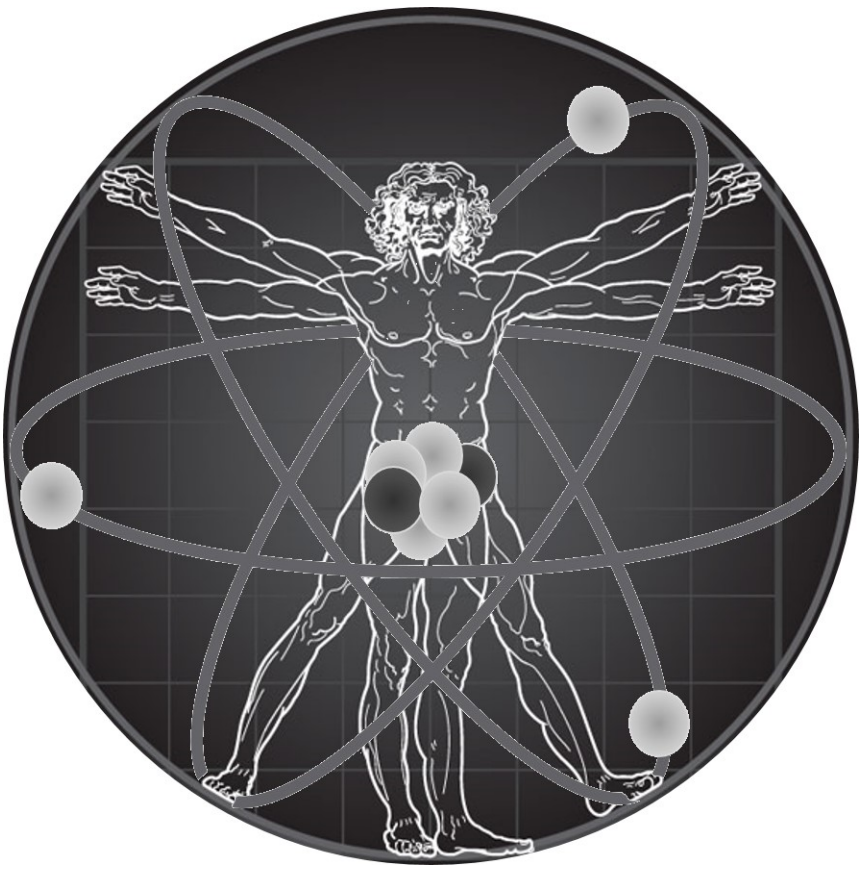
Computational Design in Nuclear Engineering

An Undergraduate Approach to Radiation Transport

Second Edition

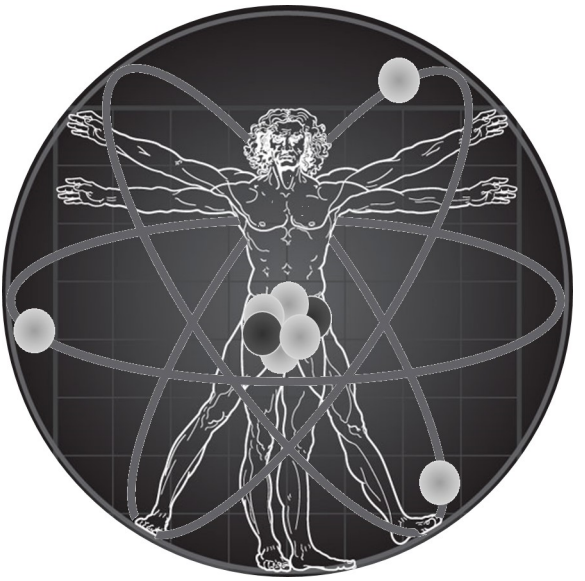


Computational Design in Nuclear Engineering

An Undergraduate Approach to Radiation Transport

Second Edition

Kenneth S. Allen Ph.D. P.E.
United States Military Academy



Self Publishers Worldwide
Seattle San Francisco New York
London Paris Rome Beijing Barcelona

For Information:
Dr. Kenneth S. Allen P.E.
Department of Physics and Nuclear Engineering
BLDG 753 Bartlett Hall
West Point, NY 10996
Kenneth.Allen@westpoint.edu

Copyright © 2026 Kenneth S. Allen Ph.D. P.E.
License: All rights reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the author.

Preface

Nuclear engineering, as a discipline, has always been at the forefront of computational simulations used for engineering design and analysis. This is largely due to the radioactivity of the materials used. Radioactivity causes unique safety concerns with respect to exposure and criticality, but it also provides a large well-known statistical set which lends itself to predictive modeling of natural events. The use of computational methods for nuclear engineering design has its roots in the earliest days of computing and has continued to evolve with each new technological advancement. Modern computational methods and equipment allow for the design of complex transport and criticality models with far less approximations and assumptions required for rapid solutions. The speed and accuracy of these models allows for better optimization of designs resulting in increased safety and greater financial return.

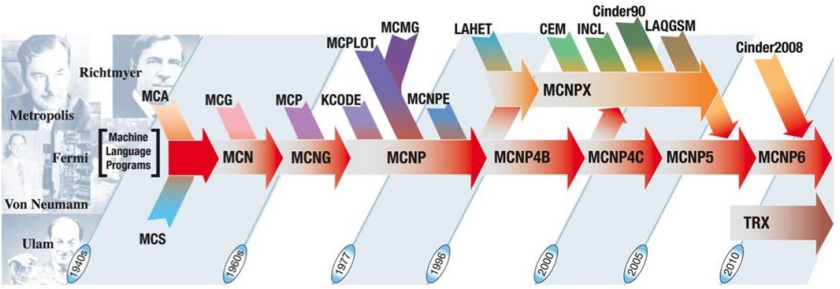


Figure 1: A timeline showing the development of the Monte Carlo Neutral Particle transport code. Source LA-UR-14-21281 (Hughes and James; 2014)

Purpose

The purpose of this text is to provide undergraduate students with an introduction to neutron transport and radiation interactions while developing a basic understanding of the design and use of advanced computational tools. These include Monte Carlo and deterministic approximations to the neutron transport

equation. The primary computer modeling code demonstrated in this text is Julia. The use of Julia could easily be substituted by any computer code language and compiler capable of doing conditional statements such as PYTHON, C++, or FORTRAN. The primary advantage to using an interpreted program like Julia is it eliminates the need to compile the source code for use on an operating system. The obvious drawback is it requires the user to run the solution in Julia and it cannot be easily transferred cross-platform for use as a stand-alone program.

Students using this text should have completed courses in calculus and physics and have a basic understanding of radiation and nuclear engineering. Although this text uses some of the most advanced computer codes offered such as MCNP6.3 and SCALE6.3, it does not require students to understand all of the detailed theory of radiation interactions and transport used to develop those codes. The text does introduce the neutron transport equation and provides basic background on interactions and radiation transport. Several textbooks are available with much more in-depth theoretical development of radiation transport and many of these texts are references for this work such as Computational Methods of Neutron Transport by E.E. Lewis and W.F. Miller, Jr. and Nuclear Reactor Analysis by James J. Duderstadt and Louis J. Hamilton. (Lewis and Miller; 1993; Duderstadt and Hamilton; 1976)

Whenever complex computer codes are used to solve problems there exists the potential "black box" paradigm often summarized as "garbage in = garbage out". This occurs when the user cannot distinguish between good and poor data because they do not adequately understand the problem being solved, the methods the computer is using to solve the problem or important assumptions and approximations being used and their effect on the solution. Whenever possible, this text will include basic hand calculations combined with analysis of the results of example problems that ensure the reader understands whether or not the results are reasonable. In addition, the text will make use of multiple codes to solve the same or similar problems. The method of validating results by use of more than one code allows the student to understand the similarities and differences between various solution techniques. Additionally, it provides a good method to demonstrate advantages and disadvantages to code packages and helps the student determine when one particular code package is best used for a problem or application.

Methods

The book begins with a review of radiation basics and introduces foundations of computational techniques. There is no expectation that students have a background in computer science or programming. Simple models and examples of

radiation decay and interactions are verified and serve as building blocks for the student's confidence and understanding of radiation transport.

Once the student is familiar with the basics, they will transition to deterministic and Monte Carlo techniques for solving simple criticality and shielding problems. These problems serve as the basis for developing the student's understanding of how the various computer codes solve problems and allow them to verify results. These problems become progressively more complex and include some advanced methods for solving problems with repeated lattices and performing variance reduction techniques.

The last introductory section involves a chapter on solid-body modeling in computer aided design. This section is not intended to serve as an entire CAD course, but rather as a means to develop a student's ability to adequately portray physical designs in three dimensions and in mechanical drawings. The use of CAD is essential to the transition from simple models to using unstructured mesh input and complex geometry models.

The remainder of the book focuses on advanced modeling of transport problems in deterministic and Monte Carlo programs and is suitable for more experienced users of computational tools. This includes using Attila4MC® to perform unstructured mesh calculations, plotting results using TecPlot® and ParaView®, performing radiographic calculations, and doing advanced calculations on reactor assemblies.

Bibliography

Duderstadt, J. J. and Hamilton, L. J. (1976). *Nuclear Reactor Analysis*, Wiley New York.

Hughes, III, H. G. and James, M. R. (2014). Mcnp6 class.
URL: *https://www.osti.gov/biblio/1122054*

Lewis, E. E. and Miller, W. F. (1993). *Computational Methods of Neutron Transport*, American Nuclear Society.

Table of Contents

1 Review of Radiation Basics **1**
- 1.1 Radioactive Decay . . . 1
- 1.2 Three Primary Decay Processes . . . 2
- 1.3 The Radioactive Decay Equation and Half-Life . . . 4
- 1.4 Calculating Radioactive Decay . . . 4
- 1.5 Basic Gamma Interactions with Matter . . . 13
- 1.6 Basic Neutron Interactions with Matter . . . 15
- 1.7 Video Tutorials . . . 22
- 1.8 Problems . . . 23

2 Introduction to Computation **25**
- 2.1 Program Design . . . 26
- 2.2 Logical Flowcharts . . . 27
- 2.3 Computer Language Levels . . . 29
- 2.4 Function Syntax for Operations . . . 30
- 2.5 Programming in Julia . . . 32
- 2.6 DOS Commands and Batch Files . . . 40
- 2.7 Macros In Textpad® . . . 45
- 2.8 Video Tutorials . . . 49
- 2.9 Problems . . . 50

3 Neutron Transport & Deterministic Codes **51**
- 3.1 Transport as it is Applied to Neutrons . . . 52
- 3.2 The Discrete Ordinates Approximation . . . 54
- 3.3 P_N and Diffusion Theory Approximations . . . 57
- 3.4 Deterministic Solutions to Transport Problems . . . 60
- 3.5 Variations to Discrete Ordinates Solutions . . . 70
- 3.6 Depletion Calculations in TRITON . . . 75
- 3.7 Video Tutorials . . . 81
- 3.8 Problems . . . 82

4 The Monte Carlo Method for Transport 85
4.1 Introduction to the Monte Carlo Technique . . . 85
4.2 Example Monte Carlo Calculation of π . . . 88
4.3 Precision, Accuracy, and Uncertainty in Monte Carlo . . . 93
4.4 Simple Monte Carlo Transport Example . . . 96
4.5 Video Tutorials . . . 105
4.6 Problems . . . 106

5 Introduction to MCNP 6.2 107
5.1 General Rules for MCNP Input . . . 108
5.2 Building the Input Deck–Define Materials . . . 110
5.3 Building the Input Deck–Surface Cards . . . 113
5.4 Building the Input Deck–Cell Cards . . . 118
5.5 Building the Input Deck–Data Cards VOL and SDEF . . . 120
5.6 Building the Input Deck–Tallies and Misc. Cards . . . 133
5.7 First Example–Estimate of π Using MCNP . . . 136
5.8 Plotting in MCNP . . . 140
5.9 Running A Model in MCNP . . . 144
5.10 Reading the MCNP Output . . . 145
5.11 Video Tutorials . . . 149
5.12 Problems . . . 150

6 Advanced MCNP Topics 153
6.1 Variance Reduction Techniques . . . 153
6.2 Stochastic Volume Calculation . . . 164
6.3 Temperature Effects in MCNP . . . 166
6.4 Burnup and Depletion . . . 171
6.5 Universes and Repeated Lattice Structures . . . 175
6.6 Using Vised for Live Particle Tracking . . . 184
6.7 Video Tutorials . . . 191
6.8 Problems . . . 192

7 Introduction to CAD 195
7.1 Basic Construction of Solid Bodies . . . 196
7.2 Formatting Part and Assembly Templates Solidworks® . . . 200
7.3 Creating a Part in Solidworks® . . . 202
7.4 Scaling and Changing Appearances of Parts . . . 212
7.5 Creating an Assembly from Parts . . . 216
7.6 Creating A Mechanical Drawing . . . 220
7.7 Using Solidworks® to Create 3-D Printed Objects . . . 227
7.8 Introduction to Computer Aided Manufacturing . . . 231
7.9 Video Tutorials . . . 234
7.10 Problems . . . 235

8 Advanced SCALE Topics **237**
8.1 Introduction to Advanced SCALE Topics . . . 238
8.2 Adding Geometry Units . . . 239
8.3 Creating a Hexagonal Lattice . . . 243
8.4 Working With Lattices . . . 257
8.5 Power and Flux Across a Lattice . . . 259
8.6 Developing Post-Processing Tools in Julia . . . 260
8.7 Creating Three-Dimensional KENO Models in SCALE . . . 269
8.8 Video Tutorials . . . 287
8.9 Problems . . . 288

9 Attila4MC® **291**
9.1 Introduction to Attila4MC® . . . 293
9.2 Creating Parasolids in Solidworks® . . . 294
9.3 Mesh Generation . . . 299
9.4 Analog Model Creation and Execution . . . 306
9.5 Creating a Source and Reusing a Model . . . 318
9.6 Tecplot Visulization of Results . . . 325
9.7 Variance Reduction Using CADIS . . . 331
9.8 Advanced Visualization Techniques . . . 343
9.9 Using a KCODE as a Source in Attila4MC® . . . 362
9.10 Video Tutorials . . . 370
9.11 Problems . . . 371

Index **375**

List of Figures

1 MCNP Timeline v

1.1 Nuclear Radii 2
1.2 Chart of the Nuclides 2
1.3 Decay of ^{137}Cs 3
1.4 Photoelectric Effect 13
1.5 Compton Scattering 14
1.6 Pair Production 14
1.7 Gamma Cross Sections 15
1.8 Neutron Interactions 16
1.9 Cross Section of ^{235}U at 300k 18
1.10 Cross Section of ^{235}U at 1200k 19
1.11 Resonance Cross Section of ^{235}U at 300k 19
1.12 Resonance Cross Section of ^{235}U at 1200k 20

2.1 Computer Languages 26
2.2 Logical Flowchart Example 28

3.1 Transport Example Analogy 52
3.2 Integration Approximation 55
3.3 Discrete Ordinates 55
3.4 Voxel Ray Positions 56
3.5 Common Neutron Energy Groups 57
3.6 Fick's Law 58
3.7 Mirrored and Periodic Boundary Conditions 60

4.1 Example Random Process 86
4.2 Fission Random Walk Process 87
4.3 Circle for Monte Carlo π Estimate 89
4.4 Quarter Circle for Monte Carlo π Estimate 90
4.5 Precision and Accuracy 94
4.6 Number Line Example 95
4.7 Simple Shield Example 96

5.1 Historical Punch Cards 107

5.2 Human Modeled in MCNP . . . 108
5.3 Neutron Cross Section Appendix G . . . 111
5.4 Compendium of Modeling . . . 113
5.5 Rectangular Box Model . . . 117
5.6 MCNP Union and Intersection . . . 119
5.7 MCNP PWR Assembly . . . 132
5.8 VISED Plot Window . . . 140
5.9 Visual Editor Plot XZ . . . 141
5.10 Visual Editor Plot XY . . . 142
5.11 Integrated Plotter Default Orientation . . . 143
5.12 Integrated Plotter in XZ . . . 143

6.1 Example Pig Model for MCNP . . . 155
6.2 Simple Geometry Splitting MCNP . . . 157
6.3 Stochastic Volume Calculation . . . 164
6.4 PWR Assembly Cutaway . . . 176
6.5 Checkerboard Model for Universes . . . 177
6.6 Rod Locations in PWR Assembly . . . 178
6.7 Symmetry in PWR Assembly . . . 182
6.8 Peak-to-Average Flux Map . . . 183
6.9 Peak-to-Average Power Map . . . 184

7.1 Historical Drafting Class . . . 195
7.2 Basic CAD Concept . . . 196
7.3 CAD Structure Developed . . . 197
7.4 Revolved Boss . . . 198
7.5 Applying a Fillet to Boss . . . 198
7.6 Applying a Photo Effect . . . 199
7.7 Example Mechanical Drawing . . . 199
7.8 Basic Part Workspace . . . 200
7.9 Workspace with Z-up . . . 202
7.10 Complete Shipping Container Part . . . 208
7.11 Complete Shipping Container Lid . . . 209
7.12 Final Shipping Container Lid . . . 211
7.13 Adjusted Final Cap Design . . . 213
7.14 Final Render of Shield Lid . . . 215
7.15 Final Render of Pig Base . . . 216
7.16 Final Render of Pig Assembly . . . 219
7.17 Final Drawing of Pig Assembly . . . 227
7.18 3-D Print Rendering of the Pig Base . . . 228
7.19 3-D Print Rendering of the Pig . . . 229
7.20 Pig Lid in Printer Software . . . 230
7.21 Pig Base in Printer Software . . . 230
7.22 Pig Lid in Printer Software . . . 231
7.23 MasterCAM® Simulator . . . 232

8.1 SCALE User's Manual 238
8.2 Rod Locations in PWR Assembly 239
8.3 VVER Reactor Assembly 244
8.4 Clinch River Breeder Reactor Assembly 246
8.5 Hexagonal Geometry Notation 247
8.6 Reactor Model 258
8.7 Example SCALE Assembly Output Zones 259
8.8 Example SCALE Assembly Output Bodies 260

9.1 Sabrina Plot 292
9.2 Attila4MC® from Silver Fir 293
9.3 M1 Abrams Tank Model 295
9.4 Attila4MC Solid Model 299
9.5 Attila4MC Contiguous Mesh 300
9.6 Attila4MC Dis-Contiguous Mesh 300
9.7 Cf-252 fission-neutron spectrum 319
9.8 Solid Model Modified with Berm 325
9.9 Dose Plot of Berm Problem. 325
9.10 HVLs for Various CADIS Shields. 333
9.11 TecPlot Image without Tally Multiplier 344
9.12 Data Set Window 345
9.13 Specify Equations Window 345
9.14 Adjusted Tank Example with Tally Multiplier 346

List of Tables

1.1 MT Numbers . . . 17

2.1 Logical Flowchart Symbols . . . 27
2.2 Computer Code Levels . . . 30
2.3 Common Julia Math Operators . . . 31
2.4 DOS Commands . . . 41
2.5 Command Prompt Shortcuts . . . 41
2.5 Command Prompt Shortcuts . . . 42
2.5 Command Prompt Shortcuts . . . 43

5.1 MCNP Input File Structure . . . 109
5.2 Standard Units in MCNP . . . 110
5.3 MCNP Surface Cards . . . 114
5.4 Macrobody Surface Cards . . . 115
5.5 Macrobody Surface Cards . . . 116
5.6 Source Variables for SDEF . . . 120
5.6 Source Variables for SDEF . . . 121
5.7 KCODE Input . . . 131
5.8 Tally Types in MCNP . . . 133

6.1 MCNP Statistical Checks . . . 154
6.2 MCNP BURN Card . . . 171
6.2 MCNP BURN Card . . . 172
6.2 MCNP BURN Card . . . 173
6.3 PWR Assembly Specifications . . . 178

7.1 Storage Pig Dimensions for CAD . . . 203

8.1 SFR Assembly Specifications . . . 245

Chapter 1

Review of Radiation Basics

1.1 Radioactive Decay

Radioactive decay is a mechanism by which an isotope spontaneously changes its nuclear form through disintegration in its nuclei. Radioactive isotopes are either naturally occurring or formed through interactive processes such as neutron capture. Regardless of their origin, the instability in the nucleus is caused by a ratio of neutrons and protons that does not support the balance between the short-ranged strong nuclear forces and long-ranged electrostatic forces. The strong forces are present in all nucleons (protons and neutrons) while the electrostatic forces are found only in charged nucleons (protons). Strong forces act over a distance of around 1×10^{-15} m or the diameter of a medium-sized nucleus. For light isotopes, the distances between all nucleons are close allowing for a neutron to proton balance of 1:1. As the nucleus grows with atomic number, the distances between protons from one side of the nucleus to the other become great enough that the electrostatic forces will start to pry the nucleus apart. To bind the nucleus, more neutrons are needed to provide additional strong forces to "glue" it together. An illustration of the differences in small and large nuclei is presented in Figure 1.1.

To provide the correct balance of forces, heavy isotopes will have ratios of neutrons to protons of up to 1:2 or more. The chart of the nuclides from the National Nuclear Data Center from Brookhaven National Laboratories is shown in Figure 1.2. Stable isotopes fall along the "line of stability", represented by the dark spots on the figure. Unstable isotopes will undergo decay processes in order to change their nucleus neutron-proton balance to move closer to the line of stability. Some isotopes will take several decays to eventually move to a stable isotope.

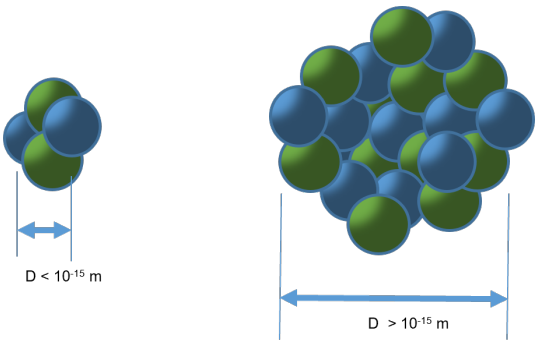


Figure 1.1: Visual illustration of the relative distances between protons on the outside of small and larger nuclei. The greater the gap distance between protons, the larger the effect of the electrostatic repulsive force.

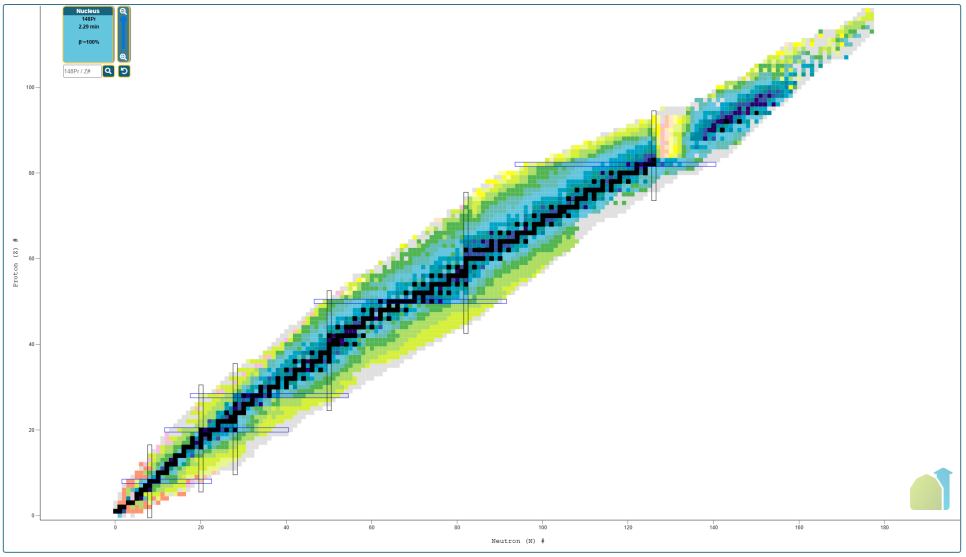


Figure 1.2: Chart of the Nuclides. A graphical representation of all known isotopes organized according to their number of nucleons and number of protons. The darkest squares represent stable isotopes.(NNDC; 2025)

1.2 Three Primary Decay Processes

The majority of unstable nuclides will decay by one of three processes —alpha, beta, or gamma decay. In alpha decay , the atom emits a helium nucleus consisting of two neutrons and two protons resulting in a decrease in the atomic number by two and the atomic mass by four (e.g. $^{216}\mathrm{Po} \rightarrow ^{212}\mathrm{Pb} + \alpha$). Most alpha emitters are heavy nuclides such as uranium and transuranics. Beta decay

can either be positive or negative. The sign of the beta decay refers to the charge of the beta particle emitted. A negative beta particle is an electron while the positive particle is a positron. During negative beta decay, the atom converts a neutron in the nucleus to a proton and emits an electron and an antineutrino. The resulting nucleus has the same atomic number as the parent but the number of protons increases by one and the number of neutrons decreases by one (e.g. $^{14}\mathrm{C} \rightarrow ^{14}\mathrm{N} + \beta^- + \bar{\nu}$). During positive beta decay, the atom converts a proton to a neutron and emits a positron and a neutrino. The result is a nucleus with an additional neutron and one less proton (e.g. $^{18}\mathrm{F} \rightarrow ^{18}\mathrm{O} + \beta^+ + \nu$). For both beta decay processes, the atomic mass number for the parent and daughter isotopes are the same. The third common decay process, gamma decay , occurs when the nucleus of the unstable atom transitions from one excited state to another and emits a photon. Gamma emissions often accompany another radioactive process such as a beta decay where the parent nuclide decays to a metastable daughter product with excess energy in the nucleus (e.g $^{137}\mathrm{Cs} \rightarrow ^{137m}\mathrm{Ba} + \beta^- + \bar{\nu} \rightarrow ^{137}\mathrm{Ba} + \gamma$ [662 keV]).

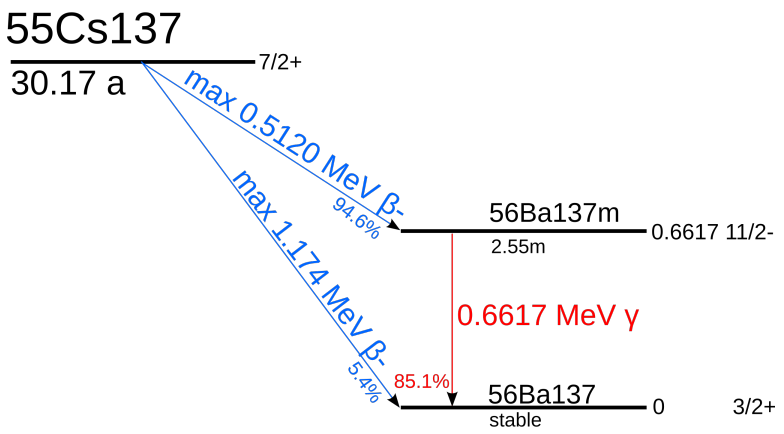


Figure 1.3: The radioactive decay scheme for ^{137}Cs. The majority (94.6%) of the decays result in a metastable daughter (^{137m}Ba) that will undergo a gamma decay to stable ^{137}Ba.(Podgoršak; 2009)

During the fission process, a significant amount of gamma rays are emitted in addition to neutrons and fission fragments. Because fission fragments are produced from the splitting of large nuclides, like uranium, they generally have many more neutrons than the number of protons that are necessary for stability. As a result, most fission fragments will undergo one or more negative beta decays to reach a stable isotope. Many of these will result in accompanying gamma emissions. A visual representation of the transitions of a radioactive isotope with the possible branches and their associated probabilities is called a decay scheme. Figure 1.3 is an example of the ^{137}Cs radioactive decay scheme.

1.3 The Radioactive Decay Equation and Half-Life

The radioactive decay of a given nucleus, such as ^{137}Cs, is stochastic in nature meaning that the process occurs randomly. Although a random process for any given nucleus, through experimental observation and counting, a dataset can be compiled that will accurately predict the time it takes for a given isotope to decay. This is possible because radioactive isotopes have nearly constant decay times that do not depend on the environment or age of the nucleus. The decay time only depends on the particular radioactive isotope species. The decay constant (λ) represents the rate of decay of atoms with respect to time and has units of inverse time (i.e. sec^{-1}). With very large numbers, such as the number of atoms of an isotope present in a sample at a given initial time (N_0), it is possible to accurately predict the number of atoms that have not underwent decay ($N(t)$) after some time (t) using the exponential decay equation, Equation 1.1. (Duderstadt and Hamilton; 1976)

$$N(t) = N_0 e^{-\lambda t} \tag{1.1}$$

The decay constant can be related to the average amount of time it takes for half of the radioactive sample population to decay. This time is referred to as the half-life of the species and its relationship to the decay constant is shown in Equation 1.2.

$$T_{1/2} = \frac{\ln 2}{\lambda} \tag{1.2}$$

Both values of decay constant and half-lives for radionuclides are widely tabulated in texts and online. One excellent source for information is the National Nuclear Data Center's interactive Chart of the Nuclides found at https://www.nndc.bnl.gov. (NNDC; 2025)

In addition to calculating the half-life, the decay constant can be multiplied by the number of atoms in a sample to determine its rate of radioactive decay, or activity (α) given by Equation. 1.3.

$$\alpha = \lambda N_0 \tag{1.3}$$

Because the units for the decay constant are inverse time, activity is reported in decays per unit time. Standard units for activity are the Becquerel (decay/sec) in SI units or Curie which is 3.7×10^{10} Bq or roughly the activity in 1 gram of radium.

1.4 Calculating Radioactive Decay

If all radioactive isotopes underwent a single radioactive decay to reach stability, it would be easy to calculate the number of atoms remaining after a given time

and the number of daughter atoms produced using Equation 1.1. However, as mentioned earlier, many radioactive species undergo several transitions before they become stable. The half-lives of each of those intermediate decay products are not equal. Therefore, a series of linear independent differential equations must be set up and solved, in order to predict the daughter products produced after a given time. The solution to the set of differential equations characterizing a decay chain is known as the Bateman equations after the original solution published in 1910 by Harry Bateman using Laplace transforms. (Bateman; 1910) Since Bateman's first work, several others have published techniques for solving decay chains. A method for solving a set of equations for the decay of ^{214}Po to ^{206}Pb is presented in Example 1.1 using Julia inside a Jupyter Notebook. Note in order for the Julia kernel to work in the Jupyter Notebook, the user must add the IJulia package to their Julia library. Additional packages required for this example include DifferentialEquations, Plots, DataFrames and StatsPlots.

In this text, we will use Julia within a Jupyter Notebook to demonstrate mathematical solutions that generally do not require user input. When we desire an interactive environment, we will generate scripts using Visual Studio Code to program in Julia or another language. Instructions on installing and running Julia can be found at https://julialang.org. A similar approach to solving this example could be achieved using Python or a mathematical program such as Matlab® with little modification.

Example 1.1
The isotope ^{214}Po decays to ^{206}Pb (stable) through the following transitions:

$$\begin{aligned}
^{214}\text{Po} &\rightarrow ^{210}\text{Pb} + \alpha \quad & T_{1/2} &= 1.643 \times 10^{-4}\ \text{s} \\
^{210}\text{Pb} &\rightarrow ^{210}\text{Bi} + \beta^- \quad & T_{1/2} &= 22.20\ \text{y} = 8103\ \text{d} \\
^{210}\text{Bi} &\rightarrow ^{210}\text{Po} + \beta^- \quad & T_{1/2} &= 5.012\ \text{d} \\
^{210}\text{Po} &\rightarrow ^{206}\text{Pb} + \alpha \quad & T_{1/2} &= 138.376\ \text{d}
\end{aligned}$$

Setup and solve a system of linear first-order differential equations for the decay of 1.0 gram of ^{214}Po to ^{206}Pb using Julia and plot the results showing the amount of each daughter present over time from zero to 20 years or 7300 days.

Solution:

First, write the appropriate equations that describe the number of nuclides present at any given time:

$$\frac{dN_{^{214}Po}}{dt} = -\lambda_{^{214}Po} N_{^{214}Po} + R_{^{214}Po}$$

$$\frac{dN_{^{210}Pb}}{dt} = -\lambda_{^{210}Pb} N_{^{210}Pb} + \lambda_{^{214}Po} N_{^{214}Po} + R_{^{210}Pb}$$

$$\frac{dN_{^{210}Bi}}{dt} = -\lambda_{^{210}Bi}N_{^{210}Bi} + \lambda_{^{210}Pb}N_{^{210}Pb} + R_{^{210}Bi}$$

$$\frac{dN_{^{210}Po}}{dt} = -\lambda_{^{210}Po}N_{^{210}Po} + \lambda_{^{210}Bi}N_{^{210}Bi} + R_{^{210}Po}$$

$$\frac{dN_{^{206}Pb}}{dt} = -\lambda_{^{206}Pb}N_{^{206}Pb} + \lambda_{^{210}Po}N_{^{210}Po} + R_{^{206}Pb}$$

Where R represents the production rate of that nuclide. For this problem, the given initial amount of ^{214}Po is fixed and the production rate is zero for all of the nuclides aside from their production from radioactive decay of their parent. Also, ^{206}Pb is stable with a decay constant of zero. Because the half-life for ^{214}Po is $\gg$ than the half-life for ^{210}Pb, it can be assumed that 100% of the ^{214}Po decays to ^{210}Pb before any significant decay of ^{210}Pb This simplifies the differential equations to:

$$\frac{dN_{^{210}Pb}}{dt} = -\lambda_{^{210}Pb}N_{^{210}Pb}$$

$$\frac{dN_{^{210}Bi}}{dt} = -\lambda_{^{210}Bi}N_{^{210}Bi} + \lambda_{^{210}Pb}N_{^{210}Pb}$$

$$\frac{dN_{^{210}Po}}{dt} = -\lambda_{^{210}Po}N_{^{210}Po} + \lambda_{^{210}Bi}N_{^{210}Bi}$$

$$\frac{dN_{^{206}Pb}}{dt} = +\lambda_{^{210}Po}N_{^{210}Po}$$

The necessary information is present to create a solution in Julia. Begin by creating an new Jupyter Notebook using the Julia kernel. For this case, we have called it “BatemanExample.ipynb”.

Within a Jupyter Notebook, the user can change the cell function using the drop-down or selecting the "cell" $\rightarrow$ "cell type" menu option. This is particularly useful to add titles and annotations to a series of calculations. Changing the cell type from "code" to "markdown" allows the user to input text, figures, or other information that will not be executed. This is similar to including comments in a computer language, but has greater capability. In a markdown cell, the user can include equations written in LaTeX and, when the user selects "run" for that cell it will display the solution in math-type. Note that only the plot output is presented in the notebook displayed below to conserve space.

Solving Bateman Equations for Decay of Po-214 Over 20 Years Using Julia

Load Julia modules needed for calcualtions

In [1]:
```
using DifferentialEquations
using Plots; gr()
using DataFrames
using StatsPlots
```

Set up the ODEs - See Example 1.1 from textbook

$$\frac{dN_{^{210}Pb}}{dt} = -\lambda_{^{210}Pb} N_{^{210}Pb}$$

$$\frac{dN_{^{210}Bi}}{dt} = -\lambda_{^{210}Bi} N_{^{210}Bi} + \lambda_{^{210}Pb} N_{^{210}Pb}$$

$$\frac{dN_{^{210}Po}}{dt} = -\lambda_{^{210}Po} N_{^{210}Po} + \lambda_{^{210}Bi} N_{^{210}Bi}$$

$$\frac{dN_{^{206}Pb}}{dt} = +\lambda_{^{210}Po} N_{^{210}Po}$$

In [ ]:
```
function btmn!(dN, N, p, t)
    dN[1] = -(log(2)/8103)*N[1]
    dN[2] = -(log(2)/5.012)*N[2]+(log(2)/8103)*N[1]
    dN[3] = -(log(2)/138.376)*N[3]+(log(2)/5.012)*N[2]
    dN[4] =  (log(2)/138.376)*N[3]
end
```

Set the inital conditions where there is 1.0 g of Pb-210 and 0.0 g of the daughters to start

In [ ]:
```
N0 = [1.0,0.0,0.0,0.0]
```

Read in the intial mass and time boundary conditons to the function from 0 to 7300 days (20 years)

In [ ]:
```
tspan = (0.0,7300.0)
prob = ODEProblem(btmn!,N0,tspan)
```

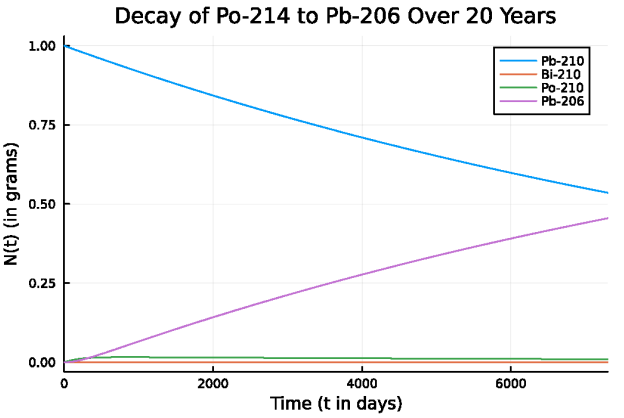
Solve the ODEs

In [ ]:
```
sol = solve(prob,Tsit5())
```

Plot the results

In [6]:
```
plot(sol, linewidth=1,
    title="Decay of Po-214 to Pb-206 Over 20 Years",
    xaxis="Time (t in days)", yaxis="N(t) (in grams)",
    labels=["Pb-210" "Bi-210" "Po-210" "Pb-206"])
```

Out[6]:



The Plot works well to show the mass in grams over time. However, if we desire activity, we will need to convert mass at each timestep to activity. We could save the solution results to an array and convert the data. Another option is to save the resutls as a DataFrame. From there, we can plot the columns of the DataFrame using the StatPlot module and multipy each column by the conversion constant from mass to activity for that isotope. Note that Pb-206 will not be plotted as it is stable and does not have a half-life.

Save solution as a DataFrame

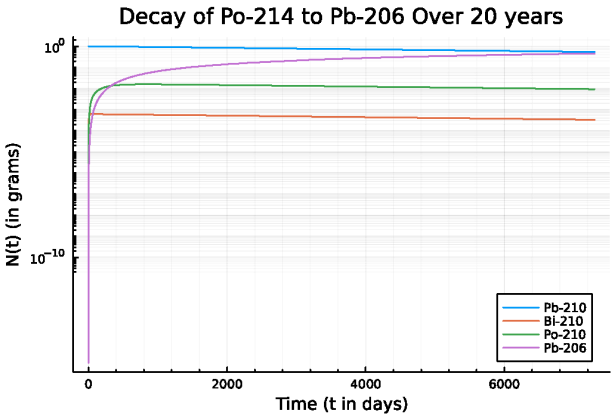
In [ ]:
```
df = DataFrame(sol)
```

Plot DataFrame using StatPlots in a semi-log plot for y-axis.

In [8]:
```
@df df plot(:timestamp,
[:value1.*1e-15 :value2.*1e-15 :value3.*1e-15 :value4.*1e-15],
    labels=["Pb-210" "Bi-210" "Po-210" "Pb-206"],
```

```
    title="Decay of Po-214 to Pb-206 Over 20 years",
    xaxis="Time (t in days)", yaxis=:log,
    ylabel="N(t) (in grams)",
    legend=:bottomright,minorgrid=true)
```

Out[8]:



Create conversions from radioactive isotopes in grams to activity in Bq starting with Activity

$$A(t) = \lambda N(t)$$

Use the definition of decay constant to convert from half-life (NOTE: must be in seconds to achieve Bq)

$$\lambda = \frac{\ln 2}{T_{1/2}}$$

The number of atoms in each isotope is related the the mass which varies over time as follows:

$$N(t) = \frac{m(t)\ (\mathrm{g}) \cdot N_A \left(\frac{\mathrm{atoms}}{\mathrm{mol}}\right)}{A\left(\frac{\mathrm{g}}{\mathrm{mol}}\right)}$$

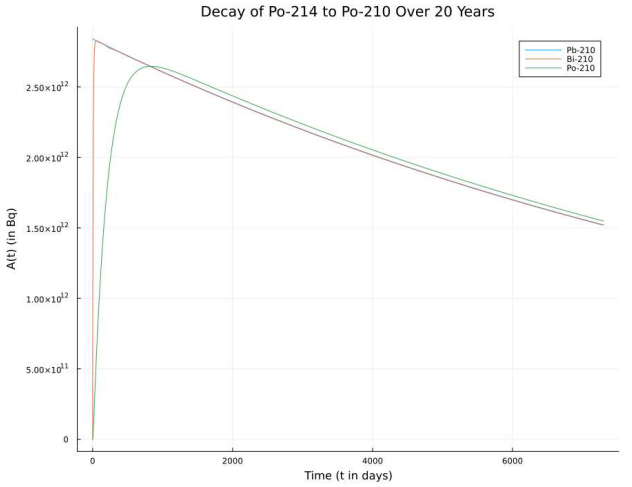
Therefore, the conversion for the ith isotope from mass to acivitiy is given by:

$$C_i = \frac{\ln 2 \cdot N_A \left(\frac{\text{atoms}}{\text{mol}}\right)}{T_{(1/2)i} \cdot A_i \left(\frac{\text{g}}{\text{mol}}\right)}$$

In [ ]:
```
N_A = 6.0221408e23
A = 210 #All isotopes have the same atomic
        #mass number in this example
T1 = 8103*24*3600  #convert half-life to
                   #seconds with 24h and 3600s
T2 = 5.012*24*3600
T3 = 138.376*24*3600

C1 = (log(2)*N_A)/(T1*A)
C2 = (log(2)*N_A)/(T2*A)
C3 = (log(2)*N_A)/(T3*A)
```

In [10]:
```
@df df plot(:timestamp,
    [:value1*C1 :value2*C2 :value3*C3],
    labels=["Pb-210" "Bi-210" "Po-210"],
    title="Decay of Po-214 to Po-210 Over 20 Years",
    xaxis="Time (t in days)", yaxis="A(t) (in Bq)",size = (900, 700),
```

Out[10]:



The single decay chain, as in Example 1.1, is a simple decay problem when compared to actual problems encountered where there are multiple radioactive species mixed together such as in spent nuclear fuel assemblies. Because the decay constants for radioisotopes are known, the solutions to Bateman equations for decay chains lend themselves to computational methods. One of the most well-known programs that can solve for radioactive decay is the Oak Ridge Isotope Generator or ORIGEN. The ORIGEN code is distributed as part of the SCALE package. The solution to the same decay chain calculated in Example 1.1 using Julia is demonstrated in Example 1.2 using ORIGEN in SCALE6.3.

Example 1.2
Calculate the decay of 1 gram of ^{214}Po to ^{206}Pb using ORIGEN and plot the results showing the amount of each daughter present over time from zero to 20 years or 7300 days.

Solution: Open the FULCRUM interface for SCALE6.3 and create a new document with the following input:

```
'------------------DESCRIPTION-----------------------------------------
'
' Example of the simple decay of Po-214
' The initial amount of Po-214 is 1 gram or 1/214 moles = 0.0046729 mole
'----------------------------------------------------------------------
'---------------------ORIGEN-----------------------------------------
=origen
case{
lib{file="end7dec"}
mat{iso=[po214=(0.0046729)]}
time{
  t=[ 1.0 7.0 31.0 180.0 365.0 1095.0 3285.0 7300.0]
  units="days"
  start=0.0
}

}
End
```

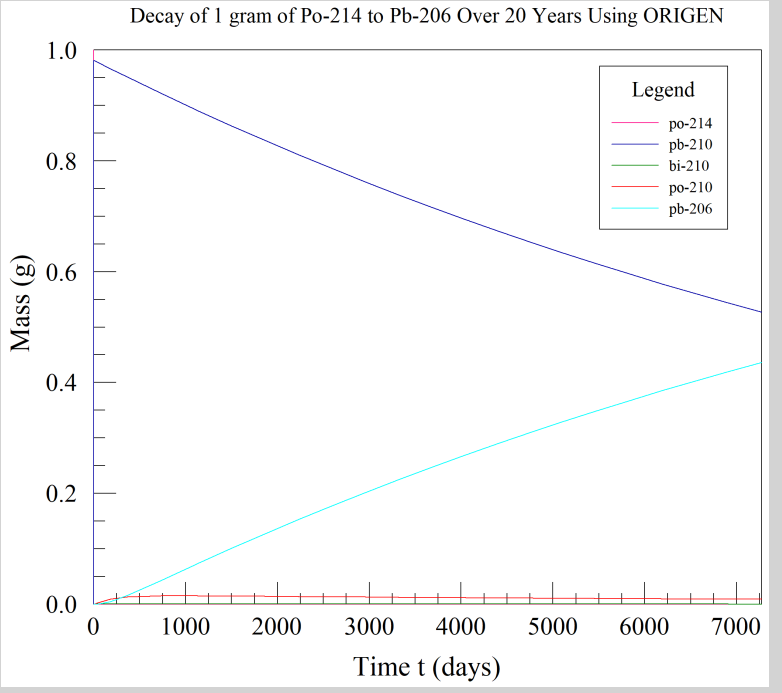
This input file directs SCALE to execute an ORIGEN calculation using the Evaluated Nuclear Data File 7 library for a material containing 1/214 mole (i.e. 1.0 gram) of ^{214}Po during the time steps specified. The default units for input are moles. The user can specify alternate units to include activity (Becquerels or Curies) or mass (grams). The user can also specify multiple radioisotopes in the same input file. ORIGEN requires time steps to generally follow the "rule of threes" where each time step is no longer than 3 times the previous time. The unit of time is set to "days" and the start time is 0.0. After executing SCALE the following table is in the results:

Nuclide concentrations in grams for case '1' (#1/1)

(relative cutoff; integral of concentrations over time > 1.00E-06 % of integral of all concentrations over time)

	0.000d	1.000d	7.000d	31.000d	180.000d	365.000d	1095.000d	3285.000d	7300.000d
he-4	0.0000E+00	1.8704E-02	1.8704E-02	1.8707E-02	1.8798E-02	1.9010E-02	2.0073E-02	2.3029E-02	2.7203E-02
pb-206	0.0000E+00	9.1773E-09	2.5772E-06	1.2135E-04	4.8087E-03	1.5761E-02	7.0453E-02	2.2258E-01	4.3733E-01
pb-210	0.0000E+00	9.8117E-01	9.8067E-01	9.7866E-01	9.6627E-01	9.5111E-01	8.9357E-01	7.4101E-01	5.2573E-01
bi-210	0.0000E+00	7.8338E-05	3.7604E-04	5.9696E-04	5.9764E-04	5.8827E-04	5.5268E-04	4.5832E-04	3.2517E-04
po-210	0.0000E+00	5.5325E-06	2.0834E-04	1.8762E-03	9.4834E-03	1.3489E-02	1.5307E-02	1.2654E-02	8.9777E-03
po-214	1.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
totals	1.0000E+00	9.9996E-01	9.9996E-01	9.9996E-01	9.9996E-01	9.9996E-01	9.9996E-01	9.9973E-01	9.9957E-01

The values from this table are easily imported into a spreadsheet and plotted or they could be plotted using the OPUS function within SCALE. The results are presented on the below and they exactly match those generated using Julia with the notable exception at just after t = 0. This is because the ORIGEN code accounted for the initial rapid decay of ^{214}Po which was neglected in the Julia model.



The straight decay of an isotope presented in Example 1.2 is one of the simplest applications of ORIGEN. The code has several more features that

allow it to predict the waste composition in reactor fuel based upon the fuel's initial enrichment and burnup history. The final composition of nuclear waste post-irradiation in a reactor is essential to the calculation of spent fuel cooling pools and dry casks designed to store waste.

1.5 Basic Gamma Interactions with Matter

Gamma rays are photons that originate in the nucleus of an atom. How they interact with matter is well-understood and modeled. In general, gamma interactions with matter depend strongly on the energy of the gamma ray and the atomic number, Z, of the atoms in the media they are traversing. There are three primary gamma ray interactions. They include the photoelectric effect, Compton scattering and pair production.

The photoelectric effect , also called photoelectric absorption, occurs when a photon interacts with an atom and disappears causing the atom to eject a photo-electron from one of its orbital shells. As shown in Figure 1.4, the interaction is with the whole atom versus the nucleus or the orbital electrons. The photoelectric effect is normally dominate at lower photon energies (E $<$ 0.1 MeV) and increases with higher atomic number (Z). Often when an inner orbital electron is ejected, an outer electron will fall in to fill the vacancy. The difference in binding energy from the outer to inner electron shell is conserved through the emission of a characteristic x-ray. This x-ray has the exact energy corresponding to the difference in binding energy between the two shells.

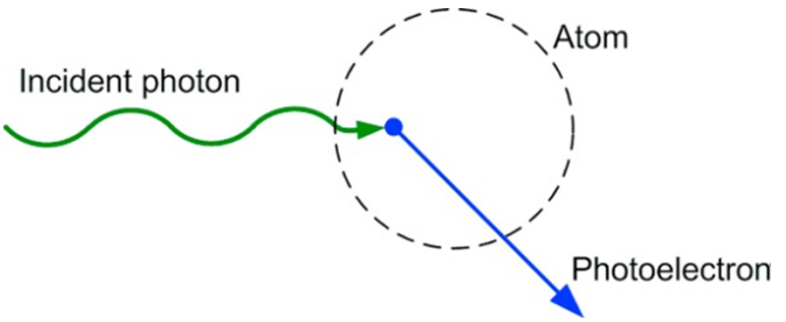


Figure 1.4: Photoelectric effect of a photon with an entire atom resulting in the absorption of the photon and ejection of the photoelectron. (Murray and Holbert; 2016)

Compton scattering occurs when a photon elastically scatters off an orbital electron. In the scattering process, energy is transferred from the incident photon to the electron which recoils (Figure 1.5). The scattered photon has an energy equal to the difference between the incident photon energy and the energy of the scattered electron. Compton scattering is most prominent for

photons of intermediate energies from 0.1 MeV up to 1 MeV. The angle of the scattered photon with respect to the incident photon is dependent on the energy of the photon and is described in detail in Example 2.2.

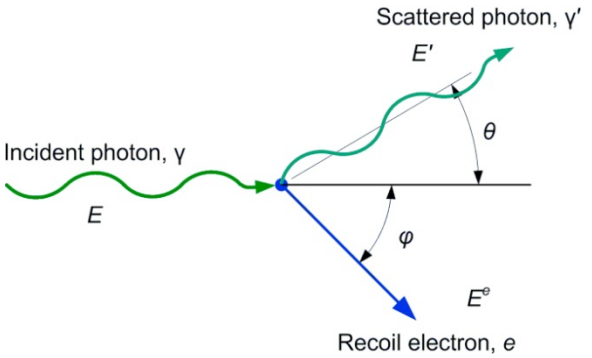


Figure 1.5: Compton scattering of a photon off of an orbital electron. The total energy is conserved between the scattered photon and recoil electron. (Murray and Holbert; 2016)

Pair Production occurs when an incident photon with an energy greater than 2 times the rest mass energy of an electron (E > 1.02 MeV) interacts with the nucleus of an atom and is absorbed (Figure 1.6). Energy is conserved by the atom emitting two particles (an electron and a positron). The threshold energy is necessary in order to have enough energy to convert to mass of two particles each with a rest mass of 511 keV. After emission, the positron will normally travel a short distance before it interacts with an electron resulting in the annihilation of the two particles and the creation of two photons, each with an energy of 511 keV traveling in opposite directions.

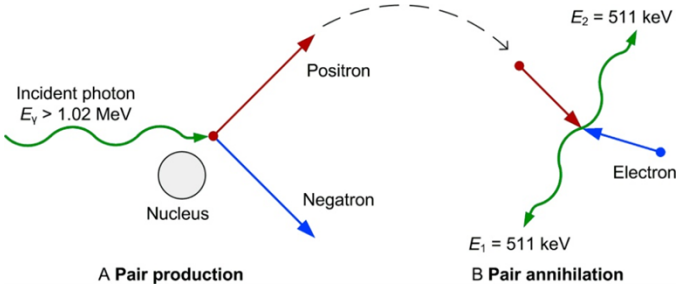


Figure 1.6: Pair production interaction occurs when incident photon with E > 1.02 MeV is absorbed by the nucleus of an atom resulting in the production of an electron and positron pair. Positron will undergo annihilation a short distance from the initial interaction. (Murray and Holbert; 2016)

The total probability of an interaction occurring in a path length is expressed

as the linear attenuation coefficient (μ) given in Equation 1.4 and has units of inverse length (cm^{-1}). The total attenuation coefficient is the sum of the photoelectric (τ), Compton (σ) and pair production (κ) attenuation coefficients for that particular energy photon in that media. The relationship of photon cross section probability with the energy of the incident photon and the atomic number, Z, of the material is presented in Figure 1.7.

$$\mu = \tau + \sigma + \kappa \tag{1.4}$$

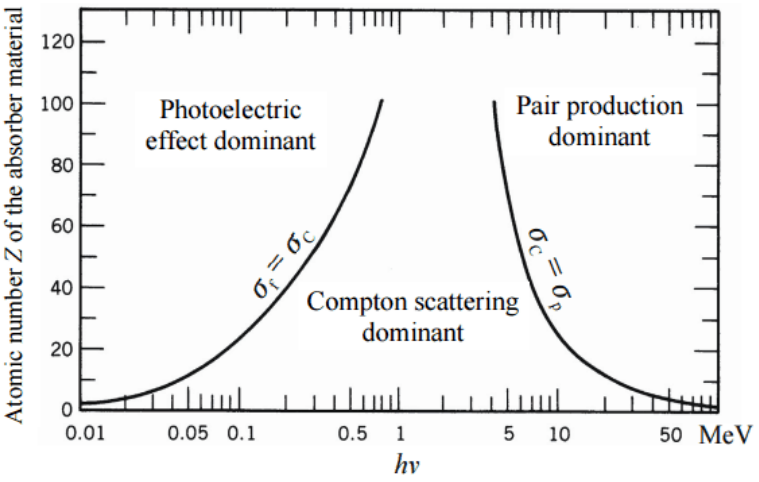


Figure 1.7: Importance of gamma interactions as a function of atomic number (Z) and energy. In this case, energy is represented by the product of Plank's Constant (h) and the frequency of the photon (ν) (Evans; 1955)

1.6 Basic Neutron Interactions with Matter

Neutron interactions can be characterized into two broad groups – scatter and absorption. Figure 1.8 is a representation of the hierarchy of neutron interactions. Scatter includes elastic and inelastic scatter . In elastic scatter collisions, energy is transferred from the neutron to the nucleus of the scattering medium which recoils according to the laws of classical mechanics. In inelastic scatters, some of the energy of the incident neutron is transferred to the nucleus causing the target atom to become excited.

Absorption interactions include all the interactions in which the target atom absorbs the neutron. These include fission (n, f), radiative capture (n, γ) and the interactions where the neutron is absorbed and another particle or multiple particles are emitted such as (n, 2n), (n, 3n), (n, p) and (n, α) interactions. The probability of an interaction occurring depends strongly on the neutron energy

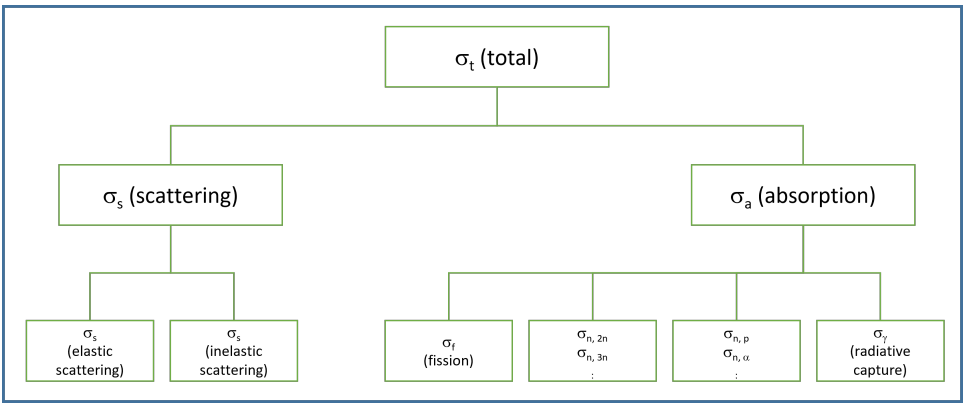


Figure 1.8: Neutron interaction hierarchy. Interactions are broken into two large groups – scattering and absorption.(Duderstadt and Hamilton; 1976)

and the isotope of the target atom. The interaction probability is represented by a quantity referred to as the microscopic cross section (σ) which has units of area in either cm^2 or commonly barns where 1 barn $= 1 \times 10^{-24}$ cm^2. The microscopic cross section for isotopes is measured and tabulated and based upon a single atom of that isotope. To get a linear interaction probability similar to the linear attenuation coefficient referenced for gamma rays, the microscopic cross section is multiplied by the number density (N) of the given material. The number density is calculated using the physical density (ρ) of the material, Avogadro's number (N_A) and the atomic mass (M) of the isotope as shown in Equation 1.5.

$$N = \frac{\rho N_A}{M} \tag{1.5}$$

The product in Equation 1.6 represents probability of interaction per unit length and is referred to as the macroscopic cross section (Σ) and has units of inverse distance (cm^{-1}).

$$\Sigma = N\sigma \tag{1.6}$$

While gamma interaction cross sections can be calculated directly based upon derived and empirical formulas, neutron cross sections are more complicated and years of theoretical and experimental work has been performed to tabulate the various cross sections for hundreds of isotopes. In the United States, Brookhaven National Laboratories publishes the Evaluated Nuclear Data File (ENDF) which serves as the single source for the majority of cross sectional libraries used in US computer codes. Having good cross sectional data is essential to ensuring accuracy in any computer model. Regardless of the accuracy of the code and detail in the model, without good cross sections, the solution will

not be correct. Cross sections are the foundation of all computer simulations.

Within the ENDF files, cross section reactions are assigned MT numbers. The MT numbers represent the various possible reactions and are standard regardless of the particle involved in the interaction (n, p, d, t, He-3, a, or photonuclear gamma). A partial list of MT numbers is presented in Table 1.1. A complete list of MT numbers and their meanings can be found at http://t2.lanl.gov/nis/endf/mts.html.

Table 1.1: Select Examples of MT Numbers Used in Evaluated Nuclear Data Files.(LANL; 2007)

MT	Reaction	Description
1	(n,total)	Neutron total cross section. Sum of MT=2, 4, 5, 11, 16-18, 22-26, 28-37, 41-42, 44-45, and 102-117.
2	(z,z0)	Elastic scattering cross section for incident particles.
16	(z,2n)	Production of two neutrons, plus a residual.
17	(z,3n)	Production of three neutrons, plus a residual.
18	(z,fission)	Total fission. Equal to the sum of MT=19, 20, 21, and 38, if present.
102	(z,gamma)	Radiative capture.

The term "z" refers to a generic particle (n, p, d, t, gamma, alpha etc.)

The ENDF cross sections are stored as datasets within most computer codes for specific temperatures. In some instances, cross sections may not be readily available for a particular isotope or temperature that the user's problem requires. The computer code NJOY99 allows for the processing of raw ENDF data to reflect specific temperatures, isotopes and MT numbers for use in the Monte Carlo N-Particle (MCNP) code. This book will not demonstrate the uses for NJOY99-derived cross sections for using in MCNP, but will rather depend on the cross sectional temperatures associated within the standard ENDF libraries. In addition to use within a code, NJOY99 can be used to plot cross sections for various isotopes, energies and interactions. The raw data for NJOY99 is downloaded from the National Nuclear Data Center Evaluated Nuclear Data File online database or the ENDF Database from Los Alamos at http://t2.lanl.gov/nis/data.shtml. From the website, the user can select datasets from the US ENDF, the European JEFF, the Japanese JENDL, Chinese CENDL or Russian ROSFOND. Each major contributor may have slightly different measured values for the isotope and reaction. A similar code to NJOY99, AMPX, is used to produce cross section and libraries for use in other transport codes such as SCALE. The examples in SCALE in this book will specify temperature for cross section processing, but do not include specific instructions on using AMPX. Documentation on the use of AMPX is provided within the SCALE distribution package.

The probability of a neutron interaction, i.e. its cross section, is dependent on the material and the incident neutron's energy. For many materials, the cross section is also highly dependent on temperature. In particular, Doppler broadening of the resonance regions in nuclear fuel can have a significant effect on the criticality of a reactor. Either AMPX or NJOY99 can process and plot the cross sectional information at a variety of temperatures. For example the ^{235}U total neutron cross section at room temperature (300 kelvin) and at 1200 kelvin processed by NJOY99 is presented in Figures 1.9 through 1.12. For all of the possible energies, it is difficult to see a difference in the total cross sections at different temperatures in Figures 1.9 and 1.10. However if we focus on Figure 1.12, the resonance cross sections, we will see a difference. Look closely at the width of the peaks plotted at 300k versus 1200k. Notice how the peaks are sharper at the lower temperature and broader at the higher temperature. Although this may not seem important now, you will see later on that this Doppler broadening of the resonances is very important to the stability and safety of a reactor.

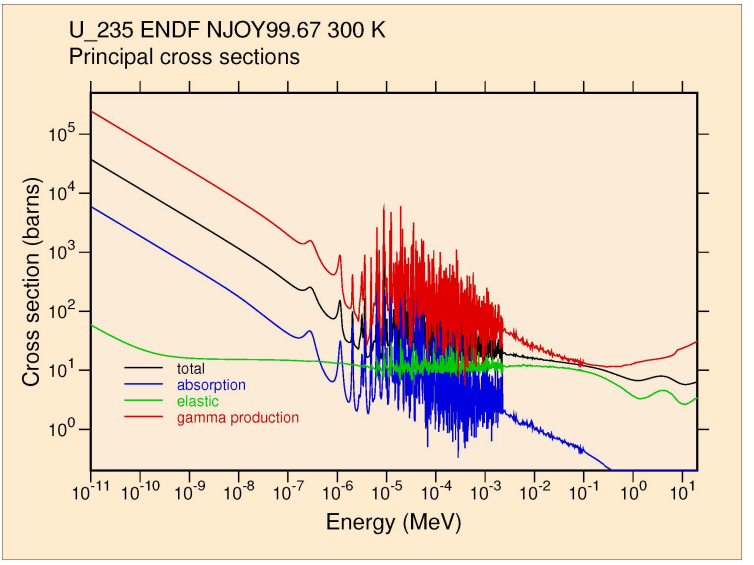


Figure 1.9: NJOY99 Cross sectional data for total fission cross section of ^{235}U at 300k

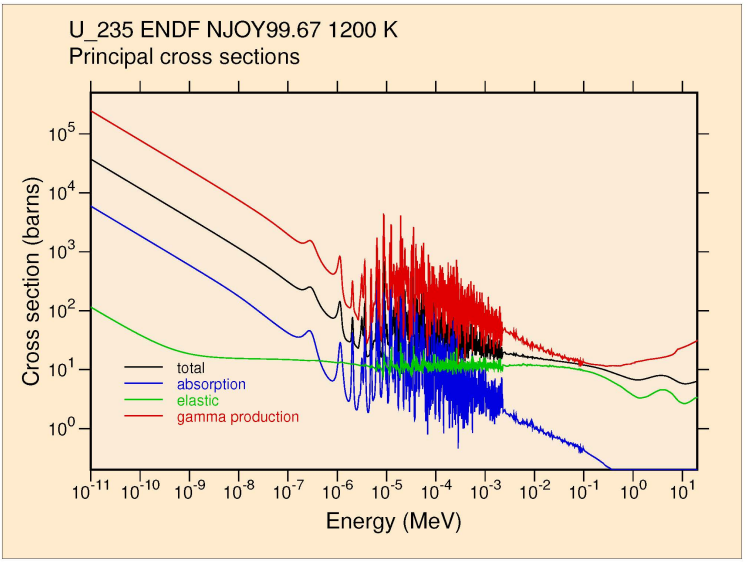


Figure 1.10: NJOY99 Cross sectional data for total fission cross section of ^{235}U at 1200k

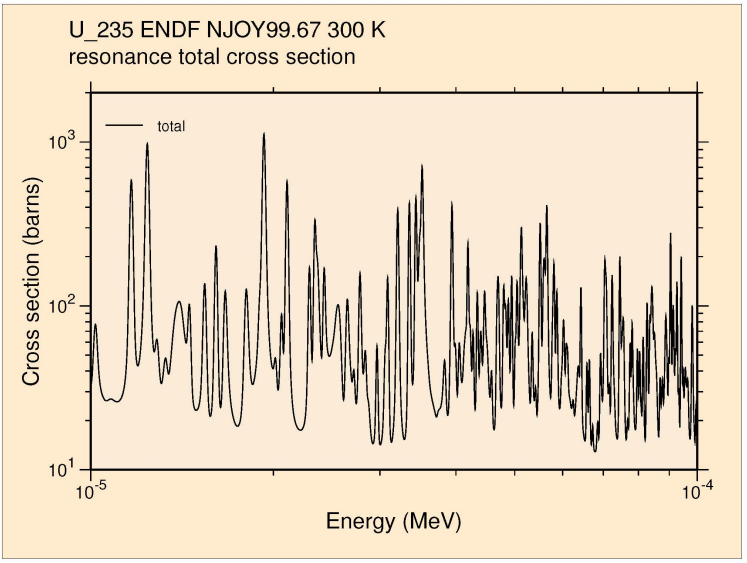


Figure 1.11: NJOY99 Cross sectional data for total fission cross section of ^{235}U at 300k within resonance region

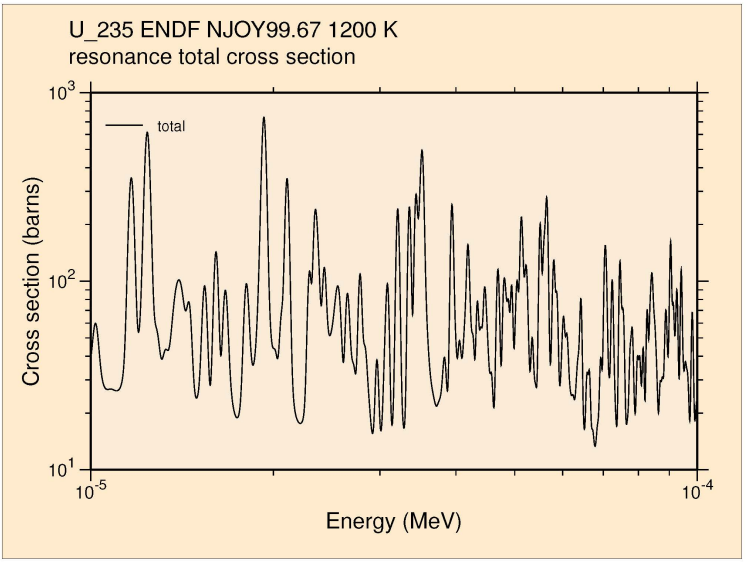


Figure 1.12: NJOY99 Cross sectional data for total fission cross section of ^{235}U at 1200k within resonance region

Bibliography

Bateman, H. (1910). *The Solution of a System of Differential Equations Occurring in the Theory of Radioactive Transformations.*, Vol. 15, V edn, Proceedings from Cambridge Philosophical Society.

Duderstadt, J. J. and Hamilton, L. J. (1976). *Nuclear Reactor Analysis*, Wiley New York.

Evans, R. D. (1955). *The Atomic Nucleus*, McGraw-Hill.

LANL, G. T.-. N. D. (2007). Nuclear information service los alamos.
URL: *http://t2.lanl.gov*

Murray, R. L. and Holbert, K. E. (2016). *Nuclear Energy an Introduction to the Concepts, Systems, and Applications of Nuclear Processes*, seventh edn, Elsevier.

NNDC (2025). US Department of Energy Chart of the Nuclides Database.
URL: *http://www.nndc.bnl.gov*

Podgoršak, E. B. (2009). Modes of Radioactive Decay, *Radiation Physics for Medical Physicists* p. 475–521.

1.7 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- Solving Bateman Equations Using Julia
- ORIGEN Module of SCALE for Calculating Radioactive Decay
- Format Graphs PSI Plot and Julia

1.8 Problems

1. Use the chart of the nuclides from the National Nuclear Database at https://www.nndc.bnl.gov to find four radioactive isotopes that primarily decay by α decay, four that β^- decay and four that β^+ decay. List each isotope and its half-life and decay constant.

2. Plot the activity for 1 gram of ^{137}Cs as a function of time for four half-lives.

3. Write the decay chain for ^{226}Ra to stable ^{206}Pb to include half-lives and reactions.

4. Set up the first order ordinary differential equations (ODEs) for the ^{226}Ra decay chain to ^{206}Pb.

5. In radioactive decay, the term transient equilibrium refers to the condition where the parent nuclide decays with a half-life on the order of the observation time and the daughter half-life is considerably shorter. Set up and solve the first order ODEs for the decay of 1.0 gram of ^{99}Mo to ^{99M}Tc for 100 days and plot the result.

6. Using Plank's constant (h), calculate the frequency and wavelength of a 250 keV photon.

7. Use the ORIGEN code in SCALE to decay of 1 gram of ^{99}Mo to ^{99M}Tc for 100 days and plot the result. After how many days does the number of atoms of ^{99}M equal the number of atoms of ^{99M}Tc?

8. Use the ENDF-VII database at the National Nuclear Database found at https://www.nndc.bnl.gov/endf to plot the microscopic elastic scattering cross section (n, el) for natural carbon (c*).

9. Given natural carbon at nominal density of 1.7 g/cm^3, calculate and plot the macroscopic elastic scattering cross section for natural carbon from 1×10^{-5} to 1.49×10^8 eV. (Hint: download the "Full evaluation in pointwise ENDF VI format" by selecting the PEN button from the results from the NNDC database search and use the first two columns to the left below the header information – energy and microscopic cross section).

10. Following the same procedure used in problem 9, calculate the macroscopic elastic scattering cross section for ordinary water at nominal density over the same energy range and compare the results to carbon.

Chapter 2

Introduction to Computation

As mentioned in the preface, nuclear engineering was at the forefront of most computational methods to solve complex problems. Many graduate courses in radiation interactions and transport require students to write computer programs to calculate Monte Carlo and/or deterministic solutions to the transport problem. In order to write computer codes, students must first develop adequate skills as programmers. This text is not intended for students who will be writing their own neutron transport computational codes. There are several references that provide the level of detail necessary to write neutron transport programs such as Lewis and Miller's book *Computational Methods of Neutron Transport* (Lewis and Miller; 1993).

Although this text is not intended to instruct students on how to write transport codes, it is still necessary to describe the basics of programming and computation to ensure that there is an understanding of the logical operation of how the programs work. As shown in Figure 2.1, there are many language options available to solve computational problems. To this end, the book provides a description of the basic methods of program design and logic. The Julia program is used to demonstrate program logic and write basic scripts. The Julia examples presented could easily be replaced by another programming language such as Matlab®, C++, FORTRAN or Python. The major advantages to using a code like Julia are threefold. First, it is completely open-source and does not require a license or purchase to use. Second, it does not require the user to compile the code. Thirdly, it lends itself to plotting the results with minimal understanding of programming. Additionally, Julia has thousands of built-in functions available to the user as packages that eliminates the need to write many functions. The major disadvantage to Julia is it is an interpreted program. This means there is a requirement to have Julia loaded on the computer so it can interpret the code and it will not operate as a stand-alone program.



Figure 2.1: There are numerous languages and programs available to the nuclear engineer to solve complex transport problems. The selection of the correct tool for the desired result is important.

To this end, it is recommended that students learn a programming language to write scripts to quickly read output files from computer codes and perform iterative adjustments to input files. Simple techniques for writing batch files and using macros in a text editor are presented to help bridge the gap between Julia and independent complied or other interpreted program languages.

2.1 Program Design

All computer programs, regardless of purpose, serve as a sequence of instructions that execute a particular function. That function may be to control a character on the screen in a video game or it could be to calculate the appropriate distance and direction a neutron will travel after an elastic scatter with a moderator. In either case, the computer performs a set of well-defined steps in a logical format to achieve a predictable result. These steps are known as an algorithm . An algorithm is a method to solve a problem. A function is a block of code in the program that performs a specific task. Programs are made of multiple algorithms that execute functions to perform achieve a desired result. To create a computer program, the programmer uses source code statements to write the functions. These statements are readable by the programmer but, except in rare instances, are not understood by the computer until the program is compiled into an executable code. The source code consists of specific instructions that execute the logical flow of arguments using commands such as WHILE, DO, PRINT, IF, THEN etc. Programming languages have syntax and commands specific to that language. Much like spoken languages, computer languages have specific vocabulary and rules. Also, as with verbal a language, a

knowledge of one code facilitates the learning of another one. The programmer can often translate their knowledge of C++ to FORTRAN, Julia or Python or some other application.(Lindeburg; 2021)

The majority of processes performed in nuclear engineering programs are mathematical calculations. Depending on the programming language, to begin the algorithm, the programmer should specify the variables and constants used in the program. The specification of variables is known as declaration . Declarations describe the variable type (INTEGER, REAL, BOOLEAN, LONG etc.) and give it an assignment which reserves computer memory space to hold the information. The declaration of variable types is required in programs like C++ and Java. In some programs, there are pre-defined constants (such as pi) that are available for use. In Julia, the user defines variables and constants that are not already in the program in the same manner as in a compiled program, but do not have to specify the type or provide an assignment. Variables in Julia are declared by writing their name. Julia does not require the user to define a datatype of the variable. Initializing variables is performed at the time of declaring variables. (GeeksforGeeks; 2023)

2.2 Logical Flowcharts

Logical flowcharts are useful to visually display how the computer steps through a particular algorithm. Table 2.1 lists some of the various symbols used to create a flowchart.

Table 2.1: List of Key Logical Flowchart Symbols and their Meanings

Symbol	Purpose	Description	Symbol	Purpose	Description
	Flow line	Indicates the flow of logic by connecting symbols		Decision	Operation with two alternatives, true and false.
	Terminal (Stop/ Start)	Represent start and end of flowchart		On-page Connector	Join different flow line
	Input/ Output	Input and output operation		Off-page Connector	Connect flowchart portion on different page
	Processing	Arithmetic operations and data-manipulations		Predefined Process/ Function	Group of statements performing one task

Figure 2.2 is a representation of a logical flowchart used for a simple workout plan. In the plan, the program first evaluates the time (morning or afternoon). Based on that result, it transitions to the day of the week and then the workout for that period. The program terminates once a workout has been assigned.

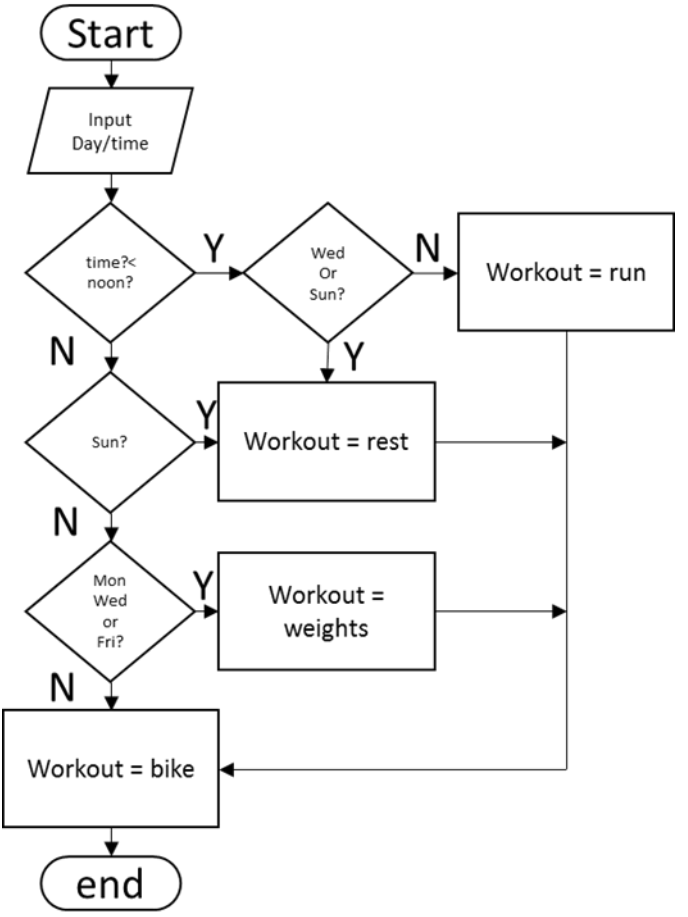
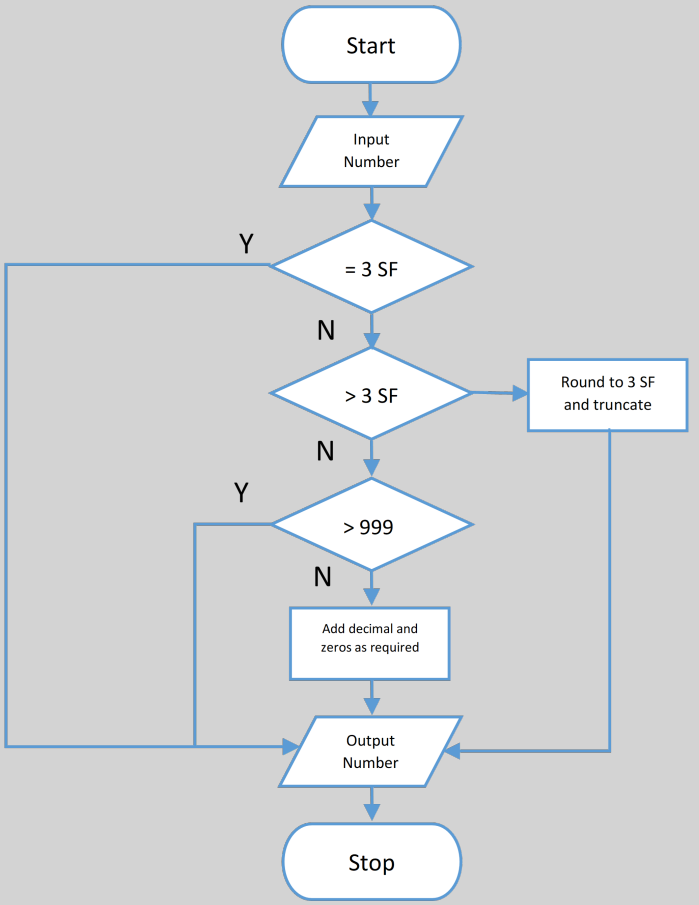


Figure 2.2: Logical flowchart for a program that assigns a workout based upon the day of the week and the time of day. (Lindeburg; 2021)

Example 2.1
Using the symbols presented in Table 2.1, write a flowchart that shows the logic of an algorithm that rounds numbers to three significant figures if possible.

Solution:



Note that if a number is > 999 and has < 3 SFs, such as 1000, it is not possible to round to 3 SFs. However, a number like 900 can have 3 SFs with the addition of a decimal (900.). Similarly, 0.0045 has only two SFs while 0.00450 has three.

2.3 Computer Language Levels

As indicated earlier, there are various types of languages used in programming. In general, there are two types of languages —low level and high level. Low level languages are machine and assembly language. Machine language is understood by the computer's central processing unit and is not normally used for coding alone. Assembly language code is translated into machine language using an assembler. Assembly language code is usually very efficient but rather complex

to understand and not normally used for large applications. Table 2.2 is a summary of high and low level codes.

Table 2.2: Summary of High and Low Level Computer Codes

Code Level	**Description**	**Examples**
Low	Codes that work directly with the machine's CPU	Numerical Machine Code, X86
High	Codes that use commands or functions and are either interpreted or compiled into machine code	Julia, Python, Java, C++, FORTRAN

High level languages are more intuitive to use because they use commands that are similar to written language. The programmer uses specific words and characters, or syntax, specific to the programming language to generate the functions. High level languages can be either compiled or interpreted. Interpreted codes, like Julia and Python, use a computer code to run the user's commands. This, interpreter, is the interface between the computer's machine language and the programming language. Interpreted codes can't run on a computer without the interpreter installed. Compiled languages, on the other hand, are programmed in the programming language and then a compiler creates an executable for the program to work on the machine in the machine language. The machine language is specific to the operating system, DOS, Linux, Apple-OS etc. Once compiled, the source code can no longer be easily viewed or modified. However, the compiled executable is now usable on any machine with the correct operating system installed. The major disadvantage to a compiled program is that it will often take more time to compile and debug than an interpreted program and every modification to the program will require it to be compiled again. It is for these reasons that many engineering students will use an interpreted program to write simple scripts to solve problems. Because of the popularity of interpreted programs, there are now some tools developed to serve as pseudo-compilers to allow portability for scripts.

2.4 Function Syntax for Operations

There are several excellent references for compiled and interpreted programs that detail all of the various rules and commands for syntax and even more online resources like https://julialang.org. Whatever the programming language, there are some essential commands necessary to complete a working script for engineering problems. The compiled or interpreted program must be able to interface with the user (input/output), it must be able to conduct basic mathematical operations, store and retrieve information (variables and constants)

and it must be capable of conducting loops and conditional statements. Within these constructs, the programmer can write functions that will solve problems, export data, evaluate and plot results.

Table 2.3: Common Julia Mathematical Operations and Functions (JuliaLang.org; 2023)

Expression	Name	Description
`+x`	unary plus	the identity operation
`-x`	unary minus	maps values to their additive inverses
`x + y`	binary plus	performs addition
`x - y`	binary minus	performs subtraction
`x * y`	times	performs multiplication
`x / y`	divide	performs division
`x ÷ y`	integer divide	x / y, truncated to an integer
`x \ y`	inverse divide	equivalent to y / x
`x ^ y`	power	raises x to the yth power
`x % y`	remainder	equivalent to rem(x,y)
`==`	equality	comparison exactly equal to
`!= , ≠`	inequality	comparison not equal to
`<`	less than	comparison less than
`<=, ≤`	less than or equal to	comparison less than or equal to
`>`	greater than	comparison greater than
`>=, ≥`	greater than or equal to	comparison greater than or equal to
`sqrt(x), √x`	square root	square root of x
`abs(x)`	absolute value	a positive value magnitude of x
`exp(x)`	exponential	natural exponential function at x
`log(x)`	natural log	natural logarithm of x
`mod(x,y)`	modulus	remainder of modulo function
`cos(x)`	cosine	cosine of x
`acos(x)`	arc cosine	inverse cosine of x
`sin(x)`	sine	sine of x
`asin(x)`	arc sine	inverse sine of x
`tan(x)`	tangent	tangent of x
`atan(x)`	arc tangent	inverse tangent of x

NOTE: (The ÷ sign can be conveniently typed by writing \div<tab>)

High-level programs, especially interpreted codes, have many built-in functions available to the user that facilitate solving problems. An excellent illustration was presented in Example 1.1 of this text. In Example 1.1, we used the **ODEProblem** solver from the **DifferentialEquations** package within Julia to solve the series of first-order linear ordinary differential equations. The **ODEProblem** is one of the 21 integrated ODE solvers in Julia's **DifferentialEquations** package that uses the Runge-Kutta integration algorithm. (Rackauckas and Nie; 2017) We could have written our own function to solve the ODE where we read the equations into a matrix and solved using standard

mathematical techniques. This would have negated the advantage of using Julia over another programming language.

Many programming languages use similar functions and syntax. Julia uses several mathematical operators and commands similar to C++ and Python. Most engineering applications do not require the use of many of the high-level features of languages that are necessary for graphics, Windows® interfacing, and similar operations. Engineering programs are often optimized for speed and flexibility and are utilitarian in design with little concern for graphics, colors, sounds etc. Table 2.3 is a consolidated list of many of the operations and functions used in Julia. (JuliaLang.org; 2023)

2.5 Programming in Julia

Programming in Julia is like programming in any computer language with the added features of having built-in functions available to assist with calculations. One of the first steps to programming is to declare variables and constants. Some constants are built within Julia such as π (pi). In Julia, it is written as either "`pi`" or the TeX completion of `\pi` followed by the tab which will appear as the Greek letter. To declare a constant, write the name for the constant and assign it a value (such as `c = 2.99792E8`). To declare a variable, give a name and use it in an expression. You can also initialize it by assigning it a number (`x=1`) which will assign an integer to the variable x. You can also use an array for a variable with multiple values. For example, (`A = Array{ Float64, 1}(undef, 3)`) is an array with three values that are all initialized to an undefined value. The value will change to when numbers are read into the array and they will take on a Float64 format. Information on integers and floating-point numbers can be found at https://docs.julialang.org/en/v1/manual/integers-and-floating-point-numbers.

A key to programming a working algorithm is the use of loops and conditional statements. Loops are operations that operate over a set number of iterations or execute if a condition is either met or not met. In Julia, there are two primary loop operations – "while" and "for".

The **while** command repeats the series of operations until a termination condition is met. For example:

```
Energy=0.001; #initialize the value for variable x
m= 1.6750E-27; #constant mass of a neutron in kg
eV= 1.60218E-19; #constant eV=1.60218E-19 Joules
while Energy < 0.1
        v=sqrt((2*Energy*eV)/m) #energy to velocity
        sigma_slope=1/v #assign value to sigma
global Energy=Energy+0.001 #iterate energy by 0.001eV
end
```

This example will start with a value of 0.001 eV and then converts that value to a velocity using the mechanics equation for kinetic energy; $E = \frac{1}{2}mv^2$. It then uses the velocity to calculate the slope of the microscopic cross section σ. After it passes the value of the slope to the variable sigma_slope, the energy value is increased by 0.001 eV. The loop will continue to calculate and pass values for the slope until E = 0.1 eV when it will terminate.

The **for** loop repeats a series of operations between the command for and the termination command end. For example:

```
for ii = 1:4 #index ii in range object from 1 to 4
        mu = cos(ii*pi/2)
        println(mu)
end
```

```
6.123233995736766e-17
-1.0
-1.8369701987210297e-16
1.0
```

This loop iterates the cosine of the angle (μ) from angles of $\frac{\pi}{2}$ to 2π radians and prints the results. It will terminate after the four values are calculated.

Conditional statements allow for the computer to decide an action based on a true or false relationship. The format for the conditional construct uses **if**, **elseif** and **else**. As with the loops, the conditional statements terminate with an end. The conditional allows for a comparison and multiple outcomes based upon an initial value. For example:

```
if Energy <= 1E-7
        Sigma_Region = 'thermal'
elseif 1E-7<E<=1E-2
        Sigma_Region = 'resonance'
else
        Sigma_Region = 'fast'
end
```

This example compares the given energy in MeV to three different ranges – less than or equal to 1×10^{-7}; greater than 1×10^{-7} and less than 1×10^{-2} and any other value. The third condition could have also used an elseif with a condition of `E > 1E-2`.

A program will often nest loops and conditional statements within each other. For instance, a program may have a conditional that, when true, causes a looped calculation to be performed.

A general programming procedure for setting up and developing a computer program follows:

1. *Map out the algorithm steps using a logical flowchart.* Refer to the symbology in Table 2.1 and create a visual map of the logic flow for the program.

2. *Write a meaningful header with appropriate comments.* Comments in Julia are preceded with a # symbol. The Julia editor will make the text following the # `teal blue` and in italics indicating that it won't be executed. In-line comments are also inserted using the same syntax. Detailed comments to start the program and throughout the body of the program are essential for good programming. Comments not only allow others to understand the steps used in the program, but also reminds the programmer in the future what he or she wrote and why. You can use a #= to start a block of comments and close with a =# and everything in between over multiple lines will not be executed. This is especially useful if you want to make a header or need to comment out many lines of code.

3. *Declare constants and variables.* If the programming language requires, as in C++, initialize all variable types.

4. *Write a routine to receive input.* This can be reading in values from a file or requesting input from a user. Input is then stored for use.

5. *Write routines and subroutines to execute desired calculations.* Depending on the program, it may be beneficial to write subroutines for calculations that are performed multiple times. A subroutines is just a short program that the main program calls. It can greatly shorten and increase the efficiency of the program. For example, a program might perform a statistical check on the results of several different calculations. In this case, it makes sense to write a subroutine that performs the statistical check and calls it each time it is needed versus writing a separate statistical check after each calculation. Subroutines are also useful because they can be easily transferred from one program to another.

6. *Write routine that handles output.* This may be writing the output to a file, printing it to a screen, creating an array that can be used in another program, or plotting the results. Within Julia, the manipulation and plotting of results can be either written directly into a script or written separately in the command line.

These steps are followed in Example 2.2 which outlines a Julia program for plotting the Klein-Nishina differential gamma ray scattering cross section.

Example 2.2
The differential scattering cross section for Compton scattered photons depends upon the initial energy of the incident photon and the angle of the scattered

photon. The cross section is given by the Klein-Nishina formula (Weinberg; 2011):

For an incident photon of energy E_γ,the differential cross section is:

$$\frac{d\sigma}{d\Omega} = \alpha^2 {r_c}^2 P(E_\gamma, \theta)^2 + [P(E_\gamma, \theta) + P(E_\gamma, \theta)^{-1} - 1 + \cos^2(\theta)]/2$$

Where $\frac{d\sigma}{d\Omega}$ is the differential cross section with respect to solid angle, α is the fine structure constant ($\approx 1/137.04$), θ is the scattering angle, r_c is the reduced Compton wavelength of the electron (0.38616 pico-meters), and $P(E_\gamma, \theta)$ is the ratio of the photon energy after and before the collision given by:

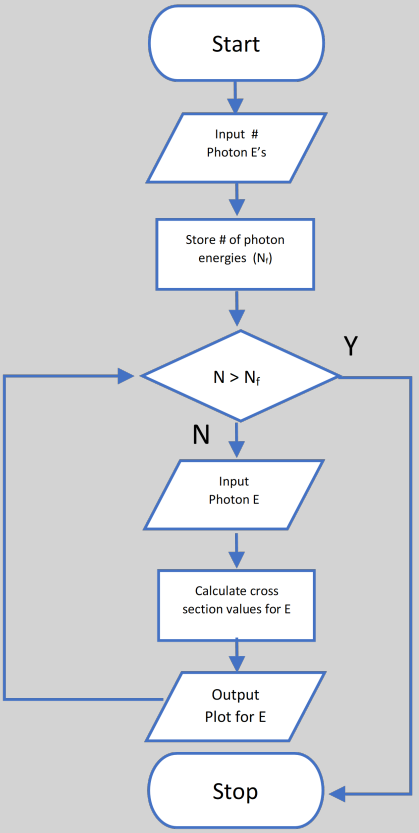
$$P(E_\gamma, \theta) = \frac{1}{1 + \left(\frac{E_\gamma}{m_e c^2}\right)(1 - \cos\theta)}$$

Where $m_e c^2$ is the rest mass energy of an electron (511 keV).

Write a Julia program that plots the differential scattering cross section for all values of scattered angle (θ= 0 to 2π) for up to 5 user-supplied incident photon energies (E_γ). Compare the results to other published examples. Based on the plot, what conclusions can be drawn about the angular probability of a Compton scattered photon as a function of energy?

Solution: For this program example, we will follow the steps for programming outlined earlier.

1. *Map out the algorithm steps using a logical flowchart.*



2. *Write a meaningful header with appropriate comments.* Open Visual Studio Code and create a new file named "Klien Nishina Plot Program.jl" and start with comments and include the packages necessary for program.

Klien Nishina Plot Program.jl > ...

```
#=Differential Klien Nishina Plot Program
|-------------------------------------------------------------------|
|Function to plot the Klein-Nishina differential cross section for |
|for Compton Scattered photon per steradian for up to 5 different  |
|user-supplied energies in keV                                      |
|Common energies are 2.75E-3, 60, 511, 1460, & 10000 keV           |
|the user can choose any energy                                     |
|Written by: Kenneth Allen 5 December 2022                          |
|-------------------------------------------------------------------|=#
using IJulia
using Plots
```

3. *Declare constants and variables.* Note use of comments.

```
#-------------SET CONSTANTS USED IN PROGRAM-------------------------
#Reduced Compton wavelength r(c)/hm(e)c
r = 0.38616E-12 # 0.3861 pico meters
#Fine structure constant (~1/137.04)
α = (1/137.04) #Using IJulia, can get Greek by \<lettername> + tab
#Electron rest mass energy
mc² = 511 # 511 keV
#Angle theta from zero to 2π  in 500 increments for plotting
θ = range(0, stop=2π, length=500)
#Create a counter and intialize to zero
k = 1;
E = Float64[]  #create an empty vector to pass energies into
```

4. *Write a routine to receive input.* Because this is a simple script, we can request the user supply several energies (up to 5 total) to plot and read those values one by one into the plotting routine. A better approach would have been to enter the values into an array and use that array as the input for the plotting function. This program does not do that to demonstrate the conditional and loop operations. Also, for simplicity, this program does not perform an error check on the user's input. For example, the user does not receive a message that an invalid entry is given if they provide values for the number of energies that are less than 1 or greater than 5. A simple conditional statement can be added to include an error check.

```
#---------------REQUEST USER INPUT--------------------------------
#Request number of energies to plot
print("Enter number of energies to plot (up to 5):\n\n")
# Calling rdeadline() function
nₑ  = readline()
nₑ  = parse(Int64, nₑ )

while k <= nₑ

    print("Enter energy of the incident photon in kev:\n\n")
    Evalue = readline()
    Evalue = parse(Float64, Evalue)
    # Pass values into vector
    push!(E, Evalue)
    global k += 1

end
```

5. *Write routines and subroutines to execute desired calculations.* Referring to the flowchart, this section begins with a loop. The loop will first determine how many iterations of the calculation to perform. This example uses a **while** loop that repeats the calculation for each user-supplied energy up to the number entered. For each step, the program will prompt for an energy and then generate a plot. The plotting routine uses a conditional **if**, **elseif**, and **else** routine to determine how many energies to plot based on the user's input.

```
#--------PROCESS DATA AND PLOT RESULTS----------------------------
# Set up conditional to handle 5 different cases
if  nₑ == 1
#  The differential Klien Nishina scattering cross section equation
    ρ(θ,E) = (α^2)*(r^2)*((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^2)*(((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))+
    ((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^-1.0)-1.0+(cos(θ))^2)/2)
    #  plot RESULTS
    plot([(θ, ρ.(θ,E[1]))], proj=:polar )
elseif  nₑ == 2
    #  The differential Klien Nishina scattering cross section equation
    ρ(θ,E) = (α^2)*(r^2)*((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^2)*(((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))+
    ((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^-1.0)-1.0+(cos(θ))^2)/2)
    #  plot RESULTS
    plot([(θ, ρ.(θ,E[1])),(θ, ρ.(θ,E[2]))], proj=:polar )
elseif  nₑ == 3
    #  The differential Klien Nishina scattering cross section equation
    ρ(θ,E) = (α^2)*(r^2)*((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^2)*(((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))+
    ((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^-1.0)-1.0+(cos(θ))^2)/2)
    #  plot RESULTS
    plot([(θ, ρ.(θ,E[1])),(θ, ρ.(θ,E[2])),(θ, ρ.(θ,E[3]))], proj=:polar )
elseif  nₑ == 4
    #  The differential Klien Nishina scattering cross section equation
    ρ(θ,E) = (α^2)*(r^2)*((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^2)*(((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))+
    ((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^-1.0)-1.0+(cos(θ))^2)/2)
    #  plot RESULTS
    plot([(θ, ρ.(θ,E[1])),(θ, ρ.(θ,E[2])),(θ, ρ.(θ,E[3])),(θ, ρ.(θ,E[4]))], proj=:polar )
elseif  nₑ == 5
    #  The differential Klien Nishina scattering cross section equation
    ρ(θ,E) = (α^2)*(r^2)*((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^2)*(((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))+
    ((1.0/(1.0+(E/mc²)*(1.0-cos(θ))))^-1.0)-1.0+(cos(θ))^2)/2)
    #  plot RESULTS
    plot([(θ, ρ.(θ,E[1])),(θ, ρ.(θ,E[2])),(θ, ρ.(θ,E[3])),(θ, ρ.(θ,E[4])),(θ, ρ.(θ,E[5]))], proj=:polar )
else
    print("Too many energies selected")
end
```

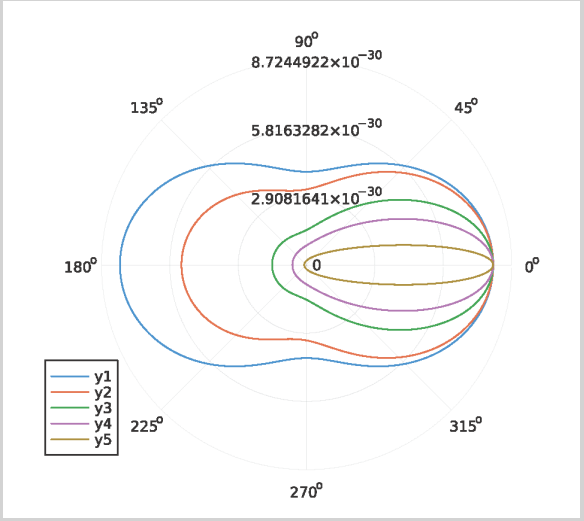
This conditional is added for demonstration purposes and would not be necessary if an array was used to store the results from the differential cross section calculations and then the plot was called one time. In that case, Julia would assign a different color to each data series in the plot.

6. *Write routine that handles output.* In this example, there is no requirement to make a separate routine for the output. The program plots the results as soon as they are calculated and terminates when all of the user-supplied values are plotted.

Now that the entire program, Klien Nishina Plot Program.jl, is written, it can be executed. To run the program within Visual Studio Code, use ALT+J+O to bring up the REPL terminal within Visual Studio Code. At the Julia> prompt enter **include("Klien Nishina Plot Program.jl")** which will start the program. HINT: when writing the filename in the command line, start with **include("K** and then press the tab key. Julia will recognize the file you intend to access and will auto-complete the line. When the file opens answer the prompts:

```
Enter energy of the incident photon in kev:

60
Enter energy of the incident photon in kev:

511
Enter energy of the incident photon in kev:

1460
Enter energy of the incident photon in kev:

10000
```

The plot will open in another window within VSC as:



Now the results of the program plot can be analyzed. Several references have similar plots for comparison. When compared to Figure 2.19 in John Knoll's Radiation Detection and Measurement, the Julia plot matches the published result. (Knoll; 2020)

So, what does the plot tell us about the Compton cross section? By looking at the plot, we can see that at low energies, the Compton scattered photon is just as likely to scatter backwards as it is forwards. However, as the incident photon energy increases, the cross section becomes more forward-peaked meaning backscatter is less probable. This is important when we perform shielding and detection calculations.

2.6 DOS Commands and Batch Files

Good pre- and post-processing greatly increases the engineer's efficiency when working with computational programs. This includes programs written by the engineer or other large-scale programs used to calculate results. Most undergraduate nuclear engineering students have little to no experience using computer programs that have a command prompt interface such as the DOS prompt or Linux command prompt. Many nuclear engineering computer programs, like MCNP, are still operated through a DOS or Linux prompt. This text will work exclusively with the DOS command prompt. For Linux operations, refer to an appropriate reference.

Being familiar with basic DOS commands will greatly increase the productivity of the engineer and allow him or her to navigate the operation of a program with ease. In addition to running a program that uses the command prompt, DOS commands can be written into a batch file. Batch files can greatly increase productivity by executing multiple input files in sequence or performing repeated functions. Batch files allow the engineer to perform many iterations of a calculation without having to wait for the program to complete the operation and then start the next file and repeating until all of the files are complete. Advanced users can write scripts into an executable using Julia, Python, C++ or FORTRAN to do even more. For example, a script can read in an input file, change a parameter, execute the program, read through the results, tabulate items of interest and repeat. The NJOYHelper program, used assist in generating the cross section plots presented in Chapter 1, was written to read in temperatures and isotopes to generate NJOY input files and a batch file that executes NJOY as many times as necessary to generate the required cross sections. Table 2.4 contains many common DOS commands and their meanings for reference. (Hope; 2023)

There are many more DOS commands that have been used throughout the years. Some commands are no longer available depending on the version of Windows being used. CAUTION, some DOS commands can delete, format, or lock a hard drive and these operations may not be repairable and DOS may not prompt the user that there is potential damage prior to executing them.

In addition to the typed commands, there are a number of shortcut keys that are available to the user to navigate the DOS command prompt faster. Because DOS does not use a graphical user interface (i.e. a mouse and pointer), knowing the shortcut keys can greatly help the user efficiently move through the screen prompts and commands. Table 2.5 lists many of the command prompt shortcut keys and their actions. (GeeksforGeeks; 2023)

Table 2.4: Several Useful DOS Commands and their Meanings

Command	**Description**
assoc	View the file associations
call	Calls a batch file from another batch file
cd	Change directories
comp	Compares files
copy	Copy one or more files to an alternate location
del	Deletes one or more files
dir	List the contents of one or more directory
echo	Display messages and enables and disables echo
fc	Compare files
find	Search for text within a file
for	Boolean used in batch files
if	Allows for batch files to perform conditional processing
mkdir	Command to create a new directory
more	Display one page at a time
move	Move one or more files from one directory to another directory
print	Prints data to a printer port
prompt	View and change the MS-DOS prompt
rename	Renames a file or directory
rmdir	Removes an empty directory

Table 2.5: Useful Command Prompt Shortcuts

Shortcut	**Meaning**
Shift+Home	Extend current selection to the beginning of a command. Press Shift+Home again to include the path (e.g., C:\Windows\system32) in the selection.
Shift+End	Extend current selection to the end of the current line.
Ctrl+Shift+Home/End	Extent current selection to the beginning or end of the screen buffer (respectively).
Shift+Page Up/Page Down	Extend current selection by one page up or down.
Ctrl+C (or Ctrl+Insert)	Copy currently selected text.
F2 and then a letter	Copy text to the right of the insertion point up to the letter you typed.

Table 2.5: Useful Command Prompt Shortcuts

Shortcut	**Meaning**
Ctrl+V (or Shift+Insert)	Paste text from the clipboard.
Backspace	Delete the character to the left of the insertion point.
Ctrl+Backspace	Delete the word to the left of the insertion point.
Tab	Autocomplete a folder name.
Escape	Delete the current line of text.
Insert	Toggle insertion mode. When insertion mode is on, anything you type is inserted at your current location. When it's off, anything you type overwrites what's already there.
Ctrl+Home/End	Delete text from the insertion point to the beginning or end of the current line.
Ctrl+Z	Marks the end of a line. Text you type after that point on that line will be ignored.
F3	Repeat the previous command.
Up/Down Arrow	Scroll backwards and forwards through previous commands you've typed in the current session. You can also press F5 instead of the Up Arrow to scroll backwards through the command history.
Right Arrow (or F1)	Recreate the previous command character by character.
F7	Show a history of previous commands. You can use the Up/Down arrow keys to select any command and then hit Enter to execute the command.
Alt+F7	Clear the command history.
F8	Move backward in the command history to commands matching the current command. This is useful if you want to type part of a command you've used several times and then scroll back in your history to find the exact command you want to repeat.

Table 2.5: Useful Command Prompt Shortcuts

Shortcut	Meaning
Ctrl+C	When working with the command history, abort the current line you're typing or a command that is currently executing.

Batch files can be written using a text editor, such as TexPad®, and saved as *.bat file. To execute, type the filename at the command prompt. Example 2.3 demonstrates how to write a batch file that will clean up the MCNP Working folder by deleting the files written to the directory where the input file is located after MCNP is used without deleting the input and output files.

During an MCNP run, the program creates **runtapes**, **sourcetapes** and other files that store some of the computational information which can be used later if needed. The naming convention in MCNP names the first **runtape**, *"runtpe"*. The next run of MCNP from that same working directory will write the **runtape** to a file named *"runtpf"* and subsequent runs will alphabetically iterate the last letter of the file name through z and back to *"runtpd"*. If the user runs files in the same working directory without deleting the **runtape** files, the program will exhaust names for the **runtape** (26 runs). This will cause a **Bad Trouble** error in MCNP that is troublesome because no MCNP files will run as there is no place to store the **runtape**. The error message on the console will not tell the user the cause and inexperienced users may believe their MCNP program has just stopped working.

The Cleanup.bat batch file deletes unwanted files and keeps the files the user needs by identifying any files with the file format for the **runtape** and **sourcetape** files along with files created by Visual Editor during plotting that are no longer needed. The batch file then deletes the files. CAUTION – all the files identified in the batch file within the working directory will be lost when this batch file is executed. An alternate approach would be to write a batch file that deleted specific files within the folder. This can be accomplished by naming the files explicitly or by using the * before the extension to delete all files of that type. For example, @ del *.txt will delete all text files in that directory.

Example 2.3
The MCNP and Visual Editor program write numerous files to the working directory that routinely need to be deleted. These include files in the format

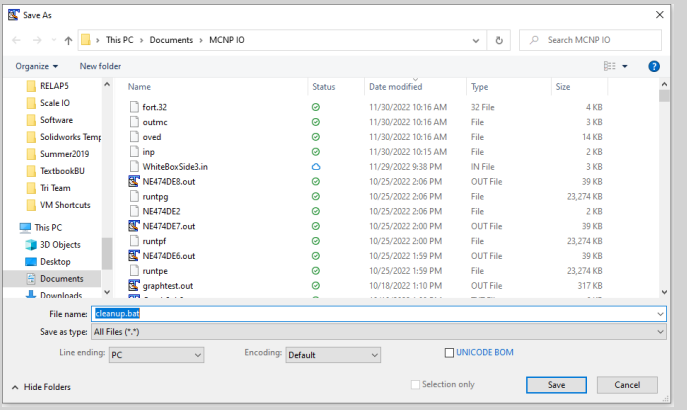
out*, runtp*, srct*, and comou* where the * indicates the last letter of the filename that is iterated each time the program runs. After 26 runs (i.e. when the alphabet is exhausted) the program will stop working. Write a batch file named cleanup.bat that will delete out all of the files of the type specified and leave other files unchanged within a directory.

Solution: Use a text editor, like Textpad® to write the following commands:

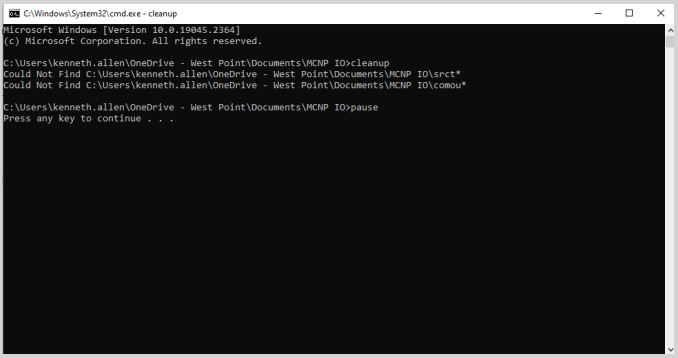
cleanup.bat

```
@ rem ----- MCNP bin cleanup batch file
@ del out*
@ del runtp*
@ del srct*
@ del comou*
pause
```

Then save the file as Cleanup.bat. You can do that by selecting "Save as" and then using the All Files (*.*) option.



Once the batch file is created. You can execute it either by going to the file manager and double-clicking the cleanup.bat filename or going to the command prompt (click start button and type "CMD" enter) and type the following:



Note that the command prompt will indicate if files of that type are not present and ask for any key to close.

2.7 Macros In Textpad®

The Textpad® text editor is one of many programs that allow for the easy manipulation of text files. It is essential to use a text editor when working with codes versus a word processor, like Microsoft Word® . This is because word processors will embed coding characters within the text that are not visible to the user but will cause issues when the computer attempts to execute them. Within Textpad®, there are numerous features that are not available from simple text editors like NotePad®. One of the most useful features allows the user to highlight a vertical block of text and copy it. This allows the text to be pasted into a spreadsheet for data manipulation. Block select mode is an option under the "Configure" tab in Textpad®. Another useful feature in Textpad® is the macro function . The macro function allows the user to record his or her keystrokes and save them to a macro filename. Then the user can repeat those keystrokes later by selecting the macro tab and clicking on the saved macro filename. Example 2.4 demonstrates how to create a macro that turns rows of numbers in an output file to columns so they can be easily imported into a spreadsheet. This is another option as opposed to using the copy "transpose" function in Microsoft Excel® .

Example 2.4
The ORIGEN output in Example 1.2 displayed the concentrations of isotopes in the following table:

```
=   Nuclide concentrations in grams for case '1' (#1/1)                                                                =
-------------------------------------------------------------------------------------------------------------------------
  (relative cutoff; integral of concentrations over time >   1.00E-06 % of integral of all concentrations over time)
.
                    0.000d      1.000d      7.000d     31.000d    180.000d    365.000d   1095.000d   3285.000d   7300.000d
  he-4         0.0000E+00  1.8704E-02  1.8704E-02  1.8707E-02  1.8798E-02  1.9010E-02  2.0073E-02  2.3029E-02  2.7203E-02
  pb-206       0.0000E+00  9.1773E-09  2.5772E-06  1.2135E-04  4.8087E-03  1.5761E-02  7.0453E-02  2.2258E-01  4.3733E-01
  pb-210       0.0000E+00  9.8117E-01  9.8067E-01  9.7866E-01  9.6627E-01  9.5111E-01  8.9357E-01  7.4101E-01  5.2573E-01
  bi-210       0.0000E+00  7.8338E-05  3.7604E-04  5.9696E-04  5.9764E-04  5.8827E-04  5.5268E-04  4.5832E-04  3.2517E-04
  po-210       0.0000E+00  5.5325E-06  2.0834E-04  1.8762E-03  9.4834E-03  1.3489E-02  1.5307E-02  1.2654E-02  8.9777E-03
  po-214       1.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
------------
  totals       1.0000E+00  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9973E-01  9.9957E-01
=========================================================================================================================
```

Write a macro in Textpad® to convert the data from rows to columns.

Solution:

Open up Textpad® and copy the table from the ORIGEN output to a new document. Place the cursor on the row beneath the last row on the table. This will indicate the starting position for the macro. When you record the macro, each movement and keystroke will be recorded until you stop.

```
=   Nuclide concentrations in grams for case '1' (#1/1)                                                                =
-------------------------------------------------------------------------------------------------------------------------
  (relative cutoff; integral of concentrations over time >   1.00E-06 % of integral of all concentrations over time)
.
                    0.000d      1.000d      7.000d     31.000d    180.000d    365.000d   1095.000d   3285.000d   7300.000d
  he-4         0.0000E+00  1.8704E-02  1.8704E-02  1.8707E-02  1.8798E-02  1.9010E-02  2.0073E-02  2.3029E-02  2.7203E-02
  pb-206       0.0000E+00  9.1773E-09  2.5772E-06  1.2135E-04  4.8087E-03  1.5761E-02  7.0453E-02  2.2258E-01  4.3733E-01
  pb-210       0.0000E+00  9.8117E-01  9.8067E-01  9.7866E-01  9.6627E-01  9.5111E-01  8.9357E-01  7.4101E-01  5.2573E-01
  bi-210       0.0000E+00  7.8338E-05  3.7604E-04  5.9696E-04  5.9764E-04  5.8827E-04  5.5268E-04  4.5832E-04  3.2517E-04
  po-210       0.0000E+00  5.5325E-06  2.0834E-04  1.8762E-03  9.4834E-03  1.3489E-02  1.5307E-02  1.2654E-02  8.9777E-03
  po-214       1.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
------------
  totals       1.0000E+00  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9973E-01  9.9957E-01
=========================================================================================================================
```

From the "macro" tab, select "start recording". Insert a return moving the cursor down one line. Use the arrow keys to move the cursor to the beginning of the first number in the first column. Press and hold the shift key and the right arrow until the first number and the two spaces after it are highlighted. Use the Ctrl –X function to cut the number and the two spaces. Use the arrows to move the cursor back to the newly created line at the bottom of the table and use the Ctrl-V function to paste the value. Now use the arrows to move up to the 2nd number in column one and cut it and paste it next to the first number. Continue cutting and pasting the first column into a row until it looks like the following:

```
=   Nuclide concentrations in grams for case '1' (#1/1)                                                                =
-------------------------------------------------------------------------------------------------------------------------
  (relative cutoff; integral of concentrations over time >   1.00E-06 % of integral of all concentrations over time)
.
                    0.000d      1.000d      7.000d     31.000d    180.000d    365.000d   1095.000d   3285.000d   7300.000d
  he-4         1.8704E-02  1.8704E-02  1.8707E-02  1.8798E-02  1.9010E-02  2.0073E-02  2.3029E-02  2.7203E-02
  pb-206       9.1773E-09  2.5772E-06  1.2135E-04  4.8087E-03  1.5761E-02  7.0453E-02  2.2258E-01  4.3733E-01
  pb-210       9.8117E-01  9.8067E-01  9.7866E-01  9.6627E-01  9.5111E-01  8.9357E-01  7.4101E-01  5.2573E-01
  bi-210       7.8338E-05  3.7604E-04  5.9696E-04  5.9764E-04  5.8827E-04  5.5268E-04  4.5832E-04  3.2517E-04
  po-210       5.5325E-06  2.0834E-04  1.8762E-03  9.4834E-03  1.3489E-02  1.5307E-02  1.2654E-02  8.9777E-03
  po-214       0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
------------
  totals       1.0000E+00  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9996E-01  9.9973E-01  9.9957E-01
=========================================================================================================================

0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 1.0000E+00
```

Now move the cursor to the starting point and stop recording. Textpad® will then prompt for a name and description. You can then save the macro.

Once complete, you can select your macro name under the macro tab, and it will repeat the previous routine by entering a new line and then taking the values from the 2nd column and creating a new row. Once complete, the table can be selected using the block select mode and pasted into a spreadsheet for plotting.

Bibliography

GeeksforGeeks (2023). Variables in julia.
URL: *https://www.geeksforgeeks.org/variables-in-julia*

Hope, C. (2023). Ms-dos and command line overview.
URL: *http://www.computerhope.com/overview.htm*

JuliaLang.org (2023). Julia language documentation.
URL: *https://docs.julialang.org*

Knoll, G. F. (2020). *Radiation Detection and Measurement*, John Wiley.

Lewis, E. E. and Miller, W. F. (1993). *Computational Methods of Neutron Transport*, American Nuclear Society.

Lindeburg, M. R. (2021). *FE Other Disciplines Review Manual: Rapid Preparation for the Other Disciplines Fundamentals of Engineering Exam*, PPI.

Rackauckas, C. and Nie, Q. (2017). Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia, *Journal of Open Research Software* **5**(1).

Weinberg, S. (2011). *The quantum theory of Fields*, Cambridge Univ. Press.

2.8 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- Number Density in Julia and C++
- Using Julia to Read in a Data File

2.9 Problems

1. Use Julia in a Jupyter Notebook to calculate the velocity of a 2 MeV α particle, a 2 MeV neutron, and a 2 MeV β particle.

2. Total mean free path (λ_{total}) of a neutron is the inverse of its macroscopic cross section (λ_{total} =1 Σ_{total}) and has units of cm. Draw a logical flowchart for an algorithm that calculates the mean free path for 0.0253 eV neutrons traveling in room temperature through H_2O, D_2O, and graphite. Assume nominal density for each material.

3. Write a program in Julia that performs the algorithm designed in problem 2.

4. Basic shielding of neutrons through materials is given by the exponential equation $N = N_0 e^{-\lambda t}$ where N represents the number of neutrons, λ is the mean free path of neutrons in the material and t is the thickness of the material. Draw a logical flowchart for an algorithm that calculates the thickness required to reduce the initial number of neutrons by one-half for 0.0253 eV neutrons traveling in room temperature through H_2O, D_2O, and graphite. Assume nominal density for each material.

5. Write a program in Julia that performs the algorithm designed in problem 4.

6. Using the half-thicknesses calculated in problem 5, calculate the total thickness of H_2O, D_2O, and graphite, in cm, required to shield 0.0253 eV neutrons to 1% of their initial number.

7. Write a batch file that renames files with a .txt file extension to a file with no file extension. For example rename the file input.txt to just input.

8. Rewrite the Julia program for plotting the Klein-Nishina differential cross section presented in Example 2.2 using a while loop in place of a for loop.

9. Radioisotopes ingested in the body are subject to both radioactive decay and natural removal from the body. The natural remove process has its own half-life known as the biological half-life. The effective half-life accounts for both the radiological decay process and the biological half-life and is given by the equation $\frac{1}{t_{1/2e}} = \frac{1}{t_{1/2}} + \frac{1}{t_{1/2b}}$. Write a program that calculates the effective half-life for a radioisotope given the radiological and biological half-lives.

10. Radioactive iodine, ^{131}I has a radioactive half-life of around 8.0 days and a biological half-life of 5.5 days in the thyroid. Using the formula in Problem 9 for effective half-life, write a program that plots the decay rate for 1 gram of ^{131}I using the radiological, biological and effective half-lives for a period of 30 days and compare the results.

Chapter 3

Neutron Transport & Deterministic Codes

Neutron transport is one of the most important and challenging topics in nuclear engineering. Transport theory has been studied and refined since the earliest days of reactor design. Neutron transport theory was founded on gas transport theory proposed by Ludwig Boltzmann in 1872. The complexity of the problem was the catalysis for the development of computer codes and faster and more powerful computers to handle those codes. With each evolution of hardware capability, engineers have been able to further refine the solutions to the transport equation and eliminate some of the approximations that were necessary in early models. As an example, computer codes were able to go from rough cross section groups with averages over large energy ranges to finer and finer group structures to a near continuous-energy spectrum. The refinement of computer models has allowed for more efficient designs with greater safety and more realistic results. As processor speed and memory has increased, higher level codes now dominate the landscape for solutions that were once based entirely on diffusion theory.

This text is not intended to provide a detailed derivation of the neutron transport equation. The derivation of the transport equation and the development of the many approximations used in deterministic transport codes can be found in several texts to include *Computational Methods of Neutron Transport* by Lewis and Miller. (Lewis and Miller; 1993)

The intent of this text is to provide an introduction to neutron transport and some of the major types of solutions used in computer simulations, so the student is aware of some of the capabilities and limitations of various deterministic codes. This foundation will allow for an introduction to Monte Carlo methods of neutron transport and a comparison of results of problems solved in both methods.

3.1 Transport as it is Applied to Neutrons

The major challenge with neutron transport is to develop a single equation that describes all of the possible interactions a neutron can have over all variables of energy, space, position, and time. As an analogy we can look at traffic on a highway as points in our neutron system. At any given time, we have cars traveling at different speeds and directions. At a finite point in time we can examine cars traveling at a particular speed and direction and count them as a group. If we total all of the groups over a given time period, we will have a complete picture of all of the vehicle traffic on the highway. Figure 3.1 explains the balance of cars in a segment of highway.

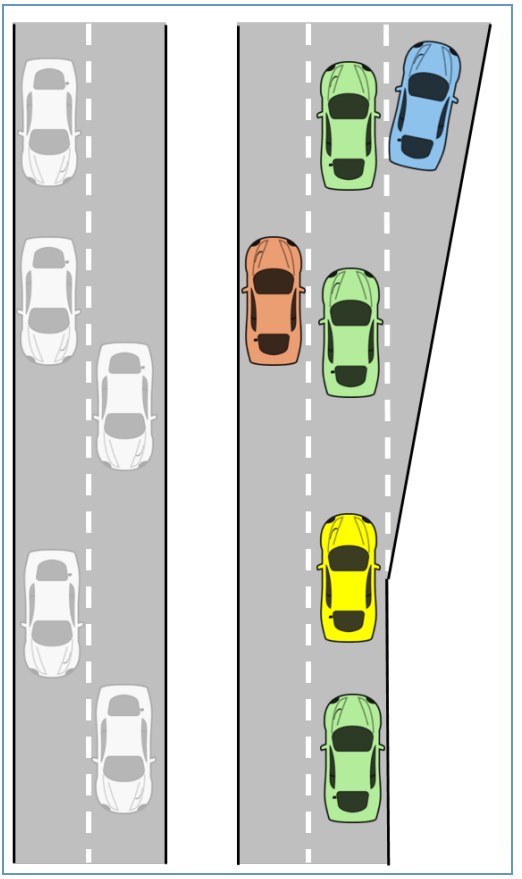


Figure 3.1: In this section of highway, we are evaluating only cars headed north at speed x, represented by the green cars. The southbound, white, cars are not counted. Additionally, the red car is traveling at a speed $>$ x, the yellow car is at a speed $<$ x, and the blue car is at a different angle so none of them are counted in the system. If the red car was to slow down to x or another car was to turn into the highway at speed x, they could join the group. If any of the cars were involved in a collision, they would be removed from the system. The total traffic over time would be a balance all of cars entering and leaving the highway at all speeds and traveling in all directions.

The equation for neutron transport is more difficult to develop than our simple car analogy. This is because neutrons interact differently at different energies, they lose energy and change direction in scatter, can be absorbed in materials, can leak from the system, and are produced in fission or from another source. The complexity of the problem is readily observed when we look at the

neutron transport equation, Equation 3.1. (Duderstadt and Hamilton; 1976)

$$\frac{\partial n}{\partial t} + v\hat{\mathbf{\Omega}} \cdot \nabla n + v\Sigma_t n(\mathbf{r}, E, \hat{\mathbf{\Omega}}, t) = \int_{4\pi} d\hat{\mathbf{\Omega}}' \int_0^\infty dE' v' \Sigma_s(E' \to E, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}) n(\mathbf{r}, E', \hat{\mathbf{\Omega}}', t) + s(\mathbf{r}, E, \hat{\mathbf{\Omega}}, t) \quad (3.1)$$

This equation represents the balance of neutrons within a volume. With respect to neutron population, we are referring to a population of neutrons of the same energy, E, traveling in a particular solid angle, $\hat{\mathbf{\Omega}}$. In this format, bold variables indicate vector quantities. The total population of neutrons (gains and losses) in the system must be equal.

We can get a better understanding of the equation if we look at what each part of the equation represents in terms of the overall neutron balance. The left-hand side of the equation: $\frac{\partial n}{\partial t} + v\hat{\mathbf{\Omega}} \cdot \nabla n + v\Sigma_t n(\mathbf{r}, E, \hat{\mathbf{\Omega}}, t)$, describes all losses of neutrons from the system. It describes the time rate of change of neutrons in the problem within the unit volume as a function of the speed of the neutron, v, its total macroscopic cross section, Σ_t, and the neutron number density, n. The product of neutron speed (cm/s) and number density (neutrons/cm^3) is flux, ϕ. Flux is in units of neutrons/cm^2·s. The term $\nabla n + v\Sigma_t n$ represents the leakage of neutrons into or out of the system. The product of the flux and the macroscopic cross section is the reaction rate density (reactions/ cm^3·s). The total macroscopic cross section accounts for all possible collision interactions. All reactions are removal reactions to include scatter because the neutron loses energy and changes direction in a scatter and will no longer be in the same population of neutrons with the energy and direction of interest.

Note that the neutron number density is a function of the position, $\mathbf{r}$, energy, E, solid angle, $\hat{\mathbf{\Omega}}$ and time, t. This is important because neutrons in most systems occur in a broad spectrum of energies. The energy dependence of cross sections means there are thousands of different reaction rates occurring at any time within the system.

When we move to the right-hand side of the equation, there are two major terms. The first, $dE'v'\Sigma_s(E' \to E, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}})n(\mathbf{r}, E', \hat{\mathbf{\Omega}}', t)$, represents the in-scatter of neutrons. It is a double differential because it accounts for the scatter of neutrons from a higher energy, E', into the energy of our population, E, and the scatter of neutrons from a solid angle, $\hat{\mathbf{\Omega}}'$, into the solid angle of our population, $\hat{\mathbf{\Omega}}$. This in-scatter adds to our population (gain).

The final term in Equation 3.1, $s(\mathbf{r}, E, \hat{\mathbf{\Omega}}, t)$, is the source term and represents the number of neutrons that are created in the unit volume that have the energy and direction of interest. These neutrons could be generated from

fission or some other source and add to the population (gain).

The two gain terms are differential in angle and energy. To calculate the total number of neutrons in the system, Equation 3.1 integrates the right-hand side over all 4π steradians of angle and over all energies from zero to infinity.

Equation 3.1 is an integro-differential equation with no closed-form solution. This means that there is no mathematical method available to solve the population of neutrons in a system in terms of energy, position, angle and time. Because of the lack of a solution, we have to use approximation techniques to generate a useful equation that we can write into a computer code so we can model our systems. Numerous methods have been created to attempt to adequately approximate a solution to the neutron transport equation. Codes that apply approximations to the transport equation are referred to as deterministic codes. In this text, we will describe two of the most common deterministic methods, discrete ordinates (S_N) and Legendre (P_N) approximations to include the P_1 approximation as it relates to diffusion theory.

3.2 The Discrete Ordinates Approximation

To begin the process of simplifying the neutron transport problem, several assumptions can be made. Equation 3.1 depends upon too many variables to be solved directly. One of the easiest approximations that we can use is to assume that our neutron population is not changing significantly with time. This is a pretty good approximation for systems that are essentially at steady state like a reactor operating at power. If our system is time-independent, the transport equation reduces some with the elimination of the variable t and becomes Equation 3.2.

$$v\hat{\mathbf{\Omega}} \cdot \nabla n + v\Sigma_t n(\mathbf{r}, E, \hat{\mathbf{\Omega}}) = \int_{4\pi} d\hat{\mathbf{\Omega}}' \int_0^\infty dE' v' \Sigma_s(E' \to E, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}) n(\mathbf{r}, E', \hat{\mathbf{\Omega}}') + s(\mathbf{r}, E, \hat{\mathbf{\Omega}}) \quad (3.2)$$

To calculate change in flux over time, we can then solve the problem at discrete time steps (t1, t2, t3...). However, even without time-dependence, Equation 3.2 is still too complex to be solved directly. We can use the power of a computer to help with the evaluation of the problem. Computers have strengths and weaknesses. They are excellent at storing large volumes of data and completing algebraic calculations.

Computers are not good at performing more abstract mathematical constructs like calculus. When a computer program performs and integral, for

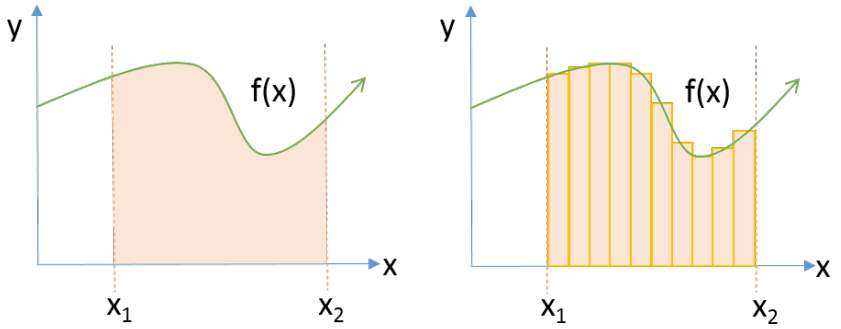


Figure 3.2: Integration of the function, f(x), from x1 to x2 explicitly on the left and approximated by a summation of discrete approximations on the right.

example, it normally performs a numerical integration where it breaks the components of the function into small discrete parts and then sums the solution of the equation to get an approximation of the result as presented in Figure 3.2. If the steps are sufficiently small, there is little accuracy lost in this approach. However, the finer the steps, or mesh, the more calculations performed and the longer it will take to reach a solution and more memory resources will be used.

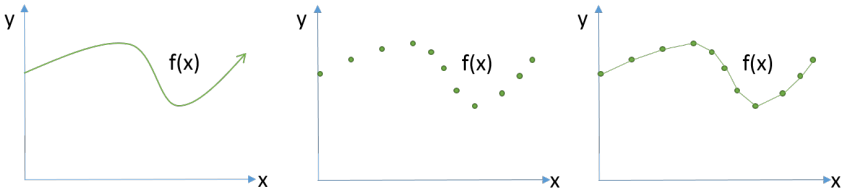


Figure 3.3: The discrete order approximation of the function f(x). The function is solved at discrete points and then interpolated between the points. The more points, the closer the approximation to the solution.

When the transport equation is discretized, all the variables of time, angle, space and energy have to be solved at discrete points. The discretization of time has already been discussed. The angular discretization is a little more involved. A common approach is to represent the angular distribution as a series of coupled weighted equations that are summed to perform a numerical integration over all possible angles. This set of equations is commonly called the S_N equations after early approaches that used a series of N straight line segments to treat the angular change of flux. (Duderstadt and Hamilton; 1976)

A common approach to the spatial distribution is to divide the geometry into 2-D and 3-D shapes or voxels in what is referred to as a mesh. Codes will solve the transport equation for the individual voxel and use the boundary

condition with the adjacent voxels to account for the transport of neutrons in the system. The leakage from one voxel serves as a source for the adjacent voxel until the boundary of the problem is reached. The size and shape of the voxel will greatly affect the accuracy of the problem. Smaller voxels allow for neutrons to travel in paths that are more natural and help reduce the error associated with discretization of space. The cost is computer time and memory.

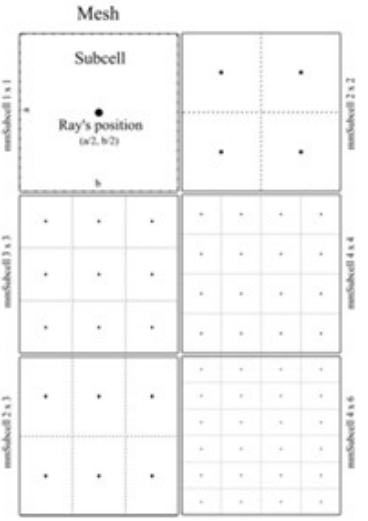


Figure 3.4: Ray positions within a voxel with different mmSubCells parameters. (Rearden and Jessee; n.d.)

Discretization of the energy is common in all computer models. This discretization is done by breaking the problem up into energy groups. The equation is solved using average cross sections for each group and the scatter of neutrons down from the higher group serves as a loss in that population and a gain in the lower group. Early computer codes used two or three-group models. These group structures would average the cross sections for interactions over both fast and thermal or fast, epithermal and thermal groups. Later, as computer capabilities, increased, computer codes would increase the number of cross section groups to 4 and 44-group models. Some of them would do initial calculations at 44 groups and then collapse to 4-group models for to speed up calculations. Modern computers can easily handle a near-continuous energy spectrum of 252-group models. However, for some cases, it makes sense to use less than 252 groups. For example, a fast reactor model does not require a fine resolution of thermal energies because neutrons are not being moderated. Additionally, large calculations can benefit from cross sections that are collapsed from a 252 group to a smaller group model. This is especially efficient in regions within the system where scatter is not as prevalent or resonances are not important. SCALE 6.2 performs multi-group analyses using either the older,

238-group structure or the newer 252-group structure and has a 56-group that is intended for light water reactors. (Rearden and Jessee; n.d.)

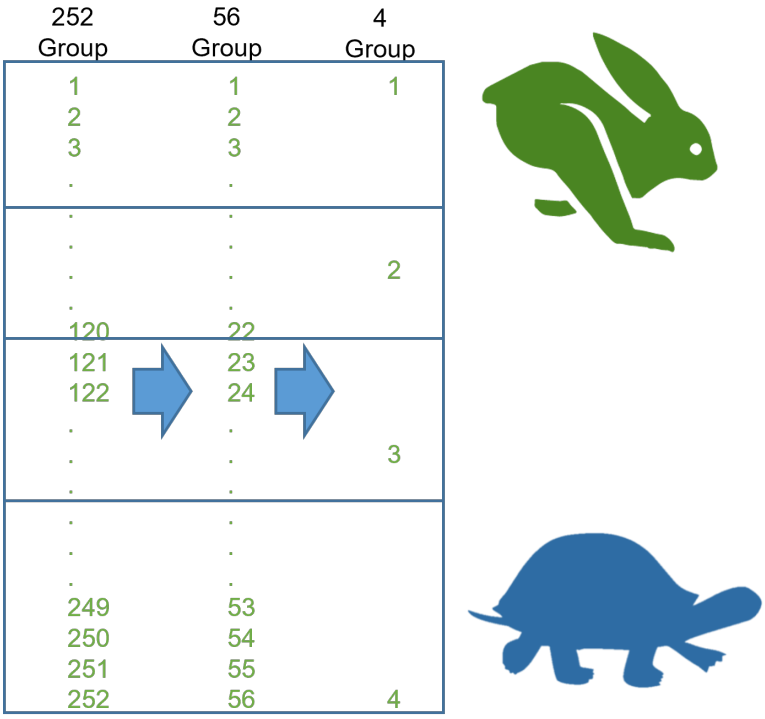


Figure 3.5: Common neutron energy groups. Smaller numbers are always fast neutron groups and larger are thermal. Some codes collapse from fine groups to course groups to increase efficiency.

After the discretization of all the variables, the computer is left with many algebraic expressions representing the flux. The computer then solves these expressions and sums up the results. Depending on the number of mesh points, energy groups and angular points, the problem can become enormous. For a problem with 100 x 100 x 100 mesh points, 10 energy groups and 10 angle points, there would be 10^8 simultaneous algebraic equations for each time step. (Duderstadt and Hamilton; 1976) It is for this reason, early computer codes relied on further simplifications down to diffusion theory when problems allowed for lower order approximations to achieve acceptable results.

3.3 P_N and Diffusion Theory Approximations

Another method of approximation to the neutron transport equation is to use a functional expansion to discretize the angular variables. This method is known

as the P_N method. Much like how the S_N equations started as a series of quadrature equations to break up the angular component, the P_N method relies on a series of equations. Specifically P_N expansion is based upon an expansion that corresponds to Legendre polynomials, $P_i(\mu)$ where μ corresponds to the cosine of the scattering angle. As with most polynomial expansions, the series can continue to higher and higher orders resulting in a finer discretization of the angle. The higher the order, the finer the calculations and greater the accuracy and, once again, the greater cost in computational speed and memory.

Without getting into the details of the derivation, when the series expansion is truncated at N = 1 (i.e. after two terms), the angular flux takes the form of Equation 3.3. (Duderstadt and Hamilton; 1976)

$$\varphi(\mathbf{r}, E, \hat{\mathbf{\Omega}}, t) \cong \frac{1}{4\pi}\varphi_{00}(\mathbf{r}, E, t) + \frac{3}{4\pi}[\varphi_{1x}(\mathbf{r}, E, t)\Omega_x + \varphi_{1y}(\mathbf{r}, E, t)\Omega_y + \varphi_{1z}(\mathbf{r}, E, t)\Omega_z] \quad (3.3)$$

Equation 3.3 is known as the P_1 approximation to the angular flux. It is one of the simpler forms of deterministic solutions to the transport equation. The P_1 approach is closely related to neutron diffusion theory.

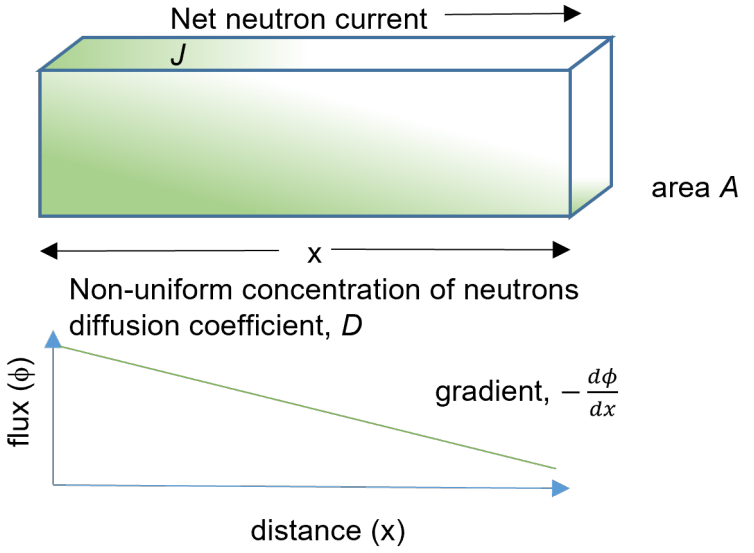


Figure 3.6: Using a diffusion theory approximation, neutrons move from areas of high concentration to low concentration according to Fick's Law.

Neutron diffusion theory is based upon Fick's law in chemistry which describes the movement of a solute in solution. As shown in Figure 3.6, the solute

will travel from regions of higher concentration to lower concentration where the rate of flow is proportional to the negative gradient. Neutrons in a large system, like a reactor, will generally flow from areas of higher concentration to lower concentration much in the way that a solute moves in solution. When applied to neutron transport, Fick's law becomes Equation 3.4. (Lamarsh and Baratta; 2001)

$$J_x = -D\frac{d\phi}{dx} \tag{3.4}$$

Where J_x is the neutron current or the net number of neutrons traveling through a unit area per second and D is the diffusion coefficient. Neutron diffusion equations lend themselves to hand calculations for simple models and rapid computer codes which can be discretized in terms of space, energy groups, and time. For many years, diffusion codes were the primary methods for solving reactor core problems because computers could not efficiently handle higher order S_N or P_N codes. The major drawback to the use of diffusion theory occurs in instances where the diffusion approach breaks down. This occurs at points where the problem becomes strongly dependent on angle. Namely:

1. Near regions of strong absorbers (such as control rods)
2. Near the boundary of the system
3. Near a neutron source
4. Anywhere the scattering of the neutrons are strongly an-isotropic (forward peaked such as in a beam).

In these cases, the neutron diffusion equation needs the fidelity of angle that is removed in the simplification of the problem. One approach to correcting the issue is to apply a correction factor, known as an extrapolation distance, on boundaries that forces the flux to travel to a point that it should go versus where the diffusion equation predicts it will go.

Diffusion approaches are still very useful for hand calculations and frequently taught in introductory reactor analysis courses. Even though modern computer systems have more than enough capability to handle codes that use either high order discrete ordinates methods or Monte Carlo transport to get more accurate results, diffusion theory still has many practical applications. Within high order codes, there is the possibility of using diffusion theory to help get to a solution faster within the construct of a higher order method. Additionally, core simulators will use a few-group nodal code for neutronics and couple it with another calculation for thermal hydraulics, fuel performance etc. The fact remains that large reactors contain so many individual fuel elements that the explicit modeling of each pin is unnecessary and unrealistic for many applications. (Rearden and Jessee; n.d.)

3.4 Deterministic Solutions to Transport Problems

Often it is helpful to analyze a heterogeneous reactor that is constructed of fuel rods in assemblies by looking at a single fuel pin surrounded by a column of coolant. This entity is known as a unit cell. The unit cell for a pressurized water reactor would consist of a cylinder of UO_2 fuel surrounded by a cylinder of metal cladding and a rectangular parallelepiped of water with dimensions of pitch x pitch x length of the fuel assembly. When we apply a reflected boundary condition to the sides of the parallelepiped, all neutrons leaving the cell are scattered back with the same energy and angle as though they were physically bounced off a mirror. When a single unit cell is mirrored, the resulting model represents an infinite array of unit cells and, in many ways, reacts like a full reactor without leakage. A similar boundary is known as a periodic boundary condition. In a periodic boundary, all the neutrons leaving the right side of the unit cell are copied and enter the left side. Depending on the problem, a periodic condition may be a better representation of the actual flux in the system.

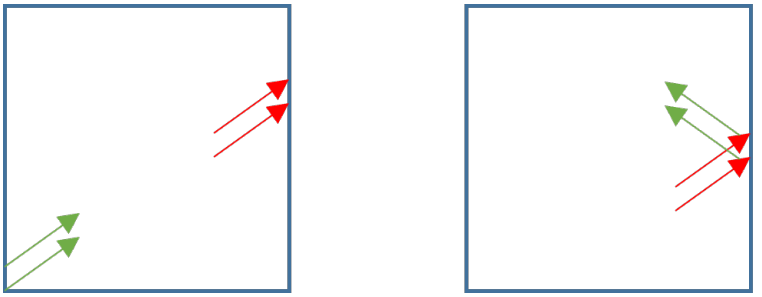


Figure 3.7: Example of two common boundary conditions. The left represents a periodic condition where particles leaving (red) on one side are repeated as entering (green) on the other side with the same angle of travel. The right figure represents a reflected condition where particles are returned as though they bounced off a mirror.

We describe computer codes that use approximations to solve the neutron transport code, such as discrete ordinate methods, as deterministic codes. These codes have several common properties and many advantages and some disadvantages when compared to the stochastic Monte Carlo codes we will explore later. The remainder of this chapter will be dedicated to using the SCALE6.2 package to solve neutron transport problems with deterministic methods. We will look at a single unit-cell calculation for a pressurized water reactor fuel pin using discrete ordinates and diffusion methods in the NEWT module of SCALE. Within NEWT, we will explore changes to the order of quadrature and changes to group structure. We will also use NEWT in the TRITON module to process

it for burnup. Finally, we will iterate the results for the fuel pin over various temperatures to calculate a fuel temperature coefficient for an infinite reactor.

Example 3.1
Use the T-NEWT module of SCALE6.2 to determine k_∞ for a reflected geometry model of a PWR unit cell with the following parameters:

Pitch:	1.311 cm
Fuel:	density: 10.2 g/cm3, 750 °C
	composition: UO_2 enriched to 4.5 wt% ^{235}U
	pellet radius 0.385 cm
Cladding:	density 6.407 g/cm^3, 300 °C
	composition: zirconium
	inner radius 0.390 cm
	outer radius 0.461 cm
Gap:	assume helium for material at 300 °C
Moderator:	density 0.715 g/cm^3, 300 °C
	composition: H_2O

Solution: To begin working with SCALE6.2, enter the FULCRUM editor. You can enter comments in the top of the file as a header using the apostrophe (') to lead the comments. The FULCRUM editor will indicate that the line is a comment by displaying it in green font. You should always put meaningful comments at the top of the file for future reference. Following the comments in the header, use the CTRL + SPACE keystroke to bring up the drop-down menu that allows you to select the module you wish to use as your input file. Select the T-NEWT module from the drop-down menu to tell SCALE which module to run. The FULCRUM editor will then populate a base of the input that has comments to let the user know what needs to be completed.

The following image displays what the input will look like with a couple of comment lines followed by the skeleton input inserted by the editor when user select the T-NEWT module. Note that the input has blue code to show the start and stop locations for blocks required for input labeled "read" and "end". Between those commands are comments that indicate to the user what information is required within each block. In some cases, sample data in the correct form is placed in those blocks to help the user. That sample data will almost always have to be adjusted to match the requirements for the problem being modeled.

```
'Computational Design in NE Example 3.1
'Kenneth Allen 14 January 2023
=t-newt parm=(   )
title-goes-here
xslib-goes-here

read comp

' TODO: define comp

end comp

read celldata

' TODO: define celldata

end celldata


read model
title-goes-here

read parameters

' TODO: define parameters

end parameters


read materials

  mix=1 end

' TODO: define materials

end materials


read geometry

global unit 1

' TODO: define global unit here

media 1 1 1


boundary 1 2 2


' TODO: add units below

end geometry


end model

end
```

Start to fill in the structure of the input file. Within the parenthesis following param write "bonami". This specifies the Bondarenko method for calculating the resonances for the cross sections. On the next line, write a title for the file. This will be repeated in the output file. On the next line write the library. In this case, we are using the ENDF 7 in a 252-group structure. At this point, the top input file should look like the following:

```
'Computational Design in NE Example 3.1
'Kenneth Allen 14 January 2023
=t-newt parm=(bonami)
NE361 Example 3.1 UO2 Unit Cell
v7-252
```

Move the cursor in between the read comp and end comp block and use the CTRL + SPACE keystroke to bring up the drop down menu to get assistance filling in the material compositions. The first material in the unit cell will be the UO_2 fuel at the weight percent, temperature and density indicated. The easiest way to specify this fuel is to use the

`wtptcomp -basic +volume fraction + temperature + isotopics`

option from the list. This option will default to the following:

```
read comp
wtptuo2 1 10.96 2 92000 88.15 8000 11.85 1.0 293
                  92234 0.5 92235 5.0 92238 94.5 end
```

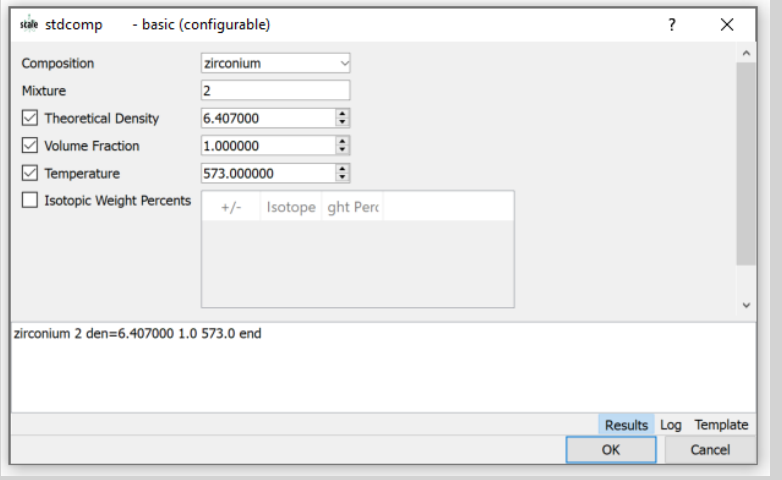
This is a default for weight fraction UO_2. The first line reads a name for the mixture starting with the letters "wtpt" indicating the fractions of elements and isotopes are in weight percent. The name is followed by the mixture number (1), the density (10.96 g/cm^3) the atom identifier for U in given by the Z number followed by three zeros (92000), the weight fraction of U in UO_2 (88.15%), the atom identifier for O (8000), the weight fraction of O (11.85%), the temperature in kelvin (293). The second line reads the isotope numbers given by the atomic number (Z) followed by the mass number (A). This identification number is known as a ZAID number. The three isotopes of U are given followed by their mass fractions. (^{234}U at 0.5%, ^{235}U at 5%, and ^{238}U at 94.5%). We can modify the input to meet our needs by changing the density, temperature (750 °C = 1023 k) and mass isotope fractions to read the following:

```
read comp
wtptuo2 1 10.2 2 92000 88.15 8000 11.85 1.0 1023
                  92234 0.5 92235 4.5 92238 95.0 end
```

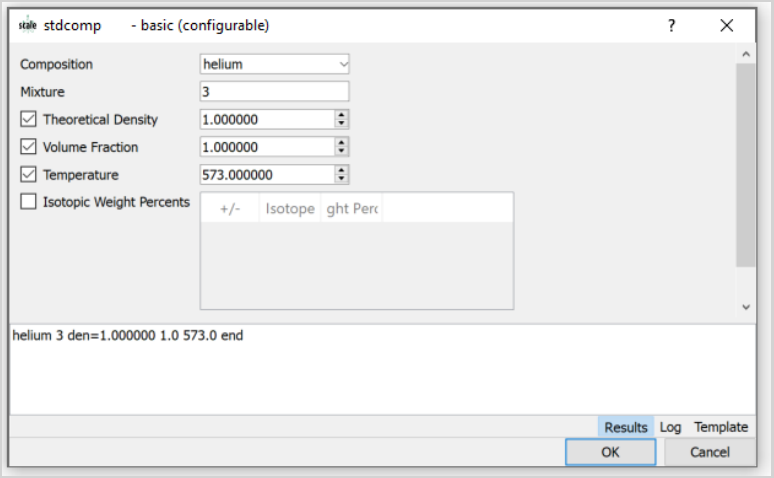
Now that we have the fuel composition, we can move on to specify the cladding, gap and moderator materials. There are several options for creating materials in the drop-down menu. For the fuel, we selected the weight

percent, because the fuel composition was provided in weight percent and the only additional information needed was the weight percentage of uranium and oxygen in the fuel which is always around 88.15% and 11.85% for UO_2 regardless of enrichment which are the provided values in the wtpt selection. We will use a different menu option for the other materials.

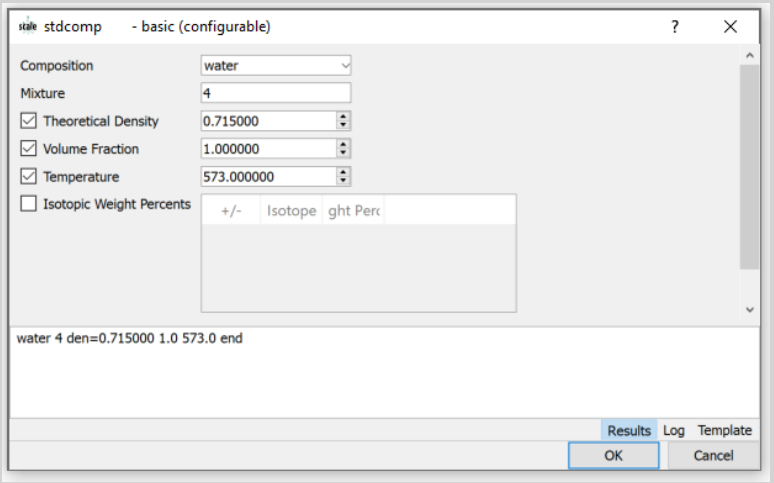
Now it is time to set up the cladding. Move the cursor down one line and use the CTRL + SPACE to get the composition drop down again. This time select the first option: `stdcomp - basic (configurable).` This will bring up a window that allows you to select the composition. Use the options in the window to select a zirconium cladding at the correct temperature. It should be material 2. The window should look like:



Repeat for helium in material 3. This will serve the area in the gap between the fuel pellet and the inner radius of the cladding.



Repeat again for water:



At this point, all the compositions are defined in the problem and we can proceed to define the unit cell regions and geometry to facilitate cross section processing.

Move the cursor after **`read celldata`** and CTRL + SHIFT to get the drop-down menu. Select:

```
lattice - squarepitch
```

The default will give you:

```
read celldata
latticecell squarepitch hpitch=0.0 1 fuelr=0.0 2 gapr=0.0 3 cladr=0.0 4 end
```

Modify the numerical entries to match the desired geometry and make sure the numbers following the dimensions correspond to the mixture numbers assigned in the composition block such as 1 for the fuel mixture and 4 for the moderator which is the outermost region of the unit cell.

```
read celldata
latticecell squarepitch hpitch=0.6555 4 fuelr=0.385 1
                        gapr=0.390 3 cladr=0.461 2 end
end celldata
```

After the **read model** line enter text for a title. Move the cursor after **read parameters** and CTRL + SHIFT to get the drop down menu. There are ≈65 different parameters that you can specify! For our problem the default values will suffice for all of them. However, we will specify the discrete ordinate quadrature number (S_N). By selecting

```
sn - order of sn level symmetric quadrature set
```

which will default to:

```
read parameters
sn=6
end parameters
```

We will leave it as an SN = 6 for now and change it in the future to see how the values and computation times vary.

The next region after the parameters includes the mixing table inside the materials block. The default information includes a mixture 1 denoted by **mix=1 end**. If we try to use the drop-down menu here, it will add another line that is the same. To generate the mixing table, we will modify the line by adding a command, **pn**, to control the scattering order and another optional command, **com**, for a comment where we will write the name of the material. As a rule of thumb, the scattering order should be 1 for everything except materials with high scatter cross sections, like a moderator. Using the same number convention we used for the materials, our materials block will look like:

```
read materials
mix=1 pn=1 com='UO2 Fuel' end
mix=2 pn=1 com='Zirc Clad' end
mix=3 pn=1 com='gap He' end
mix=4 pn=2 com='Water' end
end materials
```

We will now specify the geometry. The default global unit is set to 1. This will suffice for our simple model. Everything we define will be part of this global unit.

Move the cursor after **global unit 1** and CTRL + SHIFT to get the drop down menu. You will see a list of geometry shapes to include a cylinder and a cuboid. Select cylinder and the default will give:

```
read geometry

global unit 1
cylinder 1 1.0
```

This line reads that a cylinder with an integer name of 1 has a radius of 1.0 cm. We will change this to reflect that this our innermost cylinder which is our fuel with a radius of 0.385 cm. Repeat for the gap and clad and add a cuboid with dimensions of the half-pitch in the negative to positive x and y directions. We will name each geometry surface with an unique integer number. For our problem, we will choose to name the surfaces from 1 - 4, however we are not constrained to using sequential numbers. The total unit cell geometry should now look like:

```
read geometry
global unit 1
cylinder 1 0.385
cylinder 2 0.390
cylinder 3 0.461
cuboid 4 0.6555 -0.6555 0.6555 -0.6555
```

Now we will specify the media and the boundary condition to finish the geometry. The skeleton gives us the first media default as **media 1 1 1**.

This line shows the material id number, the bias placeholder and the region definition vector. The bias placeholder is not used in the NEWT model and is there to be consistent with the KENO model. Therefore, all of our bias numbers will be 1.

The region definition vector in SCALE is defines as positive values are "inside" the surface boundary number and negative values are outside. In this case, we said that our geometry consisted of three cylinders named 1, 2, and 3 and one cuboid named 4. For our first media (UO_2) it is defined as:

```
media 1 1 1
```

which is to say, media 1 (UO_2) has a bias placeholder of 1 and is inside cylinder 1. Our next media entry will be:

```
media 3 1 2 -1
```

which says media 3 (gap) has a bias of 1 and is outside cylinder 1 and inside of cylinder 2. Just to note, the positive and negative sense of composition is exactly opposite in MCNP from SCALE. We will see this in later examples.

After defining all four regions, we will have to define the boundary conditions for the problem. The default is given as:

```
boundary 1 2 2
```

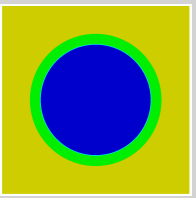
This states that the boundary of our problem is unit 1 with a local grid of 2 by 2. We need to adjust this because our cuboid is the boundary, and we can make a finer grid of 10 by 10.

```
boundary 4 10 10
end geometry
```

We can finish up our problem by specifying the boundary conditions. The default boundary is reflective so that will work. However, in order to specify the conditions, move the cursor after end geometry and CTRL + SHIFT to get the drop down menu. Select **bounds**. This will get the default read and end statements. In the middle we will write `all=refl`. We could also have specified periodic, vacuum or white as an option. Our final input deck is:

```
'Computational Design in NE Example 3.1
'Kenneth Allen 14 January 2023
=t-newt parm=(bonami)
NE361 Example 3.1 UO2 Unit Cell
v7-252

read comp
wtptuo2 1 10.2 2 92000 88.15 8000 11.85 1.0 1023
                  92234 0.5 92235 4.5 92238 95.0 end
zirconium 2 den=6.407000 1.0 573.0 end
helium 3 1.0 573.0 end
water 4 den=0.715000 1.0 573.0 end
end comp

read celldata
latticecell squarepitch hpitch=0.6555 4 fuelr=0.385 1
                        gapr=0.390 3 cladr=0.461 2 end
end celldata

read model
UO2 Unit Cell

read parameters
sn=6
end parameters

read materials
  mix=1 pn=1 com='UO2 Fuel' end
  mix=2 pn=1 com='Zirc Clad' end
  mix=3 pn=1 com='helium gap' end
  mix=4 pn=2 com='water mod' end
end materials

read geometry
global unit 1
cylinder 1 0.385
cylinder 2 0.390
cylinder 3 0.461
cuboid 4 0.6555 -0.6555 0.6555 -0.6555
media 1 1 1
media 3 1 2 -1
media 2 1 3 -2
media 4 1 4 -3
boundary 4 10 10
end geometry

read bounds
all=refl
end bounds

end model

end
```

Now that we have completed the input, we can check for errors by first looking at the "Validation" window on the bottom right tab in FULCRUM. If the window is blank, then the program has not found any errors. We can then look at our geometry by selecting the "View" tab at the top and "View Geometry". We will get a default top-down view for our unit cell colored by different materials:



The viewer has some options to include outlines and change colors as well as viewing from the side versus the top.

We can now run our model of an $S_N = 6$ discrete ordinate unit cell for a PWR at operating temperature. Do so by clicking on the "Run" tab at the top of the page. You can observe the progress by clicking on the messages tab at the bottom and looking at the window. When it is done running, you will see something like:

```
   Scale job C:/Users/Kenneth.Allen/Example3-1.inp is finished.
Output is stored in C:\Users\Kenneth.Allen\Example3-1.out

   Process finished with 0 return code; ran in 16 secs,
finished at Sun Jan 15 17:59:52 2023
```

Navigate to the file folder where the output is stored and open it in a text editor, like TextPad®. As you scroll through the output, you will see a lot of data that describes the input values, defaults and calculations performed. As you move through, you will see:

```
Four-Factor Estimate of k-infinity.  Fast/Thermal boundary:   0.6250 eV
      Fiss. neutrons/thermal abs. (eta):          1.869959
      Thermal utilization (f):                    0.938067
      Resonance Escape probability (p):           0.632055
      Fast-fission factor (epsilon):              1.257538
                                                 ----------
      Infinite neutron multiplication             1.394254
```

This is the value for k_∞ we were asked to determine.

3.5 Variations to Discrete Ordinates Solutions

Often the most challenging part of any computational design is getting a working, error-free, input file. Once we have that example input, we can modify it to determine many design elements and gain a better understanding of those parameters and their effects on our problem. In Example 3.1, we completed a simple single unit cell calculation for criticality for a UO_2 fueled PWR. For this problem, we used most of the default parameters and the given geometry

and materials for the initial calculation. Now we can adjust those elements and learn a great deal about our design and effects of changes to optimize both our calculations, materials and geometry. As a rule of thumb, only modify ONE parameter at a time when analyzing effects. If you change a material and its temperature and its density and the order of quadrature for the calculation, you will not have any idea which change affected the outcome. You could have changed two parameters that cancelled each other out and you will be unaware of the effect. Also, it is important to analyze the output and effect and make sure that the given change and response makes sense. If you change the amount of fissile material, you should see a change in the criticality in the calculation. Never take the output from a computer code at face value. If a calculation gives you an unexpected response, you need to verify that the calculation is correct and determine the underlying reason for the result.

The first variation we will perform in Example 3.2 is to change the order of quadrature , or S_N, for our unit cell calculation. We will plot the changes in both criticality and computer time to see how the order of quadrature affects each.

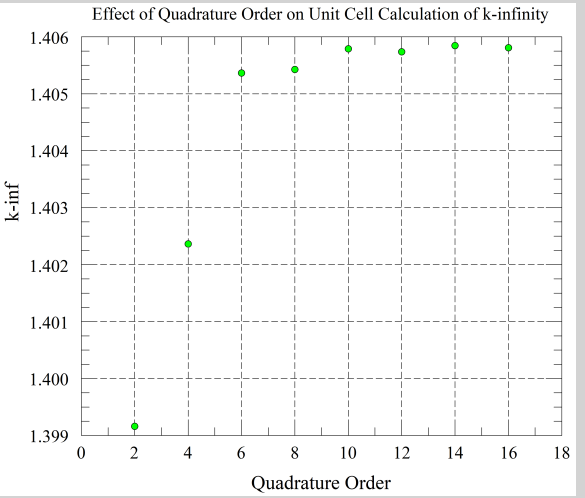
Example 3.2
Given the PWR unit cell in Example 3.1, re-run the calculations in t-NEWT and change the order of quadrature from 2, to 4, 6, 8, 10, 12, 14, 16. Record the k_∞ and computer time for each calculation. Plot the results for each and analyze the effect of the order of quadrature on the reported value for criticality and the time for the calculation.
Solution: To change the order of quadrature, the only thing that must be adjusted is the line:

```
read parameters
sn=6
end parameters
```

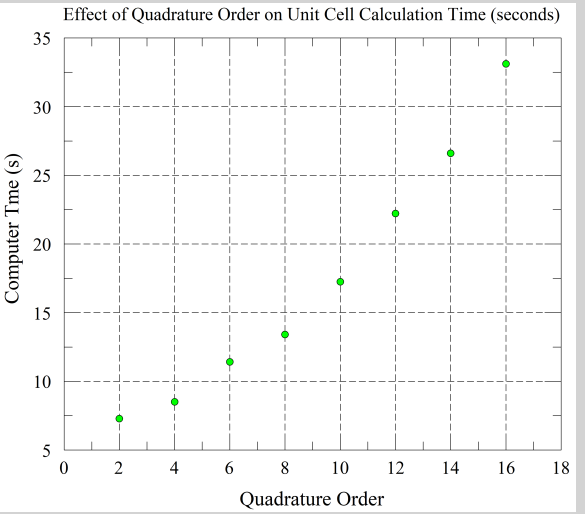
Change 6 to 2 and run the model and record the results. NOTE: the official computer time is found at the end of the output file and not in the message block from Fulcrum. Continue for each until reaching $S_N = 16$. Input results in a table for plotting.

Quadrature Order	k-inf	Computer Time (s)
2	1.399160	7.28249
4	1.402365	8.50198
6	1.405365	11.4146
8	1.405428	13.4041
10	1.405790	17.2534
12	1.405736	22.2252
14	1.405845	26.6050
16	1.405809	33.1177

The computer time may vary depending on the computer and processor speed. Any change in speed due to the machine's capability should be proportional and the overall trend should look the same when plotted. The plot of criticality as a function of quadrature order looks like:



The plot of computer time as a function of quadrature order looks like:



When we look at the results, we see that computer time increases nearly exponentially as the order of quadrature increases. The criticality reported for the SN = 6 through $S_N = 16$ are all very close (within 0.01% of each other)

For our simple unit cell single calculation, the overall time is still short (< 1 min). However, the time for an order 16 calculation is about 3 times as long as an order 6 calculation and the criticality results are the same out to 4 significant figures and are within 0.01% of each other. If we were doing many criticality calculations for a long depletion model, the time increase would be significant. The goal should be to get accurate results as efficiently as possible. Additional changes to our model that may speed up the calculation could be to collapse the cross sections energy groups from 252 to a more course structure such as 56 or 4 group models or making a broader mesh within the unit cell. We will not analyze those effects to the model at this time and will leave that to the problems at the end of this chapter. A more efficient method of running multiple calculations, versus changing a parameter and rerunning the code manually is to set up a restart within the input file. Information on doing a restart can be found in the SCALE manual.

By changing a parameter in the input, we can determine more about our model than just optimization of the calculation. For example, we can adjust the temperature of the fuel to determine the fuel temperature coefficient for the infinite reactor model. As presented in Example 1.3, the resonance cross section changes for many isotopes with temperature. Specifically, the resonances in the fuel will get wider at higher temperatures due to what is known as Doppler broadening. Within the life-cycle for neutrons in a thermal reactor, the increased width of the resonances increases the probability of resonance absorption as the neutrons scatter from high energies to low energies. The effect is a reduction in the resonance escape probability (p) in the four-factor formula. The reduction in p causes a reduction in k. The overall effect is a negative feedback response where, as the temperature of the fuel increases, the criticality decreases and vice versa. The temperature coefficient, α, presented in Equation 3.5 is a measure of how much the criticality changes with temperature and has units of criticality per degree kelvin.

$$\alpha = \frac{\Delta k}{\Delta T} \tag{3.5}$$

A negative coefficient is a safety design in a reactor. The temperature of the reactor is proportional to the flux and the power. As the power increases, the temperature and flux will increase. Because of the negative temperature coefficient, the increase in power and temperature will slow the response in criticality. This self-regulating feature makes reactors with a negative coefficient inherently stable and is a requirement for the licensing of reactors in the United States. In Example 3.3, we calculate the temperature coefficient for our infinite reactor model using our unit cell model.

Example 3.3
Given the PWR unit cell in Example 3.1, re-run the calculations in t-NEWT to calculate the fuel temperature coefficient for the infinite reactor model. Use an S_N=6 quadrature and adjust the fuel temperature from room temperature (293 K) to 2093 k in increments of 200 k. Plot the results and use the slope of the line to determine the coefficient. Is it positive or negative? Do the results make sense?

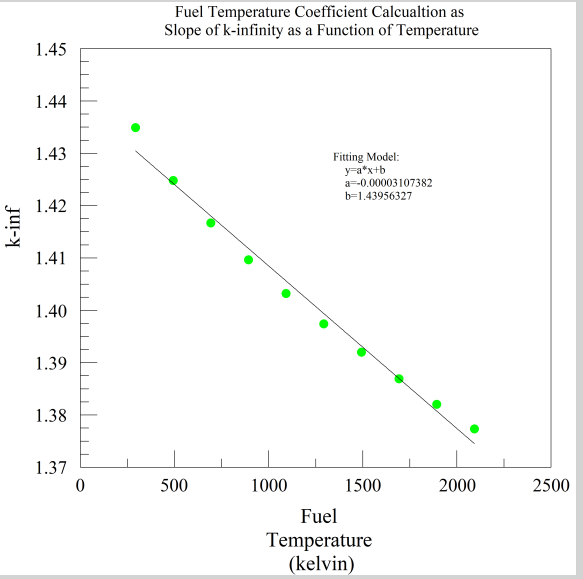
Solution: To adjust the temperature of the fuel, change the value in the composition for the UO_2 (shown below) from 1023 k to 293 k.

```
read comp
wtptuo2 1 10.2 2 92000 88.15 8000 11.85 1.0 1023
                92234 0.5 92235 4.5 92238 95.0 end
zirconium 2 den=6.407000 1.0 573.0 end
helium 3 1.0 573.0 end
water 4 den=0.715000 1.0 573.0 end
end comp
```

Run the model at each of the temperatures and record the results.

Fuel Temp (K)	k-inf
293	1.434917
493	1.424811
693	1.416695
893	1.409628
1093	1.403209
1293	1.397407
1493	1.392006
1693	1.386895
1893	1.382012
2093	1.377342

Now plot the results and apply a linear trendline:



For our model, we can estimate the fuel temperature coefficient as:

$$\alpha = -3.107382 \times 10^{-5} \frac{\Delta k}{\text{degree K}}$$

It is a negative coefficient which makes sense because of the Doppler broadening of the resonances at higher temperatures in the fuel which will decrease the resonance escape probability resulting in a lower k_∞.

3.6 Depletion Calculations in TRITON

We have been using the TRITON module in SCALE to perform a single point in time calculation with the NEWT discrete ordinates method (t-NEWT). The criticality calculation examined the unit cell at the beginning-of-life (BOL) meaning the cell had not been depleted or operated at power over a time. When a reactor is initially fueled, it has excess reactivity meaning that, in the absence of control mechanisms, it would be supercritical with a multiplication factor greater than 1.0. For our unit cell representing an infinite reactor, the multiplication factor was approximately 1.403209 at operating temperature at BOL. Naturally, the reactor isn't operated in a supercritical condition. The excess reactivity is held down using control rods, chemical shim, and other burnable absorbers in the fuel to keep k = 1.0. As the fuel fissions, it has less reactivity. The plant operators then reduce the control mechanisms by withdrawing control rods, removing chemical shim and the burnable absorbers deplete as they

absorb neutrons. At end-of-life (EOL) all of the control rods are withdrawn, and the burnable poisons have depleted and the chemical shim is removed. When the reactor can no longer maintain k = 1.0 it has to be shut down and refueled. Our unit cell model does not have control rods, burnable absorbers, or chemical shim. We could add a burnable absorber to the fuel and/or boric acid to the water. Adding control rods would be more of a challenge requiring the model be more representative of the assembly with a square array of unit cells with open cells for the control rod guide tubes. A typical example would be a Westinghouse 17 x 17 assembly. This text won't get into a discussion of using SCALE for a unit assembly model until later chapters where we look at repeated lattice structures in SCALE.

For our problem, we will do a straight depletion of our unit cell without any control mechanisms in place. This will give us a good measure of how long the reactor could operate at full power if it were fueled with all new fuel rods of the same weight percent without leakage. When the depletion curve crosses 1.0, it represents the EOL for the reactor fuel. The number of days it takes to go from BOL to EOL represents the number of effective full power days (EFPDs) in this model. This model is simple as it assumes a constant power and temperature. The only values that are needed to add to our t-NEWT model are the time steps for depletion and the average specific power for the model. The units for average specific power are given in megawatts per metric tonne (1000 kg) of initial heavy metal (MW/MTHM) material. Assuming a large PWR operates around 3000 MW thermal and has around (17 x 17 x 177) or about 50,000 fuel rods each with a volume of around 210 cm^3 a density of 10 gm/cm^3 and 88% mass of U in UO−2 the power density is around 32 MW/MTHM. Typical values for a large PWR are between 30 and 40 MW/MTHM.

An actual full core model would account for variations across all the fuel rods in each assembly and all of the assemblies (on order of 177 to 193) in the core. Each assembly will have a different position in the core and will have a unique flux and burnup history. The single unit cell is still useful for doing scoping calculations to understand trends and analyze effects of design changes quickly. Large, full-core, models will sometimes use single unit cells in detailed calculations and then apply the results to average values that represent the assemblies to speed up the calculations.

Example 3.4
Rewrite the unit cell problem in Example 3.1 for a TRITON depletion (t-depl) calculation. Use a specific power density of 35 MW/MTHM and deplete for the following depletion times of 2, 10, 50, and 15 steps of 100 days each for a total depletion of 1562 days.
Solution: Open the input file for Example 3.1. Change the **t-NEWT** to **t-depl**. The validation tab will turn red and there will be two warnings

indicating that the input file is missing burnup and depletion information. Move the cursor to the line right after `end cell data` and use the CTRL + SHIFT to get the drop-down menu and select depletion. After you click enter, replace the comment, `'TODO: define depletion` with **1**. This indicates that material 1 (UO_2) fuel will be depleted. For our model, this is the only material that can be consumed.

After the depletion block is complete, move the cursor on the line after `end depletion` command and use the CTRL + SHIFT to get the drop-down menu and select burndata. Place the cursor between the begin and end burndata command and use the CTRL + SPACE to select cycle and the following default information will appear:

```
read burndata
    power=40.0 burn=365.0 down=0 nlib=1 end

end burndata
```

Change the default data to reflect the burnup steps specified in the problem statement. The `down=0` statement indicates the number of days the reactor is at zero power after the burnup step. This is useful if you include a down time for decay following discharge but not needed for this problem so it will remain zero. The statement `nlib=1` reflects the number of subintervals. Our problem will only have one. The final burndata card will look like:

```
read burndata
    power=35.0 burn=2.0 down=0 nlib=1 end
    power=35.0 burn=10.0 down=0 nlib=1 end
    power=35.0 burn=50.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
    power=35.0 burn=100.0 down=0 nlib=1 end
end burndata
```

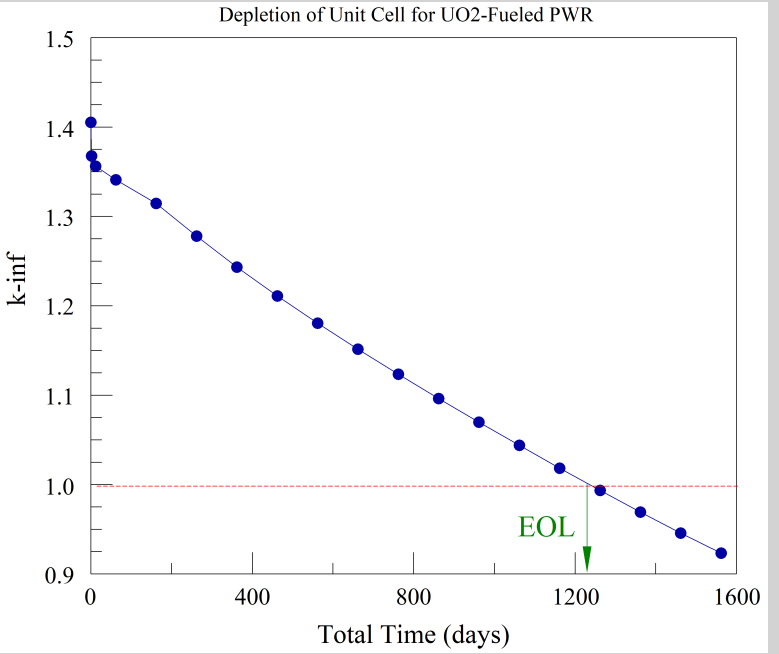
Once the model is completed, we can go to the output file and retrieve the k-infinity values for each time step and record and plot the results.

Time Step	Total Time	k-inf
0	0	1.405365
2	2	1.367746
10	12	1.356093
50	62	1.340834
100	162	1.314479
100	262	1.277912
100	362	1.243256
100	462	1.210924
100	562	1.180493
100	662	1.151464
100	762	1.123490
100	862	1.096301
100	962	1.069736
100	1062	1.043728
100	1162	1.018261
100	1262	0.993360
100	1362	0.969130
100	1462	0.945669
100	1562	0.923109

We can see from the table and the following figure that the infinite reactor would operate for approximately 1225 days before it would no longer remain critical. This would be over 3.0 years. However, a real reactor is not filled with all fresh fuel at 4.5% enrichment and this model does not have leakage. For an actual PWR, it would need to be refueled every $18-24$ months.

Notice how the depletion curve has a very sharp drop and then becomes nearly linear. This is due to the buildup of fission product poisons (mostly Xe and Sm) during the first 48 hours of operation. Once the fission products reach an equilibrium state, the burnup is nearly linear.

The output file has much more information besides criticality. For example, the end of the file has a burnup table that shows all the major nuclides and their concentrations. This includes both fission products and actinides. This data is useful for determining residual heat and radioactivity in spent nuclear fuel.


Depletion of Unit Cell for UO2-Fueled PWR
k-inf
1.5
1.4
1.3
1.2
1.1
1.0
0.9
EOL
0
400
800
1200
1600
Total Time (days)

Bibliography

Duderstadt, J. J. and Hamilton, L. J. (1976). *Nuclear Reactor Analysis*, Wiley New York.

Lamarsh, J. R. and Baratta, A. J. (2001). *Introduction to Nuclear Engineering*, Addison-Wesley series in nuclear science and engineering, 3rd edn, Prentice Hall, Upper Saddle River, N.J.

Lewis, E. E. and Miller, W. F. (1993). *Computational Methods of Neutron Transport*, American Nuclear Society.

Rearden, B. T. and Jessee, M. A. (n.d.). Scale code system.
URL: *https://www.osti.gov/biblio/1426571*

3.7 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- Create MOX format for SCALE

3.8 Problems

1. Complete a parametric analysis for the unit cell presented in Example 3.1 by changing the local grid on the boundary from 2 to 4, 6, 8, 10, 12, 14, 16, 18 and 20. Record and plot the k-infinity values and computer time as a function of grid number. Comment on the effect of a larger grid.

2. Complete a parametric analysis for the unit cell presented in Example 3.1 by changing the energy group structure from a 252-group model to a 56 and a 4-group model. Record and plot the k-infinity values and computer time as a function of group size. Comment on the effect of group size on the accuracy and speed of the model.

3. Complete a parametric analysis for the unit cell presented in Example 3.1 by changing the enrichment of the fuel from 1% to 5% enriched by steps of 0.5%. Record and plot k-infinity versus enrichment. Comment on the effect of enrichment on k-infinity. Is it linear?

4. Complete a parametric analysis for the unit cell presented in Example 3.1 by changing the radius of the fuel pellet from 0.25 cm to 0.50 cm using a step interval of 0.05 cm. Keep the thickness of the gap and cladding the same (their radii will have to change with the fuel pellet dimension). Also adjust the pitch to keep the spacing between fuel rods proportional. Record and plot the k-infinity as a function of fuel radius. Comment on the effect of radius on k-infinity.

5. Change the fuel composition in Example 3.1 from a standard UO−2 mixture to a standard mixed-oxide (MOX) fuel consisting of plutonium and uranium. The fuel should be at 4.5% enriched in fissile material. Show your calculations and references for the determination of which isotopes and concentrations you used to get the MOX composition. Report the k-infinity for the MOX-fueled model.

6. Besides the fuel temperature coefficient presented in Example 3.3, another important coefficient is the void coefficient. This coefficient compares the criticality as a function of void in the moderator. Change the density of the water in the unit cell used in Example 3.1 from 1.0 g/cm^3 down to 0.5 g/cm^3 by a step interval of 0.05 g/cm^3. Run each model and record and plot the results where void is 1.0 - density in the x-axis. Use the slope of a trendline to estimate the void coefficient as the change in reactivity as a function of percent void. Is the void coefficient positive or negative? Does it make sense? Does the void coefficient make the reactor more inherently stable or unstable and why?

7. Calculate the temperature coefficient for the MOX-fueled unit cell created in Problem 5 and compare it to the results in Example 3.3.

8. Calculate the void coefficient for the MOX-fueled unit cell created in Problem 5 and compare it to the results in Problem 6.

9. Complete the depletion analysis performed in Example 3.4 on the MOX-fueled unit cell created in Problem 5 and compare the results to Example 3.4.

10. Use SCALE manual to write an input with a restart to do multiple runs to do parametric analysis problems- like 1-9 using the restart command.

Chapter 4

The Monte Carlo Method for Transport

As previously outlined, this text is not designed to provide all the background and details necessary for the student to write his or her own discrete ordinates deterministic transport code. In the same way, it is not intended to provide all the necessary instruction in Monte Carlo methods so a student could write a complete Monte Carlo transport code. There are several good references that can provide that level of detail to include Lewis and Miller's *Computational Methods of Neutron Transport* and Dupree and Fraley's *A Monte Carlo Primer.* (Dupree and Fraley; 2002; Lewis and Miller; 1993) As with the deterministic solutions, the purpose of this chapter is to provide the student with a basic understanding of how Monte Carlo codes work and a practical approach to solving simple problems using one of the most powerful Monte Carlo codes available – Monte Carlo Neutral Particle (MCNP 6.2).

4.1 Introduction to the Monte Carlo Technique

Naturally occurring events are subject to random processes. Given enough data about the probability of occurrences, these events can be predicted using a computer code. A Monte Carlo code is one that mimics a randomly occurring event using the probability distributions associated with that event. The term Monte Carlo was applied to the technique because of the history of casinos and gaming in the city in Monaco. A computer is uniquely suited to Monte Carlo methods because of its ability to do large numbers of calculations and hold enormous data files.

Suppose we set up an experiment where we had a group of individuals throw identical paper airplanes from a balcony overlooking a gymnasium floor. How far the airplane would travel and where it lands would depend on the

velocity of the throw, the angle of the release, and any prevailing wind. Given a large enough sample of trials, we could begin to discover trends. For instance, a headwind could cause the distance the plane traveled to go a percentage shorter and an initial angle of attack of 30 degrees may provide the optimum glide distance. Based on a very large data set, we could measure the initial conditions and predict the result for the throw without having to actually have the person throw the plane. If we could predict the initial conditions for a large number of throws, we could predict where the group of planes would land and their distribution across the floor without launching a single one.

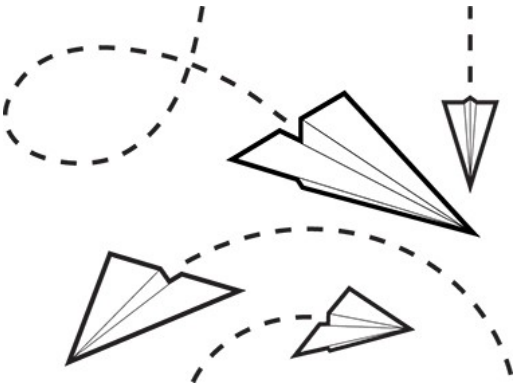


Figure 4.1: The flight path of a paper airplane is dependent on both the environmental effects of wind and the skill of the person throwing it. For a large data set, trends can be predicted using random processes.

Our prediction of initial conditions would be based on the probabilities that we measured experimentally. For example, we may have observed that 1 in 10 throws was at the optimum angle of 30 degrees or that for 2 in 10 throws, there was a significant headwind. We could then have a computer pick a random number between 1 and 10 and, if it was 1, it would be scored as a launch at 30 degrees. Similarly, if another random number between 1 and 10 resulted in either a 1 or 2, it would be scored as having a headwind. Note that a random number was picked for each initial condition because they are mutually independent meaning there is the same probability of a headwind regardless of the initial angle of throw. The data set corresponding to each variable is known as the probability distribution function (PDF) . When the PDF is summed for the entire group, it results in the cumulative distribution function (CDF) which is a model of the real-world event. Note that there is an error associated with both the experimental measured data set and with the randomly generated Monte Carlo model. The amount of error in the measured data can be reduced by increasing the number of measurements. Similarly, the error in the Monte Carlo model can be reduced by increasing the number of trials. Assuming the PDFs correctly describe the individual events, the CDF's accuracy increases as the number of trials increases.

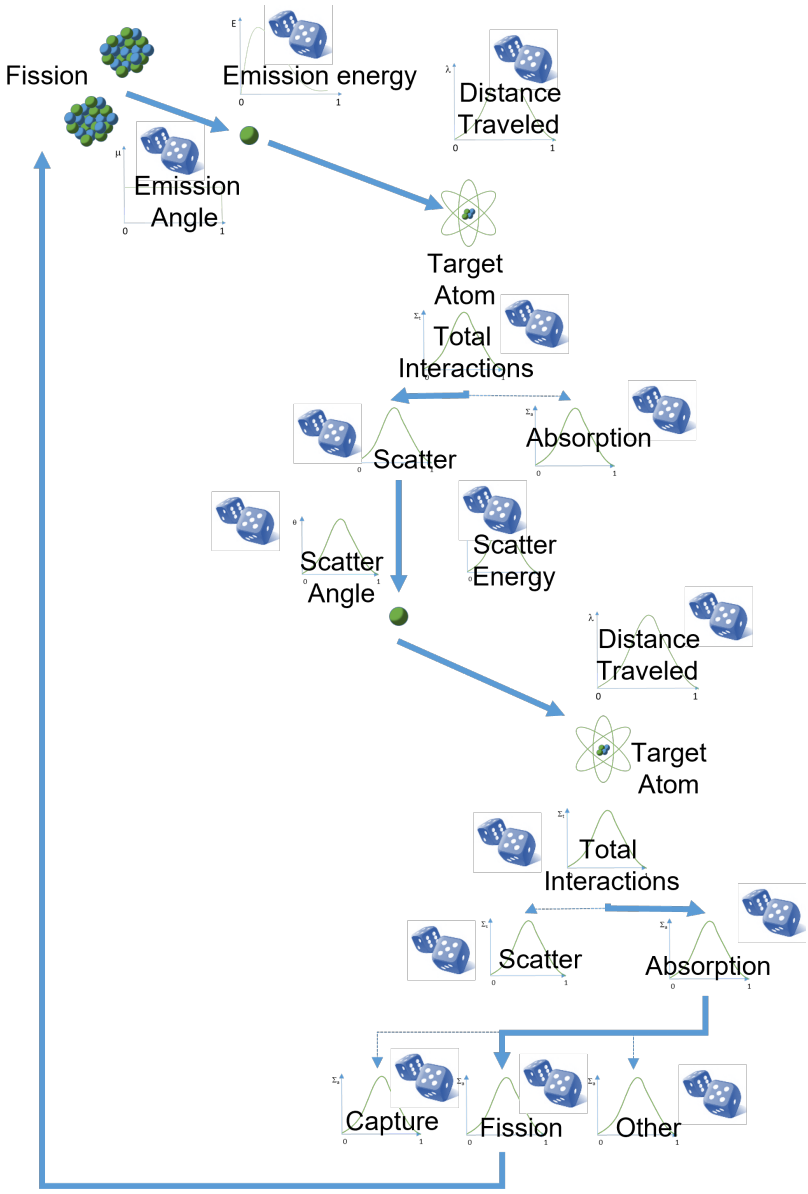


Figure 4.2: The random walk process for a neutron born in fission. At each point along the path, a random number is chosen (dice) and applied to a probability distribution function associated with that event. The thick line is one example path of what could occur.

As we saw in Chapter 3, the neutron transport problem is complex and a function of many variables including time, space, and energy. Although complex in a holistic sense for a flux of neutrons, each interaction for a single neutron is well-defined by a probability distribution function (PDF). The life cycle of a neutron can be tracked from birth to absorption using a series of PDFs. This is known as a "random walk process" and is presented in Figure 4.2. For example, if a neutron is born in fission, it will have an energy associated with a PDF based upon the fission spectrum. It will travel from the fission site in a PDF that represents an isotropic point source and then move a distance which is a function of its mean free path in that material. The mean free path is based on the macroscopic cross section of the material which is dependent on the neutron's energy. The neutron will then have an interaction which will either be a scatter or an absorption (Figure 1.7). Each of the interactions has a PDF associated with the neutron energy and the material. If the neutron scatters, the angle and residual energy is based upon another PDF. Eventually, the neutron will be absorbed and will generate either a gamma particle, or a fission or some other particle. Because all the cross sections and interactions for radiation have been closely studied and measured, we can use a Monte Carlo code to walk each particle from "birth" to "death." Figure 4.2 is an illustration of one such process. The neutron is born in fission, and, at each interaction point, the computer picks a pseudo-random number on a scale from 0 to 1 and compares the value to a PDF for that process normalized to 1.0. Where the random number falls on the PDF determines the outcome for the step in the random walk (i.e. how far the neutron travels, what energy it has, what angle it scatters into). The Monte Carlo code tracks the particle until it is absorbed and is removed from the population, or it leaves the boundary defined by the user. Radiation transport problems generally consist of very large populations with known PDFs, making them especially suited to Monte Carlo solutions.

4.2 Example Monte Carlo Calculation of π

One of the earliest examples of the Monte Carlo process was performed by Georges-Louis Leclerc, Comte de Buffon in the 18th century. Buffon's experiment was conducted by dropping a needle onto a surface randomly. The surface was bisected by a series of equally spaced parallel lines. The length of the needle was exactly one half the distance between the parallel lines. Through some math, he was able to determine a relationship to when the needle touched one of the lines to the ratio of the circumference of a circle to its diameter (i.e. π) (Dupree and Fraley; 2002)

Another method for estimating π using a random process involves using square and a circle. Say we have a unit circle (radius = R), inside a square of sides 2R x 2R where the sides of the square are tangent to the circle as presented in Figure 4.3.

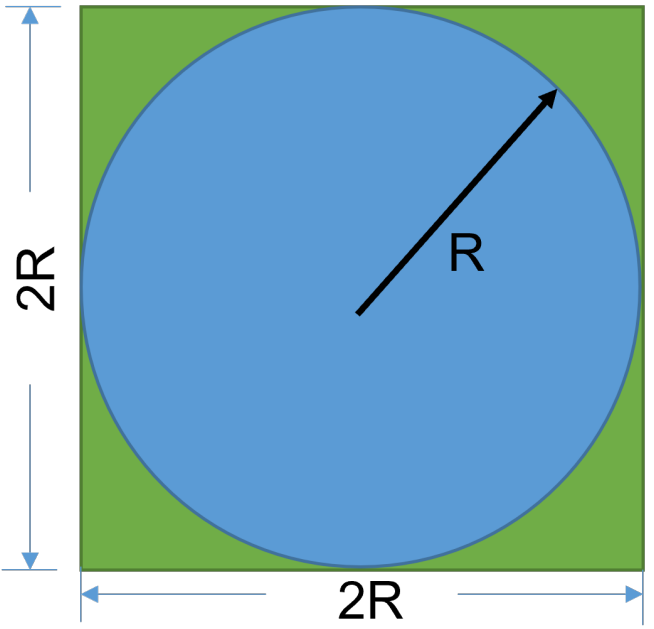


Figure 4.3: A circle radius R just inside a square tangent to 4 points.

The area for the square is:

$$A_{square} = side^2 = 2R^2 = 4R^2 \tag{4.1}$$

The area for the circle is:

$$A_{circle} = \pi R^2 \tag{4.2}$$

If we randomly sample points within the square, some of those points will fall within the circle. The number of points sampled will be proportional to the area. We can then set up a ratio where:

$$\frac{N_{circle}}{N_{square}} \cong \frac{\pi R^2}{4R^2} = \frac{\pi}{4} \tag{4.3}$$

Solving for π:

$$\pi \cong 4\frac{N_{circle}}{N_{square}} \tag{4.4}$$

Because all four quadrants of the circle are the same, we can simplify the problem by just focusing on the positive (top-right) quadrant. If we plot the positive quadrant on x and y axis the result is the left side of Figure 4.4.

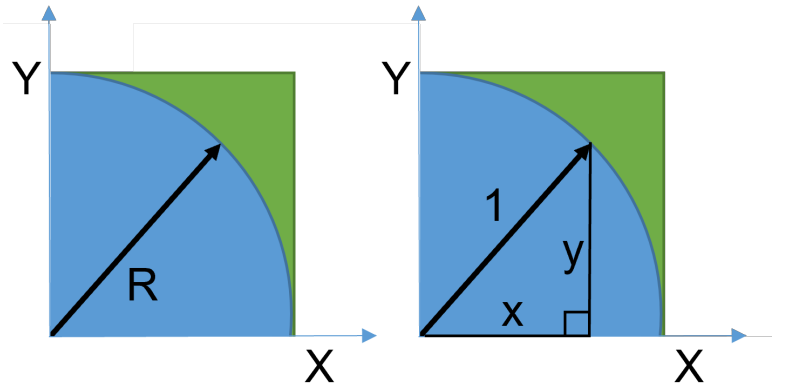


Figure 4.4: The positive quadrant of the circle in Figure 4.3 on the left. The same quadrant on the right with the radius set to 1 and a right triangle consisting of the x and y distances to the intersection of the arc and radius lines.

If we set the radius of the arc = 1, we see that, by the Pythagorean Theorem, the radius is given by:

$$R = 1 = x^2 + y^2 \tag{4.5}$$

We can then use a random sample of points with coordinates of x and y between 0 and 1 and evaluate if they are inside the circle if the following condition is true:

$$x^2 + y^2 < 1 \tag{4.6}$$

Now we can use Equations 4-4 and 4-6 to get a relationship.

$$\pi \cong 4\frac{N_{x^2+y^2<1}}{N_{Total}} \tag{4.7}$$

This relationship is an approximation because the area is being represented by points. The more samples taken, the closer the approximation will be. As with any random process, we must evaluate the precision of the result. We can do this by measuring the standard deviation of Equation (4-7). The standard deviation is:

$$\sigma = 4\sqrt{\frac{1}{N_{Total}}\sum_{i=1}^{N_{Total}}\left(N_i - \frac{\pi}{N_{Total}}\right)^2} \tag{4.8}$$

Where N_i represents each individual calculation performed using Equation 4-6. This estimate for π can be easily written into a Julia script that allows the user to vary the number of samples to get a value and a standard deviation. This script and the analysis of the results is presented in Example 4.1. (Dupree and Fraley; 2002)

Example 4.1
Write a Julia program that uses the positive quadrant for a unit circle inside a square with an area of 1 to estimate the value of π. The script should calculate the estimate using random points selected from x = 0 to x = 1 and y = 0 to y =1. It should prompt the user for the number of samples and output the value for π and the standard deviation of the measurement. Report the value and standard deviation for sample sizes from 1.0E4 to 1.0E10 and plot the results compared to the known value.
Solution: Open a new Jupyter Notebook with the Julia kernel. Use the procedures outlined in Chapter 2 to develop the following input:

Estimate π Using Monte Carlo

This Notebook demonstrates how we can estimate the value of π using a random number generator and the area on a quadrant of a unit circle inside a square with area of 1.0. The user supplies the number of trials.

Set initial conditions for counters

```
sumsq = 0.0
npi = 0
k = 0
```

Start by requesting the number of trials

```
println("Enter the number of trials:\n\n")
n_trials = readline()
n_trials = parse(Float64, n_trials)
```

Count how many X,Y pairs end up inside the circle

```
while k <= n_trials
    x=rand()
    y=rand()
    if (x^2+y^2 < 1.0)
        npi+=1
        sumsq+=1
    end
    k+=1
end
```

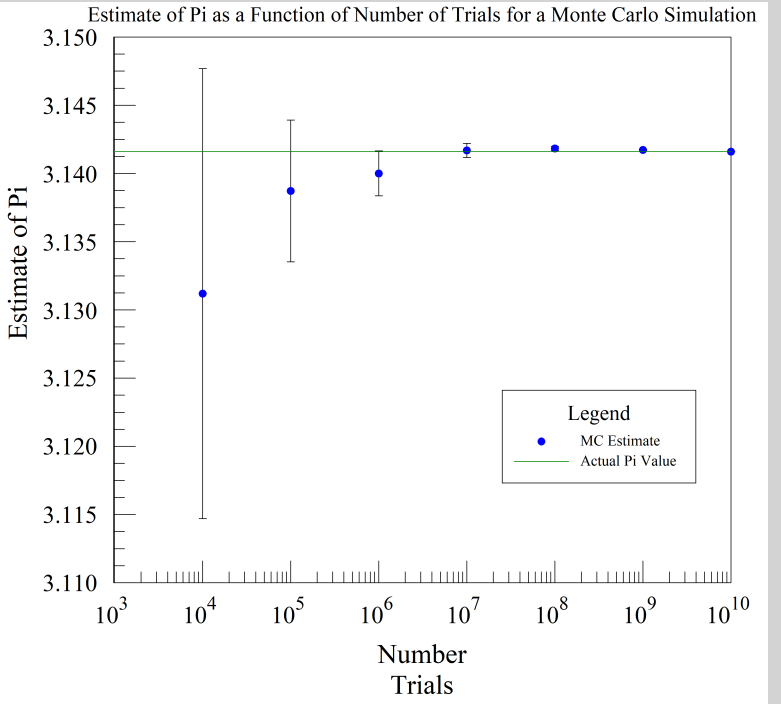
```
Calculate estimate for π

pi_value = 4*(npi/n_trials)

Estimate the standard deviation

standev = sqrt((sumsq/n_trials - pi_value)^2/n_trials)

Report final results

println("Using ", n_trials, " trials, the final estimate for ", pi, " is ", pi_value,
        " with a standard deviation of +- ", standev)
```

Note that the relationship x=rand() uses an internal Julia function to assign a random number between 0 and 1 to the x variable. The program will prompt for the number of trials and output the value of π and its standard deviation. The results for the trials from 1.0E4 to 1.0E10 are as follows:

Number of Trials	Estimate of π	Standard Deviation
1.00E+04	3.1312	1.6494E-02
1.00E+05	3.13872	5.1993E-03
1.00E+06	3.140008	1.6433E-03
1.00E+07	3.141692	5.1928E-04
1.00E+08	3.14183408	1.6420E-04
1.00E+09	3.141727868	5.1927E-05
1.00E+10	3.141605163	1.6422E-05

If we plot the results and compare them to the known value of π and use the standard deviation as the error bars we get the following graph.

The results show how the number of trials increases the accuracy of the value and greatly reduces the error associated with the value. As the number of trials approaches 1.0E8, the error is so small, it cannot be seen beyond the size of the marker on the graph. Note that even at 1.0E4, the accepted value is within the error bar. It is also noted that the calculation time for 1.0E9 and 1.0E10 was significantly longer than for the lower order calculations.



4.3 Precision, Accuracy, and Uncertainty in Monte Carlo

Experimental results are subject to statistical uncertainty associated with the measured value. Because Monte Carlo techniques attempt to model actual random processes, they are also subject to the same uncertainty as experimental measurements. Unlike deterministic transport code results, Monte Carlo results should ALWAYS be reported with some measurement of the uncertainty included. As we saw in the figure in Example 4.1, the error associated with a Monte Carlo calculation is very sensitive to the number of trials performed.

Figure 4.5 is a depiction of the concept of the terms precision and accuracy which are used to describe our experimental and Monte Carlo results. A precise measurement has a small overall standard deviation while an accurate measurement is close to the expected or known value. A measurement can be either precise or accurate or, ideally, both.

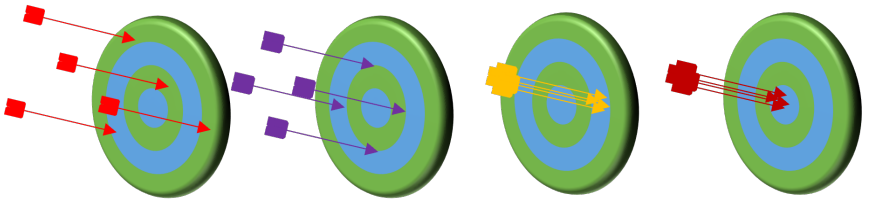


Figure 4.5: A graphical description of precision and accuracy as represented by arrows striking a target. The leftmost target represents a group of arrows that are neither precise (close together) or accurate (close to the bullseye). The second target is accurate (evenly spaced around bullseye) but not precise. The third is precise (close together) but not accurate and the fourth target is both precise and accurate.

As we saw earlier, we can measure the precision of our Monte Carlo calculations by calculating the standard deviation of the mean. In its general form, the standard deviation, σ, is given by Equation 4.9.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{4.9}$$

Where N is the total number of trials, x is each measured value, and μ is the mean or average of all the measured values. From Equation 4.9, it is readily apparent the standard deviation will decrease as the number of trials increases which was observed in our Monte Carlo estimate of π in Example 4.1. If we have sufficiently large populations, the difference between the average and measured values will be small and we can simplify Equation 4.9 to and approximation given by Equation 4.10.

$$\sigma = \sqrt{\frac{1}{N}} \tag{4.10}$$

If we make a computer program to perform a Monte Carlo calculation, we should include a routine to calculate the standard deviation. Fortunately, MCNP6.2 always performs a statistical analysis for the reported values. Unfortunately, users will often overlook the statistical analysis or report outcomes without the included variances. It is possible to have a model in MCNP that will give a reported value with a variance that is on the same order or even greater than the values such as $k_{eff} = 1.25 \pm 1.50$. This value is, of course, meaningless. It is up to the user to analyze the output and adjust the model to reduce variances and ensure precision in the reported results. When a model is not precise, the dominate form of error is random. The simplest method to reduce random error in a Monte Carlo model is to increase the sample size. In certain cases, this method may be insufficient and other methods, known as variance reduction techniques, may have to be applied. We will discuss some

variance reduction techniques available in MCNP later in the text.

Accuracy in Monte Carlo modeling is another potential issue. The overall model accuracy depends strongly on the foundation of the random walk process – the cross-sectional data. It is possible to have an excellent geometrical representation of a radiation transport model with details that exactly match the source, dimensions and materials and still achieve terrible results with the use of bad cross-sectional data. As we discovered when we plotted the cross sections for ^{235}U at different temperatures and when we performed the estimate for the fuel temperature coefficient using SCALE, some cross sections are very temperature dependent. In addition to temperature, other factors may affect how the model simulates the environment. For example, if we mix hydrogen and oxygen in MCNP in the ratio and density of water, we will not get the same effect as if we specify the cross-section, is water. The reason for the difference is due to the rotational and vibrational states in the water molecule that affect how neutrons interact. The molecular cross section is not the same as the free-gas cross section. Again, it is up to the user to understand the output and to correctly account for any assumptions or simplifications to achieve results that are both precise and accurate.

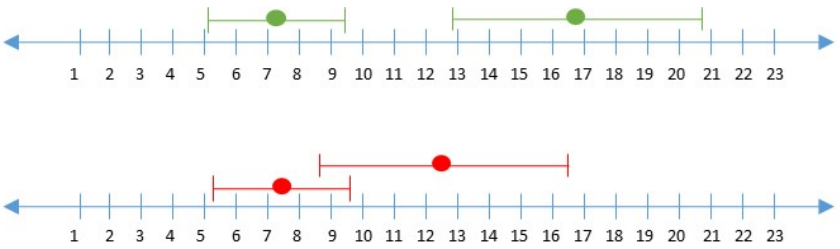


Figure 4.6: An example of confidence intervals on number lines. Because the two green intervals do not overlap, the dominant form of error is systemic. For the red intervals, the dominant error is random.

When a measurement or model is inaccurate, the confidence intervals of the accepted and experimental results do not overlap. The dominate error associated with an inaccurate measurement is systematic. It is possible that the materials or geometry in an experiment were not those used in a model. Another example might be that the intrinsic detector efficiency was not the same as what was modeled. If systemic errors can be reasonably explained, the model can either be refined or the results may be useful.

4.4 Simple Monte Carlo Transport Example

Let us examine a simple gamma shielding problem to gain a better understanding of how Monte Carlo can be applied to a radiation transport problem. Figure 4.7 is an illustration of a simple shield for a beam of mono-energetic γ-rays.

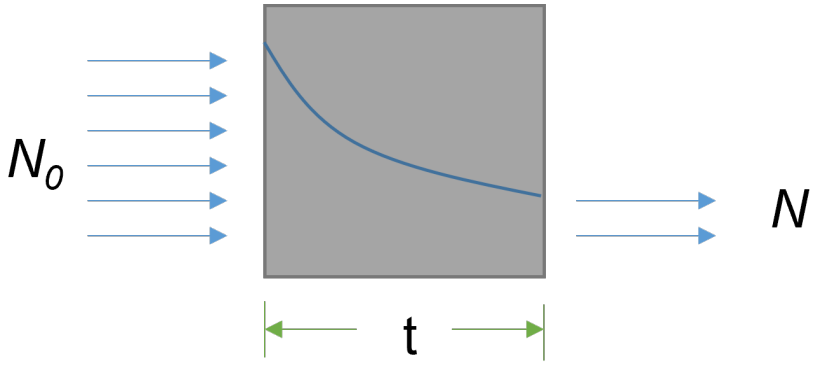


Figure 4.7: Simple shielding problem of mono-energetic beam of γ-rays impacting a shield of thickness, t.

As mentioned in Chapter 1, the total probability of interaction for gamma rays per unit path length is given by the linear attenuation coefficient, μ. If we analyze the number of the gamma rays that are not attenuated, N, in a shield of thickness, t, as a function of the initial number of gamma rays, N_0, and the linear attenuation coefficient, we get the deterministic negative exponential function shown in Equation 4.11 which is in the same form as the basic equation for activity presented in Equation 1.1.

$$N = N_0 e^{-\mu t} \tag{4.11}$$

Using the definition of the linear attenuation coefficient, we know that the mean free path for a gamma ray is:

$$\lambda = \frac{1}{\mu} \tag{4.12}$$

Equation 4.11 represents the deterministic solution to the transport of uncollided gamma rays through our shield. We can perform the same calculation using the probability of interaction, μ, and a set of random numbers to represent our particles to describe the distance the particle travels in our material. The mean free path, Equation 4.12, is the average distance a gamma will travel through a material prior to having an interaction and is the inverse of the attenuation coefficient. This provides avergage distance traveled, but we require a general solution. The probability density function (PDF) for our shield is

probability that the first interaction of an incident gamma will take place at a depth between x and $x + dx$ is given by Equation 4.13.

$$P(x)dx = P(x)\mu dx \tag{4.13}$$

If we integrate the probability function we get the cumulative probability. We can use the cumulative probability of interaction and set that equal to a random number (R_1) between 1 and 0. The result is presented in Equation 4.14.

$$P_c(x) = 1 - e^{-\mu x} = R_1 \tag{4.14}$$

Solving for x (the location of the first collision) we get Equation 4.15.

$$x = -\frac{1}{\mu}\ln(1 - R_1) \tag{4.15}$$

We can now write a program that uses a random number to get a location of the first collision. We solve for the number of particles that transmit through our shield, N, using a conditional loop, to tally the number of particles that have an uncollided distance, x, beyond the thickness of the shield. If we use enough particles, the fraction of particles passing through the shield, N/N_0 should compare favorably to the deterministic solution from Equation 4.11. We should include a standard deviation for our solution to perform a good statistical analysis of our results.

The same approach outlined above for the basic Monte Carlo calculation of gamma shielding can also be performed for neutron shielding. For neutrons, the total macroscopic cross section, Σ, is used in place of the total attenuation coefficient, μ. In Example 4.2, a different simple Monte Carlo solution is presented where the remaining activity of a radioactive source is calculated and compared to the deterministic results in Equation 1.1.

Example 4.2
Given the half-life for a radioisotope, write a Monte Carlo program in Julia to estimate the percentage of remaining number of atoms, activity, or mass for a radioactive isotope after a given period of time, t. The user will provide the half-life of the source, elapsed time, and the number of particles to sample for the Monte Carlo solution. Using ^{60}Co decayed for 2 years as a test case, compare the results for sample sizes of 100, 1000, 1×10^5, and 1×10^6 source particles to the deterministic solution given by Equation 1.1.

Solution: The percentage of radioactive atoms that have not decayed is equivalent to the activity in the material and its mass. This program will allow the user to multiply the results by a starting number of atoms, activity, or mass

and determine the remainder. Our solution depends on the decay constant, λ, for the isotope which is:

$$\lambda = \frac{\ln 2}{T_{1/2}}$$

To begin the Monte Carlo solution we first develop the probability density function (PDF). The PDF for our decay problem is probability that the day will take place at a time between t and $t + dt$ is given by:

$$P(t)dt = P(t)\lambda dt$$

If we integrate the probability function we get the cumulative probability. We can use the cumulative probability of decay and set that equal to a random number (R_1) between 1 and 0.

$$P_c(t) = 1 - e^{-\lambda t} = R_1$$

Solving for t (the time for decay) we get:

$$t = -\frac{1}{\lambda}\ln(1 - R_1)$$

Using the approximation given by Equation 4.10, the standard deviation is given by:

$$\sigma = \sqrt{\frac{1}{N}}$$

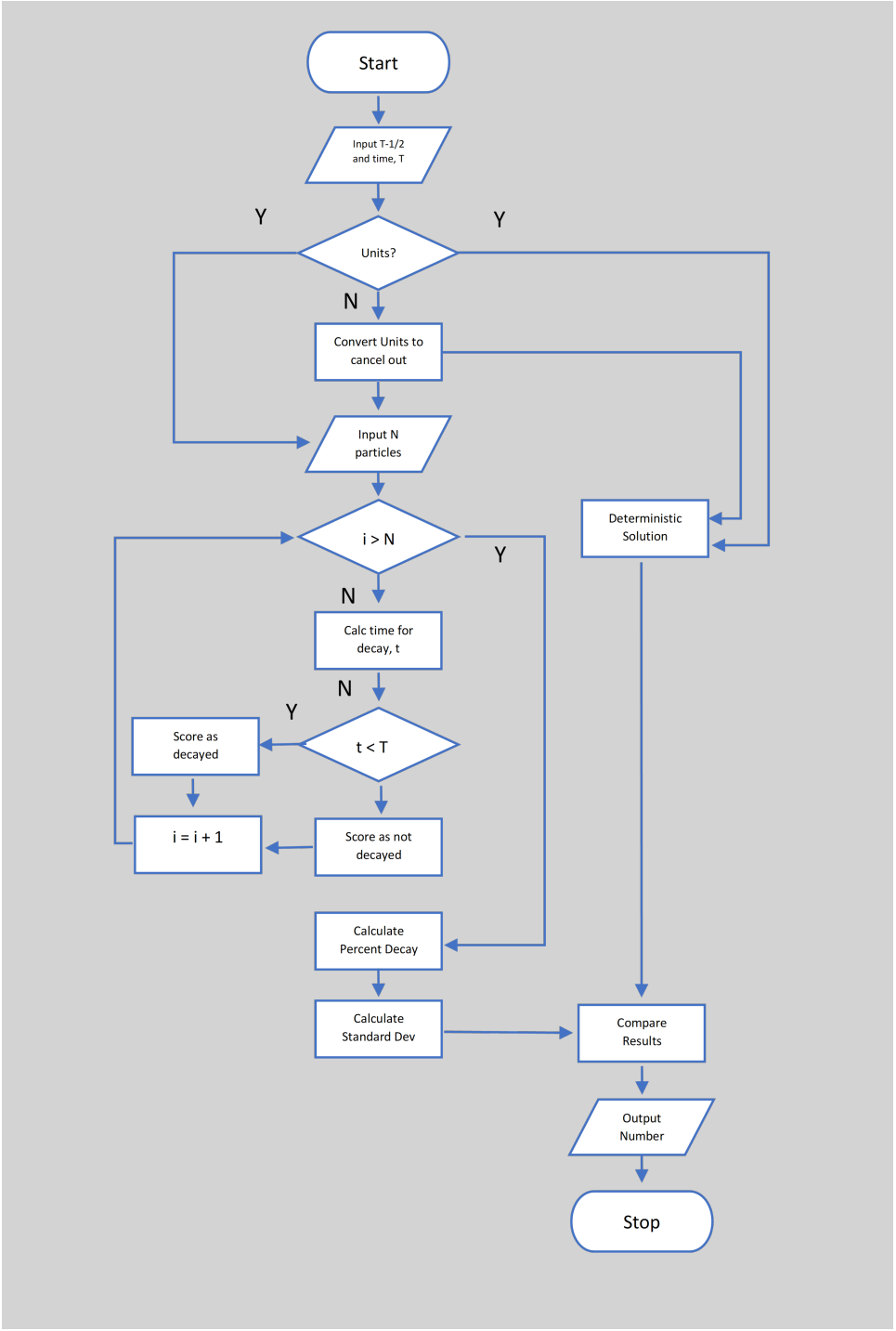
Now that we have the governing equation for the Monte Carlo solution, we will use the steps outlined in Chapter 2 for the development of a computer code to write the solution.

1. *Map out the algorithm steps using a logical flowchart.*

The following flowchart includes an extra check in the beginning of the code that verifies the units for the elapsed time and the time entered for the half-life are the same. This is essential as the units must cancel out. By using a conditional, the code will check the units and then put everything into seconds. We could have chosen any unit of time as it will cancel in the calculation.

In addition to the conditional on the entry, the code requires a loop to iterate each particle. In this process, the code will select a random number and then test it in the equation we solved for t above to determine if it is longer than the elapsed time for the test.

The parallel process in the code will use the deterministic equation derived from Equation 1.1 to calculate the value for the percent decayed. This will be used in the final process that will compare the Monte Carlo and deterministic solutions to each other.


Start
Input T-1/2 and time, T
Y
Y
Units?
N
Convert Units to cancel out
Input N particles
i > N
Y
N
Deterministic Solution
Calc time for decay, t
N
Y
Score as decayed
t < T
i = i + 1
Score as not decayed
Calculate Percent Decay
Calculate Standard Dev
Compare Results
Output Number
Stop

2. *Write a meaningful header with appropriate comments.* Open Visual Studio Code and create a new file named "ActivityMC.jl" and start with comments and include the packages necessary for program.

```
#|Function to estimate the percentage of undecayed mass or activity |
#|of a radioisotope using a Monte Carlo approach and comparing      |
#|to a deterministic solution                                       |
#|                                                                  |
#|Written by: Kenneth Allen 18 January 2023                         |
#|------------------------------------------------------------------|
# Load Modules necessary
using IJulia
using Printf
# Set initial conditions for counters
#------------------------------------------------------------------|
```

3. *Declare constants and variables.* This problem uses three counters within the Monte Carlo conditional statement. One for the while loop counter, and two for the tallies for the particles that have decayed. These counters are iterated from an initial condition of 0.0 specified here. Because the final standard deviation calculation uses the approximation presented in Equation 4.10, the sumsq value does not contribute to the final results and could be eliminated from the program entirely.

```
# Set initial conditions for counters
#------------------------------------------------------------------|
sumsq = 0.0;
npi = 0;
k = 0;
# NOTE using iJulia, can assign Greek letters as variables by typing
# a \ followed by the phonetic name and then ctrl+space  such as
#   \lambda
```

4. *Write a routine to receive input.* For this program, the user will specify the half-life and its units, the elapsed time and its units, and the number of trials for the Monte Carlo estimate calculation.

```
#           Write a routine to request input
#-------------------------------------------------------------------|
# Request the half-life of the radioisotope
println("Enter the half-life of the radioistope:\n\n")
halflife = readline()
halflife = parse(Float64, halflife)

#Request the units for the half-life
println("Enter the units for half-life as s, m, h, d or y:\n\n")
units = readline()

# Request the elapsed time for decay
println("Enter the elapsed time of decay:\n\n")
T = readline()
T = parse(Float64, T)

#Request the units for the elapsed time
println("Enter the units for elapsed time as s, m, h, d or y:\n\n")
units_T = readline()

# Request the number of trials for the Monte Carlo calculation
println("Enter the number of trials:\n\n")
n_trials = readline()
n_trials = parse(Float64, n_trials)
```

5. *Write routines and subroutines to execute desired calculations.* Referring to the flowchart, this section begins with a request for two times followed by two conditional **if**, **elseif**, and **else** routine to determine the units for each and to covert each into seconds. The program then calculates the decay constant and uses a while loop to perform the Monte Carlo estimate. When the while loop finishes, it uses Equation 4.10 to estimate the standard deviation and then calculates the deterministic solution and qualitatively compares the two results.

```
#  Write routines and subroutines to calculate results
#-------------------------------------------------------------------|
# Check and convert half-life to seconds for  calc
if (units == "s")
    halflifec = halflife
elseif (units == "m")
    halflifec = halflife*60
elseif(units == "h")
    halflifec = halflife*3600
elseif(units == "d");
    halflifec = halflife*3600*24
elseif(units == "y");
    halflifec = halflife*3600*24*365
end # end if conditional
```

```
#Calculate decay constant in inverse units of time
λ = (log(2))/halflifec

# Check and convert elapsed time to seconds for calc
if (units_T == "s")
    Tc = T
elseif (units_T == "m")
    Tc = T*60
elseif(units_T == "h")
    Tc = T*3600
elseif(units_T == "d");
    Tc = T*3600*24
elseif(units_T == "y");
    Tc = T*3600*24*365
end # end if conditional

#Set up while loop to calculate number of atoms
while k <= n_trials
    R = rand()
    t=(-1/λ)*log(1-R)
    if(t>=Tc)
      global npi=npi+1
      global sumsq+=1
    end
    global k+=1
end

#Calculate the Monte Carlo Estimate for percent decayed
percentMC = npi/n_trials

#Calcualte standard deviation of Monte Carlo estimate
standev =  Float64(sqrt(1/n_trials))

#Calcualte the deterministic solution for percent decayed
percentDET = exp(-(λ)*Tc)

#Calcuate difference between solutions
diff = 100 * abs(percentMC-percentDET)/percentDET
```

6. *Write routine that handles output.* This program writes the results to the screen. It uses the @print function to limit the number of decimal points in the float numbers to 4 decimals using "%.4f". This makes the results easier to read for the user.

```
# Write a routine that handles output
#-------------------------------------------------------------------------|
#declare funtction that represents decay over time
println("Using ", n_trials,
" trials, the final estimate for the percent decayed is ")
@printf("%.4f", percentMC)
print("\n")
println(" with a standard deviation of +- ")
@printf("%.4f", standev)
print("\n")
println("The deterministic solution is ")
@printf("%.4f",percentDET)
print("\n")
println("The difference between the two values is ")
@printf("%.4f", diff)
print(" %")
```

Now that the entire program, MCActivity.jl, is written, it can be executed. To run the program within Visual Studio Code, use ALT+J+O to bring up the REPL terminal within Visual Studio Code. At the Julia> prompt enter **include("MCActivity.jl")** which will start the program. HINT: when writing the filename in the command line, start with **include("M** and then press the tab key. Julia will recognize the file you intend to access and will auto-complete the line. When the file opens answer the prompts. For this example, the entries for ^{60}Co are half-life of 1925.28 days and elapsed time of 2 years with the number of trials specified in the problem statement.

N Trials	Fraction Decayed	Standard Deviation	Percent Difference
100	0.800	0.1000	4.0472%
1000	0.7940	0.0316	3.2668%
1E5	0.7677	0.0032	0.1576%
1E6	0.7688	0.0010	0.0113%

The results from the program support the accuracy of the Monte Carlo estimate and the trend of the reduction of the Standard Deviation and the convergence between the solution as the number of trials increased. Because of the simplicity of the program, there is not a noticeable difference in computational time between 100 and 1E6 trial.

Bibliography

Dupree, S. A. and Fraley, S. K. (2002). *A Monte Carlo Primer : a Practical Approach to Radiation Transport*, Kluwer Academic/Plenum, New York.

Lewis, E. E. and Miller, W. F. (1993). *Computational Methods of Neutron Transport*, American Nuclear Society.

4.5 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- Use PowerPoint to Create Number Line Figure

4.6 Problems

1. Describe how a random walk process could be applied to create a simulation of the rate at which motor vehicles will have an accident. Consider factors such as speed, weather conditions, driver experience etc.

2. A Monte Carlo calculation resulted in the following dataset: 9, 2, 5, 4, 12, 7, 8, 11, 9, 3, 7, 4, 12, 5, 4, 10, 9, 6, 9, 4. Calculate the mean of the dataset.

3. Calculate the standard deviation for the dataset in problem 2 and report the results in a confidence interval.

4. If the accepted value for the measurement in Problem 2 is 7.15 ± 0.21, is the Monte Carlo calculation in Problem 3 accurate, precise, neither or both?

5. What is the dominate form of error in the model in Problem 2? What could you do to reduce the error?

6. Draw the PDF for the cosine of the polar angle (μ) associated with the isotropic emission of a fission neutron.

7. Using the PDF from Problem 6, calculate the probability a neutron will be emitted into $\mu = 0.5$.

8. Write a Julia program that uses Monte Carlo sampling and the PDF from Problem 6 to estimate the fraction of fission neutrons emitted into $\mu = 0.5$.

9. Given the exponential PDF for gamma attenuation through lead, write a Julia program that uses a Monte Carlo sampling technique to estimate the percent of source particles from 137Cs that are not attenuated and are transmitted through a 2 cm thick shield of lead.

10. Calculate the standard deviation associated with the estimate in Problem 9 and incorporate the calculation into the Julia program.

Chapter 5

Introduction to MCNP 6.2

While MCNP is recognized as one of the most powerful and important Monte Carlo-based radiation transport codes, students often find it challenging to learn and use. In most cases, modern students have little to no experience using computer programs that do not have a graphical user interface (GUI) and do not use a mouse or other pointer device. The input for MCNP has remained essentially unchanged from when it was first designed for input using punch cards. In fact, the input lines are still referred to as cards and the total file is called a deck from the days of punch card use.

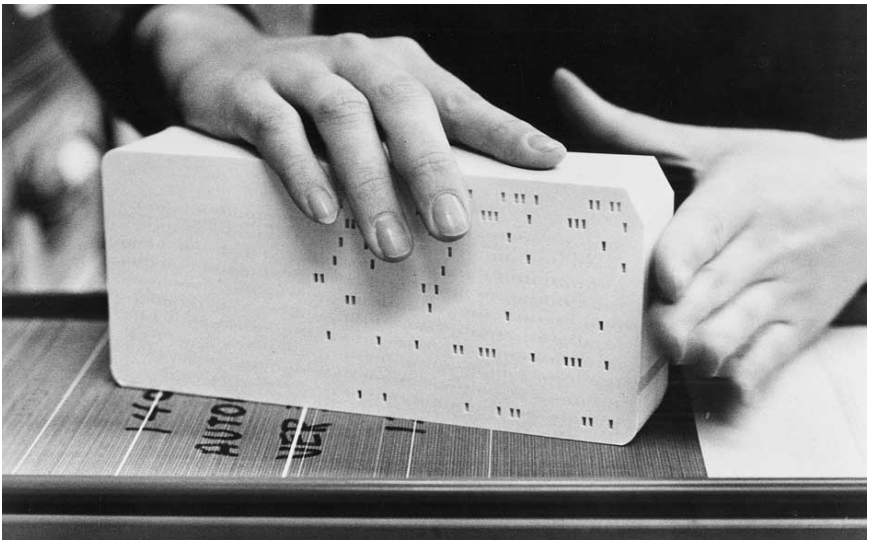


Figure 5.1: Image of a deck of computer punch cards. (IBM; 2016)

For a generation of students who have never had to boot a computer from a DOS prompt, the input file made of lines on a text editor can be frustrating

and considered archaic. However, when students eventually learn how to do the input for MCNP, they understand how the free format allows the flexibility to model almost any conceivable transport problem. In fact, MCNP has been used to model everything from complex structures like nuclear reactors to even the human body.

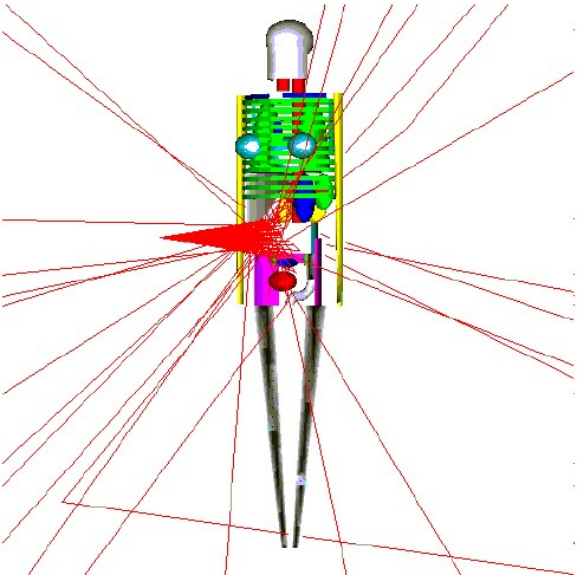


Figure 5.2: MCNP model of human in an x-ray plotted in SABRINA (Van Riper; 1996)

For years students have been introduced to using MCNP through the book *An MCNP Primer by Shultis and Faw.* (Shultis; 2011) The primer is an excellent introduction to the code for first-time users. It was last updated for MCNP 5. This text uses much of the information from Shultis and Faw's primer with some updates for MCNP 6.2 to include the depletion, or BURN, card that was added when MCNP 5 was combined with MCNPX to form MCNP 6. In addition to depletion, MCNPX was designed to transport charged particles as well as neutrons and photons. When the two versions were combined, MCNP 6 gained those features as well.

5.1 General Rules for MCNP Input

The MCNP input file is written in a free field format in a text editor. Free format indicates that, after the card name, the number of spaces between entries is not fixed. This text uses TextPad® as the text editor of choice because some of its advanced features, such as block select mode and macros are not available on some other text editors, like Notepad®. You should never use a word pro-

cessor to try and write the input for MCNP. A word processor, like Microsoft Word®, will put unseen formatting characters in the input file that will cause the code to fail. Even within a plain text editor, you have to be careful to not use characters other than numbers and letters, returns and spaces. You cannot use the tab key for an MCNP input file.

The input file is broken into the mandatory structure presented in Table 5.1. If you do not follow the structure exactly, the input will fail to execute. The blank lines act as deliminators and are necessary and you can only have one wherever specified. This is unlike SCALE and Julia which will allow for multiple blank lines in the input. The deliminators tell the code to look for the next section (i.e. surface cards or data cards). If you include an additional blank line the code will terminate the reading of the input file early and provide an error that may not make sense to the user such as "no material card present." This can be frustrating to the user who knows there is a material card in the input and does not recognize that the second blank line indicates to MCNP the data card block has started and any input past a third blank line is not read. The order for the blocks is also important. For example, you cannot put the surface cards in the cell card block.

Table 5.1: MCNP Input File Structure.

Description	Comments
`Problem Title Card`	This title is printed in the output
`Cell Cards`	
	One blank line deliminator
`Surface Cards`	
	One blank line deliminator
`Data Cards`	
	One blank line terminator (optional)

In addition to the restrictions on the blank lines, the input for MCNP is limited to a maximum line length of 80 characters. This is because the code was designed to be used with the 80-column punch card introduced by IBM in 1928. (IBM; 2016) To ensure you do not exceed 80 characters in TextPad®, look at the lower right corner of the window and you will see two white numbers that change whenever you move the cursor through the file. The first is the row number and the second is the column number. Also the first 5 characters in the line are reserved for the card name for the input command. If a line of input text has to be longer than 80 characters, it can continue to the next line as long as there are 5 spaces in the first 5 columns. For example, the following material card has input that wraps onto two lines:

```
M2     92235.70C -4.10779E+00 92238.70C -8.40421623E+01
       8016.70C -1.18500E+01 $UO2 4.66% EN at 293K
```

As we outlined in the procedure for writing Julia scripts, it is important to include meaningful comments within the input file for an MCNP file. Like how in Julia the number or hashtag (#) symbol and in SCALE the apostrophe (') indicate comments, the letter c (upper or lowercase) followed by a space to lead a line indicates a comment in MCNP. Also, the dollar sign ($) in an input line indicates that everything that follows is a comment. Note, you must have a space after the letter c or MCNP will think that the comment is a card name.

In general, an input line in MCNP will consist of the card name, mnemonic, in the first 5 columns followed by parameters associated with that card, known as cell entries. The entries are normally numerical. The standard units used in MCNP are listed in Table 5.2.

Table 5.2: Standard Units in MCNP

Measurement	**Unit**
Length	cm
Energy	MeV
Time	shakes (10^{-8} s)
Temperature	MeV (kT)
Mass density	g/cm^3
Cross section	barns (10^{-24} cm^2)

5.2 Building the Input Deck–Define Materials

While at first it may seem to make sense to begin by starting at the top of the input deck format described in Table 5.1, it is not possible. To fully describe each of the cells in the geometry, you first must know the materials and the surfaces that make up those cells. As a technique, we will start our MCNP input deck by defining the materials first.

The material card is one of the cards in the Data Cards section of the input. Each material card represents a unique substance in the model geometry. Once defined, that material can be used in multiple portions of the model. The density of the material is defined in the cell card. Therefore, the only time you would have to redefine a material with a given isotopic composition is if it is used at different temperatures in the same model. Materials are defined by the isotopes that make them up in either their atom percent or weight percent. The identification for the isotope is the ZAID number, as was used in SCALE, followed by the cross-section library identification. The complete list of isotopes and libraries that are part of the MCNP code suite can be found in APPENDIX G – MCNP Data Libraries in the MCNP User's Manual. (Team; 2008) An example for hydrogen neutron cross sections from APPENDIX G is

presented in Figure 5.3 from the MCNP5 manual.

Appendix G - MCNP Data Libraries
Neutron Cross-Section Libraries

Table G.2 - Continuous-Energy and Discrete Neutron Data Libraries Maintained by X-1 NAD

ZAID	AWR	Library Name	Source	Eval Date	Temp (K)	Length Words	NE	Emax (MeV)	GPD	Nu	CP	DN	UR
Z = 1 ********** Hydrogen **********													
** H-1 **													
1001.24c	0.999	la150n	B-VI.6	1998	293.6	10106	686	150.0	yes	no	yes	no	no
1001.42c	0.999	endl92	LLNL	<1992	300.0	1968	121	30.0	yes	no	no	no	no
1001.50c	0.999	rmccs	B-V.0	1977	293.6	2766	244	20.0	yes	no	no	no	no
1001.50d	0.999	drmccs	B-V.0	1977	293.6	3175	263	20.0	yes	no	no	no	no
1001.53c	0.999	endf5mt[1]	B-V.0	1977	587.2	4001	394	20.0	yes	no	no	no	no
1001.60c	0.999	endf60	B-VI.1	1989	293.6	3484	357	100.0	yes	no	no	no	no
1001.62c	0.999	actia	B-VI.8	1998	293.6	10128	688	150.0	yes	no	yes	no	no
1001.66c	0.999	endf66a	B-VI.6:X	1998	293.6	10128	688	150.0	yes	no	yes	no	no
1001.70c	0.999	endf70a	B-VII.0:x	2005	293.6	8177	590	20.0	yes	no	yes	no	no
1001.71c	0.999	endf70a	B-VII.0:x	2005	600.0	8177	590	20.0	yes	no	yes	no	no
1001.72c	0.999	endf70a	B-VII.0:x	2005	900.0	8177	590	20.0	yes	no	yes	no	no
1001.73c	0.999	endf70a	B-VII.0:x	2005	1200.0	8177	590	20.0	yes	no	yes	no	no
1001.74c	0.999	endf70a	B-VII.0:x	2005	2500.0	8177	590	20.0	yes	no	yes	no	no

Figure 5.3: Neutron cross section table for Hydrogen from APPENDIX G of the MCNP 5 User's Manual.(Team; 2008)

Note from the example in Figure 5.3, the ZAID has not only the atomic number (Z) and the mass number (A) but also a decimal followed by an extension consisting of a two-digit number and a letter. This extension specifies the neutron cross section library to be used. The cross-section libraries in MCNP include charged particles, discrete energy, and continuous energy neutron libraries to name a few. In general, we will only be using the continuous energy neutron libraries which are annotated with a "C" at the end of their ZAID. The two numbers indicate the specific Evaluated Data File the user wants to use. In general, the user should specify the newest data file available (ENDF 7) because it has the most current measurements. It is also very important to specify the library that corresponds to the temperature for the material in the model. For ENDF7, the *.70c library is at room temperature while the *.71c, *.72c, *.73c, and *.74c libraries are at 600, 900, 1200 and 2500 kelvin respectively. If the user needs more specific temperatures, they can use NJOY99 to create cross sections at other temperatures and add them to the MCNP library. The custom creation of cross-sections is not covered in this text. If the cross-section library extension is not added to the material ZAID, MCNP will pick the most recent cross section by default. Because the temperature effect on some cross sections can be important, it is better to specify the library being used when performing neutron calculations. It is ok to specify the neutron cross section library for a photon-only problem. MCNP will just give a warning and ignore the neutron table.

When specifying a material, write the designation "M" followed by a unique

number for that material (i.e. M1) in the first 5 columns of the line followed by the ZAID for each isotope, and the atom fraction or mass fraction for that isotope. For example:

```
M1      8016.70C 1 1001.70C 2 $ room temp water by atom fraction
```

This line represents material 1 in the atomic fraction of H_2O.

```
M1      8016.70C -8.88100E+01
        1001.60C -1.11900E+01 $ room temp water by mass fraction
```

This line represents material 1 in the mass fraction of H_2O. Materials do not have to be numbered in order within the input deck. They just require a unique number identifier. Notice how the negative sign before the fraction denotes mass versus atom fraction in the compound. Before we move on to another material example, it is important to discuss the treatment of water, especially in thermal reactors. Hydrogen when bound in water has different scattering cross sections than the free gas model of hydrogen and oxygen in the ratio of water presented in the previous example. It is important to add a special treatment card (MT) to the input that accounts for the scattering of hydrogen in water. APPENDIX G has a tabular list of all of the .XXt cross section libraries that account for the scatter of neutrons off of water and other important compounds that have unique binding effects. A correct input example for water at room temperature for neutrons would be:

```
M1      8016.70C 1 1001.70C 2 $ room temp water by atom fraction
MT1     lwtr.10t $ light water at room temperature
```

The number 1 following MT indicates that the treatment is applied to material 1. Another example of an input is the one we presented earlier for UO2:

```
M2      92235.70C -4.10779E+00 92238.70C -8.40421623E+01
        8016.70C -1.18500E+01 $UO2 4.66% EN at 293K
```

If the mass fractions for all the isotopes in a given material do not add up to 1.0, MCNP will normalize the input. If you know the isotopic concentration, atom or mass fractions of each, and the density of the material, you can model anything. Often the challenge is finding out what isotopes are in a particular material.

There is an excellent reference published by the Pacific Northwest National Laboratory called *Compendium of Material Composition Data for Radiation Transport Modeling.* (Detwiler et al.; 2021) It is publicly available at: PNL Compendium This reference has the isotopic compositions for thousands of materials. For example, it contains references for materials like concrete, different rubber compounds, plastics, and wood. The isotopic concentrations are

listed with their ZAID numbers and the atom fractions, weight fractions and atom densities. The reference also provides the physical density of the material which will be needed in the cell description.

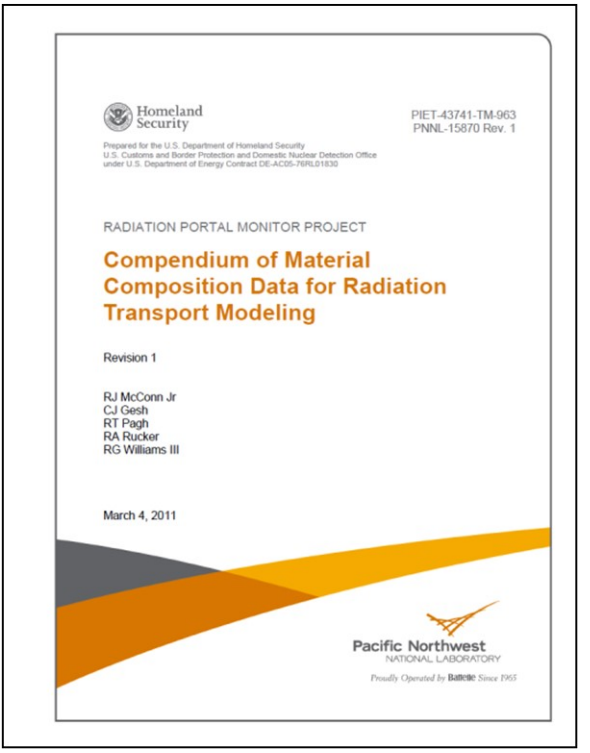


Figure 5.4: *The Compendium of Material Composition Data for Radiation Transport Modeling* is an excellent resource for material compositions modeled in MCNP.(Detwiler et al.; 2021)

5.3 Building the Input Deck–Surface Cards

The geometry for MCNP consists of cells which are built up of surfaces and materials. Once all of the materials are created, we can begin working on the surface cards. Surfaces are standard geometric shapes defined in MCNP arranged in a Cartesian coordinate system (x, y, z). All the units for surfaces are in cm. Tables 5.3 and 5.4 are from the MCNP6 User's Manual and define most of the surfaces used in the program.(Kulesza et al.; 2022)

Table 5.3: MCNP Surface Cards (Kulesza et al.; 2022)

Name	Type	Description	Equation	Card Entries
P	Plane	General	$Ax + By + Cz - D = 0$	A B C D
PX		Normal to x-axis	$x - D = 0$	D
PY		Normal to y-axis	$y - D = 0$	D
PZ		Normal to z-axis	$z - D = 0$	D
SO	Sphere	Centered at Origin	$x^2 + y^2 + z^2 - R^2 = 0$	R
S		General	$(x-\bar{x})^2 + (y-\bar{y})^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{x}$ $\bar{y}$ $\bar{z}$ R
SX		Centered on x-axis	$(x-\bar{x})^2 + y^2 + z^2 - R^2 = 0$	$\bar{x}$ R
SY		Centered on y-axis	$x^2 + (y-\bar{y})^2 + z^2 - R^2 = 0$	$\bar{y}$ R
SZ		Centered on z-axis	$x^2 + y^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{z}$ R
C/X	Cylinder	Parallel to x-axis	$(y-\bar{y})^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{y}$ $\bar{z}$ R
C/Y		Parallel to y-axis	$(x-\bar{x})^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{x}$ $\bar{z}$ R
C/Z		Parallel to z-axis	$(x-\bar{x})^2 + (y-\bar{y})^2 - R^2 = 0$	$\bar{x}$ $\bar{y}$ R
CX		On x-axis	$x^2 + y^2 - R^2 = 0$	R
CY		On y-axis	$x^2 + z^2 - R^2 = 0$	R
CZ		On z-axis	$y^2 + z^2 - R^2 = 0$	R
K/X	Cone	Parallel to x-axis	$\sqrt{(y-\bar{y})^2 + (z-\bar{z})^2} - t(x-\bar{x}) = 0$	$\bar{x}$ $\bar{y}$ $\bar{z}$ t^2 ± 1
K/Y		Parallel to y-axis	$\sqrt{(x-\bar{x})^2 + (z-\bar{z})^2} - t(y-\bar{y}) = 0$	$\bar{x}$ $\bar{y}$ $\bar{z}$ t^2 ± 1
K/Z		Parallel to z-axis	$\sqrt{(x-\bar{x})^2 + (y-\bar{y})^2} - t(z-\bar{z}) = 0$	$\bar{x}$ $\bar{y}$ $\bar{z}$ t^2 ± 1
KX		On x-axis	$\sqrt{y^2 + z^2} - t(x-\bar{x}) = 0$	$\bar{x}$ t^2 ± 1
KY		On y-axis	$\sqrt{x^2 + z^2} - t(y-\bar{y}) = 0$	$\bar{y}$ t^2 ± 1
KZ		On z-axis	$\sqrt{x^2 + y^2} - t(z-\bar{z}) = 0$	$\bar{z}$ t^2 ± 1 ± 1 used only for 1 sheet cone
SQ	Ellipsoid Hyperbolioid Paraboloid	Axes parallel To x-, y-, or z-axis	$A(x-\bar{x})^2 + B(y-\bar{y})^2 + C(z-\bar{z})^2 + 2D(x-\bar{x}) + 2E(y-\bar{y}) + 2E(z-\bar{z}) + G = 0$	A B C D E F G $\bar{x}$ $\bar{y}$ $\bar{z}$
GQ	Cylinder Cone Ellipsoid Hyperbolioid Paraboloid	Axes not parallel To x-, y-, or z-axis	$Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Eyz + Fzx + Gx + Hy + Jz + K = 0$	A B C D E F G H J K
TX	Elliptical or Circular Torus	Axes not parallel To x-, y-, or z-axis	$(x-\bar{x})^2/B^2 + \left(\sqrt{(y-\bar{y})^2 + (z-\bar{z})^2} - A\right)^2/C^2 - 1 = 0$	$\bar{x}$ $\bar{y}$ $\bar{z}$ A B C
TY			$(y-\bar{y})^2/B^2 + \left(\sqrt{(x-\bar{x})^2 + (z-\bar{z})^2} - A\right)^2/C^2 - 1 = 0$	$\bar{x}$ $\bar{y}$ $\bar{z}$ A B C
TZ			$(z-\bar{z})^2/B^2 + \left(\sqrt{(x-\bar{x})^2 + (y-\bar{y})^2} - A\right)^2/C^2 - 1 = 0$	$\bar{x}$ $\bar{y}$ $\bar{z}$ A B C

Table 5.4: MCNP Surface Cards (Kulesza et al.; 2022)

Macrobody Box (BOX)	
Input Parameter	**Description**
$v_x\ v_y\ v_z$	The x, y, z coordinates of a corner of the box.
$a1_x\ a1_y\ a1_z$	Vector of 1st side from the specified corner coordinates.
$a2_x\ a2_y\ a2_z$	Vector of 2nd side from the specified corner coordinates.
$a3_x\ a3_y\ a3_z$	Vector of 3rd side from the specified corner coordinates.
Macrobody Rectangular Parallelepiped (RPP)	
Input Parameter	**Description**
$x_{min}\ x_{max}$	Termini of box sides normal to the x-axis
$y_{min}\ y_{max}$	Termini of box sides normal to the y-axis
$z_{min}\ z_{max}$	Termini of box sides normal to the z-axis
Macrobody Sphere (SPH)	
Input Parameter	**Description**
$v_x\ v_y\ v_z$	The x, y, z coordinates of the center of the sphere.
r	Radius of sphere.
Macrobody Right Circular Cylinder (RCC)	
Input Parameter	**Description**
$v_x\ v_y\ v_z$	The x, y, z coordinates at the center of the base for the RCC
$h_x\ h_y\ h_z$	Right circular cylinder axis vector, which provides both the orientation and the height of the cylinder.
r	Radius of right circular cylinder.
Macrobody Right Hexagonal Prism (HEX or RHP)	
Input Parameter	**Description**
$v_x\ v_y\ v_z$	The x, y, z coordinates of the bottom of the hexagonal prism.
$h_1\ h_2\ h_3$	Vector from the bottom to the top of the hexagonal prism. For a z-hex with height h, h_1, h_2, and h_3 = 0 0 h
$r_1\ r_2\ r_3$	Vector from the axis to the center of the 1st facet. For a pitch 2p facet normal to y-axis, r_1, r_2, and r_3 = 0 p 0
$s_1\ s_2\ s_3$	Vector to center of the 2nd facet
$t_1\ t_2\ t_3$	Vector to center of the 3rd facet

Table 5.5: MCNP Surface Cards (Kulesza et al.; 2022)

Macrobody Right Elliptical Cylinder (REC)	
Input Parameter	**Description**
$v_x\ v_y\ v_z$	The x, y, z coordinates of the cylinder bottom.
$h_x\ h_y\ h_z$	Cylinder axis height vector.
$v1_x\ v1_y\ v1_z$	Ellipse major axis vector (normal to $h_x\ h_y\ h_z$).
$v2_x\ v2_y\ v2_z$	Ellipse minor axis vector (orthogonal to vectors $\vec{h}$ and $\overrightarrow{v1}$.
Macrobody Truncated Right-Angle Cone (TRC)	
Input Parameter	**Description**
$v_x\ v_y\ v_z$	The x, y, z coordinates of the cone bottom.
$h_x\ h_y\ h_z$	Cone axis height vector.
r_1	Radius of lower cone base.
r_2	Radius of upper cone base, where $r_1 > r_2$.
Macrobody Ellipsoid (ELL)	
Input Parameter	**Description**
$v1_x\ v1_y\ v1_z$	If rm > 0, the coordinates of the 1st foci. If rm < 0, the coordinates of the center of the ellipsoid.
$v2_x\ v2_y\ v2_z$	If rm > 0, the coordinates of the 2nd foci. If rm < 0, major axis vector with length = major radius.
rm	If rm > 0, major radius length. If rm < 0, minor radius length.

When we think of a three-dimensional object that we want to model, we can begin to construct it out of its requisite parts. For example, if we wanted to model a simple rectangular box, we could define it as the intersection of six planes normal to each other, see Figure 5.5. As geometry becomes more complex, we must think about how curved or asymmetrical surfaces intersect with each other. To begin the model, we should first determine the most appropriate place for the origin. It is possible to arrange the surfaces with respect to any origin, but we can save quite a bit of time and frustration by selecting an origin that makes use of any symmetry in the problem.

For the example in Figure 5.5, we can place the origin in the center of the box. If the box has dimensions 10 cm x 20 cm x 10 cm and we wanted to define it using six planes, we would have the following surfaces:

```
1 PX  -5
2 PX   5
3 PY -10
4 PY  10
5 PZ  -5
6 PZ   5
```

Following the input description in Table 5.1, the surface cards follow the single blank line after the cell card blocks. The format for input is a surface identifier (a number in the first 5 columns) followed by the mnemonic (plane normal to an axis) and the input parameter (vector from the origin). We could have chosen to place the origin at the corner of the box or anywhere else in space. However, by placing it in the center, we can make use of the natural symmetry in the problem.

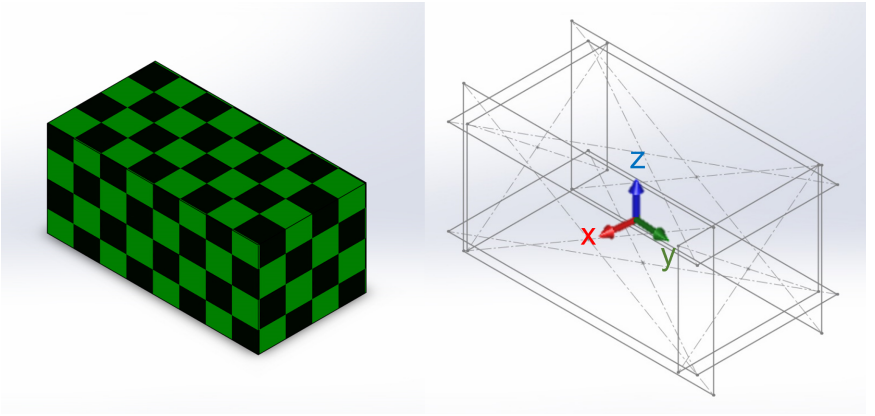


Figure 5.5: A simple rectangular box can be defined as the intersection of six planes normal to each other.

MCNP has an easier way to define commonly shaped objects such as a rectangular box. The same object defined by the six planes above can be defined using the macrobody for a rectangular parallelepiped, RPP. Common macrobodies and their definitions are listed in Table 5.4. To define the same object in Figure 5.5 using a macrobody, the input would be:

```
1 RPP -5 5 -10 10 -5 5
```

Note that the RPP object is only defined when the object is normal to the axis of the origin. If we wanted to define a similar object at some angle to one or more axes, we would have to define it using planes. We gain a lot of efficiency in our model definition if we can use a macrobody to define a surface. The efficiency is not just in the definition of the surface but especially for the definition of the cell.

As we virtually construct our model in three dimensions, we will add surfaces together to create cells. When we add to the model, we should avoid coincident planes. If we have, for example, a cylinder inside a rectangular parallelepiped (RPP), such as the unit cell we modeled in SCALE, we should avoid having the

end of the cylinder match up exactly with the end of the RPP. Because MCNP is a stochastic process, there exists the possibility for a particle to travel exactly along the infinitesimally small border of the junction of two surfaces. This will cause the model to have particles that are "lost" and the run will fail.

All surfaces in MCNP have a "positive" and "negative" side in the 3-dimensional world. For planes, the positive side corresponds to the side where the normal vector points toward the positive axis. For a macrobody, the positive side is oriented outside the body while the negative side is inside. The positive and negative orientation are critical for the definition of cells.

Unless otherwise defined, boundary conditions for surfaces are transparent to particles. If we desired to have a reflected boundary condition, as we did with the unit cell problem in SCALE, the surface card number identifier is preceded by an asterisk (*) for example:

```
*1 RPP -5 5 -10 10 -5 5
```

is the same surface as Figure 5.5 with a reflected boundary condition. There are other boundary conditions available, such as a periodic condition, and they are defined in the User's Manual in Chapter 3. (Kulesza et al.; 2022)

5.4 Building the Input Deck–Cell Cards

Now that we have defined the materials and the surfaces, we can create the cells in the first block of the MCNP input file. As described in Table 5.1, the cell cards immediately follow the title card in the input. The structure for the cell card is (Kulesza et al.; 2022):

1. The cell number, arbitrarily assigned by the user, in the first five columns.

2. The cell material number as described in Section 5.2. The material is described on a material card (M) that has the same material number. If the cell is a void, a zero is entered for the material number. The cell and material numbers cannot exceed eight digits each.

3. Cell material density. A positive entry is interpreted as atom density in units of 1024 atoms/cm3. A negative entry is interpreted as mass density in units of g/cm^3. No density is entered for a void cell.

4. A complete specification of the geometry of the cell. This specification includes a list of the signed surfaces bounding the cell where the sign denotes the sense of the regions defined by the surfaces. The regions are combined with the Boolean intersection and union operators. A space indicates an intersection and a colon indicates a union. Figure 5.6 depicts a graphical example of a union and an intersection.(Shultis; 2011) If

parenthesis are used, the surfaces in the parenthesis are joined first such as (A:B) C.

5. Optionally, after the geometry description, cell parameters can be entered. The form is `KEYWORD = value`. We will use of the `IMP` parameter here for the importance for particles in that cell area. Other parameters, such as volume, could be specified here or, alternatively, in the Data Card block.

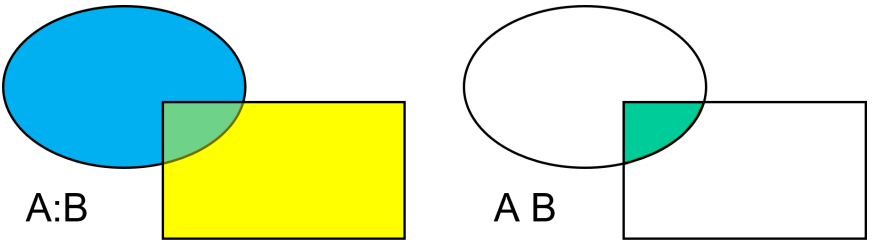


Figure 5.6: Left: the union A:B or "A or B" which includes all the blue, yellow and green areas. Right: the intersection A B or "A and B" which is the green area only.

As an example, the cell card below defines cell 1. Cell 1 is the rectangular cube from Figure 5.5 full of water at nominal density (1.0 g/cm^3) using the material card (M1) defined in section 5.2 and the six planes defined in section 5.3. The cell has an importance for neutrons of 1. This importance weighting indicates that MCNP will count and track 100% of the neutrons in the cell with a weight of 1.0. For an area outside the model, we could set the cell importance to 0 and have the code stop tracking and kill the particles when they leave the box.

```
1 1 -1.0 1 3 5 -2 -4 -6 IMP:N=1
```

Alternatively, we could define the same cell using the macrobody (Section 5.3).

```
1 1 -1.0 -1 IMP:N=1
```

We can see that the definition "-1" for the third value on the card which indicates inside the RPP "1" is a simpler definition for the cell than using the union of six planes. Alternatively, the importance can be omitted from the cell card and listed for all the cells in sequence order in the Data Card section. We can use a shortcut nR to repeat the preceding entry n times to shorten the input. For example, in the Data Card section, `IMP:n 1 4R` replaces `IMP:n 1 1 1 1 1`.

5.5 Building the Input Deck–Data Cards VOL and SDEF

After the geometry of the model is completely defined, we can finish our model by completing the Data Card section. Remember, we already made some data cards when we defined the materials for the model. As described in Table 5.1, the data block section follows a single blank line delimiter after the cell block. In addition to the materials, the Data Card section can include parameters for the cells, source definitions, burnup cards, tally cards, and print controls.

As a technique, we will use a `VOL` card to define the volume for all of the cells in our problem. MCNP is capable of calculating the area and volume of many cells, however there are some geometries that it cannot adequately define such as asymmetric, non-polyhedral, or infinite cells. In order to adequately define all of the cells, we will demonstrate the use of a stochastic volume calculation in the next chapter. For the simple geometries in this chapter, we can easily calculate the volumes by hand. To specify the volume for each cell in the Data Card section write the following:

```
VOL value1 value2 value3...
```

Each value is the volume for the cells defined in the Cell Block in cm^3. The values are in the order that cells are listed.

Another method to specify the volume of the cells is in the cell parameters, like we did with the importance using a `VOL=(value)` where the value is the volume in cm^3. If we use the stochastic volume calculation, it is easier to specify the volume in the Data Card section.

The definition of the source is just as important to the problem as the definition of the geometry. Within MCNP, the `SDEF` command controls the source command. Table 5.5. is a list of the variables for the `SDEF` command that are used if other than the default value is needed to define the source.

Table 5.6: Source Variable for the SDEF Command

Variable	**Meaning**	**Default**
CEL	cell	determined from XXX, YYY, ZZZ and possibly UUU, VVV, WWW
SUR	surface	0 (means cell source)
ERG	energy (MeV)	14 MeV

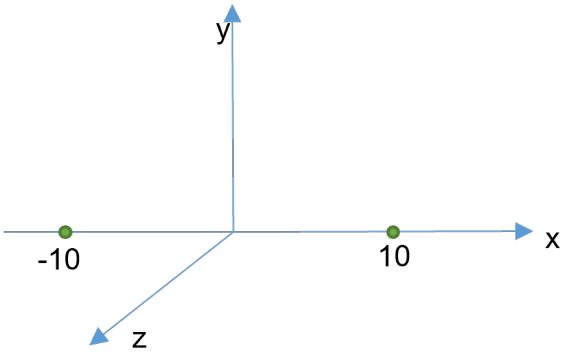
Table 5.6: Source Variable for the SDEF Command

Variable	Meaning	Default
DIR	μ, the cosine of the angle between VEC and UUU, VVV, WWW. The azimuthal angle is always sampled uniformly in [0, 2]	Volume case: μ is sampled uniformly in [-1.1] (isotropic). Surface case: p(μ) = 2μ for μ [0, 1] (cosine distribution).
VEC	reference vector for VEC	Volume case: required unless isotropic. Surface case: vector normal to the surface with sign determined by NRM.
NRM	sign of the surface normal	1
POS	reference point for positioning sampling	0, 0, 0
RAD	radial distance of the position from POS or AXS	0
EXT	Cell case: distance from POS along AXS. Surface case: cosine of angle from AXS	0
AXS	reference vector for EXT and RAD	no direction
X	x-coordinate of position	no X
Y	y-coordinate of position	no Y
Z	z-coordinate of position	no Z
CCC	cookie-cutter cell	no cookie-cutter cell
ARA	area of surface (required only for direct contributions to point detectors from a plane surface source	none
WGT	particle weight	1
EFF	reference efficiency criterion for position sampling	0.01
PAR	type of particle source emits	= 1 (neutron) if MODE N or P or N P E; = 2 (photon) if MODE P; = 3 (electron) if MODE E

The MCNP User's Manual has examples of complicated sources and description of the SDEF command line. This description is typically challenging for new users to understand. Often problems can be described with simple source

definitions. The following source definitions from *An MCNP Primer* by Shultis and Faw are excellent examples for simple sources. (Shultis; 2011)

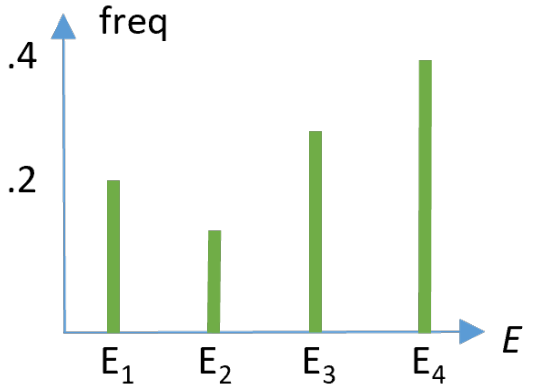
Two Point Isotropic Sources at Different Positions




```
C ---- Source: two point isotropic 1-MeV photon sources on x-axis
SDEF    ERG=1.00  PAR=2  POS=d5  $ energy, particle type, location
SI5 L   -10 0 0 10 0 0           $ (x,y,z) cords of the two pt sources
SP5     .75       .25            $ relative strengths of each source
```

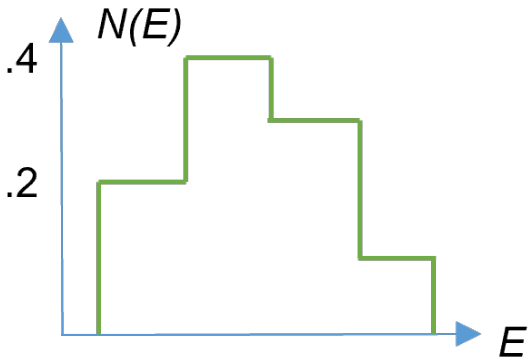
Note: The label 5 in the distribution d5 links to the SI5 and SP5 cards

Point Isotropic Source with Discrete Energy Photons



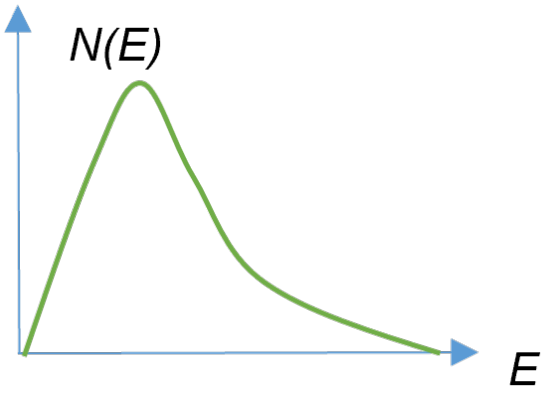

```
C ---- Source: point isotropic source with 4 discrete photon energies
SDEF    POS 0 0 0 ERG=d1 PAR=2   $ location, energy, particle type
SI1 L   .3 .5 1. 2.5             $ the 4 discrete energies (MeV)
SP1     .2 .1 .3 .4              $ frequency of each energy
```

Point Isotropic Source with a Histogram of Energies



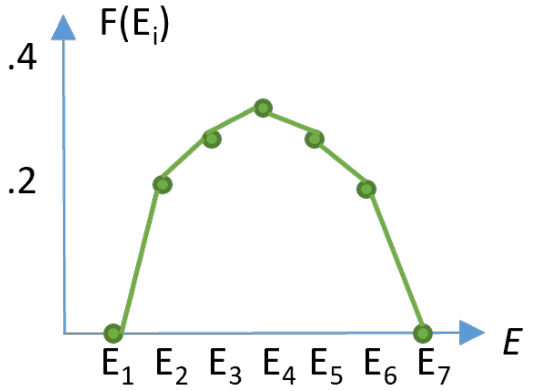

```
C ---- Source: point isotropic source 4 histogram energy bins
SDEF    POS 0 0 0 PAR=2 ERG=d1   $ location, particle type, energy
SI1 H   .1 .3 .5 1. 2.5          $ histogram boundaries
SP1     0 .2 .4 .3 .1            $ probabilities for each bin
```

Point Isotropic Source with a Continuum of Energies



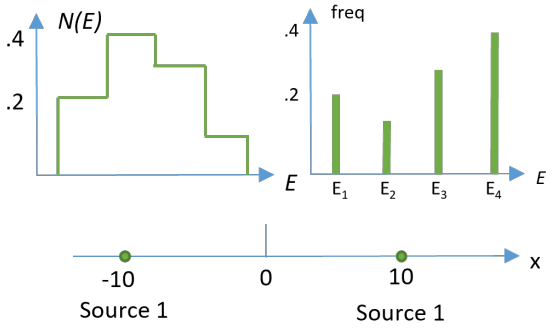

```
C ---- Source: point isotropic with Maxwellian energy spectrum
SDEF    POS 0 0 0 PAR=2 ERG=d1   $ location, particle type, energy
SP1     -2 0.5             $ Maxwellian spectrum (2) with tem a=0.5 MeV
```

Point Isotropic Source with Tabulated Energy Distribution



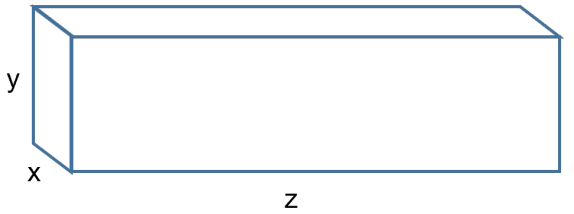

```
C ---- Source: continuum energies tabulated at discrete energies
SDEF     POS 0 0 0 PAR=2 ERG=d1    $ location, particle type, energy
SI1     A 1 2 3 4 5 6 7           $ tabulated energies E1 ...E7
SP1      0 .2 .27 .3 .28 .18 0     $ distribution values f(Ei)
```

Two Point Sources with Different Energy Distributions



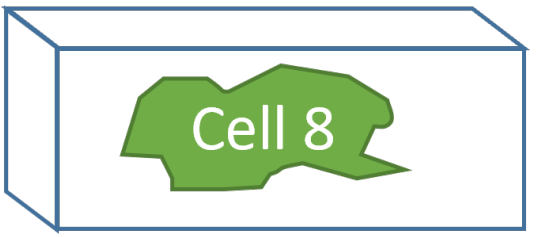

```
C ---- 2 pt iso sources: src 1 (4-bins) src 2 (4 discrete Ei)
SDEF    PAR=2 POS=d1 ERG FPOS d2 $ particle type, energy, location
SI1   L -10 0 0  10 0 0          $ cords of srcs on x-axis
SP1        .4     .6             $ rel strengths of sources
DS2 S      3      4              $ energy distributions
SI3 H      .1 .3 .5. 1. 2.5      $ E bin limits src 1
SP3 D      0 .2 .4 .3 .1         $ bin prob for src 1
SI4 L      .3 .5 .9 1.25         $ discrete Ei for src 2
SP4       .20 .10 .30 .40        $ rel freq for src 2
```

Rectangular Parallelepiped Parallel to Axes




```
C ---volumetric monoenergetic source inside a rectangular parallelepiped
SDEF    X=d1 Y=d2 Z=d3 ERG=1.25 PAR=2   $ location, energy, par type
SI1 -10. 10.          $ x-range limits for source volume
SP1     0 1           $ uniform probability over x-range
SI1 -15. 15.          $ y-range limits for source volume
SP1     0 1           $ uniform probability over y-range
SI1 -20. 20.          $ z-range limits for source volume
SP1     0 1           $ uniform probability over z-range
```

Source in a Complex Cell: Enclosing RPP Rejection Method



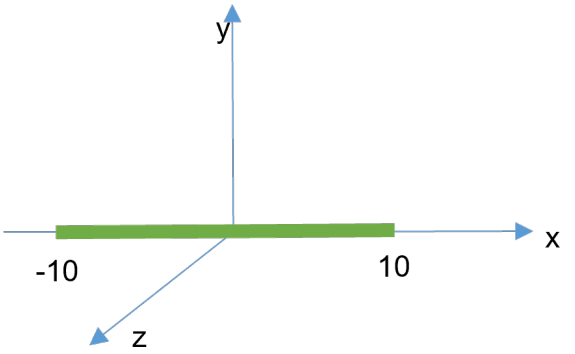

```
C --- Cell 8 is some complex cell in which a monoenergetic isotropic
C     volumetric source exists.  A rectangular parallelepiped envelops
C     this cell (MCNP does NOT check this!).  Points, randomly picked
C     in the rectangular parallelepiped, are accepted as source points
C     only if they are inside cell 8.
C
SDEF    X=d1 Y=d2 Z=d3 ERG=1.25 PAR=2 CEL=8  $ location, energy, par type
C   NOTE: source parallelepiped is larger than cell 8, and hence
C         source positions sampled outside cell 8 are rejected.
SI1 -12. 12.          $ x-range limits for source volume
SP1     0 1           $ uniform probability over x-range
SI1 -11. 11.          $ y-range limits for source volume
SP1     0 1           $ uniform probability over y-range
SI1 -13. 13.          $ z-range limits for source volume
SP1     0 1           $ uniform probability over z-range
```

Source in a Complex Cell: Enclosing Sphere Rejection Method



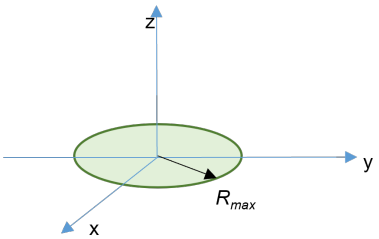

```
C --- Cell 8 is some complex cell in which a monoenergetic isotropic
C     volumetric source exists.  A sphere envelops this cell (MCNP
C     does NOT check this!).  Points, randomly picked in the sphere,
C     are accepted as source points if they are inside cell 8
C
SDEF    POS=0 0 0 RAD=d1 CEL=8
SI1      0  20.         $ radial sampling range: 0 to Rmax (=20 cm)
SP1    -21 2            $ weighting for radial sampling: here r^2
```

Line Source (Degenerate Rectangular Parallelepiped)



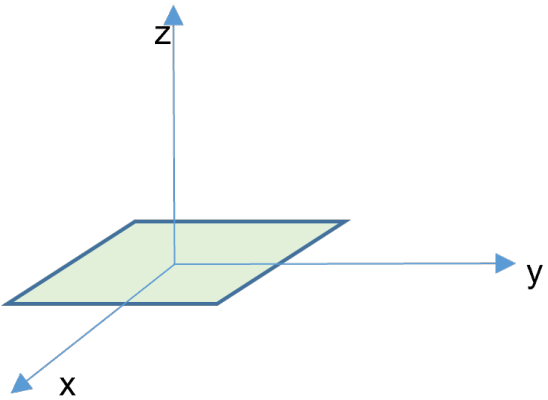

```
C --- Line monoenergetic photon source lying along x-axis
C     This uses a degenerate Cartesian volumetric source.
C
SDEF    POS=0 0 0 X=d1 Y=0 Z=0 PAR=2 ERG=1.25
SI1      -10 10         $ Xmin to Xmax for line source
SP1    -21 0            $ uniform sampling on line Here x^0
```

Disk Source (Degenerate Cylindrical Source)



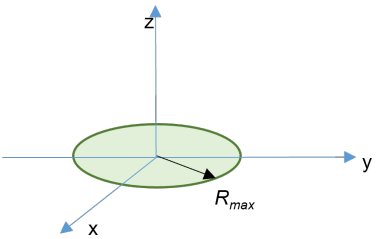

```
C --- disk source in x-y plane centered at the origin.
C     This is a degenerate cylindrical volume source.
C
SDEF    POS=0 0 0 AXS=0 0 1 EXT=0 RAD=d1 PAR=2 ERG=1.25
SI1     0  11         $ radial sampling range: 0 to Rmax
SP1    -21 1          $ radial sampling weighting: r^1 for disk source
```

Plane Source (Degenerate Rectangular Parallelepiped)



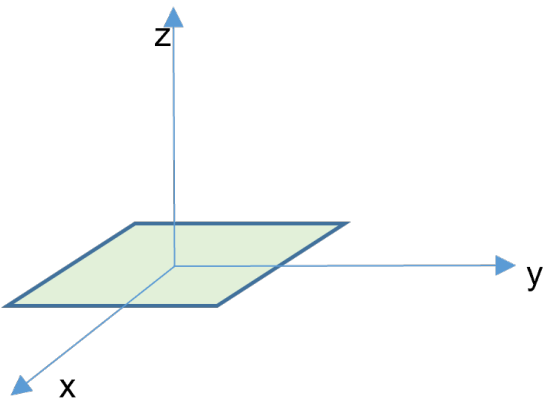

```
C --- rectangular plane source centered on the origin and perpendicular
C     to the y-axis. This uses a degenerate Cartesian volumetric source.
C
SDEF    POS=0 0 0 X=d1 Y=d2 Z=0 PAR=2 ERG=1.25
SI1     -10 10        $ sampling range Xmin to Xmax
SP1     0 1           $ weighting for x sampling: here constant
SI2     -15 15        $ sampling range Ymin to Ymax
SP1     0 1           $ weighting for y sampling: here constant
```

Disk Source (Degenerate Cylindrical Source)



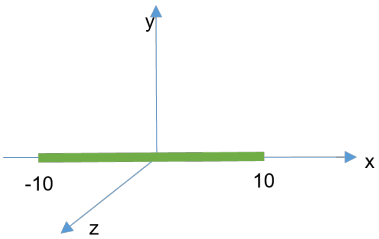

```
C --- disk source in x-y plane centered at the origin.
C     This is a degenerate cylindrical volume source.
C
SDEF    POS=0 0 0 AXS=0 0 1 EXT=0 RAD=d1 PAR=2 ERG=1.25
SI1     0  11         $ radial sampling range: 0 to Rmax
SP1    -21 1          $ radial sampling weighting: r^1 for disk source
```

Plane Source (Degenerate Rectangular Parallelepiped)



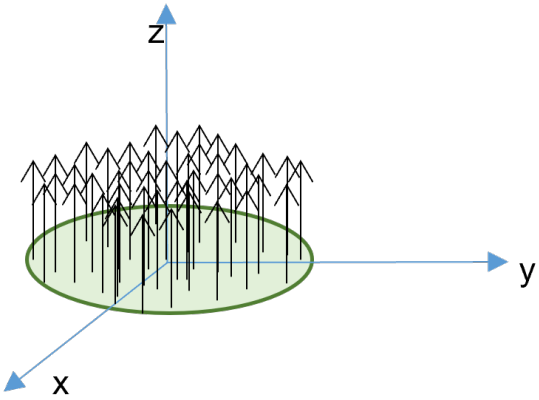

```
C --- rectangular plane source centered on the origin and perpendicular
C     to the y-axis. This uses a degenerate Cartesian volumetric source.
C
SDEF    POS=0 0 0 X=d1 Y=d2 Z=0 PAR=2 ERG=1.25
SI1     -10 10        $ sampling range Xmin to Xmax
SP1     0 1           $ weighting for x sampling: here constant
SI2     -15 15        $ sampling range Ymin to Ymax
SP1     0 1           $ weighting for y sampling: here constant
```

Disk Source (Degenerate Cylindrical Source)



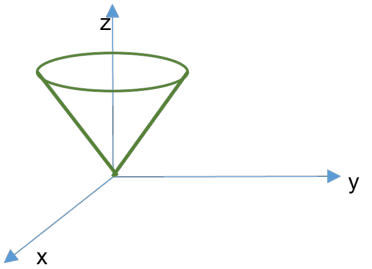

```
C --- line source (degenerate cylindrical volumetric source)
SDEF    POS=0 0 0 AXS=1 0 0 EXT=d1 RAD=0 PAR=2 ERG=1.25
SI1    -10  10       $ axial sampling range: -X to X
SP1    -21 0         $ axial sampling weighting: here constant
```

Monodirectional Disk Source




```
C --- disk source perpendicular to z-axis uniformly emitting
C     1.2-MeV neutrons monodirectionally in the + z-direction
C
SDEF    POS=0 0 0 AXS=0 0 1 EXT=0 RAD=d1 PAR=1 ERG=1.2
        VEC=0 0 1 DIR=1
SI1     0  16        $ radial sampling range: 0 to Rmax
SP1    -21 1         $ radial sampling weighting: r^1 for disk source
```

Point Source Collimated into a Cone of Direction




```
C --- Point isotropic 1.5 MeV photon source collimated into
C     an upward cone.  Particles are confined to an upward
C     (+z axis) cone whose half-angle is a cos(0.9) = 25.8
C     degrees about the z-axis.  Angles are with respect to
C     the vector specified by VEC
C
SDEF    POS=0 0 0 ERG=1.25 PAR=2 VEC=0 0 1 DIR=d1
SI1     -1  0.9   1  $ histogram for cosine bin limits
SP1    0 0.95 0.05   $ frac. Solid angle for each bin
SB1    0.   0.  1.   $ source bias for each bin
```

With this conical source, tally normalization is per source particle in 4π steradians. To normalize the tally per source particle in the cone, put WGT=1/fsa2 on the SDEF card, where fsa2 is fraction solid angle of the cone (0.05 in the above example). Calculate the fsa by using the cosine bin limits and taking the following:

$$\frac{[(1--1)-(1-0.9)]}{2} = 0.95$$

$$\frac{[(1-0.9)-(1-1)]}{2} = 0.05$$

The *Primer* has more specifications for multiple volumetric sources and some other sources that are not repeated here. The User's Manual also has many examples of a variety of sources and additional options and cases that are beyond the scope of this text. As a good practice, use the PRINT 110 card in the Data Card section. This option prints the locations, directions, and energies of the first 50 particles to be printed in the output.

Criticality Source

One additional source that we will cover does not have a specific geometry associated with it. That is the KCODE and KSRC source definitions used for criticality calculations. The definition of a criticality model takes a slightly different form than the previous examples. To use a criticality source, there

must be fissile material in the model. The criticality source consists of two cards, KCODE and KSRC. Table 5.6 lists the parameters for the KCODE card and their descriptions. The default parameter is used unless another value is specified. The KCODE card has the following form (Kulesza et al.; 2022):

Table 5.7: Input parameters for KCODE data card.

Input Parameter	Description
nsrck	Number of source histories per cycle. (DEFAULT: nsrck=1000)
rkk	Initial guess for k_{eff}. (See Note 1.) (DEFAULT: rkk=1.0)
ikz	Number of cycles to be skipped before beginning tally accumulation. (DEFAULT: ikz=30)
kct	Total number of cycles to be done. If kct=0, never terminate on the number of cycles, but terminate on time. (DEFAULT: kct=ikz+100)
msrk	Number of source pints for which to allocate storage (See Note 2.) (DEFAULT: msrk=maximum of 4500 or 2 x nsrck)
knrm	Controls normalization of tallies. If knrm=0, normalize tallies by weight. (DEFAULT) If knrm=1, normalize tallies by number of histories.
mrkp	Maximum number of cycle values on MCTAL or RUNTPE files. (DEFAULT: mrkp=6500)
kc8	Controls the number of cycles over which summary to average. If kc8=0, average over all cycles. (See Note 3.) If kc8=1, average over active cycles only. (DEFAULT)

Note 1: If in the first cycle the source being generated overruns the current source, the initial guess (rkk) is probably too low. The code then proceeds to print a comment, continues without writing a new course calculates k_{eff}, reads the initial source back in, and begins problem using k_{eff} instead of rkk. If the generated source again overruns the current source after the first cycle, the job terminates and either a better initial guess (rkk) or more source space (msrk) should be specified on the next try.

Note 2: If an SRCTP file with a larger value of msrk is read for the initial source, the larger value is not used.

Note 3: Setting the parameter kc8 to zero causes tallies and summary table information to be for both active and inactive cycles and should not be used. Setting kc8 = 0 also results in strange MCTAL file normalization.

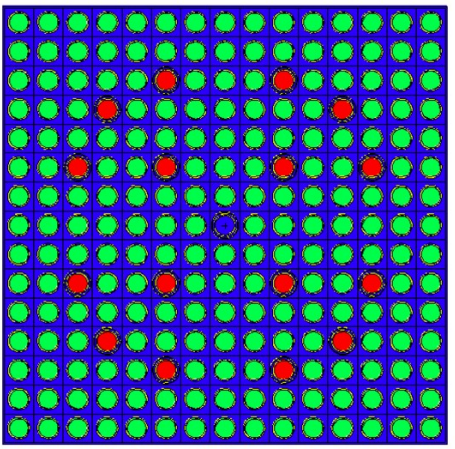


Figure 5.7: Top view of example MCNP model of a PWR assembly. Fuel rods are green, control rod guide tubes are red, and the center instrument guide tube is filled with water which is blue.

The second required card for a criticality source is the KSRC card which specifies the criticality source points. These are the locations for the initial points for the KCODE calculation. They must be located in a region in the model where there is fissile material. The KSRC card has the following form:

```
KSRC x1 y1 z1 x2 y2 z2...
```

where `x y z` are the locations of the initial source points. For small models, like our unit cell, we can use just one source point. For larger models, like a reactor core, multiple points will be needed throughout the core to get the criticality calculation started.

An example for a criticality source for a small single assembly model displayed in Figure 5.7 is presented below:

```
KCODE 2000 1.3 5 55 500
KSRC 1.443 0.0 0.0
```

In this case, the initial guess for k is 1.3 and the location for the source starting point is slightly off the origin by the value of the rod pitch (1.443 cm) because the center rod location did not have fissile material in it. The center of the model in Figure 5.7 is an instrument guide tube filled with water.

5.6 Building the Input Deck–Tallies and Misc. Cards

Tally cards are cards in the Data Card section of the input that the user uses to retrieve specific information from the model. Table 5.7 lists the types of tallies in MCNP. (Shultis; 2011) The *User's Manual* for MCNP describes the theory and application for all the possible tallies.

Table 5.8: Tally Types in MCNP. Particle type is given by *pl*.

Mnemonic	**Tally Type**	**particles** *pl*	**Fn Units**	***Fn Units**
F1:*pl*	surface current	N or P or N,P or E	#	MeV
F2:*pl*	average surface flux	N or P or N,P or E	#/cm^2	MeV/cm^2
F4:*pl*	average flux in a cell	N or P or N,P or E	#/cm^2	MeV/cm^2
FMESH4:*pl*	track-length tally over 3D mesh	N or P or E	#/cm^2	MeV/cm^2
F5:*pl*	flux at a point or ring	N or P	#/cm^2	MeV/cm^2
FIP5:*pl*	pin-hole flux image	N or P	#/cm^2	MeV/cm^2
FIR5:*pl*	planar radiograph flux image	N or P	#/cm^2	MeV/cm^2
FIC5:*pl*	cylindrical radiograph flux image	N or P	#/cm^2	MeV/cm^2
F6:*pl*	energy deposition	N or P or N,P	MeV/g	jerks/g
F7:*pl*	fission energy deposition in a cell	N	MeV/g	jerks/g
F8:*pl*	pulse height distribution in a cell	P or E or P,E	pulses	MeV

Initially in this text, we will only describe the use of the F1, F2, F4 and F5 tallies as they are the most used. We will discuss the FIR5 tally in a latter chapter when we discuss radiographic imaging. To specify a tally, at least one tally card is required with the first entry being F*n:pl*. The entry n is the tally id number. The last digit determines the type of tally and pl is the type of tally so a F12:N is a F2 tally of neutrons and a F31:P is a F1 tally of photons. You can also specify a N,P for a joint tally or E for electrons. After the tally type is the name of the surface for F1 or F2 tallies or cells for F4 tallies. A F5 tally has a position for the detector.

The following descriptions and examples of surface, cell and point-detector tally cards are from *An MCNP Primer* by Shultis and Faw (Shultis; 2011):

Cards for Surface and Cell Tallies

The card

```
F1:E 1 2 T $ current through a surface
```

specifies electron current tallies through surfaces 1 and 2, and the total (T) over both surfaces. Note that the current tally is not divided by surface area. The card

```
F2:P 1 (1 2) (2 3 4) T $ fluence averaged over surfaces
```

specifies photon surface-integrated fluence tallies for surface 1, the average over surfaces 1 and 2, the average over surfaces 2 through 4, and the average (T) over all surfaces 1 through 4. Similarly, the card

```
F4:N 1 (2 3 4) $ fluences averaged over cells
```

specifies cell-averaged neutron fluence tallies for cell 1 and for cells 2 through 4. No composite average is called for.

Cards for Point-Detector Tallies

For some models, there is not a cell or surface located where a tally is needed. For example, we may be interested in the flux located in the top of the reflector region in a reactor. Rather than adding a cell or surface just to do a tally, we can use a F5 point detector tally. In the sense of an experiment or a Monte Carlo calculation, as the volume of a cell approaches zero, the path length segments in the cell and the number of particles intersecting the surface of the cell also approach zero and, hence, the flux tally becomes indeterminate. However, there is a way of computing the flux at a point by using the deterministic last-flight-estimator tally F5. This tally is invoked by a card such as

```
F75:P X Y Z R $ point detector
```

Here 75 is the tally number, the last digit 5 denotes the F5 tally type, and P specifies the tally is for photons. The values of X, Y, and Z specify the coordinates of the point detector, and R designates the radius of a spherical exclusion zone surrounding the detector point. The need for an exclusion zone is evident from the $1/r^2$ term in the flux contribution tallied as presented in Equation 5.1.

$$\delta\Phi = \frac{W}{4\pi r^2} e^{-\mu(E)r} \tag{5.1}$$

Where r is the distance between the particle interaction site and the point

detector. If r approaches zero, the tally contribution approaches infinity. Such large contributions make the F5 tally much less stable than the cell (F4) or surface (F2) flux tally. This instability is minimized by establishing a spherical "exclusion volume" of radius R centered on the point detector. For interactions occurring within this exclusion zone, an abnormally large tally contribution is avoided by scoring the fluence uniformly averaged over the exclusion spherical surface. See the MCNP manual for a more detailed description. The exclusion radius R can be specified, as a positive number (centimeters, and is the preferred method), or a negative number (mean free paths). Typically, R should be about 0.2 to 0.5 mean free path (averaged over the energy spectrum at the sphere). For a point detector inside a void region, no interactions can occur near the detector and R should be set to zero. Finally, several point detectors may be specified on one tally card, e.g.,

```
F5:P X1 Y1 Z1 R1 X2 Y2 Z2 R2
```

Optional Cards for Tally Features

Note that the value reported for the tallies we are using in MCNP is per source particle. MCNP does not have a sense of time. In order to get an appropriate value for tally of flux, for example, the user has to multiply the value reported from the F4 tally by the actual number of source particles per second in the real-world problem this will convert the given value of n particles / cm^2 per source particle to particles/cm^2·s . A way to scale the value for the tally within MCNP to use a FMn card which is a tally multiplier. The value n corresponds to the tally id. For example:

```
F4:N 1 (2 3 4) $ fluences averaged over cells
Fm4 4.1E10
```

In this case, the values from the F4 tally will be multiplied by a constant given as 4×10^{10}. We can also use a tally energy card to divide the output into bins based on the energy of the particles being counted. For example,

```
F24:N 1 (2 3 4) $ fluences averaged over cells E24 0.01 0.1
1.0 10.0 $ energy bin boundaries
```

In this case, the values from the F4 tally will be binned into four energy groups where the values 0.01, 0.1, 1.0, and 10.0 are the upper limits of the energy bins in MeV.

Another useful option for tallies is the tally comment card (FCn) which allows the user to add a comment to the output file. This is different than a typical comment which is only in the input file. We will use the tally comment card in our first example problem.

Other Required Input for Data Cards

Two other cards are required in the Data Card section in order for the input description to be complete. The first is the MODE card. This card specifies the type of source particles to be tracked. It can be either a MODE P or MODE N or MODE E or a combination such as a MODE N P. The second is the NPS card which specifies the number of particle histories to run. This card will tell MCNP when to finish the run. It will not be used in KCODE problem as the KCODE will finish when the specified number of active cycles is reached.

5.7 First Example–Estimate of π Using MCNP

Now we have fully defined the input file, we are ready to make our first model. For this model, we will be using MCNP to perform an estimate of the value of π. Building on the concept presented in section 4.2 of this text, we can set up the same upper right quadrant of a unit circle in a square presented in Figure 4.4. We can bombard the model with a beam of parallel particles and then use a flux tally on the circle and square to estimate the areas of each. Using Equation 4.4, we can calculate the estimate of π.

Example 5.1
Using the model presented in Figure 4.4 and the formula in Equation 4.4, create an MCNP input file that will calculate an estimate for the value of π.

Solution: Open up TextPad® and create an input file. Start with materials. For this case, we do not want particles to be scattered back towards the source. We can use air as a material. We could have used a void but that would not allow us to demonstrate how to make a material. Because the Visual Editor and the MCNP plotter will plot different materials different colors, it is helpful if we name three identical materials. One material will be in the square outside of the circle, one will be in the circle and one will be outside the model but inside a large sphere that serves as the overall boundary to the model. As a matter of demonstration, we will specify the neutron library on the materials even though we will be using 1.0 MeV photons as our particles. The composition for air comes from *Introduction to Radiological Physics and Radiation Dosimetry* by Frank Attix. (Attix; 1986)

```
C MATERIAL DRY AIR AT STP (w/o TAKEN FROM ATTIX PG 532)
M1       6000.70C -0.0124 7014.70C -75.5267 8016.70C -23.1781 GAS=1
M2       6000.70C -0.0124 7014.70C -75.5267 8016.70C -23.1781 GAS=1
M3       6000.70C -0.0124 7014.70C -75.5267 8016.70C -23.1781 GAS=1
```

After the materials, we can specify the surfaces. We will use a rectangular parallelepiped for the square surface, a right circular cylinder for the circular surface, two planes to cut the problem into the positive quadrant and a large sphere to surround the entire problem. We have chosen to make the RPP 10 cm x 10 cm x 10 cm. The size is arbitrary because the calculation is based on a ratio of the areas for the square and the circle. We will set the importance to zero for particles as they cross boundary of the sphere to kill them off and allow the model to stop tracking them. This area will be known as the "graveyard".

```
C ****************************SUFACE CARD BLOCK*********************
C  Square RPP - one side faces the beam and will be used for a tally
1            RPP  -5 5  -5 5  -5 5
C  Right cylinder just inside RPP 1 - end will be used for tally
2            RCC  0 -4.999999 0  0 9.999998 0  4.999999
C  Plane normal to z axis to reduce the problem into 1/4
3            PZ 0
C  Plane normal to x axis to reduce the problem into 1/4
4            PX 0
C  Sphere outside of the problem for boundary to kill off particles
999          so 100
```

Now we can write the cell block using the density and material numbers for air and the surfaces we have just defined.

```
C ************************CELL CARD BLOCK*****************************
1      1    -0.001297  -1 2 (3 4)       $Air in square outside circle
2      2    -0.001297  -2 (3 4)         $Air inside circle
9      3    -0.001297  (1:-3:-4) -999   $Outside model air
99     0               999              $Outside sphere void graveyard
```

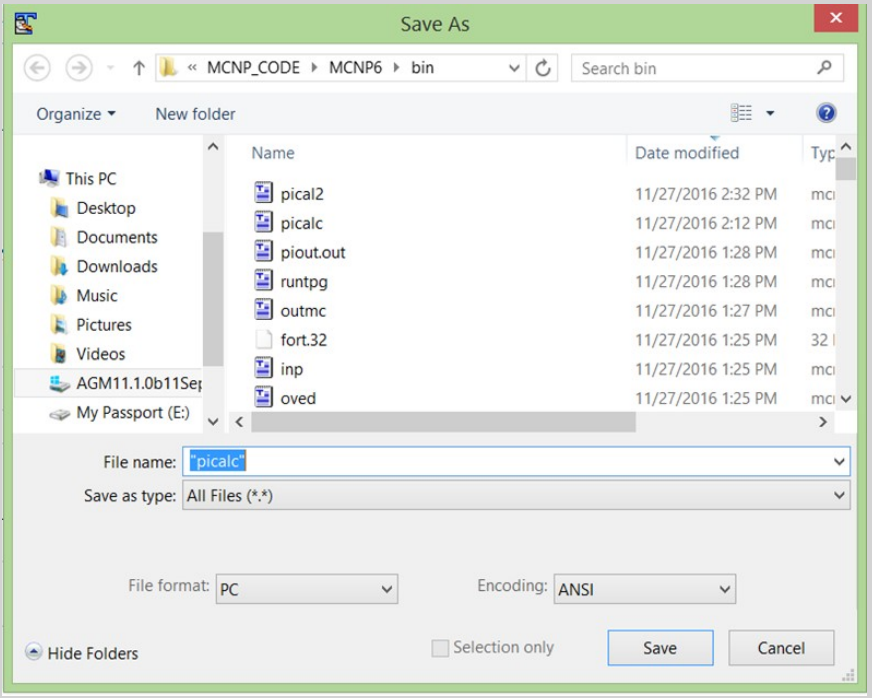
Note that we have not specified the volume or importance in the cell block. We will specify those in the data block in this example. In addition to those parameters, we will specify the mode, source, tallies, materials (previously done), number of histories and print options. The source has the same form as the disk source presented in Section 5.5 of this text.

The following is the complete input file.

```
NE361 Demonstration of Calculation of pi with MCNP
C ------------------------------------------------------------------------
C | This demo uses a disk beam source striking a circle inside a        |
C | square and uses the relationship between the tallies of             |
C | particles in the circle to the tallies of particles in the          |
C | square times 4 to estimate pi                                       |
C ------------------------------------------------------------------------
C ************************CELL CARD BLOCK*****************************
1      1    -0.001297  -1 2 (3 4)      $Air in square outside circle
2      2    -0.001297  -2 (3 4)        $Air inside circle
9      3    -0.001297  (1:-3:-4) -999  $Outside model air
99     0               999             $Outside sphere void graveyard

C ****************************SUFACE CARD BLOCK*********************
C  Square RPP - one side faces the beam and will be used for a tally
1            RPP  -5 5  -5 5  -5 5
C  Right cylinder just inside RPP 1 - end will be used for tally
2            RCC  0 -4.999999 0  0 9.999998 0  4.999999
C  Plane normal to z axis to reduce the problem into 1/4
3            PZ 0
C  Plane normal to x axis to reduce the problem into 1/4
4            PX 0
C  Sphere outside of the problem for boundary to kill off particles
999          so 100

C ****************************DATA CARD BLOCK************************
MODE    P
IMP:P   1.0 1.0 1.0 0.0 $importance for cells 1 2 9 and 99
VOL     214.6018366 785.3981634 124663.7061 $volumes cells 1 2 and 9
C  Disk source perpendicular to y-axis uniformly emitting
C  1.0 MeV photons in a beam directed toward cube and cylinder
SDEF    POS=2.5 7 2.5 AXS=0 1 0 EXT=0 RAD=d1 PAR=2 ERG=1.0
        VEC=0 1 0 DIR=-1
SI1 0 3.6 $Radius of disk that ensures source is larger than model
SP1 -21 1
C  Surface tally for the circle and square areas in upper quadrant
F11:P 1.3  2.2 $surface tallies for the square and the circle
FC11 Calculate Pi by taking the ratio of results of (2.2 / 1.3 ) x 4
C MATERIAL DRY AIR AT STP (w/o TAKEN FROM ATTIX PG 532)
M1      6000.70C -0.0124 7014.70C -75.5267 8016.70C -23.1781 GAS=1
M2      6000.70C -0.0124 7014.70C -75.5267 8016.70C -23.1781 GAS=1
M3      6000.70C -0.0124 7014.70C -75.5267 8016.70C -23.1781 GAS=1
NPS 1E7
PRINT 110
```

Note that the tallies are on surfaces 1.3 and 2.2. Because we are using macrobodies for the model, each face is given a specific number where the digit before the decimal is the number ID of the macrobody and the number after the decimal is the face number assigned by MCNP. A RPP will have six faces and a RCC will have three. We will see how to determine the number assigned using the plotting features in MCNP in the next section. We will also demonstrate running this input problem and compare the results for the output to the example problem we did with our Julia Monte Carlo problem in Chapter 4.

Save the file using the **Save As** and **Save as Type** as *.* all files to select an option other than the default of *.txt.



As a convention, MCNP input files do not have a filename extension. However, the user can use a *.txt extension, it only requires the inclusion of the extension in the command line when the file is run. The destination can be in any folder on the computer where MCNP is run, the user will need to navigate to that specified folder from the MCNP Command prompt in order to execute. A suggested method is to create a folder within the user's documents folder on the PC named MCNPIO to keep MCNP input and output files. For this example the filename selected will be "picalc". Including the quotation marks in the name. This will eliminate the filename extension from the input file. For example, without the quotes, the file will be called picalc.txt and with the quotes, the filename will be just picalc. As previously stated, either will work for execution.

This completes the creation of the input file. This input file will be used in the following sections as a demonstration for the plotting, running, and reading the output for an MCNP file.

5.8 Plotting in MCNP

The most effective method to visualize the geometry for a MCNP model and to check for errors is to plot it. There are two plotting programs available in the MCNP package. The first we will discuss is the Visual Editor or VISED. The VISED program was distributed with the MCNPX package and is still available from many older users who have legacy versions of the code, but is no longer supported or distributed with MCNP. To use VISED for plotting, double-click the VISED icon and the program window will open.

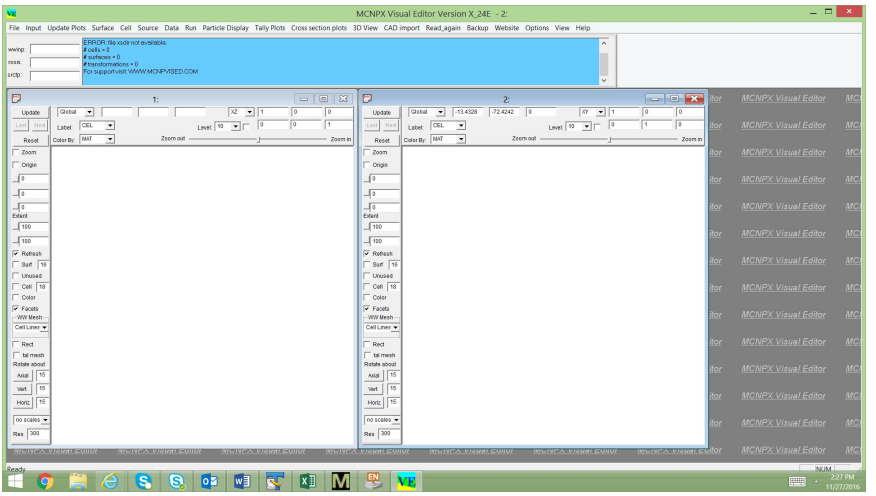


Figure 5.8: The VISED window when initially opened.

Use the "File" "Open" command to load the input file. As it loads, the program will search for errors and will notify you if there are any fatal errors – those errors that will cause the model not to run. You can choose to ignore the fatal errors and continue to load the model. Of course, you will have to correct them in order to run the model in MCNP. Once the model is loaded, click on the "Update" button on the upper left corner of each of the plot windows. For the input file we created in Example 5.1 named picalc we get two views of the model – XZ and XY. The default views show solid black lines for the outside of the cells. If any of the lines are red dashed lines, we would have an error in our geometry. We can then use the radio buttons to zoom in and change the orientation of the origin and viewpoint to look at the model from all sides. Just because there are no errors presented on one view does not mean that there are no geometry errors in the model. We can click on the color button and view the geometry in the XZ orientation to get the view in Figure 5.9.

After we are satisfied with the model boundaries, we can select the XY view and use the radio buttons to turn on the color by material and the surface numbers. When we move the origin to 2.5 7.0 1.0 and update the plot, we

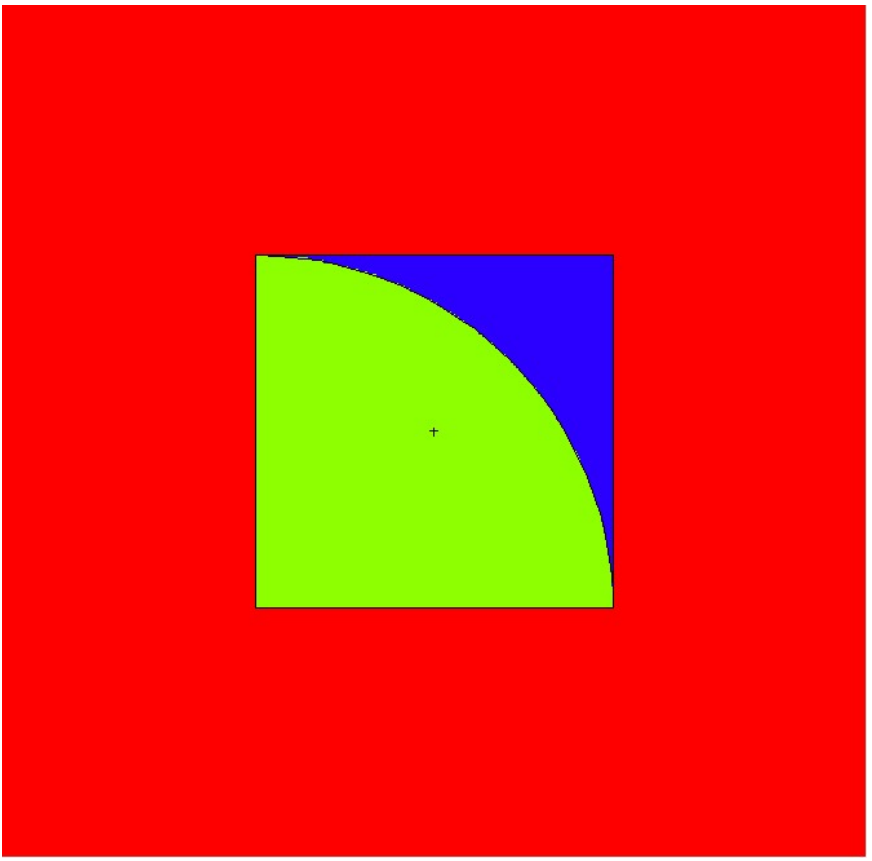


Figure 5.9: VISED plot of the picalc model in an XZ orientation. The three colors represent materials 1, 2, and 3 which all happen to be air.

get the view in Figure 5.10. By looking closely at the top of the model, we see there are two surfaces each pointing towards the positive y-axis where our source beam is centered. The center of the disk source is at 2.5 7.0 0.0 and has a radius of 10 cm. The orientation of the disk source is towards the negative y and strikes the two surfaces identified on the plot as 1.3 and 2.2. This is how we know which two surfaces we want to use in our tally line for our input.

The other program for plotting in MCNP has many of the same features and options as VISED. It is the integrated plotting function in MCNP6.1. The plotting function relies on a UNIX-based window to plot the model. For it to work on a Windows PC, we must use an X11 display server to make the plotter work. In this text, we use Xming which is a free download as our server.

To plot our picalc model using the integral plotter, first double-click the

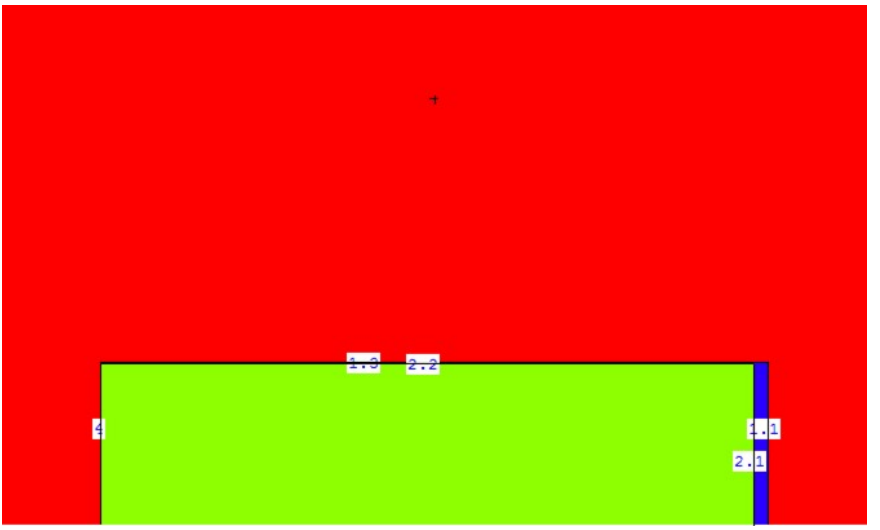


Figure 5.10: VISED plot of the picalc model in the XY view. The + is located with the origin which we placed at (2.5 7.0 1.0) which is at the XY position of the center of the source. The source is directed in the negative Y (down) and the beam strikes surfaces 1.3 and 2.3.

Xming icon to activate the server. The default server ID is 0.0. There is a line in our MCNP environmental batch file, which opens when we open the MCNP command prompt that recognizes a local server with the identification of 0.0. Once we have started Xming, we will double click the MCNP command prompt and enter the command,

```
C:\MCNPIO\> mcnp6 ip i=picalc o= picalc.out
```

This command tells the program to run mcnp6, use the plotter to plot and check for errors (ip) with the input filename of picalc and output filename of picalc.out. When we hit enter, the plot window pops up and tells us to click for the default plot which is presented in Figure 5.11.

With some inspection and selection of options, we can get the same information for errors, materials, and surfaces that we got with the VISED program. For example, the plot in Figure 5.12 is the same orientation as the VISED plot in Figure 5.9 for the same model.

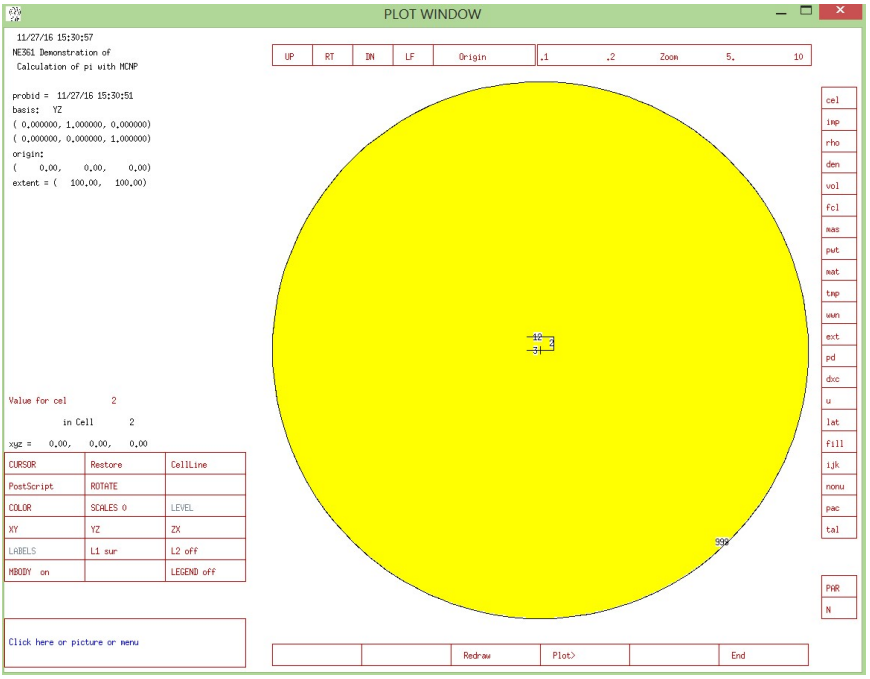


Figure 5.11: Default orientation plot for picalc in the integrated plotter for MCNP.

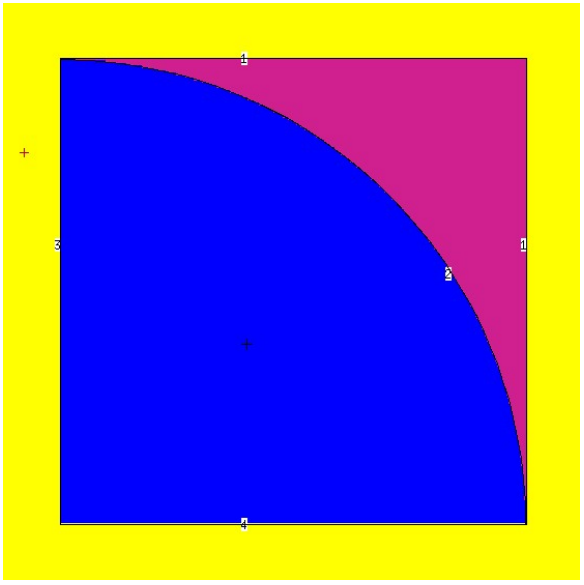


Figure 5.12: Integrated plotter plot of the picalc model in XZ orientation.

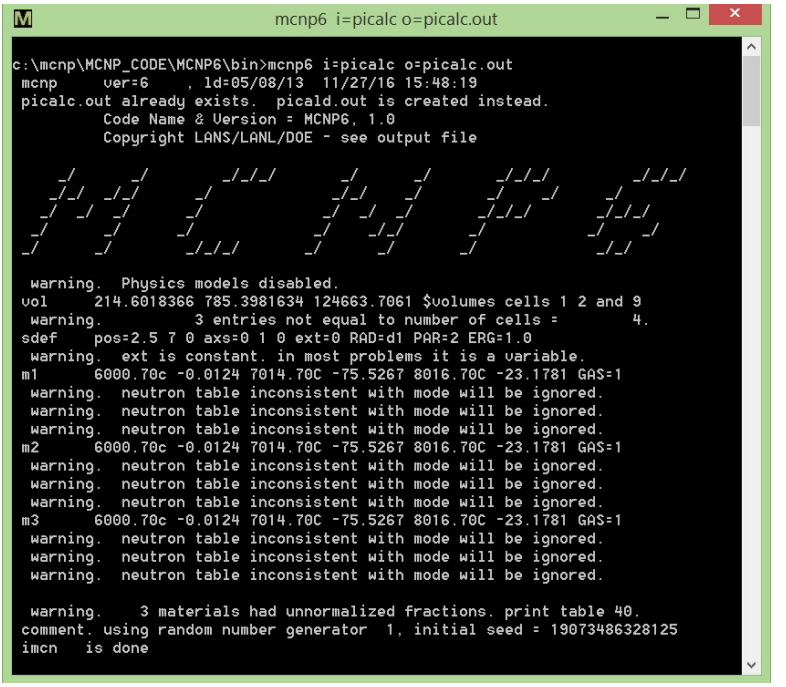
5.9 Running A Model in MCNP

Running the MCNP model is straightforward and the same as the plot command we just used without the ip. For our example problem, we would go to the MCNP command prompt and write:

```
C:\MCNPIO\> mcnp6 i=picalc o= picalc.out
```

Because we have set up the target destination in our prompt properties to the bin folder of the MCNP6 directory, we do not have to change directories to get there. The bin directory has the command executable mcnp6.exe. The input file should be saved in the same directory. If it is not, you will have to designate the location of the input file after the "i=" and before the filename. Similarly, you can do the same for the output if you want to save it to a different directory. It can become cumbersome to type in a long directory. Additionally, you should limit your filenames to simple designations with no spaces. MCNP was not written to accept filenames that are long and have spaces. Using good comments in the beginning of the input file will help you organize your files and know what they are designed to do when you access them later.

When we enter the command line to execute MCNP, we will get the following window:



Note that the warnings about the neutron tables have appeared as we mentioned. This is because we added the neutron libraries to the cross sections. MCNP will give warnings to help prompt the user to possible problems with the model. The user will have to know whether or not the warnings are important. If MCNP gives an error, the code will normally not run and they must be addressed to get results.

There are several control functions that are useful in case you want to stop a MCNP run that is operating:

<ctrl-c> k kills the job immediately

<ctrl-c> q stops the job normally after the current history

<ctrl-c> s give the status of the job

5.10 Reading the MCNP Output

We will discuss the error and statistical checks performed by MCNP in the output in the next chapter when we talk about variance reduction. For now, we will look at the output from our picalc model understanding that "the tally in the tally fluctuation chart bin did not pass 2 of the 10 statistical checks." We can open the output file in TextPad® and we will see that there is a lot of information provided. It starts with a repeat of the input file with warnings inter-spaced for information about possible errors. The next sections of the output include information about the cells and materials. We should inspect the values and ensure that the values are consistent with what we wanted.

Because we used the command PRINT 110, we will get the first 50 particle source points in tabular form. We can inspect the information and see that all of the particles start at a value of y = 7 and a direction with a vector of 0 -1 0 which is directed towards our model.

Finally, we can see the output for the tallies:

```
1tally       11        nps =    10000000
+     Calculate Pi by taking the ratio of results of (2.2 / 1.3 ) x 4
           tally type 1    number of particles crossing a surface.
           particle(s): photons

 surface  1.3
                  6.14168E-01 0.0003

 surface  2.2
                  4.82256E-01 0.0003
```

The values for surfaces 1.3 and 2.2 are the values for the surface current in particles / cm^2·source particle. The values 0.0003 and 0.0003 to the right of the tally values represent the variance in the values. We can calculate π using

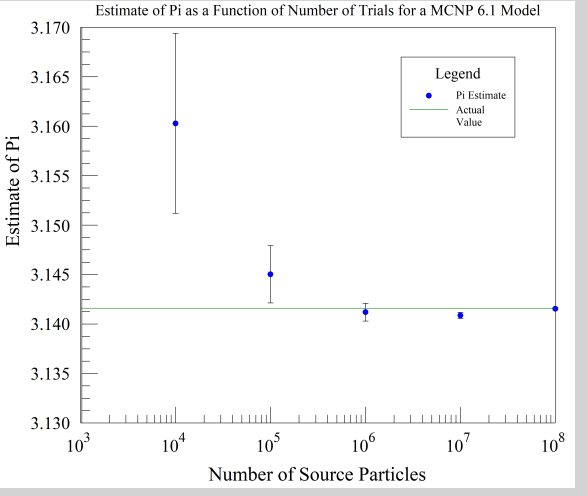
Equation 4.4 and we will get 3.140873507. If we propagate the uncertainty, we get the confidence interval 3.1409 ± 0.0003 for $1.0x10^7$ source particles. In Example 5.2, we will use our MCNP model picalc to repeat the analysis we did in Example 4.1 for an estimate of π and its variance as a function of particles.

Example 5.2
Use the MCNP model created in Example problem 5.1 to estimate the value of π. Vary the number of source particles from 1.0E4 to 1.0E8. Report the value and standard deviation for sample sizes from 1.0E4 to 1.0E8 and plot the results compared to the known value.

Solution: The only adjustment to the MNCP input file in Example 5.1 is to change the values for NPS. Run each model and record the results for the tallies and do the calculation for π using Equation 4.4 and propagate the uncertainty to report the variance.

Trials	Tally 1.3	Var 1.3	Tally 2.2	Var 2.2	Pi	Var	Actual	ctm (min)
1.00E+04	6.15E-01	0.0079	4.86E-01	0.0103	3.160292921	9.1000E-03	3.141593	0.01
1.00E+05	6.15E-01	0.0025	4.84E-01	0.0033	3.145036182	2.9000E-03	3.141593	0.02
1.00E+06	6.14E-01	0.0008	4.82E-01	0.001	3.141202969	9.0000E-04	3.141593	0.1
1.00E+07	6.14E-01	0.0003	4.82E-01	0.0003	3.140873507	3.0000E-04	3.141593	1.23
1.00E+08	6.14E-01	0.0001	4.82E-01	0.0001	3.141554356	1.0000E-04	3.141593	13.1

Note that we can get the computer time from the bottom of the input file. This is the time, in minutes, it took to do the calculation. Notice how much longer it took to solve the problem as we increased the number of source particles.



The results can be plotted with the variance as the error as we did in example 4.1. Some of the errors with the accuracy of the model come from the fact that the circle is not exactly tangent to the square. There is a very small gap between the two surfaces that contributes to the tally in the square so the result may be slightly lower than the actual. Also, we had a material of air for demonstration purposes. We could set all our materials to void to eliminate any scatters or absorption. We did not run the model out to 1E10 source histories just to save time as the result for 1.0E8 histories took about twelve times as long as 1.0E7 histories and we have a result that is both accurate and precise.

Bibliography

Attix, F. H. (1986). *Introduction to Radiological physics and radiation Dosimetry*, Wiley, New York.

Detwiler, R. S., McConn, R. J., Grimes, T. F., Upton, S. A. and Engel, E. J. (2021). *Compendium of Material Composition Data for Radiation Transport Modeling.*

IBM (2016). Image of puchcards.
URL: *http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/punchcard/*

Kulesza, J. A., Adams, T. R., Armstrong, J. C., Bolding, S. R., Brown, F. B., Bull, J. S., Burke, T. P., Clark, A. R., Forster III, R. A., Giron, J. F., Grieve, T. S., Josey, C. J., Martz, R. L., McKinney, G. W., Pearson, E. J., Rising, M. E., Solomon Jr., C. J., Swaminarayan, S., Trahan, T. J., Wilson, S. C. and Zukaitis, A. J. (2022). *MCNP Code Version 6.3.0 Theory User Manual.*

Shultis, J. K. ; Faw, R. E. (2011). *An MCNP Primer*, Department of Mechanical and Nuclear Engineering, Manhattan, KS.

Team, X.-. M. C. (2008). *MCNP - A General Monte Carlo N-Particle Transport Code, Version 5*, Los Alamos National Security, LLC, Los Alamos, NM.

Van Riper, Kenneth A. ; Metzger, R. L. (1996). Mcnp studies of doses from clinical x-ray procedures, *Transactions of the American Nuclear Society*, Vol. 75.
URL: *http://www.whiterockscience.com/bodybuilder/ans.html*

5.11 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- MCNP Materials Definition
- MCNP Surfaces Cards
- MCNP Cell Cards
- MCNP KCODE Source Definition
- Running MCNP Input File
- MCNP SDEF Source Definition

5.12 Problems

1. A simple shield of 2 cm of lead is placed in front of a parallel beam of gamma rays from a ^{137}Cs source in an arrangement depicted in Figure 4.6. Write an MCNP input file for the model and tally the uncollided flux (i.e. the gammas with the same energy as the source) on the far side the shield. Do this by setting an energy bin equal to the source strength in the tally. Report the results for the uncollided flux in the form of a confidence interval. Compare the value to the expected value given by equation 4.10.

2. Plot the model in Problem 1 using either VISED or the integrated plotter. Present the plot as a figure complete with dimensions and a descriptive caption.

3. Add another tally to the model in Problem 1 to get the total flux of gammas on the backside of the shield. Your output will have two tallies – one for uncollided flux and one for total flux. Vary the thickness of the shield from 1 mm to 2 cm thick by increments of 2 mm. Plot the values for the total flux and the uncollided flux on the same figure as a function of shield thickness. Comment on how the total flux changes as the shield becomes thicker. Does it follow the same negative exponential function that we used in Equation 4.11?

4. Write an MCNP input file that calculates the criticality for a bare sphere of ^{239}Pu at room temperature without a reflected boundary at a nominal density of 18 g/cm^3. While maintaining all other variables constant, vary the radius of the sphere from 3 cm to 10 cm to determine where the sphere is just critical (k-eff=1.0). Note: your volume card will have to change with the radius.

5. While maintaining all other variables constant, reduce the volume of a sphere as a function of compression and note the change in k-eff. This means reducing the radius and, therefore the volume of the sphere, while increasing the density of the material proportionally. Start with a 20 cm radius sphere at nominal density.

6. While maintaining all other variables constant, add Pu-240 to the sphere and note the change in k-eff. Increase the weight percent of Pu-240 from zero to 20%.

7. Write a batch file to run multiple MCNP runs sequentially to complete the iterative approaches where we vary one variable at a time.

8. Write an MCNP input file to calculate the criticality of the same unit cell model we calculated in SCALE in Example 3.1. Report the results of the criticality as a confidence interval. Use either VISED or the integrated plotter to plot the model for Problem 7.

9. Change the material card in the model used in Problem 8 from light water to a free gas treatment (i.e. comment out the MT card). Run the problem again and comment on the effect on the value reported for k-infinity.

10. Complete a parametric analysis using the unit cell model in Problem 8 by changing the number of inactive and active cycles in the criticality calculation to the uncertainty of the results. Increase the number of inactive cycles from 5 to 10 to 15 to 20 while keeping the number of active cycles constant. Next hold the number of inactive cycles constant at 10 and change the number of active cycles from 25 to 50 to 75 to 100.

Chapter 6

Advanced MCNP Topics

Now that we have experimented with simple MCNP models, we can start to stretch ourselves and test more complex problems. The great advantage to MCNP is its ability to handle complicated geometries and real-world problems with little need for approximations. The challenge becomes when we start to perform more complex problems and the geometry becomes more difficult to define and volumes are harder to calculate. Additionally, a complex problem may have a large variance in the solution that cannot be reduced without running extremely large number of source particles which takes a lot of computation time. In this chapter we will examine some techniques available to reduce variance in our model and the use of some other tools that allow for more realistic and complicated models.

6.1 Variance Reduction Techniques

When we created our first MCNP model, the estimate of π using MCNP, we did not closely examine the output from MCNP beyond getting the result from the tally and its associated variance. As we increased the number of source histories, the variance went down, and the computation time increased, which is what we would expect. When we look closer at the output, we notice that there is a warning that the tally did not pass one of the ten statistical checks performed by MCNP. This may or may not be a serious problem with the results of our model. MCNP performs 10 statistical checks on any tally. The 10 checks and the desired behavior are listed in Table 6.1. The first statistical check that we should be concerned with is the relative error, R, which is the fractional 1-sigma estimated uncertainty in the tally mean. Expect for point/ring detectors, $R < 0.1$ is considered acceptable. For values of R of 0.2 to 0.5 the value of the mean is within a factor of a few and may be useful for scoping calculations. If $R > 0.5$, the reported value should be considered meaningless. In our first MCNP case, the statistical check that fails is the FOM is decreasing as the NPS increases versus remaining constant. If we look closely at the Tally Fluctuation

Chart in the output, we will see that the FOM is very large for this model (on the order of 1E7) and the variance of the variance is zero. The FOM is large because we have a small variance and computer time due to the simple nature of the problem and the fact that there are nearly zero interactions of the 1.0 MeV photons in the air before they cross the surfaces for the tally. Nearly every source particle is counted in the tally. Because the FOM is so large and remains constant for the last half of the problem, we should not be concerned with the fluctuations presented early and we can be satisfied that our model is statistically sound.

Table 6.1: Name, Description and Desired Behavior of Statistical Checks on MCNP Tallies.

Name	Description	Desired Behavior
Mean	Average value of tally	Random
Relative Error Value	Ratio of the standard deviation of the tally to the mean.	<0.1*
Relative Error Decrease	The trend for the R as the number of histories (NPS) increases	Decreases as NPS increases
Relative Error Decrease Rate	The function of how the relative error decreases	Decrease as a function of $1/\sqrt{NPS}$
Variance of Variance Value	Estimate of the accuracy of R	< 0.1 is reliable for all tallies
Variance of the Variance Trend	Trend as NPS increases	Decreases as NPS increases
Variance of the Variance Decrease Rate	The function of how the variance of the variance decreases	Decreases as a function of $1/NPS$
Figure of Merit (FOM) Value	FOM is defined as $FOM \equiv \frac{1}{(R^2 T)}$ where T is the simulation time	Should remain relatively constant except near beginning of simulation
Figure of Merit Behavior	How the FOM varies throughout the problem	Random
Probability Density Function (PDF) Slope	The constructed PDF for the tally to assess the quality of the confidence interval for the mean	Slope of PDF >3.00

*Except for point/ ring detectors where R should be <0.05

Say, for instance, we had a different arrangement for our sample MCNP

problem where we did not have as many source particles being detected. This case could occur if we had a source that was well-shielded, and we were examining the flux on the far side of the shield (e.g. the lead pig and detector arrangement presented in Figure 6.1.) To reduce the variance, we could just simply throw more particles at the problem until we passed the statistical checks. This will, of course, cost more computer time and is inefficient. There are other ways to manipulate the problem to get good statistics without increasing the number of source particles. These variance reduction techniques essentially cause the problem to be focused and make the most use of each particle history. Variance reduction techniques should be used with caution as they can create errors in the result if used improperly.

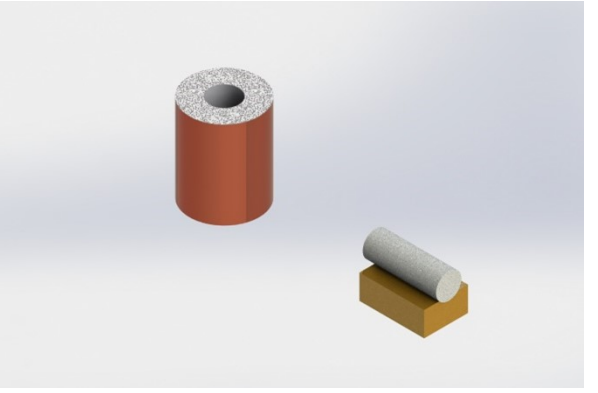


Figure 6.1: An example of a model of a thick lead shield coated in copper, a "pig" some distance from a detector. Depending on the strength of the source, variance reduction may be necessary to get reasonable results in MCNP.

There are two main approaches to variance reduction. The first is to simplify the model, also known as truncation. MCNP will, by default, track every particle and interaction at all energies and regions within our model down to the preset cutoff values in the program, which can cost a lot of computing time. We can simplify the model by using truncation techniques that kill off particles that are outside regions of interest or are not in the energy ranges of interest. In addition, we can simplify the physics calculations to ignore some secondary effects (like coherent scatter or Bremsstrahlung if these effects do not contribute to the information we are trying to gain from the problem. The two examples of model simplification we will discuss are Cutoffs and Physics Simplifications. On their own, they will not reduce the variance in the problem, however they will allow the user to increase the number of particles that are run without increasing computer time which will reduce variance. Remember that variance will always be reduced when we increase the number of particles.

Say we had the problem presented in Figure 6.1 and we were modeling it

as a point source of ^{137}Cs. If the lead shield was sufficiently thick, the number of counts on the detector face would be so small that the statistics would be meaningless. We could apply some simplifications to our model to truncate the problem and get better results. One of these is the CUT command. It is used to specify a minimum energy, time, or particle weight below which the particle is killed. This command applies to the entire model. For example:

```
CUT:p j 0.075 $kills photons with E < 75 keV
```

The CUT command has five parameters. They are time limit per history, energy cutoff, and parameters 3 to 5 are limits on the weights of particles. The j in the input line says to jump over the default first value for time and specify the 2nd value which is an energy cutoff.

The ELPT command is like the CUT command but cuts off particles at specified energies for each cell. They are listed in the order that the cells appear just like the VOL and IMP parameters in our example problem. The ELPT command can help in this problem because we may not be interested in particles below a particular energy in our detector if we are only looking for a specific energy. If the low energy particles are killed off, the computer time will be shortened.

In addition to the CUT command, we can use the PHYS command to truncate our problem. The PHYS command for photons has the form:

```
PHYS:P EMCPF IDES NOCHOL PNINT NODOP
```

The EMCPF parameter is the energy in MeV above which simple physics is to be used. In simple physics, no fluorescence from photoelectric interactions is produced, no binding effects are used in photon scattering, and no coherent scattering is included. IDES=0/1 indicates that Bremsstrahlung is included/ignored for MODE P. NOCOH=0/1 species that coherent scattering is included/ignored. PNINT= -1/0/1 photonuclear interactions are used in an analog manner / not used/ used with a bias. If PNINT $\neq$ 0 there must be a MPNn card following the material Mn card. NODOP=0/1 turns Doppler broadening (from the speed of bound electrons) on/off. The default is (Shultis; 2011):

```
PHYS:P 100 0 0 0 0 $100 MeV, brems, coh scat, no PN, Doppler
```

Depending on the problem, changing the physics options can greatly affect the overall computer time. For our first example, MCNP estimate of π, there is no noticeable speed-up or change because most of the particles do not interact before they reach the boundary of the problem where they are killed off. Later in this chapter, we will model the pig and detector in Figure 6.1 and see if

changing the physics models for this problem affects the results.

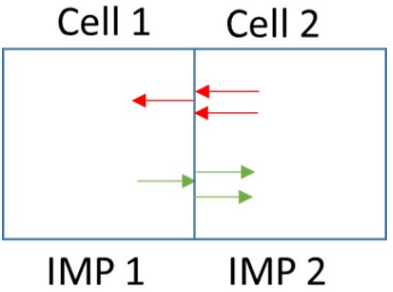


Figure 6.2: Simple example of geometry splitting. The weight for the green particle doubles when it goes from Cell 1 to Cell 2 and the weight for the red particles are reduced in half when they travel from Cell 2 to Cell 1.

The second primary method to reduce variance is to use non-analog simulations. What this means is that we can bias or focus the problem into our specific region of interest by a variety of methods. These methods include adjusting the particle's worth, or weight, as it goes from regions of low importance to regions of high importance, biasing the source towards the area of interest or killing off particles that are in areas of low or no importance and adding their weights to regions of high importance. For the purposes of this text, we will limit our non-analog simulation to the simple case of geometry splitting and source biasing. Further descriptions of all of the various variance reduction techniques and examples can be found in the *User's Manual* and *An MCNP Primer*. (Shultis; 2011)

As presented in Figure 6.2, geometry splitting allows us to change the importance for particles from one cell to another. In general, to use this approach, we give a greater importance to cells closer to the tally region than those farther away. The weight of a particle moving from the region of importance 1 to importance 2 is given the ratio of I2/I1 . What this means is that if cell 1 has an importance of 1 and cell 2 has an importance of 2, when a particle goes from cell 1 to 2 it will have a new weight of 2/1 or 2. Conversely, when a particle goes from cell 2 to cell 1, it will have a new weight of 1/2.

For our pig and detector model, we can divide up the cylindrical pig shield into multiple cells throughout the thickness of the shield and assign importance values that increase from the inside to the outside of the shield. A method for increasing the importance it to use the multiply shortcut "m". For example:

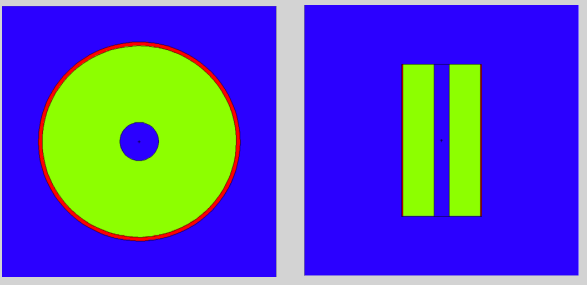
```
IMP 1 2m 2m 2m 2m 2m 0
```

This line says that the first cell has an importance of 1 and the following are 2, 4, 8, 16, 32, 0.

In addition to geometry splitting, there are other non-analog approaches to reducing the variance we can try. For the pig and detector model we could also work with the source in the problem. We know that our detector is in a particular solid angle from the detector. Instead of modeling the source as an isotropic point source with equal weight in 4π steradians, we could consider modeling it as a cone source pointed at the detector or as a point source with a bias that applies greater weight to the particles traveling towards the detector. In Example 6.1, we will model the pig and detector and use various variance reduction techniques and analyze the effects.

Example 6.1
A small spherical 0.1 curie ^{137}Cs source is in the container in Figure 6.1. The container is constructed of a cylinder with a radius of 5.2 cm and 20 cm high. The wall has a thickness of 4 cm of lead and 0.2 cm of copper. The detector is located along the centerline of the source at 1 m. Use MCNP to tally the uncollided flux and total gamma flux at the detector.

Solution: We will start by making a strict analog model without variance reduction. Use an isotropic point source of 0.662 MeV photons located at the origin for the source and a point detector (F5) tally at 100 cm. Apply a tally multiplier to convert the number of particle histories to actual flux. This is done by multiplying the tally by the source strength in decays/sec $= 0.1$ Ci $= 3.7\times10^9$ decays/sec. Bin the tally into three bins – E $<$ 0.662 Mev (collided flux), E $\geq$ 0.662 MeV (uncollided flux) and the sum (total flux). Use the nominal density and isotopic composition of natural lead and copper from the NIST website and the composition of air from Example 5.1 for the materials. Ignore the shielding on the top and bottom of the container because the detector is located along the center of the side and we will assume that scatters from particles streaming from the top and bottom are negligible. Once we set up the model, we can plot it from the top and side and see how the model looks from a plane through the middle horizontally and vertically.



In this case, blue is air, green is lead and red is copper. The source is in the center of the shield at the origin. Again, note that the pig is open on the top and the bottom as we are only concerned with shielding through the sides for this example. The complete input file is shown below:

```
NE361 Pig - Detector Model - No Variance Reduction
C ------------------------------------------------------------------
C | This is a model of a shielded container, a "pig" with a Cs-137  |
C | point source located inside.  The container is a cylinder       |
C | 20 cm high and has an overall radius of 5.2 cm.  The wall is    |
C | 4 cm thick lead with 0.2 cm copper. The detector is 100 cm away.|
C ------------------------------------------------------------------
C ************************CELL CARD BLOCK*****************************
1      1    -0.001297  -1       $air in middle of container
2      2    -1.135E+01  1 -2    $Lead inner cylinder
3      3    -1.135E+01  2 -3    $Copper outside lead
4      1    -0.001297   3 -999  $air outside the copper
99     0                999             $Outside sphere void graveyard

C ****************************SUFACE CARD BLOCK*********************
C  Right cylinder on origin filled with air
1            RCC  0 -10 0 0 20 0 1
C  Right cylinder on origin filled with lead
2            RCC  0 -10 0  0 20 0  5
C  Right cylinder on origin filled with copper
3            RCC  0 -10 0  0 20 0  5.2
C  Sphere outside of the problem for boundary to kill off particles
999          so 105

C ****************************DATA CARD BLOCK************************
MODE    P
IMP:P   1.0 1.0 1.0 1.0 0.0 $importance for cells 1 2 3 4 and 99
VOL     62.83185307 1507.964474 128.1769803 4847349.288
c       volumes cells 1 2 3 and 4
C  point source at the origin uniformly emitting
C  0.662 MeV photons isotopically
SDEF    POS=0 0 0 PAR=2 ERG=0.662
C  Point Tally at 100 cm along the x axis
F5:P 100 0 0 2 $a point tally with a 2 cm radius exclusion zone
Fm5  3.7E9 $Multiply tally by 0.1 Cu in decays/sec to get real flux
E5   0.661  10 T  $3 energy bins - collided, uncollided, and total
C Material Dry Air at STP (w/o Taken from Attix PG 532)
M1      6000 -0.0124 7014 -75.5267 8016 -23.1781 GAS=1
C Lead at nominal density from NIST at 1.135E+01 gm/cm^3
C Use the stable isotopes and abundances for natural lead from
C The Chart of the Nuclides online
M2      82204 -0.014 82206 -0.241 82207 -0.221 82208 -0.524
C Copper at nominal density from NIST at 8.960E+00 gm/cm^3
C Use the stable isotopes and abundances for natural copper from
C The Chart of the Nuclides online
M3      29063 -0.6915 29065 -0.3085
NPS 1E7
PRINT
```

When we run the problem, it takes 6.50 mins (on the example computer) we get the tally table below:

```
1tally        5        nps =    10000000
          tally type 5    particle flux at a point detector.
          particle(s): photons

          this tally is all multiplied by  3.70000E+09

 detector located at x,y,z = 1.00000E+02 0.00000E+00 0.00000E+00
      energy
    6.6100E-01   2.26835E+02 0.0079
    1.0000E+01   2.15495E+02 0.0205
      total      4.42330E+02 0.0108
```

We can see that the flux values are very small with the total flux being less than 500 photons/cm^2·s. Also, the variances are fairly large and the output indicates that the results failed 2 of the 10 statistical checks – the variance of the variance was > 0.1 and the slope for the PDF was < 3.0.

We will now apply some variance reduction techniques and see what happens. First, let's apply a CUT parameter to kill off all low-energy particles.

```
CUT:p j 0.075 $kills photons with E < 75 keV
```

By cutting off the low-energy photons, the simulation took less time to run – now 4.02 mins. However, the statistics are worse as it now fails 3 statistical checks. This makes sense because we now have less histories reaching the detector. We can now choose to increase our NPS value to attempt to get better statistics in the same time it took to do the original run of about 6 min.

As another test, we will remove the CUT command and apply a PHYS card that will change the treatment of the problem to simple physics and turn off Bremsstrahlung, coherent scatter, photonuclear effects and Doppler broadening.

```
PHYS:p 0 1 1 0 1
```

Using the simple physics treatment, the model took 3.13 minutes to run and we get results that now only fail one statistical check – the slope of the PDF. It may be possible to use a simple physics model and increase the number of particles to get good statistics, however, we may be sacrificing some valuable information using the simple model.

Return to the default physics model by removing the PHYS:p card and we will now focus on non-analog approaches to variance reduction. The first will be to change the source from a point source to a cone pointed at our detector. This will limit our source particles to those that have a reasonable chance of interacting with our detector. We will make a point source collimated into a cone. We will use the same example presented for the cone in Chapter 5. The

tally will be automatically normalized per source particle in 4π steradians so it should give us the same numerical count at our detector that we would get from an isotropic point source at the same location. The new SDEF is:

```
SDEF     POS=0 0 0 ERG=0.662 PAR=2 VEC=1 0 0 DIR=d1
SI1      -1  0.9   1  $ histogram for cosine bin limits
SP1       0 0.95 0.05   $ frac. Solid angle for each bin
SB1       0.   0.  1.   $ source bias for each bin
```

The non-analog conical source took 5.57 mins to complete, and it now passed all of the 10 statistical checks. The results from the conical source:

```
1tally        5        nps =    10000000
           tally type 5    particle flux at a point detector.
           particle(s): photons

           this tally is all multiplied by  3.70000E+09

 detector located at x,y,z = 1.00000E+02 0.00000E+00 0.00000E+00
      energy
    6.6100E-01   1.10887E+02 0.0045
    1.0000E+01   2.16168E+02 0.0039
      total      3.27055E+02 0.0030
```

Note that the total of 327 photons/cm^2·s from our conical source is significantly less than the value of around 440 photons/cm^2·s we were getting with our previous models. This may be an indication that the cone is too narrow, and we are not correctly modeling the source. We can broaden the cone by increasing the half-angle from 25.8 degrees to 36.9 degrees and change the SDEF to the following:

```
SDEF     POS=0 0 0 ERG=0.662 PAR=2 VEC=1 0 0 DIR=d1
SI1      -1  0.8   1  $ histogram for cosine bin limits
SP1       0 0.90 0.10   $ frac. Solid angle for each bin
SB1       0.   0.  1.   $ source bias for each bin
```

The broader conical source took 6.27 mins to complete and failed 1 of the 10 statistical checks. The results from the model are:

```
1tally        5        nps =    10000000
           tally type 5    particle flux at a point detector.
           particle(s): photons

           this tally is all multiplied by  3.70000E+09

 detector located at x,y,z = 1.00000E+02 0.00000E+00 0.00000E+00
      energy
    6.6100E-01   1.68015E+02 0.0061
    1.0000E+01   2.15446E+02 0.0053
      total      3.83461E+02 0.0040
```

There is a lower variance from the point source and a higher number of counts than the narrow cone, but we may still be missing some values. Because we failed one of the statistical checks, broadening the cone more is unlikely to be successful.

We have one other non-analog method for variance reduction we will try with our model. For this case, we will divide the lead shield into two 2.0 cm thick cells radially. We will then increase the importance for particles from the source through the cells of the lead.

This model will allow us to multiply the weight of particles directed outward from the source and reduce any particles that are scattered inward. It will effectively multiply our source particles and should result in a higher FOM.

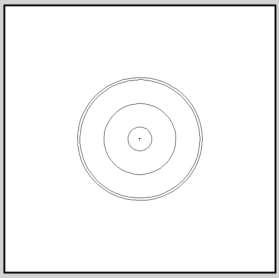
We will run the problem twice, once with two intermediate cells and once with three cells. The cell and surface definition blocks for the new input file are as follows:

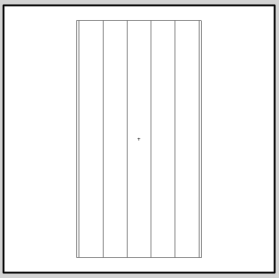
```
C ************************CELL CARD BLOCK*****************************
1      1    -0.001297  -1       $air in middle of container
2      2    -1.135E+01  1 -22    $Lead inner cylinder
22     2    -1.135E+01  22 -23   $Lead layer 2
3      3    -1.135E+01  23 -3    $Copper outside lead
4      1    -0.001297   3 -999  $air outside the copper
99     0               999             $Outside sphere void graveyard

C ****************************SUFACE CARD BLOCK*********************
C  Right cylinder on origin filled with air
1            RCC  0 -10 0 0 20 0 1
C  Right cylinder on origin filled with lead (layer 1)
22           RCC  0 -10 0  0 20 0  3
C  Right cylinder on origin filled with lead (layer 2)
23           RCC  0 -10 0  0 20 0  5
C  Right cylinder on origin filled with copper
3            RCC  0 -10 0  0 20 0  5.2
C  Sphere outside of the problem for boundary to kill off particles
999          so 105
```

We also must change the VOL cards to reflect the new cells. We will apply a non-analog variance reduction by modifying the importance of each cell. The new importance card looks like:

```
C importance for cells 1 2 22 3 4 and 99
IMP:P 1.0 2m 2m 1.0 1.0 0.0
```

The new plot of the model without color shows the cells:





The computation time for the geometry splitting model was 10.72 min or about twice as long as the analog solution. The final FOM for the tally is 3027 which is 2.3 times greater than the analog solution. To get the same FOM, the analog model would have to run 2.3^2 or 5.28 times longer. Also, the model now passes 9 of the 10 statistical checks as opposed to 8. The results from the tally table are:

```
1tally        5        nps =    10000000
          tally type 5    particle flux at a point detector.
          particle(s): photons

          this tally is all multiplied by  3.70000E+09

 detector located at x,y,z = 1.00000E+02 0.00000E+00 0.00000E+00
      energy
    6.6100E-01   2.30749E+02 0.0075
    1.0000E+01   2.15479E+02 0.0081
      total      4.46229E+02 0.0056
```

To try and get better results, we will spit the model into three cells increasing the importance by 2m and run it again. By increasing the number of cells to three the computer time increases to 12.94 mins. The model now passes all 10 statistical checks, and the results are below:

```
1tally        5        nps =    10000000
          tally type 5    particle flux at a point detector.
          particle(s): photons

          this tally is all multiplied by  3.70000E+09

 detector located at x,y,z = 1.00000E+02 0.00000E+00 0.00000E+00
      energy
    6.6100E-01   2.31576E+02 0.0063
    1.0000E+01   2.18714E+02 0.0099
      total      4.50290E+02 0.0058
```

We can now be confident in the results for the model:
Uncollided Flux: $2.1871 \times 10^2 \pm 0.0099$ photons/cm^2·s
Total Flux: $4.5029 \times 10^2 \pm 0.0058$ photons/cm^2·s

6.2 Stochastic Volume Calculation

When we are working with simple geometries, we can easily calculate the volumes of cells and, for most cases, MCNP can also calculate the volumes. However, MCNP cannot calculate the volumes and areas of asymmetric, non-polyhedral, or infinite cells. (Kulesza et al.; 2022) When we begin working with complex geometries and repeated lattice structures, it is important that we have a good calculation of the cell volumes and calculation by hand can be very difficult. Fortunately, there is a simple way to use MCNP to calculate the volume for us using what is known as a stochastic volume calculation.

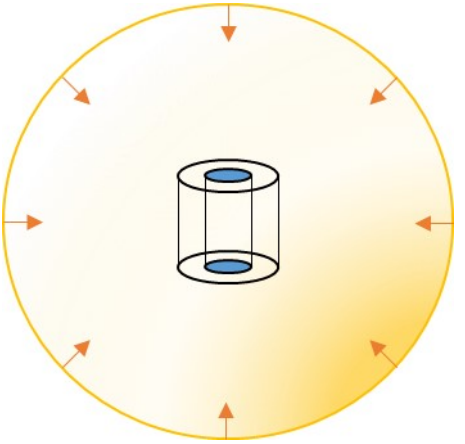


Figure 6.3: A stochastic volume calculation uses a bounding sphere as a source around the model with inwardly directed particles and an F4 volume tally to determine the volume of each cell.

For a stochastic volume calculation, we are going to take our entire model and place it inside a bounding sphere that serves as an inwardly directed source. We will set the source weight equal to the cross-sectional area of the sphere, πr^2. We will also include a VOID card that will void out all the materials so the particles will pass right through all the cells without collision. Finally, we will set all the volumes for the cells in the model equal to 1 and perform a F4 volume tally. The results of the volume tally will be equal to the volume in the model. Example 6.2 is a stochastic volume tally for the pig – detector problem from Example 6.1 with three cells in the shield region.

In Example 6.2, we will add a generic block at the end of the data card sections that we can use with little modification for all our future models for volume calculations.

Example 6.2
Perform a stochastic volume calculation on the pig detector model from Example 6.1 that had three cells in the lead shielded region.

Solution: Use sphere 999 as the source surface and add the stochastic volume tally operation to the end of the Data Card section and comment out the appropriate cards. The complete input file is below:

```
NE361 Pig - Detector Model - Importance Variance Reduction VOLUME CAL
C ------------------------------------------------------------------------
C | This is a model of a shielded container, a "pig" with a Cs-137  |
C | point source located inside.  The container is a cylinder       |
C | 20 cm high and has an overall radius of 5.2 cm.  The wall is    |
C | 4 cm thick lead with 0.2 cm copper. The detector is 100 cm away.|
C ------------------------------------------------------------------------
C ***********************CELL CARD BLOCK*****************************
1      1    -0.001297  -1       $air in middle of container
2      2    -1.135E+01  1 -21    $Lead inner cylinder
22     2    -1.135E+01  21 -22   $Lead layer 2
23     2    -1.135E+01  22 -23   $Lead layer 3
3      3    -1.135E+01  23 -3    $Copper outside lead
4      1    -0.001297   3 -999  $air outside the copper
99     0               999            $Outside sphere void graveyard

C ******************************SUFACE CARD BLOCK*********************
C  Right cylinder on origin filled with air
1            RCC  0 -10 0 0 20 0 1
C  Right cylinder on origin filled with lead (layer 1)
21           RCC  0 -10 0  0 20 0  2.333333333
C  Right cylinder on origin filled with lead (layer 2)
22           RCC  0 -10 0  0 20 0  3.666666666
C  Right cylinder on origin filled with lead (layer 3)
23           RCC  0 -10 0  0 20 0  5
C  Right cylinder on origin filled with copper
3            RCC  0 -10 0  0 20 0  5.2
C  Sphere outside of the problem for boundary to kill off particles
999          so 105

C ****************************DATA CARD BLOCK************************
MODE    P
C XXXXXXXXXX  COMMENT OUT IMP, VOL, SDEF, AND TALLY   XXXXXXXXXXXXXX
C  importance for cells 1 2 22 23 3 4 and 99
C IMP:P   1.0 2m 2m 2m 1.0 1.0 0.0
C VOL     62.83185307
C        279.2526705
C        502.6545272
C        726.057276
C        128.1769803
C        4847349.288
C       volumes cells 1 2 22 23 3 and 4
C  point source at the origin uniformly emitting
C  0.662 MeV photons isotropically
C SDEF    POS=0 0 0 PAR=2 ERG=0.662
C  Point Tally at 100 cm along the x axis
C F5:P 100 0 0 2 $a point tally with a 2 cm radius exclusion zone
C Fm5  3.7E9 $Multiply tally by 0.1 Cu in decays/sec to get real flux
C E5   0.661  10 T  $3 energy bins - collided, uncollided, and total
C Material Dry Air at STP (w/o Taken from Attix PG 532)
C XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
M1      6000 -0.0124 7014 -75.5267 8016 -23.1781 GAS=1
C Lead at nominal density from NIST at 1.135E+01 gm/cm^3
C Use the stable isotopes and abundances for natural lead from
C The Chart of the Nuclides online
M2      82204 -0.014 82206 -0.241 82207 -0.221 82208 -0.524
C Copper at nominal density from NIST at 8.960E+00 gm/cm^3
C Use the stable isotopes and abundances for natural copper from
C The Chart of the Nuclides online
M3      29063 -0.6915 29065 -0.3085
NPS 1E7
PRINT
C  ********************************************************************
C Uncomment these cards to do the stochastic volume tally.
C Note you must comment out the IMP, VOL, SDEF and TALLY cards above.
C If present, comment out KCODE, KSRC and BURN cards as well.
C Remove reflected boundary conditions if present
C Set all IMP for tally cells to 1 and VOL to 1
C Wgt calculated as area of xs of plotting sphere (pi R^2)
C
IMP:P  1 1 1 1 1 1 0
Vol    1 1 1 1 1 1
F4:P   1 2 22 23 3 4 $cell names for volume tally
VOID $ Voids out all materials
SDEF   SUR=999  NRM=-1  DIR=D1  WGT= 34636.05901 PAR=2
SP1    -21 1
C  NPS    10000000
```

The result of the tally is:

```
cell  1    6.27335E+01 0.0116
cell  2    2.80016E+02 0.0073
cell  22   5.02768E+02 0.0057
cell  23   7.29121E+02 0.0048
cell  3    1.29519E+02 0.0055
cell  4    4.84638E+06 0.0001
```

The values are nearly identical to the hand calculations we performed earlier for the volumes. Now we can use the block-select mode in TextPad® to quickly copy and paste the values into the VOL block for the model and comment out the stochastic volume calculation cards and uncomment the original cards.

The stochastic volume calculation can be performed with either photons or neutrons as the source. The only requirement is that the mode of the problem is consistent with the tally type.

6.3 Temperature Effects in MCNP

As we previously discussed in Chapter 1, the cross sections for many isotopes significantly change with temperature. In section 1.6, we discussed how NJOY99 can be used to generate cross section libraries at various temperatures which we were able to view. In Example 3.3, we changed the fuel temperature within our unit cell model in NEWT to calculate the fuel temperature coefficient, α, for our infinite reactor model. If the user has access to NJOY, they can use it

to create user-defined cross sections to add to the cross-section library in our MCNP code. Care will have to be maintained modifying some files that are integral to how MCNP calls data and operates.

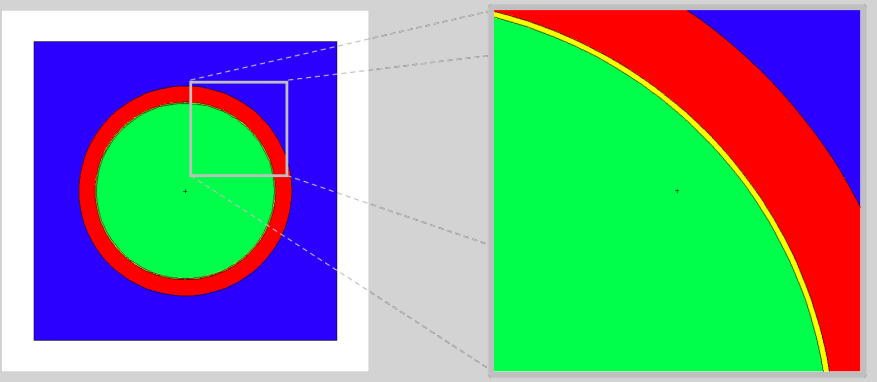
The MCNP code uses a few different files to link the working program with the data and other information it needs to calculate the problem. One file, "xsdir" is a file located in the MCNP_DATA directory. The xsdir file is a library index. It lists the cross-section file names which point to the file where the cross-section data files are located and their place on the computer. When we make new cross sections using NJOY99, we will have to add the cross-section file to the data library and add information to the xsdir file so MCNP can find our new cross-sections. Often, a user-defined cross-section will be named using a file extension that is unique, like .90c, so it does not overwrite an integral library in the program. Once completed, the new cross-section libraries can be accessed by adding the library extension to the isotope ZAID nomenclature.

Because MCNP has some temperature-specific cross sections already, can look at the effect of fuel temperature on reactivity without generating our own cross sections. We did this calculation using the NEWT module in SCALE in Example 3.3. For Example 6.3, we will model the same unit cell pin in MCNP. This pin will have the same dimensions, material compositions and boundary conditions as the NEWT model. We will run the problem 5 times. For the sake of simplicity and to get a broader range of temperatures, we will use the ENDF7 libraries of *.70C, *.71C, *.72C, *.73C and *.74C, instead of the libraries we generate using a program like NJOY99. These libraries are already in the MCNP database and correspond to 293.6, 600, 900, 1200 and 2500 K. Once complete, we will plot the k_∞ as a function of temperature and use the slope of the line to estimate the fuel temperature coefficient. Finally, we will compare the results from our MCNP model to those we calculated in SCALE.

Example 6.3
Use MCNP to model the same unit fuel pin used in Example 3.1. Perform a series of KCODE calculations changing the neutron cross section libraries for the fuel from *.70C to *.71C to *.72C to *.73C and to *.74C. Report the results for the k_∞ and variance at each temperature. Plot the results and use the same technique used in Example 3.3 to calculate the fuel temperature coefficient. Compare the results to those in Example 3.3.

Solution: First make a unit cell model for MCNP. Include a stochastic volume calculation that can be commented in and performed.

We can plot our unit cell model using either the MCNP plotter or VISED:



The plot allows us to check for errors and ensures it is consistent with the model we made in SCALE. In this case, the fuel is green, cladding is red, and the moderator is blue. The zoomed in view on the right shows the gap is present in yellow.

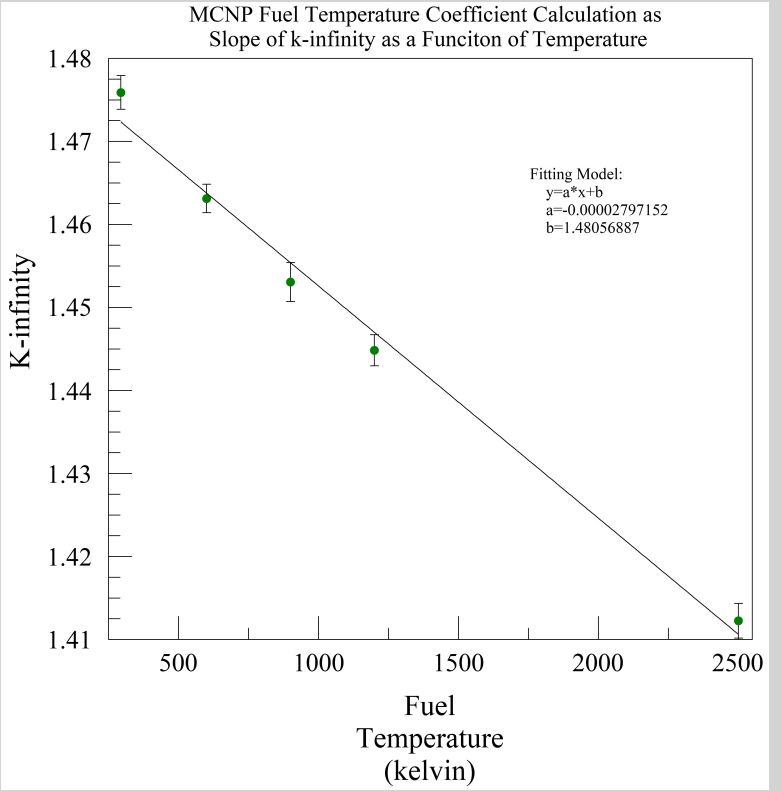
```
NE361 Single PWR Unit Cell
C ------------------------------------------------------------------------
C | This is a KCODE calculation for a single PWR unit cell pin.          |
C | The cell has a pitch of 1.331 cm, length of 320 cm.  The fuel        |
C | radius is 0.385 cm.  It is fueled with UO2 at 4.5 w/o enriched       |
C | U-235.   The cladding is zirconium and the gap is He                 |
C ------------------------------------------------------------------------
C ************************CELL CARD BLOCK*****************************
1       2     -10.2   -2 -91 90         $UO2 FUEL PELLET
2       3     -6.407   2 -3 -91 90      $FUEL PIN CLAD
3       4     -0.001297   3 -4 -91 90   $He IN GAP
4       1     -0.715    -1 #1 #2 #3     $UNIT CELL WATER
199     0   1   -999                    $Void Outside of pin
999     0   999                         $Graveyard

C  OUTSIDE OF UNIT CELL (1.311 cm PITCH)
C  REFLECTED BOUNDARY ON ALL SIDES
*1     RPP -0.6555 0.6555 -0.6555 0.6555 -160 160
C  CYLINDER OUTER RADIUS OF FUEL PELLET 0.385 cm
2      CZ 0.385
C  CYLINDER OUTER RADIUS OF GAP 0.390 cm
3      CZ 0.390
C  CYLINDER OUTER RADIUS OF CLADDING 0.461 cm
4      CZ 0.461
C  PLANES FOR THE TOP AND BOTTOM OF FUEL ROD
90              PZ      -159
91              PZ       159
c
999 so 600  $sphere for graveyard
```

```
c ************************ Data Cards ****************************
c
IMP:n     1 1 1 1 0 0
VOL       1.52396E+02
          3.85329E+00
          6.27137E+01
          3.31355E+02
c
c ****************** Criticality and Burn Control Cards **********
MODE     N
KCODE    2000     1.45      55        100       500
KSRC     0.0      0.0       0.0
M1       8016.71C      -8.88100E+01
         1001.71C      -1.11900E+01
MT1      LWTR.04T $H20 AT 600 K
M2       92234.74C     -4.69781E-05
         92235.74C     -0.039186291
         92238.74C     -0.842188828
         8016.74C      -0.118577903
         $UO2 4.5% at 293K
M3       40000.58C 1.0 $ZIRCONIUM (APPROX. FOR ZIRCALLOY)
M4       2004.70C  1.0 $ Helium
PRINT -128
C
C  ********************************************************************
C Uncomment these cards to do the stochastic volume tally.
C Note you must comment out the kcode, ksrc, and vol cards above.
C You must change the importance from 1 1 1 1 0 0 to 1 1 1 1 1 0.
C Also, remove reflected boundary conditions on surfaces 1
C
C Wgt calculated as area of xs of plotting sphere (pi R^2)
C
C Vol     1 1 1 1
C F4:N    1 2 3 3
C VOID
C SDEF    SUR=999  NRM=-1  DIR=D1  Wgt= 1130973.35529
C SP1     -21 1
C nps     10000000
```

We will leave the problem the same for all five runs except we will change the neutron library identifier from *70c - *.74c for the isotopes in the fuel. After we run the simulation at each temperature we get:

Fuel Temperature (kelvin)	$\mathbf{k_\infty}$	**Variance**
293.6	1.47589	0.00203
600	1.46312	0.00172
900	1.45307	0.00236
1200	1.44484	0.00187
2500	1.41226	0.00209

As with all MCNP results, we include the variance in the reported value. We also use this variance for the error bars in the following plot:



When we look at the results for the KCODE calculation and compare them to the NEWT calculation for the same unit fuel pin we can make some observations. First, the MCNP values are higher for k_∞. At room temperature, the value for MCNP is about 2% greater. In fact, all of the MCNP values are about 2% higher at the same or similar temperatures. The result is a fuel temperature coefficient that is nearly the same at -3×10^{-5} Δk/ degree K. What this implies is there is a systematic difference between the two models. Perhaps the order of quadrature in the deterministic model has some effect. More likely is a difference in the way the fuel enrichment was calculated when we specified the enrichment in NEWT. If we were using the two models to validate each other or if we had some experimental data we were trying to match, it would be essential for us to determine the exact cause for the systemic differences and account for them.

6.4 Burnup and Depletion

In Section 3.6 we described how reactor fuel is depleted over time. In Example 3.4, we modeled our single pin unit cell in TRITRON and depleted it for just over 1500 days so we could see how the reactivity changed over time. In MCNP it is possible to deplete nuclear fuel in the same manner. Just like SCALE uses ORIGIN to estimate the burnup and depletion of fuel in between time steps, MCNP uses CINDER to calculate the isotopic concentrations based on the length of the burnup step, the initial composition of the fuel, and the power applied to the model.

Depletion can only be added to a KCODE model as MCNP will iterate between KCODE cycles and BURN steps in what is known as a predictor-corrector method. The BURN card is added in the DATA CARD section of the input file using the following format:

```
BURN KEYWORD=values(s)
```

The values associated with the primary keywords are presented in Table 6.2.

Table 6.2: Keywords for BURN card and Descriptions. The complete list is in the User's Manual. (Kulesza et al.; 2022)

Keyword	Value
TIME=t$_1$ t$_2$...	Incremental time duration t_i for each successive burn step. Time unit is days. (DEFAULT: one time step lasting on day)
PFRAC=f$_1$ f$_2$ ...	Fraction f_i of total system power (pwr) applied to burn step t_i. Caution: if only a single PFRAC value (f_1) is provided, but multiple time steps (t_i) are specified, the first time step (t_1) wilL be assigned a power fraction of f_1; subsequent time steps will have a power fraction of 0. (DEFAULT: f_i = 1.0 for all t_i)
POWER=pwr	Total recoverable fission system power (MW). (DEFAULT: POWER=1.0)
MAT=m$_1$ m$_2$ ...	Material number m_i of material to be burned. Corresponds identically to material number specified on a material specification card Mm (DEFAULT: burn all materials) If m<o, then recoverable energy per fission and neutrons per fission are computed and contribute to the power normalization, but the material is not burned. (Warning: if you have duplicate ZAIDs on an M card of a material to be burned, wrong answers can result. MCNP6 will not give you a warning in this instance).

Table 6.2: Keywords for BURN card and Descriptions. The complete list is in the User's Manual. (Kulesza et al.; 2022)

Keyword	Value
BOPT=b_1 b_2 b_3	b_1=Q value multiplier. (DEFAULT: b_1=1.0) The parameter b_2 is used to control the ordering and content of the output and is the additive result of two integer values: b_2=I1+I2. The first value, I1, selects among three tiers (see Table 3-71 in the User's Manual) of fission product content: If I1=0, include only Tier 1 fission products (DEFAULT) If I1=10, include Tier 2 fission products, which comprise all fission products listed in the XSDIR cross-section directory of MCNPX 2.5.0. If I1=20, include Tier 3 fission products, which comprise fission products in ENDF/B-VII.0 that have CINDER90 yield information. The second value, I2, selects among four ordering options: If I2=1, order output inventory high to low based on mass. (DEFAULT) If I2=2, order output inventory high to low based on total activity. If I2=3, order output inventory high to low based on specific activity. If I2=4, order output inventory based on increasing ZZZAAA. The parameter for b_2 is defined by: If $b_2>0$, output will be printed at end of job only. (DEFAULT) If $b_2<0$ output will be printed at end of each burnup step. The parameter b−3 allows the user to disallow the use of high-energy physics models if the cross-section energy is less than the benchmarked values. If b_3 =-1, a fatal error will occur if tabular data are unavailable for any isotope. (DEFAULT) If b_3 =0, the atom fraction of any data using a model is set to 0. If b_3 =1, use cross-section models for nuclides not containing tabular data and then allow CINDER90 to calculate the 1-group cross section for these nuclides by importing a 63-group flux and matching to a 63-group cross-section set.

Table 6.2: Keywords for BURN card and Descriptions. The complete list is in the User's Manual. (Kulesza et al.; 2022)

Keyword	Value
OMIT= m$_1$ n$_1$ j$_{11}$ j$_{12}$ ... j$_{1n1}$ m$_2$ n$_2$ j$_{21}$ j$_{22}$ ... j$_{2n2}$	For each specified material number, m$_i$, omit the following n$_i$ isotopes from the transport calculation: j$_{i1}$ j$_{i2}$ ... j$_{ini}$. Each j$_{ink}$ must be provided in the form ZZZAAA, where ZZZ is the isotope's atomic number and AAA is its atomic mass number.
AFMIN= af$_1$ af$_2$	af$_1$=Atom fraction below which an isotope will no longer be tracked in the transport calculation. If the atom fraction of an isotope falls below this limit, the atom fraction is set to 0. (DEFAULT: af$_1$ = 1.0E-10) af$_2$=Transmutation chain convergence criteria used in CINDER90. (DEFAULT: af$_2$=1.0E-10)
MATVOL=v$_1$ v$_2$... v$_n$	Used to provide the volume of all cells containing a burn material in a repeated structure or lattice geometry. Each v$_i$ entry is the volume of all cells containing burn material mi. Each mi on MAT keyword needs a v$_i$.

When we complete a BURN calculation, we may not need to use all of the KEYWORDs in Table 6.2. For example, we will only need to use a MATVOL keyword if we have a repeated lattice, which we will do in the next section. The fission product Tiers are described in the User's Manual in Table 3-71. Unless we are tracking specific fission product isotopes, we can normally use Tier 1. In Example 6.4 we will add a BURN card to the model we used in Example 6.5. We will specify the model burns material 2, the fuel. For the power in the burnup, we will need to calculate the total MW that the single fuel pin contributes to the reactor. This is different than the specific power we used in SCALE. The power for a single fuel pin for a reactor that has on the order of 50,000 fuel rods operating at about 3000 MW is around 0.06 MW for each fuel pin.

Example 6.4
Use the model for the single unit fuel pin used in Example 6.4 to complete a burnup calculation for around 1500 days. Keep the moderator, cladding and gap temperature at 600 K and use 1200 K for the fuel. Plot the criticality as a function of time.

Solution: For our model, we added the following BURN card:

```
BURN   TIME =   1 2 4 25 25 50 100 100 100 100 100 100 100 100 100
                100 100 100 100 100
       MAT = 2
       POWER = .06
       OMIT = 2  4 6014 7016 8018 9018
```

The card can go right after KSRC line. We used small time steps to begin the model to build up the fission products. The total time is 1507 days. We are burning material 2 which is the UO_2 at a power of 0.06 MW. Because we are using the default values for BOPT, when we run the model without the OMIT card, we get a fatal error indicating that tabular data is not available for 4 isotopes (6014 7016 8018 and 9018).

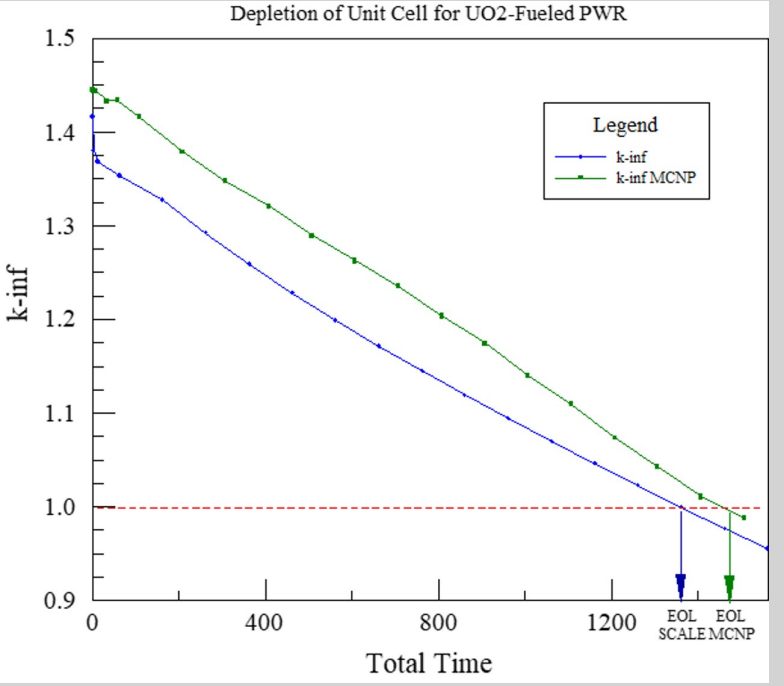
Because these isotopes are all formed in very small fractions in the fission process and they do not have significant neutron cross sections, we can add an OMIT card to remove them from the problem and prevent the fatal error from occurring. We could have chosen a different BOPT option, or we could have created cross sections for those isotopes using NJOY99 and added them to MCNP if we wanted to track them. If we choose a larger tier for fission products, we will get more isotopes that will not have tables.

Note that the total calculation will take some time. Each time step requires two complete KCODE calculations (1 predictor and 1 corrector). This means our model will do 40 total KCODE calculations and it may take an hour or more to complete.

When done, we can open the output file and find the burnup table and get the results. Note that the burnup table does not print the standard deviation. You must search through the output for each standard deviation for each step.

Total Time MCNP	**k-Inf MCNP**	**Variance**	**Total Time MCNP**	**k-Inf MCNP**	**Variance**
0.00E+00	1.44507	0.00176	6.07E+02	1.26293	0.00216
1.00E+00	1.44435	0.00201	7.07E+02	1.23589	0.00181
3.00E+00	1.44434	0.00198	8.07E+02	1.20415	0.00220
7.00E+00	1.44444	0.00211	9.07E+02	1.17485	0.00231
3.20E+01	1.43341	0.00209	1.01E+03	1.14024	0.00236
5.70E+01	1.43380	0.00169	1.11E+03	1.10995	0.00207
1.07E+02	1.41621	0.00214	1.21E+03	1.07386	0.00183
2.07E+02	1.37887	0.00178	1.31E+03	1.04276	0.00171
3.07E+02	1.34795	0.00201	1.41E+03	1.01094	0.00227
4.07E+02	1.32105	0.00204	1.51E+03	0.98851	0.00184
5.07E+02	1.28978	0.00180			

We can then plot the depletion results for the MCNP model on the same chart as the results from the SCALE TRITON model we did in Example 3.4.



When we look at the results from both models, we see that the depletion curves are very similar in slope. The systematic difference in k_∞ that is initially present is consistent throughout. The MCNP model has a higher value. If we were trying to verify these results, we would need to determine the exact cause of the differences between the two approaches. Note that the standard deviation for the MCNP model is so small that the error bars are not visible in the plot.

6.5 Universes and Repeated Lattice Structures

Up until now, we have modeled some very simple problems. We have been able to fully define a model in about five cells with a handful of surfaces. However, real problems are often much more complex. Take, for example, a pressurized water reactor fuel assembly (Figure 6.4). The assembly has on the order of 17 x 17 of the unit cell pins we defined in Example 6.4. We could specify all 289 unit cells by making cylinders located on the x and y axes at the appropriate distances. This would be a very daunting task. Fortunately, MCNP has a method for specifying repeated structures in a lattice. This allows us to model

a single fuel pin and a single control rod guide tube and then repeat them in the appropriate arrangement to match the assembly. After we make a single assembly, we can repeat that assembly in an appropriate lattice to make an entire reactor.

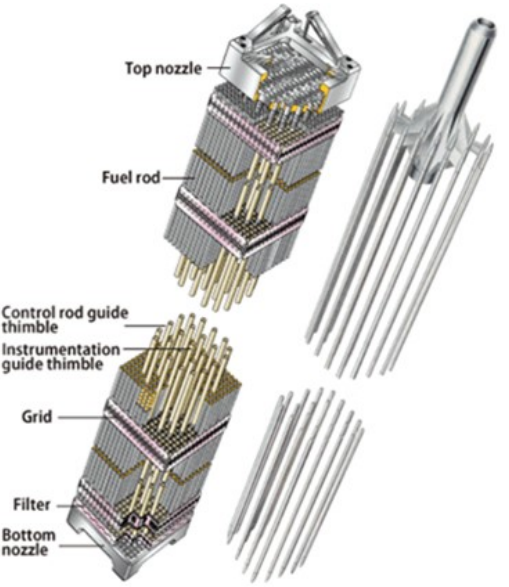
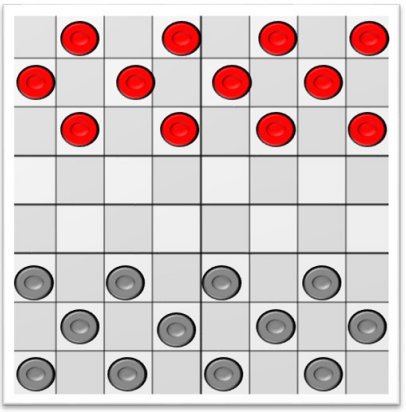


Figure 6.4: A cutaway segment of a PWR assembly on the left and the control rod cluster on the right. The cluster drops through the top of the assembly through the control rod guide thimbles. (Nuclear; 2016)

The method for nesting one geometry in another is accomplished using universes. We could describe the technique as students in a classroom or soldiers in formation. However, because all students and soldiers are not identical, a better analogy would be checkers on a checkerboard. We can think of all the red checkers as being the same and all the black checkers being the same. Although red and black checkers are similar, they are different from each other and are arranged on a square grid – the checkerboard. Let's define the checkerboard as Universe 1. It is a regularly ordered arrangement of equal-sized squares in eight rows and eight columns. We can then define a blank space as Universe 2, a red checker as being Universe 3 and a black checker as Universe 4. If we were to map out the arrangement for the board at the start of a game, we would get the arrangement presented in Figure 6.5.

So far we have modeled the equivalent of a single checker in MCNP. That is the unit fuel pin. In order to make a fuel assembly like what is pictured in Figure 6.4, we will need to first define the other possible unit cells that could go into our assembly. A standard PWR assembly will have fuel pins, control rod



2 3 2 3 2 3 2 3
3 2 3 2 3 2 3 2
2 3 2 3 2 3 2 3
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
4 2 4 2 4 2 4 2
2 4 2 4 2 4 2 4
4 2 4 2 4 2 4 2

Figure 6.5: A normal checkerboard set up for the start of a game. If the board is universe 1, a blank square is universe 2, a red checker is universe 3 and a black checker is universe 4, the map of the sub-universes within universe 1 is on the right.

guide tubes and an instrument guide tube. The control rod guide tubes and the instrument guide tube are empty zirconium-alloy tubes that allow for control rods to be inserted from the top or measuring instruments to be installed from the bottom. They also serve as the vertical support structure for the assembly. The entire assembly is held together with metal spacer grids and will have a nozzle assembly on the top and bottom. For an MCNP model of the assembly, we will normally not model the spacer grids or the nozzles because they don't significantly affect the criticality of the assembly. Therefore, in order to model the assembly, we need to have the arrangement of the rods and the dimensions of the control rod guide tubes and the instrument tube.

Once we have made the unit cells for all of our different types of cells, we will need to assign locations for the rods. For the Westinghouse 17 x 17 design, we will use the arrangement presented in Figure 6.6. We will use the dimensions in Table 6.3. For simplicity, we will use the same clad dimensions for the control rod and instrument guide thimbles as we use for the fuel rod.

To create a repeated lattice in MCNP, make a unit cell for both the fuel and the control rod/instrumentation thimble rods. The unit cells will each part of the cell will have a universe identifier "U=n" where n is a number assigned to that universe. For example, the cells for fuel, gap, clad, and water for the fuel cell would all have U=2 in their definition. After the unit cells are defined, create a cell that is a void the size of the assembly that will be filled with universe 1. Universe 1 will be the lattice. MCNP can make either a rectangular (LAT=1) or triangular pitch (LAT=2) lattice. The lattice parameter is added

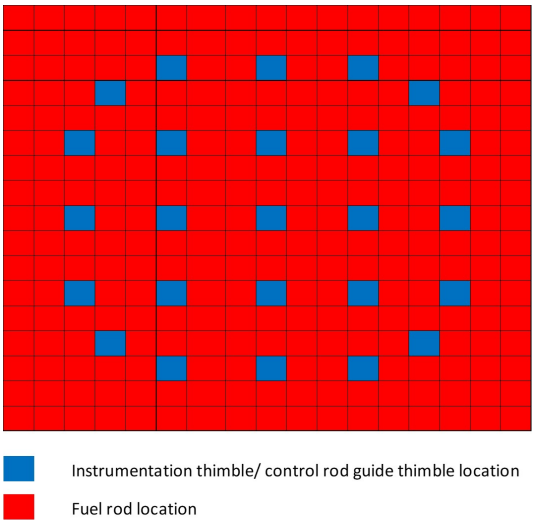


Figure 6.6: Standard rod locations for a 17 x 17 PWR assembly.

Table 6.3: Specifications for Westinghouse 17 x 17 PWR Assembly for MCNP Modeling

Parameter	**Value**
Geometry	Square 17 x 17 matrix
Fuel assembly dimension	Square 21.4 x 21.4 cm
Active fuel height	420 cm
Composition per assembly	Total: 289
	Fuel: 264
	Control rod guide thimble: 24
	Instrumentation thimble: 1
Fuel material	UO_2 at 4.5% enriched ^{235}U
Cladding material	Zircaloy (use zirconium as an approximation)
Gap filler	Helium gas
Fuel average density	95 – 96% Theoretical Density
	Use density 10.96 g/cm^3
Moderator (coolant)	Light water
	Average density 0.7295 g/cm^3
Fuel pellet diameter	0.819 cm
Pellet clad gap	0.0082 cm
Clad thickness	0.0572 cm
Outer diameter of fuel rod	0.95 cm
Pitch (center-to-center)	1.254 cm

to a cell definition and includes the following format:

```
LAT=1 U=1 FILL=-8:8 -8:8 0:0
```

This notation indicates a square lattice, named Universe 1, will be filled from the center location 8 spaces to the negative X and 8 spaces to the positive X and 8 spaces to the positive and negative Y. We could also fill the lattice in the Z direction if we wanted to specify a change in the vertical direction. Example 6.5 details the model for a 17 x 17 Westinghouse PWR assembly with the specifications in Table 6.3.

Example 6.5
Write an MCNP input file for the Westinghouse 17 x 17 assembly detailed in Figure 6.6 and Table 6.3. Complete the KCODE calculation and report the results. Include a plot of the geometry from the top of assembly.

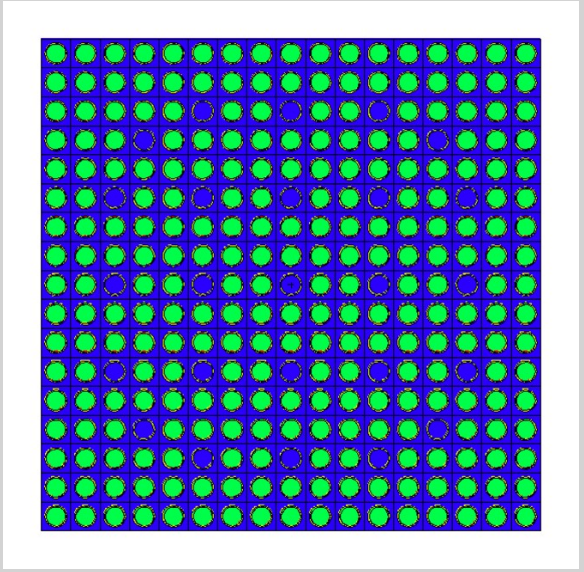
Solution:

```
NE361 Single PWR Unit Cell
C ------------------------------------------------------------------------
C | This is a KCODE calculation for a single PWR assembly.               |
C | Assembly is based on a Westinghouse 17 x 17 assembly with no         |
C | control rods installed.                                              |
C | The cell has a pitch of 1.254 cm, length of 420 cm.  The fuel        |
C | radius is 0.4095 cm.  It is fueled with UO2 at 4.5 w/o enriched      |
C | U-235.   The cladding is zirconium, and the gap is He                |
C ------------------------------------------------------------------------
C ************************CELL CARD BLOCK*****************************
C Cell the size of the outside of assembly - void filled with U=1
1      0 -7 FILL=1
C Cell representing U=1.  Lattice filled with water and U=2 & U=3
2      1 -0.7295 30 -31 32 -33 LAT=1 U=1 FILL=-8:8 -8:8 0:0
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
C Cells that represent U=3 which is the instrument/control unit cell
3      3    -6.407 4 -5 U=3 $thimble cladding
4      1    -0.7295  #3  U=3 $unit cell of water
C Cells that represent U=4 which is the fuel rod unit cell
5      2    -10.96    -6 U=2 $UO2 fuel pellet
6      4    -0.001297  6 -4 U=2 $He gap radius
7      3    -6.407   4 -5 U=2 $fuel pin cladding
8      1    -0.7295   #5 #6 #7 U=2 $fuel unit cell water
9      1    -0.7295   7 -8      $boundary of assembly
99            0      8 -999   $void region outside of assembly
999           0      999 $ bounding sphere graveyard
```

```
C ************************SURFACE CARD BLOCK*************************
C  4 planes for outside unit cell (1.254 cm pitch)
30              PX     -0.627
31              PX      0.627
32              PY     -0.627
33              PY      0.627
C  cylinder water hole/gap radius
4               CZ      0.4177
C  cylinder clad outer radius
5               CZ      0.4749
C  cylinder radius of fuel pellet
6               CZ      0.4095
C  rect parallel piped outside of assembly
7              RPP     -10.658 10.658 -10.658 10.658 -210 210
*8              RPP     -10.659 10.659 -10.659 10.659 -210.1 210.1
C  2 planes for top and bottom of assembly
90              PZ      -210
91              PZ       210
999 so 600  $sphere for stochastic volume calc

c ************************ Data Cards ****************************
c
MODE    N
IMP:N   1.0 8R 0.0 0.0
VOL     1.90574E+05
        1.90574E+05
        1.69098E+03
        1.48677E+04
        5.83405E+04
        2.35247E+03
        1.77224E+04
        9.55999E+04
c
c ****************** Criticality and Burn Control Cards **********
KCODE   2000    1.3     5       55      500
KSRC    1.254     0.0     0.0
M1      8016.71C      -8.88100E+01
        1001.71C      -1.11900E+01
MT1     LWTR.04T $H20 AT 600 K
M2      92234.73C     -4.69781E-05
        92235.73C     -0.039186291
        92238.73C     -0.842188828
        8016.73C      -0.118577903
        $UO2 4.5% at 1200K
M3      40000.58C 1.0 $ZIRCONIUM (APPROX. FOR ZIRCALLOY)
M4      2004.71C  1.0 $ Helium
PRINT
C
C  ********************************************************************
C Uncomment these cards to do the stochastic volume tally.
C Note you must comment out the kcode, ksrc, and vol cards above.
C You must change the importance from 1.0 7R 0.0 0.0 to 1.0 8R 0.0.
C Also, remove reflected boundary conditions on surface 7
C
C Wgt calculated as area of xs of plotting sphere (pi R^2)
C
C Vol    1 1 1 1 1 1 1 1
C F4:N   1 2 3 4 5 6 7 8
C VOID
C SDEF   SUR=999  NRM=-1  DIR=D1  Wgt= 1130973.35529
C SP1    -21 1
C nps    10000000
```

We used the stochastic volume calculation to get the volumes. The KCODE resulted in $k_\infty = 1.40115 \pm 0.00145$

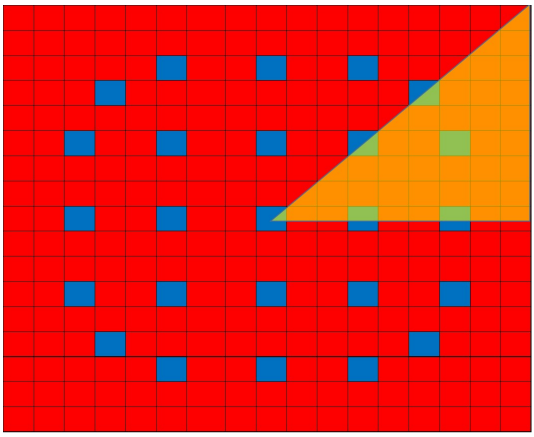
When we plot the model we get:



If we look at the lattice structure of our assembly, we will see that there is symmetry to the arrangement. Figure 6.7 is an illustration of the symmetry within the fuel assembly lattice and assigns numbers to each of the input values for the cells in the order in which they are entered in the input.

Because of the symmetry and the reflected boarder on the model, we can tally information on $1/8^{\text{th}}$ of the cells in the lattice and know that the results will be repeated. One of the most important values we can get for any assembly is a measure of the power in each fuel rod and a comparison to the total average power in the assembly. When we find the maximum ratio of pin power to average power, we will know the peak-to-average power for the assembly which is an important design characteristic. For our simple reflected model, we should have a very uniform power across the assembly, especially at BOL when no burnup has occurred. We know that flux is proportional to power so we can use an F4 volumetric flux tally on each of our fuel pins. We could also use an F6 energy deposition tally to examine the fission energy deposited in each cell.

To tally on a repeated lattice, we have to set our tally name followed by an identifier of the cell that we want to tally. Because we are tallying within a lattice, we will need to use a slightly different convention.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85
86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102
103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136
137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153
154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187
188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204
205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221
222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238
239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272
273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289

Instrumentation thimble/ control rod guide thimble location

Fuel rod location

Region of 1/8th symmetry

Figure 6.7: Illustration of the 17 x 17 assembly region of symmetry.

```
F4:N    (5<2 [17])
        (5<2 [33])
        (5<2 [34])
        (5<2 [49])
        (5<2 [50])
        (5<2 [51])
        (4<2 [65])
        (5<2 [66])
        (5<2 [67])
        (5<2 [68])
        (5<2 [81])
        (5<2 [82])
        (5<2 [83])
        (5<2 [84])
        (5<2 [85])
        (4<2 [97])  continued for all of the highlighted cells
```

The input above is for the first 16 cells in the highlighted area in Figure

6.7. The first line gives the command to conduct an F4 tally on cell 5 (fuel pin) within cell 2 (assembly) at location 17 in the lattice. The last line is for cell 4 (water cell for the control rod/ instrument thimble). We would extend the input for all of the highlighted unit cells to get an output for the entire assembly.

Once we are done, the output will give tally information in the form of:

```
cell (5<2[8 -8 0])
                    7.91930E-07 0.0202
cell (5<2[7 -7 0])
                    7.97535E-07 0.0171
```

This output continues for all the cells. Note that the first cell is identified by the coordinates [8 -8 0] which is the location of pin 17. The positive sense of y is inverted from what we present in Figure 6.7 but the symmetry for the model is maintained.

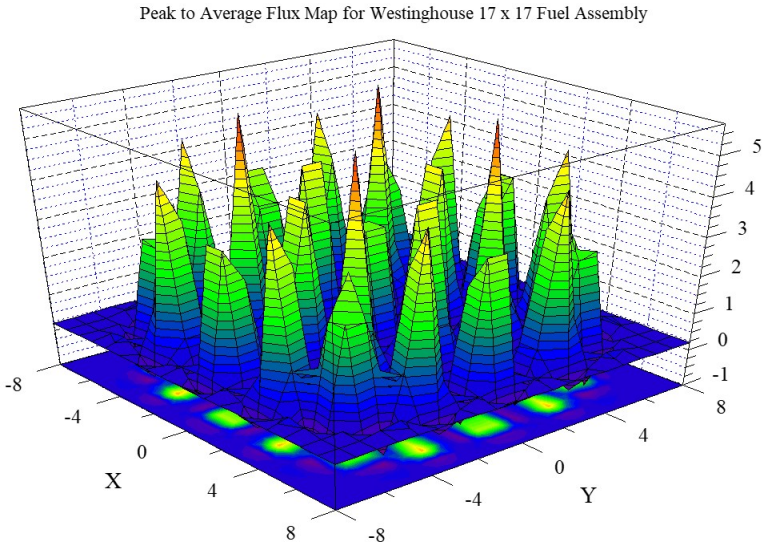


Figure 6.8: Peak-to-average flux map for 17 x 17 assembly based on flux tally. Note the high peaks are at the control rod / instrument guide tube locations.

Now that we have the tally information, we can use a spreadsheet to copy the information to an entire grid of x, y, and z coordinates for the whole assembly. We can also take an average for the assembly and then calculate the peak-to-average flux for each cell. Finally, we can make a 3-D plot for the peak-to-average flux at BOL which is presented in Figure 6.8. Notice how the flux is strongly peaked in the control rod and instrument guide tubes. This is an artifact of using the flux tally and the total flux is much higher in the water in the empty control rod guide tubes. If we zero out the results for the control

rods and repeat the plot for only the fuel we get Figure 6.9. This is a representation of the power in the cells generated from fission. The highest power fuel rods are closest to the empty control rod guide tubes which makes sense because there is more moderator in those regions. The highest peak-to-average power is around 1.15 which is acceptable for design purposes.

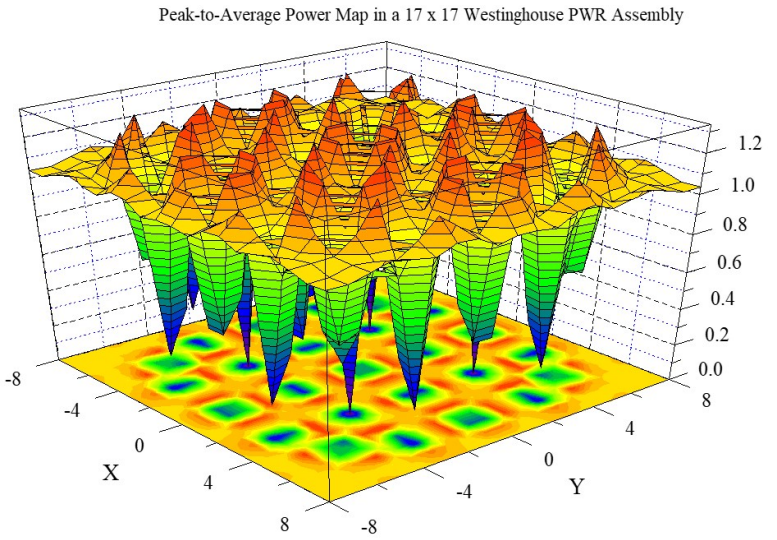


Figure 6.9: Peak-to-average power map for 17 x 17 assembly. The highest power cells are the closest to the empty control rod guide tubes.

6.6 Using Vised for Live Particle Tracking

Up to this point, we have used the Visual Editor Program to plot the geometry of our model in MCNP. VISED has many other features that are useful for visualizing the working MCNP model. One of those is the plotting of tracks and particles. A couple of steps are necessary to make VISED work with MCNP6.2 beyond just plotting. For VISED to run MCNP input files, it must be able to reference the appropriate cross section directory file, xsdir. When a user starts MCNP, he or she will open the MCNP command prompt. When properly installed, this prompt will open the environmental batch file which is normally located in the same directory as the mcnp6.exe file. The environmental batch file tells the program where the various files needed for MCNP calculations are located on the computer. When VISED is opened as a plotter only, the environmental batch file is not run, and it cannot execute MCNP files.

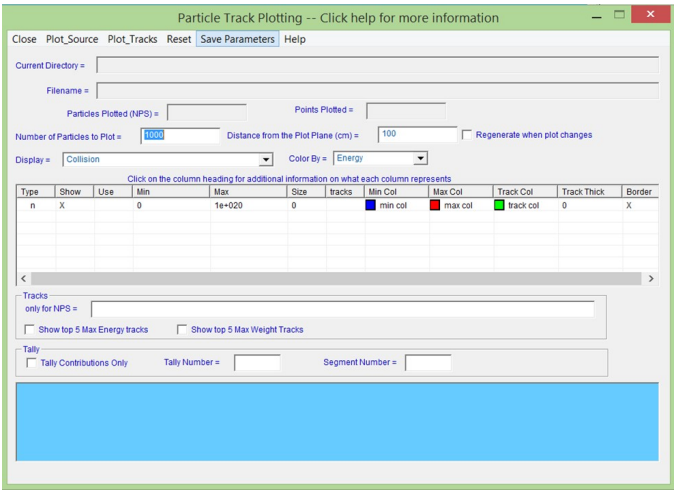
Therefore, in order to plot particles, VISED should be opened from the

MCNP command prompt instead of using the desktop icon. This can be done by opening the MCNP command prompt and navigating to the directory where the executable visedX_24.exe is located on the computer and then entering the executable name. Remember, you cannot use the regular command prompt as it will not run the appropriate environmental batch file when opened. To streamline the process, you can copy the MCNP command prompt shortcut and modify it. Go to the properties for the copied shortcut and change the "start in" location to where the visedX_24.exe executable is located. To avoid confusion between the original and the copied prompt, you can rename the copied prompt and even give it its own icon such as:



Once you run the executable, visedX_24E.exe, the Visual Editor will open as it normally does. The difference is that it will now execute and run MCNP files just like MCNP6.2.

To plot MCNP particle information, you can open a file like you normally would for plotting and then adjust the scale and origin to the area of interest in one of the windows. Next click on "Particle Display" from the top menu and then select either "Plot Particle Tracks" for a fixed source problem or "Plot KCODE Source Generation Points" for a KCODE problem. For a fixed source problem, you will get the following pop-up window:



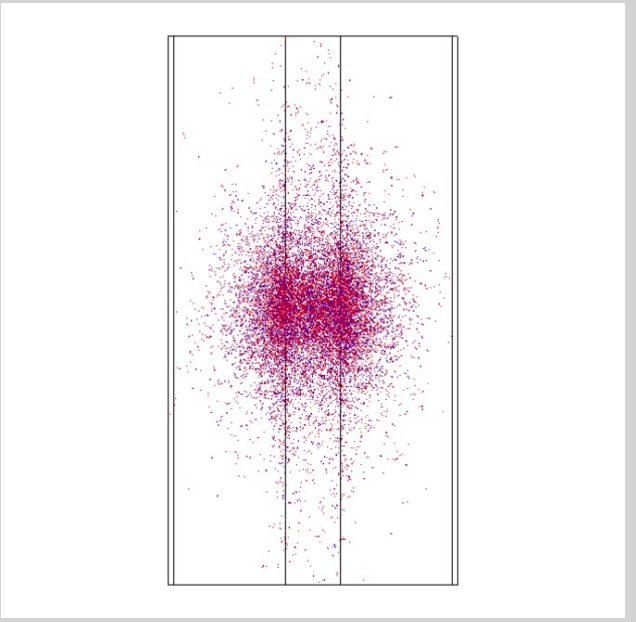
You can navigate through the options to optimize the plot for the number of particles, how to color the points, whether to plot tracks etc. The options are well-designed as you can focus your plotting on the specifics for your problem.

For instance, if you are only interested in scatters, you can filter out absorptions or vice versa. You can also bin the tracks by energy so you can get a visual representation for how the particles are transmitting through the material.

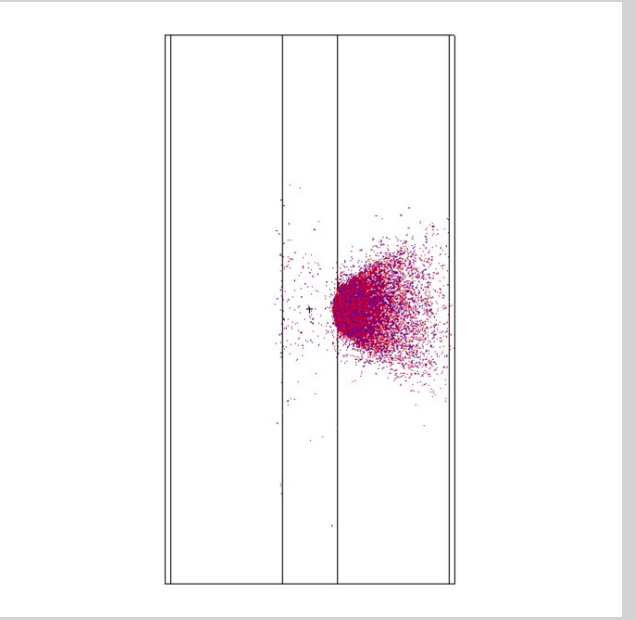
For the KCODE model, you get a different pop-up window that allows you to select the number of cycles to plot the source points and you can plot the points as a cumulative or an animated plot where the points for fission sources pop up as the fissions occur. Example 6.6 shows the use of plotting for the pig detector model for both the isotropic point source and the conical source while Example 6.7 shows the VISED plotting for our Westinghouse 17 x 17 assembly we made in Example 6.5.

Example 6.6
Use the VISED program to plot the source particles for the pig detector model made in Example 6.1 for the isotropic point source from the XY view. Compare to the same plot for the conical source from the XY view.

Solution: Start VISED from the VISED command prompt to activate the particle plotting capability. Load the MCNP model for the isotropic point source and select the Particle Display/Plot Particle Tracks from the menu. Change the Number of Particles to Track to 10000 so you can get more resolution in the plot. Click on Plot_Tracks on the menu at the top of the Plot Particle Tracks window and you will see the plot develop. When it is done running it should look like:



Repeat the process for the conical source and it should look like:



The two plots are exactly what we would expect to see. The first shows the source particles in an isotropic distribution and the second is a conical source directed along the positive x-axis toward the detector.

It should also be noted that there are very few particles making it to the edge of the pig without collision. That is a good thing because it is a shield. It also emphasizes why we had such a low count rate on our tally that was one meter from the source and why we had to use variance reduction to get good statistics from such a small problem set.

Example 6.7
Use the VISED program to plot the KCODE source points for the Westinghouse 17 x 17 assembly in example 6.5 from the XZ viewpoint and comment on the results.

Solution: Once again, load VISED from the command prompt. Load the MCNP model for the assembly and select Particle Display/Plot KCODE Source Generation Points from the menu. Select cycles 50-55 on the range to plot and then select Run_To_Generate_Plot_Data from the menu at the top of the KCODE Source Generation window. The model will run in the window. When

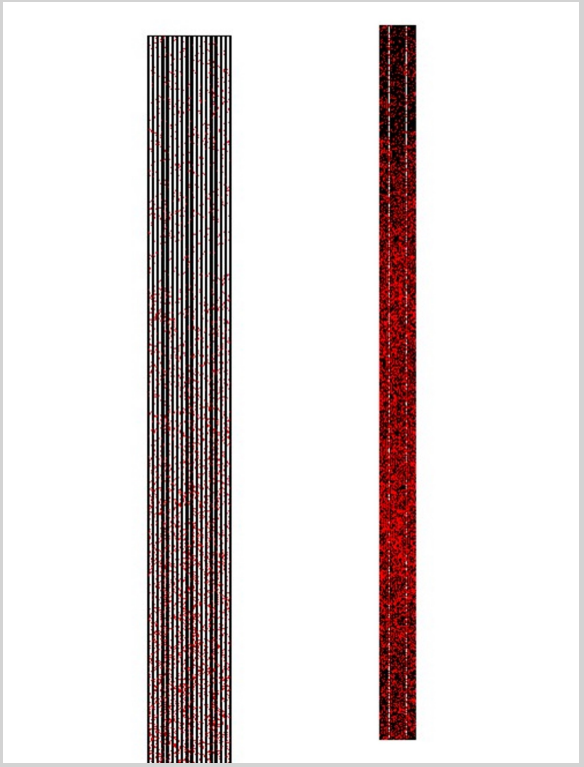
it is complete, click on Plot_Saved_Data. The plots for the entire length and only the top half of the assembly side by side look like.



Notice how the points are concentrated in the center of the assembly and do not reach the ends of the assembly in significant numbers. This is a potential problem with the model. This means that our fissions are only occurring in the middle of the assembly for the most part. We can do one of two things to help out. We can run more active cycles to get saturation in the model or we can add more source points along the length of the fuel. We will go ahead and add source point by changing the KSRC card to:

```
KSRC    1.254    0.0     0.0
        1.254    0.0    -100.0
        1.254    0.0     100.0
```

Recall that the x coordinate must be outside of the center cell because it needs to be in a fueled region and the center cell is water. We now have points roughly halfway up and down from the centerline. We run it again and the following plots:



For the second set of plots, the source particles are more evenly distributed along the length of the assembly. Note that the final k_∞ value is very similar between the two models. This is because the fuel is all the same axially and the assembly has a reflected boundary. It really does not matter how long the fuel assembly is in this model beyond a length of a few mean free paths of the neutron. It would matter if we were doing a burnup model and looking at how the entire fuel changed with respect to time. For a more realistic model, we would look at adding an axial blanket of low enriched or natural uranium fuel to serve as a reflector blanket.

Bibliography

Kulesza, J. A., Adams, T. R., Armstrong, J. C., Bolding, S. R., Brown, F. B., Bull, J. S., Burke, T. P., Clark, A. R., Forster III, R. A., Giron, J. F., Grieve, T. S., Josey, C. J., Martz, R. L., McKinney, G. W., Pearson, E. J., Rising, M. E., Solomon Jr., C. J., Swaminarayan, S., Trahan, T. J., Wilson, S. C. and Zukaitis, A. J. (2022). *MCNP Code Version 6.3.0 Theory User Manual.*

Nuclear, W. (2016).
URL: *www.world-nucler.org*

Shultis, J. K. ; Faw, R. E. (2011). *An MCNP Primer*, Department of Mechanical and Nuclear Engineering, Manhattan, KS.

6.7 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- Running VISED from Command Prompt

6.8 Problems

1. Modify the pig detector model from Example 6.1 by changing the lead to iron in the natural isotopic composition and density. Change the geometry of the pig until the flux at the detector matches the value for 4 cm of lead. Plot the geometry of the iron pig using either VISED or the integrated plotter.

2. Plot the particle tracks using VISED for the model in Problem 1. Change the source to a conical source and plot the source again.

3. Divide the iron region in the model from Problem 1 into 3 equal regions. Perform a stochastic volume calculation for the model and report the values and uncertainties.

4. Use geometry splitting in the three-region model from Problem 3 to reduce the variance in the problem. Report the results and compare to the model in problem 1 and the conical source in problem 2.

5. Change the fuel composition in Example 6.3 from a standard UO_2 mixture to a standard mixed-oxide (MOX) fuel consisting of plutonium and uranium. The fuel should be at 4.5% enriched in fissile material. Show your calculations and references for the determination of which isotopes and concentrations you used to get the MOX composition. Report the k-infinity for the MOX-fueled model.

6. Besides the fuel temperature coefficient presented in Example 6.3, another important coefficient is the void coefficient. This coefficient compares the criticality as a function of void in the moderator. Change the density of the water in the unit cell used in Example 6.3 from 1.0 g/cm^3 down to 0.5 g/cm^3 by a step interval of 0.05 g/cm^3. Run each model and record and plot the results. Use the slope of a trendline to estimate the void coefficient as the change in reactivity as a function of percent void. Is the void coefficient positive or negative? Does it make sense? Does the void coefficient make the reactor more inherently stable or unstable and why? How does it compare to the unit cell model in SCALE?

7. Calculate the temperature coefficient for the MOX-fueled unit cell created in Problem 5 and compare it to the results in Example 6.3.

8. Calculate the void coefficient for the MOX-fueled unit cell created in Problem 5 and compare it to the results in Problem 6.

9. Complete the depletion analysis performed in Example 6.4 on the MOX-fueled unit cell created in Problem 5 and compare the results to Example 6.4.

10. Complete a burnup depletion on the single assembly model in Example 6.5. Remember to include the material volume in the BURN parameters for the sum of all the fuel in the assembly.

Chapter 7

Introduction to CAD

The primary purpose of engineering is to design something for a practical purpose. This is what separates engineering from art. Everything in an engineered product should have a purpose. Most engineers do not physically construct their design. It is essential that engineers can accurately and completely convey their design intent to the machinists or fabricators that will manufacture their product. Even if they are building their entire design themselves, accurate plans and drawings are necessary to ensure that it is made to specifications.



Figure 7.1: West Point cadets learning drafting from the Department of Drawing in 1904. (Stockbidge; 1904)

For years, students took courses in mechanical drawing and drafting. They learned the skills to work with drafting instruments to make accurate scale drawings for engineering. Even after personal computers became relatively ubiquitous in the late 1900's computer aided design programs were not widespread. They were very expensive and required so much computing power that they were only available to a few users within large engineering firms or special academic settings operating on dedicated machines. As technology continued to improve, CAD programs, like Solidworks®, began to be offered for use on Windows computers at a reasonable price. Now CAD has completely replaced the pencil and ruler for mechanical drafting. With the widespread use CAD and CAD classes, there are many excellent resources in print and online that will help the user. One such resource is William E. Howard and Joseph C. Musto's Introduction to Solid Modeling Using Solidworks® 2014. (Howard and Musto; 2015) This text will not provide the complete depth of books that are dedicated to the subject. It will provide some basic examples and instruction in CAD specifically for nuclear engineering projects.

7.1 Basic Construction of Solid Bodies

Before we go into the actual methods of using the Solidworks® program, we will discuss the general approach to 3-D modeling. The primary difference between drafting and computer aided design is the concept of making a solid body or 3-D rendering of an object.

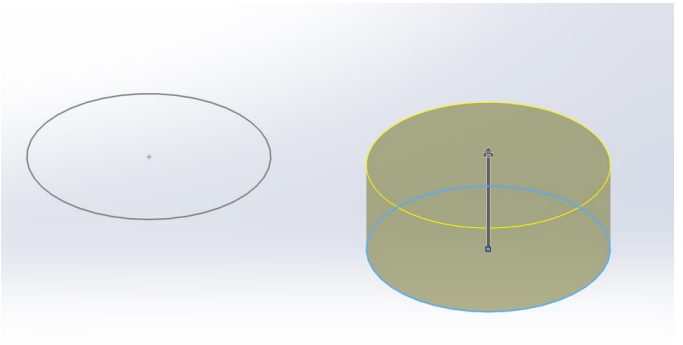


Figure 7.2: The basic concept for constructing a solid body is to draw a 2-D profile and then apply a boss to create height.

The great advantage of using CAD is there are simple ways to obtain high-quality scale 2-D, and 3-D design drawings as well as realistic 3-D images of completed objects. Advanced uses include checking and analyzing mechanisms and how parts work together and even designing molds and manufacturing techniques. Additionally, 3-D solid body models are uniquely suited for computer aided manufacture (CAM) and 3-D printing technologies.

All solid bodies are born from some form a 2-D construct. An easy example would be a cylinder. To construct a simple cylinder as a solid body, you first draw a circle and then apply what is known as a "boss" to give it height.

Once the basic shape is built, you can then use a Cut-Extrude feature to remove material and further refine the shape. For example, a simple cylinder-shaped container might have one cylinder for the outside shape, such as Figure 7.3, and then another cylinder is used to cut the material out of the middle.

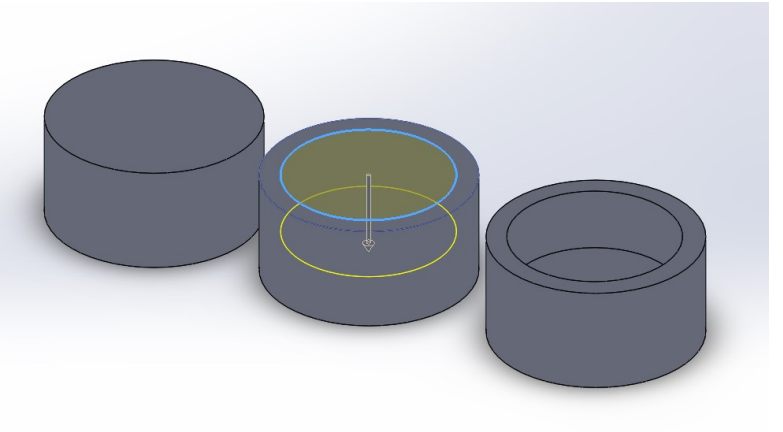


Figure 7.3: Once the basic outside shape is created, draw interior contours and then use the Cut-Extrude feature to remove material.

Features can then be added to the base structure by adding additional bosses or several small objects, or parts, can be combined to make an assembly. Often an object does not have a shape that lends itself to either a straight boss or cut to define it. For example, say we have an object that looks more like a vase or say a reactor cooling tower as in Figure 7.4. Making a cylinder and adding multiple bosses and cuts would be an unreasonable approach to develop such a shape. Fortunately, there is an easier way made possible by making a revolved boss.

Other tools are available for making bosses that are swept out along a feature or lofted between two features as well. Once you have the general shape, additional refinements can be made to include rounding edges and corners with fillets or chamfers. You can also add details like hinges or threading to prescribe hardware components.

There are many other features within the CAD program that will allow you to rapidly create the drawing. These include the ability to repeat structures in

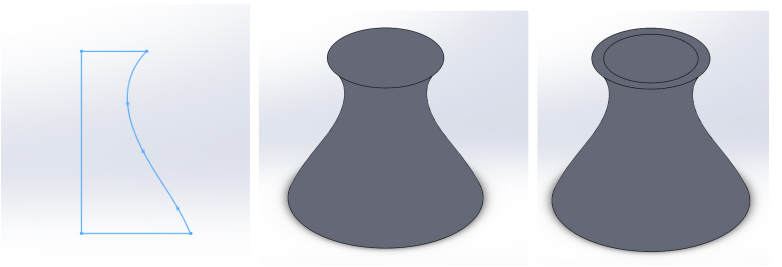


Figure 7.4: Use a revolved boss to create solid bodies from a profile that does not lend itself to a straight boss or cut. A revolved cut also works in the same manner.

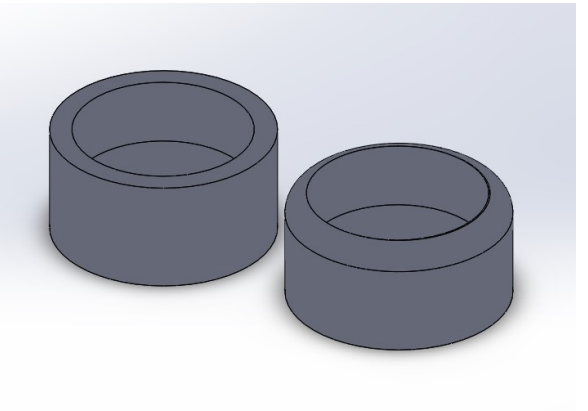


Figure 7.5: Apply a fillet to round off corners as needed for the design.

a linear or circular pattern. This is essential if you want to make a circular bolt pattern or perhaps design a spacer grid for a reactor assembly.

Although not always necessary, a great feature of CAD is the ability to model the object using the colors, textures, decals, and lighting for the final object. This photo-realistic effect allows the designer to visualize exactly how the component will look based upon the materials and finishes used.

The final product of the process should be a detailed scale design drawing that includes all the dimensions and other information that would be required for production. A good design drawing requires no explanation. Any machinist or manufacturer should be able to take your drawing and build it exactly as you prescribe. Creating a drawing from a solid body is an essential built-in feature of the CAD program. The great advantage to using the software is that the drawings are linked to the parent object. That means that if you go in and edit or change the solid body, the subsequent drawings are automatically updated

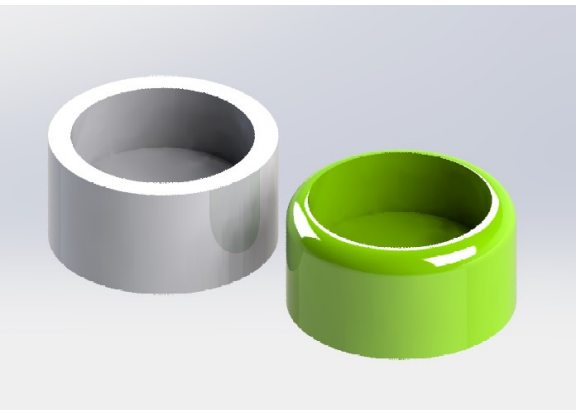


Figure 7.6: Apply a photo-realistic effect to the image to get a better idea of how the final image will look.

with the new shape, part, or dimension.

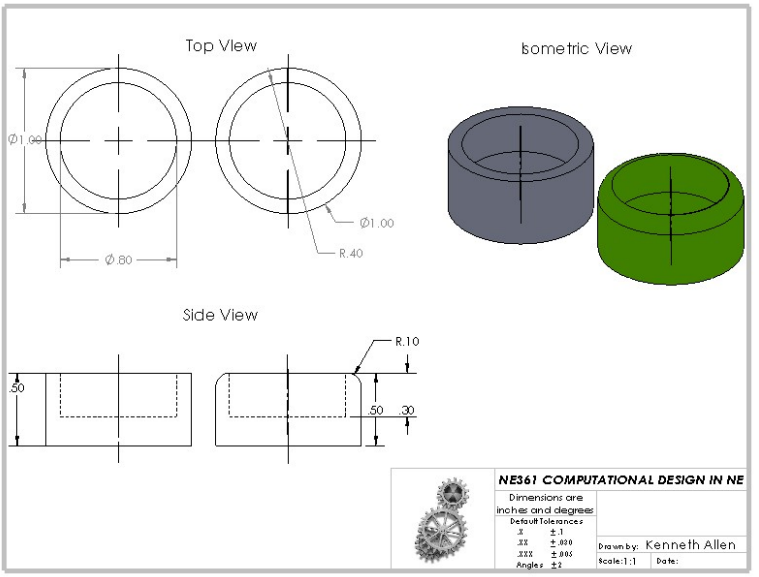


Figure 7.7: Mechanical drawing of parts complete with dimensions and drawn to scale.

As we proceed through this chapter, we will explore the basics for creating a part in Solidworks®, scaling and changing its appearance, creating an assembly from a group of parts and creating a mechanical drawing. From these basics,

you should be able to use other references to acquire skills and tips to improve the efficiency and accuracy of your CAD abilities.

7.2 Formatting Part and Assembly Templates Solidworks®

To begin formatting the part and assembly templates in Solidworks®, first open the program and select "File" – "New" and click on "Part". The default workspace will appear as in Figure 7.8. It will include a blank Graphics Area in the middle, the Command Manger toolbar at the top, the Task Pane on the right and the Feature Manager Design Tree on the left. In the lower left of the Graphics Manager is the Reference Triad which indicates the orientation of the object in x-y-z coordinates.

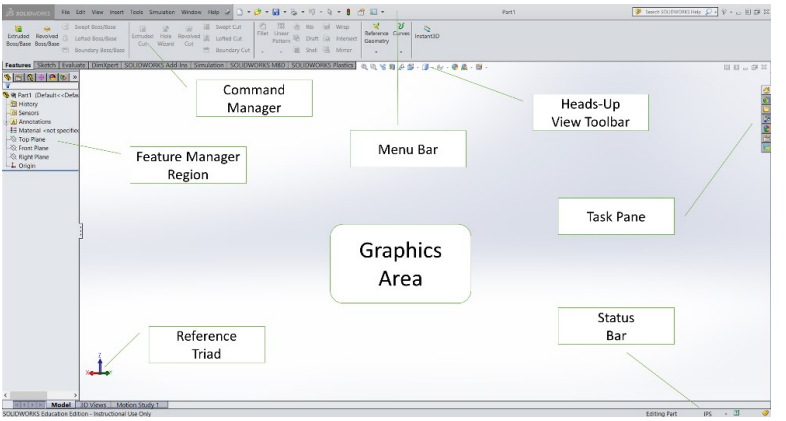
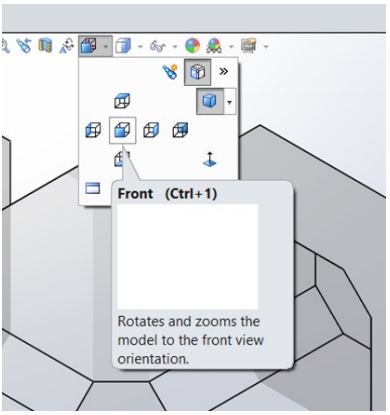


Figure 7.8: The basic layout of the various menus and regions in the Solidworks® workspace. (Howard and Musto; 2015)

The default for Solidworks® is a y-up orientation where the positive y-axis is vertical, positive x-axis is right and the positive z-axis is out of the page. However, this orientation is not the standard for most computer aided manufacturing equipment such as CNC mills and 3-D printers where the z-axis is vertical. For this reason, we will change the default reference triad axis to be a z-up orientation such as:

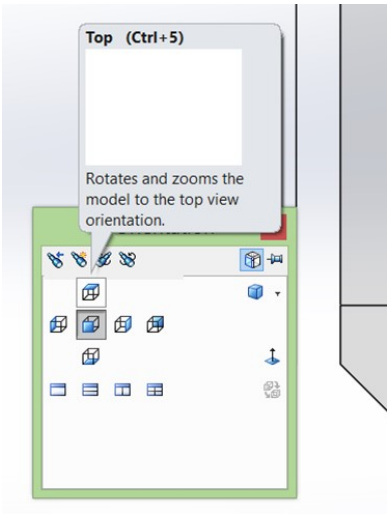


In order to change the orientation, go to the heads up view toolbar at the top of the Graphics Area and change the view to the "Front".



The image will change to reflect the triad with the z-axis directed straight outward and the y-axis up. Right-click on the triad and select "View Orientation". You will get another pop-up with a variety of views to select.

Select the third icon on the top row to "Update Standard Views". The program will then tell you to select the view you want to assign to the current orientation. Select the "Top" orientation. It will give a warning that you will have to select "yes" to change. This will change the x-axis as up.



After changing the axis, the drawing planes in the Feature Manager Design Tree will be incorrectly named. The "Top" and "Front" planes will be backwards. This is easily corrected by right clicking on the name "Top Plane" and

going to the properties and renaming it. Be aware that you will not be able to name it "Front Plane" right away because there is already a plane called "Front Plane". You can call it just "Plane" and then rename the "Front Plane" to "Top Plane" and then go back and rename the "Plane" to "Front Plane." Now the workspace will be correctly aligned. If we select the "Features" tab at the top and then select "Extruded Boss/Base" we will see the three drawing planes pop up at the same time which will confirm that everything is correct as in Figure 7.9.

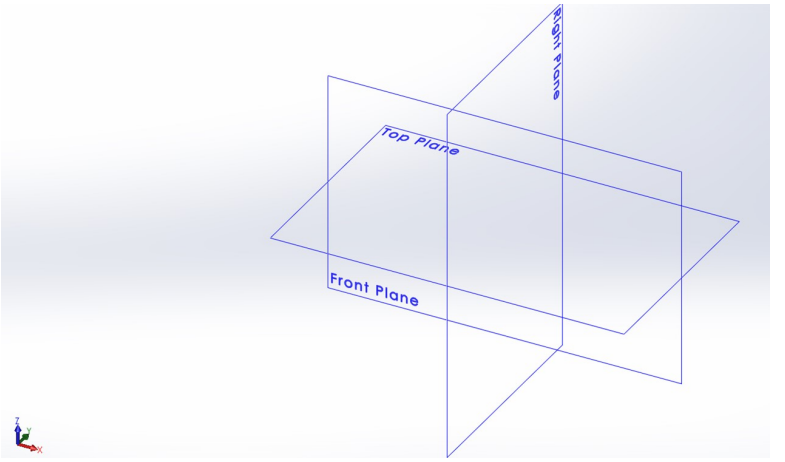


Figure 7.9: Drawing plane and reference triad set to a z-up orientation.

Now that we have the part orientation correct, we can use the "Save as" as a "Part Template" and replace the current part template. It will prompt and ask if you want to overwrite the current template – select "yes". We should now open up a new file as an assembly and repeat the process for the triad, the names for the planes, and the assembly template. Now the orientation for our parts and assemblies will match and will translate correctly for use in CAM and 3-D print applications without significant manipulation.

7.3 Creating a Part in Solidworks®

As mentioned in Section 7.1, the process of building a solid body involves drawing a 2-D sketch and manipulating it to give it height or depth. For our first project, we will draw a radioactive material storage pig similar to what we designed in Chapter 6 in MCNP. Instead of having a simple open cylinder like we had in Figure 6.1, we will go ahead and create two parts, one will be the container and the other will be a fitted lid that can be lifted off with a handle. Later in this chapter we will create an assembly and a mechanical drawing from these parts. The critical dimensions and materials are presented in Table 7.1.

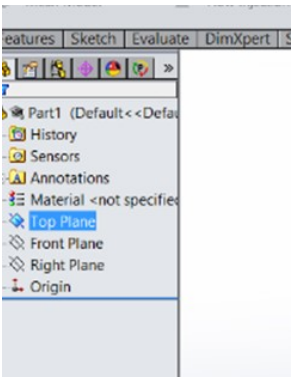
Note that the model is more refined in shape than the simple model we made in MCNP. It is shorter, includes a cap, a flange to bolt the cap on, a handle on the cap etc. However, to make our first design simple enough for an introduction, we won't model the separate layer of copper or include the fasteners. We will also use whole centimeters for dimensions as much as possible.

Table 7.1: Critical Features of Radiation Storage Pig

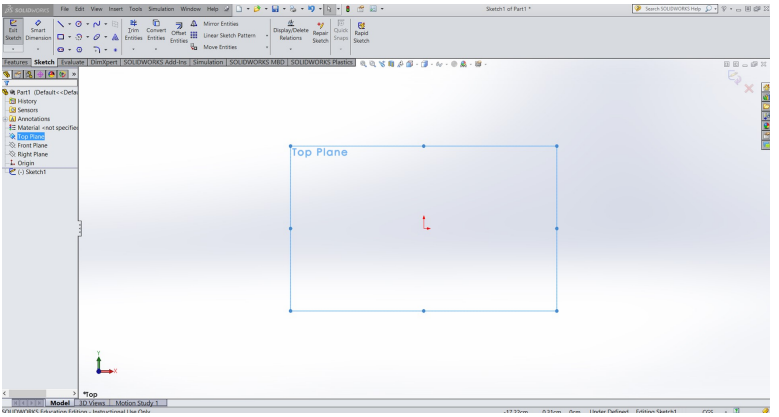
Feature	**Value**
General Shape	Right cylinder
Materials	-Lead core for shielding -Copper coating on all surfaces
Outer Radius Shield (cm)	5
Outer Radius of Flange (cm)	7
Lead thickness (cm)	4
Inner Radius (cm)	1
Cap thickness (cm)	1 cm flange and 3 cm plug
Bolt-hole Diameter (cm)	0.5
Number of Bolt Holes	6
Thickness of Bottom (cm)	4
Height of base (cm)	9

Now that we know the basic dimensions of the structure, we can start to model the solid body. When we open Solidworks®, the default dimensions will be inches-pounds-seconds. This is good if we are working with CNC machinery in the United States which is normally set up for SAE units. However, because our original MCNP model was in centimeters and all of the dimensions in Table 7.1 are in centimeters, it makes sense to change our settings for this part to centimeters-gram-seconds. To do so, open Solidworks® and create a new part. In the lower right side of the window, you will see information in the Status Bar area that says "Editing Part" and "IPS". If you click on "IPS" you can change the units to "CGS" for this part. The default will still be "IPS" unless you change the part template. We will leave the template alone.

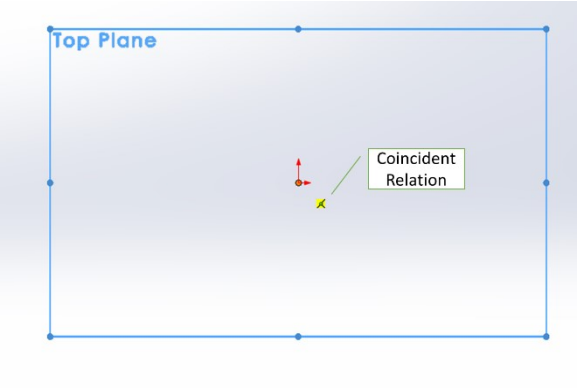
Now we can begin to sketch out our part. We should think about how we want to build it. Should it be centered on the origin, built up from the top plane, or some other position? If we were making a part to be milled, it may make sense to set the origin on the top of the part so the probe for the CNC machine could locate itself or "touch-off" from the top of the part. Because we are just making a drawing, it makes sense to build up our part from the top plane centered on the origin. This will make drawing the objects easier and give a good reference plane when we want to apply a boss or cut to a shape.



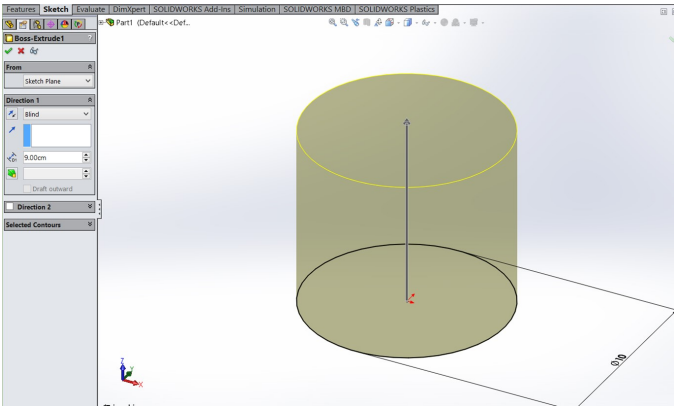
Go ahead and click on the "Top Plane" in the Feature Manager Region. The top plane will show up in the Trimetric view and will be blue in color indicating it is active. When you select the "Sketch" tab followed by the "Sketch" button, the plane will rotate to a 2-D view with x to the right and y upwards.



Now we can pick up the circle sketch tool to draw our first shape. Note that many of the shape buttons have drop down menus indicated by a small downward facing triangle. If you click on the triangle, you can select a variety of types of that feature. For example, you can choose between a center point circle (default) or a diameter circle. For our project we will be using a center point circle. When we click on the tool, the pointer turns to a pencil with a circle. Move the cursor to the origin indicated by red in the center of the plane. When the point of the pencil gets over the origin, a small yellow box will pop up next to the pencil with a cross in it.



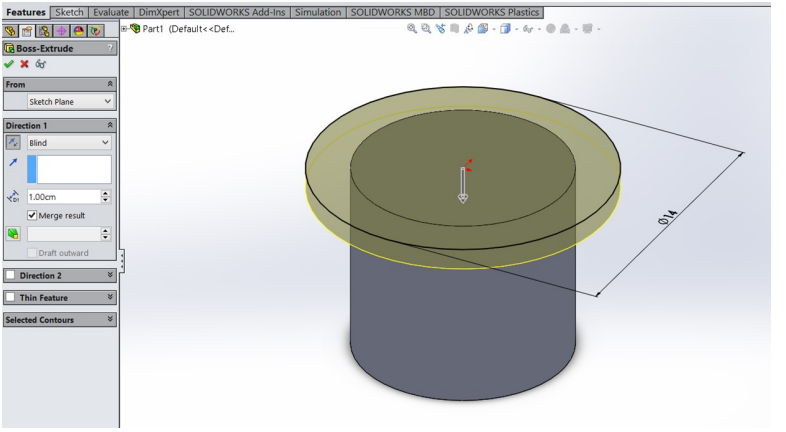
This symbol indicates that it is a coincident relation meaning that the center of the circle will be coincident with the origin. This relation is extremely useful as we attempt to attach and line-up various shapes in our part. Click on the origin and move the mouse outward. The circle will be drawn and a box will pop up showing the radius in cm. Do not try to hit the dimension right on. It is nearly impossible and the program has much better ways to get the accurate scaling that we will use next. If you look at the Feature Manager section on the left of the screen, you will see properties of the circle to include its origin and radius. At this point, we could click in the radius box and change it to our desired dimension. As an alternative approach, click on the "Smart Dimension" tool next to the "Exit Sketch" button and then click on the circle. Move the mouse away from the circle and a diameter dimension will appear. When you release the mouse key a pop up will appear asking for the dimension and units. Change it to 10 cm. Click the check mark on the Feature Manager region and the circle and dimension marking will turn from blue to black.



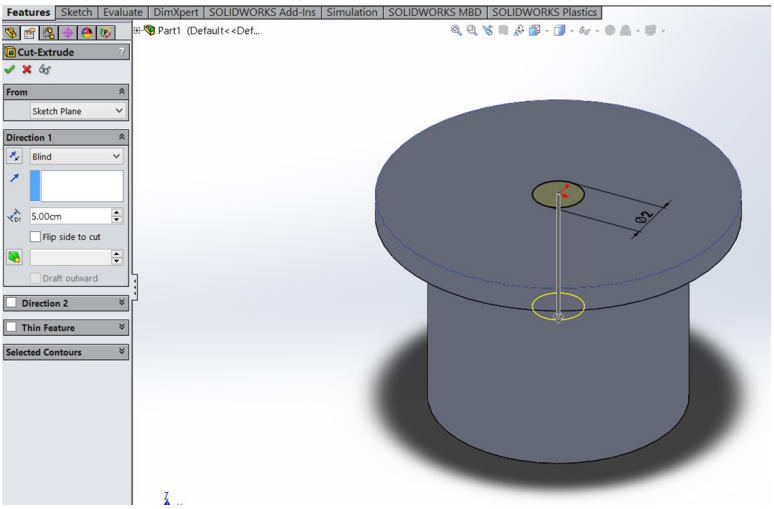
Now we will add the third dimension by selecting the "Features" tab and click the "Extruded Boss/Base". The sketch will rotate to the trimetric view

and automatically preview a 1 cm boss. Change the dimension for the boss to 9 cm.

Click on the green check mark and the outside of the base will turn grey. We can add the flange for bolting the lid on by clicking on the top of boss. This will allow us to sketch on the surface. Click on the "Sketch" tab and the "Sketch" button. Select the center circle tool again and draw a circle on the top of the base centered on the origin. Use the smart dimension tool to change the diameter to 7 cm. Click on the "Features" tab and the "Extruded Boss/Base" button. Change the view from the top to the trimetric view to get a better sense of the boss. The default dimension will add vertically to the previous base. We will change the direction of the boss to go down by clicking the double arrow just below "Direction 1" in the Features Area. The boss preview will now go downward. Change the depth of the boss to 1 cm.

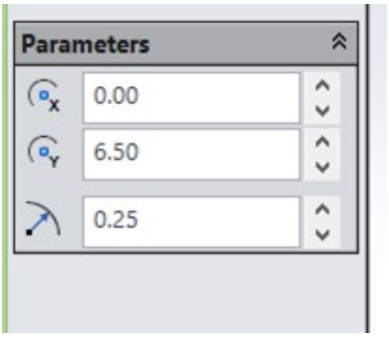


Now we can cut out the center hole in the base. We can easily do this by first selecting the top view of the model again and then clicking on the top of the cylindrical boss. Select the center circle tool again and draw a circle on the top of the base centered on the origin. Use the smart dimension tool to change the diameter to 2 cm. Click on the "Features" tab and the "Extruded Cut" button. Change the view from the top to the trimetric view to get a better sense of the cut.



The preview shows a cut from the top to the bottom of the base. We can change the depth by changing the 9.0 cm to 5 cm which will leave 4 cm of material on the bottom – the same thickness as the side of the shield.

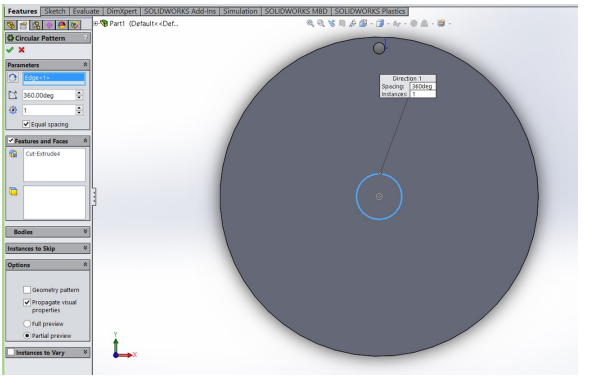
Now the total shape is defined, and we can add the bolt holds. To do this, we will make one hole near the top of the flange. We will use the same procedure we used to cut the center hole but we will specify the origin for the hole at x=0 and y=6.25 with a radius of 0.25 cm in the feature portion of the circle.



Then use the same "Extruded Cut" feature to cut a hole all the way through the flange.

Now you could repeat the hole and cut manually by specifying the x and y position for the next 5 holes. However, we can take advantage of the symmetry in the design and use the repeated feature option in Solidworks®. You do this by selecting the cut-extrude for the bolt hole and then clicking on the drop-down arrow next to the "Linear Pattern" button on the "Features" tab to reveal

the "Circular Pattern" option. Click on the circular pattern and then click on the edge of the circle for the 2 cm diameter center hole. This will define the feature that the bolt hole will be patterned around.



Now we can just change the number of instances from 1 to 6 and make sure we have checked the option for "Equal Spacing". We will now have 6 evenly spaced holes around the outside of the flange. Our basic part for the base is complete as shown in Figure 7.10. We will save the file as "PigBase.sldprt" and we will proceed to make the lid for our pig shield.

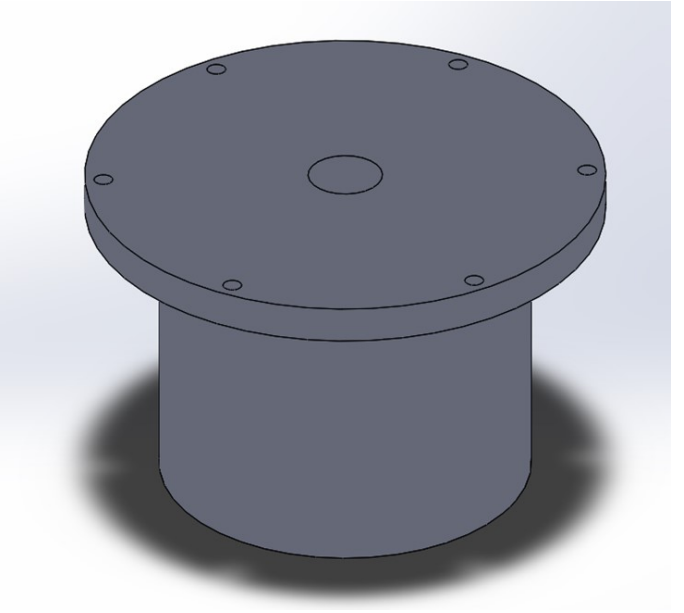


Figure 7.10: Final pig shield base solid body.

The procedure for making the cap is the same as the base up until the point where we add the handle. Start with a new document. Make sure to change

the units to CGS. Draw the plug portion of the cap first. Make the diameter of the plug smaller than the 2 cm diameter of the hole so it will fit inside. In this case make the plug 1.8 cm in diameter. The final finished plug will provide the same 4 cm of shielding as the sides and bottom of the base. The empty space in the pig will be a right circular cylinder with a height and diameter of 2 cm. Without the handle, the cap for the pig looks like Figure 7.11.

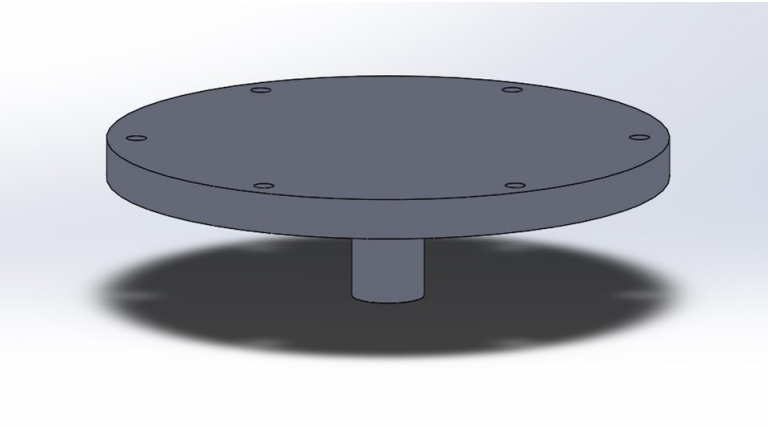
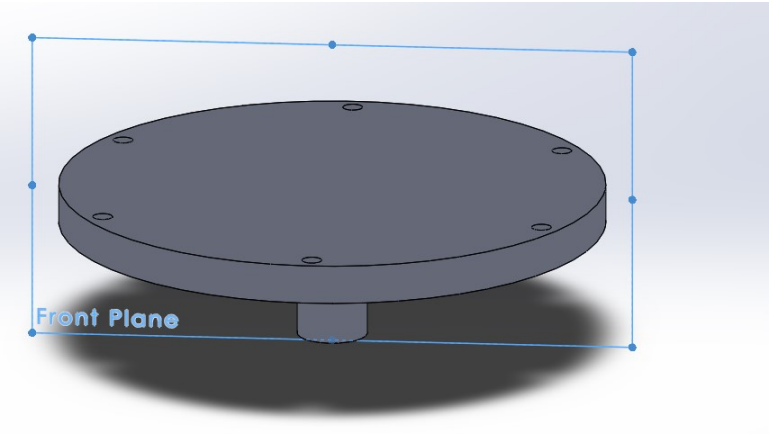
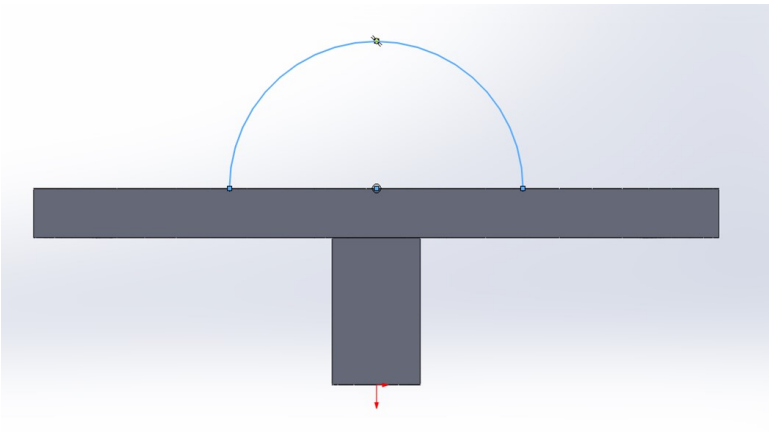


Figure 7.11: Cap for pig shield without the handle.

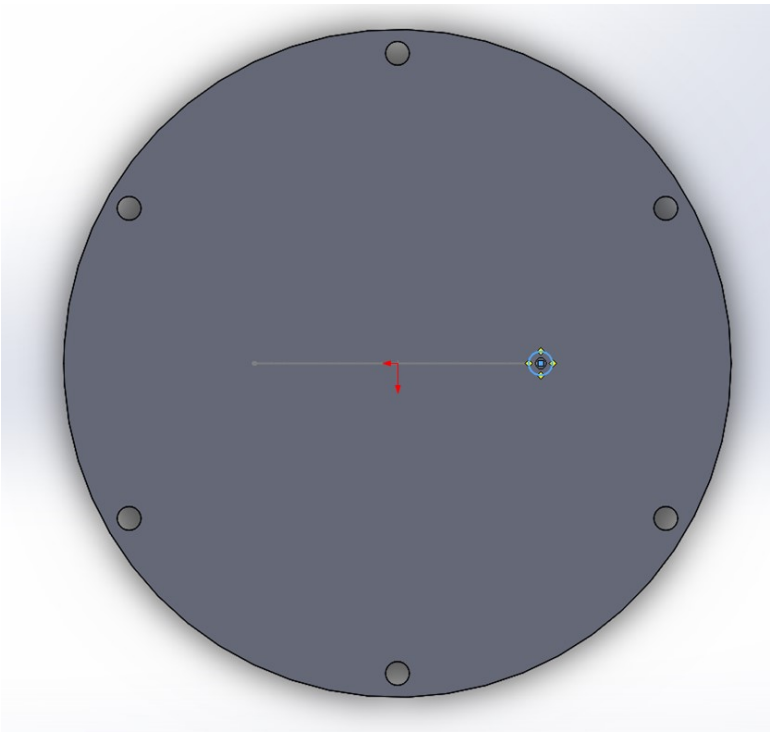
The final addition to our cap will be a handle. In order to add this shape, we will create a swept boss along a curved line. First, select the "Front Plane" in the Feature Manager Region. This will activate a drawing plane that runs vertically through the center of the cap.



Select the "Front View" and "Sketch". We can then draw the shape of our handle profile on this plane ensuring that it is coincident with the top of the cap (little yellow coincident icon appears). Use the center point arc tool to sweep out an arc centered on the top of the lid with a radius of 3 cm.

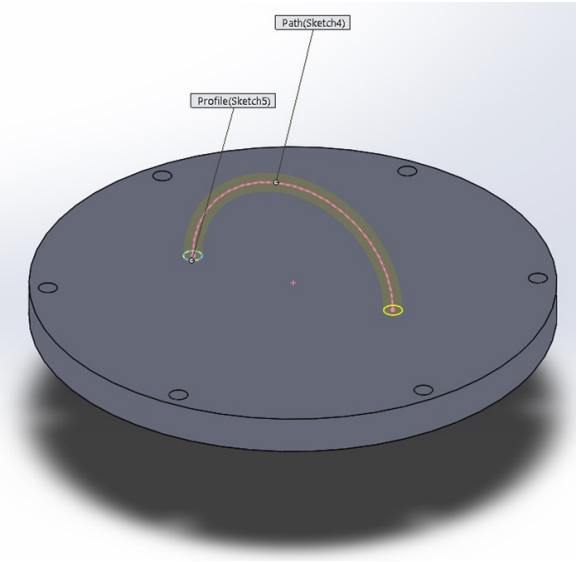


This will serve as a path for our swept boss. Click the check mark and exit the sketch. Now select the top view and click on the top of the lid to allow drawing on the surface. Select "Sketch" and draw a circle centered on one end of the swept arc we just drew with a radius of 0.25 cm. This will serve as the profile that we will sweep out along the arc.



Click the check mark and exit the sketch. Return to the trimetric view and click on the "Features" tab and the "Swept Boss/Base" button. In the Feature

Manager Region there are two areas to specify surfaces under the title "Profile and Path". Click on the blue box and then the small circle we just made which is the profile. The word Sketch5 will pop up in the blue box indicating that this Sketch will be used to follow the designated path. Now click on the pink box and then the arc and the word Sketch 4 will pop up in the box and you will get a preview of the swept boss.



Click on the green check mark and the handle will be fully defined. Save the file as "PigLid.swprt". We now have a complete design of the base and the lid to work with shown in Figure 7.12.

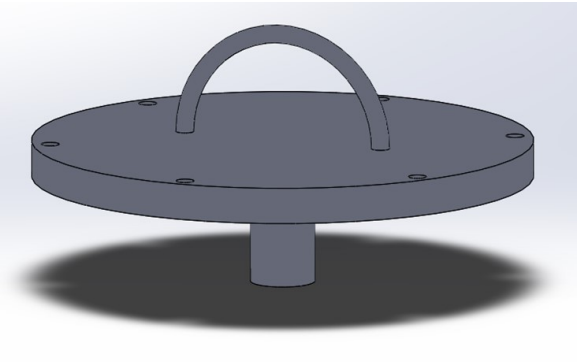
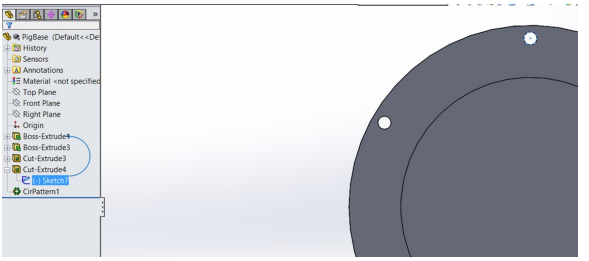


Figure 7.12: Final pig cap design.

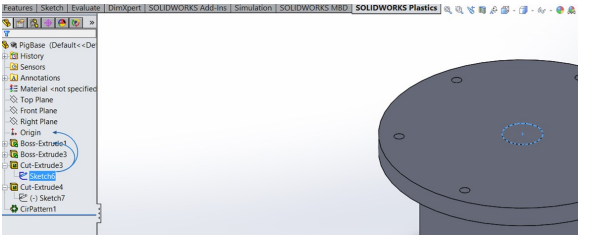
7.4 Scaling and Changing Appearances of Parts

One of the best features of solid modeling in CAD is the ability to easily change or modify the dimensions and shapes of our model. For instance, if we rotate the model of our pig base and look at it from the bottom, we will notice that the bolt holes are not centered in the area of the flange. They are too far outward. This is easily fixed. Go to the Feature Manager Region and then click on the plus sign on the Feature Tree next to Cut Extrude4. This will expand it and you will see the parent Sketch7.



Right click on the Sketch7 label in the tree and then select the "Edit Sketch" button in the pop-up menu. The top bolt hole will turn blue and the properties will show up in the Feature Manager Region. You can then change the y-coordinate from 6.5 to 6 cm and click on the green check mark to save the change. Now select "Exit Sketch" in the Command Manager region. Notice how all of the bolt hole are now centered in the flange. This is the advantage of using a repeated structure to make the pattern. You only need to change the parent feature and all the daughters will move.

We will change one other dimension on our model. The interior hole for storing radioactive material is very small and the shield is extremely thick for this size of pig. To make our container more realistic, we will change the interior hole from a 2 cm right cylinder to a 4 cm x 6 cm cylinder. For the base, this means changing the diameter of the cut from 2 to 4 cm and the depth of the cut from 5 to 7 cm. First change the view to trimetric to better see the feature. Then expand the Cut-Extrude feature to reveal the Sketch6 feature. Right click on it and change the radius from 2 to 4 cm.



Exit the sketch and right click on the Cut-Extrude feature and select the Edit Feature button. Change the depth from 5 to 7 cm and then save the changes. Now the base will have the correct dimensions.

Open the pig cap part and make the same change to the bolt hole arrangement – change the y-coordinate from 6.5 to 6 cm. Also change the dimensions on the extrusion that fits into the base to 3.8 cm in diameter and 1 cm deep.

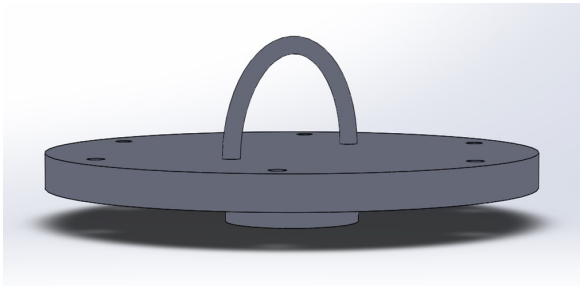
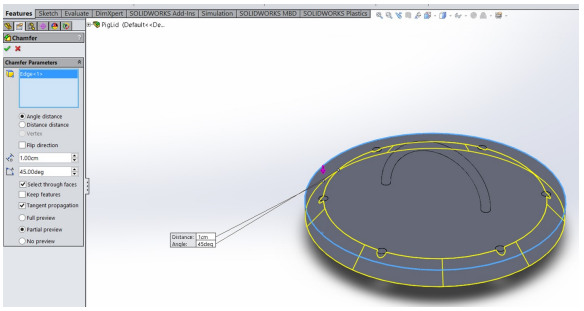
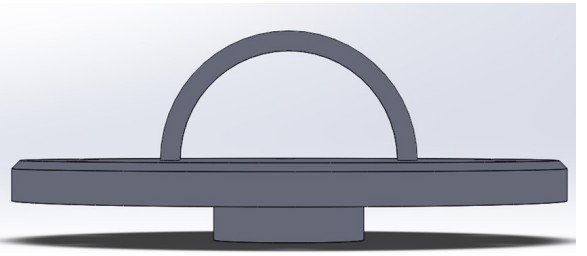


Figure 7.13: Adjusted final cap design.

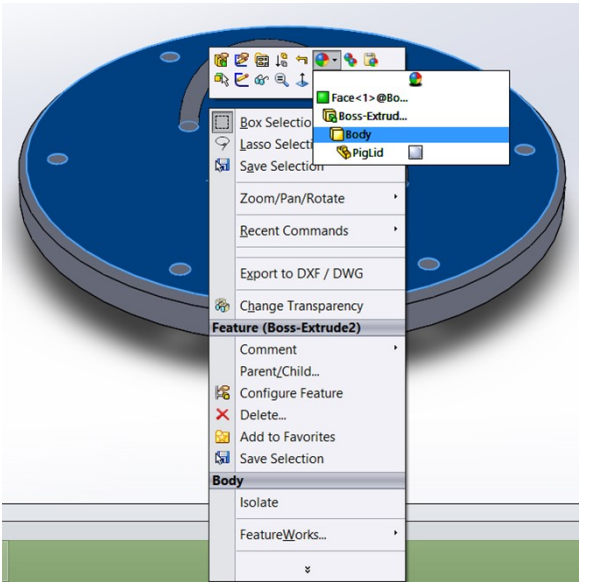
Now we can change some of the cosmetic appearances of our parts. If we look at the parts, they all have sharp 90-degree corners. This may be appropriate for the parts that mate between the lid and base but the top of the flange is probably curved or has a chamfer (slope) to it. We will apply a chamfer to the top of the flange on the lid and repeat it to the bottom of the flange on the base. To create a chamfer, click on the "Features" tab and then click on the down-arrow next to the "Fillet" button to reveal the chamfer button. Select the chamfer button and you will see the options appear in the Feature Manager Region. Click on the edge of the flange and you will get a preview of what the chamfer would look like.



Leave the angle at 45 degrees and change the length of the chamfer to 0.2 cm. Select the green check mark and the top will now have a sloped edge. Repeat the process to add a chamfer to the bottom of the flange on the base.

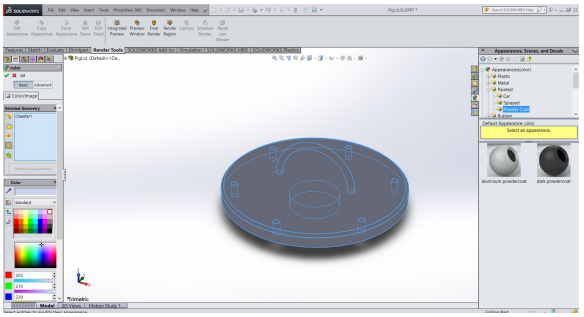


Now we can modify the models' appearance by applying color and texture to give a more realistic rendering of the final product. We can get basic colors using the basic appearance features but to get more realistic rendering we will need an Add-In. Solidworks® has a number of additional features that can be loaded to assist with applications. The Solidworks Visualize add-in is not normally loaded when you open the program. In order to load it, select the Options button in the Menu Bar and select Add-ins. Check the box next to Solidworks Visualize and the add-in will load and the menu bar at the top will now have a Solidworks Visualize menu.



To apply a texture and color to the lid, right click on the top and then select the appearance button and then select the "body" option. This will apply the texture and color to the entire part. If you wanted to color one face differently, you can apply an appearance to just that face. For simplicity, we will color the entire body. The Appearance Scenes and Decals window will open on the right side of the Graphics Area in the Task Pane. Expand the tree labeled "Appearances Color" and "Painted" and select "Powder Coat". The lower region

of the window will show two options for powder coated coloring – aluminum and dark.



We can select the Aluminum powder coat on the right and then pick a color (yellow) from the pallet on the left. The feature will change colors and then you can select the green check mark to save the appearance and exit the menu.

You may have noticed that the appearance does not look much different. In order to get the photo-realistic effect, go to the Solidworks Visualize menu at the top of the window and select Final Render. The rendering window will then pop up and it will take some time to make the image. Once it is done, select the option to save the image and it will be available for use in other applications. The final rendered images for the lid and base are presented in Figures 7.14 and Figure 7.15.

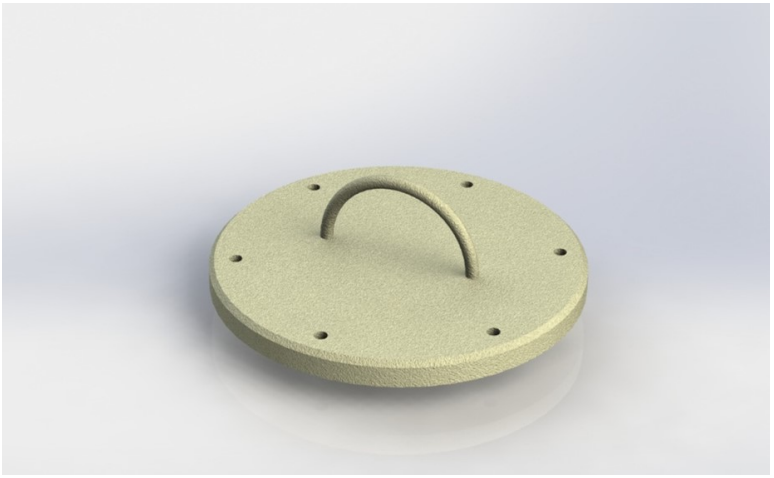


Figure 7.14: Final render of shield lid.

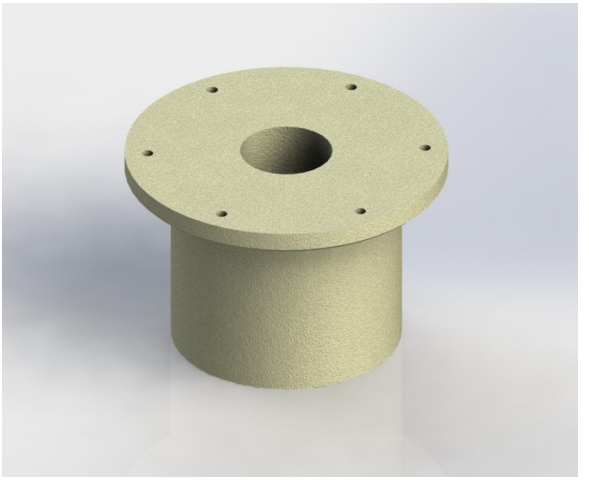
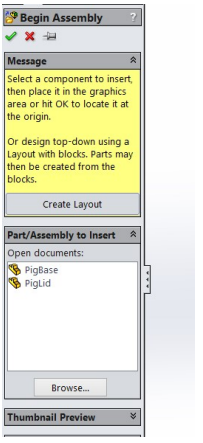


Figure 7.15: Final render of pig base.

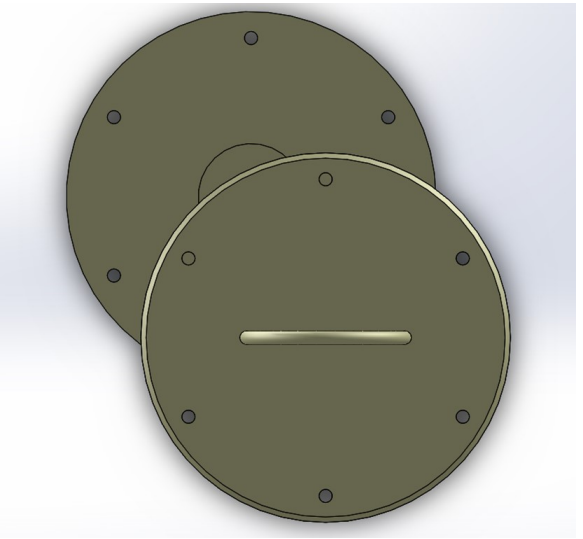
7.5 Creating an Assembly from Parts

We will now create a simple assembly by placing the lid on top of the base. Go to the menu and select "File" "New" "Assembly". If the base and lid parts are open, you should see them as possible parts to include in the assembly in the Feature Manger Region.

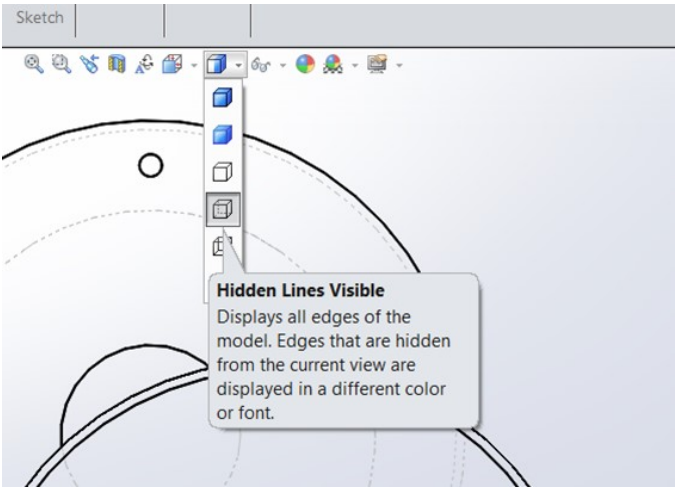


If they are not open, you can browse for them. You can then click on the PigBase and select the check mark "OK" and the base will be inserted at the origin. Once the part is inserted, go to the lower right of the window, and change the units form IPS to CGS to match the parts.

Now we can use the Insert Components button on the Assembly tab at the top and select the lid. Click on the PigLid part name and then move the mouse to the Graphics Area. The lid part will move in and follow the pointer. When you have it in position, left-click the mouse to fix it in place. It may look like it is in the right position but when you change the view to the top view you may see that it is not lined up.

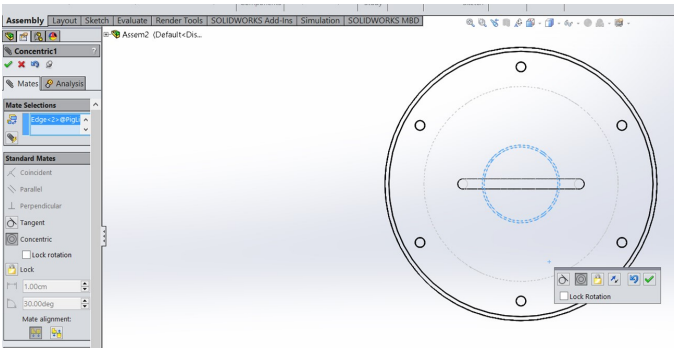


We will use the mate function to line the top up with the base. To use it, first change the view from "Shaded with Lines" to "Hidden Lines Visible"

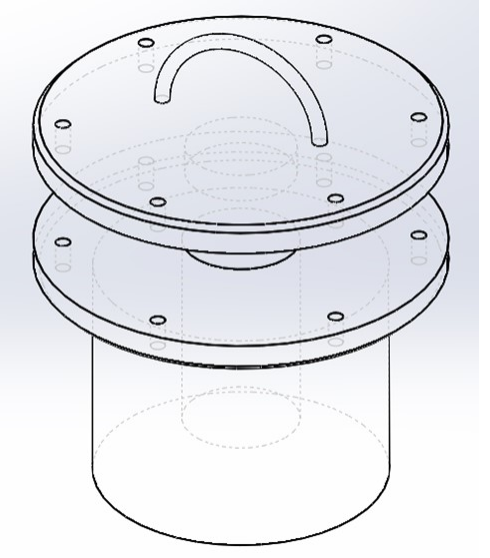


This will allow us to see the interior lines in the shape so we can use them to line up the two parts. We can select the "Mate" button on the Command

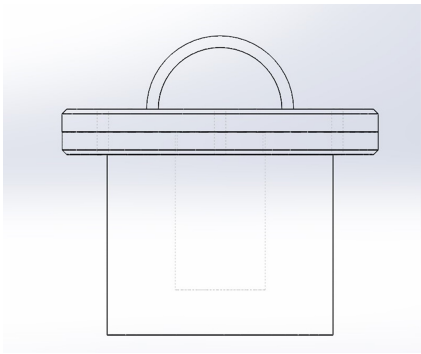
Toolbar and then click on the two circles that make up the center hole in the base and the stem for the cap. The two parts will then line up in a concentric arrangement. We can then select the checkmark on the pop-up in the graphics window followed by the checkmark in the feature window.

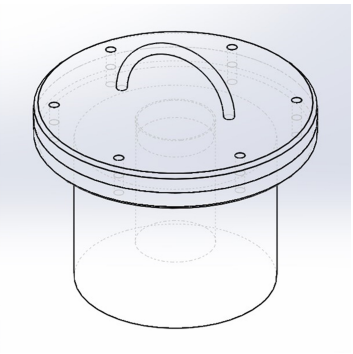


If we go back to the trimetric view, we will see the lid is lined up over the base but it may be too high.



We can now select the front view and click on the Mate button again. This time, click on the bottom edge of the lid and the top edge of the base. The lid will move down to the base. Select the green check on the pop-up and on the Feature Manager and the two parts will be lined up.





We can save the assembly as Pig.sldasm. We can also use the Solidworks Visualize to get a final rendered image of the assembly. Because the parts already have the powder coat appearance, we do not have apply it to the assembly shown in Figure 7.16.

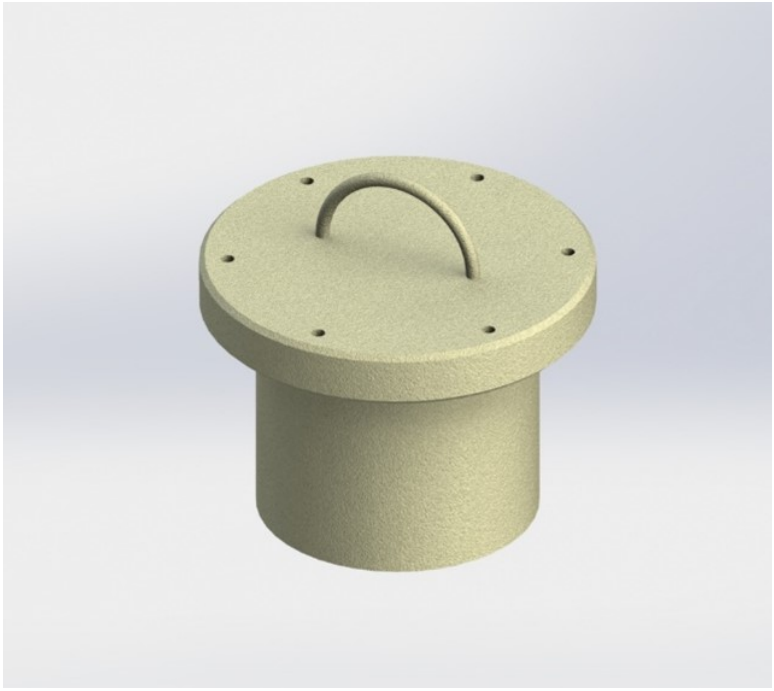


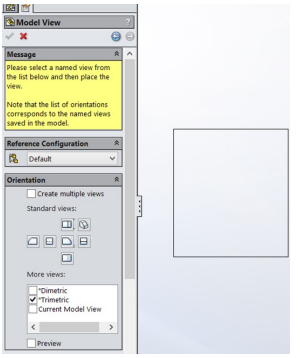
Figure 7.16: Final render of pig assembly.

7.6 Creating A Mechanical Drawing

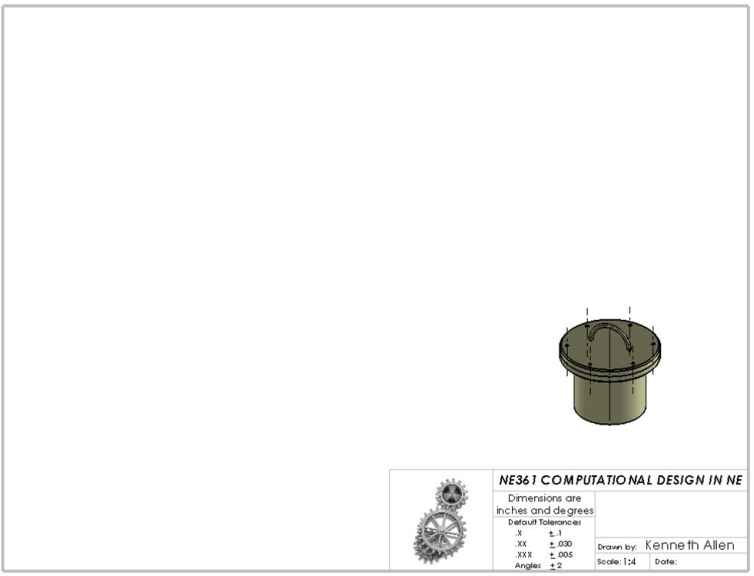
Now that we have our parts and the assembly, we can make a detailed drawing that includes all of the critical dimensions and views that would allow a manufacturer to create it to our specifications. To create a drawing, go to the menu and select "File" "New" "Drawing". You will need to select the drawing paper size and type. If you desire, you can use a drawing template that has a boarder and title block. Alternatively, you can make your own template as outlined in Howard and Musto's book in Chapter 2. (Howard and Musto; 2015)

Once the file is opened, the "Model View" option automatically appears on the left side of the Graphics Region. If the two parts and assembly files are still open, their names will appear as options in the "Part/Assembly to Insert" area. We will include two drawings for each of the parts and assembly.

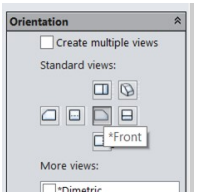
Select on the assembly Pig and then click on the "Next" arrow on the top of the window.



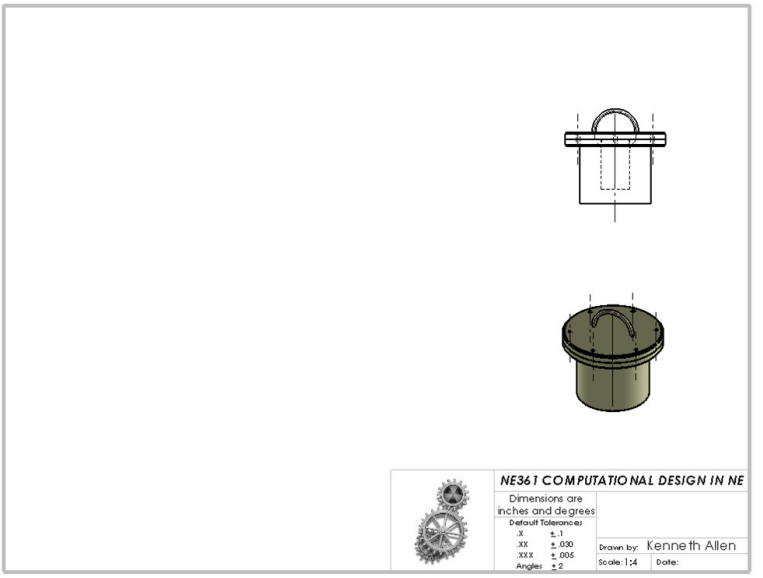
The trimetric view will be the default. We could select another orientation to include selecting multiple views. In this case, we will keep the trimetric view that is shaded with lines and place that drawing to the right side of the drawing. We could change the scale at this point if we desired.



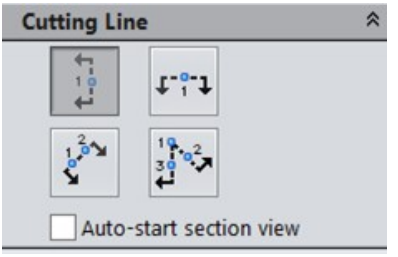
We will now add another view of the assembly. Click on the View Layout tab and the Model View button to open up the Model View window on the left side again. Once again click on the name for the Pig assembly and the next arrow. This time, select the "front" view for the Standard Views.



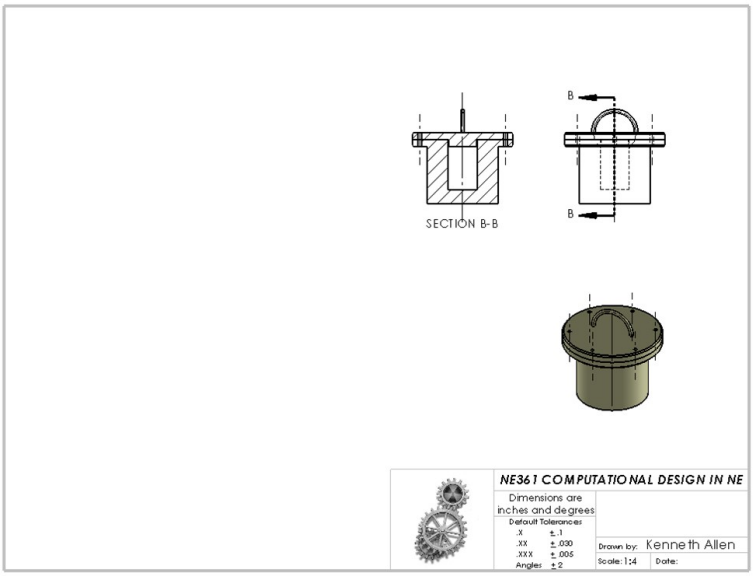
This will automatically change the image from shaded with lines to a show hidden line visible drawing. We will put this drawing above the trimetric view.



We can add a cross section cut to our drawing to show the detail better. Click on the Section View button on the Command Manager area. The window will pop up on the left of the Graphics Region that will have a variety of section options. Select the vertical section option.

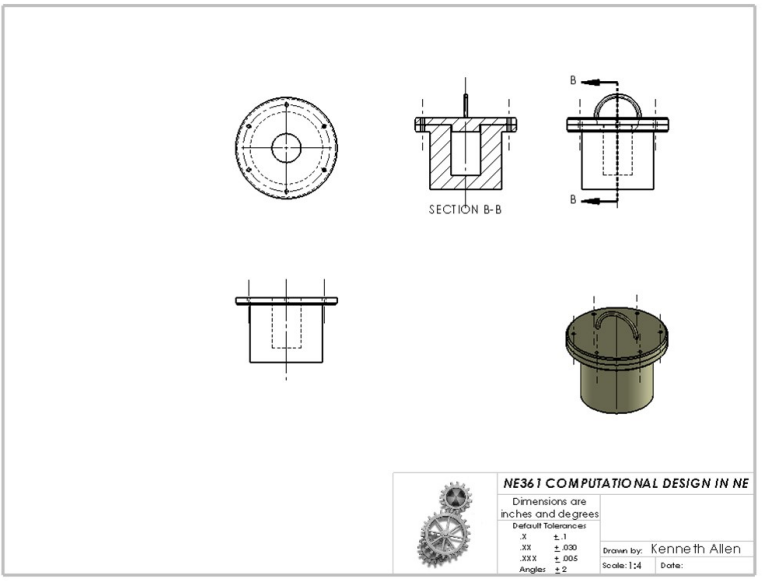


Move the section line to a coincident location in the center of the front view of the pig and left-click the mouse. Move the mouse to the left and the section drawing will appear and will move horizontally with the mouse movement. Select an appropriate distance and click again and select the green check mark.

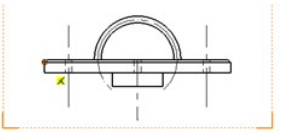


Now we have a section that shows how the lid and base mate up and the region in the center of the assembly.

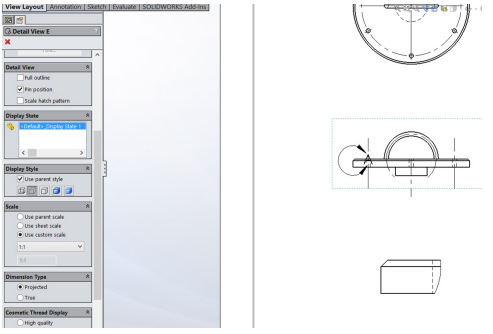
We can add the drawings for the parts. Because all sides are essentially the same, we will only present the base and the lid from two orientations – top and front. Click on the Model View button again and select the PigBase part name and the next arrow. This time we will select the "Create Multiple Views" box. We will select the Top and Front views and turn off the Trimetric view. When we select the check mark, the two views will be added to the drawing. We can use the mouse to move the base left and right moving the front view and we can move the drawings up and down using the mouse on the top view. Note how the two drawings are linked to each other and move as a unit.

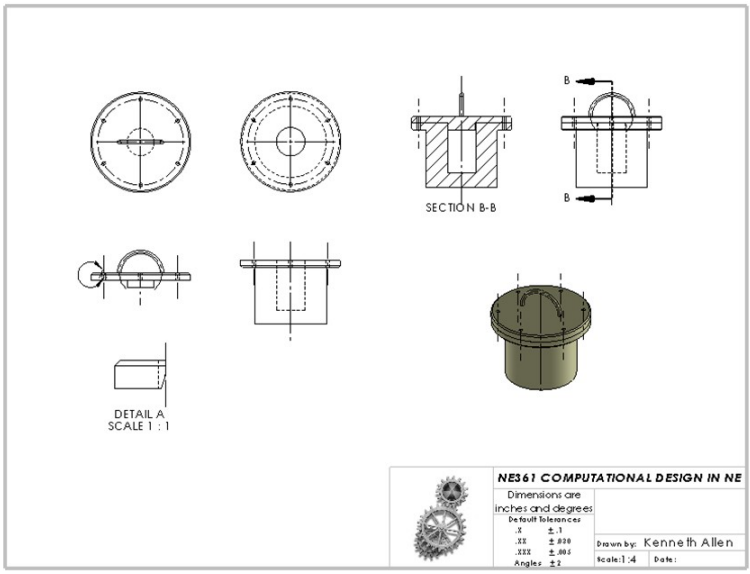


Repeat to add the lid drawing for the top and front views. We can move the drawings around to make a better arrangement. The last part of the drawing we will add will be a detail view for the chamfer on the lid. Select the Detail View button on the Command Manager area. Move the pointer to the corner of the chamfer until the coincident relation indicator pops up.

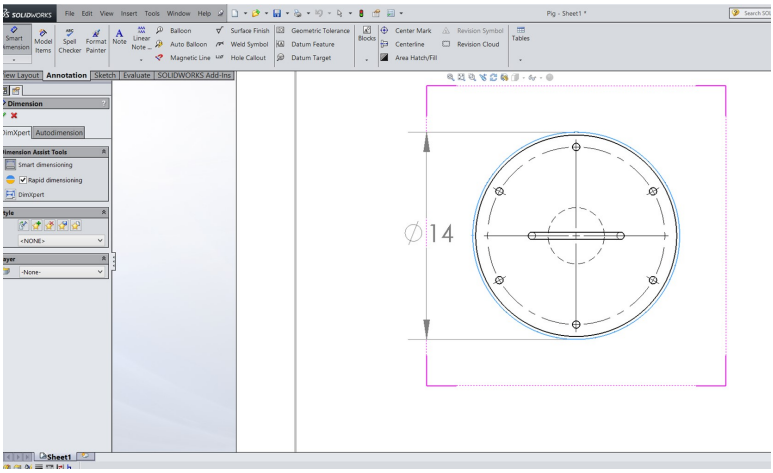


Now drag out a circle until you get the end of the flange and release the mouse. Change the scale to 1:1 and move the detail blow up to below the image for the lid. Click to place it in the drawing.

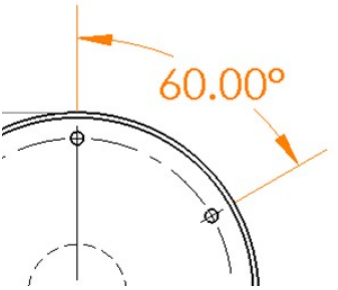




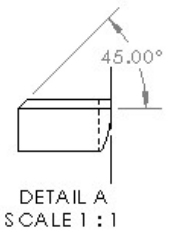
Now we have a complete drawing except for the dimensions and annotations. Adding dimensions can be accomplished by selecting the Annotation tab and then using the "Smart Dimension" button. Make sure that you are in the correct units in the Status Region (i.e., CGS) otherwise the dimensions will be off. To add the dimension, zoom in to the drawing area and click on the feature and move the mouse away from the object. As you move the mouse around you will get several automatic options for how the dimension lines will look. For example, an outside diameter may be a single arrow callout pointing to a circle at first but, as you move farther away, it may show two tangent indices with a dimension arrow between.



When adding dimensions, it is important to place them so that the drawing is still readable and ensure that everything is labeled. This includes any angular relationships such as the angle of the chamfer or the arrangement of the bolt holes. To get the angle between the bolt holes. Click the Smart Dimension Button and then click both centerline intersections for two adjacent bolt holes and move the mouse away. The angular relationship will automatically show up on the drawing indicating that they are 60 degrees apart.



Similarly, by selecting the two intersecting lines on the Detail region for the chamfer and moving away, we will add the dimension for 45 degrees.



There is no need to repeat a dimension for a part that is the same in two views. For example, the outer diameter for the lid has already been declared on the Top view drawing. Do not include it on the front view because it does not provide additional information and clutters up the drawing.

The last step for our drawing will be to include notes. Click on the Note button and click in the graphics region to include labels for the part views. Also fill out the data block with the appropriate information regarding name, date, units etc. The completed drawing should be ready for use by the fabricator and is presented in Figure 7.17.

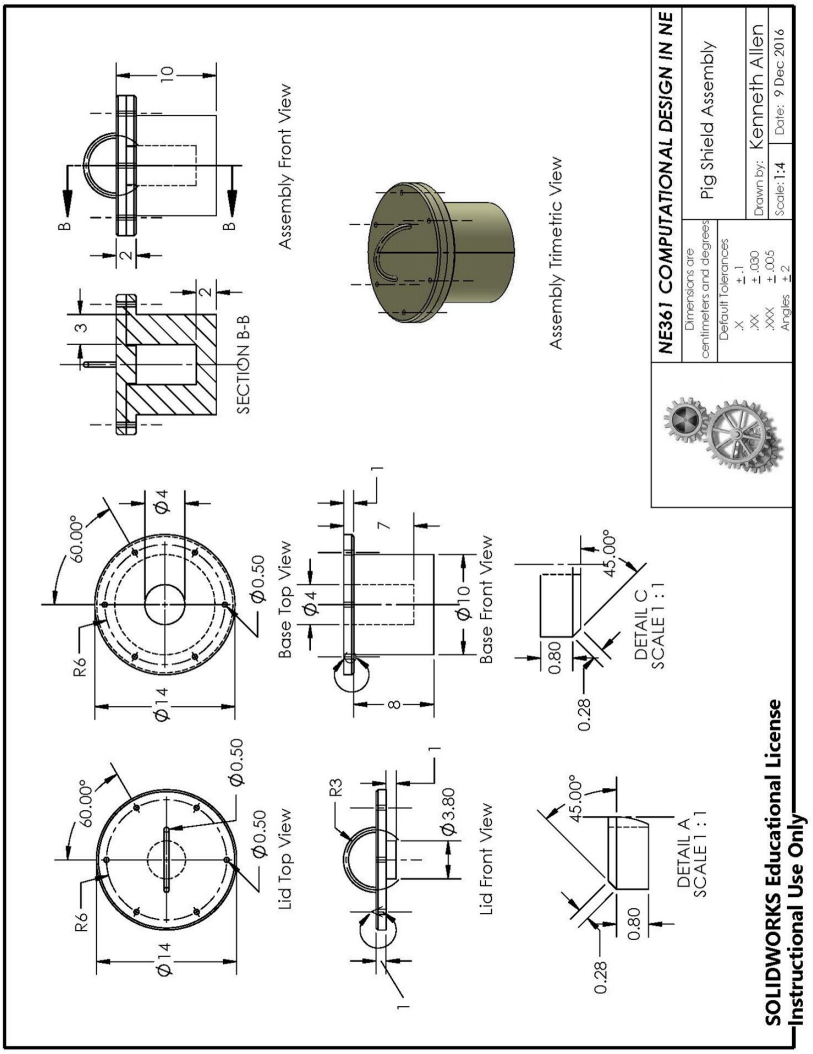


Figure 7.17: Final drawing for pig assembly.

7.7 Using Solidworks® to Create 3-D Printed Objects

The growth of 3-D printer technology has made creating prototypes and custom designs, especially out of plastic, extremely simple and effective. The Solidworks® solid model is easily converted to a file that can be read by the

3-D printer. Each printer system has its own unique software interface. For example, we will use a Catalyst® software for use on a Statsys® Dimension 1200 printer. Many 3-D printers have their own interface software or use the integrated Print 3-D program that is part of Windows®.

To print a solid body, the printer needs the model in a format it can understand. For most printers that is a stereolithographic format or a *stl file. To get a *.stl file from Solidworks® you use the "File" – "Save As" command and select the *.stl. Solidworks® will then us a function that will approximate the shape into a mesh of triangles which make up the stereolithograph. You can then check the *.stl by reading it into a program like Print 3-D which will allow you to view the model from all sides as shown in Figure 7.18.

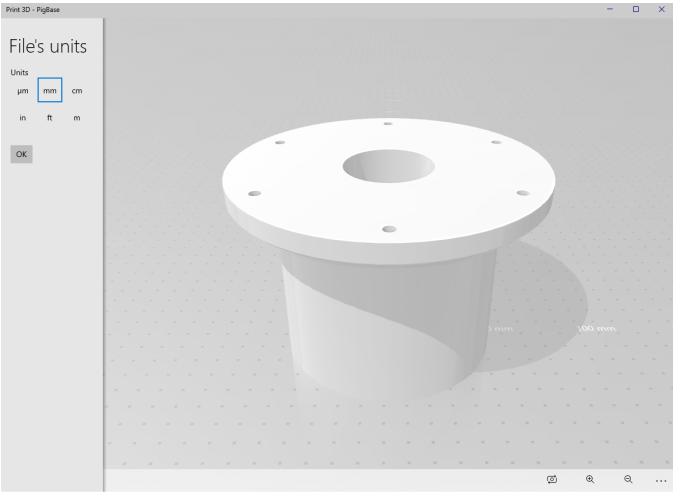


Figure 7.18: 3-D Print rendering of the pig base stereolithograph.

Some programs will also allow you to import the top in the same window and you can manipulate the two stereolithographs to verify that the lid will fit in the base. If we wanted to print the assembly as a combined unit, we would first have to save the assembly in Solidworks® as a part. Once we saved it as a part we open the part file and save it as a *.STL. If you save the assembly as a *.STL directly, it will save as mulitple *.STLs and will not combine them. The combined *STL is presented in Figure 7.19.

For printing a prototype of this model, it would make more sense to print the lid and the base separately so we can test the fit of the two. How big the prototype depends on the size of the model, the size capability of the printer, the budget for materials, time, and resolution of the printer. Because our model is small (roughly 10 cm square) we could certainly print the entire model on our Dimension 1200 printer which will hold models 10 in x 10 in x 18 in. The raw

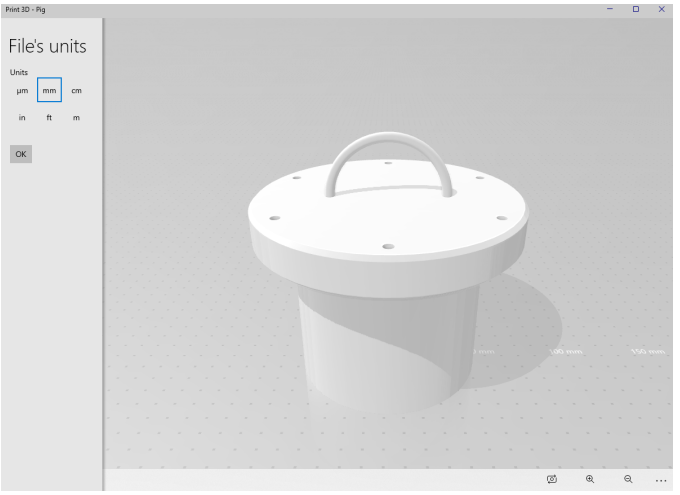


Figure 7.19: 3-D Print rendering of the combined pig with lid in single stereolithograph.

material for the ABS plastic costs on the order of $3 per cubic inch. We can select some options to reduce the amount of material used by including either a low density or high-density mesh in the inside of the feature versus a solid part. This is good practice for prototyping something that does not require structural strength. It will also speed up the print process because the printer does not have to lay down as much material. The resolution of the printer and the details in the model will ultimately decide how small you can print. Details like the holes in the lid cannot be so small that the stream of plastic from the print head cannot draw out the material. The printer resolution varies widely by brand and cost of machine.

The typical 3-D printer will have two types of materials: model and support. The model material is the structural material that makes up the solid body. The support material fills in gaps and holes as the model is being built and it is dissolved in solution after the printing process is complete. The software for the printer chops up the stereolithograph into a series of layers from the bottom up. If there are structural elements that are not supported from the bottom at each layer (such as the handle on our lid), it will fill in the area below with support material. How much support is used depends on the orientation of the model. If we place the base of our pig upright, the flange will not be supported on its own as the model is being built from the bottom up. The program would add a ring of support material around the base to the diameter of the flange so there would be a scaffold for it to build on. If we flip the base over and build it upside down, the printer will not need support material for the flange. I will, however, need to put support material in the center of the pig where there is a

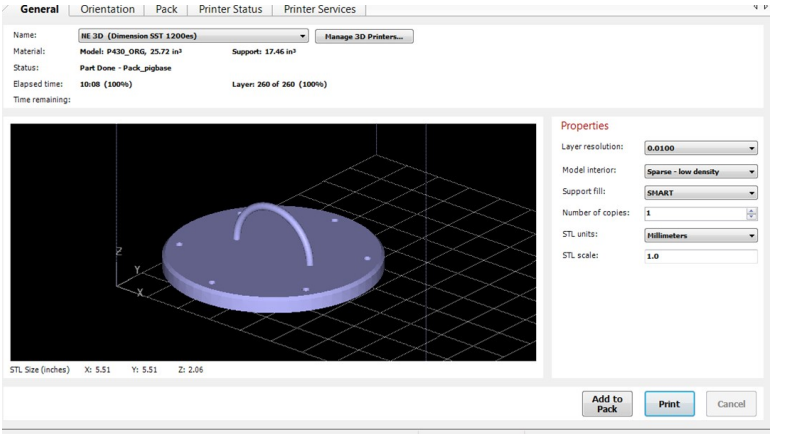


Figure 7.20: Pig lid imported into Catalyst® prior to being added to the pack.

hole so it can build the bottom of the pig.

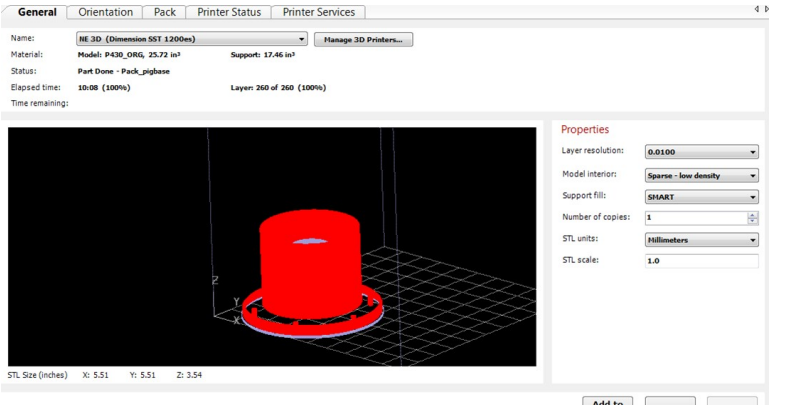


Figure 7.21: Pig base in Catalyst® after processing for the pack. Red is model material and grey indicates support.

The Catalyst® software is easy to use and allows the user to import a stereolithograph, scale its size, change the density of the interior material and the model's orientation. Once the parameters are set, the user adds the model to the print pack and the program will run and compute the layer information, how long it will take to print and how much model and support material are needed. From there, the user can duplicate the number of copies if there is room on the base and then select "Print".

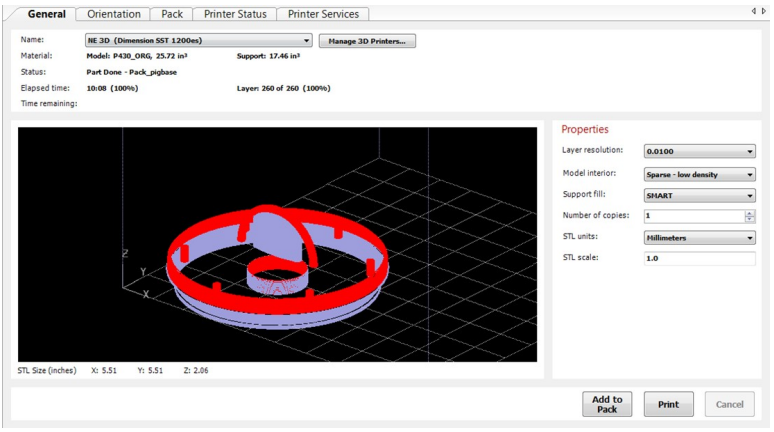


Figure 7.22: Pig lid in Catalyst® after processing for the pack. Red is model material and grey indicates support.

7.8 Introduction to Computer Aided Manufacturing

Computer Aided Manufacturing (CAM) is a similar process to 3-D printing in that it takes 3-D solid bodies and uses a computer program to operate machinery to create a final process. In CAM, the process is normally to run a CNC Mill or CNC Lathe. A mill is analogous to a 3-D printer but rather than adding material to make a structure, it subtracts material using various bits and tooling. CNC machines use programming codes to tell the gantry holding the tool to move in two or more dimensions. A common code for many machines is G-code. This coding has a series of commands, much like the cards in MCNP, followed by parameters telling the machine things like spindle speed, feed rate, x, y, z position, which tool to use etc.

Programming a machine in straight G-code is cumbersome and difficult. Fortunately, there are computer software programs that will interpret solid body models and assist the user to generate the correct G-code to import into the machine. One such package is MasterCAM® and MasterCAM® for Solidworks®. MasterCAM® for Solidworks® is an add-on that can be purchased separately from Solidworks® and installed for use in the CAD program.

The entire use of MasterCAM® is beyond the scope of this text and far more involved than 3-D printing. The program has tremendous capability and can ensure that the machining of parts are both accurate and repeatable. The repeatable nature of CAM is one of its most desirable features. While a skilled machinist could build virtually any part by hand, making several of the same parts to the exact same tolerance in a short period of time is overly labor

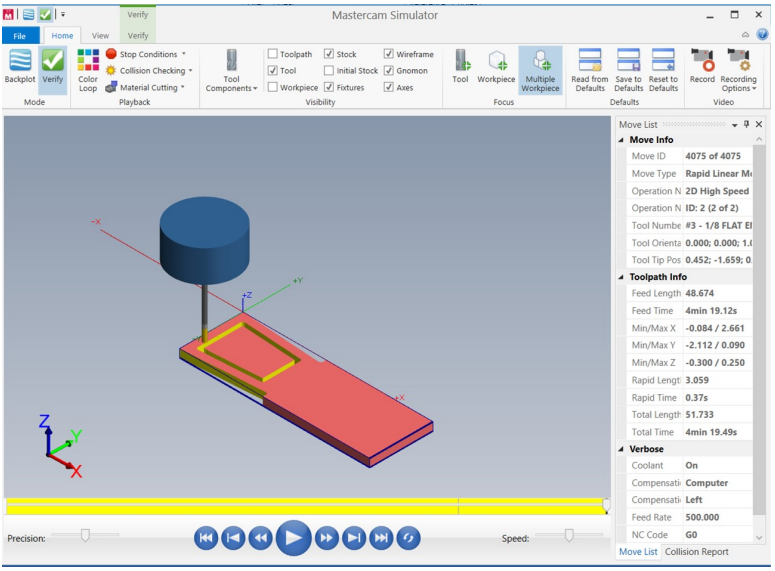


Figure 7.23: MasterCAM® simulator demonstrating the milling of a part.

intensive when the CNC machine can do the same task repeatedly without error. Also, the CAD and CAM software allow for the rapid modification of a part. For example, if we wanted a part that was 1 cm taller than our model, we could add that dimension in CAD and regenerate our code in CAM and be set with the new part. Additionally, the CAM software allows for a simulation of the machining process that can be run virtually to check tolerances, toolpaths, and the time for manufacture. This allows for the machinist to optimize the process and identify any problems to the design engineer before any material is cut.

Bibliography

Howard, W. E. and Musto, J. C. (2015). *Introduction to Solid Modeling Using SolidWorks 2014*, McGraw-Hill Education, New York, NY.

Stockbidge, W. H. (1904). Second class in drawing academy.

7.9 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- Solidworks Making First Part
- Solidworks Making Second Part
- Solidworks Making an Assembly
- Solidworks Making Third Part
- Solidworks Photo-realistic View
- Solidworks Mechanical Drawing
- Solidworks Making Assembly Exploded View
- Solidworks Create Cross Sectional Drawings

7.10 Problems

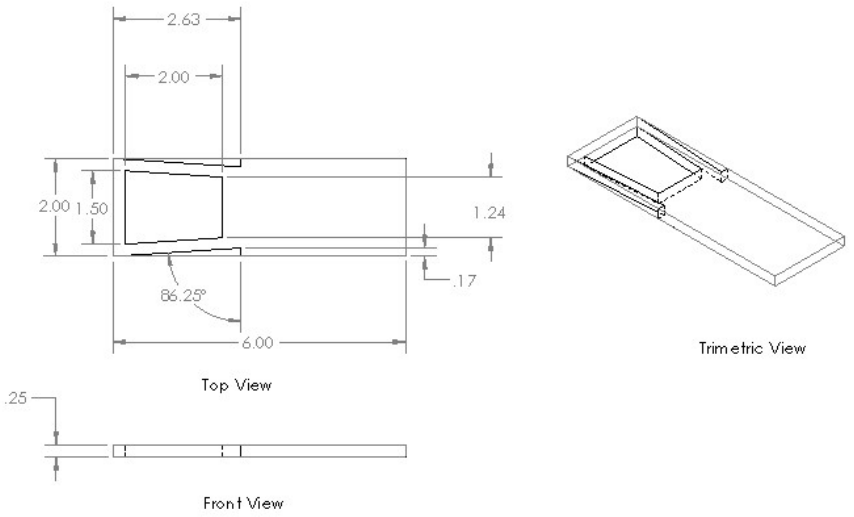
1. Redraw the radiation pig base and lid solid body parts demonstrated in this chapter using a 10 cm square box as the basic shape for the container.

2. Make a photo-realistic image for the container made in Problem 1.

3. Make an assembly from the base and lid for the container made in Problem 1.

4. Create a complete mechanical drawing for the container in Problem 1.

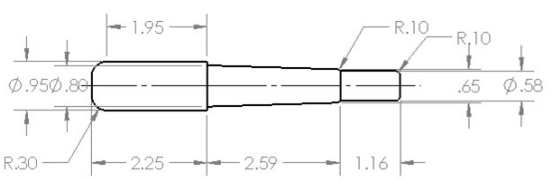
5. Make a solid body from the following drawing:

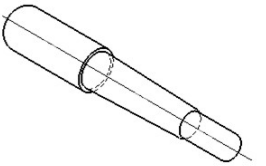


6. Create a photo-realistic model of the part in Problem 5 using a grey polished aluminum finish.

7. Create a complete mechanical drawing for the part in Problem 5.

8. Make a solid body from the following drawing:





9. Create a photo-realistic model of the part in Problem 8 using a polished copper finish.

10. Create a complete mechanical drawing for the part in Problem 8.

Chapter 8

Advanced SCALE Topics

We first explored deterministic transport models in Chapter 4. In this text, we used the t-NEWT module in SCALE to create a unit cell for a reactor. Later we modified that cell to determine characteristics like the fuel and moderator coefficients. Finally, we used the TRITON model to deplete the cell and determined with the infinite reactor would become sub-critical. Following our deterministic modeling approach, we completed the same calculations using MCNP in Chapter 7. In addition to the single unit cell, we modeled a repeated lattice in MCNP which could be expanded to a full-core design.

One advantage to using multiple high-level computer codes in design is that the results from the various techniques can be compared. This code-to-code benchmarking is not a true substitute for creating and testing in real life, but it will give some assurances that the scope of the problem is correct. This is especially true when two codes agree within a small percent error, and they use different calculation methods to achieve results.

In this chapter, we will expand on the unit cell model and develop a reactor assembly in SCALE. We can then complete burnup calculations and compare the results to calculations performed in MCNP. We will also move on to include hexagonal lattices. The hexagonal lattice, or triangular pitch, is used frequently in fast reactors, certain PWRs, and in some subcritical assemblies. In the end of the chapter, we will convert the 2-D NEWT models into 3-D KENO models within SCALE. The KENO module of SCALE is the three-dimensional Monte Carlo code developed for nuclear criticality safety analysis. The code is similar to the KCODE calculations performed in MCNP and will allow us to use another 3-D method to calculate criticality of unit cells, assemblies, and reactors within the same FULCRUM user interface used for t-NEWT.

8.1 Introduction to Advanced SCALE Topics

In our previous uses of SCALE, we used the ORIGEN module to complete radioactive decay of materials, t-NEWT for 2-D deterministic transport on a simple unit cell and TRITON for burnup and depletion on that cell. "NEWT (New ESC-based Weighting Transport code) is a multigroup discrete-ordinates radiation transport computer code with flexible meshing capabilities that allow 2-D neutron transport calculations using complex geometric models." (Lefebvre and Wieselquist; 2025) Which is to say we can expand our simple single unit cell into a repeated lattice with a variety of cells and perform a 2-D discrete ordinates calculation. The code will not account for changes in the axial dimension for the fuel explicitly. There are techniques that we will employ to apply a correction to the results based on the height of the fuel. For large assemblies, especially those with uniform fuel composition along the length, the 2-D calculation will provide excellent results and will be faster than a 3D model.

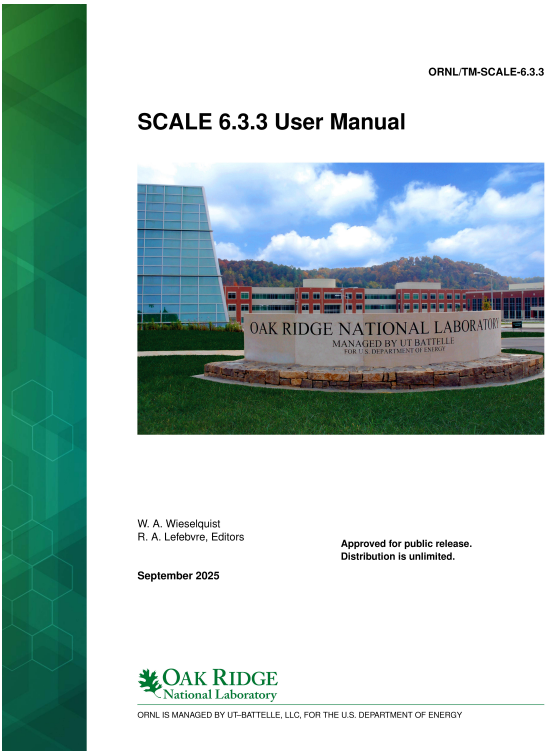


Figure 8.1: SCALE User's Manual - 2018 (Lefebvre and Wieselquist; 2025)

The entire SCALE package extends far beyond ORIGEN, NEWT, and TRITON and there are many aspects of these three packages not covered in this

text. The SCALE manual provides excellent example problems and chapters devoted to criticality safety, reactor physics, shielding, and their internal Monte Carlo solver, KENO.

8.2 Adding Geometry Units

The basic building block for complex models within NEWT is called a unit. These units can be arranged together in an array. This is analogous to the universe that we used in MCNP to create the PWR lattice in Chapter 6. The various named units will be combined in the global unit. The single unit cell NEWT model from Chapter 3 was the only object in the global unit for that model. The method for creating a PWR assembly by building on the unit cell is presented in Example 8.1.

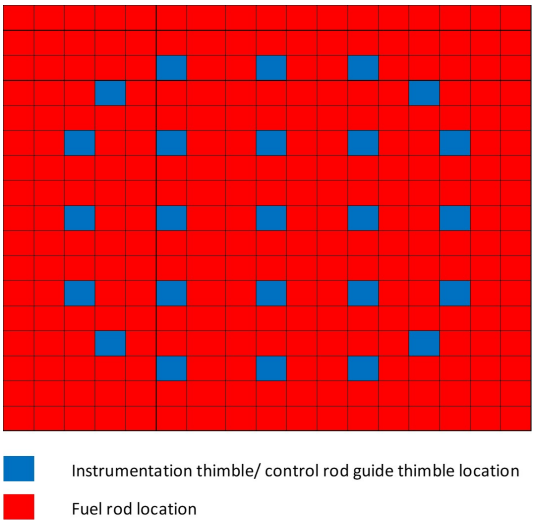


Figure 8.2: Standard rod locations for a 17 x 17 PWR assembly.

Example 8.1
Write a t-NEWT SCALE input file for the Westinghouse 17 x 17 assembly detailed in Figure 8.2 and Table 6.3. Complete the t-NEWT calculation and report the results. Compare to the MCNP solution from Example 6.5. Include a plot of the geometry from the top of assembly.

Solution: Start with the input file for the unit cell for the assembly from Example 3.1.

```
'Computational Design in NE Example 3.1
'Kenneth Allen 14 January 2023
=t-newt parm=(bonami)
NE361 Example 3.1 UO2 Unit Cell
v7-252

read comp
wtptuo2 1 10.2 2 92000 88.15 8000 11.85 1.0 1023
                92234 0.5 92235 4.5 92238 95.0 end
zirconium 2 den=6.407000 1.0 573.0 end
helium 3 1.0 573.0 end
water 4 den=0.715000 1.0 573.0 end
end comp

read celldata
latticecell squarepitch hpitch=0.6555 4 fuelr=0.385 1
                        gapr=0.390 3 cladr=0.461 2 end
end celldata

read model
UO2 Unit Cell

read parameters
sn=6
end parameters

read materials
  mix=1 pn=1 com='UO2 Fuel' end
  mix=2 pn=1 com='Zirc Clad' end
  mix=3 pn=1 com='helium gap' end
  mix=4 pn=2 com='water mod' end
end materials

read geometry
global unit 1
cylinder 1 0.385
cylinder 2 0.390
cylinder 3 0.461
cuboid 4 0.6555 -0.6555 0.6555 -0.6555
media 1 1 1
media 3 1 2 -1
media 2 1 3 -2
media 4 1 4 -3
boundary 4 10 10
end geometry

read bounds
all=refl
end bounds

end model

end
```

Note that the cross section and material composition will remain unchanged. The only modification will be to the geometry. In this example, the global unit includes the geometry for the fuel unit cell. The assembly includes the fuel unit cell and control rod guide tube unit cells which have the same geometry with the fuel and gap replaced with water. Start by renaming the global unit to just unit 2. The term global unit will be reserved for the entire model which will

include the array of fuel units and control rod guide tube units. Additionally, remove the mesh values on the boundary. The mesh division will be applied to the global unit boundary.

```
read geometry
' fuel unit cell
unit 2
cylinder 1 0.385
cylinder 2 0.390
cylinder 3 0.461
cuboid 4 0.6555 -0.6555 0.6555 -0.6555
'Define UO2 fuel
media 1 1 1
'Define gap
media 3 1 -1 2
'Define clad
media 2 1 -2 3
'define moderator
media 4 1 -3 4
boundary 4
```

Add unit 3 which will contain the geometry and materials for the control rod guide tubes.

```
' control rod guide tube unit cell
unit 3
cylinder 2 0.390
cylinder 3 0.461
cuboid 4 0.6555 -0.6555 0.6555 -0.6555
'Define water inside
media 4 1 2
'Define clad
media 2 1 -2 3
'define moderator
media 4 1 -3 4
boundary 4
```

Add a global unit 1 which is a cuboid the size of the assembly (cuboid 10) and contains the array of unit 2 and unit 3 cells. Note that the array will be defined after the geometry is ended. The array is named “ara 1” and is within cuboid 10 and the position 9, 9 in the array is located at 0.0 and 0.0 or at the origin. The media of water is also placed within cuboid 10 to fill any undefined regions and the boundary is defined as the outside of cuboid 10 with a mesh division of 40 units in the x and y dimensions.

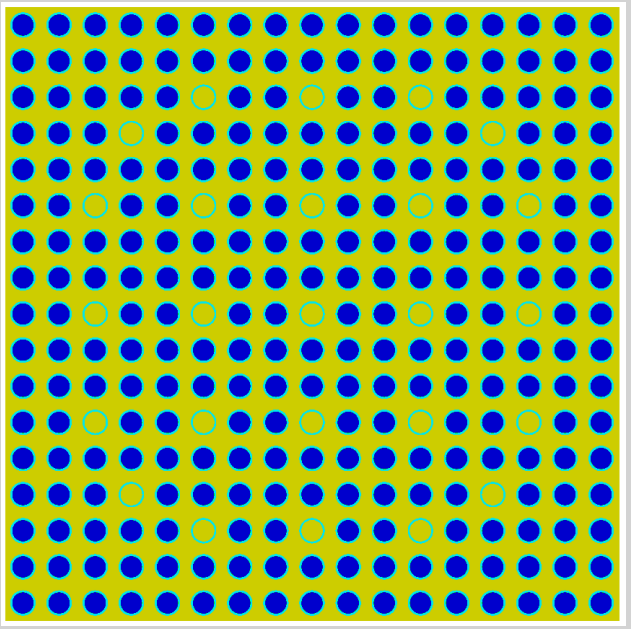
```
' global unit to contain array
global unit 100
cuboid 10 11.14 -11.14 11.14 -11.14
array 1 10 place 9 9 0.0 0.0
media 4 1 10
boundary 10 40 40

end geometry
```

After ending the geometry, create the array. It has a name of "ara 1" and consists of 17 units in the x and y dimensions. The square lattice is defined as a "cuboidal" type array in SCALE. As in the unit cell, the boundary condition will be reflected, and the model is complete.

```
read array
       ara=1 nux=17 nuy=17 typ=cuboidal
fill
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
end fill
end array

read bounds
all=refl

end bounds


end model


end
```

After the array is input, the validation message section in FULCRUM should be clear of errors indicating that the input is ready to run. Click on the "view"

tab at the top of the window to check the geometry of the model. It should look like the following:



After the model runs, the result is presented below:

```
Four-Factor Estimate of k-infinity.  Fast/Thermal boundary:   0.6250 eV
   Fiss. neutrons/thermal abs. (eta):          1.870682
   Thermal utilization (f):                    0.928866
   Resonance Escape probability (p):           0.663596
   Fast-fission factor (epsilon):              1.225289
                                              ----------
   Infinite neutron multiplication             1.412848
```

The KCODE from MCNP in Example 6.5 resulted in $k_\infty = 1.40115 \pm 0.00145$. As expected, the two models have similar values. The SCALE model is 0.8% higher. The difference between the models may be due to the SCALE model being 2D versus the 3D model within MCNP but is more likely the result of the temperature differences in the fuel cross sections.

8.3 Creating a Hexagonal Lattice

Several reactor designs use fuel assemblies where the rods are arranged on a triangular pitch within a hexagonal array. Some examples are some subcritical assemblies, sodium cooled fast reactor (SFR) assemblies, and the Russian-designed VVER pressurized water reactor assembly shown in Figure 8.3.

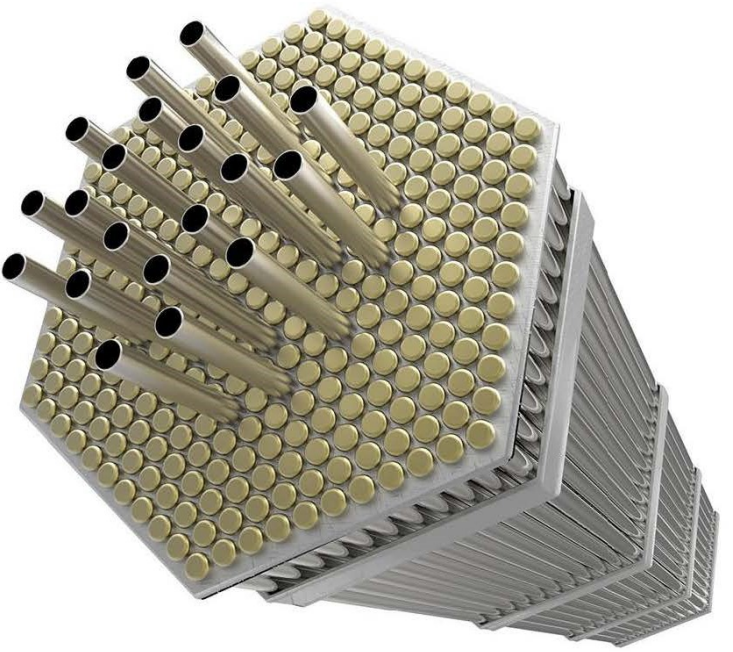


Figure 8.3: Pressurized Water Reactor Assembly for a Russian-designed VVER (Simopt; 2023)

The triangular pitch in a hexagonal assembly creates a tighter lattice within the reactor. This arrangement also allows less streaming of neutrons between fuel rods. This is especially important in fast reactor designs where neutrons will not moderate in the coolant and must be absorbed in the fuel or in depleted uranium breeding blankets. A triangular lattice will help reduce leakage in these designs and allow for higher criticality.

The creating of a triangular pitched, hexagonal assembly follows the same methods used for the square pitch. The first step is normally to create a single fuel unit cell. The primary difference in the design will be the outside geometry will be a rectangular hexagon versus a rectangular cuboid (parallelepiped in MCNP). Example 8.2 demonstrates an input file for a unit cell for a SFR in both MCNP and SCALE.

The assembly characteristics are presented in Table 8.1 with an illustration of the cross section of the assembly in Figure 8.4. The axial center of the reactor is fueled with a mixed-oxide fuel of plutonium and uranium with a total fissile

Table 8.1: Specifications for Clinch River Breeder Reactor Assembly for MCNP and SCALE Modeling

Parameter	Value
Fuel Radius (cm)	0.2921
Clad Radius (cm)	0.32
Pitch (cm)	0.84
Fuel Height (cm)	91.44
Axial Blanket Thickness (cm)	35.56
Fuel Mixture	U-Pu O_2
Isotopic comp. (w/o HM)	
^{235}U	0.14
^{238}U	69.20
^{238}Pu	0.85
^{239}Pu	16.45
^{240}Pu	7.58
^{241}Pu	3.61
^{242}Pu	2.17
^{244}Pu	9.87E-05
Fuel Density (g/cm^3)	10
Blanket Material	UO_2
Isotopic comp. (w/o HM)	
^{235}U	0.20
^{238}U	99.80
Blanket Density (g/cm^3)	10
Coolant Material	Sodium (Na)
Coolant Density (g/cm^3)	0.927
Power (MW)	6.25
Total Burnup (d)	1096
Cooling Period (d)	1825
Cladding / Can Material	Stainless Steel
Clad Density (g/cm^3)	7.8
Number Fuel Rods	217
Inside radius of Can (cm)	6.2
Outside radius of Can (cm)	6.3
Outside radius of Assembly (cm)	6.4
Fuel Temperature ({°C)	1200
Coolant Temperature ({°C)	900

weight-percentage of just under 20%. The upper and lower regions of the fuel are filled with depleted uranium-dioxide pellets which serve as axial breeder blankets. These regions shield the reactor vessel while capturing neutrons in the ^{238}U which breeds plutonium for recycling. The outside of the entire reactor core would have several rows of assemblies with full-length depleted uranium

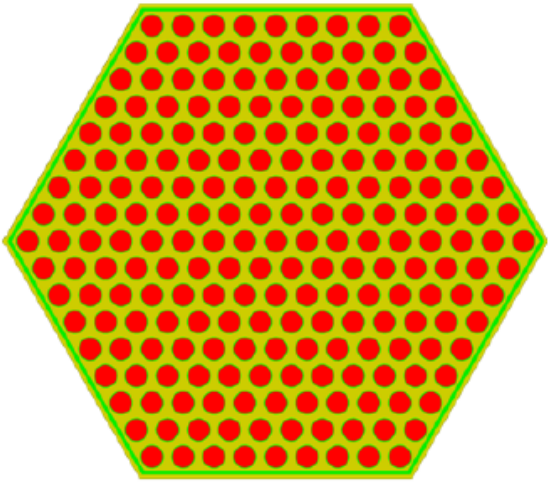


Figure 8.4: Example of Sodium-Cooled Fast Reactor assembly with no transmutation targets installed. Red represents fuel, yellow represents sodium coolant, and bright green represents the stainless-steel cladding around the fuel and the can that goes around the outside of the assembly.

rods to serve as shielding and radial breeding blankets. Note that the entire assembly consists of fuel rods and there are no empty control rod guide tubes in this example. Also, there is no gap modeled in this example. The fuel completely fills the area up to the inner surface of the cladding.

When working with hexagonal geometries, the radius of the hexagon is defined as the dimension from the center to a point normal to one side. Within MCNP, the notation for the rectangular hexagonal piped, or RHP, is like the specification of a cylinder with the X, Z, Z location for the base followed by the displacement in the X, Y, and Z direction. The major difference occurs with the final parameter for radius. While a cylinder has only one entry, the RHP will have three. The three entries indicate the radial dimension in X, Y, and Z. Figure 8.5 illustrates two RHPs of equal height and positioned centered on the origin. The difference between the two entries is the specification of the radius as either along the X or along to the Y and normal to the adjacent side. Similarly, SCALE has the option for specifying geometry in either a hexagonal prism (hexprism) or a rotated hexagonal prism (rhexprism). These geometries will become important as the arrangement of the unit cells in a lattice will make a larger hexagonal shape that is rotated when compared to the unit cell's orientation.

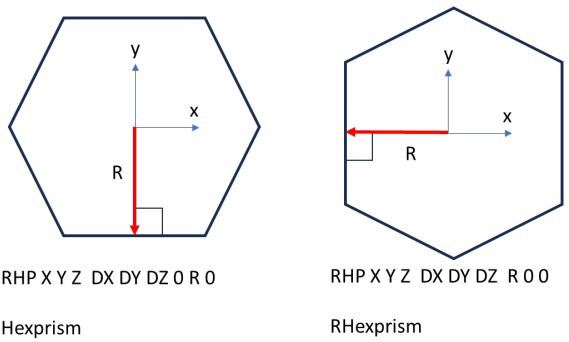


Figure 8.5: Example of hexagonal prism geometry in MCNP and SCALE. The top line represents the input format for MCNP and the bottom is for SCALE. Note SCALE is 2D and only requires the annotation of the radius.

Example 8.2
Write a SCALE and MCNP input for a unit cell for the Clinch River Breeder reactor assembly presented in Figure 8.4 and Table 8.1. Complete the t-NEWT and KCODE calculation and report and compare the results. Include a plot of the geometry from the midplane of the unit cell for each code.

Solution: The SCALE input file follows the same steps outlined for the PWR input file in Example 3.1. The MCNP input file follow the same steps outline for the PWR input file in Example 6.3. The notable differences are the material composition of the fuel, the use of sodium instead of water for coolant, and the geometry of the boundary is a hexprism instead of a cuboid. The only other significant difference is the use of a white boundary condition versus a reflected boundary condition in SCALE. The reflected boundary condition is not supported in a hexagonal geometry.

Note that the MCNP input file is three-dimensional and, as such, it defines the regions for the depleted uranium breeder blankets above and below the center region where the driver fuel is located. As SCALE is two-dimensional, the only fuel defined is the driver fuel and the geometry represents the center cross-section of the unit cell. The height parameter is included in the SCALE input to help adjust the criticality to reflect the actual height of the fuel but the only fuel composition in SCALE will be the driver. To get an good comparison of the criticality results, we have substituted the driver fuel in the DU regions in the MCNP model. In the KENO examples at the end of this chapter, the DU blanket is modeled in both the SCALE and MCNP models as both are 3D.

The following is the SCALE input file:

```
'Computational Design in Nuclear Engineering Example 8.2
'| SCALE NEWT for SCFR Unit Cell
=t-newt parm=(bonami)
hex lattice scfr
v7-252
read composition
 wtptfuel        1  10  9
                                8016 11.85185
                                92235 0.123
                                92238 61
                                94238 0.75
                                94239 14.5
                                94240 6.68
                                94241 3.18
                                94242 1.915063
                                94244 8.7e-05
                      1 1200    end
 wtptss          2  7.8  5
                                25055 2
                                14000 1
                                24000 19
                                28000 10
                                26000 68
                      1 1200    end
 na-23           3 den=0.927 1 735    end

end composition
read celldata
  latticecell triangpitch fuelr=0.2921 1 cladr=0.32 2
  hpitch=0.4032 3 end
end celldata

read model
read parameters
sn=6
kguess=1.75
dz=91.44
drawit=yes
height=91.44
end parameters

read materials

  mix=1 pn=1 com='Fuel MOX' end
  mix=2 pn=1 com='SS' end
  mix=3 pn=1 com='Na' end
```

```
' TODO: define materials

end materials


read geometry
global unit 1
com='fuel unit cell'
 cylinder 10   0.2921
 cylinder 20   0.32
 hexprism 30     0.42
 media 1 1 10
 media 2 1 -10 20
 media 3 1 -20 30
 boundary 30 2 2

end geometry

read bounds
     all=white
  end bounds
end data
end model
end
```

When run to completion the results are:

```
Four-Factor Estimate of k-infinity.  Fast/Thermal boundary:   0.6250 eV
   Fiss. neutrons/thermal abs. (eta):          1.286910
   Thermal utilization (f):                    0.989161
   Resonance Escape probability (p):           0.000000
   Fast-fission factor (epsilon):         **************
                                          --------------
   Infinite neutron multiplication             1.758026
```

The following is the MCNP input file:

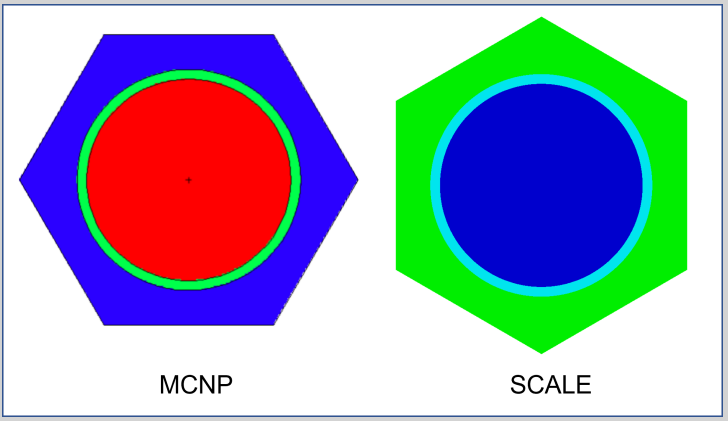
```
Clinch River Breeder Reactor Project Unit Cell
C Computational Design in Nuclear Engineering Example 8.2
C ----------------------------CELL CARD BLOCK----------------------------------
C DU BLANKET ROD on TOP
4   4  -10    -1  10          $DU BLANKET FUEL on TOP
5   2  -7.8    1 -2 10         $SS CLADDING
C FUEL ROD
13  4  -10    -1 9 -10        $PuO2 + UO2
14  2  -7.8    1 -2 9 -10      $SS CLADDING
C DU BLANKET ROD on BOTTOM
16  4  -10    -1  -9          $DU BLANKET FUEL on BOTTOM
17  2  -7.8    1 -2 -9         $SS CLADDING
18  1  -0.927  -41  #4 #5 #13 #14 #16 #17    $Na COOLANT
99  0     41 -999              $VOID OUTSIDE OF UNIT CELLS
999 0 999                      $GRAVEYARD

C ---------------------------SUFRACE CARD BLOCK--------------------------------
1 RCC 0 0 -81.2 0 0 162.4 0.2921 $INNER RADIUS OF CLAD OUTSIDE R OF FUEL
2 RCC 0 0 -81.25 0 0 162.5 0.32   $OUTTER RADIUS OF CLAD
C  OUTSIDE OF UNIT CELL
*41 RHP 0 0 -81.28 0 0 162.56 0 0.42 0
C
9 pz -45.72
10 pz 45.72
999 so 200  $sphere for graveyard

C ------------------------------DATA CARD BLOCK---------------------------------
MODE N
IMP:N  1 7R 0
KCODE   2000    1.3     5       55      500
KSRC    0.0     0.0     0.0
M1 11023.72c 1.0 $Na COOLANT
C Stainless Steel, 202
M2   6000.72c  -0.000750   $Carbon, weight fraction
      7014.72c  -0.001250   $Nitrogen
      14028.72c -0.004611   $Silicon
      14029.72c -0.000234
      14030.72c -0.000155
      15031.72c -0.000300   $Phosphorus
      16032.72c -0.000150   $Sulfur
      24050.72c -0.007740   $Chromium
      24052.72c -0.150660
      24053.72c -0.017100
      24054.72c -0.004140
      25055.72c -0.087500   $Manganese
      26054.72c -0.039153   $Iron
      26056.72c -0.619021
      26057.72c -0.014171
      26058.72c -0.001350
      28058.72c -0.034000  $Nickel
      28060.72c -0.013100
      28061.72c -0.000550
      28062.72c -0.001800
      28064.72c -0.000450
C DU BLANKET
M3  8016.73c  2.0  92235.73c  0.002  92238.73c 0.998 $ DU
C
C FUEL
M4     8016.73c  -11.85185
      92235.73c  -0.123
      92238.73c  -61
      94238.73c  -0.75
      94239.73c  -14.5
      94240.73c  -6.68
      94241.73c  -3.18
      94242.73c  -1.915063
      94244.73c  -8.7e-05
PRINT
```

When run to completion the results are:

```
| the final estimated combined collision/absorption/track-length keff = 1.74485 with an estimated standard deviation of 0.00154 |
| the estimated 68, 95, & 99 percent keff confidence intervals are 1.74330 to 1.74640, 1.74174 to 1.74795, and 1.74070 to 1.74899 |
| the final combined (col/abs/tl) prompt removal lifetime = 3.9048E-07 seconds with an estimated standard deviation of 9.3396E-10 |
| the average neutron energy causing fission = 7.8526E-01 mev
| the energy corresponding to the average neutron lethargy causing fission = 1.6134E-01 mev
| the percentages of fissions caused by neutrons in the thermal, intermediate, and fast neutron ranges are:
|         (<0.625 ev):   0.00%         (0.625 ev - 100 kev):  38.70%          (>100 kev):  61.30%
| the average fission neutrons produced per neutron absorbed (capture + fission) in all cells with fission = 1.7676E+00
| the average fission neutrons produced per neutron absorbed (capture + fission) in all the geometry cells = 1.7415E+00
| the average number of neutrons produced per fission = 2.954
```

The results are as expected and follow the pattern established with the PWR unit cell. Namely there are small differences in the k-effective values of less than 1%. These differences can be attributed to different temperature cross sections used and the fact that the SCALE model is two-dimensional and does not include the breeder blanket.

The plot from the VISED model from MCNP and the SCALE model are shown below side by side. They, of course, have the same geomtry with different associated colors for the materials. The user could specify the colors to match. Additionally, the hexagonal prisms are rotated between each model. The user could choose to change the cord for the RHP in MCNP or change to a rhexprisism in SCALE to match the orientations. For the unit cell, it will not affect the final results. For Example 8.3, it will be important to position the unit cell within the larger hexagonal prism correctly.

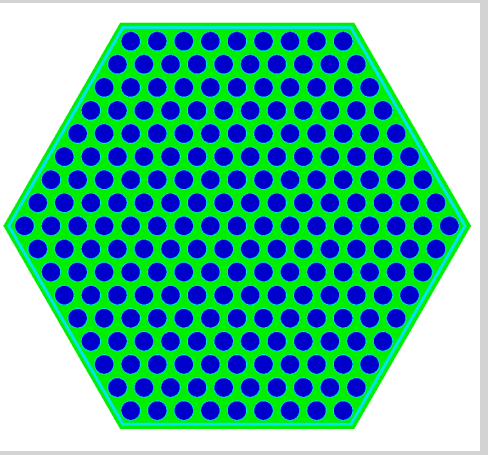


Expanding the unit cell to a full lattice in both MCNP and SCALE is presented in Example 8.3. This expansion is done following the same methods used in Example 8.1 for SCALE and Example 6.5 in MCNP. The notable addition in

the MCNP example is the use of three lattice elements in the axial dimension. This geometry allows for the depleted uranium blanket rods to be separated from the driver fuel rather than having one universe including the three fuel types. This is a realistic example if the driver fuel had a different geometry or was going to be removed at a different time during burnup. The lattice uses a fill in the blanket that simply repeats the blanket fuel cell everywhere while the driver region uses a lattice that explicitly places the fuel elements in each location. Because the assembly does not contain multiple fuel elements or control rod guides, the same lattice geometry used for the blanket could be used for the driver. However, the specification of the explicit universes in the driver region helps the student understand the nomenclature for a hexagonal lattice in MCNP.

Example 8.3
Write a t-NEWT SCALE and MCNP input for an assembly for the Clinch River Breeder reactor assembly presented in Figure 8.4 and Table 8.1. Complete the t-NEWT and KCODE calculations and report and compare the results. Include a plot of the geometry from the midplane of the unit cell for each code.

Solution: The SCALE input file follows the same steps outlined for the PWR in Example 8.1. Note that the latticecell is a triangpitch and the array type is a shexagonal. The shexagonal is a hexagonal array where the elements are shifted to account for the triangular pitch. The presentation of the unit numbers in the array helps to visualize how the 19 elements in the x and y stack up within the array. Not that any elements that are outside the largest boundary hexagon will be cut off from the final geometry. The following is the plot of the SCALE input followed by the input file:



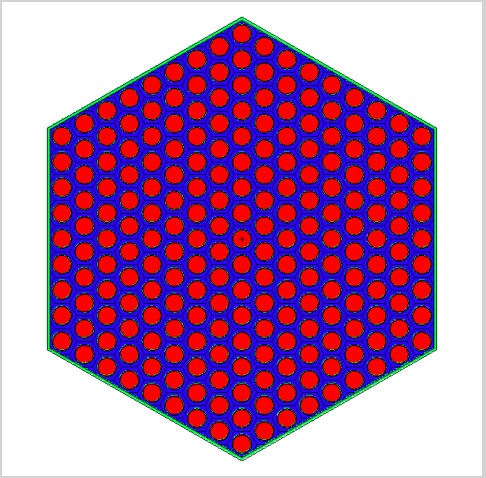
```
'Computational Design in Nuclear Engineering Example 8.3
' SCALE NEWT for SCFR Assembly
=t-newt parm=(bonami)
hex lattice scfr
v7-252
read composition
 wtptfuel        1  10  9
                                  8016 11.85185
                                  92235 0.123
                                  92238 61
                                  94238 0.75
                                  94239 14.5
                                  94240 6.68
                                  94241 3.18
                                  94242 1.915063
                                  94244 8.7e-05
                        1 1200   end
 wtptss          2  7.8  5
                                  25055 2
                                  14000 1
                                  24000 19
                                  28000 10
                                  26000 68
                        1 1200   end
 na-23           3 den=0.927 1 735   end

end composition
read celldata
  latticecell triangpitch fuelr=0.2921 1 cladr=0.32 2
  hpitch=0.4032 3 end
end celldata

read model
read parameters
sn=2
kguess=1.75
dz=91.44
drawit=yes
height=91.44

end parameters

read materials

  mix=1 pn=1 com='Fuel UO2' end
  mix=2 pn=1 com='SS' end
  mix=3 pn=1 com='Na' end


end materials


read geometry
unit 1
com='fuel unit cell'
 cylinder 10   0.2921
 cylinder 20   0.32
 hexprism 30     0.42
 media 1 1 10
 media 2 1 -10 20
 media 3 1 -20 30
 boundary 30
```

```
global unit 2
com='outside of assembly'
 rhexprism 110      6.2
 rhexprism 120      6.3
 rhexprism 130      6.4
 array 1 110  place 10 10 0 0
 media 2 1 -110 120
 media 3 1 -120 130
 boundary 130 25 25
end geometry
read array
ara=1 nux=19 nuy=19  typ=shexagonal
 com=''
 fill
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 end fill
end array

read bounds
    all=white
  end bounds
end data
end model
end
```

The MCNP input file follows the PWR in Example 6.5. Note that the specification of lattice type 2 indicates it is a triangular pitch. The array elements in MCNP do not have a shifted option so they are read in the same way the square lattice elements are read. To obtain a visual that reflects a hexagonal assembly, the lattice element universes are often input in a pattern that looks like a parallelogram. For this example, one additional step is taken where blank commented lines and additional horizontal spaces are included in the lattice specification to clearly delineate where the fuel elements occupy the hexagonal interior of the can and where they will be cut off.

As mentioned earlier, the interior RHP for the unit cells must be oriented 90 degrees off from the external larger RHPs that define the exterior of the assembly and the two hexagonal surfaces that make up the assembly can.

The following is the plot of the MCNP input followed by the input file:



```
Clinch River Breeder Reactor Project Single Fuel Assembly
C Computational Design in Nuclear Engineering Example 8.3
C ----------------------------CELL CARD BLOCK----------------------------------
1   2  -7.8 -21 31 $STAINLESS STEEL CAN OUTSIDE OF ASSEMBLY
C Na COOLANT FILLED UNIT CELL
2   1 -0.927 -41 u=1
3   1 -0.927 #2  u=1
C DU BLANKET ROD
4   4  -10    -1     u=3  $DU BLANKET FUEL
5   2  -7.8   +1 -2  u=3  $SS CLADDING
6   1  -0.927 #4 #5 u=3  $Na COOLANT
C LATTICE TO FILL BLANKET RODS ON TOP OF ASSEMBLY
7   0  -31 10 -11 FILL=11
8   1  -0.927  -41  LAT=2 U=11 FILL=3
C LATTICE TO FILL BLANKET RODS ON BOTTOM OF ASSEMBLY
9   0  -31 8 -9 FILL=21
10   1  -0.927  -41 LAT=2 U=21 FILL=3
C LATTICE TO FILL FUEL RODS IN MIDDLE OF ASSEMBLY
11   0  -31 9 -10 FILL=10
12  1  -0.927  -41 LAT=2  U=10 FILL=-9:9 -9:9 0:0
      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
c
       1 1 1 1 1 1 1 1 1 1   4 4 4 4 4 4 4 4 4 4   1
        1 1 1 1 1 1 1 1 1   4 4 4 4 4 4 4 4 4 4   1
         1 1 1 1 1 1 1 1   4 4 4 4 4 4 4 4 4 4 4   1
          1 1 1 1 1 1 1   4 4 4 4 4 4 4 4 4 4 4 4   1
           1 1 1 1 1 1   4 4 4 4 4 4 4 4 4 4 4 4 4   1
            1 1 1 1 1   4 4 4 4 4 4 4 4 4 4 4 4 4 4   1
             1 1 1 1   4 4 4 4 4 4 4 4 4 4 4 4 4 4 4   1
              1 1 1   4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4   1
               1 1   4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4   1
                1   4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4   1
                 1   4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4   1 1
                  1   4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4   1 1 1
                   1   4 4 4 4 4 4 4 4 4 4 4 4 4 4 4   1 1 1 1
                    1   4 4 4 4 4 4 4 4 4 4 4 4 4 4   1 1 1 1 1
                     1   4 4 4 4 4 4 4 4 4 4 4 4 4   1 1 1 1 1 1
                      1   4 4 4 4 4 4 4 4 4 4 4 4   1 1 1 1 1 1 1
                       1   4 4 4 4 4 4 4 4 4 4 4   1 1 1 1 1 1 1 1
                        1   4 4 4 4 4 4 4 4 4 4   1 1 1 1 1 1 1 1 1
c
       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
C FUEL ROD
13  4  -10    -1       u=4   $PuO2 + UO2
14  2  -7.8   +1 -2   u=4   $SS CLADDING
15  1  -0.927 #13 #14 u=4   $Na COOLANT
99  0     +21 -999   $VOID OUTSIDE OF ASSEMBLY
999 0 999
```

```
C --------------------------SUFRACE CARD BLOCK-------------------------------
1 cz 0.2921 $INNER RADIUS OF CLAD OUTSIDE R OF FUEL
2 cz 0.32   $OUTTER RADIUS OF CLAD
C   OUTSIDE OF ASSEMBLY
*21 RHP 0 0 -81.29 0 0 162.58 6.3 0 0
C  SS CAN
31 RHP 0 0 -81.28 0 0 162.56 6.2 0 0
C
C  OUTSIDE OF UNIT CELL
41 RHP 0 0 -81.28 0 0 162.56 0 0.42 0
C
8 pz -81.28
9 pz -45.72
10 pz 45.72
11 pz 81.28
999 so 200  $sphere for plotting

C ------------------------------DATA CARD BLOCK--------------------------------
IMP:N  1 15R 0
vol      7.03781E+02 1.44129E+02 0.00000E+00 4.14132E+03
         8.29217E+02 4.50764E+03 4.70622E+03 4.70622E+03
         4.77196E+03 4.77196E+03 1.21238E+04 1.21238E+04
         5.29312E+03 1.06005E+03 5.62648E+03 0.00000E+00
         0.00000E+00
MODE    N
KCODE   2000    1.3     5       55      500
KSRC    1.6     0.0     0.0
M1 11023.72c 1.0 $Na COOLANT
C Stainless Steel, 202
M2   6000.72c  -0.000750   $Carbon, weight fraction
      7014.72c  -0.001250   $Nitrogen
      14028.72c -0.004611   $Silicon
      14029.72c -0.000234
      14030.72c -0.000155
      15031.72c -0.000300   $Phosphorus
      16032.72c -0.000150   $Sulfur
      24050.72c -0.007740   $Chromium
      24052.72c -0.150660
      24053.72c -0.017100
      24054.72c -0.004140
      25055.72c -0.087500   $Manganese
      26054.72c -0.039153   $Iron
      26056.72c -0.619021
      26057.72c -0.014171
      26058.72c -0.001350
      28058.72c -0.034000  $Nickel
      28060.72c -0.013100
      28061.72c -0.000550
      28062.72c -0.001800
      28064.72c -0.000450
C DU BLANKET
M3  8016.73c  2.0  92235.73c  0.002  92238.73c 0.998 $ DU
C
C FUEL
M4     8016.73c  -11.85185
      92235.73c  -0.123
      92238.73c  -61
      94238.73c  -0.75
      94239.73c  -14.5
      94240.73c  -6.68
      94241.73c  -3.18
      94242.73c  -1.915063
      94244.73c  -8.7e-05
PRINT
```

When run to completion the results are presented below from SCALE and MCNP:

```
Four-Factor Estimate of k-infinity.  Fast/Thermal boundary:   0.6250 eV
   Fiss. neutrons/thermal abs. (eta):          1.287560
   Thermal utilization (f):                    0.988201
   Resonance Escape probability (p):           0.000000
   Fast-fission factor (epsilon):            ***************
                                             --------------
   Infinite neutron multiplication             1.749659
```

```
the final estimated combined collision/absorption/track-length keff = 1.72343 with an estimated standard deviation of 0.00161

the estimated 68, 95, & 99 percent keff confidence intervals are 1.72182 to 1.72505, 1.72020 to 1.72666, and 1.71912 to 1.72774

the final combined (col/abs/tl) prompt removal lifetime = 4.0646E-07 seconds with an estimated standard deviation of 8.5978E-10

the average neutron energy causing fission = 7.6746E-01 mev
the energy corresponding to the average neutron lethargy causing fission = 1.5261E-01 mev

the percentages of fissions caused by neutrons in the thermal, intermediate, and fast neutron ranges are:
        (<0.625 ev):   0.00%         (0.625 ev - 100 kev):  39.38%          (>100 kev):  60.62%

the average fission neutrons produced per neutron absorbed (capture + fission) in all cells with fission = 1.7512E+00
the average fission neutrons produced per neutron absorbed (capture + fission) in all the geometry cells = 1.7173E+00

the average number of neutrons produced per fission = 2.948
```

It is notable that the multiplication factor is further differentiated in the full lattice models with the MCNP model value around 1.5% lower.

8.4 Working With Lattices

The same basic parametric analysis models demonstrated in chapters 3 and 5 performed on a single unit cell can be completed on a lattice. The user can modify geometries, isotopes, or materials, and assess changes in reactivity and other characteristics of the assembly. After completing scoping calculations on a unit cell, the next level of analysis should include a lattice, especially if there are non-fuel elements within the assembly or if the assembly has non-uniform fuels such as integrated fuel burnable absorbers (IFBA) rods mixed within. If the fuel or assembly is non-uniform along the axial length, it may require a three-dimensional code to adequately model it.

As with parametric changes to geometry and materials, the fuel temperature and coefficient calculations performed earlier on the unit cell are easily repeated in the lattice. As with the other parameters, the actual values for these values are dependent on the model's accuracy. The fuel temperature coefficient is dependent on the composition of the fuel which changes over time due to burnup and the transmutation of uranium into plutonium isotopes and actinides.

Completing a burnup calculation within an assembly in either MCNP or SCALE is the same procedure used for the unit cell. Care must be taken to

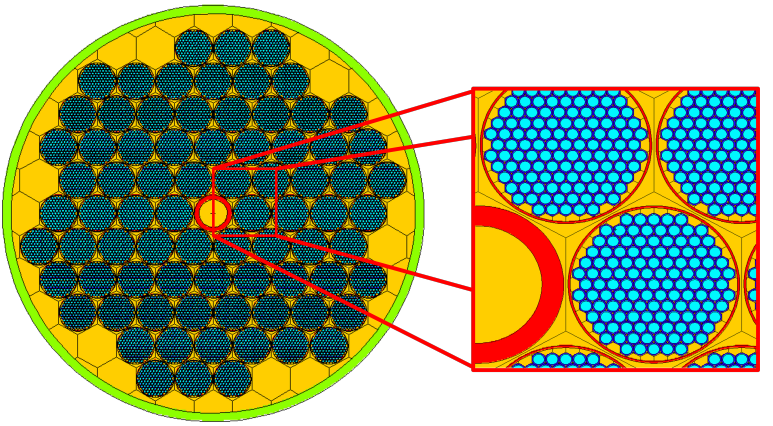


Figure 8.6: Example reactor core modeled in MCNP. The core consists of uniform hexagonal assemblies in a larger hexagonal array surrounding a center control region in a cylindrical core.

ensure that the power used in either the TRITON SCALE model or the MCNP BURN model is set correctly. Recall that the SCALE model uses average specific power given in megawatts per metric tonne (1000 kg) of initial heavy metal (MW/MTHM) material. MCNP, however, uses power in megawatts for the model. The specific power for a fuel rod may be the same as an assembly, if the assembly consists entirely of that fuel rod. The total power, will depend on the number of rods in the assembly.

Just as the more explicit geometry improves the accuracy of the model from the unit cell to the assembly, a full reactor model is more accurate than the assembly model. A reactor core can be modeled in either SCALE or MCNP by nesting units or universes within larger lattices. Figure 8.6 is an example of a reactor model that has fuel assemblies in hexagonal arrays modeled within a larger hexagonal array for the core. The overall geometry for this reactor is a cylinder. Modeling a reactor core allows for leakage and the multiplication factor given by the code finally represents k-effective vs. k-infinity. When modeling a reactor, the user sometimes can make use of symmetry to reduce the overall size of the array. For example, if the reactor core is symmetric in one-eighth or one-quarter geometry, reflective planes can be used to cut a wedge out of the core. This technique may greatly reduce the computational time for the code.

8.5 Power and Flux Across a Lattice

In Section 6.5, we covered how to apply a tally to a square lattice in MCNP to obtain flux or power in a specific lattice element. This method can be used in a hexagonal lattice and be expanded within a reactor core. The notation used for tally included the cell name where the F4 volume tally was to be performed separated by the higher cell name with a < character followed by the lattice number location inside square brackets (`[]`). This method can be expanded to include higher-level lattices within a core model by including multiple cell names such as `F4:N (5 < 4 < 1 [65])` which would be the volume tally on the 65th lattice element in cell 5 which is in cell 4 and in cell 1.

The fission power and flux are provided automatically within the SCALE NEWT output for each lattice element. Figure 8.7 is a partial table from a SCALE lattice assembly output. The Zone Data table presents flux and fission power for each mixture in each lattice location. This table can be copied and sorted in a spreadsheet to determine the maximum values. However, it is difficult to determine where the zone locations are within the assembly. The geometry of each zone is specified earlier in the SCALE output file in a table labeled Body Characteristics. The first 10 zones are presented in Figure 8.8.

Zone data

Zone No.	Mix No.	Relative Flux	Relative Fission Power	Zone Volume
1	3	1.22694E-03	0.00000E+00	4.74678E-01
2	2	1.22691E-03	0.00000E+00	8.57221E-01
3	3	1.22704E-03	0.00000E+00	6.34205E-02
4	1	1.22730E-03	3.69062E-03	1.96817E-02
5	2	1.22708E-03	0.00000E+00	9.48098E-03
6	3	1.22702E-03	0.00000E+00	1.11649E-01
7	1	1.22721E-03	3.69025E-03	5.18546E-02
8	2	1.22700E-03	0.00000E+00	1.84228E-02
9	3	1.22692E-03	0.00000E+00	1.11649E-01
10	1	1.22712E-03	3.68996E-03	5.18546E-02

Figure 8.7: First ten rows of the Zone Data table from a SCALE NEWT output on an assembly.

Close analysis of the two tables demonstrates that the Body Characteristics table and Zone Data table do not have the same number of rows. In this model, there are 1086 zones in the Body Characteristics table while there are only 815 zones in the Zone Data table with fission power and flux. The zone information cannot be matched to get the coordinate locations for the fuel elements within the assembly. However, closer inspection of the Body Characteristics includes the unit numbers and the geometry associated with each element. In Figure 8.8, the unit 2 objects represent the outside elements defining the stainless steel can and the sodium border on the outside of the SCFR assembly. The unit 1 elements represent each fuel pellet, clad, and coolant. The power and flux

values in the Zone Data table. The geometry for the X and Y locations can be distilled from the Body Characteristics by eliminating the unit 2 bodies and removing the duplicate X and Y locations provided by the concentric bodies of fuel, cladding, and coolant for each rod. These X and Y locations would then match the 815 zones in the Zone Data table and could be used to plot the results. To accurately plot in a three-dimensional space, a hexagonal assembly would require additional dummy elements for the X and Y locations that are outside the hexagonal shape with a flux and power of zero. This is required because the plotting space for most three-dimensional surface plots is performed on a square Cartesian space. This process can be performed using a spreadsheet and sorting function. However, it is time consuming and prone to errors. The following section provides a method for writing a Julia program to parse the SCALE output from either a square or hexagonal lattice and automatically plot either the flux or fission power across the assembly.

Body Characteristics. Rotation angle in degrees counterclockwise.
Size parameters. cyl: A=radius, B=no. of sides. cub: A=width, B=height. hex:

Body Type	Input Unit	Zone Number	Subdiv nx	Subdiv ny	--Center-- x	--Center-- y	Size Parameters A	Size Parameters B	Rotation Angle
hex	2	1	25	25	0.000	0.000	0.000	6.400	30.000
hex	2	2	0	0	0.000	0.000	0.000	6.200	30.000
hex	2	3	0	0	0.000	0.000	0.000	6.300	30.000
hex	1	4	0	0	-7.980	-6.547	0.000	0.420	0.000
cyl	1	5	0	0	-7.980	-6.547	0.292	12.000	0.000
cyl	1	6	0	0	-7.980	-6.547	0.320	12.000	0.000
hex	1	7	0	0	-7.140	-6.547	0.000	0.420	0.000
cyl	1	8	0	0	-7.140	-6.547	0.292	12.000	0.000
cyl	1	9	0	0	-7.140	-6.547	0.320	12.000	0.000
hex	1	10	0	0	-6.300	-6.547	0.000	0.420	0.000

Figure 8.8: First ten rows of the Body Characteristics table from a SCALE NEWT output on an assembly.

8.6 Developing Post-Processing Tools in Julia

Writing codes that can read data, perform calculations on that data, and write or plot output is an essential skill for engineering. It is common for the output from nuclear engineering codes to contain large amounts of data in a variety of tables in a plain text format. As we have seen, this is the case for SCALE and MCNP. If a user only needs to find and record one or two data points from a single output file, it makes sense to use a text editor, like Textpad, to open the output file and search for the required information. This is the procedure we have used in the examples in this text to this point. As described in the previous section, the process for collecting and sorting the data from a lattice from a SCALE NEWT output file is challenging and prone to error. Other situations occur in codes, like MCNP, where the data is spread over many tables or pages, and it is not easily extracted manually. For example, the production of actinides and fission products throughout the burnup of a fuel rod or assembly requires the user to search through pages of data for the desired output at each timestep in the process.

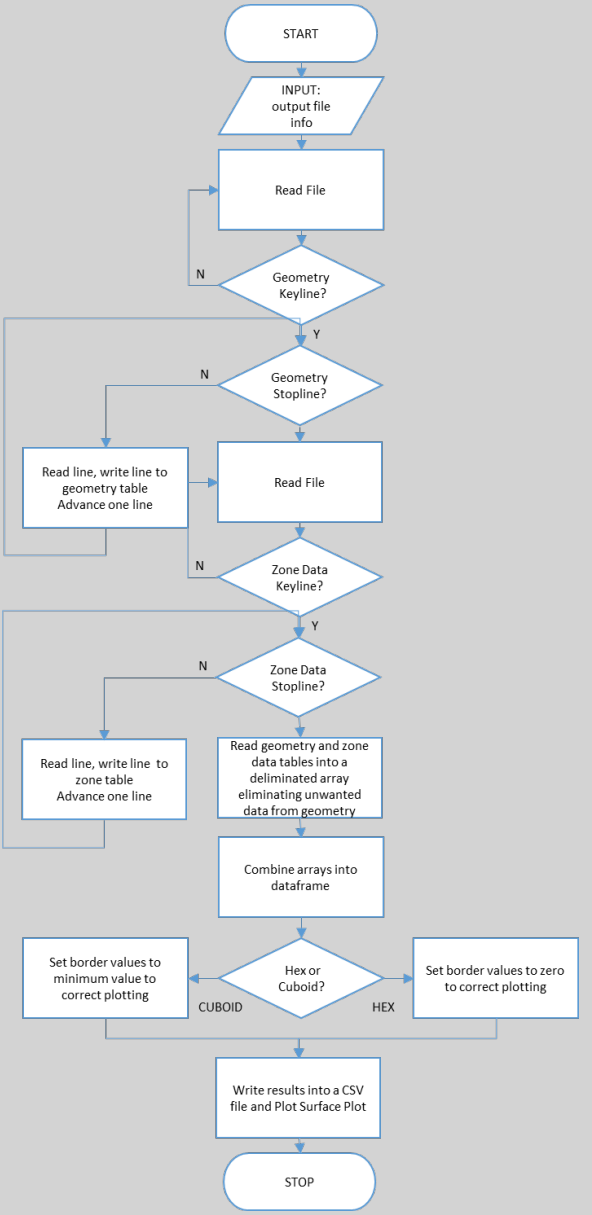
When the process of extracting and recording the data from an output file becomes difficult, tedious, or repetitive, the user should look to write a parser program. The parser program reads the output file, extracts the data, arranges, or sorts the data into a format the user can use, and, if desired, plots the results. A parser can be written in most any computer language. For this book, we will continue to use Julia as our language. The steps can be followed and adjusted in another interpreted code, like Python, or in a compiled code, like C++. The parser example in Example 8.4 is versatile as it will plot the flux or fission power for any rectangular or hexagonal assembly from SCALE NEWT. It also writes a comma separated variable (CSV) file in the process. This file could be saved, and the user could manipulate or plot using a spreadsheet program. The parser program can be improved or modified by including other calculations. For example, the program could calculate the peak-to-average power for each fuel element and plot that quantity. Typically, the peak-to-average power must remain below an upper limit within a reactor design to ensure there are not hot spots where local fuel element failure can occur.

Example 8.4
Write a parser program in Julia that reads in a SCALE NEWT output file for either a rectangular or hexagonal fuel assembly and generates a surface plot of either the flux or fission power.

Solution: To begin the process of writing a parser, the user needs to identify lines or characters within the output file that serve as landmarks within the file to find the desired data. For our parser, we will refer to the line that starts the data table as the "keyline" and the line that terminates the data table as the "stopline". Different codes will read in text files in different ways and the process of extracting data may be different. For example, C++ will read in a line of text and numbers will be read as numbers. Also, the user can specify the code to read to a particular column within C++ and then read a number into a array. This process is especially effective to retrieve information and requires little sorting or manipulation. Unfortunately, Python will read in an entire line of text and will, by default, read the information as a string. The code will require a second step that divides the string into individual words and then the words must be reassigned to an appropriate number format, such as an integer or float. Julia also reads in a line of code and can be written, like Python, to parse the line into an array. However, Julia has a more powerful method to get the results we want. In Julia, we can read the table data we want, write that table to a text file, and then catenate, or combine, the table with another table in a dataframe. The dataframe converts the data to numbers and allows for sorting and query functions like a spreadsheet. The dataframe can also be easily plotted.

This program follows the same method used for programming explained in Section 2.5.

1. *Write out the algorithm using a logical flowchart.*



2. *Write a meaningful header with appropriate comments.* Open Visual Studio Code and create a new file named "Parser1.jl" and start with comments and include the packages necessary for program.

```
#= ************************************************************************
Julia parser program
Purpose to develop program to read plain text input file and retrieve values
Written 21 Feb 2023 Dr. Kenneth Allen
Program reads SCALE ouput files for assembly models from either triangular
pitch or square pitch assemblies.  It searches for the geometry specifications
for X and Y locations of fuel and the assoicated flux or fission power and
plots the results. ALT+J+O include("Parser1.jl")
************************************************************************=#
#Load packages needed
using DataFrames
using DelimitedFiles
using Query
using Plots; gr()
using CSV
```

3. *Declare constants and variables.* The program will have two keylines representing the unique lines in the output file to mark the start of the geometry and zone tables. These lines are text and must be exactly as they are presented in the output file. This means the line cannot have more or less spaces between the words etc. A technique is to cut and paste directly from the output file to the VSC editor and surround by quotes. While the keyline is unique to the output file, the stopline does not have to be. The stopline is a line that follows the keyline that, when read, terminates the loop. The stopline text can occur in other places within the file and it will not trigger action unless the keyline has been read and the loop is initiated reading in data. The other items in this block are counting variables initialized to zero.

```
# Set constants used in the function
#-------------------------------------------------------------------------------|
keyline = " Type  Unit   Number  nx  ny      x        y        A        B      Angle      Information"
stopline = "1         ***************************"
keyline2 = "No.   No.      Flux     Fission Power     Volume"
stopline2 = "module xsproc is finished. completion code 0."
line_count = 0
instance = 0
instance2 = 0
countblock = 0
countblock2 = 0
```

4. *Write a routine to receive input.* This program will ask the user for the SCALE output filename, the unit for the fuel rod, the mixture for the fuel, whether the assembly is rectangular or hexagonal, and a name for the type of assembly which will be displayed on the chart title. The majority of the input are strings with the exception of the material and zone numbers which are parsed into integers.

```
#           Write a routine to request input
#-------------------------------------------------------------------------------|
# Request name of SCALE ouput file
# Request compression factor for problem
println("Enter the name of the SCALE output file:")
F = readline()
# opening a file in read_mode
# r is the default mode
f = open(F, "r")
# Request type of assembly, hex or rectangular
println("Enter type of array cub for rectangular or hex for hexagon:")
arraytype = readline()
# Request zone number of fuel
println("Enter unit number of fuel from SCALE geometry:")
ut = readline()
ut = parse(Int64,ut)
# Request mixture number of fuel
println("Enter mixture number of fuel from SCALE mixing table:")
mat = readline()
mat = parse(Int64,mat)
# Request to plot either Flux or Fission Power
println("Enter either Flux or Power to plot:")
type = readline()
# Request to name of assembly type
println("Enter type of assembly:")
title = readline()
```

5. *Write routines and subroutines to execute desired calculations.* Referring to the logical flowchart, there are two major loops to gather the data from the output file. The program will first open the file in the read mode. It then then proceeds to go line-by-line to search through the file to find the first keyline. When found, it will go into a loop that will continue until the stopline if found. Within this loop, it will read the line and write it to separate text file. When the stopline is found, it continues to search for the second keyline and completes the same operation for the zone data.

   The use of **global** for the counter variables allows for the counters to be updated in multiple loops. As a technique, the global variables have separate names for the two major loops associated with the geometry table and the zone data table. However, because they are reset after the stopline, they could be reused in the second loop.

```
#  Write routines and subroutines to calculate results
#-------------------------------------------------------------------------------
# Open the file and gather the data needed
for lines in readlines(f)
    #read in the output file and find the geometry block
    if contains(lines, keyline) #finds geometry
        global instance += 1
    end
    if  instance > 0
        if lines != stopline  #reads until stop
        global countblock +=1
            if countblock > 2 #prints data to file
                open("geodata.txt", "a") do file
                write(file, lines, "\n")
            end
        end
        elseif lines == stopline #stopline terminates
            instance = 0
        end
    end #concludes gathering of geometry in separate file

     #read in the output file and find the fission and flux
    if contains(lines, keyline2) #finds zone data
        global instance2 += 1
    end
    if  instance2 > 0
        if lines != stopline2  #reads until stop
        global countblock2 +=1
            if countblock2 > 2 #prints data to file
                open("zonedata.txt", "a") do file
                write(file, lines, "\n")
            end
        end
        elseif lines == stopline2 #stopline terminates
            instance2 = 0
        end
    end #concludes gathering of zone data in separate file

    # increment line_count

    global line_count += 1

end
close(f)
```

The next step to compiling the data involves reading the two text files created for the tables into a delineated array. This process begins by reading the text file into an array named A for the geometry data. This array is then placed in a DataFrame. The DataFrame can then be queried to sort out the X and Y data only associated with the unit number of the fuel elements. The same process is applied to the zone data table by sorting out only the elements that have the fuel mixture and a power greater than zero.

```
#  Take the gathered data and arrange in a format that can be sorted
#reads geomtry data into an deliminated array
A = readdlm("geodata.txt")
header=["Type", "Unit", "Number", "nx", "ny", "x", "y", "A", "B", "Angle"]
rawdata = DataFrame(A, header )
data = identity.(rawdata)
q1 = @from i in data begin
    @where i.Type ==arraytype && i.Unit == ut
    @select {i.x, i.y}
    @collect DataFrame
end

#reads zone data into a delimited array
B = readdlm("zonedata.txt")
header2=["Zone",  "Mix", "Flux", "Power", "Volume"]
rawdata2 = DataFrame(B, header2 )
data2 = identity.(rawdata2)
q2 = @from i in data2 begin
    @where i.Mix == mat && i.Power > 0
    @select {i.Flux, i.Power}
    @collect DataFrame
end
```

6. *Write routine that handles output.* The program will then delete the text files as they are no longer needed. The program then combines the two dataframes into a single dataframe. The catenate function completes the combination.

   To get the program to properly present the surface plot, values must be assigned to the borders of the plot. The irregular surface plot will attempt to average unspecified z-axis values between known values. With the border, the values will go off in unusual directions when not available. This is especially confusing with a hexagonal lattice. The working area for the plot is rectangular between X and Y and there is a significant lack of data points on the corners of the hexagon.

   Because the program needs to be useful for either rectangular or hexagonal lattices, the parser will need to have two routines for addressing the borders. The method for solving the plot borders starts by completing a query in the combined dataframe to look for the maximum and minimum values for the flux and power.

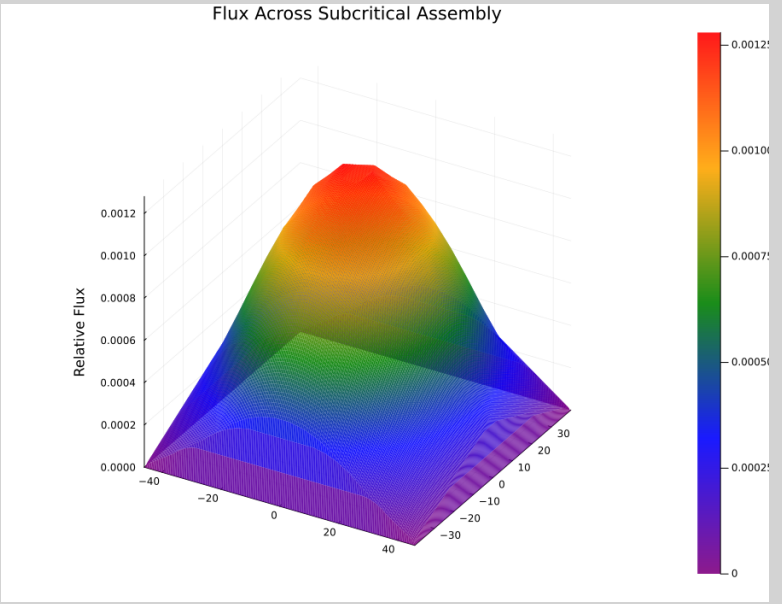
   The program then uses the type of lattice (hex or cuboid) to either apply a minimum value of zero for the hexagonal lattice or use the minimum value from the dataframe for the rectangular lattice.

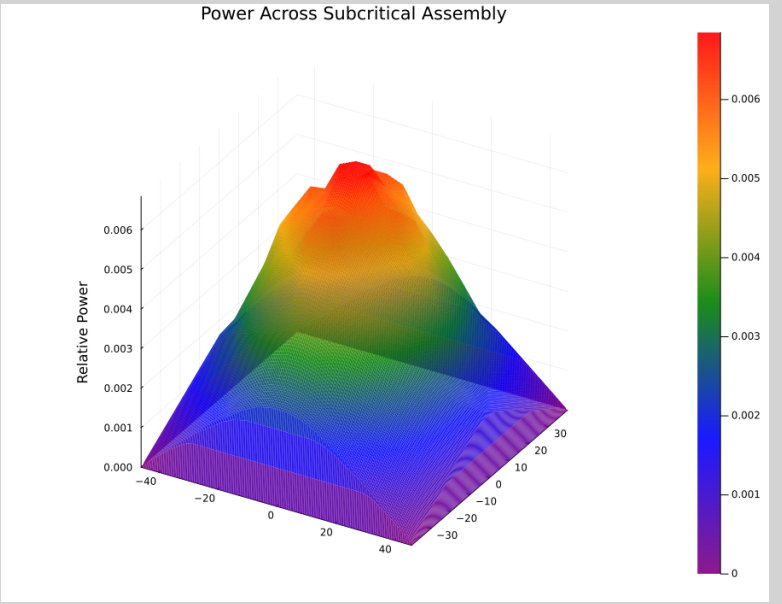
   The program concludes by transferring the dataframe information to a

CSV file and then plotting that information with the name of the assembly the user provides.

```
# Complete the sort and combine files and plot results
#delete text files that held data
rm("geodata.txt")
rm("zonedata.txt")
#combine two DataFrames
df1 = hcat(q1, q2)

maxxvalue = maximum(df1[!,:x])
minxvalue = minimum(df1[!,:x])
maxyvalue = maximum(df1[!,:y])
minyvalue = minimum(df1[!,:y])
minFluxvalue = minimum(df1[!,:Flux])
minPowervalue = minimum(df1[!,:Power])
minx = minxvalue-1
miny = minyvalue-1
maxx = maxxvalue+1
maxy = maxyvalue+1

if type == "Flux" && arraytype == "cub"
minz = minFluxvalue
elseif type == "Power" && arraytype == "cub"
minz = minPowervalue
else
minz = 0.0
end
C = [minx miny  minz minz;
     minx maxy  minz minz;
     maxx miny  minz minz;
     maxx maxy  minz minz]

header3 =["x","y","Flux","Power"]
borderdata = DataFrame(C, header3)
df = vcat(df1,borderdata)

CSV.write("plotdata.csv", df)

x = CSV.read("plotdata.csv", values, select=[1])
y = CSV.read("plotdata.csv", values, select=[2])
if type == "Flux"
    z = CSV.read("plotdata.csv", values, select=[3])
elseif type == "Power"
    z = CSV.read("plotdata.csv", values, select=[4])
end
plt = surface(x, y, z, c = :rainbow, alpha = 0.9, title="$type Across $title Assembly"
zaxis= "Relative $type", size = (900, 700))
display("image/png",plt)
```

The following plots are power and flux plots for a subcritical assembly that is a cylindrical core with a hexagonal lattice of fuel rods. Within the assembly, there are five non-fuel locations. The power plot shows the location of those rods as they do not have fuel to produce power. The plot representation is as expected with the peak at the center for flux and power tapering uniformly across the core.


Flux Across Subcritical Assembly
Relative Flux
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−40
−20
0
20
40
30
20
10
0
−10
−20
−30
0.0012
0.0010
0.0007
0.0005
0.0002
0


Power Across Subcritical Assembly
Relative Power
0.006
0.005
0.004
0.003
0.002
0.001
0.000
−40
−20
0
20
40
30
20
10
0
−10
−20
−30
0.006
0.005
0.004
0.003
0.002
0.001
0

8.7 Creating Three-Dimensional KENO Models in SCALE

The SCALE FULCRUM user interface makes converting our t-NEWT examples in 2-D to 3-D KENO problems easier than creating a new model from scratch. The SCALE package distribution contains the KENO-VI Primer in addition to the SCALE User's Manual. The "primer is designed to help the user understand and use the KENO-VI three-dimensional (3D) Monte Carlo code for nuclear criticality safety analysis. KENO-VI is part of the SCALE code system." (Bekar et al.; 2020)

The KENO criticality calculations are part of the Criticality Analysis Sequence (CSAS). The CSAS control module is the primary criticality safety control module for the calculation of the neutron multiplication factor of a system. As with the NEWT calculations we performed earlier, the FULCRUM interface will link together multiple modules to complete the desired calculation. For the KENO-VI calculations, the first module called will be CSAS which will complete the cross section processing for KENO. The user specifies the material compositions, isotopic compositions, density, and temperature along with the basic 1-D cell data, just like the first operation in the t-NEWT calculation. From there, CSAS will develop the cross sectional information and store it for the KENO calculation. The CSAS module will perform resonance shielding, temperature corrections, and a 1D discrete ordinates calculation automatically using the BONAMI, CENTRM, PMC and XSDRNPM functional modules. (Bekar et al.; 2020)

In addition to providing k-effective estimates for our criticality models, the KENO-VI models can be viewed in 3-D within FULCRUM and tallies can be applied across a mesh within the models to obtain either flux or fission source distribution results. These tally results can also be viewed in either 2 or 3-D within FULCRUM. For this brief introduction, we will provide two examples of KENO-VI calculations. The first is a 17 x 17 PWR model that is the same model presenting in Example 8.1. The second model is a SCFR assembly model that is the same composition as the SCFR assembly completed in NEWT and MCNP in Example 8.2. Note that the SCFR assembly model in Example 8.2 intentionally did not have the depleted uranium blanket modeled in MCNP. The model was modified so the material in the blanket region was the same as the enriched driver fuel. This was done because the SCALE NEWT model was 2-D and only modeled the reactor assembly across the mid-plane and no depleted uranium was present. With the KENO example, the model is 3-D and the axial blanket is modeled explicitly. This will all an comparison to an MCNP model with the correct materials in all regions.

Example 8.5
Write a KENO-VI SCALE input file for the Westinghouse 17 x 17 assembly from Example 8.1. Complete the criticality calculation and report the results. Compare to the t-NEWT solution from Example 8.1 and the MCNP solution from Example 6.5. Include a plot of the geometry in 3-D.

Solution: Start with the input file for the assembly from Example 8.1.

```
'Compuational Design in NE Example 8.1
'Kenneth Allen 23 June 2023
=t-newt parm=(bonami)
PWR Lattice Example
v7-252
' ENDF 7 cross section with 252 energy groups

read comp
atomuo2 1 10.2 2 92000 1 8000 2 1.0 1023
                92234 0.5 92235 4.5 92238 95.0 end
 zr-90 2 den=6.407000 1.0  end

helium 3  1.0 573.0 end

water 4 den=0.715000 1.0 573.0 end

end comp

read celldata
latticecell squarepitch hpitch=0.6555 4 fuelr=0.385 1
                        gapr=0.390 3 cladr=0.461 2 end
end celldata

read model
PWR Lattice Example

read parameters
sn=6

end parameters

read materials

  mix=1 pn=1 com='UO2 Fuel' end
  mix=2 pn=1 com='zirc cladding' end
  mix=3 pn=1 com='helium gap' end
  mix=4 pn=2 com='water' end

end materials

read geometry
' fuel unit cell
unit 2
cylinder 1 0.385
cylinder 2 0.390
cylinder 3 0.461
cuboid 4 0.6555 -0.6555 0.6555 -0.6555
```

```
'Define UO2 fuel
media 1 1 1
'Define gap
media 3 1 -1 2
'Define clad
media 2 1 -2 3
'define moderator
media 4 1 -3 4
boundary 4
' control rod guide tube unit cell
unit 3
cylinder 2 0.390
cylinder 3 0.461
cuboid 4 0.6555 -0.6555 0.6555 -0.6555
'Define water inside
media 4 1 2
'Define clad
media 2 1 -2 3
'define moderator
media 4 1 -3 4
boundary 4

' global unit to contain array
global unit 100
cuboid 10 11.14 -11.14 11.14 -11.14
array 1 10 place 9 9 0.0 0.0
media 4 1 10
boundary 10 40 40

end geometry

read array
       ara=1 nux=17 nuy=17 typ=cuboidal
fill
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
end fill
```

```
read bounds
all=refl

end bounds


end model


end
```

Modify the first module to reflect a CSAS calculation instead of a t-NEWT. The composition section will remain the same as well as the cell data section for the 1-D cross section calculations of the unit cell. The CSAS block will terminate following the cell data with an end statement.

```
'Compuational Design in NE Example 8.5
'Kenneth Allen 9 July 2026
=csas-mg
xsproc to prepare 252 group working format xsec lib for kenovi
v7.1-252

read comp

wtptuo2 1 10.2 2 92000 88.15 8000 11.85 1.0 1023
                92234 0.5 92235 4.5 92238 95.0 end


zr-90 2 den=6.407000 1.0 end

helium 3  1.0 573.0 end

water 4 den=0.715000 1.0 573.0 end

end comp

read celldata
latticecell squarepitch hpitch=0.6555 4 fuelr=0.385 1
                  gapr=0.390 3 cladr=0.461 2 end

end celldata

end
'end the csas module and start the keno problem
```

After the cross section processing is completed, replace the model, parameters, and materials with the KENOVI start, the parameters for the KENO model, and the dimensions for the grid for plotting tally information. The total number of parameters and their default values for KENO are listed in the SCALE 6.3.3 User's Manual in Table 8.1.1. For this model, we deviate from the default values for the listed parameters. In particular, we included the neutron production and the grid fluxes. By setting those values to "yes", the KENO calculation will complete a tally on the neutron production and fluxes by energy across a specified grid and make a 3D map file that can be loaded and viewed in FULCRUM. The next block after parameters specifies the grid mesh with X, Y, and Z planes for limits and the number of divisions divided evenly between specified upper and lower limits in X, Y, and Z. We have chosen to extend the mesh to encompass the entire model with a mesh density of approximately 0.5 cm in all 3 dimensions.

```
=kenovi
NE461 Example of PWR Assembly
'Table 8.1.1 from SCALE6.3.3 manual describes all
'parameters these are the ones changed from default
read parameters
'turn on fission densities
  fdn=yes
'library number to store cross section data
  lib=4
'deactivate the html output
  htm=no
'number of particles per generation
  npg=2000
'accumulate neutron production
  cds=yes
'compute grid fluxes
  gfx=yes
end parameters

read grid 1
xplanes  20 -20 end
yplanes  20 -20 end
zplanes  200 -200 end
xlinear  80 -20 20
ylinear  80 -20 20
zlinear  160 -200 200
' TODO: define grid

end grid
```

When working in KENO, the user must specify the mixtures to include isotope compositions for the media used in the geometry section. The isotopes specified in this module must be specified in the first block for the CSAS calculation and they must match. For example, if a naturally occurring elemental ZAID is used, like 40000, for the zirconium in the mixture, all isotope cross sections must be available for that element from CSAS. In some cases, it may be easier to not use elemental ZAIDs and specify each isotope in both modules. The mixture specification default is in atom density per 1×10^{24} cm^3. Because we are comparing our work to an MCNP file, we can use the MCNP output file to obtain the atom density for each mixture and multiply that value by the atom percent of each isotope to calculate the atom density for each isotope in this model. Alternatively, the value is calculated directly using the weight percent of each isotope, the isotope's atomic mass in AMU, the physical density of the mixture in g/cm^3 and Avogadro's number.

$$N_i = w_i \frac{\rho N_A}{M_i}$$

Note that Avogadro's number is 0.60221408 in this case to produce results in units of 1×10^{24} cm^3.

```
'create mixtures for model in atom density (atoms E24/cm^3)

read mixt
  sct=2
  mix=1  ncm=1
    92238       2.16099E-02
    92235       1.03496E-03
    92234       1.15488E-04
    8016        4.55107E-02


  mix=2  ncm=2
   40090        4.22952E-02

  mix=3  ncm=3
    2004        2.49752E-05

  mix=4  ncm=4
    1001        4.78138E-02
    8016        2.39068E-02

end mixt
```

After the mixture, the geometry block is updated.

```
read geometry
' fuel unit cell
unit 2
cylinder 1 0.385 182 -182
cylinder 2 0.390 182.1 -182.1
cylinder 3 0.461 182.2 -182.2
cuboid 4 0.6555 -0.6555 0.6555 -0.6555 182.3 -182.3
'Define UO2 fuel
media 1 1 1
'Define gap
media 3 1 -1 2
'Define clad
media 2 1 -2 3
'define moderator
media 4 1 -3 4
boundary 4

' control rod guide tube unit cell
unit 3
cylinder 2 0.390 182.1 -182.1
cylinder 3 0.461 182.2 -182.2
cuboid 4 0.6555 -0.6555 0.6555 -0.6555 182.3 -182.3
'Define water inside
media 4 1 2
'Define clad
media 2 1 -2 3
'define moderator
media 4 1 -3 4
boundary 4

' global unit to contain array
global unit 100
cuboid 10 11.14 -11.14 11.14 -11.14 181.5 -181.5
array 1 10 place 9 9 1 3*0.0
boundary 10

end geometry
```

Modifying the geometry block from the 2-D NEWT calculation to the 3-D

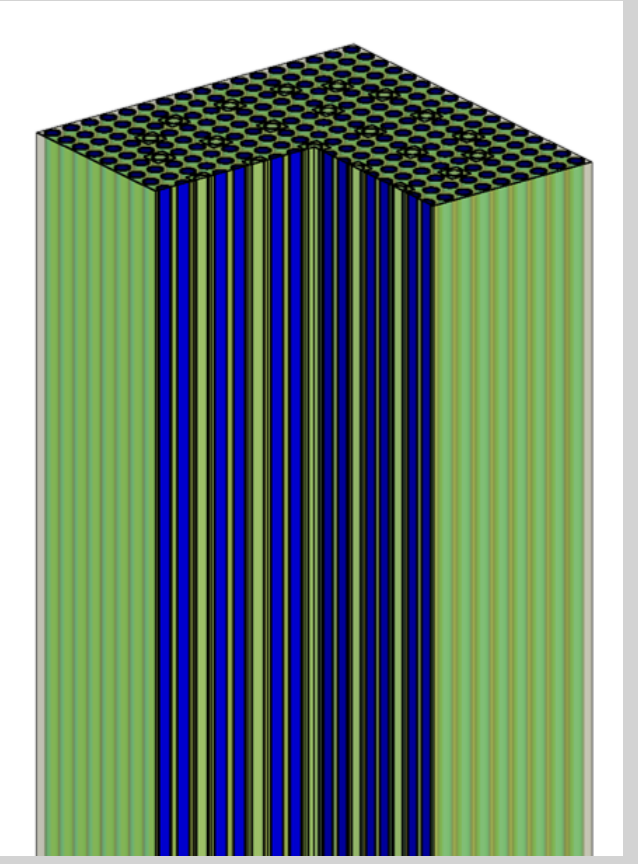
KENO model is straightforward. The units for the fuel and control rod guide tubes are the same with the addition of the Z dimension in centimeters for each of the cylinders and cuboids. Just as we did with MCNP, we will can offset the end points to prevent coincident planes on the ends of the rods and unit cells. The Z parameter in SCALE for the cylinder is a displacement from the origin of the model which differs from the method of specifying the RCC in MCNP where the user first specifies the base and then the displacement from that location in X, Y and Z.

The global unit is also modified with the addition of the Z-dimension for the cuboid. To ensure the array completely fills the global boundary, the height for the outside cuboid was set to a value shorter than the length of any element in either unit 1 or 2. The other modifications to the global unit include adding a place of 1 for the Z location and a 0 0 0 (written as 3*0.0) for the location of the origin. Finally the boundary is changed to remove the 40 40 which were mesh specifications in the discrete ordinate calculation and not appropriate for the Monte Carlo KENO calculation.

The final modifications for the input include adding a nuz=1 value to indicate the array only has one element in the Z direction, and to specify the volume calculation is random and changing the **end model** to **end data** to terminate the KENOVI module.

```
read array
       ara=1 nux=17 nuy=17 nuz=1 typ=cuboidal
fill
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 3 2 2 3 2 2 3 2 2 3 2 2 3 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2 2
       2 2 2 2 2 3 2 2 3 2 2 3 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
end fill
end array

read volume
  type=random
end volume

read bounds
all=refl
end bounds

end data

end
```

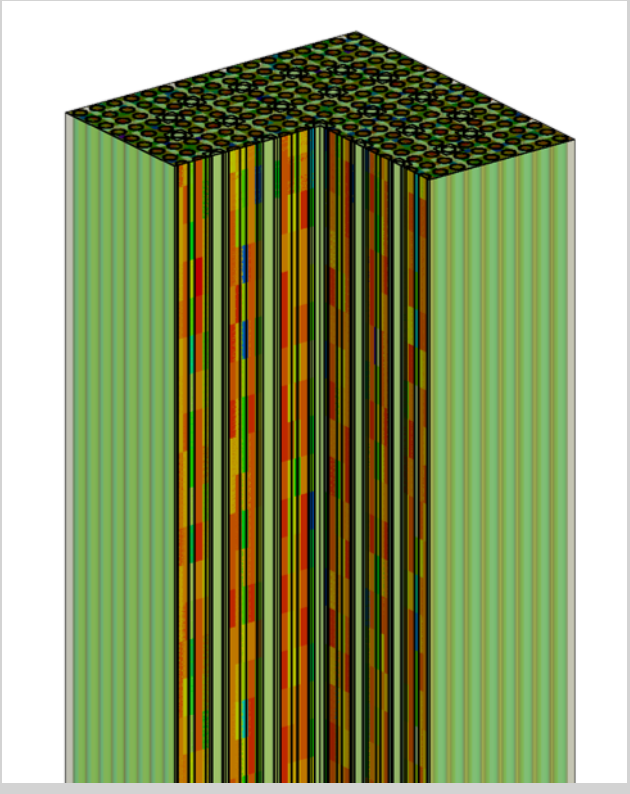
As with other modules in FULCRUM, the Validation tab at the bottom of the user interface will identify syntax errors and the user should check that it is clear prior to running the model. Additionally, the user can use the View tab at the top of the window to view the geometry. In addition to the 2-D geometry observed in other SCALE models, the user can view the model in 3-D in KENO by changing the view from Top (X-Y) to 3-D. Using the controls, the user can adjust the opacity of the material colors along with the zoom and add cuts to view the inside of the model as shown below.

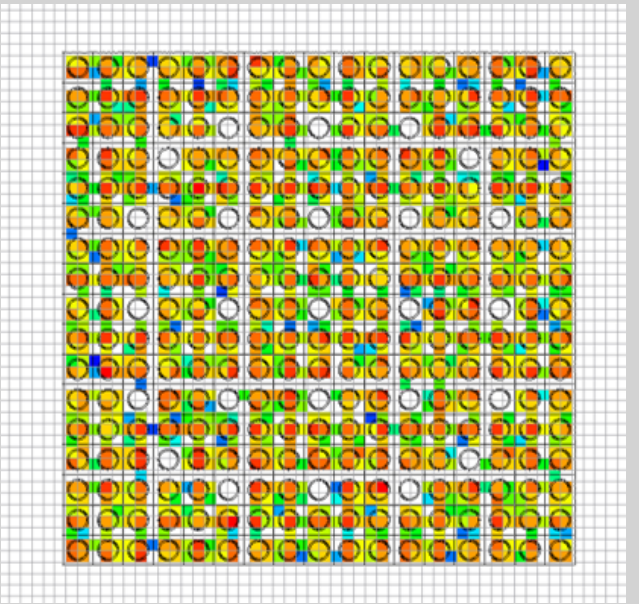


Upon selecting "Run" the model will execute the CSAS followed by the KENO sequence. During the run, the Messages window at the bottom of the screen will display information to include the cycles much like the KCODE execution in the command prompt in MCNP. When the model terminates, it will produce an output file along with several other files to include a folder with plots of the generational progress during the calculations. These are similar to the "tornado plots" seen in the KCODE output in MCNP. In addition,

because we requested tallies in the parameters section, there are two *.3dmap files created. One for the fission source and another for the flux. This data can be plotted in the geometry view in Fulcrum. First the user will select the "mesh" tab at the top of the geometry view and right-click anywhere in the window that says "Available meshes". The user can then select either of the *.3Dmap files. To plot the information, the user will need expand the loaded mesh and select the energy group value desired. The total value is located need the bottom of the list. Once selected, the user can change the plot from "Material" to "Overlay" or "Overlay with Boundaries". For this example, we have selected the total fission source distribution and produced the following 2-D and 3-D plots.

As expected the model indicated that fission does not occur in the control rod guidetubes that are not fueled.





The KCODE from MCNP for this model resulted in k_∞ = 1.41891 ± 0.00190. The final KENO result was 1.4275 ± 0.0011. As expected, the two models have similar values. The SCALE model is 0.6% higher. The difference between the models likely the result of the temperature differences in the fuel cross sections.

In Example 8.3, we created a t-NEWT SCALE and MCNP input for an assembly for the Clinch River Breeder reactor assembly presented in Figure 8.4 and Table 8.1. Because the t-NEWT model was in 2-D across the midplane, the MCNP model was adjusted to replace the axial breeder blankets with driver fuel and the results were compared for the criticality calculations. This modification clearly demonstrates the limitations of the 2-D model when the axial material is not homogeneous along the height of the core. Many other reactor cores will have variations along the length of the core due to changes in control rod positions or, like in BWRs, the density of the moderator will change along the height of the core. For these reasons, 3-D models are better at predicting overall criticality and simulating any changes in flux and power distribution within the core. In Example 8.6, the t-NEWT model is rewritten as a KENO 3-D model for the Clinch River Breeder reactor that includes the depleted uranium blanket on the top and bottom to compare to the MCNP model in Example 8.3 where that model is run with the depleted uranium in the blanket regions.

Example 8.6
Write a KENO-VI SCALE input file for an assembly for the Clinch River Breeder reactor assembly presented in Example 8.3. Include a plot of the geometry in 3-D.

Solution: Start with the input file for the assembly from Example 8.3.

```
'Computational Design in Nuclear Engineering Example 8.3
' SCALE NEWT for SCFR Assembly
=t-newt parm=(bonami)
hex lattice scfr
v7-252
read composition
 wtptfuel        1  10  9
                                 8016 11.85185
                                 92235 0.123
                                 92238 61
                                 94238 0.75
                                 94239 14.5
                                 94240 6.68
                                 94241 3.18
                                 94242 1.915063
                                 94244 8.7e-05
                       1 1200    end
 wtptss          2  7.8  5
                                 25055 2
                                 14000 1
                                 24000 19
                                 28000 10
                                 26000 68
                       1 1200    end
 na-23           3 den=0.927 1 735   end

end composition
read celldata
  latticecell triangpitch fuelr=0.2921 1 cladr=0.32 2
  hpitch=0.4032 3 end
end celldata

read model
read parameters
sn=2
kguess=1.75
dz=91.44
drawit=yes
height=91.44

end parameters

read materials

  mix=1 pn=1 com='Fuel UO2' end
  mix=2 pn=1 com='SS' end
  mix=3 pn=1 com='Na' end


end materials
```

```
read geometry
unit 1
com='fuel unit cell'
 cylinder 10   0.2921
 cylinder 20   0.32
 hexprism 30     0.42
 media 1 1 10
 media 2 1 -10 20
 media 3 1 -20 30
 boundary 30

global unit 2
com='outside of assembly'
 rhexprism 110      6.2
 rhexprism 120      6.3
 rhexprism 130      6.4
 array 1 110  place 10 10 0 0
 media 2 1 -110 120
 media 3 1 -120 130
 boundary 130 25 25
end geometry
read array
ara=1 nux=19 nuy=19  typ=shexagonal
 com=''
 fill
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 end fill
end array

read bounds
    all=white
  end bounds
end data
end model
end
```

The modification for the KENO model is the same process used for the PWR assembly. Note that the input file now contains a depleted uranium material in both the composition and KENO mixture tables. Also, the composition of the stainless steel cladding and can has been adjusted in the composition and mixture to detail more isotopes. The final composition exactly matches the MCNP input file for materials in Example 8.3. A second unit is designated as the unit cell for the depleted uranium. The global unit is changed from 2 to 100 and the new unit 2 matches the unit 1 with the exception of material 4 for the fueled area which is depleted uranium. Note that both units have a z-dimension that is the entire length of the fuel assembly. This is acceptable as they will be part of an array that is placed inside a region that is shorter than the unit cell and their final dimensions will be capped at the top and bottom by that array region.

The other major adjustment for the 3-D model occurs in the Global Unit and Array regions for the geometry. The Global unit now has three inner rectangular hexagonal prisms stacked on top of each other to delineate the three fueled regions. They are designated 110, 111, and 112. The array command now has two arrays. The first array remains the same with the driver fuel and

unit 1 and the second is an array of unit 2. Because we set the length of the unit cell equal to the height of the entire assembly, the same array can fill either the top or the bottom axial region and not have an area undefined. If we wanted, we could have created an array for unit 2 that only filled the top region and another array and unit that filled the bottom region, but it is unnecessary.

```
'Computational Design in Nuclear Engineering Example 8.6
' SCALE KENO for SCFR Assembly
=csas-mg
xsproc to prepare 252 group working format xsec lib for kenovi
v7.1-252
read composition
 wtptfuel        1  10  9
                                 8016 11.85185
                                 92235 0.123
                                 92238 61
                                 94238 0.75
                                 94239 14.5
                                 94240 6.68
                                 94241 3.18
                                 94242 1.915063
                                 94244 8.7e-05
                       1 1200    end
 wtptss          2  7.8  21
                                 6000     0.075
                                 7014     0.125
                                 14028    0.4611
                                 14029    0.0234
                                 14030    0.0155
                                 15031    0.03
                                 16032    0.015
                                 24050    0.774
                                 24052    15.066
                                 24053    1.71
                                 24054    0.414
                                 25055    8.75
                                 26054    3.9153
                                 26056    62.0836
                                 26057    1.4171
                                 26058    0.135
                                 28058    3.4
                                 28060    1.31
                                 28061    0.055
                                 28062    0.18
                                 28064    0.045
                       1 1200    end
 na-23           3 den=0.927 1 735   end

 wtptblanket     4  10 3
                                  8016    11.8466
                                 92235    0.174084
                                 92238    87.9793
                       1 1200   end


end composition
read celldata
  latticecell triangpitch fuelr=0.2921 1 cladr=0.32 2
  hpitch=0.4032 3 end
end celldata

end
'end the csas module and start the keno problem
```

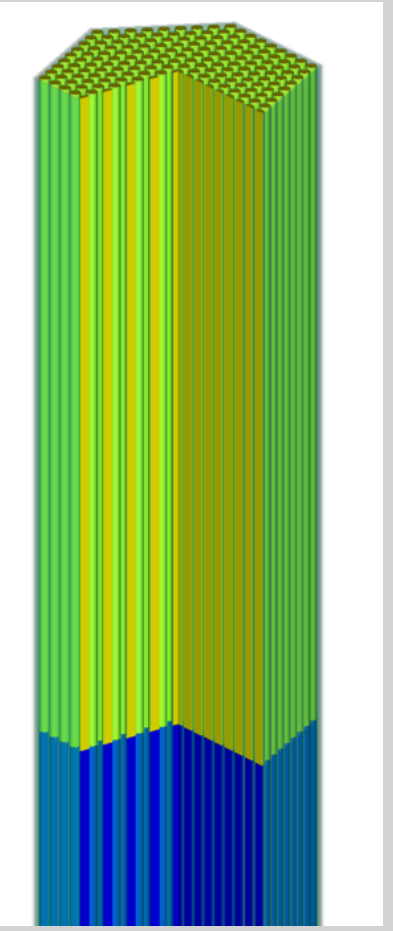
```
=kenovi
NE461 Example of SCFR Assembly
read parameters
  fdn=yes
  nub=yes
  lib=4
  htm=no
  npg=2000
  cds=yes
  gfx=yes
end parameters

read grid 1
xplanes  20 -20 end
yplanes  20 -20 end
zplanes  80 -80 end
xlinear  100 -20 20
ylinear  100 -20 20
zlinear  150 -80 80
' TODO: define grid

end grid
'create mixtures for model in atom density (atoms E24/cm^2)
read mixt
  sct=2
  mix=1  ncm=1
    8016        4.46219E-02
    92235       3.15137E-05
    92238       1.54314E-02
    94238       1.89731E-04
    94239       3.65275E-03
    94240       1.67576E-03
    94241       7.94422E-04
    94242       4.76438E-04
    94244       2.14664E-08

  mix=2  ncm=2
     6000       2.93838E-04
    7014        4.20062E-04
    14028       7.75573E-04
    14029       3.80012E-05
    14030       2.43343E-05
    15031       4.55780E-05
    16032       2.20774E-05
    24050       7.29235E-04
    24052       1.36496E-02
    24053       1.51997E-03
    24054       3.61182E-04
    25055       7.49484E-03
    26054       3.41574E-03
    26056       5.20774E-02
    26057       1.17124E-03
    26058       1.09656E-04
    28058       2.76161E-03
    28060       1.02861E-03
    28061       4.24768E-05
    28062       1.36776E-04
    28064       3.31245E-05

  mix=3  ncm=3
    11023       2.42823E-02

  mix=4  ncm=4
    8016        4.46020E-02
    92235       4.46020E-05
    92238       2.22564E-02
end mixt
```

```
read geometry
'driver fuel in center
unit 1
com='fuel unit cell'
 cylinder 10    0.2921 81.5 -81.5
 cylinder 20    0.32   81.5 -81.5
 hexprism 30      0.42    81.5 -81.5
 media 1 1 10
 media 2 1 -10 20
 media 3 1 -20 30
 boundary 30

'DU Axial Blanket top
unit 2
com='DU Blanket fuel unit cell'
 cylinder 10    0.2921 81.5 -81.5
 cylinder 20    0.32   81.5 -81.5
 hexprism 30      0.42    81.5 -81.5
 media 4 1 10
 media 2 1 -10 20
 media 3 1 -20 30
 boundary 30

global unit 100
com='outside of assembly'
 rhexprism 110       6.2   45.72 -45.72
 rhexprism 111       6.2   81.5 45.72
 rhexprism 112       6.2   -45.72 -81.5
 rhexprism 120       6.3   81.5 -81.5
 rhexprism 130       6.4   81 -81
 array 1 110  place 10 10 1 0 0 0
 array 2 111  place 10 10 1 0 0 0
 array 2 112  place 10 10 1 0 0 0
 media 2 1 -110 -111 -112 120
 media 3 1 -120 130
 boundary 130
end geometry
read array
ara=1 nux=19 nuy=19  typ=shexagonal
 com=''
 fill
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 end fill
```

```
ara=2 nux=19 nuy=19  typ=shexagonal
 com=''
 fill
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
      2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
end fill
end array

read volume
  type=random
end volume
read bounds
    all=white
  end bounds

end data



end
```

The KCODE from MCNP for this model resulted in $k_\infty = 1.544129 \pm 0.001487$. Note that this is with the depleted uranium in the blanket regions which significantly reduced the criticality from the model in Example 8.3 with the driver fuel along the entire length. The final KENO result was 1.57352 ± 0.00099. Once again, as expected, the two models have similar values. The SCALE model is 1.9% higher which is a simliar difference between SCALE and MCNP experienced in Example 8.1 and is likely due to temperature differences in the cross section processing.

A plot of the 3-D geometry with the cladding and coolant opacity turned down zoomed in along the top shows the two different fueled regions.



Bibliography

Bekar, K. B., Clarity, J. B., Dupont, M., Lefebvre, R. A., Marshall, W. B. and Saylor, E. M. (2020). Keno-vi primer: Performing calculations using scale's criticality safety analysis sequence (csas6) with fulcrum, *Technical report*, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States).
URL: *https://www.osti.gov/biblio/1760129*

Lefebvre, R. A. and Wieselquist, W. (2025). Scale 6.3.3 user manual, *Technical report*, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States).
URL: *https://www.osti.gov/biblio/3002301*

Simopt (2023). Vver assembly.
URL: *https://www.energyencyclopedia.com/en/nuclear-energy/nuclear-fuel/fuel-assemblygid=1pid=1*

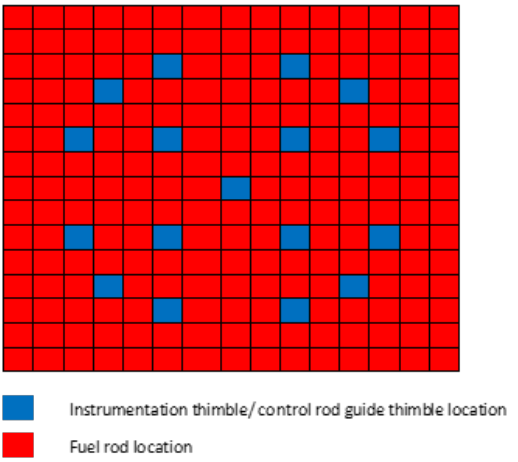
8.8 Video Tutorials

The following video tutorials are available to assist your understanding of the material from this chapter.

- Julia Parser Use This video shows the parser program developed in Example 8.4 being used for a hexagonal and rectangular assembly.
- SCALE Plotting Lattice Output This video details using Textpad, Excel and PSI-Plot to achieve 3D plots from SCALE output. This process is better accomplished by the parser.
- Flux Mapping Note that this video details how to plot flux from F4 tallies from a hexagonal lattice output in MCNP using a text editor, spreadsheet, and PSIPlot. The process is complex and should highlight the need to write a parser program to automate the process.
- MCNP Parser Flux Mapping This video details the use of a parser written in Julia to plot the flux from F4 tallies from either a hexagonal or rectangular lattice output in MCNP.

8.9 Problems

1. Another widely used PWR assembly on a square lattice is a Framatome 15 x 15 model. The arrangement of the fuel pins and control rod guidetubes for the Framatome model is shown below:



   Create SCALE NEWT model of the Framatome assembly using the dimensions and specifications for the unit fuel cell and the instrumentation/ control rod guide tube unit cells from Example 8.1 in the text.

2. Given the PWR assembly from Problem 1, re-run the calculations in t-NEWT to calculate the fuel temperature coefficient for the infinite reactor model. Use an SN=6 quadrature and adjust the fuel temperature from room temperature (293 K) to 2093 k in increments of 200 k.

3. Change the density of the water in the assembly in Problem 1 from 1.0 g/cm^3 down to 0.5 g/cm^3 by a step interval of 0.05 g/cm^3. Run each model and record and plot the results where void is 1.0 - density in the x-axis. Use the slope of a trendline to estimate the void coefficient as the change in reactivity as a function of percent void.

4. Complete a burnup calculation on the assembly from Problem 1 using the TRITON module for SCALE. Use an appropriate specific power, plot the results and note where the reactor goes subcritical.

5. Use the Julia parser program presented in Example 8.4 to plot the relative flux and fission power for the assembly from Problem 1.

6. Add six control rod guide tubes evenly spaced in the Clinch River Breeder Reactor Assembly t-NEWT problem presented in Example 8.2. Plot the geometry and run the model. Report the k-infinity value for the assembly.

7. Complete a fuel temperature coefficient calculation on the SCFR model from Problem 4 using the parameters in problem 2 and plot the results.

8. Complete a void coefficient calculation on the SCFR model from Problem 4 using the parameters in problem 3 and plot the results. Is the coefficient positive or negative? Does it make sense?

9. Complete a burnup calculation on the assembly from Problem 4 using the TRITON module for SCALE. Use an appropriate specific power, plot the results and note where the reactor goes subcritical.

10. Use the Julia parser program presented in Example 8.4 to plot the relative flux and fission power for the assembly from Problem 4.

Chapter 9

Attila4MC®

New users to MCNP quickly discover the challenges associated with building complex geometries with surfaces and cell cards. Getting the geometry correct can often be the most difficult part of an MCNP calculation. When computers moved from punch card and later plain text input, several front-end programs were created with the goal of simplifying the geometry construction process in MCNP. One of the first programs was SABRINA. As early as 1984, SABRINA was billed as, "an interactive, three-dimensional, geometry-modeling code system, primarily for use with MCNP. SABRINA's capabilities include creation, visualization, and verification of three-dimensional geometries specified by either surface- or body-base combinatorial geometry; display of particle tracks are calculated by MCNP; and volume fraction generation." (West; 1986)

From SABRINA to VISED and everything in between there have been several attempts to create a single program that would allow MCNP to build geometries, run MCNP problems, and visualize results. None of the attempts has become widely accepted and supported. With the combination of MCNP5 and MCNPX, Los Alamos Laboratory ended support for VISED leaving the integrated plotter as the primary visual plotter available for MCNP6. As we have seen, the integrated plotter can be challenging to use on Windows computers and new users find it easier to debug and plot problems in VISED. However, there are functions within MCNP6 that are not supported in VISED and slowly the program is becoming less compatible as advances in MCNP are not matched in VISED.

Many of the issues with early front-end programs involved a lack of financial support and a lack of flexibility in their use. Some would work well with certain geometries or sources but would not support others. There were also limitations within the computer environments where the codes were operating. Codes that had high graphical memory requirements could not operate on most personal computers.

As computational speed increased, additional functionality was built into MCNP allowing the calculation across unstructured meshes. This feature changes the requirements for geometry input and allows for complex geometries to be built and run on personal computers in reasonable times. Unstructured meshes can be created with a variety of methods and codes. For our purposes, we will use the Attila4MC® computer code to create our unstructured mesh problem and run MCNP.

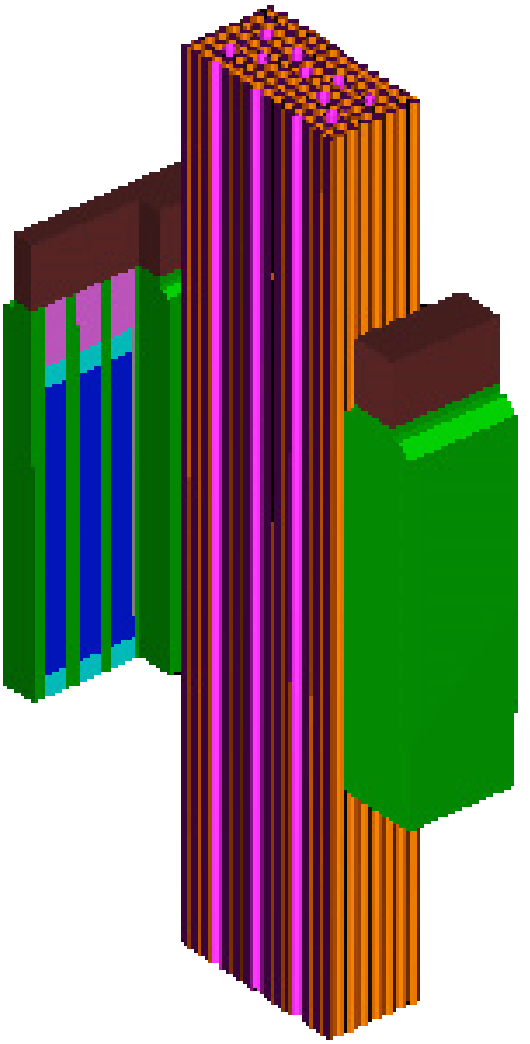


Figure 9.1: Sabrina model of uranium neutron coincidence collar (UNCL) with fuel assembly in center. (Root; 2014)

9.1 Introduction to Attila4MC®

The Attila4MC program is made and distributed by the Silver Fir corporation. From the Silver Fir website, "Attila4MC was developed to help MCNP® users become more productive. While the accuracy of MCNP is widely established, it relies on command line input for defining model geometry, calculation input, and variance reduction, which can often create significant bottlenecks limiting calculation throughput. Attila4MC provides an easy-to-use graphical interface, allowing novice and advanced MCNP users to easily set up, run, and visualize MCNP solutions from CAD data. Attila4MC leverages the state-of-the-art Attila deterministic solver to easily generate highly efficient weight windows variance reduction, shortening MCNP calculation times and substantially reducing user investment over manual variance reduction techniques. Once a calculation is performed, Attila4MC provides intuitive and advanced solution visualization options, aiding solution verification and allowing solution field insights to be easily conveyed to external parties." (SilverFir; 2023)

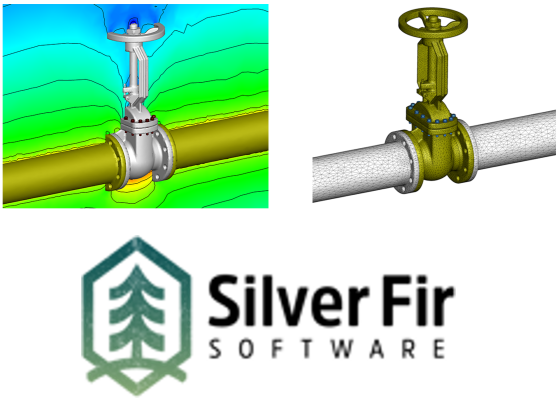


Figure 9.2: Attila4MC® from Silver Fir

Attila4MC® is a graphical user interface (GUI) that allows the user to set up, run and visualize MCNP solutions from CAD data. As mentioned in their web introduction, the program has its own deterministic solver, CADIS, that will generate weight windows for variance reduction. This will increase the accuracy of the results while reducing computer time, especially for thick shielding problems. The program interfaces with MCNP on the user's computer and will setup and run problems all within the GUI interface. The user can also choose to have Attila4MC assist in the setup of the unstructured mesh problem and then he or she can export the input file and run it separately.

Attila4MC uses TecPlot® made by ANSYS Incorporated for visualization

within the user interface. There are many options for visualization to include three-dimensional models and surfaces that display flux and or dose tally information on the CAD model. The user can also export the EEOUT file produced by the unstructured mesh run in Attila4MC® and use a Python script provided by Los Alamos National Laboratory to parse the file into a format that can be read by another visualization code, ParaView. ParaView® is free for download while TecPlot® can be purchased with the Attila4MC license.

The Attila4MC® license gives the user access to the video tutorials and other resources on the Silver Fir website. The user also gets access to upgraded versions of the code and update notices as part of the community. Within the Attila4MC tutorials, the primary CAD program used is SpaceClaim® which is also made by ANSYS. SpaceClaim® does not come with the Attila4MC license, however, most CAD programs can save models in the Parasolid format which is used by Attila4MC® to generate the mesh for the problem. For our problems, we will continue to use Solidworks® as our CAD program.

9.2 Creating Parasolids in Solidworks®

It is important to note that Attila4MC still requires skill and knowledge to get results. The user must have a good understanding of MCNP to achieve acceptable, realistic, and believable results. To get the Parasolid information into Attila in a manner that will allow it to quickly and accurately represent the problem and provide good results, the user must pay close attention to how the program generates the mesh. Fortunately, there are numerous help topics and buttons within the GUI and Silverfir has many video tutorials that show and explain a variety of problems to include mesh generation.

In general, obtaining a suitable mesh is the most challenging aspect of using Attila4MC. There are several best practices to help avoid issues with mesh generation. Note that the geometry mesh will consist of tetrahedral elements. These are groups of planar surfaces that represent the various components in cells where they join to make a closed object. Think of a brick wall where each brick represents part of the wall. The material is the same for each brick and rather than describing the wall as a single cell, it is a stack of smaller cells. The Parasolid is imported into Attila4MC, and the user specifies the various parts or sub-assemblies within the assembly as regions. These regions can then be independently broken into tetrahedral cells. Regardless of the fineness of the mesh, planar regions will be explicitly represented. However, curved surfaces will be approximated by a series of straight-edged surfaces. The accuracy of the model in these regions will depend on how fine the mesh is made. It should be apparent that the finer the mesh, in general, the more accurate the model will be. However, this approach will make the model more complex and require more computer time and memory.

The ability for Attila4MC to mesh the model properly depends on the model itself. If, for example, the user intends on applying the deterministic CADIS code to generate weight windows, the overall model must be in a rectangular volume. This is because CADIS applies a Cartesian grid on the model, and it must be square and aligned with the coordinate axis. Additionally, there are several methods for mesh generation to include MCNP and Attila non-manifold. The Attila non-manifold option is required for CADIS calculations and will identify empty regions within the model but requires better precision in the CAD model for successful meshing. If the user desires all regions to be presented with an MCNP load of the model, he or she will have to explicitly define those regions as parts within the CAD assembly. For example, a model that includes a person standing in a room with a radioactive source would require the air filling the area within the room to be a part in the assembly to get dose contours in those areas or to account for attenuation within the air.

Regardless of the type of Attila4MC problem, care must be taken to ensure that individual parts within the CAD model do not overlap or are undefined. The code must be able to distinguish between the various parts in the assembly to apply the appropriate mesh which will result in the cells and materials to be properly assigned in the MCNP input file. This is similar to the issue in MCNP where a model can have coincident planes or undefined regions which will cause the model to fail due to lost particles. When creating an assembly, the user should make use of mates within the model that precisely fit parts together. In some cases, it may make sense to leave intentional gaps between surfaces to prevent unintended overlapping parts. For example, a model of a person or object on a floor with an uneven surface can be offset from the ground by a small fraction which will not affect the overall results of the calculation, but will prevent a region where two objects occupy the same volume.

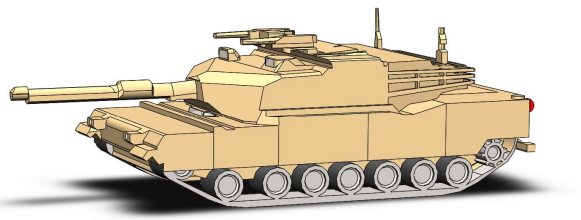


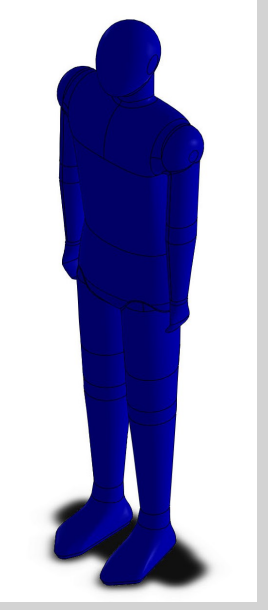
Figure 9.3: M1 Abrams tank model downloaded from GrabCad.

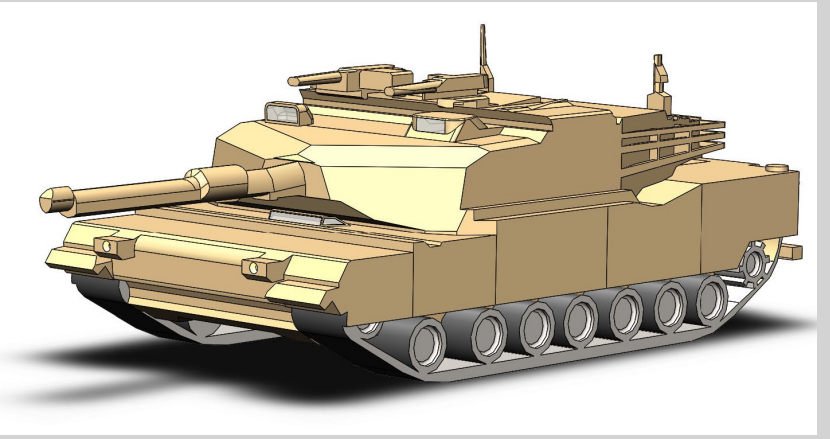
The CAD community online is a great resource for finding parts and objects to include in Attila4MC calculations. Various sites online to include GrabCAD and Yeggi offer users the ability to upload and download objects in a variety of CAD formats. Many models are posted for free use. When building a com-

plex assembly, the user may be able to make use of CAD parts that support their model in their CAD software format. Of course, proper citations should be given to any intellectual property accessed from these sites. Example 9.1 demonstrates how to make a CAD model of two soldiers standing at a fixed distance from a spherical source. One of the soldiers is shielded by an M1 Tank. The tank and human parts were downloaded from GrabCAD.

Example 9.1
Create a Parasolid model with two humans an M1 Abrams tank, soil and a sphere to represent a radioactive source. The two personnel will be arranged so that they are equidistant from the source with one of them shielded by the tank.

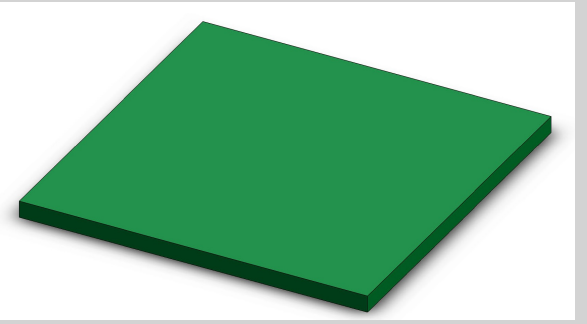
Solution: Begin by downloading part models for an M1 Tank and a human. There are options available for each on multiple sites for free. Care must be used when selecting parts from outside sources. I addition to the obvious dangers of downloading corrupt files or viruses, if the downloaded model is too detailed, the model may not mesh correctly, or it may take too long to run to completion. There are no set rules for determining whether a model is going to work or not. For this example, we will be making an assembly of parts. If we download an assembly from the internet for one of our objects, we can use Solidworks to "save as" a part. This is important because, when the Parasolid is imported into Attila4MC, we will specify that it was created by parts. Each part will represent the individual materials in the model. We will make the simple assumption that that tank is one material. The parts we downloaded are presented below:

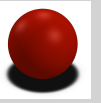


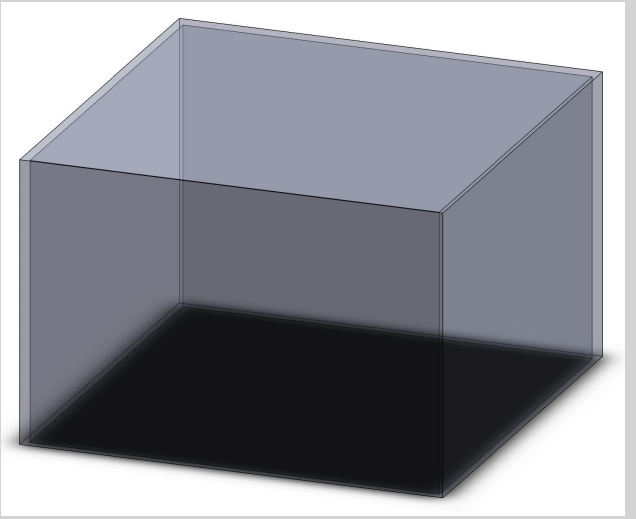


Because the two humans will need to have separate tallies associated on them, we will need to save the human part as "soldier1" and another copy as "soldier2". If we were only making an assembly, we would import the same part twice. However, if we did that, Attila would not distinguish between the two soldiers, and we could not get a tally on each and compare the results of standing behind the tank.

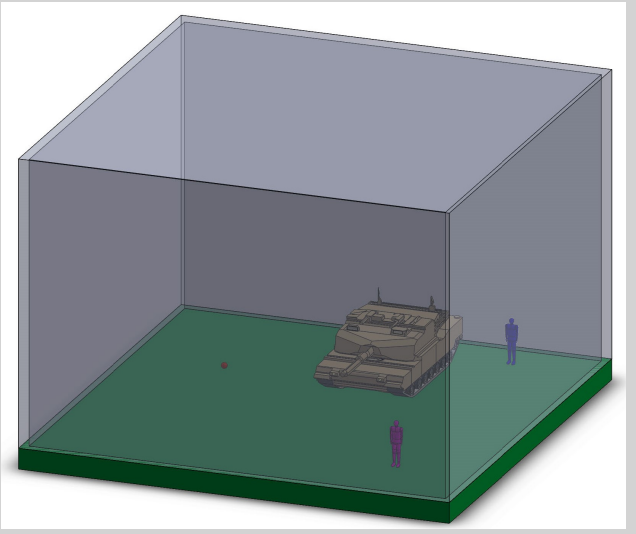
Using simple CAD skills, we create the block of soil, sphere for the source and a hollow box border for the air that is like a display case. The base is a square of earth 750 in x 750 in and 3 ft thick while the source is a sphere with a diameter of 20 cm. For this model, we will not fill in the empty region. The air box is not needed for the analog MCNP run, but it will be needed for a CADIS calculation to create the rectangular mesh so we are adding it now. The additional parts are below:







The final assembly is arranged to allow the source to be in a direct line of sight to both soldiers at equal distance. The difference is the tank in between one of the soldiers. When the tank and the soldiers are added to the assembly, a small air gap is placed between them and the soil. This air gap ensures that there are no overlapping parts. The air box is mated to the soil so it precisely fits on top and the interior region is big enough to ensure it does not intersect with the tank or soldiers inside. The final assembly is below:



Once the assembly is created, it is just a matter of saving as a Parasolid. Some versions of Attila4MC may require an older Parasolid version than what

is available. If that is the case, select options under the “save as” “Parasolid” window and change the version to match the maximum version Attila4MC will support. For our purposes, when we save the assembly as a Parasolid, we will to select "options" and choose a version 26.0 and check flatten assembly hierarchy to ensure it is compatible

9.3 Mesh Generation

After our Parasolid is built, we can move on to Attila4MC to import the solid body and generate the mesh. Mesh generation is one of the most important steps to getting good results that accurately represent the problem in a timely manner. The Attila4MC interface is a familiar window-based GUI with intuitive menus and help options. Silver Fir has several example cases on their website with the geometry, source, cross section, and response functions needed for calculations. Additionally, in the “Downloads” section, Silver Fir has a variety of source and cross section libraries for download and use.

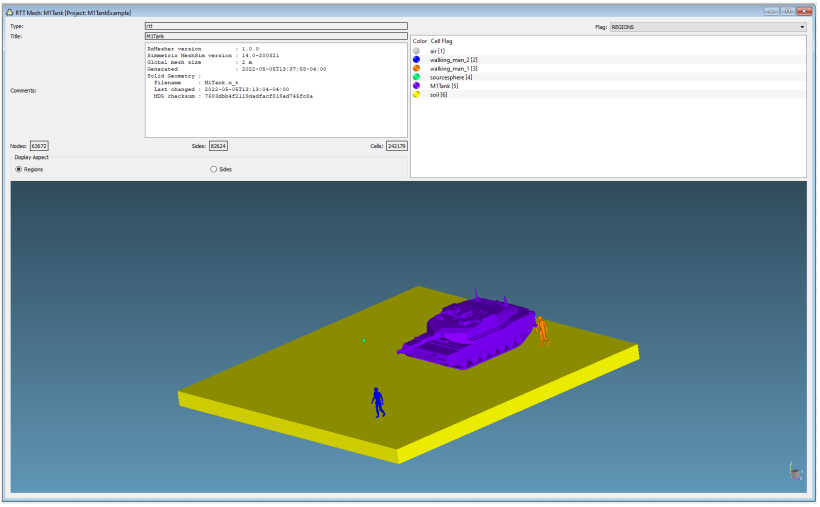


Figure 9.4: Window of solid model in Attila4MC with air hidden.

After loading the solid model into Attila4MC, we will need to determine if we are going to have a contiguous or dis-contiguous mesh. A contiguous mesh, shown in Figure 9.5, is where elements on adjacent part boundaries share the same nodes. A dis-contiguous mesh, shown in Figure 9.6, is a mesh where no nodal connectivity exists between adjacent parts and background space (e.g. air, void) only meshed where solution is desired. For the Attila solver, particles (flux) can only travel through the mesh. For MCNP, the solver can track particles through the dis-contiguous mesh. (SilverFir; 2023)

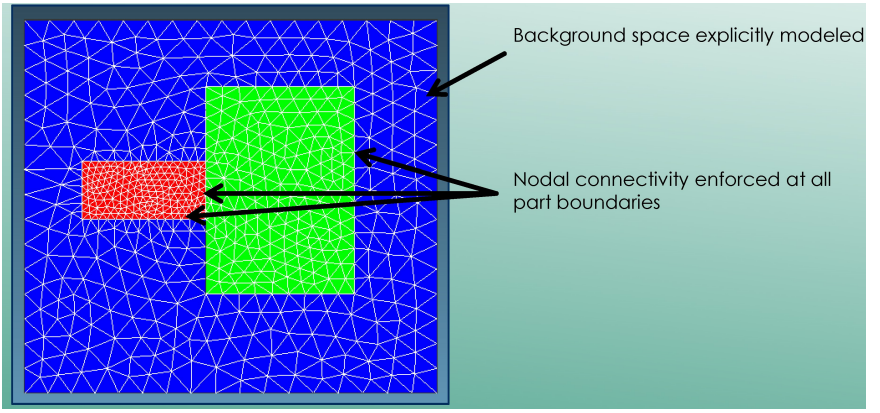


Figure 9.5: Attila4MC contiguous mesh (SilverFir; 2023).

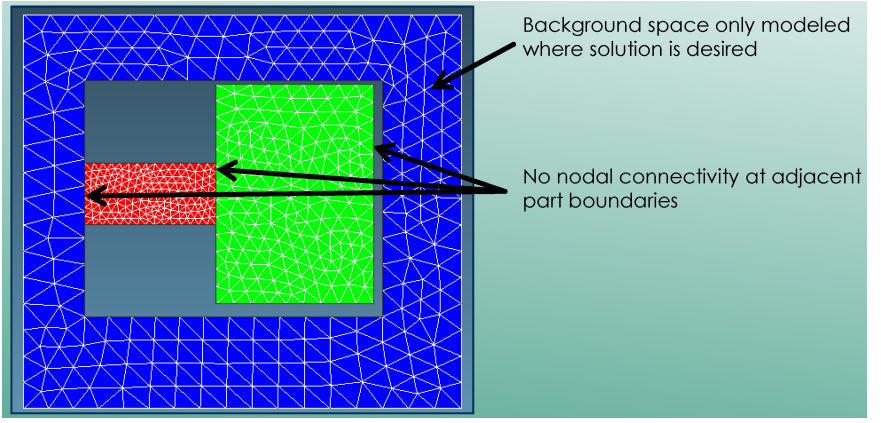
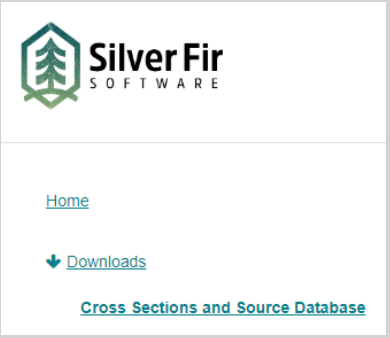


Figure 9.6: Attila4MC dis-contiguous mesh (SilverFir; 2023).

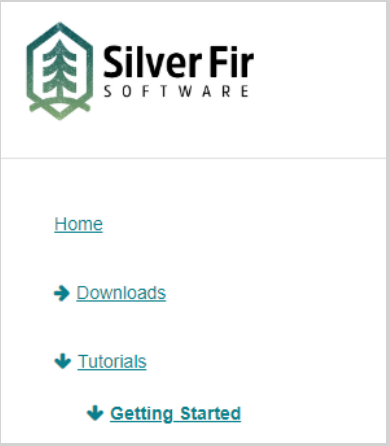
In Example 9.2, we will demonstrate creating a new Attila4MC project, loading the Parasolid from Example 9.1 and completing the dis-contiguous mesh operation on that model.

Example 9.2
Create a new Attila4MC project, load the Parasolid created in Example 9.1, and generate a mesh using the MCNP load option for a dis-contiguous mesh.

Solution: Create a folder on the computer that has both Attila4MC and MCNP6 installed. This will be the working folder for the problem. Open the Silver Fir website and navigate to the "Downloads" and "Cross Section and Source Database" page.

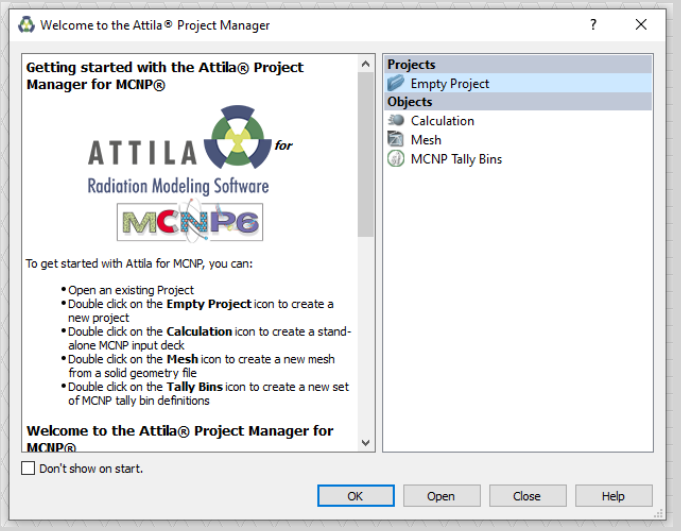


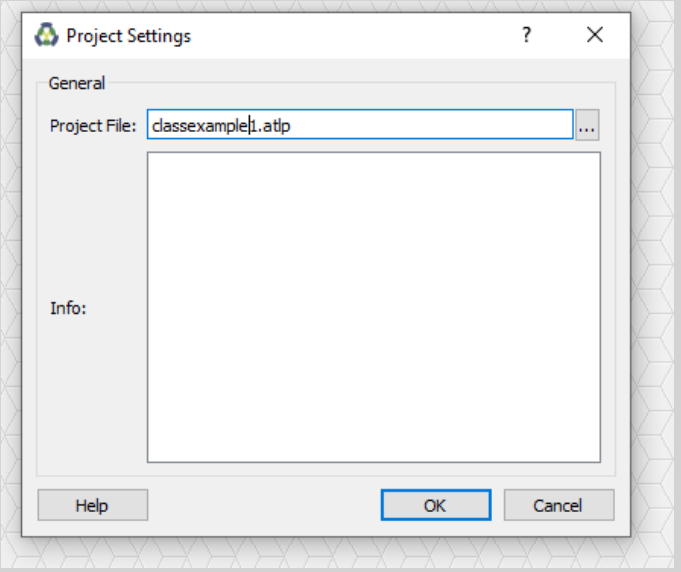
Select and download the XSmc (MCNP) cross section files for this problem. Go back to the main Silver Fir website, navigate to the "Tutorials" and "Getting Started" page.



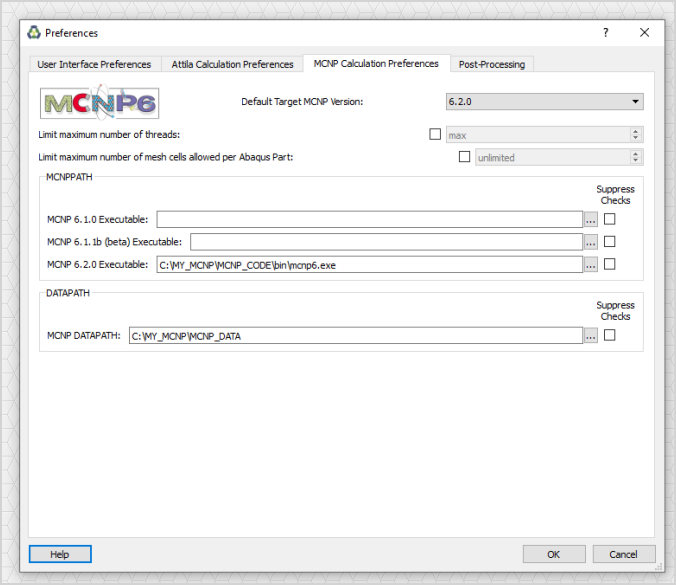
Download the "Getting Started Exercise Project Files" and unzip the file folder. Within that folder, there is a file named, "ICRU57.resp.inp". This is the response function we will use for our Attila4MC models. It will convert gamma and neutron tally flux to dose response. There is another file named "GammaSpectra.spec.inp" which is a Cobalt-60 spectra. For this example, we will use this spectra file. We could also select Cobalt-60 from the Source Spectra file from the large "Source Spectra" file available from the same downloads where we retrieved the cross section file.

Copy the Parasolid, cross-section definitions, response function, and gamma spectra files to the same folder. Start Atilla4MC and select and double-click "Empty Project" and provide it a name and some information (optional).

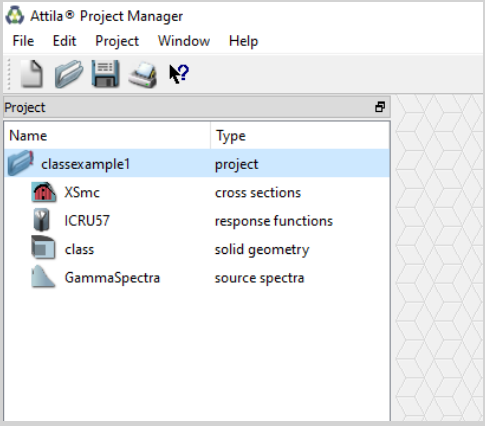




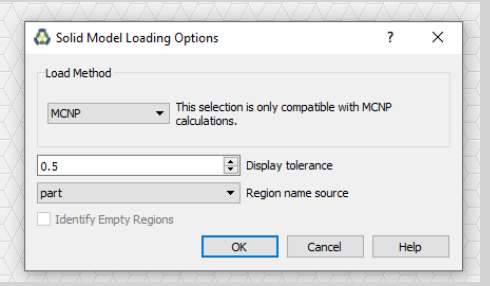
Whenever you start using Attila4MC it is important to verify that the code is setup with the preferences that allow it to start and run MCNP. On top menu select "Edit" and "Preferences". Check the preferences to ensure the path for the executable and data are correct and the version of MCNP is correct.



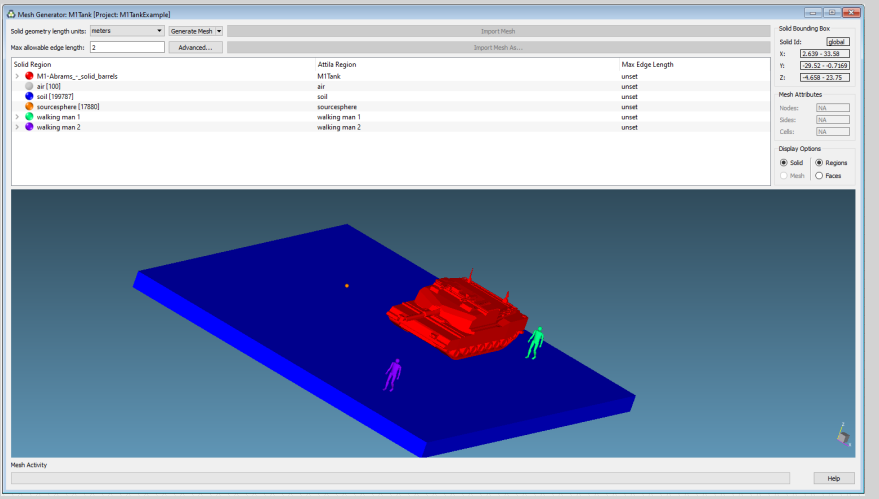
Next import files into the project tree. To load files, right-click on the file-name for the project and use the menu to load cross-section, response function, solid body and source definition files.



Begin the mesh generation process. Right-click on the solid geometry to open the mesh generator. Select MCNP for the load method along with region name of “part”. This is important because the Parasolid was made from parts not from sub-assemblies



The mesh generator will load with a preview of the part in the lower window. We have options to view in 3D rotation (click-and-drag the left mouse button), pan (click-and-drag the middle mouse wheel), and zoom (rotate the middle mouse wheel or select ctrl-left and click the mouse button). In the preview we will inspect the part. We can move rotate it and then select a part with the left mouse and then right-click and hide the part to see layers. Note the overall dimensions and global coordinates in the Solid Bounding Box in upper right. Double check with geometry length units. We can change region names by double-clicking names and changing them. Set the Max Allowable Edge Length. This is applied to the mesh edges in the model, restricting edge lengths at or below this value. Setting a Max Allowable Edge Length between 1/10th and 1/20th of the max model extents is generally recommended. For our model 2 meters is good. Set the Max Edge Length for regions. We are not going to change the max edge length because there are better ways to refine the mesh using the Advanced options.



We will select the down-arrow on the Generate Mesh Button and then select Generate Surface Mesh Only to preview how the mesh will look without

any options Note the roughness of the mesh for small curvatures around items like the source sphere, the barrel of the tank and the people's heads.

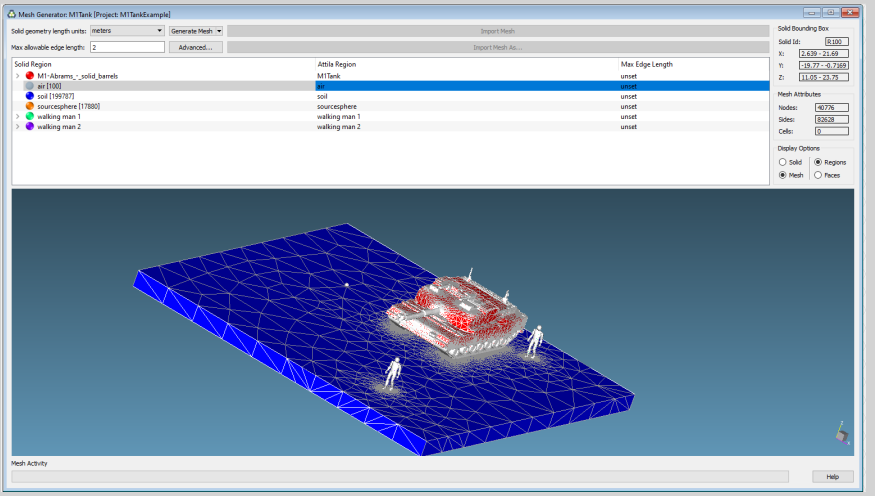
Select Advanced to open the Advanced Mesh Generation Attributes window Advanced Mesh Generation Attributes allow the user to control transitions between mesh sizes, curvatures in the mesh, and provide the user with the ability to refine the mesh anisotropically. The features in this window are typically applied to aid in resolution or reduce element count to decrease calculation times.

Select Curvature Refinement and keep the default values for Size Transition Factor, Maximum d/h, and Minimum edge length of 1.00, 0.02, and 0.05 The Size Transition Factor controls the rate of transition from fine to coarse elements and can be used to further reduce the element count in the mesh. Curvature Refinement allows the mesh to appropriately mesh curved solid bodies. Maximum d/h is a dimensionless value, and 0.02 is sufficient for most applications. The Minimum Edge Length is model specific, and, generally, can be to set to 1/3 of the smallest diameter curvature desiring resolution. The Refine Anisotropically feature will reduce the cell count by refining the edge lengths only in the direction of curvature.

Once again select the down-arrow on the Generate Mesh Button and then select Generate Surface Mesh Only to preview and note how much more detailed the mesh is on curved areas. Also note how many more cells are created and how we have added refinement in areas on the earth and other associated areas that we may not want refined.

Once again select Advanced to open the Advanced Mesh Generation Attributes window and select Refine Anisotropically to force the mesh to refine curvature along curved regions, but leave the flat areas at a rougher mesh. Once again select the down-arrow on the Generate Mesh Button and then select Generate Surface Mesh Only to preview and note how the mesh is detailed on the curved areas, but rougher on the flats which is a better balance for our problem. Select the down-arrow on the Generate Mesh Button and then select Continue Volume Mesh Generation.

The number of cells in a mesh heavily influence calculation time. However, the goal of mesh generation is not to obtain the smallest number of cells possible; it is to optimize the efficacy of the unstructured mesh by balancing the application needs of the user with the specifications of the radiation transport problem. The final window will look like the following:



The mesh is now ready to be loaded into the Project tree. We will select Import Mesh located in the upper right of the Mesh Generator window and import it. Once the mesh is created, it can be imported into other projects as well without having to generate it again.

9.4 Analog Model Creation and Execution

The Atilla4MC code is particularly effective for setting up and modeling shielding problems using MCNP. In the previous section, we discussed creating a shielding problem geometry in Soldworks and importing that model as a Parasolid into a new project in Attila4MC. Our imported geometry mesh for the tank example contains parts associated with each of the structures to include the air, soil, each solider, the source and the tank itself. Example 9.3 continues the example problem setup and execution. In the end, the final values for the effective dose in mrem per source particle.

The setup and execution of the problem requires the user to assign materials to the various regions associated with the parts in the Parasolid. In Example 9.3, the user builds materials using isotopic concentrations through the GUI. These materials are later assigned to the regions in the problem. In later exercises, we will import the material library from the Pacific Northwest Laboratory Compendium of materials. This will greatly speed up the model creation time as most materials needed are in that library.

After material creation, the user will assign MCNP controls to include the MODE and NPS controls. As stated previously, the user should have a good

understanding of MCNP to accurately use Attila4MC. The model will not give valid results if the MODE does not match the type of particles transported and tallied and the number of source particles greatly affects the overall variance in the final answer as well as the computational time for the problem.

The user continues to build the input file by specifying the source information. The initial example uses the cobalt 60 source provided from the Attila4MC tutorial example problem. In a later example, we will create a neutron source and change the problem from a gamma to neuron problem.

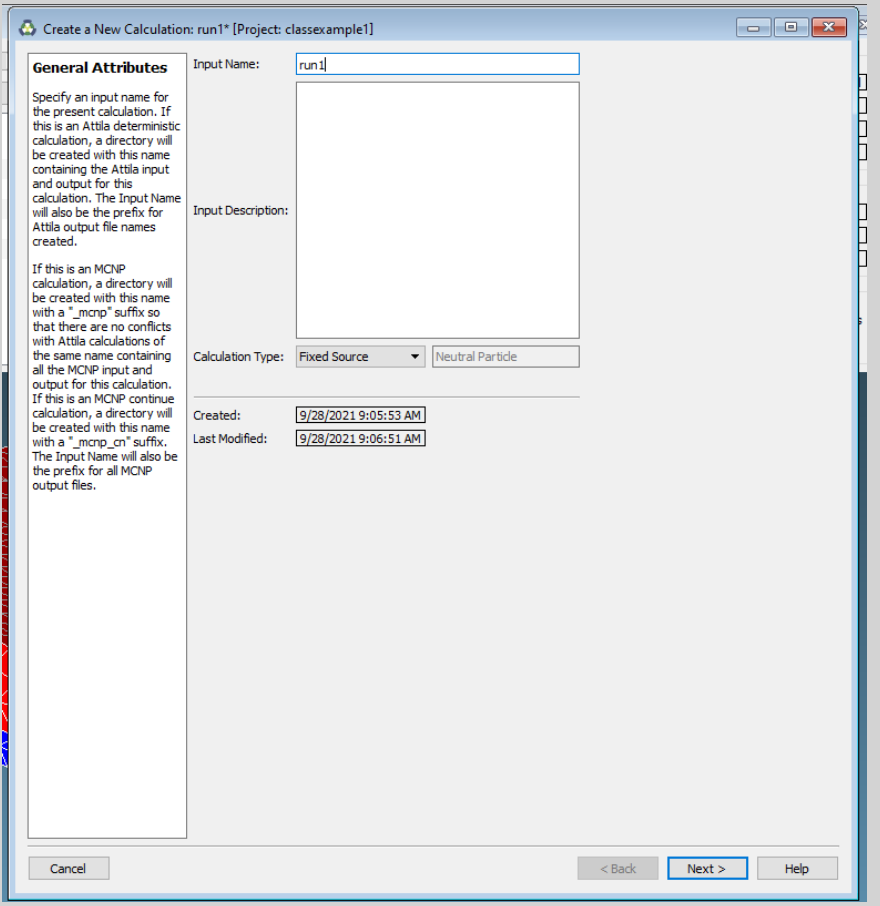
The final step in the problem setup is to create tallies. The Attila4MC program allows the user to apply tallies to individual regions as wells as creating a tally known as "elemental edits" which will create a tally across the entire problem mesh. This tally will allow the user to visualize dose changes as a surface contour plot. The tallies allow the user to include a tally multiplier as well as a response function. As explained in Section 5.6, MCNP has no sense of time, and all tallies are reported per source particle. To achieve actual values of flux, the user must include a tally multiplier of particles per second, otherwise the result for flux or associated dose equivalent will be normalized per source particle.

To obtain dose equivalent in Attila4MC, the user only needs to specify either a built-in or user-defined response function. We downloaded the ICRU57 response function from the example problem. Opening the response function in Textpad reveals that it modifies the flux tally from either gammas or neutrons based on energy and reports an output in mrem per source particle. When combined with an appropriate tally multiplier, the result will be in mrem per second or hour (depending on whether the FM is in particles per second or hour). The training materials available to the user on the Attila4MC website contains instructions on creating a response function, however that is beyond the scope of this text.

Execution of the problem requires the user to save the file and then select the appropriate execution method. Attila4MC has the option to run the analog MCNP file within the Attila GUI, run a CADIS variance reduction model, or package the file for the user to run in MCNP. For our analog problem, the GUI will write the input file, run the problem in a pop-up window, and write the output file along with providing appropriate TecPlot files that can be opened to visualize the results.

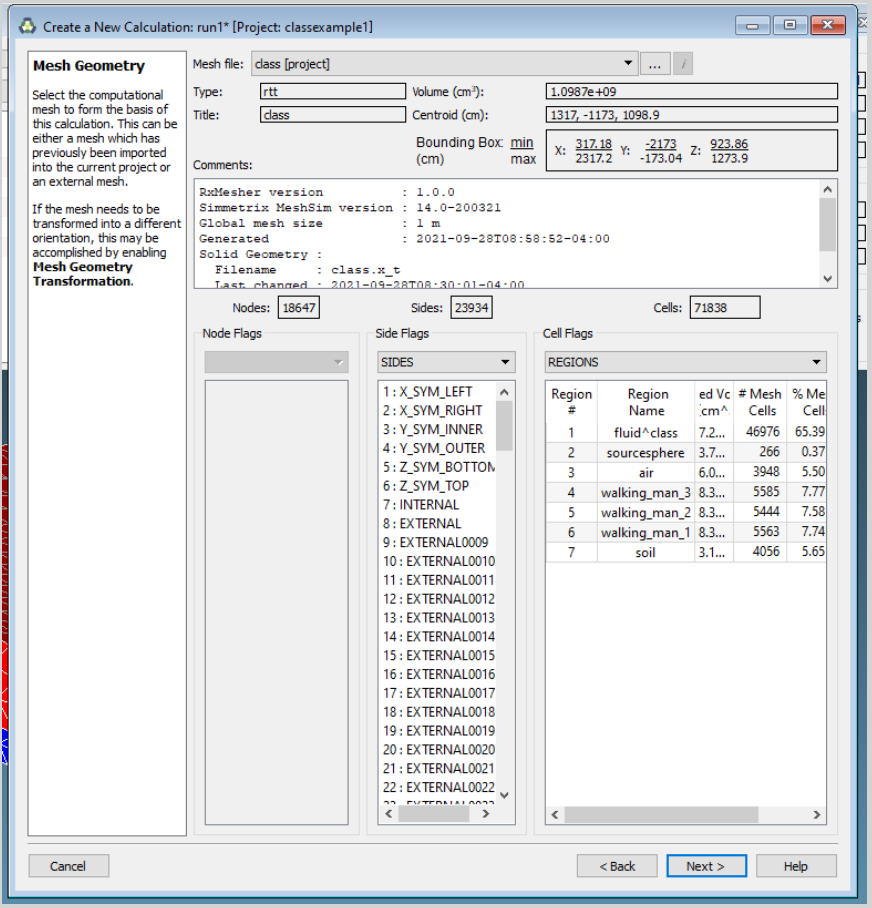
Example 9.3
Complete the example problem from Example 9.1 and 9.2 for a cobalt 60 source and read the results.

Solution: Right-click the project name on the project tree and select Create Object – Calculation to bring up the Calculation window and enter a name "run1" Select fixed source for calculation type.



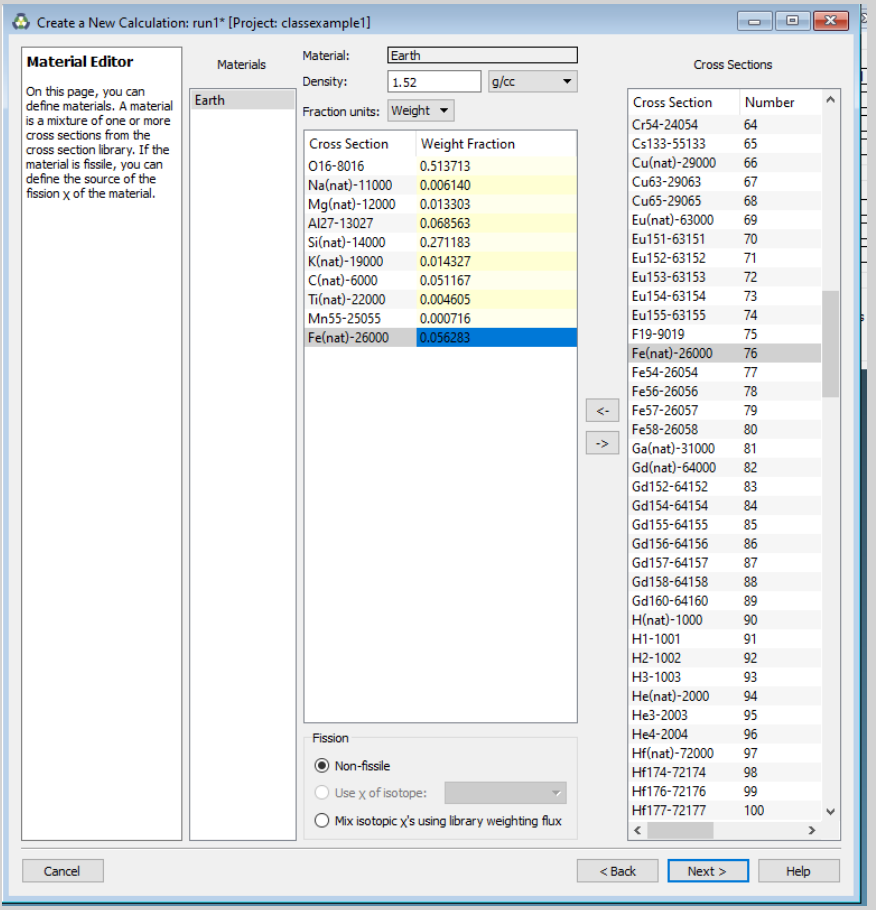
Select Next

Use the drop-down menu to select the mesh file saved from Example 9.2. Once loaded, information about the mesh will be visible. The Cell Flags section in the bottom right of the window provides the user with the % Mesh Cells associated with each (Attila) Region Name. This metric is a useful tool for performing a quality check on a mesh.



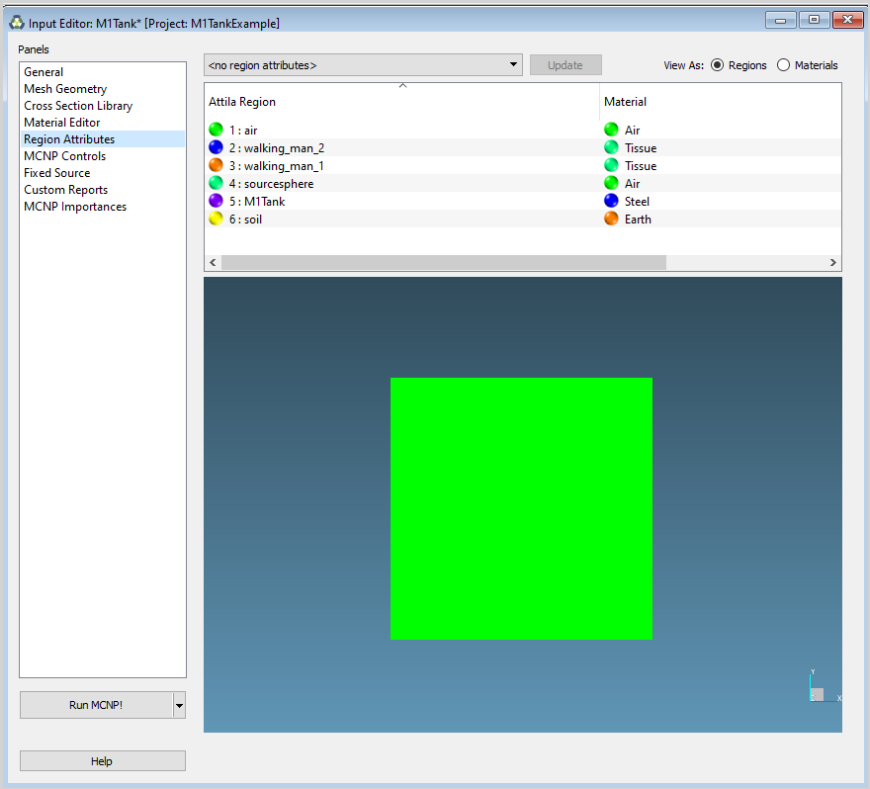
Select Next

Right-click in the Materials region and select a new material. Name the material "earth". Using the *PNLL Compendium of Materials* to find compositions. From the *Compendium*, go to Earth, U.S. Average. Enter the density in the appropriate units "g/cc". Next, select each isotope listed to add and then modify the weight percent or atom percent. Right-click to add next material, source (Co-59 for mat which is close to Co-60 for self-shielding) Repeat for air (Air (Dry, Near Sea Level) and tissue (Tissue, Soft (ICRU Four Component).



Select Next

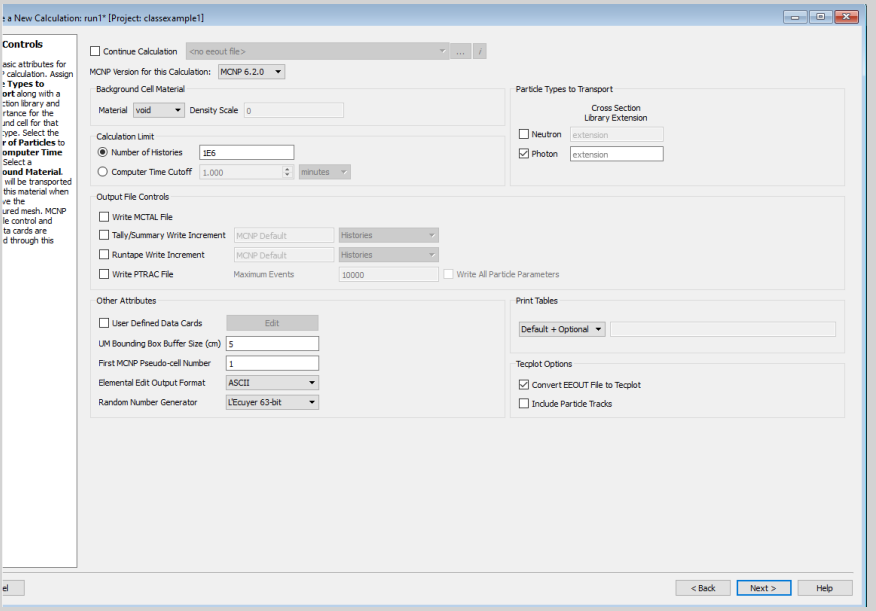
The next window will allow the user to select the region attributes. It displays a model of the problem in the lower window and the different parts are listed in a table at the top. In the top table, double-click each material column and use the drop-down menu to select the appropriate material to assign to that region.



Select Next

After setting up the materials and assigning them to regions, the next page will set up the MCNP controls. This process is the same as specifying the MODE and the NPS when writing the MCNP input file. The user inputs the number of histories to run (nps in the MCNP input deck), particle types to transport (mode in the MCNP input deck), and Tecplot options are all specified in the MCNP Controls panel. If the user wishes to add more inputs.

- Under Calculation Limit, change the Number of Histories to 1e6.
- Under Particle Types to Transport, uncheck Neutron and select Photon.

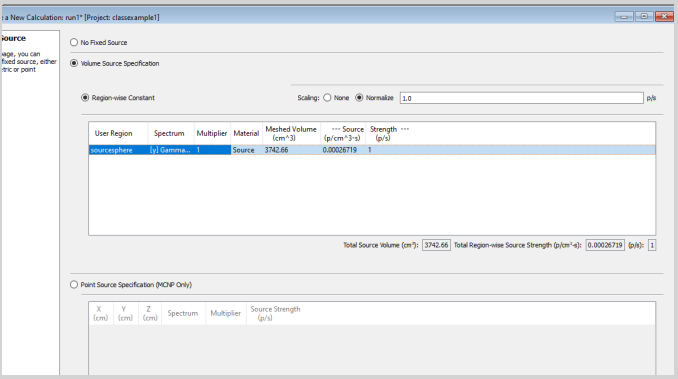


Select Next

The program proceeds to the source setup. The Fixed Source panel is used to create a source definition for the radiation transport problem. Options in the Atilla4MC GUI include an isotropic volume source, in which an Attila Region is assigned as a uniformly distributed volume source, or an isotropic point source. Selection of a volume source requires a scaling specification (none or an integer normalization in particles per source term). Both the volume and point source specification need to be assigned a source Spectrum. The point source location is based off of a 3D Cartesian coordinate system associated with the imported solid geometry. Because a volume source is desired for the radiation transport problem, a point source will not be created in this exercise.

- Select Volume Source Specification
- Select Region-wise Constant
- For Scaling options, select Normalize and enter a value of 1.0
- Right-click in the white region underneath Region-wise Constant and select Add Source Region
- Double-click on Fluid under User Region and select sourcesphere from the dropdown menu
- Double-click on <none> underneath Spectrum and select [γ] GammaSpectra: Co60 from the dropdown menu

A volume source normalized to 1.0 has now been created in the Attila Region Src using the Cobalt 60 gamma spectrum that was imported into the Project tree. The correct source definition can be seen below. Note: if you desired to add a mulitplier to convert tally results from per source particle to particles per second or per hour, you would change the Normalize value from 1.0 to the source strength on this page.



Select Next

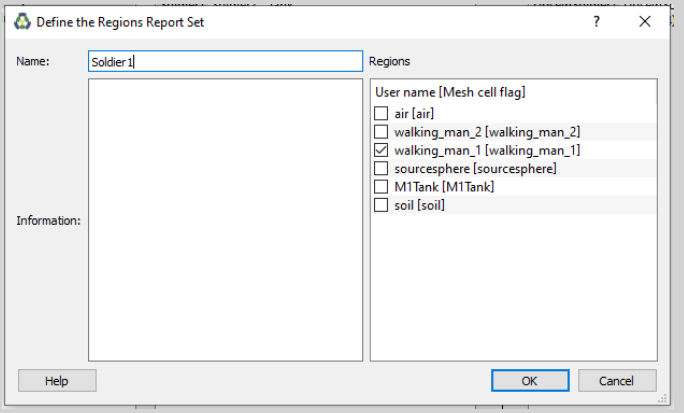
The GUI now pages to the tally page. On this page the user can select F4, F6, and F8 tallies, and the Elemental Edits tally. The Elemental Edits tally is required if the user wishes to visualize the solution using Tecplot. A point or ring detector (MCNP F5 tally) is also an option; however, this tally has not been adequately tested and verified within MCNP, and it is recommended to not be used until this has occurred. Before a MCNP tally can be created, a Spatial Set, the volume in which the user wishes the tally to occur, must be designated.

- Select User Defined
- Right-click in the white region under Spatial and Energy Sets and select Add Region Sets

The Define the Regions Report Set window will appear, showing all Attila Regions that had been defined earlier.

- For Name, enter Soldier 1
- Select Solider 1 [walking man 1] in the Regions table

Repeat for each soldier and the tank



Select Next

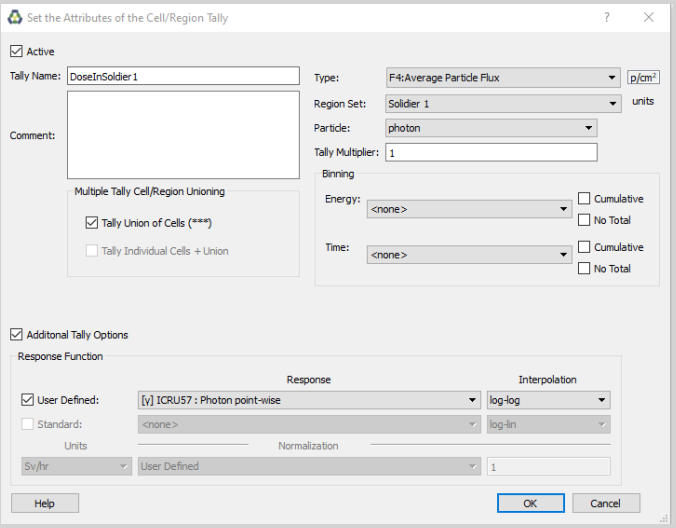
The Operator Spatial Set now populates the Spatial and Energy Sets group. Now that a Spatial Set exists, a tally can be created.

- Right-click in the white region under MCNP Tallies and select Add Cell/Region Tally (F4, F6, F7, F8)

The Set the Attributes of the Cell/Region Tally window will appear. The options in this window allow the user to customize the output of the tally being created. For example, the Type of tally, the Region Set in which the tally will occur, the particle type, and Response Function (dose conversion factor) are all chosen here.

- For the Tally Name, enter DoseInSoldier1
- Click on <none> in the Type section,
- Select F4: Flux → F4: Average Particle Flux
- Click on <none> in the Region Set section, select Soldier 1 from the drop-down menu
- Click on <none> in the Particle section, select Photon from the drop-down-menu
- Click the box next to Additional Tally Options
- Click the box next to User Defined and select [γ]ICRU57 : Photon point-wise

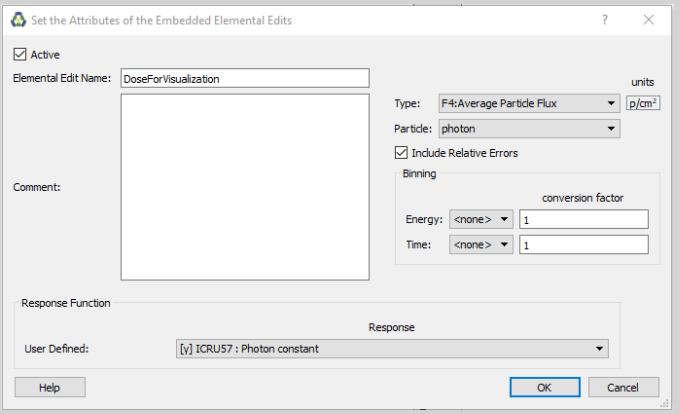
Select OK and repeat for each soldier and the tank



The Elemental Edits tally does not require a spatial set, as the tally is taken over the entire unstructured mesh.

- Right-click in the white region under MCNP Tallies and select Add Elemental Edits
- For the Tally Name, enter DoseForVisualization
- Click on <none > in the Type section, select F4: Flux → F4: Average Particle Flux
- Click on <none > in the Particle section, select Photon from the down-menu
- Click the box next to Include Relative Errors
- Select [γ] ICRU57 : Photon constant in the Response Function section
- Select OK

Elemental edits tallies require a multigroup response function, and only multigroup response functions will be shown in the pull-down menu. Again, this response function must already be created or imported into the Project tree. Creation of this tally will enable the dose solution, and corresponding uncertainties, to be displayed in Tecplot. It should be noted that the response function is only appropriate for an Attila Region that has been assigned the material Air. The Spatial and Energy Sets should contain Soldier1, 2 and 3, and the MCNP Tallies should contain DoseInSoldier1, 2 and the tank and DoseForVisualization.



The user can manually assign importances to specific Attila Regions, or elect to have Attila automatically assign importances to a collection of Regions to approximately maintain a constant number of particles through those Regions.

If the Enable Importances for MCNP remains unselected, all Regions will be assigned an importance of 1.

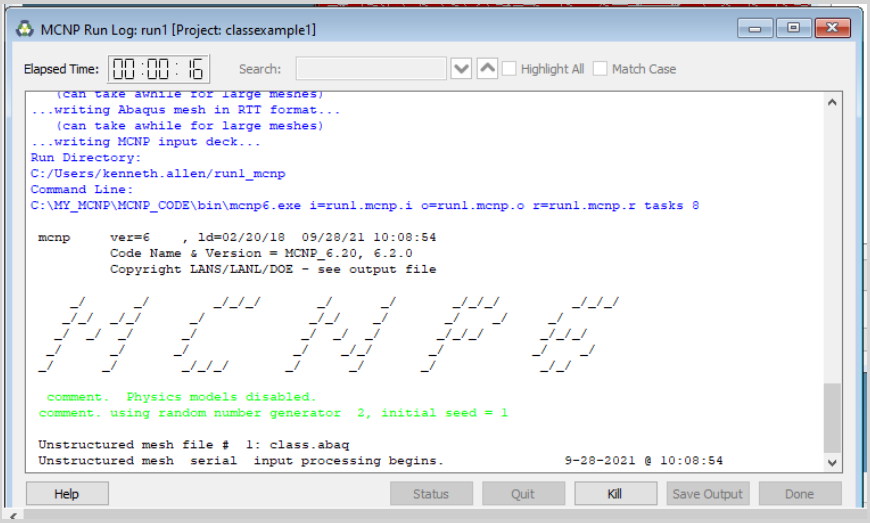
- Select Finish

Run1 can now be seen in the Project tree, and the calculation is ready to run. To run the calculation:

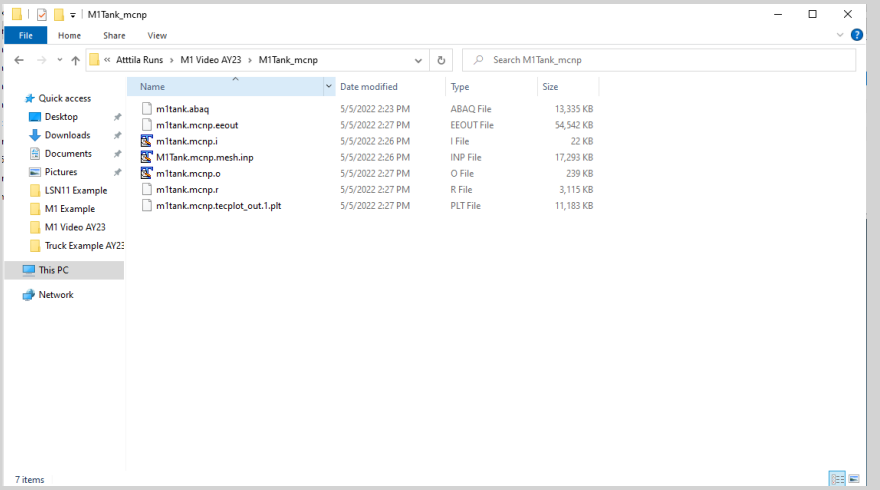
- Right-click on Run1 in the Project tree and
- Select Edit
- Select Run MCNP in the bottom left corner of the window

The MCNP Run Log will appear. Once the run has completed, the user will need to import the Tecplot file generated by the calculation into the Project tree using the same technique presented earlier in this exercise. The file is located inside of the Run1_mcnp file folder.

A calculation can also be ran using the Pack for MCNP option, which punches an MCNP input deck that can be ran on any machine that has MCNP6.2 installed.



Attila creates a file folder in the working directory where you stored the Parasolid and other files. Within this folder, you will find the several files that include the input, output and files used by Tecplot to read the tally results and mesh.



We can open the output file which is designated with a $*$.o extension "m1tank.mcnp.o" and find the tally results for the dose in the soldiers and the tank. If we look at the Dose in Solider 1, we will see the results are 1.976E-13 with a variance of 0.7884 mrem per source particle.

```
ltally        4        nps =     1000000
+                                   tally "DoseInSoldier1"
           tally type 4    track length estimate of particle flux.
           particle(s): photons
           this tally is modified by dose function DE4 and DF4.

           volumes
                   cell:       4
                         8.42938E+04

 cell  4
                  1.97606E-13 0.7884


 ============================================================================
```

9.5 Creating a Source and Reusing a Model

The Attila4MC environment is well-suited to modify and reuse problems for a variety of purposes. The modular nature of the problem package allows the

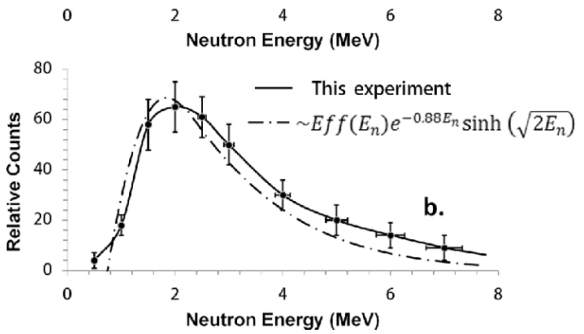


Figure 9.7: The Cf-252 fission-neutron energy spectrum deduced from the measured n-ToF spectrum: (a) measured spectrum compared with the shape obtained using a Maxwell-Boltzmann-type fission neutron spectrum (Equation) for Cf-252 and (b) same as (a) but for a watt spectral function. (Becchetti Jr; 2013)

user to use an established working problem and modify it with a new geometry, mesh, materials, source, or tally. Depending on the modifications made, the user may be required to re-mesh the geometry, assign the materials to regions, or change other parameters within the input file to complete the problem. In Example 9.3, the problem was a gamma shielding exercise. If, for example, we wanted to reassess the dose to the soldiers with the same geometry for a neutron source, we can modify the input file from a gamma problem with a cobalt source and photon tallies to a neutron problem without having to generate a new problem from scratch.

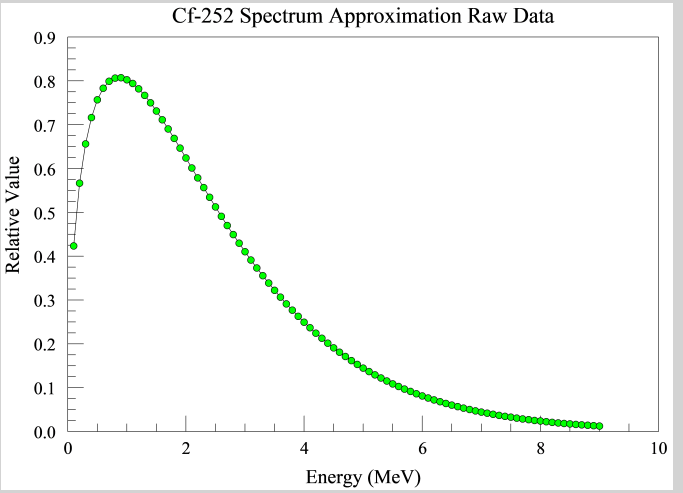
The gamma source used in Example 9.3 was downloaded from the example problem posted to the Attila user's website. In Example 9.4, we will generate a neutron source using a Californium spectrum. The Californium-252 source which has a spontaneous fission rate like a Watt Spectrum. If we were modeling in MCNP, we could specify a Watt Spectrum for our ERG distribution and apply the appropriate coefficients from Appendix D in the MCNP manual for a Californium source.

For Example 9.4, we are going to create a distribution to model the probability to gain practice with creating the source. The Cf-252 spectrum is well-approximated by the equation presented in numerous references to include the one presented in Figure 9.7.

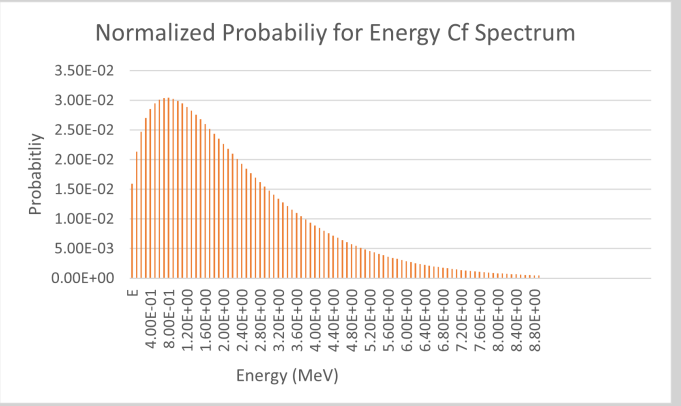
Example 9.4
Create a Cf-252 fission-neutron source for Attila4MC using the equation presented in Figure 9.7, modify the problem for a neutron source and run the

problem.

Solution: Begin by using a spreadsheet to create several energy bins from 0.1 - 9 MeV. Use the equation from Figure 9.7 to generate the raw (unnormalized) values across the energy spectrum.



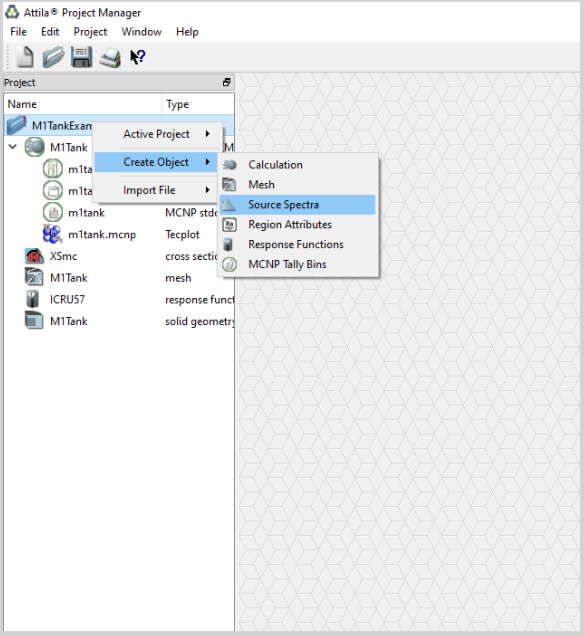
The values need to be normalized to get a cumulative probability of 1. MCNP will normalize a distribution set so the user can simply include the raw data, however we can divide each of the raw energy values by the sum of the set in the spreadsheet to get a normalized distribution.



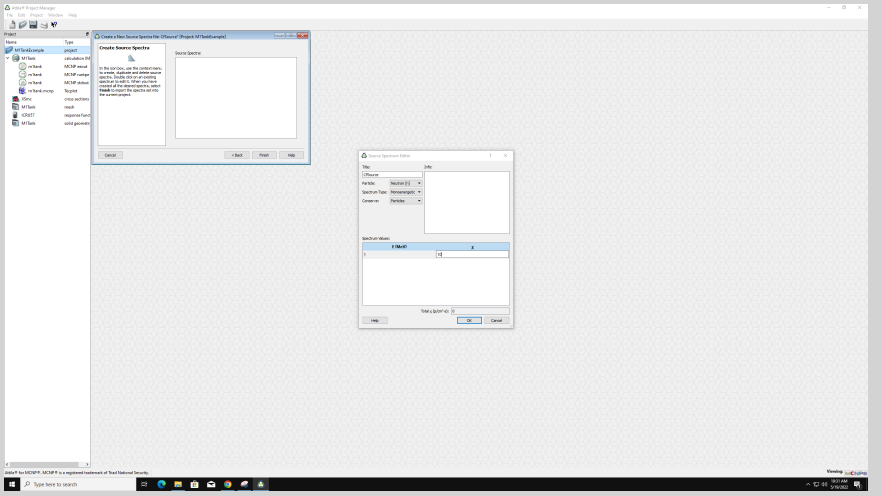
If we plot the normalized distribution as a bar chart, this represents the line distribution we would use in an MCNP SDEF similar to the point

isotropic source with various discrete energies example on page 122. We could write an MCNP SDEF with the SI equal to the energies and the SP equal to the associated normalized probabilities from our normalized distribution. However, we will use Attila to create the source file for use in our problem.

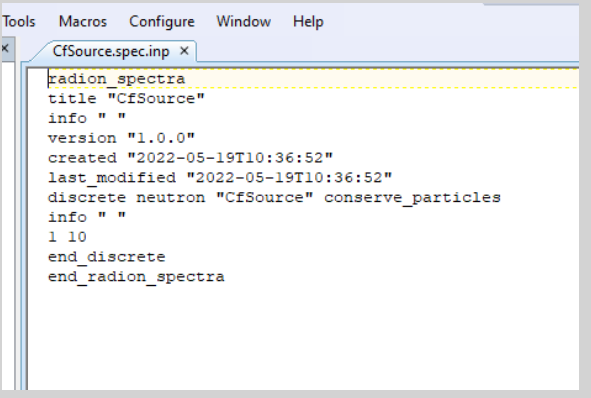
We can open our gamma source program from Example 9.3 and modify it with the new source data we generated in the spreadsheet. Because we will be using the same geometry, we can modify our gamma problem with a neutron source and change the MCNP controls and the tallies to neutrons. Note that this will overwrite our gamma problem files so we should copy them to another directory if we need them. To change our source, we will first right-click on the gamma source in our tree and remove it from the project. To create a new source spectra, right-click on the project name and select *Create Object* and *Source Spectra*.

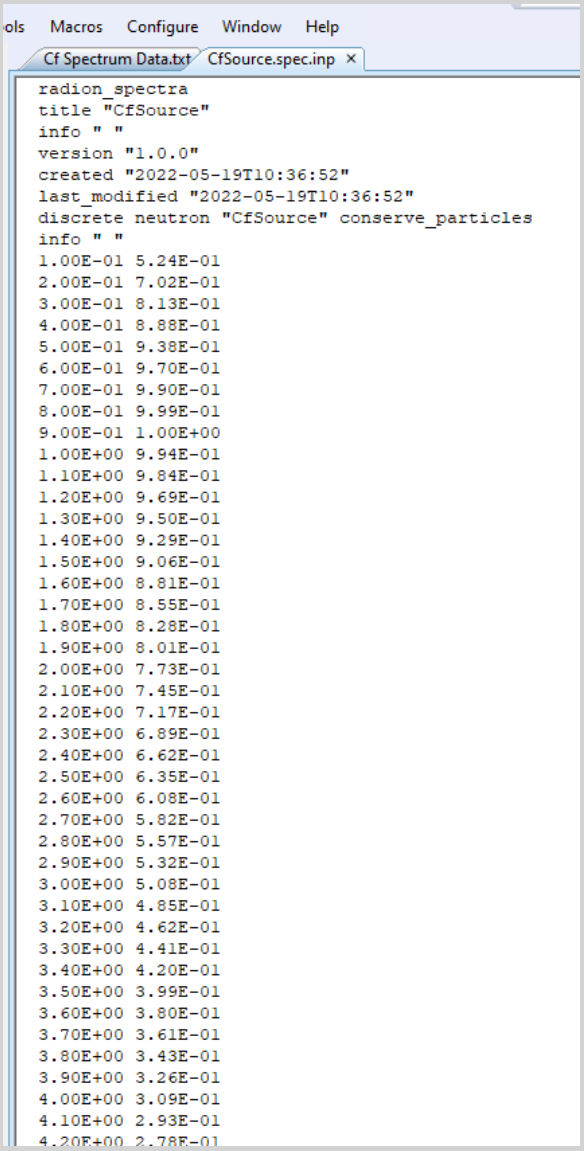


When the wizard opens, we will give the spectrum a name "CfSource" and select Next then right-click in the "Source Spectra" box and select *Add Source Spectrum*. In the window that pops up add a title, choose neutron for particles and change spectrum from *Constant* to *Monoenergetic*. Right-click in the area for *Spectrum Values* and select *Add*. Enter in one value for energy and probability just as a place-holder. This does not have to be a real energy or probability from our distribution and select *OK* and Finish.

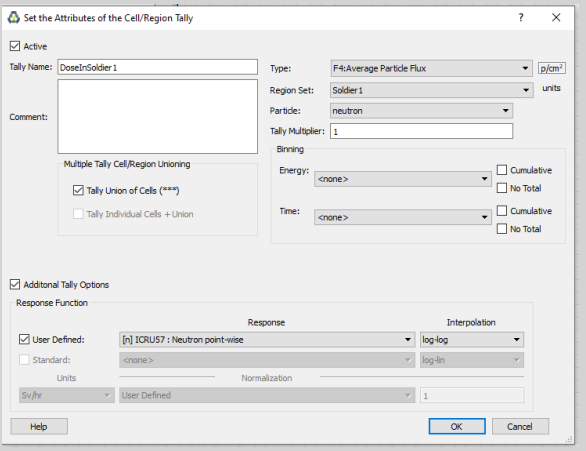


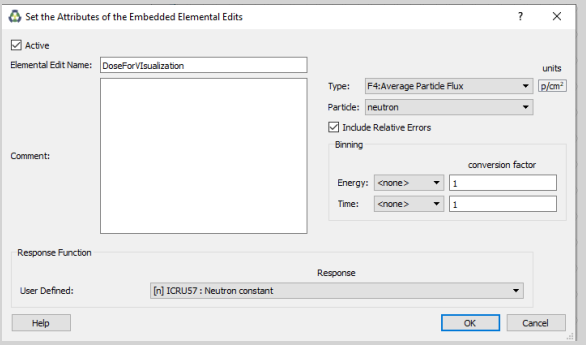
The source spectra we created will now show up in the project tree for our problem. We will right-click on that item in the tree and select Export and place the file in the location where with the rest of the files for our new problem. Open the file using TextPad and you will see the two place holder numbers for the energy and the probability. Replace the place holders with the all the energy and normalized probability values and save the file. Now go back to Attila and right-click on tree and import the Source spectra file we just put in and replace the current file in the problem tree. We can right-click on the file in the tree and select edit to view and make sure the entire spectrum is present.





Once the source is changed to a neutron, we need to edit the Calculation. Right-click on the calculation in the project tree and select edit. Choose MCNP Controls and change the particle type to Neutrons. Go to the Fixed Source window and double-click on the Spectrum and select the CfSource. The final step involves changing the tallies and elemental edits. Go to the Custom Reports and change each of the MCNP tallies and elemental edits to neutrons with a response function of IRCU57 Neutron.





Save the project and then select "Run MCNP" to execute as a neutron source problem.

Just as it is possible to reuse a problem with a different source, a problem can be reused with a different geometry. For example, we could modify our Parasolid in Example 9.3 by adding a mound of dirt in front of the tank and shielding the person out in the open as shown in Figure 9.8.

To do so, we would first create a part in Solidworks for the berm. We would then open the assembly we created for our tank model and add the berm and save as a new Parasolid. Once the new Parasolid is created, we can modify our model in a similar way we did when we changed the source. In this case, because the geometry is different, we will copy the project file, source spectra, and the cross-section files into a new subdirectory. Once complete, copy the new Parasolid into the same subdirectory. Open the project and modify it by importing the new Parasolid, generating the mesh, and importing it into

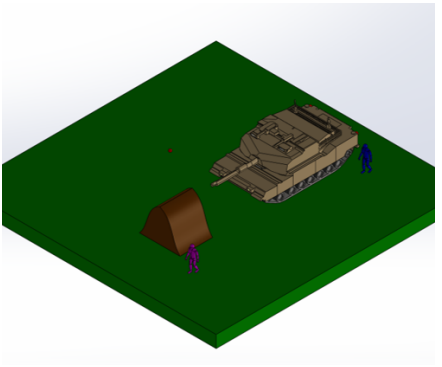


Figure 9.8: Solid model from Example 9.3 modified with berm.

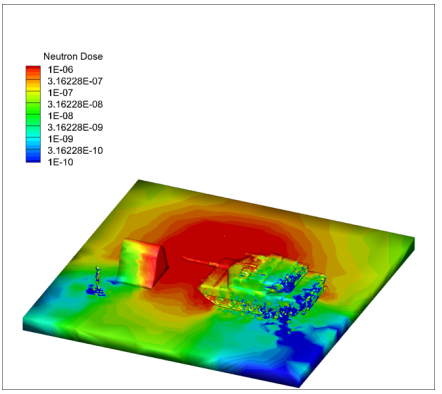


Figure 9.9: Dose results from Example 9.3 modified with berm. Note how the shielding from the berm protects the solider much like the tank.

our model. We then can edit the calculation to include associating the new mesh geometry, assigning the region attributes, and modifying any associated regions for the source and tallies as needed. By updating models, we can build a material library and streamline our calculation setup process. Caution - the associated cross-sections used to create those materials must be copied and remain the same to reuse materials in subsequent calculations.

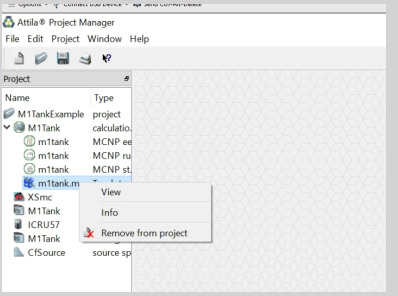
9.6 Tecplot Visulization of Results

We have created three problems in Attila so far. The first was the tank in open with gamma source. Next was the tank in open with neutron source. Finally, we had the tank in open with neutron source and earth berm. Figure 9.9 is an example of the Tecplot of our elemental edits results for the final problem.

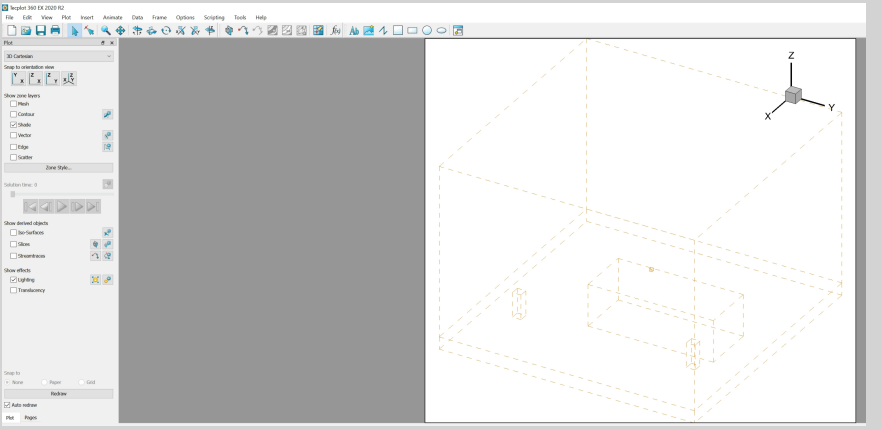
We can use Tecplot to visualize the dose across the problem and the variance associated with the dose tally. Example 9.5 details the steps used to get a contour surface from your problem much like the one presented in Figure 9.9. Later advanced techniques will be described in Section 9.8.

Example 9.5
Create a meaningful surface contour plot of the neutron dose from the problem in Example 9.3 with the neutron source from Example 9.4.

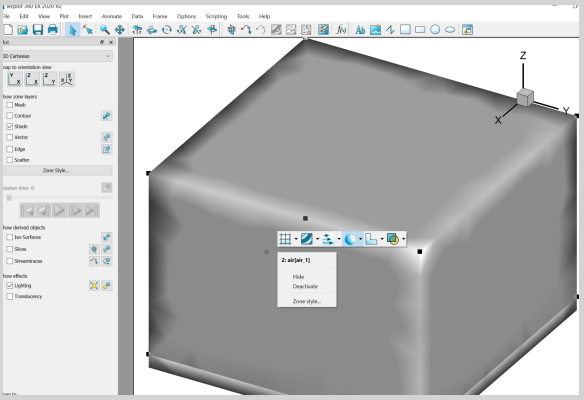
Solution: Begin by launching Attila and open the M1 Tank example with the neutron source without the berm. Expand the tree for the calculation and right-click on the Tecplot item and select View. If, the Tecplot did not load into the problem when MCNP concluded, you can right-click on the project and select import, Tecplot file, and the .plt file should be visible in the folder where your other files are located.

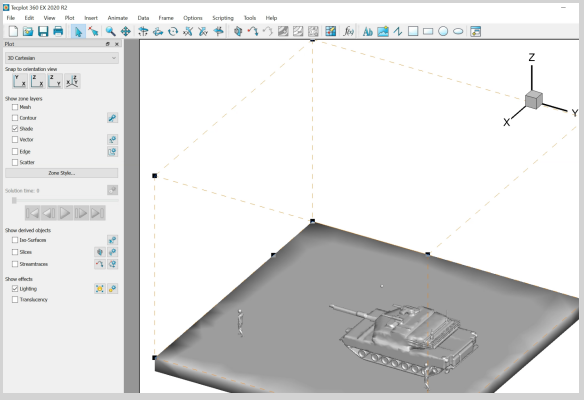


Tecplot will open in a new window with dashed lines for bounding boxes around parts in the problem.

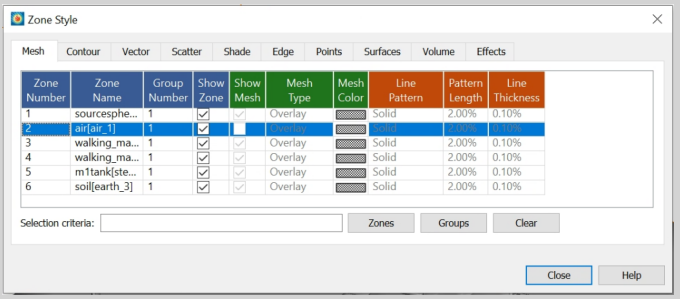


First, we will get a shaded plot of the objects by unchecking and rechecking the **Shade** box and the left When prompted to turn on active zones, select **yes**. You will have a large square showing the outside of the model. Click on the middle of the plot and then right-click to get the menu and select hide to turn off the air.

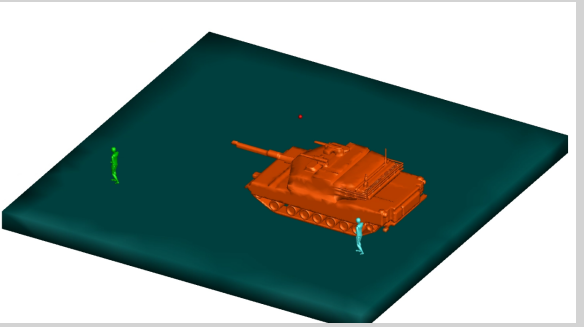




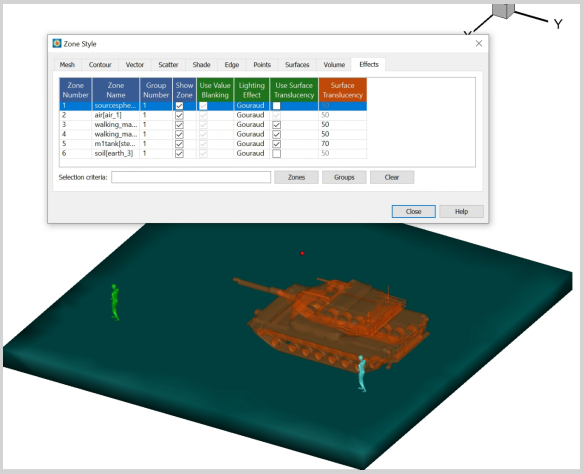
We can further refine how the model looks by selecting the **Zone Style** button.



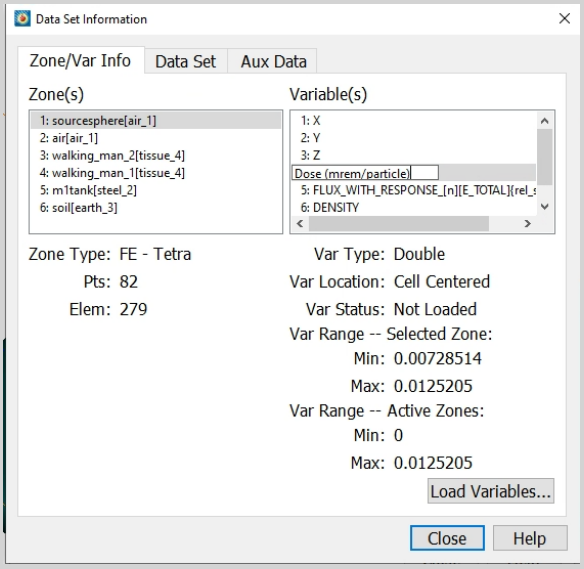
Go to shade tab and you will see that **Shade Show** is deselected for air which we did by hiding it. We could also hide the air here by turning off **Show Shade**. Do not turn off **Show Zone** for any region. Right-clicking on the **Shade color** for any of the zones will bring up a color pallet that will allow you to change the color of the object in those zones. We can also turn off the bounding box by selecting the **Options** menu at the top and deselect the **Show Bounding Boxes** for Enabled Volume Zones with No Style.



Now select the **Slices** option to the left and this will open the slices menu. Note that because the air does not fill the entire region between the outside boundary and the objects, the slice plane will not show a full plane across the model. Then select the button to the right of the **Slices** option with the gear on it to bring up the **Slice Details** menu. From here you can change the location of the slice and move through it. Note the title of the legend FLUX_WITH_REPSONSE_[N][E_TOTAL] which is not a good description. Select the **Data** menu option at the top and then **Data Set Information**. You can then double-click the name of the dataset under **Variable(s)** and change to something more meaningful.

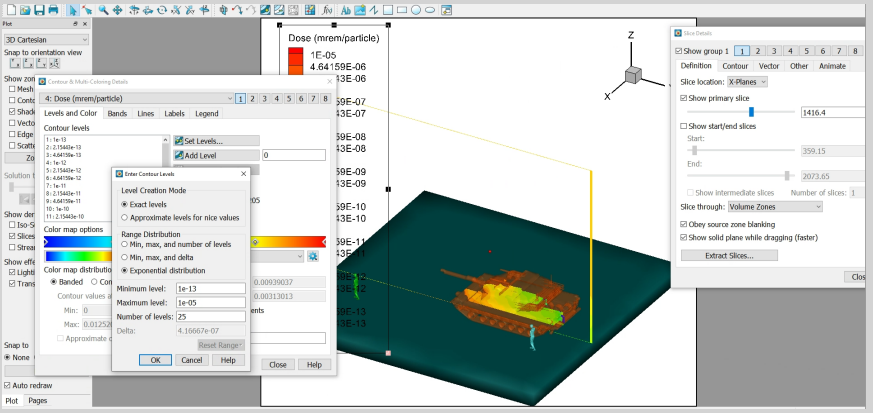


If we want to see the results inside an object, we can select **Translucency** under the Show Effects region to the left. If we only want to see inside some of the objects, we can open up **Zone Style** again and go to the Effects tab and select and deselect **Use Surface Translucency** as well selecting the amount of translucency (default is 50) Let's turn off translucency for the source and earth and change the tank to 70.

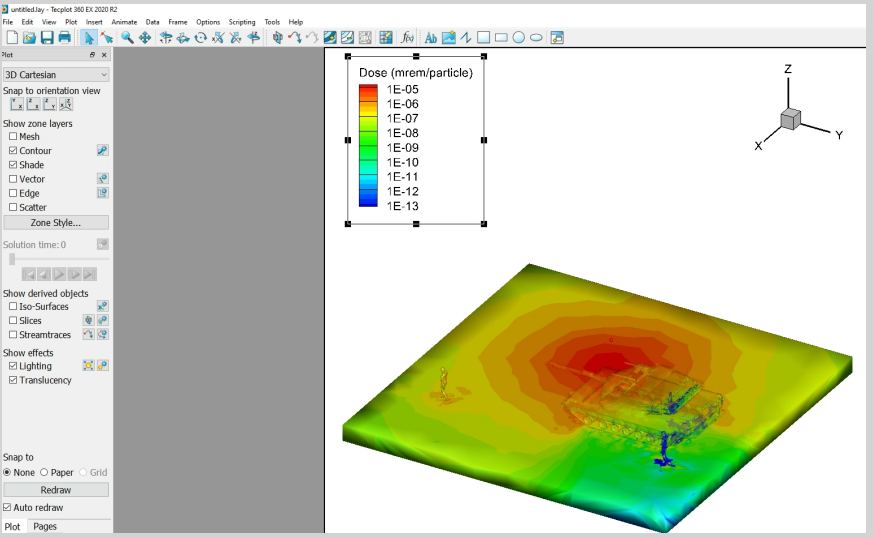


We will need to change the contour intervals to get a more meaningful representation of our data. Select the settings button next to the contour

option to the left above Shade Select the Set Levels option and change the Range Distribution to Exponential and set the minimum to 1E-13, maximum to 1E-05 and the number of levels to 25. This may take some time to get a good representation as you adjust the options.

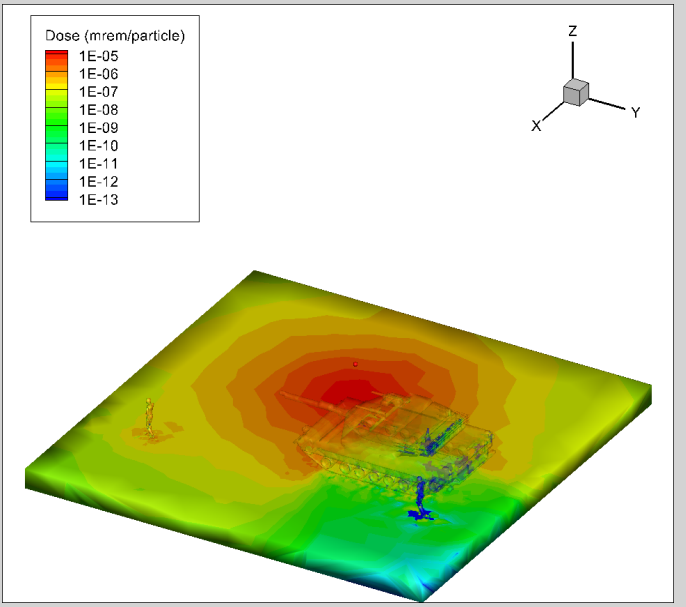


Right-click on the legend and select **Legend Details** and check the resize automatically option. We can turn off the **Shade** option and get just the slice contour through the model Finally, turn back on the **Shade**, turn off the **Slices**, and select the **Contour** option.



Once you have the orientation and plot like you want, you can then select

File and **Export** to save the plot as a good picture for use in your document.



9.7 Variance Reduction Using CADIS

We first examined variance reduction in MCNP in Chapter 6. Recall that shielding problems, by their very nature, cause poor statistics in Monte Carlo simulations. The standard error in Monte Carlo is inversely proportional to the square root of the number of particles striking the tally. For a thick shield, few particles will transport to the tally region resulting in a large standard error. To obtain better statistical results, we can either increase the number of source particles or we can employ variance reduction techniques or both. The cost of more source particles is always increased computational time. Therefore, adequate use of variance reduction techniques is often necessary to achieve statistically relevant results in a reasonable time.

For our original purposes, we limited our variance reduction techniques to changing the importance of particles throughout the depth of layers of our shield. This method can be incorporated in Attila4MC using the MCNP importance panel in the calculation, however there are more powerful and automated ways to achieve better statistics within Attila. The method we will incorporate for variance reduction in Attila will be the use of the built-in Consistent

Adjoint-Driven Implementation Sampling, or CADIS, deterministic code. The CADIS code is a deterministic transport solver. This is like the solver we used in SCALE to calculate particle transport. It consists of a discrete ordinates solver with ordered Legendre polynomials for the discretization of the angular quadrature.

The variance reduction methodology using CADIS creates Weight Windows within our MCNP model. Weight Windows can be applied on a rectangular mesh across the three-dimensional space in an MCNP model. Each voxel is given a weight which corresponds to the importance of the particle in that region of the problem. The user can specify regions of interest and the window will have higher importance in those areas. This is like when we specified higher importance for particles on the outside of the thick shield in Example 6.1. In this case, rather than the user specifically dividing the geometry of the shield into smaller regions and manually adjusting the IMP card, the entire body of the problem is broken into voxels and the IMP is adjusted from regions of low to high importance.

The Forward CADIS, or FW-CADIS, calculation is like a CADIS calculation, but it will work to calculate the weight windows isotopically away from the source and will give a region-wise reduction of variance away from the source. It does not require a motivating tally, although one is normally applied. The FW-CADIS is useful in reducing the variance across a region on the outside of a thick shield.

The use of CADIS to reduce the variance on a tally is presented in Example 9.6. For this example, the tank with berm problem is modified by removing the tank and a model is generated with and without the CADIS variance reduction applied. The results are apparent in both the final tally to the solider protected by the berm and the overall TecPlot of the variance across the model.

There are several considerations that must be considered when specifying the mesh grid size for the weight windows for a CADIS problem. These recommendations are based on the half-value layer (HVL) thickness for the radiation in the material. The HVL is defined as the thickness of material that will reduce the uncollided flux by one-half. These values are tabulated or they can be easily calculated using the method described in Chapter 4 and Equation 4.11. The tetrahedral mesh for a CADIS problem has a recommended edge length of 4-10 HVLs while the Weight Window grid is recommended to have a size of 2-8 HVLs. Figure 9.10 shows the approximate HVLs for typical material an source combinations to obtain optimal mesh and weight windows generation.

The use of CADIS to reduce the variance on a tally is presented in Example 9.6. For this example, the tank with berm problem is modified by removing the tank and a model is generated with and without the CADIS variance re-

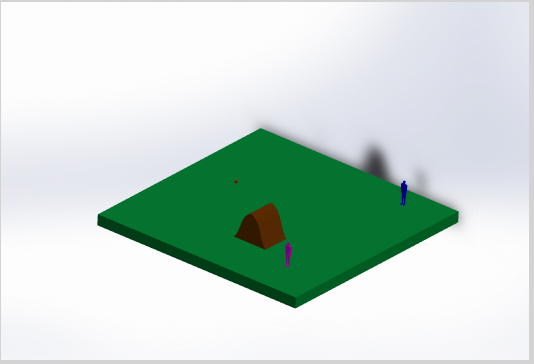
	Approximate HVLs			
Energy (MeV) / Source	Lead	Iron	Concrete	Water
0.5 (γ)	0.5	1.0	3.5	7.5
1.0 (γ)	1.0	1.5	4.5	10.0
1.5 (γ)	1.5	2.0	6.0	12.0
2.0 (γ)	1.5	2.0	6.5	14.0
Ir-192 (γ)	0.5	1.0	4.5	14.5
Cs-137 (Ba137m) (γ)	0.5	1.5	5.0	17.0
Co-60 (γ)	1.0	2.5	6.0	20.0
14.1 (n)	1.0	11.5	11.0	20.0
Cf-252 (n) / U-235 (n)	1.0	9.0	5.0	11.5

Figure 9.10: Approximate HVLs for typical material and source combinations to support optimal mesh and weight window generation. (SilverFir; 2023)

duction applied. The results are apparent in both the final tally to the solider protected by the berm and the overall TecPlot of the variance across the model.

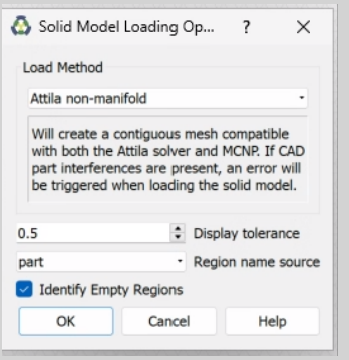
Example 9.6
Modify the geometry of the problem in Figure 9.8 by removing the tank and set up and run two shielding calculations, one analog MCNP and one CADIS. Compare the results for the variance on the soldier in the open and the solider behind the berm for both cases and plot the standard error as a countour map in TecPlot.

Solution: Modify the tank and berm example assembly by removing the tank. This will greatly simplify the problem and allow for easier calculations for demonstration purposes. Save the assembly as a Parasolid.

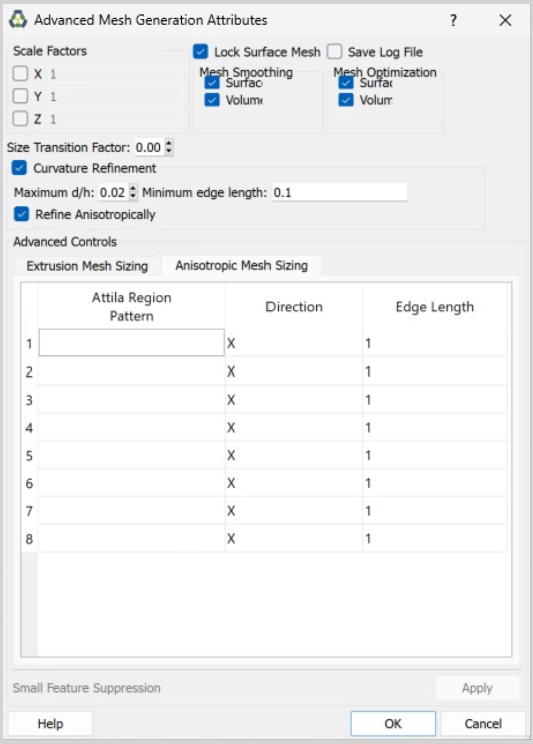


Download the Cask in Room Zip problem from the Silverfir tutorial location. Save and unzip the files. This will allow us to make use of the problem to include the cross sections required for CADIS and the entire PNNL materials. Copy the Parasolid for the model without the berm into the folder and open the Attila project. You will see two calculations that include the PNNL databases. One is isotopic and one is exact. The folder also includes a response function and a cobalt source. Import the Parasolid we created with the berm into the project. You can remove the cask in room solid model if you wish.

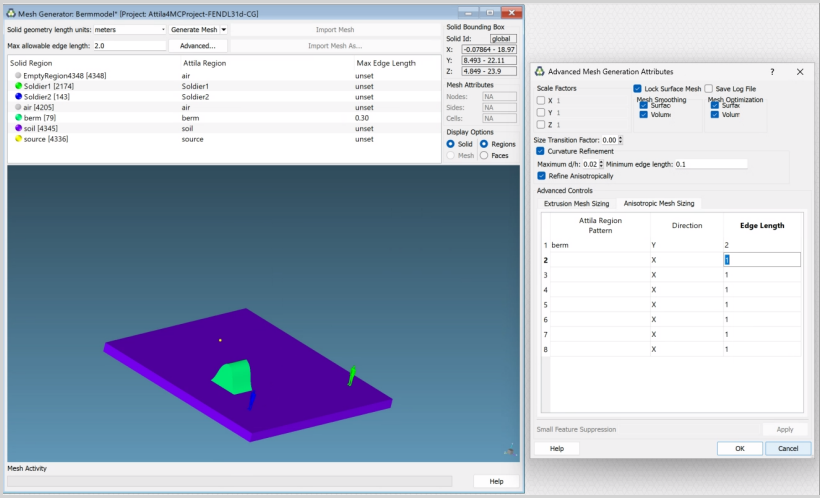
Right-click on the solid body imported and select **Generate Mesh**. CADIS needs a contiguous mesh so we will select Attila non-manifold and keep identify empty regions selected. This will provide CAD regions for any areas not defined in our model. For example, the air that is inside the box region on the outside down to the surface of the soldiers, soil and berm, will be filled and we can specify the material as air. We will change the region name source from assembly to part as our assembly was constructed from parts.



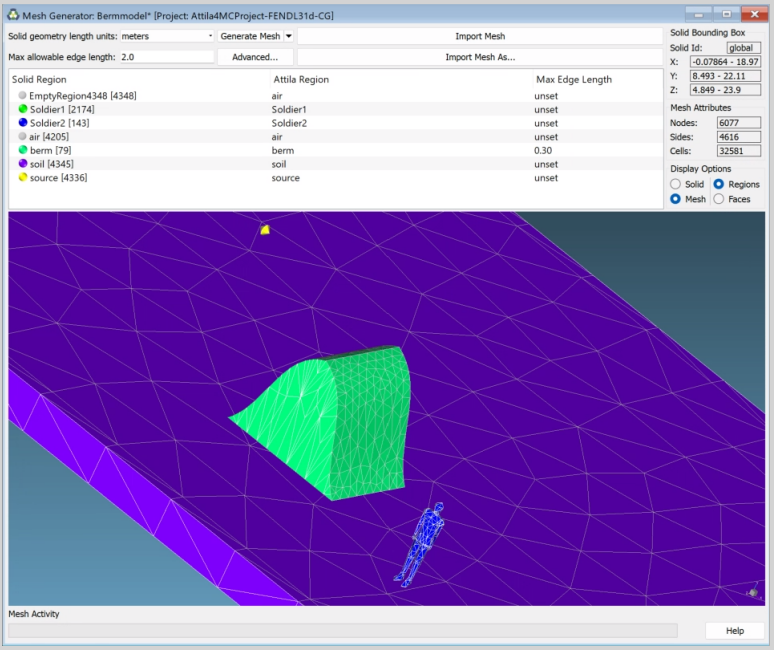
In the mesh generator, select max mesh size – 2 m as this is still 1/10 to 1/20 the maximum dimension of the problem. Note that there is an Empty Region created. This is the air that filled in the area between the outside air box we defined in Solidworks down to the surface of the soil not including the berm and the soldiers. We can view this region by hiding the outside air. We will double-click under the Attila Region column and rename it "air" to match the other air region. Under the advanced option, select **Curvature Refinement** and **Refine Anisotropically**. Set to 0.2 for 10 cm and set size transition factor to zero.



Look at problem and examine where we want to resolve the regions. The soil is inconsequential, however the berm is important. The problem requires resolution through any thick shields. Referencing the weight window grid size (FIGURE 9.10), the grid needs to be smaller than the tetrahedral mesh. Using the approximate settings in Figure 9.10 for cobalt gammas in concrete, which is close to soil, the result is 2-9 HVLs. We can approximate our soil berm as concrete. Set the mesh size for the berm to 30 cm for Co-60. We can change the refinement in the vertical direction to make a courser region. Leave to max edge length to match the model in the vertical dimension. For this example, the model was loaded with the y-axis as vertical. Return to the advanced features and set the Attila Region Pattern to "berm" and change the direction to Y and set the Edge Length to 2.

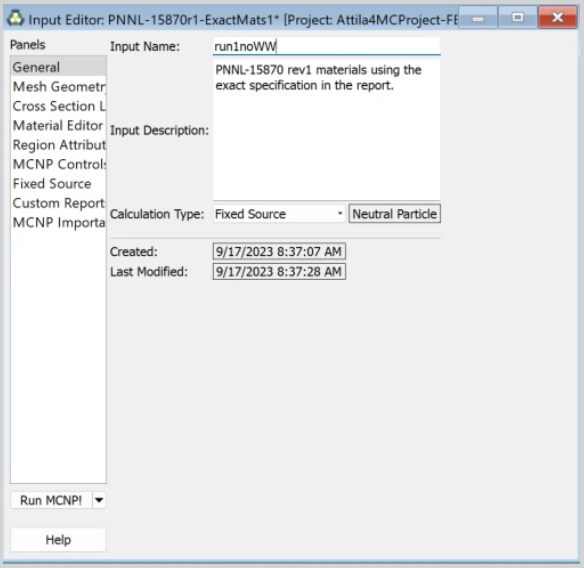


Generate the mesh. Resolution on air and outside should be 2 m and the berm should have a narrower edge length in the horizontal plane.

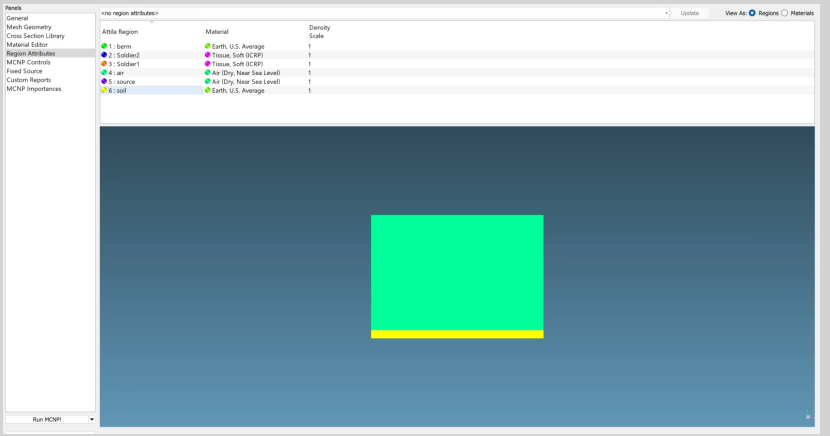


Import mesh into project and save the project. The next step is to set up analog calculation. Close mesh generation window as it is no longer

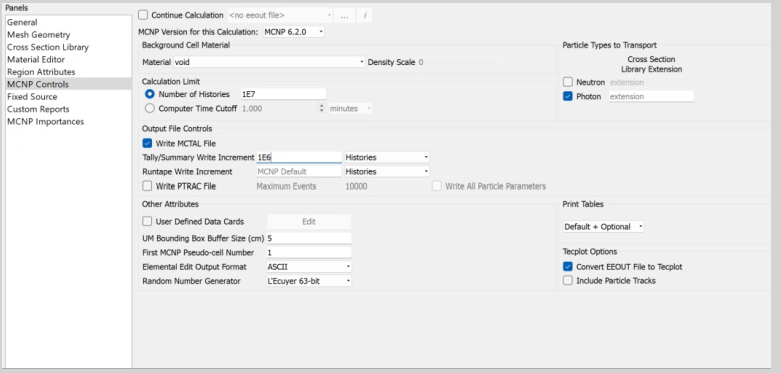
needed. Right-click on the exact calculation in the tree and select **duplicate**. This will copy the calculation and allow us to rename and use it. Edit the duplicate copy and give it a name for "run1noWW". Select the calculation type as either Fixed Source, CADIS or FW CADIS. For the first problem, we will do an analog MCNP problem and use the Fixed Source. This is the same as the problem performed in Example 9.3 with a different solid body. It will set the base parameters for variance with no reduction techniques applied.



Go through the panels and open the "Mesh Geometry" to select mesh and "Region Attributes" to assign materials. Note that all the air is one now because we renamed the empty region “air” when we generated the mesh. For this problem, we will set the source as air as well.



In the "MCNP Controls" panel, change to a Photon problem, Set 1E7 particles. Write summary tally increment as 1E6.



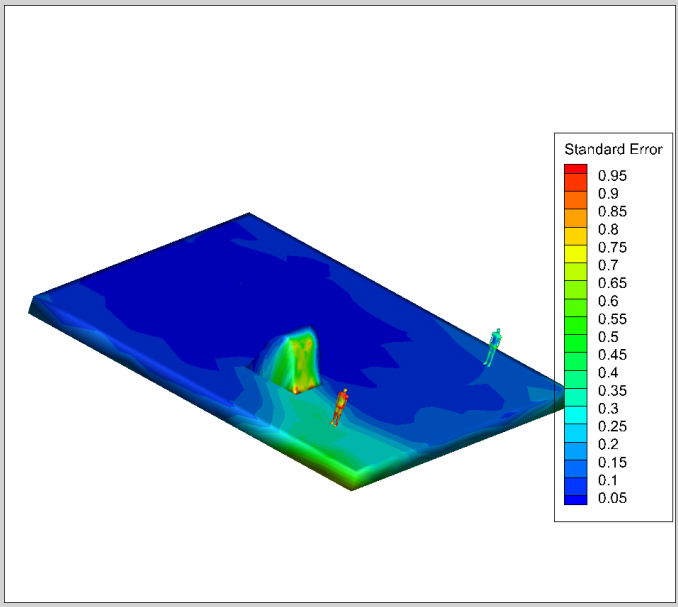
Set the source and spectrum and the tallies on both people and one for elemental edits as done in Example 9.3. Save and run the MCNP problem. View the output file to see the tally results and variance. Note that the "Dose in Soldier 1" results pass the 10 statistical checks and the results are:

```
1tally        4        nps =     10000000
+                                    tally "Dose in Soldier 1"
          tally type 4    track length estimate of particle flux.
          particle(s): photons
          this tally is modified by dose function DE4 and DF4.

          volumes
                   cell:       4
                         7.97977E+04

 cell  4
                  2.34612E-14 0.0212
```

The results in Solder 2 are presented below and it failed at least one statistical check:

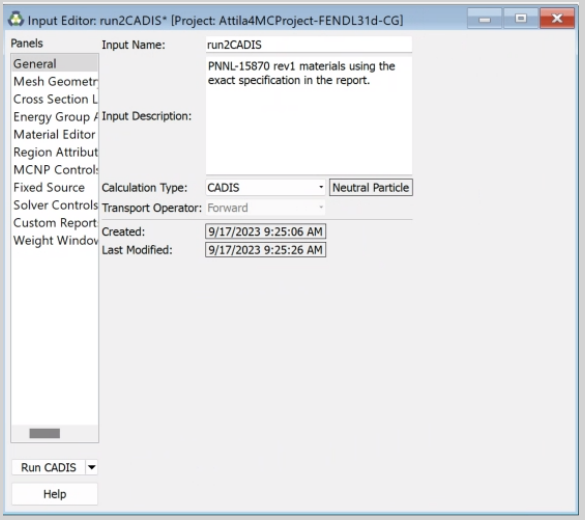
```
1tally       14        nps =     10000000
+                                  tally "Dose in Soldier 2"
            tally type 4    track length estimate of particle flux.
            particle(s): photons
            this tally is modified by dose function DE14 and DF14.

            volumes
                    cell:        3
                          7.97977E+04

 cell  3
                   7.23890E-16 0.0844
```

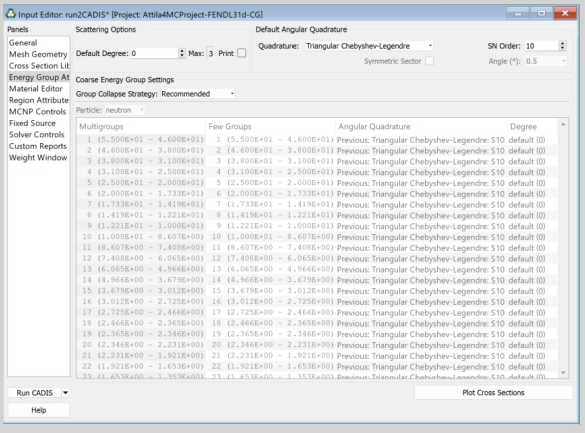
This exactly confirms what we expected to see. The dose for the soldier 2, behind the berm, is much less than the soldier in the open and the uncertainty is much higher for this tally because fewer particles transmit through the shield. Plot the variance from our TecPlot file and see that the areas of higher uncertainty exist behind the berm.

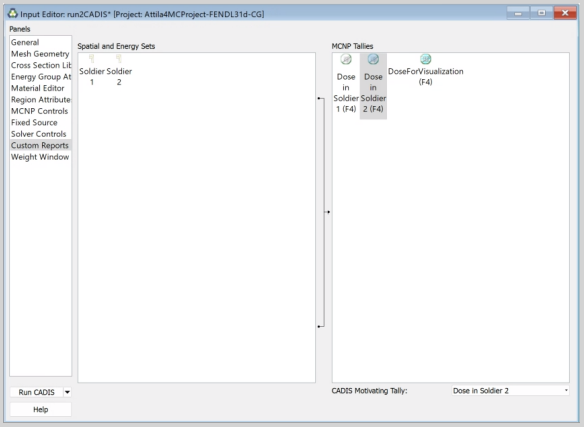


To setup and run CADIS problem, first right-click on run1 and duplicate and edit it. Rename it to run2-CADIS. Change calculation type to CADIS. This will provide a couple of new panels in the program. Go to the "Energy Group Attributes" panel and select the recommended group collapse. Attila will try to balance the run time vs. the FOM in MCNP. Set the degree of 0 for scattering options. This will prevent negative flux results. This is the Legendre order. Default angular quad is the Sn. If the problem has significant scattering the Sn order should be higher. For this problem, 10 is sufficient. In the "Custom Reports" panel, the problem requires a specification of a CADIS motivating tally. This requires a valid tally. For our problem, we have tallies on Soldier 1 and Soldier 2. We want the motivating tally to be Soldier 2 becuse it is the

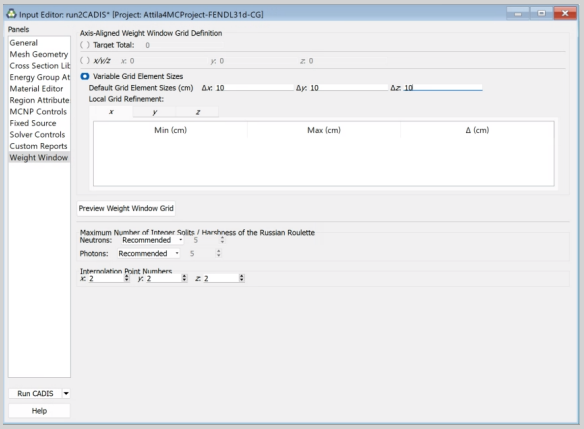
one behind the thick shield. The drop down will not initially show either tally as an option. When there is a valid tally but it cannot be selected, double-click the tally and open and close and then it will show. Select soldier 2 for motivating tally as this is the soldier behind the berm where the variance is too high.



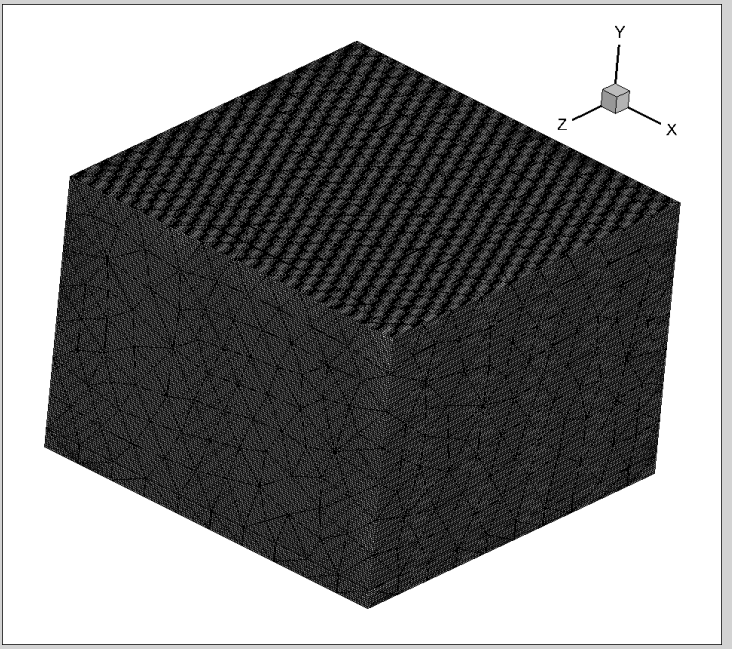




Select the "Weight Windows Settings" panel and select the default grid size. The user can specify a smaller grid size than the minimum grid for the tetrahedral grid element. We will use the size needed in the horizontal region. If we had a region of iron or something with much smaller level, we would need a smaller mesh and we can use local grid requirements to obtain the area where that is needed. We will set the weight windows grid for 10 cm in all directions.



Select preview weight windows grid. This option launches TecPlot. The result is a weight windows grid on top of the tetrahedral mesh. The user can inspect the grid to see that it matches the requirements. The user can choose to have regions of higher and lower concentration within the weight windows mesh which would be visible. This would allow for faster calculations and achieve better results if used properly. For our problem, there is a uniform rectangular mesh applied throughout and it is visible.



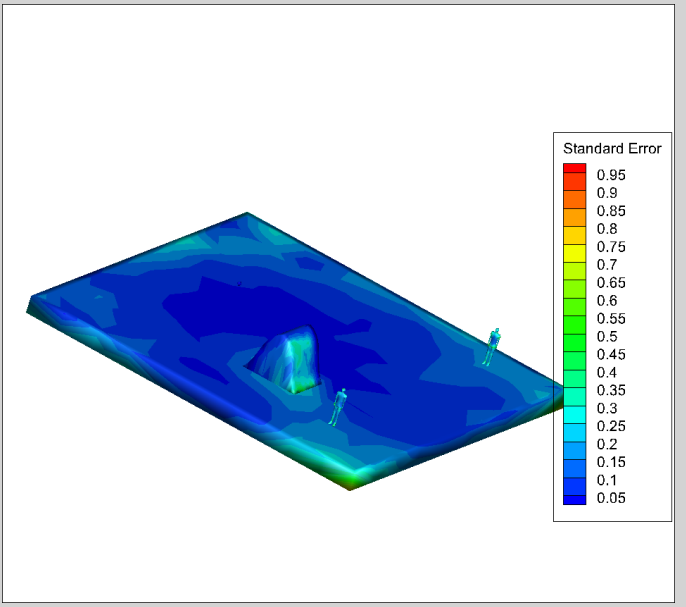
To run the problem, select the down-arrow next to "Run CADIS" and select "Run CADIS + MCNP". The program will run the CADIS deterministic solver, generate the WW and then run MCNP with those variance reduction techniques applied. When complete, examine at the output file for the results for the dose in solider 1 and soldier 2 again. The results are:

```
1tally        4        nps =    10000000
+                                 tally "Dose in Soldier 1"
          tally type 4    track length estimate of particle flux.
          particle(s): photons
          this tally is modified by dose function DE4 and DF4.

          volumes
                  cell:       4
                        7.97977E+04

 cell  4
                2.38013E-14 0.0194
```

```
1tally       14        nps =    10000000
+                                 tally "Dose in Soldier 2"
          tally type 4    track length estimate of particle flux.
          particle(s): photons
          this tally is modified by dose function DE14 and DF14.

          volumes
                  cell:       3
                        7.97977E+04

 cell  3
                7.25842E-16 0.0435
```

Note that the values for the tally are similar for both the analog and CADIS calculations, however the variance for the soldier 2 CADIS calculation goes down from 0.0844 to 0.0435 and, if we plot the Standard Error for the CADIS results, we will see the visualization of the effect where the standard

error is significantly reduced behind the berm. We could refine our problem with greater NPS if we wanted better statistics as well.



9.8 Advanced Visualization Techniques

A significant feature of Attila4MC is the ability to provide visualization of output from MCNP calculations. Providing meaningful data is important to gain understanding of phenomena, especially those that cannot be seen such as radiation. The unstructured mesh input in Attila4MC allows for complex geometries to be easily incorporated into MCNP models. Additionally, the program can be used to generate elemental edits or EEOUT files that can be interpreted by TecPlot or other software programs, such as Paraview. As presented in Example 9.3, the creation of the elemental edits simply involves the user creating the tally under the Custom Reports panel during the calculation setup. This process is the same whether it is an analog fixed-source, or CADIS problem. Once created, MCNP will write an EEOUT file that contains the data necessary for the plotting program to interpret and generate a variety of visualization products.

In Section 9.6, we introduced the basic visualization of the contour surface plot of the EEOUT file on our model. This process can be modified to adjust

the response from the given dose per source particle to dose per hour. The calculation of the multiplier was described in Section 9.4 as it applied to a single tally. For the elemental edits, there is not an option to add a tally multiplier when setting up the report. However, the user can modify the results of the elemental edits in Tecplot. The standard output for our tank model for a contour plot is presented in Figure 9.11.

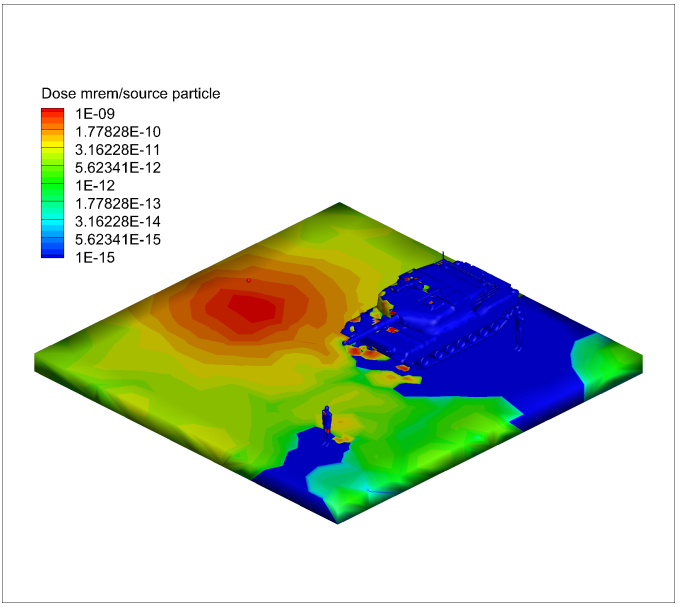


Figure 9.11: Model of tank example in TecPlot where contours do not include a tally multiplier and are elemental edits of the F4 tally in mrem per source particle.

To add the tally multiplier, the user can open up the Data Set information window and see that the variable 7 is unused as shown in Figure 9.12.

Using the Data – Alter – Specify Equations option, the user can declare that variable 7 is equal to variable 4 times a multiplier using the annotation shown in Figure 9.13. After selecting, Compute, the values in variable 7 will now reflect the F4 tally multiplied by the tally multiplier in the equation. In this example, the multiplier was 3.6E10. The user will need to calculate the proper tally multiplier based on the actual source strength as described in Section 9.4.

After V7 is calculated, the plot can be adjusted by selecting V7 as the source for the data, changing the title for the data set from V7 to Dose (mrem/hour), and adjusting the contour levels as we did in the previous examples. The result is a plot of the dose rate shown in Figure 9.14.

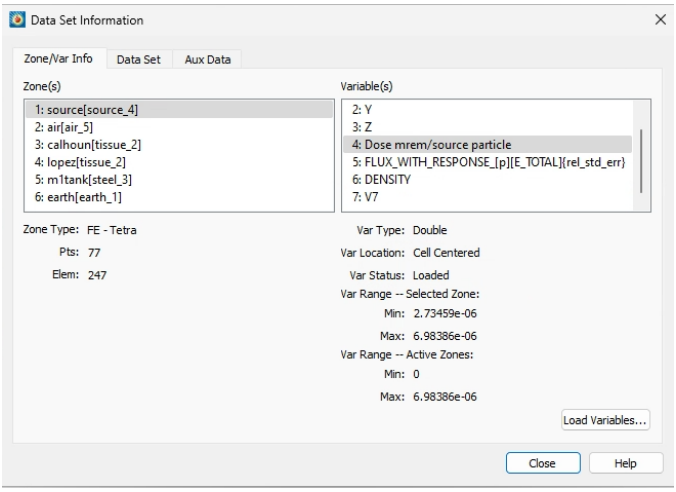


Figure 9.12: Example of the Data Set information where V7 is not used.

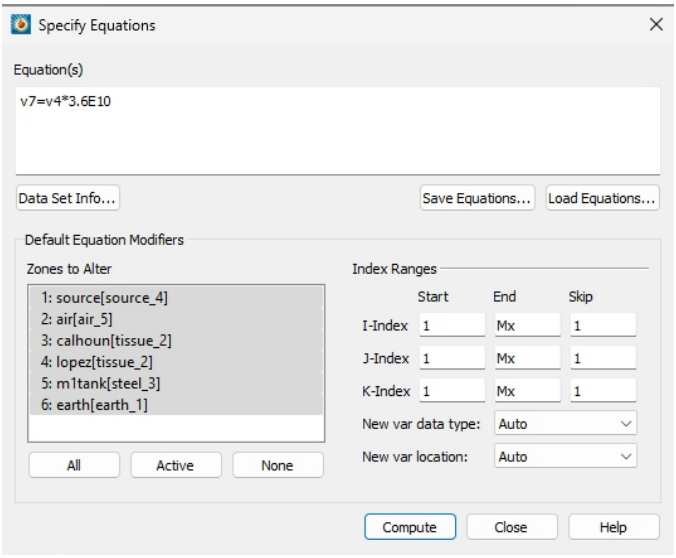


Figure 9.13: Example equation where variable 7 is the product of variable 4 and a constant.

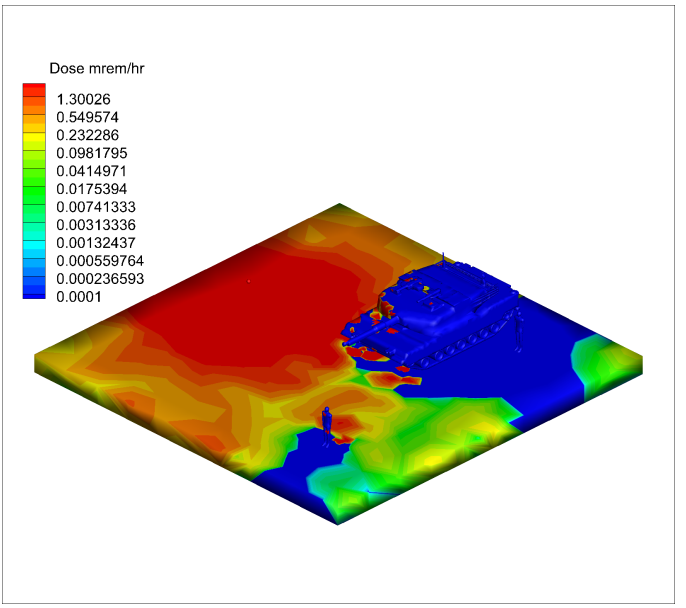
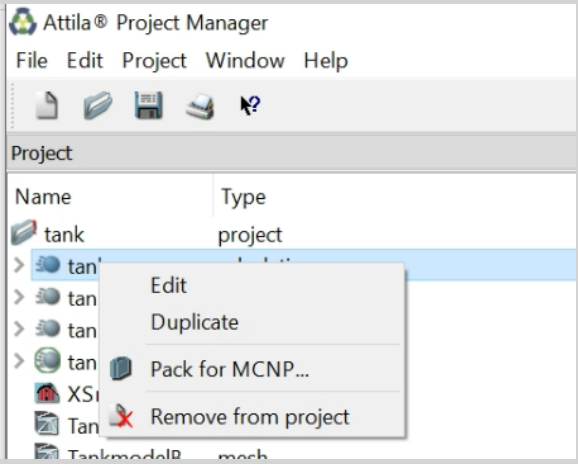


Figure 9.14: Model of tank example in TecPlot with the tally multiplier converting dose to mrem per hour.

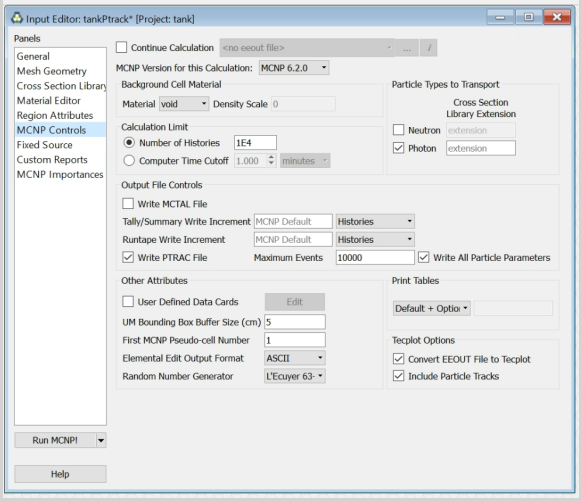
Other useful visual outputs from TecPlot include the plotting of particle tracks, performing animation of contour slices, and creating iso-surface animations. Example 9.7 describes the steps for creating particle tracks, while Example 9.8 introduces the animation of contour slices and iso-surface animations from our tank and berm problem. More detailed examples are provided in the video tutorials linked at the end of the chapter.

Example 9.7
Using the tank example problem, plot the particle tracks from the source.

Solution: To start with the plotting of particle tracks, the original calculation will need to be edited and re-ran. Right-click on the initial model with the gamma source and select **Duplicate**.

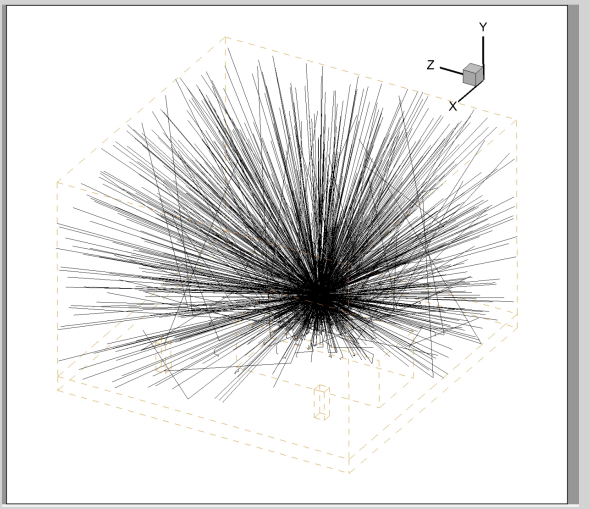


Right-click the newly created calculation and select **Edit** and change the Input Name under **General**. The only other significant change will be under the MCNP Controls panel. Change the number of histories down to 1E4 to reduce the number of tracks presented. MCNP will run a PTRAC model only on a single processor so it will take too long at 1E6 and provide excess data. Check to select **Write PTRAC File** and have a maximum events of 10000 and check **Write All Particle Parameters** and **Include Particle Tracks to Tecplot**. Save model and select **Run MCNP!** Check on **Continue** when the warning message appears.

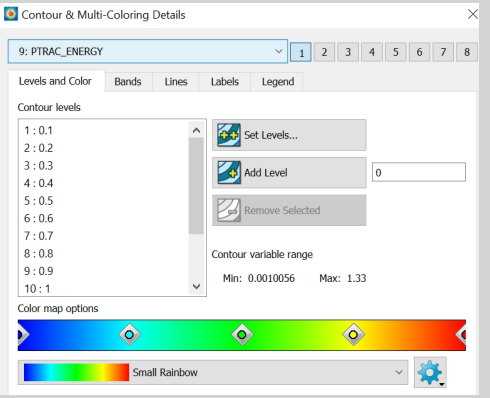


Once the run is complete, click on the arrow next to the name of the calculation to expand the names of the supporting files and then right-click on

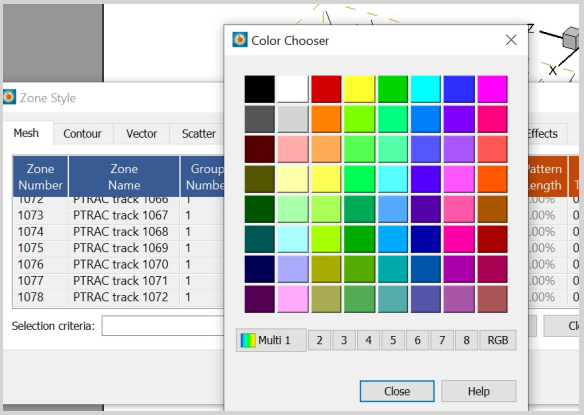
the Tecplot and select **View**. After you orient the plot by rotating it to the appropriate XYZ rotation, select Mesh and you will see the PTRAC data as all black lines.

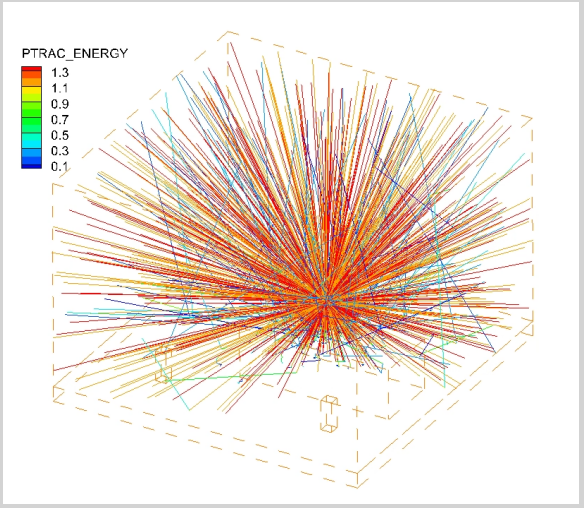


To color the particles by energy, select the **Details** button next to the **Contour** option and pull down PTRAC_ENERGY from the particle variables window, select 1 and close the menu.

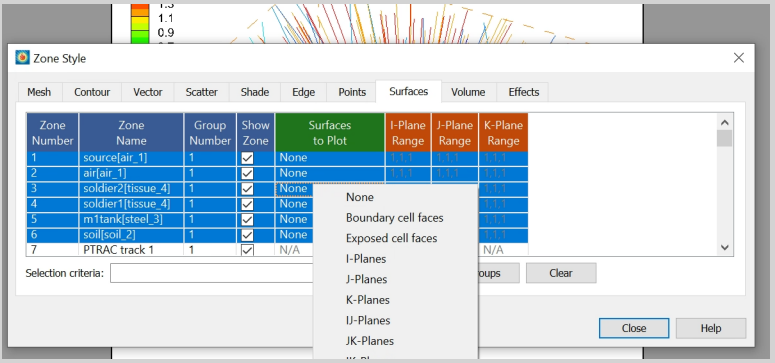


Now select the **Zone Style** button and go to the **Mesh** tab. Select all the zones that start with PTRAC. You can click on the first and then hold the shift key while moving the mouse to the end Right-click under Mesh Color and select Multi 1 for color to get tracks associated with energy.

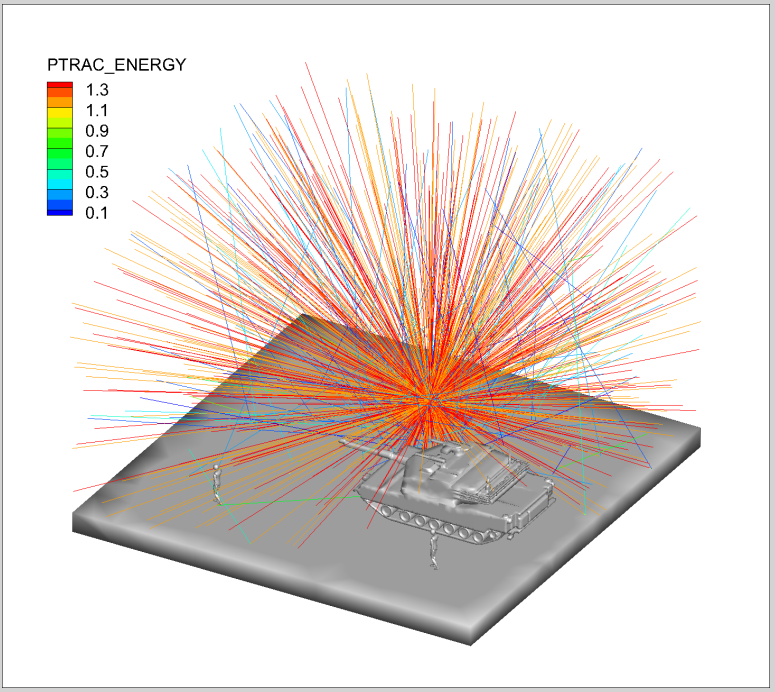




The final adjustment will be to turn on the shade for the various components in the model. This is accomplished by again selecting **Zone Style** and go to the **Surfaces** Tab. Highlight the non PTRAC surfaces and right-click under **Surfaces to Plot** and select **Boundary Cell Faces**.

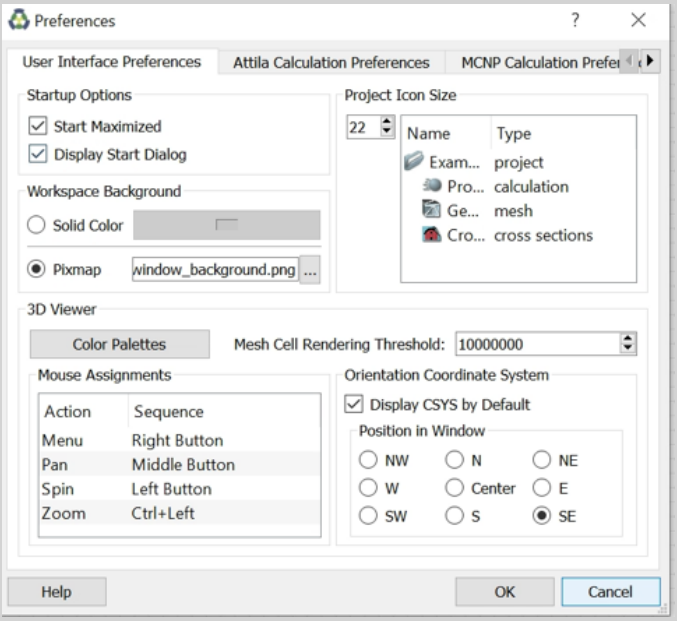


This will include the air box which we don't want so we can right-click on that surface and turn it off. Finally, select the **Mesh** tab and highlight the non-PTRAC zones and right-click under **Show Mesh** and select **No**. Finalize the format by arranging the format of the legend and turning off the bounding box. Use the File Export function to get an high-quality PNG picture of the output.

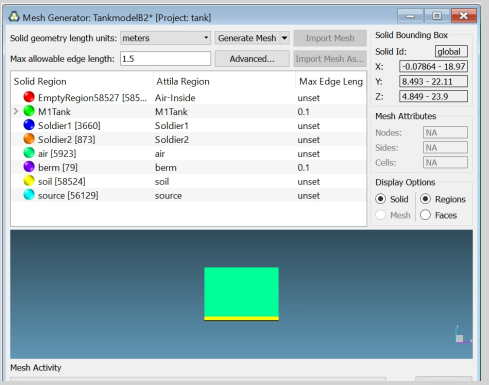


Example 9.8
Using the tank with berm example problem, animate contour slices and iso-surfaces.

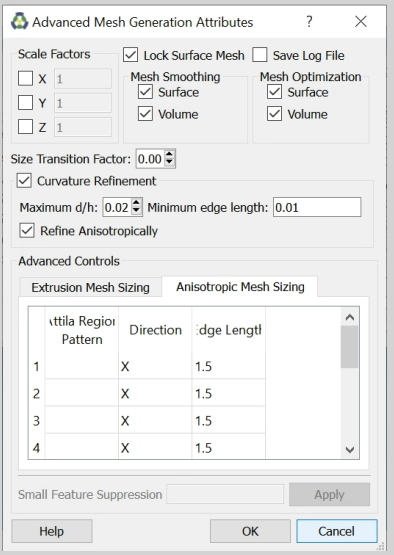
Solution: For this problem, the number of cells produced in the mesh will exceed the default number allowed in the program. We can avoid issues by opening the **Edit- Preferences** window and select the **User Interface Preferences** tab. Once open change the **Mesh Cell Rendering Threshold** from 1000000 to 10000000 (add a zero).



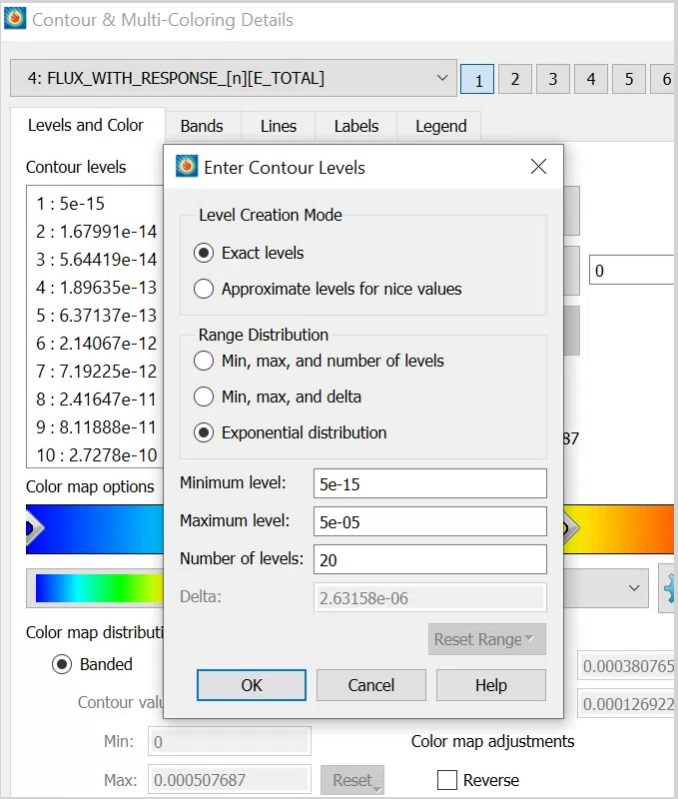
Close the preferences and load the model by right-clicking on the solid geometry and selecting **Generate Mesh**. The load method we used for the original problem was MCNP. For this model, we need to load the problem using the Attila Non-Manifold Non-Imprint. The reason is because we want Attila to identify the empty regions so we can assign Air to the space from the air box down to the surfaces. Because of the size and complexity of the problem, the standard Attila Non-Manifold load won't work. Once loaded, change the name of the Empty Region identified by Attila to Air-Inside.



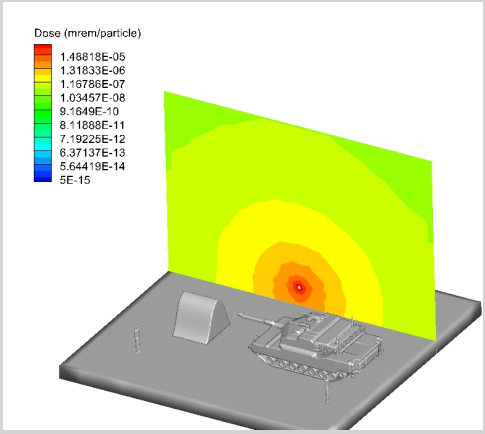
Set the maximum edge length on the main menu as 1.5 meters (1/10th to 1/20th the maximum dimension of the problem). Add a smaller max edge length of 0.1 for the two shielding objects (tank and the berm). Change the **Advanced** options to include a **Size Transition Factor** of 0.00, include **Curvature Refinement**, a minimum edge length of 0.01 and **Refine Anisotropically**. Generating and importing the mesh can take quite some time due to the complexity of the problem. After importing the mesh, save the project. Duplicate the previous MCNP calculation of the model of the tank with berm and update it with the new mesh. Verify each of the menu options are still correct with the new geometry. Run the MCNP model. This may take $2-3$ hours.



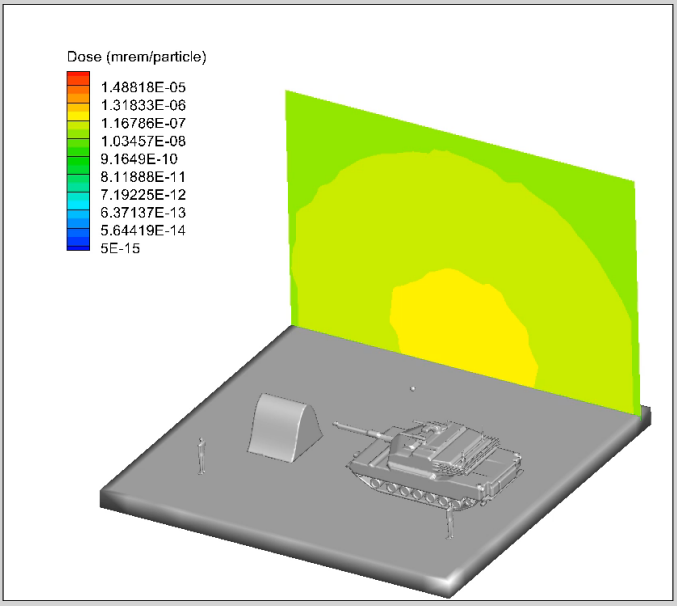
When the model is complete, expand the calculation and open the Tecplot. Turn on the **Shade** and right-click on the model, you can hide the air box and there will be an additional area of air just inside. Hide that area too. Select the **Slices** option and you will get a slice through the entire model not just the borders. Select the **Contour** settings button next to Contour and **Set Levels** and change to an **Exponential** distribution with a min of 5e-15 and a max of 5e-05 wit 20 levels.



Adjust the legend to clean up the graph area. There are two buttons to the right of the slices option. The first is an interactive slicing tool which allows the user to click anywhere on the model and the slice will move to that point. The second button gives a menu option to allow the user to change the slice location, position it exactly in the model and has an **Animate** tab.

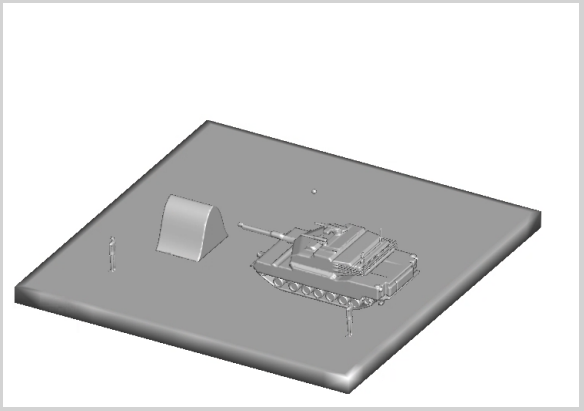


In the **Animate** tab, we can change the speed, animation direction, frame rate etc. Press play, it moves through the model. Change the direction to Loop and it will continue to go through until you press pause. If we select the Export button, it will save the animation as a variety of formats.

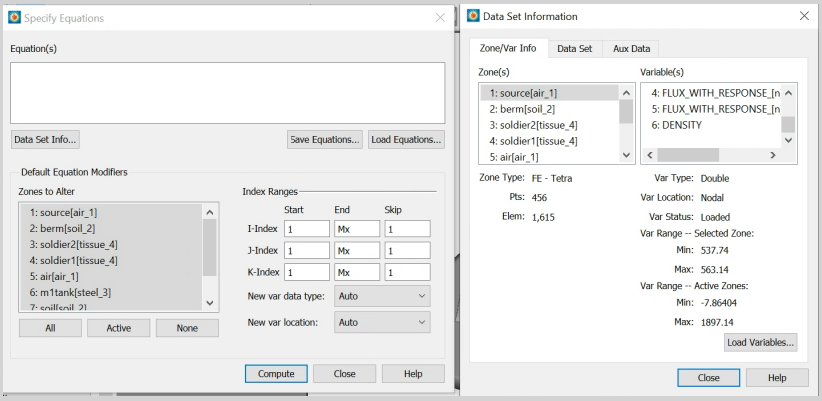


Iso-surfaces can be useful for visualizing radiation transport, especially local hot spots or areas of leakages. Begin by opening our Tecplot for the model and selecting the shade for the zones that we want visible along with the animation. For this problem, use the same Tecplot we used for the slices

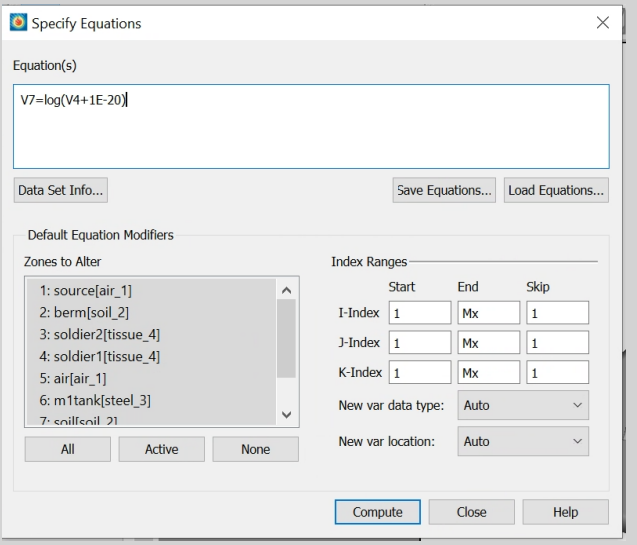
– Tank with Berm using the Attila Non-Manifold Non-imprint load for the mesh.



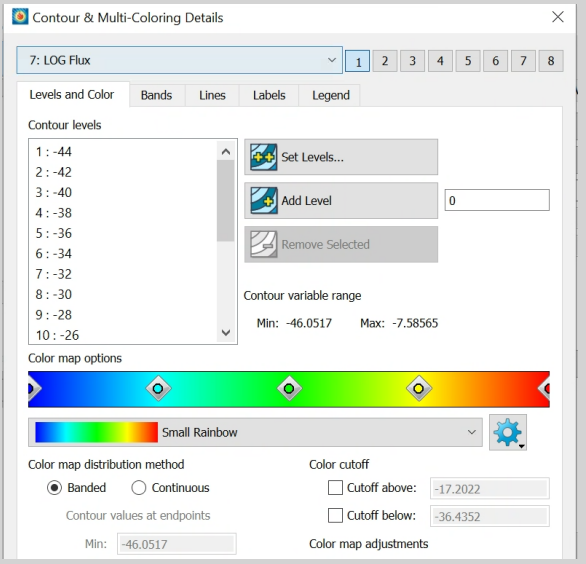
To get a good plot, we will need to create a new variable that is the log of the flux Go to the top menu and select **Data – Alter** and **Specify Equations**. Now select data set info to see which is the next variable that is unused. In this case, Variable 6 is DENSITY and Variable 7 is unused.



Enter an equation that states variable 7 is the log of variable 4 (the flux). Enter V7=log(V4) and it returns an error because some of the V4 values are zero. To get around that, add a very small value to V4 to eliminate zero values in the log function, i.e. V7=log(V4 + 1E-20). Select Compute and the V7 shows in the Data Set Information window. Double-click that variable and rename it Flux Log. Close the Specify Equations and Data Set Info Windows.

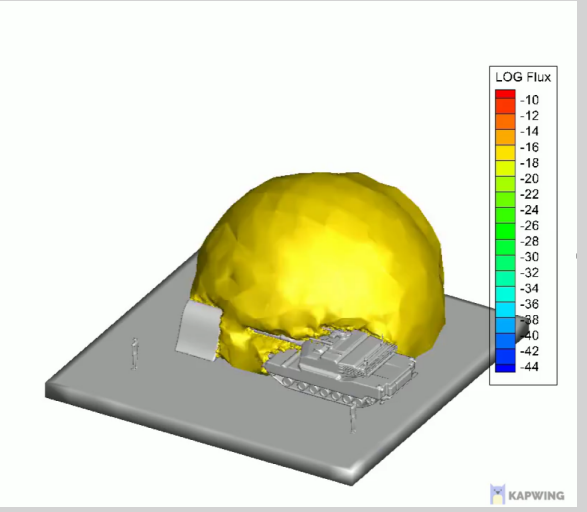


Select the **Iso-surfaces** box and open the details by selecting the button next to the name Iso-surfaces. Select the button to the right of the Define Iso-Surfaces Using and you will get the Contour and Multi-Coloring Details window. Select the Log flux variable 7 from the drop-down menu. Close the window.



We can slowly toggle through the Value 1 variable to get a sense of how the

Log Flux Iso-surfaces change in the model. Go to the animate tab and press play and the iso-surfaces will go in reverse. Change the direction to Backward and change the number of steps to 50 and the animation speed to 10. Save the animation as a video.



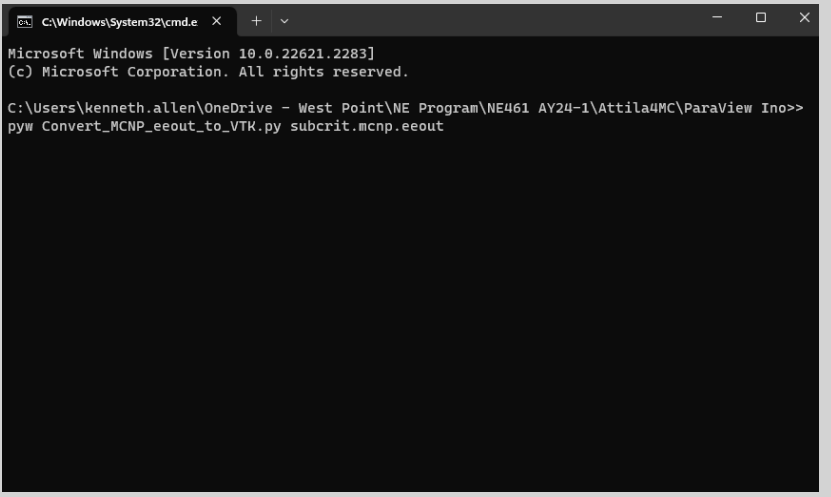
TecPlot is the integrated visualization platform distributed with the Attila license package. There are other programs available to view and manipulate the elemental edits output files from MCNP. Notably, ParaView is an open source post-processing visualization engine funded by Los Alamos National Laboratory. (ParaView; 2023) ParaView has more options and capabilities than TecPlot. For some applications, ParaView may be a better choice to visualize elemental edits from a model. For example, if the elemental edits file produces values listed as NaN or, not a number, TecPlot will not normally plot the results. This could occur for regions with negative values for flux. The user can modify the file and set the elemental edits file by replacing the NaN values with zero and attempting to load into TecPlot. However, another option is to use ParaView in conjunction with the python parser distributed by LANL. Example 9.9 details the use of a ParaView to generate a contour sketch of a complex model from an elemental edit file created by an Attila run.

Example 9.9
Use the elemental edits file generated in an Attila run to create a surface contour plot in ParaView.

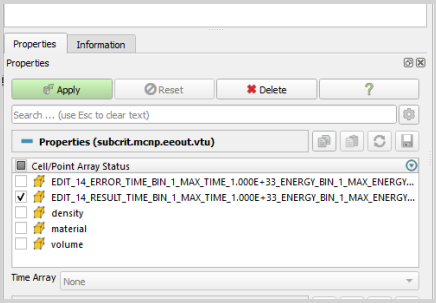
Solution: The elemental edits file used for this problem is from a model

where a person is standing approximately one meter from a subcrital assembly consisting of a hexagonal lattice of natural uranium fuel inside a tank of light water with five PuBe sources. The elemental edits file is stored in the same file folder location as the *.o, or output file. The elemental edits is saved with a *.EEOUT extension and is usually the largest file in the run folder.

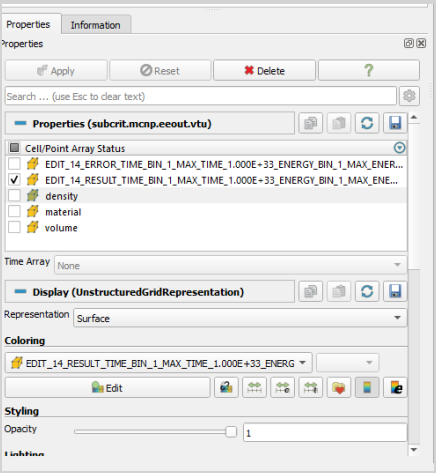
To use ParaView, the elemental edits file will need to be parsed into a *.VTU file which is the format used for input. Los Alamos Laboratory provides instruction on developing a python script that will parse and format the elemental edits file correctly. (Kulesza and McClanahan; 2019) To use the script, ensure Python is loaded on the computer. The user can go to https://www.python.org/downloads/ and download and install the latest version for their operation system. Once installed, search for the executable in the root drive for example, C:\Windows\pyw.exe. Within the Windows file explorer, navigate to the folder location where the python script and the EEOUT file is located and type CMD to open a command prompt in that location. The user can either copy the python script into the run folder with the EEOUT file or create a new folder and copy both files into it. Upon opening the command prompt in the file location, the user will execute the script by giving the command pyw Convert_MCNP_eeout_to_VTK.py filename.eeout. The script will then create a VTU file with the same name as the elemental edits file.



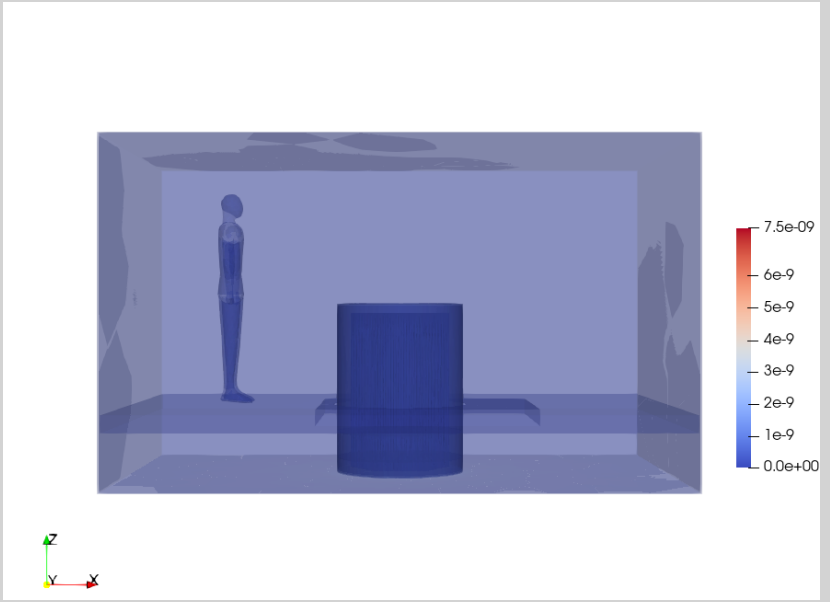
Once the VTU file is created, open ParaView and load the file. There are several options for variable to load visible in the lower left of the screen. Select either energy bins or the error whichever is desired for plotting and deselect the density, material, and volume.



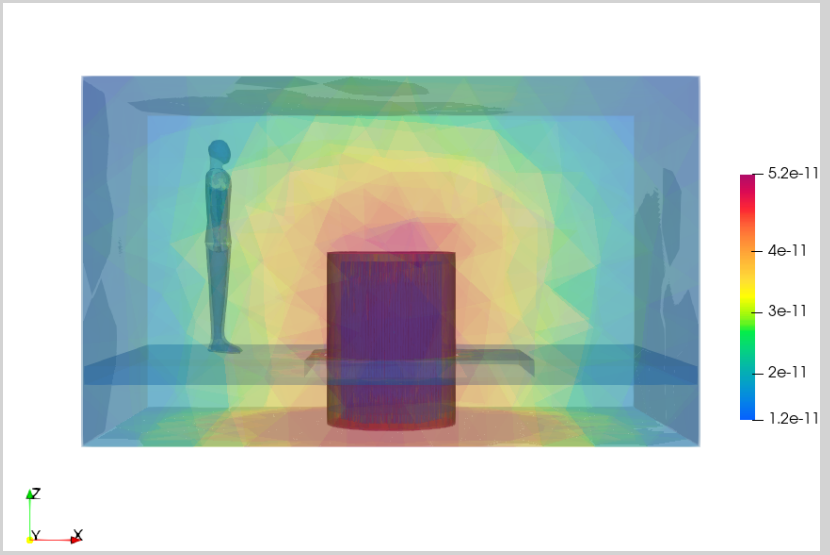
Select **Apply** and a shade grey box will appear in the center of the plotting region which represents the overall boundary of the model. Under **Coloring** in the lower-left Properties region, change from Solid Color to the name for the tally result.



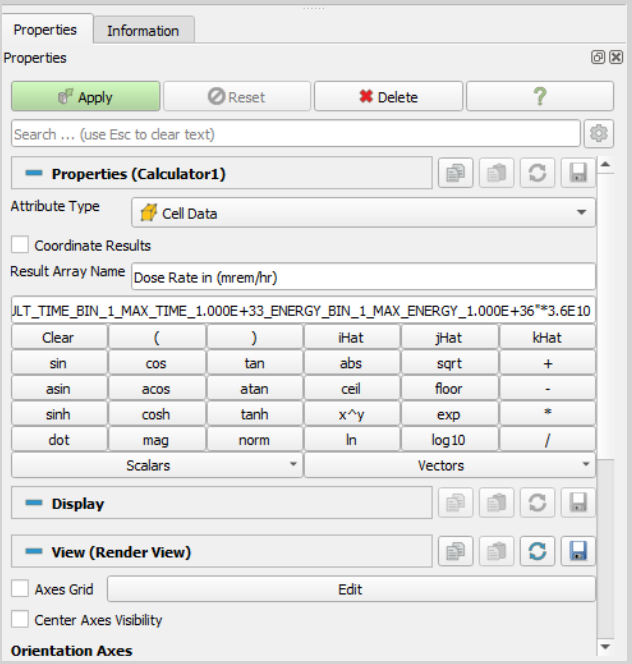
The Color Map Editor will appear on the right side of the model and the box will display as a solid blue. Turning down the **Opacity** will reveal the interior of the model, but the scaling will need to be adjusted to better represent the differences in the dose across a gradient.



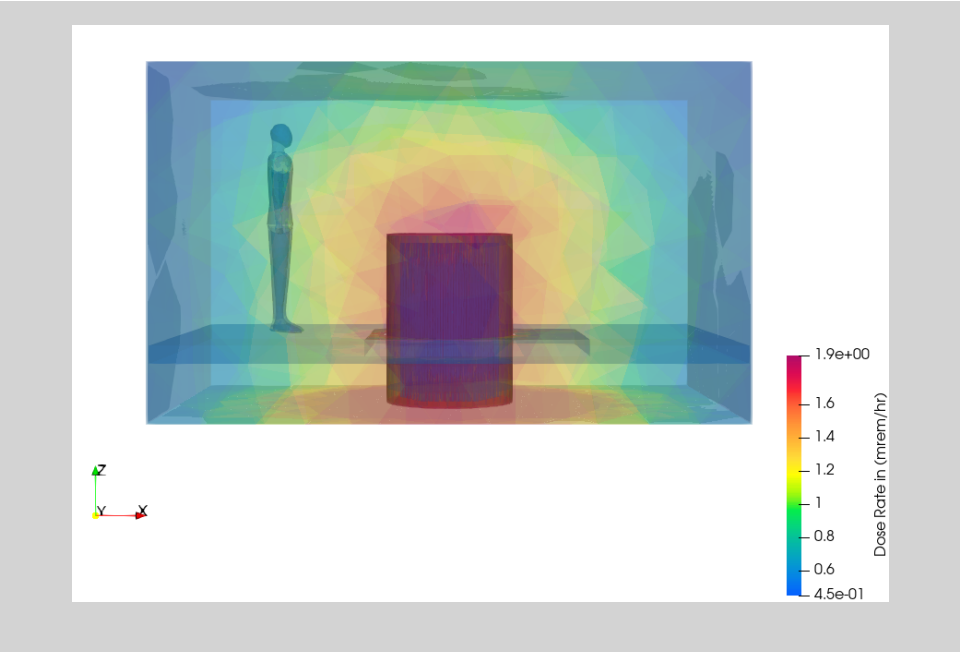
Using the Color Map Editor, change the Mapping Data to **Rainbow Uniform** and select the option to **Rescale to Visible Range** which is the third fourth button option on the right side of the mapping data scale. This option will result in an image that is more representative of the change in flux.



The user can then move the model about for different views and adjust lighting and camera angles to best represent the results. Additionally, a tally multiplier can be added to convert from dose per source particle to dose per hour as was done in TecPlot. To do so in ParaView, the user will select the **Calculator**. Select the calculator button on the menu to the far left above the file tree. When the calculator opens, select **Scalars** and choose the tally data. The name for the data will appear in the space above the calculator buttons inside quotes. Navigate to the end of the name and include *(value of tally multiplier). The example below shows the use of 3.6E10 as a multiplier. Change the **Result Array Name** to reflect the appropriate name, in this case Dose Rate (mrem/hr).



Select **Apply** and the model will update with the new data and revert to the original settings for color and shading. Once again, change the opacity, set the color mapping, and scaling. The user can then change the angle and lighting to view from various orientations. ParaView also allows the user to export the images to a high-quality PNG or JPG format using the **Save Extracts** option to see rendered views.



9.9 Using a KCODE as a Source in Attila4MC®

By now we have used Attila4MC for a variety of fixed-source shielding applications. We have used both neutron and photon sources and we have explored the use of the CADIS variance-reduction methods. Attila is especially useful in setting up and running unstructured mesh problems on complex geometries. However, as of now, Attila will not run KCODE calculations to determine criticality. This is a limitation when the user desires to accurately calculate the dose rate from a criticality source such as a reactor or spent fuel pool.

It is possible to obtain an accurate model of the shielding from a criticality source using Attila if we break the problem into steps. First, the critical source must be modeled in MCNP. For a reactor, this would be the reactor core region. The user models the core region to include the water reflector just inside the pressure vessel. The model will have a vacuum boundary condition and the user will apply a F1 tally on the outside surface. The F1 tally will measure the current, or leakage on the outside surface of the model. The F1 tally will need to include several energy bins across the entire spectrum of energy from the source down to low-energy thermal neutrons. The results of the tally on the surface will be used to create a neutron source spectrum for the Attila model.

Once the results are known for the leakage from the criticality source, the

user will be able to create a Parasolid matching the model with the addition of a thin region on the same surface where the F1 tally was taken in MCNP. This thin region will serve as a source region for the Attila run. If there is a significant difference in the spectrum from different sides of the model, multiple F1 tallies should be applied with multiple source definitions and regions. For most problems, the user can apply a single F1 tally on the outside of the criticality source and use a single region around the model.

As with any MCNP tally, the results are presented normalized to source particle. For a fixed source, we have calculated the tally multiplier several times to generate flux or dose rate per unit time. Calculating the tally multiplier for a critical source is performed by using the power of the model. Criticality problems always involve power. Power is normally measured in Watts or J/s. The recoverable energy per fission for thermal fission is approximately 200 MeV or 3.20435×10^{-11} Joules. Each fission from ^{235}U gives off approximately 2.5 neutrons. For a critical source of X Watts of power, the number of neutrons per second N is computed using Equation 9.1 which reduces to Equation 9.2

$$N = X \left[\frac{\mathrm{J}}{\mathrm{s}}\right] \left[\frac{1\mathrm{MeV}}{1.6022 \times 10^{-13}\mathrm{J}}\right] \left[\frac{1\mathrm{fission}}{200\mathrm{MeV}}\right] \left[\frac{2.5\mathrm{neutrons}}{1\mathrm{fission}}\right] \tag{9.1}$$

$$N = X \left[\frac{\mathrm{J}}{\mathrm{s}}\right] [7.802 \times 10^{10}] \tag{9.2}$$

Example 9.10 uses the same water-moderated subcritical assembly presented in Example 9.9 to create a neutron source from the F1 tally results from MCNP and then integrate the source spectrum into an Attila fixed-source calculation of dose rate with the appropriate tally multiplier for the system.

Example 9.10
A subcrital assembly consists of fuel rods of natural uranium in a light-water moderator and is pictured below.



The assembly details are given in the table below:

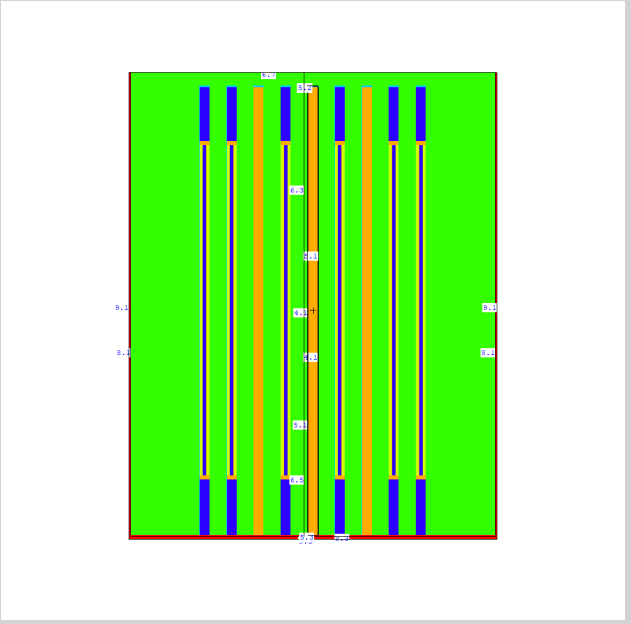
Specifications for USMA Subcritical Assembly

Parameter		Value
Tank	Diameter	116.8 cm
	Height	152.4 cm
	Thickness	0.59 cm
Active Fuel Height		145 cm
Composition		Total: 276 rods
		Fuel: 271
		Neutron Source: 5
Neutron Source		1 Ci, Pu-Be
Fuel Material		Natural Uranium
Moderator		Light Water
Fuel Holder (Lucite)		0.287 cm
Air Gap Thickness		0.056 cm
Fuel Thickness		0.901 cm
Second Air Gap Thickness		0.017 cm
Aluminum Clad Thickness		0.121 cm
Pitch (center-to center)		5.0 cm

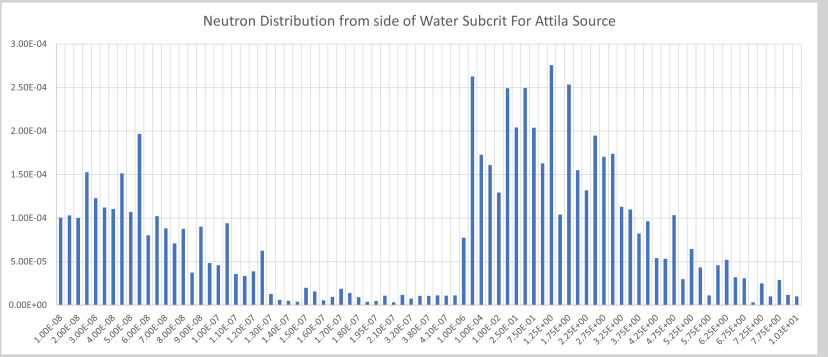
For the given subcrital assembly, assume a power of 9.0×10^{-5} Watts. Use Attila to calculate the dose rate to a person standing 1 meter from the side of

the tank in mrem per hour and create a dose rate contour plot of the assembly in the room with a person standing 1 meter away.

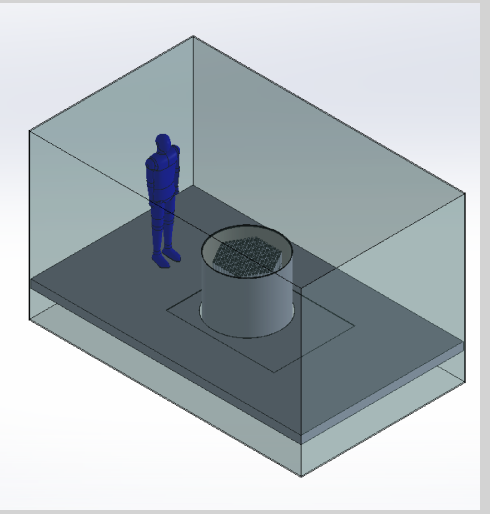
Solution: To begin the process of using a criticality source in Attila, create an MCNP model of the subcritical assembly. The assembly is a hexagonal array in a triangular pitch. Modeling the individual unit cells and the assembly follows the same methods presented in Example 8.2 and 8.3 for the SCFR assembly. Add an F1:N tally to the outside surface of the subcrit. Include a wide and detailed distribution of energy bins to get a good spectrum for the neutrons exiting the tank. The results of the F1 tally will be the basis for the neutron source file for the Attila model. Determine the appropriate surface to apply the F1 tally. The easiest way to do so is to plot the model in Visual Editor or the MCNP Plotter and turn on the surface indices.



Recall that the F1 tally requires a surface number. The macrobody RCC for the tank outside has three surface (9.1, 9.2 and 9.3). For this model, the leakage as potential dose to our persons is best defined by the side surface (9.1). After the KCODE run is complete, export the tally results to Excel. We can plot the tally vs. energy as we did with the californium source creation in Example 9.4. Note there are several energy bins with zero hits. Apply a filter to hide the values that are zero before creating the Attila source or they will cause issues during the calculation.

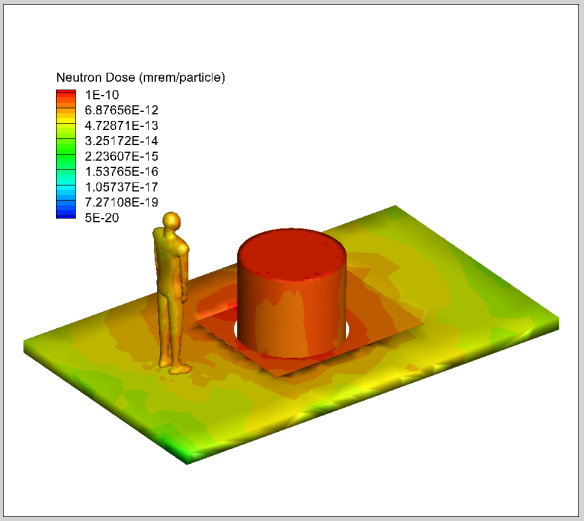


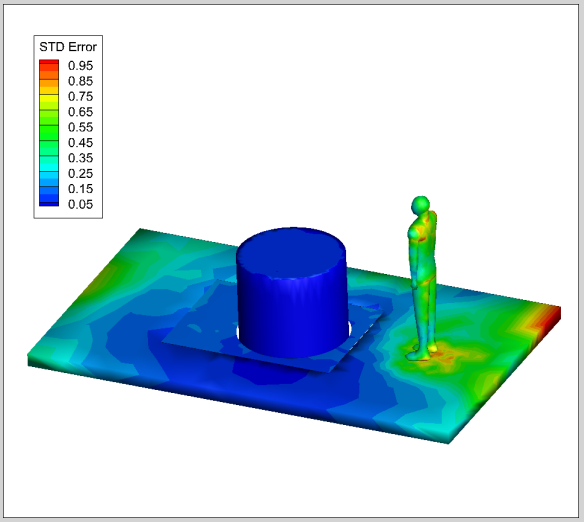
The next step will require modeling the problem in Solidworks to create the solid body for use in Attila. For the Solidworks model, include everything in the subcrit and add a shell of air on the outside of the tank. This thin shell of air on the outside will serve as the source region in Attila. By explicitly modeling the interior geometry, the model accounts for the shielding provided by the core itself. Because of the complexity of the model, create the water region and the air inside within Solidworks and load the model as an MCNP vs. loading as an Attila method that will identify empty regions.

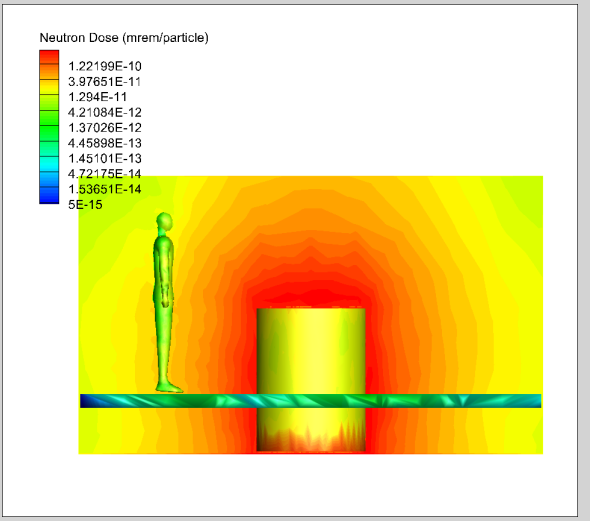


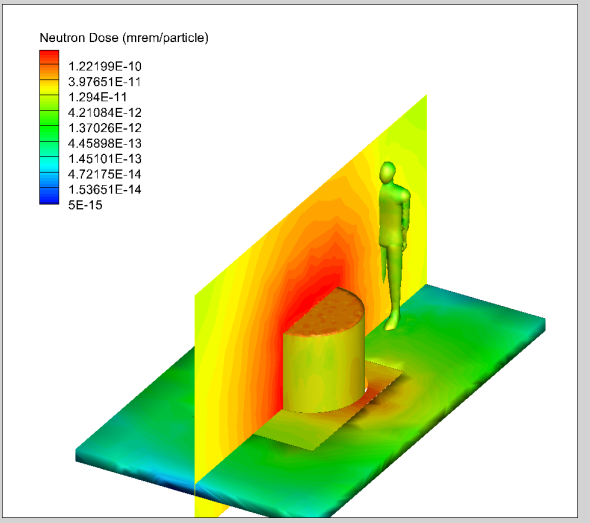
After the solid body and the source neutron distribution is created, the steps to modeling the problem are the same as Example 9.4 for creating a source from a distribution and the previous non-CADIS calculations for a fixed source analog MCNP model in Attila. The user can use the Cask in Room example to facilitate the use of the pre-made materials from PNNL's *Compendium*. Using Equation 9.2, the tally multiplier for the F4 tally on the person will be 2.5×10^{10}

to obtain a dose rate in units of mrem per hour. That same value can be used in TecPlot as an equation to generate plots in dose rate or the plots can be left in units of dose per source particle. Example plots of the dose and error are presented below:






Neutron Dose (mrem/particle)
1.22199E-10
3.97651E-11
1.294E-11
4.21084E-12
1.37026E-12
4.45898E-13
1.45101E-13
4.72175E-14
1.53651E-14
5E-15


Neutron Dose (mrem/particle)
1.22199E-10
3.97651E-11
1.294E-11
4.21084E-12
1.37026E-12
4.45898E-13
1.45101E-13
4.72175E-14
1.53651E-14
5E-15

Bibliography

Becchetti Jr, Frederick Febbraro, M. . T.-I. R. . O. M. . B. L. (2013). Cf-252 fission-neutron spectrum using a simplified time-of-flight setup: An advanced teaching laboratory experiment, *American Journal of Physics* **81**: 112–119.

Kulesza, J. A. and McClanahan, T. C. (2019). A python script to convert mcnp unstructured mesh elemental edit output files to xml-based vtk files.
URL: *https://www.osti.gov/biblio/1574726*

ParaView (2023). Paraview, about.
URL: *https://www.paraview.org/about/*

Root, M. (2014). *Technical basis for use of a correlated neutron source in the uranium neutron coincidence collar*, PhD thesis.

SilverFir (2023). Introduction to attila4mc.
URL: *https://silverfirsoftware.com/Attila4MC.php*

West, III, J. T. (1986). Sabrina: an interactive three-dimensional geometry-modeling program for mcnp.
URL: *https://www.osti.gov/biblio/7152240*

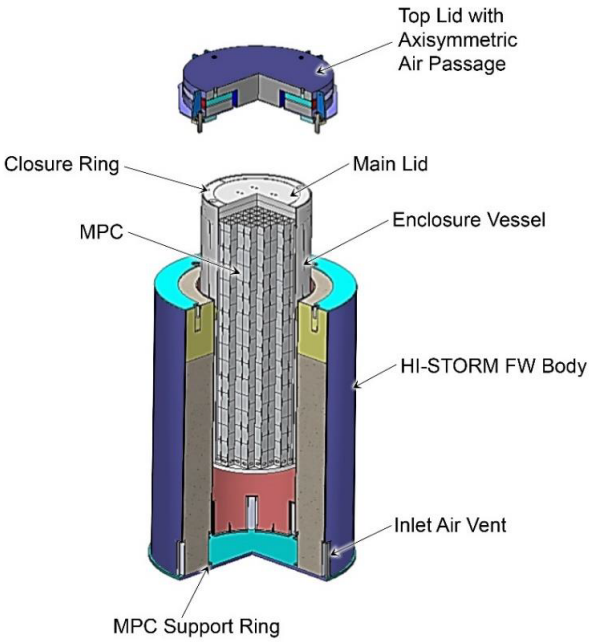
9.10 Video Tutorials

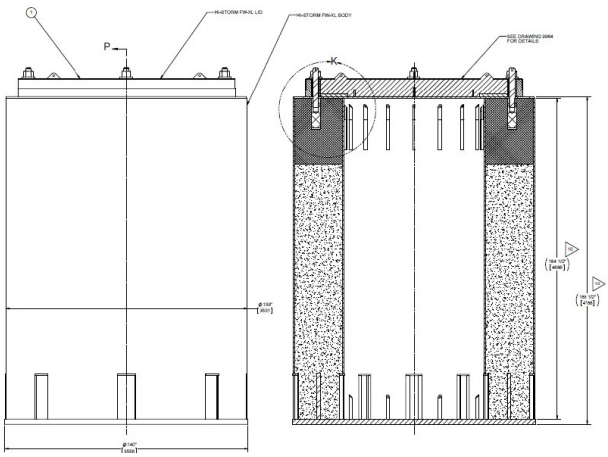
The following video tutorials are available to assist your understanding of the material from this chapter.

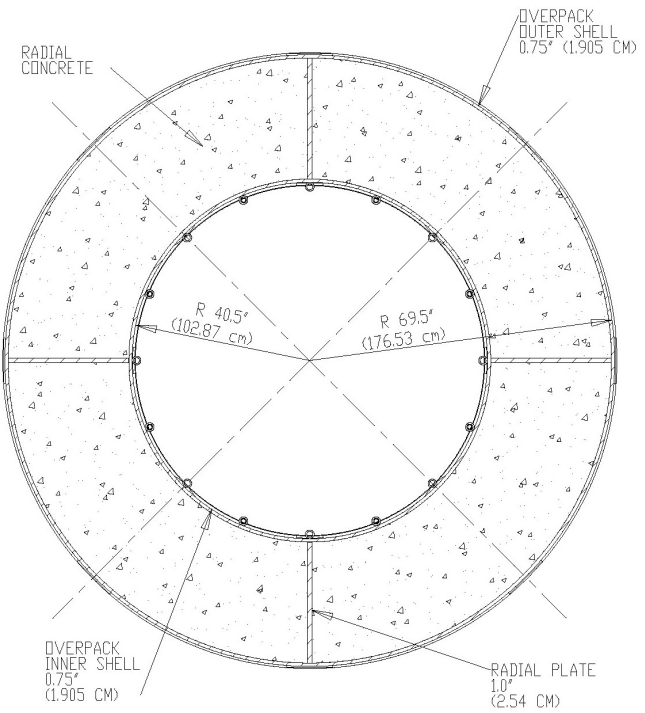
- Parasolid Model M1 Tank Example
- Generate Mesh M1 Tank Example
- Fill Empty Regions with Solidworks
- Setup Attila Calculation M1 Tank Example
- Output and TecPlot Tank Example
- CADIS Example Part 1
- CADIS Example Part 2
- Viewing Results in TecPlot
- Plot Particle Track in Attila
- Animate Slice Attila
- Animate Iso Surfaces
- Create PuBe Source
- Tally Multiplier in TecPlot
- View EEOUT in ParaView
- Using KCODE in MCNP to Generate Source for Attila

9.11 Problems

1. The Holtec corporation manufacturers dry storage casks for used nuclear fuel. Rough diagrams for the Holtec Highstorm overpack are provided below.







Create a solid model of Holtec Highstorm overpack shield and Parasolid.

2. Create an Attila file for the Parasolid in problem 1 using a 1 mega-Curie Californium neutron source and report the dose rate in mrem/source particle to a person standing one meter from the surface.

3. Calculate the appropriate tally multiplier for problem 2 and report the dose rate in mrem/hour from the same problem modeled in problem 2.

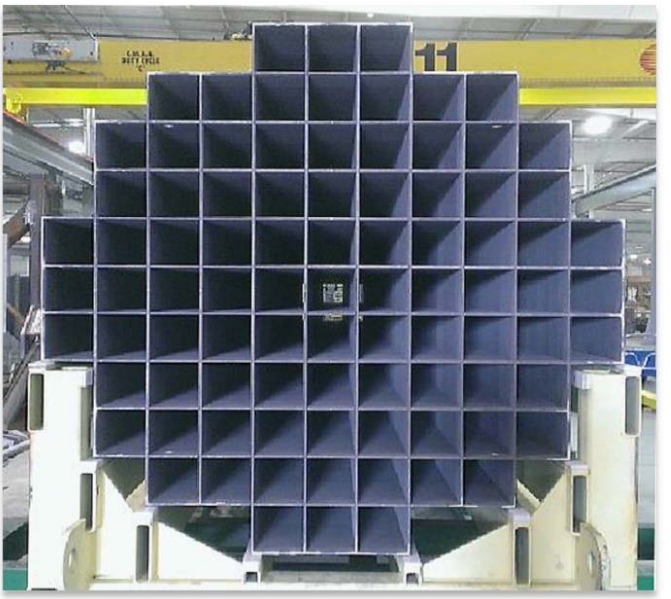
4. Report the variance from problem 3 in the MCNP model. Create a CADIS problem from the model in problem 3 with the person as the motiving tally. Report the variance for the CADIS problem and the change from the analog MCNP version.

5. Generate TecPlot images for Problem 3 and 4 to include a contour plot of the dose rate, slices through the model, and plots for the analog and CADIS standard error.

6. Use the BURN feature in MCNP to deplete a Westinghouse 17 x 17 PWR assembly like the one in Example 6.5 to a criticality value less than 1.0. From the output file, use the material composition for the

actinide and non-actinide materials in the burnup where the assembly goes subcritical to recreate the assembly with that material composition. This will represent the composition of the used nuclear fuel.

7. Create repeated lattice of Westinghouse 17 x17 PWR assemblies with the fuel composition from Problem 6 arranged to fit in the grid inside the Holtec Highstorm storage cask pictured below. Complete a KCODE calculation on the model with an F1 tally for neutrons and photons on the outside area around the array.



8. Create a solid model of an array of 17 x 17 Westinghouse PWR assemblies, like those modeled in Example 6.5 to fill the grid assembly in Holtec Highstorm cask.

9. Create two Attila models of the Holtec Highstorm cask solid body model created in Problem 8. Each will use the results from problem 7 as the source. One neutron and one photon source with dose rate tallies on a person standing 1 meter from the surface of the cask. Assume an appropriate power from the assemblies to generate a tally multiplier.

10. Generate TecPlot images for the models in Problem 9 of the surface contour dose rates for photons and neutrons.

Index

A
absorption interactions, 15
algorithm, 24
alpha decay, 2
assembly, 208
Attila4MC, 283

B
batch files, 41
Bateman equations, 5
beta decay, 3
boss, 197
burnup, 165

C
CADIS, 321
Compton scattering, 13
conditional statements, 31
confidence intervals, 91
cumulative dist. function (CDF), 82
cut, 198

D
decay constant, 4
declaration, 25
depletion, 74
diffusion theory, 56
discrete ordinates approximation, 52
DOS commands, 38

E
elastic scatter, 15
energy groups, 54
Evaluated Nuclear Data File, 16

F
Fick's law, 56
flux, 51
for loop, 31

G
gamma decay, 3
geometry splitting, 151
global unit, 231

H
half-life, 4
hexagonal lattice, 235
high level language, 28

I
inelastic scatter, 15
intersection, 115

L
Legendre polynomials, 56
line of stability, 1
linear attenuation coefficient, 14
logical flowcharts, 25
low level language, 27

M
macro function, 43
macrobody, 113
macroscopic cross section, 16
material card, 106
mesh, 289
microscopic cross section, 16
mirror boundary condition, 58
Monte Carlo methods, 81
MT numbers, 17

N
neutron interactions, 15

neutron transport equation, 51

O
order of quadrature, 69
ORIGEN, 10

P
P_1 approximation, 56
P_N method, 56
pair production, 14
Parasolid, 284
ParaView, 347
particle plotting, 178
peak-to-average flux, 177
periodic boundary condition, 58
photoelectric effect, 13
probability dist. function (PDF), 82
programming procedure, 31

R
radioactive decay, 1
random error, 90
random walk process, 84
Reactor Core Model, 250
repeated pattern, 200

S
SABRINA, 281
SCALE, 54
solid body, 188
standard deviation, 90
stochastic volume calculation, 158
surface cards, 109
swept boss, 202
systematic error, 91

T
temperature coefficient, 71, 160
template, 192

U
uncertainty, 89
union, 115
unit, 231
unit cell, 58
universes, 169

V
variance reduction, 147, 321
Visual Editor, 136
voxel, 53

W
while loop, 30

Z
ZAID, 61, 107