Automated Mode-Matching of Gaussian Beams

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science degree in Physics from the College of William and Mary

by

Matthew Argao

Advisor: Eugeniy Mikhailov

Senior Research Coordinator: Henry Krakauer

Date: April 2013

1. Abstract

This project seeks to develop a program which would aid in laboratory experiments requiring precise focusing of Gaussian Beams. In optics, many lasers emit profiles that are approximated by the Gaussian intensity distributions and thus serve as a reliable characterization of the electromagnetic radiation emitted by such lasers. Using Matrix Laboratory (MATLAB), we developed a program which simulates the propagation of a Gaussian beam through a given optical system. Given the input parameters of the Gaussian beam and the desired output parameters, the program automatically chooses what type of lens to use, where to place the lens in the system, determines the accuracy between the simulated and desired output beam and the stability of the given solution.

2. Intro

In the field of optics and laser related physics, Gaussian beams are a well-documented model of electromagnetic radiation whose characteristics of transverse electric field and intensity are approximated by Gaussian functions of the form:

$$E(r,z) = \frac{E_0 w_0}{w(z)} \exp\left(-\frac{r^2}{w^{2(z)}} - ikz - ik\frac{r^2}{2T(z)} + i\zeta(z)\right) \qquad Eq.1.0$$

Where r is the radial distance from the center axis of the beam, z is the axial distance from the beam's narrowest point, also termed waist, k is the wave number, E_0 is the electric field at E(0,0), w(z) is the radius where the field amplitude and intensity drop to 1/e and $1/e^2$ of their axial values respectively, w_0 is the waist size w(0), R(z) is the wave front radius of curvature, and ζ (z) is the Gouy phase shift which is an extra contribution to phase seen in Gaussian beams. w(z) and R(z) are given by following equations:

$$w(z) = w_0\sqrt{1+\left(\frac{z}{z_R}\right)^2} \qquad Eq.1.1$$

$$R(z) = z\left[1+\left(\frac{z_R}{z}\right)^2\right] \qquad Eq.1.2$$

These quantities are more easily visualized in Figure 1.

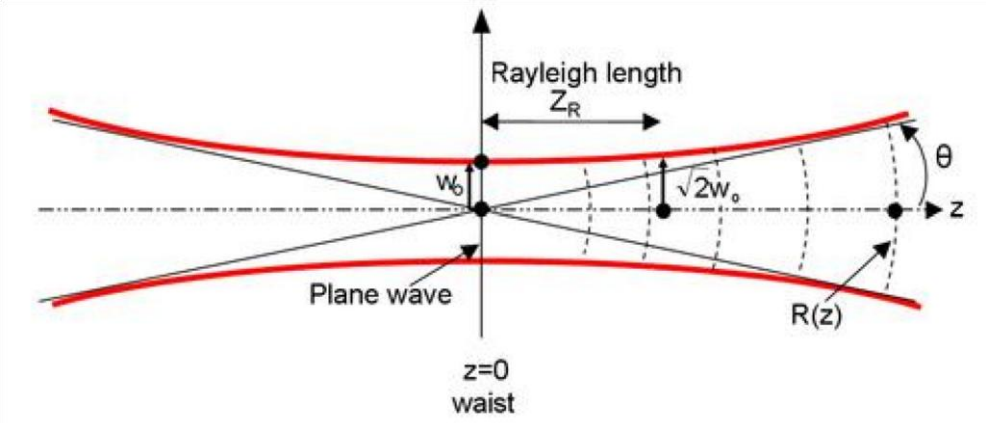

Figure 1. General profile of a Gaussian beam.

Where z_R is the Rayleigh Length, defined as the distance along the propagation direction from w_0 where the area of the beam cross section is doubled. As many lasers emit beams that follow this Gaussian profile. It becomes natural, then, for researchers to wish to produce a desired Gaussian beam that meets certain specifications, determined on a case by case basis of the experiment. For example, when one wishes to inject a given pump laser beam into another laser cavity, it is required that the input beams matches the cavity modes. In other words, one must mode-match the two beams such that the output parameters of the first beam matches the input parameters of the second beam. The simplest way in which we manipulate Gaussian beams is through the use of lenses; lenses transform an input Gaussian beam with certain parameters into another Gaussian beam which consists of different parameters. For our research, the main parameters of Gaussian beams which we focus on are the complex beam parameter, q(z), the beam's waist, w(z), and its radius of curvature, R(z). It becomes advantageous to represent the complex beam parameter in terms of its reciprocal to show the relationship between parameters q(z), w(z) and R(z), given as:

$$\frac{1}{q(z)} = \frac{1}{R(z)} - \frac{i\lambda}{\pi w^2(z)} \qquad Eq.\,1.3$$

Where λ is the wavelength of the beam.

The complex beam parameter readily contains information on both the radius of curvature and waist of a Gaussian beam at any position of its propagation. Thus, the complex beam parameter becomes essential to the simulation of Gaussian beam propagation.

3. Ray Transfer Matrix Analysis

The ABCD transfer matrix is a matrix which characterizes optical elements, such as lenses or free space, with a matrix of the form:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

When determining the optical elements' effects on, say a laser, we consider two reference planes: the input and output planes. Using this expression:

$$\begin{pmatrix} x_2 \\ \theta_2 \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x_1 \\ \theta_1 \end{pmatrix} \qquad Eq.\,2.0$$

We consider the ray, or beam, entering the input plane a distance x_1 from the optical axis (which is taken to coincide with the z-axis) at an angle of θ_1. As the beam travels it eventually crosses the output plane this time with characteristics x_2 and θ_2. An example of this can be seen through the propagation of a beam through free space. The ray transfer matrix of free space is given by:

$$S = \begin{pmatrix} 1 & d \\ 0 & 1 \end{pmatrix}$$

Where d is the distance traveled along the optical axis. Using the expression given previously, we get:

$$\begin{pmatrix} x_2 \\ \theta_2 \end{pmatrix} = \begin{pmatrix} 1 & d \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ \theta_1 \end{pmatrix} \qquad Eq. 2.1$$

Which we can then use to relate the parameters of the input and output rays visualized in Figure 2:

$$x_2 = x_1 + d\theta_1$$
$$\theta_2 = \theta_1$$

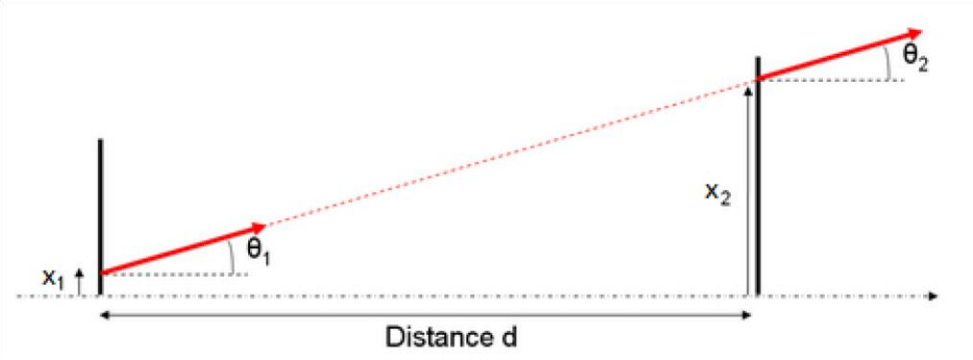

Figure 2. Free-space propagation example.

Similarly we can go through the same geometric model using the ABCD matrix of a thin-lens and free-space given as:

$$thin\ lens, L = \begin{pmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{pmatrix}$$
$$free\ space, S = \begin{pmatrix} 1 & d \\ 0 & 1 \end{pmatrix}$$

Where f is the focal length of the lens with positive values corresponding to convex or converging lenses and d is the distance of free space the beam travels. Similar to the geometric model we can apply this useful matrix formalism and apply it to describe Gaussian beams. Using the previously defined parameter q we apply the following equation:

$$\begin{pmatrix} q_2 \\ 1 \end{pmatrix} = k \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} q_1 \\ 1 \end{pmatrix}$$

Where q_1 and q_2 are the input and output complex beam parameters respectively and k is the normalization constant chosen to ensure that the second component of the ray vector equals one. Upon expansion of the equation we receive:

$$q_2 = (Aq_1 + B)$$
$$1 = (Cq_1 + D)$$

Dividing the first equation by the second to eliminate the normalization constant and taking the reciprocal we finally get the general form:

$$\frac{1}{q_2} = \frac{C + \frac{D}{q_1}}{A + \frac{B}{q_1}} \qquad Eq. 2.2$$

From which we may extract all relevant parameters of the resultant Gaussian beam given the ABCD matrix of the optical element. We use the initial beam parameter, q_i, and the ABCD transfer matrix of the optical system to then find the output beam, q_f.

4. Focusing of Spherical Gaussian Beams

In the thin-lens approximation, diffraction on aperture is neglected and point objects, images and uniform spherical waves whose radii of curvature equal the distance from the point object or image are utilized. For a standard thin lens the relation between object, image and focal length of the lens geometrically are given by the equation:

$$\frac{1}{s} + \frac{1}{s'} = \frac{1}{f} \qquad Eq. 3.0$$

Where s is the object distance, s' is the image distance and f is the focal length of the lens.

In general, the laser of a given output is a spherical Gaussian beam which has a waist where the wave front is planar and the beam's diameter is at a minimum—otherwise known as the waist position. The beam radius w was given previously by Eq. 1.1 and nested within it is the Rayleigh range z_R.

The Rayleigh range can be further represented with the following equation:

$$z_R = \left(\frac{\pi w_0^2}{\lambda}\right) \qquad Eq. 3.1$$

Where λ is the wavelength of the given beam. From the Rayleigh range we can define two distances: near field and far field; where near field corresponds to $z < z_R$ and far field corresponds to $z > z_R$. In regards to the beam's radius of curvature we also previously defined it in Eq. 1.2.

As one can see for the near field ($z << z_R$), R tends to $z_R^2/$. If we take this value at the waist position (z = 0), R tends to positive infinity. In the far field approximation ($z >> z_R$), R simply tends towards z and the wave approximates radiation from a point source centered about the waist.

Provided the input beam waist radius w_0 and the object distance s, we are able to calculate the Rayleigh range (z_R), beam radius (w), and beam radius of curvature (R). For thin lenses, the beam radius is unchanged by the lens, however the radius of curvature is changed by the amount (1/f) as in the geometrical case. Using this information we can now calculate the

output characteristics of a Gaussian beam as it passes through a thin lens. Applying the formulas above we arrive to:

$$\frac{1}{\left(s+\frac{z_R^2}{s-f}\right)}+\frac{1}{s'}=\frac{1}{f}$$

Reorganizing this equation as a function of s/f:

$$\left(\frac{s'}{f}\right)=1+\frac{\left[\frac{s}{f}-1\right]}{\left[\left(\frac{s}{f}\right)-1\right]^2+\left(\frac{z_R}{f}\right)^2}$$

Which is reminiscent of the usual lens formula:

$$\frac{s'}{f}=1+\frac{1}{\left[\left(\frac{s}{f}\right)-1\right]}$$

s' and s in these equations for Gaussian beams now correspond to the final position of the beam waist and the initial position of the beam waist respectively.

Using these formulas we are able to calculate the output image location and waist as the Gaussian beam passes through a thin lens. To apply this to a multi-lens optical system, one simply applies the formula in stages as it passes through each lens—the output parameters of the beam after the first lens become the input parameters for the second lens and so forth for each lens thereafter.

5. The Process of Mode-Matching

In order to properly mode-match there are certain considerations we must take into account. Firstly, the final beam output must match our desired output and the mode-matching algorithm must reflect that. Secondly, we must ensure that solutions we have are physically possible thus two extra conditions are required: lenses are within the system and they do not overlap. Thirdly, which has the least concern as it is a matter of convenience, we must ensure a collimated region between the 2nd and 3rd lens. Keeping this in mind we may then begin mode-matching.

To begin the mode-matching process there are a number of factors we must input in order to run the program as coded which include the input parameters of the Gaussian beam, the length of the optical system desired, the final parameters desired and various aesthetic options editable by the user—for example, the self_flag variable determines the overall algorithm used when determining solution fitness; when set to 1 we use a method that is based off of Sidney A. Self's "Focusing of spherical Gaussian beams" paper, otherwise it is set to 0 and we use our own algorithm.

```
%Permute all possible lens combinations out of set of lenses
lens_set = [.075, .203]; %Given lenses of unique focal lengths
lens_permutations = pick(lens_set,3,'or'); %3 lens solutions

%Pre-defined Constants
lambda= 1.064E-6 ; %Wavelength of beam
Ltot= 1.010675025828971 ; %Length of optical system
r0= 1.0E+100 ; %Initial radius of curvature
w0= 2.563E-5 ; %Initial waist
x0= 0 ; %Starting position of beam
wf= 3.709E-5 ; %Desired final waist
rf= 1.0E+100 ; %Desired final radius
lens_width = .03; %Lens width
show_lens_width = 1; %Set to 1 to enable display of lens width on solution propagation plot
show_lens_position = 1; %Set to 1 to enable display of position of center of lens on solution propagation plot
display_prop = [show_lens_width, show_lens_position];
n_truncate = 3; %number of digits in truncated solution
n_visualizations = 2; %number of best solutions to visualize
n_hist = 1000; %number of sample points in histogram
stability_max = 1; %max stability (y-axis) shown on energy vs. stability graph
self_flag = 0; %Set to 1 to use Self's gaussian beam propagation, otherwise set to 0
%End list
```

Figure 1. Typical user settings for the program accompanied by comments explaining their usage.

The first step the automation process we must consider is the lens set that it we are able to work with. In the laboratory setting, there is a limited subset of available lenses with given focal lengths, thus the ability to edit the program to the needs of the user is of utmost importance. The user must input all of the unique focal lengths of each lens available to them within the array lens_set. This array is then run through a separate function (pick.m) where all possible three lens permutations and combinations of the lens set are placed within the array lens_permutations—it is important to note the permutations included involve repeats of lenses as well (i.e. three of the same lens) and thus if this is undesired an additional look into pick.m documentation allows the user to change it as they see fit. After all possible combinations and permutations of the lenses are provided, the program then mode matches to retrieve solutions that cater to the desired final properties. The mode_match.m function places lenses into random positions and stores all possible solutions; these shuffles are done over the entire lens permutation array for a user-set variable (n_shuffles) number of times for each set of three lenses. Although certain lens positions are truly chosen at random, not all positions are chosen at random. The third lens, for example, is always placed to transfer a collimated region to a focused spot (which typically occurs at a distance of the lens focal length from the optical system's final position). The other two positions for the lenses are then chosen at random, ensuring that the lens position always preserve order and checking whether the two random positions chosen are too close to each other (making sure it is larger than a threshold quantity set by lens_width)—in the case that the two positions are too close to each other, the program simply runs the random placement again until they are sufficiently far enough apart. At the end of this process, we have an initial random placement of lenses which we then send to our fitness function which minimizes "Energy" for given solutions.

Two stages are performed to calculate the energy of a given lens set: a preliminary check and a deep optimization. The preliminary check uses MATLAB's built-in fminsearch function to find

and minimize solutions with a max iteration value of 10. This serves as a rough overview of potential solutions for a given initial random placement of lenses. However, this approximation is too unrefined as the tolerance for solutions is too rough and not enough iterations are run through in fminsearch to find a more exact solution. Optimal solutions are desirable as optical systems are very sensitive; differences on the order of centimeters or even microns can affect the overall beam profile. It is with this in mind that a deep optimization is then taken with these rough solutions in mind, using the same fminsearch with more iterations. From the positions found by the rough approximation, a second pass is done on these points using a max iteration value of 100. In essence, the preliminary pass allows us to approximate where minima are likely to be around and the deep pass allows us to refine those positions to greater precision. A two pass approach has proven to lead to faster runtimes as opposed to one pass with high iteration values. Once the mode_match.m function has iterated through each possible lens set, solutions are outputted and the "energy" is sorted from lowest to highest within the array.

The fitness function itself is the next function to consider as it determines how the fminsearch ranks energy for particular solutions. It is important to note that "energy" in this sense does not refer to the classical meaning of energy but is used as a colloquialism to easily refer to the optimization level of a solution. Lower energy solutions correspond to more optimized solutions and conversely higher energy solutions correspond to lesser optimized solutions. Within the fitness function there are two methods by which we determine whether a solution is sufficient and two fitness factors that are held constant between these two methods. The two factors which we hold constant are the checks which ensure that the lens positions are: between the beginning and end of the optical system as well as not placed too close together. To ensure that the lenses are within the optical system the distance of the first lens from the beginning and the third lens from the end of the optical system are calculated. Values which are small, implying that the lens is close to the ends, have higher values of penalty and are thus added to the overall energy of the solution (higher energies mean poorer solutions and lower energies mean better solutions). Taking a look at a particular visualization of a fitness function we can see these energy barriers quite plainly.

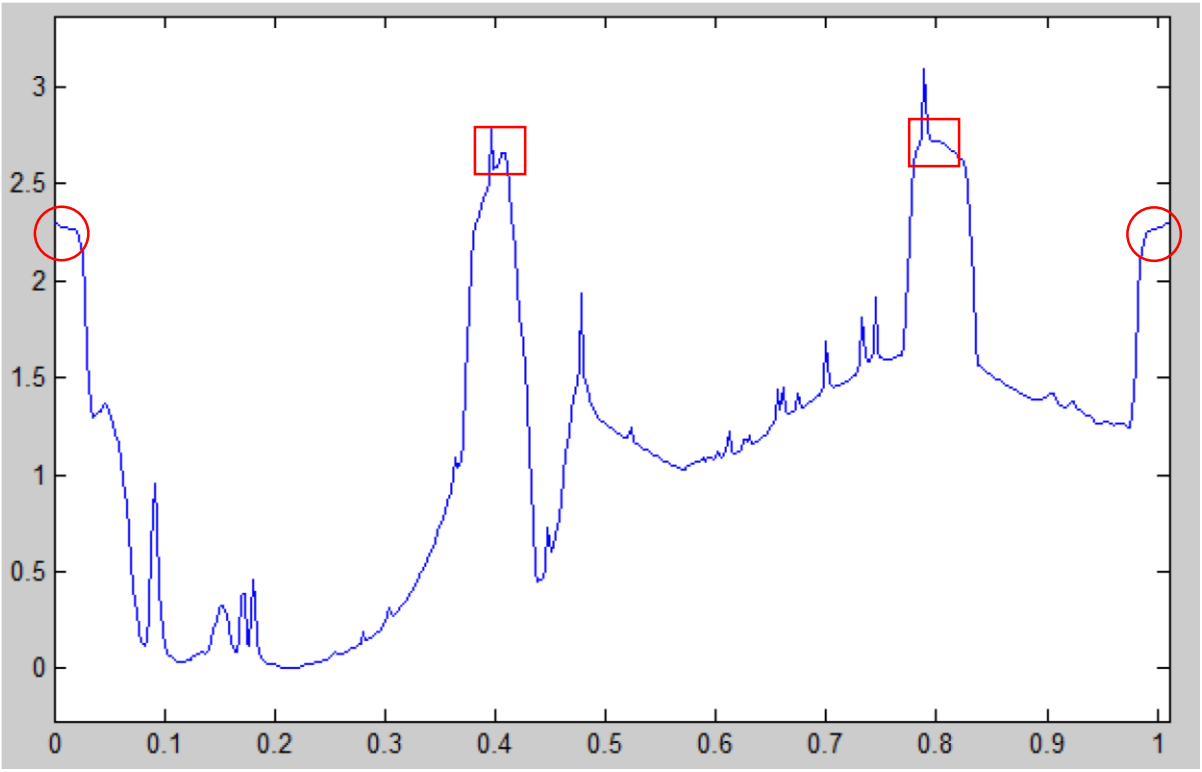

Figure 2. Fitness function for optical system where x_lens_two = .403m, x_lens_three = .803m. The x-axis is position of the first lens and the y-axis depicts the penalty attributed to that position. The effects of the various penalties can be seen within the plot; penalties associated with being within x_0 (beginning of optical system) and x_f (end of optical system) are seen in the peaks near the ends of the plot (which are highlighted by circles). Penalties associated with lens overlap are apparent in peaks ~.4m and ~.8m (which are highlighted by squares).

The next factor, not placing the lenses too close to each other, is calculated by taking the difference between the positions of the lenses themselves. Lower values of difference correspond to lenses being closer together. Based on the user set width of the lenses, the closer that difference is to the lens width the worse of a solution it is, thus also corresponding to a higher energy solution.

Which brings us to the two methods by which mainly determine minimum solutions. The first method, using Sidney A. Self's "Focusing of spherical Gaussian beams" equations, relies on ensuring that there is a collimated region between the second and third lens as well as matching the waist at the end with the desired waist input by the user. Self's equations allow for us to calculate the resultant image and position of the image from a given input Gaussian beam as it passes through a thin lens. Using this output, we then use it as an input for the second lens whose output is then used as an input for the third lens, finally culminating to a final waist position and radius. A collimated region between the second and third lens is highly desirable—collimated regions mean that the third lens is not sensitive to placement as the

waist of the beam diverges very slowly when properly collimated. This is widely applicable to lab setups where inaccuracies in position are bound to occur due to limitations in ability to place lenses with the precision of micrometers. To create this collimated region the first image resulting from the second lens, should be sufficiently far away from the third lens in either the positive or negative direction. At large distances, a collimated region is created as it passes through the lens. The closer the resultant image from the third lens is to the second lens the higher energy is attributed to the solution and thus added to the overall energy of the solution. Next we compare the final waist resulting from the third lens to the desired waist supplied by the user at the beginning of the program. A difference is taken between the two and is added respectively to the overall energy of the solution based on how close or far apart the two values are (closer values corresponding to lower energies).

The second method of choice utilizes Professor Mikhailov's Gaussian beam propagation code. In a similar fashion, a collimated region is desired in this method though how it is calculated and added to the fitness differs greatly from the Self-based method. To check for collimation, a linearly spaced array is created within the optical system. At each point in this array, we calculate (using gbeam_propagation.m) the q values associated with the point in the beams propagation. Doing complete propagations of both forward and backward simulations, we then examine the q values. We specifically looked at the intermediate q values between the second and third lens. If the standard deviation of these q values were high then we added a higher energy, as this means that the beam is diverging instead of being properly collimated. Smaller values of standard deviation meant that the beam diverged or converged slowly which is exactly what collimation requires. In addition to using this check for collimation, we also checked whether the forward and backward propagations of the beam aligned properly. Qualitative analysis of the plot allows judgment of whether a beam is mode-matched or not; forward and backward propagations of the beam with the same parameters and optical system should ideally yield identical beams. If the forward and backward propagation do not match then the sensitivity of the q parameter to small deviations is apparent. However, if they do match then we do know that it is solution that won't deviate from the current solution if we perturb the position of the lenses. Figure 3 shows an example of a poorly mode matched solution.

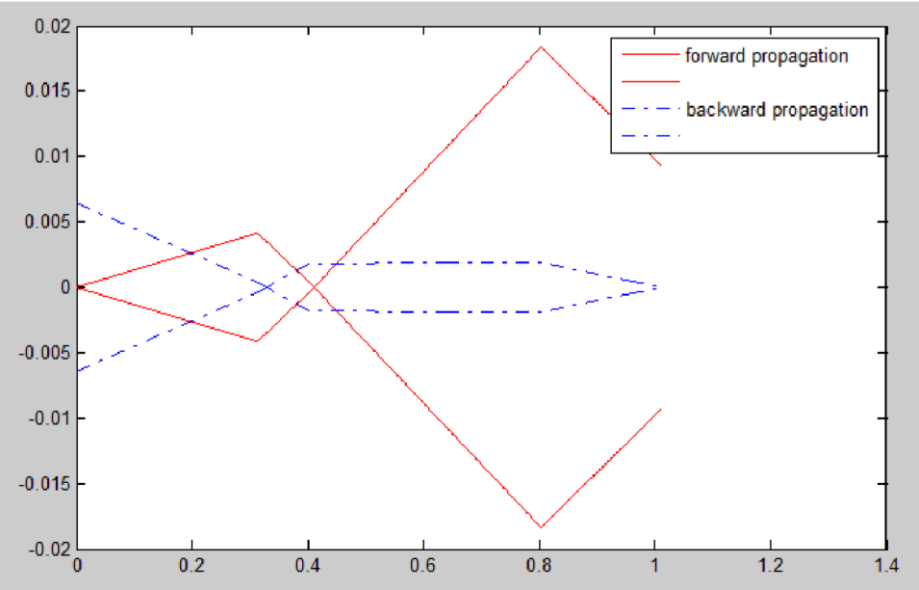

Figure 3. A poorly mode-matched beam solution. As seen qualitatively, the forward and backward propagation of the beam do not coincide at all. The x-axis is the propagation direction of the beam and the y-axis is the waist distance from the optical axis.

Once everything has been properly minimized the final potential solutions are outputted in a single array.

The next step in the program to consider is the outputted solutions themselves. As stated earlier, solutions are sensitive on orders of micrometers and thus a great number of similar solutions may appear within all possible solutions. It is important then, for the sake of the user, to find a way to consolidate solutions to only those which differ enough to truly be considered a separate solution. The user sets, as in figure 1, an n_truncate value whose value determines how many decimal places to consider to determine solution similarity—determining whether the placement of lenses between solutions are too similar or not. For example, a value of n_truncate = 3 would yield an edited solution array whose values are truncated to the 3rd decimal place. After determining the desired tolerance, the solution array is rounded to that particular decimal place and the MATLAB built-in function unique is used to output an array of solutions that are unique by decimal standards. For example, in the solution array:

$$\begin{bmatrix} .1234 & .2222 & .3333 \\ .1233 & .2222 & .3333 \\ .2222 & .3345 & .5555 \end{bmatrix}$$

Where elements in the first column, second column and third column represent the placement of the first, second and third lens respectively. An n_truncate value of 3 would round values to the third decimal place to yield:

$$\begin{bmatrix} .123 & .222 & .333 \\ .123 & .222 & .333 \\ .222 & .334 & .555 \end{bmatrix}$$

As one can see, the first and second row solutions are exactly the same within this approximation. When run through MATLAB's unique function we get the final output array:

$$\begin{bmatrix} .123 & .222 & .333 \\ .222 & .335 & .555 \end{bmatrix}$$

or put simply, two unique solutions. In addition to outputting a unique solution array, another array is created to keep track of the indices corresponding to their row element position in the original array. For example, for the above two unique solution array, we obtain an index array of:

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

The first element, 1, corresponds to the fact that first row solution in the final output array corresponds to the first row solution of the original array. The second element, 3, corresponds to the fact that the second solution in the final output array corresponds to the third row solution of the original array. This is important for the visualization aspects of the Gaussian profile as a non-rounded value for lens position is more advantageous when simulating the beam—a more exact value for position creates a more accurate simulation of the beam's interaction with the lens.

Which brings us to the simulation and profiling of the Gaussian beam as it passes through our calculated solutions. The propagation and plotting of the Gaussian beam is based on Professor Mikhailov's original solution_visualization.m code; within it we take 1000 sample points linearly spaced across the optical system and calculate the q values for both the forward and backward propagation of the beam. The beam profile is then plotted using MATLAB's builtin plot function. In addition to this, I've added the ability to plot the size of the lens itself, in relation to the entire optical system—allowing for the user to visually simulate how the program's thin-lens approximation looks when we transpose actual lens width upon it. For even greater clarity, I included another line which represents the center of the lens and essentially a point-object lens as the thin-lens approximation requires. All of this is color coded to the legend, to avoid inconsistency and provide an easier read for the user. Outputted along with the graph is a title which includes all the necessary information to setup the situation in a laboratory setting. The solutions are numbered, the energy of the solution is listed, the positions and focal lengths of each individual lens is given and the final waist and radius of the solution is given. An example of what the solutions look like are given in Figure 4:

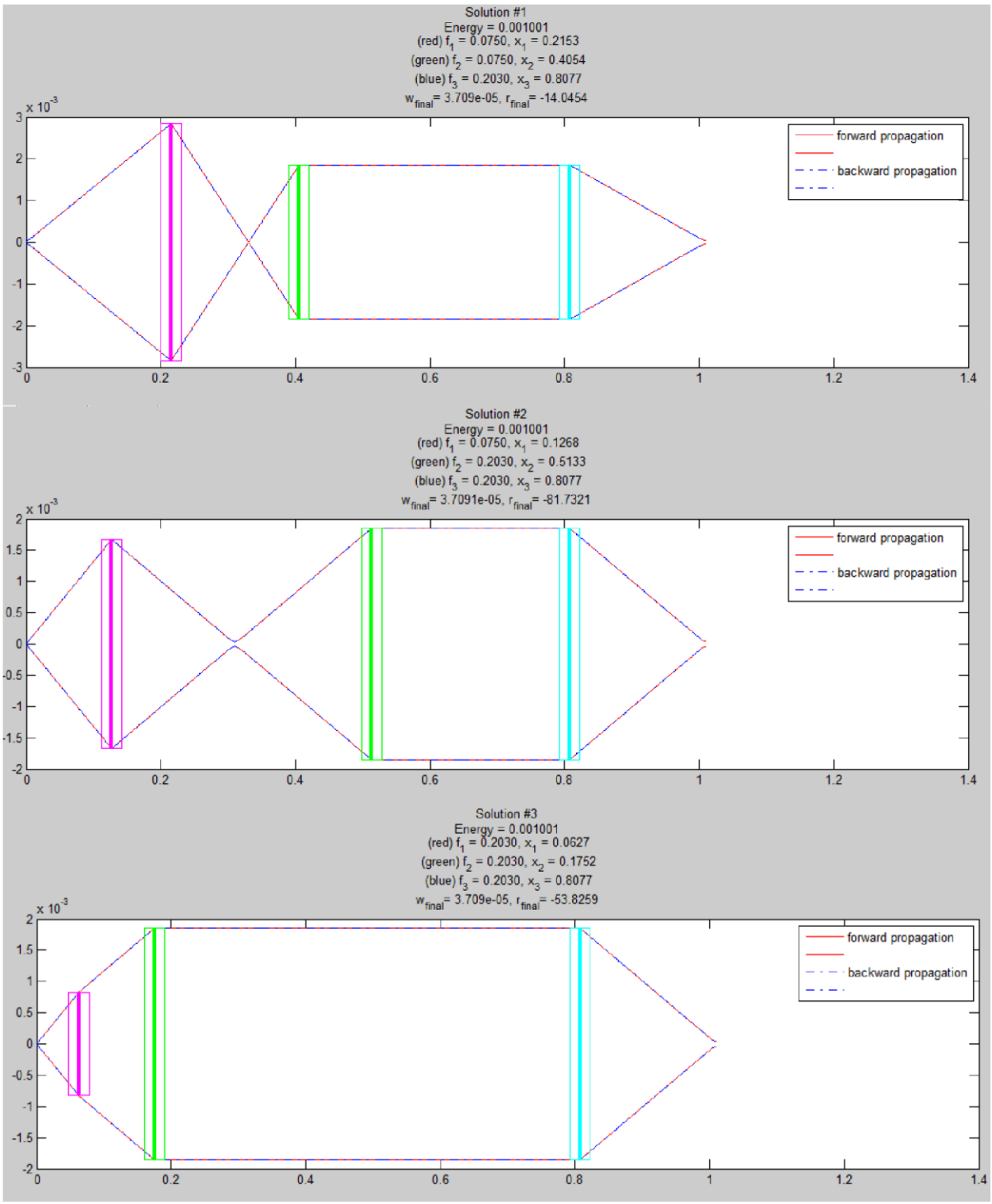

Figure 4. Three potential solutions for a lens set consisting only of .2030 and .0750 focal length lenses. The x-axis is the distance along the propagation axis and the y-axis is the radius of the beam. The collimated regions occur between the 2nd and 3rd lens and display as two parallel lines.

The last and final step of the mode matching process is the stability calculation and visualization. Stability is a major factor in determining whether a solution is feasible or not; a solution which is too sensitive to perturbations is seldom useful as the precision required to place the lenses in that situation are outside the bounds of reality. So, a stability function is produced and calculated in order to better allow the user to choose which solution they would like to use for their laboratory. It is up to the user to decide whether to compromise energy for stability and vice versa, as every user has different needs from the program. The stability of a solution is calculated by randomly perturbing the lens positions and creating another fitness function out of these perturbations. Based on the user set amount (n_hist), the program will perturb it n_hist amount of times which is then stored in a histogram array. After storing histogram points, it is then visualized and plotted. The area under the histogram curve is calculated by the MATLAB built-in function trapz which is a trapezoidal numerical integration. The final output plot can be seen in the example in Figure 5.

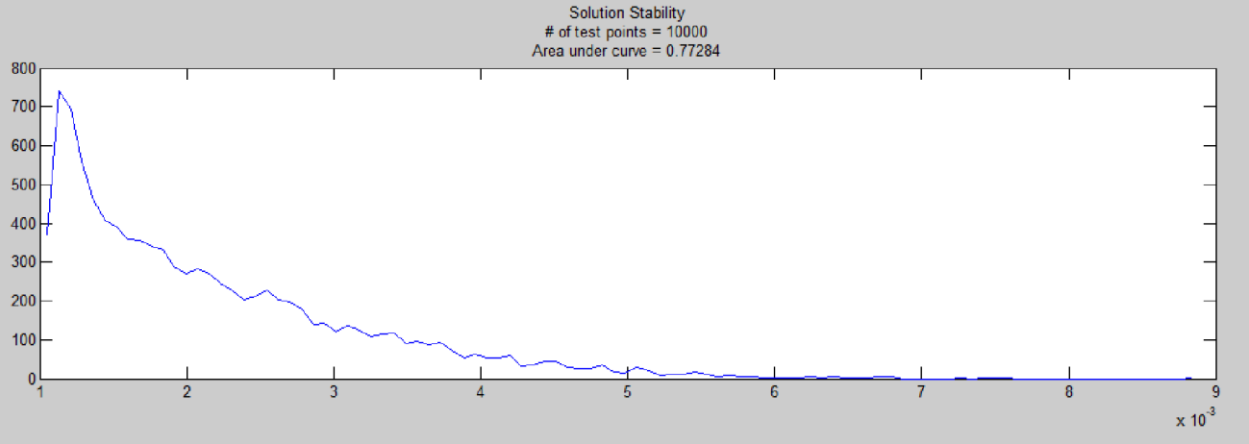

Figure 5. An example histogram showing solution stability. The number of test points and total area under the curve are given; this is outputted for each solution visualized.

These plots are then subplotted beneath their corresponding solution in order to associate them more easily.

The final output solution looks something like Figure 6.

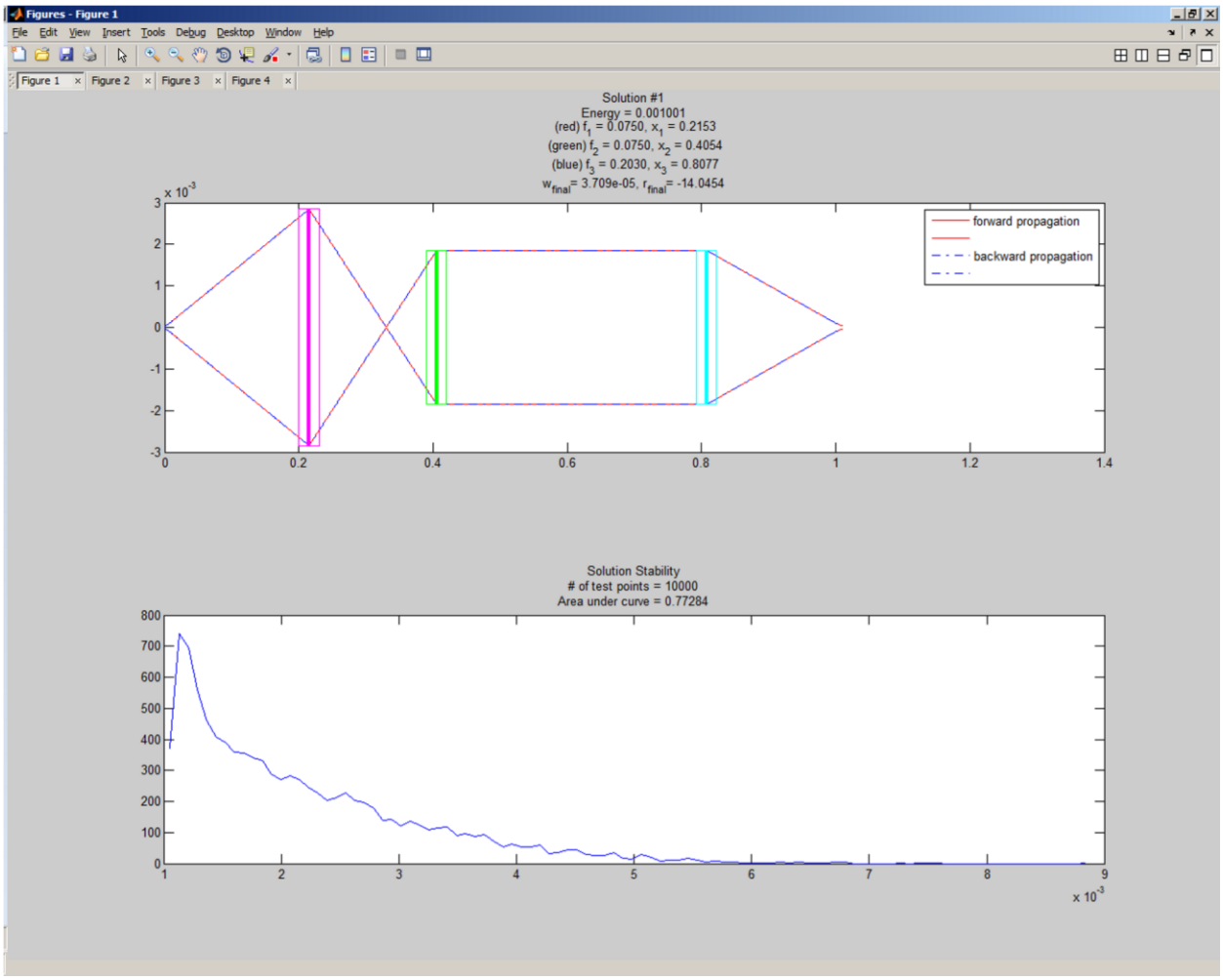

Figure 6. A desktop view of the MATLAB workspace with one solution being displayed.

Additionally, each solution then has its energy graphed versus stability in order to compare the solutions together on one plot (as seen in figure 7).

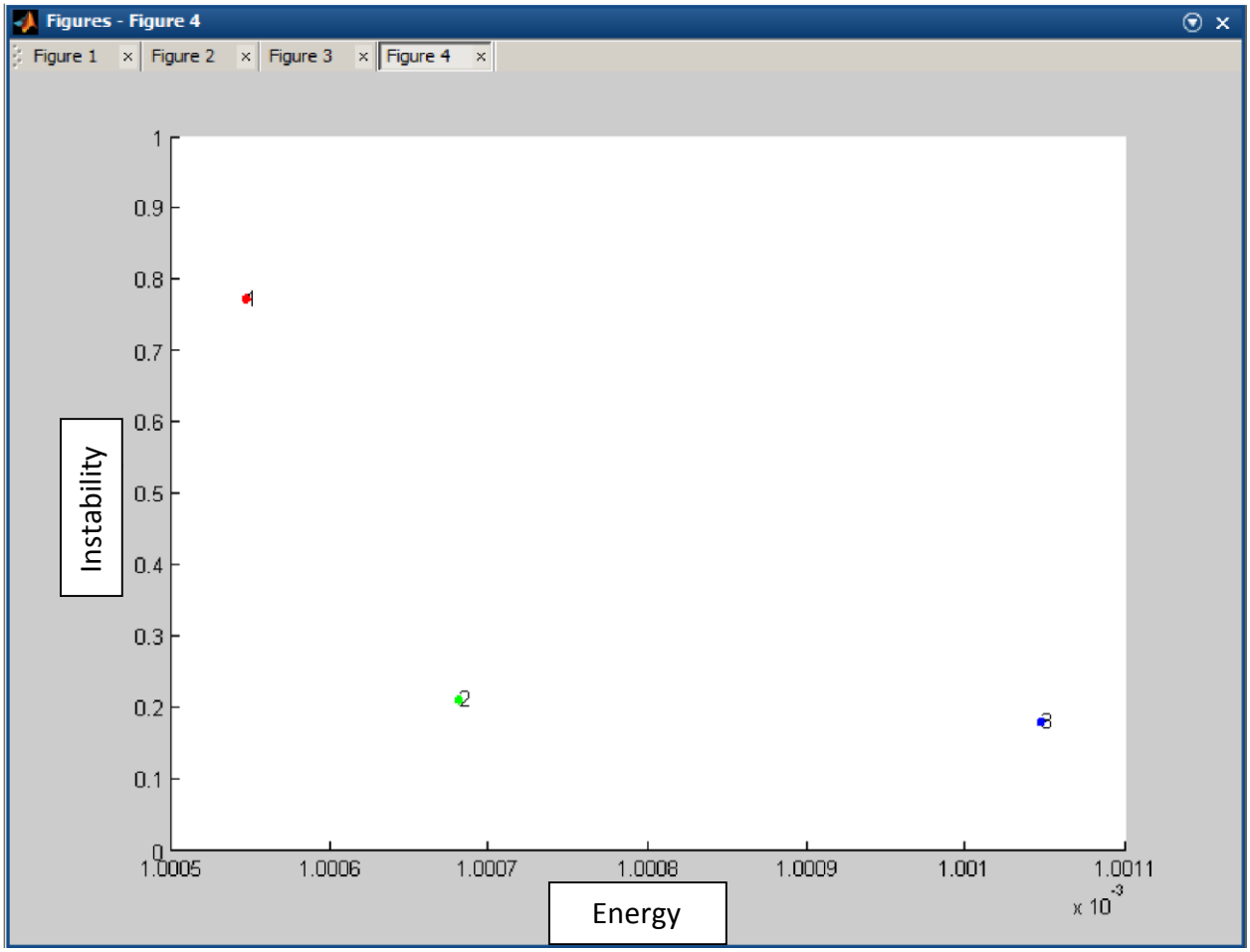

Figure 7. Energy vs. Stability graph. Each solution is placed on this plot in order for the user to interpret the data more easily. As one can see, solution #1 has high value for instability but the lowest value for energy. Meanwhile, solution #3 has a low value for instability and the highest value for energy. A user must then decide which they favor more or caters to their experiment more—or even choosing a middle ground such as with solution #2.

6. Conclusion

In conclusion, the automated mode matching system functions and outputs solutions which are theoretically possible and analytically sound. The program runs for a total of 46.2 seconds on average for a two unique lens set and outputting 5 different solutions. Previously, mode-matching was done manually. A general familiarity with lenses and optics allowed for experimental placement of lenses to make an educated guess towards what should give the output beam desired. However, we can automate this process to great efficiency and remove the need to manually mode-match. Practicality, efficiency and accuracy are what is most needed from this program and in that regard it does deliver. However, that is not to say the program is finished. The ability to incorporate not only thin lenses but other types of optical medium (such as thick lenses) would prove invaluable to laboratory settings and the ability to set the number of lenses or optical elements desired would be key to catering to equipment deficient projects.

References

Alda, Javier. "Laser and Gaussian Beam Propagation and Transformation." Encyclopedia of Optical Engineering (2003): 999-1013.

Self, Sidney A. "Focusing of Spherical Gaussian Beams." APPLIED OPTICS 22.5 (1983): 658-61.