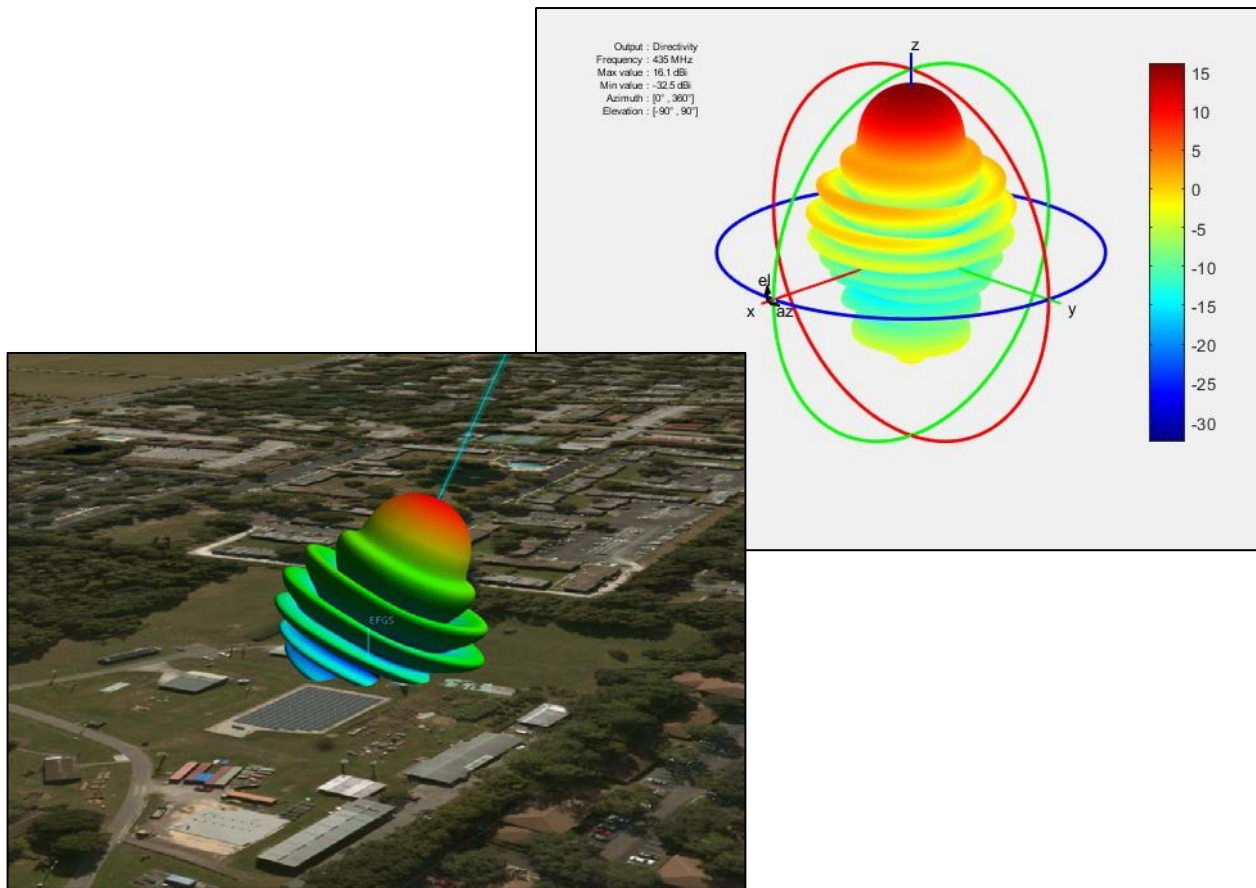


Modeling and Exporting Custom Antenna Radiation Patterns in MATLAB for Use in STK



Ansys Government Initiatives (AGI)

Table of Contents

Table of Contents	2
Table of Figures	3
Overview	5
Prerequisites	5
Modeling antennas in MATLAB Antenna Toolbox	5
The turnstile antenna	5
Creating two dipole objects	6
Modifying the dipole objects' properties	6
Orienting the two dipole objects	7
Combining the dipole objects into a conformal array	8
Generating a turnstile antenna radiation pattern	8
The Yagi-Uda antenna	11
Creating a yagiUda object	11
Modifying a yagiUda object's properties	12
Generating a yagiUda object's antenna radiation pattern	13
The cross-polarized Yagi-Uda antenna	15
Creating two yagiUda objects	15
Modifying the yagiUda objects' properties	16
Orienting the two yagiUda objects	18
Combining the yagiUda objects into a conformal array	18
Generating a cross-polarized Yagi-Uda antenna radiation pattern	19
More information and other antenna types.....	22
Formatting and exporting antenna radiation patterns for use in STK	22
External antenna pattern file format	22
Generating the correct antenna radiation pattern data	22
Setting up the external antenna pattern file	23
Writing the antenna radiation pattern data	24
Using an external antenna pattern file in STK	26
Importing into an antenna object	26
Importing into a transmitter or receiver object	26
Displaying the external antenna radiation pattern	26

Table of Figures

Figure 1 - Example of a turnstile antenna on a CubeSat	6
Figure 2 - Creating two dipole objects	6
Figure 3 - Modifying the dipole objects' properties	7
Figure 4 - Orienting the two dipole objects	8
Figure 5 - Combining the dipole objects into a conformal array	8
Figure 6 - Showing the visual representation of the turnstile antenna	9
Figure 7 - Visual representation of the turnstile antenna	9
Figure 8 - Generating the radiation pattern of the turnstile antenna	9
Figure 9 - Antenna radiation pattern of turnstile antenna	10
Figure 10 - Transforming turnstile antenna radiation pattern into variables	10
Figure 11 - Computing front-to-back ratio and beam widths of turnstile antenna	11
Figure 12 - Design of a typical Yagi-Uda antenna	11
Figure 13 - Creating a yagiUda object	11
Figure 14 - Modifying a yagiUda object's properties	12
Figure 15 - Visually displaying the yagiUda object	13
Figure 16 - Visual representation of the yagiUda object	13
Figure 17 - Generating the radiation pattern of the yagiUda object	14
Figure 18 - Antenna radiation pattern of yagiUda object	14
Figure 19 - Transforming yagiUda object's antenna radiation pattern into variables	14
Figure 20 - Computing front-to-back ratio and beam widths of yagiUda object	15
Figure 21 - Example cross-polarized Yagi-Uda antenna	15
Figure 22 - Creating two yagiUda objects	16
Figure 23 - Modifying the yagiUda objects' properties	17
Figure 24 - Orienting the two yagiUda objects	18
Figure 25 - Combining the yagiUda objects into a conformal array.....	19
Figure 26 - Showing the visual representation of the cross-polarized Yagi-Uda antenna	19
Figure 27 - Visual representation of the cross-polarized Yagi-Uda antenna	20
Figure 28 - Generating the radiation pattern of the yagiUda object	20
Figure 29 - Antenna radiation pattern of the cross-polarized Yagi-Uda antenna	21
Figure 30 - Transforming yagiUda object's antenna radiation pattern into variables	21
Figure 31 - Computing front-to-back ratio and beam widths of yagiUda object	21
Figure 32 - Defining the angle ranges	23

Figure 33 - Transforming an antenna radiation pattern into variables	23
Figure 34 - Creating and opening a new file for writing	23
Figure 35 - Writing the external antenna pattern file header	24
Figure 36 - Writing the antenna radiation pattern data	24
Figure 37 - Closing the external antenna pattern file	25
Figure 38 - The resulting external antenna pattern file.....	25

Overview

STK does not include a built-in antenna model for the turnstile or the Yagi-Uda antenna. These types of antennas are commonly used by universities and amateur radio enthusiasts as they are relatively inexpensive and operate well in amateur radio frequency bands such as UHF and VHF.

This knowledge article describes (1) how you can model these types of antennas in MATLAB using the MATLAB Antenna Toolbox and (2) how you can export these models and format them for use in STK to accurately simulate systems that use these types of antennas.

The antenna types explored in this knowledge article are the *turnstile*, *Yagi-Uda*, and *crosspolarized Yagi-Uda* antennas. The antennas modeled in this knowledge article are designed for UHF communications at a frequency of 435 MHz. This is a common frequency used for small satellite communications.

The commercial-off-the-shelf (COTS) Yagi-Uda antenna referenced in this document is the HyGain UB-7030SAT Cross-Polarized UHF Yagi-Uda antenna. For more information on this antenna see the [Hy-Gain PDF](#).

You can use the concepts explored in this article to model and export more complex, custom antennas and arrays for use in STK simulations.

Prerequisites

The following software packages and modules are required to complete this tutorial:

- AGI Systems Tool Kit (STK) 11.0.1 or newer
- MathWorks MATLAB R2016a or newer
- MathWorks MATLAB Antenna Toolbox R2016a or newer

For information on evaluation and licensing for STK, click [here](#).

For information on evaluation and licensing for MathWorks MATLAB, click [here](#).

Information on evaluation and licensing for MATLAB Antenna Toolbox, click [here](#).

Modeling antennas in MATLAB Antenna Toolbox

The turnstile antenna

The turnstile antenna consists of a set of two identical dipole antennas aligned at right angles to one another and fed in a phase quadrature. That is, the currents applied to each dipole are 90 degrees out of phase with one another. Figure 1 shows an example turnstile antenna mounted on a CubeSat satellite platform.

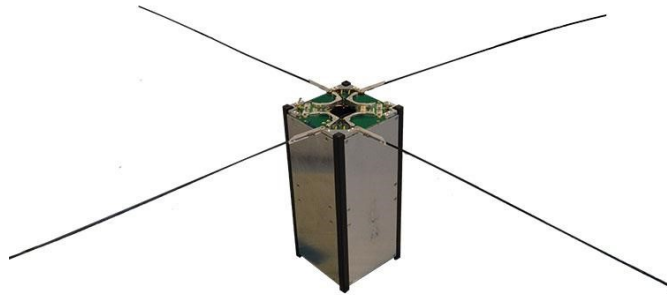


Figure 1 - Example of a turnstile antenna on a CubeSat

Creating two dipole objects

To create a turnstile antenna in MATLAB Antenna Toolbox, you must create the two *dipole* objects that make up the antenna. This process is shown in Figure 2.

```
% create the first dipole object
dipoleAntenna1 = dipole;

% create the next dipole object
dipoleAntenna2 = dipole;
```

Figure 2 - Creating two dipole objects

The code in Figure 2 creates two new *dipole* objects in MATLAB that contain different properties that define the size and orientation of the antenna. You can modify these settings to your specifications as shown in Figure 3.

Modifying the dipole objects' properties

Next, you must modify the design parameters of the two *dipole* objects to create an antenna tuned to the 435 MHz frequency. Using classical equations and relationships, a list of design variables is established and used to modify the properties of the two *dipole* objects. This is shown in Figure 3.

The value $3e8$ represents the speed of light. This is used to calculate the wavelength (λ) given the desired frequency.

```
% set up design variables for analysis
frequency = 435e6;
lambda = 3e8/frequency;
length = lambda/2.1;
width = lambda/50;

% set the length of the first dipole
dipoleAntenna1.Length = length;

% set the width of the first dipole
dipoleAntenna1.Width = width;

% set the length of the second dipole
dipoleAntenna2.Length = length;

% set the width of the second dipole
dipoleAntenna2.Width = width;
```

Figure 3 - Modifying the dipole objects' properties

Orienting the two dipole objects

With the design parameters of the two *dipole* objects established, you must orient the two objects at right angles to one another to achieve the desired turnstile antenna configuration. The *Tilt* and *TiltAxis* properties of the *dipole* object in MATLAB are used to orient the two elements. This is shown in Figure 4.

The *TiltAxis* property is an array of characters that describes which axis (X, Y, or Z) the object should be tilted on. Then, the *Tilt* property is an array of degree values (-180 to 180) that describes how much to tilt the object on each axis.

The order of values in the *Tilt* and *TiltAxis* properties must match up. In this case, the first entry in the *Tilt* property corresponds to the 'Y' value in the *Tilt Axis* property.

The tilt on the Y axis is set to -90 degrees to orient each dipole in the same plane. The tilt on the Z axis is set to -45 degrees for the first dipole and 45 degrees for the second dipole to create a crossed orientation that orients the two dipoles at right angles to each other.

The resulting orientation due to these tilt values is shown in Figure 7.

```

% set the tilt of the first dipole
dipoleAntenna1.Tilt = [-90,-45];

% set the tilt axis of the first dipole
dipoleAntenna1.TiltAxis = ['Y','Z'];

% set the tilt of the second dipole
dipoleAntenna2.Tilt = [-90,45];

% set the tilt axis of the firstdipole
dipoleAntenna2.TiltAxis = ['Y','Z'];

```

Figure 4 - Orienting the two dipole objects

Combining the dipole objects into a conformal array

To combine the two *dipole* objects into a single antenna, use the *conformal array* object. Each of the two dipoles is represented as an element in the array. The two dipoles are slightly offset from one another, to avoid a mesh collision, and they are also phase shifted to model the phase quadrature of a turnstile antenna. This is shown in Figure 5.

```

% create a conformal array to represent the composite antenna
turnstileAntenna = conformalArray;

% put the ground plane at the origin (it includes the first dipole offset)
turnstileAntenna.ElementPosition(1,:) = [0 0 0];

% put the second dipole offset from the ground plane
turnstileAntenna.ElementPosition(2,:) = [0 0 offset];
% fill the conformal array with the antennas
turnstileAntenna.Element = {dipoleAntenna1, dipoleAntenna2};
% apply the appropriate 90 deg phase shift to model the antenna
turnstileAntenna.PhaseShift = [0 90];

```

Figure 5 - Combining the dipole objects into a conformal array

Generating a turnstile antenna radiation pattern

Use the *show* method to display the visual representation of the resulting turnstile antenna. The syntax is shown in Figure 6.


```
% show the turnstile antenna
show(turnstileAntenna);
```

Figure 6 - Showing the visual representation of the turnstile antenna

The resulting figure is shown in Figure 7.

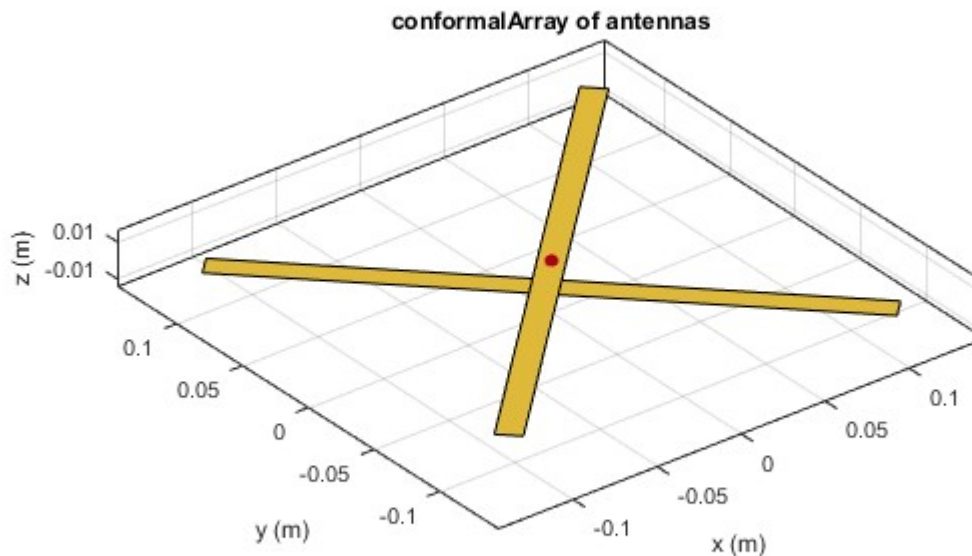


Figure 7 - Visual representation of the turnstile antenna

To generate the antenna radiation pattern of the resulting turnstile antenna, use the MATLAB Antenna Toolbox *pattern* method. The syntax for the *pattern* method is shown in Figure 8.

```
% set up variables for analysis
azimuth_range = 0:1:360;
elevation_range = -90:1:90;

% show the antenna radiation pattern at 435 MHz
pattern(turnstileAntenna, frequency, azimuth_range, elevation_range);
```

Figure 8 - Generating the radiation pattern of the turnstile antenna

The variables *frequency*, *azimuth_range*, and *elevation_range* are defined to allow easy modification of the *pattern* method usage. The *frequency* is set to 435 MHz. The *azimuth_range* variable would normally be set to $-180:1:180$; however, STK interprets this type of data for an azimuth range of $0:1:360$. This is described in more detail later.

The resulting antenna radiation pattern is shown in Figure 9.

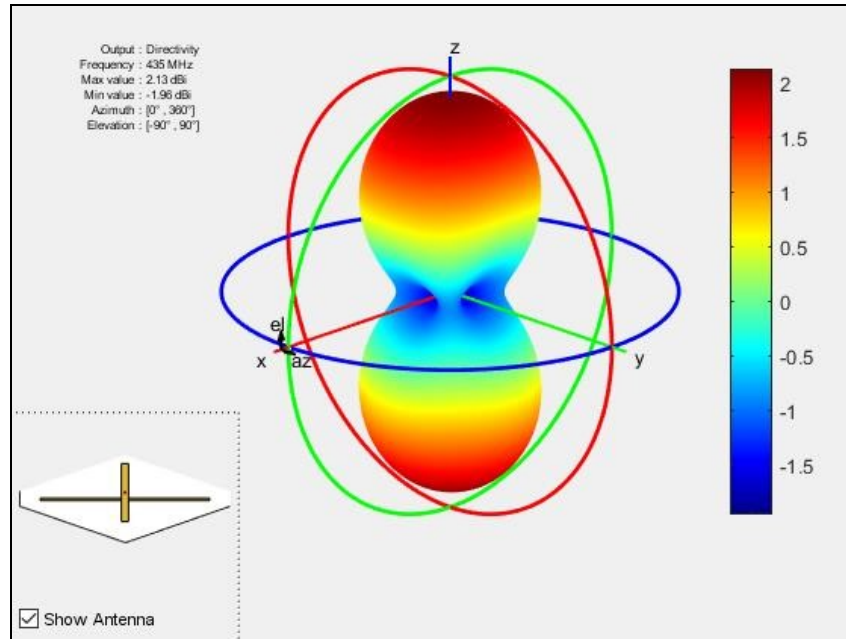


Figure 9 - Antenna radiation pattern of turnstile antenna

You can transform the radiation pattern for the antenna into a set of variables. In this case, the radiation pattern is split into the variables *gain*, *phi*, and *theta*. This is shown in Figure 10.

```
% transform the radiation pattern into variables
[ gain, phi, theta ] = pattern(yagiAntenna, frequency, azimuth_range, elevation_range);
```

Figure 10 - Transforming turnstile antenna radiation pattern into variables

Finally, you can calculate the antenna front-to-back ratio and beam widths as shown in Figure 11.

```

% computer the front-to-back-ratio d_max =
pattern(yagiAntenna,frequency,0,90); d_back =
pattern(yagiAntenna,frequency,0,-90);
fb_ratio = d_max - d_back

% computer the eplane and hplane beamwidths
eplane_beamwidth = beamwidth(yagiAntenna,frequency,0,1:1:360)
hplane_beamwidth = beamwidth(yagiAntenna,frequency,90,1:1:360)

```

Figure 11 - Computing front-to-back ratio and beam widths of a turnstile antenna

The Yagi-Uda Antenna

The basic Yagi-Uda antenna consists of a boom, a driven element (or exciter), a reflector, and a series of directors of decreasing size to direct the beam. Figure 12 shows the design of a typical Yagi-Uda antenna.

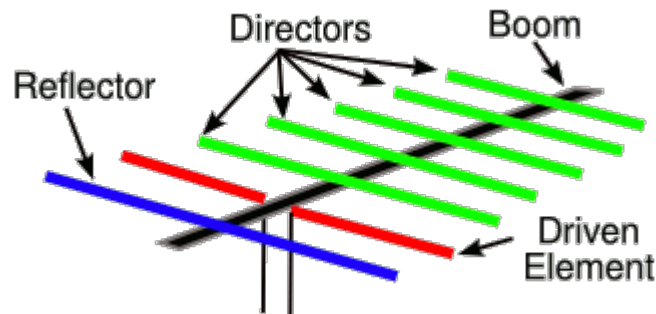


Figure 12 - Design of a typical Yagi-Uda antenna

Creating a yagiUda Object

Using MATLAB with MATLAB Antenna Toolbox installed, model the Yagi-Uda antenna by first creating a *yagiUda* object. This is shown in Figure 13.

```

% create a yagiUda object called "yagiAntenna"
yagiAntenna = yagiUda;

```

Figure 13 - Creating a yagiUda object

The code in Figure 13 creates a new *yagiUda* object in MATLAB that contains different properties that define the size and orientation of the antenna's exciter, reflector, and directors.

Modifying a yagiUda Object's Properties

You can modify the different properties of the antenna, such as the number of directors, the director lengths, and the reflector sizing, using the syntax shown in Figure 14.

Use a *dipoleFolded* object to represent the antenna's exciter. This is a commonly used type of exciter and is essentially a dipole that is folded to create a loop or rectangle. The *dipoleFolded* object has its own properties that you can modify in MATLAB. These properties are also shown in Figure 14.

The values in Figure 14 were taken from the COTS Yagi-Uda antenna mentioned in the overview.

```
% set the exciter of the antenna to a dipoleFolded object
yagiAntenna.Exciter = dipoleFolded;

% set the width of the exciter
yagiAntenna.Exciter.Width = 0.0120;

% set the spacing of the exciter
yagiAntenna.Exciter.Spacing = 0.0061;

% set the length of the exciter (311 mm from the datasheet)
yagiAntenna.Exciter.Length = 0.311;

% set the number of directors (13 from the datasheet)
yagiAntenna.NumDirectors = 13;

% set the director lengths (D1 - D13 from the datasheet)
yagiAntenna.DirectorLength = [0.298 0.298 0.295 0.286 0.284 0.281 ...
    0.278 0.275 0.275 0.275 0.275 0.275 0.275];

% set the director spacing (8.375 in from the datasheet)
yagiAntenna.DirectorSpacing = 0.212045;

% set the reflector length (330 mm from the datasheet)
yagiAntenna.ReflectorLength = 0.330;

% set the reflector spacing (136.75 mm from the datasheet)
yagiAntenna.ReflectorSpacing = 0.13675;
```

Figure 14 - Modifying a yagiUda object's properties

The property *DirectorLength* must be an array of lengths with a size equal to the number of directors specified by the *NumDirectors* property. This array defines the lengths for each of the directors on the Yagi-Uda antenna.

Generating a yagiUda object's antenna radiation pattern

With the properties of the *yagiUda* object specified, the model of the Yagi-Uda antenna is fully defined. You can display a visual representation of the model in MATLAB using the *show* command. The syntax of the *show* command is shown in Figure 15.

```
% show the Yagi-Uda antenna  
show(yagiAntenna);
```

Figure 15 - Visually Displaying the yagiUda object

The resulting figure is shown in Figure 16.

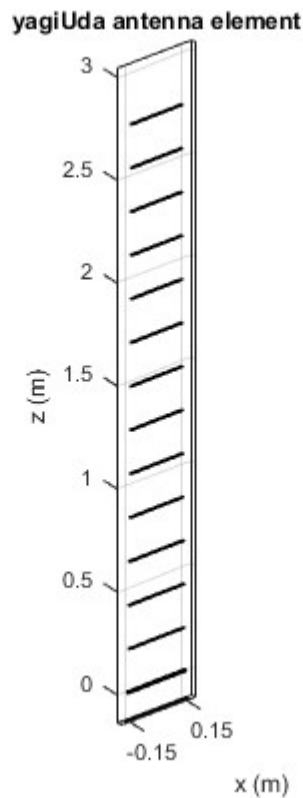


Figure 16 - Visual Representation of the yagiUda Object

To generate the antenna radiation pattern of the *yagiUda* object, use the MATLAB Antenna Toolbox *pattern* method. The syntax for the *pattern* method is shown in Figure 17.

```

% set up variables for analysis
frequency = 435e6;
azimuth_range = 0:1:360;
elevation_range = -90:1:90;

% show the antenna radiation pattern at 435 MHz
pattern(yagiAntenna, frequency, azimuth_range, elevation_range);

```

Figure 17 - Generating the radiation pattern of the yagiUda object

The variables *frequency*, *azimuth_range*, and *elevation_range* are defined to allow easy modification of the *pattern* method usage. The *frequency* is set to 435 MHz. The *azimuth_range* variable would normally be set to *-180:1:180*; however, STK interprets this type of data for an azimuth range of *0:1:360*. This is described in more detail later.

The resulting antenna radiation pattern is shown in Figure 18.

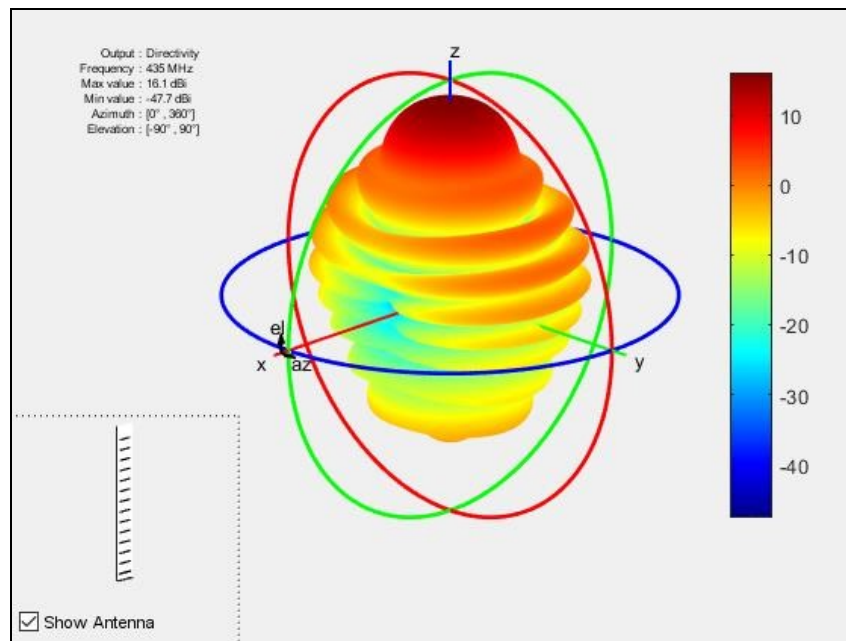


Figure 18 - Antenna radiation pattern of yagiUda object

You can transform the radiation pattern for the antenna into a set of variables. In this case, the radiation pattern is split into the variables *gain*, *phi*, and *theta*. This is shown in Figure 19.

```

% transform the radiation pattern into variables
[ gain, phi, theta ] = pattern(yagiAntenna, frequency, azimuth_range, elevation_range);

```

Figure 19 – Transforming the yagiUda object’s antenna radiation pattern into variables

Finally, you can calculate the antenna front-to-back ratio and beam widths as shown in Figure 20.

```
% compute the front-to-back-ratio
d_max = pattern(yagiAntenna,frequency,0,90);
d_back = pattern(yagiAntenna,frequency,0,-
90); fb_ratio = d_max - d_back

% compute the eplane and hplane beamwidths
eplane_beamwidth = beamwidth(yagiAntenna,frequency,0,1:1:360)
hplane_beamwidth = beamwidth(yagiAntenna,frequency,90,1:1:360)
```

Figure 20 - Computing front-to-back ratio and beam widths of the yagiUda object

The cross-polarized Yagi-Uda antenna

Similar to the turnstile antenna, the cross-polarized Yagi-Uda antenna is a combination of two individual antennas. The cross-polarized Yagi-Uda antenna consists of two identical Yagi-Uda antennas oriented at right-angles and slightly offset to one another. Cross-polarized Yagi-Uda antennas are commonly circularly polarized and thus provide both vertical and horizontal polarization for communication purposes. An example cross-polarized Yagi-Uda antenna is shown in Figure 21.



Figure 21 - Example cross-polarized Yagi-Uda antenna

Creating two yagiUda objects

To create a cross-polarized Yagi-Uda antenna, you must create two individual *yagiUda* objects. This process is shown in Figure 22.

```
% create a yagiUda object called "yagiAntenna1"
yagiAntenna1 = yagiUda;

% create a yagiUda object called "yagiAntenna2"
yagiAntenna2 = yagiUda;
```

Figure 22 - Creating two yagiUda objects

The code in Figure 22 creates two new *yagiUda* objects in MATLAB that contain different properties that define the size and orientation of each antenna's exciter, reflector, and directors.

Modifying the yagiUda Objects' properties

You can modify the different properties of each *yagiUda* antenna, such as the number of directors, the director lengths, and the reflector sizing, using the syntax shown in Figure 23.

Use a *dipoleFolded* object to represent each antenna's exciter. This is a commonly used type of exciter and is essentially a dipole that is folded to create a loop or rectangle. The *dipoleFolded* object has its own properties that you can modify in MATLAB. These properties are also shown in Figure 23.

The values in Figure 23 were taken from the COTS Yagi-Uda antenna mentioned in the overview.


```

% set the exciters of the antennas to a dipoleFolded object
yagiAntenna1.Exciter = dipoleFolded;
yagiAntenna2.Exciter = dipoleFolded;

% set the width of the exciters
yagiAntenna1.Exciter.Width = 0.0120;
yagiAntenna2.Exciter.Width = 0.0120;

% set the spacings of the exciters
yagiAntenna1.Exciter.Spacing = 0.0061;
yagiAntenna2.Exciter.Spacing = 0.0061;

% set the lengths of the exciters (311 mm from the datasheet)
yagiAntenna1.Exciter.Length = 0.311;
yagiAntenna2.Exciter.Length = 0.311;

% set the number of directors (13 from the datasheet)
yagiAntenna1.NumDirectors = 13;
yagiAntenna2.NumDirectors = 13;

% set the director lengths (D1 - D13 from the datasheet)
yagiAntenna1.DirectorLength = [0.298 0.298 0.295 0.286 0.284 0.281 ...
    0.278 0.275 0.275 0.275 0.275 0.275 0.275];
yagiAntenna2.DirectorLength = [0.298 0.298 0.295 0.286 0.284 0.281 ...
    0.278 0.275 0.275 0.275 0.275 0.275 0.275];

% set the director spacings (8.375 in from the datasheet)
yagiAntenna1.DirectorSpacing = 0.212045;
yagiAntenna2.DirectorSpacing = 0.212045;

% set the reflector lengths (330 mm from the datasheet)
yagiAntenna1.ReflectorLength = 0.330;
yagiAntenna2.ReflectorLength = 0.330;

% set the reflector spacings (136.75 mm from the datasheet)
yagiAntenna1.ReflectorSpacing = 0.13675;
yagiAntenna2.ReflectorSpacing = 0.13675;

```

Figure 23 – Modifying the yagiUda objects' properties

The property *DirectorLength* must be an array of lengths with a size equal to the number of directors specified by the *NumDirectors* property. This array defines the lengths for each of the directors on the Yagi-Uda antenna.

Orienting the two yagiUda objects

With the design parameters of the two *yagiUda* objects established, the two objects must be oriented at right angles to one another to achieve the desired turnstile antenna configuration. The *Tilt* and *TiltAxis* properties of the *dipole* object in MATLAB are used to orient the two elements. This is shown in Figure 24.

The *TiltAxis* property is an array of characters that describes which axis (X, Y, or Z) the object should be tilted on. The *Tilt* property is an array of degree values (-180 to 180) that describes how much to tilt the object on each axis.

The tilt on the Z axis is set to 45 degrees for the first Yagi-Uda antenna and -45 degrees for the second Yagi-Uda antenna. This creates a resulting 'X' pattern with the two antennas where they are 90 degrees offset from one another.

```
% set the tilt of the first antenna
yagiAntenna1.Tilt = 45;

% set the tilt axis of the first antenna
yagiAntenna1.TiltAxis = 'Z';

% set the tilt of the second antenna
yagiAntenna2.Tilt = -45;

% set the tilt axis of the second
antenna yagiAntenna2.TiltAxis = 'Z';
```

Figure 24 - Orienting the two yagiUda objects

The resulting orientation due to these tilt values is shown later in Figure 27.

Combining the yagiUda objects into a conformal array

To combine the two *yagiUda* objects into a single antenna, use the *conformal array* object. Each of the two *yagiUda* objects are represented as an element in the array. The two objects are slightly offset from one another, to avoid a mesh collision, and they are also phase-shifted to model the phase quadrature of a cross-polarized antenna. This is shown in Figure 25.

```

% create a conformal array to represent the composite antenna
crossPolYagiAntenna = conformalArray;

% fill the conformal array with the antennas
crossPolYagiAntenna.Element = [yagiAntenna1 yagiAntenna2];
% put the first element at the origin
crossPolYagiAntenna.ElementPosition(1,:) = [0 0 0];

% offset the antenna in the other plane (0.17145 m from
datasheet) crossPolYagiAntenna.ElementPosition(2,:) = [0 0
0.17145];
% apply a 90 degree phase shift between the two
antennas crossPolYagiAntenna.PhaseShift = [0 90];

```

Figure 25 - Combining the yagiUda objects into a conformal array

The third value in the *yagiAntenna.ElementPosition(2,:)* array is *0.17145*, which is the offset value of the two antennas per the datasheet of the antenna.

Generating a cross-polarized Yagi-Uda antenna radiation pattern

Use the *show* method to display the visual representation of the resulting turnstile antenna. The syntax is shown in Figure 26.

```

% show the Cross-Polarized Yagi-Uda
antenna show(crossPolYagiAntenna);

```

Figure 26 - Showing the visual representation of the cross-polarized Yagi-Uda antenna

The resulting figure is shown in Figure 27.

conformalArray of yagiUda antennas

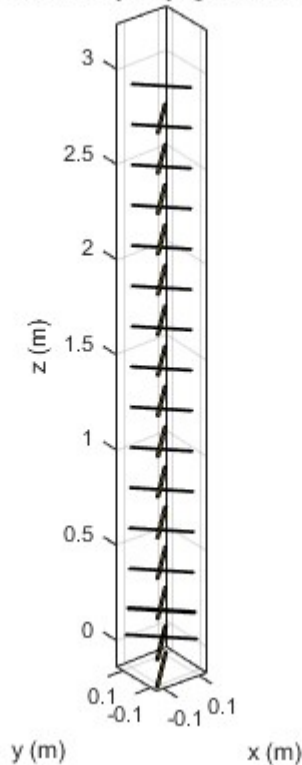


Figure 27 - Visual representation of the cross-polarized Yagi-Uda antenna

To generate the antenna radiation pattern of the cross-polarized Yagi-Uda antenna, use the Antenna Toolbox *pattern* method. The syntax for the *pattern* method is shown in Figure 28.

```
% setup variables for analysis
frequency = 435e6;
azimuth_range = 0:1:360;
elevation_range = -90:1:90;

% show the antenna radiation pattern at 435 MHz
pattern(crossPolYagiAntenna, frequency, azimuth_range, elevation_range);
```

Figure 28 - Generating the radiation pattern of the yagiUda object

The variables *frequency*, *azimuth_range*, and *elevation_range* are defined to allow easy modification of the *pattern* method usage. The *frequency* is set to 435 MHz. The *azimuth_range* variable would normally be set to *-180:1:180*; however, STK interprets this type of data for an azimuth range of *0:1:360*. This is described in more detail later.

The resulting antenna radiation pattern is shown in Figure 29.

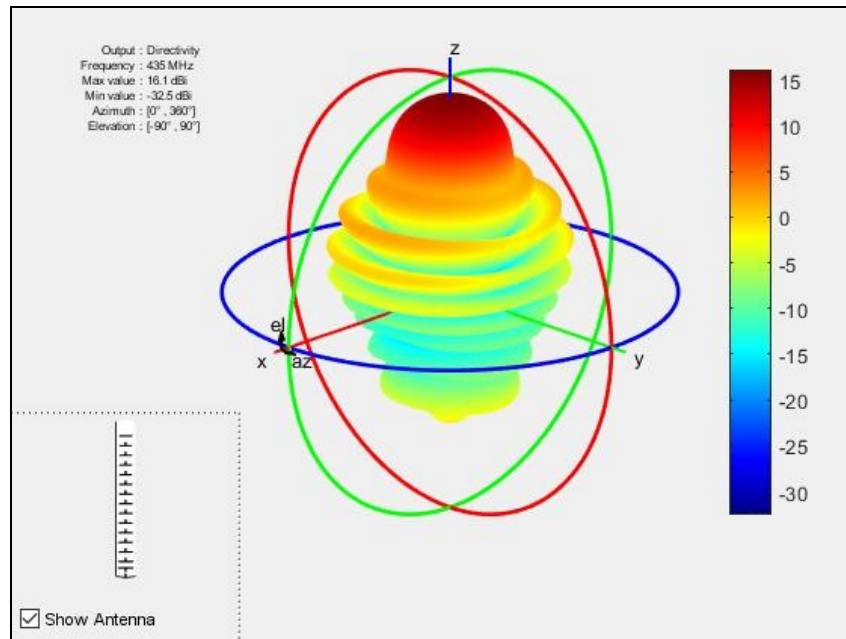


Figure 29 - Antenna radiation pattern of the cross-polarized Yagi-Uda antenna

You can transform the radiation pattern for the antenna into a set of variables. In this case, the radiation pattern is split into the variables *gain*, *phi*, and *theta*. This is shown in Figure 30.

```
% transform the radiation pattern into variables
[ gain, phi, theta ] = pattern(crossPolYagiAntenna, frequency, azimuth_range, elevation_range);
```

Figure 30 – Transforming the yagiUda Object’s antenna radiation pattern into variables

Finally, calculate the antenna front-to-back ratio and beam widths as shown in Figure 31.

```
% computer the front-to-back-ratio
d_max = pattern(crossPolYagiAntenna, frequency, 0, 90);
d_back = pattern(crossPolYagiAntenna, frequency, 0, -90);
fb_ratio = d_max - d_back

% computer the eplane and hplane beamwidths
eplane_beamwidth = beamwidth(crossPolYagiAntenna, frequency, 0, 1:1:360)
hplane_beamwidth = beamwidth(crossPolYagiAntenna, frequency, 90, 1:1:360)
```

Figure 31 - Computing front-to-back ratio and beam widths of yagiUda object

More information and other antenna types

[Help documentation on the MATLAB Antenna Toolbox](#)

[Different types of antennas that you can model in MATLAB](#)

[A more detailed tutorial on modeling turnstile antennas in MATLAB](#)

[More detailed information on modeling Yagi-Uda antennas in MATLAB](#)

[A detailed tutorial on Yagi-Uda antenna optimization](#)

Formatting and Exporting Antenna Radiation Patterns for Use in STK

External Antenna Pattern File Format

When generating an antenna radiation pattern to be used in STK, the azimuth and elevation ranges used in MATLAB must match the ranges required by STK. You can get more information on the [formatting of external antenna pattern files](#) in the STK Help documentation.

There are different types of external antenna pattern files that you can generate. These include *PhiThetaPattern*, *ThetaPhiPattern*, *AzElPattern*, *ElAzPattern*, *SymmetricPattern*, and *IEEE1979*.

Considering MATLAB Antenna Toolbox defines its ranges in terms of *azimuth* and *elevation*, it may seem obvious that you should use either the *AzElPattern* or the *ElAzPattern* type of file. However, the figure shown when the *pattern* method is used is actually expressed in *Phi* and *Theta* terms. Therefore, when formatting and exporting antenna radiation patterns from MATLAB, you should use the *PhiThetaPattern* or *ThetaPhiPattern* type of file.

The order of the words in *PhiThetaPattern* and *ThetaPhiPattern* determines the column order of the data in each file. That is, in the *PhiThetaPattern*, the first column of data corresponds to the *Phi* angles, and the second column of data corresponds to the *Theta* angles. For the *ThetaPhiPattern*, the first column of data corresponds to the *Theta* angles, and the second column of data corresponds to the *Phi* angles. The third column in both of these formats corresponds to the antenna gain at each *Phi* and *Theta* angle.

This document describes formatting and exporting antenna radiation pattern data in the *PhiThetaPattern* format. See the [example file](#) using this format.

Generating the Correct Antenna Radiation Pattern Data

After having created a model of the desired antenna in MATLAB, the *frequency*, *azimuth_range*, and *elevation_range* variables must be defined for use with the *pattern* method. This is shown in Figure 32.

```
% set up variables for analysis
frequency = 435e6;
azimuth_range = 0:1:360;
elevation_range = -90:1:90;
```

Figure 32 - Defining the angle ranges

Per the STK documentation on the *PhiThetaPattern* format described previously, the range of the *Phi* angles (*azimuth_range*) is set to *0:1:360* resulting in 361 distinct angles. According to the STK documentation, the range of the *Theta* angles should be *0:1:180*; however, MATLAB needs a range of *-90:1:90* to generate the full antenna radiation pattern. For this reason, the variable *elevation_range* is set to *-90:1:90* to satisfy MATLAB's constraint. Later in the formatting process, the *elevation_range* angles are shifted by 90 degrees to satisfy the STK *PhiThetaPattern* formatting constraint.

Next, you must generate the antenna radiation pattern using the *pattern* method and transform the data into separate variables that MATLAB can use to loop through and export the data. The *pattern* method syntax used is shown in Figure 33.

```
% transform the radiation pattern into variables
[ gain, phi, theta ] = pattern(crossPolyYagiAntenna, frequency, azimuth_range, elevation_range);
```

Figure 33 - Transforming an antenna radiation pattern into variables

This example uses the cross-polarized Yagi-Uda antenna described earlier in this document. The syntax in Figure 33 produces three variables: *gain*, *phi*, and *theta*. The *phi* variable is an array of size *1x361*. The *theta* variable is an array of *1x181*. The *gain* variable is an array of size *181x361*. The *phi* and *theta* variables provide an array of all the angles. The *gain* variable provides the value of the antenna gain at each of these variables; hence its size is *181x361*.

Setting up the external antenna pattern file

To write to an external file in MATLAB, use the *fopen* method. In this case, a new file called *CrossPolyYagiAntennaPattern.txt* is opened for *writing*. The syntax used is shown in Figure 34.

```
% open a new file for writing
output_file = fopen('CrossPolyYagiAntennaPattern.txt', 'w');
```

Figure 34 - Creating and Opening a New File for Writing

The first parameter of the *fopen* method is the file path, including the name. The syntax shown in Figure 34 creates a new file in the same directory the MATLAB script is located in. The 'w' parameter in the *fopen* method denotes that the file being opened is being opened to write to.

Next, you must write the header of the external antenna pattern file. The header includes information such as the format of the file (in this case *PhiThetaPattern*), the units of the angles, the number of distinct data points, etc. The process of writing this header is shown in Figure 35.

The *NumberOfPoints* parameter is calculated by multiplying the number of rows by the number of columns for the *gain* variable in MATLAB. Thus, 181 rows and 361 columns results in 65341 distinct data points.

```
% write the external antenna pattern file header
fprintf(output_file, 'stk.v.11.1.0\n');
fprintf(output_file, 'PhiThetaPattern\n');
fprintf(output_file, 'AngleUnit Degrees\n');
fprintf(output_file, 'NumberOfPoints 65341\n');
fprintf(output_file, 'PatternData\n');
```

Figure 35 - Writing the external antenna pattern file header

Writing the antenna radiation pattern data

The text following *PatternData* in the external antenna pattern file is the actual antenna radiation pattern data, including the *Phi* angles, *Theta* angles, and antenna gain. In order to write this information, the data in the *gain*, *phi*, and *theta* variables in MATLAB are looped through and written to the external antenna pattern file using the *fprintf* method. This is shown in Figure 36.

```
% loop through the azimuth, elevation, and gain values and write to file for
i = 1:1:361
    k =
    181;
    for j =
    1:1:181
        if i == 361 &&
j == 181

            fprintf(output_file, '%f\t%f\t%f', phi(i), (theta(k) + 90), gain(j,i));

        else
            fprintf(output_file, '%f\t%f\t%f\n', phi(i), (theta(k) + 90), gain(j,i));

        end
        k
    = k - 1;
    end
end
```

Figure 36 - Writing the Antenna Radiation Pattern Data

The value `'\t'` inserts a tab using `fprintf`. This essentially creates three columns of data in the resulting file. Also, the value of `theta` is offset by 90 degrees when writing the angle to the pattern file. This solves the problem discussed earlier where MATLAB needs an elevation range of `-90:1:90` while the `PhiThetaPattern` format requires a range of `0:1:180`. Offsetting the `theta` value by 90 degrees solves this problem.

With the data written to the pattern file, you must close the file in MATLAB to apply the changes, using the `fclose` method. The syntax is shown in Figure 37.

```
% close the file
fclose(output_file);
```




Figure 37 - Closing the external antenna pattern file

The resulting external antenna pattern file looks like the example data shown in Figure 38. You can then import this file into STK, a process which is described in the next section.



```
stk.v.11.1.0
PhiThetaPattern
AngleUnit Degrees
NumberOfPoints 65341
PatternData
0.000000    180.000000 -0.452313 0.000000
    179.000000 -0.473423 0.000000
    178.000000 -0.536944 0.000000
    177.000000 -0.643566 0.000000
    176.000000 -0.794439 0.000000
    175.000000 -0.991178 0.000000
    174.000000 -1.235860 0.000000
    173.000000 -1.531005 0.000000
    172.000000 -1.879534 0.000000
    171.000000 -2.284668 0.000000
    170.000000 -2.749723 0.000000
    169.000000 -3.277710 0.000000
    168.000000 -3.870588 0.000000
    167.000000 -4.527892 0.000000
    166.000000 -5.244298 0.000000
    165.000000 -6.005516 0.000000
    164.000000 -6.781999
0.000000    163.000000 -7.521226
...
```

Figure 38 - The resulting external antenna pattern file






Using an External Antenna Pattern File in STK

In STK, you can use an external antenna pattern file on an Antenna object (), on a Transmitter object (), or on a Receiver object ().





Importing into an Antenna object

1. Right-click the Antenna () object and click Properties ().
2. In the Properties window, select the Definition page under the Basic category.
3. In the Type field, click the ellipsis (...) button.
4. In the Select Model window, select External Antenna Pattern and click OK.
5. In the Design Frequency field, enter the design frequency of the antenna, in this case 435 MHz or 0.435 GHz.
6. In the External Filename field, click the ellipses (...) button.
7. Browse to the location of the TXT file generated by the MATLAB script described previously; then click Open.
8. Click Apply to apply the changes and click OK to close the Properties window.

Importing into a Transmitter or Receiver object

1. Right-click the Transmitter () or Receiver () object and click Properties ().
2. In the Properties window, select the Definition page under the Basic category.
3. In the Type field, click the ellipsis (...) button.
4. If the object is a Transmitter (), select Complex Transmitter Model and click OK. If the object is a Receiver (), select Complex Receiver Model and click OK.
5. Select the Antenna tab. Under the Model Specs tab, in the Type field, click the ellipsis (...) button.
6. In the Select Model window, select External Antenna Pattern and click OK.
7. In the Design Frequency field, enter the design frequency of the antenna, in this case 435 MHz or 0.435 GHz.
8. In the External Filename field, click the ellipsis (...) button.
9. Browse to the location of the TXT file generated by the MATLAB script described previously; then click Open.
10. Click Apply to apply the changes and click OK to close the Properties window.

Displaying the external antenna radiation pattern

1. Right-click the Antenna (), Transmitter (), or Receiver () object and click Properties ().
2. In the Properties window, select the Attributes page under the 3D Graphics category.
3. In the Volume Graphics section, select the Show Volume check box. You can modify the volume settings to adjust the size and resolution of the volume representation of the antenna radiation pattern.
4. Click Apply to apply the changes and click OK to close the Properties window.
5. Return to the 3D Graphics Window. Zoom to the object that the Antenna, Transmitter, or Receiver is attached to. You will see the 3D representation of the radiation pattern.