A.D.O.

A 2D SPHERICAL ORDINATES SOLVER

"Much Ado About Neutrons"

ALEX HAGEN

May 2013 – version 0.1

Alex Hagen: *A.D.O.- A 2D Spherical Ordinates Solver,* "Much Ado About Neutrons", © May 2013

ABSTRACT

A discrete ordinate solver was generated for a simplified geometry. The discrete ordinate method is the state of the art method for neutronics calculations, which are extremely important in the nuclear industry. The method requires discretization in angle, space, and energy. Comparisons were made to scale to ensure correct implementation of the algorithm. The SCALE-NEWT 2D package provided an eigenvalue of 1.2 and a flux profile matching conventional wisdom. The ADO code written for this project did not converge towards an eigenvalue, and the flux mapping is disturbingly inaccurate. The code should be correct before any analysis can be done, but that is beyond the scope of time in this semester.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [2]

ACKNOWLEDGMENTS

Contributors to this code must be acknowledged. The principal contributors were Kevin Fischer and his help with Fortran syntax, Brian Archambault and his discrete ordinate guidance, and Tom Grimes and his knowledge of neutronics.

Also to be thanked are professors of this class (Dr. Satvat of Purdue Nuclear) and of my numerical methods class (Dr. Koslowski).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1 INTRODUCTION

1.1 MOTIVATION

With the decreasing price and increasing speed of computing resources in today's society, simulation of physical systems has become practical and desirable in a host of different applications. In fact, simulation of physical systems has become a staple of the design iteration process for almost any consumer, industrial, or scientific good. Simulation methods have been utilized in applications ranging from heat transfer, to fluid flows, to optics, and even to trajectory calculations. In general, any situation where measurement is difficult, safety is compromised, or experimentation cost is high are ideal for simulation. The three parameters listed previously are all met for nuclear reactor systems designs, as these are high cost, high risk systems that require intrusive and expensive electronics for the measurements of pertinant parameters.

COMSOL, Inc., a commercial platform has a yearly revenue of $13M, illustrating simulation's popularity

A nuclear fission reactor system generates electricity off of one main premise, that fissionable fuel can generate a very favorable neutron economy. This fuel will reproduce neutrons at a rate set by the setup of the reactor. When the reactor is critical, or when the neutron economy is exactly steady state, an ideal point for power generation is reached. Each fission occuring in the reactor generates heat, which then can be used in a thermodynamic cycle to produce power. As long as the reactor is running at critial or very close there-to, that power generated will stay steady for long periods of time (for current commercial systems, up to 18 months before refueling). In order to optimize this neutron economy, reactor physics has long been utilized.

Reactor physics is the science of solving the Boltzmann Transport equation shown below through energies and spatially to determine the overall criticality in a reactor system, and also to identify the distribution of flux throughout the reactorOtt and Bezella [3].

The Boltzmann Transport equation and its use in neutronics will be discussed in 1.2

$$\begin{aligned}\overrightarrow{\Omega} \cdot \nabla \psi(\overrightarrow{r}, E, \overrightarrow{\Omega}) + \Sigma_t(\overrightarrow{r}, E)\psi(\overrightarrow{r}, E, \overrightarrow{\Omega}) = \\ \int_{\overrightarrow{\Omega}} \int_{E'} \Sigma_s(\overrightarrow{r}, E' \to E, \overrightarrow{\Omega}' \to \overrightarrow{\Omega})\psi(\overrightarrow{r}, E, \overrightarrow{\Omega}) dE d\overrightarrow{\Omega} \\ + \chi(E) \int_{E'} \nu\Sigma_f(\overrightarrow{r}, E')\psi(\overrightarrow{r}, E', \overrightarrow{\Omega}) dE' + S(\overrightarrow{r}, E, \overrightarrow{\Omega}) \quad (1)\end{aligned}$$

It is highly mathematical, and because of the nature of the partial differential equation extremely complex. Thus, it is near impossible to analytically solve the neutronics in even the simplest systems. Several different types of codes have been developed to numerically approximate these equations. The first, and more simple codes, probabilistic codes, are much more computationally expensive, and also ill suited for criticality estimations. This generates the need for deterministic codes, of which those using the method of Discrete Ordinates is currently state of the art. ADO is a two-dimensional Discrete Ordinates solver for generalized geometry and composition, although only tested on one reactor system, which fills the need described in the preceding paragraph.

1.2 BACKGROUND

In order to obtain useful information from the Boltzmann Transport equation, the equation and it's dependences must be understood. To obtain this equation from first principals, the mechanisms of interactions of neutrons must be considered. For practical purposes, neutrons will be absorbed, will fission, and will scatter into and out of both energy groups and space. Other reactions between neutrons and the matter is less likely and therefore not taken into account. Accounting for these reactions, and creating the form of a generalized source sink problem, we obtain the mathematical expression below.

$$\begin{bmatrix} change\,in \\ number\,of \\ neutrons \\ over\,time \end{bmatrix} = \begin{bmatrix} amount\,of \\ neutrons \\ entering\,cv \end{bmatrix} - \begin{bmatrix} amount\,of \\ neutrons \\ leaving\,cv \end{bmatrix} \tag{2}$$

Converting this to typical mathematical notation, and understanding that each of the operators in the following equation are matrix and also differential operators, we obtain the form (for steady state) below.

$$M\Phi = \lambda F\Phi + S \tag{3}$$

where M is the migration operator, λ is the eigenvalue (or criticality) of the reactor, F is the fission operator, and S is an external source Ott and Bezella [3].

1.2.1 *Separability and Analytical Solutions*

Unfortunately, as stated, each of these operators is differential and also matrix, creating an extremely difficult equation to solve. Also very difficult is the dependence in both space and energy. Luckily, the equations can be analytically separated in energy and space, giving the Boltzmann Transport equation (Equation 1), which can be separated into the Boltzmann Space equation (Equation 4), and the Slowing Down equation (Equation 5) Ott and Bezella [3].

$$-\frac{\nabla^2\phi(\mathbf{r})}{\phi(\mathbf{r})} = B^2 \therefore \nabla^2\phi(\mathbf{r}) + B^2\phi(\mathbf{r}) \tag{4}$$

$$\begin{aligned}\left[D(E)B^2 + \Sigma_t(E)\right]\varphi(E) - \int \Sigma_s(E' \to E)\varphi(E')dE' \\ = \lambda\chi(E)\int \nu\Sigma_f(E')\varphi(E')dE'\end{aligned} \tag{5}$$

1.2.2 *Discrete Ordinates*

In order to solve these extremely difficult analytical equations numerically, a discretization method must be used. The most common and state of the art method is called Discrete Ordinate method, and it takes advantage of the Boltzmann Transport Equation to discretize a generalized region in several variables. While the discretization in space and energy is obvious, the most important part of Discrete Ordinates is the discretization in angle. By calculating angular flux at many different angles at any given point, higher orders of scatting can be taken into account, and more accurate results generated. The angular discretization is done using the concept of level sets, where each quadrant of the unit sphere is split up into a defined amount of angles (called the S_n) order. An example diagram of this is shown below. By using weighting given by Legendre Polynomials, the scalar flux can be generated from each set of angular fluxes, and in these cases the source term can be updated. By doing this, convergence in both flux and eigenvalue is achieved.

1.3 METHODOLOGY

The programming of the discrete ordinates method is exceedingly complex, and even more complex to ensure efficiency. The use of a matrix friendly programming language can be helpful, as can parallelized computing, but in all, the iterative method is highly computationally expensive. The methods shown are not optimized in any sense, and state of the art packages will use many accelerators. These accelerating methods are beyond the scope of this class and writeup.

1.3.1 *Programming Language - Fortran*

As is the case with many engineering (and especially nuclear engineering), Fortran was chosen as the language of choice. As well as having robust array handling within the code, the code is also extremely explicit, which for numerical methods, is good as a way to fine tune precision. Several numerical methods and import packages that were previously written were used to ease the authoring of the program. The input deck format was developed on the fly, and all of the output is text based (or SVG).

1.3.2 *Computational Resources*

A server using Four Intel Xeon E5504 CPUs each running at 2.00 GHz were used. The ram installed was 4.00 GB, and the operating system was a bleeding edge linux distribution of ubuntu. F95, the gnu fortran compiler was used to compile and run the code.

For information on how to compile and run the code, please see Appendix B.1 and B.3

1.3.3 *Overall Methodology*

By following the algorithm 1.1 provided, the overall methodology may be followed. This methodology again discretizes in not only space, but angles, and energy. The general structure is simple, with many loops of discretization, but the creation of level sets, the generalization of import parameters, and the development of the quickly converging code are extremely difficult and caused many problems.

1.3.4 *Importing*

Using a very strict import sequence, each of these files were read in from an input deck supplied while running the program. This could easily be converted to CCCC input. The input deck must provide cross sections as well as the geometry and submeshing parameters desired. The cross sections could be outsourced to a cross section generation subroutine if given time, and likely would be in a commercial code. The importing routine also provides a visualization.

For an example of the input deck, please see Appendix B.2

1.3.5 *Resolving Geometry and Meshing*

The visualization in the importing of the geometry is shown in figure 1. The geometry has been colored by the composition provided in the input deck. For now, the importing is generalized to only square imputs, but it could be expanded for triangular elements easily. Circular elements then could be estimated using a conglomeration of triangles.

In order to mesh this geometry, first splits must be made. In this case, a split was made four times across each cell and the intersections of these points saved to a list. Then, the midpoint (in both x and y coordinates) of these slices was taken, and this was also saved to the list. Then, in order to traverse in the "snake" pattern required for solving (see Appendix A) the finite difference method, a recursive pointer follow method was used, as is shown in algorithm 1.2. This then only requires that the main program follow each pointer from

Algorithm 1.1 Discrete Ordinates Overall Code Pseudocode

ε is the current error in a given variable, and cc is the convergence criteria for that variable

```
call importparameters(in:inputdeck,out:$N_{rows}$,$N_{columns}$,
                                      $S_n$,$\sigma(\overrightarrow{r}, E, rxn)$)
call generateangles(in:$S_n$,out:$\overrightarrow{\Omega}$)
while $\varepsilon_k > cc_k || \varepsilon_\phi > cc_\phi$ do
    for $E = 1 \rightarrow 2$ do
        for $N_{\overrightarrow{\Omega}} = 1 \rightarrow S_n$ do
            $\theta \leftarrow \overrightarrow{\Omega}(N_{\overrightarrow{\Omega}}, 1)$
            $\mu \leftarrow \cos(\theta)$
            $\eta \leftarrow \sin(\theta)$
            for $i = 1 \rightarrow N_{rows}$ do
                for $j = 1 \rightarrow 2 \cdot N_{columns}$ do
                    call getpointparams(in:$i$,$j$,out:$\Delta x$,$\Delta y$ ,$S_{i,j}$,
                                      $\sigma_{tr,i,j}$,$\psi_{tl}$,$\psi_{br}$,$\psi_{tr}$,$\psi_{bl}$)
                    $\psi_{i,j} = \frac{S_{i,j} + \frac{2|\mu m|}{\Delta x}(\psi_{tl} - \psi_{br}) + \frac{2|\eta m|}{\Delta y}(\psi_{tr} - \psi_{bl})}{\sigma_{tr,i,j} + \frac{2|\mu m|}{\Delta x} + \frac{2|\eta m|}{\Delta y}}$
                    $i \leftarrow nexti, j \leftarrow nextj$
                end for
            end for
            $\phi(\overrightarrow{r}, E) \leftarrow \frac{1}{4} \sum_{n=1}^{N(N-1)/2} w_n \psi_{n,i,j}$
            $k^m \leftarrow k^{m-1} \cdot \frac{\sum_{i=1}^{N_{columns}} \sum_{j=1}^{N_{rows}} \phi_{ij}^m}{\sum_{i=1}^{N_{columns}} \sum_{j=1}^{N_{rows}} \phi_{ij}^{m-1}}$
        end for
        call outputresults(in:$k$,$\phi(\overrightarrow{r}, E)$,geo ,out:outputfile)
    end for
end while
```

A visualization of the meshing of every run is generated and output in SVG format at `./drawing.svg`

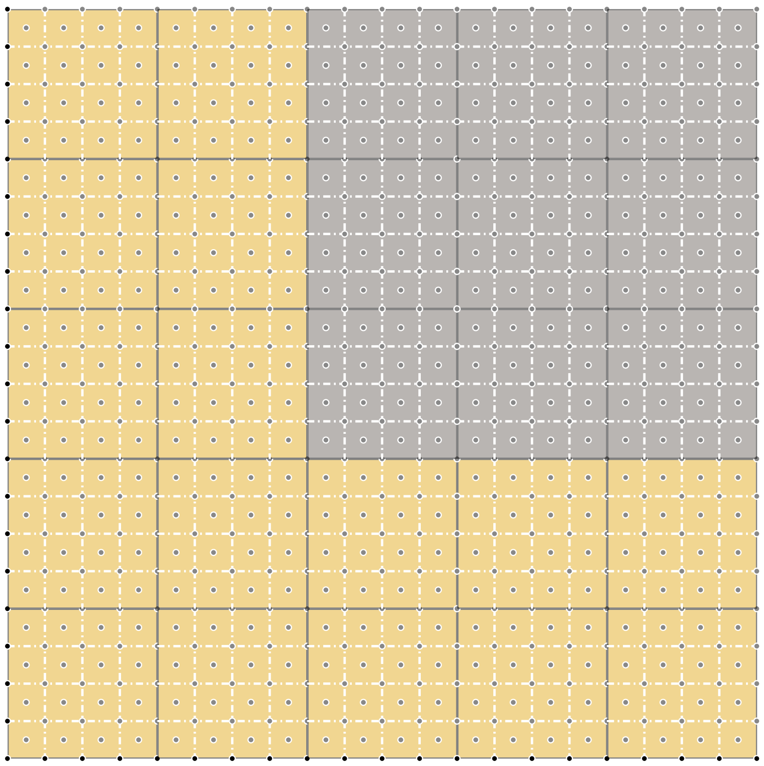

Figure 1: Meshing of Given Geometry (4 × 4 Submesh)

start to end and it will have traversed the geometry in the correct manner. By doing this, it will allow the visual and readability of the main program to be linear. The traversing pattern is further illustrated in figure 2 for the geometry involved.

Because of discrete ordinates method's reliance on angles, there are four cases in which the geometry must be traversed. These cover the space of which direction spatial it is possible to be going (positive x, negative x, positive y and negative y). The equation which governs these cases is shown below.

$$N_{\vec{\Omega}} = \begin{cases} 1 & left \rightarrow right,\ bottom \rightarrow top & \begin{matrix} \mu_n > 0 \\ \eta_n > 0 \end{matrix} \\ 2 & right \rightarrow left,\ bottom \rightarrow top & \begin{matrix} \mu_n < 0 \\ \eta_n > 0 \end{matrix} \\ 3 & left \rightarrow right,\ top \rightarrow bottom & \begin{matrix} \mu_n > 0 \\ \eta_n < 0 \end{matrix} \\ 4 & right \rightarrow left,\ top \rightarrow bottom & \begin{matrix} \mu_n < 0 \\ \eta_n < 0 \end{matrix} \end{cases} \tag{6}$$

Algorithm 1.2 Mesh Traversing Algorithm

*i is the case (given by equation **??**)*

```
$dx \leftarrow \frac{lim_x}{2 \cdot subm_x}$
$dy \leftarrow \frac{lim_y}{2 \cdot subm_y}$
$startpoints \leftarrow \begin{bmatrix} 0.00 + dx & 0.00 + dy \\ lim_x - dx & 0.00 + dy \\ 0.00 + dx & lim_y - dy \\ lim_x - dx & lim_y - dy \end{bmatrix}$
for $i = 1 \rightarrow 4$ do
    $x \leftarrow startpoint(i, 1)$
    $y \leftarrow startpoint(i, 2)$
    while $y <= y_{lim}$ && $y >= 0.00$ do
        while $x <= x_{lim}$ && $x >= 0.00$ do
            call findindex($x$,$y$)
            $meshpoints(index, 2 + i) \leftarrow index$
            $x \leftarrow x + dir_x \cdot dx$
        end while
        $dir_x \leftarrow (-1) \cdot dir_x$
        $y \leftarrow y + dir \cdot dy$
        $x \leftarrow x_{lim}$
        while $x <= x_{lim}$ && $x >= 0.00$ do
            call findindex($x$,$y$)
            $meshpoints(index, 2 + i) \leftarrow index$
            $x \leftarrow x + dir_x \cdot dx$
        end while
        $y \leftarrow y + dir \cdot dy$
        $dir_x \leftarrow (-1) \cdot dir_x$
        $x \leftarrow startpoint(i, 1)$
    end while
end for
```

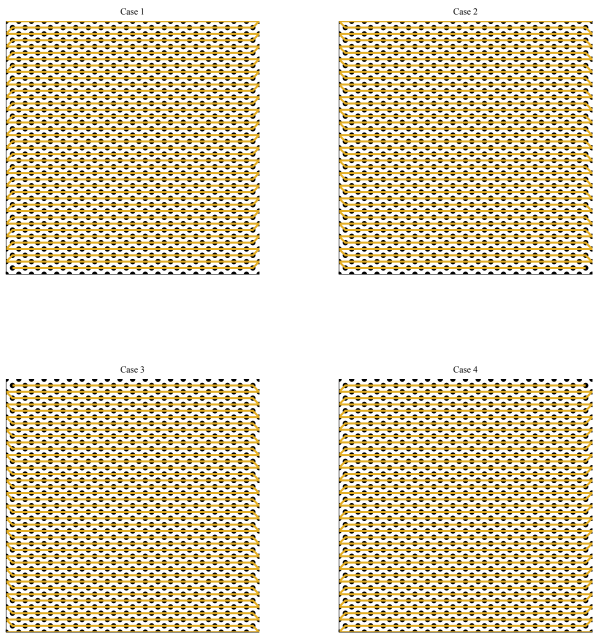

Figure 2: Traversing of Geometry

1.3.6 *Finite Difference Method*

The traversing described above and the equations involved is shown below. Of special interest is the setup of the matrix of meshing in table 1, which includes pointers to the next step in the traverse, but also includes pointer to the top-left, top-right, bottom-right, and bottom-left node too. This makes invoking the actual calculation section of the code extremely easy, and also allows for the intialization of boundary condtions which are impoosible to be broken.

*i is the index of the mesh point for given argument, ns, $N_{\vec{\Omega}}$ indicates next step and case (given by equation **??**), v, h indicates position (e.g. t, l indicates "top-left")*

Table 1: Meshing Matrix Setup

x	y	$i_{ns,1}$	$i_{ns,2}$	$i_{ns,3}$	$i_{ns,4}$	$i_{t,l}$	$i_{t,r}$	$i_{b,r}$	$i_{b,l}$
0.000	0.000	6	...	...	...	2	3	4	5
				...					
1.260	0.000	11	...	...	...	7	8	9	10
				...					

1.3.7 *Calculation of Scalar Flux and Criticality*

The following equations are the definitions for scalar flux and criticality. Because of Fortran's built in sum function, these became extremely easy to code and were a small milestone in developing the code.

for equation 7, k is the criticality, m is the iteration number, and the summations are over space (i, j)

$$k^m = k^{m-1} \cdot \frac{\sum_{i=1}^{N_{columns}} \sum_{j=1}^{N_{rows}} \phi_{ij}^{m}}{\sum_{i=1}^{N_{columns}} \sum_{j=1}^{N_{rows}} \phi_{ij}^{m-1}} \tag{7}$$

for equation 8, n is over the discretized angular domain, with weight w

$$\phi(\overrightarrow{r}, E) = \frac{1}{4} \sum_{n=1}^{N(N-1)/2} w_n \psi_{n,i,j} \tag{8}$$

1.3.8 *Output*

Output was done in E15.9 format to provide sufficient precision. A listing of outers as well as the criticality calculated at each step was written to an output file. Also, the angular flux at a certain point was provided, and the scalar flux over the whole domain. An svg plot could be provided using the tools provided in `./svgtools.f90`, but this has not been implemented in the code yet.

2 RESULTS

2.1 SCALE - NEWT RESULTS

SCALE is a discrete ordinate method solver package, which includes the 2D package NEWT. To utilize and compare results to the ADO code, a specified geometry of a GE 8x8 core was modeled and run in SCALE-NEWT. The parameters on this were P_n order of 1 to match that of the ADO code, and S_n order of 8, which is slightly higher than that of the ADO code. The geometry can be seen in the diagram at figure 3, where circular fuel rods are surrounded by water moderator. The compositions of these rods and the water was supplied by the instructor.

Running of SCALE-NEWT took nearly 4 hours on one 2.00\, GHz core. It performed 53 outers, and converged near the value of 1.2 that was desired. While the full results for eigenvalue convergence have not been shown here (selected values shown in table [tab:Scale-Outer-Iteration]), they show that the convergence is logarithmic, as would be expected for a discrete ordinate code. The excessive computational time required to run this core could be eased by using a lower S_{n} order, or by creating reflected boundaries so that only $\frac{1}{4}$ core was solved for. Finally, the submeshing was fine, and could be made coarser in order to speed up the results.

The flux profiles output from the SCALE-NEWT code are given in post script form and are shown in figure 4. The colors represent the magnitude of the flux, with blue being the lowest and moving up through green and into red, which shows the maximum. It can be seen that the most fast neutrons (group 1) are in the fuel regions,

Table 2: Scale Outer Iteration Convergence

Outer Iteration #	Eigenvalue
m	k
1	1.000
2	0.7417
3	0.7539
...	
51	1.1761
52	1.1761
53	1.1761

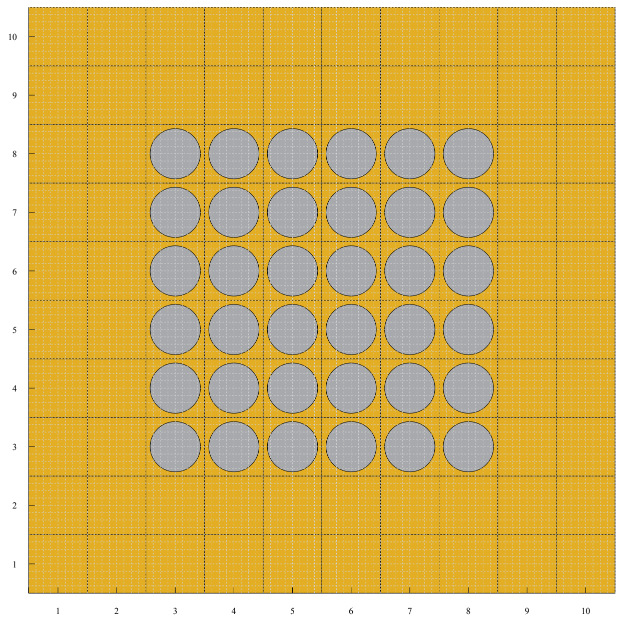

Figure 3: Scale Geometry Setup

whereas in the moderator, group 2 neutrons are more likely to reside. This is logical, as the high absorption cross section of the fuel will remove the thermal neutrons from these regions, and because the fission source provides *MeV* neutrons in the fuel regions.

Angular flux profiles were not provided from SCALE-NEWT. It would be interesting to rerun the analysis to generate anuglar flux profiles. Then, the angular flux in different angles around points could be analyzed. Some of these points that would be interesting would be at each of the corners of the fuel region, and to move just

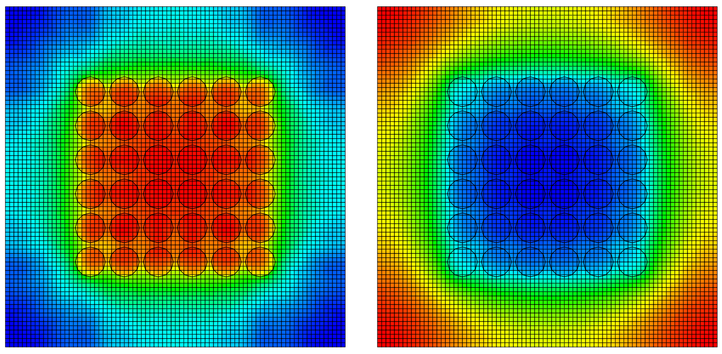

Figure 4: SCALE Flux Profile Results

Table 3: Outer Iterations of ADO

#	k	#	k
1	0.281281527	11	1.95252247
2	0.792616515	12	0.512328343
3	1.77733406	13	1.94180662
4	0.47360058	14	0.969952259
5	1.89769684	15	1.96948097
6	0.960008281	16	0.152952564
7	1.9421711	17	1.83289319
8	0.271525595	18	0.852822018
9	1.90604859	19	2.01192901
10	0.963321108	20	0.495220099

The oscilation shown in these 20 trials propagated itself over several hundred more, and thus it is believed that the code is outside of its limits of convergence

outside that to see how anisotropic scattering would affect this profile.

In all, the SCALE-NEWT results were computationally expensive, but provide detailed and accurate results. The ADO code will look to emulate this.

2.2 ADO RESULTS

The results from ADO are mixed. The geometry and meshing descriptive picture shown above is provided, which is very similar to the SCALE package outputs (figure 1). Unfortunately, the convergence of the code showed that the solution provided was unstable. Following is a table of the first 20 outers of the ADO code, and it can be seen that the outers are oscilating around the true value (1.2). This is likely a problem in the source term.

In order to further analyze the ADO results, the scalar flux was plotted using a MATLAB interpolation program. The flux plot below shows even more of the problem that is occurring in the code. It is apparent that the geometry is not being meshed properly, or that the upper corner (the fuel) has no source term. In fact, it looks as though one single beam of flux is coming in, when that is not the case in reality at all. This may be because of the way that the meshing was done, in that the program may not be following the mesh points well. I am disappointed with my results.

The angular flux looks more promising. The angular flux was plotted at point 5 (bottom left corner), and it showed that the most flux occurred upward, which makes sense. It also showed that this flux occurred at higher levels in the thermal region, and since we are deep into moderator region here, this also makes sense. It would be inter-

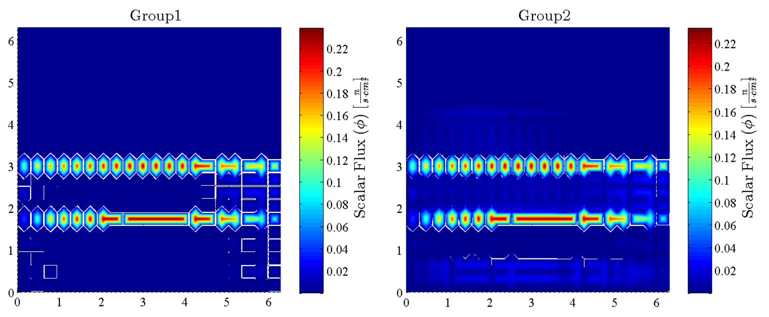

Figure 5: Interpolated Flux Profile from ADO Results

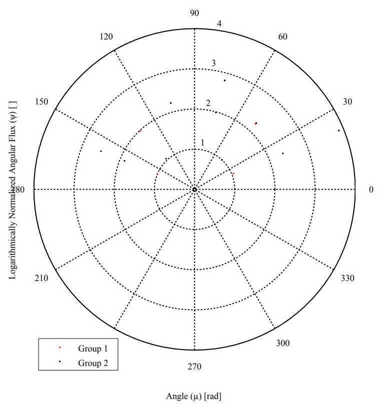

Figure 6: Angular Flux Profile at Mesh Point 5

esting to correct the code and be able to provide better insight into what is going on with these results.

Part I

APPENDIX

A NUMERICAL METHODS

As described in Brian Bradie's *A Friendly Introduction to Numerical Analysis* Bradie [1], and given the setup in Figure 7, where the boundary condition at $j = 0$, $i = 0$ is vacuum, reflected boundary at $j = 1$ and $i = 2$, and running at 45^o.

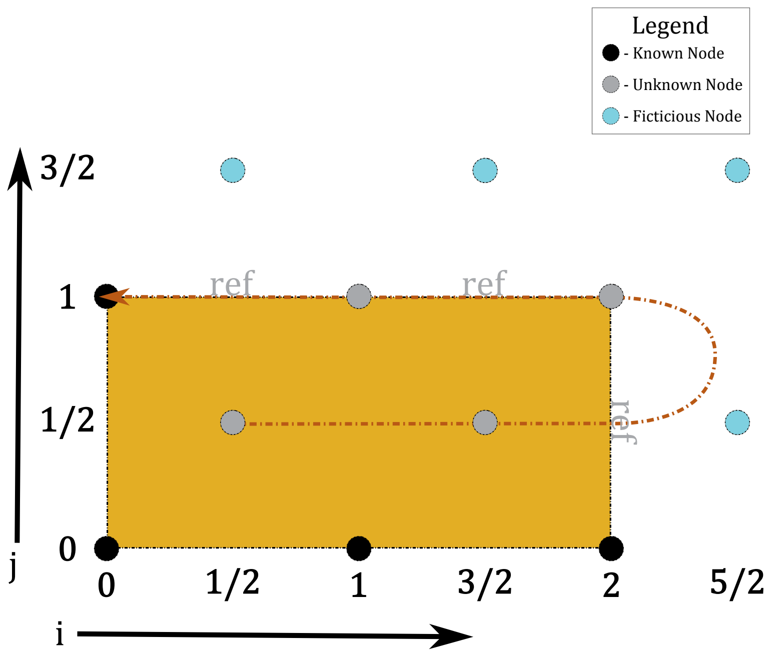

Figure 7: Illustration of Finite Difference Method

The following will solve the problem
For point $(\frac{1}{2}, \frac{1}{2})$

$$\psi_{\frac{1}{2},\frac{1}{2}} = \frac{S_{\frac{1}{2},\frac{1}{2}} + \frac{2|\mu_m|}{\Delta x}\left(\psi_{1,1} - \psi_{0,0}\right) + \frac{2|\eta_m|}{\Delta y}\left(\psi_{1,0} - \psi_{0,1}\right)}{\sigma_{tr,\frac{1}{2},\frac{1}{2}} + \frac{2|\mu_m|}{\Delta x} + \frac{2|\eta_m|}{\Delta y}} \tag{9}$$

where $S_{\frac{1}{2},\frac{1}{2}} = \nu\sigma_{f,\frac{1}{2},\frac{1}{2}} \cdot \phi_{old,\frac{1}{2},\frac{1}{2}}$, and $\sigma_{tr,\frac{1}{2},\frac{1}{2}}$, μ_m, Δx, Δy, and η_m are known quantities. in this case, $\psi_{0,0}$, $\psi_{0,1}$, and $\psi_{1,0}$ are known (vacuum boundary). Thus, the above equation has two unknowns: $\psi_{\frac{1}{2},\frac{1}{2}}$, and $\psi_{1,1}$.

Now we move on to point $(\frac{3}{2}, \frac{1}{2})$, and we have the equation

$$\psi_{\frac{3}{2},\frac{1}{2}} = \frac{S_{\frac{3}{2},\frac{1}{2}} + \frac{2|\mu_m|}{\Delta x}\left(\psi_{2,1} - \psi_{1,0}\right) + \frac{2|\eta_m|}{\Delta y}\left(\psi_{1,1} - \psi_{0,2}\right)}{\sigma_{tr,\frac{3}{2},\frac{1}{2}} + \frac{2|\mu_m|}{\Delta x} + \frac{2|\eta_m|}{\Delta y}} \tag{10}$$

Now, we have three unknowns: $\psi_{\frac{3}{2},\frac{1}{2}}$, $\psi_{2,1}$, and $\psi_{1,1}$, but $\psi_{1,1}$ is also a function of $\psi_{\frac{1}{2},\frac{1}{2}}$, so in all, we have two equations with three unknowns.

Then we move on to the point $(2,1)$ and have the equation

$$\psi_{2,1} = \frac{S_{2,1} + \frac{2|\mu_m|}{\Delta x}\left(\psi_{\frac{5}{2},\frac{3}{2}} - \psi_{\frac{3}{2},\frac{1}{2}}\right) + \frac{2|\eta_m|}{\Delta y}\left(\psi_{\frac{3}{2},\frac{3}{2}} - \psi_{\frac{5}{2},\frac{1}{2}}\right)}{\sigma_{tr,2,1} + \frac{2|\mu_m|}{\Delta x} + \frac{2|\eta_m|}{\Delta y}} \tag{11}$$

where we have five unknowns: $\psi_{2,1}$,$\psi_{\frac{5}{2},\frac{3}{2}}$, $\psi_{\frac{3}{2},\frac{1}{2}}$, $\psi_{\frac{3}{2},\frac{3}{2}}$, and $\psi_{\frac{5}{2},\frac{1}{2}}$. Recall that we have the point $\psi_{\frac{3}{2},\frac{3}{2}}$ in expressions of other unknowns, leaving us with four unknowns for this equation.

Using the reflective boundary, we have $\psi_{\frac{5}{2},\frac{3}{2}} = \psi_{\frac{3}{2},\frac{1}{2}}$, $\psi_{\frac{5}{2},\frac{1}{2}} = \psi_{\frac{3}{2},\frac{1}{2}}$, leaving us with three equations and five unknowns.

Thus we move on to the point $(1,1)$, and have the equation

$$\psi_{1,1} = \frac{S_{1,1} + \frac{2|\mu_m|}{\Delta x}\left(\psi_{\frac{3}{2},\frac{3}{2}} - \psi_{\frac{1}{2},\frac{1}{2}}\right) + \frac{2|\eta_m|}{\Delta y}\left(\psi_{\frac{1}{2},\frac{3}{2}} - \psi_{\frac{3}{2},\frac{1}{2}}\right)}{\sigma_{tr,1,1} + \frac{2|\mu_m|}{\Delta x} + \frac{2|\eta_m|}{\Delta y}} \tag{12}$$

where we have five unknowns: $\psi_{1,1}$, $\psi_{\frac{3}{2},\frac{3}{2}}$, $\psi_{\frac{1}{2},\frac{1}{2}}$, $\psi_{\frac{1}{2},\frac{3}{2}}$, and $\psi_{\frac{3}{2},\frac{1}{2}}$. Recall that we have $\psi_{1,1}$, $\psi_{\frac{3}{2},\frac{3}{2}}$, $\psi_{\frac{3}{2},\frac{1}{2}}$ and $\psi_{\frac{1}{2},\frac{1}{2}}$as an expression of other unknowns, so we have four equations with six unknowns.

Additionally, though, we have the reflective boundary condition though, so $\psi_{\frac{1}{2},\frac{3}{2}} = \psi_{\frac{1}{2},\frac{1}{2}}$.

This leaves us with four equations and five unknowns.

In order to remove the final unknown, we move backwards along the boundary to the vacuum boundary, giving us the following equation

$$\psi_{0,1} = \frac{S_{0,1} + \frac{2|\mu_m|}{\Delta x}\left(\psi_{\frac{1}{2},\frac{3}{2}} - \psi_{-\frac{1}{2},\frac{1}{2}}\right) + \frac{2|\eta_m|}{\Delta y}\left(\psi_{-\frac{1}{2},\frac{3}{2}} - \psi_{\frac{1}{2},\frac{1}{2}}\right)}{\sigma_{tr,0,1} + \frac{2|\mu_m|}{\Delta x} + \frac{2|\eta_m|}{\Delta y}} \tag{13}$$

where we know $\psi_{0,1}$, $\psi_{-\frac{1}{2},\frac{1}{2}}$, and $\psi_{-\frac{1}{2},\frac{3}{2}}$ because of the boundary vacuum condition. The values $\psi_{\frac{1}{2},\frac{3}{2}}$ and $\psi_{\frac{1}{2},\frac{1}{2}}$ we have in expressions of other variables, so overall we now have five equations and five unknowns.

B SAMPLE USAGE

B.1 SAMPLE USAGE FROM TERMINAL

To use the ado package, with a comiled (for instructions compiling see B.3) binary file called ADO, the following command will provide results.

```
./ado inputdeck outputdeck
```

where `inputdeck` is the path to the preformatted input deck (sample given in Appendix B.2), and `outputdeck` is a path to an empty or write permissioned file that the output will be written to.

B.2 SAMPLE INPUT DECK

File Input Deck:

```
!SUBMESHING
!--|--#x--|--#y--|
SM       4      4
!GEOMETRY
!Name   | # |  Comp  |         Corner          |          Size          |
!Name---|-#-|--Comp--|[         x1,         y1]|[          x,          y]|
!If #>1
!Name---|-#-|[  x1,  y1;  x2,  y2;...]
G  mod    16        1 [0.00000E+00,0.00000E+00] [1.26000E+00,1.26000E+00]
GA mod    16 [0.00000E+00,0.00000E+00;0.00000E+00,1.26000E+00;...
0.00000E+00,2.52000E+00;0.00000E+00,3.78000E+00;0.00000E+00,5.04000E+00;...
1.26000E+00,0.00000E+00;1.26000E+00,1.26000E+00;1.26000E+00,2.52000E+00;...
1.26000E+00,3.78000E+00;1.26000E+00,5.04000E+00;2.52000E+00,0.00000E+00;...
2.52000E+00,1.26000E+00;3.78000E+00,0.00000E+00;3.78000E+00,1.26000E+00;...
5.04000E+00,0.00000E+00;5.04000E+00,1.26000E+00;]
G  fuel    9        2 [2.52000E+00,2.52000E+00] [1.26000E+00,1.26000E+00]
GA fuel    9 [2.52000E+00,2.52000E+00;2.52000E+00,3.78000E+00;...
2.52000E+00,5.04000E+00;3.78000E+00,2.52000E+00;3.78000E+00,3.78000E+00;...
3.78000E+00,5.04000E+00;5.04000E+00,2.52000E+00;5.04000E+00,3.78000E+00;...
5.04000E+00,5.04000E+00;]
!CROSS SECTIONS
!-Comp-|---S_a_1---|---S_a_2---|---S_tr_1--|---S_tr_2--|---S_f_1---|---S_f_2---|
!-Comp-|---nS_f_1--|---nS_f_2--|---S_s_11--|---S_s_12--|---S_s_21--|---S_s_22--|
XS1   1|3.46183E-04|1.32850E-02|1.89973E-01|1.52944E+00|0.00000E+00|0.00000E+00|
XS2   1|0.00000E+00|0.00000E+00|6.26957E-01|3.25824E-02|2.00906E-04|2.32415E+00|
!-------------------------------------------------------------------------------
XS1   2|1.85092E-02|2.17512E-01|2.53081E-01|1.10833E+00|5.20891E-03|1.58724E-01|
XS2   2|1.32055E-02|3.86762E-01|4.88184E-01|1.42033E-02|1.64414E-04|1.29876E+00|
```

B.3 SAMPLE COMPILATION

In the case of recompiling from source, the following command can be invoked within a directory containing all of the source files.

```
f95 -o ado variables.f90 functions.f90 svgtools.f90 ...
     subroutines.f90 ado.f90
```

where `f95` is the command for a Fortran compiler (at least Fortran 90 compliant) that will be used.

BIBLIOGRAPHY

[1] Brian Bradie. *A Friendly Introduction to Numerical Analysis*. Pearson Prentice Hall, Upper Saddle River, New Jersey, 2006. ISBN 0-13-013054-0. (Cited on page 17.)

[2] Donald E Knuth. {C}omputer {P}rogramming as an {A}rt. *Communications of the ACM*, 17(12):667–673, December 1974. (Cited on page iv.)

[3] K Ott and W Bezella. *Introductory Nuclear Reactor Statics*. American Nuclear Society, La Grange Park, Illinois, 1989. (Cited on pages 1 and 2.)