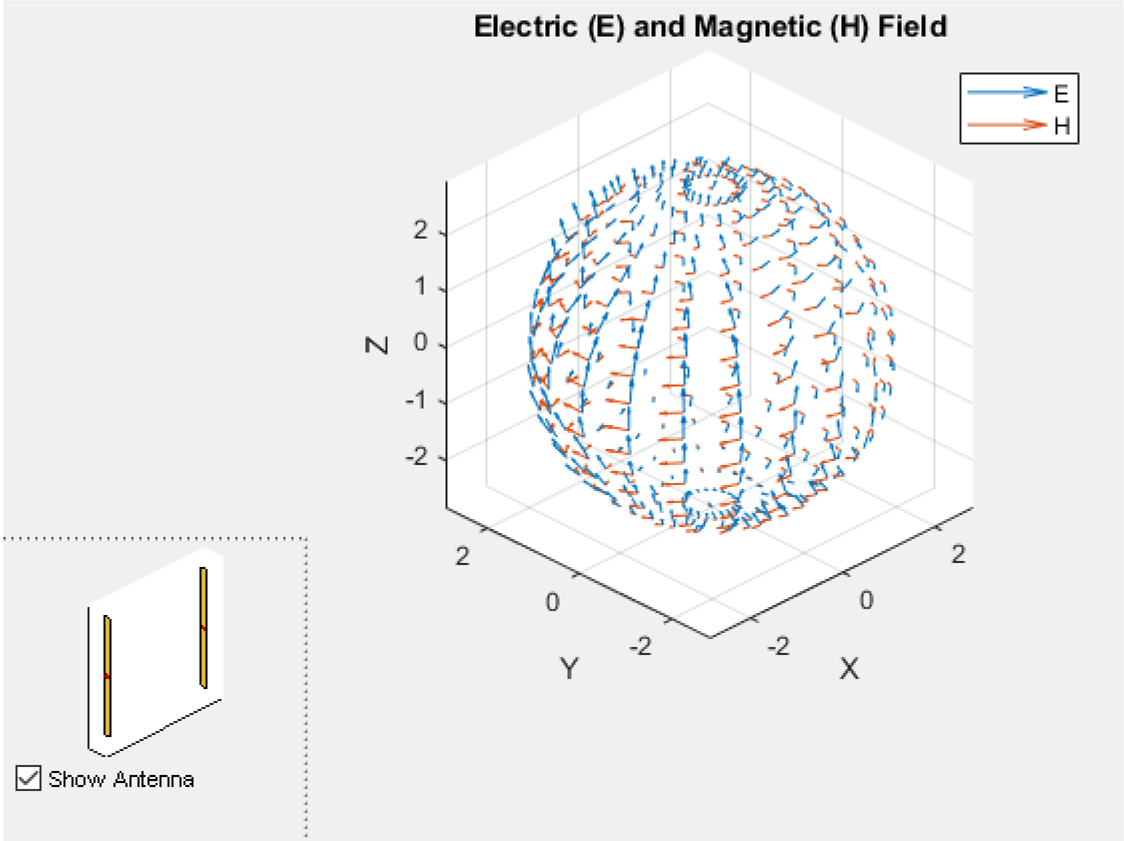
```
EHfields (h,4e9,p)
```



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);
```



Input Arguments

`object` — Antenna or array object

scalar handle

Antenna or array object, specified as a scalar handle.

Example: `h = spiralArchimedean`

Data Types: `function_handle`

`frequency` — Frequency used to calculate electric and magnetic fields

scalar in Hz

Frequency used to calculate electric and magnetic fields, specified as a scalar 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 a text input. `'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`

Output Arguments

e — *x*, *y*, *z* components of electrical field

3-by-*p* complex matrix in V/m

x, *y*, *z* components of electrical field, 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, returned as a 3-by-*p* complex matrix in H/m. The dimension *p* is the number of points in space at which the electric and magnetic fields are computed.

See Also

`axialRatio` | `beamwidth`

Introduced in R2015a

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.

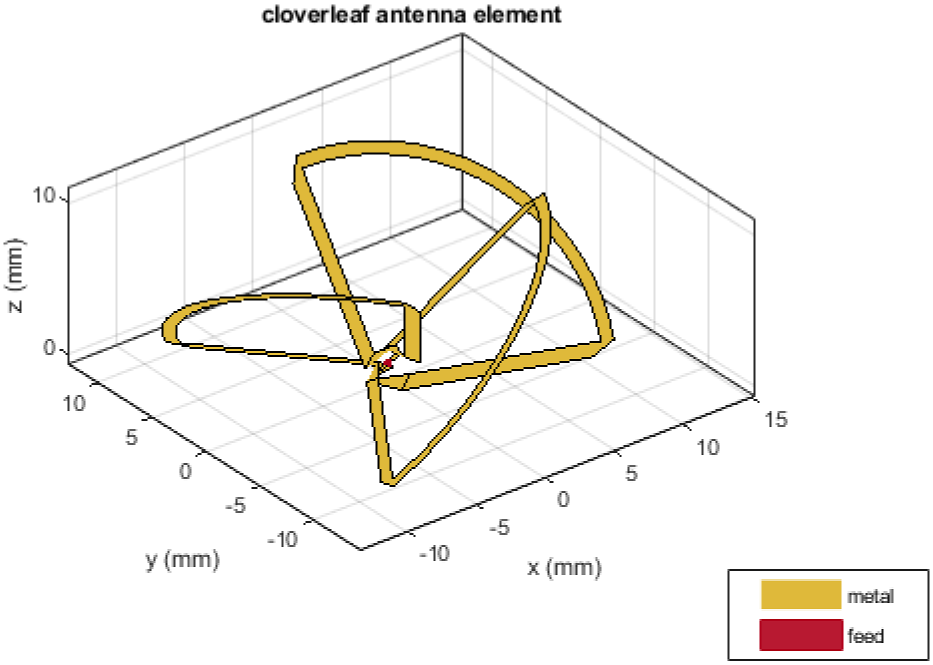
```
s  = spiralEquiangular;
ar = axialRatio(s,3e9,0,0)
```

```
ar = Inf
```

Axial Ratio of Cloverleaf Antenna

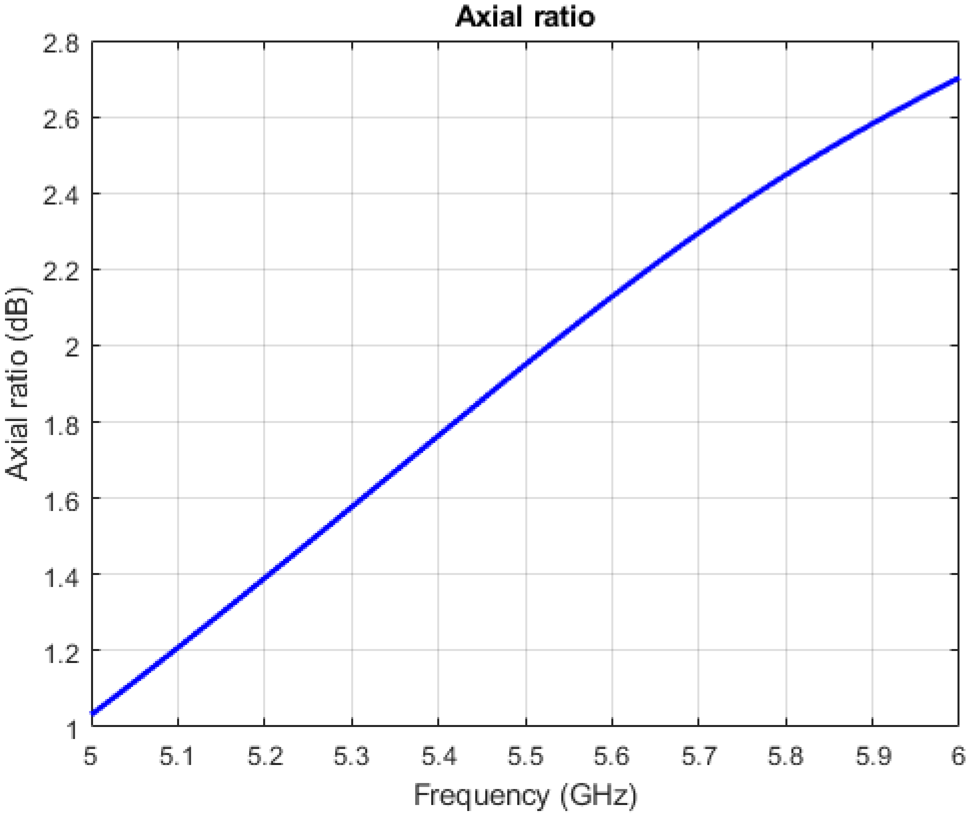
Create a cloverleaf antenna.

```
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);
```



You can see from the axial ratio plot 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.

`elevation` — Elevation angle of antenna
scalar | vector

Elevation angle of antenna, specified as a scalar or vector with each element in degrees.

Output Arguments

ar — Axial ratio of antenna

scalar in dB

Axial ratio of antenna, returned as a scalar in dB.

See Also

`beamwidth` | `pattern`

Introduced in R2015a

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.

`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.

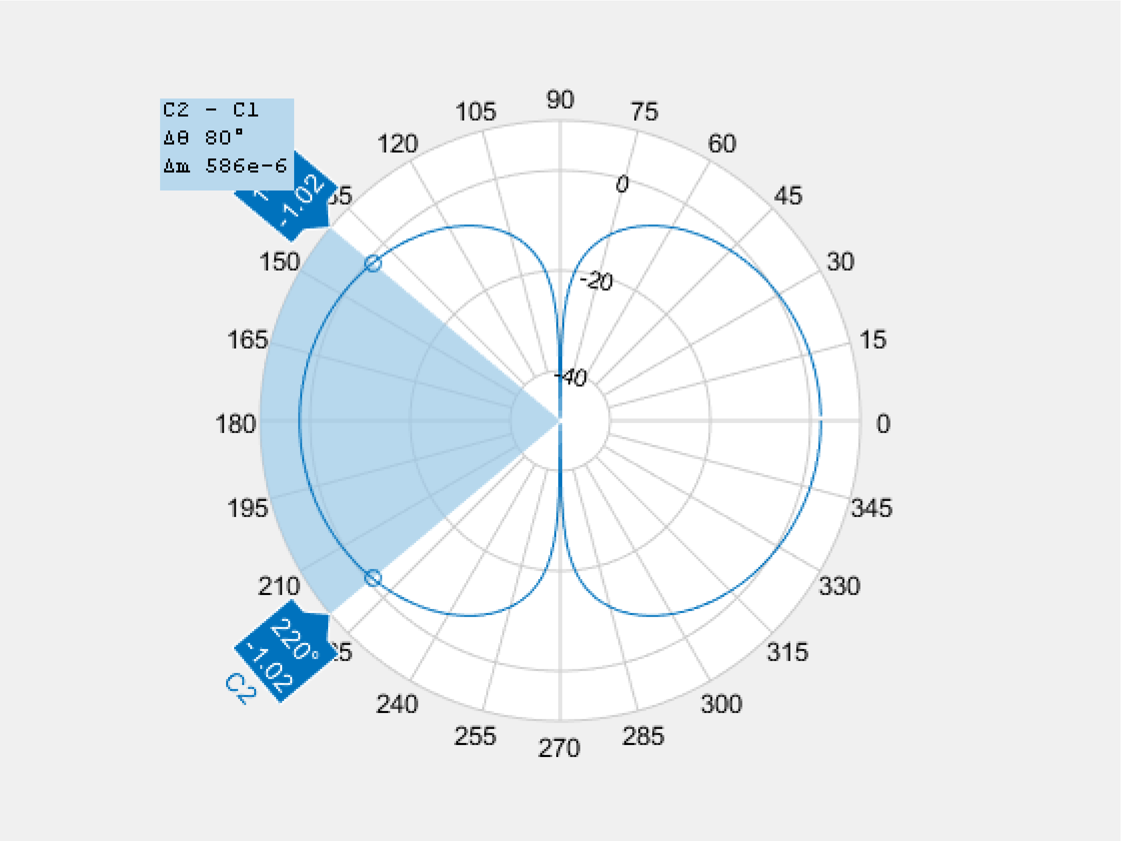
`[bw,angles] = beamwidth(____)` returns the beamwidth and angles (points in a plane) using any input arguments from previous syntaxes.

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 = 145
```

```
angles = 1×2

   143   288
```

Input Arguments

antenna — Antenna object
scalar handle

Antenna object, specified as a scalar handle.

`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 in degrees

Beamwidth of antenna, returned as a scalar in degrees.

`angles` — Points on plane

vector in degrees

Points on plane used to measure beamwidth, returned as a vector in degrees.

See Also

`axialRatio` | `pattern`

Introduced in R2015a

mesh

Mesh properties of metal or dielectric antenna or array structure

Syntax

```
mesh(object)
mesh(shape)
mesh(object,Name,Value)
meshdata = mesh( ___ ,Name,Value)
```

Description

`mesh(object)` plots the mesh used to analyze antenna or array element.

`mesh(shape)` plots the mesh for the shapes.

`mesh(object,Name,Value)` changes and plots the mesh structure of an antenna or array element, using additional options specified by the name-value pairs. You can also determine the number of unknowns from the number of basis functions in the output.

`meshdata = mesh( ___ ,Name,Value)` returns a mesh structure that specifies the properties used to analyze the antenna or array.

Examples

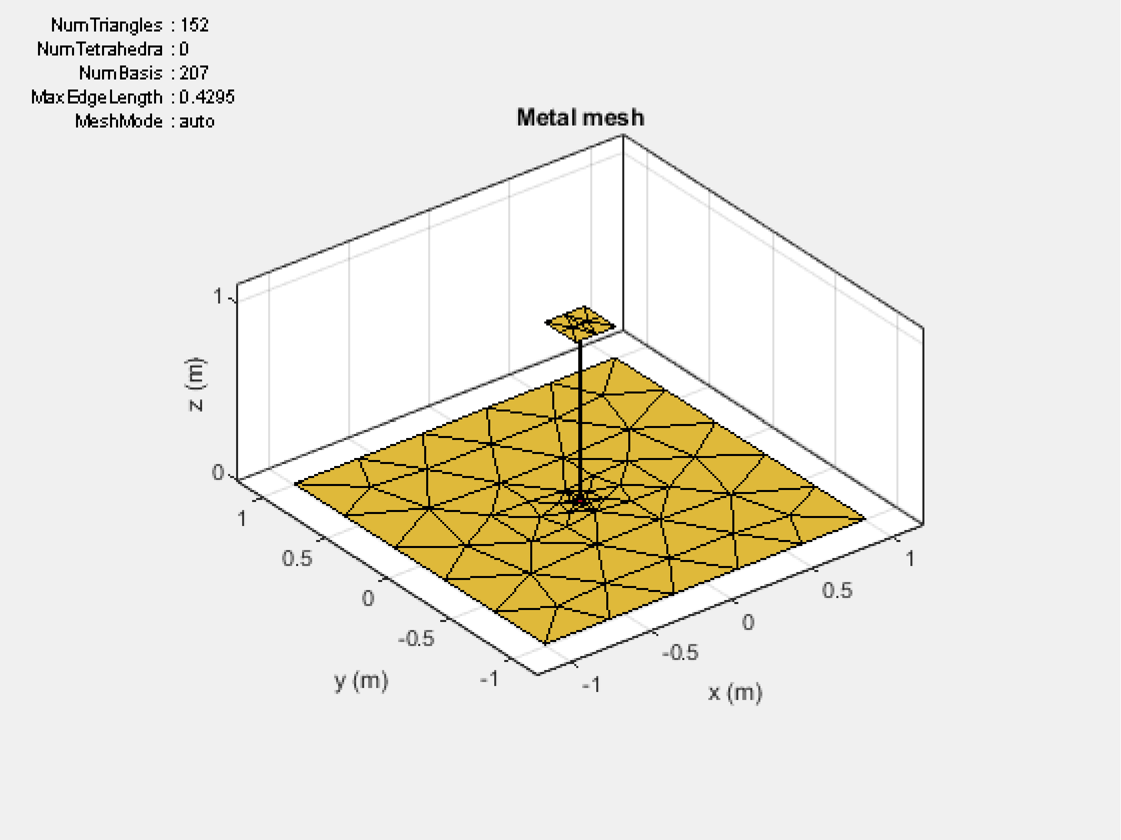
View Mesh Structure of Antenna

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 = 2.5322e+02 + 6.0784e+02i
```

```
mesh(h)
```



```
m = mesh(h)
```

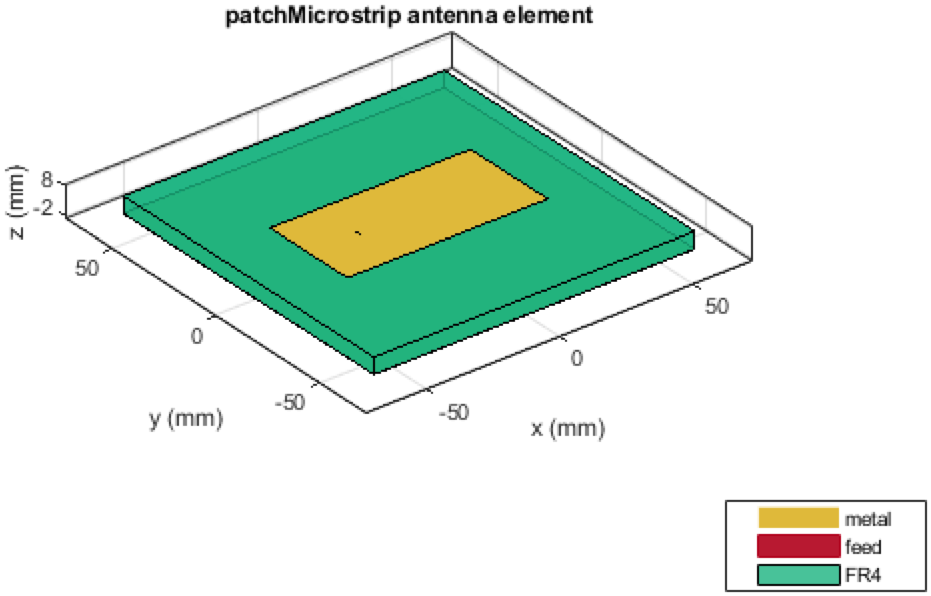
```
m = struct with fields:
     NumTriangles: 152
    NumTetrahedra: 0
        NumBasis: 207
    MaxEdgeLength: 0.4295
         MeshMode: 'auto'
```

Mesh Microstrip Patch Metal-Dielectric Antenna

Radiation Pattern of Microstrip Patch Antenna

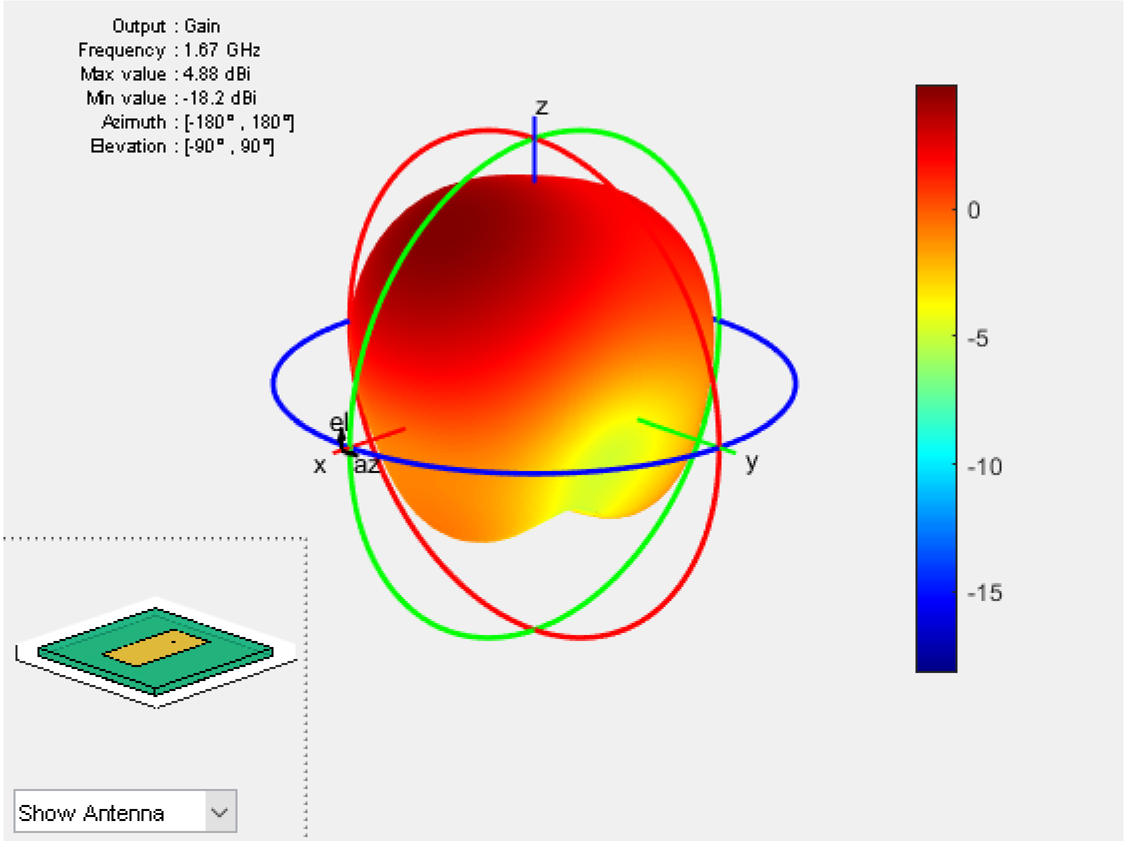
Create a microstrip patch antenna using **'FR4'** as the dielectric substrate.

```
d = dielectric('FR4');
pm = patchMicrostrip('Length',75e-3, 'Width',37e-3,                 ...
        'GroundPlaneLength',120e-3, 'GroundPlaneWidth',120e-3, ...
        'Substrate',d);
show(pm)
```



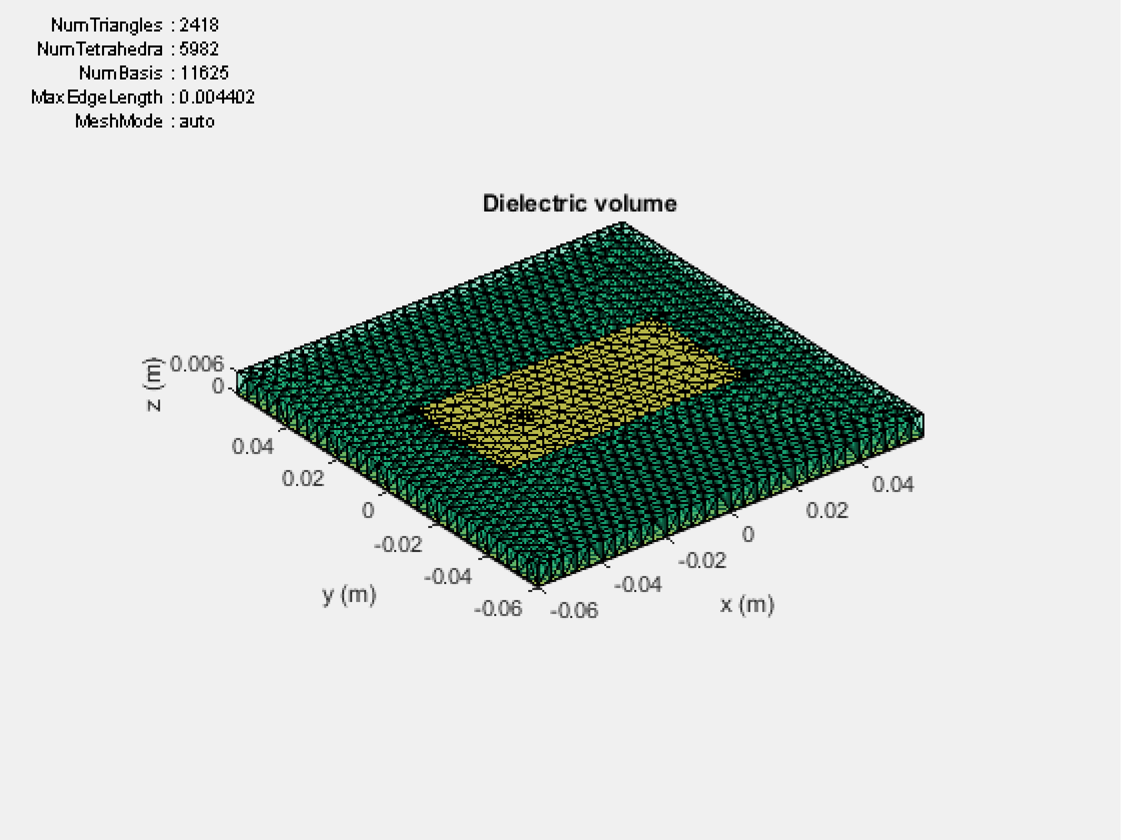
Plot the radiation pattern of the antenna at a frequency of 1.67 GHz.

```
figure
pattern(pm,1.67e9)
```



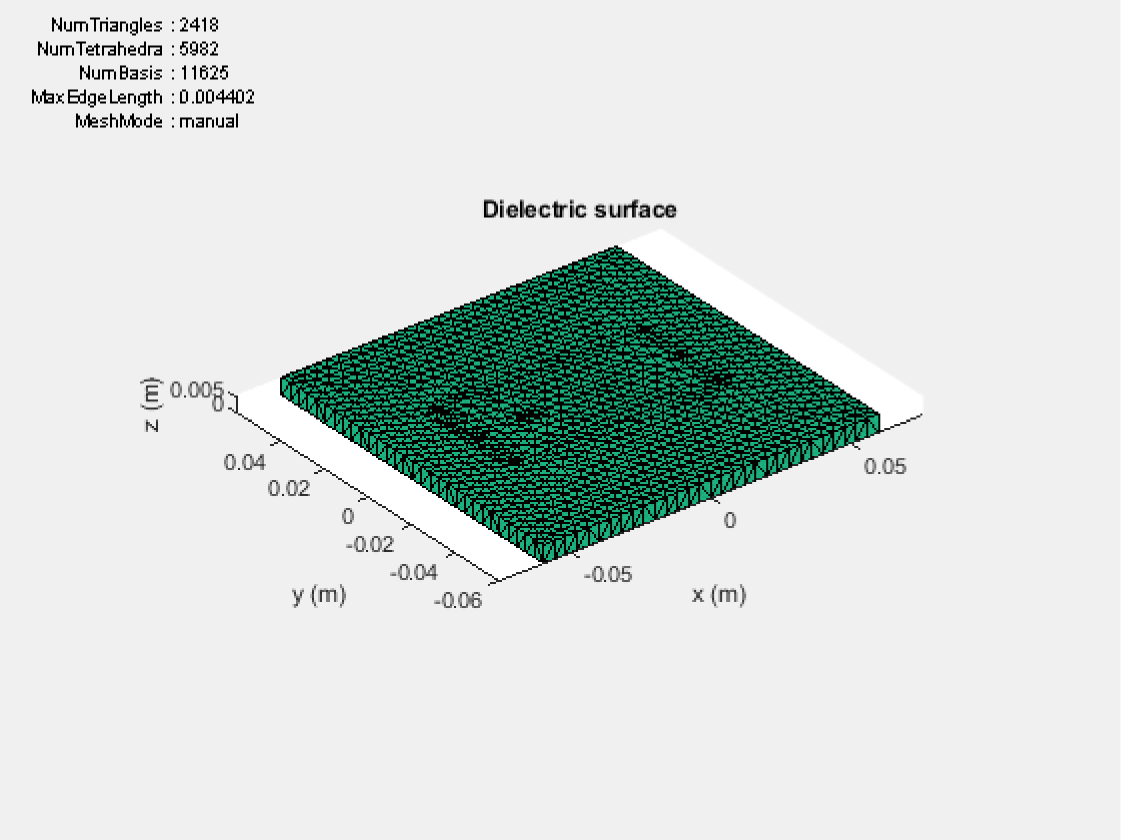
Mesh the whole antenna.

```
figure
mesh(pm)
```



Mesh only the dielectric surface of the antenna.

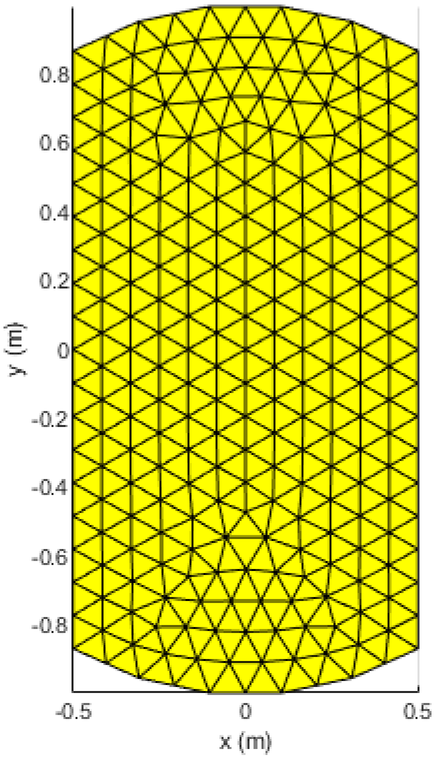
```
figure
mesh(pm,'View','dielectric surface')
```



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);
```



Input Arguments

`object` — Antenna or array element
object

Antenna or array element, specified as an object.

`shape` — Shape created using custom elements and shape objects
object handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle. You can create the shapes using `antenna.Circle`, `antenna.Polygon`, or `antenna.Rectangle`.

Example: `c = antenna.Rectangle; mesh(c)`

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: ‘MaxEdgeLength’, 0.1

`MaxEdgeLength` — Maximum edge length of triangles in mesh
scalar

Maximum edge length of triangles in mesh, specified as a comma-separated pair consisting of `'MaxEdgeLength'` and a scalar. All triangles in the mesh have sides less than or equal to the `'MaxEdgeLength'` value.

Data Types: `double`

`MinEdgeLength` — Minimum edge length of triangles in mesh

scalar

Minimum edge length of triangles in mesh, specified as a comma-separated pair consisting of `'MinEdgeLength'` and a scalar. All triangles in the mesh have sides less than or equal to the `'MinEdgeLength'`.

Note You can use this property only with the `pcbStack` object.

Data Types: `double`

`GrowthRate` — Mesh growth rate

`0.7` (default) | scalar

Mesh growth rate, specified as a comma-separated pair consisting of `'GrowthRate'` and a scalar. The default value of `0.7` states that the growth rate of the mesh is 70 percent. Growth rate values lie between `0` and `1`.

Note You can use this property only with the `pcbStack` object.

Data Types: `double`

`View` — 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.

Data Types: `char`

See Also

`meshconfig` | `plot` | `show`

Introduced in R2015a

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.

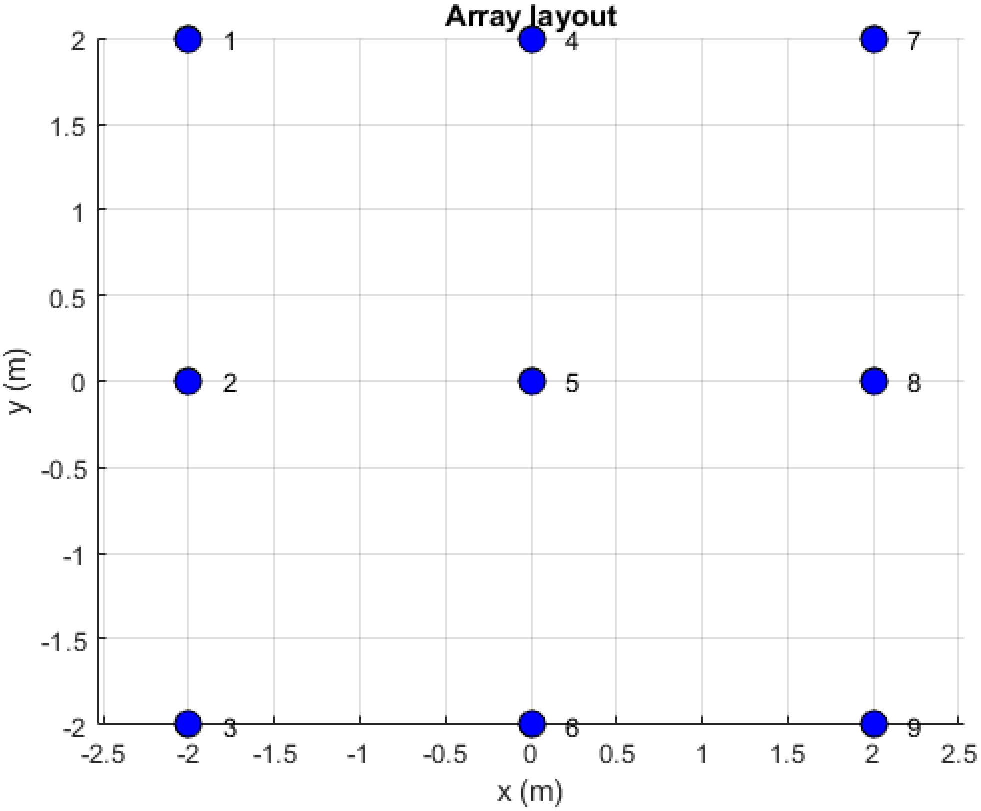
`layout(pcbstack)` displays the layout of the PCB stack object. The circles in the layout corresponds to antenna feed points on the PCB.

Examples

Display Array Layout on X-Y Plane

Create and view a 3x3 rectangular array layout on the X-Y plane.

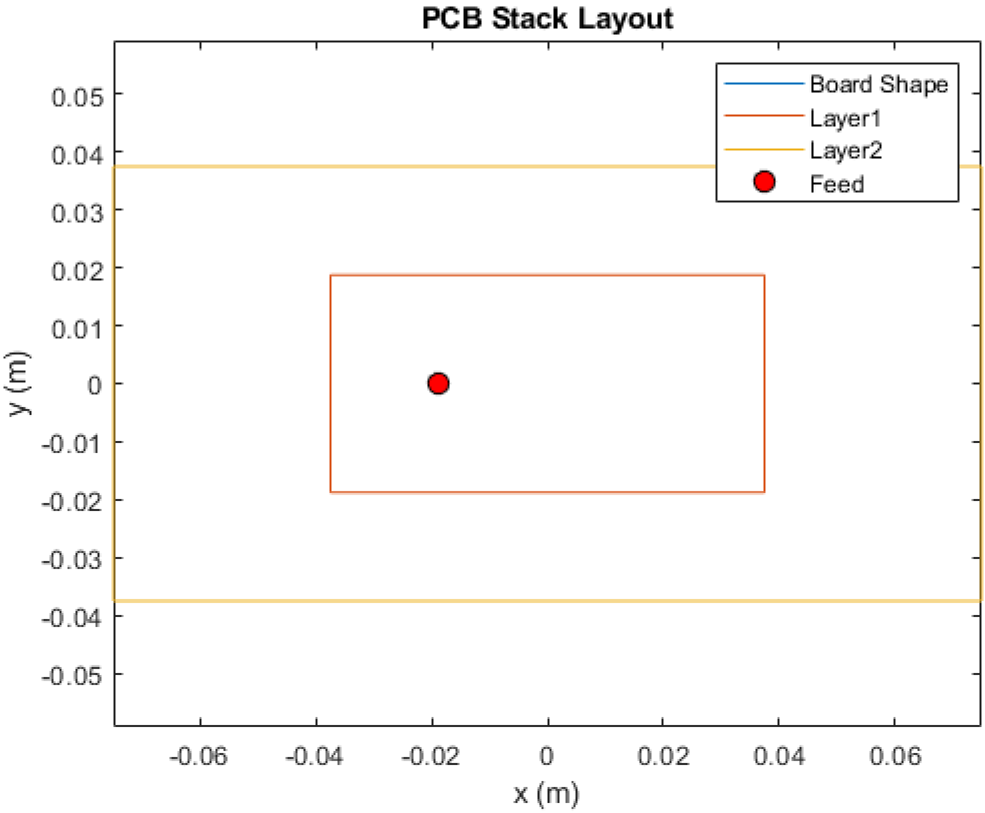
```
h = rectangularArray('Size',[3 3]);
layout(h)
```



Display PCB Stack Layout

Default PCB stack layout.

```
p = pcbStack;
layout(p)
```



Input Arguments

`array` — Array object

scalar handle

Array object, specified as a scalar handle.

`pcbstack` — PCB stack

`pcbStack` object

PCB stack, specified as a `pcbStack` object.

See Also

`pcbStack` | `show`

Introduced in R2015a

lumpedElement

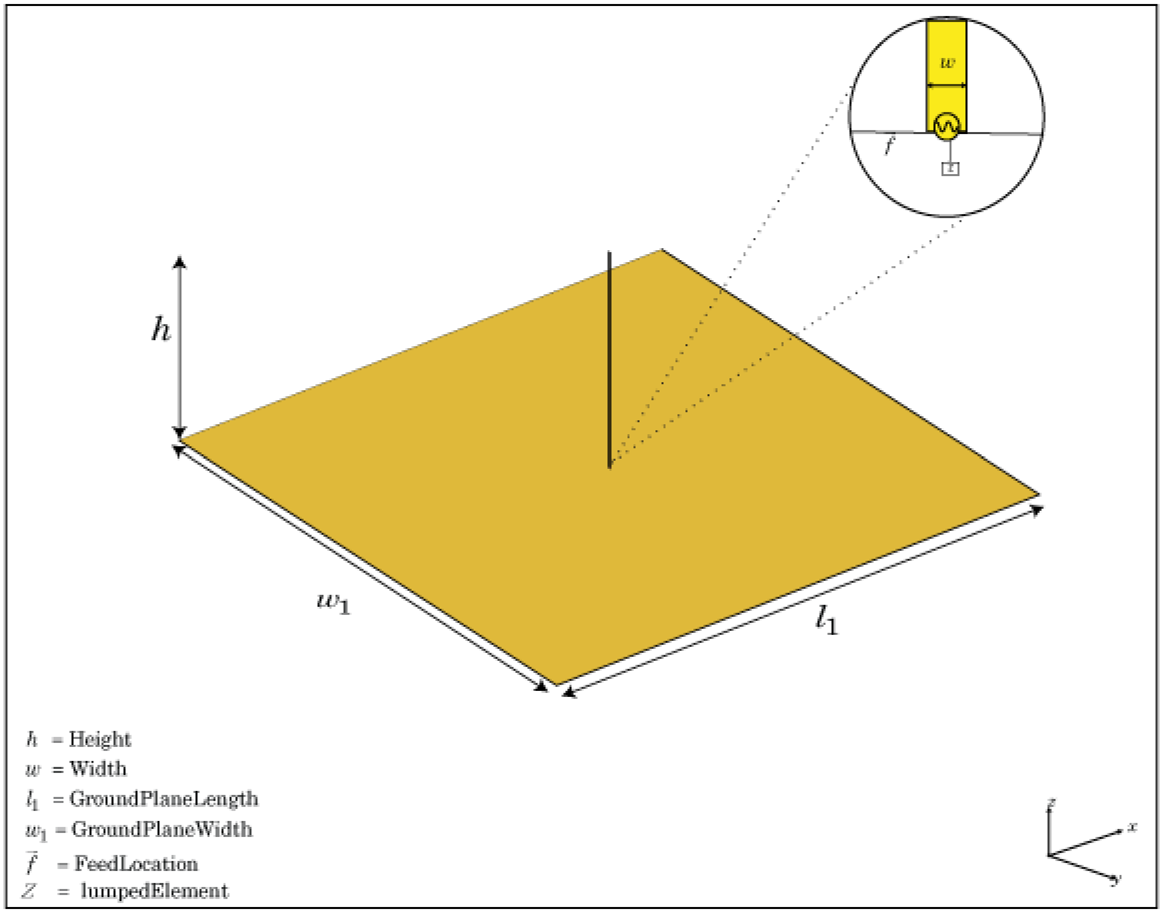
Lumped element circuit to load antenna

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 properties specified by one or more `Name,Value` pair 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 = 72.9381 - 1.2090i
```

Load the antenna using a frequency-independent resistor. Calculate the impedance of the antenna.

```
d.Load = le;
i1e1 = impedance(d,70e6)
```

```
i1e1 = 1.2294e+02 - 1.2090e+00i
```

Change the frequency to 85 MHz and calculate the impedance of the antenna.

```
ile2 = impedance(d,85e6)
```

```
ile2 = 2.3009e+02 + 1.1005e+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.

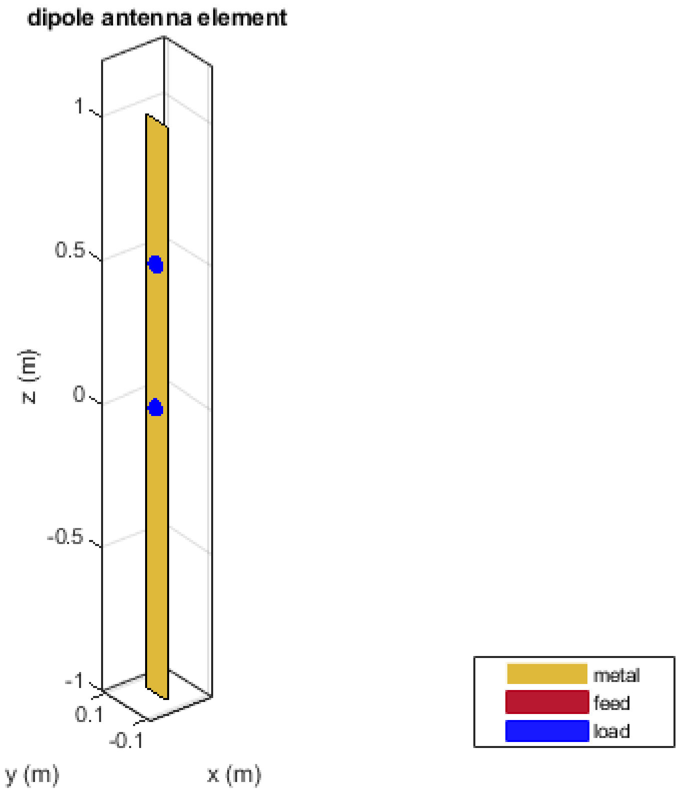
```
l2 = lumpedElement('Impedance', complex(50, -20), 'Location', [0 0 0.5]);
```

Add the two loads to the dipole antenna.

```
 d.Load = [l1, l2];
```

View the dipole antenna.

```
show(d);
```



Input Arguments

Name-Value Pair Arguments

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

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.

See Also

`dielectric`

Introduced in R2016b

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`.

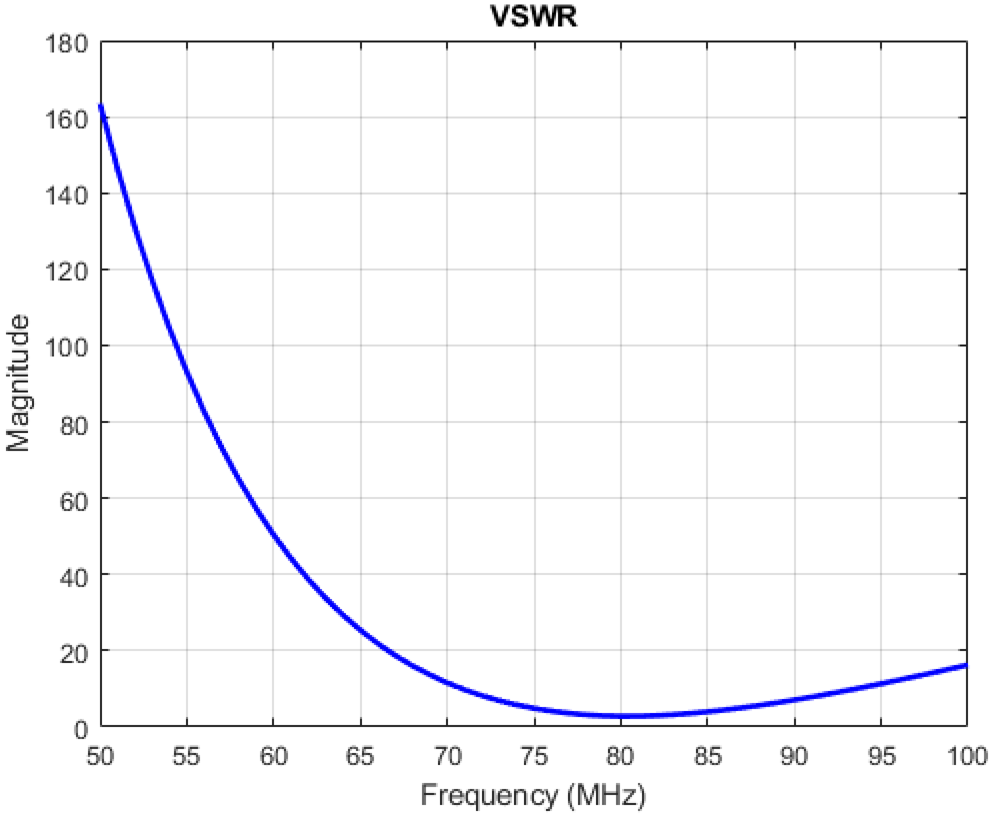
`vswrant = vswr(antenna,frequency,z0)` returns the vswr of the antenna.

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

    3.5730    6.7043    3.3598
```

Input Arguments

antenna — Antenna object
scalar handle

Antenna object, specified as a scalar handle.

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.

See Also

`impedance`

Introduced in R2015a

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`.

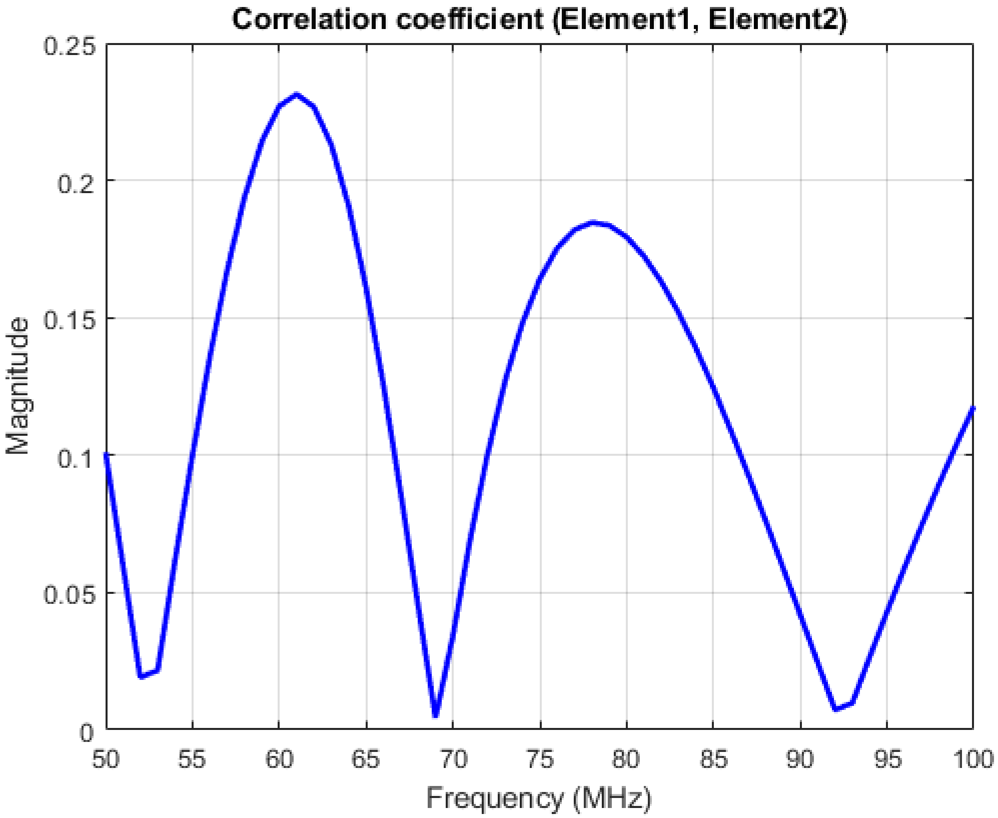
`rho = correlation(array,frequency,elem1,elem2,z0)` returns the correlation coefficient between two antenna elements, `elem1` and `elem2` of an array.

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
    0.0822
    0.0520
    0.0212
    0.0106
    0.0433
    0.0767
    0.1098
    0.1412
      ⋮
```

Input Arguments

`array` — Array object

scalar handle

Array object, specified as a scalar handle.

`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 handle

Antenna elements in an array, specified as a scalar handle.

`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.

See Also

`impedance` | `returnLoss` | `sparameters`

Introduced in R2015a

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

Output Arguments

w — Equivalent width of strip
scalar | vector

Equivalent width of strip, returned as a scalar or vector.

See Also

`helixpitch2spacing`

Introduced in R2015a

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.

See Also

`cylinder2strip`

Introduced in R2015a

meshconfig

Change mesh mode of antenna structure

Syntax

```
meshconfig(antenna,mode)
```

Description

`meshconfig(antenna,mode)` changes the meshing mode of the antenna according to the text input mode.

Examples

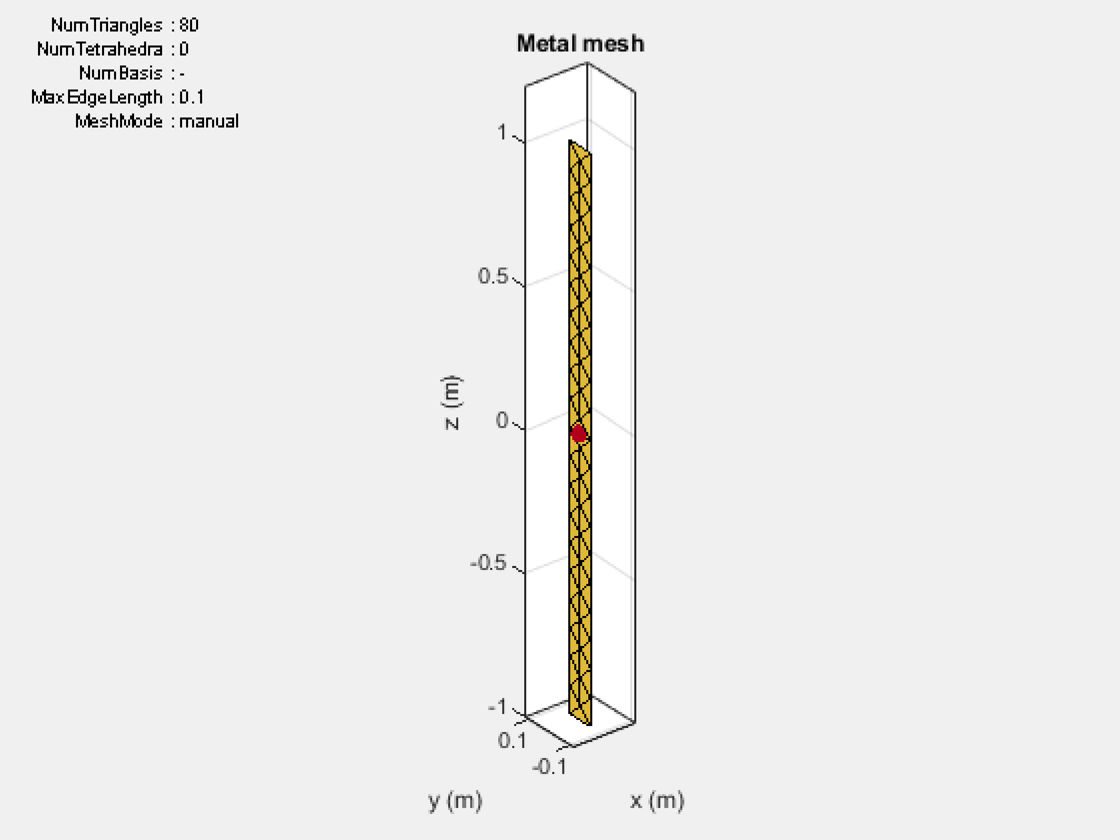
Change Mesh Configuration of Antenna

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: []
         MeshMode: 'manual'
```

```
mesh(h,'MaxEdgeLength',0.1)
```



Input Arguments

antenna — Antenna object
scalar handle

Antenna object, specified as a scalar handle.

mode — Meshing mode
'auto' (default) | 'manual'

Meshing mode, specified as `'auto'` or `'manual'`.

Data Types: `char`

See Also

`mesh` | `show`

Introduced in R2015a

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 handle

Infinite array, specified as a scalar handle.

`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 = 83.3052 + 68.7832i
```

Change the number of summation terms to 51. Calculate the scan impedance again.

```
numSummationTerms(h,25)
s = impedance(h,1e9)
```

```
s = 83.4474 + 68.8191i
```

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 = 83.4918 + 68.8244i
```

See Also

`beamwidth` | `pattern`

Topics

"Infinite Arrays"

Introduced in R2015b

feedCurrent

Calculate current at feed for antenna or array

Syntax

```
feedCurrent(obj,frequency)
```

Description

`feedCurrent(obj,frequency)` calculates the current 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])
cur = feedCurrent(h,75e6)
```

```
h = 

  planeWaveExcitation with properties:

         Element: [1×1 monopole]
       Direction: [1 0 0]
    Polarization: [0 0 1]


cur = 

   0.0132 - 0.0133i
```

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 = 0.0294 - 0.0012i
```

Input Arguments

`obj` — Antenna or array object

object handle

Antenna or array object, specified as an object handle.

`frequency` — Frequency to calculate feed current

scalar integer in Hz

Frequency to calculate feed current, specified as a scalar integer in Hz.

See Also

`current`

Introduced in R2017a

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.

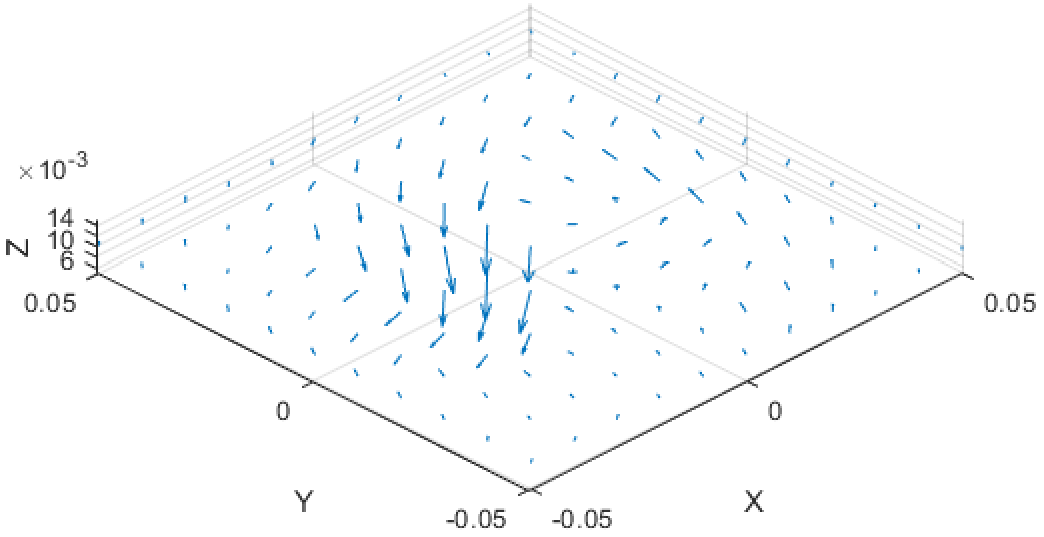
`fieldsCustom(axeshandle,___)` plots into the axes specified by `axeshandle` instead of the current axes.

Examples

Visualize Magnetic Field of Antenna Using `fieldsCustom`

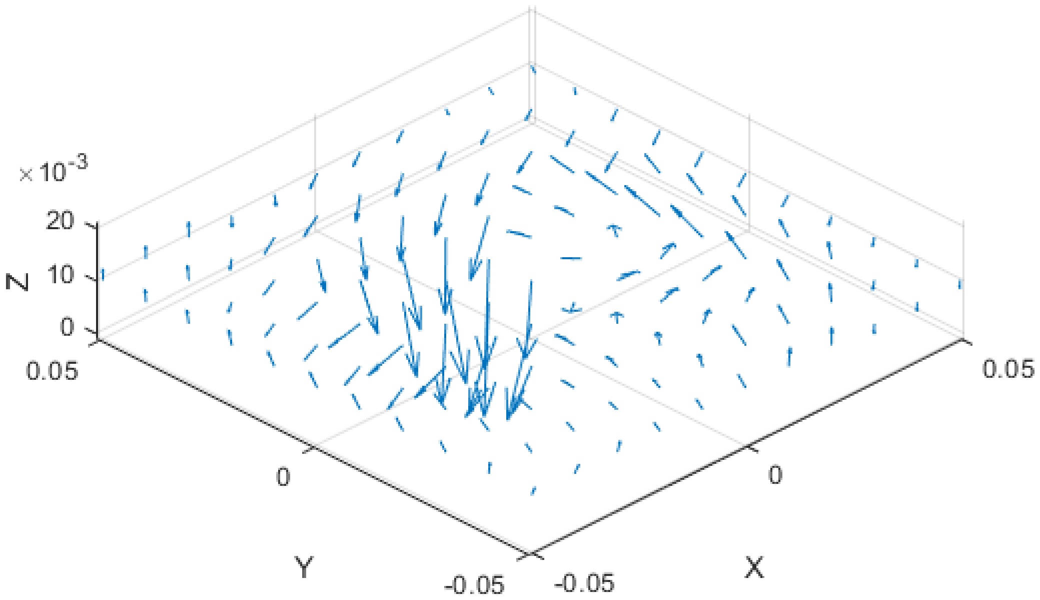
Load and visualize the magnetic field data available in the file '`fielddata.mat`'.

```
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 handle

Axes object, specified as an object handle.

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 handle

Electric or magnetic field plot, returned as quiver object handle.

See Also

EHfields | pattern | patternCustom

Introduced in R2016a

patternCustom

Plot radiation pattern

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 magnitude, `magE` over the specified `phi` and `theta` angle vectors.

`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

ExamplesVisualize Radiation Pattern From Antenna Data File

Visualize 3-D Electric Field Pattern of Dipole

Calculate the magnitude, azimuth, and elevation angles of a dipole's electric field at 75 MHz.

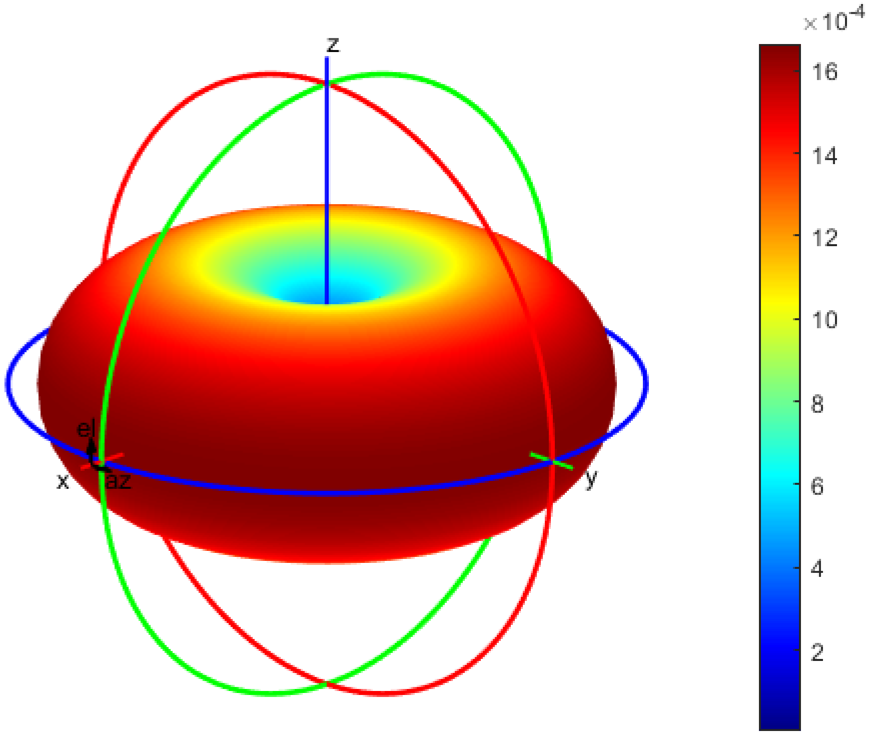
```
d = dipole;
[efield,az,el] = pattern(d, 75e6,'Type','efield');
```

Extract the theta and phi angles of the electric field magnitude of the antenna.

```
phi = az';
theta = (90-el);
MagE = efield';
```

Plot the 3-D electric field pattern.

```
patternCustom(MagE,theta,phi);
```



Visualize 2-D Radiation Patterns of Helix Directivity

Calculate the magnitude, azimuth, and elevation angles of a helix's directivity at 2 GHz.

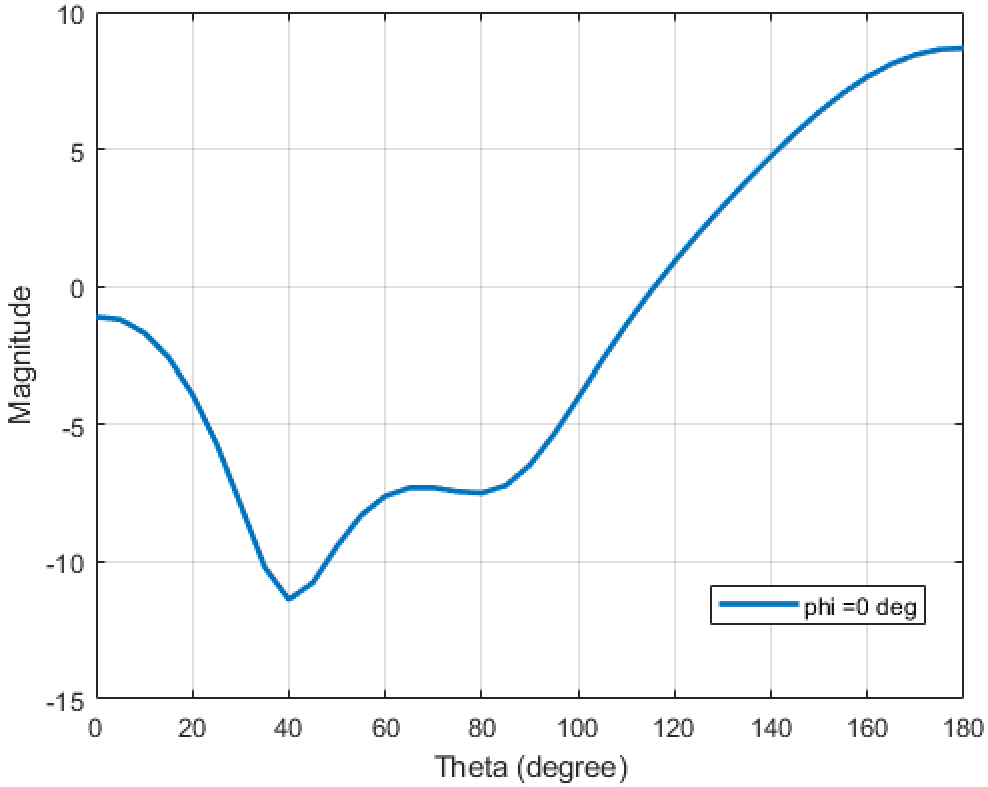
```
h = helix;
[D,az,el] = pattern(h,2e9);
```

Extract theta and phi angles of the directivity magnitude.

```
phi = az';
theta = (90-el);
MagE = D';
```

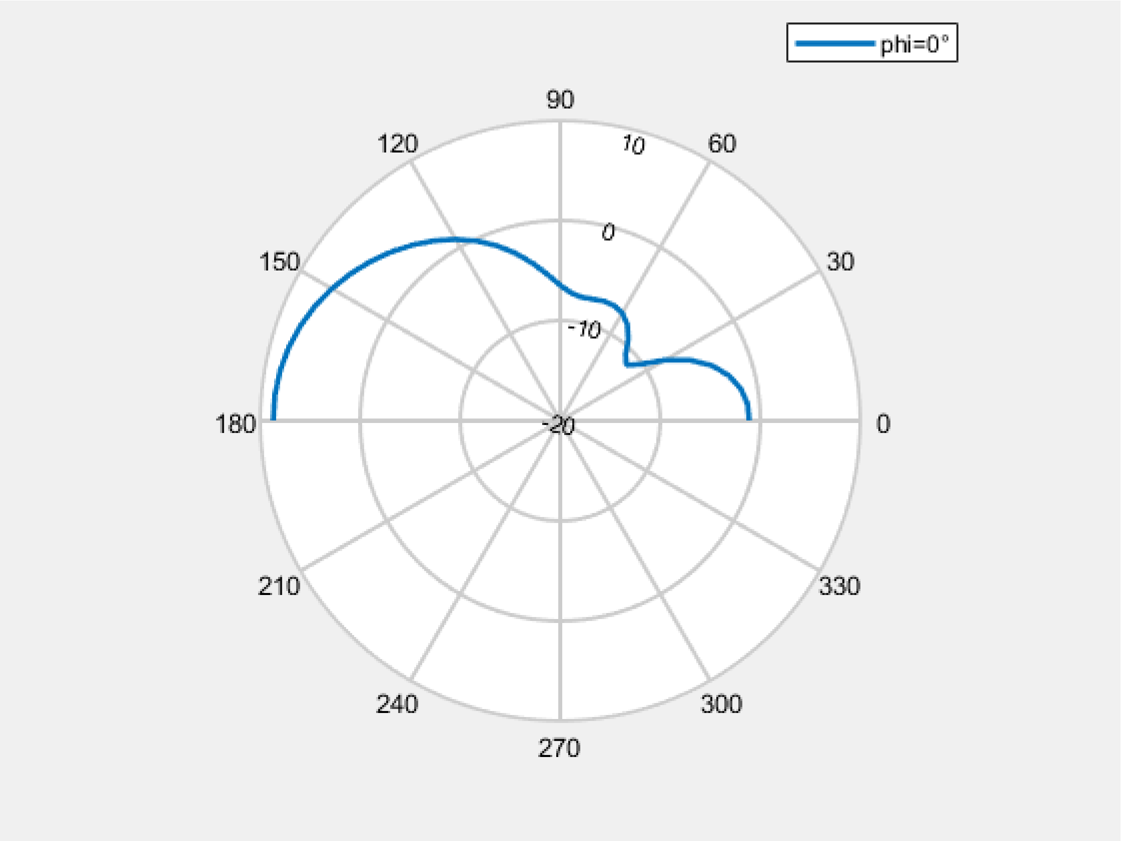
Plot 2-D phi slice of the antenna in rectangular coordinates.

```
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);
```

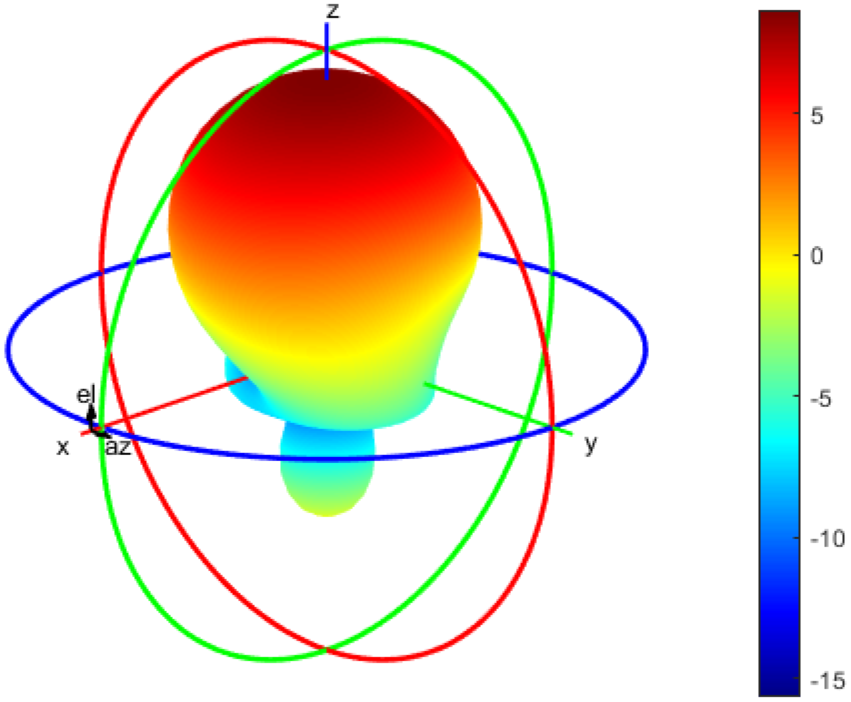


Consider a helix antenna data file in .csv format. This file contains the magnitude of the antenna directivity in phi and theta angles. Read the file .

```
helixdata = csvread('antennadata_test.csv',1,0);
```

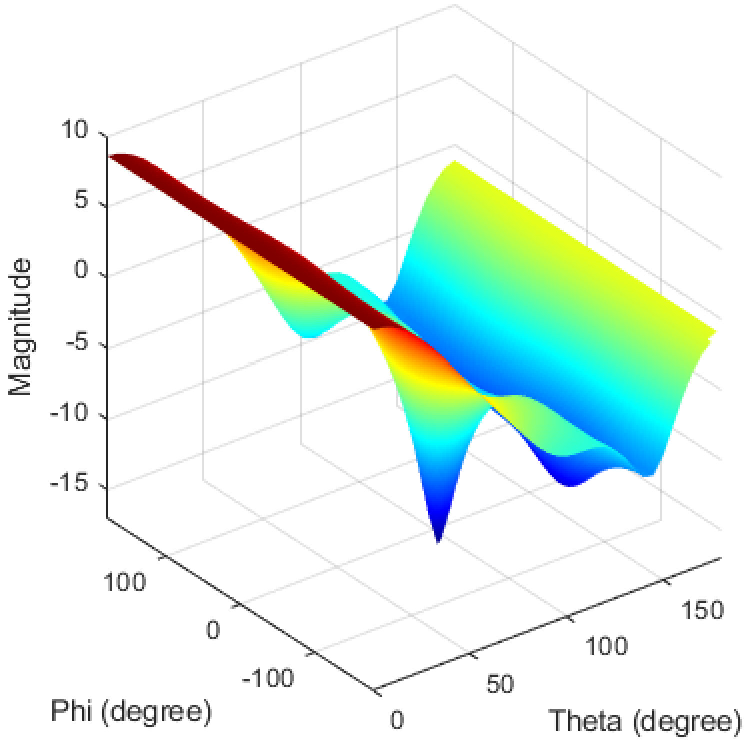
Use patternCustom to extract the magnitude of directivity, and the phi, and theta angle values. Plot the 3-D polar radiation pattern.

```
patternCustom(helixdata(:,3),helixdata(:,2),helixdata(:,1));
```



Use the same data to plot the 3-D rectangular radiation pattern.

```
figure
patternCustom(helixdata(:,3),helixdata(:,2),helixdata(:,1),...
 'CoordinateSystem','rectangular');
```



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 `phixtheta`.

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.

Data Types: `double`

phi — Phi angles in spherical coordinates

vector in degrees

Phi angles in spherical coordinates, specified as a vector in degrees.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside 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'`

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.

Output Arguments

`hplot` — Lines or surfaces in figure window
object handle

Lines or surfaces in figure window, returned as object handle.

See Also
`EHfields` | `fieldsCustom` | `pattern` | `polarpattern`

Introduced in R2016a

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.

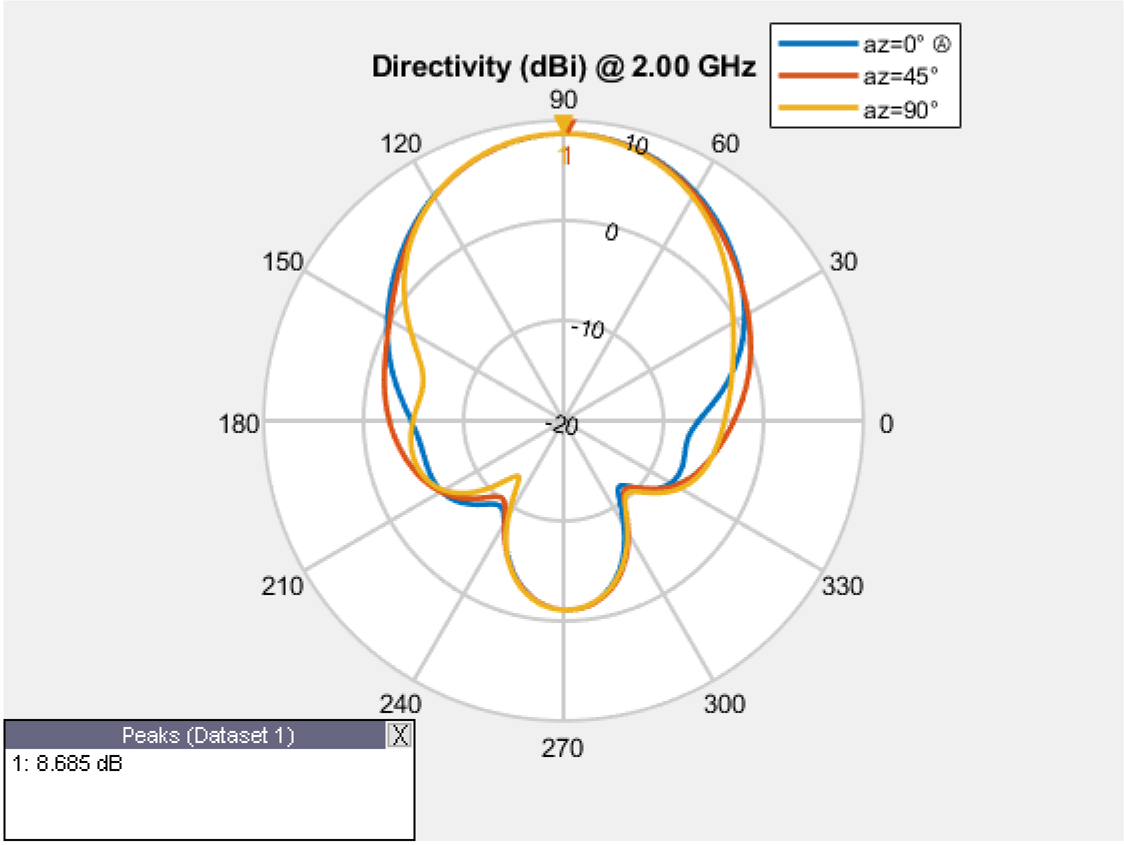
`[horizontal,vertical,optional] = msiread(fname)` reads the file and returns structures containing horizontal gain data, vertical gain data, and all additional data in the file.

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`.

See Also

`msiwrite`

Topics

"Read, Visualize and Write MSI Planet Antenna Files"

Introduced in R2016a

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`.

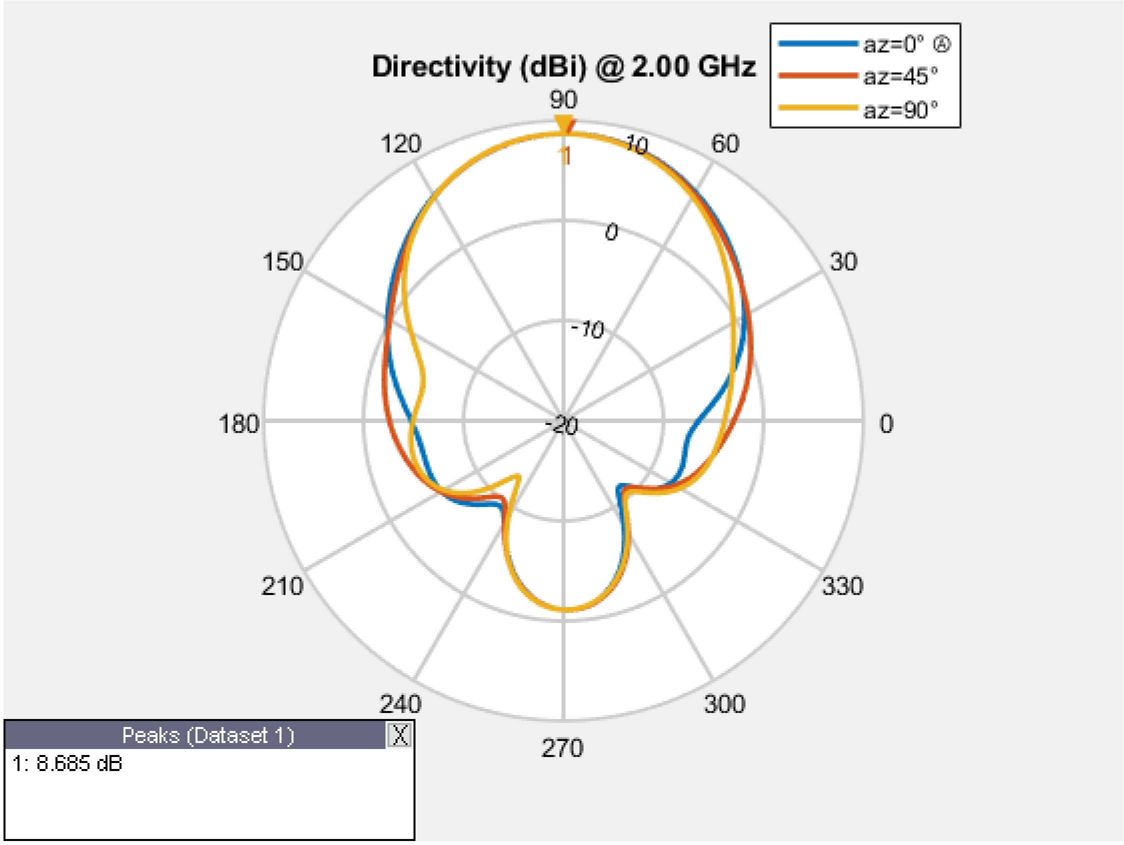
`msiwrite(objname,frequency,fname,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

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 handle

Antenna or array object, specified as an antenna or array handle.

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

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`

See Also

`msiread`

Topics

"Read, Visualize and Write MSI Planet Antenna Files"

Introduced in R2016a

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')
```

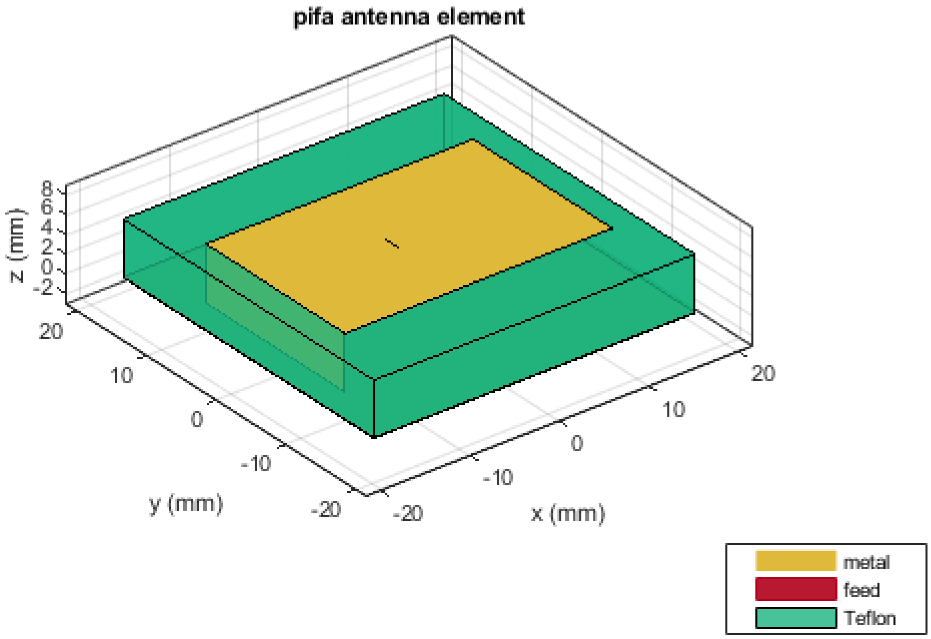
```
d = 
  dielectric with properties:

           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]
                 Tilt: 0
             TiltAxis: [1 0 0]
```

```
                    Load: [1x1 lumpedElement]
```

```
show(p)
```



Custom Dielectric Properties

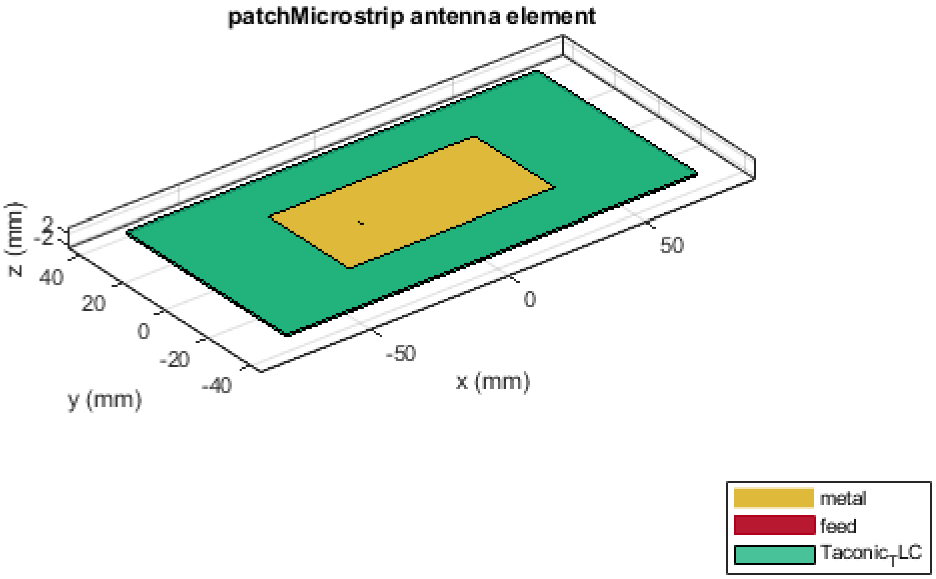
Create a patch microstrip antenna using a substrate with a relative permittivity of 2.70, a loss tangent of 0.002 and a thickness of 0.0008 m. View the antenna.

```
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]
```

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

```
show(p)
```



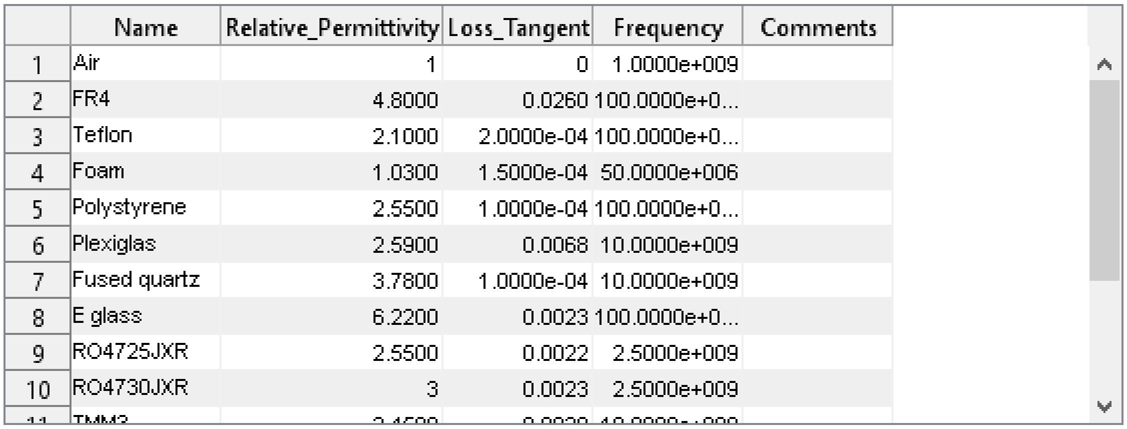
Patch Antenna with Air Gap between Groundplane and Dielectric

Create a microstrip patch antenna.

```
p = patchMicrostrip;
```

For properties of air and teflon dielectrics use Dielectric Catalog.

```
openDielectricCatalog
```



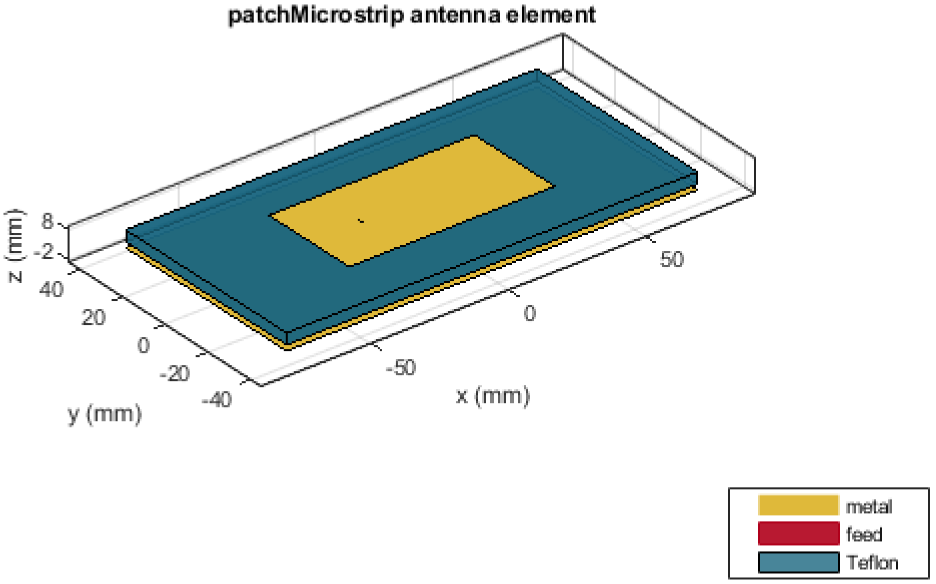
	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+009	
2	FR4	4.8000	0.0260	100.0000e+0...	
3	Teflon	2.1000	2.0000e-04	100.0000e+0...	
4	Foam	1.0300	1.5000e-04	50.0000e+006	
5	Polystyrene	2.5500	1.0000e-04	100.0000e+0...	
6	Plexiglas	2.5900	0.0068	10.0000e+009	
7	Fused quartz	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)
```



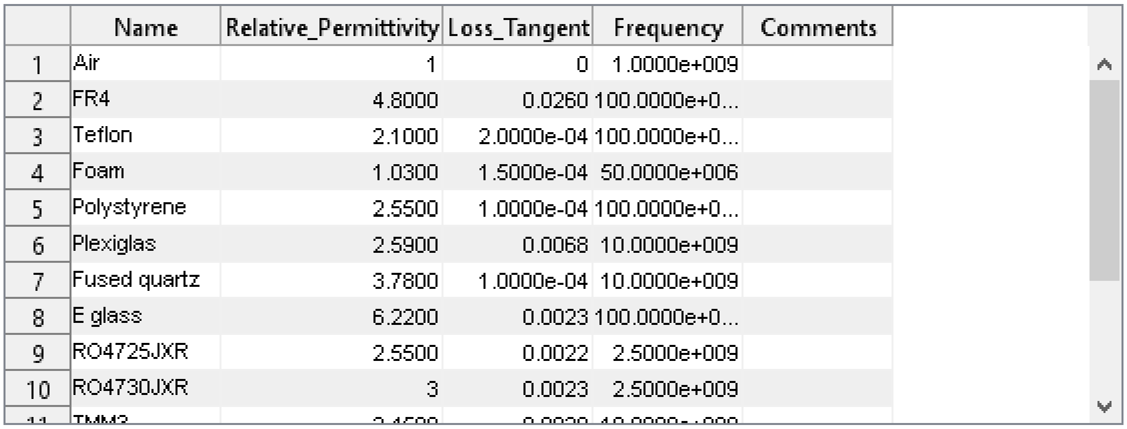
Three Layer Dielectric Substrate between Patch and Ground Plane

Create a microstrip patch antenna.

```
p = patchMicrostrip;
```

For dielectric properties, use the Dielectric Catalog.

```
openDielectricCatalog
```



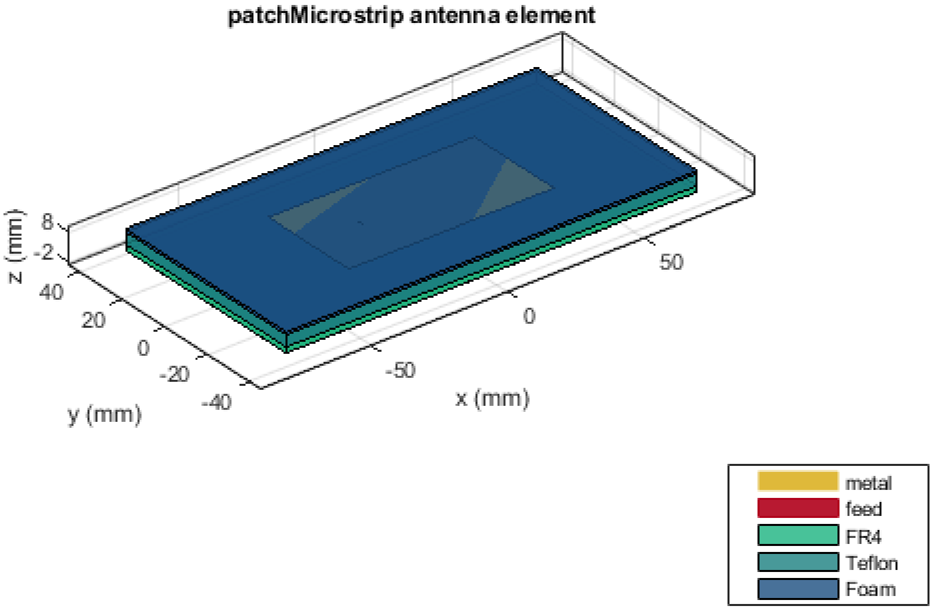
	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+009	
2	FR4	4.8000	0.0260	100.0000e+0...	
3	Teflon	2.1000	2.0000e-04	100.0000e+0...	
4	Foam	1.0300	1.5000e-04	50.0000e+006	
5	Polystyrene	2.5500	1.0000e-04	100.0000e+0...	
6	Plexiglas	2.5900	0.0068	10.0000e+009	
7	Fused quartz	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]);
```

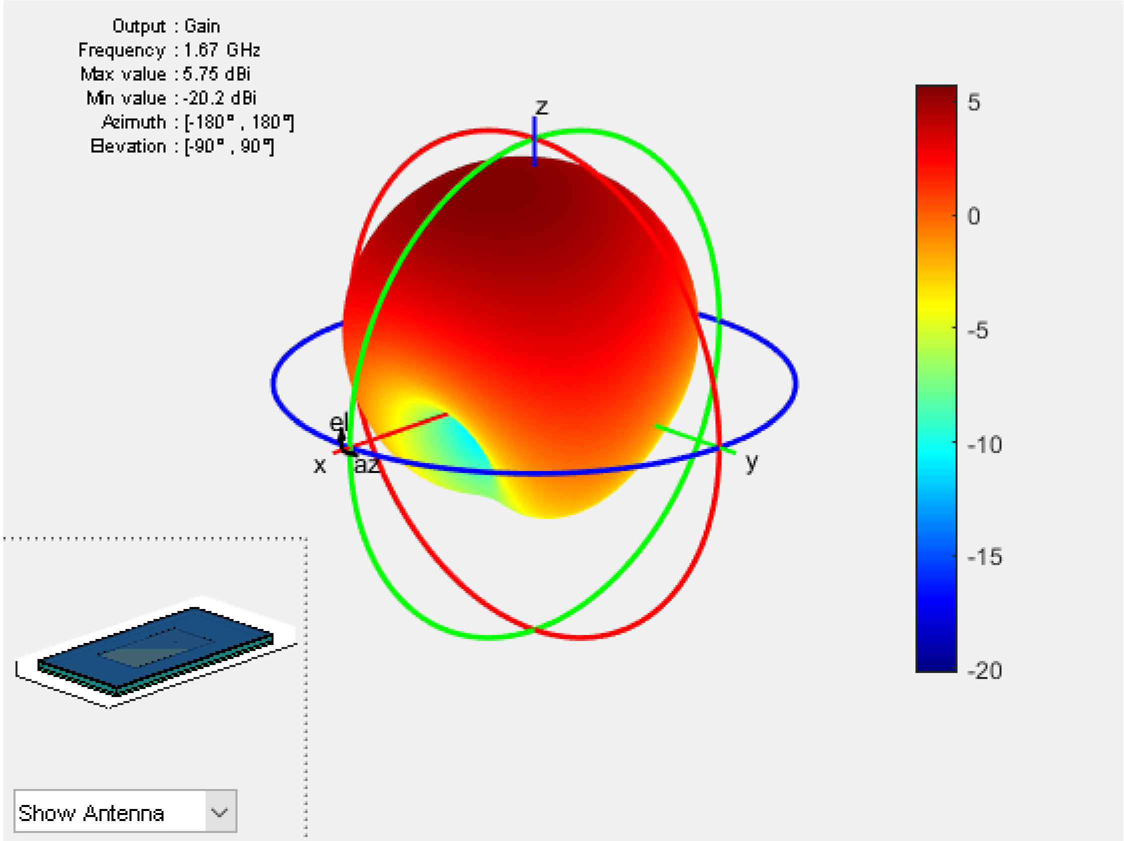
Add the three layer substrate to the patch antenna.

```
p.Substrate = sub;
figure
show(p)
```



Plot the radiation pattern of the antenna.

```
figure
pattern(p,1.67e9)
```



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.

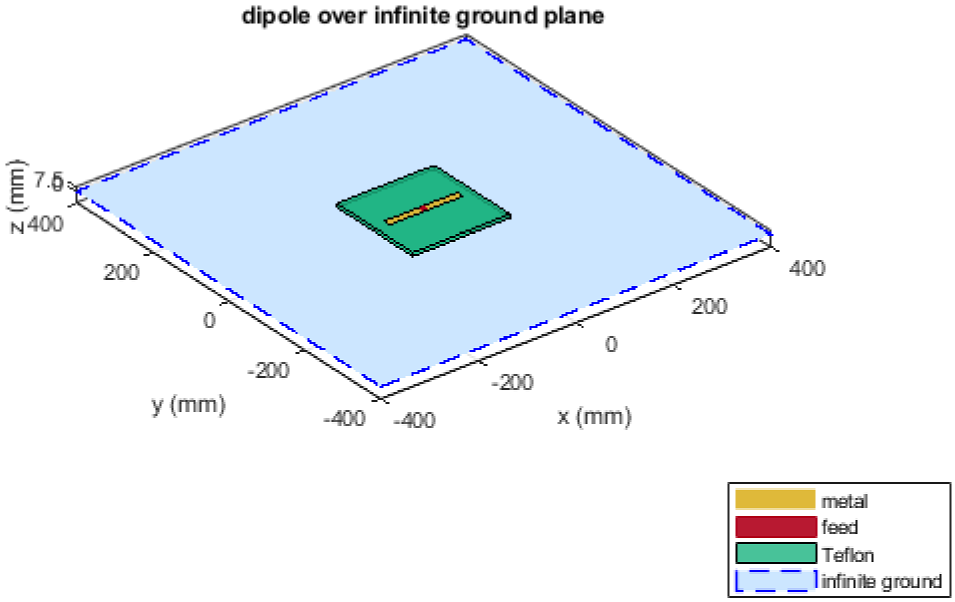
```
t = dielectric('Teflon')
```

```
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);
```

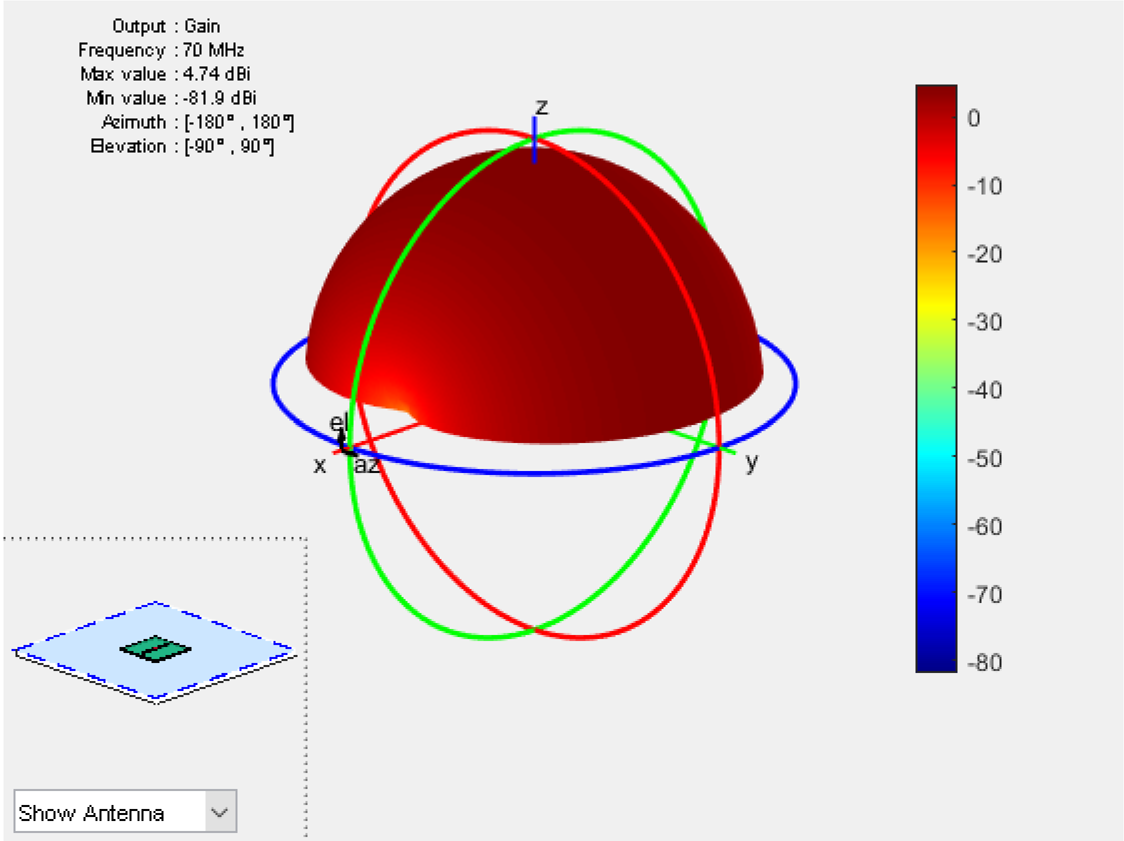
Set the groundplane length of the reflector to `inf`. View the structure.

```
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 : 4.74 dBi
Min value : -81.9 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

Antenna On Dielectric Substrate - Compare Gain Values

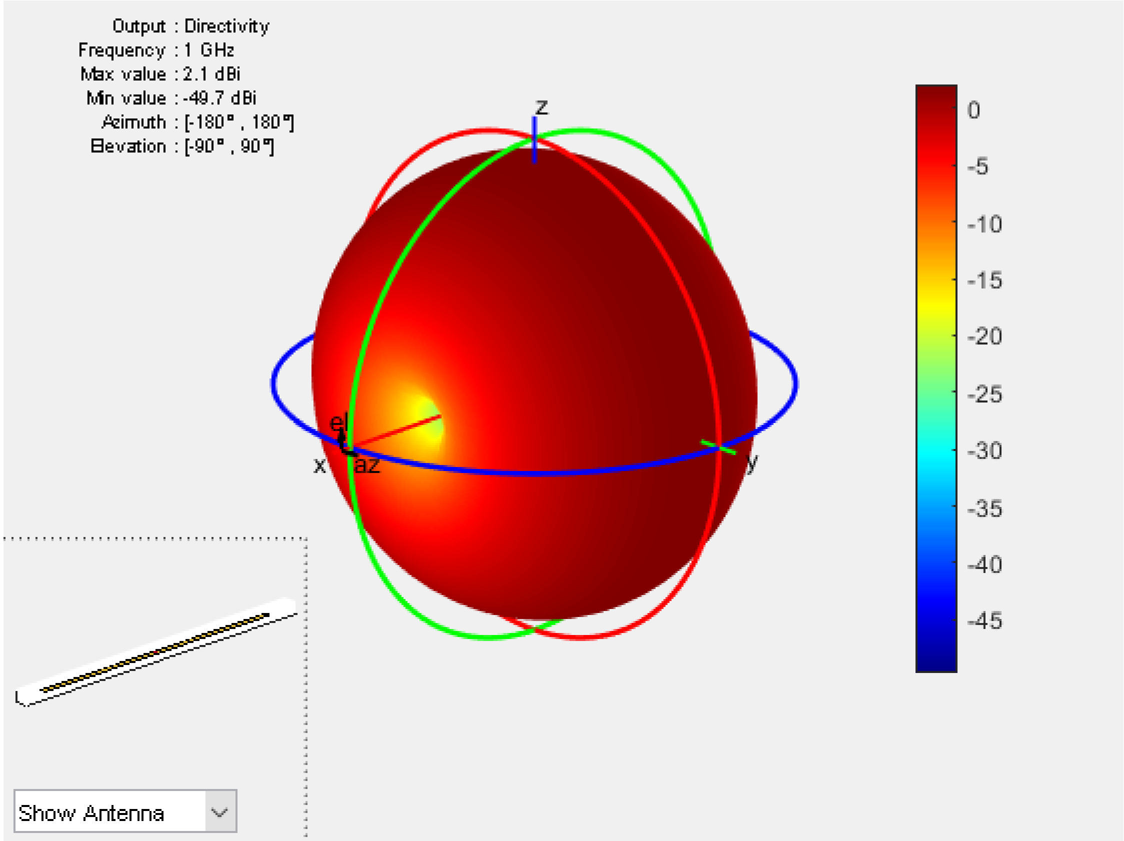
Compare the gain values of a dipole antenna in free space and dipole antenna on a substrate.

Design a dipole antenna at 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 1GHz.

```
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);
```

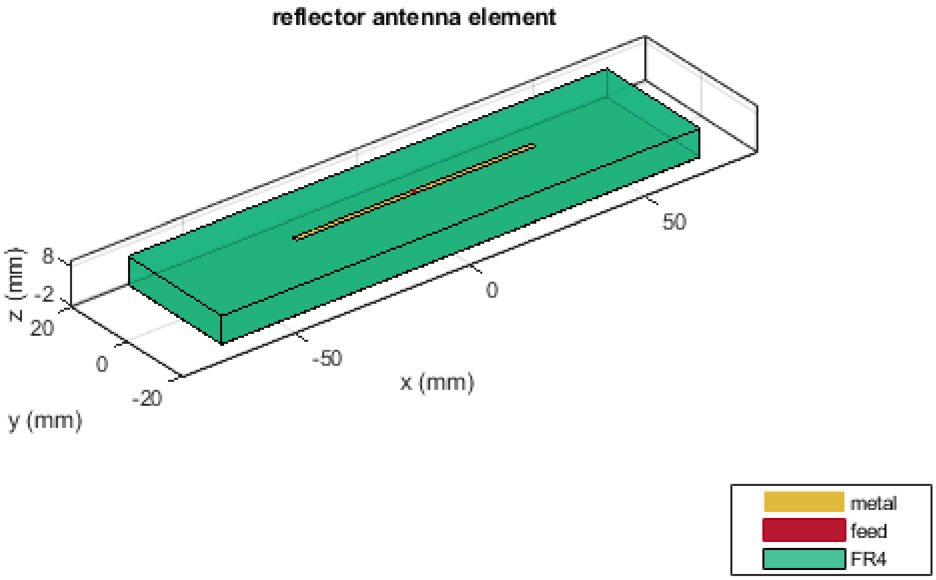
Adjust the length of the dipole based on the wavelength.

```
d.Length = lambda_d/2;
d.Width = d.Length/l_by_w;
```

Design a reflector at 1 GHz with the dipole as the excitor and FR4 as the substrate.

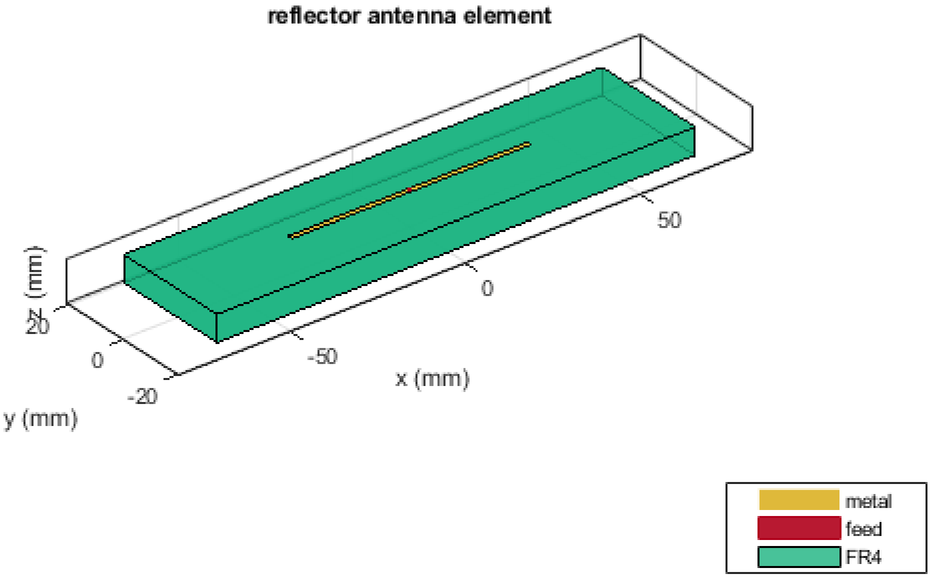
```
rf = design(reflector,1e9);
rf = reflector('Exciter',d,'Spacing',7.5e-3,'Substrate',t);
```

```
rf.GroundPlaneLength = lambda_d;
rf.GroundPlaneWidth = lambda_d/4;
figure
show(rf)
```



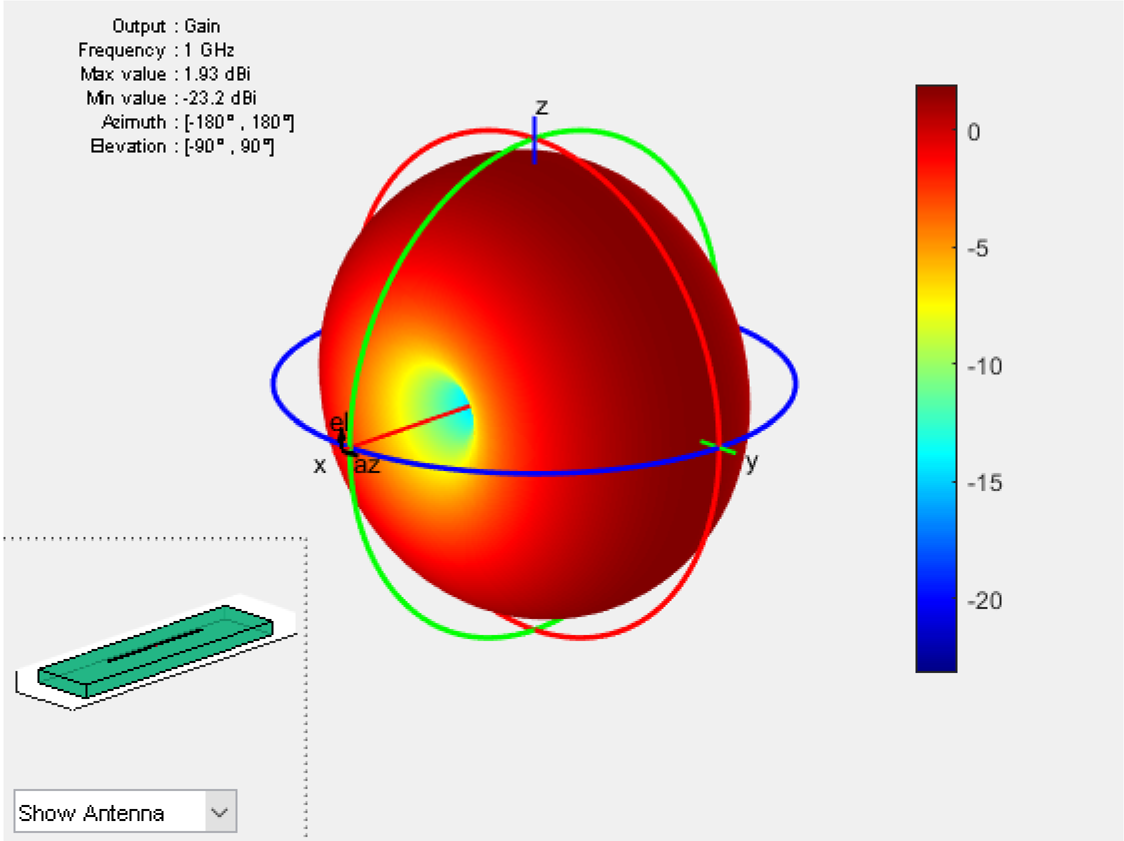
Remove the groundplane for plotting the gain of the dipole on the substrate.

```
rf.GroundPlaneLength = 0;
show(rf)
```



Plot the radiation pattern of the dipole on the substrate at 1 GHz.

```
figure
pattern(rf,1e9);
```



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)

Material from the dielectric catalog, specified as one of the values from the `DielectricCatalog`.

Example: `'FR4'`

Data Types: `char`

Name-Value Pair Arguments

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

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.

Example: `'LossTangent'`,`0.0260`

Data Types: `double`

Note In Antenna Toolbox, the upper limit to loss tangent value is 0.03.

`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 handle

Dielectric material, returned as an object handle. You can use the dielectric material object handle to add dielectric material to an antenna.

See Also

`DielectricCatalog`

Topics
“Antenna Toolbox Limitations”

Introduced in R2016a

DielectricCatalog

Catalog of dielectric materials

Syntax

```
dc = DielectricCatalog
```

Description

`dc = DielectricCatalog` creates an object handle for the dielectric catalog.

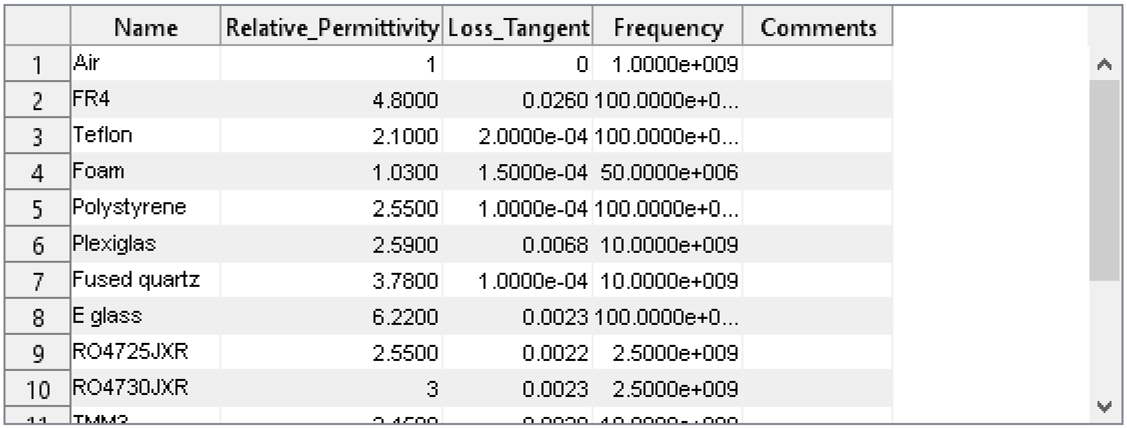
- To open the dielectric catalog, use `open(dc)`
- To know the properties of a dielectric material from the dielectric catalog, use `s = find(dc, name)`.

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')
```

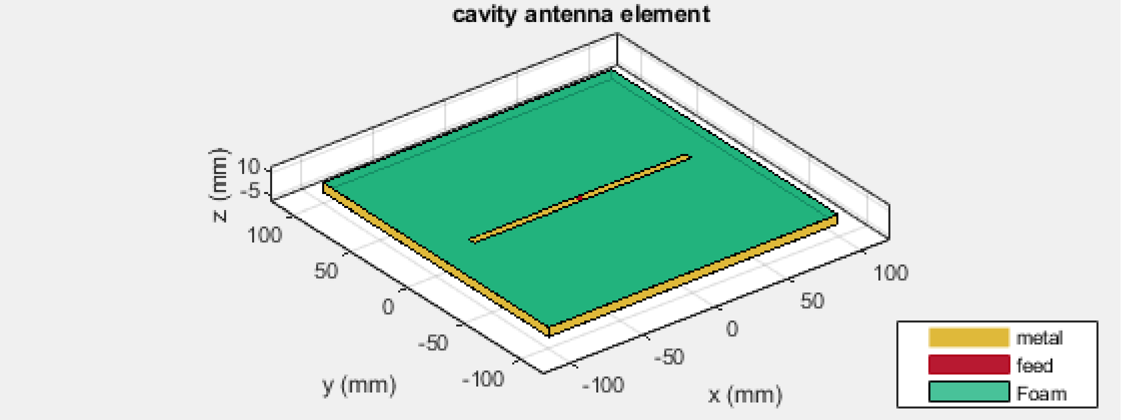
```
s = struct with fields:
                     Name: 'Foam'
    Relative_Permittivity: 1.0300
             Loss_Tangent: 1.5000e-04
                Frequency: 50000000
                 Comments: ''
```

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

```
show (c)
```



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 handle

Dielectric catalog, specified as an object handle.

Data Types: char

Output Arguments

dc — Dielectric catalog

object handle

Dielectric catalog, returned as an object handle.

s — Parameters of dielectric material

structure

Parameters of a dielectric material from the dielectric catalog, returned as a structure.

See Also

`dielectric`

Introduced in R2016a

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 as the comma-separated pair consisting of `'Width'` and a scalar in meters.

Data Types: `double`

height — Rectangular waveguide height
scalar in meters

Rectangular waveguide height, specified as the comma-separated pair consisting of `'Height'` and a scalar in meters.

Data Types: `double`

flarelength — Flare length of horn
scalar in meters

Flare length of horn, specified as the comma-separated pair consisting of `'FlareLength'` and 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`

See Also

`horn`

Introduced in R2016a

add

Class: `polarpattern`

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

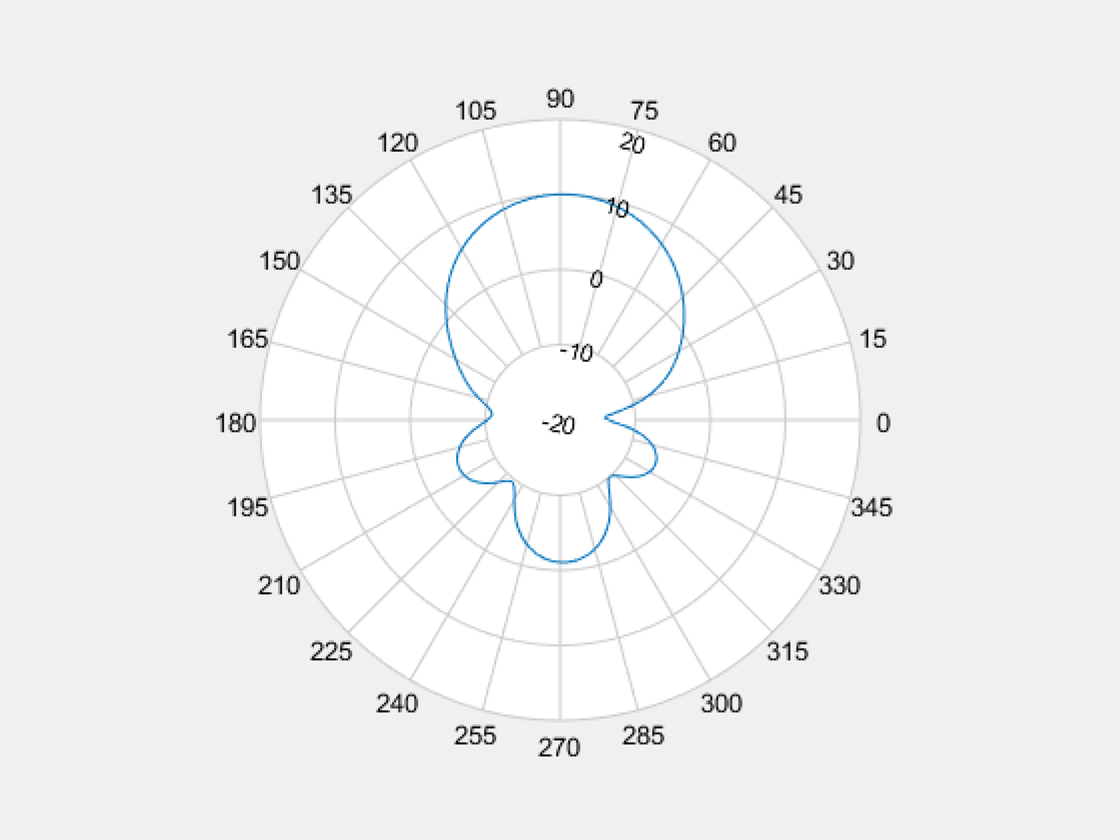
Add Data To Polar Plot

Create a helix antenna that has 28 mm radius, 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);
```

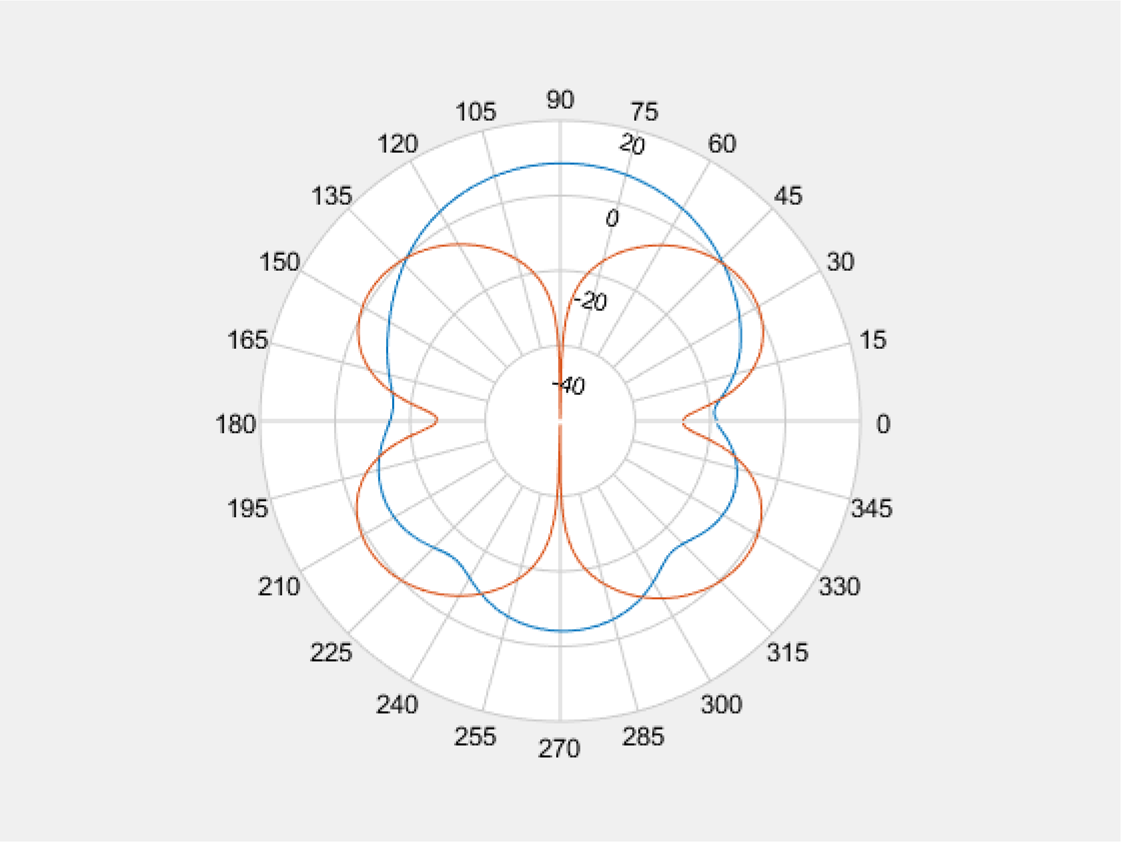


Create a dipole antenna and calculate the directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

Add the directivity of the dipole to the existing polar plot of helix antenna.

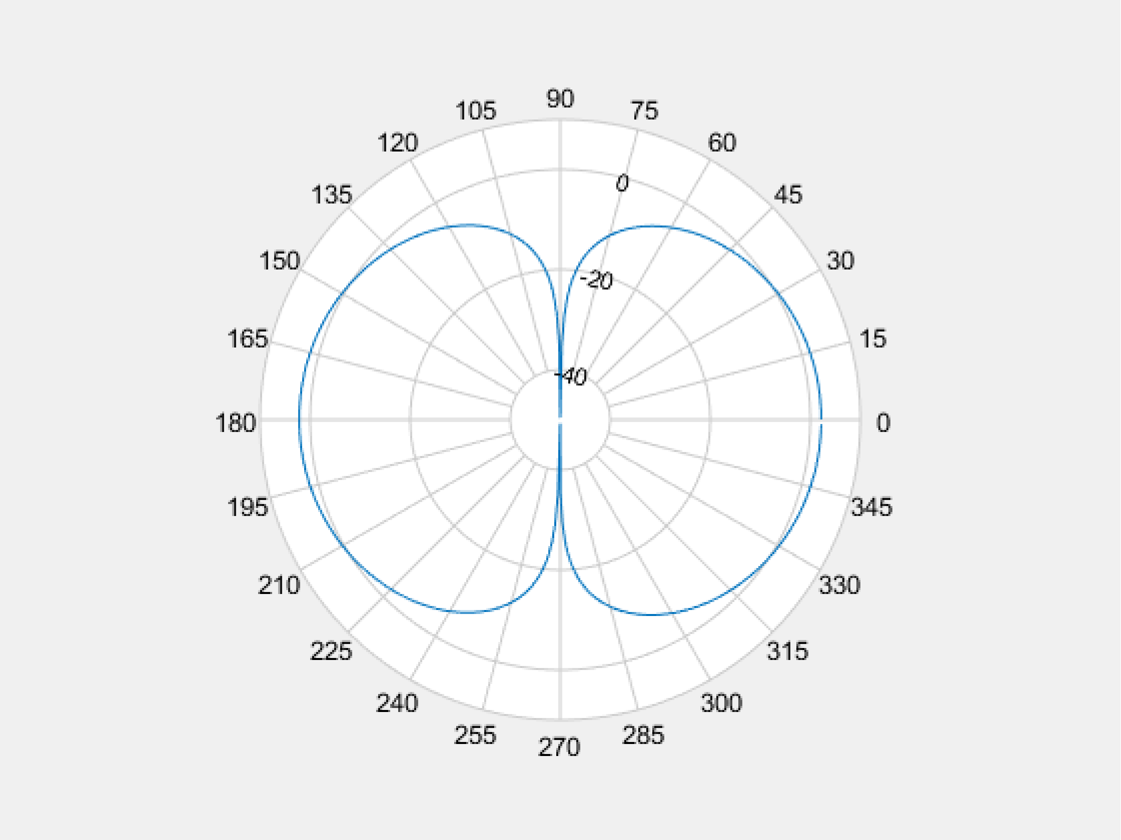
```
add(P,D);
```



Add Angle and Magnitude Data to Polar Pattern

Create a dipole and plot the polar pattern of its directivity at 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.

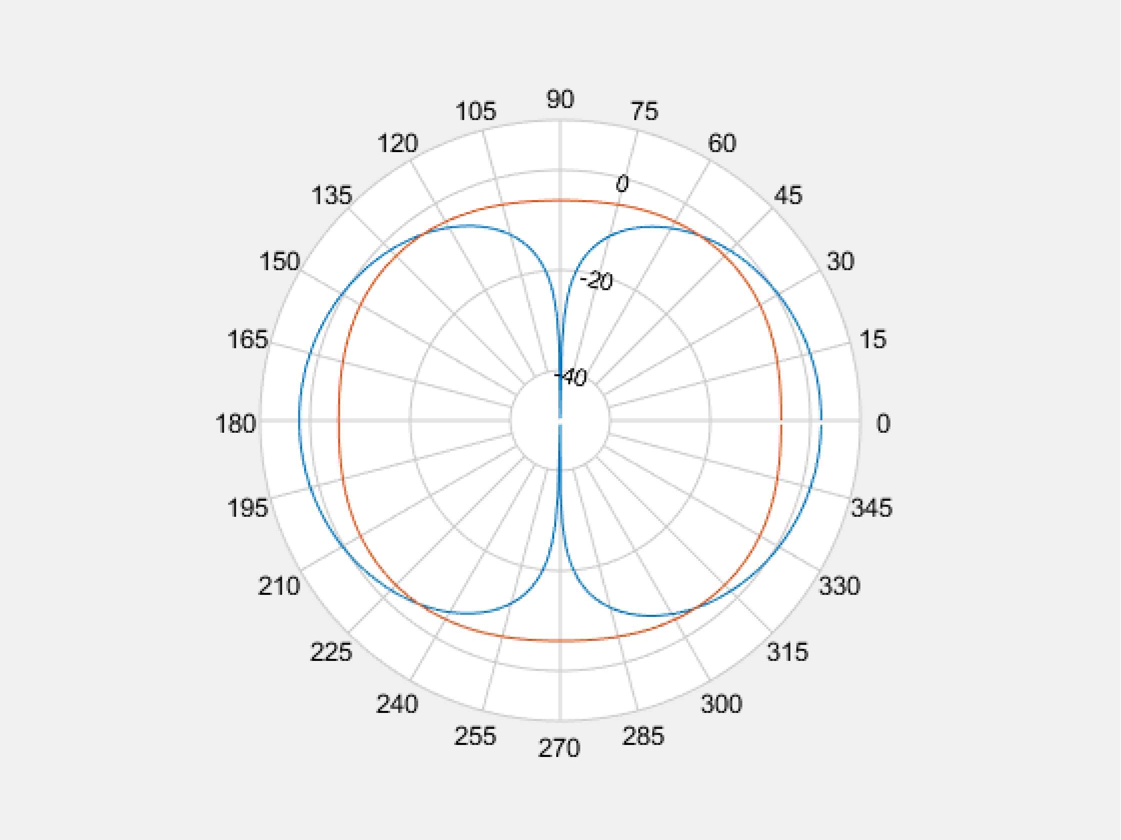
```
[Horizontal,Vertical,Optional] = msiread('cavity.pln')
```

```
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);
```



See Also

addCursor | animate | createLabels | findLobes | replace | showPeaksTable | showSpan

Introduced in R2016a

addCursor

Class: `polarpattern`

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

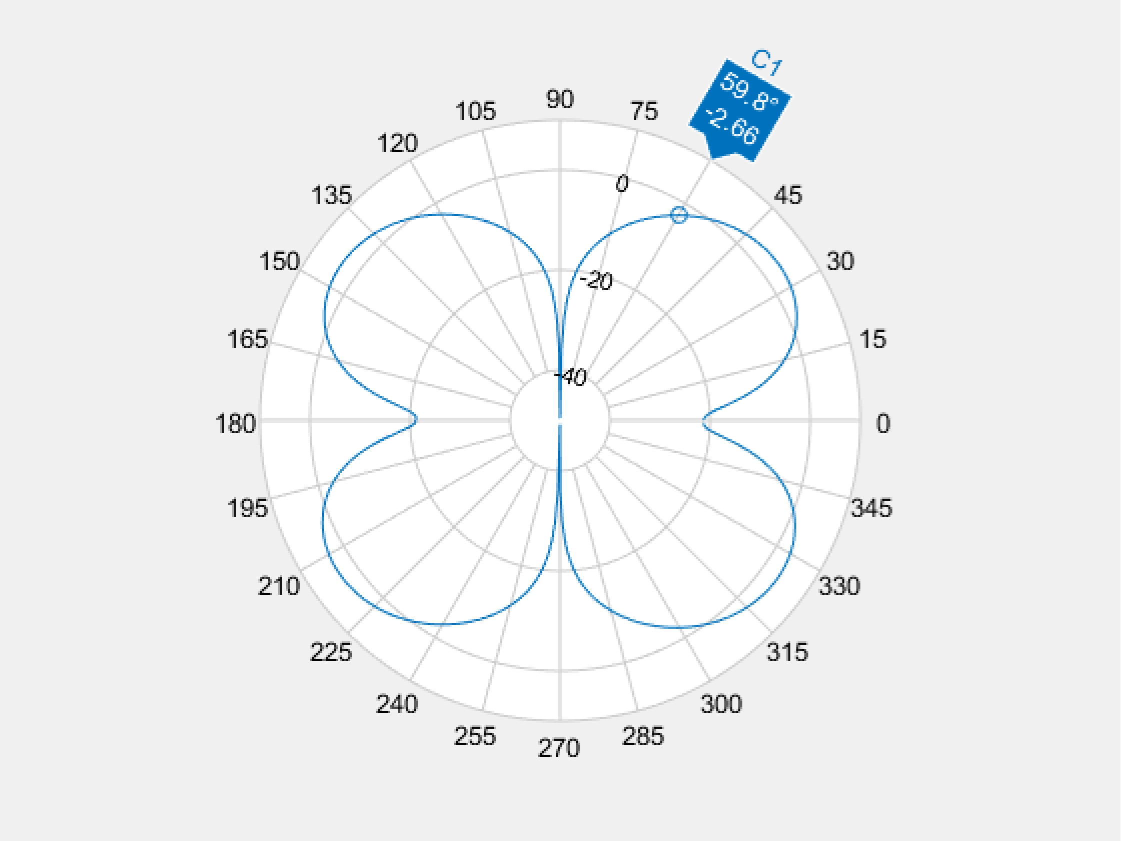
Add Cursor to Plot

Create a dipole antenna and calculate its directivity at 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.

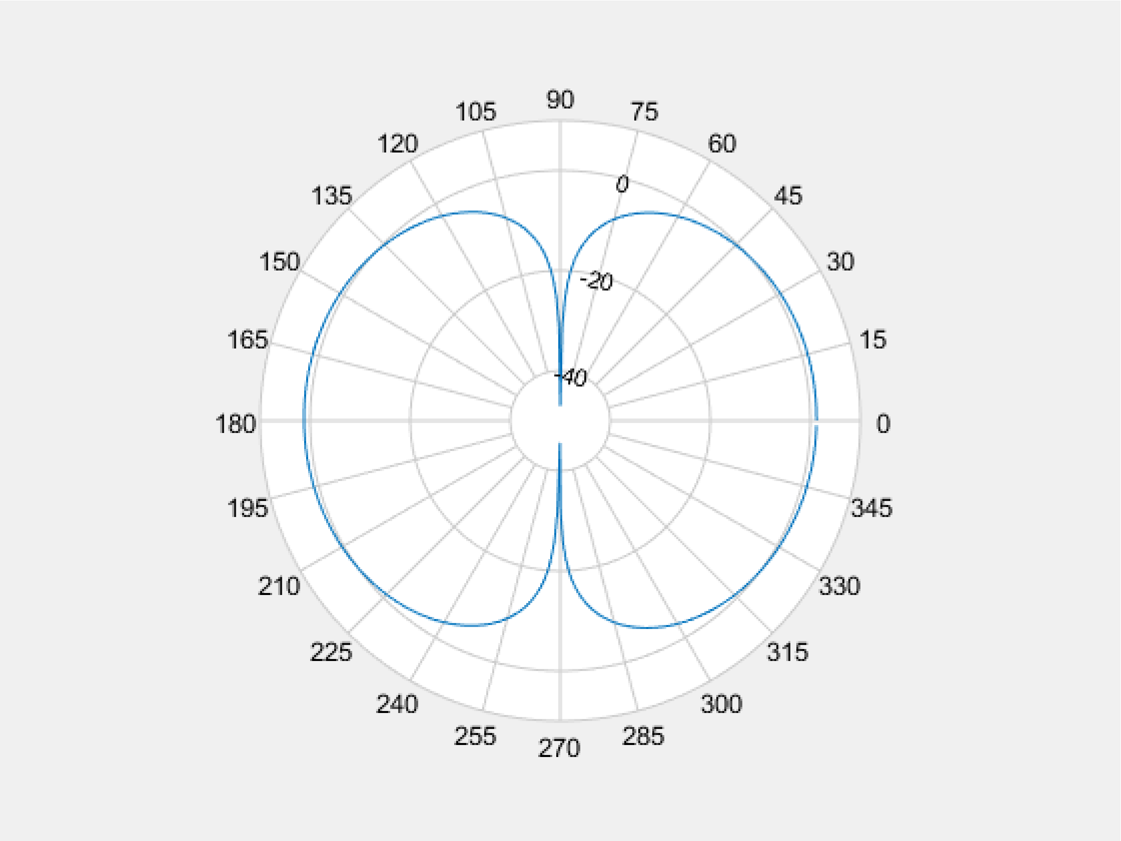
```
p = polarpattern(D);
addCursor(p,60);
```



Add Cursors to Two Data Sets

Create a top-hat monopole and plot its directivity at 75 MHz.

```
m = monopoleTopHat;
M = pattern(m,75e6,0,0:1:360);
P = polarpattern(M);
```

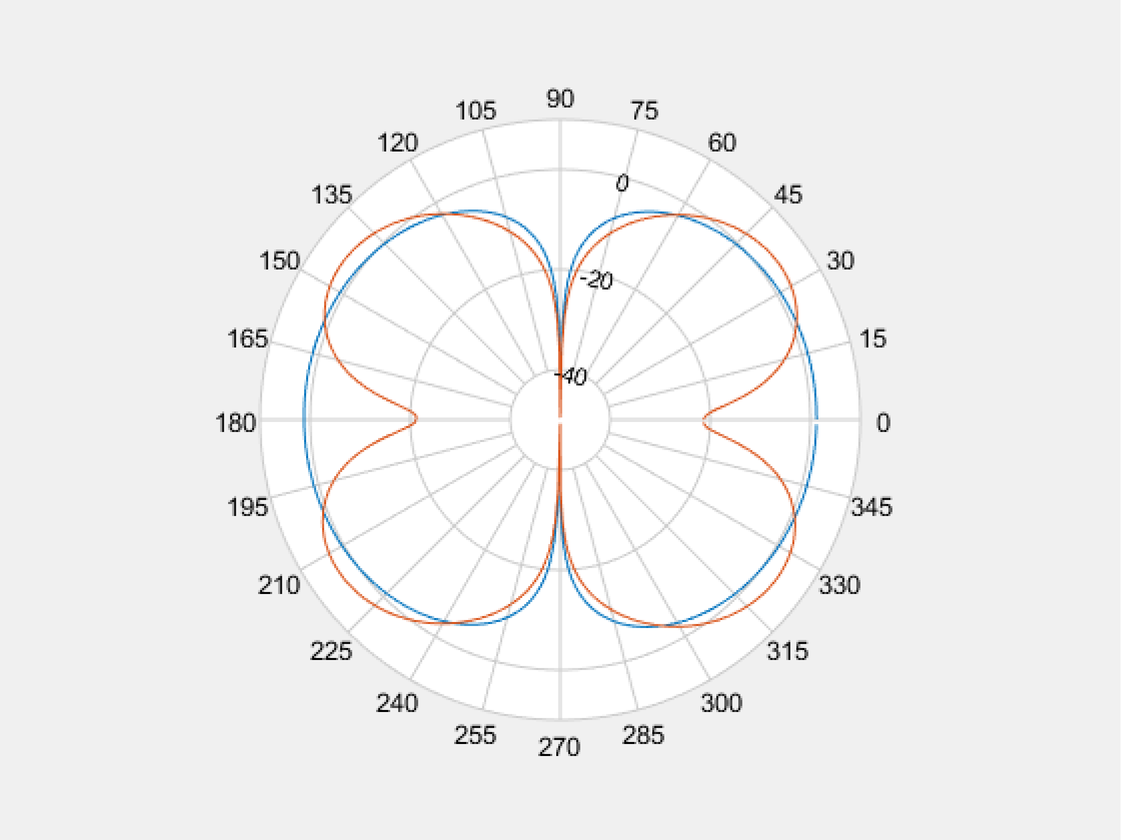


Create a dipole antenna and calculate its directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
```

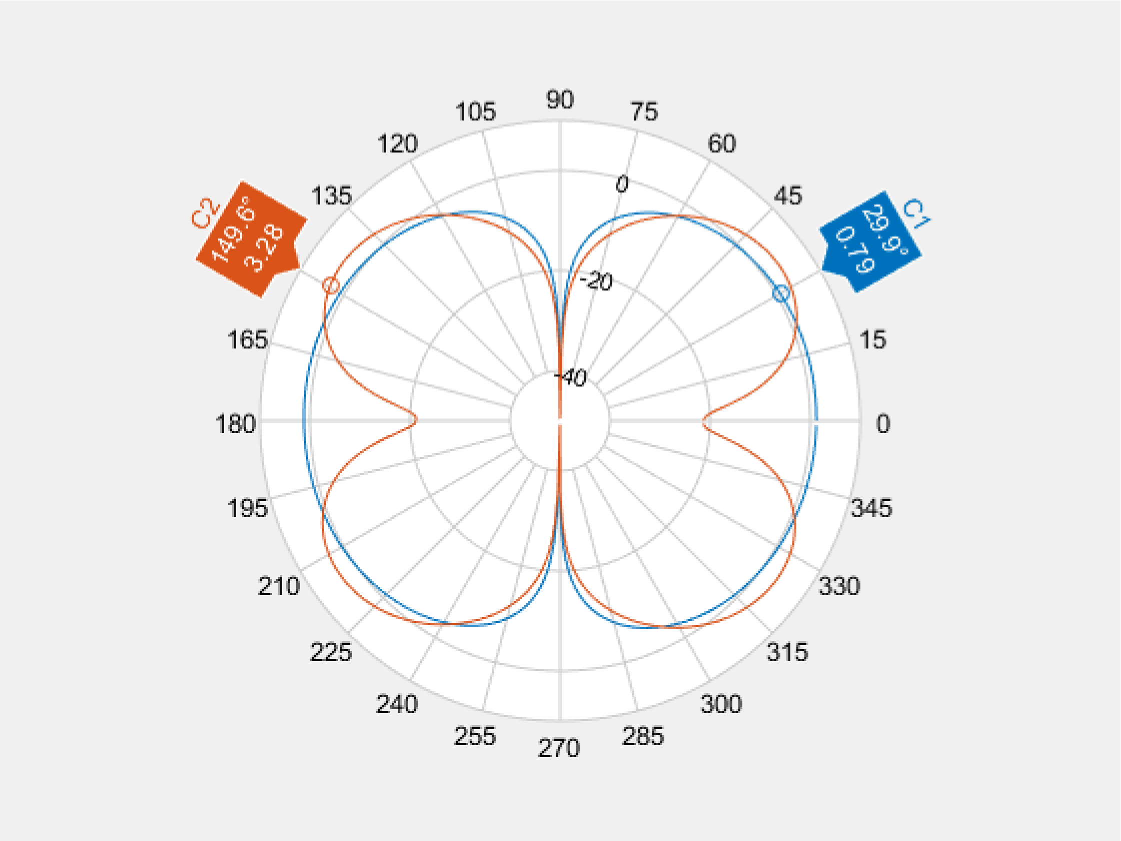
Add the directivity pattern of the dipole to the polar plot of the top-hat monopole.

```
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]);
```



See Also

`add` | `animate` | `createLabels` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

Introduced in R2016a

animate

Class: `polarpattern`

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

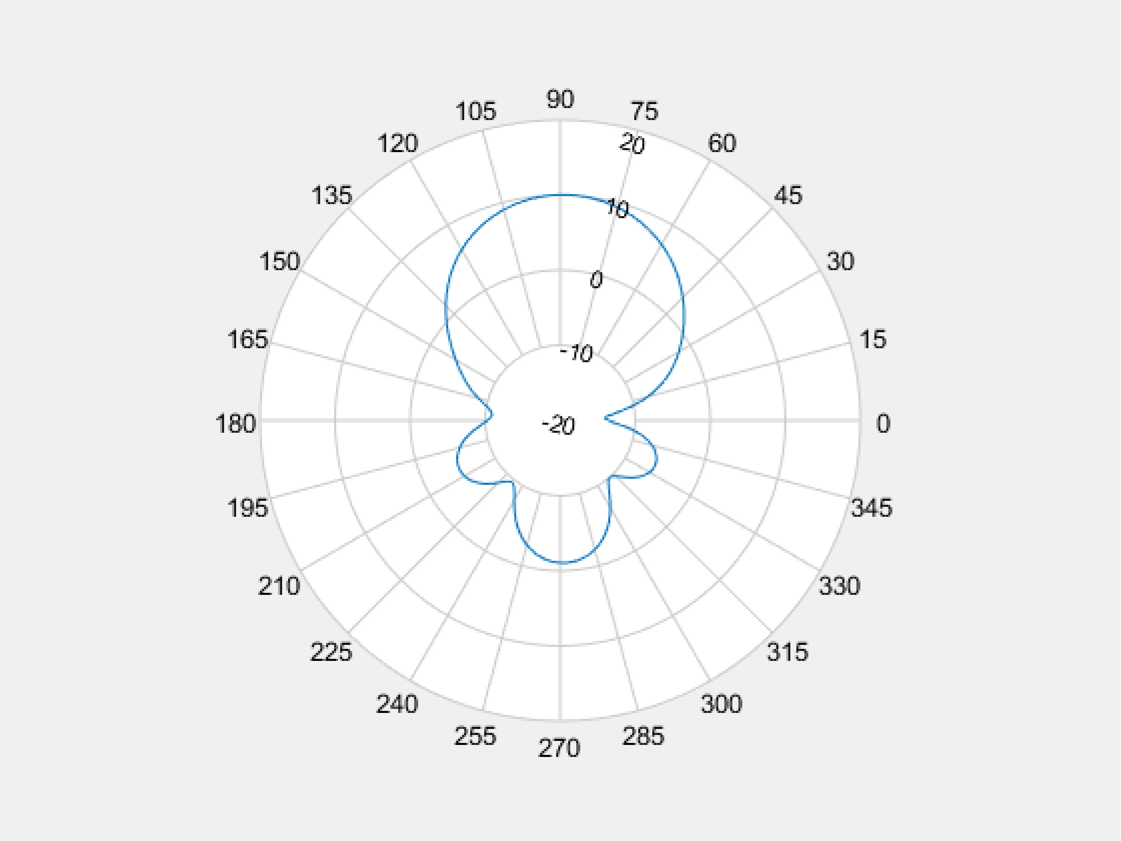
Set of magnitude values, specified as a vector or a matrix. For a matrix of magnitude values, each column is an independent set of magnitude values and corresponds to the same set of angles.

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);
```

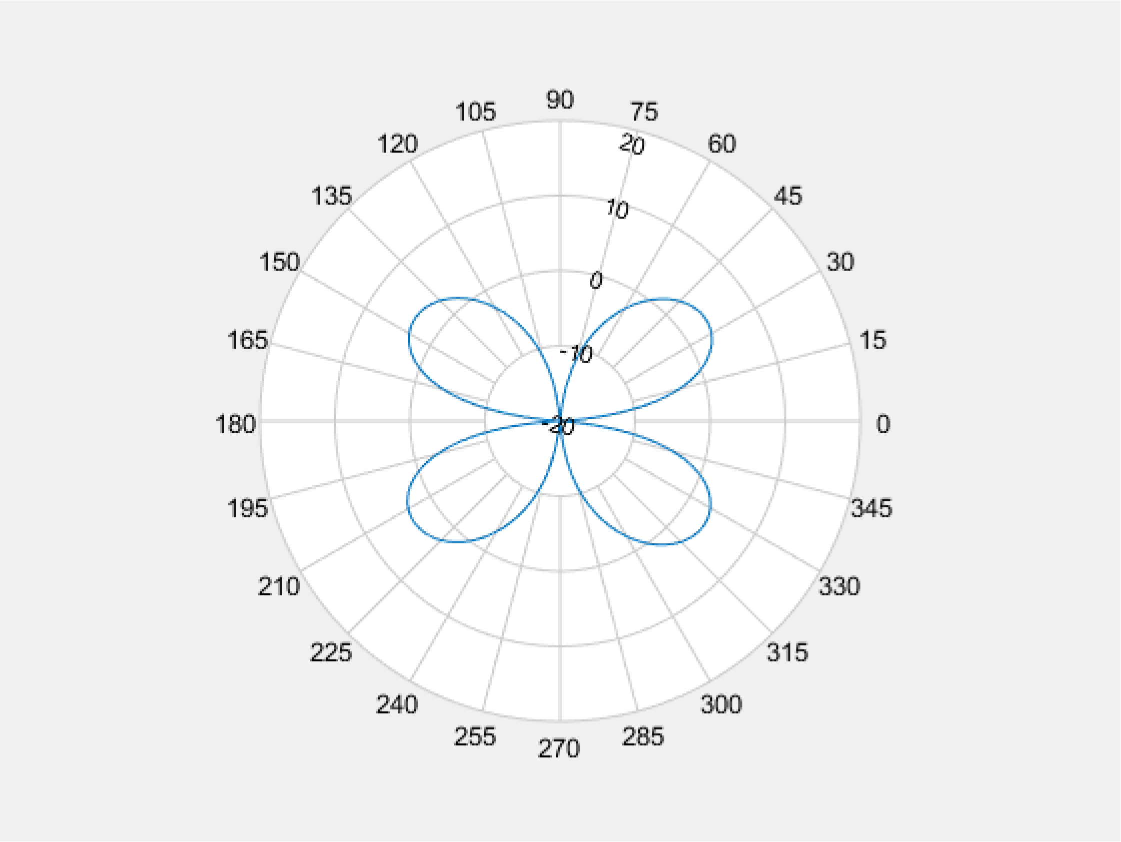


Create a dipole antenna and calculate its directivity at 270 MHz.

```
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.

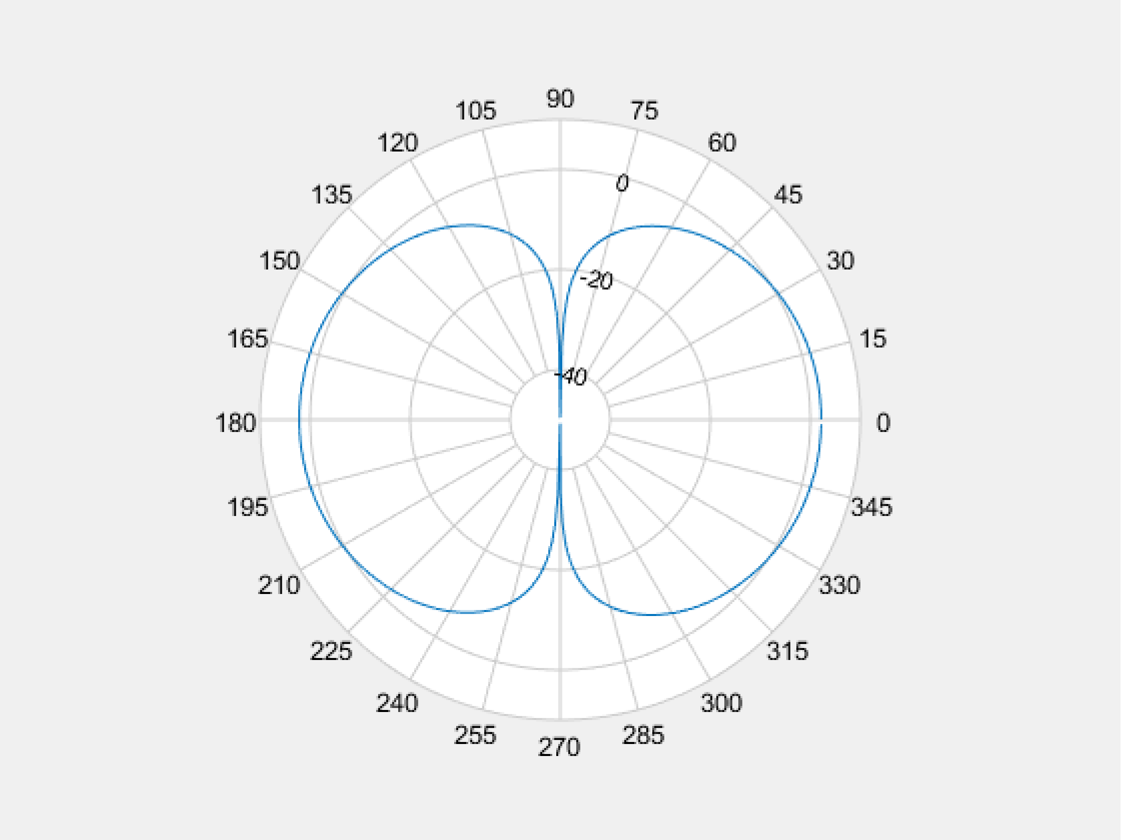
```
animate(P,D);
```



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.

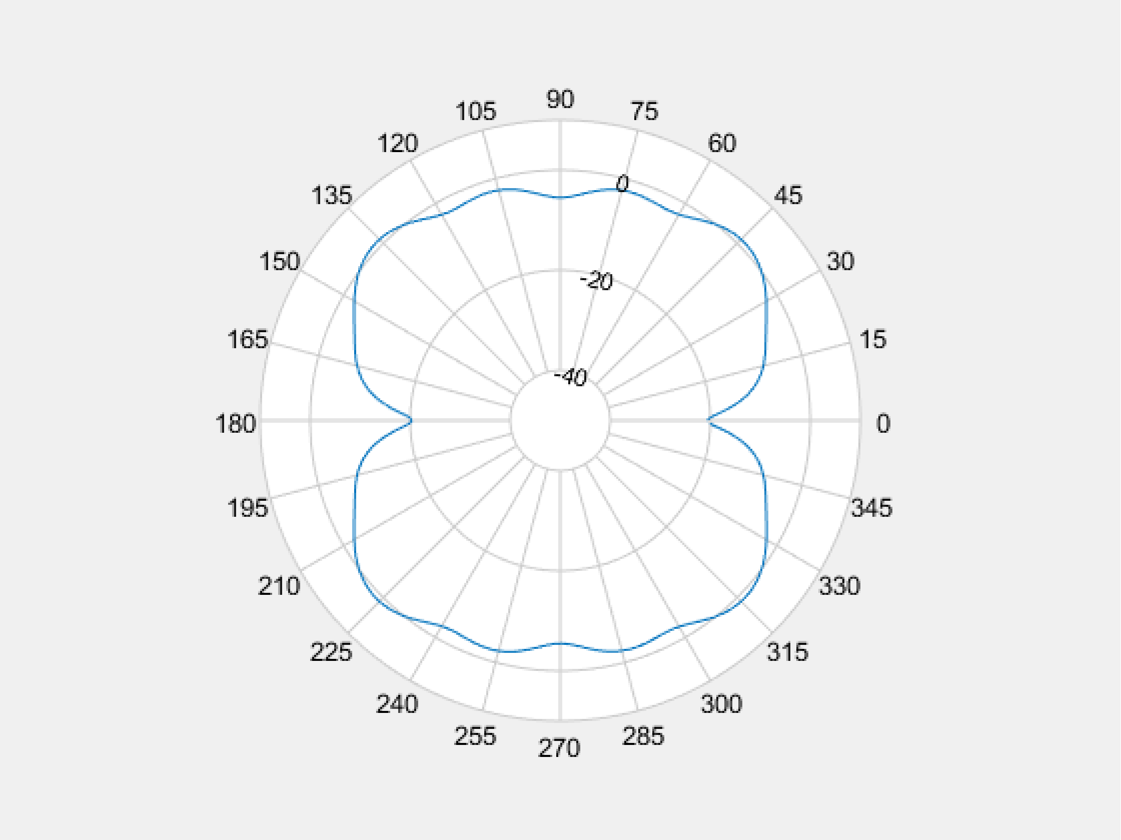
```
[Horizontal,Vertical,optional]= msiread('cavity.pln')
```

```
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);
```



See Also

add | addCursor | createLabels | findLobes | replace | showPeaksTable | showSpan

Introduced in R2016a

createLabels

Class: polarpattern

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

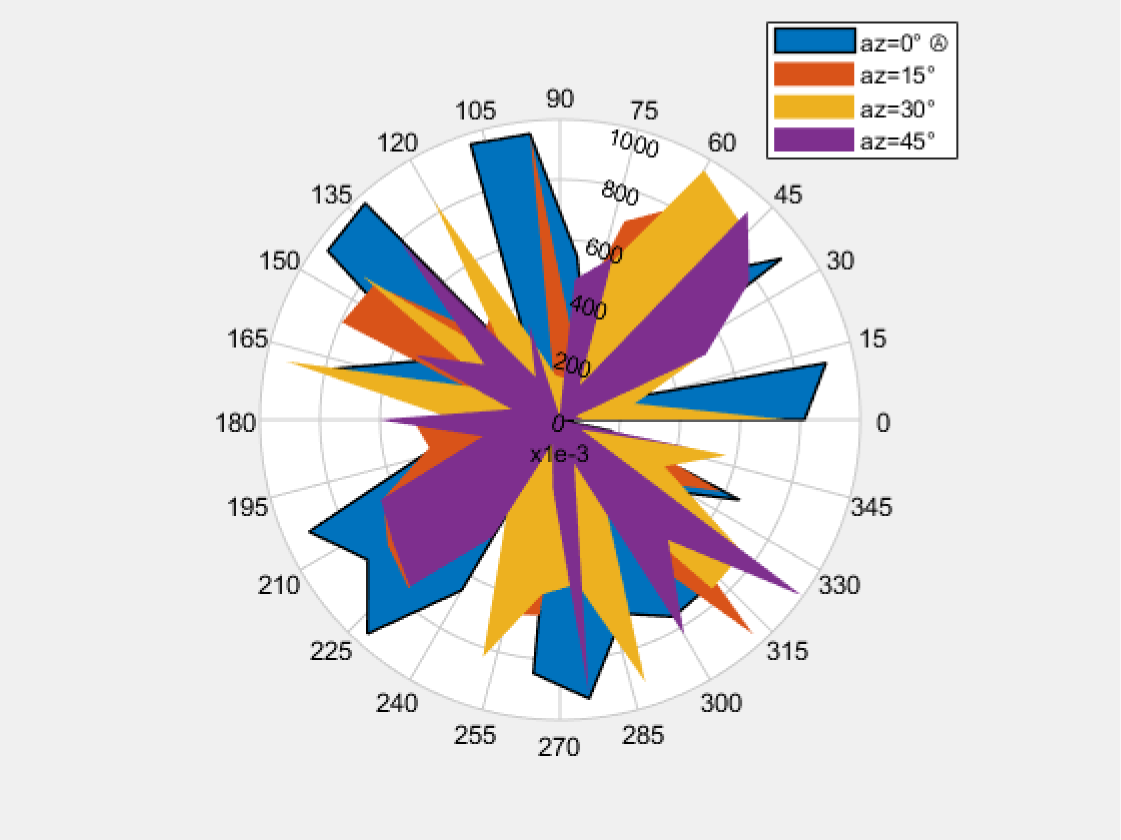
Values to apply to format , specified as an array. The values can be an array of angles or array of magnitude.

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)
```



See Also

`add` | `addCursor` | `animate` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

Introduced in R2016a

findLobes

Class: polarpattern

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.

Examples

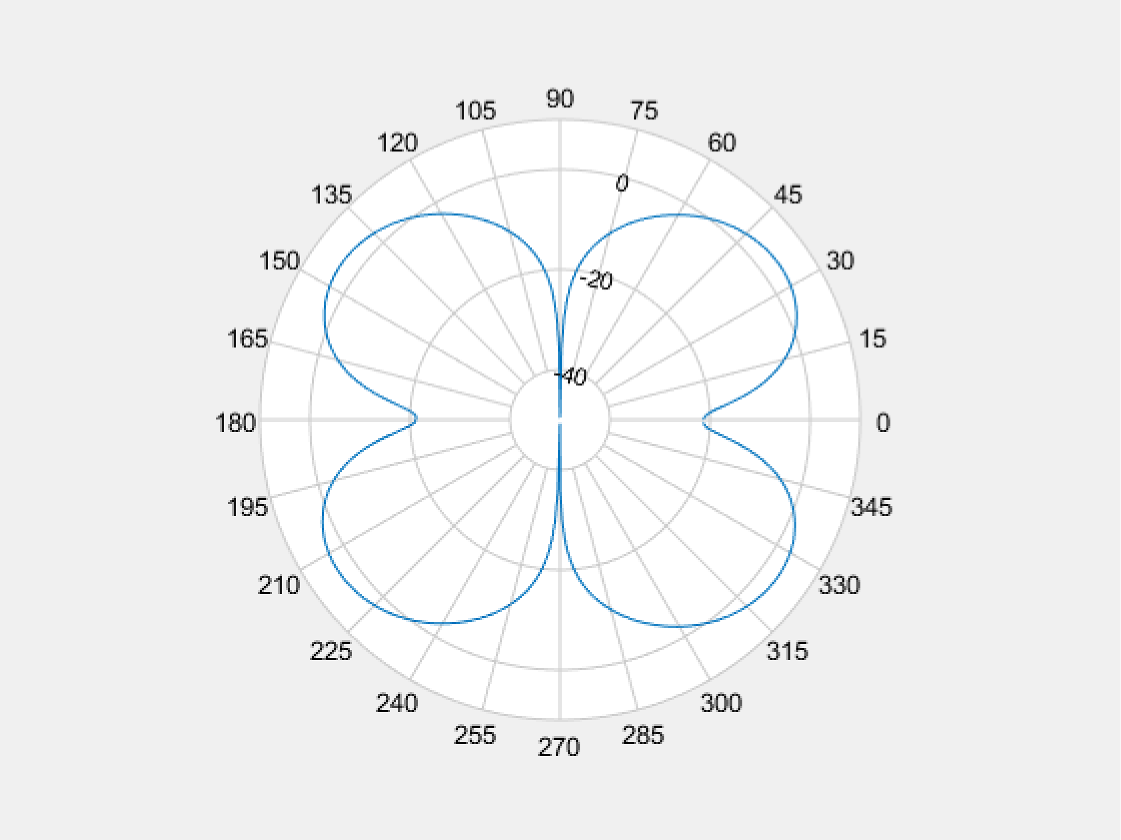
Find Main, Back, and Side Lobes

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: 0.0124
          SLL: 0
         HPBW: 30.9141
         FNBW: 89.7507
        FBIdx: [146 326.5000]
       SLLIdx: [146 36]
      HPBWIdx: [129 160]
      HPBWAng: [127.6454 158.5596]
      FNBWIdx: [91 181]
```

Inspect main, back, and side lobe data.

```
MainLobe = L.mainLobe
```

```
MainLobe = struct with fields:
        index: 146
    magnitude: 3.6675
        angle: 144.5983
       extent: [91 181]
```

```
BackLobe = L.backLobe
BackLobe = struct with fields:
    magnitude: 3.6551
        angle: -35.4017
       extent: [271 361]
        index: 326.5000
```

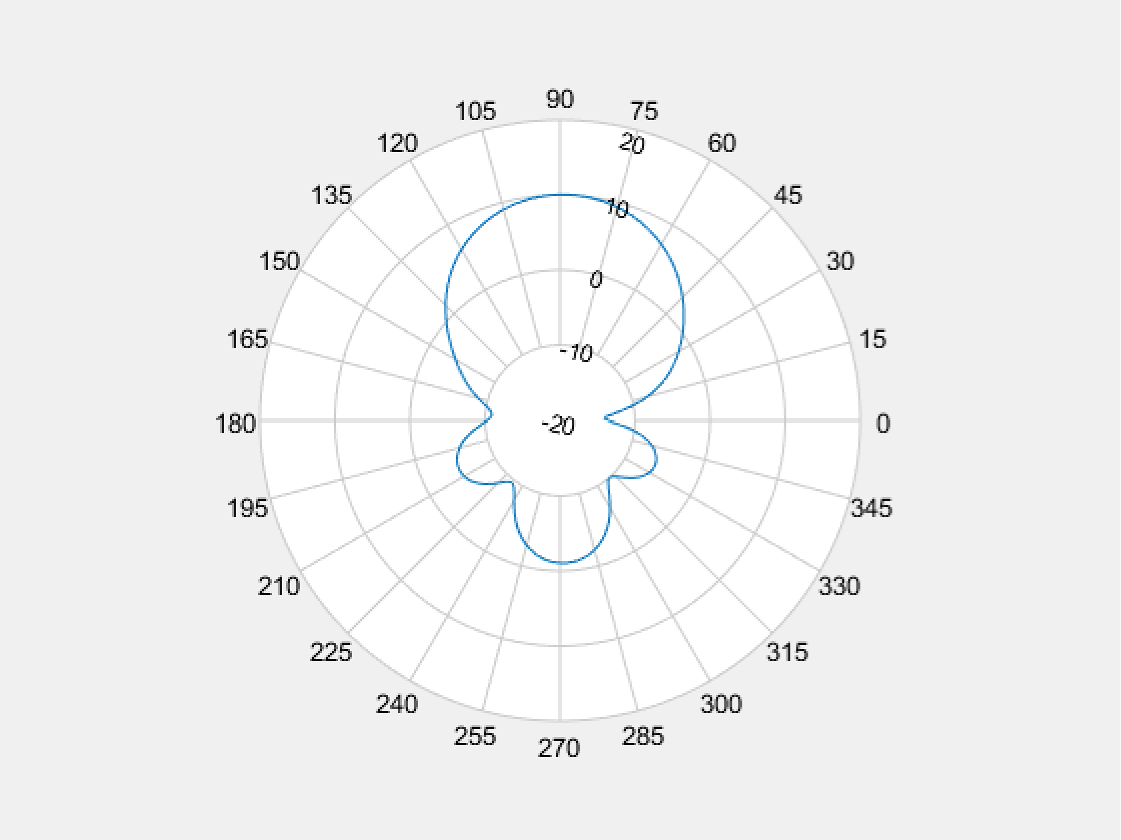
```
SideLobe = L.sideLobes
```

```
SideLobe = struct with fields:
        index: 36
    magnitude: 3.6675
        angle: 34.9030
       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);
```

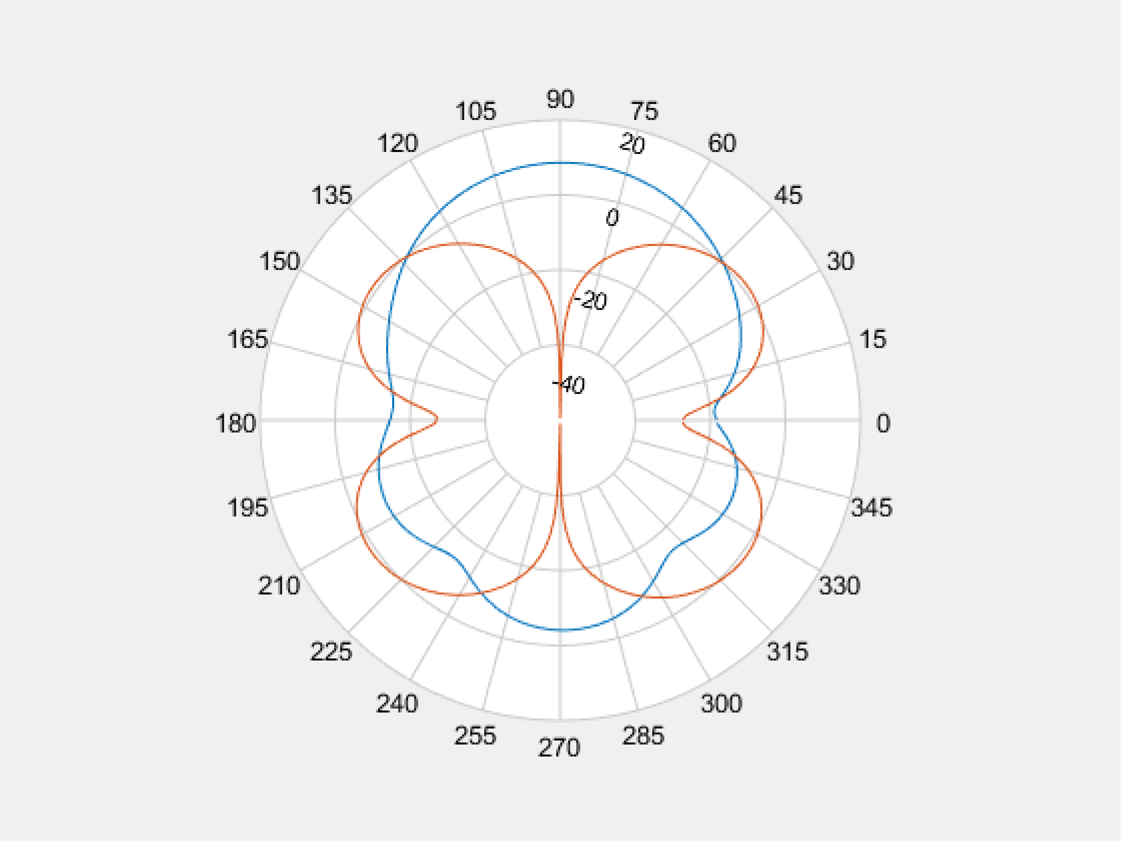


Create a dipole antenna and calculate the directivity at 270 MHz.

```
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: 11.1523
          SLL: 11.0997
         HPBW: 56.8421
         FNBW: 172.5208
        FBIdx: [90 270.5000]
       SLLIdx: [90 273]
      HPBWIdx: [61 118]
      HPBWAng: [59.8338 116.6759]
      FNBWIdx: [4 177]
```

See Also

`add` | `addCursor` | `animate` | `createLabels` | `replace` | `showPeaksTable` | `showSpan`

Introduced in R2016a

replace

Class: `polarpattern`

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

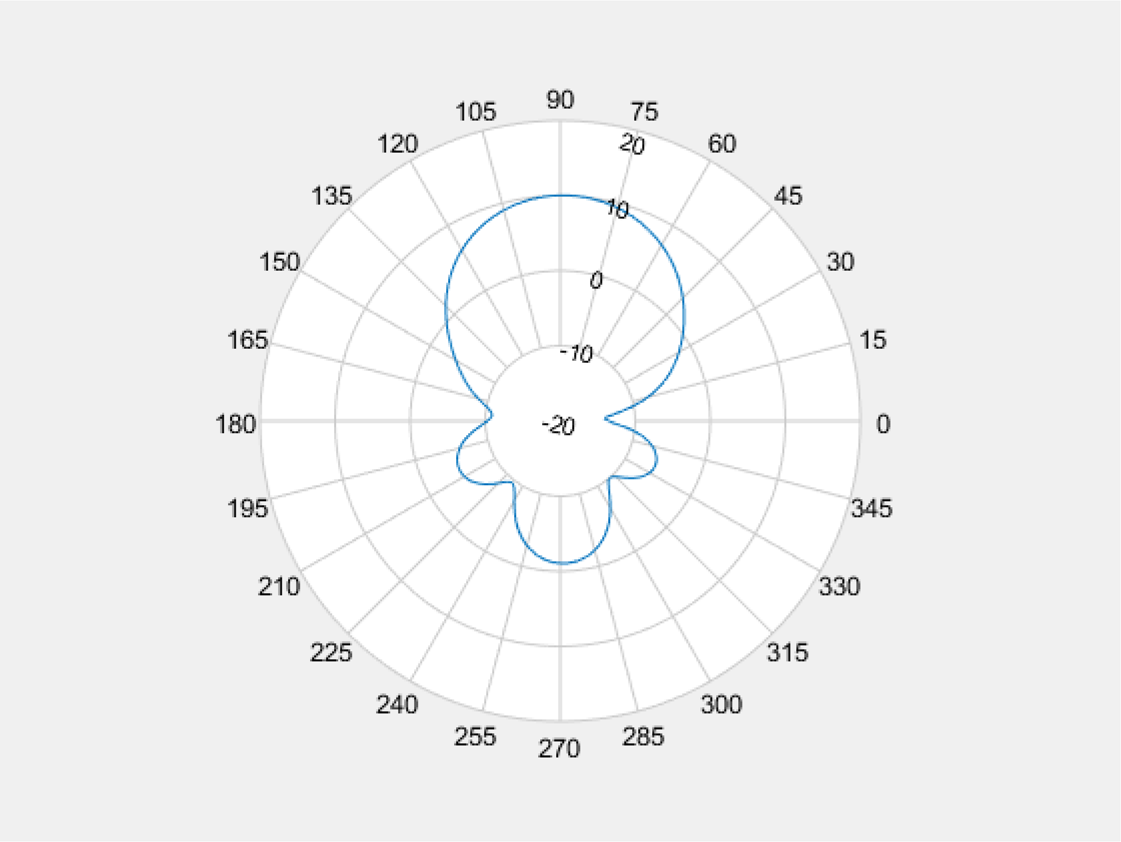
Replace Polar Plot Data with New Data

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);
```

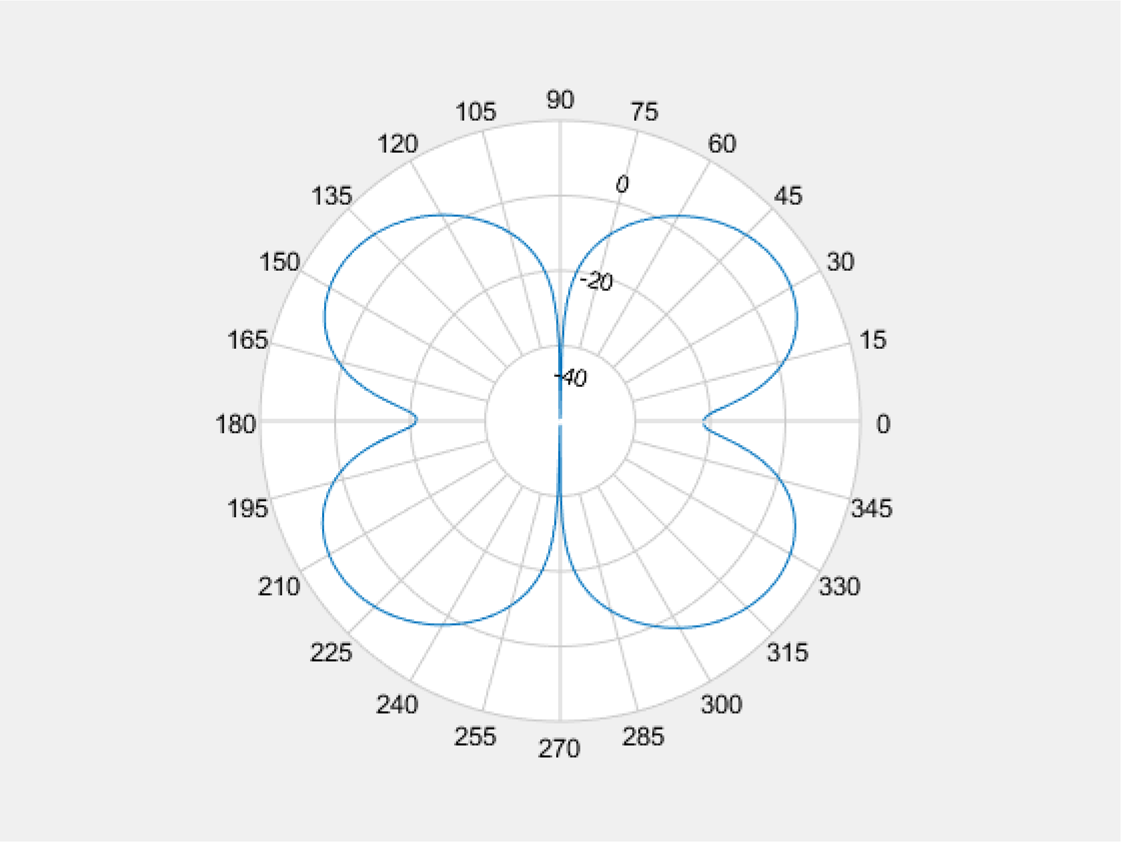


Create a dipole antenna and calculate its directivity at 270 MHz.

```
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);
```



See Also

`add` | `addCursor` | `animate` | `createLabels` | `findLobes` | `showPeaksTable` | `showSpan`

Introduced in R2016a

showPeaksTable

Class: `polarpattern`

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

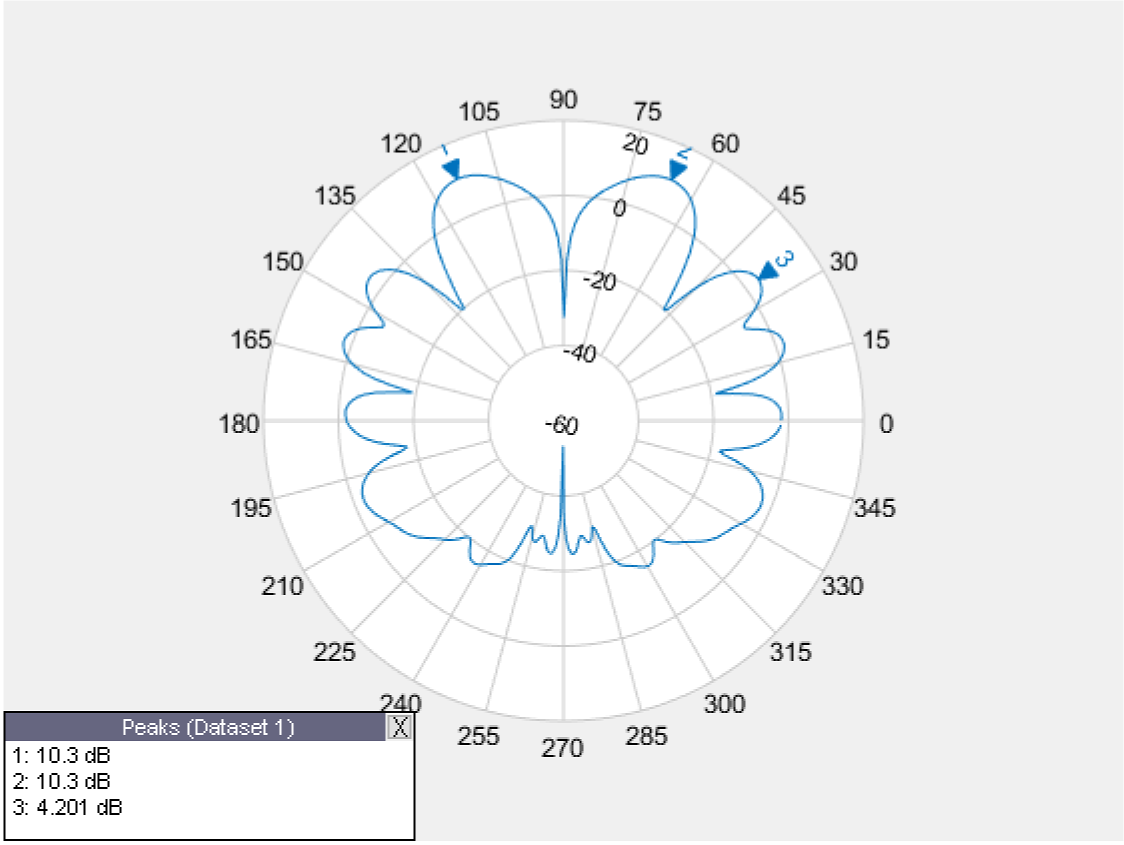
Peaks of Antenna in Polar Pattern

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);
```

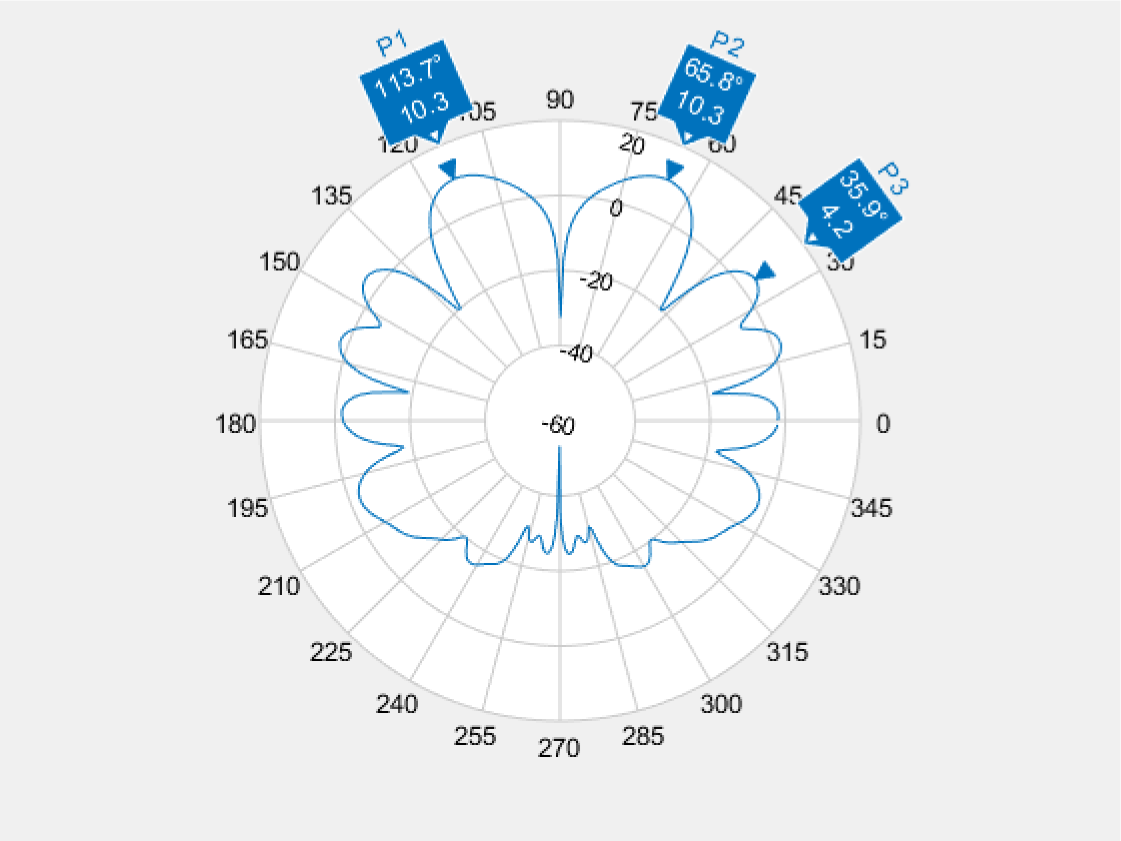


Peaks (Dataset 1)

```
1: 10.3 dB
2: 10.3 dB
3: 4.201 dB
```

Hide the table. When the peaks table is hidden, the peak markers display the peak values.

```
showPeaksTable(P,0);
```



See Also

add | addCursor | animate | createLabels | findLobes | replace | showSpan

Introduced in R2016a

showSpan

Class: `polarpattern`

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.

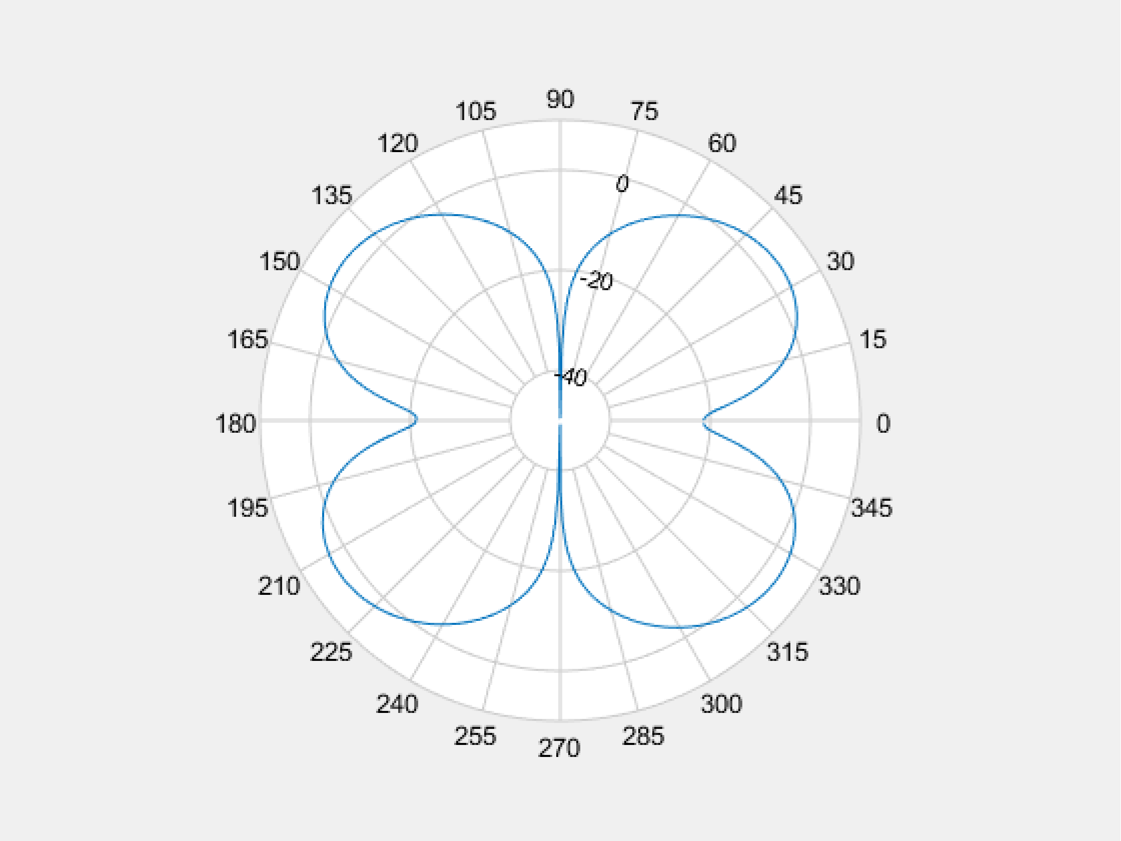
Example: `showspan(p,'C1','C2')`. Displays the angle span between cursors, `C1` and `C2` in polar plot, `p`.

Examples

Show Angle Span

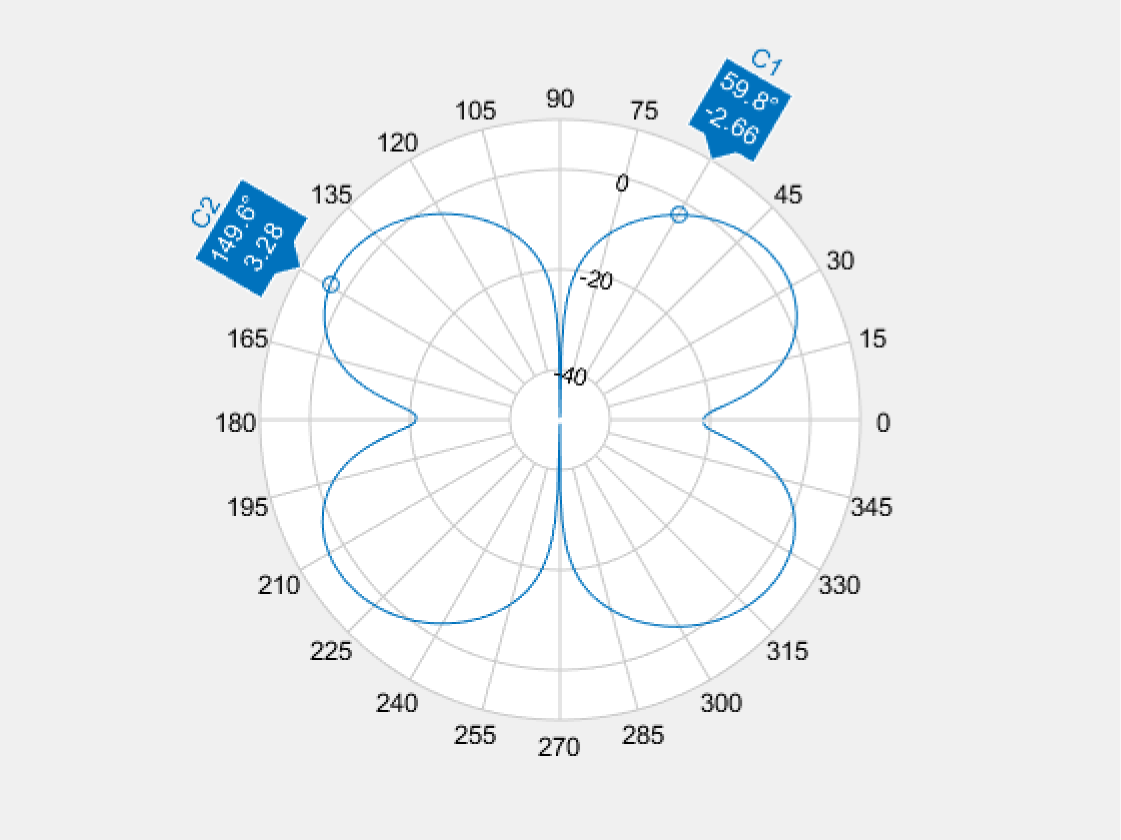
Create a dipole antenna and plot the directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
p = polarpattern(D);
```



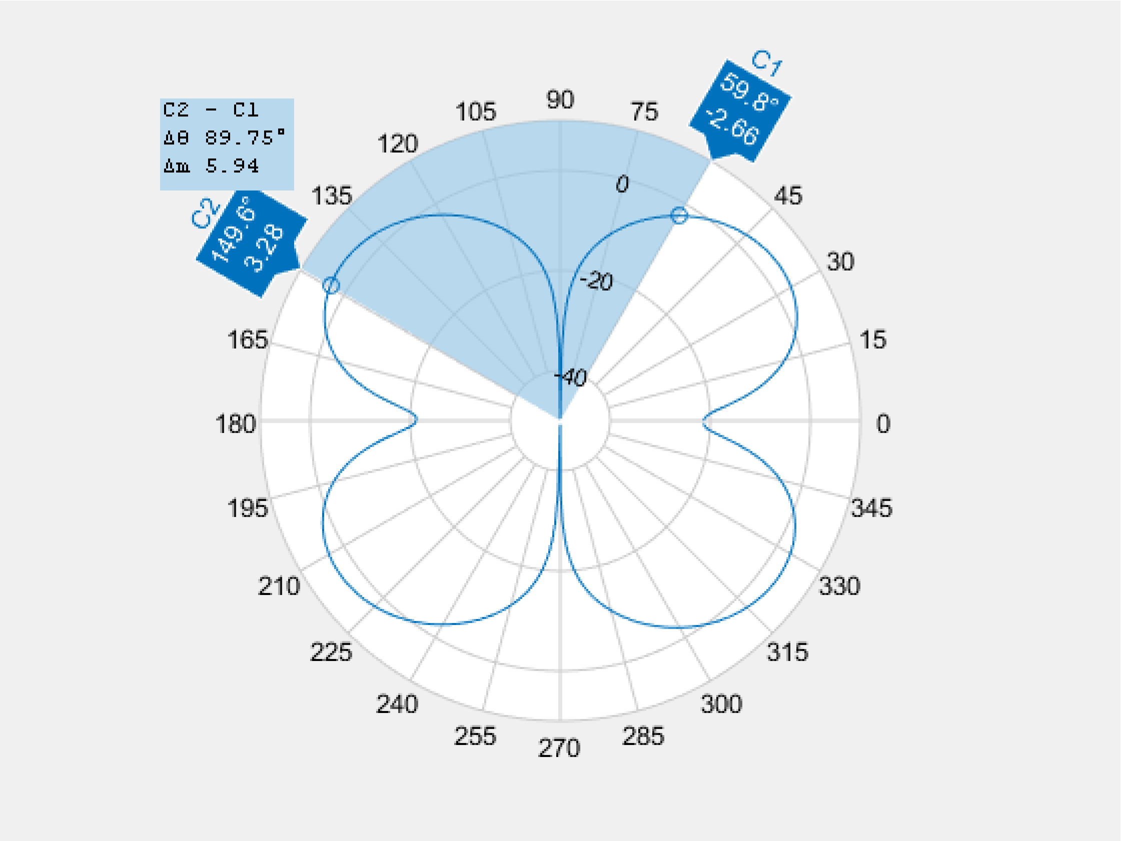
Add cursors to the polar plot at approximately 60 and 150 degrees.

```
addCursor(p,[60 150]);
```



Show the angle span between the two angles.

```
showSpan(p,'C1','C2');
```



See Also

add | addCursor | animate | createLabels | findLobes | replace | showPeaksTable

Introduced in R2016a

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.

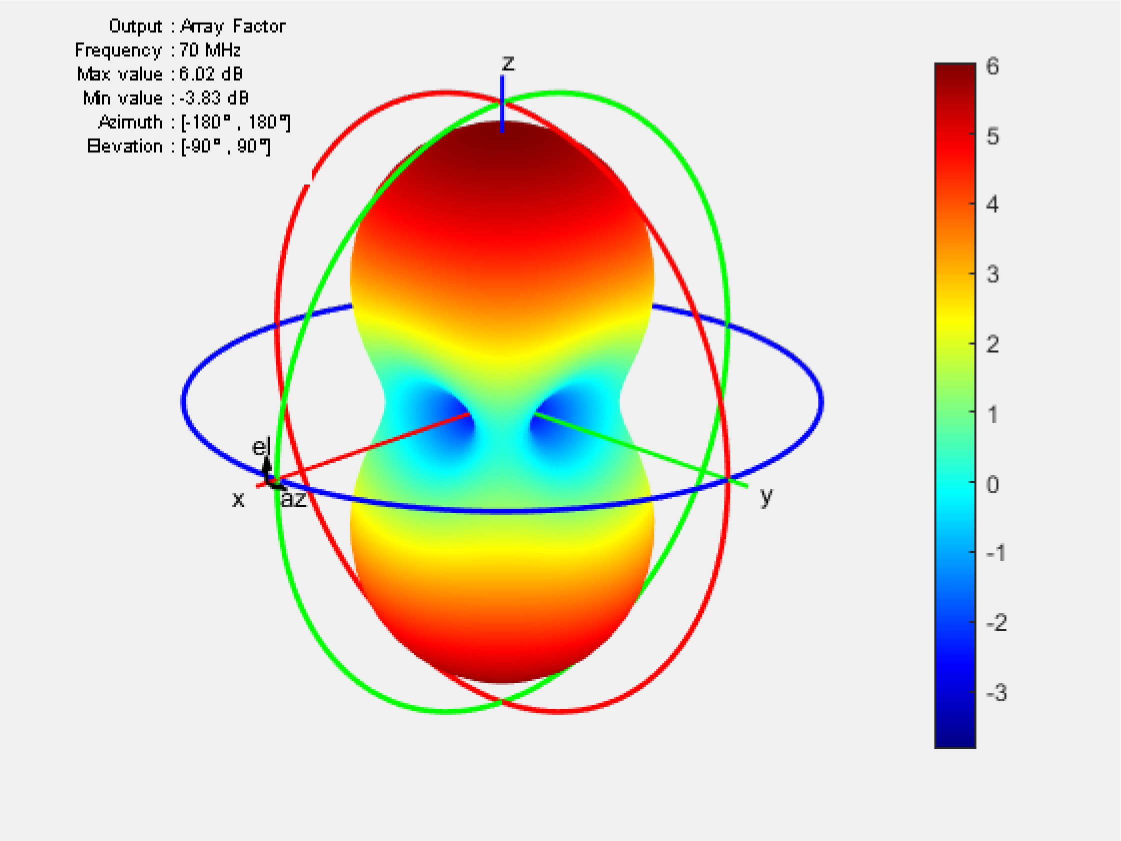
`[af,azimuth,elevation] = arrayFactor( ___ ,Name,Value)` returns the array factor using additional options specified by one or more `Name,Value` pair arguments. Specify name-value pair arguments after all other input arguments.

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 handle

Input antenna array object, specified as an object handle.

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`

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.

See Also

`feedCurrent` | `pattern` | `patternMultiply`

Introduced in R2017a

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.

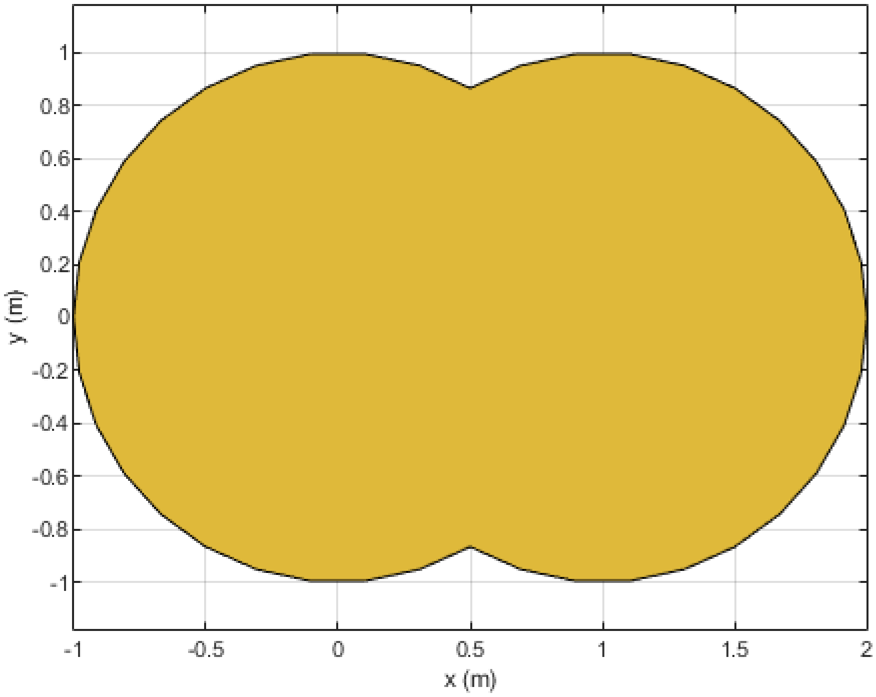
```
circle1 = antenna.Circle;
```

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 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.

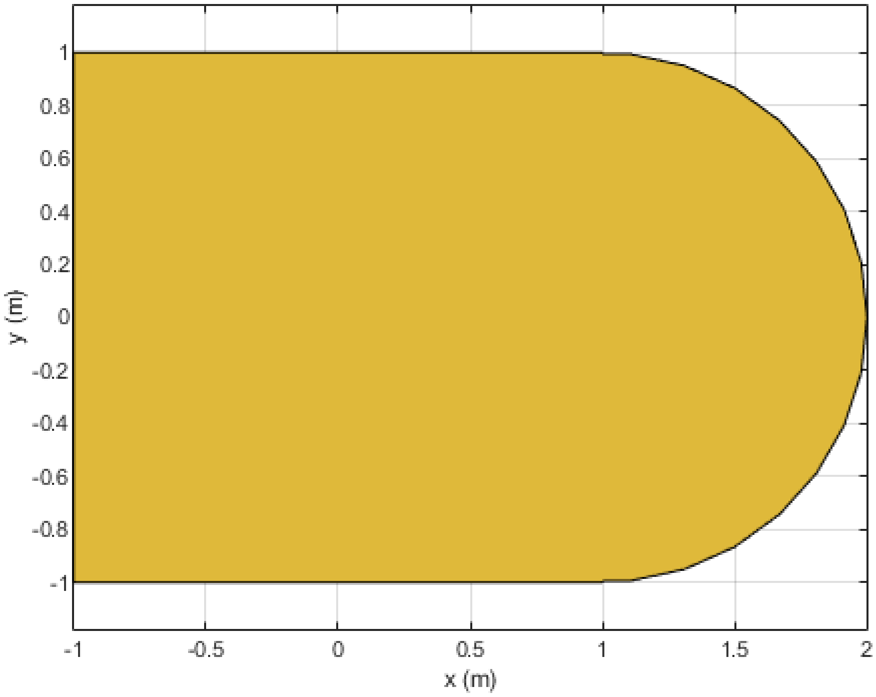
```
rect1 = antenna.Rectangle('Length',2,'Width',2);
```

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 handle

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `c = add(rectangle1, rectangle2)` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

See Also

`area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

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.

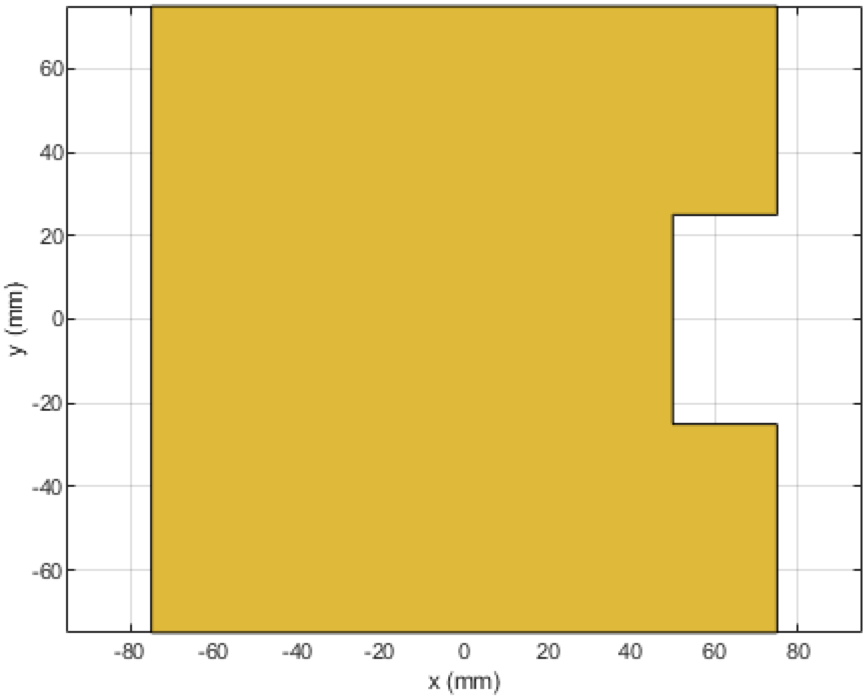
```
r  = antenna.Rectangle('Length',0.15,'Width',0.15);
```

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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `c = area(rectangle)` where rectangle is created using `antenna.Rectangle` object.

See Also

`add` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

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.

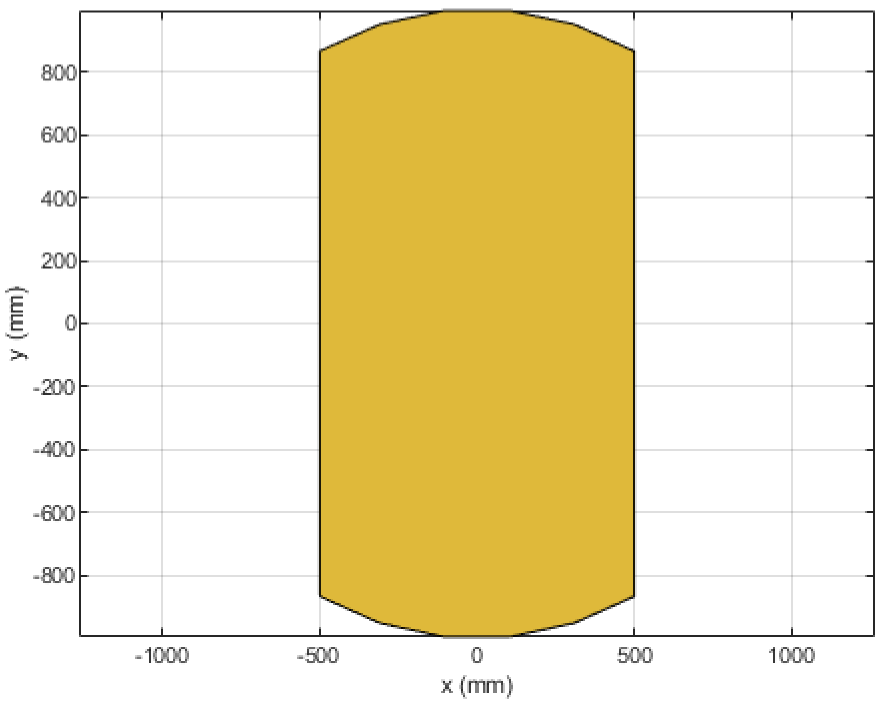
```
c = antenna.Circle;
```

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 handle

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `c = intersect(rectangle1, rectangle2)` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

See Also

`add` | `area` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `show` | `subtract` | `translate`

Introduced in R2017a

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.

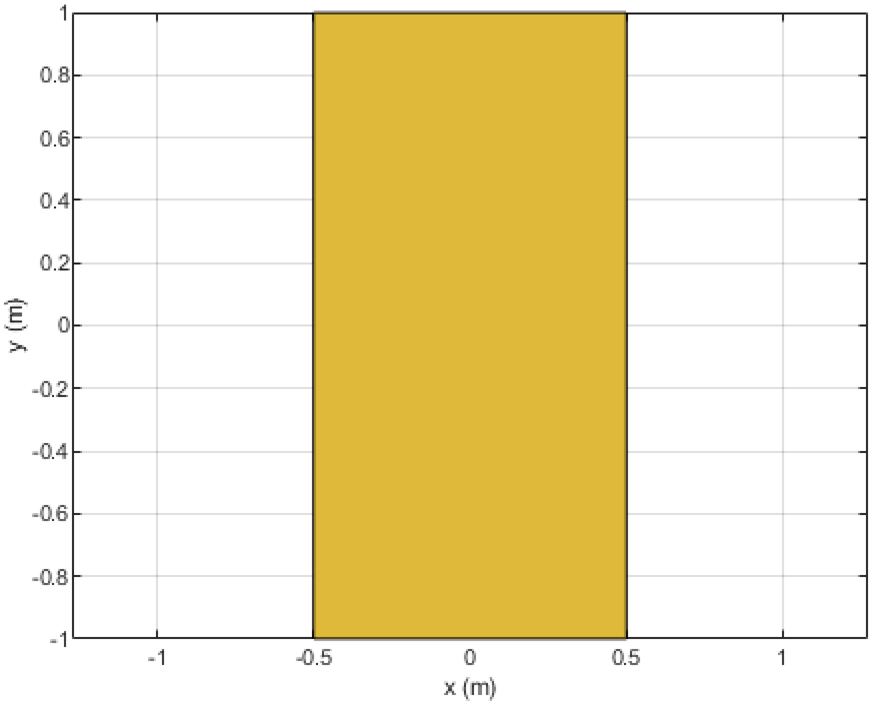
`c = rotate(shape,angle,axis1,axis2)` rotate shape about an axes object and angle.

Examples

Rotate Rectangle

Create a rectangle shape.

```
r = antenna.Rectangle;
show(r)
axis equal
```

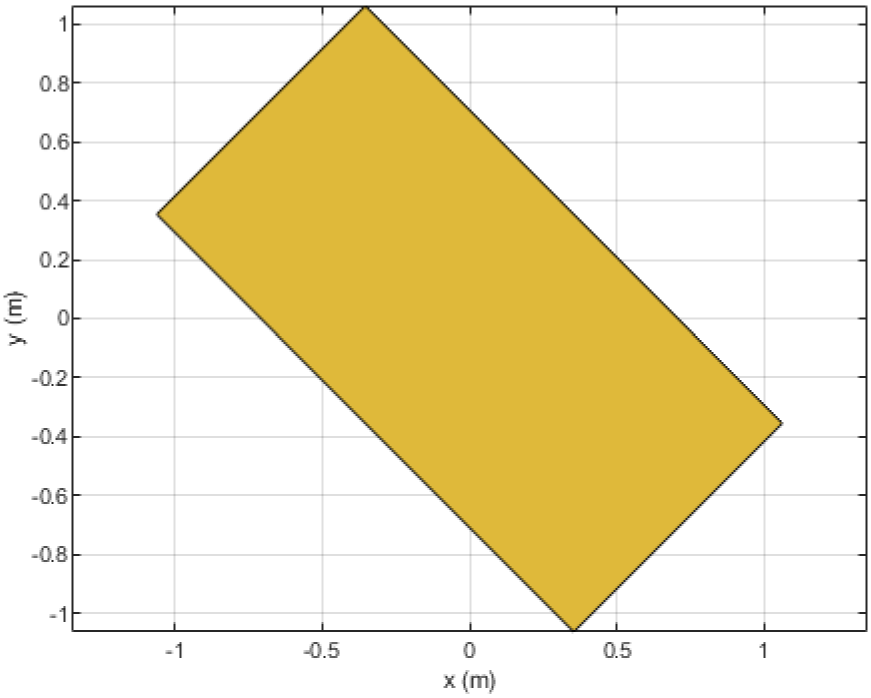


Rotate the rectangle at 45 degrees about the Z-axis.

```
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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

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`

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

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.

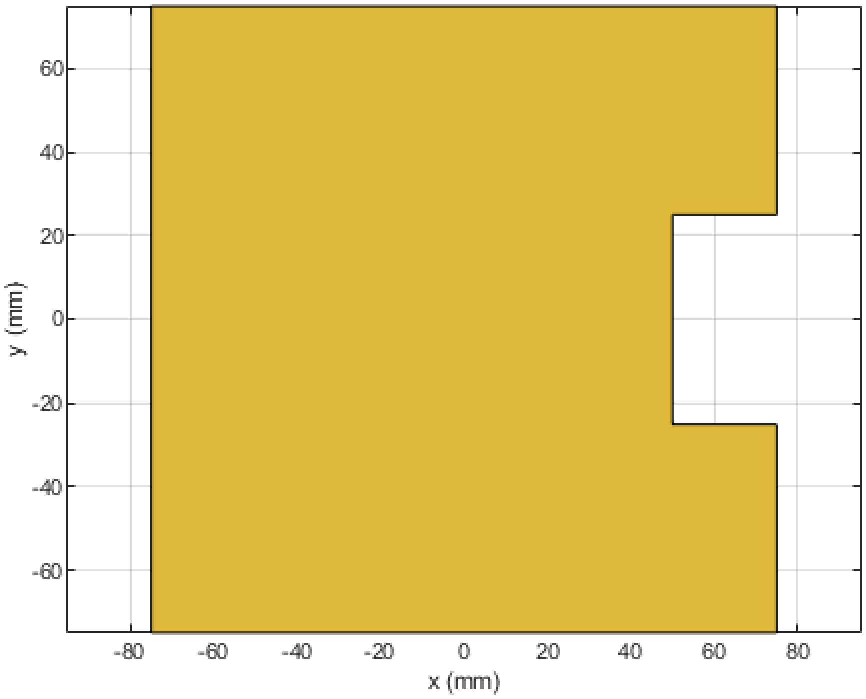
```
r  = antenna.Rectangle('Length',0.15,'Width',0.15);
```

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 handle

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `c = subtract(rectangle1, rectangle2)` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `translate`

Introduced in R2017a

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.

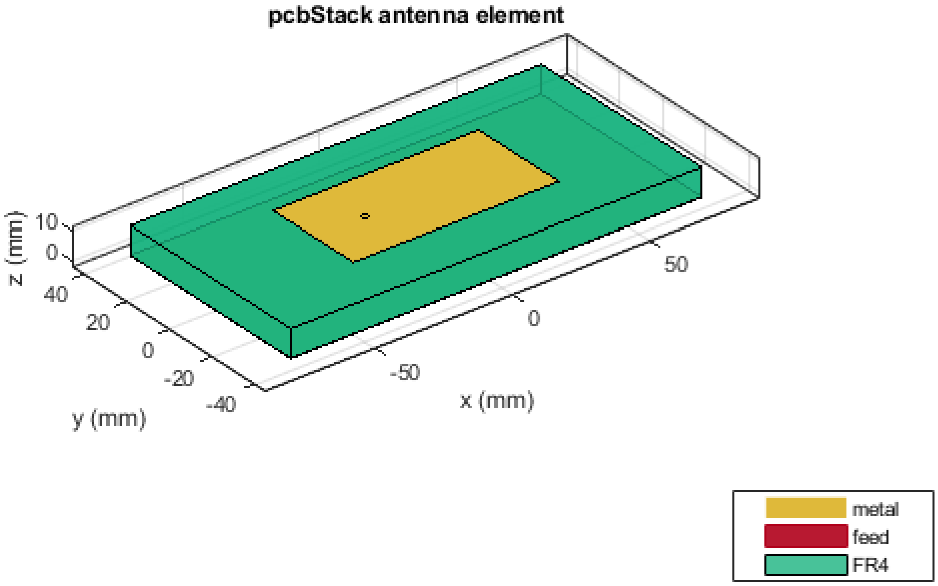
`[a,g] = gerberWrite(designobject,writer,rfconnector)` creates a Gerber file using specified PCB writer and connector services.

Note You can only use output arguments if the `designobject` is a `pcbStack` object.

Examples

Antenna Gerber Files from PCB Stack

```
Create a patch antenna with FR4 as dielectric material using |pcbStack|
object.

p = pcbStack;
d = dielectric('FR4');
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\Bdoc20a_1326390_8984\ib9D0363\4\tpce80fbf8\antenna-ex96485213\antenna_design_file'
```

Show Antenna PCB Design Using Mayhew Manufacturing Service

Create a coplanar inverted F antenna.

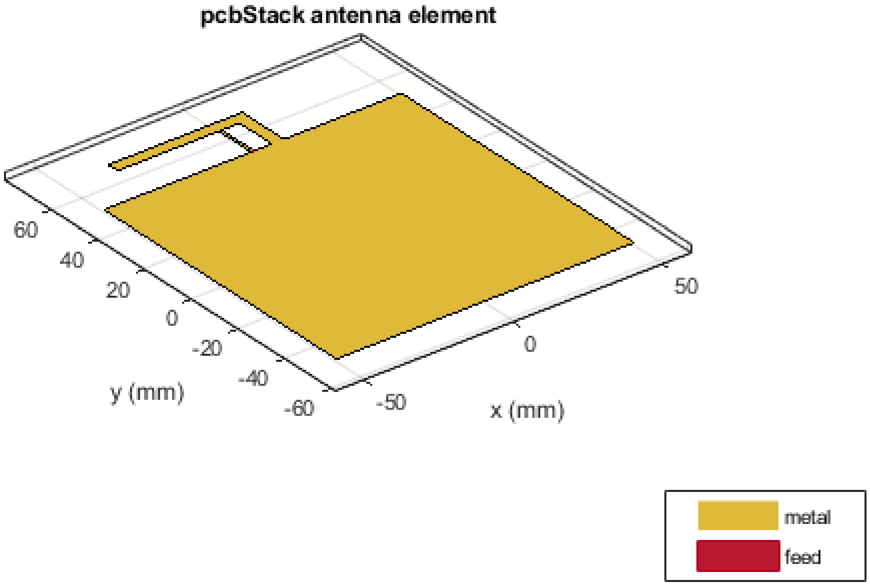
```
fco = invertedFcoplanar('Height',14e-3,'GroundPlaneLength', 100e-3,  ...
                  'GroundPlaneWidth', 100e-3);
```

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` .

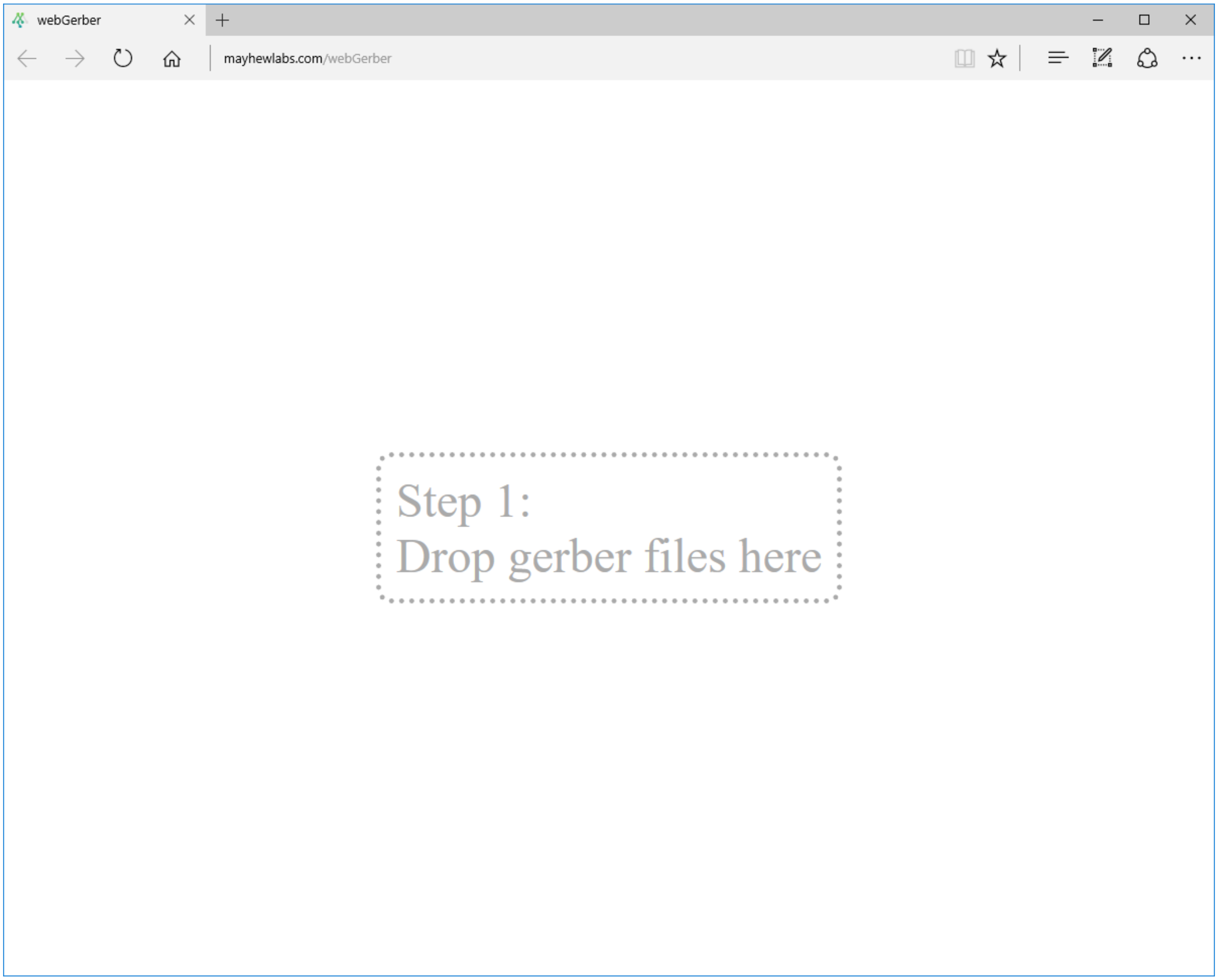
```
PW = PCBWriter(p,s,c)
```

```
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.

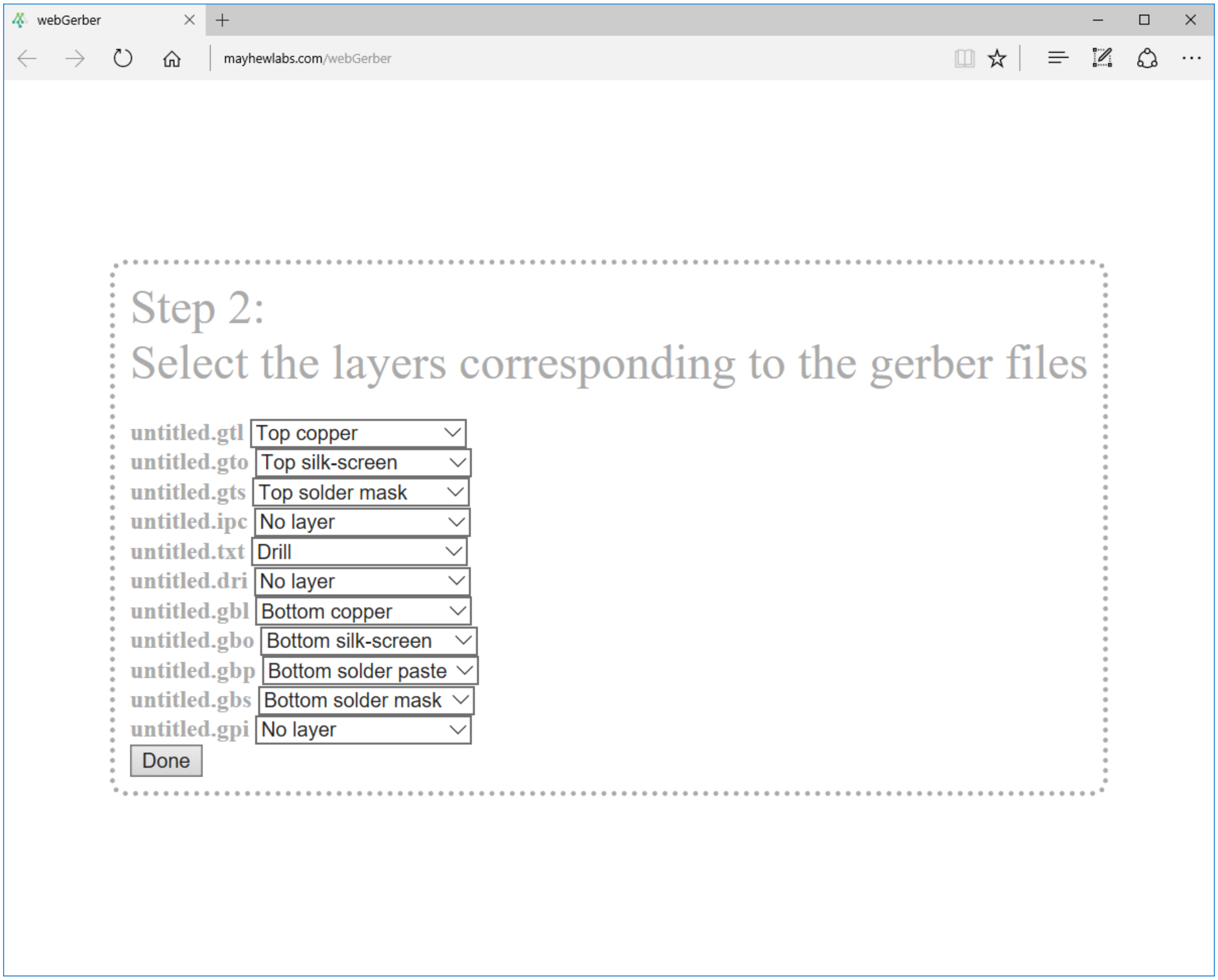
```
gerberWrite(PW)
```

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

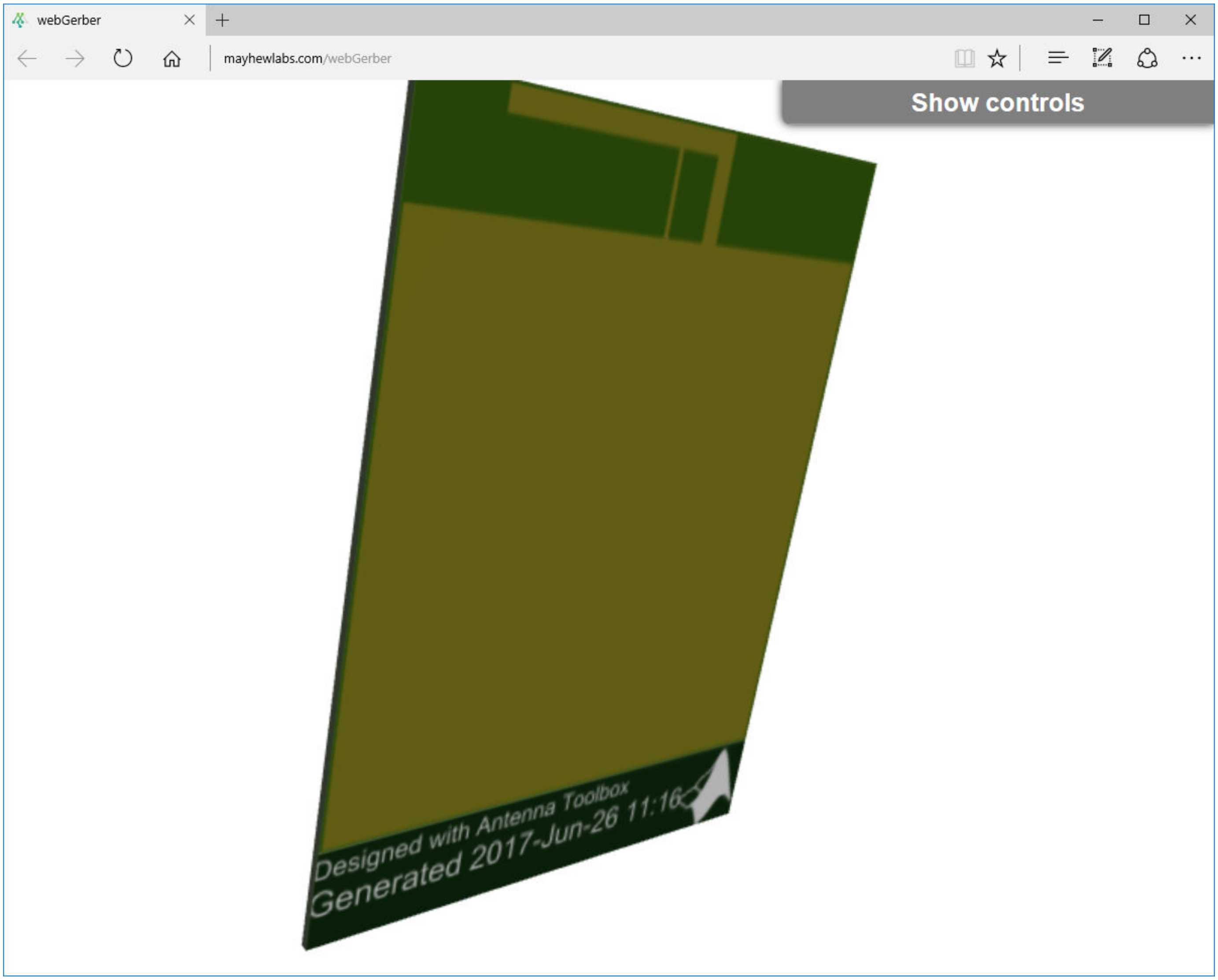
Drag and drop all your files from the "untitled" folder.



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;
```

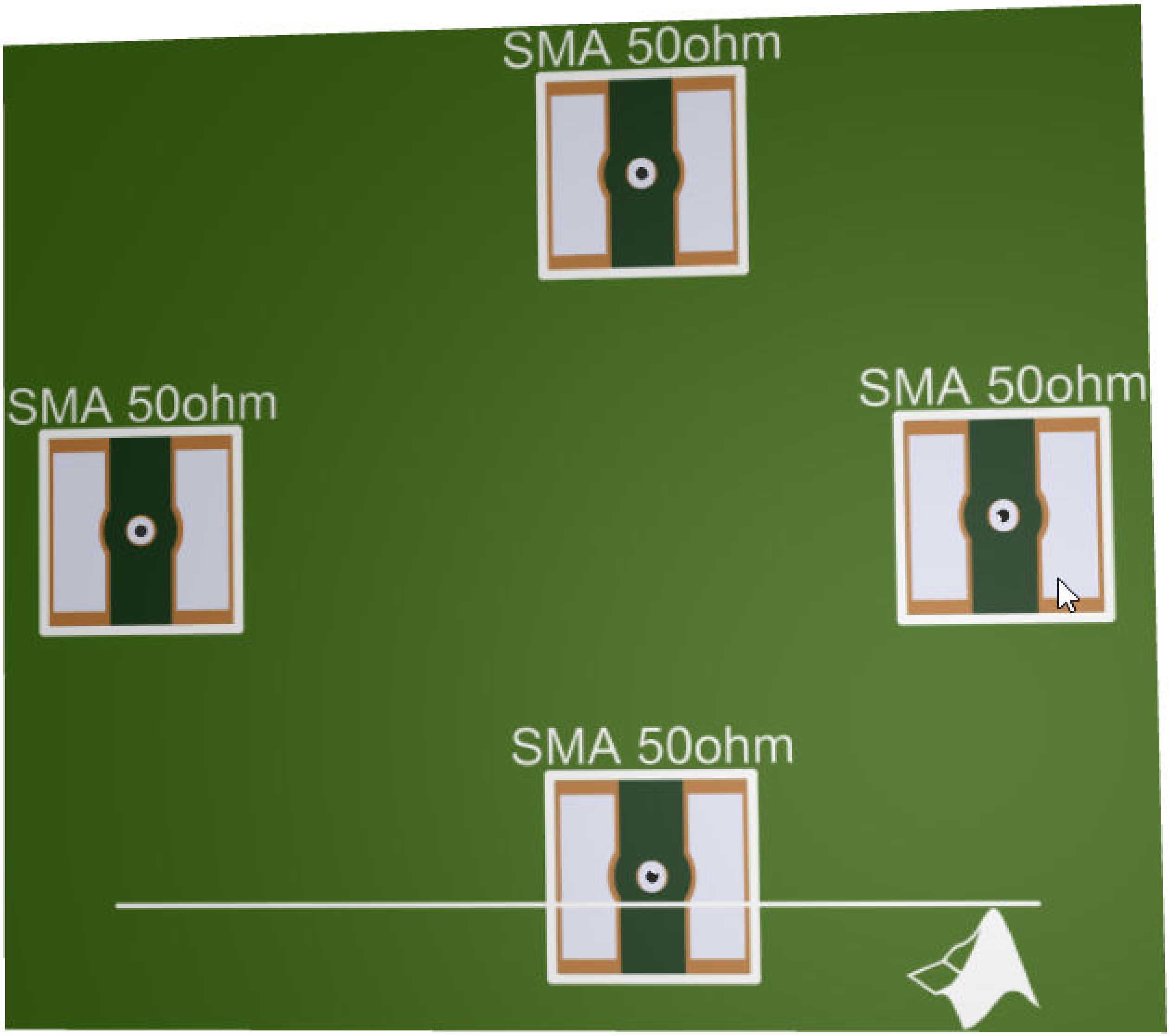
Pick a manufacturing service.

```
Wr = PCBServices.MayhewWriter;
```

Create a Gerber file and generate it.

```
A = PCBWriter(pb,Wr,C);
gerberWrite(A)
```

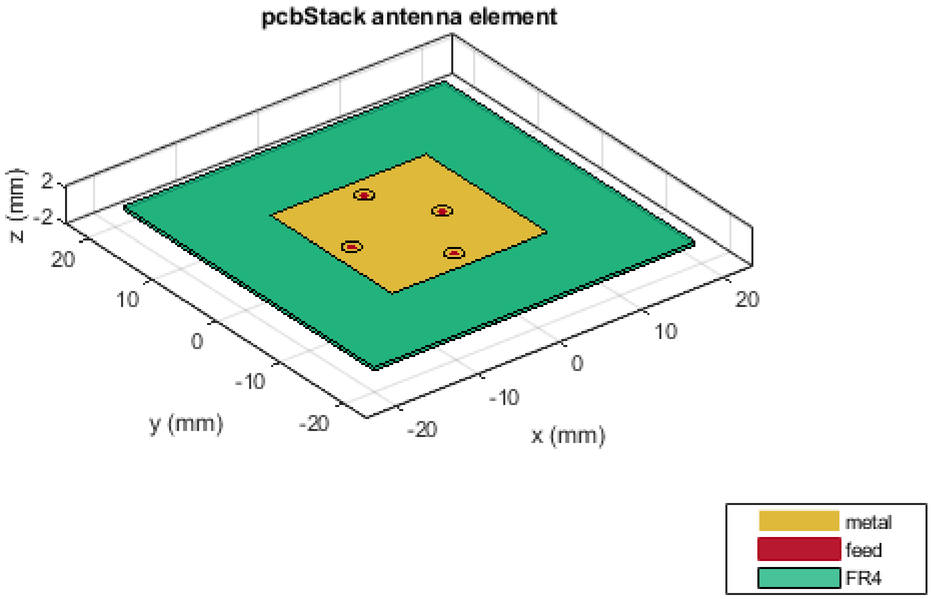
```
Warning: No metal specified for PCB
```



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.

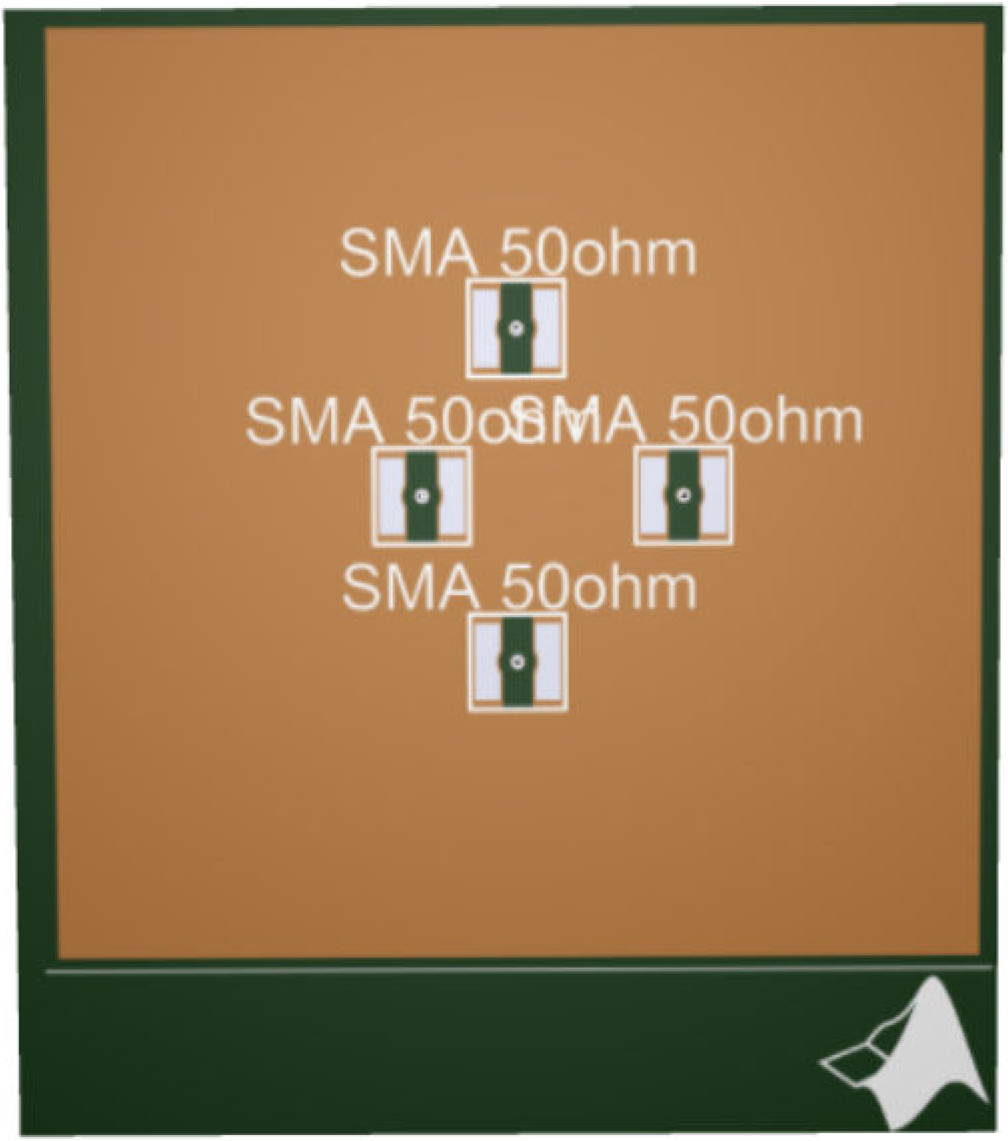
```
Wr = PCBServices.MayhewWriter;
Wr.Filename = 'Microstrip antenna-4ports';
```

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)
```



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 handle

`PCBWriter` object that generated the Gerber files, returned as an object handle.

g — Path to generated Gerber files folder
character vector

Path to generated Gerber files folder, returned as character vector.

See Also

`PCBConnectors` | `PCBServices`

Introduced in R2017b

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.

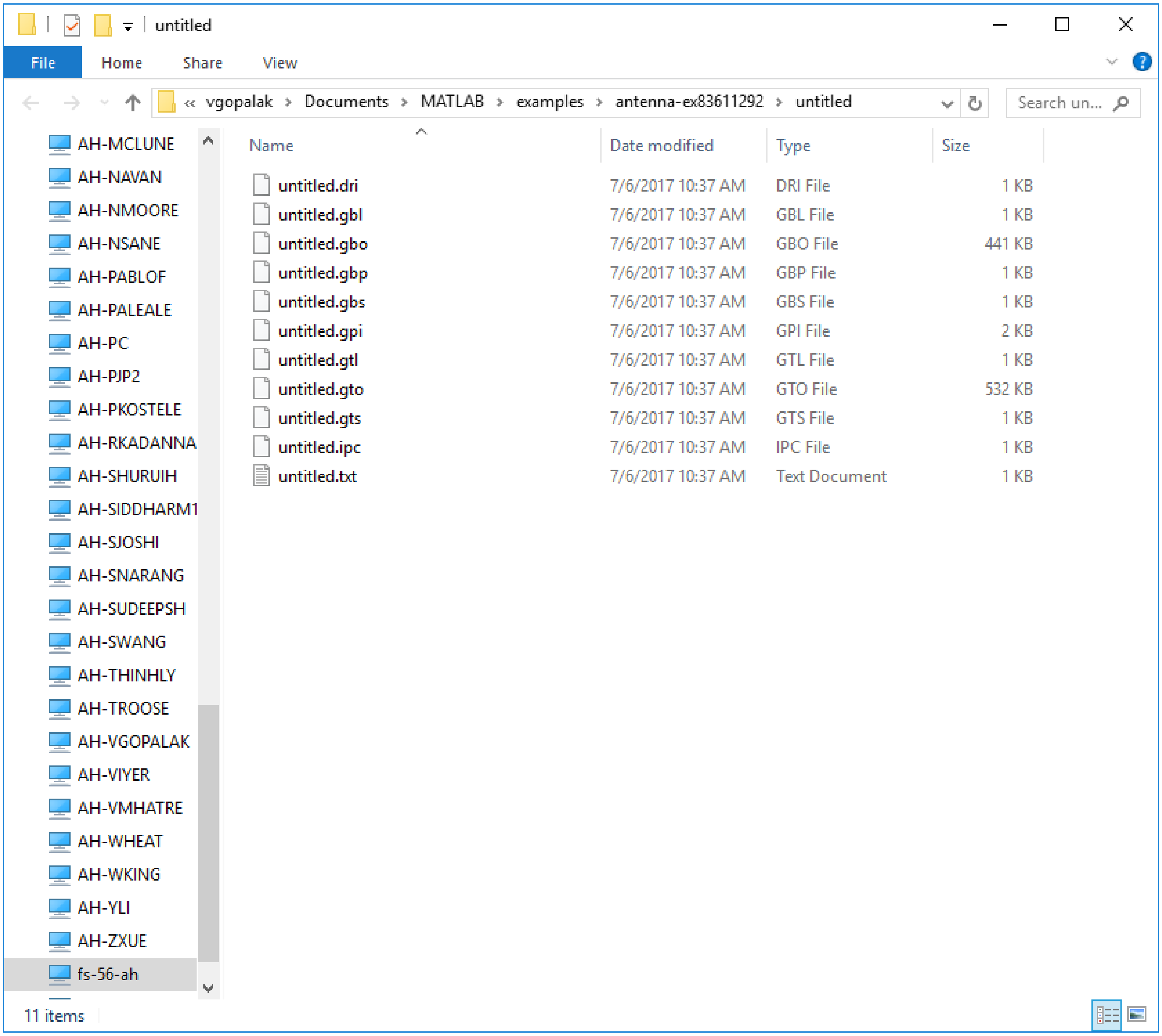
```
PW = PCBWriter(p,s,c);
```

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)`.

See Also

`gerberWrite` | `info` | `sendTo`

Introduced in R2017b

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 = 72.9381 - 1.2090i
```

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`;

See Also

`show`

Introduced in R2017b

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.

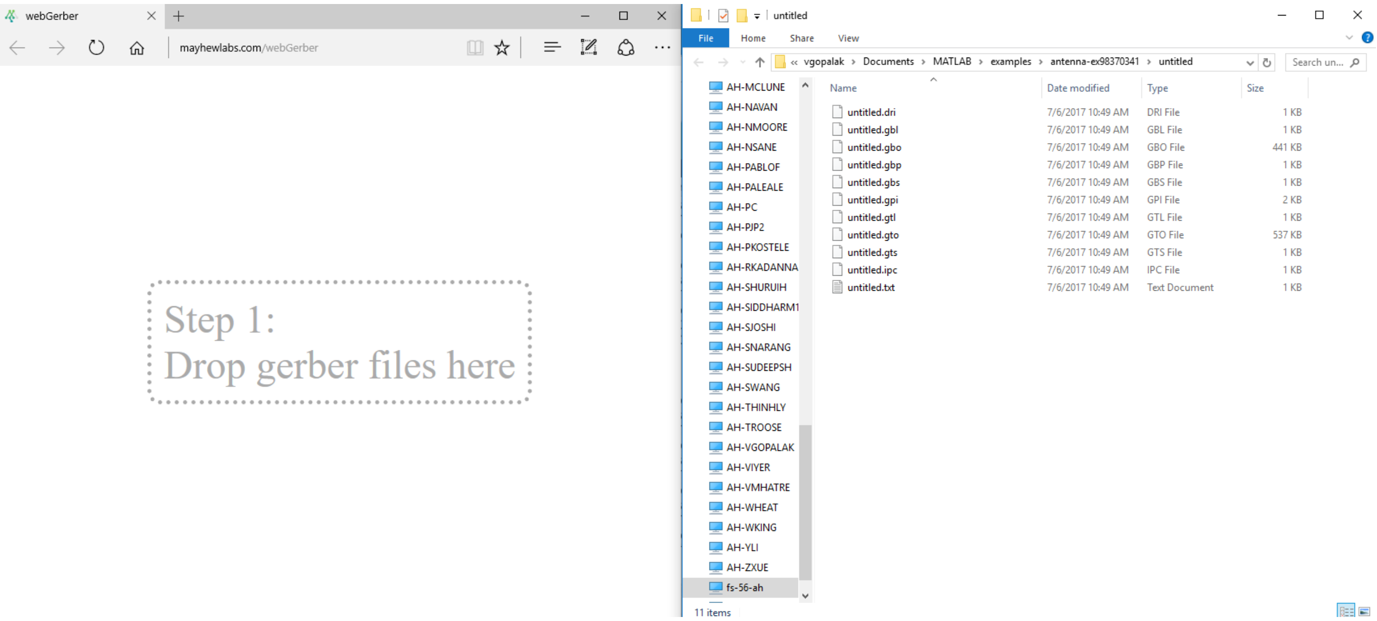
```
PW = PCBWriter(p,s,c);
```

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).

See Also

gerberWrite | info | sendTo

Introduced in R2017b

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`.

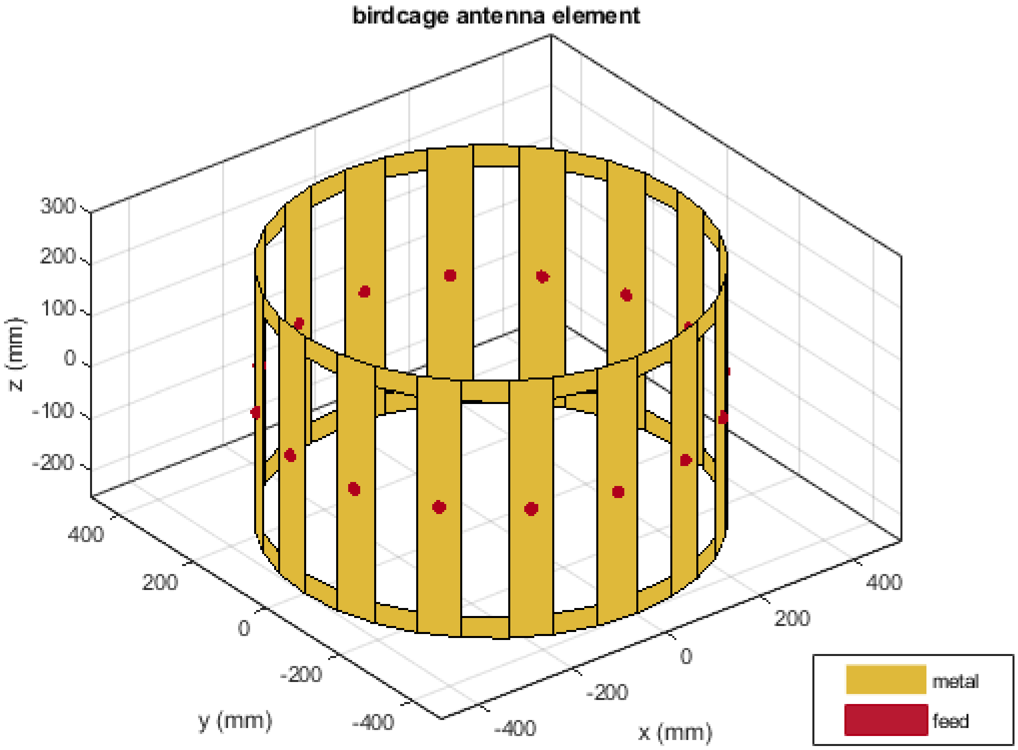
Examples

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
             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)`

See Also

Introduced in R2017b

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`.

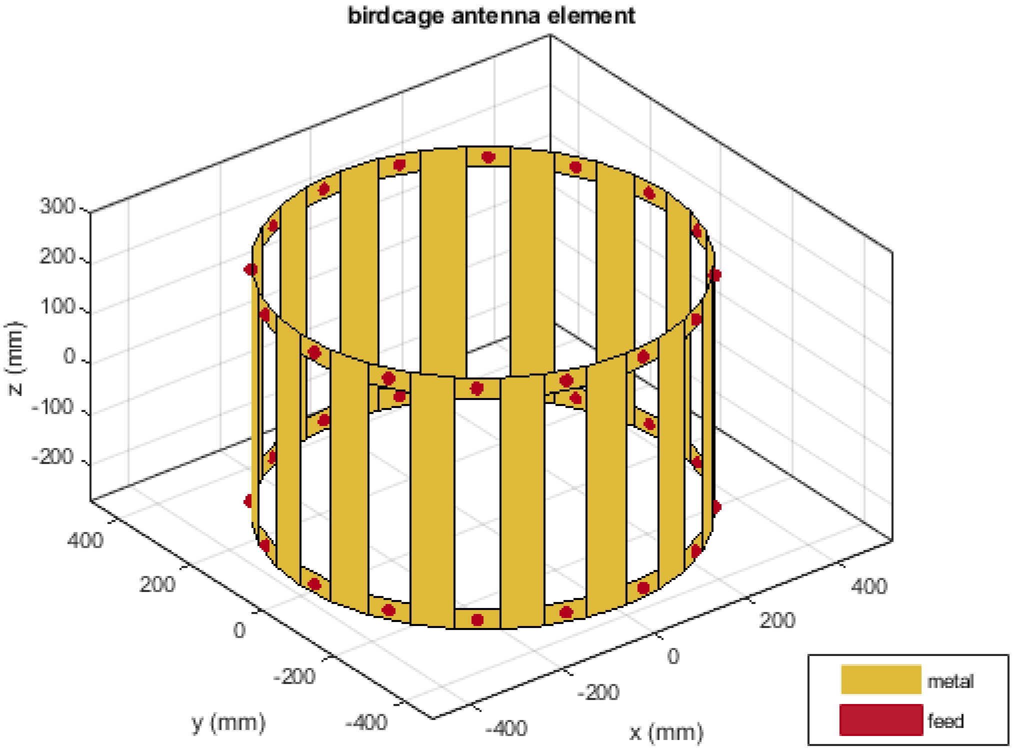
Examples

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
             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)`

See Also

Introduced in R2017b

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.

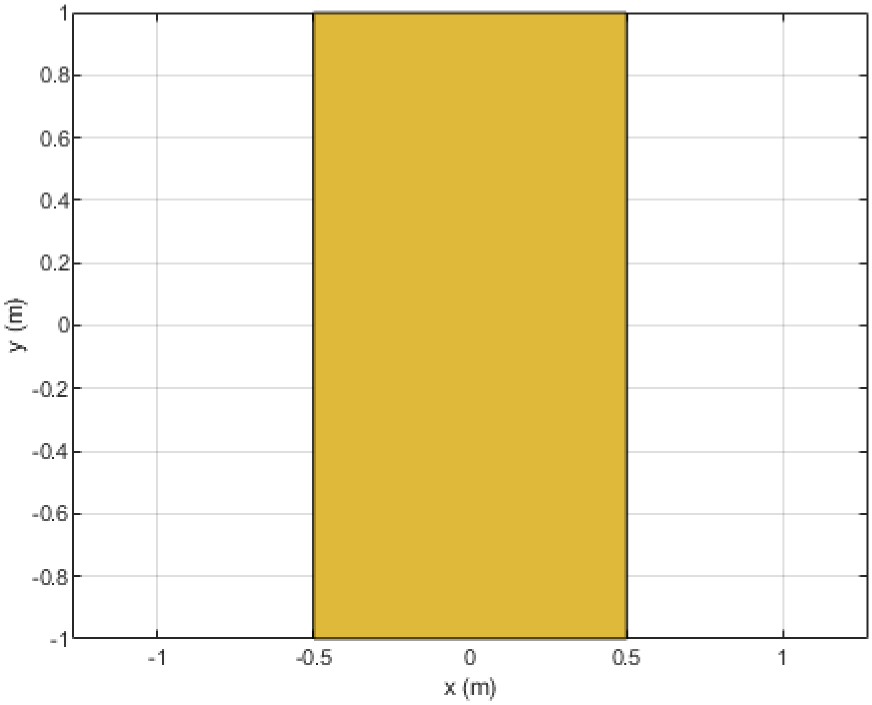
`c =c rotateX(shape,angle)` rotate shape about an axes object and angle.

Examples

Rotate Rectangle About X-Axis

Create a rectangle shape.

```
r = antenna.Rectangle;
show(r)
axis equal
```

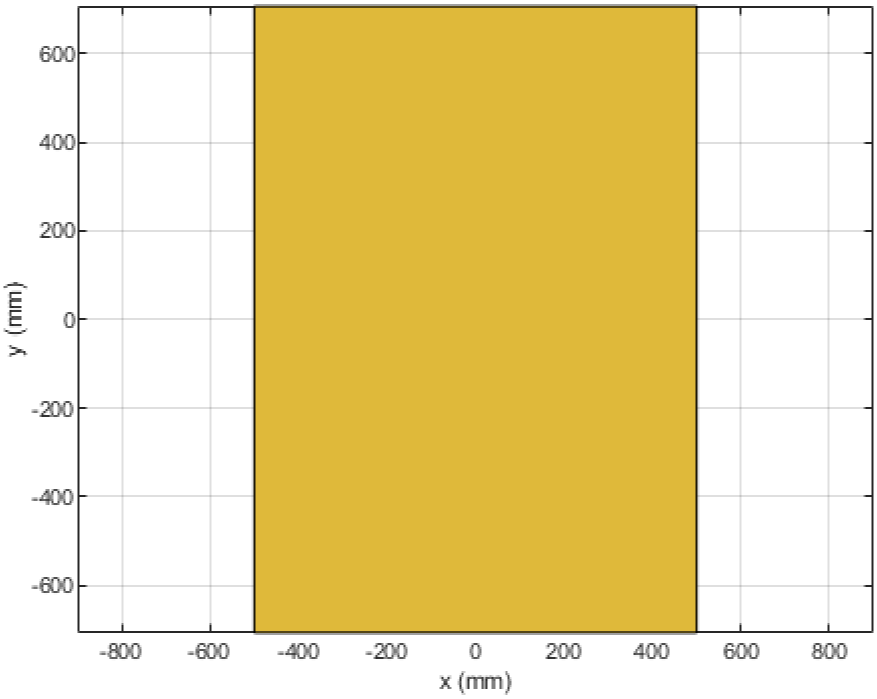


Rotate the rectangle at 45 degrees about the x-axis.

```
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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

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`

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

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.

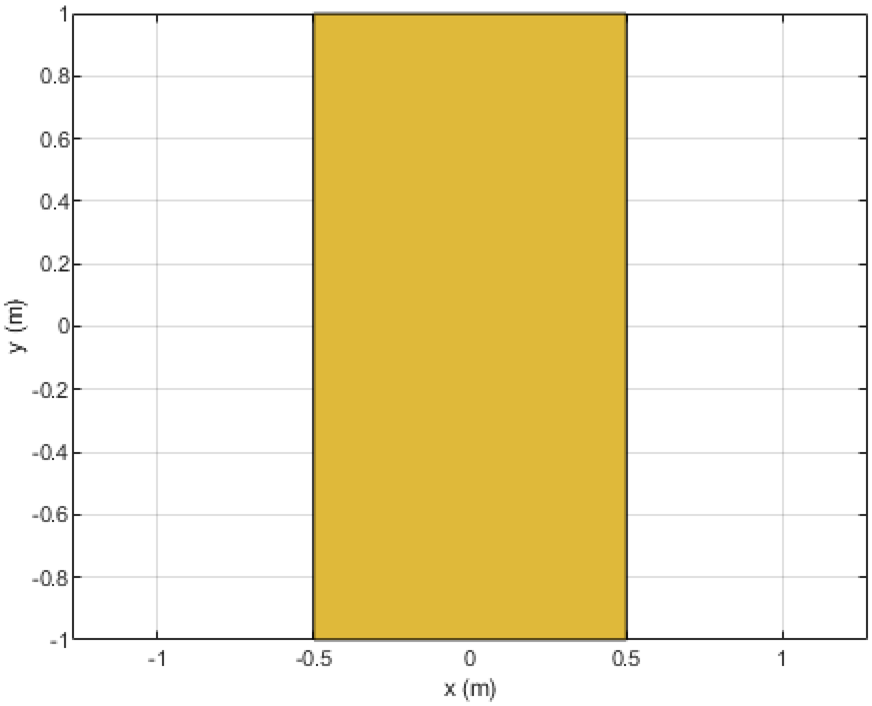
`c = rotateY(shape,angle)` rotate shape about the Y-axis and angle.

Examples

Rotate Rectangle About Y-Axis

Create a rectangle shape.

```
r = antenna.Rectangle;
show(r)
axis equal
```

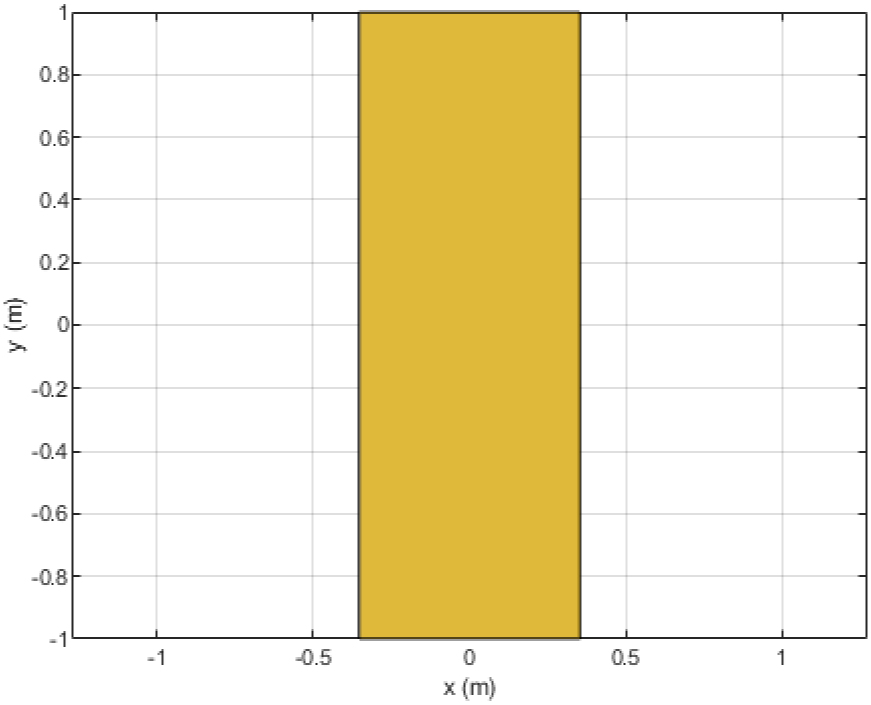


Rotate the rectangle at 45 degrees about the Y-axis.

```
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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

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`

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

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.

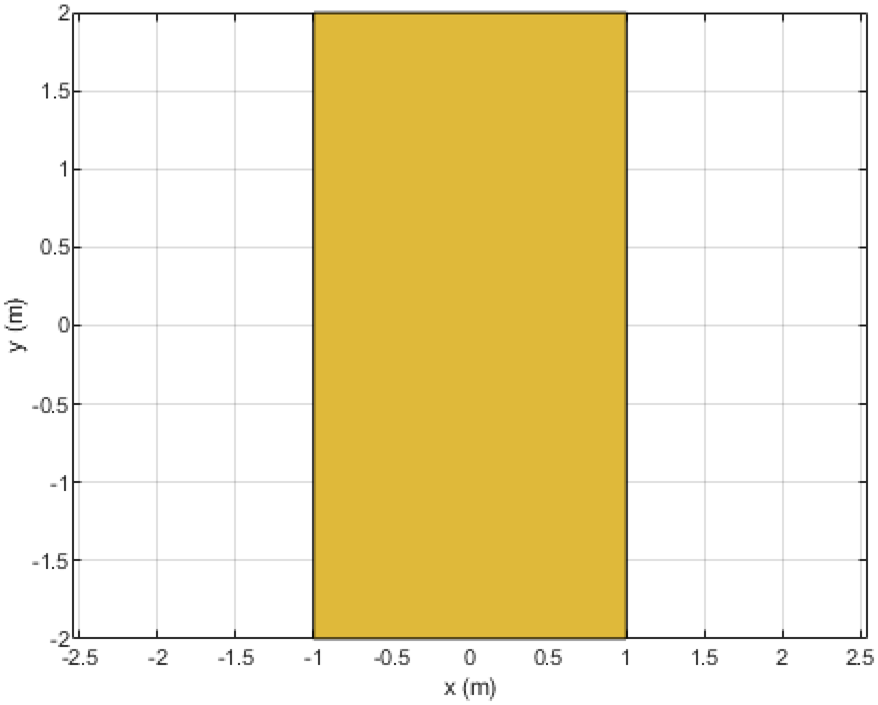
`c = rotateZ(shape,angle)` rotate shape about the Z-axis and angle.

Examples

Create and Rotate Rectangle Using Specified Properties

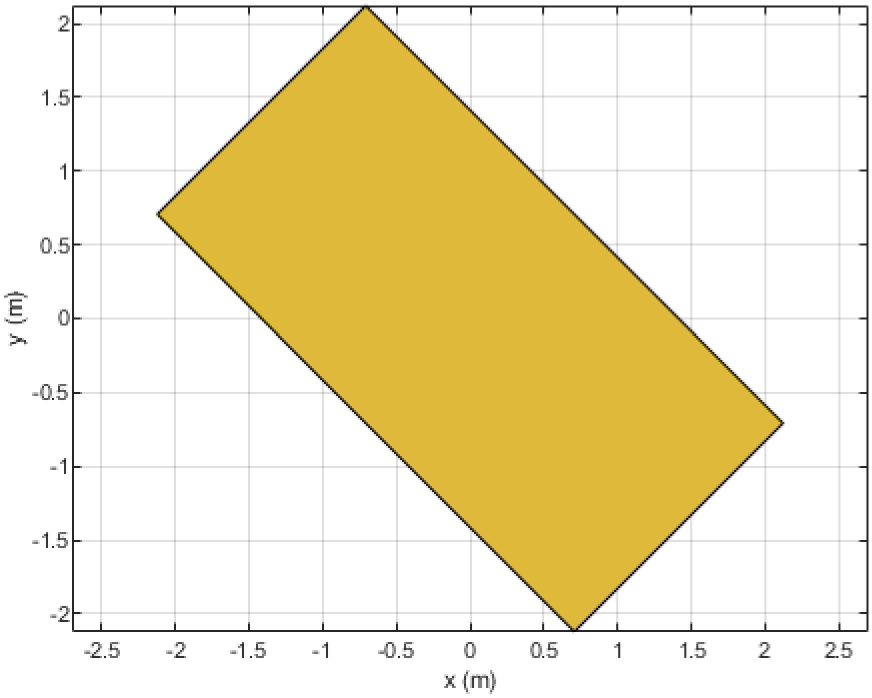
Create and view a rectangle with a length of 2 m and a width of 4 m.

```
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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

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`

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

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

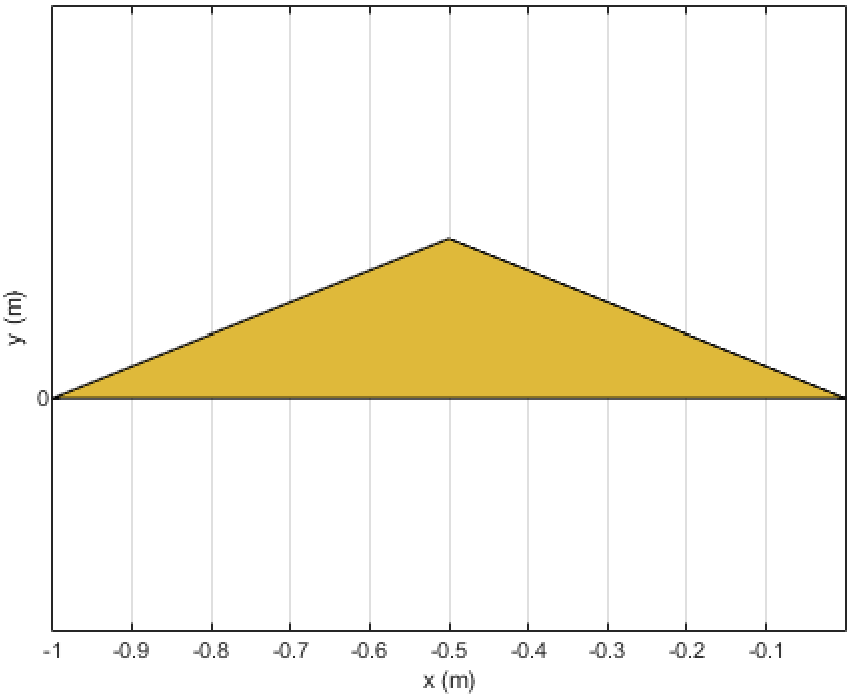
Create and Transform Polygon

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])
```

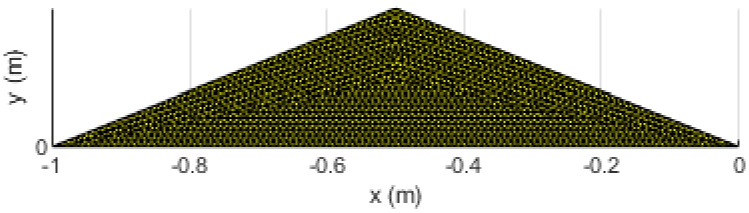
```
p = 
  Polygon with properties:

        Name: 'mypolygon'
    Vertices: [3x3 double]
```

```
show(p)
axis equal
```



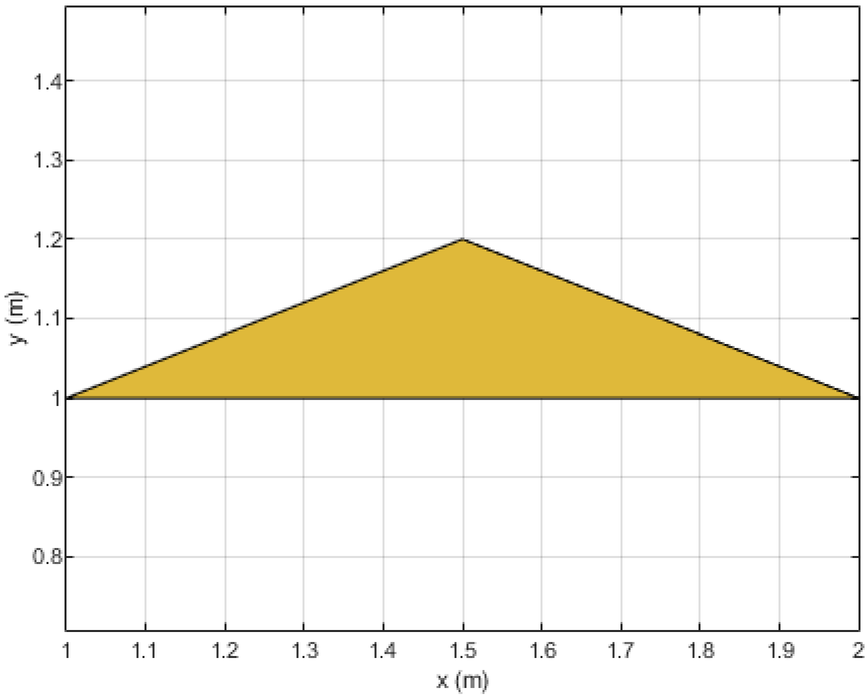
Mesh the polygon and view it.

```
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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

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`

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract`

Introduced in R2017a

plot

Plot boundary of shape

Syntax

```
p = plot(shape,varargin)
```

Description

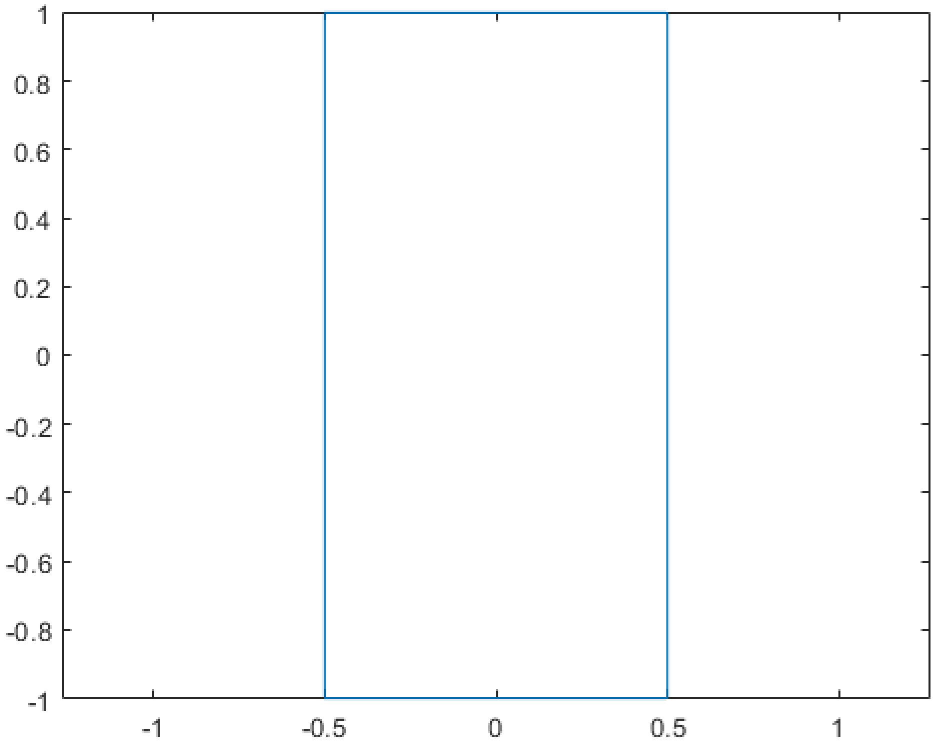
`p = plot(shape,varargin)` plots the boundary of the shape and returns the line handle.

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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `plot(rectangle)` where rectangle is created using `antenna.Rectangle` object.

See Also

`mesh` | `show`

Introduced in R2017a

scale

Change the size of the shape by a fixed amount

Syntax

```
c = scale(shape,scaling)
```

Description

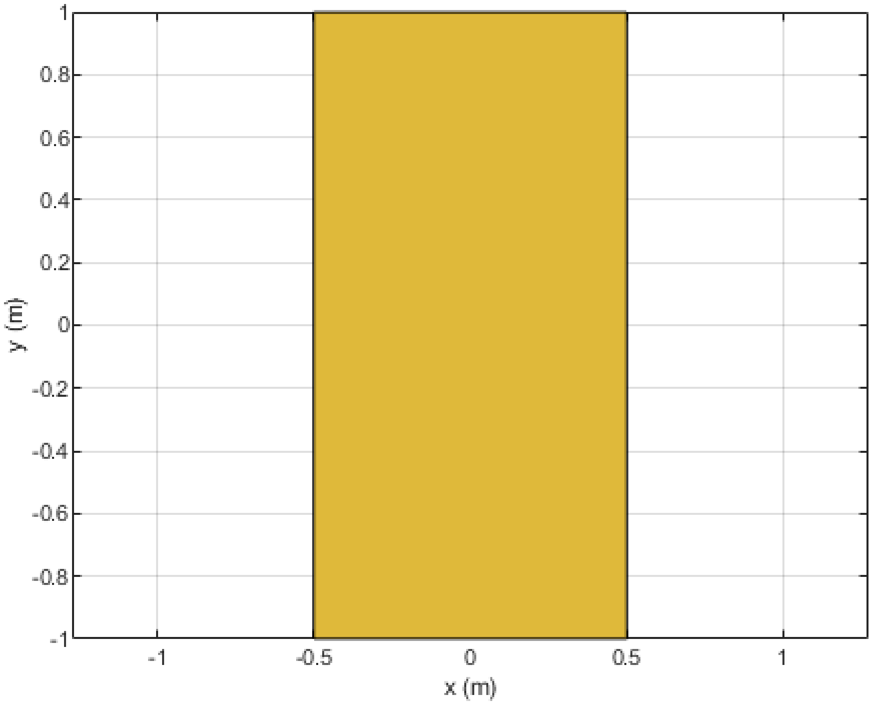
`c = scale(shape,scaling)` scales the shape by a constant factor

Examples

Scale Rectangle Shape

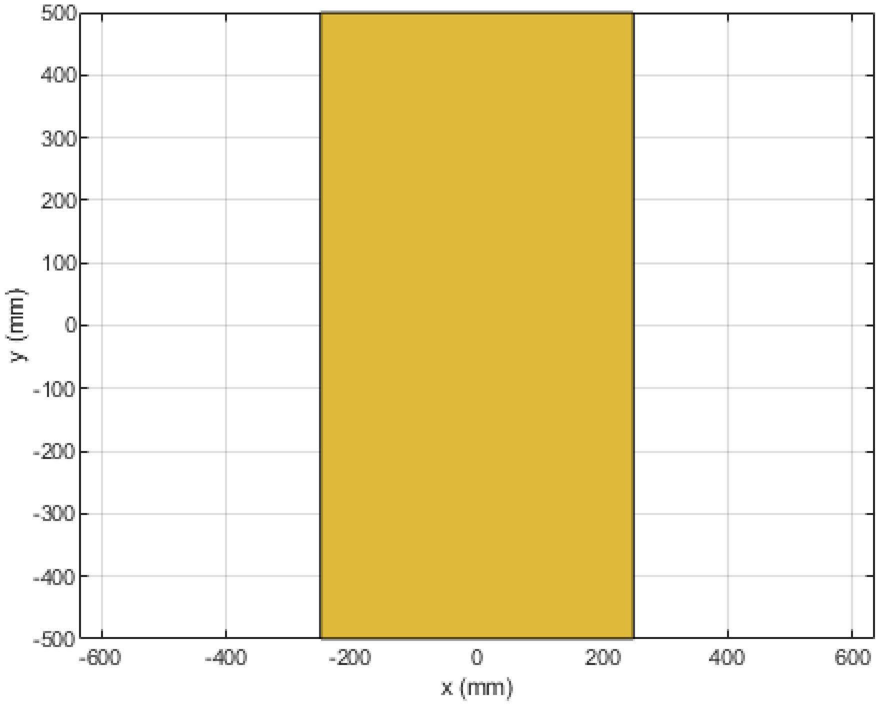
Create a rectangular shape.

```
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 handle

Shape created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

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`

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `show` | `subtract`

Introduced in R2017a

plus

Shape1 + Shape2

Syntax

```
c = plus(shape1,shape2)
```

Description

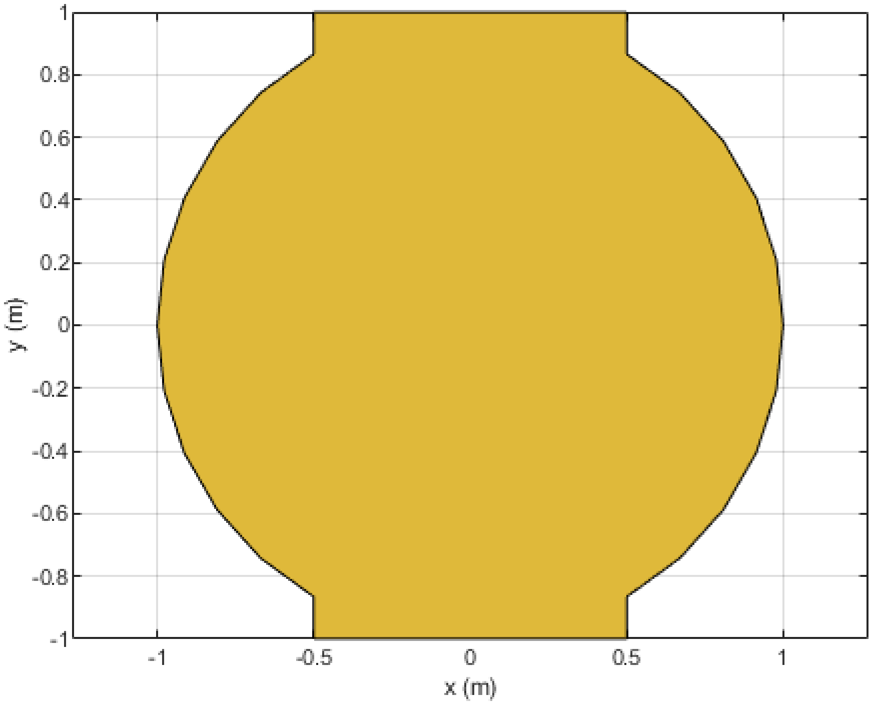
`c = plus(shape1,shape2)` calls the syntax `shape1 + shape2` to unite two shapes.

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 handle

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `rectangle1+rectangle2` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

minus

Shape1 - Shape2

Syntax

```
c = minus(shape1,shape2)
```

Description

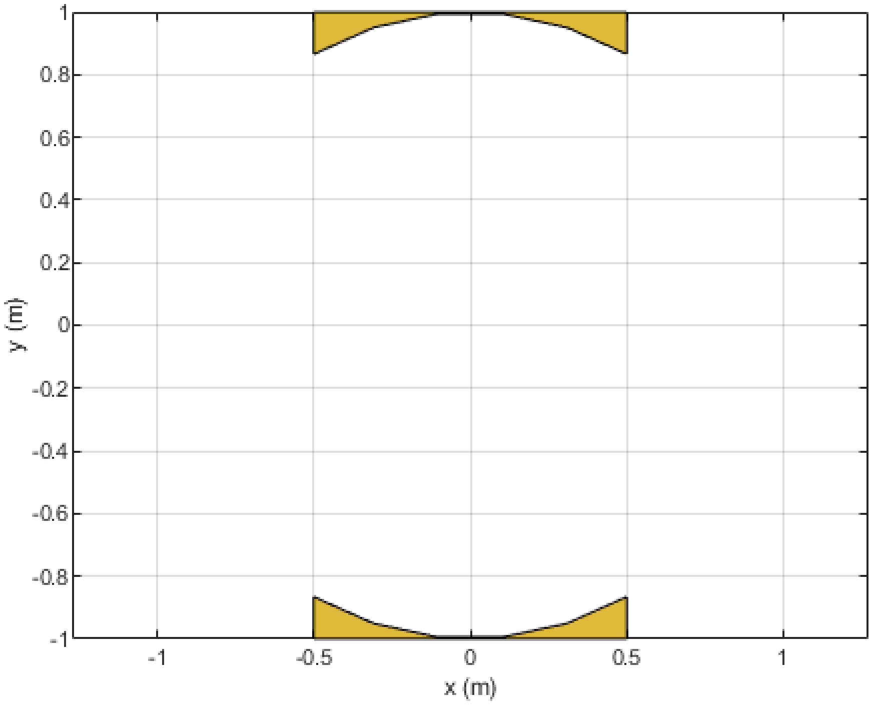
`c = minus(shape1,shape2)` calls the syntax `shape1 - shape2` to subtract two shapes.

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 handle

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `rectangle1-rectangle2` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

and

Shape1 & Shape2

Syntax

```
c = and(shape1,shape2)
```

Description

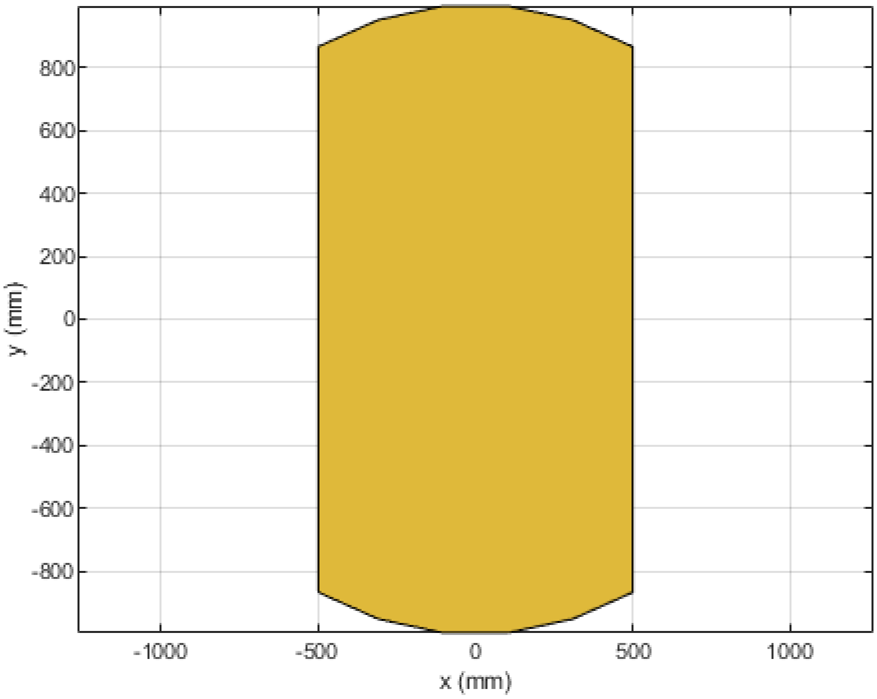
`c = and(shape1,shape2)` calls the syntax `shape1 & shape2` to intersect two shapes.

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 handle

Shapes created using custom elements and shape objects of Antenna Toolbox, specified as an object handle.

Example: `rectangle1&rectangle2` where rectangle1 and rectangle2 are shapes created using `antenna.Rectangle` object.

See Also

`add` | `area` | `intersect` | `mesh` | `plot` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `scale` | `show` | `subtract` | `translate`

Introduced in R2017a

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

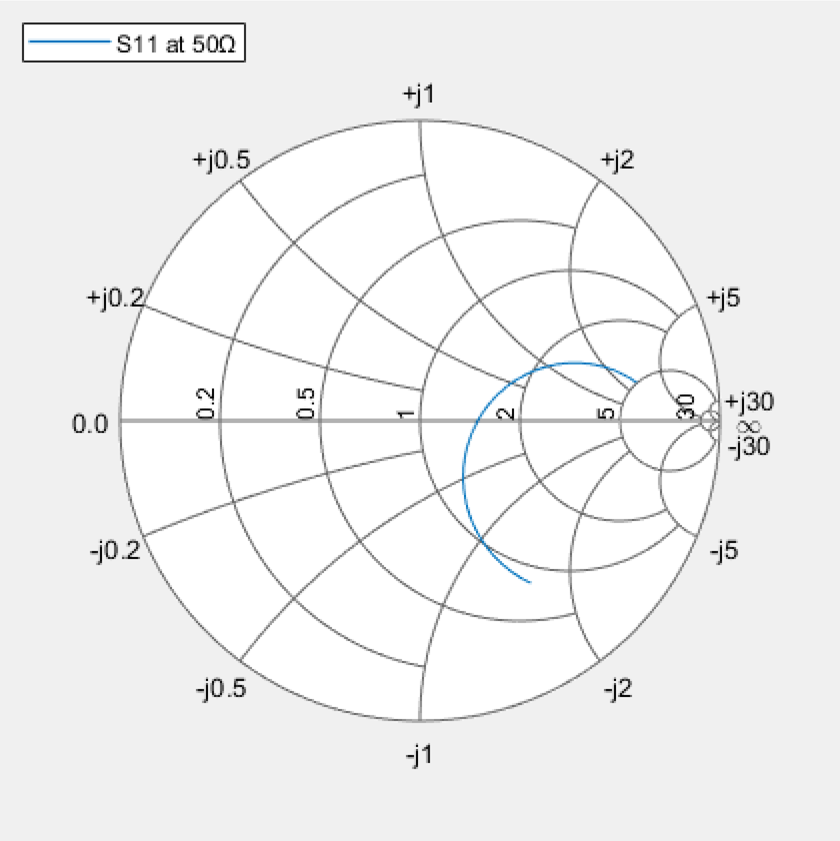
Add S-Parameter Data to an Existing Smith Plot

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.

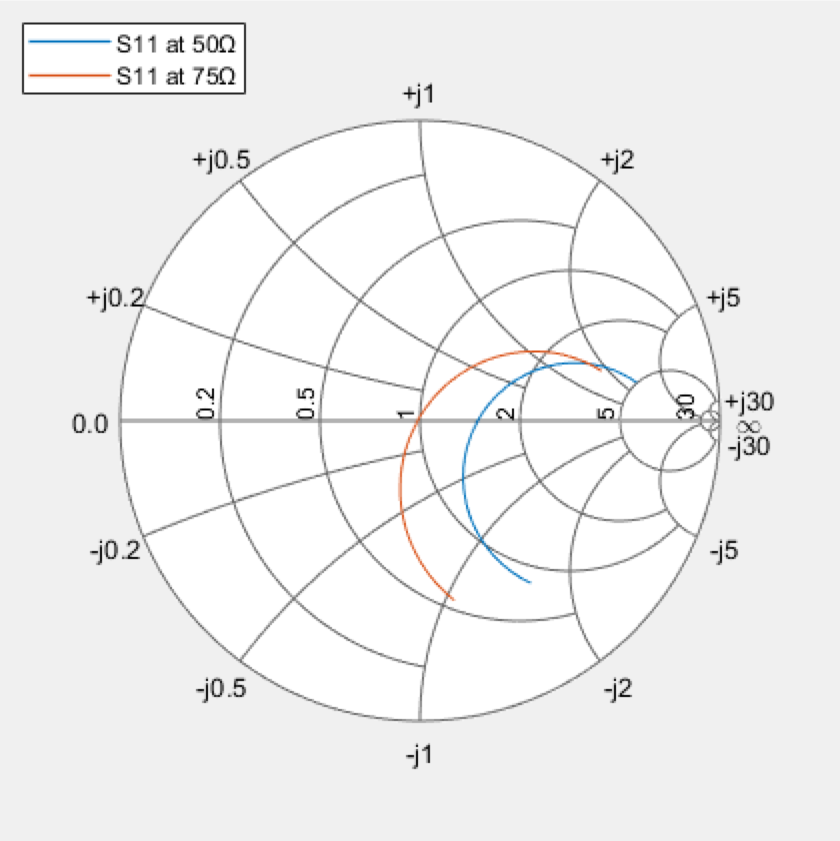
Plot S11 on a Smith plot for a reference impedance of 50 ohm.

```
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`

See Also

`replace` | `smithplot`

Introduced in R2017b

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

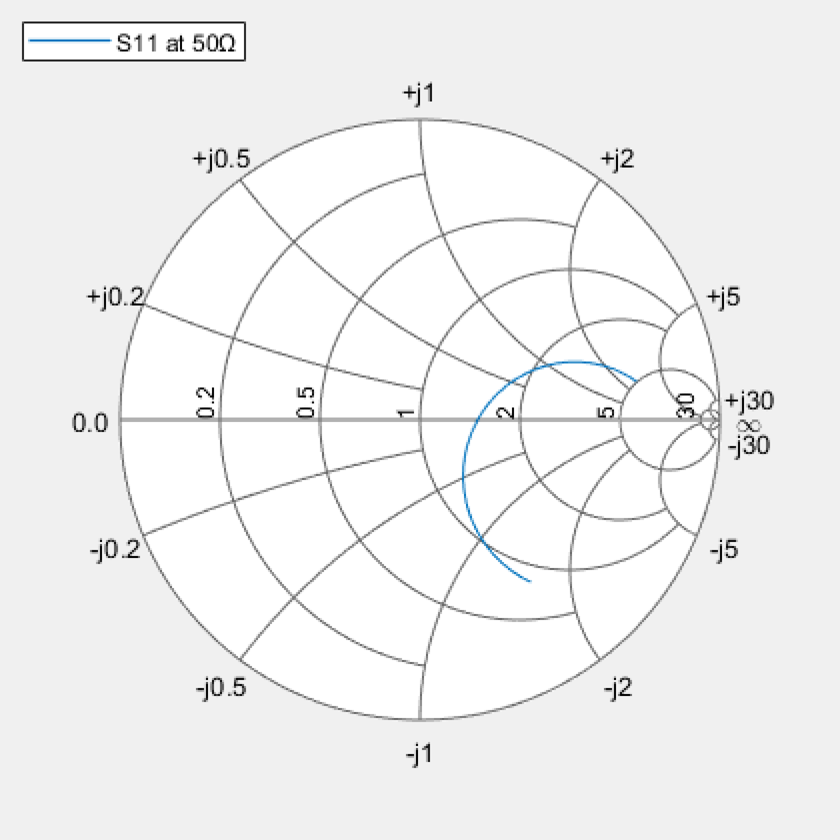
Replace S-Parameter Data on Existing Smith Chart

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.

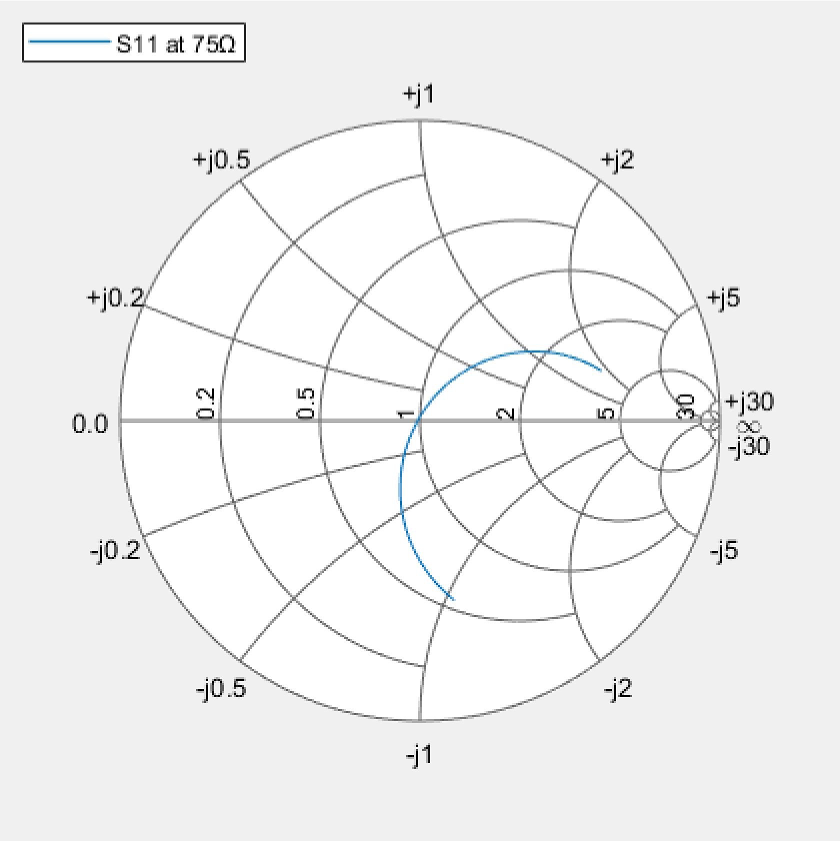
Plot S11 on a Smith chart for a reference impedance of 50 ohm.

```
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';
```



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`

See Also

add | smithplot

Introduced in R2017b

smithplot

Plot measurement data on Smith chart

Syntax

```
smithplot(data)
smithplot(frequency,data)
smithplot(ax,___)
smithplot(hnet)
smithplot(hnet,i,j)
smithplot(hnet,[i1,j1;i2,j2;....,in,jn])
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,[i1,j1;i2,j2;....,in,jn])` plots multiple parameters ($i_1, j_1, i_2, j_2, \ldots, i_n, j_n$) 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.

Note The property `'Parent'` might be used to control the location where Smith chart gets plotted.

Examples

Plot the Reflection Coefficient of a Dipole Antenna

Smith Plot of the Reflection Coefficient of a Dipole Antenna

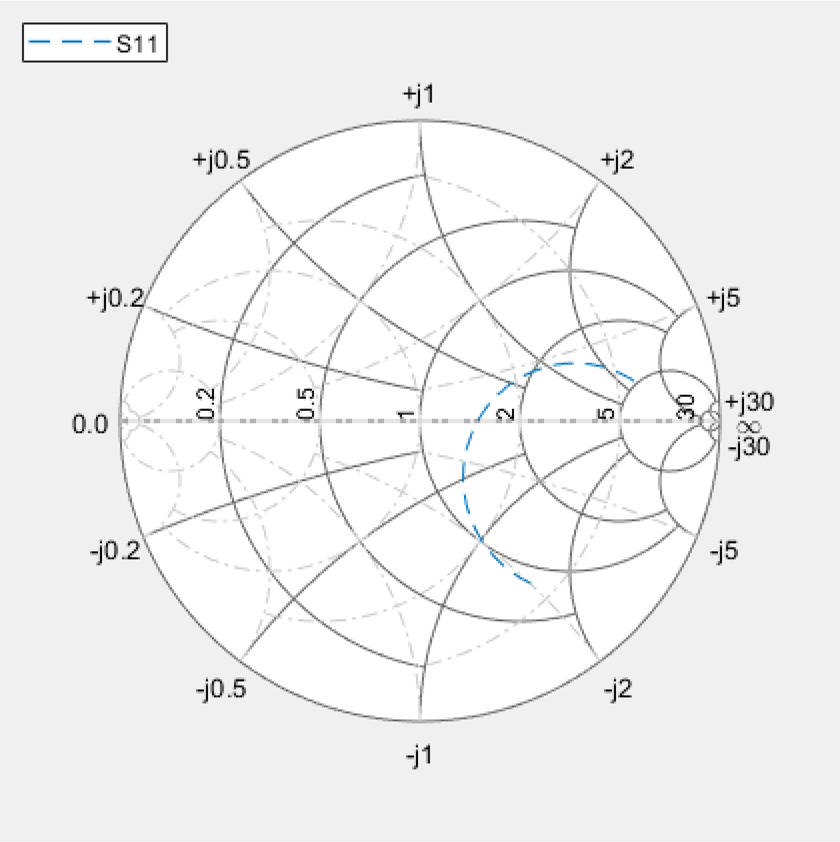
Create a strip dipole antenna on the Y-Z plane. Calculate the complex s-parameters of the dipole antenna from 60 MHz to 90 MHz, with an interval of 150 kHz.

Plot the S11 on a Smith plot.

```
d = dipole;
freq = linspace(60e6, 90e6, 200);
s = sparameters(d, freq);
hg = smithplot(s,1,1, 'GridType','ZY')
```

```
hg = 
  smithplot with properties:

         Data: [200x1 double]
    Frequency: [200x1 double]

   Show all properties, methods
```

```
hg.LineStyle = '--';
```

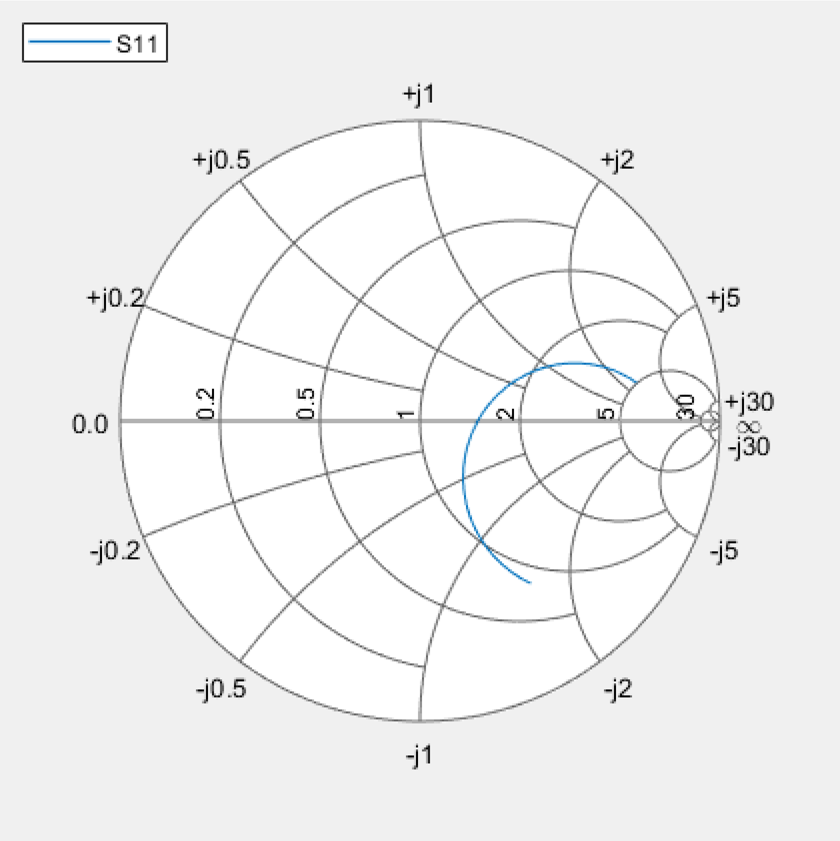


Smith Plot Interactive Menu

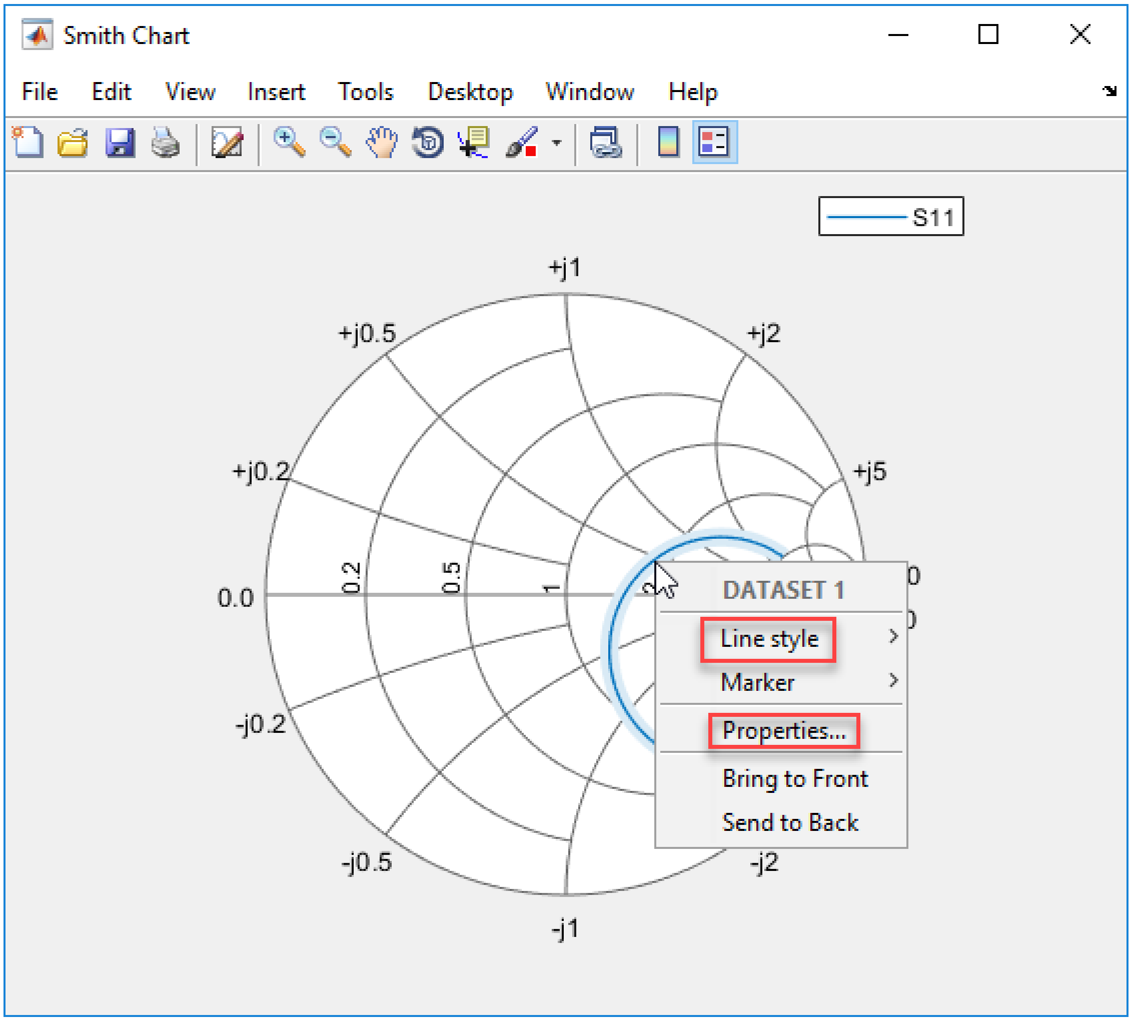
Use the Smith plot interactive menu for changing line and marker styles.

Plot the Smith plot of s-parameters of dipole d.

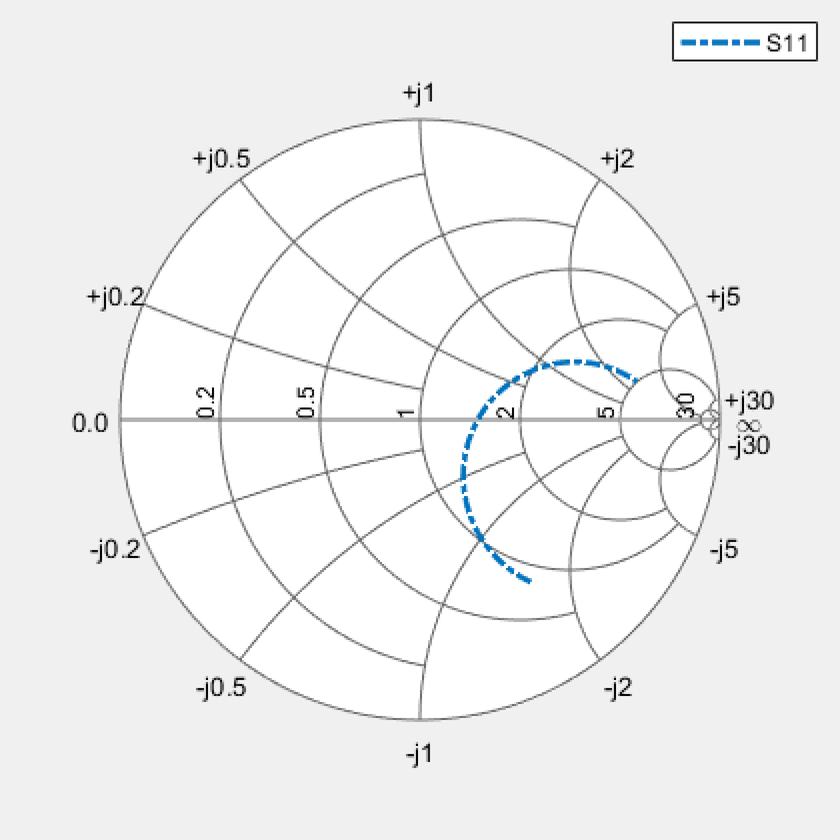
```
smithplot(s)
```



Right click on the S11 line to reveal interactive menu, `DATASET 1.` Use Line style and Properties to change the line style and width of S11 line on the Smith plot.



You can see the changes you made on the Smith plot.



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
Antenna Toolbox network parameter object

Input objects, specified as a network parameter object.

Data Types: `double`

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.
- You can use the `smithplot` interactive menu to change the line and marker styles.

See Also

`add` | `replace`

Introduced in R2017b

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])
```

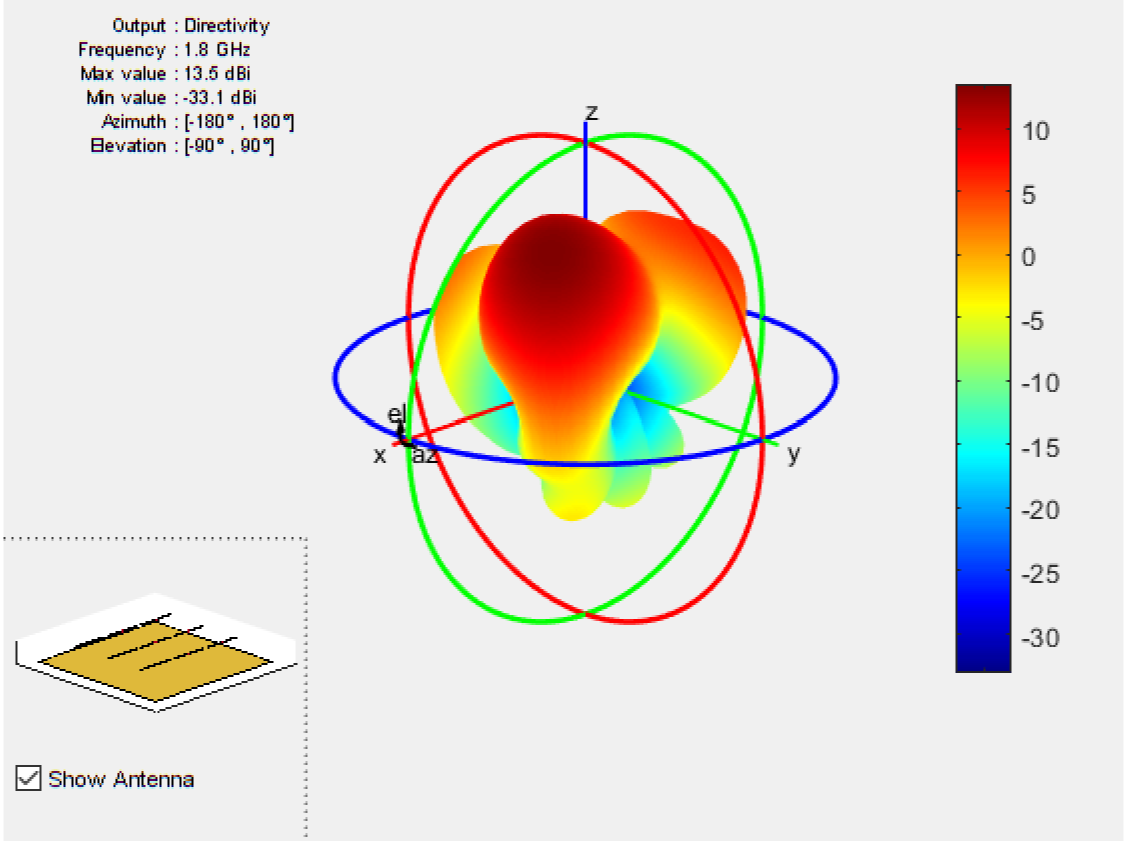
```
ps = 9×1

  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: [1×1 reflector]
              Size: [3 3]
        RowSpacing: 0.0833
     ColumnSpacing: 0.0833
           Lattice: 'Rectangular'
    AmplitudeTaper: 1
        PhaseShift: [9×1 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 : -33.1 dBi
Azimuth : [-180° , 180°]
Elevation : [-90° , 90°]

☑ Show Antenna

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:

`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.

See Also

`feedCurrent` | `pattern` | `patternMultiply`

Introduced in R2018b

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, along with the assumption 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)` provides a way to specify customization and tuning options to the pattern reconstruction method.

Examples

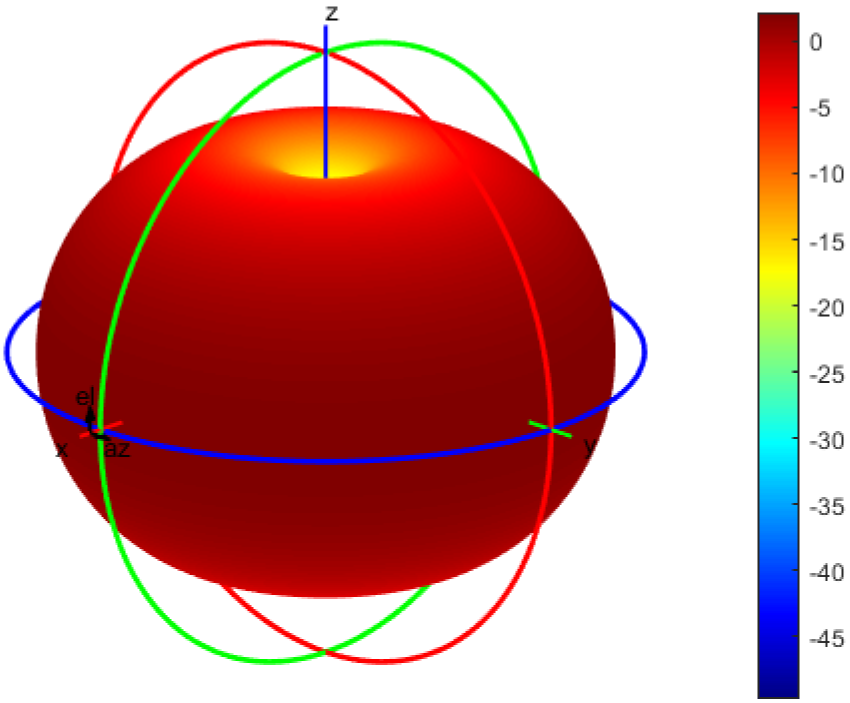
Reconstruct Pattern of Dipole Antenna from 2-D Slices

Load the `MAT` file containing the data of the dipole pattern.

```
load dipoleAntennaSlices.mat
```

Reconstruct the pattern from the data provided using the `CrossWeighted` method.

```
patternFromSlices(vertSlice,theta,horizSlice,phi,'Method','CrossWeighted')
```



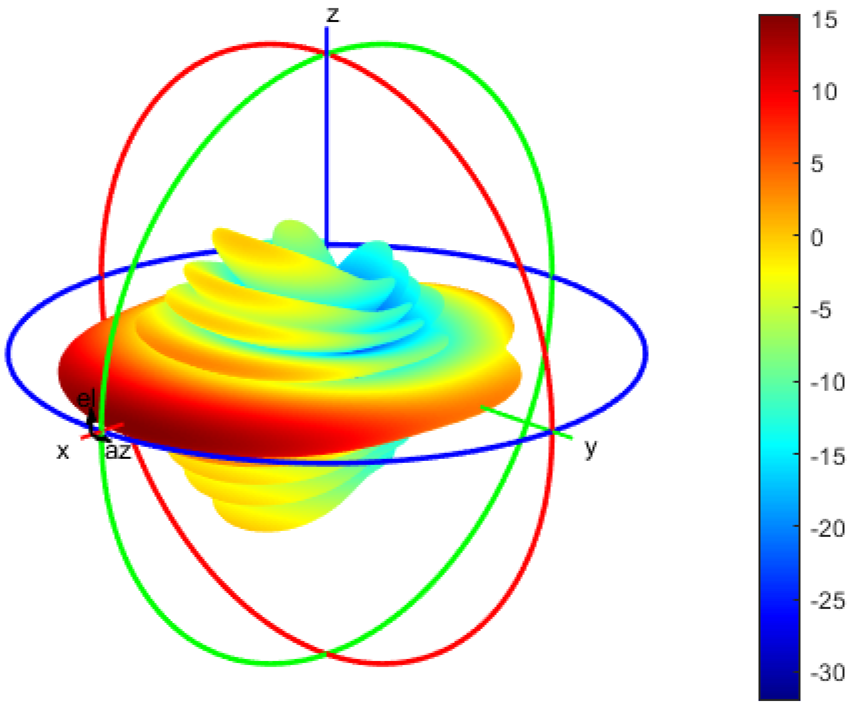
Reconstruct Pattern of Sector Antenna from 2-D Slices

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` method.

```
 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. The `numel(vertislice)` must be equal to `numel(theta)`.

Data Types: `double`

`theta` — Polar or inclination angles in spherical coordinates

real-valued vector

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.

Example: 70e6

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`.

Example: 70e6

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`

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.

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

$$G_H(\phi, \theta) = \frac{G_H(\phi) \bullet w_1 + G_V(\theta) \,.\, w_2}{\sqrt[k]{w_1^k + w_2^k}}$$

where,

- $\begin{cases} w_1(\phi, \theta) = \mathrm{vert}(\theta) \bullet [1 - \mathrm{hor}(\varphi)] \\ w_2(\phi, \theta) = \mathrm{hor}(\varphi) \bullet [1 - \mathrm{vert}(\theta)] \end{cases}$
- $G_H(\Phi)$ 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 a normalization parameter.

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

`pattern` | `patternAzimuth` | `patternCustom` | `patternElevation`

Introduced in R2019a

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

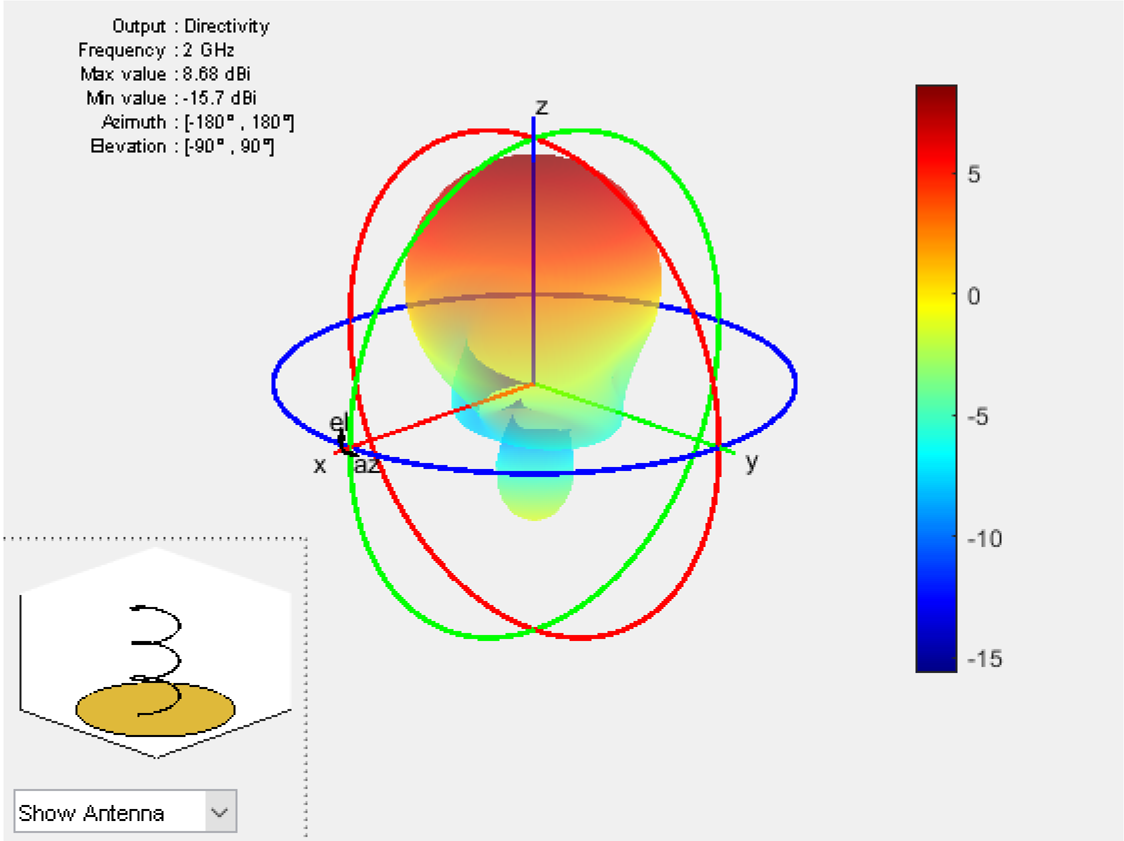
Radiation Pattern of Helix Antenna

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]
```

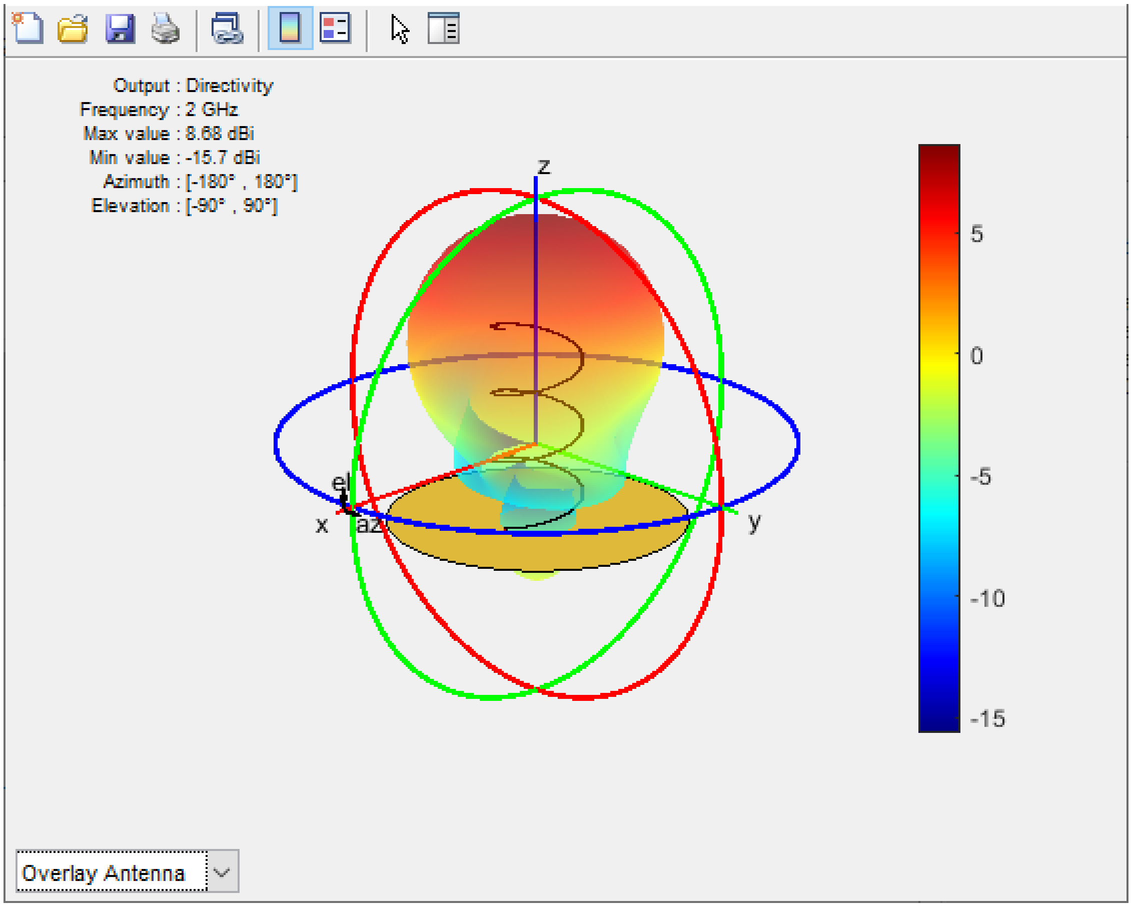
```
p.Transparency = 0.5;
ant = helix;
pattern(ant,2e9,'patternOptions',p)
```



Output : Directivity
Frequency : 2 GHz
Max value : 8.68 dBi
Min value : -15.7 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'`, `[1,0]`

Example: `patternplot.MagnitudeScale = [1,0]`

Data Types: `double`

See Also

`pattern` | `patternAzimuth` | `patternCustom` | `patternElevation`

Introduced in R2019a

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 Waveguide Antenna

Create a waveguide antenna for operation at 8 GHz and compute the impedance.

```
w = design(waveguide,8e9);
Z = impedance(w,8e9);
```

Create an STL file for the above antenna.

```
stlwrite(w,'waveguide_8GHz.stl')
```

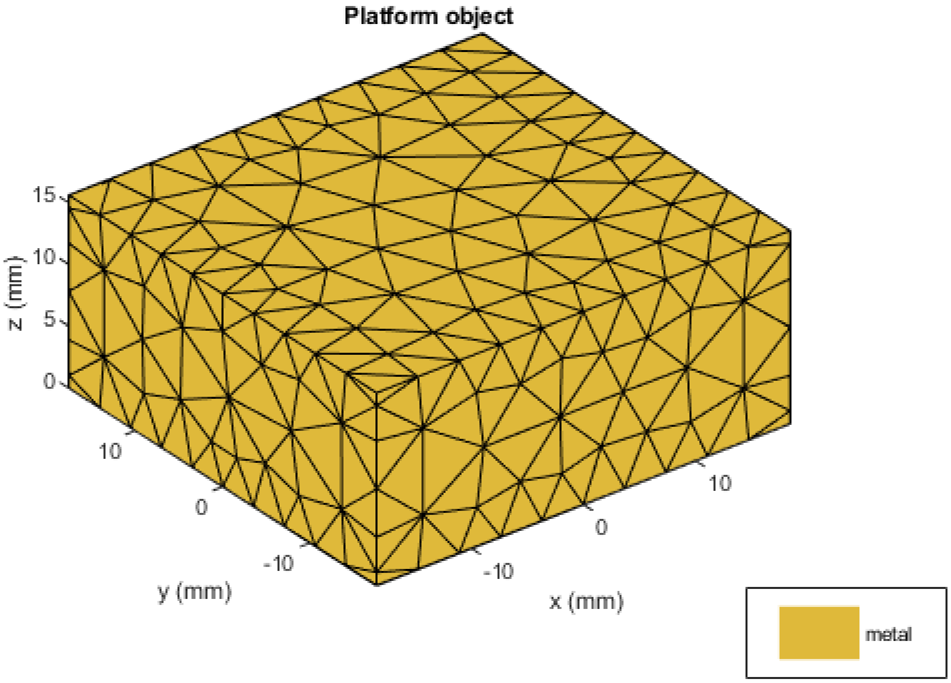
You will see the `waveguide_8GHz.stl` file in your current folder.

Load `waveduide_8GHz.stl` and visualize the platform.

```
plat = platform('FileName','waveguide_8GHz.stl','Units','m')
```

```
plat = 
  platform with properties:

         FileName: 'waveguide_8GHz.stl'
            Units: 'm'
    UseFileAsMesh: 0
             Tilt: 0
         TiltAxis: [1 0 0]
```

```
show(plat)
```



Input Arguments

`objname` — Antenna or array object
antenna or array handle

Antenna or array object, specified as an antenna or array handle.

`filename` — Name of STL file
character vector

Name of STL file, specified as a character vector in STL format.

See Also

`meshconfig` | `platform` | `show`

Introduced in R2019a

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 5-312.

`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.

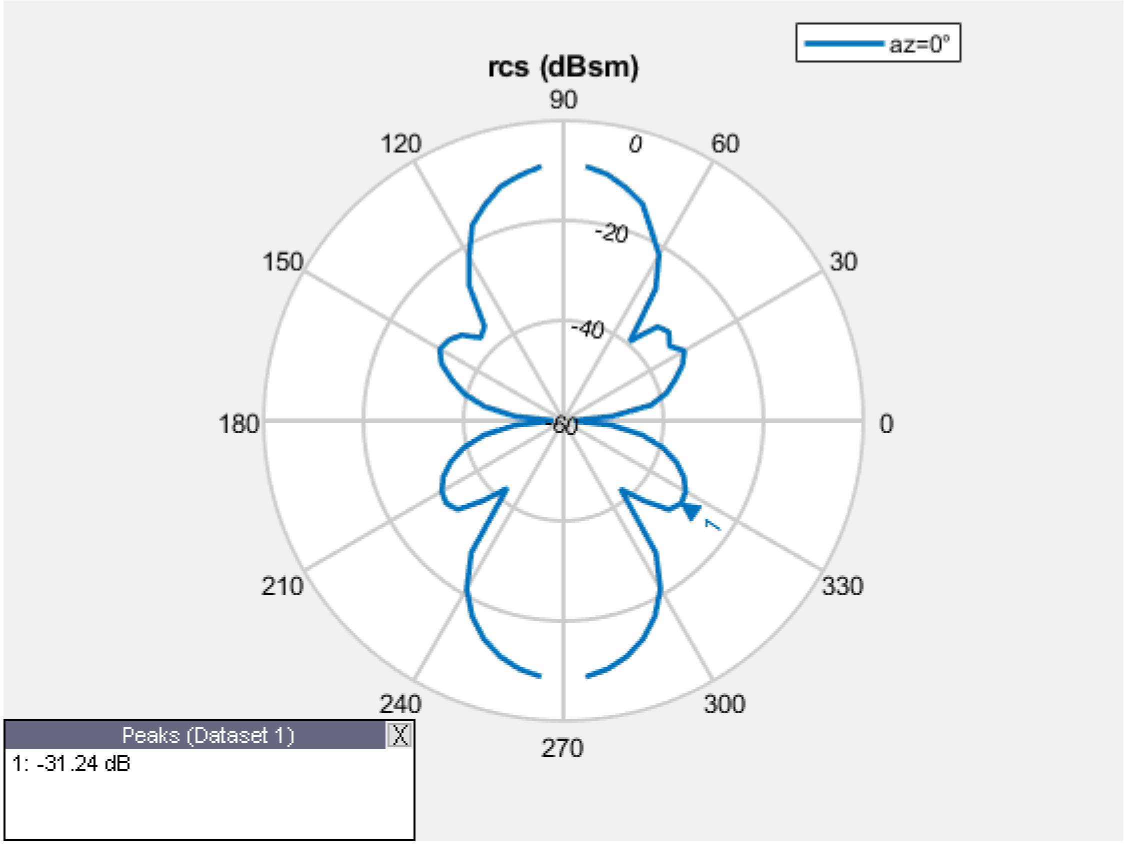
`[rcsval,azimuth,elevation] = rcs( ___ ,Name,Value)` returns the RCS value with additional properties specified using one or more Name, Value pair arguments. This parameter can be used with any of the input arguments from the previous syntaxes.

Examples

RCS of Helix

Create a default helix antenna and plot the RCS at 2 GHz.

```
ant = helix;
rcs(ant,2e9)
```

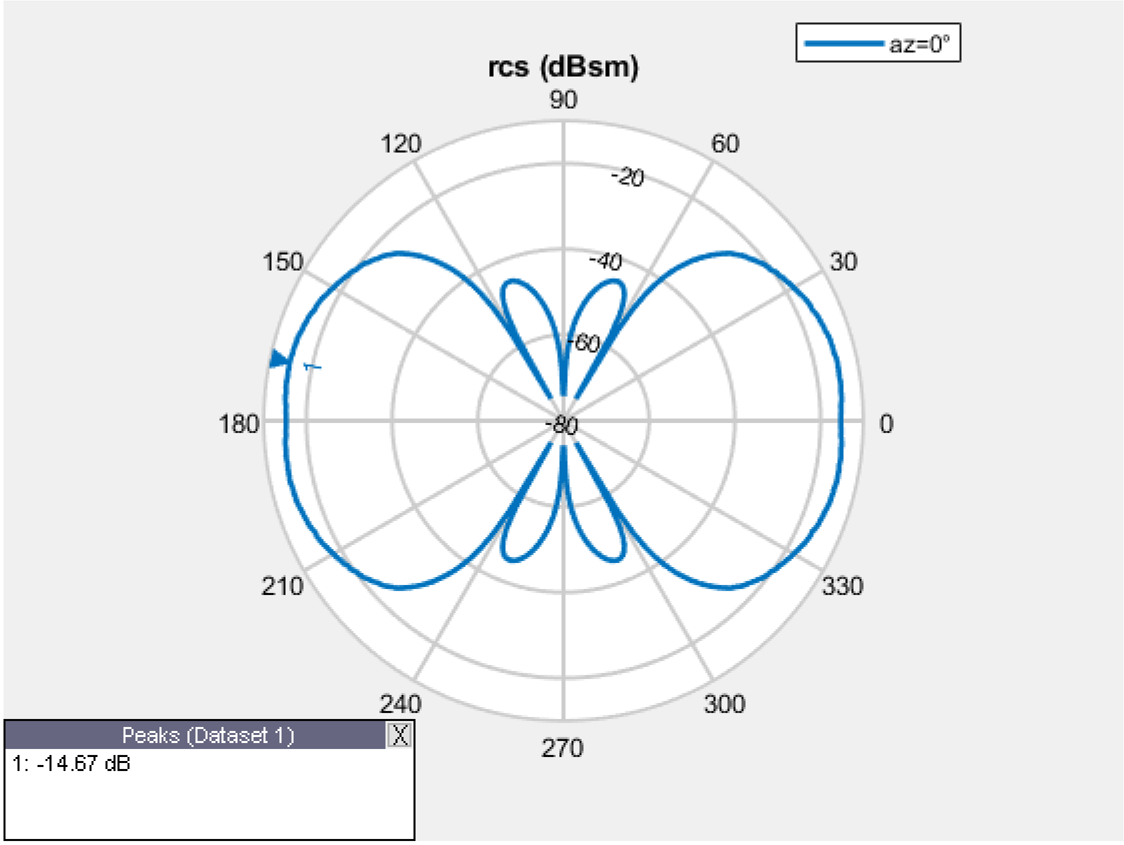


1: -31.24 dB

RCS of Linear Array

Create a default linear array and plot the RCS at 75 MHz in the elevation pane.

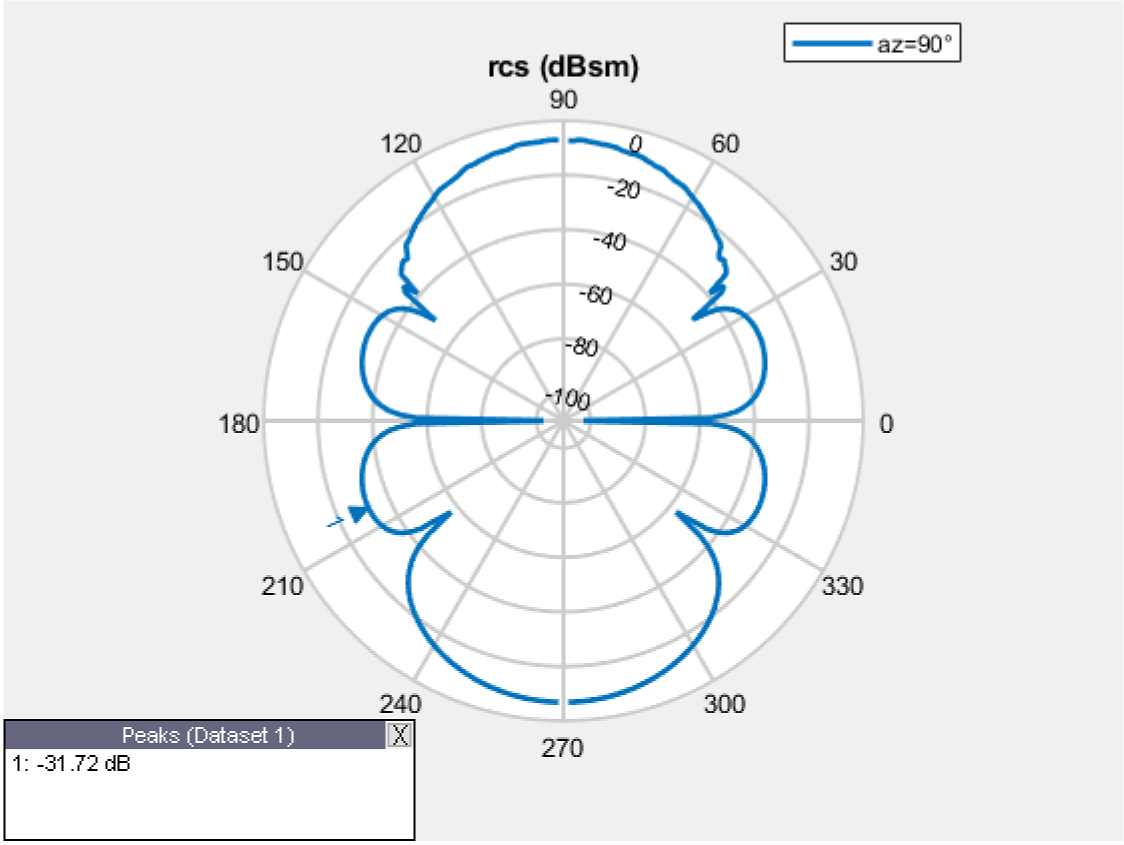
```
array = linearArray;
rcs(array,75e6,0,0:1:360)
```



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)
```



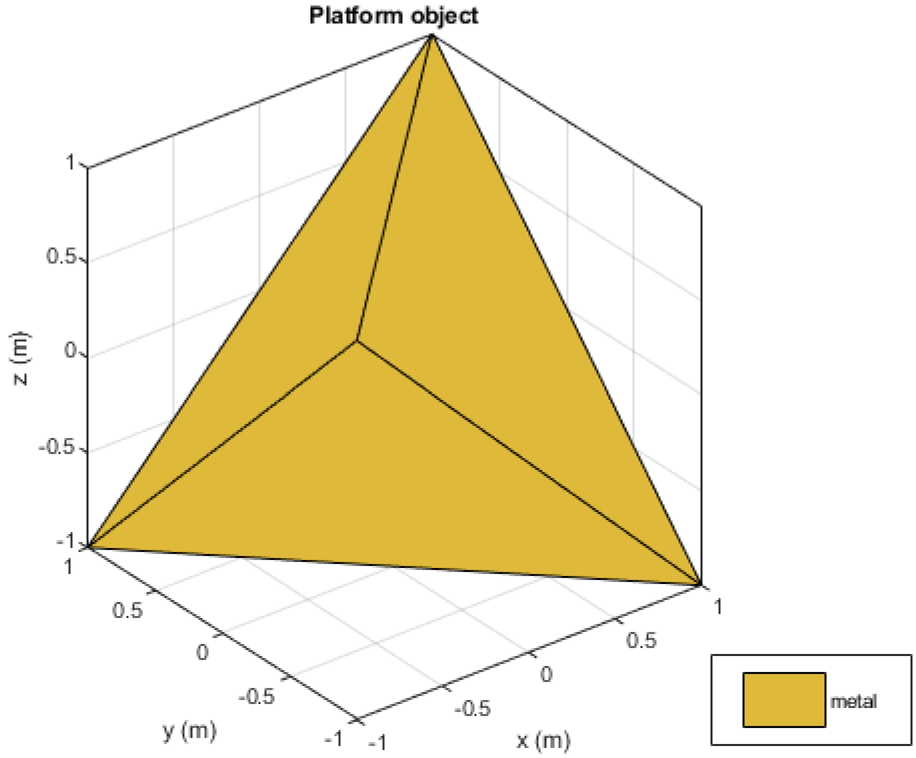
Peaks (Dataset 1)

1: -31.72 dB

RCS of Tetrahedron Platform

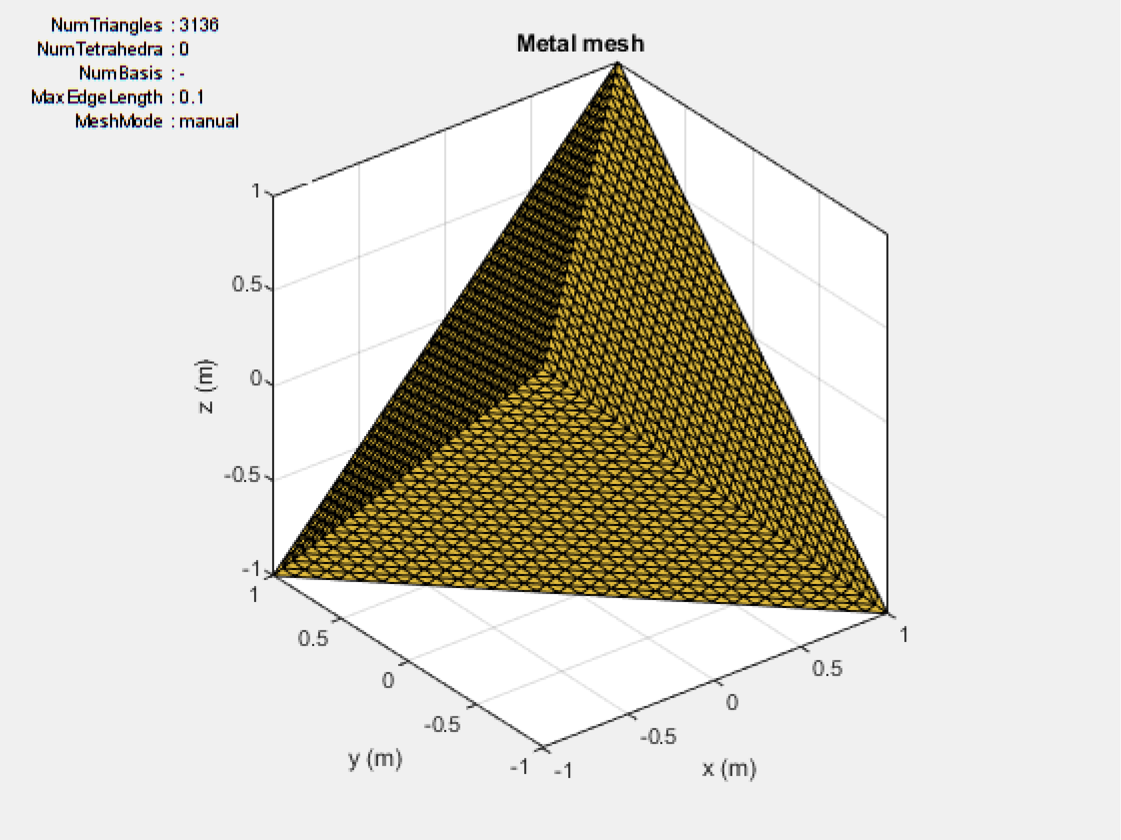
Create a tetrahedron platform from an STL file.

```
p = platform;
p.FileName = 'tetrahedra.stl';
p.Units = 'm';
figure
show(p)
```



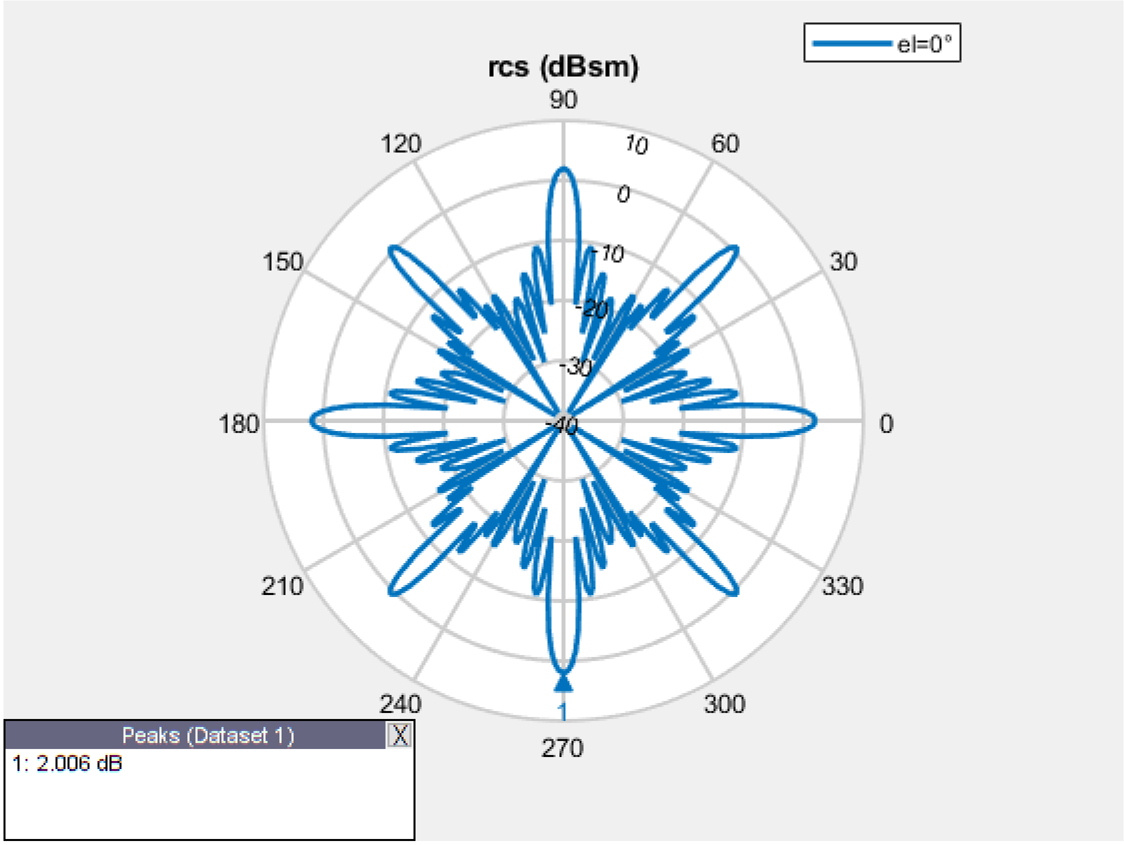
Mesh the platform with edge length of 0.1

```
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

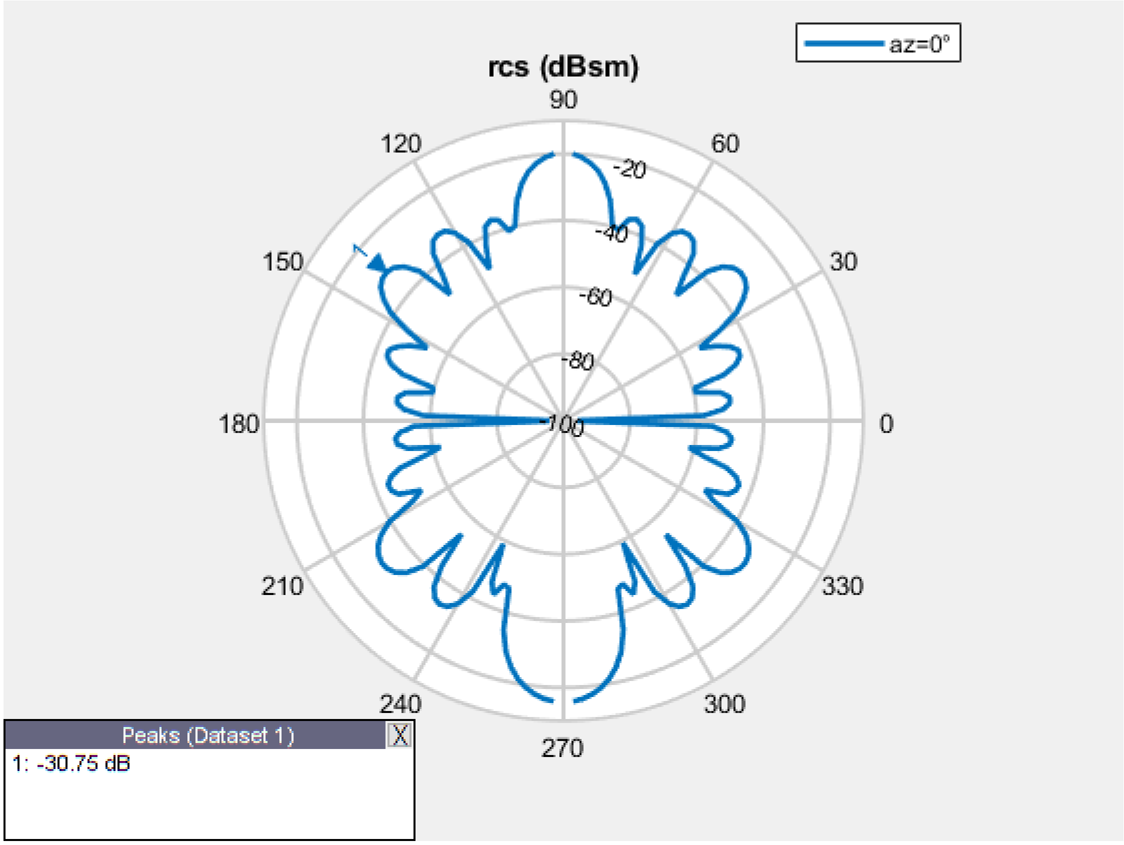
RCS of Corner Reflector

Create a corner reflector-bakced antenna.

```
f = 2e9;
c = design(reflectorCorner,750e6);
```

Plot the RCS in the elevation plane.

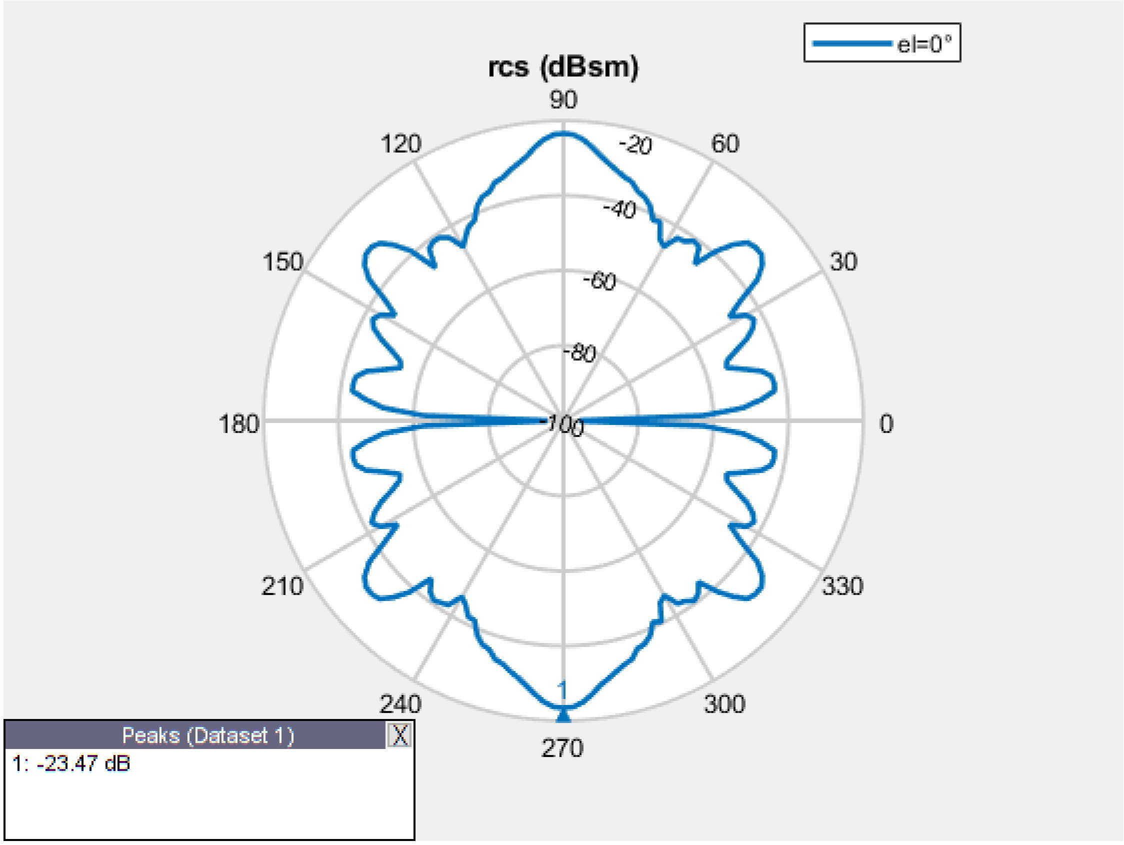
```
figure
rcs(c,f,0,0:2:360)
```



1: -30.75 dB

Plot the RCS in the azimuth plane.

```
figure
rcs(c,f,0:2:360,0)
```



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`

`TransmitDirection` — Transmit wave direction

2-by-*N* real matrix

Transmit wave direction, specified as the comma-separated pair consisting of `'TransmitDirection'` and a 2-by-*N* real matrix representing azimuth and elevation pairs, with each element unit in degrees.

Example: `'TransmitDirection',[30;60]`

Data Types: `double`

`ReceiveDirection` — Receive wave direction

2-by-*M* real matrix

Receive wave direction, specified as the comma-separated pair consisting of `'ReceiveDirection'` and a 2-by-*M* real matrix representing azimuth and elevation pairs, with each element unit in degrees.

Example: `'ReceiveDirection',[30;60]`

Data Types: `double`

`Solver` — Solver for RCS analysis

`'PO'` (default) | `'MOM'`

Solver for RCS analysis, specified as the comma-separated pair consisting of `'Solver'` and `'PO'` (physical optics) or `'MOM'` (method of moments).

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.

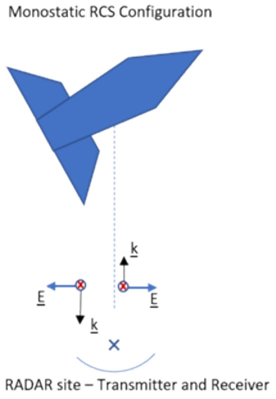
RCS is calculated using two typical configurations:

- Monostatic
- Bistatic

By default, the `rcs` function calculates a monostatic RCS. To calculate a bistatic RCS, restrict the `'TransmitDirection'` to 2-by-1.

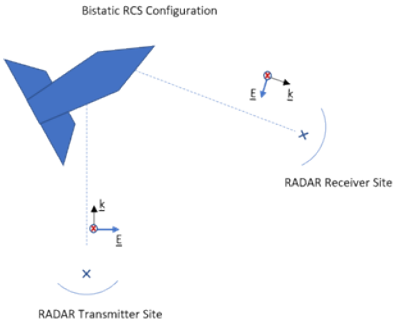
Monostatic RCS

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 colocated.


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.

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`

Introduced in R2019b

rectspirallength2turns

Calculate number of turns for specified arm length in rectangular spiral antenna

Syntax

```
Nturns = rectspirallength2turns(ant,reqtotalarmlength)
```

Description

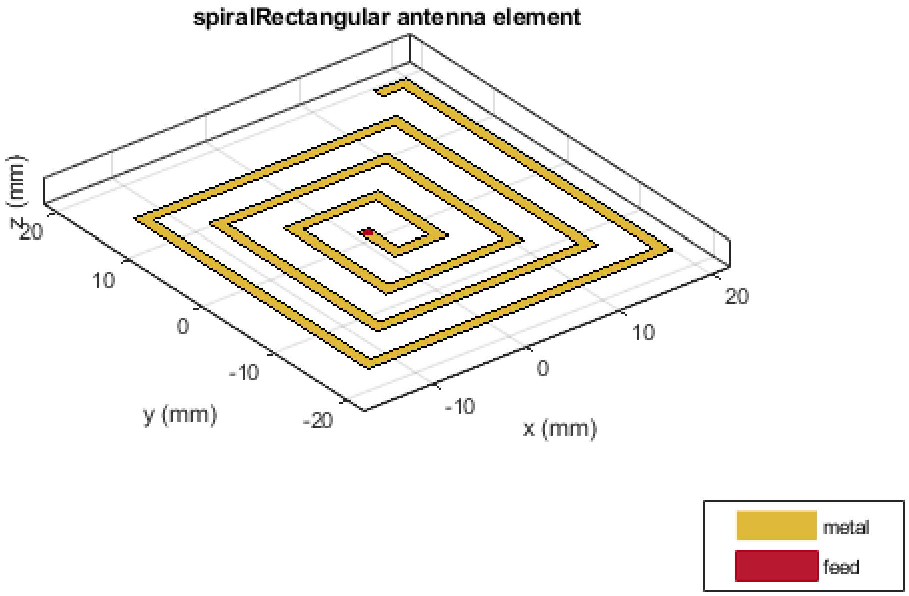
`Nturns = rectspirallength2turns(ant,reqtotalarmlength)` calculates the equivalent number of turns for a specified total arm length in a rectangular spiral antenna.

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.

See Also

`spiralRectangular`

Introduced in R2020a

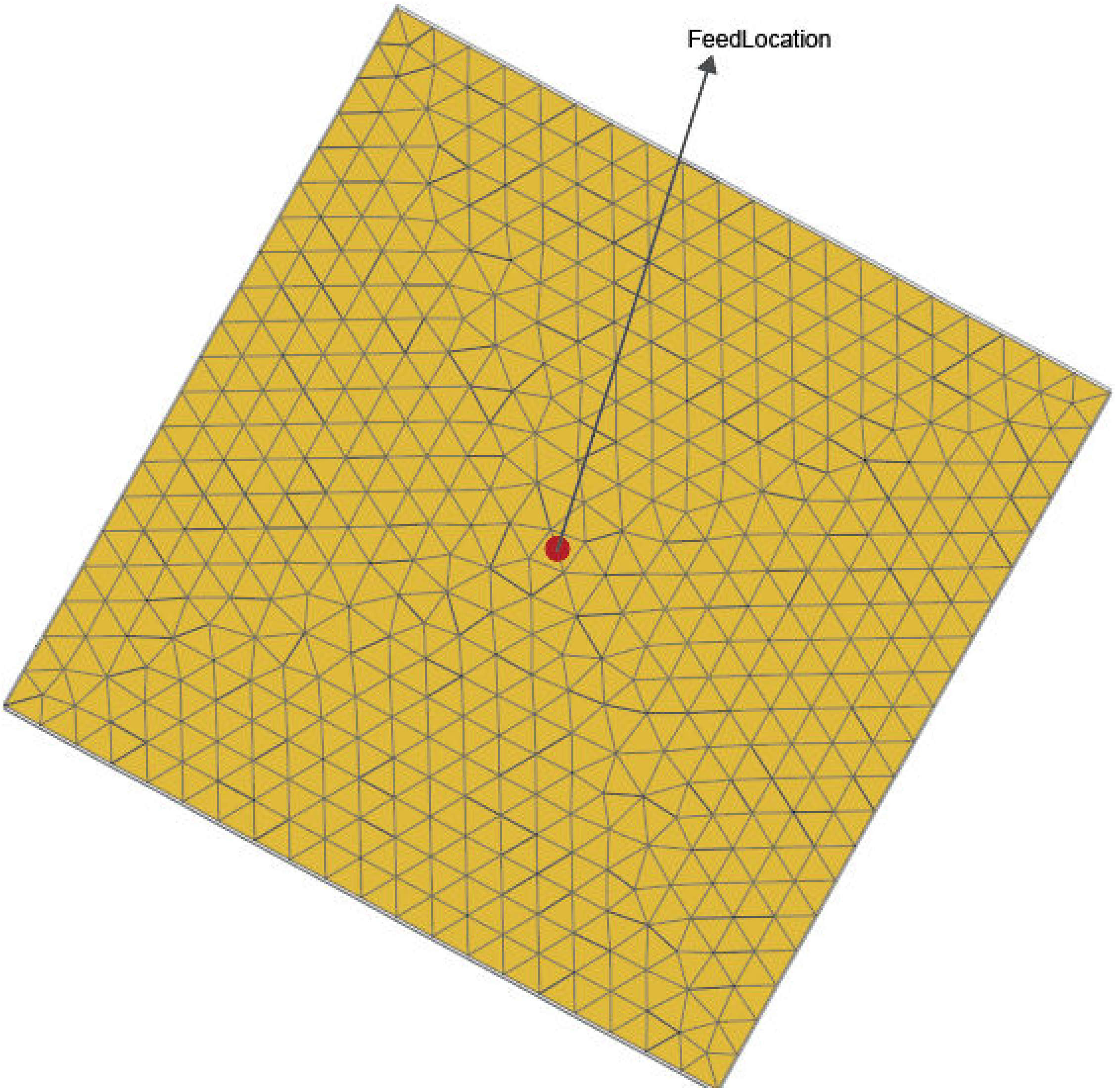
createFeed

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 `customAntennaStl`object. 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]
```

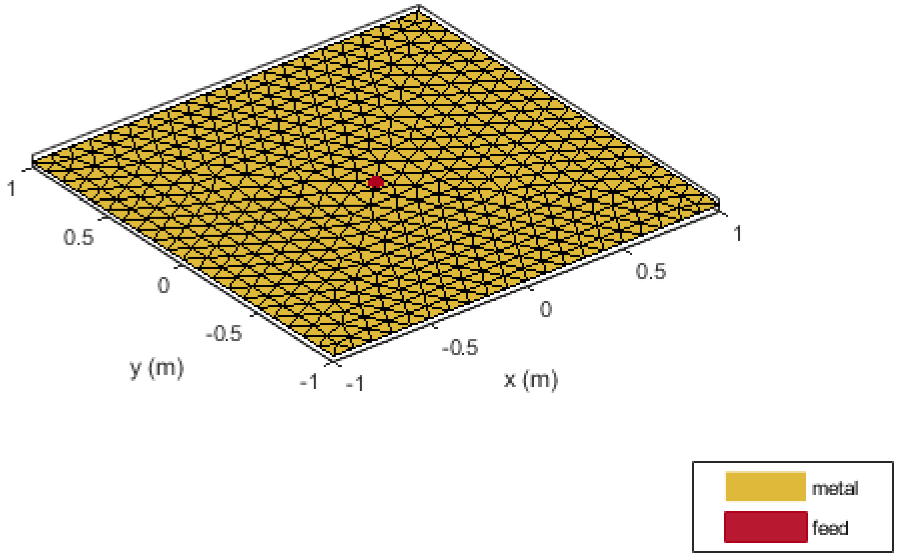
Specify the file name of the STL file to determine the antenna structure.

```
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.

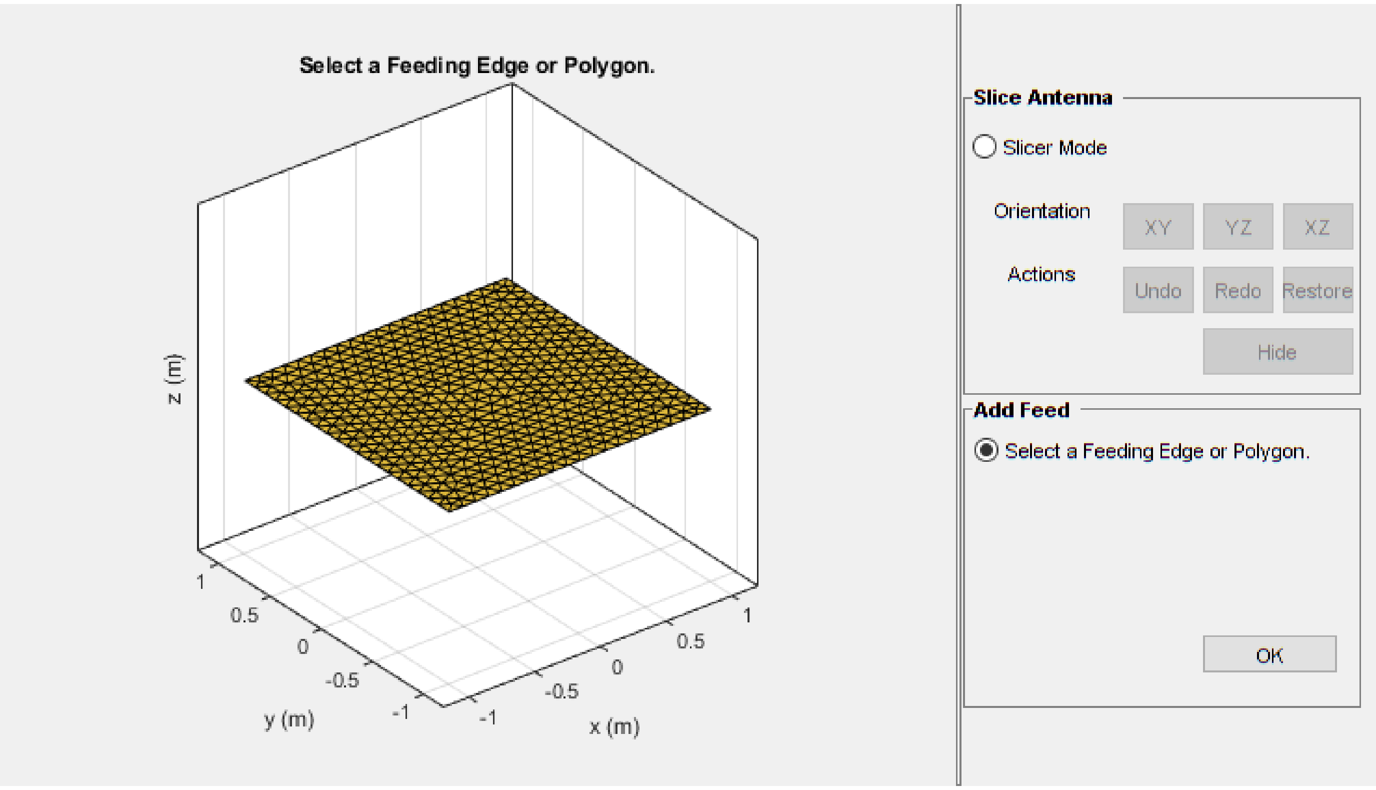
```
ant= customAntennaStl;
```

Import the STL file.

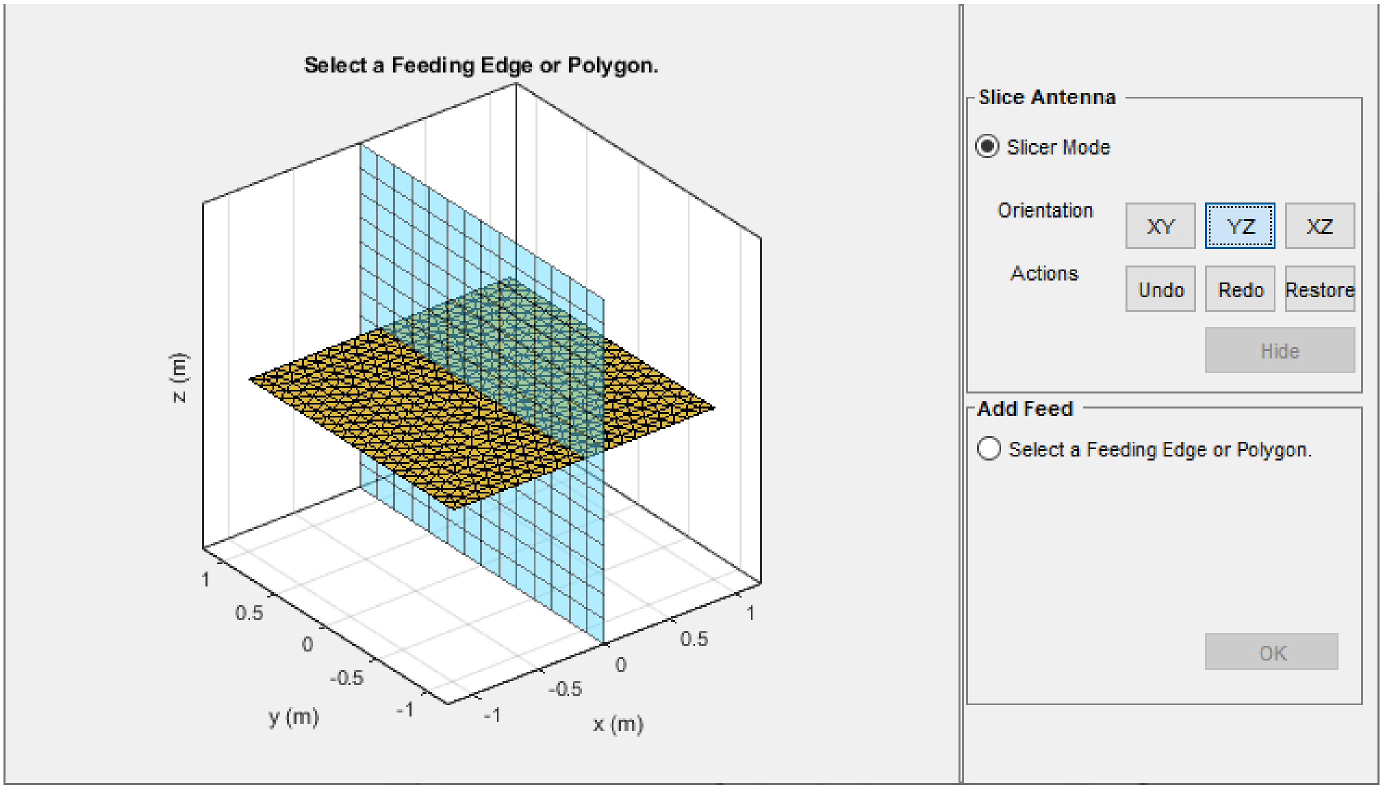
```
ant.FileName = 'plateMesh.stl';
```

Open the UI figure window.

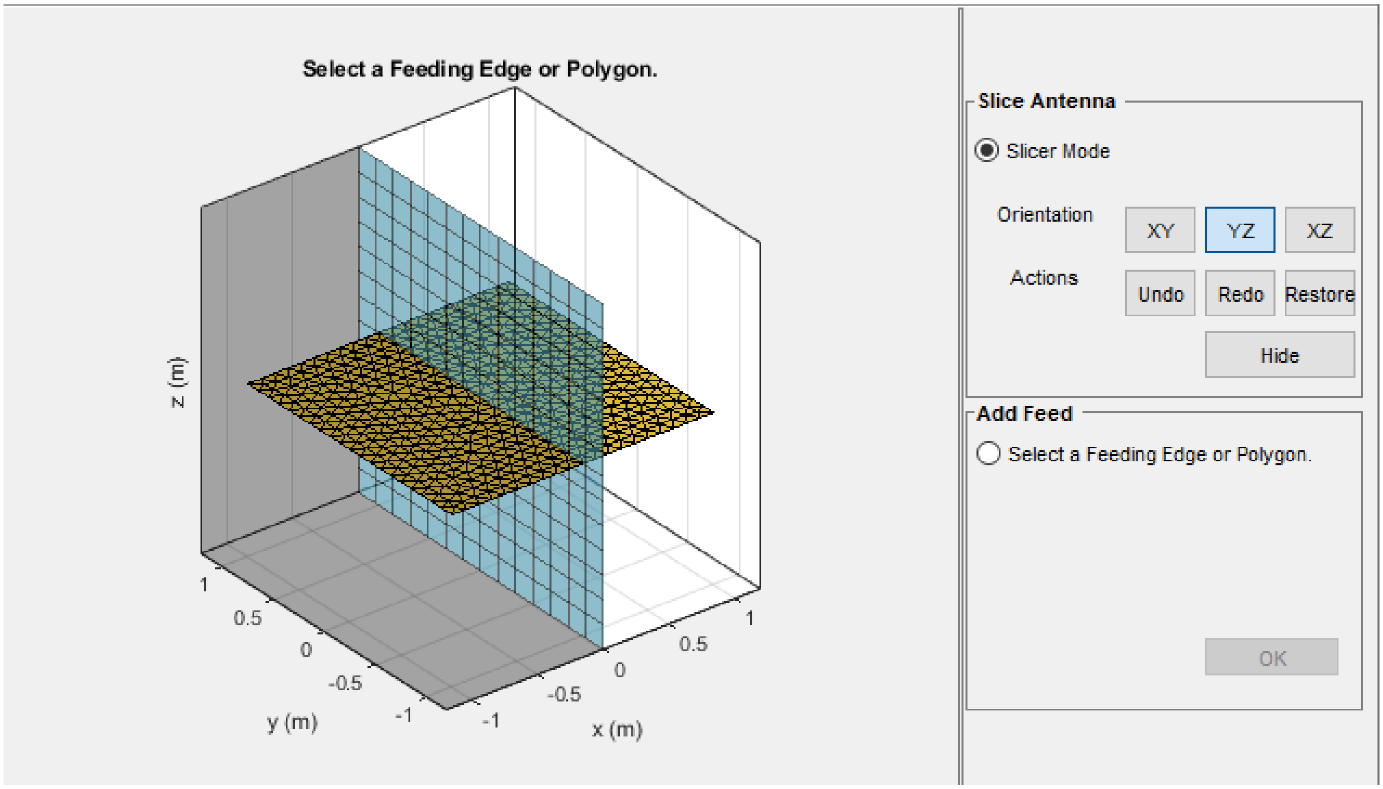
```
createFeed(ant);
```



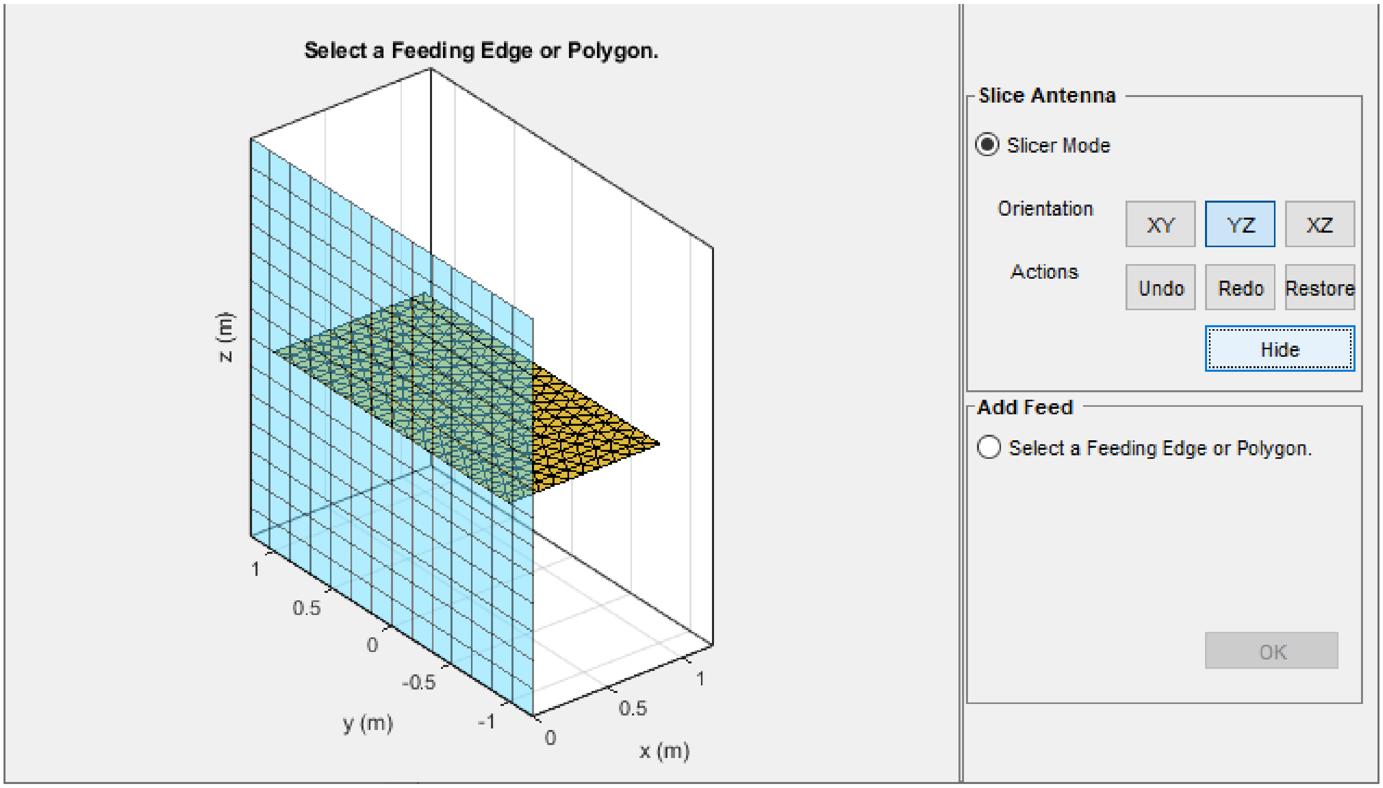
The UI figure window consists of two panes, **Slice Antenna** and the **Add Feed** pane. Select the **Slicer Mode**, then click **YZ** to select that as the plane along which to slice your antenna.



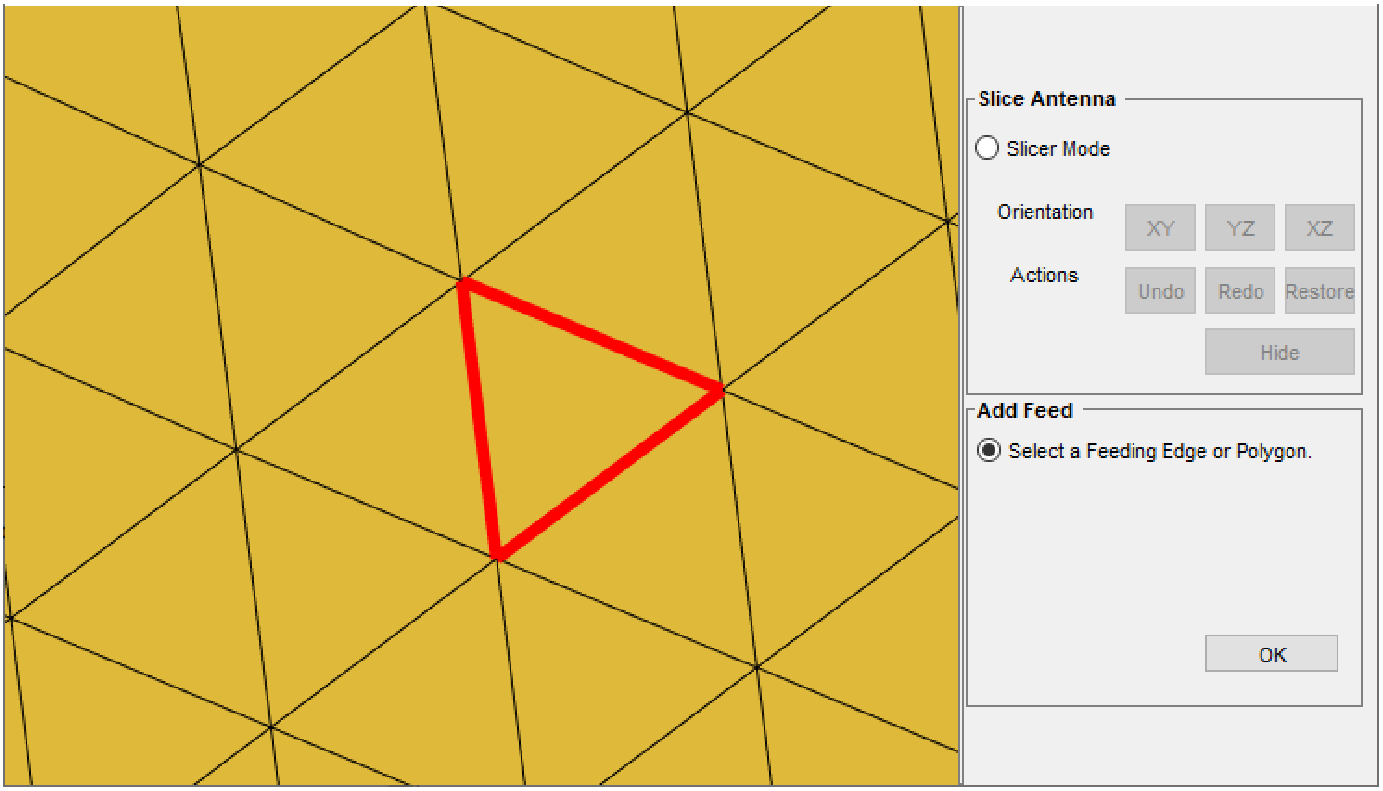
Select the region you want to hide and then click **Hide** to hide the selected region.



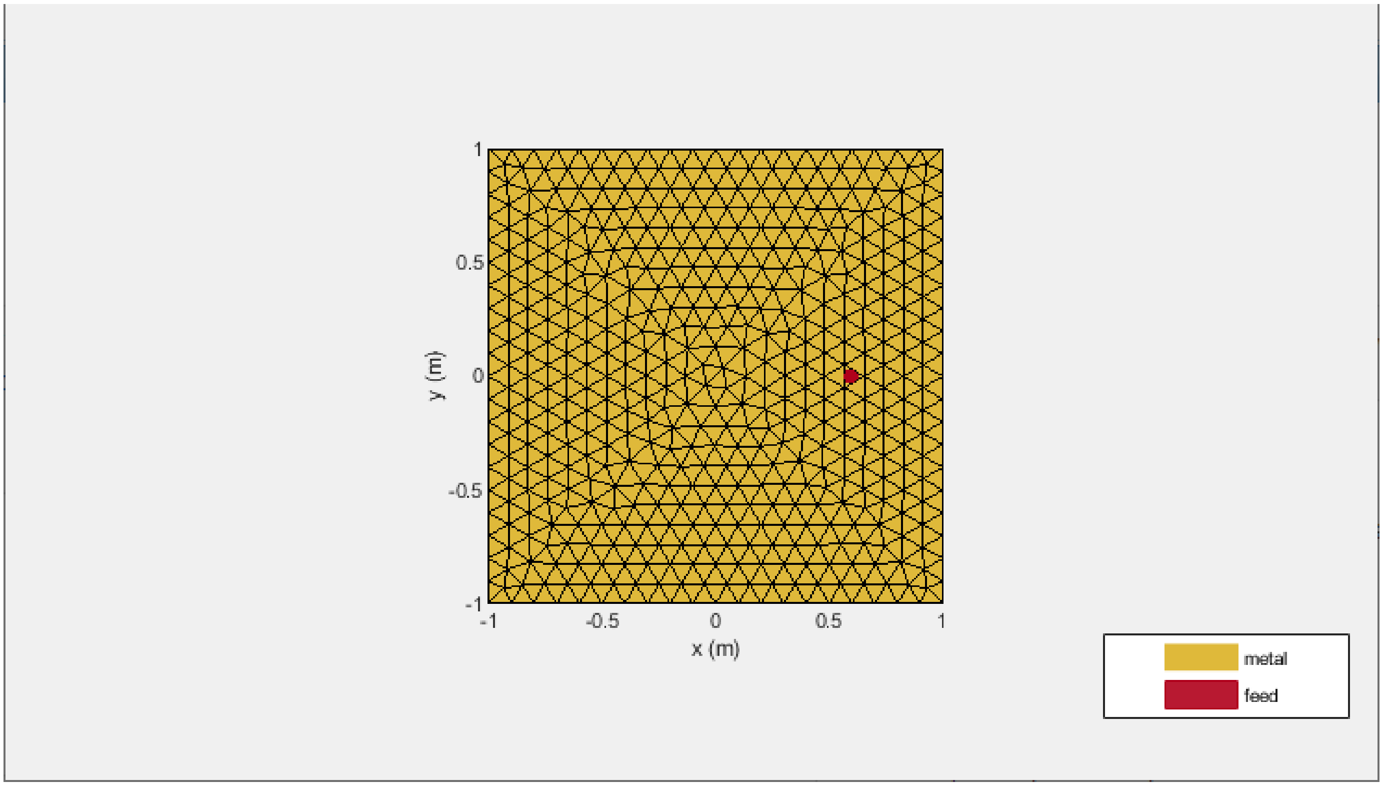
Repeat the process until you reach the region of interest.



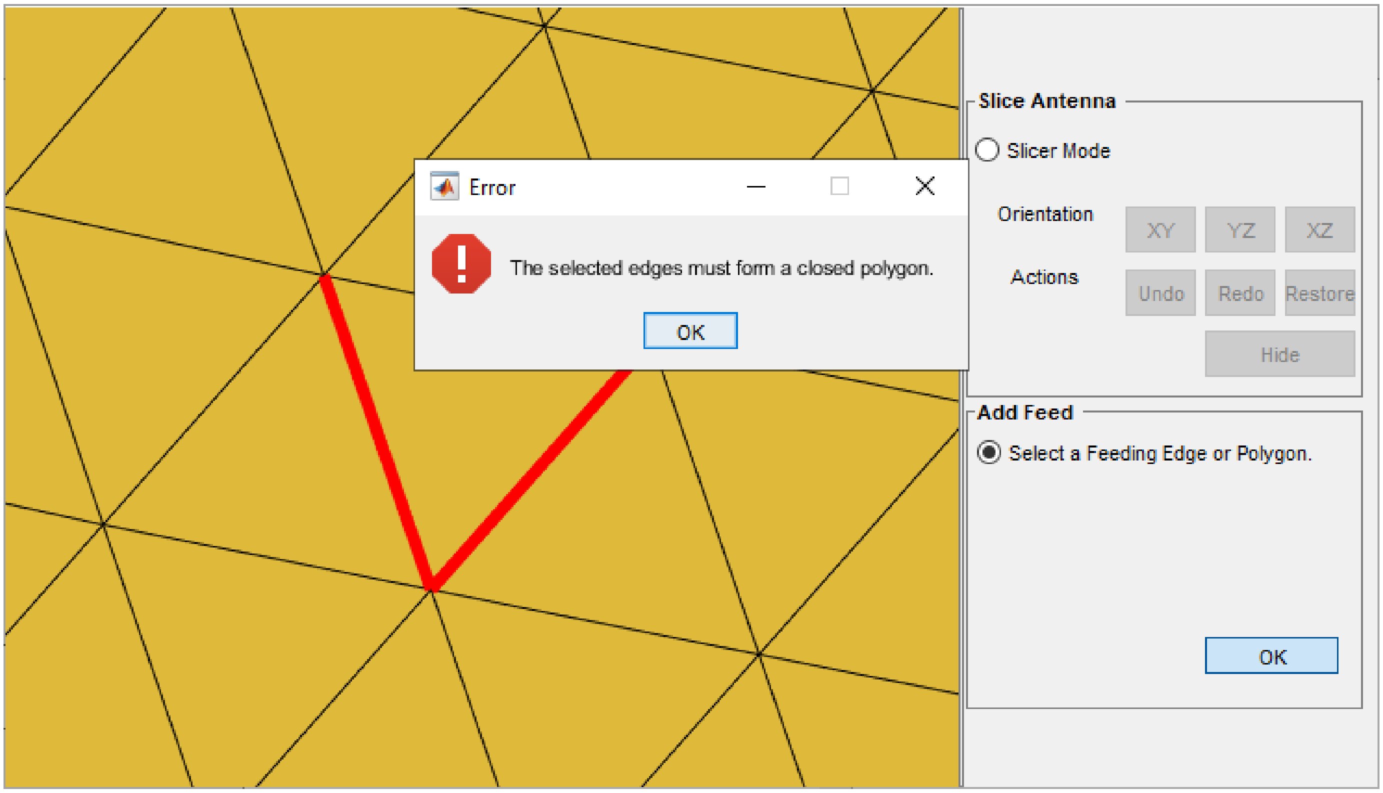
Select **Select a Feeding Edge or Polygon** under the **Add Feed** pane to select the edges to form a closed polygon. Click **OK** to define the selected edges as the feeding edges.



The feed location is displayed.



The selected edges must be connected to other edges, else UI figure window will display an error.



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);`

See Also

`customAntennaStl` | `returnLoss` | `sparameters`

Introduced in R2020a

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

See Also

`cylinder2strip`

Introduced in R2020a

6

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

'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

'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'`

Modifier	Description	Example
`\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'`

Modifier	Description	Example
\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 specifer 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'	🟦

RGB Triplet	Hexadecimal Color Code	Appearance
`[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'`	🟨

RGB Triplet	Hexadecimal Color Code	Appearance
`[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.

Value	Description
`'o'`	Circle
`'+'`	Plus sign
`'*'`	Asterisk
`'.'`	Point
`'x'`	Cross
`'square'` or `'s'`	Square
`'diamond'` or `'d'`	Diamond
`'^'`	Upward-pointing triangle
`'v'`	Downward-pointing triangle
`'>'`	Right-pointing triangle
`'<'`	Left-pointing triangle
`'pentagram'` or `'p'`	Five-pointed star (pentagram)
`'hexagram'` or `'h'`	Six-pointed star (hexagram)
`'none'`	No markers

`'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'`	🟪

RGB Triplet	Hexadecimal Color Code	Appearance
`[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”

RF Propagation Objects and Methods

siteviewer

Create Site Viewer map display for visualizing sites

Description

Use the `siteviewer` object to create a map viewer for visualizing transmitter and receiver sites.

Note Site Viewer is a 3-D map display and requires hardware graphics support for WebGL™.

Creation

Syntax

```
viewer = siteviewer
viewer = siteviewer(Name,Value)
```

Description

`viewer = siteviewer` creates a "Site Viewer" map display for visualizing transmitter or receiver sites.

`viewer = siteviewer(Name,Value)` creates a Site Viewer map display with properties specified by one or more name-value pairs. Properties you do not specify retain their default values.

Properties

`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`

`Basemap` — Map imagery used to visualize sites
`'satellite'` (default) | `'streets'` | `'openstreetmap'` | `'darkwater'` | `'grayland'` | `'bluegreen'` | `'colorterrain'` | `'grayterrain'` | `'landcover'`

Map imagery used to visualize sites, specified as a one of the following:

- `'satellite'` - Satellite imagery provided by ESRI
- `'streets'` - Street maps provided by ESRI.
- `'openstreetmap'` - Street maps provided by `OpenStreetMap`.
- `'darkwater'` - Two-tone map with light gray for land and dark gray for water.
- `'grayland'` - Two-tone map with gray for land and white for water.
- `'bluegreen'` - Two-tone map with green for land and blue for water.
- `'colorterrain'` - Shaded relief map derived from elevation and climate.
- `'grayterrain'` - Shaded relief map in shades of gray.
- `'landcover'` - Shaded relief map derived from satellite data.

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 is read-only once the Site Viewer is created.

For limitations, see “Limitations” on page 7-12.

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 is read-only once the Site Viewer is created.

For limitations, see “Limitations” on page 7-12.

Data Types: `char` | `string`

Object Functions

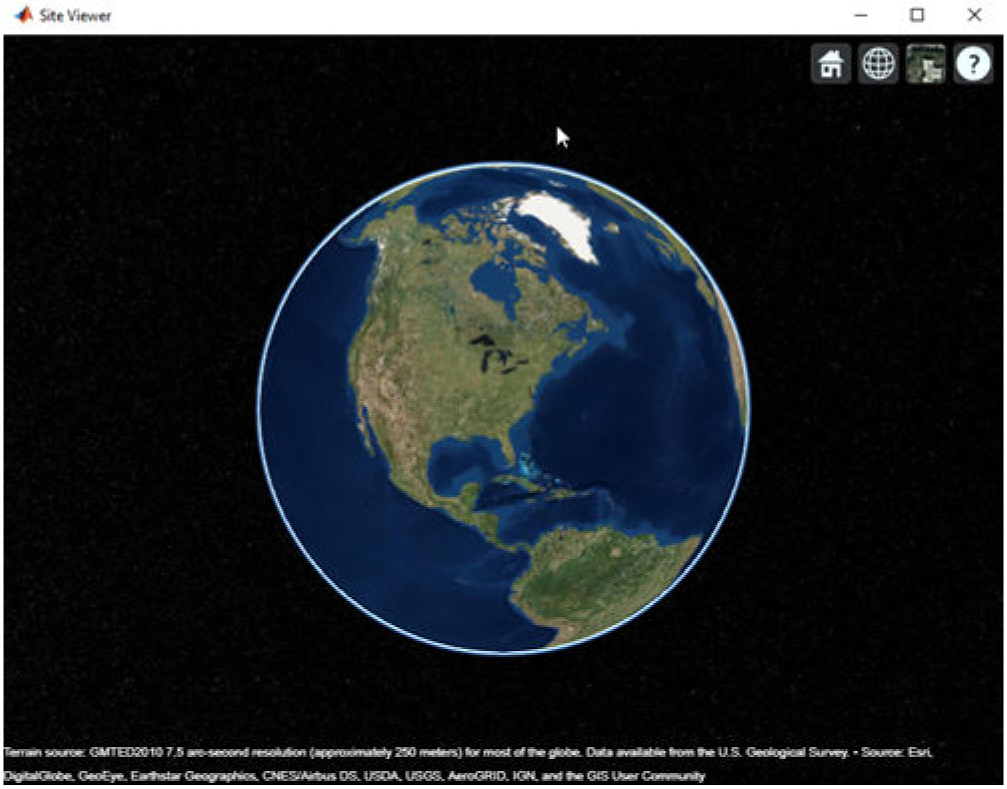
clearMap	Clear map visualizations
close	Close map viewer window

Examples

Default Site Viewer Map Display

Create a default Site Viewer map display.

```
viewer = siteviewer;
```

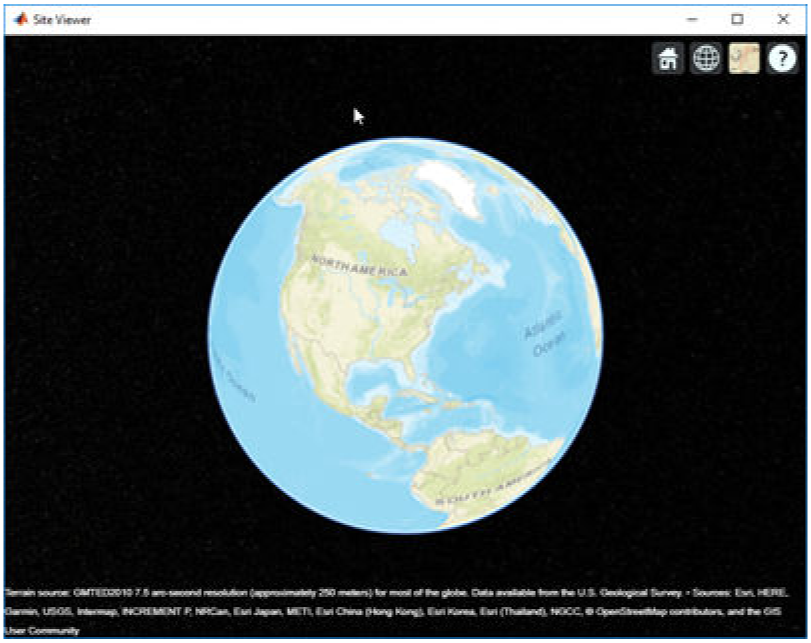


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

View Transmitter Site On Site Viewer

Launch a Site Viewer with `streets` basemap.

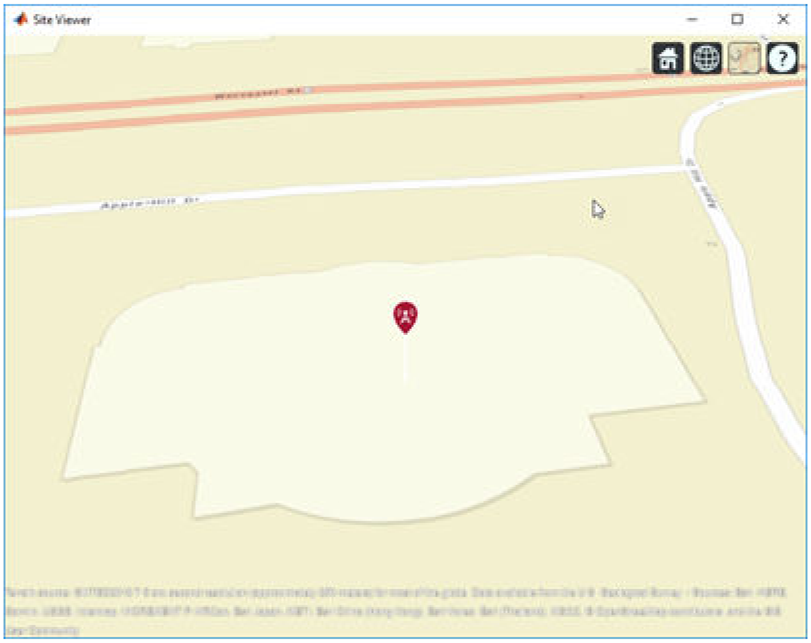
```
viewer = siteviewer("Basemap","streets");
```



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, © OpenStreetMap contributors, and the GIS User Community

View a transmitter site on this map.

```
tx = txsite;
show(tx)
```

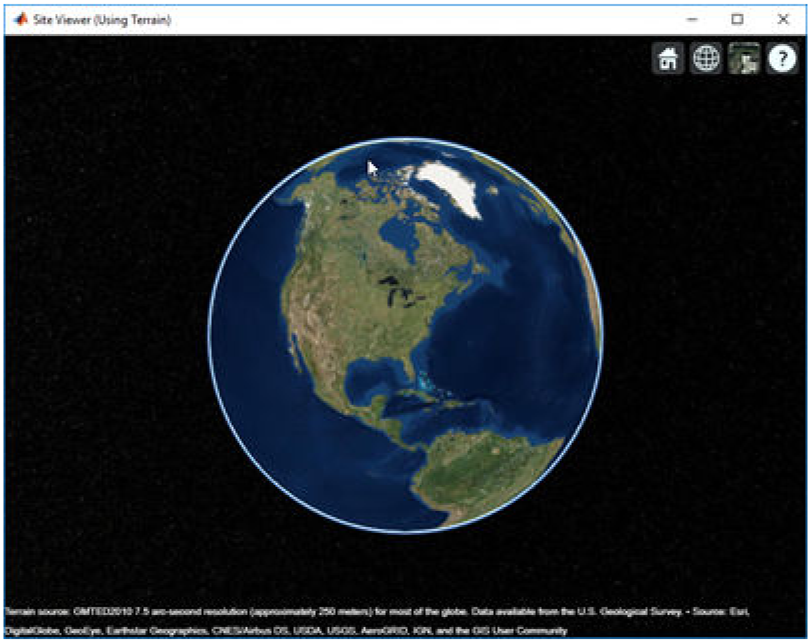


Compare Coverage Maps

Launch two Site Viewer windows.

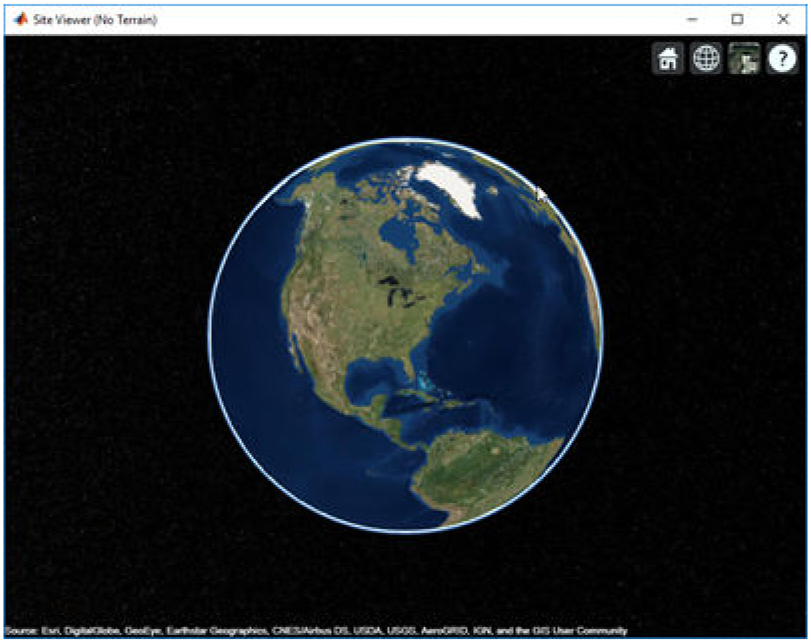
One Site Viewer window uses the terrain model.

```
viewer1 = siteviewer("Terrain","gmted2010","Name","Site Viewer (Using Terrain)");
```


Site Viewer (Using Terrain)
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

The second Site Viewer window does not use the terrain model.

```
viewer2 = siteviewer("Terrain","none","Name","Site Viewer (No Terrain)");
```

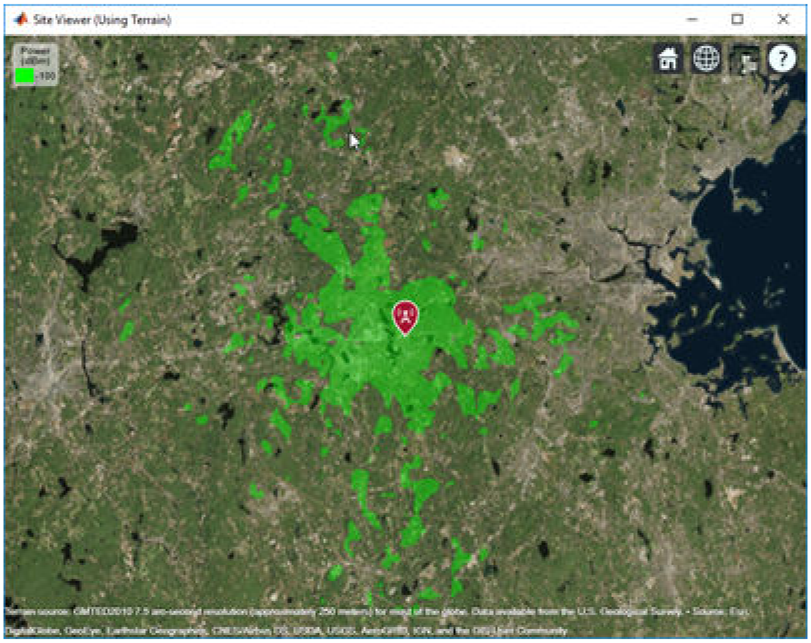

Site Viewer (No Terrain)
Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create a transmitter site.

```
tx = txsite;
```

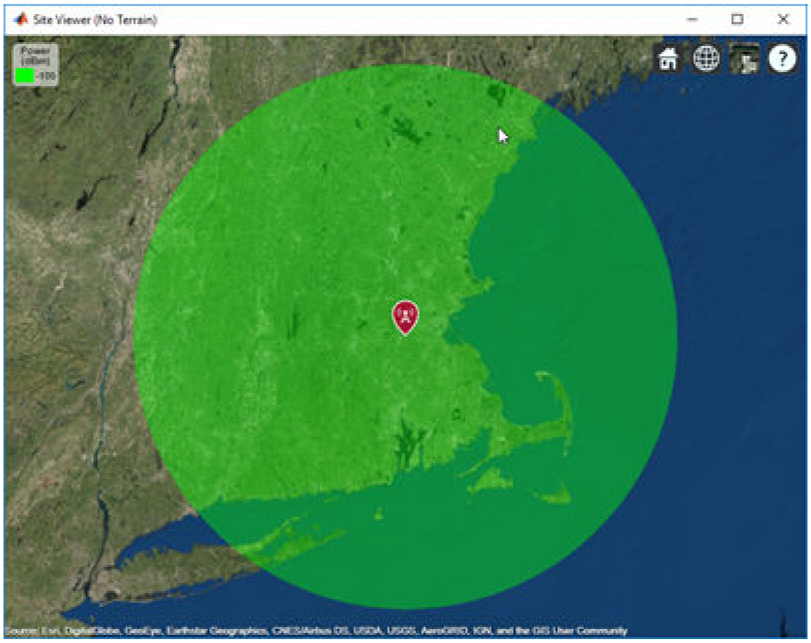
Generate a coverage map on each window. The map with terrain uses the Longley-Rice propagation model by default.

```
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, DigitalGlobe, 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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

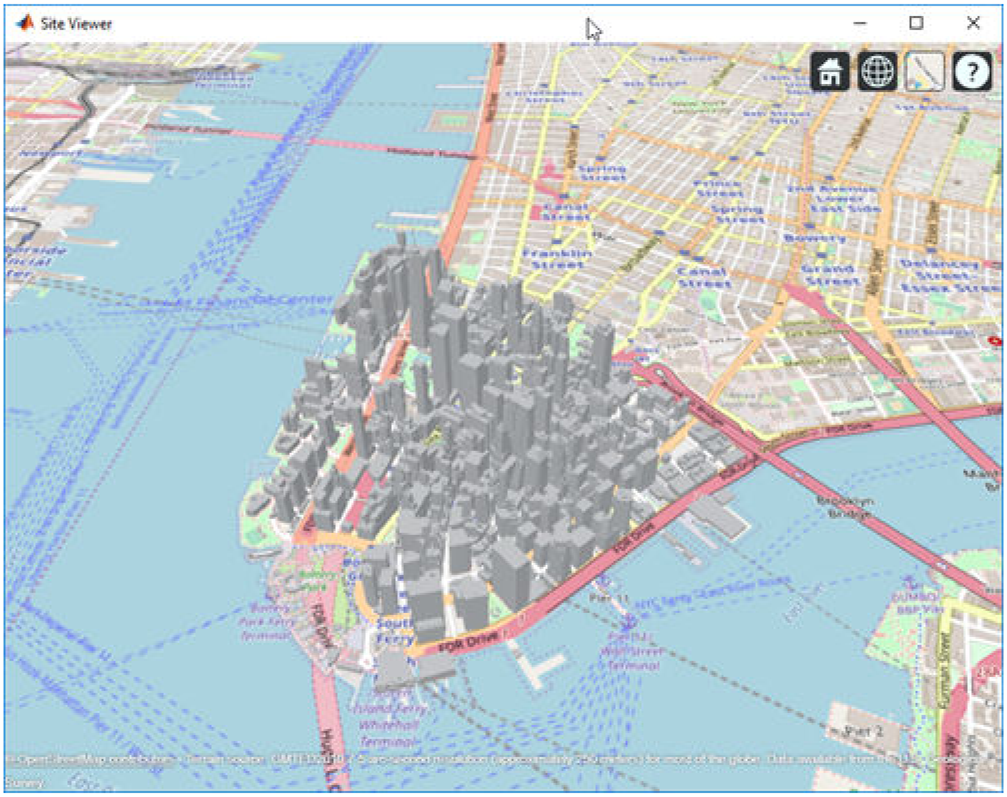
Close the maps.

```
close(viewer1)
close(viewer2)
```

Site Viewer with Buildings

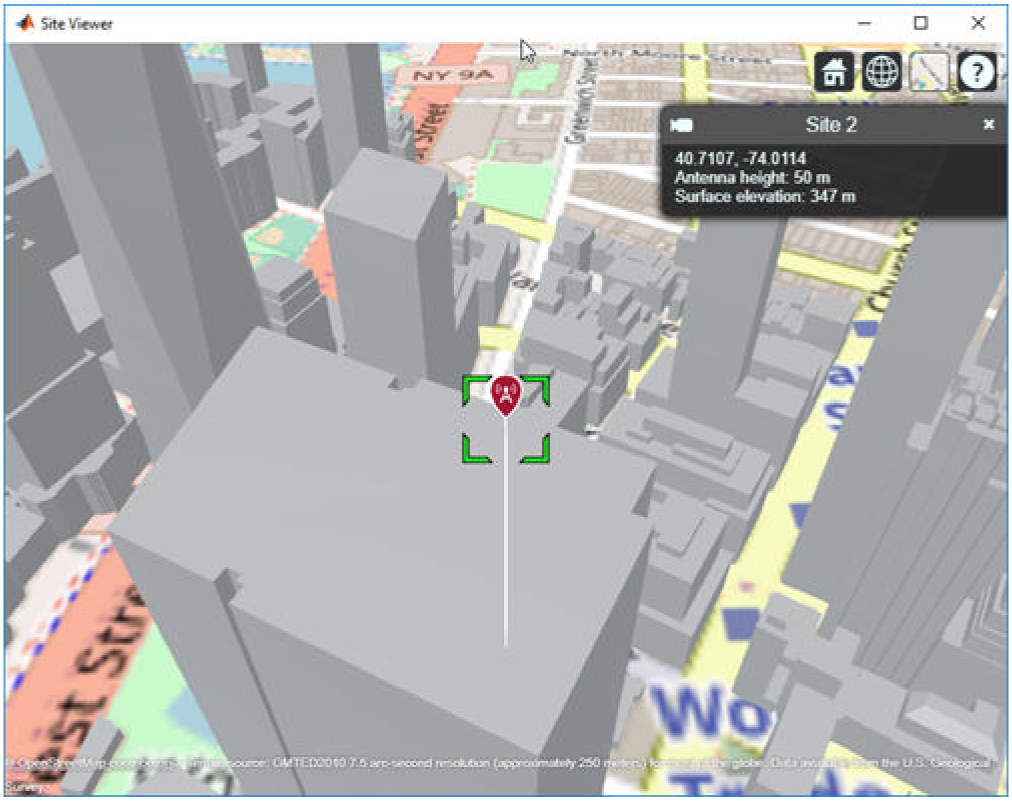
Launch siteviewer map window with basemap and buildings file.

```
viewer = siteviewer("Basemap","openstreetmap",...
       "Buildings","manhattan.osm");
```



Show a transmitter site on a building.

```
tx = txsite("Latitude",40.7107,...
       "Longitude",-74.0114,...
       "AntennaHeight",50);
show(tx)
```



Site Viewer
NY 9A
North Moore Street
Greenwich Street
Church
West Str
Wo
© OpenStreetMap contributors, [illegible] source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey.

Site 2

40.7107, -74.0114
Antenna height: 50 m
Surface elevation: 347 m

Add and Remove a Custom Basemap

Add a custom basemap to view locations on an OpenTopoMap® basemap, then remove the custom basemap from `siteviewer`.

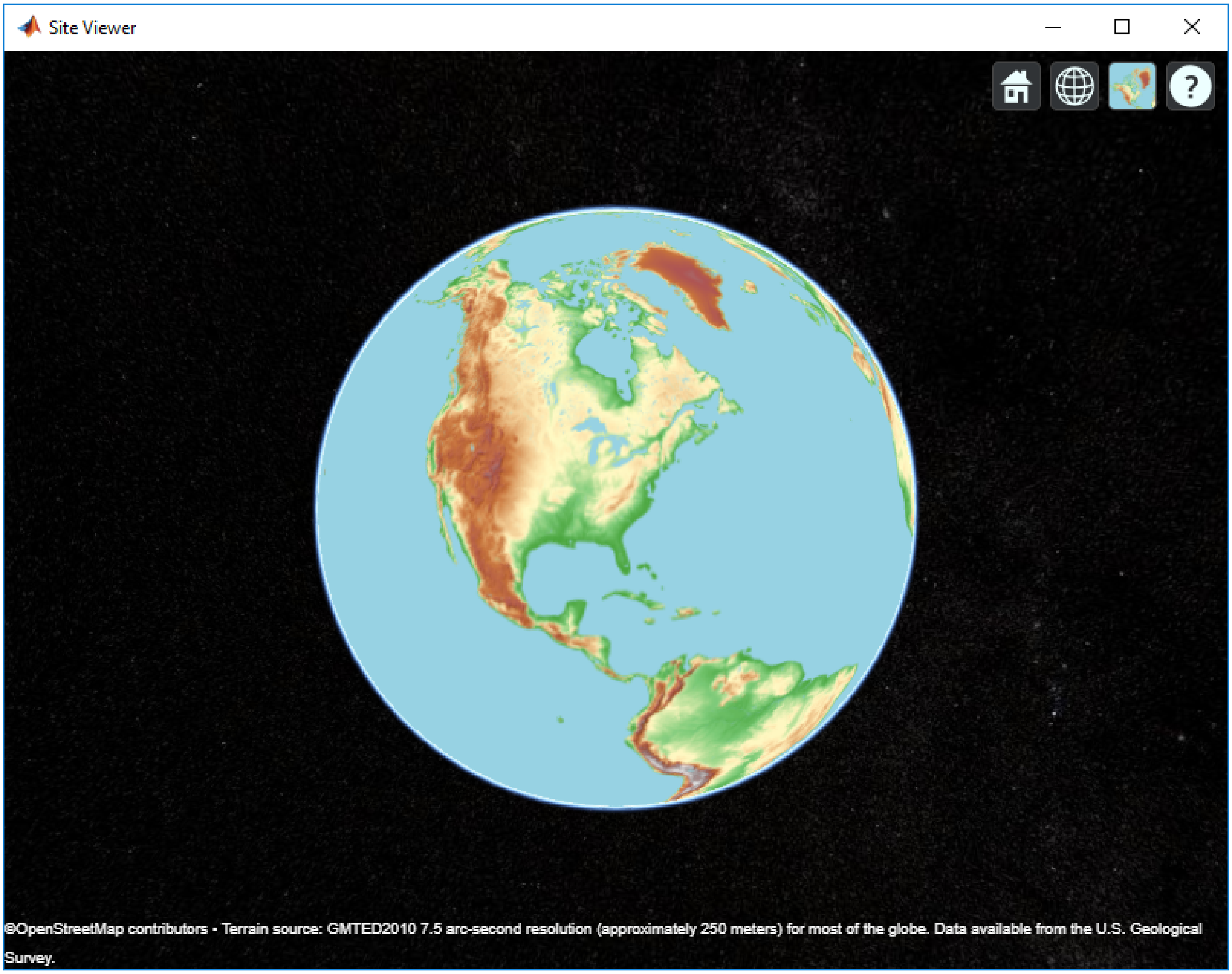
Initialize simulation variables to:

- 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.

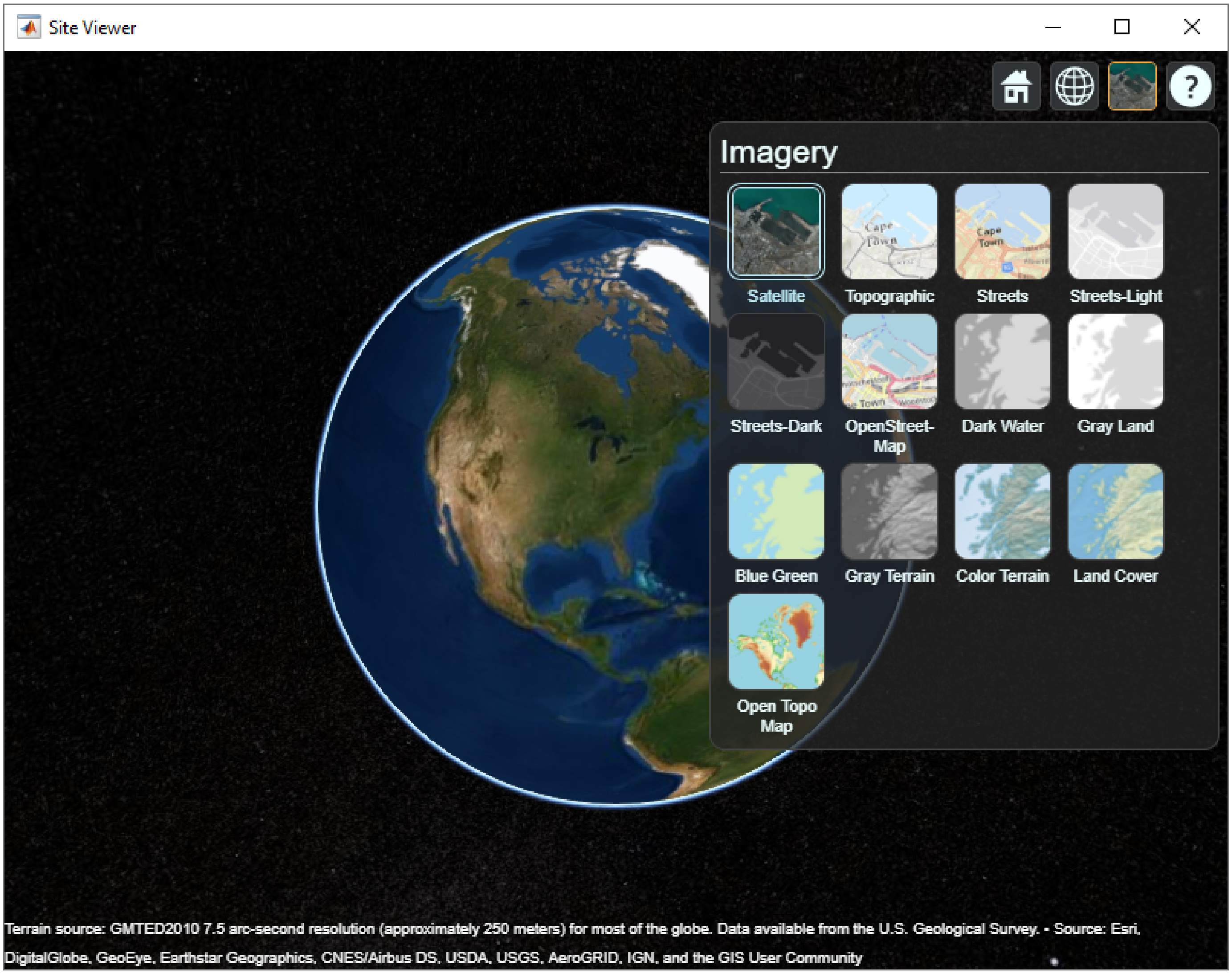
```
addCustomBasemap(name,url,'Attribution',attribution','DisplayName',displayName)
viewer = siteviewer('Basemap',name);
```



©OpenStreetMap contributors • Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey.

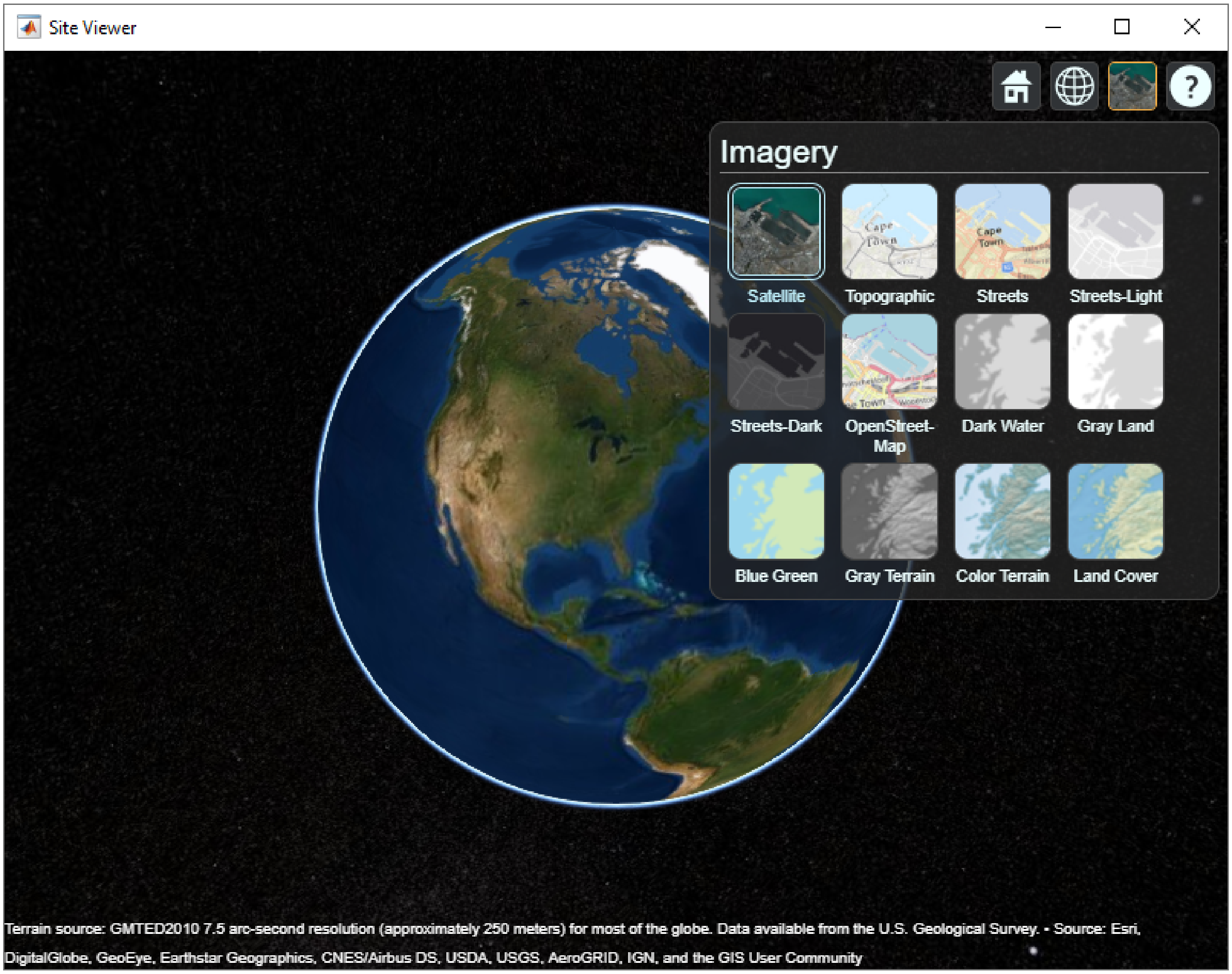
After a custom basemap is added to `siteviewer`, the custom map is available for future calls to `siteviewer`. Note the `'Open Topo Map'` icon in the `Imagery` tab.

```
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, DigitalGlobe, 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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Limitations

Terrain

- Default terrain access requires 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 trying to show a txsite or rxsite 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 3D buildings.

See Also

`addCustomBasemap` | `addCustomTerrain` | `removeCustomBasemap` | `removeCustomTerrain` | `rxsite` | `txsite`

Topics

"Site Viewer"

Introduced in R2019a

txsite

Create radio frequency transmitter site

Description

Use `txsite` object to create a radio frequency transmitter site.

Creation

Syntax

```
tx = txsite
tx = txsite(Name,Value)
```

Description

`tx = txsite` creates a radio frequency transmitter site.

`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 name `TX Site`. Enclose each property name in quotes.

Properties

Name — Site name
character vector | string | row or column vector

Site name, specified as a character vector or string, or row or column vector of *N* elements.

Example: `'Name','Site 2'`

Example: `TX.Name = 'Fenway Park'`

Example: If you want to assign multiple values then - `names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"]; TX = txsite('Name',names)`

Data Types: `char` | `string`

Latitude — Site latitude coordinates
`42.3001` (default) | numeric scalar | 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. Coordinates are defined using Earth ellipsoid model WGS-84. Latitude is the north/south angle.

Example: `'Latitude'`,`45.098`

Example: `TX.Latitude = 45.098`

Example: If you want to assign multiple values then - `latitude = [42.3467,42.3598,42.3763]; TX = txsite('Latitude',latitude)`

`Longitude` — Site longitude coordinates
`-71.3504` (default) | numeric scalar | row or column vector

Site longitude coordinates, specified as a numeric scalar or as a row or column vector of *N* elements. Coordinates are defined using Earth ellipsoid model WGS-84. Longitude is the east/west angle.

Example: `'Longitude',-68.890`

Example: `TX.Longitude = -71.0972`

Example: If you want to assign multiple values then - `longitude = [-71.0972,-71.0545,-71.0611]; TX = txsite('Longitude',longitude)`

`Antenna` — Antenna element or array
`'isotropic'` (default) | object | row vector

Antenna element or array specified as an object or `'isotropic'`. By default, the antenna is `'isotropic'`, which defines an antenna that radiates uniformly in all directions.

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, specified as a numeric scalar or a 2-by-1 vector or a 2-by-*N* matrix in degrees.

The azimuth angle measured counterclockwise from the east to the antenna x-axis.

The elevation angle measures from the horizontal plane to antenna x-axis from `-90` to `90` degrees.

Example: `'AntennaAngle',25`

Example: `TX.AntennaAngle = [25,-80]`

`AntennaHeight` — Antenna height above surface
`10` (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: `TX.AntennaHeight = 15`

Data Types:

`SystemLoss` — System loss
`0` (default) | nonnegative numeric scalar | row vector

System loss, specified as a non-negative numeric 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) | numeric scalar | row vector

Transmitter operating frequency, specified as a numeric scalar in Hz. The range is from `1e3` to `200e9`.

Example: `'TransmitterFrequency',30e9`

Example: `txsite.TransmitterFrequency = 30e9`

Data Types: `double`

`TransmitterPower` — Signal power at transmitter output

`10` (default) | positive numeric scalar

Signal power at transmitter output, specified as a positive numeric 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 location on map
hide	Hide site location on map
distance	Distance between sites
angle	Angle between sites
elevation	Elevation of site
location	Location coordinates at a given distance and angle from site
los	Plot or compute the line-of-sight (LOS) visibility between sites on a map
coverage	Display coverage map
sinr	Display signal-to-interference-plus-noise ratio (SINR) map
pattern	Plot antenna radiation pattern on map

Examples

Default Transmitter Site

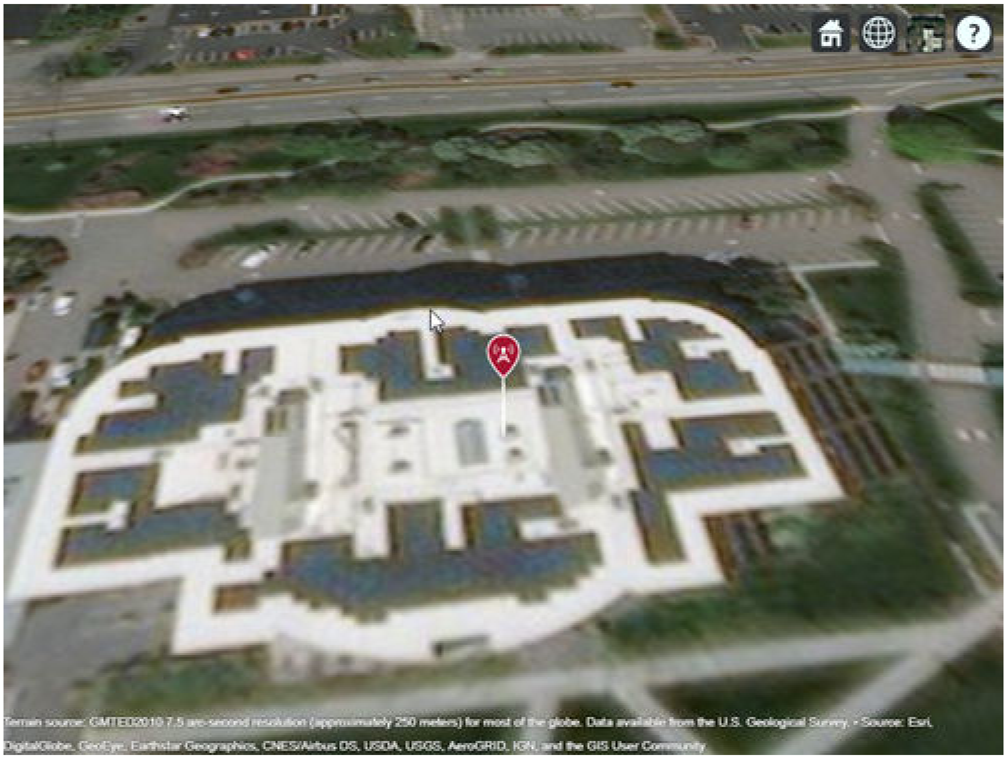
Create and view a transmitter site at a latitude of 42.3001 and a longitude of -71.3504.

```
tx = txsite('Name','MathWorks Apple Hill','Latitude',42.3001,...
    'Longitude',-71.3504)
```

```
tx = 
  txsite with properties:

                  Name: 'MathWorks Apple Hill'
              Latitude: 42.3001
             Longitude: -71.3504
               Antenna: 'isotropic'
```

```
           AntennaAngle: 0
          AntennaHeight: 10
             SystemLoss: 0
    TransmitterFrequency: 1.9000e+09
        TransmitterPower: 10
```

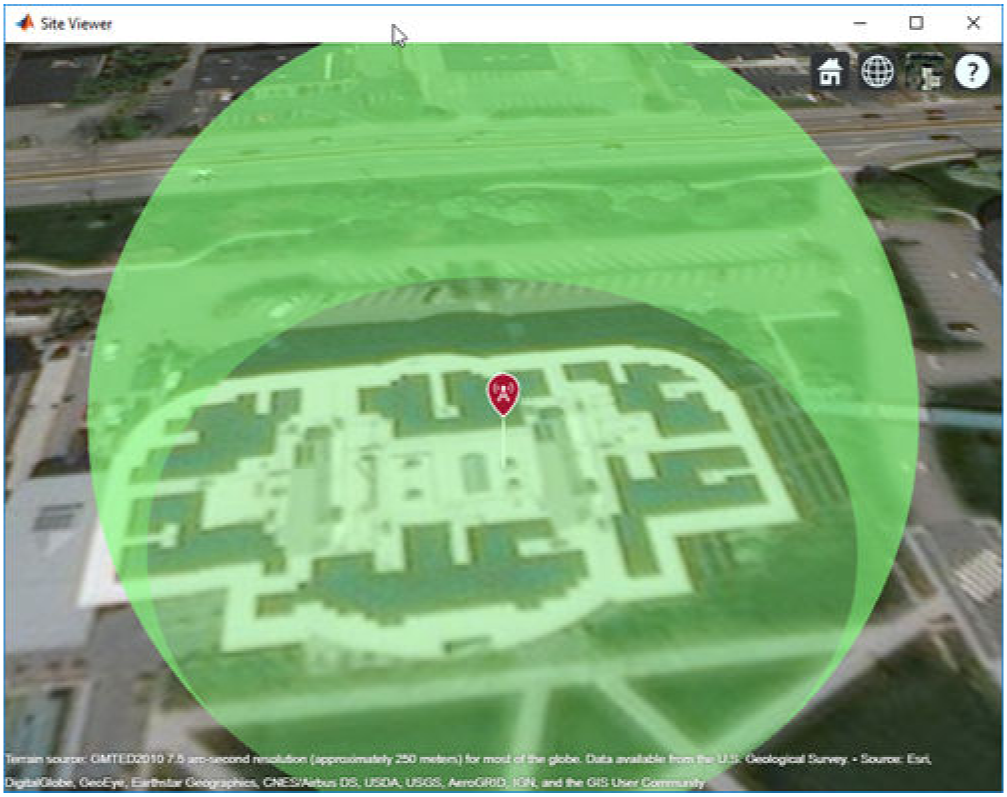
```
show(tx)
```



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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

View the coverage of the antenna.

```
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, DigitalGlobe, 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 and a longitude of -71.3504.

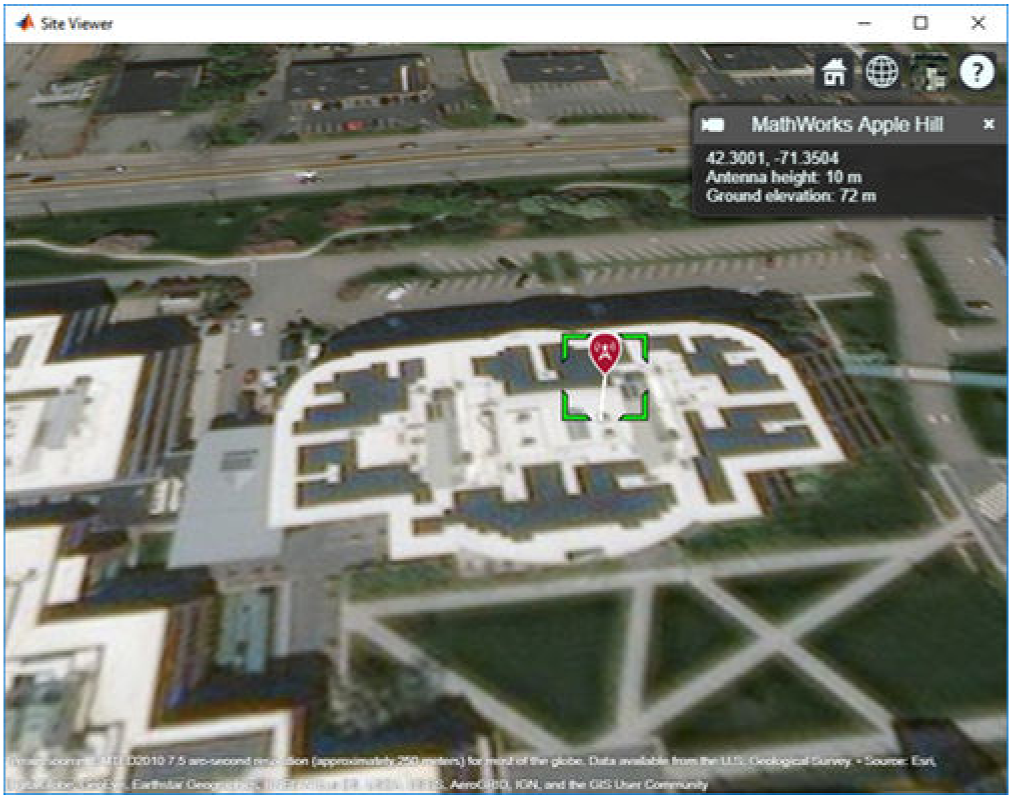
```
fq = 2.5e9
```

```
fq = 2.5000e+09
```

```
tx = txsite('Name','MathWorks Apple Hill','Antenna',dipole,'Latitude',42.3001,...
    'Longitude',-71.3504,'Antenna',design(dipole,fq),'TransmitterFrequency',fq)
```

```
tx = 
  txsite with properties:

                    Name: 'MathWorks Apple Hill'
                Latitude: 42.3001
               Longitude: -71.3504
                 Antenna: [1×1 dipole]
            AntennaAngle: 0
           AntennaHeight: 10
              SystemLoss: 0
    TransmitterFrequency: 2.5000e+09
        TransmitterPower: 10
```

```
show(tx);
```


Site Viewer
[illegible] 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, [illegible] AeroGRID, IGN, and the GIS User Community

MathWorks Apple Hill

42.3001, -71.3504
Antenna height: 10 m
Ground elevation: 72 m

Transmitter Array Using Dipole Antenna

Specify the names, latitudes, and longitudes of three transmitter locations.

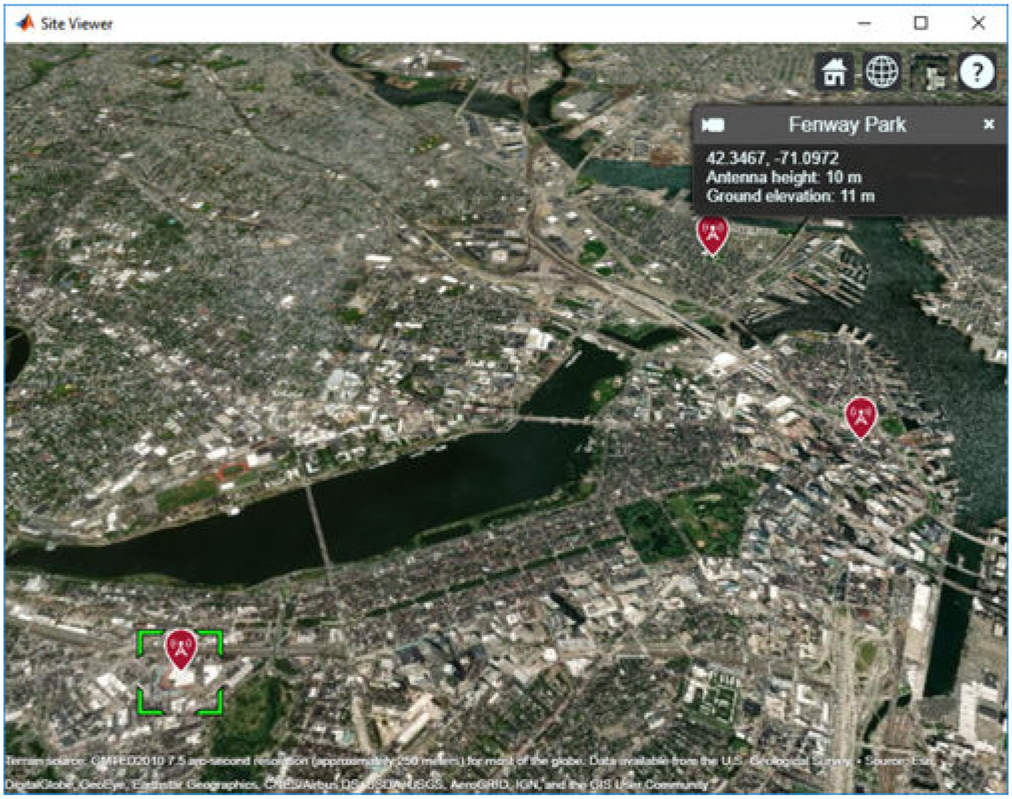
```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

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)
```


Site Viewer
Fenway 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 • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

42.3467, -71.0972
Antenna height: 10 m
Ground elevation: 11 m

See Also

`rxsite` | `siteviewer`

Introduced in R2017b

rxsite

Create radio frequency receiver site

Description

Use the `rxsite` object to create a radio frequency receiver site.

Creation

Syntax

```
rx = rxsite
rx = rxsite(Name,Value)
```

Description

`rx = rxsite` creates a radio frequency receiver site.

`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.

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`

`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 = 45.098`

Example: If you want to assign multiple values then - `latitude = [42.3467,42.3598,42.3763]; RX = rxsite('Latitude',latitude)`

`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)`

`Antenna` — Antenna element or array

`'isotropic'` (default) | object | row vector

Antenna element or array specified as an object or `'isotropic'`. By default, the antenna is `'isotropic'`, which defines an antenna that radiates uniformly in all directions.

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, specified as a numeric scalar, a 2-by-1 vector, or a 2-by-*N* matrix in degrees.

The numeric scalar is the azimuth angle measured counterclockwise from the east to the antenna x-axis.

In the 2-by-1 vector, the first element is the azimuth angle and the second element elevation angle. The elevation angle measures from the horizontal plane to antenna x-axis from `-90` to `90` degrees.

Example: `'AntennaAngle',25`

Example: `RX.AntennaAngle = [25,-80]`

Data Types: `double`

`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`

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 location on map
hide	Hide site location on map
distance	Distance between sites
angle	Angle between sites
elevation	Elevation of site
location	Location coordinates at a given distance and angle from site
sigstrength	Signal strength due to transmitter
los	Plot or compute the line-of-sight (LOS) visibility between sites on a map
link	Display communication link on map
pattern	Plot antenna radiation pattern on map

Examples

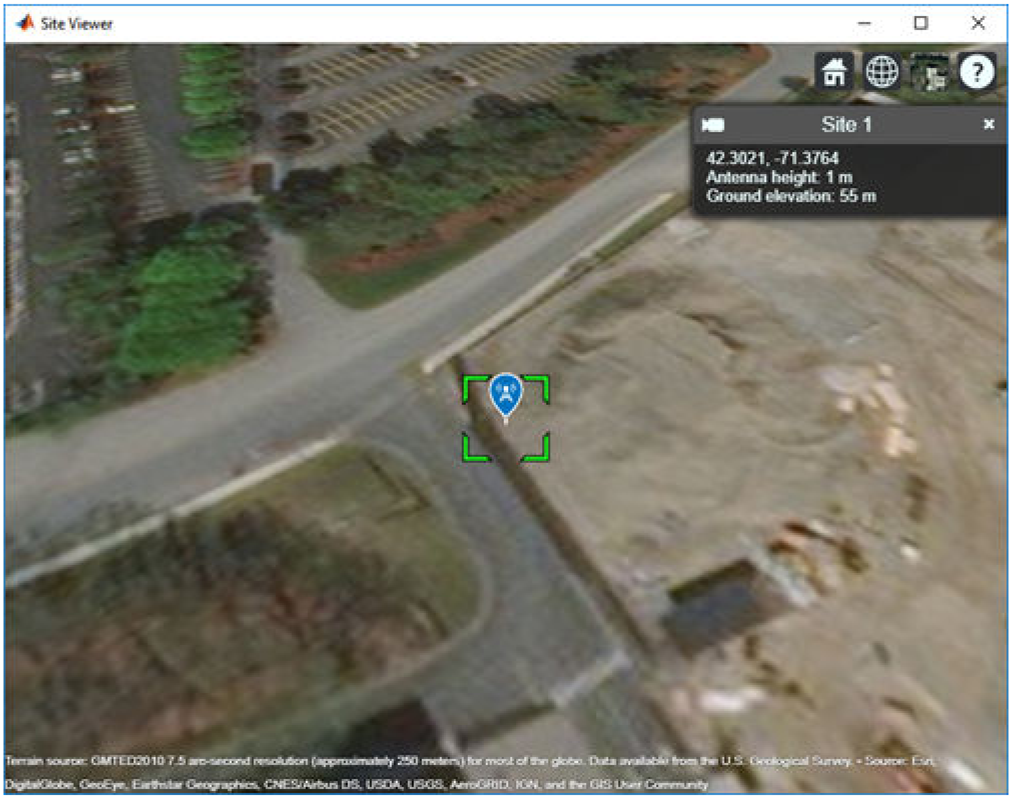
Default Receiver Site

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

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.

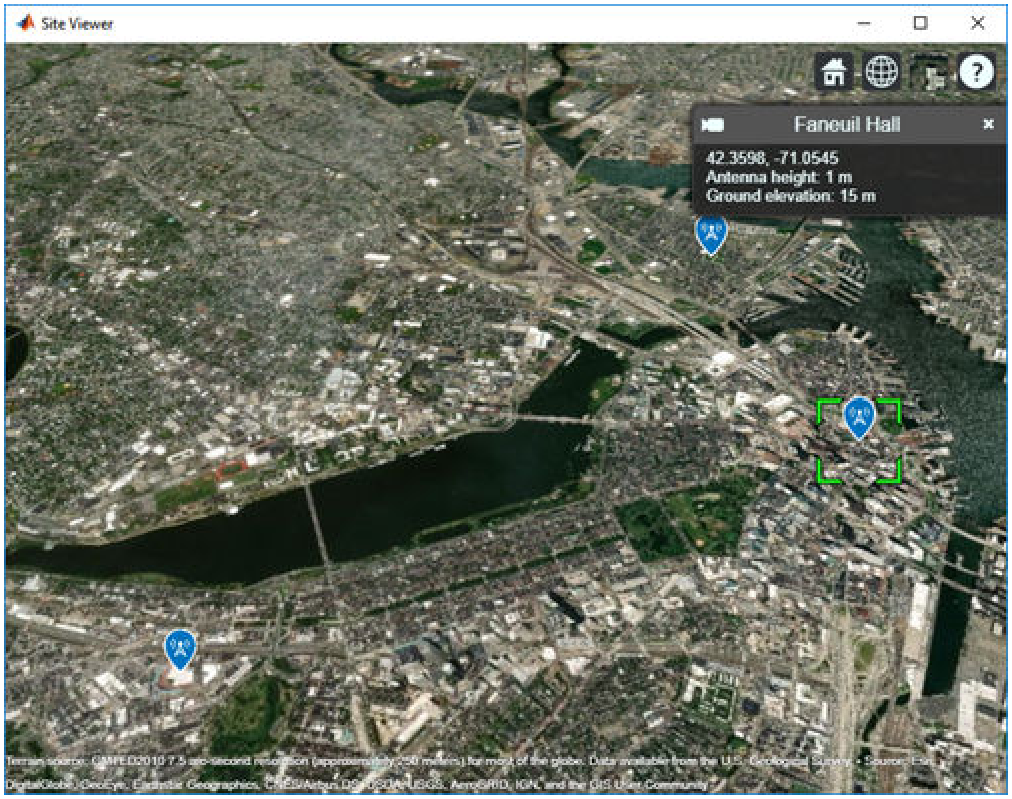
```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

Define the sensitivity of the receivers.

```
 sens = -90;
```

Create and show receiver site array.

```
rxs = rxsite('Name', names,...
      'Antenna',dipole, 'Latitude',lats,...
       'Longitude',lons, ...
       'ReceiverSensitivity',sens);
show(rxs)
```


Site Viewer
Faneuil Hall
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

42.3598, -71.0545
Antenna height: 1 m
Ground elevation: 15 m

See Also

`siteviewer` | `txsite`

Introduced in R2017b

propagationData

Create RF propagation data container

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 container 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

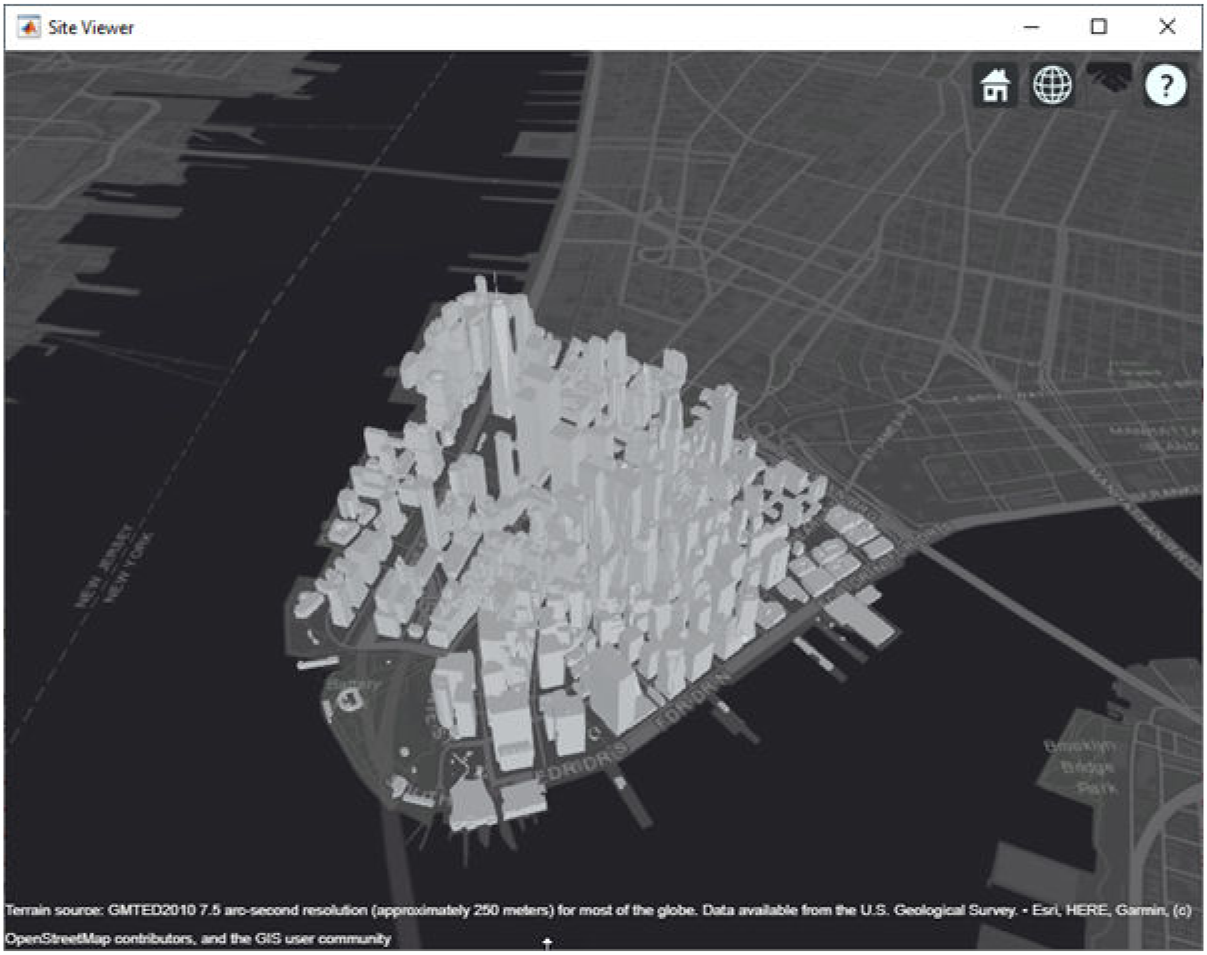
plot	Plot propagation data on map
contour	Display contour map
location	Data location coordinates
getDataVariable	Get data variable values of data points in propagation data object
interp	Geographic data interpolation

Examples

Compute Signal Strength Data in Urban Environment

Launch Site Viewer with basemaps and building files.

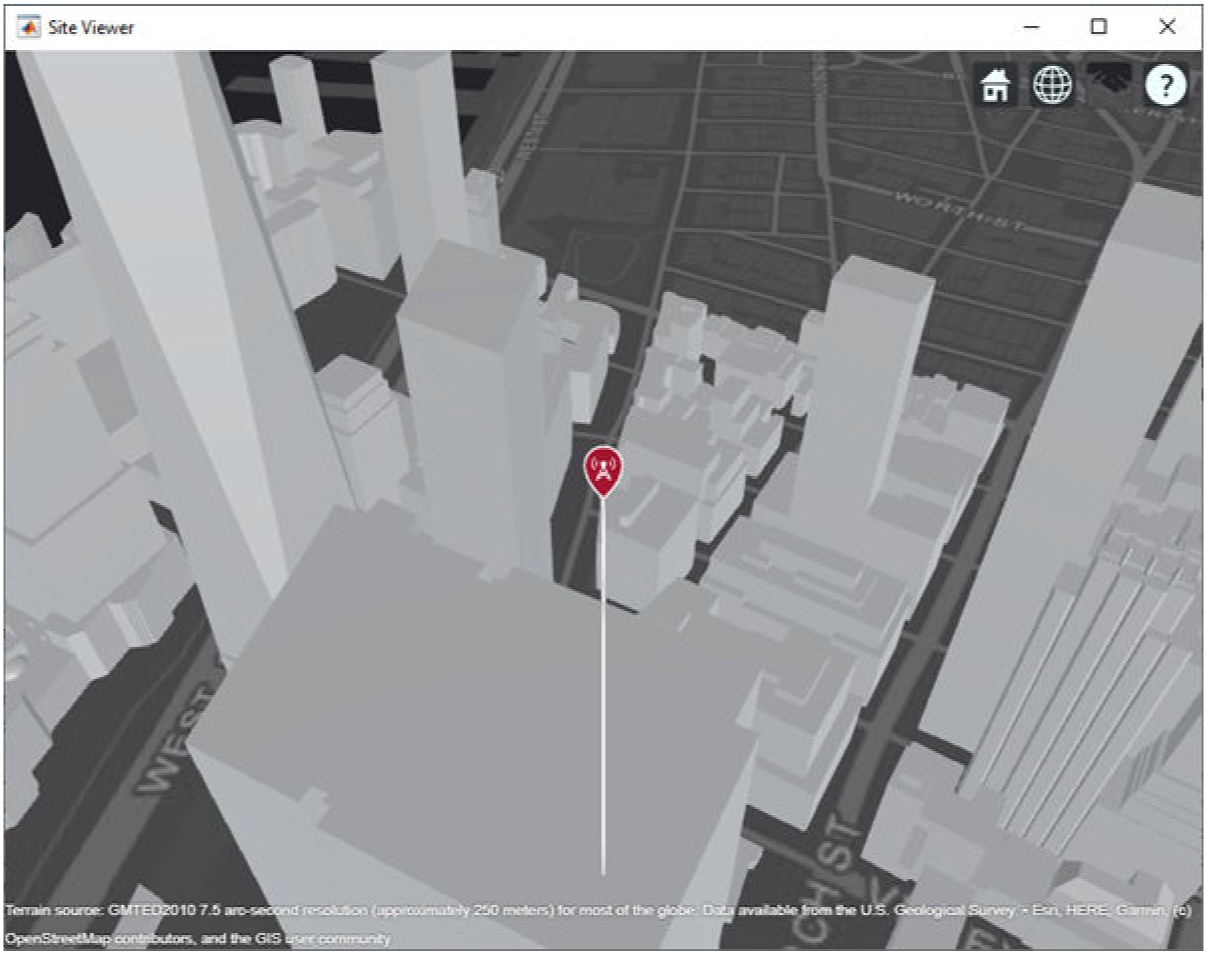
```
viewer = siteviewer("Basemap","streets_dark",...
        "Buildings","manhattan.osm");
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • 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)
```


Site Viewer
WORTH ST
WEST
CH ST
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.

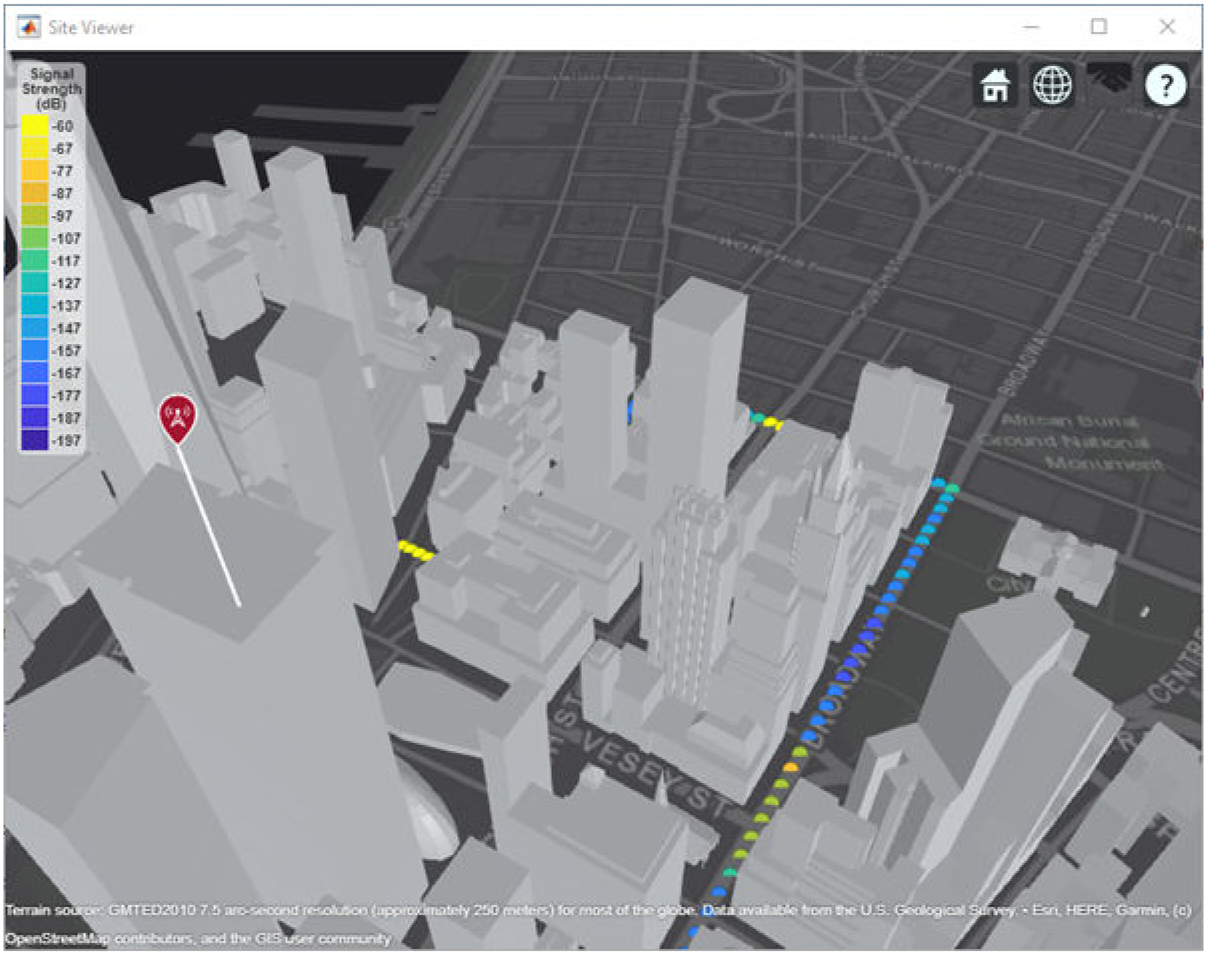
```
signalStrength = sigstrength(rxs, tx)';
```

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);
```


Site Viewer
Signal Strength (dB)
-60
-67
-77
-87
-97
-107
-117
-127
-137
-147
-157
-167
-177
-187
-197
VESEY ST
BROADWAY
African Burial Ground National Monument
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

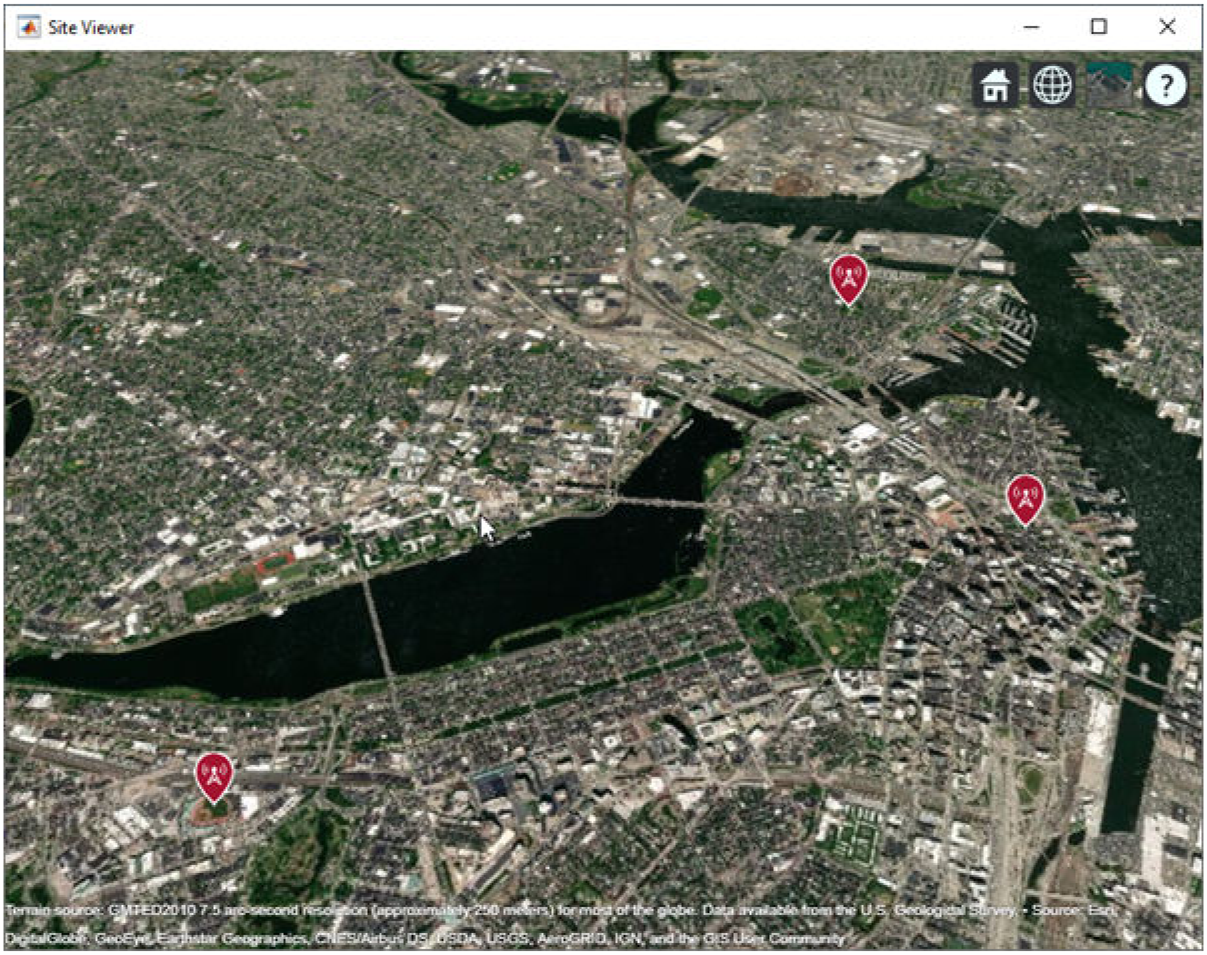
Capacity Map Using SINR Data

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];
```

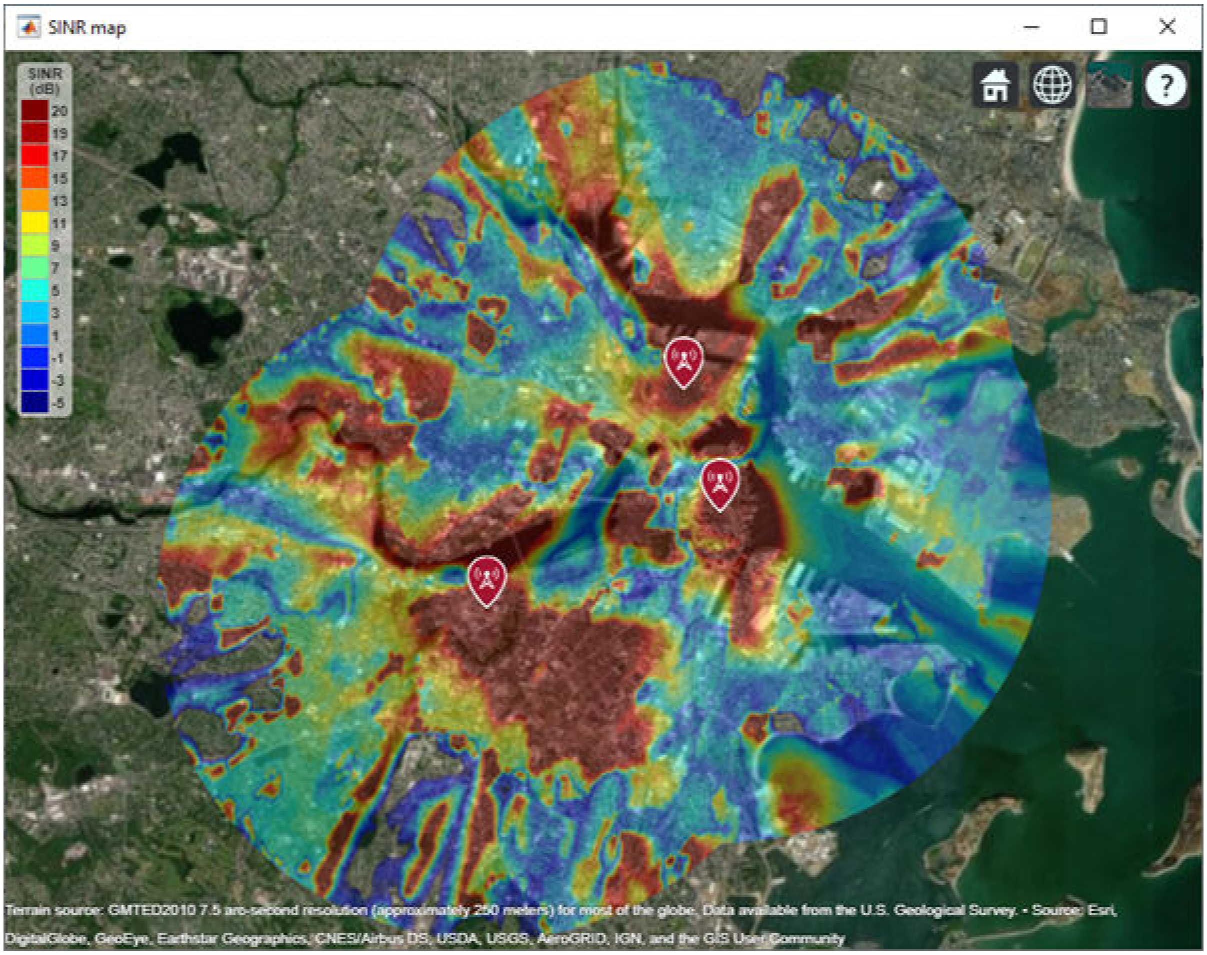
Create an array of transmitter sites.

```
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, DigitalGlobe, 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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Return SINR propagation data.

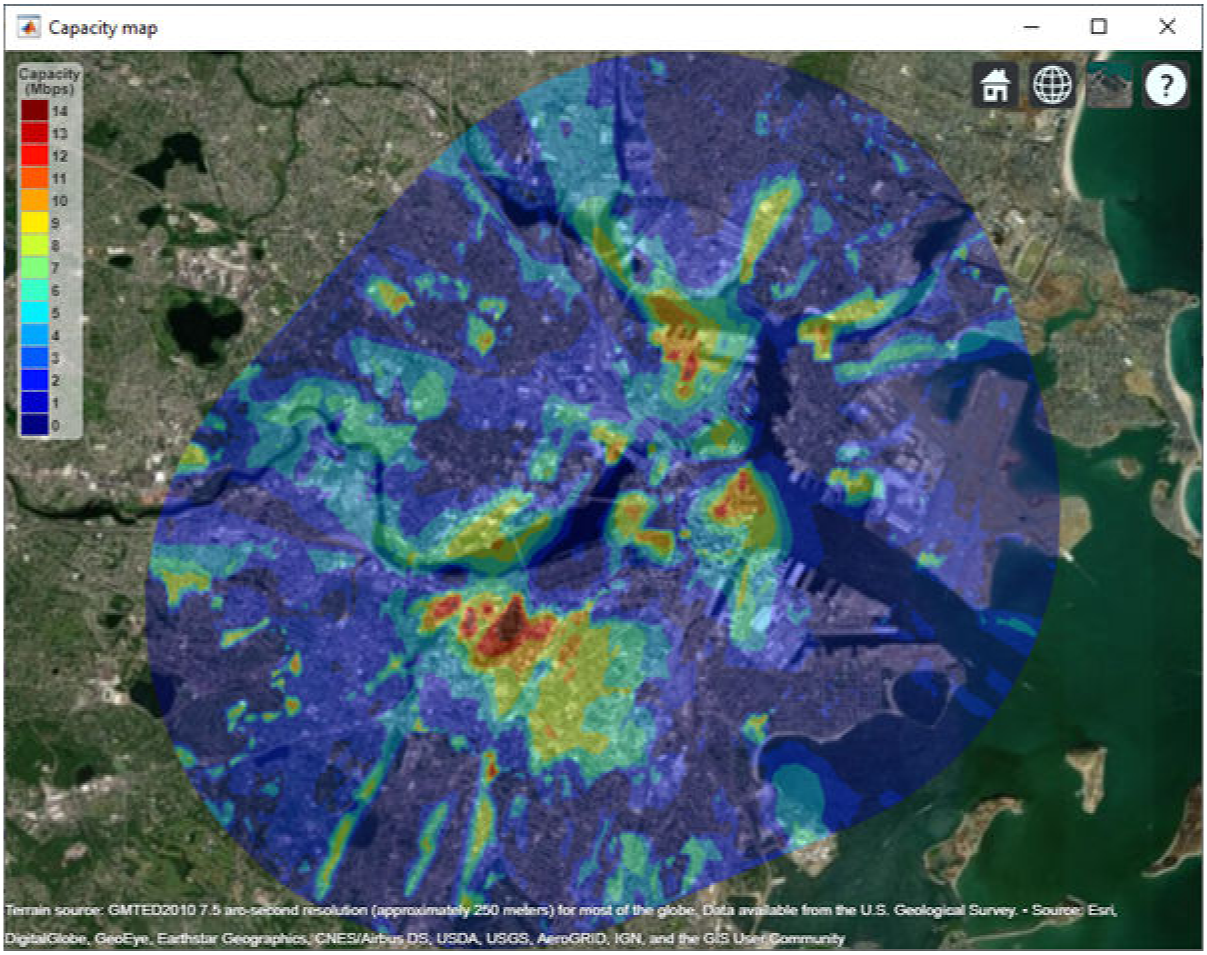
```
pd = sinr(txs,"MaxRange",5000);
[sinrDb,lats,lons] = getDataVariable(pd,"SINR");
```

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 capacity 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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

See Also

`readtable` | `rxsite` | `siteviewer` | `txsite`

Introduced in R2020a

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

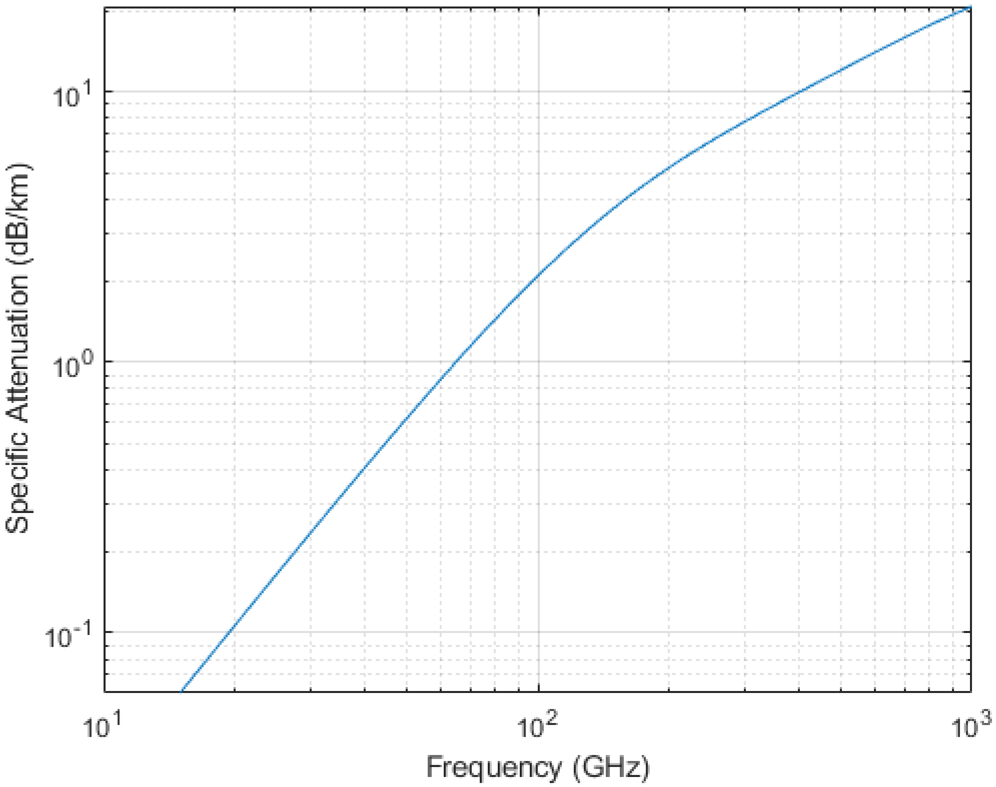
Attenuation in Cumulus Clouds

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.

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.

Input Arguments

`R`

real-valued 1-by-*M* or *M*-by-1 vector

Propagation distance of signal. Units are in meters.

`lambda`

real-valued 1-by-*N* or *N*-by-1 vector

The wavelength is the speed of propagation divided by the signal frequency. Wavelength units are meters.

Output Arguments

`L`

Path loss in decibels. *M*-by-*N* nonnegative matrix. A value of zero signifies no path loss. When `lambda` is a scalar, `L` has the same dimensions as `R`.

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
```

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 $R \leq \lambda/4\pi$.

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`, when signals propagate through the atmosphere. `range` represents the signal path length, and `freq` represents the signal carrier frequency. `T` represents the ambient temperature, `P` represents the atmospheric pressure, and `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 uniform environment. Atmospheric parameters do not vary along the signal path. 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.

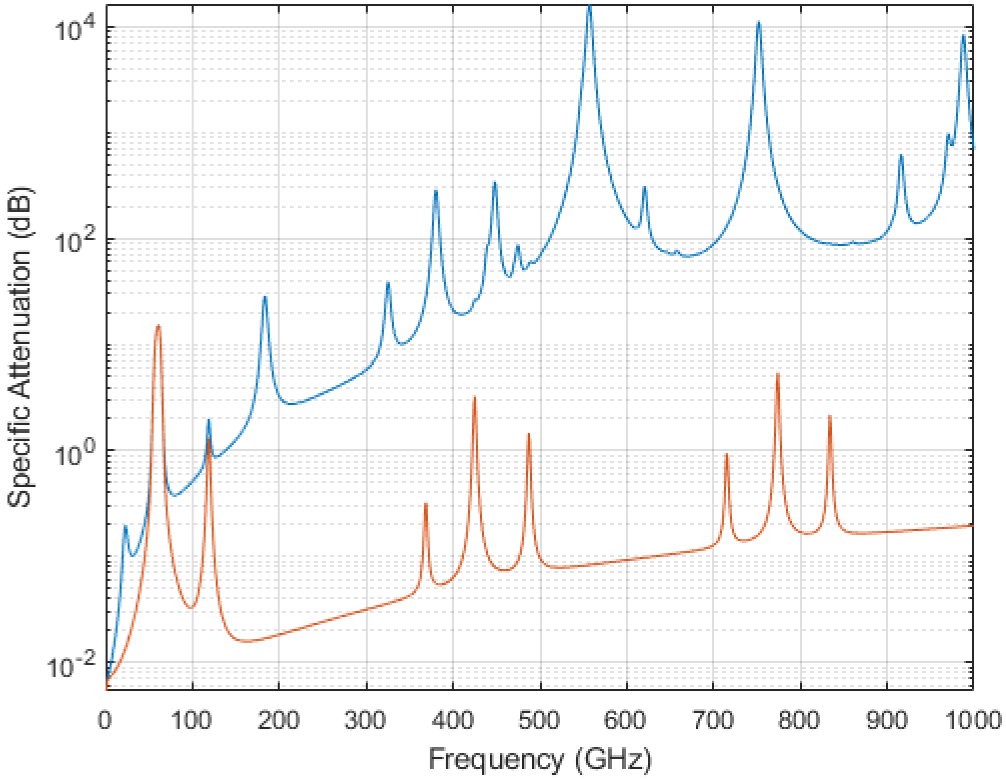
```
T = 15;
P = 101300.0;
W = 7.5;
L = gaspl(R,freq,T,P,W);
```

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.

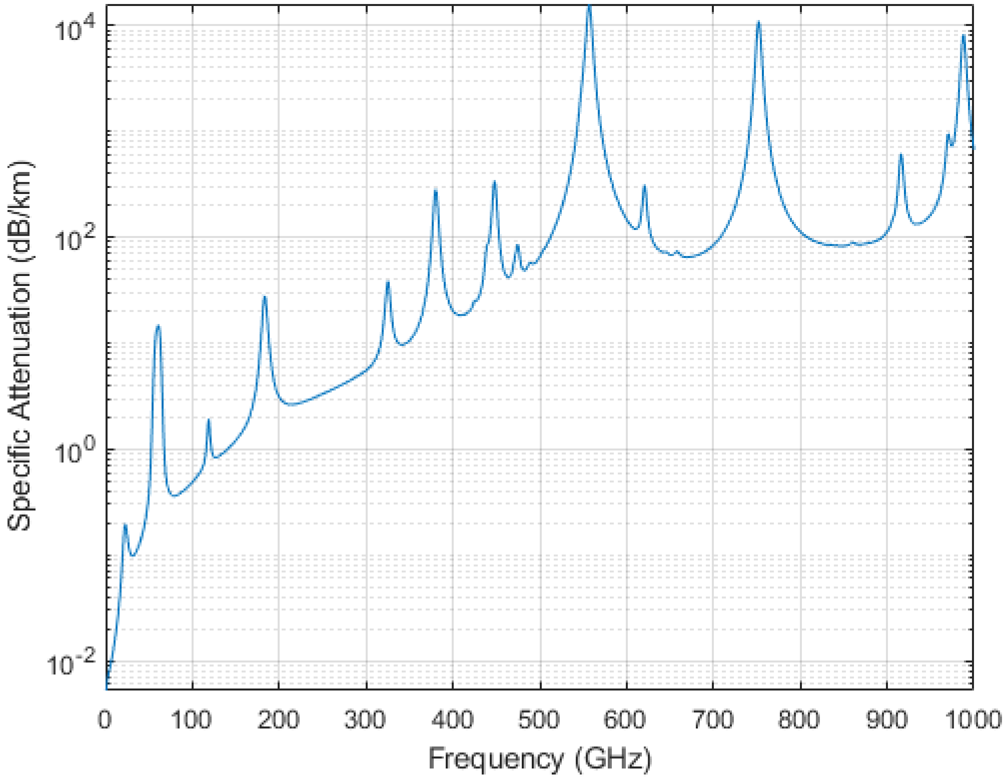
```
T = 20.0;
Patm = 101.325e3;
rho_wv = 7.5;
```

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.

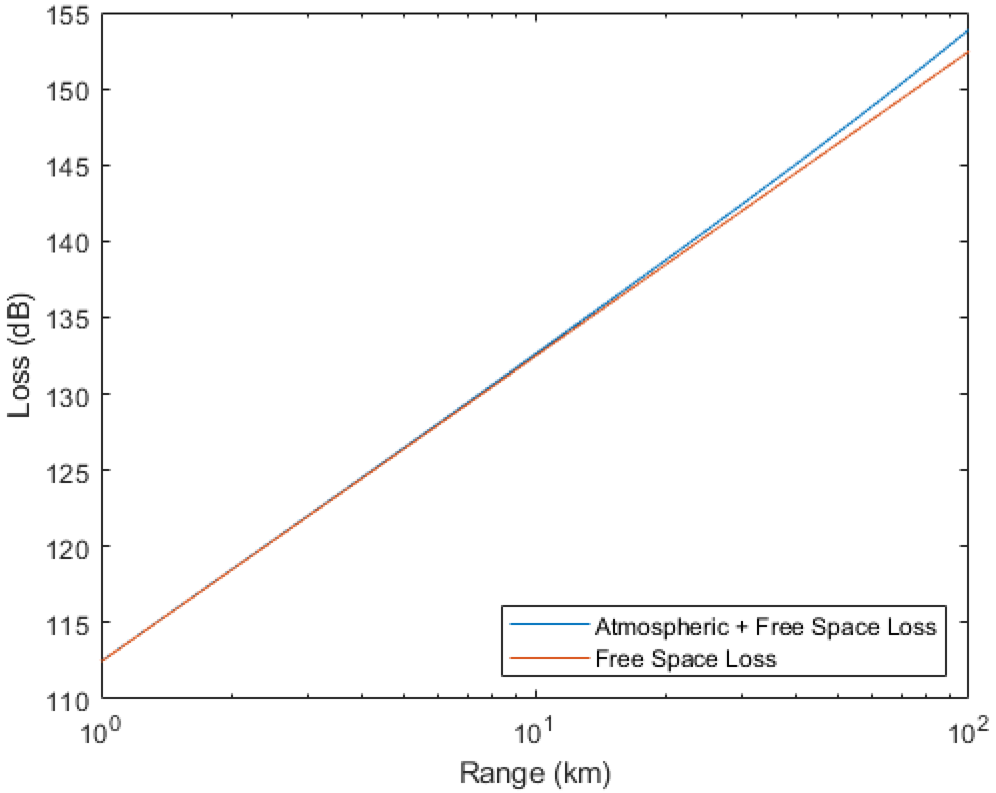
```
loss = gaspl(km,freqs,T,Patm,rho_wv);
semilogy(freqs/1e9,loss)
grid on
xlabel('Frequency (GHz)')
ylabel('Specific Attenuation (dB/km)')
```



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
```

```
L = 0.7104
```

```
L = 0.7104
```

Propagate the signal in a heavy rainfall of 10 mm/hr.

```
rr = 10.0;
L = rainpl(10000,20.0e9,rr)
```

```
L = 8.1584
```

```
L = 7.8413
```

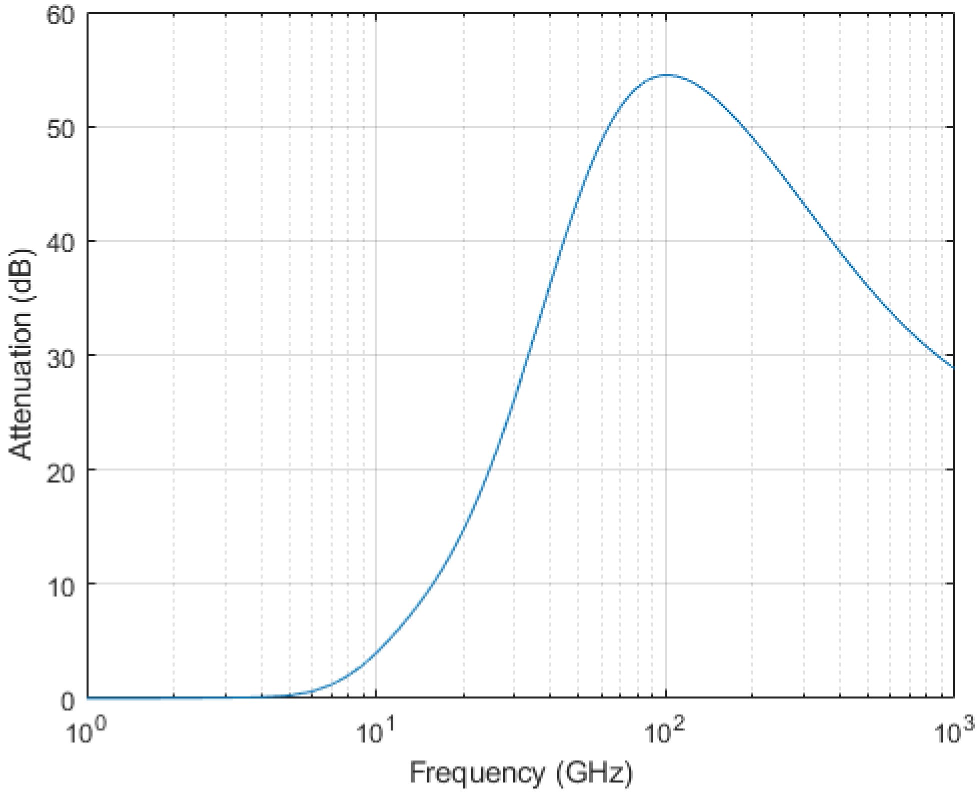
```
L = 7.8413
```

Signal Attenuation Due to Rainfall as Function of Frequency

Plot the signal attenuation due to moderate rainfall for signals in the frequency range from 1 to 1000 GHz. The path distance is 10 km.

Set the rain rate value for moderate rainfall to 20 mm/hr.

```
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.

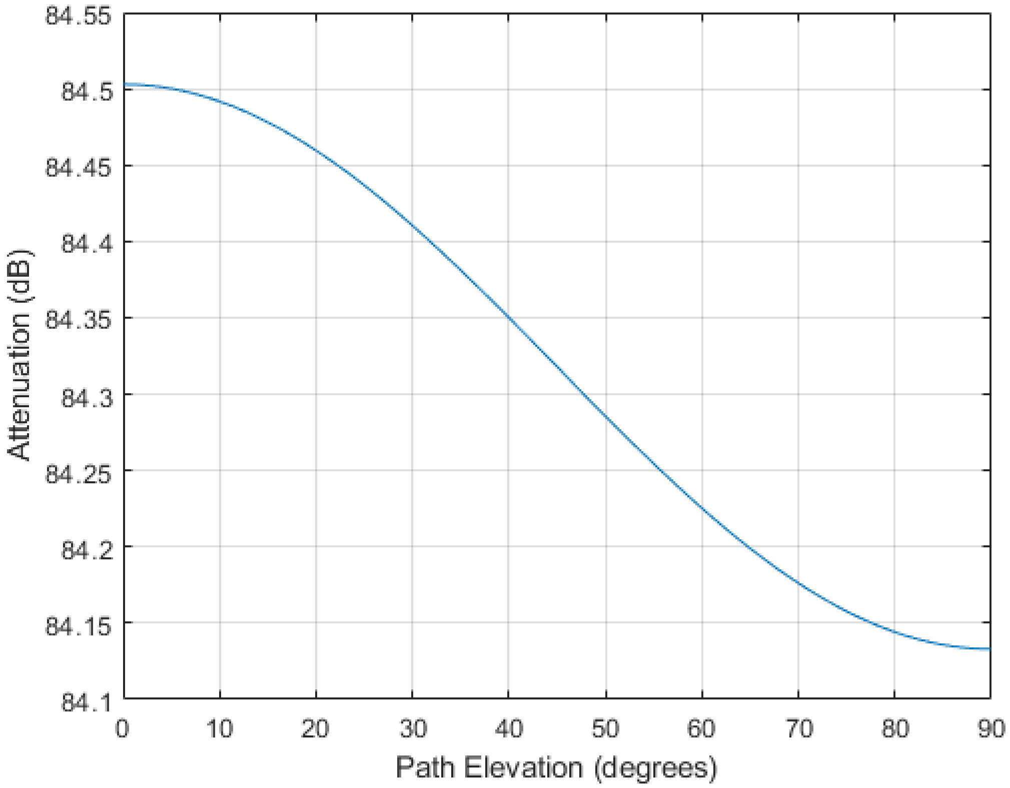
```
rr = 10.0;
```

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 signal, 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.

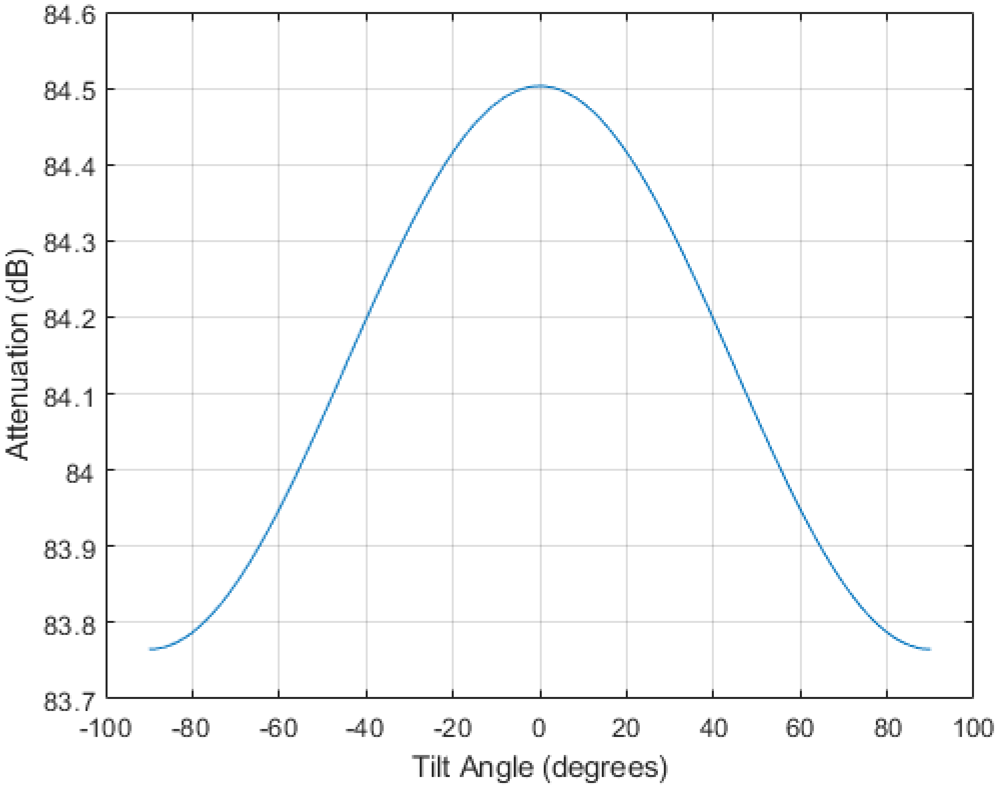
```
tau = -90:90;
```

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, χ_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 $L = d_{\text{eff}}\gamma_R$.

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

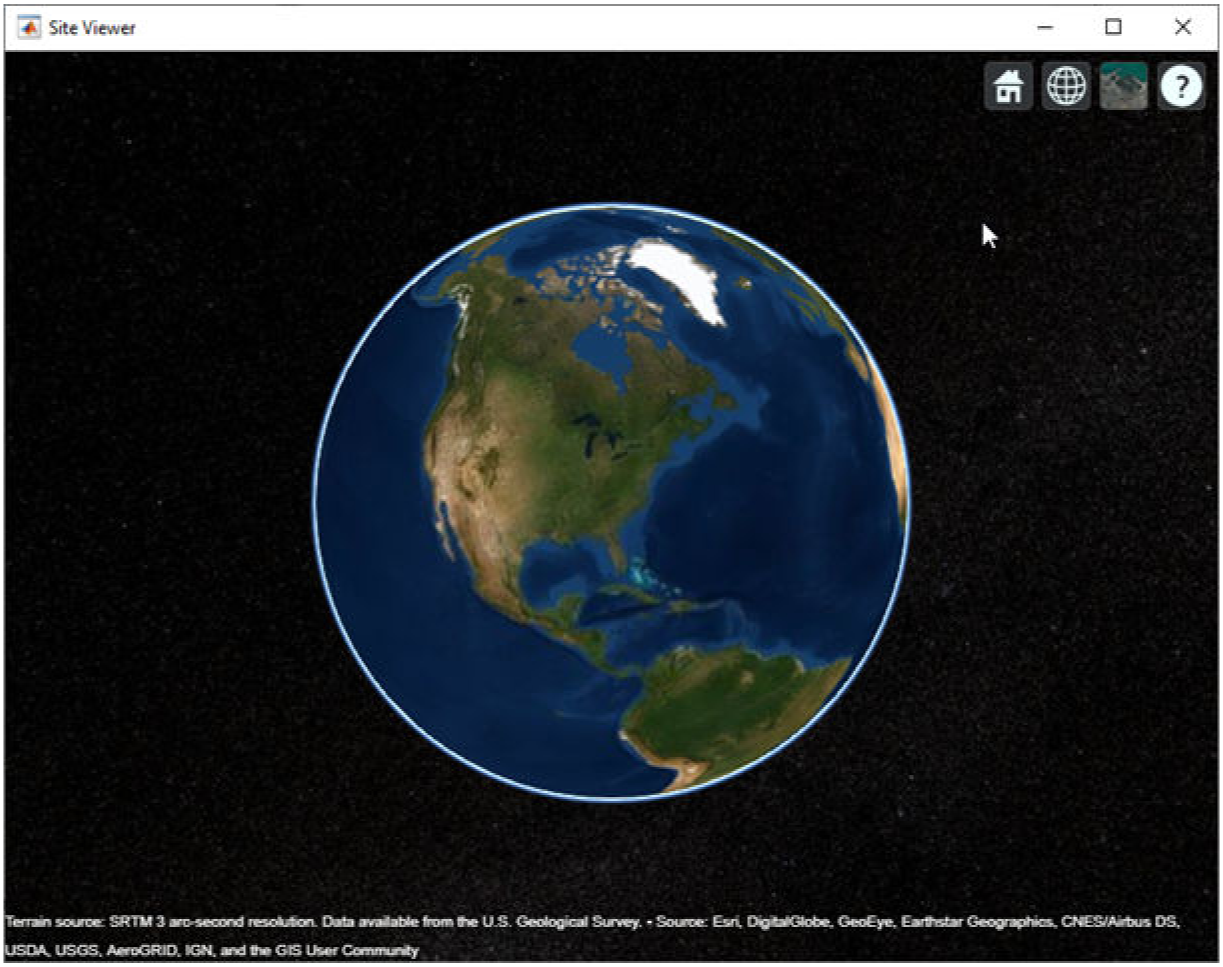
Site Viewer Maps Using Custom Terrain

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.

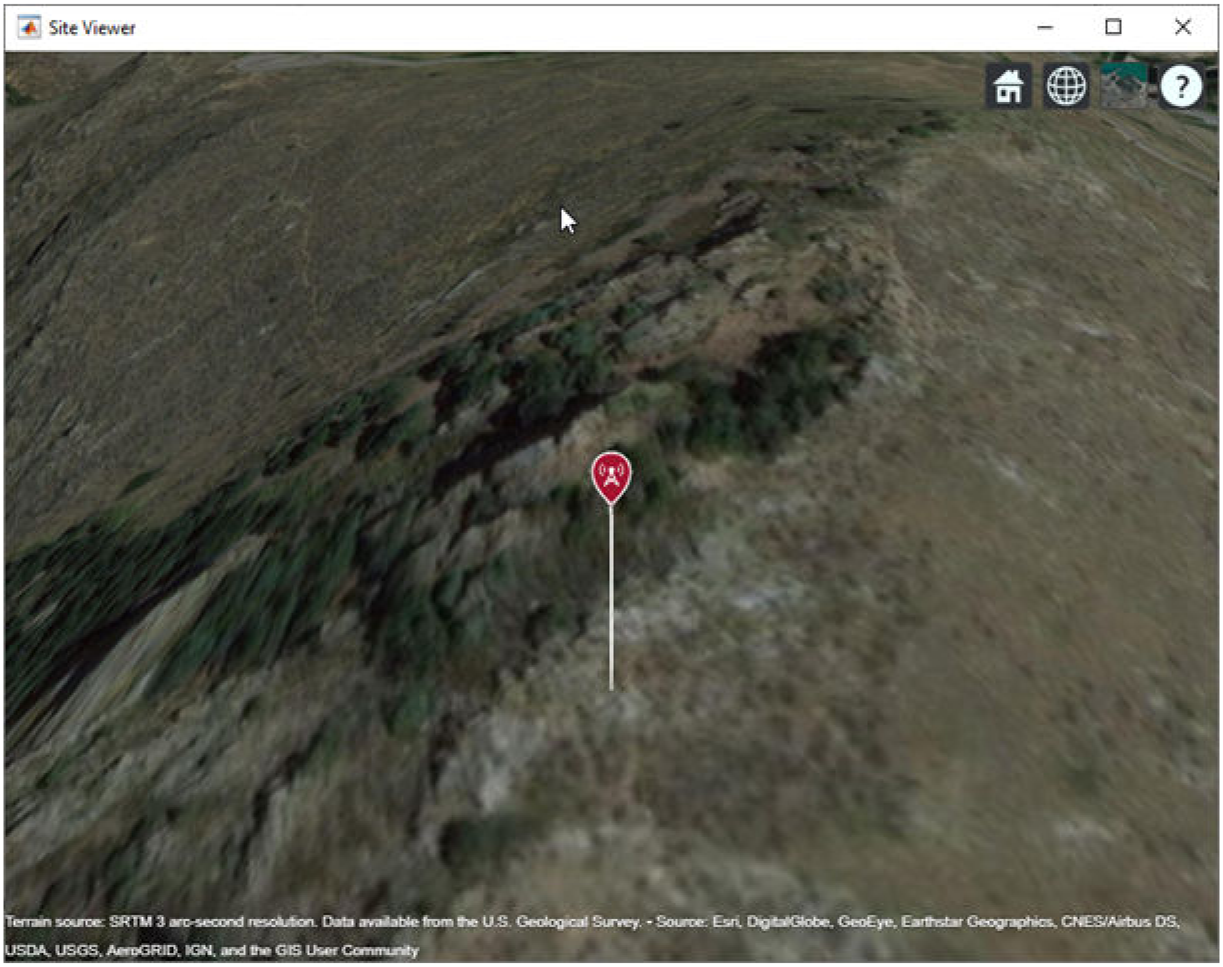
```
viewer = siteviewer("Terrain","southboulder");
```



Terrain source: SRTM 3 arc-second resolution. 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

Create a site with the terrain region.

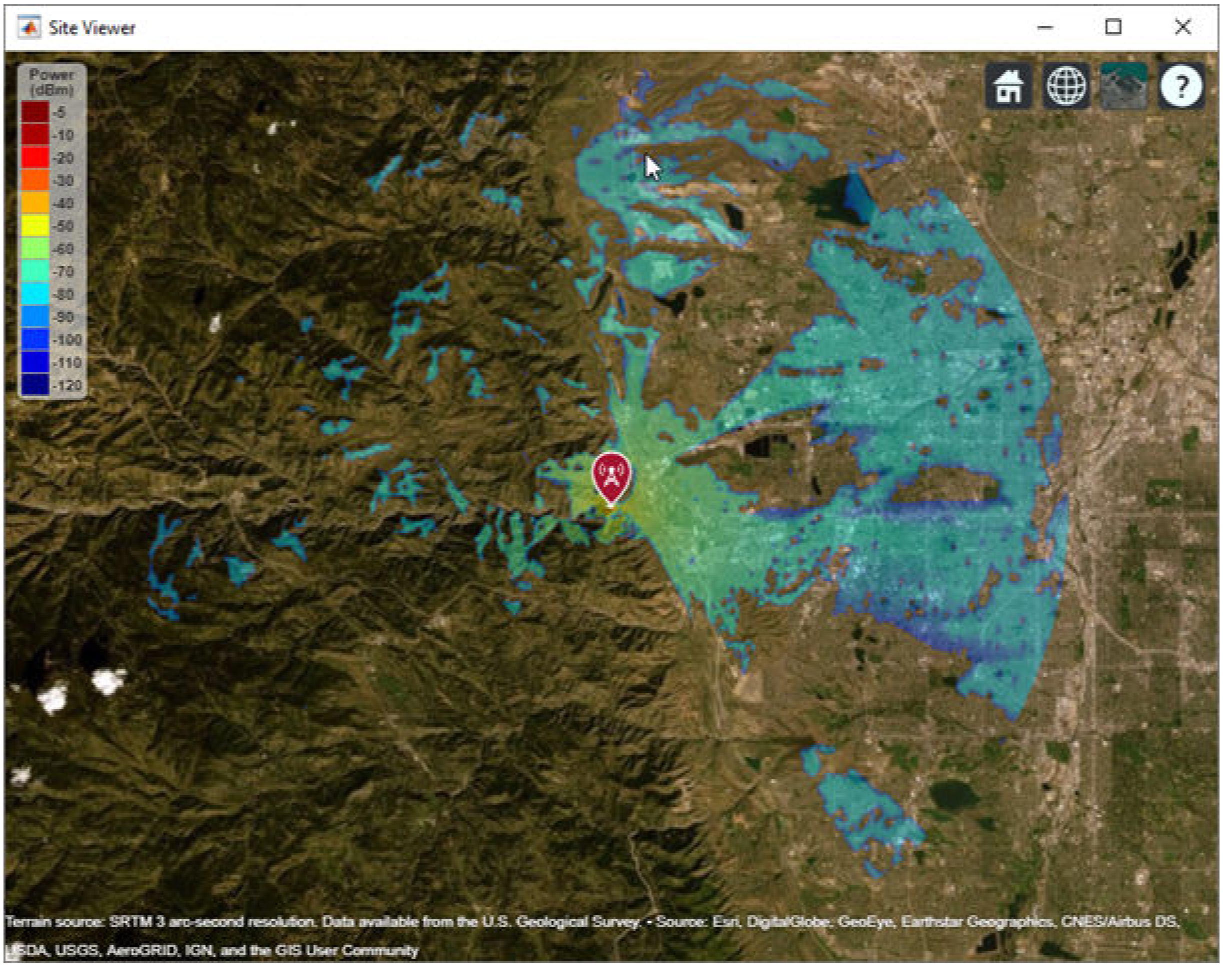
```
mtzion = txsite("Name","Mount Zion", ...
   "Latitude",39.74356, ...
   "Longitude",-105.24193, ...
   "AntennaHeight", 30);
show(mtzion)
```



Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. - Source: Esri, DigitalGlobe, 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, DigitalGlobe, 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` — List of DTED files

string scalar | character vector | cell array of character vectors

List of DTED files, specified as a string scalar, a character vector or a cell array of character vectors.

Note If you specify multiple files, they must combine to define a complete rectangular geographic region. If not, you must set the name-value pair `'FillMissing'` to `'true'`.

Data Types: `char` | `string`

Name-Value Pair Arguments

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

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`

See Also

`removeCustomTerrain` | `siteviewer`

Introduced in R2019a

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 site 1 and site

`[az,el] = angle(site1,site2,path)` returns the angles using a specified path type, either Euclidean or great circle path.

`[az,el] = angle(___,Name,Value)` returns the azimuth and elevation angles with additional options specified by name-value pairs.

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` or `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'` or `'greatcircle'`

Measurement path type, specified as one of the following:

- `'euclidean'`: Uses the shortest path through space connecting the antenna center positions of the site 1 and site 2.
- `'greatcircle'`: Uses the shortest path on the surface of the earth connecting the latitude and longitude locations of site 1 and site 2. This path uses a spherical Earth model.

Data Types: `char`

Name-Value Pair Arguments

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

Example: `'Map','siteviewer1'`

`Map` — Map for visualization or surface data

`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

Data Types: `char` | `string`

Output Arguments

`az` — Azimuth angle between site 1 and site 2

M-by-*N* arrays

Azimuth angle between site 1 and site 2, returned as *M*-by-*N* arrays in degrees. *M* is the number of sites in sites 2 and *N* is the number of sites in sites 1. The values range from `-180` to `180`.

el — Elevation angle between site 1 and site 2
M-by-*N* arrays

Elevation angle between site 1 and site 2, returned as *M*-by-*N* arrays in degrees. *M* is the number of sites in sites 2 and *N* is the number of sites in sites 1 The values range from `-90` to `90`.

When the path type specified is `'greatcircle'`, elevation angle is always zero.

See Also

`distance`

Introduced in R2017b

clearMap

Clear map visualizations

Syntax

```
clearMap(viewer)
```

Description

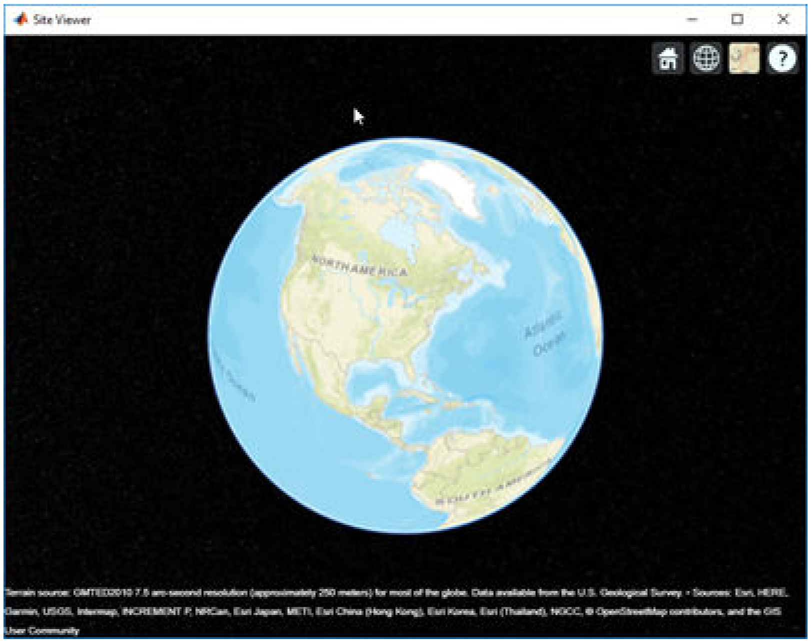
`clearMap(viewer)` removes all visualizations from the map.

Examples

View Transmitter Site On Site Viewer

Launch a Site Viewer with streets basemap.

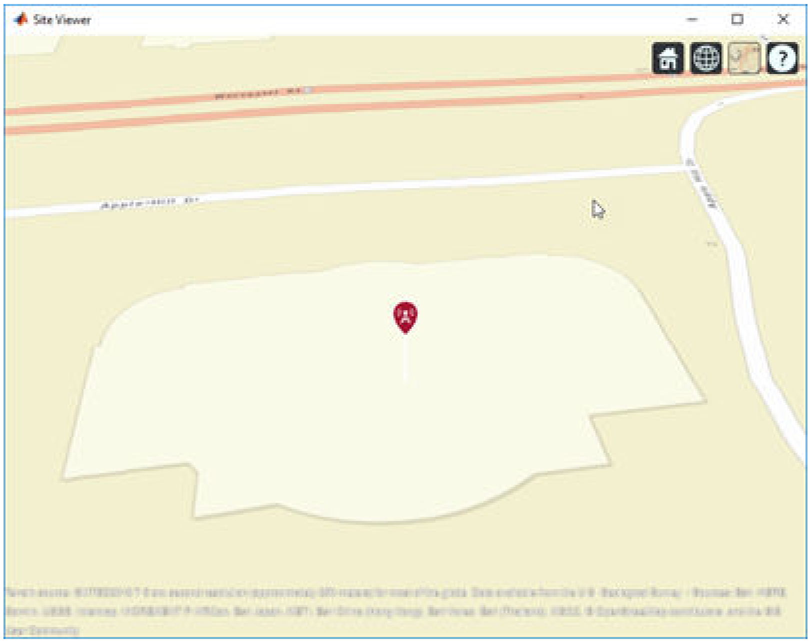
```
viewer = siteviewer("Basemap","streets");
```



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, © OpenStreetMap contributors, and the GIS User Community

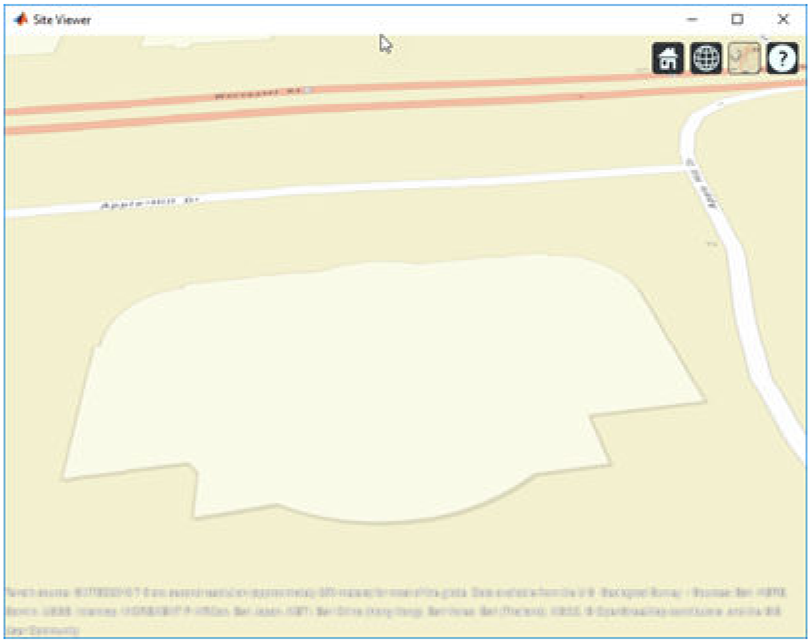
View a transmitter site on this map.

```
tx = txsite;
show(tx)
```



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.

See Also

`close` | `siteviewer`

Introduced in R2019a

close

Close map viewer window

Syntax

```
close(viewer)
```

Description

`close(viewer)` closes the map viewer window and deletes the handle

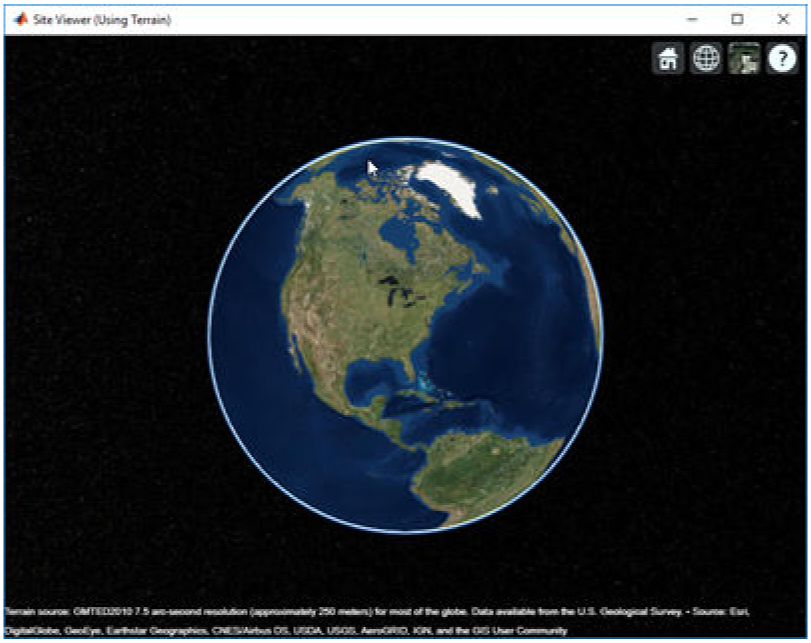
Examples

Compare Coverage Maps

Launch two Site Viewer windows.

One Site Viewer window uses the terrain model.

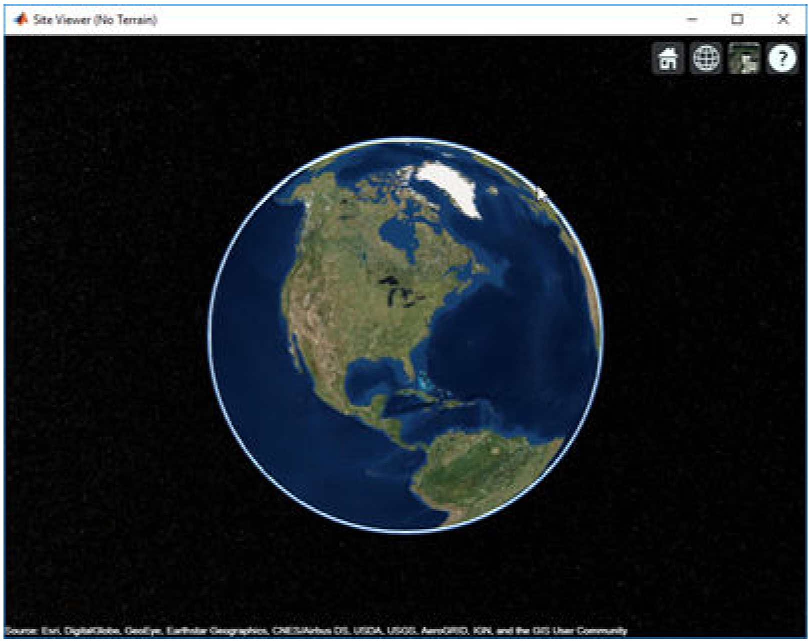
```
viewer1 = siteviewer("Terrain","gmted2010","Name","Site Viewer (Using Terrain)");
```



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

The second Site Viewer window does not use the terrain model.

```
viewer2 = siteviewer("Terrain","none","Name","Site Viewer (No Terrain)");
```



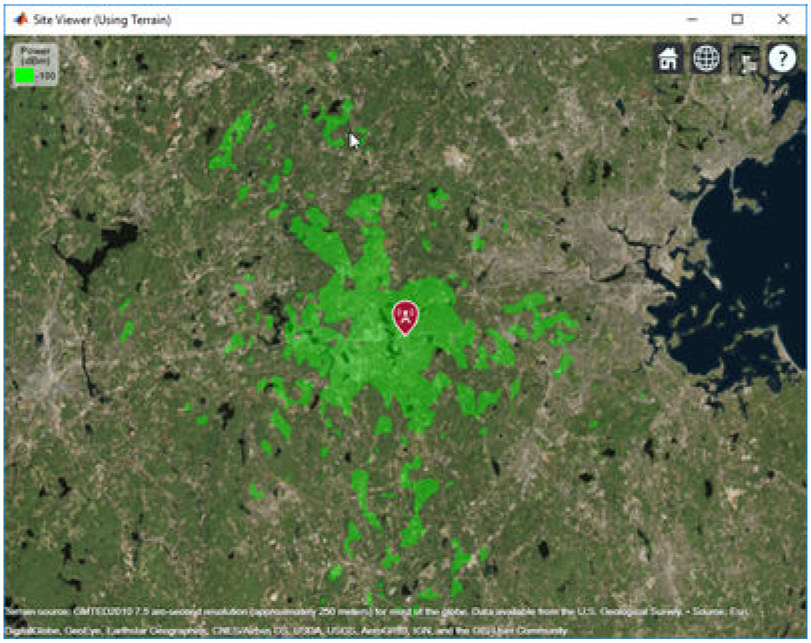
Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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.

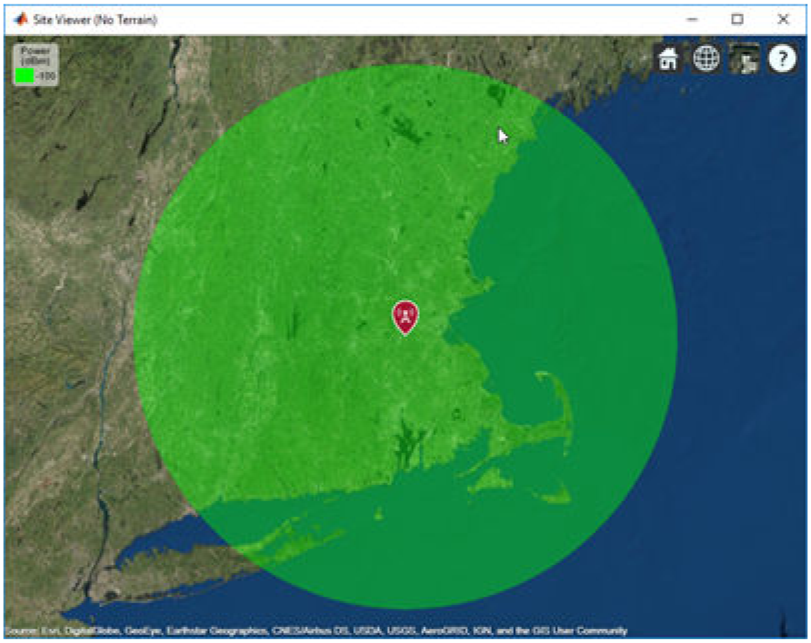
```
coverage(tx,"Map",viewer1)
```



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

The map without terrain uses the free-space model by default.

```
coverage(tx,"Map",viewer2)
```


Site Viewer (No Terrain)
Power (dBm)
-100
Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Close the maps.

```
close(viewer1)
close(viewer2)
```

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.

See Also

`clearMap` | `siteviewer`

Introduced in R2019a

coverage

Display coverage map

Syntax

```
coverage(tx)
coverage(tx,propmodel)
coverage(tx,rx)
coverage(tx,rx,propmodel)
coverage( ___ ,Name,Value, ___ )
pd = coverage(tx, ___ )
```

Description

`coverage(tx)` displays the coverage map for the transmitter site. Each colored contour of the map defines an area where the corresponding signal strength is transmitted to the mobile receiver.

`coverage(tx,propmodel)` displays the coverage map based on the specified propagation model.

`coverage(tx,rx)` displays the coverage map based on the receiver site properties.

`coverage(tx,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(tx, ___ )` returns computed coverage data in the propagation data object, `pd`. No plot is displayed and any graphical only name-value pairs are ignored.

Examples

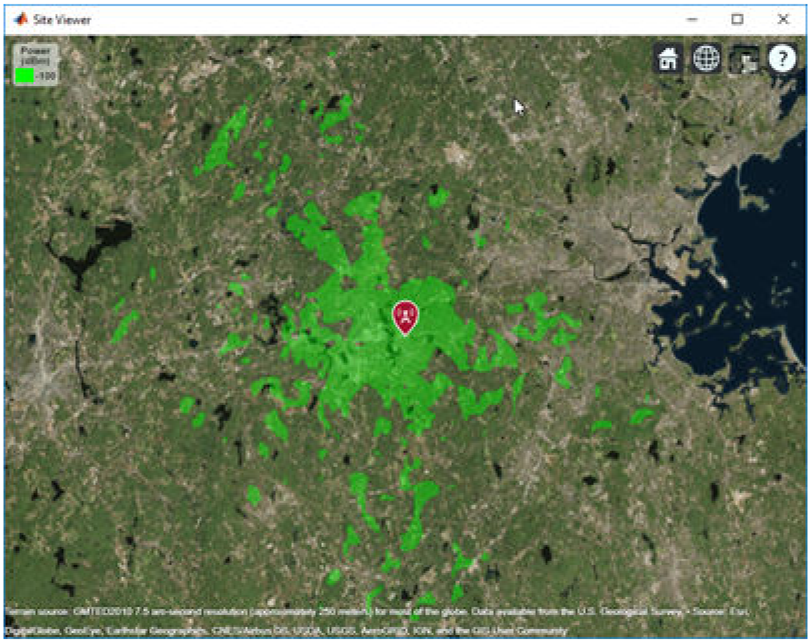
Coverage Map of Transmitter

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.

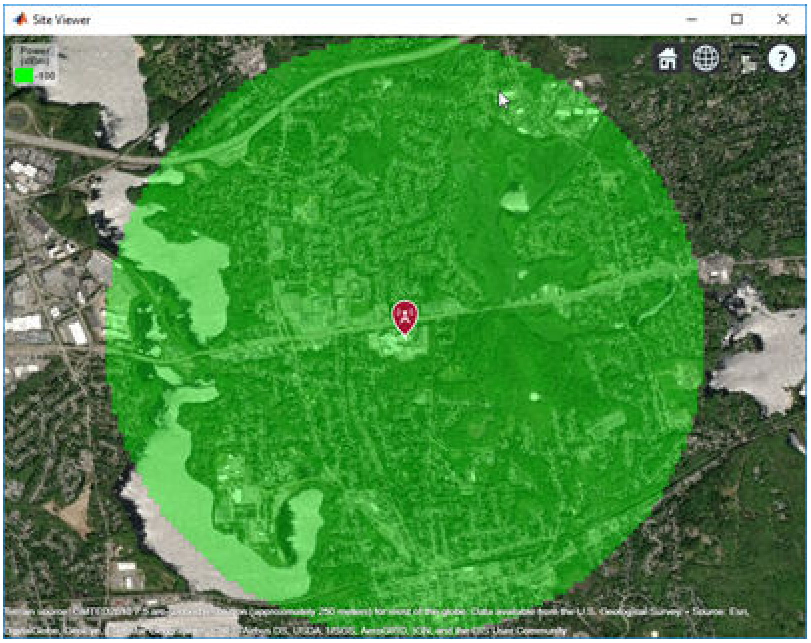
```
tx = txsite('Name','MathWorks', ...
        'Latitude', 42.3001, ...
        'Longitude', -71.3503);
```

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')
```



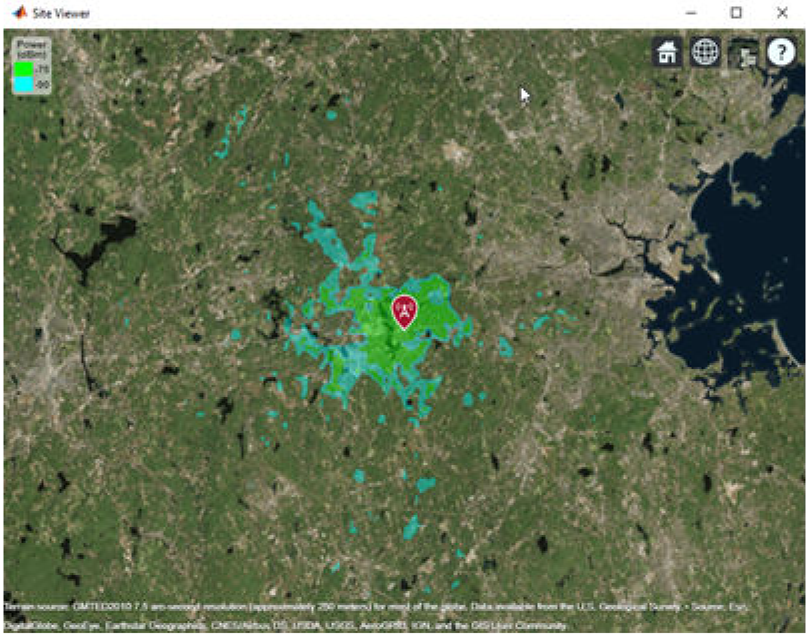
Coverage Map for Strong and Weak Signals

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)
🟩 -70
🟦 -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).

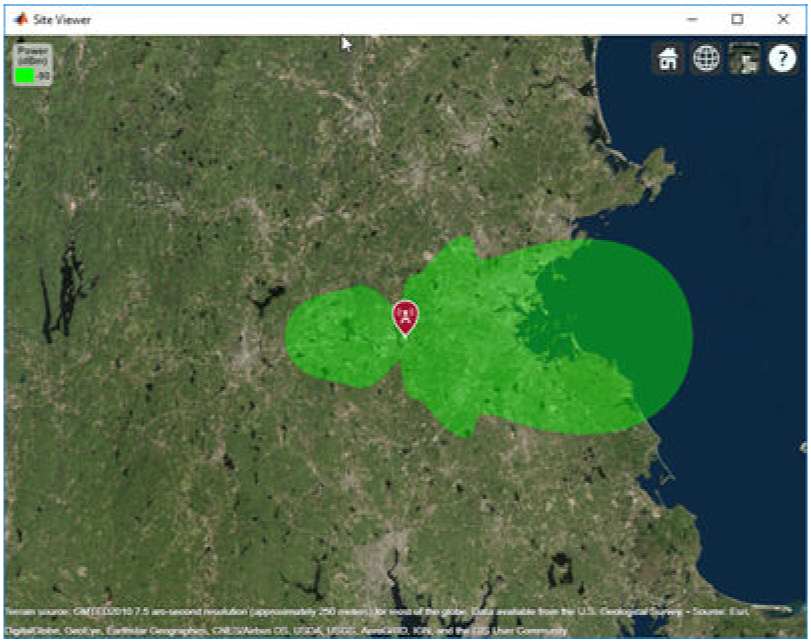
```
fq = 4.5e9;
y = design(yagiUda,fq);
y.Tilt = 90;
y.TiltAxis = 'y';
```

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.

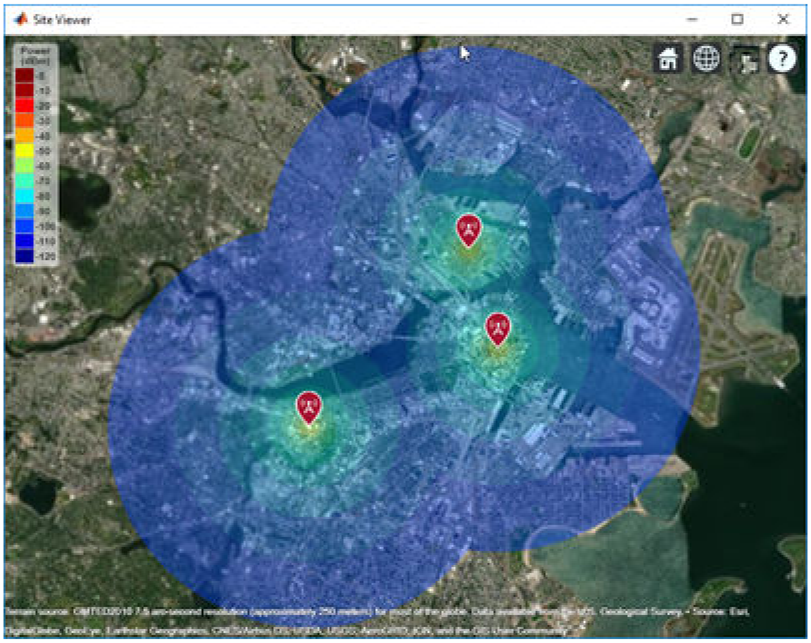
```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

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.

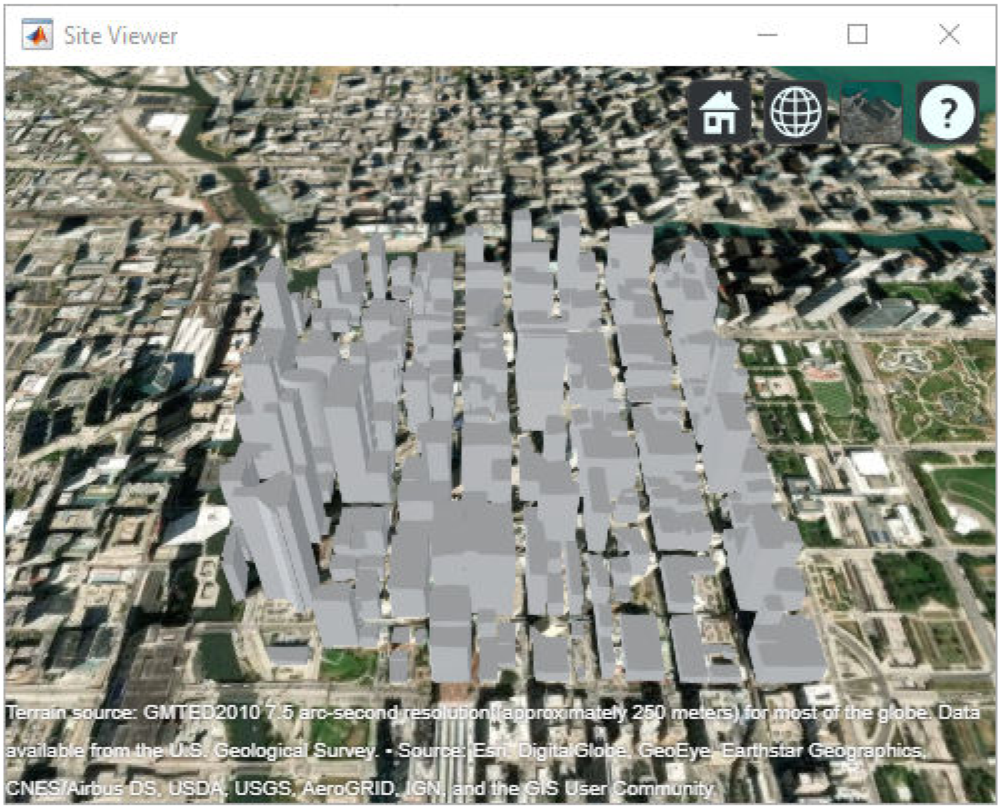
```
coverage(txs,'close-in','SignalStrengths',-100:5:-60)
```


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

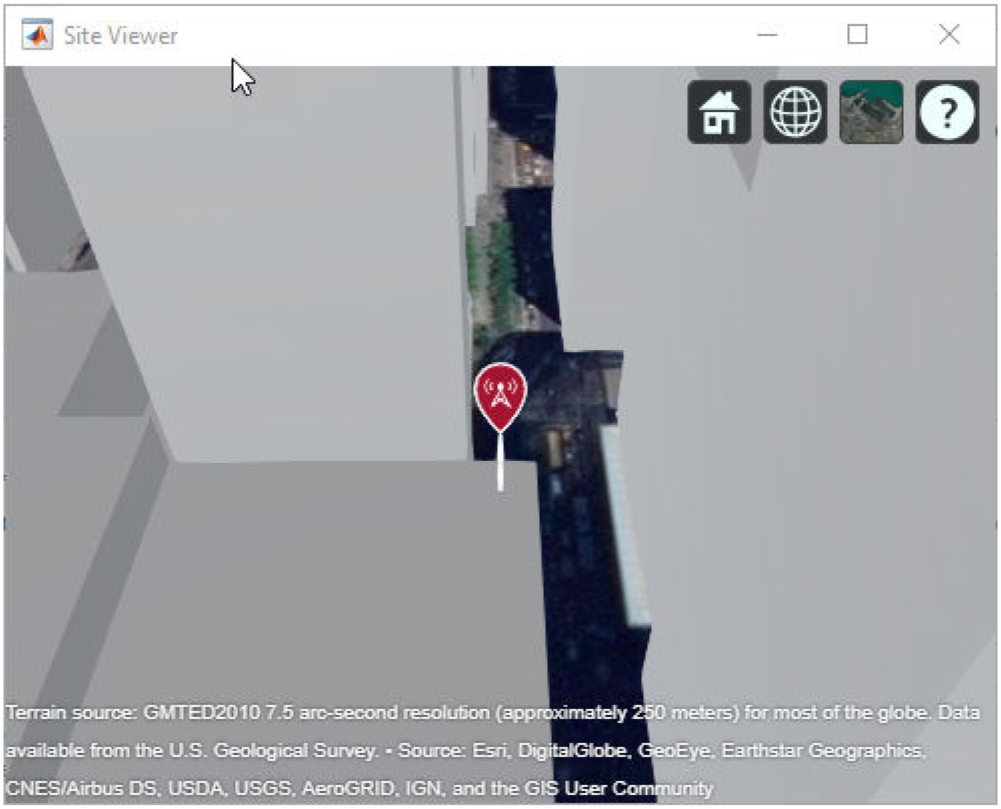
Launch Site Viewer using buildings in Chicago.

```
viewer = siteviewer("Buildings","chicago.osm");
```


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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, 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)
```

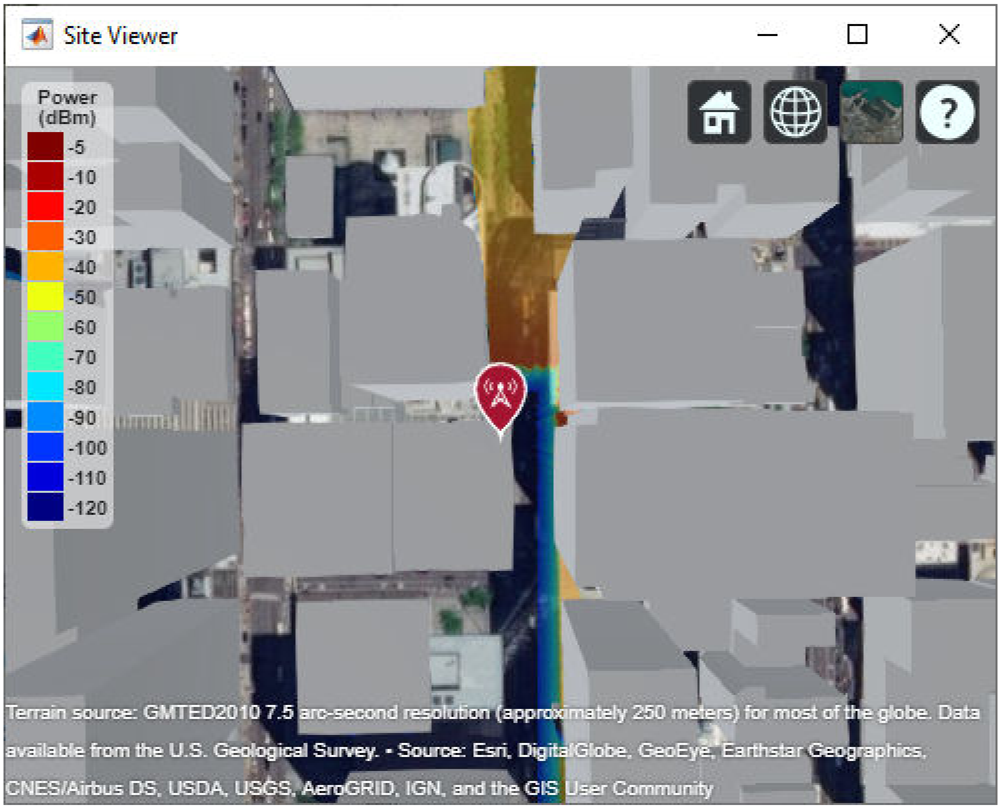


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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)
```

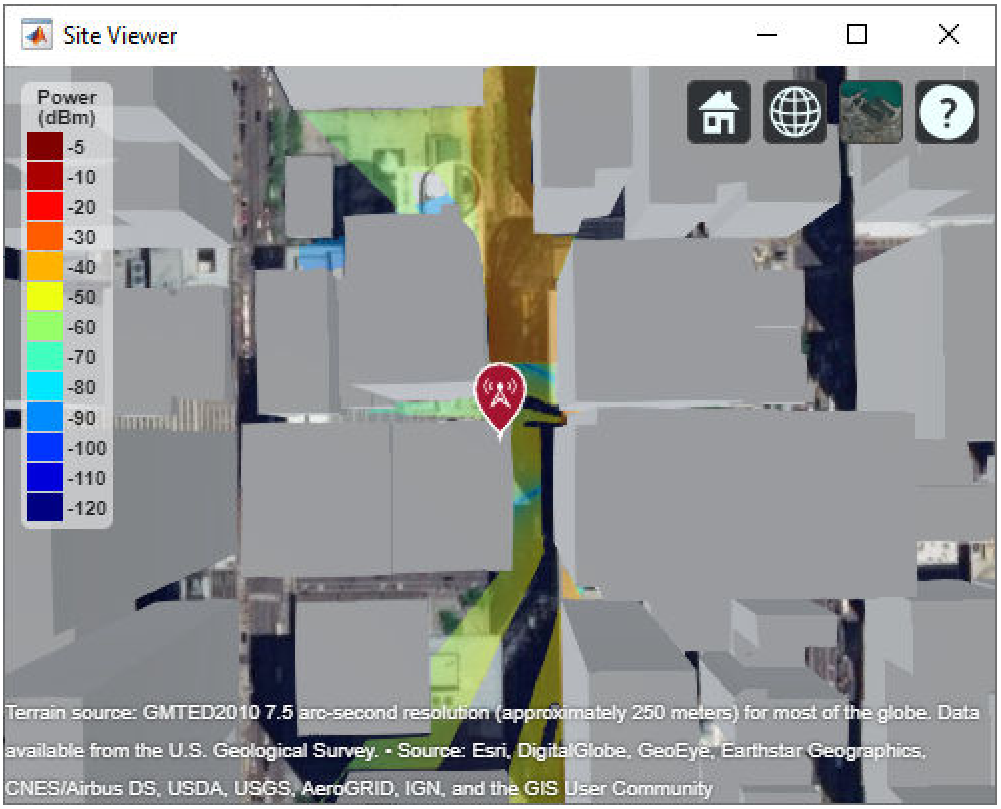

Site Viewer
Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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

Create a coverage map of the city using the ray tracing image method propagation model.

```
coverage(tx,"raytracing-image-method","SignalStrengths",-100:-5,"MaxRange",250,"Resolution",2)
```


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

This coverage map shows new regions that are in service due to reflected propagation paths.

Input Arguments

`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.

`rx` — Receiver site

`rxsite` object

Receiver site, specified as an `rxsite` object. You can also use the name-value pairs `'ReceiverGain'` and `'ReceiverAntennaHeight'` to specify the receiver values.

`propmodel` — Propagation model

character vector | string

Propagation model, specified as a character vector or string. You can also use the name-value pair `'PropagationModel'` to specify this parameter. You can also use the `propagationModel` function to define this input. The default propagation model is `'longeley-rice'` when terrain is enabled and `'freespace'` when terrain is disabled.

Data Types: `char` | `string`

Name-Value Pair Arguments

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

Example: `'Type','power'`

`Type` — Type of signal strength to compute

`'power'` (default) | `'efield'`

Type of signal strength to compute, specified as the comma-separated pair consisting of `'Type'` and `'power'` or `'efield'`.

When type is `'power'`, `SignalStrengths` is expressed in power units (dBm) of the signal at the mobile receiver input. When type is `'efield'`, `SignalStrengths` is expressed 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 the comma-separated pair consisting of `'SignalStrengths'` and a numeric vector.

Each strength uses different colored filled contour on the map. The default value is `-100` dBm if the `'Type'` name-value pair is `'power'` and `40` dBμV/m if `'Type'` is `'efield'`.

Data Types: `double`

`PropagationModel` — Propagation model to use for path loss calculations

`'longley-rice'` (default) | `'freespace'` | `'close-in'` | `'rain'` | `'gas'` | `'fog'` | `'raytracing-image-method'` | propagation model object

Propagation model to use for the path loss calculations, specified as the comma-separated pair consisting of `'PropagationModel'` and `'freespace'`, `'close-in'`, `'rain'`, `'gas'`, `'fog'`, `'longley-rice'`, `'raytracing-image-method'`, or as an object created using the `propagationModel` function. The default propagation model is `'longeley-rice'` when terrain is enabled and `'freespace'` when terrain is disabled. If `'raytracing-image-method'` is specified, the value of `'MaxNumReflections'` property must be lesser than 1.

Data Types: `char`

`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 propagation model type as shown:

Propagation Model	MaxRange
Basic or urban	Range of minimum value in `SignalStrengths`.
Terrain	30 km or distance to the furthest building.

Propagation Model	MaxRange
Multipath	500 m

Data Types: `double`

`Resolution` — Resolution of coverage map

`'auto'` (default) | numeric scalar

Resolution of coverage map, specified as the comma-separated pair consisting of `'Resolution'` and 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 the comma-separated pair consisting of `'ReceiverGain'` and 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 the `'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 the comma-separated pair consisting of `'ReceiverAntennaHeight'` and 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

Filled contours color of coverage map, specified as the comma-separated pair consisting of `'Colors'` and an *M*-by-3 array of RGB triplets, an array of strings, or a cell array of character vectors.

Colors are assigned element-wise to `'SignalStrengths'` values for coloring the corresponding filled contours.

`'Colors'` cannot be used with `'ColorLimits'` or `'ColorMap'`.

For more information, see `ColorSpec (Color Specification)`.

Data Types: `char` | `string` | `double`

`ColorLimits` — Color limits for colormap

two-element vector

Color limits for colormap, specified as the comma-separated pair consisting of `'ColorLimits'` and 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'` name-value pair is `'power'` and `[20 135]` if `'Type'` is `'efields'`.

`'ColorLimits'` cannot be used with `'Color'`.

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 the comma-separated pair consisting of `'ColorMap'` and 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 the comma-separated pair consisting of `'ShowLegend'` and `true` or `false`.

Data Types: `logical`

`Transparency` — Transparency of coverage map

`0.4` (default) | numeric scalar

Transparency of coverage map, specified as the comma-separated pair consisting of `'Transparency'` and a numeric scalar in the range `0` to `1`. `0` is transparent and `1` is opaque.

Data Types: `double`

`Map` — Map for visualization or surface data

`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

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"`.

See Also

`link` | `propagationModel` | `sigstrength` | `sinr`

Topics
`ColorSpec (Color Specification)`

Introduced in R2017b

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 Euclidean or great circle path.

`d = distance( ___ ,Name,Value)` returns the distance with additional options specified by name-value pairs.

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 or rxsite object

Transmitter or receiver site, specified as a `txsite` or `rxsite`. 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'`: Uses the shortest path through space that connects the antenna center positions of the site 1 and site 2.
- `'greatcircle'`: Uses the shortest path on the surface of the earth that connects the latitude and longitude locations of site 1 and site 2. This path uses a spherical Earth model.

Data Types: `char`

Name-Value Pair Arguments

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

Example: `'Map','siteviewer1'`

`Map` — Map for visualization or surface data
`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

Data Types: `char` | `string`

Output Arguments

d — Distance between sites
M-by-*N* numeric array

Distance between sites, returned as *M*-by-*N* arrays in degrees. *M* is the number of sites in site 2 and *N* is the number of sites in site 1.

See Also

`angle`

Introduced in R2017b

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.

`z = elevation( ___ ,Name,Value)` returns the ground elevation of the antenna in meters with additional options specified by name-value pairs.

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 = 1.8675e+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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Map','siteviewer1'`

Map — Map for visualization or surface data

`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of 'Map and a siteviewer object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else 'gmted2010' if the function is called with an output argument.

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.

See Also

angle | distance | rxsite | txsite

Introduced in R2018b

hide

Hide site location on map

Syntax

```
hide(site)
hide( ___ ,Name,Value)
```

Description

`hide(site)` hides the site location of the antenna site on a map.

`hide( ___ ,Name,Value)` hides the site location with additional options specified by one or more name-value pairs.

Examples

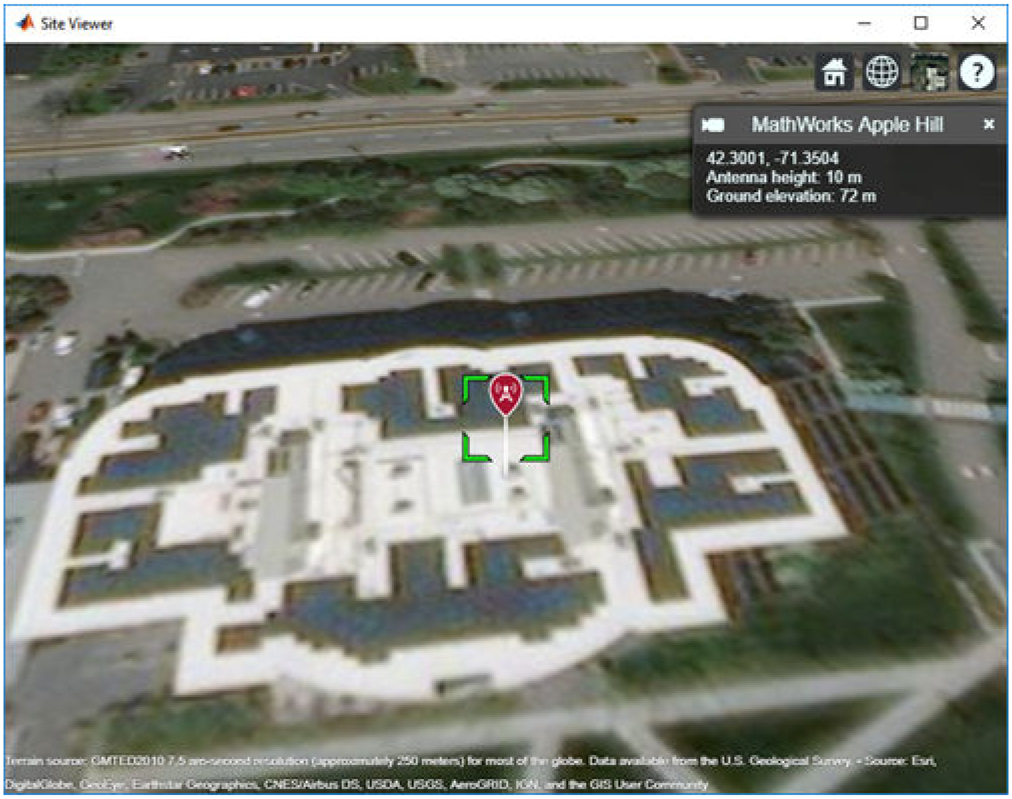
Show and Hide Transmitter Site

Create a transmitter site.

```
tx = txsite('Name','MathWorks Apple Hill',...
       'Latitude',42.3001, ...
       'Longitude',-71.3504);
```

Show the transmitter site.

```
show(tx)
```



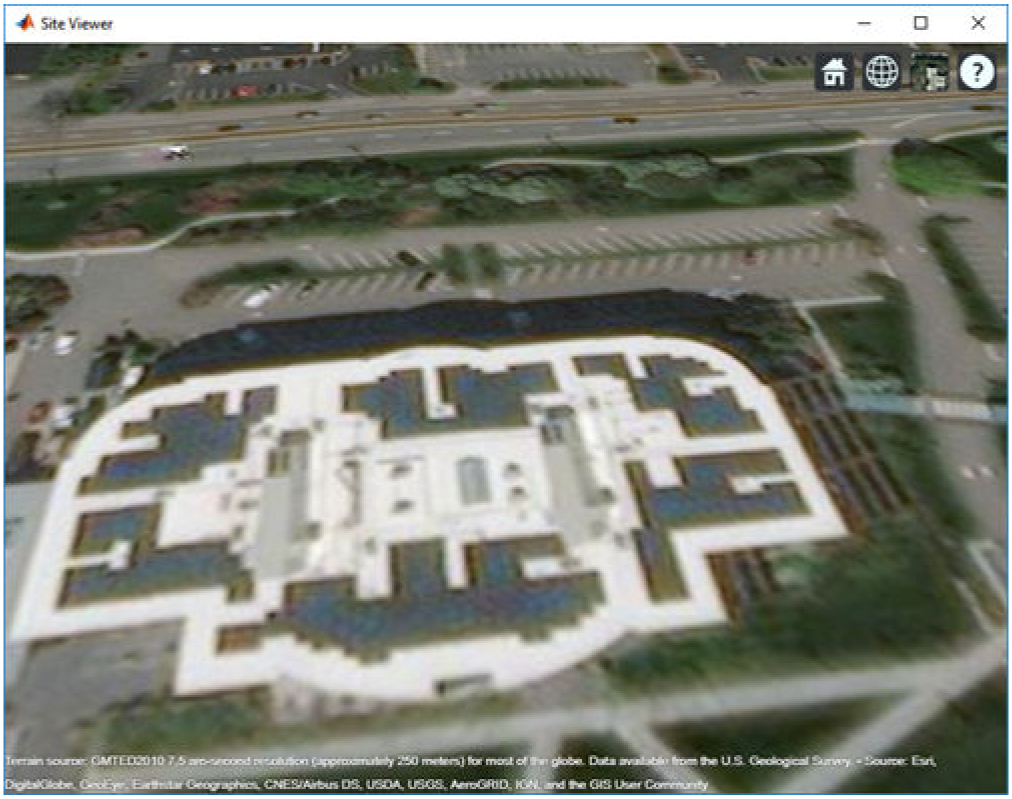
MathWorks Apple Hill

42.3001, -71.3504
Antenna height: 10 m
Ground elevation: 72 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

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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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

Example: 'Map','siteviewer1'

Map — Map for visualization or surface data
siteviewer object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of 'Map and a siteviewer object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else 'gmted2010' if the function is called with an output argument.

Data Types: char | string

See Also

`show`

Introduced in R2017b

link

Display communication link on map

Syntax

```
link(rx,tx)
link(rx,tx,propmodel)
link( ___ ,Name,Value)
status = link( ___ )
```

Description

`link(rx,tx)` plots a one-way point-to-point communication link between a receiver site and transmitter site. The plot is color coded to identify the link success status.

`link(rx,tx,propmodel)` plots the communication link based on the specified propagation model.

`link( ___ ,Name,Value)`plots 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 Transmitter and Receiver

Create a transmitter site.

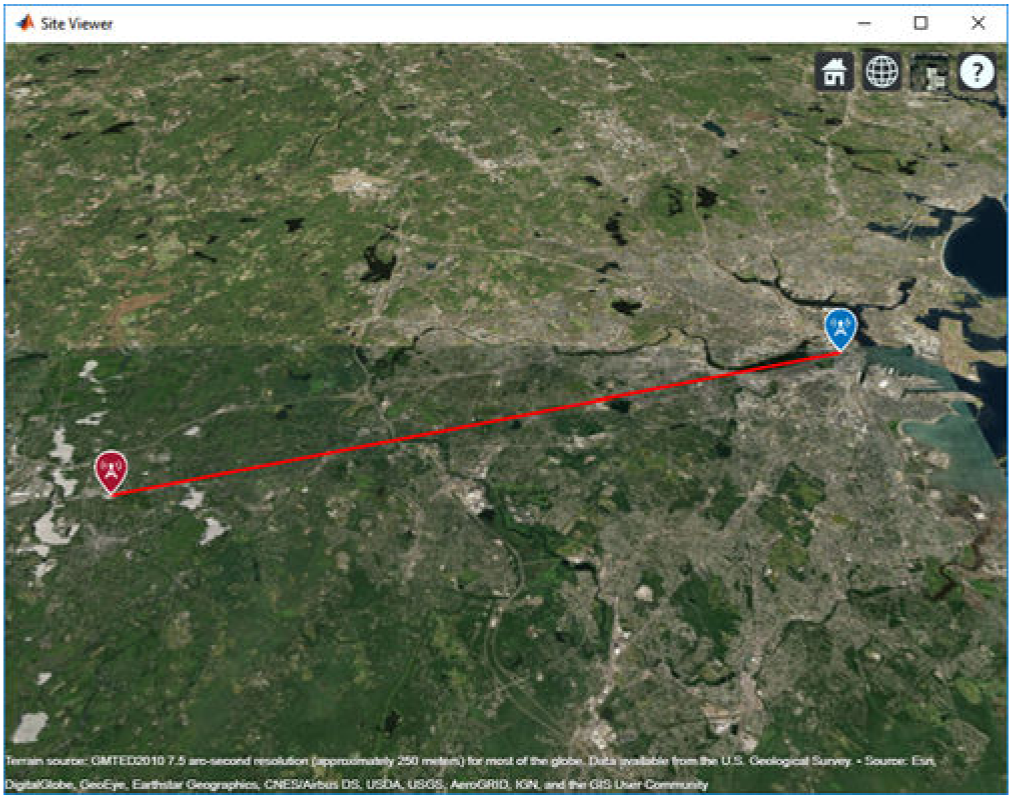
```
tx = txsite('Name','MathWorks', ...
        'Latitude', 42.3001, ...
        'Longitude', -71.3503);
```

Create a receiver site with 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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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

character vector | string

Propagation model, specified as a character vector or string. You can also use the name-value pair `'PropagationModel'` to specify this parameter.

Data Types: `char` | `string`

Name-Value Pair Arguments

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

Example: `'Type','power'`

`PropagationModel` — Propagation model to use for path loss calculations

`'longley-rice'` (default) | `'freespace'` | `'close-in'` | `'rain'` | `'gas'` | `'fog'` | `'raytracing-image-method'` | propagation model object

Propagation model to use for the path loss calculations, specified as the comma-separated pair consisting of `'PropagationModel'` and `'freespace'`, `'close-in'`, `'rain'`, `'gas'`, `'fog'`, `'longley-rice'`, `'raytracing-image-method'`, or as an object created using the `propagationModel` function. The default propagation model is `'longeley-rice'` when terrain is enabled and `'freespace'` when terrain is disabled. If `'raytracing-image-method'` is specified, the value of `'MaxNumReflections'` property must be lesser than 1.

Data Types: `char`

`SuccessColor` — Color of successful links

`'green'` (default) | RGB triplet | character vector

Color of successful links, specified as the comma-separated pair consisting of `'SuccessColor` and an RGB triplet or character vector. For more information, see `ColorSpec (Color Specification)`.

Data Types: `char` | `double`

`FailColor` — Color of unsuccessful links

`'red'` (default) | RGB triplet | character vector

Color of unsuccessful links, specified as the comma-separated pair consisting of `'FailColor` and RGB triplet or character vector. For more information, see `ColorSpec (Color Specification)`.

Data Types: `char` | `double`

`Map` — Map for visualization or surface data

`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

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.

See Also

`coverage` | `los` | `propagationModel` | `sigstrength` | `sinr`

Topics

`ColorSpec (Color Specification)`

Introduced in R2017b

location

Location coordinates at a given 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.

`[ ___ ] = 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.

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
```

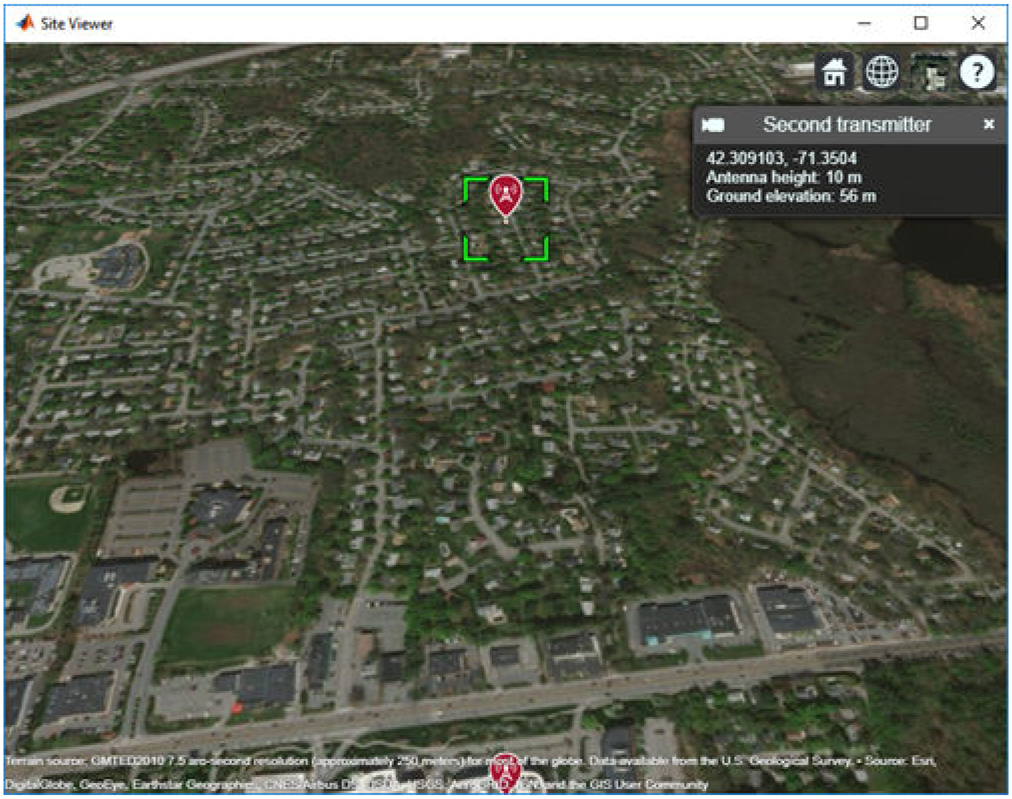
```
lon = -71.3504
```

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])
```


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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Second transmitter

42.309103, -71.3504
Antenna height: 10 m
Ground elevation: 56 m

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.

`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]`.

See Also

`angle` | `distance` | `rxsite` | `txsite`

Introduced in R2018a

los

Plot or compute the line-of-sight (LOS) visibility between sites on a map

Syntax

```
los(site1,site2)
los(site1,site2,Name,Value)
vis = los(site1,site2,Name,Value)
```

Description

`los(site1,site2)` plots the LOS from site 1 to site 2. The plot is color coded to identify the visibility of the points along the LOS.

`los(site1,site2,Name,Value)` sets properties using one or more name-value pairs. For example, `los(site1,site2,'ObstructedColor','red')` plots the LOS using 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

Plot the LOS from the MathWorks Apple Hill campus to the MathWorks Lakeside campus.

Create a transmitter site with an antenna of height 30 m.

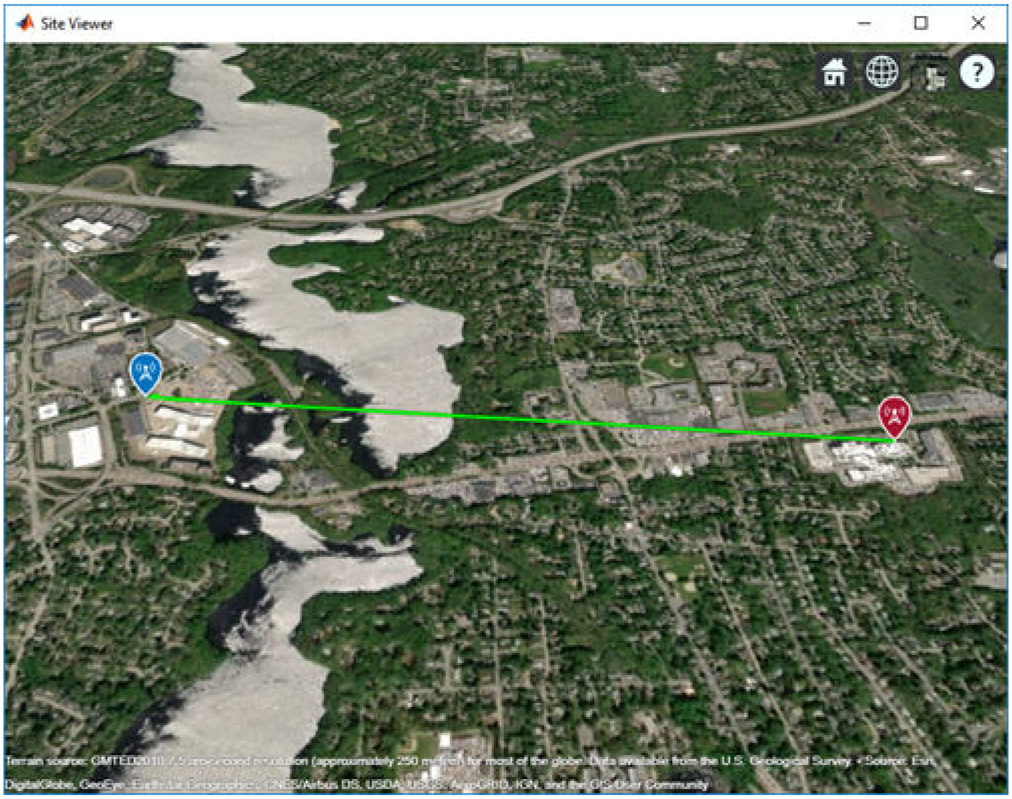
```
tx = txsite('Name','MathWorks Apple Hill',...
          'Latitude',42.3001,'Longitude',-71.3504,'AntennaHeight',30);
```

Create a receiver site with an antenna at ground level.

```
rx = rxsite('Name','MathWorks Lakeside', ...
           'Latitude',42.3021,'Longitude',-71.3764);
```

Plot the LOS between the two sites.

```
los(tx,rx);
```


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, DigitalGlobe, 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.

```
tx = txsite('Name','MathWorks Apple Hill',...
          'Latitude',42.3001,'Longitude',-71.3504,'AntennaHeight',30);
```

Create two receiver sites with antennas at ground level.

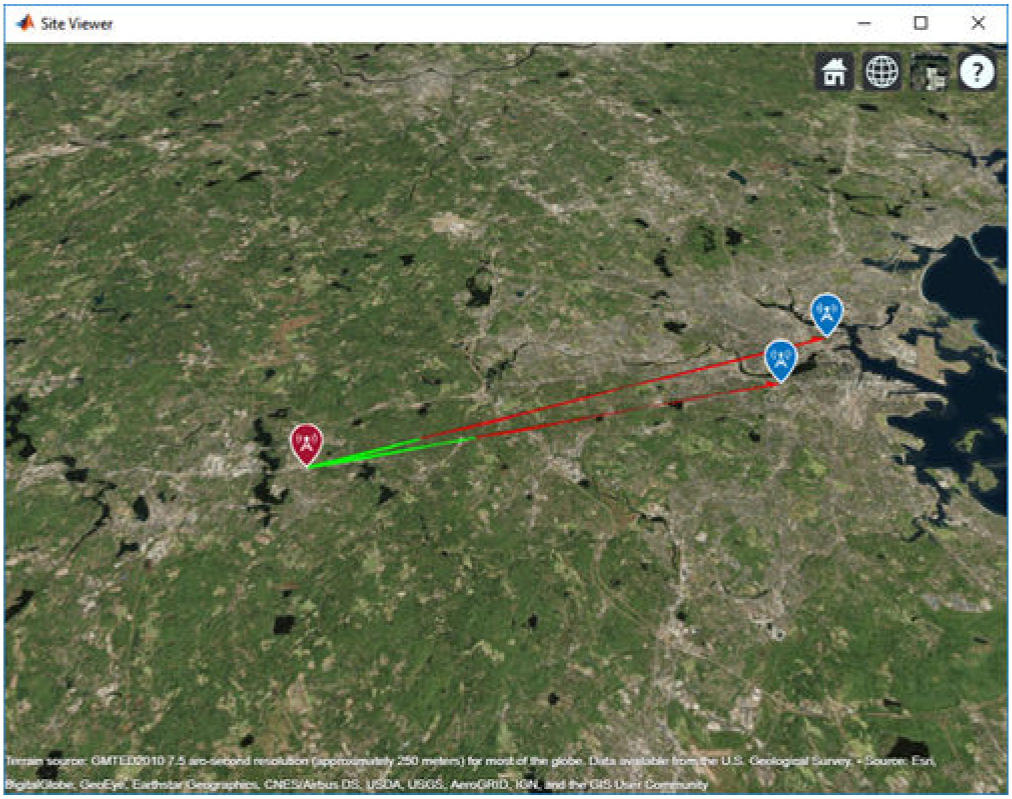
```
names = ["Fenway Park","Bunker Hill Monument"];
lats = [42.3467,42.3763];
lons = [-71.0972,-71.0611];
```

Create the receiver site array.

```
rxs = rxsite('Name', names,...
      'Latitude',lats,...
      'Longitude',lons);
```

Plot the lines of sight to the receiver sites. The red portion of the LOS represents obstructed visibility.

```
los(tx,rxs);
```


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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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

Example: `'ObstructedColor','blue'`

`VisibleColor` — Plot color for successful visibility

`'green'` (default) | RGB triplet | character vector | color name string

Plot color for successful visibility, specified as an RGB triplet, a character vector, or a color name specified as a string. For more information, see `ColorSpec (Color Specification)`.

ObstructedColor — Plot color for blocked visibility

'red' (default) | RGB triplet | character vector | color name string

Plot color for blocked visibility, specified as an RGB triplet, a character vector, or a color name specified as a string. For more information, see ColorSpec (Color Specification).

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 | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of 'Map and a siteviewer object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else 'gmted2010' if the function is called with an output argument.

Data Types: char | string

Output Arguments

vis — Status of LOS visibility

'true' | 'false' | *n*-by 1 logical array

Status of LOS visibility, returned as 'true' or 'false'. If there are multiple target sites, the function returns a logical array of *n*-by-1.

See Also

angle | distance | link

Topics

ColorSpec (Color Specification)

Introduced in R2018a

pathloss

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 = 127.1559
```

Input Arguments

propmodel — Propagation model

character vector or string

Propagation model, specified as a character vector or string.

Data Types: char

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

Example: 'Map','none'

Map — Map for visualization or surface data

siteviewer object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of 'Map and a siteviewer object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else 'gmted2010' if the function is called with an output argument.

Data Types: char | string

Output Arguments

pl — Path loss

scalar | *M*-by-*N* arrays

Path loss, returned as a scalar or *M*-by-*N* arrays with each element 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* struct array | *M*-by-*N* cell array containing vector of structs in each cell

Information corresponding to each propagation path, returned as a *M*-by-*N* cell array containing vector of structs in each cell for ray-tracing-image-method propagation model and *M*-by-*N* struct array fro 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 raytrtacing-image-method.

Angle values are defined using the antenna's local East-North-Up coordinate system. Azimuth angle is measured counter-clockwise from east in the range 0 to 360, and elevation angle is measured from the horizontal plane in the range -90 to 90.

See Also

propagationModel | range

Introduced in R2017b

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)` sets properties using one or more name-value pairs. For example, `pm = propagationModel('rain','RainRate',96)` creates a rain propagation model with a rain rate of 96 mm/h. Enclose each property name in quotes.

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 transmitter site.

```
tx = txsite
```

```
tx = 
  txsite with properties:

                   Name: 'Site 1'
               Latitude: 42.3001
              Longitude: -71.3504
                Antenna: 'isotropic'
           AntennaAngle: 0
          AntennaHeight: 10
             SystemLoss: 0
    TransmitterFrequency: 1.9000e+09
        TransmitterPower: 10
```

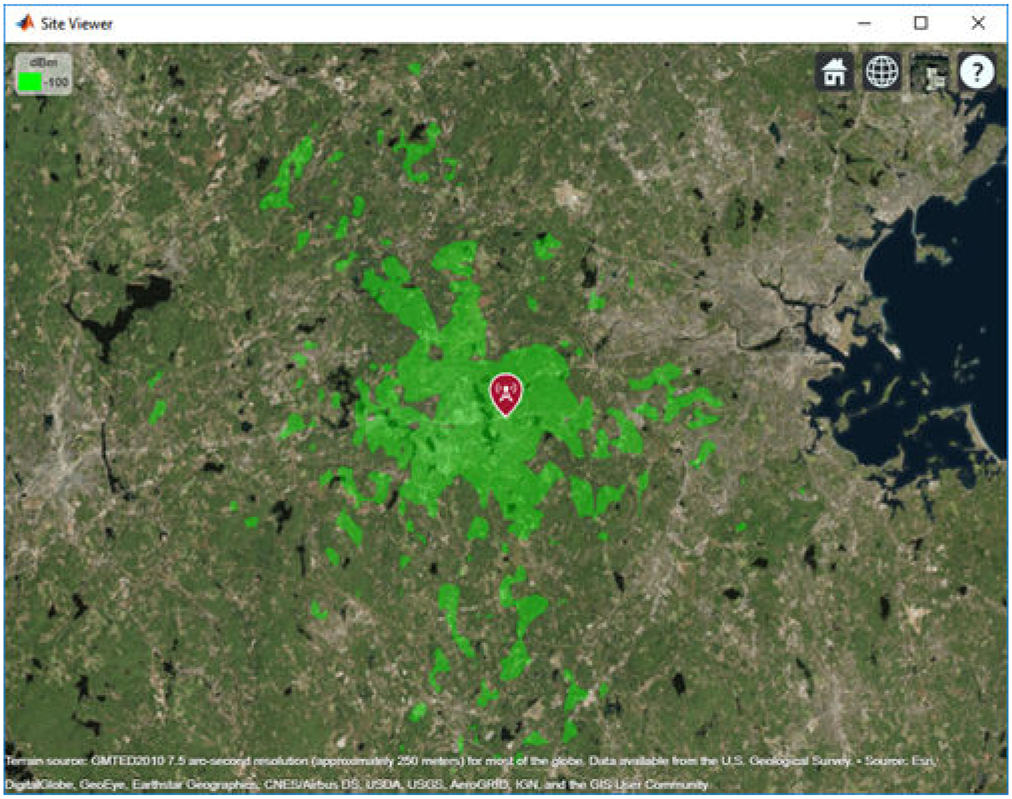
Create a Longley-Rice propagation model using the `propagationModel` function.

```
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 using the defined propagation model.

```
coverage(tx,'PropagationModel',pm)
```


Site Viewer
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

Input Arguments

`modelname` — Type of propagation model

`'freespace'` | `'rain'` | `'gas'` | `'fog'` | `'close-in'` |`'longley-rice'`

Type of propagation model:

- `'freespace'` – Free space propagation model
- `'rain'` – Rain propagation model. For more information, see [3].
- `'gas'` – Gas propagation model
- `'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 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 irregular terrain, including buildings. Path loss is calculated from free-space loss, terrain diffraction, ground reflection, refraction through atmosphere, tropospheric scatter, and atmospheric absorption. The 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 at 200 GHz.
- `'raytracing-image-method'` – The ray tracing propagation model is a multipath propagation model that uses ray tracing analysis to compute propagation paths and their corresponding path losses. Path loss is calculated from free-space loss, reflection loss due to material, and antenna polarization loss. The ray tracing analysis uses the method of images, which includes surface reflections but does not include effects from refraction, diffraction, or scattering. This model is valid from 100 MHz to 100 GHz.

You can use the following functions on RF propagation models:

- `range` – Calculate the range of the radio wave under different propagation scenarios. `range` function does not support Longley-Rice or TIREM propagation models. This function does not support the TIREM or `'raytracing-image-method'` propagation models.
- `pathloss` – Calculate the path loss of radio wave propagation between the transmitter and receiver sites under different propagation scenarios.
- `add` – Add propagation models.

Data Types: `char`

Name-Value Pair Arguments

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

Example: `'RainRate',50`

Rain

`RainRate` — Rain rate

`16` (default) | positive scalar

Rain rate, specified as a positive scalar in millimeters per hour (mm/h).

Dependencies

To specify `'RainRate'`, you must specify `'rain'` propagation model.

Data Types: `double`

`Tilt` — Polarization tilt angle of the signal

`0` (default) | scalar

Polarization tilt angle of the signal, specified as scalar in degrees.

Dependencies

To specify `'Tilt'`, you must specify `'rain'` propagation model.

Data Types: `double`

Gas

`Temperature` — Air temperature

`15` (default) | scalar

Air temperature, specified as a scalar in Celsius (C).

Dependencies

To specify `'Temperature'`, you must specify `'gas'` propagation model.

Data Types: `double`

`AirPressure` — Dry air pressure

`101300` (default) | scalar

Dry air pressure, specified as a scalar in pascals (Pa).

Dependencies

To specify `'AirPressure'`, you must specify `'gas'` propagation model.

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).

Dependencies

To specify `'WaterDensity'`, you must specify `'gas'` propagation model.

Data Types: `double`

Fog

`Temperature` — Air temperature

`15` (default) | scalar

Air temperature, specified as a scalar in Celsius (C).

Dependencies

To specify `'Temperature'`, you must specify `'fog'` propagation model.

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).

Dependencies

To specify `'WaterDensity'`, you must specify `'fog'` propagation model.

Data Types: `double`

Close-In

ReferenceDistance — Free-space reference distance

1 (default) | scalar

Free-space reference distance, specified as a scalar in meters.

Dependencies

To specify 'ReferenceDistance', you must specify the 'close-in' propagation model.

Data Types: double

PathLossExponent — Path loss exponent

2.9 (default) | scalar

Path loss exponent, specified as a scalar.

Dependencies

To specify 'PathLossExponent', you must specify 'close-in' propagation model.

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).

Dependencies

To specify 'Sigma', you must specify 'close-in' propagation model.

Data Types: double

NumDataPoints — Number of data points

1869 (default) | integer

Number of data points of zero-mean Gaussian random variable, specified as an integer.

Dependencies

To specify 'NumPoints', you must specify 'close-in' propagation model.

Data Types: double

Note The close-in model is valid for distances greater than or equal to the 'ReferenceDistance' property. If a distance less than the 'ReferenceDistance' is used, path loss is 0.

Longley-Rice

AntennaPolarization — Polarization of transmitter and receiver antennas

'horizontal' (default) | 'vertical'

Polarization of transmitter and receiver antennas, specified as 'horizontal' or 'vertical'. Both antennas are assumed to have the same polarization. This value is used to calculate path loss due to ground reflection.

Dependencies

To specify `'AntennaPolarization'`, you must specify `'longley-rice'` propagation model.

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). This value is used to calculate path loss due to ground reflection. The default value corresponds to average ground.

Dependencies

To specify `'GroundConductivity'`, you must specify `'longley-rice'` propagation model.

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. This value is used to calculate the path loss due to ground reflection. The default value corresponds to average ground.

Dependencies

To specify `'GroundPermittivity'`, you must specify `'longley-rice'` propagation model.

Data Types: `double`

`AtmosphericRefractivity` — Atmospheric refractivity near ground
`301` (default) | scalar

Atmospheric refractivity near the ground, specified as a scalar in N-units. This value is used to calculate the path loss due to refraction through the atmosphere and tropospheric scatter. The default value corresponds to average atmospheric conditions.

Dependencies

To specify `'AtmosphericRefractivity'`, you must specify `'longley-rice'` propagation model.

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. This value is used to calculate the variability due to changing atmospheric conditions. The default value corresponds to average atmospheric conditions in a particular climate zone.

Dependencies

To specify `'ClimateZone'`, you must specify `'longley-rice'` propagation model.

Data Types: `char` | `string`

`TimeVariabilityTolerance` — Time variability tolerance level
`0.5` (default) | scalar

Time variability tolerance level of the path loss, specified as a scalar between [0.001, 0.999]. Time variability occurs due to changing atmospheric conditions. This value gives the required system reliability or the fraction of time during which the actual path loss is expected to be less than or equal to model prediction. For more information, see [5].

Dependencies

To specify `'TimeVariabilityTolerance'`, you must specify `'longley-rice'` propagation model.

Data Types: `double`

`SituationVariabilityTolerance` — Situation variability tolerance level

`0.5` (default) | scalar

Situation variability tolerance level of the path loss, specified as a scalar in between [0.001, 0.999]. Situation variability occurs due to uncontrolled or hidden random variables. This value gives the required system confidence or the fraction of similar situations for which the actual path loss is expected to be less than or equal to the model prediction. For more information, see [5].

Dependencies

To specify `'SituationVariabilityTolerance'`, you must specify `'longley-rice'` propagation model.

Data Types: `double`

TIREM

`AntennaPolarization` — Polarization of transmitter and receiver antennas

`'horizontal'` (default) | `'vertical'`

Polarization of transmitter and receiver antennas, specified as `'horizontal'` or `'vertical'`. Both antennas are assumed to have the same polarization. This value is used to calculate path loss due to ground reflection.

Dependencies

To specify `'AntennaPolarization'`, you must specify `'tirem'` propagation model.

Data Types: `char` | `string`

`GroundConductivity` — Conductivity of ground

`0.005` (default) | numeric scalar

Conductivity of the ground, specified as a numeric scalar in Siemens per meter (S/m) in the range of 0.0005 to 100. This value is used to calculate path loss due to ground reflection. The default value corresponds to average ground.

Dependencies

To specify `'GroundConductivity'`, you must specify `'tirem'` propagation model.

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 to 100. Relative permittivity is expressed as a 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 average ground.

Dependencies

To specify `'GroundPermittivity'`, you must specify `'tirem'` propagation model.

Data Types: `double`

`AtmosphericRefractivity` — Atmospheric refractivity near ground

`301` (default) | 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 refraction through the atmosphere and tropospheric scatter. The default value corresponds to average atmospheric conditions.

Dependencies

To specify `'AtmosphericRefractivity'`, you must specify `'tirem'` propagation model.

Data Types: `double`

`Humidity` — Absolute air humidity near ground

`'9'` (default) | numeric scalar

Absolute air humidity near ground,specified as a numeric scalar in `g/m^3` units in the range of 0 to 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.

Dependencies

To specify `'Humidity'`, you must specify `'tirem'` propagation model.

Data Types: `double`

raytracing-image-method

`MaxNumReflections` — Maximum number of path reflections

`1` (default) | `0` | `2`

Maximum number of reflections in the propagation paths to search for using ray tracing, specified as `0`, `1`, or `2`. The default value results in a search for a line-of-sight propagation path along with propagation paths that each contain a single reflection.

Dependencies

To specify `'MaxNumReflections'`, you must specify `'raytracing-image-method'` propagation model.

Data Types: `double`

`BuildingsMaterial` — Surface material of buildings

`'concrete'` (default) | `'perfect-reflector'` | `'brick'` | `'wood'` | `'glass'` | `'metal'` | `'custom'`

Surface material of buildings, specified as one of the following: `'perfect-reflector'`, `'concrete'`, `'brick'`, `'wood'`, `'glass'`, `'metal'`, or `'custom'`. The material type is used to calculate reflection loss where propagation paths reflect off of building surfaces. When

'BuildingsMaterial' is set to 'custom', the material permittivity and conductivity are specified in the BuildingsMaterialPermittivity and BuildingsMaterialConductivity properties.

Dependencies

To specify 'BuildingsMaterial', you must specify 'raytracing-image-method' propagation model.

Data Types: char | string

BuildingsMaterialPermittivity — Relative permittivity of buildings surface materials

5.31 (default) | non-negative numeric scalar

Relative permittivity of the buildings surface material, specified as a non-negative numeric scalar. Relative permittivity is expressed as a ratio of absolute material permittivity to the permittivity of vacuum. This value is used to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To specify 'BuildingsMaterialPermittivity', you must specify 'raytracing-image-method' propagation model and set 'BuildingsMaterial' to 'custom'.

Data Types: double

BuildingsMaterialConductivity — Conductivity of buildings surface materials

0.0548 (default) | non-negative numeric scalar

Conductivity of the buildings surface material, specified as a non-negative numeric scalar in Siemens per meter (S/m). This value is used to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To specify ' BuildingsMaterialConductivity ', you must specify 'raytracing-image-method' propagation model and set 'BuildingsMaterial' to 'custom'.

Data Types: double

TerrainMaterial — Surface material of terrain

'concrete' (default) | 'perfect-reflector' | 'brick' | 'water' | 'vegetation' | 'loam' | 'custom'

Surface material of terrain, specified as one of the following: 'perfect-reflector', 'concrete', 'brick', 'water', 'vegetation', 'loam', or 'custom'. The material type is used to calculate reflection loss where propagation paths reflect off of terrain surfaces. When 'TerrainMaterial' is set to 'custom', the material permittivity and conductivity are specified in the 'TerrainMaterialPermittivity' and 'TerrainMaterialConductivity' properties.

Dependencies

To specify 'TerrainMaterial', you must specify 'raytracing-image-method' propagation model.

Data Types: char | string

TerrainMaterialPermittivity — Relative permittivity of terrain materials

5.31 (default) | non-negative numeric scalar

Relative permittivity of the terrain material, specified as a non-negative numeric scalar. Relative permittivity is expressed as a ratio of absolute material permittivity to the permittivity of vacuum. This value is used to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To specify `'TerrainMaterialPermittivity'`, you must specify `'raytracing-image-method'` propagation model and set `'TerrainMaterial'` to `'custom'`.

Data Types: `double`

`TerrainMaterialConductivity` — Conductivity of terrain materials

`0.0548` (default) | non-negative numeric scalar

Conductivity of the terrain material, specified as a non-negative numeric scalar in Siemens per meter (S/m). This value is used to calculate path loss due to reflection. The default value corresponds to concrete at 1.9 GHz.

Dependencies

To specify `'TerrainMaterialConductivity '`, you must specify `'raytracing-image-method'` propagation model and set `'TerrainMaterial'` to `'custom'`.

Data Types: `double`

More About

Propagation models

Basic propagation models predict path loss as a function of distance between sites and assume line-of-sight (LOS) conditions, disregarding the curvature of the Earth, terrain, or other obstacles. Urban propagation models also predict path loss as a function of distance but use empirical models that are derived from measurements in non-line-of-sight (NLOS) conditions. Terrain propagation models predict path loss as a function of the terrain elevation profile between sites including buildings, which may be used to compute whether LOS or NLOS conditions apply.

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 one. To make the calculation more convenient, you can use N units which are given by the formula:$N = (n - 1) \times 10^6$

References

[1] Sun, S.,Rapport, T.S., Thomas, T., Ghosh, A., Nguyen, H., Kovacs, I., Rodriguez, I., Koymen, O.,and Prartyka, A. "Investigation of prediction accuracy, sensitivity, and parameter stability of large-scale propagation path loss models for 5G wireless communications." *IEEE Transactions on Vehicular Technology*, Vol.65, No 5, pp 2843-2860, May 2016.

[2] ITU-R P.840-6. "Attenuation due to cloud and fog." *Radiocommunication Sector of ITU*

[3] ITU-R P.838-3. "Specific attenuation model for rain for use in prediction methods." *Radiocommunication Sector of ITU*

[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*. Pg-7.

[5] *SoftWright Homepage* https://www.softwright.com/faq/support/longley_rice_variability.html

[6] Seybold, John. *Introduction to RF Propagation*. Wiley, 2005

[7] ITU-R P.676-11. "Attenuation by atmospheric gases." *Radiocommunication Sector of ITU*

See Also

`coverage` | `link` | `los` | `pathloss` | `range` | `sigstrength` | `sinr` | `tiremSetup` | `tirempl`

Topics

“Access TIREM Software”

Introduced in R2017b

range

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

character vector or string

Propagation model, specified as a character vector or string.

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`

`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 specified `pl`.

See Also

`pathloss` | `propagationModel`

Introduced in R2017b

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

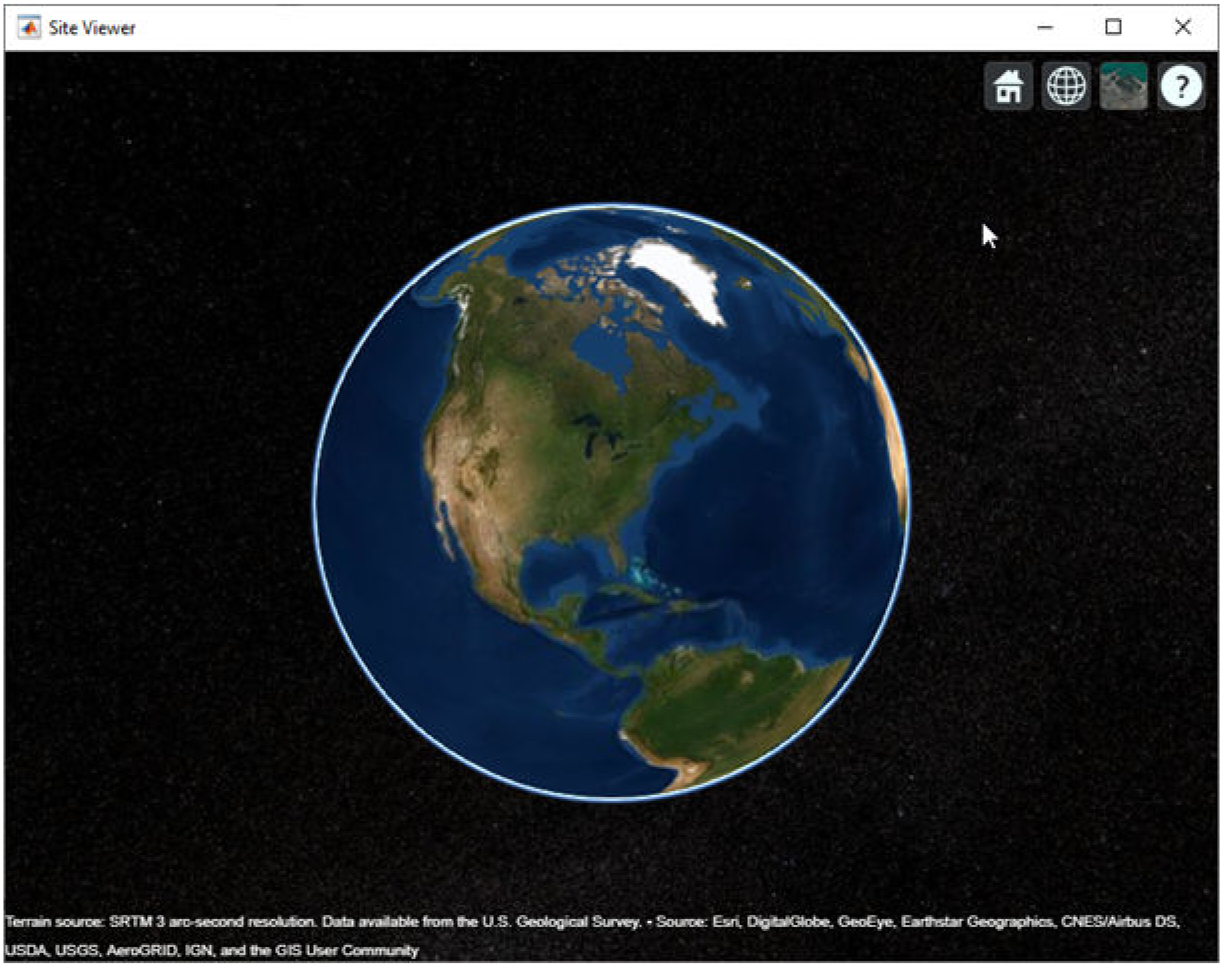
Site Viewer Maps Using Custom Terrain

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.

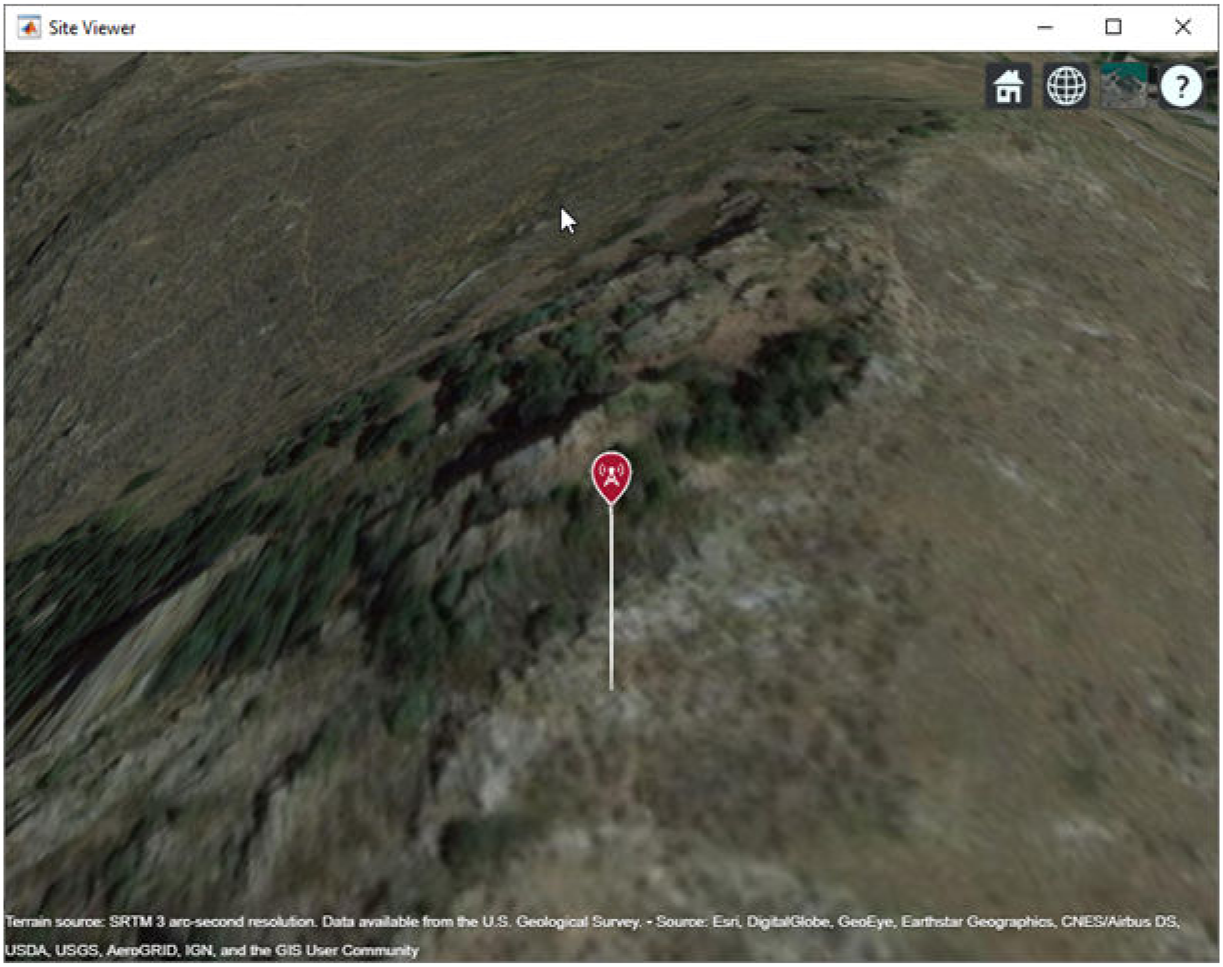
```
viewer = siteviewer("Terrain","southboulder");
```



Terrain source: SRTM 3 arc-second resolution. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

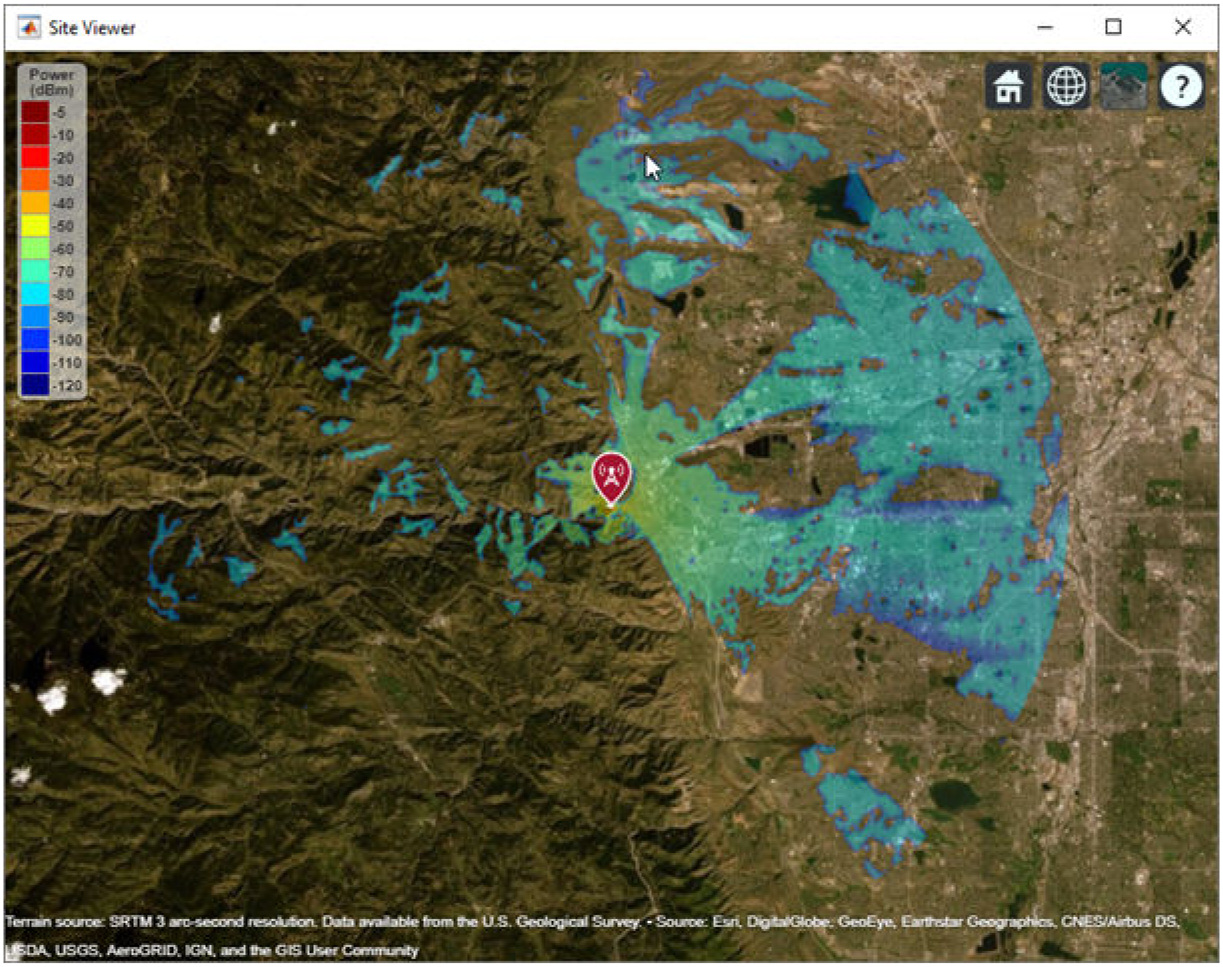
Create a site with the terrain region.

```
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, DigitalGlobe, 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, DigitalGlobe, 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`

See Also

`addCustomTerrain` | `siteviewer`

Introduced in R2019a

pattern

Plot antenna radiation pattern on map

Syntax

```
pattern(tx)
pattern(rx,frequency)
pattern( ___ ,Name,Value)
```

Description

`pattern(tx)` plots the 3-D antenna radiation pattern for the transmitter site, `txsite`. Signal gain value (dBi) in a particular direction determines the color of the pattern.

`pattern(rx,frequency)` plots the 3-D radiation pattern for the receiver site, `rxsite` for the specified `frequency`.

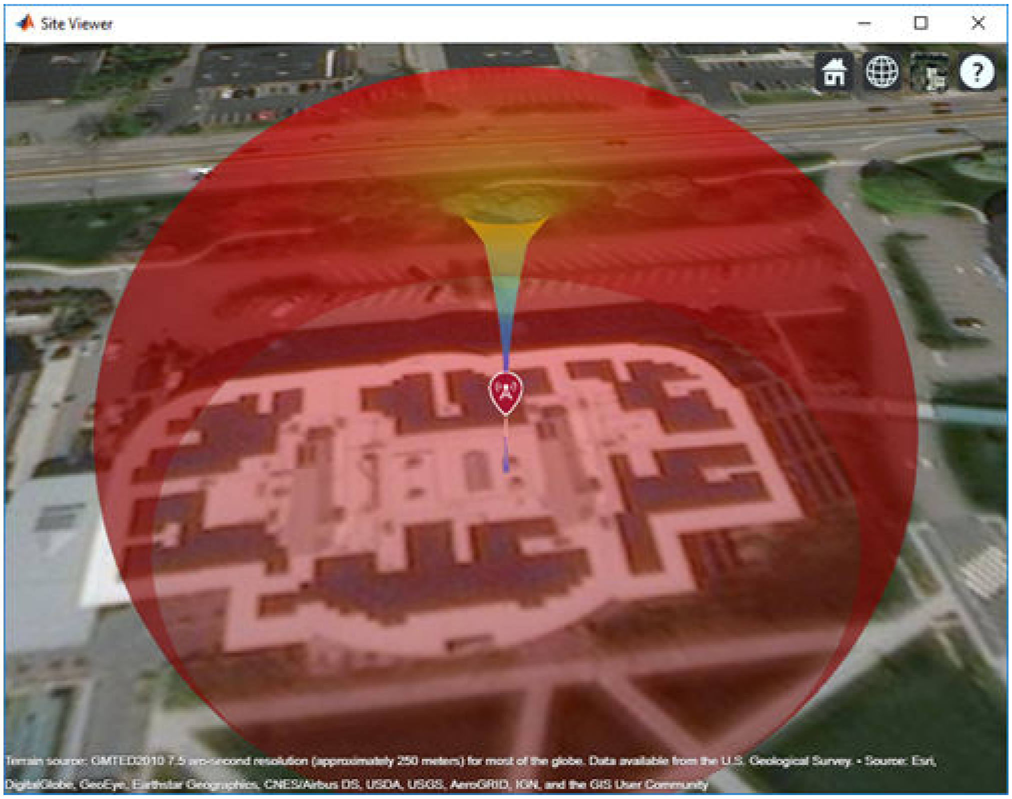
`pattern( ___ ,Name,Value)` plots the 3-D radiation pattern with additional options specified by name-value pair arguments.

Examples

Single Transmitter Site Pattern

Define and visualize the radiation pattern of a single transmitter site.

```
tx = txsite;
pattern(tx)
```



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

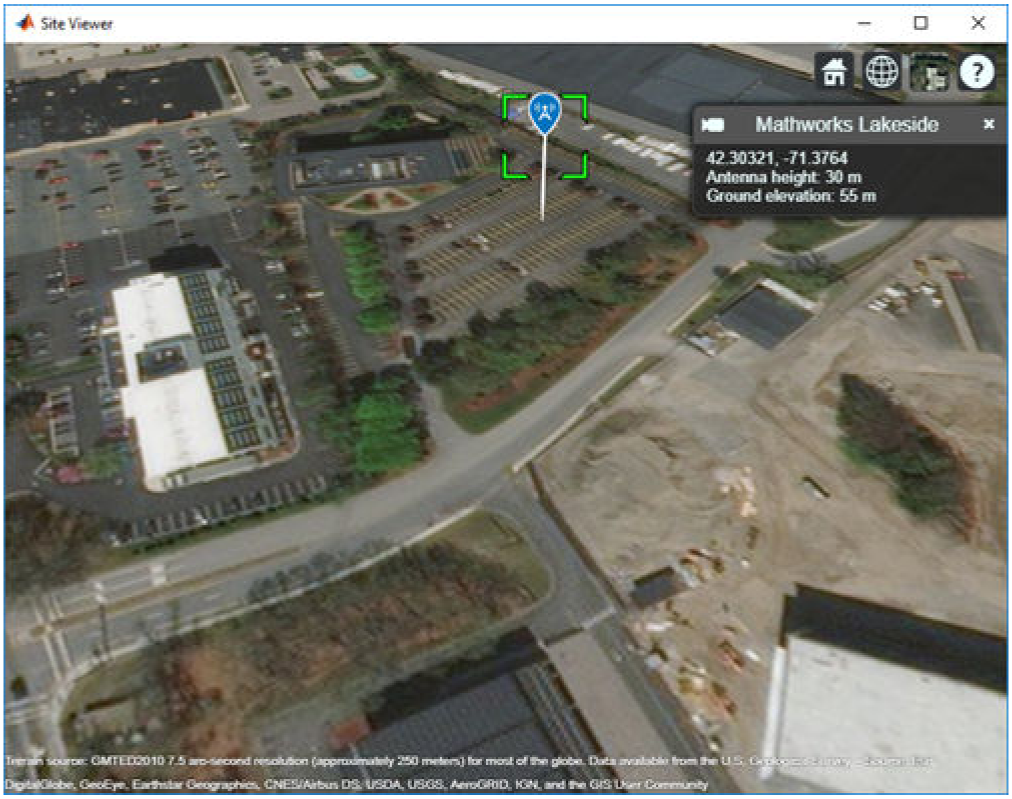
Single Receiver Site Pattern

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)
```

```
rx = 
  rxsite with properties:

                   Name: 'Mathworks Lakeside'
               Latitude: 42.3032
              Longitude: -71.3764
                Antenna: [1×1 dipole]
           AntennaAngle: 0
          AntennaHeight: 30
             SystemLoss: 0
    ReceiverSensitivity: -100
```

```
show(rx)
```



Mathworks Lakeside

42.30321, -71.3764

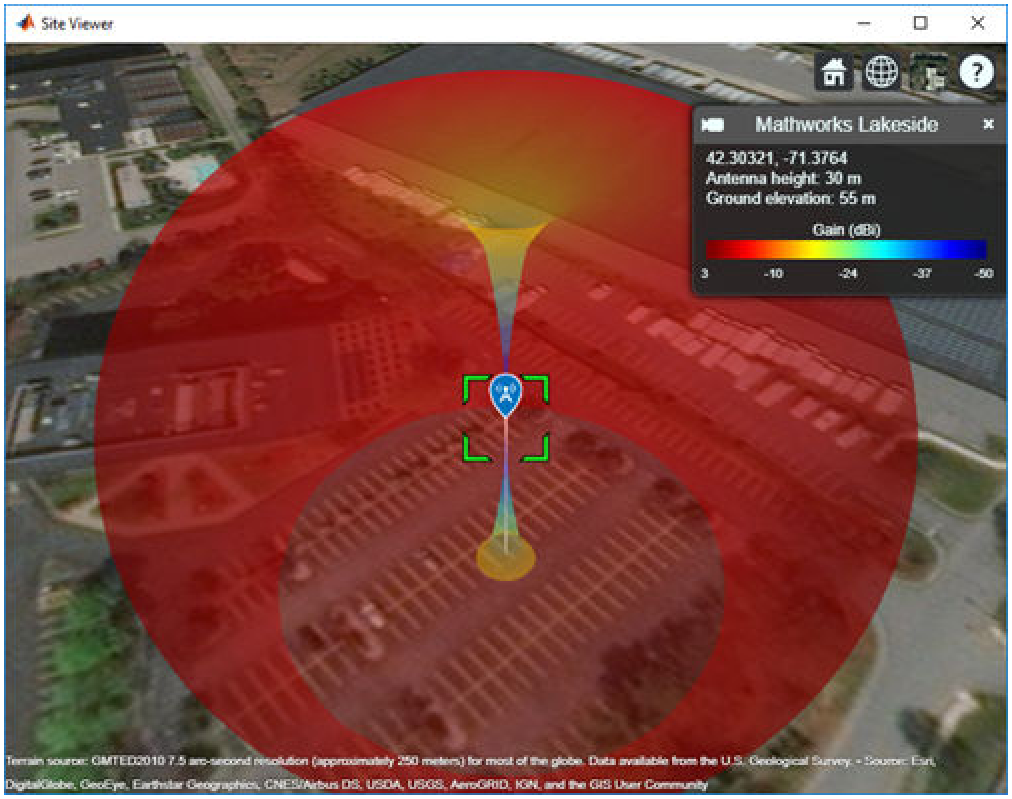
Antenna height: 30 m

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. Sources: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Visualize the pattern of the receiver site at 75 MHz.

```
pattern(rx,75e6)
```



42.30321, -71.3764
Antenna height: 30 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

Pattern for Directional Transmitter and Receiver

Create 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.

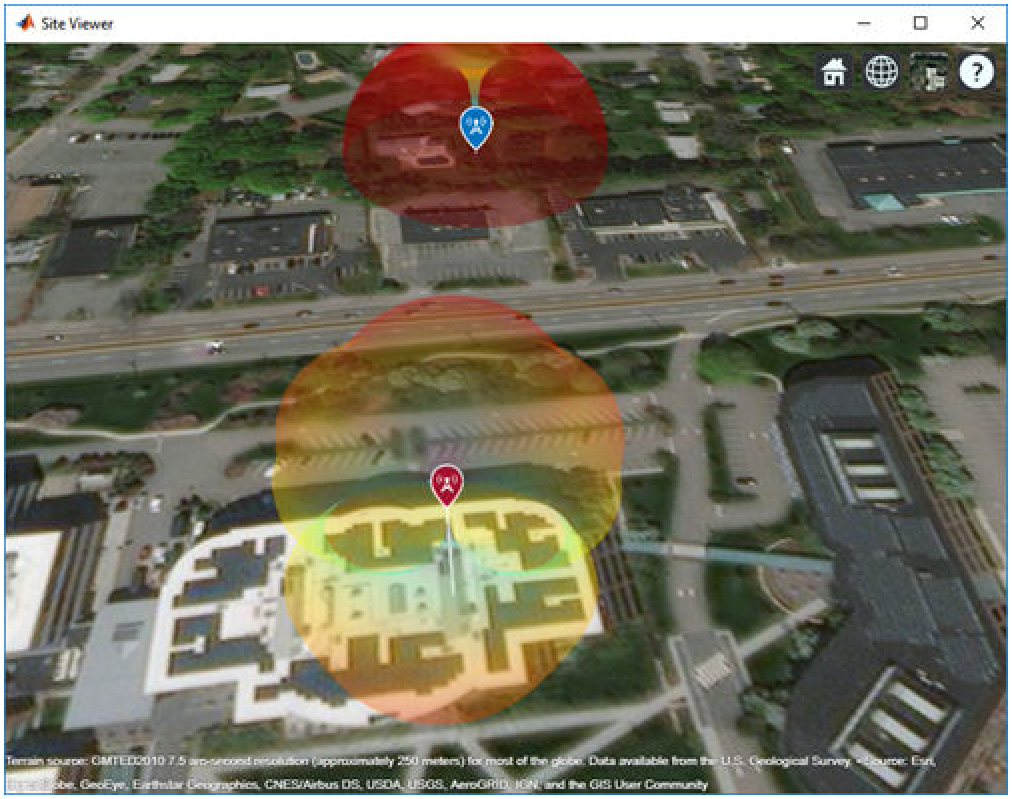
```
fq = 4.5e9;
tx = txsite('Name','MathWorks','Latitude',42.3001,'Longitude', -71.3503, ...
      'Antenna', yagiAntenna,'AntennaAngle', 90,'AntennaHeight', 30, ...
      'TransmitterFrequency', fq,'TransmitterPower', 10);
rx = rxsite('Antenna',design(dipole, fq));
```

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.

```
pattern(tx,'Transparency',0.2);
pattern(rx, fq);
```


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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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

Example: 'Size',2

`Size` — Size of pattern plot

`50` (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.

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`

`Map` — Map for visualization or surface data

`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

Data Types: `char` | `string`

See Also

`coverage`

Introduced in R2018b

show

Show site location on map

Syntax

```
show(site)
show(site,Name,Value)
```

Description

`show(site)` displays the location of transmitter or receiver site on a map using a marker.

`show(site,Name,Value)` uses `icon` displays the site map using additional options specified by the `Name,Value` pairs.

Examples

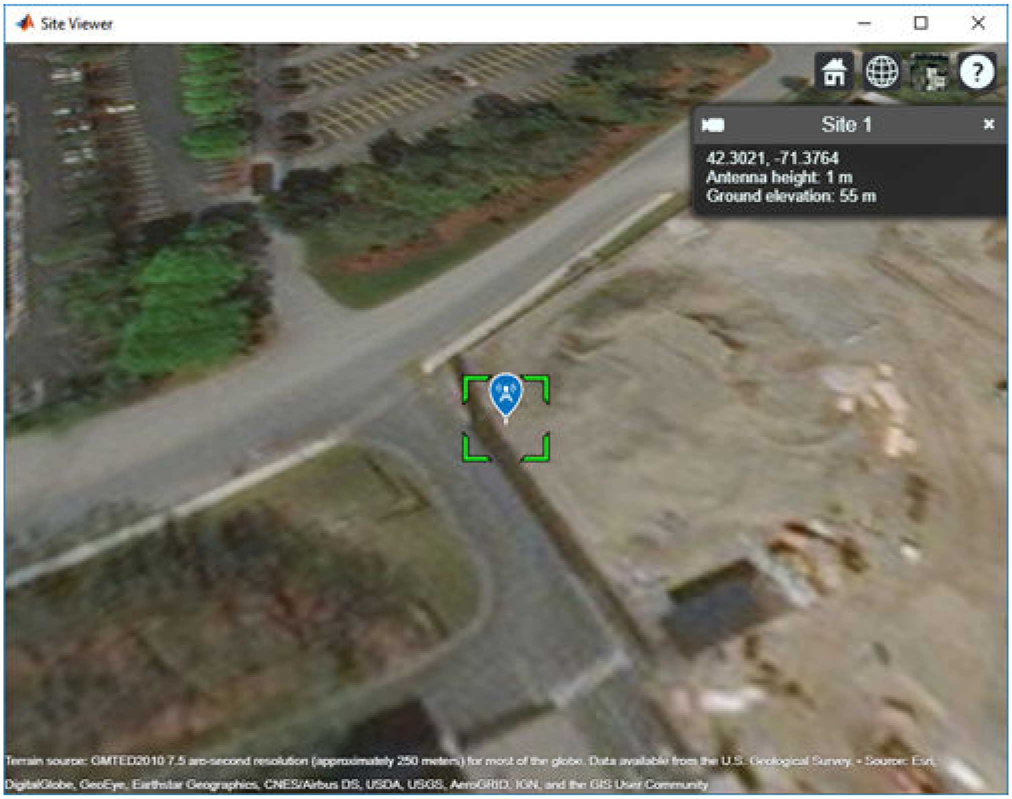
Default Receiver Site

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

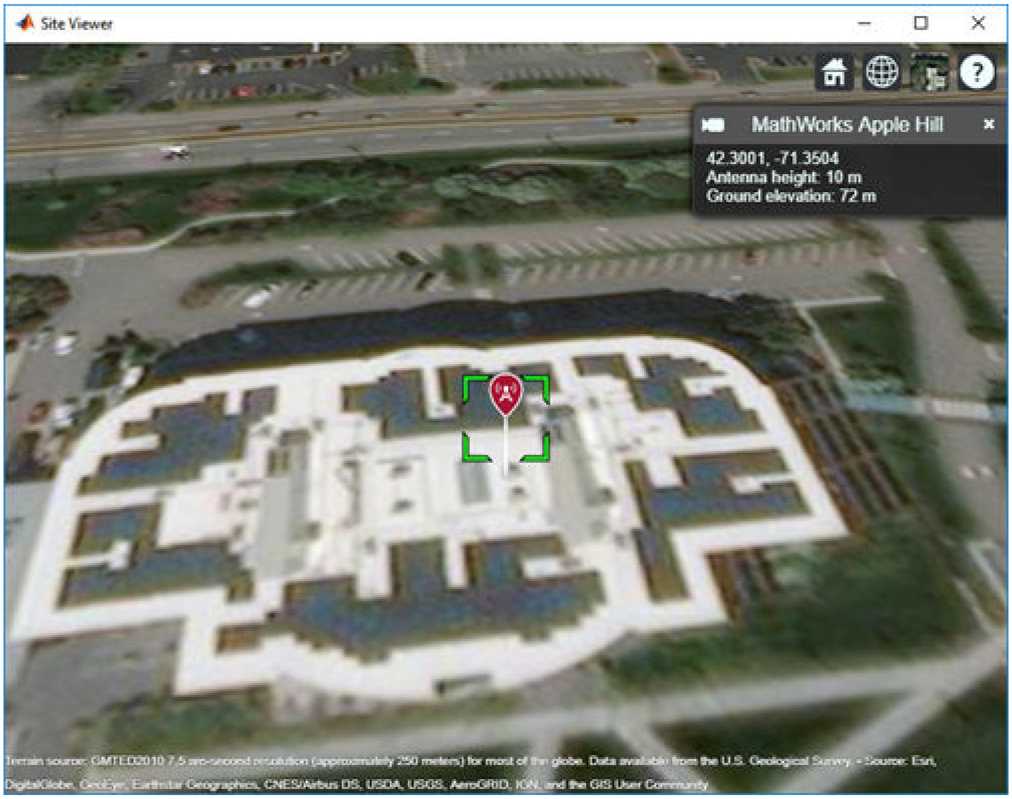
Show and Hide Transmitter Site

Create a transmitter site.

```
tx = txsite('Name','MathWorks Apple Hill',...
       'Latitude',42.3001, ...
       'Longitude',-71.3504);
```

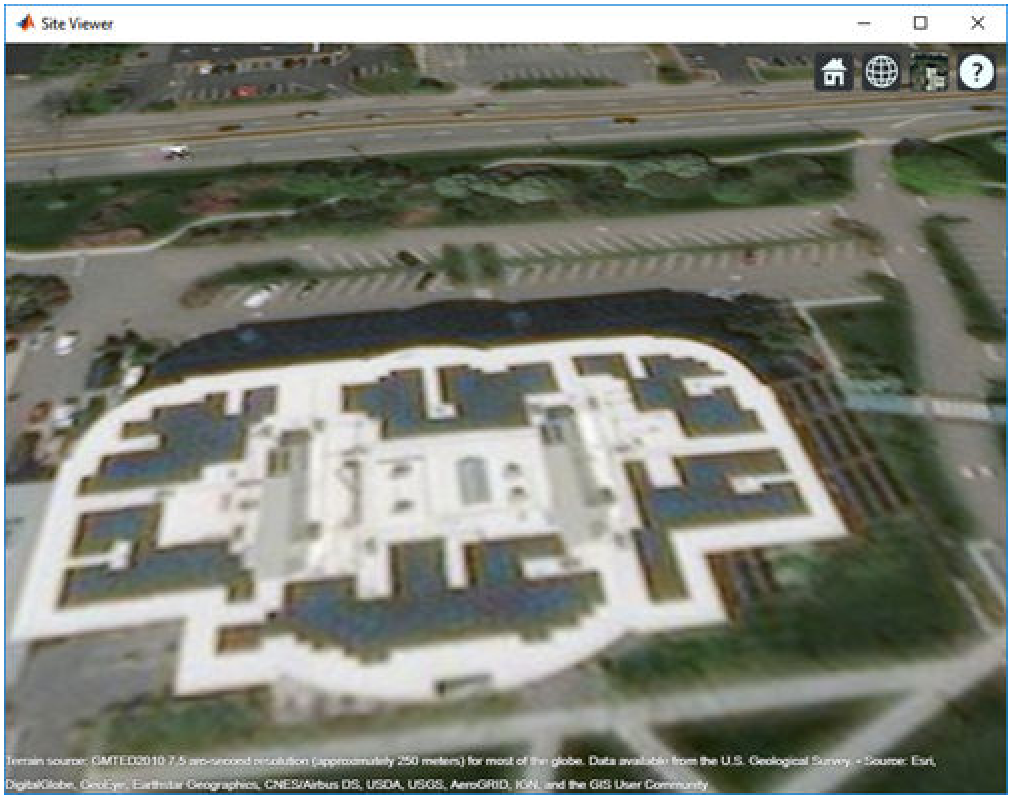
Show the transmitter site.

```
show(tx)
```


Site Viewer
MathWorks Apple Hill
42.3001, -71.3504
Antenna height: 10 m
Ground elevation: 72 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

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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

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

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 or surface data
`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

Data Types: `char` | `string`

See Also

`hide`

Introduced in R2017b

sigstrength

Signal strength due to transmitter

Syntax

```
ss = sigstrength(rx,tx)
ss = sigstrength(rx,tx,propmodel)
ss = sigstrength( ___ ,Name,Value)
```

Description

`ss = sigstrength(rx,tx)` returns the signal strength 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 propagation model is same as specifying the `'PropagationModel'` name-value pair.

`ss = sigstrength( ___ ,Name,Value)` returns the signal strength using additional options specified by `Name,Value` pairs and either of the previous syntaxes.

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
```

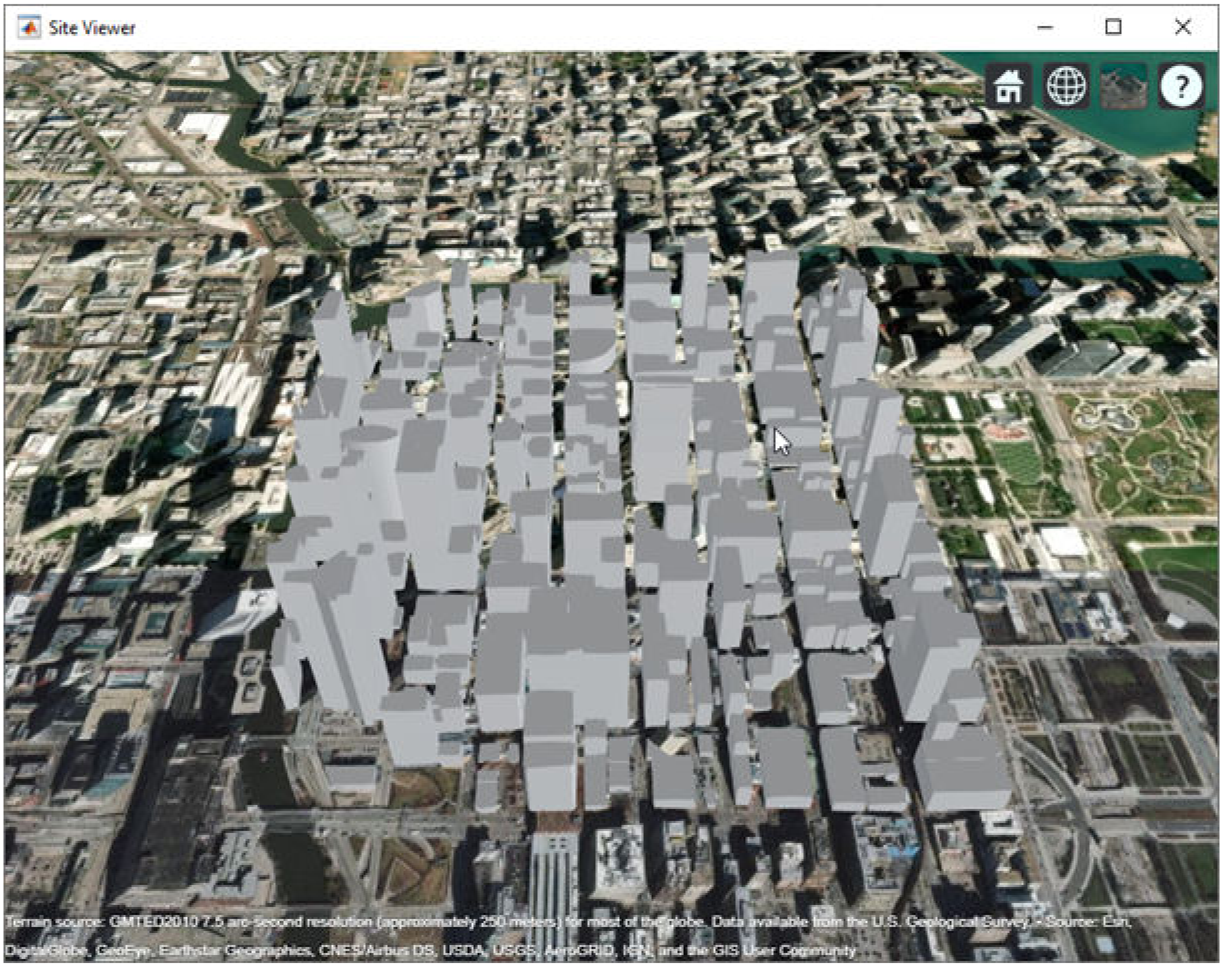
```
margin = abs(rx.ReceiverSensitivity - ss)
```

```
margin = 18.8586
```

Signal Strength Using Ray Tracing Image Method Propagation Model

Launch Site Viewer with buildings in Chicago.

```
viewer = siteviewer("Buildings","chicago.osm");
```


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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create transmitter site on a building.

```
tx = txsite('Latitude',41.8800, ...
    'Longitude',-87.6295, ...
    'TransmitterFrequency',2.5e9);
```

Create receiver site near another building.

```
rx = rxsite('Latitude',41.881352, ...
    'Longitude',-87.629771, ...
    'AntennaHeight',30);
```

Compute signal strength using ray tracing propagation model and default single-reflection analysis.

```
pm = propagationModel("raytracing-image-method");
ssOneReflection = sigstrength(rx,tx,pm)
```

```
ssOneReflection = -55.2839
```

Compute signal strength with analysis up to two reflections, where total received power is the cumulative power of all propagation paths

```
pm.MaxNumReflections = 2;
ssTwoReflections = sigstrength(rx,tx,pm)
```

```
ssTwoReflections = -53.1827
```

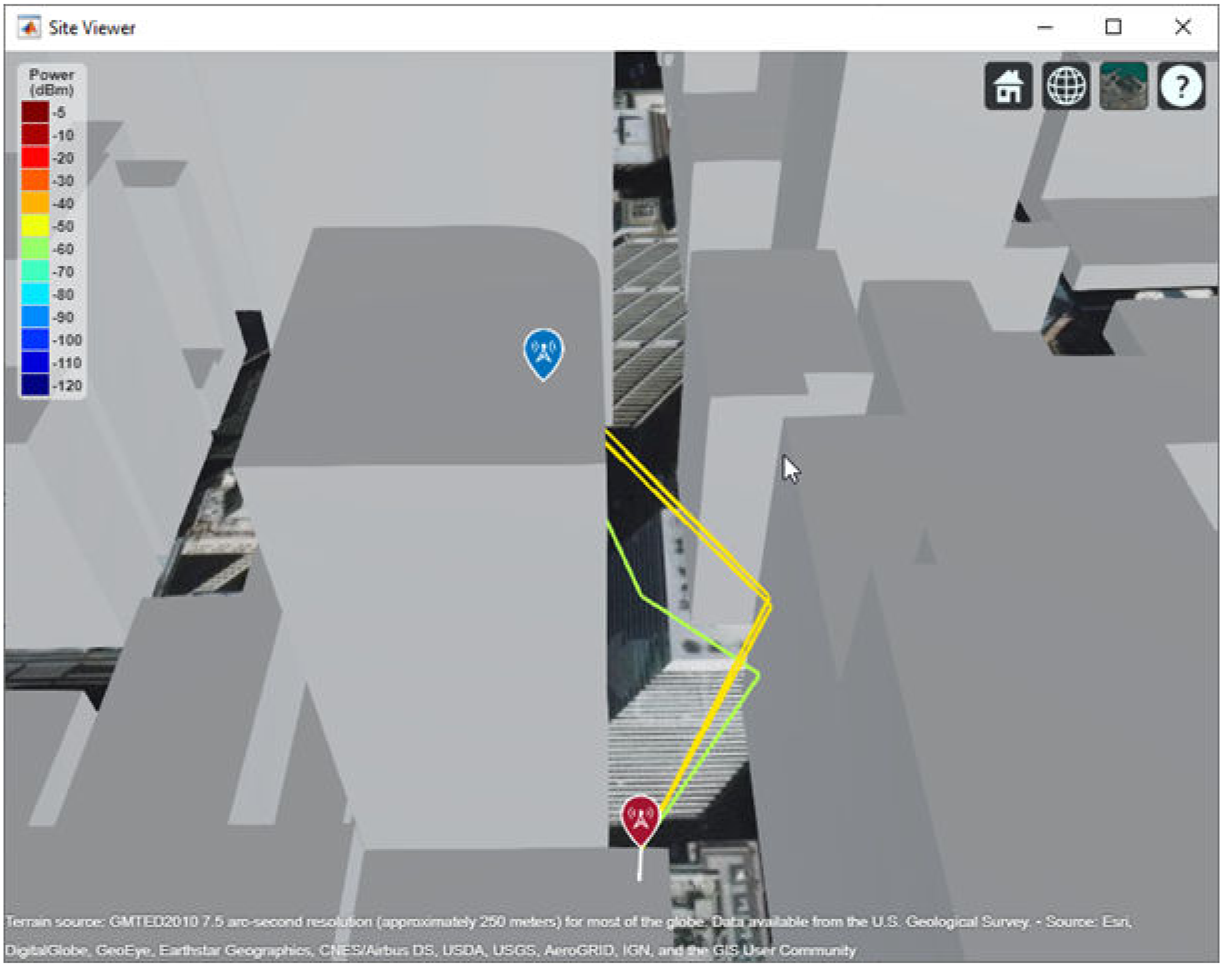
Observe effect of material by replacing default concrete material with perfect reflector.

```
pm.BuildingsMaterial = 'perfect-reflector';
ssPerfect = sigstrength(rx,tx,pm)
```

```
ssPerfect = -42.0872
```

Plot propagation paths.

```
raytrace(tx, rx, pm)
```


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

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
character vector | string

Propagation model, specified as a character vector or string. You can also use the name-value pair `'PropagationModel'` to specify this parameter. You can also use the `propagationModel` function to define this input.

Data Types: `char` | `string`

Name-Value Pair Arguments

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

Example: `'Type','power'`

Type — Type of signal strength to compute
`'power'` (default) | `'efield'`

Type of signal strength to compute, specified as the comma-separated pair consisting of `'Type` and `'power'` or `'efield'`.

When type is `'power'`, signal strength is expressed in power units (dBm) of the signal at the mobile receiver input. When type is `'efield'`, signal strength is expressed 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
`'longley-rice'` (default) | `'freespace'` | `'close-in'` | `'rain'` | `'gas'` | `'fog'` | `'raytracing-image-method'` | propagation model object

Propagation model to use for the path loss calculations, specified as the comma-separated pair consisting of `'PropagationModel'` and `'freespace'`, `'close-in'`, `'rain'`, `'gas'`, `'fog'`, `'longley-rice'`, `'raytracing-image-method'`, or as an object created using the `propagationModel` function. The default propagation model is `'longeley-rice'` when terrain is enabled and `'freespace'` when terrain is disabled.

Data Types: `char`

Map — Map for visualization or surface data
`siteviewer` object | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

Data Types: `char` | `string`

Output Arguments

`ss` — Signal strength
M-by-*N* array

Signal strength, returned as *M*-by-*N* array in dBm. *M* is the number of TX sites and *N* is the number of RX sites.

See Also

`link` | `propagationModel` | `sinr`

Introduced in R2017b

sinr

Display signal-to-interference-plus-noise ratio (SINR) map

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`. Each colored contour of the map defines the areas where the corresponding SINR is available to a mobile receiver. For each location, the signal source is the transmitter site in `txs` with the greatest signal strength. The remaining transmitter sites in `txs` act as interference. If `txs` is scalar, or there are no sources of interference, the resultant map displays signal-to-noise ratio (SNR).

`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.

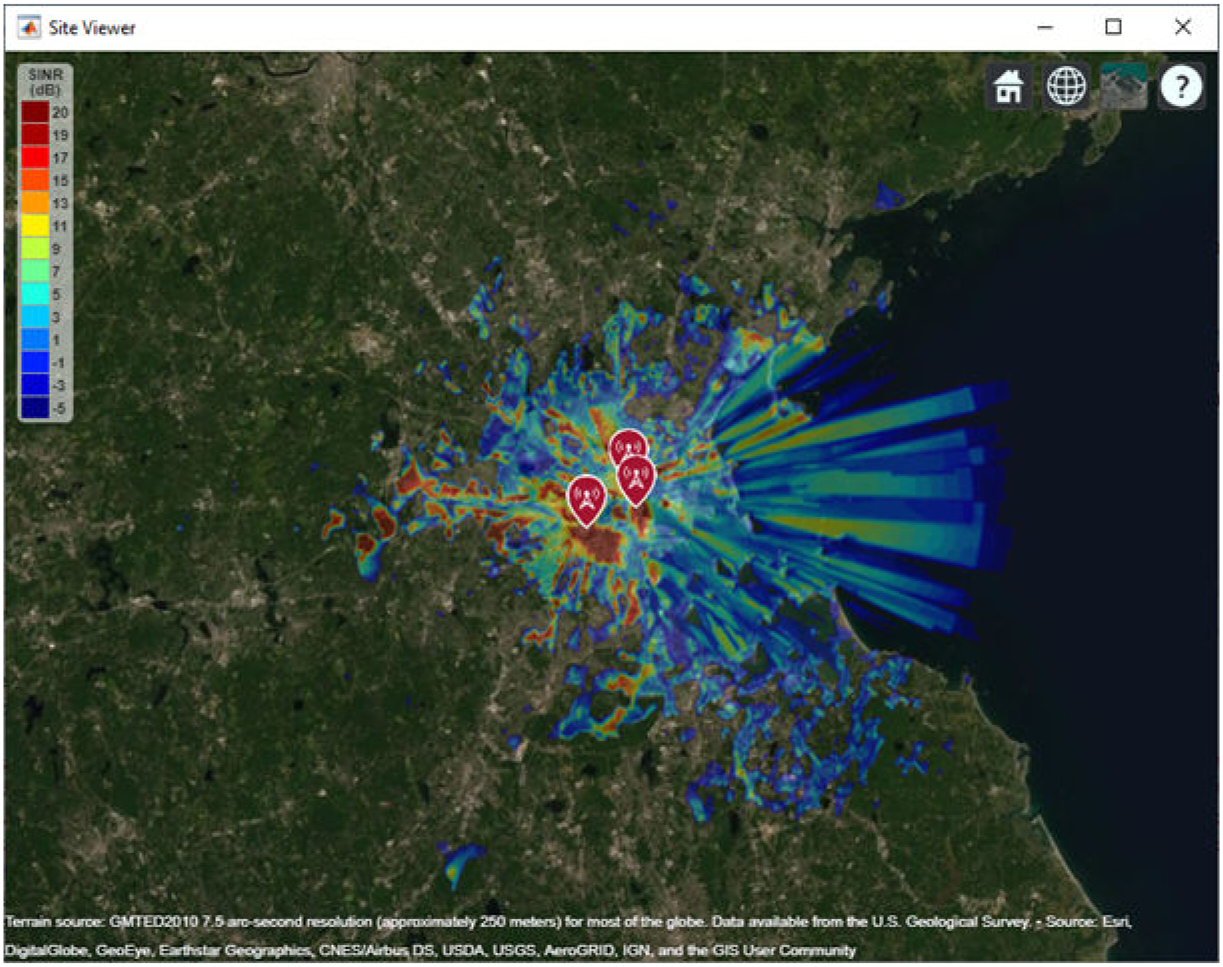
```
names = ["Fenway Park","Faneuil Hall","Bunker Hill Monument"];
lats = [42.3467,42.3598,42.3763];
lons = [-71.0972,-71.0545,-71.0611];
```

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, DigitalGlobe, 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.

rxs — Receiver sites
rxsite object | array of rxsite objects

Receiver site, specified as a `rxsite` object. Use array inputs to specify multiple sites.

propmodel — Propagation model
character vector | string

Propagation model, specified as a character vector or string. You can use the `propagationModel` function to define this input.

You can also use the name-value pair 'PropagationModel' to specify this parameter.

Name-Value Pair Arguments

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

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
'longley-rice' (default) | 'freespace' | 'close-in' | 'rain' | 'gas' | 'fog' | 'raytracing-image-method' | propagation model object

Propagation model to use for the path loss calculations, specified as the comma-separated pair consisting of 'PropagationModel' and 'freespace', 'close-in', 'rain', 'gas', 'fog', 'longley-rice', 'raytracing-image-method', or as an object created using the propagationModel function. The default propagation model is 'longeley-rice' when terrain is enabled and 'freespace' when terrain is disabled. If 'raytracing-image-method' is specified, the value of 'MaxNumReflections' property must be lesser than 1.

Data Types: char

ReceiverNoisePower — Total noise power at receiver
-107 (default) | scalar

Total noise power at receiver, specified as the comma-separated pair consisting of 'ReceiverNoisePower' and 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 the comma-separated pair consisting of 'ReceiverGain' and 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 the comma-separated pair consisting of `'ReceiverAntennaHeight'` and 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 | terrain name

Map for visualization or surface data, specified as the comma-separated pair consisting of `'Map` and a `siteviewer` object or a terrain name. A terrain name may be specified 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 Site Viewer. If no Site Viewer is open, the default value is a new Site Viewer or else `'gmted2010'` if the function is called with an output argument.

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 the comma-separated pair consisting of `'Values'` and 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 propagation model type as shown:

Propagation Model	MaxRange
Basic or urban	`30` km
Terrain	`30` km or distance to the furthest building.
Multipath	`500` m

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 the comma-separated pair consisting of `'Resolution'` and `'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 the comma-separated pair consisting of `'ColorMap'` and 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 the comma-separated pair consisting of `'ColorLimits'` and 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 the comma-separated pair consisting of `'ShowLegend'` and `'true'` or `'false'`.

`Transparency` — Transparency of SINR map

`0.4` (default) | numeric scalar

Transparency of SINR map, specified as the comma-separated pair consisting of `'Transparency'` and 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"`.

See Also

`coverage` | `propagationModel`

Introduced in R2018a

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

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.

See Also

`propagationModel` | `tiremSetup`

Topics

“Access TIREM Software”

Introduced in R2019a

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 7-147.

`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`

See Also

`propagationModel` | `tirempl`

Topics

"Access TIREM Software"

Introduced in R2019a

raytrace

Plot propagation paths between sites

Syntax

```
raytrace(tx,rx)
raytrace(tx,rx,propmodel)
raytrace( ___ ,Name,Value)
rays = raytrace( ___ )
```

Description

`raytrace(tx,rx)` plots the propagation paths from the transmitter site (`tx`) to the receiver site (`rx`). The propagation paths are found using ray tracing with the terrain and buildings data defined in the Site Viewer map. Each propagation path is color-coded according to the received power (dBm) or path loss (dB) along the path.

Note The ray tracing analysis includes surface reflections but does not include effects from refraction, diffraction, or scattering.

`raytrace(tx,rx,propmodel)` plots 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, please use the `'raytracing-image-method'` propagation model and set the properties to specify building materials.

`raytrace( ___ ,Name,Value)` plots propagation paths with additional options specified by one or more name-value pairs.

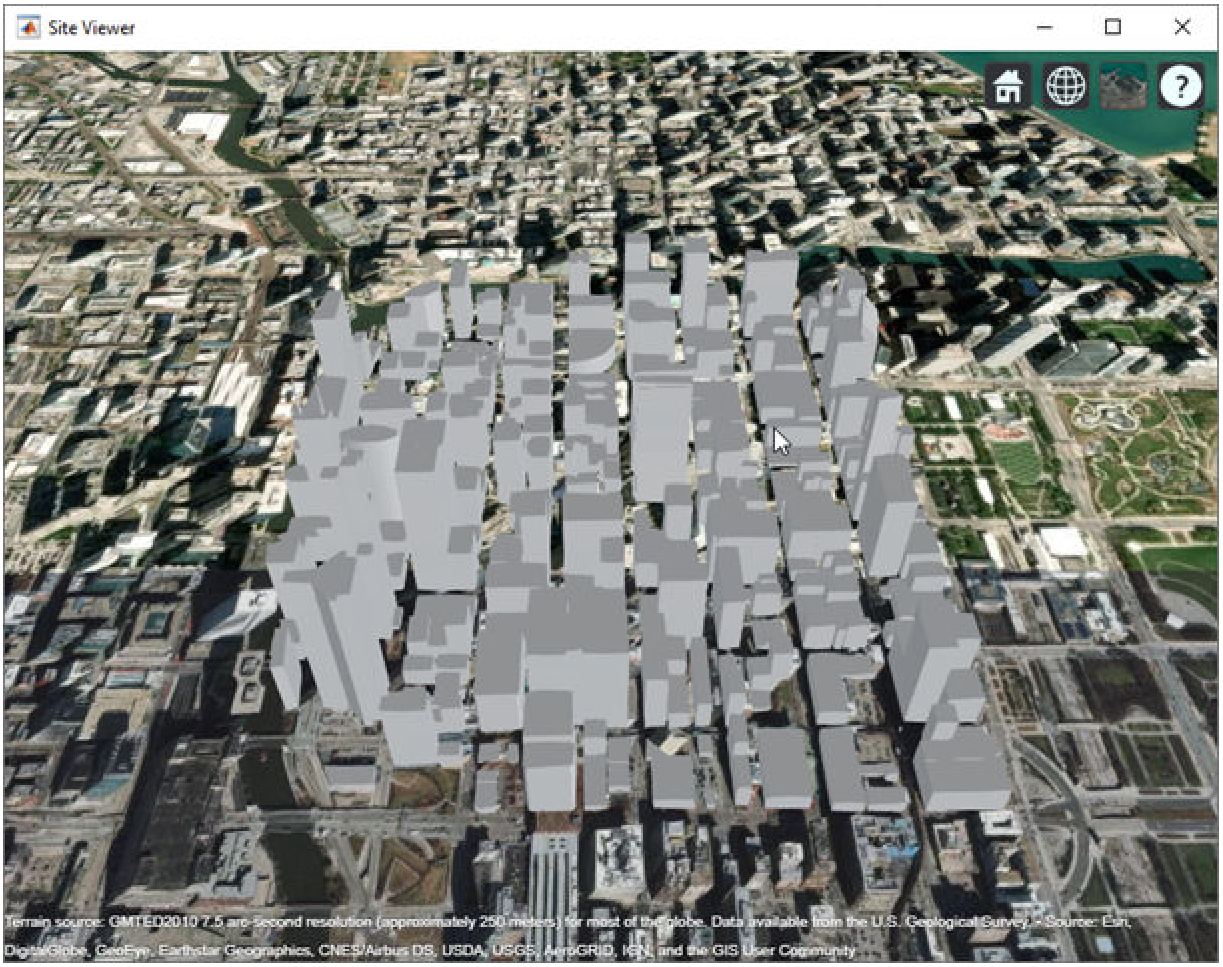
`rays = raytrace( ___ )` returns the propagation paths in `rays`.

Examples

Signal Strength Using Ray Tracing Image Method Propagation Model

Launch Site Viewer with buildings in Chicago.

```
viewer = siteviewer("Buildings","chicago.osm");
```


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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Create transmitter site on a building.

```
tx = txsite('Latitude',41.8800, ...
    'Longitude',-87.6295, ...
    'TransmitterFrequency',2.5e9);
```

Create receiver site near another building.

```
rx = rxsite('Latitude',41.881352, ...
    'Longitude',-87.629771, ...
    'AntennaHeight',30);
```

Compute signal strength using ray tracing propagation model and default single-reflection analysis.

```
pm = propagationModel("raytracing-image-method");
ssOneReflection = sigstrength(rx,tx,pm)
```

```
ssOneReflection = -55.2839
```

Compute signal strength with analysis up to two reflections, where total received power is the cumulative power of all propagation paths

```
pm.MaxNumReflections = 2;
ssTwoReflections = sigstrength(rx,tx,pm)
ssTwoReflections = -53.1827
```

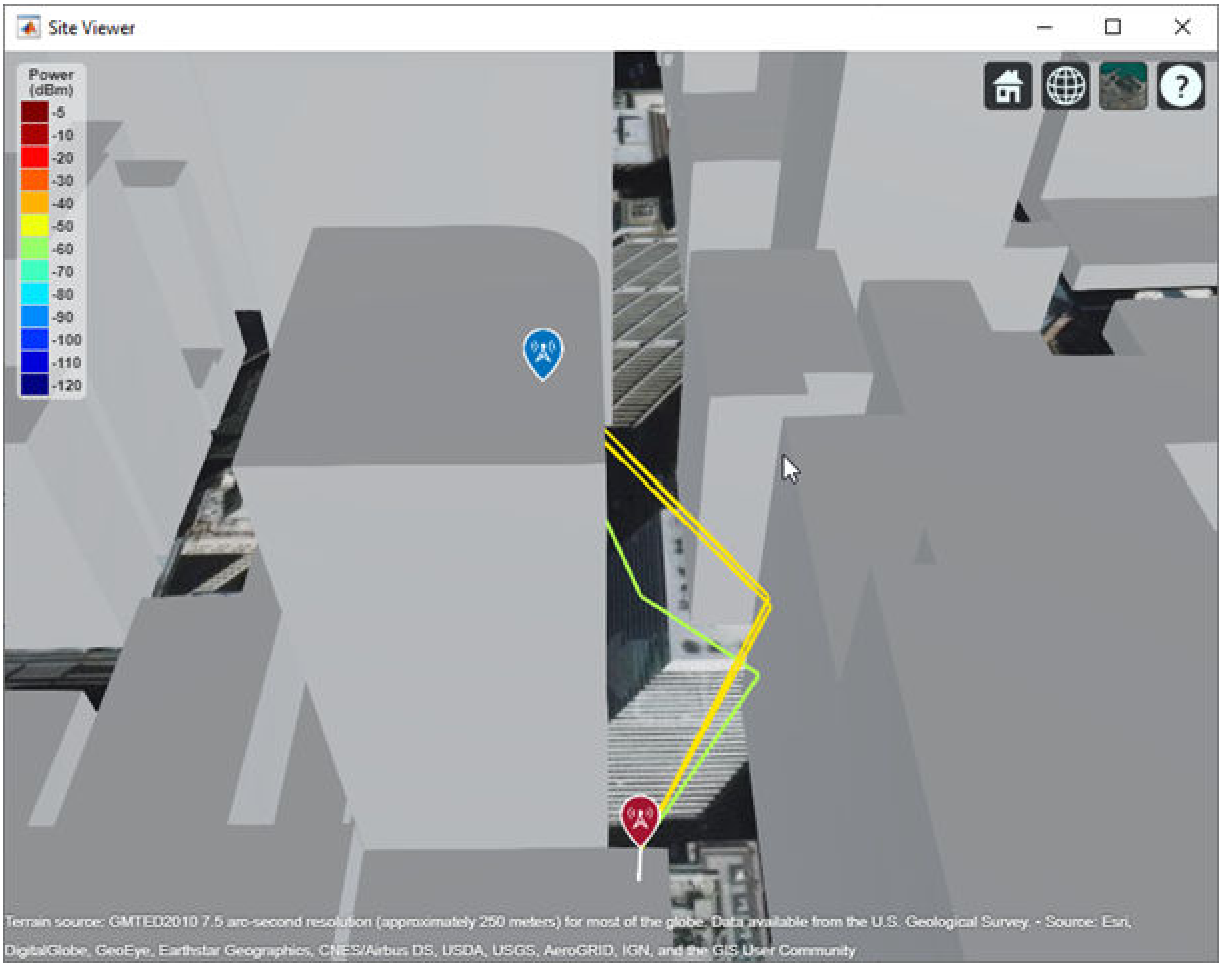
Observe effect of material by replacing default concrete material with perfect reflector.

```
pm.BuildingsMaterial = 'perfect-reflector';
ssPerfect = sigstrength(rx,tx,pm)
```

```
ssPerfect = -42.0872
```

Plot propagation paths.

```
raytrace(tx, rx, pm)
```

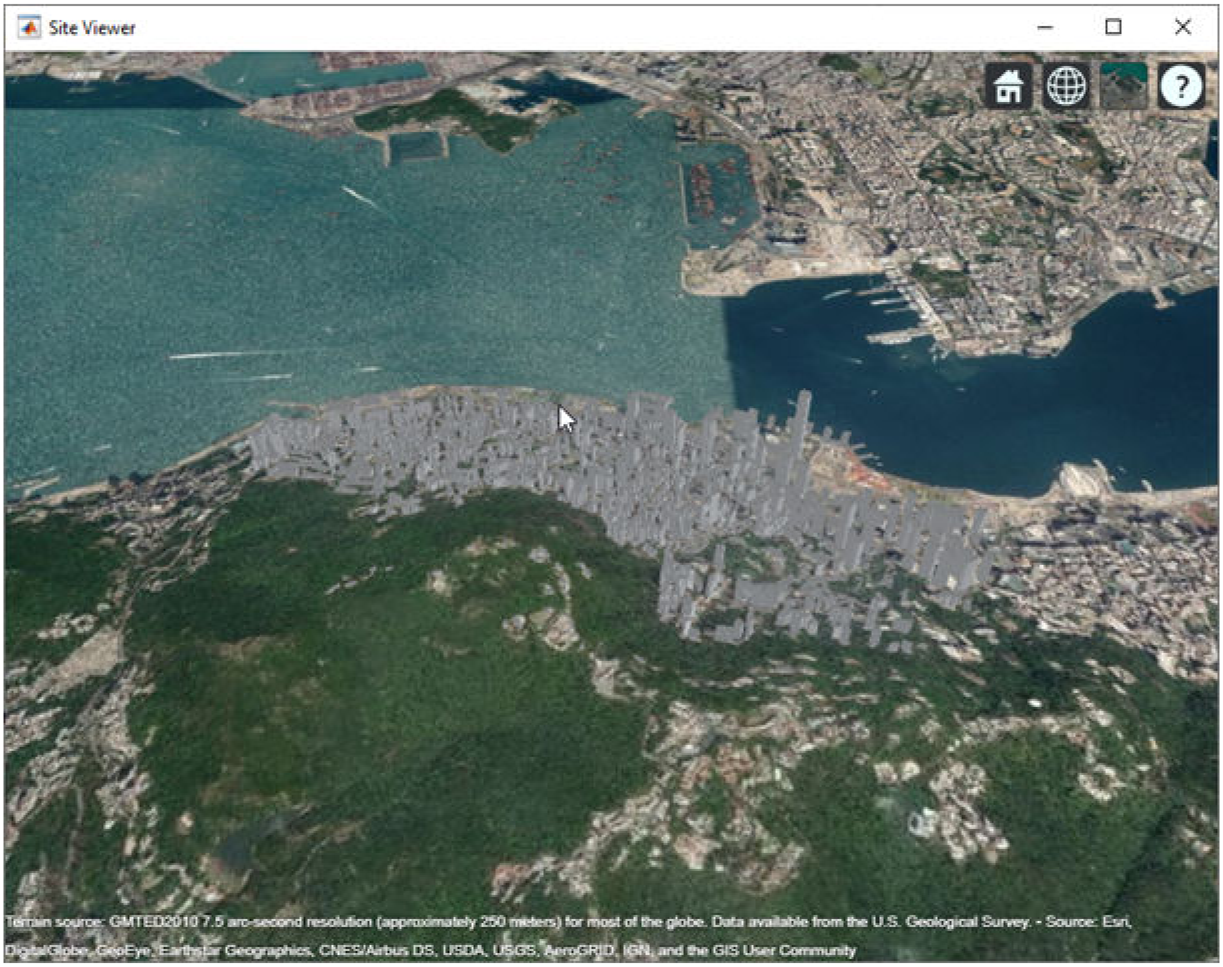


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. - Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Path Loss Due to Material Reflection and Atmosphere

Launch Site Viewer with buildings in Hong Kong.

```
viewer = siteviewer("Buildings","hongkong.osm");
```


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, DigitalGlobe, 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.

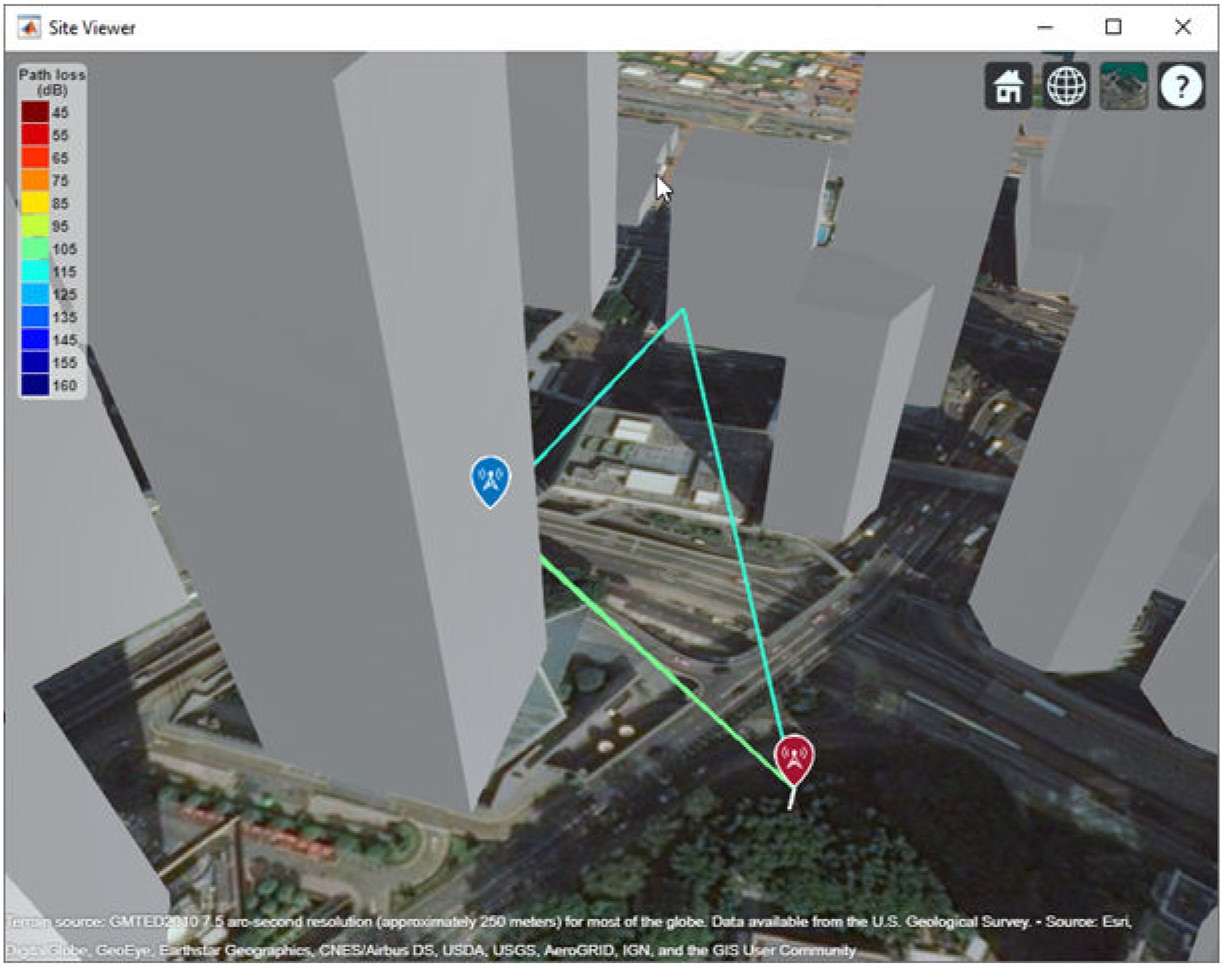
```
tx = txsite("Name","Small cell transmitter", ...
       "Latitude",22.2789, ...
       "Longitude",114.1625, ...
       "AntennaHeight",10, ...
       "TransmitterPower",5, ...
       "TransmitterFrequency",28e9);
rx = rxsite("Name","Small cell receiver", ...
       "Latitude",22.2799, ...
       "Longitude",114.1617, ...
       "AntennaHeight",1);
```

Create ray tracing propagation model for perfect reflection.

```
pm = propagationModel("raytracing-image-method", ...
       "BuildingsMaterial","perfect-reflector", ...
       "TerrainMaterial","perfect-reflector");
```

Visualize propagation paths and compute corresponding path losses.

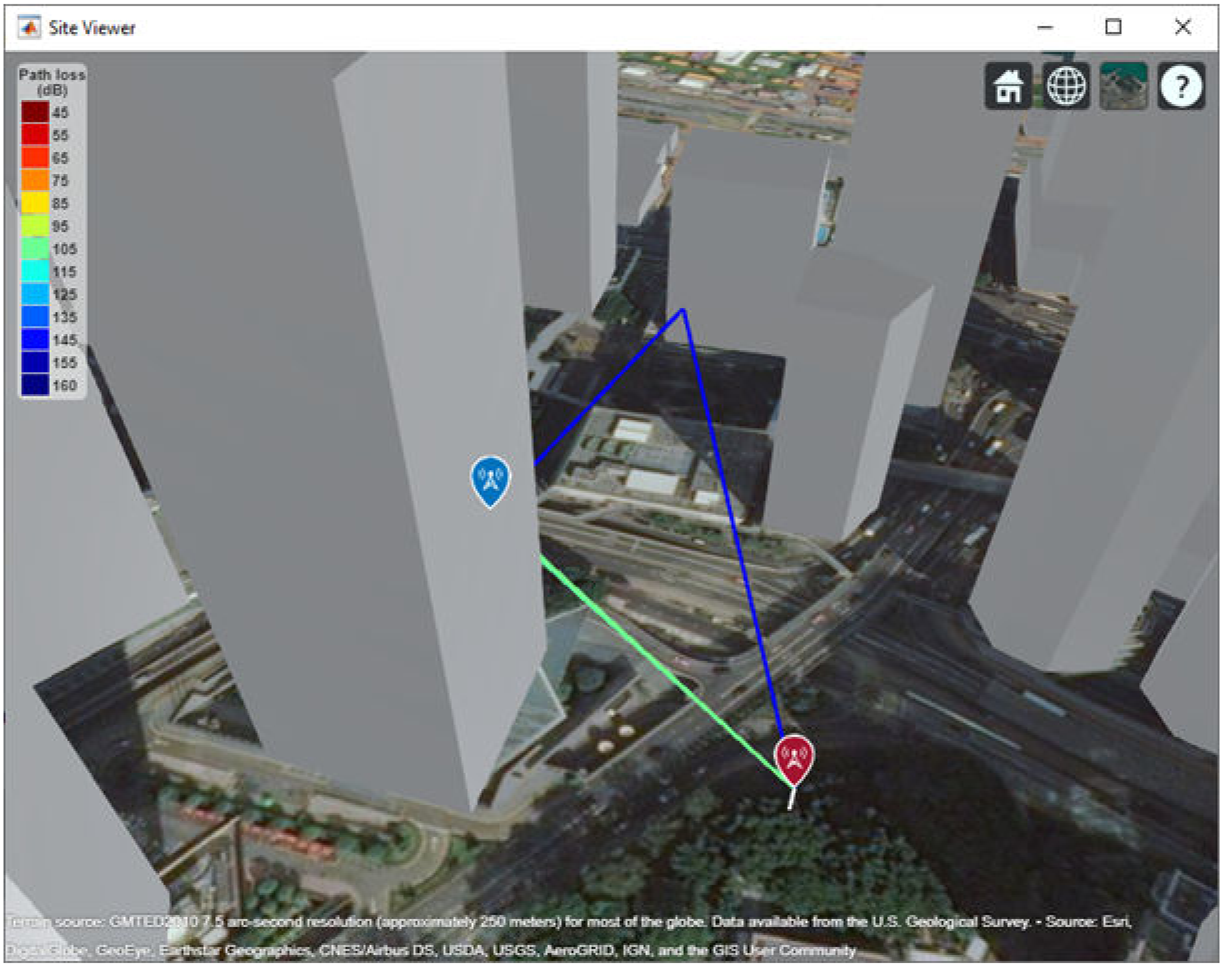
```
raytrace(tx,rx,pm,"Type","pathloss")
raysPerfect = raytrace(tx,rx,pm,"Type","pathloss");
plPerfect = [raysPerfect{1}.PathLoss]
plPerfect = 1×3

  104.2656   104.2745   112.0095
```


Site Viewer
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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Re-compute with material reflection loss by setting material type on 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 = 1×3

  104.2656   106.2547   146.5527
```



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

`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.

`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.

`propmodel` — Propagation model

character vector | string

Propagation model, specified as a character vector or string. You can also use the name-value pair `'PropagationModel'` to specify this parameter. You can also use the `propagationModel` function to define this input. The default propagation model is `'raytracing-image-method'`.

Data Types: `char` | `string`

Name-Value Pair Arguments

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

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-image-method'` (default) | ray tracing propagation model object

Type of propagation model for ray tracing analysis, specified as the comma-separated pair consisting of `'PropagationModel'` and `'raytracing-image-method'` or a ray tracing propagation model object created using `propagationModel`.

Data Types: `char`

`NumReflections` — Number of reflections to search for in propagation paths

`[0 1]` (default) | numeric row vector

Number of reflections to search for in propagation paths using ray tracing, specified as the comma-separated pair consisting of `'NumReflections'` and a numeric row vector whose elements are `0`, `1`, or `2`.

The default value results in the search for a line-of-sight propagation path along with propagation paths that each contain a single reflection.

Data Types: `double`

`Colormap` — Color map for coloring propagation paths

`'jet'` (default) | predefined color map name | *M*-by-3 array of RGB

Color map 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`

`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`

`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 and surface data

`siteviewer` object

Map for visualization and surface data, specified as a `siteviewer` object. The default value is the current Site Viewer.

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. array. Within each row vector, the `comm.Ray` objects are ordered by increasing number of reflections, and where number of reflections are equal they are ordered by increasing propagation distance.

See Also

`los` | `siteviewer`

Introduced in R2019b

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 pairs 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`.

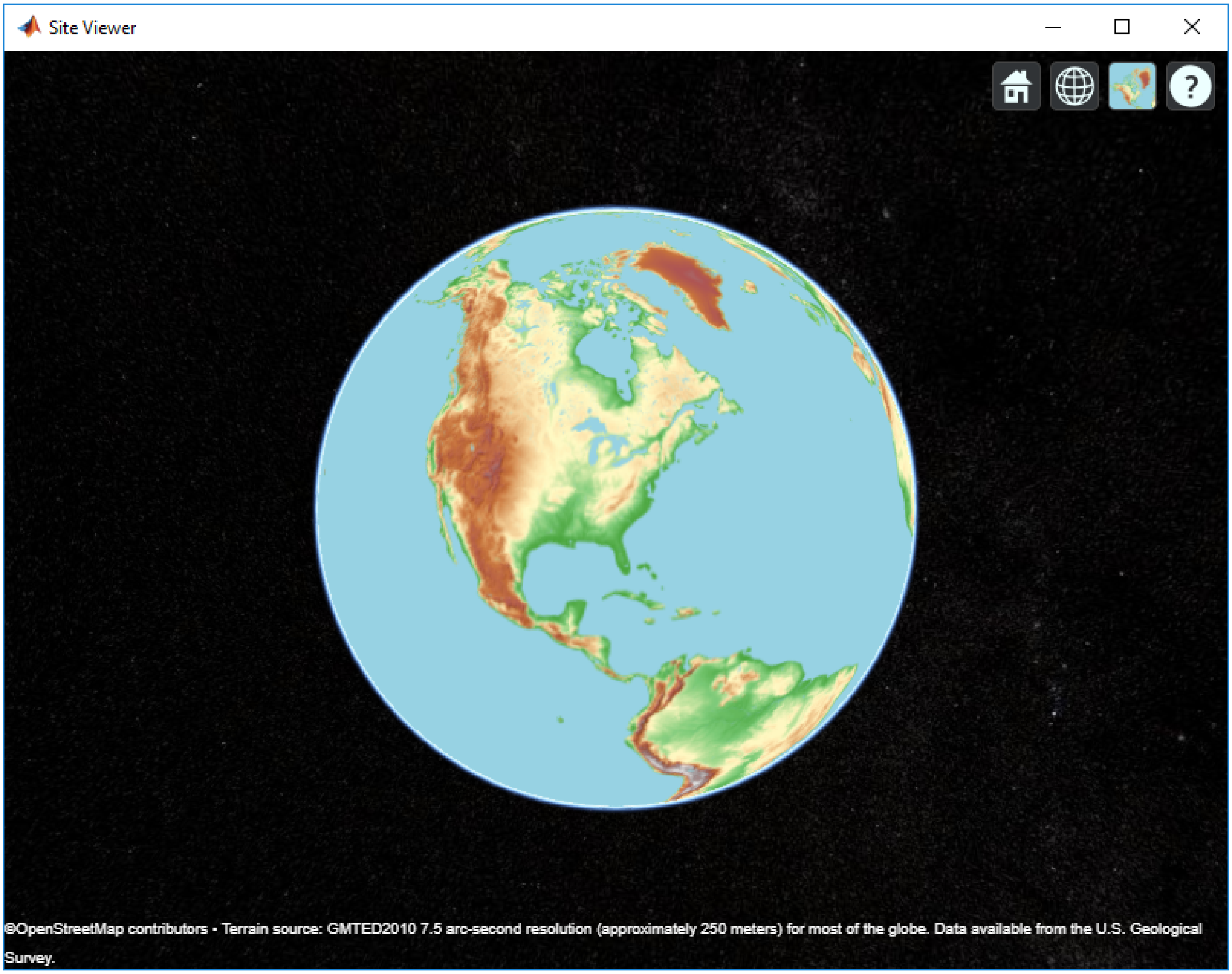
Initialize simulation variables to:

- 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.

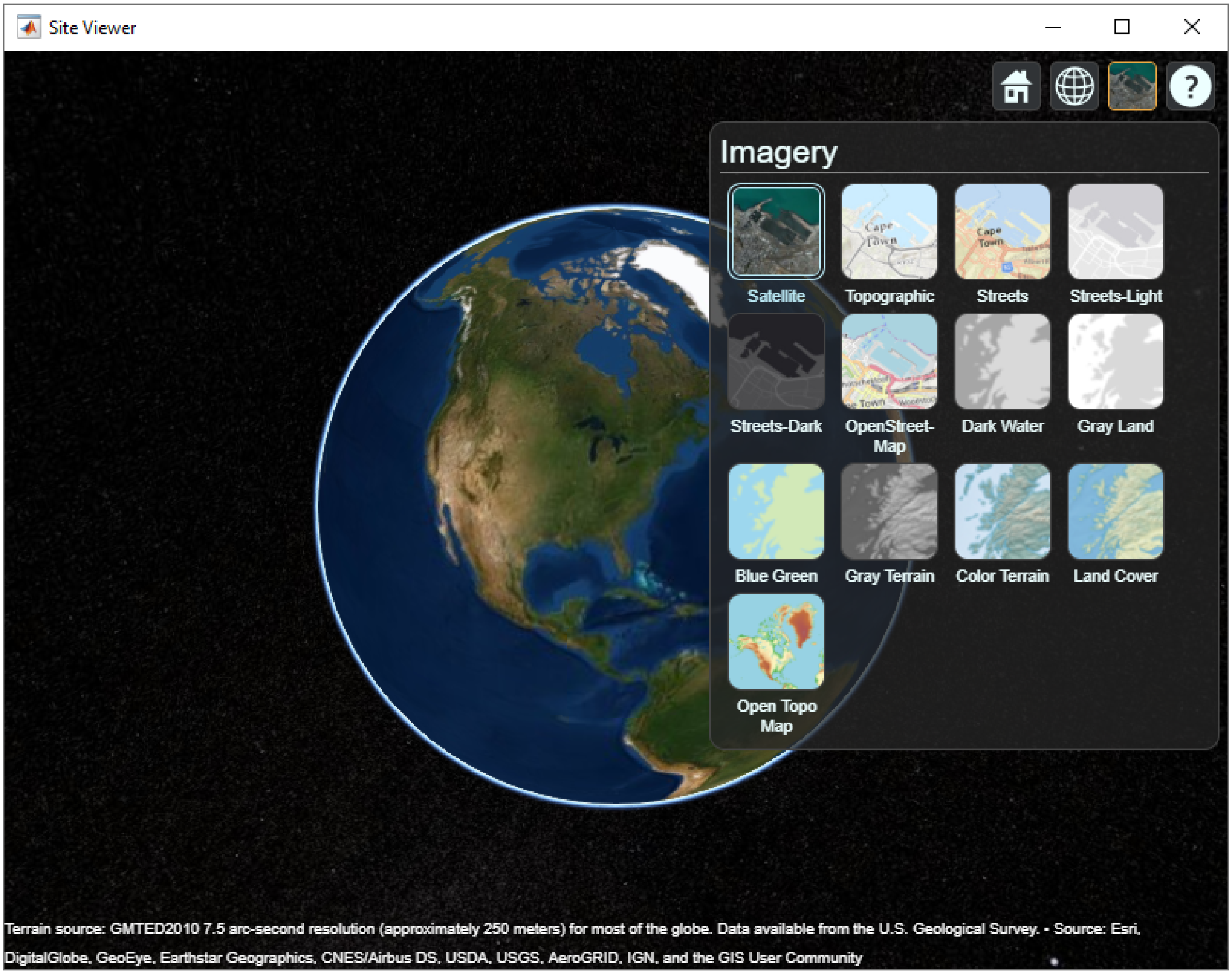
```
addCustomBasemap(name,url,'Attribution',attribution','DisplayName',displayName)
viewer = siteviewer('Basemap',name);
```



©OpenStreetMap contributors • Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey.

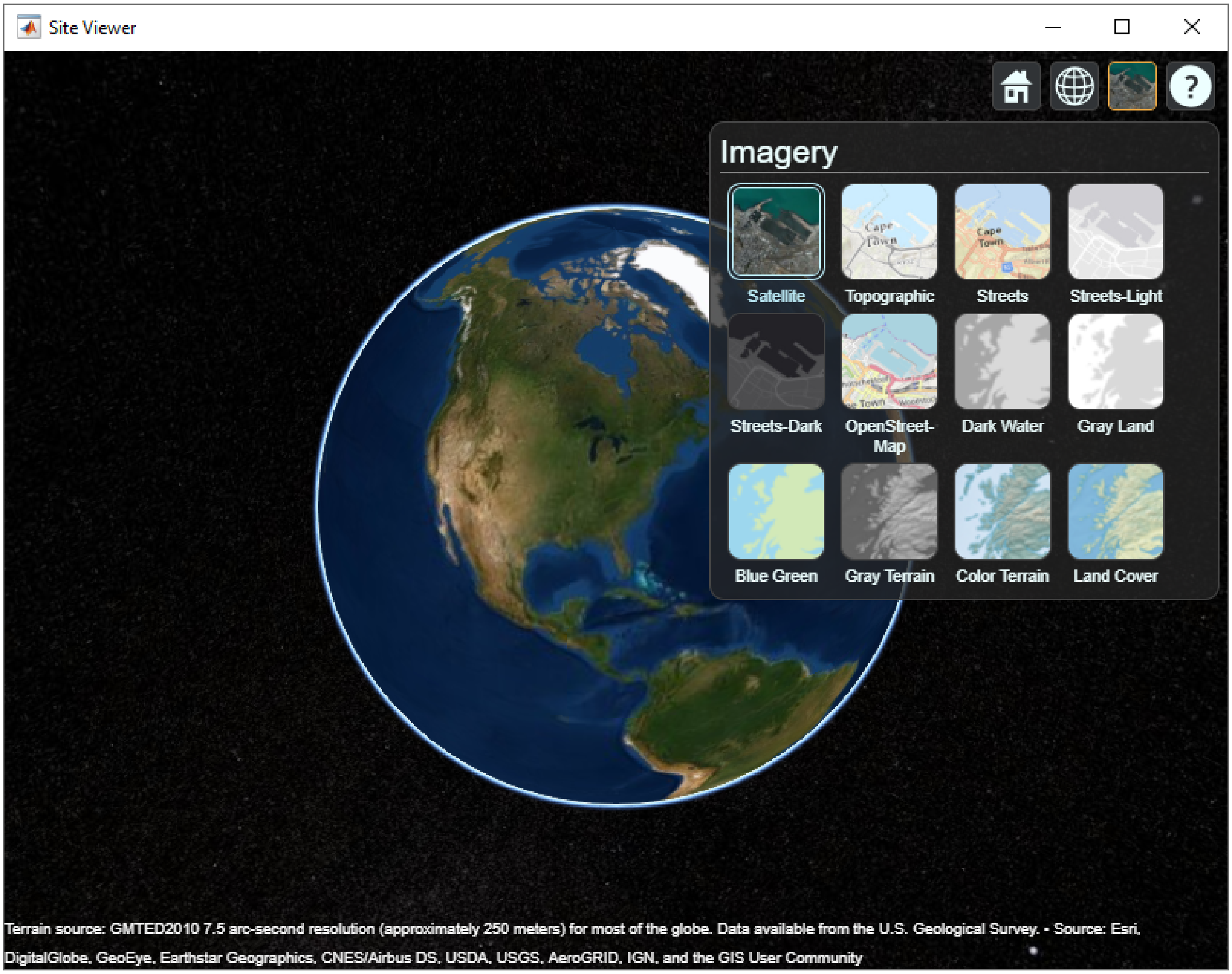
After a custom basemap is added to `siteviewer`, the custom map is available for future calls to `siteviewer`. Note the `'Open Topo Map'` icon in the `Imagery` tab.

```
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, DigitalGlobe, 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, DigitalGlobe, 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 Pair Arguments

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

Example: `addCustomBasemap(basemapName,URL,'Attribution',attribution)`

`Attribution` — Attribution of custom basemap
`'Tiles courtesy of` *`DOMAIN_NAME_OF_URL`*`'` (default) | string scalar | string array | character vector | cell array of character vectors

Attribution of custom basemap, specified as the comma-separated pair consisting of `'Attribution'` and a string scalar, string array, character vector, or cell array of character vectors. If the host is `'localhost'`, or if `URL` contains only IP numbers, specify an empty value (`''`). To create a multiline attribution, specify a string array or nonscalar cell array of character vectors.

If you do not specify an attribution, the default attribution is `'Tiles courtesy of` *`DOMAIN_NAME_OF_URL`*`'`, where the `addCustomBasemap` function obtains the domain name from the `URL` input argument.

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 the comma-separated pair consisting of `'DisplayName'` and a string scalar or character vector.

Example: `'OpenStreetMap'`

Data Types: `string` | `char`

`MaxZoomLevel` — Maximum zoom level of basemap
`18` (default) | integer in the range [0, 25]

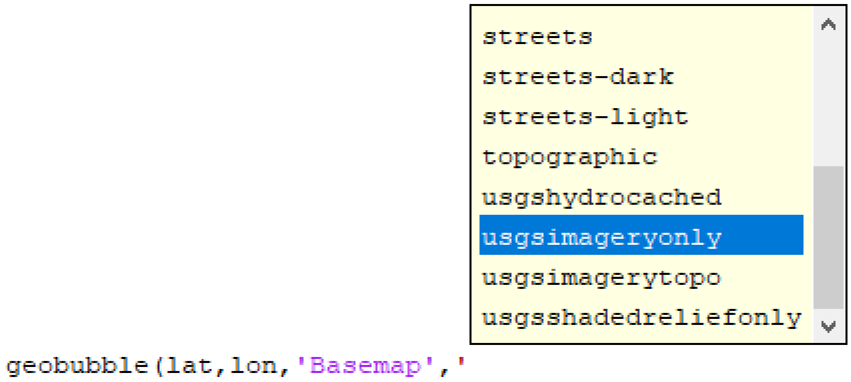
Maximum zoom level of the basemap, specified as the comma-separated pair consisting of `'MaxZoomLevel'` and an integer in the range [0, 25].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

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.

- To access a list of available basemaps, press **Tab** before specifying the basemap in your plotting function.



```
geobubble(lat,lon,'Basemap','
```

See Also

geoaxes | geobasemap | geobubble | removeCustomBasemap

removeCustomBasemap

Remove custom basemap

Syntax

`removeCustomBasemap(basemapName)`

Description

`removeCustomBasemap(basemapName)` removes the custom basemap specified by `basemapName` from the list of available basemaps.

Examples

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

`addCustomBasemap` | `geoaxes` | `geobasemap` | `geobubble` | `geodensityplot` | `geoplot` | `geoscatter`

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
```

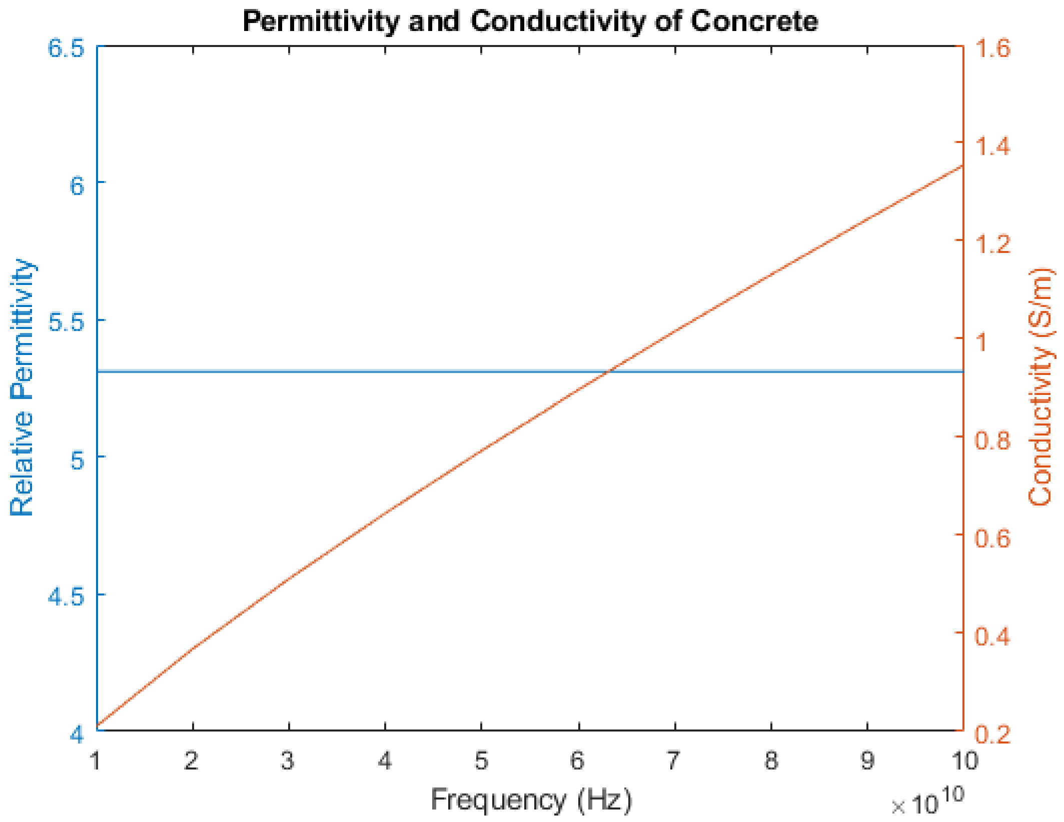
Plot Permittivity and Conductivity of Concrete at Various Frequencies

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 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`.

`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`.

`complexepsilon` — Complex relative permittivity
complex scalar

Complex relative permittivity of the building material, returned as a complex scalar calculated by

`complexepsilon` = `epsilon` – 1*i* `sigma` / (2π`fc`ε_0),

where ε_0 = 8.854187817e-12 Farads/m. ε_0 is the electric constant for the permittivity of free space.

The output dimension of `complexepsilon` matches that of the input argument `material`.

References

[1] ITU-R P.2040-1. "Effects of Building Materials and Structures on Radiowave Propagation Above 100MHz." *International Telecommunications Union - Radiocommunications Sector (ITU-R)*. July 2015.

See Also

Functions
earthSurfacePermittivity | raypl | raytrace

Objects
comm.Ray

Introduced in R2020a

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 tempera
```

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
```

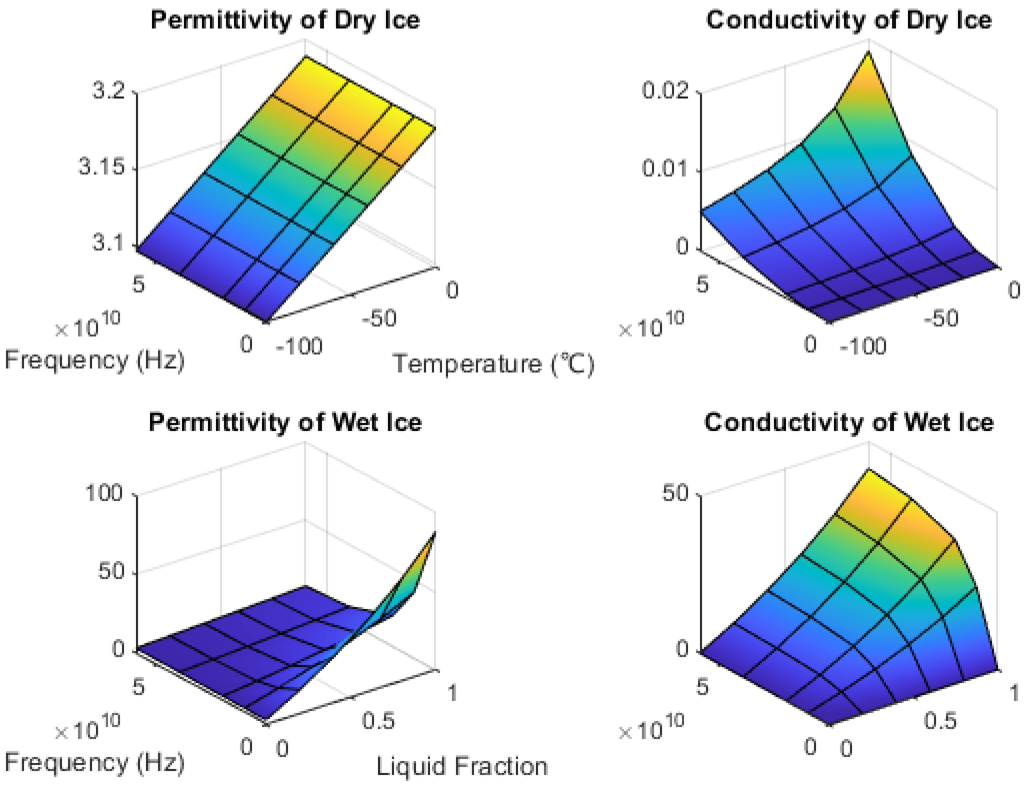
```
ans = -9.2179e-16
```

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.

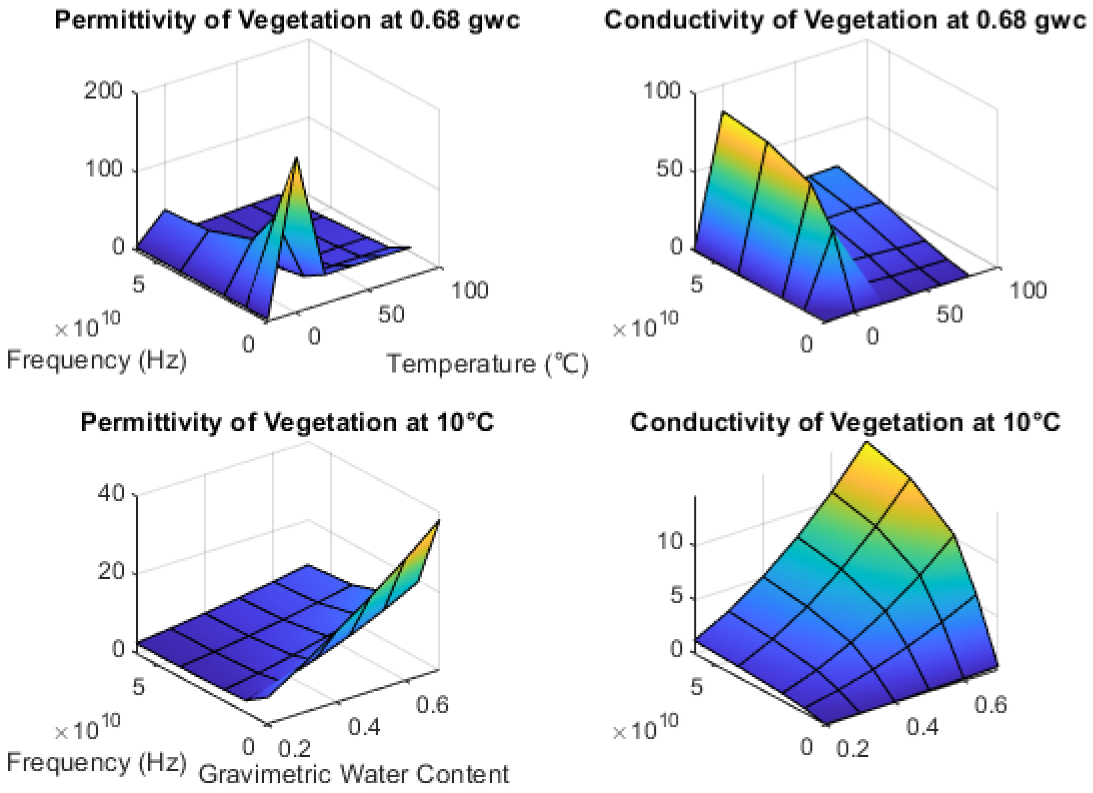
```
fc = repmat([0.1,10,20,40,60]*1e9,6,1);
gwc1 = 0.68;
temp1 = repmat((-20:20:80)',1,5);
[epsilon_veg_gwc,sigma_veg_gwc] = ...
    arrayfun(@(x,y)earthSurfacePermittivity('vegetation',x,y,gwc1),fc,temp1);
```

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')
```



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 7-178.

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 7-178.

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 7-178.

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 by

$$\texttt{complexepsilon} = \texttt{epsilon} - 1i\ \texttt{sigma} / (2\pi \texttt{fc} \varepsilon_0),$$

where ε_0 = 8.854187817e-12 Farads per meter (F/m). 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`)

References

[1] ITU-R P.527-5. "Electrical characteristics of the surface of the Earth." *International Telecommunications Union - Radiocommunications Sector (ITU-R)*. August 2019.

See Also

Functions

`buildingMaterialPermittivity` | `raypl` | `raytrace`

Objects

`comm.Ray`

Introduced in R2020a

raypl

Calculate path loss and phase shift for ray

Syntax

```
[pl,phase] = raypl (ray)
[pl,phase] = raypl (ray,Name,Value)
```

Description

`[pl,phase] = raypl (ray)` returns the path loss in dB and phase shift in radians based on the properties specified by `ray`. The path loss and path shift computations consider the free space loss and reflection loss derived from the propagation path, reflection materials, and polarizations. The function accounts for geometric coupling between horizontal and vertical polarizations only when both transmit and receive antennas are polarized. For more information, see “Path Loss Computation” on page 7-183.

`[pl,phase] = raypl (ray,Name,Value)` calculates the path loss and phase shift with additional options specified by one or more name-value pair arguments.

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. 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.

```
rays = raytrace(tx,rx,"NumReflections",0:2);
```

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}.NumReflections] == 2,1))
```

```
ray = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
```

```
        ReceiverLocation: [3×1 double]
             LineOfSight: 0
     ReflectionLocations: [3×2 double]
               Frequency: 2.8000e+10
          PathLossSource: 'Custom'
                PathLoss: 125.5038
              PhaseShift: 1.4198

   Read-only properties:
        PropagationDelay: 8.3062e-07
     PropagationDistance: 249.0122
        AngleOfDeparture: [2×1 double]
          AngleOfArrival: [2×1 double]
          NumReflections: 2
```

```
plot(ray);
```

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
     ReflectionLocations: [3×2 double]
               Frequency: 2.8000e+10
          PathLossSource: 'Custom'
                PathLoss: 127.6379
              PhaseShift: 4.6001

   Read-only properties:
        PropagationDelay: 8.3062e-07
     PropagationDistance: 249.0122
        AngleOfDeparture: [2×1 double]
          AngleOfArrival: [2×1 double]
          NumReflections: 2


ray = 
  Ray with properties:

       PathSpecification: 'Locations'
        CoordinateSystem: 'Geographic'
     TransmitterLocation: [3×1 double]
        ReceiverLocation: [3×1 double]
             LineOfSight: 0
     ReflectionLocations: [3×2 double]
               Frequency: 2.8000e+10
```

```
         PathLossSource: 'Custom'
               PathLoss: 127.6379
             PhaseShift: 4.6001

   Read-only properties:
       PropagationDelay: 8.3062e-07
    PropagationDistance: 249.0122
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
         NumReflections: 2
```

```
plot(ray);
```

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);
```

Input Arguments

ray — Ray configuration

`comm.Ray` object

Ray configuration, specified as one `comm.Ray` object. The object must have the `PathSpecification` property set to `"Locations"`.

Data Types: `comm.Ray`

Name-Value Pair Arguments

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

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 | 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, 1-by-*NR* string vector, 2-by-1 numeric vector, or 2-by-*NR* numeric matrix. *NR* represents the number of reflections as specified by the `comm.Ray.NumReflections` property.

- When `ReflectionMaterials` is specified as a string scalar or string vector, the reflection material must be one of `"concrete"`, `"brick"`, `"wood"`, `"glass"`, `"metal"`, `"water"`, `"vegetation"`, `"loam"`, or `"perfect-reflector"`. When specified as a string scalar, the setting applies to all the reflections.
- When `ReflectionMaterials` is specified as a 2-by-1 numeric vector, the [relative permittivity; conductivity] value pair applies to all the reflections.

- When `ReflectionMaterials` is specified as a 2-by-*NR* numeric matrix, the [relative permittivity; conductivity] value pair in each column applies for each of the *NR* reflection points, respectively.

Example: `"ReflectionMaterials",["concrete","water"]`, specifies that a ray with two reflections will use 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) | `"H"` | `"V"` | `"RHCP"` | `"LHCP"` | normalized 2-by-1 Jones vector

Transmit antenna polarization type, specified as `"none"`, `"H"`, `"V"`, `"RHCP"`, `"LHCP"`, or a normalized [H; V] Jones vector. For more information, see “Jones Vector Notation” on page 7-185.

Example: `'TransmitterPolarization','RHCP'`, specifies right-hand circular polarization for the transmit antenna.

Data Types: `double` | `char` | `string`

`ReceiverPolarization` — Receive antenna polarization type

`"none"` (default) | `"H"` | `"V"` | `"RHCP"` | `"LHCP"` | normalized 2-by-1 Jones vector

Receive antenna polarization type, specified as `"none"`, `"H"`, `"V"`, `"RHCP"`, `"LHCP"`, or a normalized [H; V] Jones vector. For more information, see “Jones Vector Notation” on page 7-185.

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 7-183.

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 7-183.

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
scalar

Path loss in dB, returns the path loss calculated for the input ray object, accounting for any modifications specified by `Name,Value` pairs.

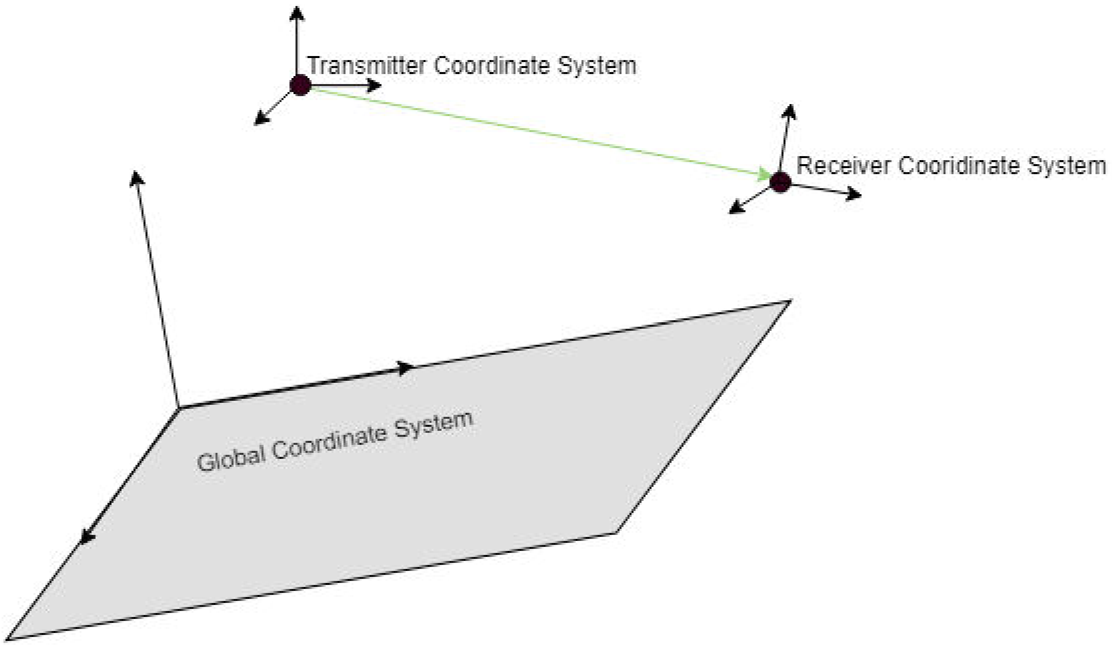
phase — Phase shift
scalar

Phase shift in radians, returns the phase shift calculated for the input ray object, accounting for any modifications specified by `Name,Value` pairs.

More About

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 path loss computations in raypl follow the path loss and reflection matrix computations as described in IEEE Document 802.11-09/0334r8 [1]. The function accounts for geometric coupling between horizontal and vertical polarizations only when both transmit and receive antennas are polarized.

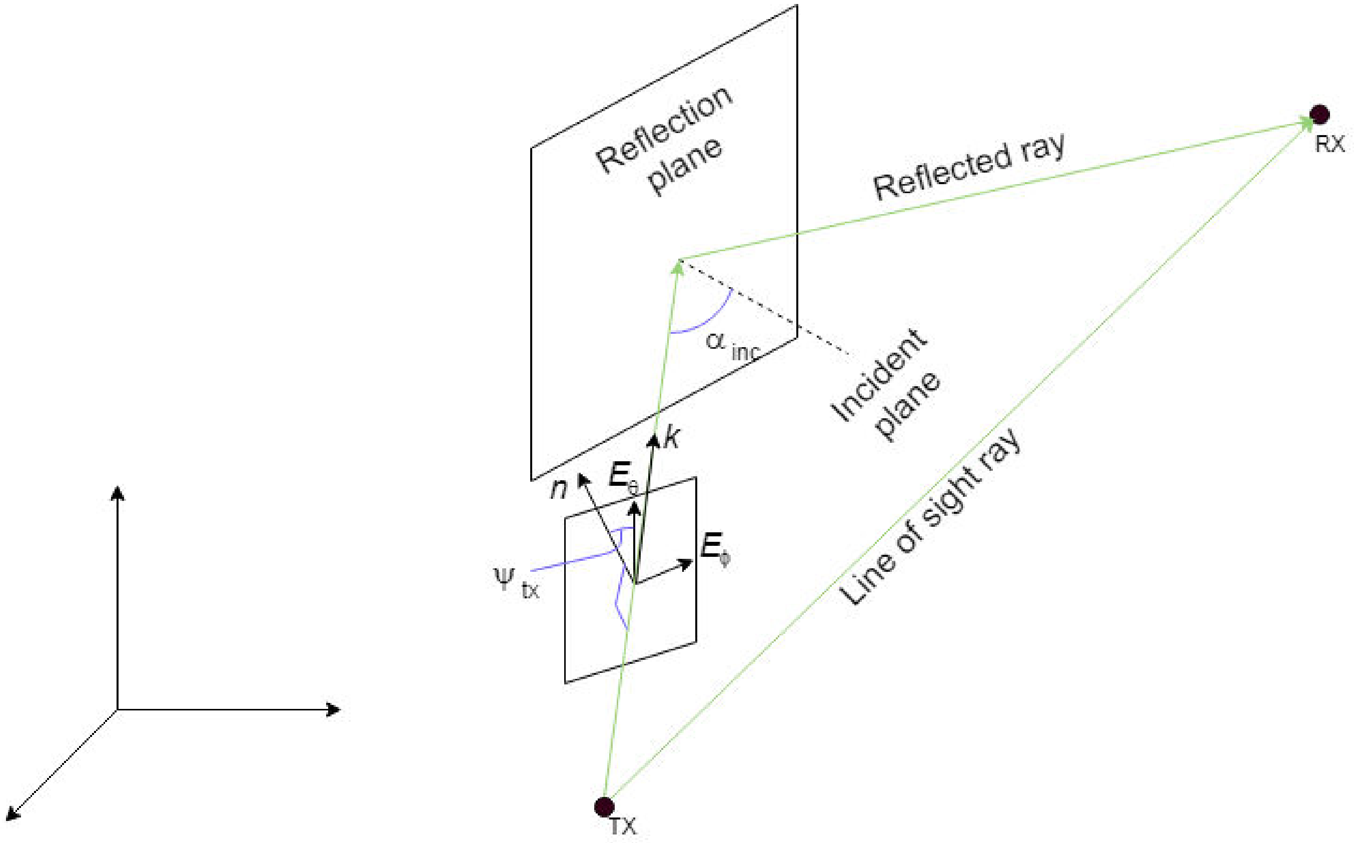
For a first order signal reflection, the reflection matrix, H_{ref1}, is computed as

$$H_{ref1} = \begin{bmatrix} \cos(\psi_{rx}) & \sin(\psi_{rx}) \\ -\sin(\psi_{rx}) & \cos(\psi_{rx}) \end{bmatrix} \times \begin{bmatrix} R_{\perp}(\alpha_{inc}) & 0 \\ 0 & R_{\parallel}(\alpha_{inc}) \end{bmatrix} \times \begin{bmatrix} \cos(\psi_{tx}) & \sin(\psi_{tx}) \\ -\sin(\psi_{tx}) & \cos(\psi_{tx}) \end{bmatrix}$$

The terms in the channel propagation matrix computation represent

- RX geometric coupling matrix — Recalculation of the polarization vector from the plane of incidence basis to RX coordinates.
- Polarization matrix — Matrix includes the reflection coefficients *R*□ and *R*∥ for the perpendicular and parallel components of the electric field *E* □ and *E* ∥ respectively.
- TX geometric coupling matrix — Recalculation of the polarization vector from the TX coordinates basis to the plane of incidence.

This figure illustrates a first order reflected signal path.



Where

- The reflection plane is offset from the global coordinate system origin.
- *k* represents the waveform propagation vector.
- *n* represents the vector normal to the incident plane.

- E_θ and E_φ represent the vertical and horizontal electromagnetic field vectors.
- α_{inc} represents the incident angle of k.
- ψ_{tx} represents the angle between E_θ and a normal to the incident plane.
- TX represents the transmit antenna.
- RX represents the receive antenna.

The reflection matrix computations for second order signal reflections extend from the first order signal reflection computations. For more information, see IEEE Document 802.11-09/0334r8 [1].

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)	$\begin{pmatrix} H \\ V \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} -j \\ 1 \end{pmatrix}$

References

[1] Maltsev, A., et al. "Channel models for 60 GHz WLAN systems." IEEE Document 802.11-09/0334r8, May 2010.

See Also

Functions
`buildingMaterialPermittivity` | `earthSurfacePermittivity` | `propagationModel` | `raytrace`

Objects
`comm.Ray` | `siteviewer`

Introduced in R2020a

location

Data location coordinates

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.

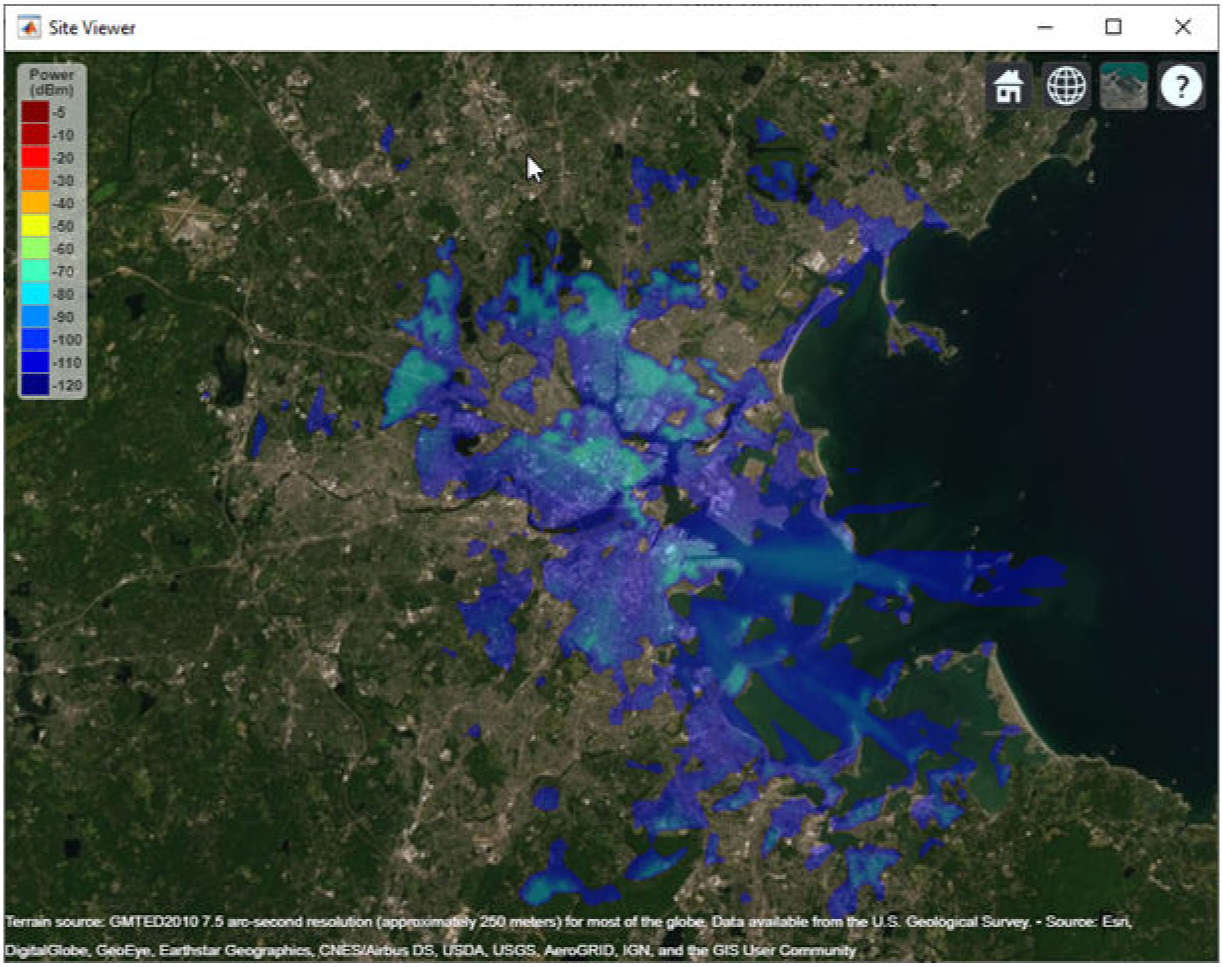
```
v1 = interp(pd1,lats,lons);
v2 = interp(pd2,lats,lons);
v3 = interp(pd3,lats,lons);
```

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, DigitalGlobe, 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]`.

See Also

`getDataVariable` | `interp`

Introduced in R2020a

plot

Plot propagation data on map

Syntax

```
plot(pd)
plot(___,Name,Value)
```

Description

`plot(pd)` plots the propagation data on a map. Each data point is displayed as a circular marker that is colored according to the corresponding value.

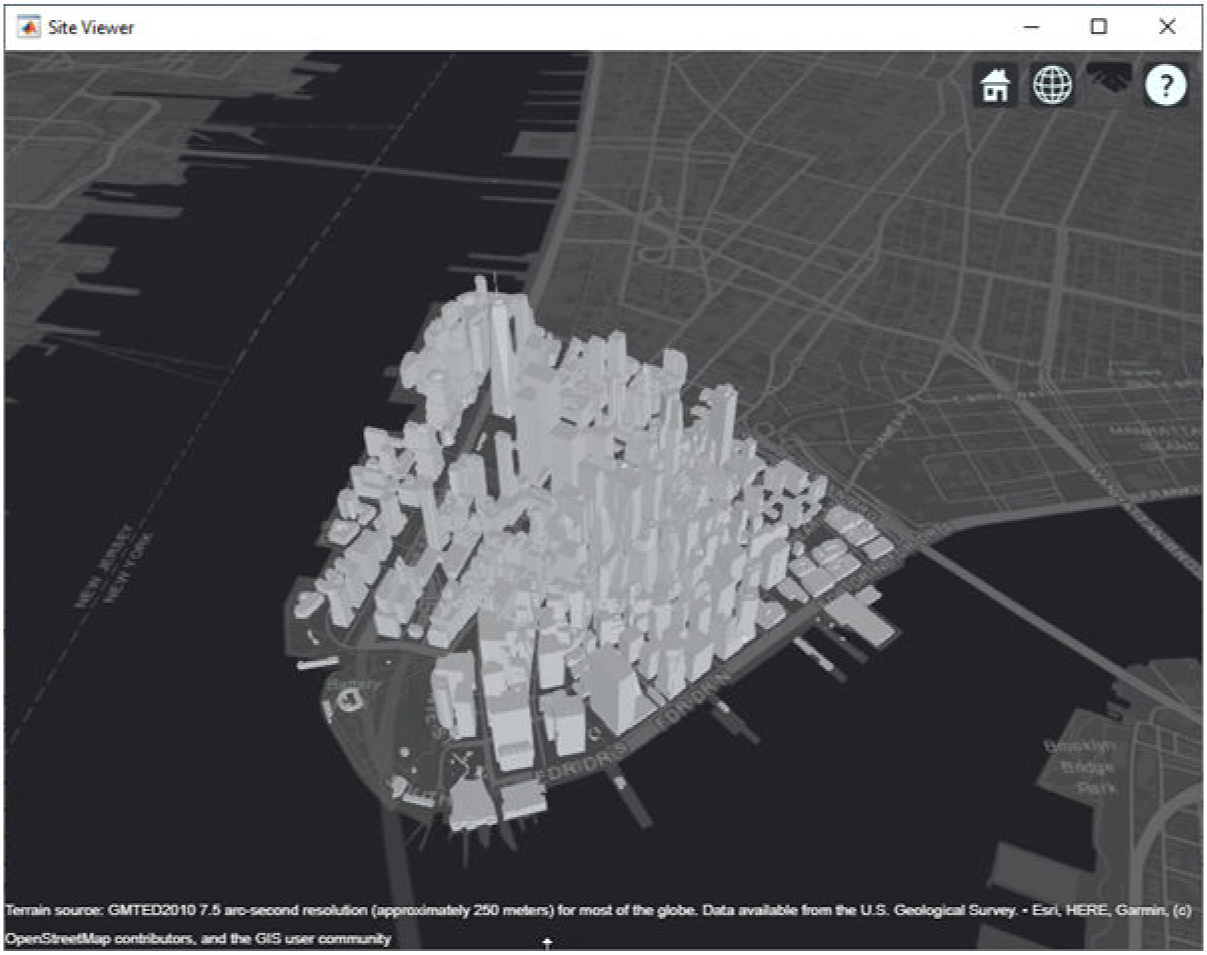
`plot(___,Name,Value)` plots the propagation data with additional options specified by name-value pair arguments.

Examples

Compute Signal Strength Data in Urban Environment

Launch Site Viewer with basemaps and building files.

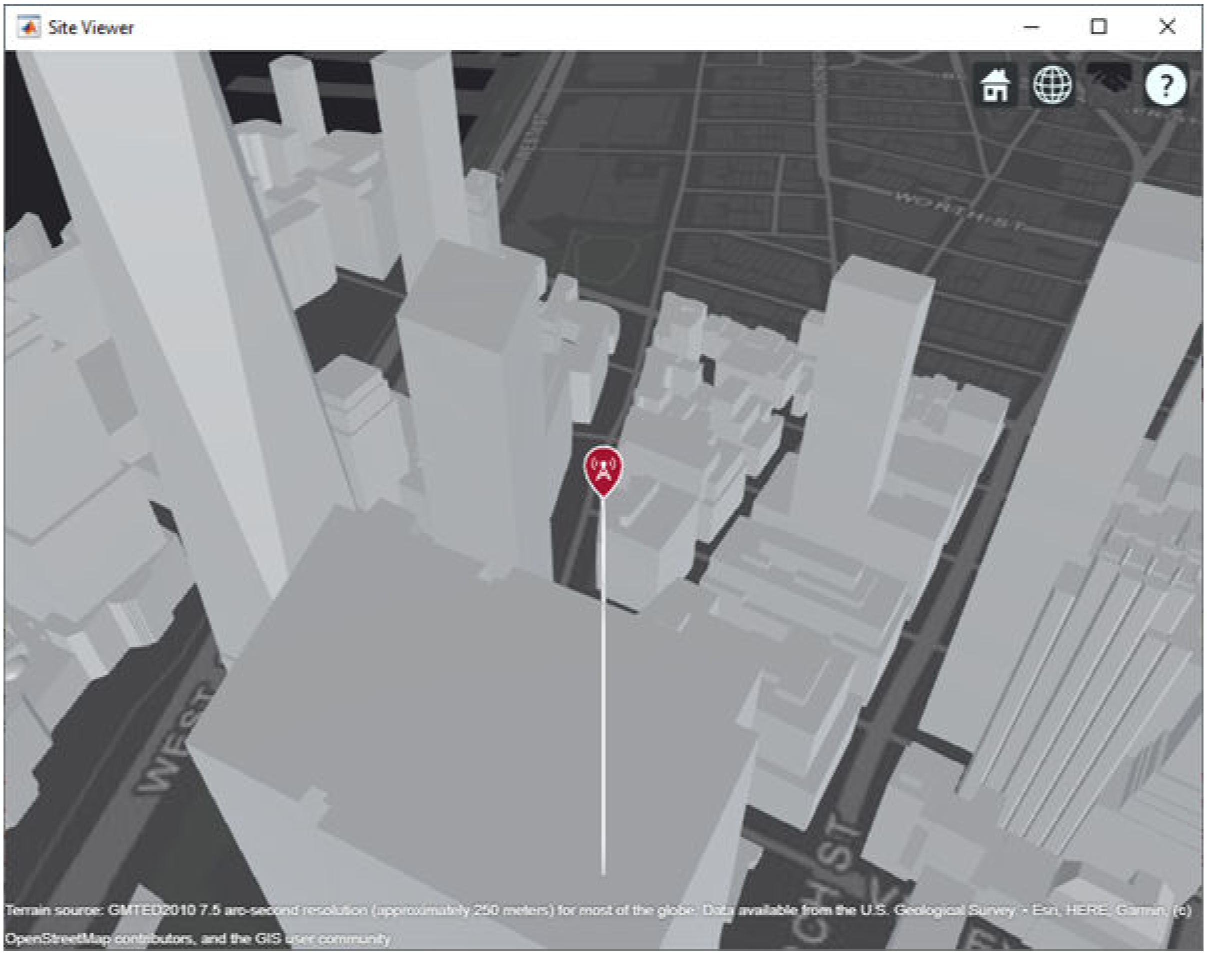
```
viewer = siteviewer("Basemap","streets_dark",...
        "Buildings","manhattan.osm");
```



Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • 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)
```


Site Viewer
WORTH ST
WEST
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.

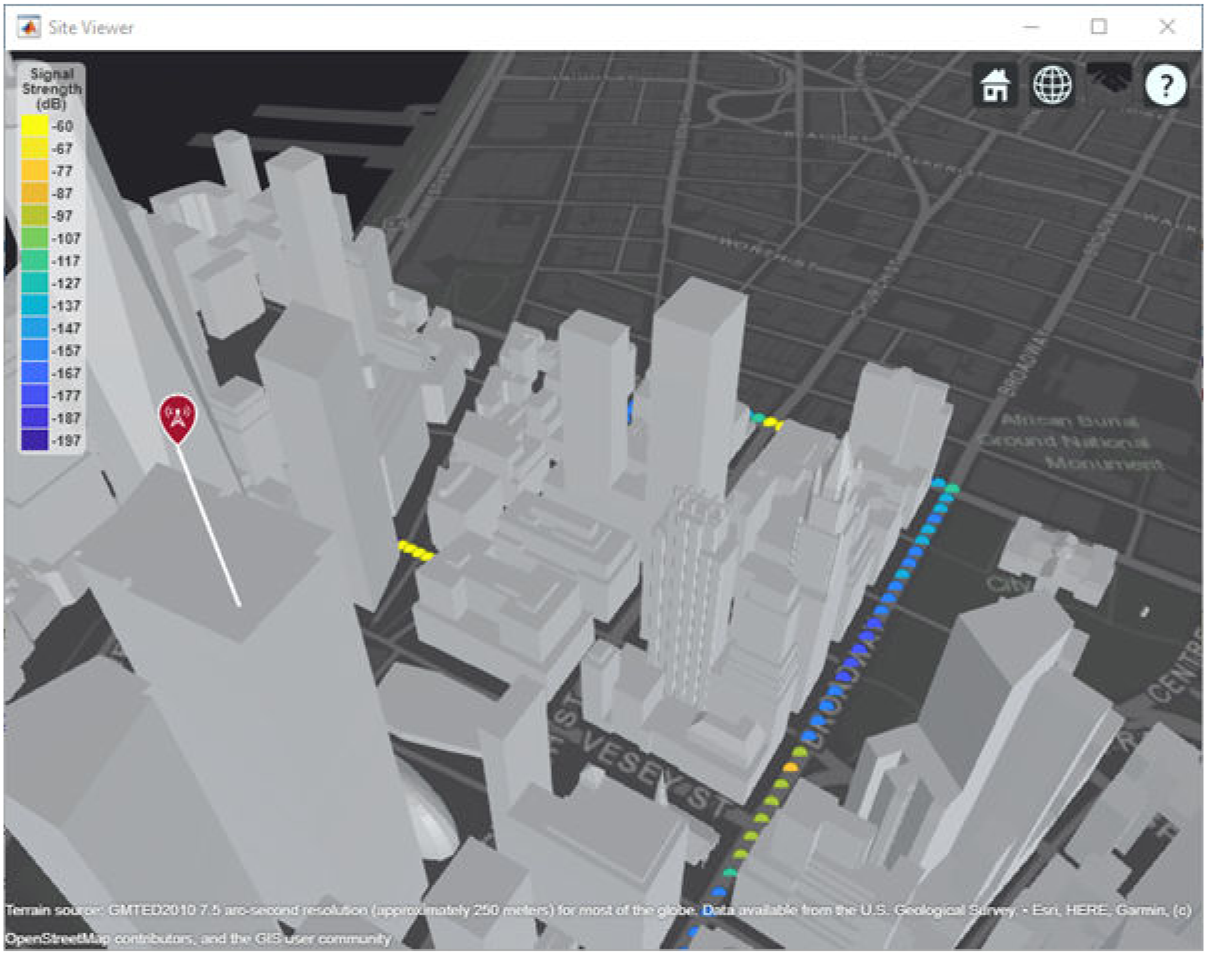
```
signalStrength = sigstrength(rxs, tx)';
```

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);
```


Site Viewer
Signal Strength (dB)
-60
-67
-77
-87
-97
-107
-117
-127
-137
-147
-157
-167
-177
-187
-197
VESEY ST
BROADWAY
African Burial Ground National Monument
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

Input Arguments

pd — Propagation data

propagationData object (default)

Propagation data, specified as a propagationData object.

Name-Value Pair Arguments

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

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. The default value is the current Site Viewer or a new Site Viewer, if none is open.

Data Types: `char` | `string`

See Also

`contour` | `interp`

Introduced in R2020a

getDataVariable

Get data variable values of data points in propagation data object

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

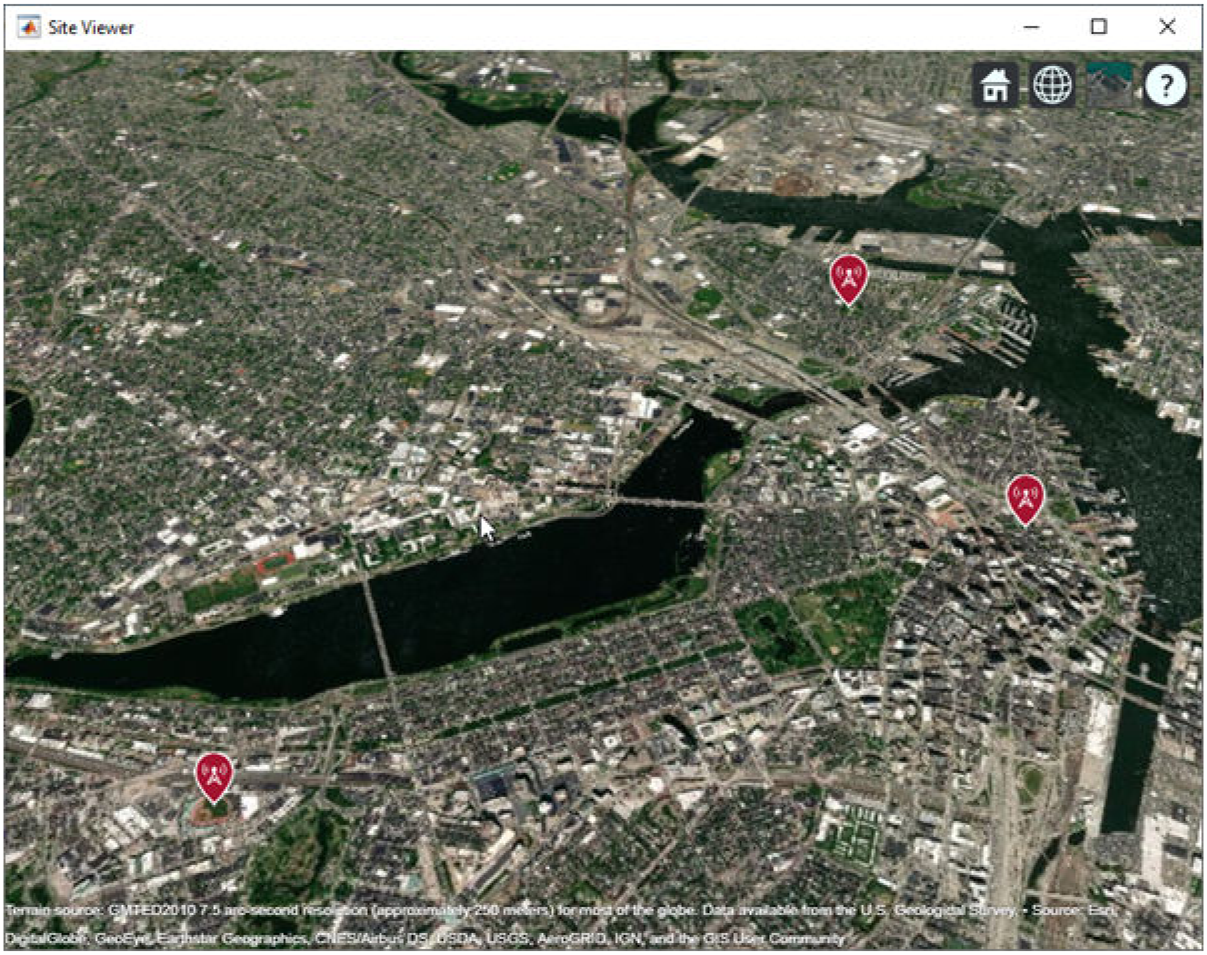
Capacity Map Using SINR Data

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];
```

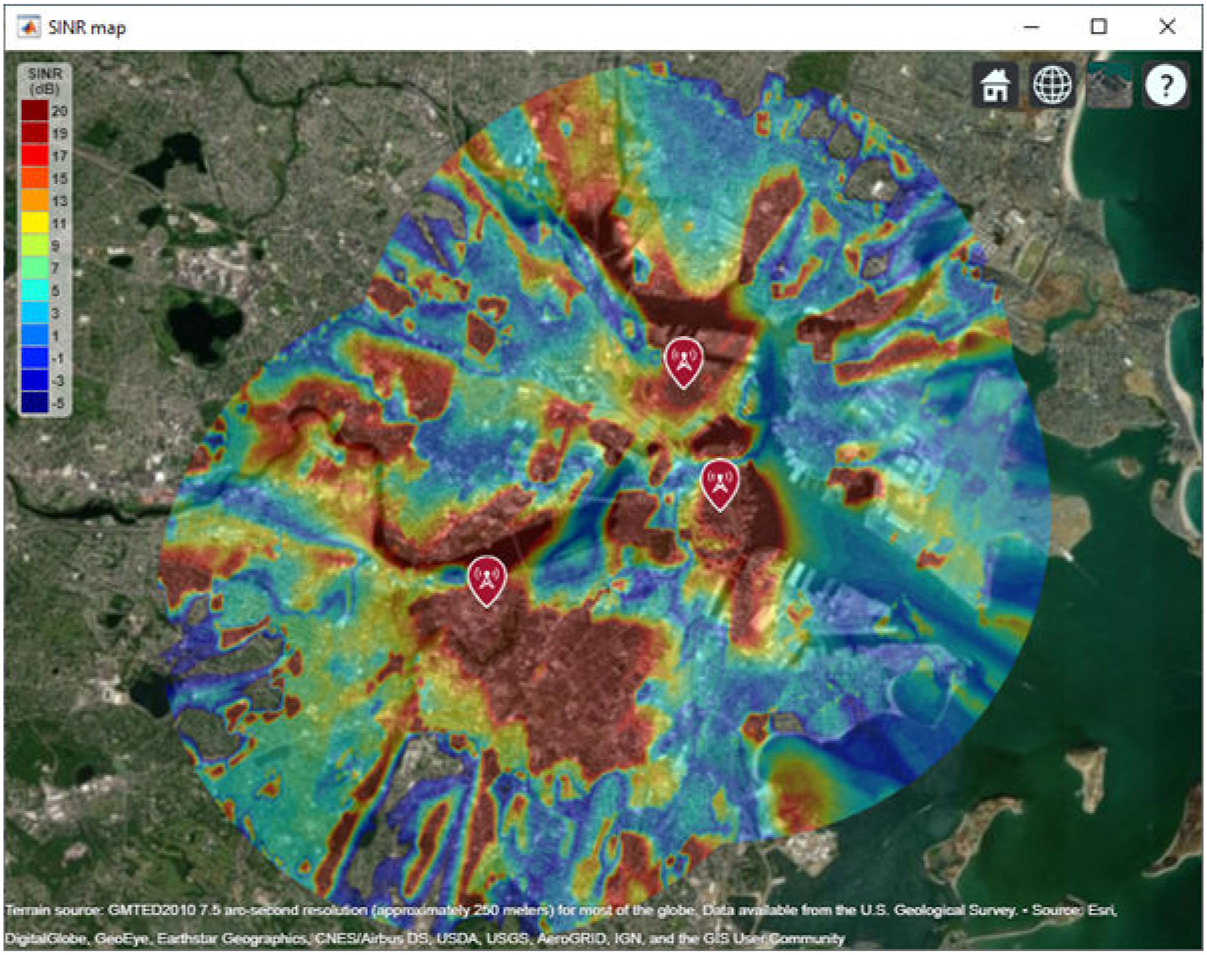
Create an array of transmitter sites.

```
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, DigitalGlobe, 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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Return SINR propagation data.

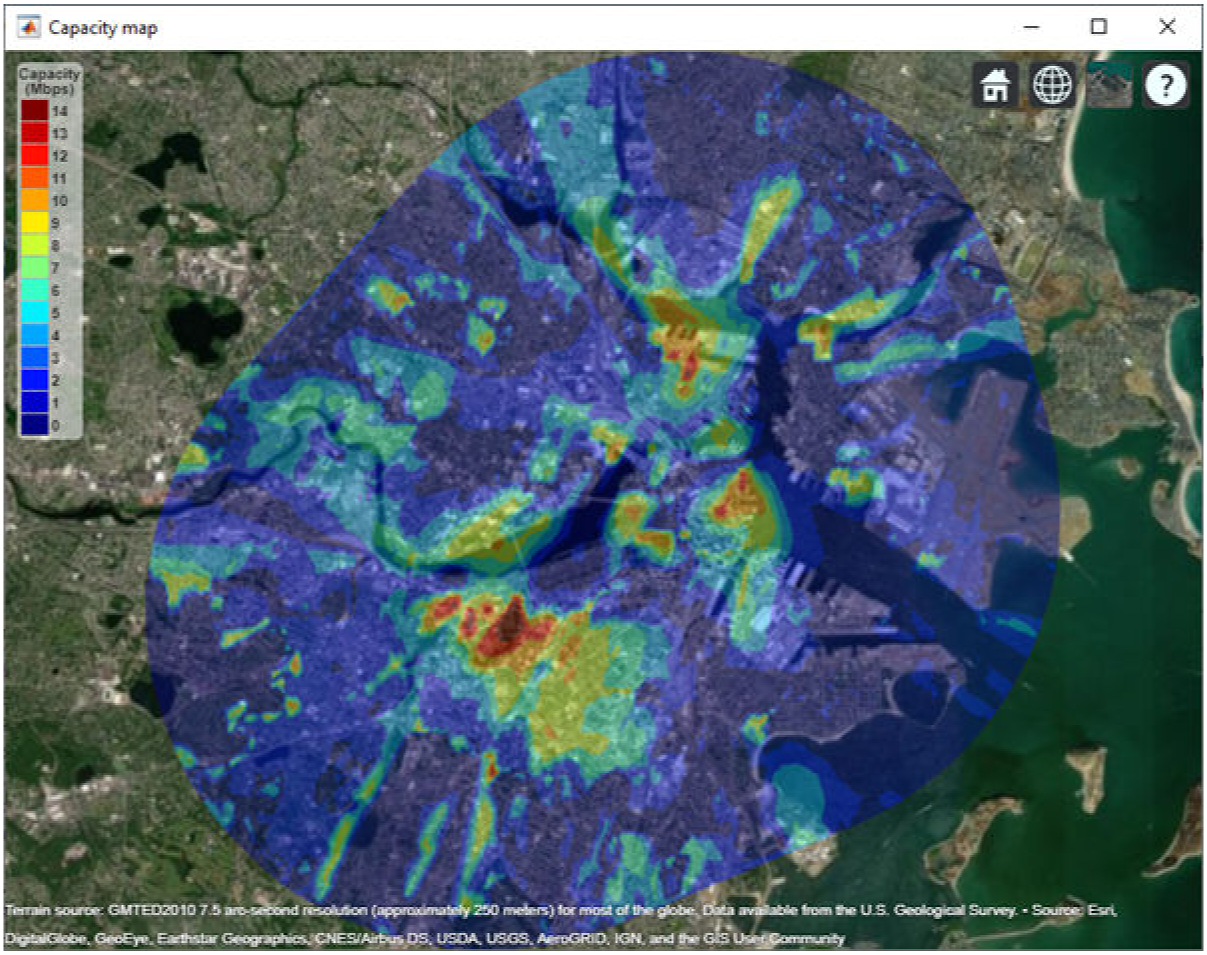
```
pd = sinr(txs,"MaxRange",5000);
[sinrDb,lats,lons] = getDataVariable(pd,"SINR");
```

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 capacity 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, DigitalGlobe, 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]`.

See Also

`interp` | `location`

Introduced in R2020a

interp

Geographic data interpolation

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.

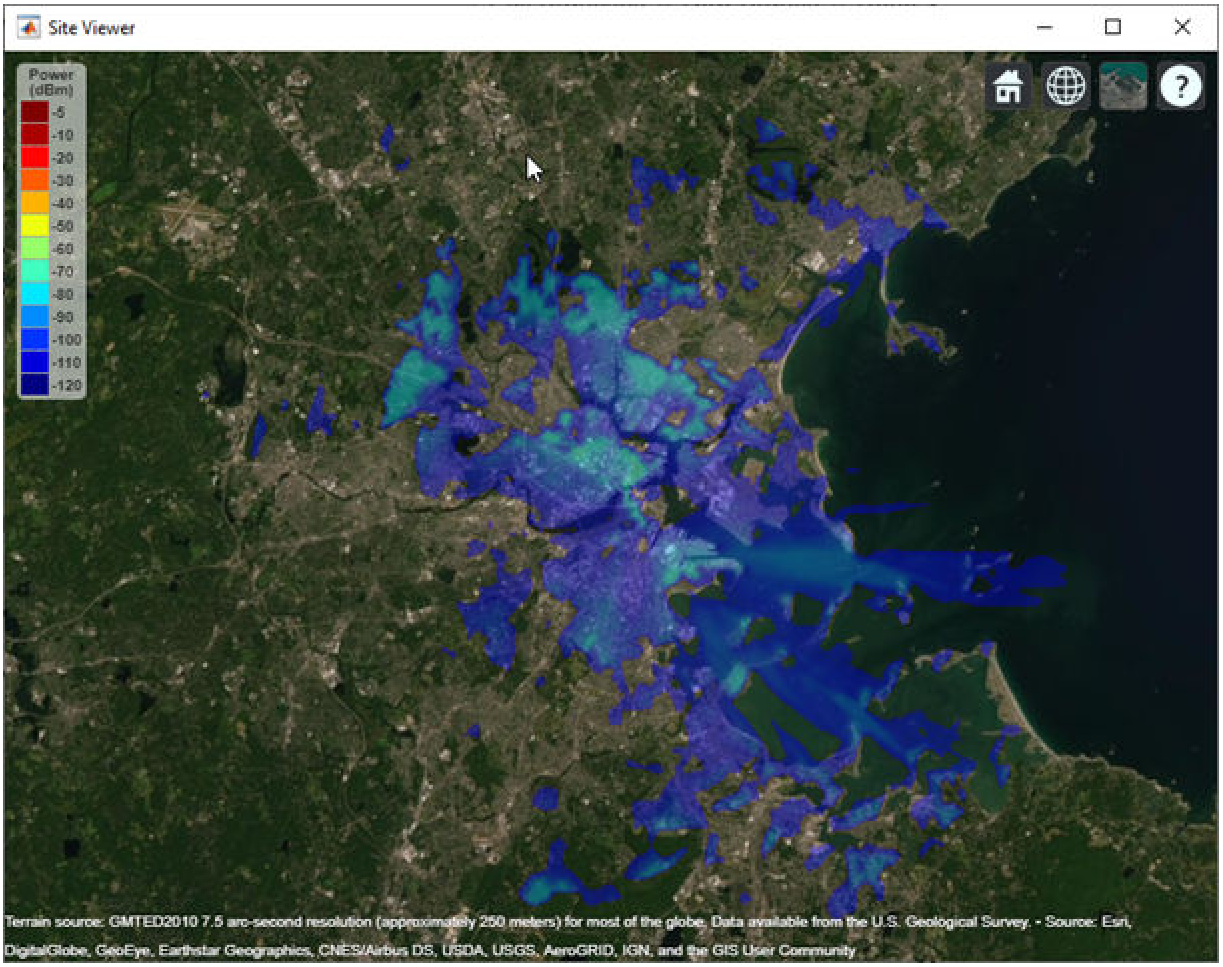
```
v1 = interp(pd1,lats,lons);
v2 = interp(pd2,lats,lons);
v3 = interp(pd3,lats,lons);
```

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

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.

See Also

`contour` | `getDataVariable` | `location` | `plot`

Introduced in R2020a

contour

Display contour map

Syntax

```
contour(pd)
contour( ___ ,Name,Value)
```

Description

`contour(pd)` creates a filled contour plot on a map. 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

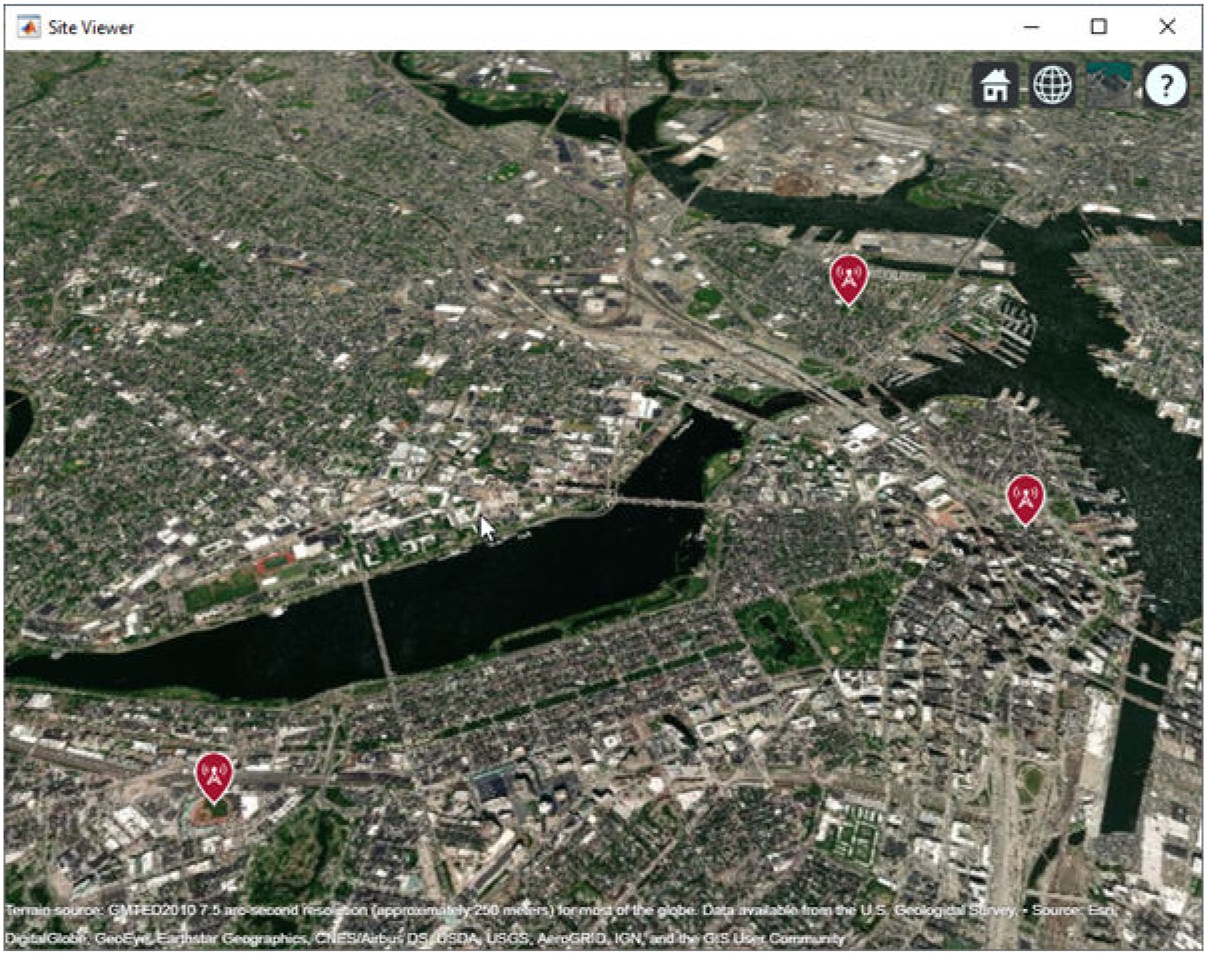
Capacity Map Using SINR Data

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];
```

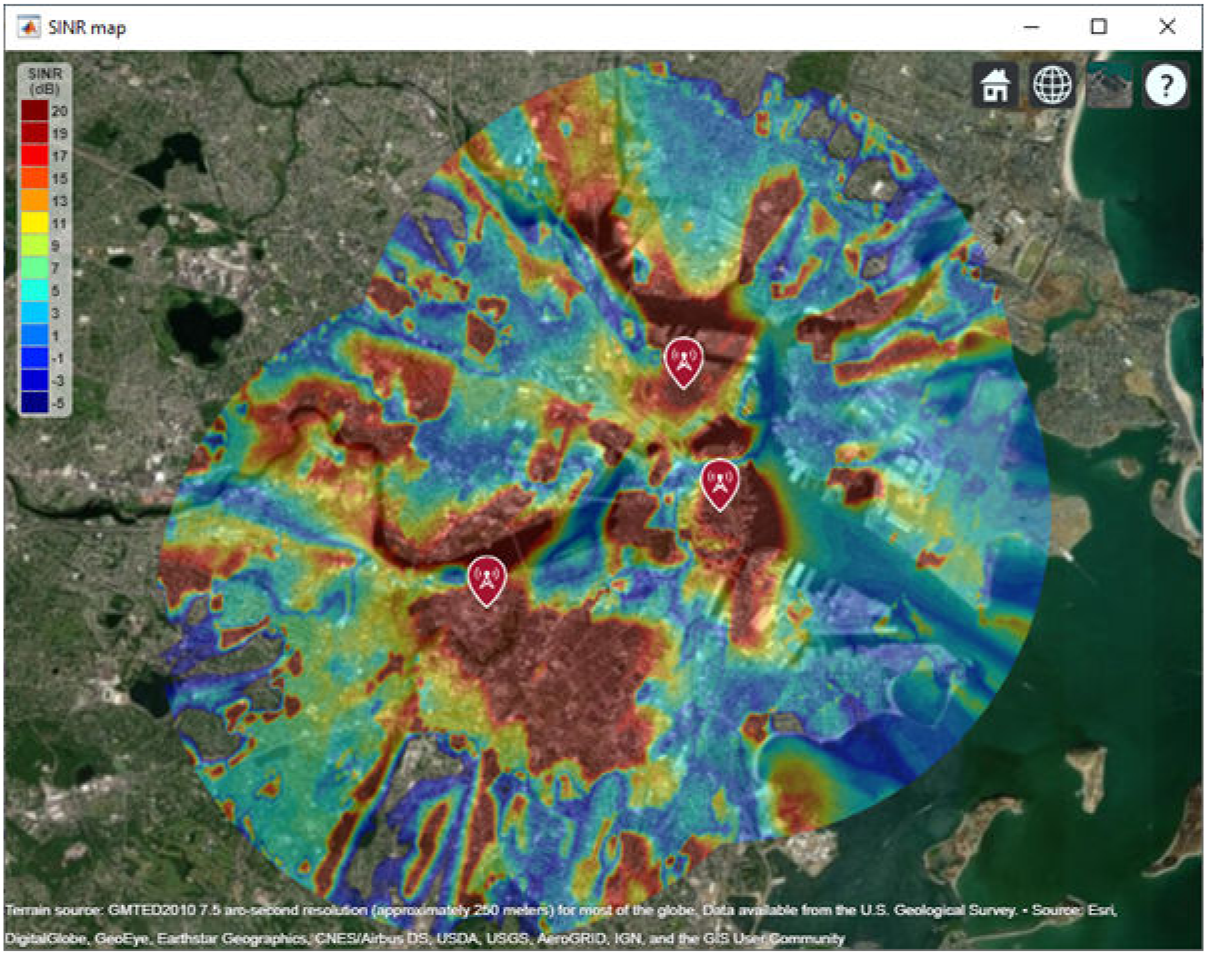
Create an array of transmitter sites.

```
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, DigitalGlobe, 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, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Return SINR propagation data.

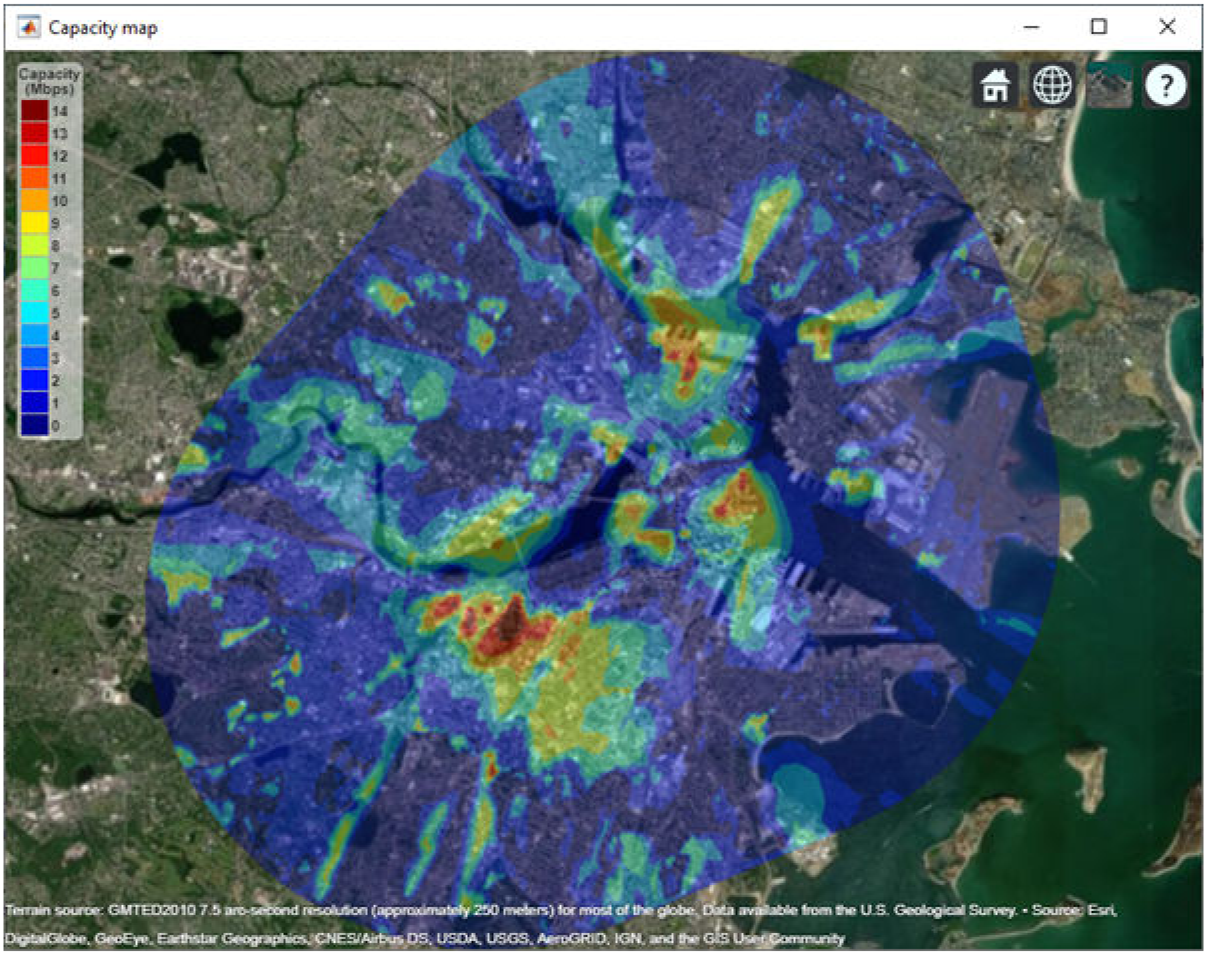
```
pd = sinr(txs,"MaxRange",5000);
[sinrDb,lats,lons] = getDataVariable(pd,"SINR");
```

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

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 `'Colormpa'` 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. The default value is the current Site Viewer or a new Site Viewer, if none is open.

Data Types: `char` | `string`

See Also

`getDataVariable` | `interp` | `plot`

Introduced in R2020a

add

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 multipath propagation model, then pmc is also a multipath propagation model where path losses from rain, gas, or fog models in the composite are added to the path loss computed for each propagation path.

See Also

propagationModel | range

Introduced in R2020a

comm.Ray

Propagation ray container object

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 container 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], and *height* must be nonnegative. The object does not perform range validation for *longitude*.

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], and *height* must be nonnegative. The object does not perform range validation for *longitude*.

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

ReflectorLocations — Reflector locations

[10;10;0] (default) | 3-by-*N* numeric matrix

Reflector locations, specified as a 3-by-*N* numeric matrix containing the coordinates of the reflection points for the ray. *N* is the number of reflection points for the ray and is set by the NumReflections property. Each column represents the coordinate location of one reflection point along the propagation path from transmitter to receiver. The order of the columns is the same as the order of the points along the path. Columns (reflection point coordinates) are of one of these forms.

- [*x*; *y*; *z*] — when the CoordinateSystem property is set to 'Cartesian'. The object does not perform range validation for *x*, *y*, and *z*.
- [*latitude*; *longitude*; *height*] — when the CoordinateSystem property is set to 'Geographic'. *latitude* must be in the range [–90, 90], and *height* must be nonnegative. The object does not perform range validation for *longitude*.

Dependencies

To enable this property, set the PathSpecification property to 'Locations' and the LineOfSight property to 0 (false).

.

Data Types: double

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 ReflectionLocations.
- 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 `ReflectionLocations`.
- 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 `ReflectionLocations`.
- 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 `ReflectionLocations`.
- 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`

`NumReflections` — Number of reflection points

`0` (default) | nonnegative integer

This property is read-only.

Number of reflection points 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 ReflectionLocations.

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)	Plot rays in Site Viewer map

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.

```
viewer = siteviewer('Buildings','hongkong.osm');
```

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);
```

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,'Type','pathloss','ColorLimits',[100 250])
```

```
rays = 1×1 cell array
    {1×3 comm.Ray}
```

Display the properties of the first `comm.Ray` object. The `LineOfSight` property value is `1`, and the `NumReflections` 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: 4.6390

   Read-only properties:
       PropagationDelay: 4.6442e-07
    PropagationDistance: 139.2294
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
         NumReflections: 0
```

Display the properties of the second and third `comm.Ray` objects. The `LineOfSight` property values are `0`, and the `NumReflections` 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
    ReflectionLocations: [3×1 double]
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 106.1296
             PhaseShift: 3.5385

   Read-only properties:
       PropagationDelay: 4.6490e-07
    PropagationDistance: 139.3720
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
         NumReflections: 1
```

```
rays{1}(3)
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
    ReflectionLocations: [3×1 double]
              Frequency: 2.8000e+10
         PathLossSource: 'Custom'
               PathLoss: 146.9796
             PhaseShift: 3.5367

   Read-only properties:
       PropagationDelay: 1.1327e-06
    PropagationDistance: 339.5692
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
         NumReflections: 1
```

Visualize ray tracing results.

```
plot(rays{1});
```

Plot Propagation Rays Between Sites in Chicago

Return ray tracing results in `comm.Ray` objects and plot the ray propagation path after relaunching the Site Viewer map.

Create a Site Viewer map, loading building data for Chicago.

```
viewer = siteviewer('Buildings','chicago.osm');
```

Create and show a transmitter site on one building and a receiver site on another building.

```
tx = txsite('Latitude',41.8800,'Longitude',-87.6295, ...
    'TransmitterFrequency',2.5e9);
show(tx);
rx = rxsite('Latitude',41.881352,'Longitude',-87.629771, ...
    'AntennaHeight',30);
show(rx);
```

Perform ray tracing, returning the ray object results. For the configuration defined, ray tracing returns a cell array containing one ray object. Display the ray object properties. Then, close the Site Viewer map.

```
rays = raytrace(tx,rx)
```

```
rays = 1×1 cell array
    {1×1 comm.Ray}
```

```
rays{1}
```

```
ans = 
  Ray with properties:

      PathSpecification: 'Locations'
       CoordinateSystem: 'Geographic'
    TransmitterLocation: [3×1 double]
       ReceiverLocation: [3×1 double]
            LineOfSight: 0
    ReflectionLocations: [3×1 double]
              Frequency: 2.5000e+09
         PathLossSource: 'Custom'
               PathLoss: 95.4412
             PhaseShift: 4.4413

   Read-only properties:
       PropagationDelay: 5.7088e-07
    PropagationDistance: 171.1462
       AngleOfDeparture: [2×1 double]
         AngleOfArrival: [2×1 double]
         NumReflections: 1
```

```
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 one ray object at a time.

```
siteviewer('Buildings','chicago.osm');
show(tx);
show(rx);
plot(rays{1},'Type','power', ...
    'TransmitterSite',tx,'ReceiverSite',rx);
```

See Also

Functions
buildingMaterialPermittivity | earthSurfacePermittivity | propagationModel | raypl | raytrace

Objects
siteviewer

Introduced in R2020a

plot (rays)

Package: `comm`

Plot rays in Site Viewer map

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 path after relaunching the Site Viewer map.

Create a Site Viewer map, loading building data for Chicago.

```
viewer = siteviewer('Buildings','chicago.osm');
```

Create and show a transmitter site on one building and a receiver site on another building.

```
tx = txsite('Latitude',41.8800,'Longitude',-87.6295, ...
    'TransmitterFrequency',2.5e9);
show(tx);
rx = rxsite('Latitude',41.881352,'Longitude',-87.629771, ...
    'AntennaHeight',30);
show(rx);
```

Perform ray tracing, returning the ray object results. For the configuration defined, ray tracing returns a cell array containing one ray object. Display the ray object properties. Then, close the Site Viewer map.

```
rays = raytrace(tx,rx)
```

```
rays = 1×1 cell array
    {1×1 comm.Ray}
```

```
rays{1}
```

```
ans = 
  Ray with properties:
       PathSpecification: 'Locations'
        CoordinateSystem: 'Geographic'
     TransmitterLocation: [3×1 double]
        ReceiverLocation: [3×1 double]
             LineOfSight: 0
     ReflectionLocations: [3×1 double]
               Frequency: 2.5000e+09
          PathLossSource: 'Custom'
                PathLoss: 95.4412
              PhaseShift: 4.4413

    Read-only properties:
        PropagationDelay: 5.7088e-07
     PropagationDistance: 171.1462
        AngleOfDeparture: [2×1 double]
          AngleOfArrival: [2×1 double]
          NumReflections: 1
```

```
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 one ray object at a time.

```
siteviewer('Buildings','chicago.osm');
show(tx);
show(rx);
plot(rays{1},'Type','power', ...
    'TransmitterSite',tx,'ReceiverSite',rx);
```

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" and the CoordinateSystem property set to "Geographic".

Data Types: comm.Ray

Name-Value Pair Arguments

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

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. The default is the current siteviewer object, or if no Site Viewer is open a new siteviewer object opens.

Data Types: siteviewer object

See Also

Functions
raytrace

Objects
comm.Ray | siteviewer

Introduced in R2020a