Demo: Computing and Displaying Antenna Patterns

In this demo, we will illustrate some basic MATLAB tools for computing and displaying antenna patterns. Specifically, you will learn to:

- Define simple antennas using MATLAB's antenna toolbox
- Plot antenna patterns in 2D and 3D
- Compute free-space path loss
- Use the antenna patterns and free-space path loss functions to compute the path loss along a trajectory

Simulation constants

For the remainder of the demo, we will use the following simulation constants

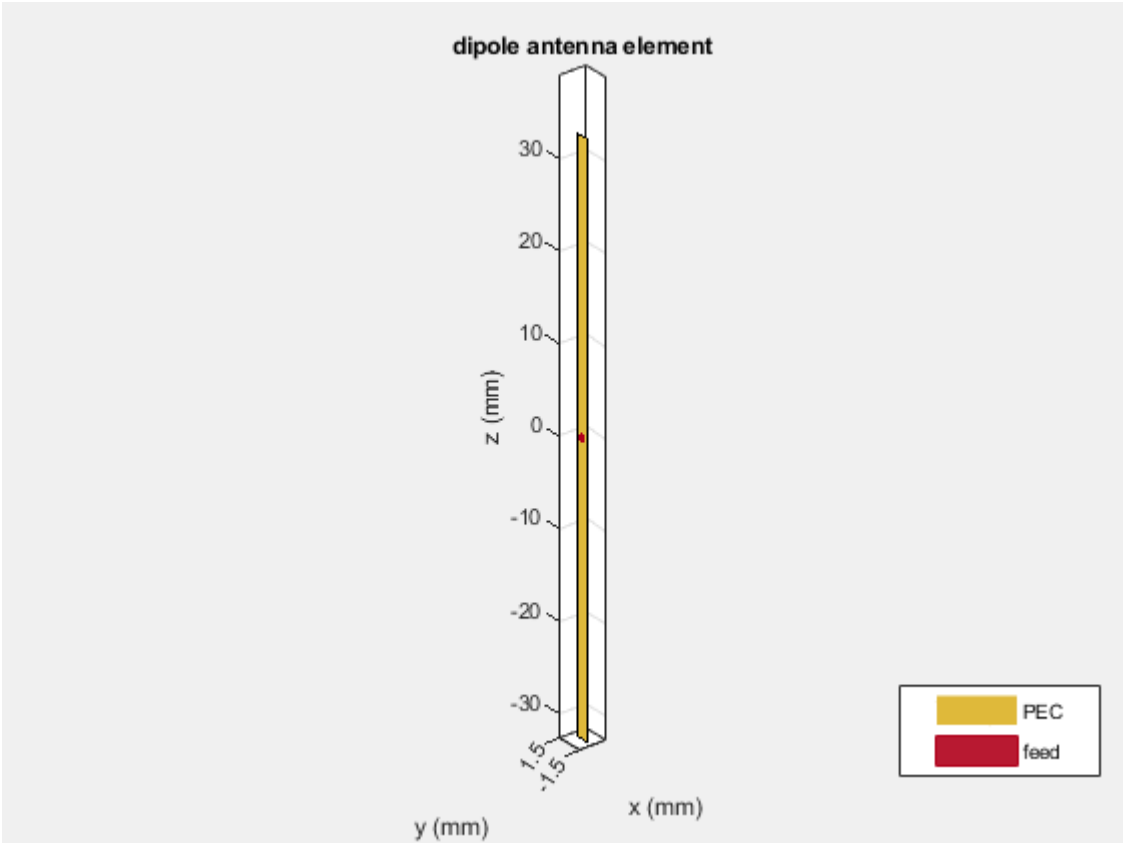
Note: In MATLAB, all values are in metric units m, s, Hz, etc. Not GHz or MHz.

```
fc = 2.3e9;     % Carrier frequency
vp = physconst('lightspeed'); % speed of light
lambda = vp/fc;   % wavelength
```

Dipole antenna

For a first antenna, we construct a simple dipole. Note that this command will require the antennas toolbox.

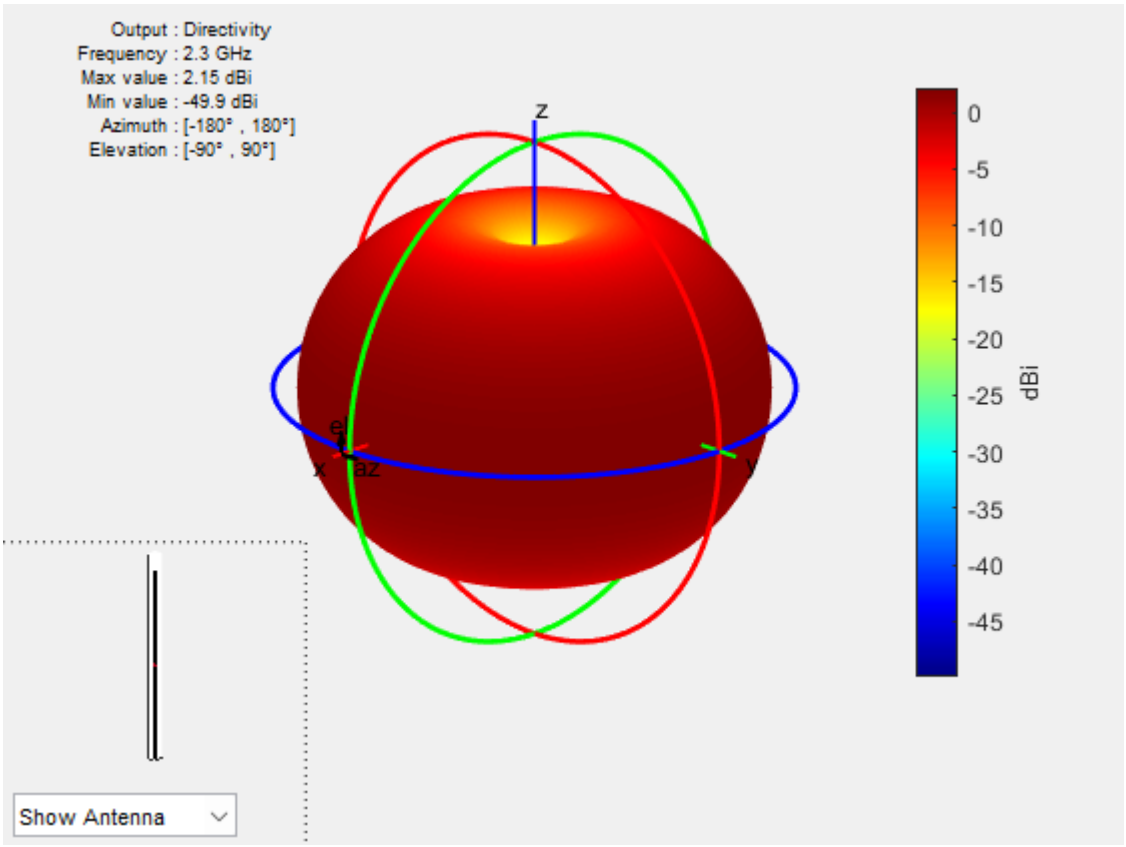
```
% Construct the antenna object
ant = dipole(...
    'Length', lambda/2,...
    'Width', 0.01*lambda );

% Display the antenna
ant.show();
```



Displaying the pattern

We can display the antenna pattern with the following command.

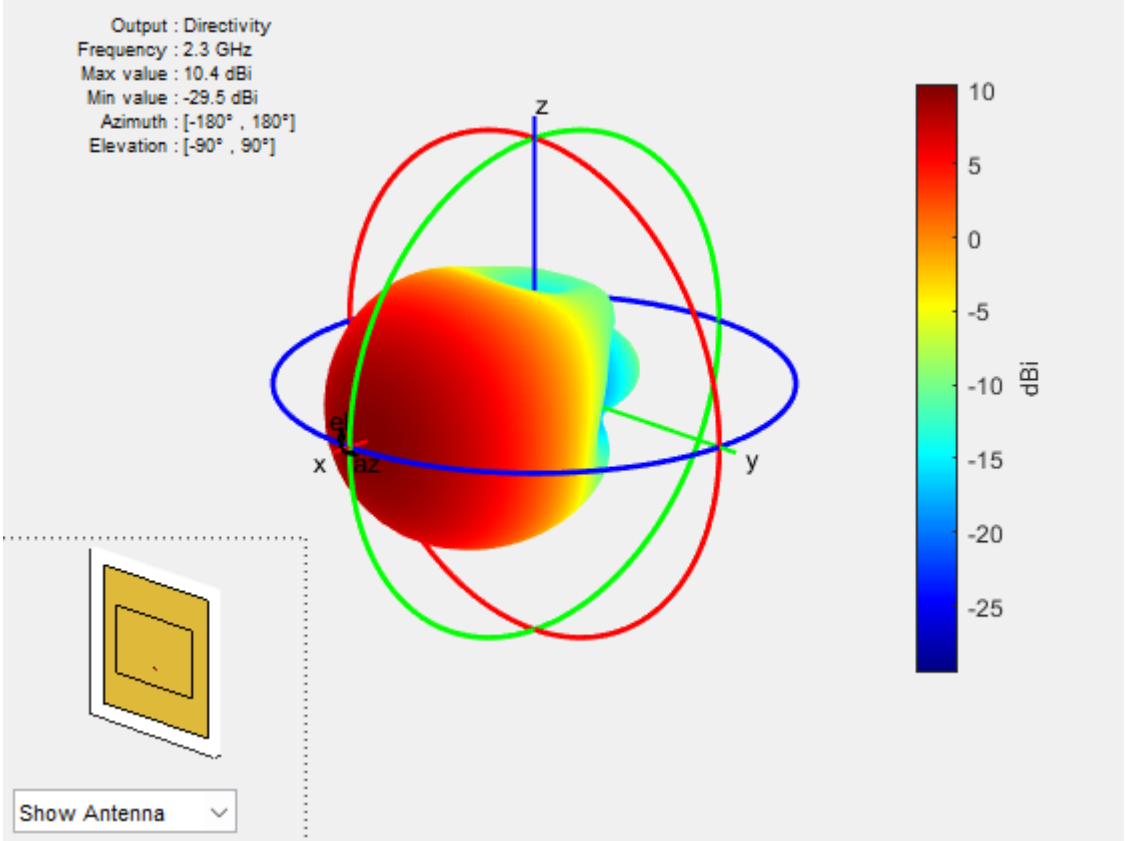
```
ant.pattern(fc);
```



Patch Element

We now consider a more complex antenna. The antenna toolbox can analyze a number of antennas in use. However, once the antenna is more complex, you will start to notice that the analysis becomes very slow.

```
len = 0.49*lambda;
groundPlaneLen = lambda;
ant2 = patchMicrostrip(...
    'Length', len, 'Width', 1.5*len, ...
    'GroundPlaneLength', groundPlaneLen, ...
    'GroundPlaneWidth', groundPlaneLen, ...
    'Height', 0.01*lambda, ...
    'FeedOffset', [0.25*len 0]);

% Tilt the element so that the maximum energy is in the x-axis
ant2.Tilt = 90;
ant2.TiltAxis = [0 1 0];

% Display the antenna pattern after rotation.
% This may take a few minutes.  So be patient
ant2.pattern(fc, 'Type', 'Directivity');
```

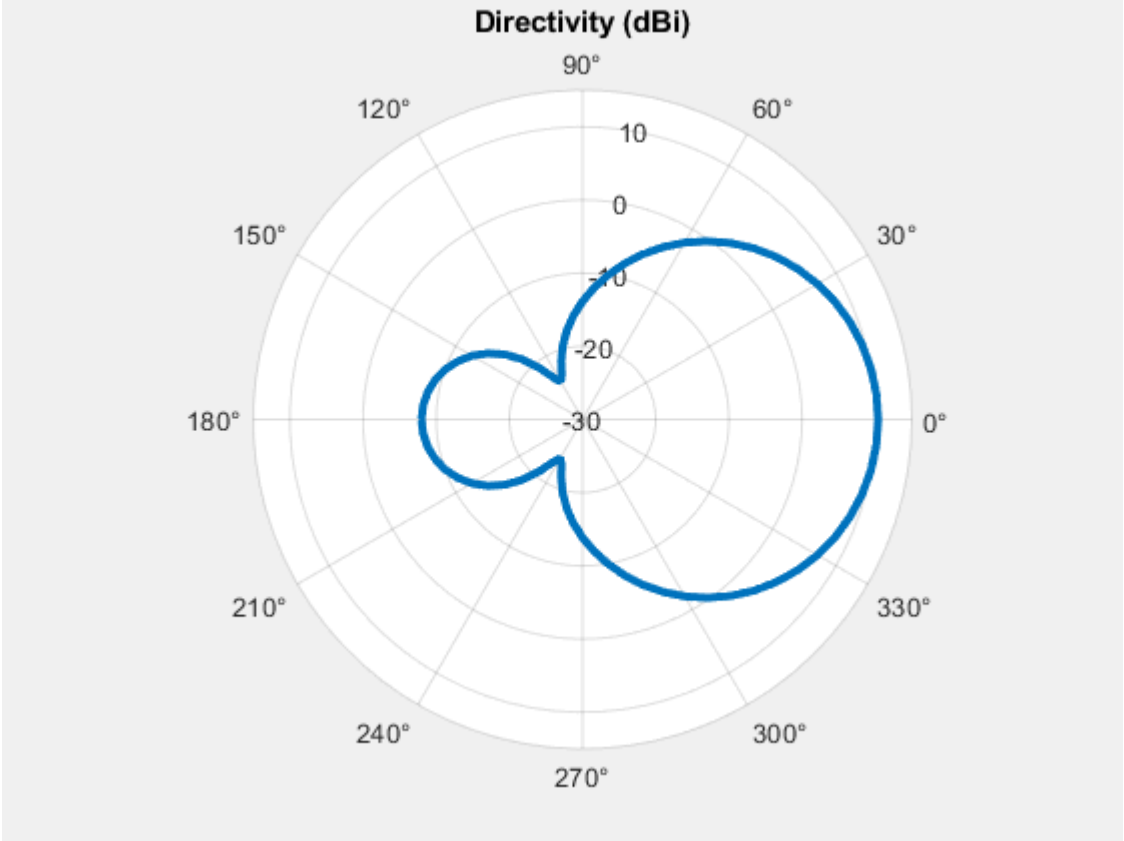



```
% You can also save the pattern
[dir,az,el] = ant2.pattern(fc, 'Type', 'Directivity');
```

Plotting a cross-section

Once the antenna pattern is stored in an array, you can plot cross sections as follows. Suppose we want to plot the cross-section at an elevation angle of 0

```
% Elevation angle to plot
elPlot = 0;

% Find the index closest to the desired angle
[~, iel] = min(abs(el - elPlot));

% Plot using the polar plot.
% Note the conversion to radians.  You also have to use the |rlim|
% command to set the limits.
polarplot(deg2rad(az), dir(iel,:),'LineWidth', 3);
rlim([-30, 15]);
title('Directivity (dBi)');
```



Computing free-space path loss

Suppose we want to compute the omni-directional free-space path loss, meaning the path loss without antenna gain at some distance d:

```
d = 500;  % distance in meters

% We can compute the FSPL manually from Friis' law
% Note the minus sign
plOmni1 = -20*log10(lambda/4/pi/d);

% Or, we can use MATLAB's built in function:
plOmni2 = fspl(d, lambda);

fprintf(1,'Omni PL - manual: %7.2f\n', plOmni1);
```

```
Omni PL - manual:   93.66
```

```
fprintf(1,'Omni PL - MATLAB: %7.2f\n', plOmni2);
```

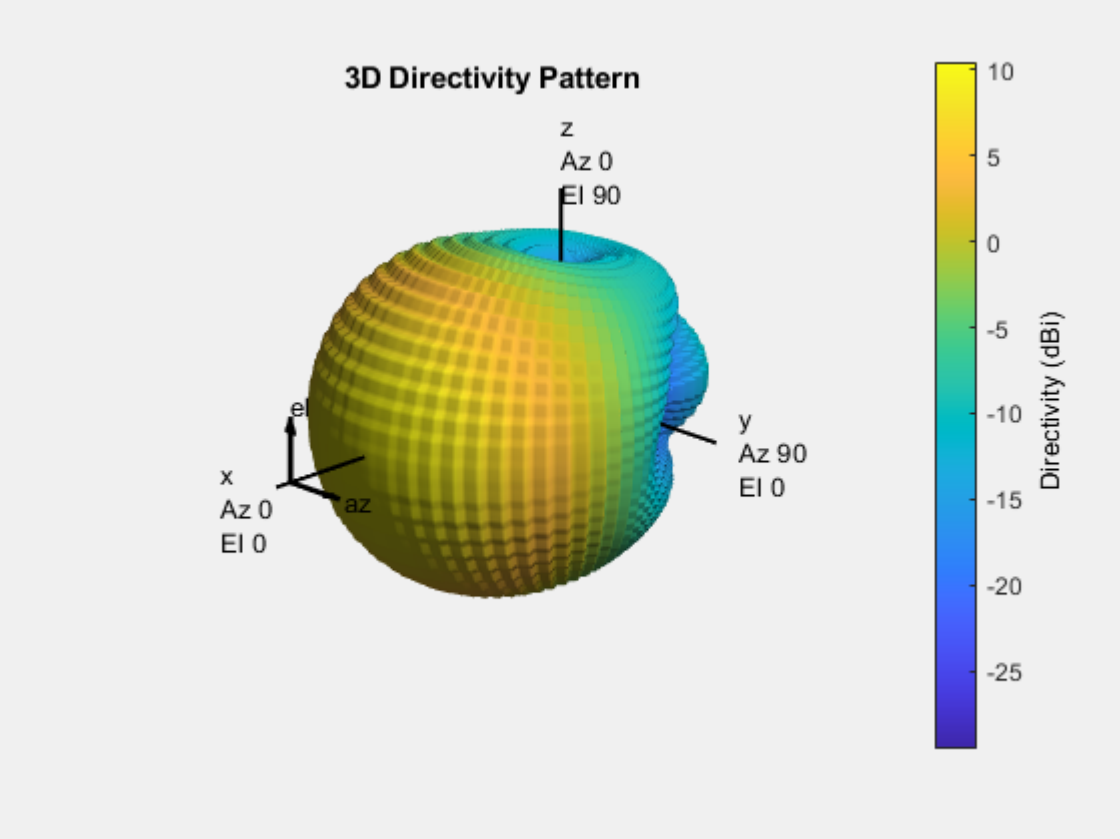
```
Omni PL - MATLAB:   93.66
```

Creating a custom antenna pattern

While MATLAB has many common antennas, you will often need to load antenna data from a manufacturer or other source. Also, even when using MATLAB's antenna elements, it is often useful to compute the pattern once

and store it. For this purpose, you can create a custom antenna element. Here, we will create a custom antenna element with directivity pattern we just computed from the microstrip element.

```
phasePattern = zeros(size(dir));
ant3 = phased.CustomAntennaElement(...
    'AzimuthAngles', az, 'ElevationAngles', el, ...
    'MagnitudePattern', dir, ...
    'PhasePattern', phasePattern);

% Plot the antenna pattern.
% Note the format is slightly different since we are using
% the pattern routine from the phased array toolbox
clf;
ant3.pattern(fc);
```

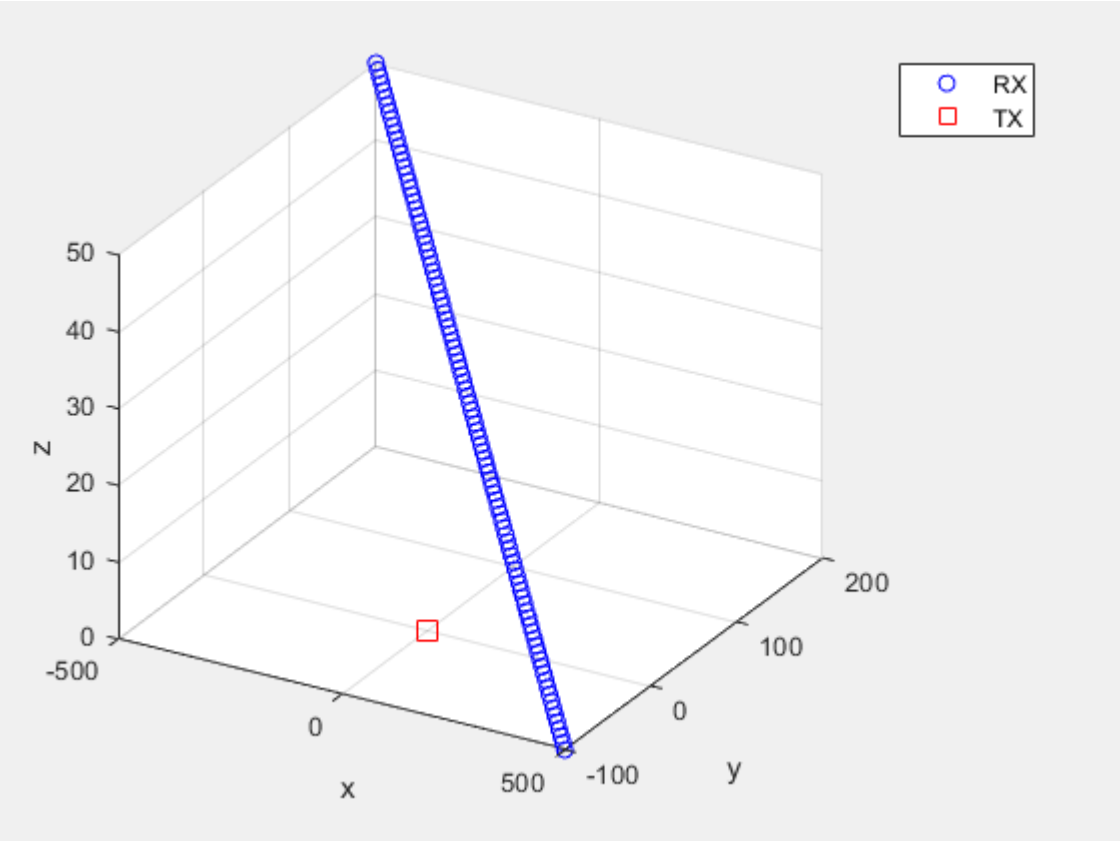


Interpolating the directivity in the custom pattern

Once we have the antenna pattern, we can interpolate the values of the gain at other directions. To illustrate we will plot the total path loss between a TX at the origin and an RX object traveling in a linear path along a 3D path. First, we create and plot the path

```
% Define the linear path
npts = 100;
xstart = [500 -100 0]';
xend = [-500 200 50]';
t = linspace(0,1,npts)';
```

```
% Compute points on the path
X = (1-t)*xstart' + t*xend';

% Plot the path
clf;
plot3(X(:,1), X(:,2), X(:,3), 'bo', 'DisplayName', 'RX');
hold on;
plot3(0, 0, 0, 'rs', 'MarkerSize',10, 'DisplayName', 'TX');
hold off;
grid on;
view(30,30);
xlabel('x');
ylabel('y');
zlabel('z');
legend();
```



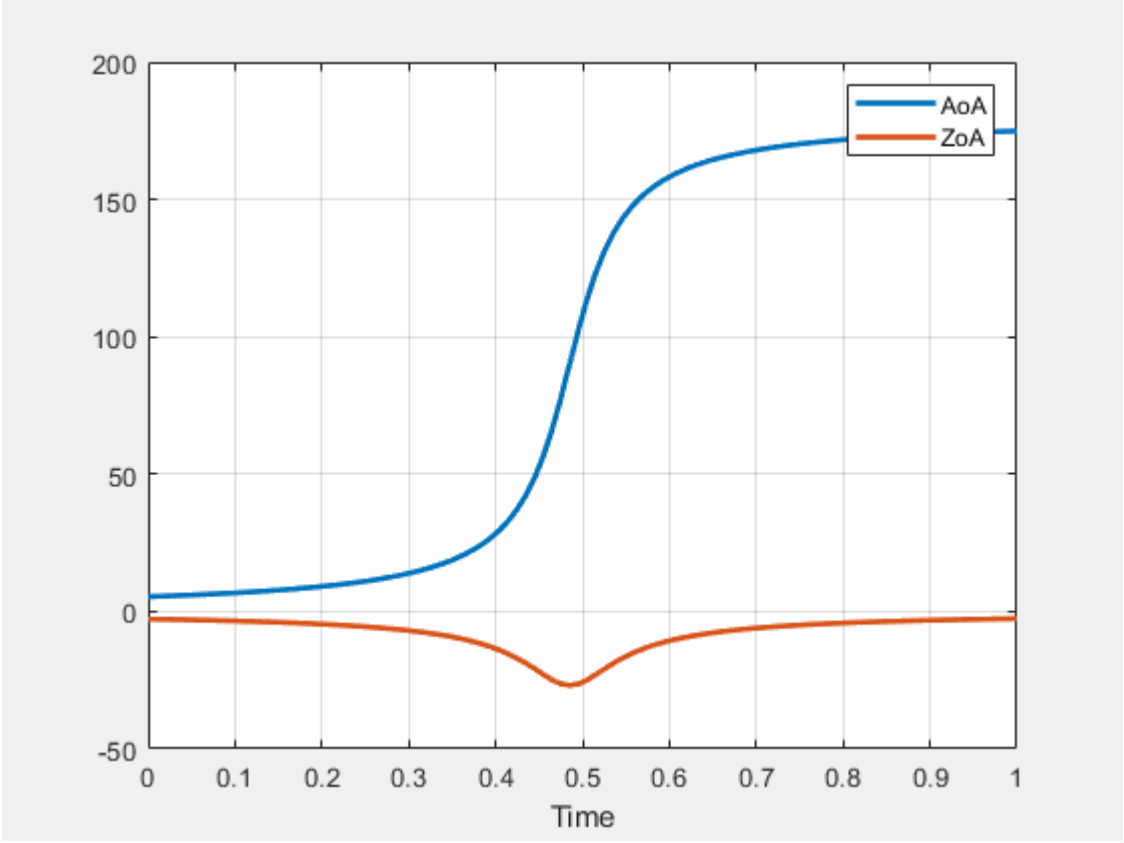
Compute a frame of reference with the x axis aligned on the direction of motion

```
% Get direction of motion
v = xend-xstart;

% Compute the angle of direction of motion
[azDir, elDir, ~] = cart2sph(v(1),v(2),v(3));

% Find a rotation matrix aligned to direction of motion
```

```
yaw = azDir;
pitch = -elDir;  % Note the negative sign since
roll = 0;
R = eul2rotm([yaw pitch roll], 'ZYX');
```

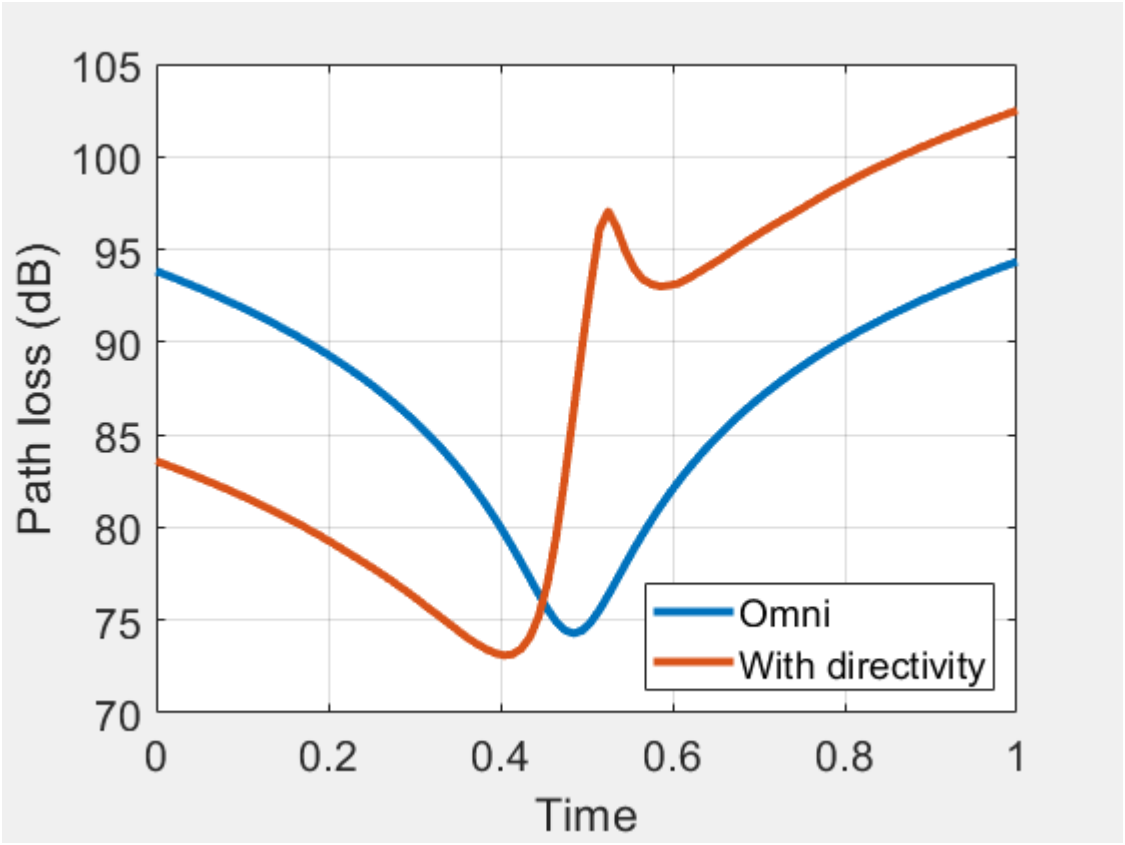
Find the angles of arrival from the TX to the RX in the RX frame of reference

```
% Create the vector from the local antenna to remote signal source
Zpath = -X;

% Rotate to the RX frame of reference
% Note:  No transpose since we are multiplying on the right
Zrot = Zpath*R;

% Compute angles in local frame of reference
[azpath, elpath, dist] = cart2sph(Zrot(:,1), Zrot(:,2), Zrot(:,3));

% Convert to degrees
azpath = rad2deg(azpath);
elpath = rad2deg(elpath);

% Plot the angles over time
plot(t, [azpath elpath], 'Linewidth', 2);
grid on;
legend('AoA', 'ZoA');
xlabel('Time');
```



Compute the free space omni-direcitonal path loss, gain and path loss with gain.

```
% Compute the free space path loss along the path without
% the antenna gain.  We can use MATLAB's built-in function
plOmni = fspl(dist, lambda);

% Compute the directivity using interpolation of the pattern.
% We can use the ant3.resp method for this purpose, but the
% interpolation is not smooth.  So, we will do this using
% MATLAB's interpolation objects.   First, we create the
% interpolation object.
F = griddedInterpolant({el,az},dir);

% Then, we compute the directivity using the object
dirPath = F(elpath,azpath);

% Compute the total path loss including the directivity
plDir = plOmni - dirPath;

% Plot the path loss over time.  Can you explain the
clf;
plot(t, [plOmni plDir], 'Linewidth', 3);
grid();
set(gca, 'Fontsize', 16);
legend('Omni', 'With directivity', 'Location', 'SouthEast');
xlabel('Time');
ylabel('Path loss (dB)');
```


105
100
95
90
85
80
75
70
Path loss (dB)
0
0.2
0.4
0.6
0.8
1
Time
Omni
With directivity