

Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-2001

Modeling Pressurized Water Reactor Kinetics

William H. Harman

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Nuclear Engineering Commons](#)

Recommended Citation

Harman, William H., "Modeling Pressurized Water Reactor Kinetics" (2001). *Theses and Dissertations*. 4622.

<https://scholar.afit.edu/etd/4622>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.



Modeling Pressurized Water Reactor Kinetics

Thesis

William H. Harman, Major, USA

AFITGNEVENP01M-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

20010730 032

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Modeling Pressurized Water Reactor Kinetics

THESIS

Presented to the Faculty

Department of Engineering Physics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Nuclear Engineering

William H. Harman, B.S., M.S., P.E.

Major, USA

March 2001

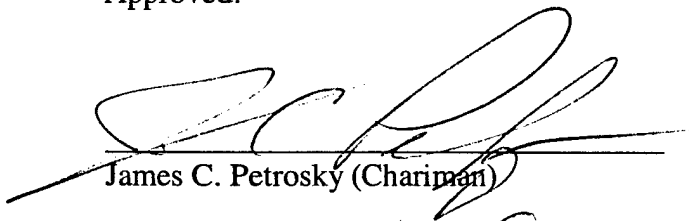
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Modeling Pressurized Water Reactor Kinetics

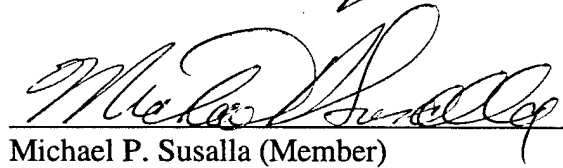
William H. Harman, B.S., M.S., P.E.

Major, USA

Approved:


James C. Petrosky (Chairman)

20 FEB 01
date


Michael P. Susalla (Member)

20 FEB 01
date


Vincent J. Jodoin (Member)

20 FEB 01
date

Acknowledgements

I would like to thank my advisor, LTC James Petrosky, for his guidance, counsel, and willingness to allow me to approach and solve the problem in my own way. A special thanks goes to Dr. William Baker for his hours of tutoring and assistance in developing the mathematical background to solve the numerical equations, to Capt. Mark Suriano for his assistance in computational problem solving, and to Commander Michael Susalla for his professional teaching during my first four quarters at AFIT.

Finally, I would like to thank my wife for giving me her full support and encouragement. I could not have completed this thesis without her.

Table of Contents

	Page
Acknowledgements	iv
Table of Contents	v
List of Figures	vii
List of Tables	ix
Abstract	x
I. Introduction	1
Background	1
Problem Statement	1
Approach	2
II. Theoretical Development	6
Diffusion Theory	6
Two Dimensional, One Energy Group Model	12
<u>Criticality Solution Technique</u>	12
<u>Numerical Development</u>	18
Three Dimensional, One Energy Group Model	20
<u>Criticality Solution Technique</u>	20
<u>Numerical Development</u>	21
<u>Matrix Solution Methods</u>	25
Three Dimensional, Two Energy Group Model	27
<u>Criticality Solution Technique</u>	27
<u>Numerical Development</u>	28
III. Program Development and Validation	30
Program Development	30
Operating the Code	32
Program Validation	41
<u>Two Dimensional, One Energy Group Model</u>	41
<u>Three Dimensional, One and Two Energy Group Models</u>	44
IV. Conclusions and Recommendations	53
Conclusions	53

Recommendations	54
Appendix A. Derivation of Three Dimensional Source Integration	57
Appendix B. Derivation of Right Circular Cylinder Reactor Core Solution	59
Appendix C. Two Dimension, One Energy Group Visual BASIC Code	62
Appendix D. Three Dimension, One Energy Group Visual Basic Code	70
Appendix E. Three Dimension, Two Energy Groups Visual Basic Code	83
Appendix F. Sample Output Charts and Data	98
Appendix G. Relative Error Plots of Test Cases	102
Appendix H. Linking Visual BASIC and Excel	119
Bibliography	123
Vita	125

List of Figures

Figure	Page
Figure 1 Extrapolated distance at outer boundary.....	9
Figure 2 Energy Groups	10
Figure 3 Energy Spectra for Two Energy Group Model.....	11
Figure 4 Criticality Search Technique, 2D Model	15
Figure 5 Boundary Conditions	19
Figure 6 Criticality Solution Technique.....	21
Figure 7 Boundary Conditions	22
Figure 8 Example 4x4-Three Dimensional Mesh System	23
Figure 9 Relabeled Interior Mesh Points.....	24
Figure 10 Criticality Solution Technique for Two Energy Groups	28
Figure 11 Typical Drop Down File Menu.....	33
Figure 12 Welcome Screen	34
Figure 13 Reactor Schematic Form.....	35
Figure 14 Boundary Condition Form	35
Figure 15 2D, One Energy Group Form	37
Figure 16 3D, One Energy Group Form	38
Figure 17 3D, Two Energy Groups Axial Choice.....	39
Figure 18 3D, Two Energy Group Form.....	40
Figure 19 Flux profile of infinite right circular cylinder (Bessel J function).....	42
Figure 20 Normalized flux from one-speed computer model	43
Figure 21 Flux profile of Bessel J function and one-speed computer model together....	43
Figure 22 Initial Test, Mesh Spacing = 3 cm	48
Figure 23 3D, Total 2 Energy Groups, Blocked Tridiagonal Tolerance = 1E-8 with Mesh Spacing = 1 cm.....	51
Figure 24 Fuel-Cell Homogenization.....	55
Figure 25 Two Dimensional Output with Mesh Spacing = 0.5 cm.....	98
Figure 26 3D, One Energy Group, Half Core Plot with Mesh Spacing = 6 cm.....	99
Figure 27 3D, One Energy Group, Quarter Core Plot with Mesh Spacing = 6 cm.....	99
Figure 28 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups.....	100
Figure 29 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups.....	100
Figure 30 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups Half Rx	101
Figure 31 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups Half Rx	101
Figure 32 Radial Plot, Mesh Spacing = 30 cm.....	102
Figure 33 Axial Plot, Mesh Spacing = 30 cm	102
Figure 34 Radial Plot, Mesh Spacing = 15 cm.....	103
Figure 35 Axial Plot, Mesh Spacing = 15 cm	103
Figure 36 Radial Plot, Mesh Spacing = 10 cm.....	104
Figure 37 Axial Plot, Mesh Spacing = 10 cm	104
Figure 38 Radial Plot, Mesh Spacing = 5 cm.....	105
Figure 39 Axial Plot, Mesh Spacing = 5 cm	105
Figure 40 Radial Plot, Mesh Spacing = 4 cm.....	106
Figure 41 Axial Plot, Mesh Spacing = 4 cm	106
Figure 42 Radial Plot, Mesh Spacing = 3 cm.....	107

Figure 43 Axial Plot, Mesh Spacing = 3 cm	107
Figure 44 Radial Plot, Mesh Spacing = 2 cm.....	108
Figure 45 Axial Plot, Mesh Spacing = 2 cm	108
Figure 46 Radial Plot, Mesh Spacing = 1 cm.....	109
Figure 47 Axial Plot, Mesh Spacing = 1 cm	109
Figure 48 Radial Plot, Mesh Spacing = 30 cm.....	110
Figure 49 Axial Plot, Mesh Spacing = 30	110
Figure 50 Radial Plot, Mesh Spacing = 20 cm.....	111
Figure 51 Axial Plot, Mesh Spacing =20	111
Figure 52 Radial Plot, Mesh Spacing = 15 cm.....	112
Figure 53 Axial Plot, Mesh Spacing = 15 cm	112
Figure 54 Radial Plot, Mesh Spacing = 10 cm.....	113
Figure 55 Axial Plot, Mesh Spacing = 10 cm	113
Figure 56 Radial Plot, Mesh Spacing = 5 cm.....	114
Figure 57 Axial Plot, Mesh Spacing = 5 cm	114
Figure 58 Radial Plot, Mesh Spacing = 4 cm.....	115
Figure 59 Axial Plot, Mesh Spacing = 4 cm	115
Figure 60 Radial Plot, Mesh Spacing = 3 cm.....	116
Figure 61 Axial Plot, Mesh Spacing = 3 cm	116
Figure 62 Radial Plot, Mesh Spacing = 2 cm.....	117
Figure 63 Axial Plot, Mesh Spacing = 2 cm	117
Figure 64 Radial Plot, Mesh Spacing = 1 cm.....	118
Figure 65 Axial Plot, Mesh Spacing = 1 cm	118
Figure 66 Sample OLE Embedded Object.....	122

List of Tables

Table	Page
Table 1 One-Speed Reactor Input Data.....	41
Table 2 Two Group Diffusion Theory Constants (Table 7-2 Duderstadt).....	46
Table 3 Relative Errors for Quarter Core 3D Models, Blocked Tridiagonal Tolerance =1E-8 with Maximum of 20 Iterations.....	47
Table 4 Relative Errors for Half Core 3D Models, Blocked Tridiagonal Tolerance =1E-8 with Maximum of 20 Iterations.....	47
Table 5 Relative Error for Quarter Core 3D, Blocked Tridagonal Tolerance = 0.001	49
Table 6 Relative Errors for Half Core 3D, Blocked Tridiagonal Tolerance = 0.001	49
Table 7 Half Core 3D, Blocked tridiagonal Tolerance = 1E-4	50
Table 8 Half Core 3D, Blocked Tridiagonal Tolerance = 1E-6	50
Table 9 Half Core 3D, Blocked Tridiagonal Tolerance = 1E-8	50
Table 10 Half Core 3D, k Tolerance =1E-7, Blocked Tridiagonal Tolerance = 1E-8	52
Table 11 Options for Three Dimensional Models.....	52
Table 12 Excel Constants for Charts.....	120

Abstract

A computer model of a pressurized water reactor (PWR) was developed for use as a teaching tool in graduate level nuclear reactor courses. The development, based on the diffusion equation, includes the methodology for solving the steady state spatial dependence of the neutron power output in a homogeneous right circular cylinder unreflected PWR system. This includes a two dimensional one energy group model, a three dimensional one energy group model, and a three dimensional two energy group model.

To solve the homogeneous diffusion equation, a method was developed to search for criticality of the reactor based on the geometry and reactor core material composition. For the one energy group models, a perturbation technique was developed to assist the program user in modifying the macroscopic absorption coefficient to drive the reactor to criticality. For the three dimensional models, a blocked tridiagonal solver was developed to solve the numerical linear system of equations approximating the diffusion equation.

The model was coded using Visual BASIC 5.0™. This provides a platform that is exportable to personal computers and allows direct linkage to Windows based programs. The code automatically charts and displays the power distribution profile using Excel™ and displays the calculated multiplication factor determining criticality.

MODELING PRESSURIZED WATER REACTOR KINETICS

I. Introduction

Background

Basic nuclear reactor courses at the graduate and undergraduate level focus on teaching students how to calculate radial and axial flux and power for steady state (non-time dependent) reactors. Many nuclear reactor textbooks cover the fundamentals of nuclear physics and apply the diffusion equation to approximate the behavior of neutrons within the reactor core. Typically, the reactor books review one-dimensional, one speed, homogeneous models for various geometric shapes in great detail. Some even outline numerical approaches for solving the approximate solution.

While students often gain a basic understanding of the general physics, they typically lack a qualitative and intuitive understanding of the reactor core nucleonic behavior based on core geometry and composition. A computer model can be used to provide the students with a tool that can visually explain how flux and power are impacted by changing the core geometry and composition.

Problem Statement

The problem statement for this thesis was to develop a working computer model of a pressurized water reactor (PWR) for use as a teaching tool in graduate level nuclear

reactor courses. The development includes the equations and methodology for solving the steady state spatial dependence of the neutron flux and power output in a homogeneous right circular cylinder unreflected PWR system. This includes a two dimensional one energy group model, a three dimensional one energy group model, and a three dimensional two energy group model.

Approach

The fundamental approach was to model the reactor using the diffusion equation. For a steady state system, the diffusion equation reduces to the Helmholtz equation.

$$\nabla^2 \phi(r) + B_g^2 \phi(r) = 0$$

where

$$B_g^2 = \text{Geometric buckling} = \frac{\nu \Sigma_f - \Sigma_a}{D} \text{ (1/cm}^2\text{)}.$$

Since this is a homogeneous equation, one must determine the eigenvalues to achieve a non-trivial solution. For cylindrical geometries, the eigenfunction corresponding to the smallest eigenvalue is non-negative everywhere within the reactor. This is physically the only value of importance because the flux cannot be negative in a reactor.

To achieve a physical solution, we rewrite the Helmholtz equation and insert an eigenvalue $\frac{1}{k}$.

$$-\nabla \cdot D \nabla \phi(r) + \Sigma_a \phi(r) = \frac{1}{k} \nu \Sigma_f \phi(r)$$

Where

ν = average number of neutrons per fission

Σ_a = the probability of absorption per unit path length (1/cm).

For a particular value of k , this equation will have a unique solution. As will be shown later, if k equals one the reactor is critical. If k does not equal one, the core geometry and/or material composition must be changed. Searching for the flux when k equals one is called the criticality search. This criticality search, using the diffusion equation, is the basis for the development of the code.

The first step in developing the code was to solve the two dimensional, one energy group diffusion equation using the finite central difference method. The finite central difference method provides a satisfactory order of accuracy and is generally used as the initial method for modeling or designing reactors. The finite difference method results in solving a tridiagonal matrix system using a power iterative technique to solve for the flux at criticality. The program in this thesis uses the Crout factorization method to solve the tridiagonal system of equations. A perturbation technique is used to perturb an initial guess of the macroscopic cross section to drive the modeled reactor to a critical level. This perturbation will assist the user in selecting the macroscopic cross section that will result in a critical condition. The one group model assumes that the energy of the neutrons is equal at every spatial point within the reactor. The model, based on a homogeneous un-reflected reactor, which is not time dependent, yields a two dimensional solution of power versus radial position. The initial two dimensional model was developed using FORTRAN™ and then later converted to Visual BASIC 5.0™.

The two dimensional one group model was expanded to a third dimension by adding a solution in the axial direction. The output of this model provides both radial and axial power plots in three dimensions. I used the finite central difference method to approximate the diffusion equation. This changed the system from a pure tridiagonal to a

blocked tridiagonal system because of the additional sub and super diagonals. This required the development of a blocked tridiagonal solver to solve the system of equations and provide the flux at each interior mesh point. With all flux values known at the interior points, both the axial and radial power distributions can then be plotted. The three dimensional model was written in Visual BASIC 5.0.

The final step was to develop a two energy group three dimensional model. The model assumes only down scatter of neutrons that are directly coupled, meaning neutrons only scatter to the next lowest energy level. This model uses the finite central difference method and the blocked tridiagonal solver.

Visual BASIC 5.0 was chosen because it offers many advantages over scientific languages such as FORTRAN. It allows the programmer to build an executable file that links automatically into simple plotting tools such as Excel. With Visual BASIC 5.0, you can command and control Excel as well as other Microsoft Windows software. For example, the Visual BASIC 5.0 reactor code populates an Excel spreadsheet with the solution data and then builds the charts all from within Visual BASIC 5.0. The charts are linked and updated to appear as an object on the Visual BASIC 5.0 form. This capability provides the user with automated graphs of the power based upon the input parameters as well as access to the output data on spreadsheets. This advantage precludes the user from having to manually create the charts or plots in another computer language. These features outweigh the advantages of FORTRAN such as computational speed and built-in intrinsic functions. Additionally, one can export the program packaged with the runtime dynamic link language, thus not requiring Visual BASIC.

The advancements in computer technology have made using Visual BASIC 5.0 an alternative to scientific programs. Less than five years ago personal computers were too slow to solve three dimensional diffusion problems using Visual BASIC 5.0. The low cost and improvements of memory and processors allow personal computers to be capable of solving complex numerical problems using Visual BASIC 5.0 in a fraction of previous times.

II. Theoretical Development

Diffusion Theory

“Reactor kinetics is the area of reactor physics concerned with predicting what happens to the neutron flux density when the balance condition associated with the critical state is disturbed (Henry, 1986:296).” The generation of heat in a reactor system is proportional to the fission rate, which is a function of the neutron flux. The neutrons in a thermal reactor range in energies from 0.001 eV to about 10 MeV. To simplify the design process of reactors, neutrons are divided into energy groups. The one group model deals with the thermal neutrons only; however, it also accounts for those produced from both prompt and delayed neutrons. The two-group model deals separately with both thermal and fast neutrons.

It is common practice to approximate the exact neutron transport equation using diffusion theory. The neutron transport equation accounts for the angular dependent neutron density within a volume. The diffusion equation is the result of removing the angular dependence from the transport equation.

The diffusion equation is based on Fick’s Law and the equation of continuity. Fick’s law is shown in equation (1).

$$J_x = -D \frac{d\phi}{dx} \quad (1)$$

where

J_x = the net number of neutrons passing a unit area perpendicular to the x-direction in a unit of time

D = the diffusion coefficient (cm)

ϕ = the flux (neutrons/cm³)(cm/sec)

Fick's law was originally used to predict the flow of chemicals from one region of higher concentration to another region of lower concentration solute. The flow was found to be equal to the negative gradient of the solute concentration. Although neutrons do not actually flow, their behavior can be modeled using this concept (Lamarsh, 1983:192). Early reactors were designed using this technique. Today, more sophisticated and computationally demanding methods are available to design reactor cores.

To develop the diffusion equation one begins by using the equation of continuity.

The equation of continuity states that:

$$\left[\begin{array}{l} \text{The rate of change in} \\ \text{number of neutrons per} \\ \text{volume (V)} \end{array} \right] = \left[\begin{array}{l} \text{production rate} \\ \text{of neutrons in V} \end{array} \right] - \left[\begin{array}{l} \text{absorption rate} \\ \text{of neutrons in V} \end{array} \right] - \left[\begin{array}{l} \text{leakage rate of} \\ \text{neutrons in V} \end{array} \right]. \quad (2)$$

By substituting Fick's law into the equation of continuity, the general diffusion equation becomes:

$$D\nabla^2\phi - \Sigma_a\phi + \nu\Sigma_f\phi = \frac{dn}{dt}, \quad (3)$$

where equation (3) is the non-steady state diffusion equation and

$$\begin{aligned} D &= \text{the diffusion coefficient (cm)} \\ \nabla^2 &= \text{the Laplacian (divergence of the gradient)} \\ \phi &= \text{the neutron flux (neutron cm/cm}^3 \text{ sec)} \\ \Sigma_a &= \text{macroscopic absorption cross section (1/cm)} \\ \Sigma_f &= \text{macroscopic fission cross section (1/cm)} \\ \nu &= \text{neutrons/fission.} \end{aligned} \quad (4)$$

Removing the time dependence results in the Helmholtz equation.

$$-D\nabla^2\phi + \Sigma_a\phi = \nu\Sigma_f\phi \quad (5)$$

This is the fundamental equation to be solved for the solutions to the problem statement.

The development of the boundary conditions is key to the solution of the diffusion equation for finite cylindrically shaped reactor cores. In order to develop a physical meaning, the total flux must be positive and real in all areas within the core. The diffusion equation and Fick's law are not valid at physical boundaries since they approximate the value several mean free paths inside the boundary. To account for the physical boundaries, the diffusion method models the measured flux by assuming the flux is zero at an extrapolated distance beyond the outer physical boundary layer of the reactor core. The exact flux does not reduce to zero beyond the boundary; however, the diffusion theory assumption allows for reasonable flux calculations within a few mean free paths of the boundary (Duderstadt and Hamilton, 1976:144). See Figure 1 for a graphical comparison of the measured flux and the diffusion theory.

The extrapolated distance for plane geometries is calculated by using equation (6).

$$d = 0.71\lambda_{tr} \text{ (cm)} \quad (6)$$

where the transport mean free path is

$$\lambda_{tr} = 3D = \frac{1}{\Sigma_{tr}}$$

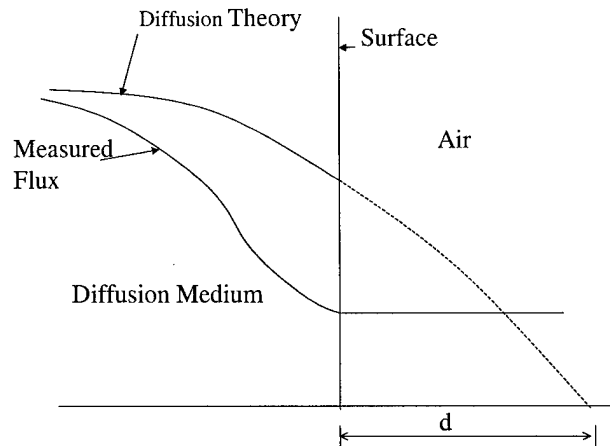


Figure 1 Extrapolated distance at outer boundary

However, for relatively large reactors the extrapolated distance can be neglected without significantly impacting the order of accuracy because the extrapolated distance is on the order of centimeters or less as compared to the radius of approximately one to two meters on average. In this model, the assumption is made that the flux is zero at the top, bottom, and sides of the reactor core and the derivative of the axial and radial flux at the centerline of the reactor equals zero. This is accomplished by setting the flux at the centerline equal to the flux at the first interior mesh point away from the centerline.

The multi-group diffusion equation discretizes the range of neutron energies into energy groups as shown in Figure 2. Notice that the grouping begins with the highest energy group number and works toward the lowest energy group number. The highest energy group number corresponds to the lowest energy level of the neutrons.

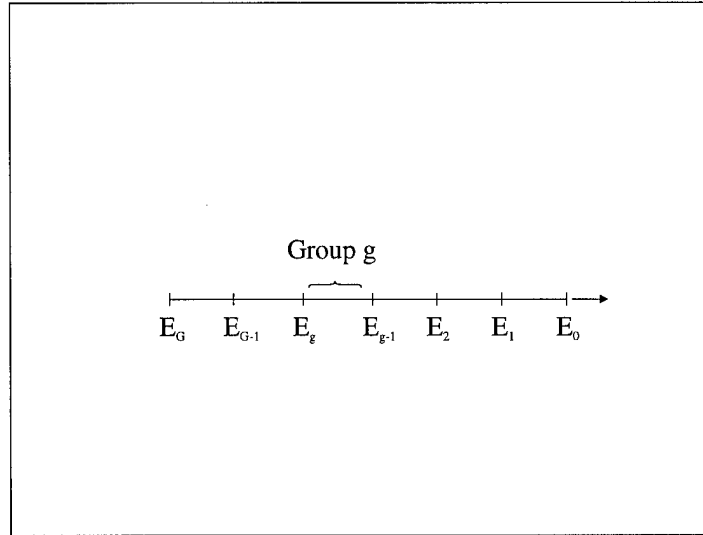


Figure 2 Energy Groups

Equation (7) can be described by the energy dependent version of the diffusion equation. The equation is based on integrating the neutron energy (averaging) over the energy group of concern, $E_g < E < E_{g-1}$.

$$\begin{aligned}
 \left[\begin{array}{l} \text{The rate of change} \\ \text{of neutrons in} \\ \text{Group g} \end{array} \right] &= - \left[\begin{array}{l} \text{Change due to} \\ \text{leakage} \end{array} \right] - \left[\begin{array}{l} \text{absorption in} \\ \text{group g} \end{array} \right] + \left[\begin{array}{l} \text{source} \\ \text{neutrons} \\ \text{in group g} \end{array} \right] \\
 &\quad - \left[\begin{array}{l} \text{neutrons} \\ \text{scattering} \\ \text{out of group g} \end{array} \right] + \left[\begin{array}{l} \text{neutrons} \\ \text{scattering into} \\ \text{group g} \end{array} \right]
 \end{aligned} \tag{7}$$

For the two energy group model, the energy groups are shown in Figure 3.

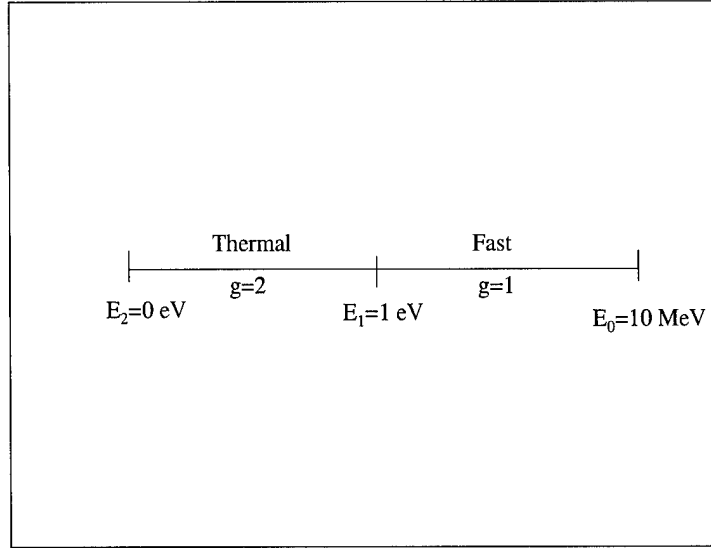


Figure 3 Energy Spectra for Two Energy Group Model

The development of the two energy group system is based on the assumptions that all fission neutrons are born in the fast group and that there is no up scatter from the thermal group. The final form of the two energy group diffusion equation becomes

$$-\nabla \cdot D_1 \nabla \phi_1 + \Sigma_{R1} \phi_1 = \frac{1}{k} (v_1 \Sigma_{f1} \phi_1 + v_2 \Sigma_{f2} \phi_2) \quad (8)$$

$$-\nabla \cdot D_2 \nabla \phi_2 + \Sigma_{a2} \phi_2 = \Sigma_{S12} \phi_1 \quad (9)$$

where

subscripts 1 and 2 refer to groups 1 and 2 respectively

$$\Sigma_{R1} = \Sigma_{t1} - \Sigma_{S12}$$

Σ_{S12} = Cross section for scatter from group 1 to group 2.

Two Dimensional, One Energy Group Model

To develop this model, I chose to use a criticality search technique outlined in several references (Duderstadt and Hamilton, 1976:214-226), (Clark and Hansen, 1964:175-178), (Glasstone and Sesonske, 1981:208-213), and (Ott, 1989:349-356). This section will derive the methodology for the criticality search and the numerical development to solve the two dimensional, one energy group diffusion equation.

As stated earlier, the energy level of neutrons within a typical PWR ranges from about 0.001 eV to 10 MeV. Modeling the reactor proves very complicated when attempting to incorporate the entire energy range. Historically, attempts to solve the diffusion equation assumed all neutrons were at the same energy level. The key to solving the one group model is selecting the appropriate macroscopic cross section data. The cross sections are dependent upon the neutron energy level. By choosing the appropriate cross section, the one group model can provide quantitative as well as qualitative analysis of the reactor behavior. The parameters chosen for this model were based on the homogenized data from a typical reactor (Duderstadt and Hamilton, 1976:210). Certainly, nuclear reactor designers would not use the one group model for design purposes. The value of using a one group model is its ease of calculation and proven qualitative similarity to more rigorous models.

Criticality Solution Technique.

Determining the flux at criticality becomes an eigenvalue problem as

$$-\nabla \cdot D \nabla \phi(r) + \Sigma_a \phi(r) = \frac{1}{k} \nu \Sigma_f \phi(r) \quad (10)$$

where $\frac{1}{k}$ is the eigenvalue. For criticality, we seek k equal to one. Rewritten in matrix

form, equation (10) yields

$$M\phi = \frac{1}{k}F\phi \quad (11)$$

where

$$F = v\Sigma_f \text{ (1/cm)}$$

$$\text{and the operator } M = -\nabla \cdot D\nabla + \Sigma_a \text{ (1/cm).}$$

To solve this problem, we guess an initial "source" term $S(r)$ and k value where

$$S(r) = F\phi(r) \cong S^{(0)}(r) \text{ and } k \cong k^{(0)} \quad (12)$$

and solve for the flux $\phi^{(1)}$ using a tridiagonal solver.

$$M\phi^{(1)} = -\nabla \cdot D\nabla\phi^{(1)} + \Sigma_a\phi^{(1)} = \frac{1}{k^{(0)}}S^{(0)} \quad (13)$$

After solving for the flux we must recalculate the source and k values. The source is easy to recalculate based on known values.

$$S^{(1)} = F\phi^{(1)} = v\Sigma_f\phi^{(1)} \quad (14)$$

The iterative scheme is shown in equations (15) and (16). This repetitive process yields the flux at successive values until equation (17) is approximately true within set tolerances.

$$M\phi^{(n+1)} = \frac{1}{k^{(n)}}S^{(n)} \quad (15)$$

$$S^{(n+1)} = F\phi^{(n+1)} \quad (16)$$

$$M\phi^{(n+1)} \cong \frac{1}{k^{(n+1)}}F\phi^{(n+1)} \quad (17)$$

To solve for the next k iterative value, we recognize

$$M\phi^{(n+1)} = \frac{1}{k^{(n)}} F\phi^{(n)} \cong \frac{1}{k^{(n+1)}} F\phi^{(n+1)}. \quad (18)$$

Solving for $k^{(n+1)}$, we then integrate the flux over space. This is essentially averaging the values to obtain a new eigenvalue, where

$$k^{(n+1)} \cong \frac{\int S^{(n+1)}(r)d^3r}{\frac{1}{k^{(n)}} \int S^{(n)}(r)d^3r} \cong \frac{\int_0^R S^{n+1}(r)dr}{\frac{1}{k^{(n)}} \int_0^R S^n(r)dr}. \quad (19)$$

The integration is accomplished numerically using the composite trapezoid rule

$$\int_a^b S(r)dr \cong \frac{\Delta r}{2} \left(S(a) + S(b) + 2 \sum_{i=1}^{n-1} S(r_i) \right) \quad (20)$$

where n is the number of mesh points. The iteration process continues until the tolerances for k and the source are within a specified tolerance.

$$\left| \frac{k^{(n)} - k^{(n-1)}}{k^{(n)}} \right| < \varepsilon_1 \text{ and/or } \left| \frac{S^{(n)} - S^{(n-1)}}{S^{(n)}} \right| < \varepsilon_2 \quad (21)$$

where

$$\varepsilon_1 = 0.00001$$

$$\varepsilon_2 = 0.015$$

The tolerance setting for ε_1 is critical to achieving low relative errors compared to the mathematical solution. Ott recommends a tolerance of 1E-5 for most calculations (Ott, 1986; 351). See Chapter III, Program Validation for details. As the number of iterations gets large, we expect the flux to converge to the fundamental eigenfunction (Duderstadt and Hamilton, 1976:216-219). This will provide the correct flux mode shape to enable power and flux calculations. Figure 4 is a flowchart of the technique used in the code to solve for the flux and criticality based on the core material composition and geometry input.

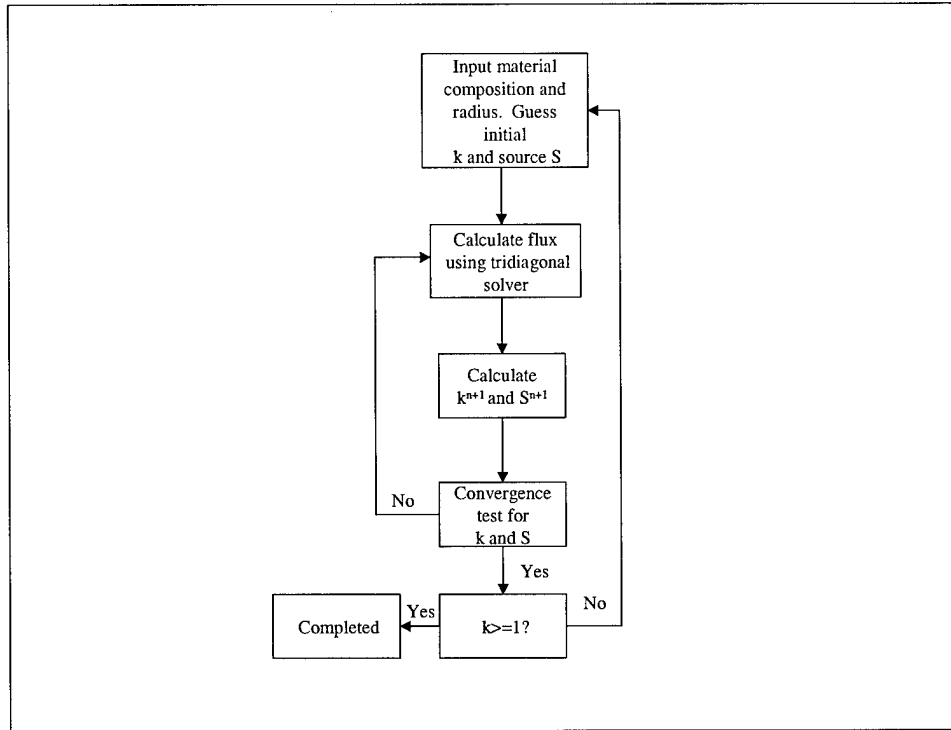


Figure 4 Criticality Search Technique, 2D Model

One must provide an initial guess for k and the source flux in order for the power iteration process to converge resulting in criticality. Because it is difficult to guess a sufficiently close guess, one must use perturbation theory. Perturbation methodology assists the user by adjusting the macroscopic cross section until criticality is met.

Changing the macroscopic cross section by some small amount such as

$$\Sigma'_a(r) = \Sigma_a(r) + \delta\Sigma_a(r), \quad (22)$$

where

Σ'_a is the value perturbed by some small positive or negative change $\delta\Sigma_a$, yields a revised equation in matrix form.

$$M'\phi' = \frac{1}{k'} F\phi' \quad (23)$$

The perturbation in the cross section changes the diffusion operator.

$$M' = M + \delta M' \quad (24)$$

where

$$\delta M' \equiv \delta \Sigma_a(r).$$

We then calculate the change in k by applying the scalar product equation (23) with the adjoint flux ϕ^* of the unperturbed core obtaining equation (25).

$$\langle \phi^*, M\phi' \rangle + \langle \phi^*, \delta M\phi' \rangle = \frac{1}{k} \langle \phi^*, F\phi' \rangle \quad (25)$$

Using the inner product of the adjoint operator, yields

$$\langle \phi^*, M\phi' \rangle = \langle M^*\phi^*, \phi' \rangle = \left\langle \frac{1}{k} F^*\phi^*, \phi' \right\rangle = \frac{1}{k} \langle \phi^*, F\phi' \rangle \quad (26)$$

where, for the one group diffusion model

ϕ^*, F^* , and M^* are the adjoint values

$$\phi = \phi^*$$

$$F = F^*$$

$$M = M^*.$$

Substituting equation (26) into equation (25) yields

$$\left(\frac{1}{k'} - \frac{1}{k} \right) = \frac{\langle \phi^*, \delta M\phi' \rangle}{\langle \phi^*, F\phi' \rangle}. \quad (27)$$

However, this requires us to know the adjoint and perturbed fluxes that cannot be calculated directly. We can rewrite the left-hand side in terms of reactivity.

$$\Delta\rho = \left(\frac{1}{k} - \frac{1}{k'} \right) = - \frac{\langle \phi^*, \delta M\phi' \rangle}{\langle \phi^*, F\phi' \rangle} \quad (28)$$

Using perturbation theory we can translate the unknowns into known values. A small change in absorption cross section is assumed to result in a small change in flux (Duderstadt and Hamilton, 1976:223). Expanding equation (28) provides the following.

$$\Delta\rho = -\frac{\langle\phi^*,\delta\Sigma_a\phi\rangle}{\langle\phi^*,F\phi\rangle} - \frac{\langle\phi^*,\delta\Sigma_a\delta\phi\rangle}{\langle\phi^*,F\phi\rangle} - \frac{\langle\phi^*,\delta\Sigma_a\phi\rangle\langle\phi^*,\Sigma_a\delta\phi\rangle}{\langle\phi^*,F\phi\rangle^2} + \dots \quad (29)$$

Using the self adjointness of the flux provides the form required to calculate the small change in macroscopic cross section,

$$\Delta\rho \cong -\frac{\int\phi(r)^2\delta\Sigma_a(r)d^3r}{\int\phi(r)^2\nu\Sigma_f(r)d^3r} = \frac{\delta\Sigma_a}{\nu\Sigma_f} \quad (30)$$

$$\delta\Sigma_a = \left(\frac{1}{k} - \frac{1}{k'}\right)\nu\Sigma_f \quad (31)$$

where, for criticality,

$$k' = 1.$$

With a known change in the macroscopic absorption cross section, the program user can iterate the program until criticality is achieved for the geometry and material composition specified, if achievable. The change in the macroscopic absorption cross section can only be accomplished physically by changing the material composition in the homogeneous reactor core because

$$\Sigma_a = \sum_{i=1}^n N_i\sigma_a^i \quad (32)$$

where

n = the number of materials

N_i = the number density of the material i (neutrons/cm³)

σ_a^i = the microscopic absorption cross section of material i (cm²).

Numerical Development.

The numerical development of the one group model is based on the central difference approximation to the diffusion equation in right circular cylinder coordinates.

$$\phi''(r) = -\frac{1}{r}\phi'(r) + \frac{\Sigma_a}{D}\phi(r) - \frac{1}{D}v\Sigma_f\phi(r) \quad (33)$$

Expanding ϕ in a Taylor series about r

$$\phi_{i+1} = \phi_i + \Delta \left. \frac{d\phi}{dr} \right|_i + \frac{\Delta^2}{2} \left. \frac{d^2\phi}{dr^2} \right|_i + \dots \quad (34)$$

$$\phi_{i-1} = \phi_i - \Delta \left. \frac{d\phi}{dr} \right|_i + \frac{\Delta^2}{2} \left. \frac{d^2\phi}{dr^2} \right|_i + \dots \quad (35)$$

Now adding equations (34) and (35) yields the standard central difference formula with an order of Δ^2 .

$$\left. \frac{d^2\phi}{dr^2} \right|_i \equiv \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta^2}. \quad (36)$$

For the standard differential the central difference yields

$$\left. \frac{d\phi}{dr} \right|_i = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta}. \quad (37)$$

The final form of the numerical equation becomes

$$\left(-1 - p(r)\frac{h}{2}\right)\phi_{i-1} + (2 + h^2q(r))\phi_i + \left(-1 + p(r)\frac{h}{2}\right)\phi_{i+1} = -h^2r(r) \quad (38)$$

where

h is the distance between nodes (cm)

$$p(r) = \frac{1}{r_i} \text{ (1/cm)}$$

$$q(r) = \frac{\sum_a}{D} \text{ (1/cm}^2\text{)}$$

$$r(r) = \frac{1}{kD} \nu \sum_f \phi_0 \left(\frac{1}{\text{cm}^2} \frac{\text{neutrons} - \text{cm}}{\text{cm}^3 - \text{sec}} \right).$$

The boundary conditions are shown in Figure 5 for a typical reactor core.

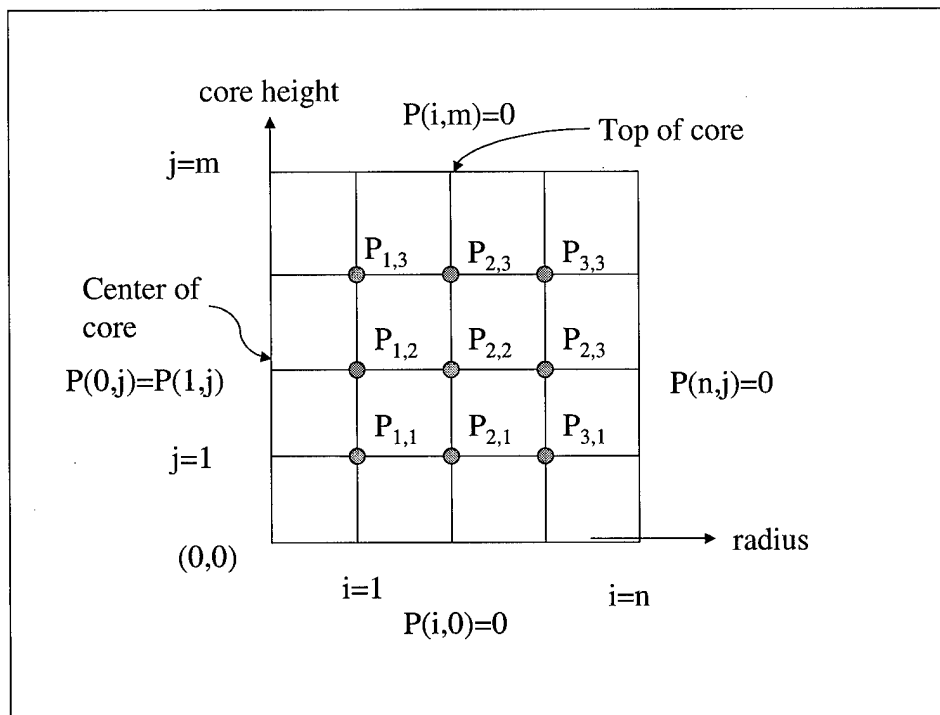


Figure 5 Boundary Conditions

Using a standard tridiagonal solver rapidly provides the flux values for this system of equations along the radius of the core.

Three Dimensional, One Energy Group Model

I used the same criticality search technique for this model as in the two dimensional model; however, the numerical solution technique is quite different. Adding the third dimension increased the boundary conditions and allowed me to develop a model that allows the user to choose to calculate the power distribution and criticality in either a half or a quarter of the reactor core. Duderstadt recommends using a Gauss-Seidel or a successive relaxation method to solve the numerical equations (Duderstadt and Hamilton, 1976:191). I chose to develop and use a blocked tridiagonal solver because it reduced the computational time and computer memory requirements over those recommended.

Criticality Solution Technique

The three dimensional model uses the same criticality iterative search technique to solve for the flux as the two dimensional model; however, the derivation of the volume source integration used in equation (19) to solve for k^{n+1} is more complicated. See Appendix A for a complete derivation. As shown in Figure 6, the three dimensional model uses the blocked tridiagonal solver vice the tridiagonal solver for the two dimensional model.

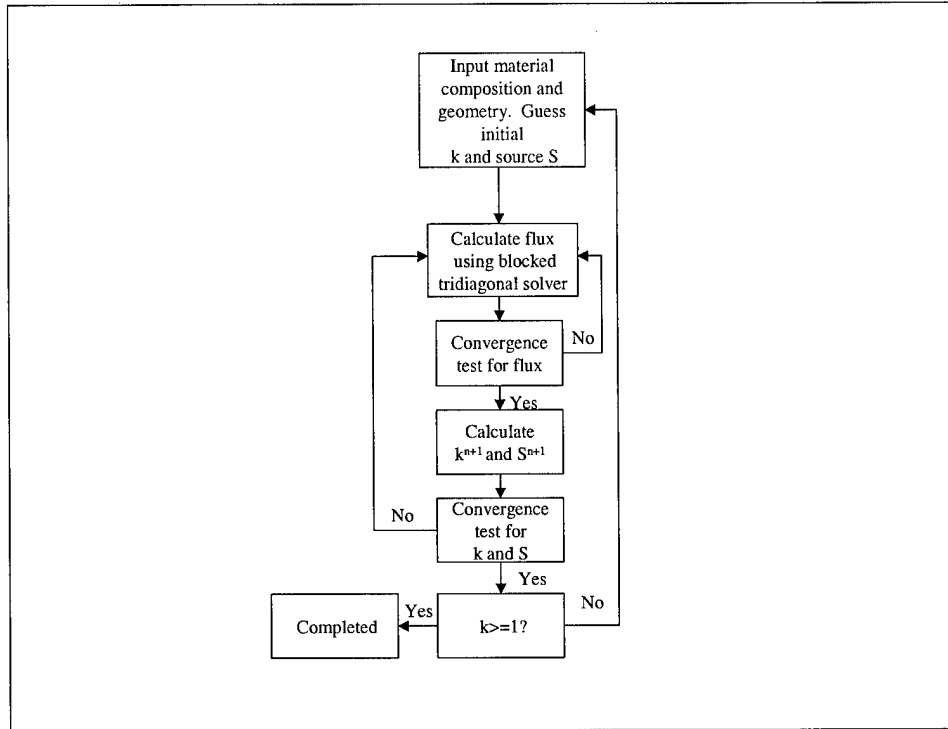


Figure 6 Criticality Solution Technique

Numerical Development.

In three dimensions, the diffusion equation in right circular cylinder coordinates becomes

$$\left(\frac{\partial^2 \phi}{\partial r^2} + \frac{\partial \phi}{r \partial r} + \frac{\partial^2 \phi}{\partial z^2} \right) - \frac{\Sigma_a \phi}{D} = \frac{-S}{D} \quad (39)$$

where the appropriate Laplacian is

$$\nabla^2 = \frac{1}{r} \frac{\partial}{\partial r} r \frac{\partial}{\partial r} + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2} + \frac{\partial^2}{\partial z^2}.$$

Owing to symmetry, $\frac{\partial^2}{\partial \theta^2} = 0$.

Using central differences and collecting terms yields

$$\left(\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta r^2} \right) + \left(\frac{\phi_{i+1,j} - \phi_{i-1,j}}{r2\Delta r} \right) + \left(\frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta z^2} \right) - \frac{\Sigma_a}{D} \phi_{i,j} = \frac{-S}{D} \quad (40)$$

and

$$\left(\frac{1}{\Delta r^2} - \frac{1}{2r\Delta r} \right) \phi_{i-1,j} + \left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta z^2} - \frac{\Sigma_a}{D} \right) \phi_{i,j} + \left(\frac{1}{\Delta r^2} + \frac{1}{2r\Delta r} \right) \phi_{i+1,j} + \frac{1}{\Delta z^2} (\phi_{i,j-1} + \phi_{i,j+1}) = \frac{-S}{D} \quad (41)$$

where the boundary conditions are shown in Figure 7. The model provides solutions for half of the reactor core or quarter of the reactor core owing to symmetry of the homogeneous system.

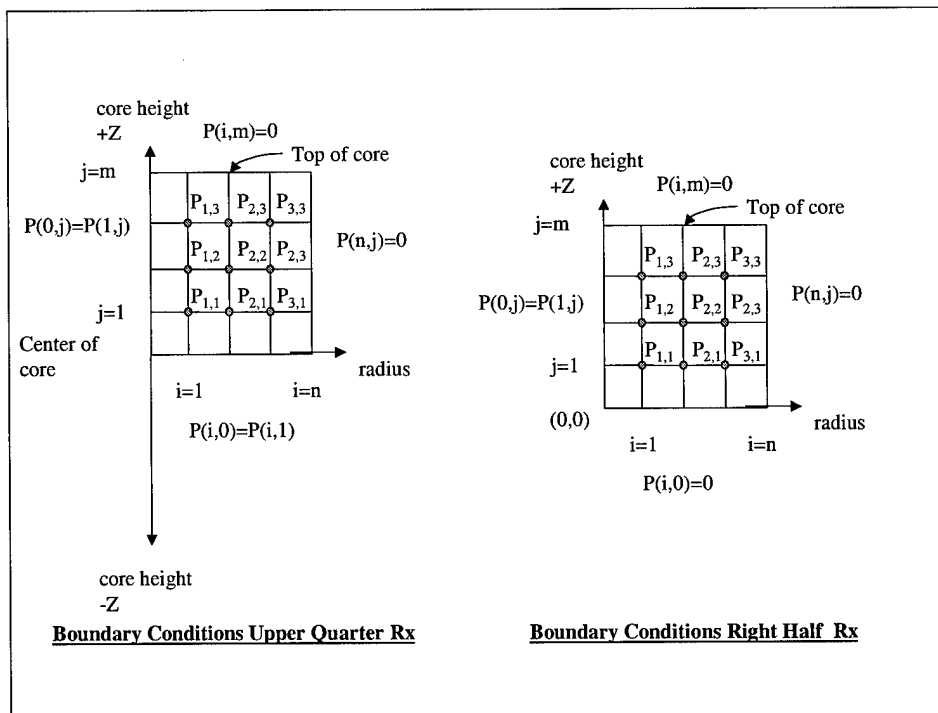


Figure 7 Boundary Conditions

To explain the method of converting the numerical equation into a system of linear equations, I will use the sample mesh system shown in Figure 8.

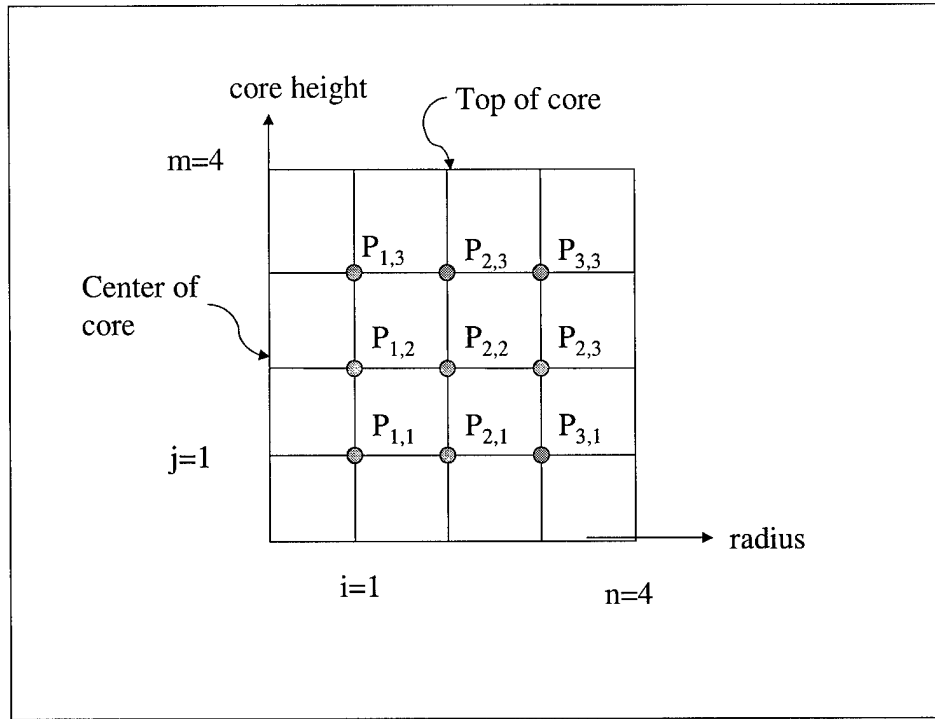


Figure 8 Example 4x4-Three Dimensional Mesh System

To convert these equations into a solvable linear algebra system, it is necessary to convert the (i,j) indices into a single value using

$$l = i + (m-1-j)(n-1) \quad (42)$$

where m is the number of nodes along the z -axis and n is the number of nodes along the radius (Barden and Faires, 1997:676). This conversion results in re-numbering the interior mesh points as shown in Figure 9.

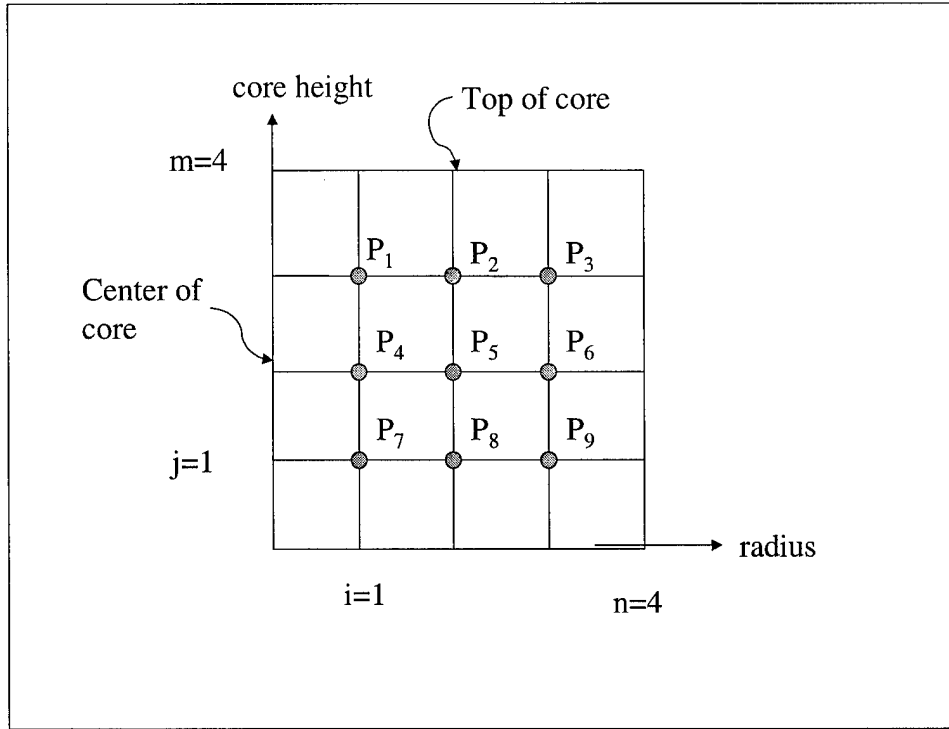


Figure 9 Relabeled Interior Mesh Points

Referring to equation (41), this system has five terms instead of the three terms used in the two-dimension model. This corresponds to the sample 9×9 matrix shown in equation (43). The matrix is a tridiagonal system with a sub and super diagonal. The sub and super diagonal, in this case, contain constant and equal values in each component. The upper and lower tridiagonal diagonals are variables that depend on the position along the radius as shown in equation (41).

$$\begin{bmatrix}
P_1 & P_2 & 0 & P_4 & 0 & 0 & 0 & 0 & 0 \\
P_1 & P_2 & P_3 & 0 & P_5 & 0 & 0 & 0 & 0 \\
0 & P_2 & P_3 & 0 & 0 & P_6 & 0 & 0 & 0 \\
\hline
P_1 & 0 & 0 & P_4 & P_5 & 0 & P_7 & 0 & 0 \\
0 & P_2 & 0 & P_4 & P_5 & P_6 & 0 & P_8 & 0 \\
0 & 0 & P_3 & 0 & P_5 & P_6 & 0 & 0 & P_9 \\
\hline
0 & 0 & 0 & P_4 & 0 & 0 & P_7 & P_8 & 0 \\
0 & 0 & 0 & 0 & P_5 & 0 & P_7 & P_8 & P_9 \\
0 & 0 & 0 & 0 & 0 & P_6 & 0 & P_8 & P_9
\end{bmatrix}
\begin{bmatrix}
\phi_1 \\
\phi_2 \\
\phi_3 \\
\phi_4 \\
\phi_5 \\
\phi_6 \\
\phi_7 \\
\phi_8 \\
\phi_9
\end{bmatrix}
=
\begin{bmatrix}
B_1 \\
B_2 \\
B_3 \\
B_4 \\
B_5 \\
B_6 \\
B_7 \\
B_8 \\
B_9
\end{bmatrix}
\tag{43}$$

This same pattern applies to any size matrix depending upon the number of interior mesh points selected. The boundary conditions are incorporated into positions P_1 , P_4 , and P_7 in the matrix.

Matrix Solution Methods.

There are several methods available to solve these types of matrix problems. One is the Jacobi method. This method converges too slowly for practical use in large matrices because of the number of required operations. Another method, Gauss-Seidel, is often used to solve small to medium sized matrices. Gauss-Seidel also converges slowly and, like the Jacobi method, requires storing every point in the matrix in computer memory. As a result, it is slow and computationally inefficient. Successive over-relaxation (SOR) is an improved version of the Gauss-Seidel method. It makes an over correction by anticipating future corrections. To reduce the error by a factor of 10^{-p} , the SOR method requires on the order of J iterations compared to J^2 for the other methods (Press, 1996:858) where

$$r \approx \frac{1}{3} pJ \tag{44}$$

and

r = the r th stage of the iteration process

J = the number of iterations.

Each of these methods requires filling and using the entire matrix to obtain a solution. This is computationally inefficient when dealing with relatively large matrices. Using a reactor size of 120 cm radius and 360 cm height and 0.5 cm node spacing requires a 171,841 x 171,841 matrix. This is a fairly large matrix and using any of the above techniques would increase the computation time and memory requirements of the computer.

To solve this system, I chose a blocked tridiagonal solver technique. This method requires only storing the values in the diagonals of the tridiagonal matrix of size $(m-1 \times n-1)$ as compared to $(m-1 \times n-1)^2$ for the Gauss-Seidel method. Using the previous 9x9 example, the system of equations in (43) becomes

$$\begin{bmatrix}
 A_1 & & & & & & & & & \\
 & D & & & & & & & & \\
 & & D & & & & & & & \\
 & & & D & & & & & & \\
 & & & & D & & & & & \\
 & & & & & D & & & & \\
 & & & & & & D & & & \\
 & & & & & & & D & & \\
 & & & & & & & & D & \\
 & & & & & & & & &
 \end{bmatrix}
 \begin{bmatrix}
 X_1 \\
 X_2 \\
 X_3 \\
 X_4 \\
 X_5 \\
 X_6 \\
 X_7 \\
 X_8 \\
 X_9 \\
 X_{10}
 \end{bmatrix}
 =
 \begin{bmatrix}
 B_1 \\
 B_2 \\
 B_3 \\
 B_4 \\
 B_5 \\
 B_6 \\
 B_7 \\
 B_8 \\
 B_9 \\
 B_{10}
 \end{bmatrix}
 \quad (45)$$

where D is the sub and super diagonal with constant coefficients, $A_{1,2,3}$ are tridiagonal matrices, $X_{1,2,3}$ are the unknown flux values at the interior mesh points, and $B_{1,2,3}$ are the solutions at each interior mesh point.

This can be further broken down into three equations that can each be solved using a standard tridiagonal solver.

$$\begin{aligned}
 A_1 X_1 + D X_2 &= B_1 \\
 D X_1 + A_2 X_2 + D X_3 &= B_2 \\
 D X_2 + A_3 X_3 &= B_3
 \end{aligned}
 \quad (46)$$

Inverting the D diagonal (which is constant) and rearranging the equations allows them to become

$$\begin{aligned}
 D^{-1}A_1X_1^{n+1} &= D^{-1}B_1 - X_2^n \\
 D^{-1}A_jX_j^{n+1} &= D^{-1}B_j - X_{j-1}^{n+1} - X_{j+1}^n \\
 D^{-1}A_{m-1}X_{m-1}^{n+1} &= D^{-1}B_{m-1} - X_{m-2}^{n+1}
 \end{aligned}
 \tag{47}$$

where j is the jth row along the z-axis. This allows one to solve the system only using the main tridiagonal components. To solve the system of equations in (47), set the initial values of X_j to zero and solve each equation in a tridiagonal solver making use of the previous X_j value. This iterative approach is similar to the Gauss-Seidel approach without the excess storage or computations. Once the values of X_j are within the set tolerance of the previous X_{j-1} , then the process has converged to the solution. This tolerance level is critical to achieving low relative errors between the old and the new flux values. For example, setting the tolerance equal to 0.001 provides a maximum relative error of 18 % for the axial power using a mesh spacing of one centimeter. Changing the tolerance to 1E-6, reduces the maximum relative error to 8 %. Reducing the tolerance does however increase the computational time dramatically. See Chapter III, Program Validation for further details.

Three Dimensional, Two Energy Group Model

Criticality Solution Technique

The solution technique is the same as the one group homogeneous method except one must guess an initial value for ϕ_1 and ϕ_2 to solve for ϕ_1 in equation (8). ϕ_1 is then used to solve for ϕ_2 in equation (9). The code iterates as in the one group model solving for

$S^{(n+1)}$ and $k^{(n+1)}$ per equations (16) and (19) and then checks for convergence per equation (21). This is shown in Figure 10.

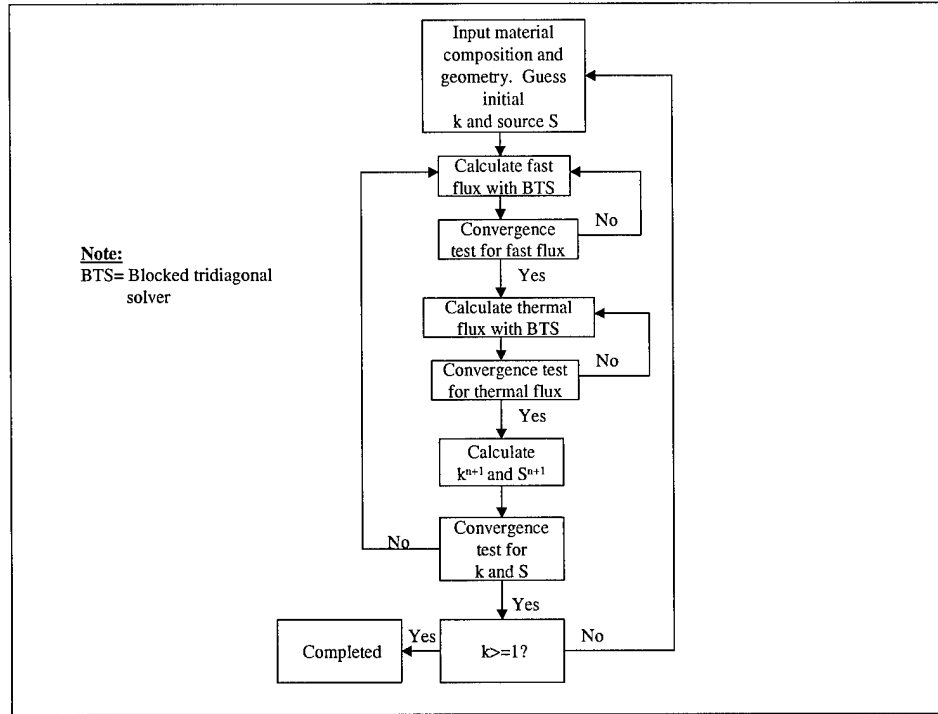


Figure 10 Criticality Solution Technique for Two Energy Groups

There is a difference in calculating the perturbation of the cross sections. The method is not as simple as in the one energy case because the multi-group criticality problem is not self-adjoint. I did not include the perturbation because of the complexity of having to change multiple parameters in both energy groups. The method is outlined in Duderstadt and Hamilton.

Numerical Development.

The development of the numerical solution technique is very similar to the three dimensional one group technique; however, there are now two coupled equations to

solve. The first equation, derived from applying central differences to equation (8), is shown in equation (48).

$$\begin{aligned} & \left(\frac{1}{\Delta r^2} - \frac{1}{2r\Delta r} \right) \phi_{i-1,j} + \left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta z^2} - \frac{\Sigma_{R1}}{D_1} \right) \phi_{i,j} + \left(\frac{1}{\Delta r^2} + \frac{1}{2r\Delta r} \right) \phi_{i+1,j} \\ & + \frac{1}{\Delta z^2} (\phi_{i,j-1} + \phi_{i,j+1}) = \frac{-\chi_1 S}{kD_1} \end{aligned} \quad (48)$$

where

$$\begin{aligned} S &= (v_1 \Sigma_{f1} \phi_1 + v_2 \Sigma_{f2} \phi_2) \\ \chi_1 &= 1. \end{aligned}$$

The program solves this equation for the fast flux, ϕ_1 , and then solves the second coupled equation for the thermal flux, ϕ_2 using the blocked tridiagonal solver. The second coupled equation, derived by applying central differences to equation (9), is

$$\begin{aligned} & \left(\frac{1}{\Delta r^2} - \frac{1}{2r\Delta r} \right) \phi_{i-1,j} + \left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta z^2} - \frac{\Sigma_{R2}}{D_2} \right) \phi_{i,j} + \left(\frac{1}{\Delta r^2} + \frac{1}{2r\Delta r} \right) \phi_{i+1,j} \\ & + \frac{1}{\Delta z^2} (\phi_{i,j-1} + \phi_{i,j+1}) = \frac{-\chi_2 S - \Sigma_{S12} \phi_1}{kD_2} \end{aligned} \quad (49)$$

where

$$\chi_2 = 0.$$

III. Program Development and Validation

Program Development

In general, the program provides much flexibility for the user. The user can choose between a two dimensional, one energy group model, a three dimensional one energy group model, or a three dimensional two energy group model. The program provides a drop down Windows menu to allow the user to select various command options such as models to run or help files to view. Within each model, the user can point the mouse over input boxes and get definitions or additional explanations. This serves to assist the user in understanding either the physics involved or how to proceed.

The program requires initial data input that represents a homogenized reactor core in cylindrical geometry. It requires the user to input the reactor core dimensions and material composition/cross sections. If the user is not sure of the material composition, the program provides recommended input values. The values for the macroscopic cross sections (absorption and fission), the diffusion coefficient, the number of interior mesh points, and the initial guess values for $k_{\text{effective}}$ and flux are required for input. The program will only allow for equal mesh spacing in both the axial and radial directions and requires the core height and radius be multiples of mesh spacing. The boundary conditions are established within the program. For all exterior points the flux is assumed to be zero. At the center of the reactor core, the flux is set equal to the flux value at the first interior mesh point away from the centerline. This makes use of the symmetry of the homogeneous core and meets the requirement that the first derivative of the flux equals zero at the center.

Given the initial input, the program will calculate the initial source term value ($\nu \Sigma_f \phi^0$) for the right hand side of the diffusion equation at each interior mesh point. If the user requested to solve the two dimensional simulation, the program calls a linear finite difference solver that uses the Crout factorization method for tridiagonal linear systems. The solver returns a vector of flux values along the radial interior mesh points. Next, the program begins to iterate to converge to the solution. It evaluates the tolerance between the old and the new flux values and the old and new $k_{\text{effective}}$ values. To calculate the new $k_{\text{effective}}$, the program integrates the old and new source terms over space and essentially averages them as in equation (19). With the updated source and $k_{\text{effective}}$, the program iterates again. This process continues until convergence is met. Next, the program checks if the reactor is critical. If $k_{\text{effective}}$ is greater than or equal to one, the system is critical. If the reactor is not critical, the program uses perturbation theory to provide a revised macroscopic cross section that should assist the user in achieving a critical reactor. In either case, the data are automatically loaded onto an Excel spreadsheet and plotted in Excel. The Excel chart is automatically updated onto the Visual BASIC form and saved to a location provided by the user. The user of the program does not see Excel running in the background.

For the three dimensional one energy group problem, the user has the option of selecting to calculate the power profile for either a half or a quarter of the reactor core. Because of the symmetry of the core geometry, the solution to the half of the reactor core is a mirror image of the quarter core solution. The iterative processes are very similar to the two dimensional problem; however, the solver is different. Because the numerical analysis problem generates a blocked tridiagonal system, the solution technique changes.

The program builds blocked tridiagonal vectors that create a $(m-1)$ system of equations. It fills each diagonal component with its corresponding coefficient and then iterates through the Crout factorization method for each tridiagonal linear system until the desired convergence is met for the iterative solution. Next, the program updates $k_{\text{effective}}$ and tests for convergence of the old and new source terms similar to the two dimensional problem. If the reactor is not critical, the program will again recommend an adjusted value for the macroscopic cross section and plot both the radial and axial power values in Excel.

The three dimensional, two energy group model uses the same process as the three dimensional, one energy group model with a few modifications. In addition to the previous input requirements, the user must provide the macroscopic scatter and removal cross sections for the fast and thermal energy ranges as well as the fast and thermal flux initial guesses. Using this data, the program solves for the fast flux using the blocked tridiagonal solver and then it uses that value to solve for the thermal flux, again using the blocked tridiagonal solver. The program iterates as before until convergence is met for the total source term values and k .

The user can double click each Excel chart on the form and open Excel to access the chart or the data. A complete program along with typical output data is included in Appendices C, D, E, and F.

Operating the Code

Each window in the program is a form that allows the program user several options. There is typically a drop down menu window structured as shown in Figure 11.

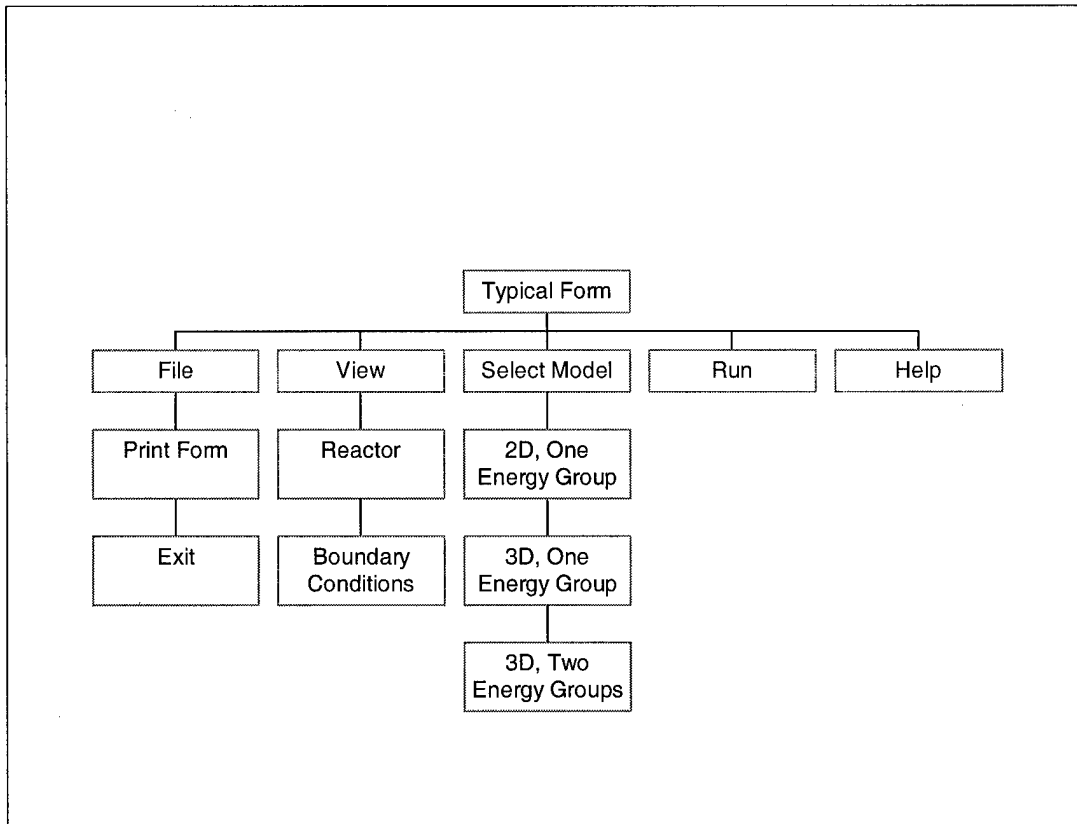


Figure 11 Typical Drop Down File Menu

The initial welcome form is shown in Figure 12 below.

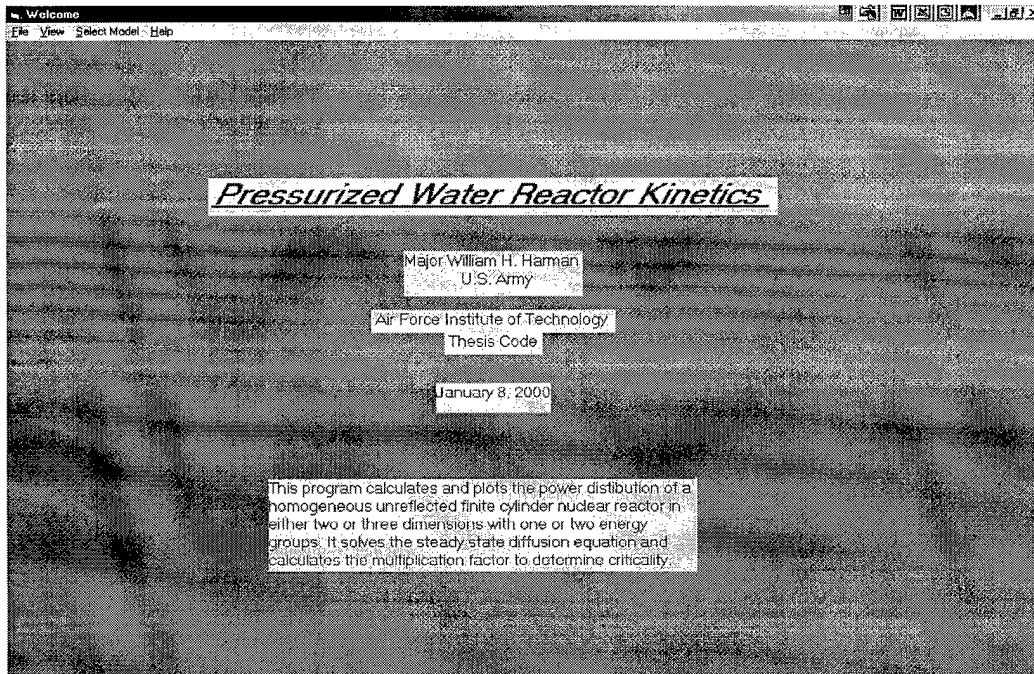


Figure 12 Welcome Screen

The PrintForm menu function prints the current form to the default printer. The Exit menu function exits the program. Clicking the View menu function provides two options. The program user can either view a schematic of the reactor or the boundary conditions as shown in Figure 13 and Figure 14 respectively. From either of them the user can return to the previous screen by clicking the return menu. To choose one of the three models to run, click the Select Model menu and choose the desired model. From the chosen model, the user can return to the starting window or select from any of the options shown. Clicking the run menu function will execute the selected module. The help menu function will bring up a window with a help object written in Word. To view the help information, double click the object window and scroll through the file. To close the file and return to the previous screen, click the "x" on the windows screen.

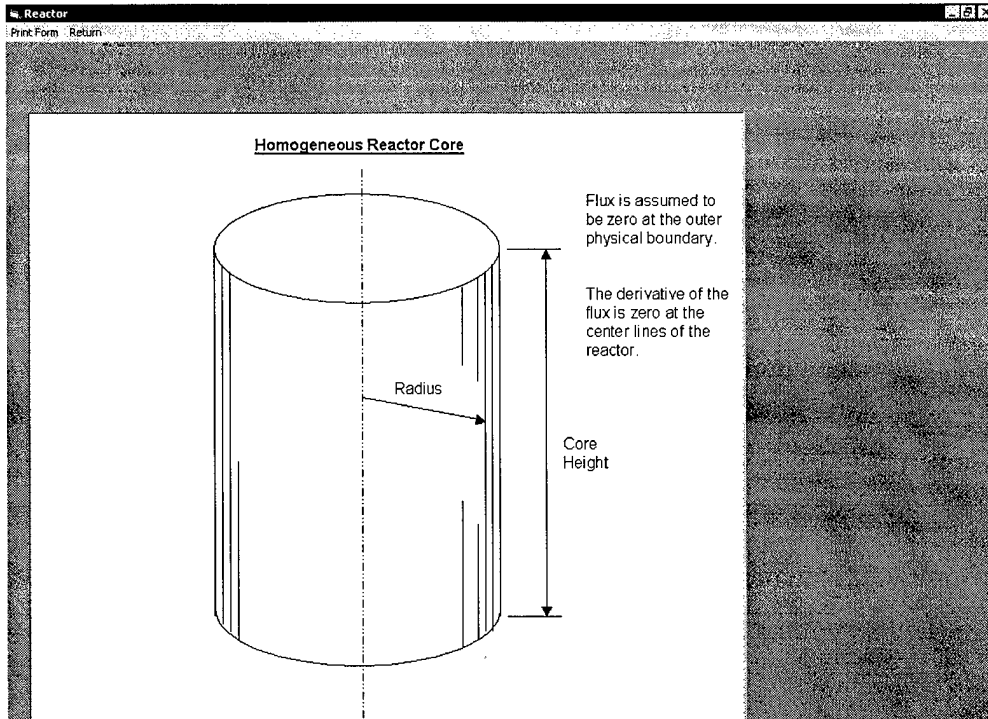


Figure 13 Reactor Schematic Form

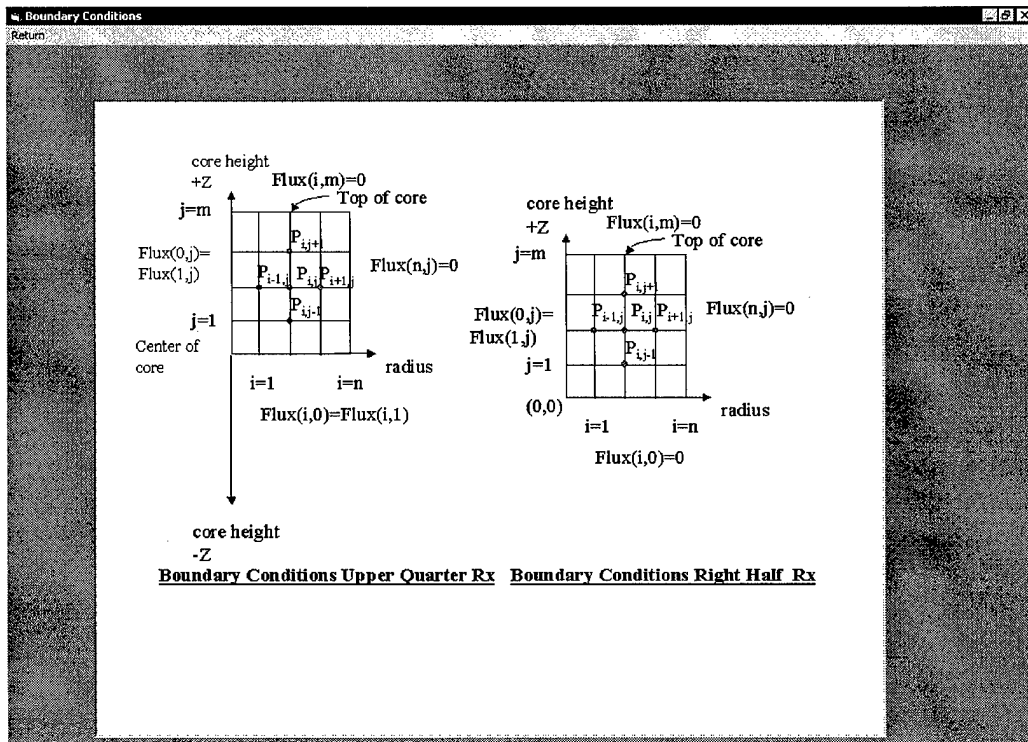


Figure 14 Boundary Condition Form

The program has several built in error commands that prevent the program from crashing should the user fail to input required data or input incorrect data. For example, if the user fails to input $k_{effective}$, the program will anticipate and prevent the division by zero in the code. The code will display a message explaining the error and insert $k_{effective} = 0.9$. While the code is certainly not yet totally failsafe, it provides error corrections to many anticipated runtime errors.

To run the two dimensional one energy group model, the code requires input for Σ_a , $v\Sigma_f$, ϕ_{guess} , diffusion coefficient, radius, and k_{guess} . All appropriate data must include units of centimeters. Each of the three reactor model forms provides tips for the user when the cursor is placed over some of the input description boxes. The user must also input the mesh spacing between mesh points. For typical reactors a mesh spacing of 0.5 cm provides a maximum relative error of less than two percent and runs within a few seconds. Finally, the user must input a file name and location to save the Excel workbook with an extension of filename.xls.

Figure 15 shows an example of the two dimensional model form. The form lets the user know if the reactor is critical and if not provides a recommended Σ_a that will drive the reactor to criticality.

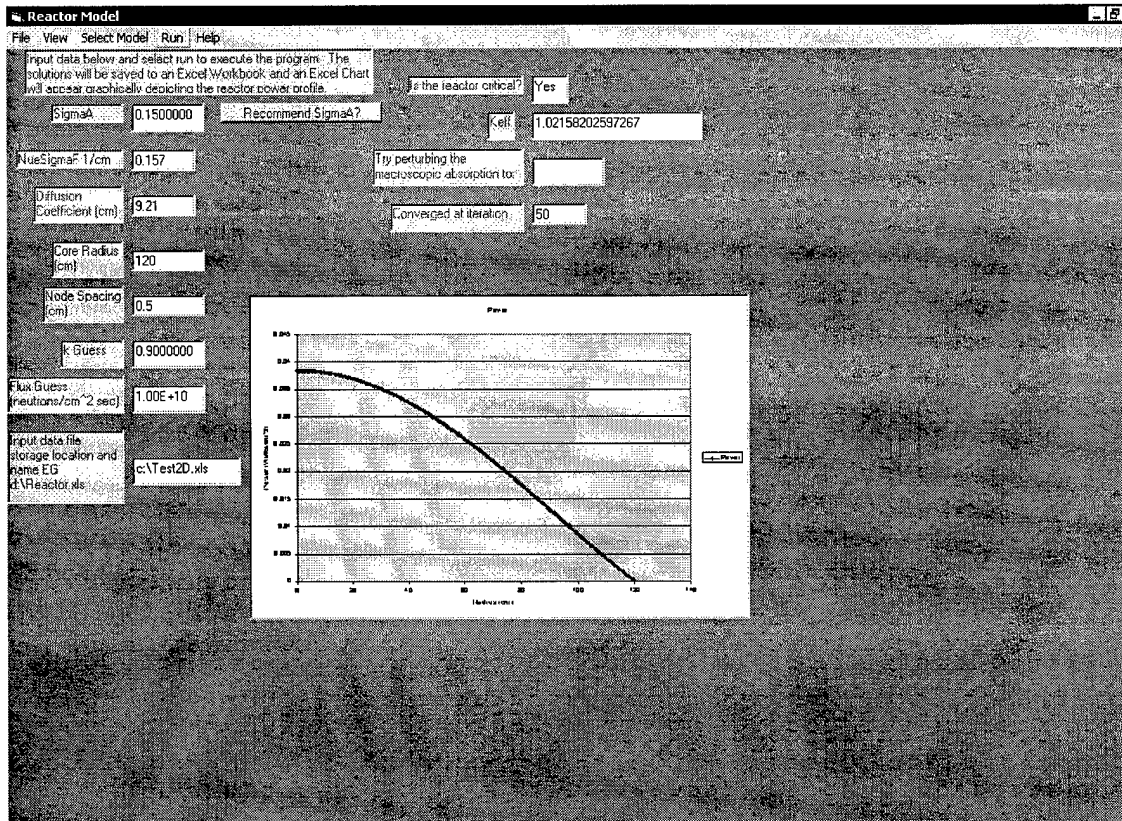


Figure 15 2D, One Energy Group Form

It also provides a power distribution plot. To access the plot and data, double click the chart and Excel will open the workbook containing the data and chart. Closing Excel returns the user to the form; however, the chart will not be resized to fit the screen. The chart will automatically resize upon running another problem.

To run the three dimensional one energy group model, input the same information required for the two dimensional model along with the reactor height. The program requires the user to select whether to calculate the power distribution for either a half or a quarter of the core. The mesh spacing will be equally applied to the axial and radial directions. Additionally, there is an option to choose between reduced relative error; slower run times and average relative error; faster run times. These correspond to the

results provided in Table 5, Table 6, and Table 10. Figure 16 shows the results of running a sample problem using half of the core.

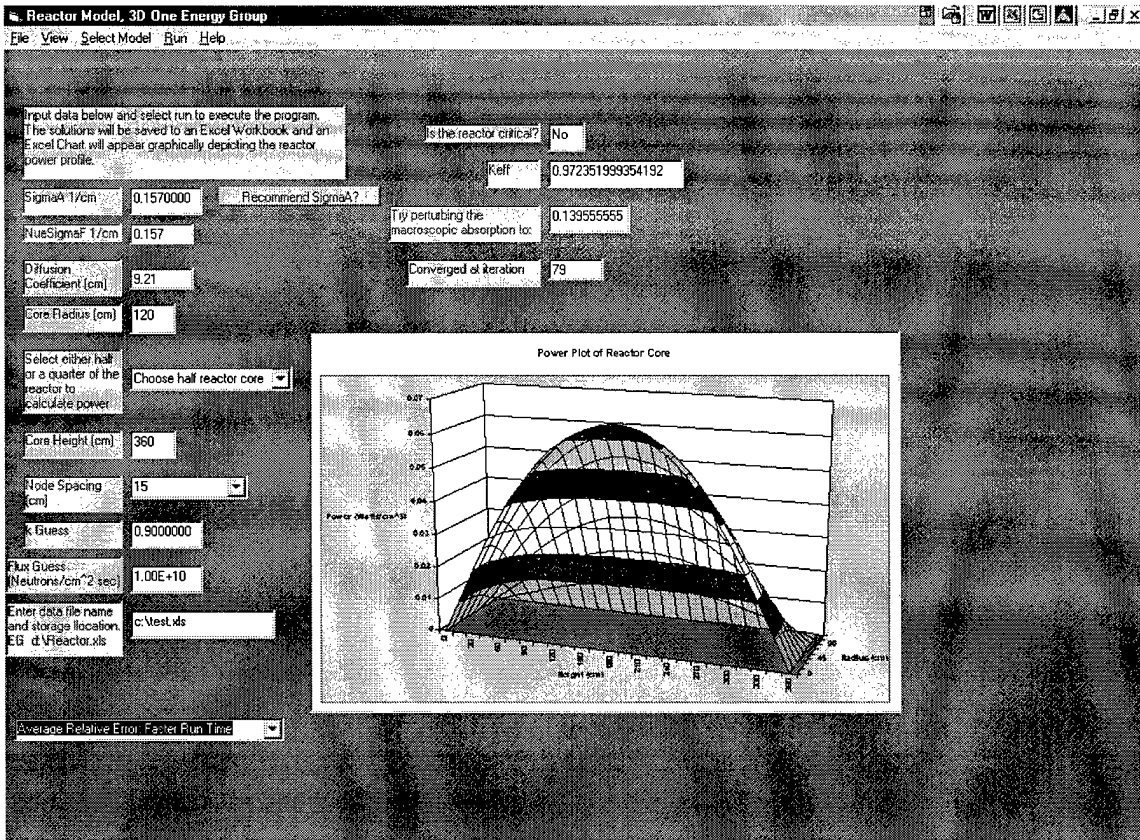


Figure 16 3D, One Energy Group Form

The reactor core height input corresponds to the height of either the half or quarter reactor core selection. Again, the program will provide the criticality information.

The three dimensional two energy group model requires more information than the previous models. The input data for both the thermal and fast energy groups must also include Σ_{Removal} and Σ_{scatter} . Table 2 provides sample input data for two energy groups.

Additionally, the user must select the radial and axial positions to plot the power distribution. Figure 17 displays the input box to select the radial position upon which to plot the axial power distribution. The maximum axial power will occur at the zero

position for either the half or quarter core selection. For the radial power distribution, the maximum will occur at the zero axial position for the quarter core or center axial position for the half core. The plots are independent of one another allowing the user to select the power distribution along any section of the core.

Like the three dimensional one energy group model, the user must choose between reduced relative error; slower run times and average relative error; faster run times.

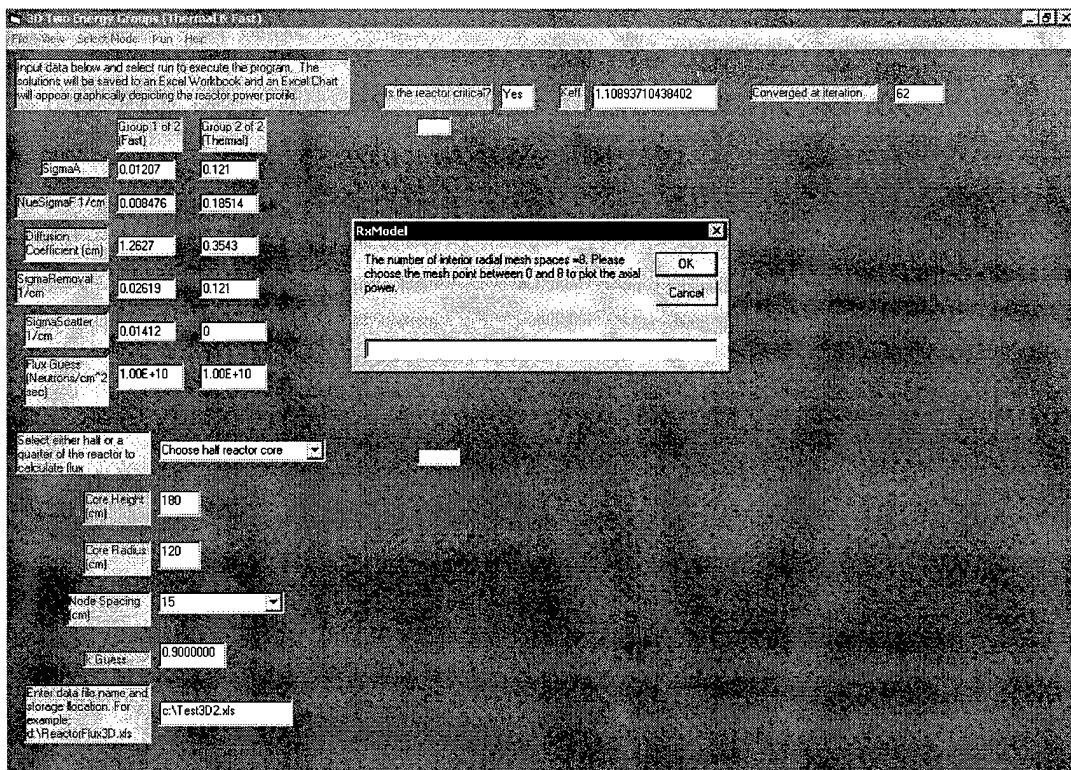


Figure 17 3D, Two Energy Groups Axial Choice

Figure 18 is an example of the final solution. The plots include the thermal, fast, and total power distributions. As in the other models, the program user can access the Excel workbook directly by double clicking either chart. The workbook will contain the power distribution data for the radial and axial plots on separate worksheets.

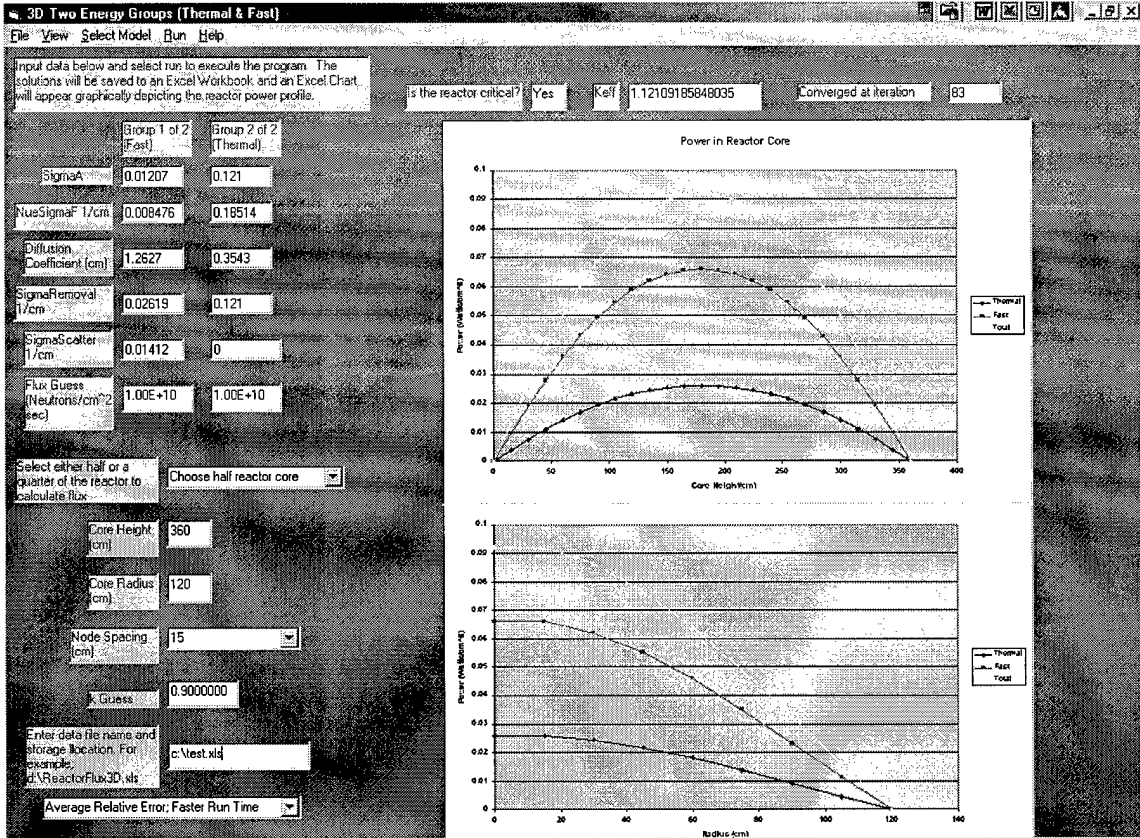


Figure 18 3D, Two Energy Group Form

Program Validation

To validate the code, I tested each of the three models independently. I compared the code's normalized power distribution to the normalized analytical solution to determine the point-by-point relative error. For the three dimensional models, I tested and validated the blocked tridiagonal solver using various matrices solved with Mathematica before testing the entire code.

Two Dimensional, One Energy Group Model.

I tested the model using data for a typical homogeneous reactor as shown in Table 1.

Table 1 One-Speed Reactor Input Data

	Reactor Data
Σ_a	0.1532 1/cm
$\nu\Sigma_f$	0.1570 1/cm
Diffusion Coefficient	9.21 cm
Σ_{tr}	0.0362 1/cm

I compared the model to the flux distribution profile for an infinite right circular cylinder.

$$\phi = J_0\left(\frac{\nu_0 r}{R}\right) \quad (50)$$

where

$\nu_0 = 2.405 =$ smallest zero of J_0

R is the radius of the cylinder

r is the position along the radius.

Figure 19 shows the normalized flux profile for the reactor with a radius of 120 cm. Figure 20 shows the normalized flux profile for the data from the one-group model. Figure 21 is a combination of both. They overlap each other indicating that the one-speed model is producing the correct fundamental flux mode shape. Using a radius of 120 cm and a mesh spacing of 0.5 cm, the maximum relative point-by-point error was 0.02.

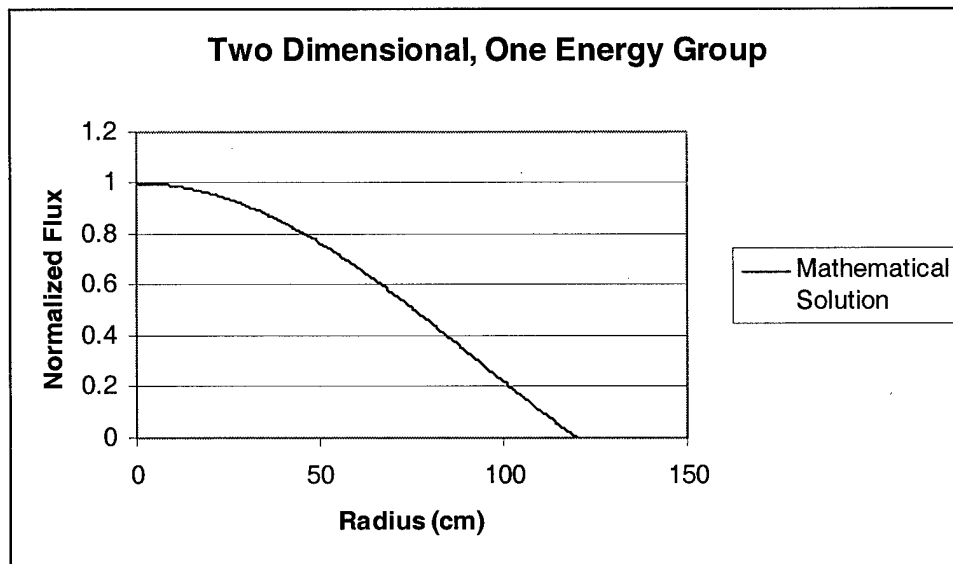


Figure 19 Flux profile of infinite right circular cylinder (Bessel J function)

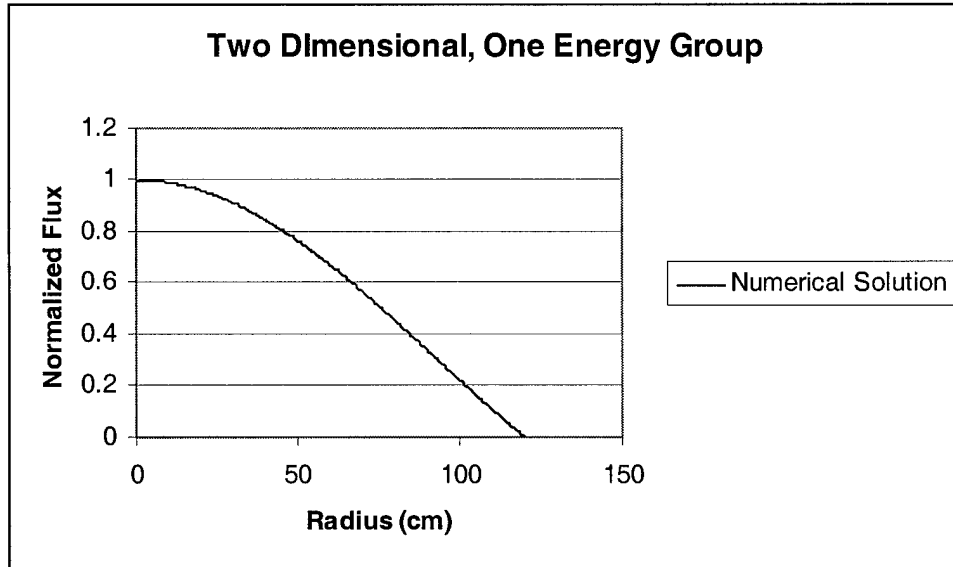


Figure 20 Normalized flux from one-speed computer model

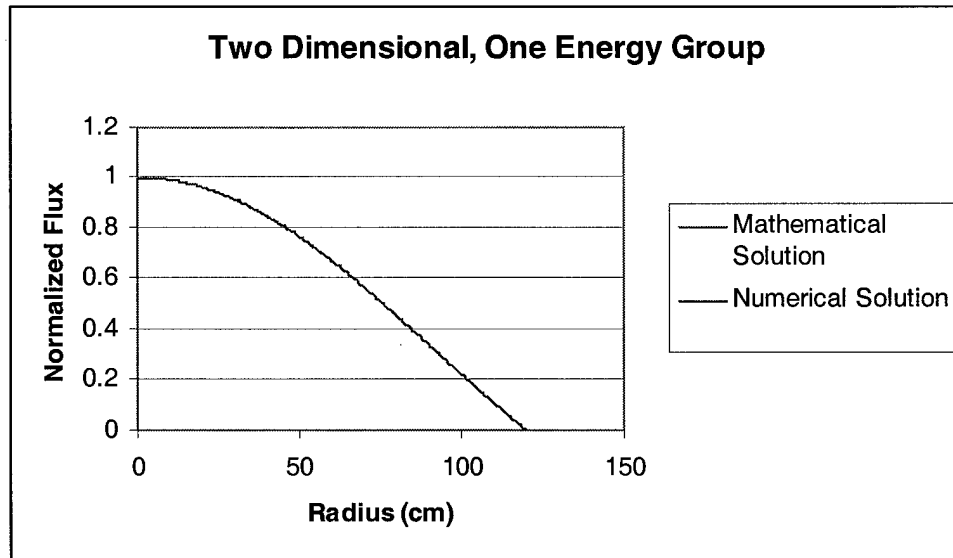


Figure 21 Flux profile of Bessel J function and one-speed computer model together

Three Dimensional, One and Two Energy Group Models.

To test the two models, I verified the blocked tridiagonal solver and then each code separately. I initially tested the blocked tridiagonal solver using Mathematica with a diagonally dominant system of equations as shown in the augmented matrix (51).

$$\begin{bmatrix} -5.16 & 1.25 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1.894 \\ 0.5 & -5.16 & 1.167 & 0 & 1 & 0 & 0 & 0 & 0 & -1.894 \\ 0 & 0.75 & -5.16 & 0 & 0 & 1 & 0 & 0 & 0 & -1.894 \\ 1 & 0 & 0 & -5.16 & 1.25 & 0 & 1 & 0 & 0 & -1.894 \\ 0 & 1 & 0 & 0.5 & -5.16 & 1.167 & 0 & 1 & 0 & -1.894 \\ 0 & 0 & 1 & 0 & 0.75 & -5.16 & 0 & 0 & 1 & -1.894 \\ 0 & 0 & 0 & 1 & 0 & 0 & -5.16 & 1.25 & 0 & -1.894 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0.5 & -5.16 & 1.167 & -1.894 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.75 & -5.16 & -1.894 \end{bmatrix} \quad (51)$$

The approximate solution is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.678681 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.678681 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0.678681 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0.678681 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0.678681 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.678681 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0.678681 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.678681 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.678681 \end{bmatrix} \quad (52)$$

In this case, the system resulted in a nine by nine matrix where $m=4$ and $n=4$. The pattern and coefficients are similar to the actual pattern and coefficients generated by numerically solving the diffusion equation using the one group data. The solution generated by Mathematica matched the solution given by the blocked tridiagonal solver. The blocked tridiagonal solver typically converges to the approximate solution within 10-

15 iterations with a tolerance of 1E-6, using mesh spaces greater than 10 centimeters.

Their agreement indicates that the solver is converging to the correct solution.

Varying the values, constant and non-constant coefficients, of the lower and upper diagonals while maintaining diagonal dominance resulted in correct solutions as well.

This was one of the several comparisons I made, each of which agreed with the blocked tridiagonal solver's solution.

Next, I verified that the codes produced the approximate correct power distribution within the reactor core. The compiled program was tested using an AMD-K3 450 megahertz personal computer with 64 megabytes of random access memory. I compared the relative maximum error of the normalized numerical solutions at each mesh point to the normalized mathematical solution for a finite cylinder.

$$\phi(r, z) = J_0\left(\frac{2.405r}{R}\right) \text{Cos}\left(\frac{\pi z}{H}\right) \quad (53)$$

where

H = the height of the cylinder

z = the position along the z axis (height)

For the three dimensional one group model, I used the same input data used in the two dimensional one group model. I used the data in Table 2 as input for the three dimensional two energy group model (Duderstadt and Hamilton, 1976:312).

Table 2 Two Group Diffusion Theory Constants (Table 7-2 Duderstadt)

Group Constant	Group 1 Fast	Group 2 Thermal
Σ_a (1/cm)	0.01207	0.1210
$v\Sigma_f$ (1/cm)	0.008476	0.18514
Σ_f (1/cm)	0.003320	0.07537
Diffusion Coefficient (cm)	1.2627	0.3543
SigmaRemoval (1/cm)	0.02619	0.1210
SigmaScatter (1/cm)	0.01412	0

I originally limited the blocked tridiagonal solver to 20 iterations because of the typical convergence within 10-15 iterations. However, this produced erroneous results as the mesh spacing was reduced to less than six centimeters. The blocked tridiagonal solver failed to completely converge after 20 iterations causing the large relative errors shown in Table 3 and Table 4. Notice the relative error converges and then begins to diverge below mesh spacings of around 4-6 centimeters. Table 3 and Table 4 show the relative errors and approximate code running times for several test runs for half and quarter of the core test runs.

Table 3 Relative Errors for Quarter Core 3D Models, Blocked Tridiagonal Tolerance =1E-8 with Maximum of 20 Iterations

Mesh Spacing Centimeters	One Energy Group			Two Energy Groups		
	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
30	0.1	0.1	00:08	0.1	0.1	00:16
20	0.07	0.06	00:13	0.07	0.06	00:21
15	0.04	0.05	00:21	0.04	0.05	00:32
12	0.03	0.04	00:36	0.03	0.04	00:51
10	0.02	0.03	00:52	0.02	0.03	01:21
6	0.01	0.01	02:20	0.01	0.02	03:52
5	0.008	0.03	03:20	0.008	0.005	05:52
4	0.006	0.2	05:27	0.006	0.1	09:54
3	0.005	0.7	Untimed	0.005	0.5	Untimed

Table 4 Relative Errors for Half Core 3D Models, Blocked Tridiagonal Tolerance =1E-8 with Maximum of 20 Iterations

Mesh Spacing Centimeters	One Energy Group			Two Energy Groups (Total Power)		
	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
30	0.1	0.0001	00:11	0.1	0.0001	00:19
20	0.07	0.00009	00:21	0.07	0.0001	00:34
15	0.04	0.00009	00:39	0.04	0.0001	00:59
12	0.03	0.00009	01:12	0.03	0.0001	01:49
10	0.02	0.00009	01:42	0.02	0.0001	02:45
6	0.01	0.004	04:41	0.01	0.0009	08:03
5	0.008	0.03	06:41	0.008	0.01	11:57
4	0.006	0.2	07:52	0.006	0.6	19:04
3	0.005	1.0	Untimed	0.005	0.4	21:37

I initially thought the divergence was due to instability of the finite central difference method; however, Figure 22 shows that the relative error was not symmetrical.

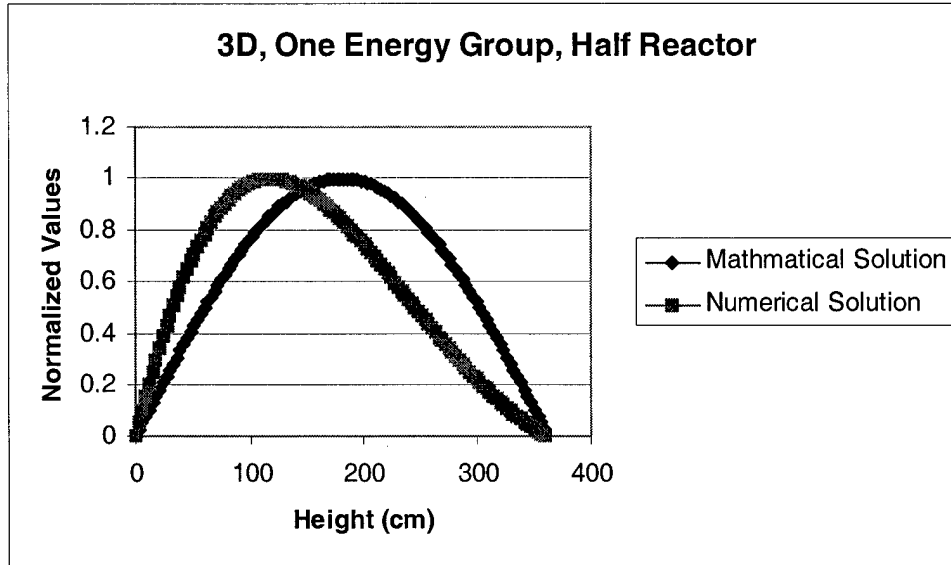


Figure 22 Initial Test, Mesh Spacing = 3 cm

This suggested that the error was not due to instability. Increasing the maximum number of iterations in the blocked tridiagonal solver from 20 to 1000 allowed for complete convergence with smaller mesh sizes and corrected the error.

The tolerances set for the convergence of the multiplication factor k and the blocked tridiagonal solver are critical to achieving useful solutions. For most cases, a k tolerance of $1E-5$ provides acceptable results (Ott, 1989:351). Table 5 and Table 6 provide a summary of the results using a tolerance of 0.001 for the blocked tridiagonal solver and a k tolerance of $1E-5$. This tolerance setting provided maximum relative errors of less than six percent for a mesh spacing of two centimeters when analyzing a quarter of the reactor. Analyzing half the reactor core increased the maximum relative error and run times as expected. I chose this setting because it yielded reasonable results with fast run times.

Table 5 Relative Error for Quarter Core 3D, Blocked Tridagonal Tolerance = 0.001

Mesh Spacing Centimeters	One Energy Group			Two Energy Groups		
	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
30	0.1	0.2	00:06	0.1	0.2	00:09
20	0.07	0.1	00:07	0.07	0.2	00:09
15	0.04	0.1	00:07	0.04	0.1	00:10
12	0.03	0.1	00:09	0.03	0.1	00:11
10	0.02	0.09	00:11	0.02	0.1	00:13
6	0.01	0.07	00:20	0.01	0.1	00:33
5	0.009	0.06	00:49	0.009	0.09	00:52
4	0.007	0.04	01:32	0.008	0.08	01:36
3	0.006	0.006	03:43	0.007	0.04	03:47
2	0.006	0.05	13:56	0.007	0.02	14:13
1	0.01	0.2	≅180:00	No test	No test	No test

Table 6 Relative Errors for Half Core 3D, Blocked Tridiagonal Tolerance = 0.001

Mesh Spacing Centimeters	One Energy Group			Two Energy Groups		
	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
30	0.1	0.09	00:07	0.1	0.1	00:09
20	0.07	0.09	00:07	0.07	0.1	00:11
15	0.04	0.08	00:10	0.04	0.1	00:13
12	0.03	0.08	00:15	0.03	0.1	00:16
10	0.02	0.09	00:18	0.02	0.1	00:22
6	0.01	0.1	00:58	0.01	0.1	01:01
5	0.008	0.1	01:41	0.009	0.1	01:39
4	0.007	0.1	03:02	0.008	0.1	03:09
3	0.006	0.1	07:20	0.007	0.2	07:54
2	0.006	0.2	27:45	0.007	0.2	29:26
1	No test	No test	No test	0.01	0.3	Untimed

What impact does reducing the tolerance of the blocked tridiagonal solver have on the maximum relative error? Table 7 and Table 8 show the results of changing the tolerance

to 1E-4 and 1E-6 respectively for half of the core using two energy groups. There was not a significant reduction in the maximum relative error by changing the tolerance to 1E-4. Reducing the tolerance to 1E-6 did not significantly reduce the error in the radial direction; however, the error was reduced by over ten percent in the axial direction. The trade off is doubling the run time from approximately 29 minutes as shown in Table 6 to 60 minutes as shown in Table 8.

Table 7 Half Core 3D, Blocked tridiagonal Tolerance = 1E-4

Two Energy Groups			
Mesh Spacing Centimeters	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
2	0.006	0.2	33:42

Table 8 Half Core 3D, Blocked Tridiagonal Tolerance = 1E-6

Two Energy Groups			
Mesh Spacing Centimeters	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
2	0.006	0.08	60:05

Reducing the tolerance to 1E-8 provided even better results as shown in Table 9. This tolerance reduced the maximum axial error as shown in Table 6 by approximately 19 percent for the mesh spacing of one centimeter.

Table 9 Half Core 3D, Blocked Tridiagonal Tolerance = 1E-8

Two Energy Groups			
Mesh Spacing Centimeters	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
1	0.01	0.1	Several Hours

Figure 23 is the normalized power distribution plot compared to the normalized mathematical solution for the reduced blocked tridiagonal tolerance of $1E-8$. Although the maximum error is about 10 percent, the error is symmetric about the center of the reactor core.

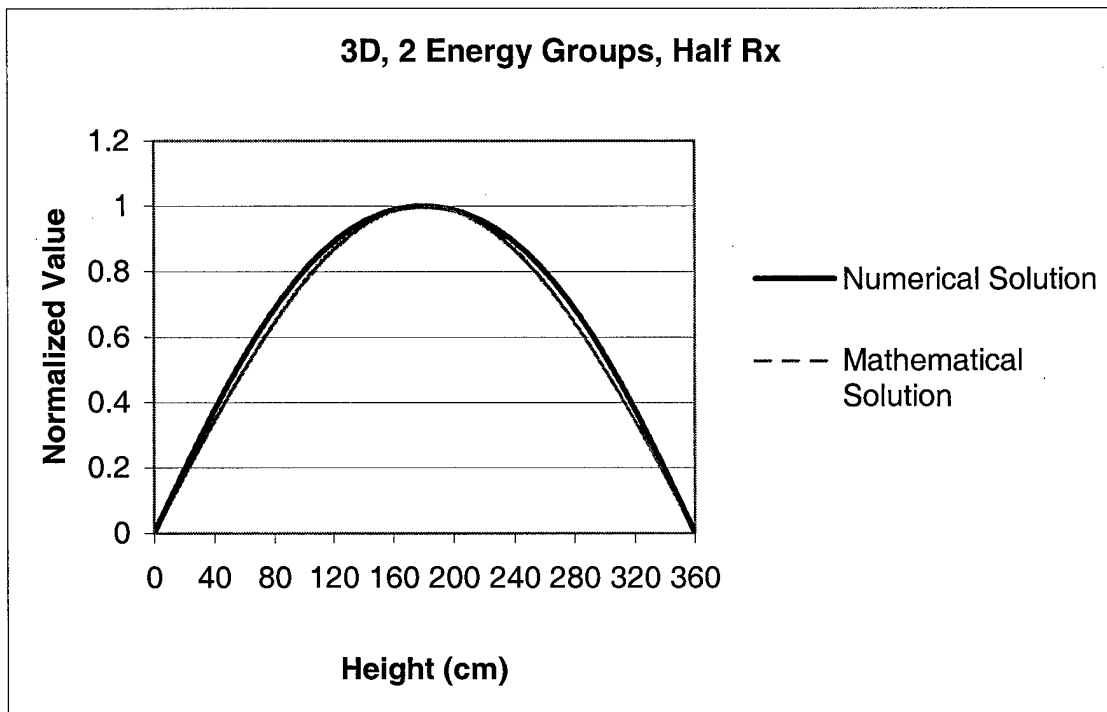


Figure 23 3D, Total 2 Energy Groups, Blocked Tridiagonal Tolerance = $1E-8$ with Mesh Spacing = 1 cm

To reduce the maximum relative error even more, I set the tolerances for convergence of the multiplication factor k and the blocked tridiagonal solver to $1E-7$ and $1E-8$ respectively. Table 10 is a summary of the maximum relative errors for test runs calculating the power distribution for a half reactor core, using two energy groups. The data indicates that the radial maximum relative error continues to reduce while the axial error remains constant at about 0.001. As before, the running times increase significantly as the mesh spacing reduces to one centimeter.

Table 10 Half Core 3D, k Tolerance =1E-7, Blocked Tridiagonal Tolerance = 1E-8

Two Energy Groups			
Mesh Spacing Centimeters	Radial Maximum Relative Error	Axial Maximum Relative Error	Running Time (Min:Sec)
30	0.1	0.001	00:10
20	0.07	0.001	00:15
15	0.04	0.001	00:25
12	0.03	0.001	00:40
10	0.02	0.001	01:04
6	0.009	0.001	04:29
5	0.007	0.001	07:55
4	0.006	0.001	16:59
3	0.005	0.001	47:30

Appendix G contains plots comparing the relative errors as indicated in Table 5 and Table 6 using a blocked tridiagonal tolerance of 0.001 and a k tolerance of 1E-5.

The final code provides the user with the options as shown in Table 11 below. This provides the user with the flexibility to choose between a level of maximum relative errors and run times corresponding to Table 5, Table 6, and Table 10.

Table 11 Options for Three Dimensional Models

	Tolerance for k	Tolerance for blocked Tridiagonal solver
Reduced relative error, Increased run time	1E-7	1E-8
Average relative error, Faster run time	1E-5	0.001

IV. Conclusions and Recommendations

Conclusions

The model provides the power distribution of a homogeneous unreflected reactor core in two or three dimensions using either one or two energy groups for a steady state reactor. Teachers can use the program to augment fundamental nuclear reactor courses by providing students with an additional resource to enhance learning. The program allows the user to modify the reactor dimensions and/or core composition and see the impacts on the power distribution and criticality within the reactor core.

Reducing the allowed acceptable tolerance for convergence in the blocked tridiagonal solver and the multiplication factor will reduce the normalized maximum relative error for the three dimensional models. The major trade off is increasing the computational time. Using $1E-5$ and 0.01 as the convergence tolerances for the multiplication factor and blocked tridiagonal solver respectively, the model yields a power distribution with a maximum relative error of about four percent for a mesh spacing of three centimeters, using two energy groups for a quarter core calculation. Using $1E-7$ and $1E-8$ respectively, the model yielded a maximum relative error of about one half of one percent for the half core calculations. To achieve such a low relative error, the running times increased from about four minutes to 48 minutes. Because this is a homogeneous system, one should take advantage of symmetry and calculate the power distribution in a quarter of the core.

The Visual BASIC 5.0 program is completely exportable to most Windows based personal computers. It automatically plots the power distribution, based on the core

dimensions and composition input, using Excel and links the chart to the Visual BASIC 5.0 form. Additionally, it calculates the multiplication factor to determine criticality.

Recommendations

The code is flexible enough to allow for future, user-friendly improvements while the finite central difference method and criticality search technique are the foundation for more complex reactor codes. The model can be the basis for adding a heterogeneous core and other modules including thermal-hydraulics, control adjustments, and depletion.

Although, the code has some built in error checks, it is not totally “crash proof”. Several additional error checks should be added as the program is used and tested by teachers and students alike. Additionally, an improved help file and automated read input statement should be added.

One approach to developing a heterogeneous model is to convert the unit cells of the lattice core to homogeneous cells as shown in Figure 24. Reactor cores are constructed of several material compositions including fuel rods, cladding, and coolant. Using the general assumption that the net neutron current flow across cell boundaries equals zero, one spatially averages the multigroup cross sections of the materials to obtain a group cross section for the unit cell. This is usually done for the fast and thermal group effects. Equation (54) defines the cell averaged group constant.

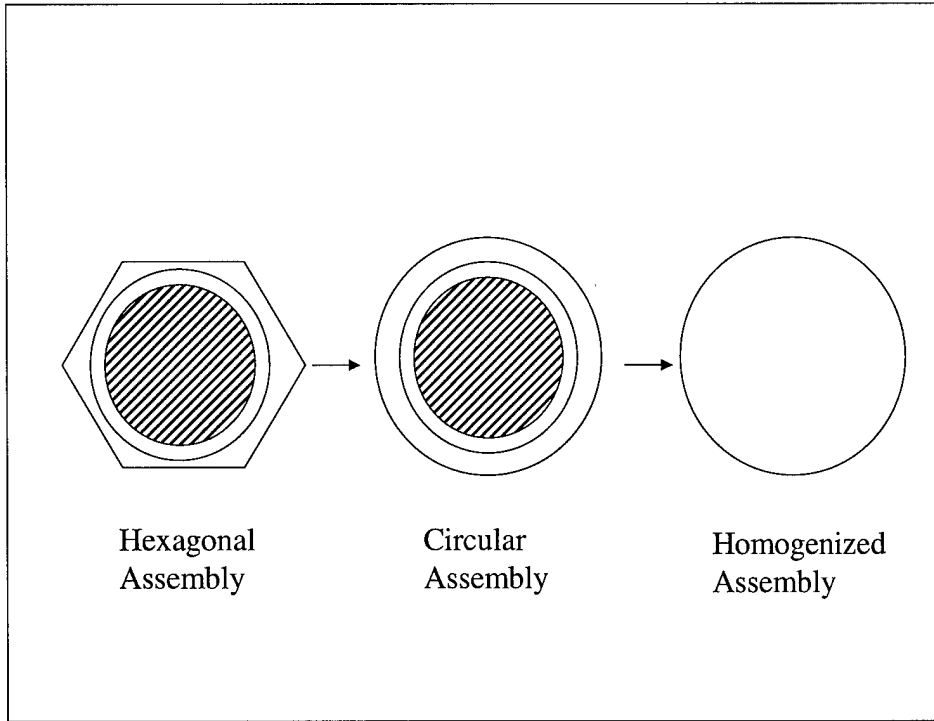


Figure 24 Fuel-Cell Homogenization

$$\langle \Sigma_g \rangle_{cell} = \frac{\int_{E_g}^{E_{g-1}} dE \int_{V_{cell}} \Sigma(r, E) \phi(r, E) d^3r}{\int_{E_g}^{E_{g-1}} dE \int_{V_{cell}} \phi(r, E) d^3r} \quad (54)$$

After homogenizing the unit cells, control rods can be added to the homogenized core. One can use the cell group constants along with the control rod cross sections in a multigroup two dimensional diffusion calculation. These revised flux values can then be used to calculate the final group constants for the homogenized fuel assembly. The final step is to calculate the flux and power levels in the homogenized core. This can be accomplished by dividing the core into equal lattice structures of squares or other

geometric shapes that take advantage of symmetry to reduce the computation requirements.

The addition of a thermal-hydraulic module would significantly enhance the model's capability. PWR power distributions are coupled to temperature. PWRs use water as a coolant that typically enters the bottom of the core and leaves near the top. The coolant decreases in density as it absorbs heat moving up through the core. The power density of the current model predicts a symmetric power peaking profile that is not truly the case. Because of the change in density, the axial power peak is actually slightly toward the bottom. The average fuel and moderator temperatures can then be used to adjust the macroscopic cross sections for use in the power distribution model.

Finally, a control adjustment and depletion module can be added. The control adjustment module would calculate the adjustments necessary for control rod insertion or withdrawal to maintain criticality and the depletion module would account for fuel burn up impacts on the reactor core.

Appendix A. Derivation of Three Dimensional Source Integration

Let

$$I^n = \int_0^{z_0} \int_0^{2\pi} \int_0^R \phi^n(r, z) r dr d\theta dz \quad (55)$$

for a given homogeneous material. Integrating ϕ yields

$$I^n = 2\pi \int_0^{z_0} \int_0^R \phi^n(r, z) r dr dz. \quad (56)$$

Let

$$I^n(r) = 2\pi \int_0^{z_0} \phi^n(r, z) dz. \quad (57)$$

Using the trapezoid rule to numerically solve the integration with respect to z , where the trapezoid rule is

$$\int_a^b f(x) \approx \frac{\Delta x}{2} \left(f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_i) \right) \quad (58)$$

and

$$x_i = i\Delta x, \quad i = 1, 2, 3, \dots, n$$

$$\Delta x = \frac{b}{n}$$

yields

$$I^n(r) \approx 2\pi \frac{\Delta z}{2} \left(\phi^n(r, z_0) + \phi^n(r, 0) + 2 \left(\sum_{j=1}^{m-1} \phi^n(r, z_j) \right) \right) \quad (59)$$

where

$\phi^n(r, z_0) = 0$ and $\phi^n(r, 0) = 0$ due to boundary conditions

$$z_j = j\Delta z$$

$$\Delta z = \frac{z_0}{m}$$

Integrating with respect to r , yields

$$I^n \approx \frac{\Delta r}{2} \left(I^n(0)0 + I^n(R)R + 2 \sum_{i=1}^{n-1} I^n(r_i)r_i \right) \quad (60)$$

where

$$r_i = i\Delta r$$
$$\Delta r = \frac{R}{n}$$

but from equation (57)

$$I^n(R) = 2\pi \int_0^{z_0} \phi^n(R, z) dz = 0. \quad (61)$$

Simplifying and combining equations (59) and (60) results in

$$I^n \approx 2\pi\Delta z(\Delta r)^2 \sum_{i=1}^{n-1} i \sum_{j=1}^{m-1} \phi^n(r_i, z_j) \quad (62)$$

Appendix B. Derivation of Right Circular Cylinder Reactor Core Solution

The basic equation is

$$\nabla^2 \phi - \frac{\Sigma_a}{D} \phi = \frac{-\nu \Sigma_f}{D} \phi \quad (63)$$

where the boundary conditions are

$$\begin{aligned} \phi(r, \pm \tilde{H}) &= 0 \\ \phi(\tilde{R}, z) &= 0. \end{aligned}$$

Converting this into right circular cylinder coordinates results in

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \phi}{\partial r} \right) + \left(\frac{\partial^2 \phi}{\partial z^2} \right) + B^2 \phi = 0 \quad (64)$$

where

$$B^2 = \frac{1}{D} (\nu \Sigma_f - \Sigma_a).$$

Separating the variables and letting

$$\phi(r, z) = R(r)Z(z)$$

yields

$$\frac{1}{r} \frac{\partial}{\partial r} (rR'(r)Z(z)) + R(r)Z''(z) + B^2 R(r)Z(z) = 0. \quad (65)$$

Collecting the terms provides

$$\frac{1}{r} \frac{(rR')'}{R(r)} + \frac{Z''(z)}{Z(z)} + B^2 = 0. \quad (66)$$

Now setting the equation equal to a constant provides

$$\frac{1}{r} \frac{(rR')'}{R(r)} + B^2 = \frac{-Z''(z)}{Z(z)} = \lambda^2. \quad (67)$$

Solving

$$Z''(z) + \lambda^2 Z(z) = 0 \quad (68)$$

where

$$Z\left(\pm \frac{H}{2}\right) = 0$$

yields

$$Z_n(z) = \text{Cos}\left(\frac{n\pi z}{\tilde{H}}\right) \quad (69)$$

where

$$\lambda^2 = \left(\frac{n\pi}{\tilde{H}}\right)^2 \quad n = 1, 3, \dots$$

Now solving the second equation

$$\frac{1}{r} \frac{(rR')'}{R(r)} + B^2 = \lambda^2 \quad (70)$$

where

$$R(\tilde{R}) = 0$$

$$\frac{1}{r} (rR')' + \mu^2 R = 0 \quad (71)$$

and

$$\mu^2 = B^2 - \lambda^2 = \frac{1}{D} (v \Sigma_f - \Sigma a) - \left(\frac{\pi}{\tilde{H}}\right)^2.$$

The solution is in the form of the zeroth order Bessel functions.

$$R(r) = AJ_0(\mu r) + CY_0(\mu r) \quad (72)$$

However as $r \rightarrow 0$, $Y_0(\mu r) \rightarrow \infty$, therefore C must equal zero. At the boundary condition $r = \tilde{R}$,

$$R(\tilde{R}) = 0 = AJ_0(\mu \tilde{R}) \quad (73)$$

only if A does not equal zero and if $\mu \tilde{R} = \nu_n$, where ν_n is the zeros of the J_0 .

Therefore the eigenfunctions and eigenvalues are

$$R_n(r) = J_0\left(\frac{\nu_n r}{\tilde{R}}\right) \quad (74)$$

$$\mu_n^2 = \left(\frac{\nu_n}{\tilde{R}}\right)^2 \text{ for } n = 0, 1, 2, \dots$$

$$B^2 = \frac{1}{D}(\nu \Sigma_f - \Sigma a) = \left(\frac{\nu_0}{\tilde{R}}\right)^2 - \left(\frac{\pi}{2\tilde{H}}\right)^2. \quad (75)$$

Equation (75) represents the geometric buckling of the reactor. The general solution the problem then becomes

$$\phi(r, z) = AJ_0\left(\frac{\nu_0 r}{\tilde{R}}\right) \text{Cos}\left(\frac{\pi z}{\tilde{H}}\right) \quad (76)$$

where A is a normalization factor.

Appendix C. Two Dimension, One Energy Group Visual BASIC Code

'Thesis code by MAJ Will Harman

'This program calculates the radial flux/power profile for a typical right circular
'cylinder in two dimensions. It uses the standard diffusion equation in a one energy
group (one-speed)

'homogeneous unreflected reactor core. The equation is solved by using the finite central
'difference technique. The scheme uses a power iterative technique to solve for
'flux based upon an initial guess of k effective and the flux. It finds the eigenfunction
'for the maximum eigenvalue providing the fundamental mode shape for flux.

```
Const mnErrDivByZero = 11, mnErrOverflow = 6
```

```
Const mnErrBadCall = 5
```

```
Private Sub Form_Load()
```

```
'Load initial values from Duderstadt page 210-211
```

```
Dim kGuess, FluxGuess, SigmaA, NueSigmaF As Double
```

```
Dim DiffusionCoefficient, Radius, h As Double
```

```
Text1 = ""
```

```
Text2 = 0.157 'NueSigmaF 1/cm
```

```
Text3 = 9.21 'DiffusionCoefficient cm
```

```
Text4 = 120 'Radius cm
```

```
Text5 = 0.5 'mesh spacing
```

```
Text6 = 0.9 'K guess
```

```
Text8 = "" 'Critical Rx?
```

```
Text9 = "" 'Keffective will be calculated
```

```
Text10 = " "
```

```
Text11 = ""
```

```
Text14 = ""
```

```
End Sub
```

```
Sub Kinetics()
```

```
Dim prompt 'ask user for input
```

```
Dim Valu(2) As Double
```

```
Dim kGuess As Double
```

```
Dim SigmaA As Double
```

```
'Set source document for Excel chart to name and location by user
```

```
OLE2.SourceDoc = ("Text1")
```

```
OLE2.Visible = True
```

```
'Check for numerical entries
```

```
ok = 0
```

```
For j = 0 To 1
```

```
  ykk = Checkin(MaskedTextBox(j))
```

```
  ok = ok + ykk
```

```
Next j
```

```
If ok > 0 Then
```

```

MsgBox ("You must enter positive numbers")
End If
'Assign variables to input boxes
kGuess = MaskedTextBox(0)
FluxGuess = MaskedTextBox(2) 'neutron/cm^2
SigmaA = MaskedTextBox(1) '1/cm
NueSigmaF = CDbI(Text2.Text) '1/cm
DiffusionCoefficient = CDbI(Text3.Text) 'cm
Radius = CDbI(Text4.Text) 'cm
h = CDbI(Text5.Text) 'spatial distance between nodes cm
n = Radius / h 'number of nodes along radius
DeltaR = h 'cm
DeltaZ = h 'cm
MaxIterations = 1000 'number of iterations for convergence of k
and source
Epsilonk = 0.00001 'acceptable error in k
EpsilonS = 0.015 'acceptable error in S (nueSigmaF*Flux)
kCriticalityTolerance = 0.0001
Dim S1(), S2(), k(), Flux(), ErrorS(), FluxData() 'As Integer
ReDim S1(n)
ReDim S2(n)
ReDim Flux(n)
ReDim k(MaxIterations + 1)
ReDim ErrorS(n - 1)
ReDim FluxData(n, 1)
'build initial flux and source guess
Restart1: 'Used in ProgramError
For i = 0 To n - 1
Flux(i) = FluxGuess
S1(i) = NueSigmaF * Flux(i)
Next i
Restart2: 'Used in ProgramError
k(0) = kGuess
On Error GoTo ProgramError
test = 1 / k(0) 'Check if user input k value.
'!Outer iterations
For i = 0 To MaxIterations
'Flux is the only term coming out of Call statement
Call LinearFiniteDifference(S1, n, k(i), SigmaA, DiffusionCoefficient, h, Radius, Flux)
'Convert flux back into source for comparison with previous source
For m = 0 To n
S2(m) = NueSigmaF * Flux(m) 'convert flux to S(n+1; neutron/cm^3
Next m
'!Calc area under curve using composite trap rule
Sum1 = (h / 2#) * S1(0) '0
Sum2 = (h / 2#) * S2(0) '0

```

```

For j = 1 To n - 1
Sum2 = Sum2 + h * S2(j)
Sum1 = Sum1 + h * S1(j)
Next j
Integral S(n+1)/(1/k(n))*Integral S(n)
'Build array of Source errors to use in tolerance test
On Error GoTo ProgramError
k(i + 1) = k(i) * Sum2 / Sum1 'Equation 5-274 Duderstadt
'Calculate relative error between old and new source
!run to n-1 because S=0 at BC
For l = 0 To n - 1
ErrorS(l) = Abs((S2(l) - S1(l)) / S2(l)) 'Equation 5-275 Duderstadt
Next
'Find maximum value of ErrorS()
MaxErrorS = ErrorS(0)
For l = 1 To n - 1
If (ErrorS(l) > MaxErrorS) Then
MaxErrorS = ErrorS(l)
End If
Next l
'Check for tolerances
If ((Abs((k(i + 1) - k(i)) / k(i)) < Epsilonk)) And (MaxErrorS < EpsilonS) Then 'Equation
5-275 Duderstadt
'End outer iterations check for convergence if true
kEffective = k(i + 1)
Exit For
End If
'reset S1=S2 for next iteration
For j = 0 To n
S1(j) = S2(j)
Next j
kEffective = k(i + 1)
Next i 'end outer iteration
'Check if k=1, if so k=keff=critical Rx
NumberIterations = I
MultFactor = Format(k(i), "#.#####")
If (kEffective > 1# - kCriticalityTolerance) Then
Text8 = "Yes"
Text9 = MultFactor 'k(i)
Else
Text8 = "No"
Text9 = MultFactor 'k(i)
'Use perturbation to assist the user in changing SigmaA to get criticality
DeltaSigmaA = Abs(1# / kGuess - 1#) * (-NueSigmaF) 'Equation 5-306 Duderstadt
Text10 = SigmaA + DeltaSigmaA 'adjust new SigmaA for criticality
End If

```

```

If (i < MaxIterations) Then
    Text11 = NumberIterations
Else
    Text14 = "Exceeded maxiterations before convergence"
End If
#####
'Build chart using Excel and OLE capability
Dim ExcelApp As Object
Dim ExcelChart As Object
Dim ChartTypeVal As Integer
    '4100 is the value for the MS Excel constant xl3DColumn. Visual
    'BASIC does not understand MS Excel constants, so the value must be
    'used instead.
    'xlLine=4
    'xlXYScatter = -4169
    'xl3DSurface=-4103
'Define my chart typ
ChartTypeVal = -4169
Set ExcelApp = CreateObject("excel.application")
ExcelApp.Visible = False 'Will not see Excel load, build, and chart
ExcelApp.Workbooks.Add
'Populate the worksheet in Excel with the power (W/cm^3)
'Power conversion per Ott
For rwIndex = 0 To n
    ExcelApp.Cells(rwIndex + 2, 1).Value = h * rwIndex
    ExcelApp.Cells(rwIndex + 2, 2).Value = Flux(rwIndex) * NueSigmaF / (2.43 * 3.1 *
10 ^ 10) 'W/cm^3
Next rwIndex
'select rows and columns in worksheet to chart Starts at A1 and highlights all values
ExcelApp.Range("A1").CurrentRegion.Select
Set ExcelChart = ExcelApp.Charts.Add()
ExcelChart.Type = ChartTypeVal
ExcelChart.SeriesCollection(1).Name = ""Power""
With ExcelChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Radius (cm)"
    .Axes(xlValue, xlPrimary).HasTitle = True
    .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Power (Watts/cm^3)"
End With
ExcelChart.SaveAs [Text1] 'Save chart/data per user input
    'Using the square brackets tells Visual Basic that this is an
    'MS Excel command not a Visual Basic command.
OLE2.CreateLink (Text1) 'Link to saved chart
OLE2.Update 'allow immediate update of excel chart
ExcelApp.Quit
Set ExcelChart = Nothing

```

```

Set ExcelApp = Nothing
#####
ProgramError:
Select Case Err.Number
Case mnErrOverflow
MsgBox ("You must guess an initial flux to get a non-trivial solution; the code will guess
1.0E10 neutrons/sec-cm^2")
FluxGuess = 10000000000#
Resume Restart1
Case mnErrDivByZero
MsgBox ("You must input a value for k, the code will assume kguess=0.9")
kGuess = 0.9
Resume Restart2
End Select
End Sub

```

```

Sub LinearFiniteDifference(S1, m, k, SigmaA, DiffusionCoefficient, h, Radius, Flux)

```

```

'-----
'!Solves  $y(n+1) = -1/r * y(n+1) + \text{SigmaA}/D * y(n+1) - 1/(D * k(n)) * \text{NueSigmaF} * y(n)$ 
'!for  $y(n+1)$ .  $k(n)$  and  $y(n)$  are calculated in main program.
'Algorithm 11.3 Burden & Faires
LowerLimit = 0 'inner radius
UpperLimit = m * h 'outer radius
alpha = 0 'I.C.  $y(\text{LowerLimit}) = \text{alpha}$ 
beta = 0 'I.C.  $y(\text{upperLimit}) = \text{beta}$ 
n = m - 1
ReDim A(n) 'lower diagonal
ReDim b(n) 'diagonal
ReDim C(n) 'upper diagonal
ReDim D(n) 'A.x=d The d vector
ReDim l(n)
ReDim u(n)
ReDim Z(n)
ReDim w(n + 1)
'!Set distance of first node
x = LowerLimit + h 'cm
'!Build diagonals;
'!a(1) = w(i)+w(i-1) because flux(0)=flux(1)at interior BC
A(1) = 2 + h ^ 2 * q(x, SigmaA, DiffusionCoefficient) + (-1 - (h / 2#) * p(x)) 'No units
b(1) = -1 + (h / 2#) * p(x) 'no units
D(1) = -h ^ 2 * r(x, h, n, k, DiffusionCoefficient, S1) 'neutron/cm^2
For i = 2 To n - 1
x = LowerLimit + i * h
A(i) = 2 + h ^ 2 * q(x, SigmaA, DiffusionCoefficient) 'no units
b(i) = -1 + (h / 2#) * p(x) 'no units

```

```

    C(i) = -1 - (h / 2#) * p(x) 'no units
    D(i) = -h ^ 2 * r(x, h, n, k, DiffusionCoefficient, S1) 'neutrons/cm^2
Next
x = UpperLimit - h
A(n) = 2 + h ^ 2 * q(x, SigmaA, DiffusionCoefficient) 'no units
C(n) = -1 - (h / 2#) * p(x) 'no units
D(n) = -h ^ 2 * r(x, h, n, k, DiffusionCoefficient, S1) + (1 - (h / 2#) * p(x)) * beta
'neutron/cm^2
'Crout Factorization for tridiagonal linear systems
!Back substitute for solution
l(1) = A(1) 'no units
u(1) = b(1) / A(1) 'no units
Z(1) = D(1) / l(1) 'neutron/cm^2
For i = 2 To n - 1
    l(i) = A(i) - C(i) * u(i - 1) 'no units
    u(i) = b(i) / l(i) 'no units
    Z(i) = (D(i) - C(i) * Z(i - 1)) / l(i) 'neutron/cm^2
Next
l(n) = A(n) - C(n) * u(n - 1) 'no units
Z(n) = (D(n) - C(n) * Z(n - 1)) / l(n) 'neutron/cm^2
!Set solution flux values
Flux(n + 1) = beta !Flux at outer boundary; neutron/cm^2
Flux(n) = Z(n) 'neutron/cm^2
For i = n - 1 To 1 Step -1
    Flux(i) = Z(i) - u(i) * Flux(i + 1) 'neutron/cm^2
Next
Flux(0) = Flux(1) !Set BC dFlux/dr=0; neutron/cm^2
End Subroutine LinearFiniteDifference

End Sub
!NOTE: x=radius of core in cm
Function p(x)
Real (8):: x
p = -1 / x '1/cm
End Function

Function q(x, SigmaA, DiffusionCoefficient)
q = SigmaA / DiffusionCoefficient '1/cm^2
End Function

Function r(x, h, n, k, DiffusionCoefficient, S1)
!S1 is an array filled by node position. Must convert x to nodal points.
If (x = h) Then
x = 1
ElseIf (x = n) Then
x = n

```

```
Else
x = x / h
End If
r = -1 / k * S1(x) * 1 / DiffusionCoefficient 'neutron/cm^4
```

```
End Function
```

```
Function Checkin(Box)
If Len(Box) = 0 Then
Checkin = 1
End If
End Function
```

```
Private Sub mnu3D2EnergyGroups_Click()
Load TwoEnergyGroup
TwoEnergyGroup.Show
Unload k
End Sub
```

```
Private Sub mnu3DOneEnergyGroupItem_Click()
Load rxfrm
rxfrm.Show
Unload k
End Sub
```

```
Private Sub mnuExitItem_Click()
End
End Sub
```

```
Private Sub mnuHelpItem_Click()
Load Help
Help.Show
End Sub
```

```
Private Sub mnuPrintItem_Click()
k.PrintForm
End Sub
```

```
Private Sub mnuReactorItem_Click()
Load Reactor
Reactor.Show
End Sub
```

```
Private Sub mnuRunItem_Click()
Call Kinetics
End Sub
```



```
Private Sub mnuStartItem_Click()  
Load ReactorCoreModel  
ReactorCoreModel.Show  
Unload k  
End Sub
```

```
Private Sub Option1_Click()  
MsgBox ("Try SigmaA=0.1532 1/cm")  
End Sub
```

Appendix D. Three Dimension, One Energy Group Visual Basic Code

'Thesis code by MAJ Will Harman

'This program calculates the radial and axial flux profile (3D) for a typical right
'circular cylinder. It uses the standard diffusion equation in a one energy group
'(one-speed)homogeneous reactor core. The equation is solved by using the finite central
'difference technique using a blocked tridiagonal solver. The scheme uses a power
iterative
'technique to solve for flux based upon an inital guess of k effective and the flux.
It finds the eigenfunction for the maximum eigenvalue providing the fundamental mode
shape for flux.

'Common error statements

Const mnSaveAsFailed = 1004

Const mnTypeMismatch = 13

Private Sub Form_Load()

Load initial values

Dim kGuess, FluxGuess, SigmaA, NueSigmaF As Double

Dim DiffusionCoefficient, Radius, h As Double

Text1 = 180 'half Rx hieght cm

Text2 = 0.157 'NueSigmaF 1/cm

Text3 = 9.21 'Diffusion Coefficient cm

Text4 = 120 'Core radius cm

Text6 = ""

Text7 = 100000000000# ' * 10 ^ 10 'neutrons/cm^2

Text8 = ""

Text9 = ""

Text10 = " "

Text11 = ""

Text14 = ""

'build fixed selection of mesh spacing in axial and radial cm

Combo1.AddItem "30"

Combo1.AddItem "20"

Combo1.AddItem "15"

Combo1.AddItem "12"

Combo1.AddItem "10"

Combo1.AddItem "6"

Combo1.AddItem "5"

Combo1.AddItem "4"

Combo1.AddItem "3"

Combo1.AddItem "2"

Combo1.AddItem "1"

Combo2.AddItem "Choose half reactor core" 'Select List Case 0

Combo2.AddItem "Choose quarter Rx core" 'Select List Case 1

```

Combo3.AddItem "Reduced Relative Error; Slower Run Time"
Combo3.AddItem "Average Relative Error; Faster Run Time"
End Sub

```

```

Sub Kinetics()
Dim Valu(2) As Double
Dim kGuess As Double
Dim SigmaA As Double
Dim FluxGuess As Double
'Set source document for Excel chart to name and location by user
OLE1.SourceDoc = ("Text6")
OLE1.Visible = True
'Check for numerical entries
ok = 0
For j = 0 To 1
    ykk = Checkin(MaskedTextBox(j))
    ok = ok + ykk
Next j
If ok > 0 Then
MsgBox ("You must enter positive numbers")
End If
'Check for output file location and name
If Text6 = "" Then
MsgBox ("You must input an output file location and name")
End If
'Assign variables to input boxes
kGuess = MaskedTextBox(0)
FluxGuess = MaskedTextBox(2) 'Cdbl(Text7.Text) 'neutron/cm^2
SigmaA = MaskedTextBox(1) 'Cdbl(Text1.Text) '1/cm
NueSigmaF = Cdbl(Text2.Text) '1/cm
DiffusionCoefficient = Cdbl(Text3.Text) 'cm
Radius = Cdbl(Text4.Text) 'cm
ZHeight = Cdbl(Text1.Text) 'cm
On Error GoTo ProgramError
h = Cdbl(Combo1.Text) 'spatial distance between nodes; cm
n = Radius / h 'number of nodes along radius
m = ZHeight / h 'number of nodes along z axis
DeltaR = h 'cm
DeltaZ = h 'cm
MaxIterations = 1000 'number of iterations for convergence of k and source
'Provide the user with a choice of relative run times and errors
If (Combo3.Text = "Reduced Relative Error; Slower Run Time") Then
kTolerance = 0.0000001
kCriticalTol = 0.000001
Else
kTolerance = 0.00001

```

```

kCriticalTol = 0.0001
End If
Epsilonk = kTolerance 'acceptable error in k: Ref Ott
EpsilonS = 0.015 'acceptable error in S (nueSigmaF*Flux)
kCriticalityTolerance = kCriticalTol
Dim S1(), S2(), k(), Flux(), ErrorS(), FluxRadial(), FluxAxial() 'As Integer
Dim A(), l(), u()
ReDim S1(n - 1, m - 1) 'S(n)
ReDim S2(n - 1, m - 1) 'S(n+1)
ReDim k(MaxIterations + 1)
ReDim Flux(n - 1, m - 1)
ReDim ErrorS(n - 1, m - 1)
ReDim FluxRadial(n, 1)
ReDim FluxAxial(m, 1)
ReDim A(n - 1, m - 1) 'stores main diagonal of matrix
ReDim l(n - 1, m - 1) 'stores lower diag of tridiag matrix
ReDim u(n - 1, m - 1) 'stores upper diag of tridiag matrix
'Calculate initial source
For i = 0 To n - 1
For j = 0 To m - 1 '(m - 1) / 2 '
Flux(i, j) = FluxGuess 'neutron/cm^2
S1(i, j) = NueSigmaF * Flux(i, j) 'neutron/cm^3;
Next j
Next i
Next i
*****
'Build diagonals of the tridiagonal in the blocked system by selecting reactor
Select Case Combo2.ListIndex 'either half or quarter of Rx. Boundaries change.
Case 0 'Half Rx core
For j = 1 To m - 1 'j=row position along z axis
For i = 1 To n - 1 'i=column position along radius
If i = 1 Then
Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaA / DiffusionCoefficient
Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * DeltaR ^ 2) 'add in boundary condition
flux(0,j)=flux(1,j)
A(i, j) = ((Hold1 + Hold2) * DeltaZ ^ 2) 'no units
Else
'no units for A(i,j)
A(i, j) = (((-2# / DeltaR ^ 2) - 2# / DeltaZ ^ 2 - SigmaA / DiffusionCoefficient) *
DeltaZ ^ 2)
End If
Next i
Next j

Case 1 'Quarter Rx core
For j = 1 To m - 1 'j=row position along z axis
For i = 1 To n - 1 'i=column position along radius

```

```

If (i = 1 And j = m - 1) Then 'flux(i,j)=flux(i,j-1)=flux(i-1,j)
  Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaA / DiffusionCoefficient
  Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
flux(0,j)=flux(1,j)
  A(i, j) = ((Hold1 + Hold2 + 1 / DeltaZ ^ 2) * DeltaZ ^ 2) 'no units
  ElseIf i = 1 Then 'flux(i,j)=flux(i-1,j)
    Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaA / DiffusionCoefficient
    Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
flux(0,j)=flux(1,j)
  A(i, j) = ((Hold1 + Hold2) * DeltaZ ^ 2) 'no units
  ElseIf j = m - 1 Then 'flux(i,j)=flux(i,j-1)
    A(i, j) = DeltaZ ^ 2 * ((-2# / DeltaR ^ 2) - (1# / DeltaZ ^ 2) - SigmaA /
DiffusionCoefficient)
  Else
    A(i, j) = (((-2# / DeltaR ^ 2) - 2# / DeltaZ ^ 2 - SigmaA / DiffusionCoefficient) *
DeltaZ ^ 2)
  End If
Next i
Next j
End Select
'build lower diagonal of tridiagonal matrix
For j = 1 To m - 1 'j=row position along z axis
  For i = 1 To n - 2 'i=column position along radius
    l(i, j) = (1# / DeltaR ^ 2 - 1# / (2# * ((i + 1#) * DeltaR ^ 2))) * DeltaZ ^ 2 'no units
  Next i
Next j
'build upper diag. of tridiag. matrix
For j = 1 To m - 1
  For i = 1 To n - 2 'no units for u(i,j)
    u(i, j) = (1# / DeltaR ^ 2 + 1# / (2# * (i * DeltaR ^ 2))) * DeltaZ ^ 2
  Next i
Next j
*****
k(0) = kGuess
'!Outer iterations
For i = 0 To MaxIterations
  Flux is the only term coming out of Call statement and it is built
  'so that the first column of the flux matrix equals the flux in the
  'm-1 row of the Rx core.
  Call ThreeDSolver(S1, A, l, u, n, m, k(i), SigmaA, DiffusionCoefficient, DeltaR, DeltaZ,
Flux)
  'build 3D S2
  Select Case Combo2.ListIndex 'half or quarter Rx
  Case 0 'half of Rx
    For ii = 0 To n - 1
    For j = 0 To m - 1

```

```

If j = 0 Then
S2(ii, 0) = 0 'BC
ElseIf (ii = 0 And j <> 0) Then
S2(0, j) = NueSigmaF * Flux(1, m - j) 'Flux(0,j)=Flux(1,j)
Else
S2(ii, j) = NueSigmaF * Flux(ii, m - j)
End If
Next j
Next ii

```

```

Case 1 'Quarter of Rx
For ii = 0 To n - 1
For j = 0 To m - 1
If (ii = 0 And j = 0) Then
S2(0, 0) = NueSigmaF * Flux(1, m - 1)
ElseIf (ii = 0 And j <> 0) Then
S2(0, j) = NueSigmaF * Flux(1, m - j)
ElseIf (j = 0 And ii <> 0) Then
S2(ii, 0) = NueSigmaF * Flux(ii, m - 1)
Else
S2(ii, j) = NueSigmaF * Flux(ii, m - j)
End If
Next j
Next ii
End Select

```

```

Sum1 = 0
Sum2 = 0
'Build 3D integration of S(n+1) and S(n) using composite trap. rule
For j = 1 To m - 1
For b = 1 To n - 1
Sum1 = Sum1 + b * S1(b, j)
Sum2 = Sum2 + b * S2(b, j)
Next b
Next j
Integral S(n+1)/(1/k(n))*Integral S(n) to find next k value
'Build array of Source errors to use in tolerance test
k(i + 1) = k(i) * Sum2 / Sum1 'Equation 5-275 Duderstadt
'Calc 3D relative error between old and new source
For ii = 1 To n - 1
For j = 1 To m - 1
ErrorS(ii, j) = Abs((S2(ii, j) - S1(ii, j)) / Abs(S2(ii, j)))
Next j
Next ii
'Find maximum value of ErrorS()
MaxErrorS = ErrorS(1, 1)

```

```

For ii = 1 To n - 1
For j = 1 To m - 1
If (ErrorS(ii, j) > MaxErrorS) Then
MaxErrorS = ErrorS(ii, j)
End If
Next j
Next ii
'Check for tolerances
If ((Abs((k(i + 1) - k(i)) / k(i)) < Epsilonk)) And (MaxErrorS < EpsilonS) Then
'End outer iterations check for convergence
kEffective = k(i + 1)
Exit For
End If
'Reassign S2 to S1 for the next iteration
For ii = 1 To m - 1
For j = 1 To n - 1
S1(j, ii) = S2(j, ii)
Next j
Next ii
kEffective = k(i + 1)
Next i 'end outer iteration

'Check if k=1, if so k=keff=critical Rx
NumberIterations = I
MultFactor = Format(k(i), "#.#####")
If (kEffective > 1# - kCriticalityTolerance) Then
Text8 = "Yes"
Text9 = MultFactor & " k(i)
Else
Text8 = "No"
Text9 = MultFactor & " k(i)
'Use perturbation to assist the user in changing SigmaA to get criticality
DeltaSigmaA = Abs(1# / kGuess - 1#) * (-NueSigmaF)
Text10 = SigmaA + DeltaSigmaA
End If
Text11 = NumberIterations
' Text14 = "Exceeded maxiterations before convergence"
#####
'Build Excel chart and spreadsheet
Dim ExcelApp As Object
Dim ExcelChart As Object
Dim ChartTypeVal As Integer
'4100 is the value for the MS Excel constant xl3DColumn. Visual
'Basic does not understand MS Excel constants, so the value must be
'used instead.
xlLine=4

```

```

xlXYSscatter = -4169
xl3DSurface=-4103
ChartTypeVal = -4103
Set ExcelApp = CreateObject("excel.application")
ExcelApp.Visible = False 'Hide the Excel application from the user
ExcelApp.Workbooks.Add
'Populate the Excel spreadsheet with core power values and locations
Select Case Combo2.ListIndex 'half or quarter Rx
Case 0 'half Rx
    For rwIndex = 0 To n
        ExcelApp.Cells(rwIndex + 2, 1).Value = h * rwIndex
        For colIndex = 0 To m
            ExcelApp.Cells(1, colIndex + 2).Value = h * colIndex
            If rwIndex = n Then
                ExcelApp.Cells(rwIndex + 2, colIndex + 2).Value = 0
            ElseIf colIndex = m Then
                ExcelApp.Cells(rwIndex + 2, colIndex + 2).Value = 0
            ElseIf colIndex = 0 Then
                ExcelApp.Cells(rwIndex + 2, 2).Value = 0
            Else
                ExcelApp.Cells(rwIndex + 2, colIndex + 2).Value = S2(rwIndex, colIndex) / (2.43 *
3.1 * 10 ^ 10) 'NueSigmaF
            End If
        Next colIndex
    Next rwIndex
Case 1 'Quarter Rx
For rwIndex = 0 To n
    ExcelApp.Cells(rwIndex + 2, 1).Value = h * rwIndex
For colIndex = 0 To m
    ExcelApp.Cells(1, colIndex + 2).Value = h * colIndex
    If rwIndex = n Then
        ExcelApp.Cells(rwIndex + 2, colIndex + 2).Value = 0
    ElseIf colIndex = m Then
        ExcelApp.Cells(rwIndex + 2, colIndex + 2).Value = 0
    ElseIf colIndex = 0 Then
        ExcelApp.Cells(rwIndex + 2, colIndex + 2).Value = S2(rwIndex, colIndex) / (2.43
* 3.1 * 10 ^ 10) 'NueSigmaF
    Else
        ExcelApp.Cells(rwIndex + 2, colIndex + 2).Value = S2(rwIndex, colIndex) / (2.43 *
3.1 * 10 ^ 10) 'NueSigmaF
    End If
    Next colIndex
Next rwIndex
End Select
'select rows and columns in worksheet to chart
ExcelApp.Range("A1").CurrentRegion.Select

```



```

Set ExcelChart = ExcelApp.Charts.Add()
'Add legend information
ExcelChart.Type = ChartTypeVal
  With ExcelChart
    .HasTitle = True
    .ChartTitle.Characters.Text = "Power Plot of Reactor Core"
    .Axes(xlCategory).HasTitle = True
    .Axes(xlCategory).AxisTitle.Characters.Text = "Height (cm)"
    .Axes(xlSeries).HasTitle = True
    .Axes(xlSeries).AxisTitle.Characters.Text = "Radius (cm)"
    .Axes(xlValue).HasTitle = True
    .Axes(xlValue).AxisTitle.Characters.Text = "Power (Watts/cm^3)"
  End With
  With ExcelChart.Axes(xlCategory)
    .HasMajorGridlines = False
    .HasMinorGridlines = False
  End With
  With ExcelChart.Axes(xlSeries)
    .HasMajorGridlines = False
    .HasMinorGridlines = False
  End With
  With ExcelChart.Axes(xlValue)
    .HasMajorGridlines = True
    .HasMinorGridlines = False
  End With
  ExcelChart.WallsAndGridlines2D = False
  ExcelChart.HasLegend = False
On Error GoTo ProgramError
ExcelChart.SaveAs [Text6]
'Link Excel chart to saved name and location for update
OLE1.CreateLink (Text6)
OLE1.Update 'allow immediate update of excel chart
ExcelApp.Quit
  Set ExcelChart = Nothing
  Set ExcelApp = Nothing
#####
ProgramError:
Select Case Err.Number
Case mnTypeMismatch

  MsgBox ("You must select a mesh spacing; the code will assuem 20 cm")
  h = 20
  Resume Next
Case mnSaveAsFailed
  MsgBox ("You must provide a name and file storage location to update the plot and store
data")

```

```
Resume Next
End Select
End Sub
```

```
*****
```

```
Function Checkin(Box)
If Len(Box) = 0 Then
  Checkin = 1
End If
End Function
```

```
*****
*****
```

```
Sub ThreeDSolver(S1, A, l, u, n, m, kEffective, SigmaA, DiffusionCoefficient, DeltaR,
DeltaZ, X2)
```

```
'This 3D block tridiagonal solver uses an iterative technique similar to Gauss-Seidel.
'It sets up the banded tridiagonal system based on the discretized right circular cyl
'diffusion equation and dismantles that into a blocked tridiagonal system.
```

```
'Each block is then solved with a standard tridiagonal solver using initial guesses
'for the solution. The system iterates until convergence of the solution (flux at the
'inner mesh points)
```

```
ReDim b(n - 1, m - 1) 'stores B of A.x=B
```

```
ReDim X2(n - 1, m - 1) 'stores solution at mesh points
```

```
ReDim X1(n - 1, m - 1) 'Stores the previous solution at mesh points
```

```
ReDim e(n - 1, m - 1) 'stores rhs of block tridiag system
```

```
ReDim AA(n - 1) 'Stores the main diag (jth column of A array) of block tridiag
```

```
ReDim LL(n - 1) 'Stores the lower diag (jth column of l array) of block tridiag
```

```
ReDim UU(n - 1) 'Stores the upper diag (jth column of u array) of block tridiag
```

```
ReDim EE(n - 1) 'Stores the rhs diag of block tridiag
```

```
ReDim XX(n) 'stores jth column solution vector from tridiagonal solver
```

```
ReDim ErrorX(n - 1, m - 1) 'holds the max error in convergence of solution in tridiag
```

```
'build B vector of Ax=B. B contains the iterative guess for flux.
```

```
'Uses S1(i,m-j) to convert the (i,j) values into m-j rows for the b matrix.
```

```
For j = 1 To m - 1
```

```
  For i = 1 To n - 1
```

```
    X1(i, j) = 0 'fill convergence test array X1 with 0
```

```
    X2(i, j) = 0 'fill with 0
```

```
    b(i, j) = DeltaZ ^ 2 * (-S1(i, m - j) / (kEffective * DiffusionCoefficient)) 'neutron/cm^2
```

```
  Next i
```

```
Next j
```

```
'solve block tridiagonal system. See Solution Methods of my thesis.
```

```
For w = 1 To 10000
```

```
  For i = 1 To n - 1
```

```
    e(i, 1) = b(i, 1) - X2(i, 2) 'neutron/cm^2
```

```
    EE(i) = e(i, 1) 'neutron/cm^2 typical
```

```
    LL(i) = l(i, 1) 'LL Lower diagonal of tridiagonal;no units typical
```

```
    AA(i) = A(i, 1) 'AA main diagonal of tridiagonal; no units typical
```

```

UU(i) = u(i, 1) 'UU upper diagonal of tridiagonal; no units typical
Next i
Call Tridiag(n, LL, AA, UU, EE, XX) 'returns flux at row m-1 in core
For i = 1 To n - 1
  X2(i, 1) = XX(i) 'neutrons/cm^2 typical
Next i
For j = 2 To (m - 2)
  For i = 1 To n - 1
    e(i, j) = b(i, j) - X2(i, j - 1) - X2(i, j + 1)
    EE(i) = e(i, j)
    LL(i) = l(i, j)
    AA(i) = A(i, j)
    UU(i) = u(i, j)
  Next i
  Call Tridiag(n, LL, AA, UU, EE, XX)
  For i = 1 To n - 1
    X2(i, j) = XX(i)
  Next i
Next j
For i = 1 To n - 1
  e(i, m - 1) = b(i, m - 1) - X2(i, m - 2)
  EE(i) = e(i, m - 1)
  LL(i) = l(i, m - 1)
  AA(i) = A(i, m - 1)
  UU(i) = u(i, m - 1)
Next i
Call Tridiag(n, LL, AA, UU, EE, XX)
For i = 1 To n - 1
  X2(i, m - 1) = XX(i)
Next i

```

```

For v = 1 To n - 1
  For j = 1 To m - 1
    ErrorX(v, j) = Abs((X2(v, j) - X1(v, j)) / Abs(X2(v, j)))
  Next j
Next v
Find maximum value of ErrorS()
MaxErrorX = ErrorX(1, 1)
For i = 1 To n - 1
  For j = 1 To m - 1
    X1(i, j) = X2(i, j) 'update X1
    If (ErrorX(i, j) > MaxErrorX) Then
      MaxErrorX = ErrorX(i, j)
    End If
  Next j

```

```

Next i
If (Combo3.Text = "Reduced Relative Error; Slower Run Time") Then
Tolerance = 0.00000001
Else
Tolerance = 0.001
End If

If (MaxErrorX < Tolerance) Then
Exit For
End If
*****

'Set boundary conditions
Next w
If (w > 10000) Then
MsgBox ("Exceeded block tridiagonal maxiterations before convergence")
End If
End Sub

Sub Tridiag(n, LL, AA, UU, EE, XX) 'LL=lower diag, AA=Diag, UU=Upper
diag,EE=A.x
'Crout Factorization for tridiagonal linear systems
'!Back substitute for solution
m = n - 1
ReDim Lower(m)
ReDim Upper(m)
ReDim Z(m)
Lower(1) = AA(1) 'no units
Upper(1) = UU(1) / AA(1) 'no units
Z(1) = EE(1) / Lower(1) 'neutron/cm^2
For i = 2 To m - 1
    Lower(i) = AA(i) - LL(i - 1) * Upper(i - 1) 'no units
    Upper(i) = UU(i) / Lower(i) 'no units
    Z(i) = (EE(i) - LL(i - 1) * Z(i - 1)) / Lower(i) 'neutron/cm^2
Next
Lower(m) = AA(m) - LL(m - 1) * Upper(m - 1) 'no units
Z(m) = (EE(m) - LL(m - 1) * Z(m - 1)) / Lower(m) 'no units
'!Set solution flux values
XX(m + 1) = 0 '!Flux at outer boundary; neurton/cm^2
XX(m) = Z(m) 'neutron/cm^2
For i = m - 1 To 1 Step -1
    XX(i) = Z(i) - Upper(i) * XX(i + 1) 'neutron/cm^2
Next
XX(0) = XX(1) '!Set BC dFlux/dr=0; 'neutron/cm^2
End Sub 'Tridiagonal

```

```
Private Sub mnu2DOneEnergyGroupItem_Click()  
Load k  
k.Show  
Unload rxfrm  
End Sub
```

```
Private Sub mnu3DTwoEnergyGroupItem_Click()  
Load TwoEnergyGroup  
TwoEnergyGroup.Show  
Unload rxfrm  
End Sub
```

```
Private Sub mnuExitItem_Click()  
End  
End Sub
```

```
Private Sub mnuHelpItem_Click()  
Load Help  
Help.Show  
End Sub
```

```
Private Sub mnuPrintItem_Click()  
rxfrm.PrintForm  
End Sub
```

```
Private Sub mnuReactorItem_Click()  
Load Reactor  
Reactor.Show  
End Sub
```

```
Private Sub mnuRunItem_Click()  
Call Kinetics  
End Sub
```

```
Private Sub mnuStartFormItem_Click()  
Load ReactorCoreModel  
ReactorCoreModel.Show  
Unload rxfrm  
End Sub
```

```
Private Sub Option1_Click()  
MsgBox ("Try SigmaA=0.1532 1/cm")  
End Sub
```

```
Private Sub Text5_Change()
```

End Sub

Appendix E. Three Dimension, Two Energy Groups Visual Basic Code

Thesis code by MAJ Will Harman

This program calculates the radial and axial Flux/Power profile (3D) for a typical right circular cylinder homogeneous reactor core using two energy groups. It uses the standard

diffusion equation. The equation is solved by using the finite central difference technique using blocked tridiagonal solver.

The scheme uses a power iterative technique to solve for Flux1 and Flux2 based upon an initial guess of k effective and Flux1 and Flux2. It finds the eigenfunction for the maximum

eigenvalue providing the fundamental mode shape for Flux1 and Flux2.

```
Private Sub Form_Load()
```

```
Load initial values
```

```
Dim kGuess, Flux1Guess, SigmaA1, NueSigmaF1 As Double
```

```
Dim DiffusionCoefficient1, Radius, h As Double
```

```
Text1 = 180 half Rx hieght cm
```

```
Text2 = 0.008476 NueSigmaF1 1/cm
```

```
Text3 = 1.2627 DiffusionCoefficient1 cm
```

```
Text4 = 120 Core radius cm
```

```
Text5 = 0.18514 NueSigmaF2 1/cm
```

```
Text6 = ""
```

```
Text7 = 0.3543 DiffusionCoefficient2 cm
```

```
Text8 = ""
```

```
Text9 = ""
```

```
Text10 = " "
```

```
Text11 = ""
```

```
Text12 = 0.121 SigmaR2 1/cm
```

```
Text13 = 0.02619 SigmaR1
```

```
Text14 = 0.01207 SigmaA1 1/cm
```

```
Text15 = 0.121 SigmaA2 1/cm
```

```
Text16 = 0.01412 SigmaScatter12 1/cm
```

```
Text17 = 0 SigmaScatter22 1/cm
```

```
build fixed selection of mesh spacing in axial and radial cm
```

```
Combo1.AddItem "30"
```

```
Combo1.AddItem "20"
```

```
Combo1.AddItem "15"
```

```
Combo1.AddItem "12"
```

```
Combo1.AddItem "10"
```

```
Combo1.AddItem "6"
```

```
Combo1.AddItem "5"
```

```
Combo1.AddItem "4"
```

```
Combo1.AddItem "3"
```

```
Combo1.AddItem "2"
```

```

Combo1.AddItem "1"
Combo2.AddItem "Choose half reactor core" 'Select List Case 0
Combo2.AddItem "Choose quarter Rx core" 'Select List Case 1
Combo3.AddItem "Reduced Relative Error; Slower Run Time"
Combo3.AddItem "Average Relative Error; Faster Run Time"
End Sub

```

```

Sub Kinetics()
Dim Valu(2) As Double
Dim kGuess As Double
Dim SigmaA1 As Double
Dim Flux1Guess As Double
Dim Flux2Guess As Double
'Set source document for Excel chart to name and location by user
OLE1.SourceDoc = ("Text6")
OLE1.Visible = True
OLE2.SourceDoc = ("Text6")
OLE2.Visible = True
'Check for numerical entries
ok = 0
For j = 0 To 1
    ykk = Checkin(MaskedTextBox(j))
    ok = ok + ykk
Next j
If ok > 0 Then
MsgBox ("You must enter positive numbers")
End If
'Check for output file location and name
If Text6 = "" Then
MsgBox ("You must input an output file location and name")
End If
'Assign variables to input boxes
kGuess = MaskedTextBox(0)
Flux1Guess = MaskedTextBox(1) 'Cdbl(Text7.Text) 'neutron/cm^2
Flux2Guess = MaskedTextBox(2)
SigmaA1 = Cdbl(Text14.Text) '1/cm
SigmaA2 = Cdbl(Text15.Text) '1/cm
NueSigmaF1 = Cdbl(Text2.Text) '1/cm
NueSigmaF2 = Cdbl(Text5.Text) '1/cm
DiffusionCoefficient1 = Cdbl(Text3.Text) 'cm
DiffusionCoefficient2 = Cdbl(Text7.Text) 'cm
SigmaR1 = Cdbl(Text13.Text) '1/cm (SigmaTotal-SigmaScatter)
SigmaR2 = Cdbl(Text12.Text) '1/cm (SigmaTotal-SigmaScatter)
SigmaScatter12 = Cdbl(Text16.Text) '1/cm
Radius = Cdbl(Text4.Text) 'cm

```



```

ZHeight = CDBl(Text1.Text) 'cm
h = CDBl(Combo1.Text) 'spatial distance between nodes; cm
n = Radius / h 'number of nodes along radius
m = ZHeight / h 'number of nodes along z axis
DeltaR = h 'cm
DeltaZ = h 'cm
MaxIterations = 1000 'number of iterations for convergence of k and source
'Provide the user a choice of run times and relative error
If (Combo3.Text = "Reduced Relative Error; Slower Run Time") Then
kTolerance = 0.0000001
kCriticalTol = 0.000001
Else
kTolerance = 0.00001
kCriticalTol = 0.0001
End If
EpsilonK = kTolerance 'acceptable error in k; Ref Ott
EpsilonS = 0.015 'acceptable error in S (NueSigmaF1*Flux1)
kCriticalityTolerance = kCriticalTol
Dim S1(), S2(), k(), Flux1(), ErrorS() ', Flux1Radial(), Flux1Axial() 'As Integer
Dim Flux2(), Flux2Radial(), Flux2Axial() 'As Integer
Dim A1(), A2(), l(), u()
ReDim S1(n - 1, m - 1) 'S(n)
ReDim S2(n - 1, m - 1) 'S(n+1)
ReDim k(MaxIterations + 1)
ReDim Flux1(n - 1, m - 1)
ReDim Flux2(n - 1, m - 1)
ReDim ErrorS(n - 1, m - 1)
ReDim A1(n - 1, m - 1) 'stores main diagonal of matrix
ReDim A2(n - 1, m - 1) 'stores main diagonal of matrix
ReDim l(n - 1, m - 1) 'stores lower diag of tridiag matrix
ReDim u(n - 1, m - 1) 'stores upper diag of tridiag matrix
'Calculate initial source two D
For i = 0 To n - 1
For j = 0 To m - 1 '(m - 1) / 2 '
Flux1(i, j) = Flux1Guess 'neutron/cm^2
Flux2(i, j) = Flux2Guess 'neutron/cm^2
S1(i, j) = NueSigmaF2 * Flux2(i, j) + NueSigmaF1 * Flux1(i, j) 'neutron/cm^3;
Cos(3.141592654 * j * DeltaZ / Height) * Bessel
Next j
Next i
*****
'Build diagonals of the tridiagonal in the blocked system for Flux 1
Select Case Combo2.ListIndex 'either half or quarter of Rx. Boundaries change.
Case 0 'Half Rx core
For j = 1 To m - 1 'j=row position along z axis
For i = 1 To n - 1 'i=column position along radius

```

```

If i = 1 Then
  Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaR1 / DiffusionCoefficient1
  Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
Flux1(0,j)=Flux1(1,j)
  A1(i, j) = ((Hold1 + Hold2) * DeltaZ ^ 2) 'no units
  Else
  'no units for A(i,j)
  A1(i, j) = (((-2# / DeltaR ^ 2) - 2# / DeltaZ ^ 2 - SigmaR1 / DiffusionCoefficient1) *
DeltaZ ^ 2)
  End If
Next i
Next j

```

Case 1 'Quarter Rx core

```

For j = 1 To m - 1 'j=row position along z axis
  For i = 1 To n - 1 'i=column position along radius
  If (i = 1 And j = m - 1) Then Flux1(i,j)=Flux1(i,j-1)=Flux1(i-1,j)
  Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaR1 / DiffusionCoefficient1
  Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
Flux1(0,j)=Flux1(1,j)
  A1(i, j) = ((Hold1 + Hold2 + 1 / DeltaZ ^ 2) * DeltaZ ^ 2) 'no units
  ElseIf i = 1 Then Flux1(i,j)=Flux1(i-1,j)
  Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaR1 / DiffusionCoefficient1
  Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
Flux1(0,j)=Flux1(1,j)
  A1(i, j) = ((Hold1 + Hold2) * DeltaZ ^ 2) 'no units
  ElseIf j = m - 1 Then Flux1(i,j)=Flux1(i,j-1)
  A1(i, j) = DeltaZ ^ 2 * ((-2# / DeltaR ^ 2) - (1# / DeltaZ ^ 2) - SigmaR1 /
DiffusionCoefficient1)
  Else
  A1(i, j) = (((-2# / DeltaR ^ 2) - 2# / DeltaZ ^ 2 - SigmaR1 / DiffusionCoefficient1) *
DeltaZ ^ 2)
  End If
  Next i
Next j
End Select

```

'Build diagonal of the tridiagonal for Flux2

Select Case Combo2.ListIndex 'either half or quarter of Rx. Boundaries change.

Case 0 'Half Rx core

```

For j = 1 To m - 1 'j=row position along z axis
  For i = 1 To n - 1 'i=column position along radius
  If i = 1 Then
  Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaR2 / DiffusionCoefficient2
  Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
Flux1(0,j)=Flux1(1,j)
  A2(i, j) = ((Hold1 + Hold2) * DeltaZ ^ 2) 'no units

```

```

Else
  'no units for A(i,j)
  A2(i, j) = (((-2# / DeltaR ^ 2) - 2# / DeltaZ ^ 2 - SigmaR2 / DiffusionCoefficient2) *
DeltaZ ^ 2)
  End If
Next i
Next j

Case 1 'Quarter Rx core
For j = 1 To m - 1 'j=row position along z axis
  For i = 1 To n - 1 'i=column position along radius
    If (i = 1 And j = m - 1) Then Flux1(i,j)=Flux1(i,j-1)=Flux1(i-1,j)
    Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaR2 / DiffusionCoefficient2
    Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
    Flux1(0,j)=Flux1(1,j)
    A2(i, j) = ((Hold1 + Hold2 + 1 / DeltaZ ^ 2) * DeltaZ ^ 2) 'no units
    ElseIf i = 1 Then Flux1(i,j)=Flux1(i-1,j)
    Hold1 = (-2# / DeltaR ^ 2) - (2# / DeltaZ ^ 2) - SigmaR2 / DiffusionCoefficient2
    Hold2 = (1# / DeltaR ^ 2) - 1# / (2# * i * DeltaR ^ 2) 'add in boundary condition
    Flux1(0,j)=Flux1(1,j)
    A2(i, j) = ((Hold1 + Hold2) * DeltaZ ^ 2) 'no units
    ElseIf j = m - 1 Then Flux1(i,j)=Flux1(i,j-1)
    A2(i, j) = DeltaZ ^ 2 * ((-2# / DeltaR ^ 2) - (1# / DeltaZ ^ 2) - SigmaR2 /
DiffusionCoefficient2)
  Else
    A2(i, j) = (((-2# / DeltaR ^ 2) - 2# / DeltaZ ^ 2 - SigmaR2 / DiffusionCoefficient2) *
DeltaZ ^ 2)
  End If
  Next i
Next j
End Select

'build lower diagonal of tridiagonal matrix
For j = 1 To m - 1 'j=row position along z axis
  For i = 1 To n - 2 'i=column position along radius
    l(i, j) = (1# / DeltaR ^ 2 - 1# / (2# * ((i + 1#) * DeltaR ^ 2))) * DeltaZ ^ 2 'no units
  Next i
Next j

'build upper diag. of tridiag. matrix
For j = 1 To m - 1
  For i = 1 To n - 2 'no units for u(i,j)
    u(i, j) = (1# / DeltaR ^ 2 + 1# / (2# * (i * DeltaR ^ 2))) * DeltaZ ^ 2
  Next i
Next j
*****
k(0) = kGuess
'!Outer iterations

```

```

For i = 0 To MaxIterations
'Flux1 is the only term coming out of Call statement and it is built
'so that the first column of the Flux1 matrix equals the Flux1 in the
'm-1 row of the Rx core.
SolvingFlux = 1 'Selection of B vector in A.x=B
'Solve for fast flux values
Call ThreeDSolver(SolvingFlux, S1, A1, l, u, n, m, k(i), DeltaR, DeltaZ, Flux1, Flux1)
SolvingFlux = 2 'Selection of B vector in A.x=B
'Use fast flux values and solve for thermal values
Call ThreeDSolver(SolvingFlux, S1, A2, l, u, n, m, k(i), DeltaR, DeltaZ, Flux1, Flux2)
'build 3D S2
Select Case Combo2.ListIndex 'half or quarter Rx
Case 0 'half of Rx
For ii = 0 To n - 1
For j = 0 To m - 1
If j = 0 Then
S2(ii, 0) = 0 'BC
ElseIf ii = 0 Then
S2(0, j) = NueSigmaF1 * Flux1(1, m - 1) + NueSigmaF2 * Flux2(1, m - j)
'Flux1/2(0,j)=Flux1/2(1,j)
Else
S2(ii, j) = NueSigmaF1 * Flux1(ii, m - j) + NueSigmaF2 * Flux2(ii, m - j)
End If
Next j
Next ii

Case 1 'Quarter of Rx
For ii = 0 To n - 1
For j = 0 To m - 1
If (ii = 0 And j = 0) Then
S2(0, 0) = NueSigmaF1 * Flux1(1, m - 1) + NueSigmaF2 * Flux2(1, m - 1)
ElseIf ii = 0 Then
S2(0, j) = NueSigmaF1 * Flux1(1, m - j) + NueSigmaF2 * Flux2(1, m - j)
ElseIf (j = 0 And ii <> 0) Then
S2(ii, 0) = NueSigmaF1 * Flux1(ii, m - 1) + NueSigmaF2 * Flux2(ii, m - 1)
Else
S2(ii, j) = NueSigmaF1 * Flux1(ii, m - j) + NueSigmaF2 * Flux2(ii, m - j)
End If
Next j
Next ii
End Select

Sum1 = 0
Sum2 = 0
'Build 3D integration of S(n+1) and S(n) using composite trap. rule
For j = 1 To m - 1

```

```

For b = 1 To n - 1
    Sum1 = Sum1 + b * S1(b, j)
    Sum2 = Sum2 + b * S2(b, j)
Next b
Next j
Integral S(n+1)/(1/k(n)*Integral S(n)) to find next k value
'Build array of Source errors to use in tolerance test
k(i + 1) = k(i) * Sum2 / Sum1 'Equation 5-275 Duderstadt
'Calc 3D relative error between old and new source
For ii = 1 To n - 1
    For j = 1 To m - 1
        ErrorS(ii, j) = Abs((S2(ii, j) - S1(ii, j)) / Abs(S2(ii, j)))
    Next j
Next ii
'Find maximum value of ErrorS()
MaxErrorS = ErrorS(1, 1)
For ii = 1 To n - 1
    For j = 1 To m - 1
        If (ErrorS(ii, j) > MaxErrorS) Then
            MaxErrorS = ErrorS(ii, j)
        End If
    Next j
Next ii
'Check for tolerances
If ((Abs((k(i + 1) - k(i)) / k(i)) < Epsilonk)) And (MaxErrorS < EpsilonS) Then
    'End outer iterations check for convergence
    kEffective = k(i + 1)
Exit For
End If
'Reassign S2 to S1 for the next iteration
For ii = 1 To m - 1
    For j = 1 To n - 1
        S1(j, ii) = S2(j, ii)
    Next j
Next ii
kEffective = k(i + 1)
Next i 'end outer iteration
'Check if k=1, if so k=keff=critical Rx
NumberIterations = I
MultFactor = Format(k(i), "#.#####")
If (kEffective > 1# - kCriticalityTolerance) Then
    Text8 = "Yes"
    Text9 = MultFactor 'k(i)
Else
    Text8 = "No"
    Text9 = MultFactor 'k(i)

```

```

MsgBox ("The system is not critical. Please try changing the core composition density"
& _
" or core geometry")
End If

```

```

Text11 = NumberIterations
If (i > 1000) Then
MsgBox ("Exceeded maxiterations before convergence")
End If
#####
'Build Excel chart and spreadsheets
Dim ExcelApp As Object
Dim ExcelChart1 As Object
Dim ExcelChart2 As Object
Dim ChartTypeVal As Integer
'4100 is the value for the MS Excel constant xl3DColumn. Visual
'Basic does not understand MS Excel constants, so the value must be
'used instead.
'xlLine=4
'xlXYScatter = -4169
'xl3DSurface=-4103
ChartTypeVal = -4169 '4103
Set ExcelApp = CreateObject("excel.application")
ExcelApp.Visible = False
ExcelApp.Workbooks.Add
'Allow the user to choose which node to plot the data on both radial and axial
Dim prompt1, prompt2
prompt1 = "The number of interior radial mesh spaces =" & n _
& ". Please choose the mesh point between 0 and " & n & " to plot the axial power."
plotAxial = InputBox$(prompt1)
prompt2 = "The number of interior axial mesh spaces =" & m _
& ". Please choose the interior mesh point between 0 and " & m & " to plot the radial
power."
plotRadial = InputBox$(prompt2)
Select Case Combo2.ListIndex 'half or quarter Rx
Case 0 'half Rx
For rwIndex = 0 To n
Fill Excel sheet1 with Radial power data
ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 1).Value = h * rwIndex
ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 2).Value = Flux1(rwIndex, m -
plotRadial) * NueSigmaF1 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 3).Value = Flux2(rwIndex, m -
plotRadial) * NueSigmaF2 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 4).Value = (Flux1(rwIndex, m -
plotRadial) * NueSigmaF1 + Flux2(rwIndex, m - plotRadial) * NueSigmaF2) / (2.43 *
3.1 * 10 ^ 10) ^W/cm^3

```

```

Next rwIndex
For colIndex = 0 To m
'fill Excel sheet2 with axial power data
    ExcelApp.Sheets("Sheet2").Cells(m - colIndex + 2, 1).Value = h * colIndex
    ExcelApp.Sheets("Sheet2").Cells(colIndex + 2, 2).Value = Flux1(plotAxial,
colIndex) * NueSigmaF1 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
    ExcelApp.Sheets("Sheet2").Cells(colIndex + 2, 3).Value = Flux2(plotAxial,
colIndex) * NueSigmaF2 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
    ExcelApp.Sheets("Sheet2").Cells(colIndex + 2, 4).Value = (Flux1(plotAxial,
colIndex) * NueSigmaF1 + Flux2(plotAxial, colIndex) * NueSigmaF2) / (2.43 * 3.1 * 10
^ 10) ^W/cm^3
Next colIndex
Case 1 'Quarter Rx
For rwIndex = 0 To n
    ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 1).Value = h * rwIndex
    ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 2).Value = Flux1(rwIndex, m -
plotRadial) * NueSigmaF1 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
    ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 3).Value = Flux2(rwIndex, m -
plotRadial) * NueSigmaF2 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
    ExcelApp.Sheets("Sheet1").Cells(rwIndex + 2, 4).Value = (Flux1(rwIndex, m -
plotRadial) * NueSigmaF1 + Flux2(rwIndex, m - plotRadial) * NueSigmaF2) / (2.43 *
3.1 * 10 ^ 10) ^W/cm^3
Next rwIndex
For colIndex = 0 To m
    ExcelApp.Sheets("Sheet2").Cells(m - colIndex + 2, 1).Value = h * colIndex
    ExcelApp.Sheets("Sheet2").Cells(colIndex + 2, 2).Value = Flux1(plotAxial,
colIndex) * NueSigmaF1 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
    ExcelApp.Sheets("Sheet2").Cells(colIndex + 2, 3).Value = Flux2(plotAxial,
colIndex) * NueSigmaF2 / (2.43 * 3.1 * 10 ^ 10) ^W/cm^3
    ExcelApp.Sheets("Sheet2").Cells(colIndex + 2, 4).Value = (Flux1(plotAxial,
colIndex) * NueSigmaF1 + Flux2(plotAxial, colIndex) * NueSigmaF2) / (2.43 * 3.1 * 10
^ 10) ^W/cm^3
Next colIndex
End Select
'select rows and columns in worksheet to chart
ExcelApp.Sheets("Sheet1").Range("A1").CurrentRegion.Select
Set ExcelChart1 = ExcelApp.Charts.Add()
ExcelApp.Sheets("Sheet2").Range("A1").CurrentRegion.Select
ExcelChart1.Type = ChartTypeVal
ExcelChart1.SeriesCollection(1).Name = """"Thermal""""
ExcelChart1.SeriesCollection(2).Name = """"Fast""""
ExcelChart1.SeriesCollection(3).Name = """"Total""""
' ExcelChart.Location Where:=xlLocationAsObject, Name:="Sheet1"
With ExcelChart1
    .HasTitle = True
    .ChartTitle.Characters.Text = "Power in Reactor Core"

```

```

        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Radius (cm)"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Power (Watts/cm^3)"
End With
ExcelChart1.HasLegend = True
'Build Chart 2 for Core Height profile
ExcelApp.Sheets("Sheet2").Select 'Range("A1").CurrentRegion.Select
ExcelApp.Sheets("Sheet2").Range("A1").CurrentRegion.Select
Set ExcelChart2 = ExcelApp.Charts.Add()
    ExcelChart2.Type = ChartTypeVal
    ExcelChart2.SeriesCollection(1).Name = """"Thermal""""
    ExcelChart2.SeriesCollection(2).Name = """"Fast""""
    ExcelChart2.SeriesCollection(3).Name = """"Total""""
With ExcelChart2
    .HasTitle = True
    .ChartTitle.Characters.Text = "Power in Reactor Core"
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Core Height(cm)"
    .Axes(xlValue, xlPrimary).HasTitle = True
    .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Power (Watts/cm^3)"
End With
ExcelChart2.HasLegend = True
'save chart, activate chart, OLE link chart, OLE update chart for chart 1 & 2
ExcelChart1.SaveAs [Text6]
ExcelApp.Sheets("Chart1").Select 'activate chart
OLE1.CreateLink (Text6)
OLE1.Update 'allow immediate update of excel chart
ExcelApp.Sheets("Chart2").Select 'Activate chart
OLE2.CreateLink (Text6)
OLE2.Update
ExcelApp.Quit
Set ExcelChart = Nothing
Set ExcelApp = Nothing
#####
End Sub
*****

Function Checkin(Box)
If Len(Box) = 0 Then
    Checkin = 1
End If
End Function

*****
*****
Sub ThreeDSolver(SolvingFlux, S1, A, l, u, n, m, kEffective, DeltaR, DeltaZ, Flux1, X2)

```


This 3D block tridiagonal solver uses an iterative technique similar to Gauss-Seidel. It sets up the banded tridiagonal system based on the discretized right circular cylinder diffusion equation and dismantles that into a blocked tridiagonal system.

Each block is then solved with a standard tridiagonal solver using initial guesses for the solution. The system iterates until convergence of the solution (Flux1 at the inner mesh points)

```

SigmaScatter12 = CDbI(Text16.Text) '1/cm
DiffusionCoefficient1 = CDbI(Text3.Text) 'cm
DiffusionCoefficient2 = CDbI(Text7.Text) 'cm
ReDim b(n - 1, m - 1) 'stores B of A.x=B
ReDim X2(n, m) 'stores solution at mesh points
ReDim X1(n, m) 'Stores the previous solution at mesh points
ReDim e(n - 1, m - 1) 'stores rhs of block tridiag system
ReDim AA(n - 1) 'Stores the main diag (jth column of A array) of block tridiag
ReDim LL(n - 1) 'Stores the lower diag (jth column of l array) of block tridiag
ReDim UU(n - 1) 'Stores the upper diag (jth column of u array) of block tridiag
ReDim EE(n - 1) 'Stores the rhs diag of block tridiag
ReDim XX(n) 'stores jth column solution vector from tridiagonal solver
ReDim ErrorX(n - 1, m - 1) 'holds the max error in convergence of solution in tridiag
Fill solutions with 0
For j = 0 To m
    For i = 0 To n
        X1(i, j) = 0 'fill convergence test array X1 with 0
        X2(i, j) = 0 'fill with 0
    Next i
Next j
'build B vector of Ax=B. B contains the iterative guess for Flux1.
If SolvingFlux = 1 Then
For j = 1 To m - 1
    For i = 1 To n - 1
        b(i, j) = DeltaZ ^ 2 * (-S1(i, m - j) / (kEffective * DiffusionCoefficient1))
'neutron/cm^2
    Next i
Next j
Else 'SolvingFlux = 2
For j = 1 To m - 1
    For i = 1 To n - 1
        b(i, j) = DeltaZ ^ 2 * (-SigmaScatter12 * Flux1(i, j) / DiffusionCoefficient2)
'neutron/cm^2
    Next i
Next j
End If
'solve block tridiagonal system. See Solution Methods in my thesis.
For w = 1 To 100000
    For i = 1 To n - 1
        e(i, 1) = b(i, 1) - X2(i, 2) 'neutron/cm^2
    
```

```

EE(i) = e(i, 1) 'neutron/cm^2 typical
LL(i) = l(i, 1) 'LL Lower diagonal of tridiagonal;no units typical
AA(i) = A(i, 1) 'AA main diagonal of tridiagonal; no units typical
UU(i) = u(i, 1) 'UU upper diagonal of tridiagonal; no units typical
Next i
Call Tridiag(n, LL, AA, UU, EE, XX) 'returns Flux1 at row m-1 in core
For i = 1 To n - 1
  X2(i, 1) = XX(i) 'neutrons/cm^2 typical
Next i
For j = 2 To (m - 2)
  For i = 1 To n - 1
    e(i, j) = b(i, j) - X2(i, j - 1) - X2(i, j + 1)
    EE(i) = e(i, j)
    LL(i) = l(i, j)
    AA(i) = A(i, j)
    UU(i) = u(i, j)
  Next i
  Call Tridiag(n, LL, AA, UU, EE, XX)
  For i = 1 To n - 1
    X2(i, j) = XX(i)
  Next i
Next j
For i = 1 To n - 1
  e(i, m - 1) = b(i, m - 1) - X2(i, m - 2)
  EE(i) = e(i, m - 1)
  LL(i) = l(i, m - 1)
  AA(i) = A(i, m - 1)
  UU(i) = u(i, m - 1)
Next i
Call Tridiag(n, LL, AA, UU, EE, XX)
For i = 1 To n - 1
  X2(i, m - 1) = XX(i)
Next i
For i = 0 To m - 1
  X2(0, i) = X2(1, i)
Next i
For i = 0 To n
  Select Case Combo2.ListIndex 'either half or quarter of Rx. Boundaries change.
  Case 0 'Half Rx core
  X2(i, m) = 0 'update X1
  Case 1 'Quarter Rx
  X2(i, m) = X2(i, m - 1) 'update X1
  End Select
  X2(i, 0) = 0
Next i
*****

```

```

For v = 1 To n - 1
  For j = 1 To m - 1
    ErrorX(v, j) = Abs((X2(v, j) - X1(v, j)) / Abs(X2(v, j)))
  Next j
Next v
'Find maximum value of ErrorS()
MaxErrorX = ErrorX(1, 1)
For i = 1 To n - 1
  For j = 1 To m - 1
    X1(i, j) = X2(i, j) 'update X1
    If (ErrorX(i, j) > MaxErrorX) Then
      MaxErrorX = ErrorX(i, j)
    End If
  Next j
Next i
If (Combo3.Text = "Reduced Relative Error; Slower Run Time") Then
  Tolerance = 0.00000001
Else
  Tolerance = 0.001
End If
If (MaxErrorX < Tolerance) Then
  Exit For
End If
*****
'Set boundary conditions
Next w
If (w > 10000) Then
  MsgBox ("Exceeded block tridiagonal maxiterations before convergence")
End If
End Sub

Sub Tridiag(n, LL, AA, UU, EE, XX) 'LL=lower diag, AA=Diag, UU=Upper
diag,EE=A.x
'Crout Factorization for tridiagonal linear systems
!'Back substitute for solution
m = n - 1
ReDim Lower(m)
ReDim Upper(m)
ReDim Z(m)
Lower(1) = AA(1) 'no units
Upper(1) = UU(1) / AA(1) 'no units
Z(1) = EE(1) / Lower(1) 'neutron/cm^2
For i = 2 To m - 1
  Lower(i) = AA(i) - LL(i - 1) * Upper(i - 1) 'no units
  Upper(i) = UU(i) / Lower(i) 'no units

```

```

    Z(i) = (EE(i) - LL(i - 1) * Z(i - 1)) / Lower(i) 'neutron/cm^2
Next
Lower(m) = AA(m) - LL(m - 1) * Upper(m - 1) 'no units
Z(m) = (EE(m) - LL(m - 1) * Z(m - 1)) / Lower(m) 'no units
'!Set solution Flux1 values
XX(m + 1) = 0 'Flux1 at outer boundary; neurton/cm^2
XX(m) = Z(m) 'neutron/cm^2
For i = m - 1 To 1 Step -1
    XX(i) = Z(i) - Upper(i) * XX(i + 1) 'neutron/cm^2
Next
XX(0) = XX(1) '!Set BC dFlux1/dr=0; 'neutron/cm^2
End Sub 'Tridiagonal

```

```

Private Sub Option1_Click()
MsgBox ("Try SigmaA1=0.1532 1/cm")
End Sub

```

```

Private Sub mnu2DOneEnergyGroupItem_Click()
Load k
k.Show
Unload TwoEnergyGroup
End Sub

```

```

Private Sub mnu3DOneEnergyGroupItem_Click()
Load rxfrm
rxfrm.Show
Unload TwoEnergyGroup
End Sub

```

```

Private Sub mnuExitItem_Click()
End
End Sub

```

```

Private Sub mnuHelpItem_Click()
Load Help
Help.Show
End Sub

```

```

Private Sub mnuPrintItem_Click()
TwoEnergyGroup.PrintForm
End Sub

```

```

Private Sub mnuReactorItem_Click()
Load Reactor
Reactor.Show
End Sub

```

```
Private Sub mnuRunItem_Click()  
Call Kinetics  
End Sub
```

```
Private Sub mnuStartFormItem_Click()  
Load ReactorCoreModel  
ReactorCoreModel.Show  
Unload TwoEnergyGroup  
End Sub
```

```
Private Sub OLE1_Updated(Code As Integer)  
OLE1.Visible = True  
End Sub
```

```
Private Sub OLE2_Updated(Code As Integer)  
OLE2.Visible = True  
End Sub
```

Appendix F. Sample Output Charts and Data

The figures below are typical charts provided as output to the user. In addition to the charts, the program saves the data on an Excel worksheet for later use by the program user.

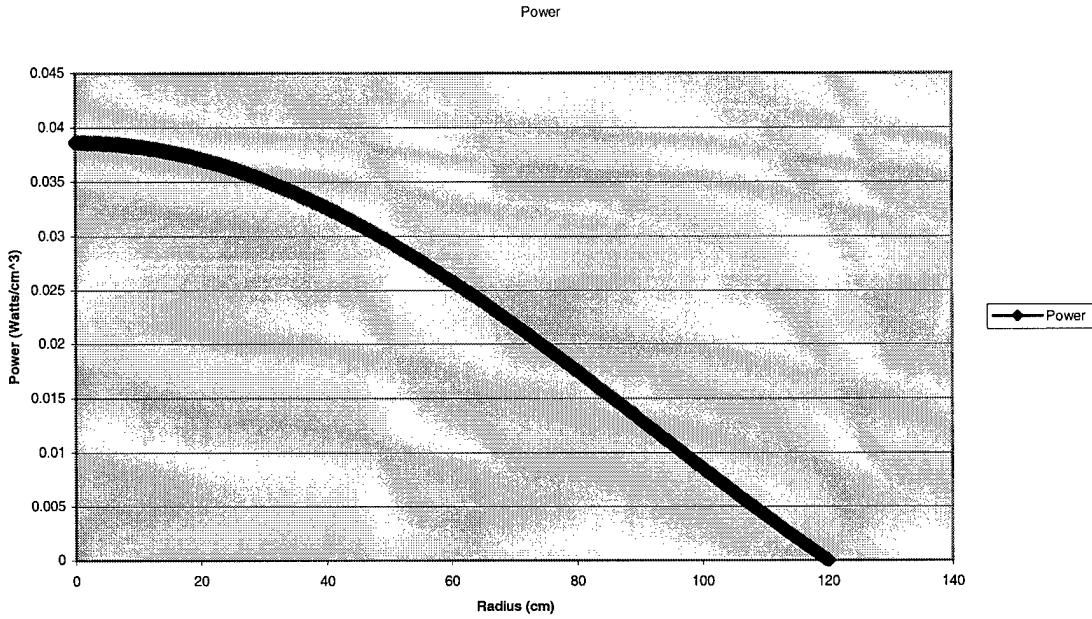


Figure 25 Two Dimensional Output with Mesh Spacing = 0.5 cm

Power Plot of Reactor Core

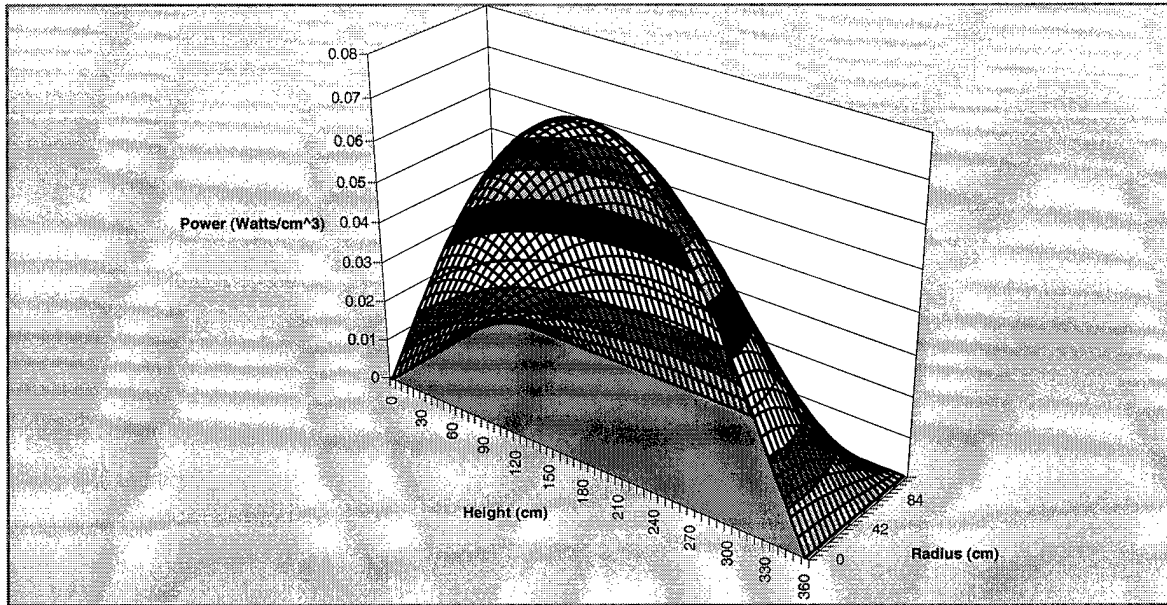


Figure 26 3D, One Energy Group, Half Core Plot with Mesh Spacing = 6 cm

Power Plot of Reactor Core

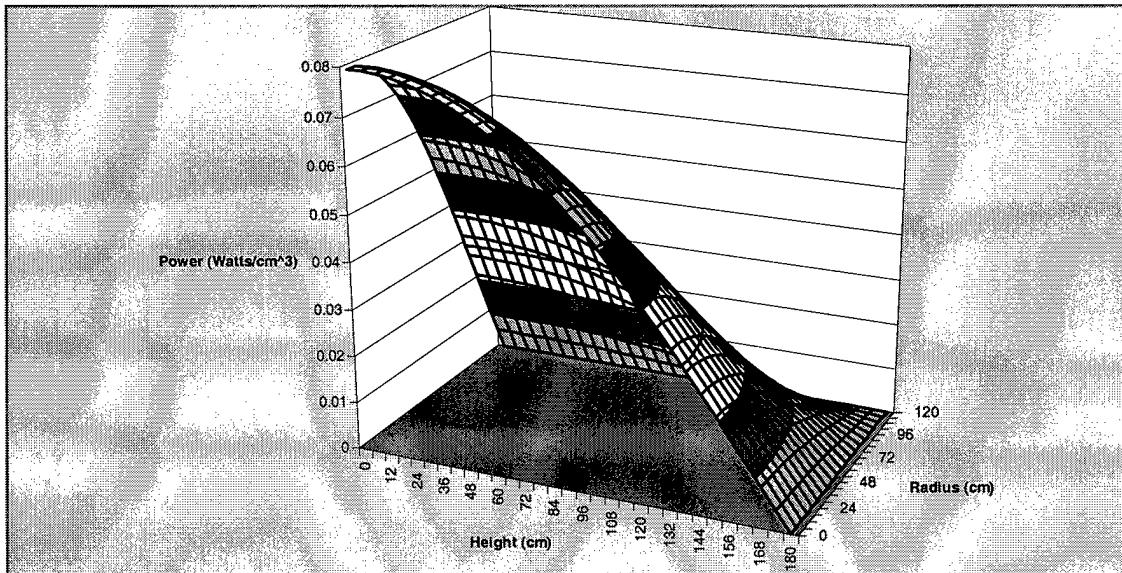


Figure 27 3D, One Energy Group, Quarter Core Plot with Mesh Spacing = 6 cm

Power in Reactor Core

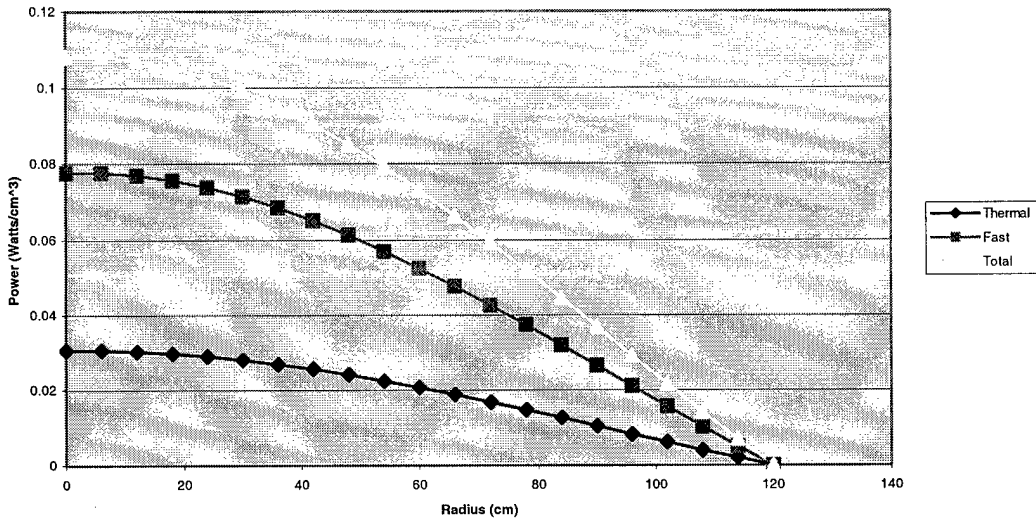


Figure 28 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups

Power in Reactor Core

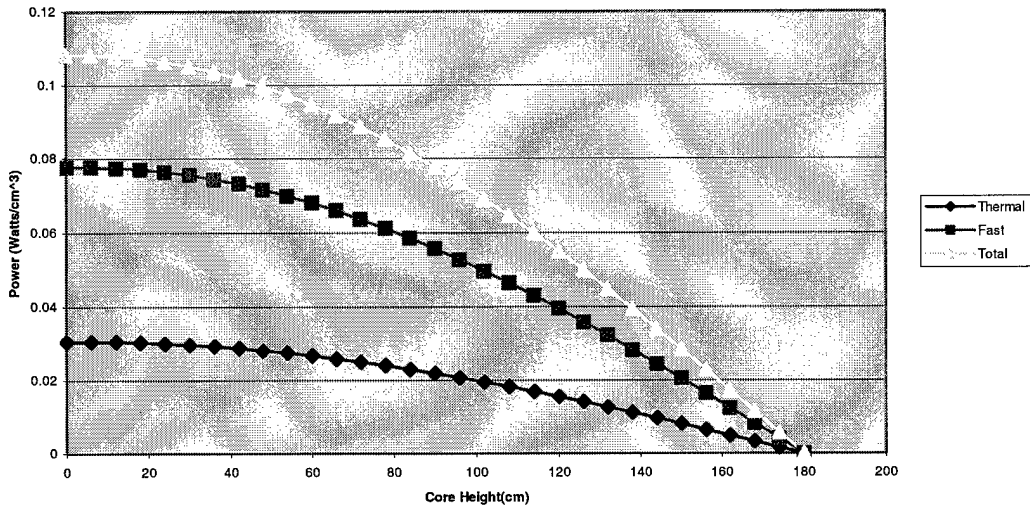


Figure 29 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups

Power in Reactor Core

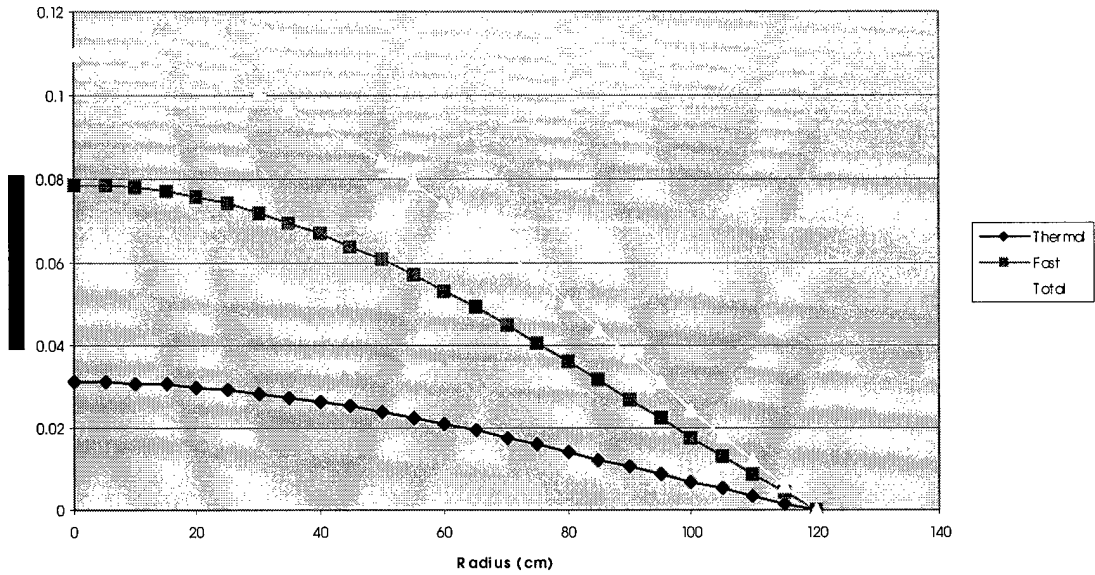


Figure 30 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups Half Rx

Power in Reactor Core

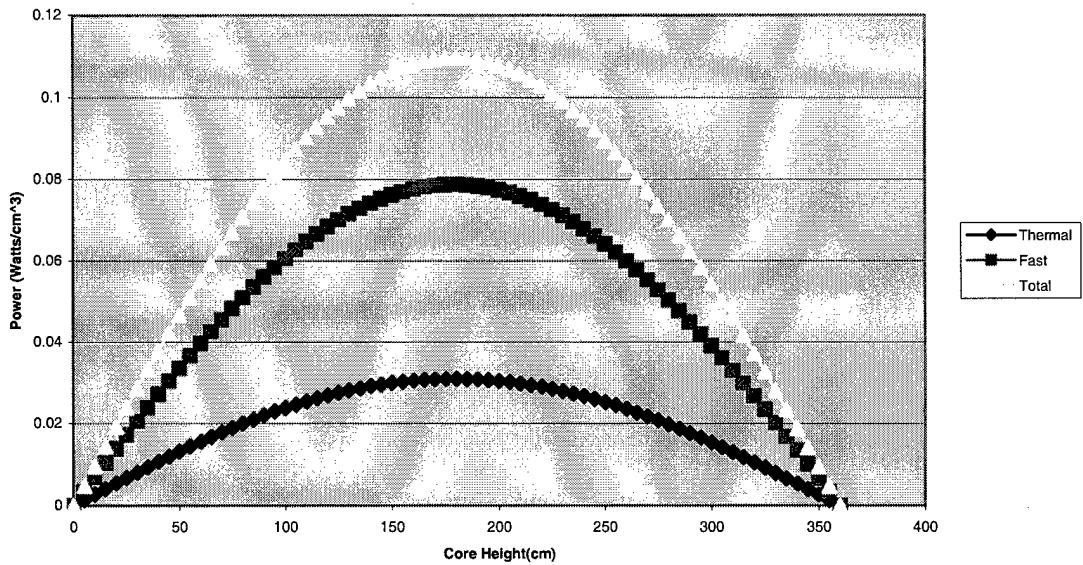


Figure 31 3D, Output with Mesh Spacing = 6 cm, Two Energy Groups Half Rx

Appendix G. Relative Error Plots of Test Cases

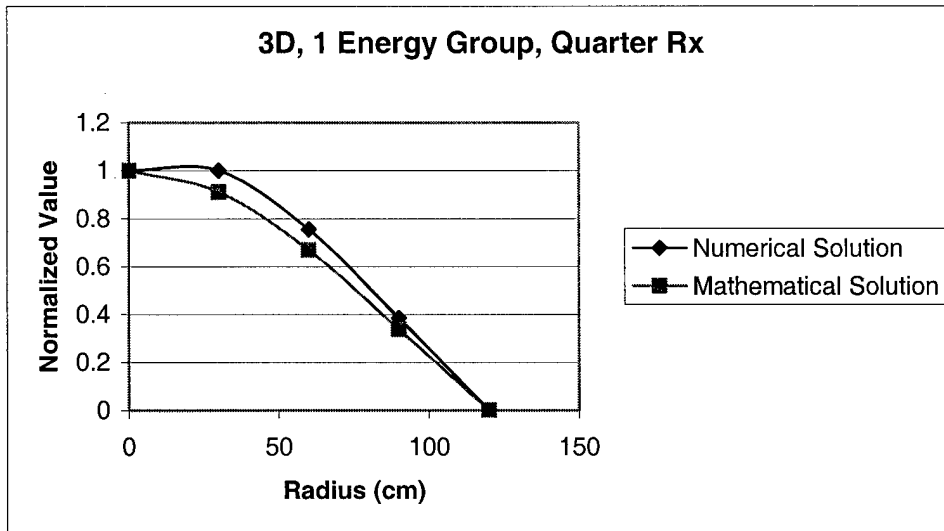


Figure 32 Radial Plot, Mesh Spacing = 30 cm

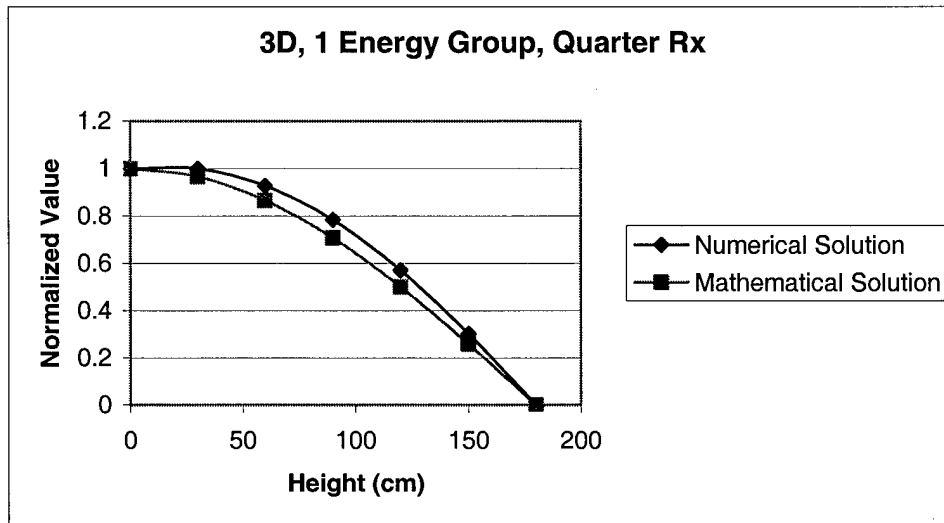


Figure 33 Axial Plot, Mesh Spacing = 30 cm

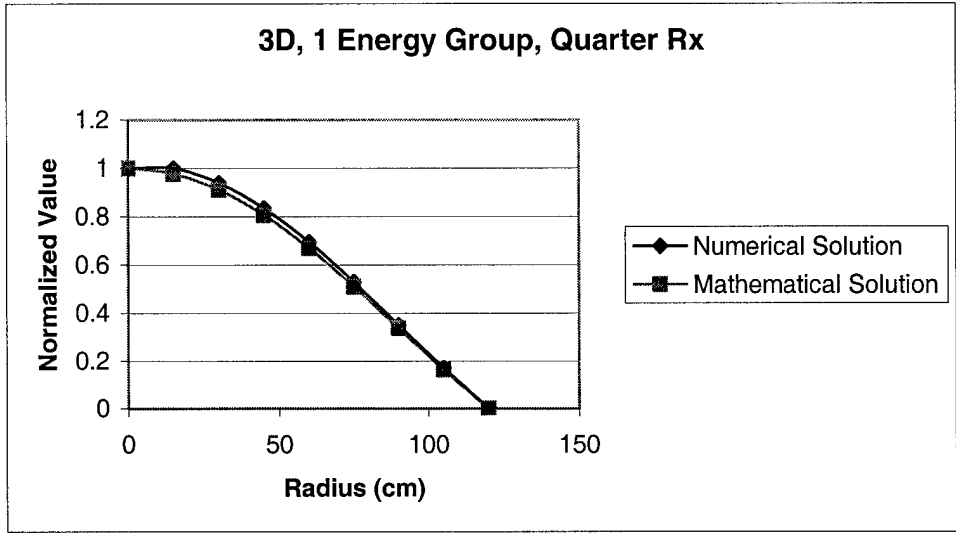


Figure 34 Radial Plot, Mesh Spacing = 15 cm

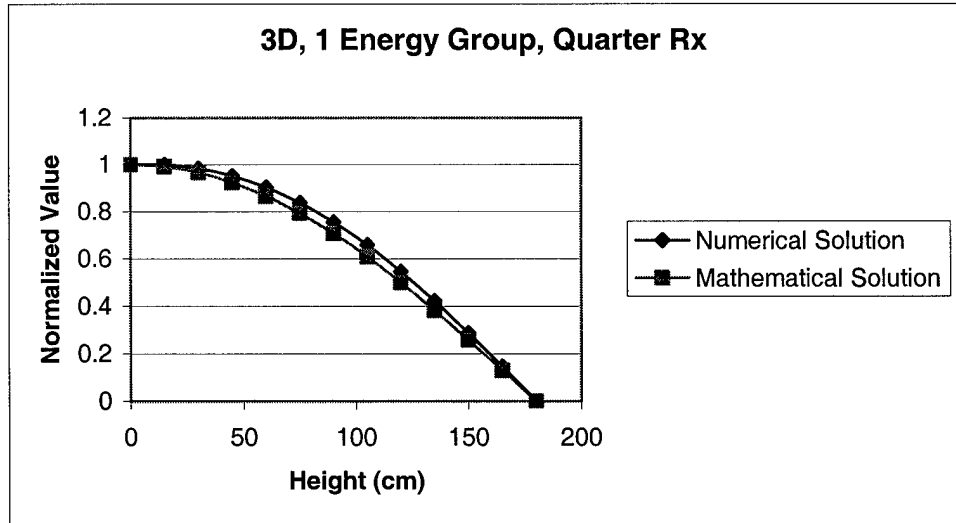


Figure 35 Axial Plot, Mesh Spacing = 15 cm

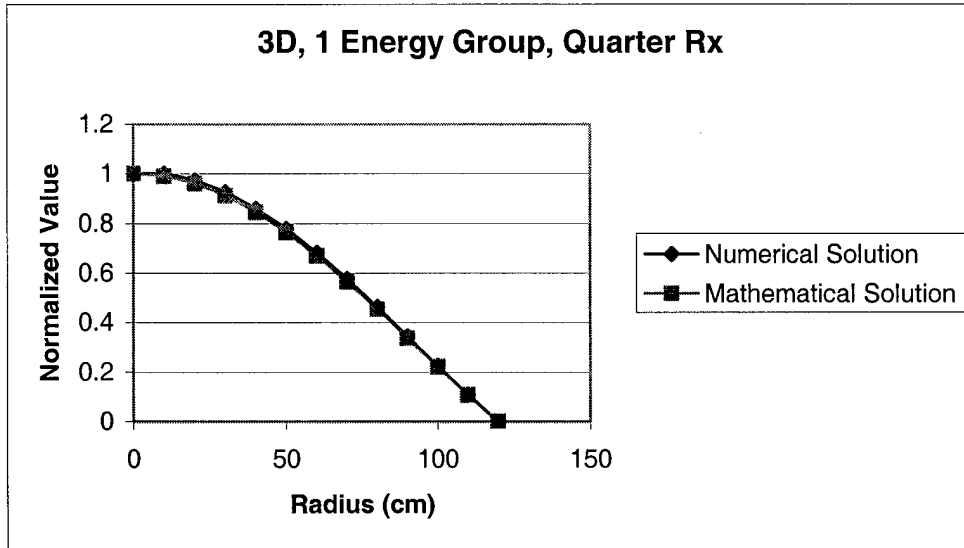


Figure 36 Radial Plot, Mesh Spacing = 10 cm

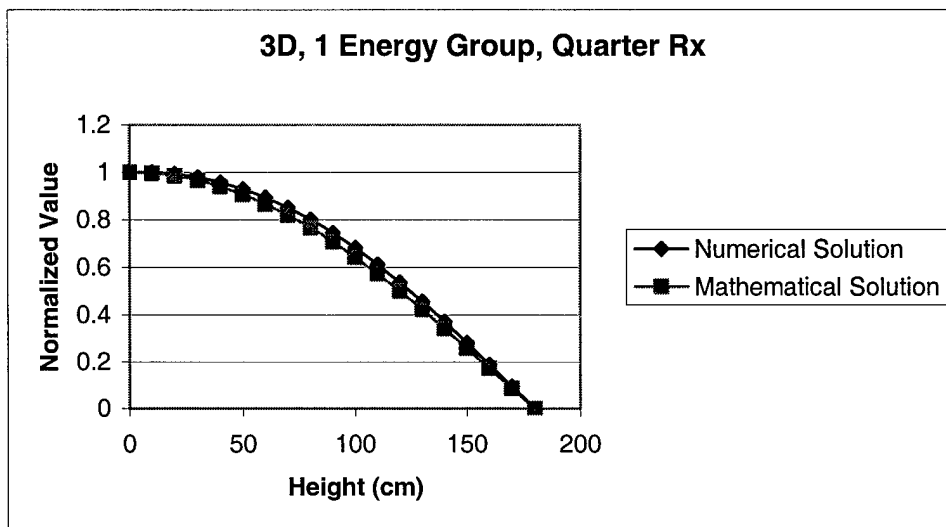


Figure 37 Axial Plot, Mesh Spacing = 10 cm

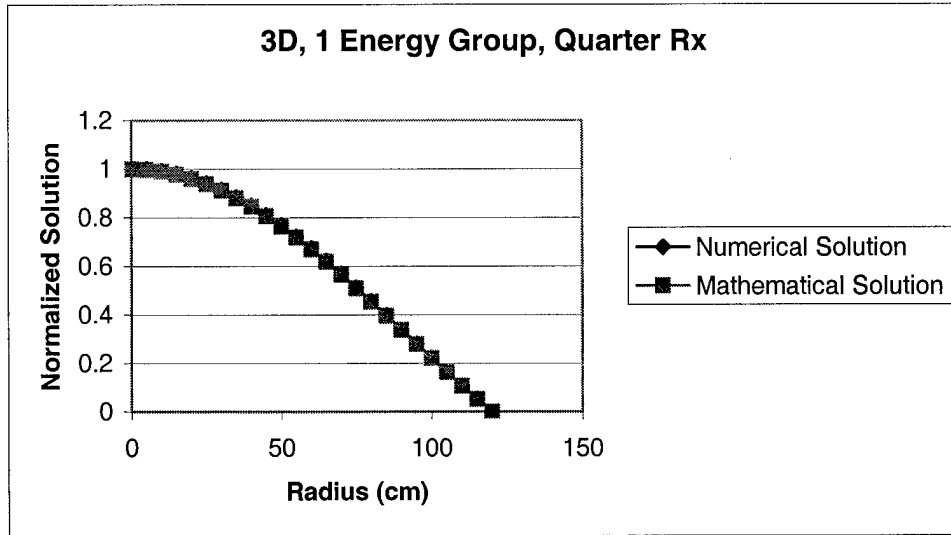


Figure 38 Radial Plot, Mesh Spacing = 5 cm

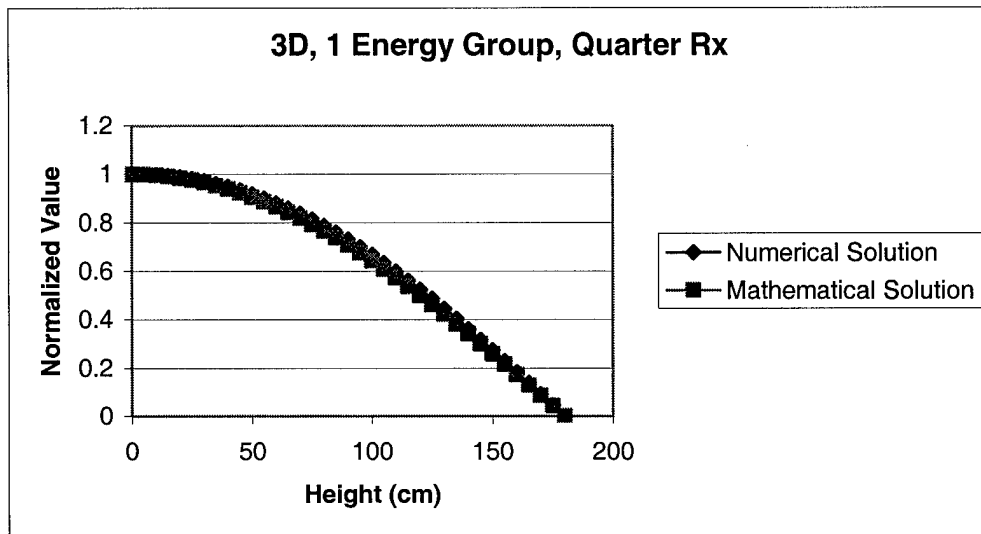


Figure 39 Axial Plot, Mesh Spacing = 5 cm

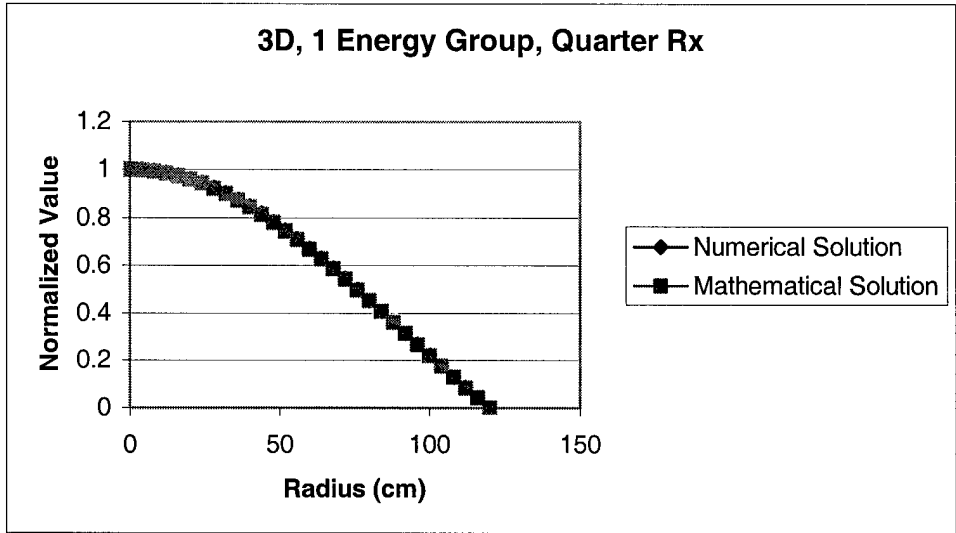


Figure 40 Radial Plot, Mesh Spacing = 4 cm

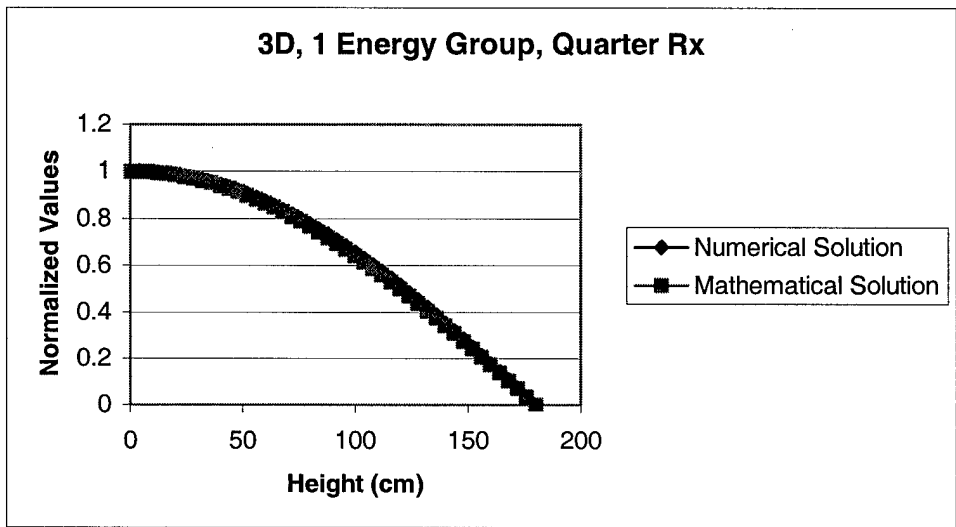


Figure 41 Axial Plot, Mesh Spacing = 4 cm

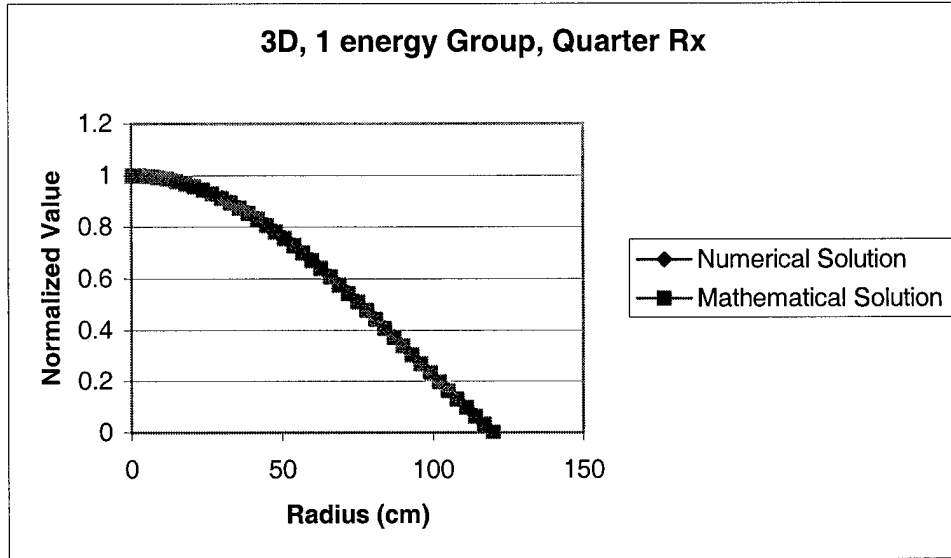


Figure 42 Radial Plot, Mesh Spacing = 3 cm

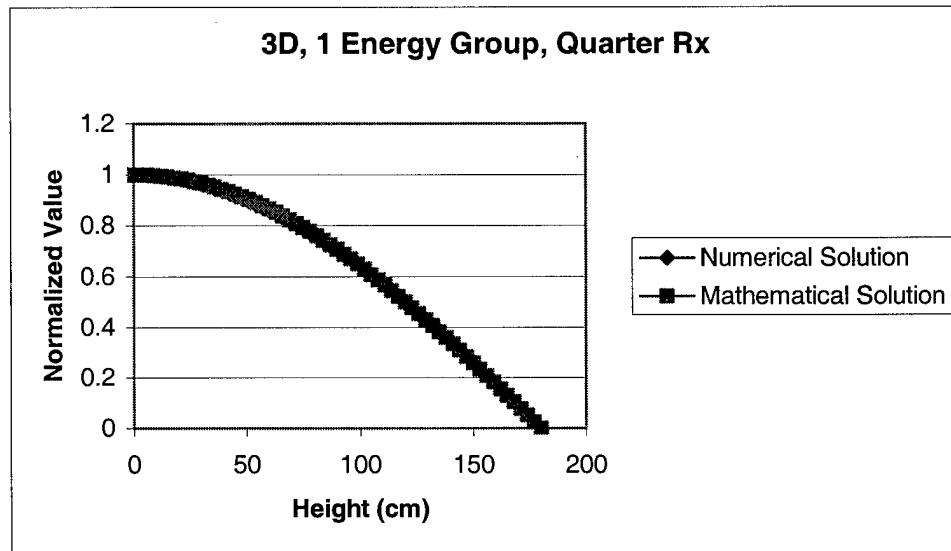


Figure 43 Axial Plot, Mesh Spacing = 3 cm

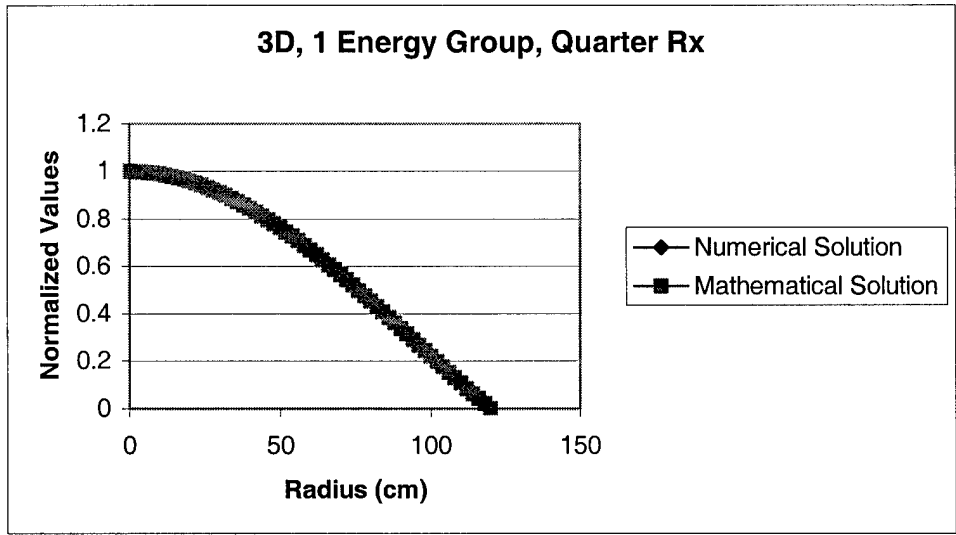


Figure 44 Radial Plot, Mesh Spacing = 2 cm

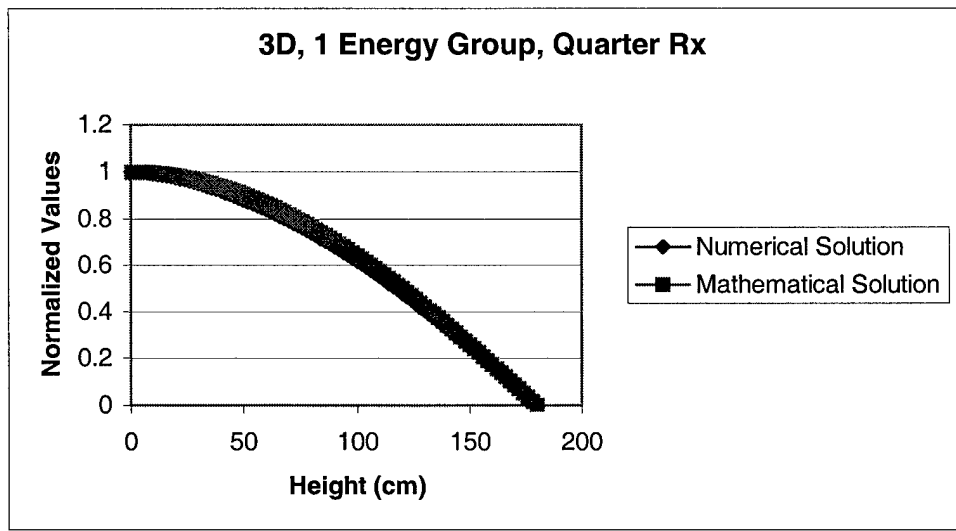


Figure 45 Axial Plot, Mesh Spacing = 2 cm

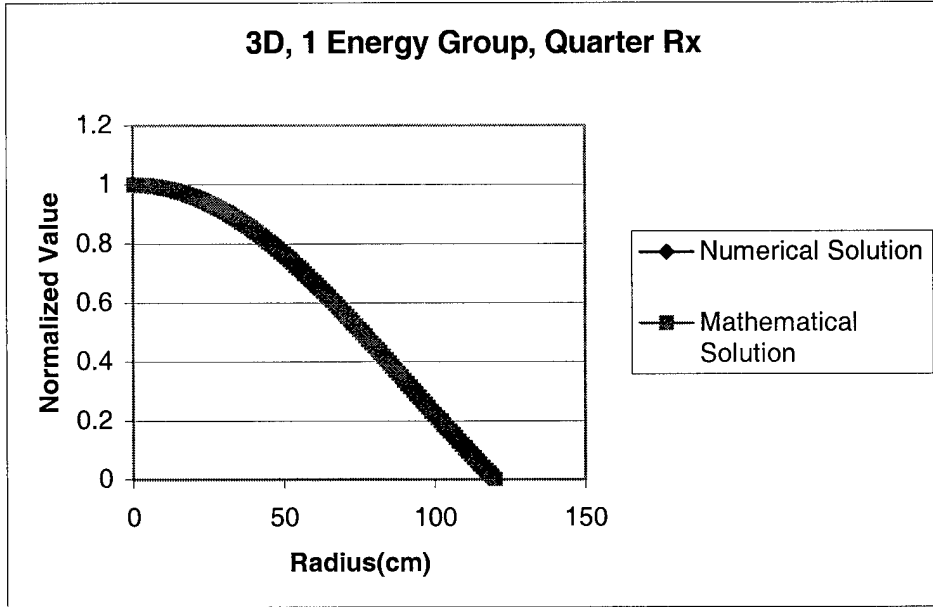


Figure 46 Radial Plot, Mesh Spacing = 1 cm

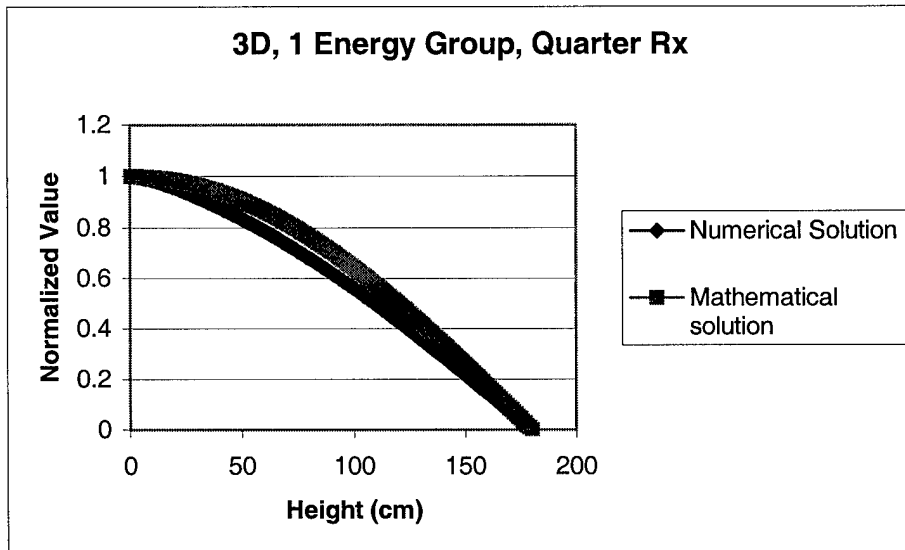


Figure 47 Axial Plot, Mesh Spacing = 1 cm

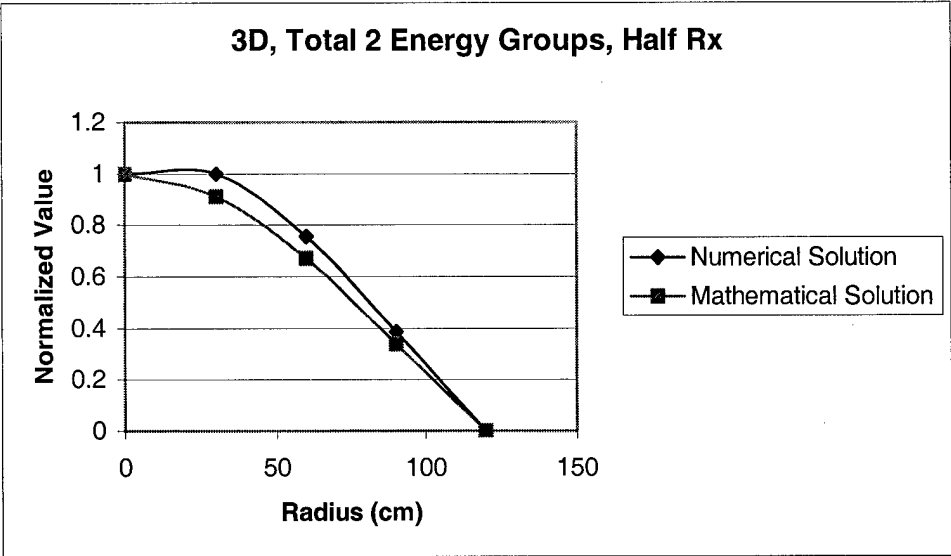


Figure 48 Radial Plot, Mesh Spacing = 30 cm

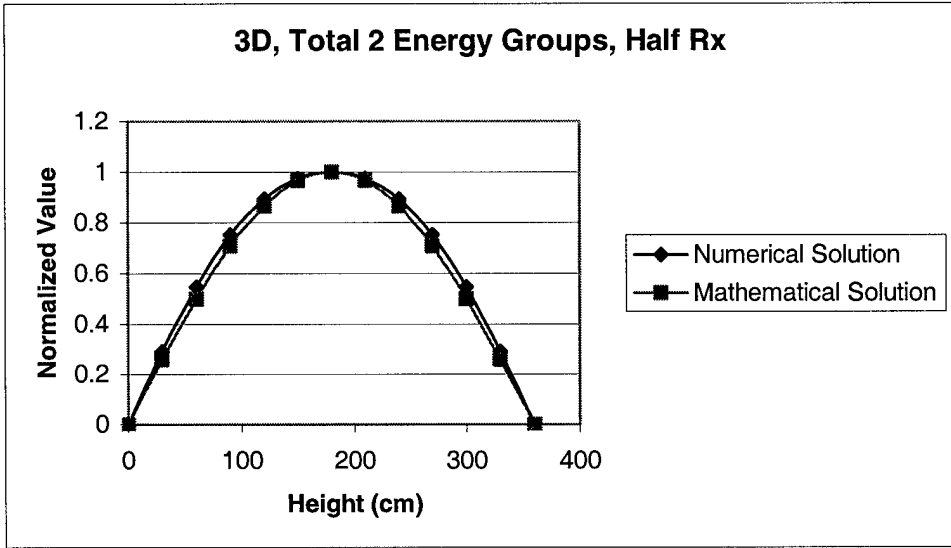


Figure 49 Axial Plot, Mesh Spacing = 30

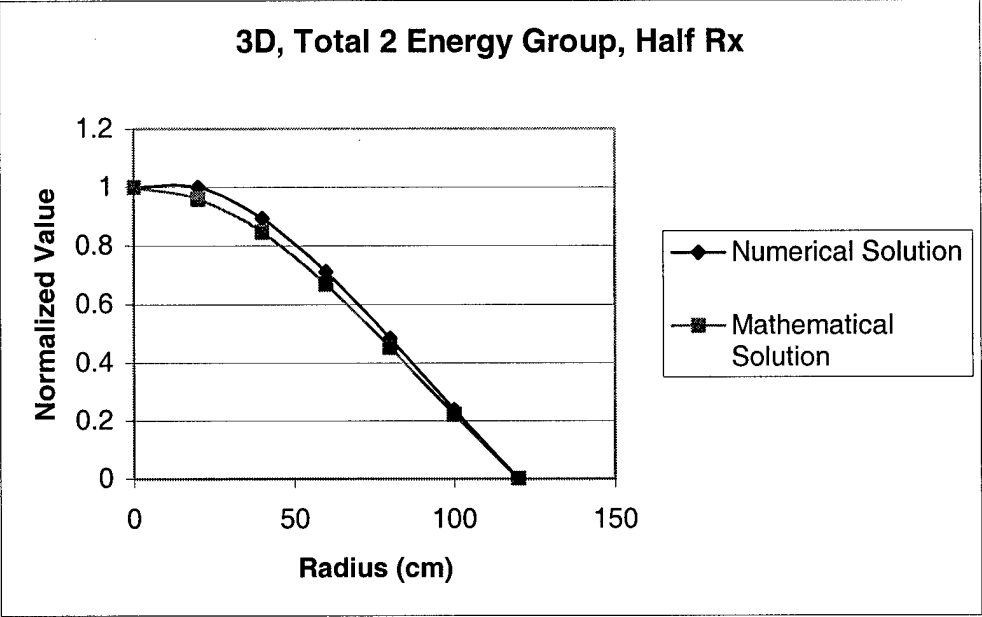


Figure 50 Radial Plot, Mesh Spacing = 20 cm

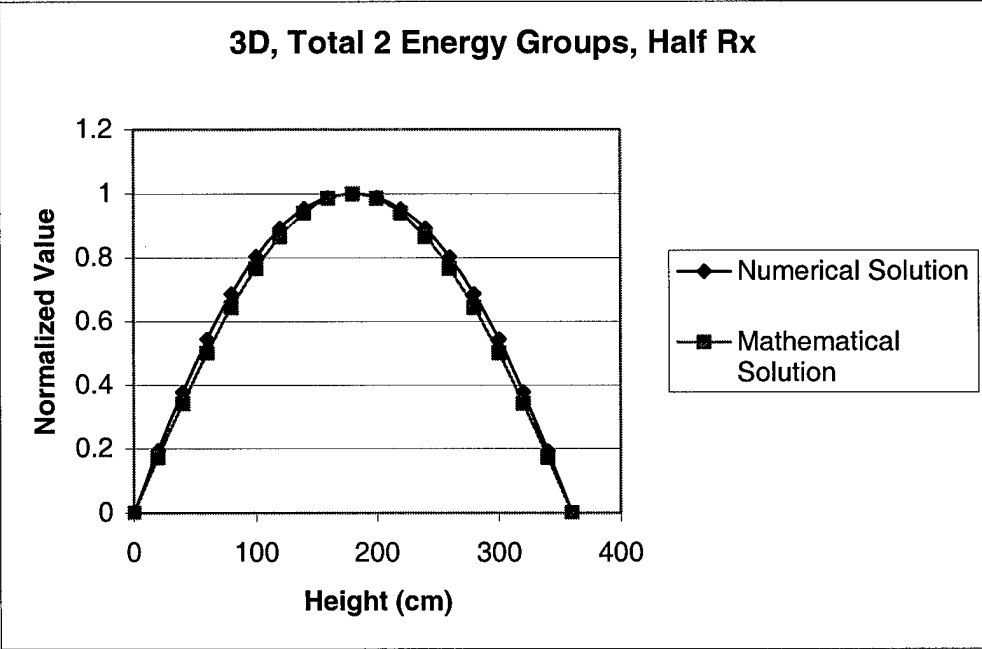


Figure 51 Axial Plot, Mesh Spacing =20

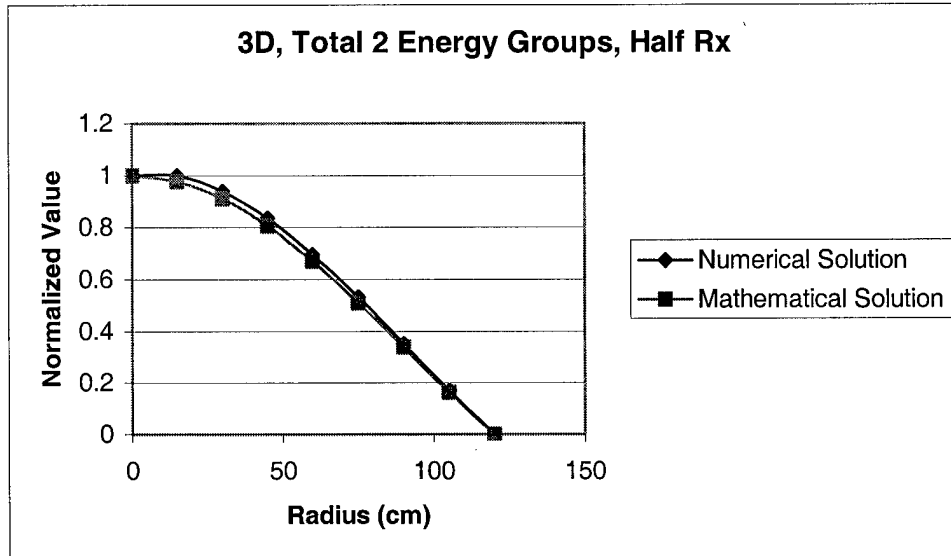


Figure 52 Radial Plot, Mesh Spacing = 15 cm

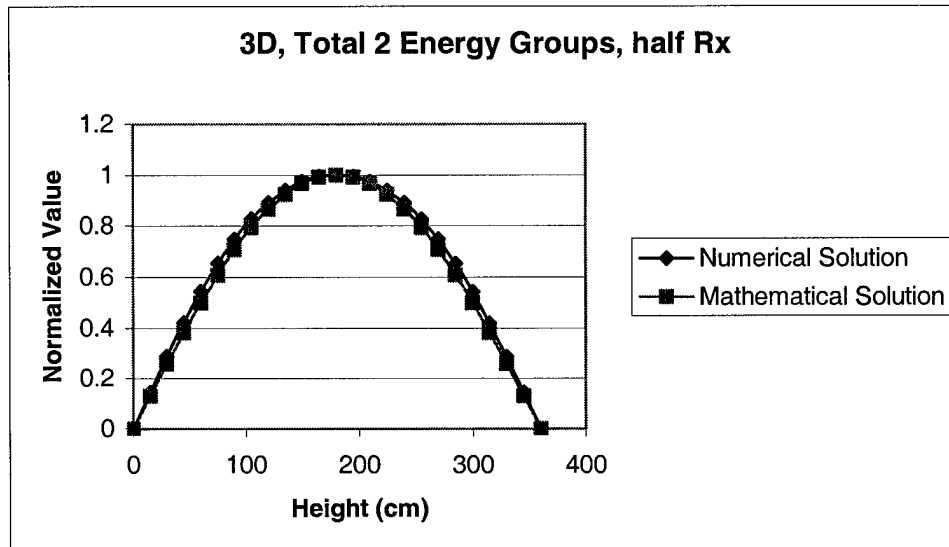


Figure 53 Axial Plot, Mesh Spacing = 15 cm

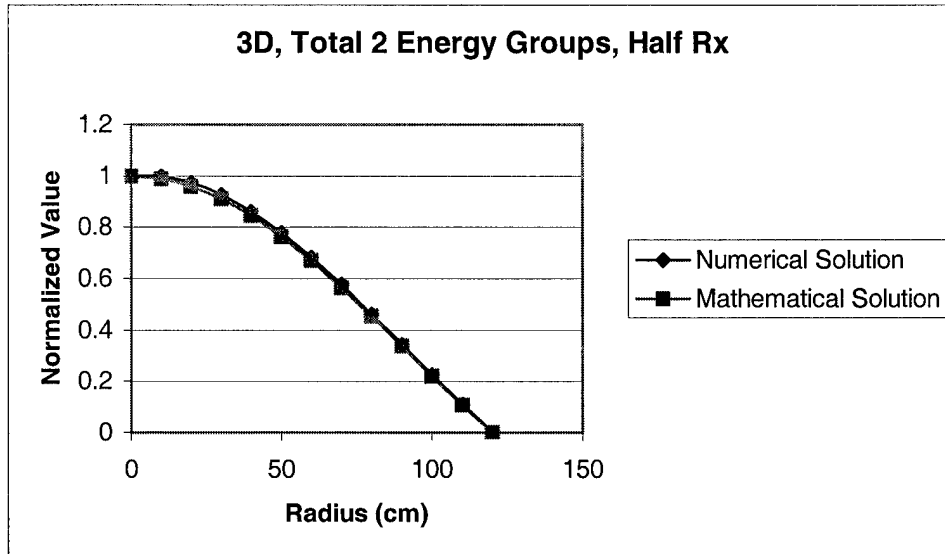


Figure 54 Radial Plot, Mesh Spacing = 10 cm

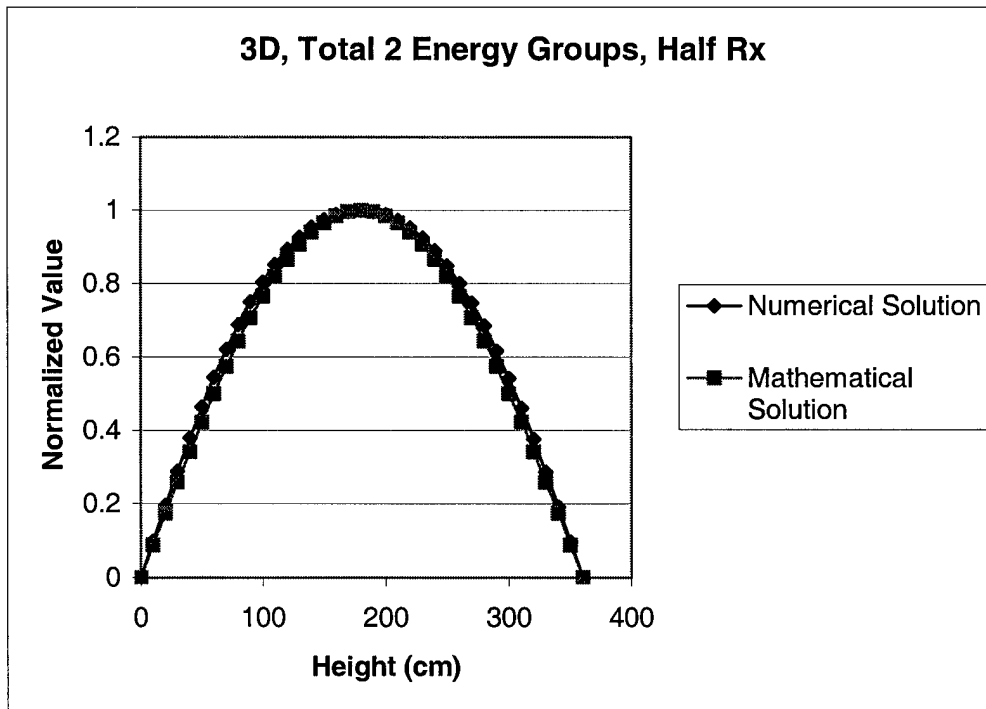


Figure 55 Axial Plot, Mesh Spacing = 10 cm

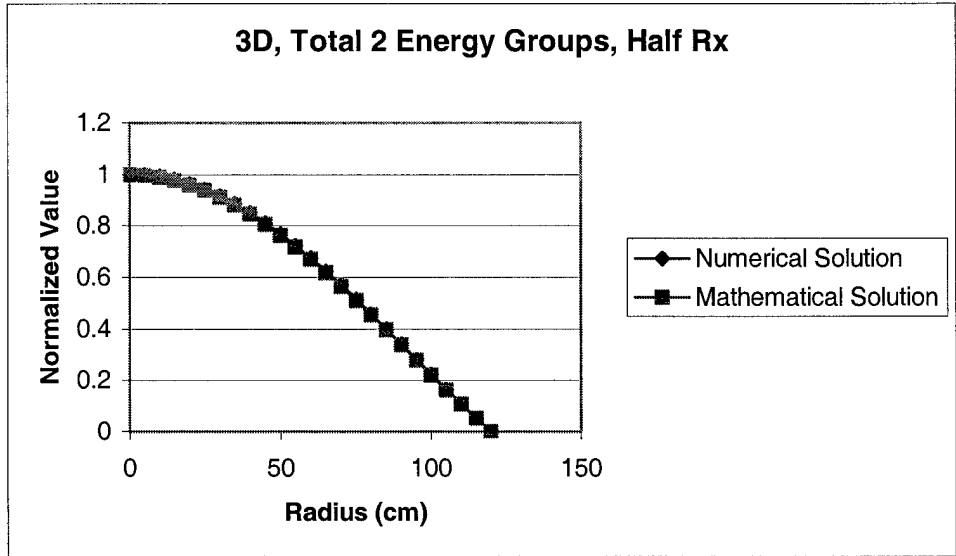


Figure 56 Radial Plot, Mesh Spacing = 5 cm

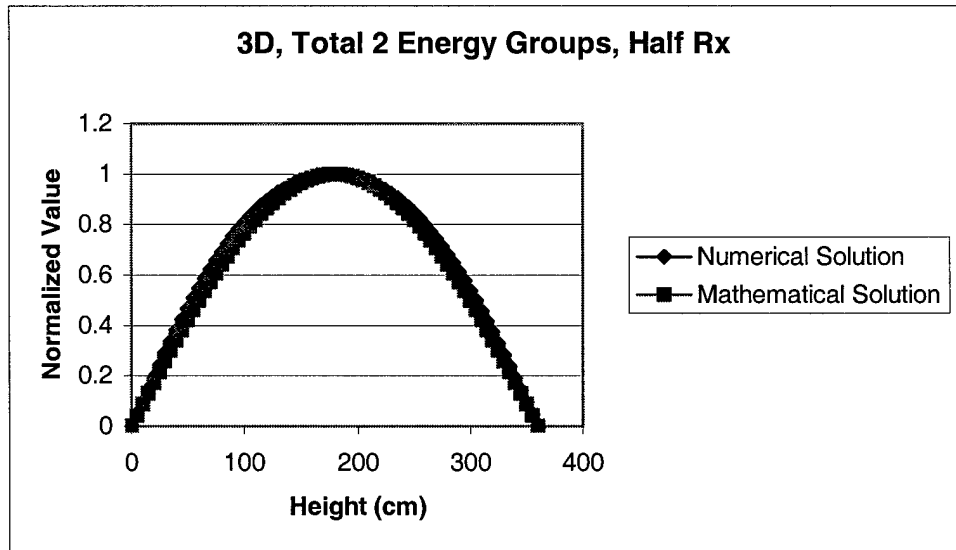


Figure 57 Axial Plot, Mesh Spacing = 5 cm

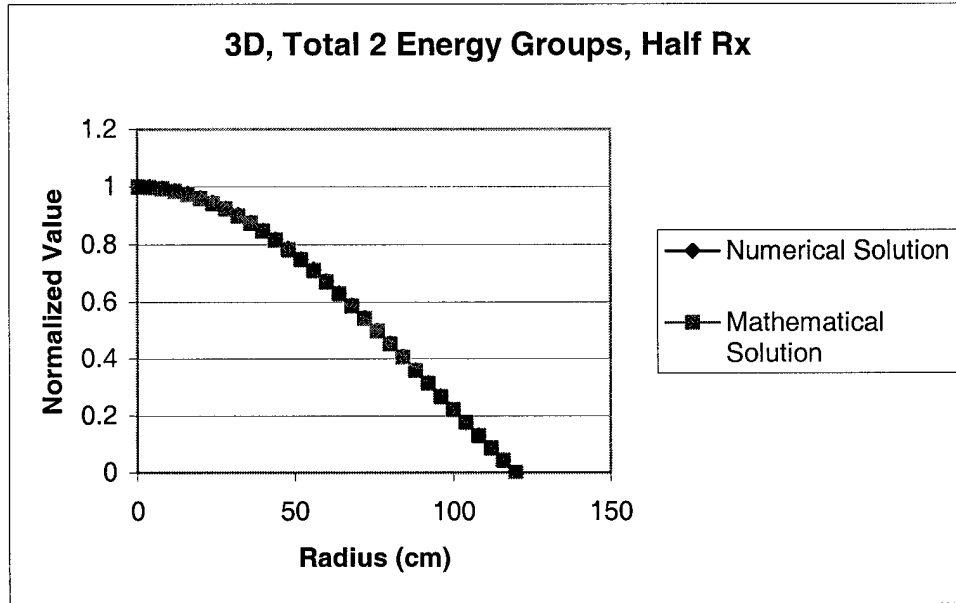


Figure 58 Radial Plot, Mesh Spacing = 4 cm

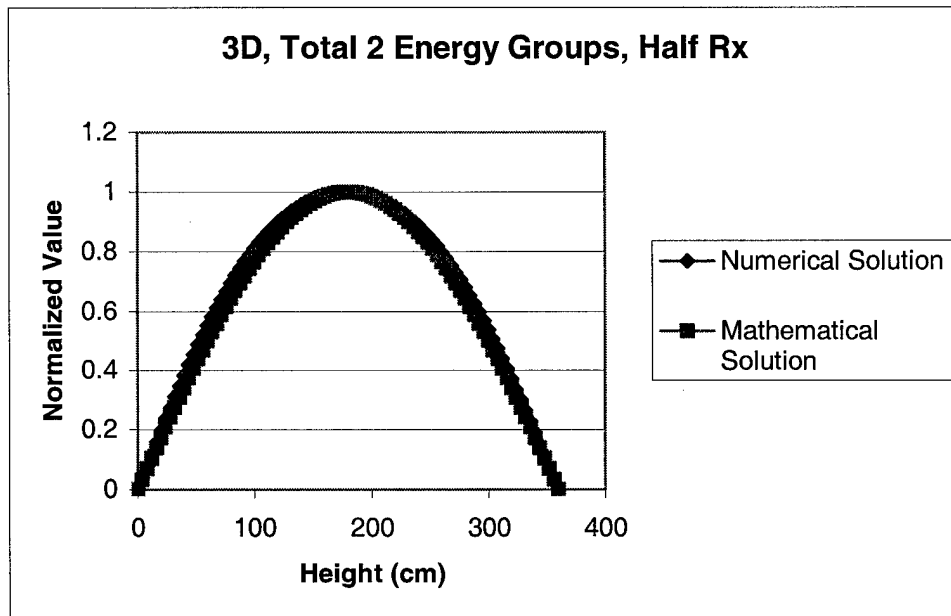


Figure 59 Axial Plot, Mesh Spacing = 4 cm

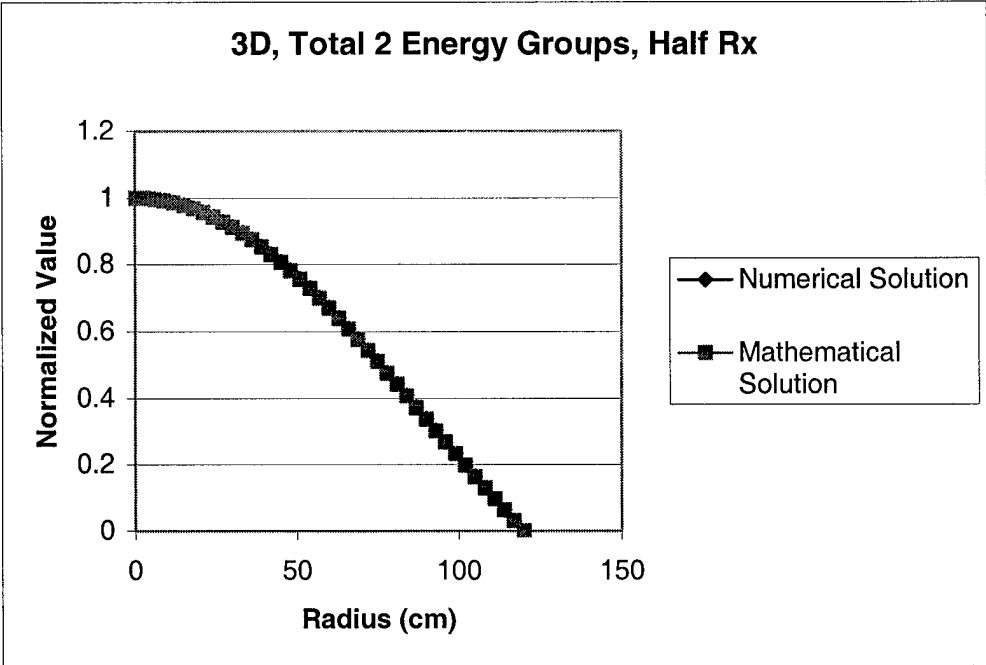


Figure 60 Radial Plot, Mesh Spacing = 3 cm

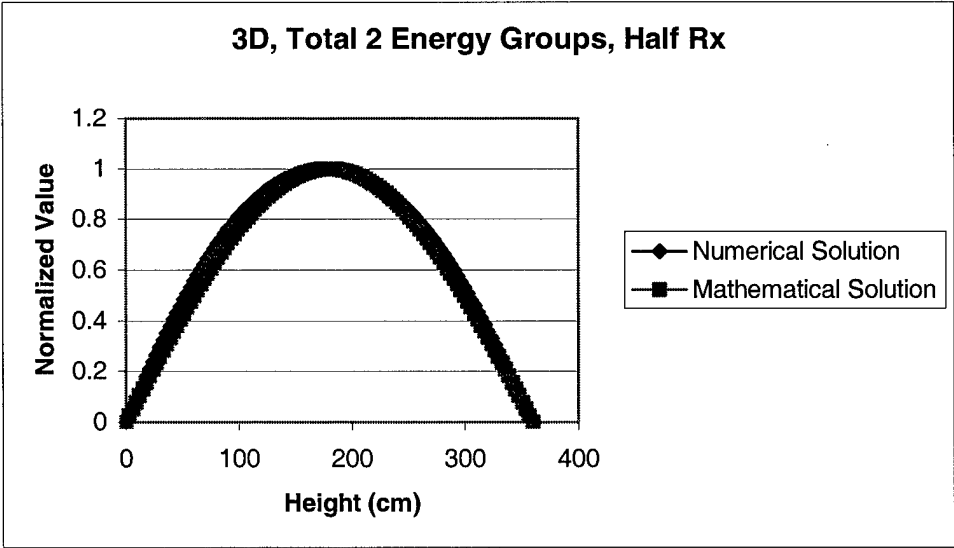


Figure 61 Axial Plot, Mesh Spacing = 3 cm

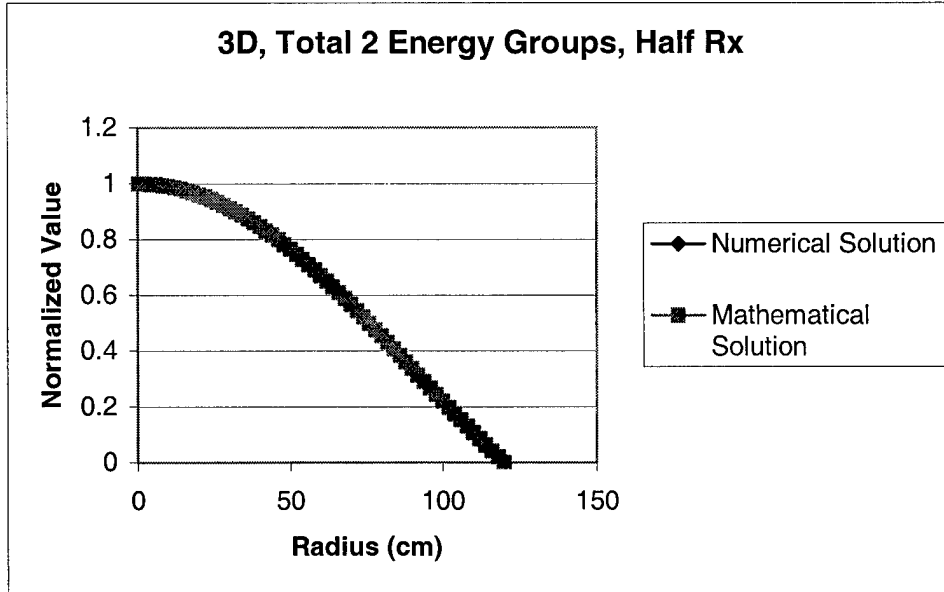


Figure 62 Radial Plot, Mesh Spacing = 2 cm

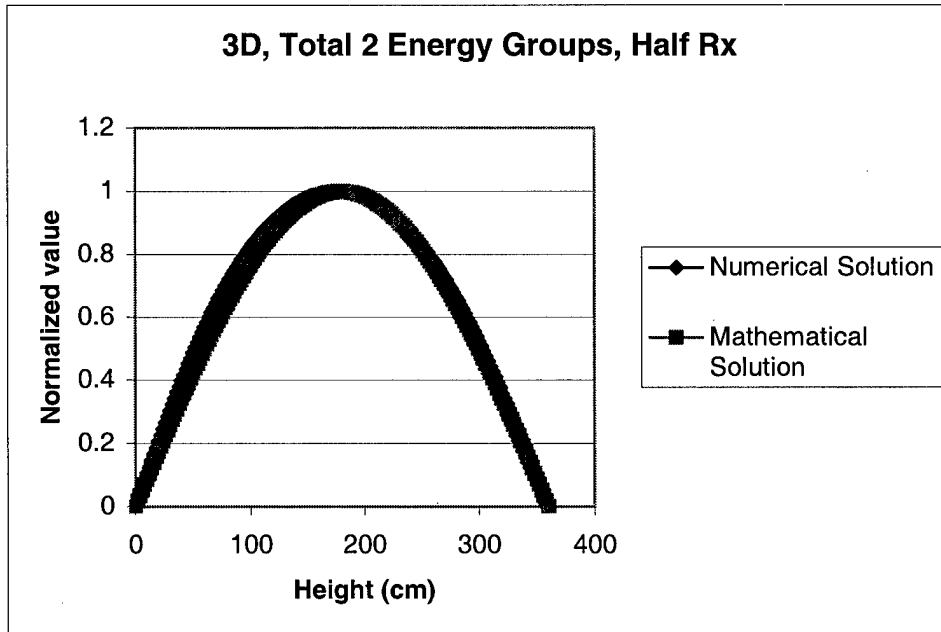


Figure 63 Axial Plot, Mesh Spacing = 2 cm

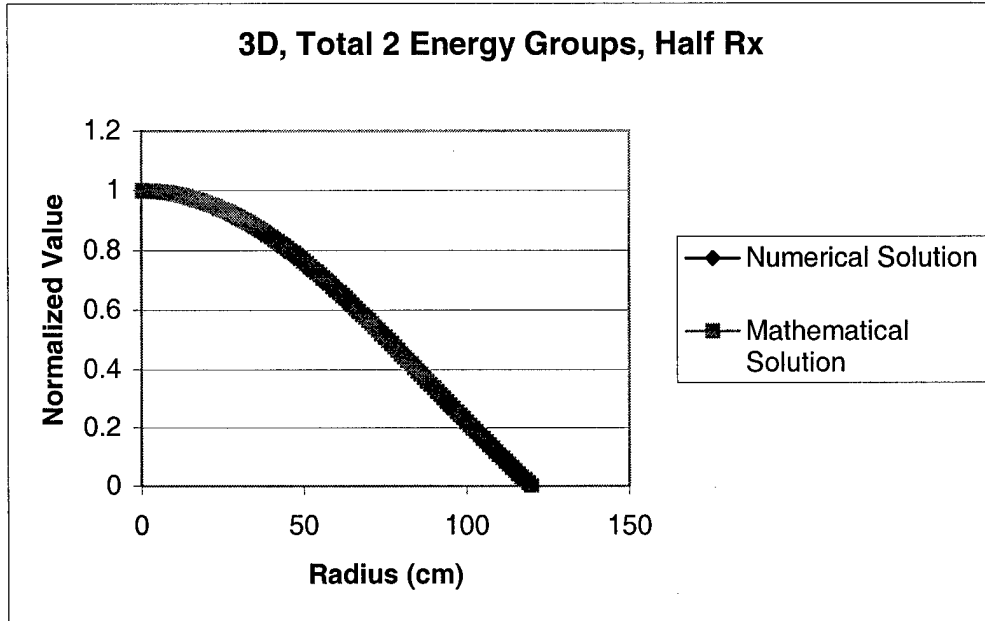


Figure 64 Radial Plot, Mesh Spacing = 1 cm

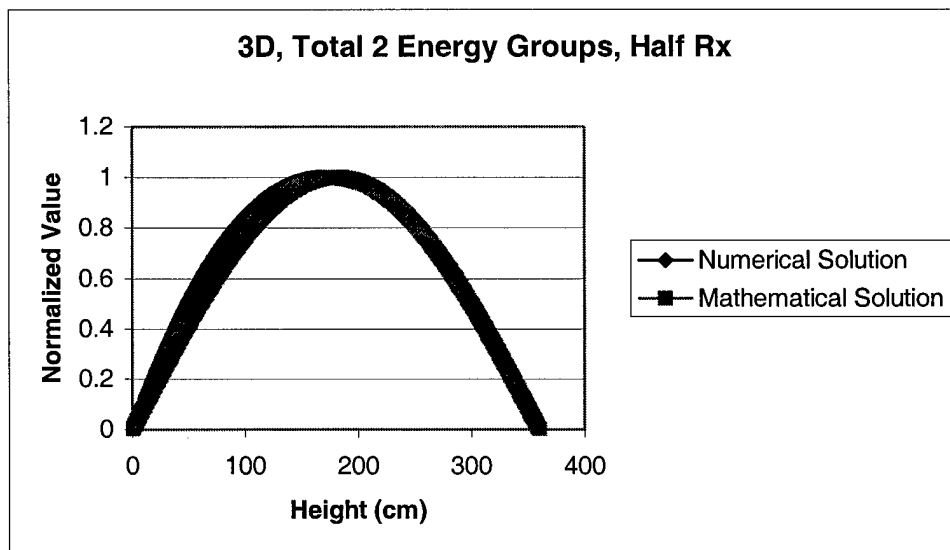


Figure 65 Axial Plot, Mesh Spacing = 1 cm

Appendix H. Linking Visual BASIC and Excel

There are several references available to assist in creating embedded OLE Excel charts within Visual BASIC; however, they do not fully explain how to chart an array of data. This appendix provides the general procedure to chart and embed an OLE Excel object consisting of an array of data as well provide tips to creating more complicated charts and links.

The key to creating an embedded OLE Excel chart is to understand the OLE commands and the Excel application commands. To set the OLE source documentation to link the OLE to the source file, provide the name and location of the Excel file that will contain the data generated by the Visual BASIC code.

```
OLE.SourceDoc = ("Text6")
```

In this example, "Text6" is the TextBox on the Visual BASIC form that the program user uses to input the name and location of the Excel file. `SourceDoc` is a procedure that links the OLE to the source document. To make the OLE object visible on the Visual BASIC Form, use the following code.

```
OLE.Visible = True
```

This should be placed in the code so that the OLE object becomes visible only when desired. Visual BASIC is object oriented. Objects have built-in procedures and settings that allow the programmer to control the functionality of the object. To build the Excel chart, dimension each object as shown below.

```
Dim ExcelApp As Object  
Dim ExcelChart As Object
```

This will allow each of the newly defined objects to have an associated property or method drop down window displaying the available commands in Visual BASIC.

To define the Excel chart type, Visual BASIC must be given the Excel constants instead of the chart name. Some common Excel constants are given in Table 12.

Table 12 Excel Constants for Charts

Chart Type	Excel Constant
xlLine	4
xlColumn	3
xlXYScatter	-4169
xl3DBar	-4099
xl3DSurface	-4103

To create a three dimensional chart, dimension the variable name for the chart type and assign it an Excel constant value.

```
Dim ChartTypeVal As Integer  
ChartTypeVal = -4103
```

Build an Excel Workbook and Worksheet using the following commands.

```
Set ExcelApp = CreateObject("excel.application")  
ExcelApp.Visible = False  
ExcelApp.Workbooks.Add
```

This adds a workbook to Excel and keeps the Excel code running in the background without being visible to the program user. To see Excel run during the Visual BASIC runtime mode, change "false" to "true". This can assist the programmer during debugging because it allows the program user to see how the data is being added the worksheet.

To add data to the Excel Worksheet, use a For-Next loop as shown.

```

For rwIndex = 0 to n
    ExcelApp.Sheets("Sheet1").Cells(rwIndex,colIndex).Value = your data
Next rwindex

```

This adds the data to Worksheet one in the cells corresponding to the Excel (row, column) coordinate system. Once the data is added to the Worksheet it can then be charted and linked to the Visual BASIC OLE. The following sample code selects the data on the sheet, defines the chart type, adds data series and axis labels, saves the chart, and links the chart to the OLE.

```

ExcelApp.Sheets("Sheet1").Range("A1").CurrentRegion.Select
Set ExcelChart1 = ExcelApp.Charts.Add()
ExcelChart1.Type = ChartTypeVal
ExcelChart1.SeriesCollection(1).Name = """"Thermal""""
ExcelChart1.SeriesCollection(2).Name = """"Fast""""
ExcelChart1.SeriesCollection(3).Name = """"Total""""
With ExcelChart1
    .HasTitle = True
    .ChartTitle.Characters.Text = "Power in Reactor Core"
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Radius (cm)"
    .Axes(xlValue, xlPrimary).HasTitle = True
    .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Power (Watts/cm^3)"
End With
ExcelChart1.HasLegend = True
'save chart, activate chart, OLE link chart, OLE update chart for chart 1
ExcelChart1.SaveAs [Text6]
ExcelApp.Sheets("Chart1").Select 'activates the desired chart
OLE1.CreateLink (Text6)         'Creates the link to the name and location
                                'given in TextBox six
OLE1.Update                     'allow immediate update of excel chart on
                                'the Visual BASIC Form
ExcelApp.Quit                   'Quits the application
Set ExcelChart = Nothing        ' Clears the previous settings
Set ExcelApp = Nothing

```

This sample code produced the OLE embedded object shown in Figure 66 directly on the Visual BASIC Form. To open Excel and access the chart and data, double click the OLE on the Visual BASIC Form.

Power Plot of Reactor Core

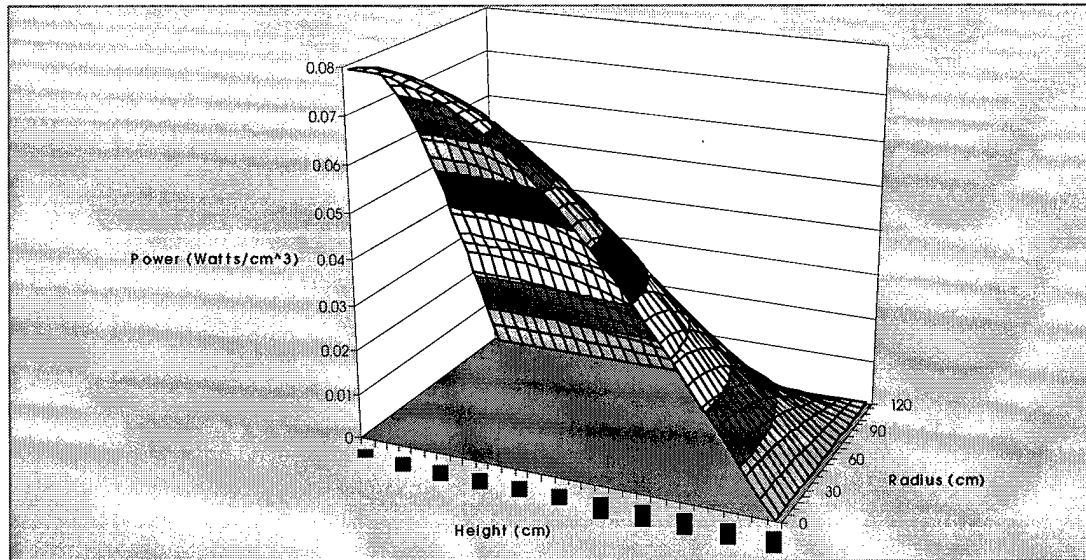


Figure 66 Sample OLE Embedded Object

Excel contains Visual BASIC with Applications that is a limited version of Visual BASIC. Although the Visual BASIC with Applications will not run many Visual BASIC commands, it can assist in developing Visual Basic code needed to create embedded charts. For example, to develop sample Visual BASIC codes use the macro command while in Excel to record the steps in building an Excel chart and then display the code using Visual BASIC with Applications. This will provide the general coding language to develop variations to Excel charts. In some cases, the sample code displayed in Visual BASIC with Applications can be copied directly into Visual BASIC.

Bibliography

Aitken, Peter G., *Developing Solutions with Office 2000 Components and VBA*, Prentice-Hall, Inc., 2000

Burden, Richard L. and Faires, Douglas J., *Numerical Analysis*, Brooks/Cole Publishing Company, 1997

Clark, Melville and Hansen, Kent F., *Numerical Methods of Reactor Analysis*, Academic Press, 1964

Duderstadt, James J. and Hamilton, Louis J., *Nuclear Reactor Analysis*, John Wiley and Sons, Inc., 1976

Feltus, M.A., *Nuclear Engineering 431 Class Notes*, Penn State Univeristy, Spring 1995

Glasstone, Samuel and Sesonske, Alexandria, *Nuclear Reactor Engineering*, Van Nostrand Reinhold Company, 1981

Halvorson, Michael, *Microsoft Visual Basic 6.0 Professional Step by Step*, Microsoft Press, 1998

Henry, Allan, *Nuclear-Reactor Analysis*, MIT Press, 1986

Jedruch, Jacek, *Nuclear Engineering Data Bases, Standards, and Numerical Analysis*,
Van Nostrand Reinhold Company, 1985

Knief, Ronald, *Nuclear Engineering: Theory and Technology of Commercial Nuclear
Power*. Washington, D.C.: Taylor and Francis, 1992

Larmarsh, John R. *Introduction to Nuclear Engineering*, Addison-Wesley Publishing
Company, November 1982

Ott, Karl O. and Neuhold, Robert J., *Introductory Nuclear Reactor Dynamics*, American
Nuclear Society, 1985

Ott, Karl O. and Bezella, Winfred A., *Introductory Nuclear Reactor Statics*, American
Nuclear Society, 1989

Press, William H. et al, *Numerical Recipes in FORTRAN 77: The Art of Scientific
Computing*, Cambridge University Press, 1996

Vita

Major William H. Harman entered undergraduate studies at Old Dominion University where he graduated with a Bachelor of Science degree in Civil Engineering Technology and was commissioned in the U.S. Army in December 1985. His civilian education also includes an Associate in Applied Science in Architectural Drafting, a Master of Science in Construction, and he is a registered Professional Engineer in California.

Before entering graduate school at the Air Force Institute of Technology, his military career included assignments as a platoon leader and executive officer in Germany, a staff officer and company commander at Fort Bragg, a project engineer in the Sacramento Engineer District, and an engineer trainer and battalion operations officer in a Training Support Brigade. Upon graduation, he will be assigned to the Defense Threat Reduction Agency in Alexandria, Virginia where he and his family will reside.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY) 02-08-2001	2. REPORT TYPE Master's Thesis	3. DATES COVERED (From - To) Jun 2000 - Sep 2001
--	--	--

4. TITLE AND SUBTITLE Modeling Pressurized Water reactor Kinetics	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S) Harman, William H., Major, U.S. Army	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GNE/ENP/01M-03
---	--

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT A computer model of a pressurized water reactor (PWR) was developed for use as a teaching tool in graduate level nuclear reactor courses. The development, based on the diffusion equation, includes the methodology for solving the steady state spatial dependence of the neutron power output in a homogeneous right circular cylinder unreflected PWR system. This includes a two dimensional one energy group model, a three dimensional one energy group model, and a three dimensional two energy group model. To solve the homogeneous diffusion equation, a method was developed to search for criticality of the reactor based on the geometry and reactor core material composition. For the one energy group models, a perturbation technique was developed to assist the program user in modifying the macroscopic absorption coefficient to drive the reactor to criticality. For the three dimensional models, a blocked tridiagonal solver was developed to solve the numerical linear system of equations approximating the diffusion equation. The model was coded using Visual BASIC 5.0™. This provides a platform that is exportable to personal computers and allows direct linkage to Windows based programs. The code automatically charts and displays the power distribution profile using Excel™ and displays the calculated multiplication factor determining criticality.

15. SUBJECT TERMS
Pressurized water reactor, kinetics, Visual Basic, diffusion equation, one energy group, two energy groups, blocked tridiagonal solver, homogeneous, steady state, power output, macroscopic absorption coefficient

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	136	LTC James C. Petrosky, ENP
U	U	U			19b. TELEPHONE NUMBER (Include area code) (937) 255-3636, ext 4600

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

	<i>Form Approved</i> <i>OMB No. 074-0188</i>
--	---