Project 3

Jordan Baxter, Chelsea Starr, and Ross Dewberry
Oregon State University, Corvallis, OR, USA

***Abstract*—The purpose of this paper is to introduce a student created program that simulates antenna radiation patterns and directivity. Simple dipole antennas are simulated as well as isotropic, perpendicular, and co-linear uniform linear arrays.**

I. Project Summary

Students learning about antennas may struggle with visualization of the electric and magnetic fields in space. Showing them a still image of a cross cut of an antenna here and there does not always provide sufficient conceptual understanding to facilitate the desired intuition. Using a graphical user interface (GUI) to allow students the ability to alter, move, and study single dipole and antenna array radiation patterns may help close this knowledge gap.

The course Antennas and Propagation is required to cover topics from a Hertzian dipole antenna to multi-antenna arrays. The requirement to quickly become familiar with so many unique radiators and physical interactions they may have is overwhelming in a 10 week course. The visualizations provided in literature and results of assignments does assist although the static representations leave room for improvement. As prospective engineers we have a desire to push, prod, and play with concepts until the mysteries are explored. With the material provided from the reference documentation and class notes we have all the information we need to implement a highly customizable tool that any student will know how to use and learn from.

The objective of this work is to deliver a GUI that can be used by any student in future classes. This GUI produces the radiation pattern, directivity, and a 3D plot of the radiation pattern. The radiation pattern will be in both the E-plane and the H-plane. The directivity will be available in the horizontal plane. The user can alter these outputs by changing the inputs on the GUI. Variations include antenna length, number of elements, separation, orientation, and excitation.

The source code is provided to assist students in future developments programming in Python.

II. Simulation Options

The program simulates the 2D normalized radiation pattern as well as directivity for both single dipole antennas and various linear antenna arrays. In addition, an interactive 3D plot of the normalized radiation pattern is generated.

For the single dipole, antenna lengths ranging from that of a Hertzian dipole to $\frac{l}{\lambda} = 1.75$ can be simulated. The uniform linear antenna array simulation generates plots for isotropic, co-linear dipole, and perpendicular dipole elements. Antenna lengths, spacing, number of elements, and excitation phasing are all user configurable.

III. User Interface

The GUI can be programmed to run on Linux, Windows, and MacOS to allow ease of use. The plots can be updated in real time, allowing the user to instantly compare different antenna lengths and array configurations. For example, a student could add elements and instantly visualize the increase in directivity.

IV. Calculations

A. Single Dipole

For the single dipole, $F_{un}(\theta)$ was defined in lecture. It was derived from the summation of Hertzian dipoles, using far-field approximations.

$$F_{un}(\theta) = \frac{cos(k_o \frac{l}{2} cos(\theta)) - cos(k_o \frac{l}{2}))}{sin(\theta)} \text{ where } k_o = \frac{2\pi}{\lambda_o}$$

For the normalized radiation pattern, the function $F_{un}(\theta)$ is divided by it's maximum value.

$$F(\theta) = \frac{F_{un}(\theta)}{F_{max}}.$$

For the program calculations, it is assumed that the dipole lies along the Z axis and θ is measured as the angle from the Z axis. $F(\theta)$ gives the pattern in planes that include the Z axis. In other words, $F_{un}(\theta)$ shows the E-plane pattern. For a dipole with this orientation, the H-plane pattern lies XY plane and is uniform.

B. Uniform Linear Antenna Arrays

Although a single dipole is useful in some cases, it has a nearly omnidirectional pattern and relatively low directivitty. A higher gain can be achieved with an antenna array.

The principle of pattern multiplication results in the radiation pattern for an array being given by the product of the radiation pattern for an individual antenna and the antenna factor.

$$F_{AR}(\theta, \phi) = F(\theta, \phi) AF(\theta, \phi)$$

The program simulates uniform, linear arrays. These arrays consist of uniform elements, oriented in the same direction, equally spaced along a line, and driven by equal magnitude currents of a uniform progressive phase shift. The array factor for a uniform linear array is given in the lecture notes.

$$AF(\psi) = \frac{1}{N} \left| \frac{sin(N\frac{\psi}{2})}{sin(\frac{\psi}{2})} \right|$$
$$\text{where } \psi = kdcos(\gamma) + \Delta\phi$$

Variable ψ accounts for the spacing and phasing of the individual antennas. $\Delta\phi$ describes the phase shift, d is the

physical spacing (in terms of λ), and γ is the variable describing the angle between the array axis and the direction of observation. To represent γ in terms of θ and ϕ, the relationship $cos^{-1}(sin(\phi)sin(\theta))$ was derived (See Appendix A).

1) Isotropic Array: For an isotropic array, the individual normalized antenna pattern is is a sphere of radius 1.

$$F(\theta, \phi) = 1$$

Plotting the radiation pattern of an isotropic element array simplifies to plotting the antenna pattern.

$$F_{AR}(\theta, \phi) = F(\theta, \phi)AF(\theta, \phi)$$

$$F_{AR}(\theta, \phi) = AF(\theta, \phi)$$

To plot in 3 dimensions, the antenna factor is rewritten in terms θ and ϕ.

$$AF(\psi) = \frac{1}{N}\left|\frac{sin(N\frac{\psi}{2})}{sin(\frac{\psi}{2})}\right|$$

$$AF(\gamma) = \frac{1}{N}\left|\frac{sin(N(kdcos(\gamma)+\Delta\phi)\frac{1}{2})}{sin(kdcos(\gamma)+\Delta\phi)\frac{1}{2}}\right|$$

$$AF(\theta, \phi) = \frac{1}{N}\left|\frac{sin(N(kdcos(cos^{-1}(sin(\phi)sin(\theta)))+\Delta\phi)\frac{1}{2})}{sin(kdcos(cos^{-1}(sin(\phi)sin(\theta)))+\Delta\phi)\frac{1}{2}}\right|$$

$$AF(\theta, \phi) = \frac{1}{N}\left|\frac{sin(N(kd(sin(\phi)sin(\theta)))+\Delta\phi)\frac{1}{2})}{sin(kd(sin(\phi)sin(\theta)))+\Delta\phi)\frac{1}{2}}\right|$$

2) Co-linear Array: For a co-linear array, the array axis and individual element axes are the same. Therefore, the array factor is given as follows:

$$F_{AR}(\theta, \phi) = F(\theta)AF(\theta, \phi)$$

The antenna element factor, $F(\theta)$, is the array factor of a simple dipole antenna.

3) Perpendicular Array: For a perpendicular array, the axes of the individual elements are not same as the array axis. To calculate $F(\theta, \phi)$ we adjusted the angle relationship used to form a new basis. Appendix A shows the derivation of γ in detail.

C. Directivity

The same function is used to determine the directivity for all the antenna configurations. The GUI displays directivity in the E-Plane.

$$D(\theta, \phi) = \frac{4\pi}{\int_0^{2\pi}\int_0^{\pi} F(\theta,\phi)sin(\theta)d\theta,\phi}$$

V. Example Patterns

A. Single Dipole

1) Hertzian Dipole:

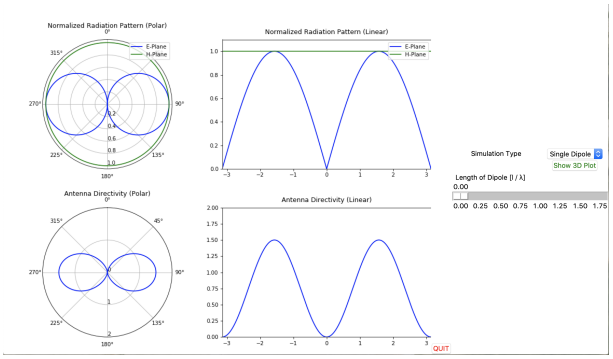


Figure 1: 2D Hertzian Dipole

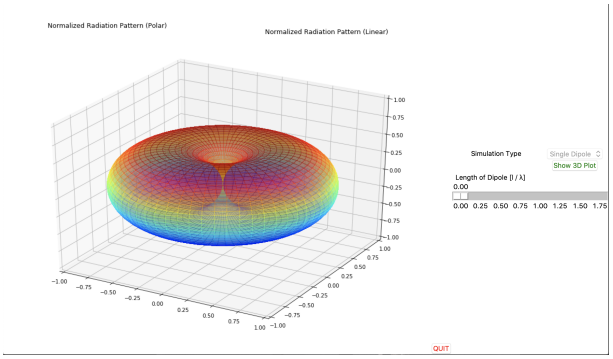


Figure 2: 3D Hertzian Dipole

2) Half Wave Dipole:

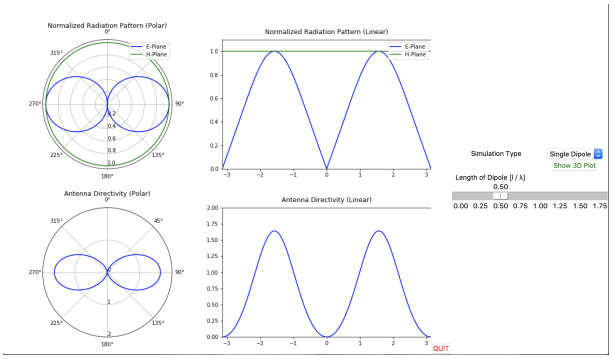


Figure 3: 2D Half Wave Dipole

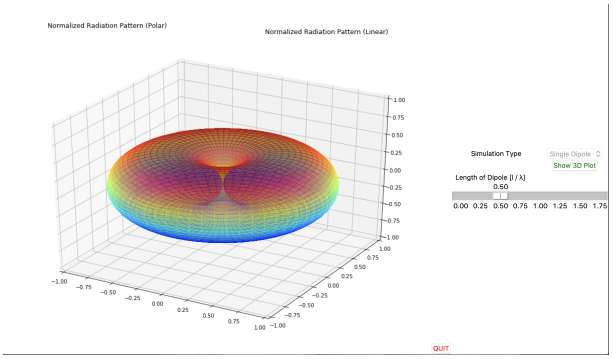


Figure 4: 3D Half Wave Dipole

3) $\frac{3}{2}$ Wave Dipole:

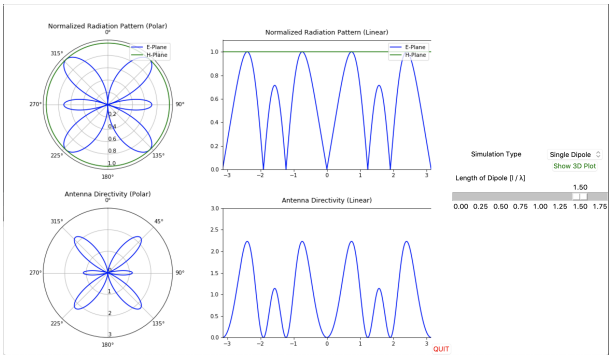


Figure 5: 2D $\frac{3}{2}$ Wave Dipole

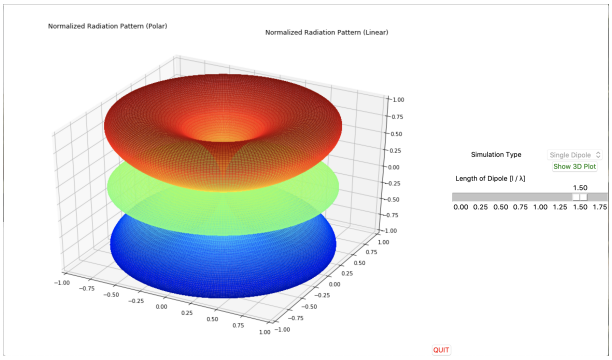


Figure 6: 3D $\frac{3}{2}$ Wave Dipole

B. Uniform Linear Antenna Arrays

To illustrate the usefulness of the program, endfire arrays for isotropic radiators, perpendicular dipoles, and parallel dipoles are given (similar to slide 185 of the class notes).

1) Isotropic: $N = 4$; $\frac{d}{\lambda} = 0.25$; $\Delta\psi = 90$
For the isotrpoic radiator, the configuration of the array can be taken into consideration without accounting for specific element interaction.

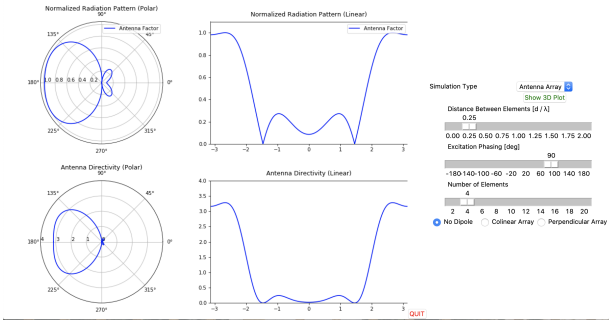


Figure 7: 2D Isotropic Array

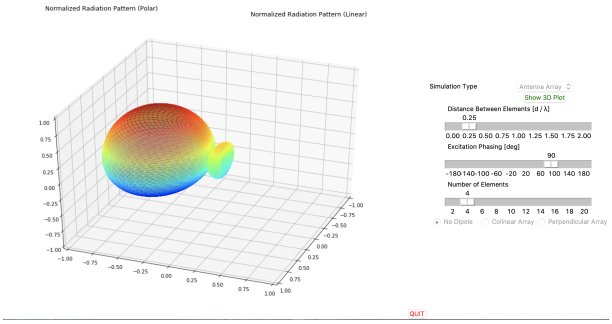


Figure 8: 3D Isotropic Array

2) Co-linear: $N = 4$; $\frac{d}{\lambda} = 0.25$; $\Delta\phi = 90$; $\frac{l}{\lambda} = 0.5$
For the Co-linear array, the maximum of the array factor, $AF(\theta, \phi)$, and the nulls of the individual elements, $F(\theta, \phi) = 0$. This results in an inefficient configuration, as shown by Figure 7.

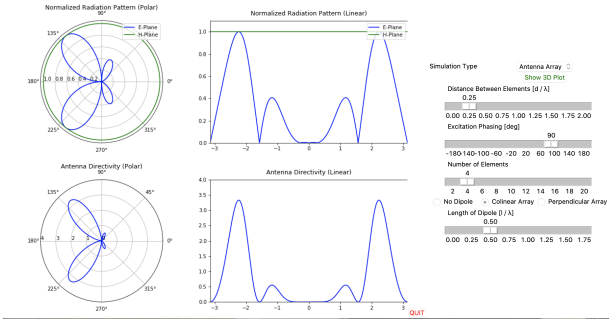


Figure 9: 2D Co-linear Array

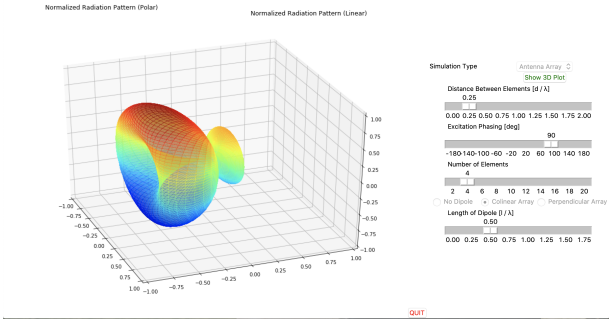


Figure 10: 3D Co-linear Array

3) Perpendicular: $N = 4$; $\frac{d}{\lambda} = 0.25$; $\Delta\phi = 90$; $\frac{l}{\lambda} = 0.5$
For the perpendicular array, the direction of maximum directivity for the individual elements and the array factor lie in the same direction. This configuration can be visualized with the figures below.

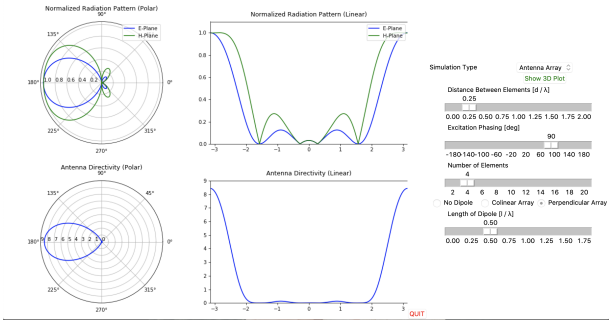


Figure 11: 2D Perpendicular Array

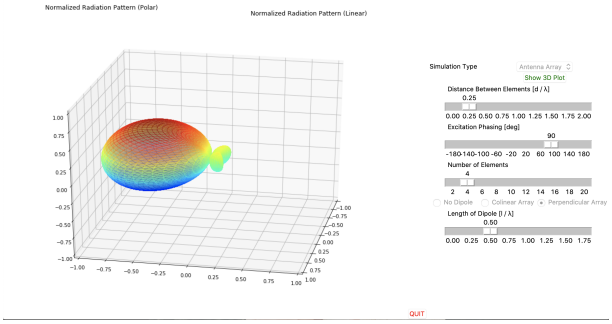


Figure 12: 3D Perpendicular Array

C. KOAC

To elaborate on the usefulness of the GUI, a model of the KOAC antennas is given. The KOAC specifications sheet lists the 2 element monopole array with an antenna height as 325.5' and the spacing at 70.6'. The array axis is defined with a bearing of 115°, but the simulation does not specify cardinal orientation, only orientation of the array axis relative to element orientation. At a frequency of 550KHz, $\frac{d}{\lambda} = 0.37$ and $\frac{l}{\lambda} = 0.36$.

1) Daytime: According to the KOAC specifications sheet, the current phasing of the antennas is 50°. The daytime configuration is non-uniform, as the current excitations are 1.0 and 0.89. Regardless, the program gives a quick approximation of the radiation pattern and directivity.

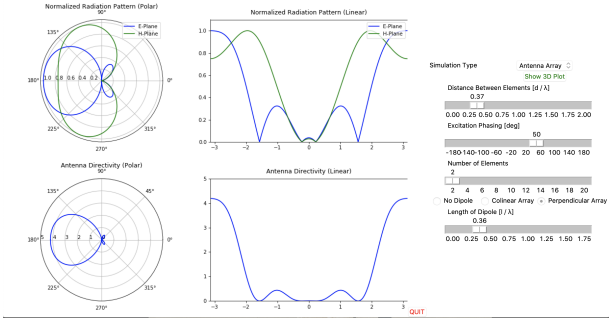


Figure 13: KOAC Day 2D Approximation

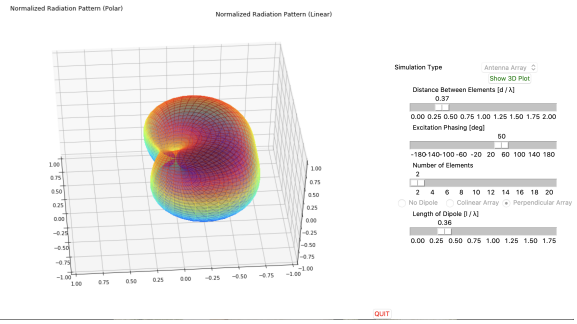


Figure 14: KOAC Day 3D Approximation

2) Nightime: The nightime configuration is a uniform linear array, which can be represented accurately with the program. The current phasing is -15°.

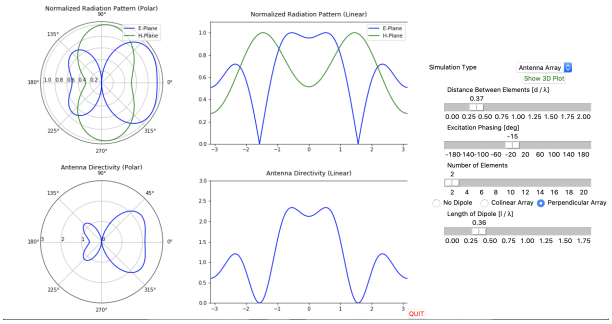


Figure 15: KOAC Night 2D

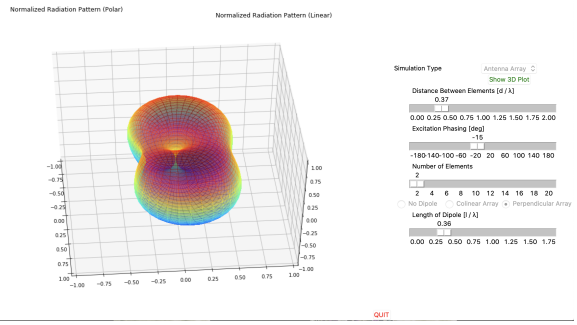


Figure 16: KOAC Day 3D

VI. Antenna Designs

Using only the interactive program we were able to quickly understand approximately what antenna array would best perform the project 3 performance requirements. The limitations of the program in choosing different excitation current values does not allow for perfect antenna array design. As you will see the outputs do set us up in the correct direction and would assist future students in gaining this insight during the initial project design steps.

A. Endfire

The first project design challenge was an endfire array of linear half wave dipoles. As we saw earlier the use of a perpendicular array is ideal for this design. We also need to maintain a 3 dB beam width of $< 40°$ and side lobe levels of $< 25\%$. We will be able to achieve these levels by including a larger number of elements and tuning the element spacing.

Finally we use an excitation phasing of -60° to adjust the main beam into the endfire position.

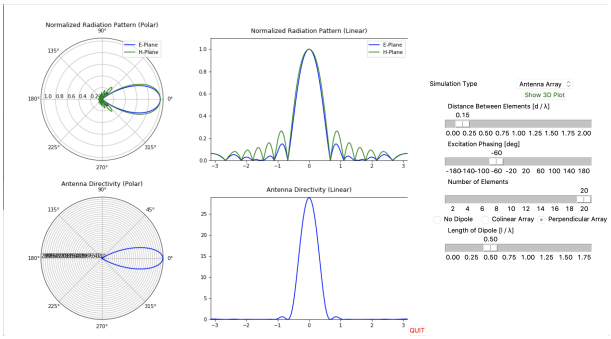


Figure 17: Endfire 2D

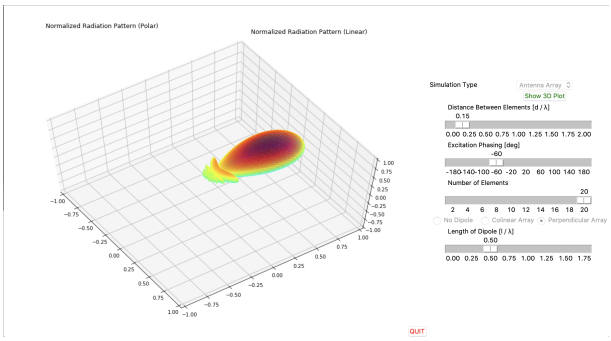


Figure 18: Endfire 3D

B. Broadside

The second project design challenge was a broadside array of linear dipoles. As we saw earlier the use of a collinear array is ideal for this design. We also need to maintain a 3 dB beam width of $< 12°$ and side lobe levels of $< 20\%$. We will be able to achieve these levels by including a larger number of elements and tuning the element spacing. Finally we use an excitation phasing of 0° to adjust the angle the broadside fires the adjustment can be made here. We found a wavelength dipole with a wavelength spacing produced the highest directivity while minimizing side lobes.

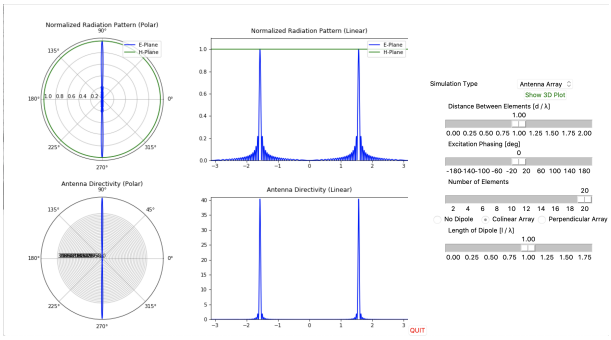


Figure 19: Broadside 2D

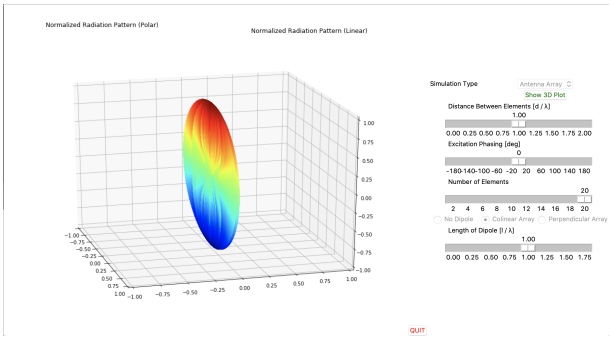
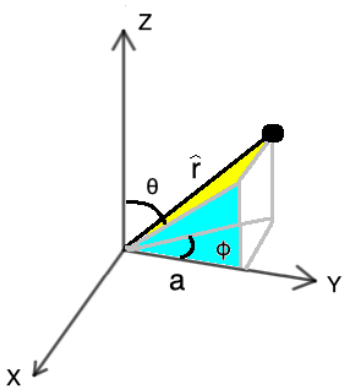


Figure 20: Broadside 3D

VII. References

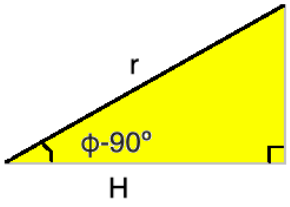
[1] Ramo, s. Whinnery, J. Duzer, T. Fields and Waves in Communication Electronics. *John Wiley & Sons, Inc.*. **3**rd. (1994)

APPENDIX A
DERIVATION OF γ

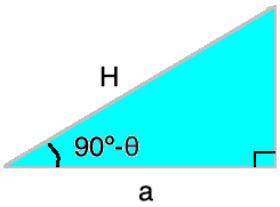


If the array axis is defined as the Y axis, and γ is the angle between the vector along the line of observation and the array axis, γ can be defined as follows:

$$\hat{r} \cdot \hat{y} = a = cos(\gamma)$$
$$\gamma = cos^{-1}(a)$$



$$H = cos(\phi - 90)$$



$$a = Hcos(90 - \theta)$$
$$a = cos(\phi - 90)cos(90 - \theta)$$
$$a = sin(\phi)sin(\theta)$$
$$\gamma = cos^{-1}(sin(\phi)sin(\theta))$$

APPENDIX B
CODE

A. *GUI.py*

```
        # -*- coding: utf-8 -*-
"""
Antenna Visualizer
Created on Thu Feb 27 18:05:13 2020

Description: A GUI based antenna simulator. Tweak sliders to
adjust antenna parameters such as length, array patterns, and
excitation phasing.

Authors:
    Jordan Baxter
    Chelsea Starr
    Ross Dewberry
"""

import tkinter as tk
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)
# Implement the default Matplotlib key bindings.
from matplotlib.backend_bases import key_press_handler
from matplotlib.figure import Figure
import plots

class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        master.title("GUI Based Antenna Simulation")
        self.pack()
        self.create_frames()
        self.create_widgets()


    def create_frames(self):
        self.p_frame = tk.Frame(self.master)
        self.p_frame.pack(side = 'left')
        self.w_frame = tk.Frame(self.master)
        self.w_frame.pack(side = 'right')
        self.plots = plots.Plots(figsize=(11,9), dpi=75)
        self.canvas = FigureCanvasTkAgg(self.plots, master=self.p_frame)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)


    def create_widgets(self):

        ### DROPDOWN OPTIONS MENU ###
        self.simType = tk.StringVar()
        self.simType.set("Single Dipole")
```

```
self.simTypeMenu = tk.OptionMenu(self.w_frame,
                                 self.simType,
                                 "Single Dipole",
                                 "Antenna Array",
                                 command=self.updateControls)
self.simTypeMenu.grid(row=0,
                      column=1,
                      columnspan=2)
tk.Label(self.w_frame, text="Simulation Type").grid(row=0, column=0)
### DROPDOWN OPTIONS MENU ###

### DELTA PHI SLIDER ###
self.dp_sc = tk.Scale(self.w_frame,
                      from_=-180,
                      to=180,
                      resolution=10,
                      length=300,
                      orient='horizontal',
                      tickinterval=20,
                      command=self.upDPhi,
                      label="Excitation Phasing [deg]")
### /DELTA PHI SLIDER ###


### D SLIDER ###
self.d_sc = tk.Scale(self.w_frame,
                      from_=0,
                      to=2,
                      resolution=0.01,
                      length=300,
                      orient='horizontal',
                      tickinterval=0.25,
                      command=self.upD,
                      label="Distance Between Elements [d / \u03bb]")
### /D SLIDER ###

### L SLIDER ###
self.l_sc = tk.Scale(self.w_frame,
                      from_=0,
                      to=1.75,
                      resolution=0.01,
                      length=300,
                      orient='horizontal',
                      tickinterval=0.25,
                      command=self.upL,
                      label="Length of Dipole [l / \u03bb]")
self.l_sc.grid(row=2,
               columnspan=3)
### /L SLIDER ###

### NUMBER OF ELEMENTS SLIDER ###
self.ne_sc = tk.Scale(self.w_frame,
                      from_=2,
                      to=20,
                      resolution=1,
                      length=300,
```

```
                            orient='horizontal',
                            tickinterval=2,
                            command=self.upNumEle,
                            label="Number of Elements")
        ### /NUMBER OF ELEMENTS SLIDER ###

        ### INSERT DIPOLE CHECKBOX ###
        self.insDipVar = tk.IntVar()
        self.insDipVar.set(1)

        self.noDip = tk.Radiobutton(self.w_frame,text="No Dipole", variable=self.insDipVar, va
        self.coLin = tk.Radiobutton(self.w_frame, text="Colinear Array", variable=self.insDipV
        self.perp = tk.Radiobutton(self.w_frame, text="Perpendicular Array", variable=self.ins
        ### /INSERT DIPOLE CHECKBOX ###

        ### Toggle 3D Button ###
        self.button3D = tk.Button(self.w_frame,
                                  text="Show 3D Plot", fg="green",
                                  command=self.up3D)
        self.button3D.grid(row=1,
                           column=1,
                           columnspan=2)

        ### QUIT BUTTON ###
        self.quit = tk.Button(self.master, text="QUIT", fg="red",
                              command=self.master.destroy)
        self.quit.pack(side="bottom")

    def upNumEle(self, slidevalue):
        self.plots.setNumEle(slidevalue)
        self.canvas.draw()

    def upDPhi(self, slidevalue):
        self.plots.setDPhi(slidevalue)
        self.canvas.draw()

    def upD(self, slidevalue):
        if (float(slidevalue) == 0):
            newVal = 0.0001
        else:
            newVal = slidevalue
        self.plots.setD(newVal)
        self.canvas.draw()

    def upL(self, slidevalue):
        if (float(slidevalue) == 0):
            newVal = 0.0001
        else:
            newVal = slidevalue
        self.plots.setL(newVal)
        self.canvas.draw()

    def up3D(self):
        if (self.plots.toggle3D() == True):
            self.toggleWidgets("off")
        else:
```

```
            self.toggleWidgets("on")
        self.canvas.draw()

    def insDip(self):
        arrType = self.insDipVar.get()
        if(arrType == 1):
            self.l_sc.grid_forget()
            self.plots.setArrType("NoDip")
        elif(arrType == 2):
            self.l_sc.grid(row=6,
                           columnspan=3)
            self.plots.setArrType("ColArray")
        elif(arrType == 3):
            self.l_sc.grid(row=6,
                           columnspan=3)
            self.plots.setArrType("PerpArray")
        self.canvas.draw()

    def updateControls(self, value):
        simType = self.simType.get()
        if(simType == "Single Dipole"):
            self.dp_sc.grid_forget()
            self.d_sc.grid_forget()
            self.ne_sc.grid_forget()
            self.noDip.grid_forget()
            self.coLin.grid_forget()
            self.perp.grid_forget()
            self.insDipVar.set(1)
            self.l_sc.grid(row=2,
                           columnspan=3)
        elif(simType == "Antenna Array"):
            self.l_sc.grid_forget()
            self.d_sc.grid(row=2,
                           columnspan=3)
            self.dp_sc.grid(row=3,
                            columnspan=3)
            self.ne_sc.grid(row=4,
                            columnspan=3)
            self.noDip.grid(row=5,
                            column=0)
            self.coLin.grid(row=5,
                            column=1)
            self.perp.grid(row=5,
                            column=2)
        self.plots.setSimType(simType)
        self.canvas.draw()


    def toggleWidgets(self,onOff='on'):
        if(onOff == "off"):
            self.dp_sc.configure(state='disabled')
            self.d_sc.configure(state='disabled')
            self.noDip.configure(state='disabled')
            self.coLin.configure(state='disabled')
            self.perp.configure(state='disabled')
            self.l_sc.configure(state='disabled')
```

```
            self.simTypeMenu.configure(state='disabled')
        else:
            self.dp_sc.configure(state='normal')
            self.d_sc.configure(state='normal')
            self.noDip.configure(state='normal')
            self.coLin.configure(state='normal')
            self.perp.configure(state='normal')
            self.l_sc.configure(state='normal')
            self.simTypeMenu.configure(state='normal')



### Constuct Figures ###

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

B. plots.py

```
        # -*- coding: utf-8 -*-
"""
Plots Module for Antenna Simulator
Created on Fri Feb 28 14:36:18 2020

Author: Jordan Baxter
"""
import matplotlib.pyplot as plt
import numpy as np
from numpy import pi
from matplotlib.figure import Figure
import antennas as ant
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

### Create Plot Class ###
class Plots(Figure):
def __init__(self, figsize=None, dpi=None):
super().__init__(figsize,dpi)
self.figsize = figsize
### CREATE PLOTS ###
self.ax = self.add_subplot(221, polar=True)
self.bx = self.add_subplot(222)
self.cx = self.add_subplot(223, polar=True)
self.dx = self.add_subplot(224)
self.subplots_adjust(left=0, right=1,top=0.9, bottom=0.05, wspace=0.05, hspace=0.3)
### FORMAT PLOTS ###
self.ax.set_rmin(0)
self.ax.set_rlabel_position(180)
self.ax.set_title("Normalized Radiation Pattern (Polar)", pad=10)


self.bx.set_xlim(-pi,pi)
self.bx.set_ylim(0,1.1)
self.bx.set_title("Normalized Radiation Pattern (Linear)", pad=10)
```

```
self.d_ticks = np.linspace(0, 30, 31)
self.cx.set_rmin(0)
self.cx.set_rlabel_position(180)
self.cx.set_rticks(self.d_ticks)
self.cx.set_title("Antenna Directivity (Polar)", pad=10)

self.dx.set_xlim(-pi,pi)
self.dx.set_title("Antenna Directivity (Linear)", pad=10)

### SET INITIAL ANTENNA PARAMETERS ###
self.theta2D = np.linspace(-pi, pi, 10000)
self.numEle = 2
self.simType = "Single Dipole"
self.arrType = "NoDip"
self.d_phi = float(0)
self.d = float(0.0000001)
self.len = float(0.0000001)
self.plot3D = False
self.antProf = ant.AntennaProfile(self)
self.init_2Dplots()



def init_2Dplots(self):
self.ax.plot(self.theta2D, self.antProf.eRad2D,'b', label="E-Plane")
self.bx.plot(self.theta2D, self.antProf.eRad2D,'b', label="E-Plane")
self.ax.plot(self.theta2D, self.antProf.hRad2D,'g', label="H-Plane")
self.bx.plot(self.theta2D, self.antProf.hRad2D,'g', label="H-Plane")
self.cx.plot(self.theta2D, self.antProf.DtPat,'b')
self.dx.plot(self.theta2D, self.antProf.DtPat,'b')
self.ax.legend(loc='upper right')
self.cx.set_rmax(int(self.antProf.direc) + 1)
self.dx.set_ylim(0, int(self.antProf.direc) + 1)
self.bx.legend(loc='upper right')

def init_3Dplot(self):

self.theta3D = np.linspace(0.0000000000001, pi, 40 + self.numEle * 10)
self.phi3D = np.linspace(-pi, pi, 40 + self.numEle * 10)
self.THETA, self.PHI = np.meshgrid(self.theta3D, self.phi3D)
self.gamma3D = np.arccos(np.sin(self.PHI)*np.sin(self.THETA))

self.antProf.init_3DPlot(self)
self.X = self.antProf.rad3D * np.sin(self.THETA) * np.cos(self.PHI)
self.Y = self.antProf.rad3D * np.sin(self.THETA) * np.sin(self.PHI)
self.Z = self.antProf.rad3D * np.cos(self.THETA)
self.ex.plot_surface(self.X, self.Y, self.Z, rstride=1, cstride=1, cmap=plt.get_cmap('jet'),
linewidth=0, antialiased=False, alpha=0.5)
self.ex.set_xlim(-1,1)
self.ex.set_ylim(-1,1)
self.ex.set_zlim(-1,1)

def update_plots(self):
if(self.plot3D):
    self.antProf.init_3DPlot(self)
```

```
        del self.ex.lines[0:len(self.ex.lines)]
        self.X = self.antProf.rad3D * np.sin(self.THETA) * np.cos(self.PHI)
        self.Y = self.antProf.rad3D * np.sin(self.THETA) * np.sin(self.PHI)
        self.Z = self.antProf.rad3D * np.cos(self.THETA)
        self.ex.plot_surface(self.X, self.Y, self.Z, rstride=1, cstride=1, cmap=plt.get_cmap('jet'
        linewidth=0, antialiased=False, alpha=0.5)
    else:
        self.antProf.update_2DPlot(self)
        del self.ax.lines[0:len(self.ax.lines)]
        del self.bx.lines[0:len(self.bx.lines)]
        del self.cx.lines[0:len(self.cx.lines)]
        del self.dx.lines[0:len(self.dx.lines)]
        if(self.simType == "Single Dipole"):
            self.ax.plot(self.theta2D, self.antProf.eRad2D,'b', label="E-Plane")
            self.bx.plot(self.theta2D, self.antProf.eRad2D,'b', label="E-Plane")
            self.ax.plot(self.theta2D, self.antProf.hRad2D,'g', label="H-Plane")
            self.bx.plot(self.theta2D, self.antProf.hRad2D,'g', label="H-Plane")
            self.cx.plot(self.theta2D, self.antProf.DtPat,'b')
            self.dx.plot(self.theta2D, self.antProf.DtPat,'b')
            self.ax.set_theta_direction(-1)
            self.ax.set_theta_zero_location("N")
            self.cx.set_theta_direction(-1)
            self.cx.set_theta_zero_location("N")
        elif(self.simType == "Antenna Array" and self.arrType == "NoDip"):
            self.ax.plot(self.theta2D, self.antProf.eRad2D,'b', label="Antenna Factor")
            self.bx.plot(self.theta2D, self.antProf.eRad2D,'b', label="Antenna Factor")
            self.cx.plot(self.theta2D, self.antProf.DtPat,'b')
            self.dx.plot(self.theta2D, self.antProf.DtPat,'b')
            self.ax.set_theta_direction(1)
            self.ax.set_theta_zero_location("E")
            self.cx.set_theta_direction(1)
            self.cx.set_theta_zero_location("E")
        elif(self.simType == "Antenna Array"):
            self.ax.plot(self.antProf.gamma, self.antProf.eRad2D,'b', label="E-Plane")
            self.bx.plot(self.antProf.gamma, self.antProf.eRad2D,'b', label="E-Plane")
            self.ax.plot(self.antProf.gamma, self.antProf.hRad2D,'g', label="H-Plane")
            self.bx.plot(self.antProf.gamma, self.antProf.hRad2D,'g', label="H-Plane")
            self.cx.plot(self.antProf.gamma, self.antProf.DtPat,'b')
            self.dx.plot(self.antProf.gamma, self.antProf.DtPat,'b')
            self.ax.set_theta_direction(1)
            self.ax.set_theta_zero_location("E")
            self.cx.set_theta_direction(1)
            self.cx.set_theta_zero_location("E")
        self.ax.legend(loc='upper right')
        self.cx.set_rmax(int(self.antProf.direc) + 1)
        self.dx.set_ylim(0, int(self.antProf.direc) + 1)
        self.bx.legend(loc='upper right')

    def setDPhi(self, newDPhi):
    self.d_phi = np.radians(int(newDPhi))
    self.update_plots()

    def setD(self, newD):
    self.d = float(newD)
    if(self.d == 0):
        self.d = 0.0001
```

```
self.update_plots()

def setL(self, newLen):
self.len = float(newLen)
self.update_plots()

def setSimType(self, newtype):
self.simType = newtype
self.update_plots()

def setArrType(self, newtype):
self.arrType = newtype
self.update_plots()

def setNumEle(self, newNumEle):
self.numEle = int(newNumEle)
self.update_plots()

def toggle3D(self):
self.plot3D = not self.plot3D
if(self.plot3D == True):
    self.ax.axis('off')
    self.bx.axis('off')
    self.cx.axis('off')
    self.dx.axis('off')
    self.ex = self.add_subplot(111, projection='3d')
    self.init_3Dplot()
else:
    self.ax.axis('on')
    self.bx.axis('on')
    self.cx.axis('on')
    self.dx.axis('on')
    self.ex.remove()
return self.plot3D
```

C. *antennas.py*

```
# -*- coding: utf-8 -*-
"""
Antenna Module for Antenna Simulator

Author: Jordan Baxter
"""

import numpy as np
from numpy import pi, cos, sin, tan
from scipy import integrate

class AntennaProfile():
def __init__(self, Plots):
self.init2DPlot(Plots)
self.direc = self.getDirectivity(Plots)
self.DtPat = self.initDirPlot()

def init2DPlot(self, Plots):
self.eRad2D = abs(((cos(Plots.len*pi*cos(Plots.theta2D)) - cos(Plots.len*pi))/sin(Plots.theta2
self.eRad2D = np.divide(self.eRad2D, np.amax(self.eRad2D))
```

```
self.hRad2D = np.ones(Plots.theta2D.shape[0])

def initDirPlot(self):
return self.direc * self.eRad2D**2

def update_2DPlot(self, Plots):
if(Plots.simType == "Single Dipole"):
self.eRad2D = abs(((cos(Plots.len*pi*cos(Plots.theta2D)) - cos(Plots.len*pi))/sin(Plots.theta2
self.hRad2D = np.ones(Plots.theta2D.shape[0])
elif(Plots.simType == "Antenna Array"):
if(Plots.arrType == "NoDip"):
    sigma = np.add (2 * pi * Plots.d * cos(Plots.theta2D), Plots.d_phi)
    N = Plots.numEle
    self.arrFact = (1 / N) * np.abs(np.divide(sin(N * sigma /2), sin(sigma / 2)))
    self.eRad2D = np.divide(self.arrFact, np.max(self.arrFact))
elif(Plots.arrType == "ColArray"):
    self.gamma = Plots.theta2D
    self.antPat = abs(((cos(Plots.len*pi*cos(Plots.theta2D)) - cos(Plots.len*pi))/sin(Plots.th
    self.antPat = np.divide(self.antPat, np.amax(self.antPat))
    sigma = np.add (2 * pi * Plots.d * cos(self.gamma), Plots.d_phi)
    N = Plots.numEle
    self.arrFact = (1 / N) * np.abs(np.divide(sin(N * sigma /2), sin(sigma / 2)))
    self.eRad2D = np.multiply(self.antPat, self.arrFact)
    self.hRad2D = np.ones(Plots.theta2D.shape[0])
elif(Plots.arrType == "PerpArray"):
    self.gamma = Plots.theta2D
    sigma = np.add (2 * pi * Plots.d * cos(self.gamma), Plots.d_phi)
    N = Plots.numEle
    self.arrFact = (1 / N) * np.abs(np.divide(sin(N * sigma /2), sin(sigma / 2)))
    self.antPat = abs(((cos(Plots.len*pi*cos(Plots.theta2D - pi / 2)) - cos(Plots.len*pi))/sin
    self.antPat = np.divide(self.antPat, np.amax(self.antPat))
    self.eRad2D = np.multiply(self.antPat, self.arrFact)
    self.hRad2D = self.arrFact
self.hRad2D = np.divide(self.hRad2D, np.max(self.hRad2D))
self.eRad2D = np.divide(self.eRad2D, np.amax(self.eRad2D))
self.direc = self.getDirectivity(Plots)
self.DtPat = self.direc * self.eRad2D**2

def getDirectivity(self, Plots):
I = integrate.cumtrapz(self.eRad2D[5000:len(self.eRad2D)] ** 2 * sin(Plots.theta2D[5000:len(Pl
return round((2 / I[len(I)-1]), 2)

def init_3DPlot(self, Plots):
if(Plots.simType == "Single Dipole"):
self.antPat3D = ((cos(Plots.len*pi*cos(Plots.THETA)) - cos(Plots.len*pi))/sin(Plots.THETA))
self.antPat3D = np.divide(self.antPat3D, np.amax(self.antPat3D))
self.rad3D = self.antPat3D
elif(Plots.simType == "Antenna Array"):
if(Plots.arrType == "NoDip"):
    sigma = np.add (2 * pi * Plots.d * cos(Plots.gamma3D), Plots.d_phi)
    N = Plots.numEle
    self.arrFact3D = (1 / N) * np.abs(np.divide(sin(N * sigma /2), sin(sigma / 2)))
    self.rad3D = np.divide(self.arrFact3D,np.max(self.arrFact3D))
elif(Plots.arrType == "ColArray"):

    sigma = np.add (2 * pi * Plots.d * cos(Plots.gamma3D), Plots.d_phi)
```

```
    N = Plots.numEle
    self.arrFact3D = (1 / N) * np.abs(np.divide(sin(N * sigma /2), sin(sigma / 2)))

    self.antPat3D = ((cos(Plots.len*pi*cos(Plots.gamma3D)) - cos(Plots.len*pi))/sin(Plots.gamm
    self.antPat3D = np.divide(self.antPat3D, np.amax(self.antPat3D))

    self.rad3D = np.multiply(self.antPat3D, self.arrFact3D)
    self.rad3D = np.divide(self.rad3D, np.max(self.rad3D))
elif(Plots.arrType == "PerpArray"):

    sigma = np.add (2 * pi * Plots.d * cos(Plots.gamma3D), Plots.d_phi)
    N = Plots.numEle
    self.arrFact3D = (1 / N) * np.abs(np.divide(sin(N * sigma /2), sin(sigma / 2)))

    self.antPat3D = ((cos(Plots.len*pi*cos(Plots.THETA)) - cos(Plots.len*pi))/sin(Plots.THETA)
    self.antPat3D = np.divide(self.antPat3D, np.amax(self.antPat3D))

    self.rad3D = np.multiply(self.antPat3D, self.arrFact3D)
    self.rad3D = np.divide(self.rad3D, np.max(self.rad3D))
```