Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/5307--20-10,118

Computation of Power Beaming Efficiency in the Fresnel Zone with Application to Amplitude and Phase Optimization

Judith V. Hutson

Formely, Advanced Concepts Group
Radar Division

Christopher T. Rodenbeck

Advanced Concepts Group
Radar Division

April 15, 202

REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 15-04-2021	**2. REPORT TYPE** NRL Memorandum Report	**3. DATES COVERED** *(From - To)* Aug 2019 – June 2020
4. TITLE AND SUBTITLE Computation of Power Beaming Efficiency in the Fresnel Zone with Application to Amplitude and Phase Optimization		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER 64020A
6. AUTHOR(S) Judith V. Hutson* and Christopher T. Rodenbeck		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER 1M11
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory, 4555 Overlook Avenue, SW, Washington, DC 20375-5320		**8. PERFORMING ORGANIZATION REPORT NUMBER** NRL/MR/5307--20-10,118
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Research Laboratory, 4555 Overlook Avenue, SW, Washington, DC 20375-5320		**10. SPONSOR / MONITOR'S ACRONYM(S)** NRL
		11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

*Formerly, Advanced Concepts Group, Radar Division

14. ABSTRACT

A MATLAB program capable of calculating the efficiency of a wireless power transfer system is described. The script's operation is based upon an equation derived from the Huygens-Fresnel principle, which was discretized for use in a numerical solution. The software's user interface and outputs are briefly outlined, with specific features explained separately. Transfer efficiency improvement via phase focusing and amplitude optimization is also discussed. These features are explained and demonstrated using a simulation test case. Finally, comparison is drawn with existing efficiency data.

15. SUBJECT TERMS

Fresnel zone Huygens-Fresnel principle Wireless power transfer Aperture Transmitter Receiver Antenna Phased array Optimization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Christopher Rodenbeck
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	U	58	**19b. TELEPHONE NUMBER** *(include area code)* (202) 404-1596

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

This page intentionally left blank.

CONTENTS

This page intentionally left blank.

COMPUTATION OF MICROWAVE POWER BEAMING EFFICIENCY, WITH APPLICATION TO AMPLITUDE AND PHASE OPTIMIZATION

1. INTRODUCTION

The following report describes an application for simulating microwave power beaming based on the Huygens-Fresnel principle. The program determines the power distribution and total received power at a given distance from a transmitting antenna or array. Amplitude and phase distribution across the transmitter may be specified to observe the effects, or optimized automatically for power transfer efficiency. The software's user interface and principles of operation are described, and its source code is given in Appendix 1. An alternative version with inputs directly within the main MATLAB script, rather than a GUI, is described in Appendix 2.

2. CALCULATION METHODS

The fundamental operation of this simulation is based upon the relationships described below.

2.1 Basic Principles

The following equation relates the electromagnetic field *F* across a source aperture with the resultant field *F'* across the plane of interest (POI) [1] defined in Figure 1:

$$F'(\zeta,\eta,\kappa) = \iint\limits_A \frac{f\big(\theta(x,y)\big)}{j\lambda r(x,y)} F(x,y) e^{-jkr(x,y)} dxdy \tag{1}$$

The function f(x,y) was determined using a combination of [1] and [2] to be

$$f(x,y) = \frac{\cos(\theta) + \vec{\boldsymbol{s}} \cdot \hat{\boldsymbol{z}}}{2}$$

The term $\vec{\boldsymbol{s}} \cdot \hat{\boldsymbol{z}}$ refers to the direction of propagation $\vec{\boldsymbol{s}}$ of the wave emanating from a given source point in relation to the normal vector of the aperture $\hat{\boldsymbol{z}}$, and depends upon the phase gradient in the x- and y-directions at that point [2].

$F(x,y) = Ae^{-j\varphi}$ is, in general, a complex number defined by the amplitude *A* and phase φ at point *P(x,y,0)* on the transmit aperture [2]. Likewise, $F'(\zeta,\eta,\kappa) = A'e^{-j\varphi'}$ where *A'* and φ' are the amplitude and phase of the resultant field at a point *P'(ζ,η,κ)* on the POI. Referring to Figure 1, the distance *r* and angle θ between *P* and *P'* are

$$r = \sqrt{(\zeta - x)^2 + (\eta - y)^2 + \kappa^2},$$

$$\theta(x,y) = \arctan\left(\frac{\sqrt{(\zeta-x)^2+(\eta-y)^2}}{r(x,y)}\right).$$

Manuscript approved April 6, 2021.

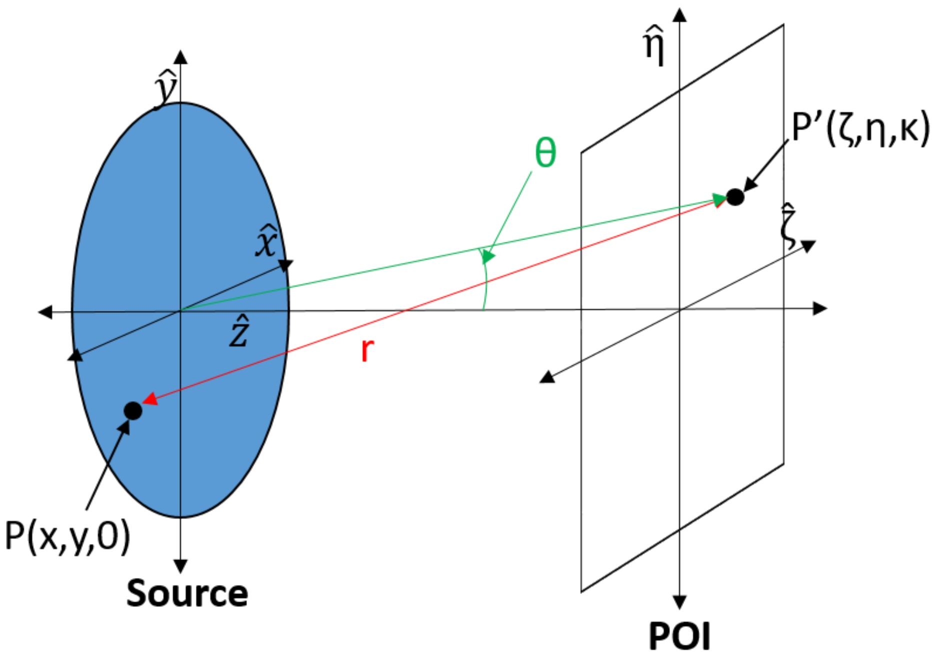


Fig. 1 - A description of the coordinate system and variables used in the calculations discussed here

This equation can be derived from the Huygens-Fresnel principle, which states that an electromagnetic source or scatterer may be represented as the sum of individual point sources, each of which radiates isotropically. Any electromagnetic wave is therefore composed of an infinite number of spherical waves, weighted accordingly [2].

2.2 Discretization

Equation 1 was derived for a continuous aperture, but can be adapted into a discrete form necessary for numerical calculations and phased arrays. For an array with a total number of elements *N*,

$$F'(\zeta,\eta,\kappa) = \sum_{i=1}^{N} \frac{f\big(\theta(x_i,y_i)\big)}{j\lambda r(x_i,y_i)} F(x_i,y_i) e^{-jkr(x_i,y_i)} \Delta x_i \Delta y_i \tag{2}$$

where (x_i,y_i) are the coordinates of the center of an element, and $\Delta x_i \Delta y_i$ is that element's area. This assumes uniform amplitude and phase across the element's aperture.

2.3 Assumptions

Although Equations 1 and 2 are based upon the Huygens-Fresnel principle, they also rely on simplifying assumptions. Firstly, the emitted radiation's wavelength must be much lower than the dimensions of the aperture, and the plane of interest must be several wavelengths away at a minimum. Ultimately, the frequency must be high when compared with the general dimensions of the problem.

Additionally, there is the assumption that the radiated field, as well as the resultant calculated field, is concentrated near the *z*-axis [1] so that the flow of power is highly directional. As a result, these formulas are most useful for antennas or arrays that are already known to possess high gain. A third condition is that the field is uniformly polarized across the source aperture [2]. Variation in field direction across the aperture at any given point in time must be minimal, as it could lead to cancellation effects that are not accounted for.

3. APPLICATION

This program allows the calculation of the electromagnetic fields a specified distance away from a transmitting antenna aperture, and the power incident upon a receiving array at that range. The simulated transmitter may be a continuous antenna or a phased array, each of which will produce slightly different models. In the case of a phased array, each source point represents an element, and they are spaced to model the elements' centers. As a result, the source points do not extend to the absolute edge of the aperture; half of an antenna element would exist between the two. Aperture shape may be circular or square, and the radiation shadow caused by the sub-reflector of certain types of antennas may also be included.

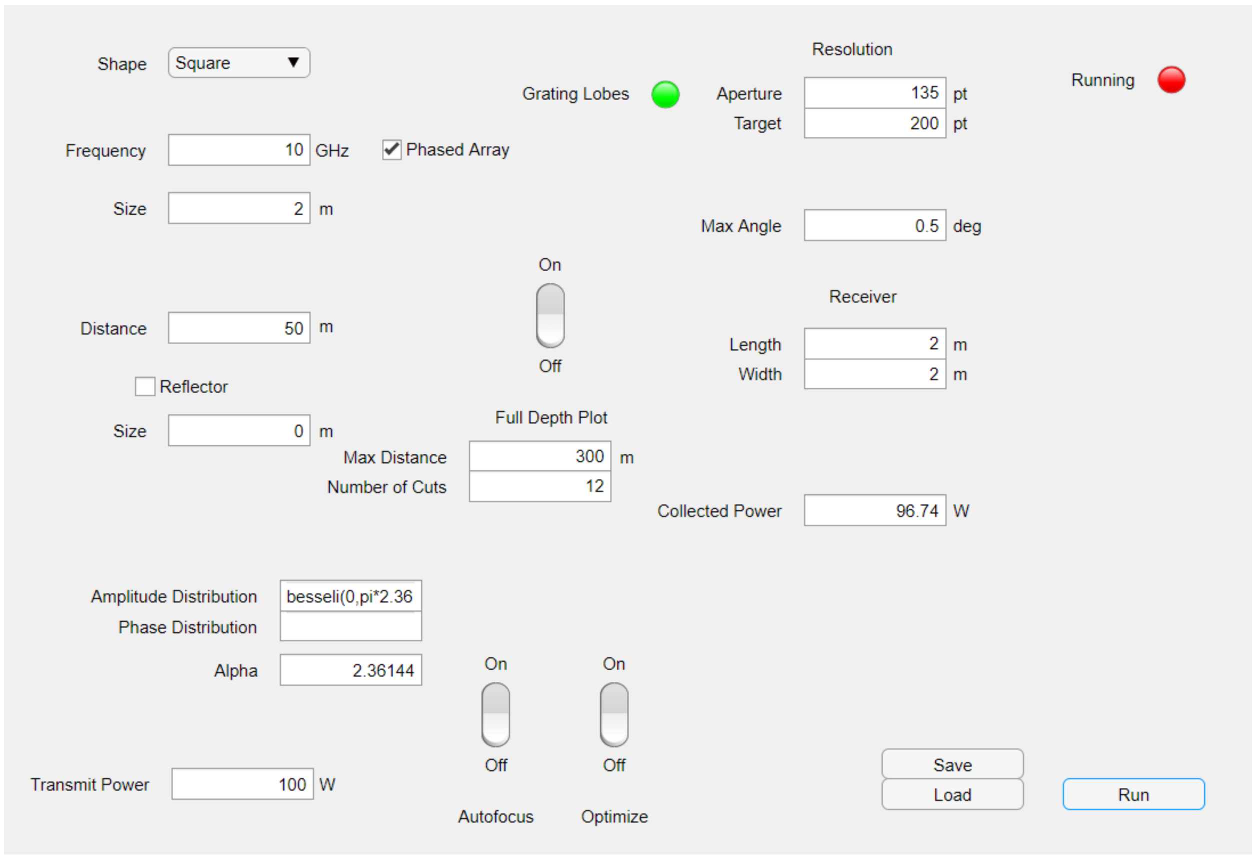
The application also possesses the ability to perform power transfer optimization. The 'Autofocus' switch enables or disables phase focusing at the specified distance, while the 'Optimize' option performs a simple optimization algorithm on the source amplitude distribution, outputting the results to the 'Alpha' field. These processes are explained further in Section 4.

An optional output is a top-down visualization of the beam produced by the transmitting aperture. Although this increases computing time, it can be helpful in understanding the radiation propagation pattern and the effects of optimization. If enabled, this feature requires additional inputs to set graph parameters, which are explained in Section 3.3.

Limitations and cases that may produce inaccurate results do exist, and are described in Section 3.4.

3.1 Interface

Figure 2 shows the developed application's user interface, with its elements described in Table 1. Additionally, the quantities of 'Size', 'Resolution: Aperture', 'Max Angle' and 'Distance' are illustrated in Figures 3 and 4.



Shape: Square

Frequency: 10 GHz

☑ Phased Array

Size: 2 m

Distance: 50 m

☐ Reflector

Size: 0 m

Resolution — Aperture: 135 pt; Target: 200 pt

Max Angle: 0.5 deg

Receiver — Length: 2 m; Width: 2 m

Full Depth Plot — Max Distance: 300 m; Number of Cuts: 12

Collected Power: 96.74 W

Amplitude Distribution: besseli(0,pi*2.36

Phase Distribution:

Alpha: 2.36144

Transmit Power: 100 W

Fig. 2 - The application user interface

Table 1 - User interface fields

Shape	Source aperture shape
Frequency	Transmitted frequency
Size	Source max size in axial directions (Fig. 3)
Phased Array	Check if source is phased array rather than continuous
Distance	Distance between source and POI (Fig. 4)
Reflector	Check if subreflector is present
Reflector: Size	Size of subreflector using same convention as 'Size'
Max Distance	Range of depth plot
Number of Cuts	Number of calculated planes in depth plot
Amplitude Distribution	Radial amplitude distribution; output if 'Optimize' is on
Phase Distribution	Radial phase distribution; no effect if 'Autofocus' is on
Alpha	Optimized amplitude distribution parameter
Transmitted Power	Total power transmitted

Resolution: Aperture	Source resolution along one axis (Fig. 3)
Resolution: Target	POI resolution along one axis
Max Angle	Determines size of POI (Fig. 4)
Receiver: Length	Size of receive array in y-dimension
Receiver: Width	Size of receive array in x-dimension
Collected Power	Power incident upon receive array
Autofocus	Automatic phase focusing
Optimize	Automatic amplitude optimization
Running	Green if currently running
Grating Lobes	Red if resolution spacing is greater than half-wavelength

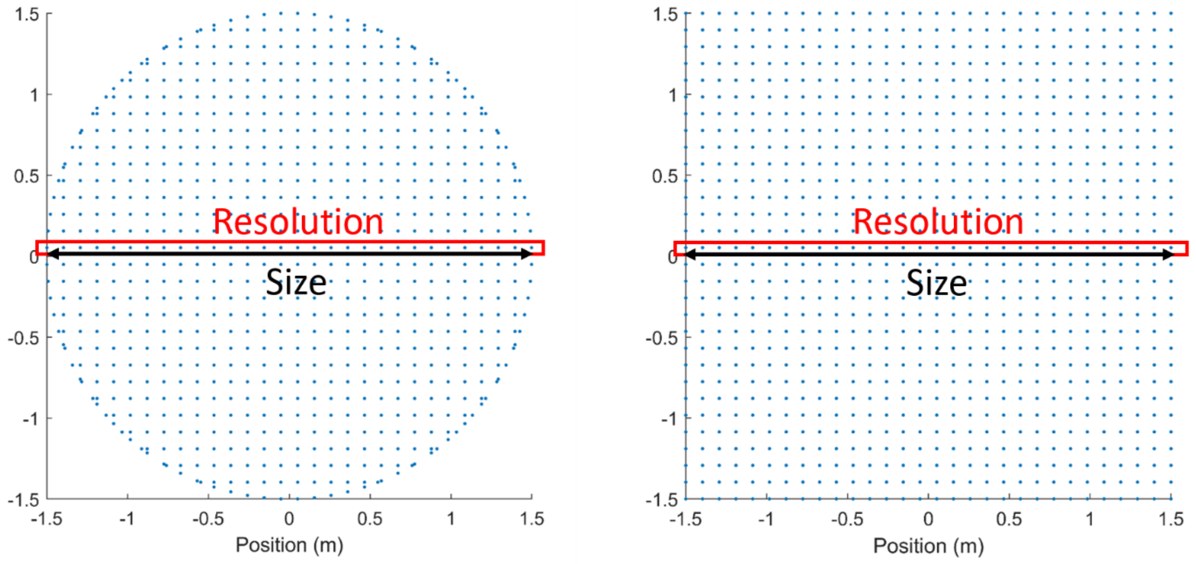


Fig. 3 - 'Size' refers to the width of the aperture in each axial dimension. 'Resolution: Aperture' is the number of points that represent the aperture along each axis.

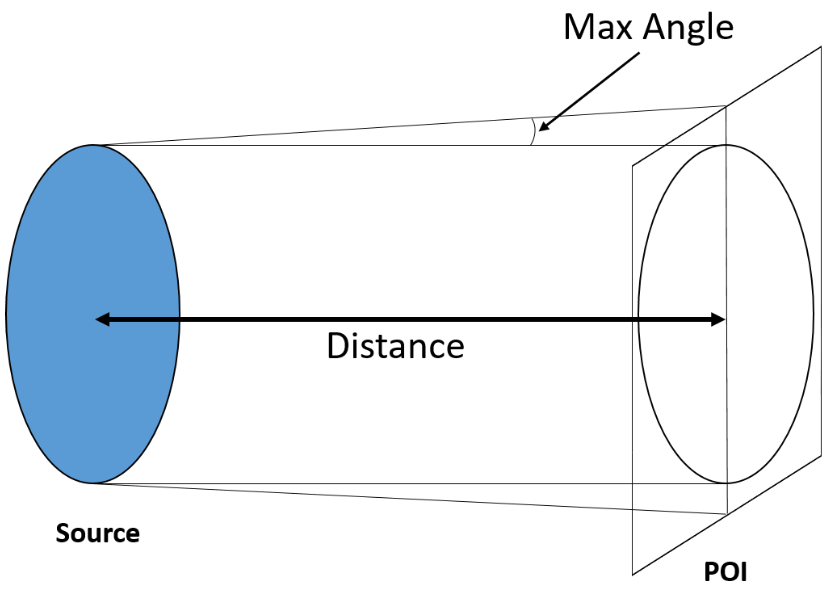
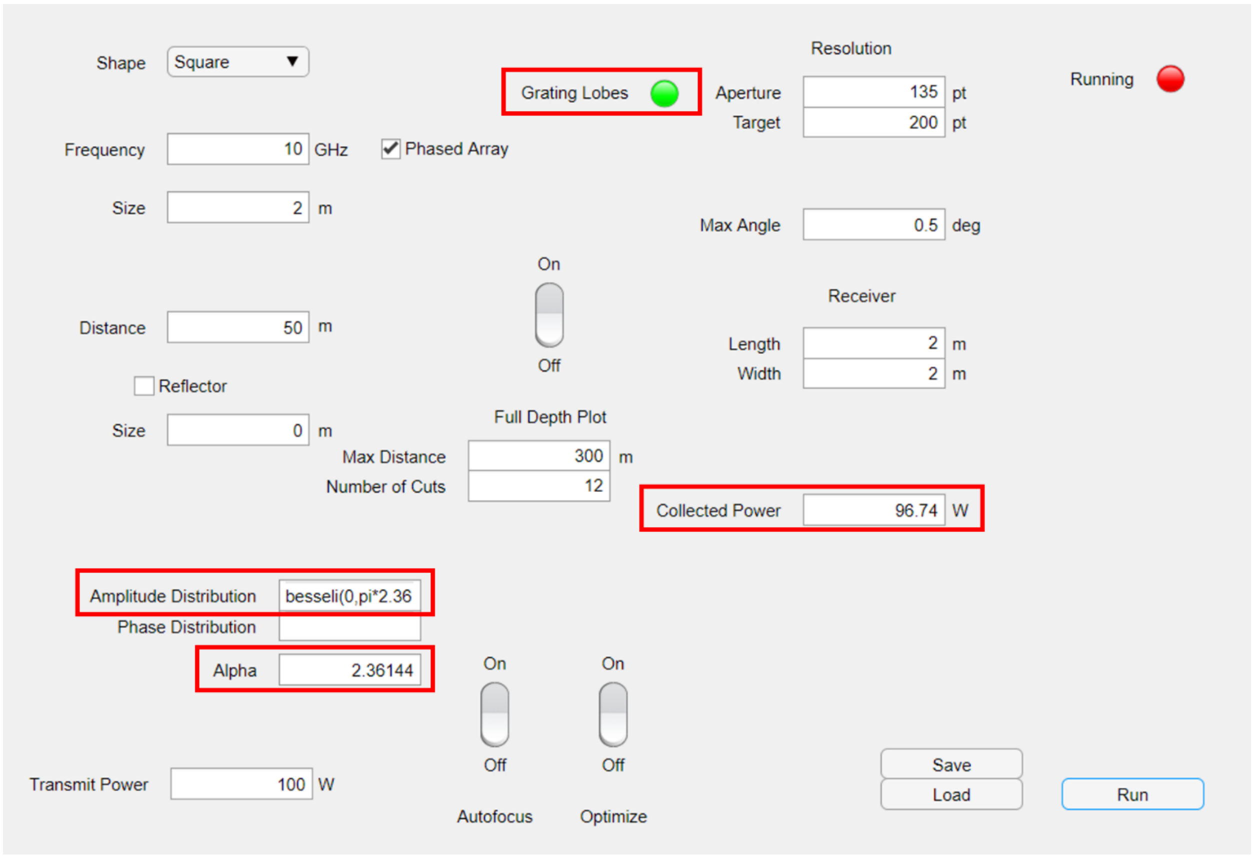


Fig. 4 - 'Max Angle' determines how much larger the POI is than the aperture. It is based upon angle for more intuitive input at large distances.

3.2 Outputs

Program outputs are routed to the user interface, as shown in Figure 5. They are the power incident upon the receiving array, the optimal amplitude distribution if that function is enabled, and the presence of grating lobes in the simulation, which if captured on the plane of interest could lead to inaccuracies in the collected power. The 'Grating Lobes' indicator will turn red if the source point spacing is wide enough to produce grating lobes. However, if the grating lobes themselves are spaced widely enough that they do not interfere with the plane of interest, the calculated result may be acceptable.



Shape: Square

Grating Lobes ●

Resolution

Aperture: 135 pt

Target: 200 pt

Running ●

Frequency: 10 GHz

✓ Phased Array

Size: 2 m

Max Angle: 0.5 deg

Distance: 50 m

Receiver

Length: 2 m

Width: 2 m

Reflector

Size: 0 m

Full Depth Plot: Off

Max Distance: 300 m

Number of Cuts: 12

Collected Power: 96.74 W

Amplitude Distribution: besseli(0,pi*2.36

Phase Distribution:

Alpha: 2.36144

Transmit Power: 100 W

Autofocus: Off

Optimize: Off

Fig. 5 - Program GUI outputs, based upon the inputs in Figure 4

The application also produces four figures, shown in Figure 6. The first two visualize the construction and amplitude distribution of the source aperture, and the latter two show power density incident upon the plane of interest.

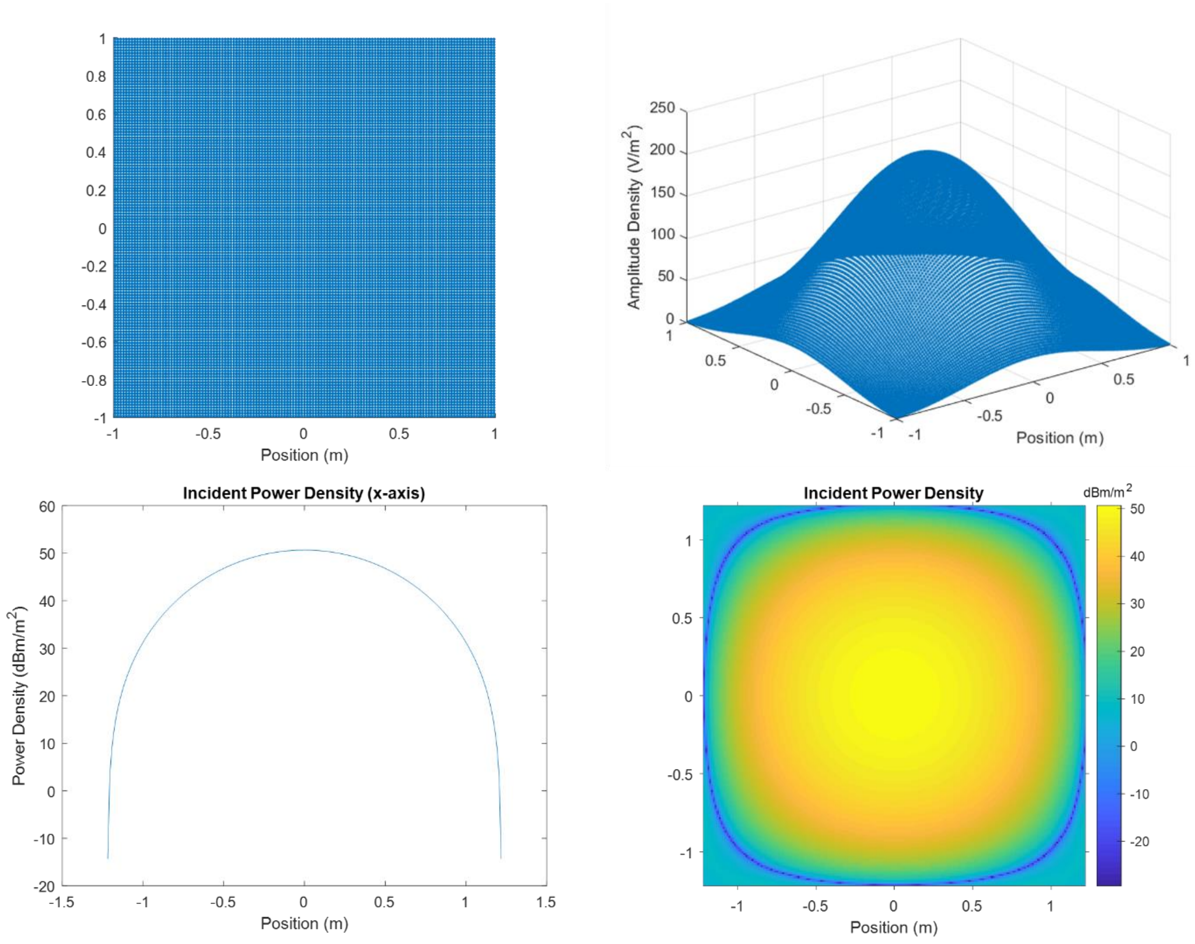


Fig. 6 - Program graphical outputs, based on the inputs in Figure 4. Only the first four graphical outputs are shown here, as the fifth is optional and described later.

3.3 Features

This section provides more detail on specific program features.

3.3.1 Amplitude and Phase Distribution

For direct control over the transmitting aperture's amplitude and phase, the user can input equations into the 'Amplitude Distribution' and 'Phase Distribution' fields. The inputs must be MATLAB-readable text, with 'x' as the variable. 'Amplitude Distribution' will be an input if 'Optimize' is off and will output the result of the optimization algorithm otherwise. Additionally, the related parameter 'α', described in Section 4.2.1, will be displayed separately for better readability. If 'Autofocus' is off, the 'Phase Distribution' field will act as an input; if it is on, that field is ignored and phase focusing is applied automatically.

3.3.2 Reflector Shadow

If present, the antenna sub-reflector is assumed to take the same shape as the main aperture and be positioned at its center. As a result, the only parameter necessary to input for this feature is 'Size', which is measured as shown in Figure 3. Figure 7 shows the effect of a blockage on the source amplitude distribution, as well as the result on the plane of interest. The generating parameters are the same as those used to create the plots in Figure 6, with the exception that the aperture distribution was not re-optimized after the blockage was added. This was done to create a similar comparison. Plots corresponding to the blocked aperture are on the right of the figure, with their unblocked counterparts on the left.

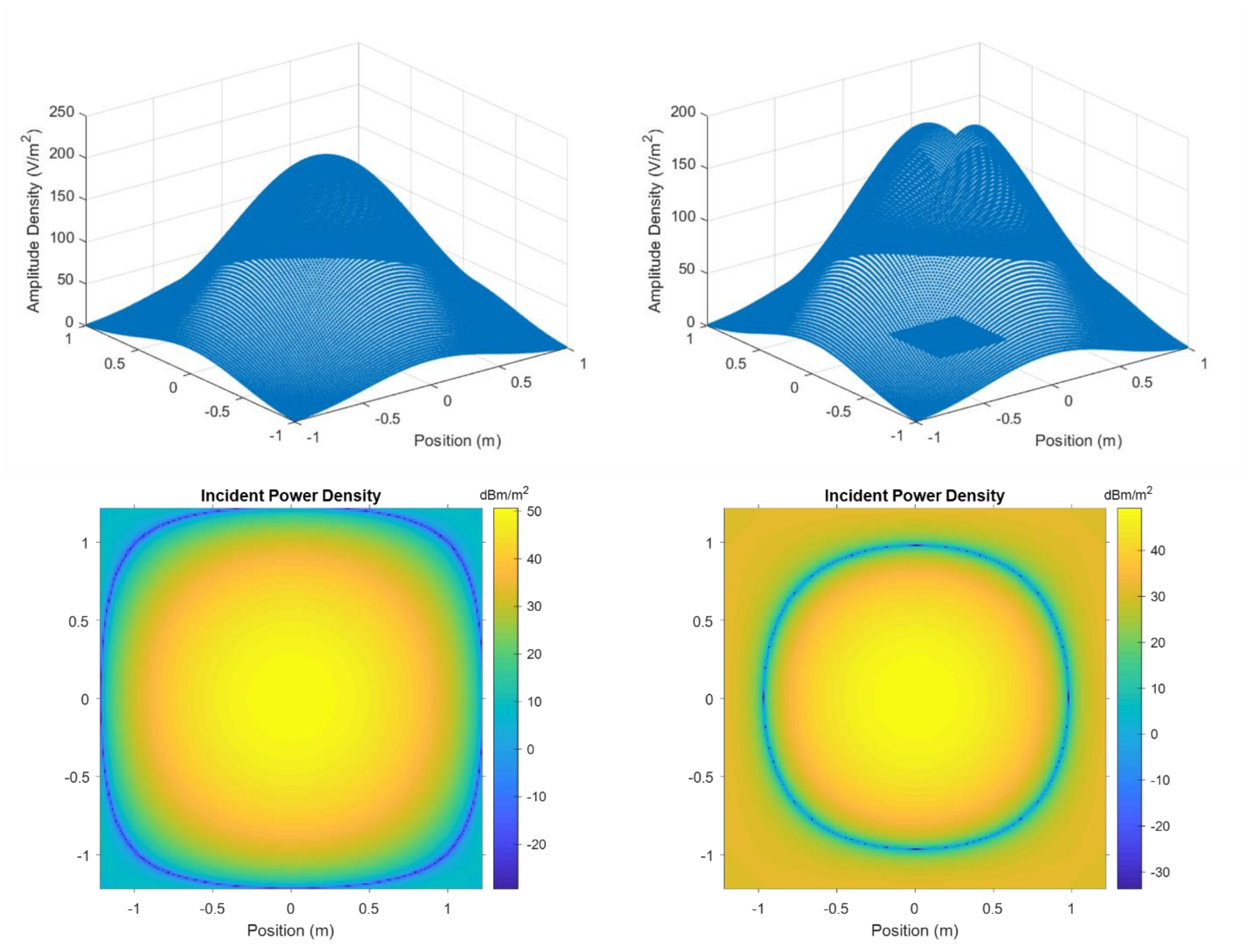


Fig. 7 – Comparison of output figures with and without an aperture blockage. In this case the blockage was .5m per side.

3.3.3 Depth Plot

If the ‘Full Depth Plot’ switch is enabled, the ‘Max Distance’ field must be populated in addition to ‘Distance’. This determines the total range of the depth plot. All other plots and calculated values, such as received power, will use the ‘Distance’ parameter. The field labeled ‘Number of Cuts’ will set the number of planes calculated in producing the plot. As each plane increases computation time, the calculation is expected to be coarse, and linear interpolation is used as a smoothing method. An example of the resultant graphic is shown in Figure 8.

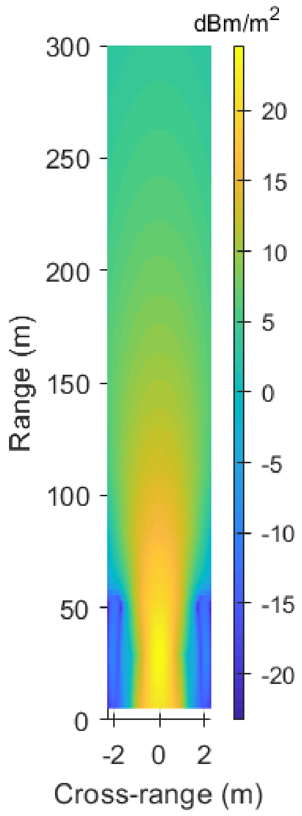


Fig. 8 – An optional fifth output figure, based on the inputs in Figure 4, showing a representation of the beam as range varies. These plots rely on interpolation, so may not be fully accurate at all ranges.

3.4 Limitations

The calculations upon which this program is based stand upon a few underlying assumptions, which have been described previously. Additionally, if a continuous aperture is being simulated, and the aperture resolution is greater than one half of the wavelength, the program produces nonphysical grating lobes that may interfere with the resulting power pattern and/or the receiving array incident power calculation. When this occurs, the 'Grating Lobes' lamp in the user interface will turn red. To remedy this situation, the user can increase the 'Aperture Resolution' field, or simply narrow the POI so that only the main lobe is visible in the resulting plots.

4. POWER TRANSFER IMPROVEMENT

Although the equations described in Section 2 form the basis of the simulator's operation, the automated optimization of power transfer requires additional methods. Although it is fairly simple to create a phase distribution that focuses power to the requisite distance, the ideal amplitude distribution is more complex, and employs a simple optimization algorithm.

An example of transfer improvement using each method is also given. The test case used in this section is a square 2m by 2m phased array with 135 elements per side transmitting 100W at 10GHz toward a 2m by 2m receiver. Simulations were performed for a separating distance of 50, 100, 150, and 200 meters. The plane of interest had a resolution of 200 points per side.

4.1 Phase Focusing

4.1.1 Method

To maximize power flow through a given area, the beam must be focused. This is a straightforward task if one uses ray approximations. Constructive interference is desired at the focal point, meaning the energy coming from each direction must be aligned in phase. Incoming radiation should therefore follow a spherical equiphase surface centered at the focal point [3]. If the focus lies along the central axis of the antenna aperture, the radius of this sphere will simply be the distance between the two planes, as shown in figure 9. Points may be brought into alignment by adding initial phase corresponding to the distance between the source point and that equiphase surface. This can be expressed by the equation

$$\varphi = -\left(\sqrt{x^2+y^2+z^2}-z\right)\cdot\frac{2\cdot\pi}{\lambda} \tag{3}$$

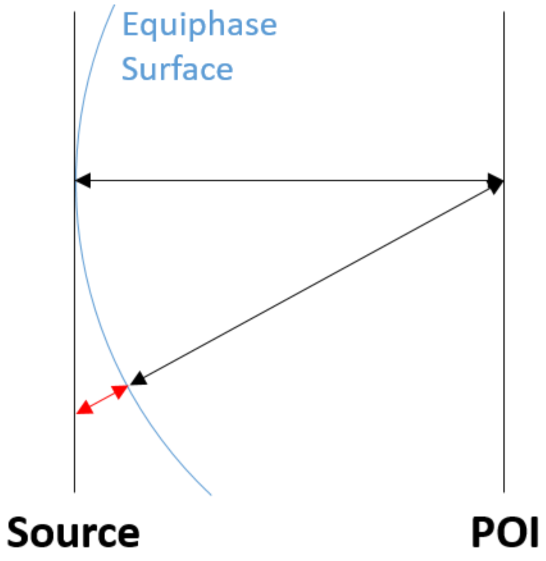


Fig. 9 - The distance shown in red can be multiplied by the wavenumber k to obtain a point's initial phase

This method, of course, fails to account for diffraction effects. However, it should bring close to an optimal phase distribution without the need for computational optimization, and provides a starting point for said optimization if it is determined to be necessary.

4.1.2 Results

With uniform amplitude and phase distribution, the amount of power flowing through a 2m x 2m area 50m from the transmitting array is calculated to be 68.34W. This corresponds to an efficiency of 68.3%, and will be considered the base case for this distance. If power is focused by applying a quadratic phase distribution, the received power increases by 19.56% to 81.71W. Cross-sections of the power incident on the plane of interest are shown in Figure 10.

At 200m, however, the increase is only 4.81%, from 32.25 to 33.8W. As the distance between the transmitting and receiving arrays increases, the improvement seen by focusing the transmitted energy diminishes. This is because at a large distance, the radius of the spherical equiphase surface necessary to focus power is large in comparison to the aperture size; therefore, the imparted phase distribution is near uniform. The array's gain is near the maximum value allowed by its aperture size even without any effort to focus the beam, so little improvement can be achieved. This is shown in Figure 11.

Table 2 - Effect of phase focusing

Distance (m)	Unfocused Power (W)	Focused Power (W)	Improvement
50	68.34	81.71	19.56%
100	62.24	72.94	17.19%
150	46.70	50.52	08.18%
200	32.25	33.80	04.81%

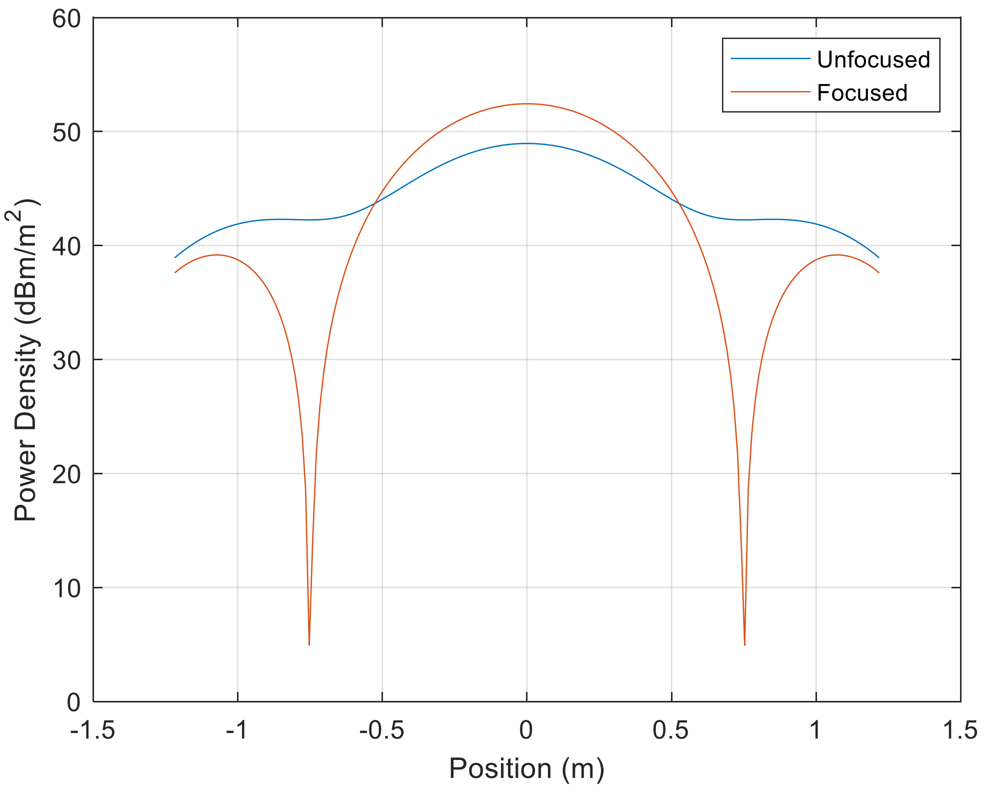


Fig. 10 - Cross-section (x-axis) of power incident on plane of interest at 50m

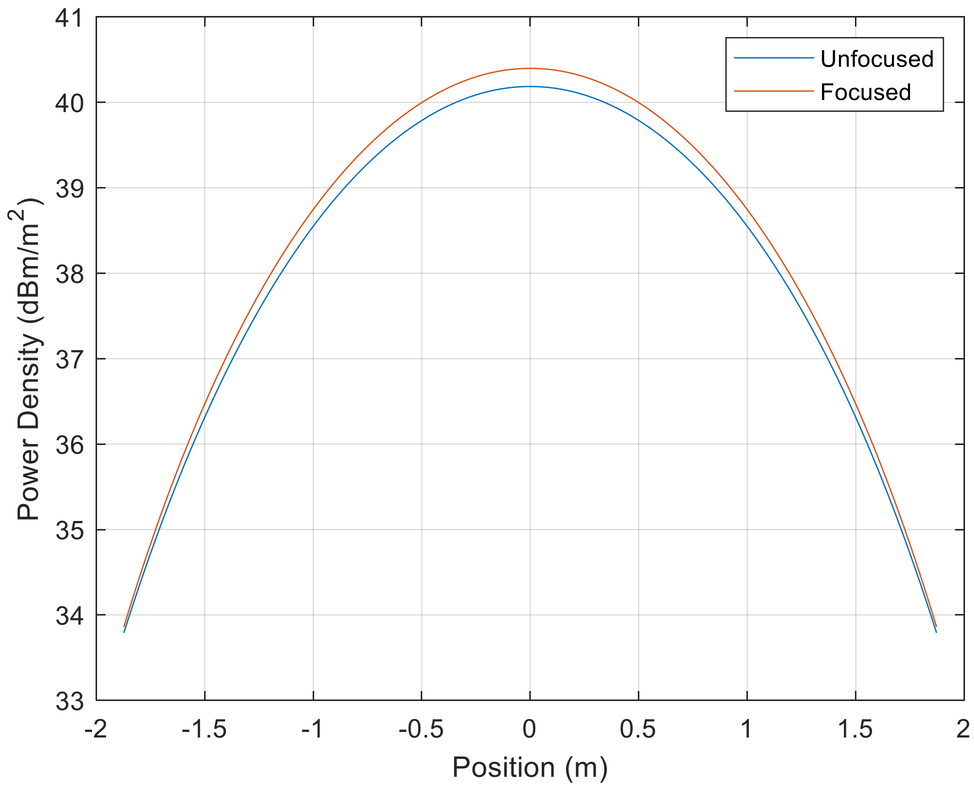


Fig. 11 - Cross-section (x-axis) of power incident on plane of interest at 200m. The change produced by focusing a beam is far more dramatic at shorter distances.

4.2 Amplitude Distribution

The amplitude distribution optimization method used by the software is chosen from three possibilities after simulation using the test case. This process is described below.

4.2.1 Experiment Setup

Determining the optimal amplitude distribution for a given aperture and receiver is not as simple as focusing the beam. There is not an obvious analytical solution, and if each point were to be considered an independent variable, the computational resources required to optimize the problem would be unreasonable. Furthermore, because the final result would be discrete, further optimization would be needed to determine how best to interpolate on a continuous aperture, or on a phased array of higher resolution than that simulated. If the shape of the distribution is constrained, the number of parameters can be reduced to one or two, leading to a reasonable optimization. Of course, this approach does not guarantee a theoretically optimal solution, but significant improvement can still be seen. Three major distribution types were investigated, and each function was applied radially, rotated about the aperture's normal axis.

A distribution commonly used in wireless power transfer is the Gaussian [4]. This distribution is written

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{4}$$

where μ is the center value, chosen here as the geometric center of the array, and σ is the curve's standard deviation [5]. σ is optimized using a modified golden-section search algorithm. A golden-section search tests four values of a parameter, narrowing its range and adding a new test value based upon the results of those four [6]. The modified version used here expands its range if the optimum test value lies at the edge of the range tested, ensuring that a maximum is found.

In [4], it was reported that a trapezoidal distribution could perform better in some situations than the Gaussian. It is also simpler in implementation [4], meriting investigation. This distribution depends upon two parameters- the width of the central plateau w (m) and the slope m of the sides (m^{-1}). They were optimized together using a pattern search algorithm.

Additionally, as the goal is essentially to maximize energy flow through a given solid angle, a standard phased array window function was also investigated. Window functions are commonly used to reduce array side lobes at the expense of a wider main lobe, which can be tolerated to an extent in some instances. A Kaiser window was chosen due to its dependence on a single parameter, leading to the possibility of optimization beyond function choice [7].

The Kaiser window is

$$f(x) = \begin{cases} \dfrac{I_0\left[\pi\alpha\sqrt{1-\left(\frac{2x}{L}\right)^2}\right]}{I_0[\pi\alpha]} & \dfrac{-L}{2} \le x \le \dfrac{L}{2} \\ 0 & |x| > \dfrac{L}{2} \end{cases} \tag{5}$$

where I_0 is the zeroth-order modified Bessel function of the first kind and L is the length of the window [8]. In the test case, the transmitting array is square in shape, and L was taken as the diagonal so that no part of the array fell outside of the window. Figure 12 illustrates the distribution as it might be applied to a discrete array.

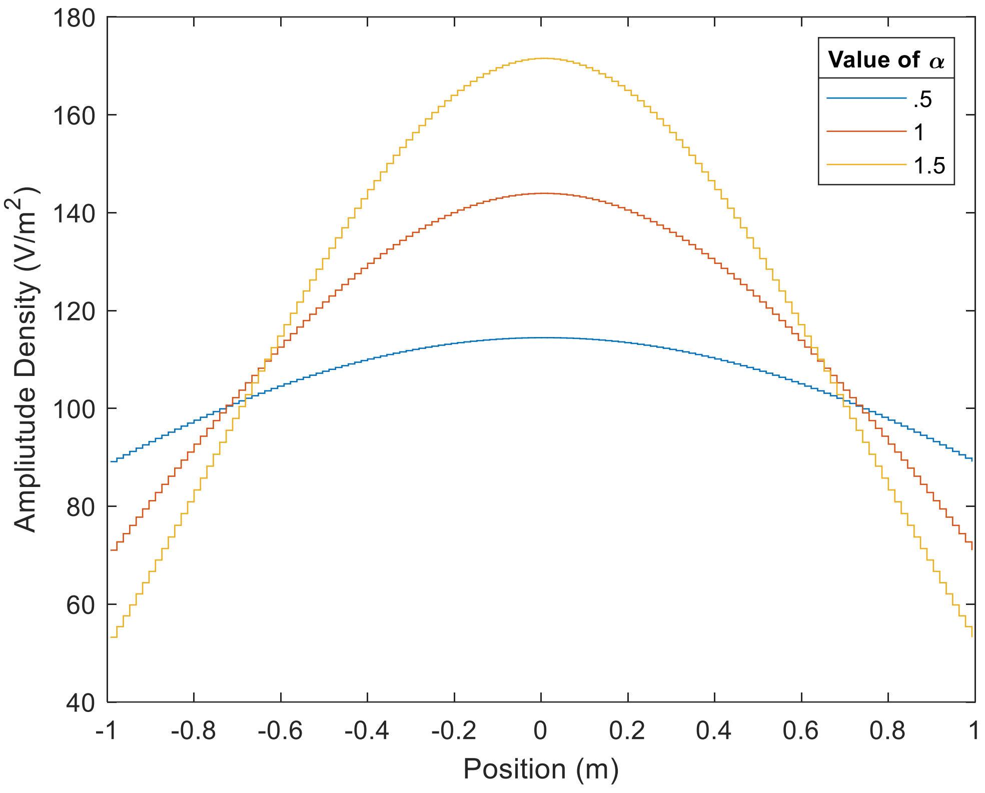


Fig. 12 - An example of the amplitude across one row of elements of an array, simulated with the parameters of the test case used elsewhere in this section. Kaiser distributions with varying values of α have been applied.

The parameter α, like σ, was optimized using a modified golden-section search algorithm. The convergence criterion for both optimization algorithms was a maximum difference in test results less than 0.5% of the highest result. Results of amplitude optimization are shown below.

4.2.2 Results

Table 3 illustrates the results of amplitude optimization for the test case described at the beginning of this section. At near distances, amplitude optimization offers a significant increase in efficiency, and although all three distribution shapes give nearly the same amount of power after optimization, the Kaiser window is generally most effective, with the Gaussian distribution only slightly behind. As distance increases, although improvement is still seen, it becomes very small, and the difference between the three shapes is negligible. Similarly to phase focusing, due to the limited gain of the transmitting array, a receiver of a given size that is far away will not capture the majority of the power beam. As the power cannot be further concentrated by introducing an amplitude distribution, efficiency improvement is limited.

Table 3 - Amplitude optimization results

Distance=50m

Distribution	Parameter Value(s)	Power	Improvement
None		81.71W	
Gaussian	σ=.50798	96.68W	18.32%

Kaiser	α=2.3614	96.74W	18.39%
Trapezoidal	w=.8147; m=.9845	96.07W	17.57%

Distance=100m

Distribution	Parameter Value(s)	Power	Improvement
None		72.94W	
Gaussian	σ=.90261	77.76W	6.61%
Kaiser	α=.90223	77.77W	6.62%
Trapezoidal	w=.6155 ; m=.7	77.75W	6.59%

Distance=150m

Distribution	Parameter Value(s)	Power	Improvement
None		50.62W	
Gaussian	σ=1.2470	51.45W	1.64%
Kaiser	α=.55764	51.45W	1.64%
Trapezoidal	w=.5 ; m=.5	51.42W	1.58%

Distance=200m

Distribution	Parameter Value(s)	Power	Improvement
None		33.80W	
Gaussian	σ=1.8042	33.99W	.56%
Kaiser	α=.34474	33.98W	.53%
Trapezoidal	w=.5 ; m=.3	33.98W	.53%

4.3 Analysis

The power received by a rectifying array can be significantly improved by adding an amplitude and phase distribution to the transmitting aperture. Phase focusing greatly raises efficiency at shorter distances, and because it is based off of simple constructive interference principles, should remain applicable to an arbitrary transmitter shape. If a transmitter is square, a Kaiser window function was found to be superior to both a Gaussian and a trapezoidal distribution, although any of the three offered significant improvement over a uniform amplitude. The Gaussian and Kaiser distributions are similar, and thus performed nearly identically. The Kaiser was very slightly superior, and is the shape used in the application as a result. When the optimization is run, the distribution equation used in the final calculation appears in the 'Amplitude Distribution' field, with α also displayed alone in 'Alpha' for easier recognition.

5. VERIFICATION

Efficiency data generated by the application was compared to a reference figure included in William C. Brown’s 1973 paper “Adapting Microwave Techniques to Help Solve Future Energy Problems” [9], which was itself taken from an earlier work by George Goubau [10]. Its independent variable is the parameter τ, which is defined as

$$\tau = \frac{\sqrt{A_t A_r}}{\lambda \kappa} \tag{6}$$

where A_t is the area of the transmitting aperture and A_r is the receiver area. λ and κ are defined as in Section 2.1.

As can be seen in Figure 13, this tool agrees well with the data given in previous work. Both phase focusing and amplitude distribution optimization were applied, with similar conditions noted in [9]. The simulation used a circular phased array with a diameter of 2m broken into 135 elements, and a corresponding 2m by 2m receiving area. Energy was transferred at 10GHz, and the distance between the transmitter and receiver was varied to produce the appropriate values of τ. Target plane resolution was 250 by 250 points.

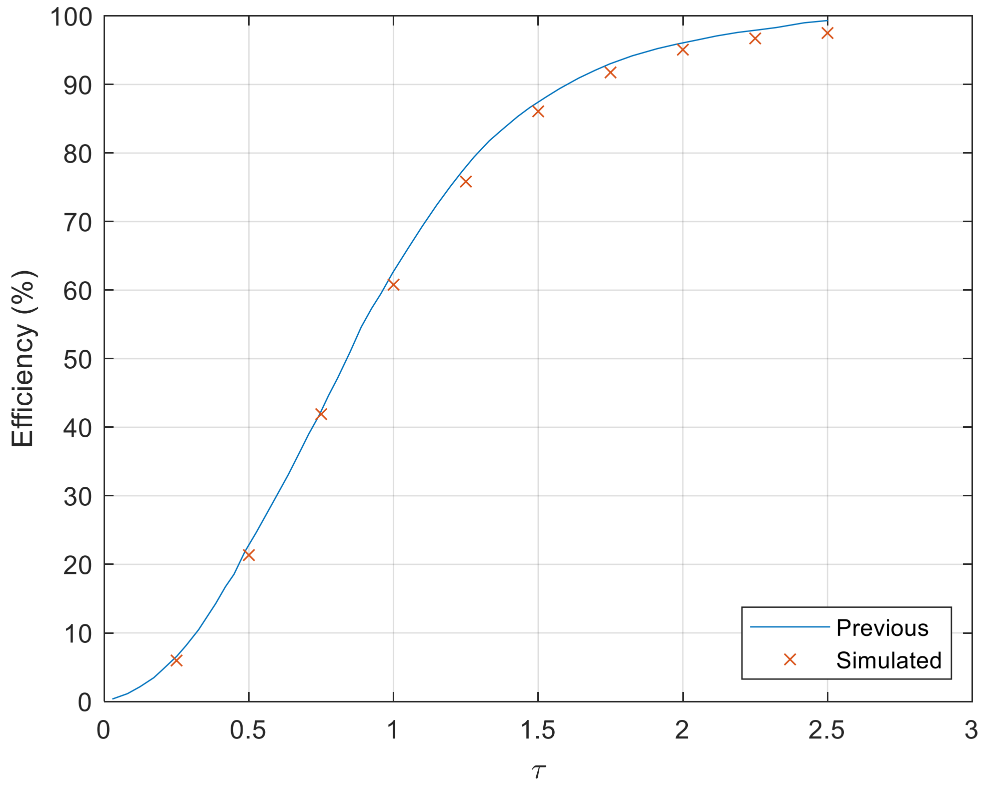


Fig. 13 – Comparison between previous efficiency data and simulation data created by this software

6. CONCLUSION

The MATLAB tool presented here could be of use in calculating the radiating fields used in wireless power transfer, and optimizing the source distribution for maximum efficiency. Its operation is derived from the Huygens-Fresnel Principle, with optimization performed using phase focusing and a golden sector search algorithm. Its source code is documented in Appendix 1, with an alternative version without a GUI described in Appendix 2.

REFERENCES

[1] R.R.A Syms and J.R. Cozens, *Optical Guided Waves and Devices*, New York: McGraw-Hill, 1992, Section 4.4. [Online]. Available: https://pdfs.semanticscholar.org/0f76/e2ac9a786c4e9fdfc4ca2e601ab9507ec9af.pdf

[2] S. Silver, Ed., *Microwave Antenna Theory and Design*. London: McGraw-Hill, 2008.

[3] S. Hong, “Phased array excitations for efficient near-field wireless power transmission,” M.S. thesis, Dept. of Electrical and Computer Eng., Naval Postgraduate School, Monterey, Sept. 2016. Accessed on: Nov. 15, 2019. [Online]. Available: https://pdfs.semanticscholar.org/2615/cef3e3cac0f724b849512394923ad43fe1af.pdf

[4] A. Massa, G. Oliveri, F. Viani, and P. Rocca, “Array Designs for Long-Distance Wireless Power Transmission: State-of-the-Art and Innovative Solutions,” *IEEE Proc.*, vol. 101, no. 6, pp. 1464-1481, June 2013. [Online]. Available doi: 10.1109/JPROC.2013.2245491

[5] A. Lyon, “Why are normal distributions normal?” *Brit. J. Phil. Sci.*, vol. 65, pp. 621-649, Sept. 2013. Accessed on: Nov. 18, 2019. [Online]. Available: https://aidanlyon.com/normal_distributions.pdf

[6] J. Kiefer, “Sequential minimax search for a maximum,” *Proc. Amer. Math. Soc.*, vol. 4, no. 3, pp. 502-506, Feb. 1953. Accessed on: Nov. 18, 2019. [Online]. Available: https://www.ams.org/journals/proc/1953-004-03/S0002-9939-1953-0055639-3/S0002-9939-1953-0055639-3.pdf

[7] M. Srinivasan, *Design of Antenna Arrays using Windows*, The Shy Bulb, Nov. 4, 2015. Accessed on: Nov. 15, 2019. [Online]. Available: http://theshybulb.com/2015/11/04/antenna-arrays-windows.html

[8] F. J. Harris, “On the use of windows for harmonic analysis with the discrete Fourier transform,” *IEEE Proc.*, vol. 66, no.1, pp. 51-83, Jan. 1978. Accessed on: Nov. 18, 2019. [Online]. Available: http://web.mit.edu/xiphmont/Public/windows.pdf

[9] W. Brown, “Adapting microwave techniques to help solve future energy problems,” *IEEE Trans. Microw. Theory Techn.*, vol. 21, no. 12, pp. 753-763, Dec. 1973. Accessed on: May 7, 2020. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1128128&tag=1

[10]G. Goubau, “Microwave Power Transmission from an orbiting solar power station,” *Journal of Microwave Power*, vol. 5, no. 4, pp. 224-231, 1970. Accessed on: May 7, 2020. [Online]. Available: https://www.tandfonline.com/doi/pdf/10.1080/00222739.1970.11688767

Appendix 1

diff_app_lite.mlapp

```
classdef diff_app_lite < matlab.apps.AppBase


    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        ShapeDropDownLabel              matlab.ui.control.Label
        ShapeDropDown                   matlab.ui.control.DropDown
        FrequencyEditFieldLabel         matlab.ui.control.Label
        FrequencyEditField              matlab.ui.control.NumericEditField
        SizeEditFieldLabel              matlab.ui.control.Label
        SizeEditField                   matlab.ui.control.NumericEditField
        DistanceEditFieldLabel          matlab.ui.control.Label
        DistanceEditField               matlab.ui.control.NumericEditField
        TargetEditFieldLabel            matlab.ui.control.Label
        TargetEditField                 matlab.ui.control.NumericEditField
        ApertureEditFieldLabel          matlab.ui.control.Label
        ApertureEditField               matlab.ui.control.NumericEditField
        ResolutionLabel                 matlab.ui.control.Label
        RunButton                       matlab.ui.control.Button
        MaxAngleEditFieldLabel          matlab.ui.control.Label
        MaxAngleEditField               matlab.ui.control.NumericEditField
        LoadButton                      matlab.ui.control.Button
        SaveButton                      matlab.ui.control.Button
        AmplitudeDistributionEditFieldLabel  matlab.ui.control.Label
        AmplitudeDistributionEditField  matlab.ui.control.EditField
        PhaseDistributionEditFieldLabel  matlab.ui.control.Label
        PhaseDistributionEditField      matlab.ui.control.EditField
        WidthEditField_2Label           matlab.ui.control.Label
        WidthEditField_2                matlab.ui.control.NumericEditField
        LengthEditField_2Label          matlab.ui.control.Label
        LengthEditField_2               matlab.ui.control.NumericEditField
        ReceiverLabel                   matlab.ui.control.Label
        CollectedPowerEditFieldLabel    matlab.ui.control.Label
        CollectedPowerEditField         matlab.ui.control.NumericEditField
        TransmitPowerEditFieldLabel     matlab.ui.control.Label
        TransmitPowerEditField          matlab.ui.control.NumericEditField
        RunningLampLabel                matlab.ui.control.Label
        RunningLamp                     matlab.ui.control.Lamp
        GHzLabel                        matlab.ui.control.Label
        mLabel                          matlab.ui.control.Label
        mLabel_3                        matlab.ui.control.Label
        WLabel                          matlab.ui.control.Label
        ptLabel                         matlab.ui.control.Label
        ptLabel_2                       matlab.ui.control.Label
        degLabel_3                      matlab.ui.control.Label
        mLabel_4                        matlab.ui.control.Label
        mLabel_5                        matlab.ui.control.Label
        WLabel_2                        matlab.ui.control.Label
        AutofocusSwitchLabel            matlab.ui.control.Label
        AutofocusSwitch                 matlab.ui.control.RockerSwitch
        OptimizeSwitchLabel             matlab.ui.control.Label
        OptimizeSwitch                  matlab.ui.control.RockerSwitch
        GratingLobesLampLabel           matlab.ui.control.Label
        GratingLobesLamp                matlab.ui.control.Lamp
        PhasedArrayCheckBox             matlab.ui.control.CheckBox
        AlphaEditFieldLabel             matlab.ui.control.Label
        AlphaEditField                  matlab.ui.control.NumericEditField
        FullDepthPlotSwitchLabel        matlab.ui.control.Label
        FullDepthPlotSwitch             matlab.ui.control.RockerSwitch
        NumberofCutsEditFieldLabel      matlab.ui.control.Label
        NumberofCutsEditField           matlab.ui.control.NumericEditField
        MaxDistanceEditFieldLabel       matlab.ui.control.Label
        MaxDistanceEditField            matlab.ui.control.NumericEditField
        ReflectorCheckBox               matlab.ui.control.CheckBox
        SizeEditField_2Label            matlab.ui.control.Label
        SizeEditField_2                 matlab.ui.control.NumericEditField
        mLabel_6                        matlab.ui.control.Label
        mLabel_7                        matlab.ui.control.Label
    end


    % Callbacks that handle component events
    methods (Access = private)


        % Button pushed function: RunButton
        function RunButtonPushed(app, event)
            cd '.\';
            app.RunningLamp.Color = [0 1 0];
            drawnow;
            diffraction_calculator_autofocus_lite(app);
            app.RunningLamp.Color = [1 0 0];
            drawnow;
        end


        % Button pushed function: SaveButton
        function SaveButtonPushed(app, event)
            params =
struct('shape',app.ShapeDropDown.Value,'freq',app.FrequencyEditField.Value,...

'size',app.SizeEditField.Value,'array',app.PhasedArrayCheckBox.Value,...

'distance',app.DistanceEditField.Value,'reflector',app.ReflectorCheckBox.Value,...
'reflectSize',app.SizeEditField_2.Value,'depthplot',app.FullDepthPlotSwitch.Value,...

'planenum',app.NumberofCutsEditField.Value,'maxdist',app.MaxDistanceEditField.Value,..
.

'target',app.TargetEditField.Value,'aper',app.ApertureEditField.Value,...

'angle',app.MaxAngleEditField.Value,'ampdist',app.AmplitudeDistributionEditField.Value
,...

'sigma',app.AlphaEditField.Value,'phasedist',app.PhaseDistributionEditField.Value,...

'recvlen',app.LengthEditField_2.Value,'recvwid',app.WidthEditField_2.Value,...

'recvpow',app.CollectedPowerEditField.Value,'ptrans',app.TransmitPowerEditField.Value)
;

            [savefile,savepath] = uiputfile;
            savefile = savefile(1:end-7);
            save(strcat(savepath,savefile,'.mat'),"params");
        end


        % Button pushed function: LoadButton
        function LoadButtonPushed(app, event)
            [loadfile,loadpath] = uigetfile;
            load(strcat(loadpath,loadfile));

            app.ShapeDropDown.Value = params.shape;
            app.FrequencyEditField.Value = params.freq;
            app.SizeEditField.Value = params.size;
            app.PhasedArrayCheckBox.Value = params.array;
            app.DistanceEditField.Value = params.distance;
            app.ReflectorCheckBox.Value = params.reflector;
            app.SizeEditField_2.Value = params.reflectSize;
            app.FullDepthPlotSwitch.Value = params.depthplot;
            app.NumberofCutsEditField.Value = params.planenum;
            app.MaxDistanceEditField.Value = params.maxdist;
            app.TargetEditField.Value = params.target;
            app.ApertureEditField.Value = params.aper;
            app.MaxAngleEditField.Value = params.angle;
            app.AmplitudeDistributionEditField.Value = params.ampdist;
            app.AlphaEditField.Value = params.sigma;
            app.PhaseDistributionEditField.Value = params.phasedist;
            app.LengthEditField_2.Value = params.recvlen;
            app.WidthEditField_2.Value = params.recvwid;
            app.CollectedPowerEditField.Value = params.recvpow;
            app.TransmitPowerEditField.Value = params.ptrans;
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 878 601];
            app.UIFigure.Name = 'UI Figure';

            % Create ShapeDropDownLabel
            app.ShapeDropDownLabel = uilabel(app.UIFigure);
            app.ShapeDropDownLabel.HorizontalAlignment = 'right';
            app.ShapeDropDownLabel.Position = [62 548 40 22];
            app.ShapeDropDownLabel.Text = 'Shape';

            % Create ShapeDropDown
            app.ShapeDropDown = uidropdown(app.UIFigure);
            app.ShapeDropDown.Items = {'Circle', 'Square'};
            app.ShapeDropDown.Position = [117 548 100 22];
            app.ShapeDropDown.Value = 'Circle';

            % Create FrequencyEditFieldLabel
            app.FrequencyEditFieldLabel = uilabel(app.UIFigure);
            app.FrequencyEditFieldLabel.HorizontalAlignment = 'right';
            app.FrequencyEditFieldLabel.Position = [40 487 62 22];
            app.FrequencyEditFieldLabel.Text = 'Frequency';

            % Create FrequencyEditField
            app.FrequencyEditField = uieditfield(app.UIFigure, 'numeric');
            app.FrequencyEditField.Position = [117 487 100 22];

            % Create SizeEditFieldLabel
            app.SizeEditFieldLabel = uilabel(app.UIFigure);
            app.SizeEditFieldLabel.HorizontalAlignment = 'right';
            app.SizeEditFieldLabel.Position = [73 446 29 22];
            app.SizeEditFieldLabel.Text = 'Size';
```

```
% Create SizeEditField
app.SizeEditField = uieditfield(app.UIFigure, 'numeric');
app.SizeEditField.Position = [117 446 100 22];


% Create DistanceEditFieldLabel
app.DistanceEditFieldLabel = uilabel(app.UIFigure);
app.DistanceEditFieldLabel.HorizontalAlignment = 'right';
app.DistanceEditFieldLabel.Position = [50 362 52 22];
app.DistanceEditFieldLabel.Text = 'Distance';


% Create DistanceEditField
app.DistanceEditField = uieditfield(app.UIFigure, 'numeric');
app.DistanceEditField.Position = [117 362 100 22];


% Create TargetEditFieldLabel
app.TargetEditFieldLabel = uilabel(app.UIFigure);
app.TargetEditFieldLabel.HorizontalAlignment = 'right';
app.TargetEditFieldLabel.Position = [509 506 39 22];
app.TargetEditFieldLabel.Text = 'Target';


% Create TargetEditField
app.TargetEditField = uieditfield(app.UIFigure, 'numeric');
app.TargetEditField.Position = [563 506 100 22];
app.TargetEditField.Value = 400;


% Create ApertureEditFieldLabel
app.ApertureEditFieldLabel = uilabel(app.UIFigure);
app.ApertureEditFieldLabel.HorizontalAlignment = 'right';
app.ApertureEditFieldLabel.Position = [496 527 52 22];
app.ApertureEditFieldLabel.Text = 'Aperture';


% Create ApertureEditField
app.ApertureEditField = uieditfield(app.UIFigure, 'numeric');
app.ApertureEditField.Position = [563 527 100 22];
app.ApertureEditField.Value = 30;


% Create ResolutionLabel
app.ResolutionLabel = uilabel(app.UIFigure);
app.ResolutionLabel.Position = [569 558 62 22];
app.ResolutionLabel.Text = 'Resolution';


% Create RunButton
```

```
            app.RunButton = uibutton(app.UIFigure, 'push');
            app.RunButton.ButtonPushedFcn = createCallbackFcn(app, @RunButtonPushed,
true);
            app.RunButton.Position = [745 35 100 22];
            app.RunButton.Text = 'Run';

            % Create MaxAngleEditFieldLabel
            app.MaxAngleEditFieldLabel = uilabel(app.UIFigure);
            app.MaxAngleEditFieldLabel.HorizontalAlignment = 'right';
            app.MaxAngleEditFieldLabel.Position = [486 434 62 22];
            app.MaxAngleEditFieldLabel.Text = 'Max Angle';

            % Create MaxAngleEditField
            app.MaxAngleEditField = uieditfield(app.UIFigure, 'numeric');
            app.MaxAngleEditField.Position = [563 434 100 22];
            app.MaxAngleEditField.Value = 0.5;

            % Create LoadButton
            app.LoadButton = uibutton(app.UIFigure, 'push');
            app.LoadButton.ButtonPushedFcn = createCallbackFcn(app, @LoadButtonPushed,
true);
            app.LoadButton.Position = [618 35 100 22];
            app.LoadButton.Text = 'Load';

            % Create SaveButton
            app.SaveButton = uibutton(app.UIFigure, 'push');
            app.SaveButton.ButtonPushedFcn = createCallbackFcn(app, @SaveButtonPushed,
true);
            app.SaveButton.Position = [618 56 100 22];
            app.SaveButton.Text = 'Save';

            % Create AmplitudeDistributionEditFieldLabel
            app.AmplitudeDistributionEditFieldLabel = uilabel(app.UIFigure);
            app.AmplitudeDistributionEditFieldLabel.HorizontalAlignment = 'right';
            app.AmplitudeDistributionEditFieldLabel.Position = [58 174 122 22];
            app.AmplitudeDistributionEditFieldLabel.Text = 'Amplitude Distribution';

            % Create AmplitudeDistributionEditField
            app.AmplitudeDistributionEditField = uieditfield(app.UIFigure, 'text');
            app.AmplitudeDistributionEditField.Position = [195 174 100 22];

            % Create PhaseDistributionEditFieldLabel
            app.PhaseDistributionEditFieldLabel = uilabel(app.UIFigure);
            app.PhaseDistributionEditFieldLabel.HorizontalAlignment = 'right';
            app.PhaseDistributionEditFieldLabel.Position = [77 153 103 22];
            app.PhaseDistributionEditFieldLabel.Text = 'Phase Distribution';


            % Create PhaseDistributionEditField
            app.PhaseDistributionEditField = uieditfield(app.UIFigure, 'text');
            app.PhaseDistributionEditField.Position = [195 153 100 22];


            % Create WidthEditField_2Label
            app.WidthEditField_2Label = uilabel(app.UIFigure);
            app.WidthEditField_2Label.HorizontalAlignment = 'right';
            app.WidthEditField_2Label.Position = [512 330 36 22];
            app.WidthEditField_2Label.Text = 'Width';


            % Create WidthEditField_2
            app.WidthEditField_2 = uieditfield(app.UIFigure, 'numeric');
            app.WidthEditField_2.Position = [563 330 100 22];


            % Create LengthEditField_2Label
            app.LengthEditField_2Label = uilabel(app.UIFigure);
            app.LengthEditField_2Label.HorizontalAlignment = 'right';
            app.LengthEditField_2Label.Position = [506 351 42 22];
            app.LengthEditField_2Label.Text = 'Length';


            % Create LengthEditField_2
            app.LengthEditField_2 = uieditfield(app.UIFigure, 'numeric');
            app.LengthEditField_2.Position = [563 351 100 22];


            % Create ReceiverLabel
            app.ReceiverLabel = uilabel(app.UIFigure);
            app.ReceiverLabel.Position = [581 385 53 22];
            app.ReceiverLabel.Text = 'Receiver';


            % Create CollectedPowerEditFieldLabel
            app.CollectedPowerEditFieldLabel = uilabel(app.UIFigure);
            app.CollectedPowerEditFieldLabel.HorizontalAlignment = 'right';
            app.CollectedPowerEditFieldLabel.Position = [455 234 93 22];
            app.CollectedPowerEditFieldLabel.Text = 'Collected Power';


            % Create CollectedPowerEditField
            app.CollectedPowerEditField = uieditfield(app.UIFigure, 'numeric');
            app.CollectedPowerEditField.Position = [563 234 100 22];
```

```
% Create TransmitPowerEditFieldLabel
app.TransmitPowerEditFieldLabel = uilabel(app.UIFigure);
app.TransmitPowerEditFieldLabel.HorizontalAlignment = 'right';
app.TransmitPowerEditFieldLabel.Position = [15 42 89 22];
app.TransmitPowerEditFieldLabel.Text = 'Transmit Power';

% Create TransmitPowerEditField
app.TransmitPowerEditField = uieditfield(app.UIFigure, 'numeric');
app.TransmitPowerEditField.Position = [119 42 100 22];

% Create RunningLampLabel
app.RunningLampLabel = uilabel(app.UIFigure);
app.RunningLampLabel.HorizontalAlignment = 'right';
app.RunningLampLabel.Position = [746 537 50 22];
app.RunningLampLabel.Text = 'Running';

% Create RunningLamp
app.RunningLamp = uilamp(app.UIFigure);
app.RunningLamp.Position = [811 537 20 20];
app.RunningLamp.Color = [1 0 0];

% Create GHzLabel
app.GHzLabel = uilabel(app.UIFigure);
app.GHzLabel.Position = [220 487 29 22];
app.GHzLabel.Text = 'GHz';

% Create mLabel
app.mLabel = uilabel(app.UIFigure);
app.mLabel.Position = [223 446 25 22];
app.mLabel.Text = 'm';

% Create mLabel_3
app.mLabel_3 = uilabel(app.UIFigure);
app.mLabel_3.Position = [223 364 25 22];
app.mLabel_3.Text = 'm';

% Create WLabel
app.WLabel = uilabel(app.UIFigure);
app.WLabel.Position = [223 42 25 22];
app.WLabel.Text = 'W';
```

```
% Create ptLabel
app.ptLabel = uilabel(app.UIFigure);
app.ptLabel.Position = [668 527 25 22];
app.ptLabel.Text = 'pt';


% Create ptLabel_2
app.ptLabel_2 = uilabel(app.UIFigure);
app.ptLabel_2.Position = [668 506 25 22];
app.ptLabel_2.Text = 'pt';


% Create degLabel_3
app.degLabel_3 = uilabel(app.UIFigure);
app.degLabel_3.Position = [668 434 26 22];
app.degLabel_3.Text = 'deg';


% Create mLabel_4
app.mLabel_4 = uilabel(app.UIFigure);
app.mLabel_4.Position = [668 351 25 22];
app.mLabel_4.Text = 'm';


% Create mLabel_5
app.mLabel_5 = uilabel(app.UIFigure);
app.mLabel_5.Position = [668 330 25 22];
app.mLabel_5.Text = 'm';


% Create WLabel_2
app.WLabel_2 = uilabel(app.UIFigure);
app.WLabel_2.Position = [668 234 25 22];
app.WLabel_2.Text = 'W';


% Create AutofocusSwitchLabel
app.AutofocusSwitchLabel = uilabel(app.UIFigure);
app.AutofocusSwitchLabel.HorizontalAlignment = 'center';
app.AutofocusSwitchLabel.Position = [318 20 59 22];
app.AutofocusSwitchLabel.Text = 'Autofocus';


% Create AutofocusSwitch
app.AutofocusSwitch = uiswitch(app.UIFigure, 'rocker');
app.AutofocusSwitch.Position = [337 78 20 45];


% Create OptimizeSwitchLabel
app.OptimizeSwitchLabel = uilabel(app.UIFigure);
```

```
            app.OptimizeSwitchLabel.HorizontalAlignment = 'center';
            app.OptimizeSwitchLabel.Position = [404 20 53 22];
            app.OptimizeSwitchLabel.Text = 'Optimize';

            % Create OptimizeSwitch
            app.OptimizeSwitch = uiswitch(app.UIFigure, 'rocker');
            app.OptimizeSwitch.Position = [420 78 20 45];

            % Create GratingLobesLampLabel
            app.GratingLobesLampLabel = uilabel(app.UIFigure);
            app.GratingLobesLampLabel.HorizontalAlignment = 'right';
            app.GratingLobesLampLabel.Position = [360 527 81 22];
            app.GratingLobesLampLabel.Text = 'Grating Lobes';

            % Create GratingLobesLamp
            app.GratingLobesLamp = uilamp(app.UIFigure);
            app.GratingLobesLamp.Position = [456 527 20 20];

            % Create PhasedArrayCheckBox
            app.PhasedArrayCheckBox = uicheckbox(app.UIFigure);
            app.PhasedArrayCheckBox.Text = 'Phased Array';
            app.PhasedArrayCheckBox.Position = [267 487 95 22];

            % Create AlphaEditFieldLabel
            app.AlphaEditFieldLabel = uilabel(app.UIFigure);
            app.AlphaEditFieldLabel.HorizontalAlignment = 'right';
            app.AlphaEditFieldLabel.Position = [140 122 40 22];
            app.AlphaEditFieldLabel.Text = 'Alpha';

            % Create AlphaEditField
            app.AlphaEditField = uieditfield(app.UIFigure, 'numeric');
            app.AlphaEditField.ValueDisplayFormat = '%11.6g';
            app.AlphaEditField.Position = [195 122 100 22];

            % Create FullDepthPlotSwitchLabel
            app.FullDepthPlotSwitchLabel = uilabel(app.UIFigure);
            app.FullDepthPlotSwitchLabel.HorizontalAlignment = 'center';
            app.FullDepthPlotSwitchLabel.Position = [344 300 84 22];
            app.FullDepthPlotSwitchLabel.Text = 'Full Depth Plot';

            % Create FullDepthPlotSwitch
            app.FullDepthPlotSwitch = uiswitch(app.UIFigure, 'rocker');
            app.FullDepthPlotSwitch.Position = [375 358 20 45];


            % Create NumberofCutsEditFieldLabel
            app.NumberofCutsEditFieldLabel = uilabel(app.UIFigure);
            app.NumberofCutsEditFieldLabel.HorizontalAlignment = 'right';
            app.NumberofCutsEditFieldLabel.Position = [223 251 90 22];
            app.NumberofCutsEditFieldLabel.Text = 'Number of Cuts';


            % Create NumberofCutsEditField
            app.NumberofCutsEditField = uieditfield(app.UIFigure, 'numeric');
            app.NumberofCutsEditField.Position = [328 251 100 22];


            % Create MaxDistanceEditFieldLabel
            app.MaxDistanceEditFieldLabel = uilabel(app.UIFigure);
            app.MaxDistanceEditFieldLabel.HorizontalAlignment = 'right';
            app.MaxDistanceEditFieldLabel.Position = [235 272 78 22];
            app.MaxDistanceEditFieldLabel.Text = 'Max Distance';


            % Create MaxDistanceEditField
            app.MaxDistanceEditField = uieditfield(app.UIFigure, 'numeric');
            app.MaxDistanceEditField.Position = [328 272 100 22];


            % Create ReflectorCheckBox
            app.ReflectorCheckBox = uicheckbox(app.UIFigure);
            app.ReflectorCheckBox.Text = 'Reflector';
            app.ReflectorCheckBox.Position = [94 321 70 22];


            % Create SizeEditField_2Label
            app.SizeEditField_2Label = uilabel(app.UIFigure);
            app.SizeEditField_2Label.HorizontalAlignment = 'right';
            app.SizeEditField_2Label.Position = [73 290 29 22];
            app.SizeEditField_2Label.Text = 'Size';


            % Create SizeEditField_2
            app.SizeEditField_2 = uieditfield(app.UIFigure, 'numeric');
            app.SizeEditField_2.Position = [117 290 100 22];


            % Create mLabel_6
            app.mLabel_6 = uilabel(app.UIFigure);
            app.mLabel_6.Position = [223 290 25 22];
            app.mLabel_6.Text = 'm';
            % Create mLabel_7
            app.mLabel_7 = uilabel(app.UIFigure);
            app.mLabel_7.Position = [434 272 25 22];
            app.mLabel_7.Text = 'm';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = diff_app_lite

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

diffraction_calculator_autofocus_lite.m

```
function diffraction_calculator(uihandle)

c = 299792458;

%Read in inputs and map source points

depthPlotOn = strcmp(uihandle.FullDepthPlotSwitch.Value,'On');
planeNum = uihandle.NumberofCutsEditField.Value;

if ~depthPlotOn
    planeNum = 1;
end

inputVals = inputVars_lite(uihandle);
inputVals.tiltX = 0;
inputVals.tiltY = 0;
lambda = c/inputVals.frequency;
aper = aperture(inputVals);
aper.mapAreas(inputVals);

if depthPlotOn
    farVals = copy(inputVals);
    farVals.distance = uihandle.MaxDistanceEditField.Value;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Update UI lamps

if ((aper.spaceX>lambda/2)||(aper.spaceY>lambda/2))
    uihandle.GratingLobesLamp.Color = [1 0 0];
    drawnow;
else
    uihandle.GratingLobesLamp.Color = [0 1 0];
    drawnow;
end

%Illustrate source point locations

figure;

scatter(aper.coords(:,1),aper.coords(:,2),'.');
pbaspect([1 1 1]);
xlabel('Position (m)');

%Map POI points

theplane = poi(inputVals);

if depthPlotOn
    bigplane = poi(farVals);
    biggestPlane = bigplane.planeSize;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Amplitude distribution optimization

if ((inputVals.optimize)&&(inputVals.recvLength~=0)&&(inputVals.recvWidth~=0))
    alf = goldenSearchKaiser(inputVals,aper,theplane);
    uihandle.AlphaEditField.Value = alf;
```

```
else
    uihandle.AlphaEditField.Value = 0;
end

%Calculation of field based on source amplitude distribution

aper.findPhasors(inputVals);
theplane.findField(inputVals,aper);
theplane.analyze(inputVals);

if depthPlotOn
    %Create and solve additional solution planes to generate depth plot
    distVec = linspace(0,farVals.distance,planeNum);
    distVec = [distVec(2)./8 distVec(2:end)];
    if isempty(find(distVec==inputVals.distance))
        distVec = [distVec inputVals.distance];
        distVec = sort(distVec);
    end

    newInput = copy(inputVals);
    planeVec = theplane;
    slices = theplane.power(floor(newInput.planeRes./2),:);

    for ii=1:size(distVec,2)
        newInput.distance = distVec(1,ii);
        newInput.viewField = atand((biggestPlane-
newInput.maxY)./2./newInput.distance);
        planeVec = [planeVec; poi(newInput)];
        planeVec(end,1).findField(newInput,aper);
        planeVec(end,1).analyze(newInput);
        slices = [slices; planeVec(end,1).power(floor(newInput.planeRes./2),:)];
    end

    planeVec = [planeVec; planeVec(1,1)];
    planeVec(1,:) = [];
    slices(1,:) = [];

end

%Illustrate source point amplitudes

figure;

amps = scatter3(aper.coords(:,1),aper.coords(:,2),abs(aper.ampDensity),'.');
xlabel('Position (m)');
zlabel('Amplitude Density (V/m^2)');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Take axial slice of calculated field

beamvecx = theplane.powerdB(floor(inputVals.planeRes/2),:);

%Plot slice

figure;

plot(theplane.zeta,beamvecx);
title('Incident Power Density (x-axis)');
xlabel('Position (m)');
ylabel('Power Density (dBm/m^2)');

%Plot field
```

```
figure;

result = pcolor(theplane.zeta,theplane.eta,theplane.powerdB);
result.LineStyle = 'none';
pbaspect([1 1 1]);
title('Incident Power Density');
xlabel('Position (m)');
bar = colorbar;
title(bar,'dBm/m^2');
ax = gca;
ax.TickDir = 'both';
bar.TickDirection = 'both';

%Return values to UI

uihandle.AmplitudeDistributionEditField.Value = inputVals.ampDistX;
uihandle.CollectedPowerEditField.Value = theplane.recvPower;

%Generate depth plot, if applicable

if depthPlotOn
    depthinterp = interp1(distVec,slices,linspace(0,farVals.distance,planeNum.*10));
    depthdB = 10.*log10(depthinterp);

    figure;
    theplot = surf(linspace(-
biggestPlane./2,biggestPlane./2,inputVals.planeRes),linspace(0,farVals.distance,planeN
um.*10),depthdB);
    theplot.LineStyle = 'none';
    view(0,90);
    daspect([1 10 6.9846]);

    xlabel('Cross-range (m)');
    ylabel('Range (m)');

    bar = colorbar;
    title(bar,'dBm/m^2');
    ax = gca;
    ax.TickDir = 'both';
    bar.TickDirection = 'both';

end

% [1]   XY Plane Triangle Area Calculator. Accessed on: Mar. 26, 2020. [Online].
Available: https://ncalculators.com/geometry/triangle-area-by-3-points.htm (accessed
March 26, 2020).
% [2]   S. Hong, "Phased array excitations for efficient near-field wireless power
transmission," M.S. thesis, Dept. of Electrical and Computer Eng., Naval Postgraduate
School, Monterey, Sept. 2016. Accessed on: Nov. 15, 2019. [Online]. Available:
https://pdfs.semanticscholar.org/2615/cef3e3cac0f724b849512394923ad43fe1af.pdf
% [3]   R.R.A Syms and J.R. Cozens, Optical Guided Waves and Devices, New York:
McGraw-Hill, 1992, Section 4.4. [Online]. Available:
https://pdfs.semanticscholar.org/0f76/e2ac9a786c4e9fdfc4ca2e601ab9507ec9af.pdf
% [4]   S. Silver, Ed., Microwave Antenna Theory and Design. London: McGraw-Hill,
2008.
% [5]   J. Kiefer, "Sequential minimax search for a maximum," Proc. Amer. Math. Soc.,
vol. 4, no. 3, pp. 502-506, Feb. 1953. Accessed on: Nov. 18, 2019. [Online].
Available: https://www.ams.org/journals/proc/1953-004-03/S0002-9939-1953-0055639-
3/S0002-9939-1953-0055639-3.pdf
% [6]   F. J. Harris, "On the use of windows for harmonic analysis with the discrete
Fourier transform," IEEE Proc., vol. 66, no.1, pp. 51-83, Jan. 1978. Accessed on: Nov.
18, 2019. [Online]. Available: http://web.mit.edu/xiphmont/Public/windows.pdf
```

inputVars_lite.m

```
classdef inputVars_lite < matlab.mixin.Copyable
    % Class definition for struct that holds GUI input information

    properties
        shape

        maxX
        maxY

        phasedArray

        frequency
        distance
        viewField

        reflector
        reflectSize

        apertureRes
        planeRes

        Ptransmit
        ampDistX
        ampDistY
        ampPW
        ampCells
        phaseDistX
        phaseDistY
        phasePW
        phaseCells
        tiltX
        tiltY

        recvLength
        recvWidth

        autofocus
        optimize
    end

    methods
        function obj = inputVars_lite(uihandle)
            % Constructor

            obj.shape = uihandle.ShapeDropDown.Value;

            obj.maxY = uihandle.SizeEditField.Value;

            obj.phasedArray = uihandle.PhasedArrayCheckBox.Value;

            obj.frequency = uihandle.FrequencyEditField.Value.*1e9;
            obj.distance = uihandle.DistanceEditField.Value;
            obj.viewField = uihandle.MaxAngleEditField.Value;

            obj.reflector = uihandle.ReflectorCheckBox.Value;
            obj.reflectSize = uihandle.SizeEditField_2.Value;

            obj.apertureRes = uihandle.ApertureEditField.Value;
            obj.planeRes = uihandle.TargetEditField.Value;

            obj.Ptransmit = uihandle.TransmitPowerEditField.Value;
            obj.ampDistX = uihandle.AmplitudeDistributionEditField.Value;
            obj.ampDistY = '';
            obj.phaseDistX = uihandle.PhaseDistributionEditField.Value;
            obj.phaseDistY = '';

            obj.recvLength = uihandle.LengthEditField_2.Value;
            obj.recvWidth = uihandle.WidthEditField_2.Value;

            obj.autofocus = strcmp(uihandle.AutofocusSwitch.Value,'On');
            obj.optimize = strcmp(uihandle.OptimizeSwitch.Value,'On');

            obj.ampPW = false;
        end
    end

end
```

aperture.m

```
classdef aperture < handle

    properties
        elemNum
        coords
        powerDist
        areas
        spaceX
        spaceY

        power
        powerDensity
        ampDensity
        phases
        rayz
        phasors
    end

    methods
        function obj = aperture(inputObject)
            %Constuctor
            if (strcmp(inputObject.shape,'Circle'))
                d = inputObject.maxY;
                r = d/2;
            end

            if (strcmp(inputObject.shape,'Square'))
                side = inputObject.maxY;
                d = side;
            end

            %Set up x-axis
            x = linspace(-d/2,d/2,inputObject.apertureRes);
            %%%%%%%%%%%%%%%%%%% PHASED ARRAY %%%%%%%%%%%%%%%%
            %Adjust axis for phased array
            if inputObject.phasedArray
                spacing = diff(x);
                x = linspace(-d/2+.5*spacing(1),d/2-
.5*spacing(1),inputObject.apertureRes);
            end
            %%%%%%%%%%%%%%%%%%% /PHASED ARRAY %%%%%%%%%%%%%%%

            [meshx,meshy] = meshgrid(x,x');

            %Create square grid
            mesh = cat(3,meshx,meshy);
            obj.spaceX = mesh(1,2,1)-mesh(1,1,1);
            obj.spaceY = mesh(2,1,2)-mesh(1,1,2);

            %Convert points to list format
            obj.coords = reshape(mesh,[],2);

            if (strcmp(inputObject.shape,'Circle'))
                %Remove corners
                ii = 1;
                while ii<=size(obj.coords,1)
                    if ((obj.coords(ii,1)^2+obj.coords(ii,2)^2)>r^2)
                        obj.coords(ii,:) = [];
                    else
                        ii = ii+1;
                    end
                end

                %Add outer curve to better define shape
                if ~inputObject.phasedArray
                    for ii=1:size(x,2)
                        if (x(1,ii)>-r)&&(x(1,ii)<r)
                            obj.coords(end+1,:) = [x(1,ii) sqrt(r.^2-x(1,ii).^2)];
                            obj.coords(end+1,:) = [x(1,ii) -sqrt(r.^2-x(1,ii).^2)];
                        end
                    end

                    for ii=1:size(x,2)
                        if (x(1,ii)>-r)&&(x(1,ii)<r)
                            obj.coords(end+1,:) = [sqrt(r.^2-x(1,ii).^2) x(1,ii)];
                            obj.coords(end+1,:) = [-sqrt(r.^2-x(1,ii).^2) x(1,ii)];
                        end
                    end
                end
            end

            if (strcmp(inputObject.shape,'Square'))
                %Square grid is sufficient
            end

            obj.elemNum = size(obj.coords,1);

        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        function mapAreas(obj,inputObject)
            %Assign represented area to each source point

            %Triangulate
            triangles = delaunayTriangulation(obj.coords(:,1),obj.coords(:,2));

            trilist = triangles.ConnectivityList;
            areas = zeros(size(triangles,1),1);

            %Find area of each triangle
            for ii=1:size(trilist,1)
                for jj=1:size(trilist,2)
                    id = trilist(ii,jj);
                    trilist(ii,jj,1) = triangles.Points(id,1);
                    trilist(ii,jj,2) = triangles.Points(id,2);
                end

                areas(ii) = .5.*abs((trilist(ii,2,1)-
trilist(ii,1,1)).*(trilist(ii,3,2)-trilist(ii,1,2)) - (trilist(ii,3,1)-
trilist(ii,1,1)).*(trilist(ii,2,2)-(trilist(ii,1,2))));     % [1]
            end

            attached = vertexAttachments(triangles);
            coverage = zeros(size(triangles.Points,1),1);

            %Approximate area represented by each point
            for ii=1:size(attached,1)
                sumvar = 0;

                for jj=1:size(attached{ii},2)
                    sumvar = sumvar+areas(attached{ii}(jj));
                end
                coverage(ii) = sumvar/3;
            end

            obj.coords = [];
            obj.coords(:,1) = triangles.Points(:,1);
            obj.coords(:,2) = triangles.Points(:,2);
            obj.areas = coverage;
            obj.elemNum = size(obj.coords,1);
            %%%%%%%%%%%%%%%%%%% PHASED ARRAY %%%%%%%%%%%%%%%%%%%
            if inputObject.phasedArray
                obj.areas = ones(obj.elemNum,1).*sum(coverage,1)./obj.elemNum;
            end
            %%%%%%%%%%%%%%%%%%% /PHASED ARRAY %%%%%%%%%%%%%%%%%%
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        function findPhasors(obj,inputObject)
            %Set source point amplitude and phase

            c = 299792458;

            %Set default for blank amplitude input
            if (strcmp(inputObject.ampDistX,''))
                inputObject.ampDistX = '1';
            end

            if (strcmp(inputObject.ampDistY,''))
                inputObject.ampDistY = '1';
            end

            %Attach input amplitude distributions to functions
            afx = @(x) eval(inputObject.ampDistX);
            afy = @(x) eval(inputObject.ampDistY);
            %Apply amplitude functions to source points
            if
(strcmp(inputObject.shape,'Circle')||strcmp(inputObject.shape,'Square'))
                for ii=1:obj.elemNum
                    radius = sqrt(obj.coords(ii,1).^2+obj.coords(ii,2).^2);
                    obj.powerDist(ii,1) = afx(radius).^2;
                end
            end

            %Renormalize for correct transmitted power
            obj.power = obj.powerDist.*obj.areas;
            scale = inputObject.Ptransmit/sum(obj.power,1);
            obj.power = obj.power.*scale;

            obj.powerDensity = obj.power./obj.areas;
            obj.ampDensity = sqrt(obj.powerDensity.*120.*pi);

            %Remove power from blocked source points
            if inputObject.reflector
                if (inputObject.shape=='Circle')
                    for ii=1:obj.elemNum
                        if
(sqrt(obj.coords(ii,1).^2+obj.coords(ii,2).^2)<(inputObject.reflectSize./2))
                            obj.power(ii,1) = 0;
                            obj.powerDensity(ii,1) = 0;
                            obj.ampDensity(ii,1) = 0;
                        end
                    end
                else
```

```
                    for ii=1:obj.elemNum
                        if
((abs(obj.coords(ii,1))<(inputObject.reflectSize./2))&&((abs(obj.coords(ii,2))<(inputO
bject.reflectSize./2))))
                            obj.power(ii,1) = 0;
                            obj.powerDensity(ii,1) = 0;
                            obj.ampDensity(ii,1) = 0;
                        end
                    end
                end
            end

            %Set default for blank phase input
            if (strcmp(inputObject.phaseDistX,''))
                inputObject.phaseDistX = '0';
            end

            if (strcmp(inputObject.phaseDistY,''))
                inputObject.phaseDistY = '0';
            end

            z = inputObject.distance;
            lambda = c/inputObject.frequency;

            if inputObject.autofocus
                %Get phases and direction of propagation for autofocused source
                rhos = sqrt(obj.coords(:,1).^2+obj.coords(:,2).^2+z^2);
                phases = (rhos-z).*2.*pi./lambda;    % [2]
                obj.rayz = sqrt(1-
(obj.coords(:,1).^2+obj.coords(:,2).^2)./(obj.coords(:,1).^2+obj.coords(:,2).^2+z^2));
%[silver]
                obj.phases = exp(j.*phases);
            else
                %Apply input phase distributions
                px = @(x) eval(inputObject.phaseDistX);
                py = @(x) eval(inputObject.phaseDistY);
                if
(strcmp(inputObject.shape,'Circle')||strcmp(inputObject.shape,'Square'))
                    for ii=1:obj.elemNum
                        radius = sqrt(obj.coords(ii,1).^2+obj.coords(ii,2).^2);
                        obj.phases(ii,1) = exp(j.*px(radius));
                    end
                end

                %Get direction of propagation for arbitrary phase distribution
                for ii=1:obj.elemNum
                    %Create test points near every source to find phase slope
                    deltax = obj.spaceX./10;
                    deltay = obj.spaceY./10;

                    x = obj.coords(ii,1);
                    y = obj.coords(ii,2);

                    testpoints = [x-deltax y; x+deltax y; x y-deltay; x y+deltay];
                    testrad = sqrt(testpoints(:,1).^2+testpoints(:,2).^2);
                    for jj=1:4
                        testphases(jj,1) = lambda./(2.*pi).*px(testrad(jj,1));
                    end

                    %Find phase slope in x- and y-directions
                    slopes = [testphases(2)-obj.phases(ii,1); obj.phases(ii,1)-
testphases(1);...
```

```
                        testphases(4)-obj.phases(ii,1); obj.phases(ii,1)-
testphases(3)];
                    slopes = slopes./[deltax; deltax; deltay; deltay];
                    slopesavg = [(slopes(1)+slopes(2))./2 (slopes(3)+slopes(4))./2];

                    obj.rayz(ii,1) = sqrt(1-slopesavg(1).^2-slopesavg(2).^2);
                end

            end

            %Set final source quantities
            obj.phasors = obj.ampDensity.*obj.phases.*obj.areas;    % [3]
        end

    end

end
```

poi.m

```
classdef poi < handle

    properties
        planeSize
        zeta
        eta
        zetam
        etam
        kappam
        tiltFactor

        field
        fieldAlt
        power
        powerdB
        powerAlt
        powerAltdB

        recvPower
    end

    methods
        function obj = poi(inputObject)
            %Constructor

            %Calculate POI dimensions
            obj.planeSize =
2*inputObject.distance*tan(inputObject.viewField/180*pi/2)+1*inputObject.maxY;

            %Create grid
            obj.zeta = linspace(-
obj.planeSize/2,obj.planeSize/2,inputObject.planeRes);
            obj.eta = obj.zeta;

            obj.tiltFactor =
cos(atan(sqrt(tan(inputObject.tiltX).^2+tan(inputObject.tiltY).^2)));


            [zetamraw,etamraw] = meshgrid(obj.zeta,obj.eta');
            obj.zetam = zetamraw.*cosd(inputObject.tiltX);
            obj.etam = etamraw.*cosd(inputObject.tiltY);
            zx = zetamraw.*sind(inputObject.tiltX);
            zy = etamraw.*sind(inputObject.tiltY);
            obj.kappam = zx+zy+inputObject.distance;
        end

        function findField(obj,inputObject,ap)
            %Calculate POI fields
            c = 299792458;
            lambda = c/inputObject.frequency;

            xs = reshape(ap.coords(:,1),1,1,[]);
            ys = reshape(ap.coords(:,2),1,1,[]);
            ss = reshape(ap.rayz,1,1,[]);
            phasors = reshape(ap.phasors,1,1,[]);

            R = sqrt(obj.zetam.^2 + obj.etam.^2 + obj.kappam.^2);   % [3]
            ra = (xs-obj.zetam).^2./(2*obj.kappam) + (ys-obj.etam).^2./(2*obj.kappam);
% [4]
            theta = atan(sqrt(obj.zetam.^2+obj.etam.^2)./obj.kappam);   % [3]
            ftheta = (cos(theta)+ss)./2;     %[3],[4]
            r = sqrt((obj.zetam-xs).^2 + (obj.etam-ys).^2 + obj.kappam.^2);     % [3]
            fieldraw = 1./(j.*lambda.*r).*ftheta.*phasors.*exp(-j.*2.*pi./lambda.*r);
%[3]
            fieldrawAlt = j./(2.*lambda).*exp(-
j.*2.*pi./lambda.*obj.kappam)./R.*phasors.*exp(-j.*2.*pi./lambda.*ra).*(cos(theta)+1);
% [4]

            obj.field = sum(fieldraw,3);    % [3]
            obj.fieldAlt = sum(fieldrawAlt,3);  %  [4]
            obj.power = abs(obj.field).^2./(120*pi);
            obj.powerAlt = abs(obj.fieldAlt).^2./(120*pi);
        end

        function analyze(obj,inputObject)
            %Process calculated results

            %Convert to power
            [maxrow,rowindex] = max(obj.power);
            [pmax,colindex] = max(max(obj.power));
            %Renormalize to max power
            maxindex = [rowindex(colindex) colindex];
            normscale = 1/pmax;

            [maxrowAlt,rowindexAlt] = max(obj.powerAlt);
            [pmaxAlt,colindexAlt] = max(max(obj.powerAlt));
            maxindexAlt = [rowindexAlt(colindexAlt) colindexAlt];
            normscaleAlt = 1/pmaxAlt;

            powerdB = 10*log10(obj.power.*normscale);
            powerAltdB = 10*log10(obj.powerAlt.*normscaleAlt);

            %Renormalize to 1 mW
            obj.powerdB = 10*log10(obj.power./.001);
            obj.powerAltdB = 10*log10(obj.powerAlt./.001);

            edgepoints = [];
            list = [];

            for ii=1:size(powerdB,1)
                for jj=1:size(powerdB,2)
                    %Determine which points fall within the receiving array
                    if ((obj.zetam(ii,jj)>-
inputObject.recvWidth/2)&&(obj.zetam(ii,jj)<inputObject.recvWidth/2)&&(obj.etam(ii,jj)
>-inputObject.recvLength/2)&&...
                            (obj.etam(ii,jj)<inputObject.recvLength/2))
                        list = [list; obj.zetam(ii,jj) obj.etam(ii,jj)
obj.power(ii,jj).*obj.tiltFactor];
                    end

                end
            end

            %Sum power among points within receiving array
            if (size(list)==[0 0])
                prect = 0;
            else
                prectavg = sum(list(:,3),1)./size(list,1);
                rectarea = (list(end,1)-list(1,1)).*(list(end,2)-list(1,2));
                prect = prectavg.*rectarea;
            end

            obj.recvPower = prect;
        end
```

```
    end

end
```

goldenSearchKaiser.m

```
function alphFinal = goldenSearchKaiser(inputObject,ap,plane)

    %%% Golden sector search process described by [5]
    %%% Kaiser distribution formula described by [6]

    bounds = [.001 10];
    ratio = (1+sqrt(5))/2;
    mid1 = (bounds(2)-bounds(1))/(ratio+1)+bounds(1);
    mid2 = bounds(1)+(bounds(2)-mid1);
    alpha = [bounds(1) mid1 mid2 bounds(2)];
    result = zeros(1,4);
    L = inputObject.maxY*sqrt(2);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %Create the initial four test points
    for ii=1:4
        kaiser = strcat('besseli(0,pi*',num2str(alpha(ii)),'*sqrt(1-
(2*x/',num2str(L),')^2))/besseli(0,pi*',num2str(alpha(ii)),')');    %[6]
        inputObject.ampDistX = kaiser;
        inputObject.ampDistY = kaiser;
        ap.findPhasors(inputObject);
        phasors(:,ii) = ap.phasors;
        plane.findField(inputObject,ap);
        plane.analyze(inputObject);
        result(ii) = plane.recvPower;
    end

    %Find difference between test point results
    jj=1;
    [maximum,maxindex] = max(result);
    maxdiff = max(diff(result));

    %Refine search based on results at least 3 times until convergence criterion is
reached
    while ((jj<=3)||(maxdiff>(maximum*.005)))

        switch maxindex
            case 1
                bounds = [bounds(1)*.5 mid1];
                mid1 = (bounds(2)-bounds(1))/(ratio+1)+bounds(1);
                mid2 = bounds(1)+(bounds(2)-mid1);
                alpha = [bounds(1) mid1 mid2 bounds(2)];
                result = [0 0 0 result(2)];

                for ii=1:3
                    kaiser = strcat('besseli(0,pi*',num2str(alpha(ii)),'*sqrt(1-
(2*x/',num2str(L),')^2))/besseli(0,pi*',num2str(alpha(ii)),')');    %[6]
                    inputObject.ampDistX = kaiser;
                    inputObject.ampDistY = kaiser;
                    ap.findPhasors(inputObject);
                    phasors(:,ii) = ap.phasors;
                    plane.findField(inputObject,ap);
                    plane.analyze(inputObject);
                    result(ii) = plane.recvPower;
                end
            case 2
                bounds = [bounds(1) mid2];
                mid1 = (bounds(2)-bounds(1))/(ratio+1)+bounds(1);
                mid2 = bounds(1)+(bounds(2)-mid1);
                alpha = [bounds(1) mid1 mid2 bounds(2)];
                result = [result(1) 0 result(2) result(3)];

                kaiser = strcat('besseli(0,pi*',num2str(alpha(2)),'*sqrt(1-
(2*x/',num2str(L),')^2))/besseli(0,pi*',num2str(alpha(2)),')');  %[6]
                inputObject.ampDistX = kaiser;
                inputObject.ampDistY = kaiser;
                ap.findPhasors(inputObject);
                phasors(:,2) = ap.phasors;
                plane.findField(inputObject,ap);
                plane.analyze(inputObject);
                result(2) = plane.recvPower;
            case 3
                bounds = [mid1 bounds(2)];
                mid1 = (bounds(2)-bounds(1))/(ratio+1)+bounds(1);
                mid2 = bounds(1)+(bounds(2)-mid1);
                alpha = [bounds(1) mid1 mid2 bounds(2)];
                result = [result(2) result(3) 0 result(4)];

                kaiser = strcat('besseli(0,pi*',num2str(alpha(3)),'*sqrt(1-
(2*x/',num2str(L),')^2))/besseli(0,pi*',num2str(alpha(3)),')');  %[6]
                inputObject.ampDistX = kaiser;
                inputObject.ampDistY = kaiser;
                ap.findPhasors(inputObject);
                phasors(:,3) = ap.phasors;
                plane.findField(inputObject,ap);
                plane.analyze(inputObject);
                result(3) = plane.recvPower;
            case 4
                bounds = [mid2 bounds(2)*2];
                mid1 = (bounds(2)-bounds(1))/(ratio+1)+bounds(1);
                mid2 = bounds(1)+(bounds(2)-mid1);
                alpha = [bounds(1) mid1 mid2 bounds(2)];
                result = [result(3) 0 0 0];

                for ii=2:4
                    kaiser = strcat('besseli(0,pi*',num2str(alpha(ii)),'*sqrt(1-
(2*x/',num2str(L),')^2))/besseli(0,pi*',num2str(alpha(ii)),')');    %[6]
                    inputObject.ampDistX = kaiser;
                    inputObject.ampDistY = kaiser;
                    ap.findPhasors(inputObject);
                    phasors(:,ii) = ap.phasors;
                    plane.findField(inputObject,ap);
                    plane.analyze(inputObject);
                    result(ii) = plane.recvPower;
                end
        end

        [maximum,maxindex] = max(result);
        maxdiff = max(diff(result));
        jj = jj+1;

    end

    %Return golden sector search results
    kaiser = strcat('besseli(0,pi*',num2str(alpha(maxindex)),'*sqrt(1-
(2*x/',num2str(L),')^2))/besseli(0,pi*',num2str(alpha(maxindex)),')');    %[6]
    inputObject.ampDistX = kaiser;
    inputObject.ampDistY = kaiser;
    alphFinal = alpha(maxindex);
end
```

Appendix 2

For compatibility reasons, an alternative version of the software that does not make use of the GUI was also created. Instead of input and output fields, the user interacts directly with program variables using literal constants. Input quantities and their units are specified by inline comments. To launch this version of the code, the user will run ‘diffraction_calculator_noGUI.m’. The struct used for storing input information was modified slightly for the alternate version, and is titled ‘inputVars_noGUI.m’. All other scripts are shared with the main version, so are not shown again here.

diffraction_calculator_noGUI.m

```
clear all

depthPlotOn = true;             %Full Depth Plot
planeNum = 5;                    %Number of Cuts

if ~depthPlotOn
    planeNum = 1;
end

c = 299792458;                   %m/s

%Read in inputs and map source points

inputVals = inputVars_noGUI;

inputVals.shape = 'Square';      %Shape

inputVals.maxY = 2;              %Size (m)

inputVals.phasedArray = true;    %Phased Array

inputVals.frequency = 10.*1e9;   %Frequency (Hz)
inputVals.distance = 50;         %Distance (m)
inputVals.viewField = .75;        %Max Angle (degrees)

inputVals.reflector = false;      %Reflector
inputVals.reflectSize = .5;      %Reflector:Size (m)

inputVals.apertureRes = 135;      %Resolution:Aperture
inputVals.planeRes = 200;        %Resolution:Target

inputVals.Ptransmit = 100;       %Transmit Power (W)
inputVals.ampDistX = '';         %Amplitude Distribution
inputVals.ampDistY = '';         %not used
inputVals.phaseDistX = '';       %Phase Distribution
inputVals.phaseDistY = '';       %not used

inputVals.recvLength = 2;        %Receiver:Length
inputVals.recvWidth = 2;         %Receiver:Width

inputVals.autofocus = true;     %Autofocus
inputVals.optimize = true;      %Optimize

inputVals.tiltX = 0;    %not used
inputVals.tiltY = 0;    %not used

lambda = c/inputVals.frequency;
aper = aperture(inputVals);
aper.mapAreas(inputVals);

if depthPlotOn
    farVals = copy(inputVals);
    farVals.distance = 500;      %Max Distance
end

if ((aper.spaceX>lambda/2)||(aper.spaceY>lambda/2))
    gratingLobes = true;
else
    gratingLobes = false;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Illustrate source point locations

figure;

scatter(aper.coords(:,1),aper.coords(:,2),'.');
pbaspect([1 1 1]);
xlabel('Position (m)');

%Map POI points

theplane = poi(inputVals);

if depthPlotOn
    bigplane = poi(farVals);
    biggestPlane = bigplane.planeSize;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Amplitude distribution optimization

if ((inputVals.optimize)&&(inputVals.recvLength~=0)&&(inputVals.recvWidth~=0))
    alf = goldenSearchKaiser(inputVals,aper,theplane);
end

%Calculation of field based on source amplitude distribution

aper.findPhasors(inputVals);
theplane.findField(inputVals,aper);
theplane.analyze(inputVals);

if depthPlotOn
    %Create and solve additional solution planes to generate depth plot
    distVec = linspace(0,farVals.distance,planeNum);
    distVec = [distVec(2)./8 distVec(2:end)];
    if isempty(find(distVec==inputVals.distance))
        distVec = [distVec inputVals.distance];
        distVec = sort(distVec);
    end
    %distVec = [0:50:300];
    %biggestPlane = poi();
    newInput = copy(inputVals);
    planeVec = theplane;
    slices = theplane.power(floor(newInput.planeRes./2),:);

    for ii=1:size(distVec,2)
        newInput.distance = distVec(1,ii);
        newInput.viewField = atand((biggestPlane-
newInput.maxY)./2./newInput.distance);
        planeVec = [planeVec; poi(newInput)];
        planeVec(end,1).findField(newInput,aper);
        planeVec(end,1).analyze(newInput);
        slices = [slices; planeVec(end,1).power(floor(newInput.planeRes./2),:)];
    end

    planeVec = [planeVec; planeVec(1,1)];
    planeVec(1,:) = [];
    slices(1,:) = [];

end
```

```
%Illustrate source point amplitudes

figure;

amps = scatter3(aper.coords(:,1),aper.coords(:,2),abs(aper.ampDensity),'.');
xlabel('Position (m)');
zlabel('Amplitude (V)');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Take axial slice of calculated field

beamvecx = theplane.powerdB(floor(inputVals.planeRes/2),:);

%Plot slice

figure;

plot(theplane.zeta,beamvecx);
title('Incident Power Density (x-axis)');
xlabel('Position (m)');
ylabel('Power Density (dBm/m^2)');

%Plot field

figure;

result = pcolor(theplane.zeta,theplane.eta,theplane.powerdB);
result.LineStyle = 'none';
pbaspect([1 1 1]);
title('Incident Power Density');
xlabel('Position (m)');
bar = colorbar;
title(bar,'dBm/m^2');
ax = gca;
ax.TickDir = 'both';
bar.TickDirection = 'both';

%Generate depth plot, if applicable

if depthPlotOn
    depthinterp = interp1(distVec,slices,linspace(0,farVals.distance,planeNum.*10));
    depthdB = 10.*log10(depthinterp);

    figure;
    theplot = surf(linspace(-
biggestPlane./2,biggestPlane./2,inputVals.planeRes),linspace(0,farVals.distance,planeN
um.*10),depthdB);
    theplot.LineStyle = 'none';
    view(0,90);
    daspect([1 10 6.9846]);

    xlabel('Cross-range (m)');
    ylabel('Range (m)');

    bar = colorbar;
    title(bar,'dBm/m^2');
    ax = gca;
    ax.TickDir = 'both';
    bar.TickDirection = 'both';

end
```

inputVars_noGUI.m

```
classdef inputVars_noGUI < matlab.mixin.Copyable

    properties
        shape

        maxX
        maxY

        phasedArray

        frequency
        distance
        viewField

        reflector
        reflectSize

        apertureRes
        planeRes

        Ptransmit
        ampDistX
        ampDistY
        ampPW
        ampCells
        phaseDistX
        phaseDistY
        phasePW
        phaseCells
        tiltX
        tiltY

        recvLength
        recvWidth

        autofocus
        optimize
    end

end
```