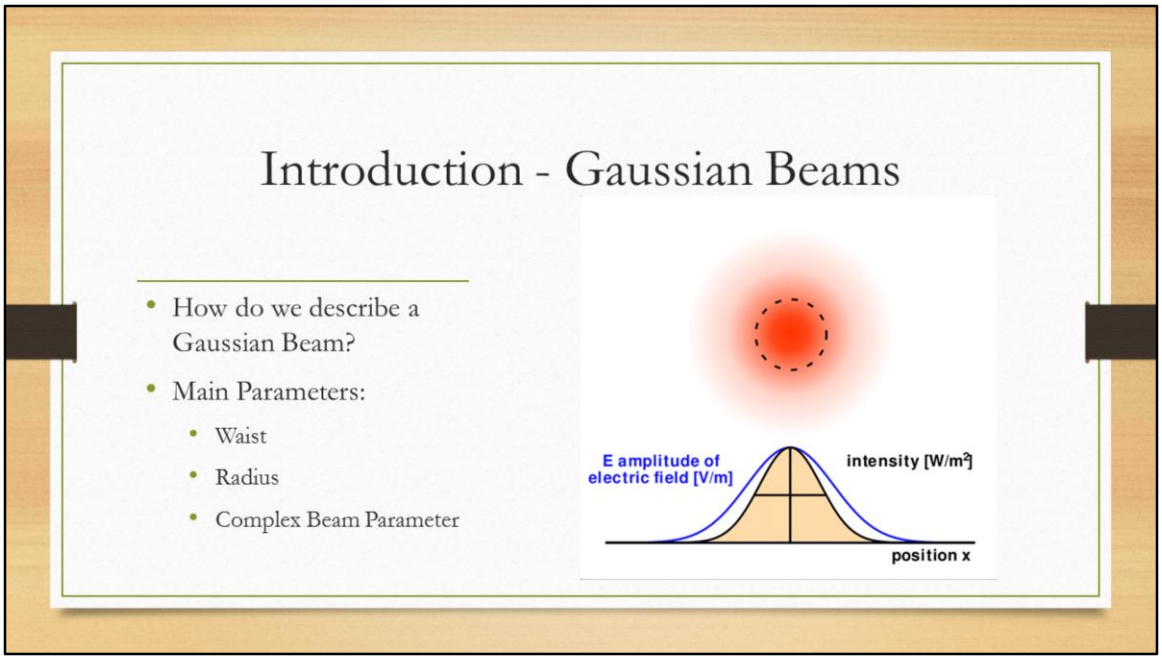
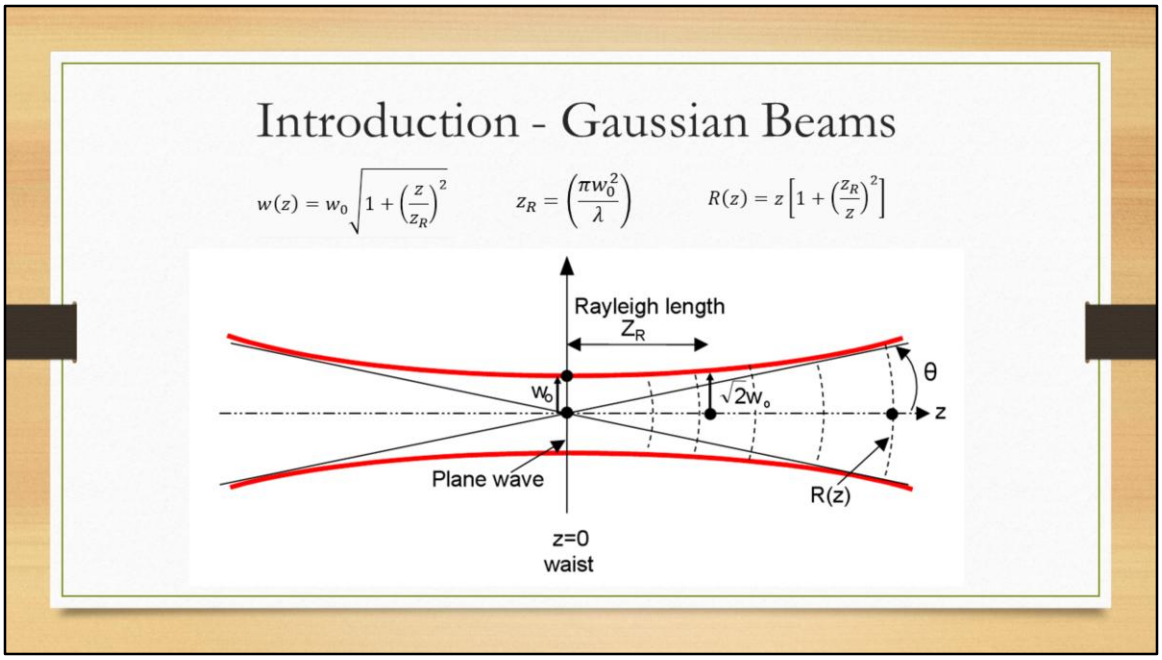
Automated Mode-Matching of Gaussian Beams

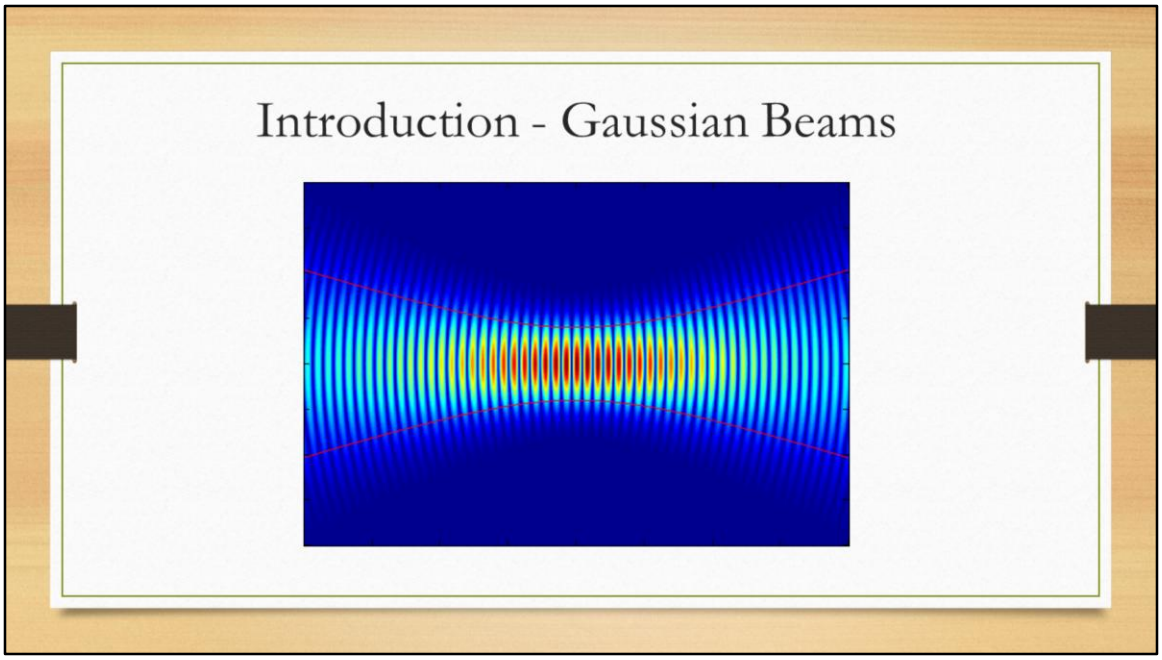
Matthew Argao



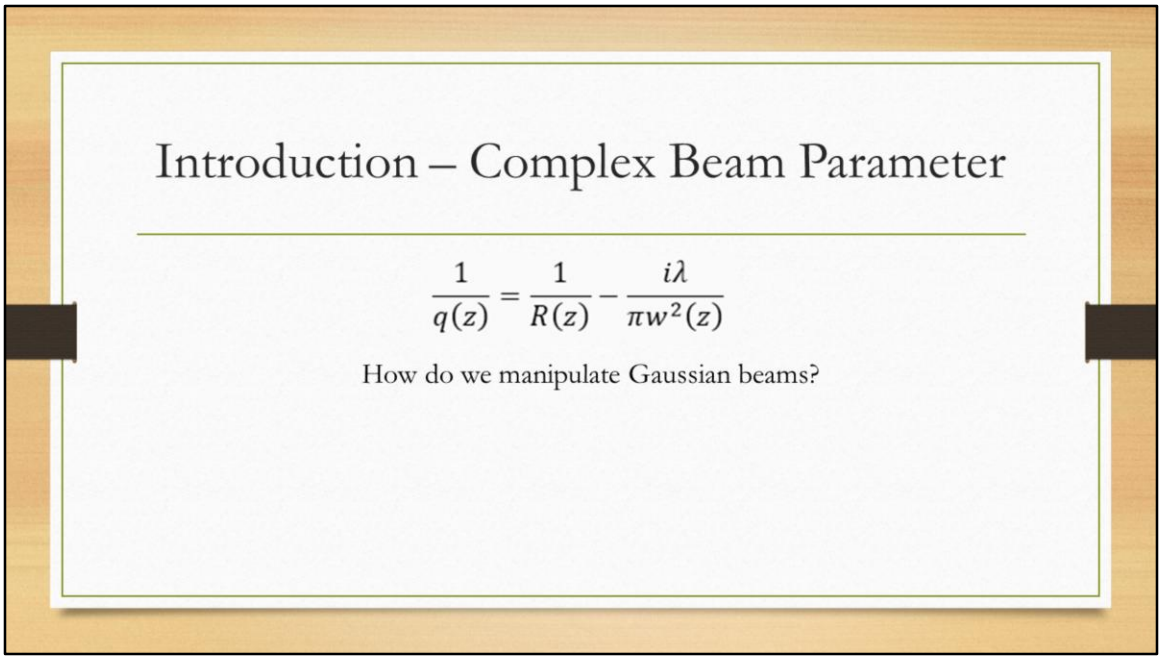
Before we get into mode-matching we must first examine just what IS a Gaussian beam. In the field of optics and laser related physics, Gaussian beams are a well documented model of electromagnetic radiation whose characteristics such as transverse electric field and intensity are approximated by Gaussian functions of the form shown here on the figure to the right. The image above the graph depicts what this distribution may look like physically– in this case, the beam spot left by a laser on, say, a wall. As one can see, the intensity (represented by the red) is strongest at the center, corresponding to x = 0 and as we move further out radially the intensity drops off. Since all Gaussian beams follow this definition we must define parameters that differentiate one Gaussian beam from a another. The main parameters which we are concerned with for this experiment are the Waist, Radius, and Complex beam parameter.



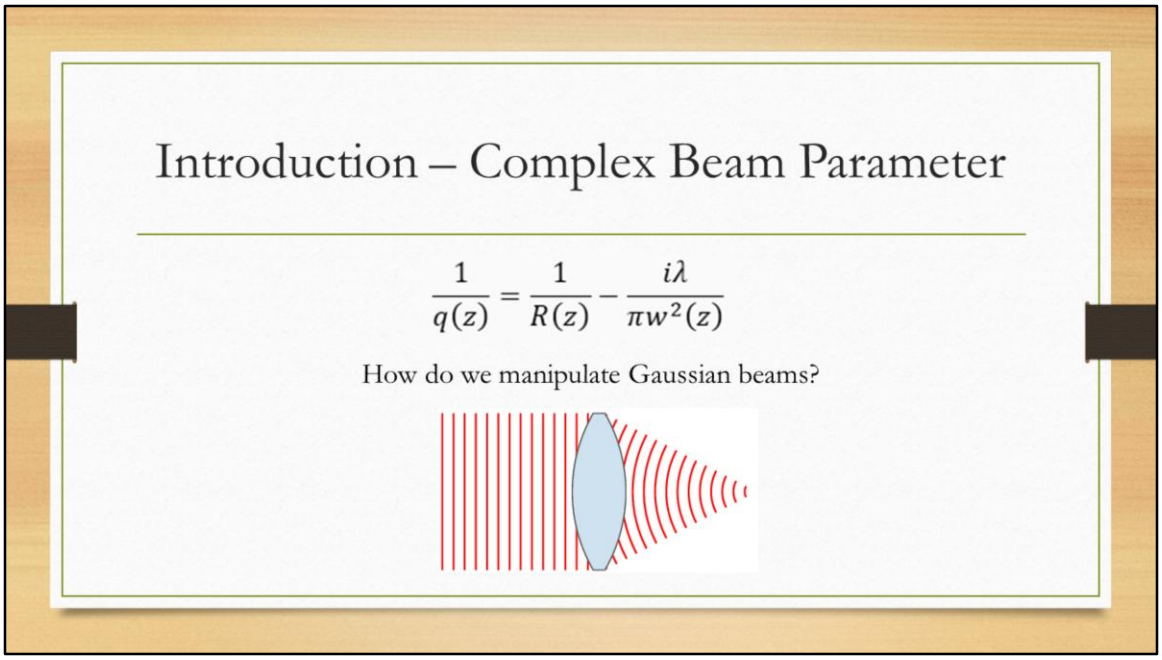
To more visualize these quantities we must first examine the shape of a Gaussian beam, shown here with a hyperbolic profile in red. The waist, labeled as w, defines the width of the of beam; the radius corresponds to the radius of curvature of the wavefronts, shown here by the dotted lines. To more formally define these quantities I have listed the equations above. As one can see, the radius and waist are functions of the distance along the propagation axis, z. The quantity w0 is defined as the point at which the waist is a minimum where the curvature of the wavefronts are essentially planar. The other quantity, z_r, corresponds to the Rayleigh length which is defined as the point at which the cross sectional area of the beam is twice the area at the waist minimum.



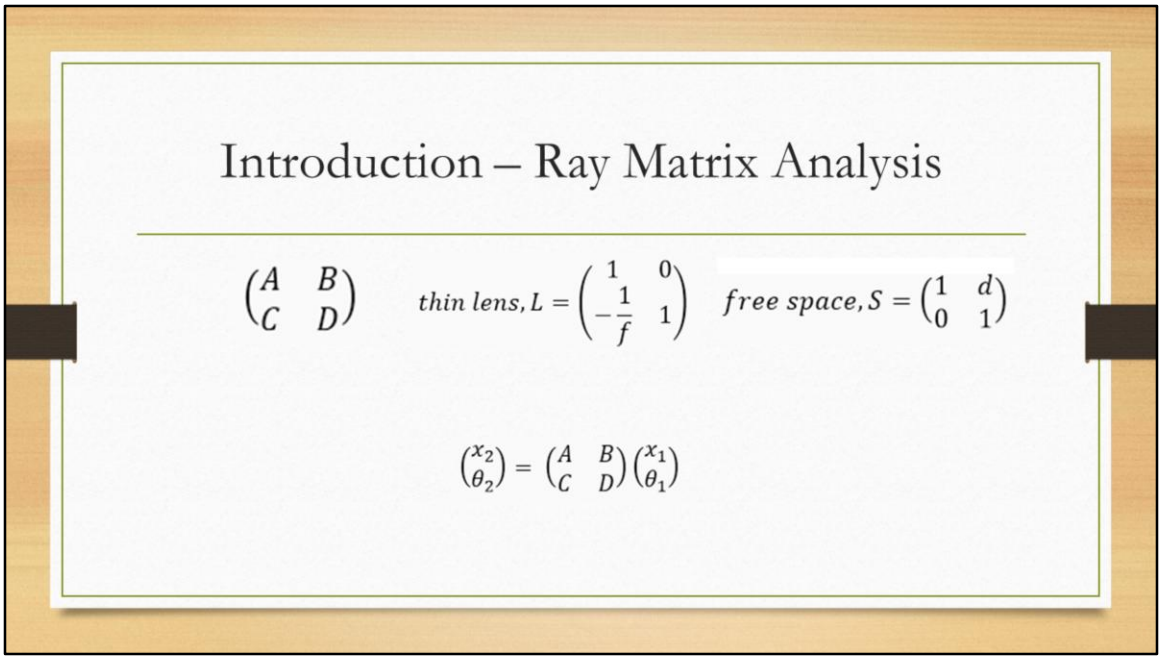
To get a better idea of what Gaussian beams might physically look like when in motion I have included the following image. The colors correspond to the instantaneous intensity at a given moment of the Gaussian beam. As the wave propagates from left to right we can see the individual wavefronts slowly become more planar as we approach the waist minimum and then gradually become more curved as we move past the waist minimum.



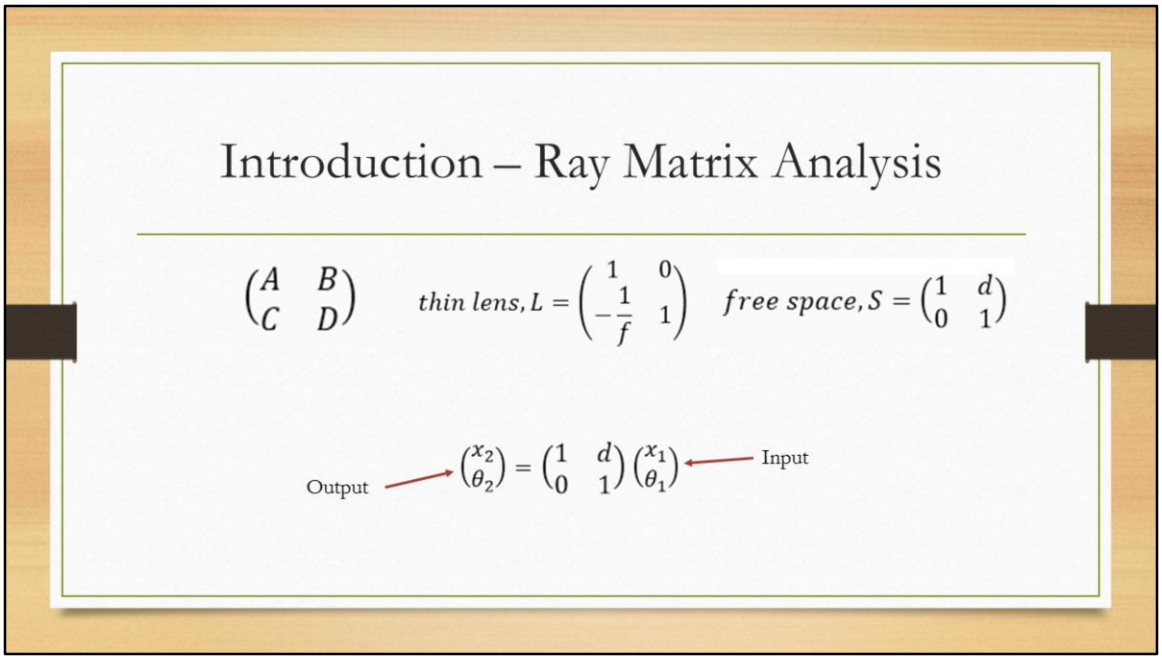
The last parameter I mentioned before, the Complex Beam Parameter, is distinct, however, from the radius and waist. This is because the complex beam parameter has nested within it, information on both the waist and radius of curvature. The relationship, shown above as a reciprocal, is invaluable to the analysis of Gaussian beams. We can analytically solve for q at any given point along the propagation and extract from q the radius of curvature and waist of the beam at that particular point in space. So now that we know what a Gaussian beam is and the parameters which define it– how we do we manipulate Gaussian beams into other Gaussian beams?



The most simple way to do this is through the use of lenses. As you can see in this small animation, the input beam has a given waist that seems to be uniform, consistent with the definition of a collimated beam, as well as a radius of curvature that approaches infinity (due to it being planar). However, once this input beam meets the lens it acquires a phase which causes the beam to become convergent.



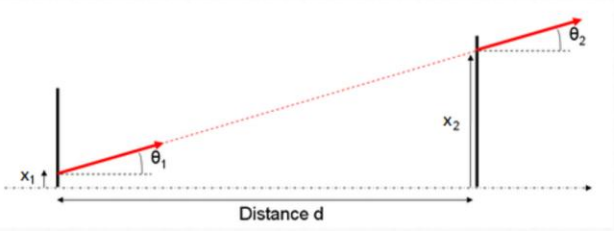
The next step to consider is how do we represent the effects optical elements, such as lenses, have on Gaussian beams? One way to this is through something called Ray Matrix Analysis. In RMA we work with matrices to represent the various optical elements of a system. Each element is assigned an ABCD transfer matrix, appropriately named due to its form shown in the top left. Two examples of what the ABCD transfer matrix might look like for an optical element are shown in the thin lens and free space matrices; where f corresponds to focal length of the lens and d corresponds to the distance travelled. When determining the ABCD matrix's effect on the Gaussian beam we must consider two reference planes: the input and output planes.



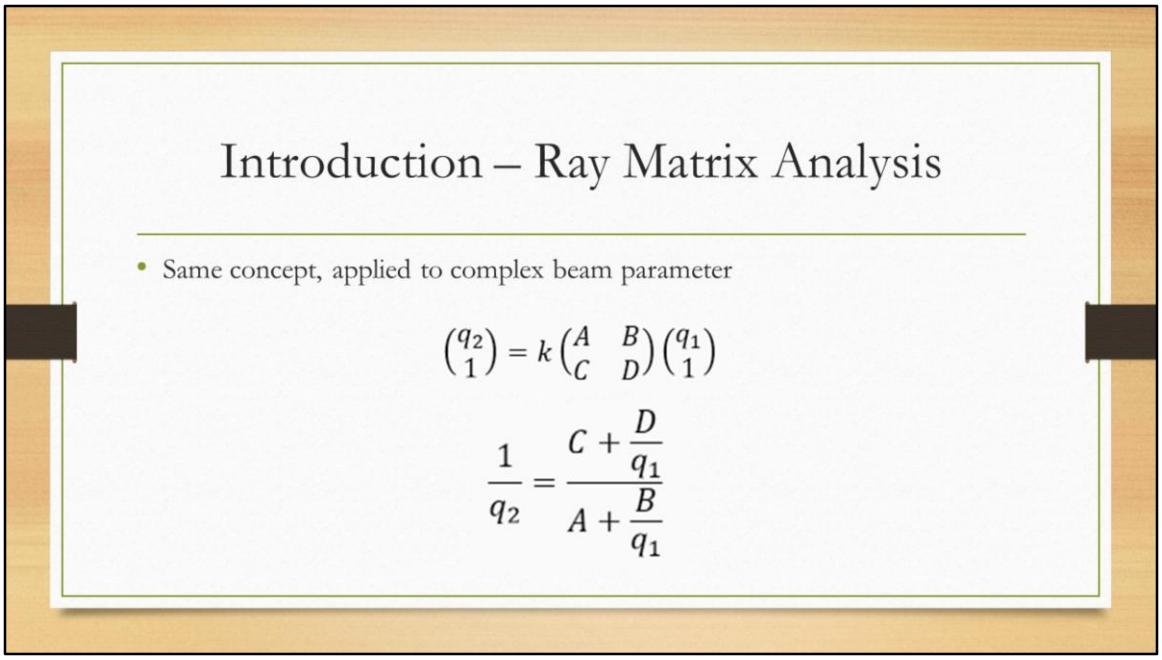
We consider the ray, or beam, entering the input plane a distance x_1 from the optical axis (which is taken to coincide with the z-axis) at an angle of θ_1. As the beam travels it eventually crosses the output plane this time with characteristics x_2 and θ_2. An example of this can be seen through the propagation of a beam through free space– whose ABCD matrix I've properly inserted into the equation. Expanding these matrices we get:

Introduction – Ray Matrix Analysis

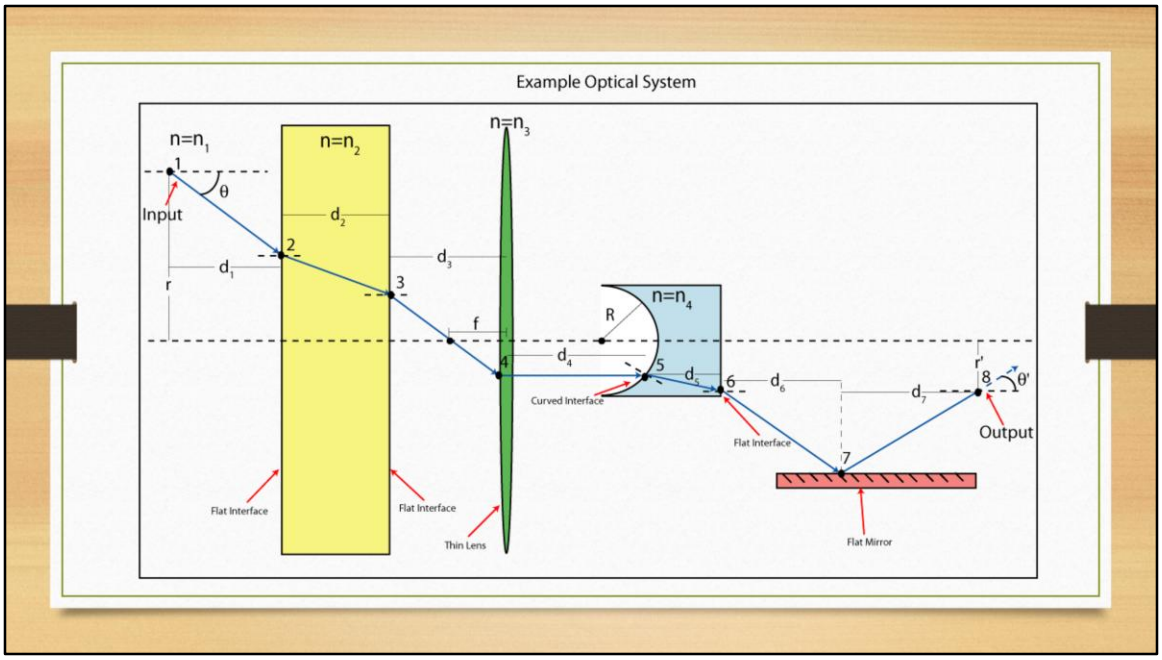
$$\tan\theta \approx \theta$$
$$x_2 = x_1 + d\theta_1$$
$$\theta_2 = \theta_1$$



The following system of equations. It's important to note that here we are assuming the paraxial approximation, where tan(theta) is approximately equal to theta. The figure above shows the before and after of the propagation.



Now that we have a grasp of the geometrical case we may apply these same principles to our complex beam parameter. Once the equation is expanded, the normalization factor k is cancelled out and (for convenience) the reciprocal is taken in order to obtain the general equation shown below. Using this equation we may extract all relevant parameters of Gaussian beams before an optical element and after interacting with the optical element.

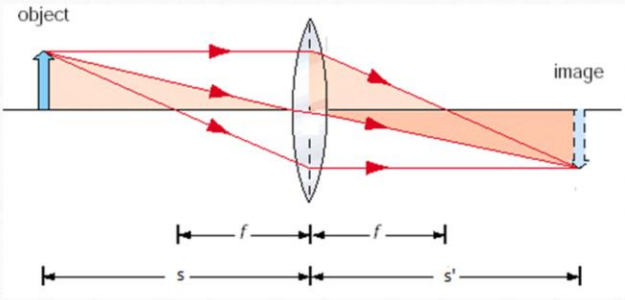


Although for the purposes for our project we are only concerned with the ABCD matrix of thin lenses and free space, it is important to note that Ray Matrix Analysis is a very general method which can easily be applied to other optical elements such as mirrors or materials with different indices of refraction. An example of what that system and the corresponding ray diagram might look like is shown here.

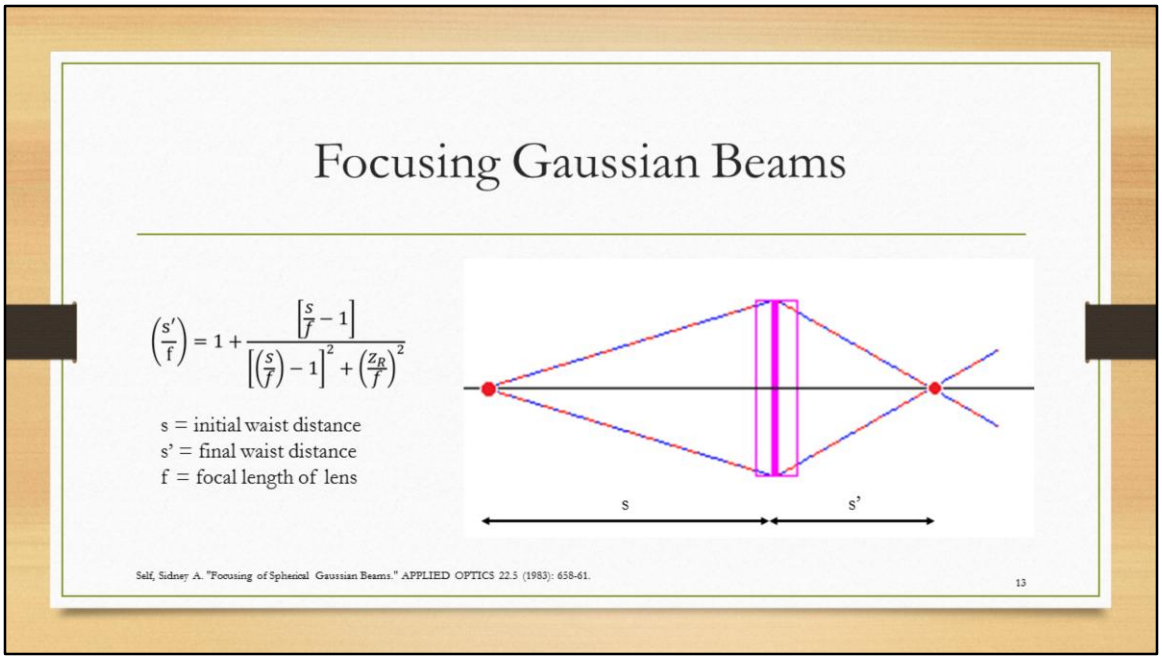
Focusing Gaussian Beams

$$\frac{1}{s}+\frac{1}{s'}=\frac{1}{f}$$

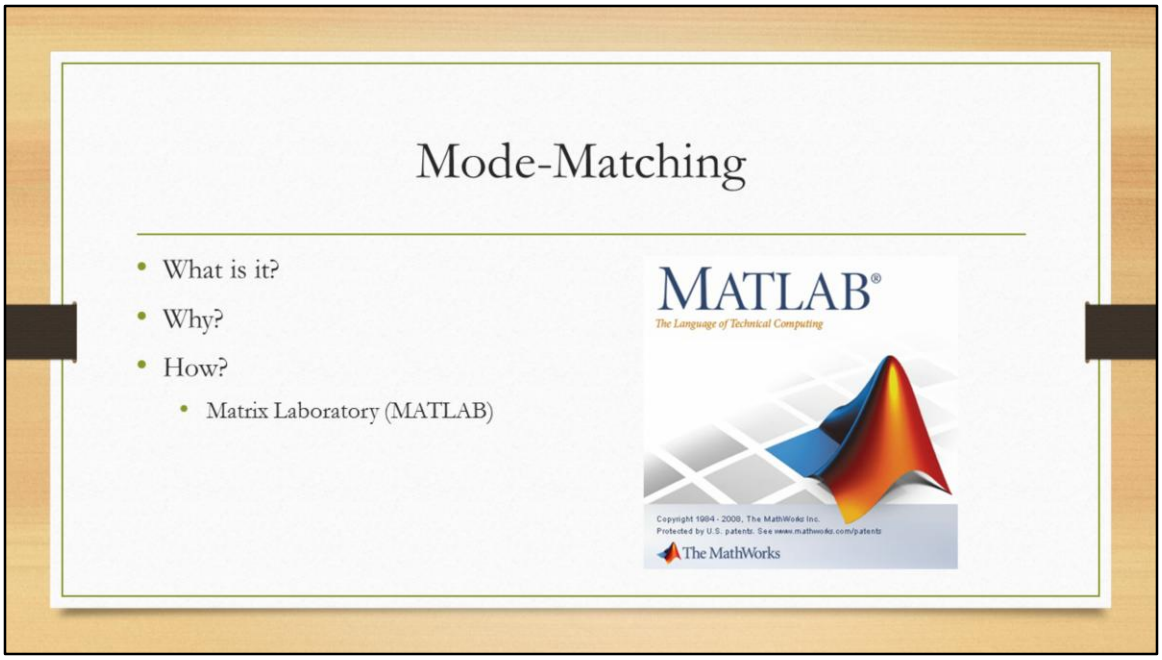
s = object distance
s' = image distance
f = focal length of lens



In the thin-lens approximation we ignore effects due to the lens thickness when the thickness of the lens is negligible compared to the focal length of the lens. For a standard thin lens the relation between object, image and focal length of the lens geometrically are given by the equation shown on the left. The physical quantities these variables correspond to are labeled here in the ray diagram on the right.



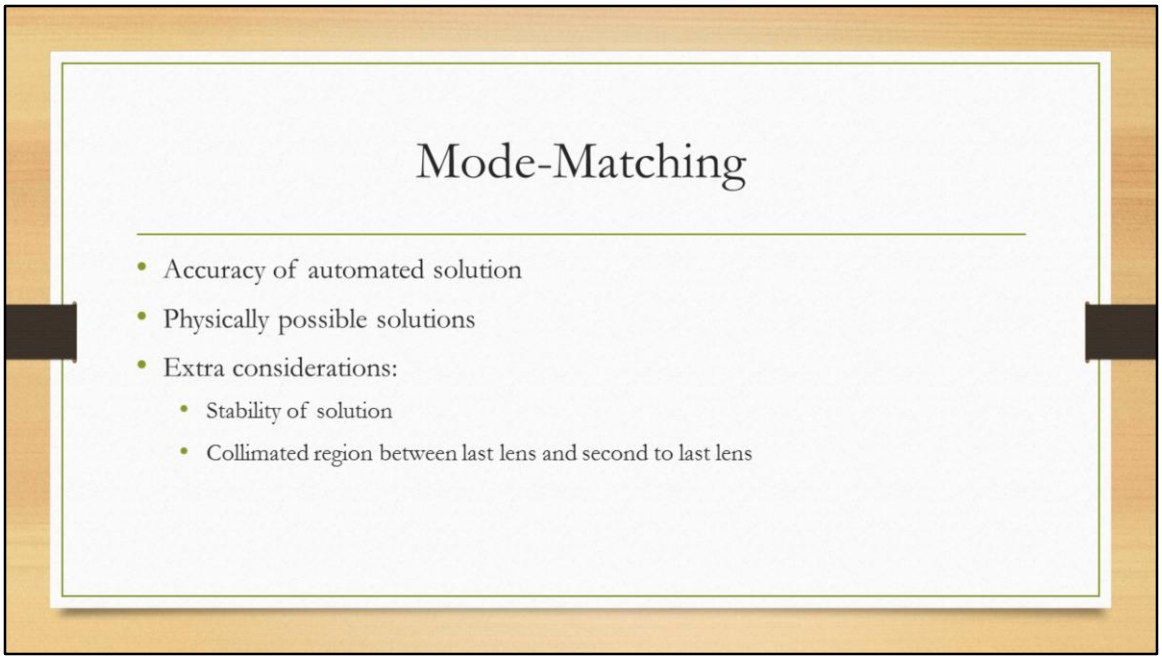
This same concept can be applied to Gaussian Beams. Provided the input beam waist radius w_0 and the object distance s, we are able to calculate the Rayleigh range (z_R), beam radius (w), and beam radius of curvature (R). For thin lenses, the beam radius is unchanged by the lens, however the radius of 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. The equation we use to calculate such quantities is given on the left, as pulled from Sydney A. Self's paper "Focusing of Spherical Gaussian Beams". Note that the quantities of s and s' have changed meaning; s now refers to the input beam's waist minimum position from the lens, shown in pink, while s' now refers to the output beam's waist minimum position from the 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.



Which brings us to mode-matching. Now that we know what Gaussian beams are and how they're manipulated the next question to answer is "How do we use Gaussian beams?". Researchers may wish to produce a particular Gaussian beam with particular characteristics but these vary on a case by case basis for each experiment. But as an 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. 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, saving time and energy. The program places lenses into an optical system defined by the user and arranges them such that it transforms the input beam into the desired output beam.

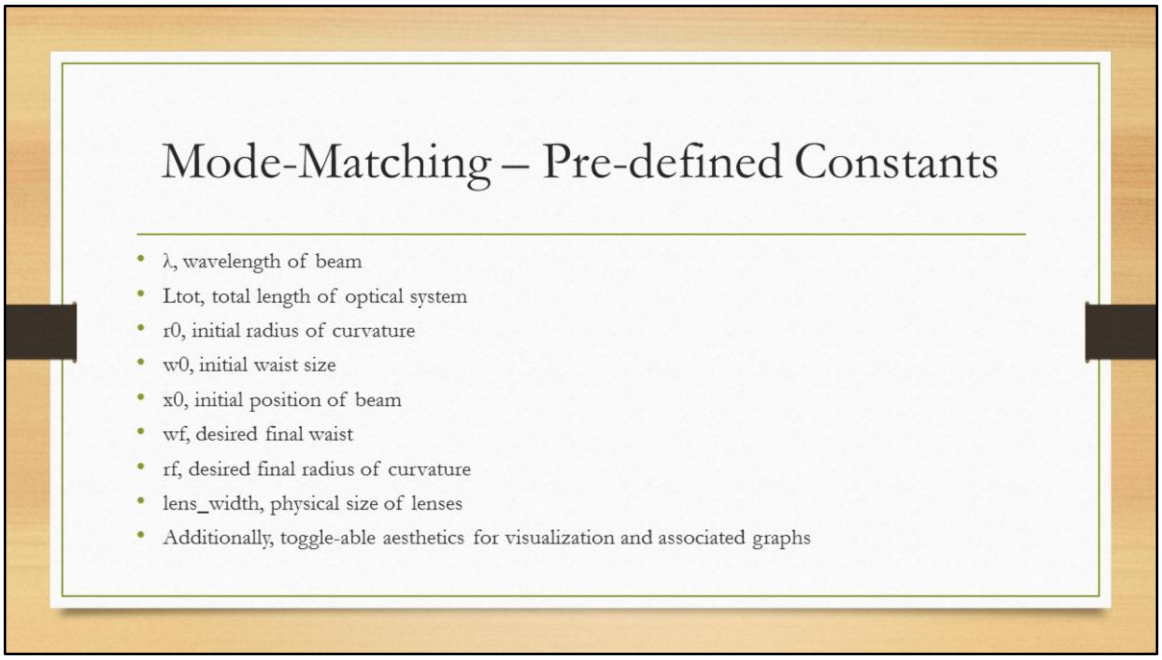
As one can imagine, we would expect to use and manipulate a great number of matrices due to methods such as Ray Matrix Analysis. Thus, we chose to work in a software aptly named Matrix Laboratory, or MATLAB for short. It allows us to program and simulate the propagation of Gaussian beams as well as incorporate how

the beams would interact with various optical elements.

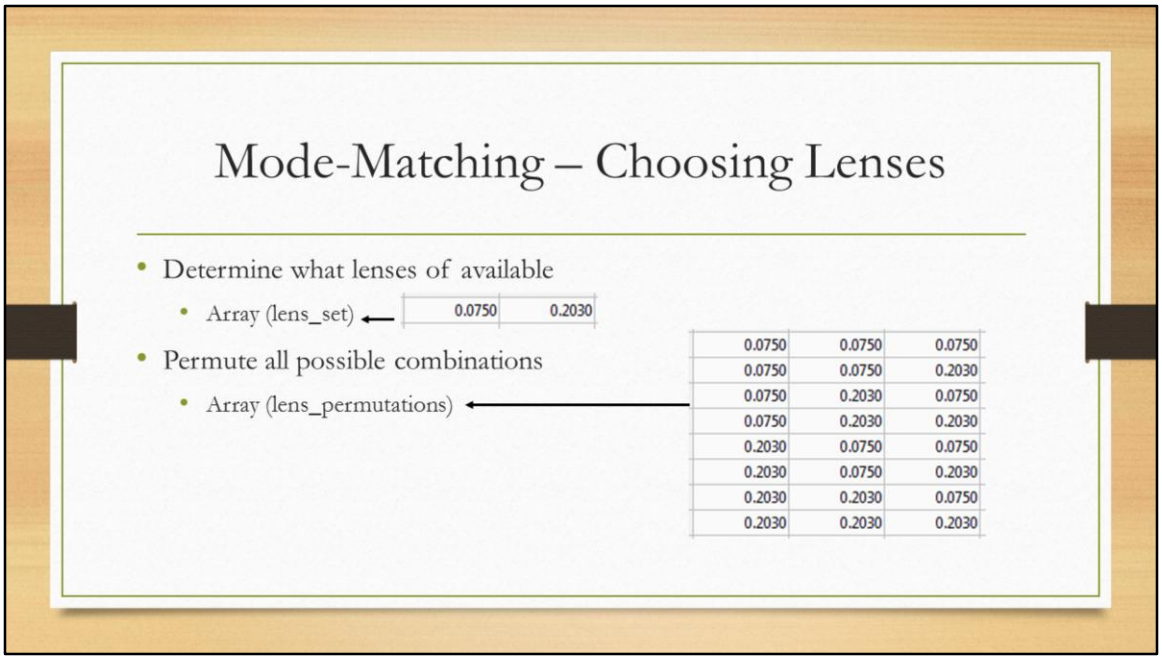


In order to properly mode-match there are certain considerations we must take into account. Firstly, we desire accurate automated solutions-- basically, the solution given should be as close as possible to our desired output thus our mode-matching algorithm should reflect that. The second consideration to take is ensuring that the solutions given are physically possible. Solutions which have overlapping lenses or have elements that exist outside the defined optical system are to be avoided. Overlapping lenses can never be placed properly due to the physical thickness of lenses and elements outside the system correspond to floating lenses off of the bench which is obviously impossible (at least for now).

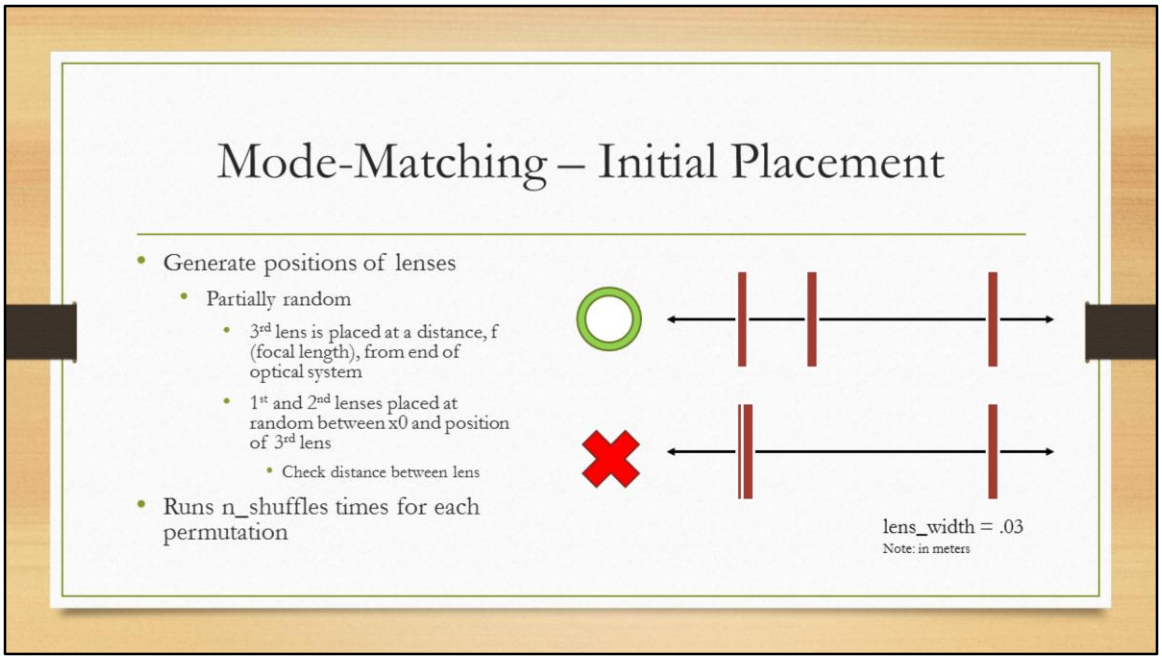
Additional considerations are taken but are more targeted towards ease of setup. For example, we include the stability of solutions in order to more easily setup the system physically as more stable solutions are less sensitive to perturbations. We also include a collimated region between the last and second to last lens which gives leeway in placement for last lens due to the slow divergence collimated beams exhibit. This is but a brief overview of what the program seeks to accomplish.



So before we get into the nitty-gritty of the algorithm itself we must first take a look at what is needed. Initial characteristics of the beam such as its initial waist, radius and wavelength must be inputted. Followed by the length of the optical system desired, size of the lenses and desired output characteristics. Once we have that…

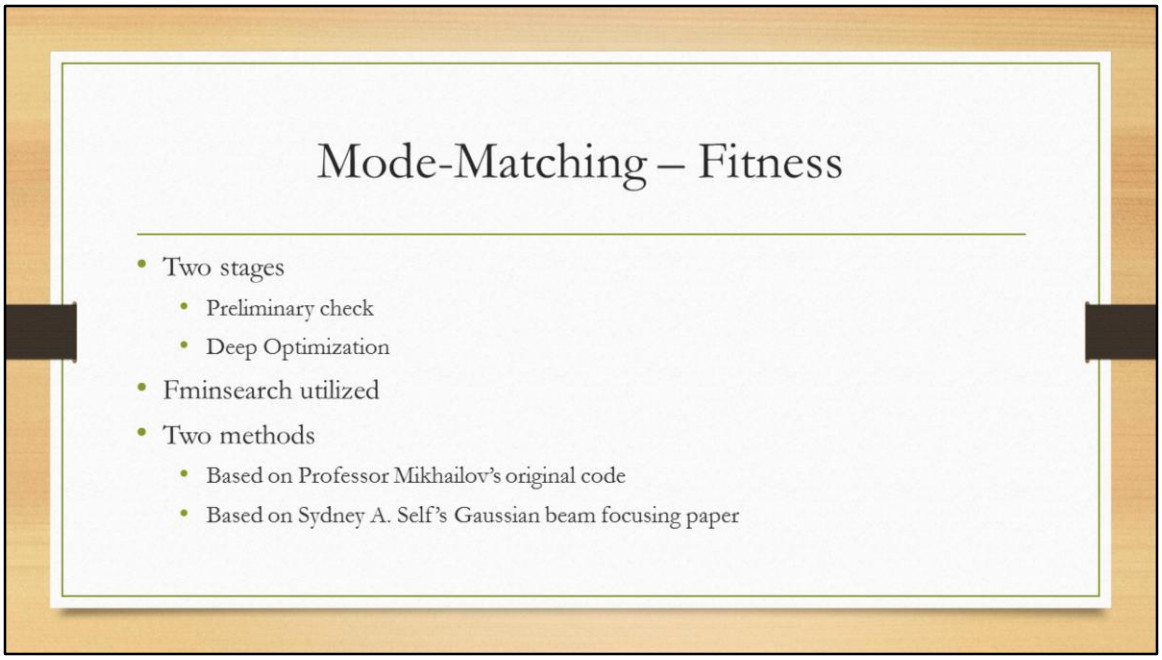


We must consider what lenses we have available to us in the lab as each laboratory setting has a different number of available lenses to them. Within the array lens_set the user defines the unique lenses available to them in the lab. An additional function is then run which permutes all possible combinations of the set and places these sets into an array labeled lens_permutations. An example of this code at work is shown here. The available lenses for this example are lenses of focal length .075 and .203. The function then permutes all possible 3 lens combinations shown in the next array. Each row corresponds to a single lens set; for example the first lens set consists of 3 lenses each with .075 focal length and the second row corresponds to the lens 1 & 2 having a focal length of .075 and the 3rd lens having a focal length of .203.



The next step is to initially place the lenses onto the optical system. A pseudo-random placement algorithm is used to properly place the lenses. By that I mean the 3rd lens is placed at focal length distance from the end of the optical system and the 1st and 2nd lens are then placed at random between the beginning of the optical system and the position of the 3rd lens. The order of the lenses are important in this case and the order is preserved when placing the lenses. This runs for a user defined n_shuffles number of times for each lens set permutation and each initial placement is collected and populates an output array.

You may notice that when placing the 1st and 2nd lens randomly we may run into the case discussed earlier of lenses accidentally being placed too close together (physically impossible cases). The algorithm checks the distance between each lens and ensures that it that the difference does not fall below some user defined value for lens_width. Lens_width corresponds to the physical size of the lens, thus if they're placed such that they end up overlapping, the algorithm simply runs the random number generator again until they're sufficiently far apart.

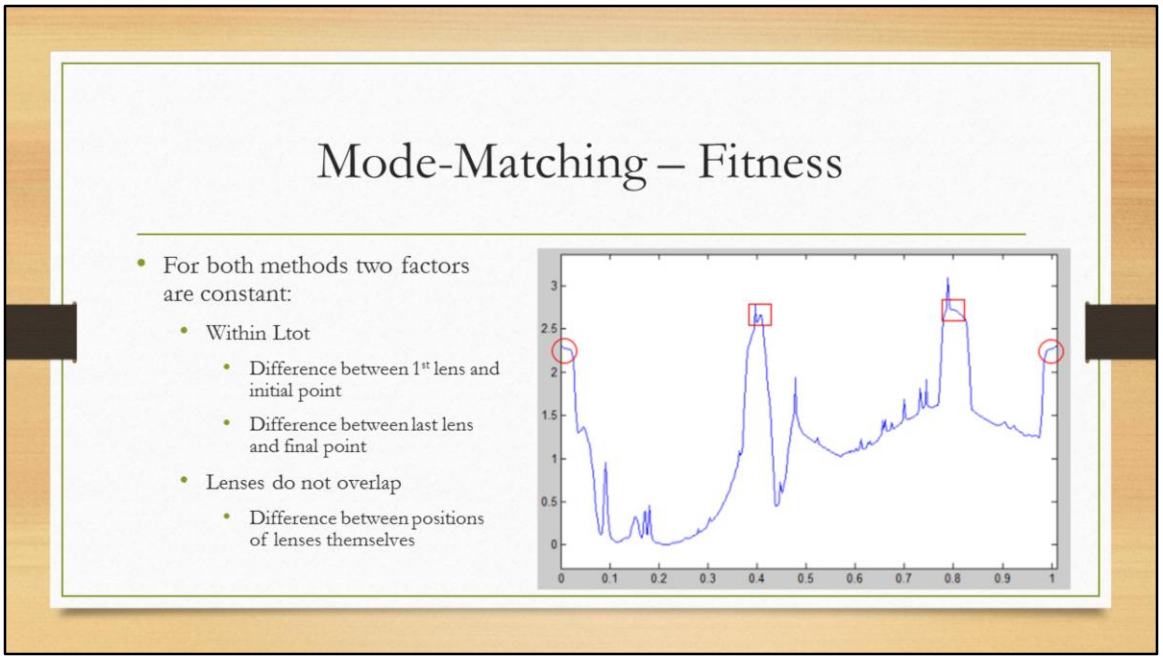


After placing the lenses we must now check the fitness of the solution– or we must now shuffle these lenses from the initial placement to output a beam that meets out desired output.

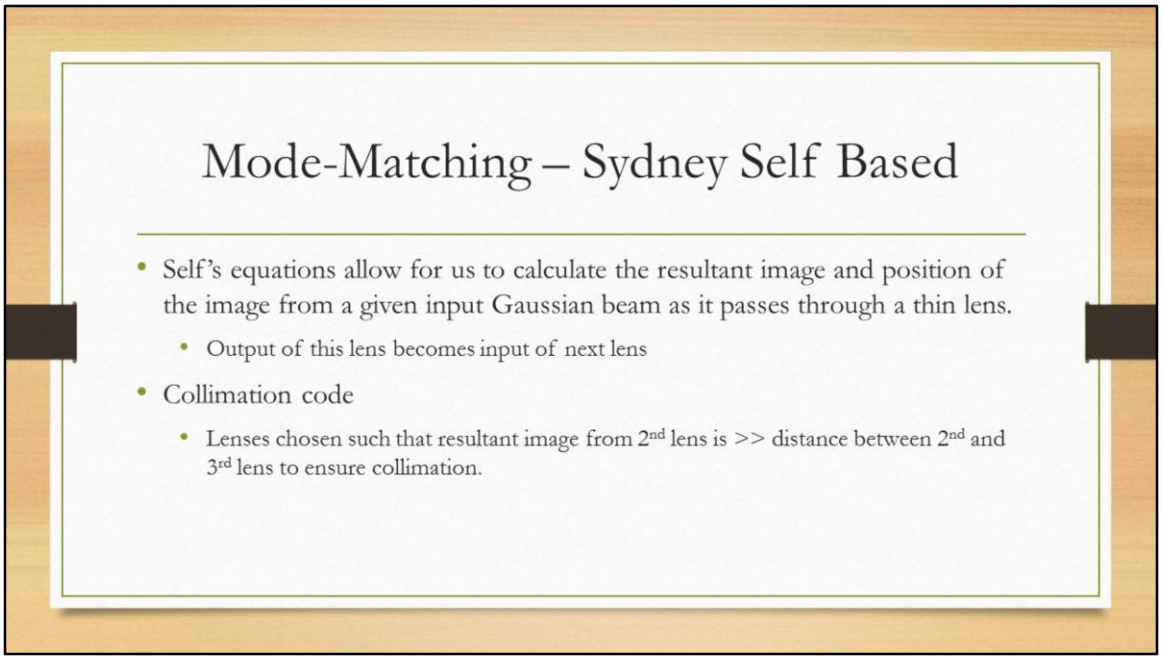
Two stages are performed to calculate the energy of a given lens set: a preliminary check and a deep optimization. This is energy in a colloquial sense and is simply referred to as such to simplify concepts of fitness (high energy correspond to less fit solutions and low energy correspond to more fit solutions).

The preliminary check uses MATLAB's built-in fminsearch function to find and minimize solutions. This serves as a rough overview of potential solutions for a given initial random placement of lenses. However, this approximation is too unrefined and suboptimal. 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. 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.

Going deeper into the fitness algorithm we have utilized two toggleable methods to calculate fitness of solutions each with their own ups and downs: one based on Prof. Mikhailov's original code and one based on Sydney Self's Gaussian beam focusing paper.

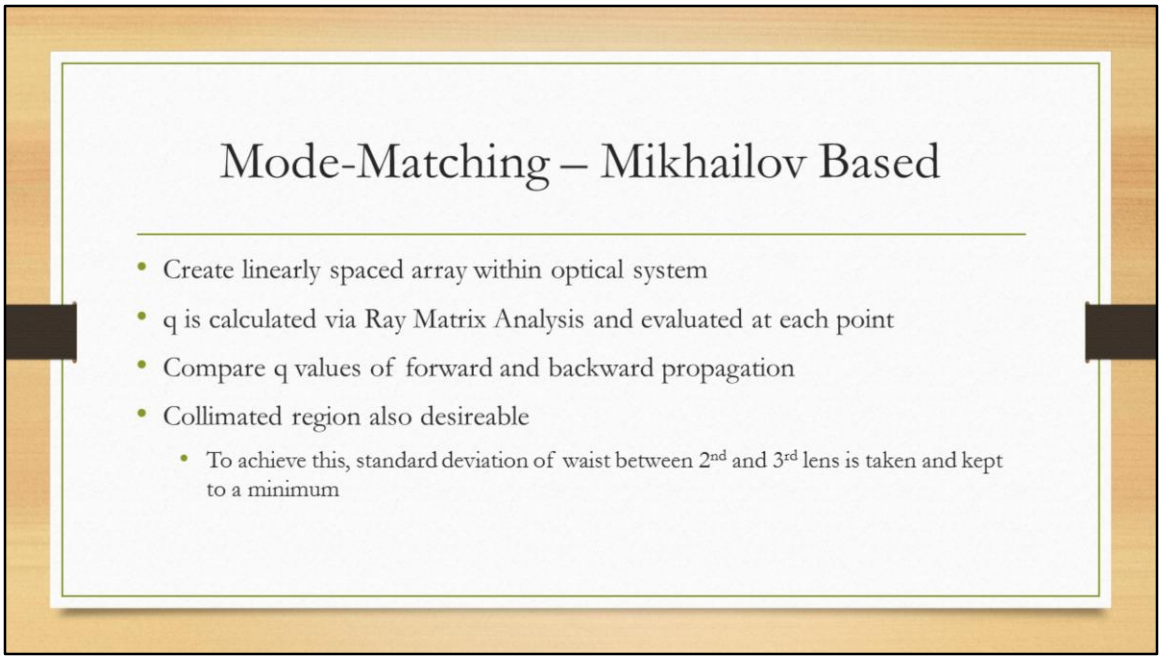


For both methods there are two factors which have the same algorithm. Checks which ensure that the solutions are within the optical system (Ltot) and that lenses do not overlap. Essentially, we check the difference between the end points and end lenses then the difference is taken between the individual lenses themselves. The figure to the right shows an example of what a fitness function for a solution might look like. In this fitness function, lenses 2 and 3 are fixed at .4m and .8m respectively and lens 1 is moved across the x-axis and its fitness is calculated at each point. We can see the penalties associated with placing the lens too close to the ends of the optical system with the red circles near the edge of the graph. We also can see the penalties of overlapping lenses with the spikes at around .4 and .8 with the red squares.



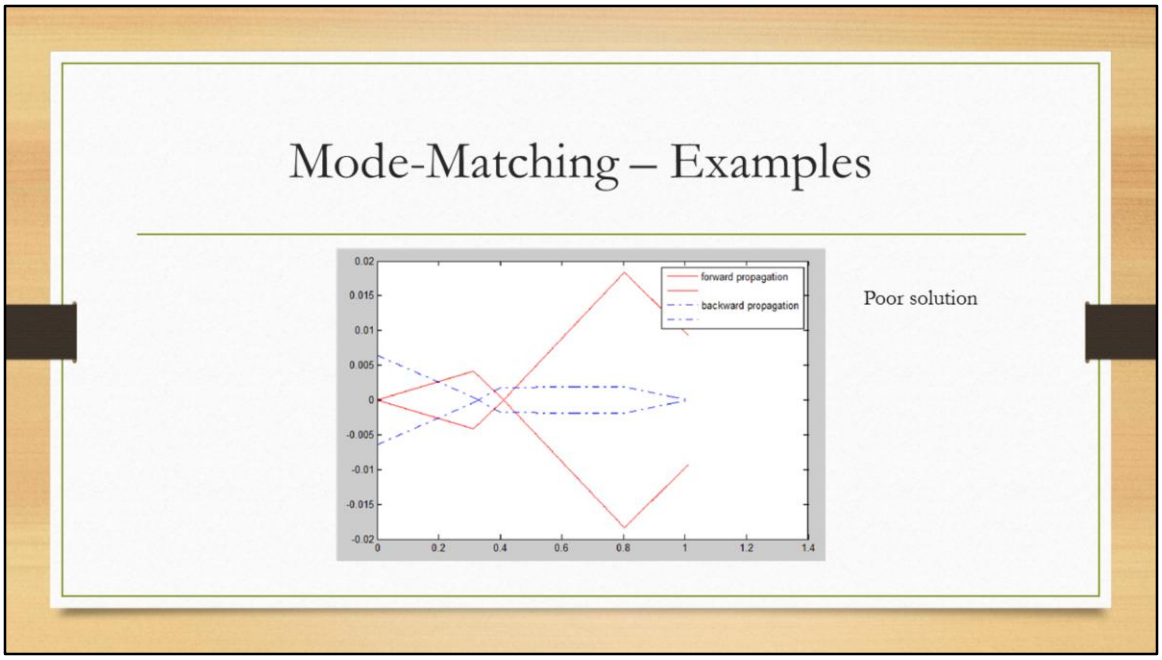
Using the Gaussian beam focusing equation shown in the introduction we are able to calculate the resultant image and position of the image from a given input Gaussian beam as it passes through a thin lens. This output is then used as the input for the next optical element and so on until it iterates through all lenses. We then determine the fitness of the solution by comparing the final waist of the solution to the desired final waist.
To ensure that a collimated region exists, we choose the preferred position of the resultant image from the 2nd lens in a 3 lens system to be far away compared to the distance between the 2nd and 3rd lens.

Which brings us to the other method's algorithm…

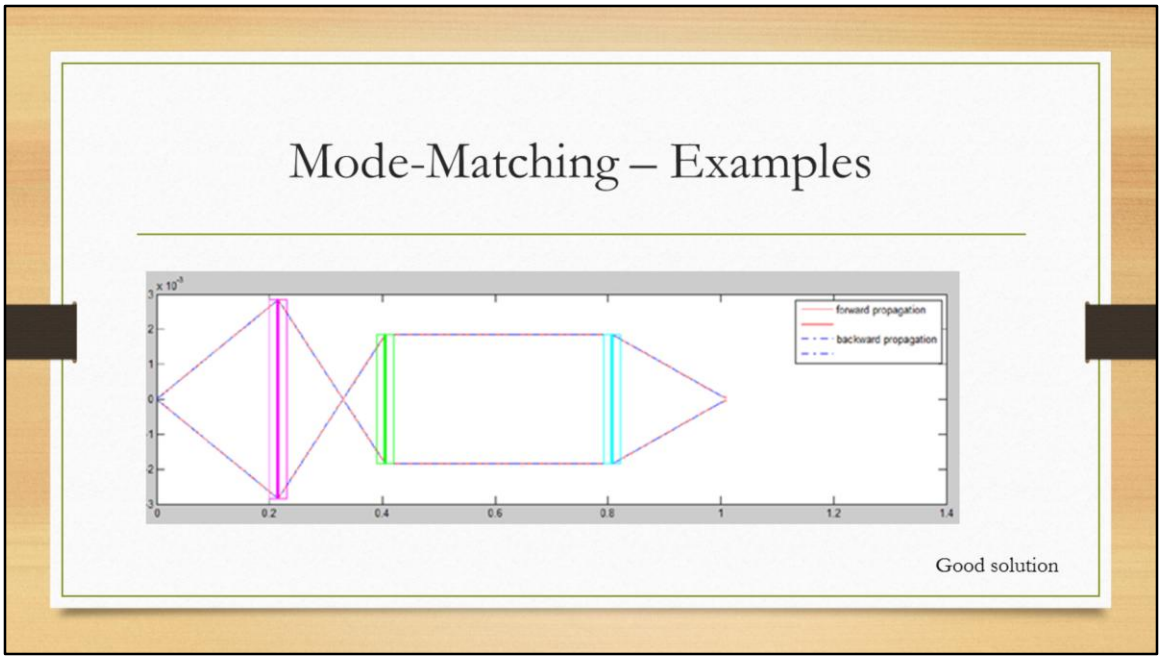


The second method of choice utilizes Professor Mikhailov's Gaussian beam propagation code. A linearly spaced array is created within the optical system. At each point in this array, we calculate the q values associated with the point in the beams propagation. Doing complete propagations of both forward and backward simulations, we then examine the q values. We want to visually confirm the beam is successfully mode-matched by matching the waists of the forward and backward propagation. 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 a good solution that won't deviate from the current solution if we, say, move one lens in one direction.

In a similar fashion, a collimated region is desired in this method. To check for collimation, we specifically looked at the intermediate q values between the second and third lens making sure the algorithm showed preference for low deviations.



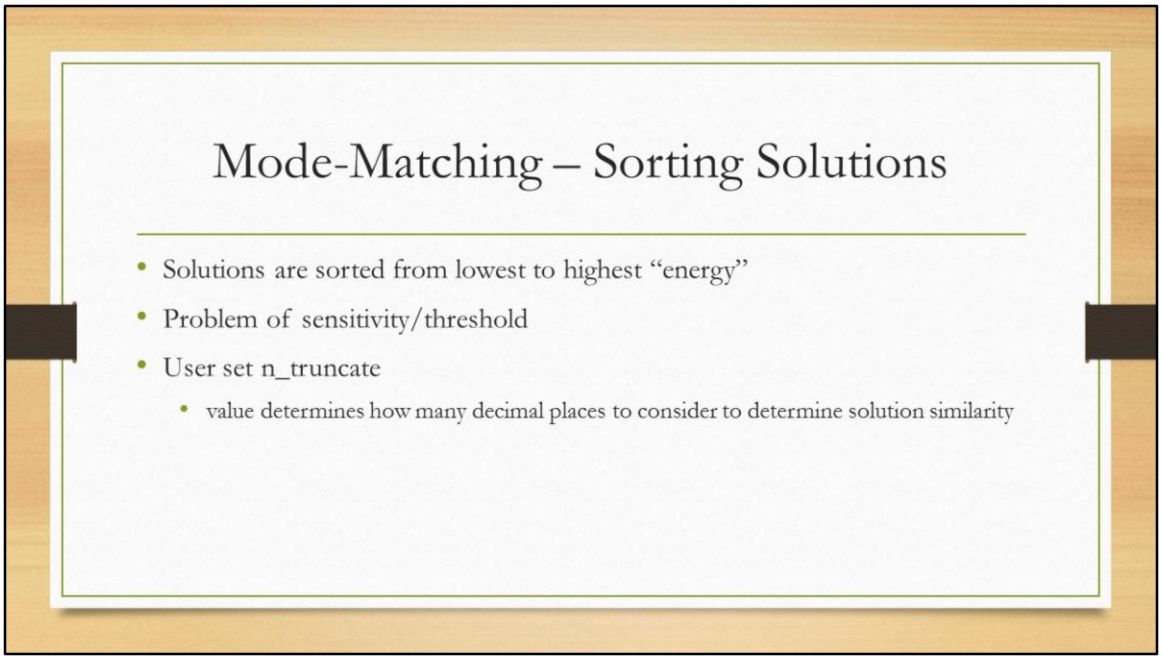
The above picture shows an example of a bad solution, the red line representing forward propagation and the blue line representing backward propagation. The y-axis is the waist from the optical axis and the x-axis is position in meters. As seen qualitatively, the forward and backward propagation of the beam do not coincide at all.



And on the other side, here is a solution that is considered good. The forward and backward propagation overlap seemingly perfectly, the lenses (at approx. .2, .4, and .8) are within the bounds of 0 to 1, the lenses are not overlapping each other, and there is a collimated region between the 2nd and 3rd lens.

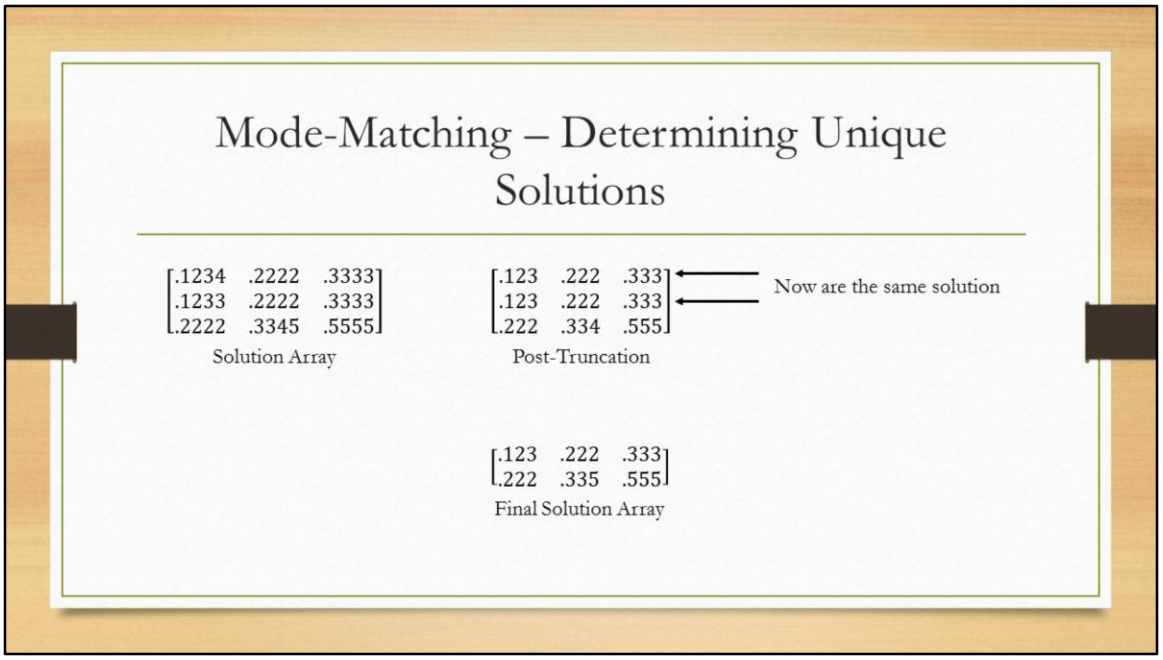
Which then brings up the questions: Why have two different methods?

Well each method has its own ups and downs. The Mikhailov based method requires the calculation of q across a very great number of points– keeping in mind that this occurs for EVERY initial placement of lenses it's easy to see that runtimes can grow exponentially for the program. Despite this, this code is the most refined as it was the initial path taken when development started and is thus likely more accurate. The Self based method requires only 3 calculations for a 3 lens solution and thus runs significantly faster than the other method. However, this was developed late in the cycle and is not as refined as the other method. For that reason, and to keep in mind future developments of this code, we made the decision to use which method toggleable by the user.



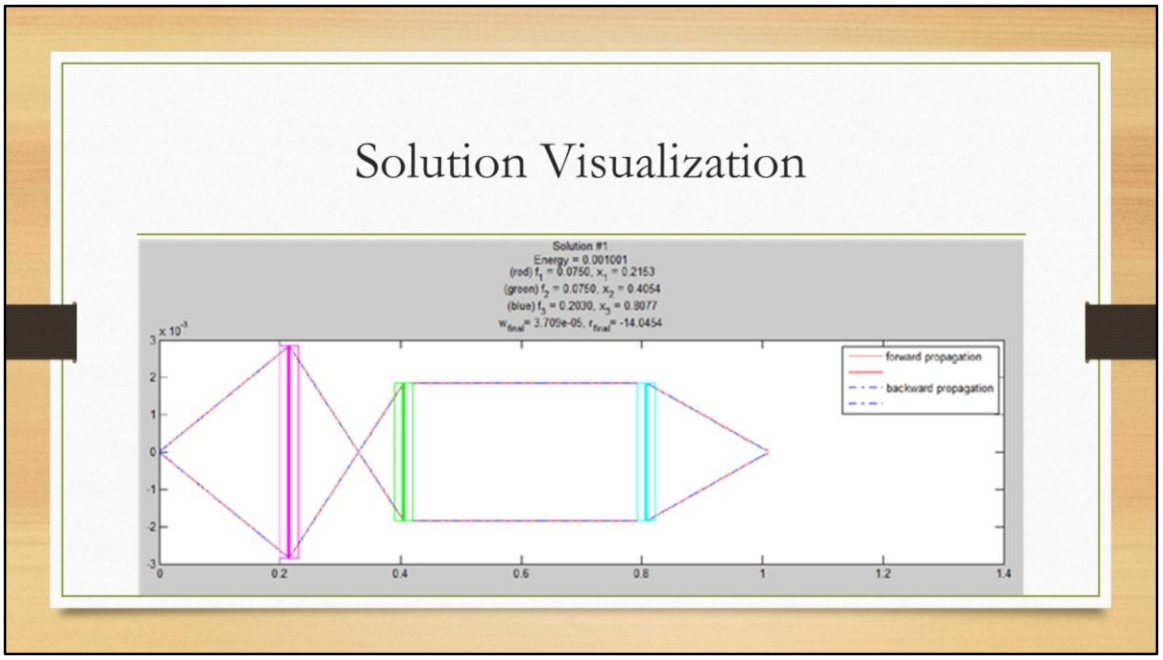
Going back to the algorithm we are now left with our minimized array of solutions arranged from lowest to highest "energy" or most fit to least fit.

There is a problem, however. Assuming a stable solution, the algorithm considers two solutions with similar beam profiles but perhaps varying slightly in the position of one lens as two separate solutions. While true that they are two separate solutions they are of no real functional use to the user as the overall profile is still the same, simply one lens has been moved slightly from its original position. Thus there is a need to set some threshold value or sensitivity by which the program determines whether solutions are the same or different. By setting n_truncate we can begin to determine unique solutions with the solution array.

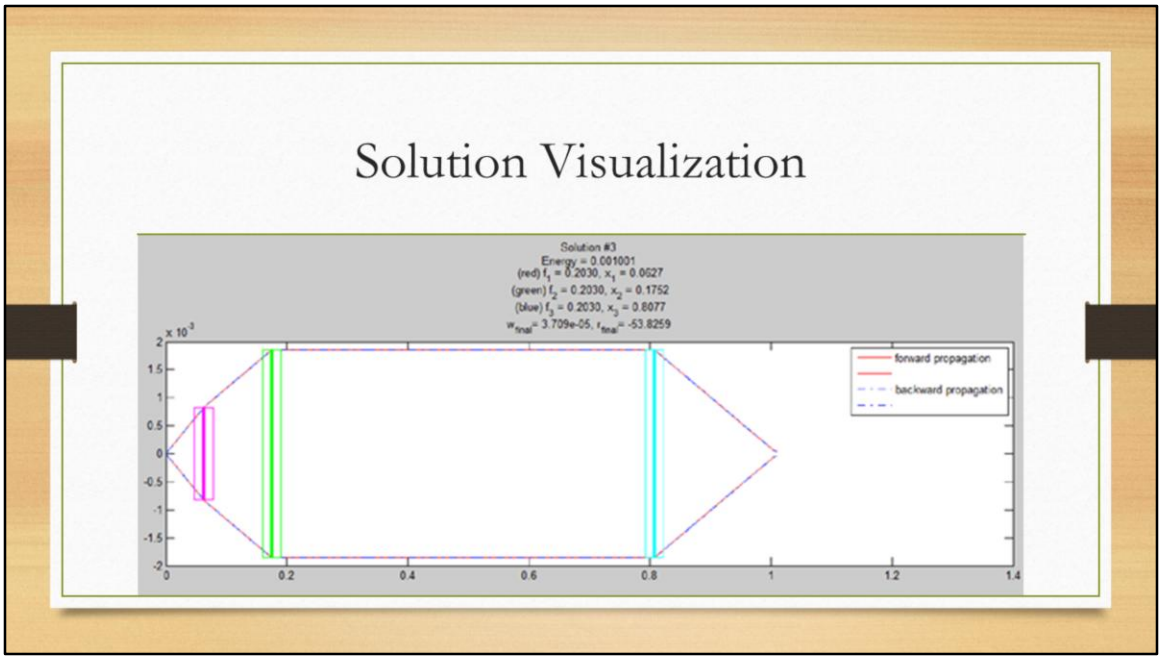


As an example of how this is determined we have an example array shown here in the top left corner. Each row corresponds to a solution with each individual element corresponding to the lens position respectively. Assuming a user set value of 3 to n_truncate this solution array truncates the solutions to 3 decimal places. The resultant array is shown in the top right labeled Post-truncation. If we look at the first and second row we now notice that these two solutions are the same. The second solution is then removed from the array to output the final solution array in the bottom left.

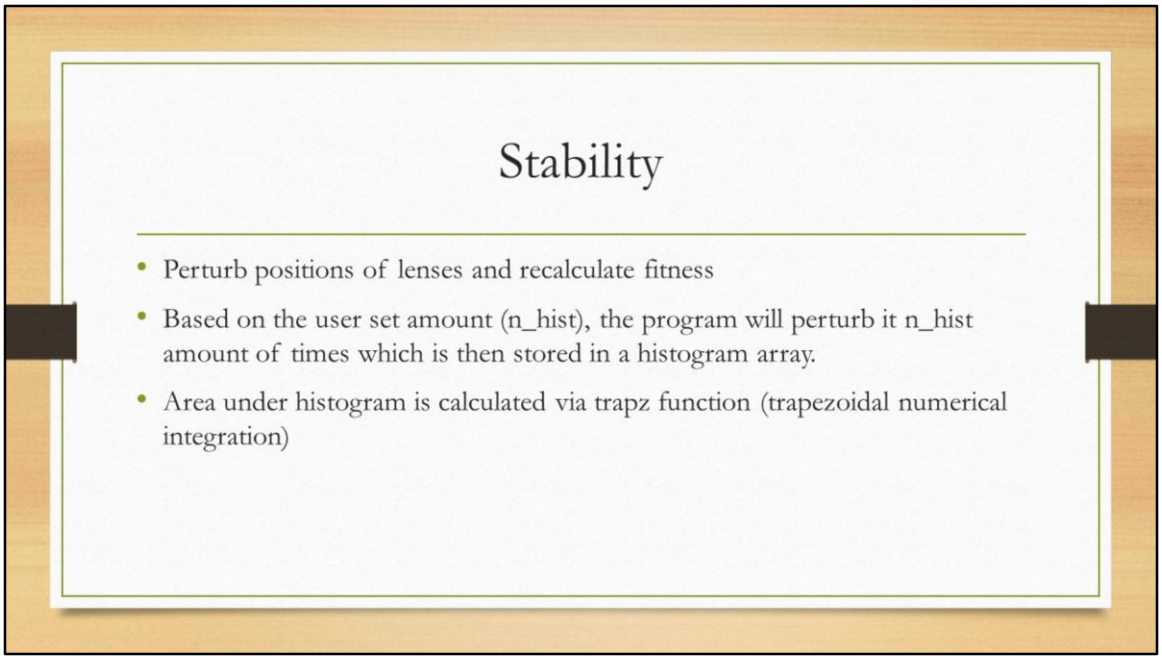
You may notice an extra array in the bottom right termed index of solution.



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 the figure.

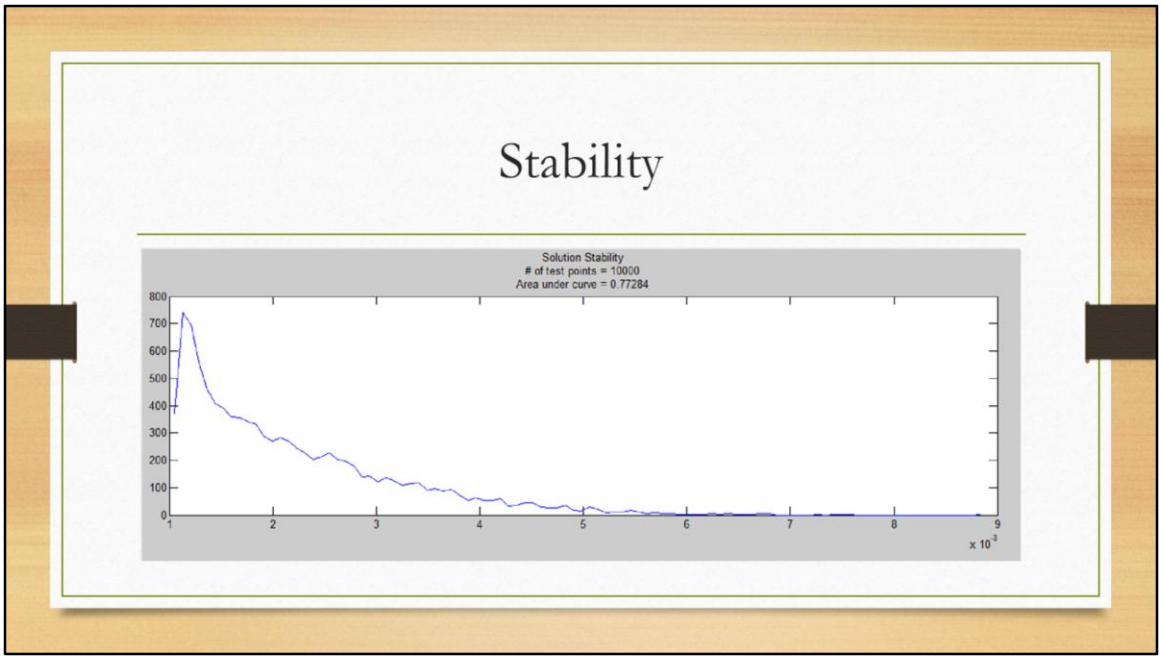


Here is another example of solution found. Note that it has a unique beam profile compared to the previous solution but still adheres to our requisites for fit functions (within system, no overlapping lenses etc.)

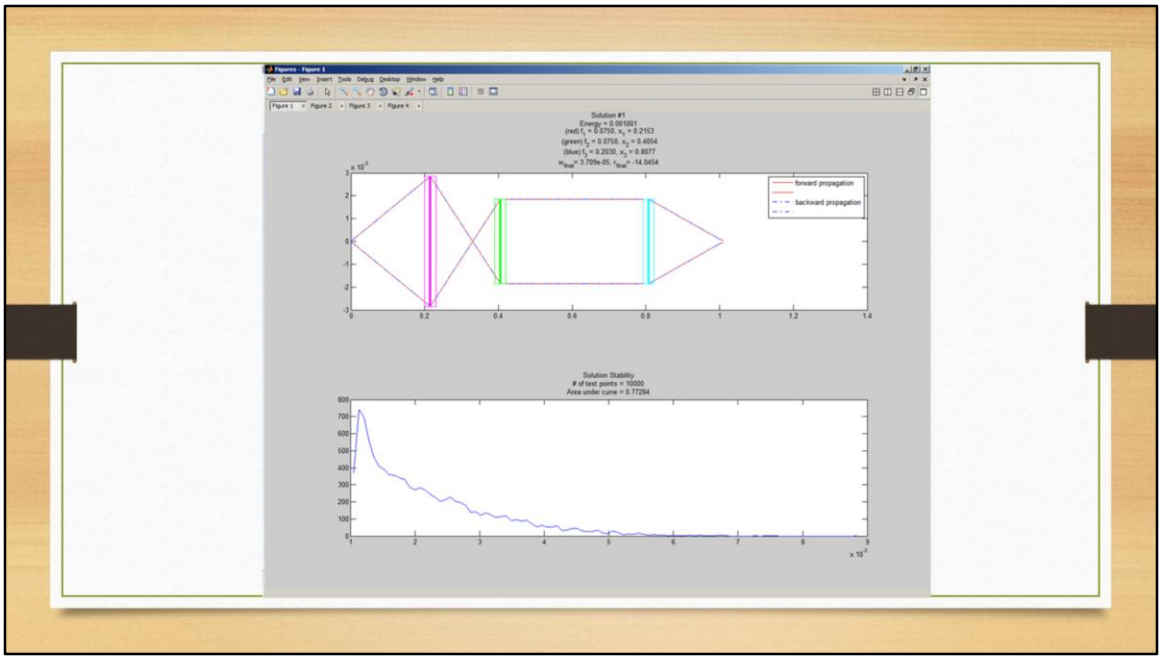


The last and final step of the mode matching process is the stability calculation and visualization. Stability is a major factor in determining whether a solution is feasible or not; a solution which is too sensitive to perturbations is seldom useful as the precision required to place the lenses in that 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.

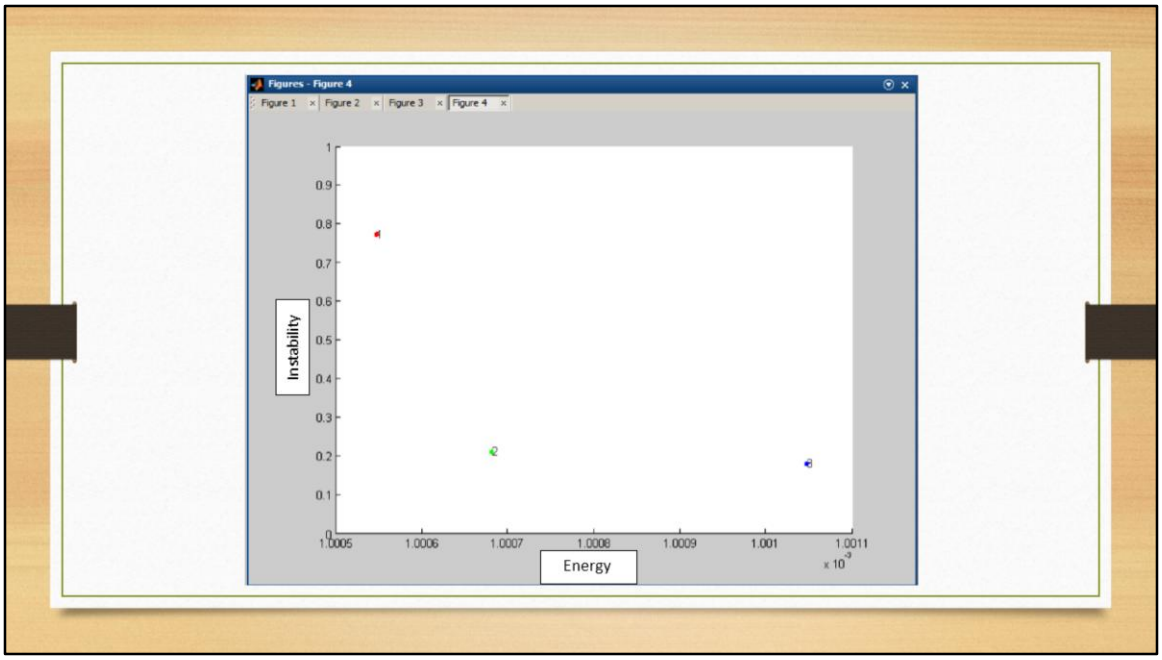
The lens positions are essentially perturbed a certain amount of times and fitness is recalculated. The recalculated fitness is then stored in a histogram array where the area under the curve is calculated via trapezoidal numerical integration.



Here is an example of what the stability of a solution may look like with area under the curve being shown in the title. Lower values correspond to more stable solutions.



These plots are then subplotted beneath their corresponding solution in order to associate them more easily. Here is what the final output for a given solution would look like for the user.



Additionally, each solution then has its energy graphed versus stability in order to compare the solutions together on one plot. This allows the user to decide whether to compromise energy for stability and vice versa, as every user has different needs from the program.

In conclusion, the automated mode matching system functions and outputs solutions which are theoretically possible and analytically sound. The program runs for ~30 seconds on average for a three unique lens set and outputting 5 different solutions. 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. Continued development of fitness algorithm would ensure proper solutions are found– though whether the Mikhailov based method or Self method should be used is up to future programmers. Additionally, the introduction of different optical elements would open the doors to solutions to more complex systems and going along with the theme of usability, an added GUI element would provide additional ease of use for those who are less familiar with programming in MATLAB.

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.