Efficient Dynamically Adaptive Mesh

Aaron Ellis

Undergraduate Research Project: Final Report
Under the Supervision of Moysey Brio and Dustin Ditchen
Fall 2004: This project is a continuation from Summer 2004

Introduction

Numerical solutions of partial differential equations are found by approximating the solution at a discrete set of points. Without using a sufficiently large set of data points we may not discover all features of the solution. In order to avoid this problem it may seem clear, add more points. However, in doing so another problem arises, computer runtime and memory may exceed practical limitations. Changing from a hundred points to cover a thousand points over the same interval could add minutes even hours to computation time. We know many solutions to PDEs only contain certain areas of interests while in the rest of the domain it varies little. There have been many attempts to exploit this feature to increase computer efficiency. We have begun to study one of these methods involving dynamically adaptive meshes.

This method involves moving points within the mesh, while doing this we are not adding more data points but instead simply focusing more points to the area of interest. To determine these areas of interests we have used a monitor function. A monitor function considers various aspects of a solution, such as the first and second derivatives using this information to change the location of points within a given domain. In order find these points we first solved for x and y values, such as in the example below [1]:

$$x_\tau = \nabla' \cdot (w \nabla' x),$$
$$y_\tau = \nabla' \cdot (w \nabla' y),$$

Where our adaptive mesh monitor function is

w=√((1+(β1*u)^2+(β2*u')^2+ (β3*u'')^2))

Goal

It is our goal to continue the ongoing research to find the most efficient and accurate way to implementing this method to problems related to optical devices. While Dustin Ditchen configured such an equation using MATLAB to find areas of interest I have begun the task of running the monitor function for efficiency and accuracy. It has been be my job to test and discover the most efficient configuration of the monitor function for these given equations in order to create an adaptive mesh that interprets a solution using as little points as possible.

Monitor in single dimension

The first testing was performed over a simple Gaussian spike. Our monitor function depends on three quantities: amplitude, first derivative, and second derivative given weights of β1, β2, and β3, respectively. My next task was running the monitor against time, testing how fast we could push the equation to run while keeping the number of necessary points at a maximum. Past work [1] had suggested using only the weight of β2 (gradient variable) within the monitor function. We felt placing the gradient, laplacian and PDE solution in equilibrium would give the most accurate and efficient dynamically adaptive mesh to the solution curve. From our results we determined placing values of equal weight for all three β values is far more efficient and faster than β2 alone at 1. We also determined that while equal weighting beta values seemed better, it was also faster and more efficient when the reference was taken from β1 and it being 10. See Table 1.

β1	β2	β3	numpts	comprat	iter
1	0	0	52	0.0234	340
0	1	0	94	0.0222	454
0	0	1	100	0.0094	262
0.01	0.0054	0	28	0.0905	155
0.1	0.05	0.01	46	0.0468	95
1	0.54	0.11	82	0.0243	141
10	5.4	1.1	96	0.021	129
10	5.9	0.9	88	0.0255	83
100	54	11	108	0.019	270

Table 1: Monitor in 1-D

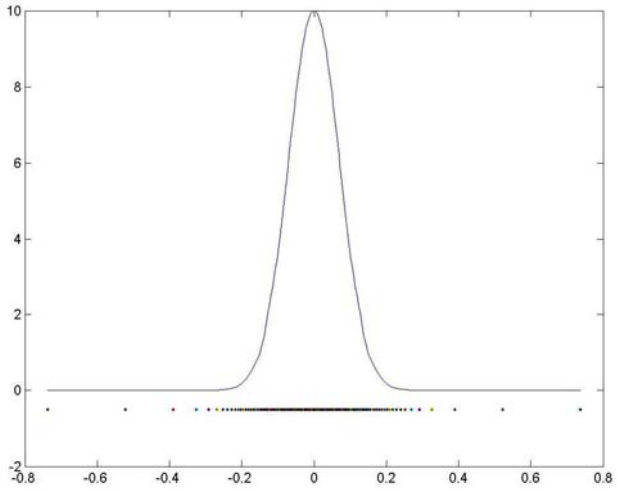

Figure 1: Graph taken from results in Table 1

It is apparent that our idea to equally weight the beta values was allowing for a faster and efficient monitoring function, but more improvement could still possibly be made. Now that we had found target beta values, I tested for a continuously changing beta function to determine new beta values for equal weight at each iteration. The results had no dramatic improvement but we still retain that continuously changing variable values could be important for use later on.

Adaptive mesh in single dimension through heat and advection equations

Discovering the targeted variable values for our monitor function allowed testing to proceed for more complex solutions. A Gaussian solution has served for the basis of our monitor function, because we knew the exact solution, and will continue to be our solution curve we test through the heat and advection equations while running our monitor. Again, we found having equal weight within the monitor during the heat equation tests produced result with minimal error. The advection equation tested very differently than expected. The most efficient results produced during the advection equation was turning off beta variables for amplitude and laplacian and leaving the

gradient variable (β2) at a value of one hundred. Some of these results can be seen below; initial values for beta are given when continuously changing reference is used. Adaptive reference is taken when finding either the max amplitude, gradient, laplacian at every interval, each beta value being adjusted after every iteration.

β1	β2	β3	Adp ref	2-norm Error
10	4.35	0.43	none	0.0051
230	100	10	none	0.0086
1	0.435	0.043	β1	0.0086
2.3	1	0.1	β2	0.0120
23	10	1	β3	0.1291
10	4.35	0.43	β1	0.0992
23	10	1	β2	0.1133
230	100	10	β3	0.1458
0	10	0	none	0.0037

Table 2: Results from Heat Equation in 1-D

β1	β2	β3	Adp ref	2-norm Error
10	5.8	1.04	none	0.6895
17.25	10	1.8	none	0.6231
1	0.58	0.104	β1	0.9927
1.725	1	0.18	β2	0.3233
9.61	5.57	1	β3	0.6938
10	5.8	1.04	β1	0.3291
17.25	10	1.8	β2	1.0917
96.1	55.7	10	β3	1.7461
0	100	0	none	0.0448

Table 3: Results from Advection Equation in 1-D

Runtime comparison for single dimension

Concluding our efficiency testing in one-dimension we followed with runtime comparisons between adaptive mesh code and standard fixed uniform code. The runtime for the heat equation was at best seven times slower than the uniform grid. This had been the case because as the heat equation runs the solution curve becomes flatter and the grid from the monitoring function becomes more like a uniform grid and has no real advantage. The runtime for the advection equation was approximately two hundred times faster than that of the uniform grid. As the spike in the solution curve moved, in order to keep with comparable error we had to increase the number of points to five thousand in the uniform grid as opposed to twenty-five for the monitor.

Monitoring Function in two-dimension

As with the monitor function in one dimension, we would test the adaptive two dimension mesh with equal weights in addition to simply turning the β variables on and off with values of one and zero. The results behaved as expected with equal weights providing far more efficient results with a minimal number of iterations. Again we saw having β1 as our reference value and with a factor of ten the best results were produced. We now felt we had sufficient values to test our monitor through the heat and advection equations in two dimensions. In addition to the Gaussian spike, we also tested the mesh over other typical optical waveguide structures.

Figures 2-6 represent results using our two-dimension adaptive monitor code to create meshes for structures common in optical waveguides.

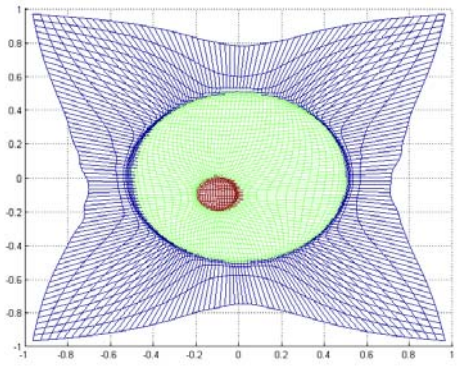

Figure 2

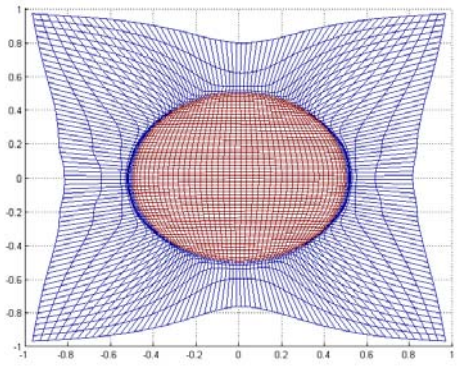

Figure 3

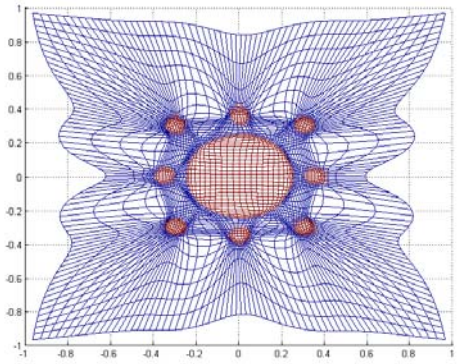

Figure 4

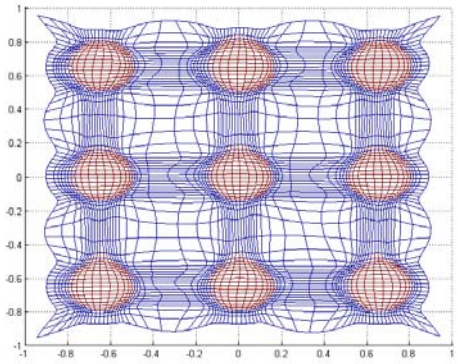

Figure 5

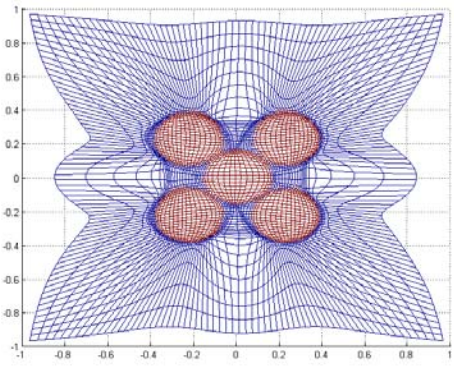

Figure 6

Heat and Advection Equations in two-dimension

Moving closer to our objective the testing procedure has become expanded to include many more situations and different factors. Beginning with simple monitoring function variables turned on and off (one or zero), we also are testing with equal weight with and without adaptive variable values, we are also constantly changing reference points and multiplication factors, adaptive time steps, different amplitude values and

number of total points in the solution, and even changing final time and converging tolerance.

The advection equation seemed not to be affected as the heat equation had previously in our monitor function. These results were much more ideal. With error values small, we were able to use these results and also find the computer runtime. Again we discovered our weighted values to be more efficient. Our previous reference point of β1 providing the best results were now joined with referencing our equal weights with β2. Both of which provided comparable efficient results with β2 providing a faster runtime. The most significant results are below:

β1	β2	β3	Adp. Ref.	Runtime	Error
1	1	0	none	98.232	2.0766
1	0.037	0.008	none	62.374	7.0845
2.726	1	0.206	none	92.685	2.5507
27.26	10	2.06	none	116.418	2.98230
13.232	4.85	1	none	115.9030	2.9866
132.32	48.5	10	none	108.2000	3.1346
2.726	1	0.206	b2	98.1840	1.4532
27.26	10	2.06	b2	121.0900	3.9314
10	1	0	b1	112.9020	1.0734

Table 4: Advection Equation in 2D

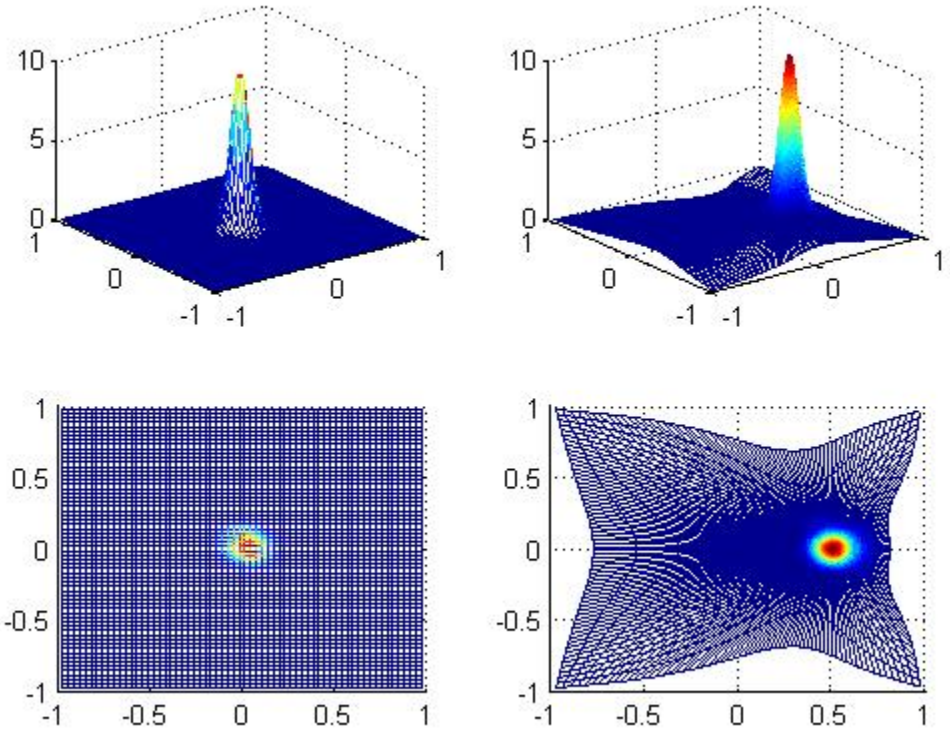

Figure 7: Graph taken from results in Table 4

Mentioned earlier, our initial testing of the heat equation in two-dimension through our monitoring function had given frustrating results. Working through the bugs and slight changes we were able to configure very useful results. Surprisingly, the results far exceeded our expectations. Initial we thought the heat equation, while flattening out, would have no need for an adaptive mesh, however to our astonishment the adaptive mesh had performed in nearly half the time, while the advection equation was accomplished at a mere fraction of the runtime for uniform grids. This was accomplished primarily with the advantage of using far less points in our adaptive grids.

β1	β2	β3	Adp. Ref.	Runtime	Error
1	0	0	none	19.093	0.2789
10	0.37	0.08	none	138.465	0.2218
13.232	4.85	1	none	274.352	0.1008
132.32	48.5	10	none	125.684	0.3132

Table 5: Heat Equation in 2D

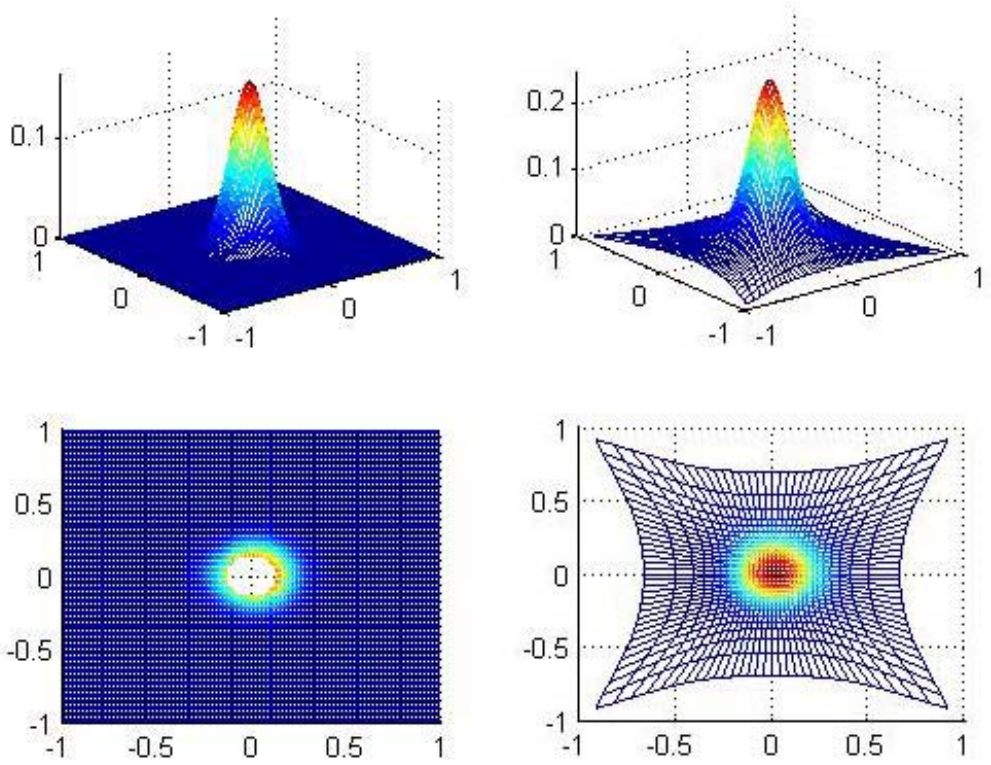

Figure 8: Graph taken from results in Table 5

Runtime comparison for two-dimension Heat and Advection equations

Finishing our efficiency testing in two-dimension, we followed with runtime comparisons between adaptive mesh code and standard fixed uniform code. As I described before our expectations for the heat equation were far exceeded. Given our results for single-dimension runtime being so much slower than the uniform grid, we had anticipated the same results in two-dimension. As we later found this was not the case at all. The heat equation ran in almost half the time as compared to uniform final runtime,

though not as much an advantage as the advection equation. The advection equation runtime was a fraction of the runtime for a uniform grid. Will maintaining similar error from exact solutions, the adaptive mesh for the advection equations were ten times faster, while using an equal ten times less grid points.

Adaptive Mesh for Optical Structures Using a Scalar Beam Propagation Method

In order to effectively run the beam propagation method through our monitor and adaptive mesh code, we needed to decide whether the monitor should consider only the solution or the index (geometry) of the optical structure. As we have done throughout our project we felt the need to combine and weight these two by variables. In our beam propagation code you will find a variable W, which when set at 1 is dependent solely upon the exact solution while when at zero it is dependent on the geometry of the structure. After testing numerous values for W, we found that a value of 0.1 gave the accuracy and lowest error, with 0.6 a close second. This value places most the weight on geometry but still utilizes the solution as well.

Limited testing has been completed for the scalar BPM. Early results have shown an odd improvement of error as a larger time step is made. Essentially we would love the adaptive mesh to handle cases of a growing time step but previous testing has revealed error grows with the time step. Boggled by our current results, improving error at larger time steps we fear is false hope. Further investigation into these results is necessary.

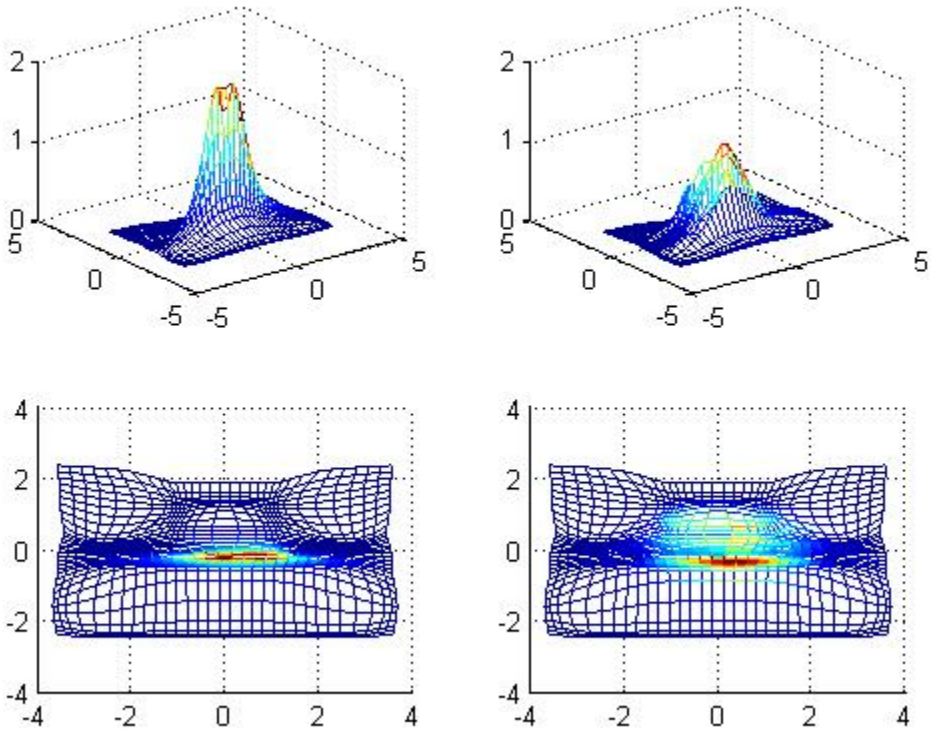

Figure 9: BPM equation in adaptive mesh

Further Testing of Increasing Time Step with the Beam Propagation Method

The interesting results I had gathered while maximizing the time step between iterations while using the Beam Propagation Method over a standard optical rib waveguide encouraged further testing and research to be done. With similar tests ran, we continued to end up in the same place as the error decreasing with an increasing in time. Many times in life we choose to over analyze a situation and this was definitely one of them. It had finally dawned on us that a larger time step, that is to say a relatively large time step, would cause the solver to virtually move nothing in the solution. Since we were using a planer waveguide for testing we know that the initial solution and final solution should in fact end in the same location and intensity. Therefore, if our solution within the model has not moved it too represents the initial condition and thus giving very little error.

Splitting the Wave Slab

Over various experimentations we were able to successfully model a straight waveguide or rib and thus comparing our final solution to the exact solution, as the initial propagation would incur no changes. We were then ready to move further and test for a wave guide that beginning as a single wave, split the signal into two and then converge back into a single wave. We see this in the real world anywhere from computer networking, optical wiring, telescope light refraction, etc. While we do expect to lose some intensity from the initial strength over the splitting, as found too in real world applications, this loss should be minimal.

As stated before our solver using the Beam Propagation Method would be weighted considering the geometry index of the solution in addition to solution values as well. I had found using a weighted value of 0.1 with 0 solely dependent on geometry and 1 dependent only on the solution value gave the least error from the exact solution. However, using this result as the weighted value, while splitting the slab, did not fair to give the smallest error. Though testing the splitting slab with previously found erroneous results from other testing is unusual, instincts were telling me different. In fact, the second lowest error value found in the straight waveguide preliminaries, W = 0.6, gave much better results. The changing W variable value was coupled with a change in beta values, a return to the monitoring function testing. Testing again these variables weighted over the amplitude and gradient in the monitoring function gave us even better results than only weighting solely on the gradient. These results are found in Table 6 and Figure 10.

β1	β2	W	Runtime	Error
0.15	0.7	0.6	68.674	0.0283
0.1	0.7	0.6	69.091	0.028
0.05	0.7	0.6	68.908	0.0284
0.01	0.7	0.6	68.814	0.0286

Table 6: Splitting Slab in 2-D

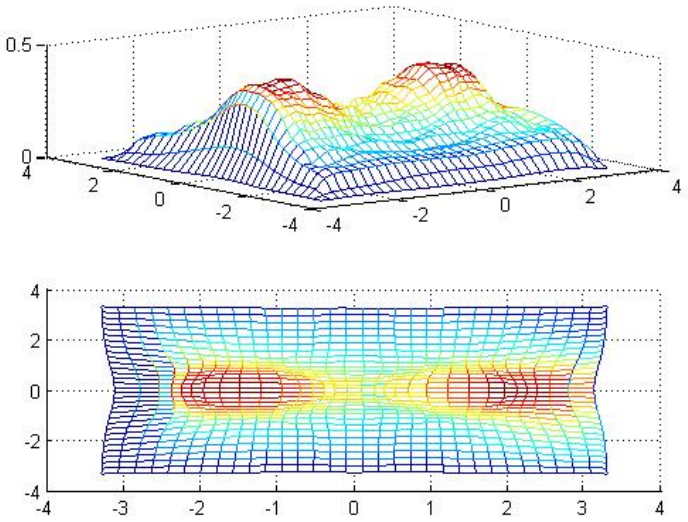

Figure 10: Splitting Slab 2D

Determining an Order of Accuracy

Moving further into our testing of the Beam Propagation Method, I also tested the waveguide in single dimension and two-dimension using a gradually decreasing time step. In addition to adjusting the time step, we made sure to keep the final time fixed so all tests under all time-steps ran to the same final time. With the previous results we received while increasing the time step, we were uncertain of what might happen with a smaller time step. Fluctuating the time-steps while comparing error values is a critical key in determining the order of accuracy, as this being the overall goal of the time step tests.

Though testing is still in its initial phase for determining the order of accuracy we expect to have at least 2^{nd} order accuracy. While testing the single dimension waveguide we were able to find a max time step of 6.3 before our solution would "blow up." As I ran tests to a half, a quarter, an eighth, etc, minimizing the time step the error though remaining small was not decreasing in a stable manner. The bouncing up and down surely indicated more testing needed to be done. For the two dimensional waveguide, results were inconclusive as well concerning the decreasing time step. Initial debugging has brought upon concerns over the boundary conditions and further changes to the code seem inevitable.

Future Work

Results for the splitting slabs in singled and two dimensions are still inconclusive and incomplete, more testing is necessary. Until we finish this testing an exact order of accuracy is still unknown. Dustin Ditchen is also currently reworking the single dimension BPM code to stabilize the model for the decreasing time step. The two-dimensional code will also need to be refactored.

Continuing the project involves more runtime comparisons for our scalar BPM over various optical structures and wave guides. We expect to finally see a large discrepancy in the favor of the monitor. Finishing these results we will begin the task of adapting our monitor to include more solutions other than the current Gaussian we are using now. Becoming more adaptable for multiple solutions is key for real world usage. As our tests are many times long and time consuming, our goal remains to eventually convert the MATLAB code into the C language to speed up the testing process and for a more user friendly design. In addition to converting the code to C, I am also in the process of designing numerous graphical user interfaces for the monitoring function and model to handle various structures and designs in multiple dimensions.

References

[1] Ceniceros, Hector D. and Hou, Thomas Y. "An Efficient Dynamically Adaptive Mesh for Potentially Singular Solutions." Journal of Computational Physics **172,** 609–639 (2001).

[2] Knupp, Patrick and Steinberg, Stanley. Fundamentals of Grid Generation. CRC Press, Inc. 1993. p. 25-43