2D Diffusion Equation Solver with Vacuum and Reflecting Boundary Conditions

Gabe Kaufman and Kyle Leverett

May 11, 2016

NE155 Introduction to Numerical Simulation
Professor Rachel Slaybaugh
May 11, 2016
University of California Berkeley
Department of Nuclear Engineering

1 Introduction

The goal of this project is to construct a 2D diffusion solver with vacuum boundaries on the bottom and left faces, and reflecting boundaries on the top and right faces. This project will utilize both MCNP and Python, with Python serving as the primary code language and MCNP as a method of testing the output of the the Python code.

This paper will include the mathematics, algorithms, and code discussion for the solver. All of this information is intended to be used as a guide, so the methods used to create the code can be easily reproduced.

2 Mathematics

In order to produce a working numerical method of calculation, one must first understand the mathematics of the problem. The 2D diffusion equation is as follows[1]

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

Where the flux, ϕ, and diffusion coefficient, D, are now dependent on x and y. Σ_a is the absorbing term, and S is the Source term. Since our boundaries are know, we can have fixed boundary conditions which will be dependent on materials at the boundary. Computation of this system will be done on a cell by cell basis, with the terms taking on the following forms:

$$D\left(x,y\right)=D_{i,j},\qquad x_{i-1}\leq x\leq x_i \text{ and } y_{i-1}\leq y\leq y_i$$

$$\Sigma_a\left(x,y\right)=\Sigma_{a,i,j},\qquad x_{i-1}\leq x\leq x_i \text{ and } y_{i-1}\leq y\leq y_i$$

$$S\left(x,y\right)=S_{i,j},\qquad x_{i-1}\leq x\leq x_i \text{ and } y_{i-1}\leq y\leq y_i$$

$$\Delta x_i\equiv\delta_i=x_i-x_{i-1}$$

$$\Delta y_i\equiv\epsilon_i=y_i-y_{i-1}$$

Also the flux within each cell is assumed to be constant such that

$$\phi\left(x,y\right)=\phi_{i,j}\qquad \text{for } \left(x_i-\frac{\delta_i}{2}\right)\leq x\leq\left(x_i-\frac{\delta_{i+1}}{2}\right) \text{ and } \left(y_i-\frac{\epsilon_j}{2}\right)\leq y\leq\left(y_i-\frac{\epsilon_{j+1}}{2}\right).$$

Each term within the diffusion equation will now have a dependence on the surrounding cells of the simulation. Streaming terms are calculated by utilizing a surface integral and partial derivatives with respect to direction on the surface.

$$\begin{aligned}\frac{\partial}{\partial\hat{n}}\phi\vec{r}&=\frac{\phi_{i,j-1}-\phi_{i,j}}{\epsilon_j}\ on\ S_2,S_3\\&=\frac{\phi_{i,j+1}-\phi_{i,j}}{\epsilon_{j+1}}\ on\ S_7,S_6\\&=\frac{\phi_{i-1,j}-\phi_{i,j}}{\delta_i}\ on\ S_1,S_8\\&=\frac{\phi_{i+1,j}-\phi_{i,j}}{\delta_{i+1}}\ on\ S_4,S_5\end{aligned}$$

These will produce the value for streaming when integrating them through the midpoint rule.

The absorption term will assume the following form, after completing four integrals with the midpoint method

$$\int\int dxdy\Sigma_a\left(x,y\right)\phi\left(x,y\right)=\phi_{i,j}\left(\Sigma_{a,i,j}V_{i,j}+\Sigma_{a,i+1,j}V_{i+1,j}+\Sigma_{a,i+1,j+1}V_{i+1,j+1}+\Sigma_{a,i,j+1}V_{i,j+1}\right)$$

With the Volume terms equal to

$$V_{i,j}=\frac{1}{4}\delta_i\epsilon_i \qquad V_{i+1,j}=\frac{1}{4}\delta_{i+1}\epsilon_j \qquad V_{i+1,j+1}=\frac{1}{4}\delta_{i+1}\epsilon_{j+1} \qquad V_{i,j+1}=\frac{1}{4}\delta_i\epsilon_{j+1}$$

The source term takes on the following form in 2D

$$\int\int dxdy\ S\left(x,y\right)=S_{i,j}V_{i,j}+S_{i+1,j}V_{i+1,j}+S_{i+1,j+1}V_{i+1,j+1}+S_{i,j+1}V_{i,j+1}$$

With all of these equations, we can collect them to find a difference equation in order to form our algorithm.

3 Algorithms

To Discretize the formula, we can collect the terms for a difference equation:

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

The lower indexing term is for which cell you are coupling to, and the upper is the cell that the terms are in. The coefficient terms are of the following format:

$$a^{ij}_{i-1,j}=-\frac{D_{i,j}\epsilon_j+D_{i,j+1}\epsilon_{j+1}}{2\delta_i}$$

a^{ij}_L capturing influence of left flux on center cell

$$a^{ij}_{i+1,j}=-\frac{D_{i+1,j}\epsilon_j+D_{i+1,j+1}\epsilon_{j+1}}{2\delta_{i+1}}$$

a^{ij}_R capturing influence of right flux on center cell

$$a^{ij}_{i,j-1}=-\frac{D_{i,j}\delta_j+D_{i+1,j}\delta_{j+1}}{2\epsilon_j}$$

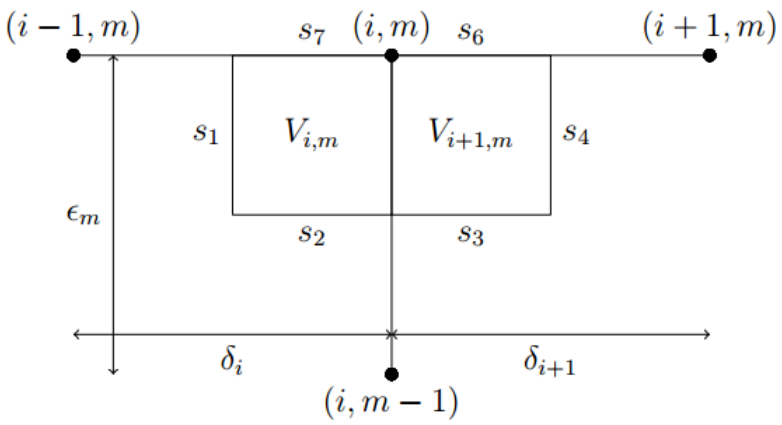
a^{ij}_B capturing influence of lower flux on center cell

$$a^{ij}_{i,j+1}=-\frac{D_{i,j+1}\delta_j+D_{i+1,j+1}\delta_{j+1}}{2\epsilon_{j+1}}$$

a^{ij}_T capturing influence of upper flux on center cell

These equations allow for the construction of a matrix of form $\mathrm{A}\vec{x}=\vec{b}$ and solve for the fluxes of each cell. In Figure 1 and Figure 2, the $\mathrm{A}\vec{x}=\vec{b}$ form of the 2D diffusion solver is included for a simple solution. The actual computation solution will be much more discrete and have many more data points.

Since boundaries do exist in this problem, the solution must utilize boundary conditions to determine behavior at the boundaries. The full derivation for the top boundary condition will be included. All of the other terms will simply have their solution form presented.



Top:

The top boundary condition is:

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

Where an integration must be done in the last half cell, or $y=b-\epsilon_m/2$ to $y=b$.The volume integral can be converted into a surface integral.

$$-\int_V d\vec{r}\left[\nabla\cdot\left(D\left(\vec{r}\right)\nabla\phi\left(\vec{r}\right)\right)\right]$$

$$=\int_s d\vec{S}D\left(\vec{r}\right)\frac{\partial}{\partial\hat{n}}\phi\left(\vec{r}\right)$$

Since the partial derivative is now defined with respect to the direction on the surfaces, the zero current boundary condition can be applied.

$$\frac{\partial}{\partial\hat{n}}\phi\left(\vec{r}\right)=\frac{\phi_{i-1,m}-\phi_{i,m}}{\delta_i}\qquad on\ S_1$$

$$=\frac{\phi_{i+1,m}-\phi_{i,m}}{\delta_{i+1}}\qquad on\ S_4$$

$$=\frac{\phi_{i,m-1}-\phi_{i,m}}{\epsilon_m}\qquad on\ S_2,S_3$$

$$=0\qquad on\ S_7,S_6$$

Using the midpoint rule to integrate, the left and right surface cells are solved for.

$$-\int_{S_1} d\vec{S}D\left(\vec{r}\right)\frac{\partial}{\partial\hat{n}}\phi\left(\vec{r}\right)=\frac{\phi_{i,m}-\phi_{i-1,m}}{\delta_1}\left(\frac{D_{i,m}\epsilon_m}{2}\right)$$

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

The absorption and streaming terms are then solved for

Absorption Term:

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

Source Term:

$$\int\int dxdy\ S\left(x,y\right)=S_{i,m}V_{i,m}+S_{i+1,m}V_{i+1,m}\equiv S^*_{im}$$

Collecting there terms and separating them produces a 4-point difference equation for i = 1,...,n - 1, j = m:

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

where the coefficients take on the following values:

$$a^{*,im}_{i-1,m} = -\frac{D_{i,m}\epsilon_m}{2\delta_i}$$

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

$$a^{*,im}_{i+1,m} = -\frac{D_{i+1,m}\epsilon_m}{2\delta_{i+1}}$$

$$a^{*,im}_{n,j} = \Sigma_{a,im} - \left(a^{*,im}_{i-1,m} + a^{*,im}_{i,m-1} + a^{*,im}_{i+1,m}\right).$$

Right:

Below are the Coefficient terms for the Right side boundary condition solution:

$$a^{*,nj}_{n-1,j}\phi_{n-1,j} + a^{*,nj}_{n,j-1}\phi_{n,j-1} + a^{*,nj}_{n,j+1}\phi_{n,j+1} + a^{*,nj}_{n,j}\phi_{n,j} = S^{*}_{nj}$$

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

$$a^{*,nj}_{n,j-1} = -\frac{D_{n,j}\delta_n}{2\epsilon_j}$$

$$a^{*,nj}_{n,j+1} = -\frac{D_{n,j+1}\delta_n}{2\epsilon_{j+1}}$$

$$a^{*,nj}_{n,j} = \Sigma_{a,nj} - \left(a^{*,nj}_{n-1,j} + a^{*,nj}_{n,j-1} + a^{*,nj}_{n,j+1}\right)$$

Left and Bottom:

The terms for the left and bottom boundaries given that both of them are vacuum conditions and not reflecting.

$$\phi(-a, y) = \Phi_L, \qquad \phi(a, y) = \Phi_R, \qquad \phi(x, -b) = \Phi_B, \qquad \phi(x, b) = \Phi_T$$

Choosing the values at the corners, we find:

$$\phi_{0,0} = \Phi_B, \qquad \phi n, 0 = \Phi_R, \qquad \phi_{0,m} = \Phi_L, \qquad \phi_{n,m} = \Phi_T$$

$$\phi_{0,j} = \Phi_L \qquad j = 1, ..., m-1$$

$$\phi_{n,j} = \Phi_R \qquad j = 1, ..., m-1$$

$$\phi_{i,0} = \Phi_B \qquad i = 1, ..., n-1$$

$$\phi_{i,m} = \Phi_T \qquad i = 1, ..., n-1$$

To determine the left boundary, i set to 0 nets:

$$a^{0,j}_{C} = 1 \qquad and \qquad \phi^{0,j}_{c} = 0$$

For i is set to 1:

$$a^{1j}_{2,j}\phi_{2,j} + a^{1j}_{1,j-1}\phi_{1,j-1} + a^{1j}_{1,j+1}\phi_{1,j+1} + a^{1j}_{1,j}\phi_{1,j} + a^{1j}_{0,j}\phi_L = s_{1j}$$

Gaussian Siedel:

The method for determining coefficient values in this matrix makes use of the Gaussian Siedel iterative solver algorithm where:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^k\right).$$

In matrix format, the GS solver takes the following form:

$$(D+L)\,\vec{x}^{k+1} = -U\vec{x}^{(k)} + \vec{b}$$

where A has been separated into its Lower, Diagonal, and Upper forms.

4 Code Use

The input file resembles that for MCNP. However, given the more limited geometry, the cell and surface definitions are adjustedi to match a simple mesh. The file starts with a title; the following lines beginning with the character 'C' are considered comments, and will thus be ignored (capitalization, for this and following characters, is not important). The first non-commented line determines the total width, number of cells in the x-direction, height, and number of cells in the y-direction (in order). This also determines the cell widths and heights, or δ and ϵ respectively, by dividing the total width and height by their respective number of cells. This sets the overall grid for the system. Non-commented lines following this input will be considered "macrocells." Since it is an inherently rectangular system, all macrocells are likewise rectangular in nature. As such, the lines start with the cell number, followed by the material number, and then the minimum and maximum x and y coordinates. Overlaps of different materials are considered combinations of their respective properties, while overlaps of the same material will simply be overwritten (i.e., it will combine the two regions). The next set of inputs will be the materials. Lines giving material properties start with the character 'M', followed by the material number (as per MCNP material definitions), the value of the diffusion coefficient D, and the macroscopic cross section Σ_a. Material overlaps from cell definitions use the inverse sum of D^{-1} and the sum of Σ_a for each material.

Finally, the input gives the user the option to define a source, designating the definition with a line starting with 'S'. This source can be defined as a constant or a function for a given x and y range. For example, using the keyword "COS_X" gives the source strength as a cosine function in the x-direction, and constant along the y-direction (more detailed definitions are given in the README). The function will always be taken as the absolute value of the desired function, such that no negative flux is given. After the function has been defined, the user has the option to determine the range. This takes the form of four numbers following the function parameters; if fewer than four numbers follow, the remaining dimensions are assumed to encompass the minimum/maximum dimensions for the grid. Multiple source definitions will combine as a product on an elementwise basis in the grid.

In order to use the code, the user needs to be in the code directory. They will also need to have NumPy and SciPy previously installed[2], as the program makes use of both packages. The proper call to the code takes the form of "python diff-2d.py [-h] [-p] [-g] [-s] infile [outdir]." The input file, or "infile", is the only

required input, and can be located in a directory outside of the source directory (so long as the user properly defines the path). The user also has the option to define an output directory, where the output file will be written. If it is not defined, the output file will be written to the same directory as the input. The file itself will have the same name as the input file, but terminated with a ".out" filetype (any other filetype suffixes will be removed). The rest of the inputs are flags:

- "-h" or "--help" – Print input definitions as well as code description.
- "-p" or "--print" – Only print inputs to output file; do not carry out any calculations.
- "-g" or "--graph" – Display heat map of calculated flux (ignored if "-p" is flagged).
- "-s" or "--sgraph" – Display heat map of calculated source term, and print source vector to file.

Following the call, the code will write the following information to the beginning of the output file (as well as print them to the screen): the code title and version number, author names, the input file used, the input file's title, and the start time. The code will then print status updates as it parses the input file, as well as write the parsed information (e.g., cell/material definitions) to the output file. If any errors arise, such as incorrect input file formatting or nonsensical cell boundaries, the code will stop prematurely and output the error to screen, and no output file will be written. Once the source is parsed, if "-s" is flagged the source vector used in the equation will be written to file starting from the line "Source output: ", and will consist of three columns: the x-coordinate, y-coordinate, and normalized source strength (based on the mesh). If "-p" is parsed, the code will terminate, and the output file will be closed. Else, the code will then build the matrices necessary for computing the flux, again printing its progress (including the "macro-rows" of the coefficient matrix, each of which is built of n $m \times m$ matrices for an $m \times n$ mesh). During the actual solution process, or the Gauss-Seidel subroutine, the code will print its current iteration as well as the relative error, which is determined from matrix norms:

$$\text{Relative error} \equiv \frac{\|\mathbf{A}\vec{\phi} - \vec{S}\|}{\|\vec{S}\|}$$

By default, the code will attempt to get the relative error below 10^{-4} before stopping. If it reaches a standstill, i.e. the error term remains stagnant with 100 successive iterations, the code will stop regardless of its proximity to this limit. Else, for larger meshes the user may wish to end the program prior to reaching this limit. In this case, using a keyboard interrupt ("Ctrl+C") will end the subroutine prematurely, and use the current flux vector as the final result. Note that this is the *only* time a keyboard interrupt can be used without ending the entire code as a whole; used elsewhere, it will act as normally intended, and no output file will be written. Following this calculation, the number of Gauss-Seidel iterations processed and final relative error are written to the output file. The flux vector will follow, in the same format as the source vector described previously: starting at the line "Flux output: " the x-coordinate, y-coordinate, and flux will be tabulated in a list. Finally, the end time will be printed to screen and written to the output file, and the file will be closed. If any plotting flags are enabled, these plots will be drawn with the appropriate titles (i.e., "Source Strength" for source and "Flux" for flux) using PyPlot. Closing these windows will cause the program to terminate; else, the program will terminate as soon as the output file is closed.

5 Test Problems and Results

In order to test the validity of the deterministic code, a simple geometry was created in MCNP. MCNP requires that a system be 3D to run, so a geometry that matches the x and y coordinates of the tested deterministic code while limited z dimension was utilized. These various examples used both simulations to test for accuracy of the deterministic code. All of the simulations that were run were plotted, and the plots are included in the Appendix section of this report.

The first two simulations, Figure 4 and Figure 30x30, were run with a 10 cm by 10 cm square geometry meshed into 30 x 30 cells. The flux values of both trend upward toward the top and right surface, which is likely due to the reflector. Both simulations are in agreement, so the deterministic model is likely working properly.

The next two simulations, Figure 5 and Figure 6, utilize a point source instead of a plane source and a 100x100 meshing. As with the previous set of simulations, both MCNP and the deterministic model are generally in agreement. We can now assume that both the plane source and point source in the deterministic model are functioning properly.

Since the MCNP and Deterministic model are in general agreement, the following simulations were carried out using only the deterministic model. Each was design to test whether a certain input parameter was operating properly.

The specific simulation, Figure 7, was run with the meshing of x and y set to different values. The mesh mxn size was set to 20 by 30. Since the code initialized and produced results in agreement with the 30x30 meshing, we can assume the A matrix for the values of the coefficient is properly initializing.

The next two sets of plots, Figures8, 9, 10, and 11, are included to show how different source terms operate in the deterministic function. Figures 8 and 9 show a linear source that decreases across the gradient from (0,0) to the extreme. The flux is greater close to (0,0) but still is significant close to the reflector boundary. Figure 10 and 11 show a cosine source. The right side reflector maintains flux better than the left vacuum.

6 Conclusions

The deterministic 2D diffusion solver creates iterative solutions to set mesh problems with consistent relative error across all cells. This is useful to create simulations with higher accuracy across all of the cells. MCNP provided a very useful benchmark to test the simulation against. Considering the deterministic solution had agreement with two different source terms, it seems to function appropriately.

References

[1] Slaybaugh, Rachel. "2-D Finite Difference/Volume Methods." (n.d.): n. pag. 16 Mar. 2016. Web. 12 Apr. 2016.

[2] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011).

7 Appendix

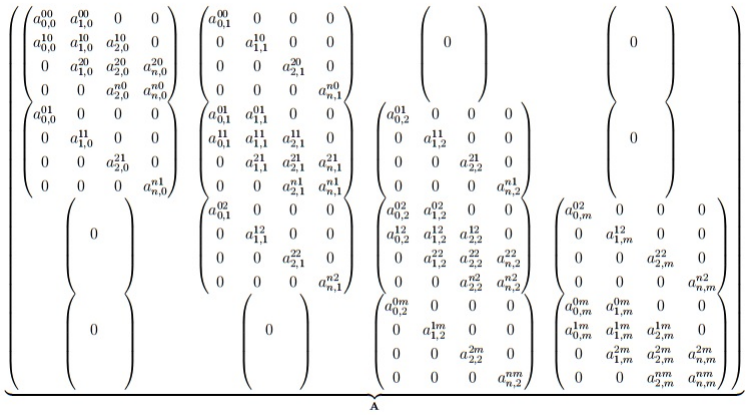


Figure 1: Coefficient Matrix for 2D Solution

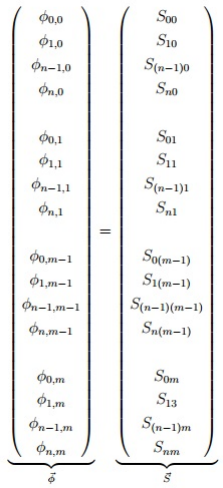


Figure 2:

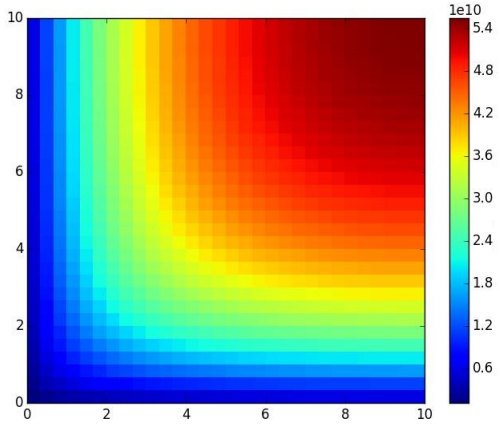


Figure 3: Deterministic Place Source, 30x30 meshing, 10cmx10cm

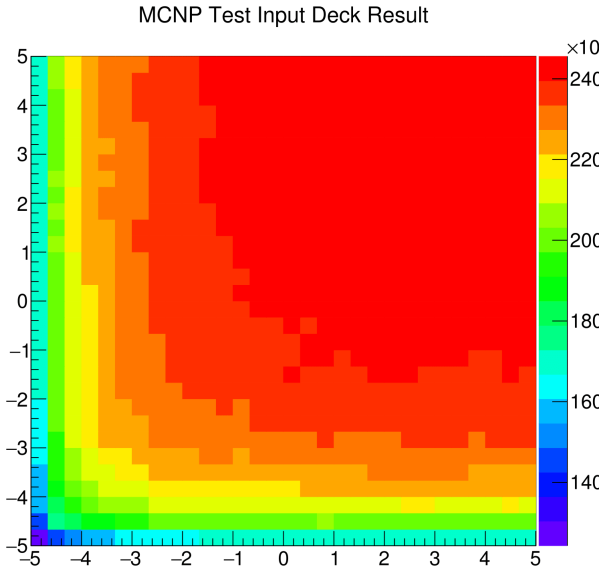


Figure 4: MCNP Plane Source, 30x30 meshing, 10cmx10cm

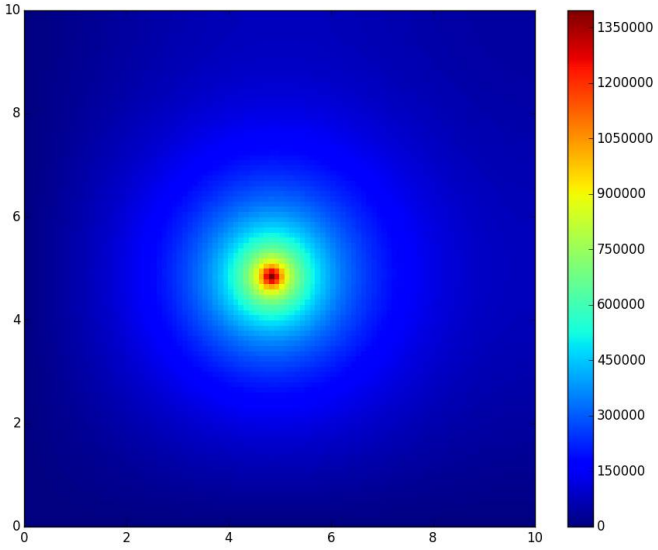


Figure 5: Deterministic Point Source, 100x100 Meshing

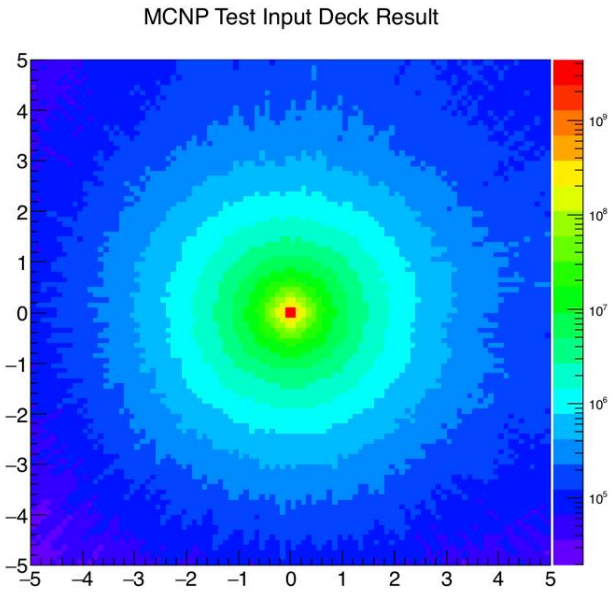


Figure 6: MCNP Point Source, 100x100 Meshing

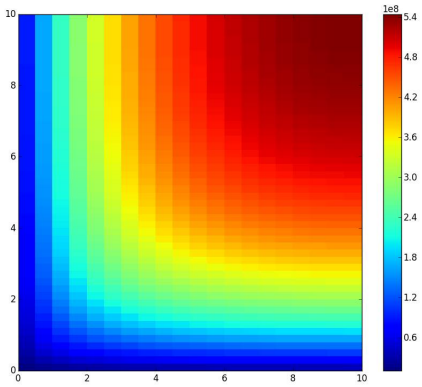


Figure 7: Deterministic, 20x30 Meshing

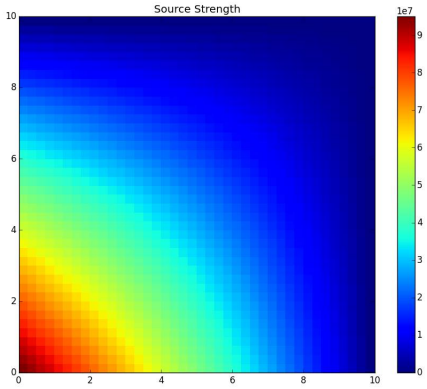


Figure 8: Plot of Linear Source Strength

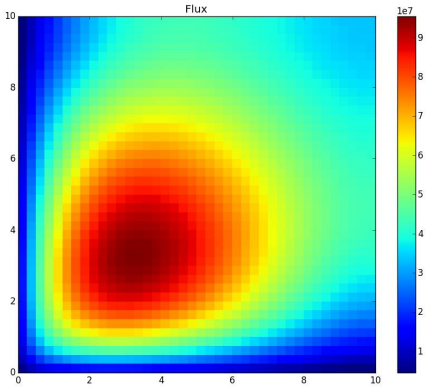


Figure 9: Plot of Linear Source Flux

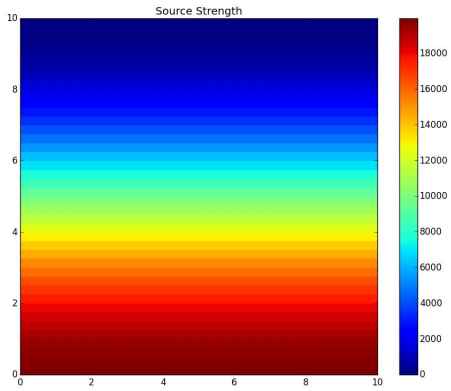


Figure 10: Plot of Cosine Source Strength

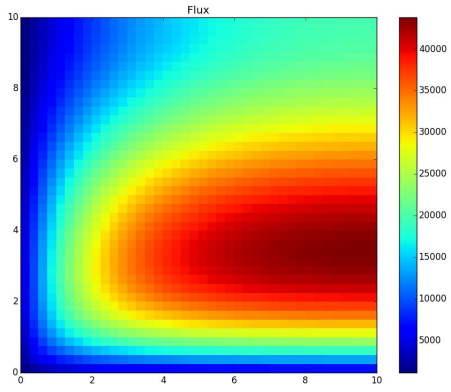


Figure 11: Plot of Cosine Source Flux