1-1-2019

Automation of data collection from an anechoic chamber

Thomas S. Garner

AUTOMATION OF DATA COLLECTION FROM AN ANECHOIC CHAMBER

A Thesis

presented in partial fulfillment of requirements

for the degree of Master of Science

in the Department of Electrical Engineering

The University of Mississippi

Thomas Garner

August 2019

ABSTRACT

This paper describes the calibration methods, and the process that is used on the Ole Miss electrical engineering anechoic chamber. Furthermore, the MATLAB code that operates the anechoic chamber has been updated to collect sweep data. The GUI also now includes a tab for calibration, giving the user the ability to calibrate for their desired frequency range for when there is not sufficient calibration data already available.

DEDICATION

I would like to dedicate this to my mother and father. Their support throughout my life is the reason why I am where I am today.

LIST OF ABBREVIATIONS OR SYMBOLS

AUT – Antenna Under Test

DUT – Device Under Test

UUT – Unit Under Test

EM – Electromagnetic

CP – Circularly Polarized

LP – Linearly Polarized

dBic – gain relative to an isotropic, perfect CP antenna

CR – Carriage Return

LF – Line Feed

ACKNOWLEDGMENTS

I would like to acknowledge my brother Corey Garner. He was an irreplaceable resource for information and help. In addition, I would also like to thank Dr.Hutchcraft, as he provide guidance and knowledge throughout the project.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Acquisition of good data is a goal of all experimentation and testing. We collect data to understand the behavior of existing and new items. We test to see that an existing piece of equipment is still behaving as designed or to understand the change that has occurred from the original condition. New devices, like antennas, are designed and need to be tested. This requires the collection of good data. Good data is defined as accurate, relevant, timely, sufficient and worth its cost. In this paper, the focus is on accurate data. To acquire accurate data, we use instrumentation designed to acquire the desired measurements. Instrumentation accuracy can vary depending on the environment, component wear, material degradation, changes to measurement architecture and other less obvious attributes. When performing experiments or collecting any set of data, we seek to have repeatable results. This requires us to have a standard measurement baseline. By doing proper calibration we can remove extraneous variables and achieve an accurate measurement. The electrical engineering department has a network analyzer, cabling for sending and receiving an RF signal, a holder for a transmitter, a PVC assembly on a computer controlled rotator and the anechoic chamber itself. The chamber is a partial faraday cage with absorber lining the inside. To calibrate we seek to have the same measurements when compared to some standard. When working with systems it is best to calibrate each instrument first. It is outside of the scope for the vast majority of organizations to calibrate a network analyzer, which would require it to be sent off to a specialty company. This leaves the remaining instrumentation to be calibrated as a

system with the network analyzer. This makes proper calibration crucial to instrumentation use. In this paper, a step-by-step procedure is developed to calibrate the assorted instrumentation used with the University of Mississippi Electrical Engineering department's anechoic chamber in far field measurements.

Anechoic chambers are valuable for obtaining antenna radiation patterns and gain, but to obtain the trustworthy results, the chamber must be calibrated. Whether the calibration is done using a vector signal analyzer or using a power meter for more advance calibrations, the following must be considered: return loss in cables and adapters, impedance mismatch error between the RF signal generator and the Unit Under Test (UUT), and the VSWRs for the RF signal generator and the UUT [1]. It is also important to consider whether the anechoic chamber can have the Antenna Under Test, AUT, in the far field region. The antenna size, the operating frequency, and the anechoic chamber size will determine if the AUT's far field pattern can be obtained inside the chamber. This information is important to know before even starting a measurement inside the chamber. The objective will be to calibrate the Ole Miss electrical engineering department's anechoic chamber to allow the testing of antennas and RF devices for future research and commercial testing. With the help of Raytheon SAS, a Narda 642 standard gain horn was measured to collect gain data. This gain data will be useful when testing calibration procedures in the Ole Miss anechoic chamber. Future testing will involve three frequency bands including the L-band, XN-Band, and X-Band, and performing the three antenna method with three different standard gain horns. This will give results at varying frequencies to compare to the same standard gain horn results from the Ole Miss anechoic chamber. These comparison results will allow confirmation in the calibration results.

The American National Standards Institute (ANSI) is an institution created a century ago

on October 19, 1918. Its mission is to define standards and how to assess conformity to those standards. In June of 2017, the American National Standards Institute announced a new revision to the C63.5 antenna calibration standard. Its last revision had been in 2006. This standard is titled "American National Standard for Electromagnetic Compatibility Radiated Emission Measurements in Electromagnetic Interference (EMI) Control Calibration and Qualification of Antennas (9 kHz to 40 GHz)" [2]. This standard includes different methods including three antenna, reference antenna, equivalent capacitance substitution, standard transmit loop, standard antenna, and standard field methods. Since the University anechoic chamber is a controlled environment, this paper will emphasize the use of broadband horns and standard field method in a free space environment.

FAR FIELD REGION

The far field and near field are two regions around an electromagnetic-radiation-source. These two regions vary based on radiation frequency and tolerance error [3]. The far-field distance, r_{ff}, begins at a distance

$$r_{ff} = \frac{2D^2}{\lambda} \quad (1)$$

where D is the largest dimension of the antenna and λ is the wavelength [4, pp. 36]. Equation (1) works well with antennas that operate at low frequencies.

However, at ultra-high frequencies (UHF), other condition must be met. For line sources where $D > 2.5\lambda$, the far field conditions are as followed:

$$r > \frac{2D^2}{\lambda} \quad (2a)$$

$$r > 5D \quad (2b)$$

$$r > 1.6\lambda \quad (2c)$$

where $r \geq r_{ff}$ [4, pp. 43].

A common way to measure far field is the use of an anechoic chamber. Anechoic chambers are designed to keep other frequencies outside and contain absorbing material that reduce reflections off the walls inside the chamber. These chambers must be large enough to keep antennas in the far field range and to keep incidence normal. If the anechoic chamber is

$$R \leq 2.75W \quad (3)$$

where the distance between the source and test antenna is R, and the width and height of the chamber is W, then proper far field measurements can be met [4, pp. 568]. Typically, the antenna under test (AUT) is used as a receiving antenna inside the anechoic chamber during measurements. The source antenna is stationary as the AUT changes angular position. As the AUT is rotating, the pattern can be obtained [4, pp.565].

NEAR FIELD REGION

The region interior to the far field is called the near field. There are two regions of the near field, the closest field to the AUT is the reactive near field and in between the reactive near field and far field is the radiating near field, also known as the Fresnel region. For a line source when $D > 2.5\lambda$, the reactive near field is inside distance $r = 0.62\sqrt{\frac{D^3}{\lambda}}$ from the AUT, and the radiating near field is between $r = 0.62\sqrt{\frac{D^3}{\lambda}}$ and $r = \frac{2D^2}{\lambda}$ [4, pp. 42].

CHAPTER II: FAR FIELD AND NEAR FIELD MEASUREMENT TECHNIQUES

THREE ANTENNA METHOD

There are various methods to use in calibration of the equipment associated with the anechoic chamber and use known gain values to achieve a more accurate measurement. The three antenna method is a common method for determining the gain of an AUT. The three antenna method can find the gain for each antenna measured in the method without knowing any gain values before measurements. However, this gain measurement method requires at least two of the antennas not be circularly polarized. Determining the gain using this method also requires that the antennas are boresight and antennas are match, i.e. the reflection parameters S_{11}, S_{22}, and S_{33} are zero [5]. An illustration of this is shown in Figure 1. The gain can then be determined using the inverse of the Friis equation (4) [5].

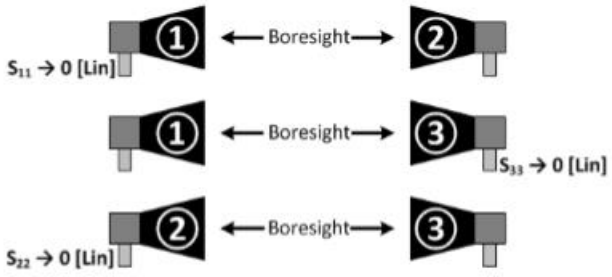

Figure 1: Three Antenna Method (Acquired from [5])

$$[G_1, G_2, G_3,] = \left[\frac{P_1}{P_2}, \frac{P_1}{P_3}, \frac{P_2}{P_3}\right] \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}^{-1} \quad (4)$$

Where P_i (i = 1, 2, 3) is the power received/transmitted at the i^{th} antenna port and G_j (j = 1, 2, 3) is the gain of the j^{th} antenna [5].

PLANAR, CYLINDRICAL, AND SPHERICAL NEAR FIELD SCANNING

According to the IEEE recommended practice for near field measurements, the three common near field measurement methods are planar near field (PNF) scanning, cylindrical near field (CNF) scanning, and spherical near field (SNF) scanning. These scans require a probe with known characteristics [6]. A common probe is the open-ended rectangular waveguide.

The PNF scan is measured using the probe moving in a plane, as the name suggests. The AUT is stationary as the probe moves along the x-axis and y-axis. The PNF requires less absorber around the measurement area and does not require as much mathematical analysis compared to the CNF and SNF scans. The PNF scan is best for measuring highly directive antennas [6]. If the EM waves are not captured in the field of measurement, then the AUT's field far patterns cannot be accurately measured. Raytheon will use the PNF scan method on the three standard gain horns mentioned earlier to allow us to make a comparision in our chamber.

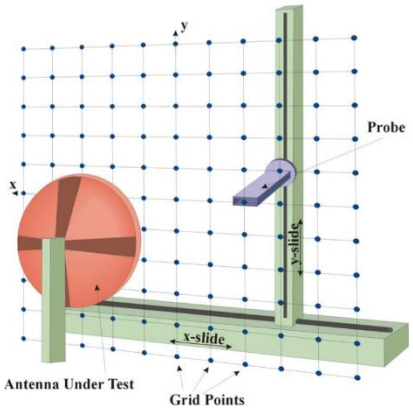

Figure 2: Planar Near Field Scan (acquired from [6])

The second type of scan is the CNF scan. As the name suggests, the CNF method measures around the AUT in a cylindrical pattern. The probe is able to move in the z-axis, while the AUT is on a rotator. A type of antenna that is suitable for the CNF method would be a fan-beam antenna [6].

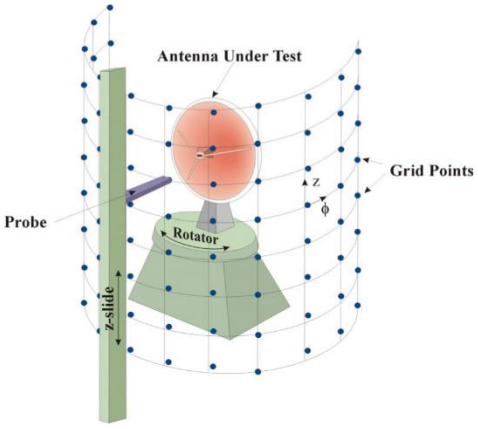

Figure 3: Cylindrical Near Field Scan (acquired from [6])

The third type of scan is the SNF scan. The SNF scan is more mathematically intensive, and it requires more than one rotator to achieve the measurement. A common configuration is to have the probe fixed, and the AUT is able to rotate in conjugation with a rotating table to map out a sphere. The SNF scan is suitable for antennas, such as an isotropic antenna, where the PNF and CNF scans are not reasonable to use [6].

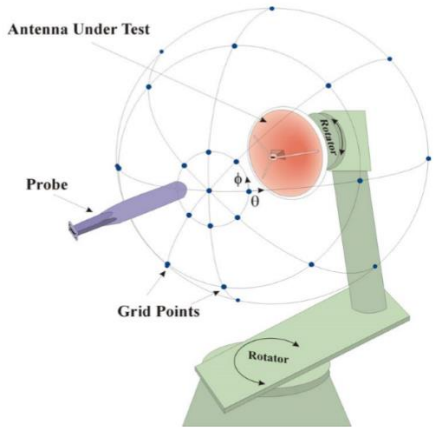

Figure 4: Spherical Near Field Scan (acquired from [6])

GAIN MEASUREMENT OF LINEARLY AND CIRCULARLY POLARIZED ANTENNAS

Stutzman and Thiele describe linearly polarized antennas as vertically polarized or horizontally polarized, if the electric field moves back and forth along a line. They also state a left-hand or right-hand circularly polarized antenna if the electric field vector is constant in length and rotates around in a circular path [4, pp. 62].

A linearly polarized (LP) antenna's gain can be measured using the gain comparison method. This technique requires the use of an antenna of a known gain. Typically, the antenna used is a standard gain horn. To perform this measurement method, a source antenna is supplied a fixed input power, P_t. The source antenna's pattern peak is boresight with the test antenna to measure the receive power, P_T. The same measurement is done with the SGH to measure its received power, P_S. After these measurements are completed, using equation (5) the gain of the linearly polarized test antenna is calculated. Using equation (6), the gain can be represented in dB [4, pp. 572].

$$G_T = \frac{P_T}{P_S} G_S \tag{5}$$

$$G_T(dB) = P_T(dBm) - P_S(dBm) + G_S(dB) \tag{6}$$

Where G_T is the gain of the AUT, G_S is the gain of the SGH, P_T is the power received by the AUT, and P_S is the power received by the SGH.

A circularly polarized (CP) antenna can be measured using the same method described above; however, this requires a good-quality circularly polarized source of known gain [4, pp. 573]. As before a gain standard is used, but for a CP antenna, this gain standard needs a low cross-polarization. This is more difficult to create than for a LP antenna, so normally two orthogonal LP

antennas, or one LP antenna in two orthogonal positions, is used to for the gain comparison method. Thus, we conclude a slightly different measurement called the partial gain method. This method includes the gain measured for vertically polarized and horizontally polarized, which are G_{Tv} and G_{Th} respectively [4, pp. 573]. The total gain equation for the partial gain method is calculated using equation (7).

$$G_T(dB) = 10\,log(G_{Tv} + G_{Th}) \quad [dBic] \tag{7}$$

COMPARING ANTENNA PATTERNS

When measuring for the antenna pattern of an AUT, it is important to measure that antenna multiple times and compare each result. When comparing the results, methods are used to interpolate accurate and reliable results. For each measurement, it is important that the same coordinate system is used and every measurement the AUT is boresight. Any discrepancies in the alignment of the AUT will cause errors in the patterns measured [7]. After obtaining the results, separate the main beam region and the side lobe region. Both regions are analyzed differently. In the main beam region, the comparisons that are important to observe are the following: beam pointing, peak gain, directivity, and beam width. For the side lobe region, first normalize the patterns to the peak of the main beam. The purpose for normalizing the side lobe region is to remove the main beam region from the comparisons of the side lobe regions. Next, the ratio of an Equivalent Stray Signal (ESS) to the peak of a main beam is calculated by comparing the normalized patterns' differences. Using the ESS, an estimate of uncertainty for a measurement error source [7]. This will be quite useful comparing the SGHs measurements in one of Raytheon's chambers and the measurements in Ole Miss's chamber.

To measure the ESS of two antenna pattern measurements, the first step is to measure the distance, Δ, between both patterns at each point. The equation for plotting the ESS relative to the peak of the main beam is shown in equation (8) [7].

$$ESS/SIG = 20 * log(10^{\frac{\Delta_{dB}}{20}} - 1) \quad (8)$$

The ESS is greatly affected by even the smallest amount of misalignment. One method to correct this error is by manually shifting the patterns until both main beams peaks are aligned. This will result in the minimal value for the calculated ESS. For calculating ESS for the side lobes, removing the main beam from the calculations will give more accurate results [7].

The ESS is useful information because it can be used to determine the difference between the patterns. With this information, the estimation due to signal error source can be calculated. It is important to note that the two measured patterns are not the “true” value. This means that the ESS is an approximation of the uncertainty at a specific angle [7]. Since the ESS will vary for every angle, calculating the Root Mean Square (RMS) of the ESS curve will improve reliability for the estimated uncertainty in less reliable regions [7]. It is important to calculate the RMS level for the main lobe and side lobes separately as the main beam lobe can greatly affect the results of the RMS level of the side lobes. This can easily obtained in MATLAB using the rms() function.

CHAPTER III

OLE MISS ANECHOIC CHAMBER: THE WORKING ENVIROMENT

The purpose of the anechoic chamber is to isolate the AUT from any external electro-magnetic interference. The anechoic chamber itself is a metal structure forming a Faraday cage. The inside is lined with absorber material to prevent any reflection interior to the chamber. The AUT is positioned in front of the door and the transmitting antenna is at the back side of the chamber as seen in Figure 5. There are right angle joint connectors and rotating SMA connectors as shown in Figures 6 and 7 to prevent binding of the coax cables. The rotating SMA connector is wedged into a square piece of wood to prevent the coax cable from rotating and allowing the rotating part of the SMA connector to perform its function. The AUT can rotate 360 degrees, and the transmitting antenna can be positioned vertical or horizontal to measure the E-field and H-field. RF cables run through the floor and the wall and then back to the instrumentation.

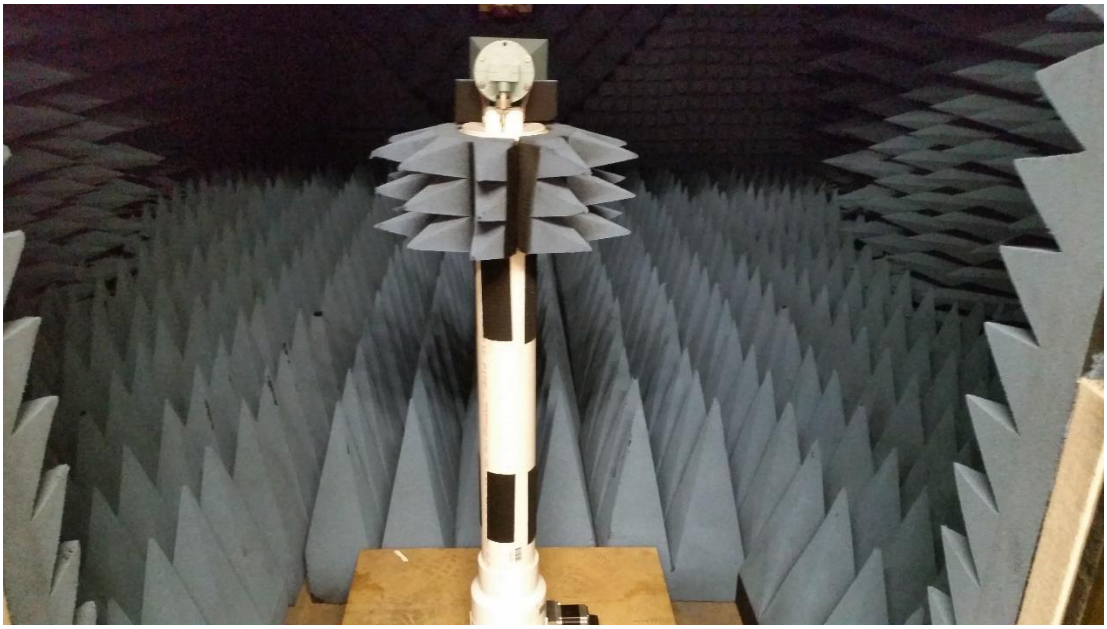

Figure 5: Anechoic Chamber

Figure 6: Right Angle Connector

Figure 7: Rotating SMA Connector

VECTOR NETWORK ANALYZER AND THE PROGRAMMING CODE

The software architecture is a standard object oriented development project. It consists of a rotator class (rotator), measurement equipment class (8530A), input/output GPIB interface class (gpib), and a graphical user interface class (chamberGUI1) to handle the operation logic and human interface. The rotator class manages the rotators by managing calibration, reset, speed and direction of rotation, angle increment, start angle and ending angle. The measurement equipment class is for the vector network analyzer to load antenna definitions, generate calibration sets, measure at a single frequency, measure a sweep of a frequency range, collect and save generated data. The GUI is the interface for the operator to enter in frequencies to measure, the angles to measure, the number of increments, etc.

The previous work used a GUI created in Matlab. This GUI allowed for setting initial parameters to collect data from the HP 8350A network analyzer as shown in Figure 8. The receiver has several menu buttons for various needs. An important part of these buttons includes the softkeys. These softkeys are located on the right side of the monitor of the network analyzer. They are used by the user to navigate through menus.

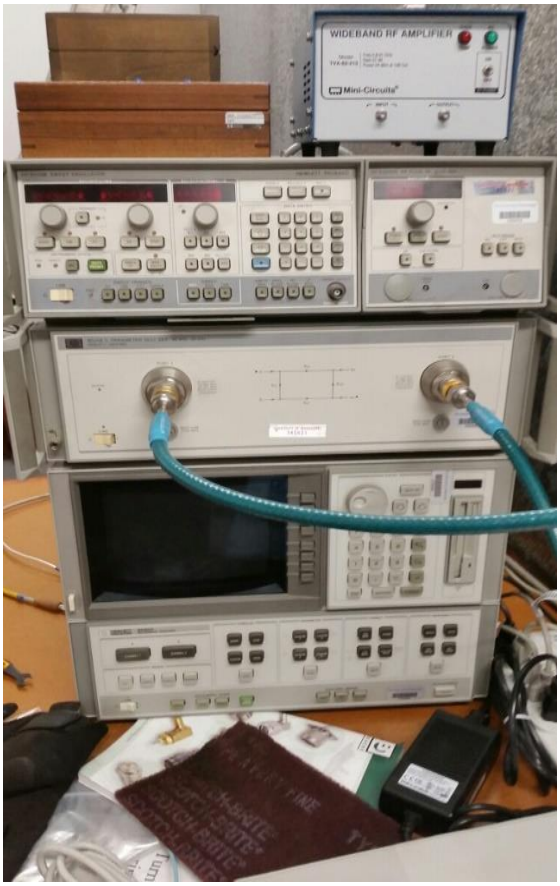

Figure 8: HP 8530A Network Analyzer

The key point of this architecture is the modularity of the software. If a new rotator is used, the only thing required to be rewritten is the rotator class. One would keep all of the previous functions and adapt them with the new rotator commands. If one replaced the vector network analyzer, one would rewrite the commands required to achieve the same function calls. The GUI would require only minimal modification if any at all. This architecture will allow the software to evolve over a longer period of time as compared to the previous non object oriented approach in BASIC.

At initial stages of this project, the GUI and accompanying software had not been successful in collecting data. During diagnostics tests, an error in data being transferred from the network analyzer to the computer was occurring. It was discovered that not all requested commands were being performed. In order to diagnose the problem, it was necessary to learn the operations for doing a manual, button pushing, data collection process. This clearly was not an efficient way to collect data, but it did provide for a more complete understanding of the network analyzer. To collect data, many parameters can be set, but then it must be triggered to collect the data and store it in the network analyzer's memory. During this learning process, it was found that once the command to go to the Stimulus Menu, the soft 5 and soft 6 key are activated, the network analyzer is no longer able to receive commands. This was because connection from the computer, the HPIB port, was closed even though no close port command was sent. However, even when these commands were removed, there was still no data being stored for retrieval. The next step involved internally triggering the network analyzer to create data that could be pulled from the computer. After the trigger on the network analyzer was performed, a port to the network analyzer using GPIB connections was opened from the Matlab command window. Using the commands *fprintf(netA.port, 'FORM4; OUTPDATA;'); data = fscanf(netA.port);* data was stored as points separated by commas. When performing this test, it was noticed that the "addrs" light was on when the network analyzer was being triggered. The current Matlab code did not give this light, hinting at the fact the network analyzer was not being triggered by the Matlab code. Aikmin, a previous graduate student, had code that was written using Basic that went ahead and selected the HPIB trigger. The current code followed after this but gave us the problem as described. The reason for this is Basic handles inputs and outputs differently than Matlab so a slightly different method had to be performed. The network analyzer can be triggered internally or by HPIB. When just

selecting the HPIB trigger soft key, the network analyzer would not trigger. However, it was discovered that performing an internal trigger then and HPIB trigger, the network analyzer could be triggered via HPIB. The commands previously were used and these worked. This is a distinctly different behavior than described in the HP manual. Future programmers of the system need to understand this behavior when using the built in fscanf function. The fscanf function is equivalent to the HP BASIC input command. The data was then successfully stored in both a file and a Matlab variable. Below is the modified working function.

```
function data = getData(netA)
            disp('in getData')
            fprintf(netA.port, 'FORM4;'); %ASCII separated by commas
            %format
            fprintf(netA.port, 'OUTPDATA;');%transfer data to computer
            disp('in getData and sent setup commands')
            pause(1)
            data = fscanf(netA.port);%store NA data into variable data
            disp('in getData read data')
```

Notice the object netA in the above code. See Appendix 2 for the HP8530A object code. This is the instantiation of the network analyzer object, which is controlled by the GUI. This software controls access to the network analyzer. Properties can be set by the GUI and these include:

- port- this is the communication port used by the HPIB interface
- startFreq- the starting frequency of a sweep
- stopFreq- the stopping frequency of a sweep
- centerFreq- the center point for the frequency sweep
- spanFreq- the width of the frequency sweep
- incrementFreq- the distance between each frequency measurement
- markFreq- the frequency marker for the network analyzer
- unit- default frequency scale

- format- network analyzer plotter setting
- points- the number of frequency divisions data points in the frequency sweep
- measurementType- measures magnitude or phase
- dataA- the data retrieved from the network analyzer
- realData- converts the string format data into a complex number

Each of the properties follows the standard programming methodology where each property is set by a function call. Other functions include:

- openPort(netA)- opens communication to the network analyzer
- closePort(netA)- closes communication the network analyzer
- preset(netA)- perform factory preset on network analyzer
- x = measureFreq(netA)- measure the frequency of the marker
- y1 = measureMag(netA)- measure the magnitude of the marker
- y2 = measurePhase(netA)- measure the phase of the marker
- incMarkFreq(netA)- move marker to next data point to be collected
- triggerON(netA)- allows the data point to be captured and stored into memory
- triggerOFF(netA)- stops further capture of data
- incrementCalc(netA)- determines the number of points that will be collected during a frequency sweep
- A = trace(netA)- collects the trace from the network analyzer
- Q = measure(netA)- performs the measurement
- angle_preset(netA)- sends the commands to prepare the network analyzer for data output
- logPlot(netA)- shows a polar log plot on the network analyzer screen
- trigger_netA(netA)- performs the data collection

- data = getData(netA)- retrieve the data stored in the network analyzer's memory
- data2 = formatData(netA)- format the string data into complex numbers

This is a general description of the application programming interface (API) for the HP8530A network analyzer.

This data is being stored as a point, where the x value is the real number and the y value is the imaginary number. The data is also characters and not numbers, so the data was converted to a double and each point was stored as a complex number.

A Narda 612A 5.3-8.2 GHz gain horn was used as the antenna under test (AUT) for these simulations shown in Figure 9. Figure 10, 11, and 12 show the measured data from the network analyzers The network analyzer was set to transmit at 7 GHz and the rotator was set to increment one degree, three-hundred and sixty-one times. The code was written that way to return the rotator back to zero. Currently the actual patterns or the actual gains of the AUT is not known. This was a test to see if we could pull data from the network analyzer and use said data.

Figure 9: NARDA Standard Gain Horn

The Cartesian magnitude plot shown in Figure 10 gives the pattern for the AUT but does not show the actual gain of the antenna. These values are normalized showing the main lobe to have a max at 0dB.

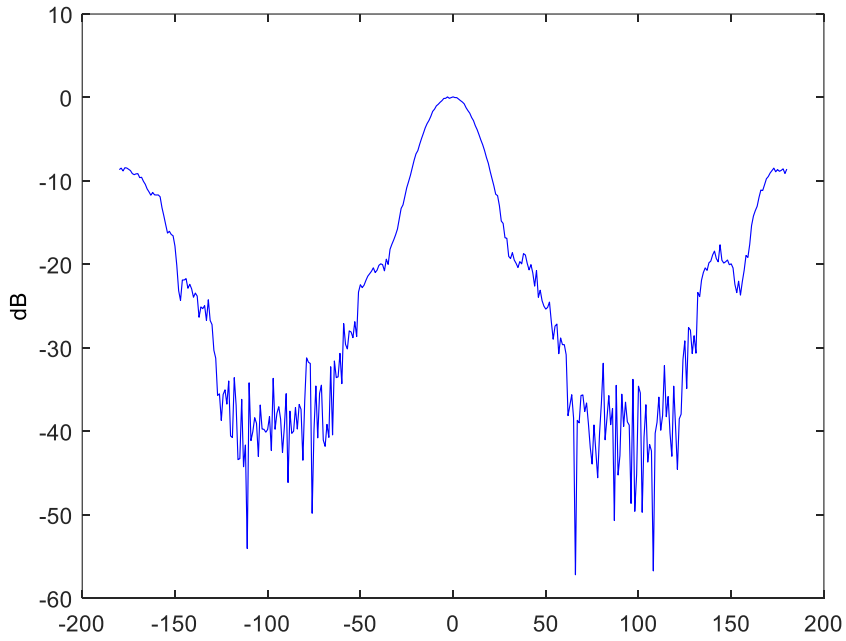

Figure 10: Normalized Cartesian Magnitude Plot (Before Calibration)

The phase plot shown in Figure 11 should be symmetric but without proper leveling of the AUT there are errors in this measurement. The dip and spike in the center of the graph is a problem that has been seen in graphing incorrect points in the past. It is possible that this spike is a result of a drift error from a second measurement at the same angle. However, this shows that we can pull data and use that data to make graphs. Also, note the x-axis is actually showing -180 to 180 degrees not 0 to 360 degrees.

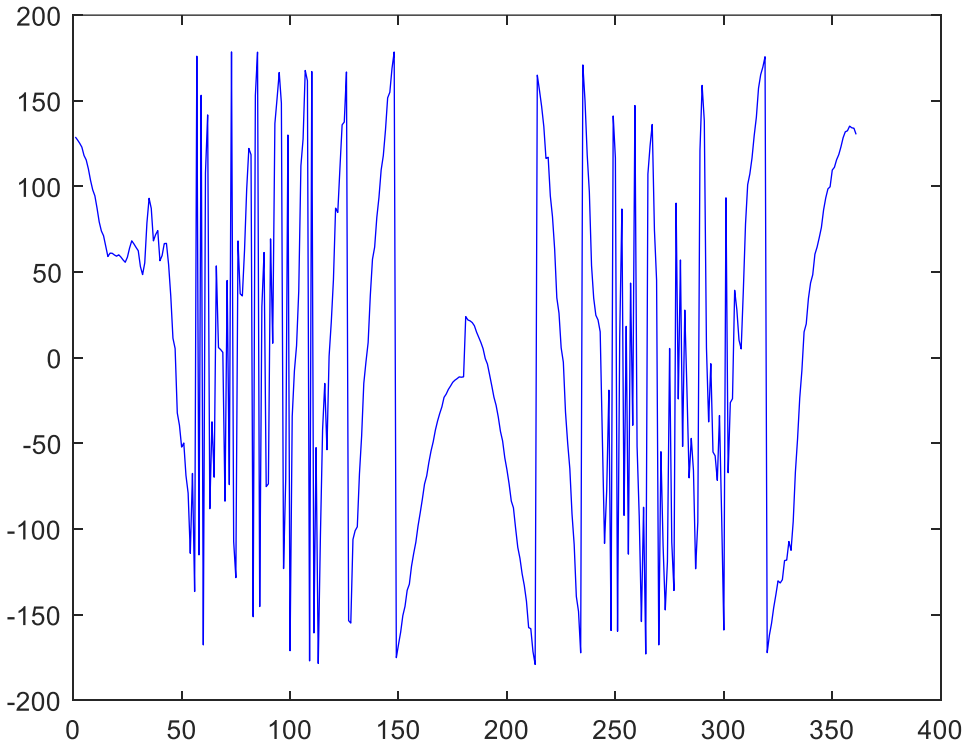

Figure 11: Phase Plot (Before Calibration)

The polar magnitude plot can be seen in Figure 12. This plot shows a normalized version of what is currently visible on the network analyzer during tests. The plot is another method which can be used to easily display the main lobe of an antenna pattern along with any side lobes.

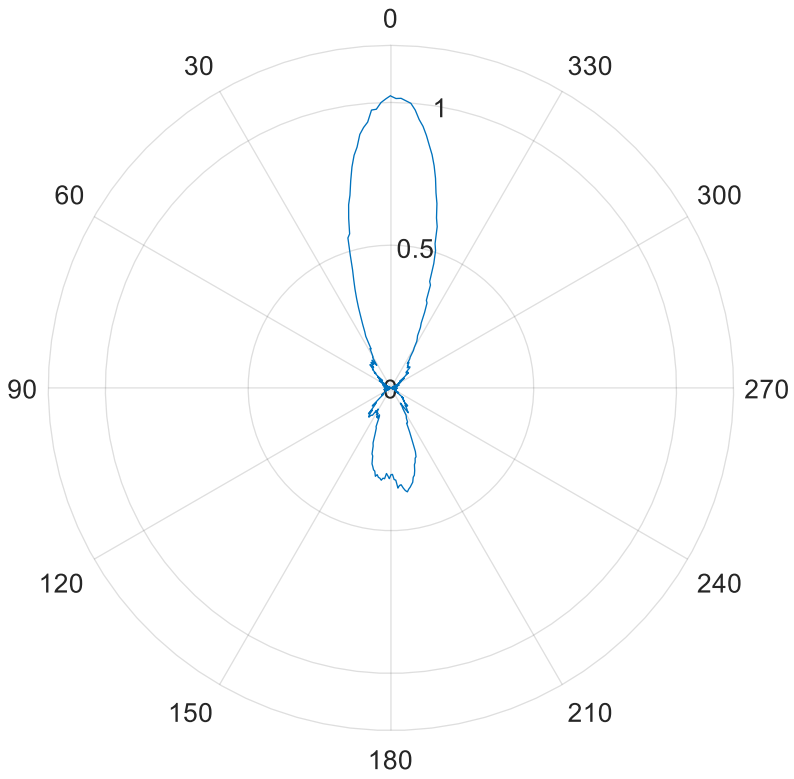

Figure 12: Normalized Polar Magnitude Plot (No Calibration)

CHAPTER IV

NETWORK ANALYZER SWEEP DATA COLLECTION

The anechoic chamber now has the capability to preform a frequency sweep. The sweep capabilities will allow collection of antenna data for various frequencies at every desired angle. The code is designed to sweep across a frequency range using a user-selected number of points at a single angle before continuing onto the next angle. The available sweep increments that are listed on the HP8530 via soft keys are 51, 101, 201, 401, and 801. The GUI increment dropdown box has been updated to reflect these values.

Previously the function angle_preset(netA) allowed a far field measurement for a single frequency. In addition, this frequency had to be changed within the code. The GUI start and stop frequencies inputs did not change the measured frequencies. This has not been changed for the function angle_preset(netA). However, this function should no longer be needed, and the function call in 8530A.m is commented out. The code below is the function to sweep a frequency range, and the number of points between the start and stop frequencies.

```
function sweep_preset(netA)
        fprintf(netA.port,'menudoma; soft1;');
        netA.setStartFreq(netA);
        netA.setStopFreq(netA);
        netA.connect_S21(netA);
        fprintf(netA.port,'menustim; soft8; soft5; soft7; logp;');
```

```
        disp(netA.points)
         netA.setNumPoints(netA);
    end
% Sets the number of points of netA in frequency sweep
    function setNumPoints(netA)
        switch netA.points
        case 51  % User selects 1
                fprintf(netA.port,'menustim; soft3; soft1');
        case 101  % User selects 2
                fprintf(netA.port,'menustim; soft3; soft2');
        case 201  % User selects 5
                fprintf(netA.port,'menustim; soft3; soft3');
        case '401'  % User selects 10
                fprintf(netA.port,'menustim; soft3; soft4');
        case '801'  % User selects 20
                fprintf(netA.port,'menustim; soft3; soft5');
        end
    end
```

In chamberGUI1.m the line of code that only allows for a single frequency far field measurement is:

```
set(handles.pb_start, 'Callback', {@Start_Callback,handles}); %use for single frequency
```

This desired frequency must be changed within the HP8530A.m file under the function angle_preset(netA). In most cases though, it would be easier to just use the sweep data collection and perform the post processing after the sweep is finished. Selecting the same start and stop

frequency values should give only one row of data. If a frequency sweep is desired, the line of code that should be uncommented is:

```
set(handles.pb_start, 'Callback', {@Sweep_Callback,handles}); %use for a sweep frequency
```

This data set could not be saved in the same way as the single frequency measurement. This required concatenation of every angle in the data set. This was achieved with the following code:

```
% Creates an array of the frequency points collected. This will be
% written to the final array 'Z' which contains all relevant data.
C = zeros(1,handles.netA.points);    % initialize array
for m = 1:handles.netA.points
        C(m) = handles.netA.startFreq + (handles.netA.incrementFreq*(m-1)); %contains
        %frequencies
end
    C = C';
    save('C:\temp\sweep_data\C.mat','C'); %contains frequencies measured
Z=double.empty;
for n = 0:handles.rot_aut.degree*direction:finalPos
                if stopflag == 1
                        break
                end
        % Wait for rotator to not be in motion
            handles.rot_aut.settle(handles.rot_aut);
        % ------new data collection ------------
            if n ~= finalPos
                % Trigger the network analyzer to take a data point
                handles.netA.trigger_netA(handles.netA);
                % Increment the aut rotator position
```

```
    handles.rot_aut.increment(handles.rot_aut);


    data = handles.netA.getData(handles.netA);%puts measured data in array
    handles.netA.dataA = data;
    data2 = handles.netA.formatData(handles.netA); %data is now complex numbers
    save('C:\temp\sweep_data\data2.mat','data2');
    handles.netA.realData = data2;
    % Fill the array with sweep_data
    if isempty(Z)
       Z = horzcat(C,data2); %concatinate frequencies into first column of Z
    else
       Z = horzcat(Z,data2); %concatinate measured data starting at column 2 because
       %frequencies are in first column
    end
    data2 = double.empty; %empty data2 before colleting next angle measurement
    end
end
```

This does not evaluate the code and provide pattern results as angle_preset(netA) did upon finishing data collection. A new file was created in MATLAB called Post_Processing.m to evaluate the desired frequency and angle inside the set of data retrieved from the network analyzer. The following code is for post processing:

```
%This code is for post processing sweep data collected by the HP 8530A
%To use this code, look at the row you want to extract and change the row
%variable to the row number. NOTE I DO NOT GRAB THE FREQUENCY WHEN
%EVALUATING THE DATA. IT IS UP TO YOU TO KNOW WHICH FREQUENCY YOU
GRABBED
```

```
clear;
clc;
row = 1; %Change Row to the row you want to evaluate
angle_inc = 1; %Change angle_inc for the increment you used in the GUI for the rotator

%Load sweep_data
filename = 'C:\Users\Thomas\Documents\MATLAB\sweep_data.mat';
data = importdata(filename);
data2 = data(row,[2:end]); %contains data for the row you selecting. does not have frequency
freq = data(row,1);
dataDB = 20*log10(abs(data2));          %convert data2 to decibels
dataDBshifted = circshift(dataDB, 180);   %shift data so main lobe is center
%save('C:\temp\Test_Sweep\dataDBshifted\test_sweep_dataDBshifted.mat','dataDBshifted');
figure;
plot(-180:angle_inc:180, dataDBshifted);
ylabel('dB');
xlabel('angle');
title(['Measured Frequency is ',num2str(freq),' MHz']);
dataPhs = rad2deg(angle(data2));
dataPhsshifted = circshift(dataPhs, 180);
%save('C:\temp\Test_Sweep\dataPhsshifted\test_sweep_dataPhsshifted.mat','dataPhsshifted');
figure;
plot(dataPhsshifted);
```

The first step for the user is to open sweep_data.mat from the saved location they selected in chamberGUI1.m. This should create a variable Z in the variable window in MATLAB. Open this variable to observe the frequencies and their respective complex data that was collected for

each angle. To obtain the far field magnitude and phase pattern the user should find the row that contains the frequency he or she wants. Locate the variable row in Post_Processing.m and change the number to the respective row. Next, is to identify what angle increment was used during measurement. This angle increment was set in the GUI before the far field measurement started. Change the value of angle_inc to the increment value that was used for measurement. If it is desired to save dataDBshifted or dataPhsshifted, uncomment the save() functions and change the save location. When this program runs, it should display two figures. One is the far field pattern in dB and the other is the phase. The following two figures, Figure 13 and Figure 14, are examples obtained using the NARDA 640 standard gain horn for a frequency sweep between 8.2-10 GHz with 51 points. To confirm the frequency row that was selected, the frequency is retrieved from the row of data in sweep_data, and it is put in the title of the far field pattern.

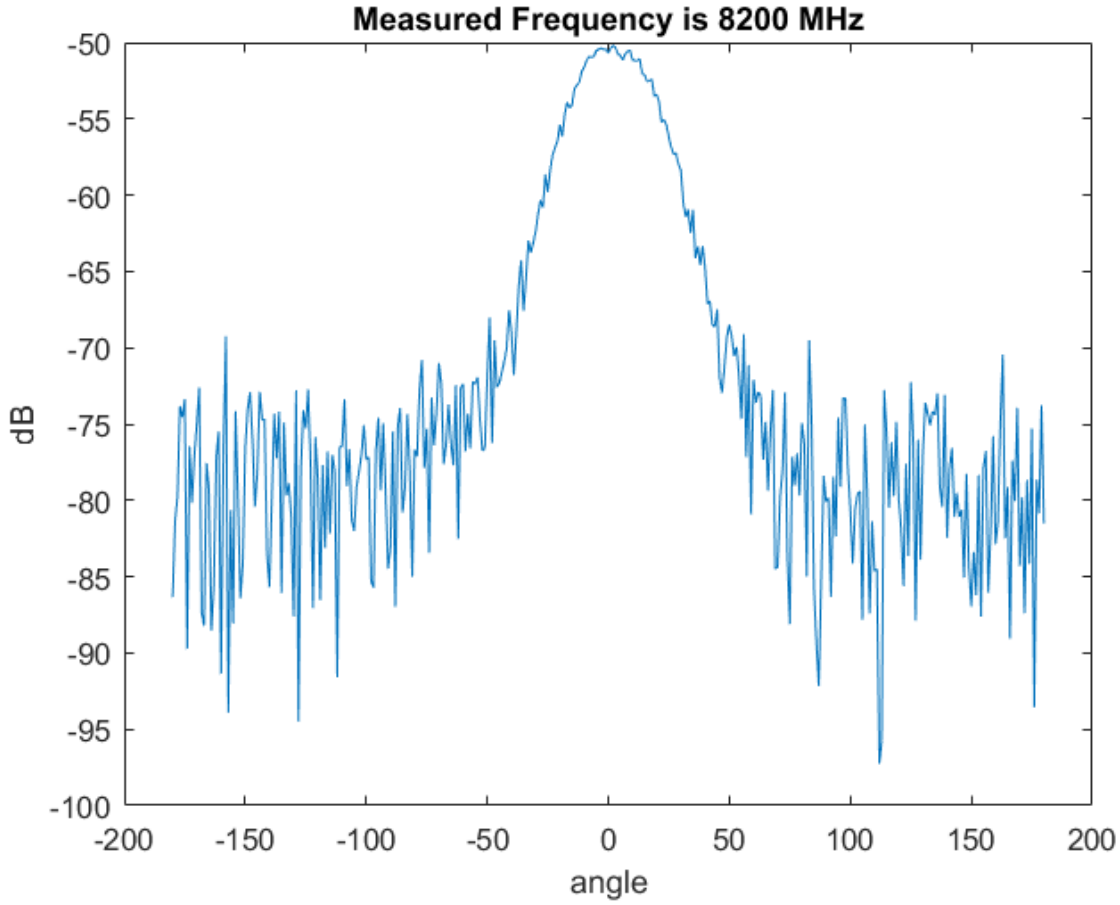

Figure 13: NARDA 640 dB Sweep Data Collection Example

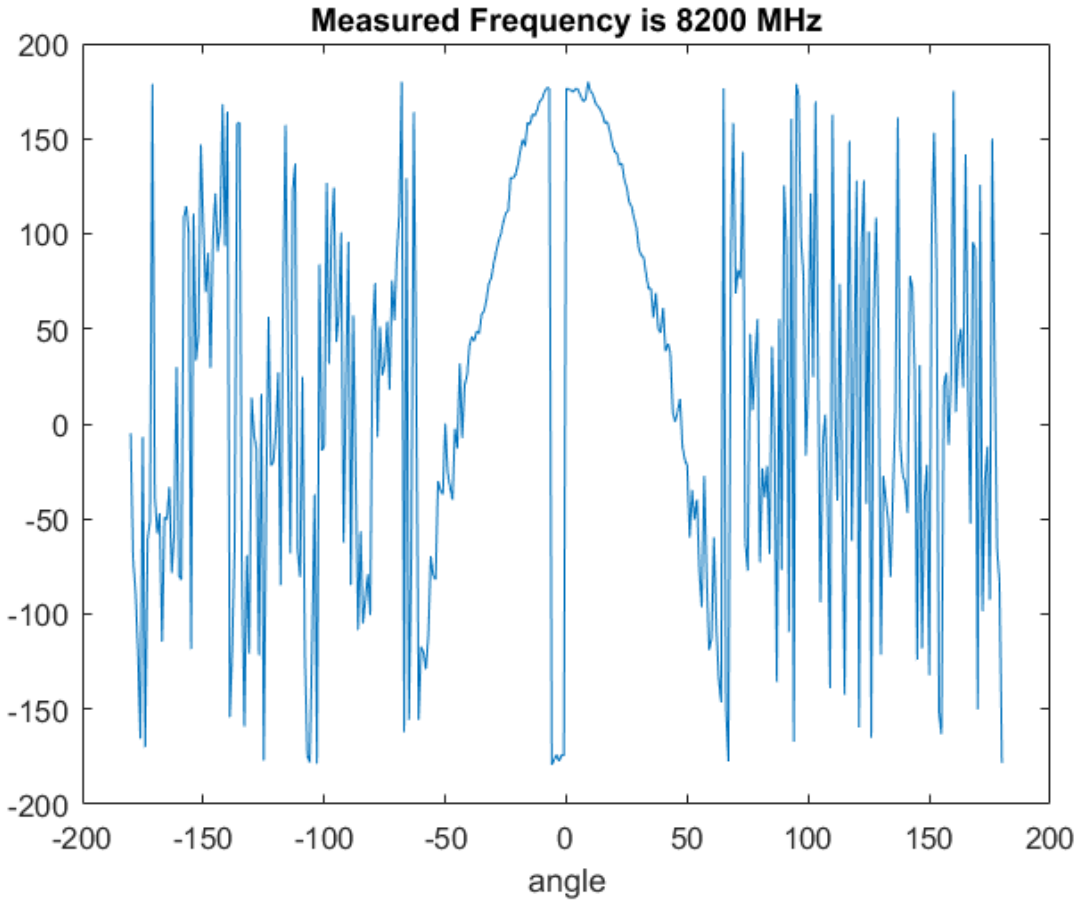

Figure 14: NARDA 640 Phase Data Collection Example

If the user decides to use the Sweep_Callback function for a single frequency, the option to average for a smoother graph is available. In Post_Processing.m, the option to average this data is available by uncommenting a specified section of code. Figure 15 shows a graph of averaged data. Note that this data was not collected at the same time as Figure 13 & 14.

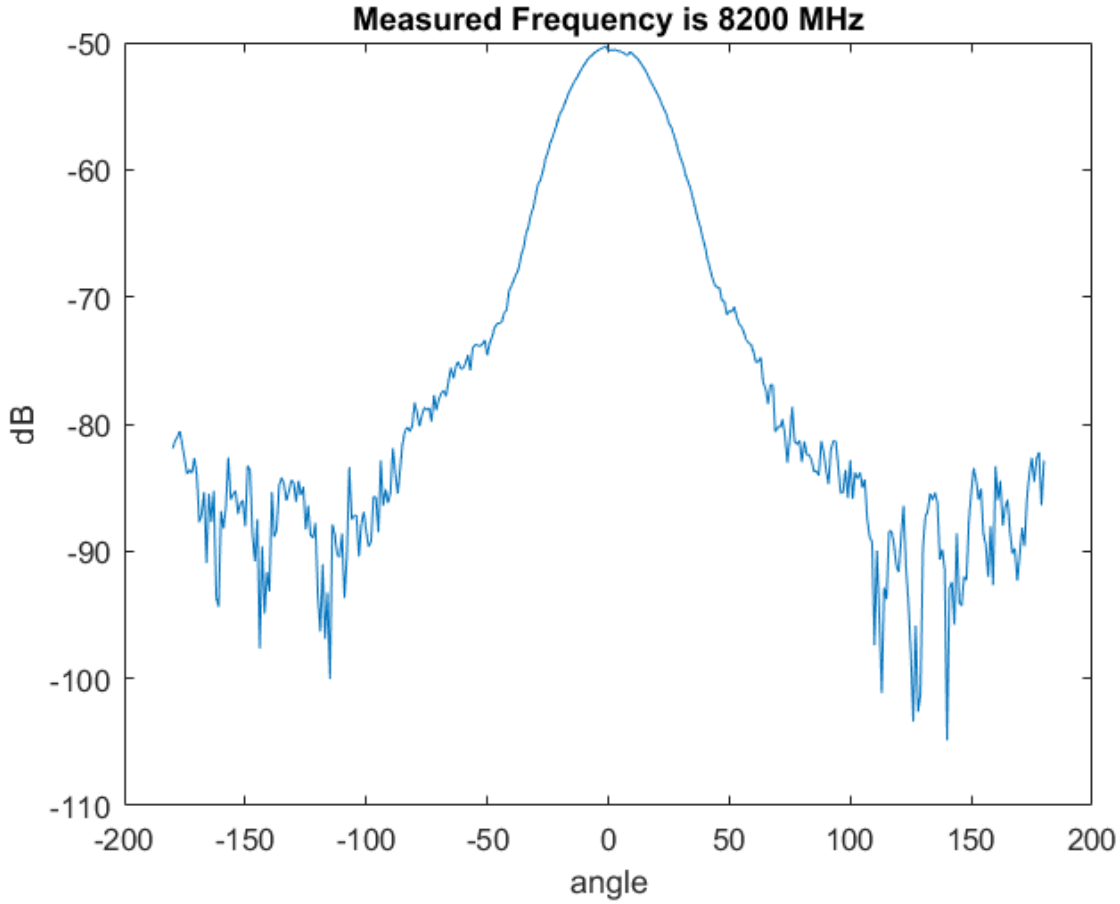

Figure 15: Averaged NARDA 640 dB Sweep Data Collection Example

CHAPTER V

CALIBRATION TECHNIQUES

The objective of calibration is to reduce repeatable systematic errors in the entire system, including the chamber itself. In this case, standard gain horns are measured, and it is compared to the known characteristics of that standard gain horn. The network analyzer will calculate the exact amount of inaccuracy and the adjustments needed to compensate for the errors. This data is called "error coefficients" and is stored on the diskette. This data is then used to give more accurate measurement data during actual measurements [8]. It is important that if any changes are made in the system, collection of calibration data must be performed again. For example, if attenuators, power amplifier, or RF cables are added, replaced or removed for a data set, a new set of error coefficients is required. Another reason to collect a new set of calibration data is wanting to perform an actual measurement with a higher number of points than the calibration data on the diskette.

There are three types of calibration methods that the HP 8530A can perform. The three are: Antenna Calibration, RCS Calibration, and Network Analyzer Calibration [8]. In this thesis project, antenna calibration was chosen and a method was designed in MATLAB to allow a user to perform calibrations. The antenna calibration uses a standard gain horn with defined gain values, in dBi, at specific frequencies to calibrate the system. Using this method with standard gain horns allows corrections for transmission response errors [8].

Two domain calibrations that can be performed are angle domain and frequency domain

calibrations. If a calibration is performed in the angle domain, the calibration is at a single frequency. Whereas, a frequency domain calibration will allow the user to calibrate over a range of frequencies. HP recommends using the frequency domain method. This gives the user access to switch to the angle domain and pick any of the frequencies from the frequency domain calibration [8].

When creating an antenna definition on a diskette, the frequencies must be evenly spaced. The SEG line of the file must contain the start and stop of the frequency along with the number of points. An example of this can be observed in Figure 16. It is important to note that your measured frequency does not have to match the exact frequencies in the antenna definitions file. The receiver, in this case the HP 8530A, will interpolate between frequency points in the antenna definition [8].

The diskette that is inserted into the HP 8530A must be an ASCII file that follows the CITIfile format. It is important that the file name is represents AC_XXXXX.XXX, as this is what the HP 8530A will read. The X's may be characters or letters, and it is recommended the extension is a .txt. Figure 16 shows the format and has variable portions of the file pointed out [8]. This example shows an antenna definition that contains a single antenna definition. It is possible to have multiple antenna definitions in a single file and will be discussed later in this section.

Now, an in-depth description of the file will be discussed. The top four lines are the header for the entire file. It is important that these four lines only occur once, at the top of the file [8]. Let us observe the first three lines:

CITIFILE A.01.01

#NA VERSION HP8530A.01.00

#NAME ANTENNA_DEF

These three lines do not need to be edited. Even if a different firmware update is being used, there

is no need to change line two [8]. It is advised to leave these three lines as is. The next line is:

#NA DEF_LABEL uWaveA.1

This is the line represents the title that will appear for the antenna calibration softkey. “uWaveA.1” can be changed to the user’s desired name. However, this title is limited to 10 characters [8]. This concludes the header lines of the file. Continuing onto the fifth line:

#NA STANDARD 1

This line defines the antenna definition, ranging from one to seven. When creating multiple antenna definitions in a single file, the user should begin with one and increment by one for every additional definition added [8]. Next is labeling the softkey:

#NA STANDARD_LABEL SASGH-1.10

The user should change “SASGH-1.10” that reflects the type and frequency range of the standard gain horn. This name should not exceed ten characters [8]. To define the number of points for the antenna definition, change the number in:

#VAR FREQ MAG 21

Currently, the code is selecting the number of points via softkeys on the network analyzer for measurements. The lowest value that is displayed under number of points via softkeys is 51. This means, the user should never select a value less than 51 unless the code is updated to allow smaller number of points during measurements.

#DATA GAIN [1] DB

This line does not affect the antenna definition. However, it is recommended by the manual to leave this line in [8]. The next three lines represent the start and stop frequencies followed by the number of points.

#SEG LIST BEGIN

#SEG 2000000000 4000000000 21

#SEG LIST END

This example will set 2 GHz as the start frequency and 4 GHz as the stop frequency with the number of points being 21. If the user sets the number of points to 51, then there should be 51 gain values listed between BEGIN and END. The gain values must be entered on separate lines. These values should ascend in order corresponding to their respective frequency [8]. It is critical that the user advances to a new line on the END statement. If the new line is not added, the network analyzer will display the error CAUTION: DATA OVERFLOW. To add the new line, simply press the return key on the keyboard while the cursor is at the end of END. In this standard ASCII ANSI format file created in notepad on a Windows 10 operating system, new line is a carriage return (CR) followed by a line feed(LF). Also, HP8530A.01.00 may be changed to HP8530A.01.64.

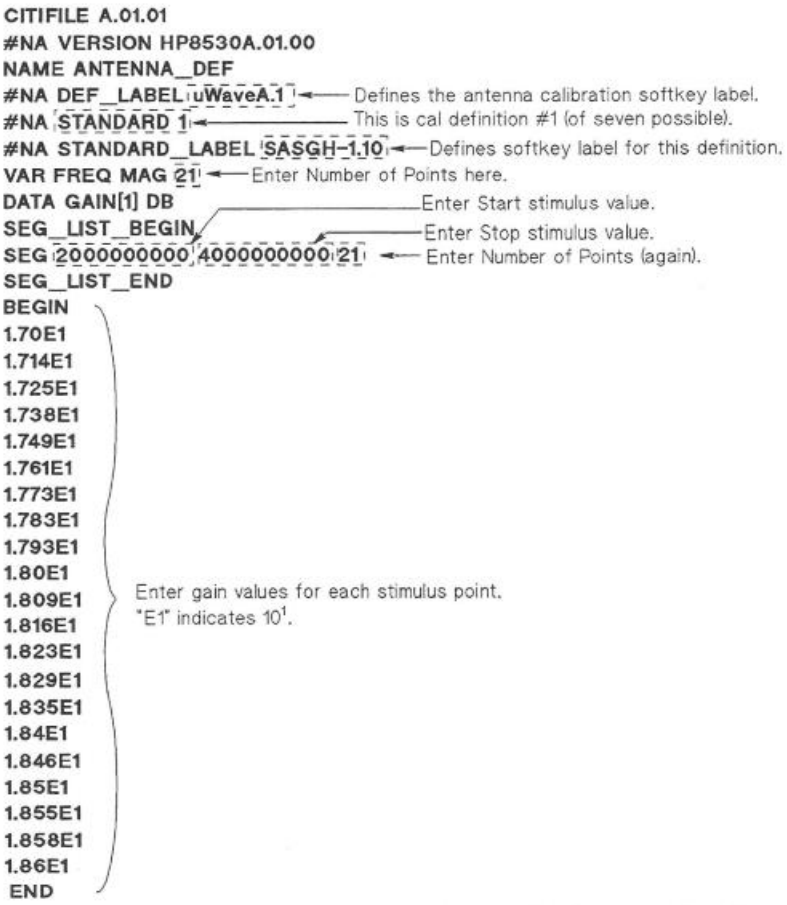

Figure 16: Standard Gain Antenna Definition Example (Acquired from [8])

Seven antenna definitions are allowed in a single calibration definition file. This is achieved by adding an antenna definition following another in the file. Except, the user should not add the first four lines, or header lines, again. Also, the user should make sure "#NA STANDARD #" is updated, such that, the first antenna definition is "STANDARD 1", the second antenna definition is "STANDARD 2", etc. [8]. The user should also make sure to change "#NA STANDARD LABEL" for each antenna definition created in the file. The names should not include underscores and only use keys A through Z, numbers 0 through 9, and dashes (-) [8]. An example of two antenna definitions in a single file can be seen in Figure 17.

```
CITIFILE A.01.01
#NA VERSION HP8530A.00.26
NAME ANTENNA_DEF
#NA DEF_LABEL uWave A.1
#NA STANDARD 1
#NA STANDARD_LABEL SASGH-1.10
VAR FREQ MAG 10
DATA GAIN[1] DB
SEG_LIST_BEGIN
SEG 1100000000 1700000000 10
SEG_LIST_END
BEGIN
1.63E1
1.64E1
1.65E1
1.66E1
1.67E1
1.68E1
1.685E1
1.69E1
1.7E1
1.71E1
END
#NA STANDARD 2
#NA STANDARD_LABEL SA12-1.70
VAR FREQ MAG 10
DATA GAIN[1] DB
SEG_LIST_BEGIN
SEG 1700000000 2600000000 10
SEG_LIST_END
BEGIN
1.61E1
1.62E1
1.64E1
1.65E1
1.67E1
1.68E1
1.69E1
1.7E1
1.71E1
1.72E1
END
```

Figure 17: Two Standard Gain Antenna Definition Example (Acquired from [8])

CALIBRATION PROCEDURE

The HP 8530A should first be loaded with an MS-DOS compatible disc with an ASCII file that follows the CITIfile formatting. This file should contain the antenna definitions for a frequency range that is within the frequency range of the actual measurement. To load the antenna calibration definition, press the following buttons on the network analyzer in the following order:

(1) Disc

(2) LOAD

(3) CAL KITS

(4) ANTENNA CAL DEF

The screen will now show the files on the diskette. The user can manually select the desired file by turning the dial. Next press LOAD FILE [8]. Currently, when opening the GUI, the network analyzer performs a user preset. File selection is not possible via soft keys. Therefore, only having the wanted antenna definition file on the diskette simplifies loading the file. This means having a single button to load the antenna definition is possible. The code below loads the user defined antenna definition by calling the function loadfile(netA), when pushing the button “Load Ant Def” on the GUI tab “Calibration” shown in Figure 18:

```
function loadfile(netA)
    fprintf(netA.port,'menudisc; soft3; soft7; soft2; soft8;'); %add a wait before soft key
    %currently only will load one the first file
end
```

Currently, the code will begin to load the diskette. However, the network analyzer has an error showing the file when using the computer to load the file. It will display to use the knob to select the file but will return to the home screen. Therefore, it is necessary to load the file manually.

The buttons Open GPIB Port and Close GPIB Port make this possible without having to close the GUI. These two buttons will be discussed later. If using the diskette that came with the HP8350A and an antenna calibration is performed, the screen will show a list of Narda horns in the softkey menu for the user to select. It is important to note that the calibration data on AC_NAR1 is not necessarily the same as the horns the university owns [8]. If the calibration is being performed over a wide frequency range, the user may need to use more than one standard gain horn. Also, if the two antenna definitions have overlapping frequencies, the network analyzer receiver will use the calibration data for the standard gain horn last measured [8].

Since it is recommended to perform a frequency domain calibration, the rest of this section will be covering the process and code that is to perform this type of calibration. After the above is completed, we want to begin initial setup and find boresight. The following is a step-by-step process given by the manual if [8]:

1. Mount the standard gain horn on the AUT side on the anechoic chamber.
2. Press RECALL then press MORE followed by FACTORY RESET. I currently do not recommend doing this step, since I am unsure if this will remove currently loaded Cal Sets.
3. Press DOMAIN then press FREQUENCY
4. Press the STIMULUS MENU then press SINGLE POINT
5. Press CENTER and enter a frequency in the approximate center of the calibration frequency range
6. Press MARKER
7. Have the source horn oriented in the desired position, i.e. vertical or horizontal (this is because the AUT currently cannot change axes
8. Have the standard gain horn boresight

9. Move the AUT until the flat line reaches maximum amplitude (or minimum if the antenna has a null at boresight)
10. Repeat steps 5, 6, and 7 for each axis until true boresight is found

The function that allows the user to before a boresight is below, and the button on the GUI is labeled “Boresight” as shown in Figure 18:

```
function boresight(netA)
    centerfreq = 0;
    centerfreq = (netA.setStartFreq(netA)+ netA.setStopFreq(netA))/2;
    %fprintf(netA.port,'menureca; soft8; soft8;'); %factory preset. Unsure if will clear stored
    %non-volatile memory, so it is currently commented out
    fprintf(netA.port,'menudoma; soft1;'); %domain -> frequency
    fprintf(netA.port,'menustim; soft5;'); %stimulus menu -> single point
    fprintf(netA.port,'centerfreq MHz;'); %enter center frequency;  menumark always sets
    %C.W. to 4 GHz so I skipped this instruction
end
```

At this point there are two options, calibrate for a single frequency or over a frequency sweep. The following instructions will continue assuming a calibration for a frequency sweep. If the frequency sweep overlaps more than one standard gain horn, enter the entire frequency sweep during the procedure below. Using the recommended Frequency List mode, perform the following steps [8]:

1.
    i. Press STIMULUS MENU then press MORE followed by EDIT LIST
    ii. Press EDIT then press SEGMENT: START and enter the start frequency for the actual measurement

iii. Press STOP and enter the stop frequency for the actual measurement

iv. Press STEP SIZE and enter the desired frequency increment value. The user can use smaller increment values than the documented gain values step size. The network analyzer will interpolate to make this possible.

v. Press DONE

vi. Press DONE again

vii. Press FREQUENCY LIST

2. Press CAL key (located in the MENUS block), then press your DEF_LABEL name from the loaded file. For example, ANT. CAL uWave A.1 from Figure 16. Then press FAR FIELD: RESPONSE.

   At this point, any antenna definitions created in the loaded file will appear. This cannot exceed seven standard gain antenna definitions.

3. Select the desired definition for the current standard gain horn.

   This will begin the measurement, and create a list of offsets call a “Cal Set” upon completion. If only one standard gain horn is to be measured, press DONE RESPONSE.

In the Calibration tab shown in Figure 18, the button “Freq Domain Cal” will call the function bellow to perform the frequency domain calibration:

```
function freqdomainCal(netA)
    fprintf(netA.port,'menustim; soft8; soft6; soft3; soft1; netA.setStartFreq(netA)MHz;');
    %path to segment: start
    fprintf(netA.port,'soft2; netA.setStopFreq(netA)MHz;'); %set stop freq
    fprintf(netA.port,'soft3; netA.stepsize(netA);;'); %set step size
```

```
fprintf(netA.port,'soft8; soft8; soft4'); %press done twice then select freq list
fprintf(netA.port,'menucal; soft5; soft1;'); %open the file that was loaded to network
%analyzer and press far field response
fprintf(netA.port,'soft1; soft8'); % select the desired definition and press done response.
%may need to be adjusted in future
end
```

Step 4 is for the user to use multiple gain horns when the frequency range is too wide for a single antenna definition.

4. If the frequency range for the actual measurement requires the calibration using more than one standard gain horn, follow the following steps:
   i. Mount the next standard gain horn.
   ii. If necessary, boresight the new horn. Do not leave the current Domain and Sweep mode. For example, if the user is at the Frequency List, leaving this will ruin calibration. With a marker still on, adjust the horn so that boresight is found. There will be a delay between completed sweeps so adjust the horn in small steps.
   iii. Select the antenna definition for the horn via softkey menu.
   iv. Repeat these sub-steps under (4) for every horn that the user needs to measure.
   v. Press DONE RESPONSE once finished.

   If the network analyzer displays CAUTION: ADDITIONAL STANDARDS NEEDED, then the definitions on the loaded file does not cover the entire frequency range that you are trying to perform a calibration over for your desired antenna measurement [8].
5. A cal set containing the offsets is almost complete. Currently, this cal sets contains a

correction for gain relative to the measured standard gain horn(s) and response correction one would typically see when calibrating on a stand-alone network analyzer. The next step is to perform an isolation calibration, which reduces crosstalk between inputs. This can be skipped by pressing OMIT ISOLATION.

The Calibration tab in the GUI, shown in Figure 18, includes the option to omit isolation by pressing Omit Isolation. The code below allows the user to omit isolation as part of the calibration:

```
function omitisolationCal(netA)
    fprintf(netA.port,'menucal; soft5; soft3;'); %omit isolation cal
end
```

If you have chosen not to skip isolation:

6. Replace the standard gain horn with a 50 Ω load.
7. Press ISOLATION

The Isolation button can be observed in Figure 18 and calls the following function:

```
function isolationCal(netA)
    fprintf(netA.port,'menucal; soft5; soft2;'); %perform isolation cal
end
```

8. Press SAVE FAR FIELD. Save the cal set to any of the eight resgisters that appear on the screen using the softkeys. Calibration will be turned on, and any measurement will be calibrated. It is also possible to save the calibration to a diskette [8].

To save the cal set with the GUI, press the button Save Cal shown in Figure 18. The Save Cal button calls the following code:

```
function saveCal(netA)
    fprintf(netA.port,'menucal; soft5; soft8; soft1'); %save cal set to desired register; soft1
```

%save to register 1

end

9. MANUAL PROCEDURE: Insert a formatted disc the user would like to save the cal set data too.
    i. Press DISC then press STORE followed by CAL SET 1-8 and select the cal register that is desired to be stored.
    ii. Use the knob to select letters or numbers. When the pointer is on the desired letter or number press SELECT LETTER. Up to seven characters may be entered. Finally Press STORE FILE.

Earlier, it was mentioned that the network analyzer would interpolate the definition data to create cal set data. However, this does not happen during the actual measurement. The cal set frequencies must match exactly with the actual measurement frequencies. The user needs to make sure that when selecting step sizes for the Frequency List, the frequencies will match during the actual measurements [8]. Also, the user may find it necessary to manually take control of the network analyzer after opening the GUI. Once the GPIB port has been opened, manually pressing buttons on the network analyzer is not possible. To allow the user to have the option to rotate between computer controlled and manual controlled, two buttons called Open GPIB Port and Close GPIB Port are available. Closing the GPIB port allows the user to have manual control of the network analyzer and opening the GPIB port allows the computer to control it.

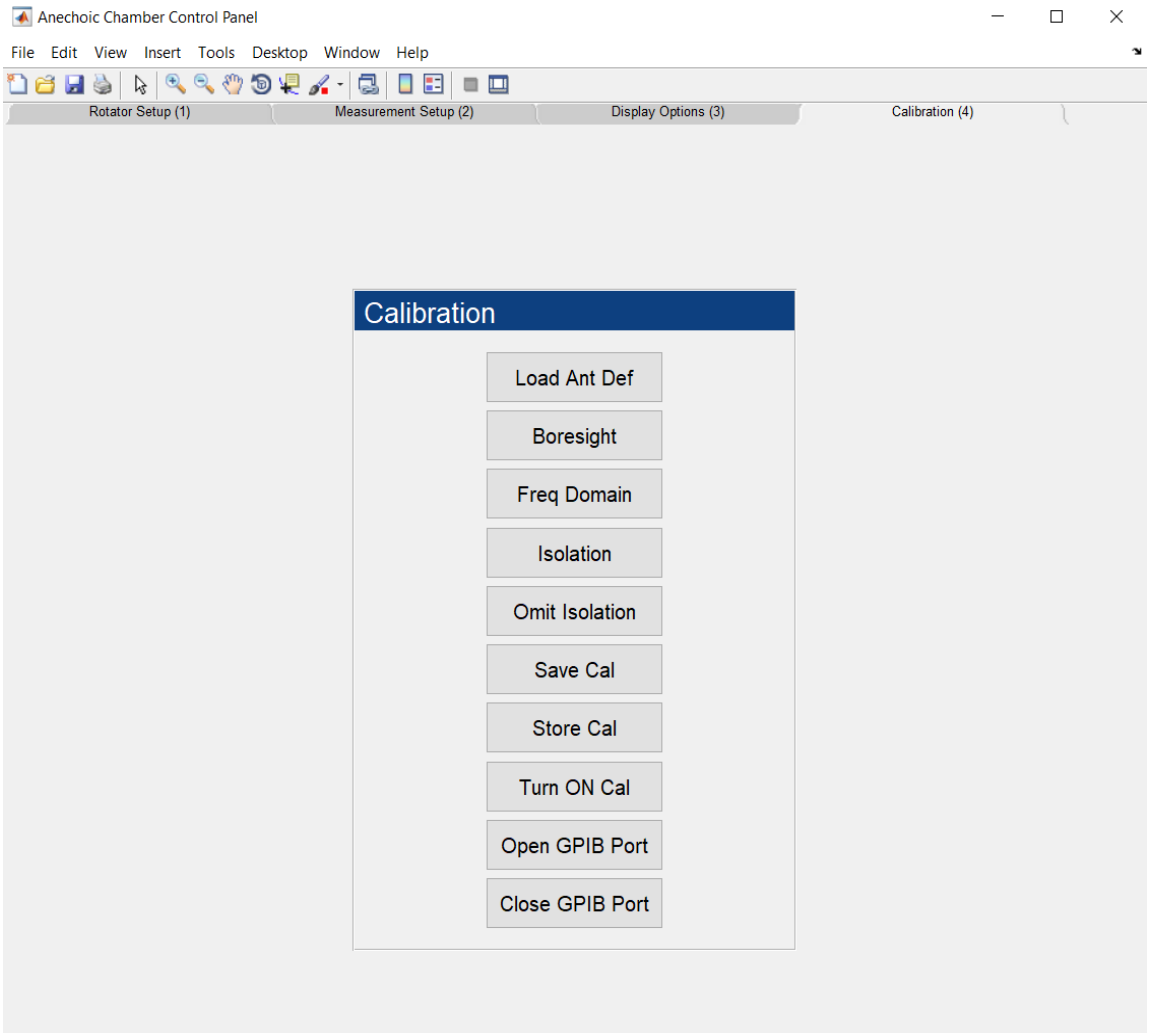

Figure 18: Anechoic Chamber GUI Calibration Tab

The following procedure of instructions is the recommended way to run a measurement with a cal set. This method does involve opening and closing the port for certain steps. In addition, this procedure does not include the boresight step from the manual. For future students and professors, this should be done for calibrated measurements.

1. **Mount Standard Gain Horn on AUT side of chamber**
2. **Put diskette into HP8530A drive**
3. **Start the GUI from Matlab by running chmaberGUI1.m**. This is necessary because on the GUI startup, an internal source trigger begins.

4. **Initialize AUT (GUI)**. This ensures the AUT is set to 0 degrees
5. **Click the Measurement Setup Tab (GUI)**
6. **Enter Start Frequency (GUI)**
7. **Enter Stop Frequency (GUI)**
8. **Open the Frequency Points drop down menu and select desired number of points (GUI)**. Currently, this relies on soft keys to perform, so it cannot be below 51 points.
9. **Click the Calibration tab (GUI)**
10. **Push Load Ant Def (GUI)**. This will take the user to the desired locating to load an antenna and will close the port between the computer and the network analyzer. From this point, the user will need to manual operate the network analyzer until the Open GPIB Port button is pushed.
11. **Soft key 2 (Antenna Cal Def)**
12. **Use knob to find your Antenna Definition**
13. **Soft key 8 to load file**
14. **Push Open GPIB Port (GUI)**
15. **Push Freq Domain (GUI).** This takes the user to the edit Frequency List menu. The User will need to manually operate the network analyzer until the port is open again.
16. **Soft key 1. Enter start frequency**
17. **Soft key 2. Enter stop frequency**
18. **Soft key 5. Enter number of points.** This number of points should be equal to the number of points the user put in the GUI's Measurement Setup tab.
19. **Soft key 8 (Done)**
20. **Soft key 8 (Done).**

21. **Soft key 4 (Freq List)**
22. **Menu Cal**
23. **Soft key 5**
24. **Soft key 1 (Far Field: Response)**
25. **Select your Antenna Definition.** This could be any soft key depending if you have multiple definitions in a single file.
26. **Wait for calculations**
27. **Soft 8 (Done Response)**
28. **Push Open GPIB Port (GUI)**
29. **Push Omit Isolation (GUI).** Isolation is not always needed. One can choose to skip this for most measurements.
30. **Push Save Cal (GUI).** This directs the user to the location to save the Cal Set to a register. This will close port and will require manually doing the following instructions, until the port is open again.
31. **Select a register to save Cal Set using soft keys 1-8**
32. **Push Open GPIB Port (GUI)**
33. **Push Store Cal Set (GUI).** This will close port and will require manually doing the following instructions, until the port is open again.
34. **Select the register the Cal Set was saved to from step 31.** Should be underlined on screen
35. **Name File**
36. **Soft key 8 (Store File)**
37. **Remove Diskette after storing is complete**
38. **Close GUI**

39. **Change the three desired save functions in the Sweep_Callback in chamber_GUI1.m to save the data files to the desired location and names**
40. **Press User Preset on network analyzer**
41. **Unmount the standard gain horn**
42. **Mount the antenna for desired mesuarements on the AUT side**
43. **Open GUI.** If GUI does not reset the network analyzer do the following:
    i. **Turn off the monitor and receiver**
    ii. **Turn back on, and then**
    iii. **Open GUI**
44. **Initialize AUT (GUI)**
45. **Open Measurement Setup Tab (GUI)**
46. **Enter Start Frequency (GUI)**
47. **Enter Stop Frequency (GUI)**
48. **Select frequency number of points (GUI)**
49. **Press Start (GUI).** This is when actual measurements begins.
50. **After measurement is complete, load diskette into floppy drive and save the cal set colleted from steps 30-34 to the computer as a .txt file**
51. **Open Post_Processing_Calibrated.m**
52. **Do one of the following:**
    i. **Uncomment the section of code that plots a single frequency from a wide variety of frequencies. Then comment out the section that averages a frequency. Do this if your START and STOP frequencies were different**
    ii. **OR uncomment section of code that plots an average of the data, then comment**

out the other section. Do this if your START and STOP frequencies are equal.

53. **Run Post_Processing_Calibrated.m**

54. **Select your cal set**

55. **Select the saved file that contained the variable 'Z'**

56. **The user should now have a graph of the far field pattern's magnitude and phase.**

After performing the steps above, a proof of concept graph can be observed in Figure 19. Comparing the figure to Figure 15, the difference can be observed; the level has been calibrated so that the graph is showing the gain of the antenna. Thus, the calibration has been applied to the measured data. The calibration also corrected the dip that appeared in the phase graph in figure 14. An averaged and calibrated phase graph can be observed in Figure 20.

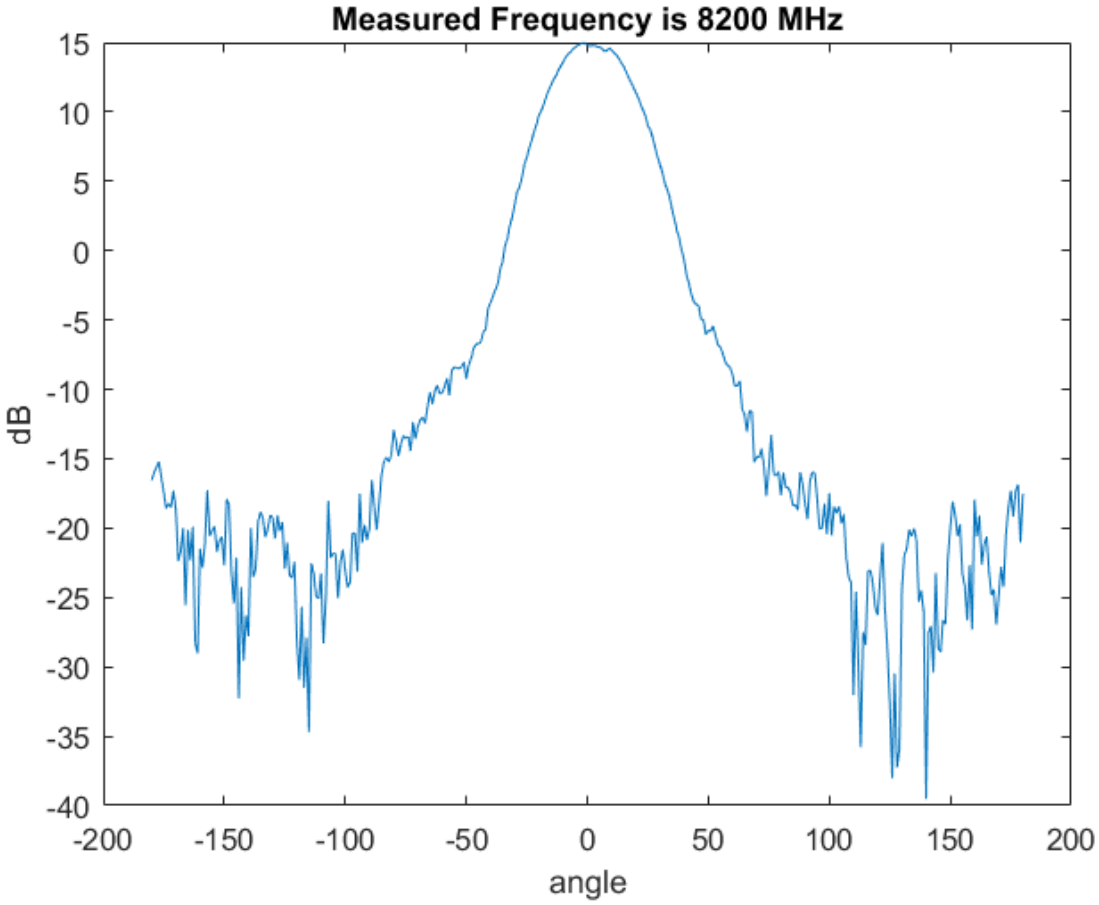

Figure 19: Averaged and Calibrated NARDA 640 dB Sweep Data Collection Example

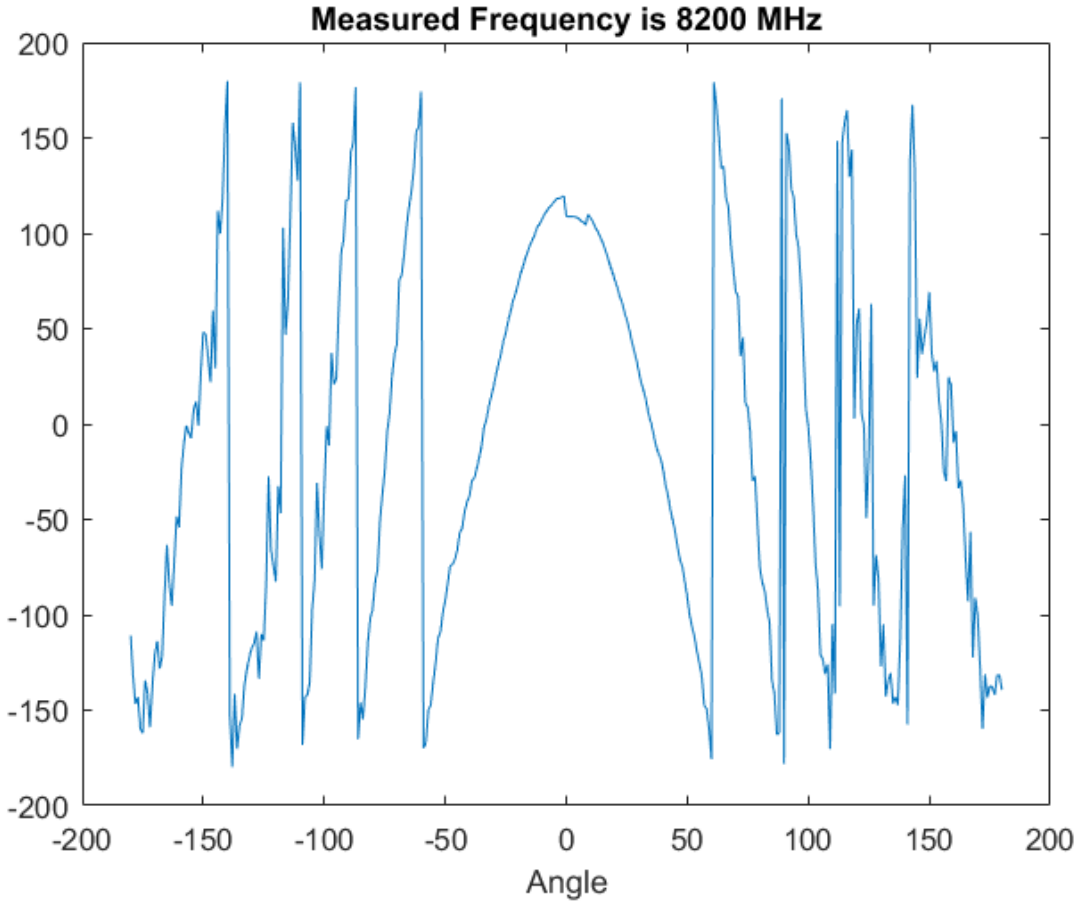

Figure 20: Averaged and Calibrated NARDA 640 Phase Sweep Data Collection Example

Appendix H has a polar plot of the NARDA 640 calibrated by Applied Silicon Inc. Canada [9]. The polar plot in appendix H does not show the backside of the measurement, and it contains mostly the main beam. However, comparing Figure 21 to the polar plot in appendix H, it is revealed that an accurate measurement was obtained. At 0° both measurements are roughly 15 dB, and at 45° appendix H shows approximately 2.5 dB, where as the measured results inside the Ole Miss anechoic chamber resulted in approximately 0 dB. However, at -45° the standard gain horn resulted in approximately -7dB. This could be a result of the AUT not being perfectly stable and having perferct boresight. Also, gain values for side lobes were close to noise levels. This could be corrected by using an amplifer to aplly more power to the system.

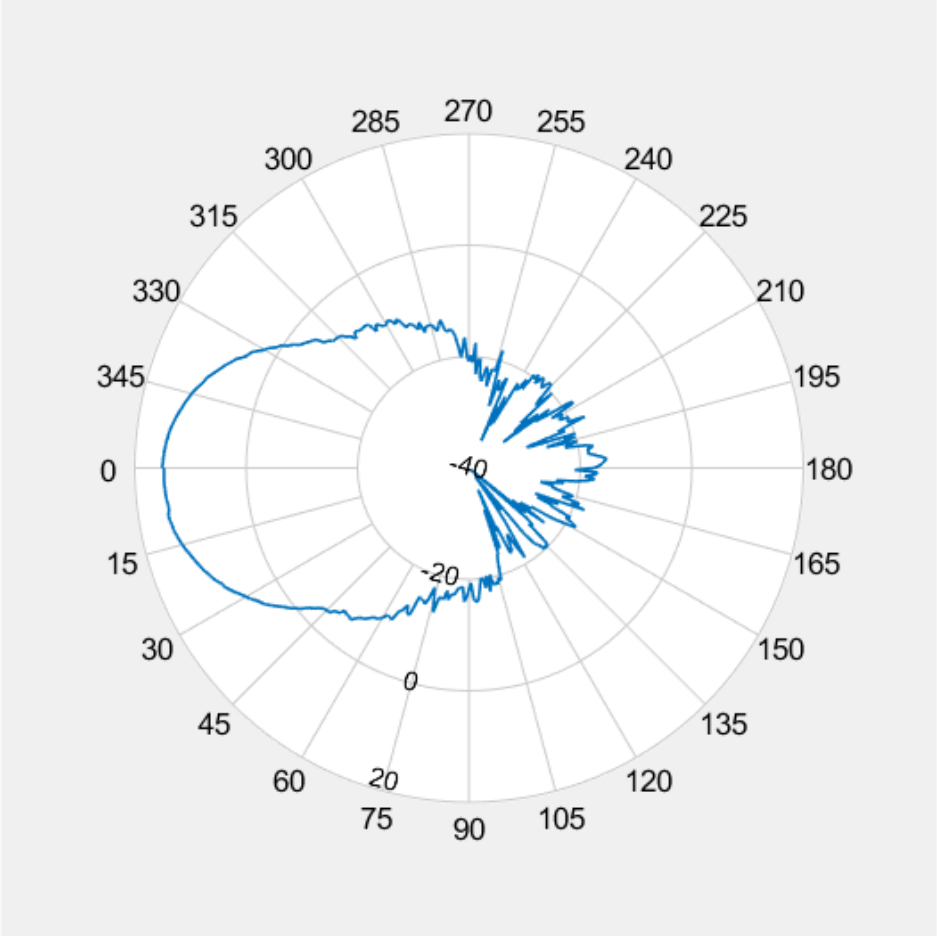

Figure 21: Averaged and Calibrated NARDA 640 Polar Plot Example

CONCLUSION

By automating key parts of the calibration and measurement process in the Ole Miss anechoic chamber, it is now possible to take more measurements in a shorter time, and gain measurements can be performed. This will aid in the research of future students and professors by allowing for faster collection of the data, which leads to faster design and test times. As an added benefit, it has made an older set of equipment more usable than originally designed and prolonged its useful life. This will save the University and the School of Engineering time and money. Software extensions of previous work in Matlab will also make the processing and analysis of the data easier. Solving the problem of getting sweep data with each angle in an automated process and the formatting of that data in an array has allowed for faster processing of measured data. Just as this work was built on previous work, future students would benefit by continuing the development and improvement of this system.

LIST OF REFERENCES

REFERENCES

[1] National Instruments, "Improving RF System Accuracy With Methodical Calibration Techniques," 21 Feburary 2013. [Online]. Available: http://www.ni.com/tutorial/9864/en/. [Accessed 7 May 2019].

[2] IEEE, "C63.5-2017 - American National Standard for Electromagnetic Compatibility--Radiated Emission Measurements in Electromagnetic Interference (EMI) Control--Calibration and Qualification of Antennas (9 kHz to 40 GHz)," 8 May 2017. [Online]. Available: https://ieeexplore-ieee-org.umiss.idm.oclc.org/document/7920447. [Accessed 10 June 2019].

[3] C. Capps, Delphi Automotive Systems, 16 August 2001. [Online]. Available: https://people.eecs.ku.edu/~callen58/501/Capps2001EDNpp95.pdf. [Accessed 7 May 2019].

[4] W. L. Stutzman and G. A. Thiele, Antenna Theory and Design 3rd Edition, Wiley, 2012.

[5] H. Shakhtour, R. Cornelius and D. Heberling, "Three Antenna Gain Determination Method in Compact Antenna Test Ranges," in *Loughborough Antennas & Propagation Conference (LAPC)*, Loughborough, 2013.

[6] IEEE, "IEEE Recommended Practice for Near-Field Antenna Measurements," IEEE, 2012.

[7] A. Newell and G. Hindman, "NSI-MI," EuCAP 2011, 2011. [Online]. Available: https://www.nsi-mi.com/images/Technical_Papers/2011/2011EuCAP_Antenna_Pattern_Comparison_Using_Pattern_Subtraction.pdf. [Accessed 7 May 2019].

[8] HP, Operating and Programming Manual: HP 8530A Reciever Edition 2, HP, 1994.

[9] R. Moore, "EARS Acceptance Test and Calibration Document," Applied Silicon Inc. Canada, Ottawa, 1996.

LIST OF APPENDICES

APPENDIX A

```
% HP8530A.m
% Corey Garner
% 1/7/2014
% Edit: 6/19/2019
% This defines a network analyzer, HP8530A

classdef HP8530A < handle
    properties
        port;
        startFreq;
        stopFreq;
        centerFreq;
        spanFreq;
        incrementFreq;
        markFreq;
        unit;
        format;
        points;
        measurementType;
        dataA;
        realData;

    end


    methods(Static)
        %constructor
```

```
function netA = HP8530A
    % Find a GPIB object.
    netA.port = instrfind('Type', 'gpib', 'BoardIndex',...
                          0, 'PrimaryAddress', 16, 'Tag', '');
    % Create the GPIB object if it does not exist
    % otherwise use the object that was found.
    if isempty(netA.port)
        netA.port = gpib('NI', 0, 16);
    else
       fclose(netA.port);
       netA.port = netA.port(1);
    end

    netA.port.InputBufferSize = 100000;

    % Set property values
    netA.startFreq = 2000; %MHz
    netA.stopFreq = 10000;     %MHz
    netA.centerFreq = 6000; %MHz
    netA.spanFreq = 8000;     %MHz
    netA.incrementFreq = 1;
    netA.markFreq = 8;
    netA.unit = 'MHz';
    netA.format = 'pol';
    netA.points = 11;
    netA.measurementType = 'phas';
    netA.dataA = '';
    netA.realData = 0;
end
```

```
% Opens the netA port
function openPort(netA)
    % Connect to instrument object, HP8530A.
    fopen(netA.port);
end

% Closes the netA port
function closePort(netA)
    fclose(netA.port);
%       delete(netA.port);
end

% Factory preset of netA
function preset(netA)
    fprintf(netA.port, 'factpres;wait;'); % Set to factory preset
    fprintf(netA.port, 'S21;'); % Set to view S21 (Thru)
end

% Sets the start frequency of netA
function setStartFreq(netA)
    str = sprintf('star %d %s;', netA.startFreq, netA.unit);
    fprintf(netA.port, str);
end

% Sets the stop frequency of netA
function setStopFreq(netA)
    str = sprintf('stop %d %s;', netA.stopFreq, netA.unit);
    fprintf(netA.port, str);
end
```

```
% Sets the center frequency of netA
function setCenterFreq(netA)
   str = sprintf('cent %d %s;', netA.centerFreq, netA.unit);
   fprintf(netA.port, str);
end

% Sets the span frequency of netA
function setSpanFreq(netA)
   str = sprintf('span %d %s;', netA.spanFreq, netA.unit);
   fprintf(netA.port, str);
end

% blg Sets the number of points of netA in frequency sweep
function setNumPoints(netA)
   switch netA.points
         case 51  % User selects 1
      fprintf(netA.port,'menustim; soft3; soft1');
            case 101  % User selects 2
               fprintf(netA.port,'menustim; soft3; soft2');
            case 201  % User selects 5
               fprintf(netA.port,'menustim; soft3; soft3');
            case '401'  % User selects 10
      fprintf(netA.port,'menustim; soft3; soft4');
   case '801'  % User selects 20
      fprintf(netA.port,'menustim; soft3; soft5');
   case '1601'
               fprintf(netA.port,'menustim; soft3; soft6');
      end
end
```

```
% Sets the start and stop frequency of netA and sets the marker to
% the start frequency
function setFreqRange(netA)
   str = sprintf('star %d %s;stop %d %s;poin%d', netA.startFreq,...
         netA.unit, netA.stopFreq, netA.unit, netA.points);
   fprintf(netA.port, str);
   str1 = sprintf('mark1 %d %s;', netA.startFreq, netA.unit);
   fprintf(netA.port, str1);
end

% Sets the format of netA
function setFormat(netA)
   str = sprintf('calc1:form %s;', netA.format);
   fprintf(netA.port, str);
end

% Measures the frequency of the marker of netA
function x = measureFreq(netA)
   x = str2double(query(netA.port, 'calc1:mark1:x?;'));
end

% Measures the magnitude of the marker of netA
function y1 = measureMag(netA)
   y1 = str2double(query(netA.port, 'calc1:mark1:y:magn?;'));
end

% Measures the phase of the marker of netA
function y2 = measurePhase(netA)
   y2 = str2double(query(netA.port, 'calc1:mark1:y:phas?;'));
end
```

```
% Set the frequency of the marker
function setMarkerX(netA)
   str = sprintf('calc1:mark1:x %d %s;', netA.markFreq, netA.unit);
   fprintf(netA.port, str);
end

% Increment marker frequency
function incMarkFreq(netA)
   netA.markFreq = netA.markFreq + netA.incrementFreq;
   netA.setMarkerX(netA);
end

% Stops the image
function triggerON(netA)
   disp('in trigger')
   fprintf(netA.port, 'abort\n');
   fprintf(netA.port, 'init1:cont off\n');
   fprintf(netA.port, 'init1;\n');
end

% Starts the image
function triggerOFF(netA)
   fprintf(netA.port, 'init1:cont on\n;');
end

% Calculates the number of points to be measured
function incrementCalc(netA)
   netA.incrementFreq = ((netA.stopFreq - netA.startFreq) / (netA.points-1));
end
```

```
% Gets the trace from the network analyzer
function A = trace(netA)
   fprintf(netA.port, 'sens1:swe:poin %i;', netA.points);
   fprintf(netA.port, 'form:data ascii,5;');
   if strcmp(netA.format,'phas')
      fprintf(netA.port, 'calc2:form %s\n', netA.format);
      A = query(netA.port, 'CALC2:DATA?\n');
   else
      fprintf(netA.port, 'calc1:form %s\n', netA.format);
      A = query(netA.port, 'CALC1:DATA?\n');
   end

end

% Takes a measurement
function Q = measure(netA)
   disp('in measure')
   netA.triggerON(netA);
   Q = netA.trace(netA);
   netA.triggerOFF(netA);
end

function angle_preset(netA)
%        str = sprintf(' cent 180 HZ; span 360 hz; menustim; soft3 1 HZ; soft8; soft2; menustim; soft5; soft6');
      fprintf(netA.port,'menudoma; soft3; menustim; soft2; 0.935 GHz;');%, netA.startFreq);
      fprintf(netA.port,'menustim; soft8; soft5; soft6; S21;'); % connect aut and src sma cables together after this line
      fprintf(netA.port,'menucal; soft4; soft2; soft3; wait; soft8; soft3;'); % this will perform a thru calibration. connect the cable ends together(tx and rx). this should be a separate function.
```

```
        fprintf(netA.port,'menustim; soft8; soft5; soft7; logp;');
        fprintf(netA.port,'cent 180 HZ; span 360 hz; menustim; soft3 1 HZ; soft6;');
%           fprintf(netA.port,'soft2; menustim; soft5 soft6;');
        %fprintf(netA.port,'SIGN; MENUCOPY; SOFT3; SOFT2; FORM4; OUTPDATA;');
    end

    function connect_S21(netA)
        fprintf(netA.port,'menustim; soft8; soft5; soft6; S21;'); % connect aut and src sma cables together after this line
    end

    function thru_calibration(netA)
       fprintf(netA.port,'menucal; soft4; soft2; soft3; wait; soft8; soft3;'); % this will perform a thru calibration. connect the cable ends together(tx and rx).
    end

    function sweep_preset(netA) %blg
        fprintf(netA.port,'menudoma; soft1;');
        netA.setStartFreq(netA);
        netA.setStopFreq(netA);
        netA.connect_S21(netA);
        fprintf(netA.port,'menustim; soft8; soft5; soft7; logp;');
        disp(netA.points)
          netA.setNumPoints(netA);
    end

    function logPlot(netA)
        % This sets trigger mode. soft 6 is internal trigger. soft 7 is
        % HPIB. Then set to s21 measurement
        fprintf(netA.port,'menustim; soft8; soft5; soft6; S21;');
```

```
    end

    function trigger_netA(netA)
       trigger(netA.port)
    end

    function data = getData(netA)
%         fprintf(netA.port, 'POIN201;PARA1;LOGM;SING;FORM4;');
        disp('in getData')
%          fprintf(netA.port, 'HOLD;FORM4;INPUDATA');
%          fprintf(netA.port, 'SING; MENUDISP; SOFT7; FORM4; OUTPDATA');
%         fprintf(netA.port, ' MENUCOPY; SOFT3; SOFT2; FORM4; OUTPDATA'); % this
seems to be it!
%         fprintf(netA.port, 'POIN 361; PARA1; LOGM; SING;');
       fprintf(netA.port, 'FORM4;');
       fprintf(netA.port, 'OUTPDATA;');
        disp('in getData and sent setup commands')
        pause(1)
%          data = fscanf(netA);
        data = fscanf(netA.port);
        disp('in getData read data')
%           disp(data)
    end

    function data2 = formatData(netA)
       c = 0;
       r = 1;

       cmplxStr = netA.dataA;                  %grab points
```

```
        cmplxCell = strread(cmplxStr,'%s','delimiter',',');    %split real and imaginary value of
point


        comma = ',';
        count = sum(ismember(netA.dataA,comma)); %number of points in dataA
       for i = 1:count
        c = c+1;
        realChar = cell2mat(cmplxCell(r,1));    %
        realNum = str2num(realChar);            %convert the real data value to a number
        imagChar = cell2mat(cmplxCell(r+1,1));    %
        imagNum = 1i.*(str2num(imagChar));      %convert the imaginary data value to a number
        data2(c,1) = realNum + imagNum;         %store the complex value into a single cell
        r = r+2;
       end
       save('C:\temp\cmplxCell_2019_4_24.mat','cmplxCell');
    end


    function loadfile(netA)
        %fprintf(netA.port,'menutape; soft#; soft#; soft#; soft8;');%pushes tape/disc button
        fprintf(netA.port,'menudisc; soft3; soft7;'); %add a wait before soft key 8 currently only
will load one the first file
        netA.closePort(netA);
    end




    function boresight(netA)
        centerfreq = 0;
        centerfreq = (netA.setStartFreq(netA)+ netA.setStopFreq(netA))/2;
        %fprintf(netA.port,'menureca; soft8; soft8;'); %factory preset
```

```
    fprintf(netA.port,'menudoma; soft1;'); %domain -> frequency
    fprintf(netA.port,'menustim; soft5;'); %stimulus menu -> single point
    fprintf(netA.port,'centerfreq MHz;'); %enter center frequency;  menumark always sets C.W. to 4 GHz so I skipped this instruction
end

function freqdomainCal(netA)
    fprintf(netA.port,'menustim; soft8; soft6; soft3;'); %path to segment: start
    %fprintf(netA.port,'soft2; netA.setStopFreq(netA)MHz;'); %set stop freq
    %fprintf(netA.port,'soft3; netA.stepsize(netA););'); %set step size
    %fprintf(netA.port,'soft8; soft8; soft4'); %press done twice then select freq list
    %fprintf(netA.port,'menucal; soft5; soft1;'); %open the file that was loaded to network analyzer and press far field response
    %fprintf(netA.port,'soft1; soft8'); % select the desired definition and press done response. may need to be adjusted in future
    netA.closePort(netA);
end

function isolationCal(netA)
    fprintf(netA.port,'soft2;'); %perform isolation cal. menucal; soft5; soft2;
end

function omitisolationCal(netA)
    fprintf(netA.port,'soft3;'); %omit isolation cal menucal; soft5; soft3;
end

function saveCal(netA)
    fprintf(netA.port,'soft8;'); %save cal set to desired register; menucal; soft5; soft8
     netA.closePort(netA);
end
```

```
    function storeCal(netA)
        fprintf(netA.port,'menudisc; soft2; soft5;'); %directs user to store cal set to diskette
        netA.closePort(netA);
    end

    function calON(netA)
       fprintf(netA.port,';menucal; soft1; soft6'); %turn on cal set on soft key 1
    end

  end

end
```

APPENDIX B

```
% chamberGUI.m
% Corey Garner
% 8/17/2013
% Edit: 6/19/2019
% This purpose of this program is to design a better looking, more coherent
% design than antennaGUI.fig. The user should have a better sense of what
% to do in the program. Help buttons will be included. More error coding
% will be included, hopefully.
%
% This needs to be turned into a function type eventually.

%
% Naming convention: a mix of c-style and lowerCamelCase.
% type_nameOfObject_specialidentifier
% hb = horizontal box
% vb = vertical box
% pb = push button
% bb = button box
% rb = radio button
% tb = text box
% cb = check box
% eb = edit box
% pum = pop up menu


function varargout = chamberGUI1(varargin)
% ANTENNAGUI MATLAB code for antennaGUI.fig
%      ANTENNAGUI, by itself, creates a new ANTENNAGUI or raises the existing
%      singleton*.
%
%      H = ANTENNAGUI returns the handle to a new ANTENNAGUI or the handle to
%      the existing singleton*.
%
%      ANTENNAGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ANTENNAGUI.M with the given input arguments.
%
%      ANTENNAGUI('Property','Value',...) creates a new ANTENNAGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before antennaGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to antennaGUI_OpeningFcn via varargin.

%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help antennaGUI

% Last Modified by GUIDE v2.5 05-Aug-2013 11:00:17

% Begin initialization code - DO NOT EDIT
mOutputArgs = {};
% hsplash = splash('splash/Chamber_splash.jpg'); % Creates splash screen
hMainFigure = chamberGUI_LayoutFcn;

mOutputArgs{1} = hMainFigure;
if nargout>0
   [varargout{1:nargout}] = mOutputArgs{:};
end
chamberGUI_OpeningFcn(hMainFigure);
% disp(hMainFigure)
% pause(1)
% splash(hsplash,'off')   % Destroys splash screen
% End initialization code - DO NOT EDIT
end

function chamberGUI_OpeningFcn(hObject, eventdata, handles, varargin)
handles = guidata(gcf);

handles.output = hObject;
% handles.guifig = gcf;
% disp()
% Initialize several variables
% global f  c polarMin polarMax new_extrema plane sourcePol;
% c = 0;
% f = 'trash.csv';
% polarMin = get(handles.editPolarMin,'Value');
% polarMax = get(handles.editPolarMax,'Value');
% new_extrema = 0;
% sourcePol = 0;
% plane = 1;

% Create timer
handles.tmr = timer;
set(handles.tmr,'timerfcn', {@position_callback,handles.guifig},...
                'ExecutionMode','FixedRate',...
                'Period',0.1);

% Antenna Under Test
```

```
handles.rot_aut = rotator('COM3',800);      % create new rotator object and set initSteps
handles.rot_aut.openPort(handles.rot_aut);        % opens the rotator serial port

% Source Antenna
handles.rot_source = rotator('COM4',756);   %create new rotator object and set initSteps
handles.rot_source.openPort(handles.rot_source);    %opens the rotator serial port

% Network Analyzer
%handles.netA = HP8712;      % create new HP8712 object (may not work with this gui)
handles.netA = HP8530A;      % create new HP8530A object
handles.netA.openPort(handles.netA);      % opens the network analyzer port

% Array-handler object
% handles.graph = plotter;

% updating after every 0.1 secs
guidata(handles.guifig,handles);

start(handles.tmr)
% Factory preset of netA
handles.netA.preset(handles.netA);
% Update handles structure
guidata(hObject, handles);
% disp(get(handles.tmr))
% UIWAIT makes antennaGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);

end

% This is the constructor (Layout Function)
function gui_fig = chamberGUI_LayoutFcn(hObject, eventdata, handles, varargin)

% Create a new figure.
gui_fig = figure( 'Position', [ 0 2000 900 800],...
                'Name', 'Anechoic Chamber Control Panel',...
                'NumberTitle', 'off', 'Visible','off' );
movegui(gui_fig,'center')
set( gui_fig, 'CloseRequestFcn', @my_closeFcn );

% Set up handles structure
handles = guihandles( gui_fig );

% Create a tab panel layout within gui_fig.
handles.tabLayout = uiextras.TabPanel( 'Parent', gui_fig , 'Padding', 5 );

handles.guifig = gui_fig;
```

```
% Set text sizes
panelTextSize1 = 16;
panelTextSize2 = 14;
buttonTextSize = 12;
textSize = 12;
buttonSize = [ 140 40 ];
buttonSize2 = [ 70 40 ];


% ---------------------------------------------------------------
% ---------------------- Rotator Setup Tab ----------------------
% ---------------------------------------------------------------

% Create horizontal box layout within tabLayout
hb_1 = uiextras.HBox( 'Parent', handles.tabLayout,...
                'Padding', 10, 'Spacing', 20 );

% Create two blank boxes, AUT panel, and Source panel. Empty boxes on
% either side allow centering.
hb_blank1 = uiextras.HBox('Parent',hb_1);
vb_blank1 = uiextras.VBox('Parent',hb_1);
vb_blank2 = uiextras.VBox('Parent',hb_1);
hb_blank2 = uiextras.HBox('Parent',hb_1);

uiextras.VBox('Parent', vb_blank1);
uipanel_aut = uiextras.BoxPanel( 'Parent', vb_blank1, 'Title', ' AUT',...
                        'FontSize', panelTextSize1);
uiextras.VBox('Parent', vb_blank1);

uiextras.VBox('Parent', vb_blank2);
uipanel_source = uiextras.BoxPanel( 'Parent', vb_blank2, 'Title', ' Source',...
                          'FontSize', panelTextSize1 );
uiextras.VBox('Parent', vb_blank2);

% ---------------------------------------------------------------
%                         AUT
% ---------------------------------------------------------------

% Create vertical box layout within AUT panel
vb_1 = uiextras.VBox( 'Parent' ,uipanel_aut,...
                'Padding', 20, 'Spacing', 20 );

% Create horizontal box layouts within vb_1
hbb_1 = uiextras.HButtonBox( 'Parent', vb_1, 'Padding', 0,...
                    'ButtonSize', buttonSize );
```

```
% ---------------------- Init and zero ref ----------------------

% Create a pushbutton for initializing the AUT. Set callback fcn.
handles.pb_initialize_aut = uicontrol( 'Parent', hbb_1,...
                                'Style', 'pushbutton',...
                                'String', 'Initialize',...
                                'FontSize', buttonTextSize );
set(handles.pb_initialize_aut, 'Callback', {@initialize_aut, handles})

% Create a pushbutton for setting the zero reference of the AUT. Set
% callback fcn.
handles.pb_zeroRef_aut = uicontrol( 'Parent', hbb_1,...
                               'Style', 'pushbutton',...
                               'String', 'Set Zero Ref',...
                               'FontSize', buttonTextSize );
set(handles.pb_zeroRef_aut, 'Callback', {@zeroRef_aut, handles})

% Create box panels within vb_1 horizontal boxes. Increment, Direction, and
% Position.
uipanel_increment_aut = uiextras.BoxPanel( 'Parent', vb_1,...
                                  'Title', ' Increment',...
                                  'FontSize', panelTextSize2 );
uipanel_direction_aut = uiextras.BoxPanel( 'Parent',vb_1,...
                                  'Title', ' Direction',...
                                  'FontSize', panelTextSize2 );
uipanel_position_aut = uiextras.BoxPanel( 'Parent', vb_1,...
                                 'Title', ' Position',...
                                 'FontSize', panelTextSize2 );

% Create vertical box layout within uipanel_increment_aut
vb_2 = uiextras.VBox( 'Parent' ,uipanel_increment_aut,...
                 'Padding', 5, 'Spacing', 0 );

% Create horizontal button box layout within uipanel_direction_aut and
% vb_2.
handles.bg_dir_aut = uibuttongroup( 'Parent' ,uipanel_direction_aut,...
                       'BorderType', 'none');
hbb_2 = uiextras.HButtonBox( 'Parent' ,vb_2, 'Padding', 5, 'Spacing', 5,...
                     'ButtonSize', buttonSize );
hbb_3 = uiextras.HButtonBox( 'Parent' ,vb_2, 'Padding', 5, 'Spacing', 5,...
                     'ButtonSize', buttonSize );

% ------------------------ Increment buttons ----------------------

% Create plus one pushbutton. Set callback.
```

```
handles.pb_plusOne_aut = uicontrol( 'Parent', hbb_2,...
                            'Style', 'pushbutton',...
                            'String', '+1',...
                            'FontSize', buttonTextSize );
set(handles.pb_plusOne_aut, 'Callback', {@plusOne_aut, handles} )

% Create plus one pushbutton. Set callback.
handles.pb_minusOne_aut = uicontrol( 'Parent', hbb_2,...
                             'Style', 'pushbutton',...
                             'String', '-1',...
                             'FontSize', buttonTextSize );
set(handles.pb_minusOne_aut, 'Callback', {@minusOne_aut, handles} )

% Create plus one pushbutton. Set callback.
handles.pb_plusFive_aut = uicontrol( 'Parent', hbb_3,...
                             'Style', 'pushbutton',...
                             'String', '+5',...
                             'FontSize', buttonTextSize );
set(handles.pb_plusFive_aut, 'Callback', {@plusFive_aut, handles} )

% Create plus one pushbutton. Set callback.
handles.pb_minusFive_aut = uicontrol( 'Parent', hbb_3,...
                              'Style', 'pushbutton',...
                              'String', '-5',...
                              'FontSize', buttonTextSize );
set(handles.pb_minusFive_aut, 'Callback', {@minusFive_aut, handles} )

% Create radio buttons to control rotator direction within hbb_2. Set
% callback
handles.rb_cw_aut = uicontrol( 'Parent', handles.bg_dir_aut,...
                        'Style', 'radiobutton',...
                        'String', 'CW',...
                        'FontSize', buttonTextSize,...
                        'Units', 'normalized',...
                        'Position', [ 0.2 0.3 0.3 0.5 ] );
%                           'Position', [ 55 10 60 22] );
handles.rb_ccw_aut = uicontrol( 'Parent', handles.bg_dir_aut,...
                         'Style', 'radiobutton',...
                         'String', 'CCW',...
                         'FontSize', buttonTextSize,...
                         'Units', 'normalized',...
                         'Position', [ 0.55 0.3 0.3 0.5 ] );
%                            'Position', [ 130 10 60 22]);
set(handles.bg_dir_aut, 'SelectionChangeFcn', @rb_control_aut,...
                'SelectedObject', handles.rb_cw_aut);
```

```
% Create textbox within p5
handles.tb_position_aut = uicontrol( 'Style', 'Text',...
                            'Parent', uipanel_position_aut );
set( handles.tb_position_aut, 'String', '0 degrees',...
                        'FontSize', textSize );

% Set v1 box sizes. -1 fills space
set( vb_1, 'Sizes', [ 50 160 90 70 ] )
% set( vb_1, 'Sizes', [ -1 -2 -1 -1 ] )
set(hb_1, 'Sizes', [ -1 300 300 -1])
set(vb_blank1, 'Sizes', [-1 510 -1])
set(vb_blank2, 'Sizes', [-1 510 -1])
% align( [ handles.rb_cw_aut handles.rb_ccw_aut ], 'distribute', 'bottom' );

% ---------------------------------------------------------------
%                          Source
% ---------------------------------------------------------------

% Create vertical box layout for Source panel
vb_4 = uiextras.VBox( 'Parent' ,uipanel_source,...
                'Padding', 20, 'Spacing', 20 );

% Create three horizontal box layouts within v2
% h = horizontal, BB = button box, PB = panel box
hbb_4 = uiextras.HButtonBox( 'Parent', vb_4,...
                        'Padding', 5, 'Spacing', 5,...
                        'ButtonSize', buttonSize );

% Create two buttons within hBB1, initialize and reference. Set callback.
handles.pb_initialize_source = uicontrol( 'Parent', hbb_4,...
                                    'Style', 'pushbutton',...
                                    'String', 'Initialize',...
                                    'FontSize', buttonTextSize );
set(handles.pb_initialize_source, 'Callback', {@initialize_source, handles})

handles.pb_zeroRef_source = uicontrol( 'Parent', hbb_4,...
                                'Style', 'pushbutton',...
                                'String', 'Set Zero Ref',...
                                'FontSize', buttonTextSize );
set(handles.pb_zeroRef_source, 'Callback', {@zeroRef_source, handles})

% Create panels within parent horizontal boxes
uipanel_increment_source = uiextras.BoxPanel( 'Parent', vb_4,...
                                    'Title', ' Increment',...
                                    'FontSize', panelTextSize2 );
uipanel_direction_source = uiextras.BoxPanel( 'Parent', vb_4,...
```

```
                                    'Title', ' Direction',...
                                    'FontSize', panelTextSize2 );
uipanel_position_source = uiextras.BoxPanel( 'Parent', vb_4,...
                                    'Title', ' Position',...
                                    'FontSize', panelTextSize2 );

% Create vertical box layout within uipanel_incrementSource
vb_5 = uiextras.VBox( 'Parent' ,uipanel_increment_source,...
                'Padding', 5, 'Spacing', 5 );

handles.bg_dir_source = uibuttongroup( 'Parent', uipanel_direction_source,...
                    'BorderType', 'none');

% Create horizontal box layouts within v5
hbb_5 = uiextras.HButtonBox( 'Parent' ,vb_5,...
                    'Padding', 5, 'Spacing', 5,...
                    'ButtonSize', buttonSize );
hbb_6 = uiextras.HButtonBox( 'Parent' ,vb_5,...
                    'Padding', 5, 'Spacing', 5,...
                    'ButtonSize', buttonSize );

% Create buttons to control rotator increment within hBB2 and hBB3
handles.pb_plusOne_source = uicontrol( 'Parent', hbb_5,...
                                'Style', 'pushbutton',...
                                'String', '+1',...
                                'FontSize', buttonTextSize );
set(handles.pb_plusOne_source, 'Callback', {@plusOne_source, handles})

handles.pb_minusOne_source = uicontrol( 'Parent', hbb_5,...
                                'Style', 'pushbutton',...
                                'String', '-1',...
                                'FontSize', buttonTextSize );
set(handles.pb_minusOne_source, 'Callback', {@minusOne_source, handles})

handles.pb_plusFive_source = uicontrol( 'Parent', hbb_6,...
                                'Style', 'pushbutton',...
                                'String', '+5',...
                                'FontSize', buttonTextSize );
set(handles.pb_plusFive_source, 'Callback', {@plusFive_source, handles})

handles.pb_minusFive_source = uicontrol( 'Parent', hbb_6,...
                                'Style', 'pushbutton',...
                                'String', '-5',...
                                'FontSize', buttonTextSize );
set(handles.pb_minusFive_source, 'Callback', {@minusFive_source, handles})
```

```
% Create radio buttons to control rotator direction within v2
handles.rb_cw_source = uicontrol( 'Parent', handles.bg_dir_source,...
                          'Style', 'radiobutton',...
                          'String', 'CW',...
                          'FontSize', buttonTextSize,...
                          'Units', 'normalized',...
                          'Position', [ 0.2 0.3 0.3 0.5 ] );
%                             'Position', [ 55 10 60 22] );
handles.rb_ccw_source = uicontrol( 'Parent', handles.bg_dir_source,...
                           'Style', 'radiobutton',...
                           'String', 'CCW',...
                           'FontSize', buttonTextSize,...
                           'Units', 'normalized',...
                           'Position', [ 0.55 0.3 0.3 0.5 ] );
%                              'Position', [ 130 10 60 22] );
set(handles.bg_dir_source, 'SelectionChangeFcn', @rb_control_source,...
               'SelectedObject', handles.rb_cw_source);

% Create textbox within p5
handles.tb_position_source = uicontrol( 'Style', 'Text',...
                                'Parent', uipanel_position_source );
set( handles.tb_position_source, 'String', '0 degrees',...
                           'FontSize', textSize );

% Set v4 box sizes. -1 fills space
set( vb_4, 'Sizes', [ 50 160 90 70 ] )
% set(vb_4, 'Sizes', [ -1 -2 -1 -1 ] )

% ---------------------------------------------------------------
% -------------------- Measurement Setup Tab ---------------------
% ---------------------------------------------------------------

hb_2 = uiextras.HBox( 'Parent', handles.tabLayout,...
                'Padding', 10, 'Spacing', 20 );

vb_blank1 = uiextras.VBox( 'Parent' ,hb_2 );
vb_7 = uiextras.VBox( 'Parent' ,hb_2, 'Padding', 0, 'Spacing', 10 );
vb_8 = uiextras.VBox( 'Parent' ,hb_2, 'Padding', 0, 'Spacing', 10 );
vb_blank2 = uiextras.VBox( 'Parent' ,hb_2 );

% Create four panels
uiextras.VBox( 'Parent' ,vb_7 );
bp_autFreqRange = uiextras.BoxPanel( 'Parent', vb_7,...
                             'Title', ' AUT Frequency Range',...
                             'FontSize', panelTextSize1 );
bp_autAngRange = uiextras.BoxPanel( 'Parent', vb_7,...
```

```
                                'Title', ' AUT Angular Range',...
                                'FontSize', panelTextSize1 );

uiextras.VBox( 'Parent' ,vb_7 );
uiextras.VBox( 'Parent' ,vb_8 );
bp_autPlaneCut = uiextras.BoxPanel( 'Parent', vb_8,...
                                'Title', ' AUT Plane Cut',...
                                'FontSize', panelTextSize1 );
bp_sourcePol = uiextras.BoxPanel( 'Parent', vb_8,...
                               'Title', ' Source Polarization',...
                               'FontSize', panelTextSize1 );
bp_measure = uiextras.BoxPanel( 'Parent', vb_8,...
                              'Title', ' Measure',...
                              'FontSize', panelTextSize1 );
uiextras.VBox( 'Parent' ,vb_8 );

% Create vertical box layout in AUT_Freq_Range
grid_autFreqRange = uiextras.Grid('Parent', bp_autFreqRange,...
                               'Spacing', 20, 'Padding', 10);
grid_autAngRange = uiextras.Grid('Parent', bp_autAngRange,...
                               'Spacing', 20, 'Padding', 10);

% Create horizontal box layout for ;
bg_planeCut_aut = uibuttongroup( 'Parent' ,bp_autPlaneCut,...
                                'BorderType', 'none' );
bg_pol_source = uibuttongroup( 'Parent' ,bp_sourcePol,...
                              'BorderType', 'none' );
hbb_7 = uiextras.HButtonBox( 'Parent', bp_measure,...
                            'Padding', 10, 'Spacing', 5,...
                            'ButtonSize', buttonSize );

% Text and edit boxes AUT_Freq_Range
% First column
tb_3 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                  'String', 'Start', 'FontSize', textSize );
tb_5 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                  'String', 'Stop', 'FontSize', textSize );
tb_7 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                  'String', 'Freq Pts', 'FontSize', textSize );
tb_9 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                  'String', 'Freq Inc', 'FontSize', textSize );
% Second column
handles.eb_startFreq = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Edit',...
                       'BackgroundColor','w', 'FontSize', textSize );
handles.eb_stopFreq = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Edit',...
                      'BackgroundColor','w', 'FontSize', textSize );
```

```
numPoints_list = ['   ';'11 ';'21 ';'51 ';'101';'201';'401'];
list = cellstr(numPoints_list);
handles.pum_numPoints = uicontrol( 'Parent', grid_autFreqRange,...
                        'Style', 'Popupmenu',...
                        'String', list,...
                        'BackgroundColor','w',...
                        'FontSize', textSize );

set(handles.pum_numPoints, 'Callback', {@freq_inc_popupmenu_Callback,handles})
%                   'SelectedObject', handles.rb_cw_source);

handles.tb_freqInc = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                    'String', 'X', 'FontSize', textSize );

% Third column
tb_4 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                'String', 'MHz', 'FontSize', textSize );
tb_6 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                'String', 'MHz', 'FontSize', textSize );
tb_8 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                'String', '#', 'FontSize', textSize );
tb_10 = uicontrol( 'Parent', grid_autFreqRange, 'Style', 'Text',...
                'String', 'MHz', 'FontSize', textSize );

% Text and edit boxes AUT_Ang_Range
tb_11 = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Text',...
                'String', 'Start', 'FontSize', textSize );
tb_13 = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Text',...
                    'String', 'Stop', 'FontSize', textSize );
tb_15 = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Text',...
                'String', 'Ang Inc', 'FontSize', textSize );

handles.eb_startAngle = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Edit',...
                        'BackgroundColor','w', 'FontSize', textSize,...
                        'String', '0');
handles.eb_stopAngle = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Edit',...
                        'BackgroundColor','w', 'FontSize', textSize,...
                        'String', '361');
handles.eb_angleInc = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Edit',...
                        'BackgroundColor','w', 'FontSize', textSize,...
                        'String', '1');

tb_12 = uicontrol( 'Style', 'Text', 'Parent', grid_autAngRange,...
                'String', 'Degrees', 'FontSize', textSize );
tb14 = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Text',...
                    'String', 'Degrees', 'FontSize', textSize );
```

```
tb_16 = uicontrol( 'Parent', grid_autAngRange, 'Style', 'Text',...
                'String', 'Degrees', 'FontSize', textSize );

% plane cut radio buttons
rb_xy = uicontrol( 'Parent', bg_planeCut_aut, 'Style', 'radiobutton',...
                'String', 'XY', 'FontSize', buttonTextSize,...
                'Position', [ 40 10 60 22 ] );
rb_yz = uicontrol( 'Parent', bg_planeCut_aut, 'Style', 'radiobutton',...
                'String', 'YZ', 'FontSize', buttonTextSize,...
                'Position', [ 115 10 60 22 ] );
rb_xz = uicontrol( 'Parent', bg_planeCut_aut, 'Style', 'radiobutton',...
                'String', 'XZ', 'FontSize', buttonTextSize,...
                'Position', [ 190 10 60 22 ] );

% source positon radio buttons
rb_vertical = uicontrol( 'Parent', bg_pol_source, 'Style', 'radiobutton',...
                    'String', 'Vertical', 'FontSize', textSize,...
                    'Position', [ 30 30 100 22 ]);
rb_horizontal = uicontrol( 'Parent', bg_pol_source, 'Style', 'radiobutton',...
                     'String', 'Horizontal', 'FontSize', textSize,...
                     'Position', [ 150 30 100 22 ]);

% measure buttons
handles.pb_start = uicontrol( 'Parent', hbb_7, 'String', 'Start',...
                  'FontSize', buttonTextSize );
%blg set(handles.pb_start, 'Callback', {@Start_Callback,handles}); %use for
%single frequency
set(handles.pb_start, 'Callback', {@Sweep_Callback,handles}); %use for a sweep frequency

handles.pb_stop = uicontrol( 'Parent', hbb_7, 'String', 'Stop',...
                'FontSize', buttonTextSize );
set(handles.pb_stop, 'Callback', {@Stop_Callback,handles});

set(grid_autFreqRange,'ColumnSizes', [-1 -1 -1],'RowSizes',[-1 -1 -1 -1])
set(grid_autAngRange,'ColumnSizes', [-1 -1 -1],'RowSizes',[-1 -1 -1])

set( vb_7, 'Sizes', [ -1 220 180 -1 ] )
set( vb_8, 'Sizes', [ -1 90 200 100 -1 ] )
% set( vb_7, 'Sizes', [ -1 -1 -0.25 ] )
% set( vb_9, 'Sizes', [ 40 40 40 40 ] )
% set( vb_9, 'Sizes', [ 40 40 40 40 ] )
% set( vb_10, 'Sizes', [ 40 40 40 ] )
set( hb_2, 'Sizes', [ -1 300 300 -1 ] )

% ------------------------------------------------------------
% ----------------------Display Options Tab ------------------------
```

```
% ---------------------------------------------------------------

hb_10 = uiextras.HBox( 'Parent', handles.tabLayout, 'Padding', 10, 'Spacing', 10 );
vb_11 = uiextras.VBox( 'Parent', hb_10, 'Padding', 10, 'Spacing', 10 );
vb_12 = uiextras.VBox( 'Parent', hb_10, 'Padding', 10, 'Spacing', 10 );

% Create panels
plotType = uiextras.BoxPanel( 'Parent', vb_11,...
                        'Title', 'Plot Type',...
                        'FontSize', panelTextSize1);
indepParam = uiextras.BoxPanel( 'Parent', vb_11,...
                         'Title', 'Independent Parameter',...
                         'FontSize', panelTextSize1 );
plotProp = uiextras.BoxPanel( 'Parent', vb_11,...
                        'Title', 'Plot Properties',...
                        'FontSize', panelTextSize1 );
polarParam = uiextras.BoxPanel( 'Parent', vb_11,...
                         'Title', 'Polar Parameters',...
                         'FontSize', panelTextSize1 );
loadData = uiextras.BoxPanel( 'Parent', vb_11,...
                        'Title', 'Load',...
                        'FontSize', panelTextSize1 );
saveData = uiextras.BoxPanel( 'Parent', vb_11,...
                        'Title', 'Save',...
                        'FontSize', panelTextSize1 );

% Create axes for plotting
axes1 = axes('Parent', vb_12);
axes2 = axes('Parent', vb_12);

% Create box layouts and button groups
bg_plotType = uibuttongroup( 'Parent', plotType, 'BorderType', 'none' );

hb_11 = uiextras.HBox( 'Parent', polarParam, 'Padding', 0, 'Spacing', 10 );

vb_13 = uiextras.VBox( 'Parent', hb_11, 'Padding', 10, 'Spacing', 10 );
vb_14 = uiextras.HBox( 'Parent', indepParam, 'Padding', 0, 'Spacing', 10 );
vb_15 = uiextras.VBox( 'Parent', plotProp, 'Padding', 0, 'Spacing', 0 );

hbb_8 = uiextras.HButtonBox( 'Parent', loadData,...
                        'Padding', 10, 'Spacing', 10,...
                        'ButtonSize', buttonSize );
hbb_9 = uiextras.HButtonBox( 'Parent', saveData,...
                        'Padding', 10, 'Spacing', 10,...
                        'ButtonSize', buttonSize );
```

```
hb_12 = uiextras.HBox( 'Parent', vb_13, 'Padding', 5, 'Spacing', 10 );
hb_13 = uiextras.HBox( 'Parent', vb_13, 'Padding', 5, 'Spacing', 10 );
hb_14 = uiextras.HBox( 'Parent', vb_13, 'Padding', 5, 'Spacing', 10 );

bg_indepParam = uibuttongroup( 'Parent', vb_14, 'BorderType', 'none' );
vb_16 = uiextras.VBox( 'Parent', vb_14, 'Padding', 10, 'Spacing', 10 );

hb_15 = uiextras.HBox( 'Parent', vb_15, 'Padding', 10, 'Spacing', 10 );
% hb_16 = uiextras.HBox( 'Parent', vb_15, 'Padding', 10, 'Spacing', 10 );

% Buttons, text boxes, etc
% plot type
rb_rect = uicontrol( 'Parent', bg_plotType,...
                    'Style', 'radiobutton',...
                    'String', 'Rect',...
                    'FontSize', buttonTextSize,...
                    'Position', [ 55 10 70 25 ] );
rb_polar = uicontrol( 'Parent', bg_plotType,...
                     'Style', 'radiobutton',...
                     'String', 'Polar',...
                     'FontSize', buttonTextSize,...
                     'Position', [ 170 10 70 25 ] );

% independent parameter
rb_freq = uicontrol( 'Parent', bg_indepParam,...
                    'Style', 'radiobutton',...
                    'String', 'Freq',...
                    'FontSize', buttonTextSize,...
                    'Position', [ 30 17 75 25] );
rb_angle = uicontrol( 'Parent', bg_indepParam,...
                     'Style', 'radiobutton',...
                     'String', 'Angle',...
                     'FontSize', buttonTextSize,...
                     'Position', [ 30 55 75 25 ] );
pum_freqList = uicontrol( 'Parent', vb_16,...
                         'Style', 'Popupmenu',...
                         'String', 'Freq List',...
                         'BackgroundColor','w',...
                         'FontSize', buttonTextSize);
pum_angList = uicontrol( 'Parent', vb_16,...
                        'Style', 'Popupmenu',...
                        'String', 'Ang List',...
                        'BackgroundColor','w',...
                        'FontSize', buttonTextSize );
% plot properties
cb_grid = uicontrol( 'Parent', hb_15,...
```

```
                'Style', 'checkbox',...
                'String', 'Grid',...
                'FontSize', buttonTextSize );
cb_hold = uicontrol( 'Parent', hb_15,...
                'Style', 'checkbox',...
                'String', 'Hold',...
                'FontSize', buttonTextSize );
cb_norm = uicontrol( 'Parent', hb_15,...
                'Style', 'checkbox', ...
                'String', 'Norm',...
                'FontSize', buttonTextSize );
cb_dB = uicontrol( 'Parent', hb_15,...
              'Style', 'checkbox', ...
              'String', 'dB',...
              'FontSize', buttonTextSize );


% polar parameters
tb_17 = uicontrol( 'Parent', hb_12,...
              'Style', 'Text',...
              'String', 'Step Size',...
              'FontSize', textSize );
eb_stepSize = uicontrol( 'Parent', hb_12,...
                'Style', 'Edit',...
                'BackgroundColor', 'w',...
                'FontSize', textSize  );

tb_18 = uicontrol( 'Parent', hb_13,...
              'Style', 'Text',...
              'String', 'Maximum',...
              'FontSize', textSize );
eb_maxSize = uicontrol( 'Parent', hb_13,...
                'Style', 'Edit',...
                'BackgroundColor', 'w',...
                'FontSize', textSize  );

tb_19 = uicontrol( 'Parent', hb_14,...
              'Style', 'Text',...
              'String', 'Minimum',...
              'FontSize', textSize  );
eb_minSize = uicontrol( 'Parent', hb_14,...
                'Style', 'Edit',...
                'BackgroundColor', 'w',...
                'FontSize', textSize  );

% load data
```

```
pb_loadData = uicontrol( 'Parent', hbb_8,...
                    'String', 'Load Data Set',...
                    'FontSize', buttonTextSize );

% save data
pb_saveData = uicontrol( 'Parent', hbb_9,...
                    'String', 'Save?',...
                    'FontSize', buttonTextSize );

% box sizes
set( hb_10, 'Sizes', [ 350 -1 ] )
set( vb_11, 'Sizes', [ 80 120 80 180 80 80 ] );
set( vb_12, 'Sizes', [ -1 -1 ] );


% ---------------------------------------------------------------
% ----------------------Calibration Options Tab ------------------------
% ---------------------------------------------------------------

hb_3 = uiextras.HBox( 'Parent', handles.tabLayout,...
                'Padding', 10, 'Spacing', 20 );

vb_blank1 = uiextras.HBox( 'Parent' ,hb_3 );
vb_20 = uiextras.VBox( 'Parent' ,hb_3, 'Padding', 0, 'Spacing', 10 );
vb_blank2 = uiextras.HBox( 'Parent' ,hb_3 );

% Create one panel
uiextras.VBox( 'Parent' ,vb_20 );
bp_calibration = uiextras.BoxPanel( 'Parent', vb_20,...
                            'Title', ' Calibration',...
                            'FontSize', panelTextSize1 );
uiextras.VBox( 'Parent' ,vb_20 );

% Create vertical box layout in calibration
grid_cal = uiextras.Grid('Parent', bp_calibration,...
                        'Spacing', 20, 'Padding', 10);

% Text and edit boxes calibration
% First column
hbb_20 = uiextras.VButtonBox( 'Parent', bp_calibration,...
                    'Padding', 10, 'Spacing', 5,...
                    'ButtonSize', buttonSize );
handles.pb_LCalReg1 = uicontrol( 'Parent', hbb_20, 'String', 'Load Ant Def',...
                'FontSize', buttonTextSize );
handles.pb_boresight = uicontrol( 'Parent', hbb_20, 'String', 'Boresight',...
                'FontSize', buttonTextSize );
```

```
handles.pb_freqdomainCal = uicontrol( 'Parent', hbb_20, 'String', 'Freq Domain',...
                'FontSize', buttonTextSize );
handles.pb_isolationCal = uicontrol( 'Parent', hbb_20, 'String', 'Isolation',...
                'FontSize', buttonTextSize );
handles.pb_omitisolationCal = uicontrol( 'Parent', hbb_20, 'String', 'Omit Isolation',...
                'FontSize', buttonTextSize );
handles.pb_saveCal = uicontrol( 'Parent', hbb_20, 'String', 'Save Cal',...
                'FontSize', buttonTextSize );
handles.pb_storeCal = uicontrol( 'Parent', hbb_20, 'String', 'Store Cal',...
                'FontSize', buttonTextSize );
handles.pb_calON = uicontrol( 'Parent', hbb_20, 'String', 'Turn ON Cal',...
                'FontSize', buttonTextSize );
handles.pb_openPort = uicontrol( 'Parent', hbb_20, 'String', 'Open GPIB Port',...
                'FontSize', buttonTextSize );
handles.pb_closePort = uicontrol( 'Parent', hbb_20, 'String', 'Close GPIB Port',...
                'FontSize', buttonTextSize );

set(handles.pb_LCalReg1, 'Callback', {@loadfile,handles}); %Load Antenna Def
set(handles.pb_boresight, 'Callback', {@boresight,handles}); %Boresight the AUT
set(handles.pb_freqdomainCal, 'Callback', {@freqdomainCal,handles}); %Frequency domain
calibration
set(handles.pb_isolationCal, 'Callback', {@isolationCal,handles}); %Isolation calibration
set(handles.pb_omitisolationCal, 'Callback', {@omitisolationCal,handles}); %Omit Isolation
calibration
set(handles.pb_saveCal, 'Callback', {@saveCal,handles}); %Save calibration data
set(handles.pb_calON, 'Callback', {@calON,handles}); %Turn ON calibration set.
set(handles.pb_storeCal, 'Callback', {@storeCal,handles}); %Store calibration set to diskette.
set(handles.pb_openPort, 'Callback', {@openPort,handles}); %Turn ON calibration set.
set(handles.pb_closePort, 'Callback', {@closePort,handles}); %Turn ON calibration set.



set( vb_20, 'Sizes', [-1 510 -1] )
set( hb_3, 'Sizes', [ -1 350 -1 ] )

% Name tabs
handles.tabLayout.TabSize = 200;
handles.tabLayout.SelectedChild = 1;
handles.tabLayout.TabNames = {'Rotator Setup (1)',...
                'Measurement Setup (2)','Display Options (3)',...
                'Calibration (4)'};

set(gui_fig,'Visible','on')
guidata(gui_fig, handles);
% data = guidata(gui_fig);
end
```

```
% ------------------------------------------------------------------
% ------------------------ Callbacks & Fcns ------------------------
% ------------------------------------------------------------------

function my_closeFcn(hObject, eventdata, handles)
   handles = guidata(gcbo);
   stop(handles.tmr);
   % close the rotator serial ports
   handles.rot_aut.closePort(handles.rot_aut);
   handles.rot_source.closePort(handles.rot_source);
   % close the network analyzer hpib port
   handles.netA.closePort(handles.netA);
   disp('Closing now!');
   % close gui
   delete(hObject);
end

function position_callback(hObject, eventdata, handles)
   handles = guidata(handles);
%     handles = guidata(gcbo);
   % reads the current position of the rotator(aut)
   p_aut = handles.rot_aut.position(handles.rot_aut);
%     str_p_aut = sprintf('%.3f degrees', p_aut);
   % reads the current position of the rotator(source)
   p_source = handles.rot_source.position(handles.rot_source);

   % Display position of the AUT and source rotators
   set(handles.tb_position_aut,'String',p_aut)
   set(handles.tb_position_source,'String',p_source)
   guidata(handles.guifig, handles);
   %guidata(hObject, handles);
end

% ------------------------------------------------
% ----------------------  AUT  ------------------------
% ------------------------------------------------

% --- Executes on button press in initialize_aut.
% Initializes the rotator to the zero position.
function initialize_aut(hObject, eventdata, handles)
   handles = guidata(gcbo);
   % Initializes the rotator to the 0 degree mark
   handles.rot_aut.initialize(handles.rot_aut);
end
```

```
% --- Executes on button press in zeroRef_aut.
function zeroRef_aut(hObject, eventdata, handles)
   handles = guidata(gcbo);
   % Set the current position as 0
   handles.rot_aut.zeroRef(handles.rot_aut);
end

% --- Executes on button press in minusFive_aut.
function minusFive_aut(hObject, eventdata, handles)
   handles = guidata(gcbo);
   handles.rot_aut.rotateSpeed = 1000; % Define rotator speed
   handles.rot_aut.speed(handles.rot_aut); % Set rotator speed
   handles.rot_aut.degree = -5;    % Define number of degrees to move
   handles.rot_aut.increment(handles.rot_aut); % Move rotator by set degree
   guidata(hObject, handles); % update
end


% --- Executes on button press in minusOne_aut.
function minusOne_aut(hObject, eventdata, handles)
   handles = guidata(gcbo);
   handles.rot_aut.rotateSpeed = 1000; % Define rotator speed
   handles.rot_aut.speed(handles.rot_aut); % set rotator speed
   handles.rot_aut.degree = -1;    % Define number of degrees to move
   handles.rot_aut.increment(handles.rot_aut); % Move rotator by set degree
   guidata(hObject, handles);
end


% --- Executes on button press in plusFive_aut.
function plusFive_aut(hObject, eventdata, handles)
   handles = guidata(gcbo);
   handles.rot_aut.rotateSpeed = 1000; % Define rotator speed
   handles.rot_aut.speed(handles.rot_aut);% set rotator speed
   handles.rot_aut.degree = 5;    % Define number of degrees to move
   handles.rot_aut.increment(handles.rot_aut); % Move rotator by set degree
   guidata(hObject, handles);
end


% --- Executes on button press in plusOne_aut.
function plusOne_aut(hObject, eventdata, handles)
   handles = guidata(gcbo);
   handles.rot_aut.rotateSpeed = 1000; % Define rotator speed
   handles.rot_aut.speed(handles.rot_aut); % set rotator speed
   handles.rot_aut.degree = 1; % Define number of degrees to move
```

```
    handles.rot_aut.increment(handles.rot_aut); % Move rotator by set degree
    guidata(hObject, handles);
end

% --- Executes on radio button change
function rb_control_aut(hObject, eventdata, handles)
    handles = guidata(gcbo);
    % If the radio button pushed in the button group is rb_cw_aut, then the
    % direction property is set to -1. Else the rb_ccw_aut button is pushed
    % and the direction is set to 1.
    if get(handles.bg_dir_aut, 'SelectedObject') == handles.rb_cw_aut
        handles.rot_aut.direct = -1; % Define direction as CW
    else
        handles.rot_aut.direct = 1; % Define direction as CCW
    end
    guidata(hObject, handles);
end

% ------------------------------------------------
%                    Source
% ------------------------------------------------


% --- Executes on button press in initialize_source.
function initialize_source(hObject, eventdata, handles)
    handles = guidata(gcbo);
    % Initializes the rotator to the 0 degree mark
    handles.rot_source.initialize(handles.rot_source);
end

% --- Executes on button press in zeroRef_source.
function zeroRef_source(hObject, eventdata, handles)
    handles = guidata(gcbo);
    % Set the current position as 0
    handles.rot_source.zeroRef(handles.rot_source);
end

% --- Executes on button press in minusFive_source.
function minusFive_source(hObject, eventdata, handles)
    handles = guidata(gcbo);
    handles.rot_source.rotateSpeed = 1000; % Define rotator speed
    handles.rot_source.speed(handles.rot_source); % set rotator speed
    handles.rot_source.degree = -5; % Define number of degrees to move
    handles.rot_source.increment(handles.rot_source); % Move rotator by set degree
    guidata(hObject, handles);
end
```

```
% --- Executes on button press in minusOne_source.
function minusOne_source(hObject, eventdata, handles)
   handles = guidata(gcbo);
   handles.rot_source.rotateSpeed = 1000;  % Define rotator speed
   handles.rot_source.speed(handles.rot_source);
   handles.rot_source.degree = -1; % Define number of degrees to move
   handles.rot_source.increment(handles.rot_source); % Move rotator by set degree
   guidata(hObject, handles);
end

% --- Executes on button press in plusFive_source.
function plusFive_source(hObject, eventdata, handles)
   handles = guidata(gcbo);
   handles.rot_source.rotateSpeed = 1000;  % Define rotator speed
   handles.rot_source.speed(handles.rot_source);
   handles.rot_source.degree = 5; % Define number of degrees to move
   handles.rot_source.increment(handles.rot_source); % Move rotator by set degree
   guidata(hObject, handles);
end

% --- Executes on button press in plusOne_source.
function plusOne_source(hObject, eventdata, handles)
   handles = guidata(gcbo);
   handles.rot_source.rotateSpeed = 1000;  % Define rotator speed
   handles.rot_source.speed(handles.rot_source);
   handles.rot_source.degree = 1; % Define number of degrees to move
   handles.rot_source.increment(handles.rot_source); % Move rotator by set degree
   guidata(hObject, handles);
end

function rb_control_source(hObject, eventdata, handles)
   handles = guidata(gcbo);
   % If the radio button pushed in the button group is rb_cw_aut, then the
   % direction property is set to -1. Else the rb_ccw_aut button is pushed
   % and the direction is set to 1.
   if get(handles.bg_dir_source, 'SelectedObject') == handles.rb_cw_source
      handles.rot_source.direct = -1; % Define direction as CW
   else
      handles.rot_source.direct = 1; % Define direction as CCW
   end
   guidata(hObject, handles);
end

% ------------------------------------------------
%                   Calibration
```

```
% ------------------------------------------------

% --- Allows user to load file.
function loadfile(hObject, eventdata, handles)

handles = guidata(gcbo);
handles.netA.loadfile(handles.netA);
guidata(hObject, handles);
end

% --- Allows user to boresight antenna.
function boresight(hObject, eventdata, handles)

   handles = guidata(gcbo);
   handles.netA.boresight(handles.netA);
   guidata(hObject, handles);
end

% --- Performs Freq Domain Cal
function freqdomainCal(hObject, eventdata, handles)

   handles = guidata(gcbo);
   handles.netA.freqdomainCal(handles.netA);
   guidata(hObject, handles);
end

% --- Allows user to perform isolation cal.
function isolationCal(hObject, eventdata, handles)

   handles = guidata(gcbo);
   handles.netA.isolationCal(handles.netA);
   guidata(hObject, handles);
end

% --- Allows user to omit isolation cal.
function omitisolationCal(hObject, eventdata, handles)

   handles = guidata(gcbo);
   handles.netA.omitisolationCal(handles.netA);
   guidata(hObject, handles);
end

% --- Save Cal
function saveCal(hObject, eventdata, handles)

   handles = guidata(gcbo);
```

```
   handles.netA.saveCal(handles.netA);
   guidata(hObject, handles);
end

% --- Store Cal set to diskette
function storeCal(hObject, eventdata, handles)

   handles = guidata(gcbo);
   handles.netA.storeCal(handles.netA);
   guidata(hObject, handles);
end

% --- Turn Cal ON
function calON(hObject, eventdata, handles)

   handles = guidata(gcbo);
   handles.netA.calON(handles.netA);
   guidata(hObject, handles);
end

% --- Opens Port.
function openPort(hObject, eventdata, handles)

handles = guidata(gcbo);
handles.netA = HP8530A;      % create new HP8530A object
handles.netA.openPort(handles.netA);      % opens the network analyzer port
disp('Port open!');
guidata(hObject, handles);
end

% --- Close Port
function closePort(hObject, eventdata, handles)

handles = guidata(gcbo);
handles.netA.closePort(handles.netA);
disp('Port closed!');
guidata(hObject, handles);
end

% ------------------------------------------------------------
%                Measurement Callbacks
% ------------------------------------------------------------

% --- Executes on selection change in freq_inc_popupmenu.
function freq_inc_popupmenu_Callback(hObject, eventdata, handles)
% Hints: contents = cellstr(get(hObject,'String')) returns freq_inc_popupmenu contents as cell
```

```
array
%       contents{get(hObject,'Value')} returns selected item from freq_inc_popupmenu
%set(handles.freq_inc_popupmenu,'Value',5);
   handles = guidata(gcbo);
%    disp(handles)
   val = get(hObject, 'Value');
   str = get(hObject, 'String');
   start = str2double(get(handles.eb_startFreq,'String')); % Get start freq from edit box
   stop = str2double(get(handles.eb_stopFreq,'String')); % Get stop freq from edit box
   points = str2double(str(val));  % get number of points selected
   % User mistakenly enters stop < start . Notify and set stop = start
   if stop < start
      disp('Error: Start must be <= stop.')
      disp('Stop has been changed to match Start.');
      set(handles.eb_stopFreq,'String',get(handles.eb_startFreq,'String'));
      stop = start;
   end
   calcFreqInc = (stop - start)/(points-1); % calculate freq increment
   % may need to uncomment netA.points eventually
   switch str{val}
%        case '3'   % User selects 1/16
%           handles.netA.points = 3;
%        case '5'  % User selects 1/8
%           handles.netA.points = 5;
      case '11'  % User selects 1/4
         set(handles.tb_freqInc,'String',calcFreqInc)
%           handles.netA.points = 11;
      case '21'  % User selects 1/2
         set(handles.tb_freqInc,'String',calcFreqInc)
%           handles.netA.points = 21;
      case '51'  % User selects 1
         set(handles.tb_freqInc,'String',calcFreqInc)
         handles.netA.points = 51;
%           handles.netA.points = 51;
      case '101'  % User selects 2
         set(handles.tb_freqInc,'String',calcFreqInc)
         handles.netA.points = 101;
%           handles.netA.points = 101;
     case '201'  % User selects 5
        set(handles.tb_freqInc,'String',calcFreqInc)
        handles.netA.points = 201;
%           handles.netA.points = 201;
%        case '401'  % User selects 10
%           set(handles.tb_freqInc,'String',calcFreqInc)
%            fprintf(netA.port,'menustim; soft3; soft4');
%           handles.netA.points = 401;
```

```
%        case '801'  % User selects 20
%           set(handles.tb_freqInc,'String',calcFreqInc)
%             fprintf(netA.port,'menustim; soft3; soft5');
%           handles.netA.points = 801;
%        case '1601'  % User selects 50
%           handles.netA.points = 1601;
   end

   handles.netA.startFreq = start;
   handles.netA.setStartFreq(handles.netA);
   handles.netA.stopFreq = stop;
   handles.netA.setStopFreq(handles.netA);
   handles.netA.connect_S21(handles.netA);
%    handles.netA.incrementCalc(handles.netA);
%    a = handles.netA.incrementFreq;
%    set(handles.tb_freqInc,'String',a);
   guidata(hObject, handles);
end

% --- Executes on button press in Start.
% --- This is the measuring function.
function Start_Callback(hObject, eventdata, ~)
   handles = guidata(gcbo);
   global stopflag;
   stopflag = 0;

   stop(handles.tmr);
   % Prepare the network analyzer for a scan
   handles.netA.angle_preset(handles.netA);
   pause(4) % give network analyzer a few seconds to complete

%    handles.rot_aut.spd = str2double(get(handles.editSpeed,'String'));
%    handles.rot_aut.spd = 400;
%    handles.rot_aut.speed(handles.rot_aut);
%    handles.netA.format = 'mlin';   %if you want dB, use mlog
%    handles.netA.setFormat(handles.netA);

   % Set the start, stop, and increment properties of the network analyzer (netA).
   % Then set the start and stop frequency of the netA.
   handles.netA.startFreq = str2double(get(handles.eb_startFreq,'String'));
%    handles.netA.setStartFreq(handles.netA);
   handles.netA.stopFreq = str2double(get(handles.eb_stopFreq,'String'));
%    handles.netA.setStopFreq(handles.netA);
   handles.netA.incrementCalc(handles.netA);
%    handles.netA.markFreq = handles.netA.startFreq;
%    handles.netA.setMarkerX(handles.netA);
```

```
%  handles.netA.angle_preset(handles.netA);
  %handles.netA.sweep_preset(handles.netA);
%  pause(4) % give network analyzer a few seconds to complete

  % Set the start, stop, and increment properties of the rotator (rot). Then set
  % the start angle of the rotator.
  handles.rot_aut.angStart = str2double(get(handles.eb_startAngle,'String'));
  handles.rot_aut.angStop = str2double(get(handles.eb_stopAngle,'String'));
  handles.rot_aut.degree = str2double(get(handles.eb_angleInc,'String'));
  handles.rot_aut.angle(handles.rot_aut);

  % Determine which way to rotate
  if handles.rot_aut.angStart < handles.rot_aut.angStop && handles.rot_aut.direct == 1 %CCW
     direction = -1;
     finalPos = (handles.rot_aut.angStop - handles.rot_aut.angStart) - 360;
  elseif handles.rot_aut.angStart < handles.rot_aut.angStop && handles.rot_aut.direct == -1
%CW
     direction = 1;
     finalPos = (handles.rot_aut.angStop - handles.rot_aut.angStart);
  elseif handles.rot_aut.angStart >= handles.rot_aut.angStop && handles.rot_aut.direct == 1
%CCW
     direction = -1;
     finalPos = -(handles.rot_aut.angStart - handles.rot_aut.angStop);
  elseif handles.rot_aut.angStart >= handles.rot_aut.angStop && handles.rot_aut.direct == -1
%CW
     direction = 1;
     finalPos = 360 + (handles.rot_aut.angStop - handles.rot_aut.angStart);
  end

  % Creates an array of the frequency points collected. This will be
  % written to the final array 'Z' which contains all relevant data.
  C = zeros(1,handles.netA.points);    % initialize array
  for m = 1:handles.netA.points
     C(m) = handles.netA.startFreq + (handles.netA.incrementFreq*(m-1));
  end

  % This is an attempt at continuous motion scanning
%    handles.rot_aut.rotate360(handles.rot_aut)
%    s=0;
%    while s ~= 360
%       d=mod(handles.rot_aut.position(handles.rot_aut),1);
%       s = handles.rot_aut.position(handles.rot_aut);
%       if d >= 0 && d <= 0.3
%          handles.netA.trigger_netA(handles.netA);
%       end
```

```
%    end

   %for n = handles.rot.angStart:handles.rot.degree*b:handles.rot.angStop
   for n = 0:handles.rot_aut.degree*direction:finalPos
      if stopflag == 1
         break
      end
      % Wait for rotator to not be in motion
      handles.rot_aut.settle(handles.rot_aut);
%        d = handles.rot_aut.position(handles.rot_aut);
%        set(handles.posText_aut,'String',d);
   % ------new data collection ------------
%        handles.netA.triggerON(handles.netA);   %Freeze the network analyzer
      if n ~= finalPos
         % Trigger the network analyzer to take a data point
         handles.netA.trigger_netA(handles.netA);
         % Increment the aut rotator position
         handles.rot_aut.increment(handles.rot_aut);
      end
      if n == finalPos
        % disp('in finalPos Done')
%           data = handles.netA.getData(handles.netA);
         %disp(data)
      end
%        handles.netA.format = 'mlog';          %Sets the format of the data
%        P = handles.netA.trace(handles.netA);   %Obtains the trace data

%        handles.netA.format = 'phas';          %Sets the format of the data
%        Q = handles.netA.trace(handles.netA);   %Obtains the trace data
%        handles.netA.triggerOFF(handles.netA);  %Un-freeze the network analyzer

      % Fill the array with data
%        x = 1;
%
%           Z(x,:,n+1) = eval(['[',P,']']);     % Magnitude
%           x = x+1;
%           Z(x,:,n+1) = eval(['[',Q,']']);     % Phase
%           x = x+1;
%           Z(x,:,n+1) = d;                       % Angle
%           x = x+1;
%           Z(x,:,n+1) = C;                       % Frequency

   end

   %handles.netA.triggerOFF(handles.netA);  %Un-freeze the network analyzer
   data = handles.netA.getData(handles.netA); %data is currently characters
```

```
   handles.netA.dataA = data;
%     disp(data);
    % disp(handles.netA.dataA);
   save('C:\temp\Test_Sweep\data\test_sweep_data.mat','data');
   % save('C:\FedEx_Antennas\data\Black_Horizontal_2588MHz_8_21_2018.mat','data'); % this
was for the fedex antennas

    % save('C:\Narda\data\Model640_10GHz_8_2_2018.mat','data');

   % fid=fopen('C:\temp\btestver4_24.txt','w');
    %fprintf(fid, data);
    %fclose(fid);

      data2 = handles.netA.formatData(handles.netA); %data is now complex numbers
      handles.netA.realData = data2;
      %disp(handles.netA.realData);
      save('C:\temp\Test_Sweep\data2\test_sweep_data2.mat','data2');
      %save('C:\FedEx_Antennas\data2\Black_Horizontal_2588MHz_8_21_2018.mat','data2');
      %save('C:\Narda\data2\Model640_10GHz_8_2_2018.mat','data2');

      %load('C:\temp\mydata2_2017_4_24.mat');
      dataDB = 20*log10(abs(data2));           %convert data2 to decibels
      dataDBshifted = circshift(dataDB, 180);   %shift data so main lobe is center
       save('C:\temp\Test_Sweep\dataDBshifted\test_sweep_dataDBshifted.mat','dataDBshifted');
     %
save('C:\FedEx_Antennas\dataDBshifted\Black_Horizontal_2588MHz_8_21_2018.mat','dataDB
shifted');
      %save('C:\Narda\dataDBshifted\Model640_10GHz_8_2_2018.mat','dataDBshifted');
      figure;
      plot(-180:1:180, dataDBshifted);
      ylabel('dB');
      dataPhs = rad2deg(angle(data2));
      dataPhsshifted = circshift(dataPhs, 180);

      save('C:\temp\Test_Sweep\dataPhsshifted\test_sweep_dataPhsshifted.mat','dataPhsshifted');

%save('C:\FedEx_Antennas\dataPHsshifted\Black_Horizontal_2588MHz_8_21_2018.mat','data
Phsshifted');
      %save('C:\Narda\dataPHsshifted\Model640_10GHz_8_2_2018.mat','dataPhsshifted');
      figure;
      plot(dataPhsshifted);

     %dataPolar = dataDB.*exp(i.*dataPhs);

    % polarplot(dataPhs, dataDB);
```

```
   % save('array_Z','Z'); old, dont use
%     set(handles.text1,'String','Done. Writing to file...');

   %Return to zero position
%     handles.rot_aut.spd = 2000;
%     handles.rot_aut.speed(handles.rot_aut);
%     handles.rot_aut.degree = 0;
%     handles.rot_aut.angle(handles.rot_aut);

%     global f;
%     fileName = inputdlg('Please enter the name for your data.');
%     str = char(strcat(fileName,'.csv'));
%     f = char(str);
   % ---old---
%     header = ['Magn,Phase,Angle,Freq'];
%     fid = fopen(f,'w+');
%     fprintf(fid,'%s',header);
%     fclose(fid);
%     dlmwrite(f,Z,'roffset',1,'-append');
   % -----old end-----
%     csvwrite(f,Z);
%     start(handles.tmr);
%     set(handles.text1,'String','Finished!');

   %Return to zero position
%     handles.rot_aut.spd = 2000;
%     handles.rot_aut.speed(handles.rot_aut);
%     handles.rot_aut.degree = 0;
%     handles.rot_aut.angle(handles.rot_aut);

   guidata(hObject, handles);
end

% Stop the rotator.
function Stop_Callback(hObject, eventdata, handles)
   global stopflag;
   stopflag = 1;
%     handles.rot_aut.emergencyStop(handles.rot_aut);
%     handles.rot_source.emergencyStop(handles.rot_source);
end
function Sweep_Callback(hObject, eventdata, handles)%blg
   handles = guidata(gcbo);
   global stopflag;
   stopflag = 0;
   stop(handles.tmr);
```

```
    % Prepare the network analyzer for a scan
    pause(4) % give network analyzer a few seconds to complete

    % Set the start, stop, and increment properties of the network analyzer (netA).
    % Then set the start and stop frequency of the netA.
    handles.netA.startFreq = str2double(get(handles.eb_startFreq,'String'));
   % handles.netA.setStartFreq(handles.netA);
    handles.netA.stopFreq = str2double(get(handles.eb_stopFreq,'String'));
   % handles.netA.setStopFreq(handles.netA);
    handles.netA.incrementCalc(handles.netA);
    handles.netA.sweep_preset(handles.netA); %sweeps range of frequencies
    pause(4) % give network analyzer a few seconds to complete
%     handles.netA.markFreq = handles.netA.startFreq;
%     handles.netA.setMarkerX(handles.netA);

    % Set the start, stop, and increment properties of the rotator (rot). Then set
    % the start angle of the rotator.
    handles.rot_aut.angStart = str2double(get(handles.eb_startAngle,'String'));
    handles.rot_aut.angStop = str2double(get(handles.eb_stopAngle,'String'));
    handles.rot_aut.degree = str2double(get(handles.eb_angleInc,'String'));
    handles.rot_aut.angle(handles.rot_aut);

    % Determine which way to rotate
    if handles.rot_aut.angStart < handles.rot_aut.angStop && handles.rot_aut.direct == 1 %CCW
        direction = -1;
        finalPos = (handles.rot_aut.angStop - handles.rot_aut.angStart) - 360;
    elseif handles.rot_aut.angStart < handles.rot_aut.angStop && handles.rot_aut.direct == -1
%CW
        direction = 1;
        finalPos = (handles.rot_aut.angStop - handles.rot_aut.angStart);
    elseif handles.rot_aut.angStart >= handles.rot_aut.angStop && handles.rot_aut.direct == 1
%CCW
        direction = -1;
        finalPos = -(handles.rot_aut.angStart - handles.rot_aut.angStop);
    elseif handles.rot_aut.angStart >= handles.rot_aut.angStop && handles.rot_aut.direct == -1
%CW
        direction = 1;
        finalPos = 360 + (handles.rot_aut.angStop - handles.rot_aut.angStart);
    end

    % Creates an array of the frequency points collected. This will be
    % written to the final array 'Z' which contains all relevant data.
    C = zeros(1,handles.netA.points);    % initialize array
    for m = 1:handles.netA.points
        C(m) = handles.netA.startFreq + (handles.netA.incrementFreq*(m-1));
    end
```

```
        C = C';
        save('C:\temp\sweep_data\C.mat','C');
    Z=double.empty;
    for n = 0:handles.rot_aut.degree*direction:finalPos
        disp(n)
        if stopflag == 1
            break
        end
        % Wait for rotator to not be in motion
        handles.rot_aut.settle(handles.rot_aut);
%         d = handles.rot_aut.position(handles.rot_aut);
%         set(handles.posText_aut,'String',d);
    % ------new data collection ------------
        if n ~= finalPos
            % Trigger the network analyzer to take a data point
            handles.netA.trigger_netA(handles.netA);
            % Increment the aut rotator position
            handles.rot_aut.increment(handles.rot_aut);

            data = handles.netA.getData(handles.netA);%puts frequency data in array
            save('C:\temp\sweep_data\data.mat','data');
            handles.netA.dataA = data;
            data2 = handles.netA.formatData(handles.netA); %data is now complex numbers
            save('C:\temp\sweep_data\data2.mat','data2');
            handles.netA.realData = data2;
            % Fill the array with sweep_data
            if isempty(Z)
                Z = horzcat(C,data2)
            else
                Z = horzcat(Z,data2);
            end
            data2 = double.empty;

        end




    end

    %handles.netA.triggerOFF(handles.netA);  %Un-freeze the network analyzer
    %data = handles.netA.getData(handles.netA); %data is currently characters
    %handles.netA.dataA = data;

    save('C:\temp\sweep_data\sweep_data.mat','Z');
    % save('C:\Narda\data\Model640_10GHz_8_2_2018.mat','data');
```

```
   % fid=fopen('C:\temp\btestver4_24.txt','w');
    %fprintf(fid, data);
    %fclose(fid);

%     data2 = handles.netA.formatData(handles.netA); %data is now complex numbers
%     handles.netA.realData = data2;
%     %disp(handles.netA.realData);
%     save('C:\FedEx_Antennas\data2\Black_Horizontal_2588MHz_8_21_2018.mat','data2');
%     %save('C:\Narda\data2\Model640_10GHz_8_2_2018.mat','data2');
%
%     %load('C:\temp\mydata2_2017_4_24.mat');
%     dataDB = 20*log10(abs(data2));           %convert data2 to decibels
%     dataDBshifted = circshift(dataDB, 180);   %shift data so main lobe is center
%
save('C:\FedEx_Antennas\dataDBshifted\Black_Horizontal_2588MHz_8_21_2018.mat','dataDB
shifted');
%     %save('C:\Narda\dataDBshifted\Model640_10GHz_8_2_2018.mat','dataDBshifted');
%     figure;
%     plot(-180:1:180, dataDBshifted);
%     ylabel('dB');
%     dataPhs = rad2deg(angle(data2));
%     dataPhsshifted = circshift(dataPhs, 180);

     %blg
save('C:\FedEx_Antennas\dataPHsshifted\Black_Horizontal_2588MHz_8_21_2018.mat','dataPh
sshifted');
     %save('C:\Narda\dataPHsshifted\Model640_10GHz_8_2_2018.mat','dataPhsshifted');
     %blg figure;
     %blg plot(dataPhsshifted);

    %dataPolar = dataDB.*exp(i.*dataPhs);

   % polarplot(dataPhs, dataDB);


%     set(handles.text1,'String','Done. Writing to file...');

   guidata(hObject, handles);
end
```

APPENDIX C

```
% Rotator.m
% Corey Garner
% 5/9/2013
% This class will define a 'rotator' object.


classdef rotator < handle
   properties
      serialPort;
      direct;
      degree;
      angStart;
      angStop;
      commPort; %allows COM selection. future use
      rotateSpeed;
      initSteps;
   end

   methods(Static)
      % constructor
      function rota = rotator(commPort,initSteps)
         rota.commPort = commPort;
         rota.serialPort = serial(rota.commPort);
         rota.direct = -1;
         rota.degree = 1;
         rota.rotateSpeed = 1000;
         rota.angStart = 0;
         rota.angStop = 0;
         rota.initSteps = initSteps;
      end

      % Opens the serial port of the rotator on COM3
      function openPort(rota)
         set(rota.serialPort,'Terminator','CR');
         fopen(rota.serialPort);
      end

      % Initializes the rotator to the 0 degree mark
      function initialize(rota)
         fprintf(rota.serialPort, 'F,C,S1M1000,I1M-0,I1M4000,I1M-0,I1M-%i,IA1M-
0,R',rota.initSteps);
      end
      % Set the current position as 0
      function zeroRef(rota)
```

```
   fprintf(rota.serialPort, 'F,C,IA1M-0,R');
end

% Increments the rotator a user defined number of degrees.
% The property 'direct' determines the direction of rotation,
% where -1 = CW, 1 = CCW.
function increment(rota)
   fprintf(rota.serialPort, 'F,C,I1M%d,R', rota.degree*rota.direct*80);
end

% Returns the current position of the rotator in steps.
function p = position(rota)
   fprintf(rota.serialPort,'X');
   % format position data
   f = strrep(fgetl(rota.serialPort),'^',');   %removes ^ from return string value
   p = str2double(strrep(f,'+',''))/80; % removes + from return string
   if p < 0
      p = p*-1;
   elseif p > 0
      p = 360 - str2double(strrep(f,'+',''))/80;
   else
      p = 0;
   end
end

% Determines if the rotator is moving
function settle(rota)
   % allow the rotator to settle
   p1 = 1;
   p2 = 2;
   while p1 ~= p2
      p1 = rota.position(rota);
      pause(0.1)
      p2 = rota.position(rota);
   end
end

% Sets the rotator to the desired angle.
function angle(rota)
   fprintf(rota.serialPort, 'F,C,IA1M%d,R', (80.*rota.angStart.*-1));
end

% stop angle: Necessary function?
function stpAngle(rota)
   fprintf(rota.serialPort, 'F,C,I1M%i,R', 80*rota.angStop.*rota.direct);
end
```

```
        % Sets the rotator speed
        function speed(rota)
            fprintf(rota.serialPort, 'F,C,S1M%i,R', rota.rotateSpeed);
        end

        % Immediately stops the rotator's rotation
        function emergencyStop(rota)
            fprintf(rota.serialPort,'K');
        end

        % Rotates the rotator 360 degrees
        function rotate360(rota)
            fprintf(rota.serialPort,'F,C,I1M%i,R', rota.direct.*28800);
        end

        % Closes the serial port of the rotator on COM3
        function closePort(rota)
            fclose(rota.serialPort);
%           delete(rota.serialPort);
        end
    end
end
```

APPENDIX D:

```
%Post_Processing
%6/19/2019
%Thomas Garner
%This code is for post processing sweep data collected by the HP 8530A
%and applying correction data. This can do averaging for a single frequency
%or select a single frequency over a wide range of frequencies
%To use this code, look at the row you want to extract and change the row
%variable to the row number. NOTE I DO NOT GRAB THE FREQUENCY WHEN
%EVALUATING THE DATA. IT IS UP TO YOU TO KNOW WHICH FREQUENCY ROW
YOU GRAB
clc;
clear;

%Load sweep data
[filename,path] = uigetfile('*.mat');
filename = fullfile(path,filename);

%Import sweep data
data = importdata(filename);

%Select row that has your desired frequency and angle_inc used
row = 1;
angle_inc = 1;

data2 = data(row,[2:end]); %contains data for the row you selecting. does not have frequency
data_avg = mean(data(:,[2:end])); %used when averaging data
freq = data(row,1); %grabs frequency

%Used when selecting a frequency out of a range of frequencies
dataDB = 20*log10(abs(data2)); %convert data2 to decibels with corrections
dataDBshifted = circshift(dataDB, 180);   %shift data so main lobe is center

%Used when average a single frequency from sweeping a singlle frequency
dataDB_avg = 20*log10(abs(data_avg));%covert data_avg to decibels with correction
dataDBshifted_avg = circshift(dataDB_avg, 180);   %shift data_avg so main lobe is center

%Uncomment if selecting a specific frequency and not averaging
%{
%Plotting a single frequency pattern
figure;
plot(-180:angle_inc:180, dataDBshifted);
ylabel('dB');
xlabel('angle');
title(['Measured Frequency is ',num2str(freq),' MHz']);
```

```
%Plotting a single frequency phase
dataPhs = rad2deg(angle(data2));
dataPhsshifted = circshift(dataPhs, 180);
figure;
plot(-180:angle_inc:180, dataPhsshifted);
xlabel('angle');
title(['Measured Frequency is ',num2str(freq),' MHz']);
%}


%Uncomment when averaging a sweep of a single frequency
%Plotting pattern with averaged data
figure;
plot(-180:angle_inc:180, dataDBshifted_avg);
ylabel('dB');
xlabel('angle');
title(['Measured Frequency is ',num2str(freq),' MHz']);

%Plotting phase with averaged data
dataPhs_avg = rad2deg(angle(data_avg));
dataPhsshifted_avg = circshift(dataPhs_avg, 180);
figure;
plot(-180:angle_inc:180, dataPhsshifted_avg);
```

APPENDIX E

```
%Post_Processing_Calibrated.m
%6/19/2019
%Thomas Garner
%This code is for post processing sweep data collected by the HP 8530A
%and applying correction data. This can do averaging for a single frequency
%or select a single frequency over a wide range of frequencies
%To use this code, look at the row you want to extract and change the row
%variable to the row number. NOTE I DO NOT GRAB THE FREQUENCY WHEN
%EVALUATING THE DATA. IT IS UP TO YOU TO KNOW WHICH FREQUENCY ROW
YOU GRAB
clc;
clear;

%Load cal file
[filename,path] = uigetfile('*.txt');
filename = fullfile(path,filename);

%Number of points used in measurement
points = 51;
delimiterIn = ',';
headerLines = 32 + points.*2;

%Concatenate frequencies to cal set
headerLines2 = 26;

% Import cal set file
cal_data = importdata(filename, delimiterIn, headerLines);%get correction data
cal_freq = importdata(filename, delimiterIn, headerLines2);%get frequencies
complexData = complex(cal_data.data(:,1),cal_data.data(:,2));%convert correction data to
complex

%Load sweep data
[filename2,path2] = uigetfile('*.mat');
filename2 = fullfile(path2,filename2);

%Concatenate frequencies to correction data
complexData = horzcat(cal_freq.data,complexData);

%Import sweep data
data = importdata(filename2);

%Select row that has your desired frequency and angle_inc used
row = 1;
angle_inc = 1;
```

```
data2 = data(row,[2:end])./complexData(row,2); %contains data for the row you selecting. does not have frequency
data_avg = mean(data(:,[2:end]))./complexData(row,2); %used when averaging data
freq = data(row,1); %grabs frequency

%Used when selecting a frequency out of a range of frequencies
dataDB = 20*log10(abs(data2)); %convert data2 to decibels with corrections
dataDBshifted = circshift(dataDB, 180);   %shift data so main lobe is center

%Used when average a single frequency from sweeping a singlle frequency
dataDB_avg = 20*log10(abs(data_avg));%covert data_avg to decibels with correction
dataDBshifted_avg = circshift(dataDB_avg, 180);   %shift data_avg so main lobe is center

%Uncomment if selecting a specific frequency and not averaging
%{
%Plotting a single frequency pattern
figure;
plot(-180:angle_inc:180, dataDBshifted);
ylabel('dB');
xlabel('angle');
title(['Measured Frequency is ',num2str(freq),' MHz']);

%Plotting a single frequency phase
dataPhs = rad2deg(angle(data2));
dataPhsshifted = circshift(dataPhs, 180);
figure;
plot(-180:angle_inc:180, dataPhsshifted);
%}

%Uncomment when averaging a sweep of a single frequency
%Plotting pattern with averaged data
figure;
plot(-180:angle_inc:180, dataDBshifted_avg);
ylabel('dB');
ylim([-40 15]);
xlabel('angle');
title(['Measured Frequency is ',num2str(freq),' MHz']);

%Plotting phase with averaged data
dataPhs_avg = rad2deg(angle(data_avg));
dataPhsshifted_avg = circshift(dataPhs_avg, 180);
figure;
plot(-180:angle_inc:180, dataPhsshifted_avg);
```

APPENDIX F

Example Antenna Definition:

CITIFILE A.01.01
#NA VERSION HP8530A.01.64
NAME ANTENNA_DEF
#NA DEF_LABEL Narda640
#NA STANDARD 1
#NA STANDARD_LABEL Narda640
VAR FREQ MAG 51
DATA GAIN[1] DB
SEG_LIST_BEGIN
SEG 8200000000 12400000000 51
SEG_LIST_END
BEGIN
1.475E1
1.481E1
1.488E1
1.494E1
1.501E1
1.507E1
1.513E1
1.520E1
1.526E1
1.533E1
1.539E1
1.546E1
1.552E1
1.558E1
1.565E1
1.571E1
1.578E1
1.584E1
1.591E1
1.597E1
1.603E1
1.610E1
1.616E1
1.623E1
1.629E1
1.635E1
1.642E1
1.648E1
1.655E1
1.661E1

1.668E1
1.674E1
1.680E1
1.687E1
1.693E1
1.700E1
1.706E1
1.713E1
1.719E1
1.725E1
1.732E1
1.738E1
1.745E1
1.751E1
1.758E1
1.764E1
1.770E1
1.777E1
1.783E1
1.790E1
1.796E1
END

APPENDIX G

Cal set generated by network analyzer using above antenna definition:

CITIFILE A.01.01
#NA VERSION HP8530A.01.64
#NA TITLE
NAME CAL_SET
#NA REGISTER 5
VAR FREQ MAG 51
DATA E[1] RI
DATA E[2] RI
#NA SWEEP_TIME 9.999994E-2
#NA POWER1 1.0E1
#NA POWER2 1.0E1
#NA PARAMS 4
#NA CAL_TYPE 6
#NA POWER_SLOPE 0.0E0
#NA POWER_SLOPE2 0.0E0
#NA SLOPE_MODE 0
#NA SLOPE_MODE2 0
#NA TRIM_SWEEP 0
#NA SWEEP_MODE 4
#NA LOWPASS_FLAG -1
#NA FREQ_INFO 1
#NA IF_BW 10000
#NA SPAN 8200000000 12400000000 51
#NA DUPLICATES 0
#NA ARB_SEG 8200000000 12400000000 51
VAR_LIST_BEGIN
8200000000
8284000000
8368000000
8452000000
8536000000
8620000000
8704000000
8788000000
8872000000
8956000000
9040000000
9124000000
9208000000
9292000000
9376000000
9460000000

9544000000
9628000000
9712000000
9796000000
9880000000
9964000000
10048000000
10132000000
10216000000
10300000000
10384000000
10468000000
10552000000
10636000000
10720000000
10804000000
10888000000
10972000000
11056000000
11140000000
11224000000
11308000000
11392000000
11476000000
11560000000
11644000000
11728000000
11812000000
11896000000
11980000000
12064000000
12148000000
12232000000
12316000000
12400000000
VAR_LIST_END
COMMENT YEAR MONTH DAY HOUR MINUTE SECONDS
CONSTANT TIME 2019 06 16 12 53 23.0
BEGIN
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0

0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
0.0E0,0.0E0
END
BEGIN

2.20954E-4,4.92245E-4
1.64687E-4,-5.92738E-4
-0.33944E-4,5.94973E-4
-1.88380E-4,-5.02586E-4
3.93375E-4,3.89337E-4
-5.36680E-4,-2.38001E-4
5.32239E-4,0.92715E-4
-4.44948E-4,1.23351E-4
3.47837E-4,-3.64571E-4
-2.12222E-4,5.13255E-4
0.48235E-4,-4.78610E-4
1.18866E-4,4.31239E-4
-2.76997E-4,-3.85195E-4
3.83943E-4,2.73942E-4
-4.00915E-4,-1.23485E-4
3.93390E-4,-0.84444E-4
-3.66598E-4,2.49996E-4
2.91243E-4,-2.58088E-4
-1.47342E-4,3.06308E-4
-0.33408E-4,-3.54111E-4
1.41501E-4,3.10868E-4
-2.17586E-4,-2.76476E-4
2.51263E-4,1.69798E-4
-3.24264E-4,-0.45210E-4
3.14146E-4,-0.30860E-4
-2.38537E-4,1.49488E-4
1.43788E-4,-2.06947E-4
-0.90599E-4,2.47627E-4
-0.24393E-4,-2.79754E-4
0.81285E-4,2.11462E-4
-1.48192E-4,-1.73360E-4
1.87575E-4,1.02065E-4
-1.69254E-4,-0.48607E-4
1.81354E-4,-0.22463E-4
-1.1589E-4,1.09013E-4
1.08726E-4,-1.23389E-4
-4.31612E-5,1.16478E-4
-0.01639E-4,-1.43334E-4
5.88372E-5,1.11881E-4
-8.02315E-5,-1.05500E-4
9.56244E-5,3.59937E-5
-1.04911E-4,3.07261E-5
6.29983E-5,-5.66095E-5
-5.02876E-5,6.44326E-5
0.94566E-5,-5.25359E-5
1.38934E-5,5.94817E-5

-3.44347E-5,-2.12583E-5
3.34586E-5,1.81999E-5
5.27547E-6,-9.20007E-6
1.29789E-5,-1.9948E-5
1.20671E-5,-1.46646E-5
END

APPENDIX H

As shown from reference [9, pp.53]:

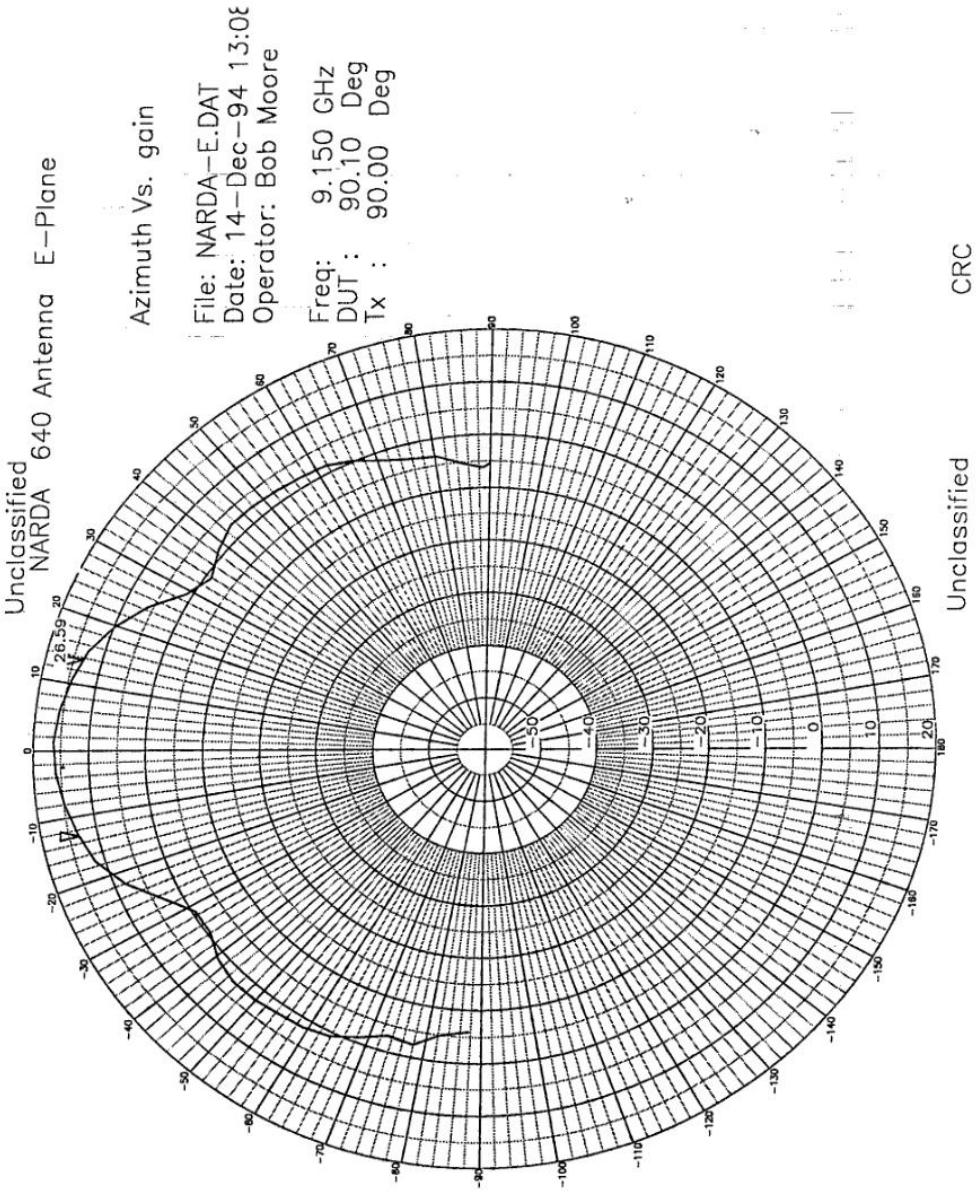

Figure 29
Narda Model 640 Standard Gain Horn E-Plane Gain vs Azimuth

53

VITA

Thomas Salvatore Garner

Personal Information Excluded

Education

B.S. in Electrical Engineering, *summa cum* laude, **University of Mississippi,** May 2017
GPA: 3.92

Relevant Course Work

- Advance Digital Systems Design
- Digital CMOS VLSI Design
- Microprocessor Systems Engineering
- Electric Circuit Theory
- Linear Systems
- Random Signals
- Theory of Control Systems
- Modulation, Noise, and Communications
- Theory of Fields
- Microwave Engineering
- Antenna Theory and Design

Awards, Fellowships, & Grants

IMAPS Huntsville, AL 2017 Additive 2017 Best Poster Award

Eta Kappa Nu Outstanding Student Award (University of Mississippi 2017)

MS Electric Power Association Scholarship (University of Mississippi 2016)

Eagle Scout Scholarship (University of Mississippi 2012)

High School Valedictorian/Salutatorian Scholarship (Valedictorian of Potts Camp High School, University of Mississippi 2012)

Technical Tools and Skills

Advance Design System (ADS), CST, Matlab, Antenna Design, C++, VHDL, Xilinx, VLSI CMOS, VLSI CAD Tools, Digital Design, Computer Architecture, and Assembly Language.

Student Research

May 2016 – Present: 3D printing and design work for antennas at WiFi frequencies. Antennas are currently tested using a network analyzer.

Future research will include using a network analyzer with an anechoic chamber to test the 3D printed and milled antennas. This will also involve adding features to existing code.

Publications

J. C. Booth, T. Garner, E. Hutchcraft, R. Gordon, E. Lackey, M. Whitely, M. Kranz and C. Rudd, "A 3D Printed Sierpinski Patch Antenna," U.S. Army RDECOM, 2017.

Previous Work

Raytheon SAS – Summer Intern (2018)

Professional Affiliations

IEEE Eta Kappa Nu - Epsilon Omega Chapter President (2016)