THE DEVELOPMENT OF A THREE-DIMENSIONAL NUCLEAR REACTOR KINETICS METHODOLOGY BASED ON THE METHOD OF CHARACTERISTICS

A Thesis in
Nuclear Engineering

By
James Bryce Taylor

© 2007 James Bryce Taylor

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2007
The thesis of James Bryce Taylor was reviewed and approved* by the following:

Anthony J. Baratta  
Professor Emeritus of Nuclear Engineering  
Thesis Advisor  
Co-Chair of Committee

Yousry Y. Azmy  
Professor of Nuclear Engineering  
Co-Chair of Committee

David G. Knott  
Senior Nuclear Engineer, Studsvik Scandpower, Inc.  
Special Member

Kostadin N. Ivanov  
Professor of Nuclear Engineering

Jesse Barlow  
Professor of Computer Science and Engineering

Jack Brenizer  
Professor of Mechanical and Nuclear Engineering  
Chair of the Graduate Program in Nuclear Engineering

* Signatures are on file in the Graduate School
The topic of this work is to develop the theoretical and practical foundations for a time-dependent method of characteristics, for use as an alternative to diffusion theory in a three-dimensional nuclear reactor kinetics computation. As a time-dependent method of characteristics has not been previously attempted, the goal is to demonstrate – affirmatively or negatively – the feasibility of such an approach and identify topics for future investigation. An explicit three-dimensional ray tracing methodology has been developed as a prerequisite task, combining various techniques that have been previously implemented into steady-state lattice physics code packages. In a reactor kinetics computation the problem geometry can be simplified beyond what would be appropriate for an assembly transport calculation, yielding a more efficient set of three-dimensional ray tracing procedures for the method of characteristics than has been previously achievable. Assessment of these spatial techniques is pursued via application to a series of steady state test problems that includes the OECD/NEA Neutron Transport Benchmark Problem. Results demonstrate that practical implementation of a three-dimensional method of characteristics will require a fine spatial discretization, which leads to longer computational runtimes.

Having successfully demonstrated the feasibility of a steady state method of characteristics in three-dimensional space, the primary task of this work is pursued: derivation and implementation of the fundamental equations for a time-dependent method
of characteristics. Solutions to the time-dependent characteristic equation are sought initially via introduction of simple backward and forward time-differencing approximations and assessment is pursued via application to the two-dimensional TWIGL seed/blanket problem. Exorbitant runtimes preclude a complete analysis, however, and instead preliminary results have been generated for special cases of this problem. These results demonstrate that numerical instabilities are inherent to a time-differenced method of characteristics and, moreover, that one particular type of instability is unique to a Lagrangian transport formulation. While this observation suggests that a time-dependent method of characteristics may not be feasible, a more successful formulation could be pursued via future examination of alternative time-discretization methods. The problem of long runtimes, which limits applicability of the methods here described, could be resolved with the eventual introduction of computational acceleration and parallelization techniques.
TABLE OF CONTENTS

LIST OF FIGURES ............................................................................................................ x
LIST OF TABLES ............................................................................................................... xv

CHAPTER 1: INTRODUCTION ....................................................................................... 1

1.1 Review of State-of-the-Art Space-Time Reactor Kinetics Methods .................... 2
  1.1.1 Nodal Diffusion Theory ............................................................................... 4
  1.1.2 The Limitations of Nodal Diffusion Theory .............................................. 8

1.2 Alternative Neutron Transport Methods .............................................................. 12
  1.2.1 Finite-Differenced Discrete Ordinates Methods ......................................... 13
  1.2.2 Spherical Harmonics Methods .................................................................. 18
  1.2.3 Simplified Spherical Harmonics Methods ................................................. 21

1.3 Proposal for the Development of New Methods .................................................... 23

CHAPTER 2: A REVIEW OF PREVIOUS EXPERIENCE WITH THE METHOD OF
CHARACTERISTICS ...................................................................................................... 27

2.1 Physical Origins of the Method of Characteristics .............................................. 28

2.2 The CACTUS Method .......................................................................................... 30
  2.2.1 Advances in Ray Tracing for the Method of Characteristics ..................... 33
  2.2.2 Acceleration of the Characteristics Solver ............................................... 36

2.3 Derivation of the Steady State Method of Characteristics .................................... 38
CHAPTER 3: A RAY TRACING METHODOLOGY FOR EXPLICIT THREE-DIMENSIONAL CARTESIAN GEOMETRY ................................................................. 56

3.1 Discretization of the Spatial and Angular Domains .................................................. 58

3.2 Selection of a Quadrature Set ...................................................................................... 62

3.3 Overview of the 3D Ray Tracing Procedures ............................................................. 65

3.4 Ray Tracing in the Azimuthal Plane ................................................................................. 68

3.4.1 Correction of the Azimuthal Angles and Azimuthal Track Separation ..................... 68

3.4.2 Calculation of Azimuthal Track Entry Locations ......................................................... 74

3.4.3 Calculation of Azimuthal Track Lengths ................................................................ 78

3.4.4 Derivation of Azimuthal Track Coupling Relationships ......................................... 81

3.5 Ray Tracing in the Polar Plane ...................................................................................... 86

3.5.1 Definition of the Periodic Polar Cell ...................................................................... 86

3.5.2 Correction of the Polar Angles and Polar Track Separation ..................................... 90
3.5.3 Calculation of Global Track Lengths in the Periodic Polar Cell .......... 93
3.5.4 Subdivision of Global Track Lengths into Final 3D Track Lengths ........ 98
3.5.4 Derivation of Polar Track Coupling Relationships......................... 101

CHAPTER 4: DERIVATION OF THE TIME-DEPENDENT METHOD OF
CHARACTERISTICS .................................................................................. 102

4.1 The Multigroup Kinetics Equations in Characteristic Form......................... 103
4.2 Review of Time-Dependent Methods..................................................... 109
  4.2.1 The $\Theta$ Method............................................................................ 111
  4.2.2 Alternating Direction Implicit Methods.......................................... 114
  4.2.3 The Stiffness Confinement Method.................................................. 116
  4.2.4 Runge-Kutta Methods...................................................................... 118
  4.2.5 The Improved Quasi-Static Method................................................ 120
4.3 Application of the $\Theta$ Method to the Time-Dependent Method of Characteristics 122
  4.3.1 Forward-Differenced Method of Characteristics.............................. 123
  4.3.2 Backward-Differenced Method of Characteristics........................... 128
  4.3.3 Alternative Formulations for the Delayed Neutron Precursor Equation .... 133

CHAPTER 5: NUMERICAL COMPUTATION OF THE 3D STEADY STATE AND
TIME-DEPENDENT METHODS OF CHARACTERISTICS............................. 135

5.1 Description of the MOCK-3D Code Package...................................... 136
5.2 Implementation of 3D Ray Tracing Methods........................................ 138
5.3 Implementation of the 3D Steady State Method of Characteristics........... 139
5.3.1 The Steady State Power Iteration Method ..................................................... 141
5.3.2 Implementation of Steady State Power Iteration in MOCK-3D ............... 145
5.4 Implementation of the Fully-Explicit Method of Characteristics ..................... 153
  5.4.1 Numerical Solution of the Fully-Explicit Method of Characteristics .............. 155
  5.4.2 Stability of the Fully-Explicit Method of Characteristics ................................ 157
5.5 Implementation of the Fully-Implicit Method of Characteristics ..................... 159
  5.5.1 Numerical Solution of the Fully-Implicit Method of Characteristics .............. 161
  5.5.2 Stability of the Fully-Implicit Method of Characteristics ............................. 166

CHAPTER 6: DEVELOPMENTAL ASSESSMENT OF THE MOCK-3D CODE

PACKAGE ...................................................................................................................... 171
6.1 Developmental Assessment of MOCK-3D ........................................................ 171
  6.1.1 2D Steady State Mini-BWR Assembly Problem ........................................... 172
  6.1.2 2D Steady State MOX/UOX Assembly Interface Problem ............................ 179
  6.1.3 OECD/NEA 3D Neutron Transport Benchmark Problem ............................ 185
    6.1.3.1 Takeda Model 1 Problem Description .................................................... 186
    6.1.3.2 Study Case A: Cell Size ........................................................................ 191
    6.1.3.3 Study Case B: Track Separation ............................................................. 197
    6.1.3.4 Study Case C: Quadrature ...................................................................... 202
    6.1.3.5 Study Case D: Inner Iteration Convergence Criterion ............................. 205
    6.1.3.6 Final Results and Analysis .................................................................... 209
  6.1.3 Developmental Assessment of Mock-3DK .................................................... 219
    6.2.1 2D Time-Dependent TWIGL Seed/Blanket Problem ................................. 221
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Finite-Difference Discretization in 2D Cartesian Geometry</td>
<td>15</td>
</tr>
<tr>
<td>2.1</td>
<td>Spatial Discretization via Characteristics</td>
<td>50</td>
</tr>
<tr>
<td>3.1</td>
<td>Notation for the 3D Cartesian Cell</td>
<td>59</td>
</tr>
<tr>
<td>3.2</td>
<td>Definitions of the Characteristic Direction, Azimuthal and Polar Planes</td>
<td>60</td>
</tr>
<tr>
<td>3.3</td>
<td>Definition of a Level-Symmetric S4 Quadrature Set</td>
<td>63</td>
</tr>
<tr>
<td>3.4</td>
<td>Azimuthal (x-y) Plane of the 3D Cell</td>
<td>66</td>
</tr>
<tr>
<td>3.5</td>
<td>Intersection of Azimuthal and Polar Planes in the 3D Cell</td>
<td>67</td>
</tr>
<tr>
<td>3.6</td>
<td>Cells Spanned by the Periodic Azimuthal Unit in the Azimuthal Plane</td>
<td>69</td>
</tr>
<tr>
<td>3.7</td>
<td>Conservation of Cell Area in the Azimuthal Plane</td>
<td>71</td>
</tr>
<tr>
<td>3.8</td>
<td>Cells Spanned by Multiple Periodic Azimuthal Units in the Azimuthal Plane</td>
<td>73</td>
</tr>
<tr>
<td>3.9</td>
<td>Projection of the Azimuthal Track Spacing to Cell Boundaries</td>
<td>75</td>
</tr>
<tr>
<td>3.10</td>
<td>Example of the Azimuthal Track Origin Calculation Procedure</td>
<td>77</td>
</tr>
<tr>
<td>3.11</td>
<td>Example of the Azimuthal Track Length Calculation Procedure</td>
<td>81</td>
</tr>
<tr>
<td>3.12</td>
<td>Demonstration of Azimuthal Coupling Relationships</td>
<td>82</td>
</tr>
<tr>
<td>3.13</td>
<td>Demonstration of Azimuthal Coupling for Multiple Periodic Units</td>
<td>83</td>
</tr>
<tr>
<td>3.14</td>
<td>Definition of the Periodic Polar Cell</td>
<td>88</td>
</tr>
<tr>
<td>3.15</td>
<td>Definition of the Periodic Polar Cell for Multiple Azimuthal Units</td>
<td>89</td>
</tr>
<tr>
<td>3.16</td>
<td>Cells Spanned by the Periodic Polar Unit in the Polar Plane</td>
<td>90</td>
</tr>
<tr>
<td>3.17</td>
<td>Global Track Lengths for the Periodic Polar Cell</td>
<td>94</td>
</tr>
<tr>
<td>3.18</td>
<td>Global Track Lengths for Multiple Polar Cells</td>
<td>95</td>
</tr>
<tr>
<td>3.19</td>
<td>The Discretized Periodic Polar Cell</td>
<td>96</td>
</tr>
<tr>
<td>3.20</td>
<td>The Discretized Polar Cell with Multiple Periodic Units</td>
<td>97</td>
</tr>
<tr>
<td>3.21</td>
<td>Description of Tracks across the Periodic Polar Cell – Case 1</td>
<td>99</td>
</tr>
<tr>
<td>3.22</td>
<td>Description of Tracks across the Periodic Polar Cell – Case 2</td>
<td>100</td>
</tr>
<tr>
<td>3.23</td>
<td>Description of Tracks across the Periodic Polar Cell – Case 3</td>
<td>100</td>
</tr>
</tbody>
</table>
Figure 5.1: Computational Flow of Steady State Power Iteration ..............................................................144
Figure 5.2: MOCK-3D Track Sweep Procedure ........................................................................................147
Figure 5.3: Computational Flow of Forward-Differenced Method of Characteristics ............................155
Figure 5.4: Computational Flow of Backward-Differenced Method of Characteristics ..........................160
Figure 6.1: Problem Schematic for 2D Mini-BWR Assembly Problem ....................................................174
Figure 6.2: MOCK-3D Normalized Scalar Flux Distribution for Mini-BWR Assembly Calculation ......178
Figure 6.3: MOCK-3D Normalized Fission Rate Distribution for Mini-BWR Assembly Calculation .....178
Figure 6.4: Problem Schematic for 2D MOX/UOX Assembly Interface Problem .................................180
Figure 6.5: MOCK-3D Group 1 Normalized Scalar Flux Distribution for MOX/UOX Assembly Interface Problem .............................................................................................................183
Figure 6.6: MOCK-3D Group 2 Normalized Scalar Flux Distribution for MOX/UOX Assembly Interface Problem .............................................................................................................183
Figure 6.7: MOCK-3D Group 1 Normalized Fission Rate Distribution for MOX/UOX Assembly Interface Problem .............................................................................................................184
Figure 6.8: MOCK-3D Group 2 Normalized Fission Rate Distribution for MOX/UOX Assembly Interface Problem .............................................................................................................184
Figure 6.9: Problem Schematic for Takeda Benchmark Problem, Model 1 ...............................................187
Figure 6.10: MOCK-3D k_{eff} versus Cell Size for Takeda Models 1a and 1b ..............................................194
Figure 6.11: MOCK-3D Rod Worth versus Cell Size for Takeda Models 1a and 1b .................................194
Figure 6.12: MOCK-3D k_{eff} versus Track Separation for Takeda Models 1a and 1b ...............................200
Figure 6.13: MOCK-3D Rod Worth versus Track Separation for Takeda Models 1a and 1b .................201
Figure 6.14: MOCK-3D k_{eff} versus Quadrature for Takeda Models 1a and 1b .......................................204
Figure 6.15: MOCK-3D Rod Worth versus Quadrature for Takeda Models 1a and 1b .........................204
Figure 6.16: MOCK-3D k_{eff} versus Inner Convergence for Takeda Models 1a and 1b .........................208
Figure 6.17: MOCK-3D Rod Worth versus Inner Convergence for Takeda Models 1a and 1b .............208
Figure 6.18: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1a ...................................................................................................212
Figure 6.19: MOCK-3D Group 2 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1a .................................................................213

Figure 6.20: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1b .................................................................213

Figure 6.21: MOCK-3D Group 2 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1b .................................................................214

Figure 6.22: MOCK-3D Group 1 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1a .................................................................214

Figure 6.23: MOCK-3D Group 2 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1a .................................................................215

Figure 6.24: MOCK-3D Group 1 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1b .................................................................215

Figure 6.25: MOCK-3D Group 2 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1b .................................................................216

Figure 6.26: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Y-Axis Plane of Symmetry for Takeda Model 1a .................................................................216

Figure 6.27: MOCK-3D Group 2 Normalized Scalar Flux Distribution along Y-Axis Plane of Symmetry for Takeda Model 1a .................................................................217

Figure 6.28: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Y-Axis Plane of Symmetry for Takeda Model 1b .................................................................217

Figure 6.29: MOCK-3D Group 2 Normalized Scalar Flux Distribution along Y-Axis Plane of Symmetry for Takeda Model 1b .................................................................218

Figure 6.30: Problem Schematic for 2D TWIGL Seed/Blanket Problem ...........................................221

Figure 6.31: Axial Detail for the 3D TWIGL Seed/Blanket Problem ......................................................224

Figure 6.32: MOCK-3D $k_{\text{eff}}$ versus Cell Size for 2D TWIGL Steady State Calculation .......................226

Figure 6.33: Total Number of Outer Iterations Required for Convergence of MOCK-3D Calculation for Various Cell Sizes in 2D TWIGL Steady State Calculation .............................227

xii
Figure 6.34: Total CPU Time Required for Convergence of MOCK-3D Calculation for Various Cell
Sizes in 2D TWIGL Steady State Calculation .................................................................228

Figure 6.35: MOCK-3D Group 1 Converged Steady State Scalar Flux Distribution for 2D TWIGL
Problem with 1.0 cm Cell Size ......................................................................................229

Figure 6.36: MOCK-3D Group 2 Converged Steady State Scalar Flux Distribution for 2D TWIGL
Problem with 1.0 cm Cell Size ......................................................................................229

Figure 6.37: MOCK-3D Group 1 Converged Steady State Scalar Flux Distribution for 2D TWIGL
Problem with 8.0 cm Cell Size ......................................................................................230

Figure 6.38: MOCK-3D Group 2 Converged Steady State Scalar Flux Distribution for 2D TWIGL
Problem with 8.0 cm Cell Size ......................................................................................231

Figure 6.39: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 1 ..........................................................234

Figure 6.40: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 2 ..........................................................234

Figure 6.41: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 3 ..........................................................235

Figure 6.42: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 4 ..........................................................235

Figure 6.43: MOCK-3D Converged Eigenvalue for 200 Time Steps of 2D TWIGL Null Transient
Case 2 ...............................................................................................................................236

Figure 6.44: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Null Transient Case 1 ..........................................................237

Figure 6.45: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Null Transient Case 2 ..........................................................238

Figure 6.46: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Null Transient Case 3 ..........................................................238

Figure 6.47: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Null Transient Case 4 ..........................................................239

Figure 6.48: MOCK-3D Extrapolated Power versus Time for 1.0 sec of Problem Time in 2D
TWIGL Problem .................................................................................................................240

Figure 6.49: MOCK-3D Group 1 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Null
Transient Case 2 ..............................................................................................................241

Figure 6.50: MOCK-3D Group 2 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Null
Transient Case 2 ..............................................................................................................241

Figure 6.51: Percent Difference between MOCK-3D Group 1 Scalar Flux Values at $t = 0$
and after 200 Time Steps of 2D TWIGL Null Transient Case 2 ...........................................242
Figure 6.52: Percent Difference between MOCK-3D Group 2 Scalar Flux Values at $t = 0$
and after 200 Time Steps of 2D TWIGL Null Transient Case 2 ..............................243

Figure 6.53: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Step Transient ........246

Figure 6.54: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Ramp Transient .......246

Figure 6.55: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Step Transient .....................247

Figure 6.56: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Ramp Transient ....................248

Figure 6.57: Extrapolated Power versus Time for 1.0 sec of 2D TWIGL Step Transient ...................249

Figure 6.58: MOCK-3D Group 1 Scalar Flux after 200 Time Steps of 2D TWIGL Step Transient ....250

Figure 6.59: MOCK-3D Group 2 Scalar Flux after 200 Time Steps of 2D TWIGL Step Transient .......250

Figure 6.60: Percent Difference between MOCK-3D Group 1 Scalar Flux Values at $t = 0$ and after
200 Time Steps of 2D TWIGL Step Transient .................................................................251

Figure 6.61: Percent Difference between MOCK-3D Group 2 Scalar Flux Values at $t = 0$ and after
200 Time Steps of 2D TWIGL Step Transient .................................................................252

Figure 6.62: MOCK-3D Group 1 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Ramp
Transient ..............................................................................................................................253

Figure 6.63: MOCK-3D Group 2 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Ramp
Transient ..............................................................................................................................253

Figure 6.64: Percent Difference between MOCK-3D Group 1 Scalar Flux Values at $t = 0$ and after
200 Time Steps of 2D TWIGL Ramp Transient .................................................................254

Figure 6.65: Percent Difference between MOCK-3D Group 2 Scalar Flux Values at $t = 0$ and after
200 Time Steps of 2D TWIGL Ramp Transient .................................................................254
LIST OF TABLES

Table 6.1: Material Properties for 2D Mini-BWR Assembly Problem .......................................................... 174
Table 6.2: Reference Normalized Scalar Flux Distribution for 2D Mini-BWR Assembly Problem .......... 174
Table 6.3: Reference Normalized Fission Rate Distribution for 2D Mini-BWR Assembly Problem .......... 175
Table 6.4: MOCK-3D Normalized Scalar Flux Distribution for 2D Mini-BWR Assembly Problem .......... 176
Table 6.5: Percent Difference between MOCK-3D and Reference Scalar Flux Distributions for 2D Mini-BWR Assembly Problem .............................................................................................. 176
Table 6.6: MOCK-3D Normalized Fission Rate Distribution for 2D Mini-BWR Assembly Problem .......... 177
Table 6.7: Percent Difference between MOCK-3D and Reference Fission Rate Distributions for 2D Mini-BWR Assembly Problem .............................................................................................. 177
Table 6.8: Material Properties for 2D MOX/UOX Assembly Interface Problem, MOX Region .......... 182
Table 6.9: Material Properties for 2D MOX/UOX Assembly Interface Problem, UOX Region .......... 182
Table 6.10: MOCK-3D \( k_{\text{eff}} \) for 2D MOX/UOX Interface Problem ......................................................... 182
Table 6.11: Material Properties for Takeda Benchmark Problem, Model 1, Core Region ................. 188
Table 6.12: Material Properties for Takeda Benchmark Problem, Model 1, Reflector Region .............. 188
Table 6.13: Material Properties for Takeda Benchmark Problem, Model 1a, Void Region ............... 188
Table 6.14: Material Properties for Takeda Benchmark Problem, Model 1b, Control Rod Region ....... 188
Table 6.15: Reference \( k_{\text{eff}} \) for Takeda Benchmark Problem, Model 1 ......................................................... 189
Table 6.16: Reference Region-Averaged Scalar Flux for Takeda Benchmark Problem, Model 1a ........ 190
Table 6.17: Reference Region-Averaged Scalar Flux for Takeda Benchmark Problem, Model 1b ........ 190
Table 6.18: Problem Specifications for MOCK-3D Takeda Cell Size Study .............................................. 192
Table 6.19: MOCK-3D \( k_{\text{eff}} \) Results for Takeda Cell Size Study, Cases A1 to A3 ................................. 193
Table 6.20: MOCK-3D \( k_{\text{eff}} \) Results for Takeda Cell Size Study, Cases A4 and A5 ......................... 193
Table 6.21: MOCK-3D \( k_{\text{eff}} \) Results for Takeda Cell Size Study, Cases A6 and A7 ......................... 193
Table 6.22: MOCK-3D Runtime Information for Takeda Cell Size Study ............................................. 196
Table 6.23: Problem Specifications for MOCK-3D Takeda Track Separation Study ............................. 197
CHAPTER 1:
INTRODUCTION

One important class of problems in nuclear engineering to which computational tools must be applied regards the various time- and space-dependent phenomena that can occur within a nuclear reactor core. Such problems occur with time scales of a few seconds or less yet may significantly affect overall reactor operations. The normal motion of control rods during reactor start-up and shutdown procedures, the abnormal motion of control rods during a rod ejection or rod drop accident, or ingress/egress of liquid during a loss-of coolant accident scenario are all situations that necessitate the development of computational tools for the analysis of space- and time-dependent reactor phenomena. The study of such problems is pursued via a space-time reactor kinetics code that calculates a time- and space-dependent solution to the neutron transport equation.

A typical reactor kinetics calculation will be costly in terms of computing resources and, therefore, methods and approximations are chosen so as to strike an optimal balance between the accuracy of a solution and the efficiency of its computation. In the earliest reactor kinetics codes, developed at a time when computing resources were limited, the space-independent point kinetics approximation was frequently utilized; however, in the current state-of-the-art a fully space- and time-dependent solution is preferred, as sought via nodal diffusion theory. The first part of this introduction will review fundamental techniques that underly the current generation of space-time reactor kinetics codes,
followed by a discussion of the increasingly common failures of diffusion-based methodologies. The development of a time-dependent method of characteristics falls within a growing field of transport-based reactor kinetics and a few of the alternative approaches will be summarized in the second part of the introduction. Finally, the scope of the present work will be outlined and specific tasks for this study will be elucidated.

1.1 Review of State-of-the-Art Space-Time Reactor Kinetics Methods

The deterministic study of the time- and space-dependent population of neutrons in a nuclear reactor requires a solution to the general neutron transport equation, which is derived from the formula developed originally by Ludwig Boltzmann to describe the kinetic theory of gases

\[
\frac{1}{v(E)} \frac{\partial \Phi(r, \Omega, E, t)}{\partial t} + \Omega \cdot \nabla \Phi(r, \Omega, E, t) + \Sigma_t(r, E) \Phi(r, \Omega, E, t) = \int \Sigma_x(r, E') f(r, \Omega', E' \rightarrow \Omega, E) \Phi(r, \Omega', E', t) d\Omega dE + Q(r, \Omega, E, t) .
\]

This general form of the transport equation is continuous in the temporal, energy, angular, and spatial domains. Key physical parameters include the magnitude of the neutron velocity, \( v(E) = |v(E)| \), the neutron angular flux, \( \Phi(r, \Omega, E, t) \), and the external neutron source term, \( Q(r, \Omega, E, t) \). Macroscopic cross-sections are given by \( \Sigma_x(r, E) \), where the subscript \( x \) indicates the particular type of neutron collision event; for example,
\[ \Sigma_t (r, E), \] indicates the total macroscopic cross-section. The term \( \Sigma_t (r, E) f (r; \Omega', E' \rightarrow \Omega, E) \) within the integral on the right hand side of Equation (1.1) represents, on the other hand, a differential cross section. The four dependent variables used in Equation (1.1) to describe the state of a neutron population are the three-dimensional vector position, \( r \); the unit direction vector of the neutron velocity, \( \Omega = \frac{v}{v} \); neutron energy, \( E \); and time, \( t \).

Solution of the transport equation is pursued via numerical techniques that require the introduction of simplifying assumptions for the various time-, space-, angle-, and energy-dependent parameters. A complete set of approximations that reduces the transport equation to a solvable form defines a particular “transport method,” and the selection of an appropriate method will be determined by the problem to which the transport equation is to be applied. In some cases, emphasis will be placed on generating extremely accurate solutions for the neutron flux distribution, while less accurate solutions will be tolerated if they can be generated in a computationally efficient manner. The implied balance between computational efficiency and accuracy is a fundamental issue in the development of all transport methods. Diffusion theory is the preferred method within current reactor kinetics methodologies because it is remarkably efficient and achieves a sufficient, albeit limited, accuracy for most problems of interest.
1.1.1 Nodal Diffusion Theory

Diffusion theory\(^3\) belongs to a class of methods that solve the neutron transport equation in integral form,

\[
\frac{1}{v} \frac{\partial}{\partial t} \phi(r, E, t) + \nabla \cdot J(r, E, t) + \sum_s (r, E, t) \phi(r, E, t) = q(r, E, t),
\]

(1.2)

where \( J(r, E, t) \) is the neutron current and \( q(r, E, t) \) is the total neutron source term, given by

\[
q(r, E, t) = \int dE' \Sigma_s (r, E' \rightarrow E, t) \phi(r, E', t)
\]

\[
+ (1 - \beta) \chi_p (E) \int dE' \nu \Sigma_f (r, E', t) \phi(r, E', t) + \sum_{d=1}^{D} \lambda_d \chi_d (E) C_d (r, t).
\]

(1.3)

Equation (1.2) is obtained by integrating the transport equation over the angular domain and describes the neutron population in terms of direction-independent variables – the neutron current and the neutron scalar flux. Equation (1.2) can be easily reduced to steady state form by transforming the time-dependent diffusion equation into a time-independent eigenvalue problem. In fully time-dependent space-time kinetics applications, on the other hand, a second equation must be introduced to describe the space- and time-dependent population of delayed neutron precursors, \( C_i (r, t) \),
\[
\frac{\partial}{\partial t} C_d (r, t) = -\lambda_d C_d (r, t) + \beta_d \int dE' \chi_d (E') \nu \Sigma_f (r, E', t) \phi (r, E', t). \tag{1.4}
\]

Solution of the coupled set of differential equations, Equations (1.3) and (1.4), within a time-dependent integral transport computation is complicated by the presence of the streaming term on the left hand side of Equation (1.2), \( \nabla \cdot J (r, E, t) \). To simplify this term the diffusion approximation is introduced, which is based on the observation that the motion of the neutrons in a nuclear reactor resembles the process of diffusive transport as governed by Fick’s Law,

\[
J (r, t) = -D (r) \nabla \phi (r, t). \tag{1.5}
\]

Inserting the Fick’s Law substitution for the neutron current into Equation (1.2) yields,

\[
\frac{1}{\nu^g} \frac{\partial \phi^g}{\partial t} - \nabla \cdot D^g \nabla \phi^g (r, t) + \Sigma_r \phi^g (r, t) =
\sum_{g'} \Sigma_s^{g'g} \phi^{g'} (r, t) + (1 - \beta) \chi_{\text{prompt}}^{g} \sum_{g'} \Sigma_f^{g'g} \phi^{g'} (r, t) + \chi_{\text{delay}}^{g} \sum_{d} \lambda_d C_d (r, t) \tag{1.6}
\]

and

\[
\frac{\partial C_d}{\partial t} = \beta_d \sum_{g} \nu^g \Sigma_f^{g} \phi^g (r, t) - \lambda_d C_d (r, t), \tag{1.7}
\]
where traditional multi-group techniques have been introduced to decompose the energy domain into a set of $g = (1, \ldots, G)$ discrete energy groups. Equations (1.6) and (1.7) are the principle equations to be solved by diffusion-based reactor kinetics code and such codes differ regarding techniques that are introduced to simplify the spatial and temporal domains, as well as the methods used to compute a numerical solution.

The most common approach to simplification of the spatial domain in Equations (1.6) and (1.7) is via the introduction of finite difference spatial discretization techniques; however, greater computational efficiency is achieved when they are first recast into a nodal form. The principle technique in all nodal methods is to integrate the multi-group diffusion equations over the volume of an arbitrary coarse node. A nodal balance formulation for the steady state diffusion equation, for example, will take the form

$$
\frac{1}{\Delta x} \left( J_{g_x}^l - J_{g_x}^l \right) + \frac{1}{\Delta y} \left( J_{g_y}^l - J_{g_y}^l \right) + \frac{1}{\Delta z} \left( J_{g_z}^l - J_{g_z}^l \right) + A_g \bar{\phi}_g = \bar{Q}_g,
$$

where the node-averaged neutron scalar flux, $\bar{\phi}_g^l$, the node-averaged neutron source, $\bar{Q}_g^l$, and the face-directed neutron currents, $\bar{J}_{g_x}^l$, $\bar{J}_{g_y}^l$, and $\bar{J}_{g_z}^l$, have been introduced. The index $l$ denotes the particular node of interest and the coefficients $A_g^l$ are introduced to simplify the neutron loss terms from Equation (1.6). The primary task of any particular nodal method is to develop a relation between the node-averaged scalar flux and the face-directed currents.
The transverse leakage approximation\(^5\) is commonly applied to the nodal balance equation, decomposing Equation (1.8) into a set of coupled one-dimensional equations, corresponding to the three coordinate directions. Each of these equations is integrated over the node dimensions that are transverse to the direction of interest, yielding

\[
\frac{d}{dx} j_g^l(x) + A_g^l \phi_g^l(x) = Q_g^l(x) - \frac{1}{\Delta y} L_{gy}^l(x) - \frac{1}{\Delta z} L_{gz}^l(x),
\]

\[
\frac{d}{dy} j_g^l(y) + A_g^l \phi_g^l(y) = Q_g^l(y) - \frac{1}{\Delta z} L_{gz}^l(y) - \frac{1}{\Delta x} L_{gx}^l(y),
\]

\[
\frac{d}{dz} j_g^l(z) + A_g^l \phi_g^l(z) = Q_g^l(z) - \frac{1}{\Delta x} L_{gx}^l(z) - \frac{1}{\Delta y} L_{gy}^l(z),
\]

(1.9)

where \( L_{gy}^l \) represent leakage in the transverse directions, \( q = (x,y,z) \). Solution of Equations (1.9) requires the introduction of an approximation for the one-dimensional scalar flux. In the nodal expansion method\(^6\) the scalar flux is assumed to have a polynomial form, while in the analytic nodal method\(^7\) a more exact approximation is introduced. A variety of other flux approximations have been utilized, but the general result will be a set of flux and leakage moments that are updated during standard source/scattering fission iteration, and a set of equations that can be arranged in response matrix form. Alternatives to the transverse integration procedure\(^8\) and response matrix iteration\(^9\) have also been suggested.
1.1.2 The Limitations of Nodal Diffusion Theory

Nodal diffusion theory is a powerful numerical method that yields an efficient solution to the transport equation in the transient analysis of large reactor systems. It is well known, however, that diffusion theory will fail to provide accurate results in situations where strong absorbers are present (such as near a control rod), when neutron streaming is significant (across voids), or where neutron scattering is otherwise significantly anisotropic. Circumstances that cause a rapid spatial variation in the neutron scalar flux generally serve as an indicator of potential problems with diffusion theory results. Models that include extreme material variations can also be troublesome; for example, diffusion theory is suitable for analyzing the interior of a core, but results will degrade towards the periphery where structural materials such as the pressure vessel are present. Until recently, such circumstances have been rarely significant to the field of power reactor engineering, but new advances in reactor and fuel designs have initiated a trend whereby diffusion theory is increasingly suspect.

Much of the success of diffusion theory in modeling LWR reactor cores has been due to the relatively simple design of most LWR fuel assemblies. These assemblies would be composed of regularly spaced fuel pins, yielding a spatial uniformity that is ideal for diffusion theory applications. New fuel designs are more complicated, including such features as large internal coolant channels that increase moderation in the assembly interior, variable enrichment of fuel rods within a single assembly, burnable poisons that act as strong absorbers within the assembly interior and whose physical characteristics
change throughout the fuel lifetime, and partial length fuel rods that yield a more dynamic axial flux profile than has been previously encountered\textsuperscript{10}. The incorporation of high burnup fuel into core loading patterns is another factor of importance\textsuperscript{11}. All of these advancements in fuel design can lead to local variations in the neutron scalar flux profile that become exacerbated during accident scenarios and for which diffusion theory may produce inaccurate results.

Another development in the field of fuel management is a trend towards core loading patterns that include mixed-oxide (MOX) fuel assemblies\textsuperscript{12}. MOX fuel assemblies are of interest as a means to reprocess and dispose of spent nuclear fuel, and may even include highly enriched weapons-grade plutonium. The loading of MOX assemblies adjacent to traditional uranium-oxide (UOX) fuel assemblies can lead to complicated physical effects at the interface between assemblies of different types. In such a situation, plutonium in the MOX fuel is a more efficient absorber of neutrons than is uranium; thus, the MOX fuel assembly will preferentially absorb neutrons near the interface and act as a poison for the UOX assembly. The result is a severe flux peak on the MOX side of the interface, a flux dip on the UOX side, and a steep flux gradient across the interface. In the presence of such complicated physical phenomena diffusion theory is known to fail, as illustrated by the recent publication of results for several different MOX-oriented benchmark problems\textsuperscript{13,14}.

Advanced LWR technologies are not limited to new developments in fuel design. A primary role for space-time kinetics is to analyze the transient behavior of a reactor
during the motion of control rods. One such situation would be the slow withdrawal of a control rod as a reactor is ramped in power during start-up procedures. In nodal diffusion theory, a particular problem arises as the tip of the control rod moves axially through the nodal model; whenever the rod tip moves through a nodal interface, a cusp in the effective multiplication factor will occur. This so-called “rod cusping problem” does not reflect an actual physical phenomenon, but is rather an effect of homogenization procedures that are required for implementation of nodal diffusion theory. Control rods are particularly troublesome to diffusion theory in other problems as well, including rod ejection or rod drop accident scenarios that are the design basis accident for some commercial reactor types. As a control rod is a strong absorber, application of diffusion theory to problems with such control rod effects can lead to significant inaccuracies.

Beyond the field of LWR engineering, improved space-time kinetics codes will be required to analyze transient problems for alternative reactor designs that are currently under development and could enter commercial operation in the next few decades. One example is the Pebble Bed Modular Reactor (PBMR) design, presently under construction in the Republic of South Africa. LWR cores are composed of a regular and periodic lattice of square fuel assemblies with an effectively infinite height. PBMR cores, on the other hand, are composed of thousands of spherical fuel pebbles that form a randomly packed bed. The coolant in a PBMR is gaseous, which leads to low-density (effectively void) regions within the core where streaming will be a dominant transport effect. Investigations of this reactor type have shown that, in a core comprised of randomly packed spherical pebbles, the void fraction will depend strongly on position.
within the core\textsuperscript{18}. Specifically, the void fraction will increase towards the outer edge of the core, approaching 100\% at the pressure vessel boundary. Such a situation introduces strong material gradients in the outer portions of the core and flux profiles across these interfaces cannot be accounted for via diffusion theory. Although the region affected is on the order of only three to five ball diameters (the fuel pebble diameter is \textsim 6.0 cm), the fact that the control rods in a PBMR lie outside the core makes this an issue of potentially critical importance. Some attempts have been made to extend nodal diffusion theory to PBMR geometry\textsuperscript{19}; however, fundamental issues such as packed-bed voiding have not been adequately addressed.

One final problem that is encountered in many commercial reactors but is difficult to model using nodal diffusion theory regards the response of external neutron detectors. In the immediate aftermath of the infamous accident at Three Mile Island Unit 2 (TMI-2), a group of researchers at the Pennsylvania State University undertook to develop a computational model for the Source Range Monitor (SRM) detectors at TMI-2, from which a stream of response data had been recorded for the entire accident duration\textsuperscript{20,21}. The study utilized a 2D finite-difference discrete ordinates calculation to predict neutron flux through the core boundary, biological shields, and pressure vessel walls to the neutron detectors located exterior to these structures. In the TMI-2 accident, however, there was considerable voiding in the downcomer region and portions of the fuel had relocated to the vessel bottom. In the latter case, fuel relocation introduced an additional streaming path directed vertically along the downcomer, for which anisotropic scattering becomes a dominant factor. Such circumstances are particularly damaging to the
accuracy of a diffusion theory calculation and limit the options for reactor operators. The
capability to predict reactor behavior based on SRM response data would be a boon to
developers of nuclear reactor simulators, for example\textsuperscript{22}.

For many of the problems mentioned above, improvements to pre-existing diffusion
theory methods have been implemented without immediate recourse to a higher-order
transport method. In many cases, corrections to homogenization procedures or
improvements to the nodal method restore sufficient accuracy to the results. Such efforts
generally aim to “capture” some of the transport effects involved in a particular scenario,
while maintaining a reliance on the computationally efficient nodal diffusion method of
choice. The above discussion illustrates, however, that a trend towards more complicated
reactor designs may lead to circumstances that cannot be captured by diffusion theory
and for which a more accurate transport method must be investigated. A few of the more
attractive transport methods for use in a space-time reactor kinetics computation are
reviewed in Section 1.2.

\section{1.2 Alternative Neutron Transport Methods}

To improve on the results of nodal diffusion theory for problems such as those that are
discussed above, a computational method that incorporates higher-order approximations
for the angular dependence of the neutron flux must be employed. A wide variety of
transport methods have been developed for reactor applications\textsuperscript{23} and, in principle, any of
these can be applied to the space-dependent reactor kinetics problem. The obstacle to
such implementation is computational expense, as most transport methods achieve gains in accuracy at the cost of long runtimes and burdensome memory requirements. For this reason, non-diffusion methods have been used rarely in the analysis of time-dependent effects for large geometries. A few of the deterministic methods that may be amenable to such a calculation are reviewed in this section, with emphasis placed on differences in spatial and angular approximations. Techniques for discretizing the temporal domain will be reviewed thoroughly in Chapter 4.

1.2.1 Finite-Differenced Discrete Ordinates Methods

One of the most significant impediments to solution of the general neutron transport equation, Equation (1.1), is the requirement for an approximation to discretize the angular domain. In discrete ordinates methods\(^{24}\) this approximation computes the neutron angular flux, \(\Phi(r, \Omega, E, t)\), for a finite set of discrete directions of neutron travel, yielding the discrete ordinates equations,

\[
\left[ \mu_m \frac{\partial}{\partial x} + \eta_m \frac{\partial}{\partial y} + \xi_m \frac{\partial}{\partial z} + \Sigma_i(r) \right] \Phi_m(r) = Q(r), \tag{1.10}
\]

where, for simplicity, it is assumed that neutron scattering is isotropic and the representation of energy-dependence has been suppressed. Directions that are not explicitly included in the discrete set, \(m = (1, \ldots, M)\), are accounted for via the application of weighting factors, and together the total set of discrete angles and weights is known as
a quadrature set. One of the most common quadrature sets for use in finite-differenced discrete ordinates methods (S_N) is based on the spherical harmonics expansion that is described in detail in Section 1.2.2.

As will be shown in Chapter 2, the method of characteristics also utilizes the discrete ordinates approximation and is properly classified as a discrete ordinates method. The primary difference between the method of characteristics and S_N methods results from the techniques that are used to discretize the spatial domain. In the latter case, this task is accomplished via the well-known spatial finite differencing technique. To illustrate the basic principles of finite differencing, a simple example from diffusion theory will be presented. In 2D Cartesian geometry, the diffusion equations are given by

\[
-D \frac{\partial^2 \phi}{\partial x^2} - D \frac{\partial^2 \phi}{\partial y^2} + \Sigma_c \phi(x, y) = Q(x, y).
\]  

(1.11)

Equations (1.11) cannot be solved directly due to the presence of the double spatial derivates, \( \frac{\partial^2 \phi}{\partial x^2} \) and \( \frac{\partial^2 \phi}{\partial y^2} \). To reduce these terms, the problem space is divided into a set of discrete cells as shown in figure 1.1.
In diffusion theory, where the neutron scalar flux is the primary unknown parameter, cell-averaged values are typically defined and the double-spatial derivatives in Equation (1.11) are approximated for cell \((x_i, y_j)\) using values from the four edge-adjacent cells, so that

\[
\frac{\partial^2 \phi}{\partial^2 x}|_{x_i,y_j} \approx \frac{\bar{\phi}_{i-1,j} - 2\phi_{i,j} + \bar{\phi}_{i+1,j}}{(\Delta x)^2} \tag{1.12}
\]
and

\[
\left. \frac{\partial^2 \phi}{\partial y^2} \right|_{y_{i,j}} \approx \frac{\bar{\phi}_{i,j-1} - 2\bar{\phi}_{i,j} + \bar{\phi}_{i,j+1}}{(\Delta y)^2}.
\] (1.13)

where \( \bar{\phi}_{i,j} \) represents the cell-averaged scalar flux in cell \((x_i, y_j)\). Equations (1.12) and (1.13) discretize the problem using a central differencing approach, which defines the flux in cell \((x_i, y_j)\) in relation to all four adjacent cell fluxes. Other simple differencing schemes include backwards differencing, which defines cell \((x_i, y_j)\) in relation to cells \((x_{i-1}, y_j)\) and \((x_i, y_{j-1})\), or forwards differencing, which defines \((x_i, y_j)\) in relation to cells \((x_{i+1}, y_j)\) and \((x_i, y_{j+1})\).

The discrete ordinates equations can be discretized in a similar manner to reduce the spatial derivatives of the neutron angular flux, \(\Phi(r, \Omega, E)\). In diamond differencing, for example, the difference relations for cell \((x_i, y_j)\) are defined by integrating the 2D discrete ordinates equation over the cell area, yielding

\[
\frac{\mu_m}{\Delta x} \left( \Phi_m^{i+1/2,j} - \Phi_m^{i-1/2,j} \right) + \frac{\eta_m}{\Delta y} \left( \Phi_m^{i,j+1/2} - \Phi_m^{i,j-1/2} \right) + \Sigma^B \Phi_m^{ij} = Q_m^{ij},
\] (1.14)
where $\Phi_{m}^{i+1/2,j}$ and $\Phi_{m}^{i,j+1/2}$ represent the angular flux averaged over the right and top cell edges, respectively, and $\Phi_{m}^{ij}$ and $Q_{m}^{ij}$ are the cell-averaged neutron angular flux and total neutron source term, respectively. Equation (1.14) is an exact representation of the neutron transport equation; however, a set of expressions must now be introduced to describe the relation between the edge-averaged cell fluxes and the area-averaged cell flux. In the diamond differencing approximation, these relations are given by

$$
\Phi_{m}^{ij} = \frac{1}{2} \left[ \Phi_{m}^{i+1/2,j} + \Phi_{m}^{i,j+1/2} \right] \quad (1.15)
$$

and

$$
\Phi_{m}^{ij} = \frac{1}{2} \left[ \Phi_{m}^{i+1/2,j+1/2} + \Phi_{m}^{i-1/2,j+1/2} \right]. \quad (1.16)
$$

The set of Equations (1.14), (1.15), and (1.16) can be solved using a variety of iterative techniques to generate a detailed neutron angular flux distribution. Further gains in accuracy of this solution can be achieved by introducing more advanced differencing approximations, such as the weighted diamond differencing formulation that is incorporated into many state-of-the-art $S_N$ codes.

Recently, a variety of investigations have extended the traditional $S_N$ formalism to include nodal methods. Application of the principles of nodal methods to the discrete ordinates equations yields a coarse mesh solution to the traditional $S_N$ problem. Such an
approach would alleviate one of the primary sources of computational inefficiency for
discrete ordinates calculations – the dependence on a fine-mesh spatial discretization
scheme. Implementation of an efficient coarse-mesh methodology that can retain much
of the accuracy of the underlying transport approximation is the focus of current nodal
transport theory research. Success in the formulation of nodal $S_N$ methods has enabled
the formulation of reasonably efficient time-dependent applications. The most notable of
these are DORT-2D$^{28}$, which solves the transport problem in 2D geometry, and TD-
TORT$^{29}$, which can solve the full time-dependent 3D transport equation.

1.2.2 Spherical Harmonics Methods

Unlike $S_N$ methods, which decompose the angular domain into a set of discrete
directions, spherical harmonics ($P_N$) methods$^{30,31}$ preserve continuity of the angular
domain via an expansion for each of the direction-dependent parameters in the neutron
transport equation. These expansions are defined in terms of spherical harmonics or
Legendre polynomials, as will be demonstrated below. $P_N$ methods are amenable to
multi-group discretization of the energy domain, although solutions are frequently sought
for the one-group transport equation. A fine-mesh spatial grid can be imposed on this in
a similar manner to $S_N$ discretization methods.

The $P_N$ approximation is typically introduced as a means to solve the one-group transport
equation for planar geometries, as the resulting equations are quite simple and relatively
easy to solve:
\[
\mu \cdot \nabla \Phi(x, \mu) + \Sigma(x) \Phi(x, \mu) = \int \Sigma_s(x, \mu, \mu') \Phi(x, \mu') d\mu' + Q(x, \mu),
\]

(1.17)

where the dependent variables \(x\) and \(\mu\) refer to the one-dimensional spatial and angular domains, respectively. In this context, the scattering cross section is expanded in terms of Legendre polynomials as

\[
\Sigma_s(x, \mu_0) = \sum_{l=0}^{\infty} \frac{2l+1}{4\pi} \Sigma_{sl}(x) P_l(\mu_0),
\]

(1.18)

where \(\mu_0 = \mu \cdot \mu'\). Next, the neutron angular flux, \(\Phi(x, \mu)\), and the neutron source term, \(Q(x, \mu)\), are expanded in a similar manner, yielding the one-group \(P_n\) equations for planar geometry,

\[
\frac{n}{2n+1} \frac{d \phi_{n-1}}{dx}(x) + \frac{n+1}{2n+1} \frac{d \phi_{n+1}}{dx}(x) + \Sigma_n(x) \phi_n(x) = Q_n(x), n = 0, 1, 2, \ldots N
\]

(1.19)

where \(\Sigma_n(x) = \Sigma(x) - \Sigma_{sn}(x)\), \(\phi_{-1} = 0\) and, due to orthogonality of the Legendre polynomials,

\[
\phi_m(x) = 2\pi \int_{-1}^{1} \Phi(x, \mu) P_m(\mu) d\mu
\]

\[
Q_m(x) = 2\pi \int_{-1}^{1} \Phi(x, \mu) P_m(\mu) d\mu
\]

(1.20)
Derivation of the $P_n$ equations for general geometry follows a similar procedure; however, the neutron angular flux is expanded in this case into a series of spherical harmonics:

$$
\Phi(r, \Omega, E) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \phi(r, E) Y_{lm}(\varphi, \theta), \quad (1.21)
$$

where the unit direction vector, $\Omega$, is expressed in terms of an azimuthal angle, $\varphi$, and polar angle, $\theta$. The spherical harmonics, $Y_{lm}$, are expressed as

$$
Y_{lm}(\varphi, \theta) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} P_l^m(\mu) e^{im\varphi}, \quad (1.22)
$$

given the Legendre polynomials, $P_l^m$. To solve these equations the $P_N$ approximation is introduced, whereby the spherical harmonics expansions are truncated at order $N$ and

$$
\nabla \phi_{N+1}(r) = 0 \quad (1.23)
$$

The $P_N$ approximation yields a system of $N + 1$ equations and $N + 1$ unknowns that can be solved using standard numerical techniques.

One of the primary advantages of the $P_N$ approximation is that the low order flux moments, $\phi_0$ and $\phi_1$, correspond to the neutron scalar flux and the neutron current,
respectively, and it can be shown that the P₁ approximation will yield the mathematical equivalent of the diffusion theory equations. For problems where diffusion theory is not applicable, however, utility of the Pₙ approximation may be limited. To accurately model problems with strong anisotropy, for example, a high order of approximation is typically required, yielding an extremely large system of equations. Numerical solution of these equations is generally far less computationally efficient than would be solution of the Sₙ equations of comparable order. As a result, Sₙ methods are preferred to Pₙ solutions for most problems of interest. In modern neutron transport applications, the Pₙ approximation has been incorporated into acceleration techniques and as a means for generation of discrete ordinates quadrature sets. The feasibility of a time-dependent Pₙ method has also been recently investigated.

1.2.3 Simplified Spherical Harmonics Methods

The simplified-Pₙ (SPₙ) method was first developed by E. M. Gelbard in the 1960s and derived independently by K.S. Smith many years later. Although initially this method was overlooked due to an unusual mathematical pedigree, recent results have shown low-order SPₙ equations to be effective at introducing transport corrections into nodal diffusion theory solutions. A detailed analysis of SPₙ method is complicated by the fact that there are at least four distinct methods of derivation presented in recent literature. Two of these formulations – the “conventional” and “canonical” formulations – are based on an observation about the relationship between planar geometry and general 3D geometry:
In the conventional form, the transformation implied by Equations (1.24) and (1.25) is applied to the $P_n$ equations for a particular low-order $P_N$ approximation; for example, the $P_2$ or $P_3$ equations. In the canonical form, the same transformation is applied to an appropriate set of even-parity $S_{N+1}$ equations. These first two approaches to the $SP_N$ approximation involve techniques that reduce the neutron transport to a low-order angular approximation. A third $SP_N$ derivation begins instead with the integral balance equation and develops an exact relation between the neutron scalar flux and divergence of the neutron current, rather than the approximate relation that is Fick’s Law.

Each of the various approaches to the $SP_N$ approximation yields an algebraically equivalent set of equations; however, this approximation is limited to relatively low orders, such as $SP_2$ or $SP_3$. As a result, $SP_N$ is constrained by the same limitations, somewhat relaxed, as is nodal diffusion theory; i.e., the scalar flux should vary slowly in space and the model should not include strong absorbing regions. Specifically, results for $SP_N$ have been shown to degrade in regions where neutron streaming dominates. $SP_N$ is
limited further by suspect mathematical foundations. $S_N$ and $P_N$ methods will asymptotically approach an exact solution to the transport equation for arbitrarily high orders of angular approximation. Most $SP_N$ derivations do not preserve this asymptotic behavior, although variational approaches to $SP_N$ have demonstrated that a more exact formulation is possible; however, this remains of theoretical interest only as the practical implementation of a high-order $SP_N$ approximation has not been attempted.

While low-order $SP_N$ approximations will not be as accurate as high-order transport methods for problems with severe transport effects, $SP_2$ and $SP_3$ yield more accurate results than nodal diffusion theory for problems where transport effects are present but not severe. Additionally, the $SP_N$ approximation yields equations that are amenable to the numerical solution techniques that are currently implemented in most diffusion-based kinetics packages. As a result, the implementation of $SP_N$ into a nodal diffusion theory code is easily achieved and is an attractive first step towards improving the results of reactor kinetics computations for complicated problems.

1.3 Proposal for the Development of New Methods

The primary goal of this thesis is to develop a space-time reactor kinetics method based on a transport method that is more accurate than diffusion theory. Similar undertakings have been successfully achieved for all of the methods reviewed in Section 1.2, and also for stochastic Monte Carlo techniques, collision probability methods, and variational formulations. In this work, the method of characteristics is selected for further study.
The method of characteristics is a well-known technique for solution of the transport equation that is applied primarily to 2D steady state assembly transport computations; however, there have been no previous attempts to develop a time-dependent formulation. Application of the method of characteristics to space-time reactor kinetics problems will thus require the development of two novel improvements to currently available characteristics methods. First, techniques for the performance of a method of characteristics calculation in explicit 3D geometry must be developed. This task has been previously investigated, with limited success, by developers of lattice physics codes. The second task – derivation of a time-dependent method of characteristics – has not been previously attempted and is the most significant contribution of this work.

The development of a new kinetics package is a significant undertaking. A complete package would incorporate the new techniques outlined above, incorporate nodal method and/or computational acceleration to ensure that the result is computationally efficient, and also include the effects of thermal-hydraulic feedback on the nuclear properties. Such features, while necessary for commercial application, would unnecessarily complicate the initial development of the proposed new kinetics method. Instead the scope of this work will be strictly limited to only those features that are required to assess the feasibility of a kinetics method based on the method of characteristics. Specifically, the following features will be neglected or simplified during this initial development:
• Neutron scattering will be approximated using a low-order Legendre expansion, effectively isotropic, thus limiting the applicability of this method to problems without significant anisotropy.

• Nodal methods, which are utilized in most space-time kinetics methodologies, will not be investigated.

• Specialized techniques for computational acceleration – coarse-mesh rebalance, for example – will not be included.

• Thermal-hydraulic feedback and coupling to an external thermal-hydraulics computation will not be considered.

• Parallel processing, which is of particular relevance to the method of characteristics, will not be attempted; instead, all computations will be performed in a serial processing mode.

As a result of these limitations on the scope of this initial development, it is expected that neither accuracy nor computational efficiency will be of a quality that would be required for commercial applications; however, that is not the goal of this project. The goal here is rather to demonstrate the feasibility of this new approach to space-time kinetics problems. If the method of characteristics is determined to be a viable alternative to diffusion theory for such applications, then the features that have been omitted here can be more thoroughly investigated and implemented in future work.

Past implementations of the method of characteristics will be discussed in detail in Chapter 2 and, subsequently, the traditional steady state derivation of the method of
characteristics will be presented. The development of a ray tracing methodology for
treating explicit three-dimensional Cartesian geometry is described in Chapter 3,
including a thorough description of all assumptions and approximations, as well as a
detailed set of procedures for computational implementation. The time-dependent
method of characteristics is derived in Chapter 4. The implementation of the three-
dimensional steady state and time-dependent methods of characteristics into the MOCK-
3D reactor package is outlined in Chapter 5. Results for a set of test problems are
produced and analyzed in Chapter 6 for the purposes of verifying the methods developed
in this project and implemented in MOCK-3D. Finally, conclusions will be drawn in
Chapter 7 regarding the feasibility of a time-dependent method of characteristics, with
emphasis placed on the needs of future development.
CHAPTER 2:
A REVIEW OF PREVIOUS EXPERIENCE WITH THE METHOD OF CHARACTERISTICS

The most significant role for the method of characteristics in state-of-the-art computational nuclear reactor analysis is to perform an assembly transport calculation, which is the primary transport solver in most lattice physics codes. The methods reviewed in Chapter 1 that solve the steady state and/or time-dependent transport equation require as input a set of material properties that are computed using lattice physics methods. Such calculations utilize a highly detailed spatial model, for which the method of characteristics is uniquely applicable; however, lattice physics implementations of the method of characteristics are typically limited to one- and two-dimensional problems for limited regions in the core for reasons of computational expense. Recent advancements to the method of characteristics have expanded these capabilities to include whole-core assembly transport calculations and, with more limited success, the modeling of three-dimensional geometry.

As the development of an accurate and efficient 3D steady state method of characteristics is a necessary first step towards a time-dependent method of characteristics, fundamental physical principles that underly the method of characteristics are discussed in Section 2.1. Past experience with the implementation of steady state characteristics formulations in
lattice physics codes is reviewed on Section 2.2, with special emphasis placed on recent advancements that are of relevance to the present work. Finally, one particular method of characteristics formulation is derived in Section 2.3, representing the approach most commonly utilized to solve the steady state assembly transport problem. While the contents of this derivation are relatively well understood, the steady state method of characteristics will provide a basis for developments that are discussed in later chapters; for example, the development of 3D ray tracing procedures – the topic of Chapter 3 – is based on recent advances that are the focus of ongoing improvements to lattice physics, while many of the approximations that will be introduced during the derivation of the time-dependent method of characteristics in Chapter 4 are identical to methods that will be defined and derived below in regards to the steady state formulation.

2.1 Physical Origins of the Method of Characteristics

In most of the transport methods reviewed in Chapter 1, neutron motion is observed in relation to an arbitrary fixed point in space – the general LAB coordinate system of classical physics. The method of characteristics belongs to a class of solutions to the general neutron transport equation that utilize a Lagrangian coordinate transformation to the neutron frame-of-reference, so that neutron motion is observed in relation to the paths of neutron propagation. Lagrangian methods allow for a more exact spatial representation that does not require the introduction of approximations such as finite differencing, although special care must be taken to ensure that neutron balance is conserved. Collision probability methods are also propagation methods and the
development of some method of characteristic techniques has drawn on collision probability experience; however, whereas the method of characteristics is utilized to solve the integro-differential transport equation, collision probability formulations are applied to the direction-independent integral transport equation that is also the focus of diffusion theory methods.

Lagrangian coordinates can be utilized in any problem where matter and/or energy are transmitted along definable propagation paths. Theoretically, the method of characteristics could be utilized in the investigation of load propagation for structural analysis, of the transmission of radiation through interstellar space, of various properties of fluid flow, and also of traffic patterns on city streets\textsuperscript{42}. In practical application, the method of characteristics is commonly employed in studies of supersonic flow in compressible fluids\textsuperscript{43,44,45} and of interstellar radiation transport\textsuperscript{41}. For non-nuclear applications the most significant obstacle is often determination of these propagation paths. In compressible fluid flow studies, for example, the propagation paths correspond to flow streamlines and can be curved in space or vary with time. In this context, the method of characteristics must focus on means for determining these complicated paths.

In neutron propagation a simplification can be employed to eliminate the need for curved and/or time-dependent propagation paths, as neutrons are neutrally charged particles and the effect of nuclear forces on a neutron’s path-of-travel can be safely neglected. If neutron-neutron collisions are also neglected, as is typically done in deriving the transport equation, then the only significant events that can affect neutron propagation are
direct collisions with atomic nuclei; thus, it can be assumed that neutrons will travel in straight lines between collisions and curved propagation paths are explicitly avoided. Time-dependence of the characteristics can be neglected if time scales for the phenomena of interest are appropriately defined. Similar circumstances are encountered during the analysis of transport in a vacuum of neutrally or weakly charged particles, as in the propagation of cosmic rays through interstellar space.

The Lagrangian spatial perspective of the method of characteristics is useful in the analysis of certain commonly encountered nuclear engineering problems. The most well known example of such application of the method of characteristics is to lattice physics codes that generate a set of macroscopic cross sections for use in further computational reactor analysis. Other formulations for the method of characteristics have been applied to shielding and deep penetration problems. In both cases the method of characteristics is uniquely applicable due to its accuracy for problems that include a detailed spatial description of the problem space. Because they utilize a similar spatial description, collision probability methods are also employed in lattice physics applications; however, in the past few decades the method of characteristics has become the preferred method due to its more accurate transport solution and faster runtimes.

### 2.2 The CACTUS Method

The first method of characteristics formulations for nuclear applications were derived by V. S. Vladimirov\textsuperscript{46} in 1959. In 1972 the development of a practical 2D steady state
method of characteristics was reported by J. R. Askew\textsuperscript{47}, as implemented by M. J. Halsall into the lattice physics code CACTUS\textsuperscript{48} eight years later. The “CACTUS method” would eventually become the computational heart of the WIMS lattice physics package\textsuperscript{49} and the most common approach to solution of the transport equation for 2D assembly transport problems. A plethora of commercial lattice physics codes are based on this formulation, including CASMO-4\textsuperscript{50}, DENT-2D\textsuperscript{51}, CHAPLET-3D\textsuperscript{52}, CRX\textsuperscript{53}, DRAGON\textsuperscript{54}, MCCG\textsuperscript{55}, and LANCER-02\textsuperscript{56}. The primary role of these codes is to perform assembly-level energy group collapse and spatial homogenization calculations, thereby generating the set of few-group material cross-sections that is required for solution of the multi-group nodal diffusion equations.

Assembly transport calculations include extremely detailed models of the reactor internals, which for a typical LWR fuel assembly will include the fuel pin, fuel cladding, and gas gap, as well as external assembly structures, control rods, and the coolant channels. The spatial model for such a calculation may include a combination of rectangular, cylindrical, and/or hexagonal features, requiring substantial spatial flexibility by the transport solver. It is for this reason that the method of characteristics has become the norm for such calculations; due to its unique Lagrangian spatial discretization, the method of characteristics performs well for problems with severe spatial heterogeneity. Traditionally, lattice physics applications of the method of characteristics have been limited to 2D geometry for small sub-sections of a reactor model due to the extreme computational expense that is associated with this method for large and detailed problems. A typical calculation would model a single fuel assembly or a small group of
adjacent assemblies and treat these as infinitely periodic in the radial direction and infinite in extent along the axial direction. For most older commercial reactor core designs this limitation has been acceptable, as these reactor cores are loaded with identical fuel assemblies of simple design and axial detail is safely assumed to be uniform. Features associated with advanced fuel designs demand a more accurate spatial treatment, however, and current research seeks to extend the capabilities of lattice physics codes to include a whole-core radial analysis that includes axial variations.

If the method of characteristics is to be applied to models of large 2D and 3D whole-core systems, the problem of potentially burdensome computational expense must be resolved. Much of this expense results from the requirements of ray tracing – a set of procedures that projects the neutron propagation paths across a discretized system model. A large system model will divide every fuel assembly into a mesh of pin cells, each of which will include internal details such as the boundaries of the fuel pellet and cladding. Characteristics are now projected across this fine-mesh spatial discretization, for each of a finite number of discrete directions, and the length of each segment defined by intersection of these characteristics with the pin cell model must be calculated. The result of such a procedure is a large set of track segments and the computational burdens of ray tracing become manifest in two ways: (1) a large amount of memory is required to store the track length information and (2) iterative computation for the entire track set can lead to long runtimes. Current research into the method of characteristics is directed by the need to reduce this computational expense via reduction of the size of track set, for example, and/or computational acceleration of the transport solution.
2.2.1 Advances in Ray Tracing for the Method of Characteristics

Collision probability methods and stochastic Monte Carlo methods, which also view neutron transport as a propagation problem, may include advanced ray tracing procedures that utilize combinatorial geometry$^{57,58}$ and/or macroband techniques$^{59}$ and these methods have been incorporated into some lattice physics codes$^{12,60}$. In most cases, however, ray tracing procedures for the method of characteristics are derived from techniques that were included in the original CACTUS implementation of the method of characteristics$^{48}$. In CACTUS, relatively simple algorithms and geometrical statements are exploited to project and calculate track lengths for a system model. Typically, separate algorithms will be constructed to account for each type of geometry – Cartesian, cylindrical, hexagonal – and preset shapes will be incorporated to speed the calculation across common spatial patterns, such as the circle-in-a-square that describes a typical pin cell.

The CACTUS ray tracing methodology has been successful in traditional lattice physics applications that model small sections of a reactor, such as a single fuel assembly in a periodic lattice. When applied to a model of an entire reactor core, however, these procedures are not particularly efficient. Additionally, numerical procedures for calculating track lengths can lead to a violation of basic principles such as conservation of area and neutron balance and in certain instances additional numerical instabilities can arise. Such problems are typically resolved via the introduction of corrections to enforce balance and constraints to avoid instabilities.
A significant and fundamental redefinition of the CACTUS ray tracing methods has been recently developed and is known as the direct neutron path linking technique (DNPL). DNPL requires that each characteristic track segment entering or leaving a cell must connect directly to a track in the adjacent cell. The outgoing neutron angular flux for any track is thus identical to the incoming flux for a track along the same direction in an adjacent cell, thus a naturally occurring set of continuity boundary conditions can be employed at all internal cell interfaces. At external boundaries tracks along one discrete direction are required to connect directly to tracks along another direction via optical reflection. These boundary conditions can be exploited to generate a cyclic tracking procedure in which the track lengths for all directions in a cell are computed simultaneously, minimizing the computational effort of ray tracing. DNPL allows for regions with similar material properties – identical fuel assemblies, for example – to be modeled via cells with uniform track sets, further reducing memory storage requirements for large problems.

Incorporation of DNPL into traditional CACTUS-type ray tracing methods has enabled the practical pursuit of whole-core 2D assembly transport calculations; however, further extension of these techniques to 3D geometry is problematic due to the added expense associated with explicit modeling of the axial dimension. One means to ameliorate this expense involves the development of hybrid computational strategies that decompose the 3D problem into simpler 1D and 2D calculations. One such approach performs a whole-core 2D method of characteristics calculation to analyze the radial reactor plane and a separate 1D method of characteristics calculation to account for details along the
axial direction. By coupling the results of these two calculations, a full 3D neutron angular flux distribution can be generated. A similar technique utilizes the same separation and coupling methods, but with a homogenized nodal $S_N$ method applied to axial calculation$^{63}$. In the latter approach coupling of the radial and axial calculations is designed so that appropriate homogenized cross sections are available for the nodal $S_N$ calculation, which results in a moderate capability to perform on-line cross-section generation. A third hybrid method avoids the need for an explicit axial calculation by performing a series of 2D calculations at various axial levels, which are coupled via a non-linear iteration procedure$^{52}$.

Hybrid techniques are computationally efficient and can utilize pre-existing 1D and 2D ray tracing procedures; however, such an approach will place limitations on the accuracy that can be obtained by the characteristics solution. Ideally, accurate calculations in 3D geometry should incorporate explicit models and, to date, two methods for performing 3D ray tracing in explicit 3D geometry have been suggested. The first of these has been incorporated into the MCCG lattice physics code$^{55}$. This approach decomposes 3D space into a set of coupled orthogonal 2D planes and then applies CACTUS-like procedures to each of these 2D domains in sequence. To ameliorate the computational expense, MCCG has included combinatorial geometry to reduce the demands of ray tracing$^{64}$. A similar ray tracing procedure, which decouples 3D space via a different set of orthogonal planes, has been incorporated into the DRAGON code$^{65}$. While explicit 3D calculations represent the ideal case in terms of accuracy, it remains unclear as to whether the
computational expense can be reduced sufficiently to render such an approach practical for use in commercial lattice physics applications\textsuperscript{66,67}.

2.2.2 \textit{Acceleration of the Characteristics Solver}

Lattice physics calculations employ a steady state formulation of the method of characteristics that is amenable to typical source iteration techniques. In source iteration a multiplicative eigenvalue is introduced, the neutron source is separated into scattering and fission components, and the neutron angular flux is calculated via nested inner and outer iterations. During the inner iteration the fission source is held constant while neutron angular flux, neutron scalar flux, and the scattering source are solved iteratively until found to converge within specified tolerance values. These converged values are passed to the outer iteration where the fission source and multiplicative eigenvalue are updated and outer iteration proceeds until the eigenvalue is similarly converged.

Source iteration converges slowly in certain circumstances for all methods and, when the added cost of repeating the neutron angular flux computation for each in a large set of track lengths is included, runtimes for the method of characteristics can become unreasonable. To reduce this computational expense techniques are employed to accelerate convergence of the eigenvalue during outer iteration or of the scalar and/or neutron angular flux during inner iteration. Acceleration of the method of characteristics is complicated by the detailed nature of the spatial model involved in a lattice physics
calculation; thus, many of the techniques available to other transport methods, such as diffusion synthetic acceleration, cannot be applied\textsuperscript{68}.

One of the most common acceleration tools included in lattice physics computations is coarse-mesh rebalance\textsuperscript{69} (CMR), which is also employed in many diffusion theory codes. CMR is applied to the neutron scalar flux distribution during the inner iteration and, via the application of a neutron balance statement for a set of coarse-mesh cells, improves the neutron scalar flux distribution at each iteration and speeds overall convergence. For the method of characteristics, however, even this basic acceleration technique can be inappropriate, as CMR can cause the neutron angular flux to diverge in complicated and detailed geometries. More stable acceleration of the method of characteristics has been achieved by using a transport synthetic technique\textsuperscript{70}, whereby a low-order transport approximation is introduced to improve the unconverged neutron angular flux distribution within the inner iteration. Synthetic acceleration that utilizes Fourier expansion has also been suggested\textsuperscript{71}.

The runtime efficiency of a method of characteristics computation can also be improved via the introduction of parallel processing. Parallelization of most other types of transport methods will generally decompose the problem based on angle or, more typically, space. Propagation solutions permit a more logical decomposition\textsuperscript{72,73,74}; as characteristics are inherently independent of one another, the total set of characteristics can be divided into clusters, each of which is calculated on a separate processor. As mentioned in the previous section, all of the previous attempts at developing explicit 3D
modeling capability for the method of characteristics have included parallel processing techniques to avoid unreasonable computational execution times\textsuperscript{65,66}. Acceleration and parallel processing have been placed beyond the scope of the present project and computational efficiency will be addressed instead via simplification of the 3D ray tracing procedures.

2.3 Derivation of the Steady State Method of Characteristics

Application of the method of characteristics to steady state problems has a long tradition and is relatively well understood. In this section, the most common formulation will be derived – the CACTUS method introduced above. This approach to the method of characteristics is chosen due to its broad applicability and current implementation in most state-of-the-art lattice physics codes. The derivation of this method of characteristics will be presented in detail and the key approximations and assumptions will be emphasized, as derivation of the time-dependent method of characteristics in Chapter 4 will utilize, with minimal modification, many of the same techniques.

2.3.1 Simplification of the General Neutron Transport Equation

The deterministic study of a neutron population in an arbitrary region of space begins with the general neutron transport equation, Equation (1.1). This form of the transport equation is continuous in the temporal, energy, angular, and spatial domains and reduction of the total transfer probability into scattering and fission components yields
\[
\frac{1}{\nu} \frac{\partial \Phi(r, \Omega, E, t)}{\partial t} + \Omega \cdot \nabla \Phi(r, \Omega, E, t) + \Sigma_f(r, E) \Phi(r, \Omega, E, t) \\
= \int \int \left[ \chi(E) \nu \Sigma_f(r, E) + \Sigma_s(r, \Omega \cdot \Omega', E \rightarrow E) \right] \Phi(r, \Omega', E', t) d\Omega' dE' + Q(r, \Omega, E, t), \tag{2.1}
\]

where

\[\chi(E) \equiv \text{Fraction of Fission Neutrons Born at Energy } E\]

\[\nu \equiv \text{Number of Neutrons Produced per Fission Event}\]

\[\Sigma_f(r; E) \equiv \text{Macroscopic Fission Cross-Section}\]

\[\Sigma_s(r, \Omega \cdot \Omega', E \rightarrow E) \equiv \text{Macroscopic Cross-Section for Neutron Scattering from } \Omega', E \text{ to } \Omega, E.\]

Application of Equation (2.1) to nuclear reactor physics problems generally involves situations where the time-scale is short (reactor kinetics) or where time-dependence is removed entirely (criticality, assembly transport, etc.), so time-dependence of the material properties has been neglected. Calculations with long time scales, such as burn-up calculations that compute changes in material composition of a reactor core over its lifetime, are beyond the scope of the present work. Reduction of Equation (2.1) to a form that is amenable to numerical computation will require the introduction of several additional simplifications that are described in the following sections.
2.3.2 Assumption: No External Sources

The primary phenomenon of interest to space-time reactor kinetics analysis is the behavior of the neutron population in a multiplicative environment; i.e., simulation of a nuclear reactor core that is operating in a critical, super-critical, or slightly sub-critical state. In such circumstances, the assumption that no external neutron source is present (i.e., sources of neutrons that result from events other than scattering and fission) is reliable. Neutron transport in highly sub-critical media, such as during radiation shielding computations or in simulations of research reactor designs that use a start-up source, would require the inclusion of an external source; thus, it may be necessary to include this term in a more generalized methodology. In the present development, however, \( Q(r, \Omega, E, t) \) is set to zero and neglected for the remainder of this derivation.

2.3.3 Assumption: Isotropy of Neutron Sources

Non-external neutron sources include neutrons born from fission and neutrons that undergo elastic and inelastic scattering collisions. For fission, the assumption of isotropy is usually acceptable; however, the same cannot be said for scattering and in many cases this simplification will lead to erroneous results. To minimize this error, a correction is introduced that preserves first-order anisotropy of the scattering kernel. Subtracting

\[
\iint_{\Omega_{s,1}} \left( r, E \right) \delta\left( \Omega - \Omega' \right) \delta\left( E - E' \right) \Phi\left( r, \Omega', E', t \right) d\Omega' dE' \quad \left\{ = \Sigma_{s,1} \left( r, E \right) \Phi\left( r, \Omega, E, t \right) \right\}
\]

(2.2)
from both sides of Equation (2.1), performing the left-hand side integration, and rearranging some terms yields

$$\frac{1}{\sqrt{\nu}} \frac{\partial \Phi \left( r, \Omega, E, t \right)}{\partial t} + \mathbf{\Omega} \cdot \nabla \Phi \left( r, \Omega, E, t \right) + \left[ \Sigma_{r} \left( r, E \right) - \Sigma_{s,1} \left( r, E \right) \right] \Phi \left( r, \Omega, E, t \right)$$

$$= \int \int \int \int \chi \left( E \right) \nu \Sigma_{r} \left( r, E', \Omega', t \right) \Phi \left( r, \Omega, E', t \right) d\Omega dE'$$

$$+ \int \int \int \Sigma_{s} \left( r, \Omega \cdot \Omega', E' \rightarrow E \right) - \Sigma_{s,1} \left( r, E' \right) \delta \left( \Omega - \Omega' \right) \delta \left( E - E' \right) \Phi \left( r, \Omega', E', t \right) d\Omega dE'$$

where \( \delta \left( \Omega - \Omega' \right) \) and \( \delta \left( E - E' \right) \) represent Dirac delta functions. By definition,

$$\Sigma_{\nu} \left( r, \Omega \cdot \Omega', E' \rightarrow E \right) \equiv \Sigma_{s} \left( r, \Omega \cdot \Omega', E' \rightarrow E \right) - \Sigma_{s,1} \left( r, E' \right) \delta \left( \Omega - \Omega' \right) \delta \left( E - E' \right)$$

(2.4)

or, where \( \mu_{0} = \Omega \cdot \Omega' = \cos \zeta \) for a scattering angle, \( \zeta \),

$$\Sigma_{\nu} \left( r, \mu_{0}, E' \rightarrow E \right) \equiv \Sigma_{s} \left( r, \mu_{0}, E' \rightarrow E \right) - \Sigma_{s,1} \left( r, E' \right) \delta \left( \mu_{0} - 1 \right) \delta \left( E - E' \right).$$

(2.5)

Expanding the individual terms from Equation (2.5) into Legendre polynomials and truncating after the first two terms yields

$$\Sigma_{\nu} \left( r, \mu_{0}, E' \rightarrow E \right) = \sum_{n=0}^{\infty} \left( 2n + 1 \right) \Sigma_{sr,n} \left( r, E' \rightarrow E \right) P_{n} \left( \mu_{0} \right)$$

(2.6)

$$= \Sigma_{sr,0} \left( r, E' \rightarrow E \right) + 3 \mu_{0} \Sigma_{sr,1} \left( r, E' \rightarrow E \right) + ...$$
\[ \Sigma_s(r, \mu_0, E' \rightarrow E) = \sum_{n=0}^{\infty} (2n+1) \Sigma_{s,n}(r, E' \rightarrow E) P_n(\mu_0) \]

(2.7)

\[ = \Sigma_{s,0}(r, E' \rightarrow E) + 3 \mu_0 \Sigma_{s,1}(r, E' \rightarrow E) + ... \]

and

\[ \Sigma_{s,1}(r, E') \delta(\mu_0 - 1) \delta(E - E') \]

(2.8)

\[ = \Sigma_{s,1}(r, E') \delta(E - E') + 3 \mu_0 \Sigma_{s,1}(r, E') \delta(E - E') + ... \]

If the P_0 and P_1 terms are equated via Equation (2.5), the result is

\[ \Sigma_{\sigma,0}(r, E' \rightarrow E) = \Sigma_{s,0}(r, E' \rightarrow E) - \Sigma_{s,1}(r, E') \delta(E - E') \]

(2.9)

and

\[ \Sigma_{\sigma,1}(r, E' \rightarrow E) = \Sigma_{s,1}(r, E' \rightarrow E) - \Sigma_{s,1}(r, E') \delta(E - E'). \]

(2.10)

Equation (2.10) is found to be zero when averaged over \( E' \), so that it can be neglected and scattering approximated by the term \( \Sigma_{\sigma,0}(r, E' \rightarrow E) \), which is the so-called “transport approximation.” Substitution of this approximation into Equation (2.3) yields
\[
\frac{1}{v} \frac{\partial \Phi(r, \Omega, E, t)}{\partial t} + \nabla \Phi(r, \Omega, E, t) + \Sigma_s(r, E) \Phi(r, \Omega, E, t) \\
= \int \left[ \chi(E') \nu \Sigma_s(r, E') + \Sigma_{s,0}(r, E \to E') \right] dE' \int \Phi(r, \Omega, E', t) d\Omega'.
\]

where

\[
\Sigma_s(r, E) \equiv \Sigma_i(r, E) - \Sigma_{s,1}(r, E) = \Sigma_u(r, E) + \Sigma_{s,0}(r, E) - \Sigma_{s,1}(r, E).
\]

The angular dependence can be removed from the integral on the right-hand side of Equation (2.11), yielding

\[
\frac{1}{v} \frac{\partial \Phi(r, \Omega, E, t)}{\partial t} + \nabla \Phi(r, \Omega, E, t) + \Sigma_s(r, E) \Phi(r, \Omega, E, t) \\
= \int \left[ \chi(E') \nu \Sigma_s(r, E') + \Sigma_s(r, E \to E') \right] \Phi(r, E', t) dE'.
\]

In Equation (2.12) the neutron scalar flux has been introduced,

\[
\phi(r, E, t) = \int \Phi(r, \Omega, E, t) d\Omega
\]

and the notation for the differential scattering cross section has been simplified to the more traditional form, \( \Sigma_s(r, E' \to E) \). The transport correction yields satisfactory results for many of the situations that are encountered in commercial light water reactor environments. In more exotic reactor configurations, however, this approximation will
fail to account for more significant anisotropic scattering components. Incorporation of an anisotropic scattering kernel into the method of characteristics is beyond the scope of the present work but would be a necessary component in most practical applications.

2.3.4 Discretization of the Energy Domain

The next stage in simplification of the transport equation is to remove the energy integral from the right-hand side of Equation (2.12). To do so, the multi-group energy discretization scheme is introduced, as is typical of most deterministic methods\(^3\). The entire energy spectrum is divided into a finite number of discrete energy groups, \(G\), where the upper and lower boundaries for an arbitrary energy group, \(g\), are defined to be \(E_g\) and \(E_{g-1}\), respectively. If the transport equation is integrated over the range of this energy group the result is

\[
\frac{1}{\nu} \frac{\partial \Phi^g(r, \Omega, t)}{\partial t} + \Omega \cdot \nabla \Phi^g(r, \Omega, t) + \Sigma_{tr}^g(r) \Phi^g(r, \Omega, t) = \Sigma_{sf}^g(r) \Phi^g(r, \Omega, t) + \chi^g \sum_{\nu} \nu \Sigma_{\nu}^g(r) \phi^{\nu}(r, t),
\]

where

\[
\Phi^g(r, \Omega, t) = \int_{E_{g-1}}^{E_g} \Phi(r, \Omega, E, t) dE
\]

(2.14)
\[
\phi^g (r,t) = \int_{E_g}^{E_{g-1}} \phi (r,E,t) \, dE
\]  

\[
\Sigma^g_{pr} (r) = \frac{\int_{E_g}^{E_{g-1}} \Sigma^g_{pr} (r,E) \Phi (r,\Omega,E,t) \, dE}{\int_{E_g}^{E_{g-1}} \Phi (r,\Omega,E,t) \, dE}
\]  

\[
\nu \Sigma^g_f (r) = \frac{\int_{E_g}^{E_{g-1}} \nu \Sigma^g_f (r,E) \Phi (r,\Omega,E,t) \, dE}{\int_{E_g}^{E_{g-1}} \Phi (r,\Omega,E,t) \, dE}
\]  

and

\[
\Sigma^{g\rightarrow g'}_s (r) = \frac{\int_{E_g}^{E_{g-1}} \Sigma^{g\rightarrow g'}_s (r,E) \Phi (r,\Omega,E,t) \, dE}{\int_{E_g}^{E_{g-1}} \Phi (r,\Omega,E,t) \, dE}.
\]

The precise number of discrete energy groups that is required to model the continuous energy domain will vary depending on the nature of the calculation to be performed. In lattice physics applications, for example, multiple computations are performed for the purposes of cross-section generation. Traditional group collapse procedures will begin with a fine-group structure that may contain hundreds or even thousands of energy groups. Subsequent calculations collapse this information to the few-group structure that
is required by most steady state or time-dependent reactor physics methodologies. Analyses of light water reactors will typically utilize a two-group structure, dividing the energy domain into fast and thermal energies, while calculations for fast reactors may include a six-group structure to more accurately account for phenomena that occur at energies above the thermal limit of ~0.5 eV.

2.3.5 Removal of the Time-Dependence

To transform the transport equation into steady state form, the time-derivative on the left-hand side of Equation (2.14) is set equal to zero; however, to ensure existence of a unique solution in a multiplying medium, the problem must be converted to an eigenvalue problem. This is accomplished via the introduction of the effective multiplication factor, $k_{\text{eff}}$. This eigenvalue modifies the parameter $\nu$, the average number of neutrons emitted by a fission event, such that

$$\nu \rightarrow \frac{\nu}{k_{\text{eff}}}.$$  \hspace{1cm} (2.20)

Theoretically, this transformation implies that the precise value of $\nu$ is variable and can be perturbed until a solution is found. With this change the time-independent transport equation becomes
\[
\Omega \cdot \nabla \Phi^g(r, \Omega) + \Sigma_{ir}^g(r) \Phi^g(r, \Omega) = \sum_g \Sigma_{g \rightarrow g}^\epsilon(r) \phi^g(r) + \frac{\chi^g}{k_n} \sum_g n \Sigma_{g}^\epsilon(r) \phi^g(r).
\] (2.21)

While appropriate for steady state applications of the method of characteristics, removal of the time-dependence is ill suited to a space-time kinetics application, as will be seen during the time-dependent method of characteristics derivation in Chapter 4.

2.3.6 The Discrete Ordinates Approximation

It was noted in Chapter 1 that the method of characteristics utilizes the same angular discretization scheme as in finite-differenced discrete ordinates codes. Decomposition of the angular domain into discrete ordinates implies that a finite set of \( M \) total directions is chosen and that a solution to the neutron angular flux is sought for these directions only,

\[
\Omega_m \cdot \nabla \Phi^g_m(r) + \Sigma_{ir}^g(r) \Phi^g_m(r) = \frac{1}{4\pi} \left[ \sum_g \Sigma_{g \rightarrow g}^\epsilon(r) \phi^g(r) + \frac{\chi^g}{k_n} \sum_g n \Sigma_{g}^\epsilon(r) \phi^g(r) \right],
\] (2.22)

where the subscript \( m \) is introduced to denote an arbitrary discrete direction. The neutron source term on the right hand side of Equation (2.22) has been divided by \( 4\pi \) – the surface area of a unit sphere – to yield the direction-dependent source term for an isotropic source, which is consistent with the transport approximation described above. The neutron scalar flux is now redefined to be
where weighting factors, $\omega_m$, have been introduced to account for all directions that are not included in the discrete direction set. These weights are normalized such that

$$\sum_{m=1}^{M} \omega_m = 1$$

and are effectively proportional to the area of the solid angle subtended by the range of directions accounted for by each particular ordinate. The selection of a set of discrete directions and their associated weights defines a quadrature set and is one of the most critical modeling steps for characteristics methods. Selection of quadrature will be discussed in more detail in Chapter 3.

### 2.3.7 Discretization of the Spatial Domain

Thus far the spatial domain has been expressed in generalized coordinates with neutron motion viewed from a neutral, fixed point-of-reference - the LAB coordinate system of classical physics. In the characteristics methodology a coordinate transformation is introduced whereby neutron motion is viewed instead from the point-of-view of the neutron-in-flight. In this Lagrangian coordinate system the three-dimensional dependent variable $r$ is replaced by a one-dimensional variable, $s_m$, the direction-dependent linear
path along which a neutron will propagate, dependent on direction as indicated by the index \(m\). The neutron streaming term from Equation (2.22) is thus restated as

\[
\mathbf{\Omega}_m \cdot \nabla \Phi_m^g \rightarrow \frac{d\Phi_m^g}{ds_m},
\]

so that streaming is described by the total derivative of the neutron angular flux along the path of motion. The transport equation in Lagrangian coordinates is given by the characteristic equation,

\[
\frac{d\Phi_m^g(s_m)}{ds_m} + \Sigma_{tr}^g(s_m) \Phi_m^g(s_m) = Q_m^g(s_m).
\]

Discretization of the Lagrangian spatial domain is pursued in two steps. The problem space is first divided into a series of smaller regions, such as a rectangular/cubic grid in Cartesian coordinates or annular sectors in cylindrical coordinates. Each region is further discretized via the characteristics approximation, whereby propagation paths are drawn across the system for each direction in the quadrature set and are assumed to describe all possible paths of neutron motion. In the most general form, there can be multiple tracks drawn through any particular problem cell, as shown in figure 2.1 for a two-dimensional Cartesian cell.
Characteristic paths account for only a small fraction of the total space in a cell. To account for the remaining area, each path is assigned a width (in 2D) or cross-sectional area (in 3D) to account for all possible paths of motion that are not explicitly analyzed.

When the sub-division of space into a set of cells and discretization via characteristics is applied to Equation (2.26), the characteristic equation becomes

\[
\frac{d\Phi_{m,n}^{i,g}(s_{m,n})}{ds_{m,n}} + \Sigma_t^{i,g}(s_{m,n})\Phi_{m,n}^{i,g}(s_{m,n}) = Q_{m,n}^{i,g}(s_{m,n}),
\]

(2.27)

where \( \Phi_{m,n}^{i,g}(s_{m,n}) \) represents the neutron angular flux at an arbitrary position, \( s_{m,n} \), along characteristic \( n \) in cell \( i \), where there are a total of \( N \) tracks in the cell and \( I \) cells in the
problem. The total neutron source term, \( Q_{m,n}^{i,g}(s_{m,n}) \), which is at this stage also track-dependent, expands to

\[
Q_{m,n}^{i,g}(s_{m,n}) = \frac{1}{4\pi} \left[ \sum_{g} \sum_{i} \phi_{m,n}(s_{m,n}) + \sum_{k} \sum_{i} \phi_{m,n}(s_{m,n}) \right].
\] (2.28)

The task of projecting characteristics across the grid of cells and computing the exact length of each track segment requires the development of a set of ray tracing procedures. Ray tracing is one of the most critical aspects of any method of characteristics and must be pursued with care to minimize computational expense. This issue will be discussed in detail in Chapter 3.

### 2.3.8 General Solution to the Characteristic Equation

Equation (2.28) is a first-order linear differential equation and can be solved analytically by integrating over the length of the track segment, \( s_{m,n} \), from the location where this track enters the cell to an arbitrary position along the track, yielding

\[
\Phi_{m,n}^{i,g}(s_{m,n}) = \Phi_{m,n}^{i,g}(s_{m,n}) \bigg|_{in} e^{-\int_{s_{m,n}}^{s_{m,n}} \sum_{i} \phi_{m,n}(s_{m,n}) ds'} + \int_{s_{m,n}}^{s_{m,n}} \phi_{m,n}(s_{m,n}) e^{-\int_{s'}^{s_{m,n}} \sum_{i} \phi_{m,n}(s_{m,n}) ds''} ds'.
\] (2.29)
where \( s_{m,n} \) is the location where a characteristic path enters a cell, \( \Phi_{m,n}^{i,g} \) is the value of the neutron angular flux at the same location, and \( \Phi_{m,n}^{i,g}(s_{m,n}) \) is the neutron angular flux at an arbitrary position along track \( s_{m,n} \) within cell \( i \). Resolution of the integrals in Equation (2.29) is complicated by the presence of the terms \( \Sigma_{tr}^{i,g} \) and \( Q_{m,n}^{i,g} \), which exhibit an explicit spatial (track) dependence. To reduce this solution further, approximations must be introduced to simplify these terms.

In all methods for solution of the transport equation, the material properties are treated as constant across a cell, so that

\[
\Sigma_{tr}^{i,g}(s_{m,n}) \Rightarrow \Sigma_{tr}^{i,g}
\]  

(2.30)

and Equation (2.29) reduces to

\[
\Phi_{m,n}^{i,g}(s_{m,n}) = \Phi_{m,n}^{i,g}|_{in} e^{-\Sigma_{tr}^{i,g}(s_{m,n} - s_{m,n}^{'})} + \int_{s_{m,n}^{'}}^{s_{m,n}} Q_{m,n}^{i,g}(s^{'}) e^{-\Sigma_{tr}^{i,g}(s_{m,n} - s^{'})} ds^{'}.
\]  

(2.31)

While the assumption of cell-constant material properties is generally acceptable, spatial dependence of the total neutron source term, \( Q_{m,n}^{i,g}(s) \), can affect the accuracy of a method of characteristics transport solution and must be considered with care. In
principle, a wide variety of functions could be used to represent this spatial dependence; however, two approximations are employed in current method of characteristics implementations. The first approach treats the neutron source term as uniform across a given cell so that all track dependence is removed,

\[ Q_{m,n}^{i,g}(s_{m,n}) \Rightarrow \overline{Q}_{i,g} \],

(2.32)

and

\[ \Phi_{m,n}^{i,g}(s_{m,n}) = \Phi_{m,n}^{i,g} \bigg|_{m} e^{-\Sigma_{s}^{i,g} (s_{m,n} - s_{m,n})} + \overline{Q}_{i,g} \bigg( 1 - e^{-\Sigma_{s}^{i,g} (s_{m,n} - s_{m,n})} \bigg) \].

(2.33)

Another approximation treats the neutron source term as a linear function of position along the characteristic path, \( s_{m,n} \). The linear source term approximation is typically utilized in arbitrarily high-order characteristics formulations\(^{76,77,78,79}\) that are applied to certain types of shielding and deep penetration problems. While a linear neutron source yields slightly improved accuracy, the constant source term approximation has been demonstrated to be sufficient for most lattice physics applications. Moreover, improved accuracy can be achieved from the constant source term approximation by simply reducing the size of the computational mesh. It is presumed, therefore, that the constant neutron source term approximation will be similarly applicable to a space-time kinetics methodology and higher-order approximations will be reserved for future work.
2.3.9 Calculation of the Neutron Scalar Flux

In Equation (2.33), the solution to the characteristic equation is given in terms of an arbitrary location; however the computational points of interest will be the location where each track exits the cell. The neutron angular flux at this location is given by

\[
\Phi_{m,n}^{i,g} = \Phi_{in}^{i,g} e^{-\Sigma tr^i_{m,n} \Delta s_{m,n}} + \frac{Q_{m,n}^{i,g}}{\Sigma tr} \left( 1 - e^{-\Sigma tr^i_{m,n} \Delta s_{m,n}} \right),
\] (2.34)

where \(\Delta s_{m,n}\) represents the total length of track \(n\) that is projected along direction \(m\).

Equation (2.34) can be solved for each track endpoint in the full system model to yield the distribution of direction-dependent neutron angular flux values. In practical application, however, the desired output will be the direction-independent cell-wise neutron scalar flux distribution.

The neutron scalar flux distribution can be computed from the neutron angular flux distribution in three steps. First, the track-averaged neutron angular flux, \(\overline{\Phi}_{m,n}^{i,g}\), is calculated for each track by integrating along the streaming path:

\[
\left( s_{m,n} \big|_{out} - s_{m,n} \big|_{in} \right) \overline{\Phi}_{m,n}^{i,g} = \int_{s_{m,n}}^{s_{m,n} \big|_{out}} \Phi_{m,n}^{i,g} (s) \, ds. \] (2.35)
Inserting Equation (2.33) into Equation (2.35) and solving the integral yields

\[
\Phi_{m,n}^{i,g} = \frac{Q_{m}^{i,g}}{\sum_{\nu}^{i,g}} + \frac{\Phi_{m,n}^{i,g} - \Phi_{m,n}^{i,g} \bigg|_{\text{out}}}{\sum_{\nu}^{i,g} \Delta s_{m,n}}.
\] (2.36)

Next, the cell-average neutron angular flux is calculated via summation for all tracks in cell \( i \) along direction \( m \),

\[
\Phi_{m}^{i,g} = \sum_{n=1}^{N} \frac{\Phi_{m,n}^{i,g} \Delta s_{m,n} \delta A_{m}}{\sum_{n=1}^{N} \Delta s_{m,n} \delta A_{m}},
\] (2.37)

where \( \delta A_{m} \) represents the separation between characteristics, as will be described in detail in Chapter 3. Finally, the neutron scalar flux can be computed via summation over all directions in the quadrature set,

\[
\phi^{i,g} = \int_{4\pi} \Phi^{i,g} (\Omega) d\Omega = 4\pi \sum_{m} \Phi_{m}^{i,g} \omega_{m}.
\] (2.38)

It is the neutron scalar flux that is of primary interest to any steady state transport computation and additional information can be derived from this known steady state dependent variable.
CHAPTER 3:

A RAY TRACING METHODOLOGY FOR

EXPLICIT THREE-DIMENSIONAL

CARTESIAN GEOMETRY

The steady state formulation for the method of characteristics that has been derived and discussed in Chapter 2 can be applied directly to any type of geometry. Traditional lattice physics applications apply one- and two-dimensional geometries to Cartesian, cylindrical, hexagonal, and even spherical coordinates systems. In alternative method of characteristics formulations, irregular or random spatial discretization schemes may also be employed. This spatial flexibility is a direct result of the coordinate transformation that lies at the heart of the method of characteristics. The characteristic equation, Equation (2.26), views neutron transport in one-spatial dimension – the linear path along which neutrons are assumed to travel – and requires only boundary conditions and the length of each track, $\Delta s$, as input. All remaining spatial details, such as the orientation and separation of individual tracks and the possible directions of neutron travel, are decoupled from the transport calculation and considered instead during a separate ray tracing calculation that generates the required track length information. Thus, whereas the method of characteristics transport calculation is highly simplified, ray tracing can be quite complex, depends strongly on the coordinate system of interest, and may require sophisticated techniques to minimize computational expense.
The focus of this chapter will be the development of three-dimensional ray tracing for Cartesian geometry. Ray tracing procedures will be far simpler in Cartesian coordinates than for any of the alternative coordinate systems mentioned above, making this an appropriate choice for new methods development. As was discussed in Chapter 2, previous attempts to develop ray tracing in three dimensions have pursued one of two approaches. Hybrid techniques generate a full spatial model by coupling multiple lower order (one- and/or two-dimensional) solutions, requiring no ray tracing procedures beyond what is currently available. Explicit modeling of three-dimensional space achieves a more accurate solution than do hybrid methods at the cost of a more significant computational expense.

The ray tracing procedures presented below will combine the basic CACTUS ray tracing methodology with the direct neutron path linking technique developed by Kosaka and Saji, while utilizing an explicit description of three-dimensional space that was suggested originally by I. R. Suslov. This approach will diverge from traditional lattice physics methods by introducing additional simplifying assumptions to the problem space that would be ill suited for an assembly transport calculation but are more appropriate in the context of space-time reactor kinetics. In so doing, three-dimensional ray tracing is rendered computationally efficient without recourse to the more sophisticated techniques that would be required by ray tracing for a large assembly transport calculation.
3.1 Discretization of the Spatial and Angular Domains

The first step in generating a set of characteristics for any particular system of interest is to define the grid of cells across which these characteristics will be projected. This grid will utilize one of a variety of available coordinate systems in one dimension (1D), two dimensions (2D), or three dimensions (3D). The choice of coordinate system is generally based on the design of the reactor to be modeled; for example, Cartesian or cylindrical coordinates are typically applied to commercial light water reactors, which are comprised of round fuel pins bundled into square fuel assemblies; certain fast reactors and the VVER reactor design utilize a hexagonal reactor shape, for which a hexagonal coordinate system is appropriate; finally, some research reactors may contain a spherical core to which a spherical coordinate system can be applied. A computational reactor analysis tool should be capable of modeling a wide variety of reactor types and may include several or all of the possible coordinate systems. As mentioned above, however, ray tracing is most easily defined for a Cartesian coordinate system, making this an appropriate choice for new methods development.

Two simplifications are introduced to the Cartesian grid for an arbitrary problem space. First, a constraint of uniform cell dimensions is imposed; that is, all cells are required to be of the same size and shape. Figure 3.1 shows an arbitrary 3D Cartesian cell, $i$, with x-, y-, and z-dimensions $dx$, $dy$, and $dz$, respectively.
While the cell in figure 3.1 is cubic, this is not required by the constraint of constant cell dimensions; i.e., $dx$, $dy$, and $dz$ must be constant, but need not equal each other. It is assumed further that each discrete cell will be materially homogeneous, which is consistent with the constant neutron source term approximation discussed in Chapter 2.

The assumption of uniform homogeneous cells simplifies ray tracing beyond what would typically be employed in an assembly transport calculation, where a detailed description of pin cell internals must be utilized to compute an accurate transport solution. Internal cell details will be less important to the performance of a space-time reactor kinetics calculation, however, and are initially disregarded. To minimize the loss of accuracy associated with such an oversimplification a fine-mesh grid is utilized, where cell
dimensions are typically no larger than one half of the transport mean free path, \( \frac{1}{\Sigma_r} \),
and will depend strongly on the materials present.

Further discretization of the 3D Cartesian cell requires that a set of characteristics be
projected across the cell for each discrete direction, \( \Omega \), that is described by a pair of
angles – the azimuthal angle, \( \phi \), and the polar angle, \( \theta \) – as defined in figure 3.2.

![Definition of the Characteristic Direction](image)

![Definition of the Azimuthal and Polar Planes](image)

Figure 3.2: a) Definition of the Characteristic Direction
b) Definition of the Azimuthal and Polar Planes
The azimuthal angle is defined to be the angle between the x-axis and a projection of the 3D characteristic onto the x-y plane. This plane is also called the *azimuthal plane*†. The polar angle is defined to be the angle between the z-axis and the 3D characteristic, both of which lie within the *polar plane*. Figure 3.2b shows that the azimuthal and polar planes lie perpendicular to one another, and the implied separation of the 3D direction space into a pair of 2D planes will be utilized below to facilitate 3D ray tracing.

A few additional constraints are placed on the characteristics themselves to further simplify the system model: (1) characteristics are required to be periodic and symmetric, (2) characteristics are required to connect directly across internal cell boundaries, and (3) specular reflection of tracks at external cell boundaries is enforced. These constraints are consistent with DNPL and yield a set of natural boundary conditions for the characteristic equation at all internal and external cell boundaries. The constraints of optical reflection and periodicity yield a cell that is rotationally invariant, thus ray tracing need be performed for directions in the first octant only; i.e., for angles $0 < \phi < \pi/2$ and $0 < \theta < \pi/2$. The track set for octants 2 through 8 can be generated simply by rotating the fully discretized octant 1 cell into each of the other octants, while holding $\phi$ and $\theta$ constant. Octant symmetry is consistent with traditional methods for generation of a discrete ordinates quadrature set. When the requirements of octant symmetry are combined with the constraint of uniform cell dimensions the set of track segments within a particular cell will be identical for all cells in the system model and, as a result, it is possible to perform ray tracing for a single representative cell and then apply the results uniformly to all cells.

† Italics are used throughout this chapter to emphasize the definition of terms that are of fundamental importance to a discussion of ray tracing in three-dimensions.
in the system model. By limiting the size of a 3D track set to a single representative cell and one eighth of the possible directions, this new approach to explicit 3D ray tracing is more efficient than most existing ray tracing techniques.

3.2 Selection of a Quadrature Set

The method of characteristics utilizes the discrete ordinates angular discretization approximation as described in Chapter 2. The angular domain of the transport equation is divided into a set of \( N \) discrete directions for neutron travel, each of which is associated with a weight, \( \omega \). In principle, substantial freedom is available in the selection of the directions and weights that comprise a quadrature set, provided that the set exhibit octant symmetry to ensure direct connection at internal cell boundaries and optical reflection at external system boundaries.

Two of the simplest useful quadrature sets are known as the “equal angles” and “equal weights” sets\(^8\). In an equal angles formulation the angles associated with each direction are assumed to be equal for the all directions in the octant. Weights are derived for each direction using an appropriate normalization condition. In the equal weights formulation, on the other hand, the \( N \) total directions are distributed in the octant so as to have equal weights and, subsequently, the associated angles are derived from a similar normalization condition. In both cases the result is an easily derived and reasonably accurate set of directions, and such techniques have been employed in many assembly transport codes.
A more accurate quadrature set that is commonly applied to both the method of characteristics and finite differenced discrete ordinates codes is level symmetric quadrature\textsuperscript{24}. A level symmetric direction set of type $S_N$ is defined to have $N/2$ levels for which an arc is projected across the surface of the unit direction sphere as shown in figure 3.3 for the case of $S_4$ quadrature.

![Figure 3.3: Definition of a Level-Symmetric $S_4$ Quadrature Set](image)
A discrete direction is defined to connect the origin with a point on the surface of the unit direction sphere where all three of these arcs intersect. Directions in figure 3.3 are expressed in terms of the three direction cosines, $\mu$, $\eta$, and $\xi$, which can be transformed into the two-angle representation of figure 3.2 via

$$
\begin{align*}
\mu &= \cos \theta \\
\eta &= \sqrt{1 - \mu^2} \cos \phi \\
\xi &= \sqrt{1 - \mu^2} \sin \phi
\end{align*}
$$

(3.1)

In general, the direction cosines are related by the requirement that

$$
\mu^2 + \eta^2 + \xi^2 = 1
$$

(3.2)

so that there are two degrees of freedom in selecting the quadrature set. Level symmetry requires that the same $N/2$ values for the direction cosine be assigned uniformly to each of the three axes and

$$
\mu_j = \eta_j = \xi_j \quad ; \quad j = 1, N/2.
$$

(3.3)

In level symmetry only one degree of freedom – the selection of a value for $\mu_1$ – is required to generate a full quadrature set. In most of the problems used throughout this work, the $S_4$ level symmetric quadrature is employed; however orders of $S_6$, $S_8$, $S_{12}$, and $S_{16}$ will also be investigated. The azimuthal and polar angles for each of these quadrature
sets have been derived from values for the direction cosines that are tabulated in Reference 24.

In addition to the azimuthal and polar angles and direction weights that comprise a traditional discrete ordinates quadrature set, the method of characteristics also requires definition of the distance that separates adjacent characteristics. In 3D geometry, this definition is composed of two parameters – the azimuthal track separation, $\delta A$, which describes the distance between adjacent projections of the 3D characteristics on the azimuthal plane, and the polar track spacing, $\delta P$, which describes the separation between adjacent characteristics on the polar plane. When combined these two parameters describe an effective cross-sectional area that can be associated with each characteristic and, as the weights in a quadrature set are incorporated to account for all possible directions not explicitly modeled, so does this cross-sectional track area account for all tracks not explicitly modeled by the method of characteristics spatial discretization. In practice, track separation and cell dimensions are the parameters most accessible to the user and can be varied to achieve a desired refinement to the system model.

### 3.3 Overview of the 3D Ray Tracing Procedures

The principles and constraints described in Sections 3.1 and 3.2 yield the information that is required for generation of a method of characteristics track set: the cell dimensions ($dx$, $dy$, and $dz$), the azimuthal and polar track separations ($\delta A$ and $\delta P$), a quadrature set ($\phi$, $\theta$, and $\omega$), and appropriately defined boundary conditions. The task of ray tracing is to
project characteristics across the system for each direction in the quadrature set. When these global characteristics are overlaid with the Cartesian grid of cells a larger set of partial track segments will result. The length of each track segment, \( \Delta s \), with endpoints at the locations where a characteristic enters and exits an arbitrary cell, is the primary parameter required for solution of the characteristic equation, Equation (2.26). Additionally, boundary conditions must be translated into numerical information that is usable by the transport computation.

Explicit modeling of 3D cell geometry is accomplished by decomposing the 3D cell into a coupled set of 2D planes, the azimuthal and polar planes defined in figure 3.2b, and subsequent application of traditional 2D ray tracing procedures to each of these sub-domains. First, an azimuthal cross-section of the cell is considered, shown in figure 3.4.

![Figure 3.4: Azimuthal (x-y) Plane of the 3D Cell](image)
Each of the 2D azimuthal tracks shown in figure 3.4 is assumed to be a line of intersection between the azimuthal plane and a perpendicular polar plane, as demonstrated in figure 3.5.

![Figure 3.5: Intersection of Azimuthal and Polar Planes in the 3D Cell](image)

All of the 3D characteristics are required to lie in one of these polar planes, so that the 3D track lengths can be generated in a two step manner: (1) perform 2D ray tracing on the azimuthal plane of the cell to determine the dimension and orientation of each polar plane and (2) perform 2D ray tracing on these polar planes to compute the length of each track segment within each polar plane. The result is a 3D ray tracing procedure that requires only the systematic implementation of previously developed and well-understood 2D ray tracing procedures.
3.4 Ray Tracing in the Azimuthal Plane

The first stage in calculating track lengths for 3D characteristics is to project these tracks onto the azimuthal (x-y) plane and compute the lengths and coupling relationships for these azimuthal projections. Figure 3.6 shows that the azimuthal track lengths calculated in this first stage will provide the width for each of the planar sections that are analyzed in Section 3.5. The problem of calculating track lengths for the azimuthal projections is virtually identical to the 2D ray tracing problem that is encountered by traditional lattice physics codes and is thus solvable by pre-existing 2D ray tracing methods. The procedures presented in this section are derived directly from the “CACTUS” ray tracing methodology discussed in Chapter 2 and will parallel the methods used in modern lattice physics codes such as CASMO-4, WIMS, and CRX. 2D ray tracing is thereby performed in four steps: (1) the azimuthal angles and track separation are corrected to enforce periodicity; (2) the entry location for each track in the azimuthal set is determined; (3) the length of each azimuthal track is computed; and (4) a coupling relationship to connect adjacent tracks across cell boundaries is derived.

3.4.1 Correction of the Azimuthal Angles and Azimuthal Track Separation

When a 3D characteristic is projected onto the azimuthal plane of the representative cell for an arbitrary discrete direction, m, this projection will cross a certain number of cell lengths and widths before aligning to its original relative cell location, yielding the periodic azimuthal unit as defined in figure 3.6.
The number of cells spanned by the periodic azimuthal unit in the x-direction will be

$$N_{x,m} = \frac{dy \cdot |\cos \varphi_m|}{\delta A_m},$$  \hspace{1cm} (3.4)$$

and in the y-direction will be

$$N_{y,m} = \frac{dx \cdot |\sin \varphi_m|}{\delta A_m}. \hspace{1cm} (3.5)$$

For any particular quadrature set, Equations (3.4) and (3.5) will typically yield non-integer results, which is unacceptable because non-integer spans can yield an azimuthal period that is arbitrarily large. In most cases this period will be much larger than the
dimensions of the system being modeled and of no practical use. If non-integer spans are rounded to integer values, the resulting azimuthal period will be smaller than the system dimensions, ensuring that the constraint of periodicity is appropriately imposed.

Rounding of the results from Equations (3.4) and (3.5), for the reasons described above, yields the integer spans, $\hat{N}_{x,m}$ and $\hat{N}_{y,m}$ that are also directly related to the number of tracks that will cross the 2D cell. The span in the x-direction, $\hat{N}_{x,m}$, is identified as the number of tracks that enter along the –x edge of the 2D cell. Similarly the y-direction span, $\hat{N}_{y,m}$, is identified with the number of tracks that enter along the –y cell edge. Values for $\varphi_m$ and $\delta A_m$ must be recomputed to generate a modified quadrature set that yields the desired periodic track set. Using simple trigonometric relationships, the corrected azimuthal angle, $\hat{\varphi}_m$, is found to be

$$\hat{\varphi}_m = \tan^{-1}\left(\frac{\hat{N}_{y,m} \cdot dy}{\hat{N}_{x,m} \cdot dx}\right). \quad (3.6)$$

Correction of the azimuthal track spacing, $\delta A_m \rightarrow \delta \hat{A}_m$, is accomplished via conservation of cell area. The area of a 2D cell can be approximated by assuming that each track length, $s_{n,m}$, and the corresponding track width, $\delta \hat{A}_m$, yield the set of rectangular strips shown in figure 3.7.
For cell area to be conserved, the sum of these rectangular sections should exactly equal the total cell area:

\[ S_m^{\text{tot}} \cdot \delta \hat{A}_m = dx \cdot dy, \]  

(3.7)

where \( S_m^{\text{tot}} \) is the sum of all the track lengths in the representative cell,

\[ S_m^{\text{tot}} \equiv \sum_{n=1}^{\hat{n}_m} s_{n,m}, \]  

(3.8)
and the total number of tracks in the cell is given by $\hat{N}_{a,m} = \hat{N}_{x,m} + \hat{N}_{y,m}$. The parameters $\hat{N}_{x,m}$, $\hat{N}_{y,m}$, and $S_{m}^{\text{tot}}$ can be related via the Pythagorean theorem, as seen in figure 3.6, so that the total track length can be expressed as

$$S_{m}^{\text{tot}} = \sqrt{\left(\hat{N}_{x,m} \cdot dx\right)^2 + \left(\hat{N}_{y,m} \cdot dy\right)^2},$$  \hspace{1cm} (3.9)

which, when substituted into Equation (3.7), yields,

$$\delta \hat{A}_{m} = \frac{dx \cdot dy}{\sqrt{\left(\hat{N}_{x,m} \cdot dx\right)^2 + \left(\hat{N}_{y,m} \cdot dy\right)^2}}. \hspace{1cm} (3.10)$$

In the example shown in figure 3.7, the total track length, $S_{m}^{\text{tot}}$, is equal to the length of the azimuthal periodic unit, $dS_{m}$; however, this will not be true for all system geometries.

Figure 3.8 illustrates a system where there are multiple periodic units.
The number of azimuthal periodic units in a particular system can be determined by finding common multiples of $\hat{N}_{x,m}$ and $\hat{N}_{y,m}$ as calculated via Equations (3.4) and (3.5).

In figure 3.6, where $\hat{N}_{x,m} = 3$ and $\hat{N}_{y,m} = 2$, the common multiple is 1, yielding only one periodic unit. In figure 3.8, where $\hat{N}_{x,m} = 2$ and $\hat{N}_{y,m} = 2$, the common multiple is 2, and thus there will be two periodic units. When there are multiple periodic units, each of these units will have equal periodic length, $dS_m$ and it follows that

$$S_m^{tot} = \rho_m \cdot dS_m,$$  \hspace{1cm} (3.11)
where $\rho_m$ is the number of azimuthal periodic units for direction $m$. Close examination of figure 3.8 shows that each periodic unit will be repeated $\rho_m$ times in crossing $\hat{N}_{x,m}$ and $\hat{N}_{y,m}$ integral numbers of cells; thus, Equations (3.9) and (3.10) need not be modified to account for multiple periodic units. In fact, the presence of multiple periodic units can be safely neglected in performing 2D ray tracing. In 3D ray tracing, however, calculation of $\rho_m$ via the common multiple analysis described above will be required to implement polar plane ray tracing procedures.

3.4.2 Calculation of Azimuthal Track Entry Locations

Calculation of the length of each track segment in the 2D azimuthal plane will require information regarding the entry and exit locations for each track along the cell perimeter. As track periodicity implies that entry and exit locations will align exactly, it is sufficient to compute only the entry locations. Initially, the azimuthal track spacing, $\delta \hat{A}_m$, is projected to the cell boundaries, as shown in figure 3.9.
These projections yield the distance between successive track entries along the \(-x\) cell-edge,

\[
\delta x_m = \frac{\delta A_m}{\cos \phi_m},
\]  
(3.12)

and along the \(-y\) cell-edge,

\[
\delta y_m = \frac{\delta A_m}{\sin \phi_m}.
\]  
(3.13)

Having evaluated the projected track spacings, \(\delta x_m\) and \(\delta y_m\), each track entry position can now be calculated. For directions in the first octant, these locations are defined relative to an origin that is placed at the lower-left corner of the cell and are calculated in
sequence from the lower-right corner to the upper-left corner in a clockwise direction. An example is illustrated in figure 3.10.

1. The first azimuthal track origin is placed along the bottom (-y) cell-edge, at position $y_{1,m}^{\text{in}}$, where

$$y_{1,m}^{\text{in}} = \frac{d x + (\hat{N}_{y,m} - 1) \cdot \delta y_m}{2}.$$  \hspace{1cm} (3.14)

This placement is required to ensure that there will be no corner crossings in the azimuthal plane, and also that the azimuthal track set will be symmetric.

2. The second track origin, $y_{2,m}^{\text{in}}$, is placed at a distance $\delta y_m$ to the left of the first track origin, i.e.:

$$y_{2,m}^{\text{in}} = y_{1,m}^{\text{in}} - \delta y_m.$$  \hspace{1cm} (3.15)

3. All remaining track origins along the –y edge are placed from right to left as in Step 2, where there are by definition $\hat{N}_{y,m}$ tracks originating along this edge.

4. The next track, $x_{1,m}^{\text{in}}$, is placed along the left (–x) cell-edge at a distance $\delta x_m^i$ from the bottom-left corner, where
\[
\delta x_m = \left( \delta y_m - y_{N_{y,m,m}}^{in} \right) \cdot \tan \phi_m
\] (3.16)

and \( y_{N_{y,m,m}}^{in} \) represents the last track entry along the \(-y\) cell edge.

5. The next track origin along the \(-x\) edge, \( x_{2,m}^{in} \), is placed at a distance \( \delta x_m \) above the previous origin, \( x_{1,m}^{in} \), i.e.:

\[
x_{2,m}^{in} = x_{1,m}^{in} + \delta x_m.
\] (3.17)

6. All remaining track origins along the \(-x\) edge are placed from bottom to top as in Step 5, where there are by definition \( \hat{N}_{x,m} \) tracks along this edge.

---

**Figure 3.10: Example of the Azimuthal Track Origin Calculation Procedure**
Imposition of symmetry on the tracks in the azimuthal cell implies that $\hat{N}_{x,m}$ represents both the number of tracks entering the cell along the $-x$ cell edge and the number of tracks exiting along the $+x$ cell edge. A similar observation can be made regarding $\hat{N}_{y,m}$. As a result, the set of track entry locations calculated above will be symmetrically equal to a set of track exit locations along the opposite cell edges.

### 3.4.3 Calculation of Azimuthal Track Lengths

Sweeping back across the track origins, the length of each track in the azimuthal plane is now computed. This procedure, an example of which is shown in figure 3.11, is completed as follows:

1. For tracks that enter through the bottom ($-y$) edge, the maximum track length will occur if the track exits through the opposite ($+y$) edge, as seen in figure 3.11a. This maximum length, $s_{m}^{y}_{\text{max}}$, is calculated according to

\[ s_{m}^{y}_{\text{max}} = \frac{dy}{\sin \hat{\phi}_{m}}. \] (3.18)

2. For each origin along the $-y$ edge, it is assumed that the track exits through the right ($+x$) edge (see figure 3.11b) and the track length is given by
\[ s_{n,m} = \frac{dx - y_{n,m}^{in}}{\left| \cos \phi_n \right|}, \]  

(3.19)

where \( y_{n,m}^{in} \) is the track entry location for track \( n \).

3. If the track length as calculated in Step 2 is larger than \( s_{m}^{\gamma} \big|_{\text{max}} \), then it is deduced that the track exits through the top (+\( y \)) edge, as described by figure 3.11c, and the track length is set equal to \( s_{m}^{\gamma} \big|_{\text{max}} \).

4. Step 2 is repeated in sequence for each track that enters through the –\( y \) edge until the condition stated in Step 3 is met. Any/all remaining tracks that enter through the –\( y \) edge are set equal to \( s_{m}^{\gamma} \big|_{\text{max}} \).

5. For tracks that enter through the left (-\( x \)) edge, the maximum track length will occur, again, if the track exits through the opposite (+\( x \)) edge, as seen in figure 3.11d. In this case the maximum length, \( s_{m}^{\delta} \big|_{\text{max}} \), is calculated according to

\[ s_{m}^{\delta} \big|_{\text{max}} = \frac{dx}{\left| \cos \phi_m \right|}. \]  

(3.20)

6. For each origin along the -\( x \) edge, it is assumed that the track leaves through the top (+\( y \)) edge (see figure 3.11e) and the track length is given by
\[ s_{n,m} = \frac{d_y - x_{n,m}^{\text{in}}}{|\sin \hat{\phi}_m|}, \]  

(3.21)

where \( x_{n,m}^{\text{in}} \) is the track entry location for track \( n \).

7. If the track length as calculated in Step 6 is larger than \( s_m^x \big|_{\text{max}} \), then it is deduced that the track exits through the +x edge, as described by figure 3.11f, and the track length is set equal to \( s_m^x \big|_{\text{max}} \).

8. Step 6 is repeated in sequence for each track that enters through the –x edge until the condition stated in Step 7 is met. Any/all remaining tracks that enter through the –x edge are set equal to \( s_m^x \big|_{\text{max}} \).
3.4.4 Derivation of Azimuthal Track Coupling Relationships

The constraint that tracks connect directly across cell boundaries requires that a set of coupling relationships be derived. These coupling relationships will be used to determine the precise sequence of polar planar sections as they are combined to generate the polar cell, as described in Section 3.5. More importantly, the coupling relationships are required during the method of characteristics computation to specify the internal and external cell boundary conditions for each characteristic. An example of such coupling is demonstrated in figure 3.12.
Figure 3.12 shows that for tracks in the first octant, each track will enter cell \((i,j)\) through either the \(-x\) or \(-y\) cell edge and exit through either the \(+x\) or \(+y\) cell edge. By tracing each track across the cell in the direction of travel, it is possible to link these tracks as follows:

- Track 1 in cell \((i, j)\) connects to Track 3 in cell \((i+1, j)\).
- Track 2 in cell \((i, j)\) connects to Track 4 in cell \((i+1, j)\).
- Track 3 in cell \((i, j)\) connects to Track 5 in cell \((i+1, j)\).
- Track 4 in cell \((i, j)\) connects to Track 1 in cell \((i, j+1)\).
- Track 5 in cell \((i, j)\) connects to Track 2 in cell \((i, j+1)\).
If these tracks are aligned in linear sequence across adjacent cells, the coupling is

- Track 1 connects to Track 3.
- Track 3 connects to Track 5.
- Track 5 connects to Track 2.
- Track 2 connects to Track 4.
- Track 4 connects to Track 1.
- and so on…

A periodic sequence is thus defined and, as figure 3.6 shows, this is exactly the sequence that is used to define the periodic azimuthal unit. Figure 3.13 illustrates the coupling where there are multiple azimuthal units.

![Diagram of azimuthal coupling](image)

Figure 3.13: Demonstration of Azimuthal Coupling for Multiple Periodic Units
In this example there are two periodic azimuthal units and the coupling will be as follows:

- Track 1 in cell \((i, j)\) connects to Track 3 in cell \((i+1, j)\).
- Track 2 in cell \((i, j)\) connects to Track 4 in cell \((i+1, j)\).
- Track 3 in cell \((i, j)\) connects to Track 1 in cell \((i, j+1)\).
- Track 4 in cell \((i, j)\) connects to Track 2 in cell \((i, j+1)\).

This example will thus yield two periodic sequences, where the first sequence is

- Track 1 connects to Track 3.
- Track 3 connects to Track 1.
- And so on…

and the second sequence is

- Track 2 connects to Track 4.
- Track 4 connects to Track 2.
- And so on…

Each of these periodic sequences will correspond to one instance of the periodic azimuthal unit, which for this example is described in figure 3.8.
It is desirable, of course, to utilize an analytical formulation for generating the coupling relationships in an arbitrary azimuthal cell. In locating the track entry positions in Section 3.3.2, it was demonstrated that \( \hat{N}_{x,j} \) tracks will enter along the \(-x\) cell-edge, \( \hat{N}_{y,j} \) tracks will enter along the \(-y\) cell-edge and, due to the imposition of symmetry on the tracks within the azimuthal cell, \( \hat{N}_{x,j} \) tracks will exit along the \(+x\) cell-edge, and \( \hat{N}_{y,j} \) tracks will exit along the \(+y\) cell-edge. If the total set of tracks is consistently numbered in a clockwise direction from the lower-right corner of the cell to the upper-right corner of the cell, then the following coupling relationships will hold for all azimuthal track sets:

\[
\begin{align*}
\text{Couples TO} & \quad \text{Couples FROM} \\
\mathcal{S}_n \rightarrow \mathcal{S}_{n+\hat{N}_x} & \quad \text{for all tracks } 1 \leq n \leq \hat{N}_x \\
\mathcal{S}_n \rightarrow \mathcal{S}_{n-\hat{N}_y} & \quad \text{for all tracks } \left(\hat{N}_x + 1\right) \leq n \leq \left(\hat{N}_x + \hat{N}_y\right) \\
\mathcal{S}_n \leftarrow \mathcal{S}_{n+\hat{N}_y} & \quad \text{for all tracks } 1 \leq n \leq \hat{N}_y \\
\mathcal{S}_n \leftarrow \mathcal{S}_{n-\hat{N}_x} & \quad \text{for all tracks } \left(\hat{N}_y + 1\right) \leq n \leq \left(\hat{N}_y + \hat{N}_x\right).
\end{align*}
\]

Azimuthal track lengths and the track coupling information are preserved for use in the remaining ray tracing procedures, while azimuthal entry locations, \( x_{n,m}^{\text{in}} \) and \( y_{n,m}^{\text{in}} \), and the projected track separations, \( \delta x_m \) and \( \delta y_m \), are no longer required and can be discarded.
3.5 Ray Tracing in the Polar Plane

The track segments that are defined in Section 3.3 for the azimuthal plane are assumed to be lines of intersection with a set of vertical planar sections. The second stage in the calculation of 3D characteristics is to execute a similar set of 2D ray tracing procedures to compute track lengths for the final 3D characteristics as they cross these polar planes. In this case, however, several modifications must be made to the standard 2D ray tracing procedures due to differences between the azimuthal and polar planes of the 3D cell.

Specifically, whereas the azimuthal plane is based on an inherently periodic 2D cell, the polar plane is comprised of multiple planes of non-uniform dimension. Application of 2D ray tracing procedures to the polar plane requires that the polar planar sections be organized into a periodic 2D cell. Subsequent to this definition, polar track lengths are computed in four steps: (1) polar angles and polar track separation are corrected to enforce periodicity; (2) the standard 2D ray tracing procedures are applied to the entire polar cell, yielding global polar track lengths and coupling relationships; (3) these global track lengths are subdivided to generate the final track segments for each individual polar planar section; and (4) the final 3D track coupling relationships are derived.

3.5.1 Definition of the Periodic Polar Cell

The azimuthal plane is defined by a single cross-section of the 3D cell with uniform dimensions $dx$ and $dy$, and is directly amenable to 2D ray tracing procedures due to its
inherent periodicity. In the polar plane, multiple planar sections are present with dimensions $dz$ and $s_{m,n}$, where $s_{m,n}$ is the width of each planar section and is analogous to the track lengths calculated during azimuthal ray tracing. As these planar widths have variable size, the individual polar planes will be non-uniform and will not conform to the requirement of periodicity. These polar planes must be modified into a periodic arrangement before the final stage of 3D ray tracing can be pursued.

Periodic arrangement of the polar planar sections can be inferred from the definition of the azimuthal periodic unit as is stated in Section 3.4.1. Re-examination of figure 3.6 shows that this periodic unit is comprised of each of the azimuthal tracks connected in linear sequence. As each of these tracks corresponds to the width of a planar section in the polar plane, polar planes can be connected in an identical linear sequence to yield the *periodic polar cell* as illustrated in figure 3.14.
In the case where there are multiple periodic azimuthal units, such as the cell described in figure 3.8, the periodic polar cell will have multiple instances of uniform external dimension, as shown in figure 3.15.
Figures 3.14 and 3.15 show that the dimensions of a 2D cell in the polar plane will be $dz$ and $dS_m$, and that each cell will contain multiple planar sections. If there is only one periodic azimuthal unit, then all of the planar sections will be contained in the periodic polar cell. If, however, there are multiple periodic azimuthal units, then the polar cell
will contain only a fraction of the total planar sections. It was demonstrated above that multiple periodic azimuthal units are identical in length, so that it is possible to speak of \( \rho_m \) instances of a single azimuthal unit. Similarly, in the polar plane there will be \( \rho_m \) instances of a single periodic polar cell.

### 3.5.2 Correction of the Polar Angles and Polar Track Separation

Correction of the polar angle, \( \theta_m \), and polar track spacing, \( \delta P_m \), can be pursued in a manner that is identical to that described in Section 3.4.1 for the azimuthal plane. First, the periodic polar unit is defined as shown in figure 3.16.

![Figure 3.16: Cells Spanned by the Periodic Polar Unit in the Polar Plane](image)
The number of cells spanned by the periodic polar unit in the z-direction will be

\[ N_{z,m} = \frac{dS_m \cdot |\cos \theta_m|}{\delta P_m}, \quad (3.24) \]

and in the s-direction will be

\[ N_{s,m} = \frac{dz \cdot |\sin \theta_m|}{\delta P_m}, \quad (3.25) \]

These values are again rounded to the next highest integer value to ensure periodicity and direct connection of tracks across cell boundaries, yielding \( \hat{N}_{z,m} \) and \( \hat{N}_{s,m} \). The corrected polar angle, \( \hat{\theta}_m \), is found via

\[ \hat{\theta}_m = \tan^{-1} \left( \frac{\hat{N}_{s,m} \cdot dS_m}{\hat{N}_{z,m} \cdot dz} \right). \quad (3.26) \]

Calculation of the corrected polar spacing, \( \delta \hat{P}_m \), will again utilize the conservation of cell area:

\[ T^\text{tot}_m \cdot \delta \hat{P}_m = dz \cdot dS_m \quad (3.27) \]
where

$$T_m^{\text{tot}} = \sum_{p=1}^{N_{\text{tot}}} s_{p,m}$$  \hspace{1cm} (3.28)

and

$$\hat{N}_{p,m} = \hat{N}_{s,m} + \hat{N}_{z,m}.$$  \hspace{1cm} (3.29)

If the total track length of the polar unit is defined in terms of $\hat{N}_{z,m}$ and $\hat{N}_{s,m}$,

$$T_m^{\text{tot}} = \sqrt{\left(\hat{N}_{z,m} \cdot dz\right)^2 + \left(\hat{N}_{s,m} \cdot dS_m\right)^2},$$  \hspace{1cm} (3.30)

and substituted into Equation (3.27), then the corrected polar spacing is found to be

$$\delta \hat{P}_m = \frac{dz \cdot dS_m}{\sqrt{\left(\hat{N}_{z,m} \cdot dz\right)^2 + \left(\hat{N}_{s,m} \cdot dS_m\right)^2}}.$$  \hspace{1cm} (3.31)

As in the case of the azimuthal periodic unit, there may be multiple periodic units in the polar plane; however, this information is not relevant to 3D ray tracing and need not be evaluated.
3.5.3 Calculation of Global Track Lengths in the Periodic Polar Cell

Tracks are projected across the entire polar cell using procedures that are identical to those described in Section 3.4 for the 2D azimuthal cell, given the following substitutions:

1. All references to the x-dimension are replaced with references to the s-dimension; thus \( dx \) is replaced with \( ds_m, \hat{N}_{s,m} \) is replaced with \( \hat{N}_{s,m} \), and so on.

2. All references to the y-dimension are replaced with references to the z-dimension; thus, \( dy \) is replaced with \( dz, \hat{N}_{y,m} \) is replaced with \( \hat{N}_{z,m} \), and so on.

3. All references to the corrected azimuthal angle, \( \hat{\phi}_m \), are replaced with references to \( \pi/2 - \hat{\theta}_m \), where \( \hat{\theta}_m \) is the corrected polar angle. The difference between these angular definitions reflects a subtle change in geometry due to the fact that \( \hat{\phi}_m \) is defined in relation to the x-axis, while \( \hat{\theta}_m \) is defined in relation to the z-axis, as described previously and shown in figure 3.2.

4. All references to the corrected azimuthal track spacing, \( \delta \hat{A}_m \), are replaced with references to the corrected polar track spacing, \( \delta \hat{P}_m \).
The 2D ray tracing procedures are otherwise implemented for the polar cell exactly as specified previously, yielding a set of global track lengths, $t_p$, and coupling relationships to connect these tracks across boundaries of the periodic polar cell. An example of the result is shown in figure 3.17.

![Figure 3.17: Global Track Lengths for the Periodic Polar Cell](image)

The discretized cell can contain multiple periodic azimuthal units and, as has been demonstrated, such circumstances will yield multiple polar cells with identical dimensions, $dz$ and $dS_n$. It follows that, when multiple polar cells are present, the global track set will be identical for each polar cell. Calculation of global track lengths
and coupling relationships need be performed for one instance of the polar cell only, to be applied uniformly to all remaining polar cells as shown in figure 3.18.
The global track lengths, \( t_p \), must be now subdivided into segments to yield the actual track lengths for each individual polar planar section, as seen in figure 3.19.

If there are multiple polar cells, this process must be repeated for each polar cell, yielding a track set like that shown in figure 3.20.
The procedures for subdividing the global track lengths into planar segments as indicated by figures 3.19 and 3.20 are quite complex and are described in detail in the next section.
3.5.4 Subdivision of Global Track Lengths into Final 3D Track Lengths

Subdivision of the global tracks within the polar cell is conceptually a straightforward process. Sweeping through the global tracks in reverse order, each track is traced in the direction of travel and, at each point where this track crosses a planar boundary the length of the track segment for the plane just crossed is calculated using simple geometric formulae. Such a procedure requires information about the polar angles, $\theta_m$, the dimensions of each planar section, $dz$ and $s_{n,m}$, the number of track entries into the periodic polar cell along each cell edge, $\hat{N}_{z,m}$ and $\hat{N}_{s,m}$, and the total length of each global track, $t_p$. Specific procedures for the tracing and subdivision of an arbitrary track across the polar cell will vary depending on how the track enters and exits the cell; thus, tracks are classified as one of four types:

- Tracks of Type 1 enter the polar cell through the $-s$ cell-edge and exit through the $+z$ cell edge.
- Tracks of Type 2 enter the polar cell through the $-z$ cell-edge and exit through the $+z$ cell edge.
- Tracks of Type 3 enter the polar cell through the $-s$ cell-edge and exit through the $+s$ cell edge.
- Tracks of Type 4 enter the polar cell through the $-z$ cell-edge and exit through the $+s$ cell edge.
The polar cells are themselves classified into one of three cases, depending on the relative values of the parameters $\hat{N}_{z,j}$ and $\hat{N}_{s,j}$. In the first case, an example of which is shown in figure 3.21, $\hat{N}_{z,j}$ is greater than $\hat{N}_{s,j}$.

In this first case, tracks $\left(\hat{N}_{z,m} + \hat{N}_{s,m}\right) \rightarrow \left(\hat{N}_{z,m} + 1\right)$ will be of Type 1, tracks $\hat{N}_{z,m} \rightarrow \left(\hat{N}_{s,m} + 1\right)$ will be of Type 2, and tracks $\hat{N}_{s,m} \rightarrow 1$ will be of Type 4. In the second case for the polar cell $\hat{N}_{z,j}$ is less than $\hat{N}_{s,j}$, as seen in figure 3.22.
In this case, tracks \((\hat{N}_{z,m} + \hat{N}_{s,m}) \rightarrow (\hat{N}_{s,m} + 1)\) will be of Type 1, tracks \(\hat{N}_{s,m} \rightarrow (\hat{N}_{z,m} + 1)\) will be of Type 3, and tracks \(\hat{N}_{z,m} \rightarrow 1\) will be of Type 4. Case 3, where \(\hat{N}_{s,j}\) is equal to \(\hat{N}_{s,j}\), is described in figure 3.23.
In this last case, tracks \( (\hat{N}_{z,m} + \hat{N}_{x,m}) \rightarrow (\hat{N}_{z,m} + 1) \) will be of Type 1 and tracks \( \hat{N}_{z,m} \rightarrow 1 \) will be of Type 4. The procedures for each of these three cases are uniform and consist of a simple loop over each of the global tracks in reverse order, selecting for each track the set of procedures that match that type of track. Procedures for subdividing each of the four types of tracks must be separately defined and are provided in Appendix A.

### 3.5.4 Derivation of Polar Track Coupling Relationships

The last stage in azimuthal ray tracing, as was described in Section 3.4.4, is to determine the coupling relationships for each track in the 2D cell. A similar task must be performed for the polar cell; however, in this case the goal is to derive the full 3D coupling information that is required by the method of characteristics transport solver. For 3D tracks that cross a system composed of uniform 3D cells, each track, \( p \), that lies along a polar plane, \( n \), will couple directly to a track \( p' \) lying within plane \( n' \). Unfortunately, a simple algorithm such as Equations (3.22) and (3.23) cannot be derived in the aperiodic polar plane. Instead the 3D track and plane coupling information is generated during the execution of ray tracing procedures so that whenever a new track length calculation is initiated, coupling information is stored to a temporary array. At the end of all ray tracing procedures this array is reorganized to align with storage of the 3D track length information and all values are stored to the tracking file for later use.
CHAPTER 4:
DERIVATION OF THE TIME-DEPENDENT METHOD OF CHARACTERISTICS

The primary tasks of this work, as enunciated in Chapter 1, are to develop a method of characteristics that retains time-dependence and then to assess its utility in a 3D space-time reactor kinetics methodology. In this chapter the theoretical development of a 3D time-dependent method of characteristics will be pursued. While the time-dependent method of characteristics will be seen to follow naturally from the 3D steady state methods that are described in Chapters 2 and 3, a thorough review of the literature has found no incidents of a previous attempt to produce a time-dependent formulation. Experience gained from other methods for solving the transport equation is not easily transferable to the method of characteristics due to inherent differences in the spatial models for these formulations. Given these circumstances, there will be numerous possible approaches to this complicated problem.

The traditional starting point for any space-time reactor kinetics study lies with the multi-group reactor kinetics equations\(^1\). Equations (1.2) and (1.4) represent the multi-group equations when formulated for diffusion theory and are the primary equations to be solved in state-of-the-art space-time reactor kinetics analysis. A similar set of equations will be derived for a Lagrangian coordinate system in Section 4.1, as is required by the
method of characteristics. Many of the steps in this derivation will directly parallel
development of the steady state method of characteristics. In the steady state case, this
process yields an ordinary differential equation that can be solved exactly via integration
over the one-dimensional spatial variable; however, the time-dependent analogue will be
a partial differential equation and its solution will require the introduction of additional
approximations to simplify the dependence on time.

A review of “time-dependent methods” that are appropriate to this problem will be
presented in Section 4.2. While many such methods are available, not all are applicable
to the method of characteristics and even fewer are appropriate for the initial
development that is the focus of this work. The $\Theta$ method, a simple approximation that is
commonly utilized to reduce various forms of the time-dependent general neutron
transport equation, is selected for further study. Specifically, two $\Theta$ method formulations
will be applied to the method of characteristics: the fully-explicit or “forward
differencing” approximation and the fully-implicit or “backward differencing”
approximation. The derivation of a solution to the time-dependent characteristic equation
via each of these approaches is pursued in Section 4.3.

4.1 The Multigroup Kinetics Equations in Characteristic Form

Derivation of a time-dependent formulation for the method of characteristics begins with
the general neutron transport equation, Equation (1.1). As in the development of the
steady state method of characteristics, a few basic assumptions are made regarding the
physics of a typical space-time kinetics problem to facilitate simplifications to Equation (1.1). First, the external neutron source term, $Q_{ext}$, is neglected. Retention of this term would be required to solve a fixed source problem; however, space-time kinetics problems typically involve a multiplying medium for which the external source is not required. Next, it is assumed that the only significant nuclear reactions of consequence to a multiplying system are fission and neutron scattering, so that the total neutron transfer rate, $\Sigma_f$, can be separated into fission and scattering components. Finally, neutron scattering is assumed to be isotropic via introduction of the transport approximation. Each of these approximations is applied to Equation (1.1) as described in Chapter 2, yielding

$$\frac{1}{\nu} \frac{\partial}{\partial t} \Phi(r, \Omega, E, t) = -\Omega \cdot \nabla \Phi(r, \Omega, E, t) - \Sigma_{\nu}(r, E) \Phi(r, \Omega, E, t) + \int \Sigma_f (r, E' \to E) \phi(r, E', t) dE' + \chi(E) \int v \Sigma_f (r, E') \phi(r, E', t) dE'. \tag{4.1}$$

From this point derivation of the time-dependent and steady state methods of characteristics will diverge. The primary difference between these two formulations regards the sources of fission neutrons, which can be classified into two types. Prompt fission neutrons are created during a fission event and constitute over 99% of all fission neutrons in a nuclear reactor. In the steady state method of characteristics, all fission neutrons are assumed safely to be prompt; however, fission may also create fragments of the original nucleus that are unstable and will decay over time to create delayed neutrons. There are many possible combinations of fission fragments that are created during a fission event, and thus numerous different types of decay occurring simultaneously in a
critical reactor. Only those decay types that generate a delayed neutron are relevant to a reactor kinetics study. The rate at which such processes occur can be expressed as

\[ \lambda_i C_i(r, t), \]  \hspace{1cm} (4.2)

where \( C_i \) is the space- and time-dependent concentration of delayed neutron precursors of a particular isotope, \( i \), and \( \lambda_i \) is the decay constant for that isotope. It is not typically feasible to consider each isotope individually and instead precursors are classified into delay groups according to the isotopic decay times; thus, the total rate of delayed neutron production can be expressed as

\[ \sum_l \lambda_l C_l(r, t), \]  \hspace{1cm} (4.3)

where \( C_l \) is the delayed neutron precursor density for delay group \( l \) and \( \lambda_l \) is the mean decay constant for precursors in that group. As with the multi-group approximation, selection of an appropriate delay group structure will depend on certain properties of the system being modeled. Well-documented one-group and six-group structures are frequently utilized.\(^8\)

To separate the total fission neutron source into prompt and delayed components, the delayed neutron source rate, Equation (4.3), is introduced to Equation (4.1) and
An additional equation is introduced to describe the space- and time-dependent population of delayed neutron precursors,

\[
\frac{\partial}{\partial t} C_i(r, t) = -\lambda_i C_i(r, t) + \beta_i \int \Sigma_f(r, E') \phi(r, E', t) dE' ,
\]

(4.5)

where \( \beta_i \) represents a fraction of the total (prompt and delayed) fission neutrons that appear from the decay of precursors in delay group \( l \). The total delayed neutron fraction, \( \beta \), is computed via summation of \( \beta_l \) over all delay groups. Equations (4.4) and (4.5) are now discretized in energy, angle, and space in like manner to what has been presented in Chapter 2 for the steady state derivation. Application of the multi-group approximation to discretize the energy domain in Equation (4.4) yields

\[
\frac{1}{v^g} \frac{\partial}{\partial t} \Phi^g(r, \Omega, E, t) = -\Omega \cdot \nabla \Phi^g(r, \Omega, E, t) - \Sigma_{ir}^g(r, \Omega, E, t) \Phi^g(r, \Omega, E, t) \\
+ \sum_{g'} \Sigma_{g' \rightarrow g}^g(r) \phi^{g'}(r, t) + \chi_p^g(1 - \beta) \sum_{g'} \Sigma_{g' \rightarrow g}^g(r) \phi^{g'}(r, t) + \chi_d^g \sum_l \lambda_l C_i(r, t)
\]

(4.6)

and, when applied to Equation (4.5),
\[
\frac{\partial}{\partial t} C_i(r,t) = -\lambda_i C_i(r,t) + \beta_i \sum_{g'=1}^G v\Sigma_{g'}^i(r) \phi^{g'}(r,t). \tag{4.7}
\]

Next, the angular domain in Equation (4.6) is discretized via the discrete ordinates approximation,

\[
\frac{1}{r^g} \frac{\partial}{\partial t} \Phi^g_m(r,t) = -\Omega_m \cdot \nabla \Phi^g_m(r,t) - \sum^g_n \Phi^g_n(r,t)
+ \frac{1}{4\pi} \left\{ \sum_{g'=1}^G \Sigma_{g'}^g(r) \phi^{g'}(r,t) + \chi_p^g(1-\beta) \sum_{g'=1}^G v\Sigma_{g'}^g(r) \phi^{g'}(r,t) + \chi_s^g \sum_l \lambda_l C_l(r,t) \right\}. \tag{4.8}
\]

Equation (4.7) contains no angular dependence and need not be modified at this stage.

Finally, the method of characteristics spatial discretization is applied in the two-step manner described previously, yielding

\[
\frac{1}{r^g} \frac{\partial}{\partial s} \Phi^g_{i,m,n,p}(s,t) = -\frac{\partial}{\partial s} \Phi^g_{i,m,n,p}(s,t) - \sum^g_{n} \Phi^g_{i,m,n,p}(s,t)
+ \frac{1}{4\pi} \left\{ \sum_{g'=1}^G \Sigma_{g'}^{i,g}(s) \phi^{g'}(s,t) + \chi_p^g(1-\beta) \sum_{g'=1}^G v\Sigma_{g'}^{i,g}(s) \phi^{g'}(s,t) + \chi_s^g \sum_l \lambda_l \overline{C}_l(t) \right\}. \tag{4.9}
\]

and

\[
\frac{\partial}{\partial t} \overline{C}_i(t) = -\lambda_i \overline{C}_i(t) + \beta_i \sum_{g'=1}^G v\Sigma_{g'}^{i,g}(t) \phi^{g'}(t). \tag{4.10}
\]
In Equations (4.9) and (4.10) the constant source term approximation has been employed, which holds that the material properties, $\Sigma_x^i$, and neutron scalar flux, $\phi^i$, are constant within each cell, $i$, and thus exhibit no dependence on the one-dimensional spatial variable, $s$. By extension, the delayed neutron precursor concentrations, $C_i^d$, are also constant in each cell, as this variable is directly dependent on the cell-constant neutron scalar flux. The reasoning behind using the constant source approximation has been presented in Chapter 2 and applies to the time-dependent method of characteristics in a similar manner.

Equations (4.9) and (4.10) represent the multi-group reactor kinetics equations in characteristic form and are the Lagrangian equivalent to Equations (1.2) and (1.4). The steady state analogue is the characteristic equation, Equation (2.26), which is an ordinary differential equation that is solvable by direct integration of the spatial variable, $s$. Equations (4.9) and (4.10) are, on the other hand, partial differential equations and their solution will require the introduction of additional approximations to simplify time-dependence of the neutron angular flux, $\Phi_{m,m,p}^{i,g}(s,t)$, and delayed neutron precursor concentrations, $C_i^d(s,t)$. The selection of an appropriate temporal approximation is addressed in the next section, after which a solution to the multi-group characteristic equations will be pursued.
4.2 Review of Time-Dependent Methods

The time-dependent characteristic equation, Equation (4.9), can be expressed in simplified operator notation as

$$\frac{\partial}{\partial t} \Phi(t) = H(t)\Phi(t), \quad (4.11)$$

where \( \Phi \) is the primary unknown variable – the neutron angular flux – and \( H \) represents the total transport operator in matrix form. In fact, most solutions to the transport equation can be expressed in the simple form of Equation (4.11), where all dependencies except time have been formally collected into \( H \). The delayed neutron precursor equation, Equation (4.10), can be similarly expressed in terms of a total precursor operator, \( P \), as

$$\frac{\partial}{\partial t} C(t) = P(t)C(t). \quad (4.12)$$

To solve Equations (4.11) and (4.12) numerically the temporal domain must be discretized in like manner to the spatial, energy, and angular domains. A wide variety of approaches to this problem have been developed for the nodal diffusion theory analogues to Equations (4.11) and (4.12) and can be loosely categorized into two types of methods\textsuperscript{82}. Direct methods use finite differencing techniques to discretize the temporal domain into a set of relatively small time steps. During each of these time steps certain
parameters are held constant; thus, the time-dependent transport equation is transformed into a quasi-steady state equation that is amenable to many of the same numerical methods that are used to solve the steady state problem. Indirect methods, on the other hand, solve the time-dependent transport equation via separation of variables. The most well known result of an indirect method is the point kinetics approximation. Point kinetics, of course, assumes that all spatial dependence can be neglected; however, indirect methods can also be utilized to derive a space- and time-dependent solution to the transport equation.

Two of the most common direct methods are the Θ-method, discussed in Section 4.2.1, and the alternating direction implicit (ADI) method, presented in Section 4.2.2. The Θ and ADI methods are simple and efficient yet their accuracy may be insufficient for problems with complicated temporal phenomena. Other advanced direct methods focus on means to increase the size of time steps, thereby reducing computational expense, for more accurate time-dependent transport methods. Two examples of such techniques are the Stiffness Confinement Method (SCM), which is described in Section 4.2.3, and Generalized Runge-Kutta (GRK) techniques, discussed in Section 4.2.4. The most commonly utilized indirect method is quasi-static method and a modern variant – the improved quasi-static method (IQS) – is reviewed briefly in Section 4.2.5.
4.2.1 The $\Theta$ Method

The $\Theta$ method is a simple direct method that discretizes the temporal domain via finite differencing\(^8\) and one of the most common approaches to solution of the time-dependent diffusion equations. The total problem time is divided into a discrete number of time steps and the endpoints of an arbitrary time step, $T$, are defined to be $t^T$ and $t^{T+1}$. The time derivative of the primary unknown is assumed to be the difference between known values at the end of the current and previous time steps divided by the time step size, $\Delta t = t^{T+1} - t^T$. Equation (4.11) then becomes

$$\frac{\partial \Phi}{\partial t} \bigg|_{t^{T+1}} = \frac{\Phi^{T+1} - \Phi^T}{\Delta t} = M^{T+1} \Phi^{T+1} + \left[ H^T - M^T \right] \Phi^T,$$

where the operator matrix $M$ is expressed in relation to the transport operator $H$ at an arbitrary matrix location, $j$, via

$$m_j^T = \theta_j^T h_j^T$$
$$m_j^{T+1} = \theta_j^{T+1} h_j^{T+1}$$

and $\theta_j$ are a series of weighting factors that range between 0 and 1. The operators $H$ and $M$ are typically assumed to be constant during a time step to facilitate the determination of optimal values for $\theta_j$. 

111
Three specific sets of weighting factors are of practical interest. If all weights are set equal to 0, the result will be the explicit Euler approximation,

\[
\frac{\Phi^{T+1} - \Phi^T}{\Delta t} = H^T \Phi^T. \tag{4.15}
\]

Equation (4.15) is also known as the forward differencing approximation. The implicit Euler, or backward differencing, approximation can be similarly obtained by setting all weights equal to 1,

\[
\frac{\Phi^{T+1} - \Phi^T}{\Delta t} = H^{T+1} \Phi^{T+1}. \tag{4.16}
\]

Finally, the Crank-Nicholson scheme is a semi-implicit solution given by

\[
\frac{\Phi^{T+1} - \Phi^T}{\Delta t} = \frac{H^{T+1} \Phi^{T+1} + H^T \Phi^T}{2}, \tag{4.17}
\]

where all weights have been set equal to \(\frac{1}{2}\).

One of the primary concerns with usage of the Θ method regards numerical stability, as discussed in detail in Reference 83. In general, stability can be assessed by expanding Φ into its eigenvectors, \(\Psi_n\), so that
\[ \Phi^T = \sum_n a_n \Psi_n, \]  \hspace{1cm} (4.18)

where

\[ H \Psi_n = \lambda_n \Psi_n \]  \hspace{1cm} (4.19)

and \( \lambda_n \) are the associated eigenvalues. In the case of the forward differencing approximation this expansion is substituted into Equation (4.15), yielding

\[ \Phi^{T+1} = \sum_n a_n (1 + \lambda_n \Delta t) \Psi_n. \]  \hspace{1cm} (4.20)

To be numerically stable, the fundamental eigenvector, \( \Psi_1 \), must grow more rapidly than all of the higher-order modes and

\[ |1 + \lambda_1 \Delta t| > |1 + \lambda_n \Delta t| ; \ n \geq 2. \]  \hspace{1cm} (4.21)

For the requirement of Equation (4.21) to be satisfied, \( |\lambda_n \Delta t| \) must be much less than one. In practice, this implies that time steps must be defined to be quite small, typically on the order of \( 10^{-8} \) sec. A less stringent stability requirement results in the case of backward differencing, where the requirement will be
Equation (4.22) will be satisfied unconditionally for all but the fastest transients, where small time steps would be required anyway. In fact, selection of time step size for the backward differencing approximation is typically governed by concerns of accuracy rather than stability.

### 4.2.2 Alternating Direction Implicit Methods

The Alternating Direction Implicit (ADI) scheme\(^{82,83}\) has a long history of application to the time-dependent diffusion equations and is related to implicit forms of the $\Theta$ method. In ADI, however, a separation of spatial variables is imposed so that

\[
\vec{\partial}_t \Phi(t) = \left[ H_x(t) + H_y(t) + H_z(t) + \Pi(t) \right] \Phi(t),
\]

where the partial operators $H_x, H_y,$ and $H_z$ contain those terms related to streaming in the x-, y-, and z-directions, respectively, and $\Pi$ contains the direction-independent neutron source term operators. The $\Theta$ method operator, $M$, is similarly defined to be

\[
M(t) = M_x(t) + M_y(t) + M_z(t) + \pi(t),
\]

\[
\left| \frac{1}{1 - \lambda_n \Delta t} \right| > \left| \frac{1}{1 - \lambda_n \Delta t} \right| ; \quad n \geq 2. \tag{4.22}
\]
where the elements of each partial matrix are defined via weighting functions, as in Equation (4.14), according to

\[
\begin{align*}
   m_{x,j}^T &= \theta_{x,j}^T h_{x,j}^T \\
   m_{y,j}^T &= \theta_{y,j}^T h_{y,j}^T \\
   m_{z,j}^T &= \theta_{z,j}^T h_{z,j}^T \\
   \pi_j^T &= \theta_j^T \Pi_j^T
\end{align*}
\]  

(4.25)

Solution of the resulting equations is pursued for each time step via a series of partial time step calculations. Weights are selected so that the first partial time step will be fully implicit in the x-directed operator \( (\theta_{x,j} = 1) \) and fully explicit in the y- and z-directed operators \( (\theta_{y,j} = \theta_{z,j} = 0) \). In the second partial time step, the y-direction is fully implicit while the x- and z- directions are fully explicit and, finally, the z-direction is treated implicitly in the final partial time step.

ADI methods are known to yield the same or better accuracy than the \( \Theta \) method when applied to solution of the nodal diffusion equations. ADI is far more efficient, however, and has been combined with adaptive mesh and time step algorithms in advanced kinetics codes to generate extremely efficient diffusion theory calculations. Unfortunately, the partial time-step procedure of ADI methods requires a finite differenced spatial domain and is, therefore, incompatible with the method of characteristics.
4.2.3 The Stiffness Confinement Method

As discussed in the previous sections, most time-dependent methods require the definition of relatively small time step sizes. In general, this problem results from the physical nature of the space-time reactor kinetics problem. The mean lifetime of prompt neutrons is several orders of magnitude shorter than that of delayed neutrons. As a result, computations of Equation (4.11), which models the population of prompt neutrons, and Equation (4.12), which concerns delayed neutrons, will operate on very different time scales. Such equations are numerically “stiff” and many of the concerns regarding numerical stability of the methods reviewed above are a direct result of this mathematical property. Specifically, a satisfactory resolution of the problem of stiffness requires time step sizes that are smaller than would be desired, yielding severely long run times for most problems of interest.

The Stiffness Confinement Method (SCM) was developed by Y. A. Chao and A. Attard as a means to reduce limitations to time step size that result from stiffness in the reactor kinetics equations, thereby yielding a more efficient computation. This method has been adapted for use in such commercial reactor kinetics codes as SPNOVA and PANTHER. SCM decouples Equations (4.11) and (4.12) via the introduction of a pair of “dynamic frequencies” that, when applied to the neutron angular flux, Φ, and delayed neutron precursor concentrations, C, confine all of the stiffness to the prompt fission source. The dynamic frequency for the angular flux is defined to be
\[ w = \frac{\partial}{\partial t} \ln[\Phi] = \frac{1}{\Phi} \frac{\partial \Phi}{\partial t} \]  

(4.26)

and the dynamic frequency for the precursor concentrations is

\[ u = \frac{\partial}{\partial t} \ln[C] = \frac{1}{C} \frac{\partial C}{\partial t}. \]  

(4.27)

With this redefinition of variables, the kinetics equations can be restated as

\[ \frac{\partial}{\partial t} \Phi(t) = \hat{H}(t) \Phi(t), \]  

(4.28)

and

\[ \frac{\partial}{\partial t} C(t) = \hat{P}(t) C(t), \]  

(4.29)

where modified definitions of the transport operator, \( \hat{H} \), and precursor operator, \( \hat{P} \), will be sensitive to the type of spatial and angular discretization methods that are applied. The dynamic frequencies, \( w \) and \( u \), appear within these operators as a modifying term to the material properties and Equations (4.28) and (4.29) are solved iteratively, where values for \( w \) are computed via nested iterations within each time step and values for \( u \) are updated at the end of the time step.
Another approach to the problem of stiffness in the multi-group kinetics equations utilizes Runge-Kutta methods, which have been successfully incorporated into a wide variety of problems involving linear ordinary differential equations. Unlike SCM, which resolves the stiffness problem by decoupling the transport and precursor equations, Runge-Kutta methods introduce a higher-order approximation to the time-dependent terms in Equations (4.11) and (4.12), thereby yielding a highly accurate solution to the space-time kinetics problem that can be more efficient than the simple methods described above. Traditional Runge-Kutta methods have been used rarely in nuclear engineering applications, however, as such methods are typically explicit in nature and thus vulnerable to the same stability requirements as the forward differencing approximation.

An implicit class of Runge-Kutta schemes has been developed more recently, as epitomized by the Generalized Runge-Kutta Method\(^{88,89}\) (GRK), which has been successfully implemented into the SPANDEX 3D nodal diffusion theory code\(^{90}\). GRK begins with the fully implicit time-differencing formula, Equation (4.16). When the term \(H^{T+1}\Phi^{T+1}\) is expanded via linear Taylor series, the result will be

\[
H^{T+1}\Phi^{T+1} = H^T\Phi^T + \Delta t \left[ \frac{\partial}{\partial \Phi} H\Phi \Bigg|_{\Phi^T \Phi^T} \cdot \frac{\partial}{\partial \Phi} \Phi^T \Bigg|_{t} \right].
\] (4.30)
Implicit time differencing is again applied to the time derivative on the right hand side of Equation (4.30) and, when the result is substituted back into Equation (4.16),

$$\frac{\Phi^{T+1} - \Phi^T}{\Delta t} = H^T \Phi^T + \Delta t \left[ \frac{\partial}{\partial \Phi} H \Phi \right]_{\Phi^T} \cdot \frac{\Phi^{T+1} - \Phi^T}{\Delta t}. \quad (4.31)$$

Equation (4.31) can be rearranged so that

$$\left[ I - \Delta t \frac{\partial}{\partial \Phi} H \Phi \right]_{\Phi^T} \Phi^{T+1} = \left[ I - \Delta t \frac{\partial}{\partial \Phi} H \Phi \right]_{\Phi^T} \Phi^T + \Delta t \cdot H^T \Phi^T, \quad (4.32)$$

Equation (4.32) is solved in stages by constructing the solution as

$$\Phi^{T+1} = \Phi^T + \sum_s c_s K^{T+1}_s, \quad (4.33)$$

where $s$ indicates the problem stage, $c_s$ are problem-independent fixed expansion coefficients, and $K^{T+1}_s$ is a vector of unknown expansion coefficients. The vector $K$ is resolved by solving a system of linear equations for each problem stage, $s$, via

$$\left[ I - \gamma \Delta t H \Phi \right]_{\Phi^T} K^{T+1}_s = \Delta t \cdot H^T \Phi^T + \Delta t \left[ \frac{\partial}{\partial \Phi} H \Phi \right]_{\Phi^T} \cdot \sum_{m=1}^{t-1} \gamma_{sm} K^{T+1}_m, \quad (4.34)$$

where $H^T \Phi^T_{\Phi^T}$ is the functional evaluation of $H\Phi$ at the intermediate points.
\[ \Phi^{T*} = \Phi^{T} + \sum_{m}^{i-1} \alpha_{sm} K_{m}^{T+1} \] (4.35)

and \( \gamma, \gamma_{sm}, \) and \( \alpha_{sm} \) are problem-independent fixed constants. Within this procedure, solution is facilitated by a series of embedded estimates for \( \Phi^{T+1} \) and, as a result, the implicit calculation can be completed in fewer computational steps than for more traditional implicit methods.

4.2.5 The Improved Quasi-Static Method

The improved quasi-static method\(^{(2,29)}\) (IQS) is an indirect flux synthesis technique whereby the time- and space-dependent neutron angular flux is expressed as the product of a shape function, \( \Psi(r,\Omega,E,t) \), that is slowly varying in time and a purely time-dependent amplitude function, \( T(t) \). This redefinition of variables is applied to the time-dependent neutron transport equation so that Equation (4.11) becomes

\[
\frac{\partial}{\partial t} \Phi(r,\Omega,E,t) = \frac{\partial}{\partial t} \Psi(r,\Omega,E,t)T(t) = H(t)\Psi(r,\Omega,E,t)T(t). \] (4.36)

This restatement of the time-dependent transport problem is accompanied by a normalization condition,

\[
\int\int\int \frac{\Psi(r,\Omega,E,t)\Psi^*(r,\Omega,E,t)}{\nu(E)} dVdEd\Omega = constant, \] (4.37)
where $\Psi^*$ is the adjoint flux that must be found via solution of the steady state adjoint transport equation during each time increment. After significant manipulation of the resulting equations, the transient problem can be expressed as an amplitude equation,

$$\frac{dT(t)}{dt} = \left( \frac{\rho(t) - \bar{\beta}(t)}{\Lambda(t)} \right) T(t) + \sum \lambda_i C_i(t), \quad (4.38)$$

a delayed neutron precursor equation

$$\frac{dC_i(t)}{dt} = \frac{\beta_i(t)}{\Lambda(t)} C_i(t) - \lambda_i C_i(t), \quad (4.39)$$

and a shape equation that closely resembles the steady state transport equation. Kinetics parameters such as the reactivity, $\rho$, and neutron generation time, $\Lambda$, are defined in terms of the neutron angular flux and the adjoint flux via formulae that can be found in Reference 2. The point kinetics approximation assumes that all space-dependent behavior and, therefore, the shape function calculation can be neglected. Modeling of a transient system via the point kinetics equations, Equations (4.38) and (4.39), is a simple and efficient process and was used frequently when computing resources were less well developed. If a full space-time kinetics computation is required, however, the computationally expensive solution to the shape function equation, as well as a full adjoint calculation during each time step, will be required.
4.3 Application of the Θ Method to the Time-Dependent Method of Characteristics

Of the time-dependent methods reviewed in Section 4.2, the Θ method and IQS are most appropriate as a first choice for incorporation into the time-dependent method of characteristics. Both techniques are well understood and time-dependent finite-differenced discrete ordinates codes have been developed using both formulations\textsuperscript{28,29}. Practical implementation of IQS for the method of characteristics would require a separate adjoint calculation at each time step, however, which would overburden an already expensive computational procedure. The Θ method achieves comparable accuracy, is more computationally efficient, and is the only formulation that will be applied to the method of characteristics in this section.

The two most general Θ method formulations are considered below – the forward differencing approximation, in Section 4.3.1, and the backward differencing approximation, in Section 4.3.2. As mentioned previously, forward differencing yields a simple formula that can be solved directly, while backward differencing will require iterative methods for solution of the resulting equations.
4.3.1 Forward-Differenced Method of Characteristics

As described in Section 4.2.1, an explicit solution to the time-differenced transport equation can be obtained via the Θ method by setting all of the weights, \( \theta \), in Equation (4.13) equal to zero. The result, Equation (4.15), solves for the primary unknown – in this case the neutron angular flux distribution at the end of time step \( T+1 \) – in terms of known quantities for the previous time step, \( T \). Expansion of the transport operator, \( \mathbf{H}^T \), in Equation (4.15) into characteristics form yields

\[
\frac{\Phi^{i,g}_{m,n,p}(s)^{T+1} - \Phi^{i,g}_{m,n,p}(s)^T}{\nu^g \Delta t} = -\frac{d}{ds} \Phi^{i,g}_{m,n,p}(s)^T - \left[ \sum_{ir} \Phi^{i,g}_{m,n,p}(s)^T - \sum_{ir} \Phi^{i,g}_{m,n,p}(s)^T \right] \\
+ \frac{1}{4\pi} \left[ \sum_{g'=1}^{G} \overline{\Phi}^{i,g'}_{m,\nu} - \overline{\Phi}^{i,g}_{m,\nu} \right]^T + \chi^E_p \left( 1 - \beta \right) \sum_{g'=1}^{G} \overline{\nu} \sum_{f} \phi^{i,g'}_{m,\nu} \left( 1 - \beta \right) \sum_{g'=1}^{G} \chi^E_d \sum_{l} \lambda_i \overline{C} \right]^T 
\]

or, when rearranged to solve for \( \Phi^{i,g}_{m,n,p}(s)^{T+1} \),

\[
\Phi^{i,g}_{m,n,p}(s)^{T+1} = \Phi^{i,g}_{m,n,p}(s)^T - \nu^g \Delta t \left[ \sum_{ir} \Phi^{i,g}_{m,n,p}(s)^T + \frac{d}{ds} \Phi^{i,g}_{m,n,p}(s)^T - \overline{O}^{i,g} \right], \quad (4.41)
\]

where the total neutron source term, \( \overline{O}^{i,g} \), is

\[
\overline{O}^{i,g} = \overline{O}_{\text{scatter}}^{i,g} + \overline{O}_{\text{prompt}}^{i,g} + \overline{O}_{\text{delay}}^{i,g} \quad (4.42)
\]
with components given by

\[
\overline{Q}_{\text{scatter}}^{i,g} = \frac{1}{4\pi} \sum_{g'=4}^{G} \sum_{s} \overline{\phi}_{i,g'=g}^{s} \phi_{i,g'}^{T}
\]  
(4.43)

\[
\overline{Q}_{\text{prompt}}^{i,g} = \frac{1}{4\pi} \chi_{p}^{g} (1 - \beta) \sum_{g'=4}^{G} \nu \Sigma_{f}^{g',g} \overline{\phi}_{i,g'}^{T}
\]  
(4.44)

and

\[
\overline{Q}_{\text{delay}}^{i,g} = \frac{1}{4\pi} \chi_{d}^{g} \sum_{g'} \lambda_{i} \overline{C}_{i}^{g'}^{T}
\]  
(4.45)

The forward time-differencing approximation can be applied in a similar manner to the delayed neutron precursor equation, Equation (4.12), so that

\[
\frac{\overline{C}_{i}^{T+1} - \overline{C}_{i}^{T}}{\Delta t} = -\lambda_{i} \overline{C}_{i}^{T} + \beta_{i} \sum_{g'=4}^{G} \nu \Sigma_{f}^{g',g} \overline{\phi}_{i,g'}^{T}
\]  
(4.46)

or, when rearranged to solve for \( \overline{C}_{i}^{T+1} \),

\[
\overline{C}_{i}^{T+1} = (1 - \lambda_{i} \Delta t) \overline{C}_{i}^{T} + \beta_{i} \Delta t \sum_{g'=4}^{G} \nu \Sigma_{f}^{g',g} \overline{\phi}_{i,g'}^{T}
\]  
(4.47)
In Equations (4.41) and (4.47), the parameters $\Phi_{m,n,p}^{i,g}(s)$ and $C_i^p(s)$ are solved at each time step entirely in terms of known parameters that are calculated during the previous time step, as is consistent with the explicit definition of a function; however, solution of Equation (4.41) is complicated by the presence of the derivative term, $\frac{d}{ds}\Phi_{m,n,p}^{i,g}(s)|^T$, as the continuous spatial dependence of this variable is unknown. This problem can be addressed via the introduction of an approximation to account for spatial variation.

One simple approximation for the neutron angular flux distribution in Equation (4.41) is to assume that this variable is constant along a particular track,

$$\Phi_{m,n,p}^{i,g}(s')|^T \Rightarrow \Phi_{m,n,p}^{i,g}|^T,$$  \hspace{1cm} (4.48)

where $\Phi_{m,n,p}^{i,g}$ is equivalent to the track-averaged angular flux and can be calculated via Equation (2.36). The derivative of any constant is zero, so Equation (4.41) becomes

$$\Phi_{m,n,p}^{i,g}(s)|^{T+1} = \Phi_{m,n,p}^{i,g}(s)|^T - v^g \Delta t \left[ \sum_{tr} \Phi_{m,n,p}^{i,g}(s)|^T - Q^{i,g} \right].$$  \hspace{1cm} (4.49)

When applied to the location where track $m$ in plane $n$ exits cell $i$, which is the computational point of interest, Equation (4.49) reduces to

$$\Phi_{m,n,p}^{i,g}|_{out}^{T+1} = \Phi_{m,n,p}^{i,g}|_{out}^T - v^g \Delta t \left[ \sum_{tr} \Phi_{m,n,p}^{i,g}|_{out}^T - Q^{i,g} \right].$$  \hspace{1cm} (4.50)
The assumption that neutron angular flux will be constant along a track segment is over-restrictive and would yield a trivial solution if extended to the full transport problem; i.e. an unvarying angular flux distribution across the entire problem space. An improved approximation that allows for spatial variation while maintaining a relatively simple treatment for the derivative term is to assume that \( \Phi_{m,n,p}^{i,g}(s) \) is a linear function of position along the track,

\[
\Phi_{m,n,p}^{i,g}(s)^T = \Phi_{m,n,p}^{i,g} \bigg|_{in}^T + \left[ \frac{\Phi_{m,n,p}^{i,g} \bigg|_{out}^T - \Phi_{m,n,p}^{i,g} \bigg|_{in}^T}{\Delta s_{m,n,p}} \right] (s - s_{in}). \tag{4.51}
\]

In this case, the derivative will reduce to

\[
\frac{d}{ds} \Phi_{m,n,p}^{i,g}(s)^T = \frac{\Phi_{m,n,p}^{i,g} \bigg|_{out}^T - \Phi_{m,n,p}^{i,g} \bigg|_{in}^T}{\Delta s_{m,n,p}}. \tag{4.52}
\]

Inserting Equation (4.52) into Equation (4.41) and evaluating the result at the point where the track segment exits the 3D cell yields

\[
\Phi_{m,n,p}^{i,g} \bigg|_{out}^{T+1} = \Phi_{m,n,p}^{i,g} \bigg|_{out}^T - v_{i,g} \Delta t \left[ \sum_{n,p} \Phi_{m,n,p}^{i,g} \bigg|_{out}^T \frac{\Phi_{m,n,p}^{i,g} \bigg|_{out}^T - \Phi_{m,n,p}^{i,g} \bigg|_{in}^T}{\Delta s_{m,n,p}} - \overline{Q}_{i,g} \right]. \tag{4.53}
\]

A third possible approximation for the neutron angular flux in Equation (4.41) can be inferred from the solution to the steady state characteristic equation, Equation (2.29),
which expresses the neutron angular flux as a function of exponentials. An analogous
exponential function for the time-dependent neutron angular flux would be

\[ \Phi_{m,n,p}^{i,g}(s) = Ae^{Bs}. \]  

Given that the known boundary values for each track are

\[ \Phi(s_{m,n,p}^{\text{in}}) = \Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}, \]
\[ \Phi(s_{m,n,p}^{\text{out}}) = \Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}, \]

the constants \( A \) and \( B \) from Equation (4.54) can be determined, yielding the exponential
form for the previous time-step angular flux,

\[ \Phi_{m,n,p}^{i,g}(s) = \Phi_{m,n,p}^{i,g} \bigg|_{\text{in}} \left( \frac{\Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}^{\text{T}}}{\Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{\text{T}} + \frac{s - s_{\text{in}}}{\Delta s_{m,n,p}}} \right)^{\text{T}}. \]  

The spatial derivative of Equation (4.56) will be

\[ \frac{d}{ds} \Phi_{m,n,p}^{i,g}(s) = \left( \frac{\Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{\text{T}}}{\Delta s_{m,n,p}} \right) \left( \frac{\Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}^{\text{T}}}{\Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{\text{T}}} \right)^{\text{T}}. \]  

127
When Equation (4.57) is inserted into Equation (4.41) and evaluated at the cell exit location, the result is

\[
\Phi_{m,n,p}^{i,g} \left[ \Phi_{m,n,p}^{i,g} \right]_{\text{out}}^{t+1} = \Phi_{m,n,p}^{i,g} \left[ \Phi_{m,n,p}^{i,g} \right]_{\text{out}}^t - v^g \Delta t \left[ \sum_{i,j}^{i,g} \frac{1}{\Delta x_{m,n,p}} \Phi_{m,n,p}^{i,g} \left[ \Phi_{m,n,p}^{i,g} \right]_{\text{out}}^t - \Phi_{m,n,p}^{i,g} \right].
\] (4.58)

Prior to implementation and testing, it is unclear which of these approximations is optimal for use in a time-dependent method of characteristics. If sensitivity to the order of the functional approximation is found to be significant, a higher-order approximation could be required; for example, an \( n \)th order polynomial could be introduced. A more precise estimate of this term could also be derived via a Taylor polynomial expansion or recursive techniques. For the remainder of this work, the scope will be limited to the simple approximations described above.

### 4.3.2 Backward-Differenced Method of Characteristics

The fully-implicit time differencing approximation, Equation (4.16), is derived from the \( \Theta \) method by setting all weights equal to 1. The result is a recursive relationship whereby \( \Phi_{m,n,p}^{i,g} \) is defined for each time step in relation to unknown parameters from the same time step. When expanded into characteristic form, Equation (4.16) becomes
while the backward-differenced precursor equation is

\[
\frac{\Phi_{m,n,p}^{i,g}(s) - \Phi_{m,n,p}^{i,g}(s)}{v^m \Delta t} = -\frac{d}{ds} \Phi_{m,n,p}^{i,g}(s) + \sum_{i \rightarrow g} \frac{\gamma_p}{\Phi} + \chi_p (1 - \beta) \sum_{g' = 1}^G \nu \Sigma_f \frac{\gamma'}{\Phi} + \chi_d \sum_{f} \lambda_f C_i^{T+1},
\]

(4.59)

Equation (4.60) can be rearranged to solve for \( C_i^{T+1} \), yielding

\[
C_i^{T+1} = \frac{C_i^T}{\Delta t} + \lambda_i C_i^T + \beta_i G \sum_{g' = 1}^G \nu \Sigma_f \frac{\gamma'}{\Phi} + \chi_d \sum_{f} \lambda_f C_i^{T+1}.
\]

(4.60)

Equation (4.60) can be rearranged to solve for \( C_i^{T+1} \), yielding

\[
C_i^{T+1} = \gamma_i C_i^T + \gamma_i \beta_i \Delta t G \sum_{g' = 1}^G \nu \Sigma_f \frac{\gamma'}{\Phi} + \chi_d \sum_{f} \lambda_f C_i^{T+1}.
\]

(4.61)

where \( \gamma_i = (1 + \lambda_i \Delta t)^{-1} \). Equation (4.61) is substituted for the last term in Equation (4.59) and, upon rearranging and redefining some terms, the result is

\[
\frac{d}{ds} \Phi_{m,n,p}^{i,g}(s) + \sum_{i \rightarrow g} \frac{\gamma}{\Phi} + \sum_{i \rightarrow g} \frac{\gamma}{\Phi} = \Omega^{i,g},
\]

(4.62)

where the effective transport cross section is defined to be
\[ \bar{\Sigma}_{tr}^{i,g} = \left( \frac{1}{v^g \Delta t} + \bar{\Sigma}_{tr}^{i,g} \right) \]  

(4.63)

and the neutron source terms are given by

\[ \bar{Q}^{i,g} = \bar{Q}^{i,g} + \Phi_{m,s,p}^{i,g}(s) \left[ \frac{1}{v^g \Delta t} \right]^T \]  

(4.64)

\[ \bar{Q}^{i,g} = \bar{Q}_{scatter}^{i,g} + \bar{Q}_{prompt}^{i,g} + \bar{Q}_{delay}^{i,g} \]  

(4.65)

\[ \bar{Q}_{scatter}^{i,g} = \frac{1}{4\pi} \sum_{g=1}^{G} \sum_{i} \bar{\Sigma}_{i}^{g \rightarrow g} \phi^{i,g}_{T+1} \]  

(4.66)

\[ \bar{Q}_{prompt}^{i,g} = \frac{1}{4\pi} \chi_{p}^{g} (1 - \beta) \sum_{g=1}^{G} \sum_{f} \bar{\Sigma}_{f}^{g \rightarrow g} \phi^{i,g}_{T+1} \]  

(4.67)

and

\[ \bar{Q}_{delay}^{i,g} = \frac{1}{4\pi} \chi_{d}^{g} \sum_{i} \lambda_{i} \left\{ \gamma_{i} \bar{C}_{i}^{T} + \gamma_{i} \beta_{i} \Delta t \sum_{g=1}^{G} \bar{\Sigma}_{g}^{p \rightarrow g} \phi^{i,g}_{T+1} \right\} \]  

(4.68)

Whereas the result of explicit differencing was an equation that is quite unlike the steady state characteristic equation, Equation (4.62) is identical in form to the steady state
analogue and must be solved via integration over the spatial variable, \( s \). The general solution to Equation (4.62) will be

\[
\Phi_{m,n,p}^{i,g} (s) = \Phi_{m,n,p}^{i,g} \left[ \int_{s_{n}}^{s_{n+1}} e^{-\xi_{\sigma} \Delta s} ds \right] + \int_{s_{n}}^{s_{n+1}} \frac{1}{v_{s} \Delta t} \Phi_{m,n,p}^{i,g} (s') e^{-\xi_{\sigma} \Delta s'} ds'.
\] (4.69)

Appearance of the neutron angular flux at the previous time step, \( \Phi_{m,n,p}^{i,g} (s) \), within a spatial derivative in the forward differencing formulation necessitated the introduction of a simplifying approximation. In the case of backward differencing, this parameter appears within the source term integral in Equation (4.69) and a similar functional approximation must be introduced. Initially, the same three approximations will be investigated. When the constant angular flux approximation, Equation (4.48), is applied, Equation (4.69) reduces to

\[
\Phi_{m,n,p}^{i,g} \left[ \int_{s_{n}}^{s_{n+1}} e^{-\xi_{\sigma} \Delta s} ds \right] + \frac{1}{v_{s} \Delta t} \Phi_{m,n,p}^{i,g} (s') e^{-\xi_{\sigma} \Delta s'} ds'.
\] (4.70)

The linear angular flux approximation, Equation (4.51), yields
Finally, the solution to Equation (4.69) via the exponential angular flux approximation, Equation (4.56), will be

\[
\Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}^{T+1} = \frac{1}{\Sigma_{ir}} \Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{T+1} + \left(1 - e^{-\sum_{ir}^{i,g} \Delta s_{m,n,p}} \right) Q_{1}^{i,g} + \left(1 - e^{-\sum_{ir}^{i,g} \Delta s_{m,n,p}} \right) \frac{1}{A_{0}} \ln \left( \frac{\Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}^{T}}{\Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{T}} \right) \Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{T}.
\]  

(4.71)

Finally, the solution to Equation (4.69) via the exponential angular flux approximation,

\[
\Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}^{T+1} = \Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{T+1} e^{-\sum_{ir}^{i,g} \Delta s_{m,n,p}} + \left(1 - e^{-\sum_{ir}^{i,g} \Delta s_{m,n,p}} \right) \frac{1}{A_{0}} \ln \left( \frac{\Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}^{T}}{\Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{T}} \right) \Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{T}.
\]  

(4.72)

where \( A_{0} = \sum_{ir}^{i,g} \left( \frac{1}{\Delta s_{m,n,p}} \right) \ln \left( \frac{\Phi_{m,n,p}^{i,g} \bigg|_{\text{out}}^{T}}{\Phi_{m,n,p}^{i,g} \bigg|_{\text{in}}^{T}} \right) \).

Sensitivity of the backward-differenced characteristic equation to the order of approximation introduced for the neutron angular flux is again unknown without empirical investigation. A comparison of forward- and backward-differenced formulations reveals one significant difference between these two approaches to the method of characteristics. An implicit solution to the method of characteristics will require iterative computation to solve for the neutron angular flux distribution, as will be...
described in detail in Chapter 5. As a result, a full pseudo-steady state calculation is performed during each time step. In forward differencing, the same task is completed via a single sweep across the problem space. The explicit formulation thus exhibits a lesser degree of sensitivity to spatial effects and is derived directly from the initial steady state flux shape. In the backward-differenced equation this flux shape is recomputed at each time step. This observation suggests that the implicit solution may be more sensitive to the accuracy of a spatial approximation than will be the explicit solution. Further analysis of this issue will be pursued via test problems that are presented in Chapter 6.

### 4.3.3 Alternative Formulations for the Delayed Neutron Precursor Equation

In the explicit and implicit method of characteristics formulations derived above, the delayed neutron precursor equation has been simplified using the same time dependent methods that are applied to the transport equation. An alternative approach to solution of the precursor equation is the time-integrated approximation \(^{83}\), whereby Equation (4.10) is integrated over the range of the time step and

\[
\begin{align*}
\overline{C}_i^j(s)_{T+1} &= \overline{C}_i^j(s)_{T} e^{-\lambda_i \Delta t} \\
+ \beta \sum_g^{\Sigma_f} &\left[ \left( \frac{1}{\lambda_i} \right) \left( 1 - e^{-\lambda_i \Delta t} \right) \overline{\phi}^{i,g'}(s)_{T+1} + \left( \frac{1}{\lambda_i} \right) \left( 1 - e^{-\lambda_i \Delta t} \right) \overline{\phi}^{i,g'}(s)_{T} \right].
\end{align*}
\]

In Equation (4.73) the scalar flux is assumed to vary linearly with time according to
The time-integrated approximation is compatible with most available time-dependent methods for simplifying the transport equation and, in fact, is typically preferred to the more consistent derivations presented above. In application, usage of the time-integrated approximation reduces the effects of mathematical stiffness while having a negligible effect on the problem solution. Nonetheless, this method will not be considered in the present work to avoid over-complication of these initial developments.
CHAPTER 5:
NUMERICAL COMPUTATION OF THE 3D
STEADY STATE AND TIME-DEPENDENT
METHODS OF CHARACTERISTICS

The mathematical derivations presented in the previous chapters provide the fundamental equations and theoretical framework necessary to perform a time-dependent method of characteristics computation in 3D Cartesian geometry. To assess the feasibility of such a calculation as part of a 3D space-time reactor kinetics methodology, these methods have been implemented into the MOCK-3D reactor kinetics code, an overview of which is presented in Section 5.1. This code consists of three stand-alone components: RayMonde, which performs the 3D ray tracing procedures; MOCK-3D, which governs the 3D steady state calculation; and MOCK-3DK, in which both the forward- and backward-differenced formulations of the time-dependent method of characteristics have been implemented. To perform a full time-dependent calculation, each of these modules is executed in sequence; i.e., RayMonde is first invoked to generate the 3D ray tracing information required by all subsequent calculations; MOCK-3D is called to generate the converged steady state angular and neutron scalar flux distributions and other data required to initialize the time-dependent calculation; and finally, MOCK-3D is executed to compute results for the transient problem of interest.
Of course, the mathematical equations that have been previously derived are insufficient to complete this task. Numerical methods must also be developed and analyzed as part of the development of each of the MOCK-3D code components. The primary topic of this chapter is to pursue a detailed discussion of these methods. For the 3D ray tracing module RayMonde this presentation will be brief, as Chapter 3 includes all procedural detail that is required for code development. A brief summary of these procedures will be presented in Section 5.2. Section 5.3 regards solution of the steady state method of characteristics and will include additional discussion of the experience gained from previous method of characteristics implementations. Finally Sections 5.4 and 5.5 will present numerical methods for solution of the fully-explicit and fully-implicit methods of characteristics, respectively. Each section will conclude with a discussion of the numerical stability and accuracy of these time-dependent formulations. Analyses of an empirical nature are reserved for Chapter 6.

5.1 Description of the MOCK-3D Code Package

The MOCK-3D computational reactor kinetics package has been developed as a platform for analysis of the method of characteristics formulations that are derived in the previous chapters. To minimize the burden of what is expected to be an extremely intensive computation, this package is developed as three separate components: RayMonde, which executes the 3D ray tracing procedures described in Chapter 3; MOCK-3D, which performs 3D steady state method of characteristics calculations as outlined in Chapter 2; and MOCK-3DK, which solves the multi-group kinetics equations in characteristic form.
that are derived in Chapter 4. These codes are written in the Fortran 90 programming language and have been successfully compiled and executed (for small problems) on a desktop computer using the Windows XP operating system. All of the formal calculations that will be presented in Chapter 6 have been executed on the LION-XO cluster, which is operated by the High Performance Computing Group within Information Technology Services at The Pennsylvania State University. This multi-processor PC cluster uses the linux operating system and is designed for parallel or serial computations that have large memory requirements.

Each of the three component modules in the MOCK-3D package is executed separately, yet information is passed between modules via external files. The first stage in any calculation is to execute RayMonde, which performs ray tracing for 3D Cartesian geometry and generates a tracking file. This tracking file contains information that is required as input for both MOCK-3D and MOCK-3DK. Next MOCK-3D is executed to compute the converged steady state neutron scalar flux distribution, the effective multiplication factor, $k_{\text{eff}}$, and other parameters that are relevant to the steady state method of characteristics solution. If a time-dependent solution is sought MOCK-3D will also store the converged steady state neutron angular flux distribution to an external file, to be input by MOCK-3DK for the purposes of initializing the time-dependent calculation. Finally, MOCK-3DK is executed to solve the multi-group kinetics equations in characteristic form. A detailed description of the numerical methods that are required to perform each of these computations is presented in the remainder of this chapter. The full source code for the MOCK-3D reactor kinetics package is provided in Appendix B.
5.2 Implementation of 3D Ray Tracing Methods

Detailed procedures for performing ray tracing in 3D Cartesian geometry are described in Chapter 3 and implemented in RayMonde exactly as described therein. At the conclusion of the RayMonde computation, information that is required for steady state and/or time-dependent method of characteristics calculations is written to an external file that is commonly referred to as a “tracking file.” This file contains the following parameters:

- Modified azimuthal angles, polar angles, and direction weights.
- Modified azimuthal and polar track separations.
- The number of periodic azimuthal units.
- The number of polar planes in each 3D cell, the number of tracks in each polar plane, and the number of plane and track penetrations along each 3D cell face.
- 3D track lengths.
- 3D plane and track coupling information.

As discussed previously, in lattice physics applications of the method of characteristics this tracking file can be quite large due in part to the large number of tracks that are required to describe an entire system model. The ray tracing procedures in RayMonde are defined so that track lengths need be stored for only a single representative cell and, as a result, the storage requirements of the tracking file are significantly reduced.
To ensure that RayMonde is properly coded and that the 3D ray tracing procedures are correctly defined, it is necessary to verify that the contents of the RayMonde tracking file are “correct” for a given problem specification. There are, however, no ray tracing benchmarks or standard test problems available in the literature for this purpose. In fact, the task of ray tracing is typically considered to be a trivial problem that is seldom discussed in detail except in code manuals that are often proprietary. To test the RayMonde module it is necessary to perform a series of hand calculations for a test cell and compare these results to the values that are stored in the tracking file. These calculations have been performed using a combination of drafting and measurement techniques, whereby the test cell is literally drawn on paper and the length of each track segment is measured by hand. Coupling relationships can be similarly verified by direct observation. In this manner, the RayMonde module has been verified using a 3D cubic test cell with cell dimensions of 1.0 cm, track separation of approximately 0.2 cm, and the S₄ level symmetric quadrature set.

5.3 Implementation of the 3D Steady State Method of Characteristics

The fundamental equations that govern the steady state method of characteristics have been derived in Chapter 2. At the heart of this calculation lies the solution to the steady state characteristic equation, Equation (2.33), which is solved for each computational point of interest to generate the neutron angular flux distribution for a complete system model. This information can be used subsequently to generate other data of interest, such
as the neutron scalar flux distribution or various core- and region-averaged quantities. To solve Equation (2.33), however, a single sweep over the problem space is insufficient. The steady state method of characteristics is an eigenvalue problem that requires the introduction of iterative numerical methods. The method most commonly utilized for this purpose is the power iteration method, which is reviewed in Section 5.3.1. The implementation of power iteration into MOCK-3D is discussed in Section 5.3.2

While the development of a time-dependent method of characteristics is the primary topic of this research and the steady state method of characteristics is reasonably well understood, an accurate and efficient steady state computation must be established before any time-dependent implementation can be pursued. Moreover, the MOCK-3D steady state module will be of use in assessing the results for a secondary goal that has been pursued in this work – development of the capability to explicitly model 3D Cartesian geometry. In Chapter 6, where the computational methods described herein will be applied to a series of test problems, this 3D modeling capability will be assessed primarily through analysis of steady state problems. Thereafter, assessment of the time-dependent method of characteristics can be focused on transient performance of the MOCK-3DK time-dependent method of characteristics module.
5.3.1 The Steady State Power Iteration Method

The steady state method of characteristics is derived from the general time-dependent neutron transport equation by neglecting time-dependence and modifying the fission neutron source term with an eigenvalue – the effective multiplication factor, $k_{\text{eff}}$. Eigenvalue problems are one of the most commonly encountered mathematical forms in science and engineering and, while various methods have been developed to solve such problems, the most common and well understood is the power iteration method. A variant of this numerical scheme is used in most of the commercial lattice physics codes reviewed in Chapter 2; thus, the performance of power iteration to solve the characteristic equation is well understood.

Power iteration$^{91,92}$ begins by reformulating the transport equation in terms of the fission neutron distribution, $\vec{F}$, which is a one-dimensional vector of length $N$ that contains the total number of fission neutrons born in each cell, $i$,

$$F^i = V^i \sum_g \Sigma_f^{i,g} \phi^{i,g}, \quad (5.1)$$

where $V^i$ is the cell volume. The eigenvalue problem can be stated as

$$A\vec{F} = \lambda \vec{F}, \quad (5.2)$$
where $\lambda$ and $\vec{F}$ are the eigenvalue and eigenvector of Equation (5.2) and $A$ represents a reformulated transport operator that is derived in Reference 92. There exist a total of $N$ eigenvalues, corresponding to the number of cells in the system and, thus, the length of the vector $\vec{F}$, where it is assumed that $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_N \geq 0$. The largest amplitude eigenvalue, $\lambda_1$, is known as the fundamental eigenvalue and is in practice identified with the effective multiplication factor, $k_{\text{eff}}$, so that

$$A\vec{F} = k_{\text{eff}} \vec{F}.$$  \hfill (5.3)

The fission distribution is solved iteratively according to

$$\vec{F}^{(j)} = \frac{1}{k_{\text{eff}}^{(j-1)}} A\vec{F}^{(j-1)},$$  \hfill (5.4)

where $\vec{F}^{(j)}$ is the estimate for the eigenvector during the $j^{\text{th}}$ iteration. The eigenvalue, $k_{\text{eff}}$, is updated at the end of each iteration via the formula

$$k_{\text{eff}}^{(j)} = k_{\text{eff}}^{(j-1)} \frac{\|\vec{F}^{(j)}\|}{\|\vec{F}^{(j-1)}\|} = k_{\text{eff}}^{(j-1)} \frac{\sum_i F_i^{(j)}}{\sum_i F_i^{(j-1)}},$$  \hfill (5.5)
where $\|F\|$ is the $L_1$-norm of $F$. Iterative computation proceeds until the eigenvalue and eigenvector are found to converge to within specified tolerance criteria. It can be further shown that

$$F^{(j)} = \left( \prod_{k=1}^{j-1} \frac{1}{k_{\text{eff}}} \right) A^j F^{(0)},$$  \hspace{1cm} (5.6)

which implies that the fission distribution for outer iteration $j$ can be computed by applying the operator $A$ a total of $j$ times to the initial guess, $F^{(0)}$. It is from this result that power iteration draws its name. In practice, power iteration is performed via a two-level nested iteration structure. The computational flow through the inner and outer iterations of a typical power iteration scheme is illustrated in figure 5.1.
At the beginning of each outer iteration the fission source distribution is fixed and control passes to the inner iteration. For this reason, power iteration is also commonly referred to as “fixed source” iteration. During the inner iteration, the neutron angular flux, neutron scalar flux, and scattered neutron source distributions are computed in sequence. This process is repeated until all angular and/or neutron scalar flux values are found to converge to within specified tolerance criteria. Subsequently, control is passed back to
the outer iteration where the eigenvalue and fission source terms are updated for use in
the next iteration cycle. The steady state computation is complete when a converged
value for $k_{\text{eff}}$ has been determined. A more thorough discussion of the procedures
required to perform power iteration in MOCK-3D is provided in the sections that follow.

5.3.2 Implementation of Steady State Power Iteration in MOCK-3D

At the beginning of a MOCK-3D computation, all of the significant parameters are
initialized to avoid numerical instability during the first few iteration cycles. The precise
value that is assigned to each of these parameters is somewhat arbitrary; however, if these
initial guesses are chosen properly, speed of convergence can be improved. The MOCK-
3D computation then follows the flow illustrated in figure 5.1, passing to the first outer
iteration in the steady state power iteration scheme.

During each inner iteration in MOCK-3D, the fission source is held constant while the
neutron angular flux, neutron scalar flux, and scattering source distributions are solved
separately for each energy group, beginning with the highest energy group and
proceeding to the lowest energy group. Upscattering, i.e., scattering of neutrons from
lower energy groups to higher energy groups, is forbidden in this initial development.
The neutron angular flux distribution is found by solving a modified form of Equation
(2.34) for each location where a characteristic exits a cell,
\[
\Phi_{m,n,p}^{i,g} \left( s_{m,n,p} \right) = \Phi_{m,n,p,0}^{i,g} \cdot P_{m,n,p}^{i,g} + \frac{Q_{i,g}^{j}}{\Sigma_{i,g}} \left( 1 - P_{m,n,p}^{i,g} \right),
\]

(5.7)

where \( P_{m,n,p}^{i,g} \) represents the probability that a neutron does not experience a collision event over the entire length of track \( s_{m,n,p} \) and is equal to \( e^{-\Sigma_{i,g} \Delta s_{m,n,p}} \). The calculation of non-collision probabilities during the inner iteration can add significantly to runtime due to the nature of the computation of an exponential in the Fortran programming language. Instead these terms are tabulated during problem initialization and found via a table look-up procedure when required by the inner iteration.

Within each energy group iteration, Equation (5.7) must be solved for each direction, \( m \), in each cell, \( i \), and for all tracks, \( p \), on all polar planes, \( n \), in that cell. The angular domain can be accounted for via a simple nested looping structure that performs the required set of calculations for each direction in the quadrature set, beginning with octant one and proceeding through each of the seven remaining direction octants. Coverage of the spatial domain is more complicated. For spatial finite differencing methods, a simple cell-sweep would be utilized to cover the spatial domain, whereby the angular flux distribution is calculated for an entire cell one cell at a time. While such a technique simplifies the coding of the neutron angular flux calculation, propagation methods such as the method of characteristics allow for a more logical track-sweep technique to be implemented. In a track-sweep, the path of a single characteristic is traced across the entire system model before proceeding to the next characteristic, as seen in figure 5.2.
This track-sweep is repeated for each characteristic in the system model. In the method of characteristics a track-sweep is advantageous for two reasons: it enables a more efficient storage of boundary condition information and is consistent with techniques for parallelization of the method of characteristics. While parallel computation procedures are not included at this time, such an advance is anticipated as a future development to reduce computation time. In that event, the most effective parallelization would decompose the problem into clusters of parallel characteristics, for which a track sweep may be required.

The first step in the sweep for an individual characteristic is to determine the incoming boundary condition at the external system boundary where this characteristic originates.
The 3D ray tracing methods in Chapter 3 were developed so as to yield a set of naturally occurring boundary conditions at internal and external cell boundaries. Tracks are required to connect directly across internal cell boundaries, yielding the continuity boundary condition

\[ \Phi_{m,n,p}^{i,g} \bigg|_{\text{incoming}} = \Phi_{m',n',p'}^{i',g} \bigg|_{\text{outgoing}}, \]  

(5.8)

where \( \Phi_{m,n,p}^{i,g} \bigg|_{\text{incoming}} \) is the desired boundary value – the incoming group neutron angular flux for track \( p \) on plane \( n \) in cell \( i \) – and \( \Phi_{m',n',p'}^{i',g} \bigg|_{\text{outgoing}} \) is the outgoing neutron angular flux from track \( p' \) on plane \( n' \) in cell \( i' \). The coupling of track \( p \) on plane \( n \) to track \( p' \) on plane \( n' \) obeys the relationships derived in Chapter 3.

At external cell boundaries, each track will again connect directly to the endpoint of another track; however, in this case the boundary condition is defined by

\[ \Phi_{m,n,p}^{i,g} \bigg|_{\text{incoming}} = \alpha_{face} \Phi_{m',n',p'}^{i',g} \bigg|_{\text{outgoing}}, \]  

(5.9)

where a track along direction \( m \) connects to a track along direction \( m' \). Tracks must obey optical reflection at external boundaries, which requires that the angle of reflection obey Snell’s Law.
Snell’s Law will cause a track along an arbitrary direction in the symmetric quadrature set to reflect to the same direction in another octant. For tracks that cross the x- or y-faces of a system model, octants are coupled as follows:

- Octant 1 will reflect to Octant 4
- Octant 2 will reflect to Octant 3
- Octant 3 will reflect to Octant 2
- Octant 4 will reflect to Octant 1
- Octant 5 will reflect to Octant 8
- Octant 6 will reflect to Octant 7
- Octant 7 will reflect to Octant 6
- Octant 8 will reflect to Octant 5

Octant coupling at the z-faces of a system model is as follows:

- Octant 1 will reflect to Octant 5
- Octant 2 will reflect to Octant 6
- Octant 3 will reflect to Octant 7
- Octant 4 will reflect to Octant 8
- Octant 5 will reflect to Octant 1
- Octant 6 will reflect to Octant 2
- Octant 7 will reflect to Octant 3
- Octant 8 will reflect to Octant 4

Leakage from the system at external boundaries is modeled via user-specified albedoes, $\alpha_{\text{face}}$, which can range from a value of 1.0 for total reflection to a value of 0.0 for total
leakage. Six different albedoes must be specified prior to program execution, corresponding to the six external faces of the 3D Cartesian system.

For an arbitrary characteristic along direction $m$, the initial boundary value at an external system boundary is found via a lookup procedure that determines the direction $m'$ from which this track would be reflected, identifies the track and plane coupling at this point of reflection, and then locates the appropriate value in the neutron angular flux distribution array. Boundary values are not stored in a separate array, but rather are updated continuously as the inner iteration proceeds. This technique is consistent with a Gauss-Seidel iteration procedure, which reduces storage requirements and accelerates convergence. As described in figure 5.2, Equation (5.7) is now solved to find the neutron angular flux at the opposite endpoint for this track segment, which is set equal to the incoming angular flux for the next track segment in an adjacent cell, and so on for all track segments along this characteristic.

After the entire neutron angular flux distribution has been calculated, the neutron scalar flux is computed according to Equations (2.35) through (2.38) and the scattering source is updated via

$$
\bar{Q}^{i,g}_{\text{scatter}} = \frac{1}{4\pi} \sum_{g'=1}^{G} \sum_{\alpha} (\Sigma_{\alpha}^{i,g\rightarrow g'} \phi^{i,g'}) .
$$

(5.10)
At the end of the inner iteration for energy group $g$, convergence of the neutron angular flux is evaluated via

$$
\varepsilon_a = \max \left[ \Phi^{i,g}_{m,n,p} - \Phi^{i,g}_{m,n,p}^{T-1} \Phi^{i,g}_{m,n,p}^{T} \right] \quad (5.11)
$$

and convergence of the neutron scalar flux is evaluated according to

$$
\varepsilon_s = \max \left[ \phi^{i,g}_{m,n,p} - \phi^{i,g}_{m,n,p}^{T-1} \phi^{i,g}_{m,n,p}^{T} \right] . \quad (5.12)
$$

If either $\varepsilon_a$ or $\varepsilon_s$ is greater than the user-specified tolerance criteria, an additional inner iteration is performed for energy group $g$. Otherwise, inner iteration for group $g$ is completed and control passes to the inner iteration for the next energy group.

When the inner iteration is completed for all energy groups control is passed back to the outer iteration. The fission neutron distribution, $F$, is updated according to Equation (5.1) and $k_{eff}$ is computed via Equation (5.5). Finally, the fission neutron source is calculated from this updated value for $k_{eff}$,

$$
\overline{Q^{i,g}}_{\text{fission}} = \frac{1}{4\pi} \frac{2\pi^g}{k_{eff}} \sum_{g'='1}^G \Sigma^{i,g'} \phi^{i,g'} , \quad (5.13)
$$
and convergence of the outer iteration is evaluated by

\[ \varepsilon = \frac{k_{\text{eff}}^\tau - k_{\text{eff}}^{\tau-1}}{k_{\text{eff}}^\tau}. \]  \hspace{1cm} (5.14)

If \( \varepsilon \) is less than a user-specified tolerance criterion, \( \varepsilon_{\text{outer}} \), then the steady state outer iterations are concluded and control passes to the post-processing routines; otherwise, control passes to the next outer iteration.

If a time-dependent calculation is to be pursued upon completion of the MOCK-3D steady state computation, several parameters must be stored to an external file to be read as input upon execution of MOCK-3DK. These quantities include the converged angular and neutron scalar flux distributions, the fission neutron distributions, and \( k_{\text{eff}} \). The neutron source terms are also required by MOCK-3DK, however, these can be calculated directly from the converged scalar flux distribution and known material properties and need not be stored to file. As the angular flux distribution can be quite large, the size of this transient restart file has been minimized to minimize storage requirements.
5.4 Implementation of the Fully-Explicit Method of Characteristics

In Section 4.2, three different formulations for the fully explicit method of characteristics were derived that differ in the specific approximation that is introduced to simplify the spatial derivative of the neutron angular flux. In the first case, $\Phi_{m,n,p}^{i,g} |^T$ is replaced by the track averaged angular flux, $\overline{\Phi}_{m,n,p}^{i,g} |^T$, and the resulting formula is given by Equation (4.50). A higher order approximation assumes that $\Phi_{m,n,p}^{i,g} |^T$ is a linear function of position along a track, yielding Equation (4.53). Finally, Equation (4.58) is derived from the assumption that $\Phi_{m,n,p}^{i,g} |^T$ can be replaced by an exponential function. While each of these approaches yields a different solution to the time-dependent characteristic equation, all are found to satisfy the generalized formula

$$\Phi_{m,n,p}^{i,g} |_{out}^{T+1} = \Phi_{m,n,p}^{i,g} |_{out}^T - \nu^g \Delta t \left[ A \Phi_{m,n,p}^{i,g} |_{out}^T - B \Phi_{m,n,p}^{i,g} |_{in}^T - \overline{Q}^{i,g} \right], \quad (5.15)$$

where definition of the constants $A$ and $B$ is dependent on the specific angular flux approximation that has been employed. In the first case,

$$A = \Sigma_{\nu}^{i,g},$$

$$B = 0 \quad , \quad (5.16)$$
while the linear approximation yields

\[ A = \Sigma_{sr}^{\perp} + \frac{1}{\Delta s_{m,n,p}} \]

\[ B = \frac{1}{\Delta s_{m,n,p}} \]  

(5.17)

and constants for the exponential approximation are

\[ A = \Sigma_{sr}^{\perp} + \frac{1}{\Delta s_{m,n,p}} \]

\[ B = 0 \]  

(5.18)

Equation (5.15) is introduced for convenience, as it will simplify the discussion that follows; however, it does not necessarily follow that this general form will hold for all solutions to the forward-differenced characteristic equation. If a higher-order source term approximation is introduced, for example, additional terms may be introduced.

Equation (5.15) is mathematically quite unlike the steady state characteristic equation in that the neutron angular flux distribution at each time step is expressed entirely in terms of known parameters for the previous time step. As a result, computation of the forward-differenced method of characteristics can be pursued using a simple and direct procedure, as will be described in Section 5.4.1. Because Equation (5.15) can be solved for all points of interest in a single sweep across the problem space, the computation time for a single time step is expected to be minimal; however, stringent requirements for stability
of the forward differencing approximation will necessitate the definition of extremely small time steps. As a consequence, more time steps are required to complete a full transient calculation than with other time-dependent methods and the overall computational expense is expected to be severe. In Section 5.4.2, the problems associated with stability and accuracy of the forward-differenced method of characteristics will be explored in greater detail.

5.4.1 Numerical Solution of the Fully-Explicit Method of Characteristics

The forward-differenced characteristic equation solves for the neutron angular flux distribution at any time step given only the track set, material properties, and the angular and neutron scalar flux distributions from the previous time step. As a result, a simple and direct computational procedure can be employed to solve Equation (5.15), without recourse to iterative methods, as illustrated in figure 5.3.

![Figure 5.3: Computational Flow of Forward-Differenced Method of Characteristics](image-url)

155
During the pre-processing stage, a steady state method of characteristics calculation is performed to generate an initial converged flux distribution, which is required to calculate the neutron angular flux distribution for the first time step according to

$$\Phi^{i,g}_{m,n,p}\big|_{\text{out}}^{T_i} = \Phi^{i,g}_{m,n,p}\big|_{\text{out}}^{\text{SS}} - \nu^{g} \Delta t \left[ A \Phi^{i,g}_{m,n,p}\big|_{\text{out}}^{\text{SS}} - B \Phi^{i,g}_{m,n,p}\big|_{\text{in}}^{\text{SS}} - \overline{Q}^{i,g,\text{SS}} \right], \quad (5.19)$$

where $\Phi^{i}\text{SS}$ and $\overline{Q}^{\text{SS}}$ represent the converged steady state neutron angular flux and source term distributions, respectively. One component of the initial total neutron source – the delayed neutron source – is not calculated during the steady state computation, however, and must be estimated according to

$$\overline{Q}^{i,g}_{\text{delay}} = \frac{1}{4\pi} \sum_{l} \lambda_{i} C_{l}^{0}, \quad (5.20)$$

where $C_{l}^{0}$ represents the equilibrium concentration of delayed neutron precursors in the initial reactor state and is given by

$$C_{l}^{0} = \frac{\beta_{l}}{\lambda_{l}} \sum_{g} \nu \Sigma_{f}^{i,g} \overline{\phi}^{i,g,\text{SS}}. \quad (5.21)$$

Each time step proceeds by computing the neutron angular flux for all computational points in the problem space via Equation (5.15). In the steady state method of
characteristics, where the neutron angular flux at the location where each track exits a cell is dependent on the value at the entry location, a track sweep algorithm is required to complete this task, as illustrated by figure 5.2. In the present case, however, a traditional cell sweep algorithm is sufficient. Further, because all values on the right hand side of Equation (5.19) are known, the total angular flux distribution can be calculated in a single sweep across the problem. The neutron scalar flux is calculated using the same three-step procedure that was defined for the steady state method of characteristics in Chapter 2. First, track-averaged neutron angular flux is calculated via Equation (2.36). Cell-averaged neutron angular flux is next computed via Equation (2.37) and, finally, neutron scalar flux is found via summation over all angles in the quadrature set according to Equation (2.38). A time step computation concludes with the calculation of each component of the total neutron source via equations that have been derived above: Equation (4.43) for the scattered neutron source, Equation (4.44) for the prompt fission source, Equation (4.47) for the delayed neutron precursor concentrations, and Equation (4.45) for the delayed neutron source. Control then passes to the next time step and computation proceeds until the total problem time has elapsed.

### 5.4.2 Stability of the Fully-Explicit Method of Characteristics

The fully-explicit formulation for the time-dependent characteristic equation includes several features that render it of little utility to a practical space-time reactor kinetics methodology. For all explicit methods the issue of time step size is of critical importance. As was discussed in Chapter 4, time step size must be defined to be quite
small – for certain transients as small as $10^{-8}$ seconds – to avoid numerical instabilities; however, another result is that many time steps are required to complete a full computation. The failure of most explicit methods occurs because the potential gains in computational efficiency that result from its simple mathematical expression are more than offset by the expense associated with this requirement of small time steps.

A more fundamental problem with the explicit formulation for the time-dependent method of characteristics is seen upon closer examination of Equation (5.15). As has been stated previously, this equation differs significantly from the mathematical form of the solution to the steady state characteristic equation, Equation (2.34). The latter equation is derived by integrating the Lagrangian transport equation over the spatial domain, a technique that lies at the heart of the method of characteristics. Equation (5.15), on the other hand, includes no spatial integration and thus omits one of the most basic components of a characteristics technique.

The importance of the absence of spatial integration becomes clear if the time- and space-dependent neutron scalar flux is separated conceptually into amplitude and shape components, as is done formally in quasi-static methods. The role of spatial integration in the time-dependent method of characteristics is to evaluate the flux shape, while amplitude is most directly affected by the term $\nu^* \Delta t$ that appears as a multiplier to the neutron balance term in Equation (5.15). In the forward-differenced method of characteristics spatial integration is performed only once – during the initial steady state initialization calculation. The flux shape at the end of all time steps is derived directly
from this initial distribution and, accordingly, accuracy of the flux shape may degrade as more problem time elapses.

Empirical investigation is ultimately required to resolve questions as to the utility of the fully-explicit method of characteristics; however, significant doubt exists as to whether this formulation can be practically implemented into a space-time reactor kinetics code. During initial attempts to implement the forward-differenced method of characteristics into MOCK-3DK it was noted that the requirements of numerical stability are, in fact, more severe than have been reported for any other explicit method. In most cases, time step sizes smaller than $10^{-10}$ seconds were required. Such a requirement is excessively restrictive for all temporal reactor phenomena and, for this reason, it was decided that further empirical investigation of an explicit solution to the time-dependent method of characteristics would be neither necessary nor feasible at this stage in development.

5.5 Implementation of the Fully-Implicit Method of Characteristics

In contrast to an explicit method of characteristics solution, which is computed via the simple and direct procedures described in Section 5.4, implicit differencing yields a formulation that is mathematically similar to the steady state method of characteristics. Solution of the backward-differenced method of characteristics will therefore use methods that are similar to those presented in Section 5.3. In fact, the solution of Equation (4.69) at a particular time step, $T$, is pursued by assuming that the time-
dependent method of characteristics can be represented by a pseudo-steady state problem to which fixed source iteration is directly applicable. The result will be a three-level iterative computation, where an outer iterative layer is introduced to loop over all time steps in the total problem time. An overview of this technique is illustrated in figure 5.4.

Figure 5.4: Computational Flow of Backward-Differenced Method of Characteristics
MOCK-3DK is capable of modeling both step and ramp changes to the material properties for a transient problem. To do so, the total problem time is divided into a set of transient states that are defined by the user. Each state is divided into equal time step increments, although different states can utilize different time step sizes. When a step change in properties is introduced all material properties are assumed to be constant for the duration of any particular transient state, while in a ramped transient the perturbed material properties will change at a constant rate. A ramped perturbation must begin and end at the endpoint of one of the transient states. By providing to the user the freedom to specify the length of each transient state, the time step size within each state, and the type (or types) of perturbations to be introduced, any simple transient problem can be modeled. In addition, a “null transient” – that is, a transient calculation without any perturbation to the steady state material properties – can be modeled by defining a single transient state with a step change perturbation, but specifying all properties to be identical to the steady state values.

5.5.1 Numerical Solution of the Fully-Implicit Method of Characteristics

The first step in a MOCK-3DK computation is to initialize all state variables using the information that was stored to the transient restart file at the end of the MOCK-3D steady state calculation. As explained in Section 5.3, the converged angular and neutron scalar flux distributions are read directly from this file, while components of the neutron source term are reconstructed using the appropriate definitions. Thus, the scattered and prompt fission neutron source terms are computed via Equations (4.66) and (4.67), respectively.
As in the case of the forward-differenced method of characteristics, the initial delayed neutron precursor concentrations and delayed neutron source term must be estimated using Equations (5.21) and (5.20). If a ramp perturbation to the material properties has been specified for any of the transient problem states, ramp increments are also calculated at this time.

Each MOCK-3DK time step will include a full fixed source computation akin to the techniques used in the MOCK-3D steady state calculation. The time-dependent method of characteristics is thus transformed to a pseudo-steady state problem and, accordingly, the delayed neutron precursor concentrations are held constant for the duration of the time step. Within the fixed source iteration an eigenvalue, $\gamma$, is introduced to the outer iteration. In contrast to the steady state, this eigenvalue is not identified with any physical property of the system and does not directly impact calculation of the neutron source. As will be demonstrated below, $\gamma$ is introduced instead to facilitate proper convergence of the total fission source during the outer iteration.

As in the steady state method of characteristics, the outer iteration begins by holding the total fission neutron source constant, which includes both the prompt and delayed neutron components, and control passes immediately to the inner iteration. Inner iteration is pursued for each energy group separately, beginning with the highest energy group, and upscattering is forbidden. The angular flux distribution for energy group $g$ is calculated via the track-sweep technique illustrated in figure 5.2 and using an appropriate solution
for the backward-differenced characteristics equation; thus, the constant angular flux approximation yields

\[
\Phi_{\text{out}}^{i,g} = \Phi_{\text{in}}^{i,g} + \Phi_{\text{out}}^{i,g} + \frac{1}{\Sigma_{\text{tr}}} \left[ \frac{1}{v^g \Delta t} \right] \left( 1 - P_{\text{out}}^{i,g} \right),
\]

the linear approximation is

\[
\Phi_{\text{out}}^{i,g} = \Phi_{\text{in}}^{i,g} + \frac{1}{\Sigma_{\text{tr}}} \left[ \frac{1}{v^g \Delta t} \right] \left( 1 - P_{\text{out}}^{i,g} \right) \Phi_{\text{out}}^{i,g} + \left( 1 - \frac{1}{\Sigma_{\text{tr}}} \right) \left( 1 - P_{\text{out}}^{i,g} \right) \Phi_{\text{out}}^{i,g},
\]

and the exponential flux approximation will be

\[
\Phi_{\text{out}}^{i,g} = \Phi_{\text{in}}^{i,g} + \frac{1}{\Sigma_{\text{tr}}} \left[ \frac{1}{v^g \Delta t} \right] \left( 1 - P_{\text{out}}^{i,g} \right) \Phi_{\text{out}}^{i,g} + \left( 1 - \frac{1}{\Sigma_{\text{tr}}} \right) \left( 1 - P_{\text{out}}^{i,g} \right) \Phi_{\text{out}}^{i,g}.
\]
where

\[ A_i = \frac{\Sigma_{\text{tr}}^{i,g}}{\Delta s_{m,n,p}} \left( \frac{1}{\Delta s_{m,n,p}} \right) \ln \left( \frac{\Phi_{m,n,p}^{i,g} e^{\Sigma_{\text{tr}}^{i,g} \Delta s_{m,n,p}}}{\Phi_{m,n,p}^{i,g} e^{\Sigma_{\text{tr}}^{i,g} \Delta s_{m,n,p}}} \right) \]  \hspace{1cm} (5.25)

In all three cases the non-collision probability, \( P_{m,n,p}^{i,g} \), has again been introduced. In the time-dependent method of characteristics \( P_{m,n,p}^{i,g} \) is defined in terms of the effective transport cross-section, \( \Sigma_{\text{tr}}^{i,g} \), as

\[ P_{m,n,p}^{i,g} = e^{\frac{\Sigma_{\text{tr}}^{i,g} \Delta s_{m,n,p}}{\Delta t}} = e^{-\left( \frac{1}{\Delta t} \Sigma_{\text{tr}}^{i,g} \Delta s_{m,n,p} \right)}. \]  \hspace{1cm} (5.26)

The non-collision probability table is not fixed, however, as either the time step size, \( \Delta t \), or the macroscopic transport cross section, \( \Sigma_{\text{tr}}^{i,g} \), may be perturbed during the transient. Instead, non-collision probabilities must be updated periodically as will be shown below.

When the neutron angular flux distribution has been estimated for group \( g \), the neutron scalar flux distribution and scattered neutron source term are updated using the same procedures defined in Section 5.3 for the forward-differenced formulation. Finally, convergence of the angular and neutron scalar flux distributions is evaluated using Equations (5.11) and (5.12). When both distributions are fully converged for group \( g \) the inner iteration is repeated for all lower energy groups.
Upon completion of the inner iteration for all energy groups, the outer iteration proceeds in essentially the same manner as for the steady state computation. The fission source distribution, $F_i^j$, is first calculated according to Equation (5.1). Equation (5.5) is recast in terms of the time step eigenvalue, $\gamma$, so that $\gamma$ is updated for the $j^{th}$ outer iteration according to

$$
\gamma^{(j)} = \frac{\sum_i F_i^{(j)}}{\sum_i F_i^{(j-1)}}.
$$

Finally, the prompt neutron source and delayed neutron source are updated using Equations (4.67) and (4.68), respectively. Convergence of the outer iteration is evaluated via Equation (5.14), as in the steady state method of characteristics.

Upon completion of the fixed source iteration for time step $T$, the delayed neutron precursor concentrations are calculated via Equation (4.61) and the total core power is evaluated using the normalized formula

$$
q^T = \frac{\sum \sum_i \Sigma_{if} \phi_i g dx dy dz}{q^0}.
$$

where $q^0$ represents the core power at $t=0$ and is calculated during problem initialization. In addition, the array that holds the neutron angular flux distribution for the previous time step, which is required to solve Equations (5.22) through (5.24), is updated.
5.5.2 Stability of the Fully-Implicit Method of Characteristics

In general, a fully-implicit numerical solution to the time-dependent transport equation will be unconditionally stable. One exception to this rule, as was noted in Chapter 4, occurs when extremely rapid temporal phenomena are observed. In such cases, numerical stability will depend on the definition of small time step sizes; however, small time sizes would be otherwise required to accurately model such phenomena and this constraint is rendered moot. In fact, the need for accuracy is typically more important to the determination of appropriate time step size for an implicit or semi-implicit method than are the requirements of computational stability.

When implicit time discretization is applied to the method of characteristics, with its unique Lagrangian description of the spatial domain, a peculiar numerical instability is observed. To illustrate this problem, the simplest formulation for the forward-differenced characteristic equation, Equation (4.70), is restated in expanded form and with most of the spatial, angular, and energy indices suppressed for clarity,

\[
\Phi_i^{n+1} = \Phi_i^n e^{-\left(\frac{1}{\phi} \Sigma_t \Delta t\right)} + \left(\frac{D}{\Sigma_{in}}\right)\left(1 - e^{-\left(\frac{1}{\phi} \Sigma_t \Delta t\right)}\right) + \left(\frac{1}{\phi} \Delta t\right)\left(1 - e^{-\left(\frac{1}{\phi} \Sigma_t \Delta t\right)}\right).
\]

From previous experience with the steady state method of characteristics it is known that the probability that a neutron will not experience a collision during the time interval \(\Delta t\) as
it traverses the track length $\Delta s$ is given by $e^{-\frac{1}{v_{tr} \Delta t}}$ and, therefore, $1 - e^{-\frac{1}{v_{tr} \Delta t} \Delta s}$ is the probability that a neutron does experience a collision. Given these definitions, the three terms on the right hand side of Equation (5.29) can be interpreted physically as a neutron streaming term, a neutron source term, and a neutron loss term. As with the steady state characteristic equation, neutron balance is maintained where upscattering has been forbidden and Equation (5.29) is conservative. If upscattering is allowed an additional corrective measure would be required to preserve neutron balance. Of particular concern in Equation (5.29) is the occurrence of the term $\frac{1}{v \Delta t}$, which occurs twice: within each instance of the collision or non-collision probability and as a multiplier to the neutron loss term.

In the case where $\frac{1}{v \Delta t}$ appears within the loss term, a singularity occurs for $\Delta t = 0$ as $\frac{1}{0} \equiv \infty$ and the loss term thus becomes infinite. If $\Delta t$ is instead defined to be infinitesimally small but non-zero, the loss term will still be much larger than the source and streaming terms and will therefore dominate the solution to Equation (5.29). To avoid numerical instability due to this singularity, time step size must be defined to be larger than an undetermined minimum threshold value. This potential instability is of little practical importance, however, because in order for the singularity to adversely affect the results of an implicit calculation, $\Delta t$ must be much smaller than would normally be desired. In fact, a similar singularity appears in spatial finite differencing forms of the reactor kinetics equations as well and has been successfully resolved in all instances.
The appearance of $\frac{1}{v\Delta t}$ within the collision and non-collision probability terms is more problematic and, because these terms are a direct result of the method of characteristics spatial integration technique, appears to be unique to a Lagrangian formulation for the time-dependent transport equation. In this case, if $\Delta t$ is defined to be arbitrarily small then $\frac{1}{v\Delta t}$ will be much larger than $\bar{\Sigma}_r$ and the relevant material effects – i.e., actual neutron collision events – are lost from the time-dependent transport solution and

$$\lim_{\Delta t \to 0} e^{-\left(\frac{1}{v\Delta t} + \bar{\Sigma}_r\right)\Delta s} \Rightarrow e^{-\infty} \Rightarrow 0.$$  \hspace{1cm} (5.30)

A non-collision property of zero implies that every neutron undergoes collision, a result that would seem to be counter-intuitive. It could be logically deduced that as the time step size approaches zero then there will be insufficient time for neutron collision events to occur and, therefore, that the non-collision probability should approach a value of 1.

The apparent inconsistency identified above can be resolved by considering the physics of a time-dependent propagation problem. Given an arbitrary but finite track length, $\Delta s$, a neutron with given velocity (magnitude) will require a certain amount of time to traverse the entire length of this track. This minimum time step size is given by

$$\hat{\Delta t} = \frac{\Delta s}{v}. \hspace{1cm} (5.31)$$
If $\Delta t$ is specified to be smaller than the minimum time step size, $\Delta_t$, then a neutron that originates at one endpoint of the track, $\Delta s$, will not reach the opposite endpoint of this track and will be lost, exactly as if it had undergone a collision event. In fact, as $v^g$ represents the mean velocity for neutrons in energy group $g$, it is assumed that all neutrons in this group move at the same velocity and, as a result, all neutrons are lost. In other words, if time step size is specified to be smaller than the as yet undetermined minimum threshold value, a temporal pseudo-collision process leads to the removal of all neutrons from the system and a break down in solution to the time-dependent transport equation.

The precise value for the minimum time step threshold, $\Delta_t$, will depend on the material properties and spatial discretization of the particular problem of interest. In the fine-mesh discretizations that are commonly employed with the method of characteristics a reasonable estimate for $\Delta s$ is 0.5 cm, while the velocity of thermal neutrons in a typical two-group energy structure is $10^5$ cm/sec. Because thermal neutrons have the slowest velocities in a reactor, these will require the longest time steps to traverse the length $\Delta s$ and, therefore, bound the determination of a minimum time step threshold. Similarly, $\Delta_t$ is bounded by the longest track length in the system model. When these sample values are inserted into Equation (5.31), $\Delta_t$ is found in this example to be

$$\Delta t \approx \frac{0.5 \text{cm}}{10^5 \text{cm/sec}} = 5 \times 10^{-6} \text{sec}.$$  (5.32)
This estimate of Equation (5.32) for the minimum time step size requirement is troublesome, as it is large enough to potentially impede the modeling of fast transients. Moreover, as $\Delta t$ must typically be less than $10^{-3}$ seconds for reasons of accuracy, a very small window of possible time step sizes is available and significant freedom is lost to the user in modeling practical transient reactor phenomena.

The problem of a minimum time step threshold for the time-dependent method of characteristics will arise from any technique that discretizes the time derivative of the neutron angular flux using an approximation of the form

$$\frac{\partial}{\partial t} \rightarrow \frac{1}{\Delta t}.$$  \hspace{1cm} (5.33)

Accordingly, implicit and semi-implicit formulations for the $\Theta$ method, Runge-Kutta methods, and many quasi-static methods all will be vulnerable to this unique instability. Of the time-dependent methods reviewed in Chapter 4, only the Stiffness Confinement Method explicitly avoids this complication via replacement of the time derivative in the the time-dependent transport equation with the dynamic frequency defined in Equation (4.26). Future investigations into a time-dependent method of characteristics should, therefore, include a study of SCM and other less common mathematical techniques as a means to avoid this particular problem.
CHAPTER 6:
DEVELOPMENTAL ASSESSMENT OF
THE MOCK-3D CODE PACKAGE

The final task remaining to complete the objectives set out in Chapter 1 is to assess empirically the methods that have been derived and implemented into MOCK-3D in the previous chapters. To accomplish this task, a set of test problems has been assembled that evaluate the performance of the various MOCK-3D code components for various steady state and time-dependent phenomena. The modular nature of MOCK-3D facilitates a natural coupling between code components as suits the needs of assessment for the two topics of this work: explicit 3D Cartesian modeling capability and a time-dependent method of characteristics computation. In Section 6.1 assessment of the steady state module MOCK-3D will be pursued with attention paid to the evaluation of the 3D modeling capability. Subsequently, testing of the time-dependent module in Section 6.2 will concentrate on the time-dependent nature of these computations.

6.1 Developmental Assessment of MOCK-3D

Three test problems of varying complexity are included in the MOCK-3D steady state test suite. The first simulates a single simplified BWR fuel assembly in 2D Cartesian geometry and is presented in Section 6.1.1. A second 2D problem investigates the effects
present at an interface between adjacent mixed-oxide and uranium-oxide fuel assemblies and is discussed in Section 6.1.2. These first two problems have been developed specifically for the developmental assessment of MOCK-3D and have utilized a special version of the CASMO-4 lattice physics code\textsuperscript{48,75} to generate material properties and reference solutions for homogeneous cells. CASMO-4 is based on a steady state method of characteristics that is quite similar to the methods implemented in MOCK-3D, making comparison of 2D steady state results from these two packages a useful assessment tool.

The third problem in the steady state test suite, analyzed in Section 6.1.3, is a formal benchmark problem for the testing of 3D neutron transport methodologies – the OECD/NEA 3D Neutron Transport Benchmark Problem\textsuperscript{94}. This problem is more useful as a validation tool for the 3D steady state method of characteristics, thus a series of sensitivity studies are conducted to investigate the effects of several important parameters on the accuracy of the MOCK-3D solution. These parameters include the cell size, track separation, quadrature, and inner iteration convergence criterion. Following these initial studies, a final set of results for the Takeda problem is generated and compared to the reference solution. Selected results from this section have been presented previously by the author in Reference 95.

\textbf{6.1.1 2D Steady State Mini-BWR Assembly Problem}

The 2D steady state mini-BWR assembly problem is a simplified 2D simulation of a miniature BWR fuel assembly, the surrounding coolant, and a portion of a fully inserted
cruciform control rod. The fuel assembly is square with x- and y- dimensions equal to 3.2 cm and is centered inside a 4.8 cm square coolant channel. One quarter of the control rod is modeled, i.e., half of two of the four cruciform blades, with each wing of the control blade being 2.4 cm long and 0.4 cm wide. Figure 6.1 illustrates a model that is decomposed into uniform 0.4 cm square cells.

![Figure 6.1: Problem Schematic for 2D Mini-BWR Assembly Problem](image)

One-group material properties for each of the three regions in this problem were calculated using the CASMO-4 lattice physics code and are provided in Table 6.1.
Table 6.1: Material Properties for 2D Mini-BWR Assembly Problem

<table>
<thead>
<tr>
<th>Region</th>
<th>$\Sigma_{tr}$ (cm$^{-1}$)</th>
<th>$\Sigma_a$ (cm$^{-1}$)</th>
<th>$\nu\Sigma_f$ (cm$^{-1}$)</th>
<th>$\Sigma_s$ (cm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel</td>
<td>0.436526</td>
<td>0.0251889</td>
<td>0.0350929</td>
<td>0.411337</td>
</tr>
<tr>
<td>Coolant</td>
<td>0.487382</td>
<td>0.00254071</td>
<td>0.0</td>
<td>0.484841</td>
</tr>
<tr>
<td>Absorber</td>
<td>0.643697</td>
<td>0.1</td>
<td>0.0</td>
<td>0.543697</td>
</tr>
</tbody>
</table>

CASMO-4 was also utilized to generate a reference solution for this assembly configuration using input specifications that are consistent with the 0.4 cm square grid of figure 6.1. The CASMO-4 reference value for the effective multiplication factor, $k_{eff}$, is calculated to be 0.82904, while reference distributions for the 2D normalized neutron scalar flux and the 2D normalized fission rate are provided in Tables 6.2 and 6.3.

Table 6.2: Reference Normalized Scalar Flux Distribution for 2D Mini-BWR Assembly Problem

<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0.806</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.823</td>
<td>0.859</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.842</td>
<td>0.886</td>
<td>0.921</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.862</td>
<td>0.908</td>
<td>0.944</td>
<td>0.969</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.884</td>
<td>0.929</td>
<td>0.965</td>
<td>0.987</td>
<td>1.004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.914</td>
<td>0.953</td>
<td>0.983</td>
<td>1.003</td>
<td>1.018</td>
<td>1.029</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.962</td>
<td>0.978</td>
<td>1.000</td>
<td>1.016</td>
<td>1.028</td>
<td>1.038</td>
<td>1.045</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.986</td>
<td>0.995</td>
<td>1.012</td>
<td>1.025</td>
<td>1.035</td>
<td>1.043</td>
<td>1.049</td>
<td>1.052</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.998</td>
<td>1.006</td>
<td>1.020</td>
<td>1.031</td>
<td>1.039</td>
<td>1.045</td>
<td>1.050</td>
<td>1.053</td>
<td>1.053</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.005</td>
<td>1.011</td>
<td>1.022</td>
<td>1.031</td>
<td>1.037</td>
<td>1.042</td>
<td>1.046</td>
<td>1.048</td>
<td>1.049</td>
<td>1.045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.008</td>
<td>1.012</td>
<td>1.017</td>
<td>1.023</td>
<td>1.028</td>
<td>1.033</td>
<td>1.036</td>
<td>1.038</td>
<td>1.037</td>
<td>1.034</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.009</td>
<td>1.012</td>
<td>1.016</td>
<td>1.020</td>
<td>1.025</td>
<td>1.028</td>
<td>1.032</td>
<td>1.034</td>
<td>1.034</td>
<td>1.033</td>
<td>1.032</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
Table 6.3: Reference Normalized Fission Rate Distribution for 2D Mini-BWR Assembly Problem

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>1 2 3 4 5 6 7 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.900</td>
<td>0.924</td>
<td>0.943</td>
<td>0.962</td>
<td>0.978</td>
<td>0.990</td>
<td>0.998</td>
<td>1.000</td>
<td>1.008 1.014 1.020 1.023 1.025 1.025 1.022</td>
</tr>
<tr>
<td></td>
<td>0.947</td>
<td>0.977</td>
<td>0.986</td>
<td>0.995</td>
<td>1.006</td>
<td>1.003</td>
<td>1.008</td>
<td>1.008</td>
<td>1.016 1.022 1.027 1.027 1.029 1.029</td>
</tr>
<tr>
<td></td>
<td>0.982</td>
<td>1.015</td>
<td>1.013</td>
<td>1.020</td>
<td>1.026</td>
<td>1.020</td>
<td>1.022</td>
<td>1.023</td>
<td>1.020 1.023 1.025 1.025 1.027 1.029</td>
</tr>
<tr>
<td></td>
<td>1.007</td>
<td>1.022</td>
<td>1.026</td>
<td>1.027</td>
<td>1.029</td>
<td>1.026</td>
<td>1.027</td>
<td>1.025</td>
<td>1.022 1.022 1.022 1.022 1.022 1.022</td>
</tr>
</tbody>
</table>

Solution of this problem by MOCK-3D was sought using the same 12 x 12 square lattice as in the CASMO-4 reference calculation. The third, axial, spatial dimension is modeled via a single 0.4 cm high axial level and albedoes of 1.0 are placed at all six external boundaries, thus treating the assembly as reflective with infinite height. The initial track separation is specified to be 0.08 cm and S₄ level symmetric quadrature is used.

The MOCK-3D steady state calculation yields $k_{eff}$ of 0.82427, which is 570 pcm below the reference value. The MOCK-3D normalized neutron scalar flux is tabulated in Table 6.4, with a map of the percent difference from the CASMO-4 reference solution following in Table 6.5. The normalized fission rate distribution and comparison to the reference are provided in Tables 6.6 and 6.7.
Table 6.4: MOCK-3D Normalized Scalar Flux Distribution for 2D Mini-BWR Assembly Problem

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0.830</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.832</td>
<td>0.859</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.849</td>
<td>0.883</td>
<td>0.915</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.871</td>
<td>0.907</td>
<td>0.940</td>
<td>0.965</td>
</tr>
<tr>
<td>8</td>
<td>0.892</td>
<td>0.930</td>
<td>0.963</td>
<td>0.984</td>
</tr>
<tr>
<td>7</td>
<td>0.922</td>
<td>0.956</td>
<td>0.982</td>
<td>1.000</td>
</tr>
<tr>
<td>6</td>
<td>0.968</td>
<td>0.976</td>
<td>0.994</td>
<td>1.011</td>
</tr>
<tr>
<td>5</td>
<td>0.991</td>
<td>0.994</td>
<td>1.007</td>
<td>1.021</td>
</tr>
<tr>
<td>4</td>
<td>1.003</td>
<td>1.005</td>
<td>1.015</td>
<td>1.026</td>
</tr>
<tr>
<td>3</td>
<td>1.010</td>
<td>1.011</td>
<td>1.019</td>
<td>1.026</td>
</tr>
<tr>
<td>2</td>
<td>1.016</td>
<td>1.016</td>
<td>1.016</td>
<td>1.020</td>
</tr>
<tr>
<td>1</td>
<td>1.018</td>
<td>1.018</td>
<td>1.016</td>
<td>1.017</td>
</tr>
</tbody>
</table>

Table 6.5: Percent Difference between MOCK-3D and Reference Scalar Flux Distributions for 2D Mini-BWR Assembly Problem

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>2.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1.14</td>
<td>-0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.82</td>
<td>-0.34</td>
<td>-0.65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.00</td>
<td>-0.15</td>
<td>-0.41</td>
<td>-0.38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.86</td>
<td>0.16</td>
<td>-0.23</td>
<td>-0.27</td>
<td>-0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.88</td>
<td>0.27</td>
<td>-0.12</td>
<td>-0.32</td>
<td>-0.26</td>
<td>-0.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.60</td>
<td>-0.17</td>
<td>-0.57</td>
<td>-0.54</td>
<td>-0.54</td>
<td>-0.60</td>
<td>-0.60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.54</td>
<td>-0.12</td>
<td>-0.48</td>
<td>-0.43</td>
<td>-0.62</td>
<td>-0.68</td>
<td>-0.64</td>
<td>-0.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.46</td>
<td>-0.12</td>
<td>-0.45</td>
<td>-0.51</td>
<td>-0.57</td>
<td>-0.54</td>
<td>-0.58</td>
<td>-0.65</td>
<td>-0.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.48</td>
<td>0.01</td>
<td>-0.30</td>
<td>-0.52</td>
<td>-0.57</td>
<td>-0.50</td>
<td>-0.43</td>
<td>-0.40</td>
<td>-0.52</td>
<td>-0.34</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.76</td>
<td>0.39</td>
<td>-0.05</td>
<td>-0.26</td>
<td>-0.21</td>
<td>-0.22</td>
<td>-0.14</td>
<td>-0.12</td>
<td>-0.10</td>
<td>-0.03</td>
<td>0.35</td>
</tr>
<tr>
<td>1</td>
<td>0.89</td>
<td>0.61</td>
<td>-0.05</td>
<td>-0.30</td>
<td>-0.22</td>
<td>-0.07</td>
<td>-0.23</td>
<td>-0.21</td>
<td>-0.09</td>
<td>0.03</td>
<td>0.43</td>
</tr>
</tbody>
</table>

176
Table 6.6: MOCK-3D Normalized Fission Rate Distribution for 2D Mini-BWR Assembly Problem

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.899</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.924</td>
<td>0.949</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.946</td>
<td>0.967</td>
<td>0.986</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.965</td>
<td>0.983</td>
<td>0.998</td>
<td>1.008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.977</td>
<td>0.993</td>
<td>1.005</td>
<td>1.014</td>
<td>1.021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.990</td>
<td>1.003</td>
<td>1.011</td>
<td>1.018</td>
<td>1.024</td>
<td>1.028</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.998</td>
<td>1.008</td>
<td>1.015</td>
<td>1.021</td>
<td>1.026</td>
<td>1.028</td>
<td>1.028</td>
</tr>
<tr>
<td>1</td>
<td>1.002</td>
<td>1.008</td>
<td>1.013</td>
<td>1.019</td>
<td>1.024</td>
<td>1.026</td>
<td>1.025</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6.7: Percent Difference between MOCK-3D and Reference Fission Rate Distributions for 2D Mini-BWR Assembly Problem

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>-0.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-0.02</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.34</td>
<td>0.12</td>
<td>0.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.29</td>
<td>0.16</td>
<td>0.27</td>
<td>0.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-0.09</td>
<td>-0.10</td>
<td>-0.14</td>
<td>-0.11</td>
<td>-0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-0.05</td>
<td>0.01</td>
<td>-0.22</td>
<td>-0.18</td>
<td>-0.16</td>
<td>-0.12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-0.04</td>
<td>0.01</td>
<td>-0.09</td>
<td>-0.08</td>
<td>-0.11</td>
<td>-0.08</td>
<td>-0.08</td>
</tr>
<tr>
<td>1</td>
<td>0.15</td>
<td>-0.03</td>
<td>-0.09</td>
<td>-0.11</td>
<td>0.05</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Finally, figures 6.2 and 6.3 show 2D plots that correspond to the values of Table 6.4 for the neutron scalar flux distribution and Table 6.6 for the fission rate distribution, respectively, as computed by MOCK-3D.
Figure 6.2: MOCK-3D Normalized Scalar Flux Distribution for Mini-BWR Assembly Calculation

Figure 6.3: MOCK-3D Normalized Fission Rate Distribution for Mini-BWR Assembly Calculation
The largest error in the MOCK-3D neutron scalar flux distribution can be seen in Table 6.5 to occur at the location nearest the center of the cruciform rod, where scalar flux is over-predicted by ~3%. This error decreases significantly with distance from the rod and is less than 1% throughout the assembly and coolant regions. Such a pattern of error indicates that the MOCK-3D calculation is failing in the presence of a strong absorber, an effect that can occur with diffusion theory methods as discussed in Chapter 1. While the method of characteristics is expected to be capable of treating such a situation, the specified geometry of this model may be too coarse to accurately capture the transport effects that occur near the control rod blade. Typically, cell size should be specified to be no larger than the transport mean free path, $1/\Sigma_r$, which is in this instance approximately 1.5 cm. A cell size of 0.4 cm does meet the mean free path criterion, although the value for $1/\Sigma_r$ is larger than encountered for more realistic problems. Track separation and quadrature order also have an effect on the overall problem geometry and a more complete investigation of the effect of these parameters will be pursued for the more rigorous test problems that follow.

### 6.1.2 2D Steady State MOX/UOX Assembly Interface Problem

The MOX/UOX Assembly Interface Problem is another small 2D test problem, designed to simulate the interface effects between mixed-oxide (MOX) and uranium-oxide (UOX) fuel assemblies in a typical light water reactor core. A single 12 x 12 MOX fuel assembly that is surrounded by UOX assemblies of identical dimension is modeled in quarter-core symmetry as shown in figure 6.4.
Each cell in figure 6.4 corresponds to a fuel pin with x- and y-dimensions of 1.435 cm. Total x- and y-dimensions of the system model are thus 25.83 cm. Preliminary results for this problem were calculated using an earlier version of the MOCK-3D code and published in Reference 96. The results presented in this section are intended to replace those preliminary, and now obsolete, results.

The MOX/UOX interface is of particular interest to the design of fuel loading patterns for commercial light water reactors, which are beginning to incorporate MOX fuel assemblies as a means to dispose of reprocessed spent fuel. Such fuel loading patterns typically contain adjacent MOX and UOX assemblies, and the simulation of such a
loading pattern with nodal diffusion theory is known to be problematic. At the interface of adjacent MOX and UOX assemblies a discontinuity can result due to differences in the material properties of these two assembly types. As discussed in Chapter 1, the MOX/UOX interface is one of the problems that motivate the search for improvements to traditional nodal diffusion theory methodologies for steady state and time-dependent analysis of reactor cores.

To investigate the effect of cell size and track separation on the accuracy of MOCK-3D near the MOX/UOX interface, three different models were developed. Model 1 uses the same discretization depicted in figure 6.4; i.e., a cell size of 1.435 cm and track separation of 0.3 cm are applied. In Model 2 cell size is 0.7175 cm and track separation equal to 0.15 cm, so that each of the fuel pins in figure 6.4 is decomposed into four computational cells. Model 3 uses 16 cells per fuel pin, with cell size of 0.35875 cm and track spacing equal to 0.075 cm. In all three models the $S_4$ level symmetric quadrature set is used. For comparison purposes, a fourth calculation is performed via the NEM-3D nodal diffusion theory code, using the spatial model described by MOCK-3D Model 1.

Two-group cross-sections for the MOX and UOX regions were generated in CASMO-4 and are provided in Tables 6.8 and 6.9, where it is assumed that all fission neutrons are born in the fast energy group.
Table 6.8: Material Properties for 2D MOX/UOX Assembly Interface Problem, MOX Region

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>$\Sigma_{tr,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{a,g}$ (cm$^{-1}$)</th>
<th>$\nu\Sigma_{f,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{n,1,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{tr,2,g}$ (cm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.268897</td>
<td>0.0113543</td>
<td>0.00749139</td>
<td>0.235804</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>1.40184</td>
<td>0.193106</td>
<td>0.284158</td>
<td>0.0218118</td>
<td>1.20873</td>
</tr>
</tbody>
</table>

Table 6.9: Material Properties for 2D MOX/UOX Assembly Interface Problem, UOX Region

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>$\Sigma_{tr,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{a,g}$ (cm$^{-1}$)</th>
<th>$\nu\Sigma_{f,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{n,1,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{tr,2,g}$ (cm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.275562</td>
<td>0.00918293</td>
<td>0.00670822</td>
<td>0.241885</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>1.31947</td>
<td>0.11604</td>
<td>0.154011</td>
<td>0.0245474</td>
<td>1.20343</td>
</tr>
</tbody>
</table>

No reference solution was generated for this problem, which is nonetheless useful as a qualitative demonstration of the capabilities of a 3D method of characteristics. Converged values for $k_{eff}$ for each of the four models are presented in Table 6.10.

Table 6.10: MOCK-3D $k_{eff}$ for 2D MOX/UOX Interface Problem

<table>
<thead>
<tr>
<th>Case</th>
<th>$k_{eff}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>1.171245</td>
</tr>
<tr>
<td>Model 2</td>
<td>1.171095</td>
</tr>
<tr>
<td>Model 3</td>
<td>1.171027</td>
</tr>
<tr>
<td>NEM</td>
<td>1.169060</td>
</tr>
</tbody>
</table>

Finally, a series of 2D plots of the normalized neutron scalar flux and fission rate distributions are provided for spatial Model 3 in figures 6.5 through 6.8.
Figure 6.5: MOCK-3D Group 1 Normalized Scalar Flux Distribution for MOX/UOX Assembly Interface Problem

Figure 6.6: MOCK-3D Group 2 Normalized Scalar Flux Distribution for MOX/UOX Assembly Interface Problem
Figure 6.7: MOCK-3D Group 1 Normalized Fission Rate Distribution for MOX/UOX Assembly Interface Problem

Figure 6.8: MOCK-3D Group 2 Normalized Fission Rate Distribution for MOX/UOX Assembly Interface Problem
The problem presented here does not include a reference solution, so only qualitative assessment can be made at this stage. A more thorough computational benchmark problem for MOX cores has recently been published and would be an excellent addition to future MOCK-3D test suites. Nonetheless, several interesting MOX/UOX interface effects can be observed from the simple problem posed above. Near the interface of the MOX and UOX assemblies, differences in the physical properties of these two fuel types combine to create a discontinuity in the thermal fission rate. Absorption and fission cross-sections are higher in the MOX fuel than in UOX, especially at thermal energies, so there is a preferential absorption by the MOX fuel near the assembly interface. Thus, on the UOX side of the interface the MOX fuel acts as a poison, reducing the thermal flux and fission distribution in this region. On the MOX side of the interface, the opposite effect is seen – the flux depression in the UOX increases thermal leakage to the MOX assembly, increasing the thermal flux and causing a sharp rise in fission density. Because the absorption in MOX is strong, this effect is seen within only a few millimeters of the interface. The fact that MOCK-3D can capture these effects suggests that the desired accuracy may be achievable with finer spatial resolution.

6.1.3 OECD/NEA 3D Neutron Transport Benchmark Problem

The 3D Neutron Transport Benchmark Problem was conducted by researchers from Osaka University from 1988 through 1991 under the auspices of the OECD/NEA Committee on Reactor Physics and is commonly referred to as the “Takeda Problem” after the principle investigator. Twenty-two organizations participated in the original
benchmark project and, since the publication of final results in 1991, this problem has been used as a standard test problem for various 3D neutron transport codes. The Takeda benchmark includes four distinct problems, three of which (Models 2 through 4) involve relatively large and spatially complex calculations for a fast breeder reactor. Model 1 is a simple model of a small solid-moderated research reactor and is the only problem of interest to the present study.

6.1.3.1 Takeda Model 1 Problem Description

The reactor core simulated by Takeda Model 1 is based on the Kyoto University Critical Assembly (KUCA) solid-moderated test reactor, in operation at the Kyoto University Research Reactor Institute since 1974. The simulated cubic reactor configuration has total dimensions of 50.0 cm cubic and consists of a 30.0 cm cubic reactor core that is surrounded on all sides by reflector material. The fuel is comprised of a combination of 93 w/o enriched U-Al alloy plates and natural uranium plates, with a 9.6 w/o core-averaged $^{235}$U enrichment. The core is moderated by polyethylene with a moderator volume to fuel volume ratio of 1.5. Two full-length axial control rods are located within the reflector region, immediately adjacent to the core, at the positions indicated in figure 6.9, which models the reactor core in 1/8 core symmetry with 1.0 cm cubic cells.
Specifications for this benchmark problem call for two calculations to be performed. The first calculation, Model 1a, assumes that the control rod is totally withdrawn and the material in the control rod region is modeled as a void. In Model 1b the control rod is fully inserted. This problem thus includes three material regions for which four different sets of material properties are specified: (1) the core region, with material properties specified in Table 6.11; (2) the reflector region, with material properties specified in Table 6.12; (3) the control rod region with the rod fully withdrawn (i.e., void), with properties given in Table 6.13; and (4) the control rod region with the rod fully inserted, with properties given in Table 6.14.
Table 6.11: Material Properties for Takeda Benchmark Problem, Model 1, Core Region

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>$\Sigma_{tr,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{a,g}$ (cm$^{-1}$)</th>
<th>$\nu\Sigma_{\xi,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{n,1,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{tr,2,g}$ (cm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.23775E-01</td>
<td>8.52709E-03</td>
<td>9.09319E-03</td>
<td>1.92423E-01</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>2</td>
<td>1.03864E+00</td>
<td>1.58196E-01</td>
<td>2.90183E-01</td>
<td>2.28253E-02</td>
<td>8.80439E-01</td>
</tr>
</tbody>
</table>

Table 6.12: Material Properties for Takeda Benchmark Problem, Model 1, Reflector Region

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>$\Sigma_{tr,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{a,g}$ (cm$^{-1}$)</th>
<th>$\nu\Sigma_{\xi,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{n,1,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{tr,2,g}$ (cm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.50367E-01</td>
<td>4.16392E-04</td>
<td>0.00000E+00</td>
<td>1.93446E-01</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>2</td>
<td>1.64482E+00</td>
<td>2.02999E-02</td>
<td>0.00000E+00</td>
<td>5.65042E-02</td>
<td>1.62452E+00</td>
</tr>
</tbody>
</table>

Table 6.13: Material Properties for Takeda Benchmark Problem, Model 1a, Void Region

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>$\Sigma_{tr,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{a,g}$ (cm$^{-1}$)</th>
<th>$\nu\Sigma_{\xi,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{n,1,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{tr,2,g}$ (cm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.28407E-02</td>
<td>4.65132E-05</td>
<td>0.00000E+00</td>
<td>1.27700E-02</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>2</td>
<td>1.20676E-02</td>
<td>1.32890E-03</td>
<td>0.00000E+00</td>
<td>2.40997E-05</td>
<td>1.07387E-02</td>
</tr>
</tbody>
</table>

Table 6.14: Material Properties for Takeda Benchmark Problem, Model 1b, Control Rod Region

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>$\Sigma_{tr,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{a,g}$ (cm$^{-1}$)</th>
<th>$\nu\Sigma_{\xi,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{n,1,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{tr,2,g}$ (cm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.52325E-02</td>
<td>1.74439E-02</td>
<td>0.00000E+00</td>
<td>6.77241E-02</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>2</td>
<td>2.17460E-01</td>
<td>1.82224E-01</td>
<td>0.00000E+00</td>
<td>6.45461E-05</td>
<td>3.52358E-02</td>
</tr>
</tbody>
</table>

Although Takeda Models 1a and 1b simulate an operable test reactor, experimental data was not provided to evaluate the participants’ results. Instead, a reference solution was derived from results that were submitted as part of the original benchmark study.
Participants used a wide range of methods that included perturbed and unperturbed Monte Carlo methods, $P_N$ spherical harmonics methods, $S_N$ discrete ordinates methods, collision probability methods, and others. The reference solution was generated by averaging the results for all of the unperturbed Monte Carlo methods, yielding the so-called “exact Monte Carlo” solution.

Reference values are provided for two sets of parameters. First, $k_{eff}$ for Models 1a and 1b are given in Table 6.15 along with the reference control rod worth, which is defined to be

\[
Worth = \frac{1}{k_{eff}^{in}} - \frac{1}{k_{eff}^{out}}.
\]

(6.1)

| Table 6.15: Reference $k_{eff}$ for Takeda Benchmark Problem, Model 1 |
|------------------------|------------------------|------------------------|------------------------|
|                        | Model 1a               | Model 1b               | Rod Worth              |
|                        | (Rod Withdrawn)        | (Rod Inserted)         |                        |
| $k_{eff}$              | 0.9780                 | 0.9624                 | 1.66E-02               |
| Absolute Standard Deviation | 0.0006               | 0.0006                 | 9.00E-04               |
| %                      | 0.06                   | 0.06                   | 5                      |

Second, reference values are given in Tables 6.16 and 6.17 for the group-dependent region-averaged neutron scalar flux in each of the three material regions for both reactor configurations.
Table 6.16: Reference Region-Averaged Scalar Flux for Takeda Benchmark Problem, Model 1a

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>Core Region</th>
<th>Reflector Region</th>
<th>Void Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalar Flux</td>
<td>Standard Deviation</td>
<td>Scalar Flux</td>
</tr>
<tr>
<td>1</td>
<td>4.75E-03</td>
<td>0.10%</td>
<td>5.93E-04</td>
</tr>
<tr>
<td>2</td>
<td>8.70E-04</td>
<td>0.12%</td>
<td>9.14E-04</td>
</tr>
</tbody>
</table>

Table 6.17: Reference Region-Averaged Scalar Flux for Takeda Benchmark Problem, Model 1b

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>Core Region</th>
<th>Reflector Region</th>
<th>Control Rod Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalar Flux</td>
<td>Standard Deviation</td>
<td>Scalar Flux</td>
</tr>
<tr>
<td>1</td>
<td>4.91E-03</td>
<td>0.10%</td>
<td>5.91E-04</td>
</tr>
<tr>
<td>2</td>
<td>8.69E-04</td>
<td>0.13%</td>
<td>8.79E-04</td>
</tr>
</tbody>
</table>

Specifications for the Takeda benchmark recommend usage of the following input and computational parameters:

- The system model is discretized with a 1.0 cm cubic mesh.
- Maximum outer iteration convergence criterion is $5.0 \times 10^{-5}$ and maximum inner iteration convergence is $5.0 \times 10^{-4}$.
- Neutron scalar flux is normalized according to

$$ \sum_{g} \left[ \int dV \Sigma_{f}^{i,g} \phi^{i,g} \right] = 1 $$

(6.2)

- $S_{4}$ quadrature is recommended, although higher-order quadrature sets are also included.
The final MOCK-3D results, presented in Section 6.1.3.6, utilize the recommended cell dimensions, quadrature order, and flux normalization, while more stringent convergence criteria are implemented. First, a series of sensitivity studies are conducted to evaluate the effects of these and other parameters on the accuracy of the MOCK-3D solution.

6.1.3.2 Study Case A: Cell Size

Observations for the previous two test problems suggest that cell size will affect significantly the accuracy and efficiency of a MOCK-3D solution to Takeda Model 1. In the method of characteristics a fine-mesh discretization is typically employed, for which the cell size should be small relative to the transport mean free path, $\frac{1}{\Sigma_T}$; however, it is also desirable to make the cells as large as possible to minimize runtimes and memory requirements. To strike an optimal balance between accuracy and computational efficiency is a critical step during the process of model development for space-time reactor kinetics analysis.

In this detailed study of the effects of perturbations to the cell size on a MOCK-3D computation, seven cases are modeled with cell dimensions chosen for each case so that the 25.0 cm cubic system model will divide into an integral number of cells in all directions. The coarsest mesh is defined by a 5.0 cm cubic cell size, yielding a 5 x 5 x 5 system model, while the finest discretization utilizes a 0.5 cm cubic cell, yielding a 50 x 50 x 50 grid. Ideally all other parameters would be held constant for this study, yet some variation in track spacing is required due to the constraint imposed during 3D ray tracing.
that no characteristic pass through the corner of a cell or plane. To ensure that the effects observed in this study are due to perturbations in cell size and not due to perturbations in track separation, a second study will be conducted in Section 6.1.3.3 to isolate the effects of track separation. The specifications for each of the seven cases in the present study are provided in Table 6.18.

<table>
<thead>
<tr>
<th>Case</th>
<th>Cell Dimension (cm)</th>
<th># Cells in Row</th>
<th>Track Separation (cm)</th>
<th>Quadrature Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.000</td>
<td>5</td>
<td>0.30</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2.500</td>
<td>10</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1.667</td>
<td>15</td>
<td>0.33</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1.250</td>
<td>20</td>
<td>0.25</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1.000</td>
<td>25</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>0.625</td>
<td>40</td>
<td>0.13</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>0.500</td>
<td>50</td>
<td>0.10</td>
<td>4</td>
</tr>
</tbody>
</table>

For each of the seven cases specified in Table 6.18, two calculations are performed: the rod-out model, Model 1a, and the rod-in model, Model 1b. The converged value for $k_{\text{eff}}$ from each of these calculations is extracted and rod worth is calculated for all cell sizes according to Equation (6.1). These results are shown in Tables 6.19 through 6.21.
Table 6.19: MOCK-3D $k_{eff}$ Results for Takeda Cell Size Study, Cases A1 to A3

<table>
<thead>
<tr>
<th>Reference</th>
<th>Case A1</th>
<th>Case A2</th>
<th>Case A3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keff</td>
<td>% Error</td>
<td>Keff</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.8607</td>
<td>-12.0</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.8434</td>
<td>-12.4</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>2.38E-02</td>
<td>43.5</td>
</tr>
</tbody>
</table>

Table 6.20: MOCK-3D $k_{eff}$ Results for Takeda Cell Size Study, Cases A4 and A5

<table>
<thead>
<tr>
<th>Reference</th>
<th>Case A4</th>
<th>Case A5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keff</td>
<td>% Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.9732</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.9567</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>1.77E-02</td>
</tr>
</tbody>
</table>

Table 6.21: MOCK-3D $k_{eff}$ Results for Takeda Cell Size Study, Cases A6 and A7

<table>
<thead>
<tr>
<th>Reference</th>
<th>Case A6</th>
<th>Case A7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k_{eff}$</td>
<td>% Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.9775</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.9618</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>1.67E-02</td>
</tr>
</tbody>
</table>

To more easily observe the effects of cell size on $k_{eff}$ and rod worth, these results are plotted versus cell size in figures 6.10 and 6.11.
Figure 6.10: MOCK-3D $k_{\text{eff}}$ versus Cell Size for Takeda Models 1a and 1b

Figure 6.11: MOCK-3D Rod Worth versus Cell Size for Takeda Models 1a and 1b
As expected, perturbation of the cell size has a significant and observable effect on the quality of the MOCK-3D solution. In Case 1, which utilizes the largest cell size of 5.0 cm, $k_{\text{eff}}$ is under-predicted by over 10% and rod worth deviates from the reference value by more than 40%. This error is unacceptably large for any practical application; however, as cell size is decreased the results improve substantially and, as figure 6.10 demonstrates, the calculated value for $k_{\text{eff}}$ will approach the reference value asymptotically.

A linear correlation between rod worth and cell size is observed in figure 6.11. In fact, a close examination of this correlation appears to suggest that as cell size is made smaller than has been modeled here, rod worth error will become negative; i.e., the overshoot will become an undershoot. This result is caused by differences in the accuracy of the MOCK-3D solution for the rod-in and rod-out cases. Errors caused by the presence of a strong absorber in the rod-in case are much larger than the error from modeling of a void in the rod-out case, and the observed pattern of rod worth behavior is due primarily to a cancellation of errors. Such a pattern should be ameliorated as the accuracy of the transport solution via fine-mesh characteristics is improved.

It was mentioned above that in the method of characteristics cell dimensions are typically constrained to be smaller than the transport mean free path, $1/\Sigma_{\nu}$. From the material properties defined in Tables 6.11 through 6.14, the smallest value for $1/\Sigma_{\nu}$ is found to be ~0.6 cm, which occurs in the reflector region for energy group 2. This mean free path length suggests that Case 6 would be appropriate, where cell size is defined to be 0.625
Results for this case are reasonably accurate, with the errors on $k_{\text{eff}}$ and rod worth being less than 0.1% and 1% respectively, as seen in Table 6.21. If accuracy were the only concern, this would be a reasonable limit for the size of cells; however, computational efficiency must also be considered. To this end, the runtimes for each of these cases are listed in Table 6.22.

<table>
<thead>
<tr>
<th>Case</th>
<th>Rod In</th>
<th>Rod Out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Runtime (hh:mm:ss)</td>
<td># Outer Iterations</td>
</tr>
<tr>
<td>1</td>
<td>00:01:09</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>00:06:16</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>00:23:23</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>01:49:38</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>03:55:29</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>12:04:16</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>12:11:20</td>
<td>17</td>
</tr>
</tbody>
</table>

The runtimes presented in Table 6.22 are not intended to be a formal study of computational efficiency and no attempt has been made to accelerate the MOCK-3D package; yet these results clearly demonstrate the existence of a threshold, beyond which additional accuracy comes at the cost of extremely long runtimes. Case 5, with a cell size of 1.0 cm, strikes a reasonable balance between accuracy and expense. This is also the cell size recommended by the Takeda Benchmark specifications. Further, the accuracy of this case is not wholly unacceptable – results for $k_{\text{eff}}$ agree to within $\sim$0.5% of the...
reference solution and the error in rod worth of 4.1% is less than the standard deviation listed in Table 6.15. In fact, while the Case 5 cell size is nearly twice as large as $1/\Sigma_r$ in the reflector region, $1/\Sigma_r$ in the core region is ~0.95 cm, as determined from Table 6.11. This observation suggests that the 1.0 cm model may yield sufficient accuracy in the core but reduced accuracy in the reflector, an issue that will be examined in more detail below.

### 6.1.3.3 Study Case B: Track Separation

Track separation is another parameter that would be expected to affect the method of characteristics transport solution. In 3D Cartesian geometry track separation is represented by two parameters, the azimuthal track separation, $\delta A$, and the polar track separation, $\delta P$. In RayMonde equal values for $\delta A$ and $\delta P$ are specified by the user and these are modified during ray tracing to enforce periodicity. Six cases are defined for study of this parameter, as defined in Table 6.23.

<table>
<thead>
<tr>
<th>Case</th>
<th>Cell Dimension (cm)</th>
<th># Cells in Row</th>
<th>Track Separation (cm)</th>
<th>Quadrature Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>25</td>
<td>0.50</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1.000</td>
<td>25</td>
<td>0.37</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1.000</td>
<td>25</td>
<td>0.24</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1.000</td>
<td>25</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1.000</td>
<td>25</td>
<td>0.10</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1.000</td>
<td>25</td>
<td>0.05</td>
<td>4</td>
</tr>
</tbody>
</table>
The modified values for $\delta A$ and $\delta P$ have been extracted from the tracking file for each case and are listed in Tables 6.24 through 6.25.

### Table 6.24: RayMonde Modified Track Separations, Case B1

<table>
<thead>
<tr>
<th>Direction</th>
<th>$\delta A$</th>
<th>$\delta P$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Modified (cm)</td>
<td>% Change</td>
</tr>
<tr>
<td>1</td>
<td>0.35</td>
<td>-29</td>
</tr>
<tr>
<td>2</td>
<td>0.45</td>
<td>-10.6</td>
</tr>
<tr>
<td>3</td>
<td>0.45</td>
<td>-10.6</td>
</tr>
</tbody>
</table>

### Table 6.25: RayMonde Modified Track Separations, Case B2

<table>
<thead>
<tr>
<th>Direction</th>
<th>$\delta A$</th>
<th>$\delta P$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Modified (cm)</td>
<td>% Change</td>
</tr>
<tr>
<td>1</td>
<td>0.35</td>
<td>-4.4</td>
</tr>
<tr>
<td>2</td>
<td>0.28</td>
<td>-25</td>
</tr>
<tr>
<td>3</td>
<td>0.28</td>
<td>-25</td>
</tr>
</tbody>
</table>

### Table 6.26: RayMonde Modified Track Separations, Case B3

<table>
<thead>
<tr>
<th>Direction</th>
<th>$\delta A$</th>
<th>$\delta P$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Modified (cm)</td>
<td>% Change</td>
</tr>
<tr>
<td>1</td>
<td>0.24</td>
<td>-1.8</td>
</tr>
<tr>
<td>2</td>
<td>0.22</td>
<td>-6.8</td>
</tr>
<tr>
<td>3</td>
<td>0.22</td>
<td>-6.8</td>
</tr>
</tbody>
</table>
Table 6.27: RayMonde Modified Track Separations, Case B4

<table>
<thead>
<tr>
<th>Direction</th>
<th>$\delta A$</th>
<th></th>
<th>$\delta P$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(cm)</td>
<td>% Change</td>
<td>(cm)</td>
<td>% Change</td>
</tr>
<tr>
<td>1</td>
<td>0.177</td>
<td>-11.6</td>
<td>0.173</td>
<td>-13.6</td>
</tr>
<tr>
<td>2</td>
<td>0.186</td>
<td>-7.2</td>
<td>0.187</td>
<td>-6.3</td>
</tr>
<tr>
<td>3</td>
<td>0.186</td>
<td>-7.2</td>
<td>0.187</td>
<td>-6.3</td>
</tr>
</tbody>
</table>

Table 6.28: RayMonde Modified Track Separations, Case B5

<table>
<thead>
<tr>
<th>Direction</th>
<th>$\delta A$</th>
<th></th>
<th>$\delta P$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(cm)</td>
<td>% Change</td>
<td>(cm)</td>
<td>% Change</td>
</tr>
<tr>
<td>1</td>
<td>0.088</td>
<td>-11.6</td>
<td>0.096</td>
<td>-4.4</td>
</tr>
<tr>
<td>2</td>
<td>0.093</td>
<td>-7.2</td>
<td>0.094</td>
<td>-5.7</td>
</tr>
<tr>
<td>3</td>
<td>0.093</td>
<td>-7.2</td>
<td>0.094</td>
<td>-5.7</td>
</tr>
</tbody>
</table>

Table 6.29: RayMonde Modified Track Separations, Case B6

<table>
<thead>
<tr>
<th>Direction</th>
<th>$\delta A$</th>
<th></th>
<th>$\delta P$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(cm)</td>
<td>% Change</td>
<td>(cm)</td>
<td>% Change</td>
</tr>
<tr>
<td>1</td>
<td>0.047</td>
<td>-5.7</td>
<td>4.924E-02</td>
<td>-1.53</td>
</tr>
<tr>
<td>2</td>
<td>0.049</td>
<td>-3.0</td>
<td>4.936E-02</td>
<td>-1.28</td>
</tr>
<tr>
<td>3</td>
<td>0.049</td>
<td>-3.0</td>
<td>4.936E-02</td>
<td>-1.28</td>
</tr>
</tbody>
</table>

For each of these six cases in this study, calculations are performed for the rod-out and rod-in models. Values for $k_{\text{eff}}$ and rod worth are provided in Tables 6.30 and 6.31.
Table 6.30: MOCK-3D $k_{\text{eff}}$ Results for Takeda Track Separation Study, Cases B1 to B3

<table>
<thead>
<tr>
<th>Reference</th>
<th>Case B1</th>
<th>Case B2</th>
<th>Case B3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keff % Error</td>
<td>Keff % Error</td>
<td>Keff % Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.9765</td>
<td>-0.2</td>
</tr>
<tr>
<td></td>
<td>0.9763</td>
<td>-0.2</td>
<td>0.9763</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.9625</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.9590</td>
<td>-0.4</td>
<td>0.9599</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>1.50E-02</td>
<td>-9.9</td>
</tr>
<tr>
<td></td>
<td>1.85E-02</td>
<td>11.2</td>
<td>1.75E-02</td>
</tr>
</tbody>
</table>

Table 6.31: MOCK-3D $k_{\text{eff}}$ Results for Takeda Track Separation Study, Cases B4 to B6

<table>
<thead>
<tr>
<th>Reference</th>
<th>Case B4</th>
<th>Case B5</th>
<th>Case B6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keff % Error</td>
<td>Keff % Error</td>
<td>Keff % Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.9757</td>
<td>-0.2</td>
</tr>
<tr>
<td></td>
<td>0.9756</td>
<td>-0.2</td>
<td>0.9764</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.9596</td>
<td>-0.3</td>
</tr>
<tr>
<td></td>
<td>0.9598</td>
<td>-0.3</td>
<td>0.9606</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>1.73E-02</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td>1.68E-02</td>
<td>1.3</td>
<td>1.69E-02</td>
</tr>
</tbody>
</table>

Plots of $k_{\text{eff}}$ and rod worth versus initial track separation are shown in figures 6.12 and 6.13, respectively.

![Figure 6.12: MOCK-3D $k_{\text{eff}}$ versus Track Separation for Takeda Models 1a and 1b](image-url)
The effects of perturbations to the track spacing are not as clear as for cell size. Figure 6.12, for example, appears to show that refining the track spacing has a somewhat random effect on the accuracy of $k_{\text{eff}}$. For example, the rod-in calculation for Case 1, which utilizes the largest track separation, yields the most accurate $k_{\text{eff}}$; while the rod-in calculation for Case 2 yields the least accurate $k_{\text{eff}}$. Figure 6.13 shows, however, that refining the track spacing does improve the accuracy of the rod worth, an effect that is more clearly observed by comparing rod worth errors in Tables 6.30 and 6.31. For track separations of less than ~0.25 cm (1/4 of the cell size), rod worth error will lie within the standard deviation for the reference solution listed in Table 6.15. Further, when track separation is less than ~0.10 cm (1/10 of the cell size) this error is reduced to ~1%. 

![Figure 6.13: MOCK-3D Rod Worth versus Track Separation for Takeda Models 1a and 1b](image)
Some of the apparent randomness in figure 6.12 may result from the process of modification of the initial track separation. Initially, a single value for track separation is specified for both $\delta A$ and $\delta P$ and for all directions in the quadrature set, while the final track separations are direction-dependent and are determined in large part by the requirement that tracks connect directly across cell boundaries. In this problem, modified track separations deviate from the initial value by as much as 29%, for Case 1 direction 1 in Table 6.24, and as little as 1%, for Case 6 directions 2 and 3 in Table 6.34. Examination of Tables 6.24 through 6.39 yields two trends of particular note. First, the change in track separation imposed during the modification process is reduced as the initial track separation is decreased. Second, modification yields a smaller track separation for every instance in this particular problem. While the latter effect is beneficial to the accuracy of the solution presented above, this observation cannot be generalized to other problems that might be encountered.

6.1.3.4 Study Case C: Quadrature

A third geometric parameter that could affect the accuracy of a method of characteristics solution is the number of directions that are included in the quadrature set and it is expected that increasing the number of directions in the quadrature set would yield greater accuracy in the MOCK-3D results. Five orders of level symmetric quadrature are included in RayMonde: $S_4$, $S_6$, $S_8$, $S_{12}$, and $S_{16}$. All of these but $S_{16}$ are included as cases in this study, as defined in Table 6.32. The $S_{16}$ quadrature could not be included because of excessive virtual memory requirements.
Table 6.32: Problem Specifications for MOCK-3D Takeda Quadrature Study

<table>
<thead>
<tr>
<th>Case</th>
<th>Cell Dimension (cm)</th>
<th># Cells in Row</th>
<th>Track Separation (cm)</th>
<th>Quadrature Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>25</td>
<td>0.10</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1.000</td>
<td>25</td>
<td>0.10</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1.000</td>
<td>25</td>
<td>0.11</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1.000</td>
<td>25</td>
<td>0.10</td>
<td>12</td>
</tr>
</tbody>
</table>

For each of these four cases, calculations are performed for the rod-out and rod-in models. The resulting $k_{eff}$ and rod worth values for each calculation are provided in Tables 6.33 and 6.34 and plots of $k_{eff}$ and rod worth values versus quadrature order are shown in figures 6.14 and 6.15.

Table 6.33: MOCK-3D $k_{eff}$ Results for Takeda Quadrature Study, Cases C1 and C2

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>Case C1</th>
<th>Case C2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$k_{eff}$</td>
<td>% Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.9756</td>
<td>-0.2</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.9598</td>
<td>-0.3</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>1.68E-02</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 6.34: MOCK-3D $k_{eff}$ Results for Takeda Quadrature Study, Cases C3 and C4

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>Case C3</th>
<th>Case C4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$k_{eff}$</td>
<td>% Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.9757</td>
<td>-0.2</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.9597</td>
<td>-0.3</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>1.71E-02</td>
<td>3.1</td>
</tr>
</tbody>
</table>
Figure 6.14: MOCK-3D $k_{eff}$ versus Quadrature for Takeda Models 1a and 1b

Figure 6.15: MOCK-3D Rod Worth versus Quadrature for Takeda Models 1a and 1b
Figure 6.14 demonstrates that refinement to the quadrature has a minimal effect on $k_{eff}$, while figure 6.15 shows a markedly counterintuitive relationship between quadrature and rod worth. In fact, the results for rod worth yield no observable pattern of behavior and, for example, the smallest rod worth error is seen for the coarsest quadrature set – $S_4$, which has only 3 directions per octant. Two important points must be emphasized. First, all of the rod worth errors for the cases in this study, given in Tables 6.33 and 6.34, are less than 4%, which lies well within the standard deviation of the reference solution. The effects seen in figure 6.15 are, therefore, not as severe as they appear. Second, in light of the similar results that were seen in the track separation and cell size studies, it is clear that these three geometrical parameters cannot be easily separated. For example, the choice of an optimal quadrature may depend on the particular cell discretization; thus, $S_4$ may yield accurate results for one particular track set, but another track set could require $S_8$ or $S_{12}$. The specification of a problem model is a complicated process and further studies are required to fully investigate the coupling between cell size, track separation, and quadrature. Nonetheless, cell size is by far the most dominant parameter, and utilization of $S_4$ quadrature should be sufficient for the remainder of the analyses to be pursued here.

6.1.3.5 Study Case D: Inner Iteration Convergence Criterion

In the three previous sensitivity studies, parameters relating to the geometric description of the problem model were investigated. To investigate numerical effects, sensitivity to the tightness of convergence during the inner iteration is assessed. In MOCK-3D, the
convergence is sought by comparing the current and previous iteration neutron flux distributions according to Equation (5.11) or Equation (5.12). When the largest change in flux for all local points is found to be less than the specified tolerance criterion, the flux distribution is “converged” and the inner iteration is concluded. If this tolerance is specified too loosely, however, the problem will end with inaccurate estimates for the angular and neutron scalar flux distributions.

In MOCK-3D convergence of the inner iteration is assessed by testing both the angular and neutron scalar flux distributions. Typically the associated convergence criteria, $\varepsilon_a$ and $\varepsilon_s$, will be defined to be equal, which will cause changes to the neutron angular flux to dominate the evaluation of convergence. To assess these effects, the convergence criteria for angular and scalar flux are perturbed uniformly for the four cases listed in Table 6.35.

<table>
<thead>
<tr>
<th>Case</th>
<th>Inner Convergence Criterion</th>
<th>Cell Dimension (cm)</th>
<th># Cells in Row</th>
<th>Track Separation (cm)</th>
<th>Quadrature Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00E-05</td>
<td>1.000</td>
<td>25</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1.00E-06</td>
<td>1.000</td>
<td>25</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1.00E-07</td>
<td>1.000</td>
<td>25</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1.00E-08</td>
<td>1.000</td>
<td>25</td>
<td>0.20</td>
<td>4</td>
</tr>
</tbody>
</table>
In addition, it should be noted that all of the input parameters and primary computational variables, including user-defined convergence criteria and the arrays that hold the angular and scalar flux distributions, are defined to be double precision variables within MOCK-3D. Calculations are performed for the rod-out and rod-in models for each of the cases in this study. The resulting $k_{\text{eff}}$ and rod worth values for each calculation are provided in Tables 6.36 and 6.37 and plots of $k_{\text{eff}}$ and rod worth values versus convergence criterion are shown in figures 6.16 and 6.17.

Table 6.36: MOCK-3D $k_{\text{eff}}$ Results for Takeda Inner Iteration Study, Cases D1 and D2

<table>
<thead>
<tr>
<th>Reference</th>
<th>Case D1</th>
<th>Case D2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k_{\text{eff}}$</td>
<td>% Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>-0.2</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>-0.3</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Table 6.37: MOCK-3D $k_{\text{eff}}$ Results for Takeda Inner Iteration Study, Cases D3 and D4

<table>
<thead>
<tr>
<th>Reference</th>
<th>Case D3</th>
<th>Case D4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k_{\text{eff}}$</td>
<td>% Error</td>
</tr>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>-0.2</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>-0.3</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>4.1</td>
</tr>
</tbody>
</table>
Figure 6.16: MOCK-3D $k_{eff}$ versus Inner Convergence for Takeda Models 1a and 1b

Figure 6.17: MOCK-3D Rod Worth versus Inner Convergence for Takeda Models 1a and 1b
Results from this study clearly show that refining the scalar flux convergence criterion beyond $1.0 \times 10^{-6}$ does nothing to improve on the accuracy of $k_{\text{eff}}$ or rod worth. One speculation from the previous studies is refuted; i.e., that some of the error that results from perturbation of geometrical parameters is due to a failure to fully converge the flux distribution. It is possible that the flux distribution is converging to the wrong value; however, given the consistency in the values for $k_{\text{eff}}$ and rod worth in the previous examples, a more systematic mathematical review of convergence is not warranted.

6.1.3.6 Final Results and Analysis

Upon completing the sensitivity studies described in the previous sections and using those results as a guide, a set of model input parameters was generated for performing a final set of solutions to the Model 1 benchmark problem. Results that were published previously in Reference 95 are reproduced and expanded in this section. The most significant input parameters are listed in Table 6.38.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Dimension (cm)</td>
<td>1.0</td>
</tr>
<tr>
<td># Cells in Row</td>
<td>25</td>
</tr>
<tr>
<td>Track Separation (cm)</td>
<td>0.2</td>
</tr>
<tr>
<td>Quadrature Order</td>
<td>4</td>
</tr>
<tr>
<td>Outer Convergence Criteria</td>
<td>$1.0 \times 10^{-6}$</td>
</tr>
<tr>
<td>Inner Convergence Criteria</td>
<td>$1.0 \times 10^{-6}$</td>
</tr>
</tbody>
</table>
As in all cases, two calculations have been performed that correspond to the two states of
the Model 1 reactor: Model 1a with the control rod fully withdrawn, and Model 1b with
the rod fully inserted. For these “best-case” calculations, a more extensive set of results
is extracted that includes the region-averaged neutron scalar fluxes and additional
parameters for which a reference solution is not available. Values for $k_{eff}$ and rod worth
are shown in Table 6.39.

Table 6.39: Final MOCK-3D $k_{eff}$ and Rod Worth for Takeda Model 1

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>$k_{eff}$</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rod Out</td>
<td>0.9780</td>
<td>0.9757</td>
<td>-0.2</td>
</tr>
<tr>
<td>Rod In</td>
<td>0.9624</td>
<td>0.9596</td>
<td>-0.3</td>
</tr>
<tr>
<td>Rod Worth</td>
<td>1.66E-02</td>
<td>1.73E-02</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Tables 6.40 and 6.41 compare the region-averaged scalar flux values to the reference
solution for both models.

Table 6.40: Final MOCK-3D Region-Averaged Scalar Fluxes for Takeda Model 1a

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>Core Region</th>
<th>Reflector Region</th>
<th>Void Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalar Flux</td>
<td>% Error</td>
<td>Scalar Flux</td>
</tr>
<tr>
<td>1</td>
<td>4.72E-03</td>
<td>-0.57</td>
<td>6.00E-04</td>
</tr>
<tr>
<td>2</td>
<td>8.73E-04</td>
<td>0.35</td>
<td>8.28E-04</td>
</tr>
</tbody>
</table>
Table 6.41: Final MOCK-3D Region-Averaged Scalar Fluxes for Takeda Model 1b

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>Core Region</th>
<th>Reflector Region</th>
<th>Control Rod Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalar Flux</td>
<td>% Error</td>
<td>Scalar Flux</td>
</tr>
<tr>
<td>1</td>
<td>4.86E-03</td>
<td>-0.98</td>
<td>5.96E-04</td>
</tr>
<tr>
<td>2</td>
<td>8.69E-04</td>
<td>-0.07</td>
<td>7.92E-04</td>
</tr>
</tbody>
</table>

In Tables 6.42 and 6.43, the region-averaged absorption rate and region averaged production rate for each model are shown. These two parameters are among the “suggested” results in the original benchmark specifications; however, no reference solution is available.

Table 6.42: Final MOCK-3D Region-Averaged Absorption and Production Rates for Takeda Model 1a

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>Core Region</th>
<th>Reflector Region</th>
<th>Void Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Absorption Rate</td>
<td>Production Rate</td>
<td>Absorption Rate</td>
</tr>
<tr>
<td>1</td>
<td>4.03E-05</td>
<td>4.30E-05</td>
<td>2.50E-07</td>
</tr>
<tr>
<td>2</td>
<td>1.38E-04</td>
<td>2.53E-04</td>
<td>1.68E-05</td>
</tr>
</tbody>
</table>

Table 6.43: Final MOCK-3D Region-Averaged Absorption and Production Rates for Takeda Model 1b

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>Core Region</th>
<th>Reflector Region</th>
<th>Control Rod Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Absorption Rate</td>
<td>Production Rate</td>
<td>Absorption Rate</td>
</tr>
<tr>
<td>1</td>
<td>4.15E-05</td>
<td>4.42E-05</td>
<td>2.48E-07</td>
</tr>
<tr>
<td>2</td>
<td>1.37E-04</td>
<td>2.52E-04</td>
<td>1.61E-05</td>
</tr>
</tbody>
</table>
Finally, a series of plots has been developed that show the 2D scalar flux distribution for various cross-sectional planes of the Model 1 reactor. A complete set of plots would be too large for incorporation into this document; however, the Takeda benchmark specifications recommend several locations of particular interest that could be extracted. The following locations are extracted in the figures provided below:

- Figures 6.18 through 6.21 show 2D scalar flux distributions on the (x-y) plane that bisects the core axially; i.e., the z-axis plane of symmetry.
- Figures 6.22 through 6.25 show 2D scalar flux distributions on the (y-z) plane that is the x-axis plane of symmetry.
- Figures 6.26 through 6.29 show 2D scalar flux distributions on the (z-x) plane that is the y-axis plane of symmetry.

![Figure 6.18: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1a](image)
Figure 6.19: MOCK-3D Group 2 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1a

Figure 6.20: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1b
Figure 6.21: MOCK-3D Group 2 Normalized Scalar Flux Distribution along Z-Axis Plane of Symmetry for Takeda Model 1b

Figure 6.22: MOCK-3D Group 1 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1a
Figure 6.23: MOCK-3D Group 2 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1a

Figure 6.24: MOCK-3D Group 1 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1b
Figure 6.25: MOCK-3D Group 2 Normalized Scalar Flux Distribution along X-Axis Plane of Symmetry for Takeda Model 1b

Figure 6.26: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Y-Axis Plane of Symmetry for Takeda Model 1a
Figure 6.27: MOCK-3D Group 2 Normalized Scalar Flux Distribution along Y-Axis Plane of Symmetry for Takeda Model 1a

Figure 6.28: MOCK-3D Group 1 Normalized Scalar Flux Distribution along Y-Axis Plane of Symmetry for Takeda Model 1b
The cell size used in the final MOCK-3D models of the Takeda problem is larger than would be ideal and it is expected that this will yield noticeable error. The error associated with results for $k_{\text{eff}}$ and rod worth, for example, are slightly larger than is desirable, although not unreasonably large. More important is the marked difference in accuracy between Models 1a and 1b for the various scalar parameters. In Model 1a the control rod is fully withdrawn, so the control rod region will contain a void and streaming will be the dominant transport effect. In Model 1b the control rod is inserted, adding a strong absorber to the problem. As noted during discussion of the BWR assembly problem in Section 6.1.1, the MOCK-3D numerical methods may be failing in the presence of strong absorbers and such an effect is seen for this problem as well. The errors for the rod-in model are consistently larger than errors from the rod-out case for all scalar parameters.
Comparison of scalar parameters for the various problem regions yields more promising features of the MOCK-3D solution. In the core region, errors for the core-averaged scalar flux are in all cases less than 1%. While errors in the reflector are as large as 10%, the MOCK-3D calculation appears to capture most of the interactions between core and reflector to correctly estimate the flux in the power-producing region. Additionally, the 2D plots of scalar flux that are presented above demonstrate an excellent spatial resolution near the primary material interfaces in this problem, such as between the core and reflector or at the edges of the control rod region. Further reduction in cell sizes should yield an increase in accuracy for these features; however, the computational expense of such a problem would be prohibitive, as will be seen more clearly during assessment of the time-dependent component, MOCK-3DK.

### 6.2 Developmental Assessment of MOCK-3DK

Empirical analysis of the steady state method of characteristics demonstrates that accuracy of the transport solution will be quite sensitive to resolution of the 3D Cartesian spatial model; however, computational efficiency will degrade rapidly if cell sizes are defined to be too small. It follows that the time-dependent method of characteristics will experience a similar tradeoff between accuracy and computational efficiency in regards to definition of the spatial model. In MOCK-3DK, additional computational expense will be encountered due to the small time steps sizes that are required for the methods derived in Chapter 4. In the case of the fully-implicit method of characteristics, for example, a full fixed source iteration procedure must be performed at each time step. To minimize
the computational burden of the time-dependent calculations that are performed during assessment of MOCK-3DK, the restrictions on cell size described previously are loosened to allow for a definition of larger cells that yield less accurate results. Such a modeling strategy will focus the analysis of test results on the more critical time-dependent phenomena and minimize the computational expense of the required computations. As will be seen, however, even this allowance is insufficient to enable a full time-dependent calculation.

A survey of the literature reveals few time-dependent test problems that are suitable for the assessment of a time-dependent transport code such as MOCK-3DK. Standard test problems have been created for highly simplified systems in reduced spatial dimensions\textsuperscript{97}; these older problems were created as a means to assess 1D and 2D diffusion theory codes during their initial development. Such problems are of less utility to assessment of MOCK-3DK, however, as they include no means to evaluate a higher-order transport solution. Other possible test problems include 3D reactor kinetics benchmarks that model large LWR reactors and may include thermal-hydraulic feedback\textsuperscript{98,99}. These whole-core problems are designed for the assessment of “best estimate” reactor kinetics codes that will be applied to the analysis of commercial reactors and tightly coupled to a simultaneous thermal-hydraulics calculation, but such problems are too large and complicated to be of use in the present work.

One problem that was found to be suitable for inclusion in a MOCK-3DK test suite is the 2D seed/blanket problem developed for the purposes of validating the TWIGL diffusion
theory code. This problem has since become a commonly used test problem for time-dependent diffusion theory and neutron transport codes and is traditionally referred to as the “TWIGL Problem.” While this problem has not been standardized and a reference solution is not available, qualitative comparison could be made to previously validated results that are available in the literature.

6.2.1 2D Time-Dependent TWIGL Seed/Blanket Problem

The 2D TWIGL problem models a 160.0 cm square reactor consisting of three material regions: unperturbed seed regions that contain the primary fissile material, an identically composed perturbed seed region to which time-dependent properties will be introduced, and a blanket region that contains fissile material and surrounds the core on all sides. The 2D model is laid out in quarter-core symmetry as shown in figure 6.30.

![Figure 6.30: Problem Schematic for 2D TWIGL Seed/Blanket Problem](image)
In its initial state, the hypothetical TWIGL reactor is slightly sub-critical and the perturbed and unperturbed seed regions have identical material properties. The initial two-group properties for both materials are provided in Tables 6.44 and 6.45.

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>( \Sigma_{tr,g} ) (cm(^{-1}))</th>
<th>( \Sigma_{a,g} ) (cm(^{-1}))</th>
<th>( \nu \Sigma_{f,g} ) (cm(^{-1}))</th>
<th>( \Sigma_{n,1,g} ) (cm(^{-1}))</th>
<th>( \Sigma_{n,2,g} ) (cm(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.3810E-01</td>
<td>1.0000E-02</td>
<td>7.0000E-03</td>
<td>2.1810E-01</td>
<td>0.0000E+00</td>
</tr>
<tr>
<td>2</td>
<td>8.3333E-01</td>
<td>1.5000E-01</td>
<td>2.0000E-01</td>
<td>1.0000E-02</td>
<td>6.8333E-01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>( \Sigma_{tr,g} ) (cm(^{-1}))</th>
<th>( \Sigma_{a,g} ) (cm(^{-1}))</th>
<th>( \nu \Sigma_{f,g} ) (cm(^{-1}))</th>
<th>( \Sigma_{n,1,g} ) (cm(^{-1}))</th>
<th>( \Sigma_{n,2,g} ) (cm(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.5641E-01</td>
<td>8.0000E-03</td>
<td>3.0000E-03</td>
<td>2.3841E-01</td>
<td>0.0000E+00</td>
</tr>
<tr>
<td>2</td>
<td>6.6667E-01</td>
<td>5.0000E-02</td>
<td>6.0000E-02</td>
<td>1.0000E-02</td>
<td>6.1667E-01</td>
</tr>
</tbody>
</table>

Tables 6.46 and 6.47 provide the one delayed group constants and additional kinetics parameters that will be required to perform a time-dependent calculation.

<table>
<thead>
<tr>
<th>Energy Group</th>
<th>( v_g ) (m/sec)</th>
<th>( \chi_{\text{prompt}} ) (-)</th>
<th>( \chi_{\text{delay}} ) (-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000E+07</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
</tr>
<tr>
<td>2</td>
<td>2.0000E+05</td>
<td>0.0000E+00</td>
<td>0.0000E+00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delay Group</th>
<th>( \beta_i ) (-)</th>
<th>( \lambda_i ) (sec(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.5000E-03</td>
<td>8.0000E-02</td>
</tr>
</tbody>
</table>
A reference value for $k_{\text{eff}}$ is given to be 0.914193. The sub-critical equilibrium reactor state described by the values in Tables 6.44 through 6.47 can be made critical by dividing all of the fission cross sections, $\nu \Sigma_{f,g}$, by this reference value for $k_{\text{eff}}$. A delayed super-critical transient is initiated in the seed/blanket reactor by decreasing the thermal macroscopic transport cross-section, $\Sigma^2_{tr}$, in the perturbed seed region from the initial value of 0.83333 cm$^{-1}$ to a final value of 0.82983 cm$^{-1}$. Two separate transient calculations are performed. In the first case, the perturbation is introduced as a step change at $t = 0.0$ sec. In the second case, a ramp change is introduced over the time period $0.0$ sec < $t$ < 0.2 sec. In both cases, the suggested total transient time is 0.5 seconds.

While most of the previous analyses of this problem have utilized the 2D specifications listed above, a 3D version does exist$^{101}$. Material properties and time-dependent perturbations are identical to those of the 2D problem, while axial detail is introduced by extending each 2D region vertically. The final 160.0 cm cubic core model is surrounded on all sides by blanket material, as shown in figure 6.31.
The 3D TWIGL problem would be an ideal test problem for MOCK-3DK, as it probes the most important features of the time-dependent method of characteristics; however, the computational expense of this computation remains unattainable. Instead, the 2D problem is explored and focus is concentrated on the time-dependent discretization and results. This analysis will begin with a full steady state calculation, as described in Section 6.2.1.1. An alternative time-dependent perturbation will then be considered in Section 6.2.1.2 – a “null transient” that allows the core in its steady state configuration to be simulated as a transient with no perturbation to the material properties. Finally, results for preliminary calculations of the step and ramp transient problems will be assessed in Section 6.2.1.3.
6.2.1.1 2D TWIGL Steady State Calculation and Results

As described in Chapter 5, the MOCK-3D steady state module must be executed to generate the initial angular flux distribution for a MOCK-3DK time-dependent computation. This steady state calculation will also yield information that is useful to predictions of computational expense that guide selection of an appropriate spatial model. As was demonstrated during analysis of the Takeda problem, computational efficiency and accuracy of the transport solution are sensitive to the effects of cell size. This issue will be probed further via the series of steady state calculations with varying cell sizes listed in Table 6.48.

<table>
<thead>
<tr>
<th>Model</th>
<th>Cell Size (cm)</th>
<th>Lattice</th>
<th>Track Separation (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>160 x 160 x 160</td>
<td>0.10</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>80 x 80 x 80</td>
<td>0.20</td>
</tr>
<tr>
<td>4</td>
<td>2.0</td>
<td>40 x 40 x 40</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>4.0</td>
<td>20 x 20 x 20</td>
<td>0.80</td>
</tr>
<tr>
<td>6</td>
<td>8.0</td>
<td>10 x 10 x 10</td>
<td>1.60</td>
</tr>
</tbody>
</table>

An additional case, Model 1 with cell size of 0.25 cm, was also specified but could not be completed due to excessive memory and runtime requirements. Ray tracing for this problem is completed using a square cell perimeter and the track separation given in Table 6.48 where the $S_4$ level symmetric quadrature is applied. MOCK-3D solves this
problem by power iteration, with convergence criterion of $10^{-6}$ for $k_{\text{eff}}$ during the outer iteration and also for convergence of angular and neutron scalar flux distributions during the inner iteration. Results for these calculations will be condensed to a few significant scalar parameters and 2D maps of the neutron scalar flux distribution.

The first scalar to be monitored is the effective multiplication factor, $k_{\text{eff}}$, which is the eigenvalue of the steady state computation. Converged values for $k_{\text{eff}}$ are plotted in terms of cell size in figure 6.32, where the red line indicates the reference value for $k_{\text{eff}}$.

![Figure 6.32: MOCK-3D $k_{\text{eff}}$ versus Cell Size for 2D TWIGL Steady State Calculation](image-url)
Figure 6.32 shows a sensitivity to cell size that was also observed in MOCK-3D results for the Takeda problem. For increasingly fine discretizations, converged $k_{\text{eff}}$ approaches the reference value asymptotically; however, accuracy degrades significantly for cell sizes larger than 1.0 cm.

Having addressed accuracy through a perturbation of cell size, two parameters are evaluated as a means to assess efficiency of the MOCK-3D steady state TWIGL computation. The number of outers required to reach problem convergence, plotted in figure 6.33, is informative to predictions of problem size in the time-dependent case. The actual CPU time required for problem convergence is displayed in figure 6.34.

![Figure 6.33: Total Number of Outer Iterations Required for Convergence of MOCK-3D Calculation for Various Cell Sizes in 2D TWIGL Steady State Calculation](image-url)
The trade-off between optimal cell size and computational efficiency that was highlighted during the assessments of Section 6.1 is clearly demonstrated in the MOCK-3D solution to the steady state TWIGL problem. Accuracy of the converged eigenvalue, $k_{\text{eff}}$, degrades rapidly once cell size is increased beyond one or two transport mean free paths, while computational expense increases dramatically as cell size is decreased. As in the Takeda problem, a cell size of 1.0 cm cubic appears to strike the most equitable balance between accuracy and efficiency. The converged steady state neutron scalar flux distributions for this core geometry are shown in figures 6.35 and 6.36.
Figure 6.35: MOCK-3D Group 1 Converged Steady State Scalar Flux Distribution for 2D TWIGL Problem with 1.0 cm Cell Size

Figure 6.36: MOCK-3D Group 2 Converged Steady State Scalar Flux Distribution for 2D TWIGL Problem with 1.0 cm Cell Size
While optimal for steady state computation, a time-dependent calculation that uses the Model 3 spatial description would be too computationally expensive to be of practical use. In the transient results that are pursued below, the coarse cell description of Model 6 is used, yielding the converged scalar flux distributions shown in figures 6.37 and 6.38.

Figure 6.37: MOCK-3D Group 1 Converged Steady State Scalar Flux Distribution for 2D TWIGL Problem with 8.0 cm Cell Size
Though the geometry of Model 6 is much too coarse to be of practical use with the method of characteristics, the scalar flux shapes in figure 6.37 and figure 6.38 capture all of the significant spatial phenomena that are seen in the more refined distributions of figure 6.35 and figure 6.36. Inaccuracy due to the coarse spatial resolution is more pronounced in relation to values for $k_{\text{eff}}$. The Model 6 converged $k_{\text{eff}}$ value is only 0.8779, which is almost 4% lower than the reference value.

6.2.1.2 Results for the 2D TWIGL Null-Transient Problem

Although the TWIGL problem specifications call for a step or ramp perturbation to the material properties, initial assessment of MOCK-3DK is pursued via the specification of a null transient problem. In the null transient all time-dependent material properties are
assumed to be identical to those used during the steady state calculation. If the steady state reactor configuration is critical, therefore, the time-dependent core power is expected to remain constant during the null transient. Similarly, a sub-critical reactor should show a loss in power over time. Analysis of such a simple problem will be instrumental in determining whether or not the fundamental methods of MOCK-3DK are correctly defined and implemented.

In Chapter 4 three different formulations for the backward-differenced method of characteristics were derived. The first two formulations, derived from the constant and linear angular flux approximations, will be applied to the TWIGL null transient problem. The third formulation is prone to numerical instabilities that result from the exponential flux approximation and will be omitted from this initial study. Two different material property sets will be used in performing each of these calculations. In the first set, the steady state material properties will be used, so that the transient reactor will be sub-critical. An initially critical material state is generated by dividing all of the macroscopic fission cross sections, \( \nu \Sigma_{f,g} \), by the reference value for \( k_{\text{eff}} \). Specifications or these four base null transient cases are outlined in Table 6.49.

<table>
<thead>
<tr>
<th>Case</th>
<th>Angular Flux Approximation</th>
<th>Material State</th>
<th>Cell Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Constant</td>
<td>Sub-Critical</td>
<td>Model 6</td>
</tr>
<tr>
<td>Case 2</td>
<td>Constant</td>
<td>Critical</td>
<td>Model 6</td>
</tr>
<tr>
<td>Case 3</td>
<td>Linear</td>
<td>Sub-Critical</td>
<td>Model 6</td>
</tr>
<tr>
<td>Case 4</td>
<td>Linear</td>
<td>Critical</td>
<td>Model 6</td>
</tr>
</tbody>
</table>
One of the most significant numerical issues addressed in Chapter 5 regards the selection of a time step size for the fully-implicit method of characteristics. It has been shown that time step sizes must be reasonably small to achieve desired accuracy in modeling time dependent phenomena, typically no larger than $10^{-3}$ seconds. In the case of the method of characteristics, time step sizes must also be larger than a minimum threshold value to avoid numerical instabilities that result from improper modeling of the physics of time-dependent neutron transport. The value of this minimum threshold will depend strongly on the material properties, and in the case of the TWIGL problem is approximately $10^{-6}$ seconds. In total, six different time step sizes were used for each of the cases listed in Table 6.49. Not all of the 24 total calculations in this problem set were found to be numerically stable, as indicated in Table 6.50.

Table 6.50: Stability of MOCK-3D Computations during 2D TWIGL Null Transient Study

<table>
<thead>
<tr>
<th>$\Delta t$ (sec)</th>
<th>Constant Flux Approximation</th>
<th>Linear Flux Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Critical</td>
<td>Sub-Critical</td>
</tr>
<tr>
<td>$10^{-1}$</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>Unstable</td>
<td>Stable</td>
</tr>
<tr>
<td>$10^{-7}$</td>
<td>Unstable</td>
<td>Unstable</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>Unstable</td>
<td>Unstable</td>
</tr>
</tbody>
</table>

For each of the stable computations in Table 6.50, a MOCK-3DK calculation of 200 time steps has been completed. Temporal changes to the core model are monitored via the total core power, which is shown for each of the stable null transient calculations in figures 6.39 through 6.42.
Figure 6.39: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 1

Figure 6.40: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 2
Figure 6.41: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 3

Figure 6.42: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Null Transient Case 4
Another parameter that is of numerical interest is the fixed source eigenvalue, $\gamma$, which controls convergence of the outer fixed source iteration during each time step. A plot of $\gamma$ for the first 200 time steps of null transient case two is shown in figure 6.43.

![Figure 6.43: MOCK-3D Converged Eigenvalue for 200 Time Steps of 2D TWIGL Null Transient Case 2](image)

A comparison of figure 6.39 and figure 6.43 shows that the temporal behavior of the total core power is identical to that of the fixed source eigenvalue. A similar result was seen for all of the MOCK-3DK null transient calculations.
Null transient calculations could not be performed for problem durations longer than the limited 200 time step computations considered here. One possible source for this computational expense is seen in estimates of the CPU time required per time step, as shown in figure 44 through figure 47.

Figure 6.44: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Null Transient Case 1
Figure 6.45: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Null Transient Case 2

Figure 6.46: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Null Transient Case 3
As the recommended duration of the 2D TWIGL problem is 5.0 seconds, no MOCK-3DK computation can be expected to yield results for a realistic portion of that time. All of the power traces in figures 6.39 through 6.42 show a linear trend, however, and these limited results can be extrapolated to an extended problem time. In figure 6.48, core power is extrapolated to one second of total problem time and shown for a few selected time-dependent cases.
Figure 6.48: MOCK-3D Extrapolated Power versus Time for 1.0 sec of Problem Time in 2D TWIGL Problem

An assessment of spatial resolution of the MOCK-3DK results is pursued via a comparison of the neutron scalar flux distribution at the beginning of the null transient calculation to that at the end of the first 200 time steps. As an example, transient Case 2 will again be considered. The initial flux distribution is given by the converged steady state results, figure 6.37 and figure 6.38, while the final distribution is illustrated in figure 6.49 and figure 6.50.
Figure 6.49: MOCK-3D Group 1 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Null Transient Case 2

Figure 6.50: MOCK-3D Group 2 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Null Transient Case 2
Qualitatively, figures 6.49 and 6.50 show the same flux shape features as figures 6.37 and 6.38. A more quantitative assessment is achieved by computing the change in flux, $\Gamma$, at each computational point according to

$$\Gamma = 100 \cdot \frac{\phi_i^{i,g,\text{NULL}} - \phi_i^{i,g,\text{SS}}}{\phi_i^{i,g,\text{SS}}}$$  \hspace{1cm} (6.3)$$

Plots of $\Gamma$ for transient case 1 with a time step size of $5 \times 10^{-5}$ seconds are shown in figure 6.51 and figure 6.52.

![Figure 6.51: Percent Difference between MOCK-3D Group 1 Scalar Flux Values at $t = 0$ and after 200 Time Steps of 2D TWIGL Null Transient Case 2](image-url)
A non-physical numerical effect occurs during the first few time steps of each null transient calculation. In the power trends of figures 6.39 through 6.42, for example, some numerical oscillation in the total core power occurs during the first few time steps before settling to a mean constant value or, in a few cases, a linear trend. This phenomenon indicates that the MOCK-3DK transient calculation is not being properly initialized; however, the source of this problem is not clear and does not necessarily result from poor programming. For a TWIGL model with 8.0 cm cubic cells converged steady state $k_{eff}$ is underpredicted by ~4%, which implies a significant difference between the initial reactor state that is required to begin a time-dependent computation and the estimated initial state that results from a steady state calculation.
One explanation for the numerical effects that are observed at $t = 0$ is that the act of modifying the initial reactor state to be critical also introduces an effective step change in reactivity at $t = 0$. The amplitude of this reactivity change will be determined by the difference between the two initial reactor states, and thus proportional to the coarseness of the spatial model. If cell sizes are sufficiently small this reactivity change will vanish. Plots of CPU time during the first few time steps of each transient case support this analysis. In all cases, more CPU time is required during the first few time steps and convergence of the outer iteration is rapid for later time steps. CPU time also increases as the level of sophistication in the problem model is improved. Such numerical behavior is typically indicative of an improperly converged steady state solution, yet is in fact due to errors that result from a coarse spatial model.

Although a MOCK-3DK computation of 200 time steps is insufficient to generate a realistic set of results for the TWIGL problem, a few notable patterns can be observed in the previous figures. MOCK-3DK computations for those cases that model a critical reactor core yield power traces that are in most cases constant, as would be expected. The precise value for this mean power level is seen to be extremely sensitive to the type of calculation performed. A comparison of power trends for Cases 2 and 4, both of which model a critical core but use different flux approximations, shows that mean power level will decrease if the order of approximation for the neutron angular flux is increased. Similarly, mean power level decreases in all cases when time step size is also decreased. Unfortunately, inaccuracies due to the coarse spatial model are large enough to obstruct a
more through analysis of the performance and accuracy of alternative implicit solutions to the time-dependent method of characteristics.

6.2.1.3 Results for the 2D TWIGL Step and Ramp Transient Problems

Given the mixed results that were obtained for the TWIGL null transient computations, it is expected that an attempt to perform the step or ramp material perturbations defined in Section 6.2.1.1 will provide little that is of quantitative value. Nonetheless, such results could yield qualitative observations regarding behavior of the time-dependent method of characteristics. Step and ramp transient calculations are performed using an input model that is equivalent to null transient Case 2 from Table 6.49 and computations are performed in each case using three different time step sizes: $5 \times 10^{-3}$ sec, $5 \times 10^{-4}$ sec, and $5 \times 10^{-5}$ sec. The duration of all calculations is again limited to 200 time steps, yielding the normalized power traces shown in figure 6.53 for the step transient and figure 6.54 for the ramp transient.
Figure 6.53: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Step Transient

Figure 6.54: MOCK-3D Normalized Power for 200 Time Steps of 2D TWIGL Ramp Transient
The CPU time required to complete each time step for the first few time steps in each case are shown in figures 6.55 and 6.56.

Figure 6.55: MOCK-3D CPU Time for 10 Time Steps of 2D TWIGL Step Transient
Normalized power in the step transient is seen in figure 6.53 to follow a linear trend, as was observed in the null transient calculations. By extrapolating these results to 0.1 seconds in all cases, the power changes shown in figure 6.57 can be produced. In the ramp transient power does not follow a linear trend and a suitable trendline could not be generated.
For the step transient calculation that used time step sizes of $5 \times 10^{-4}$ seconds, the neutron scalar flux distribution was extracted after 200 time steps had elapsed. These plots are shown in figures 6.58 and 6.59.
Figure 6.58: MOCK-3D Group 1 Scalar Flux after 200 Time Steps of 2D TWIGL Step Transient

Figure 6.59: MOCK-3D Group 2 Scalar Flux after 200 Time Steps of 2D TWIGL Step Transient
As with the null transient calculations, the relative change in scalar flux shape is evaluated via Equation (6.3). The results for the step transient with time steps of $5 \times 10^{-4}$ seconds are given in figures 6.60 and 6.61.

Figure 6.60: Percent Difference between MOCK-3D Group 1 Scalar Flux Values at $t = 0$ and after 200 Time Steps of 2D TWIGL Step Transient
Neutron scalar flux distributions after 200 time steps of the ramp transient with time steps of \(5 \times 10^{-4}\) seconds are given in figures 6.62 and 6.63, while the associated flux shape change plots are shown in figures 6.64 and 6.65.
Figure 6.62: MOCK-3D Group 1 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Ramp Transient

Figure 6.63: MOCK-3D Group 2 Scalar Flux Distribution after 200 Time Steps of 2D TWIGL Ramp Transient
Figure 6.64: Percent Difference between MOCK-3D Group 1 Scalar Flux Values at $t = 0$ and after 200 Time Steps of 2D TWIGL Ramp Transient

Figure 6.65: Percent Difference between MOCK-3D Group 2 Scalar Flux Values at $t = 0$ and after 200 Time Steps of 2D TWIGL Ramp Transient
Results for the MOCK-3DK step transient calculations exhibit the same phenomena as were observed in the null transient cases. In figure 6.53, for example, the amplitude of the normalized power is seen to rise in response to the reactivity insertion at $t = 0$, and the rate of this power change is directly related to the size of the time step, as would be expected. Numerical oscillations are again observed during the first few time steps; in this context, however, this oscillation appears to be the natural result of an instantaneous change in reactivity. For the ramp transient cases shown in figure 6.54 little or no initial oscillation is observed in response to the infinitesimally small change in reactivity introduced at each time step. Given these observations, the hypothesis presented in Section 6.2.1.2 that initialization of the MOCK-3DK time-dependent computation is introducing a step change in reactivity to the null transient problem appears to be correct.

One significant difference between results for the null transient cases presented in Section 6.2.1.2 and the step and ramp transient results shown above regards the change in flux shape from $t = 0$ until 200 time steps have elapsed. In all of the null transient cases, the flux shape was seen to change very little, as would be expected for a problem that does not include material perturbation. In the step and ramp transient cases, however, a material perturbation is introduced and a noticeable change in flux shape is seen at the location of this perturbation in figures 6.60 and 6.61 and figures 6.64 and 6.65. That MOCK-3DK is correctly capturing this dominant physical effect, in spite of the extremely low spatial resolution, demonstrates a minimal successful functionality for the time-dependent characteristic methods that have been developed in this work.
CHAPTER 7:  
CONCLUSION

Research that has been described in the previous chapters introduces several notable advances to the state-of-the-art in method of characteristics techniques. Demonstration of a 3D ray tracing procedure for explicit 3D Cartesian geometry and its implementation into a 3D steady state method of characteristics calculation were pre-requisite tasks to the development of a time-dependent method of characteristics, yet the result is a more efficient computation than has been possible in commercial lattice physics applications. This efficiency is achieved by requiring that all cells in the fine-mesh 3D Cartesian grid be uniform and homogeneous. As a result the set of tracks for an entire system model can be produced via repetitive application of the track set for a single representative cell and the memory requirements associated with storage of the tracking file become trivial.

Assembly transport calculations utilize a complicated description of the problem space that may include internal cell details, thus the assumption that cells are uniform and homogeneous would be over-restrictive in such applications. This simplification of the problem space is more appropriate to a space-time reactor kinetics calculation, especially where a fine-mesh Cartesian grid is employed. Nonetheless, there may arise circumstances for which the cell constraints should be loosened. One means to generalize the spatial model is to allow for multiple material regions, with the track set for one cell from each region stored to file. Additional refinements to the spatial
resolution of a 3D method of characteristics computation are viable but must be preceded by improvements in computational efficiency, as will be discussed thoroughly below.

While the development of an efficient 3D steady state method of characteristics is a significant accomplishment, the investigation of a time-dependent formulation has been the primary goal of this work. As no previous investigation into a time-dependent method of characteristics has been reported, the scope of these initial developments has been limited to methods that are well understood from previous application to alternative time-dependent transport methods, including diffusion theory. Two of the most common time-dependent methods – the forward and backward time differencing approximations – were determined to be most appropriate for initial investigation. Forward differencing yields a fully explicit solution to the time-dependent characteristic equation and does not require spatial integration as in the steady state method of characteristics; however, this approach is susceptible to numerical instabilities that require extremely small time step sizes. A more stable implicit solution to the method of characteristics is derived by applying the backwards time-differencing approximation to the characteristic equation.

An a priori analysis of the fully-explicit method of characteristics shows that this approach will be too computationally expensive to be of practical use. Specifically, explicit methods exhibit numerical instabilities that can be resolved only by the specification of extremely small time step sizes. The fully implicit formulation, on the other hand, has been implemented into the MOCK-3DK reactor kinetics code and applied with limited success to the 2D time-dependent TWIGL seed/blanket test problem. Due to
exorbitant computational expense that results from these calculations, however, only a problem with coarse spatial meshing and few time steps can be successfully completed. In spite of these limitations, results for the null, step, and ramp transient versions of the TWIGL problem demonstrate that the fully-implicit time-dependent method of characteristics is correctly capturing the fundamental physics of this problem. Total core power, for example, is seen to be constant for null transient analyses of a critical core, while step and ramp transients yield an increase in power that corresponds correctly to material perturbation. Plots over 2D sections of the neutron scalar flux distribution show further that the spatial effects of this perturbation are appropriately localized in the perturbed seed region, where the reactivity insertion is defined to occur.

One irrefutable conclusion of this investigation is that a significant reduction in computational expense must be achieved before research can continue into a time-dependent method of characteristics. Of particular significance are the long runtimes that were required to complete even small and highly simplified calculations for analysis of the TWIGL problem. In a serial computation procedure such as is currently incorporated into MOCK-3D, the factor most directly related to runtime is the number of times that the steady state or time-dependent characteristic equation must be solved during a full computation. This parameter will be determined by the number of computed discrete variables in the full system model as well as the number of inner and outer iterations required to complete fixed source numerical solution. A simple expression for the number of serial processes is thus given by


\[ \text{\# processes} = T \cdot G \cdot M \cdot X \cdot Y \cdot Z \cdot N \cdot P \cdot J_{\text{in}} \cdot J_{\text{out}}, \]  

(7.1) 

where \( T \) is the number of time steps in a time-dependent problem, \( G \) is the number of energy groups, \( M \) is the number of directions in the quadrature set, \( X, Y, \) and \( Z \) represent the numbers of cells in the x-, y-, and z-directions, and the term \( N \cdot P \) represents the number of tracks in the 3D cell. The parameters \( J_{\text{in}} \) and \( J_{\text{out}} \) represent an estimated number of inner or outer iterations required to converge fixed source iterations. 

Most of the terms in Equation (7.1) relate to specifications of a model for the problem geometry; i.e., cell size determines values for \( X, Y, \) and \( Z, \) track separation affects \( N \) and \( P, \) and quadrature order yields a value for \( M. \) Analysis of MOCK-3D results for the Takeda problem in Chapter 6 has demonstrated that the primary tradeoff between accuracy and efficiency is encountered with definition of these spatial parameters. Accordingly, one of the most direct means to increase the efficiency of a MOCK-3D computation would be to implement techniques that achieve greater accuracy for a coarse spatial discretization. In fact, a similar approach yielded one of the most significant advancements to time-dependent diffusion theory via the application of nodal methods. Nodal methods have also been applied with success to finite-differenced \( S_N \) methods.

Application of nodal methods to the method of characteristics spatial discretization scheme is more complicated than for approaches that use spatial finite differencing; for example, nodal methods are traditionally accompanied by node boundary coupling conditions that are defined in terms of the neutron currents. If a similar definition were
applied directly to the method of characteristics, the direct propagation of neutrons across cell boundaries would be lost. Instead, the definition of a nodal method for the method of characteristics would need to be integrated directly into ray tracing and might require a fundamental redefinition of the associated procedures. Higher-order approximations to account for spatial-dependence of the neutron source term could also be examined as a means to increase the accuracy of this term for larger cell sizes.

Nodal methods or similar refinements to the spatial model in a time-dependent method of characteristics computation promise large potential gains in computational efficiency, but their development will require substantial additional research. A more direct route to acceleration of a MOCK-3DK calculation is to investigate one or more of the available numerical acceleration schemes that are reviewed in Chapter 2, many of which are well understood in application to the steady state method of characteristics. Incorporation of a coarse mesh rebalance procedure to accelerate the inner iteration in quasi-steady fixed source iteration and/or the addition of a flux extrapolation method to the outer iteration provide ideal starting points for such a line of future study. Recent investigations into the feasibility of an explicit 3D steady state method of characteristics for lattice physics applications have also shown that parallel processing greatly enhances the efficiency of such computations and may be prerequisite to their practical implementation.

Transport methods that utilize spatial finite differencing typically utilize angular and/or cell-wise decomposition of the problem space to facilitate a parallel computation. If angular decomposition is sought, parallel computation will be performed using a separate
processor for each direction in the quadrature set. Cell-wise decomposition, on the other hand, involves the separation of cells via a simple red-black checkerboard pattern (for low order approximations to the transport equation) or blocks of cells (for higher order approximations). In both cases, the degree of parallelization is limited to a relatively small number of processors; angular decomposition is typically employed on 16 or 24 processors, for example. One of the most significant advantages to a Lagrangian transport solution over Eulerian formulations is that propagation methods are more amenable to massively parallel computation. Reference 72, for example, demonstrates the feasibility of a track-wise decomposition whereby tracks are collected into bundles and each bundle is assigned to a separate processor. As the number of such bundles in a typical method of characteristics computation will be effectively limitless, an arbitrarily large number of processors could be employed for significant gains in speed-up.

As the above discussion illustrates, the problem of extreme computation expense is critically important to a method of characteristics computation and must be resolved before further assessment can be pursued. This conclusion is not surprising, however, and is not the primary focus of this work. A more fundamental analysis of the methods included in MOCK-3DK requires a critical assessment of key decisions that are encountered during the derivation, which can be pursued in spite of the lack of useful qualitative empirical results. Of particular importance are two levels of approximation that are required for solution of the time-differenced characteristic equation. In the steady state derivation, solution of the characteristic equation is accomplished via integration over the one-dimensional spatial variable that corresponds to the path of
neutron streaming. To resolve this integral, an approximation for the path-dependent total neutron source term must be introduced. When the same spatial integration procedure is applied to the time-dependent characteristic equation, two path-dependent quantities impede solution: the total neutron source term and an occurrence of the neutron angular flux within an integral (in backward time-differencing) or derivative (in forward time-differencing). In the first case the constant neutron source term approximation has been imposed. This approach is consistent with a fine-mesh homogeneous cell description and has been the norm in most lattice physics applications. As has been discussed above, a higher-order source term approximation could be advantageous as a means to increase the minimum cell size.

The neutron angular flux approximation that is required to solve the time-dependent problem has no analogue in any previously investigated transport method and three low-order approximations were pursued in this initial work – first that angular flux is constant along each track, second that angular flux is a linear function of position along the track, or third that angular flux can be expressed as an exponential function. The constant and linear formulations were implemented into MOCK-3DK, while the exponential approximation was discarded due to numerical problems that are unique to this formulation. Unfortunately, the results that could be generated for the TWIGL problem were insufficient to permit a detailed comparison of these approaches. Specifically, greater spatial resolution is required so that changes in the amplitude of the total core power will be more accurately represented.
During the analysis of numerical methods that are utilized to solve the fully-implicit
time-dependent characteristic equation, a particular instability was noted that is unique to
a Lagrangian formulation for the time-dependent transport equation. When finite-
differencing of the temporal domain is subjected to the method of characteristics spatial
integration procedure, the resulting implicit solution will be unconditionally unstable for
all time steps sizes less than a threshold value that is determined by the time required for
a low-energy neutron to propagate along the longest track in the problem set. In practice,
this threshold is found to be \( \sim 10^{-6} \) seconds, which is large enough to forbid the modeling
of certain rapid transient phenomena that are important to space-time reactor kinetics.
Ironically, the Stiffness Confinement Method, which is typically of interest because it
permits larger time step sizes, provides a means to make time step sizes smaller in the
case of the method of characteristics. By defining time-dependent neutron transport in
terms of the dynamic frequencies described in Chapter 4, SCM would yield a
characteristic equation that exhibits none of the singularities that occur for implicit
differencing. Investigation of this and other time dependent methods should therefore be
included in future work.

Presuming that the various improvements noted above are implemented into MOCK-3D,
further assessment via application to a suite of time-dependent test problems will be
required; however, there are few standard test problems or benchmarks that are suitable
to assessment of a time-dependent method of characteristics. Investigations into time-
dependent formulations for all of the higher-order transport methods that are reviewed in
Chapter 1 necessitate the development of reactor kinetics benchmark problems that
include transport effects. Interesting problems could be developed from the specific problems that motivate the search for more accurate methods; for example, problems could be formulated to describe a time-dependent material perturbation in a simplified MOX-loaded LWR core or the streaming of neutrons to ex-core detectors during the TMI accident. A useful time-dependent transport problem could also be created via modification of an existing steady state benchmark problem. A time-dependent version of the Takeda problem, for example, could be created by modeling a rod insertion or withdrawal to transit between the two states modeled in the steady state problem.

Practical application of the time-dependent method of characteristics will eventually require its incorporation into a larger suite of computational tools that comprise a space-time reactor kinetics package. Specifically, feedback to and from a thermal-hydraulics computation must be added and a suitable tool for the generation of 3D material properties must be created. In the latter case, this tool will inevitably be 3D assembly transport methods that are currently under development and that have provided the basis for the time-dependent transport methods derived in this work. The development of a reactor kinetics methodology that uses the same transport method to generate the 3D material properties and to perform the time-dependent transport calculation could facilitate the implementation of on-line cross section generation, whereby the assembly transport calculation(s), reactor kinetics calculation, and thermal-hydraulics calculation are all performed simultaneously. While the prospects for such a computational tool lie far in the future, further investigation of a time-dependent method of characteristics towards practical implementation is warranted.
REFERENCES


11 J. S. TULENKO, G. SCHOESSOW, and J. SCAGLIONE. “New Fuel Rod Design for Ultra High
(1997).

12 P. PETKOV and T. TAKEDA. “Transport Calculations of MOX and UO$_2$ Pin Cells by the Method of

13 B.-C. NA and N. MESSAOUDI. “Blind Benchmark on the 3-D VENUS-2 MOX Core Measurements.”
NEA/SEN/NSC/WPRR(2001)1, Organization for Economic Cooperation and Development/Nuclear Energy

14 B.-C. NA and E. SARTORI. “Organization for Economic Cooperation and Development/Nuclear
Energy Agency International Benchmark on the VENUS-2 MOX Core Measurements.” *Nucl. Sci. Eng.*, 

15 Y. H. KIM and N. Z. CHO. “A Bilinear Weigh ting Method for the Control Rod Cusping Problem in

16 J. B. YASINSKY. “On the Use of Point Kinetics for the Analysis of Rod Ejection Accidents.” *Nucl.

17 D. S. KIM and N. Z. CHO. “The Analytic Function Expansion Nodal (AFEN) Method with Half-
Interface Averaged Fluxes in Mixed Geometry Nodes for Analysis of Pebble Bed Modular Reactor

ANS/Franklin Institutes, February 10-11, 1960, pp.87-104 (1960).

19 B. TYOBEKA, K. IVANOV, and A. PAUTZ. “Utilization of Two-Dimensional Deterministic Transport


75 D. KNOTT, personal communication (2002).


93 R. SANCHEZ, personal communication (2002).


APPENDIX A

ADDITIONAL PROCEDURES FOR
THE PERFORMANCE OF 3D RAY TRACING

A1: Procedures to Subdivide Tracks of Type 1
A2: Procedures to Subdivide Tracks of Type 2
A3: Procedures to Subdivide Tracks of Type 3
A4: Procedures to Subdivide Tracks of Type 4
A5: Algorithm to Locate Entry Plane for Global Track
A6: Algorithm to Locate Exit Plane for Global Track
A7: Algorithm to Determine Number of Planes Crossed by Global Track
A.1 Procedures to Subdivide Tracks of Type 1

The procedures for subdividing global tracks of Type 1, which enter the polar cell through the \(-s\) cell-edge and exit through the \(+z\) cell-edge, are as follows:

1. By definition, this track will enter into the first plane in sequence for the polar cell. The algorithm described in Section A.6 is executed to determine from which planar section the track will exit.

2. The algorithm described in Section A.7 is executed to determine the number of planar sections crossed by the global track before it exits the polar cell.

3. If \(n_{\text{crossed}} = 1\), i.e., if the track crosses only one planar section before leaving the polar cell, no subdivision of this track is required and the track segment length for this plane, \(t_{n,p}\), is set equal to the global track length, \(t_p\).

4. If \(n_{\text{crossed}} > 1\), i.e., if the track crosses multiple planar sections before exiting the polar cell, the following algorithm is executed to subdivide the global track into segments for each of the crossed planes:
   
a. The maximum length of a track segment for a global track of this type will occur when the track enters the planar section from its \(-s\) boundary and
exits from its +s boundary. In this case, the maximum segment length is given by

\[ t_{n,p} \bigg|_{\text{max}} = \frac{S_{n,m}}{\sin(\theta_m)} \]  

(A.1)

b. The track segment length for the first planar section is set equal to \( t_{n,p} \bigg|_{\text{max}} \).

c. Step 4b is repeated for each planar section up to but not including the plane from which the global track exits, \( n_{\text{exit}} \).

d. The track segment length for the planar section from which the global track exits is set equal to the global track length, \( t_p \), minus the sum of all track segments calculated via Step 4b, or

\[ t_{n_{\text{exit}},p} = t_p - (n_{\text{cross}} - 1) \cdot t_{n,p} \bigg|_{\text{max}} \]  

(A.2)

### A.2 Procedures to Subdivide Tracks of Type 2

The procedures for subdividing global tracks of Type 2, which enter the polar cell through the -z cell-edge and exit through the +z cell-edge, are as follows:
1. The algorithm described in Section A.5 is executed to determine to which planar section the track will enter.

2. The algorithm described in Section A.6 is executed to determine from which planar section the track will exit.

3. The algorithm described in Section A.7 is executed to determine the number of planar sections crossed by the global track before it exits the polar cell.

4. If \( n_{\text{crossed}} = 1 \), i.e., if the track crosses only one planar section before leaving the polar cell, no subdivision of this track is required and the track segment length for this plane, \( t_{n,p} \), is set equal to the global track length, \( t_p \).

5. If \( n_{\text{crossed}} > 1 \), i.e., if the track crosses multiple planar sections before exiting the polar cell, the following algorithm is executed to subdivide the global track into segments for each of the crossed planes:

   a. The maximum length of a track segment for a global track of this type will occur when the track enters the planar section from its \(-s\) boundary and exits from its \(+s\) boundary. In this case, the maximum segment length is again given by Equation (A.1).
b. The track segment length for the planar section into which the global track enters is computed according to

\[ t_{n,p} = \frac{\Delta S - z_{in}}{\sin(\theta_m)} \]  \hspace{1cm} \text{(A.3)}

where \( \Delta S \) is the sum of all plane widths in sequence up to and including the plane into which the global track enters, \( n_{\text{entry}} \), and \( z_{in} \) is the entry location for that track.

c. The track segment length for the second planar section in sequence is set equal to \( t_{n,p} \big|_{\text{max}} \).

d. Step 4c is repeated for each remaining planar section up to, but not including, the plane from which the global track exits, \( n_{\text{exit}} \).

e. The track segment length for the planar section from which the global track exits is set equal to the global track length, \( t_p \), minus the sum of all track segments calculated via Step 4b, or

\[ t_{n_{\text{exit}},p} = t_p - (n_{\text{cross}} - 2) \cdot t_{n,p} \big|_{\text{max}} - \left( \frac{\Delta S - z_{in}}{\sin(\theta_m)} \right) \]  \hspace{1cm} \text{(A.4)}
A.3 Procedures to Subdivide Tracks of Type 3

The procedures for subdividing global tracks of Type 3, which enter the polar cell through the -s cell-edge and exit through the +s cell-edge, are as follows:

1. By definition, this track will enter into the first plane in sequence for the polar cell and will exit from the last plane in sequence.

2. The algorithm described in Section A.7 is executed to determine the number of planar sections crossed by the global track before it exits the polar cell.

3. If $n_{\text{crossed}} = 1$, i.e., if the track crosses only one planar section before leaving the polar cell, no subdivision of this track is required and the track segment length for this plane, $t_{n,p}$, is set equal to the global track length, $t_p$.

4. If $n_{\text{crossed}} > 1$, i.e., if the track crosses multiple planar sections before exiting the polar cell, then the track segment lengths for every planar section in this instance of the planar cell are set equal to $t_{n,p}^{\text{max}}$, which is again calculated according to Equation (A.1).
A.4 Procedures to Subdivide Tracks of Type 4

The procedures for subdividing global tracks of Type 4, which enter the polar cell through the -z cell-edge and exit through the +s cell-edge, are as follows:

1. By definition, this track will exit from the last plane in sequence for the polar cell. The algorithm described in Section A.5 is executed to determine to which planar section the track will enter.

2. The algorithm described in Section A.7 is executed to determine the number of planar sections crossed by the global track before it exits the polar cell.

3. If $n_{\text{crossed}} = 1$, i.e., if the track crosses only one planar section before leaving the polar cell, no subdivision of this track is required and the track segment length for this plane, $t_{n,p}$, is set equal to the global track length, $t_p$.

4. If $n_{\text{crossed}} > 1$, i.e., if the track crosses multiple planar sections before exiting the polar cell, the following algorithm is executed to subdivide the global track into segments for each of the crossed planes:
   
   a. The maximum length of a track segment for a global track of this type will occur when the track enters the planar section from its –s boundary and
exits from its +s boundary. In this case, the maximum segment length is again given by Equation (A.1).

b. The track segment length for the planar section into which the global track enters is computed according to

\[ t_{n,p} = \frac{\Delta S - z_{in}}{\sin(\theta_n)} \]  (A.5)

where \( \Delta S \) is the sum of all plane widths in sequence up to and including the plane into which the global track enters, \( n_{entry} \), and \( z_{in} \) is the entry location for that track.

c. The track segment length for all subsequent planar sections in the polar cell is set equal to \( t_{n,p} \mid_{\text{max}} \).

**A.5 Algorithm to Locate Entry Plane for Global Track**

If a track enters the polar cell along the –s cell edge, this track will by definition originate in the first planar section in sequence. For a track that enters the polar cell along the –z edge, however, the track may originate in a planar section that is later in sequence. The following algorithm is implemented to locate the plane in which the track originates:
1. A running sum, $\Delta S$, is initialized to the width of the first planar section in sequence for the polar cell.

2. The known entry location for the global track, $z_{\text{entry}}$, is compared to the running sum, $\Delta S$. If $\Delta S > z_{\text{entry}}$ then it is deduced that the global track enters the periodic polar cell into the current planar section and the algorithm is completed. Otherwise, it is deduced that the track enters into a subsequent planar section and the algorithm continues.

3. The running sum is incremented by adding to $\Delta S$ the width of the next planar section in the sequence dictated by the coupling relationships for this polar cell.

4. Steps 2 and 3 are repeated until the entry plane, $n_{\text{entry}}$, is determined.

### A.6 Algorithm to Locate Exit Plane for Global Track

If a track exits the polar cell along the $+s$ cell edge, this track will by definition end in the last planar section in sequence. For a track that exits the polar cell along the $+z$ edge, however, the track may end in a planar section that is earlier in sequence. The following algorithm is implemented to locate the plane in which the track ends:

1. A running sum, $\Delta S$, is initialized to the width of the first planar section in sequence for the polar cell.
2. The known exit location for the global track, \( z_{\text{exit}} \), is compared to the running sum, \( \Delta S \). If \( \Delta S > z_{\text{exit}} \) then it is deduced that the global track exits the periodic polar cell from the current planar section and the algorithm is completed. Otherwise, it is deduced that the track crosses into the next planar section and the algorithm continues.

3. The running sum is incremented by adding to \( \Delta S \) the width of the next planar section in the sequence dictated by the coupling relationships for this polar cell.

4. Steps 2 and 3 are repeated until the exit plane, \( n_{\text{exit}} \), is determined.

### A.7 Algorithm to Determine Number of Planes Crossed by Global Track

For all tracks, it is necessary to know how many planar sections the track crosses to define limits of the loops in the procedures of Sections A.1 through A.4. The following algorithm is implemented to determine the number of planes crossed by each track:

1. A counter, \( n_{\text{crossed}} \), is initialized to 1 and a pointer, \( n_{\text{current}} \), is set equal to the ID for the first planar section in sequence for the polar cell.
2. The ID for the current planar section, $n_{current}$, is compared to the ID for the exit plane, $n_{exit}$. If these ID’s are equal, then the algorithm is completed. Otherwise the counter $n_{crossed}$ is incremented by one and the pointer $n_{current}$ is advanced to the next planar section in sequence, according to the coupling relationships for the periodic polar cell.

3. Step 2 is repeated until the number of planes crossed by the global track, $n_{crossed}$, is determined.
APPENDIX B

SOURCE CODE FOR THE
MOCK-3D CODE PACKAGE

B1: Source Code for RayMonde
B2: Source Code for MOCK-3D
B3: Source Code for MOCK-3DK
PROGRAM RayMonde
!
 CALL openFiles()
!
 CALL turtle()
!
 CALL closeFiles()
!
 STOP
!
 END PROGRAM

SUBROUTINE openFiles()
!
 !!!!!!!!!!!!!
 USE files
 !!!!!!!!!!!!!
 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 OPEN (UNIT = ifile, FILE = ifname)
 OPEN (UNIT = ofile, FILE = ofname, STATUS = 'replace')
 OPEN (UNIT = afile, FILE = afname, STATUS = 'replace')
 OPEN (UNIT = dfile ,FILE = dfname, STATUS = 'replace')
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
 END

SUBROUTINE turtle()
!
 !!!!!!!!!!!!!
 USE keys
 !!!!!!!!!!!!!
 ! Collect Primary Input
 ! - required for array allocation and decision logic
 !!!!!!!!!!!!!
 CALL getKeys()
!!!!!!!!!!!!
!
 ! Load Remaining Input and Generate Initial Model Info
 !!!!!!!!!!!!!
 CALL genModel()
 CALL writeModel()
!!!!!!!!!!!!
!
 ! Correct Quadrature Set to Enforce Perfect Reflection
 !!!!!!!!!!!!!
 CALL fixQuad()
 CALL writeQuad()
!!!!!!!!!!!!!
! 2D Ray Tracing --> Azimuthal Plane
!
!!!!!!!!!!!!!!!!!!!!!!
   CALL trace2D()
   CALL writeRays2D()
!!!!!!!!!!!!!!!!!!!!!!
!
3D Ray Tracing --> Polar Plane
!
!!!!!!!!!!!!!!!!!!!!!!
   IF (ngeo.eq.3) THEN
      CALL trace3D()
      CALL writeRays3D()
   ENDIF
!!!!!!!!!!!!!!!!!!!!!!
!
Perform balance checks, and other debugging/testing exercises
!
!!!!!!!!!!!!!!!!!!!!!!
   CALL writeBalChk()
!!!!!!!!!!!!!!!!!!!!!!
!
Export Results to DATA File
!
!!!!!!!!!!!!!!!!!!!!!!
   CALL writeFinal()
!!!!!!!!!!!!!!!!!!!!!!
!
END

SUBROUTINE getKeys()
!
!!!!!!!!!!
   USE files
   USE keys
!!!!!!!!!!
!
Load Geometry Type from INPUT File
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   READ (ifile,*) ngeo
   IF ((ngeo.lt.2).or.(ngeo.gt.3)) THEN
      CALL errorInput(1)
   ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Load Quadrature Type from INPUT file
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   READ (ifile,*) nsn
   IF (((((nsn.ne.2).and.(nsn.ne.4)).and.(nsn.ne.6)) .and.(nsn.ne.8)).and.(nsn.ne.12)).and.(nsn.ne.16)) THEN
      CALL errorInput(2)
   ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RETURN
END
SUBROUTINE genModel()
!
!!!!!!!!!!!!!!!!!!!!
USE files
USE keys
USE cells
USE directions
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION dlin ! Track Spacing
INTEGER level ! Quad Type Pointer
INTEGER d ! direction index
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Load Remaining Input:
! - cell dimensions (dx,dy,dz)
! - track spacing (dlin)
!
!---all values must be positive---
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
IF (ngeo.eq.2) THEN
READ(ifile,*) dx,dy
IF ((dx.lt.0.0).or.(dy.lt.0.0)) THEN
CALL errorInput(3)
ENDIF
ELSE IF (ngeo.eq.3) THEN
READ(ifile,*) dx,dy,dz
IF (((dx.lt.0.0).or.(dy.lt.0.0)) .and. 
 & .or.(dz.lt.0.0)) THEN
CALL errorInput(3)
ENDIF
ENDIF
READ(ifile,*) dlin
IF (dlin.lt.0.0) THEN
CALL errorInput(4)
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Find Pointer to Correct Sn Level
! in Level Symmetric Quadrature Data
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
IF (nsn.eq.4) THEN
level = lev4
ELSE IF (nsn.eq.6) THEN
level = lev6
ELSE IF (nsn.eq.8) THEN
level = lev8
ELSE IF (nsn.eq.12) THEN
level = lev12
ELSE IF (nsn.eq.16) THEN
level = lev16
ELSE
level = lev2
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Set DIRNUM (# Directions in Quad Set)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
dirnum = nsn * (nsn + 2) / 8

! Allocate Direction Set Array
!
ALLOCATE(dirset(dirnum))

! Initialize Direction Set Array
!
DO d = 1,dirnum
   dirset(d)%wght = wgtdata(d,level)
   dirset(d)%azim = azidata(d,level)
   dirset(d)%dazi = dlin
   dirset(d)%nnx = -1
   dirset(d)%nny = -1
   dirset(d)%nnz = -1
   dirset(d)%nns = -1
   dirset(d)%pola = poldata(d,level)
   dirset(d)%dpol = dlin
   dirset(d)%nns = -1
   dirset(d)%nnz = -1
END DO

! Build Prime Number Array
!
CALL getPrimes()

RETURN
END

SUBROUTINE errorInput(ecode)
!
USE files
!
INTEGER ecode ! error code to select correct message
!
SELECT CASE(ecode)
CASE(1)
   WRITE(*,*) 'INPUT ERROR: NSN must be 2 or 3'
   WRITE(*,*) 'Calculation Aborted'
   WRITE(ofile,*) 'INPUT ERROR: NSN must be 2 or 3'
   WRITE(ofile,*) 'Calculation Aborted'
CASE(2)
   WRITE(*,*) 'INPUT ERROR: NSN must be 4, 6, 8, 12, or 16'
   WRITE(*,*) 'Calculation Aborted'
   WRITE(ofile,*) 'INPUT ERROR: NSN must be 4, 6, 8, 12, or 16'
   WRITE(ofile,*) 'Calculation Aborted'
CASE(3)
   WRITE(*,*) 'INPUT ERROR: DX, DY, and DZ must be positive'
   WRITE(*,*) 'Calculation Aborted'
   WRITE(ofile,*) 'INPUT ERROR: DX, DY, and DZ must be positive'
END SELECT
SUBROUTINE getPrimes()
!
!!!!!!!!!!!!
USE keys
!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER g       ! Remainder of Tested Number and Prime
INTEGER num     ! Number to be Determined if Prime
INTEGER stopper ! Iteration Count (# Array Values)
LOGICAL pflag   ! Is Number Prime?
INTEGER i, j    ! looping indices
!!!!!!!!!!!!!!!!!!!!
!
! Allocate Prime Number Array
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE (prime(nprime))
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Build Prime Number Array
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
num = 2
stopper = 1
DO i = 1,nprime
100 CONTINUE
IF (stopper.gt.60000) EXIT
stopper = stopper + 1
pflag = .true.
DO j = 2,num-1
   g = MOD(num,j)
   IF (g.eq.0) THEN
      pflag = .false.
      EXIT
   ENDIF
END DO 
IF (pflag) THEN 
prime(i) = num
num = num + 1
ELSE
num = num + 1
GO TO 100
ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RETURN
END

SUBROUTINE writeModel()
!
!!!!!!!!!!!!!!!!!!!!
USE files
USE keys
USE cells
USE directions
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!
INTEGER d ! Direction Index
!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,10) WRITE (ofile,20) WRITE (ofile,21) WRITE (ofile,22) WRITE (ofile,23) WRITE (ofile,24) WRITE (ofile,25) WRITE (ofile,26) WRITE (ofile,27) WRITE (ofile,10) WRITE (ofile,40) WRITE (ofile,42) ngeo
IF (ngeo.eq.2) THEN
WRITE (ofile,44)
ELSE
WRITE (ofile,45)
ENDIF
WRITE (ofile,10) WRITE (ofile,50) WRITE (ofile,55) 'Cell x-Dimension =',dx WRITE (ofile,55) 'Cell y-Dimension =',dy IF (ngeo.eq.3) THEN
WRITE (ofile,55) 'Cell z-Dimension =',dz
ENDIF
! WRITE (ofile,10)
! WRITE (ofile,60)
! WRITE (ofile,62) dlin
WRITE (ofile,10)
WRITE (ofile,100)
WRITE (ofile,102) nsn
WRITE (ofile,104) dirnum
WRITE (ofile,10)
DO d = 1,dirnum
WRITE (ofile,110) d,dirnum
WRITE (ofile,115) dirset(d)%wght
WRITE (ofile,120) dirset(d)%azim
WRITE (ofile,125) dirset(d)%dazi
WRITE (ofile,130) dirset(d)%nnx
WRITE (ofile,135) dirset(d)%nny
WRITE (ofile,140) dirset(d)%pola
WRITE (ofile,145) dirset(d)%dpol
WRITE (ofile,150) dirset(d)%nns
WRITE (ofile,155) dirset(d)%nnz
WRITE (ofile,10)
END DO

END

SUBROUTINE fixQuad()
!
!!!!!!!!!!!!!!!!!!!!!!!!!
USE keys
USE cells
USE directions
USE rays
!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION aztmp ! Azimuthal Angle
DOUBLE PRECISION datmp ! Azimuthal Track Spacing
DOUBLE PRECISION nxtmp ! # Spans in x-direction, raw
INTEGER nnxtmp ! clng int
DOUBLE PRECISION nytmp ! # Spans in y-direction, raw
INTEGER nnytmp ! clng int
INTEGER naztmp ! Max # Tracks in Azimuthal Plane
INTEGER nputmp ! # Azimuthal Units
DOUBLE PRECISION dstmp ! Azimuthal Unit Dimension
DOUBLE PRECISION poltmp ! Polar Direction
DOUBLE PRECISION dptmp ! Cross-Sectional Polar Track Area
DOUBLE PRECISION nztmp ! # Spans in z-direction, raw
INTEGER nnztmp ! clng int
DOUBLE PRECISION nstmp ! # Spans in s-direction, raw
INTEGER nnstmp ! clng int
INTEGER npotmp ! Max # Tracks in Polar Plane
INTEGER d ! direction index

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Correct Azimuthal Angle and Spacing
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
naztmp = 0
nazmax = 0
DO d = 1,dirnum
  aztmp = dirset(d)%azim
  datmp = dirset(d)%dazi
  nxtmp = dy * abs(cos(aztmp)) / datmp
  nytmp = dx * abs(sin(aztmp)) / datmp
  nnxtmp = CEILING(nxtmp)
  nnytmp = CEILING(nytmp)
  CALL checkLCM(nnxtmp,nnytmp,nputmp)
  aztmp = atan((nnytmp * dy) / (nnxtmp * dx))
  datmp = (dx * dy) / sqrt((nnxtmp * dx)**2 + (nnytmp * dy)**2)
  dirset(d)%azim = aztmp
  dirset(d)%dazi = datmp
  dirset(d)%nnx = nnxtmp
  dirset(d)%nny = nnytmp
  dirset(d)%npu = nputmp
  naztmp = nnxtmp + nnytmp
  IF (naztmp.gt.nazmax) THEN
    nazmax = naztmp
  ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Correct Polar Angle and Spacing
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
npotmp = 0
npomax = 0
IF (ngeo.eq.3) THEN
  DO d = 1,dirnum
    poltmp = dirset(d)%pola
    dptmp = dirset(d)%dpol
    dstmp = (dx * dy) / (dirset(d)%npu * dirset(d)%dazi)
    nstmp = dz * abs(sin(poltmp)) / dptmp
    nztmp = dstmp * abs(cos(poltmp)) / dptmp
    nnstmp = CEILING(nstmp)
    nnztmp = CEILING(nztmp)
    poltmp = atan((nnstmp * dstmp) / (nnztmp * dz))
    dptmp = (dz * dstmp) / sqrt((nnztmp * dz)**2 + (nnstmp * dstmp)**2)
  END DO
ENDIF
& sqrt((nnstmp * dstmp)**2 + (nnztmp * dz)**2)
dirset(d)%pola = poltmp
dirset(d)%dpol = dptmp
dirset(d)%nns = nnstmp
dirset(d)%nnz = nnztmp
npotmp = nnstmp + nnztmp
IF (npotmp.gt.npomax) THEN
  npomax = npotmp
ENDIF
END DO
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RETURN
END

SUBROUTINE checkLCM(xnum, ynum, fnum)
  
  checkLCM -- Determines the least Common multiple of two integers;
  ! Returns the multiple by which they are reduced.
  
  !!!!!!!!!!!!!
  USE keys
  !!!!!!!!!!!!!
  
  !!!!!!!!!!!!!!!!!!!!!!!
  INTEGER xnum, ynum ! Passed Values
  INTEGER fnum       ! Reduction Factor
  !!!!!!!!!!!!!!!!!!!!!!!
  
  !!!!!!!!!!!!!!!!!!!
  INTEGER xx, yy   ! Reduced Values
  INTEGER ff       ! Reduction Factor
  INTEGER xm, ym   ! Remainder of Reduced Value / Current Prime
  INTEGER maxit    ! Largest of Two Passed Values
  INTEGER h        ! Largest of Two Reduced Values
  INTEGER maxprime ! Largest Prime to be Tested for LCM
  INTEGER g        ! Current Prime to be Tested for LCM
  LOGICAL rflag    ! Boolean - LCM Found?
  INTEGER n, p     ! Looping Indices
  !!!!!!!!!!!!!!!!!!!!
  
  ! Determine Largest
  ! Of Two Passed Values
  !
  xx = xnum
  yy = ynum
  maxit = MAX(xx, yy)
  ff = 1
  !!!!!!!!!!!!!!!!!!!!
  
  !!!!!!!!!!!!!!!!!!!!
  DO n = 1,maxit ! Begin Reduction Iteration
  !!!!!!!!!!!!!!!!!!!!
  
  ! Determine Largest Prime
  ! To Be Tested for LCM
  !
  h = MAX(xx, yy)
DO p = 1,h
    IF (prime(p).lt.h) THEN
        CONTINUE
    ELSE
        maxprime = p
        EXIT
    END IF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Reduce Passed Values Until Not Reducible
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

rflag = .true.
DO p = 1,maxprime
    g = prime(p)
    xm = MOD(xx,g)
    ym = MOD(yy,g)
    IF ((xm.eq.0).and.(ym.eq.0)) THEN
        xx = xx / g
        yy = yy / g
        ff = ff * g
        rflag = .false.
        EXIT
    END IF
END DO
IF (rflag) EXIT
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!
END DO ! End Reduction Iteration
!!!!!!!!!!
!
! Return Reduced Values
!
! xnum = xx     to return reduced number pair
! ynum = yy     include these two lines
!!!!!!!!!!!!
    fnum = ff
!!!!!!!!!!!!
!
RETURN
END
SUBROUTINE writeQuad()
!
!!!!!!!!!!!!!!!!!!!!
USE files
USE keys
USE directions
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!
    INTEGER d ! Direction Index
!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    WRITE (ofile,10)
    WRITE (ofile,20)
    WRITE (ofile,21)
    WRITE (ofile,22)
WRITE (ofile,23)
WRITE (ofile,24)
WRITE (ofile,25)
WRITE (ofile,26)
WRITE (ofile,27)
WRITE (ofile,10)
IF (ngeo.eq.2) THEN
  DO d = 1,dirnum
    WRITE (ofile,100) d,dirnum
    WRITE (ofile,10)
    WRITE (ofile,105) dirset(d)%wght
    WRITE (ofile,110) dirset(d)%azim
    WRITE (ofile,115) dirset(d)%dazi
    WRITE (ofile,120) dirset(d)%nnx
    WRITE (ofile,125) dirset(d)%nny
    WRITE (ofile,130) dirset(d)%pola
    WRITE (ofile,10)
  END DO
ELSE IF (ngeo.eq.3) THEN
  DO d = 1,dirnum
    WRITE (ofile,100) d,dirnum
    WRITE (ofile,10)
    WRITE (ofile,105) dirset(d)%wght
    WRITE (ofile,110) dirset(d)%azim
    WRITE (ofile,115) dirset(d)%dazi
    WRITE (ofile,120) dirset(d)%nnx
    WRITE (ofile,125) dirset(d)%nny
    WRITE (ofile,130) dirset(d)%pola
    WRITE (ofile,135) dirset(d)%dpol
    WRITE (ofile,140) dirset(d)%nns
    WRITE (ofile,145) dirset(d)%nnz
    WRITE (ofile,10)
  END DO
ENDIF

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Format Statements
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
10 FORMAT(1x)
20 FORMAT(3x,'******************************************************************************')
21 FORMAT(3x,'*o---------------------------------------o*')
22 FORMAT(3x,'*|                              |*')
23 FORMAT(3x,'*|  Quadrature Information  |*')
24 FORMAT(3x,'*| (Corrected)              |*')
25 FORMAT(3x,'*|                              |*')
26 FORMAT(3x,'*o---------------------------------------o*')
27 FORMAT(3x,'******************************************************************************')
100 FORMAT(3x,'DIRECTION #',i2,' OF ',i2,':')
105 FORMAT(14x,'Weight = ',es13.6)
110 FORMAT(5x,'Azimuthal Angle = ',es13.6,' rad')
115 FORMAT(3x,'Azimuthal Spacing = ',es13.6,' cm')
120 FORMAT(17x,'NNX = ',i4)
125 FORMAT(17x,'NNY = ',i4)
130 FORMAT(9x,'Polar Angle = ',es13.6,' rad')
135 FORMAT(7x,'Polar Spacing = ',es13.6,' cm')
140 FORMAT(17x,'NNS = ',i4)
145 FORMAT(17x,'NNZ = ',i4)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END
SUBROUTINE trace2D()
!
!!!!!!!!!!!!!!!!!!!!
USE keys
USE cells
USE directions
USE rays
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION aztmp ! azimuthal direction
DOUBLE PRECISION datmp ! azimuthal track spacing
INTEGER nnxtmp ! # tracks entering at -x face
INTEGER nnytmp ! # tracks entering at -y face
INTEGER d ! direction index
INTEGER n ! track index
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: inx(:) !
DOUBLE PRECISION, ALLOCATABLE :: iny(:) !
DOUBLE PRECISION, ALLOCATABLE :: track(:) !
DOUBLE PRECISION, ALLOCATABLE :: cplto(:) !
DOUBLE PRECISION, ALLOCATABLE :: cplfr(:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Allocate Azimuthal Track-Length Array
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE (xyray(nazmax,dirnum))
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Azimuthal Iteration
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO d = 1,dirnum
  aztmp = dirset(d)%azim
  datmp = dirset(d)%dazi
  nnxtmp = dirset(d)%nnx
  nnytmp = dirset(d)%nny
  ALLOCATE (inx(nnxtmp),iny(nnytmp))
  ALLOCATE (track(nnxtmp+nnytmp))
  ALLOCATE (cplto(nnxtmp+nnytmp))
  ALLOCATE (cplfr(nnxtmp+nnytmp))
  CALL traceCell(aztmp,datmp,dx,dy,nnxtmp,nnytmp, &
                 & inx,iny,track,cplto,cplfr)
  DO n = 1,nnxtmp+nnytmp
    xyray(n,d)%trcln = track(n)
    xyray(n,d)%trcto = cplto(n)
    xyray(n,d)%trcfr = cplfr(n)
  END DO
  DEALLOCATE (inx)
  DEALLOCATE (iny)
  DEALLOCATE (track)
  DEALLOCATE (cplto)
  DEALLOCATE (cplfr)
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RETURN
END
SUBROUTINE traceCell(angle,trsep,dx1,dx2,nnx1,nnx2, &
    inx1,inx2,track,trcto,trcfr)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION angle ! Direction Angle
DOUBLE PRECISION trsep ! Track Spacing
DOUBLE PRECISION dx1   ! 1st Cell Dimension
DOUBLE PRECISION dx2   ! 2nd Cell Dimension
INTEGER          nnx1  ! # Tracks Entering at -x1 Edge
INTEGER          nnx2  ! # Tracks Entering at -x2 Edge
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DOUBLE PRECISION, DIMENSION(nnx1) :: inx1   !
DOUBLE PRECISION, DIMENSION(nnx2) :: inx2   !
DOUBLE PRECISION, DIMENSION(nnx1+nnx2) :: track !
DOUBLE PRECISION, DIMENSION(nnx1+nnx2) :: trcto !
DOUBLE PRECISION, DIMENSION(nnx1+nnx2) :: trcfr !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

INTEGER          ntot  ! total # tracks crossing the cell
DOUBLE PRECISION dax1p ! track spacing projected to -x1 face
DOUBLE PRECISION dax2p ! track spacing projected to -x2 face
DOUBLE PRECISION intmp ! location where track enters cell
DOUBLE PRECISION ttmp  ! length of track
DOUBLE PRECISION tmax  ! maximum track length for this direction
INTEGER          tflg  ! flag to determine when tmax is reached
INTEGER          n,nn  ! track indices
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

ntot  = nnx1 + nnx2     ! calc. total # tracks
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

dax1p = trsep / abs(cos(angle)) ! project track spacing to
dax2p = trsep / abs(sin(angle)) ! -x1 (dax1p) & -x2 (dax2p) faces
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Find All Track Entry Locations:

*NOTE*
all entry locations are referenced to
an origin at the bottom-left corner of the cell:

+X face
-----------------
|   X  |   |
|   |   |
| -x1 face | +x1 face
|   X   |   |
|   |   |
|------X------X---
| origin -x2 face
|
X -> entry locations

inx2(1) = (dx1 + (nnx2 - 1) * dax2p) / 2  
IF (nnx2.gt.1) THEN
DO n = 2,nnx2
    inx2(n) = inx2(n-1) - dax2p  ! along -x2 face
END DO
ENDIF
inx1(1) = (dax2p - inx2(nnx2)) * tan(angle)!
IF (nnx1.gt.1) THEN
  DO n = 2,nnx1
    inx1(n) = inx1(n-1) + dax1p
  END DO
ENDIF
!

! Calculate Track Lengths:
!
! tracks that enter from -x2 face
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
! TO
  DO n = 1,nnx1
    trcto(n) = n + nnx2
  END DO
! FROM
  DO n = nnx2+nnx1,nnx1+1,-1
    trcto(n) = n - nnx1
  END DO
300
trcfr(n) = n + nnx1
END DO
DO n = nnx2+1,nnx2+nnx1
 trcfr(n) = n - nnx2
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RETURN
END

SUBROUTINE writeRays2D()
!
!!!!!!!!!!!!!!!!!!!!
USE files
USE keys
USE directions
USE rays
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER ntmp ! # Tracks in Cell
INTEGER d    ! Direction Index
INTEGER n    ! Track Index
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,10)
WRITE (ofile,20)
WRITE (ofile,21)
WRITE (ofile,22)
WRITE (ofile,23)
WRITE (ofile,24)
WRITE (ofile,25)
WRITE (ofile,26)
WRITE (ofile,10)
DO d = 1,dirnum
  WRITE (ofile,100) d
  WRITE (ofile,101)
  WRITE (ofile,102)
  ntmp = dirset(d)%nnx + dirset(d)%nny
  DO n = 1,ntmp
    WRITE (ofile,110) n,xyray(n,d)%trcln, &
    & xyray(n,d)%trcfr,xyray(n,d)%trcto
  END DO
  WRITE (ofile,10)
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Format Statements
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
10 FORMAT(1x)
20 FORMAT(3x,'**********************************')
21 FORMAT(3x,'*o------------------------------o*')
22 FORMAT(3x,'*|                              |*')
23 FORMAT(3x,'*|  2D Ray Tracing Information  |*')
24 FORMAT(3x,'*|                              |*')
25 FORMAT(3x,'*o------------------------------o*')
26 FORMAT(3x,'**********************************')
100 FORMAT(4x,'Direction #',i2,16x,'Coupling')
101 FORMAT(4x,'Track #  Length [cm]   Fr  To')
SUBROUTINE trace3D()
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
END

DOUBLE PRECISION datmp   ! Azimuthal Track Separation
DOUBLE PRECISION dptmp   ! Polar Track Separation
DOUBLE PRECISION poltmp  ! Polar Angle
DOUBLE PRECISION alpha   ! [Pi/2 - Polar Angle]
DOUBLE PRECISION dstmp   ! Plane Width (2D Track Length)
DOUBLE PRECISION intmp   ! Track Entry Location
DOUBLE PRECISION dsisum  ! Track Exit Location - Running Sum
DOUBLE PRECISION dsosum  ! Track Exit Location - Running Sum
DOUBLE PRECISION ttmp    ! Track Length
DOUBLE PRECISION tsubtot ! Track Length - Running Sum

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER naztmp      ! # Planes in Cell
INTEGER npltmp      ! # Tracks in Plane
INTEGER nnstmp      ! # Entries from -s edge
INTEGER nnztmp      ! # Entries from -z edge
INTEGER nputmp      ! # Periodic Units
INTEGER npcross,npc ! # Planes Crossed by Track
INTEGER ostmp       ! Offset Counter (ZRAY Repack)
INTEGER osflg       ! Offset Flag (ZRAY Repack)
INTEGER nplto       ! # Tracks in Next Plane
INTEGER nplfr       ! # Tracks in Previous Plane
!
DOUBLE PRECISION, ALLOCATABLE :: ins(:)   ! Entry Locations: -s edge
DOUBLE PRECISION, ALLOCATABLE :: inz(:)   ! Entry Locations: -z edge
DOUBLE PRECISION, ALLOCATABLE :: track(:) ! Track Lengths - Pdc Unit
DOUBLE PRECISION, ALLOCATABLE :: cplto(:) ! Coupling - Periodic Unit
DOUBLE PRECISION, ALLOCATABLE :: cplfr(:) ! Coupling - Periodic Unit
DOUBLE PRECISION, ALLOCATABLE :: tracks(:,,:,:,:) ! Track Lengths - Seg

INTEGER, ALLOCATABLE :: offset(:,:) ! Offset Array (ZRAY Repack)

ALLOCATE (zray(npomax,nazmax,dirnum))
ALLOCATE (npl(nazmax,dirnum))
ALLOCATE (tracks(npomax,nazmax,dirnum))
ALLOCATE (offset(nazmax,dirnum))

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Initialize Track Lengths to Negative Value
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO d = 1,dirnum
  DO n = 1,nazmax
    DO p = 1,npomax
      zray(p,n,d)%trcln = -1.0
      zray(p,n,d)%plnto = -1
      zray(p,n,d)%trcto = -1
      zray(p,n,d)%plnfr = -1
      tracks(p,n,d) = -1.0
    END DO
  END DO
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Polar Iteration
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO d = 1,dirnum
  !
  ! datmp = dirset(d)%dazi
  ! poltmp = dirset(d)%pola
  ! alpha = hpi - poltmp
  ! dptmp = dirset(d)%dpol
  ! nnstmp = dirset(d)%nns
  ! nnztmp = dirset(d)%nnz
  ! nputmp = dirset(d)%npu
  ! dstmp = (dx * dy) / (nputmp * datmp)
  !
  ! ALLOCATE (ins(nnstmp))
  ! ALLOCATE (inz(nnztmp))
  ! ALLOCATE (tracks(nnztmp))
  ! ALLOCATE (cplto(nnztmp))
  ! ALLOCATE (cplfr(nnztmp))
  !
  ! CALL traceCell(alpha,dptmp,dstmp,dz,nnstmp,nnztmp, &
  !               ins,inz,track,cplto,cplfr)
  !
  ! debugging...to check contents of ins, inz, and track arrays
  ! CALL dbg3D(d,dstmp,nnstmp,nnztmp,ins,inz,track)
  !
  ! Segmenting the Total Track Lengths
  !
  ! *******************
  ! * CASE 1: Nz > Ns *
  ! *******************
  !
  !write(*,*) 'case 1'

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

IF (nnztmp.gt.nnstmp) THEN
  DO mtnd = 1,nputmp
    pin = mtnd
    pout = mtnd
    nin = nnztmp
    nout = nnztmp
    dsisum = xyray(pin,d)%trcln
    dsosum = xyray(pout,d)%trcln
  END DO
END IF
! Case 1
! Track Set #1
! - Enter -s face
! - Leave +z face
!
! write(*,*) 'set 1'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnztmp+nnstmp,nnztmp+1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ...Find Exit Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
outmp = inz(nout)
DO
  IF (outmp.eq.dsosum) THEN
    CALL errorInput(5)
  ELSE IF (outmp.lt.dsosum) THEN
    EXIT
  ELSE
    pout = xyray(pout,d)%trcto
dsosum = dsosum + xyray(pout,d)%trcln
  ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Subdivide Total Track Length
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)
pl = mtnd
IF (npcross.eq.1) THEN
  tracks(n,pl,d) = track(n)
ELSE
  ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
  tracks(n,pl,d) = ttmp
tsubtot = ttmp
  pl = xyray(pl,d)%trcto
  DO npc = 1,npcross-2
     ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
     tracks(n,pl,d) = ttmp
     tsubtot = tsubtot + ttmp
     pl = xyray(pl,d)%trcto
  END DO
  tracks(n,pl,d) = track(n) - tsubtot
ELSE
  tracks(n,pl,d) = track(n)
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Next Track
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
nout = nout - 1
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Case 1
! Track Set #2
! - Enter -z face
! - Leave +z face
!
! write(*,*) 'set 2'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnztmp,nnstmp+1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!      ...Find Entry Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
intmp = inz(nin)
DO
   IF (intmp.eq.dsisum) THEN
      CALL errorInput(5)
   ELSE IF (intmp.lt.dsisum) THEN
      EXIT
   ELSE
      pin = xyray(pin,d)%trcto
      dsisum = dsisum + xyray(pin,d)%trcln
   ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!      ...Find Exit Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
outmp = inz(nout)
DO
   IF (outmp.eq.dsosum) THEN
      CALL errorInput(5)
   ELSE IF (outmp.lt.dsosum) THEN
      EXIT
   ELSE
      pout = xyray(pout,d)%trcto
      dsosum = dsosum + xyray(pout,d)%trcln
   ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!      ...Subdivide Total Track Length
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)
pl = pin
IF (npcross.eq.1) THEN
   tracks(n,pl,d) = track(n)
ELSE
   tsubtot = 0.0
   intmp = dsisum - inz(nin)
   tttmp = intmp / abs(sin(poltmp))
   tracks(n,pl,d) = tttmp
   tsubtot = tsubtot + tttmp
   pl = xyray(pl,d)%trcto
   DO npc = 1,npcross-2
      tttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
      tracks(n,pl,d) = tttmp
      tsubtot = tsubtot + tttmp
      pl = xyray(pl,d)%trcto
   END DO
   tracks(n,pl,d) = track(n) - tsubtot
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!      ...Next Track
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!
nin = nin - 1

305
nout = nout - 1
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Case 1
! Track Set #3
! - Enter -z face
! - Leave +s face
!
!write(*,*) 'set 3'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnstmp,1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Find Entry Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
intmp = inz(nin)
DO
IF (intmp.eq.dsisum) THEN
 CALL errorInput(5)
 ELSE IF (intmp.lt.dsisum) THEN
 EXIT
 ELSE
 pin = xyray(pin,d)%trcto
 dsisum = dsisum + xyray(pin,d)%trcln
 ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Subdivide Total Track Length
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)
pl = pin
IF (npcross.eq.1) THEN
 tracks(n,pl,d) = track(n)
 ELSE
 intmp = dsisum - inz(nin)
ttmp = intmp / abs(sin(poltmp))
 tracks(n,pl,d) = ttmp
 tsubtot = ttmp
 pl = xyray(pl,d)%trcto
 DO npc = 1,npcross-2
 ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
 tracks(n,pl,d) = ttmp
 tsubtot = tsubtot + ttmp
 pl = xyray(pl,d)%trcto
 END DO
 tracks(n,pl,d) = track(n) - tsubtot
 ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Next Track
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
nin = nin - 1
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  *******************
!  * CASE 2: Nz < Ns *
!  *******************
!
!write(*,*) 'case 2'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ELSE IF (nnstmp.gt.nnztmp) THEN
!
  DO mtnd = 1,nputmp
    pin = mtnd
    pout = mtnd
    nin = nnztmp
    nout = nnztmp
    dsisum = xyray(pin,d)%trcln
    dsosum = xyray(pout,d)%trcln
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!       Case 2
!       Track Set #1
!       - Enter -s face
!       - Leave +z face
!
!write(*,*) 'set 1'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnztmp+nnstmp,nnstmp+1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!         ...Find Exit Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
outmp = inz(nout)
DO
  IF (outmp.eq.dsosum) THEN
    CALL errorInput(5)
  ELSE IF (outmp.lt.dsosum) THEN
    EXIT
  ELSE
    pout = xyray(pout,d)%trcto
    dsosum = dsosum + xyray(pout,d)%trcln
  ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!         ...Subdivide Total Track Length
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)
pl = mtnd
IF (npcross.eq.1) THEN
  tracks(n,pl,d) = track(n)
ELSE
  ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
  tracks(n,pl,d) = ttmp
  tsubtot = ttmp
  pl = xyray(pl,d)%trcto
  DO npc = 1,npcross-2
    ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
    tracks(n,pl,d) = ttmp
    tsubtot = tsubtot + ttmp
    pl = xyray(pl,d)%trcto
  END DO
  tracks(n,pl,d) = track(n) - tsubtot

ENDIF

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
...Next Track
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
nout = nout - 1
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Case 2
!
  Track Set #2a
!
  - Enter -z face
!
  - Leave +z face
!
!write(*,*) 'set 2a'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnstmp,nnztmp+1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
...Set Exit Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
pout = xyray(mtnd,d)%trcfr
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
...Subdivide Total Track Length
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)
pl = pin
IF (npcross.eq.1) THEN
  tracks(n,pl,d) = track(n)
ELSE
  ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
  tracks(n,pl,d) = ttmp
  pl = xyray(pl,d)%trcto
  DO npc = 1,npcross-2
    ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
    tracks(n,pl,d) = ttmp
    pl = xyray(pl,d)%trcto
  END DO
  ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
  tracks(n,pl,d) = ttmp
ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Case 2
!
  Track Set #3
!
  - Enter -z face
!
  - Leave +s face
!
!write(*,*) 'set 3'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnztmp,1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
...Find Entry Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
intmp = inz(nin)
DO

308
IF (intmp.eq.dsisum) THEN  
   CALL errorInput(5)  
ELSE IF (intmp.lt.dsisum) THEN  
   EXIT  
ELSE  
   pin = xyray(pin,d)%trcto  
   dsisum = dsisum + xyray(pin,d)%trcln  
ENDIF  
END DO  

!! Subdivide Total Track Length  
!!  
CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)  
pl = pin  
IF (npcross.eq.1) THEN  
   tracks(n,pl,d) = track(n)  
ELSE  
   intmp = dsisum - inz(nin)  
   ttmp = intmp / abs(sin(poltmp))  
   tracks(n,pl,d) = ttmp  
   tsubtot = ttmp  
   pl = xyray(pl,d)%trcto  
   DO npc = 1,npcross-2  
      ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))  
      tracks(n,pl,d) = ttmp  
      tsubtot = tsubtot + ttmp  
      pl = xyray(pl,d)%trcto  
   END DO  
   tracks(n,pl,d) = track(n) - tsubtot  
ENDIF  
!!  
!! Next Track  
!!  
!!         Case 3  
!!         
!!write(*,*) 'case 3'  
ELSE  
   DO mtnd = 1,nputmp  
      
      pin = mtnd  
      pout = mtnd  
      nin = nnztmp  
      nout = nnztmp  
      dsisum = xyray(pin,d)%trcln  
      dsosum = xyray(pout,d)%trcln  
   
   ELSE  
   
      Case 3
! Track Set #1
! - Enter -s face
! - Leave +z face
!
!write(*,*) 'set 1'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnztmp+nnstmp,nnztmp+1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Find Exit Plane
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
outmp = inz(nout)
DO
  IF (outmp.eq.dsosum) THEN
    CALL errorInput(5)
  ELSE IF (outmp.lt.dsosum) THEN
    EXIT
  ELSE
    pout = xyray(pout,d)%trcto
    dsosum = dsosum + xyray(pout,d)%trcln
  ENDIF
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Subdivide Total Track Length
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)
p1 = mtnord
IF (npcross.eq.1) THEN
  tracks(n,p1,d) = track(n)
ELSE
  tttmp = xyray(p1,d)%trcln / abs(sin(poltmp))
  tracks(n,p1,d) = tttmp
  tsutot = tttmp
  p1 = xyray(p1,d)%trcto
  DO npc = 1,npcross-2
     tttmp = xyray(p1,d)%trcln / abs(sin(poltmp))
     tracks(n,p1,d) = tttmp
     tsutot = tsutot + tttmp
     p1 = xyray(p1,d)%trcto
  END DO
  tracks(n,p1,d) = track(n) - tsutot
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! ...Next Track
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
nout = nout - 1
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Case 3
!
Track Set #3
!
- Enter -z face
!
- Leave +S face
!
!write(*,*) 'set 3'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO n = nnztmp,1,-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
intmp = inz(nin)
DO
  IF (intmp.eq.dsisum) THEN
    CALL errorInput(5)
  ELSE IF (intmp.lt.dsisum) THEN
    EXIT
  ELSE
    pin = xyray(pin,d)%trcto
    dsisum = dsisum + xyray(pin,d)%trcln
  ENDIF
END DO

pout = xyray(mtnd,d)%trcfr

CALL getNPC(pin,pout,d,nnztmp+nnstmp,npcross)
pl = pin
IF (npcross.eq.1) THEN
  tracks(n,pl,d) = track(n)
ELSE
  intmp = dsisum - inz(nin)
  ttmp = intmp / abs(sin(poltmp))
  tracks(n,pl,d) = ttmp
  tsubtot = ttmp
  pl = xyray(pl,d)%trcto
  DO npc = 1,npcross-2
    ttmp = xyray(pl,d)%trcln / abs(sin(poltmp))
    tracks(n,pl,d) = ttmp
    tsubtot = tsubtot + ttmp
    pl = xyray(pl,d)%trcto
  END DO
  tracks(n,pl,d) = track(n) - tsubtot
ENDIF

nin = nin - 1
END DO

Dealocate Direction Dependent Arrays
DO d = 1, dirnum
  DO n = 1, nazmax
    ostmp = 0
    osflg = 0
    pp = 0
    pc = npomax + 1
    DO p = 1, npomax
      pc = pc - 1
      IF ((tracks(p, n, d).lt.0).and.(osflg.eq.0)) THEN
        ostmp = ostmp + 1
      ELSE IF ((tracks(p, n, d).ge.0).and.(osflg.eq.0)) THEN
        osflg = 1
        pp = pp + 1
      ELSE IF ((tracks(p, n, d).ge.0).and.(osflg.eq.1)) THEN
        pp = pp + 1
      ELSE IF ((tracks(p, n, d).lt.0).and.(osflg.eq.1)) THEN
        ENDIF
      END DO
    offset(n, d) = ostmp
    npl(n, d) = pp
  END DO
 END DO
!
!
! Repack ZRAY Array
!
DO d = 1, dirnum
  DO n = 1, nazmax
    npltmp = npl(n, d)
    ostmp = offset(n, d)
    pp = ostmp + 1
    DO p = 1, npltmp
      zray(p, n, d)%trcln = tracks(pp, n, d)
      pp = pp + 1
    END DO
  END DO
  END DO
!
!
DO d = 1, dirnum
  naztmp = dirset(d)%nnx + dirset(d)%nny
  nnstmp = dirset(d)%nns
  DO n = 1, naztmp
    nto = xyray(n, d)%trcto
    nfr = xyray(n, d)%trcfr
    npltmp = npl(n, d)
    nplto = npl(nto, d)
    nplfr = npl(nfr, d)
    pp = nplto - nnstmp + 1
    DO p = 1, nnstmp
      zray(p, n, d)%plnto = nto
      zray(p, n, d)%trcto = pp
    END DO
  END DO
END DO
pp = pp + 1
END DO
pp = 1
DO p = nnstmp+1,npltmp
   zray(p,n,d)%plnto = n
   zray(p,n,d)%trcto = pp
   pp = pp + 1
END DO
pp = nnstmp + 1
DO p = 1,npltmp-nnstmp
   zray(p,n,d)%plnfr = n
   zray(p,n,d)%trcfr = pp
   pp = pp + 1
END DO
DO p = npltmp-nnstmp+1,npltmp
   zray(p,n,d)%plnfr = nfr
   zray(p,n,d)%trcfr = pp
   pp = pp + 1
END DO
END DO
!
! Correct for Infinitesimally Small Track Lengths
!
DO d = 1,dirnum
   naztmp = dirset(d)%nnx + dirset(d)%nny
   DO n = 1,naztmp
      npltmp = npl(n,d)
      DO p = 1,npltmp
         ttmp = zray(p,n,d)%trcln
         IF (ttmp.lt.trkmin) THEN
            zray(p,n,d)%trcln = trkmin
         ENDIF
      END DO
   END DO
END DO
!
! reset NPOMAX to maximum value in NPL array
!
npomax = 0
DO d = 1,dirnum
   naztmp = dirset(d)%nnx + dirset(d)%nny
   DO n = 1,naztmp
      IF (npl(n,d).gt.npomax) THEN
         npomax = npl(n,d)
      ENDIF
   END DO
END DO
!
DEALLOCATE(offset,tracks)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RETURN
END

SUBROUTINE getNPC(plin,plout,d,ntr,npx)
!
!!!!!!!!!!!!
USE rays
!!!!!!!!!!!!
! INTEGER plin  ! first plane in sequence
INTEGER plout ! last plane in sequence
INTEGER cpl  ! current plane
INTEGER ntr  ! # planes in unit
INTEGER npx  ! # plane crossings
INTEGER d    ! direction index
INTEGER p    ! plane index

! determine number of planes crossed
! by track in polar meta-node
!
IF (plin.eq.plout) THEN
  npx = 1
ELSE
  cpl = plin
  npx = 1
  DO p = 1,ntr
    IF (cpl.eq.plout) THEN
      EXIT
    ELSE
      cpl = xyray(cpl,d)%trcto
      npx = npx + 1
      IF (cpl.eq.plin) THEN
        STOP
      ENDIF
    ENDIF
  END DO
ENDIF
RETURN
END

SUBROUTINE dbg3D(dir,dstmp,nnstmp,nnztmp,ins,inz,tracks)
!
USE files
USE keys
!
DOUBLE PRECISION dir   !
DOUBLE PRECISION dstmp !
INTEGER          nnstmp !
INTEGER          nnztmp !
INTEGER          n      !
DOUBLE PRECISION, DIMENSION(nnstmp) :: ins   !
DOUBLE PRECISION, DIMENSION(nnztmp) :: inz   !
DOUBLE PRECISION, DIMENSION(nnstmp+nnztmp) :: tracks !
!
WRITE (dfile,10)
WRITE (dfile,100) dir
WRITE (dfile,105) dstmp
WRITE (dfile,10)
WRITE (dfile,110)
WRITE (dfile,111)
DO n = 1,nnstmp
   WRITE(dfile,115) n,ins(n)
END DO
WRITE (dfile,10)
WRITE (dfile,120)
WRITE (dfile,121)
DO n = 1,nnztmp
   WRITE (dfile,125) n,inz(n)
END DO
WRITE (dfile,10)
WRITE (dfile,130)
WRITE (dfile,131)
DO n = 1,nnstmp+nnztmp
   WRITE (dfile,135) n,tracks(n)
END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Format Statements
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
10 FORMAT(1x)
100 FORMAT(4x,'Direction #',i2)
105 FORMAT(4x,'dS Cell Dimension = ',es13.6)
110 FORMAT(5x,'INS#     Location [cm]')
111 FORMAT(4x,'------   ---------------')
115 FORMAT(6x,i2,6x,es13.6)
120 FORMAT(5x,'INZ#     Location [cm]')
121 FORMAT(4x,'------   ---------------')
125 FORMAT(6x,i2,6x,es13.6)
130 FORMAT(5x,'TR#     Length [cm]')
131 FORMAT(3x,'------    --------------')
135 FORMAT(6x,i2,6x,es13.6,3x)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END

SUBROUTINE writeRays3D()  
!
!!!!!!!!!!!!!!!!!!!!
USE files
USE keys
USE directions
USE rays
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER naztmp ! # Planes in Cell
INTEGER npltmp ! # Tracks in Plane
INTEGER nnstmp ! # Entries: -s Edge
!!!!!!!!!!!!!!!!!!!!
INTEGER d ! Direction Index
INTEGER p ! Track Index
INTEGER n ! Plane Index
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,10)
WRITE (ofile,20)
WRITE (ofile,21)
WRITE (ofile,22)
WRITE (ofile,23)
WRITE (ofile,24)
WRITE (ofile,25)
WRITE (ofile,26)
WRITE (ofile,10)
DO d = 1,dirnum
   nns = dirset(d)$nns
   nnx = dirset(d)$nnx + dirset(d)$nny
   WRITE (ofile,100) d
   WRITE (ofile,102)
END DO
WRITE (ofile,104)
DO n = 1,nns
   pln = npl(n,d)
   DO p = 1,pln
      WRITE (ofile,110) n,p,zr(p,n,d)$trcln, &
      zr(p,n,d)$plnto,zr(p,n,d)$trcto, &
      zr(p,n,d)$plnfr,zr(p,n,d)$trcfr
   END DO
END DO
WRITE (ofile,10)
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Format Statements
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
10 FORMAT(1x)
20 FORMAT(3x,'**********************************')
21 FORMAT(3x,'*o-------------------------------o*')
22 FORMAT(3x,'*|                              |*')
23 FORMAT(3x,'*|  3D Ray Tracing Information  |*')
24 FORMAT(3x,'*|                              |*')
25 FORMAT(3x,'*o-------------------------------o*')
26 FORMAT(3x,'**********************************')
100 FORMAT(3x,'Direction #',i2)
102 FORMAT(4x,'Pl#',5x,'Tr#',5x,'Length [cm]',4x,'Pl# To', &
   & 3x,'Tr# To',4x,'Pl# Fr',3x,'Tr# Fr')
104 FORMAT(3x,'----- ----- --------------- ----- ', &
   & '------ ------ ------')
110 FORMAT(3x,i3,5x,i3,5x,es13.6,5x,i3,6x,i3,5x,i3,6x,i3)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END

SUBROUTINE writeBalChk()
!
!!!!!!!!!!!!!!!!!
USE files
USE keys
USE cells
USE directions
USE rays
!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION dazm,dpolar ! Azi/Pol Track Separation
DOUBLE PRECISION trsum    ! Total Track Length
DOUBLE PRECISION area1,area2 ! 2D Cell Area
DOUBLE PRECISION vol1,vol2 ! 3D Cell Volume
DOUBLE PRECISION pctd    ! % Difference

316
INTEGER naztmp ! # Planes in Cell
INTEGER npltmp ! # Tracks in Plane
INTEGER d,n,p  !

! 2D Balance Check
!

WRITE(dfile,10)
WRITE(dfile,20)
WRITE(dfile,21)
WRITE(dfile,22)
WRITE(dfile,10)
WRITE(dfile,40)
WRITE(dfile,41)
WRITE(dfile,42)
DO d = 1,dirnum
  naztmp = dirset(d)%nnx + dirset(d)%nny
dazim = dirset(d)%dazi
trsum = 0.0
  DO n = 1,naztmp
    trsum = trsum + xyray(n,d)%trcln
  END DO
area1 = trsum * dazim
area2 = dx * dy
pctd = (area2 - area1) / area2
WRITE(dfile,50) d,area1,area2,pctd
END DO

! 3D Balance Check
!

WRITE(dfile,10)
WRITE(dfile,120)
WRITE(dfile,121)
WRITE(dfile,122)
WRITE(dfile,10)
WRITE(dfile,140)
WRITE(dfile,141)
WRITE(dfile,142)
DO d = 1,dirnum
  naztmp = dirset(d)%nnx + dirset(d)%nny
dpolar = dirset(d)%dpol
  DO n = 1,naztmp
    npltmp = npl(n,d)
    DO p = 1,npltmp
      trsum = trsum + zray(p,n,d)%trcln
    END DO
vol1 = trsum * dpolar
vol2 = xyray(n,d)%trcln * dz
pctd = (vol2 - vol1) / vol2
WRITE(dfile,150) d,n,vol1,vol2,pctd
END DO
END DO

! Format Statements
!

10 FORMAT(1x)
SUBROUTINE writeFinal()
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE(afile,*) ngeo, dirnum
IF (ngeo.eq.2) THEN
  WRITE(afile,*) dx, dy
  WRITE(afile,*) nazmax
  DO d = 1,dirnum
    WRITE(afile,*) dirset(d)
    DO n = 1,nazmax
      WRITE(afile,*) xyray(n,d)
    END DO
  END DO
ELSE
  WRITE(afile,*) dx, dy, dz
  WRITE(afile,*) nazmax, npomax
  DO d = 1,dirnum
    WRITE(afile,*) dirset(d)
    DO n = 1,nazmax
      WRITE(afile,*) npl(n,d)
      DO p = 1,npomax
        WRITE(afile,*) zray(p,n,d)
      END DO
    END DO
  END DO
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END

SUBROUTINE writeFinal()
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE(afile,*) ngeo, dirnum
IF (ngeo.eq.2) THEN
  WRITE(afile,*) dx, dy
  WRITE(afile,*) nazmax
  DO d = 1,dirnum
    WRITE(afile,*) dirset(d)
    DO n = 1,nazmax
      WRITE(afile,*) xyray(n,d)
    END DO
  END DO
ELSE
  WRITE(afile,*) dx, dy, dz
  WRITE(afile,*) nazmax, npomax
  DO d = 1,dirnum
    WRITE(afile,*) dirset(d)
    DO n = 1,nazmax
      WRITE(afile,*) npl(n,d)
      DO p = 1,npomax
        WRITE(afile,*) zray(p,n,d)
      END DO
    END DO
  END DO
ENDIF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END
SUBROUTINE closeFiles()
!
!!!!!!!!!!!!!!
USE files
!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!
CLOSE (ifile)
CLOSE (ofile)
CLOSE (afile)
CLOSE (dfile)
!!!!!!!!!!!!!!!!
!
END

MODULE cells
!
! DX, DY, and DZ --> Scalar Cell Dimensions
!
!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION dx!
DOUBLE PRECISION dy!
DOUBLE PRECISION dz!
!!!!!!!!!!!!!!!!!!!!!!!
!
END MODULE cells

MODULE directions
!
! -------------------
! DEFINED DATA TYPE
! -------------------
!
! Data Type Specification for Quadrature Array
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TYPE QUADINFO!
   DOUBLE PRECISION wght !......Direction Weight
   DOUBLE PRECISION azim !......Azimuthal Angle
   DOUBLE PRECISION dazi !......dA track spacing
   INTEGER nnx !......# Spans in x-dir
   INTEGER nny !......# Spans in y-dir
   INTEGER npu !.....# Azi Periodic Units
   DOUBLE PRECISION pola !......Polar Angle
   DOUBLE PRECISION dpol !......dP track spacing
   INTEGER nns !....# Spans in (xy)-dir
   INTEGER nnz !......# Spans in z-dir
END TYPE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! DIRSET is the working array for holding
! all problem quadrature data
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TYPE(QUADINFO), ALLOCATABLE :: dirset(:), !


! Level Symmetric Quadrature Specification

********************************************
!* Level Symmetric Quadrature Specification *
********************************************

!!!!!!!!!!!!!!!!!!!
INTEGER lev2    !
INTEGER lev4    !
INTEGER lev6    !
INTEGER lev8    !
INTEGER lev12   !
INTEGER lev16   !
!!!!!!!!!!!!!!!!!!!
INTEGER g       !
!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION azidata(36,6)
DOUBLE PRECISION poldata(36,6)
DOUBLE PRECISION wgtdata(36,6)
!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!  Angles & Weights derived from                 !
!!  Level Symmetric Direction Cosines and Weights !
!!  in E. E. Lewis & W. F. Miller, Jr.            !
!!  Computational Methods of Neutron Transport    !
!!  ANS, Inc. La Grange Park, Illinois USA        !
!!  Figure 4-1, p.157; Figure 4-2, p.159;         !
!!  Figure 4-3, p.161; and Table 4-1, p.162       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!  S2 Level-Symmetric Quadrature
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DATA lev2/1/
DATA (azidata(g,1),g=1,36) /0.7853982,                        35*0.0/
DATA (poldata(g,1),g=1,36) /0.7853982,                        35*0.0/
DATA (wgtdata(g,1),g=1,36) /0.1250000,                        35*0.0/
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!  S4 Level-Symmetric Quadrature
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DATA lev4/2/
DATA (azidata(g,2),g=1,36) /0.7853982,0.3829497,1.1878466,    33*0.0/
DATA (poldata(g,2),g=1,36) /0.5178402,1.2132026,1.2132026,    33*0.0/
DATA (wgtdata(g,2),g=1,36) /0.04166666,0.04166666,0.04166666, 33*0.0/
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!  S6 Level-Symmetric Quadrature
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DATA lev6/3/
DATA (azidata(g,3),g=1,36) /0.7853982,0.3729351,1.1978613,          &
& 0.2803073,0.7853982,1.2904890, 30*0.0/
DATA (poldata(g,3),g=1,36) /0.3866412,0.8209756,0.8209756,          &
& 1.3008959,1.3008959,1.3008959, 30*0.0/
DATA (wgtdata(g,3),g=1,36) /0.02201579,0.01965089,0.01965089,       &
& 0.02201579,0.01965089,0.02201579, 30*0.0/
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!  S8 Level-Symmetric Quadrature
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DATA lev8/4/
DATA (azidata(g,4),g=1,36) /0.7853982,0.3613671,1.2094292,          &
& 0.2705498,0.7853982,1.3002466, 26*0.0/
DATA (poldata(g,4),g=1,36) /0.3137280,0.6651960,0.6651960,          &
& 1.3452829, 26*0.0/
#### S12 Level-Symmetric Quadrature

<table>
<thead>
<tr>
<th>azidata(g,5)</th>
<th>poldata(g,5)</th>
<th>wgtdata(g,4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g=1, 36</td>
<td>g=1, 36</td>
<td>g=1, 36</td>
</tr>
<tr>
<td>0.7853982, 0.3489716, 1.0222827</td>
<td>0.7853982, 1.0222827, 0.3489716</td>
<td>0.01512346, 0.01134259, 0.01134259</td>
</tr>
<tr>
<td>0.2602171, 0.7853982, 1.3105792</td>
<td>0.2602171, 0.7853982, 1.3105792</td>
<td>0.01512346, 0.01134259, 0.01134259</td>
</tr>
</tbody>
</table>

#### S16 Level-Symmetric Quadrature

<table>
<thead>
<tr>
<th>azidata(g,6)</th>
<th>poldata(g,6)</th>
<th>wgtdata(g,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g=1, 36</td>
<td>g=1, 36</td>
<td>g=1, 36</td>
</tr>
<tr>
<td>0.7853982, 0.3404295, 1.02303668</td>
<td>0.7853982, 1.02303668, 0.3404295</td>
<td>0.00612340, 0.00516620, 0.00516620</td>
</tr>
<tr>
<td>0.2531673, 0.7853982, 1.3176290</td>
<td>0.2531673, 0.7853982, 1.3176290</td>
<td>0.00612340, 0.00516620, 0.00516620</td>
</tr>
</tbody>
</table>
END MODULE directions

MODULE files

! INPUT File [UNIT = 1, NAME = input.ray]
!
! ...contains input information
! required for ray tracing calculation
! 
! must be specified by user prior to execution
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: ifile = 1                   !
CHARACTER(9), PARAMETER :: ifname = 'input.ray'   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! OUTPUT File [UNIT = 2, NAME = output.ray]
!
! ...contains full annotated output
! from ray tracing calculation
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: ofile = 2                   !
CHARACTER(10), PARAMETER :: ofname = 'output.ray' !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! DATA File [UNIT = 3, NAME = data.ray]
!
! ...contains ray tracing results that are required
! as input for the MOCK-3D angular flux calculation
! stored in unformatted but compact form
! i.e, no annotation - just the raw data
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: afile = 3                   !
CHARACTER(8), PARAMETER :: afname = 'data.ray'    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! DEBUG File [UNIT = 4, NAME = debug.ray]
!
! ...used to report additional information
! for debugging and/or sensitivity studies
! 
! if you need to monitor additional parameters
! it is recommended that you write them to this file
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: dfile = 4                   !
CHARACTER(9), PARAMETER :: dfname = 'debug.ray'   !
SHELL File [UNIT = 5, NAME = shell.ray]
...used to send ray tracing info to the
TortoiseShell Java Visualization Tool
SHELL.RAY must be in BINARY format
(specified when file is opened)

INTEGER, PARAMETER :: sfile = 5
CHARACTER(9), PARAMETER :: sfname = 'shell.ray'

END MODULE files

MODULE keys

! NGEO --> Flag: Problem Geometry Type
! = 2 --> 2D Cartesian Geometry
! = 3 --> 3D Cartesian Geometry

INTEGER ngeo !

! NSN --> Flag: Sn Quadrature Type
! possible choices are 4, 6, 8, 12, and 16
! for S4, S6, S8, usw.

INTEGER nsn !

! DIRNUM -> Number of Directions in Quadrature Set
! set during problem initialization (genModel.f)
! used frequently as an iteration limit
!
! NOTE - this implies a single iteration on DIRECTION
! rather than a nested iteration on AZIMUTHAL/POLAR ANGLE
! in all ray-tracing processes and in output data
!
INTEGER dirnum !

! Pi and her cousins (to 8 significant figures)
!
DOUBLE PRECISION, PARAMETER ::   pi = 3.1415927 !  Pi
DOUBLE PRECISION, PARAMETER ::  hpi = 1.57079633  !  Pi/2
DOUBLE PRECISION, PARAMETER ::  tpi = 6.2831853   !  2*Pi
DOUBLE PRECISION, PARAMETER ::  fpi = 12.5663706  !  4*Pi
DOUBLE PRECISION, PARAMETER :: pirt = 1.77245385  !  Root-Pi

! Minimum Threshold Value for Track Length
! (required to avoid numerical problems due to very small tracks)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, PARAMETER :: trkmin = 1.0E-06 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Prime Number Array and Limits
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, ALLOCATABLE :: prime(:) !
INTEGER, PARAMETER :: nprime = 10001 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END MODULE keys

MODULE rays
!
!-------------------
! DEFINED DATA TYPE
!-------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TYPE AZIRAYINFO          
    DOUBLE PRECISION trcln ! Track Length
    INTEGER trcto ! Track TO which Track Couples
    INTEGER trcfr ! Track FROM which Track Couples
END TYPE                 
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! XYRAY is the variable for Azimuthal (2D) Ray Tracing
!
...track length [%TRCLN] represents the PARTIAL
! path length --> in 2D, polar component is applied
! during outer iteration
!
! shape: (#Azi, #Dir)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TYPE(AZIRAYINFO), ALLOCATABLE :: xyray(:,,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TYPE POLRAYINFO          
    DOUBLE PRECISION trcln ! Track Length
    INTEGER plnto ! Plane TO which Track Couples
    INTEGER trcto ! Track TO which Track Couples
    INTEGER plnfr ! Plane FROM which Track Couples
    INTEGER trcfr ! Track FROM which Track Couples
END TYPE                 
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! ZRAY is the variable holds 3D Track Lengths
! & Coupling Matrix
!
! shape: (#Pol, #Azi, #Dir)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TYPE(POLRAYINFO), ALLOCATABLE :: zray(:,:, :) !
TYPE(POLRAYINFO), ALLOCATABLE :: zrcopy(:,:, :) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! NPL
!     shape: (#Azi, #Dir)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    INTEGER, ALLOCATABLE :: npl(:,,:)!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Max Limits for Sizing Arrays...
!
!!!!!!!!!!!!!!!!!
    INTEGER nazmax  !  Maximum # Tracks - Azimuthal Plane
    INTEGER npomax  !  Maximum # Tracks - Polar Plane
!!!!!!!!!!!!!!!!!!!!
!
END MODULE rays

B.2 Source Code for MOCK-3D

PROGRAM MOCK3D
!
!!!!!!!!!!!!!!!!!!!!!!
    CALL mdaemon() !
!!!!!!!!!!!!!!!!!!!!!!
!
STOP
!
END PROGRAM

SUBROUTINE mdaemon()
!
!!!!!!!!!!!!!!!!!!!!!!
    USE files   !
    USE scalars !
!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!
    ! Build System Model!
!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!
    OPEN (UNIT = ifile, FILE = ifname)
    OPEN (UNIT = afile, FILE = afname)
    CALL genModelSS()
    CLOSE (ifile)
    CLOSE (afile)
!
!!!!!!!!!!!!!!!!!!!!!!!
    !
    ! Problem Initialization!
!!!!!!!!!!!!!!!!!!!!!!!
!
    IF (rstflg.eq.0) THEN
!
SUBROUTINE genModelSS()
!
!!!!!!!!!!!!
USE files   !
USE scalars !
USE tracking!
USE xsects  !
!!!!!!!!!!!!
!
! GENMODELSS loads all input and initializes the problem geometry!
!
! Local Indices
!
!!!!!!!!!!!!!!!!!!!!
INTEGER g, h       ! Energy Group Indices
INTEGER d          ! Direction Index
!
INTEGER i, ii, iii ! Cell Indices
INTEGER n ! Plane Index
INTEGER p ! Track Index
INTEGER q ! XS Set Index
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Load Problem Control Flags !
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
READ(ifile,*) rstflg
READ(ifile,*) nmflag, dssflg, dtrflg
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Load Tracking File Data !
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
READ(afile,*) ngeo, dirnum
ALLOCATE(dirset(dirnum))
READ(afile,*) dx, dy, dz
READ(afile,*) nazmax,npomax
ALLOCATE(zray(npomax,nazmax,dirnum))
ALLOCATE(npl(nazmax,dirnum))
DO d = 1,dirnum
  READ(afile,*) dirset(d)
  DO n = 1,nazmax
    READ(afile,*) npl(n,d)
    DO p = 1,npomax
      READ(afile,*) zray(p,n,d)
    END DO
  END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Load Cell and Boundary Info !
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
READ (ifile,*) nx, ny, nz
READ (ifile,*) alb1, alb2, alb3, alb4, alb5, alb6
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Load Iteration Control Parameters !
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
READ (ifile,*) cck
READ (ifile,*) cca, ccs
READ (ifile,*) iteromax, iterimax
READ (ifile,*) rstnum, lngnum
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Load # Energy Groups & Cross-Section Sets !
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
READ (ifile,*) ng, nxsets
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  !  ! Load Cross-Section Data !
!  !  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (xstr(ng,nxsets))
ALLOCATE (nxsf(ng,nxsets))
ALLOCATE (xssc(ng,ng,nxsets))
ALLOCATE (xsab(ng,nxsets))
ALLOCATE (chip(ng))
READ (ifile,*) (chip(g),g=1,ng)
DO q = 1,nxsets
  DO g = 1,ng
    READ (ifile,*) xstr(g,q),nxsf(g,q),xsab(g,q)
    READ (ifile,*) (xssc(h,g,q),h=1,ng)
  END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  !  ! Load Cell/XS Map Data !
!  !  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (xsmap(nz,ny,nx))
DO iii = 1,nz
  DO ii = ny,1,-1
    READ (ifile,*) (xsmap(iii,ii,i), i=1,nx)
  END DO
END DO
!
RETURN
END
WRITE (ofile,10) !
WRITE (ofile,30) !
WRITE (ofile,31) !
WRITE (ofile,32) !
WRITE (ofile,40) !
WRITE (ofile,32) !
WRITE (ofile,31) ! Write Geometry Data
WRITE (ofile,30) !
WRITE (ofile,10) !
WRITE (ofile,100) 'GEOMETRY TYPE: ngeo = ',ngeo  !
WRITE (ofile,*) '   3D Cell Geometry' !
WRITE (ofile,10) !
WRITE (ofile,*) '   CELL DIMENSIONS:' !
WRITE (ofile,105) 'Cell x-Dimension = ',dx       !
WRITE (ofile,105) 'Cell y-Dimension = ',dy       !
WRITE (ofile,105) 'Cell z-Dimension = ',dz       !
WRITE (ofile,10) !
WRITE (ofile,*) '   SYSTEM DIMENSIONS:'          !
WRITE (ofile,110) '# Cells in x-direction = ',nx !
WRITE (ofile,110) '# Cells in y-direction = ',ny !
WRITE (ofile,110) '# Cells in z-direction = ',nz !
WRITE (ofile,10) !
WRITE (ofile,*) '   ALBEDO BOUNDARY CONDITIONS:' !
WRITE (ofile,115) '-x face = ',alb1              !
WRITE (ofile,115) '+x face = ',alb2              !
WRITE (ofile,115) '-y face = ',alb3              !
WRITE (ofile,115) '+y face = ',alb4              !
WRITE (ofile,115) '-z face = ',alb5              !
WRITE (ofile,115) '+z face = ',alb6              !
WRITE (ofile,10) !
WRITE (ofile,30) !
WRITE (ofile,31) !
WRITE (ofile,32) !
WRITE (ofile,50) !
WRITE (ofile,32) !
WRITE (ofile,31) ! Write Quadrature Data
WRITE (ofile,30) !
WRITE (ofile,10) !
DO d = 1,dirnum                                                  !
WRITE (ofile,200) d,dirnum                                     !
WRITE (ofile,10)                                               !
WRITE (ofile,205) dirset(d)%wght                               !
WRITE (ofile,210) '  Azimuthal Angle = ',dirset(d)%azim,' rad' !
WRITE (ofile,210) 'Azimuthal Spacing = ',dirset(d)%dazi,' cm ' !
WRITE (ofile,215) 'NNX = ', dirset(d)%nnx                      !
WRITE (ofile,215) 'NNY = ', dirset(d)%nny                      !
WRITE (ofile,210) '   Polar Angle = ',dirset(d)%pola,' rad' !
WRITE (ofile,210) '    Polar Spacing = ',dirset(d)%dpol,' cm ' !
WRITE (ofile,215) 'NNS = ', dirset(d)%nns                      !
WRITE (ofile,215) 'NNZ = ', dirset(d)%nnz                      !
WRITE (ofile,10)                                               !
END DO                                                           !
WRITE (ofile,30) !
WRITE (ofile,31) !
WRITE (ofile,32) !
WRITE (ofile,32) !
WRITE (ofile,31) ! Write 3D Ray Tracing Data
WRITE (ofile,30) !
WRITE (ofile,10) !
DO d = 1,dirnum                                                  !
nnstmp = dirset(d)%nns
naztmp = dirset(d)%nnx + dirset(d)%nny!
WRITE (ofile,700) d!
WRITE (ofile,702)!
WRITE (ofile,704)!
DO n = 1,naztmp!
  npltmp = npl(n,d)!
  DO p = 1,npltmp!
    WRITE (ofile,710) n,p,zray(p,n,d)%trcln, &
      & zray(p,n,d)%plnto,zray(p,n,d)%trcto, &
      & zray(p,n,d)%plnfr,zray(p,n,d)%trcfr!
  END DO!
END DO!
WRITE (ofile,10)!
END DO!
WRITE (ofile,30) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,31)!
WRITE (ofile,32)!
WRITE (ofile,80)!
WRITE (ofile,32)!
WRITE (ofile,31) ! Write Material Properties
WRITE (ofile,30)!
WRITE (ofile,10) ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,500) 'Number Energy Groups: ',ng
WRITE (ofile,500) 'Number of XS Sets:    ',nxsets
WRITE (ofile,10)
WRITE (ofile,*)! Chip: Group Value'
WRITE (ofile,*) '! ----- --------------'
DO g = 1,ng
  WRITE (ofile,505) g,chip(g)
END DO
WRITE (ofile,10)
WRITE (ofile,10)
DO q = 1,nxsets
  WRITE (ofile,500) 'Cross Section Set # ',q
WRITE (ofile,10)
WRITE (ofile,*) '! Sigma-tr: Group Value'
WRITE (ofile,*) '! ----- --------------'
DO g = 1,ng
  WRITE (ofile,505) g,xstr(g,q)
END DO
WRITE (ofile,10)
WRITE (ofile,*) '! nuSigma-fiss: Group Value'
WRITE (ofile,*) '! ----- --------------'
DO g = 1,ng
  WRITE (ofile,505) g,nxsf(g,q)
END DO
WRITE (ofile,10)
WRITE (ofile,*) '! Sigma-abs: Group Value'
WRITE (ofile,*) '! ----- --------------'
DO g = 1,ng
  WRITE (ofile,505) g,xsab(g,q)
END DO
WRITE (ofile,10)
WRITE (ofile,*) '! Sigma-scat: Group to Group Value'
WRITE (ofile,*) '! ----- ----- --------------'
DO g = 1,ng
  DO h = 1,ng
    WRITE (ofile,510) g,h,xssc(h,g,q)
  END DO
END DO
WRITE (ofile,10)
WRITE (ofile,*) 'Cell/XS Map:'
WRITE (ofile,10)
DO iii = 1,nz
    WRITE (ofile,111) 'Level # ',iii
    DO ii = ny,1,-1
        WRITE (ofile,*) (xsmap(iii,ii,i), i=1,nx)
    END DO
END DO
WRITE (ofile,10)
WRITE (ofile,30) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,31) ! Write Problem Control Information
WRITE (ofile,30) !
WRITE (ofile,10) !
WRITE (ofile,800) 'Restart Flag = ',rstflg
WRITE (ofile,10)
WRITE (ofile,805) 'Post-Processing Flux Normalization Flag = ',nmflag
IF (nmflag.eq.0) THEN
    WRITE (ofile,*) ' No Flux Normalization Will Be Performed'
ELSE IF (nmflag.eq.1) THEN
    WRITE (ofile,*) ' Flux Will Be Normalized to One Production'
ELSE IF (nmflag.eq.2) THEN
    WRITE (ofile,*) ' Flux Will Be Normalized to One Absorption'
ELSE
    WRITE(ofile,*) 'Cell-Averaged Flux Normalization Will Be Performed'
ENDIF
WRITE (ofile,10)
WRITE (ofile,810) 'Restart Dump Flag = ',dssflg
IF (dssflg.eq.0) THEN
    WRITE (ofile,*) 'Steady-State Restart Dump WILL NOT Be Generated'
ELSE
    WRITE (ofile,*) ' Steady-State Restart Dump WILL Be Generated'
ENDIF
WRITE (ofile,10)
WRITE (ofile,815) 'Transient Dump Flag = ',dtrflg
IF (dtrflg.eq.0) THEN
    WRITE(ofile,*)'Transient Initialization Dump WILL NOT Be Generated'
ELSE
    WRITE (ofile,*) 'Transient Initialization Dump WILL Be Generated'
ENDIF
WRITE (ofile,10)
WRITE (ofile,*) '   Scalar Flux      Angular Flux     Max #Inner'
WRITE (ofile,*) '   Convergence      Convergence      Iterations'
WRITE (ofile,*) '  --------------   --------------   -------------'
WRITE (ofile,820) ccs,cca,iterimax
WRITE (ofile,10)
WRITE (ofile, *) '      Keff          Max #Outer  '
WRITE (ofile,*) '   Convergence      Iterations  '
WRITE (ofile,*) '  --------------   -------------'
WRITE (ofile,825) cck,iteromax
WRITE (ofile,10)
WRITE (ofile,830) '# Outers Between Restart Dumps = ',rstnum
WRITE (ofile,830) '# Outers Between Long Outputs  = ',lngnum
WRITE (ofile,10)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Format Statements
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
10 FORMAT(1x)
SUBROUTINE init3Dss()

!!!!!!!!!!!!!!!!!!!!!!!
USE files    
USE scalars  
USE tracking 
USE xsects   
USE neutrons !
!!!!!!!!!!!!!!!!!!!!!!!
!
! INIT3DSS initializes the 3D steady-state MoC calculation
!
! Local Parameters
!
!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION cross     ! Macroscopic Cross-Sections - Transport
DOUBLE PRECISION cross1    ! Macroscopic Cross-Sections - nuFission
DOUBLE PRECISION cross2    ! Macroscopic Cross-Sections - Abs
DOUBLE PRECISION track     ! Track Length
DOUBLE PRECISION fnorm     ! Normalization Factor
DOUBLE PRECISION phi       ! Scalar Flux
DOUBLE PRECISION fdtmp     ! Fission Rate
DOUBLE PRECISION kfiss, kabs ! Numerator/Denominator - Kinf

END
DOUBLE PRECISION scat, fiss ! Source Term Components
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!
INTEGER nset ! Cross-Section Set #
INTEGER nnxtmp ! # Plane Entries along +-x Edge of Azimuthal Cell
INTEGER nnytmp ! # Plane Entries along +-y Edge of Azimuthal Cell
INTEGER naztmp ! # Planes in 3D Cell
INTEGER npltmp ! # Tracks in Polar Planar Section
!!!!!!!!!!!!!!!
!
! Local Indices
!
!!!!!!!!!!!!!!!!!!!!
INTEGER g, h ! Energy Group Indices
INTEGER d,dd,dq ! Direction Indices
INTEGER i,ii,iii ! Spatial Indices
INTEGER n ! Plane Index
INTEGER p ! Track Index
INTEGER q ! XS Set Index
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!
! Allocate Global Arrays !
!!!!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (ncp(npomax,nazmax,nxsets,dirnum,ng))
ALLOCATE (aphi(npomax,nazmax,nz,ny,nx,8*dirnum,ng))
ALLOCATE (sphi(nz,ny,nx,ng))
ALLOCATE (ssav(nz,ny,nx,ng))
ALLOCATE (fsav(nz,ny,nx,ng))
ALLOCATE (fd0(nz,ny,nx))
ALLOCATE (fd1(nz,ny,nx))
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Tabulate Non-Collision Probabilities !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
  DO d = 1,dirnum
    nnxtmp = dirset(d)%nnx
    nnytmp = dirset(d)%nny
    naztmp = nnxtmp + nnytmp
    DO q = 1,nxsets
      cross = xstr(g,q)
      DO n = 1,naztmp
        npltmp = npl(n,d)
        DO p = 1,npltmp
          track = zray(p,n,d)%trcln
          ncp(p,n,q,d,g) = exp(-cross * track)
        END DO
      END DO
    END DO
  END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Initialize Angular Flux Distribution !
DO g = 1, ng
  dd = 0
  DO dq = 1, 8
    DO d = 1, dirnum
      dd = dd + l
      DO i = 1, nx
        DO ii = 1, ny
          DO iii = 1, nz
            DO n = 1, nazmax
              DO p = 1, npomax
                aphi(p, n, iii, ii, i, dd, g) = 1.0 / fpi
              END DO
            END DO
          END DO
        END DO
      END DO
    END DO
  END DO
END DO

END DO

! Initialize Scalar Flux!

DO g = 1, ng
  DO i = 1, nx
    DO ii = 1, ny
      DO iii = 1, nz
        sphi(iii, ii, i, g) = 1.0
      END DO
    END DO
  END DO
END DO

fnorm = 0.0
DO g = 1, ng
  DO i = 1, nx
    DO ii = 1, ny
      DO iii = 1, nz
        nset = xsmap(iii, ii, i)
        cross1 = nxsf(g, nset)
        phi = sphi(iii, ii, i, g)
        fnorm = fnorm + cross1 * phi * dx * dy * dz
      END DO
    END DO
  END DO
END DO

END DO

END DO

!
! Initialize Fission Rate Parameters
!
!
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
  nset = xsmap(iii,ii,i)
  fdtmp = 0.0
  DO g = 1,ng
    cross1 = nxsf(g,nset)
    phi = sphi(iii,ii,i,g)
    fdtmp = fdtmp + phi * cross1
  END DO
  fd0(iii,ii,i) = fdtmp
  fd1(iii,ii,i) = fdtmp
END DO
END DO
END DO
!
!
Calculate Initial k-eff
!
!
DO g = 1,ng
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
  nset = xsmap(iii,ii,i)
  cross1 = nxsf(g,nset)
  cross2 = xsab(g,nset)
  kfiss = kfiss + cross1 * dx * dy * dz
  kabs = kabs + cross2 * dx * dy * dz
END DO
END DO
END DO

kinf = kfiss / kabs
keff = kinf
!
!
Calculate Initial Source Terms
!
!
DO g = 1,ng
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
  scat = 0.0
  fiss = 0.0
  nset = xsmap(iii,ii,i)
  DO h = 1,ng
    scat = scat + xssc(g,h,nset) * sphi(iii,ii,i,h)
fiss = fiss + nxsf(h,nset) * sphi(iii,ii,i,h)
END DO
ssav(iii,ii,i,g) = scat / fpi
fsav(iii,ii,i,g) = (fiss / fpi) * (chip(g) / keff)
END DO
END DO
END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!
! Initialize Output!
!!!!!!!!!!!!!!!!!!!!!!!!!
!
WRITE (ofile,*(' '))
WRITE (ofile,*('  iter       keff          cnv k      # inners'))
WRITE (ofile,*(' ------  ------------  ------------  ----------'))
WRITE (ofile,15) 0,keff,' -',' - '
WRITE (*,*)
WRITE (*,*)
!
RETURN
END

SUBROUTINE rstLoadSS()
!
!!!!!!!!!!!!!!!!!!!!
USE files      !
USE scalars    !
USE tracking   !
USE neutrons   !
USE xsects     !
!!!!!!!!!!!!!!!!!!!!
!
! Local Parameters!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION phi        ! Scalar Flux
DOUBLE PRECISION scat, fiss ! Source Term Components
DOUBLE PRECISION chitmp     !
DOUBLE PRECISION cross      !
DOUBLE PRECISION track      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER nset   ! Cross-Section Set #
INTEGER nnxtmp ! # Plane Entries along +x Edge of Azimuthal Cell
INTEGER nnytmp ! # Plane Entries along +y Edge of Azimuthal Cell
INTEGER naztmp ! # Planes in 3D Cell
INTEGER npltmp ! # Tracks in Polar Planar Section
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Local Indices
!

! INTEGER g, h      ! Energy Group Indices
! INTEGER d, dq, dd ! Direction Index
! INTEGER i, ii, iii ! Spatial Indices
! INTEGER n       ! Plane Index
! INTEGER p       ! Track Index
! INTEGER q       ! XS Set Index
!
! ALLOCATE (ncp(npomax,nazmax,nxsets,dirnum,ng))
! ALLOCATE (aphi(npomax,nazmax,nz,ny,nx,8*dirnum,ng))
! ALLOCATE (sphi(nz,ny,nx,ng))
! ALLOCATE (ssav(nz,ny,nx,ng))
! ALLOCATE (fsav(nz,ny,nx,ng))
! ALLOCATE (fd0(nz,ny,nx))
! ALLOCATE (fd1(nz,ny,nx))
!
! OPEN (UNIT = rfile, FILE = rfname, FORM = 'BINARY')
!
! READ(rfile) iterout,kinf,keff
! READ(rfile) sphi
! READ(rfile) fd0
! READ(rfile) fd1
! READ(rfile) aphi
!
! CLOSE (rfile)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Rebuild Scattering Source!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      scat = 0.0
      nset = xsmap(iii,ii,i)
      DO h = 1,ng
        phi = sphi(iii,ii,i,h)
        scat = scat + xssc(g,h,nset) * phi
      END DO
      ssav(iii,ii,i,g) = scat / fpi
    END DO
  END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Rebuild Fission Source!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
  chitmp = chip(g)
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        fiss = 0.0
        nset = xsmap(iii,ii,i)
        DO h = 1,ng
          cross = nxsf(h,nset)
          phi = sphi(iii,ii,i,h)
          fiss = fiss + xssf(g,h,nset) * phi
        END DO
        fsav(iii,ii,i,g) = fiss / fpi
      END DO
    END DO
  END DO
END DO
SUBROUTINE powitSSout()
!
!!!!!!!!!!!!!!!!!!!!
USE files      !
USE scalars    !
USE tracking   !
USE xsects     !
USE neutrons   !
!!!!!!!!!!!!!!!!!!!!
!
! LOCAL VARIABLES
!
!!!!!!!!!!!!!!!!!!!!
LOGICAL convk ! Convergence of Keff (outer)
LOGICAL maxout ! Maximum Number of Outer Iterations
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION cnvk   ! Max Convergence for Keff (outer)
DOUBLE PRECISION cross  ! Macroscopic Cross Section
DOUBLE PRECISION phi    ! Scalar Flux for Current Iteration (read)
DOUBLE PRECISION keold  ! Keff from Previous Iteration
DOUBLE PRECISION fdnum  ! Total Vol-Wghtd Fission Distrib (Current)
DOUBLE PRECISION fdden  ! Total Vol-Wghtd Fission Distrib (Previous)
DOUBLE PRECISION fiss   ! Fission Source (running sum)
DOUBLE PRECISION fdtmp  ! Fission Distribution (running sum)
DOUBLE PRECISION fd0tmp ! ...Iterate J

END

DO g = 1,ng
  DO d = 1,dirnum
    nnxtmp = dirset(d)%nnx
    nnytmp = dirset(d)%nny
    naztmp = nnxtmp + nnytmp
    DO q = 1,nxsets
      cross = xstr(g,q)
      DO n = 1,naztmp
        npltmp = npl(n,d)
        DO p = 1,npltmp
          track = zray(p,n,d)%trcln
          ncp(p,n,q,d,g) = exp(-cross * track)
        END DO
      END DO
    END DO
  END DO
END DO
!
END
DOUBLE PRECISION fd1tmp ! ...Iterate J-1
DOUBLE PRECISION chitmp ! Chi
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER nset   ! Material Set #
INTEGER lngmod ! Marker to Signal Long Output
INTEGER rstmod ! Marker to Signal Restart Dump
INTEGER ninner ! Max # Inners - to output
!!!!!!!!!!!!!!!!!!!!
!
! INDICES & POINTERS
!
!!!!!!!!!!!!!!!
INTEGER g,h ! Energy Group Indices
INTEGER i   ! Cell Indices (x-dir)
INTEGER ii  ! Cell Indices (y-dir)
INTEGER iii ! Cell Indices (z-dir)
!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                             !
!   Perform Inner Iteration   !
!                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      fd1(iii,ii,i) = fd0(iii,ii,i)
      fdtmp = 0.0
      DO g = 1,ng
        cross = nxsf(g,nset)
        phi = sphi(iii,ii,i,g)
        fdtmp = fdtmp + phi * cross
      END DO
    END DO
  END DO
END DO
fd0(iii,ii,i) = fdtmp
END DO
END DO
END DO
!
keold = keff
fdnum = 0.0
fdden = 0.0
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      fd0tmp = fd0(iii,ii,i)
      fd1tmp = fd1(iii,ii,i)
      fdnum = fdnum + fd0tmp * dx * dy * dz
      fdden = fdden + fd1tmp * dx * dy * dz
    END DO
  END DO
END DO
keff = keold * (fdnum / fdden)
cnvk = abs((keff - keold) / keff)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                              !
!   Calculate Fission Source    !
!                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
  chitmp = chip(g)
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        fiss = 0.0
        nset = xsmap(iii,ii,i)
        DO h = 1,ng
          cross = nxsf(h,nset)
          phi = sphi(iii,ii,i,h)
          fiss = fiss + cross * phi
        END DO
        fsav(iii,ii,i,g) = (fiss / fpi) * (chitmp / keff)
      END DO
    END DO
  END DO
END DO
END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                              !
!   Outer Iteration Clean-Up    !
!                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!   ...check k-eff convergence  !
IF (cnvk.lt.cck) THEN
  convk = .false.
END IF
!
!   ...simple output             !
WRITE (ofile,10) iterout,keff,cnvk,ninner
WRITE (*,    10) iterout,keff,cnvk,ninner
!
!   ...long output               !

340
lngmod = MOD(iterout,lngnum)
IF (lngmod.eq.0) THEN
   CALL write3dss()
ENDIF
!
!! ...restart dump
!!
IF (dssflg.eq.1) THEN
   rstmod = MOD(iterout,rstnum)
   IF (rstmod.eq.0) THEN
      CALL rstDumpSS()
   ENDIF
   END IF
!
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
10 FORMAT(2x,i5,3x,es11.4,3x,es11.4,3x,i6)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RETURN
END

SUBROUTINE inSScon(ninner)
!
!!!!!!!!!!!!!!!!!!!!
USE files      !
USE scalars    !
USE tracking   !
USE xsects     !
USE neutrons   !
!!!!!!!!!!!!!!!!!!!!
!
! LOCAL VARIABLES
!
!!!!!!!!!!!!!!!!!!!!
LOGICAL conva  ! Convergence of Angular Flux (inner)
LOGICAL convs  ! Convergence of Scalar Flux (inner)
LOGICAL maxin  ! Maximum Number of Inner Iterations
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION cnvaf    ! Max Convergence for Angular Flux (inner)
DOUBLE PRECISION cnvsf    ! Max Convergence for Scalar Flux (inner)
DOUBLE PRECISION cnvtmp   ! Temporary Buffer for Convergence Value
DOUBLE PRECISION albin    ! Albedo - Track that Enters @ Ext.Boundary
DOUBLE PRECISION aphin    ! Incoming Angular Flux
DOUBLE PRECISION source   ! Total Source Term
DOUBLE PRECISION cross    ! Macroscopic Cross Section
DOUBLE PRECISION track    ! Track Length
DOUBLE PRECISION mfp      ! Non-Collision Probability
DOUBLE PRECISION aphiout  ! Outgoing Angular Flux - Current Iterate
DOUBLE PRECISION aphilast ! Outgoing Angular Flux - Previous Iterate
DOUBLE PRECISION sphiold  ! Scalar Flux from Previous Iteration
DOUBLE PRECISION sphiold  ! Scalar Flux from Previous Iteration
DOUBLE PRECISION weight   ! Quadrature Weight
DOUBLE PRECISION trsep    ! Total Track Separation (dA*dP)
DOUBLE PRECISION sphinew  ! Scalar Flux for Current Iteration (write)
DOUBLE PRECISION scat     ! Scattering Source (running sum)
DOUBLE PRECISION phi      ! Scalar Flux for Current Iteration (read)
DOUBLE PRECISION sphisum  ! Unweighted Scalar Flux (running sum)
DOUBLE PRECISION numsum ! Numerator of Cell-Avg A-Flux (run sum)
DOUBLE PRECISION densum ! Denominator of Cell-Avg A-Flux (run sum)
DOUBLE PRECISION travg ! Track-Averaged Angular Flux (running sum)
DOUBLE PRECISION aphiavg ! Cell-Averaged Angular Flux
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

INTEGER ninner ! (PASSED) Max # Inners to Convergence
INTEGER nnxtmp ! # Plane Entries along +x Edge of Azimuthal Cell
INTEGER nnytmp ! # Plane Entries along +y Edge of Azimuthal Cell
INTEGER nnstmp ! # Track Entries along +z Edge of Polar Cell
INTEGER nnztmp ! # Track Entries along +s Edge of Polar Cell
INTEGER naztmp ! # Planes in 3D Cell
INTEGER nset ! Material Set #
INTEGER npltmp ! # Tracks in Polar Planar Section
INTEGER iterin ! Inner Iteration Count
INTEGER obflg ! Flag to Indicate If Track Reached System Boundary
!!!!!!!!!!!!!!!!!!!! [0=Boundary Not Reached, otherwise Boundary Reached]

! INDICES & POINTERS
!
!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER g,h ! Energy Group Indices
INTEGER d,dd,dq,dfr ! Direction Indices
INTEGER i,ix ! Cell Indices (x-dir)
INTEGER ii,iix ! Cell Indices (y-dir)
INTEGER iii,iiix ! Cell Indices (z-dir)
INTEGER n,nn,nfr,nnt ! Plane Indices
INTEGER p,pp,pfr,ppt ! Track Indices
!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
BEGIN Inner Iteration
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

ninner = 0
DO g = 1,ng
  conva = .true.
  convs = .true.
  maxin = .true.
  iterin = 0
  DO WHILE ((conva.and.convs).and.maxin)
    cnvaf = 0.0
    cnvsf = 0.0
    iterin = iterin + 1
    IF (iterin.ge.iterimax) THEN
      maxin = .false.
    ENDIF
  ENDWHILE

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Calculate Angular Flux Distribution
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

dd = 0
DO dq = 1,8
  DO d = 1,dirnum
    dd = dd + 1
    nnxtmp = dirset(d)%nnx
    nnytmp = dirset(d)%nny
nnstmp = dirset(d)\%nns
nnztmp = dirset(d)\%nnz
naztmp = nnxtmp + nnytmp

! Tracks that originate along +/-x face
!
DO ii = 1,ny
  DO iii = 1,nz
    DO n = nnytmp+1,naztmp
      npltmp = npl(n,d)
      DO p = npltmp-nnstmp+1,npltmp
        obflg = 0
        nn = n
        pp = p
        iix = ii
        iiix = iii
        SELECT CASE (dq)
          CASE (1)
            ix = 1
            dfr = dirnum + d
            albin = albl
          CASE (2)
            ix = nx
            dfr = d
            albin = alb2
          CASE (3)
            ix = nx
            dfr = 3*dirnum + d
            albin = alb2
          CASE (4)
            ix = 1
            dfr = 2*dirnum + d
            albin = alb2
          CASE (5)
            ix = 1
            dfr = 5*dirnum + d
            albin = alb1
          CASE (6)
            ix = nx
            dfr = 4*dirnum + d
            albin = alb2
          CASE (7)
            ix = nx
            dfr = 7*dirnum + d
            albin = alb2
          CASE (8)
            ix = 1
            dfr = 6*dirnum + d
            albin = alb1
        END SELECT
        nfr = zray(pp,nn,d)\%plnfr
        pfr = zray(pp,nn,d)\%trcfr
        aphin = albin * aphi(pfr,nfr,iiix,iix,ix,dfr,g)
        DO
          source = ssvav(iiix,iix,ix,g)+fsav(iiix,iix,ix,g)
          nset = xsmmap(iiix,iix,ix)
          cross = xstr(g,nset)
          mfp = ncp(pp,nn,nset,d,g)
          aphiout = (aphin * mfp) + ((source * (1 - mfp)) &
            &       / cross)
          IF (aphiout.lt.aphimin) THEN
            aphiout = aphimin
          ELSE
aphilast = aphi(pp,nn,iiix,iix,ix,dd,g)
cnvtmp = abs((aphiout - aphilast) / aphiout)
IF (cnvtmp.gt.cnvaf) THEN
  cnvaf = cnvtmp
ENDIF
ENDIF
aphi(pp,nn,iiix,iix,ix,dd,g) = aphiout
npltmp = npl(nn,d)
CALL nextTrack(dq,nn,pp,ix,iix,iiix, &
  nnxtmp,nnstmp,obflg)
IF (obflg.eq.1) THEN
  EXIT
ELSE
  aphin = aphiout
  nnt = nn
  ppt = pp
  nn = zray(ppt,nnt,d)%plnto
  pp = zray(ppt,nnt,d)%trcto
ENDIF
END DO
END DO
END DO
END DO
!
Tracks that originate along +/-y face
!
DO iii = 1,nz
DO i = 1,nx
DO n = 1,nnytmp
  npltmp = npl(n,d)
  DO p = npltmp-nnstmp+1,npltmp
    obflg = 0
    nn = n
    pp = p
    ix = i
    iiix = iii
    SELECT CASE (dq)
      CASE (1)
        iix = 1
        dfr = 3*dirnum + d
        albin = albin
      CASE (2)
        iix = 1
        dfr = 2*dirnum + d
        albin = albin
      CASE (3)
        iix = ny
        dfr = dirnum + d
        albin = albin
      CASE (4)
        iix = ny
        dfr = d
        albin = albin
      CASE (5)
        iix = 1
        dfr = 7*dirnum + d
        albin = albin
      CASE (6)
        iix = 1
        dfr = 6*dirnum + d
        albin = albin
      CASE (7)
        iix = 1
        dfr = 5*dirnum + d
        albin = albin
    END SELECT
  END DO
END DO
END DO
END DO

iix = ny
dfr = 5*dirnum + d
albin = alb4
CASE (8)
iix = ny
dfr = 4*dirnum + d
albin = alb4
END SELECT

nfr = zray(pp,nn,d)%plnfr
pfr = zray(pp,nn,d)%trcfr
aphin = albin * aphi(pfr,nfr,iix,iix,ix,dfr,g)
DO
  source = ssav(iix,iix,ix,g)+fsav(iix,iix,ix,g)
nset = xsmap(iix,iix,ix)
cross = xstr(g,nset)
mfp = ncp(pp,nn,nset,d,g)
aphiout = (aphin * mfp) + ((source * (1 - mfp)) &
    &       / cross)
IF (aphiout.lt.aphimin) THEN
  aphiout = aphin
ELSE
  aphilast = aphi(pp,nn,iix,iix,ix,ix,dd,g)
  cnvtmp = abs((aphiout - aphilast) / aphiout)
  IF (cnvtmp.gt.cnvaf) THEN
    cnvaf = cnvtmp
  ENDIF
ENDIF
aphi(pp,nn,iix,iix,ix,dd,g) = aphiout
npltmp = npl(nn,d)
CALL nextTrack(dq,nn,pp,ix,iix,iiix,nnt,nnxtmp,obflg)
IF (obflg.eq.1) THEN
  EXIT
ELSE
  aphin = aphiout
  nnt = nn
  ppt = pp
  nn = zray(ppt,nnt,d)%pln
  pp = zray(ppt,nnt,d)%trc
  aphiout = aphin
ENDIF
END DO
END DO
END DO
END DO
END DO

! Tracks that originate along +/-z face
!
DO i = 1,nx
  DO ii = 1,ny
    DO n = 1,naztmp
      npltmp = npl(n,d)
      DO p = 1,npltmp-nnstmp
        obflg = 0
        nn = n
        pp = p
        ix = i
        iix = ii
        SELECT CASE (dq)
          CASE (1)
            iixx = 1
            dfr = 4*dirnum + d
            albin = alb5
END DO
END DO
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      sphiold = sphi(iii,ii,i,g)
      dd = 0
      sphisum = 0.0
      source = ssav(iii,ii,i,g) + fsav(iii,ii,i,g)
      nset = xsmap(iii,ii,i)
      cross = xstr(g,nset)
      DO dq = 1,8
        DO d = 1,dirnum
          dd = dd + 1
          numsum = 0.0
          densum = 0.0
          weight = dirset(d)%wght
          trsep = dirset(d)%dazi * dirset(d)%dpol
          nnxtmp = dirset(d)%nnx
          nnytmp = dirset(d)%nny
          naztmp = nnxtmp + nnytmp
          nnstmp = dirset(d)%nns
          DO n = 1,naztmp
            npltmp = npl(n,d)
            DO p = 1,npltmp
              track = zray(p,n,d)%trcln
              aphiout = aphi(p,n,iii,ii,i,dd,g)
              CALL findAphin(p,n,iii,ii,i,dd,g, &
                             nnytmp,nnstmp,npltmp,aphin)
              travg = (source / cross) + ((aphin - aphiout) &
                                          / (cross * track))
              numsum = numsum + travg * track * trsep
              densum = densum + track * trsep
          END DO
        END DO
        aphiavg = numsum / densum
        sphisum = sphisum + aphiavg * weight
      END DO
    END DO
  END DO
END DO
sphinew = sphisum * fpi
sphi(iii,ii,i,g) = sphinew
cnvtmp = abs((sphinew - sphiold) / sphiold)
IF (cnvtmp.gt.cnvsf) THEN
  cnvsf = cnvtmp
ENDIF
END DO
END DO
END DO
DO i = 1, nx
  DO ii = 1, ny
    DO iii = 1, nz
      scat = 0.0
      nset = xsmap(iii, ii, i)
      DO h = 1, ng
        phi = sphi(iii, ii, i, h)
        scat = scat + xssc(g, h, nset) * phi
      END DO
      ssav(iii, ii, i, g) = scat / fpi
    END DO
  END DO
END DO

IF (cnvsf.lt.ccs) THEN
  convs = .false.
ENDIF
IF (cnvaf.lt.cca) THEN
  conva = .false.
ENDIF
END DO

RETURN
END

SUBROUTINE rstDumpSS()
USE files      !
USE scalars    !
USE tracking   !
USE neutrons   !
USE xsects  
!!!!!!!!!!!!!!!!!!!!
!
! RSTDUMPSS writes information to the restart file
! for restart calculation or transient initialization
!
! Local Parameters
!
!!!!!!!!!!!!!!!!!!!!
INTEGER nnxtmp  ! # Plane Entries along +-x Edge of Azimuthal Cell
INTEGER nnytmp  ! # Plane Entries along +-y Edge of Azimuthal Cell
INTEGER naztmp  ! # Planes in 3D Cell
INTEGER npltmp  ! # Tracks in Polar Planar Section
!!!!!!!!!!!!!!!!!!!!
!
! Local Indices
!
!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER g          ! Energy Group Index
INTEGER d, dq, dd  ! Direction Indices
INTEGER i, ii, iii ! Cell Indices
INTEGER n          ! Plane Index
INTEGER p          ! Track Index
!!!!!!!!!!!!!!!!!!!!!!!!!!
!
OPEN (UNIT=rfile,FILE=rfname,STATUS='replace',FORM='BINARY')
!
WRITE(rfile) iterout,kinf,keff
WRITE(rfile) sphi
WRITE(rfile) fd0
WRITE(rfile) fd1
WRITE(rfile) aphi
!
CLOSE (rfile)
!
END

SUBROUTINE postSS3D()
!
!!!!!!!!!!!!!!!
USE files    
USE scalars  
USE xsects   
USE neutrons 
!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER nset           !
DOUBLE PRECISION nrsum !
DOUBLE PRECISION drsum !
DOUBLE PRECISION ncsum !
DOUBLE PRECISION dcsum !
DOUBLE PRECISION fnorm !
DOUBLE PRECISION cross !
DOUBLE PRECISION phi  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!
INTEGER g          !
INTEGER i, ii, iii !
INTEGER q          !

349
ALLOCATE(rgnflux(ng,nxsets))
ALLOCATE(rgnabs(ng,nxsets))
ALLOCATE(rgnprod(ng,nxsets))
ALLOCATE(corflux(ng))
ALLOCATE(corabs(ng))
ALLOCATE(corprod(ng))

!!! Scalar Flux Normalization !!!

IF (nmflag.eq.1) THEN

! normalize to one production

  fnorm = 0.0
  DO g = 1,ng
    DO i = 1,nx
      DO ii = 1,ny
        DO iii = 1,nz
          nset = xsmap(iii,ii,i)
          cross = nxsf(g,nset)
          phi = sphi(iii,ii,i,g)
          fnorm = fnorm + cross * phi * dx * dy * dz
        END DO
      END DO
    END DO
  END DO
ELSE IF (nmflag.eq.2) THEN

! normalize to one absorption

  fnorm = 0.0
  DO g = 1,ng
    DO i = 1,nx
      DO ii = 1,ny
        DO iii = 1,nz
          nset = xsmap(iii,ii,i)
          cross = xsab(g,nset)
          phi = sphi(iii,ii,i,g)
          sphi(iii,ii,i,g) = phi / fnorm
        END DO
      END DO
    END DO
  END DO
DO g = 1,ng
   DO i = 1,nx
      DO ii = 1,ny
         DO iii = 1,nz
            phi = sphi(iii,ii,i,g)
            sphi(iii,ii,i,g) = phi / fnorm
         END DO
      END DO
   END DO
END DO
!
ELSE IF (nmflag.eq.3) THEN
!
! cell-avg flux normalization
!
ELSE
!
! no normalization
!
ENDIF
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Calculate Core- & Region-Averaged Scalar Flux
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
   DO q = 1,nxsets
      nrsum = 0.0
      drsum = 0.0
      DO i = 1,nx
         DO ii = 1,ny
            DO iii = 1,nz
               nset = xsmap(iii,ii,i)
               IF (nset.eq.q) THEN
                  nrsum = nrsum + sphi(iii,ii,i,g) * dx * dy * dz
                  drsum = drsum + dx * dy * dz
               ENDIF
            END DO
         END DO
      END DO
      rgnflux(g,q) = nrsum / drsum
   END DO
END DO
!
DO g = 1,ng
   ncsum = 0.0
   dcsum = 0.0
   DO i = 1,nx
      DO ii = 1,ny
         DO iii = 1,nz
            nset = xsmap(iii,ii,i)
            ncsum = ncsum + sphi(iii,ii,i,g) * dx * dy * dz
            dcsum = dcsum + dx * dy * dz
         END DO
      END DO
   END DO
   corflux(g) = ncsum / dcsum
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Calculate Core- & Region-Averaged Absorption Rate
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
  DO q = 1,nxsets
    nrsum = 0.0
    drsum = 0.0
    DO i = 1,nx
      DO ii = 1,ny
        DO iii = 1,nz
          nset = xsmap(iii,ii,i)
          IF (nset.eq.q) THEN
            nrsum = nrsum + xsab(g,nset) * sphi(iii,ii,i,g) *dx*dy*dz
            drsum = drsum + dx * dy * dz
          ENDIF
        END DO
      END DO
    END DO
  END DO
  rgnabs(g,q) = nrsum / drsum
END DO
END DO
!
DO g = 1,ng
  ncsnum = 0.0
  dcsnum = 0.0
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        nset = xsmap(iii,ii,i)
        ncsnum = ncsnum + xsab(g,nset) * sphi(iii,ii,i,g) *dx*dy*dz
        dcsnum = dcsnum + dx * dy * dz
      END DO
    END DO
  END DO
  corabs(g) = ncsnum / dcsnum
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Calculate Core- & Region-Averaged Production Rate
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
  DO q = 1,nxsets
    nrsum = 0.0
    drsum = 0.0
    DO i = 1,nx
      DO ii = 1,ny
        DO iii = 1,nz
          nset = xsmap(iii,ii,i)
          IF (nset.eq.q) THEN
            nrsum = nrsum + nxsf(g,nset) * sphi(iii,ii,i,g) *dx*dy*dz
            drsum = drsum + dx * dy * dz
          ENDIF
        END DO
      END DO
    END DO
  END DO
  rgnprod(g,q) = nrsum / drsum
END DO
END DO
!

DO g = 1,ng
   ncsum = 0.0
   dcsum = 0.0
   DO i = 1,nx
      DO ii = 1,ny
         DO iii = 1,nz
            nset = xsmap(iii,ii,i)
            ncsum = ncsum + nxsf(g,nset) * sphi(iii,ii,i,g) *dx*dy*dz
         END DO
      END DO
   END DO
   corprod(g) = ncsum / dcsum
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Report Final Results to OUTPUT File
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
WRITE(ofile,10)
WRITE(ofile,*)   '    **********************'
WRITE(ofile,*)   '    *Calculation Complete*'
WRITE(ofile,*)   '    **********************'
WRITE(ofile,10)
WRITE(ofile,100) '                   Final Keff = ',keff
WRITE(ofile,110) '           # Outer Iterations = ',iterout
WRITE(ofile,10)
WRITE(ofile,*)   'Core-Averaged Scalar Flux:'
WRITE(ofile,10)
WRITE(ofile,*)   '  Group        Value'
WRITE(ofile,*)   '  -----    --------------'
DO g = 1,ng
   WRITE(ofile,150) g,corflux(g)
END DO
WRITE(ofile,10)
WRITE(ofile,*)   'Core-Averaged Absorption Rate:'
WRITE(ofile,10)
WRITE(ofile,*)   '  Group        Value'
WRITE(ofile,*)   '  -----    --------------'
DO g = 1,ng
   WRITE(ofile,150) g,corabs(g)
END DO
WRITE(ofile,10)
WRITE(ofile,*)   'Core-Averaged Production Rate:'
WRITE(ofile,10)
WRITE(ofile,*)   '  Group        Value'
WRITE(ofile,*)   '  -----    --------------'
DO g = 1,ng
   WRITE(ofile,150) g,corprod(g)
END DO
WRITE(ofile,10)
WRITE(ofile,*)   'Region-Averaged Scalar Flux:'
WRITE(ofile,10)
WRITE(ofile,*)   ' Region       Energy         Rgn-Avg'
WRITE(ofile,*)   '(XS Set)      Group         Scalar Flux'
WRITE(ofile,*)   '--------      ------      --------------'
DO q = 1,nxsets
   DO g = 1,ng
      WRITE(ofile,200) q,g,rgnflux(g,q)
   END DO
END DO
WRITE(ofile,10)
WRITE(ofile,*), 'Region-Averaged Absorption Rate:'
WRITE(ofile,10)
WRITE(ofile,*), 'Region       Energy         Rgn-Avg'
WRITE(ofile,*), '(XS Set)      Group         Abs. Rate'
WRITE(ofile,*), '--------      ------      --------------'
DO q = 1,nxsets
   DO g = 1,ng
      WRITE(ofile,200) q,g,rgnabs(g,q)
   END DO
END DO
WRITE(ofile,10)
WRITE(ofile,*), 'Region-Averaged Production Rate:'
WRITE(ofile,10)
WRITE(ofile,*), 'Region       Energy         Rgn-Avg'
WRITE(ofile,*), '(XS Set)      Group         Prod. Rate'
WRITE(ofile,*), '--------      ------      --------------'
DO q = 1,nxsets
   DO g = 1,ng
      WRITE(ofile,200) q,g,rgnprod(g,q)
   END DO
END DO
WRITE(ofile,*), 'Scalar Flux Distribution'
WRITE(ofile,*), ''
USE files      !
USE scalars    !
USE tracking   !
USE xsects     !
USE neutrons   !
!!!!!!!!!!!!!!!!!!!!
INTEGER g        !
INTEGER i,ii,iii !
!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!
DO g = 1,ng
   DO iii = nz,1,-1
      WRITE(ofile,*), 'Group: ',g,' Level: ',iii
      DO ii = ny,1,-1
         WRITE(ofile,*), 'Scalar Flux Distribution'
WRITE(ofile,100) (sphi(iii,ii,i,g),i=1,nx)
END DO
END DO
END DO
WRITE(ofile,*) ''
WRITE(ofile,*)'Fission Rate Distribution'
WRITE(ofile,*) ''
DO iii = nz,1,-1
WRITE(ofile,*)'Group: ',g,' Level: ',iii
DO ii = ny,1,-1
WRITE(ofile,100) (fd0(iii,ii,i),i=1,nx)
END DO
END DO
WRITE(ofile,*) ''
WRITE(ofile,*)'Scattering Source Distribution'
WRITE(ofile,*) ''
DO g = 1,ng
DO iii = nz,1,-1
WRITE(ofile,*)'Group: ',g,' Level: ',iii
DO ii = ny,1,-1
WRITE(ofile,100) (ssav(iii,ii,i,g),i=1,nx)
END DO
END DO
WRITE(ofile,*) ''
WRITE(ofile,*)'Fission Source Distribution'
WRITE(ofile,*) ''
DO g = 1,ng
DO iii = nz,1,-1
WRITE(ofile,*)'Group: ',g,' Level: ',iii
DO ii = ny,1,-1
WRITE(ofile,100) (fsav(iii,ii,i,g),i=1,nx)
END DO
END DO
WRITE(ofile,*) ''
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

100 FORMAT(10(1x,es10.3e2))

MODULE files
!
! INPUT File [UNIT = 1, NAME = input.moc]
!
! must be specified by user prior to execution
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: ifile = 1
CHARACTER(9), PARAMETER :: ifname = 'input.moc'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DATA File [UNIT = 2, NAME = data.ray]
!
...contains ray-tracing results that are required

355
as input for the MOCK-3D angular flux calculation

stored in unformatted but compact form
i.e., no annotation - just the raw data
must be specified via execution of RayMonde prior to execution

INTEGER, PARAMETER :: afile = 2
CHARACTER(8), PARAMETER :: afname = 'data.ray'

INTEGER, PARAMETER :: ofile = 3
CHARACTER(10), PARAMETER :: ofname = 'output.moc'

INTEGER, PARAMETER :: dfile = 4
CHARACTER(9), PARAMETER :: dfname = 'debug.moc'

INTEGER, PARAMETER :: rfile = 5
CHARACTER(11), PARAMETER :: rfname = 'rstrtSS.moc'

END MODULE files

MODULE neutrons

NCP  --> Non-Collision Probability Table \[\exp(-\text{cross}*\text{track})\]
Values are Tabulated During Problem Initialization
and Found via Table-Lookup During Primary Execution
to Avoid Unnecessary Repetition of the \(\exp()\) Function

DOUBLE PRECISION, ALLOCATABLE :: ncp(:,,:,:,:,:) !

APHI --> Local Angular Flux Distribution
3D Cartesian Geometry:
8 Octants (I through VIII)

DOUBLE PRECISION, ALLOCATABLE :: aphi(:,:,,:,:,:) !

SSAV --> Scattering Source Term
FSAV --> Fission Source Term
DOUBLE PRECISION, ALLOCATABLE :: ssav(:,,:,:,:)!
DOUBLE PRECISION, ALLOCATABLE :: fsav(:,,:,:,:)!

SPHI -----> Scalar Flux Distribution!

DOUBLE PRECISION, ALLOCATABLE :: sphi(:,,:,:,:)!

FD0 --> Fission Distribution, Current Iterate (J)
FD1 --> Fission Distribution, Iterate J-1!

RGNFLUX --> Region-Averaged Scalar Flux
RGNABS  --> Region-Averaged Absorption Rate
RGNPROD --> Region-Averaged Production Rate!

DOUBLE PRECISION, ALLOCATABLE :: rgnflux(:,:)
DOUBLE PRECISION, ALLOCATABLE :: rgnabs(:,:)
DOUBLE PRECISION, ALLOCATABLE :: rgnprod(:,:)

CORFLUX --> Core-Averaged Scalar Flux
CORABS  --> Core-Averaged Absorption Rate
CORPROD --> Core-Averaged Production Rate!

DOUBLE PRECISION, ALLOCATABLE :: corflux(:)
DOUBLE PRECISION, ALLOCATABLE :: corabs(:)
DOUBLE PRECISION, ALLOCATABLE :: corprod(:)

END MODULE neutrons
INTEGER rstflg !
!!!!!!!!!!!!!!!!!!!!
|
| Post-Processing Flux Normalization Flag:
| |
| Determines whether/how to normalize the
| final scalar flux distribution.
| |
| (If steady-state results are to be stored for use in
| time-dependent calculation,
| Option 0 should be selected as normalization can bias
| the initial angular/scalar flux distributions)
| |
| = 0 : No Flux Normalization
| = 1 : Normalize to One Production
| = 2 : Normalize to One Absorption
| = 3 : Cell-Avg Flux Normalization
| |
!!!!!!!!!!!!!!!!!!!!
INTEGER nmflag !
!!!!!!!!!!!!!!!!!!!!
|
| Restart Dump Flag:
| |
| Determines whether end-of-iterate date is to be stored
| for restart of the steady state calculation
| |
| = 0 : No Dump
| = 1 : Yes Dump
| |
!!!!!!!!!!!!!!!!!!!!
INTEGER dssflg !
!!!!!!!!!!!!!!!!!!!!
|
| Transient Dump Flag:
| |
| Determines whether final problem data is to be stored
| for use in time-dependent calculation
| |
| = 0 : No Dump
| = 1 : Yes Dump
| |
!!!!!!!!!!!!!!!!!!!!
INTEGER dtrflg !
!!!!!!!!!!!!!!!!!!!!
|
| NX, NY, and NZ --> Number of Cells in x-, y-, and z-directions, respectively.
| |
| To Perform a 1D (x) Calculation, set ny = nz = 1
| To Perform a 2D (xy) Calculation, set nz = 1
| |
!!!!!!!!!!!!!!
INTEGER nx   !
INTEGER ny   !
INTEGER nz   !
!!!!!!!!!!!!!!
|
| ALB1 through ALB6 --> Albedo Boundary Coefficients
| |
| 0 < ALB < 1
|   where ALB = 1 implies total reflection at boundary
|   and ALB = 0 implies no re-entry at boundary
|
To Perform a 1D (x) Calculation, set alb3 = alb4 = alb5 = alb6 = 1
To Perform a 2D (xy) Calculation, set alb5 = alb6 = 1

DOUBLE PRECISION alb1 ! x- face
DOUBLE PRECISION alb2 ! x+ face
DOUBLE PRECISION alb3 ! y- face
DOUBLE PRECISION alb4 ! y+ face
DOUBLE PRECISION alb5 ! z- face
DOUBLE PRECISION alb6 ! z+ face

DOUBLE PRECISION cca ! Convergence of Angular Flux (Inner Iterate)
DOUBLE PRECISION ccs ! Convergence of Scalar Flux (Inner Iterate)
DOUBLE PRECISION cck ! Convergence of K-Effective (Outer Iterate)

ITERIMAX, ITEROMAX --> Maximum Iteration Counts

INTEGER iterimax ! Max Iterations : Inner Iteration
INTEGER iteromax ! Max Iterations : Outer Iteration

RSTNUM --> # Outer Iterations b/w Restart Dumps
LNGNUM --> # Outer Iterations b/w Long Outputs

INTEGER ngeo ! NGEO --> Flag: Problem Geometry Type
= 3 --> 3D Cartesian Geometry

INTEGER dirnum ! DIRNUM --> Number of Directions in Quadrature Set

INTEGER dx, dy, and dz --> Scalar Cell Dimensions

In 1D Calculation, dy and dz are used but arbitrary
In 2D Calculation, dz is used but arbitrary
DOUBLE PRECISION dx ! x-dimension - same for all cells
DOUBLE PRECISION dy ! y-dimension - same for all cells
DOUBLE PRECISION dz ! z-dimension - same for all cells
!!!!!!!!!!!!!!!!!!!!!!!
!-------------------
! DEFINED DATA TYPE
!-------------------
!
! Data Type Specification for Quadrature Array
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

TYPE QUADINFO
!
  DOUBLE PRECISION wght !......Direction Weight  
  DOUBLE PRECISION azim !......Azimuthal Angle  
  DOUBLE PRECISION dazi !......dA track spacing  
  INTEGER      nnx !......# Spans in x-dir  
  INTEGER      nny !......# Spans in y-dir  
  INTEGER      npu !..# Azi Periodic Units  
  DOUBLE PRECISION pola !...........Polar Angle  
  DOUBLE PRECISION dpol !......dP track spacing  
  INTEGER      nns !...# Spans in (xy)-dir  
  INTEGER      nnz !......# Spans in z-dir  
END TYPE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! DIRSET is the working array for holding all problem quadrature data
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

TYPE(QUADINFO), ALLOCATABLE :: dirset(:)  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Max Limits for Sizing Arrays...
!
!!!!!!!!!!!!!!!!!!!!
INTEGER nazmax  !  Maximum # Tracks - Azimuthal Plane  
INTEGER npomax  !  Maximum # Tracks - Polar Plane  
!!!!!!!!!!!!!!!!!!!!

! ****************************
! **------------------------**
! *|                        |*
! *|  internally generated  |*
! *|                        |*
! **------------------------**
! ****************************

! KINF --> Infinite Multiplication Factor
! KEFF --> Effective Multiplication Factor
!
!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION keff  
DOUBLE PRECISION kinf  
!!!!!!!!!!!!!!!!!!!!

! ITEROUT --> Outer Iteration Count
!
!!!!!!!!!!!!!!!!!!!!
INTEGER iterout  
!!!!!!!!!!!!!!!!!!!!

! ******************
! **--------------**
!              360
Minimum Threshold Value for Angular Flux
(required to avoid numerical problems due to very small tracks)

DOUBLE PRECISION, PARAMETER :: aphimin = 1.0E-12

Pi and her cousins (to 8 significant figures)

DOUBLE PRECISION, PARAMETER :: pi = 3.1415927   ! Pi
DOUBLE PRECISION, PARAMETER :: hpi = 1.57079633  ! Pi/2
DOUBLE PRECISION, PARAMETER :: tpi = 6.2831853   ! 2*Pi
DOUBLE PRECISION, PARAMETER :: fpi = 12.5663706  ! 4*Pi
DOUBLE PRECISION, PARAMETER :: pirt = 1.77245385  ! Root-Pi

END MODULE scalars

MODULE tracking
!
! DEFINED DATA TYPE
!-------------------
!

TYPE AZIRAYINFO          !
  DOUBLE PRECISION trcln ! Track Length
  INTEGER trcto ! Track TO which Track Couples
  INTEGER trcfr ! Track FROM which Track Couples
END TYPE                 !

TYPE POLRAYINFO          !
  DOUBLE PRECISION trcln ! Track Length
  INTEGER plnto ! Plane TO which Track Couples
  INTEGER trcto ! Track TO which Track Couples
  INTEGER plnfr ! Plane FROM which Track Couples

XYRAY is the variable for Azimuthal (2D) Ray Tracing

...track length [%TRCLN] represents the PARTIAL
  path length --> in 2D, polar component is applied
  during outer iteration

shape: (MAX#Tracks, #Directions)

TYPE(AZIRAYINFO), ALLOCATABLE :: xyray(:,:,)

TYPE(POLRAYINFO), ALLOCATABLE :: polray(:,:,)

END MODULE tracking
INTEGER          trcfr ! Track FROM which Track Couples
END TYPE                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! ZRAY is the variable holds 3D Track Lengths
!! & Coupling Matrix
!!
! shape: (MAX#Tracks, MAX#Planes, #Directions)
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TYPE(POLRAYINFO), ALLOCATABLE :: zray(:,,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! *note* a 2D calc. uses XYRAY only
! a 3D calc. uses XYRAY and ZRAY
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, ALLOCATABLE :: npl(:,,:) ! #Tracks in plane (3D)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END MODULE tracking

MODULE xsects

!! NG --> Number of Energy Groups
!!
!!!!!!!!!!!!!
INTEGER ng !
!!!!!!!!!!!!!
!
! NXSETS --> Number of Cross-Section Sets
!
!!!!!!!!!!!!!
INTEGER nxsets !
!!!!!!!!!!!!!
!
! XSMAP --> Matrix to Connect each Spatial Cell to a XS Set
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, ALLOCATABLE :: xsmap(:,,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! XSTR --> Macroscopic Transport Cross-Section
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: xstr(:,,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NXSF --> Macroscopic nu-Fission Cross-Section
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: nxsf(:,,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! XSSC --> Macroscopic Group-Group Scattering Cross-Section
! (transport corrected)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: xssc(:,,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

362
B.3 Source Code for MOCK-3DK

PROGRAM MOCK3DK
!
!!!!!!!!!!!!!!!!!!!!
CALL MrNeutron()!
!!!!!!!!!!!!!!!!!!!!
!
STOP
!
END PROGRAM

SUBROUTINE MrNeutron()
!
!!!!!!!!!!!!!!!
USE files!
USE scalars!
!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!
INTEGER estate!
!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
! Load & Echo Problem Input!
!!!!!!!!!!!!!!!!!!!!
!
OPEN (UNIT = ifile, FILE = ifname)
OPEN (UNIT = afile, FILE = afname)
CALL genModelTR()
CLOSE (ifile)
CLOSE (afile)
OPEN (UNIT = ofile, FILE = ofname, STATUS = 'replace')
OPEN (UNIT = dfile, FILE = dfname, STATUS = 'replace')
CALL writeModelTR()
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Pass Control to Appropriate Transient Method Driver!

SELECT CASE (trflg)

! Backward Differencing - Explicit Direct
!
CASE (1,2,3)
   CALL initBD()
   CALL mainBD(estate)
   CALL postTR3D(estate)
!
! Forward Differencing - Implicit Direct
!
CASE (4,5,6)
   CALL initFD()
   CALL mainFD(estate)
   CALL postTR3D(estate)
!
CASE DEFAULT
!trap
END SELECT

!! Clean-Up
!!
CLOSE (ofile)!
CLOSE (dfile)!

END SUBROUTINE genModelTR()

USE files    
USE scalars  
USE tracking 
USE xsects   

DOUBLE PRECISION btmp   ! beta sum
DOUBLE PRECISION statim ! total duration of time state
INTEGER stloop          ! running sum : ramp/step transient

INTEGER g, h       ! Energy Group Indices
INTEGER l          ! Delay Group Index
INTEGER d          ! Direction Index
INTEGER i, ii, iii ! Cell Indices
INTEGER n          ! Plane Index
INTEGER p          ! Track Index
INTEGER q          ! XS Set Index
INTEGER k          ! TD State Index

!!
! Load Transient/Restart Flags from Input File!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

READ(ifile,*) trflg, critflg, pcflg

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Load Tracking File Information!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

READ(afile,*) ngeo, dirnum
ALLOCATE(dirset(dirnum))
READ(afile,*) dx, dy, dz
READ(afile,*) nazmax, npomax
ALLOCATE(zray(npomax, nazmax, dirnum))
ALLOCATE(npl(nazmax, dirnum))
DO d = 1, dirnum
  READ(afile,*) dirset(d)
  DO n = 1, nazmax
    READ(afile,*) npl(n, d)
    DO p = 1, npomax
      READ(afile,*) zray(p, n, d)
    END DO
  END DO
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Load Cell and Boundary Info!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

READ (ifile,*) nx, ny, nz
READ (ifile,*) alb1, alb2, alb3, alb4, alb5, alb6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Load # Energy/Delay Groups and # of XS Sets!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

READ (ifile,*) ng, ndg, nxsets

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Load Iteration/Convergence Parameters!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

READ (ifile,*) cca, ccs, iterimax
IF (((trflg.eq.4).or.(trflg.eq.5)).or.(trflg.eq.6)) THEN
  READ (ifile,*) cco, iteromax
ENDIF

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Load Transient Control Data!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!
READ (ifile,*) nstate
ALLOCATE (statyp(nxsets,nstate))
ALLOCATE (delta(nstate))
ALLOCATE (timend(nstate))
ALLOCATE (lngtim(nstate))
DO k = 1,nstate
  READ (ifile,*) (statyp(q,k),q=1,nxsets)
  READ (ifile,*) delta(k),timend(k),lngtim(k)
END DO
!
stloop = 0
DO k = 1,nstate
  DO q = 1,nxsets
    stloop = stloop + statyp(q,k)
  END DO
END DO
IF (stloop.gt.0) THEN
  tdramp = 1
ELSE
  tdramp = 0
ENDIF
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Load State-Dependent Material Cross-Section Data !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (xstr(ng,nxsets,0:nstate+1))
ALLOCATE (nxsf(ng,nxsets,0:nstate+1))
ALLOCATE (xssc(ng,ng,nxsets,0:nstate+1))
ALLOCATE (xsab(ng,nxsets,0:nstate+1))
DO k = 0,nstate+1
  DO q = 1,nxsets
    DO g = 1,ng
      READ (ifile,*) xstr(g,q,k),nxsf(g,q,k),xsab(g,q,k)
      READ (ifile,*) (xssc(h,g,q,k),h=1,ng)
    END DO
  END DO
END DO
ALLOCATE(ramptr(ng,nxsets,nstate))
ALLOCATE(rampnf(ng,nxsets,nstate))
ALLOCATE(rampss(ng,ng,nxsets,nstate))
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Calculate XS Ramp Increments !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO k = 1,nstate
  IF (k.eq.1) THEN
    statim = timend(k)
  ELSE
    statim = timend(k) - timend(k-1)
  ENDIF
  DO q = 1,nxsets
    DO g = 1,ng
      IF (statyp(q,k).eq.1) THEN
        ramptr(g,q,k) = ((xstr(g,q,k+1) - xstr(g,q,k)) &
                         / statim) * delta(k)
        rampnf(g,q,k) = ((nxsf(g,q,k+1) - nxsf(g,q,k)) &
                         / statim) * delta(k)
      END IF
    END DO
  END DO
END DO
DO h = 1,ng
    rampss(h,g,q,k) = ((xssc(h,g,q,k+1) - &
    xssc(h,g,q,k)) / statim) * delta(k)
END DO
ELSE
    ramptr(g,q,k) = 0.0
    rampnf(g,q,k) = 0.0
    DO h = 1,ng
        rampss(h,g,q,k) = 0.0
    END DO
ENDIF
END DO
END DO
END DO

!!!!!!!!!!!!!!!!!!!!!!!
!                    
! Load Kinetics Data !
!                    
!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (vel(ng))
ALLOCATE (chip(ng))
ALLOCATE (chid(ng))
ALLOCATE (betal(ndg))
ALLOCATE (lambda(ndg))
READ (ifile,*) (vel(g),g=1,ng)
READ (ifile,*) (chip(g),g=1,ng)
READ (ifile,*) (chid(g),g=1,ng)
READ (ifile,*) (betal(l),l=1,ndg)
READ (ifile,*) (lambda(l),l=1,ndg)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                          
! Calculate Total Delayed Neutron Fraction!
!                                          
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
btmp = 0.0
DO l = 1,ndg
    btmp = btmp + betal(l)
END DO
beta = btmp
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                        
! Load Material Maps !
!                        
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (xsmap(nz,ny,nx))
DO iii = 1,nz
    DO ii = ny,1,-1
        READ (ifile,*) (xsmap(iii,ii,i), i=1,nx)
    END DO
END DO
RETURN
END

SUBROUTINE writeModelTR()
USE files
USE scalars
USE tracking
USE xsects

INTEGER nnstmp ! # Tracks Entering +/- s-edge
INTEGER naztmp ! # Planes in Cell
INTEGER npltmp ! # Tracks in Plane

INTEGER g,h      ! Energy Group Indices
INTEGER d        ! Direction Index
INTEGER i,ii,iii ! Cell Indices
INTEGER n        ! Track Index
INTEGER p        ! Plane Index
INTEGER q        ! XS Set index
INTEGER k
INTEGER l

WRITE (ofile,10) !
WRITE (ofile,30) !
WRITE (ofile,31) !
WRITE (ofile,32) !
WRITE (ofile,40) ! Write Geometry Data
WRITE (ofile,32) !
WRITE (ofile,31) !
WRITE (ofile,30) !
WRITE (ofile,10)!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,100) 'GEOMETRY TYPE: ngeo = ',ngeo !
WRITE (ofile,*) '   3D Cell Geometry'           !
WRITE (ofile,10)                                !
WRITE (ofile,*) '   CELL DIMENSIONS:'           !
WRITE (ofile,105) 'Cell x-Dimension =',dx     !
WRITE (ofile,105) 'Cell y-Dimension =',dy     !
WRITE (ofile,105) 'Cell z-Dimension =',dz     !
WRITE (ofile,10)                                !
WRITE (ofile,*) '   SYSTEM DIMENSIONS:'         !
WRITE (ofile,110) '# Cells in x-direction =',nx !
WRITE (ofile,110) '# Cells in y-direction =',ny !
WRITE (ofile,110) '# Cells in z-direction =',nz !
WRITE (ofile,10)                                !
WRITE (ofile,*) '   ALBEDO BOUNDARY CONDITIONS:'!
WRITE (ofile,115) '-x face =',alb1             !
WRITE (ofile,115) '+x face =',alb2             !
WRITE (ofile,115) '-y face =',alb3             !
WRITE (ofile,115) '+y face =',alb4             !
WRITE (ofile,115) '-z face =',alb5             !
WRITE (ofile,115) '+z face =',alb6             !
WRITE (ofile,10)!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,10) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO d = 1,dirnum
  WRITE (ofile,200) d,dirnum
  WRITE (ofile,10) !
  WRITE (ofile,205) dirset(d)%wght
  WRITE (ofile,210) ' Azimuthal Angle = ',dirset(d)%azi,' rad'
  WRITE (ofile,210) 'Azimuthal Spacing = ',dirset(d)%dazi,' cm'
  WRITE (ofile,215) 'NNX = ', dirset(d)%nnx
  WRITE (ofile,215) 'NNY = ', dirset(d)%nny
  WRITE (ofile,210) 'Polar Angle = ',dirset(d)%pola,' rad'
  WRITE (ofile,210) 'Polar Spacing = ',dirset(d)%dpol,' cm'
  WRITE (ofile,215) 'NNS = ', dirset(d)%nns
  WRITE (ofile,215) 'NNZ = ', dirset(d)%nnz
END DO
WRITE (ofile,30) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,31) !
WRITE (ofile,32) !
WRITE (ofile,70) !
WRITE (ofile,32) !
WRITE (ofile,31) ! Write 3D Ray Tracing Data
WRITE (ofile,30) !
DO d = 1,dirnum
  nnstmp = dirset(d)%nns
  naztmp = dirset(d)%nnx + dirset(d)%nny
  WRITE (ofile,700) d
  WRITE (ofile,702) !
END DO
WRITE (ofile,32) !
WRITE (ofile,31) !
WRITE (ofile,30) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DO k = 1,nstate
  WRITE (ofile,500) 'Number Transient States: ',nstate
  WRITE (ofile,10)
  WRITE (ofile,500) 'Transient State #',k
  WRITE (ofile,*') Ramp Flags: ',(statyp(q,k),q=1,nxsets)
  WRITE (ofile,*') Time Step Size = ',delta(k)
  WRITE (ofile,*') Time Step End = ',timend(k)
  WRITE (ofile,*') Long Out Period = ',lngtim(k)
END DO
WRITE (ofile,32) !
WRITE (ofile,31) ! Write Material Properties
WRITE (ofile,30) !
WRITE (ofile,10) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,500) 'Number Energy Groups: ',ng
WRITE (ofile,500) 'Number Delay Groups: ',ndg
WRITE (ofile,500) 'Number of XS Sets: ',nxsets
WRITE (ofile,10)
WRITE (ofile,*) '   Vel:            Group        Value'
WRITE (ofile,*) '                   -----    --------------'
DO g = 1,ng
   WRITE (ofile,505) g,vel(g)
END DO
WRITE (ofile,10)
WRITE (ofile,*) '   Chip:            Group        Value'
WRITE (ofile,*) '                   -----    --------------'
DO g = 1,ng
   WRITE (ofile,505) g,chip(g)
END DO
WRITE (ofile,10)
WRITE (ofile,*) '   Chid:            Group        Value'
WRITE (ofile,*) '                   -----    --------------'
DO g = 1,ng
   WRITE (ofile,505) g,chid(g)
END DO
WRITE (ofile,10)
WRITE (ofile,*) '   Beta:            Group        Value'
WRITE (ofile,*) '                   -----    --------------'
DO l = 1,ndg
   WRITE (ofile,505) l,betal(l)
END DO
WRITE (ofile,10)
WRITE (ofile,*) 'Total Beta = ',beta
WRITE (ofile,10)
WRITE (ofile,*) '   Lambda:          Group        Value'
WRITE (ofile,*) '                   -----    --------------'
DO l = 1,ndg
   WRITE (ofile,505) l,lambda(l)
END DO
WRITE (ofile,10)
k = 1
WRITE (ofile,*) 'Material Properties at Beginning of Problem (uncorrected):'
WRITE (ofile,10)
DO q = 1,nxsets
   WRITE (ofile,500) 'Cross Section Set # ',q
   WRITE (ofile,10)
   WRITE (ofile,*) '  Sigma-tr:        Group        Value'
   WRITE (ofile,*) '                   -----    --------------'
   DO g = 1,ng
      WRITE (ofile,505) g,xstr(g,q,k)
   END DO
   WRITE (ofile,10)
   WRITE (ofile,*) '  nuSigma-fiss:    Group        Value'
   WRITE (ofile,*) '                   -----    --------------'
   DO g = 1,ng
      WRITE (ofile,505) g,nxsf(g,q,k)
   END DO
   WRITE (ofile,10)
   WRITE (ofile,*) '  Sigma-abs:       Group        Value'
   WRITE (ofile,*) '                   -----    --------------'
   DO g = 1,ng
      WRITE (ofile,505) g,xsab(g,q,k)
   END DO
   WRITE (ofile,10)
   WRITE (ofile,*) '  Sigma-scat: Group  to  Group      Value'
   WRITE (ofile,*) '              -----      -----   --------------'
   370
DO g = 1,ng
   DO h = 1,ng
      WRITE (ofile,510) g,h,xssc(h,g,q,k)
   END DO
END DO
WRITE (ofile,10)
END DO
DO k = 2,nstate+1
WRITE (ofile,10)
WRITE (ofile,*') 'Material Properties at End of State ',k-1
WRITE (ofile,10)
DO q = 1,nxsets
   WRITE (ofile,500) 'Cross Section Set # ',q
   WRITE (ofile,10)
   WRITE (ofile,505) g,xstr(g,q,k)
   END DO
   WRITE (ofile,10)
   WRITE (ofile,505) g,nxsf(g,q,k)
   END DO
   WRITE (ofile,10)
   WRITE (ofile,505) g,xsab(g,q,k)
   END DO
   WRITE (ofile,10)
   WRITE (ofile,505) g,xssc(h,g,q,k)
   END DO
END DO
WRITE (ofile,10)
WRITE (ofile,*) 'Ramp Increments:'
WRITE (ofile,10)
WRITE (ofile,10)
WRITE (ofile,*') 'Cell/XS Map:'
WRITE (ofile,10)
DO iii = 1,nz
   WRITE (ofile,111) 'Level # ',iii
   DO ii = ny,1,-1
      WRITE (ofile,*) (xsmap(iii,ii,i), i=1,nx)
   END DO
END DO
WRITE (ofile,10)
END DO
WRITE (ofile,30) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE (ofile,31) !
WRITE (ofile,32) !
WRITE (ofile,90) !
WRITE (ofile,32) !
WRITE (ofile,31) ! Write Problem Control Information
WRITE (ofile,30) !
WRITE (ofile,10)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! WRITE (ofile,600) 'Steady/Transient Flag = ',stflg
! WRITE (ofile,* ) ' Steady State Calculation'
! WRITE (ofile,10) ! WRITE (ofile,* ) ' Scalar Flux         Keff          Max #Outer       Max
#Inner'
! WRITE (ofile,* ) ' Convergence     Convergence     Iterations
Iterations'
! WRITE (ofile,* ) ' --------------   --------------    -------------    -------
------'
! WRITE (ofile,605) ccf,cck,iteromax,iterimax
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Format Statements
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
10 FORMAT(1x)
20 FORMAT(1x)
30 FORMAT(3x,'***************************')!
40 FORMAT(3x,'| System Geometry   |')!
50 FORMAT(3x,'| Quadrature Data |')!
60 FORMAT(3x,'| 3D Ray Tracing Data |')!
70 FORMAT(3x,'| Material Properties |')!
80 FORMAT(3x,'| Problem Control |')!
90 FORMAT(3x,'| Transient Control |')!
93 FORMAT(3x,'| Kinetics Properties |')!
100 FORMAT(4x,a22,i1)
105 FORMAT(4x,a18,es10.3,' cm')!
110 FORMAT(3x,a25,i3)
111 FORMAT(3x,a8,i3)
115 FORMAT(4x,a9,es10.3)!!!!!!!!!!!!!!
200 FORMAT(3x,'DIRECTION #',i2, OF ',i2,' :')!
205 FORMAT(14x,'Weight = ',es13.6)!
210 FORMAT(3x,a20,es13.6,a4)!
215 FORMAT(17x,a6,i4)!
300 FORMAT(4x,'Direction #',i2,16x,'Coupling')!
305 FORMAT(4x,'Track #    Length [cm] Fr    To')!
310 FORMAT(3x,'---------   --------------   ----  ----')!
315 FORMAT(6x,i2,7x,es13.6,3x,i4,2x,i4)!
400 FORMAT(3x,'Direction #',i2)!
405 FORMAT(4x,'Pl#',5x,'Tr#',5x,'Length [cm]')!
410 FORMAT(3x,'-----   -----   --------------')!
415 FORMAT(3x,i3,5x,i3,5x,es13.6)!
500 FORMAT(3x,a22,i3)!
505 FORMAT(20x,i3,6x,es13.6)!
510 FORMAT(20x,i3,8x,i3,5x,es13.6)!
600 FORMAT(3x,a24,i1)!
605 FORMAT(3x,es13.6,4x,es13.6,7x,i5,11x,i5)!
700 FORMAT(3x,'Direction #',i2)!
702 FORMAT(4x,'Pl#',5x,'Tr#',6x,'Length [cm]',3x,'Pl# To', &
    & 3x,'Tr# To',4x,'Pl# Fr',3x,'Tr# Fr')!
704 FORMAT(3x,'-----   -----   --------------  ------   ', &
    & '-----    ------   ------')!
710 FORMAT(3x,i3,5x,i3,5x,es13.6,4x,i3,6x,i3,7x,i3,6x,i3)!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END

SUBROUTINE initBD()
USE files    !
USE scalars  !
USE tracking !
USE neutrons !
USE xsects  !
!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER iterdum          !
INTEGER nset             !
DOUBLE PRECISION scat    !
DOUBLE PRECISION phi     !
DOUBLE PRECISION chi     !
DOUBLE PRECISION fiss    !
DOUBLE PRECISION cross   !
DOUBLE PRECISION sumb    !
DOUBLE PRECISION powsum  !
DOUBLE PRECISION vol     !
DOUBLE PRECISION pcsum   !
DOUBLE PRECISION btl     !
DOUBLE PRECISION lam     !
DOUBLE PRECISION pctmp   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
allocate transient arrays !
!
ALLOCATE (aphi(npomax,nazmax,nz,ny,nx,8*dirnum,ng))
ALLOCATE (aphi2(npomax,nazmax,nz,ny,nx,8*dirnum,ng))
ALLOCATE (sphi(nz,ny,nx,ng))
ALLOCATE (sphiold(nz,ny,nx,ng))
ALLOCATE (ssav(nz,ny,nx,ng))
ALLOCATE (fsav(nz,ny,nx,ng))
ALLOCATE (psav(nz,ny,nx,ng))
ALLOCATE (pconc(ndg,nz,ny,nx))
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! open steady-state restart file !
!!
OPEN (UNIT = rsfile, FILE = rfname, FORM = 'BINARY')
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! load keff and fluxes from restart file !
READ(rsfile) iterdum,kinf,keff
READ(rsfile) sphi
READ(rsfile) fd0
READ(rsfile) fd1
READ(rsfile) aphi
CLOSE (rsfile)

DO g = 1,ng
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        scat = 0.0
        nset = xsmap(iii,ii,i)
        DO h = 1,ng
          cross = xssc(g,h,nset,0)
          phi = sphi(iii,ii,i,h)
          scat = scat + cross * phi
        END DO
        ssav(iii,ii,i,g) = scat / fpi
      END DO
    END DO
  END DO
END DO

DO g = 1,ng
  chi = chip(g)
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        fiss = 0.0
        nset = xsmap(iii,ii,i)
        DO h = 1,ng
          cross = nxsf(h,nset,0)
          phi = sphi(iii,ii,i,h)
          fiss = fiss + cross * phi
        END DO
        fsav(iii,ii,i,g) = (chi / fpi) * (1 - beta) * fiss
      END DO
    END DO
  END DO
END DO

!!initialize delayed neutron precursor concentrations!!
!!

DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      sumb = 0.0
      nset = xsmapi(iii,ii,i)
      DO g = 1,ng
        cross = nxsf(g,nset,0)
        phi = sphi(iii,ii,i,g)
        sumb = sumb + cross \times phi
      END DO
      DO l = 1,ndg
        btl = betal(l)
        lam = lambda(l)
        pconc(l,iii,ii,i) = (btl / lam) \times sumb
      END DO
    END DO
  END DO
END DO

! **************************************************
! initialize delayed neutron source !
! **************************************************

DO g = 1,ng
  chi = chid(g)
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        pcsum = 0.0
        DO l = 1,ndg
          lam = lambda(l)
          pctmp = pconc(l,iii,ii,i)
          pcsum = pcsum + lam \times pctmp
        END DO
        psav(iii,ii,i,g) = (chi / fpi) \times pcsum
      END DO
    END DO
  END DO
END DO

! **************************************************
! initialize previous time-step scalar & angular flux !
! **************************************************

DO g = 1,ng
  dd = 0
  DO dq = 1,8
    DO d = 1,dirdnum
      dd = dd + 1
      DO i = 1,nx
        DO ii = 1,ny
          DO iii = 1,nz
            sphiold(iii,ii,i,g) = sphi(iii,ii,i,g)
            DO n = 1,nazmax
              DO p = 1,npomax
                aphi2(p,n,iii,ii,i,dd,g) = aphi(p,n,iii,ii,i,dd,g)
              END DO
            END DO
          END DO
        END DO
      END DO
    END DO
  END DO
END DO
calculate initial core power!

powsum = 0.0
DO g = 1,ng
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        vol = dx * dy * dz
        nset = xsmap(iii,ii,i)
        cross = nxsf(g,nset,0)
        phi = sphi(iii,ii,i,g)
        powsum = powsum + cross * phi * vol
      END DO
    END DO
  END DO
END DO
powinit = powsum
corpwr = 1.0

IF (critflg.eq.1) THEN
  DO k = 1,nstate+1
    DO q = 1,nxsets
      DO g = 1,ng
        cross = nxsf(g,q,k)
        nxsf(g,q,k) = cross / keff
      END DO
    END DO
  END DO
ENDIF

WRITE(*,*) '    Time           Power           Eigen        Iter.Clock'
WRITE(*,*) ' ----------     ----------     ------------    ------------'
WRITE(*,*) 0.0,corpwr,0.0,0.0
WRITE(ofile,*) '    Time           Power           Eigen        Iter.Clock'
WRITE(ofile,*) ' ----------     ----------     ------------    ------------'
WRITE(ofile,*) 0.0,corpwr,0.0,0.0
100 FORMAT(2x,es13.6,2x,es13.6,2x,es13.6,2x,es13.6)
RETURN
END
SUBROUTINE mainBD(estate)
  !
  !
  !!!!!!!!!!!!!!!!!!!!!
  USE files
  USE scalars
  USE tracking
  USE neutrons
  USE xsects
  !
  !!!!!!!!!!!!!!!!!!!!!
  INTEGER estate!
  !
  !!!!!!!!!!!!!!!!!!!!!
  LOGICAL timchk!
  LOGICAL rmpflg!
  !
  !!!!!!!!!!!!!!!!!!!!!
  INTEGER state!
  INTEGER stchgflg!
  INTEGER nset!
  !
  !!!!!!!!!!!!!!!!!!!!!
  DOUBLE PRECISION timlng!
  DOUBLE PRECISION timdiff!
  DOUBLE PRECISION ssize!
  DOUBLE PRECISION stend!
  DOUBLE PRECISION cross!
  DOUBLE PRECISION fiss!
  DOUBLE PRECISION pcsum!
  DOUBLE PRECISION pcold!
  DOUBLE PRECISION pcnew!
  DOUBLE PRECISION pctmp!
  DOUBLE PRECISION ptmp!
  DOUBLE PRECISION btl!
  DOUBLE PRECISION lam!
  DOUBLE PRECISION invlam!
  DOUBLE PRECISION dcy!
  DOUBLE PRECISION chi!
  DOUBLE PRECISION powsum!
  DOUBLE PRECISION vol!
  DOUBLE PRECISION phi!
  DOUBLE PRECISION phiold!
  !
  !!!!!!!!!!!!!!!!!!!!!
  INTEGER, ALLOCATABLE :: timarrbgn(:)!
  INTEGER, ALLOCATABLE :: timarrend(:)!
  DOUBLE PRECISION, ALLOCATABLE :: lxstr(:, :)
  DOUBLE PRECISION, ALLOCATABLE :: lnxsf(:, :)
  DOUBLE PRECISION, ALLOCATABLE :: lxssc(:, :, :)
  !
  !!!!!!!!!!!!!!!!!!!!!
  INTEGER g, h!
  INTEGER i, ii, iii!
INTEGER d, dq, dd
INTEGER n
INTEGER p
INTEGER l
INTEGER q

!!!!!!!!!!!!!!!!!!!!!!!!!

ALLOCATE (timarrbgn(8))
ALLOCATE (timarrend(8))
ALLOCATE (lxstr(ng,nxsets))
ALLOCATE (lnxsf(ng,nxsets))
ALLOCATE (lxssc(ng,ng,nxsets))

! initialize time step controls
!
 timchk = .true.
 state = 1
 ssize = delta(state)
 stend = timend(state)
 outtim = ssize
 timlng = ssize
 rmpflg = .false.
 stchgflg = 0

! initialize local material property arrays
!
 DO q = 1,nxsets
  DO g = 1,ng
   lxstr(g,q) = xstr(g,q,state)
   lnxsf(g,q) = nxsf(g,q,state)
   DO h = 1,ng
    lxssc(h,g,q) = xssc(h,g,q,state)
   END DO
  END DO
 END DO

DO WHILE (timchk)
!
 CALL DATE_AND_TIME(VALUES = timarrbgn)
!
! load material beginning-of-state material configuration
!
 IF (stchgflg.eq.1) THEN
  DO q = 1,nxsets
   DO g = 1,ng
    lxstr(g,q) = xstr(g,q,state)
    lnxsf(g,q) = nxsf(g,q,state)
   END DO
  END DO
 ENDIF
!
! update material properties if ramp transient
!
 IF (rmpflg) THEN
  IF (tdramp.eq.1) THEN
   DO q = 1,nxsets
    DO g = 1,ng
     lxstr(g,q) = lxstr(g,q) + ramptr(g,q,state)
     lnxsf(g,q) = lnxsf(g,q) + rampnf(g,q,state)
    END DO
   END DO
  ENDIF
! lxxsc(h,g,q) = lxxsc(h,g,q) + rampss(h,g,q,state)
END DO
END DO
ENDIF
ENDIF
!
! call inner iteration
!
IF (trflg.eq.1) THEN
CALL inBDcc(lxstr,lxssc,ssize)
ELSE IF (trflg.eq.2) THEN
CALL inBDcl(lxstr,lxssc,ssize)
ELSE IF (trflg.eq.3) THEN
CALL inBDce(lxstr,lxssc,ssize)
ELSE
! trap
ENDIF
!
! update prompt fission source
!
DO g = 1,ng
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
nset = xsmap(iii,ii,i)
fiss = 0.0
DO h = 1,ng
cross = lnxsf(h,nset)
phi = sphi(iii,ii,i,h)
fiss = fiss + cross * phi
END DO
fsav(iii,ii,i,g) = (1 - beta) * (fiss / fpi) * chip(g)
END DO
END DO
END DO
!
! update precursor concentrations
!
IF (pcflg.eq.0) THEN
!
! time integrated
!
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
nset = xsmap(iii,ii,i)
DO l = 1,ndg
btl = betal(l)
lam = lambda(l)
invlam = 1.0 / lam
dcy = exp(-(lam * ssize))
pcold = pconc(l,iii,ii,i)
pcsum = 0.0
DO g = 1,ng
cross = lnxsf(h,nset)
phi = sphi(iii,ii,i,h)
phiold = sphiold(iii,ii,i,h)
pcsum = pcsum + cross*invlam*(((1.0 - ((1.0 - dcy) &
& / (lam * ssize))) * phi) + (((1.0 - dcy) / &
& (lam * ssize)) - dcy)) * phiold)
END DO
END
pctmp = pcold * dcy + btl * pcsum
pconc(l,iii,ii,i) = pctmp
END DO
END DO
END DO
ELSE IF (pcflg.eq.1) THEN
! fully explicit
!
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
nset = xsmap(iii,ii,i)
fiss = 0.0
DO h = 1,ng
cross = lnxsf(h,nset)
phi = sphi(iii,ii,i,h)
fiss = fiss + cross * phi
END DO
DO l = 1,ndg
btl = betal(l)
lam = lambda(l)
pcold = pconc(l,iii,ii,i)
pcnew = (1.0 - (lam*ssize)) * pcold + (btl*ssize*fiss)
pconc(l,iii,ii,i) = pcnew
END DO
END DO
END DO
ELSE
! other
ENDIF
!
! update delay source
!
DO g = 1,ng
chi = chid(g)
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
pcsum = 0.0
DO l = 1,ndg
lam = lambda(l)
pctmp = pconc(l,iii,ii,i)
pcsum = pcsum + lam * pctmp
END DO
ptmp = (chi / fpi) * pcsum
psav(iii,ii,i,g) = ptmp
END DO
END DO
END DO
END DO
!
calculate core power (total neutron production)
! - normalized to t = 0.0sec
!
powsum = 0.0
DO g = 1,ng
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
vol = dx * dy * dz
nset = xsmap(iii,ii,i)
cross = lnxsf(g,nset)
phi = sphi(iii,ii,i,g)
powsum = powsum + cross * phi * vol
END DO
END DO
END DO
END DO
corpwr = powsum / powinit
!
! calculate elapsed cpu for this time step
!
CALL DATE_AND_TIME(VALUES = timarrend)
CALL diffTime(timarrbgn,timarrend,timdiff)
!
! short output
!
WRITE(*,100) outtim,corpwr,timdiff
WRITE(ofile,100) outtim,corpwr,timdiff
!
! long output
!
IF (timlng.ge.lngtim(state)) THEN
!     CALL writeLong()
timlng = 0.0
ENDIF
!
! update previous time-step scalar & angular flux
!
DO g = 1,ng
  dd = 0
  DO dq = 1,8
    DO d = 1,dirnum
      dd = dd + 1
      DO i = 1,nx
        DO ii = 1,ny
          DO iii = 1,nz
            sphiold(iii,ii,i,g) = sphi(iii,ii,i,g)
            DO n = 1,nazmax
              DO p = 1,npomax
                aphi2(p,n,iii,ii,i,dd,g) = aphi(p,n,iii,ii,i,dd,g)
              END DO
            END DO
          END DO
        END DO
      END DO
    END DO
  END DO
END DO
!
! increment to next time step
!
outtim = outtim + ssize
timlng = timlng + ssize
rmpflg = .true.
!
! check for end of state and/or problem
!
stchgflg = 0
IF (outtim.gt.stend) THEN
  state = state + 1
  IF (state.gt.nstate) THEN
    timchk = .false.
  END IF
END IF
ELSE
    rmpflg = .false.
    ssize = delta(state)
    stend = timend(state)
    stchgflg = 1
    CALL writeEOS(state)
ENDIF
ENDIF

END DO

100 FORMAT(2x,es13.6,2x,es13.6,2x,es13.6)

estate = state
IF (estate.gt.nstate) THEN
  estate = nstate
ENDIF
RETURN
END

SUBROUTINE inBDcc(lxstr,lxssc,ssize)

!!!!!!!!!!!!!!!!!!!!!!!
USE files
USE scalars
USE tracking
USE neutrons
USE xsects
!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION lxstr(ng,nxsets) !
DOUBLE PRECISION lxssc(ng,ng,nxsets) !
DOUBLE PRECISION ssize               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
LOGICAL conva !
LOGICAL convs !
LOGICAL maxin !
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!
INTEGER nnxtmp !
INTEGER nnytmp !
INTEGER nnstmp !
INTEGER naztmp !
INTEGER npltmp !
INTEGER nset   !
!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION velt       !
DOUBLE PRECISION cnvaf      !
DOUBLE PRECISION cnvsf      !
DOUBLE PRECISION aphin      !
DOUBLE PRECISION source     !
DOUBLE PRECISION cross      !
DOUBLE PRECISION track      !
DOUBLE PRECISION aphiout    !
DOUBLE PRECISION aphiold    !
DOUBLE PRECISION aphilast   !
DOUBLE PRECISION sphold     !
DOUBLE PRECISION sphisum    !
DOUBLE PRECISION numsum     !
DOUBLE PRECISION densum     !
DOUBLE PRECISION weight     !
DOUBLE PRECISION trsep      !
DOUBLE PRECISION travg      !
DOUBLE PRECISION aphiavg    !
DOUBLE PRECISION sphnew     !
DOUBLE PRECISION cnvtmp     !
DOUBLE PRECISION scat       !
DOUBLE PRECISION phi        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER g, h       !
INTEGER d, dq, dd  !
INTEGER i, ii, iii !
INTEGER n, p       !
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! BEGIN Inner Iteration  !
!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
  conva = .true.
  convs = .true.
  maxin = .true.
  iterin = 0
  velt = vel(g) * ssize
  DO WHILE ((convs.and.conva).and.maxin)
    cnvaf = 0.0
    cnvsf = 0.0
    iterin = iterin + 1
    IF (iterin.ge.iterimax) THEN
      maxin = .false.
    ENDIF
  ENDDO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Calculate Angular Flux Distribution  !
!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO i = 1,ng
  DO ii = 1,ny
    DO iii = 1,nz
      source = ssav(iii,ii,i,g)+fsav(iii,ii,i,g)+psav(iii,ii,i,g)
      nset = xsmap(iii,ii,i)
      cross = lxstr(g,nset)
      dd = 0
      DO dq = 1,8
        DO d = 1,dirnum
          dd = dd + 1
          nxxtmp = dirset(d)%nnx
          nnytmp = dirset(d)%nny
          naztmp = nxxtmp + nnytmp
          DO n = 1,naztmp

383
npltmp = npl(n,d)
DO p = 1,npltmp
    aphiold = aphi2(p,n,iii,ii,i,dd,g)
    aphiout = aphiold - velt*((cross*aphiold) - source)
    IF (aphiout.lt.aphimin) THEN
        aphi(p,n,iii,ii,i,dd,g) = aphimin
    ELSE
        aphilast = aphi(p,n,iii,ii,i,dd,g)
        cnvttmp = abs((aphiout - aphilast) / aphiout)
        IF (cnvttmp.gt.cnvaf) THEN
            cnvaf = cnvttmp
        ENDIF
        ENDIF
        aphi(p,n,iii,ii,i,dd,g) = aphiout
    END DO
END DO
END DO
END DO
END DO
END DO

! Calculate Scalar Flux !

DO i = 1,nx
    DO ii = 1,ny
        DO iii = 1,nz
            sphold = sphi(iii,ii,i,g)
            dd = 0
            sphisum = 0.0
            source = ssav(iii,ii,i,g)+fsav(iii,ii,i,g)+psav(iii,ii,i,g)
            nset = xsmap(iii,ii,i)
            cross = lxstr(g,nset)
            DO dq = 1,8
                DO d = 1,dirnum
                    dd = dd + 1
                    numsum = 0.0
                    densum = 0.0
                    weight = dirset(d)%wght
                    trsep = dirset(d)%dazi * dirset(d)%dpol
                    nnxtmp = dirset(d)%nnx
                    nnytmp = dirset(d)%nny
                    naztmp = nnxtmp + nnytmp
                    nnstmp = dirset(d)%nns
                    DO n = 1,naztmp
                        npltmp = npl(n,d)
                        DO p = 1,npltmp
                            track = zray(p,n,dd,g,dq,d, &
                            & nnytmp,nnstmp,npltmp,aphin)
                            aphiout = aphi(p,n,iii,ii,i,dd,g)
                            travg = (source / cross) + ((aphin - aphiout) &
                            & / (cross*track))
                            numsum = numsum + travg * track * trsep
                            densum = densum + track * trsep
                        END DO
                    END DO
                END DO
            END DO
            aphiavg = numsum / densum
            sphisum = sphisum + aphiavg * weight
        END DO
    END DO
END DO

384
sphnew = sphisum * fpi
sphi(iii,ii,i,g) = sphnew
cnvtmp = abs((sphnew - sphold) / sphnew)
IF (cnvtmp.gt.cnvsf) THEN
  cnvsf = cnvtmp
ENDIF
END DO
END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Calculate Scattering Source
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      scat = 0.0
      nset = xsmap(iii,ii,i)
      DO h = 1,ng
        phi = sphi(iii,ii,i,h)
        cross = lxssc(g,h,nset)
        scat = scat + cross * phi
      END DO
      ssav(iii,ii,i,g) = scat / fpi
    END DO
  END DO
END DO
!
! check angular flux convergence...
!
  IF (cnvaf.lt.cca) THEN
    conva = .false.
  END IF
!
! check scalar flux convergence...
!
  IF (cnvsf.lt.ccs) THEN
    convs = .false.
  END IF
!
END DO
END DO
!
RETURN
END

SUBROUTINE inBDcl(lxstr,lxssc,ssize)
!
!!!!!!!!!!!!!!
USE files
USE scalars
USE tracking
USE neutrons
USE xsects
!!!!!!!!!!!!!!
!
DOUBLE PRECISION lxstr(ng,nxsets) !
DOUBLE PRECISION lxssc(ng,ng,nxsets) !
DOUBLE PRECISION ssize               !

LOGICAL conva !
LOGICAL convs !
LOGICAL maxin !

INTEGER nnxtmp !
INTEGER nnytmp !
INTEGER nnstmp !
INTEGER naztmp !
INTEGER npltmp !
INTEGER nset   !

DOUBLE PRECISION velt       !
DOUBLE PRECISION cnvaf      !
DOUBLE PRECISION cnvsf      !
DOUBLE PRECISION aphin      !
DOUBLE PRECISION source     !
DOUBLE PRECISION cross      !
DOUBLE PRECISION track      !
DOUBLE PRECISION aphioout   !
DOUBLE PRECISION aphiod     !
DOUBLE PRECISION aphiast    !
DOUBLE PRECISION sphold     !
DOUBLE PRECISION sphisum    !
DOUBLE PRECISION numsum     !
DOUBLE PRECISION densum     !
DOUBLE PRECISION weight     !
DOUBLE PRECISION trsep      !
DOUBLE PRECISION travg      !
DOUBLE PRECISION aphiavg    !
DOUBLE PRECISION sphnew     !
DOUBLE PRECISION cnvtmp     !
DOUBLE PRECISION scat       !
DOUBLE PRECISION phi        !

INTEGER g, h       !
INTEGER d, dq, dd  !
INTEGER i, ii, iii !
INTEGER n, p       !

DO g = 1,ng
  conva = .true.
  convs = .true.
  maxin = .true.
END
iterin = 0
velt = vel(g) * ssize
DO WHILE ((convs.and.conva).and.maxin)
    cnvaf = 0.0
    cnvsf = 0.0
    iterin = iterin + 1
    IF (iterin.ge.iterimax) THEN
        maxin = .false.
    ENDIF
    !
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !
    !   Calculate Angular Flux Distribution   
    !
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !
    DO i = 1,ng
        DO ii = 1,ny
            DO iii = 1,nz
                source = ssav(iii,ii,i,g)+fsav(iii,ii,i,g)+psav(iii,ii,i,g)
                nset = xsmap(iii,ii,i)
                cross = lxstr(g,nset)
                dd = 0
                DO dq = 1,8
                    DO d = 1,dirnum
                        dd = dd + 1
                        nnxtmp = dirset(d)%nnx
                        nnytmp = dirset(d)%nny
                        naztmp = nnxtmp + nnytmp
                        nnstmp = dirset(d)%nns
                        DO n = 1,naztmp
                            npltmp = npl(n,d)
                            DO p = 1,npltmp
                                track = zray(p,n,d)%trcln
                                aphiold = aphi2(p,n,iii,ii,i,dd,g)
                                CALL findAphin2(p,n,iii,ii,i,dd,g,dq,d, &
                                    & nnytmp,nnstmp,npltmp,aphin)
                                aphiout = aphiold - velt * ((cross * aphiold) &
                                    + ((aphiold - aphin) / track) - source)
                                IF (aphiout.lt.aphimin) THEN
                                    aphi(p,n,iii,ii,i,dd,g) = aphimin
                                ELSE
                                    aphilast = aphi(p,n,iii,ii,i,dd,g)
                                    cnvtmp = abs((aphiout - aphilast) / aphiout)
                                    IF (cnvtmp.gt.cnvaf) THEN
                                        cnvaf = cnvtmp
                                    ENDIF
                                ENDIF
                                aphi(p,n,iii,ii,i,dd,g) = aphiout
                            END DO
                        END DO
                    END DO
                END DO
            END DO
        END DO
    END DO
    !
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !
    !   Calculate Scalar Flux   
    !
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      sphold = sphi(iii,ii,i,g)
      dd = 0
      sphisum = 0.0
      source = ssav(iii,ii,i,g)+fsav(iii,ii,i,g)+psav(iii,ii,i,g)
      nset = xsmap(iii,ii,i)
      cross = lxstr(g,nset)
      DO dq = 1,8
        DO d = 1,dirnum
          dd = dd + 1
          numsum = 0.0
          densum = 0.0
          weight = dirset(d)%wgght
          trsep = dirset(d)%dazi * dirset(d)%dpol
          nnxtmp = dirset(d)%nnx
          nnytmp = dirset(d)%nny
          naztmp = nnxtmp + nnytmp
          nnstmp = dirset(d)%nns
          DO n = 1,naztmp
            npltmp = npl(n,d)
            DO p = 1,npltmp
              track = zray(p,n,d)%trcln
              CALL findAphin(p,n,iii,ii,i,dd,g,dq,d, &
                            nnytmp,nnxtmp,npltmp,aphin)
              aphiout = aphi(p,n,iii,ii,i,dd,g)
              travg = (source / cross) + ((aphi - aphiout) &
                             / (cross * track))
              numsum = numsum + travg * track * trsep
              densum = densum + track * trsep
            END DO
          END DO
          aphiavg = numsum / densum
          sphisum = sphisum + aphiavg * weight
        END DO
      END DO
      sphnew = sphisum * fpi
      sphi(iii,ii,i,g) = sphnew
      cnvtmp = abs((sphnew - sphold) / sphnew)
      IF (cnvtmp.gt.cnvsf) THEN
        cnvsf = cnvtmp
      ENDIF
    END DO
  END DO
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                 !
!   Calculate Scattering Source   !
!                                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      scat = 0.0
      nset = xsmap(iii,ii,i)
      DO h = 1,ng
        phi = sphi(iii,ii,i,h)
        cross = lxssc(g,h,nset)
        scat = scat + cross * phi
      END DO
    END DO
  END DO
END DO
ssav(iii,ii,i,g) = scat / fpi
END DO
END DO
END DO
!
! check angular flux convergence...
!
IF (cnvaf.lt.cca) THEN
  conva = .false.
END IF
!
! check scalar flux convergence...
!
IF (cnvsf.lt.ccs) THEN
  convs = .false.
END IF
!
END DO
END DO
!
RETURN
END

SUBROUTINE inBDce(lxstr,lxssc,ssize)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
USE files
USE scalars
USE tracking
USE neutrons
USE xsects
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION lxstr(ng,nxsets) !
DOUBLE PRECISION lxssc(ng,ng,nxsets) !
DOUBLE PRECISION ssize !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!
LOGICAL conva !
LOGICAL convs !
LOGICAL maxin !
!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER nnxtmp !
INTEGER nnytmp !
INTEGER nnstmp !
INTEGER naztmp !
INTEGER npltmp !
INTEGER nset   !
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION velt       !
DOUBLE PRECISION cnvaf      !
DOUBLE PRECISION cnvsf      !
DOUBLE PRECISION aphin      !
DOUBLE PRECISION source     !
DOUBLE PRECISION cross      !

389
DOUBLE PRECISION track!
DOUBLE PRECISION aphiout!
DOUBLE PRECISION aphiold!
DOUBLE PRECISION aphilast!
DOUBLE PRECISION sphold!
DOUBLE PRECISION sphisum!
DOUBLE PRECISION numsum!
DOUBLE PRECISION densum!
DOUBLE PRECISION weight!
DOUBLE PRECISION trsep!
DOUBLE PRECISION travg!
DOUBLE PRECISION aphiavg!
DOUBLE PRECISION sphnew!
DOUBLE PRECISION cnvtmp!
DOUBLE PRECISION scat!
DOUBLE PRECISION phi!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER g, h!
INTEGER d, dq, dd!
INTEGER i, ii, iii!
INTEGER n, p!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
BEGIN Inner Iteration!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DO g = 1,ng
conva = .true.
convs = .true.
maxin = .true.
iterin = 0
velt = vel(g) * ssize
DO WHILE ((convs.and.conva).and.maxin)
cnvaf = 0.0
cnvsf = 0.0
iterin = iterin + 1
IF (iterin.ge.iterimax) THEN
  maxin = .false.
ENDIF

!! Calculate Angular Flux Distribution !
!! !
!! !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DO i = 1,ng
  DO ii = 1,ny
    DO iii = 1,nz
      source = ssav(iii,ii,i,g)+fsav(iii,ii,i,g)+psav(iii,ii,i,g)
      nset = xsmap(iii,ii,i)
      cross = lxstr(g,nset)
      dd = 0
      DO dq = 1,8
        DO d = 1,dirnum
          dd = dd + 1
          nnxtmp = dirset(d)%nnx
          nnytmp = dirset(d)%nny
          !
          !


naztmp = nnxtmp + nnytmp
nnstmp = dirset(d)%nns
DO n = 1,naztmp
   npltmp = npl(n,d)
   DO p = 1,npltmp
      track = zray(p,n,d)%trcln
      aphiold = aphi2(p,n,iii,ii,i,dd,g)
      aphiout = aphiold - velt * ((cross + (1.0 / &
         track)) * aphiold) - source)
      IF (aphiout.lt.aphimin) THEN
         aphi(p,n,iii,ii,i,dd,g) = aphimin
      ELSE
         aphilast = aphi(p,n,iii,ii,i,dd,g)
         cnvtmp = abs((aphiout - aphilast) / aphiout)
         IF (cnvtmp.gt.cnvaf) THEN
            cnvaf = cnvtmp
         ENDIF
      ENDIF
      aphi(p,n,iii,ii,i,dd,g) = aphiout
   END DO
   END DO
END DO
END DO
END DO
END DO
END DO

!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                           !
!   Calculate Scalar Flux     !
!                           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO i = 1,nx
   DO ii = 1,ny
      DO iii = 1,nz
         sphold = sphi(iii,ii,i,g)
         dd = 0
         sphisum = 0.0
         source = ssav(iii,ii,i,g)+fsav(iii,ii,i,g)+psav(iii,ii,i,g)
         nset = xsmap(iii,ii,i)
         cross = lxstr(g,nset)
         DO dq = 1,8
            DO d = 1,dirnum
               dd = dd + 1
               numsum = 0.0
               densum = 0.0
               weight = dirset(d)%wght
               trsep = dirset(d)%dazi * dirset(d)%dpol
               nnxtmp = dirset(d)%nnx
               nnytmp = dirset(d)%nny
               naztmp = nnxtmp + nnytmp
               nnstmp = dirset(d)%nns
               DO n = 1,naztmp
                  npltmp = npl(n,d)
                  DO p = 1,npltmp
                     track = zray(p,n,d)%trcln
                     CALL findAphin(p,n,iii,ii,i,dd,g,dq,d, &
                        nnytmp,nnstmp,npltmp,aphin)
                     aphiout = aphi(p,n,iii,ii,i,dd,g)
                     travg = (source / cross) + ((aphin - aphiout) &
                        / (cross * track))
                     numsum = numsum + travg * track * trsep
                  END DO
                  END DO
               END DO
               END DO
            END IF
         ENDIF
      END DO
   END DO
END DO

DENSUM = DENSUM + TRACK * TRSEP
END DO
END DO
APHI AVERAGE = NUMSUM / DENSUM
SPHISUM = SPHISUM + APHI AVERAGE * WEIGHT
END DO
END DO
SPH NEW = SPHISUM * FPI
SPHI(iii, ii, i, g) = SPH NEW
CNVTMP = ABS((SPH NEW - SPOHLD) / SPH NEW)
IF (CNVTMP.GT.CNVSF) THEN
    CNVSF = CNVTMP
END IF
END DO
END DO
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                  !
!   Calculate Scattering Source    !
!                                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO i = 1, NX
    DO ii = 1, NY
        DO iii = 1, NZ
            SCAT = 0.0
            NSET = XSMAP(iii, ii, i)
            DO h = 1, NG
                PHI = SPHI(iii, ii, i, h)
                CROSS = LXSSC(g, h, NSET)
                SCAT = SCAT + CROSS * PHI
            END DO
            SSADV(iii, ii, i, g) = SCAT / FPI
        END DO
    END DO
END DO

! check angular flux convergence...
! IF (CNVAF.LT.CCA) THEN
    CONVA = .FALSE.
END IF
!
! check scalar flux convergence...
! IF (CNVSF.LT.CCS) THEN
    CONVS = .FALSE.
END IF
!
END DO
END DO
!
RETURN
END

SUBROUTINE INITFD()
!
!!!!!!!!!!!!
USE FILES
USE SCALARS
USE tracking !
USE neutrons !
USE xsects !
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER iterdum !
INTEGER nnxtmp !
INTEGER nnymtmp !
INTEGER nactmp !
INTEGER nnstmp !
INTEGER npltmp !
INTEGER nset !
DOUBLE PRECISION scat !
DOUBLE PRECISION phi !
DOUBLE PRECISION chi1 !
DOUBLE PRECISION chi2 !
DOUBLE PRECISION fiss !
DOUBLE PRECISION cross !
DOUBLE PRECISION sumb !
DOUBLE PRECISION powsum !
DOUBLE PRECISION vol !
DOUBLE PRECISION psum !
DOUBLE PRECISION btl !
DOUBLE PRECISION lam !
DOUBLE PRECISION pctmp !
DOUBLE PRECISION invelt !
DOUBLE PRECISION source !
DOUBLE PRECISION track !
DOUBLE PRECISION aphin !
DOUBLE PRECISION aphicout !
DOUBLE PRECISION travg !
DOUBLE PRECISION fsnew !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER g, h !
INTEGER l !
INTEGER d, dq, dd !
INTEGER i, ii, iii !
INTEGER n !
INTEGER p !
INTEGER q !
INTEGER k !
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! allocate transient arrays !
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (aphi(npomax,nazmax,nz,ny,nx,8*dirnum,ng))
ALLOCATE (aphi2(npomax,nazmax,nz,ny,nx,8*dirnum,ng))
ALLOCATE (sphi(nz,ny,nx,ng))
ALLOCATE (ssav(nz,ny,nx,ng))
ALLOCATE (fsav(nz,ny,nx,ng))
ALLOCATE (psav(nz,ny,nx,ng))
ALLOCATE (pconc(ndg,nz,ny,nx))
ALLOCATE (pconold(ndg,nz,ny,nx))
ALLOCATE (ncp(npomax,nazmax,nxsets,dirnum,ng))
ALLOCATE (fd0(nz,ny,nx))
ALLOCATE (fd1(nz,ny,nx))
OPEN (UNIT = rsfile, FILE = rsfname, FORM = 'BINARY')

READ(rsfile) iter dum, kinf, keff
READ(rsfile) sphi
READ(rsfile) f d0
READ(rsfile) f d1
READ(rsfile) aphi
CLOSE (rsfile)

IF (critflg.eq.1) THEN
  DO k = 1,nstate+1
    DO q = 1,nxsets
      DO g = 1,ng
        cross = nxsf(g,q,k)
        nxsf(g,q,k) = cross / keff
      END DO
    END DO
  END DO
ENDIF

DO g = 1,ng
  DO i = 1,nx
    DO ii = 1,ny
      scat = 0.0
      nset = xsm ap(iii,ii,i)
      DO h = 1,ng
        cross = xss c(g,h,nset,0)
        phi = sphi(iii,ii,i,h)
        scat = scat + cross * phi
      END DO
      ssav(iii,ii,i,g) = scat / fpi
    END DO
  END DO
END DO
END DO
Rebuild Prompt Fission Source!

! initialize delayed neutron precursor concentrations!

DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      fiss = 0.0
      DO h = 1,ng
        cross = nxsf(h,nset,0)
        phi = sphi(iii,ii,i,h)
        fiss = fiss + cross * phi
      END DO
      fsnew = (chi1 / fpi) * (1 - beta) * fiss
      fsav(iii,ii,i,g) = fsnew
    END DO
  END DO
END DO

! initialize delayed neutron source!

DO g = 1,ng
  chi2 = chid(g)
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        pcsum = 0.0
        nset = xsmap(iii,ii,i)
        fiss = 0.0
        DO h = 1,ng
          cross = nxsf(h,nset,0)
          phi = sphi(iii,ii,i,h)
          pcsum = pcsum + cross * phi
        END DO
        btl = betal(l)
        lam = lambda(l)
        pctmp = (btl / lam) * pcsum
        pconc(l,iii,ii,i) = pctmp
        pconold(l,iii,ii,i) = pctmp
      END DO
    END DO
  END DO
END DO
DO 1 = 1, ndg
   lam = lambda(l)
   pctmp = pconc(l, iii, ii, i)
   pcsum = pcsum + lam * pctmp
END DO
psav(iii, ii, i, g) = (chi2 / fpi) * pcsum
END DO
END DO
END DO
END DO
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Update Fixed Source Eigenvalue 
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DO i = 1, nx
   DO ii = 1, ny
      DO iii = 1, nz
         nset = xsmap(iii, ii, i)
         DO g = 1, ng
            cross = nxsf(g, nset, 0)
            phi = sphi(iii, ii, i, g)
            fd0(iii, ii, i) = phi * cross
            fd1(iii, ii, i) = fd0(iii, ii, i)
         END DO
      END DO
   END DO
END DO

END DO

DO 1 = 1, nx
   DO ii = 1, ny
      DO iii = 1, nz
         invelt = 1.0 / (vel(g) * delta(1)) 
         DO g = 1, ng
            DO d = 1, dirnum 
               DO dq = 1, 8
                  track = zray(p, n, d)
               END DO
            END DO
         END DO
      END DO
   END DO
END DO

IF (trflg.eq.4) THEN

! Constant-Constant:
! APHI2 holds the previous time step track-average angular flux

DO g = 1, ng
   DO i = 1, nx
      DO ii = 1, ny
         DO iii = 1, nz
            dd = 0
            source = ssav(iii, ii, i, g) + fsav(iii, ii, i, g) + psav(iii, ii, i, g)
            nset = xsmap(iii, ii, i)
            cross = xstr(g, nset, 0)
            DO dq = 1, 8
               track = zray(p, n, d)
            END DO
         END DO
      END DO
   END DO
END DO

DO d = 1, dirnum 
   dd = dd + 1 
   nnxtmp = dirset(d)%nnx
   nnytmp = dirset(d)%nny
   nstmp = nnxtmp + nnytmp
   nns = dirset(d)%nns
   npltmp = npl(n, d)
   DO p = 1, npltmp 
      track = zray(p, n, d)
CALL findAphin(p,n,iii,ii,i,dd,g,dq,d, &
  nnytmp,nnstmp,npltmp,aphin)
aphiout = aphi(p,n,iii,ii,i,dd,g)
travg = (source / cross) + ((aphin - aphiout) &
  / (cross * track))
aphi2(p,n,iii,ii,i,dd,g) = travg
END DO
END DO
END DO
END DO
END DO
!
ELSE IF ((trflg.eq.5).or.(trflg.eq.6)) THEN
!
! Constant-Linear and Constant-Exponential:
! APHI2 holds the previous time step track-average angular flux
!
DO g = 1,ng
  dd = 0
  DO dq = 1,8
    DO d = 1,dirnum
      dd = dd + 1
      DO i = 1,nx
        DO ii = 1,ny
          DO iii = 1,nz
            DO n = 1,nazmax
              DO p = 1,npomax
                aphi2(p,n,iii,ii,i,dd,g) = aphi(p,n,iii,ii,i,dd,g)
              END DO
            END DO
          END DO
        END DO
      END DO
    END DO
  END DO
END DO
ENDIF
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate initial core power !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
powsum = 0.0
DO g = 1,ng
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        vol = dx * dy * dz
        source = fsav(iii,ii,i,g)
powsum = powsum + source * vol
      END DO
    END DO
  END DO
powinit = powsum
corpwr = 1.0
!
SUBROUTINE mainFD(estate)
!
!!!!!!!!!!!!!!!!!
USE files
USE scalars
USE tracking
USE neutrons
USE xsects
!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!
INTEGER estate !
!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!
INTEGER state    !
INTEGER stchgflg !
INTEGER nset     !
INTEGER nnxtmp   !
INTEGER nnytmp   !
INTEGER naztmp   !
INTEGER nnstmp   !
INTEGER npltmp   !
!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION ssize   !
DOUBLE PRECISION stend   !
DOUBLE PRECISION timdiff !
DOUBLE PRECISION cross   !
DOUBLE PRECISION powsum  !
DOUBLE PRECISION vol     !
DOUBLE PRECISION invelt  !
DOUBLE PRECISION source  !
DOUBLE PRECISION track   !
DOUBLE PRECISION aphin   !
DOUBLE PRECISION aphout  !
DOUBLE PRECISION travg   !
DOUBLE PRECISION fiss    !
DOUBLE PRECISION phi     !
DOUBLE PRECISION btl     !
DOUBLE PRECISION lam     !
DOUBLE PRECISION gamma   !
DOUBLE PRECISION pcold   !
DOUBLE PRECISION pcnew   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, ALLOCATABLE :: timarrbgn(:)          !
INTEGER, ALLOCATABLE :: timarrend(:)          !
DOUBLE PRECISION, ALLOCATABLE :: lxstr(:,:)   !
DOUBLE PRECISION, ALLOCATABLE :: lnxsf(:,:)   !
DOUBLE PRECISION, ALLOCATABLE :: lxssc(:,:,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER g, h       !
INTEGER i, ii, iii !
INTEGER d, dq, dd  !
INTEGER n          !
INTEGER p          !
INTEGER l          !
INTEGER q          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
ALLOCATE (timarrbgn(8))
ALLOCATE (timarrend(8))
ALLOCATE(lxstr(ng,nxsets))
ALLOCATE(lnxsf(ng,nxsets))
ALLOCATE(lxssc(ng,ng,nxsets))
!
! Initialize Time Step Counters, Flags, and Indices
!
timchk = .true.
state = 1
ssize = delta(state)
stend = timend(state)
outtim = ssize
rmpflg = .false.
stchgflg = 0
!
! initialize local material property arrays
!
DO q = 1,nxsets
  DO g = 1,ng
    lxstr(g,q) = xstr(g,q,state)
    lnxsf(g,q) = nxsf(g,q,state)
  END DO
  DO h = 1,ng
    lxssc(h,g,q) = xssc(h,g,q,state)
  END DO
END DO
!
! initialize non-collision probability table
!
DO g = 1,ng
  invelt = 1.0 / (vel(g) * ssize)
  DO d = 1,dirnum
    nnxtmp = dirset(d)%nnx
    nnytmp = dirset(d)%nny
    naztmp = nnxtmp + nnytmp
    DO q = 1,nxsets
      cross = lxstr(g,q)
DO n = 1,naztmp
  npltmp = npl(n,d)
  DO p = 1,npltmp
    track = zray(p,n,d)%trcln
    ncp(p,n,q,d,g) = exp(-(invelt + cross) * track)
  END DO
  END DO
  END DO
  END DO
  END DO
!
DO WHILE (timchk)
!
  CALL DATE_AND_TIME(VALUES = timarrbgn)
!
  ! load material beginning-of-state material configuration
  !
  IF (stchgflg.eq.1) THEN
  !
  DO q = 1,nxsets
    DO g = 1,ng
      lxstr(g,q) = xstr(g,q,state)
      lnxsf(g,q) = nxsf(g,q,state)
    END DO
    DO h = 1,ng
      lxssc(h,g,q) = xssc(h,g,q,state)
    END DO
  END DO
  !
  ! update non-collision probability table
  !
  DO g = 1,ng
    invelt = 1.0 / (vel(g) * ssize)
    DO d = 1,dirnum
      nnxtmp = dirset(d)%nnx
      nnytmp = dirset(d)%nny
      naztmp = nnxtmp + nnytmp
      DO q = 1,nxsets
        cross = lxstr(g,q)
        DO n = 1,naztmp
          npltmp = npl(n,d)
          DO p = 1,npltmp
            track = zray(p,n,d)%trcln
            ncp(p,n,q,d,g) = exp(-(invelt + cross) * track)
          END DO
        END DO
      END DO
    END DO
  END DO
  !
  ENDIF
  !
  ! update material properties if ramp transient
  !
  IF (rmpflg) THEN
    IF (tdramp.eq.1) THEN
      DO q = 1,nxsets
        DO g = 1,ng
          lxstr(g,q) = lxstr(g,q) + ramptr(g,q,state)
          lnxsf(g,q) = lnxsf(g,q) + rampnf(g,q,state)
        DO h = 1,ng
          lxssc(h,g,q) = lxssc(h,g,q) + rampss(h,g,q,state)
        END DO
      END DO
    ENDIF
  END DO
CALL outFDeig(lxstr,lnxsf,lxssc,ssize)

powsum = 0.0
DO g = 1,ng
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        vol = dx * dy * dz
        source = fsav(iii,ii,i,g)
        powsum = powsum + source * vol
      END DO
    END DO
  END DO
END DO
corpwr = powsum / powinit

CALL DATE_AND_TIME(VALUES = timarrend)
CALL diffTime(timarrbgn,timarrend,timdiff)

WRITE(*,100) outtim,corpwr,eigen,timdiff
WRITE(ofile,100) outtim,corpwr,eigen,timdiff

DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      fiss = 0.0
      DO h = 1,ng
        cross = lnxsf(h,nset)
        phi = sphi(iii,ii,i,h)
        fiss = fiss + cross * phi
      END DO
      DO l = 1,ndg
        btl = betal(l)
        lam = lambda(l)
        gamma = 1.0 / (1.0 + lam * ssize)
        pcold = pconc(l,iii,ii,i)
        pcnew = (gamma * pcold) + (gamma * btl * ssize * fiss)
        pconold(l,iii,ii,i) = pcold
        pconc(l,iii,ii,i) = pcnew
      END DO
    END DO
  END DO
END DO

IF (trflg.eq.4) THEN
Constant-Constant:
APHI2 holds the previous time step track-average angular flux

DO g = 1, ng
  invelt = 1.0 / (vel(g) * ssize)
  DO i = 1, nx
    DO ii = 1, ny
      DO iii = 1, nz
        dd = 0
        source = &
        & ssav(iii,ii,i,g) + fsav(iii,ii,i,g) + psav(iii,ii,i,g)
        nset = xsmap(iii,ii,i)
        cross = lxstr(g,nset)
        DO dq = 1, 8
          DO d = 1, dirnum
            dd = dd + 1
            nnxtmp = dirset(d)%nnx
            nnytmp = dirset(d)%nny
            naztmp = nnxtmp + nnytmp
            nnstmp = dirset(d)%nns
            DO n = 1, naztmp
              npltmp = npl(n,d)
              DO p = 1, npltmp
                track = zray(p,n,d)%trcln
                CALL findAphin(p,n,iii,ii,i,dd,g,dq,d, &
                & nnytmp,nnstmp,npltmp,npltmp,aphin)
                aphiout = aphi(p,n,iii,ii,i,dd,g)
                travg = (source / cross) + ((aphin - aphiout) &
                & / (cross * track))
                aphi2(p,n,iii,ii,i,dd,g) = travg
            END DO
          END DO
        END DO
      END DO
    END DO
  END DO
END DO

ELSE IF ((trflg.eq.5).or.(trflg.eq.6)) THEN

Constant-Linear and Constant-Exponential:
APHI2 holds the previous time step track-average angular flux

DO g = 1, ng
  dd = 0
  DO dq = 1, 8
    DO d = 1, dirnum
      dd = dd + 1
      DO i = 1, nx
        DO ii = 1, ny
          DO iii = 1, nz
            DO n = 1, nazmax
              DO p = 1, npomax
                aphi2(p,n,iii,ii,i,dd,g)=aphi(p,n,iii,ii,i,dd,g)
              END DO
            END DO
          END DO
        END DO
      END DO
    END DO
  END DO
END DO

ELSE IF ((trflg.eq.5).or.(trflg.eq.6)) THEN


END DO
!
ENDIF
!
increment to next time step
!
outtim = outtim + ssize
rmpflg = .true.
!
check for end of state and/or problem
!
stchgflg = 0
IF (outtim.gt.stend) THEN
state = state + 1
IF (state.gt.nstate) THEN
timchk = .false.
ELSE
rmpflg = .false.
ssize = delta(state)
stend = timend(state)
stchgflg = 1
! CALL writeEOS(state)
ENDIF
ENDIF
!
END DO
!
100 FORMAT(2x,es13.6,2x,es13.6,2x,es13.6,2x,es13.6,2x,es13.6)
!
estate = state
IF (estate.gt.nstate) THEN
estate = nstate
ENDIF
!
RETURN
END

SUBROUTINE outFDeig(lxstr,lnxsflxssc,ssize)
!
!!!!!!!!!!!!!!!!!!
USE files       !
USE scalars     !
USE tracking    !
USE xsects      !
USE neutrons    !
!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
LOGICAL convo            !
LOGICAL maxout           !
INTEGER nset             !
DOUBLE PRECISION cnvo    !
DOUBLE PRECISION fiss    !
403
DOUBLE PRECISION cross !
DOUBLE PRECISION phi !
DOUBLE PRECISION chi1 !
DOUBLE PRECISION chi2 !
DOUBLE PRECISION pcsum1 !
DOUBLE PRECISION pcsum2 !
DOUBLE PRECISION lam !
DOUBLE PRECISION btl !
DOUBLE PRECISION fsnew !
DOUBLE PRECISION eigold !
DOUBLE PRECISION fdnum !
DOUBLE PRECISION fdden !
DOUBLE PRECISION fdtmp !
DOUBLE PRECISION fd0tmp !
DOUBLE PRECISION fd1tmp !
DOUBLE PRECISION pctmp !
DOUBLE PRECISION ptmp !
DOUBLE PRECISION gamma !

INTEGER g,h      !
INTEGER i,ii,iii !
INTEGER l        !

DO WHILE (convo.and.maxout)
  cnvo  = 0.0
  iterout = iterout + 1
  IF (iterout.ge.iteromax) THEN
    maxout = .false.
  ENDIF
  !
  CALL inFDcc(lxstr,lxssc,ssize,ninner)
  ELSE IF (trflg.eq.5) THEN
    CALL inFDcl(lxstr,lxssc,ssize,ninner)
  ELSE IF (trflg.eq.6) THEN
  !     CALL inFDce(lxstr,lxssc,ssize)
 ENDIF

END !

!   Update Fixed Source Eigenvalue   !
!
!   Call Inner Iteration   !
!
IF (trflg.eq.4) THEN
  CALL inFDcc(lxstr,lxssc,ssize,ninner)
ELSE IF (trflg.eq.5) THEN
  CALL inFDcl(lxstr,lxssc,ssize,ninner)
ELSE IF (trflg.eq.6) THEN
  CALL inFDce(lxstr,lxssc,ssize)
ENDIF

END
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
  nset = xsmap(iii,ii,i)
  fd1(iii,ii,i) = fd0(iii,ii,i)
  fdtmp = 0.0
  DO g = 1,ng
    cross = lnxsf(g,nset)
    phi = sphi(iii,ii,i,g)
    fdtmp = fdtmp + phi * cross
  END DO
  fd0(iii,ii,i) = fdtmp
END DO
END DO
END DO
!
eigold = eigen
fdnum = 0.0
fddden = 0.0
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
  fd0tmp = fd0(iii,ii,i)
  fd1tmp = fd1(iii,ii,i)
  fdnum = fdnum + fd0tmp * dx * dy * dz
  fddden = fddden + fd1tmp * dx * dy * dz
END DO
END DO
END DO
eigen = eigold * (fdnum / fddden)
cnvo = abs((eigen - eigold) / eigen)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                  
!   Update Prompt Fission Source   
!                                  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DO g = 1,ng
  chi1 = chip(g)
DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
  nset = xsmap(iii,ii,i)
  fiss = 0.0
  DO h = 1,ng
    cross = lnxsf(h,nset)
    phi = sphi(iii,ii,i,h)
    fiss = fiss + cross * phi
  END DO
  fsnew = (chi1 / fpi) * (1 - beta) * fiss
  fsav(iii,ii,i,g) = fsnew
END DO
END DO
END DO
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                  
!   Update Delayed Fission Source   
!                                  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

405
DO g = 1,ng
   chi2 = chid(g)
   DO i = 1,nx
      DO ii = 1,ny
         DO iii = 1,nz
            nset = xsmap(iii,ii,i)
            fiss = 0.0
            DO h = 1,ng
               cross = lnxsf(h,nset)
               phi = sphi(iii,ii,i,h)
               fiss = fiss + cross * phi
            END DO
            pcsum1 = 0.0
            pcsum2 = 0.0
            DO l = 1,ndg
               lam = lambda(l)
               btl = betal(l)
               gamma = 1.0 / (1.0 + lam * ssize)
               pctmp = pconc(l,iii,ii,i)
               pcsum1 = pcsum1 + lam * gamma * pctmp
               pcsum2 = pcsum2 + lam * gamma * btl
            END DO
            ptmp = (chi2 / fpi) * (pcsum1 + pcsum2 * ssize * fiss)
            psav(iii,ii,i,g) = ptmp
         END DO
      END DO
   END DO
END DO
END DO
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                              !
!   Outer Iteration Clean-Up   !
!                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!   ...check convergence
!
IF (cnvo.lt.cco) THEN
   convo = .false.
END IF
!
!   ...short output
!
WRITE(ofile,100) iterout,cnvo,eigen,ninner
!
END DO
!
100 FORMAT(2x,'Outer #',i3,' Outer Cnv = ',es13.6,' Eigen = ',es13.6, &
         & '# Inners = ',i4)
!
RETURN
END

SUBROUTINE inFDcc(lxstr,lxssc,ssize,ninner)
!
!!!!!!!!!!!!!!!!!!!!
USE files
USE scalars
USE tracking
USE neutrons
USE xsects

SUBROUTINE inFDcc(lxstr,lxssc,ssize,ninner)

SUBROUTINE inFDcc(lxstr,lxssc,ssize,ninner)
DOUBLE PRECISION lxstr(ng,nxsets)!
DOUBLE PRECISION lxssc(ng,ng,nxsets)!
DOUBLE PRECISION ssize!
INTEGER ninner!
LOGICAL conva!
LOGICAL convs!
LOGICAL maxin!
INTEGER nnxtmp!
INTEGER nnytmp!
INTEGER nnstmp!
INTEGER nnztmp!
INTEGER naztmp!
INTEGER npltmp!
INTEGER obflg!
INTEGER nset!
DOUBLE PRECISION cnvaf!
DOUBLE PRECISION cnvsf!
DOUBLE PRECISION invelt!
DOUBLE PRECISION albin!
DOUBLE PRECISION aphin!
DOUBLE PRECISION source!
DOUBLE PRECISION cross!
DOUBLE PRECISION mfp!
DOUBLE PRECISION track!
DOUBLE PRECISION crosseff!
DOUBLE PRECISION aphiavgold!
DOUBLE PRECISION aphiout!
DOUBLE PRECISION sphold!
DOUBLE PRECISION sphisum!
DOUBLE PRECISION numsum!
DOUBLE PRECISION densum!
DOUBLE PRECISION weight!
DOUBLE PRECISION trsep!
DOUBLE PRECISION travg!
DOUBLE PRECISION aphiavg!
DOUBLE PRECISION sphnew!
DOUBLE PRECISION cnvtmp!
DOUBLE PRECISION scat!
DOUBLE PRECISION phi!
INTEGER g, h!
INTEGER d, dq, dd, dfr!
INTEGER i, ix!
INTEGER ii, iix!
INTEGER iii, iiix!
INTEGER n, nn, nfr, nnt!
INTEGER p, pp, pfr, ppt!

407
BEGIN Inner Iteration
ninner = 0
DO g = 1, ng
  invelt = 1.0 / (vel(g) * ssize)
  conva = .true.
  convs = .true.
  maxin = .true.
  iterin = 0
  DO WHILE ((conva.or.convs).and.maxin)
    cnva = 0.0
    cnvsf = 0.0
    iterin = iterin + 1
    IF (iterin.ge.iterimax) THEN
      maxin = .false.
    ENDIF
  ENDWHILE
  DO dq = 1, 8
    DO d = 1, dirnum
      dd = dd + 1
      nnxtmp = dirset(d)%nnx
      nnytmp = dirset(d)%nny
      nnstmp = dirset(d)%nns
      nnztmp = dirset(d)%nnz
      naztmp = nnxtmp + nnytmp
      ! Tracks that originate along +/-x face
      DO ii = 1, ny
        DO iii = 1, nz
          npltmp = npl(n,d)
          DO p = npltmp - nnstmp + 1, npltmp
            obflg = 0
            nn = n
            pp = p
            iix = ii
            iiix = iii
            SELECT CASE (dq)
              CASE (1)
                ix = 1
                dfr = dirnum + d
                albin = albl
              CASE (2)
                ix = nx
                dfr = d
                albin = alb2
              CASE (3)
                ix = nx
                dfr = 3*dirnum + d
                albin = alb2
            ENDSELECT
          ENDDO
        ENDDO
      ENDDO
    ENDDO
  ENDDO
ENDDO
ENDDO
ENDDO
CASE (4)
    ix = 1
    dfr = 2*dirnum + d
    albin = alb1
CASE (5)
    ix = 1
    dfr = 5*dirnum + d
    albin = alb1
CASE (6)
    ix = nx
    dfr = 4*dirnum + d
    albin = alb2
CASE (7)
    ix = nx
    dfr = 7*dirnum + d
    albin = alb2
CASE (8)
    ix = 1
    dfr = 6*dirnum + d
    albin = alb1
END SELECT

nfr = zray(pp,nn,d)%plnfr
pfr = zray(pp,nn,d)%trcfr
aphin = albin * aphi(pfr,nfr,iiix,iix,ix,dfr,g)
DO
    source = ssav(iiix,iix,ix,g) + &
    & fsav(iiix,iix,ix,g) + psav(iiix,iix,ix,g)
nset = xsmap(iiix,iix,ix)
cross = lxstr(g,nset)
crosseff = invelt + cross
mfp = ncp(pp,nn,nset,d,g)
track = zray(pp,nn,d)%trcln
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
aphiavgold = aphi2(pp,nn,iiix,iix,ix,dd,g)
aphiout = aphin * mfp + &
&       (source + (invelt * aphiavgold)) &
&         * (1.0 / crosseff) * (1.0 - mfp)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
IF (aphiout.lt.aphimin) THEN
    aphiout = aphimin
ELSE
    aphilast = aphi(pp,nn,iiix,iix,ix,dd,d)
    cnvtmp = abs((aphiout - aphilast) / aphiout)
    IF (cnvtmp.gt.cnvaf) THEN
        cnvaf = cnvtmp
    ENDIF
ENDIF
aphi(pp,nn,iiix,iix,ix,dd,g) = aphiout
npltmp = npl(nn,d)
CALL nextTrack(dq,nn,pp,ix,iix,iiix, &
    & nnxtmp,nnttmp,obflg)
IF (obflg.eq.1) THEN
    EXIT
ELSE
    aphin = aphiout
    nnt = nn
    ppt = pp
    nn = zray(ppt,nnt,d)%plnto
    pp = zray(ppt,nnt,d)%trcto
ENDIF
END DO
END DO
END DO
Tracks that originate along +/-y face

DO iii = 1,nz
DO i = 1,nx
DO n = 1,nnytmp
npltmp = npl(n,d)
DO p = npltmp-nnstmp+1,npltmp
obflg = 0
nn = n
pp = p
ix = i
iixx = iiii
SELECT CASE (dq)
CASE (1)
iix = 1
dfr = 3*dirnum + d
albin = alb3
CASE (2)
iix = 1
dfr = 2*dirnum + d
albin = alb3
CASE (3)
iix = ny
dfr = dirnum + d
albin = alb4
CASE (4)
iix = ny
dfr = d
albin = alb4
CASE (5)
iix = 1
dfr = 7*dirnum + d
albin = alb3
CASE (6)
iix = 1
dfr = 6*dirnum + d
albin = alb3
CASE (7)
iix = ny
dfr = 5*dirnum + d
albin = alb4
CASE (8)
iix = ny
dfr = 4*dirnum + d
albin = alb4
END SELECT
nfr = zray(pp,nn,d)%plnfr
pfr = zray(pp,nn,d)%trcfr
aphin = albin * aphi(pfr,nfr,iixx,iix,ix,dfr,g)
DO
source = ssav(iixx,iix,ix,g) + 
&   fsav(iixx,iix,ix,g) + psav(iixx,iix,ix,g)
nset = xsmap(iixx,iix,ix)
cross = lxstr(g,nset)
crosseff = invelt + cross
mfp = ncp(pp,nn,nset,d,g)
track = zray(pp,nn,d)%trcln
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
aphiavgold = aphi2(pp,nn,iixx,iix,ix,dd,g)
aphiout = aphin * mfp + (source + (invelt &
& * aphιavgold)) &
& * (1.0 / crosseff) * (1.0 - mfp)

!IF (aphiout.lt.aphimin) THEN
  aphiout = aphimin
ELSE
  aphi_last = aphi(pp,nn,iiix,iix,ix,dd,g)
  cnvtmp = abs((aphiout - aphi_last) / aphiout)
  IF (cnvtmp.gt.cnvaf) THEN
    cnvaf = cnvtmp
ENDIF
ENDIF
aphi(pp,nn,iiix,iix,ix,dd,g) = aphiout
npltmp = npl(nn,d)
CALL nextTrack(dq,nn,pp,ix,iix,iiix, &
  nnxtmp,nnstmp,obflg)
IF (obflg.eq.1) THEN
  EXIT
ELSE
  aphin = aphiout
  nnt = nn
  ppt = pp
  nn = zray(ppt,nnt,d)%plnto
  pp = zray(ppt,nnt,d)%trcto
ENDIF
ENDIF
END DO
END DO
END DO
END DO
END DO
END DO
END DO

! Tracks that originate along +/-z face
!
DO i = 1,nx
  DO ii = 1,ny
    DO n = 1,naztmp
      npltmp = npl(n,d)
      DO p = 1,npltmp-nnstmp
        obflg = 0
        nn = n
        pp = p
        ix = i
        iix = ii
        SELECT CASE (dq)
        CASE (1)
          iixx = 1
          dfr = 4*dirnum + d
          albin = alb5
        CASE (2)
          iixx = 1
          dfr = 5*dirnum + d
          albin = alb5
        CASE (3)
          iixx = 1
          dfr = 6*dirnum + d
          albin = alb5
        CASE (4)
          iixx = 1
          dfr = 7*dirnum + d
          albin = alb5
        CASE (5)
          iixx = nz
          dfr = d
        END SELECT
albin = alb6
CASE (6)
   iiix = nz
   dfr = dirnum + d
   albin = alb6
CASE (7)
   iiix = nz
   dfr = 2*dirnum + d
   albin = alb6
CASE (8)
   iiix = nz
   dfr = 3*dirnum + d
   albin = alb6
END SELECT
nfr = zray(pp,nn,d)%plnfr
pfr = zray(pp,nn,d)%trcfr
aphin = albin * aphi(pfr,nfr,iiix,iix,ix,dfr,g)
DO
   source = ssav(iiix,iix,ix,g) + 
   &        fsav(iiix,iix,ix,g) + psav(iiix,iix,ix,g)
   nset = xsmap(iiix,iix,ix)
   cross = lxstr(g,nset)
   crosseff = invelt + cross
   mfp = ncp(pp,nn,nset,d,g)
   track = zray(pp,nn,d)%trcln
   aphiavgold = aphi2(pp,nn,iiix,iix,ix,dd,g)
   aphiout = aphin * mfp + (source + (invelt 
   &       * aphiavgold)) * (1.0 / crosseff) * (1.0 - mfp)
   IF (aphiout.lt.aphimin) THEN
      aphiout = aphimin
   ELSE
      aphilast = aphi(pp,nn,iiix,iix,ix,dd,g)
      cnvtmp = abs((aphiout - aphilast) / aphiout)
      IF (cnvtmp.gt.cnvaf) THEN
         cnvaf = cnvtmp
      ENDIF
   ENDIF
aphi(pp,nn,iiix,iix,ix,dd,g) = aphiout
npltmp = npl(nn,d)
CALL nextTrack(dq,nn,pp,ix,iix,iiix, 
   &        nnxtmp,nnstmp,obflg)
IF (obflg.eq.1) THEN
   EXIT
ELSE
   aphin = aphiout
   nnt = nn
   ppt = pp
   nn = zray(ppt,nnt,d)%plnto
   pp = zray(ppt,nnt,d)%trcto
ENDIF
END DO
END DO
END DO
END DO
Calculate Scalar Flux

DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      sphold = sphi(iii,ii,i,g)
      dd = 0
      sphisum = 0.0
      source = ssav(iii,ii,i,g) + &
        fsav(iii,ii,i,g) + psav(iii,ii,i,g)
      nset = xsmap(iii,ii,i)
      cross = lxstr(g,nset)
      DO dq = 1,8
        DO d = 1,dirnum
          dd = dd + 1
          numsum = 0.0
          densum = 0.0
          weight = dirset(d)%wght
          trsep = dirset(d)%dazi * dirset(d)%dpol
          nnxtmp = dirset(d)%nnx
          nnytmp = dirset(d)%nny
          naztmp = nnxtmp + nnytmp
          nnstmp = dirset(d)%nns
          DO n = 1,naztmp
            npltmp = npl(n,d)
            DO p = 1,npltmp
              track = zray(p,n,d)%trcln
              CALL findAphin(p,n,iii,ii,i,dd,g,dq,d, &
                nnxtmp,nnstmp,npltmp,aphin)
              aphiout = aphi(p,n,iii,ii,i,dd,g)
              travg = (source / cross) + ((aphin - aphiout) &
                / (cross * track))
              numsum = numsum + travg * track * trsep
              densum = densum + track * trsep
            END DO
          END DO
          aphiavg = numsum / densum
          sphisum = sphisum + aphiavg * weight
        END DO
      END DO
      sphnew = sphisum * fpi
      sphi(iii,ii,i,g) = sphnew
      cnvtmp = abs((sphnew - sphold) / sphnew)
      IF (cnvtmp.gt.cnvsf) THEN
        cnvsf = cnvtmp
      ENDIF
    END DO
  END DO
END DO

Calculate Scattering Source
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      scat = 0.0
      nset = xsmap(iii,ii,i)
      DO h = 1,ng
        phi = sphi(iii,ii,i,h)
        scat = scat + lxssc(g,h,nset) * phi
      END DO
      ssav(iii,ii,i,g) = scat / fpi
    END DO
  END DO
END DO

!  check angular flux convergence...
IF (cnvaf.lt.cca) THEN
  conva = .false.
END IF
!
!  check scalar flux convergence...
IF (cnvsf.lt.ccs) THEN
  convs = .false.
END IF
!
END DO
!
!  Update NINNER
!
IF (iterin.gt.ninner) THEN
  ninner = iterin
ENDIF
!
END DO
!
100 FORMAT(2x,'# Inners ',i6,' Inner S Cnv = ',es13.6,' &
           Inner A Cnv = ',es13.6)
!
RETURN
END

SUBROUTINE inFDcl(lxstr,lxssc,ssize,ninner)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
USE files
USE scalars
USE tracking
USE neutrons
USE xsects
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION lxstr(ng,nxsets) !
DOUBLE PRECISION lxssc(ng,ng,nxsets) !
DOUBLE PRECISION ssize               !
INTEGER ninner
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!
LOGICAL conva !
LOGICAL convs !
LOGICAL maxin !
!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!
INTEGER nnxtmp !
INTEGER nnytmp !
INTEGER nnytmp !
INTEGER nnztmp !
INTEGER nztmp !
INTEGER nztmp !
INTEGER nztmp !
INTEGER nztmp !
INTEGER nztmp !
INTEGER nztmp !
INTEGER obflg !
INTEGER nset !
!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION cnvaf      !
DOUBLE PRECISION cnvsf      !
DOUBLE PRECISION invelt     !
DOUBLE PRECISION albin      !
DOUBLE PRECISION aphin      !
DOUBLE PRECISION source     !
DOUBLE PRECISION cross      !
DOUBLE PRECISION mfp        !
DOUBLE PRECISION track      !
DOUBLE PRECISION crosseff   !
DOUBLE PRECISION aphilast   !
DOUBLE PRECISION aphinoold  !
DOUBLE PRECISION aphioutold !
DOUBLE PRECISION aphiout    !
DOUBLE PRECISION sphold     !
DOUBLE PRECISION sphisum    !
DOUBLE PRECISION numsum     !
DOUBLE PRECISION densum     !
DOUBLE PRECISION weight     !
DOUBLE PRECISION trsep      !
DOUBLE PRECISION travg      !
DOUBLE PRECISION aphivavg   !
DOUBLE PRECISION cnvtmp     !
DOUBLE PRECISION scat       !
DOUBLE PRECISION phi       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
BEGIN Inner Iteration!
!
! ninner = 0
DO g = 1,ng
   invelt = 1.0 / (vel(g) * ssize)
conva = .true.
convs = .true.
maxin = .true.
iterin = 0
DO WHILE ((conva.or.convs).and.maxin)
  cnvaf = 0.0
  cnvsf = 0.0
  iterin = iterin + 1
  IF (iterin.ge.iterimax) THEN
    maxin = .false.
  ENDIF
ENDIF

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                         !
!   Calculate Angular Flux Distribution   !
!                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
  dd = 0
DO dq = 1,8
  DO d = 1,direnum
    dd = dd + 1
    nnxtmp = dirset(d)%nnx
    nnytmp = dirset(d)%nny
    nnstmp = dirset(d)%nns
    nnztmp = dirset(d)%nnz
    naztmp = nnxtmp + nnytmp
    !
    ! Tracks that originate along +/-x face
    !
    DO ii = 1,ny
      DO iii = 1,nz
        DO n = nnytmp+1,naztmp
          npltmp = npl(n,d)
          DO p = npltmp-nnstmp+1,npltmp
            obflg = 0
            nn = n
            pp = p
            iix = ii
            iiix = iii
            SELECT CASE (dq)
            CASE (1)
              ix = 1
              df = dirnum + d
              albin = alb1
            CASE (2)
              ix = nx
              df = d
              albin = alb2
            CASE (3)
              ix = nx
              df = 3*dirnum + d
              albin = alb2
            CASE (4)
              ix = 1
              df = 2*dirnum + d
              albin = alb1
            CASE (5)
              ix = 1
              df = 5*dirnum + d
              albin = alb1
            CASE (6)
              ix = nx
              albin = alb1
            ENDSELECT
            !
          ENDDO
        ENDDO
      ENDDO
    ENDDO
  ENDDO
ENDDO

416
df1 = 4*dirnum + d  
albin = alb2  
CASE (7)  
  ix = nx  
  df1 = 7*dirnum + d  
albin = alb2  
CASE (8)  
  ix = 1  
  df1 = 6*dirnum + d  
albin = alb1  
END SELECT  
nfr = zray(pp,nn,d)%plnfr  
pfr = zray(pp,nn,d)%trcfr  
aphin = albin * aphi(pfr,nfr,iiix,iix,ix,df1,g)  
DO  
  source = ssav(iiix,iix,iix,ix,g) + &  
     fsav(iiix,iix,iix,iix,ix,g) + psav(iiix,iix,iix,ix,g)  
nset = xsmap(iiix,iix,iix,ix)  
cross = lxstr(g,nset)  
crosseff = invelt + cross  
mfp = ncp(pp,nn,nset,d,g)  
track = zray(pp,nn,d)%trcln  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
CALL findAphin(pp,nn,iiix,iix,ix,dd,g, &  
    nnytmp,nntmp,npltmp,aphinold)  
aphioutold = aphi2(pp,nn,iiix,iix,ix,dd,g)  
aphiout = aphin * mfp + (1.0 - mfp) * (source &  
     / crosseff) &  
     + (invelt / crosseff) * (1.0 - (1.0 - mfp) &  
     / (crosseff * track)) * aphinold &  
     - (invelt / crosseff) * (mfp - (1.0 - mfp) &  
     / (crosseff * track)) * aphioutold  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
IF (aphiout.lt.aphimin) THEN  
  aphiout = aphin  
ELSE  
  aphilast = aphi(pp,nn,iiix,iix,ix,dd,g)  
cnvtmp = abs((aphiout - aphilast) / aphiout)  
IF (cnvtmp.gt.cnvaf) THEN  
  cnvaf = cnvtmp  
ENDIF  
ENDIF  
aphi(pp,nn,iiix,iix,ix,dd,g) = aphiout  
npltmp = npl(nn,d)  
CALL nextTrack(dg,nn,pp,ix,iix,iix, &  
    nnxtmp,nntmp,obflg)  
IF (obflg.eq.1) THEN  
  EXIT  
ELSE  
  aphin = aphiout  
nnt = nn  
ppt = pp  
n = zray(ppt,nnt,d)%plnto  
pp = zray(ppt,nnt,d)%trcto  
ENDIF  
END DO  
END DO  
END DO  
END DO  
!
!
Tracks that originate along +/-y face
DO iii = 1, nz
DO i = 1, nx
DO n = 1, nnytmp
   npltmp = npl(n, d)
DO p = npltmp - nstmp + 1, npltmp
   obflg = 0
   nn = n
   pp = p
   ix = i
   iiix = iii
SELECT CASE (dq)
CASE (1)
   iiix = 1
   dfr = 3 * dirnum + d
   albin = alb3
CASE (2)
   iiix = 1
   dfr = 2 * dirnum + d
   albin = alb3
CASE (3)
   iiix = ny
   dfr = dirnum + d
   albin = alb4
CASE (4)
   iiix = ny
   dfr = d
   albin = alb4
CASE (5)
   iiix = 1
   dfr = 7 * dirnum + d
   albin = alb3
CASE (6)
   iiix = 1
   dfr = 6 * dirnum + d
   albin = alb3
CASE (7)
   iiix = ny
   dfr = 5 * dirnum + d
   albin = alb4
CASE (8)
   iiix = ny
   dfr = 4 * dirnum + d
   albin = alb4
END SELECT
nfr = zray(pp, nn, d) % plnfr
pfr = zray(pp, nn, d) % trcfr
aphin = albin * aphi(pfr, nfr, iiix, iix, ix, dfr, g)
DO
   source = ssav(iiix, iix, ix, g) + &
   & fsav(iiix, iix, ix, g) + psav(iiix, iix, ix, g)
   nset = xsmap(iiix, iix, ix)
   cross = lxstr(g, nset)
   crosseff = invelt + cross
   mfp = ncp(pp, nn, nset, d, g)
   track = zray(pp, nn, d) % trcln
                        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   CALL findAphin(pp, nn, iix, iiix, iix, ix, d, &
   & nnytmp, nstmp, npltmp, aphinold)
aphinoutold = aphi2(pp, nn, iiix, iix, ix, df, g)
aphinout = aphin * mfp + (1.0 - mfp) * (source &
   & / crosseff) &
   & + (invelt / crosseff) * (1.0 - (1.0 - mfp) &
   & / (crosseff * track)) * aphinold &
& - (invelt / crosseff) * (mfp - (1.0 - mfp) &
& / (crosseff * track)) * aphioutold

! IF (aphiout.lt.aphimin) THEN
! aphiout = aphioutold
ELSE
aphilast = aphi(pp,nn,iiix,iix,ix,dd,g)
cnvtmpl = abs((aphiout - aphilast) / aphiout)
IF (cnvtmpl.gt.cnvaf) THEN
  cnvaf = cnvtmpl
ENDIF
ENDIF
aphi(pp,nn,iiix,iix,ix,dd,g) = aphiout
nptmp = npl(nn,d)
CALL nextTrack(dq,nn,pp,ix,iix,iiix, &
& nxxtmp,nnstmp,obflg)
IF (obflg.eq.1) THEN
  EXIT
ELSE
  aphin = aphiout
  nnt = nn
  ppt = pp
  nn = zray(ppt,nnt,d)%plnto
  pp = zray(ppt,nnt,d)%trcto
ENDIF
ENDIF
END DO
END DO
END DO
END DO
END DO
!
Tracks that originate along +/-z face
!
DO i = 1,nx
DO ii = 1,ny
DO n = 1,Nztmp
npltmp = npl(n,d)
DO p = 1,npltmp-nnstmp
obflg = 0
nn = n
pp = p
ix = i
iix = ii
SELECT CASE (dq)
CASE (1)
iix = 1
dfr = 4*dirnum + d
albin = alb5
CASE (2)
iix = 1
dfr = 5*dirnum + d
albin = alb5
CASE (3)
iix = 1
dfr = 6*dirnum + d
albin = alb5
CASE (4)
iix = 1
dfr = 7*dirnum + d
albin = alb5
CASE (5)
iix = nz
dfr = d
albin = alb6
CASE (6)
   iiix = nz
   dfr = dirnum + d
   albin = alb6
CASE (7)
   iiix = nz
   dfr = 2*dirnum + d
   albin = alb6
CASE (8)
   iiix = nz
   dfr = 3*dirnum + d
   albin = alb6
END SELECT
nfr = zray(pp,nn,d)%plnfr
pfr = zray(pp,nn,d)%trcfr
aphin = albin * aphi(pfr,nfr,iiix,iix,ix,dfr,g)
DO
   source = ssav(iiix,iix,ix,g) + &
      &   fsav(iiix,iix,ix,g) + psav(iiix,iix,ix,g)
   nset = xsmap(iiix,iix,ix)
   cross = lxstr(g,nset)
   crosseff = invelt + cross
   mfp = ncp(pp,nn,nset,d,g)
   track = zray(pp,nn,d)%trcln
   CALL findAphin(pp,nn,iiix,iix,ix,dd,g,dq,d, &
      &   nnytmp,nnstmp,npltmp,aphinold)
aphioutold = aphi2(pp,nn,iiix,iix,ix,dd,d,g)
aphiout = aphin * mfp + (1.0 - mfp) * (source &
      &   / crosseff) &
      &   + (invelt / crosseff) * (1.0 - (1.0 - mfp) &
      &   / (crosseff * track)) * aphinold &
      &   - (invelt / crosseff) * (mfp - (1.0 - mfp) &
      &   / (crosseff * track)) * aphioutold
   CALL nextTrack(dq,nn,pp,ix,iix,iiix, &
      &   nnxtmp,nnstmp,obflg)
   IF (obflg.eq.1) THEN
      EXIT
   ELSE
      aphin = aphiout
      nnt = nn
      ppt = pp
      nn = zray(ppt,nnt,d)%plnto
      pp = zray(ppt,nnt,d)%trcto
   ENDIF
END DO
!
!!!!!!!!!!!!   !!!!
END DO !d      !!!!
END DO !dq     !!!!
!!!!!!!!!!!!   !!!!
!!!end!calculate!aphi!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Calculate Scalar Flux 
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DO i = 1,nx
DO ii = 1,ny
DO iii = 1,nz
   sphold = sphi(iii,ii,i,g)
   dd = 0
   sphisum = 0.0
   source = ssav(iii,ii,i,g) + &
   & fsav(iii,ii,i,g) + psav(iii,ii,i,g)
   nset = xsmap(iii,ii,i)
   cross = lxstr(g,nset)
   DO dq = 1,8
      DO d = 1,dirnum
         dd = dd + 1
         numsum = 0.0
         densum = 0.0
         weight = dirset(d)%wght
         trsep = dirset(d)%dazi * dirset(d)%dpol
         nnxtmp = dirset(d)%nnx
         nnytmp = dirset(d)%nny
         naztmp = nnxtmp + nnytmp
         nnstmp = dirset(d)%nns
         DO n = 1,naztmp
            npltmp = npl(n,d)
            DO p = 1,npltmp
               track = zray(p,n,d)%trcln
               CALL findAphin(p,n,iii,ii,i,dd,g,dq,d, &
               & nnytmp,nnstmp,npltmp,aphin)
               aphiout = aphi(p,n,iii,ii,i,dd,d)
               travg = (source / cross) + ((aphin - aphiout) &
               & / (cross * track))
               numsum = numsum + travg * track * trsep
               densum = densum + track * trsep
            END DO
         END DO
         aphiavg = numsum / densum
         sphisum = sphisum + aphiavg * weight
      END DO
   END DO
   sphnew = sphisum * fpi
   sphi(iii,ii,i,g) = sphnew
   cnvtmp = abs((sphnew - sphiold) / sphnew)
   IF (cnvtmp.gt.cnvsf) THEN
      cnvsf = cnvtmp
   ENDIF
   END DO
END DO
END DO
END DO

!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Check Scattering Source !
!!
!! DO i = 1,nx
!!   DO ii = 1,ny
!!     DO iii = 1,nz
!!        scat = 0.0
!!           nset = xsmap(iii,ii,i)
!!           DO h = 1,ng
!!             phi = sphi(iii,ii,i,h)
!!             scat = scat + lxssc(g,h,nset) * phi
!!           END DO
!!           ssav(iii,ii,i,g) = scat / fpi
!!         END DO
!!       END DO
!!     END DO
!!   END DO
!!
!!   Check angular flux convergence...
!!
!!     IF (cnvaf.lt.cca) THEN
!!       conva = .false.
!!     END IF
!!
!!   Check scalar flux convergence...
!!
!!     IF (cnvsf.lt.ccs) THEN
!!       convs = .false.
!!     END IF
!!
!!   END DO
!!
!!   Update NINNER
!!
!!     IF (iterin.gt.ninner) THEN
!!       ninner = iterin
!!     ENDIF
!!
!!   END DO
!!
!! 100 FORMAT(2x,'# Inners ',i6,' Inner S Cnv = ',es13.6,' &
!! & Inner A Cnv = ',es13.6)
!!
!! RETURN
!!
!! ENDSUB

SUBROUTINE diffTime(array1,array2,CPUsec)
!!
!! INTEGER, DIMENSION(8) :: array1
!! INTEGER, DIMENSION(8) :: array2
!! DOUBLE PRECISION CPUsec
!! INTEGER milli1,milli2
!! INTEGER sec1,sec2
!! INTEGER min1,min2
!! INTEGER hour1,hour2
!! INTEGER day1,day2
!! INTEGER sectot1, sectot2
!! DOUBLE PRECISION msectot1, msectot2
! milli1 = array1(8)
milli2 = array2(8)
sec1 = array1(7)
sec2 = array2(7)
min1 = array1(6)
min2 = array2(6)
hour1 = array1(5)
hour2 = array2(5)
day1 = array1(3)
day2 = array2(3)
!
IF ((day2-day1).ne.0) THEN
  write(*,*) 'dayfix'
  hour2 = hour2 + 24
ENDIF
!
sectot1 = hour1 * 3600 + min1 * 60 + sec1
sectot2 = hour2 * 3600 + min2 * 60 + sec2
!
write(*,*) sectot1, sectot2
!
CPUsec = DBLE(sectot2 - sectot1)
!
RETURN
END

SUBROUTINE findAphin(p,n,iii,ii,i,d,g,&
  octant,dir,nnytmp,nnstmp,npltmp,aphin)
!
!!!!!!!!!!!!
USE files
USE scalars
USE neutrons
USE tracking
!!!!!!!!!!!!
!
INTEGER g
INTEGER d
INTEGER p
INTEGER n
INTEGER i, ii, iii
INTEGER octant
INTEGER dir
INTEGER nnytmp, nnstmp, npltmp
DOUBLE PRECISION aphitmp
DOUBLE PRECISION aphin
DOUBLE PRECISION albedo
INTEGER dfr
INTEGER pfr
INTEGER nfr
INTEGER ifr, iifr, iiifr
!!!!!!!!!!!!
!
nfr = zray(p, n, dir)%plnfr
pfr = zray(p, n, dir)%trcfr
!
SELECT CASE (octant)
!
CASE (1)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.1) THEN
    dfr = 4*dirnum + dir
    albedo = alb5
    aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii - 1
    aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.1) THEN
      dfr = dirnum + dir
      albedo = alb1
      aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphitmp = aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 3*dirnum + dir
      albedo = alb3
      aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphitmp = aphi(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
!
CASE (2)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.1) THEN
    dfr = 5*dirnum + dir
    albedo = alb5
    aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii - 1
    aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = dir
      albedo = alb2
      aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphitmp = aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 2*dirnum + dir
      albedo = alb3
      aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
  ENDIF
ENDIF
aphitmp = aphi(pfr,nfr,iii,iiifr,ii,i,d,g)
ENDIF
ENDIF
ENDIF
!
CASE (3)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.1) THEN
    dfr = 6*dirnum + dir
    albedo = alb5
    aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii - 1
    aphi(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF (n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = 3*dirnum + dir
      albedo = alb2
      aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.ny) THEN
      dfr = dirnum + dir
      albedo = alb4
      aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii + 1
      aphi(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
ELSE
  IF (ii.eq.ny) THEN
    dfr = dirnum + dir
    albedo = alb4
    aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iifr = ii - 1
    aphi(pfr,nfr,iii,iifr,ii,i,d,g)
  ENDIF
ENDIF
ENDIF
!
CASE (4)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.1) THEN
    dfr = 7*dirnum + dir
    albedo = alb5
    aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii - 1
    aphi(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF (n.gt.nnytmp) THEN
    IF (i.eq.1) THEN
      dfr = 2*dirnum + dir
      albedo = alb1
      aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.ny) THEN
      dfr = dir
      albedo = alb4
      aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii + 1
      aphi(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ELSE
  IF (ii.eq.ny) THEN
    dfr = dir
    albedo = alb4
aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
ELSE
  iiifr = ii + 1
  aphi = aphi(pfr,nfr,iii,ii,i,iiifr,d,g)
ENDIF
ENDIF
ENDIF
CASE (5)
!
CASE (5)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.nz) THEN
    dfr = dir
    albedo = alb6
    aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii + 1
    aphi = aphi(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.1) THEN
      dfr = 5*dirnum + dir
      albedo = alb1
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphi = aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 7*dirnum + dir
      albedo = alb3
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi = aphi(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = 4*dirnum + dir
      albedo = alb2
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphi = aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 7*dirnum + dir
      albedo = alb3
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi = aphi(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
CASE (6)
!
CASE (6)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.nz) THEN
    dfr = dirnum + dir
    albedo = alb6
    aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii + 1
    aphi = aphi(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = 4*dirnum + dir
      albedo = alb2
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphi = aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 7*dirnum + dir
      albedo = alb3
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi = aphi(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = 4*dirnum + dir
      albedo = alb2
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphi = aphi(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 7*dirnum + dir
      albedo = alb3
      aphi = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi = aphi(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
ELSE
IF (ii.eq.1) THEN  
  dfr = 6*dirnum + dir  
  albedo = alb3  
  aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
ELSE  
  iiifr = ii - 1  
  aphitmp = aphi(pfr,nfr,iii,iiifr,i,d,g)  
ENDIF  
ENDIF  
ENDIF  
!
CASE (7)  
!
IF (p.le.(nptmp-nntmp)) THEN  
  IF (iii.eq.nz) THEN  
    dfr = 2*dirnum + dir  
    albedo = alb6  
    aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
  ELSE  
    iiifr = iii + 1  
    aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)  
  ENDIF  
  ELSE  
    IF(n.gt.nntmp) THEN  
      IF (i.eq.nx) THEN  
        dfr = 7*dirnum + dir  
        albedo = alb2  
        aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
      ELSE  
        ifr = i + 1  
        aphitmp = aphi(pfr,nfr,iii,ii,ifr,d,g)  
      ENDIF  
    ELSE  
      IF (ii.eq.ny) THEN  
        dfr = 5*dirnum + dir  
        albedo = alb4  
        aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
      ELSE  
        iiifr = ii + 1  
        aphitmp = aphi(pfr,nfr,iii,iiifr,i,d,g)  
      ENDIF  
    ENDIF  
  ELSE  
    IF (p.le.(nptmp-nntmp)) THEN  
      IF (iii.eq.nz) THEN  
        dfr = 3*dirnum + dir  
        albedo = alb6  
        aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
      ELSE  
        iiifr = iii + 1  
        aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)  
      ENDIF  
    ELSE  
      IF (ii.eq.ny) THEN  
        dfr = 5*dirnum + dir  
        albedo = alb4  
        aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
      ELSE  
        iiifr = ii + 1  
        aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)  
      ENDIF  
    ELSE  
      IF (n.gt.nntmp) THEN  
        IF (i.eq.1) THEN  
          dfr = 6*dirnum + dir  
          albedo = alb1  
          aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
        ELSE  
          ifr = i - 1  
          aphitmp = aphi(pfr,nfr,iii,ii,ifr,d,g)  
        ENDIF  
      ENDIF  
    ELSE  
      IF (ii.eq.ny) THEN  
        dfr = 5*dirnum + dir  
        albedo = alb4  
        aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
      ELSE  
        iiifr = ii + 1  
        aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)  
      ENDIF  
    ENDIF  
  ENDIF  
ENDIF  
!
CASE (8)  
!
IF (p.le.(nptmp-nntmp)) THEN  
  IF (iii.eq.nz) THEN  
    dfr = 3*dirnum + dir  
    albedo = alb6  
    aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
  ELSE  
    iiifr = iii + 1  
    aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)  
  ENDIF  
  ELSE  
    IF (ii.eq.ny) THEN  
      dfr = 5*dirnum + dir  
      albedo = alb4  
      aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
    ELSE  
      iiifr = ii + 1  
      aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)  
    ENDIF  
  ELSE  
    IF (n.gt.nntmp) THEN  
      IF (i.eq.1) THEN  
        dfr = 6*dirnum + dir  
        albedo = alb1  
        aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
      ELSE  
        ifr = i - 1  
        aphitmp = aphi(pfr,nfr,iii,ii,ifr,d,g)  
      ENDIF  
    ELSE  
      IF (ii.eq.ny) THEN  
        dfr = 5*dirnum + dir  
        albedo = alb4  
        aphitmp = aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo  
      ELSE  
        iiifr = ii + 1  
        aphitmp = aphi(pfr,nfr,iiifr,ii,i,d,g)  
      ENDIF  
    ENDIF  
  ENDIF  
ENDIF  
!

427
aphitmp = aphi(pfr,nfr,iii,ii,ifr,d,g)
ENDIF
ELSE
  IF (ii.eq.ny) THEN
    dfr = 4*dirnum + dir
    albedo = alb4
    aphi(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = ii + 1
    aphi(pfr,nfr,iii,iiifr,i,d,g)
  ENDIF
ENDIF
ENDIF
ENDIF
!
END SELECT
!
aphin = aphi
!
RETURN
END
SUBROUTINE findAphin2(p,n,iii,ii,i,d,g, &
&          octant,dir,nnytmp,nnstmp,npltmp,aphin)
!
!!!!!!!!!!!!!!!!
USE files
USE scalars !
USE neutrons !
USE tracking !
!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!
INTEGER g
INTEGER d
INTEGER p
INTEGER n
INTEGER i,ii,iii
INTEGER octant
INTEGER dir
INTEGER nnytmp,nnstmp,npltmp
DOUBLE PRECISION aphi
DOUBLE PRECISION albedo
DOUBLE PRECISION dfr
INTEGER pfr
INTEGER nfr
INTEGER ifr,iifr,iiifr
!!!!!!!!!!!!!!
!
    nfr = zray(p,n,dir)%plnfr
    pfr = zray(p,n,dir)%trcfr
!
SELECT CASE (octant)
!
  CASE (1)
  !
    IF (p.le.(npltmp-nnstmp)) THEN
      IF (iii.eq.1) THEN
        dfr = 4*dirnum + dir
      ENDIF
    ELSE
      iiifr = ii + 1
      aphi(pfr,nfr,iii,iiifr,i,d,g)
    ENDIF
  ENDIF
albedo = alb5
aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
ELSE
  iiifr = iii - 1
  aphi2(pfr,nfr,iiifr,ii,i,d,g)
ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF(i.eq.1) THEN
      dfr = dirnum + dir
      albedo = alb1
      aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF(ii.eq.1) THEN
      dfr = 3*dirnum + dir
      albedo = alb3
      aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi2(pfr,nfr,iii,ii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
!
CASE (2)
!
IF (p.le.(npltm.nnstmp)) THEN
  IF (iii.eq.1) THEN
    dfr = 5*dirnum + dir
    albedo = alb5
    aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii - 1
    aphi2(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = dir
      albedo = alb2
      aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 2*dirnum + dir
      albedo = alb3
      aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi2(pfr,nfr,iii,ii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
!
CASE (3)
!
IF (p.le.(nptmp-nntmp)) THEN
  IF (iii.eq.1) THEN
    dfr = 6*dirnum + dir
    albedo = alb5
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iffr = iii - 1
    aphitmp = aphi2(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nntmp) THEN
    IF (i.eq.nx) THEN
      dfr = 3*dirnum + dir
      albedo = alb2
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphitmp = aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.ny) THEN
      dfr = dirnum + dir
      albedo = alb4
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii + 1
      aphitmp = aphi2(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ELSE
  IF (ii.eq.ny) THEN
    dfr = dirnum + dir
    albedo = alb4
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iifr = ii + 1
    aphitmp = aphi2(pfr,nfr,iii,iifr,i,d,g)
  ENDIF
ENDIF
ENDIF

CASE (4)

IF (p.le.(nptmp-nntmp)) THEN
  IF (iii.eq.1) THEN
    dfr = 7*dirnum + dir
    albedo = alb5
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iffr = iii - 1
    aphitmp = aphi2(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nntmp) THEN
    IF (i.eq.1) THEN
      dfr = 2*dirnum + dir
      albedo = alb1
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphitmp = aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.ny) THEN
      dfr = dir
      albedo = alb4
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii + 1
      aphitmp = aphi2(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ELSE
  IF (ii.eq.ny) THEN
    dfr = dir
    albedo = alb4
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iifr = ii + 1
    aphitmp = aphi2(pfr,nfr,iii,iifr,i,d,g)
  ENDIF
ENDIF
ENDIF
! CASE (5)!

IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.nz) THEN
    dfr = dir
    albedo = alb6
    aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii + 1
    aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.1) THEN
      dfr = 5*dirnum + dir
      albedo = alb1
      aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphi2 = aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 7*dirnum + dir
      albedo = alb3
      aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi2 = aphi2(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF

! CASE (6)!

IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.nz) THEN
    dfr = dirnum + dir
    albedo = alb6
    aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii + 1
    aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = 4*dirnum + dir
      albedo = alb2
      aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphi2 = aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.1) THEN
      dfr = 6*dirnum + dir
      albedo = alb3
      aphi2 = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii - 1
      aphi2 = aphi2(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
ENDIF
ENDIF
ENDIF

! CASE (7)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.nz) THEN
    dfr = 2*dirnum + dir
    albedo = alb6
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii + 1
    aphitmp = aphi2(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.nx) THEN
      dfr = 7*dirnum + dir
      albedo = alb2
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i + 1
      aphitmp = aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.ny) THEN
      dfr = 5*dirnum + dir
      albedo = alb4
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      iifr = ii + 1
      aphitmp = aphi2(pfr,nfr,iii,iifr,i,d,g)
    ENDIF
  ENDIF
ENDIF
ELSE
  IF (ii.eq.ny) THEN
    dfr = 5*dirnum + dir
    albedo = alb4
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iifr = ii + 1
    aphitmp = aphi2(pfr,nfr,iii,iifr,i,d,g)
  ENDIF
ENDIF
ENDIF

! CASE (8)
!
IF (p.le.(npltmp-nnstmp)) THEN
  IF (iii.eq.nz) THEN
    dfr = 3*dirnum + dir
    albedo = alb6
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    iiifr = iii + 1
    aphitmp = aphi2(pfr,nfr,iiifr,ii,i,d,g)
  ENDIF
ELSE
  IF(n.gt.nnytmp) THEN
    IF (i.eq.1) THEN
      dfr = 6*dirnum + dir
      albedo = alb1
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphitmp = aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ELSE
    IF (ii.eq.ny) THEN
      dfr = 4*dirnum + dir
      albedo = alb4
      aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
    ELSE
      ifr = i - 1
      aphitmp = aphi2(pfr,nfr,iii,ii,ifr,d,g)
    ENDIF
  ENDIF
ELSE
  IF (ii.eq.ny) THEN
    dfr = 4*dirnum + dir
    albedo = alb4
    aphitmp = aphi2(pfr,nfr,iii,ii,i,dfr,g) * albedo
  ELSE
    ifr = i - 1
    aphitmp = aphi2(pfr,nfr,iii,ii,ifr,d,g)
  ENDIF
ENDIF
ENDIF
ENDIF
ELSE
   ii fr = ii + 1
aphitmp = aphi2(pfr,nfr,iii,ii fr,i,d,g)
ENDIF
ENDIF
ENDIF
!
END SELECT
!
aphin = aphi tmp
!
!
RETURN
END

SUBROUTINE nextTrack(octant,plane,track,xcell,ycell,zcell, & & nnx tmp,nnst mp,flag)
!
!
!!!!!!!!!!!!!!!
USE scalars !
!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!
INTEGER octant
INTEGER plane
INTEGER track
INTEGER xcell,ycell,zcell
INTEGER nnx tmp,nnstmp
INTEGER flag
INTEGER xx,yy,zz
!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
xx = xcell
yy = ycell
zz = zcell
!
SELECT CASE (octant)
!
Octant 1
!
CASE (1)
   IF (track.gt.nnst mp) THEN
      zz = zz + 1
   ELSE
      IF(plane.le.nnx tmp) THEN
         xx = xx + 1
      ELSE
         yy = yy + 1
      ENDIF
   ENDIF
ENDIF
!
IF (((xx.gt.nx).or.((yy.gt.ny).or.(zz.gt.nz))) THEN
   flag = 1
ELSE
   xcell = xx
   ycell = yy
   zcell = zz
ENDIF
! Octant 2
!
CASE (2)
   IF (track.gt.nnstmp) THEN
      zz = zz + 1
   ELSE
      IF(plane.le.nnxtmp) THEN
         xx = xx - 1
      ELSE
         yy = yy + 1
      ENDIF
   ENDIF
!
IF ((xx.lt.1).or.((yy.gt.ny).or.(zz.gt.nz))) THEN
   flag = 1
ELSE
   xcell = xx
   ycell = yy
   zcell = zz
ENDIF
!
! Octant 3
!
CASE (3)
   IF (track.gt.nnstmp) THEN
      zz = zz + 1
   ELSE
      IF(plane.le.nnxtmp) THEN
         xx = xx - 1
      ELSE
         yy = yy - 1
      ENDIF
   ENDIF
!
IF ((xx.lt.1).or.((yy.lt.1).or.(zz.gt.nz))) THEN
   flag = 1
ELSE
   xcell = xx
   ycell = yy
   zcell = zz
ENDIF
!
! Octant 4
!
CASE (4)
   IF (track.gt.nnstmp) THEN
      zz = zz + 1
   ELSE
      IF(plane.le.nnxtmp) THEN
         xx = xx + 1
      ELSE
         yy = yy - 1
      ENDIF
   ENDIF
!
IF ((xx.gt.nx).or.((yy.lt.1).or.(zz.gt.nz))) THEN
   flag = 1
ELSE
   xcell = xx
   ycell = yy
   zcell = zz
ENDIF
! Octant 5
!
CASE (5)
  IF (track.gt.nnstmp) THEN
    zz = zz - 1
  ELSE
    IF (plane.le.nnxtmp) THEN
      xx = xx + 1
    ELSE
      yy = yy + 1
    ENDIF
  ENDIF
!
  IF ((xx.gt.nx).or.((yy.gt.ny).or.(zz.lt.1))) THEN
    flag = 1
  ELSE
    xcell = xx
    ycell = yy
    zcell = zz
  ENDIF
!
! Octant 6
!
CASE (6)
  IF (track.gt.nnstmp) THEN
    zz = zz - 1
  ELSE
    IF (plane.le.nnxtmp) THEN
      xx = xx - 1
    ELSE
      yy = yy + 1
    ENDIF
  ENDIF
!
  IF ((xx.lt.1).or.((yy.gt.ny).or.(zz.lt.1))) THEN
    flag = 1
  ELSE
    xcell = xx
    ycell = yy
    zcell = zz
  ENDIF
!
! Octant 7
!
CASE (7)
  IF (track.gt.nnstmp) THEN
    zz = zz - 1
  ELSE
    IF (plane.le.nnxtmp) THEN
      xx = xx - 1
    ELSE
      yy = yy - 1
    ENDIF
  ENDIF
!
  IF ((xx.lt.1).or.((yy.lt.1).or.(zz.lt.1))) THEN
    flag = 1
  ELSE
    xcell = xx
    ycell = yy
    zcell = zz
  ENDIF
CASE (8)
  IF (track.gt.nnstmp) THEN
    zz = zz - 1
  ELSE
    IF(plane.le.nnxtmp) THEN
      xx = xx + 1
    ELSE
      yy = yy - 1
    ENDIF
  ENDIF
  IF ((xx.gt.nx).or.((yy.lt.1).or.(zz.lt.1))) THEN
    flag = 1
  ELSE
    xcell = xx
    ycell = yy
    zcell = zz
  ENDIF
END SELECT

SUBROUTINE postTR3D(state)
USE files    !
USE scalars  !
USE xsects   !
USE neutrons !

INTEGER state          !
INTEGER nset           !
DOUBLE PRECISION nrsum !
DOUBLE PRECISION drsum !
DOUBLE PRECISION ncsum !
DOUBLE PRECISION dcsum !

ALLOCATE(rgnflux(ng,nxsets))
ALLOCATE(rgnabs(ng,nxsets))
ALLOCATE(rgnprod(ng,nxsets))
ALLOCATE(corflux(ng))
ALLOCATE(corabs(ng))
ALLOCATE(corprod(ng))
DO g = 1,ng
DO q = 1,nxsets
nrsum = 0.0
drsum = 0.0
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      IF (nset.eq.q) THEN
        nrsum = nrsum + sphi(iii,ii,i,g) * dx * dy * dz
        drsum = drsum + dx * dy * dz
      ENDIF
    END DO
  END DO
END DO
rgnflux(g,q) = nrsum / drsum
END DO
!
DO g = 1,ng
ncsum = 0.0
dcsum = 0.0
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      ncsum = ncsum + sphi(iii,ii,i,g) * dx * dy * dz
      dcsum = dcsum + dx * dy * dz
    END DO
  END DO
END DO
corflux(g) = ncsum / dcsum
END DO
!

DO g = 1,ng
DO q = 1,nxsets
nrsum = 0.0
drsum = 0.0
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      IF (nset.eq.q) THEN
        nrsum = nrsum + xsab(g,nset,state) * &
        & sphi(iii,ii,i,g) * dx * dy * dz
        drsum = drsum + dx * dy * dz
      ENDIF
    END DO
  END DO
END DO
!

437
rgnabs(g,q) = nrsum / drsum
END DO
END DO

DO g = 1,ng
ncsum = 0.0
dcsum = 0.0
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      ncsum = ncsum + xsab(g,nset,state) * &
      & sphi(iii,ii,i,g) * dx * dy * dz
dcsum = dcsum + dx * dy * dz
    END DO
  END DO
END DO
corabs(g) = ncsum / dcsum
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Calculate Core- & Region-Averaged Production Rate
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
DO g = 1,ng
  DO q = 1,nxsets
    nrsum = 0.0
drsum = 0.0
  DO i = 1,nx
    DO ii = 1,ny
      DO iii = 1,nz
        nset = xsmap(iii,ii,i)
        IF (nset.eq.q) THEN
          nrsum = nrsum + nxsf(g,nset,state) * &
          & sphi(iii,ii,i,g) * dx * dy * dz
drsum = drsum + dx * dy * dz
        ENDIF
      END DO
    END DO
  END DO
rgnprod(g,q) = nrsum / drsum
END DO
END DO
!

ncsum = 0.0
dcsum = 0.0
DO i = 1,nx
  DO ii = 1,ny
    DO iii = 1,nz
      nset = xsmap(iii,ii,i)
      ncsum = ncsum + nxsf(g,nset,state) * &
      & sphi(iii,ii,i,g) * dx * dy * dz
dcsum = dcsum + dx * dy * dz
    END DO
  END DO
END DO
corprod(g) = ncsum / dcsum
END DO
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Report Final Results to OUTPUT File
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
WRITE(ofile,10)   
WRITE(ofile,*) '                    '***************
WRITE(ofile,*) ' *Calculation Complete*'        
WRITE(ofile,*) '                    '***************
WRITE(ofile,10)   
WRITE(ofile,*) 'Final Transient State = ', state
WRITE(ofile,10)   
WRITE(ofile,*) 'Core-Averaged Scalar Flux:
WRITE(ofile,10)   
WRITE(ofile,*) '  Group        Value'
WRITE(ofile,*) '  -----    --------------
DO g = 1,ng
   WRITE(ofile,150) g,corflux(g)
END DO
WRITE(ofile,10)   
WRITE(ofile,*) 'Core-Averaged Absorption Rate:'
WRITE(ofile,10)   
WRITE(ofile,*) '  Group        Value'
WRITE(ofile,*) '  -----    --------------
DO g = 1,ng
   WRITE(ofile,150) g,corabs(g)
END DO
WRITE(ofile,10)   
WRITE(ofile,*) 'Core-Averaged Production Rate:'
WRITE(ofile,10)   
WRITE(ofile,*) '  Group        Value'
WRITE(ofile,*) '  -----    --------------
DO g = 1,ng
   WRITE(ofile,150) g,corprod(g)
END DO
WRITE(ofile,10)   
WRITE(ofile,*) 'Region-Averaged Scalar Flux:'
WRITE(ofile,10)   
WRITE(ofile,*) ' Region       Energy         Rgn-Avg'
WRITE(ofile,*) ' --------      ------      --------------
DO q = 1,nxsets
   DO g = 1,ng
      WRITE(ofile,200) q,g,rgnflux(g,q)
   END DO
END DO
WRITE(ofile,10)   
WRITE(ofile,*) 'Region-Averaged Absorption Rate:'
WRITE(ofile,10)   
WRITE(ofile,*) ' Region       Energy         Rgn-Avg'
WRITE(ofile,*) ' --------      ------      --------------
DO q = 1,nxsets
   DO g = 1,ng
      WRITE(ofile,200) q,g,rgnabs(g,q)
   END DO
END DO
WRITE(ofile,10)   
WRITE(ofile,*) 'Region-Averaged Production Rate:'
WRITE(ofile,10)   
WRITE(ofile,*) ' Region       Energy         Rgn-Avg'
WRITE(ofile,*) ' --------      ------      --------------

DO q = 1,nxsets
   DO g = 1,ng
      WRITE(ofile,200) q,g,rgnprod(g,q)
   END DO
END DO
CALL write3Dtr()

SUBROUTINE write3Dtr()
!
!!!!!!!!!!!!!!!!!!!!
USE files      !
USE scalars    !
USE tracking   !
USE xsects     !
USE neutrons   !
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!
INTEGER g        !
INTEGER i,ii,iii!
!!!!!!!!!!!!!!!!!!!!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE(ofile,*) ''
WRITE(ofile,*) 'Scalar Flux Distribution'
WRITE(ofile,*) ''
DO g = 1,ng
   DO iii = nz,1,-1
      WRITE(ofile,*) 'Group: ',g,' Level: ',iii
      DO ii = ny,1,-1
         WRITE(ofile,100) (sphi(iii,ii,i,g),i=1,nx)
      END DO
   END DO
END DO
!!!
WRITE(ofile,*) ''
WRITE(ofile,*) 'Fission Rate Distribution'
WRITE(ofile,*) ''
DO iii = nz,1,-1
   WRITE(ofile,*) 'Group: ',g,' Level: ',iii
   DO ii = ny,1,-1
      WRITE(ofile,100) (fd0(iii,ii,i),i=1,nx)
   END DO
END DO
WRITE(ofile,*) ''
WRITE(ofile,*) 'Scattering Source Distribution'
WRITE(ofile,*) ''
DO g = 1,ng
   }
DO iii = nz,1,-1
    WRITE(ofile,*) 'Group: ',g,' Level: ',iii
    DO ii = ny,1,-1
        WRITE(ofile,100) (ssav(iii,ii,i,g),i=1,nx)
    END DO
END DO
END DO
END DO
WRITE(ofile,*) ''
WRITE(ofile,*) 'Fission Source Distribution'
WRITE(ofile,*) ''
DO g = 1,ng
    DO iii = nz,1,-1
        WRITE(ofile,*) 'Group: ',g,' Level: ',iii
        DO ii = ny,1,-1
            WRITE(ofile,100) (fsav(iii,ii,i,g),i=1,nx)
        END DO
    END DO
END DO
WRITE(ofile,*) ''

! Format Statements
!
100 FORMAT(10(1x,es10.3e2))
!
END

MODULE files
!
! INPUT File [UNIT = 1, NAME = input.mok]
!
! must be specified by user prior to execution
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: ifile = 1
CHARACTER(9), PARAMETER :: ifname = 'input.mok'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! DATA File [UNIT = 2, NAME = data.ray]
!
...contains ray tracing results that are required
 as input for the MOCK-3D angular flux calculation
!
 stored in unformatted but compact form
 i.e, no annotation - just the raw data
!
 must be specified by user prior to execution
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: afile = 2
CHARACTER(8), PARAMETER :: afname = 'data.ray'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! OUTPUT File [UNIT = 3, NAME = output.mok]
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER, PARAMETER :: ofile = 3
CHARACTER(10), PARAMETER :: ofname = 'output.mok'
!
DEBUG File [UNIT = 4, NAME = debug.mok]

INTEGER, PARAMETER :: dfile = 4
CHARACTER(9), PARAMETER :: dfname = 'debug.mok'

SS RESTART File [UNIT = 5, NAME = rstrtSS.mok]

INTEGER, PARAMETER :: ssfile = 5
CHARACTER(11), PARAMETER :: ssfname = 'rstrtSS.moc'

TR RESTART File [UNIT = 6, NAME = rstrtTR.mok]

INTEGER, PARAMETER :: rtfile = 6
CHARACTER(11), PARAMETER :: rtfname = 'rstrtTR.mok'

END MODULE files

MODULE neutrons

! NCP --> Non-Collision Probability Table [exp{-cross*track}]
! Values are Tabulated During Problem Initialization
! and Found via Table-Lookup During Primary Execution
! to Avoid Unnecessary Repetition of the EXP() Function

DOUBLE PRECISION, ALLOCATABLE :: ncp(:,:,:,:,:) !

! APHI --> Local Angular Flux Distribution
! Array Dimension:
! (NPOMAX,NAZMAX,8*DIRNUM,NZ,NY,NX,NG)
! where:
! NAZMAX = Max # Planar Sections in Cell
! NPOMAX = Max # Tracks In Any Planar Section
! APHI2 is used differently for different transient methods...
! BD Constant-Constant Flux Approximation:
! Track-Averaged Angular Flux for the previous time step
! BD Constant-Linear or Constant-Exponential Approximations:
! APHI2 is the explicit Angular Flux for the previous time step

DOUBLE PRECISION, ALLOCATABLE :: aphi(:,;,:,;:;)
DOUBLE PRECISION, ALLOCATABLE :: aphi2(:,;,:,;:;)

! SSAV --> Scattering Source Term
! FSAV --> Prompt Fission Source Term

442
! PSAV  --> Delayed Fission Source Term
! PCONC --> Delayed Neutron Precursor Concentrations
!
! 3D Array Dimension: (NZ,NY,NX,NG)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: ssav(:,:,:,:)    
DOUBLE PRECISION, ALLOCATABLE :: fsav(:,:,:,:)    
DOUBLE PRECISION, ALLOCATABLE :: psav(:,:,:,:)    
DOUBLE PRECISION, ALLOCATABLE :: pconc(:,:,:,:)   
DOUBLE PRECISION, ALLOCATABLE :: pconold(:,:,:,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
SPHI -----> Scalar Flux Distribution for Current Time Step
SPHIOLD --> Scalar Flux Distribution for Previous Time Step
!
! 3D Array Dimension: (NZ,NY,NX,NG)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: sphi(:,:,:,:)    
DOUBLE PRECISION, ALLOCATABLE :: sphiold(:,:,:,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
FDNEW --> Fission Distribution for Current Time Step
FDOLD --> Fission Distribution for Previous Time Step
!
! 3D Array Dimension: (NZ,NY,NX,NG)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: fd0(:,:,:) !
DOUBLE PRECISION, ALLOCATABLE :: fd1(:,:,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
RGNFLUX --> Region-Averaged Scalar Flux
RGNABS  --> Region-Averaged Absorption Rate
RGNPROD --> Region-Averaged Production Rate
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: rgnflux(:,:) !
DOUBLE PRECISION, ALLOCATABLE :: rgnabs(:,:)  
DOUBLE PRECISION, ALLOCATABLE :: rgnprod(:,:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
CORFLUX --> Core-Averaged Scalar Flux
CORABS  --> Core-Averaged Absorption Rate
CORPROD --> Core-Averaged Production Rate
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: corflux(:) !
DOUBLE PRECISION, ALLOCATABLE :: corabs(:)  
DOUBLE PRECISION, ALLOCATABLE :: corprod(:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END MODULE neutrons

MODULE scalars
!
! ************************
! **--------------------**
! *|                    |*
! *|  input parameters  |*

443
TRFLG :: Transient Flag
   = 1 : Backward Differencing w/ Constant-Constant Approximation
   = 2 : Backward Differencing w/ Constant-Linear Approximation
   = 3 : Backward Differencing w/ Constant-Exponential Approximation
   = 4 : Forward Differencing w/ Constant-Constant Approximation
   = 5 : Forward Differencing w/ Constant-Linear Approximation
   = 6 : Forward Differencing w/ Constant-Exponential Approximation

INTEGER trflg!

CRITFLG :: Criticality Flag
   = 0 : No Correction
   = 1 : Assume initial state is critical
         divide all nu-fiss XS by keff

INTEGER critflg!

PCFLG :: Precursor Concentration Approximation Flag
   = 0 : Use Time Integrated Approximation
   = 1 : Use Explicit/Implicit Approximation

INTEGER pcflg!

CCS --> Convergence Criteria for the Scalar Flux, Inner Iteration
        (used in all methods)

DOUBLE PRECISION ccs!

CCA --> Convergence Criteria for the Angular Flux, Inner Iteration
        (used in all methods)

DOUBLE PRECISION cca!

CCO --> Convergence Criteria for the Outer Iteration
        (used in ...)

DOUBLE PRECISION cco!

ITERIMAX --> Max Iterations : Inner Iteration (Scatter Source)

INTEGER iterimax!

ITEROMAX --> Max Iterations : Outer Iteration (Fission Source)

INTEGER iteromax!

NX, NY, and NZ --> Number of Cells in
        x-, y-, and z-directions, respectively

INTEGER nx! Used in 2D and 3D Cartesian
INTEGER ny  ! Used in 2D and 3D Cartesian
INTEGER nz  ! Used in 3D Cartesian
!!!!!!!!!!!!

! ALB1 through ALB6 --> Albedo Boundary Coefficients
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION alb1  ! x- face
DOUBLE PRECISION alb2  ! x+ face
DOUBLE PRECISION alb3  ! y- face
DOUBLE PRECISION alb4  ! y+ face
DOUBLE PRECISION alb5  ! z- face
DOUBLE PRECISION alb6  ! z+ face
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! NSTATE --> Number of Transient States in Problem
! STATYP --> Type of State
!        = 0 : Constant Material Properties
!        = 1 : Ramp Material Properties
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER nstate
INTEGER lstate
INTEGER, ALLOCATABLE :: statyp(:, ,) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! LNGTIM --> Time b/w Long Outputs (State-Dependent)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: lngtim(:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Transient Calculation Parameters
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
DOUBLE PRECISION, ALLOCATABLE :: delta(:)  !
DOUBLE PRECISION, ALLOCATABLE :: timend(:)  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! *******************
! ***----------------* ***
| *| tracking parameters | *
| *| -------------------*
| *******************

! NGEO --> Flag: Problem Geometry Type
!        = 2 --> 2D Cartesian Geometry (not enabled)
!        = 3 --> 3D Cartesian Geometry
!!!!!!!!!!!!!!!!!!!!
INTEGER ngeo!
!!!!!!!!!!!!!!!!!!!!

! DIRNUM --> Number of Directions in Quadrature Set
!!!!!!!!!!!!!!!!!!!!
INTEGER dirnum!
!!!!!!!!!!!!!!!!!!!!

! DX, DY, and DZ --> Scalar Cell Dimensions
DOUBLE PRECISION dx ! x-dimension - used in 2D and 3D ray tracing
DOUBLE PRECISION dy ! y-dimension - used in 2D and 3D ray tracing
DOUBLE PRECISION dz ! z-dimension - used in 3D ray tracing only

!-------------------
! DEFINED DATA TYPE
!-------------------

! Data Type Specification for Quadrature Array

TYPE QUADINFO
  DOUBLE PRECISION wght !......Direction Weight
  DOUBLE PRECISION azim !.......Azimuthal Angle
  DOUBLE PRECISION dazi !......dA track spacing
  INTEGER nnx !......# Spans in x-dir
  INTEGER nny !......# Spans in y-dir
  INTEGER npu !..# Azi Periodic Units
  DOUBLE PRECISION pola !...........Polar Angle
  DOUBLE PRECISION dpol !......dP track spacing
  INTEGER nns !...# Spans in (xy)-dir
  INTEGER nnz !......# Spans in z-dir
END TYPE

! DIRSET is the working array for holding
! all problem quadrature data

TYPE(QUADINFO), ALLOCATABLE :: dirset(:)

INTEGER nazmax  ! Maximum # Tracks - Azimuthal Plane
INTEGER npomax  ! Maximum # Tracks - Polar Plane

! KINF --> Infinite Multiplication Factor
! KEFF --> Effective Multiplication Factor

DOUBLE PRECISION keff  !
DOUBLE PRECISION kinf  !
DOUBLE PRECISION eigen  !

! CORPWR --> Normalized Total Core Power
! POWINIT -> Initial Core Power (for normalization)
DOUBLE PRECISION corpwr !
DOUBLE PRECISION powinit !

! OUTTIM --> Absolute Time for Outer Iterate
!
DOUBLE PRECISION outtim !

! ITERIN ---> Inner Iteration Count (Scatter Iteration)
! ITEROUT --> Outer Iteration Count (Fission Source Iteration)
!
INTEGER iterin !
INTEGER iterout !

! ********************************
! *                    *
! *     parameters     *
! *                    *
! ********************************
!
Pi and her cousins (to 8 significant figures)
!
DOUBLE PRECISION, PARAMETER ::   pi = 3.1415927   !  Pi
DOUBLE PRECISION, PARAMETER ::  hpi = 1.57079633  !  Pi/2
DOUBLE PRECISION, PARAMETER ::  tpi = 6.2831853   !  2*Pi
DOUBLE PRECISION, PARAMETER ::  fpi = 12.5663706  !  4*Pi
DOUBLE PRECISION, PARAMETER :: pirt = 1.77245385  !  Root-Pi

END MODULE scalars

MODULE tracking
!
!
! DEFINED DATA TYPE
!-------------------
!
TYPE AZIRAYINFO          !
DOUBLE PRECISION trcln ! Track Length
INTEGER trcto ! Track TO which Track Couples
INTEGER trcfr ! Track FROM which Track Couples
END TYPE

END MODULE tracking

447
XYRAY is the variable for Azimuthal (2D) Ray Tracing

...track length [%TRCLN] represents the PARTIAL path length --> in 2D, polar component is applied during outer iteration

shape: (MAX#Tracks, #Directions)

TYPE(ANDROIDINFO), ALLOCATABLE :: xyray(:,,:) !

TYPE(AZIRAYINFO), ALLOCATABLE :: xyray(:,,:) !

ZRAY is the variable holds 3D Track Lengths
& Coupling Matrix

shape: (MAX#Tracks, MAX#Planes, #Directions)

TYPE(POLRAYINFO), ALLOCATABLE :: zray(:,:,:)

*note* a 2D calc. uses XYRAY only
a 3D calc. uses XYRAY and ZRAY

INTEGER, ALLOCATABLE :: npl(:,,:) ! #Tracks in plane (3D)

END MODULE tracking

MODULE xsects

NG --> Number of Energy Groups

INTEGER ng !

NDG --> Number of Delayed Neutron Precursor Groups

INTEGER ndg !

NXSETS --> Number of Cross-Section Sets

END MODULE xsects
INTEGER nxsets!

! XSMAP --> Matrix to Connect each Spatial Cell to a XS Set
!
INTEGER, ALLOCATABLE :: xmap(:,,:)!

! XSTR --> Macroscopic Transport Cross-Section
!
DOUBLE PRECISION, ALLOCATABLE :: xstr(:,;);

! NXSF --> Macroscopic nu-Fission Cross-Section
!
DOUBLE PRECISION, ALLOCATABLE :: nxsf(:,;);

! XSSC --> Macroscopic Group-Group Scattering Cross-Section
  (transport corrected)
!
DOUBLE PRECISION, ALLOCATABLE :: xssc(:,;,:);

! XSAB --> Macroscopic Absorption Cross-Section
!
DOUBLE PRECISION, ALLOCATABLE :: xsab(:,;);

! RAMPTR -->
! RAMPNF -->
! RAMPSS -->

INTEGER tdramp

DOUBLE PRECISION, ALLOCATABLE :: ramptr(:,;);
DOUBLE PRECISION, ALLOCATABLE :: rampnf(:,;);
DOUBLE PRECISION, ALLOCATABLE :: rampss(:,;,:);

! VEL --> Neutron Velocities
!
DOUBLE PRECISION, ALLOCATABLE :: vel(:)

! BETAL --> Group-Dependent Delayed Neutron Fraction
!
DOUBLE PRECISION, ALLOCATABLE :: betal();

! BETA --> Total Delayed Neutron Fraction
!
DOUBLE PRECISION beta!
! LAMBDA --> Delayed Neutron Precursor Decay Constants
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   DOUBLE PRECISION, ALLOCATABLE :: lambda(:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! CHIP --> Fraction of Prompt Neutrons Borne from Fission to Energy Group
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   DOUBLE PRECISION, ALLOCATABLE :: chip(:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! CHID --> Fraction of Neutrons Borne from Fission to Delay Group
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   DOUBLE PRECISION, ALLOCATABLE :: chid(:) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
END MODULE xsects
On March 28, 1979 I was attending first grade classes at Ore Valley Elementary School in Dallastown, Pennsylvania, a small town in York County that is no more than 15 miles south of Three Mile Island, as the crow flies. I remember that day clearly. I remember that after lunch we were not allowed to go outside for recess, even though it was a beautiful sunny day. I remember that the buses picked us up after school at the front door and that no one was allowed to stay outside for more than a few seconds. I remember that the only explanation we were given for these odd events was that there was “something bad in the air.” And, like any seven-year-old child would, I promptly went home and forgot about the whole thing. Only recently did I learn that my home and school were in the primary evacuation zone and that my parents had plans to take my little sister and me to our grandparents’ house in Shippensburg if the call should come. The call never came and for almost twenty years I carried this memory with no understanding as to why that day unfolded as it did. Of course, after nearly three decades of education, I now know what happened that day and why. I understand that, while the precautions were prudent, we were not in significant danger. Yet the fear that permeated my school was very real. Every one of us knew that something was wrong, but no adult could – or would – tell us what had scared them so. I will carry this memory throughout my professional career and seek to ensure that a day like that never happens again…