NUCLEAR ENGINEERING 155 FINAL REPORT: 2D DIFFUSION SOLVER

May 10, 2017

Joseph Corvino and Kelsy Green
University of California, Berkeley
Department of Nuclear Engineering

1 INTRODUCTION

The neutron diffusion equation describes the averaged behavior of a large amount of neutrons in a non-multiplying medium. The purpose of this code is to write a numerical solver for the two dimensional neutron diffusion equation in MATLAB. There are vacuum boundary conditions on the bottom and left faces and reflecting boundaries on the top and right faces. We will examine the conceptual mathematical principles underlying the code, the algorithms used in the code, how to use the code, and present our code testing techniques.

2 MATHEMATICS

2.1 2D Diffusion Equation

The 2D diffusion equation is as follows:

$$-\frac{\partial}{\partial x}\left(D(x,y)\frac{\partial\phi(x,y)}{\partial x}\right)-\frac{\partial}{\partial y}\left(D(x,y)\frac{\partial\phi(x,y)}{\partial y}\right)+\Sigma_a(x,y)\phi(x,y)=S(x,y) \tag{1}$$

$$\text{where } x\in[-a,a] \text{ and } y\in[-b,b]$$

The terms in the diffusion equation are defined as follows:

$$\begin{aligned} D &= \text{ diffusion coefficient [cm]} \\ \phi(x,y) &= \text{ neutron flux } [\frac{n}{cm^2 s}] \\ \Sigma_a &= \text{ macroscopic absorption cross section } [cm^{-1}] \\ S(x,y) &= \text{ fixed neutron source } [\frac{n}{cm^3 s}] \end{aligned}$$

There are vacuum boundary conditions on the bottom and the left faces and reflecting conditions on the top and right boundaries.

$$\text{Top: } \left.\frac{\partial\phi(x,y)}{\partial y}\right|_{y=\tilde{b}}=0 \tag{2}$$

$$\text{Right: } \left.\frac{\partial\phi(x,y)}{\partial x}\right|_{x=\tilde{a}}=0 \tag{3}$$

$$\text{Left: } \phi(-\tilde{a},y)=0 \tag{4}$$

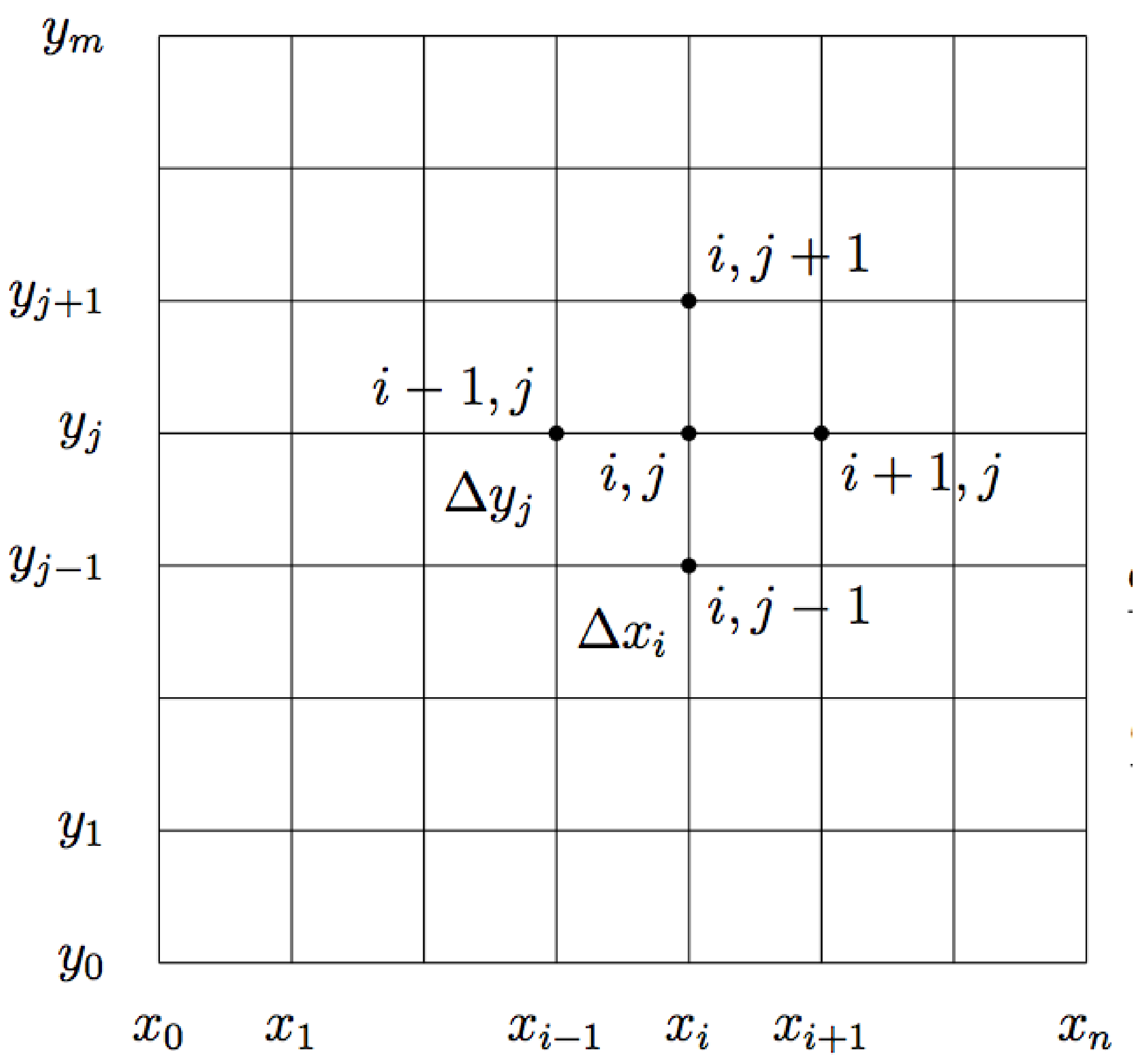
$$\text{Bottom: } \phi(x,-\tilde{b})=0 \tag{5}$$

where $\tilde{a}=a+2D\approx a$ and $\tilde{b}=b+2D\approx b$ (assume D is small relative to a and b)

2.2 Finite Volume Method

We will discretize the 2D diffusion equation by employing the finite volume method. We started with a structured 2D mesh grid with 5 points, as shown in Figure 1.

Figure 1: 5 point central differenced Laplacian.



Appendix 1 explains how to use the finite volume method in order to get the 5-point difference equation for $i = 1, \cdots, n-1$ and $j = 1, \cdots, m-1$:

$$a^{ij}_{i-1,j}\phi_{i-1,j} + a^{ij}_{i+1,j}\phi_{i+1,j} + a^{ij}_{i,j-1}\phi_{i,j-1} + a^{ij}_{i,j+1}\phi_{i,j+1} + a^{ij}_{i,j}\phi_{i,j} = S_{ij}$$

This gives us a 5-banded matrix that can be solved with MATLAB for the central points. However, the boundary conditions must also be taken into account. The vacuum boundary condition occurs along the left edge ($i = 0,\ j = 0,...,m$) and the bottom edge ($i = 0,...,n,\ j = 0$):

$$\phi_{0,j} = 0 \tag{6}$$

$$\phi_{i,0} = 0 \tag{7}$$

To account for the vacuum conditions, the matrix equations were adjusted as follows:

$$a^{ij}_{i,j}\phi_{i,j} = 0 \tag{8}$$

3 Algorithms

Implementing the finite volume method for the 2D diffusion equation, we made a matrix A using the sparse diagonal function. We then used the backslash operator to solve the system of equations. The code can account for nonuniform diffusion coefficient, absorption cross section, and source.

Within *GC_DiffSolver.m*, the cell sizes (δ_i, ϵ_j) are calculated from the x and y vectors (which is an input to *GC_Master.m*). The math derivation shown in the Appendices require symmetric conditions and so does the implementation of our code. We also check that x and y are both double class vectors that are sorted in ascending order.

For large grid sizes (large n,m), the algorithm must remain efficient and have favorable scaling properties. In double matrices, each element is stored to full precision which requires large amounts of memory, even if the matrix has few nonzero values. Hence, we employed sparse matrices as opposed to double matrices. For example, if you are using a double class matrix for A with n=1,000 and m=1,000, the matrix will take up approximately 7,500 GB when solving A/S. However, if you store A as a sparse matrix it would only take approximately 100 GB of memory storage. This is due to the fact that the sparse object class only stores the overall size (1000x1000) and the nonzero entries, which leads to large computational benefits for large diagonal matrices.

4 Code Use

4.1 How To Use

The code is structured with a main module named *GC_Master.m* that calls the different subroutines. The subroutines are Version Data (GC_VersionData.m), Input Data (GC_InputData.m), Input Echo (GC_InputEcho.m), and Diffusion Solver (*GC_DiffSolver.m*). The GC_VersionData.m writes the code name, version number, author names, date and time of execution to the output Excel file. The GC_InputData.m reads and processes the input data. It checks if the inputs of the diffusion coefficient, absorption coefficient, source, boundaries, and matrix dimensions are valid. For instance, we check the diffusion coefficient input first for its class (i.e. if it is a double or a function handle). If it is a function handle, we convert it to a double by evaluating it at the edge center (e.g. $D_{i,j} = D(x_{i-1} + \delta_i/2, y_{j-1} + \epsilon_j/2)$). We then check if it is positive, not a number, or infinity. If the input does not pass the checks, an error message will occur. The GC_InputEcho.m subroutine prints the input data for each cell to the output Excel file. The *GC_DiffSolver.m* implements the discretized diffusion solver.

The code can be found at https://github.com/TMK2017/NE155. Once downloaded, it is simple to use on MATLAB. Run the function *GC_Master.m* in the same directory as all the subroutines, and input x, y, D, S, Σ_a, and an output excel file (must be '.xlsx' format). Once the inputs are included, *GC_Master.m* will call each subroutine and output a result in an excel file. The excel file will contain author information, MATLAB version, code version, inputs, and the output (neutron flux at each mesh point). *GC_Master.m* will also output a 3D plot of $\phi(x, y)$ over the range of x and y.

5 Test Problems and Results

To demonstrate our code is correct and to present our results, we tested *GC_Master.m* with different combinations of inputs in order to test for various possible errors and situations. You can initiate our code by calling it in the MATLAB command line with your desired inputs:

GC_Master(x,y,D,S,sigma,'filename.xlsx')

x and y must be a vector (both column and row are accepted). Their elements must be in ascending order (i.e. x_0 has to be the smallest value and x_n has to the largest value).

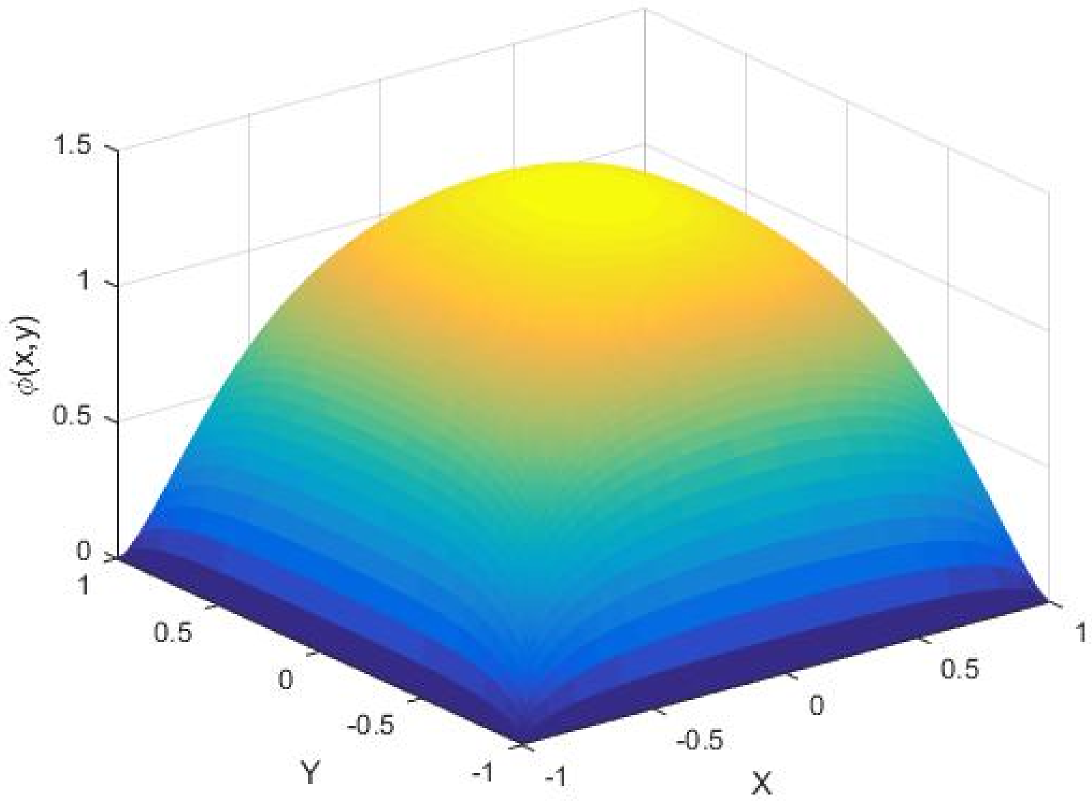
5.1 Test 1

The first input we tested was as follows:

GC_Master(linspace(-1,1,100),linspace(-1,1,100), 0.1, @(x,y) abs(cos(x)) + 2, 2,'Test1.xlsx')

This input is symmetric in x and y. It follows the original conditions postulated in the math section of this report. Thus it should run without any errors and should not take long due to the small 100x100 mesh size. This test will also check if our code can handle inputs that are function handles. The output plot from running the code is shown below. No errors were encountered. The code ran correctly, as shown in Figure 2.

Figure 2: Test 1 ran correctly and displayed a 3D graph.



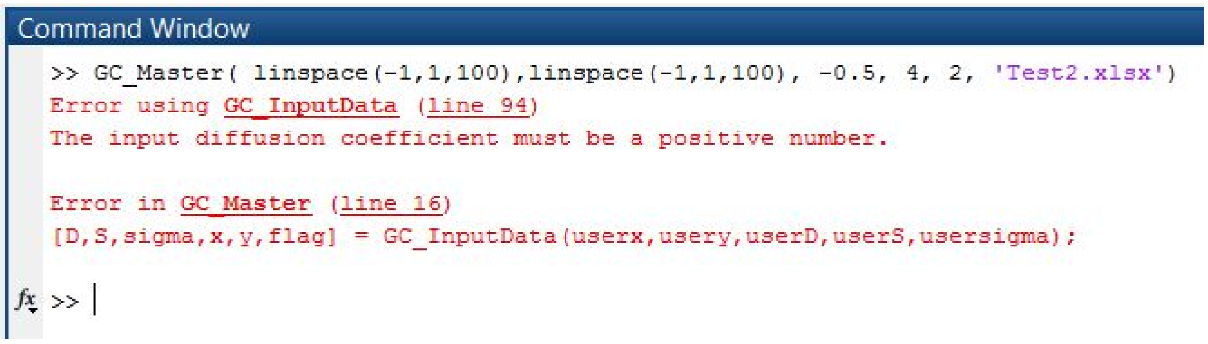
5.2 Test 2

The second input we tested was as follows:

GC_Master(linspace(-1,1,100),linspace(-1,1,100), -0.5, 4, 2, 'Test2.xlsx')

The diffusion coefficient input is negative so the output should display an error message, as shown in Figure 3.

Figure 3: Error message from Test 2 inputs.



```
Command Window
>> GC_Master( linspace(-1,1,100),linspace(-1,1,100), -0.5, 4, 2, 'Test2.xlsx')
Error using GC_InputData (line 94)
The input diffusion coefficient must be a positive number.

Error in GC_Master (line 16)
[D,S,sigma,x,y,flag] = GC_InputData(userx,usery,userD,userS,usersigma);

fx >> |
```

5.3 Test 3

The third input we tested was as follows:

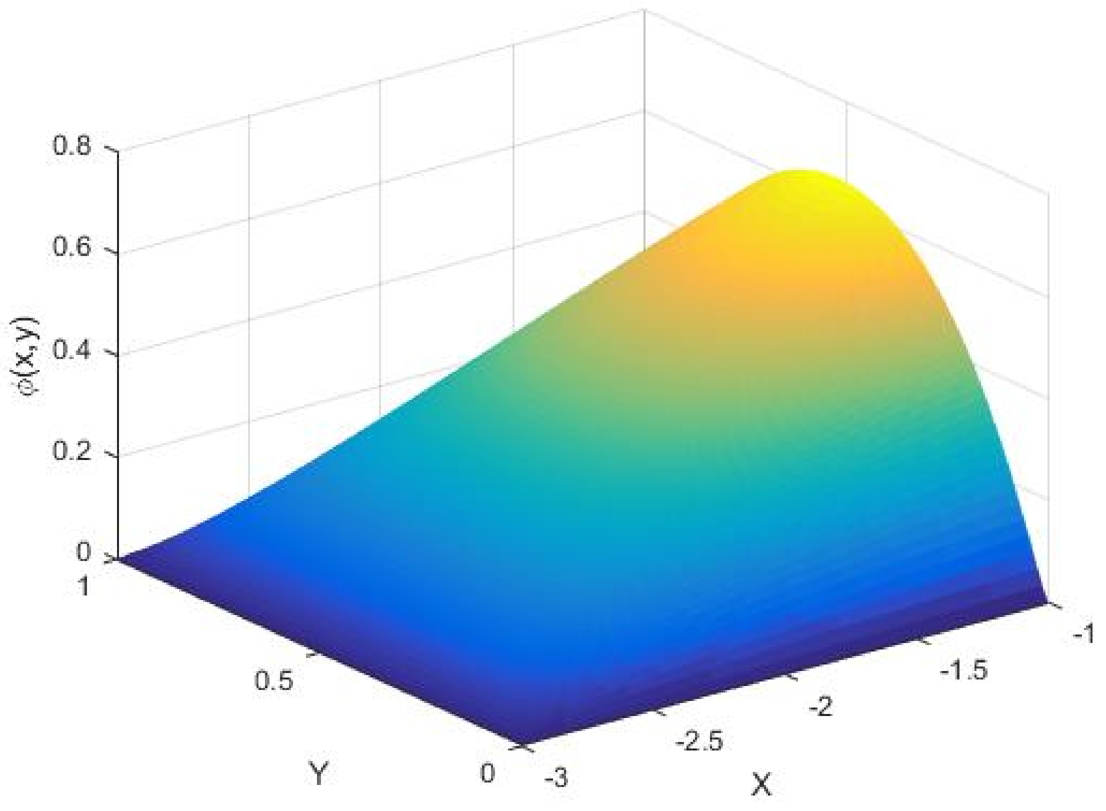
GC_Master(linspace(-3,-1,500), linspace(0,1,100), 0.1, 7, 2, 'Test3.xlsx')

This test is to check if our code can handle inputs that are not symmetric, namely that x does not go from [a,-a], y does not range from [b,-b], and n does not equal m. The output Excel file from running the code is shown in Figure 4. The code ran correctly.

6 CONCLUSION

In this report, we addressed the numerical solution to the two dimensional diffusion equation for the population of neutrons in a non-multiplying medium. Using the finite volume method for discretization and the MATLAB backslash operator, we were able to to construct a subroutine that solves the 2D diffusion equation and outputs the results to an Excel file. This subroutine among others are called from one main module.Through various tests, we validated the workings of the code. The code can be found at https://github.com/TMK2017/NE155.

Figure 4: Graph from Test 3. The highest neutron flux occurs at the peak point around x=-1 because it is hardest for the neutrons to escape from that point.



7 References

[1] Lewis, E. E., 2008. Fundamentals of Nuclear Reactor Physics, *Academic Press,* [print] Burlington, MA, USA.

[2] Slaybaugh, R., 2017. 2-D Finite Difference/Volume methods, http://rachelslaybaugh.github.io/NE155/24-2d-fd-fvm/24-2d-fd-fvm.pdf.

[3] Slaybaugh, R., 2017. 2D DE Reflecting Boundaries.$http://rachelslaybaugh.github.io/NE155/24-2d-fd-fvm/2d_DE_boundaries.pdf$

8 Appendix

8.1 Appendix 1: Finite Volume Method

We assume the diffusion coefficient, absorption cross section, and source are constant in each cell (cell-centered), e.g., for $V_{i,j}$:

$$D(x,y) = D_{i,j} \text{ for } x_{i-1} \le x \le x_i \text{ and } y_{j-1} \le y \le y_j$$
$$\Sigma_a(x,y) = \Sigma_{a,i,j}$$
$$S(x,y) = S_{i,j}$$
$$\delta_i \equiv \delta_i = x_i - x_{i-1}$$
$$\delta_j \equiv \epsilon_j = y_j - y_{j-1}$$

We do this analogously for the other three cells.

We further assume that the fluxes are constant over the interval centered around (x_i, y_j) (edge-centered):

$$\phi(x,y) = \phi_{i,j} for x_{i-\frac{\delta_i}{2}} \le x \le x_{i-\frac{\delta_{i+1}}{2}} \text{ and } y_{j-\frac{\epsilon_j}{2}} \le y \le y_{j-\frac{\epsilon_{j+1}}{2}}$$

Now, we integrate the 2D equation over 4 rectangular partial-volumes:
V = $V_{i,j}$ + $V_{i+1,j}$ + $V_{i,j+1}$ + $V_{i+1,j+1}$

$$\int_V \mathrm{dr}[-\nabla \cdot (\mathrm{D(r)}\nabla\phi(\mathrm{r})) + \Sigma_a(\mathrm{r})\phi(\mathrm{r}) = \mathrm{S(r)}]$$

Using Gauss Theorem, we can replace the first volume integral with a surface integral:

$$-\int_V \mathrm{dr}[\nabla \cdot (\mathrm{D(r)}\nabla\phi(\mathrm{r}))] = -\int_S \mathrm{dS} \cdot (\mathrm{D(r)}\nabla\phi(\mathrm{r})) = -\int_S \mathrm{dS\,D(r)}\hat{\mathrm{n}}\cdot\,\phi(\mathrm{r})) = -\int_S \mathrm{dS\,D(r)}\frac{\partial}{\partial \hat{n}}\,\phi(\mathrm{r}))$$

Next, we define the partial derivative w.r.t. direction on each surface using O(h) forward and backward difference schemes since they are simple:

$$\frac{\partial}{\partial \hat{n}}\phi(\mathrm{r}) = \frac{\phi_{i,j-1}-\phi_{i,j}}{\epsilon_j} \text{ on } S_2, S_3$$
$$\frac{\partial}{\partial \hat{n}}\phi(\mathrm{r}) = \frac{\phi_{i,j+1}-\phi_{i,j}}{\epsilon_{j+1}} \text{ on } S_7, S_6$$
$$\frac{\partial}{\partial \hat{n}}\phi(\mathrm{r}) = \frac{\phi_{i-1,j}-\phi_{i,j}}{\delta_i} \text{ on } S_1, S_8$$
$$\frac{\partial}{\partial \hat{n}}\,\phi(r) = \frac{\phi_{i+1,j}-\phi_{i,j}}{\delta_{i+1}} \text{ on } S_4, S_5$$

We then use the midpoint rule for the integration and integrate along each surface. Recall that the physics values are cell-centered while the flux is edge-centered:

$$-\int_{S_2+S_3} \mathrm{dS}\ \mathrm{D(r)}\frac{\partial}{\partial \hat{n}}\ \phi(\mathrm{r})) = \frac{\phi_{i,j-1}-\phi_{i,j}}{\epsilon_j}\left(\frac{D_{i,j}\delta_i+D_{i+1,j}\delta_{i+1}}{2}\right)$$
$$S_6 + S_7 = -\frac{\phi_{i,j+1}-\phi_{i,j}}{\epsilon_{j+1}}\left(\frac{D_{i,j+1}\delta_i+D_{i+1,j+1}\delta_{i+1}}{2}\right)$$
$$S_1 + S_8 = -\ \frac{\phi_{i-1,j}-\phi_{i,j}}{\delta_i}\left(\frac{D_{i,j}\epsilon_j+D_{i,j+1}\epsilon_{j+1}}{2}\right)$$
$$S_4 + S_5 = -\ \frac{\phi_{i+1,j}-\phi_{i,j}}{\delta_{i+1}}\left(\frac{D_{i+1,j}\epsilon_j+D_{i+1,j+1}\epsilon_{j+1}}{2}\right)$$

and we do this for each set of surfaces. To integrate absorption, we do four integrals, one over each sub-volume, and apply the midpoint scheme. Using our edge-centered flux definition:

$$\iint \mathrm{dxdy}\ \Sigma_a(x,y)\phi(\mathrm{x,y}) = \phi_{i,j}[\Sigma_{a,i,j}V_{i,j} + \Sigma_{a,i+1,j}V_{i+1,j} + \Sigma_{a,i+1,j+1}V_{i+1,j+1} + \Sigma_{a,i,j+1}V_{i,j+1}]$$
$$\equiv \Sigma_{a,ij}$$
$$\text{where } V_{i,j} = \tfrac{1}{4}\delta_i\epsilon_j$$
$$V_{i+1,j} = \tfrac{1}{4}\delta_{i+1}\epsilon_j$$
$$V_{i+1,j+1} = \tfrac{1}{4}\delta_{i+1}\epsilon_{j+1}$$
$$V_{i,j+1} = \tfrac{1}{4}\delta_i\epsilon_{j+1}$$

We use the same process for the source term to attain:

$$\iint \mathrm{dxdy}\ \mathrm{S(x,y)} = S_{i,j}V_{i,j} + S_{i+1,j}V_{i+1,j} + S_{i+1,j+1}V_{i+1,j+1} + S_{i,j+1}V_{i,j+1}$$
$$\equiv S_{ij}$$

Collecting all of the terms and separating them, we get a 5-point difference equation for $i = 1,\cdots, n-1$ and $j = 1,\cdots, m-1$:

$$a^{ij}_{i-1,j}\phi_{i-1,j} + a^{ij}_{i+1,j}\phi_{i+1,j} + a^{ij}_{i,j-1}\phi_{i,j-1} + a^{ij}_{i,j+1}\phi_{i,j+1} + a^{ij}_{i,j}\phi_{i,j} = S_{ij}$$

where

$$a^{ij}_{i-1,j} = -\frac{D_{i,j}\epsilon_j+D_{i,j+1}\epsilon_{j+1}}{2\delta_i} = a^{ij}_L \text{ (capturing influence of left flux on center cell)}$$
$$a^{ij}_{i+1,j} = -\frac{D_{i+1,j}\epsilon_j+D_{i+1,j+1}\epsilon_j}{2\delta_{i+1}} = a^{ij}_R \text{ (capturing influence of right flux on center cell)}$$
$$a^{ij}_{i,j-1} = -\frac{D_{i,j}\delta_i+D_{i+1,j}\delta_{i+1}}{2\epsilon_j} = a^{ij}_B \text{ (capturing influence of bottom flux on center cell)}$$
$$a^{ij}_{i,j+1} = -\frac{D_{i,j+1}\delta_i+D_{i+1,j+1}\delta_{i+1}}{2\epsilon_{j+1}} = a^{ij}_T \text{ (capturing influence of upper flux on center cell)}$$
$$a^{ij}_{i,j} = \Sigma_{a,ij} - (a^{ij}_{i-1,j} + a^{ij}_{i+1,j} + a^{ij}_{i,j-1} + a^{ij}_{i,j+1}) == a^{ij}_C$$

This gives us a 5-banded matrix that can be solved with MATLAB. This method works for the center cells, but we must address the boundaries. At the edges there are four entries rather than five because the edge values are known and at the corners there are only three entries.

Figure 5: An example of the matrix form of A.

$$\begin{pmatrix} \begin{pmatrix} a_C^{00} & a_R^{00} & 0 \\ a_L^{10} & a_C^{10} & a_R^{10} \\ 0 & a_L^{20} & a_C^{20} \end{pmatrix} & \begin{pmatrix} a_T^{00} & 0 & 0 \\ 0 & a_T^{10} & 0 \\ 0 & 0 & a_T^{20} \end{pmatrix} & \begin{pmatrix} \\ 0 \\ \\ \end{pmatrix} \\ \begin{pmatrix} a_B^{01} & 0 & 0 \\ 0 & a_B^{11} & 0 \\ 0 & 0 & a_B^{21} \end{pmatrix} & \begin{pmatrix} a_C^{01} & a_R^{01} & 0 \\ a_L^{11} & a_C^{11} & a_R^{11} \\ 0 & a_L^{21} & a_C^{21} \end{pmatrix} & \begin{pmatrix} a_T^{01} & 0 & 0 \\ 0 & a_T^{11} & 0 \\ 0 & 0 & a_T^{21} \end{pmatrix} \\ \begin{pmatrix} \\ 0 \\ \\ \end{pmatrix} & \begin{pmatrix} a_B^{02} & 0 & 0 \\ 0 & a_B^{12} & 0 \\ 0 & 0 & a_B^{22} \end{pmatrix} & \begin{pmatrix} a_C^{02} & a_R^{02} & 0 \\ a_L^{12} & a_C^{12} & a_R^{12} \\ 0 & a_L^{22} & a_C^{22} \end{pmatrix} \end{pmatrix}$$

8.2 Appendix 2: Boundary Conditions

8.2.1 Reflecting Boundary Conditions

The derivation for top boundary condition is as follows.

$$\left.\frac{\partial \phi(x,y)}{\partial y}\right|_{y=\tilde{b}} = 0 \tag{9}$$

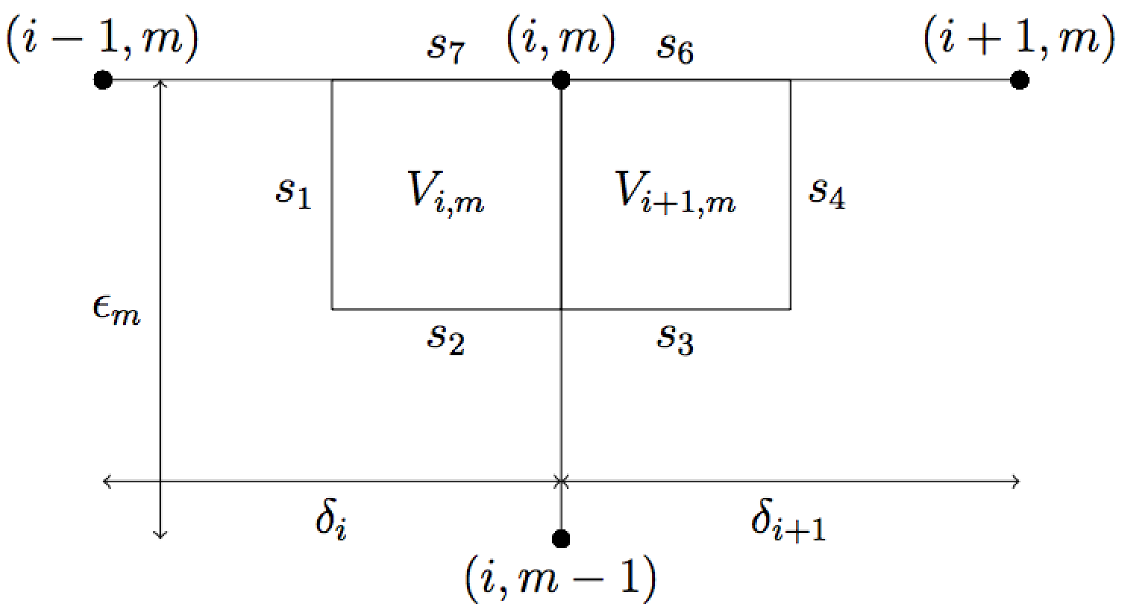
To implement the boundary condition, we integrate in the last half-cell from y = b-$\frac{\epsilon_m}{2}$ to y=b.

Previously, we used Gauss Theorem to replace the volume integral with a surface integral and we defined the partial derivative w.r.t. direction on each surface. In this case, we do not have S_8 or S_5, and we apply the zero current boundary condition to S_7 and S_6.

$$\begin{gathered} \frac{\partial}{\partial \hat{n}}\phi(\overrightarrow{r}) = \frac{\phi_{i-1,m}-\phi_{i,m}}{\delta_i} \text{ on } S_1 \\ \frac{\partial}{\partial \hat{n}}\phi(\overrightarrow{r}) = \frac{\phi_{i+1,m}-\phi_{i,m}}{\delta_{i+1}} \text{ on } S_4 \\ \frac{\partial}{\partial \hat{n}}\phi(\overrightarrow{r}) = \frac{\phi_{i,m-1}-\phi_{i,m}}{\epsilon_m} \text{ on } S_2 \text{ and } S_3 \\ 0 \text{ on } S_7 \text{ and } S_6 \end{gathered}$$

We then use the midpoint rule for the integration and integrate along each surface. The two terms that are different are for the left and right of the cell:

Figure 6: Top boundary condition.



$$-\int_{S_1} d\vec{S}D(\vec{r})\ \frac{\partial}{\partial\hat{n}}\phi(\vec{r}) = \frac{\phi_{i,m}-\phi_{i-1,m}}{\delta_i}\ (\frac{D_{i,m}\epsilon_m}{2})$$
$$-\int_{S_4} d\vec{S}D(\vec{r})\ \frac{\partial}{\partial\hat{n}}\phi(\vec{r}) = \frac{\phi_{i+1,m}-\phi_{i,m}}{\delta_{i+1}}\ (\frac{D_{i+1,m}\epsilon_m}{2})$$

The absorption and streaming terms now become:

$$\int\int \text{dxdy}\ \Sigma_a(x,y)\phi(\text{x,y}) = \phi_{n,j}[\Sigma_{a,i,m}V_{i,m} + \Sigma_{a,i+1,m}V_{i+1,m} \equiv \Sigma^*_{a,im}$$
$$\int\int \text{dxdy}\ S_(x,y) = S_{i,m}V_{i,m} + S_{i+1,m}V_{i+1,m} \equiv S^*_{a,im}$$

Collecting all of the terms and separating them, we get a 4-point difference equation for $i = 1,\cdots,n-1$ and j = m:

$$a^{*,im}_{i-1,m}\phi_{i-1,m} + a^{*,im}_{i,m-1}\phi_{i,m-1} + a^{*,im}_{i+1,m}\phi_{i+1,j} + a^{*,im}_{i,m}\phi_{i,m} = S^*_{i,m}$$

where the coefficients are now:

$$a^{*,im}_{i-1,m} = -\frac{D_{i,m}\epsilon_m}{2\delta_i}$$
$$a^{*,im}_{i,m-1} = -\frac{D_{i,m}\delta_i + D_{i+1,m}\delta_{i+1}}{2\epsilon_m}$$
$$a^{*,im}_{i+1,m} = -\frac{D_{i+1,m}\epsilon_m}{2\delta_{i+1}}$$
$$a^{*,im}_{n,j} = \Sigma_{a,im} - (a^{*,im}_{i-1,m} + a^{*,im}_{i,m-1} + a^{*,im}_{i+1,m})$$

This can be applied to the right reflecting boundary condition as well

$$\left.\frac{\partial\phi(x,y)}{\partial x}\right|_{x=\tilde{a}} = 0 \tag{10}$$

to obtain:

$$\phi_{n,j}\ [\Sigma^*_{a,nj} + \frac{D_{n,j}\epsilon_j + D_{n,j+1}\epsilon_{j+1}}{2\delta_n} + \frac{D_{n,j+1}\delta_n}{2\epsilon_{j+1}}] + \phi_{n,j-1}\ [-\frac{D_{n,j}\delta_n}{2\epsilon_j}] + \phi_{n-1,j}\ [\frac{D_{n,j}\epsilon_j + D_{n,j+1}\epsilon_{j+1}}{2\delta_n} + \phi_{n,j+1}[-\frac{D_{n,j+1}\delta_n}{2\epsilon_{j+1}}] = S_{n,j}$$

8.2.2 Vacuum Boundary Conditions

$$\phi(\widetilde{a}, y) = 0 \tag{11}$$

$$\phi(x, -\widetilde{b}) = 0 \tag{12}$$

In the code, this was implemented by assigning the following:

$$a_L^{ij} = a_R^{ij} = a_B^{ij} = a_T^{ij} = S^*_{ij} = 0$$