SPARSE CANTOR RING ANTENNA ARRAYS WITH NON-UNIFORM ELEMENT SPACING

Frederick U. Diaz, University of Pennsylvania (Electrical Engineering)
NSF-AMP Undergraduate Research Program
Advisors: Dr. Dwight L. Jaggard and Aaron D. Jaggard

ABSTRACT

This research project is based on previous work done by Dwight L. Jaggard and Aaron D. Jaggard on fractal ring arrays. We utilize fractals in the fabrication of a class of Cantor ring arrays. Fractal descriptors such as dimension, stage of growth, and lacunarity are applied in designing and characterizing these Cantor rings. Using the continuous arrays developed by Jaggard and Jaggard as a point of reference, we examine the design of analogous discrete arrays. We examine various ways of periodically and randomly thinning and building up these arrays azimuthally. The arrays are compared to their periodic and random counterparts as well as the continuous case to evaluate performance. Mainbeam quality, sidelobe level, and visible range are used to rate the arrays. The Cantor ring arrays and the corresponding array factors are simulated using programs developed in MATLAB. Our goals include developing fractal arrays that have low sidelobes comparable to those of periodic arrays while maintaining the robustness of random arrays. We aim for superior fractal performance at an equal number of elements as the periodic and random cases, as well as comparable performance using fewer elements.

1. INTRODUCTION

1.1 Antenna Arrays

While single antenna elements radiate signals in a broad distribution pattern, antenna arrays manipulate and concentrate signals in a given direction. The radiation pattern of an array is the cumulative effect of constructive and deconstructive interference among the individual antenna elements. The result is usually a single mainbeam, the region of greatest cumulative constructive interference, surrounded by numerous side beams or sidelobes of varying degrees of constructive and deconstructive interference. An example of the far field radiation of an array, calculated as the Array Factor, is given in Figure 1. An *ideal* array would result in a radiated field that looks like an upside down "T." The mainbeam would be a single untapered, undistorted beam. The energy distribution among the sidelobes would be even. High sidelobes are brought down while low sidelobes are brought up. The radiation pattern resulting from an antenna array with non-uniform distribution is called point-to-point or preferred coverage radiation. A single antenna element would be used in broadcasting applications such as television networks or radio stations where it is favorable to have a signal with as much coverage as possible. Point-to-point signals are desirable for scanning applications such as those used in air

traffic control systems or medical imaging devices. In these systems the ability to distinguish one point from another is vital. Point-to-point signals are also desired when the information being sent is private. The radiation characteristics of an antenna or an array are the same when sending or receiving a signal.

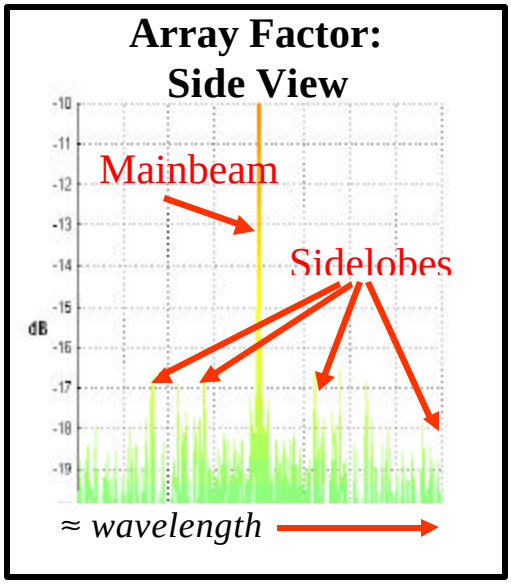

Figure 1: Arrays produce radiation patterns with a mainbeam and many sidelobes.

1.2 Fractal Electromagnetics

Antenna arrays are usually created by distributing elements over a given area, either randomly or along a periodic lattice. Other arrays are periodic arrays that are periodically randomly thinned. Periodic arrays are characterized by low sidelobes and narrow visible range. Visible range refers to the maximum wavelength and corresponding minimum frequency at which a given array operates. Random arrays are characterized by high sidelobes and robustness in terms of element failure and element positioning.

Recently, fractal geometry has been incorporated into the design of arrays as part of the research area denoted *fractal electrodynamics*. The motivation behind fractal arrays is to combine the low sidelobes characteristic of periodic arrays with the robustness characteristic of random arrays. This goal is pursued through random fractal patterns.

1.3 Jaggard and Jaggard's Fractal Ring Arrays

This research project follows up on prior work done on fractal ring arrays [1], cantor ring arrays [2], and cantor ring diffractals [3]. Specifically, this project continues the study of discrete Cantor ring arrays. We use their findings on the continuous infinitesimal width Cantor ring array derived from the Cantor set with the following

fractal descriptors: dimension = 9/10, stage = 2, number of gaps = 3, lacunarity = 37/1000. This continuous array serves as a point of reference and departure. We examine the effects of the number and arrangement of antenna elements on the array's radiated field. We explore different ways of thinning the arrays and evaluating their performance. The desired qualities are a round main beam with little or no degradation, low sidelobes, a large visible range, and coherence with the continuous case.

2. FRACTALS AND FRACTAL DESCRIPTORS

2.1 Background

While Euclidean geometry is over two thousand years old, fractal geometry is relatively new. In the mid-1970s, B.B. Mandelbrot introduced and coined the term *fractal* to describe structures often referred to as spiky, variegated, or ramified. Such irregular structures are often difficult to describe using traditional Euclidean objects. The term fractal comes from the Latin *fractus* which means broken and refers to irregular fragments. Fractals make it easy to describe complex objects such as galaxies, stock market fluctuations, and tree branching. In the words of Mandelbrot, "Clouds are not spheres, mountains are not cones, coastlines are not circles and bark is not smooth, nor does lightning travel in a straight line" [4].

A loose definition of a fractal is an object whose parts are similar to the whole in some way. This similarity can be exact and mathematical, or it can be statistical and similar over the mean. Fractals are invariant under change of scale and invariant under displacement [5]. This property, referred to as self-similarity, is the calling card of fractals. Figure 2 provides an example of self-similarity. As we repeatedly magnify the curve, we find that each subsection is composed of increasingly fine structures. This rescaled jaggedness is similar, at least in the mean, to previous and subsequent stages of magnification.

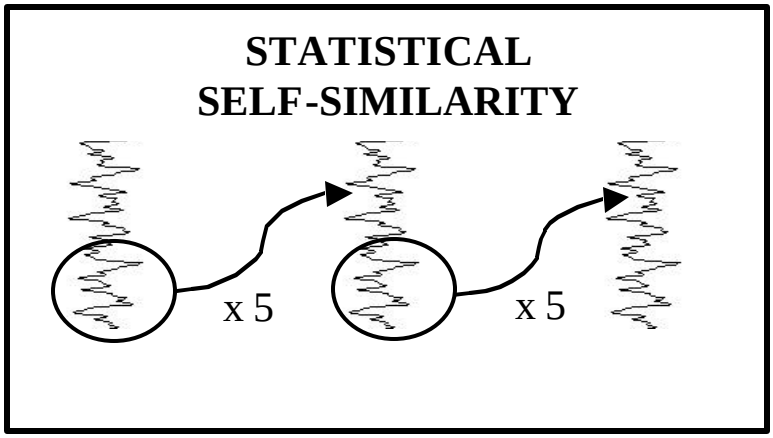

Figure 2: Magnifying this fractal curve repeatedly reveals increasingly fine structure. Each magnified portion is similar to the original structure, at least in the mean.[5]
Fractals exhibit structure on all scales. Their self-similarity suggests that they can be generated through iterative or recursive means. Figure 3 shows the formation of a

mathematical fractal known as the Triadic Cantor bar, or middle thirds Cantor bar. Performing repeated excising operations creates this fractal. We begin with the unit interval, given nonzero height to aid in visualization. Taking away the middle third from the unit interval results in the two intervals of the second row. Removing the middle thirds of the remaining two intervals results in the bars of the third row. Continuing this process an infinite number of times gives us the fractal Cantor set. In generating this fractal, each iteration represents a stage of growth. We denote the initial interval as the Stage 0 Cantor Bar. Practical fractals are self-similar over a limited range of magnifications. These bound fractals are more appropriately referred to as *prefractals*. However we shall still refer to them and their characteristics as fractal.

2.2 Fractal Dimension

From the above discussion and figure, its apparent that our normal concept of dimension needs to be revisited and redefined to accommodate fractals. It seems as though our fractal structures take up more space than their Euclidean allocation. A measure of the roughness and fragmentation of fractals is defined as the *fractal dimension*, denoted here as *D*. Unlike Euclidean dimension *d*, *D* is not restricted to integer values.

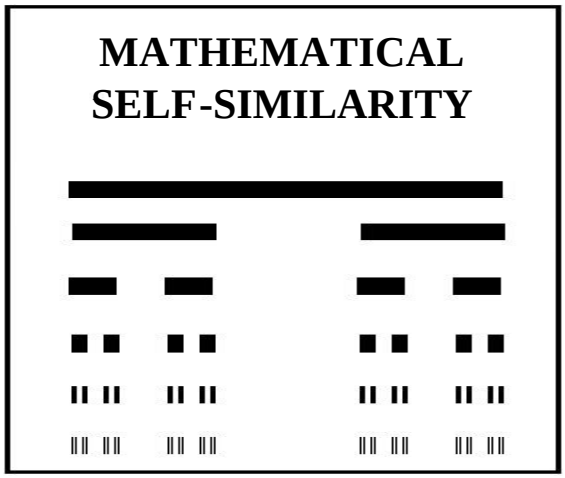

Figure 3 [5]: The Triadic Cantor bar is generated by performing repeated excising operations on existing intervals.

Our concept of dimension and its extension to fractals can be understood by the following heuristic argument [5]. Consider the measurement of the line segment with length *L* shown at the top of Figure 4. If we use a one-dimensional yardstick of length *å*, the total number *N* of yardsticks contained in the line segment is simply

$$N(\varepsilon) = \left(\frac{L}{\varepsilon}\right)^{1}. \tag{1}$$

Similarly, consider the measurement of the area A (= L^2) of the square shown in the middle of Figure 4. If we use a two-dimensional square yardstick of side ε, the total number N of yardsticks contained in the square is

$$N(\varepsilon) = \left(\frac{L}{\varepsilon}\right)^2. \tag{2}$$

Likewise, consider the measurement of the volume V (= L^3) of the cube shown at the bottom of Figure 4. If we use a three-dimensional cubical yardstick of side ε, the total number N of yardsticks contained in the cube is

$$N(\varepsilon) = \left(\frac{L}{\varepsilon}\right)^3. \tag{3}$$

In each of the relations (1)-(3), the exponent represents the Euclidean dimension d of the object under consideration.

This concept of dimension can be expanded by defining a generalized fractal dimension D through the relation of yardstick size ε and number of yardsticks N contained in an arbitrary object as

$$N(\varepsilon) \equiv C\varepsilon^{-D}, \tag{4}$$

where the appropriate constant C is dependent on the Euclidean dimension in which the object is embedded. D becomes d for simple Euclidean objects where the dimension is an integer. Rearranging the expression in (4) yields the desired dimension directly as

$$D \equiv \frac{d[\ln N(\varepsilon)]}{d[\ln(1/\varepsilon)]} \xrightarrow[D \;\; CONSTANT]{} \frac{[\ln N(\varepsilon)] - [\ln C]}{[\ln(1/\varepsilon)]} \xrightarrow[\lim \varepsilon \to 0]{} \frac{[\ln N(\varepsilon)]}{[\ln(1/\varepsilon)]} \tag{5}$$

The fractal dimension definition of (5), which is only one of many fractal dimensions defined depending on application, is the basis for the disk covering method or the box counting method. This dimension is known as the similarity dimension.

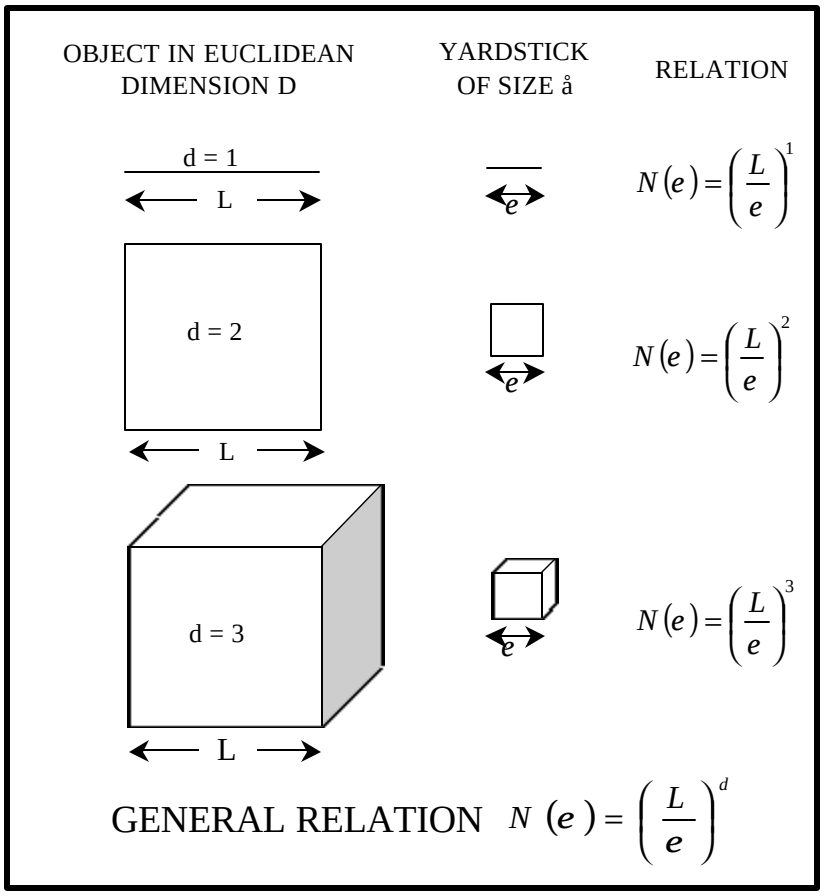

Figure 4: Use of yardstick to obtain the dimension of Euclidean objects. Here *N* is the number of yardstick lengths å in each object of side *L*.

At each stage of growth in the construction of a fractal, the dimension can be calculated by using the appropriately scaled yardstick. Lets consider the Stage 1 triadic Cantor bar (second row) of Figure 3. The length *L* of the interval, which remains the same at every stage, is 1. *N* is equal to the number of bars, while å is equal to the size of each bar relative to the size of the entire interval, 2 and 1/3 respectively. Using (5), we obtain

$$D=\frac{\ln(2)}{\ln(3)}=0.6309$$

Applying (5) to Stages 2 and 3, with å = 1/9 and 1/27 respectively, gives us

$$\textit{Stage}\quad 2\quad D=\frac{\ln(4)}{\ln(9)}=\frac{\ln\left(2^2\right)}{\ln\left(3^2\right)}=\frac{\ln(2)}{\ln(3)}=0.6309$$

$$\textit{Stage}\quad 3\quad D=\frac{\ln(8)}{\ln(27)}=\frac{\ln\left(2^3\right)}{\ln\left(3^3\right)}=D=\frac{\ln(2)}{\ln(3)}=0.6309$$

2.3 Lacunarity

Along with dimension and stage of growth, another important fractal descriptor is lacunarity. Mandelbrot defines lacunarity as a measure of the gappiness of a fractal [4]. Within a class of fractals, those with large gaps are described as having high lacunarity. Fractals whose gaps are more homogeneous throughout the structure, with less variation in gap size, are considered to have low lacunarity. In an illustration of a fractal, dimension may be regarded as the amount of ink used, while lacunarity can be considered in terms of the ink distribution over the area. Figure 5 [1] shows three Cantor bars with the same dimension, but different lacunarities. The top bar is relatively more homogeneous than the other two, while the bottom bar has the greatest difference in gap size. Each bar represents the Stage 2 case of a three-gap Cantor bar.

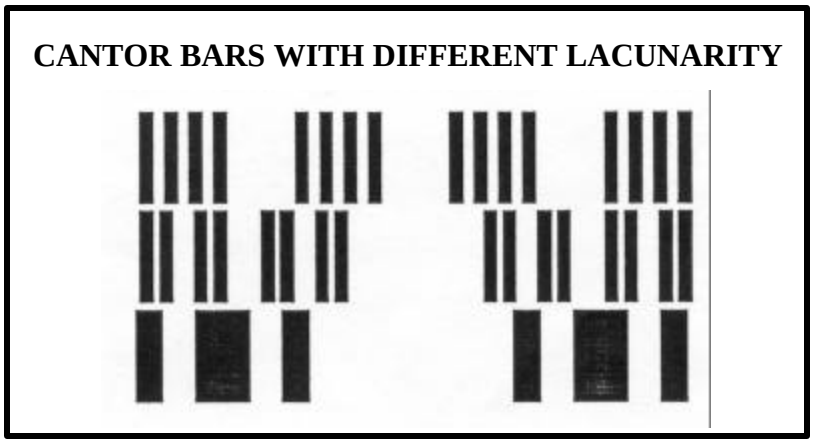

Figure 5 [1]: Three-gap Cantor bars at the second stage of growth with low, medium, and high lacunarity (top to bottom).

3. PROBLEM STATEMENT AND SOLUTION

3.1 Overview and Goals

The prime motivation for this project is to continue the work done by Jaggard and Jaggard on Cantor ring arrays [1]. Jaggard and Jaggard analyzed various classes of continuous Cantor rings using values of dimension, stage of growth, lacunarity and number of gaps that they found to be “optimal” for array performance. They also explored the discrete case where elements were dispersed evenly along each ring with a linear density that was held constant for each ring.

Our research further explores discrete arrays. The continuous arrays serve as both a point of departure and a point of reference, exhibiting the “best” possible performance we can expect from our discrete arrays as long as elements are restricted to the aforementioned Cantor rings.

We explore ways of thinning the arrays from the continuous case as well as methods of building up arrays element by element. The processes of thinning and building up involve periodic and random means, as well as a combination of the two. By spacing the arrays in aperiodic ways, we expect to obtain better performance than the periodic case in at least one of two ways: similar performance with fewer elements, or better performance with the same number of elements. When comparing arrays, we examine mainbeam quality, sidelobe level, and visible range. Seemingly favorable arrays are compared not only to the evenly spaced arrays examined by Jaggard and Jaggard, but also to traditional periodic and random arrays that are analogous in total number of elements and area of element distribution.

3.2 Preparations

Prior to beginning the analysis of antenna arrays and radiation patterns, I spent five weeks learning about fractals, antennas, array factors, and MATLAB programming. I read Mandelbrot's *Fractal Geometry of Nature* to gain an understanding and an intuitive feel for fractals and their properties. Weeks' *Antenna Engineering* provided much of the background and mathematical theory on array factors. During and after reading these and many other articles and book chapters, I worked on writing MATLAB programs that generated Cantor rings, plotted antenna elements, and produced different perspectives of the resulting radiated field. My reading and programming were supplemented and clarified through meetings with Jaggard and Jaggard.

3.3 Construction of Cantor Ring Arrays

As shown in Figure 6 [1], Cantor ring arrays are obtained from analogous Cantor bars. We begin by generating a subset of the Cantor set *C*, in this case Stage 2 of the three-gap Cantor bar of Figure 6a. The Cantor bar is rotated about its midpoint, forming an aperture array whose annuli (rings) have widths corresponding to the Cantor set, Figure 6b. Thinning the rings in a manner that results in infinitesimal width rings at the center of each finite width annuli gives Figure 6c. These are further thinned azimuthally. We replace each continuous ring by discrete radiating elements, forming the desired discrete Cantor ring array of Figure 6d.

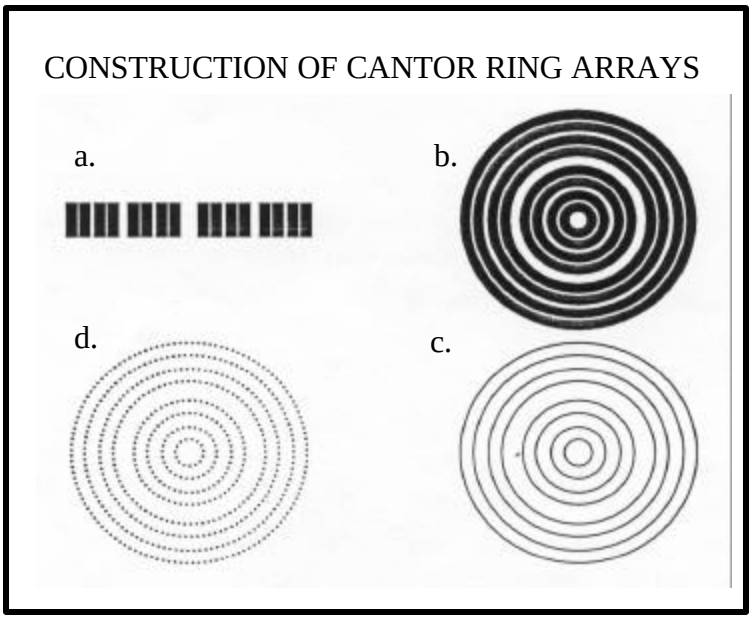

Figure 6 [1]: A discrete cantor ring array is derived from an analogous Cantor bar (a). The Cantor bar is rotated about its midpoint to form a Cantor ring array of finite width annuli (b). The annuli are replaced with infinitesimal width rings (c). These are in turn replaced with discrete radiating elements (d).

For our purposes we attribute to a given discrete Cantor ring array the same dimension as its parent Cantor set *C*. The stage of growth is treated in the same manner as the Cantor bars. We measure lacunarity by the size of the outer gap width, denoted by α. Cantor rings are further described by the number of gaps, n_{gaps}, in the Stage 1 configuration.

There are many proposed measures of lacunarity. For this work we adapt the one suggested by Allain and Cloitre [6,7] and used by Jaggard and Jaggard [1]. In generating Cantor bars and corresponding Cantor rings, we work with $n_{gaps} \geq 3$. We restrict variations in gap width so that all the outer gaps, exempting a possible central gap, are the same size in the first stage of growth. For a given Cantor bar with a specified dimension and n_{gaps} at Stage 1 there is a limited amount of space to distribute among the gaps. α, the width of the homologous outer gaps, can be varied within a given range. The space remaining after determining α for the outer gaps is given to the central gap. In this work α is measured in ten thousandths of the unit interval.

3.4 Planar Arrays and Their Radiated Fields

Figure 7 [1] shows the problem geometry of a discrete Cantor ring array. The discrete elements are placed in the x-y plane. The array factor, *AF*, which is a measure of the radiated field, is examined in the far field as a function of normalized spatial frequency $f_x a = xa/\lambda r$ and $f_y a = ya/\lambda r$ where a is the distance of the element in the radial

direction, $r = \sqrt{x^2 + y^2 + z^2}$ is the observation distance from the origin and λ is the wavelength [1]. The array factor is given by

$$AF = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} e^{-i2\pi n\left(\frac{d}{\lambda}\right)\cos\theta \sin\phi} e^{-i2\pi m\left(\frac{d}{\lambda}\right)\cos\theta \sin\phi}, \tag{6}$$

where $N = M$ = the number of elements, d is the element spacing, θ is the angle in the x-z plane between the antenna element and the observation point in the far-field, and ϕ is the angle in the y-z plane between the antenna element and the observation point in the far-field.

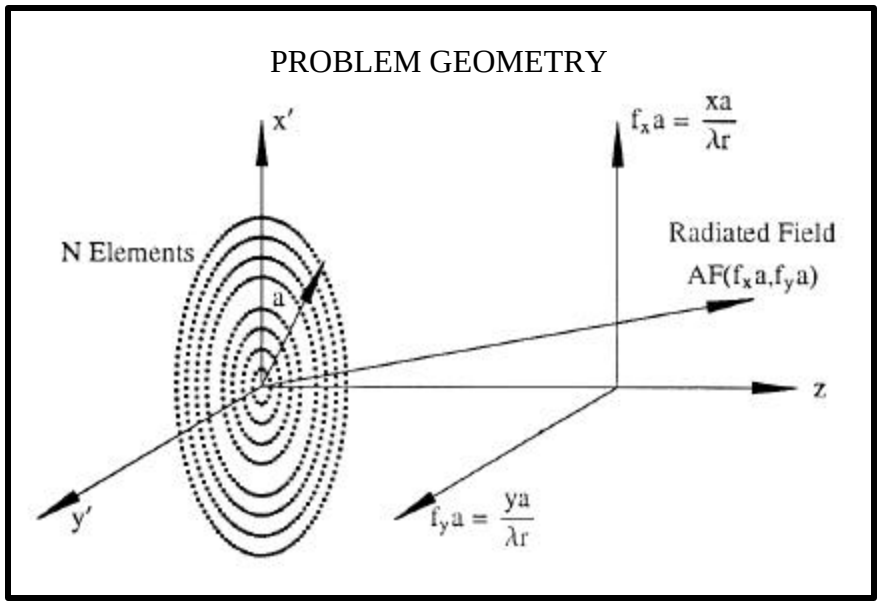

Figure 7: The array factor AF or radiated field is measured in the far-zone ($r >> a, \lambda$) as a function of normalized transverse spatial frequencies $f_x a$ and $f_y a$.

3.5 Departure from Continuous Cantor Rings

After completing MATLAB script to generate the Cantor rings, position antenna elements, and generate AF plots, we duplicated relevant results attained by Jaggard and Jaggard [1]. We generated plots for the continuous case and the discrete case with periodically spaced elements. These graphs provide a point of reference for our subsequent AF plots. Specifically, we concentrated on the class of Cantor rings with the following properties: $D = 9/10$, $S = 2$, $\alpha = 37/1000$, $n_{gaps} = 3$. For the discrete case with periodic spacing, the linear density was such that a ring of radius $r = 1$ had 130 elements. This array had 8 rings with a total of $N = 556$ elements. The elements were allocated as follows, starting with the innermost ring: 15, 29, 45, 59, 80, 94, 110, 124. From the AF

plots of the continuous case, we see that the maximum sidelobe level is –17 dB. For discrete cases we initially expect good arrays to have maximum sidelobes no lower than -13 dB for infinite visible range. The periodic case has no sidelobes higher than –17 dB when the visible range is cut off at $f_r a \approx 21$.

3.6 Azimuthal Element Configurations

We examine three groups of azimuthal configurations, as shown in Figure 8. In all cases, we keep the total number of elements at 556 and maintain the same linear density for each ring. First, we reproduced the periodic case studied by Jaggard and Jaggard. Elements were spaced evenly along each ring, with placement starting at a random point to limit the effect of elements lining up. Such lining up of elements tends to cause high sidelobes along "preferred directions" perpendicular to the line of the elements. The second azimuthal configuration we examined is the random case, where each element is placed randomly within its ring. The third class is a group of semi-random or tethered arrays. We begin by dividing into equal intervals. The number of intervals in each ring corresponds to the number of elements to be placed in that ring. We then place one antenna element randomly within each space. We can tether the elements even tighter by restricting the randomness to be in a given range, for example the middle 3/4s or the middle 1/2 of each interval.

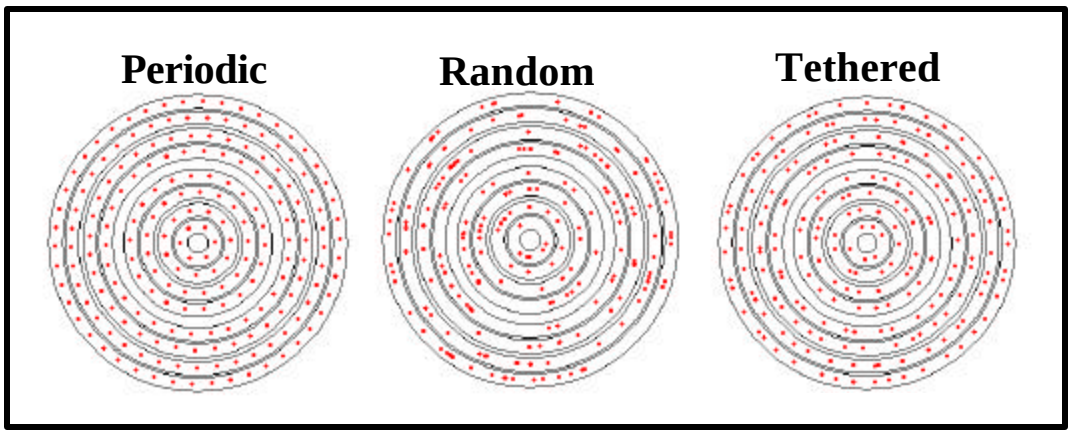

Figure 8: Different azimuthal configurations.

We generate 10 to 20 trials for each case and note the mainbeam, high sidelobe levels, and visible range. We also consider how each class compares to one another and how each compares to the continuous case. Finally, we compare these arrays to equivalent periodic lattices and random disks.

4. DISCUSSION AND CONCLUSIONS

4.1 Array Factor of the Continuous Case

The continuous case of our Cantor ring arrays yields an array factor with a maximum sidelobe level of –17 dB for any visible range. Because the array is composed

of concentric rings, the resulting array factor is a perfectly round mainbeam surrounded by radial radiation waves of varying power that extends to an infinite visible range. The power is distributed evenly in the azimuthal direction, as if, for each radial distance from the mainbeam, all the sidelobes are smeared evenly around the mainbeam. When viewed from above the array factor also resembles a set of concentric rings. High rings are located at $f_r a \approx 8-10,\quad 21.$ Thinning the rings into discrete elements breaks up the "rings" of the array factor, resulting in random sidelobes. Higher linear densities result in creating rings farther from the mainbeam, while lower linear densities result in pushing the random sidelobe region closer to the mainbeam. We refer to the region of rings and the random sidelobe region as the region of fractal control and the random region, respectively.

4.2 Fractal Array Results

The periodic azimuthal configuration results in an average maximum sidelobe level of –13.5 dB, with fluctuations of up to ± 1 dB. These fluctuations occur in the random region. Within the fractal control region at this linear density, a visible range of $f_r a \approx 2\ 1$, the maximum sidelobe level stays at –17 dB.

Randomizing the element configuration brings the average maximum sidelobe level down to -14.5 dB, with fluctuations of up to ± 1.5 dB. Fractal control over the sidelobes is barely noticeable. The rings of the Array Factor disappear, but the general heights of these regions still resemble those of the continuous case. This is expected because the random configuration, in effect, scatters the sidelobe power radially, while loosening its grip on azimuthal evenness.

Tethering the antenna elements lowers the high sidelobes even further. We examine three cases of tethering with ranges 1, middle 3/4, and middle 1/2. For our arrays, a range of 1 is slightly better than a range of 1/2 with the best range falling in-between. With the range restricted to the middle 3/4 of each ring section, our tethered arrays have an average maximum sidelobe level of –16.5 ± 1 dB over an infinite visible range. Much of the azimuthal fractal control over sidelobes is maintained, and combined with the radial redistribution resulting from the randomness. We notice that high rings, and high regions, are susceptible to higher sidelobes. However with an appropriate mixture of order and disorder, low sidelobes are brought up, while high sidelobes are brought down. Representative Array Factor graphs of the periodic and middle 3/4 tethered cases are shown in Figure 9.

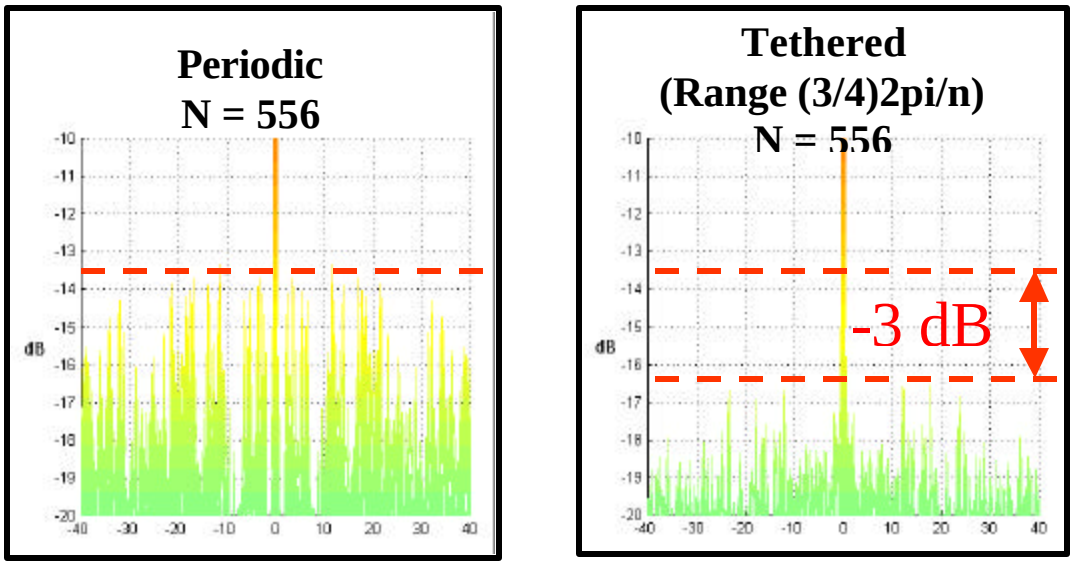

Figure 9: An improvement of 3 dB is gained by changing the azimuthal configuration from periodic to tethered.

4.3 Comparison to Periodic Lattice and Random Disk

A periodic lattice with N = 552 (24x23) has a very good mainbeam with low tapered sidelobes. However, as with all periodic lattices, the visible range is very narrow. A sidelobe threshold of –17 dB provides a visible range of $f_r a \approx 12$, while a threshold of –13 dB gives a visible range of $f_r a \approx 13$. Our cantor array with middle 3/4 range is superior in terms of visible range and robustness, while providing comparable mainbeam structure and sidelobe level.

A random disk of 556 elements has an average maximum sidelobe level of –16.5 ± 1 dB, similar to that of our middle 3/4 range tethered case. However, the mainbeam tends to be highly degraded. Random sidelobes also pop up unpredictably.

4.4 Overall Results and Conclusions

We began this project with Cantor ring arrays that have periodic azimuthal spacing, progressed to random spacing, and then examined various tethered cases. Our middle 3/4 range tethered cases show an improvement of sidelobes –3 dB lower than those of periodic arrays. This translates to arrays with sidelobe power twice as low as our starting point. These arrays not only have lower sidelobe levels, but also have greater visible range and are far more robust. Our tapered arrays perform twice as well as the periodic arrays when using the same number of elements, but also have comparable performance with half as many elements. Practically, this means that we can now build an array that functions much better than before and can have up to half its elements fail. Or to save on cost, we can build a comparable array using half as many elements as before.

5. RECOMMENDATIONS

Further examination can be done with the tethered case to find out what the best range is. Our research shows that this optimal range falls between 1 and 1/2 of the space allotted to each element if the rings are divided into equal spaces equal to the number of elements in each ring. Tethering the antenna elements is analogous to saying that the elements have an underlying structure with short-range disorder - similar in concept to random fractals. This suggests that we may be able to arrange the antenna elements using fractals and random fractals in the azimuthal direction.

We have also begun to look at tapered and inverse tapered arrays. The inverse tapering, in particular seems promising. It appears that by having a greater density of elements on the outer rings, we can push out the region of fractal control over sidelobes, effectively increasing visible range.

6. ACKNOWLEDGEMENTS

I would like to give foremost recognition to Dr. Dwight L. Jaggard for allowing me the privilege of working with him. His work spawned this project, and his guidance, teaching, and enthusiasm assured its success. Thank you for giving me insight into fractals, antennas, experimentation, and research. Thank you for considering this project, and my efforts, important and worthwhile.

I would also like to thank Aaron Jaggard and Hector Dimas for their contributions in giving me direction and insight with my work. Aaron, thanks for going over my work, and thanks for delineating much of the theoretical, mathematical, and programming aspects of this project. Hector, thanks for being as lost as I was at times, and thanks for your major contributions to my MATLAB programming.

I must acknowledge NSF – AMP for funding my research and allowing me to participate in this program. Thank you to Cora Ingrum and Donna Hampton for taking care of the program issues outside of experimentation. Your constant praise and encouragement was greatly appreciated.

I would like to thank the coordinators and facilitators of Sunfest for allowing me to take part in their program. Dr. Van der Spiegel and Lois Clearfield, thank you for making it possible for me to participate in the various seminars and group meetings.

Finally, thank you to the CETS people who handled the hundreds of papers of data generated during my research. Thanks to Bindu, Yale, and the other two girls whose names I cannot remember. Oh, and in case you were not aware, handling my papers happens to be your JOB.

7. REFERENCES

1. D.L. Jaggard and A.D. Jaggard, "Fractal Ring Arrays," to appear in *Wave Motion* (2000-2001)
2. D.L. Jaggard and A.D. Jaggard, "Cantor Ring Arrays," *Mic. and Opt. Tech. Let*., 19 (1998) pp. 121-125
3. A.D. Jaggard and D.L. Jaggard, "Cantor Ring Diffractals," *Opts. Comm.*, 158 (1998) pp. 141-148
4. B.B. Mandelbrot, *The Fractal Geometry of Nature*, Freeman, New York, 1983
5. D.L. Jaggard, "On Fractal Electromagnetics," Ch. 6 in *Recent Advances in Electromagnetic Theory*, pp. 183-224, Springer-Verlag, Berlin 1990
6. C. Allain and M. Cloitre, "Characterizing the Lacunarity of Random and Deterministic Fractal Sets," *Phys. Rev*. A 44, 3552-3558 (1991).
7. A.D. Jaggard and D.L. Jaggard, "Scattering from Fractal Superlattices with Variable Lacunarity," *J. Opt. Soc. Am*. A 15, 1626-1635 (1998).

8. BIBLIOGRAPHY

1. W.L. Weeks, *Antenna Engineering*, McGraw-Hill, New York, 1968, pp. 62-78
2. *MATLAB User's Guide*, *Student Edition*
3. Ulaby and Fawwaz, *Fundamentals of Applied Electromagnetics*, Prentice Hall, New Jersey, 1999
4. S.W. Lee, *Antenna Handbook, Theory, Applications, & Design*, Reinhold Company, New York, 1998

9. APPENDICES

Appendix A: (MATLAB script used to generate Cantor sets, Cantor rings, and AF plots)

```
%Cantor Set Generator
%Array Factor Generator
%af(2,214,37,214,70,214,37,214,0)

%Input the Stage
%Input the individual lengths of the Stage 1 intevals and gaps (Curds and Tremas)
function
af(Stage,CurdSize1,TremaSize1,CurdSize2,TremaSize2,CurdSize3,TremaSize3,...

CurdSize4,TremaSize4,CurdSize5,TremaSize5,CurdSize6,TremaSize6,CurdSize7,Trema
Size7,...
  CurdSize8,TremaSize8,CurdSize9,TremaSize9,CurdSize10);
%**************************************************************************
**************
%**************************************************************************
**************
%Generates the Stage 1 Cantor Set based on the input parameters
%Each if statement creates a curd or trema if its respective input parameters are given.
%The program takes in the CurdSize and places it in the set, then it takes in the
%TremaSize and places it an appropriate distance from the previous Curd. This continues
%as long as there are input variables available. A zero input given to any trema
%terminates the set generating at that point.

%Stage 0
if Stage == 0,
  C_Set = [0 1];
  L = 1;
  stage = 0;
end

%Stage 1 Generator
if Stage >= 1,
  %Curd1
  Curd1 = [0 CurdSize1];        %create curd
  S = Curd1;              %add the curd to the set
  L = S(length(S));           %set length

  if TremaSize1 ~= 0,
   %Trema1
   Trema1 = [L + TremaSize1];      %create trema
   L = L + TremaSize1;         %change set length to account for trema
   scaler = lcm(CurdSize1,TremaSize1); %scale factor for the set, this allows for
```

```
%Curd2              %different sized curds and tremas
Curd2 = [L + CurdSize2];       %create curd
S = [S Trema1 Curd2];        %add curd to the set
L = L + CurdSize2;          %change set length to account for curd
scaler = lcm(scaler,CurdSize2);   %scale factor for the set

if TremaSize2 ~= 0,
  %Trema2
  Trema2=[L + TremaSize2];
  L = L + TremaSize2;
  scaler = lcm(scaler,TremaSize2);
  %Curd3
  Curd3 = [L + CurdSize3];
  S = [S Trema2 Curd3];
  L = L + CurdSize3;
  scaler = lcm(scaler,CurdSize3);

  if TremaSize3 ~=0,
   %Trema3
   Trema3 = [L + TremaSize3];
   L = L + TremaSize3;
   scaler = lcm(scaler,TremaSize3);
   %Curd4
   Curd4 = [L + CurdSize4];
   S = [S Trema3 Curd4];
   L = L + CurdSize4;
   scaler = lcm(scaler,CurdSize4);

   if TremaSize4 ~= 0,
    %Trema4
    Trema4 = [L + TremaSize4];
    L = L + TremaSize4;
    scaler = lcm(scaler,TremaSize4);
    %Curd5
    Curd5 = [L + CurdSize5];
    S = [S Trema4 Curd5];
    L = L + CurdSize5;
    scaler = lcm(scaler,CurdSize5);

    if TremaSize5 ~= 0,
     %Trema5
     Trema5 = [L + TremaSize5];
     L = L + TremaSize5;
     scaler = lcm(scaler,TremaSize5);
     %Curd6
     Curd6=[L + CurdSize6];
```

```
S = [S Trema5 Curd6];
L = L + CurdSize6;
scaler = lcm(scaler,CurdSize6);

if TremaSize6 ~=0,
  %Trema6
  Trema6 = [L + TremaSize6];
  L = L + TremaSize6;
  scaler = lcm(scaler,TremaSize6);
  %Curd7
  Curd7 = [L + CurdSize7];
  S = [S Trema6 Curd7];
  L = L + CurdSize7;
  scaler = lcm(scaler,CurdSize7);

  if TremaSize7 ~= 0,
   %Trema7
   Trema7 = [L + TremaSize7];
   L = L + TremaSize7;
   scaler = lcm(scaler,TremaSize7);
   %Curd8
   Curd8 = [L + CurdSize8];
   S = [S Trema7 Curd8];
   L = L + CurdSize8;
   scaler = lcm(scaler,CurdSize8);

   if TremaSize8 ~= 0,
    %Trema8
    Trema8 = [L + TremaSize8];
    L = L + TremaSize8;
    scaler = lcm(scaler,TremaSize8);
    %Curd9
    Curd9 = [L + CurdSize9];
    S = [S Trema8 Curd9];
    L = L + CurdSize9;
    scaler = lcm(scaler,CurdSize9);

    if TremaSize9 ~= 0,
     %Trema9
     Trema9 = [L + TremaSize9];
     L = L + TremaSize9;
     scaler = lcm(scaler,TremaSize9);
     %Curd10
     Curd10 = [L + CurdSize10];
     S = [S Trema9 Curd10];
     L = L + CurdSize10;
```

```
            scaler = lcm(scaler,CurdSize10);

          end
         end
        end
       end
      end
     end
    end
   end
  end

  %Stage 1
  C_Set = S;
  %S;
  %L;
  stage = 1;
end
%*************************************************************************
**************
%*************************************************************************
**************
%This part of the program generates Cantor Sets at Stage 2 or higher. It takes the set
%created above and generates another stage by scaling up (not down!). The scaler and
%scaler_temp variables facilitate this upward scaling.

if Stage > 1,
  stage=2;
  scaler_temp = scaler;

  while stage <= Stage,          %generate stages
   C_Set = S * gcd(scaler,CurdSize1);
   if TremaSize1 ~= 0,
     shift = C_Set(length(C_Set)) + (gcd(scaler_temp,TremaSize1))*S(length(S));
     C_Set = [C_Set (S * gcd(scaler_temp,CurdSize2) + shift)];
     if TremaSize2 ~= 0,
      shift = C_Set(length(C_Set)) + (gcd(scaler_temp,TremaSize2))*S(length(S));
      C_Set = [C_Set (S * gcd(scaler_temp,CurdSize3) + shift)];
      if TremaSize3 ~=0,
        shift = C_Set(length(C_Set)) + (gcd(scaler_temp,TremaSize3))...
         *S(length(S));
        C_Set = [C_Set (S * gcd(scaler_temp,CurdSize4) + shift)];
        if TremaSize4 ~= 0,
         shift = C_Set(length(C_Set)) + (gcd(scaler_temp,TremaSize4))...
           *S(length(S));
         C_Set = [C_Set (S * gcd(scaler_temp,CurdSize5) + shift)];
```

```
        if TremaSize5 ~= 0,
          shift = C_Set(length(C_Set)) + (gcd(scaler_temp,TremaSize5))...
           *S(length(S));
          C_Set = [C_Set (S * gcd(scaler_temp,CurdSize6) + shift)];
          if TremaSize6 ~=0,
           shift = C_Set(length(C_Set)) + (gcd(scaler_temp,TremaSize6))...
             *S(length(S));
           C_Set = [C_Set (S * gcd(scaler_temp,CurdSize7) + shift)];
           if TremaSize7 ~= 0,
             shift = C_Set(length(C_Set)) + (gcd(scaler_temp,TremaSize7))...
              *S(length(S));
             C_Set = [C_Set (S * gcd(scaler_temp,CurdSize8) + shift)];
             if TremaSize8 ~= 0,
              shift = C_Set(length(C_Set)) + (gcd(scaler_temp,...
                TremaSize8))*S(length(S));
              C_Set = [C_Set (S * gcd(scaler_temp,CurdSize9) + shift)];
              if TremaSize9 ~= 0,
                shift = C_Set(length(C_Set)) + (gcd(scaler_temp,...
                 TremaSize9))*S(length(S));
                C_Set = [C_Set (S * gcd(scaler_temp,CurdSize10) ...
                  + shift)];
              end
             end
            end
           end
          end
         end
        end
      end
     end
    end

    S = C_Set;
    scaler_temp = scaler_temp * scaler;
    L = C_Set(length(C_Set));
    stage = stage + 1;
   end
end

%Stage N
%C_Set;
%L;
%**************************************************************************
**************
%**************************************************************************
**************
%Generate the set of Radii from the midpoints of the Cantor Set Tremas
```

```
r=1;
while r < length(C_Set),
  Radius((r + 1) / 2) = (C_Set(r) + C_Set(r+1)) / 2;
  r = r + 2;
end

Radius;
%*************************************************************************
**************
%*************************************************************************
**************
%Generate a list of the Points and Midpoints of the Cantor Set
ii=1;
radius_w_zeros=[];
while ii <= length(C_Set),
  if mod(ii,2) ~= 0,
   radius_w_zeros = [radius_w_zeros Radius((ii + 1) / 2)];
   ii = ii + 1;
  elseif mod(ii,2) == 0,
   radius_w_zeros = [radius_w_zeros 0];
   ii = ii + 1;
  end
end

shifted_unit_radius=[];
for ii = 1:length(radius_w_zeros),
  if radius_w_zeros(ii) == 0,
   shifted_unit_radius(ii) = 0;
  else
   shifted_unit_radius(ii) = 2 * (radius_w_zeros(ii)/L) - 1;
  end
end

unit_set = C_Set / L;
shifted_unit_set = 2 * unit_set - 1;
unit_radius = radius_w_zeros / L;
%shifted_unit_radius;

Cantor_Points = [C_Set', unit_set', shifted_unit_set', ...
   radius_w_zeros', unit_radius', shifted_unit_radius']
%C_Set;
%shifted_unit_set;
%shifted_unit_radius
%*************************************************************************
*************
```

```
%*************************************************************************
*************
%Plot the Cantor Set and the Midpoints
figure(1);
y=0;
plot(C_Set / L,y,'k.');
grid on;
hold on;
plot(Radius / L,y,'r.');
title(['Cantor Set and Midpoints, Stage ', num2str(Stage)]);
hold off;
axis('square');
%*************************************************************************
**************
%*************************************************************************
**************
%Half the Cantor Set so that its midpoint is at Zero
Half_C_Set = C_Set;
Half_C_Set(1:(length(Half_C_Set)/2)) = [];
Half_unit_set = unit_set;
Half_unit_set(1:(length(Half_unit_set)/2)) = [];
Half_shifted_unit_set = shifted_unit_set;
Half_shifted_unit_set(1:(length(Half_shifted_unit_set)/2)) = [];

Half_radius_w_zeros = radius_w_zeros;
Half_radius_w_zeros(1:(length(Half_radius_w_zeros)/2)) = [];
Half_unit_radius = unit_radius;
Half_unit_radius(1:(length(Half_unit_radius)/2)) = [];
Half_shifted_unit_radius = shifted_unit_radius;
Half_shifted_unit_radius(1:(length(Half_shifted_unit_radius)/2)) = [];

Shifted_Set = [Half_C_Set', Half_unit_set', Half_shifted_unit_set', ...
   Half_radius_w_zeros', Half_unit_radius', Half_shifted_unit_radius']

ii = 1;
while ii <= length(Half_shifted_unit_radius),
  if Half_shifted_unit_radius(ii) == 0,
   Half_shifted_unit_radius(ii) = [];
   ii = ii + 1;
  else
   ii = ii + 1;
  end
end


%Half_C_Set;
```

```
%Half_shifted_unit_set;
Array_Radius = Half_shifted_unit_radius;
%Array_Radius = [enter desired radius]
%*************************************************************************
**************
%*************************************************************************
**************
%Rings corresponding to Cantor Set
figure(2);
theta = 0:0.001:2*pi;
ii=1;
while ii <= length(Half_shifted_unit_set),
  plot((Half_shifted_unit_set(ii)*cos(theta))/...
     (Half_shifted_unit_set(length(Half_shifted_unit_set))), ...
    (Half_shifted_unit_set(ii)*sin(theta))/...
     (Half_shifted_unit_set(length(Half_shifted_unit_set))),'k-');
  ii = ii + 1;
  hold on;
end
%Different ways to plot antenna elements along midpoints of Cantor Set, Use only one at
a time
%*************************************************************************
**************
%*************************************************************************
**************
%%Periodic Lattice
%x=[];
%y=[];
%x_length=24;
%y_length=23;

%x_increment = linspace(0, x_length, x_length);
%y_increment = linspace(0, y_length, y_length);

%for ii = 1:x_length,
%  for jj = 1:y_length,
%    x = [x x_increment(ii)];
%    y = [y y_increment(jj)];
%  end
%end

%x = x*0.0720;
%y = y*0.0720;

%N = x_length * y_length;
%Lattice_Area = x(length(x)) * y(length(y))
```

```
%Cantor_Area = pi * (Array_Radius(length(Array_Radius)))^2

%x = x - x(length(x))/2;
%y = y - y(length(y))/2;
%*************************************************************************
**************
%*************************************************************************
**************
%Random Disk
%x=[];
%y=[];
%N = 556;
%ii = 1;
%while ii <= N,
%  x_gen = Array_Radius(length(Array_Radius)) * rand(1);
%  y_gen = Array_Radius(length(Array_Radius)) * rand(1);
%  if (sqrt(x_gen^2 + y_gen^2)) <= Array_Radius(length(Array_Radius)),
%   x = [x x_gen];
%   y = [y y_gen];
%   ii = ii + 1;
%  end
%end

%x_quadrant = rand(size(x));
%for ii = 1:N,
%  if x_quadrant(ii) < 0.5,
%   x(ii) = -x(ii);
%  end
%end
%y_quadrant = rand(size(y));
%for ii = 1:N,
%  if y_quadrant(ii) < 0.5,
%   y(ii) = -y(ii);
%  end
%end
%*************************************************************************
**************
%*************************************************************************
**************
%Linear Density the same for all radii of the Cantor Rings
%ii=1;
%N=0;
%n=0;
%radial_density_factor=0.295;       %used to determine N per ring
%Elements_Per_Ring=[];
%x=[];
```

```
%y=[];
%while ii <= length(Array_Radius),
%  n = round(2*2*pi*Array_Radius(ii)/(2*radial_density_factor)+1);
%  Elements_Per_Ring = [Elements_Per_Ring n-1];
%  N = N + n - 1;
%  theta = linspace(0,2*pi,n) + rand(1);
%  theta(1) = [];
%  x = [x [Array_Radius(ii) * cos(theta)]];
%  y = [y [Array_Radius(ii) * sin(theta)]];
%  ii = ii + 1;
%end
%                  %Display radii and elements per ring
%Array_Radius_AND_Elements_Per_Ring = [Array_Radius' Elements_Per_Ring']
%**************************************************************************
**************
%**************************************************************************
**************
%Outer Radii have greater linear density than inner radii
%ii=length(Array_Radius);
%N=0;
%n=0;
%Elements_Per_Ring=[];
%x=[];
%y=[];

%radial_density_factor=0.0334;
%RDFs = radial_density_factor;
%density_scaler=1.2
%exp_den_scaler=1
%Linear_Density_r5=[];
%while ii >= 1,
%  n = round(2*2*pi*Array_Radius(ii)/(2 * radial_density_factor)+1);
%  Linear_Density_r5 = [Linear_Density_r5 (2*pi/radial_density_factor)];
%  radial_density_factor = radial_density_factor * (density_scaler^exp_den_scaler);
%  RDFs = [RDFs radial_density_factor];
%  Elements_Per_Ring = [Elements_Per_Ring n-1];
%  N = N + n - 1;
%  theta = linspace(0,2*pi,n) + rand(1);
%  theta(1) = [];
%  x = [x [Array_Radius(ii) * cos(theta)]];
%  y = [y [Array_Radius(ii) * sin(theta)]];
%  ii = ii - 1;
%end

%Rev_Array_Radius = [];
%for ii = 1:length(Array_Radius),
```

```
%  Rev_Array_Radius(ii) = Array_Radius(length(Array_Radius) - ii + 1);
%end
%RDFs(length(RDFs))=[];          %Display radii, # elements, density
%Radius_NumPerRing_Density = [Rev_Array_Radius' Elements_Per_Ring'
Linear_Density_r5' RDFs']
%*************************************************************************
**************
%*************************************************************************
**************
%Specify N per ring, random placement within ring
%n = [15 29 45 59 80 94 110 124];       %number of elements per ring
%x=[];
%y=[];
%
%for ii = 1:length(n),
%  rand_theta = 2*pi*rand(ceil(sqrt(n(ii))));
%  for jj = 1:n(ii);
%   x = [x [Array_Radius(ii) * cos(rand_theta(jj))]];
%   y = [y [Array_Radius(ii) * sin(rand_theta(jj))]];
%  end
%end
%
%Radius_NumberElements = [Array_Radius' n']
%N = n(1) + n(2) + n(3) + n(4) + n(5) + n(6) + n(7) + n(8);
%*************************************************************************
**************
%*************************************************************************
**************
%%Random placing within a given range
%ii=1;
%N=0;
%n=0;
%radial_density_factor=0.0485;       %used to determine N per ring
%Elements_Per_Ring=[];
%x=[];
%y=[];
%while ii <= length(Array_Radius),
%  n = round(2*2*pi*Array_Radius(ii)/(2*radial_density_factor)+1);
%  Elements_Per_Ring = [Elements_Per_Ring n-1];
%  N = N + n - 1;
%  theta = linspace(0,2*pi,n) + 2*pi*rand(1);
%  theta(1) = [];
%  range = 2*pi/(n - 1);
% rand_theta = range*rand(ceil(sqrt(n)));
%  for jj = 1:length(theta),
%   theta(jj) = theta(jj) + rand_theta(jj);
```

```
%  end
%  x = [x [Array_Radius(ii) * cos(theta)]];
%  y = [y [Array_Radius(ii) * sin(theta)]];
%  ii = ii + 1;
%end
%                   %Display radii and elements per ring
%Array_Radius_AND_Elements_Per_Ring = [Array_Radius' Elements_Per_Ring']
%**************************************************************************
**************
%**************************************************************************
**************
%Input each linear density
n = [15 29 45 59 80 94 110 124];       %number of elements per ring
x=[];
y=[];

%r = 0.1158
theta = linspace(0,2*pi,n(1) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(1) * cos(theta)]];
y = [y [Array_Radius(1) * sin(theta)]];

%r = 0.2232
theta = linspace(0,2*pi,n(2) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(2) * cos(theta)]];
y = [y [Array_Radius(2) * sin(theta)]];

%r = 0.3448
theta = linspace(0,2*pi,n(3) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(3) * cos(theta)]];
y = [y [Array_Radius(3) * sin(theta)]];

%r = 0.4522
theta = linspace(0,2*pi,n(4) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(4) * cos(theta)]];
y = [y [Array_Radius(4) * sin(theta)]];

%r = 0.6178
theta = linspace(0,2*pi,n(5) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(5) * cos(theta)]];
y = [y [Array_Radius(5) * sin(theta)]];
```

```
%r = 0.7252
theta = linspace(0,2*pi,n(6) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(6) * cos(theta)]];
y = [y [Array_Radius(6) * sin(theta)]];

%r = 0.8468
theta = linspace(0,2*pi,n(7) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(7) * cos(theta)]];
y = [y [Array_Radius(7) * sin(theta)]];

%r = 0.9542
theta = linspace(0,2*pi,n(8) + 1) + 2*pi*rand(1);
theta(1) = [];
x = [x [Array_Radius(8) * cos(theta)]];
y = [y [Array_Radius(8) * sin(theta)]];

Radius_NumberElements = [Array_Radius' n']
N = n(1) + n(2) + n(3) + n(4) + n(5) + n(6) + n(7) + n(8);
%**************************************************************************
**************
%**************************************************************************
**************
N                   %Display the total number of elements
%x = x;
%y = y;
plot(x,y,'r.');
grid on;
hold off;
title(['N = ', num2str(N), ', D = 9/10, S = ', num2str(Stage),...
   ', n_g_a_p_s = 3, e = 37/1000, af(2,214,37,214,70,214,37,214,0)']);
axis('square');

%Antenna Element grid positions
%x;
%y;
points = [x' y'];
size(points)
%**************************************************************************
**************
%**************************************************************************
**************
%AF Plots
Num_Points = size(points,1);
a=-60;
```

```
b=60;
c=0.2;
[fx,fy] = meshgrid(a:c:b, a:c:b);
ii = 1;
AF = 0;
%Plot Array Points
figure(3);
while ii <= Num_Points,
  af = exp(-i*2*pi * (fx * points(ii,1) + fy * points(ii,2)));
  plot(points(ii,1),points(ii,2),'r.');
  AF = AF + af;
  ii = ii + 1;
  hold on;
end
title(['N = ', num2str(N), ', D = 9/10, S = ', num2str(Stage),...
   ', n_g_a_p_s = 3, e = 37/1000, af(2,214,37,214,70,214,37,214,0)']);
hold off;
axis('square');
axis([-1 1 -1 1]);
Normalized_AF = abs(AF / Num_Points);

%Add an infinetesmal amount to all the AF components
%This prevents taking the log of zero
ii=1;
while ii <= (size(Normalized_AF,1) * size(Normalized_AF,2))
  Normalized_AF(ii) = Normalized_AF(ii) + .00001;
  ii = ii + 1;
end

%Change the Array Factor into db scale
db_AF= 20 * log10(Normalized_AF);

%Filter negligible side lobes
ii=1;
cutoff = 60;
while ii <= (size(db_AF,1) * size(db_AF,2))
  if db_AF(ii) > -cutoff;
   db_AF(ii) = db_AF(ii) + cutoff;
  else db_AF(ii) = 0;
  end;
  ii = ii + 1;
end

%Plot Top View
figure(4);
pcolor(fx,fy,db_AF - cutoff)
```

```
shading interp
colormap(jet)
axis xy
title(['N = ', num2str(N), ', D = 9/10, S = ', num2str(Stage),...
   ', n_g_a_p_s = 3, e = 37/1000, af(2,214,37,214,70,214,37,214,0)']);
axis('square');

%Plot Side View
figure(5);
mesh(fx,fy,db_AF - cutoff)
shading interp
colormap(jet)
axis xy
axis([a b a b -cutoff 0])
axis('square')
view([90 0])
title(['Side View: N = ', num2str(N), ', D = 9/10, S = ', num2str(Stage),...
   ', n_g_a_p_s = 3, e = 37/1000, af(2,214,37,214,70,214,37,214,0)']);

%Plot Side View, Between -20 and -10 db
figure(6);
mesh(fx,fy,db_AF - cutoff)
shading interp
colormap(jet)
axis xy
axis([a b a b -20 -10])
axis('square')
view([90 0])
title(['Side View: N = ', num2str(N), ', D = 9/10, S = ', num2str(Stage),...
   ', n_g_a_p_s = 3, e = 37/1000, af(2,214,37,214,70,214,37,214,0)']);

%Plot 3-D
figure(7);
mesh(fx,fy,db_AF - cutoff)
shading interp
colormap(jet)
axis xy
axis([a b a b -cutoff 0])
%axis('square')
title(['N = ', num2str(N), ', D = 9/10, S = ', num2str(Stage),...
   ', n_g_a_p_s = 3, e = 37/1000, af(2,214,37,214,70,214,37,214,0)']);

%Plot Radial Cut
figure(8);
for ii = 1 : ((b - a)/c + 1),
  db_AF(ii, 2 : ((b - a)/c + 1)) = 0;   %if meshgrid has mainbeam at origin
```

```
%  db_AF(ii, 1 : b/c) = 0;          %if meshgrid has mainbeam in center
%  db_AF(ii, b/c + 2 : (b-a)/c + 1) = 0;
end
mesh(fx,fy,db_AF - cutoff)
shading interp
colormap(jet)
axis xy
axis([a b a b -cutoff 0])
axis('square')
view([90 0])
title(['Radial Cut: N = ', num2str(N), ', D = 9/10, S = ', num2str(Stage),...
   ', n_g_a_p_s = 3, e = 37/1000, af(2,214,37,214,70,214,37,214,0)']);
%*************************************************************************
**************
%*************************************************************************
**************
```