Will Kable

5/9/2016

Final Report

2D Diffusion Solver Final Project

**Introduction:**

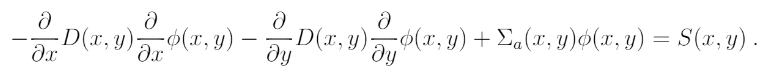
I wrote a python script that solves the time-independent 2D Diffusion Equation with various vacuum boundary conditions on the bottom and left faces reflecting boundary condition on the right and top faces. In my code I am able to take any 2D rectangular geometry with various sources configurations and model neutron diffusion through single composition materials. In my code, I have created a graphical user interface that can take these system configurations and output various maps of the neutron flux across the surface. In this paper, I will talk more about the methods used to create my solver while also discussing some of the intricacies in my code.

**Mathematics:**

**Defining Mesh:**

The version of the diffusion equation that I am using operates under the assumptions of steady state conditions and that any source we model is isotropic.For my code I am also assuming that the neutrons are mono-energetic and the absorption and transport cross-sections are constant throughout the material. The following equation is used to solve for 2-deminsional flux.

Equation 1



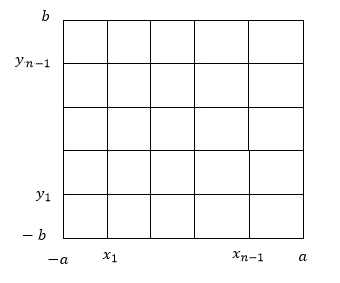
Ref. 1

We can define each of these terms by the following in terms of the created mesh:

I then defined an evenly spaced mesh on the interval of (-a, a) and (-b, b). This forms an

N x N cell grid that will define our system. An example mesh grid is shown below.

Figure . N X N Mesh

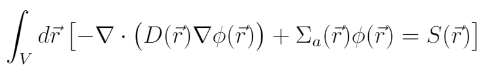


As stated before we assume that absorption and transport cross-sections are constant over the entire grid. With our system defined we will then use the finite volume method to solve *Equation 1.* for our system.

**Finite Volume Method:**

In order to solve for the flux of these systems we will eventually need to integrate the equation over the 4 cell volumes centered around. To do this we simply add the volumes of the surrounding cells:

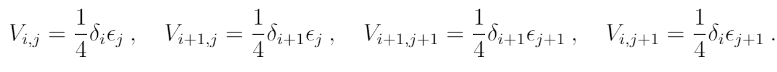
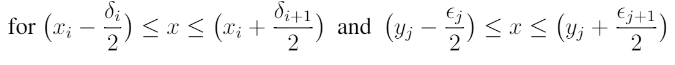
The 2D diffusion equation can then be defined as follows:



Ref. 2

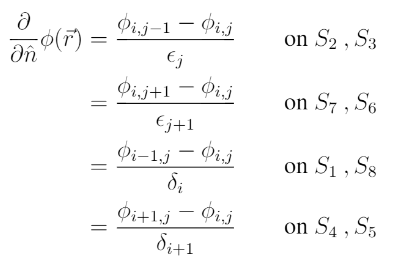
Equation 2

With these equations we can determine the flux from a given source through on our mesh grid. These points form mesh areas that can then be represented through the following



Ref. 3

By using Gauss’s Theorem we can then replace these volume integrals with surface integrals. We can then define the partial derivatives in each integral term with the following using the midpoint rule:



Ref. 4

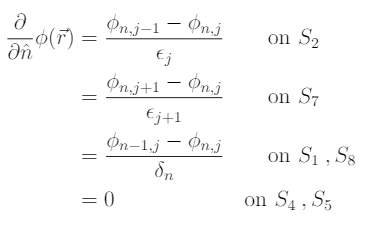
After integrating each term in equation two with the above substitution we can reduce our equation down to the following discretized equations:

Equation 3

Ref.5

These discretized equation are ones that we can put into a tridiagonal matrix that we can solve. However, first we must impose boundary conditions on the system.

**Boundary Conditions:**

For our boundary conditions it is relatively simple to impose the vacuum boundary conditions on the bottom and left faces. To do this we state that any in our tridiagonal matrix system. However, the reflecting boundary conditions on the top and right are a bit more complicated. Now we must impose the conditions that the change in flux through the right and top surface is 0. We define the right surface as and. Our discretized midpoint partial derivatives now become:

Ref. 6

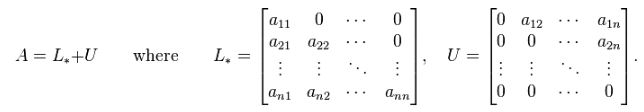
So now the term drops out and we can define the discretized equations at the right surface n as:

We do the same for the top reflecting condition and get a discretized equation at the top surface n of:

**Algorithm:**

From my discretized equations, I now have a coefficient matrix A composed of the various fluxes at each cell, and a solution matrix B composed of various source strengths at each cell. With the solution matrix fully defined, I was then able to use a Gauss-Seidel iterative solver to obtain a flux solution.

**Gauss-Seidel:**

The Gauss-Seidel method is an iterative method for solving linear systems of equations. The method involves decomposing the original A matrix into upper and lower triangular components.

Ref. 7

The original Ax = B matrix can then be defined by the following:

We now solve for some solution from an original guess solution of on the right hand side. For my code I simply used the source vector as our original guess solution. We keep performing this method for n iterations or until the iterative solution is within some tolerance of the actual direct solution.

In my code I set a max cap on iterations of 10000 and defined a tolerance of 1e-7.

**Code Use:**

For this program I have created a python graphical user interface that takes various inputs and solves the fixed source 2D diffusion equation through the method I have described above. Because I implemented my code in a GUI, it is fairly easy to run and one can examine many different situations fairly quickly. To run the program, simply import all the files in my repository into some folder. The GUI takes several inputs including: mesh size, boundary locations, source information, macroscopic cross sections (transport and absorption), and tolerance. I have also implemented a checkbox that can define the source as uniform over the entire space. Otherwise, one can directly input source strengths and their locations. Once all information is put into the GUI, several graphical representations of the flux can be made. This includes a surface plot, side plot, and heat map. The GUI also has an option to export the flux solution vector to a .txt file that you specify. Anytime one of these functions is implemented, the program will return a time of completion and the number of iterations required to solve the system.

**Test Problems and Results**

In my repository, I have also included three test output files that contain the flux solution vector of three different configurations. In the first ‘TestOut1.txt’ I created a system with a non-uniform source distribution that models neutron diffusion through water when two neutron point sources are placed in the medium. In the plots, you can clearly see the two flux peaks that quickly drop off as you move further from the source. In the ‘TestOut2.txt’ I created a situation with a uniform source. In the plot you can see the reflecting and vacuum boundary conditions with a steep drop off close to the vacuum boundaries. For the last “TestOut3.txt’ I placed three neutron point sources of equal strength evenly throughout the medium. The medium chosen has macroscopic cross-sections about twice as large as water. As a result, we can see a sharper drop off of the flux peaks. All of these tests were done with varying mesh sizes and it was clear that increasing the mesh size drastically increased the quality of the solution. However, this increased mesh size also drastically increased the number of iterations required and the time of completion for the system.

**Conclusion:**

With these mathematics and algorithms, I created an easy-to-use program that can quickly solve the 2D diffusion equation with the aforementioned boundary conditions. By implementing my code in a GUI, I was able to quickly describe various systems and obtain the flux distribution in these systems. Below, I have included a sample surface plot of ‘TestOut1.txt’

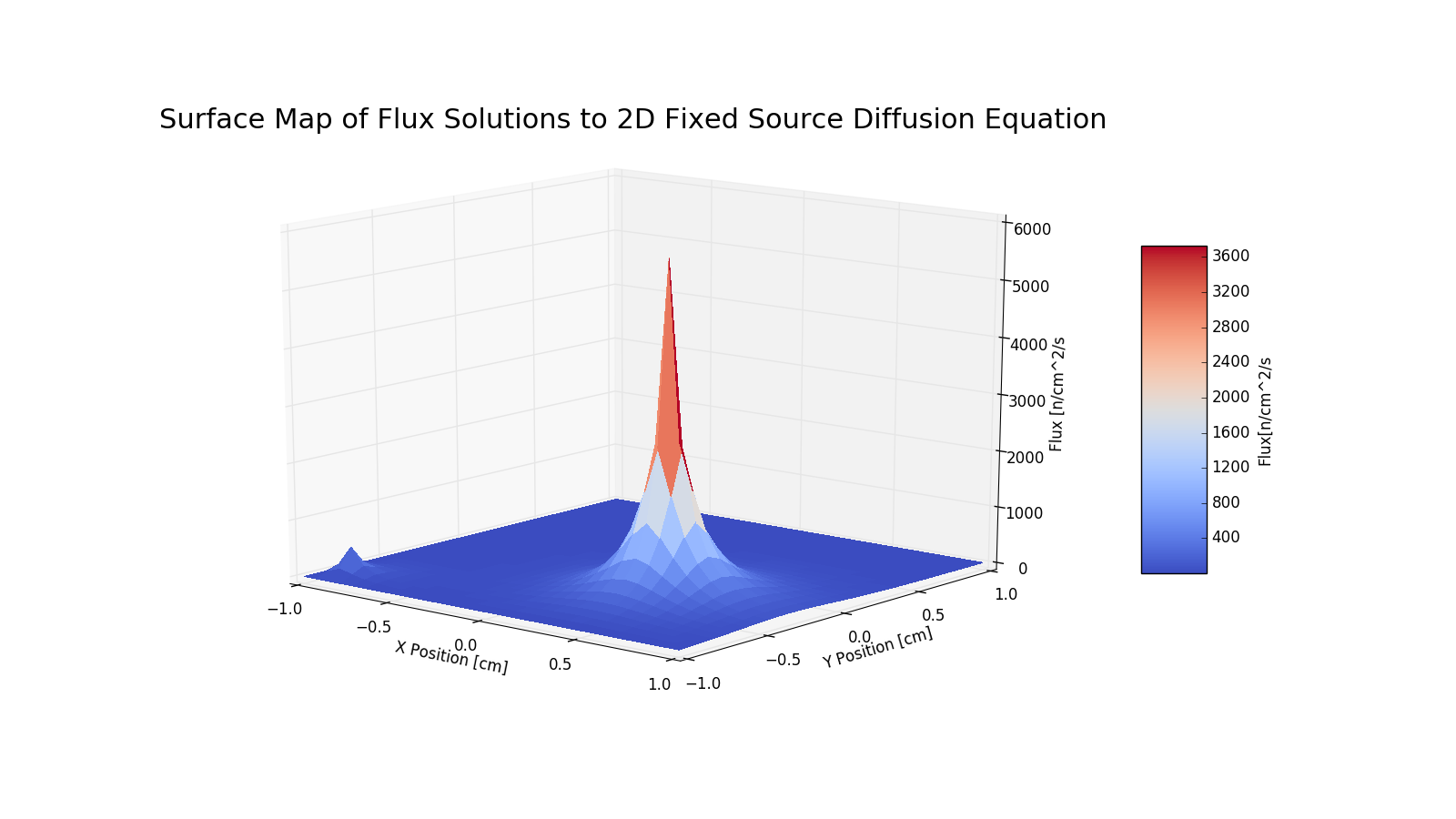


Figure 2. Surface Plot of 'TestOut1'

**References:**

1. *Ref 1, Ref 2, Ref 3, Ref 4, Ref 5: 2-D Finite Difference/Volume Methods Notes,* [*https://github.com/rachelslaybaugh/NE155/blob/gh-pages/24-2d-fd-fvm/24-2d-fd-fvm.pdf*](https://github.com/rachelslaybaugh/NE155/blob/gh-pages/24-2d-fd-fvm/24-2d-fd-fvm.pdf)*, Rachel Slaybaugh*
2. *Ref 6: NE155 HW 7 Solutions, bcources/NE155/Homework/homework7-soln.pdf,*

*Rachel Slaybaugh*

1. *Ref 7: Gauss–Seidel method, Wikipedia,* [*https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel\_method*](https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method)