Fall 2022

A Design Flow for Additively Manufactured 3D Metasurface Antennas

Justin Parkhurst
Embry-Riddle Aeronautical University, parkhurj@my.erau.edu

A Design Flow for Additively Manufactured 3D Metasurface Antennas

by

Justin Parkhurst

This thesis was prepared under the direction of the candidate's thesis committee chairman, Dr. Eduardo Rojas-Nastrucci, Department of Electrical Engineering and Computer Science, and has been approved by the members of the thesis committee. It was submitted to the Department of Electrical Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Masters of Electrical and Computer Engineering.

THESIS COMMITTEE:

____
Eduardo Rojas-Nastrucci, Ph.D
Committee Chairman

____
Jianhua Liu, Ph.D.
Committee Member

____
M. Ilhan Akbas, Ph.D.
Committee Member

____
Radu F. Babiceanu, Ph.D.
Department Chair, Electrical Engineering and Computer Science

____
Associate Vice President for Academics
Christopher Grant

DEDICATION

To my parents, Kim and Timothy, and my siblings, TJ and Tracy.

ACKNOWLEDGMENTS

The completion of this thesis would not have been possible without the assistance and opportunities provided to me by many people throughout my life.

First and foremost, I wish to thank Embry-Riddle Aeronautical University and all of the professors I've had here for sharing their knowledge and skills with me. I especially want to thank Dr. Jianhua Liu and Dr. M. Ilhan Akbas for being a part of my thesis committee. On top of being exceptional instructors, Dr. Liu and Dr. Akbas are both kind and genuine people; personality traits that show clearly in their lectures.

I would like to thank the many friends I have made since I first came to ERAU. To my brothers of the Alpha Delta Nu chapter of Alpha Phi Omega, for giving me a strong community to rely upon. To my colleagues at WiDE Lab, for supporting each other academically, and through looming deadlines. To my close friends, Avalon, Gwen, John, and Rose for giving me a creative outlet through weekly tabletop games, that gave me a chance to relax and practice my less academic skills.

Of my friends, I specifically want to acknowledge Kevin Pepin, whose friendship we have forged through countless projects since sophomore year. Kevin was always there to push me forward when circumstances seemed too much to handle. I know part of my success is owed to him.

I wish to thank my family for supporting me since the day I came into this world. To my sister Tracy for always believing in me. To my brother TJ for cheering me on in everything I do.

And to my mom for being someone who I could come to for anything and for never letting me give up on myself.

Finally, I wish to thank Dr. Eduardo Rojas-Nastrucci, who has served as the Chairman and lead advisor for my thesis. His knowledge has been invaluable in helping me understand the advanced concepts needed to perform the work shown in this thesis. More importantly, I’m grateful to Dr. Rojas for being a guide in my academic career for nearly four years now. Before I joined his research team at WideLabs, I was unsure what path I wanted to take with my electrical engineering degree. Through his instruction and the opportunity to take part in many projects across various topics in the field, I found my niche in the design of radio frequency systems. My success, both academic and professional, is due to his kindness, patience, and determination to see his students succeed.

Table of Contents

Chapter I: Introduction

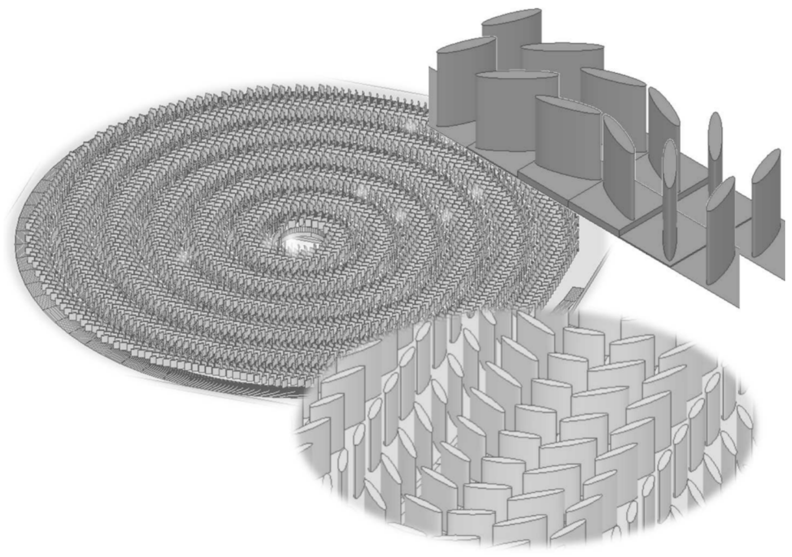


Fig. 1. Metal - Only Modulated Metasurface (MTS) Antenna [1]

The small satellite industry has grown exponentially over the last decade with the pursuit of commercial use spacecraft by a growing number of industries. The satellite industry is experiencing a shift from large spacecraft platforms that require a multi-year effort for design and manufacture, to low-cost small platforms that can be deployed and improved quickly. A key requirement for small satellites missions is a reliable wireless communication link to ground, and between satellites, with strict requirements on volume and weight. Typical space antennas use solid reflectors and phased arrays, but these typically present difficulties in achieving the size limitations of small satellites. Deployable reflectarrays have induced many creative solutions for reducing the payload size of satellites, including various methods of folding a reflect array for it to be released into its full-size when in orbit [2]. However, these designs still come with the burden of deployment mechanisms built into the payloads, that are often difficult to make reliable, while still adhering to size constraints. Metasurfaces (MTS) antennas offer a low-profile, low weight solution that is compatible with small satellites, embrace the entire 1U side of the satellite as an antenna aperture, and have a feeding mechanism that will keep it protected in a space environment. The structure of MTS antennas opens many avenues for storage and placement of antennas within a small payload such as having them printed directly onto the chassis of a satellite or similar surface, which can allow them to forgo the deployment mechanisms that are required in many existing SmallSats and CubeSats.

MTS antennas are relatively new to the field of RF and understanding how to go about designing such an antenna can require considerable time and effort. This creates a barrier against designers who may benefit from the implementation of an MTS antenna over more traditional methods, but lack the appropriate time needed to fully comprehend their mechanics and then be able to apply this knowledge. This in turn limits the number of individuals able to experiment in this area of antenna science. While there are many papers that showcase designs for MTS antennas and how effective they are, there are few that clearly explain how one would go about creating one for their own purposes. This is understandable as most papers in this field are written for those who also have a background in antenna design. But nevertheless creates an unintended roadblock for those who wish to replicate the results of a paper, expand on new developments, or simply design a similar MTS antenna for their own purposes, but may lack some critical knowledge of MTS design.

The objective of this thesis is to simplify the development for one type of MTS antenna by presenting a design flow for additively manufacturable MTS antennas. The process will be broken down into following steps:

1. Characterization of a single unit cell of the MTS Array.
2. Calculation of optimal layout pattern for all antenna elements.
3. Automated generation of antenna array in 3D modeling software.
4. Testing and analysis of antenna performance in HFSS

Through the process outlined in this paper, anyone with some background in RF should be able to design an MTS antenna like that shown in [1]. Furthermore, they will be able to modify the parameters of this MTS antenna to suit their own purposes.

Chapter II: Theoretical Background / Literature Review

Section 1: Literature Review

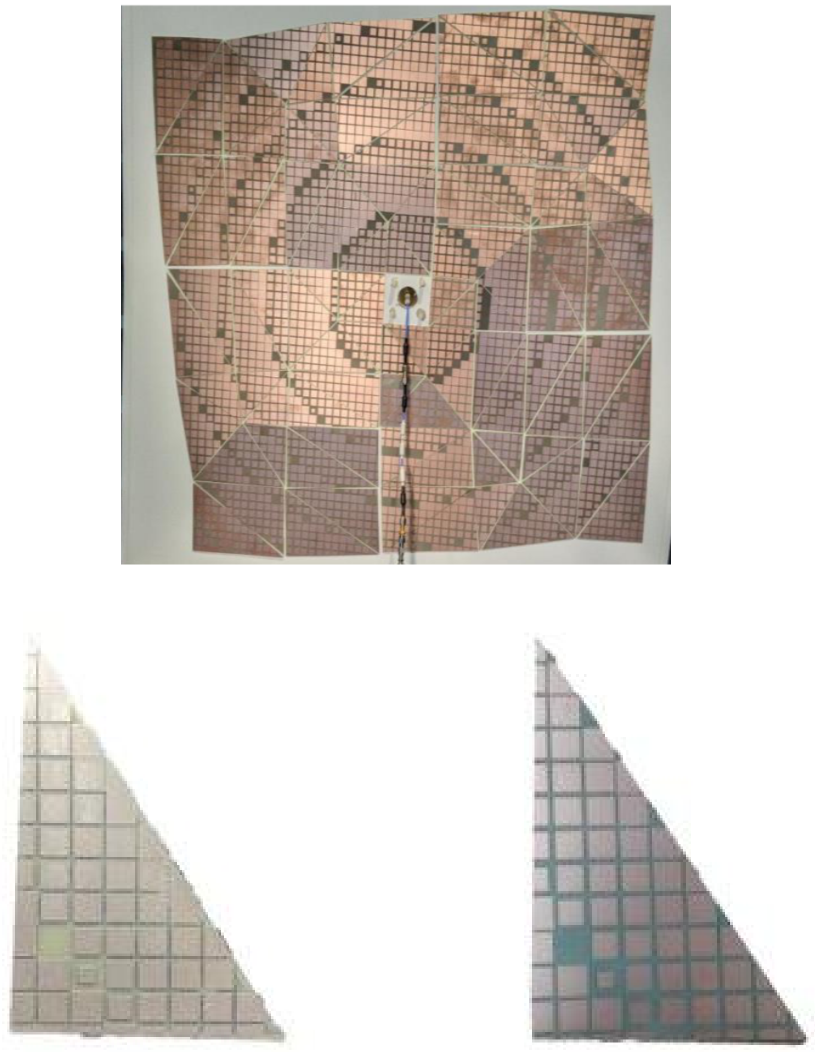


Fig. 2-1. (Top) Laser-machined origami reflectarray. (L) Additively manufactured RA element with Ultem and CB028, and (R) element manufactured with RO4003C. [2]

MTS antennas consist of an array of elements that when excited by oncoming signals will induce surface waves that will carry energy towards the aperture of the antenna, with that aperture typically connecting to a waveguide or coaxial cable. This method of energy transfer via surface waves causes MTS antennas to be relatively high gain despite their small size and low-profile. These element arrays have often taken the form of a grid of patches like the array of square patches shown in Fig. 2-1, with some arrays having more complex patch geometries [3] - [5]. These grids have a pattern to their structure that allows their behavior to be modeled through what are known as group theories, where the array can be characterized as a unit cell that is repeated across the layout [6] - [8]. This analysis takes the single unit cell and characterizes it for all parameters that define its elements. This then allows for the generation of a layout that will correctly guide surface waves towards the aperture of the antenna [9].

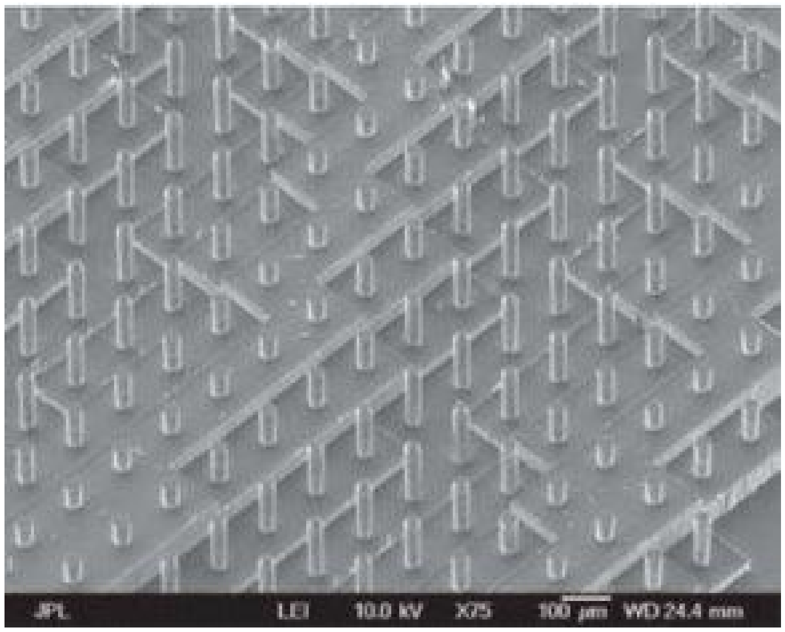


Fig 2-2. SEM images of the 300 GHz MTS antenna realized with isotropic elements. [10]

The defining feature of an MTS array is the element structure that is the basic unit cell of the layout. While there have been many designs using essentially flat elements, there have recently been designs presented that make use of three dimensional elements [11] - [13]. These 3D elements allow the resonant frequency of an element to be controlled by the height of an element, allowing for thinner elements that can be more closely placed together. This is good for situations where surface area available for the MTS array is limited. These unit cell designs take advantage of the advancements in 3D printers to create increasingly complex layouts with microscopic elements [10]. This induces a need for a design flow that can reliably develop these layouts to increase accessibility to potential designers.

Section 2: Element Characterization

The MTS array that this paper aims to provide a design flow for is composed of thousands of small metallic cylinders, acting as resonators, each exert their own influence on the performance of the antenna. The position and angle of each cylinder must be taken in consideration if the antenna is to achieve optimal performance. Since all elements in the MTS array are identical in shape, a database can be constructed by simulating a single across a range of frequencies and a range of orientation angles. The metallic cylinders are best described as reactance tensors of the second order [1]. These tensors have inherent properties that are determined by their position in the array, orientation, surface wave vectors, and phase shift components that need to be simulated. As such, the database will need to include the theoretical tensors values at each point in the array and all phase shifts that could be applied to any element in the array. This database will then be referenced, using equation (1) [14] to determine the parameters of each element that will achieve the desired center frequency for the full antenna.

$$\mathbf{(X_{xx}X_{yy} - X_{xy}X_{yx} - 1)ka_z + (X_{yx} + X_{xy})Bsw\ Bsw + (X_{xx}+X_{yy})*k^2 - Xxx*(BYsw^2)- X_{yy}*(BXsw^2)) = 0} \quad (1)$$

Equation (1) provides a relationship between the theoretical values of the reactance tensor (X_{xx}, X_{xy}, X_{yx}, and X_{yy}) and the values of phase shifts (BXsw and BYsw) obtained from the simulated element.

$$\boldsymbol{\beta}_{\mathrm{SW}} = \beta_{\mathrm{SW}}\rho = \boldsymbol{\beta}\mathbf{X}_{\mathrm{SW}} + \boldsymbol{\beta}\mathbf{Y}_{\mathrm{SW}} \tag{2}$$

The phase shifts are assigned in pairs consisting of their X and Y components alongside an orientation angle, P_{sy}, which collectively generated a resonant frequency in the simulation.

$$\mathbf{k = 2\pi/\lambda} \tag{3}$$

$$\mathbf{a_z = sqrt[(\boldsymbol{\beta}_{SW})^2 - k^2]} \tag{4}$$

The remaining variables shown in equation (1) K, the free space wave number, and a_z, the free space wave number with respect to the surface wave, are defined by equation (3) and equation (4), respectively. When provided with the required inputs, equation (1) will output a residual value, the lower this value is the better the performance of the tensor. For each tensor, the equation will iterate through all pairs of phase shifts at a given frequency, with the inputs that result in closest output to zero being the optimal solution for that tensor. The orientation angle associated with the best fit phase shifts, will be assigned to the element represented by the tensor in the calculation.

Section 3: Tensor Values of RHCP

This section describes the matrix components that define how each tensor in the antenna array will be determined. Each tensor component is defined with respect to a polar coordinate system, given the circular pattern of the MTS antenna. There are four components to each tensor which are: $X_{\rho\rho}$, $X_{\rho\phi}$, $X_{\phi\rho}$, $X_{\phi\phi}$. Equations (5-7) show how to calculate each component [15].

$$\mathbf{X_{\rho\rho}(\rho) = X_0[1+Main(2\pi\rho/p-\phi)]} \tag{5}$$

$$\mathbf{X_{\rho\phi}(\rho) = X_{\phi\rho}(\rho) = X0\ cos(2\pi\rho/p-\phi)} \tag{6}$$

$$\mathbf{X_{\phi\phi}(\rho) = X_0[1-Main(2\pi\rho/p-\phi)]} \tag{7}$$

X_0 a scaling factor the average reactance for the tensors and will be defined by the designer. M is the modulation index, determined by the designer. ρ and ϕ are the coordinates of the tensor on the polar grid. The period of modulation, p, can be calculated as shown below:

$$\mathbf{p = 2\pi/\beta_{SW}} \tag{8}$$

Note that with these tensors components calculated it will not immediately be possible to use them in equation (1), as that equation uses tensor values with respect to cartesian coordinates. Therefore these tensor matrices must next be converted from polar to cartesian.

Section 4: Tensors and Tensor Transformation

The tensor components shown in section 3, need to be converted to the cartesian coordinate system in order to be used in equation (1). In this section, the process for converting a tensor from polar form to cartesian form will be outlined. The matrix for a tensor in polar form is described in both equations (9) and (10), with equation (10) showing how the polar form has a direct link to the cartesian form for a tensor

$$\mathbf{T_{Pol}} = \begin{bmatrix} X\rho\rho & X\rho\phi \\ X\phi\rho & X\phi\phi \end{bmatrix} \tag{9}$$

$$\mathbf{T_{Pol} = Q^T * T_{Cart} * Q} \tag{10}$$

In this case the tensors are of 2nd order, which can be formed by the outer product of two vectors as shown below:

$$\boldsymbol{P} = a\widehat{\boldsymbol{x}} + b\widehat{\boldsymbol{y}} = \begin{bmatrix} a \\ b \end{bmatrix}, \ \boldsymbol{Q} = c\widehat{\boldsymbol{x}} + d\widehat{\boldsymbol{y}} = \begin{bmatrix} c \\ d \end{bmatrix}, \quad \boldsymbol{PQ} = ac\widehat{\boldsymbol{x}}\widehat{\boldsymbol{x}} + ad\widehat{\boldsymbol{x}}\widehat{\boldsymbol{y}} + bc\widehat{\boldsymbol{y}}\widehat{\boldsymbol{x}} + bd\widehat{\boldsymbol{y}}\widehat{\boldsymbol{y}} = \begin{bmatrix} ac & ad \\ bc & bd \end{bmatrix}$$

The matrix form of tensor PQ, can be remade to reflect the cartesian form of the tensor matrix by replacing the matrix components in a more familiar form with respect to equation (1).

$$\begin{bmatrix} ac & ad \\ bc & bd \end{bmatrix} = \begin{bmatrix} Xxx & Xxy \\ Xyx & Xyy \end{bmatrix}$$

$$\mathbf{T_{Cart}} = \begin{bmatrix} Xxx & Xxy \\ Xyx & Xyy \end{bmatrix} \tag{11}$$

The Q matrix is created by taking the jacobian matrix of the equations for transforming from cartesian to polar [16].

$$\mathbf{Q} = \begin{bmatrix} \cos\vartheta & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta \end{bmatrix} \tag{12}$$

From here, the cartesian form can be easily calculated by reworking equation (10) to solve for T_{cart}.

$$\mathbf{T_{Cart} = Q * T_{Pol} * Q^T =} \begin{bmatrix} \cos\vartheta & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta \end{bmatrix} \begin{bmatrix} X\rho\rho & X\rho\phi \\ X\phi\rho & X\phi\phi \end{bmatrix} \begin{bmatrix} \cos\vartheta & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta \end{bmatrix}^T \tag{13}$$

Finally, the tensors values are in a form that can be used in equation (1).

Chapter III: MTS Unit Cell Design and Numerical Electromagnetic Simulation

Section 1: Purpose of the Unit Cell Analysis

The MTS Antenna shown in Fig.1 consists of thousands of small subwavelength metallic resonators, in the shape of elliptic cylinders, that collectively determine the beam pattern of the full antenna. More specifically, each element causes phase shifts to a surface wave as it travels along the antenna's surface, which in turn results in a directive beam that resonates at a certain frequency [17]. To accurately generate the layout of the full antenna; a simulation model must be developed that can properly characterize the behavior of the individual elements in the antenna array. This can be achieved through unit cell analysis. Unit cell analysis is simply a means of reducing a MTS antenna down to its most basic pattern element and simulating how that element reacts when altered by design variables. For example, [17] presents a unit cell model that measures two rectangular patches, one with a hold in the center, and sees how the length of the patch, or the radius of the hole, affects the phase shifts applied by the unit cell. Since all of the cylinders are the same structure and the only difference between them is their orientation angle, the model only needs to consist of one elliptical cylinder. This is advantageous as it means the full antenna will not consist of any more elements than necessary and that only one model will need to be made.

Using the parameters provided in [1], we get that the elliptical cylinder for the unit cell has the following dimensions: major axis A = 1.2 mm, minor axis a = 240 µm, height h = 1.525 mm, and unit cell side length of d = 1.235. The model for this unit cell is generated in Ansys

HFSS, in order to measure the far field wave patterns that will be needed when determining the best parameters for each element in using (1).

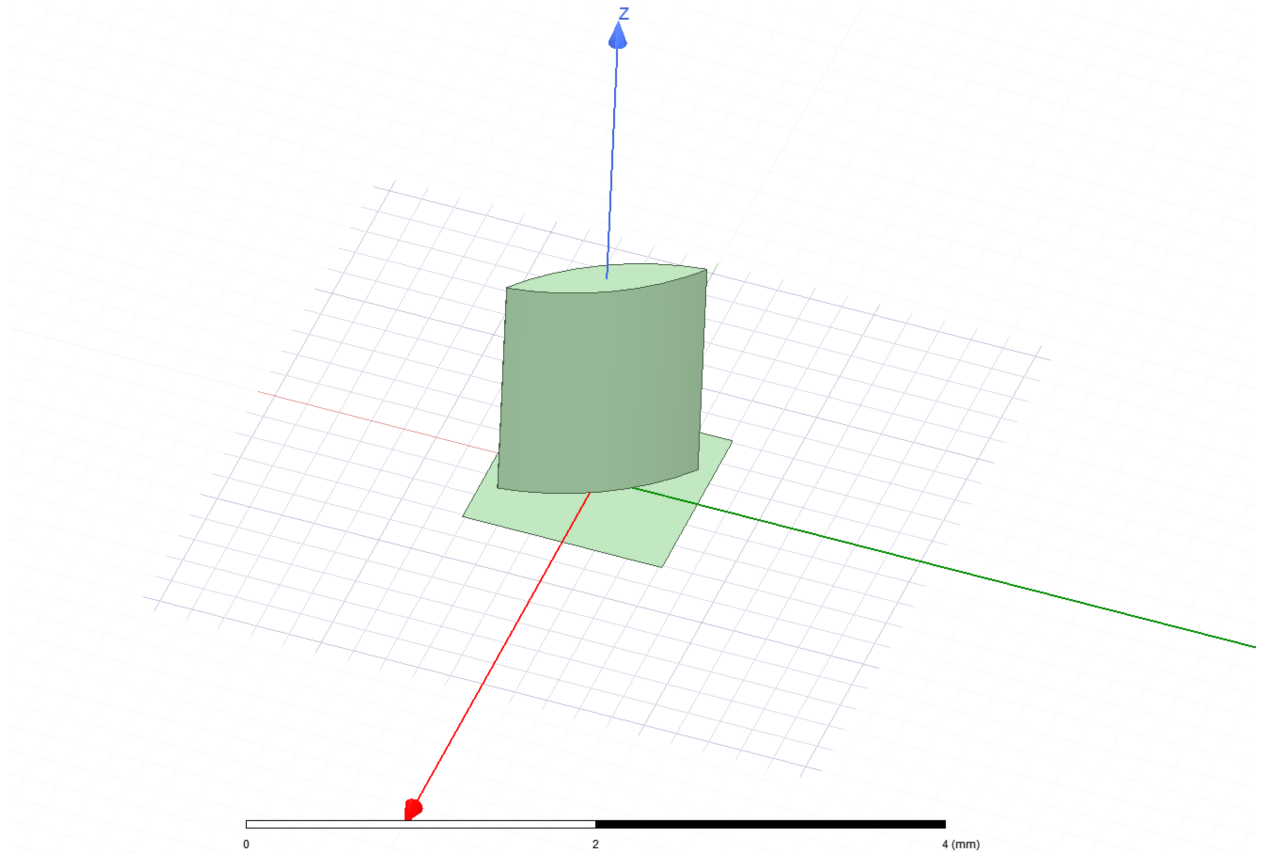


Fig. 3-1. Isometric view of a single unit cell of the MTS Array, generated in HFSS.

The process for generating the geometry shown in Fig. 3-1 is outlined in section 5. It should be noted that the angle of the elliptical cylinder was not given in [1], presumably because all angles must eventually be considered, so an angle of 30 degrees was arbitrarily chosen in this instance. Additionally, the geometry is assigned copper as its material property to help achieve high gain in the far field analysis of the full antenna.

The objective of this simulation is to evaluate what frequency is induced when a pair of phase shifts are applied to the element. This data will generate a plot of the isofrequency dispersion contours, that relates all phase shift combinations to a propagation frequency which will be useful when designing an antenna for a certain center frequency. To verify the accuracy of this element simulation, the isofrequency dispersion plot generated from the simulation will be compared to Fig. 3-2.

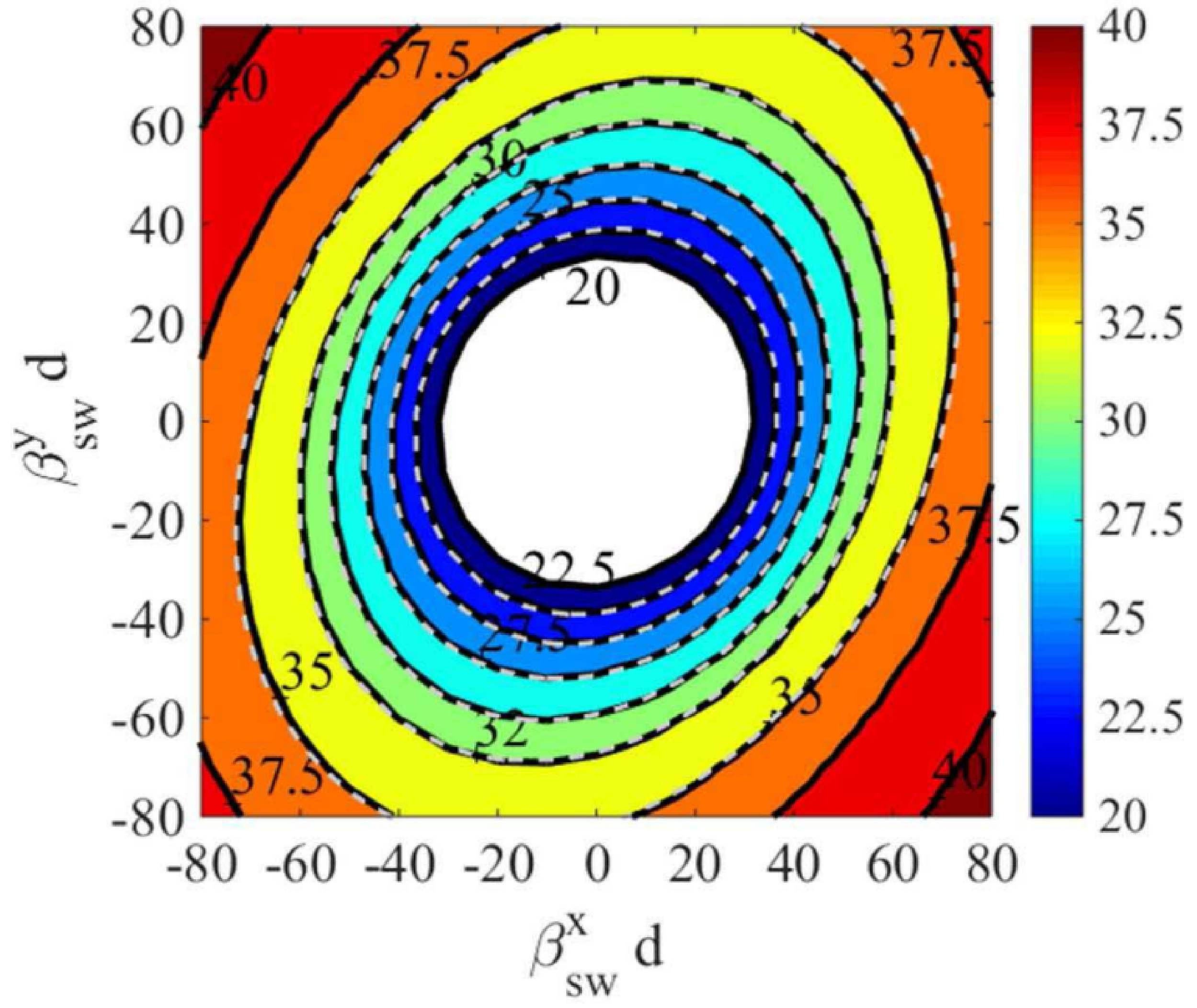


Fig. 3-2. Isofrequency dispersion contours for the unit cell used in the reference antenna. The x and y axes represent the phase shift in thc x and y directions, respectively. The colors correspond to a frequency, in gigahertz, at a given phase shift pair. [1]

Section 2: Simulation Setup - Floquet Model

The model uses an Eigenmode solution type with boundary conditions set up as would be done in Floquet Analysis [18]. An Eigenmode solution type is used as it is better suited to analyzing resonators, such as those that make up a metasurface antenna, because it considers the structure as having stored energy, and thus allows for the resulting modes and field properties of the resonator to be calculated [19]. The boundary box uses the width and length of the unit cell and has a height that is six times the cylinder height, h. The bottom side of the boundary box is a Perfect E, also known as a perfect electrical conductor (PEC), that is meant to force the electric field to be perpendicular to the xy-plane. The height of the boundary box is such that higher order modes of the unit cell are attenuated out in the simulation. The boundary conditions along the x-z and y-z sides of the boundary box, are a pair of Primary and Secondary boundaries (older versions of HFSS refer to these as Master and Slave boundaries), as shown in fig. 3-3. Since the Eigenmode solution type already defines the structure as an energy source, there is no need for an excitation to be defined in this simulation. The paired boundary conditions are applying phase shifts along the x and y axes; these phase shifts are referred to as Px and Py, respectively. In fact, HFSS recognizes this and will prevent a user from implementing any excitation, such as a wave port, into the design. By considering each pair of the phase shifts, the propagation frequency for the fundamental mode can be obtained.

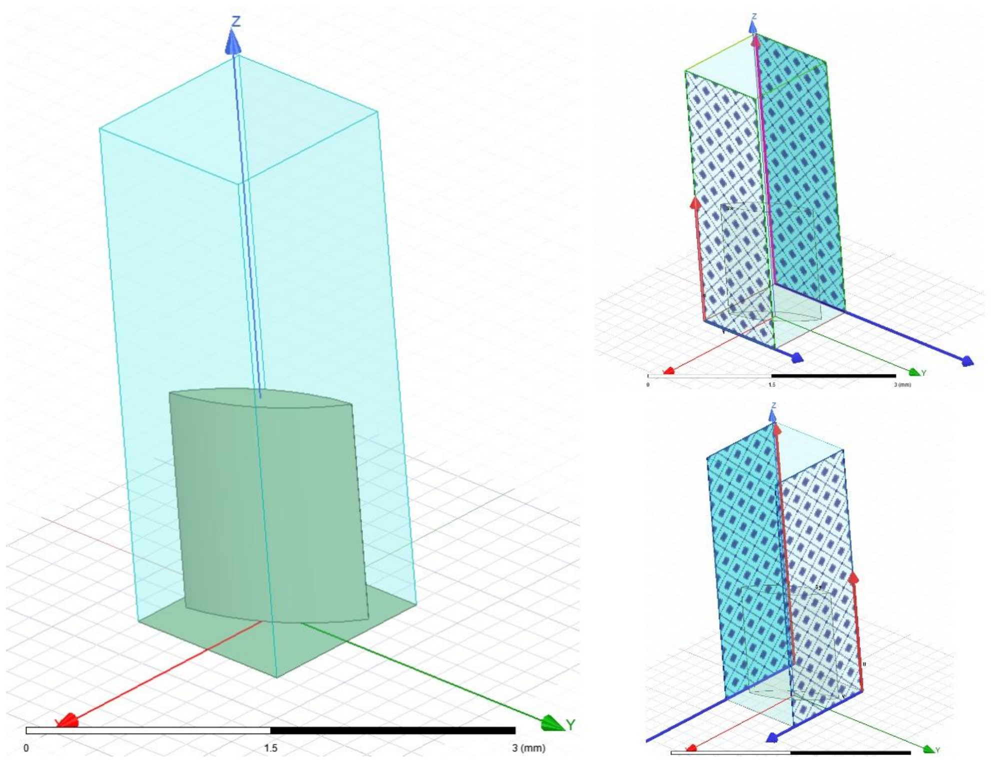


Fig. 3-3. (Left) Metallic cylinder unit cell with radiation box. (Top Right) Primary and secondary boundary for the phase delay along the x-axis. (Bottom Left) Primary and secondary boundary for the phase delay along the y-axis.

Section 3: Simulation Optimetrics

In HFSS, simulations have an analysis setup that is used to define the frequency sweep and sweep type of the simulation. The sweeping of variables like the phase shifts are referred to as Optimetrics in HFSS.

To have enough data to recreate the isofrequency dispersion curves of Fig. 3-2, the unit cell has to be analyzed with respect to a large number of combinations of phase shifts, Px and Py. The step size needs to be small enough that clear lines of frequency shift can be seen in the resulting graph. However, if the step sizes are too small, the simulation can become time intensive. Since this first simulation considers only one orientation angle of the element, the time required to gather the isofrequency dispersion data will be multiplied by the step size of the orientation angle variable when building the full database of isofrequency dispersion data. Multiple simulations of varying step sizes for the phase shifts have been run to find an optimal step size for the phase shifts. As expected, the clarity of the isofrequency dispersion curves increased with the step size, as did the simulation time increase. The step sizes determined to be best trade-off between clarity and simulation time are as follows:

- Py measured from -90 deg to 90 deg, step size of 5 deg, 37 steps
- Px measured from -90 deg to 90 deg, step size of 5 deg, 37 steps\
- Total Data Points = 37 * 37 = 1369

With this number of data points, the simulation took about 9 hours to complete. However, the time it takes for a simulation to complete will vary depending on the processor and RAM limit of the machine used. The better the processor and the more RAM, the less time it will take to complete a simulation.

Section 4: Unit Cell Simulation Results

Using the simulation setup and Optimetrics parameters described in the previous sections, the following data is obtained for a unit cell where the metallic cylinder is at an orientation angle of 30 degrees.

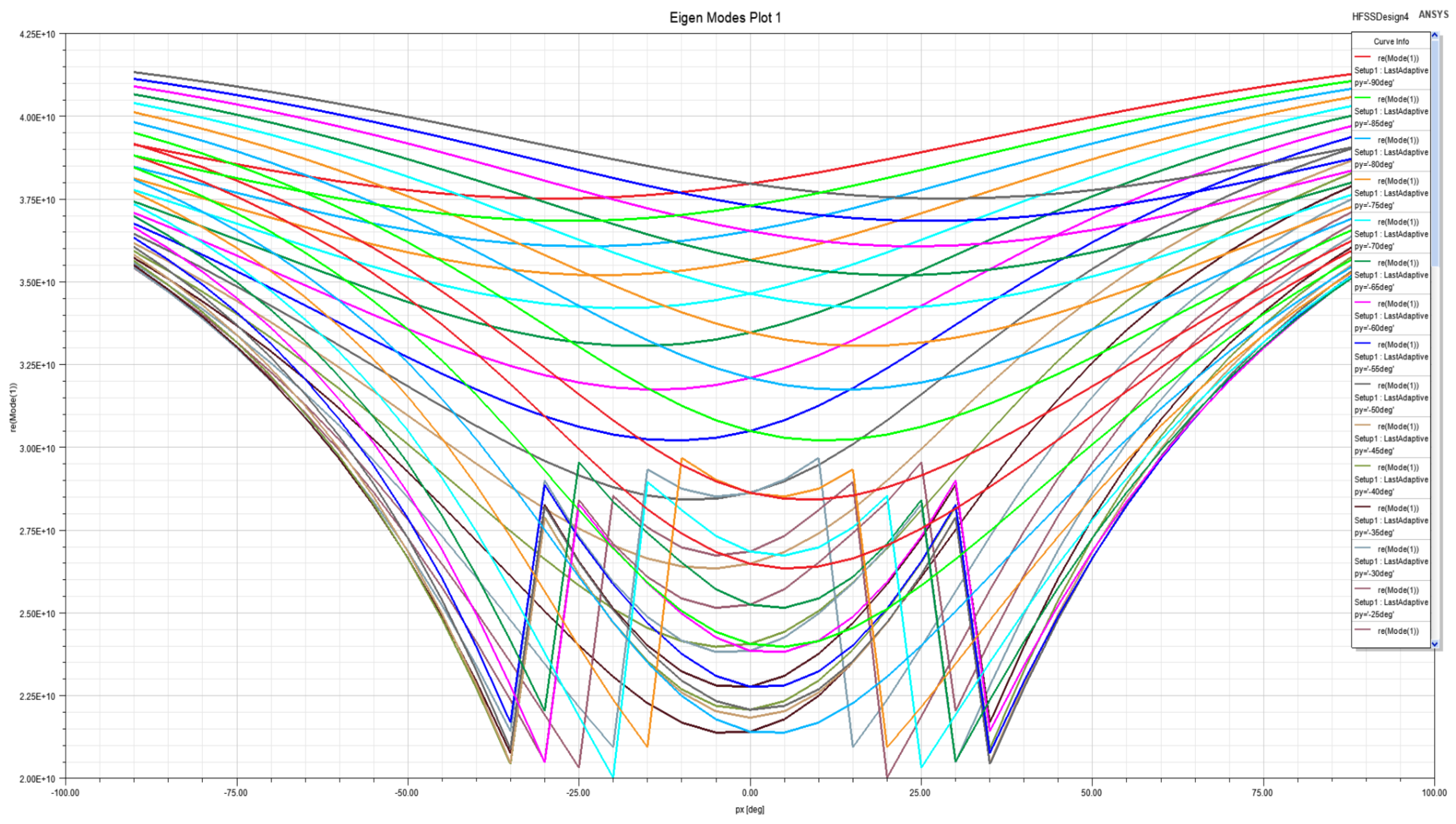


Fig. 3-4. Rectangular plot of phase shift data for Psy = 30°. Each curve represents one phase shift in the y direction, X-axis represents the phase shift in the x direction, and the Y-axis represents frequency in gigahertz.

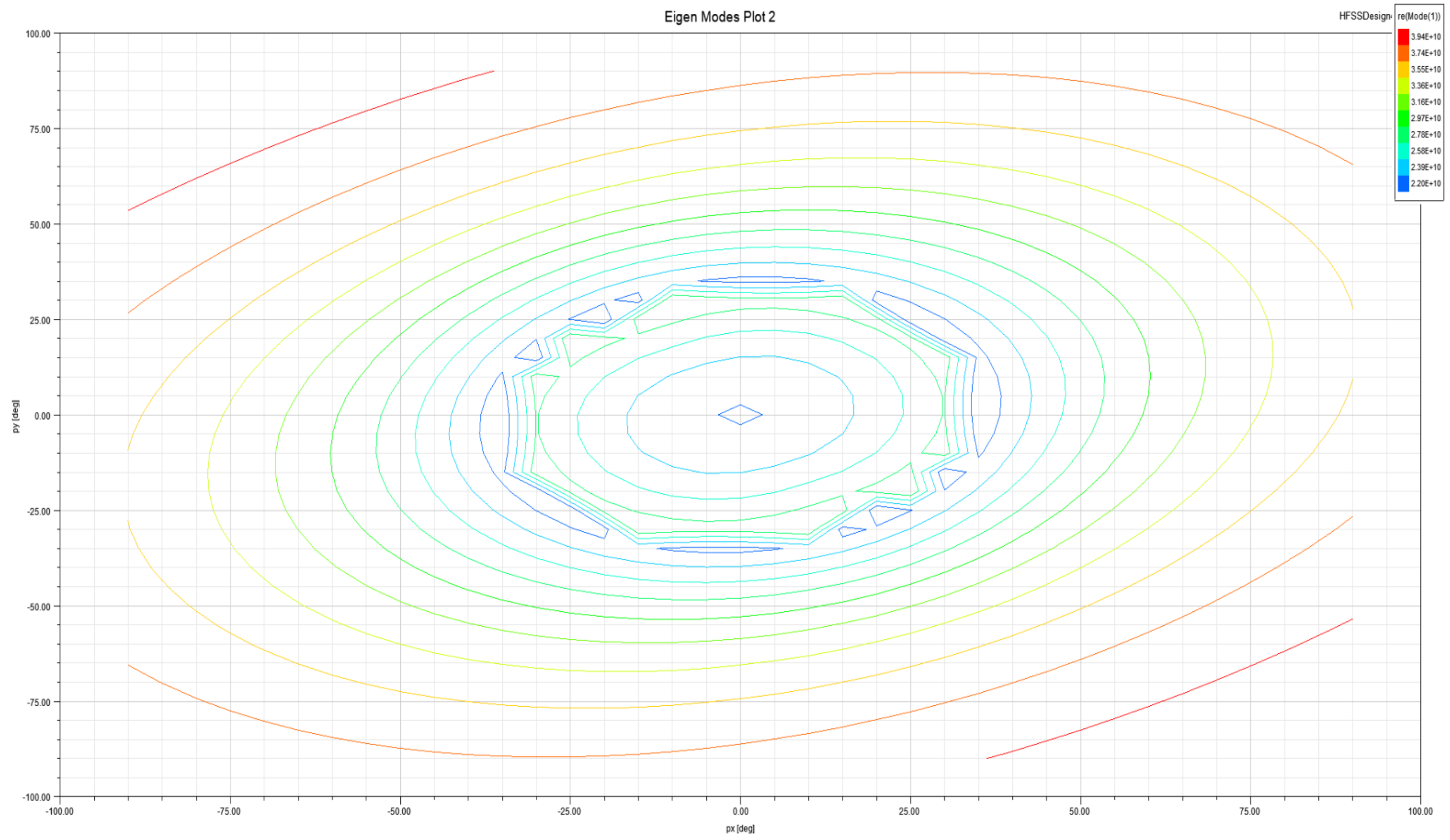


Fig. 3-5. Isofrequency dispersion contour plot generated in HFSS. X and Y axes represent the Px and Py, respectively. Colors represent frequency in gigahertz. This plot uses the same data as presented Fig. 3-4.

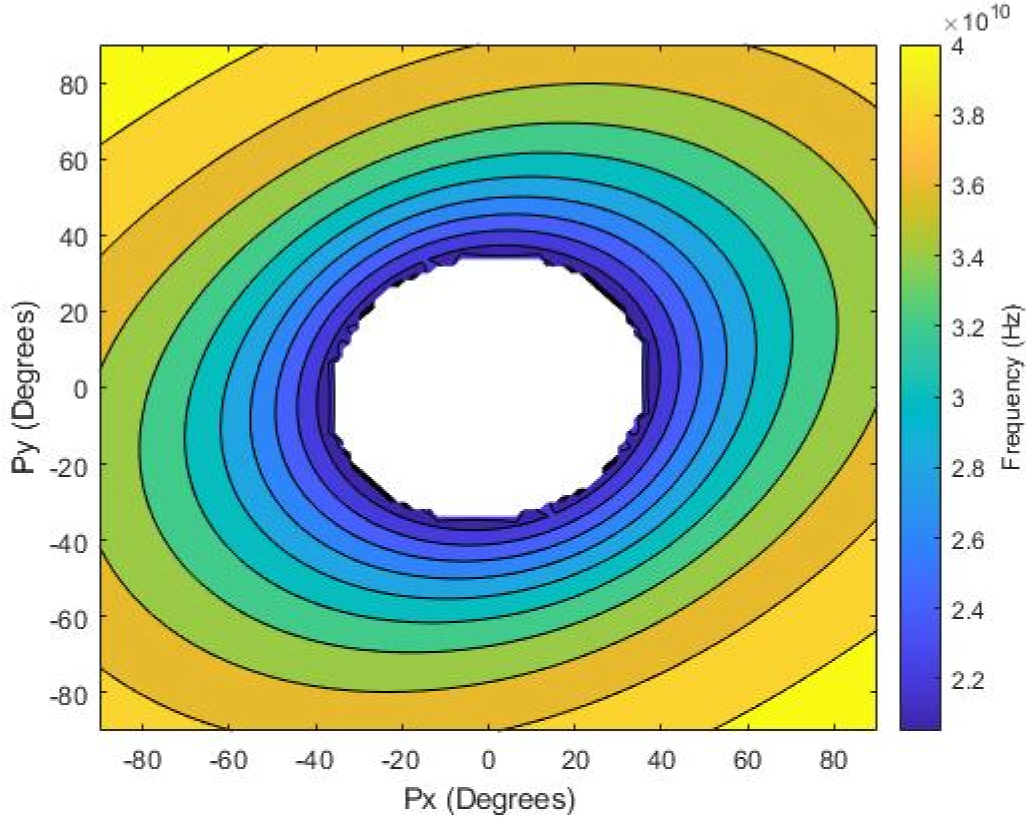


Fig. 3-6. Plot of isofrequency dispersion contours for elliptical geometry, generated using MATLAB. Same data as previous figures, but remade to better match presentation of Fig. 3-2.

Looking at the data presented in the preceding figures it is mostly clear that as the magnitude of the phase shifts increase, the propagation frequency increases. However, in Fig. 3-4 some of the curves experience a sharp increase in their frequency when Px is between -35 and 35 degrees. Looking closer these curves correspond to Py values between -35 and 35 degrees, as well. This behavior is better shown in Fig. 3-5 where the change in frequency shows two inflection areas. The frequencies range from 20 GHz to 40 GHz, with the frequency decreasing until it reaches its lowest value at a magnitude of 35 degrees from the center. Continuing towards the center of the plot from here, the frequency begins to increase again until a magnitude of about 30 degrees from the center, where the frequency change shifts again and decreases. This volatility in frequency close to the center, is likely a compensation made by HFSS due to the simulation set to a minimum frequency of 20 GHz, rather than recording data points that are less than 20 GHz to be null values. Address this problem, that data was corrected in MATLAB, to better match Fig. 3-2.

Take note of the orientation of the contour curves in Fig. 3-6. Each frequency curve has an elliptical shape that is about 30 degrees from the Py = 0 line. The orientation of the isofrequency dispersion plot's curves matches the orientation of the element. This matches with the behavior of elliptic ring MTS elements in [19]. To confirm this, below is the isofrequency dispersion curve plot of an element with an orientation angle of 0 degrees.

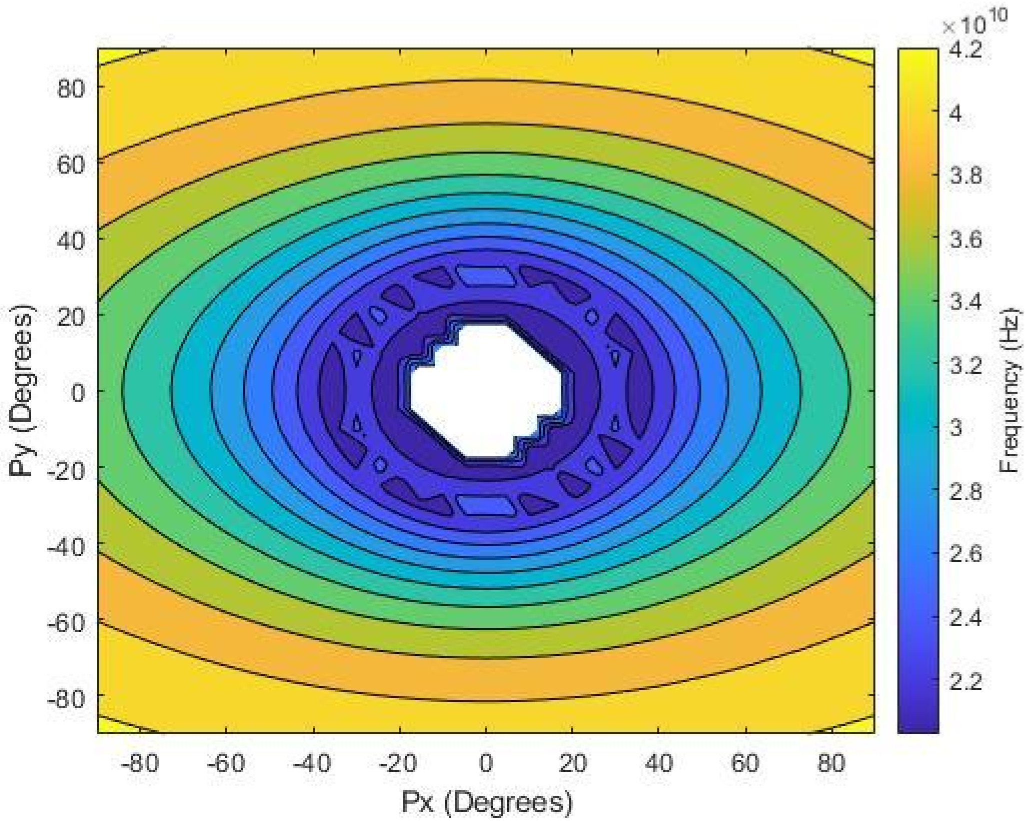


Fig. 3-7. Plot of isofrequency dispersion contours for elliptical geometry with orientation angle Psy = 0.

After proving that the unit cell simulation is working as intended and outputting the desired data, the simulation is rerun with a new Optometric for the orientation angle. The orientation angle is swept from -90 to 90 degrees with a step size of 10 degrees. Given the symmetry of the element, there is no need to sweep a full 360 degrees, as any orientation angles that are 180 degrees apart are the same structure. 90 degrees and -90 degrees are left in the sweep to provide data that confirms this symmetry effect. When this is done, all frequencies, phase shift pairs, and orientation angles are recorded into a CSV file for later use.

Chapter IV: Automated Metasurface Antenna Design Algorithm

This chapter is an overview of the code used to parse through all the data gathered in the previous chapter, apply that data to equation (1), and determine the best parameters for the layout of the antenna array.

The tensor calculations needed for equation (1) involve the multiplication of several matrices for each element individually. While these calculations can be coded in most programming languages, the software MATLAB is especially well-designed for matrix math. The software features a workspace that shows all variables in a program and allows the user to easily view the contents of matrices and arrays. This is helpful as the complexity of equation (1) lends itself to many errors being made when trying to automate it. Specifically, one can easily backtrack through matrices in the workspace to determine where calculations begin to fail expectations.

Section 1: Design Parameters and Data Inputs

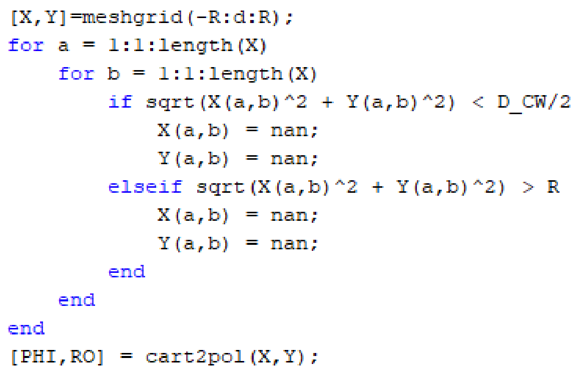
This section outlines the parameters of the MATLAB code and their purpose. Unless stated otherwise, parameters match the values listed in section IV of [1].

- X0 - Average Reactance. This is expressed as a ratio of free space impedance which is rounded to 377 Ω for this code.
- M - Modulation Index. This will directly affect the values of each tensor matrix as described in equations (5-7).
- d - Side length of the unit cell. This value will determine how many elements are present in the antenna. The smaller this number is, the more elements are present.

- f0 - Center Frequency. This value will determine the wavelength, but more importantly, it will reduce the number of time equation (1) must be run by only looking at data points close to this frequency.
- f_Offset - Frequency Offset. This will determine the acceptable range of frequencies to consider when determining which data sets from the unit cell analysis to apply to equation (1). How strict this value can be will depend on the step size used in the unit cell analysis. Setting this value too low will cause the code to either fail to find a data set, or will greatly restrict the number of data sets considered. For this code the value is set to allow any data sets within 1 GHz of the center frequency.
- lambda - Wavelength. This will determine the radius of the antenna and the wavenumber.
- R - Radius of the antenna. This will determine how far from the center elements can be placed.
- D_CW - Diameter of waveguide opening. This will determine how close to the center elements can be placed.
- k - Wavenumber. Used in equation (1) and determines the value of a_z.
- B_SW - Wavenumber for average reactance. Determines the value of a_z and it can also be used to calculate the period of modulation.
- P - Period of Modulation. This will directly affect the values of each tensor matrix as described in equations (5-7).
- Simulated_Data - This is the data extracted from simulations performed in chapter 2. It contains all of the propagation frequencies and the phase shift pairs and orientation angles that caused them.

Section 2: Function Generating Elements Grid

The layout of the antenna is determined by three factors: the radius of the antenna, the diameter of the waveguide opening, and the size of the unit cell. The radius of the antenna and the diameter of the waveguide opening decide how much surface area is available on the antenna for elements to be placed on. If the waveguide opening is made larger, then less area is available in the center of the antenna. The size of the unit cell decides the spacing between elements on the array. The smaller the unit cell is, the more closely packed the elements can be, thus increasing the total number of elements. These parameters make it necessary to define a grid of points on the antenna array where elements can be placed, before calculating the values of any elements.



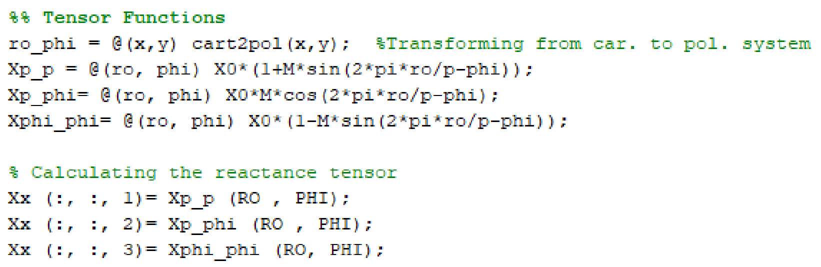
```
[X,Y]=meshgrid(-R:d:R);
for a = 1:1:length(X)
    for b = 1:1:length(X)
        if sqrt(X(a,b)^2 + Y(a,b)^2) < D_CW/2
            X(a,b) = nan;
            Y(a,b) = nan;
        elseif sqrt(X(a,b)^2 + Y(a,b)^2) > R
            X(a,b) = nan;
            Y(a,b) = nan;
        end
    end
end
[PHI,RO] = cart2pol(X,Y);
```

Fig. 4-1. Function generating grid of valid points that antenna elements can occupy.

The function above creates a square grid with sides equal to twice the radius of the antenna and is divided into sections the size of the unit cell; the center of these sections are where the elements are placed. The function then checks to see if any of the points fall either outside the radius of the antenna or inside diameter of the waveguide opening. Any points that meet these conditions are set equal to nan values. This results in any calculations done with these points to output a nan value, making it easier to identify which values need to be purged from the array layout. With the valid points in the meshgrid established, the grid is converted from cartesian coordinates to polar (or cylindrical) coordinates for use in the tensor equations.

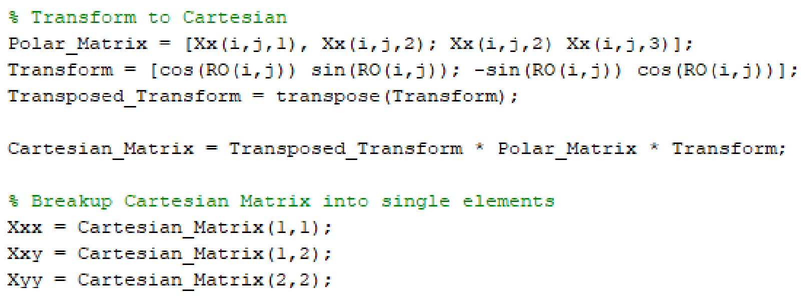
Section 3: Calculation of the Reactance Tensor and conversion to Cartesian

At each point in the formed grid there exists a tensor that represents the reactance of an antenna element. As such, it is useful to take equations (2-4) and establish them as equations in the program, as they will need to be run thousands of times.



```
%% Tensor Functions
ro_phi = @(x,y) cart2pol(x,y);  %Transforming from car. to pol. system
Xp_p = @(ro, phi) X0*(1+M*sin(2*pi*ro/p-phi));
Xp_phi= @(ro, phi) X0*M*cos(2*pi*ro/p-phi);
Xphi_phi= @(ro, phi) X0*(1-M*sin(2*pi*ro/p-phi));

% Calculating the reactance tensor
Xx (:, :, 1)= Xp_p (RO , PHI);
Xx (:, :, 2)= Xp_phi (RO , PHI);
Xx (:, :, 3)= Xphi_phi (RO, PHI);
```

Fig. 4-2. Equations (2-4) made into functions and then called to fill an array representing all tensors in the antenna within the cylindrical coordinate system.



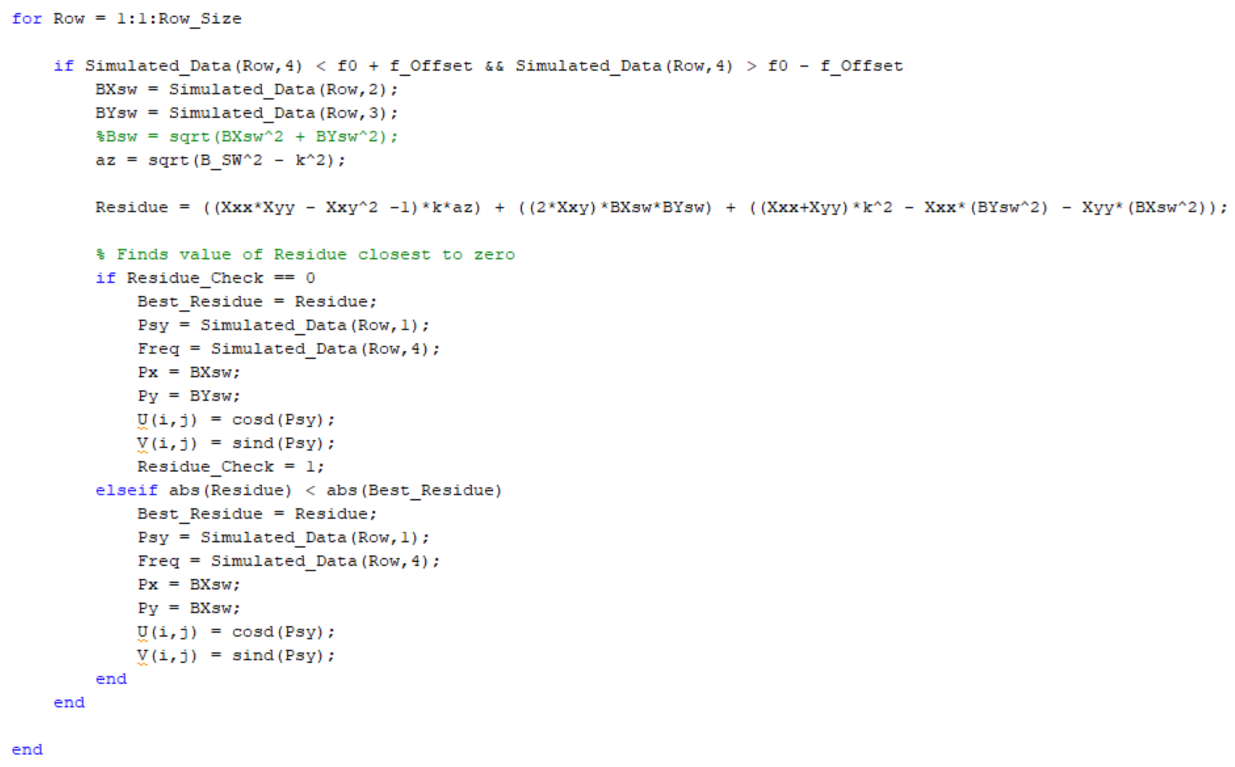
```
% Transform to Cartesian
Polar_Matrix = [Xx(i,j,1), Xx(i,j,2); Xx(i,j,2) Xx(i,j,3)];
Transform = [cos(RO(i,j)) sin(RO(i,j)); -sin(RO(i,j)) cos(RO(i,j))];
Transposed_Transform = transpose(Transform);

Cartesian_Matrix = Transposed_Transform * Polar_Matrix * Transform;

% Breakup Cartesian Matrix into single elements
Xxx = Cartesian_Matrix(1,1);
Xxy = Cartesian_Matrix(1,2);
Xyy = Cartesian_Matrix(2,2);
```

Fig. 4-3. MATLAB code that creates the tensor matrix and transforms matrices for a given set of coordinates, then performs transformation to the cartesian coordinate system. Each component of the cartesian matrix is made into separate variables.

The lines of code shown in Fig. 4-3, follow the mathematical process described in chapter 2, section 4. Two nesting for loops are used to iterate through all points in the grid. With each iteration the associated tensor is made into a matrix and the coordinate system transformation is then performed on the matrix, thus outputting the components of the tensor in the cartesian system. The components of the cartesian matrix are assigned to their own variables to make their implementation in further calculations simpler.

Section 4: Determination of Antenna Element Parameters

With the theoretical tensor values calculated and transformed to the cartesian system, there now exists all the necessary variables to make use of equation (1). For this section of the code, it is important to understand that the data collected from the unit cell analysis is organized into rows with four columns each. The columns contain the following data, in this order: orientation angle, Px, Py, and propagation frequency. To reduce the number of calculations performed, the code will only consider rows from the unit cell data that contain a propagation frequency that fall within the range defined by f_Offset, in this case 31 to 33 GHz. This also operates as a safeguard, as it's possible that rows of data with frequencies well away from the center frequency will output the lowest residual and therefore become the orientation angle that passes through. Once an appropriate frequency is found, its Px and Py values are extracted; recorded as the variables BXsw and BYsw to represent the x and y components of the surface wave number and to be easily identified with variables used in equation (1). Next, the value of az is calculated using equation (4). Finally, the residue for this combination of values is calculated.



```
for Row = 1:1:Row_Size

    if Simulated_Data(Row,4) < f0 + f_Offset && Simulated_Data(Row,4) > f0 - f_Offset
        BXsw = Simulated_Data(Row,2);
        BYsw = Simulated_Data(Row,3);
        %Bsw = sqrt(BXsw^2 + BYsw^2);
        az = sqrt(B_SW^2 - k^2);

        Residue = ((Xxx*Xyy - Xxy^2 -1)*k*az) + ((2*Xxy)*BXsw*BYsw) + ((Xxx+Xyy)*k^2 - Xxx*(BYsw^2) - Xyy*(BXsw^2));

        % Finds value of Residue closest to zero
        if Residue_Check == 0
            Best_Residue = Residue;
            Psy = Simulated_Data(Row,1);
            Freq = Simulated_Data(Row,4);
            Px = BXsw;
            Py = BYsw;
            U(i,j) = cosd(Psy);
            V(i,j) = sind(Psy);
            Residue_Check = 1;
        elseif abs(Residue) < abs(Best_Residue)
            Best_Residue = Residue;
            Psy = Simulated_Data(Row,1);
            Freq = Simulated_Data(Row,4);
            Px = BXsw;
            Py = BXsw;
            U(i,j) = cosd(Psy);
            V(i,j) = sind(Psy);
        end
    end

end
```

Fig. 4-4. Function applying theoretical tensor values and data acquired from unit cell analysis to determine the best orientation angle for an antenna element.

Each time the residue is calculated, it passes through two if statements. The first checks if the residue equals zero, as this is the best possible value that can be achieved, if this check is passed, then the calculations for this element are immediately ended and orientation angle for this instance is taken. In reality this check is intended as an error catch, as after multiple tests of this code the lowest residuals had a value in excess of 1 million. So if Residue does equal zero, one should assume there is an error somewhere in the equation or in the data collected. The second if statement checks if the current residue value is closer to zero than the current closest residue, called Best_Residue. If so, then its values are recorded. If not, the recorded values remain unchanged. It is recommended that before running the section of code shown in Fig. 4-4 that a user initializes the Best_Residue variable to be arbitrarily large. Otherwise when running calculations for the next element, there is a risk the lowest residue value from the previous element is not beaten. Once the lowest Residue is determined, all values used to calculate the lowest Residue are added to an array representing all the elements. This array is not what will be exported from MATLAB for use in generating the antenna, but instead serves as a list of all important data that a user can reference for the purpose of debugging or data analysis.

Section 5: Data Verification and Curation

Looking inside the if statements in Fig. 4-4, there are two variables called U and V, these together represent a vector that will be useful in checking the layout of the antenna in MATLAB, using a function called Quiver [20]. Quiver will generate a plot of arrows on a grid of given XY coordinates, and the arrows will have a direction determined by U and V. U is the horizontal component of the vector and is obtained by taking the cosine of the orientation angle. V is the vertical component of the vector and is obtained by taking the sine of the orientation angle. Since only the direction is important for our purposes, the value of the vector's magnitude is one for simplicity. With this function, the layout of all elements in the antenna array can be viewed immediately, allowing for verification of the code's performance. The quiver plot in Fig. 4-5 shows a layout that strongly resembles that of the antenna shown in Fig. 1-1. This means the calculations performed in this code can be considered reliable and worthy of being used in the next step of the design flow.

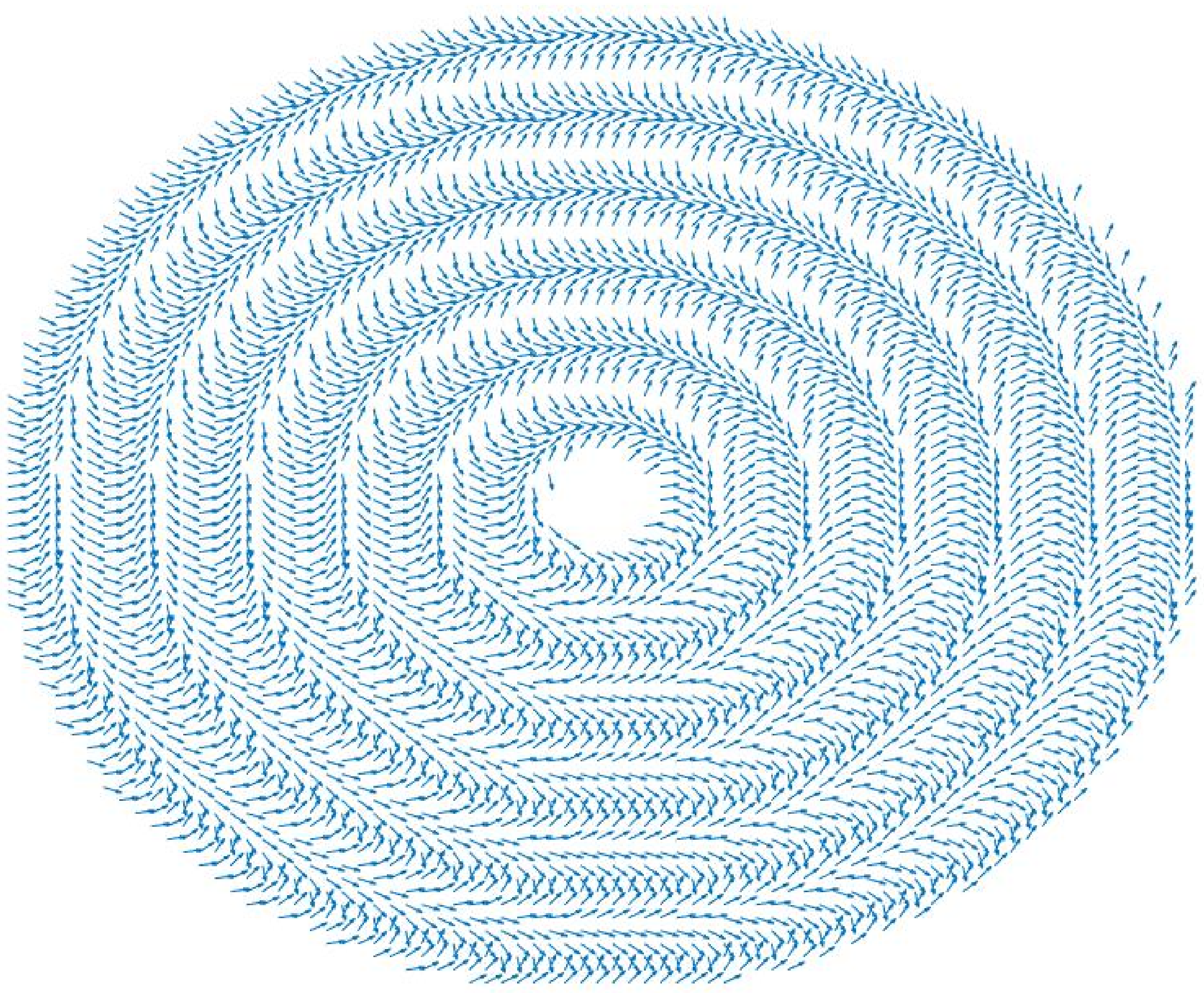
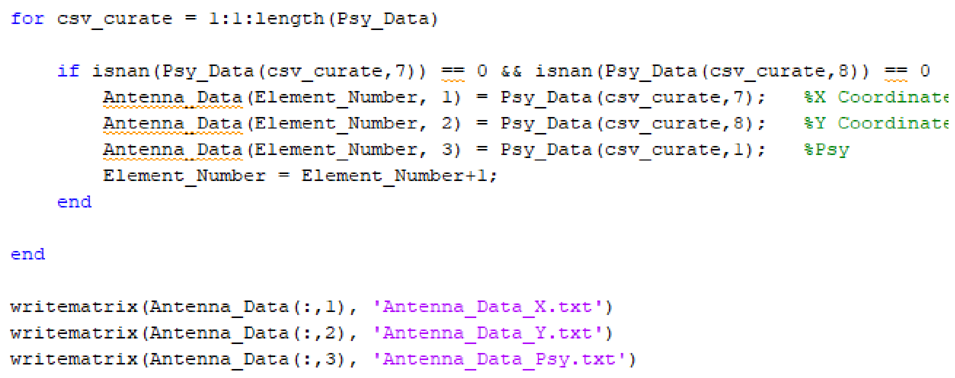


Fig. 4-5. A plot of the antenna layout generated using the Quiver function in MATLAB. Each arrow on the plot indicates an element in the array, and the direction of the arrow indicates the element’s orientation angle.

All that is left is to curate the data by extracting the values that will be needed for the generation of the antenna layout, which are the coordinates of each element and their orientation angles. Additionally, any coordinates that were changed to nan values earlier in the code should be purged. After curating the data, as in Fig. 5-5, the data is exported as three separate .txt files. It may make more sense to export all the data together as a .csv file, however, due to limitations in how data can be read from external files in the following step of the design flow, which will be explained in chapter 5, this method is found to be the easiest way of making the data usable.



```
for csv_curate = 1:1:length(Psy_Data)

    if isnan(Psy_Data(csv_curate,7)) == 0 && isnan(Psy_Data(csv_curate,8)) == 0
        Antenna_Data(Element_Number, 1) = Psy_Data(csv_curate,7);    %X Coordinate
        Antenna_Data(Element_Number, 2) = Psy_Data(csv_curate,8);    %Y Coordinate
        Antenna_Data(Element_Number, 3) = Psy_Data(csv_curate,1);    %Psy
        Element_Number = Element_Number+1;
    end

end

writematrix(Antenna_Data(:,1), 'Antenna_Data_X.txt')
writematrix(Antenna_Data(:,2), 'Antenna_Data_Y.txt')
writematrix(Antenna_Data(:,3), 'Antenna_Data_Psy.txt')
```

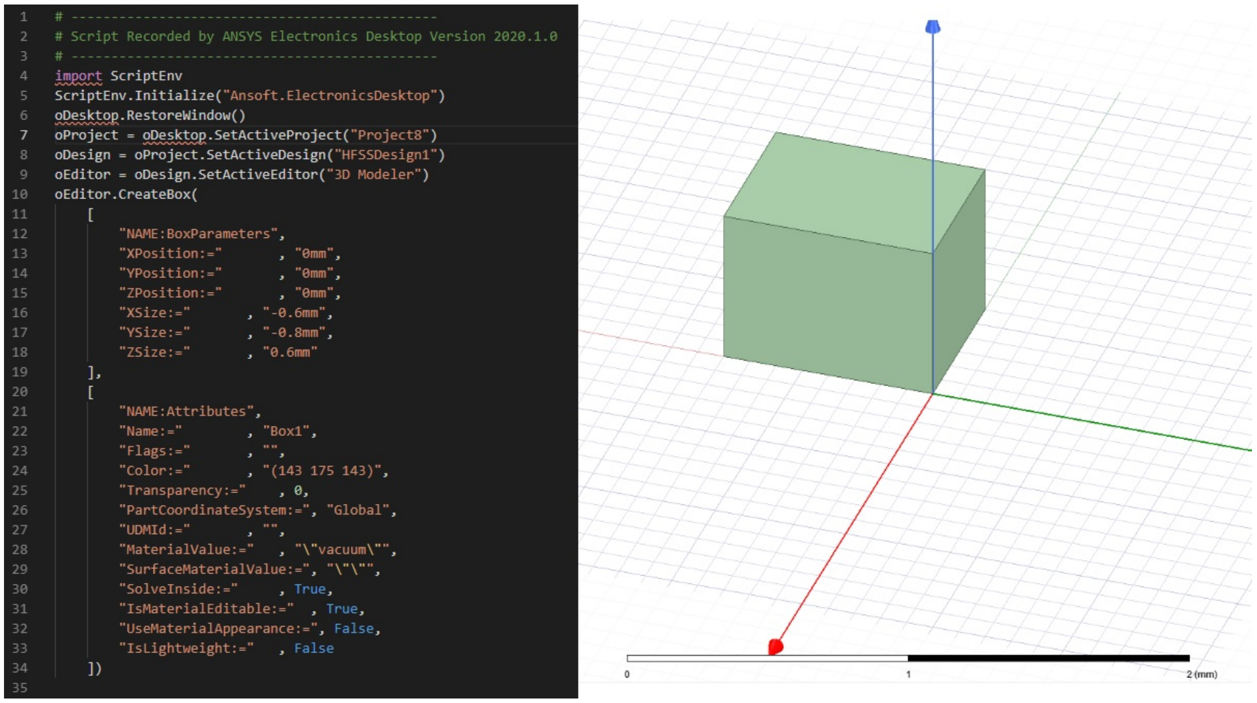
Fig. 4-6. Function for curating data of coordinate points with nan values and exporting for use in antenna generation.

Chapter V: Parametric Metasurface Element Generation

Attempting to create all of the elements of a MTS Antenna in a simulation software is tedious and prone to mistakes when done manually. This is especially true when the number of elements can scale to thousands, with each having varying directions for their structures. Therefore, it is best to establish a means of automating this process. This chapter demonstrates a method for automating the creation of 3D model antenna layouts within HFSS.

Section 1: Geometry Automation through Python Scripts

When creating models in HFSS, a user only needs to select a relevant shape tool, place the object in the modeling window, and then set the parameters of the model. What is not seen, is that each of these actions performed by the user are scripts run in the background of HFSS, in Python or Visual Basic. All Geometries made in HFSS can be represented as objects in python scripts that can be downloaded directly from HFSS. The software allows users to record the python script being generated in the background of HFSS as the user builds a geometry. When the user has finished making their desired geometry, they may download the python script. Fig. 5-1 shows one such python script generated in HFSS and the geometry it was generated from.



```
# ---------------------------------------------
# Script Recorded by ANSYS Electronics Desktop Version 2020.1.0
# ---------------------------------------------
import ScriptEnv
ScriptEnv.Initialize("Ansoft.ElectronicsDesktop")
oDesktop.RestoreWindow()
oProject = oDesktop.SetActiveProject("Project8")
oDesign = oProject.SetActiveDesign("HFSSDesign1")
oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateBox(
    [
        "NAME:BoxParameters",
        "XPosition:="       , "0mm",
        "YPosition:="       , "0mm",
        "ZPosition:="       , "0mm",
        "XSize:="       , "-0.6mm",
        "YSize:="       , "-0.8mm",
        "ZSize:="       , "0.6mm"
    ],
    [
        "NAME:Attributes",
        "Name:="        , "Box1",
        "Flags:="       , "",
        "Color:="       , "(143 175 143)",
        "Transparency:="    , 0,
        "PartCoordinateSystem:=", "Global",
        "UDMId:="       , "",
        "MaterialValue:="    , "\"vacuum\"",
        "SurfaceMaterialValue:=", "\"\"",
        "SolveInside:="     , True,
        "IsMaterialEditable:="  , True,
        "UseMaterialAppearance:=", False,
        "IsLightweight:="    , False
    ])
```

Fig 5-1. (Left) Sample script generated from "Record Script" function in HFSS. Shows geometry spatial parameters in the first set of brackets, followed by material properties in the second set of brackets. (Right) Geometry created in HFSS when script is run.

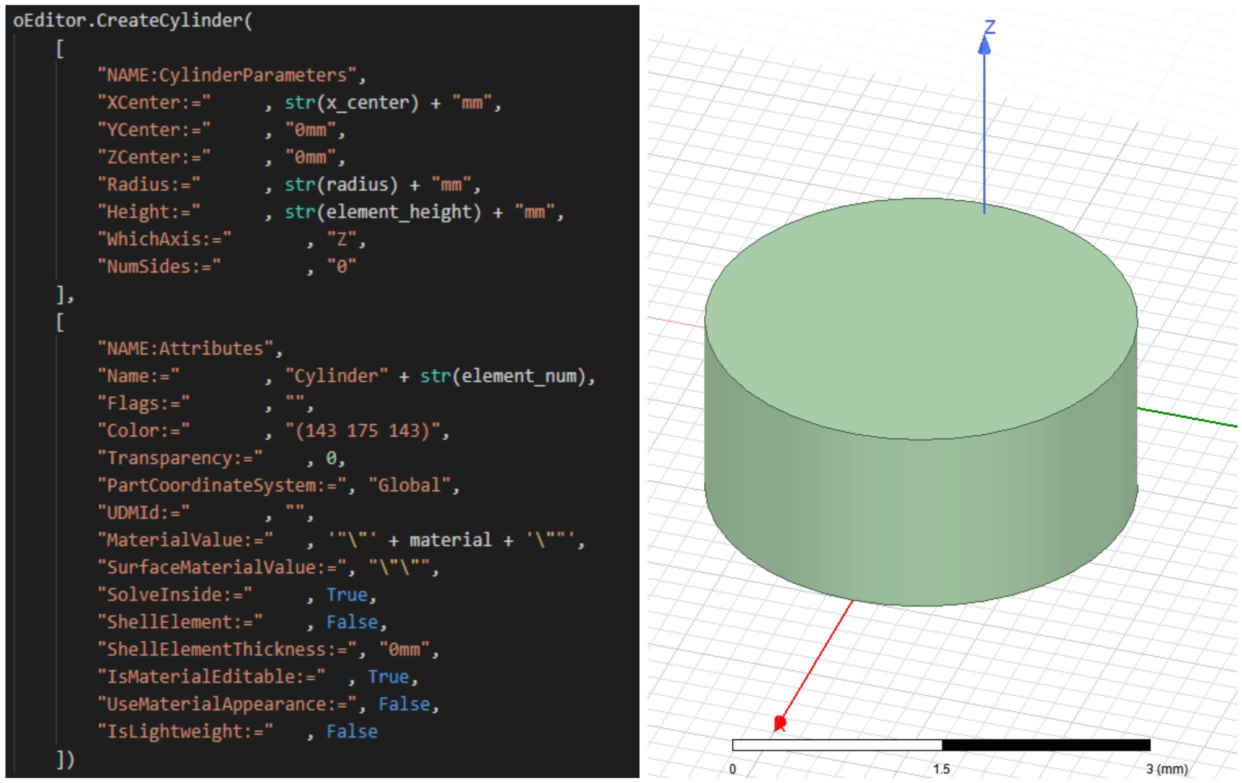
There are two key aspects of the code to discern from this example script. The first is that the script must be initialized to Ansoft Electronics Desktop and then have the project folder, design file, and editor defined so that the script knows where to place the geometry. The second is that the script constructs the object with two distinct parameter sets: BoxParameters and Attributes. The BoxParameters set the starting position of the box, relative to the origin, shown in Fig. 5-1 and the lengths of each dimension. The values in under BoxParameters can be substituted for variables, allowing one to define the position and dimensions of geometry for an entire array of geometries. The Attributes define the name of the object, its color, transparency, its coordinate reference point, type of material, and other modifiers the user may find pertinent to their simulation. Understanding the functionality of these allows one to code their own script and generate models without the use of the HFSS' built-in 3D modeling tools. This can be especially useful when designing devices with repetitive sections, such as a metasurface antenna.

As with most python scripts certain libraries must be imported for the code to operate. The libraries imported into this code are: ScriptEnv, math, and csv. ScriptEnv is pulled from the example recorded scripts made by HFSS, all functions performed by HFSS when creating and editing geometries stem from this library. The math library contains a variety of mathematical operations, as callable functions, that are not immediately available in standard Python. Lastly, the csv library provides functions for reading and extracting data from files. Those familiar with python may know that the numpy library is more commonly used for reading and writing to files. However, the numpy library will cause an error to be flagged when running a script inside HFSS. Though it may be possible to get around this error by defining the numpy library inside the HFSS environment. The csv library accomplishes the needed function from Numpy does require a somewhat unusual approach for organizing the data generated in the MATLAB script.

To read the coordinates and orientation angle for the antenna elements, each variable set had to be contained within its own .txt file, where each value is placed on a separate line. This line-by-line separation ensures that only the numerical value of each variable is read into the python code and no additional values like commas or spaces are included in the returned value. It is possible to record all values from the MATLAB script into one .txt file, and simply code the python script to retrieve values in the correct order, but the method of using three separate files was used here to make it easier to check data points should any errors be encountered when running the script.

Section 2: Create Element Function

This section outlines how the python script builds any given element of the antenna array. The proceeding process was performed once using 3D modeling tools in HFSS. The code was then extracted and modified to make it repeatable.

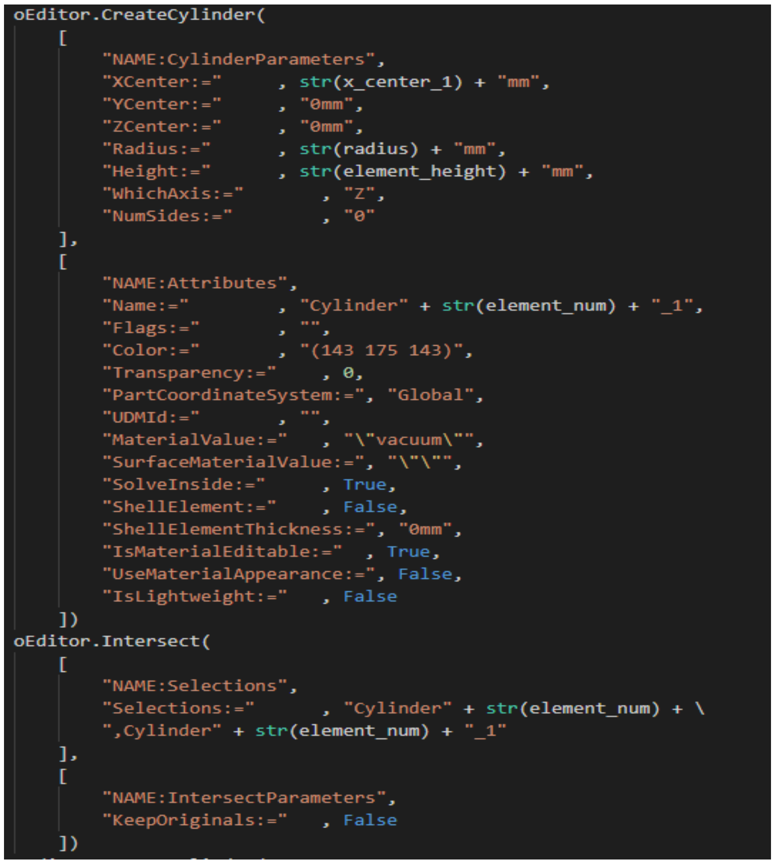


```
oEditor.CreateCylinder(
    [
        "NAME:CylinderParameters",
        "XCenter:="     , str(x_center) + "mm",
        "YCenter:="     , "0mm",
        "ZCenter:="     , "0mm",
        "Radius:="      , str(radius) + "mm",
        "Height:="      , str(element_height) + "mm",
        "WhichAxis:="       , "Z",
        "NumSides:="        , "0"
    ],
    [
        "NAME:Attributes",
        "Name:="        , "Cylinder" + str(element_num),
        "Flags:="       , "",
        "Color:="       , "(143 175 143)",
        "Transparency:="    , 0,
        "PartCoordinateSystem:=", "Global",
        "UDMId:="       , "",
        "MaterialValue:="   , '"\"' + material + '\""',
        "SurfaceMaterialValue:=", "\"\"",
        "SolveInside:="     , True,
        "ShellElement:="    , False,
        "ShellElementThickness:=", "0mm",
        "IsMaterialEditable:="  , True,
        "UseMaterialAppearance:=", False,
        "IsLightweight:="   , False
    ])
```

Fig. 5-2. (Left) Python script for first step in creating antenna element. Key parameters have been made into variables. (Right) Cylinder generated using code on left.

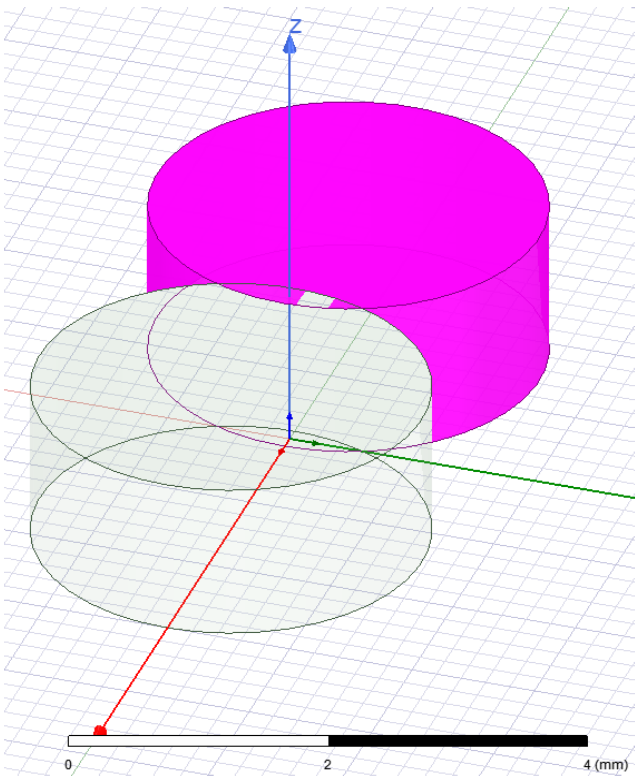
Since the elliptical cylinder shape of the antenna elements is not a standard geometry, it has to be constructed through addition and subtraction of other shapes. The method for creating the shape of the antenna element is to take the intersection of two cylinders, then round off the pointed edges of the resulting geometry. In the first step shown in Fig. 5-2, a cylinder is created that is offset from the origin, along the x-axis, such that the distance between the origin and the edge of the cylinder on the other side of the axis is equal to the length of the antenna element's minor axis. The radius of the cylinder is made sufficiently, so that the geometry created by the intersection function will have a larger major axis than the antenna element. This is done because it is easier to subtract from a geometry than add to it. The cylinders could be made so that the intersection function generates a geometry that matches the major axis of the antenna element, but this would leave pointed edges that can’t be trimmed without reducing the length of the major axis. Fig. 5-3(b) shows the same cylinder being generated, but with the offset in the opposite direction, which is followed by the execution of the intersect command, shown in Fig. 5-3(c). As the name suggests, the intersect command selects both of the cylinders, deletes everything except the overlap of the geometries, and merges the overlapping sections into one geometry.

The final step is to trim the edges and resize the major axis to match with our parameters. This can be accomplished by generating one more cylinder and performing one more intersection function. As shown in Fig. 5-4(b), a cylinder with a radius equal to the major axis is created directly on the origin, so that its center is aligned with the center of the elliptical cylinder. Another intersection trims the elliptical cylinder to the correct length and rounds its edges to match the design used in [1].



```
oEditor.CreateCylinder(
    [
        "NAME:CylinderParameters",
        "XCenter:="     , str(x_center_1) + "mm",
        "YCenter:="     , "0mm",
        "ZCenter:="     , "0mm",
        "Radius:="      , str(radius) + "mm",
        "Height:="      , str(element_height) + "mm",
        "WhichAxis:="       , "Z",
        "NumSides:="        , "0"
    ],
    [
        "NAME:Attributes",
        "Name:="        , "Cylinder" + str(element_num) + "_1",
        "Flags:="       , "",
        "Color:="       , "(143 175 143)",
        "Transparency:="    , 0,
        "PartCoordinateSystem:=", "Global",
        "UDMId:="       , "",
        "MaterialValue:="   , "\"vacuum\"",
        "SurfaceMaterialValue:=", "\"\"",
        "SolveInside:="     , True,
        "ShellElement:="    , False,
        "ShellElementThickness:=", "0mm",
        "IsMaterialEditable:="  , True,
        "UseMaterialAppearance:=", False,
        "IsLightweight:="   , False
    ])
oEditor.Intersect(
    [
        "NAME:Selections",
        "Selections:="      , "Cylinder" + str(element_num) + \
        ",Cylinder" + str(element_num) + "_1"
    ],
    [
        "NAME:IntersectParameters",
        "KeepOriginals:="   , False
    ])
```

(a)



(b)

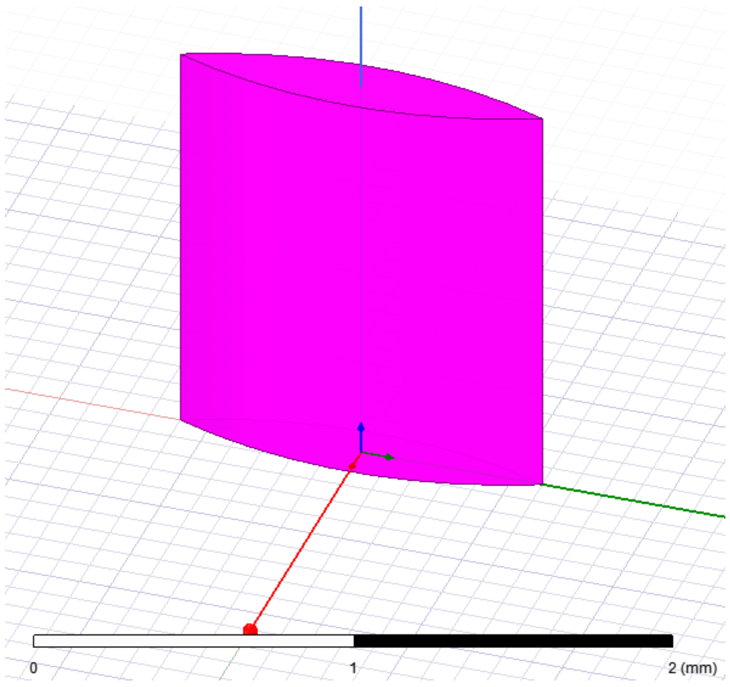
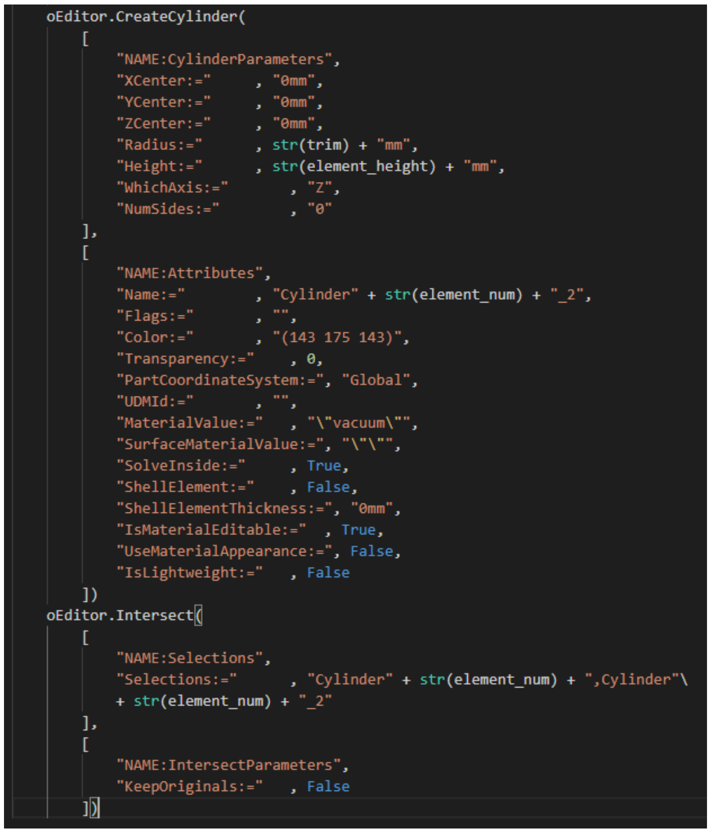
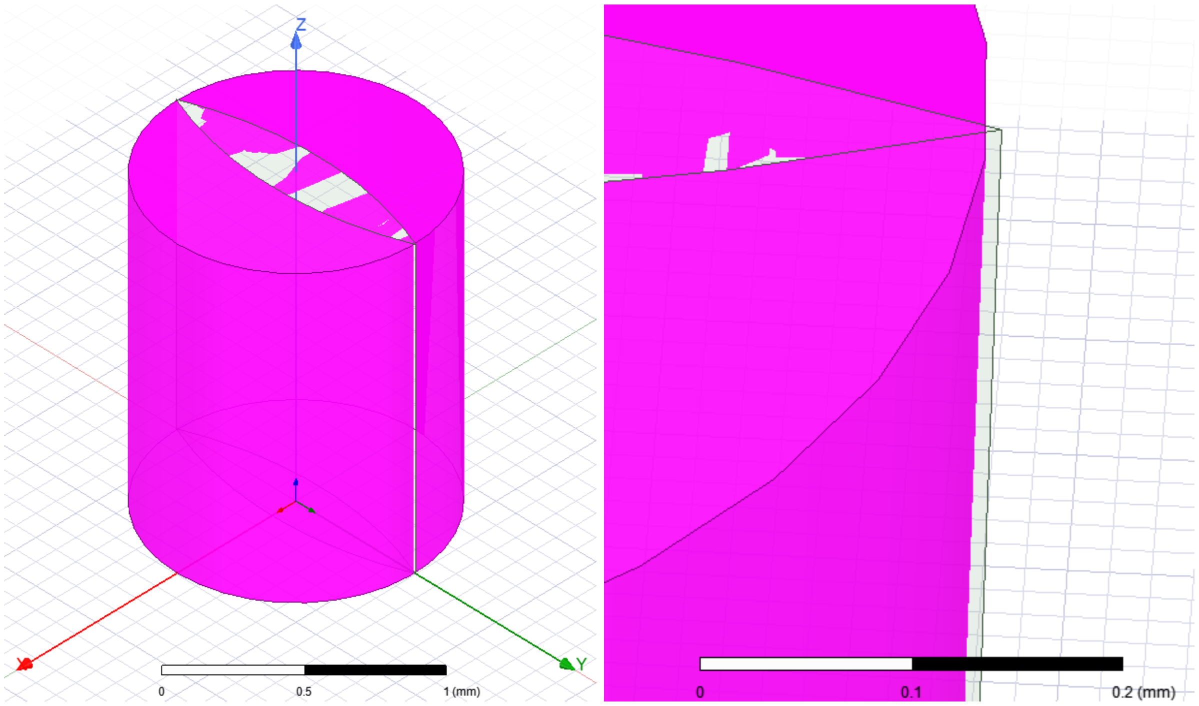
(c) 

Fig. 5-3. (a) Python script that generates a second cylinder, then performs an Intersect function. (b) Placement of the second cylinder in HFSS; the overlap with the first cylinder is an elliptic cylinder that nearly matches the antenna element. (c) Remains of HFSS geometry after Intersect function is performed.



```
oEditor.CreateCylinder(
    [
        "NAME:CylinderParameters",
        "XCenter:="     , "0mm",
        "YCenter:="     , "0mm",
        "ZCenter:="     , "0mm",
        "Radius:="      , str(trim) + "mm",
        "Height:="      , str(element_height) + "mm",
        "WhichAxis:="       , "Z",
        "NumSides:="        , "0"
    ],
    [
        "NAME:Attributes",
        "Name:="        , "Cylinder" + str(element_num) + "_2",
        "Flags:="       , "",
        "Color:="       , "(143 175 143)",
        "Transparency:="    , 0,
        "PartCoordinateSystem:=", "Global",
        "UDMId:="       , "",
        "MaterialValue:="   , "\"vacuum\"",
        "SurfaceMaterialValue:=", "\"\"",
        "SolveInside:="     , True,
        "ShellElement:="    , False,
        "ShellElementThickness:=", "0mm",
        "IsMaterialEditable:="  , True,
        "UseMaterialAppearance:=", False,
        "IsLightweight:="   , False
    ])
oEditor.Intersect(
    [
        "NAME:Selections",
        "Selections:="      , "Cylinder" + str(element_num) + ",Cylinder"\
        + str(element_num) + "_2"
    ],
    [
        "NAME:IntersectParameters",
        "KeepOriginals:="   , False
    ])
```

(a)



(b)

(c)

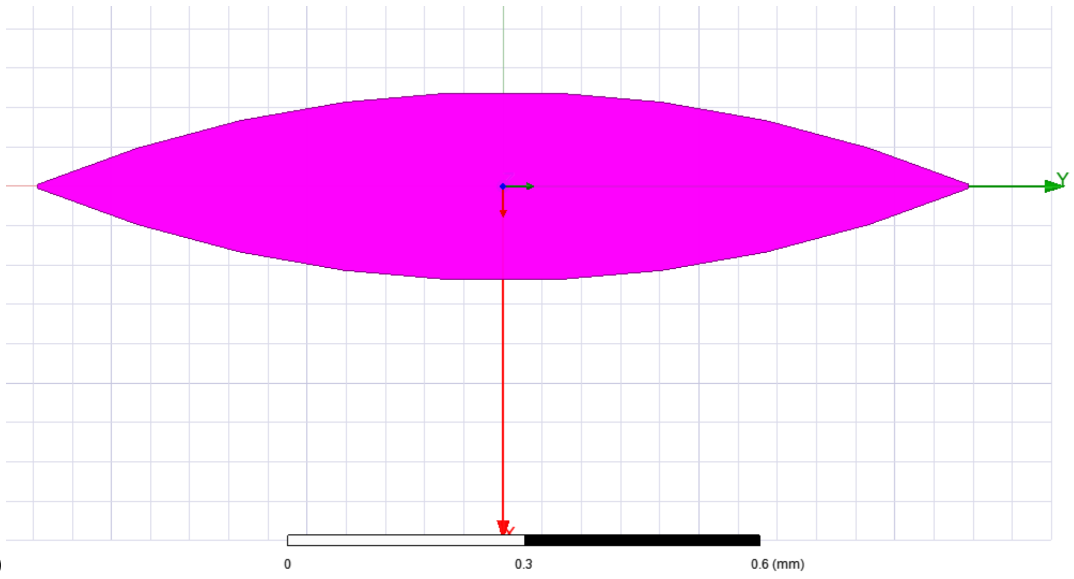


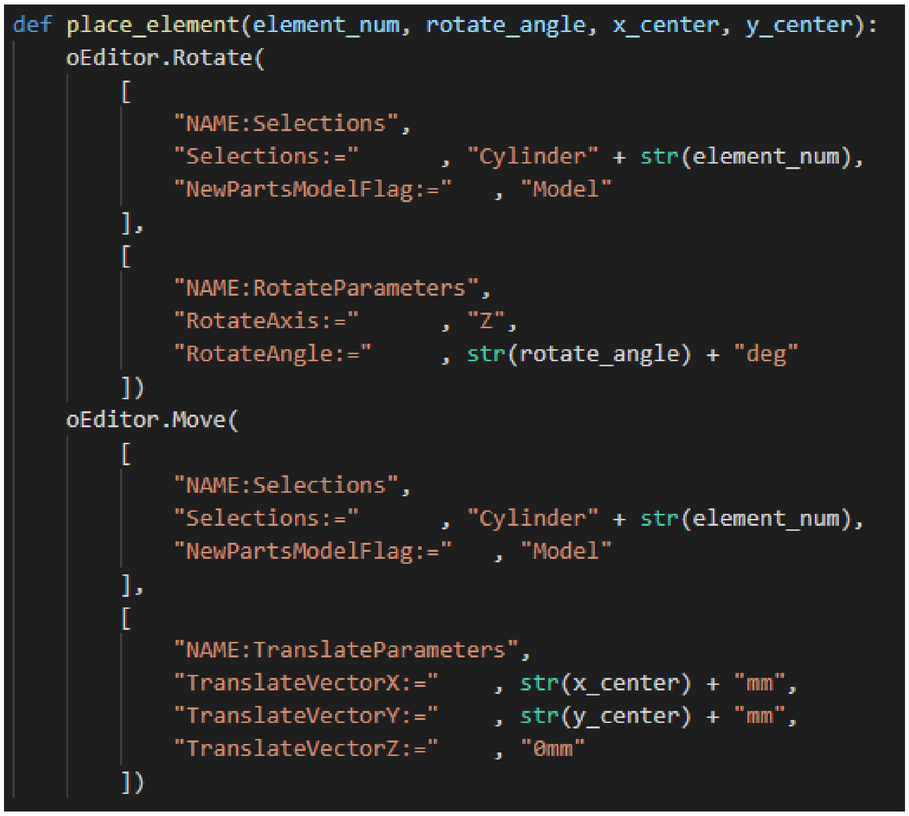
Fig. 5-4. (a) Python script that generates a third cylinder, then performs a second Intersect function. (b) Placement of the third cylinder in HFSS; the overlap between the third cylinder and the elliptic cylinder excludes the points of the elliptic cylinder. (c) Top view of the elliptic cylinder after the second Intersect function is performed.

In each block of code shown in this section, the values for element parameters are replaced by str(x_center_1), str(radius), etc. These demonstrate how to define elements through variables. Variables need to be converted from numerical value into a string type, so that it can be read and written into an element's parameters in HFSS. Note that the unit of a given value needs to be included in the parameter line. If necessary, it is possible to generate a string type value outside of the element function that includes the numerical value and unit of a parameter; in which case only the variable name needs to be placed in the parameter line without the str( ) function.

With the process for creating an element established and the method for making element parameters into variables defined, this process can now be converted into a function that will run through a loop until all elements of the array have been formed. Here the function is called create_element and it has the following parameters: element_num, element_height, major_axis, and minor axis. The parameter element_num represents which identification of the element to be generated, element_height represents the height of the element, major axis and minor axis are used to generate the cylinder radii; described earlier in this section.

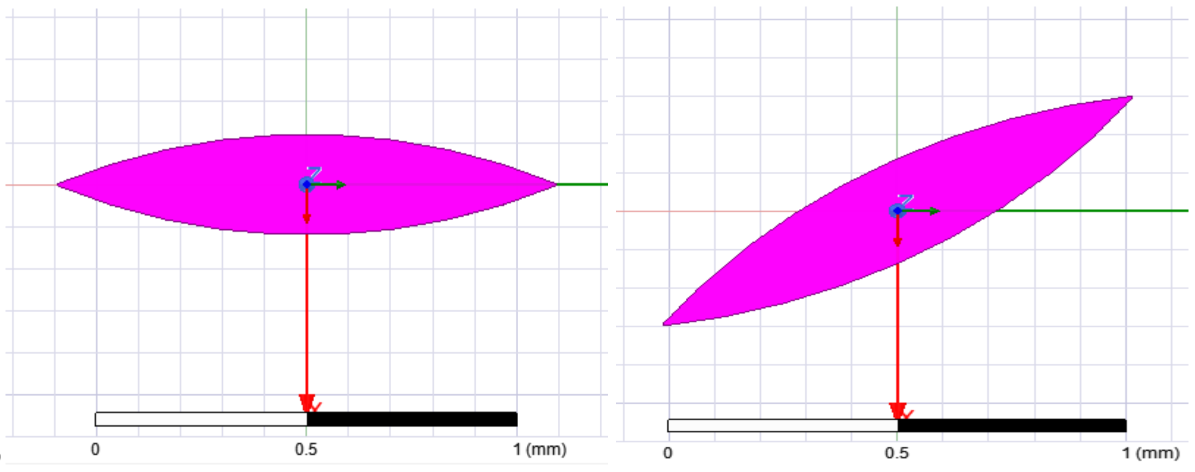
Section 3: Place Element Function

Just as important as creating the correct geometry for each element, the element must be placed correctly within the array and with the correct orientation angle. This section describes the function place_element which takes the geometry formed with create_element and moves it to a given set of coordinates, then orients it accordingly.



```
def place_element(element_num, rotate_angle, x_center, y_center):
    oEditor.Rotate(
        [
            "NAME:Selections",
            "Selections:="      , "Cylinder" + str(element_num),
            "NewPartsModelFlag:="   , "Model"
        ],
        [
            "NAME:RotateParameters",
            "RotateAxis:="      , "Z",
            "RotateAngle:="     , str(rotate_angle) + "deg"
        ])
    oEditor.Move(
        [
            "NAME:Selections",
            "Selections:="      , "Cylinder" + str(element_num),
            "NewPartsModelFlag:="   , "Model"
        ],
        [
            "NAME:TranslateParameters",
            "TranslateVectorX:="    , str(x_center) + "mm",
            "TranslateVectorY:="    , str(y_center) + "mm",
            "TranslateVectorZ:="    , "0mm"
        ])
```

(a)

(b) 

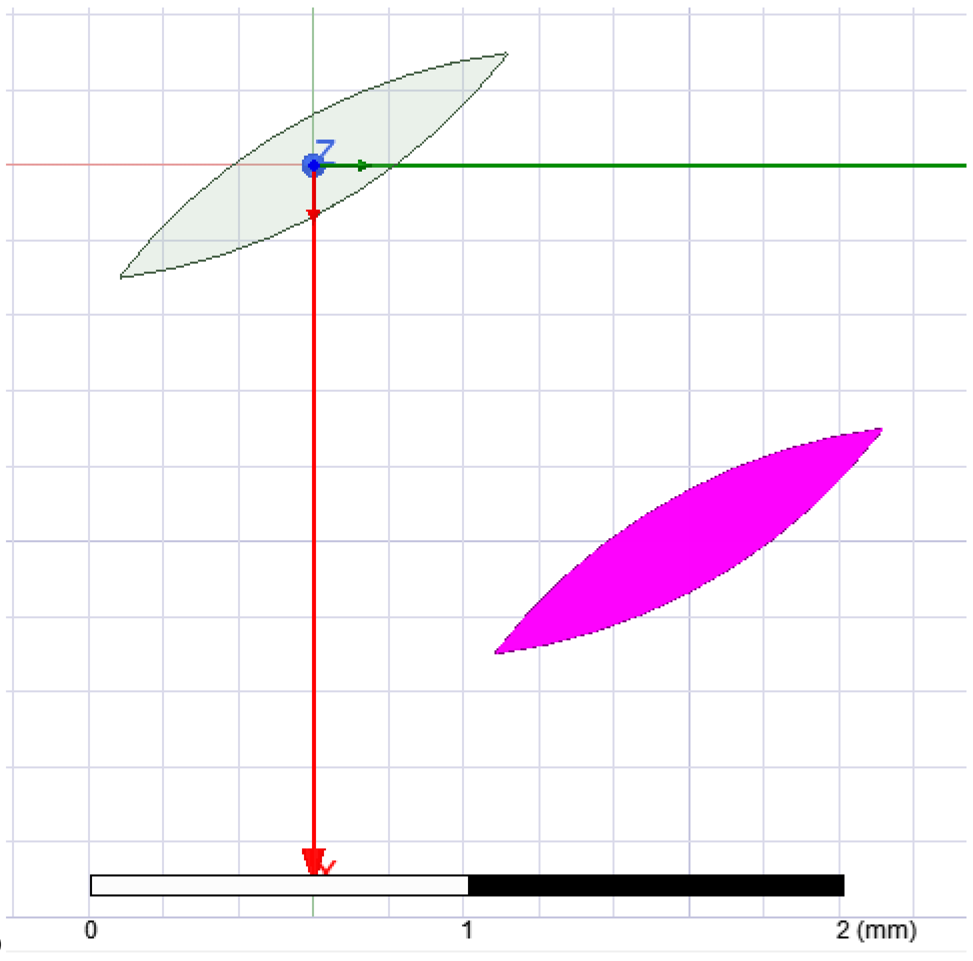
(c) 

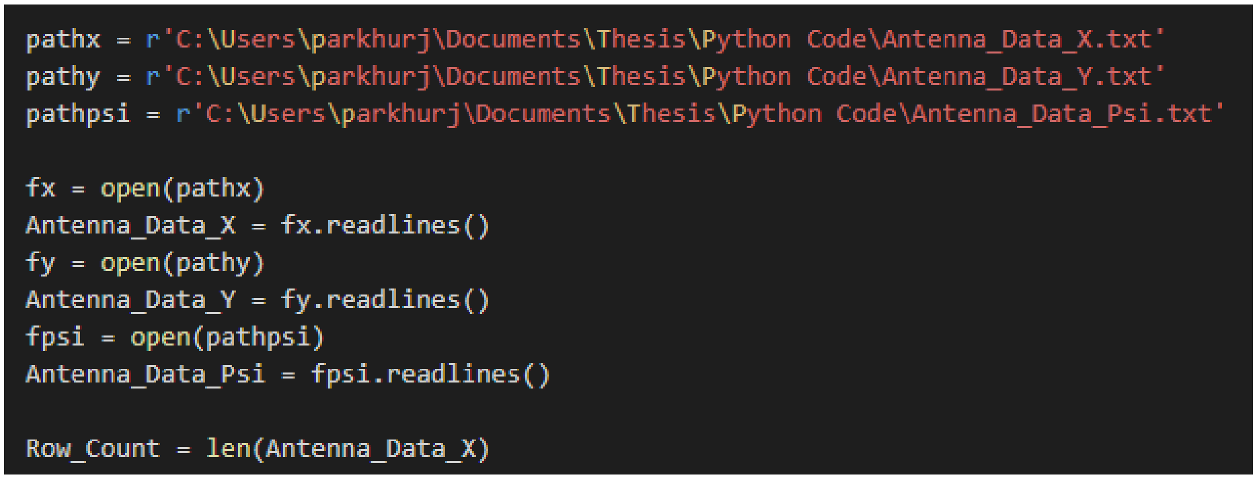
Fig. 5-5. (a) Python script of the Rotate and Move commands found in HFSS. (b) Array element being rotated to a given angle. (c) Array element moved along the XY plane.

The parameters for the is function are the following: element_num, rotate_angle, x_center, and y_center. The element_num value is the same as in the create_element function The remaining parameters are orientation angles and coordinates calculated in the Matlab code. Much like in the previous section, this function simply involves copying the python scripts for the Rotate and Move commands in HFSS, and then editing them to turn key parameters into variables. For the rotate command, the only variables needed are the element ID, to ensure the correct element is selected, and the rotation angle. Since the array is being built in the XY plane, the axis of rotation is set to the Z axis. For the move command, element ID is again needed to select the element and the coordinate variables will set where the element is translated to.

It is possible to combine the create_element and place_element functions into one function. This can be achieved by including parameters for orientation angle and coordinates into the create_element function, then adding the rotate and move commands to the end of the function. The functions were separated here to make the code easier to understand and to make troubleshooting errors in the code simpler.

Section 4: Element Map Function and Overall Code

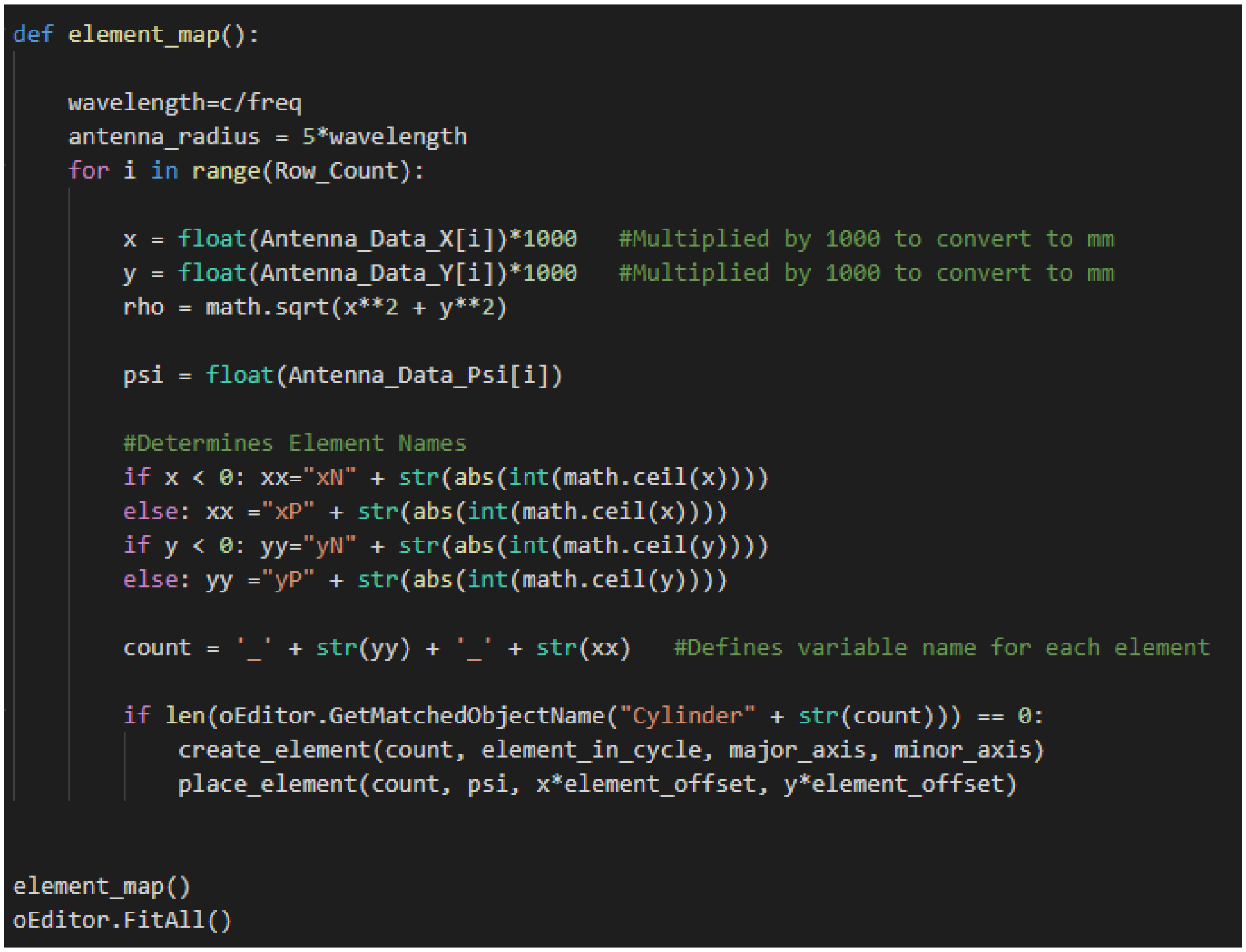
The last three features needed in the python code are a set of constant dimensions for the array, importing of data from Matlab, and a function for controlling which values are passed into the create_element and place_element functions. At the top of the code constants are defined for the diameter of the aperture, the center frequency of the antenna, the base height of the elements, and the material the antenna will be assigned in HFSS. The frequency of the antenna is used just to calculate the wavelength, and then the radius of the antenna, which is five times the wavelength.



```
pathx = r'C:\Users\parkhurj\Documents\Thesis\Python Code\Antenna_Data_X.txt'
pathy = r'C:\Users\parkhurj\Documents\Thesis\Python Code\Antenna_Data_Y.txt'
pathpsi = r'C:\Users\parkhurj\Documents\Thesis\Python Code\Antenna_Data_Psi.txt'

fx = open(pathx)
Antenna_Data_X = fx.readlines()
fy = open(pathy)
Antenna_Data_Y = fy.readlines()
fpsi = open(pathpsi)
Antenna_Data_Psi = fpsi.readlines()

Row_Count = len(Antenna_Data_X)
```

Fig. 5-6. Shows the method for importing .txt files generated in the Matlab code, as was explained in section 2 of this chapter.

The element_map function converts all of the data provided from the Matlab code into a form that can be used by the other functions. There are several reasons why these conversions are needed. The first is due to how data from MATLAB is imported, as demonstrated in Fig 5-6. All of the Matlab data is read as string values initially. The X and Y values for the elements need to be made into float values so that calculations can be performed on them. This step could be performed outside of the function, but was left in the function for the purposes of trouble shooting the calculations. The position values are multiplied by a 1000, since the HFSS functions define position values in millimeters; not having this conversion would result in X and Y values that are in micrometers. As a note, if you were to multiply the position values by 1000 in MATLAB, you would still need to define them as float values then back into string values before applying them to the MATLAB functions. This is because the method for importing these values from MATLAB has hidden spaces to appear along with the numerical values, which will cause errors to occur when HFSS tries to use these values. Converting these values to floats first, will ensure that only the numerical value is within the variable. The Psy value should also be converted to a float value for the same reason.



```python
def element_map():

    wavelength=c/freq
    antenna_radius = 5*wavelength
    for i in range(Row_Count):

        x = float(Antenna_Data_X[i])*1000   #Multiplied by 1000 to convert to mm
        y = float(Antenna_Data_Y[i])*1000   #Multiplied by 1000 to convert to mm
        rho = math.sqrt(x**2 + y**2)

        psi = float(Antenna_Data_Psi[i])

        #Determines Element Names
        if x < 0: xx="xN" + str(abs(int(math.ceil(x))))
        else: xx ="xP" + str(abs(int(math.ceil(x))))
        if y < 0: yy="yN" + str(abs(int(math.ceil(y))))
        else: yy ="yP" + str(abs(int(math.ceil(y))))

        count = '_' + str(yy) + '_' + str(xx)   #Defines variable name for each element

        if len(oEditor.GetMatchedObjectName("Cylinder" + str(count))) == 0:
            create_element(count, element_in_cycle, major_axis, minor_axis)
            place_element(count, psi, x*element_offset, y*element_offset)


element_map()
oEditor.FitAll()
```

Fig. 5-7. Shows the element map function and how the function is called in the script.

The second main reason for the conversions inside this function is to generate the naming scheme for all of the elements in the array. Each element is given a name that identifies where in the array it is located. However, since HFSS cannot read negative signs or decimal points into names, the position values are converted into strings with the following format: "_xN(X_Postion)_yN(Y_Postion)". This x and y are followed by either P or N to signify the value is positive or negative, respectively. Additionally, all values are rounded up to get rid of any decimal points. This naming scheme is not necessary for the script to function, but can be helpful when troubleshooting errors with running the script in HFSS.

You may note that this function contains a for loop that iterates through every element in the array which is controlled by the Row_Count variable rather than just iterating to the end of available data points. This allows the user to edit the for loop to only run a certain number of elements at a time. This may seem counterintuitive, however, observations from running this script show that HFSS slows down dramatically after a few hundred iterations. A user can save time in generating the full array by instead generating 1000 elements, then running the script again for the next 1000 elements.

The final line of code at the bottom of Fig 5-7 simply tells HFSS to adjust its view so that the entire antenna array is visible after it is finished generating elements.

Chapter VI: MTS 3D Geometry Automated Generation and Numerical Electromagnetic Simulation

Section 1: MTS Generation

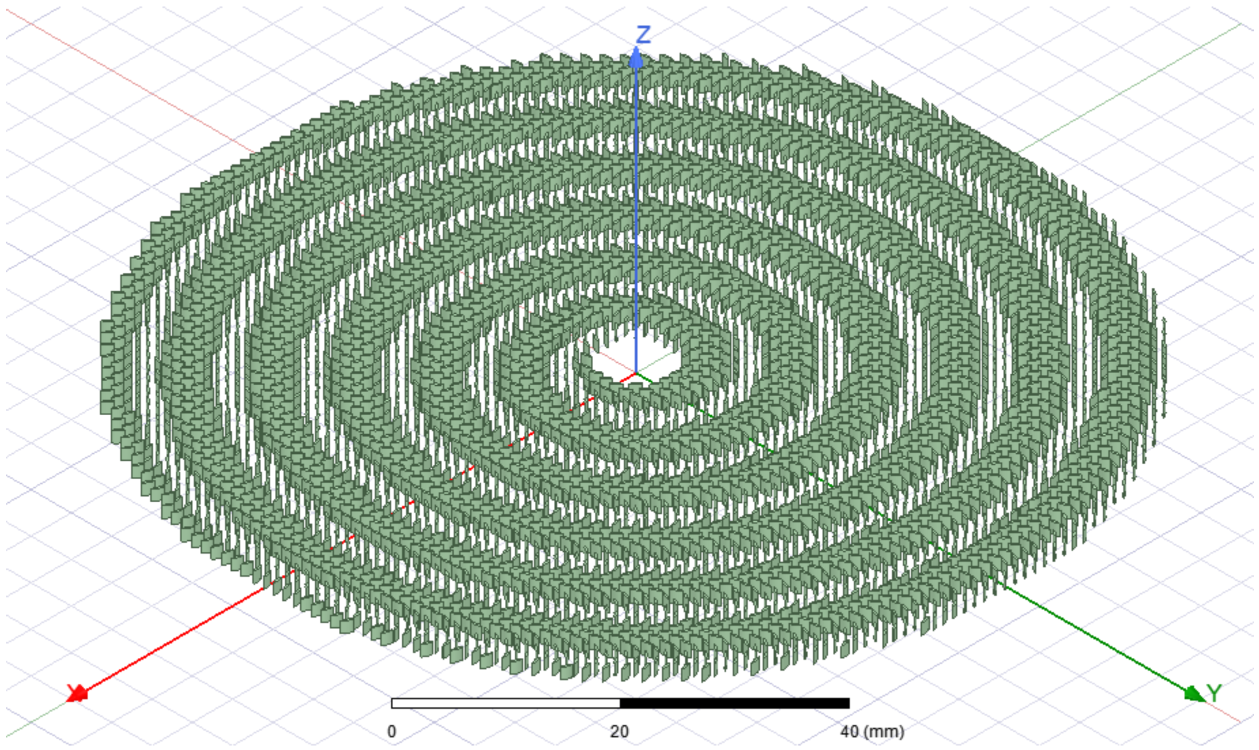


Fig. 6-1. Completed array of elements after running python scripts from Chapter 5.

When setting the script to generate all of the elements in one run, the above array takes around 100 hours to complete. The time to generate elements slows down gradually increasing from three seconds to 5 minutes per element. The slow down occurs because HFSS autosaves when an object is added to or edited in the design. As more elements are added, the more objects HFSS has to save at once. After running multiple generations, it was found that breaking up the script into smaller increments, for example 1000 elements at a time, greatly reduced the time required to generate the full array. Running the script multiple times appears to reset the time it takes to autosave the design. This may mean that HFSS is able to compress existing objects into smaller amounts of data to make for less intensive use on computer resources, but can’t do so while running a script. Using this method, the array could be generated within a day, however this comes at the cost of the user having to be present to reset the script. It may also be possible to suspend the autosave feature in HFSS.

It was also found that the entire array is slightly off-center by about 50 micrometers in the positive x and the positive y directions. This offset is enough to affect the antenna gain and should be corrected before running any analysis in HFSS. This offset can be correct by selecting all elements and using the Move function in MATLAB. This offset is due to rounding issues in the python script, it is possible to create a hardcoded value to correct this offset, but this was not done as this offset would change given different antenna parameters.

Section 2: Simulation Setup

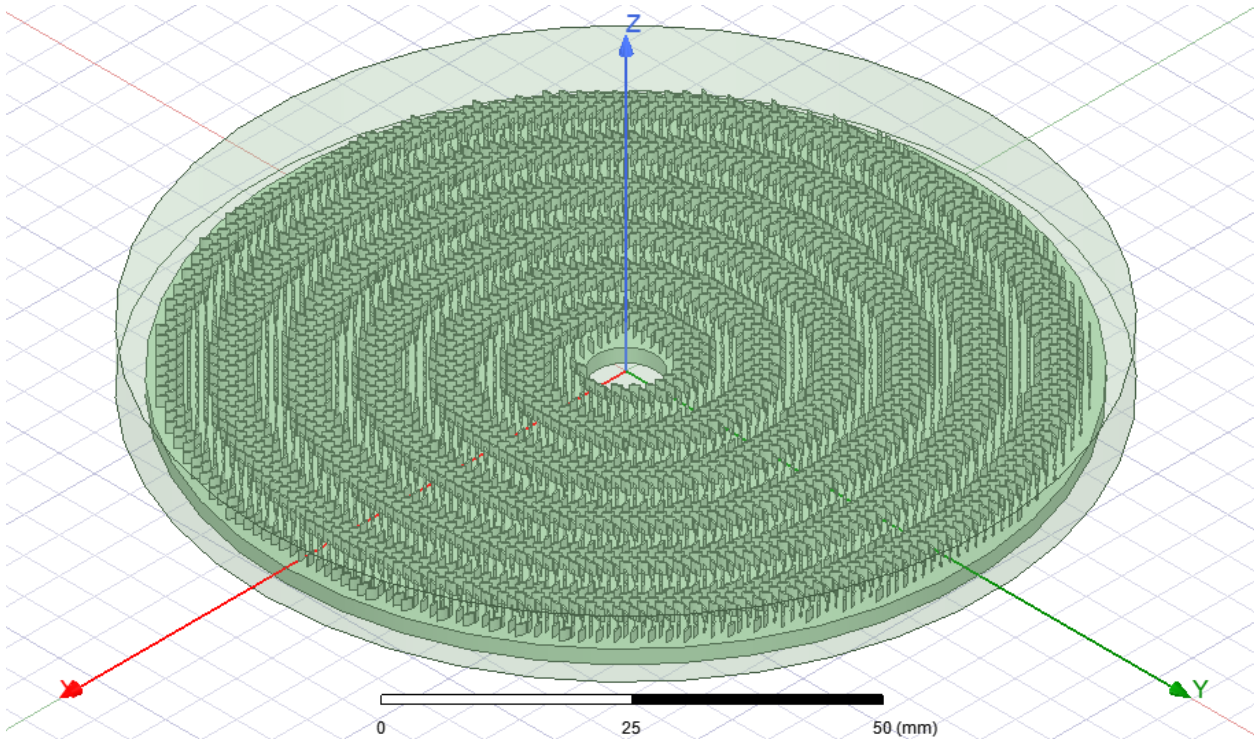


Fig. 6-2. Added mounting plate for elements array and radiation box to complete requirements for running analysis.

The final step before measuring the performance of the antenna is to mount the array onto a plate and add a radiation box. The plate has a thickness of 1.9 cm and is just slightly larger than the radius of the antenna. There is a hole for the aperture that was cut out of the center. A few elements overlapped with the aperture and needed to be deleted from the array. After this all elements and the plate were combined into one object using the Unite function in HFSS. The radiation box extends from the bottom of the plate to one wavelength above the array and has a radius that is one quarter wavelength longer than the array. Next, a waveport was assigned to the bottom side of the radiation box. Finally, a sweep setup was created with a frequency sweep from 20 GHz to 40 GHz.

Section 3: Results

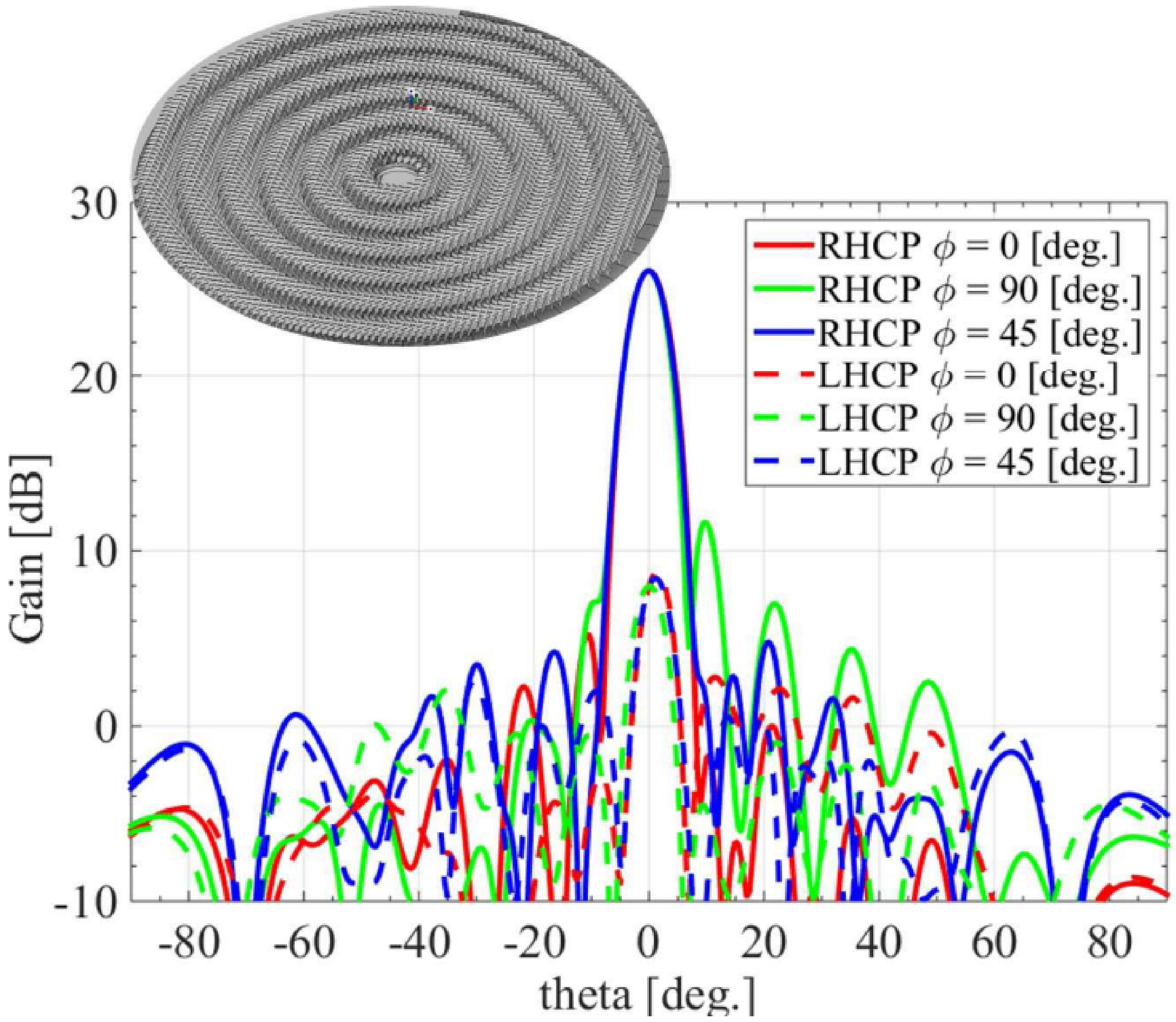


Fig. 6-3. Gain plot of antenna design from [1]. Measurements made with 45 degree steps in phi, with data taken from -90 to 90 degrees in theta.

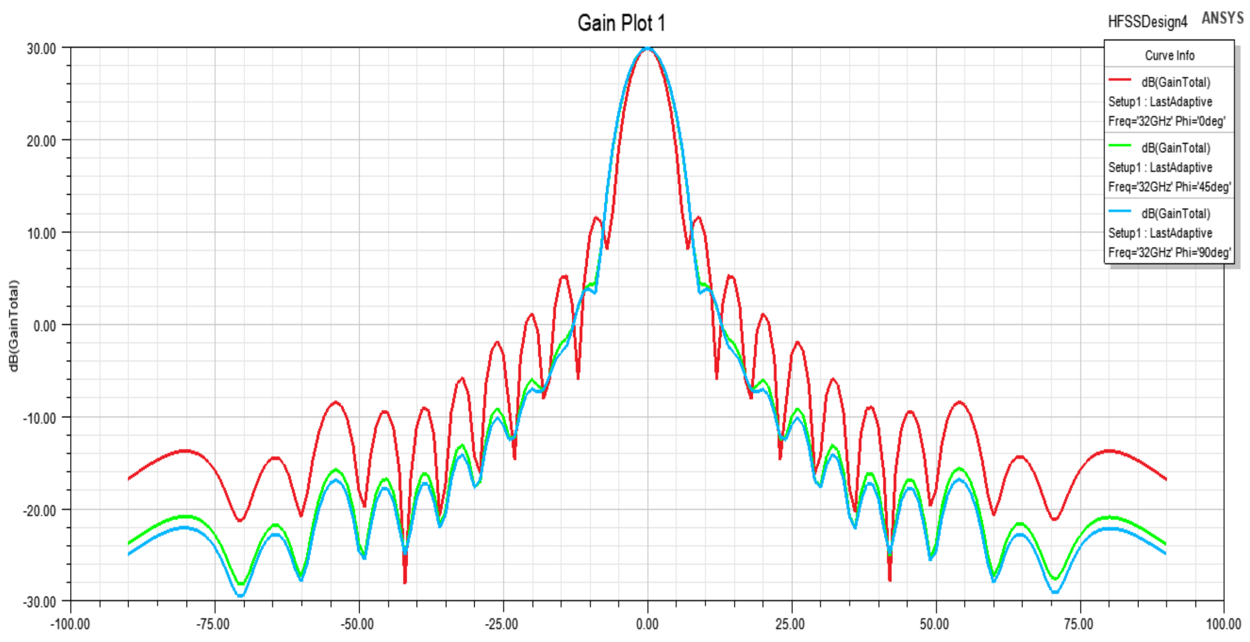


Fig. 6-4. Gain plot of generated antenna. Measurements are for a RHCP antenna.

The objective of this simulation is to determine how the antenna created through this design flow compares to the antenna presented in [1]. Looking at the gain plots of Fig. 6-3 and 6-4, for the RHCP lines, it is clear that the antenna generated in this paper strongly aligns with the performance shown in [1]. Both plots show a max gain at 0 degrees in theta with the gain dropping off in a similar pattern. However, in the reference gain plots, there is some asymmetric behavior outside of the max gain for the Φ = 90 degrees that is not present in the reference antenna. Also, the generated antenna surpasses the max gain of the reference antenna, by about 4 dB. It should be noted that the reference antenna includes a feed circuit that would reduce the gain of the antenna as energy is lost traveling through the feed circuit.

Chapter VII: Conclusion

It has been demonstrated that an antenna created through this design flow will, at least in simulation, perform as well as existing models. Through this it is possible for relative beginners in the field of RF to generate an antenna of the type shown in this paper by alleviating any need for background knowledge of MTS arrays. This does come with limitations with respect to time in generating the needed database tensor data and time to construct a given array. This process also requires the programs of HFSS and MATLAB, which can be expensive and may not be readily available to someone looking to explore this topic.

The ultimate goal of this design flow is to provide a template from which a design flow for all MTS arrays can be made. While this design flow only demonstrates how to create one type of antenna the overall concept can readily be adapted and expanded upon to create other forms of MTS arrays. The Floquet analysis described in section 3 can be reused for other unit cell types, the user would only need to construct the unit cell geometry, which will later be used to generate the full layout. From there, one would need to obtain equations that characterize the reactance tensors of the new MTS and recreate those equations in the MATLAB script. With the new tensor equations and data from the Floquet analysis, the algorithm to find the optimal tensor for each point in the array should work as is. All that remains is for the user to determine what layout is appropriate for their design and code that shape in MATLAB. Finally, the python script is run, this time creating and placing copies of the geometry created in the Floquet analysis.

In addition to increasing the number of antennas this design flow can be applied to, There are many avenues where this research can be expanded upon. First and foremost, the process itself can be expanded to include how to manufacture MTS arrays, how to measure the performance of manufacturers antennas, and how to benchmark their performance. Another area of interest is the application to conformal surfaces. There are now 3D printers with the capability to print along curved surfaces, as opposed to building up from a two dimensional surface. The algorithms used to control this kind of 3D printing can be adopted to create conformal layouts of MTS arrays, opening many new applications for antennas as additively manufactured components.

Works Cited

[1] D. González-Ovejero, N. Chahat, R. Sauleau, G. Chattopadhyay, S. Maci and M. Ettorre, "Additive Manufactured Metal-Only Modulated Metasurface Antennas," in IEEE Transactions on Antennas and Propagation, vol. 66, no. 11, pp. 6106-6114, Nov. 2018, doi: 10.1109/TAP.2018.2869135.

[2] N. Miguélez-Gómez *et al*., "Thickness-Accommodation in X-Band Origami-based Reflectarray Antenna for Small Satellites Applications," *2020 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, 2020, pp. 54-59, doi: 10.1109/WiSEE44079.2020.9262670.

[3] L. Zhou and Z. Shen, "Absorptive Coding Metasurface With Ultrawideband Backscattering Reduction," in *IEEE Antennas and Wireless Propagation Letters*, vol. 19, no. 7, pp. 1201-1205, July 2020, doi: 10.1109/LAWP.2020.2995206.

[4] M. F. El-Sewedy, M. A. Abdalla and H. A. Seregely, "High Directive Fabry-Pérot Cavity Antenna by Using Reflecting Metasurface for 5G Applications," *2020 IEEE International Symposium on Antennas and Propagation and North American Radio Science Meeting*, 2020, pp. 817-818, doi: 10.1109/IEEECONF35879.2020.9330488.

[5] J. Luo, H. Yang, Q. -X. Chu and Y. Zhang, "Gain Enhancement of a Base Station Antenna with Transmitting Metasurfaces," *2021 International Conference on Microwave and Millimeter Wave Technology (ICMMT)*, 2021, pp. 1-3, doi: 10.1109/ICMMT52847.2021.9618143.

[6] V. Dmitriev, C. Nascimento and S. Prosvirnin, "Extended Group-Theoretical Approach to Metamaterials With Application to THz Graphene Fish-Scale Array," in *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 12, pp. 5893-5897, Dec. 2015, doi: 10.1109/TAP.2015.2481490.

[7] Y. Wang, K. Chen, Y. Li and Q. Cao, "Design of Nonresonant Metasurfaces for Broadband RCS Reduction," in *IEEE Antennas and Wireless Propagation Letters*, vol. 20, no. 3, pp. 346-350, March 2021, doi: 10.1109/LAWP.2021.3049882.

[8] J. P. Turpin, D. H. Werner and D. E. Wolfe, "Design Considerations for Spatially Reconfigurable Metamaterials," in *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 8, pp. 3513-3521, Aug. 2015, doi: 10.1109/TAP.2015.2431718.

[9] G. Minatti, F. Caminita, E. Martini, M. Sabbadini and S. Maci, "Synthesis of Modulated-Metasurface Antennas With Amplitude, Phase, and Polarization Control," in *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 9, pp. 3907-3919, Sept. 2016, doi: 10.1109/TAP.2016.2589969.

[10] N. Chahat et al., CubeSat Antenna Design, NASA Jet Propulsion Laboratory/California Institute of Technology, Pasadena, CA, USA: Wiley. 2021.

[11] X. -B. Zhao, X. -Y. Tong, F. Wei, Y. Yang, X. Yang and N. Wu, "Broadband 3D Printed Conformal Dielectric Linear-to-Circular Polarization Metasurface Converter," *2021 International Applied Computational Electromagnetics Society (ACES-China) Symposium*, 2021, pp. 1-2, doi: 10.23919/ACES-China 52398.2021.9581414.

[12] B. Ratni, A. de Lustrac, G. -P. Piau and S. N. Burokur, "3D printed gradient index dielectric metasurface for beam steering applications," *2017 11th European Conference on Antennas and Propagation (EUCAP)*, 2017, pp. 3402-3404, doi: 10.23919/EuCAP.2017.7928200.

[13] E. B. Whiting, L. Kang, S. D. Campbell, D. H. Werner, P. L. Werner and D. B. Burckel, "A MWIR 3D Plasmonic Asymmetric Transmission Metasurface," 2020 IEEE International Symposium on Antennas and Propagation and North American Radio Science Meeting, 2020, pp. 775-776, doi: 10.1109/IEEECONF35879.2020.9329777.

[14] H. J. Bilow, "Guided waves on a planar tensor impedance surface," in *IEEE Transactions on Antennas and Propagation*, vol. 51, no. 10, pp. 2788-2792, Oct. 2003, doi: 10.1109/TAP.2003.817568.

[15] G. Minatti *et al.*, "Modulated Metasurface Antennas for Space: Synthesis, Analysis and Realizations," in *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 4, pp. 1288-1300, April 2015, doi: 10.1109/TAP.2014.2377718.

[16] J. H. Heinbockel, *Introduction to Tensor Calculus and Continuum Mechanics*, Victoria, B.C: Trafford Publishing. 2001.

[17] S. Chalkidis, E. Vassos, A. D. Boursianis, A. Feresidis and S. K. Goudos, "Design of Unit Cells for Intelligent Reflection Surfaces Based on Transparent Materials," *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2021, pp. 1-4, doi: 10.1109/MOCAST52088.2021.9493335.

[18] *Getting Started with HFSS™ Floquet Ports*, Ansys, Inc., Canonsburg, PA, 2015.

[19] M. Mencagli, C. D. Giovampaola and S. Maci, "A Closed-Form Representation of Isofrequency Dispersion Curve and Group Velocity for Surface Waves Supported by Anisotropic and Spatially Dispersive Metasurfaces," in *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 6, pp. 2319-2327, June 2016, doi: 10.1109/TAP.2016.2547021.

[20] The Mathworks,"Quiver or vector plot," mathworks.com. https://www.mathworks.com/help/matlab/ref/quiver.html (accessed Aug. 5, 2022).