U.S.NRC
United States Nuclear Regulatory Commission
Protecting People and the Environment

NUREG/CR-7041
ORNL/TM-2011/21

SCALE/TRITON Primer: A Primer for Light Water Reactor Lattice Physics Calculations

Office of Nuclear Regulatory Research

AVAILABILITY OF REFERENCE MATERIALS IN NRC PUBLICATIONS

NRC Reference Material

As of November 1999, you may electronically access NUREG-series publications and other NRC records at NRC's Public Electronic Reading Room at http://www.nrc.gov/reading-rm.html. Publicly released records include, to name a few, NUREG-series publications; *Federal Register* notices; applicant, licensee, and vendor documents and correspondence; NRC correspondence and internal memoranda; bulletins and information notices; inspection and investigative reports; licensee event reports; and Commission papers and their attachments.

NRC publications in the NUREG series, NRC regulations, and Title 10, "Energy," in the *Code of Federal Regulations* may also be purchased from one of these two sources.

1. The Superintendent of Documents
   U.S. Government Printing Office Mail Stop SSOP
   Washington, DC 20402–0001
   Internet: bookstore.gpo.gov
   Telephone: 202-512-1800
   Fax: 202-512-2250
2. The National Technical Information Service
   Springfield, VA 22161–0002
   www.ntis.gov
   1–800–553–6847 or, locally, 703–605–6000

A single copy of each NRC draft report for comment is available free, to the extent of supply, upon written request as follows:

Address: U.S. Nuclear Regulatory Commission
Office of Administration
Publications Branch
Washington, DC 20555-0001
E-mail: DISTRIBUTION.RESOURCE@NRC.GOV
Facsimile: 301–415–2289

Some publications in the NUREG series that are posted at NRC's Web site address http://www.nrc.gov/reading-rm/doc-collections/nuregs are updated periodically and may differ from the last printed version. Although references to material found on a Web site bear the date the material was accessed, the material available on the date cited may subsequently be removed from the site.

Non-NRC Reference Material

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, transactions, *Federal Register* notices, Federal and State legislation, and congressional reports. Such documents as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings may be purchased from their sponsoring organization.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at—

The NRC Technical Library
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852–2738

These standards are available in the library for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from—

American National Standards Institute
11 West 42nd Street
New York, NY 10036–8002
www.ansi.org
212–642–4900

Legally binding regulatory requirements are stated only in laws; NRC regulations; licenses, including technical specifications; or orders, not in NUREG-series publications. The views expressed in contractor-prepared publications in this series are not necessarily those of the NRC.

The NUREG series comprises (1) technical and administrative reports and books prepared by the staff (NUREG–XXXX) or agency contractors (NUREG/CR–XXXX), (2) proceedings of conferences (NUREG/CP–XXXX), (3) reports resulting from international agreements (NUREG/IA–XXXX), (4) brochures (NUREG/BR–XXXX), and (5) compilations of legal decisions and orders of the Commission and Atomic and Safety Licensing Boards and of Directors' decisions under Section 2.206 of NRC's regulations (NUREG–0750).

U.S.NRC
United States Nuclear Regulatory Commission
Protecting People and the Environment

NUREG/CR-7041
ORNL/TM-2011/21

SCALE/TRITON Primer: A Primer for Light Water Reactor Lattice Physics Calculations

Manuscript Completed: July 2012
Date Published: November 2012

Prepared by:
B. J. Ade

Oak Ridge National Laboratory
Managed by UT-Battelle, LLC
Oak Ridge, TN 37831-6170

I. Frankl, NRC Project Manager

NRC Job Code V6098 (formerly N7098)

Office of Nuclear Regulatory Research

ABSTRACT

The SCALE (**S**tandardized **C**omputer **A**nalyses for **L**icensing **E**valuation) computer software system developed at Oak Ridge National Laboratory is widely used and accepted around the world for many radiation transport applications. TRITON (**T**ransport **R**igor **I**mplemented with **T**ime-dependent **O**peration for **N**eutronic depletion) is a control module developed within the framework of SCALE that enables 2-D and 3-D depletion calculations to be performed by coordinating iterative calls between cross-section processing codes, a neutron transport solver, and the ORIGEN (**O**ak **R**idge **I**sotope **GEN**eration) point-depletion code. TRITON has the capability to utilize KENO-V.a or KENO-VI 3-D Monte Carlo codes, or the NEWT (**N**ew **E**SC-Based **W**eighting **T**ransport) 2-D discrete ordinates transport code for the neutron transport solution. This primer focuses on use of the NEWT transport solver with SCALE/TRITON to generate cross sections for light water reactor nodal simulators.

In TRITON, NEWT is used to calculate weighted burnup-dependent cross sections that are employed to update ORIGEN libraries and to provide localized fluxes used for multiple depletion regions. TRITON uses a two-pass cross-section update approach to perform fuel-assembly burnup and branch calculations and generates a database of cross sections and other burnup-dependent physics data that can be used for full-core analysis. To date, cross-section data generated with TRITON/NEWT have been used in a wide variety of applications, including generation of cross sections for core calculations with PARCS (**P**urdue **A**dvanced **R**eactor **C**ore **S**imulator) for NRC analyses, for NESTLE (**N**odal **E**igenvalue, **S**teady-State, **T**ransient, **L**e Core **E**valuator) for in-house methods development research, for Bold Venture for High Flux Isotope Reactor (HFIR) low-enriched uranium conversion studies, and for Attila for Advanced Test Reactor core calculations.

The primer is based on SCALE 6.1, which includes the **G**raphically **E**nhanced **E**diting **Wiz**ard (GeeWiz) Windows user interface. Each example in this primer uses GeeWiz to provide the framework for preparing input data and viewing output results. Beginning with a *Quickstart* section, the primer gives an overview of the basic requirements for SCALE/TRITON input and allows the user to quickly run a simple pin cell simulation with SCALE/TRITON. The sections that follow the *Quickstart* include a list of basic objectives at the beginning that identifies the goal of the section and the individual SCALE/TRITON features that are covered in detail in the sample problems in that section. Upon completion of the primer, a new user should be comfortable using GeeWiz to set up 2-D lattice physics problems in SCALE/TRITON.

The primer provides a starting point for the reactor engineer who uses SCALE/TRITON for lattice physics. Complete descriptions are provided in the SCALE/TRITON manual. Although the primer is self-contained, it is intended as a companion volume to the SCALE/TRITON documentation—the SCALE manual is provided on the SCALE installation DVD. The primer provides specific examples of using SCALE/TRITON for lattice physics analyses; the SCALE/TRITON manual provides detailed information on the use of SCALE/TRITON and all its modules. The primer also contains an appendix with sample input files.

FOREWORD

The purpose of this document is to provide novice and advanced SCALE users with a guide for using the SCALE graphic user interface GeeWiz to develop lattice physics models for SCALE/TRITON-NEWT. Although the document pertains only to SCALE/TRITON-NEWT, much of the information presented herein can be used for SCALE/TRITON-KENO model development. The document is not meant to replace the SCALE user manuals, but rather to provide detailed instructions and examples for application of the information in those manuals to generate lattice physics models.

Novice users will find the step-by-step instructions helpful in getting started using SCALE. After reading this guide and completing the practice problems within, novice users should be able to develop detailed lattice physics models suitable for generating broad-group nodal parameters for reactor core simulators. Advanced users who are familiar with the SCALE code system but unfamiliar with lattice physics analysis will find the advanced features and user-guidance sections of this document helpful in generating lattice physics models.

The user guidance contained in this document provides a minimum set of "best practices" that should be used for lattice physics model development. There is no guarantee that this set of best practices will be adequate for all lattice physics problems. The minimum best practices have been developed over time modeling light water reactors. In addition, features continually evolve and modeling methodologies are refined, so the best practices are likely to change. The best method for staying up to date on the SCALE code system is to subscribe to the SCALE email notification list. Subscribers are notified by email of latest newsletters, releases, updates, and corrections. To subscribe, visit the SCALE website at http://scale.ornl.gov. Also on the SCALE website, users can download the latest updates and view past and present SCALE newsletters for relevant information regarding SCALE.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

LIST OF FIGURES

Page

LIST OF FIGURES (continued)

Page

LIST OF FIGURES (continued)

Page

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

Page

LIST OF TABLES

Page

ACKNOWLEDGMENTS

This document was written under contract with the U.S. Nuclear Regulatory Commission (NRC) Office of Nuclear Regulatory Research. The author acknowledges NRC project manager Steve Frankl and NRC staff members Peter Yarsky and Scott Krepel for their guidance and review of this document. The author thanks ORNL project manager Steve Bowman and Matthew Jessee for their reviews and direction of this work. The author is grateful for detailed review of this document by Matthew Francis and Jeffrey Powers, and final preparation of this document by Debbie Weaver.

ACRONYMS

1-D	one-dimensional
238GROUPNDF5	ENDF/B-V 238-group library
2-D	two-dimensional
3-D	three-dimensional
44GROUPNDF5	ENDF/B-V 44-group library
BWR	boiling water reactor
CSAS	Criticality Safety Analysis Sequences
CSEWG	Cross-Section Evaluation Working Group
GeeWiz	Graphically Enhanced Editing Wizard
ICSBEP	International Criticality Safety Benchmark Experiments Project
IEU	intermediate-enriched uranium
INFHOMMEDIUM	infinite homogeneous medium
k_{eff}	effective multiplication factor
LEU	low-enriched uranium
LWR	light-water reactor
MIPLIB	Material Information Processor Library
NITAWL	Nordheim Integral Treatment and Working Library Production
NRC	U.S. Nuclear Regulatory Commission
ORNL	Oak Ridge National Laboratory
PARCS	Purdue Advanced Reactor Core Simulator
PFE	Programmer's File Editor
PMAXS	Purdue Macroscopic Cross Section format
PMC	Produce Multigroup Cross sections
PWR	pressurized-water reactor
RSICC	Radiation Safety Information Computational Center
SCALE	Standardized Computer Analyses for Licensing Evaluations
SGGP	SCALE Generalized Geometry Package
V6-238GROUP	ENDF/B-VI 238-group library
V7-238GROUP	ENDF/B-VII 238-group library

1. INTRODUCTION

This primer is designed to help you understand and use the SCALE/TRITON control module for steady-state (T-NEWT) and time-dependent depletion (T-DEPL) lattice physics analyses. TRITON (**T**ransport **R**igor **I**mplemented with **T**ime-dependent **O**peration for **N**eutronic depletion) is a control module that is part of the SCALE (**S**tandardized **C**omputer **A**nalyses for **L**icensing **E**valuation) code system.[1] It assumes that you have a college education in a technical field. There is no assumption of familiarity with transport codes in general or with SCALE/TRITON in particular. The primer is designed to teach by example, with each example illustrating a couple of features that might be required to generate an accurate lattice physics model. The primer is set up so that each example builds on the previous example, adding new features each step of the way, until a fully detailed lattice model is generated at the end of the primer.

The primer is based on SCALE 6.1, which includes the SCALE **G**raphically **E**nhanced **E**diting **Wiz**ard (**GeeWiz)**, a Windows program designed to assist reactor engineers in creating and executing SCALE input files for TRITON. GeeWiz version 6.0.13.04 (January 4, 2010) and TRITON version 6.1 (2011) were used for all the examples shown in this primer. Each of the examples uses GeeWiz to provide the framework for data input. Starting with a *Quickstart* section, the primer gives an overview of the basic requirements for SCALE/TRITON input and allows you to quickly run a fuel pin cell problem with SCALE/TRITON. This section is not designed to explain GeeWiz, the input, or the SCALE/TRITON options in detail; rather, it introduces the GeeWiz user interface and some basic concepts that are further explained in following sections. Each following section has a list of basic objectives at the beginning that identifies the goal of the section and the individual SCALE/TRITON features, which are then covered in detail in the example problems in that section. It is expected that on completion of the primer, you will be comfortable using GeeWiz to set up lattice physics problems in SCALE/TRITON.

In this document, "lattice physics" means modeling of nuclear reactor fuel lattices to generate data for a reactor core simulator. The methods that are used to analyze nuclear fuel lattices are accurate, high-order methods. A nuclear reactor is simply too large to be modeled using these high-fidelity methods—it would take too much computer time and too much memory to generate a solution on a modern personal computer. The fastest supercomputers in the world are only now entering the era in which it will be possible to perform a full-core or near full-core simulation with high-order deterministic methods. Due to the computational limitations, lattice physics modeling if performed for nuclear reactor analysis.

In lattice physics analysis, an analyst begins with basic nuclear reactor fuel design information. This includes all necessary dimensions (fuel pin and clad radii, pin and lattice pitch, etc.), material information (exact isotopic compositions, density), and thermal-hydraulic conditions (temperature, void fraction, density) of the fuel and moderator/coolant. This information is usually specified in the design documentation, and as few assumptions as possible should be made in collecting the needed information. From this design information a user can generate a lattice model. For a large nuclear reactor, there may be a number of different types of lattices with axial variation. A separate two-dimensional (2-D) lattice model will be required for each axial level of each lattice. Using these lattice models, the needed data for a reactor core simulator can be generated.

Reactor core simulators are typically based on "nodal diffusion theory." Nodal diffusion theory provides a very fast and reasonably accurate solution, provided that the data produced for the simulation are accurate and generated from the actual reactor design. Some nodal simulators are PARCS,[2] NESTLE[3] and SIMULATE5.[4] Nodal simulators require broad-group cross-section data, scattering matrices, and assembly discontinuity factors that are generated by lattice physics codes such as SCALE/TRITON. Using the homogenized lattice physics data, the core simulator models an entire fuel assembly (or a

quarter assembly) as one lump with properties that have been conserved by the lattice physics code. These nodes are virtually assembled by the nodal simulator to model a full 3-D reactor core. The core simulator can typically be used to perform reactor operation and safety analyses, including transients required for licensing.

After completing this primer, users will be able to use SCALE/TRITON in lattice physics calculations and will be capable of using SCALE/TRITON to generate cross sections for nodal simulators. In addition, Appendix D presents the basics of using GenPMAXS (Generation of Purdue MAcroscopic XS set) to convert SCALE/TRITON cross section data to PMAXS (Purdue MAcroscopic XS set) format for using in PARCS (Purdue Advanced Reactor Core Simulator) core simulator.

Although much of the information needed to perform an analysis is provided in the primer, there is no substitute for understanding your problem and the theory of neutron interactions. The SCALE/TRITON code is only capable of analyzing the problem as it is specified; it will not necessarily identify inaccurate modeling of the geometry, nor will it know when the wrong material has been specified. However, the 2-D color geometry plots and the input file checking functionality of SCALE/TRITON are quite useful for identifying geometry errors. Remember that a single calculation of k_{eff} and associated cross-section data produced with SCALE/TRITON or any other code is meaningless without an understanding of the context of the problem, the quality of the solution, and a reasonable idea of what the result should be.

The primer provides a starting point for the reactor engineer using SCALE/TRITON. Complete descriptions are provided in the SCALE manual[1]. Although the primer is self-contained, it is intended as a companion volume to the SCALE manual. (The SCALE manual is provided on the SCALE installation DVD.) The primer provides specific examples using SCALE/TRITON for lattice physics analyses, while the documentation provides information on the use of SCALE and all its modules.

To make the primer easy to use, there is a standard set of notations that you need to know. The text of this primer is set in Times New Roman font. Information that you type into an input file (or provide to GeeWiz) is set in `Courier font`. Characters in the `Courier font` represent commands, keywords, or data that would be used as computer input. References to items displayed on the screen by GeeWiz are highlighted in **bold font**. The primer often references the SCALE manual; these references are set in square brackets; e.g., [SCALE manual, Section x].

It is hoped that you find the primer useful and easy to read. You will get the most benefit from this tutorial if you start with **Section 2: SCALE/TRITON *Quickstart*** and proceed through the rest of the sections in order. Each section assumes that you know and are comfortable with the concepts discussed in the previous sections. Although it may be tempting to pick up the primer and immediately go to a sample problem that is similar to your analysis requirement, this approach will not provide you with the background or confidence in your analysis that is necessary for accurate and effective implementation of procedures and limits. There is no substitute for a thorough understanding of the techniques used in a SCALE/TRITON analysis. A little extra time spent going through the primer and working through the examples will save many hours of possible confusion and frustration later.

1.1 OVERVIEW OF SCALE

The SCALE code system dates back to 1969, when the current Reactor and Nuclear Systems Division at Oak Ridge National Laboratory (ORNL) began providing computational support in the use of the new KENO code to the U.S. Atomic Energy Commission (AEC) staff. From 1969 to 1976 the AEC certification staff relied on the ORNL staff to assist them in the correct use of codes and data for criticality, shielding, and heat transfer analyses of transportation packages. Shortly after creation of the U.S. Nuclear Regulatory Commission (NRC) from the AEC, the NRC staff proposed the development of

an easy-to-use analysis system that would provide the technical capabilities of the individual modules with which they were familiar. With this proposal, the concept of the SCALE code system was born.

The NRC staff provided ORNL with some general development criteria for SCALE: (1) focus on applications related to nuclear fuel facilities and package designs, (2) use well-established computer codes and data libraries, (3) design an input format for the occasional or novice user, (4) prepare "standard" analysis sequences (control modules) that will automate the use of multiple codes (functional modules) and data to perform a system analysis, and (5) provide complete documentation and public availability. The initial version of SCALE was released by the Radiation Safety Information Computational Center (RSICC) at ORNL in 1980. Since that time, the system has been considerably enhanced, and the release used in this primer is Version 6.1. SCALE runs on Unix, Windows, Linux, and Intel Mac computer platforms.

The concept of SCALE was to provide "standardized" sequences. Input for the control modules has been designed to be free-form with extensive use of keywords and engineering-type input requirements. The most important feature of the SCALE system is the capability to simplify the user knowledge and effort required to prepare material mixtures and to perform adequate problem-dependent cross-section processing.

At the center of SCALE/TRITON is the library of subroutines referred to as the **M**aterial **I**nformation **P**rocessor **Lib**rary or MIPLIB. The purpose of MIPLIB is to allow users to specify problem materials using easily remembered and easily recognizable keywords that are associated with mixtures, elements, and nuclides provided in the Standard Composition Library. MIPLIB also uses other keywords and simple geometry input specifications to prepare input for the modules that perform the problem-dependent multigroup cross-section processing: BONAMI, NITAWL, CENTRM, PMC, and XSDRNPM. A keyword supplied by the user selects the cross-section library from a standard set provided in SCALE or designates the reference to a user-supplied library.

The modular structure of SCALE allows back-to-back execution of the functional modules to perform a system analysis. However, a variety of control modules such as TRITON have been developed that automate and standardize various analytic sequences. The input format of the control module has been designed to help minimize input errors. Upon processing the user-specified input, the SCALE system control modules immediately print an input checklist in which the user (or reviewer) can easily establish that the input describes the system to be analyzed.

The T-NEWT control module performs problem-dependent cross-section processing and 2-D discrete ordinates transport using NEWT. T-DEPL is the 2-D multi-material depletion sequence using burnup-dependent cross-section preparation, 2-D discrete ordinates transport solution by NEWT, and depletion of user-specified materials by ORIGEN.

BONAMI performs resonance shielding through the application of the Bondarenko shielding factor method. BONAMI is typically used to process data in the unresolved resonance energy range. As input, BONAMI requires the presence of shielding factor data on the AMPX master library interface. As output, BONAMI produces a problem-dependent master library.

NITAWL uses the Nordheim Integral Treatment to perform neutron cross-section processing in the resolved resonance energy range. This involves a fine-energy-group calculation of the slowing-down flux across each resonance, with subsequent flux weighting of the resonance cross sections. NITAWL also assembles group-to-group transfer arrays from the elastic and inelastic scattering components and performs other tasks in producing the problem-dependent working library. NITAWL is compatible only with ENDF/B-V cross-section libraries in SCALE.

CENTRM computes continuous-energy neutron spectra using one-dimensional (1-D) discrete ordinates or infinite media geometry. CENTRM determines the problem-specific fluxes for processing resonance-shielded multigroup data using a 1-D unit cell. PMC reads the CENTRM continuous-energy flux spectra and cross-section data; calculates problem-dependent, group-averaged cross sections over some specified energy range; and then replaces the corresponding data in a problem-dependent AMPX master library for use by NEWT (or another transport solver).

NEWT is a 2-D discrete-ordinates transport code developed based on the Extended Step Characteristic (ESC) approach for spatial discretization on an arbitrary mesh structure. This discretization scheme makes NEWT an extremely powerful and versatile tool for deterministic calculations in real-world non-orthogonal problem domains. NEWT uses AMPX-formatted cross sections processed by the SCALE functional modules described above. If cross sections are properly prepared, NEWT can be run in standalone mode. However, the T-NEWT sequence within the TRITON control module of SCALE can also be used to automatically prepare cross sections and then execute NEWT for a transport solution.

NEWT uses the same nomenclature as the SCALE Generalized Geometry Package (SGGP) but has a subset of body shapes available for users corresponding to 2-D shapes. The available body shapes are

- cuboid,
- cylinder,
- hexprism,
- rhexprism (rotated hexprism),
- wedge.

The names of these shapes are generally associated with 3-D bodies but are used in NEWT to be consistent with KENO-VI nomenclature. This consistent naming convention enables easy conversion between NEWT and KENO-VI geometry input files. In NEWT, a cuboid is equivalent to a rectangle, a cylinder is equivalent to a circle, a hexprism is equivalent to a hexagon, and a wedge is equivalent to a triangle.

As this primer is primarily concerned with light water reactor (LWR) lattice physics analysis, it will focus on cylinders and cuboids to model LWR fuel lattices. Of particular interest in lattice physics analysis is the ability to define an array of fuel pins. This primer focuses on rectangular arrays, but in addition to rectangular arrays, NEWT provides three hexagonal array types. Users wishing to construct hexagonal-based geometries should refer to the SCALE users manual for further guidance.

The SCALE code system includes several problem-independent multigroup cross-section libraries for lattice physics analyses. The most up-to-date library is the 238-group ENDF/B-VII neutron library, which is recommended for lattice physics calculations. The 238-group ENDF/B-VI and ENDF/B-V libraries are also available in SCALE 6.1. The 44-group ENDF/B-V library was generated by collapsing the 238-group ENDF/B-V library with a typical LWR neutron spectrum. Using the 44-group ENDF/B-V library is a factor of approximately 5 times faster than the 238-group libraries, but it is only recommended for scoping calculations or test cases. Production runs should use the 238-group ENDF/B-VII library with the `parm=weight` option enabled (to be discussed later).

1.2 LATTICE PHYSICS MODEL DEVELOPMENT STRATEGY

When developing lattice physics models using SCALE/TRITON, there is a certain recommended strategy that can be used. This strategy is a general guideline that new users can follow during the development of lattice physics models. The model development strategy presented herein is not meant to be a rigid structure, rather it is a framework that you can use if you are unsure on how to begin model development or proceed from a certain step in the process. You might find a process other than the one presented here that is more effective for your application.

The recommended strategy begins with gathering the required information. The lattice design information is normally contained in one or more documents. You will need to work through the documentation and highlight certain items that will be needed for model development. After you have gathered all the relevant information from the documentation, you should find any shortcomings in the data. If there is a gap in the information, you will need to request that information or make an assumption about the unavailable information. If you must make assumptions, be sure to justify them. Once you have gathered all the information needed, you can begin building the SCALE/TRITON model. Building the SCALE/TRITON model has many steps that will be explained later in further detail. After the models have been completed, the model developer should have the models reviewed by a technical reviewer. It is much easier for a technical reviewer to find simple mistakes that the model developer is likely to overlook. The reviewer and the model developer can then iterate between the development and the review steps until the models reach a point where both the developer and the reviewer are satisfied. The developer can then run the models. While running lattice physics models, it is possible to check the data as it is generated using the *txtfile16* file that is in the SCALE temporary directory. Users can open and inspect the *txtfile16* file and look for eigenvalues and other data. By checking the data as it is generated, it is possible to spot errors before the entire calculation is complete. Once the models have completed running, the model developer should check or verify that the data are correct. If required by the project, the data can then be converted to other formats and tested. This process can be more easily visualized with the model development flowchart in Fig. 1.1.

The main focus of this primer is to show the user how to build lattice physics models. Each step of the model development strategy has corresponding sub-steps that must be taken. Because this primer focuses on the "build the model" portion of the development strategy in Fig. 1.1, the "build the model" step is expanded into required sub steps in Fig. 1.2. Each step contains a reference to the section number in the primer where one can find information on the subject matter. Building lattice physics models generally follows a top-down strategy. While building the models you should use comments often. You can use these comments to reference the location from which you obtained data so that a technical reviewer will be able to easily follow your work. You should also state all assumptions in the comments with a short justification. You should first choose the sequence, cross-section library, and runtime options. You can then insert all the mixtures that will be needed in your model. You should then add the appropriate cross-section processing options. After entering the cross-section processing information, add the SCALE/NEWT Parameter and Materials blocks. You should then build the fuel pin units, and then build the array that contains each fuel pin. At this point, you should run the model with Parm=check to plot the model and check for errors. Closely inspect the plots of the geometry; mistakes can often be easily identified on the geometry plots. If there are no errors, add more complex geometry features and plot the geometry again. Once the geometry is complete, add advanced lattice physics options such as depletion, branches, collapse, homogenize, and assembly discontinuity factors. If you are modeling a BWR lattice, you will need special Dancoff factors for corner and edge fuel pins. The following sections of the primer illustrate in detail how to do each of these steps.

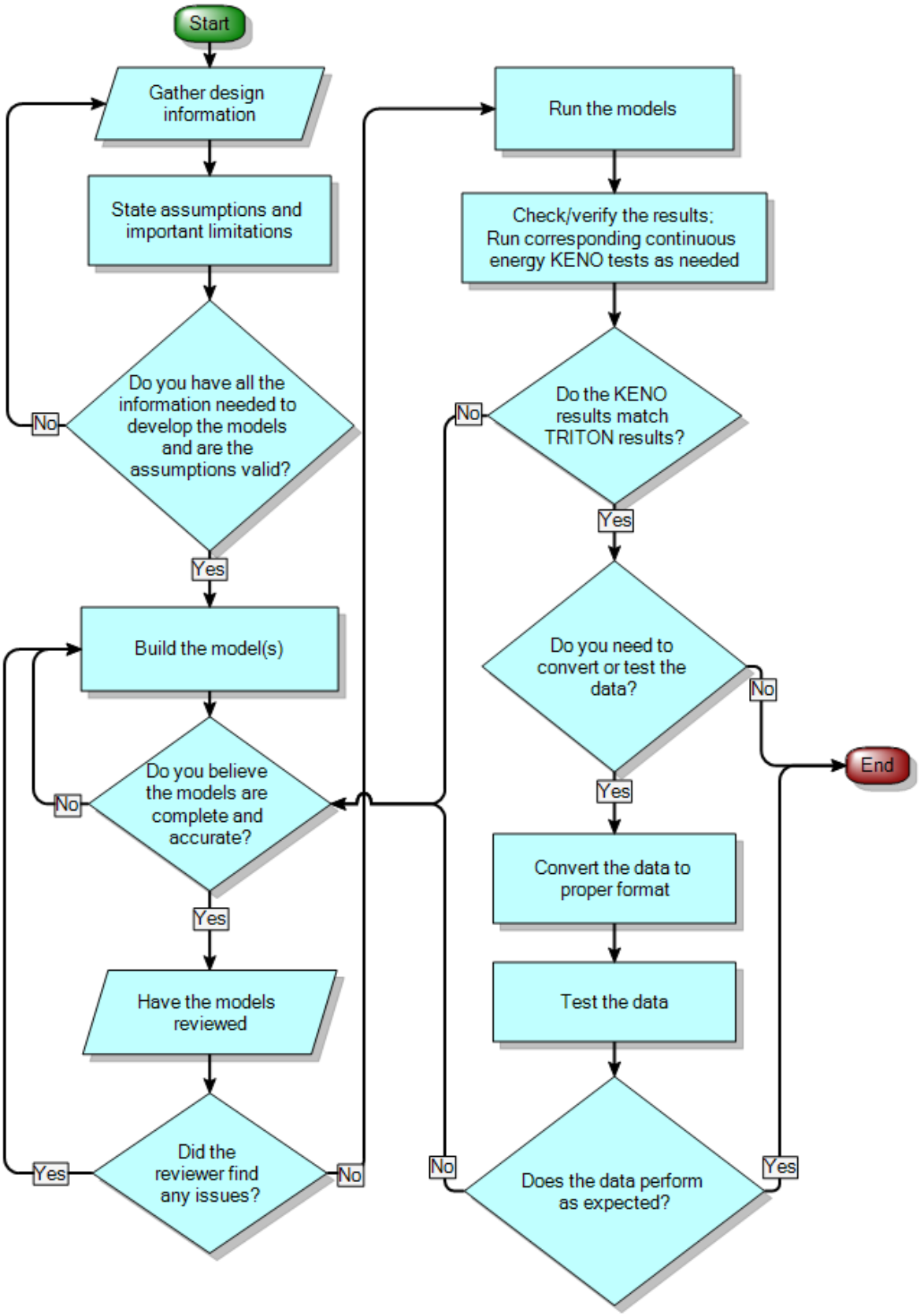


Fig. 1.1. Model development strategy flowchart.

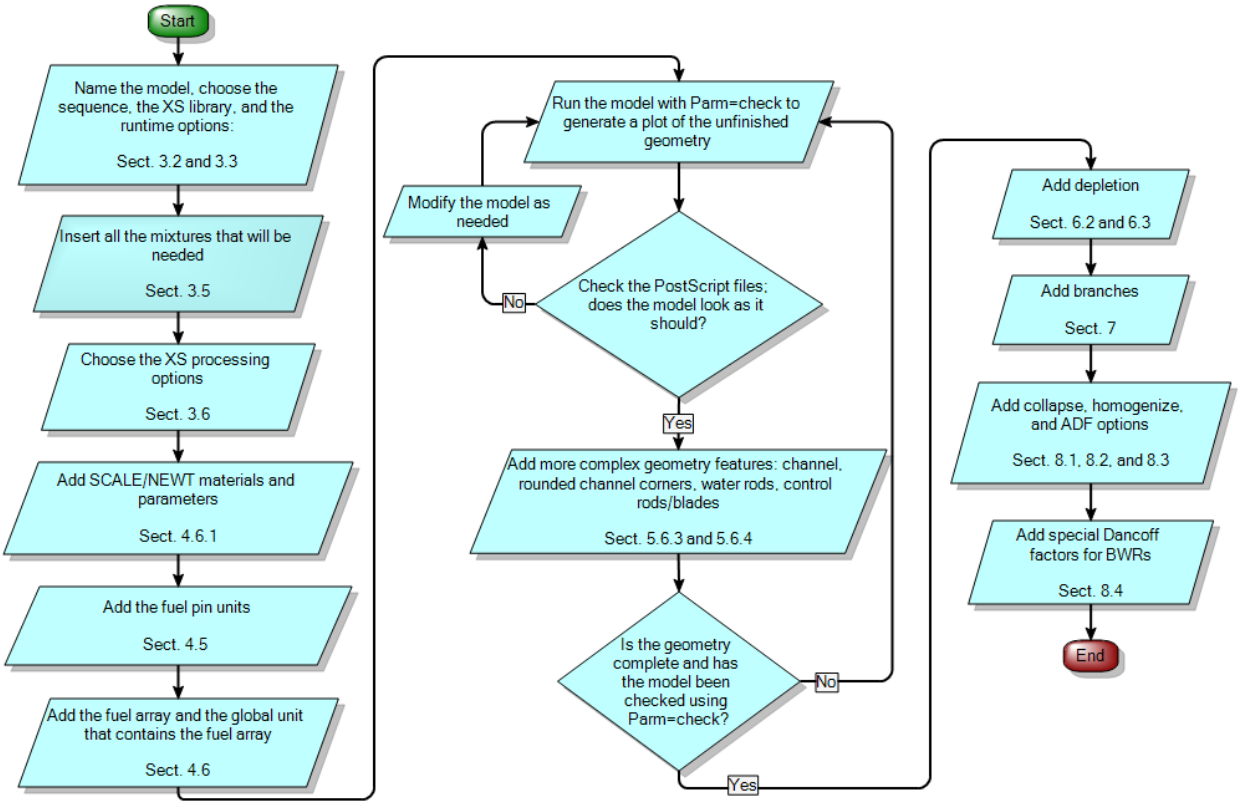


Fig. 1.2. Building the model flowchart.

2. SCALE/TRITON *QUICKSTART*

2.1 WHAT YOU WILL BE ABLE TO DO

- describe the structure of SCALE/TRITON input files
- use the GeeWiz user interface to create a SCALE/TRITON input file
- set up and run a simple criticality problem using SCALE/TRITON
- find and interpret calculated k_{eff} information from SCALE/TRITON output

2.2 SCALE/TRITON INPUT FILE

The SCALE/TRITON input file describes the problem geometry, specifies the materials, and defines the control parameters for analyzing the problem. The geometry is constructed by defining objects and their relationship with other objects in a system. Each object can be filled with a material or a void.

A SCALE/TRITON input file consists of some or all of the above data, depending on the type of problem being analyzed and the amount and type of output desired. The most user-friendly method for entering the data is to use the GeeWiz user interface.

2.3 SIMPLE SAMPLE PROBLEM – LWR PIN CELL

This section should provide enough information to run a simple sample problem. It is intended that users immediately gain confidence in using GeeWiz to enter SCALE/TRITON input data so that one can walk through this sample problem step-by-step, explaining each input step. For the present, it is important to enter this problem exactly as it is described. As more experience is gained with GeeWiz and SCALE/TRITON, users may find other ways to set up input files that are more logical to them. For example, users may find alternate methods for the geometry setup.

2.3.1 Problem Description

This problem is a simple fuel pin cell—it contains only fuel, clad, and a water moderator. Experimental parameters are as follows:

Table 2.1. *Quickstart* basic description

Region	Radius/Pitch	Material	Density
Fuel	0.5	3.5% enriched UO_2	97% TD
Clad	0.6	Zircaloy-4	6.56
Moderator	1.4	H_2O	0.9982

2.3.2 GeeWiz Input – General Information

Now begin entering the sample problem. First start the GeeWiz user interface. The initial screen should looks like Fig. 2.1.

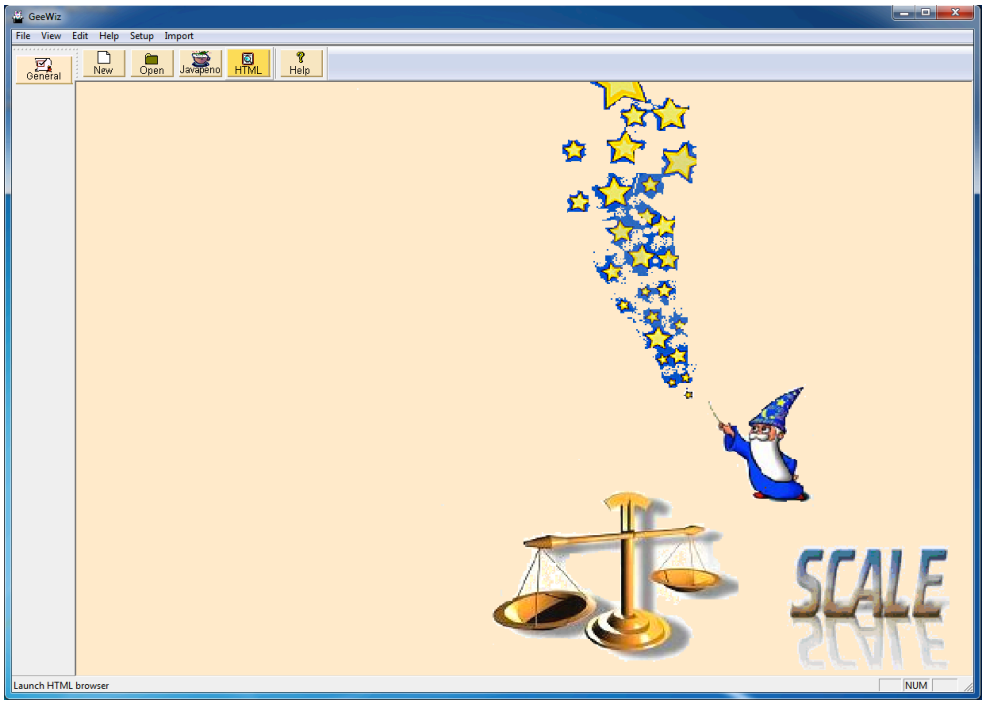


Fig. 2.1. GeeWiz start screen.

Clicking on the **General** button on the left-hand-side toolbar will bring up a window as shown in Fig. 2.2. In this window, enter the **Title**, the SCALE **Application** and **Sequence**, the program to **Use for cross-section processing**, and the **Cross-section Library**. Also included are optional batch arguments—these options will not used for now.

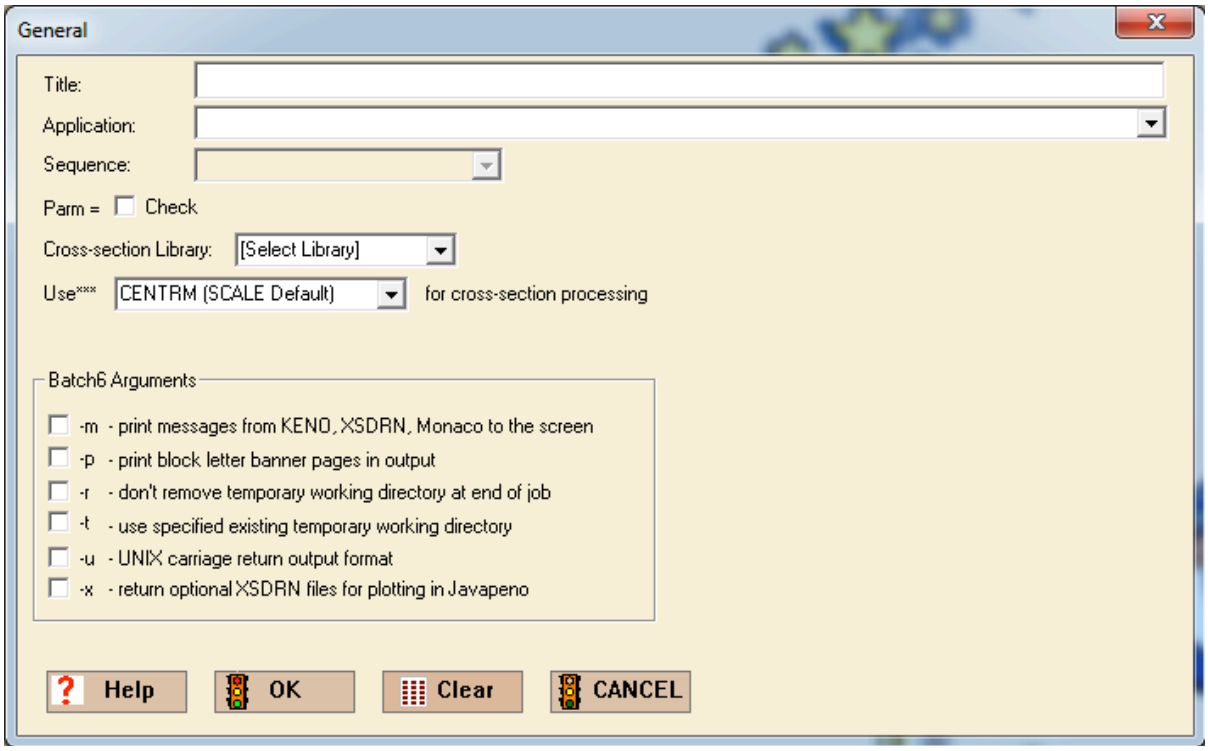


Fig. 2.2. General form.

Enter the **Title** as follows:

```
Unit cell 1
```

Again note that information to be entered into GeeWiz will appear in `Courier font`. For **Application** click on the down arrow and select `TRITON`. Then for **Sequence**, select `T-NEWT` to select a steady-state transport solution. Click the `Parm=Check` box to request that TRITON perform an input check only, i.e., no calculations will be performed. This option is recommended when initially setting up a problem and should be used for each problem in this primer. Select `v5-44` for **Cross-section Library** to specify use of the 44-group ENDF/B-V cross-section library. Select `CENTRM (SCALE Default)` **for cross-section processing**. The **General** screen should look like Fig. 2.3.

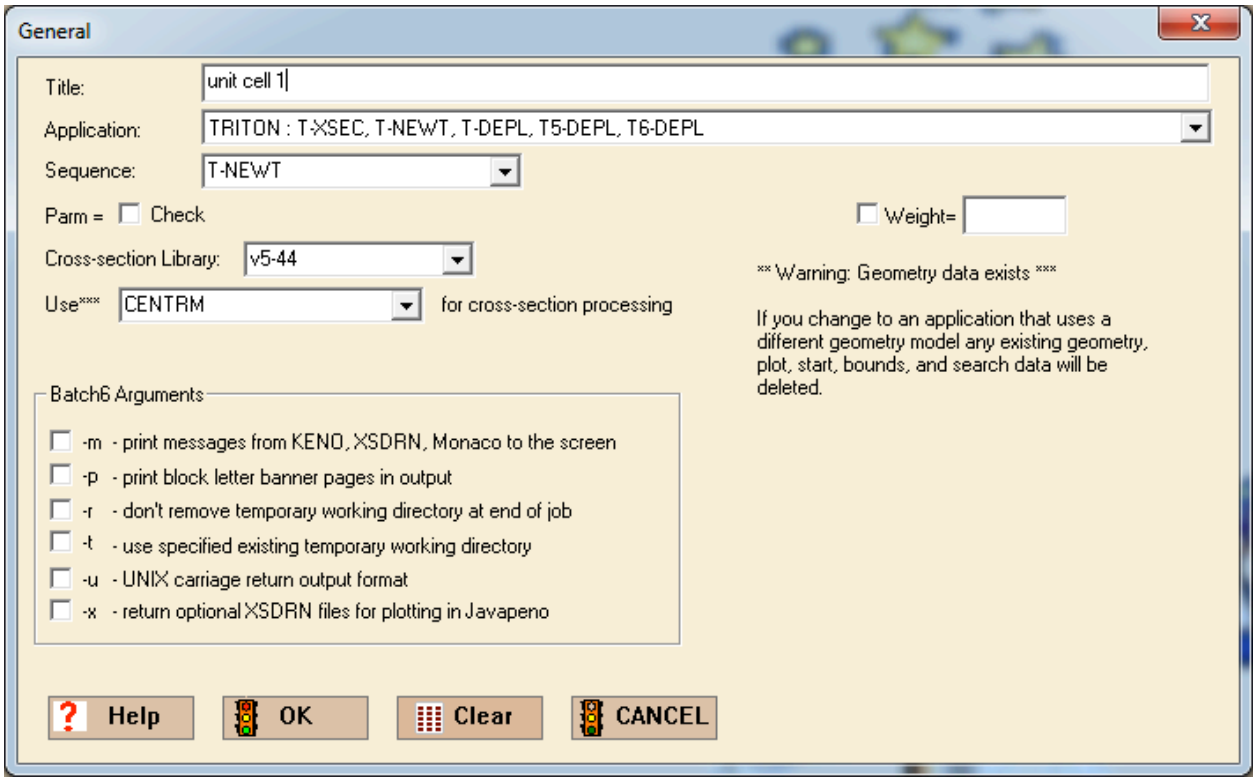


Fig. 2.3. General information for simple sample problem.

Now select **OK** to save this information in a temporary file. Later, all input information will be saved under a specific file name.

2.3.3 Materials

The next section of GeeWiz input provides information on the materials in the problem. For the sample problem, there are three materials in the unit cell description. To enter these, select **Compositions** from the menu on the left-hand side of GeeWiz and then click the **Create** button on toolbar. This button opens the form for specifying new materials in the model, as shown in Fig. 2.4.

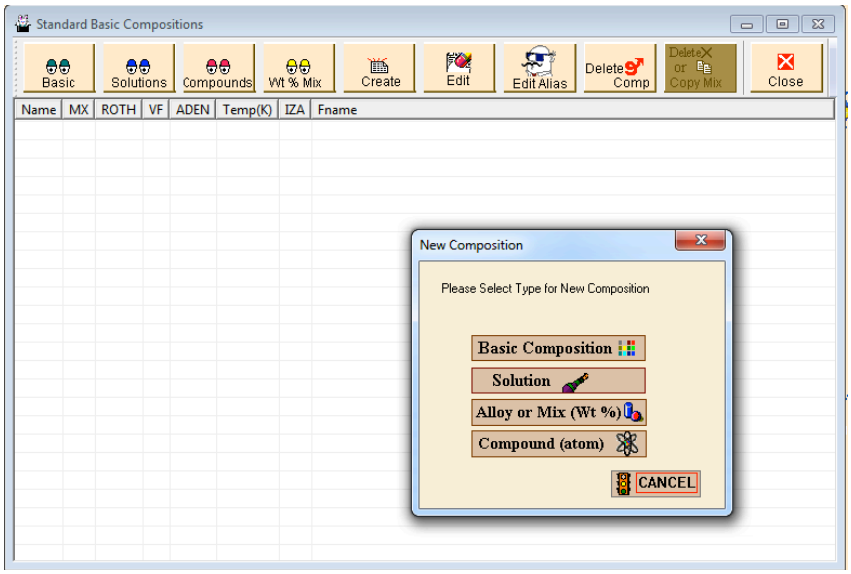


Fig. 2.4. New composition window.

You will be entering data for UO_2, Zircaloy-4, and H_2O, all of which are in the Standard Compositions Library and are Basic Compositions. Selecting the **Basic Composition** button will bring up the **Basic Standard Composition** form shown in Fig. 2.5.

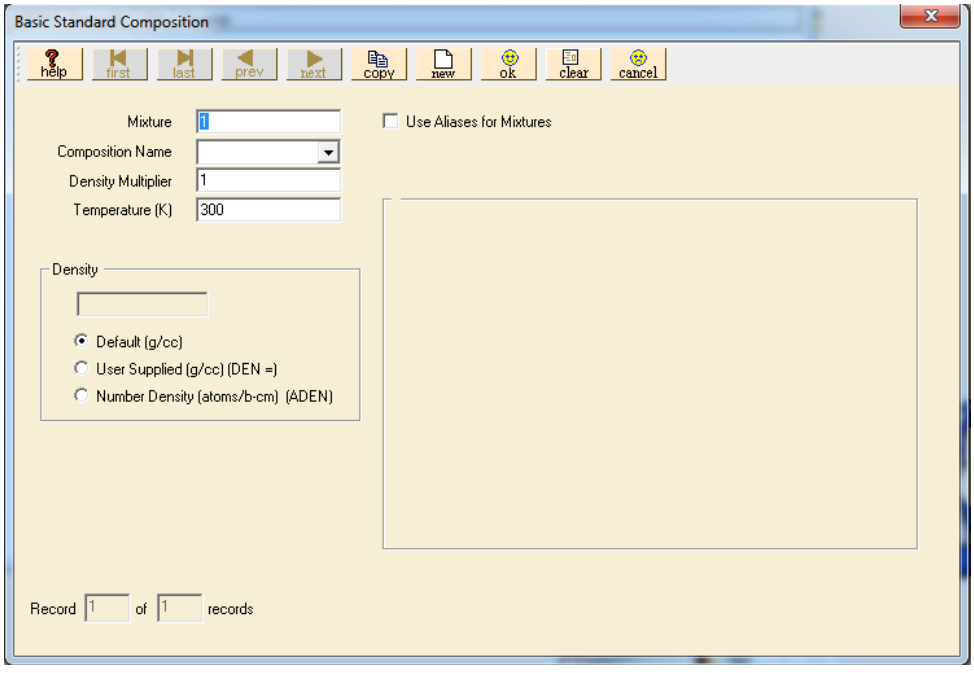


Fig. 2.5. Basic Standard Composition window.

The entry for the first material, UO_2, should look like Fig. 2.6. The **Mixture** number should already be set to `1`. Select `uo2` as the **Composition Name** from the list of materials in the library. Set the **Temperature** to `300` K. To enter the density given in the sample problem description (97% theoretical density), you need to specify `0.97` for the **Density Multiplier**.

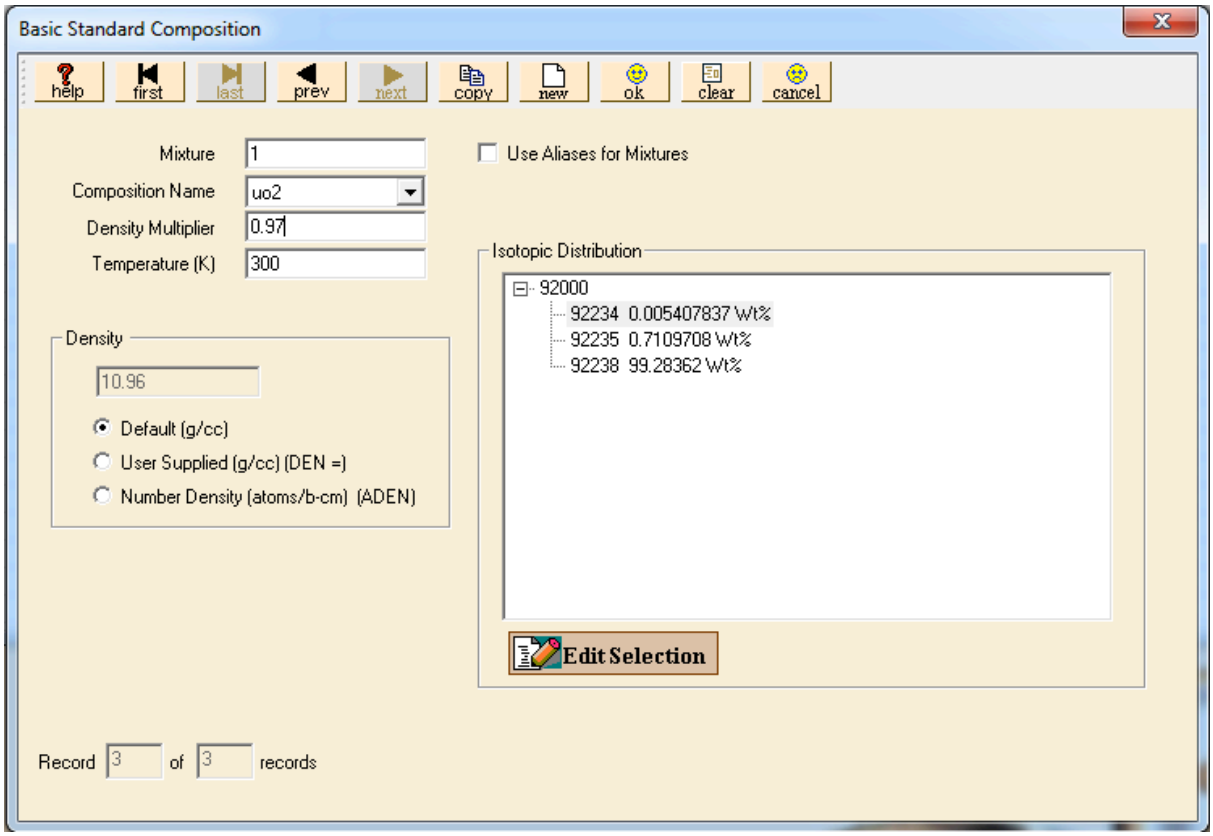


Fig. 2.6. Mixture 1 input data (before entry of isotopic distribution).

To enter the fuel enrichment, select **Edit Selection** under **Isotropic Distribution**; which will bring up an **Isotropic Distribution** window in which the user can input specific isotopics. Click inside the **Wgt Percent** box for **92235** (uranium-235) and enter `3.5` for 3.5 wt% enrichment.

Next, click inside the **Wgt. Percent** box for **92238** so that the cursor is blinking inside the box, then click the **Fill to 100%** button. This action will set the weight percent for U-238 to the fraction needed to obtain 100%. The final isotopic distribution for the UO_2 can be found in Fig. 2.7. Click **OK** to return to the input form for mixture 1.

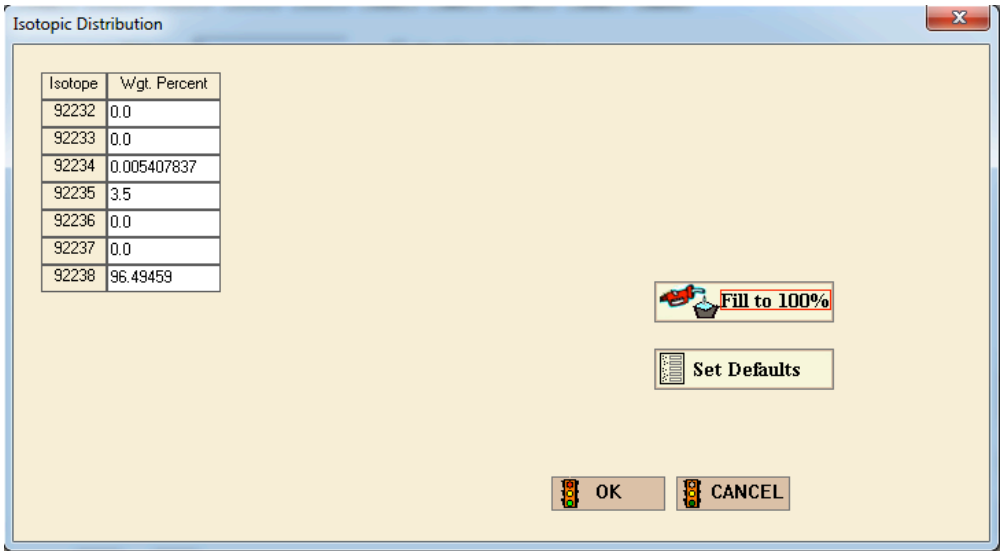


Fig. 2.7. Isotopic distribution for mixture 1.

The final Basic Standard Composition form for mixture 1 should look like Fig. 2.8.

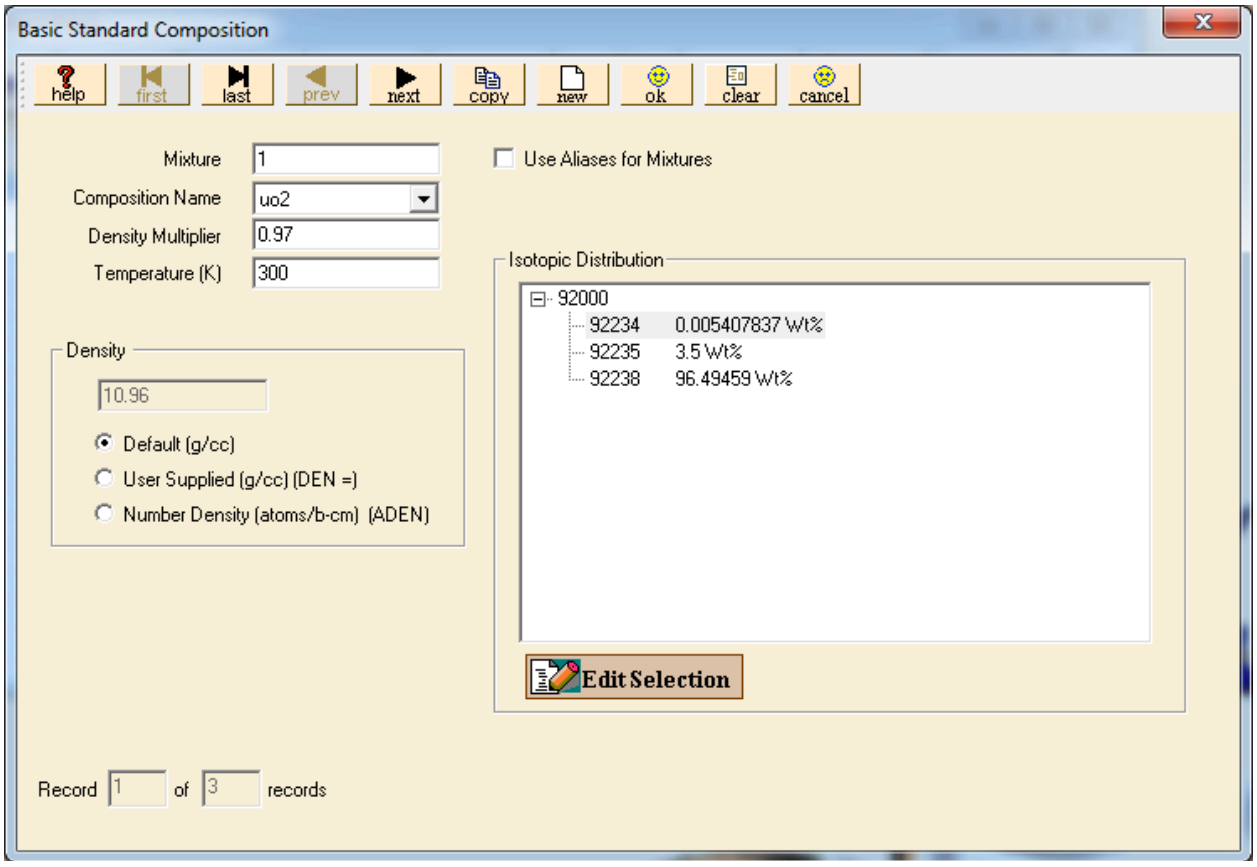


Fig. 2.8. Mixture 1 input data for UO_2 (including isotopic distribution).

Click the **OK** button on this form to save the current composition, then click **Create** on the **Standard Basic Compositions** form to open a new form where you can enter input data for Zircaloy-4 in a similar fashion; use the default SCALE density. The SCALE basic standard composition name is `zirc4`. The composition input data should look like Fig. 2.9.

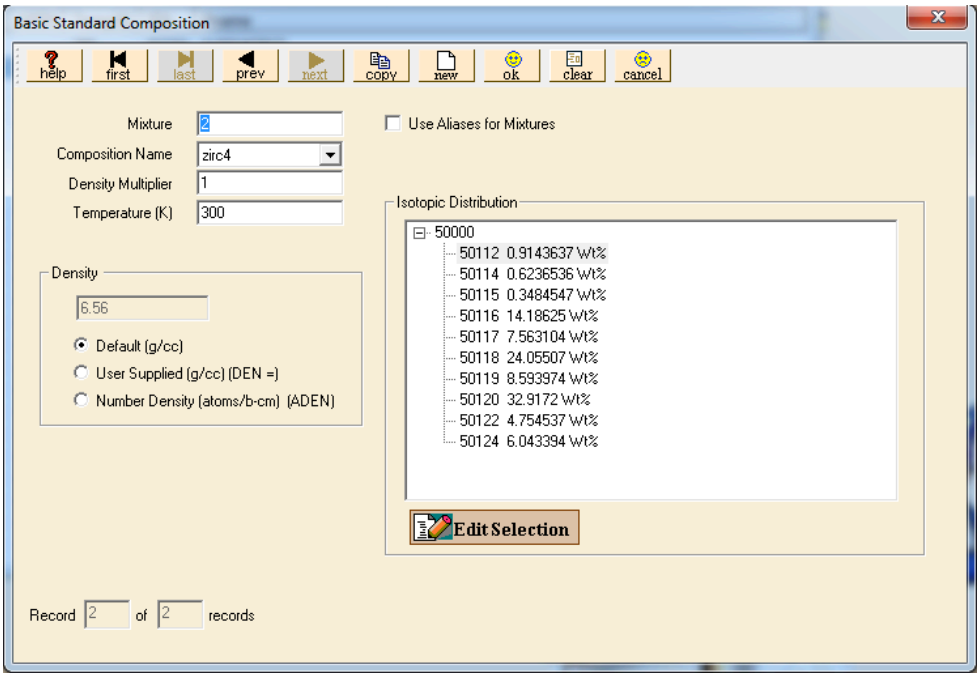


Fig. 2.9. Mixture 2 input data for Zircaloy-4.

Click the **OK** button to save the composition, the click Create on the **Standard Basic Compositions** form to open a new form where you can enter input data for H_2O in a similar fashion; use the default SCALE density. Your composition input data should look like Fig. 2.10. Click **OK** to complete this entry.

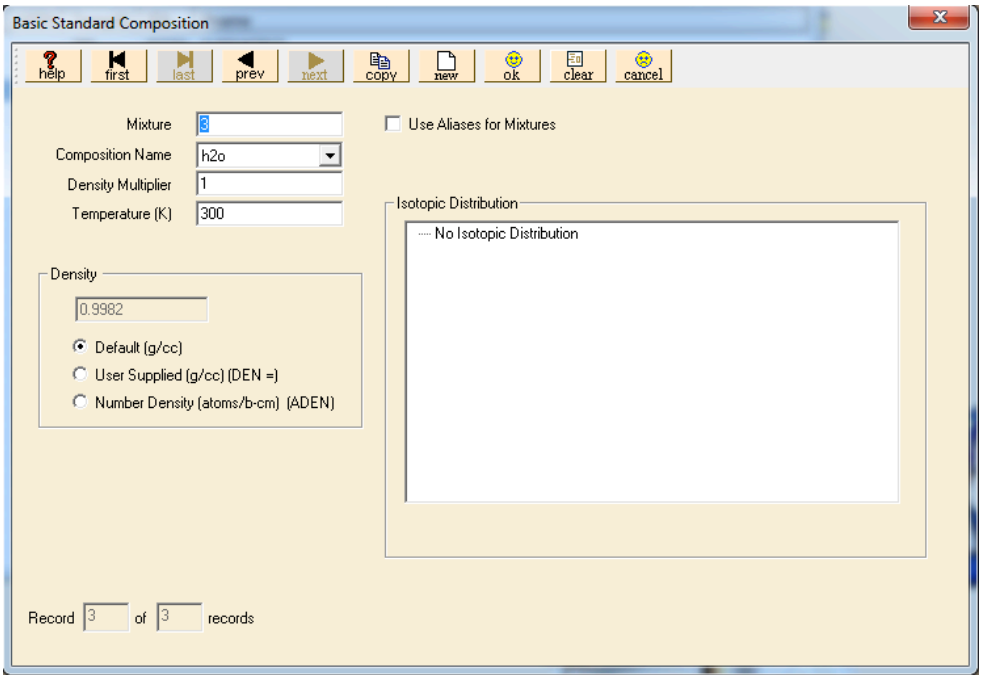


Fig. 2.10. Mixture 3 input data for H_2O.

GeeWiz should now display the **Standard Basic Compositions** summary screen as shown in Fig. 2.11. On this screen is the information for each of the three materials, including the name, the mixture number, the density multiplier (VF), and the temperature. Clicking on the + button to the left of `uo2` will display the complete isotopic distribution for UO_2. After confirming that the composition summary screen looks like Fig. 2.11, select **Close**.

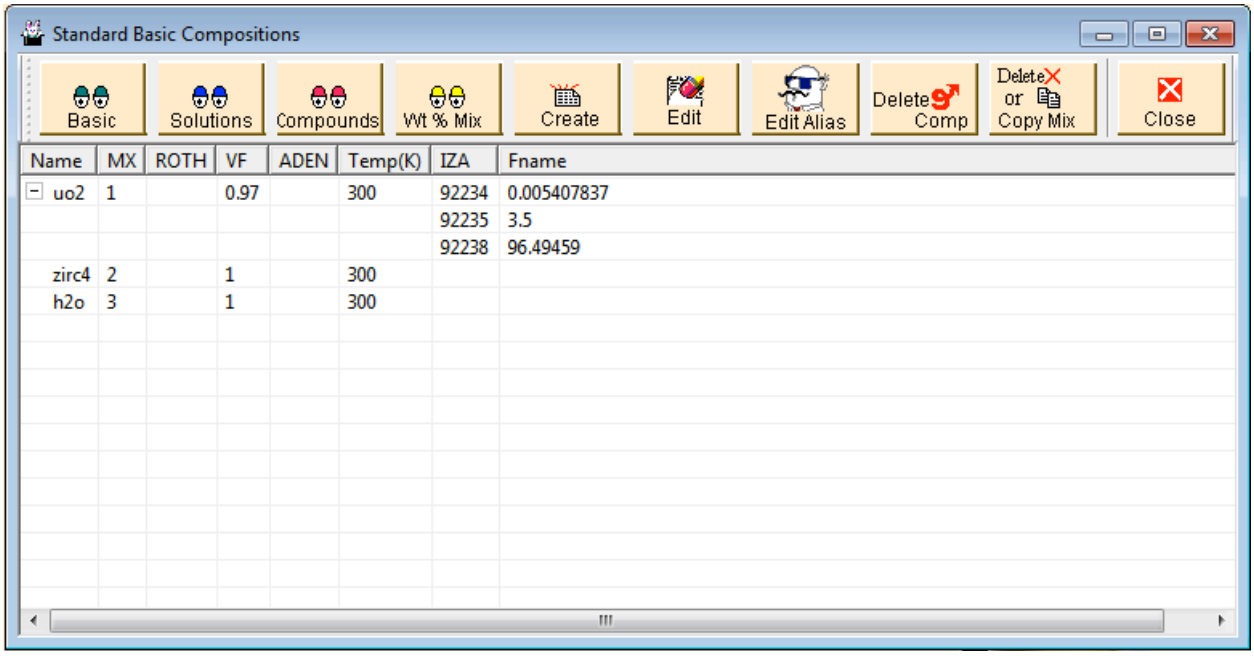


Fig. 2.11. Standard basic compositions summary.

2.3.4 Cell Data

The next information to be entered is for the unit cell. The unit cell data are used to create the problem-dependent cross-section data for NEWT. To enter this information, select **Cell Data** from the toolbar on the left-hand side of GeeWiz to open the **Unit Cell Data** screen in Fig. 2.12. Under the **Unit Cell Data** window, select the **Lattice Cell** tab along the top of the window and then select **New Cell** from the top toolbar.

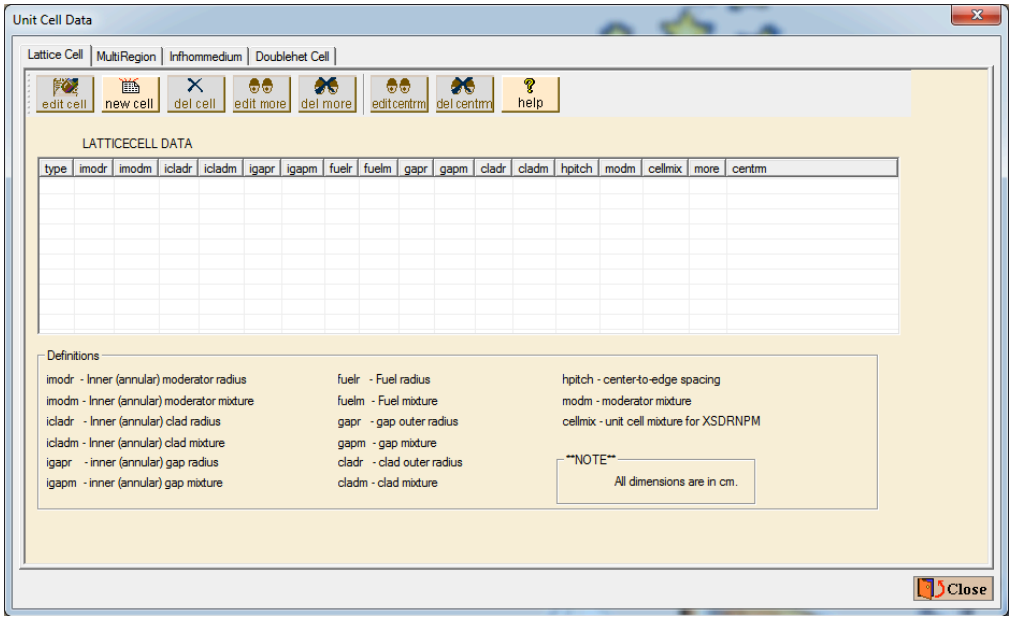


Fig. 2.12. Unit cell data screen.

This should open the **Lattice Cell Data** window (Fig. 2.13). In this sample problem, a square-pitched lattice is being modeled, so select **SquarePitch** from the **Type of Lattice** dropdown. Then specify the half pitch (P=1.4 cm), in this case 1.4/2 = 0.7 cm. Input `0.7` cm into the **Half Pitch (cm)** box. Now input the fuel radius (`0.5` cm) in the **Fuel Radius (cm)** box. Select mixture `1 uo2` for the **Fuel Mixture** and mixture `3 h2o` for the **Moderator Mixture**. Because this problem does have clad, check the **Clad/Gap Exists** box. Selecting the **Clad/Gap Exists** box displays additional options that will be used. In this section, specify the **Clad Radius (cm)** of `0.6` cm and select `2 zirc4` from the **Clad Mixture** dropdown. For this sample problem, there is no gap, so leave this section blank. The **CELLMIX** option will not be discussed in this primer, so it can also be ignored. The final **Lattice Cell Data** window should look like Fig. 2.13.

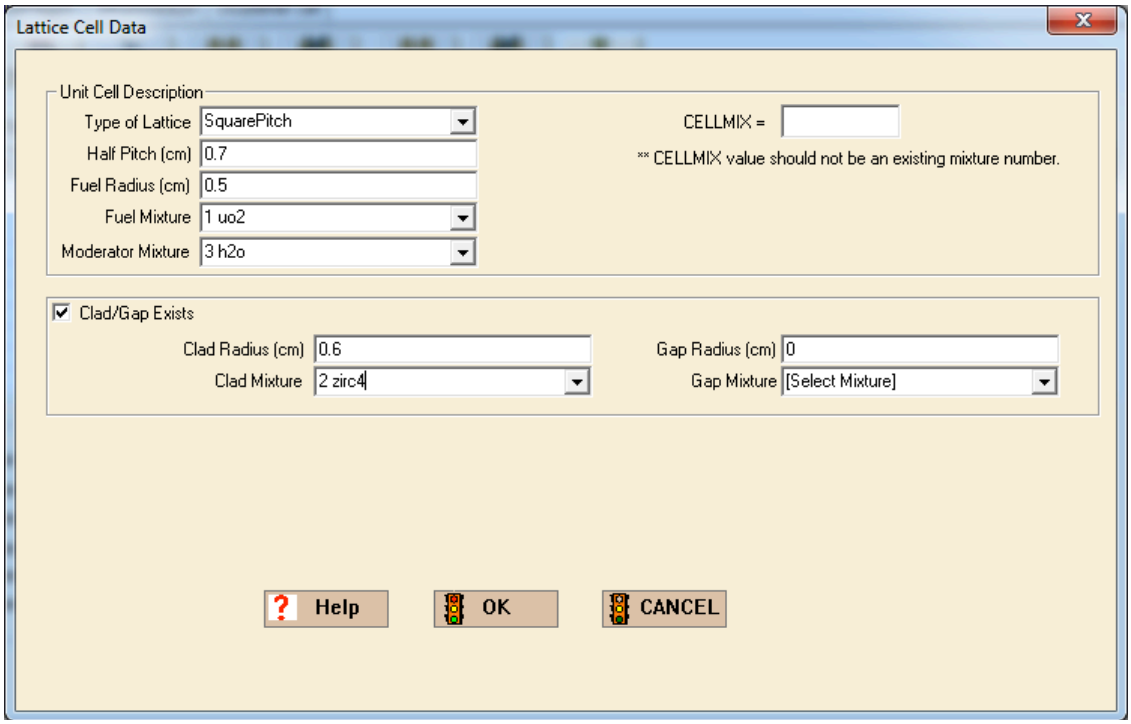


Fig. 2.13. Lattice cell data form.

Once the **Lattice Cell Data** window is identical to Fig. 2.13, click **OK**. In the **Unit Cell Data** summary window, a summary of the lattice cells that are being used in the problem can be found, as shown in Fig. 2.14. If your input matches Fig. 2.14, click **Close** on the **Unit Cell Data** window.

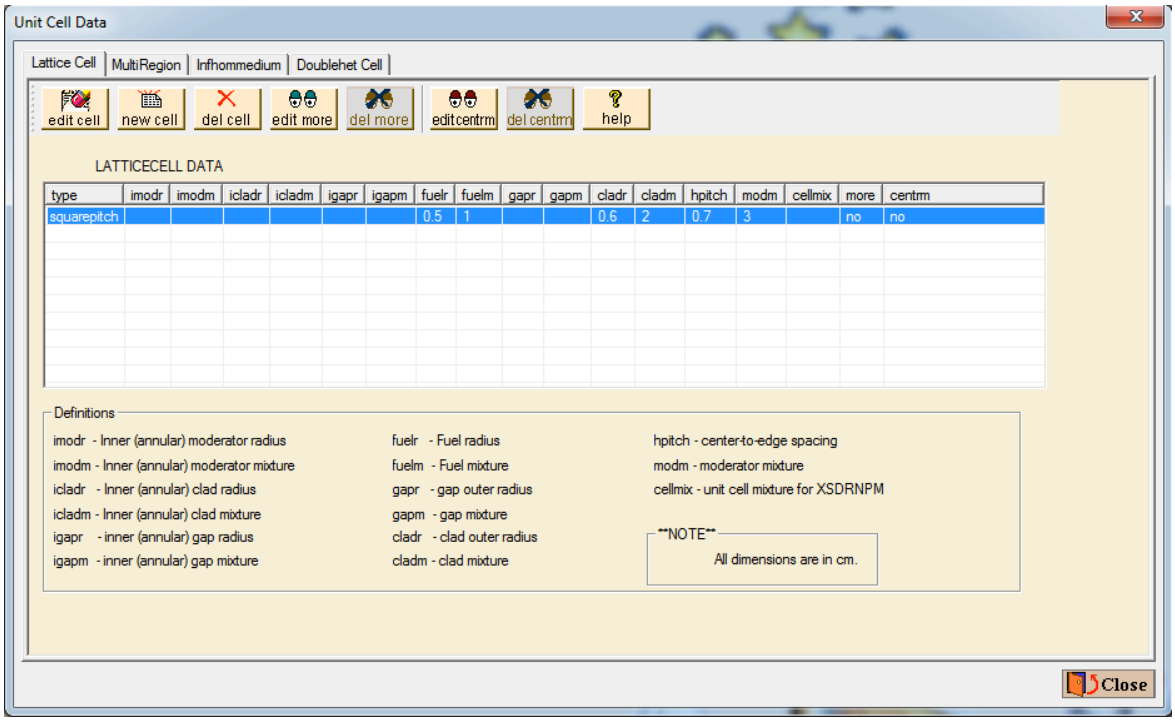


Fig. 2.14. Lattice cell summary screen.

Now that a significant portion of the input for this problem has been completed, it is a good time to save your work by clicking the **Save** button. It is recommended that you frequently save your work, especially if working on a large input file. Because the model is not entirely complete, GeeWiz will warn the user about the lack of a global unit; continue and select **Yes**. GeeWiz then asks if you would like to perform a partial save; again click **Yes**. Name the file `ucell1.inp` and set the **Save as** type to **All Files(*.*)**.

2.3.5 Materials

Before moving to the **Geometry**, it is required to specify the **Materials** that will be used in the geometry. Although it may seem redundant, the reason for this will become clear when you learn the more advanced features of TRITON. To do this, click the **Materials** button along the left vertical toolbar of the GeeWiz main window.

You should now have another window open called **Material Mixtures**. Click the **Add Mixture** button to open another window called **Material Mixture**, as seen in Fig. 2.15.

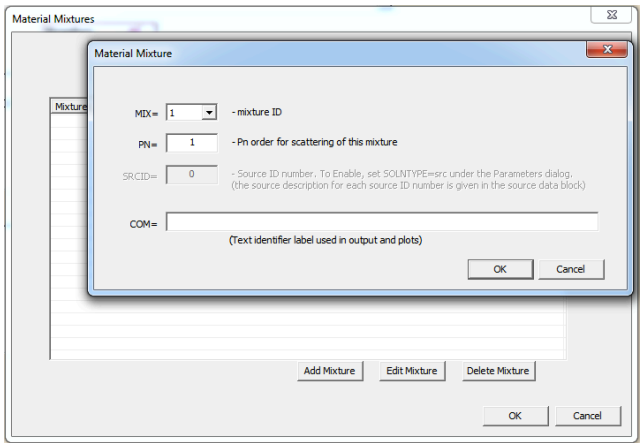


Fig. 2.15. Material mixture input form.

There are three input options available: **MIX**, **PN**, and **COM**. **MIX** is the mixture number that was specified in the compositions section. The following mixture numbers should be available: 1, 2, and 3. Remember that mixture 1 was specified as the fuel (UO_2), mixture 2 as the clad (Zircaloy-4), and mixture 3 as the moderator (H_2O). **PN** is the Pn order of scattering for this mixture. Generally, Pn=1 is recommended for fuel and structural materials, and Pn=2 for materials that are highly moderating (water, graphite, etc.). You can also insert a comment on the mixture with the **COM** option. This comment will appear on the geometry plot, so use something descriptive here. For the fuel, select **MIX**=`1` for UO_2, **PN**=`1`, and **COM**= `fuel, 3.5% enr UO2`, as shown in Fig. 2.16. Click **OK** to close the form.

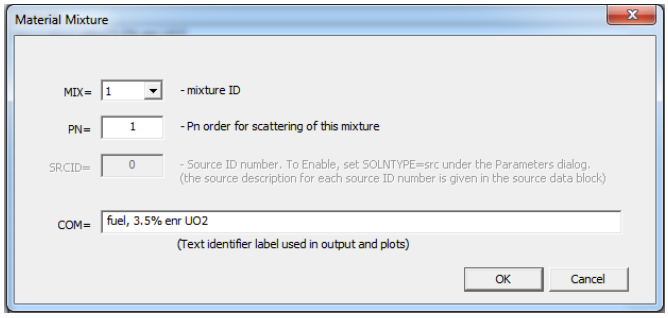


Fig. 2.16. Material mixture input for mixture 1.

You will see that the main **Material Mixtures** window has mixture 1 specified. Follow the same process for adding the clad and the water to the materials. Click **Add Mixture** and enter **MIX**=`2`, **PN**=`1`, **COM**=`clad, zirc4` to specify the clad; then click **OK**. Click **Add Mixture** and input **MIX**=`3`, **PN**=`2`, **COM**=`moderator, h2o` to specify the moderator; then click **OK**. The final Material Mixtures window should look like Fig. 2.17. Click **OK** to close the window.

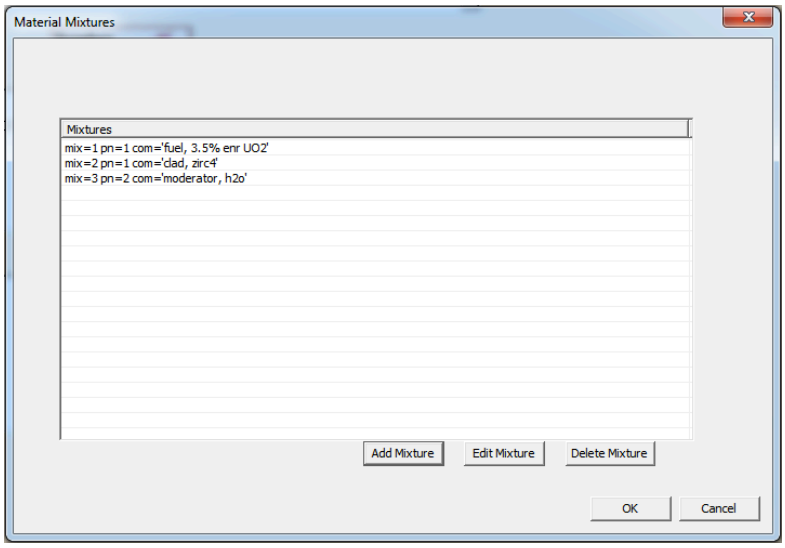


Fig. 2.17. Material mixtures summary.

2.3.6 Geometry

In the next section, the **Geometry** must be specified. The **Geometry** section assembles the geometry that NEWT will use to perform the transport calculation. In this section, the geometry shapes and dimensions, the materials in the shapes, the boundary conditions, and the grid that will be used to solve for neutron fluxes will be specified. The **Geometry** section is usually the largest section of TRITON input files, and you will likely spend most of your time working in this section.

Select the **Geometry** box on the left vertical toolbar on the GeeWiz main window. This will open the **Geometry Form** that will be used to input the geometry specifications, as can be seen in Fig. 2.18.

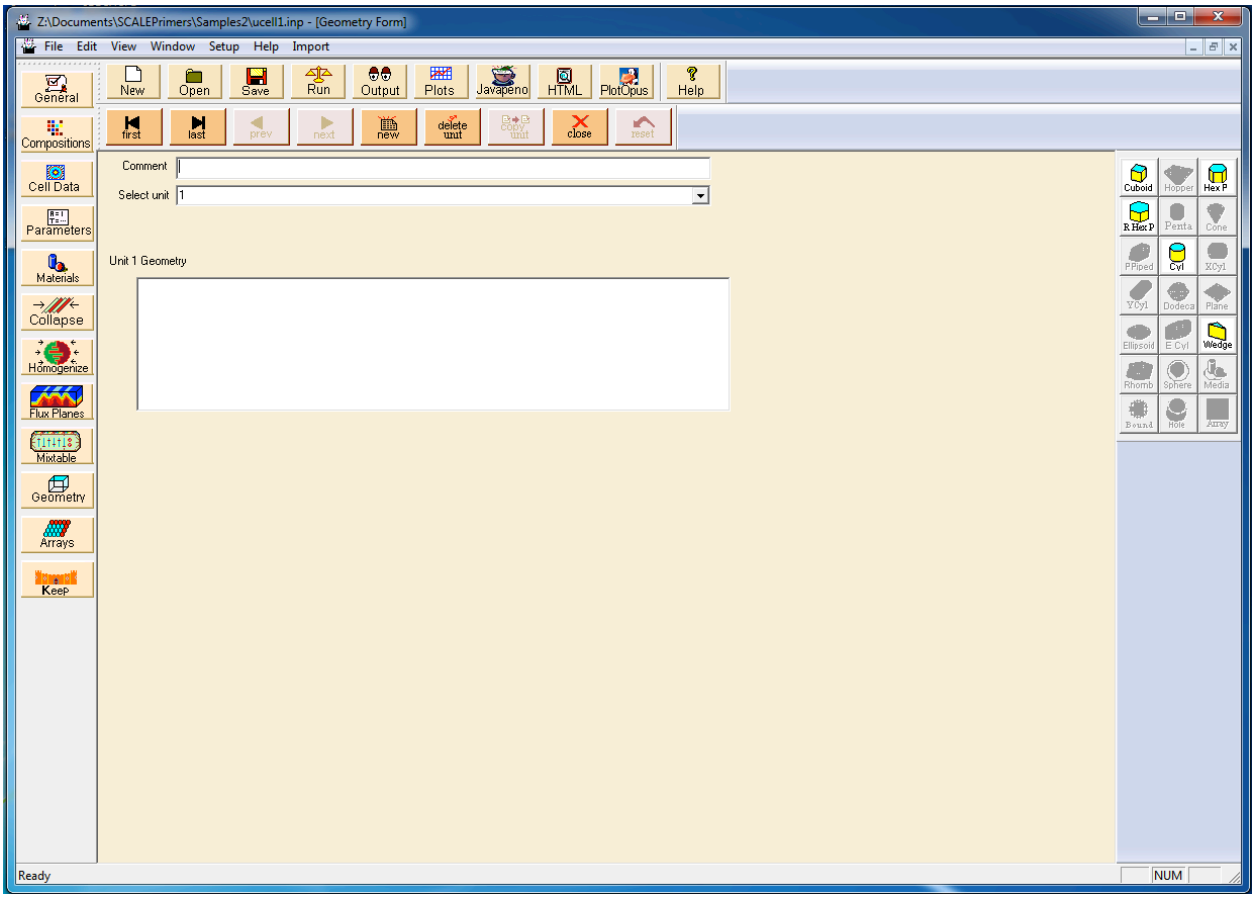


Fig. 2.18. NEWT geometry form.

Once you click the **Geometry** button, you will notice a new option on the left vertical toolbar (**Arrays**), and along the right vertical size of the GeeWiz window you will see the five different body shapes that are available in NEWT (**Cuboid**, **Hex P**, **R Hex P**, **Cyl**, and **Wedge**). In the middle of the window you will see a **Comment** box, a **Select unit** box, and a **Unit Geometry** box.

The basis of the SCALE geometry in NEWT and KENO is the unit. A unit is a geometrical space where multiple shapes can be defined and filled with materials. The geometry can be built with as many or as few units as the user desires. Each geometry specification requires at least one unit. Having multiple units generally simplifies the input file and makes it easier to understand the model. In the actual text-based SCALE input file, the unit begins with a `Unit` statement and is terminated with a `Boundary` statement, where the user specifies a surface that will act as the outer boundary of that unit. Units can be used to fill an `Array`, and units can also be placed wherever needed in the geometry using a `Hole` statement (both of these will be discussed later). The important thing to remember is to specify units as building blocks (e.g., a single unit fuel pin cell) that can be combined to build complex models. Every model must have a global unit that is filled with its own geometry or other units contained in arrays or holes.

For this simple problem, three different bodies are required: a fuel cylinder, a clad cylinder, and a cuboid of water surrounding the fuel pin. It is recommended to specify units from the inside to the outside. The first surface to specify is the fuel cylinder. To do this, click **Cyl** in the right toolbar on the **Geometry** form. This opens the **Cylinder** specification form as shown in Fig. 2.19.

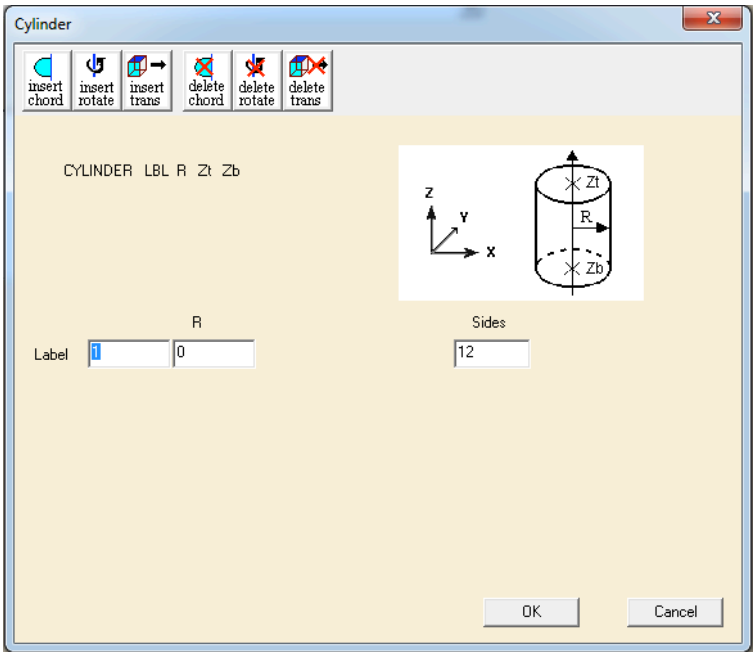


Fig. 2.19. Cylinder input form.

There are three basic inputs to the Cylinder window: **Label**, Radius (**R**), and **Sides**. The **Label** is a user-defined number that will be used to specify this surface. It is recommended that you specify surface numbers as multiples of 10 (10, 20, 30, … etc.). The reason for this is that if you need to add another surface later, such as a gap between the fuel 10 and the clad 20, you can insert a surface number that is between the two surrounding surface numbers, 15 for example, without having to modify all the other surface numbers. Because NEWT cannot solve for fluxes along a curved boundary, NEWT approximates a circle by an *n*-sided polygon—the default number of sides is 12. NEWT conserves the total volume of the cylinder (area of the circle) when constructing the *n*-sided polygon. Specify this fuel cylinder with a **Label** of `10`, **R** of `0.5`, and **Sides** of `12` (Fig. 2.20) and click **OK**.

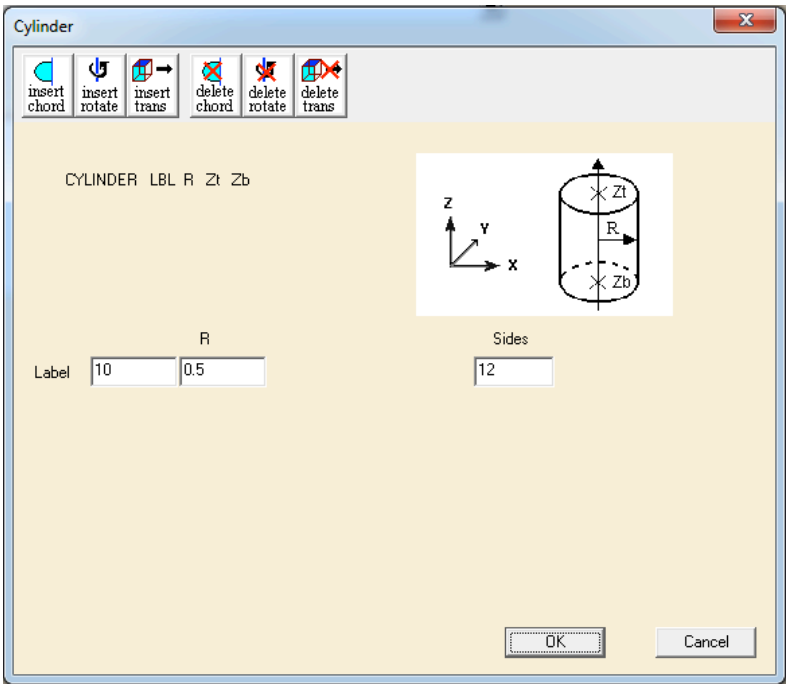


Fig. 2.20. Cylinder input for the fuel.

The main **Geometry Form** now lists cylinder 10 in the **Unit 1 Geometry** section as in Fig. 2.21.

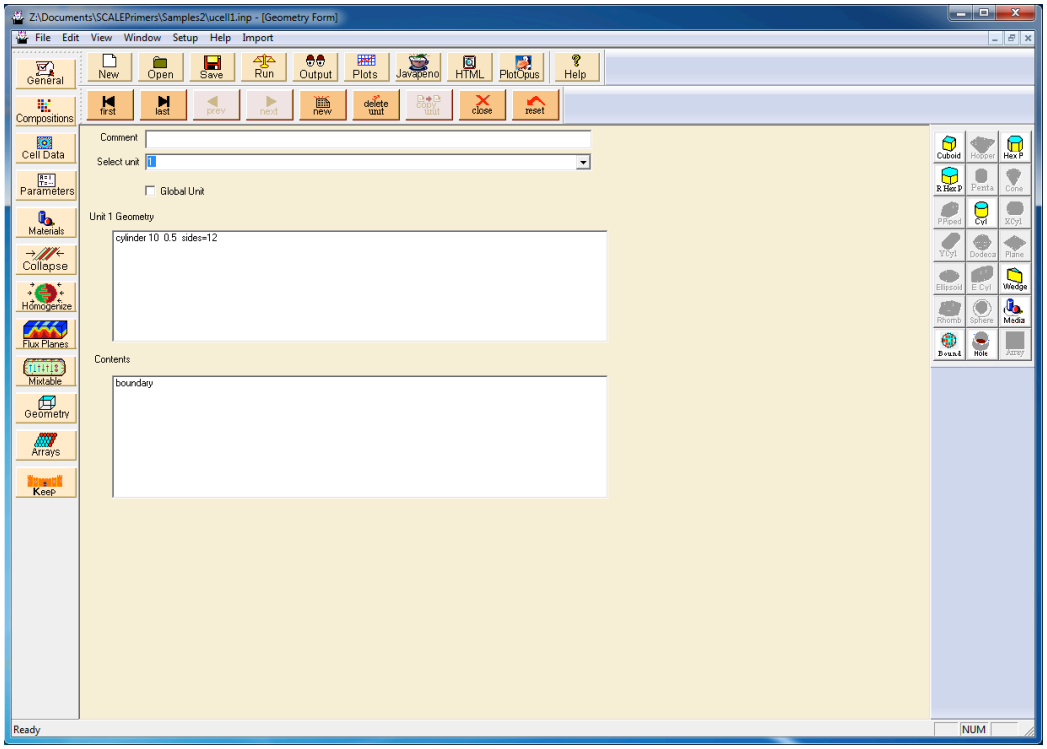


Fig. 2.21. Unit 1 geometry form (fuel only).

Click **Cyl** on the right toolbar again and specify another cylinder with a **Label** of `20`, **R** of `0.6` cm, and **Sides** of `12` for the clad and click **OK**. Next, add the cuboid of water that will act as the boundary of the unit cell. Select **Cuboid** from the right toolbar and input a **Label** of `30`, **+X** of `0.7`, **-X** of `-0.7`, **+Y** of `0.7`, and **-Y** of `-0.7` and click (Fig. 2.22). This specifies a square centered at (0,0) with sides of 1.4 cm.

Note that GeeWiz uses two shortcut keys to reduce the effort for the user. The "P" key (i.e., "P" = "plus/minus") alternates positive and negative repeating values in consecutive fields, while the "R" key (i.e., "R" = "repeat") repeats the value in the current field. For example, you can enter 7.0 for +**X** and then press the "P" key three times to fill the **–X**, **+Y**, and **–Y** fields for this cuboid.

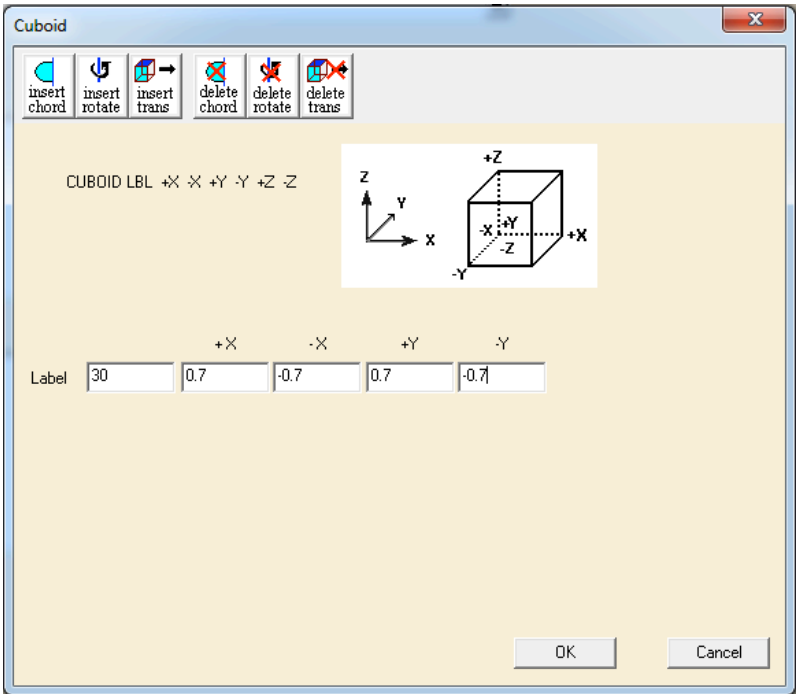


Fig. 2.22. Cuboid input for water.

The **Geometry** form should now look like Fig. 2.23.

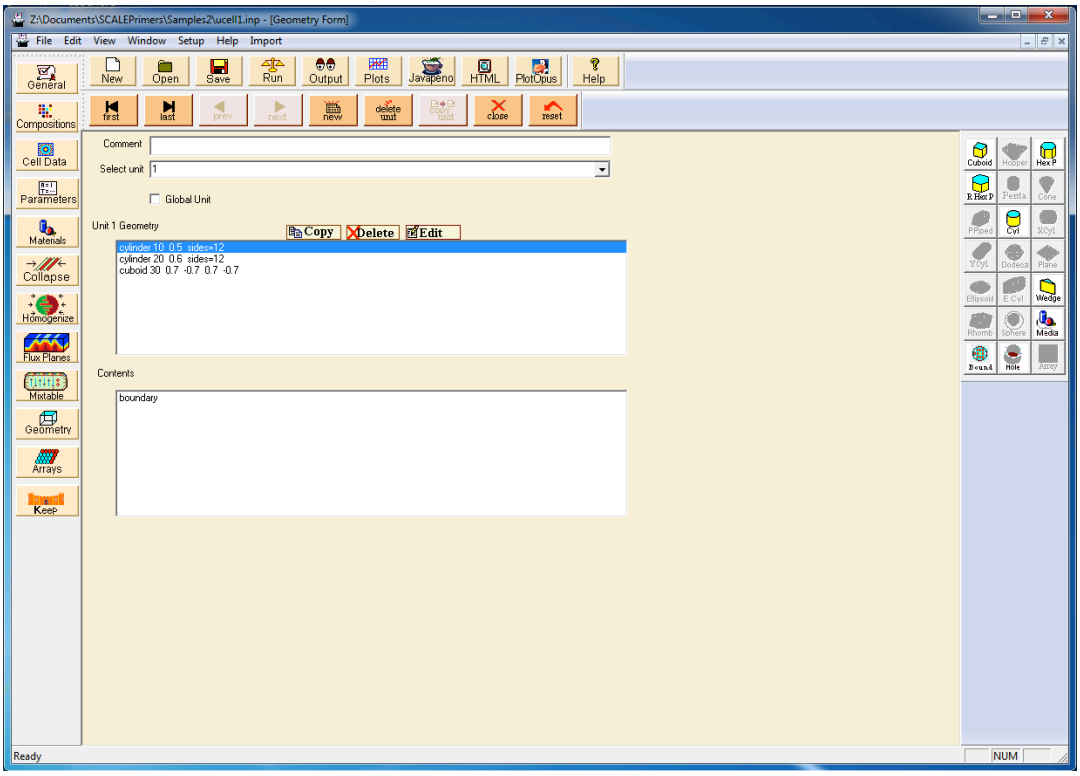


Fig. 2.23. Unit 1 geometry form (fuel, clad, water).

Now the media that fills each shape must be specified. Select **Media** from the right toolbar. This action opens a new window titled **Media for Unit 1** as seen in Fig. 2.24. In SCALE, bodies, and the media contained in those bodies are specified using surface boundaries.

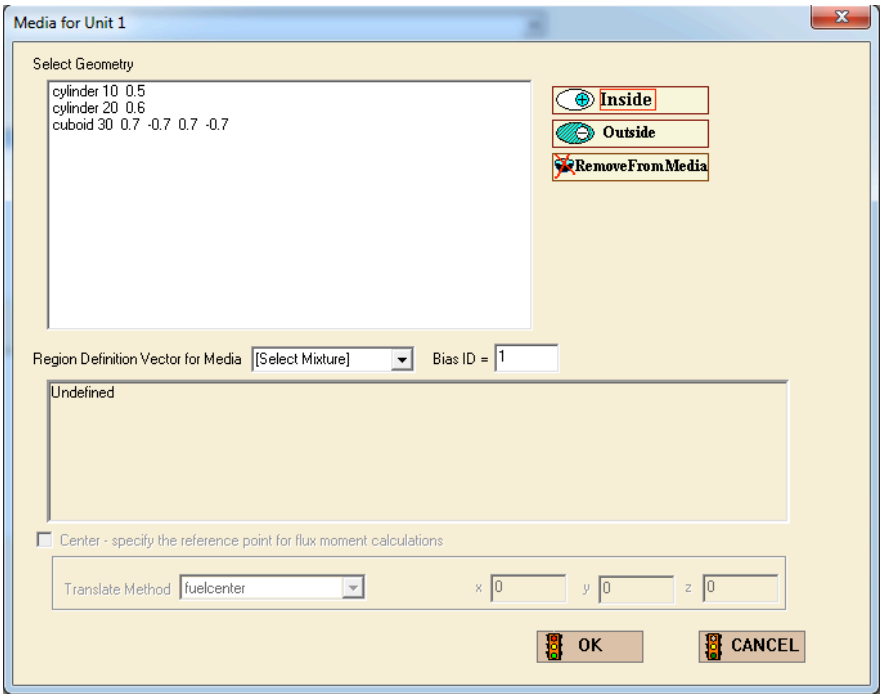


Fig. 2.24. Media for unit 1 form.

First, you should specify that the fuel mixture (`1 uo2`) be inside **cylinder 10**. To do this, select `1 uo2` from the **[Select Mixture]** dropdown. Click `cylinder 10 0.5` in the **Select Geometry** box so that it is highlighted blue and then click the **Inside** button (Fig. 2.25). The **Bias ID** will be ignored in these examples as it is a KENO-only input parameter, and it not used in NEWT.

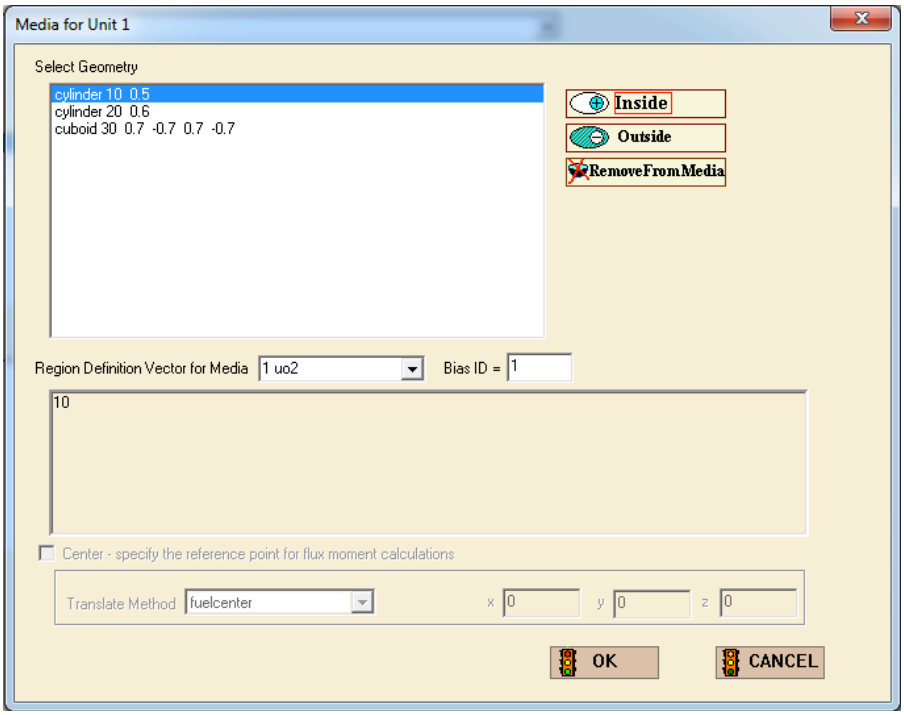


Fig. 2.25. Media for cylinder 10 in unit 1.

Click **OK** to close the **Media** form. Click the **Media** button again on the right toolbar to specify the second `media` statement. Select `2 zirc4` from the **[Select Mixture]** dropdown and click **cylinder 10 0.5** from the **Select Geometry** box so that it is highlighted blue; click **Outside** button. Now click `cylinder 20 0.6` so that it is highlighted blue and click **Inside** button (Fig. 2.26). By doing this, you have specified that material `2 zirc4` is outside the fuel `cylinder 10`, but inside the clad `cylinder 20` (i.e., the clad is contained in an annulus around the fuel). Now click **OK** to close the form.

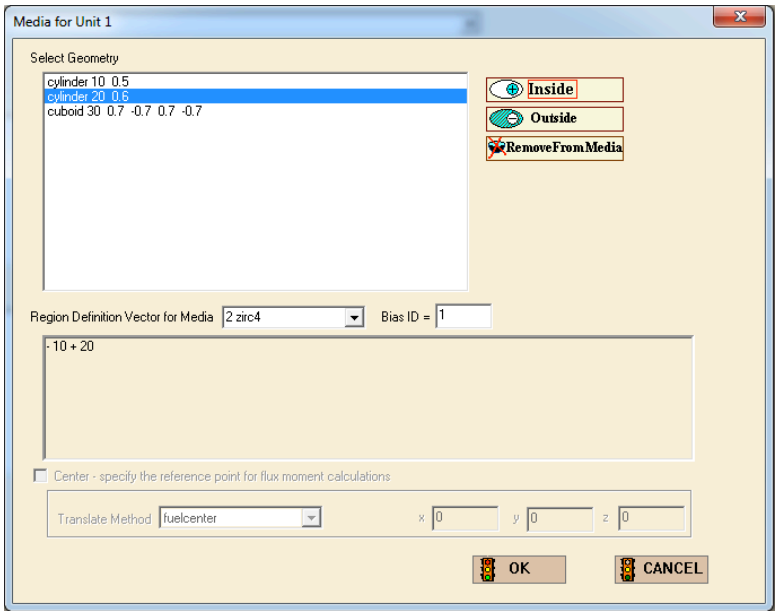


Fig. 2.26. Media for cylinder 20 in unit 1.

The **Contents** block of the **Geometry** from should now have two media statements—media 1 and media 2. To complete the unit cell, specification of a media statement for the moderator that is outside the clad surface but inside the bounding cuboid is needed. To add this, again select **Media** from the right toolbar. Select `3 h2o` from the **[Select Mixture]** dropdown and click `cylinder 20 0.6` from the **Select Geometry** box so that it is highlighted blue; click **Outside**. Now click `cuboid 30 0.7 -0.7 0.7 -0.7` so that it is highlighted blue and click **Inside** (Fig. 2.27). Click **OK** to close the form.

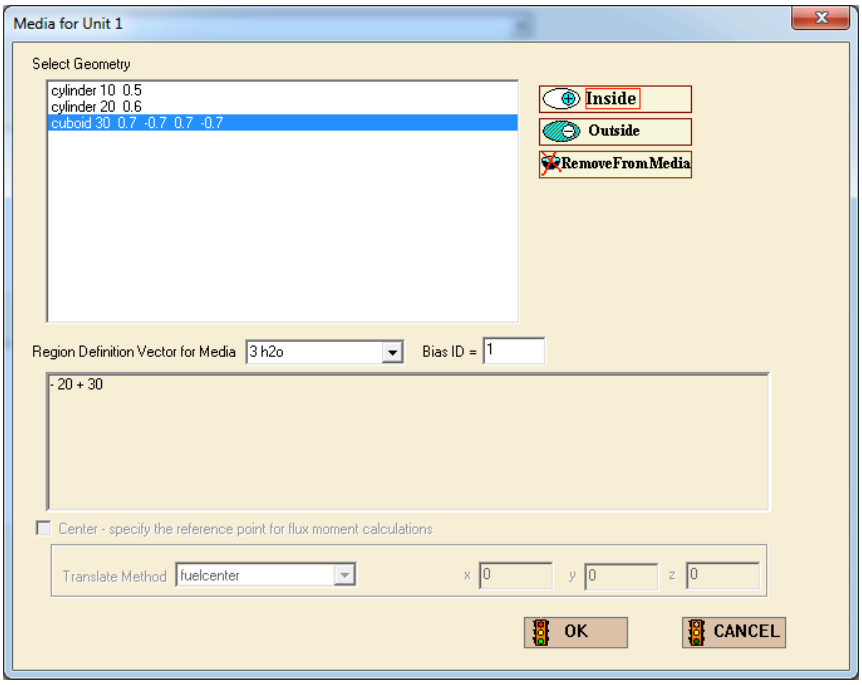


Fig. 2.27. Media for cuboid 30 in unit 1.

The **Geometry Form** window should now look like Fig. 2.28. There should be cylinders labeled 10 and 20, and a cuboid labeled 30 in the **Unit 1 Geometry** box. There should also be three media statements and a boundary statement in the **Contents box**. In the text-based input, the media statements follow a set format: `media matID biasID surfaces`. Using GeeWiz and trying to understand what appears inside the **Unit Geometry** and **Contents** box is a good way to learn the text-based input as the text that appears in these boxes is identical to the text-based input format.

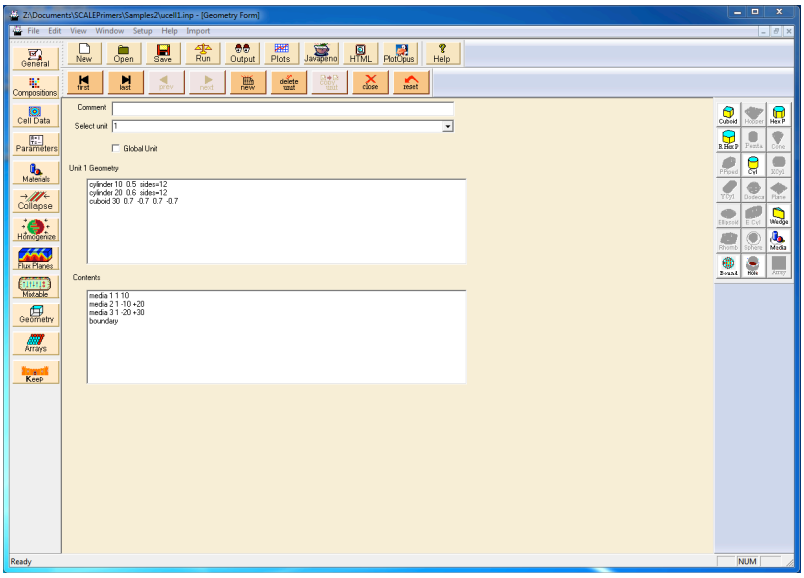


Fig. 2.28. Unit 1 geometry form (with media records).

It is now time to specify the boundary of this unit. On the **Geometry Form**, click `boundary` inside the **Contents** box so that it is highlighted, and then click **Edit**. This opens a **Media for Unit 1** window. Click `cuboid 30 0.7 -0.7 0.7 -0.7` so that it is highlighted, and then click **Inside** to specify that the boundary of unit 1 is defined by cuboid 30. You will also need to apply a mesh to the unit cell using the **Grid Input** box. The grid determines the number of regions that NEWT will use for the flux calculation. Theoretically, the more grid cells that are specified, the more accurate the solution will be. For most cases, grid spacing of 4×4 is adequate for a pin cell. To specify a 4×4 grid, input `4` in the **Grid X** box and `4` in the **Grid Y** box. Your **Media for Unit 1** window should look like Fig. 2.29. If so, click **OK**.

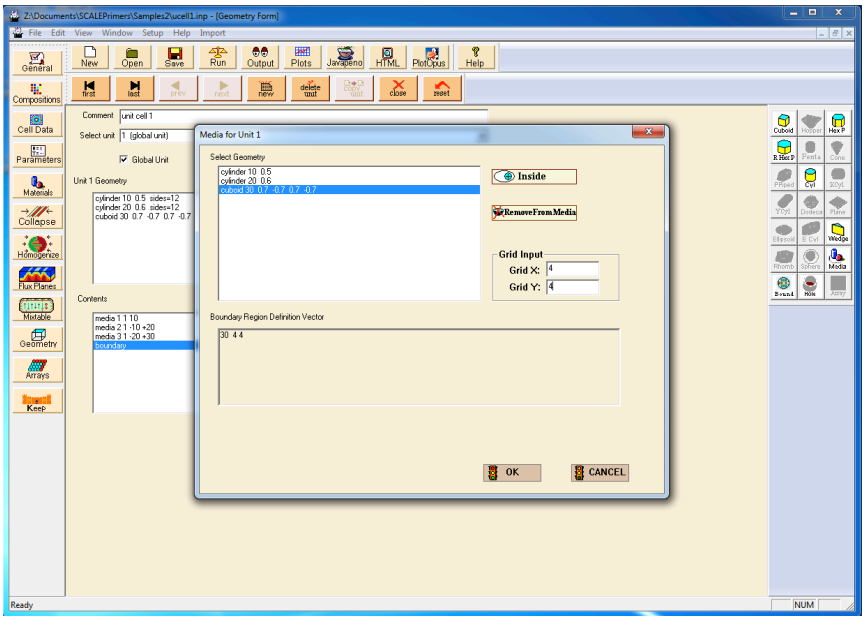


Fig. 2.29. Boundary input for unit 1.

Finally, specify the global unit and associated boundary conditions. In the **Comment** box, type a descriptive comment about the unit—it will help to keep things organized and understandable when the geometries get much more complicated. For this problem, enter `unit cell 1` in the **Comment** box. You should also click the **Global Unit** checkbox. You will notice a new option appear to the right: **Boundary Conditions,** 

. By specifying that this unit is the global unit, the boundary of this unit will act as the boundary of the problem geometry. Now click the **Boundary Conditions** button, and a new window appears named **Bounds Data**. For the **Bound Body** dropdown, select **30 – CUBOID** to specify that the cuboid labeled "30" will act as the problem boundary, and specify **ALBEDO** boundary condition as `mirror`—this represents a reflective boundary condition. The **Bounds Data** window should look like Fig. 2.30. Click **OK** to close the window.

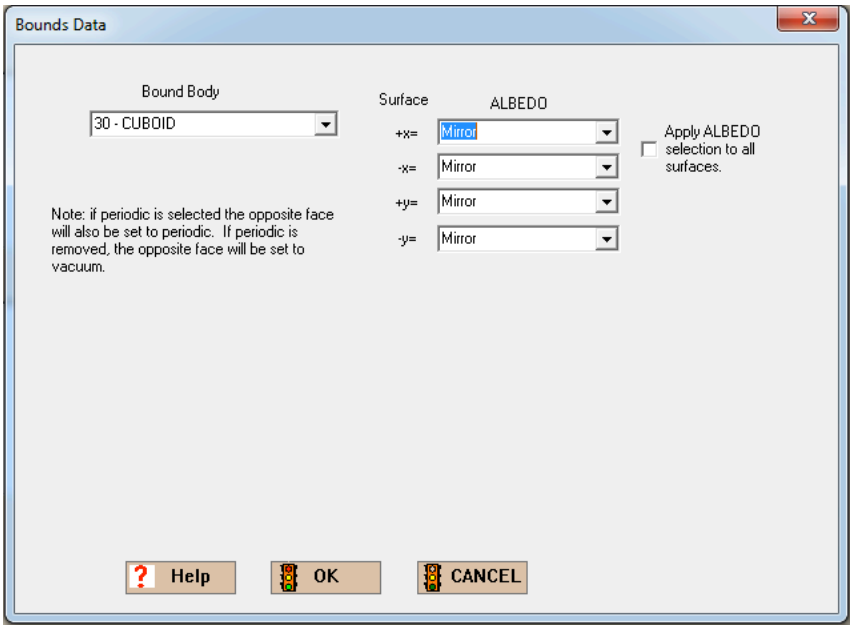


Fig. 2.30. Bounds data for global unit 1.

2.4 RUNNING SCALE/TRITON

Before running SCALE, click the **Parameters** box on the left toolbar. This opens the **Parameter Data** window with special NEWT control parameters. Check the **DRAWIT** checkbox so that TRITON will produce a plot of the geometry. When running the problem, it is helpful to see the calculation progress; to do this, check the **ECHO** box under the **Parameter Data** as well. Every other **Parameters** option can be left as the default for now. Click **OK**.

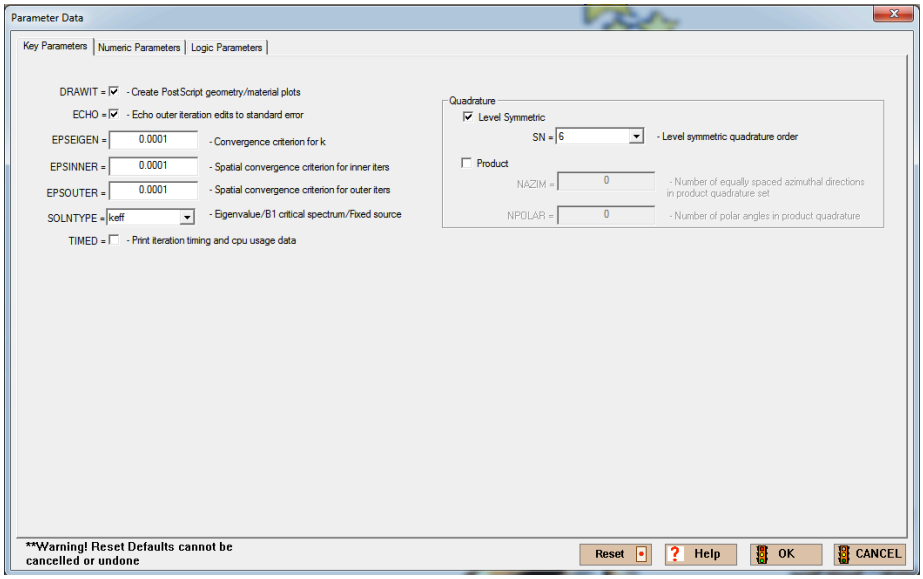


Fig. 2.31. Parameter data form.

To run SCALE/TRITON, click the **Run** button on the top toolbar. GeeWiz opens a DOS command prompt window to execute SCALE. Remember that the `Parm=Check` box was selected on the **General** form so TRITON will perform only an input check. Check the message at the end of the output file to see whether any errors were found. It is a very good idea to navigate to the directory where the input file resides and see if TRITON created two PostScript files (grid plot and material plot) that should have the extension ".ps." One PostScript file should be named `ucell1.newtmatl.ps` that displays the materials assigned to each region in NEWT. To view this file, you will need to have a PostScript viewer on your computer. Note: Adobe Reader cannot view or convert PostScript files, but the full version of Adobe Acrobat can. If you do not have a PostScript viewer/converter, you may download and install Ghostscript for free—a tutorial can be found in Appendix B.

If you have Ghostscript or some other PostScript file viewer installed, double click the `ucell.newtmatl.ps` file and view a color picture of the geometry as can be seen in Fig. 2.32. Using the PostScript files is important for verifying the accuracy of your model. It is recommend that you run TRITON with `Parm=check` and `DRAWIT=yes` before running the actual calculation. This practice minimizes time debugging geometry and other input errors. In addition, the plots can be used in reports and presentations.

If there are no errors and the geometry plot looks as it should, uncheck the `Parm=Check` box in the **General** form and click **OK**. You are now ready to run the calculation, so click **Run**. If the DOS window is still open from the parm=check run, close it first.

You can observe the progress of the calculation in the command window that appears. When SCALE starts, it echoes the input and output filenames and directory, the modules that are being executed, and the eigenvalue for each outer iteration. You can see in Fig. 2.33 that the calculation converges to k_{eff} = 1.34190627.

The screen capture in Fig. 2.33 shows that the job named **ucell1.inp** completed without error. The header indicates that the output file is called **ucell1.out** and it is stored in output directory **Z:\Documents\SCALEPrimers\Samples2**.

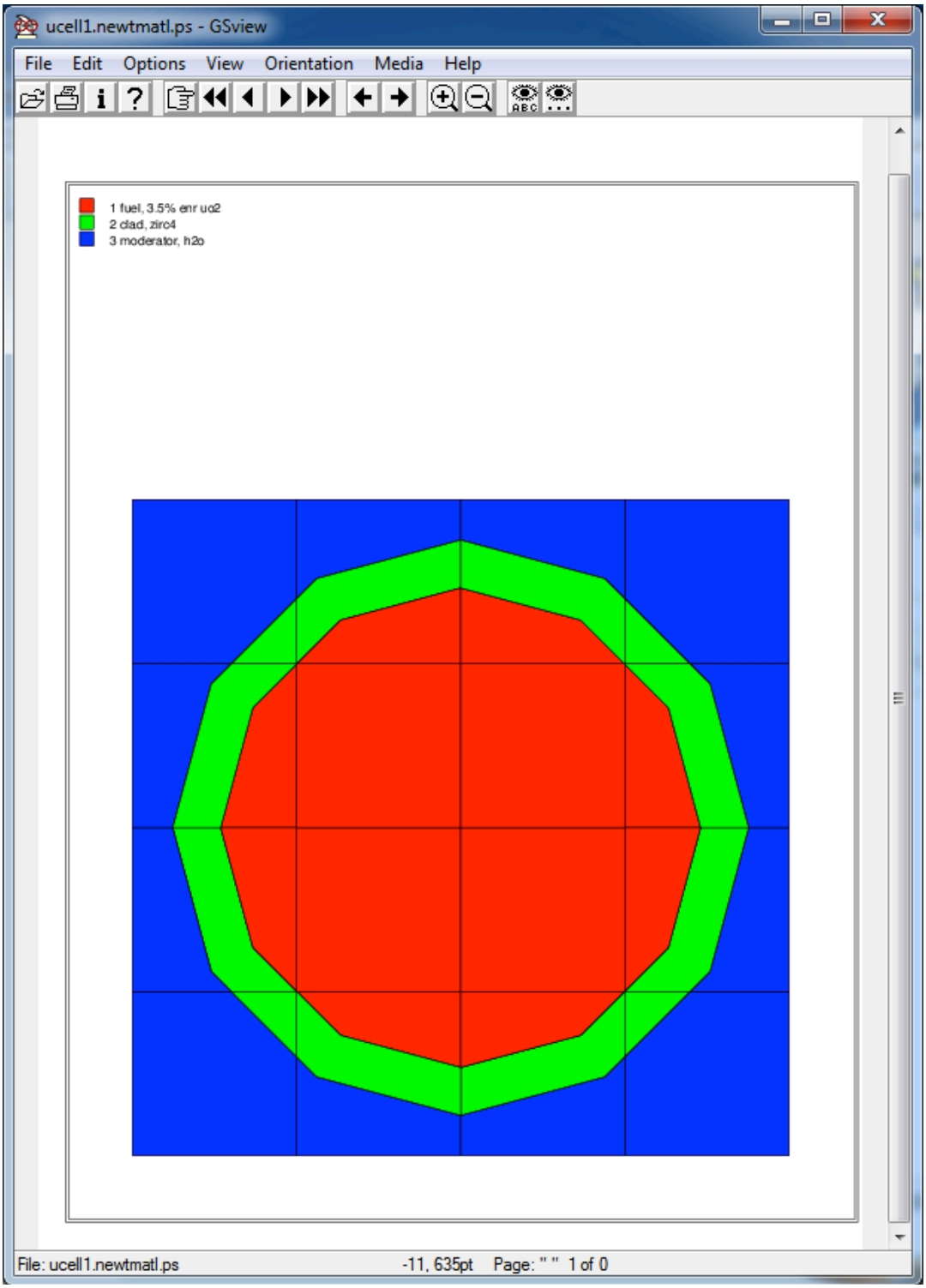


Fig. 2.32. NEWT material PostScript file.

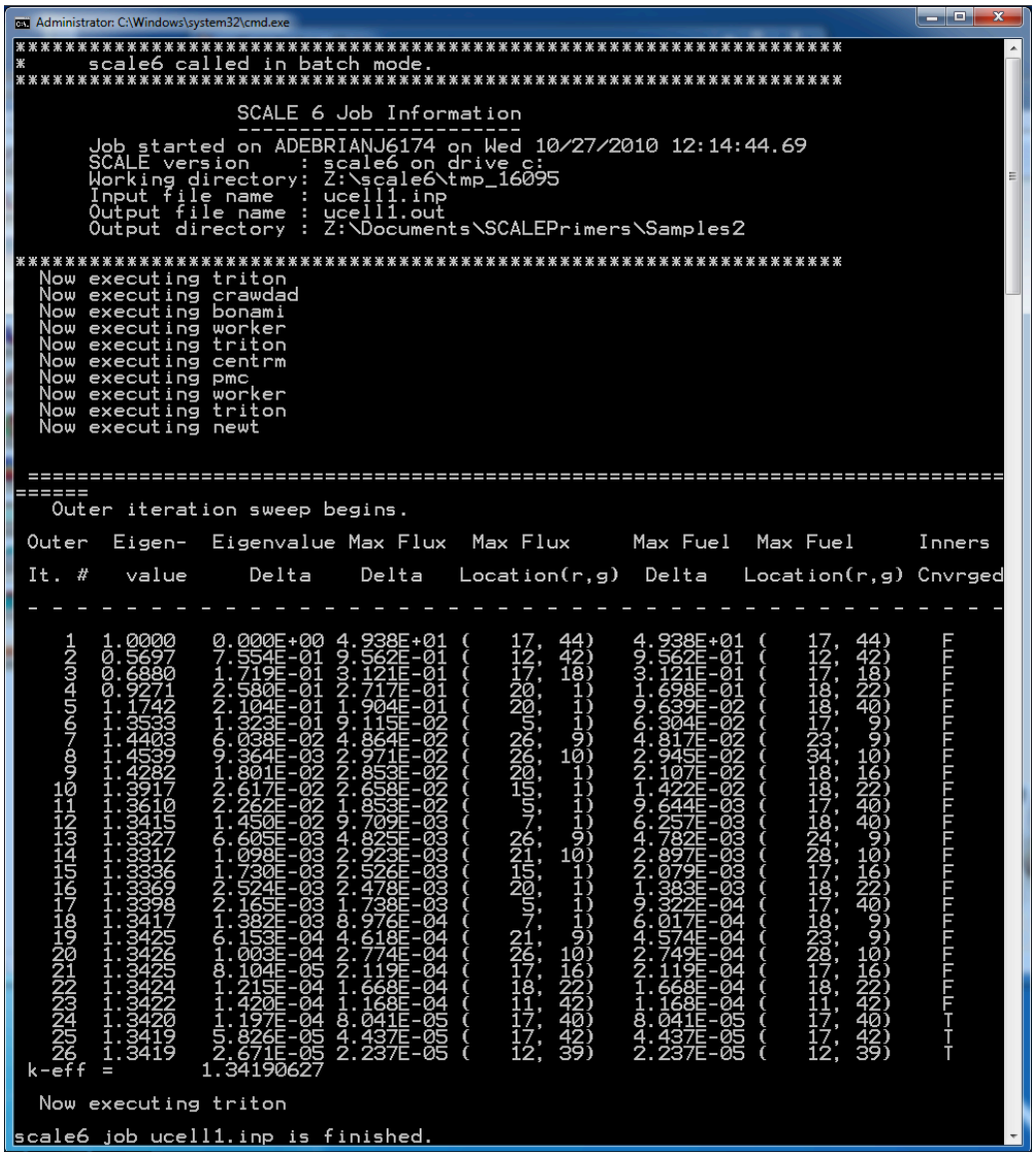


Fig. 2.33. DOS command prompt window showing completed run.

2.5 SCALE/TRITON OUTPUT

You can view the output file by returning to GeeWiz and clicking on **Output** on the top toolbar. Alternatively, you can navigate to the output file directory and open the output file (with the extension of ".out") with a text editor such as WordPad. By default, GeeWiz opens the file in the Programmer's File Editor (PFE) that is distributed with SCALE. Note: *If the default is not set or you would like to use a different text editor, you can change the default text editor using the* ***Setup*** *button in GeeWiz.*

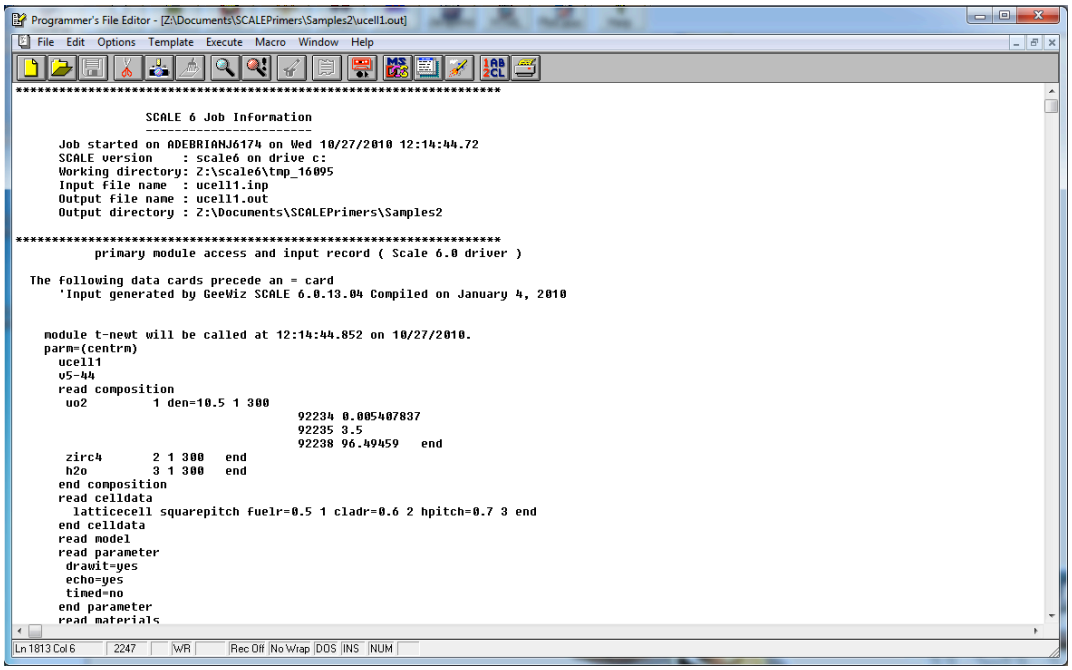


Fig. 2.34. Input echo at top of output file.

Figure 2.34 is a view of the ASCII text output file in PFE. The text output file contains the following sections:

- echo of input;
- TRITON information and brief review of input values;
- cross-section processing and data handling output from CRAWDAD, BONAMI, WORKER, CENTRM, and PMC; and
- NEWT output data.

There are many pages in the output file. Each module has a section of output, starting with the TRITON control module, which is then followed by the functional modules BONAMI, WORKER, CENTRM, PMC, WORKER, and NEWT. For lattice physics, the main interest is the cross sections that are produced, but it is essential to check the output to have confidence in the accuracy of the cross sections produced. If you did not record k_{eff} from the DOS window, you can search for "`k-eff`" in the output file. The first instance is the initial guess of k_{eff} and the second is the k_{eff} that NEWT calculates. Alternatively, you can simply search for "`k-eff =`" to jump directly to the k_{eff} calculated by NEWT. Here you can also see a four-factor summary of the problem, as shown in Fig. 2.35.

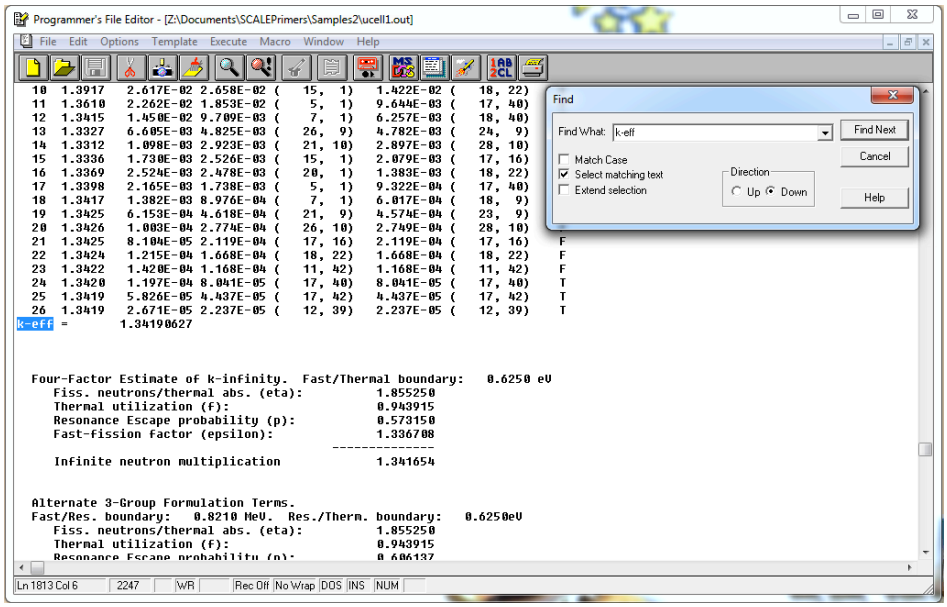


Fig. 2.35. Calculated k_{eff} in output file.

2.6 SUMMARY

This section has helped you to accomplish the following:

- understand the structure of SCALE/TRITON input files and know that there are parts describing the sequence, control parameters, materials, unit cells, and the NEWT geometry information
- use the GeeWiz user interface to create a SCALE/TRITON input file
- set up and run a simple criticality problem using SCALE/TRITON

Now that you have successfully run SCALE/TRITON, you are ready to learn the options available in each input segment and how to set up more-complex problems in detail. The sections that follow present these details in a similar format to that used here in *Quickstart*.

3. MATERIAL INFORMATION INPUT

In the *Quickstart* section (Sect. 2), you ran a simple problem with SCALE/TRITON using the GeeWiz User Interface. From this you gained confidence in using the code and some experience with GeeWiz. This section and subsequent sections provide a more detailed explanation of the commands used in the *Quickstart* section.

3.1 WHAT YOU WILL BE ABLE TO DO

- identify the cross-section libraries available for lattice physics analyses
- use the GeeWiz user interface to provide input data on more complicated basic compositions, compounds, and alloys
- use the GeeWiz user interface to learn the cross-section processing options used in lattice physics

3.2 TRITON SEQUENCE

For ease of use and minimization of human error, the SCALE data handling and program flow are automated as much as possible through the use of control modules. These control modules contain calculational sequences that prepare the input for and call the functional modules required for a particular analysis. For lattice physics analysis, the TRITON T-DEPL depletion sequence (Fig. 3.1) is used to provide automated problem-dependent cross-section processing followed by a transport calculation of the neutron multiplication factor and flux distribution and then the depletion/decay calculations.

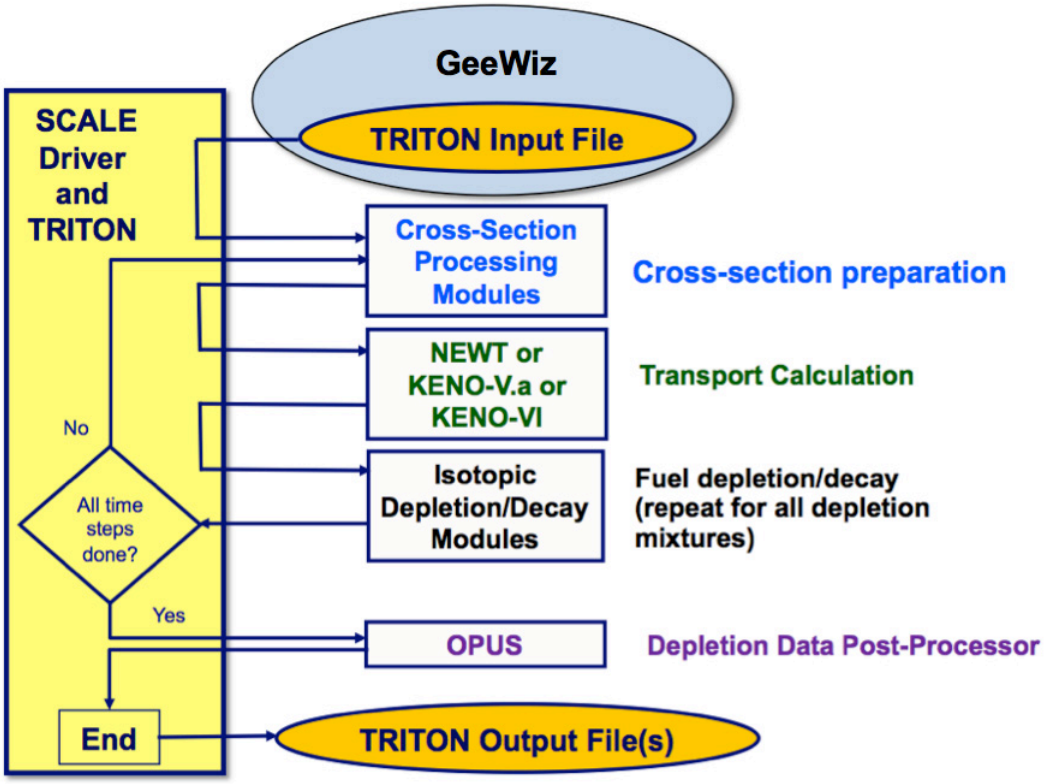


Fig. 3.1. TRITON T-DEPL depletion sequence.

This control sequence uses the cross-section processing codes BONAMI (for the unresolved resonance energy range) and NITAWL or CENTRM/PMC (for the resolved resonance range) to provide multigroup resonance-corrected cross sections. CENTRM/PMC use continuous-energy data to provide a more rigorous method than NITAWL for generating resonance-corrected multigroup cross sections. Note that CENTRM/PMC is the default for SCALE multigroup calculations and must be used with multigroup ENDF/B-VI or ENDF/B-VII libraries.

The selection of a sequence is done through the **General** information window of GeeWiz. Click on the **General** button to open this window. Information in this window includes a **Title** for the problem, the **Application**, and the **Sequence** that will be executed. For TRITON, there are a number of different sequences available. The T-XSEC sequence is used to process cross-section data; the T-NEWT sequence is used to process cross sections and run a steady-state NEWT transport calculation; the T-DEPL sequence is used to process cross sections and perform time-dependent transport and depletion calculations with NEWT and ORIGEN; and the T5-DEPL and T6-DEPL sequences are similar to T-DEPL but they use KENO-V.a or KENO-VI, respectively, for the transport calculations. This primer focuses entirely on the T-NEWT and T-DEPL sequences.

3.3 CROSS-SECTION LIBRARIES

The TRITON sequence uses the SCALE Standard Composition Library (described later in this section) for specifying the materials and mixtures used in a calculation and provides automatic problem-dependent multigroup cross-section preparation prior to the transport calculation. This section describes the cross-section libraries available for 2-D or 3-D transport calculations in SCALE 6.1.

There are several multigroup cross-section libraries distributed with SCALE suitable for lattice physics analyses. The 238-group ENDF/B-V library (238GROUPNDF5 or V5-238) contains data for all ENDF/B-V nuclides and has 148 fast and 90 thermal groups. Most resonance nuclides in the 238-group library have resonance data in the resolved resonance region and Bondarenko factors in the unresolved resonance region. The 44-group ENDF/B-V library (44GROUPNDF5 or V5-44) is a broad-group version of 238GROUPNDF5 designed for analysis of light-water-reactor (LWR) fresh and spent fuel systems and has been extensively validated against LWR critical experiments. In addition, there are 238-group ENDF/B-VI (V6-238) and ENDF/B-VII libraries (V7-238) that have the same group structure as the ENDF/B-V version. SCALE has continuous energy cross-section libraries, but they are not available to the T-NEWT and T-DEPL sequences because they are not compatible with NEWT, which is a multigroup code.

Table 3.1 lists the standard SCALE cross-section libraries, along with the alphanumeric name and the main source of data for each library. A more detailed discussion of the cross-section libraries is contained in the SCALE manual [SCALE manual, Section M4].

Table 3.1. SCALE cross-section libraries for lattice physics analyses

Name	Description	Primary source of data
v5-44 (44groupndf5)	ENDF/B-V 44-group neutron library	Collapsed from 238-group ENDF/B-V library
v5-238 (238groupndf5)	ENDF/B-V 238-group neutron library	ENDF/B-V data
v6-238	ENDF/B-VI 238-group neutron library	ENDF/B-VI Release 8
v7-238	ENDF/B-VII 238-group neutron library	ENDF/B-VII Release 0

For lattice physics simulations, use of the ENDF/B-VII 238-group library (v7-238) is recommended; this is the latest multigroup library that is available to SCALE users. Using a library with a smaller number of groups will automatically speed up the calculation, but the ENDF/B-V 44-group library was collapsed with a spectrum that may be slightly different than the spectrum in your problem of interest. This library is good for scoping and test calculations, but it is not recommended for production calculations. A good tradeoff between speed and accuracy can be obtained using TRITON's `parm=weight` function, which uses the problem-dependent spectra to collapse the 238-group library you are using to a 49-group library. This option will be discussed in depth later in the primer.

A library is selected in the **General** form from the drop-down menu for **Cross-section Library** (see Fig. 3.2).

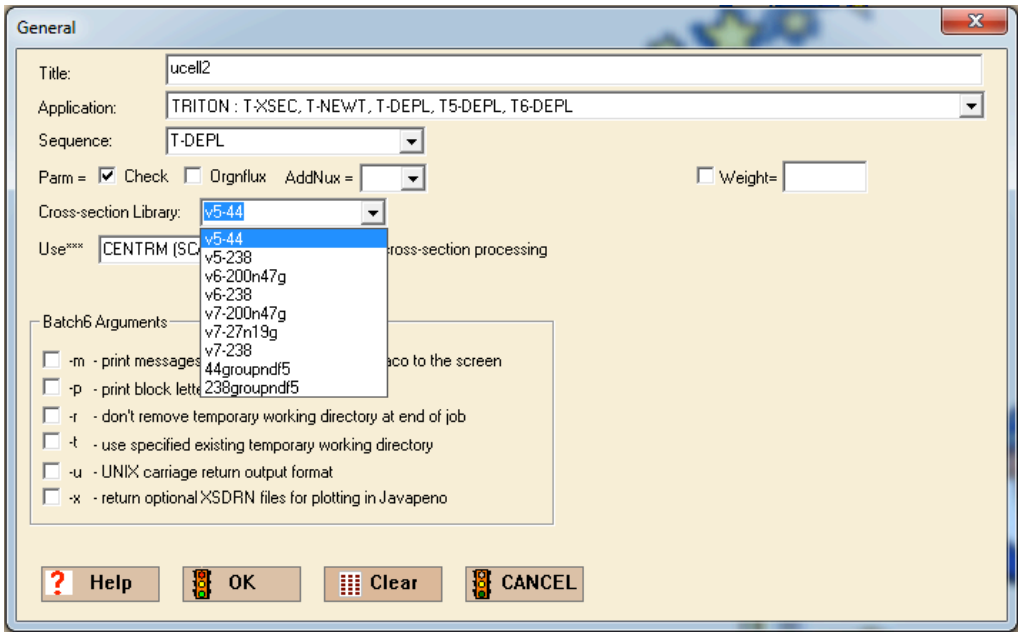


Fig. 3.2. Selection of cross-section library.

For multigroup libraries, TRITON calls BONAMI first to process nuclides with Bondarenko data. The master library output from BONAMI is input to NITAWL, where resonance data are processed via the Nordheim Integral Treatment or to CENTRM, which performs a 1-D continuous-energy calculation that is subsequently collapsed to multigroup cross sections by PMC. In either case, the result is an AMPX working-format library that is used by NEWT for the transport calculation. **Note that CENTRM must be used to process ENDF/B-VI or later multigroup cross-section libraries.** GeeWiz will limit the user options to CENTRM or 2REGION (a simplified CENTRM calculation) if an ENDFB-VI or ENDFB-VII library is selected.

3.4 MATERIAL INPUT

Information for materials in a system model is entered into GeeWiz, which converts that information into the format required by MIPLIB in TRITON. Input data to GeeWiz identify the materials from the Standard Composition Library and associated physical densities to calculate the number densities (atoms/b-cm) of each material specified in the problem. These input data include (1) the Standard Composition data that are used in the standardized number density calculations (a standardized alphanumeric name, mixture number, and other data to define materials, including volume fraction or percent theoretical density, temperature, and isotopic distribution), and (2) the unit cell description defining the materials, dimensions, and boundary conditions of the geometry that will be used in the Dancoff factor calculations, the resonance self-shielding calculations, and the flux-weighting cell calculations necessary for multigroup cross-section processing.

The Standard Composition Library describes the various predefined isotopes, elements (both symbols and full names), compounds, alloys, and other materials that can be used to define the material mixtures for a given problem. A complete description of the materials in the library is contained in the SCALE manual [SCALE manual, Section M8]. The library contains over 700 compounds, alloys, elements, and isotopes that one may use in defining the material mixtures for a given problem.

When formulating a mixture, it is often necessary to know the density (g/cc) of the mixture and the weight fractions of the various constituent materials. Because SCALE uses naturally occurring isotopic abundances for materials, SCALE default densities should not be used for materials containing enriched isotopes, especially light elements with strong absorbers such as boron, B_4C, or lithium. Also, nuclear fuels are usually specified as a percentage of theoretical density (TD). The SCALE default density for UO_2 is 100% TD. Users need to specify the percent TD or a user-specified density in such cases. The default temperature value is 300 K (room temperature). Users need to supply operating condition temperatures of each material to correctly process resonance data, Bondarenko data, and/or thermal-scattering data. If the temperature is specified in the design data, it should be used, or a best estimate of the temperature of a material should be obtained. Likewise, the moderator density at operating conditions should also be specified in the input.

3.5 MATERIAL INPUT - LWR SAMPLE PROBLEMS

A mini-assembly sample problem will be used to illustrate how to enter material information into TRITON using GeeWiz. It is not possible to complete and run this sample problem in this section because the mini-assembly contains more advanced geometry input that will be learned later. Rather, the materials that will be later used in a mini-assembly sample problem will be set up.

3.5.1 Mini-Assembly Materials

A typical lattice physics model contains many different materials. In this section you will learn how to set up materials for many of the basic materials that are typically used in lattice models. Materials for standard UO_2 fuel, UO_2 fuel containing gadolinium oxide, Zircaloy clad, stainless steel, borated water, and a typical absorber material for burnable poison or control rods will be entered in this section. Table 3.2 summarizes the materials that will be built in GeeWiz.

Table 3.2. Material data for mini-assembly problem

Material	Density (g/cm^3)	Temperature (K)
UO_2(4.00% enr)	97% TD	900
UO_2(3.50% enr) + 6 wt% Gd_2O_3	97% TD	900
Zircaloy-4	6.56	540
SS304L	7.94	500
Water + 900 ppm nat boron	0.8428	500
B_4C	70% TD	500

3.5.1.1 4.00% Enriched UO_2

First, you will learn to enter a basic fuel material, UO_2. UO_2 is the mostly widely utilized nuclear fuel so only UO_2 is used in this primer. UO_2 is included in the standard composition library of SCALE. The SCALE standard composition for UO_2 contains default isotopic concentrations for natural uranium. A user should specify the isotopic concentration corresponding to a particular enrichment; in this case, 4.00 wt% enriched in ^{235}U.

Start a new problem in GeeWiz by clicking the **New** button to start a new input file. Click the **General** button on the left toolbar. Enter `mini-assembly` for the **Title**. Choose `TRITON` for the **Application**;

choose `T-NEWT` for the **Sequence**; choose the `v7-238` **Cross-section Library**; click the `Parm=Check` checkbox (Fig. 3.3), and then click **OK**.

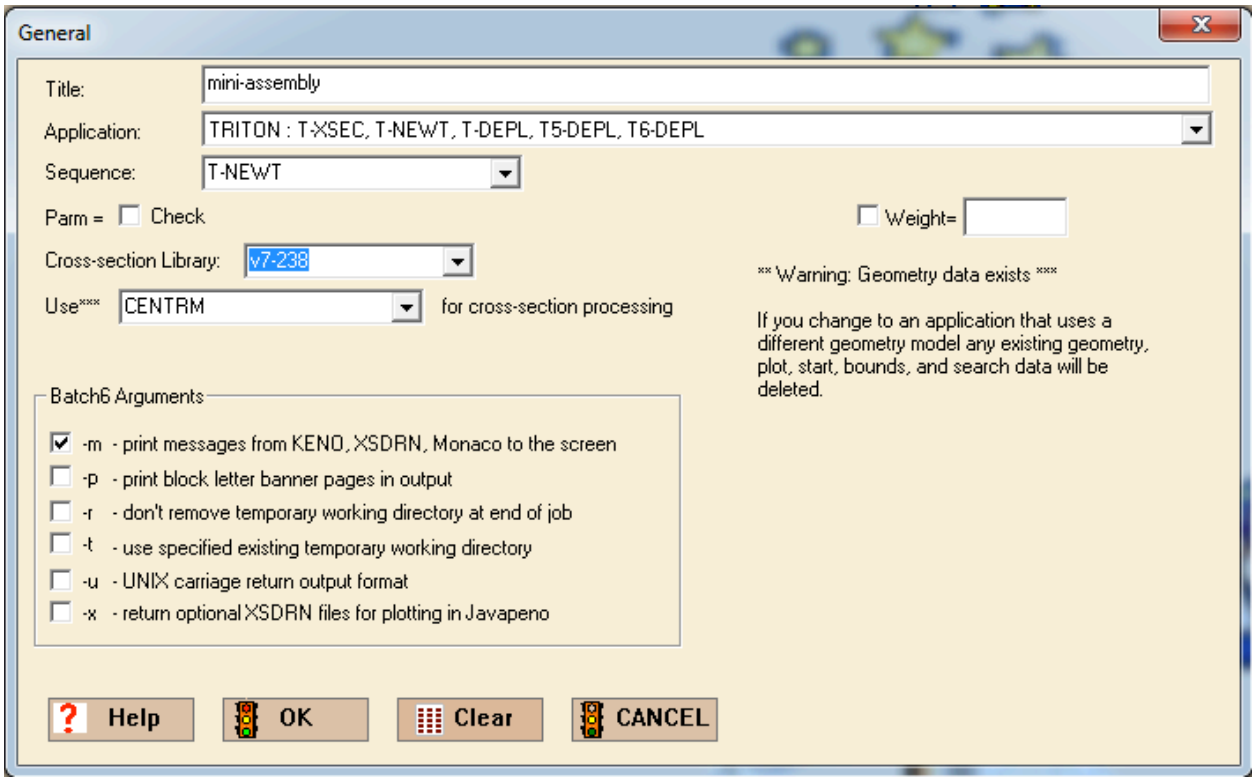


Fig. 3.3. General form for mini-assembly problem.

Then click **Compositions** from the left toolbar and the **Standard Basic Compositions** window will appear. Click **Create** to start a new composition, and then click **Basic Composition** on the **New Composition** dialog (Fig. 3.4).

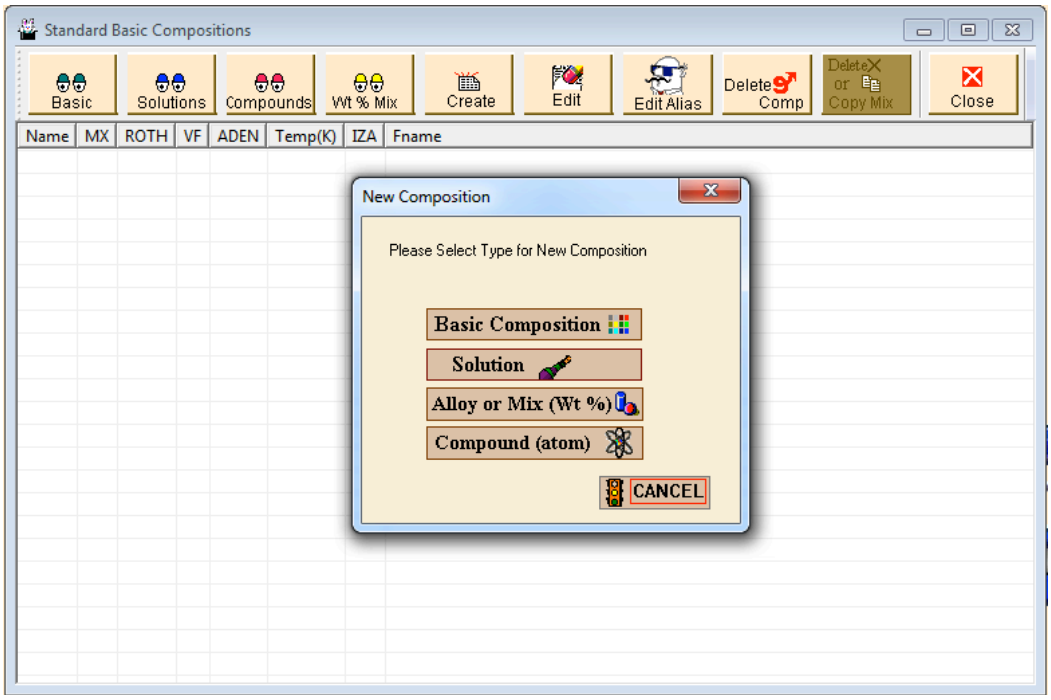


Fig. 3.4. Standard basic compositions input.

In the **Basic Standard Composition** form, leave the **Mixture** number as `1` and select `uo2` from the Composition **Name** dropdown. *Hint: The compositions are arranged in alphabetical order; you can press "u" on your keyboard to jump down the list to compositions starting with "u."* There are two different ways to specify that UO_2 has a non-default density of 97% TD. The easiest way is to use the **Density Multiplier** by entering `0.97` as the **Density Multiplier**. The other way is to input a **User Supplied** density in the **Density** box. The **User Specified** density function should be used when the fuel mixture contains additional compositions (e.g., Gd_2O_3 poison). It is important to note that the density of UO_2 decreases slightly with increasing enrichment. In this example, assume that the difference is negligible, but you might need to account for it in your calculation. Enter `900` in the **Temperature** input field. Now click **Edit Selection** under the **Isotopic Distribution** box. The **Isotopic Distribution** form will appear. Enter `4.0` for the `92235` **Wgt. Percent**, and then click inside the **Wgt. Percent** field for 92238. When the cursor is blinking in this field, click **Fill to 100%** button. GeeWiz will adjust the `92238` **Wgt. Percent** so that all weight percents sum to 100%. The **Isotopic Distribution** window should look like Fig. 3.5 when finished.

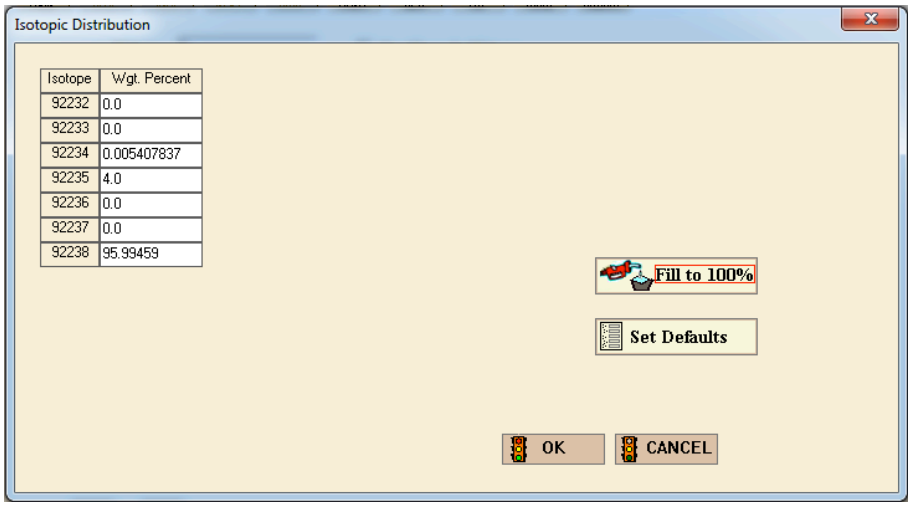


Fig. 3.5. Isotopic distribution input for mixture 1.

The final **Basic Standard Composition** form for mixture 1 (UO_2) can be found in Fig. 3.6. If your **Basic Standard Composition** window looks like this figure, click **OK**.

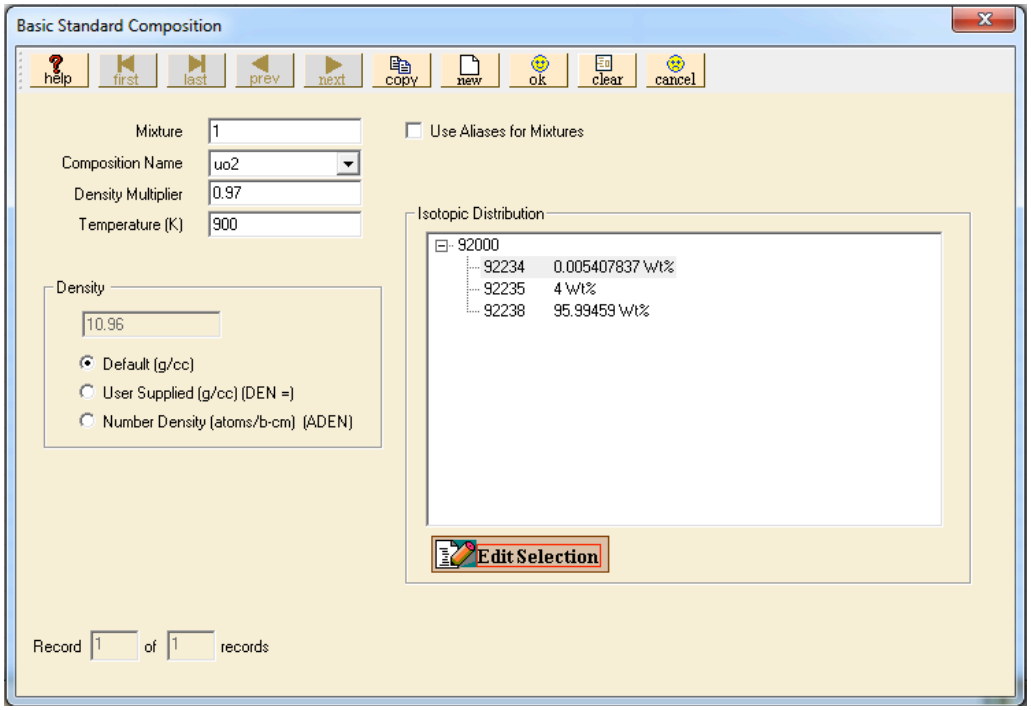


Fig. 3.6. Basic standard composition input for mixture 1.

3.5.1.2 3.50% Enriched UO_2 + 6 wt% Gd_2O_3

In this section you will learn to specify a mixture that is a combination of two SCALE standard compositions. This example will also demonstrate how to apply a burnable absorber, Gd_2O_3, to a fuel

material, UO_2. For this material, a piece of paper and a calculator, or alternatively, a spreadsheet program will be needed. It can be useful to insert these simple calculations into a spreadsheet so that they can be repeated easily anytime you need. Click **Create** from the **Standard Basic Compositions** window. Choose **Basic Composition** from the **New Composition** window. Use `2` for **Mixture**; `uo2` for **Composition Name**; leave the **Density Multiplier** of `1` for now; and use `900` for **Temperature.** Do not do anything with the **Density** at this point, rather, record the SCALE default density for UO_2 (10.96 g/cm^3). Now modify the **Isotopic Distribution** to reflect an enrichment of 3.5 percent using the method from the previous section. When finished, click **OK** to save the composition, then **Create** on the top toolbar of the **Basic Standard Composition** form to create another mixture.

Use `2` for **Mixture** again; you must manually change it from the automatically incremented value of `3`. Select `gd2o3` from the **Composition Name** dropdown; change the **Temperature** to `900`, and record the SCALE default density (7.07 g/cm^3).

Now the resulting density of the UO_2 and Gd_2O_3 should be calculated. Remember that this should be 6 wt% Gd_2O_3, and 94 wt% UO_2. This calculation can be accomplished with a simple equation:

$$\rho_{total} = wt\%_1 \rho_1 + wt\%_2 \rho_2$$
$$\rho_{total} = 0.94 * 10.96 + 0.06 * 7.07 = 10.7266 \text{ g/cm}^3$$

Also, because the fuel is 97% TD, you need to multiply the total density by 0.97. The final density is 10.4048 g/cm^3. Enter this value, `10.4048`, for both composition in mixture number 2 (UO_2 and Gd_2O_3) using the **User Specified** density functionality. Also, enter `0.94` in the **Density Multiplier** field for UO2 and `0.06` for Gd_2O_3. These values correspond to 94 wt% UO_2 and 6 wt% Gd_2O_3. The final input forms for these compositions are shown in Figs. 3.7 and 3.8.

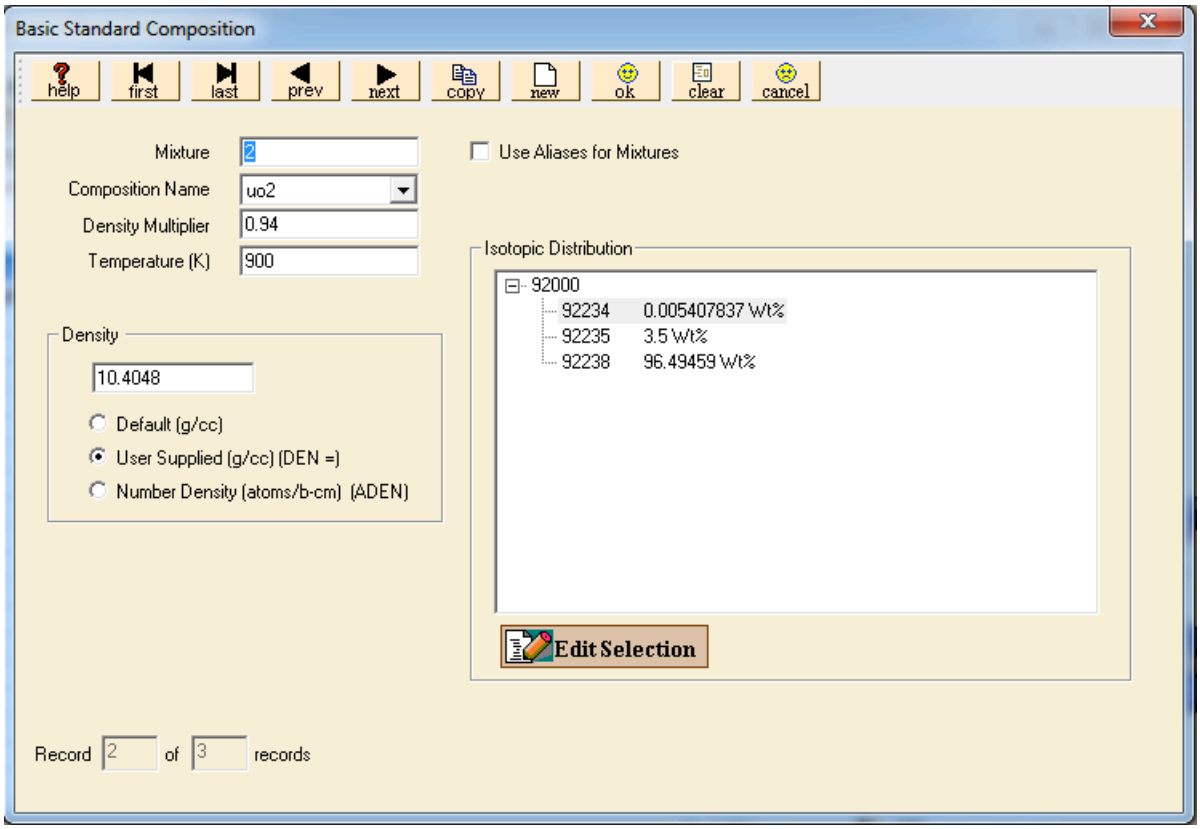


Fig. 3.7. Basic standard composition input for uo2 (mixture 2).

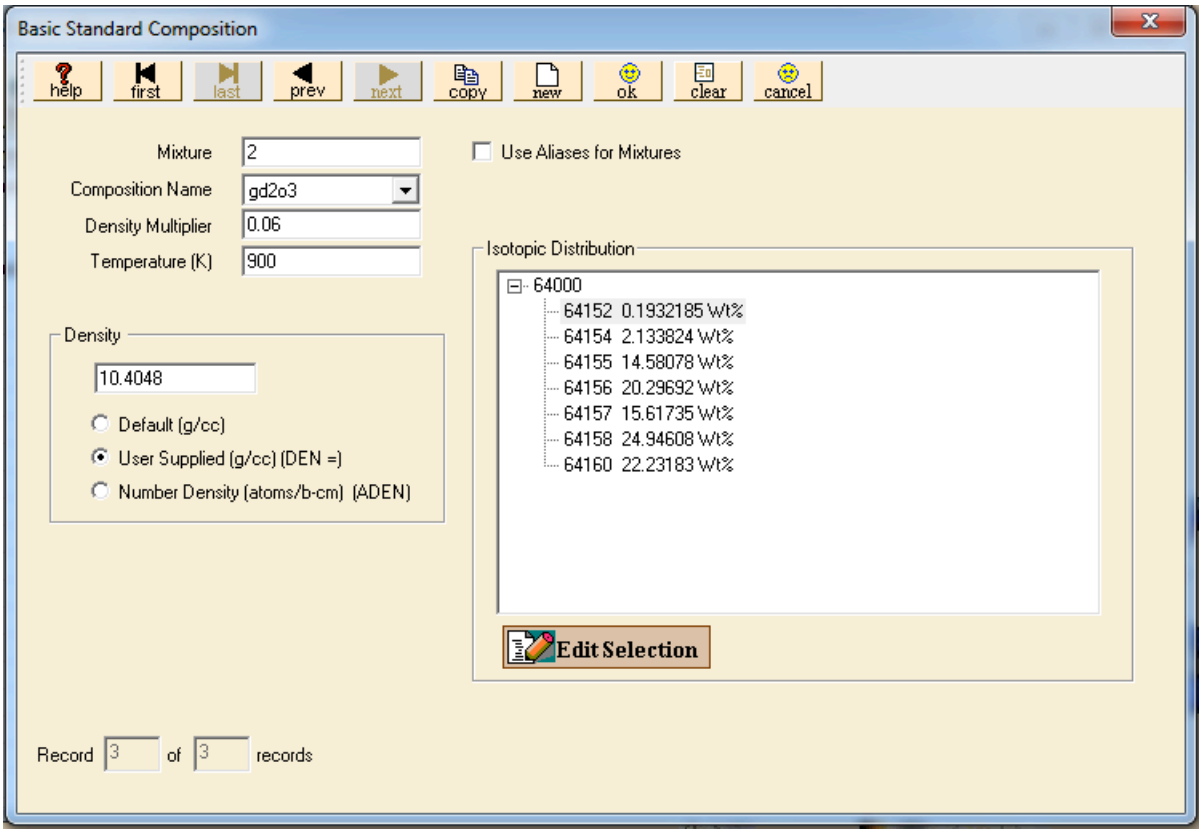


Fig. 3.8. Basic standard composition input for gd2o3 (mixture 2).

So far, two fuel materials, 4.00% enriched UO_2, and 3.50% enriched UO_2 with 6.0 wt% Gd_2O_3 have been added to the input file. The **Standard Basic Compositions** summary window should look like Fig. 3.9.

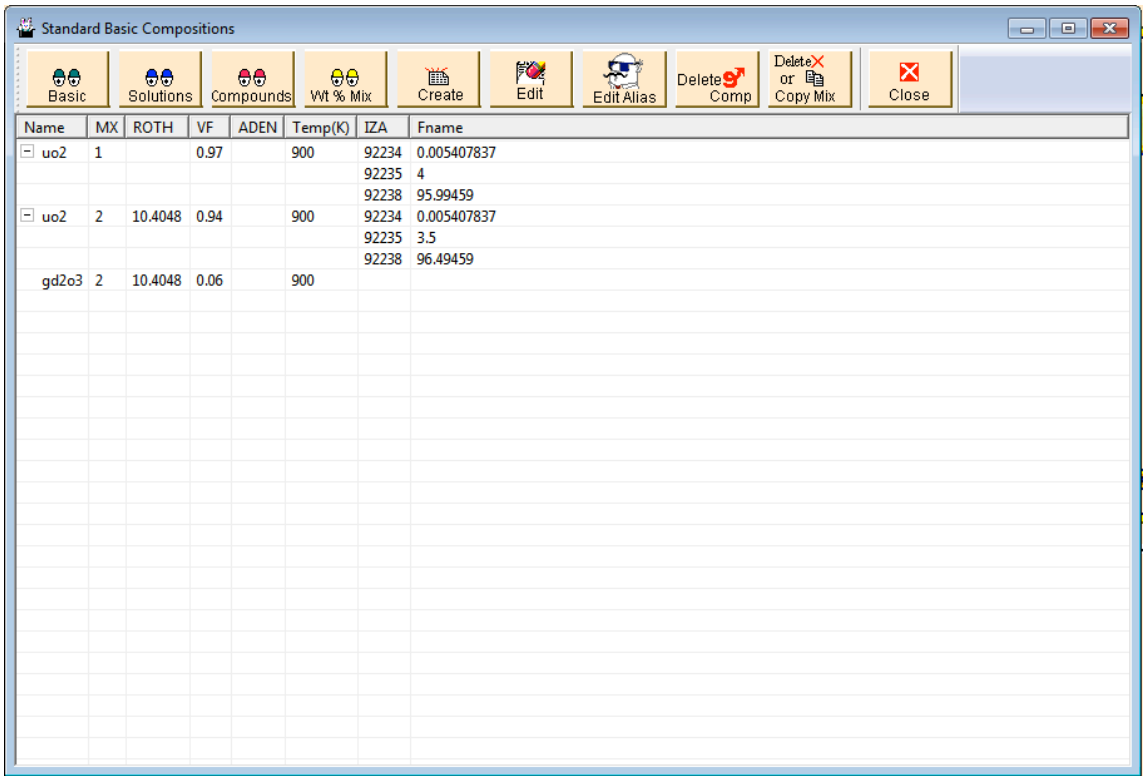


Fig. 3.9. Standard basic compositions summary for mixtures 1 and 2.

3.5.1.3 Zircaloy-4

Zircaloy-2 and Zircaloy-4 are commonly used cladding materials for nuclear fuel. They are also standard compositions `zirc2` and `zirc4` in SCALE. Click **Create**, choose **Basic Composition**, and add `zirc4` as mixture `3` at `540`K with the SCALE default density (`6.56` g/cm^3). If your Basic Standard Composition window looks like Fig. 3.10, then click **OK**.

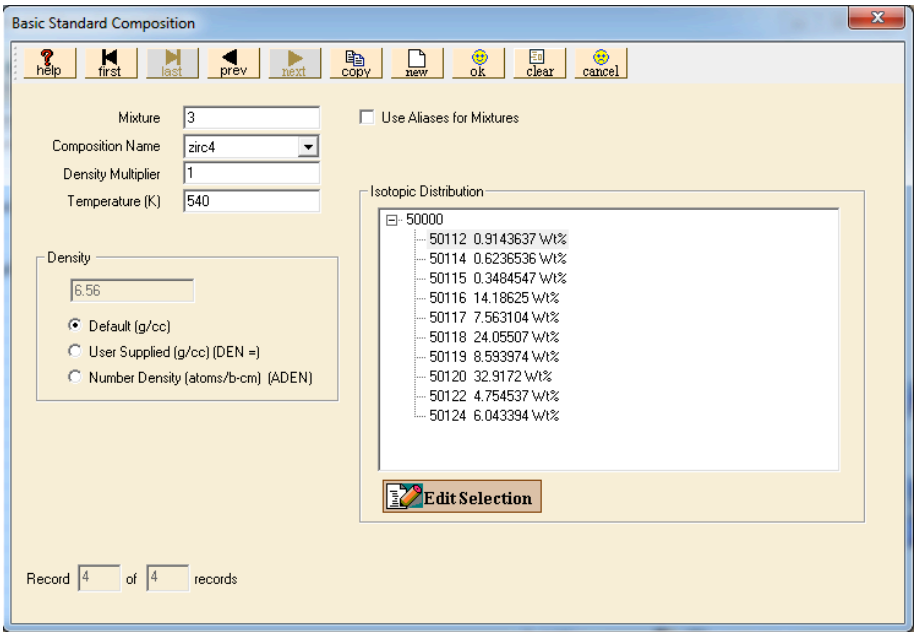


Fig. 3.10. Basic standard composition input for zirc4.

3.5.1.4 SS304L – Nonstandard Composition

Sometimes materials are specified in the design data for which a Basic Composition has not been predefined in the SCALE Standard Composition Library. Stainless steel (SS) type 304L is a low-carbon version of the standard SS type 304. It is commonly used in structural components of nuclear reactors. Currently, SCALE does not have a standard composition available for SS304L, so the data for this material must be entered manually. It is simplest to use the **Alloy or Mix (Wt%)** input for this material. This feature is convenient for alloys because alloys are usually specified as weight percentages of each element. For SS304L, use the compositions from Table 3.3. The density of SS304L is 7.94 g/cm^3.

Table 3.3. Elemental composition for SS304L[5]

Element	Wt%
Chromium	18.200
Nickel	8.500
Manganese	1.600
Silicon	0.500
Carbon	0.015
Iron	Balance

Begin from the **Standard Basic Compositions** window by clicking **Create**. You should then select **Alloy or Mix (Wt%)** from the **New Composition** window, and the **Alloy or Mixture Composition (WTPT)** window will appear (Fig. 3.11).

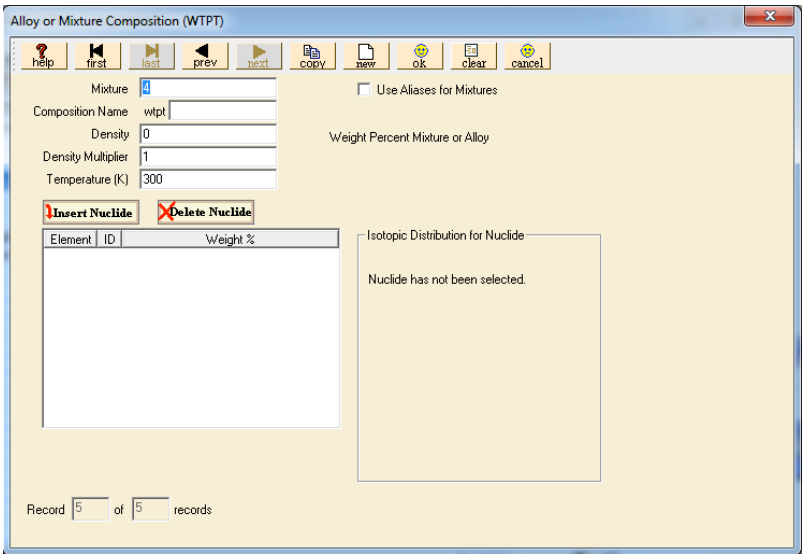


Fig. 3.11. Alloy or mixture composition form.

As before, the **Mixture** number is the number that is used to identify this material. In the **Composition Name wtpt** box, the user has a choice of what to name this material. It is always a good idea to use something descriptive. Note that this input does not change anything in the material; it is just a name that is given to the material. For this material, enter `ss304l`. Note that the prefix `wtpt` is a keyword in SCALE for user-defined compositions specified by weight percent. This prefix will be inserted by GeeWiz in the SCALE/TRITON input file, so that the mixture name will appear as `wtptss304l`. Input `7.94` g/cm^3 for the **Density** and leave the **Density Multiplier** as `1`. Change the **Temperature** to `500` K. Now click **Insert Nuclide** to enter each of the elements and their corresponding weight percentages. When you are finished specifying chromium as shown in Fig. 3.12, click **OK** to add it to the mixture.

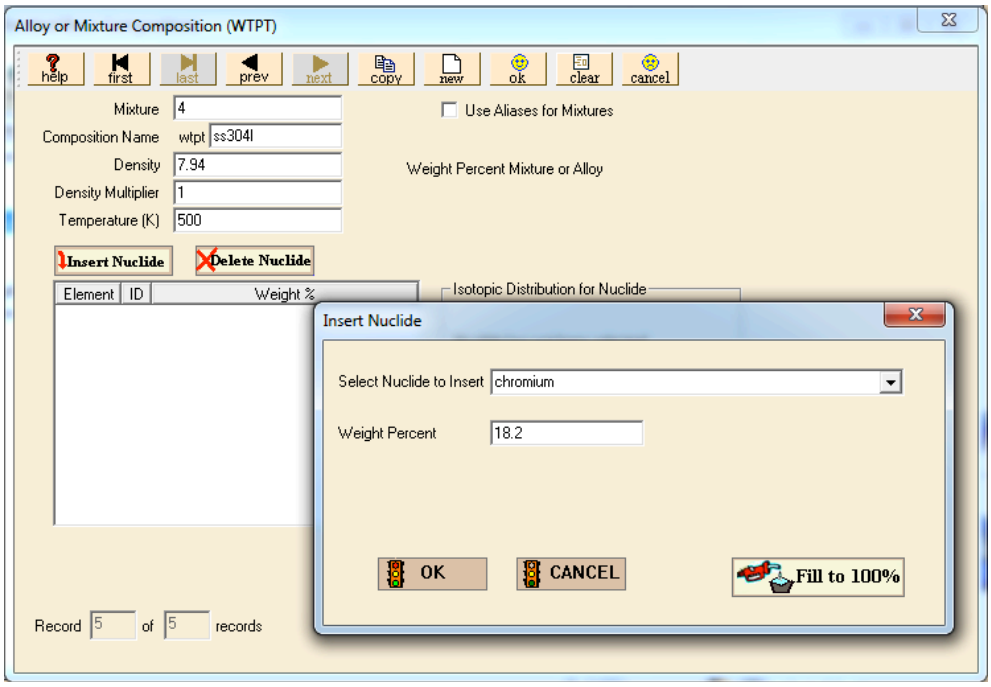


Fig. 3.12. Insert nuclide form for chromium in SS304L.

Repeat this process to enter each element and its corresponding weight percent according to Table 3.3 until you get to iron. When you are ready to enter iron, select it from the **Select Nuclide to Insert dropdown**, click inside the **Weight Percent** box, then click **Fill to 100%**. GeeWiz will complete the math, filling the rest of the material to 100% with iron as seen in Fig. 3.13.

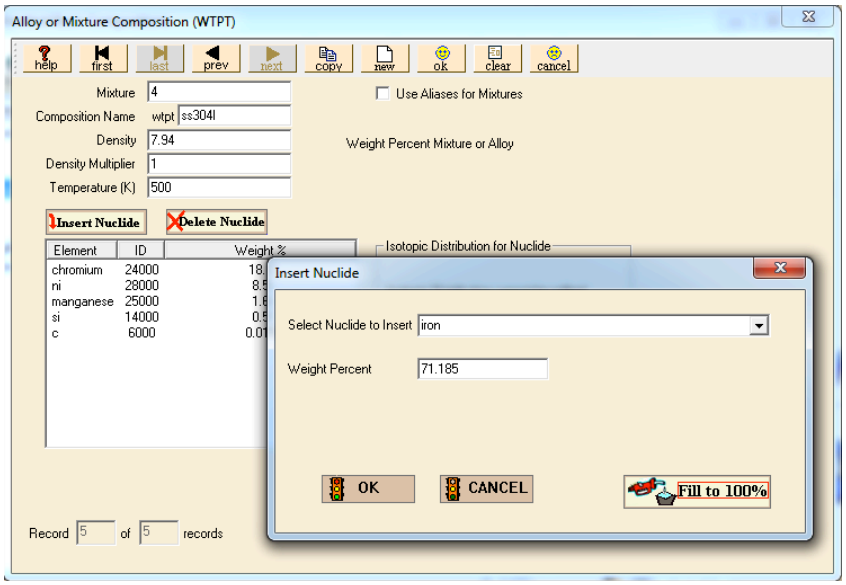


Fig. 3.13. Insert nuclide form for iron in SS304L.

The final **Alloy or Mixture Composition (WTPT)** window should look like Fig. 3.14; if so, click **OK**.

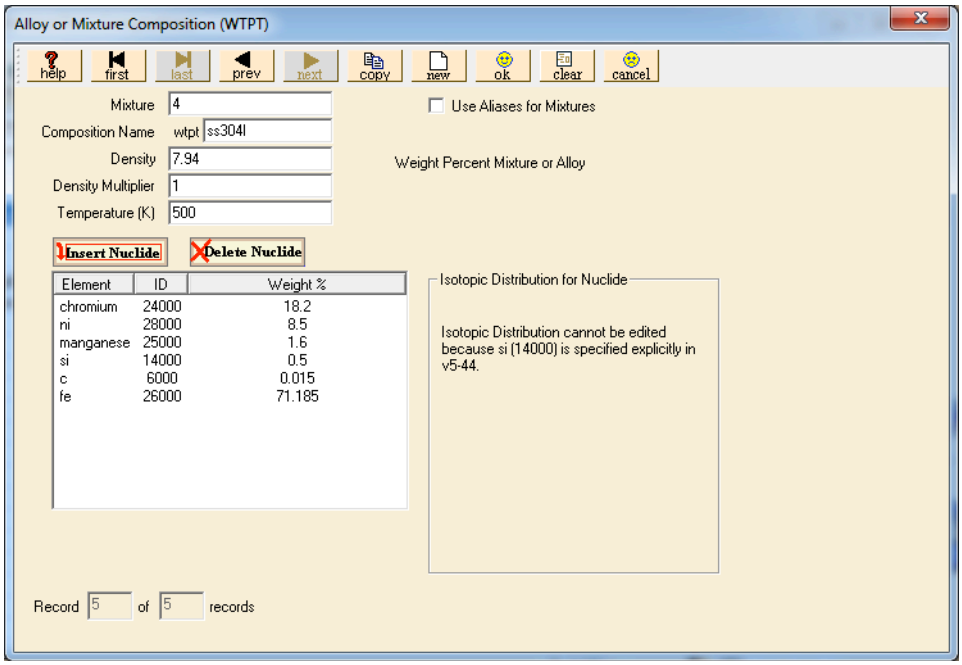


Fig. 3.14. Alloy or mixture composition form for SS304L.

In addition to the basic WTPT input, you can also change the isotopic distribution for most elements when using an ENDFB-VII library. For instance, you could use the WTPT option to specify gadolinium-bearing fuel. In that case, you would need to change the isotopic distribution of uranium to an enrichment greater than natural uranium. The WTPT input cannot use SCALE compounds, making it somewhat difficult to use for mixtures of two compounds, as is the case for gadolinium-bearing fuel ($Gd_2O_3 + UO_2$).

3.5.1.5 Water with 900 ppm Boron

The most common coolant used in nuclear reactors operating around the world is water. In the United States, commercial reactors use light water (i.e., H_2O). In other parts of the world (e.g., Canada), it is common to use heavy water (D_2O—deuterium instead of normal hydrogen) as a moderator and coolant. SCALE has standard compositions for both light water and heavy water, but this primer focuses on light water. It is important to remember that because water is a liquid, its density changes with variation of temperature and pressure. To obtain the correct density of water, you should use steam tables to calculate the density. There are a number of free online tools that you can use to calculate the properties of water, such as www.steamtablesonline.com.

To enter water with 900 ppm boron, it is best to use a similar format to that used for the Gd-bearing fuel. Use the standard compositions for boron and water, set them at the same density (the water density, 0.8458 g/cm^3), and use the density multiplier to specify boron as a 900 ppm weight fraction. Click **Create** and choose **Basic Composition**. The **Mixture** number is `5`; **Composition Name** is `h2o`; **Density Multiplier** is `1`; and **Temperature** is `500` K. Specify a **User Supplied** density of `0.8458` g/cm^3, as shown in Fig. 3.15.

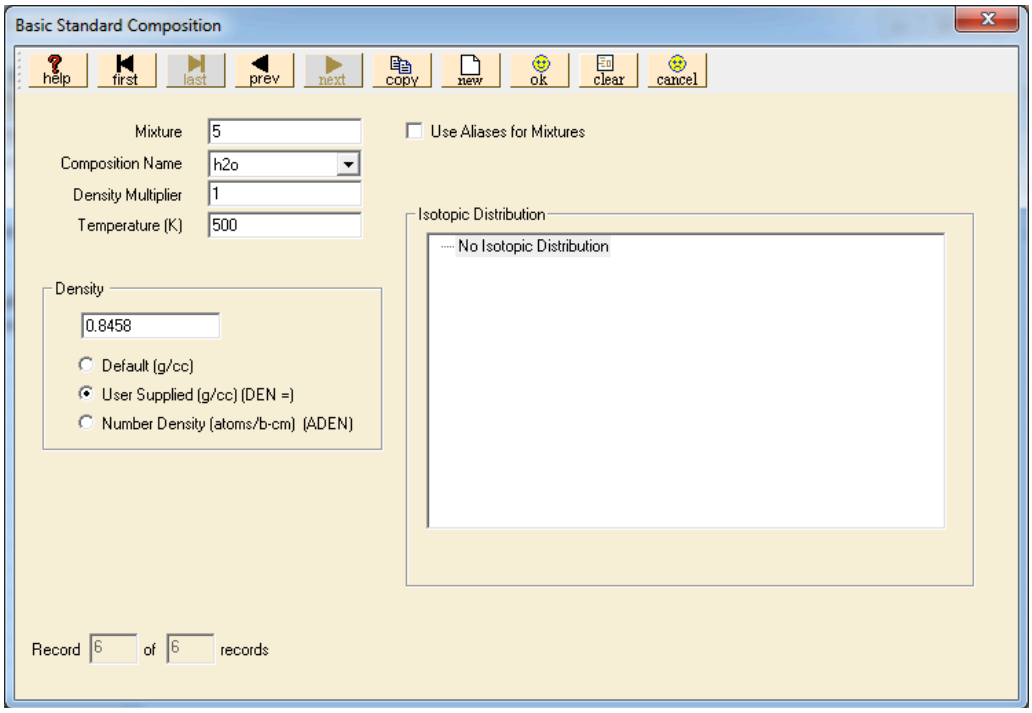


Fig. 3.15. Basic standard composition form for h2o.

Click **OK** to save the compositions; then click **Create** and choose **Basic Composition**. Change the **Mixture** number to `5` to specify that this composition is being mixed with the other composition in mixture number 5. **Composition Name** is `boron`; **Density Multiplier** is `900e-6` to specify 900 ppm; and **Temperature** is `500` K. Specifying boron in this manner ensure that boron is added in a per mass basis to be consistent with nodal simulators and thermal hydraulics codes (TRACE-PARCS). Specify a **User Supplied** density of `0.8458` g/cm^3, as seen in Fig. 3.16. Because the concentration of boron is so small, accounting for it in the density of the mixture it typically not needed; it is reasonable to simply use the water density that is obtained from the steam tables.

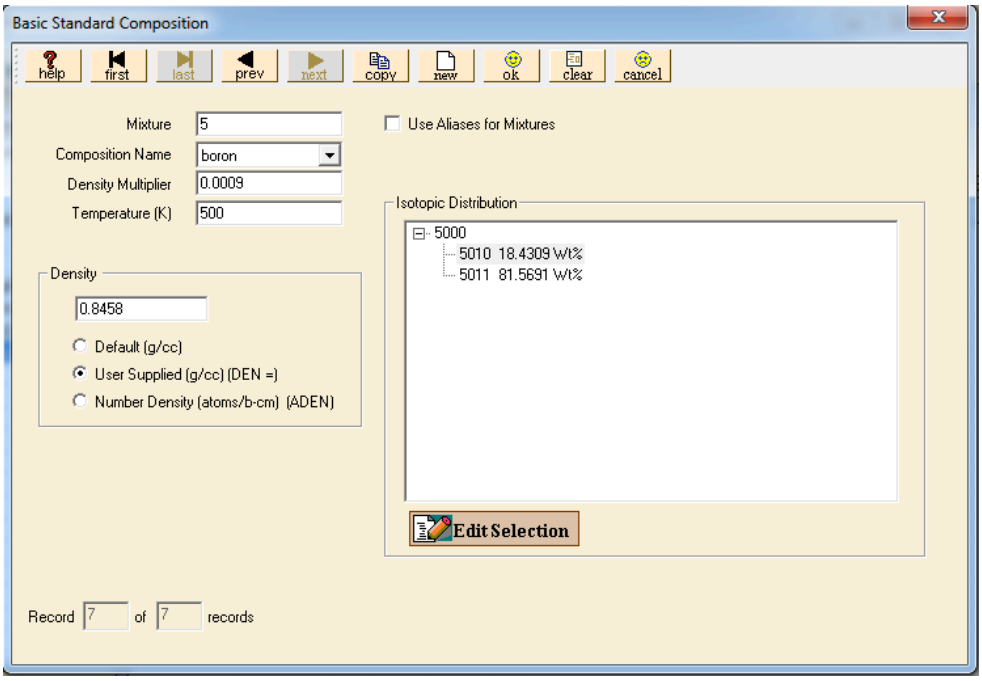


Fig. 3.16. Basic standard composition form for boron in water.

3.5.1.6 B_4C

B_4C is a commonly used as a neutron absorber in burnable poison rods. In many instances, users will need to model burnable poison or control rods, so it is important to learn how this input works. There is really no difference in the input for this material as compared to any other standard composition in GeeWiz. Most often, B_4C is in powder form instead of solid form, so a percent TD is usually specified for the density. Users will need to use either the **Density Multiplier** or a **User Supplied density** for this material.

Starting at the **Standard Basic Compositions** window, click **Create** and choose **Basic Composition**. Input **Mixture** number `6`, **Composition Name** `b4c`, **Density Multiplier** of `1`, **Temperature** of `500` K, and **User Supplied** density of 2.52*0.70 = `1.764` g/cm^3 to specify B_4C at 70% TD. **The Basic Standard Composition** window for this material can be found in Fig. 3.17.

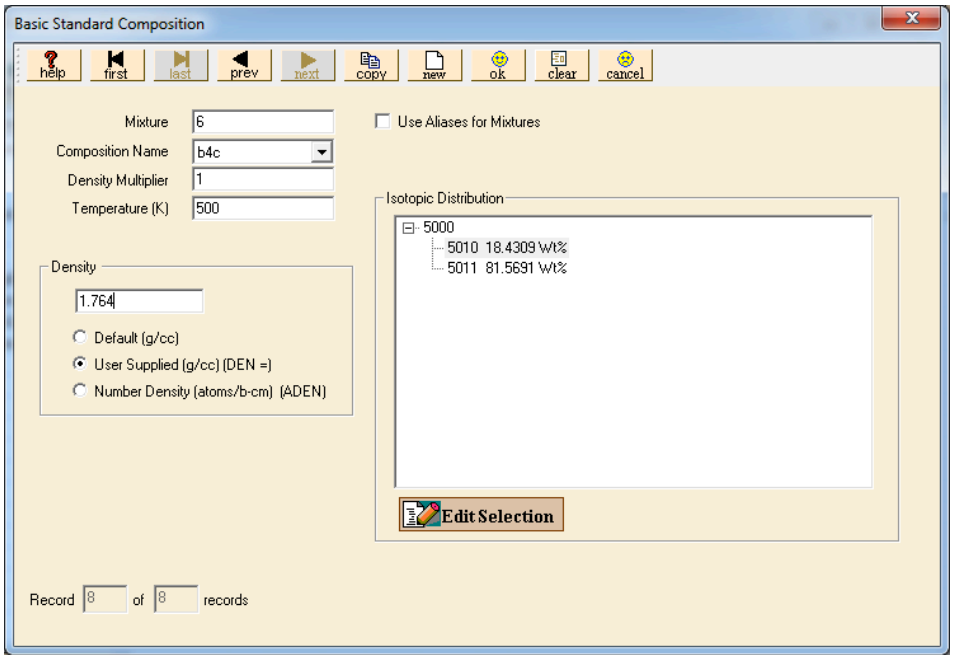


Fig. 3.17. Basic standard composition form for B_4C.

After entering all these materials, you should have a **Standard Basic Compositions** summary window with five different compositions and a **WTPT Mixtures** summary window with one composition, as seen in Fig. 3.18.

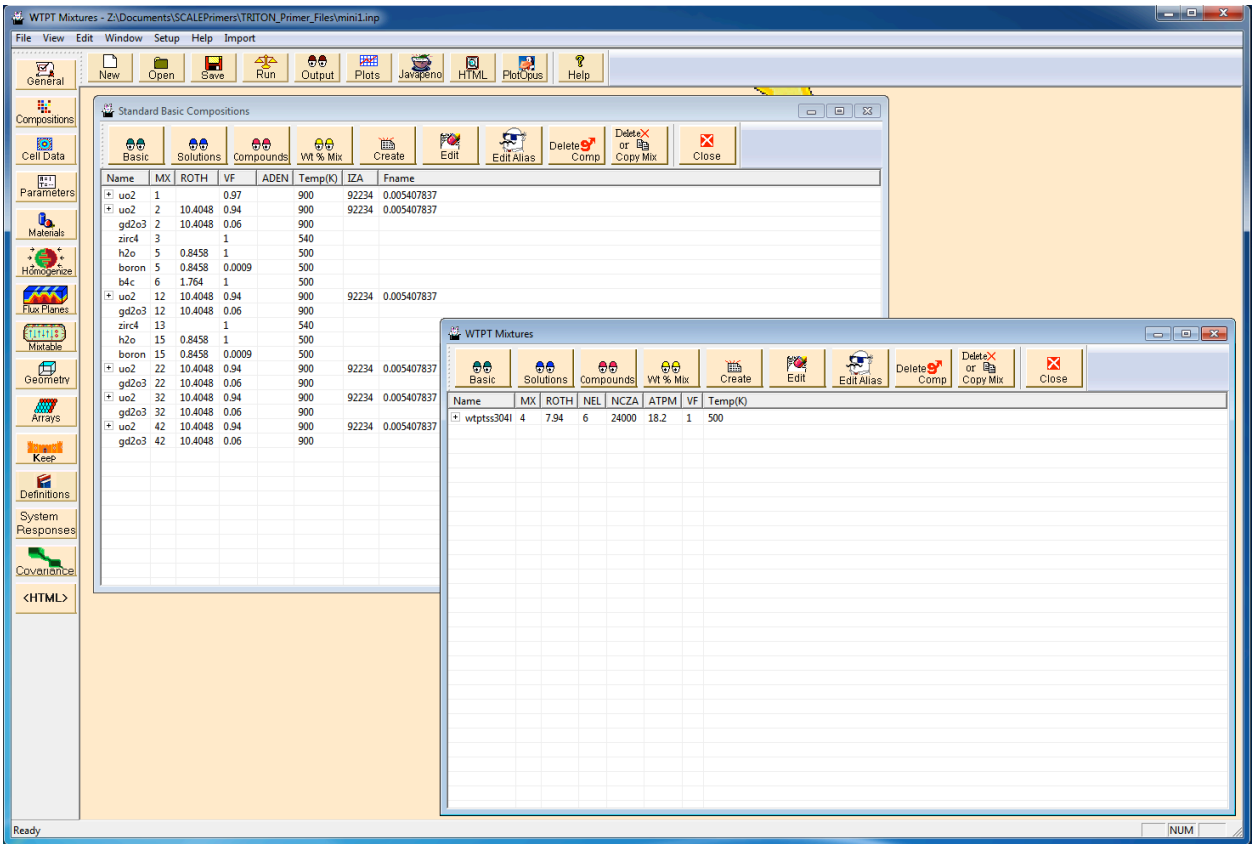


Fig. 3.18. Standard composition summary for mini-assembly problem.

3.5.1.7 The Copy Function

Before closing the compositions windows, you will learn another useful feature—how to make additional copies of a mixture. During cross-section processing, a user cannot use the same composition in multiple **Cell Data** definitions. For this reason, users will often need to specify multiple mixtures that have the same composition but different mixture numbers. For the mini-assembly problem cross-section processing, different moderator and clad mixture numbers for the normal 4.00% enriched fuel pin and the gadolinium-bearing fuel pin are needed. Users can utilize the **Delete or Copy Mix** function to copy one mixture to another. Under the **Standard Basic Compositions** window, click the clad mixture `zirc4` so that it is highlighted, and then click the **Delete or Copy Mix** button. The **Copy or Move Mixtures** window should appear. Select **Copy Mixture** and copy the mixture number `3` to mixture number `13` (Fig. 3.19) and click **OK.**

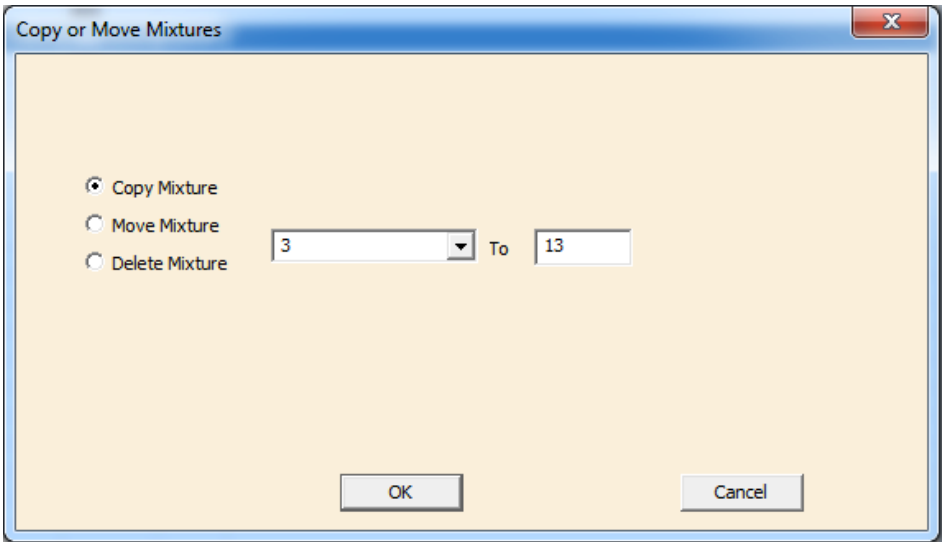
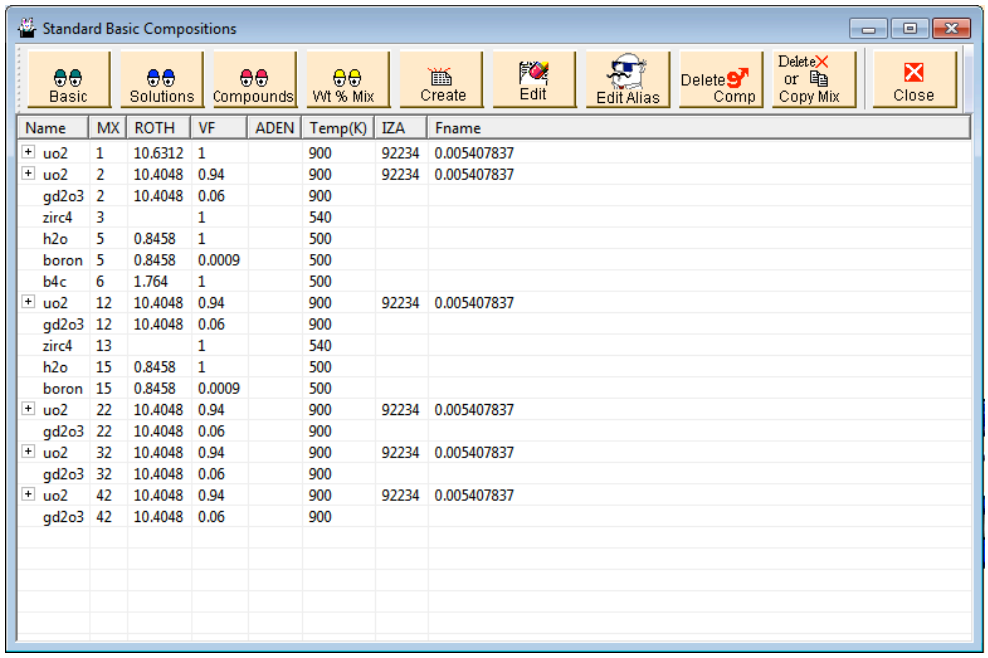


Fig. 3.19. Copy or move mixtures dialog.

Using the same process, copy mixture `5` (`h2o + boron`) to mixture `15`. *Hint: It is useful to number mixtures in a manner that is easy to follow. By naming clad and water as illustrated in this example, you know that any mixture ending in "3" is clad and any mixture ending in "5" is moderator.* You have now copied the clad and moderator to other mixture numbers that will be used during cross-section processing.

The gadolinium-bearing fuel pin requires a special type of cross-section processing in order to obtain an accurate solution. For this type of cross-section processing in this example, specification of a total of five identical gadolinium-bearing fuel mixtures that be modeled as rings in the fuel pin are required. Use the **Delete or Copy Mix** function again to copy mixture `2` (`uo2 + gd2o3`) to mixtures `12`, `22`, `32`, and `42`. GeeWiz orders the mixtures by mixture number, so the copied mixtures will not be displayed together in the **Standard Basic Compositions** window.

Your **Standard Basic Compositions** summary window should look like Fig. 3.20; if so, you may close the windows.



Name	MX	ROTH	VF	ADEN	Temp(K)	IZA	Fname
uo2	1	10.6312	1		900	92234	0.005407837
uo2	2	10.4048	0.94		900	92234	0.005407837
gd2o3	2	10.4048	0.06		900		
zirc4	3		1		540		
h2o	5	0.8458	1		500		
boron	5	0.8458	0.0009		500		
b4c	6	1.764	1		500		
uo2	12	10.4048	0.94		900	92234	0.005407837
gd2o3	12	10.4048	0.06		900		
zirc4	13		1		540		
h2o	15	0.8458	1		500		
boron	15	0.8458	0.0009		500		
uo2	22	10.4048	0.94		900	92234	0.005407837
gd2o3	22	10.4048	0.06		900		
uo2	32	10.4048	0.94		900	92234	0.005407837
gd2o3	32	10.4048	0.06		900		
uo2	42	10.4048	0.94		900	92234	0.005407837
gd2o3	42	10.4048	0.06		900		

Fig. 3.20. Standard composition summary with additional copied mixtures.

3.6 LWR CROSS-SECTION PROCESSING

One of the main functions performed in the TRITON sequences is cross-section processing. One of the options in the GeeWiz **General** form that appears on the first line of a TRITON input file is the cross-section processing option to be used. For lattice physics, there are four different options that a user can choose. Selection of a particular option depends upon the accuracy desired and the computer time that the user has available.

The cross-section processing options appear on the **Cross-Section Processing** dropdown on the **General** window of GeeWiz, as seen in Fig. 3.21 under the **Cross-Section Processing** dropdown: **NITAWL**, **CENTRM**, **NOCENTRM**, and **2REGION**. The **NOCENTRM** option is not recommended for lattice physics calculations and is not discussed in this primer.

The fastest but least rigorous solution can be obtained using NITAWL, which performs problem-dependent resonance self-shielding using the Nordheim Integral Treatment. NITAWL is only recommended for scoping calculations or calculations where great accuracy is not needed. This option can only be used with the SCALE ENDF/B-V cross-section libraries.

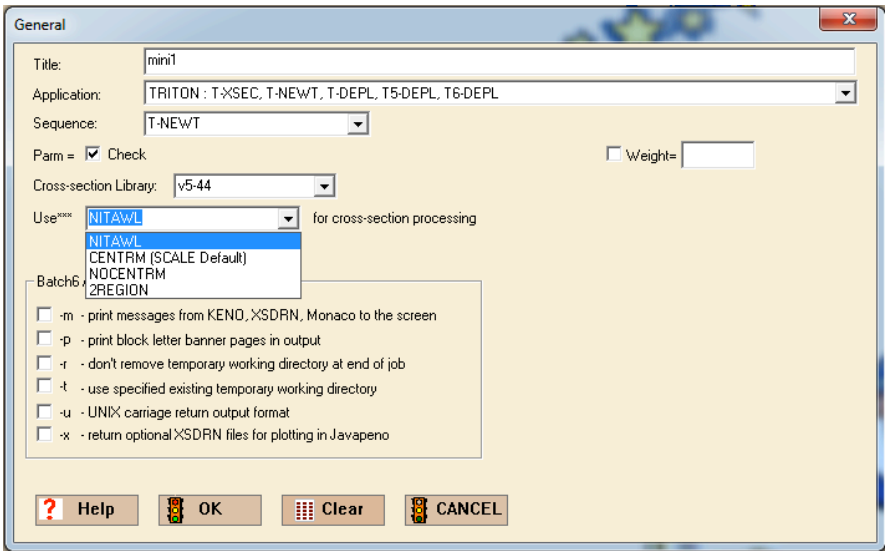


Fig. 3.21. Cross-section processing options in GeeWiz.

The most accurate solution can be obtained using CENTRM, which computes "continuous-energy" neutron spectra using various deterministic approximations to the Boltzmann transport equation in 1-D or infinite media geometry. The primary function of CENTRM is to determine problem-specific fluxes for processing resonance-shielded multigroup data. This is done by performing a CENTRM calculation for a simplified system model (e.g., a 1-D unit cell either isolated or in an infinite lattice), and then utilizing the spectrum as a problem-dependent weight function for multigroup averaging. The multigroup data processing is done by the PMC module, which reads the CENTRM continuous-energy flux spectra and cross-section data and calculates problem-dependent, group-averaged cross sections. The resulting application-specific multigroup cross-section library can be passed to higher dimensional calculations performed with a multigroup transport code such as NEWT or KENO. In this approach the multigroup cross-section processing becomes an active component in the overall transport analysis, because the group averaging is tailored specifically to the system being analyzed. For lattice physics production calculations, use of CENTRM for cross-section processing is recommended. Some advanced options, such as applying user-defined Dancoff factors to certain unit cells, are only available when using CENTRM.

A good tradeoff between accuracy and computational cost can be obtained using the 2REGION option. 2REGION is a simplified version of CENTRM that uses a two-region approximation to compute resonance self-shielding data, which is generally applicable to LWR lattices. For any integral fuel burnable absorber pins, CENTRM should be used.

For the sample problems in the primer, CENTRM will be used exclusively. There are four types of unit cells in SCALE from which to choose. You should choose the unit cell type that most accurately represents your model; a full description of all the different unit cell types is contained in the SCALE manual [SCALE manual, Section M7.3]. In this primer the focus is on the two types of unit cells that are needed in LWR lattice physics: **Lattice Cell** and **Multiregion**.

3.6.1 Lattice Cell – Fuels without Integral Fuel Burnable Absorber

In this section, additional parts required for the mini-assembly problem, mini1.inp, will be added. With this problem open, select **Cell Data** from the left toolbar, and the **Unit Cell Data** window should appear as in Fig. 3.22.

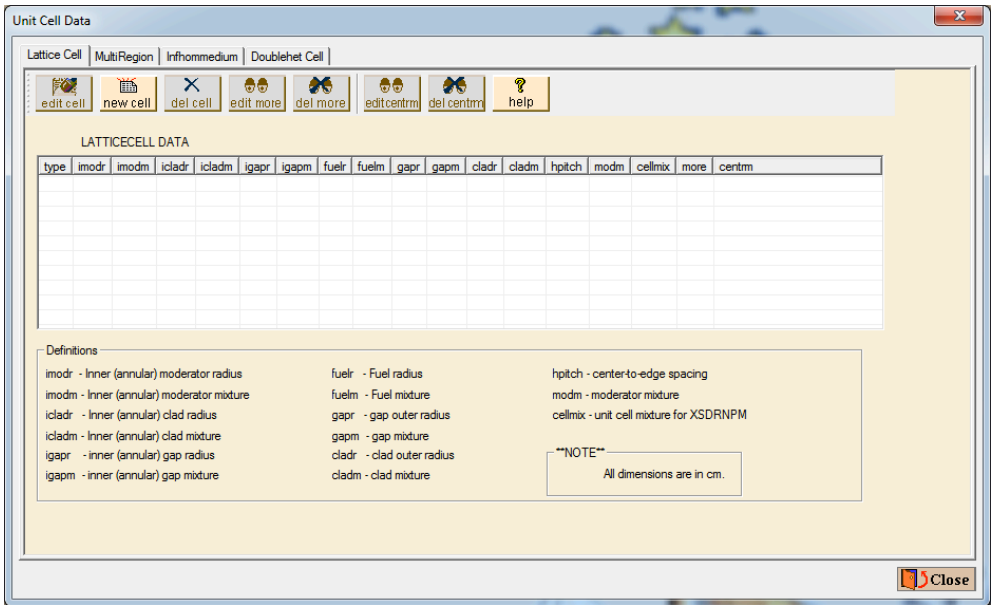


Fig. 3.22. Unit cell data window.

On the top row of tabs, make sure that the **Lattice Cell** tab is selected. Click **New Cell** to begin entry of lattice cell data. Upon clicking the **New Cell** button, a new form titled **Lattice Cell Data** appears (Fig. 3.23).

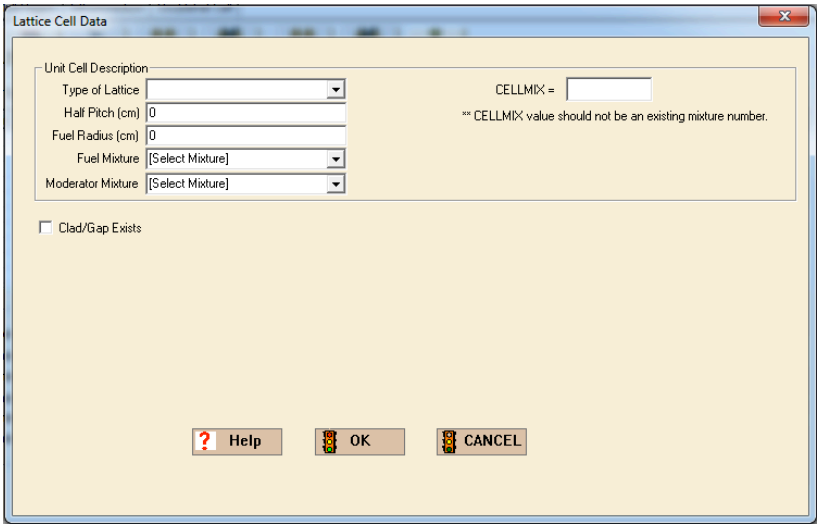


Fig. 3.23. Lattice cell data form.

For this portion, you will enter information for the primary fuel pin, 4.00% enriched UO_2 with no gadolinium. For the **Type of Lattice** dropdown, choose **SquarePitch**. The **SquarePitch** option assumes that the fuel pin lies in an infinite square lattice of identical fuel pins—this is a reasonable assumption for fuel self-shielding in most lattices, especially PWRs. For this problem, use the same geometric dimensions as the **Quickstart** problem. Enter the geometric information in the **Lattice Cell Data** window: `0.7` for **Half Pitch (cm)**, `0.5` for **Fuel Radius (cm)**, `1 uo2` for the **Fuel Mixture**, and

`5 h2o,boron` for the **Moderator Mixture**. Check the **Clad/Gap Exists** checkbox to specify `0.6` for the **Clad Radius (cm)** and `3 zirc4` for the **Clad Mixture**. The final **Lattice Cell Data** window should look like Fig. 3.24. If it does, click **OK**.

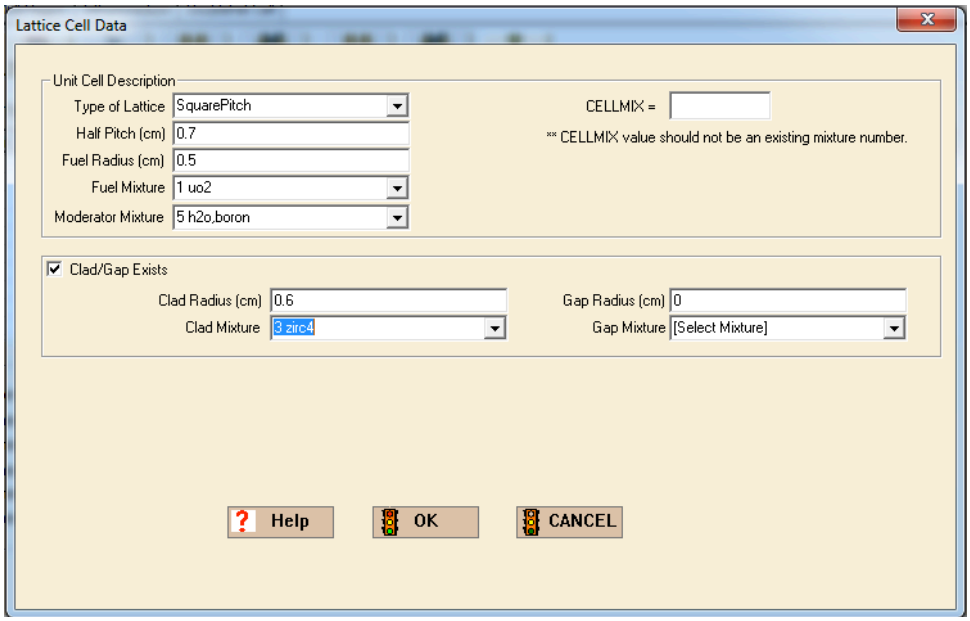


Fig. 3.24. Lattice cell data for 4.00% UO_2 fuel pin.

3.6.2 Multiregion – Fuels with Burnable Poison

The second type of frequently used unit cell in lattice physics is the Multiregion cell. The Multiregion cell is intended for unit cells whose geometry effects on the resonance self-shielding are significant but the cell cannot be defined using one of the SCALE Lattice Cell types. For fuel pins where the flux profile across the fuel region changes sharply, such as gadolinium-bearing fuel, the spatial depletion of fuel and poison is important. The user must model this type of fuel pin as a number of concentric rings with different mixture numbers. Because concentric rings of fuel cannot be modeled in a SCALE Lattice Cell, the Multiregion cell must be used. During burnup, at $t = 0$ (t = time), the mixture compositions of the fuel rings are identical. As the mixtures deplete, the gadolinium in the outer edge of the pin depletes quickly due to the high thermal flux in the moderator. With depletion, the concentration of strongly absorbing gadolinium isotopes change as a function of the radial distance from the center of the fuel pin. The gadolinium depletion continues moving toward the center of the fuel pin until the strongly absorbing gadolinium isotopes have been completely depleted.

The number of rings that should be modeled in a gadolinium-bearing pin can vary. For example, more rings will be needed for fuel pins with a larger radius. Using at least five rings is generally recommend, but more may be necessary for your problem of interest.

For this example, the five different mixtures that were copied for gadolinium-bearing fuel (mixtures 2, 12, 22, 32, and 42) to construct a five-region fuel pin will be used to construct the fuel pin. The different UO_2+Gd_2O_3 fuel regions will be constructed so that they are equal-area rings. The clad and water moderator will also be modeled as concentric rings. The user will need to calculate the radii that

correspond to the equal-area rings in the fuel and the ring of moderator that is equivalent to the lattice pin pitch. The radii of the fuel rings can be calculated with a simple equation:

$$r_n = \sqrt{\frac{n}{N} R^2} \quad , \tag{1}$$

where

r_n is radius for ring n,
R is the outermost radius (fuel radius),
n is ring number (1, 2, 3 …, N), and
N is the number of rings.

The ring of moderator should correspond to the same area as the true unit cell. In the true unit cell, the dimensions are fuel pin radius, 0.5 cm; clad radius, 0.6 cm; and unit cell pitch, 1.4 cm. Using these dimensions, calculate the radii for the five equal-area rings, and the radius of the moderator region (Fig. 3.25). The purpose of adding rings to a fuel pin that contains burnable absorber is only to obtain a detailed radial distribution of fuel and absorber concentration as a function of depletion. For the **Multiregion** specification used in lattice physics, only concentric cylinders may be used, so the outer boundary is modeled as a cylinder with a white boundary rather than infinitely reflected lattice cell.

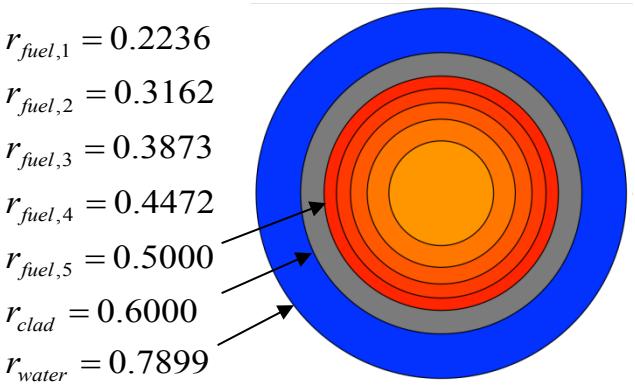


Fig. 3.25. Rings for Multiregion cell for UO_2+Gd_2O_3 fuel pin.

To enter the data, click on the **Cell Data** button on the left toolbar. The **Unit Cell Data** window will appear. Click on the **MultiRegion** tab along the top of the window, and then click **New Cell**. The **Multiregion Cell Data** window should appear, as seen in Fig. 3.26.

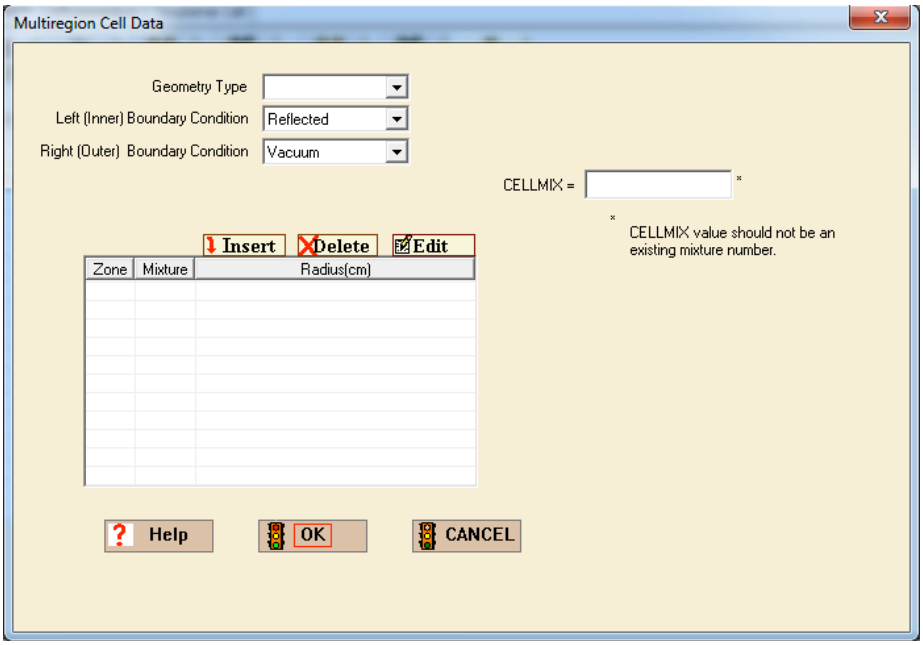


Fig. 3.26. Multiregion cell data form.

Select `Cylindrical` for **Geometry Type,** `Reflected` for **Left Boundary Condition,** and `White` for **Right Boundary Condition**. Because the outer boundary is a cylindrical cell, a reflective boundary condition is a poor choice, so select the white boundary condition. For the white boundary condition, the incoming angular fluxes are each set equal to a single value (isotropic) chosen such that the net flow across the boundary is zero. The white boundary provides isotropic return at a boundary and is suitable for all geometries. Although the white boundary condition is in a sense nonphysical on a curved surface, it is the best possible approximation of a reflective or repeating boundary condition, such as the case for a square lattice cell. Using a reflective boundary conditions on a cylindrical surface can result in inaccurate fluxes.

You can click **Insert** to begin inserting mixtures and radii for each ring in the **Zones** form (Fig. 3.27). Insert the **Radius** `0.2236` and the **Mixture** `2` for the innermost fuel ring. Click **OK** to save the data for the first zone. Repeat the process for each zone, working from inner to outer rings. Use mixtures `12`, `22`, `32`, and `42` for the remaining fuel rings. For the clad and moderator use mixtures `13` and `15`, because mixtures `3` and `5` have already been used in the lattice cell for the 4.00% UO_2 fuel pin.

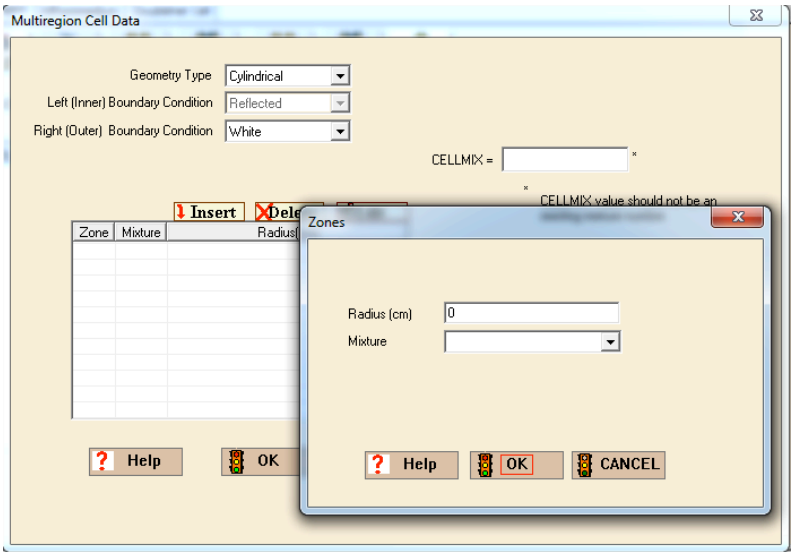


Fig. 3.27. Zones form for multiregion cell.

The final **Multiregion Cell Data** window should look like Fig. 3.28. If everything looks correct, click **OK**.

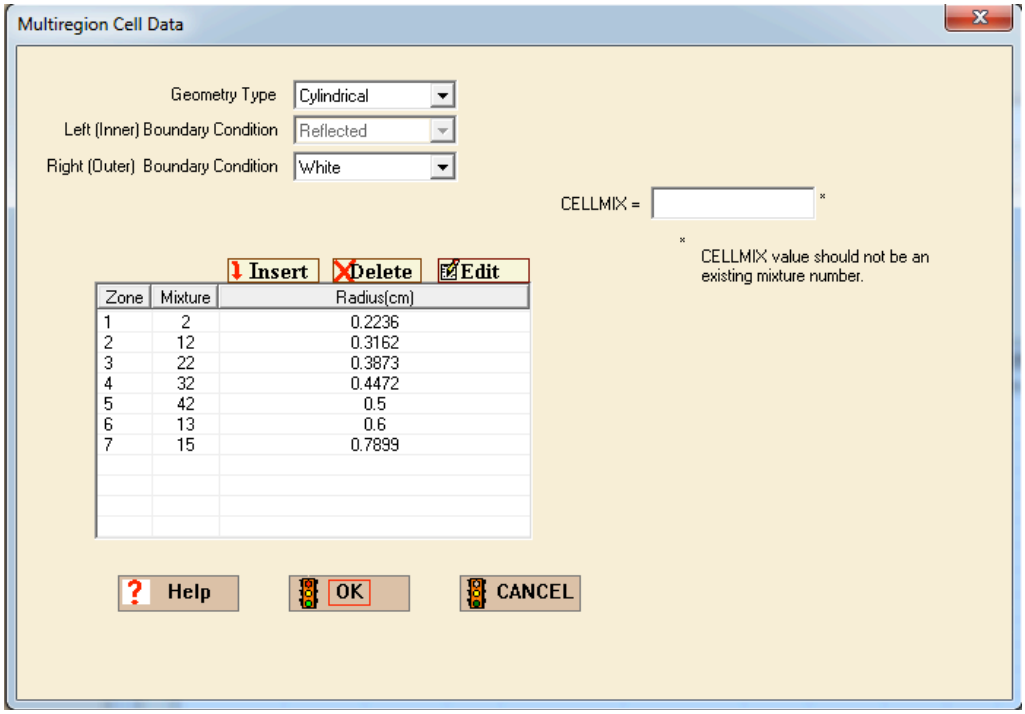


Fig. 3.28. Multiregion cell data for UO_2+Gd_2O_3 fuel pin.

You should now have a **`squarepitch` Lattice Cell** and a **`cylindrical` Multiregion** unit cell in the **Unit Cell Data** window. To view the summary of these two fuel pins, you can use the tabs along the top of the **Unit Cell Data** window to switch between **Lattice Cell** and **MultiRegion**. You can also click the

small "+" at the far left of the **Multiregion** cell to see the region details. If the **Lattice Cell** and **Multiregion** tabs look like Figs. 3.29 and 3.30, click **Close** to close the **Unit Cell Data** window.

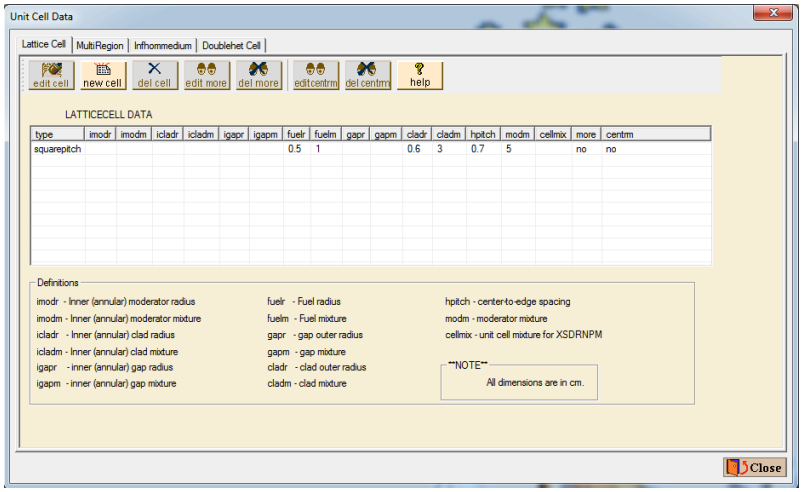


Fig. 3.29. Lattice cell data summary.

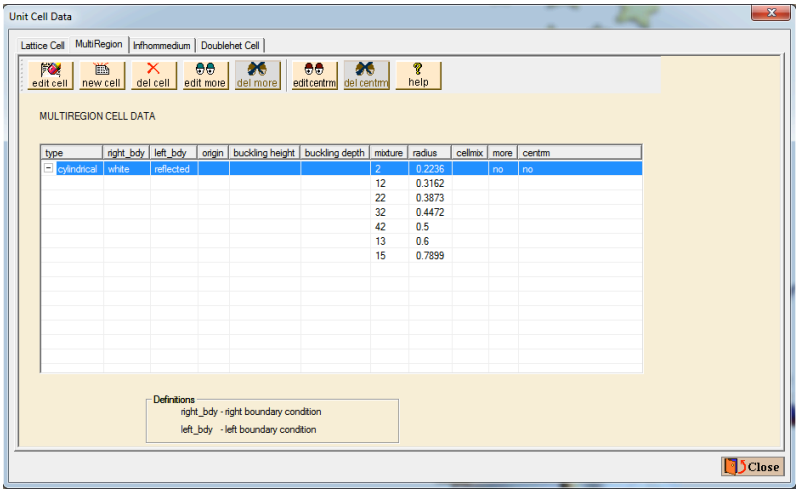


Fig. 3.30. Multiregion cell summary.

3.7 SUMMARY

This section has helped you to accomplish these objectives:

- identify the cross-section libraries available for lattice physics analyses
- use the GeeWiz user interface to provide input data on more complicated basic standard compositions, compounds, and alloys

- use the GeeWiz user interface to enter unit cell data for the two primary unit cell types used in lattice physics modeling

Now that you have gained experience with material input and cross-section processing, more complicated 2-D geometries features, such as lattices, will be introduced.

4. GEOMETRY INPUT

In the previous section you saw how to add various mixtures and learned how to process fuel for resonance self-shielding. You saw how to use the GeeWiz user interface to insert data on elements, isotopes, compounds, and solutions. You also saw how to use GeeWiz to set up two unit cells that are commonly used in lattice physics to perform resonance self-shielding of fuel mixtures. This section explains the commands used to construct basic TRITON/NEWT shapes such as cylinders, cuboids, and arrays of these shapes and how to use these shapes to build lattice models. In this section, the basic shapes in available in NEWT will first be introduced and then these shapes will be used to build a simple mini-assembly model.

After having completed the *Quickstart* section and the previous chapters, you should understand how to use GeeWiz to build a basic input file. In this section, the pace of the instruction increases by skipping steps that the user should have learned in the previous sections. At various points during setup, and at the end of each geometry option that is presented, a screenshot is presented so that the user may verify his input. In the previous sections, every step was documented with a screenshot, but in this section some basic steps will be skipped to make the document flow more smoothly and progress at a faster pace.

4.1 WHAT YOU WILL BE ABLE TO DO

- use GeeWiz to describe some basic TRITON/NEWT geometry shapes
- understand how units are created (including intersections of shapes)
- identify and change the location of the origin for shapes and units
- create simple arrays consisting of a single unit
- create arrays with multiple units

4.2 BASIC GEOMETRY SHAPES

TRITON/NEWT has a set of five basic shapes that a user can utilize to construct models. You have used GeeWiz to enter data for cylinders and cuboids in the *Quickstart* section. These and the other basic shapes are described in Table 4.1 and illustrated in Fig. 4.1.

Table 4.1. Basic shapes

Keyword	Description
CUBOID	Box or rectangular parallelepiped
CYLINDER	Right circular cylinder
HEXPRISM	Box with top and bottom faces that are hexagons
RHEXPRISM	Hexprism rotated 90 degrees about *z*-axis
WEDGE	Right-triangular prism with five faces

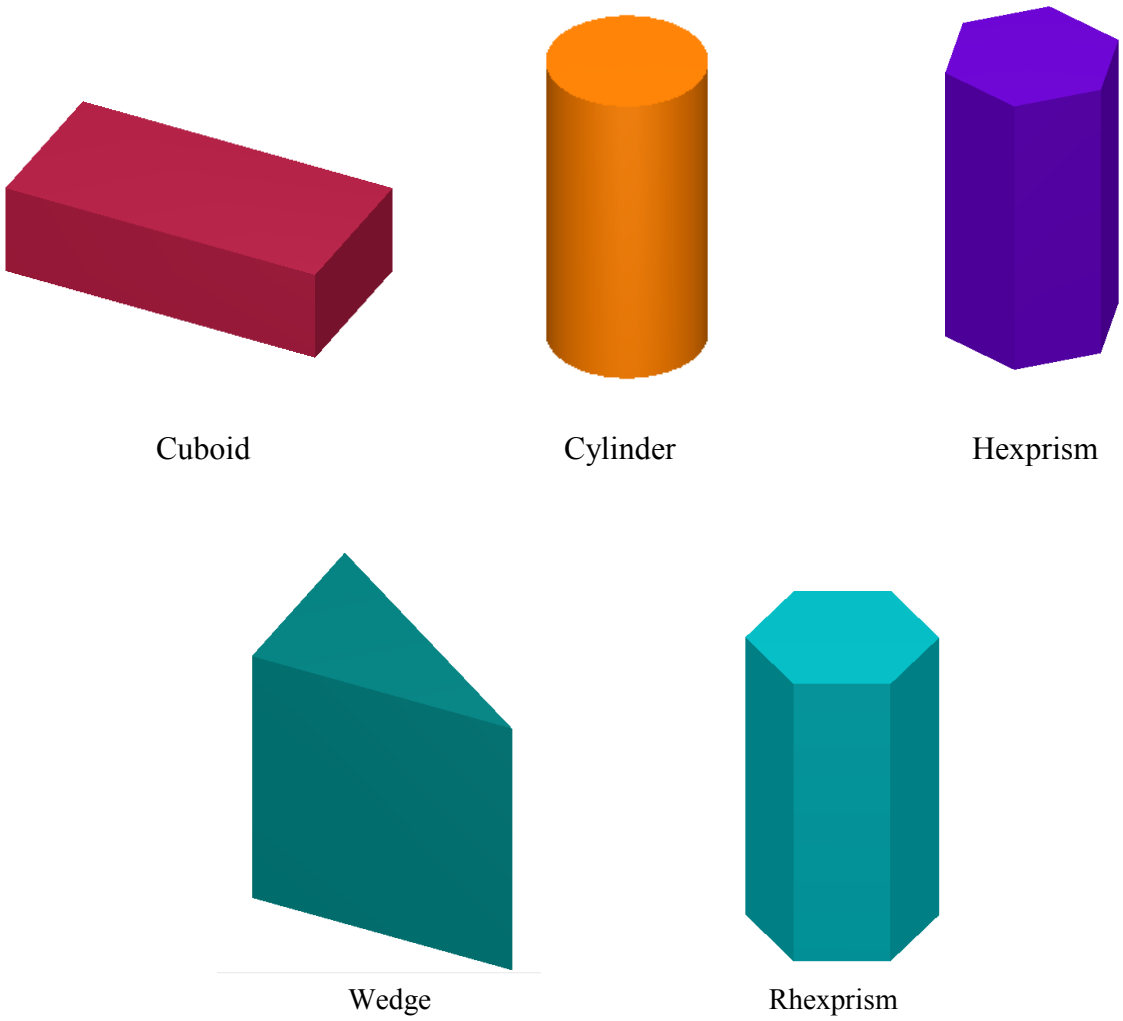


Fig. 4.1. 3-D representations of TRITON/NEWT 2-D geometry shapes.

The shapes in Fig. 4.1 are slightly deceiving—the true shapes in TRITON/NEWT are 2-D because NEWT is a 2-D transport solution code. Typical LWRs use a rectangular-based grid of fuel pins known as a lattice. This primer focuses on the geometric shapes input required to construct LWR lattices—cuboids, cylinders, and arrays. The basic geometric shapes available in TRITON/NEWT are shown on the right side of the **Geometry** input form (Fig. 4.2). Each shape has a set of quantitative information needed to describe its size and location. Selection of a geometry option will bring up the associated shape input form, where the appropriate information is entered. Figure 4.3 shows the input form for cuboids; the other input forms require similar information. All geometry input forms have figures that show a *z*-dimension because these same input forms are also used for 3-D SCALE transport modules. For SCALE/TRITON-NEWT, the *z*-dimension boxes are absent. A description of the options for each basic shape can be found in the SCALE manual [SCALE manual, Section F21.3.6]. Additional information may be entered to translate (relocate the origin), rotate, or truncate the shape (using a chord). These options are discussed later.

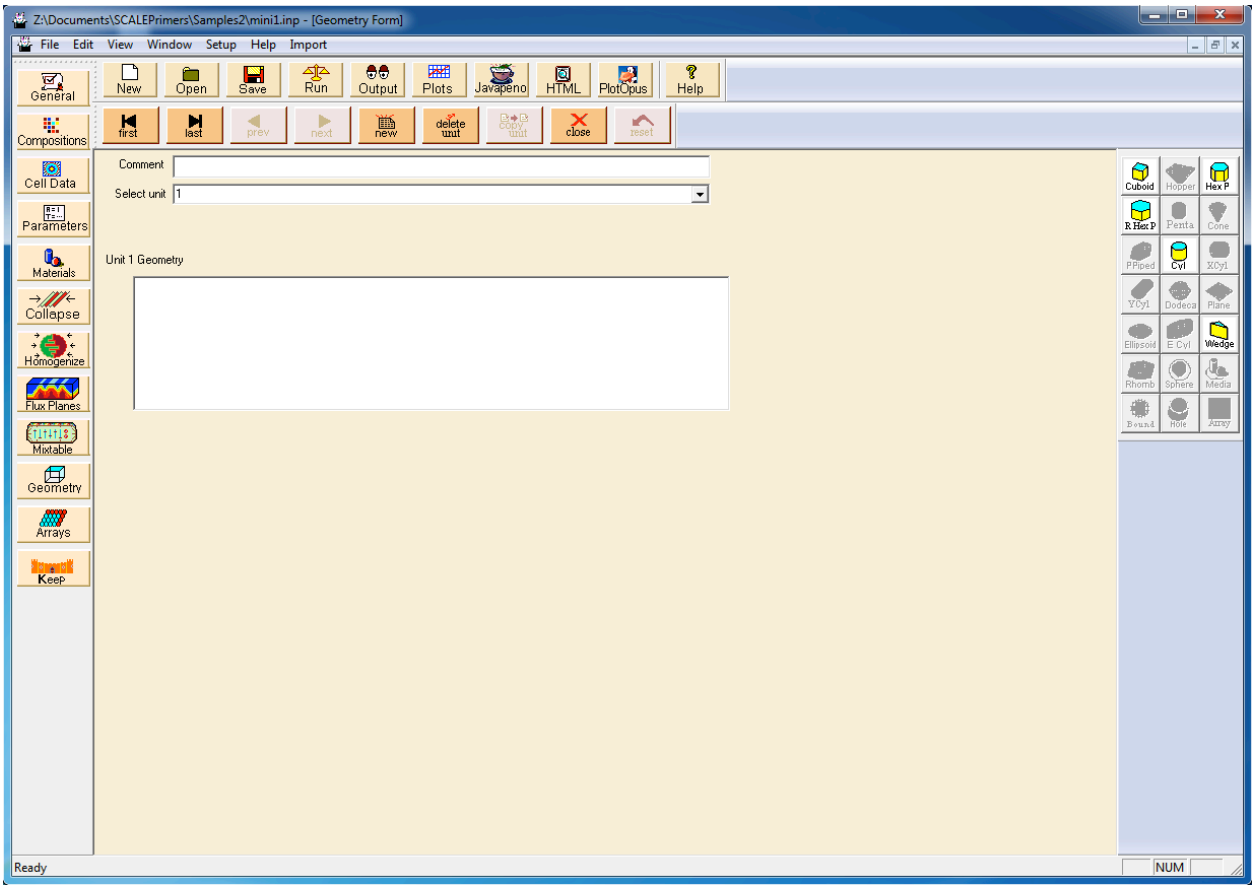


Fig. 4.2. Geometry form.

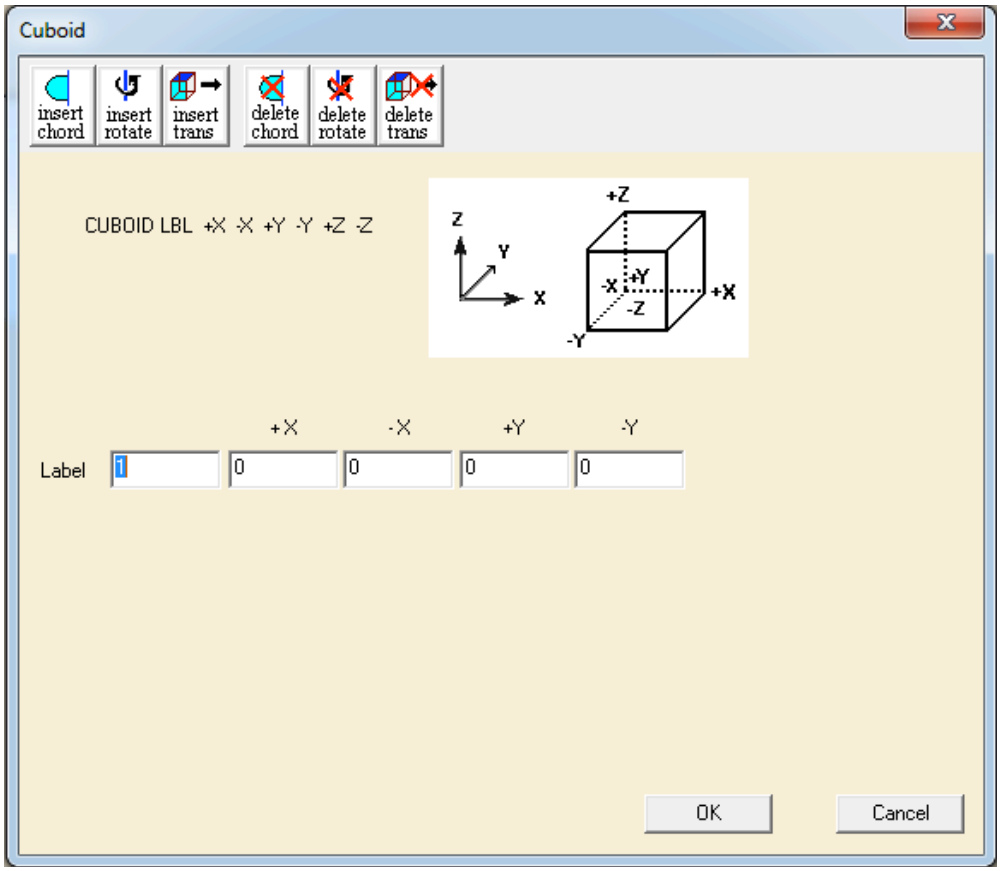


Fig. 4.3. Cuboid input form.

4.3 TRITON/NEWT BASIC GEOMETRY RULES

There are seven basic rules that guide the geometry input to TRITON/NEWT. These rules also apply to SCALE/KENO-VI geometry.

1. *Volumes are built in sections called units.* Each unit is independent of all other units and has its own coordinate system.

2. *Units are built using regions*. Regions are made using the TRITON/NEWT geometric shapes. The unit boundary must fully enclose all defined regions in the unit. See example in Fig. 4.4.

3. *Regions may share boundaries and may intersect.* See examples in Fig. 4.4.

4. *Regions may be rotated or translated.* See example in Fig. 4.4.

5. *A hole is used to place a unit within a region in a different unit.* The hole must be completely contained within the region and should not intersect other holes. As many holes as required may be placed in a unit.

6. *An array is an ordered stack of units.* The touching faces of adjacent units in an array must be the same size. Multiple arrays may be placed directly into a unit by placing them in separate regions. Arrays can also be placed within a unit using holes.

7. *A global unit that encloses the entire system must be specified.* All geometric data used in a problem are correlated by the global unit coordinate system.

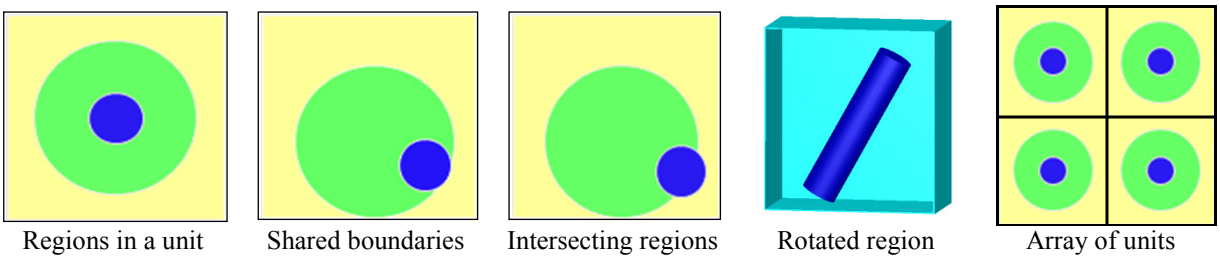


Fig. 4.4. Examples of TRITON/NEWT basic geometry rules.

4.4 GEOMETRIC ARRANGEMENTS

Geometric arrangements in TRITON/NEWT are achieved in a manner similar to using building blocks. Each building block is called a unit. Units are constructed of combinations of basic shapes. These shapes can be placed anywhere within a unit as long as they do not intersect the outer boundary of the unit.

A unit can be thought of as a container that encloses a number of shapes. In its simplest form, a unit encloses one or more basic shapes with the smallest shape completely enclosed by the next smallest and so on. The most common use of units in lattice physics is to construct one or a number of different individual fuel cells. Normal fuel cells consist of a central fuel cylinder, followed by gap and clad cylinders, and surrounded by water that fills a cuboid.

4.5 THE BASIC UNITS FOR LWR LATTICE PHYSICS

To illustrate how to enter TRITON/NEWT geometry information for nested regions in a unit via GeeWiz, the mini-assembly problem will be continued. To start, some simple unit cells will be defined in the same way as was done in the *Quickstart* section.

4.5.1 Empty Water Cell

A commonly used unit in lattice physics is a simple square (**Cuboid**) unit filled with water. These units are used to represent a vanished fuel rod (vacant pin position) in a lattice, a large water rod (BWR lattice), or control rod guide tube (PWR lattice). The water cell in this example will span from -0.7 to 0.7 on both the *x* and *y*-axes, as seen in Fig. 4.5.

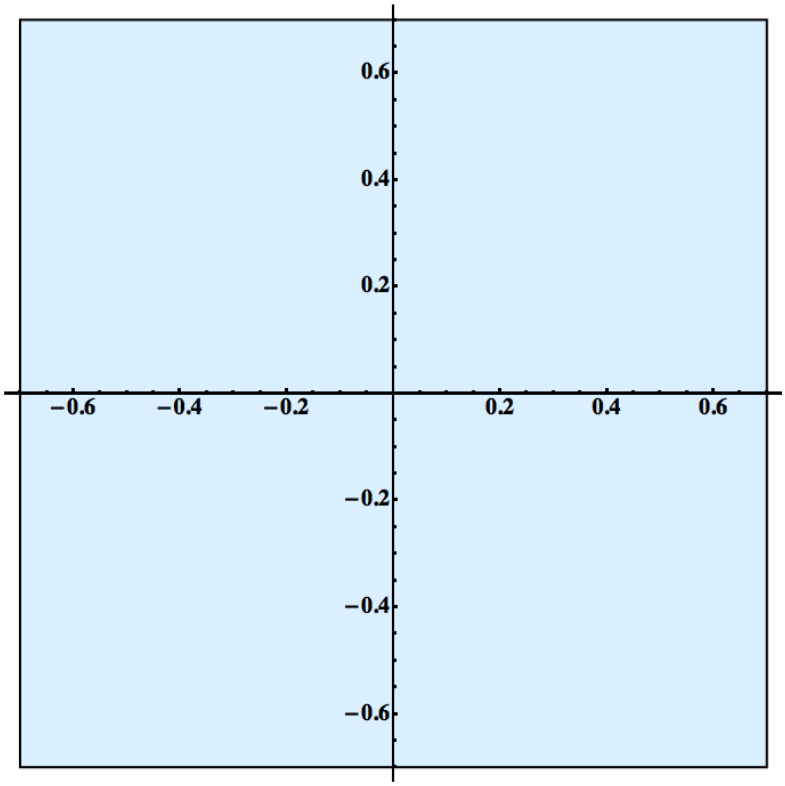


Fig. 4.5. Water cell.

To construct the water cell, click the **Geometry** button along the left vertical toolbar. In the **Comment** box, insert `Water Box`, and in the **Select unit** box, specify this unit as unit `10`. Then click the **Cuboid** button to create a cuboid for the water box. Enter `10` for **Label** and `0.7`, `-0.7`, `0.7`, `-0.7` for **+X, -X, +Y**, and **–Y** (Fig. 4.6). This specifies a box centered at (0,0) and with a side length of 1.4 cm. Note that GeeWiz uses two shortcut keys to reduce the effort for the user. The "P" key (i.e., "P" = "plus/minus") alternates positive and negative repeating values in consecutive fields, while the "R" key (i.e., "R" = "repeat") repeats the value in the current field. For example, you can enter 0.7 for +**X** and then press the "P" key three times to fill the **–X**, **+Y**, and **–Y** fields for this cuboid.

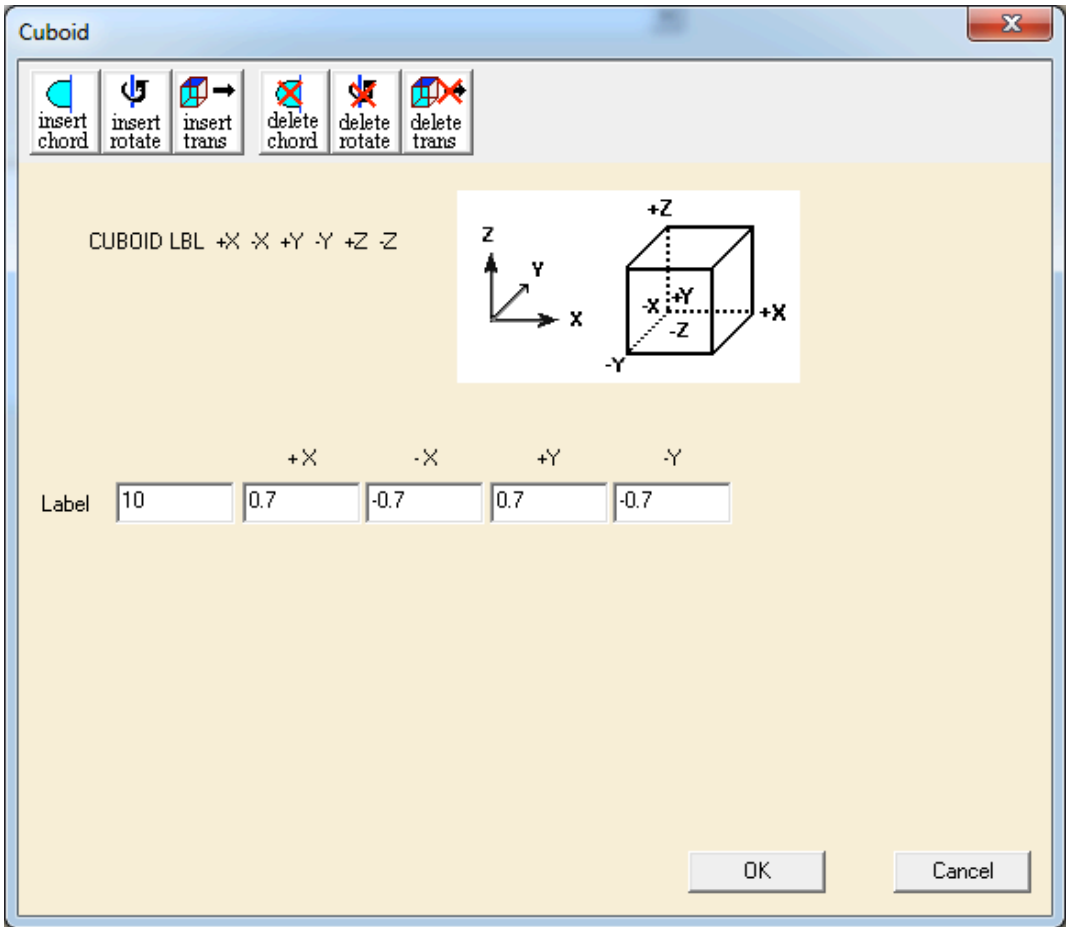


Fig. 4.6. Cuboid geometry form.

Click **Media** to fill the cuboid with borated water, `5 h2o,boron` by using the **Inside** button. Once finished, click **OK**.

Now use the **Bound** button in the right-hand toolbox to select the boundary. Select `cuboid 10` from the **Select Geometry** window and click the **Inside** button to specify that the entire unit is bounded by `cuboid 10`. Also specify a `4×4` grid. Once finished, click **OK**. The final Media Form for unit 10 can be found in Fig. 4.7. You now have a cuboid filled with borated water named unit 10 that can be placed in an array or placed into the geometry using a hole. This unit will be used as a cell in the mini-assembly array. The final Unit 10 geometry form can be found in Fig. 4.8.

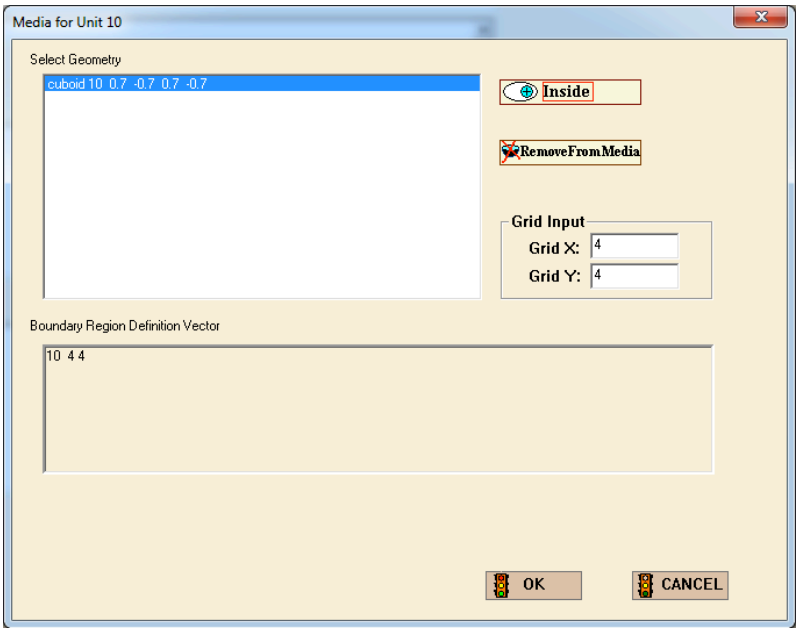


Fig. 4.7. Media form for unit 10.

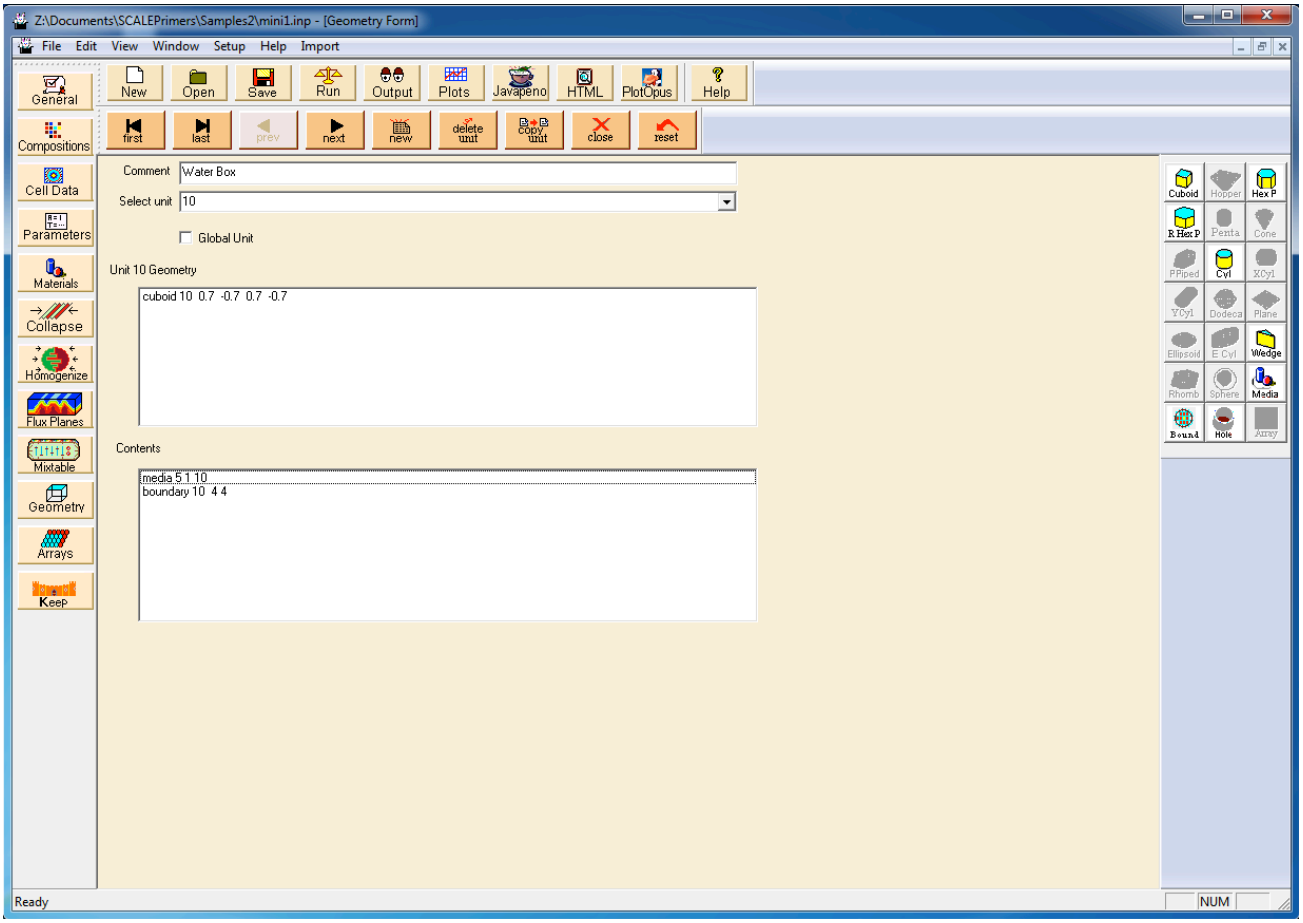


Fig. 4.8. Final geometry for the water cuboid.

4.5.2 4.00% Enriched Fuel Cell – LWR-Specific Content

The basic fuel pin unit cell is the primary building block for lattice physics analyses using TRITON/NEWT. These fuel cells are placed together in an array to construct a lattice containing one or many different types of fuel, water, or other cells. Before constructing the mini lattice array, you should specify the two types of fuel cells that will be used. For the first fuel pin, using two cylinders and a cuboid, specify a fuel cell centered at (0,0) with a pitch of 1.4 cm, a fuel radius of 0.5 cm, and a clad radius of 0.6 cm, as seen in Fig. 4.9.

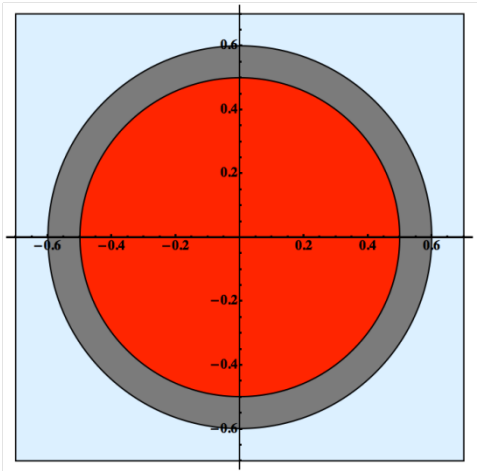


Fig. 4.9. Pin cell with a fuel radius of 0.5 cm, a clad radius of 0.6 cm, and a pitch of 1.4 cm.

Following the same procedure as the *Quickstart* section will make this unit cell relatively simple to set up. Click **New** to specify a new unit, enter `1` in the **Select unit** box, and provide a descriptive **Comment**, such as `fuel, 4.00% enr`. Now add two **Cylinders** with radii of `0.5` and `0.6` cm, and a **Cuboid** centered at `(0,0)` that has `0.7`, `-0.7`, `0.7`, `-0.7` for **+X, -X, +Y**, and **–Y** . You should then specify **Media** for each cell. Specify the 4.00% enriched fuel (`1 uo2`) for the inner cylinder, Zircaloy-4 for the larger cylinder (`3 zirc4`), and borated water (`5 h2o,boron`) for the remaining cuboid. The media statements should be:

```
media 1 is inside cylinder 10;
media 3 is outside cylinder 10 and inside cylinder 20;
media 5 is outside cylinder 20 and inside cuboid 30.
```

Also specify `cuboid 30` as the unit boundary with a `4×4` grid. The **Geometry Form** for this unit can be found in Fig. 4.10.

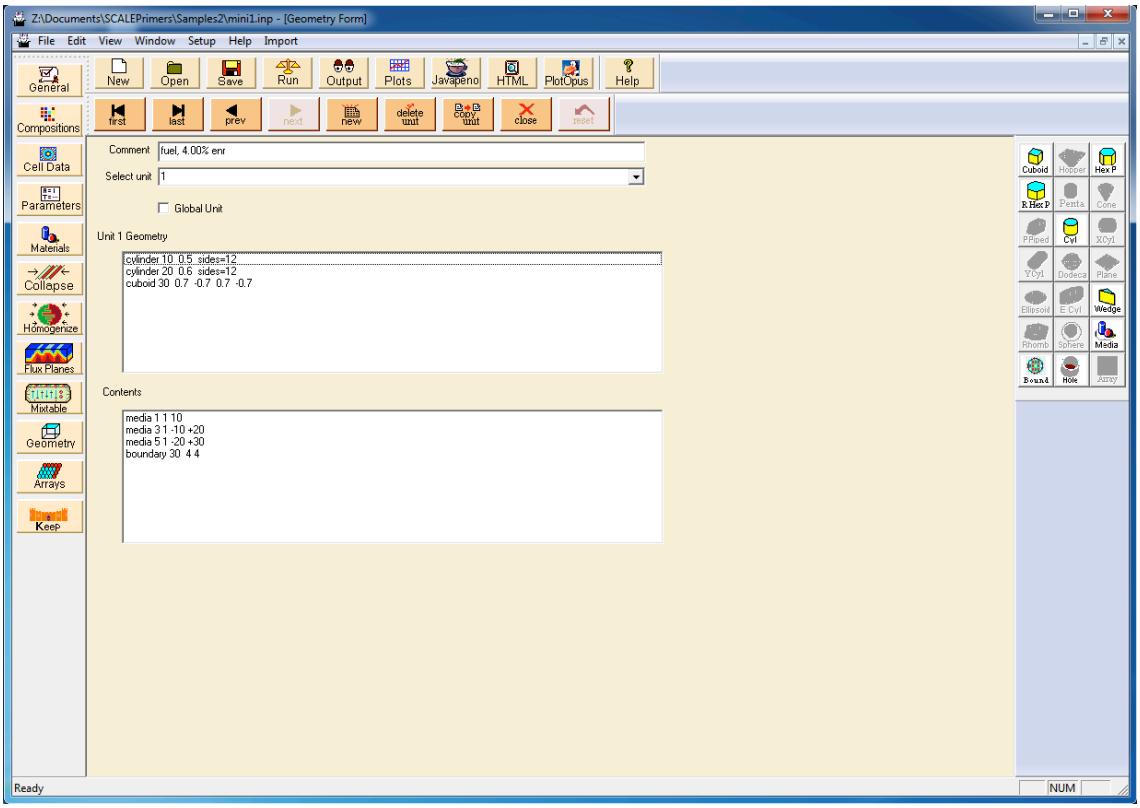


Fig. 4.10. Final Geometry Form for unit 1, 4.00% enriched fuel pin.

4.5.3 3.50% Enriched UO_2 + 6 wt% Gd_2O_3 Fuel Cell – LWR-Specific Content

For the gadolinium-bearing pin, special treatment needs to be taken in the geometric model. Similar to the **Cell Data**, this fuel pin should be constructed with a number of concentric rings to account for the strong spatial depletion of fuel and poison. You should specify the same radii for this fuel cell as were specified in the **Cell Data** statement for the multiregion cell. You should add all of the rings that were specified in the **Cell Data** in order to maintain consistency between the **Cell Data** resonance self-shielding calculation and the NEWT transport calculation. The outer fuel rings will deplete faster than the inner rings; to obtain accurate fluxes and isotopic concentrations for each ring as a function of space and time, you must specify each ring in the geometry. For this unit, you will specify five **Cylinders** for the fuel pin, one **Cylinder** for the clad, and a **Cuboid** to bound the unit. The cuboid pitch and clad radius are the same as in the 4.00% enriched fuel pin cell (`0.6` cm for the clad and `1.4` cm for the cell pitch). Use the radii that were calculated in the **Cell Data** section for the other cylinders:

$$r_{fuel,1} = 0.2236,\ r_{fuel,2} = 0.3162,\ r_{fuel,3} = 0.3873,\ r_{fuel,4} = 0.4472,\ r_{fuel,5} = 0.5000\ .$$

A figure of this unit cell can be found in Fig. 4.11.

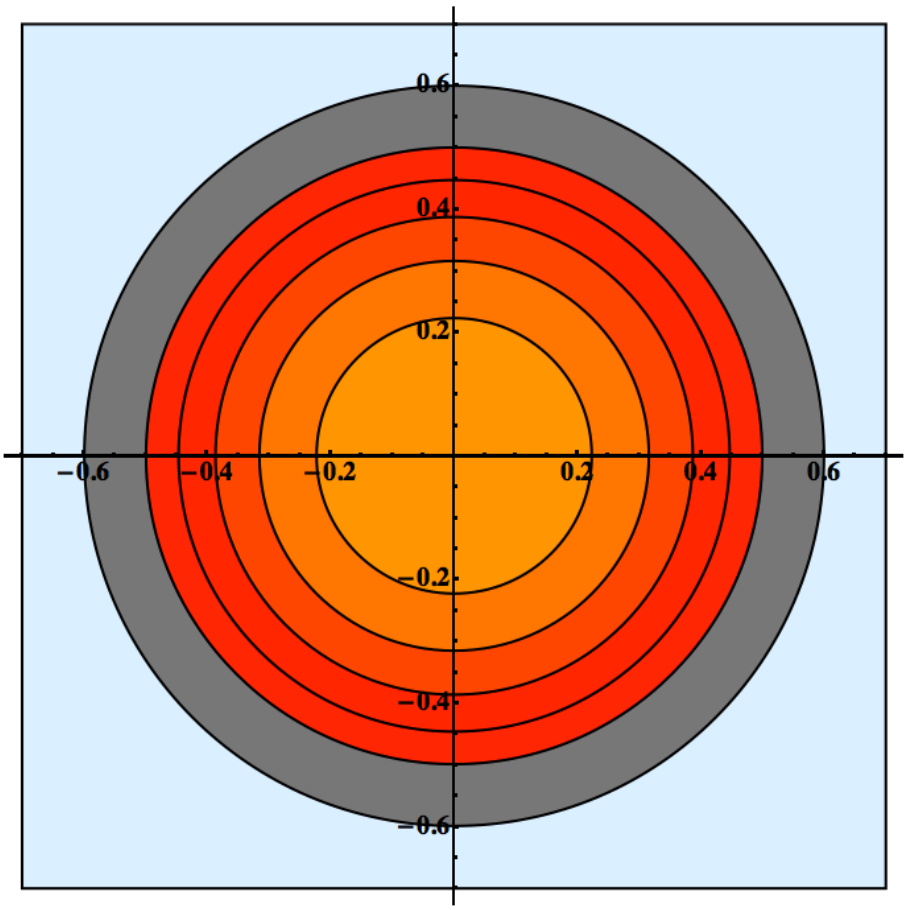


Fig. 4.11. 3.50% enriched UO_2 + 6 wt% Gd_2O_3 fuel cell.

Click **New** to begin entering information for this new unit. Provide a descriptive comment in the **Comment** box (`fuel, 3.5% enr + 6 wt% Gd`) and enter `2` in the **Select Unit** box. Then add all the cylinders (6) and the cuboid needed for this unit cell by using the geometry shape buttons **Cyl** and **Cuboid** buttons along the right size of the **Geometry Form**. Again, it is a good idea to skip numbers when labeling shapes so that it is easy to add more shapes later, if needed. After adding all the shapes for this unit, the **Geometry Form** should look like Fig. 4.12.

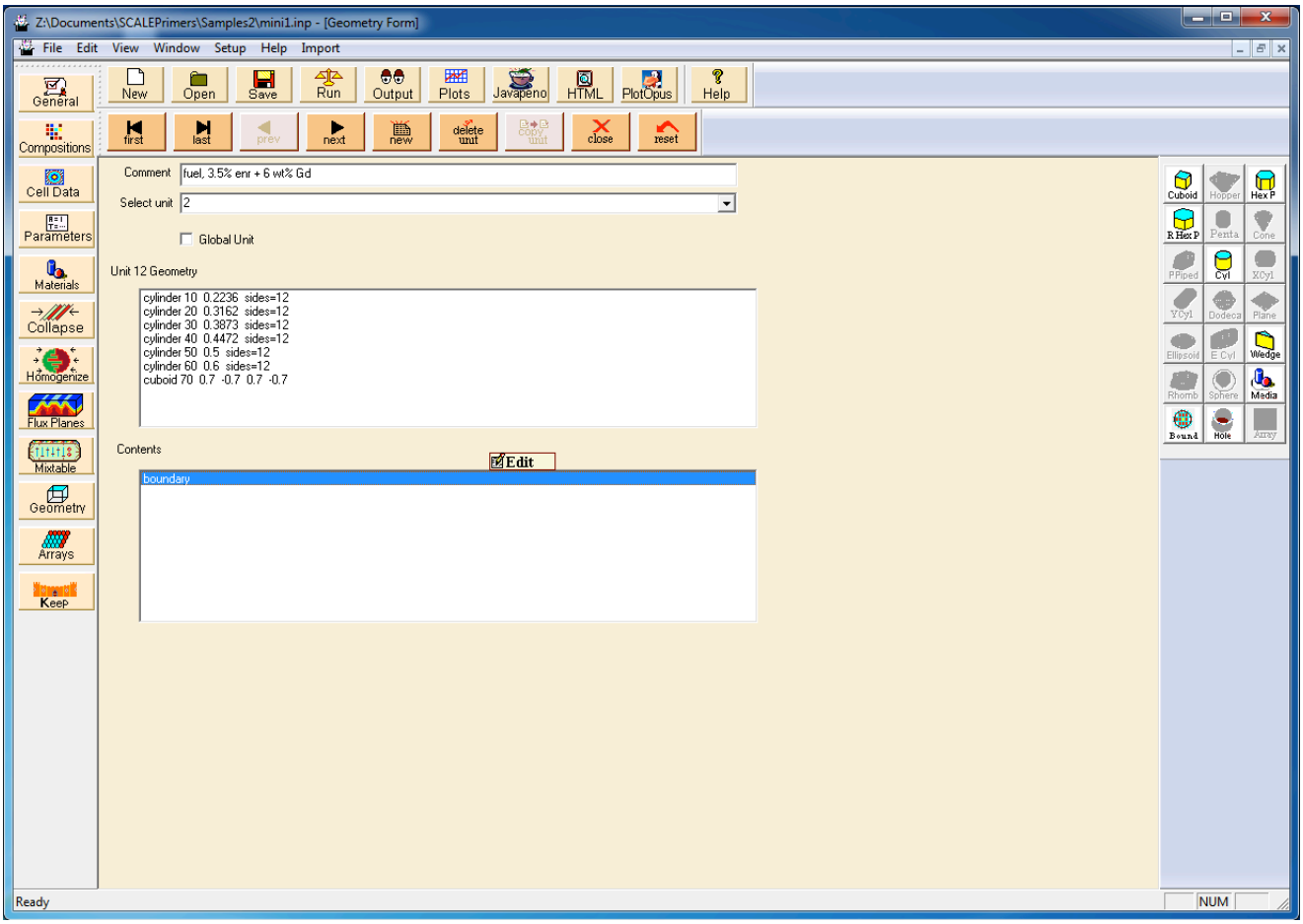


Fig. 4.12. Geometry Form for the 3.50% enriched UO_2 + 6 wt% Gd_2O_3 fuel cell.

Now add the required media statements for this unit. In the composition section, you added five different identical gadolinium-bearing fuel mixtures. In this section, you will use those mixtures. Click the **Media** button and specify the media in the **Media for Unit** window, and then click **OK** to close that window for each media statement—a total of seven times. Start with composition number `2` (`3.5% enr + 6 wt% Gd`) and specify media statements for compositions `12`, `22`, `32`, and `42` with increasing ring size. You should also specify media statements for the clad (`3 zirc4`) and moderator (`5 h2o,boron`). So the media statements should be:

`media 2` is inside `cylinder 10`;
`media 12` is outside `cylinder 10` and inside `cylinder 20`;
`media 22` is outside `cylinder 20` and inside `cylinder 30`;
`media 32` is outside `cylinder 30` and inside `cylinder 40`;
`media 42` is outside `cylinder 40` and inside `cylinder 50`;
`media 3` (zirc4 clad) is outside `cylinder 50` and inside `cylinder 60`;
`media 5` (h2o + boron moderator) is outside `cylinder 60` and inside `cuboid 70`.

Now specify the boundary of the unit using the **Bound** button. The boundary should be `cuboid 70` with a `4×4` grid. The final geometry form for unit 2, the multi-ringed gadolinium-bearing fuel pin, can be found in Fig. 4.13.

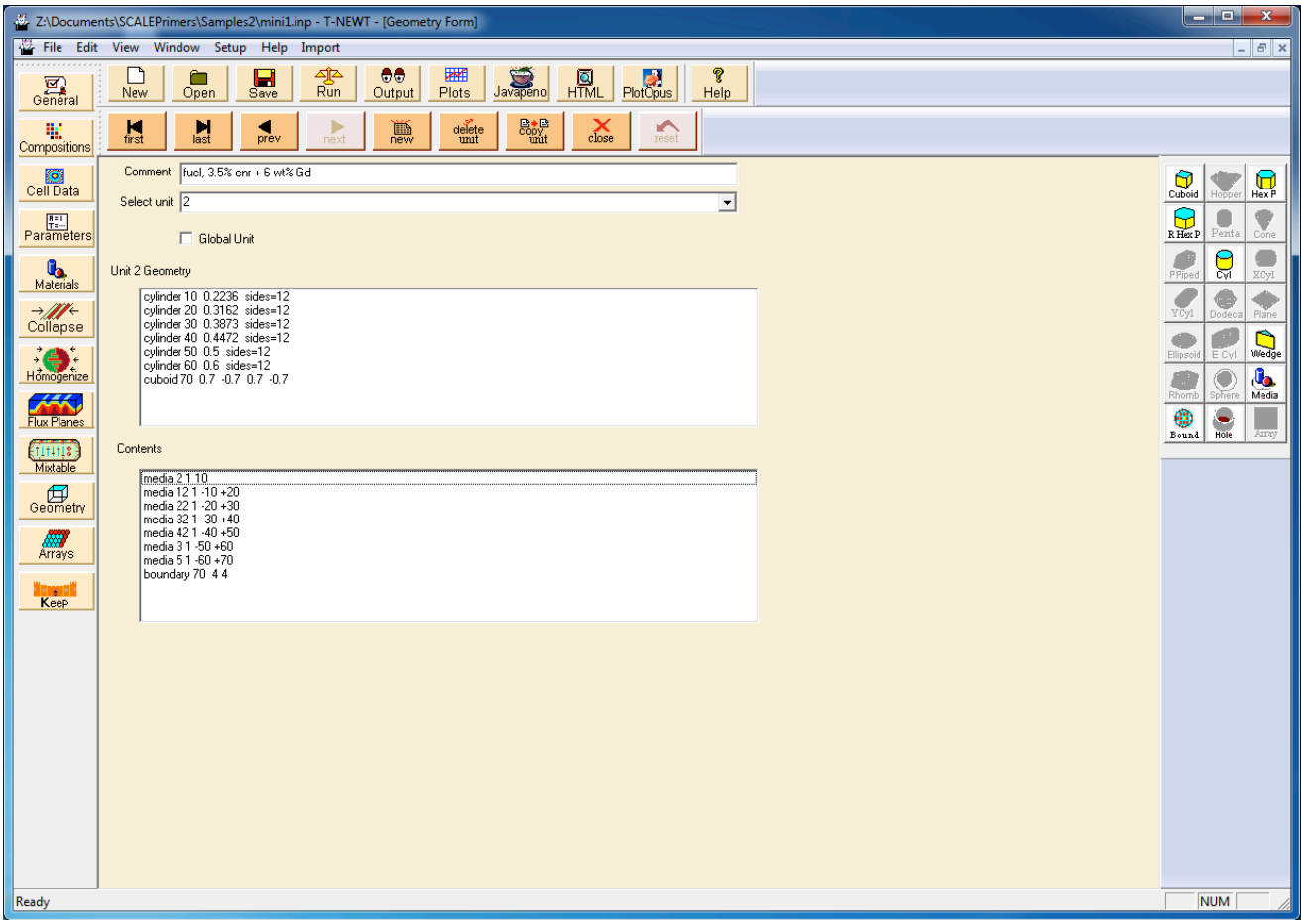


Fig. 4.13. Final Geometry Form for the 3.50% enriched UO_2 + 6 wt% Gd_2O_3 fuel cell.

You should now have three different units in the mini-assembly model: a water-filled cell (unit 10), a 4.00% enriched fuel pin (unit 1), and a five-ring 3.5% enr + 6 wt% Gd_2O_3 fuel pin (unit 2). Later you will use an array to place these units in a 2×2 mini-assembly. Before you learn how to use arrays, you should set up the global unit into which the array will be placed.

4.5.4 Global Water Cell

This unit will serve as the global unit to the mini-assembly model. It will be a simple cuboid that is exactly the size of four unit cells arranged in a 2×2 array. It is good practice to fill this unit with the material that fills the array (borated water). Filling this unit with borated water ensures that the "background" material behind the array is water, so that if the global unit is not entirely covered by arrays or holes, the leftover space is water and not some other material. You may define the center of this cuboid anywhere you like. Some users prefer to place the center of the model at (0,0), but others may choose to place the lower left corner of the model at (0,0). This example uses a cuboid centered at (0,0). Create another unit with a descriptive comment, such as `mini-assembly global unit`. It can be helpful to give the global unit an easy-to-remember unit number, such as `100`. Now add a cuboid that is 2 pitches long by 2 pitches wide (2.8 cm × 2.8 cm). With the center of the cuboid at (0,0), the cuboid should span from -1.4 to 1.4 on both the *x* and *y*-axes. Give the cuboid a label of `10`. Much like the empty water cell, specify borated water as the only media in this unit. You will also need to specify that cuboid 10 is the boundary of the unit using the **Bounds** button. In this case, specify a `2×2` grid for the

global cuboid—there is a reason for this that will discussed later. Check the **Global Unit** checkbox and the **Boundary Conditions** button will appear to the right. Click this button and specify **Mirror** boundary conditions for all surfaces and click **OK**, as seen in Fig. 4.14.

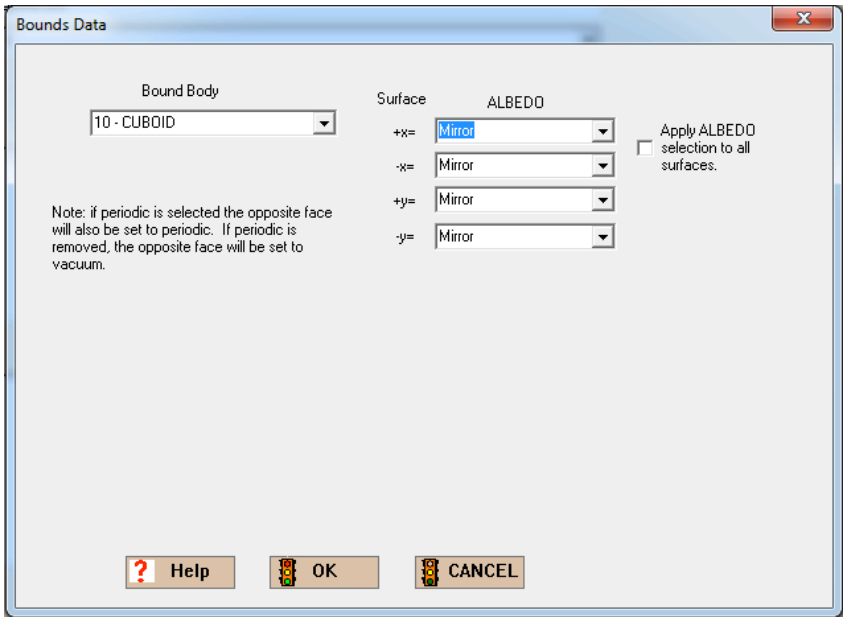


Fig. 4.14. Bounds data form.

*Hint: You may have noticed that GeeWiz will often try to renumber units for you. It is suggested that you specify your own unit numbers. It can be helpful to use the unit numbers in a logical fashion for larger, complex lattices. If GeeWiz changes the unit number that you input, change it back. It will probably work best if you wait until the last thing to restore your unit numbers; i.e., input all the data for the unit, and change the unit number before clicking **OK** on the final **Geometry Form**.*

The final **Geometry Form** for this unit should now look like Fig. 4.15. If so, you are ready to move on to adding the array to this model.

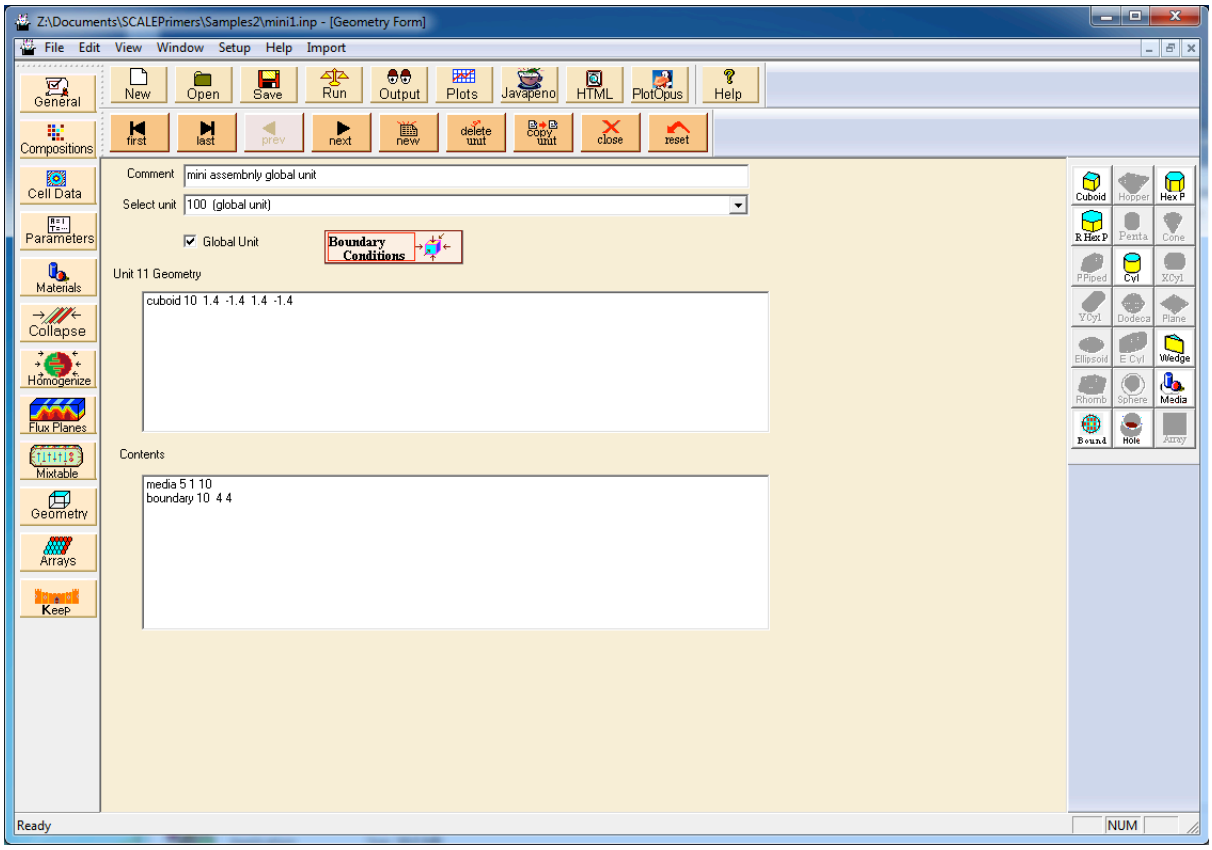


Fig. 4.15. Final Geometry Form for global unit.

4.6 ARRAYS

Arrays are constructed by stacking units. Each unit in an array has its own coordinate system; however, all coordinate systems in all units must have the same orientation. All geometric data used in a problem are correlated to the absolute coordinate system by specifying a global unit.

Arrays are created by stacking units that have a cuboidal or hexagonal outer boundary. The dimensions of adjacent faces of adjacent units stacked in this manner must match exactly. See Fig. 4.16 for a typical example.

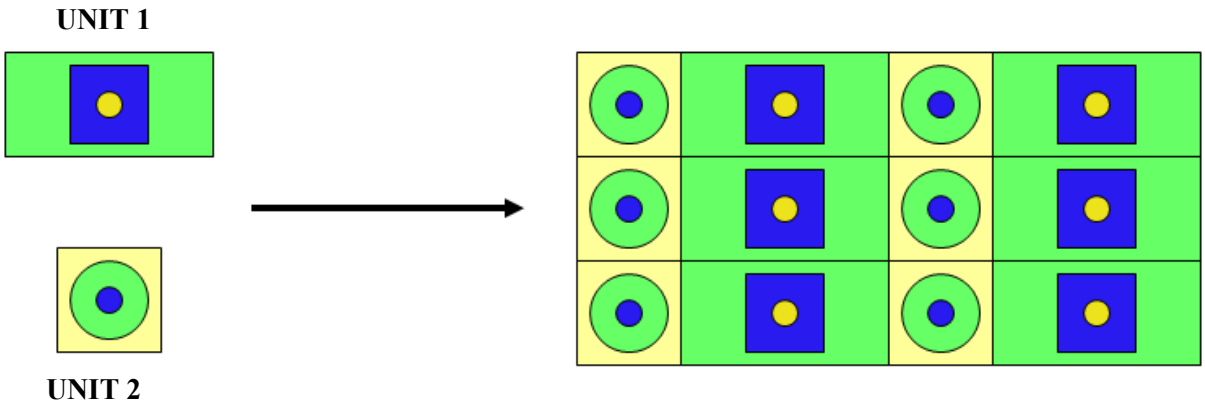


Fig. 4.16. Example of array construction.

The ARRAY option is provided to allow placing an array within a unit. Only one array can be placed directly in a unit. However, multiple arrays can be placed within a unit by using HOLEs as described in Sect. 5. Arrays of dissimilar arrays can be created by nesting units that contain arrays inside a larger array (i.e., "array of arrays").

TRITON/NEWT offers four different 2-D array types as shown in Fig. 4.17:

- cuboidal/square (or rectangular)
- hexagonal (or triangular)
- standard hexagonal
- rotated hexagonal

This primer focuses on simple cuboidal arrays—square arrays containing a single unit type or arrays with multiple units of the same size.

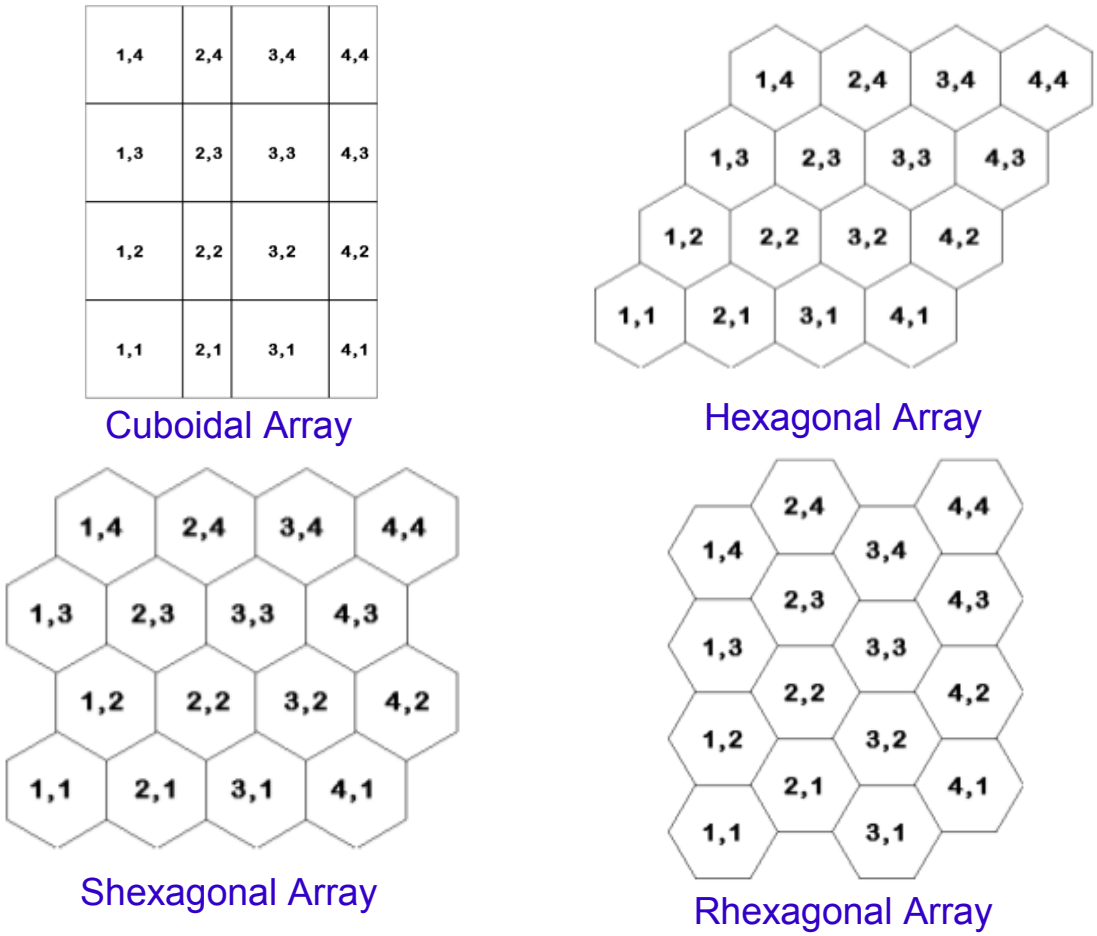


Fig. 4.17. KENO-VI array types.

4.6.1 2×2 Array with a Single Unit (4.00% Enriched Fuel) – LWR-Specific Content

This problem consists of a 2×2 array of the same type of 4.00% enriched fuel (unit 1 that was created earlier). A schematic of the array can be found in Fig. 4.18.

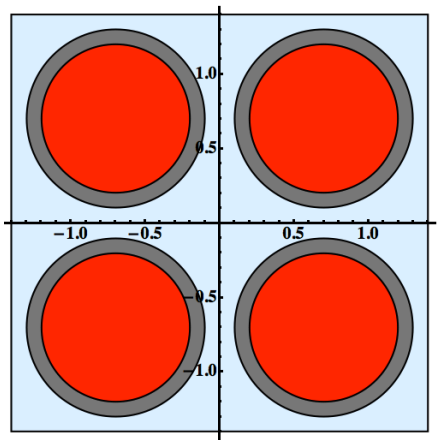


Fig. 4.18. A 2×2 array.

Although this GeeWiz input has three separate units, there is no requirement to use every unit in the array. You can use as many or as few of the units specified in the input file as desired to construct the array. Units are only added to the global geometry through the use of arrays or holes, so you can have some units that are not used in the global geometry. Other regions can be added to the global unit by defining shapes and media for those regions, but it is recommended that you minimize the complexity of the global unit to make diagnosing geometry errors easier. To build the 2×2 array for this problem, click the **Arrays** button along the left vertical toolbar. The standard **Array Form** window will appear, as seen in Fig. 4.19.

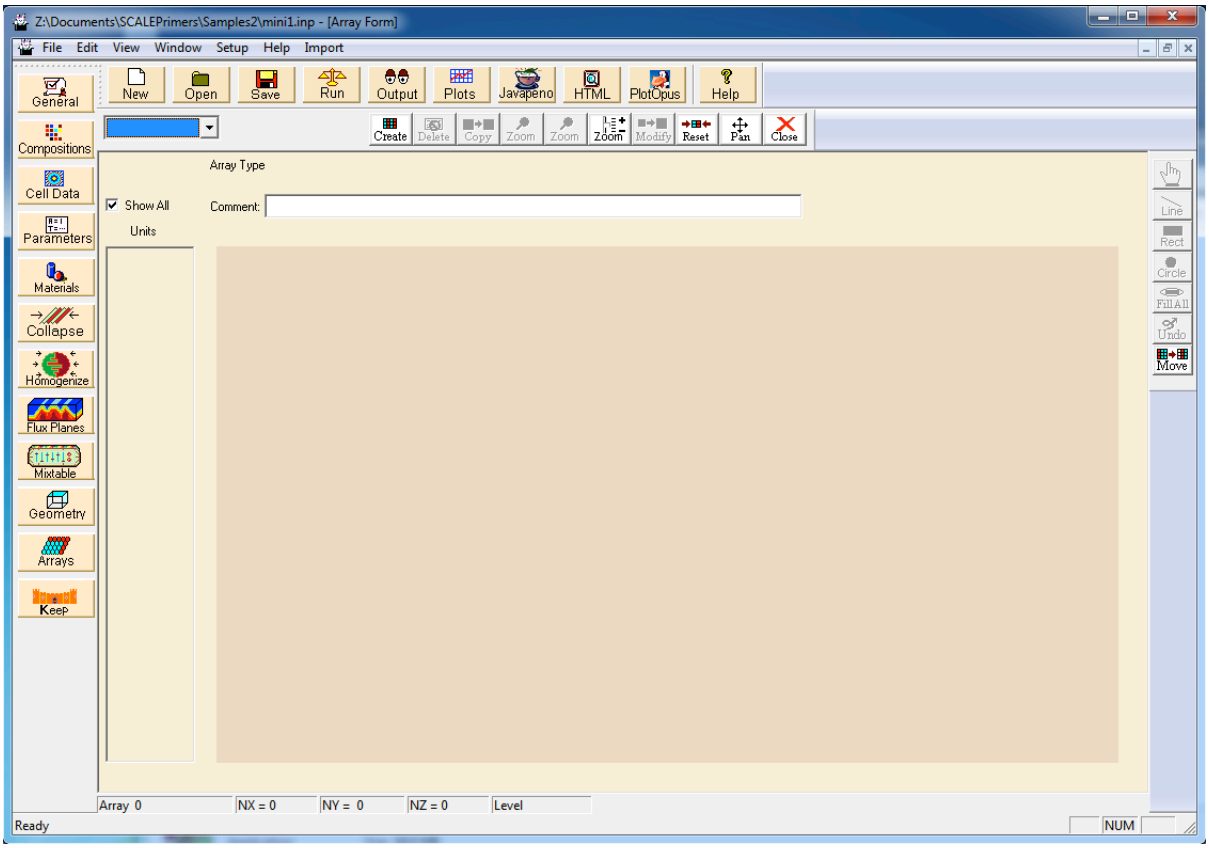


Fig. 4.19. Array form.

In the **Array Form**, you will notice two different toolbars. One toolbar is horizontal along the top of the form, and the other is vertical along the right side of the form. First click the **Create** button on the top horizontal toolbar. The **Array Properties** window will appear, as seen in Fig. 4.19. Now specify an array **Number** that will be used as a label: `1` is a good choice. Arrays are numbered independently of units, so the array number can be the same as a unit number. There should be `2` units in the *x* direction (**NUX**=`2`) and `2` units in the *y* direction (**NUY**=`2`). Also, select `unit 1` from the dropdown to initialize the entire array with `unit 1`. The **Array Properties** window should now look like Fig. 4.20. If so, click **OK**.

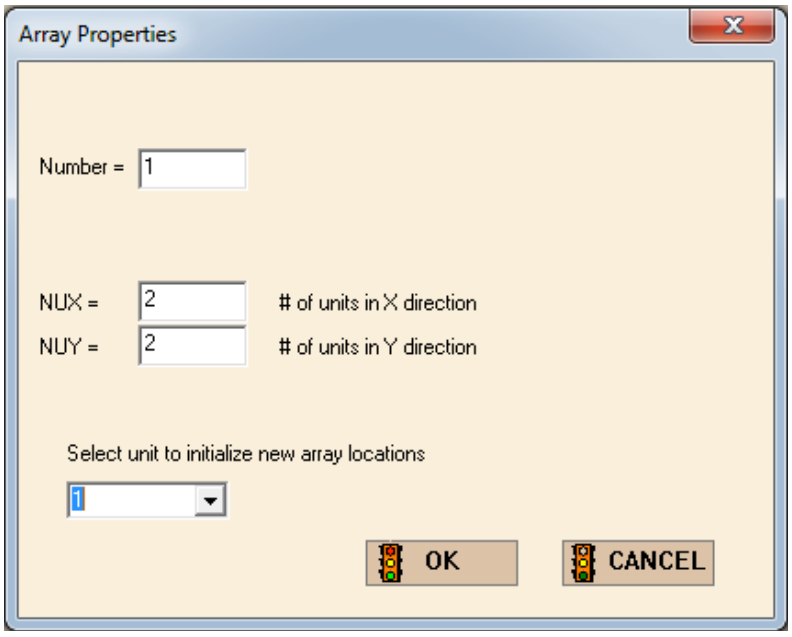


Fig. 4.20. Array properties window.

After clicking **OK**, the **Array Form** will change: a 2×2 array of boxes represents the array. Along the left side of the **Array Form**, you will see that three units are available under the **Units** box: `10`, `1`, and `2`. These units are available to be placed in the array. The new **Array Form** can be seen in Fig. 4.21.

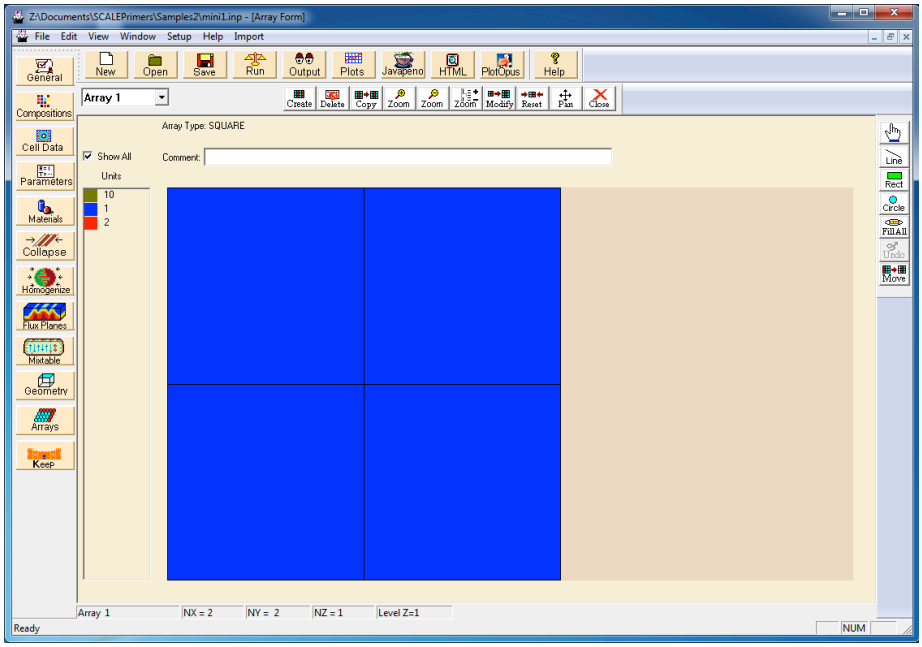


Fig. 4.21. Array form showing a 2×2 array.

Now give the array a **Comment** such as "`global fuel array`". Click **Close** to return to the **Geometry Form**. In the right toolbox of the **Geometry Form**, you will notice a new option, **Array**. On the main **Geometry Form,** select `100 (global unit)` from the **Select Unit** dropdown and then click the **Array** button. The **Array** dialog window will appear, as seen in Fig. 4.22.

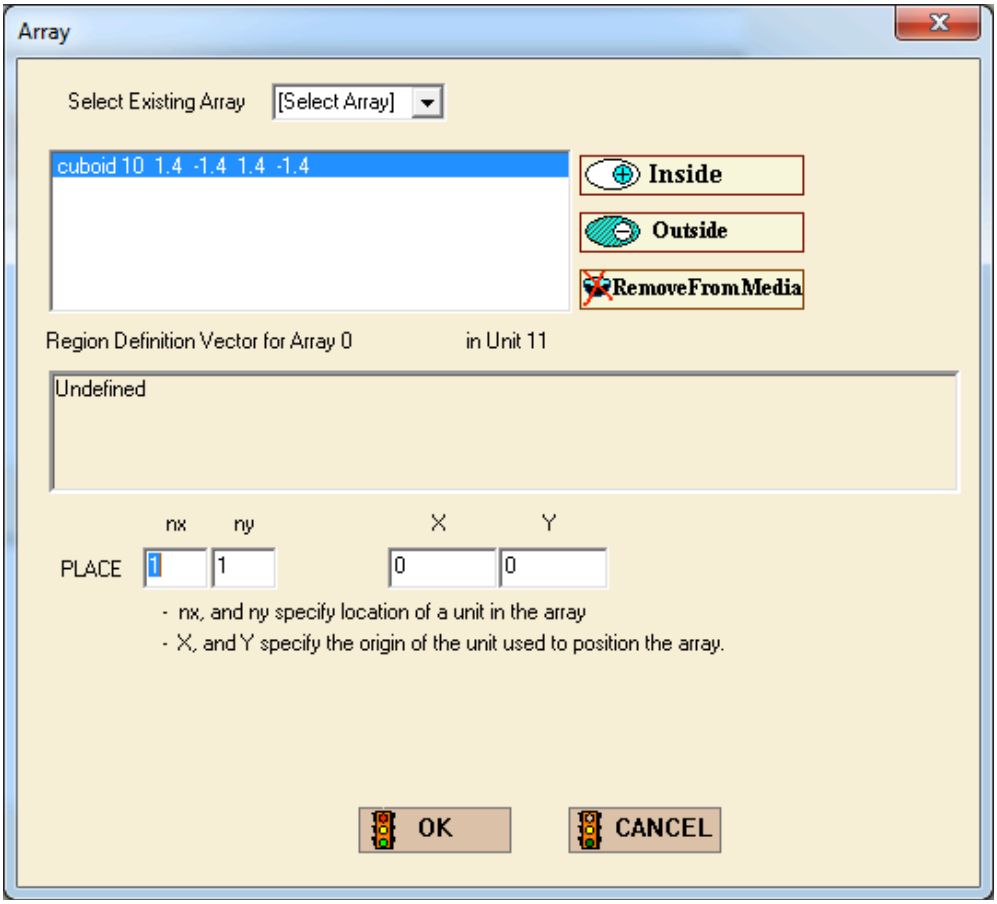


Fig. 4.22. Array dialog used to place array into units.

When placing arrays, remember that arrays are defined in TRITON/NEWT with the (1,1) element as the bottom left (southwest) index of the array (refer to Fig. 4.17).

Select `array 1` from the **Select Existing Array** dropdown, and specify that `array 1` lies inside `cuboid 10`. Now specify *x* and *y* coordinates in the global unit where the center of array index (*nx*,*ny*) will be placed. In the **PLACE** line, input array indices (*nx*,*ny*) = `(1,1)` at (*x*,*y*) = `(-0.7,-0.7)`. This places the center of array element (1,1) at (-0.7,-0.7) in the unit 100 coordinate system. A figure of the **Array** dialog can be found in Fig. 4.23. Click **OK** when finished entering information into the **Array** dialog.

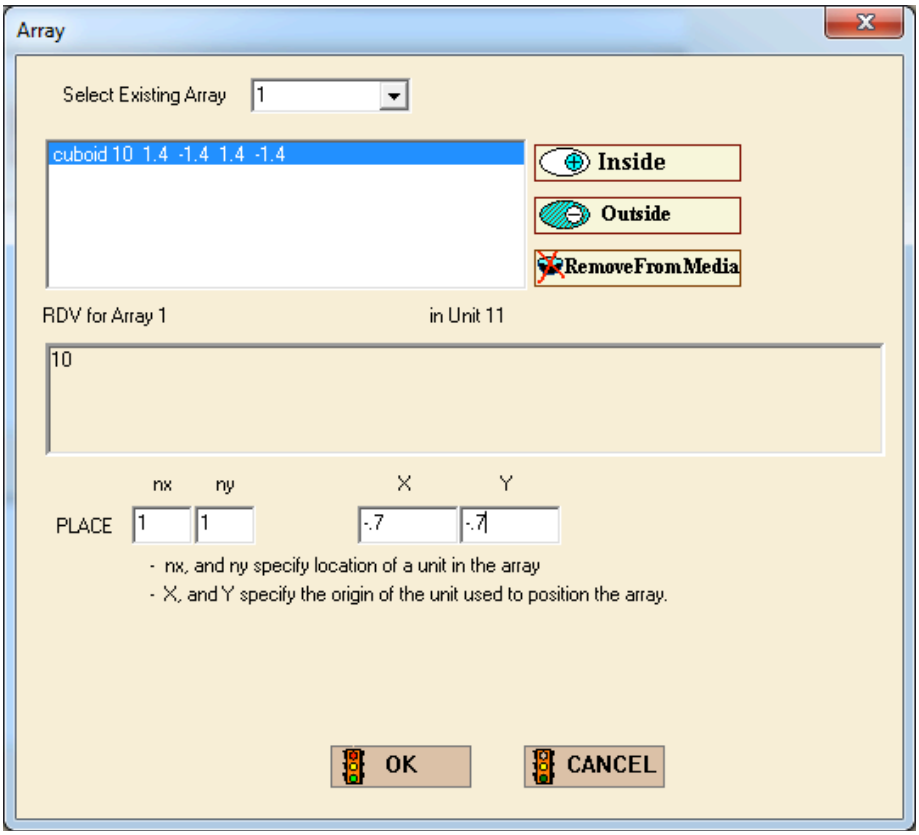


Fig. 4.23. Array dialog with element (1,1) placed at (-0.7,-0.7).

You now have model of a 2×2 array of 4.00% enriched fuel pins. The global unit for this model has reflecting boundaries, so it will simulate an infinite array of fuel pins. Before attempting to run the problem, add **Materials** so that the cross sections are loaded into NEWT. From the main GeeWiz window, click the **Materials** button on the left vertical toolbar. Add each mixture that is present in the compositions section of the model. Insert descriptive comments for each material. You should enter the following mixtures:

Mixture	Pn Order	Comment
1	1	fuel, 4.00% enriched
2	1	fuel, 3.5% enr + 6 wt% Gd2O3
12	1	fuel, 3.5% enr + 6 wt% Gd2O3
22	1	fuel, 3.5% enr + 6 wt% Gd2O3
32	1	fuel, 3.5% enr + 6 wt% Gd2O3
42	1	fuel, 3.5% enr + 6 wt% Gd2O3
3	1	clad, zirc4
4	1	structure, ss304l
5	2	mod, h2o + boron
6	1	absorber, b4c

It is recommended that you use second-order scattering in water due to the higher level of anisotropic scattering in lighter elements such as hydrogen. The final **Materials Mixtures** window should look like Fig. 4.24.

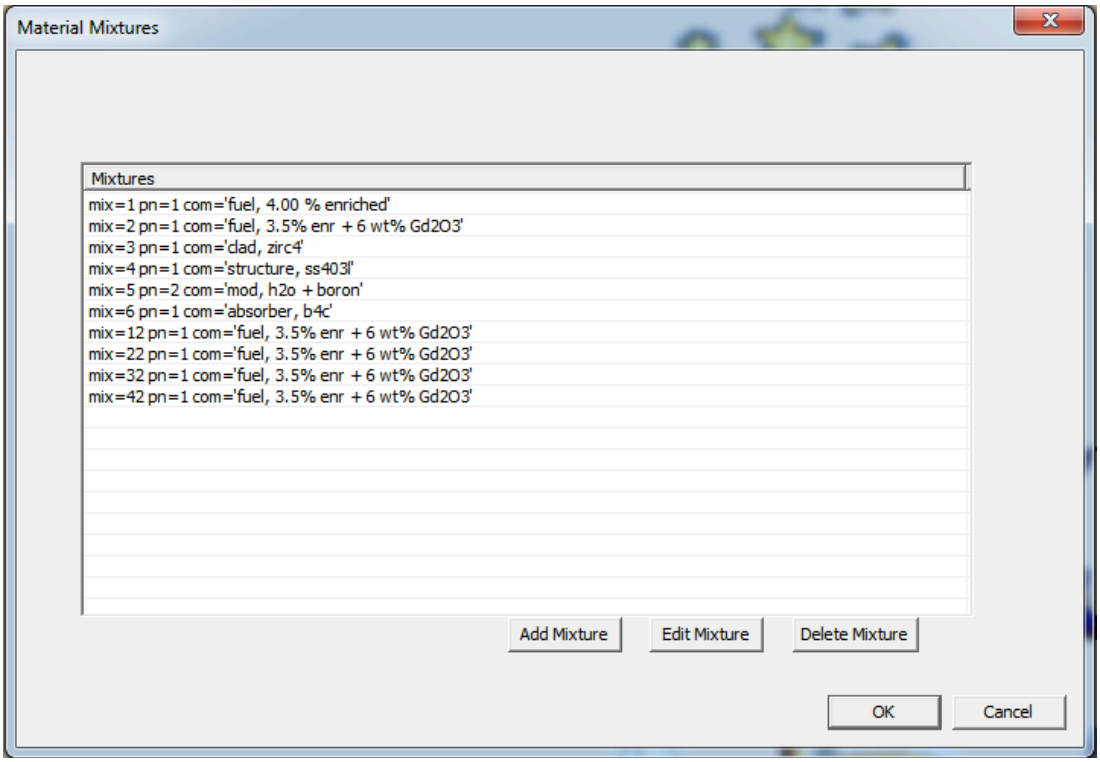


Fig. 4.24. Material mixtures window for the mini-assembly problem.

The geometry setup is complete. A few options and parameters need to be specified before submitting the job. Make sure that in the **General** settings that **Parm=check** is selected to check and plot the geometry. You can allow messages to be printed in the DOS window during execution by selecting `-m` option under **Batch6 Arguments** in the **General** window, as seen in Fig. 4.25.

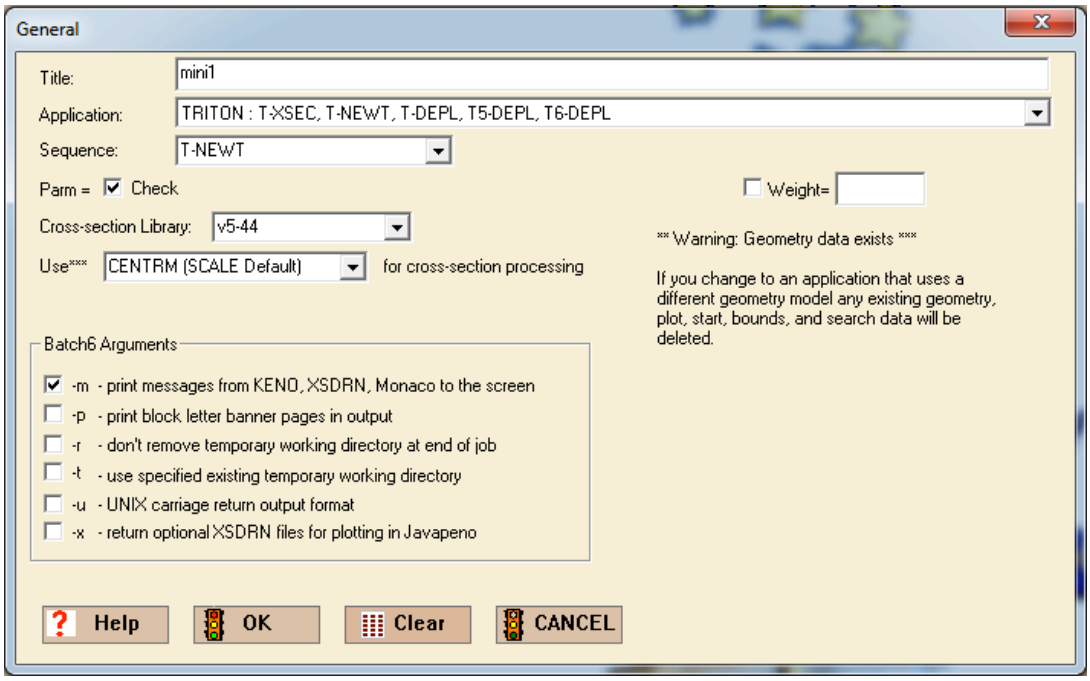


Fig. 4.25. General parameters for the mini-assembly problem.

Click on the **Parameters** button on the left toolbar to make sure that the **DRAWIT** option is checked, along with the **ECHO** option, as seen in Fig. 4.26. The **DRAWIT** option enables TRITON/NEWT to generate PostScript plots (*.ps files) of the geometry. The **ECHO** option will display NEWT outer iterations to the command prompt. Once complete, you will want to save the input file using the **Save** button. Name this problem `mini1.inp`.

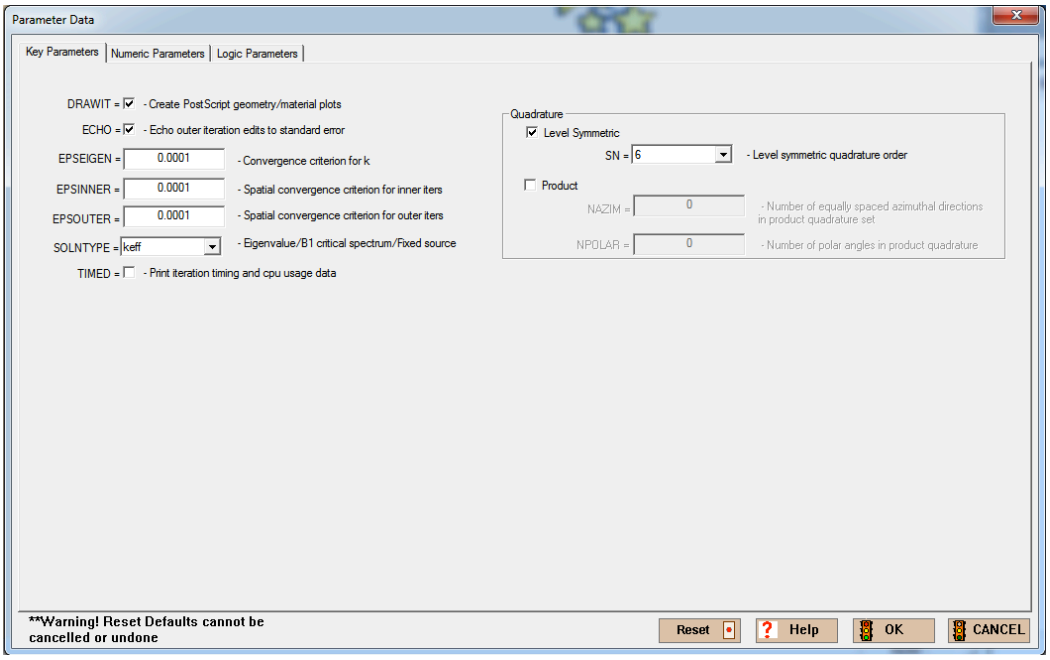


Fig. 4.26. Parameter data for the mini-assembly problem.

Now click **Run**. The command prompt will open and you will see SCALE call TRITON with some basic job information as seen in Fig. 4.27.

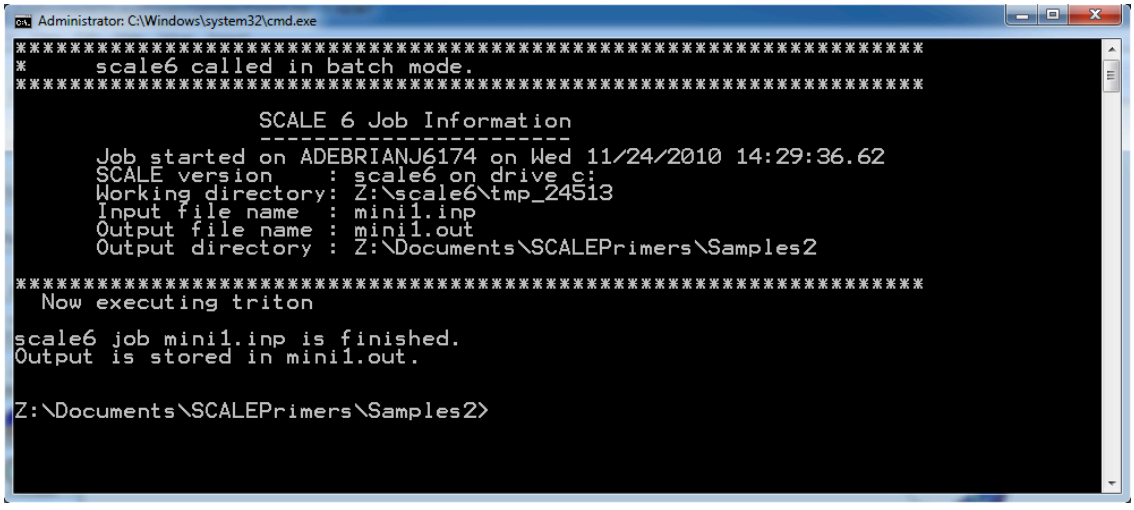


Fig. 4.27. DOS window for the mini-assembly problem with parm=check.

Now use Windows Explorer to go to the directory where you saved the input file. You should have `mini1.out`, `mini1.newtmatl.ps`, and `mini1.newtgrid.ps` in addition to the input file, `mini1.inp`. Open `mini1.newtmatl.ps` with Ghostview, Adobe Acrobat, or other PostScript conversion software. Upon opening the PostScript file, you should see the 2×2 mini lattice, as seen in Fig. 4.28.

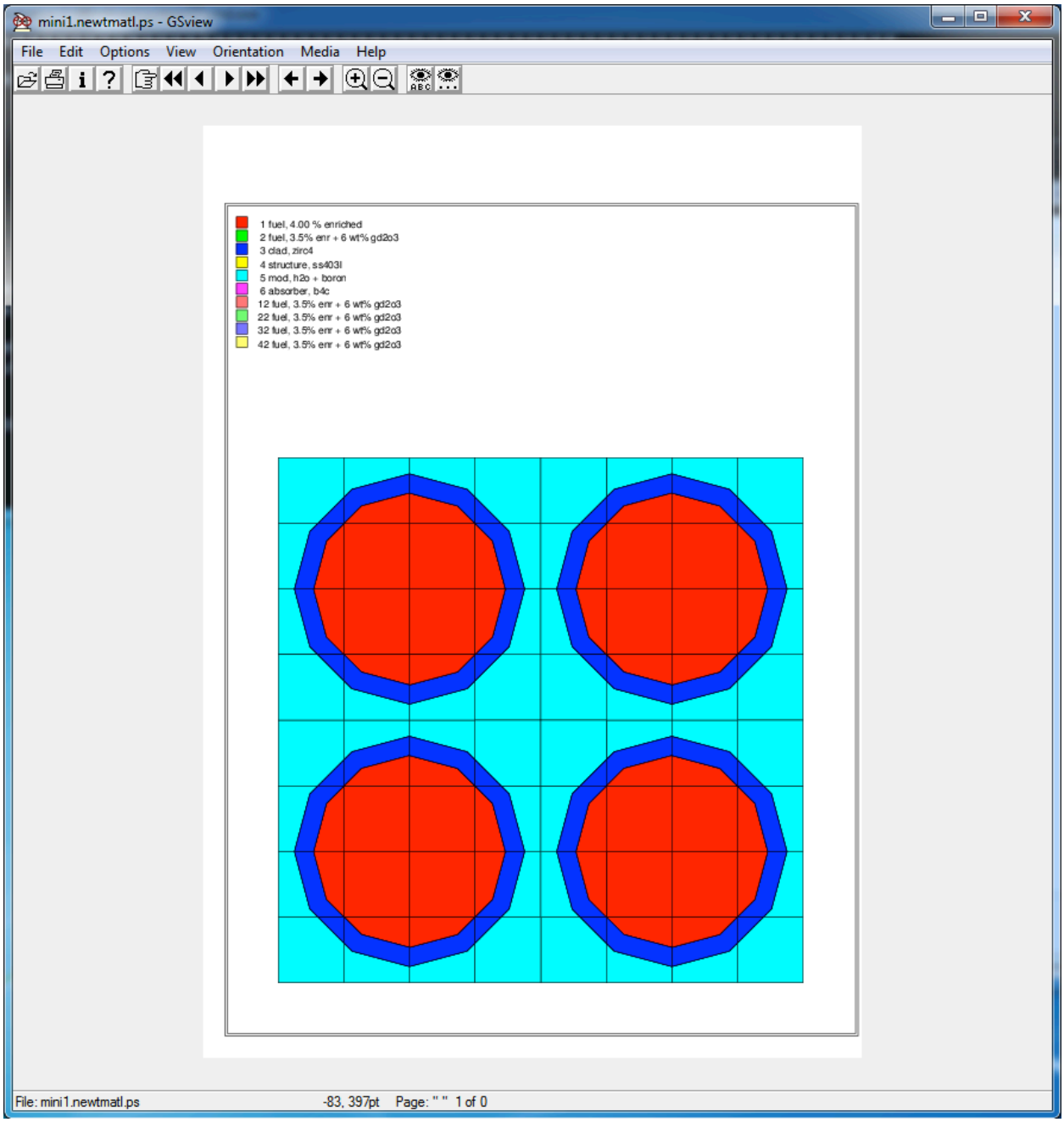


Fig. 4.28. TRITON/NEWT plot of the mini-assembly geometry.

Open the output file using GeeWiz's **Output** button, and scroll to the bottom of the output file. There should be a message that the problem was not run because parm=check was used in the analytical sequence (Fig. 4.29).

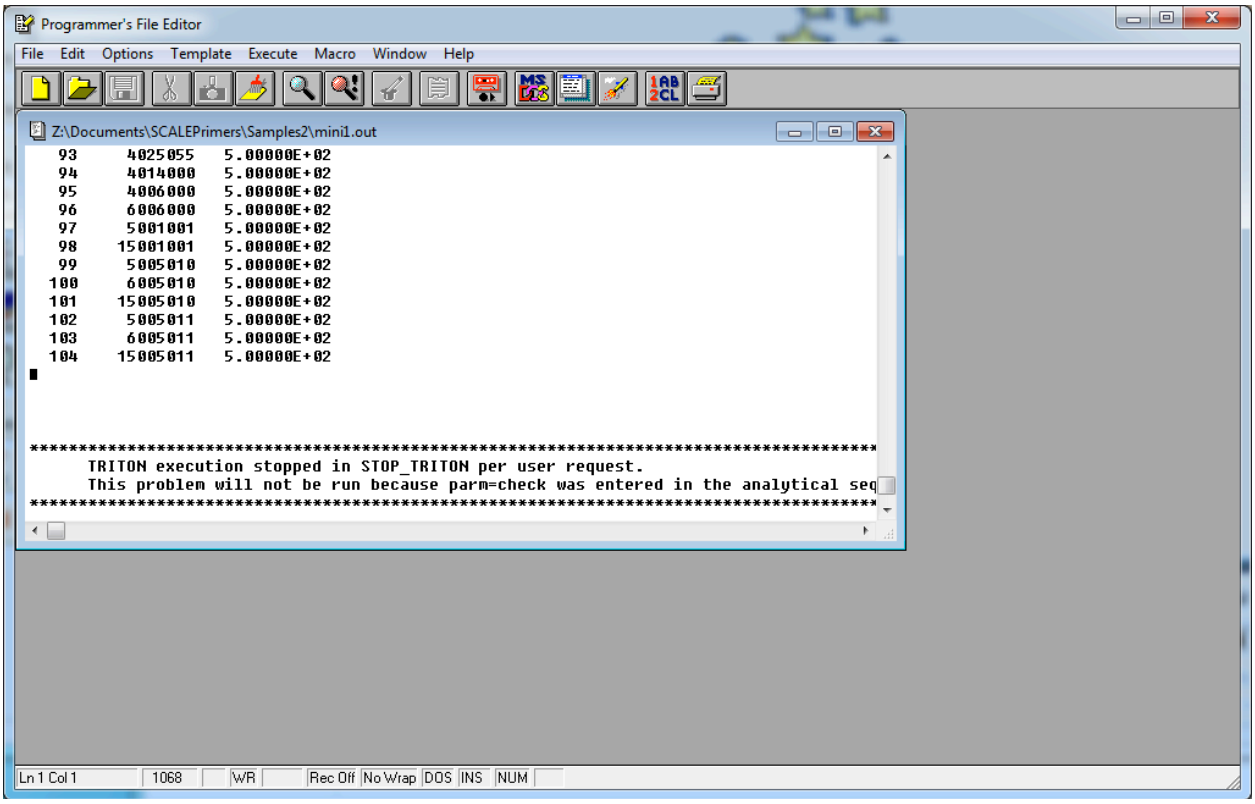


Fig. 4.29. Output file for the Parm=Check run of the mini-assembly problem.

If the output file has something else at the bottom, there might be an error in the input file. If the output file is consistent with Fig. 4.29, close the text editor output file window and close the command prompt window. Now go back to the **General** settings and uncheck the `Parm=Check` checkbox. Then click **Run**. The DOS command prompt window will open again, but this time you will see other SCALE modules running following the initial TRITON module. You will then see the outer iterations from NEWT printed to the screen.

The problem should finish and display the TRITON/NEWT estimation of the infinite multiplication factor. This information is also stored in the output file. To find this information in the output file, close the DOS command prompt window and click **Output** on the top horizontal toolbar in GeeWiz to open the output file. In the output file, you should search for two pieces of information: the total CPU time used and the final infinite multiplication factor. To do this, first search for `total cpu`. The line with `total cpu` should say `module t-newt is finished`, as seen in Fig. 4.30.

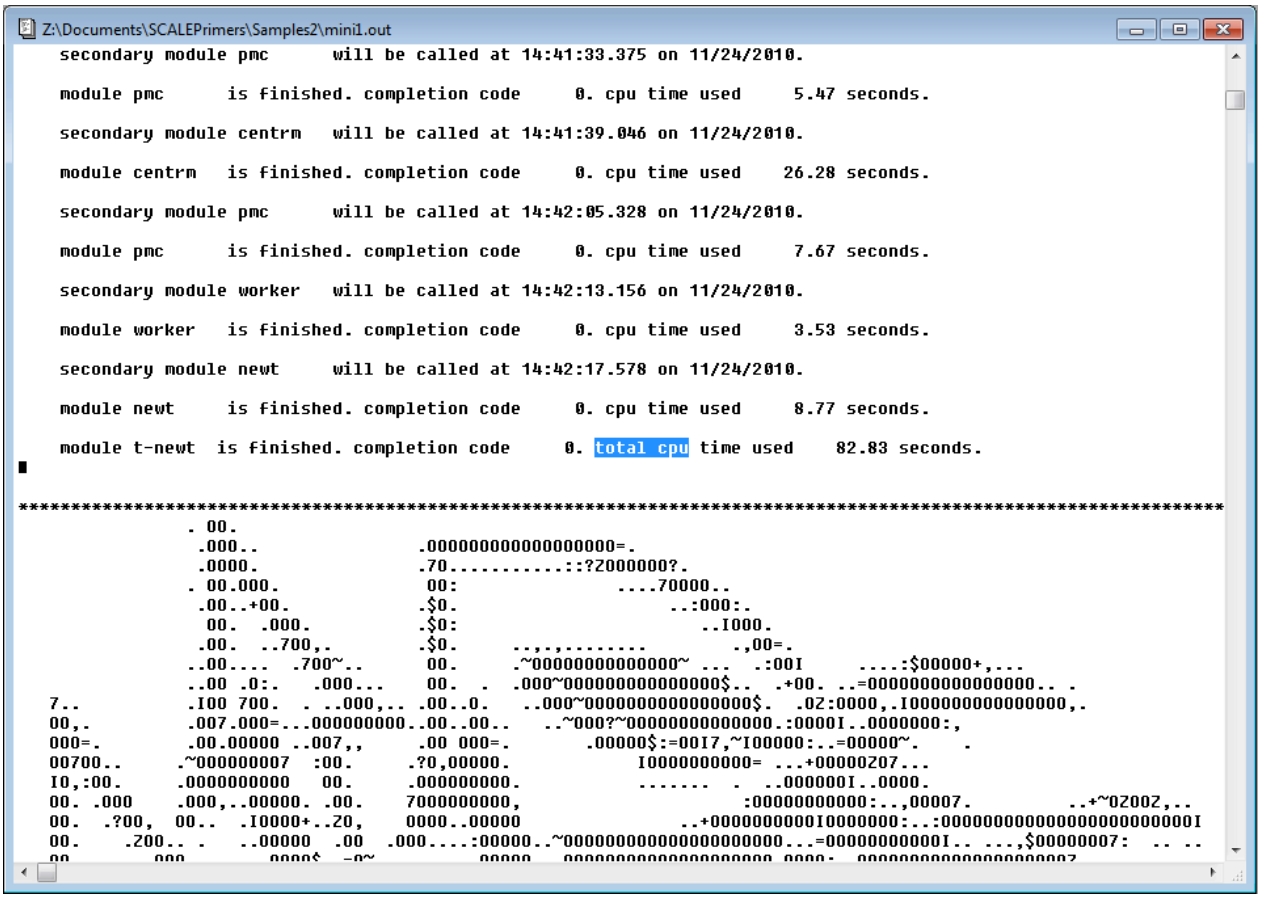


Fig. 4.30. Total CPU portion of the output file for the mini-assembly problem.

In this mini-assembly example, the total CPU time used was 82.83 seconds and the NEWT module runtime was 8.77 seconds. The runtime on your computer will vary due to the difference in computer systems. In this problem, most of the time was occupied by cross-section processing instead of the transport solution. Note that the larger the model, the longer will be the time required for the transport solution. Now search for `k-eff =` (there is a space between "`k-eff`" and "="). The calculated k_{eff} value is 1.22914946 (Fig. 4.31). In this case, k_{eff} is actually k_{inf} because the lattice has reflective boundary conditions to simulate an infinite array of pins.

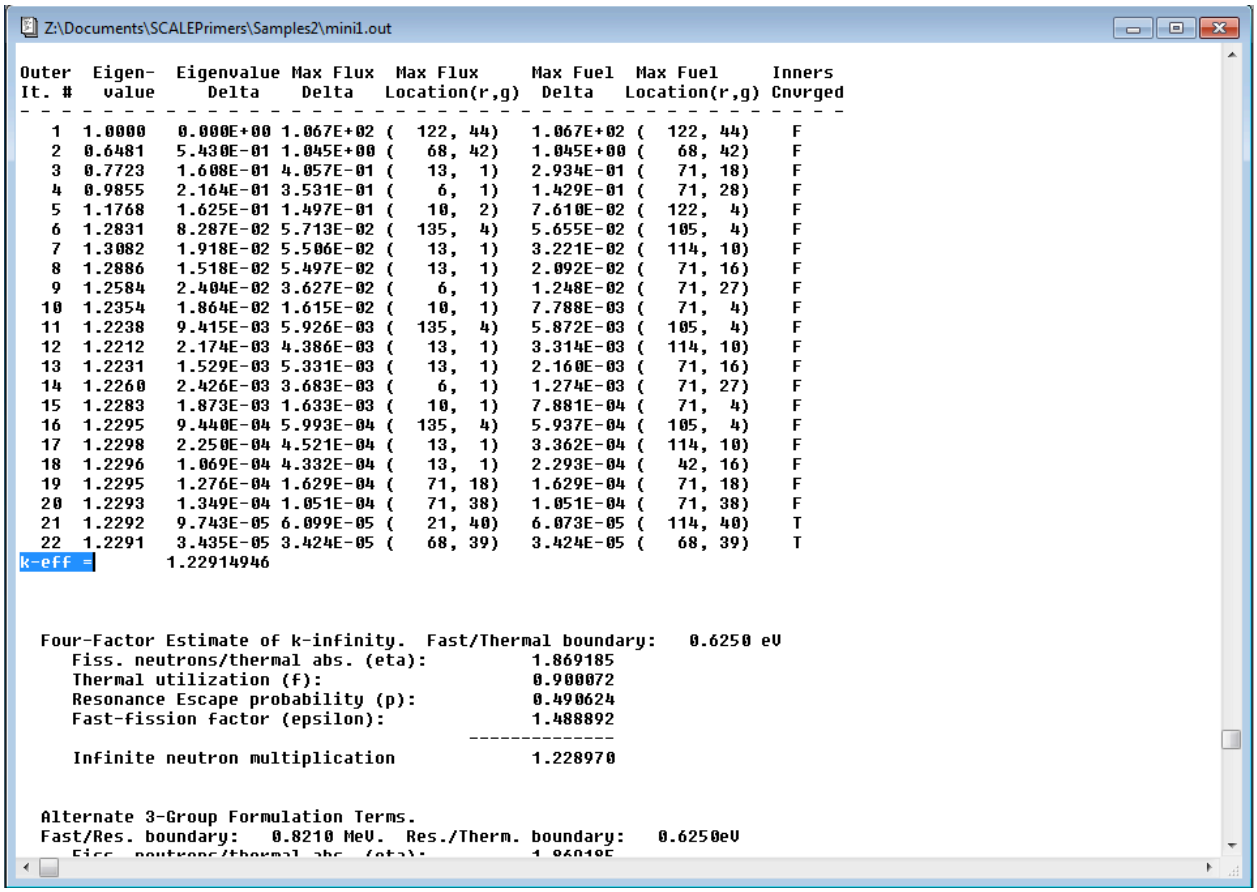


Fig. 4.31. k_{eff} portion of the output file for the mini-assembly problem.

4.6.2 2×2 Array with Single Units (4.00% Enriched Fuel) + CMFD Acceleration

This problem is identical to the previous mini-assembly problem except that you will use Coarse-Mesh Finite-Difference (CMFD) acceleration to accelerate the NEWT transport solution of the problem. The CFMD solver uses a low-fidelity solution (homogenized cells) to accelerate convergence of the high-fidelity solution. With `mini1.inp` already open in GeeWiz, go to **File**>**Save A**s and save the problem as `mini1_cmfd.inp`. Now click the **Parameters** button and the **Parameter Data** window will appear. Select the **Logic Parameters** tab in the **Parameter Data** window and check the **CMFD checkbox** (Fig. 4.32).

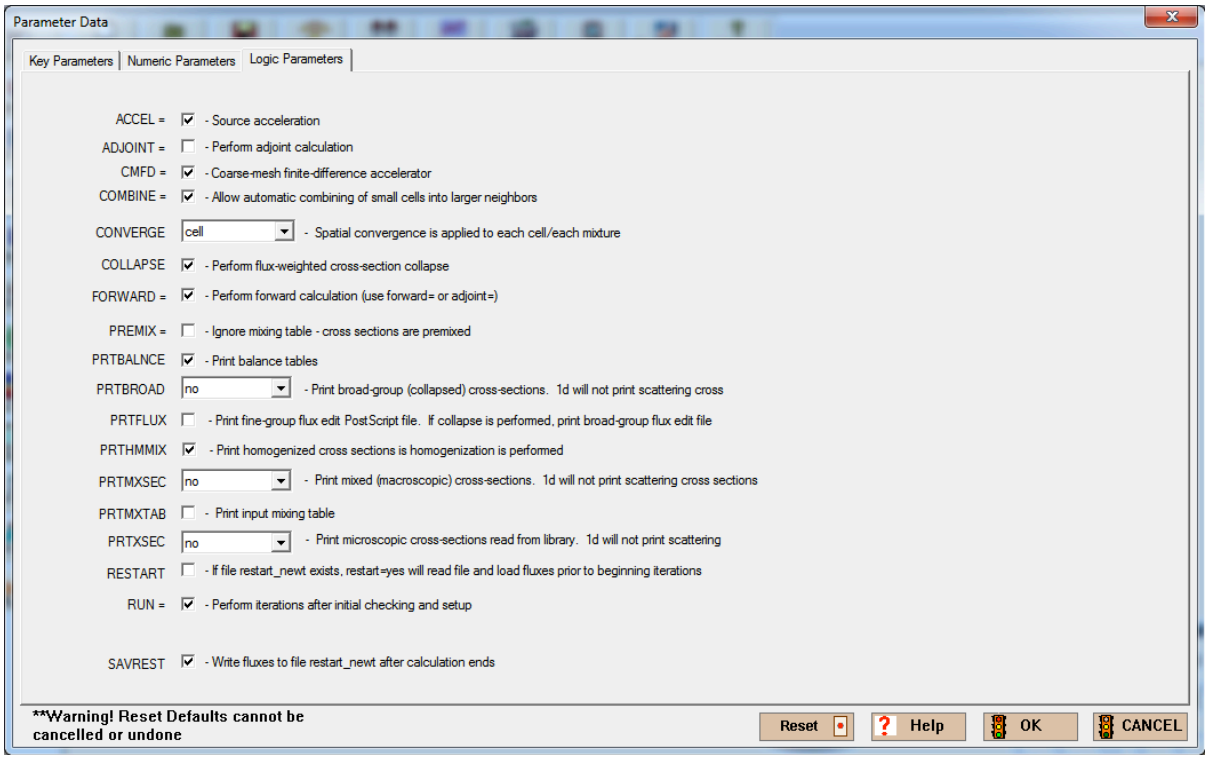


Fig. 4.32. Parameter data for the mini-assembly problem with CMFD enabled.

Click on the **Numeric Parameters** tab of the **Parameter Data** window and make sure that **XCMFD** and **YCMFD** are set to `1` (this is the default). Setting **XCMFD** and **YCMFD** to 1 constructs a CMFD grid that is one global unit grid cell in both the *x* and *y* directions. Likewise, setting **XCMFD** and **YCMFD** to 2 constructs a CMFD grid that contains two global unit grid cells in both the *x* and *y* directions. You should have set the global unit grid to 2×2, so with **XCMFD** and **YCMFD** set to `1`, the contents of each fuel pin cell will be homogenized into one block for the CMFD solution. Note that using CMFD does *not* result in a less-accurate solution. It is used only to accelerate the results of the high-fidelity solution; the converged solutions with and without CMFD should be virtually the same (within the convergence criterion). If you have completed all the steps to turn CMFD on, you can click **Run** again.

A DOS command prompt window will open again, and you will see TRITON/NEWT running. Upon completion of the run, go to GeeWiz and open the output file. Again, search for the total CPU time, the NEWT module runtime, and the infinite multiplication factor. The computer on which this sample problem was run used 69.61 seconds of total CPU time, 4.75 seconds for the NEWT module, and calculated an infinite multiplication factor of 1.22897121. The NEWT transport solution completed in approximately half the CPU time as the unaccelerated solution and gave identical results. For larger geometries that take many minutes or hours to run, it is advantageous to use CMFD to accelerate the solution.

4.6.3 Arrays with Multiple Units (2×2 array of 4.00% Enriched Fuel, 3.50% Enriched + 6 wt% Gd_2O_3 Fuel, and an Empty Lattice Location) – LWR-Specific Content

Now that you have learned how to construct a simple array in GeeWiz for TRITON/NEWT, it is time build more complicated arrays. With `mini1_cmfd.inp` open, go to **File**>**Save As** and save this problem as `mini2.inp`. You will not build any new units here—you have previously built the units

you will need for this mini-assembly problem. You will now use units 1, 2, and 10 to generate a 2×2 lattice with a water rod in the southwestern corner, UO_2 gadolinium-bearing pins in the northwestern and southeastern corners, and a 4.00% enriched UO_2 fuel pin in the northeastern corner (Fig. 4.33).

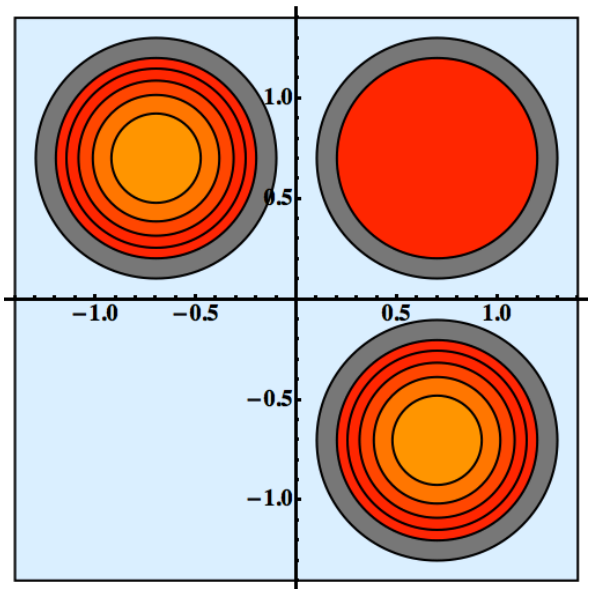


Fig. 4.33. Drawing of the mini-assembly problem with gadolinium-bearing pins and an empty lattice position.

To construct this array, click on the **Arrays** button on the left toolbar. The **Array Form** will appear and contain four blue squares (same as the last problem). Using your mouse, select the **Finger** button from the right vertical toolbar. Next click `unit 10` next to the yellow box under the **Units** box along the left side of the **Array Form**. You should then click in the southwestern (lower left) box of the four blue boxes. Upon doing so the box will turn yellow (Fig. 4.34).

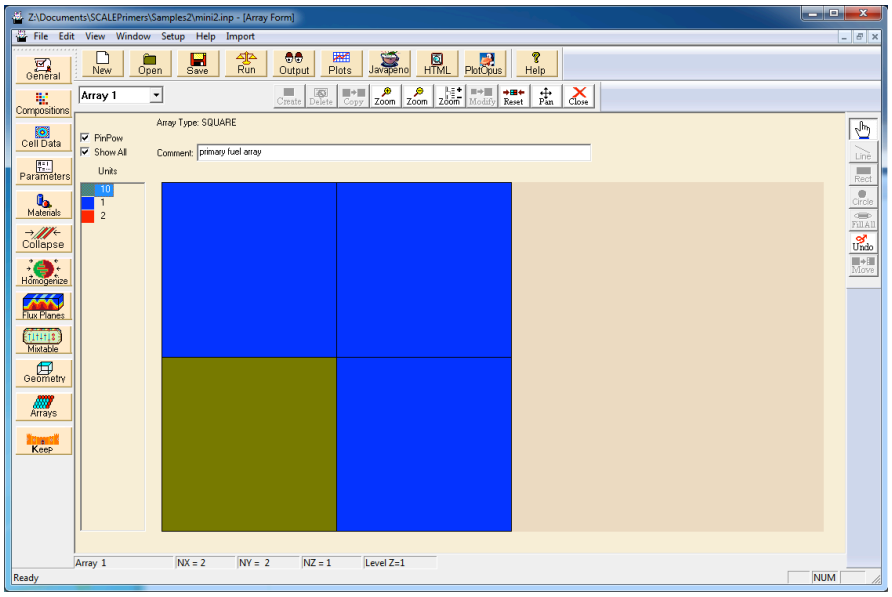


Fig. 4.34. Array form for 2×2 array with multiple units (step 1).

Then click on `unit 2` under the **Units** box of the **Array Form**, and click the southeastern and northwestern corners of the four boxes to specify that these array positions will be `unit 2` (gadolinium-bearing pins), as seen in Fig. 4.35.

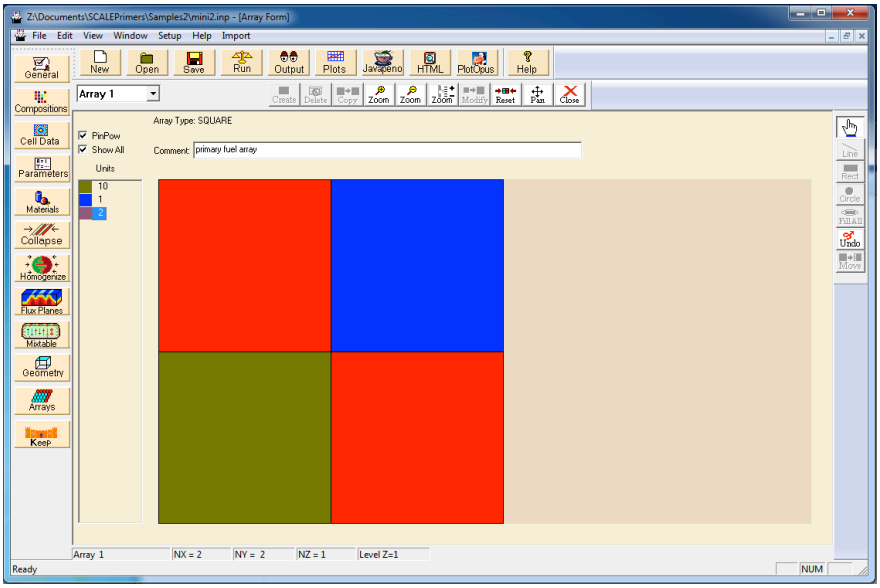


Fig. 4.35. Array form for 2×2 array with multiple units (step 2).

If the **Array Form** looks like Fig. 4.35, click **Close** on the **Array Form**. Return to the **General** settings and check the **Parm=Check** checkbox, and then save your work. Now click **Run**, and TRITON/NEWT will being running. After completion, navigate to the directory where the input file is saved and you

should have PostScript files that you can open to see the geometry. Opening `mini2.newtmatl.ps` should show you Fig. 4.36.

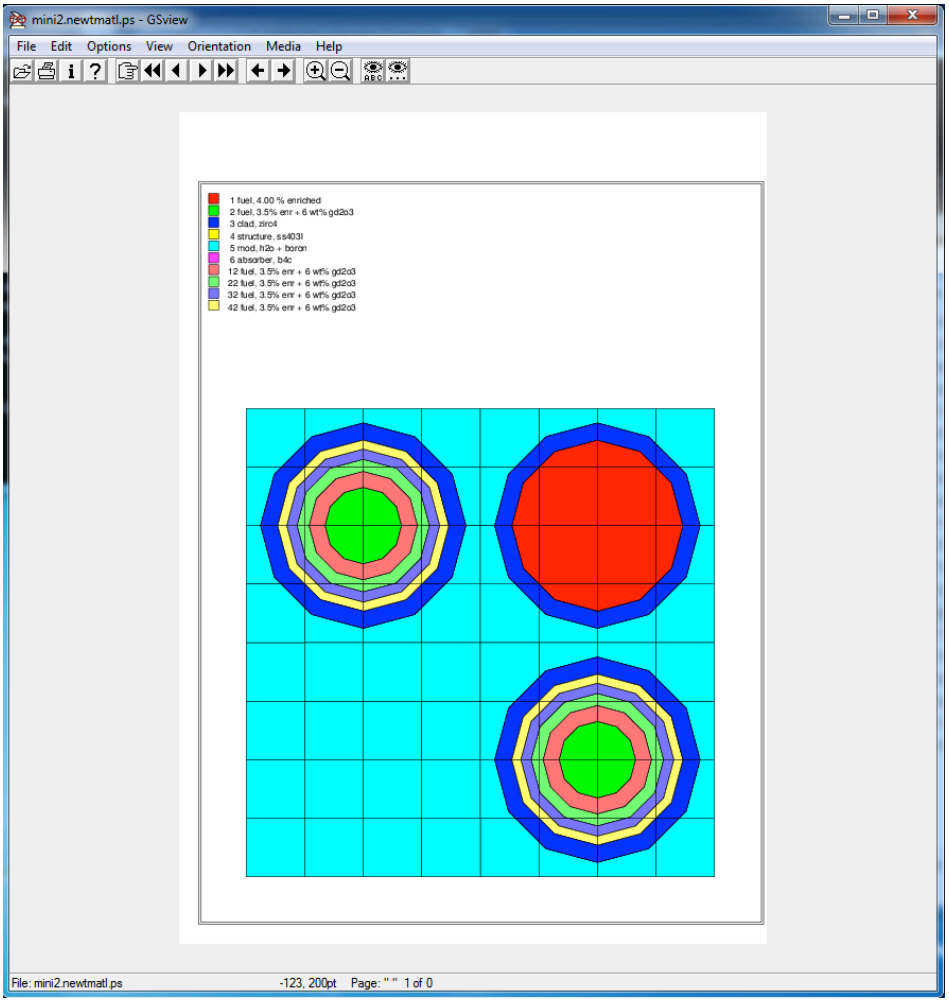


Fig. 4.36. TRITON/NEWT plot of the mini-assembly problem with gadolinium-bearing pins and an empty lattice position.

Check the output file to make sure that everything ran okay; if there are no errors, return to GeeWiz, uncheck the **Parm=Check** checkbox and rerun the problem. This problem will take longer to run because there are more regions in the five-ring gadolinium-bearing fuel pins. You will also find that the gadolinium in the fuel makes a large difference; the infinite multiplication factor will be very low. For this test problem, k_{inf} is 0.3943489. The NEWT transport solution for this case ran in 7.51 seconds, nearly twice as long as the previous problem.

4.7 SUMMARY

This section has helped you to accomplish these objectives:

- use GeeWiz to create some basic shapes (cylinder and cuboid)
- understand how units are created
- create a simple array consisting of a single unit
- use CMFD to accelerate the NEWT transport solution
- create an array with multiple units

Now that you have spent time with the material for simple geometry and the array input, more advanced geometries features will be introduced.

5. ADVANCED GEOMETRY FOR LATTICE PHYSICS

In the previous section, you set up and ran some simple geometry problems, including simple array problems. This section provides more advanced geometry options using TRITON/NEWT geometry modification data.

5.1 WHAT YOU WILL BE ABLE TO DO

- use GeeWiz and TRITON/NEWT to describe more advanced geometry models using geometry modification data and holes
- use CHORD keyword to truncate a body with a plane perpendicular to a major axis
- use ORIGIN keyword to translate the location of a body
- use ROTATE keyword to rotate a body to any angle
- use HOLE record to place one unit inside another unit
- use the geometry modification data to model common features of PWR and BWR lattices

5.2 USING THE CHORD KEYWORD TO TRUNCATE A BODY

The CHORD keyword truncates a geometric volume with a plane perpendicular to a major axis. The location of the cutting plane is specified by the subordinate keywords +X=, -X=, +Y=, and -Y=. The CHORD keyword applies to the geometry or hole record that immediately precedes it. Chords may not be applied to cuboids; users must simply specify a cuboid of the correct dimensions rather than using a chord to truncate a cuboid.

Each subordinate keyword specifies a cutting plane perpendicular to the axis in its name (e.g., "+X=" or "–X=" results in a cutting plane perpendicular to the *x*-axis). The cutting plane intersects the specified axis at the value that follows the "=" sign (e.g., "+X=5" or "-X=5" specifies a cutting plane perpendicular to the *x*-axis that crosses the axis at $x = 5$). The positive or negative sign before the axis name indicates on which side of the cutting plane the geometric volume is to be kept. For example, in Fig. 5.1, "-X=0" specifies a cutting plane perpendicular to the *x*-axis at $x = 0$ and keeps the volume on the negative side of the plane (i.e., $x \leq 0$). Conversely, "+X=0" specifies a cutting plane perpendicular to the *x*-axis at $x = 0$ and keeps the volume on the positive side of the plane (i.e., $x \geq 0$).

It is possible to apply multiple chords to a body by specifying more than one subordinate keyword to a body. Figure 5.2 shows an example where two chords are applied to a cylinder to obtain a quarter cylinder. The two subordinate keywords "+X=0" and "+Y=0" keep the quarter cylinder where $x \geq 0$ and $y \geq 0$.

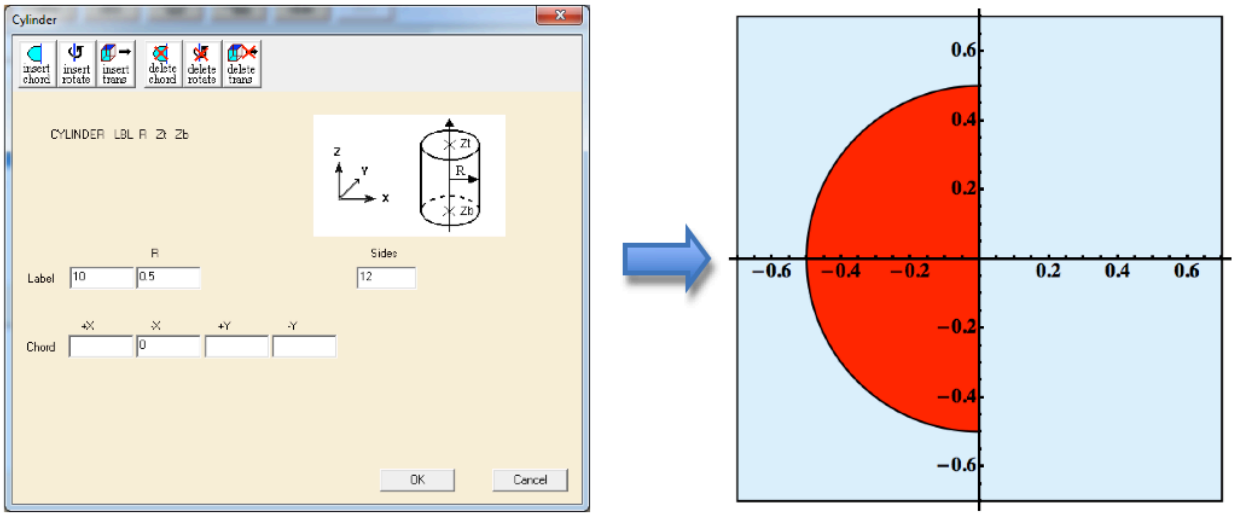


Fig. 5.1. Example of a -X=0.0 chord.

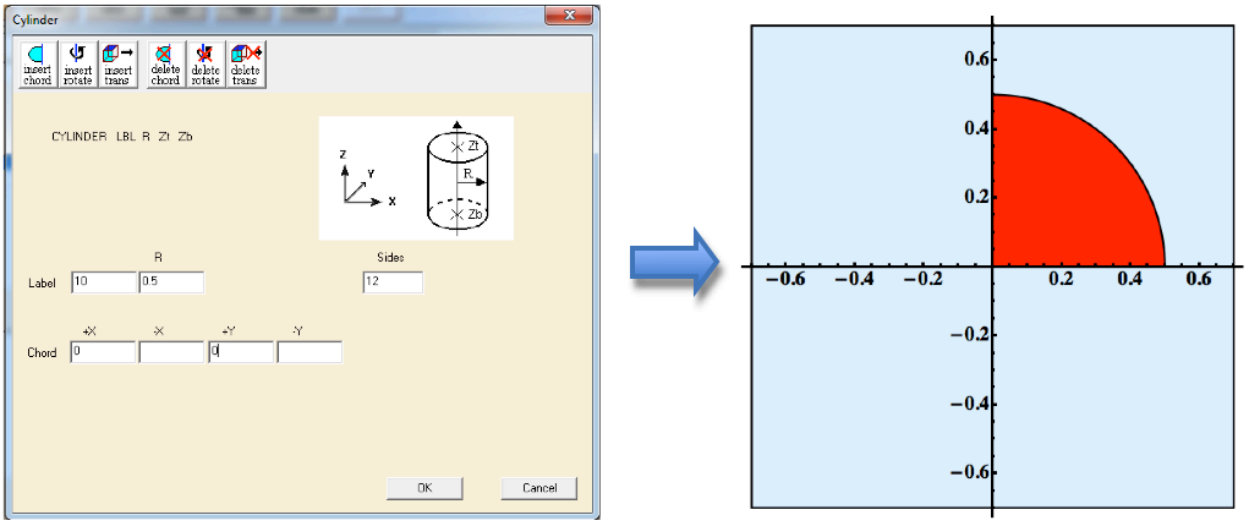


Fig. 5.2. Example of a +X=0.0 and +Y=0.0 chord.

5.3 USING THE ORIGIN KEYWORD TO TRANSLATE A BODY

The ORIGIN keyword allows the user to translate the origin of a TRITON/NEWT body to another position within a unit. ORIGIN translates the origin of a shape to a new position by moving the *x* and *y* coordinates of the body's origin to the position specified in the subordinate keywords X=, and Y=. The ORIGIN keyword applies to the geometry or hole record that immediately precedes it.

A simple example is illustrated in Fig. 5.3, where the origin of a 0.5 cm radius cylinder has been translated 0.25 cm in *x* and *y*, and truncated with chords to keep the northeast quarter of the circle. This example shows how chords and origins can be used in conjunction. Note that the chords must be moved separately—the chords will *not* translate with the ORIGIN keyword, only the shape will translate. In this example, the chords are specified at +X=0.25 and +Y=0.25, instead of +X=0.0 and +Y=0.0.

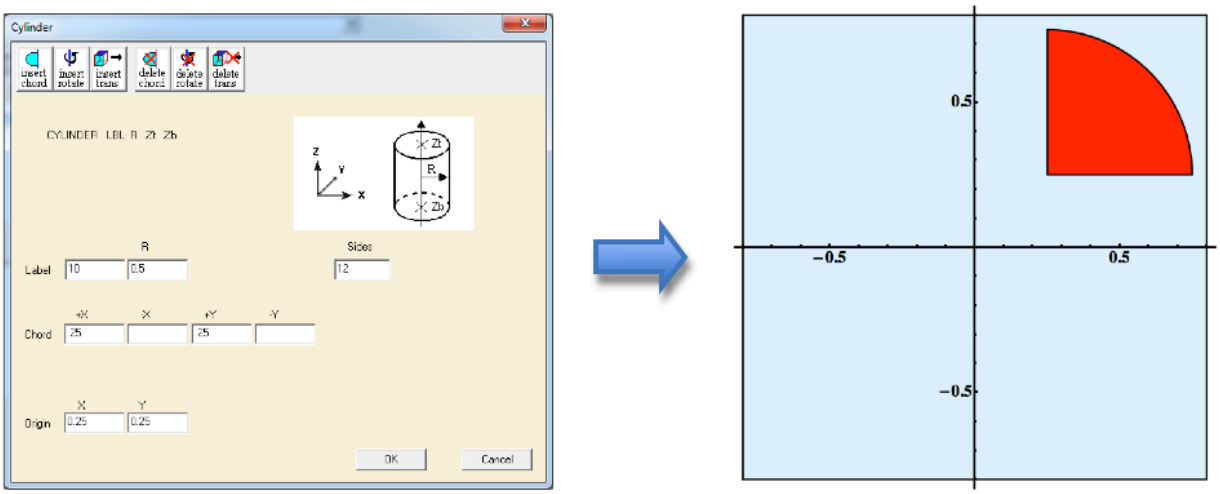


Fig. 5.3. Example using the ORIGIN and CHORD modifiers.

A more complex example is illustrated in Fig. 5.4, with three intersecting cylinders each with a radius of 9 cm. Cylinder 1 is translated 4.5 cm in the positive *y* direction; cylinder 2 is translated 4.5 cm in the negative *x* and *y* directions; and cylinder 3 is translated 4.5 cm in the positive *x* and negative *y* directions. The numbers on the top of each colored region are the region definition vectors (positive means inside, negative means outside).

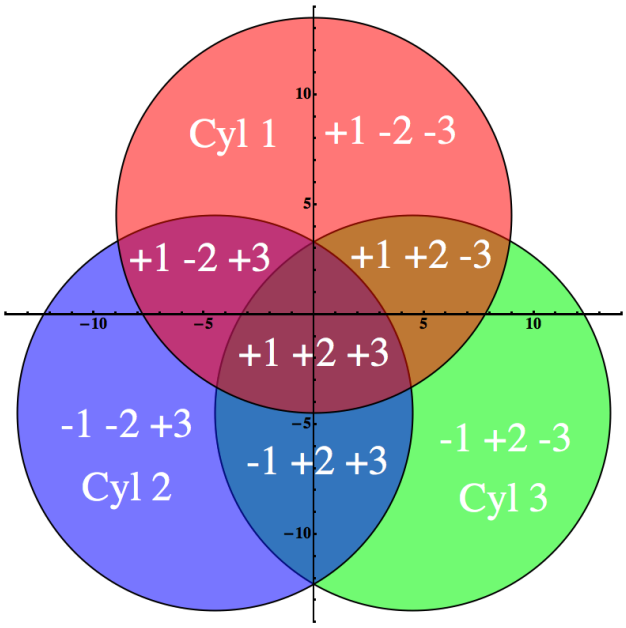


Fig. 5.4. Three intersecting cylinders example.

It is relatively easy to set up this example in GeeWiz, but it is of little value to lattice physics. It is, however, a good thought exercise for understanding how to construct complicated geometries that contain a combination of overlapping surfaces in TRITON/NEWT.

5.4 USING THE ROTATE KEYWORD TO ROTATE A BODY

The ROTATE keyword allows users to rotate the orientation of a TRITON/NEWT shape to any angle. It rotates a volume about the origin using the Euler angle-*x* convention. Like CHORD and ORIGIN, it applies to the geometry or hole record that immediately precedes it. Note that for any geometry record that contains both ORIGIN and ROTATE input, the rotation is performed in TRITON/NEWT first, prior to translating the origin of the body. The ROTATE keyword has only one subordinate keyword: A1=, which rotates a shape about the *z*-axis (out of the page).

Figure 5.5 shows the GeeWiz input and corresponding model for a 2 cm × 2 cm cuboid with origin located at (0,0) that is rotated 30 degrees counterclockwise around the *z*-axis.

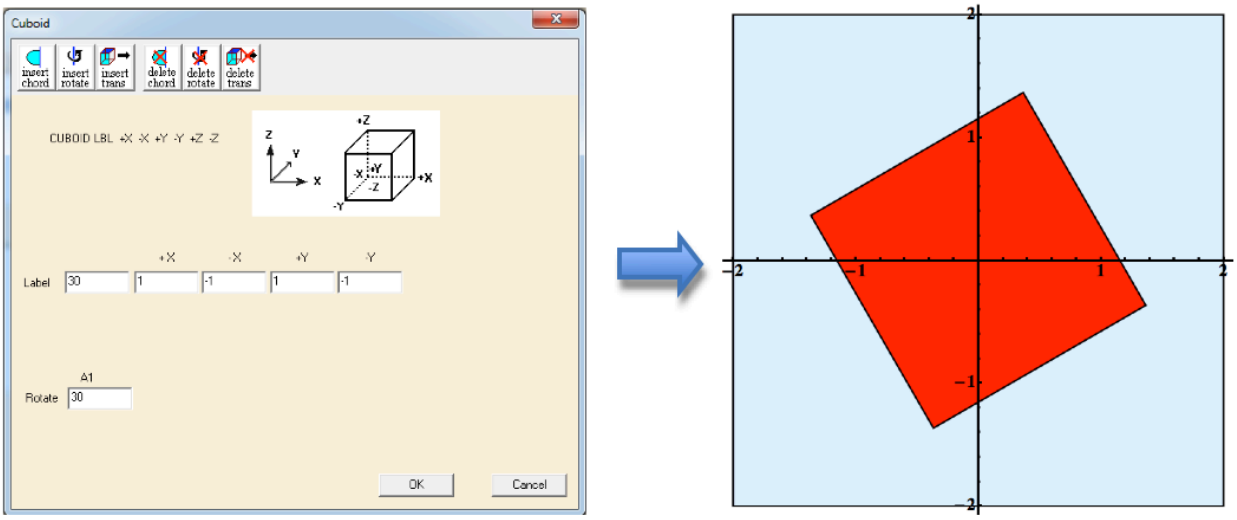


Fig. 5.5. Rotated cuboid centered at (0,0) example.

Remember that shapes are always rotated around the center of the shape. If a 2 cm cuboid's center is located at (1,1), the rotation will occur around (1,1) instead of (0,0). Figure 5.6 shows a cuboid that is 2 cm × 2 cm with the southwestern corner located at (0,0) and is rotated 30 degree counterclockwise about the *z*-axis.

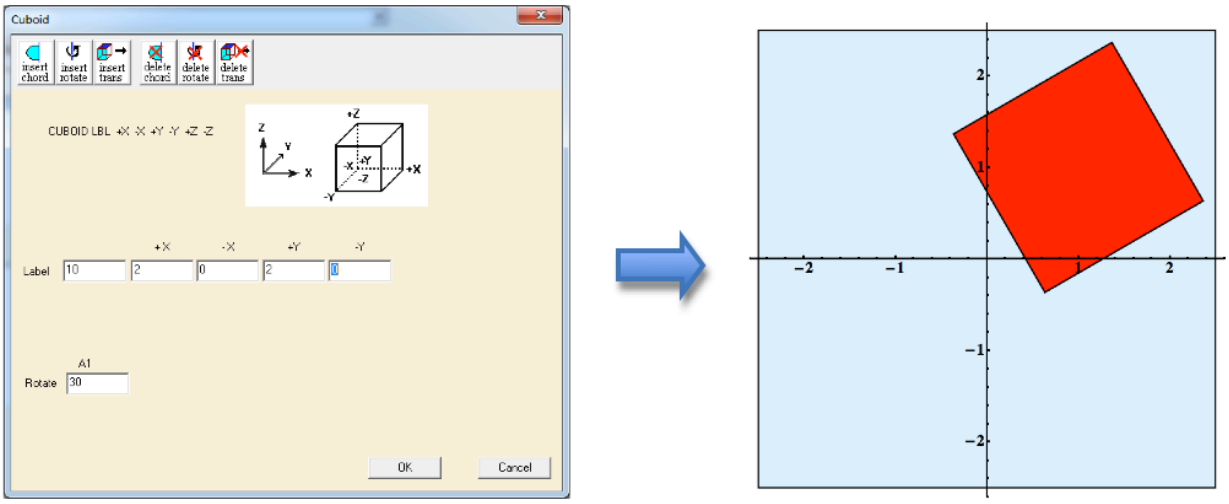


Fig. 5.6. Rotated cuboid centered at (1,1) example.

5.5 HOLES

There are three types of contents records for defining the contents of the geometry regions: media, arrays, and holes.

As you saw in Chapter 4, a media record is used to place a mixture inside a specified volume, and an array record is used to place an array inside a specified volume. For many problems, the use of media and arrays is adequate to create an accurate model.

In some situations, placement of one unit inside another unit may be necessary or helpful. For those situations, a hole is needed. A hole is used to position a unit within a surrounding parent unit relative to the origin of the surrounding parent unit. The keyword HOLE is followed by the unit number being placed inside the parent unit. The unit being placed with the hole can be rotated and translated using the previously discussed ROTATE and ORIGIN modifiers.

There are three basic rules for holes.

1. **A hole contains a single unit.**

2. **Holes may share surfaces with but may not intersect other holes or the boundary of the surrounding unit that contains the hole.**

3. **Holes may be nested (i.e., a unit containing a hole may be placed as a hole inside another unit).**

To illustrate the use of holes, use the model from mini2.inp, the mini-assembly model with gadolinium-bearing fuel, normal UO_2 fuel, and an empty lattice position. Place a hole in the global unit that is unit 10 at (0,0). This places a square hole of borated water at (0,0) and will cover ¼ of each of the other units (Fig. 5.7). It might seem as if this violates the last part of rule 2, but the hole will not intersect the boundary of the global unit in which it is inserted. It will only intersect the internal array structure.

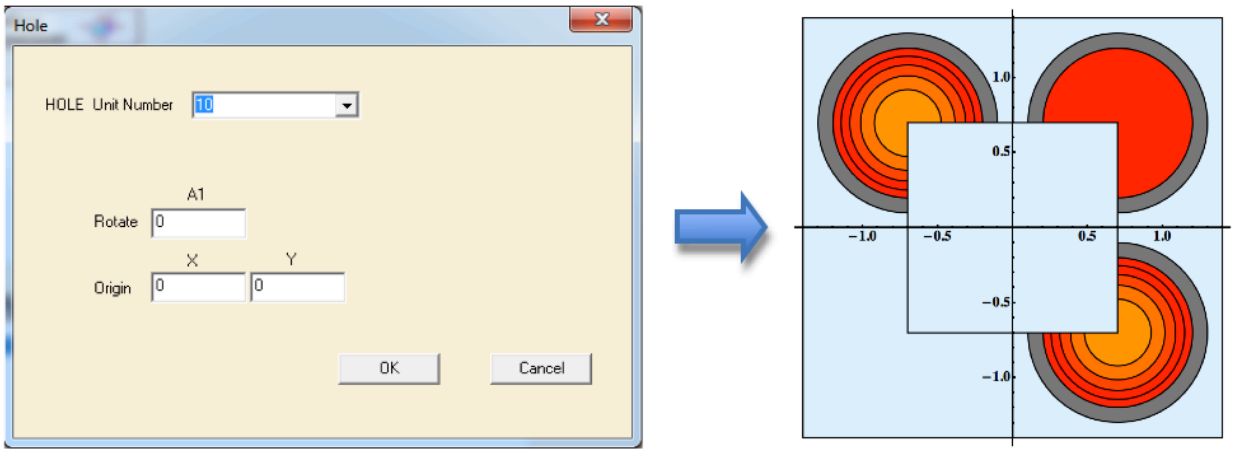


Fig. 5.7. Mini-assembly with a unit 10 hole placed at the origin of the global unit.

5.6 EXAMPLE PROBLEMS FOR A BWR LATTICE

Utilizing a series of slightly larger and more complicated sample lattice problems, the more advanced geometry features will be presented to build a more detailed lattice model. Each of the new features introduced previously in this section can play an important role in detailed lattice model development.

The basis of all the sample problems in this section is a simple 9×9 BWR lattice. This lattice uses the same unit cells that were used in the mini-assembly problem.

5.6.1 Basic 9×9 Assembly

For this sample problem, use the previously built `mini2.inp` input file as a starter. Open mini2.inp in GeeWiz and resave the file to `assembly1.inp`. You will be building a 9×9 lattice, but for this problem you will leave some extra space around the boundary of the lattice. You will use this space later to insert yet more complicated geometry features.

Click the **Arrays** button on the left toolbar. There is only one array in the model at this point so the correct array will already be selected in the **Array Form**. Now click **Modify** from the top horizontal toolbar of the **Array Form**. Clicking **Modify** opens the **Array Properties** window. In this window, specify the array as **NUX**=`9` and **NUY**=`9` and initialize the new array locations to unit `1` and then click **OK** (Fig. 5.8).

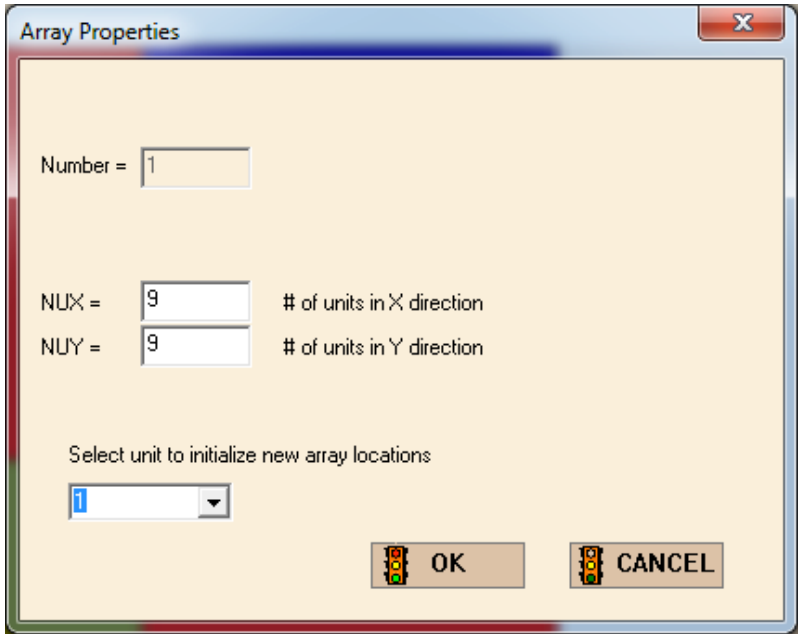


Fig. 5.8. Array properties window for the 9×9 assembly.

The **Array Form** will now show a 9×9 array. You might need to use the **Zoom** buttons along the top horizontal toolbar of the **Array Form** to zoom out to see the entire fuel array. You will notice that GeeWiz does not change the previously specified units from the 2×2 array (Fig. 5.9).

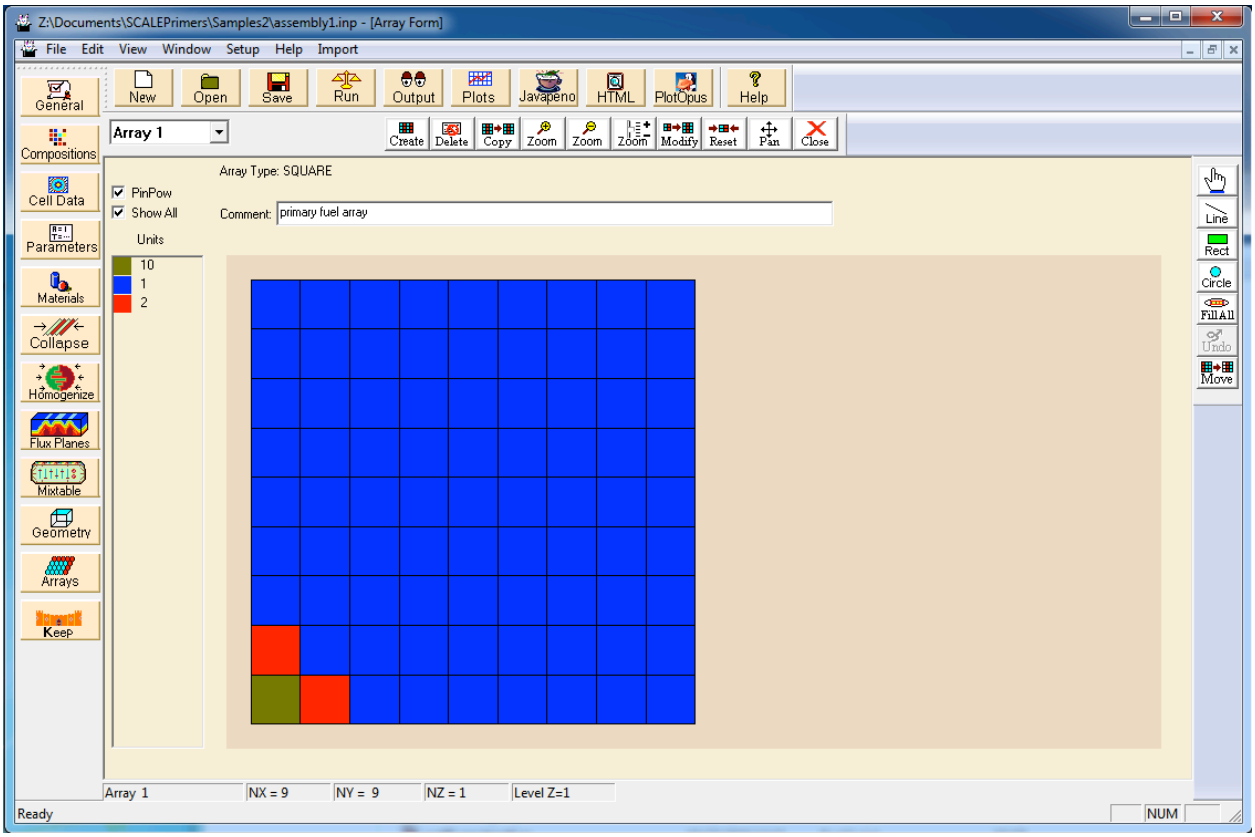


Fig. 5.9. Array form for the 9×9 array.

Modify the array such that the entire array is unit `1` (all blue). You can use the **Finger** button or the Fill All button on the right toolbar to do this. Once the entire array is blue, click **Close** along the top horizontal toolbar of the **Array Form**.

Now change the global unit of the model to accommodate the larger array. Click **Geometry** from the left vertical toolbar and select `100(global unit)` from the **Select unit** dropdown. Although the array is 9×9 with 1.4 cm unit cells, specify the global unit cuboid size corresponding to an `11x11` array (`15.4` cm x `15.4` cm) centered at (`0,0`), i.e., -X and -Y = -7.7 and +X and +Y = 7.7. You should also change the placement of the array so that the center of the central unit of the array, (**nx,ny**)=(`5,5`), is placed at (`0,0`). Also change the grid structure of the global unit boundary; the grid structure should be changed from `4×4` to `11x11`. The resulting **Geometry Form** for unit 100 (global unit) should look like Fig. 5.10. Also note that borated water has been used for the moderator in this model. Typically, BWR moderator does not contain boron, but the borated water was used for this example in favor of adding another material that is water without boron. This example is meant to show how to model common geometry features in BWR lattices rather than the mixtures that are contain in the lattices.

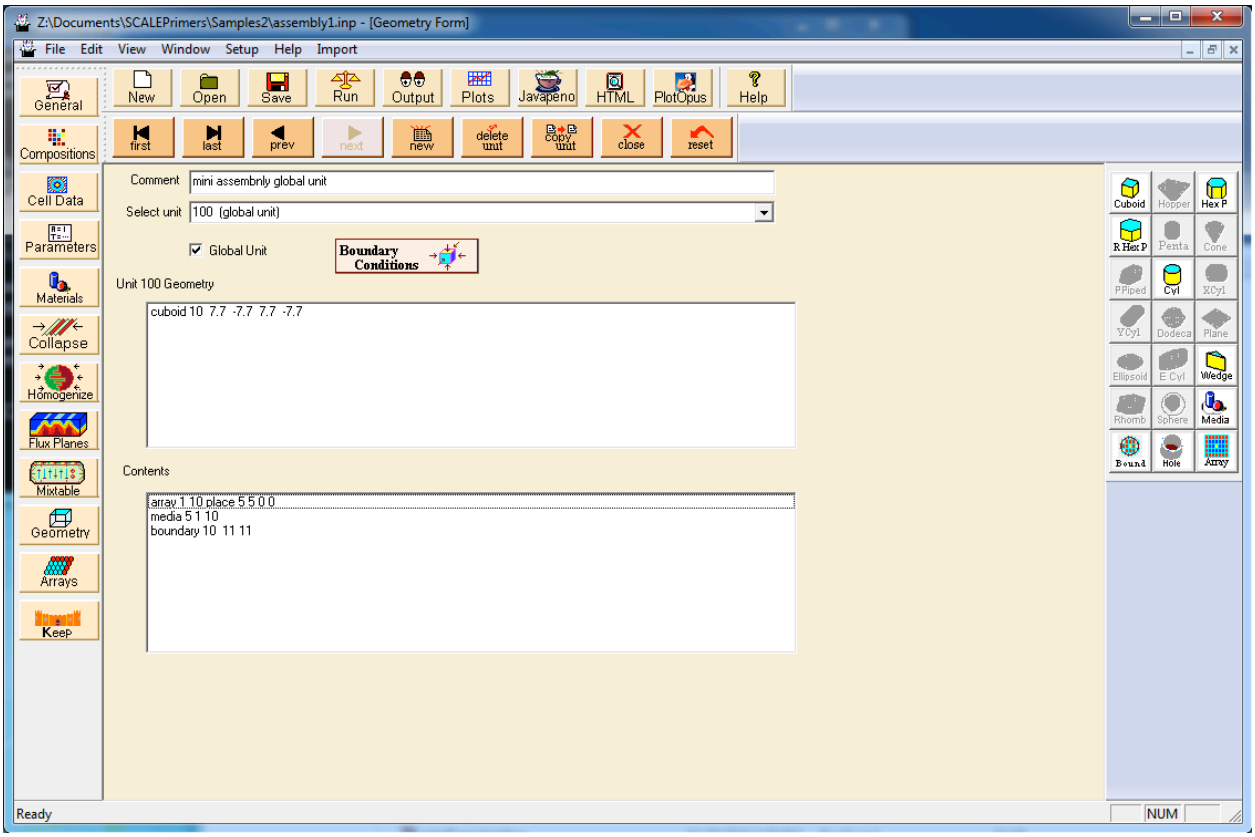


Fig. 5.10. Geometry form for the 9×9 lattice model.

Now go to the **General** parameters, and check the **Parm=check** checkbox in order to have TRITON/NEWT check and plot the geometry. You can now click **Run**. Verify that the model check was successful and open the `assembly1.newtmatl.ps` plot file. You should see a 9×9 array of fuel pins with space for water around the outside of the lattice (Fig. 5.11).

If everything looks okay, uncheck **Parm=Check** and rerun the problem. This problem will take longer to complete than the mini-assembly models because it is a full assembly rather than a 2×2 lattice. Once the problem is completed, open the **output** file and find the total runtime and the NEWT transport solution runtime, along with the overall k_{inf}. This sample problem ran in 157.52 seconds with a NEWT transport solution converging in 71.69 seconds. The eigenvalue calculated by TRITON/NEWT is 1.19561741.

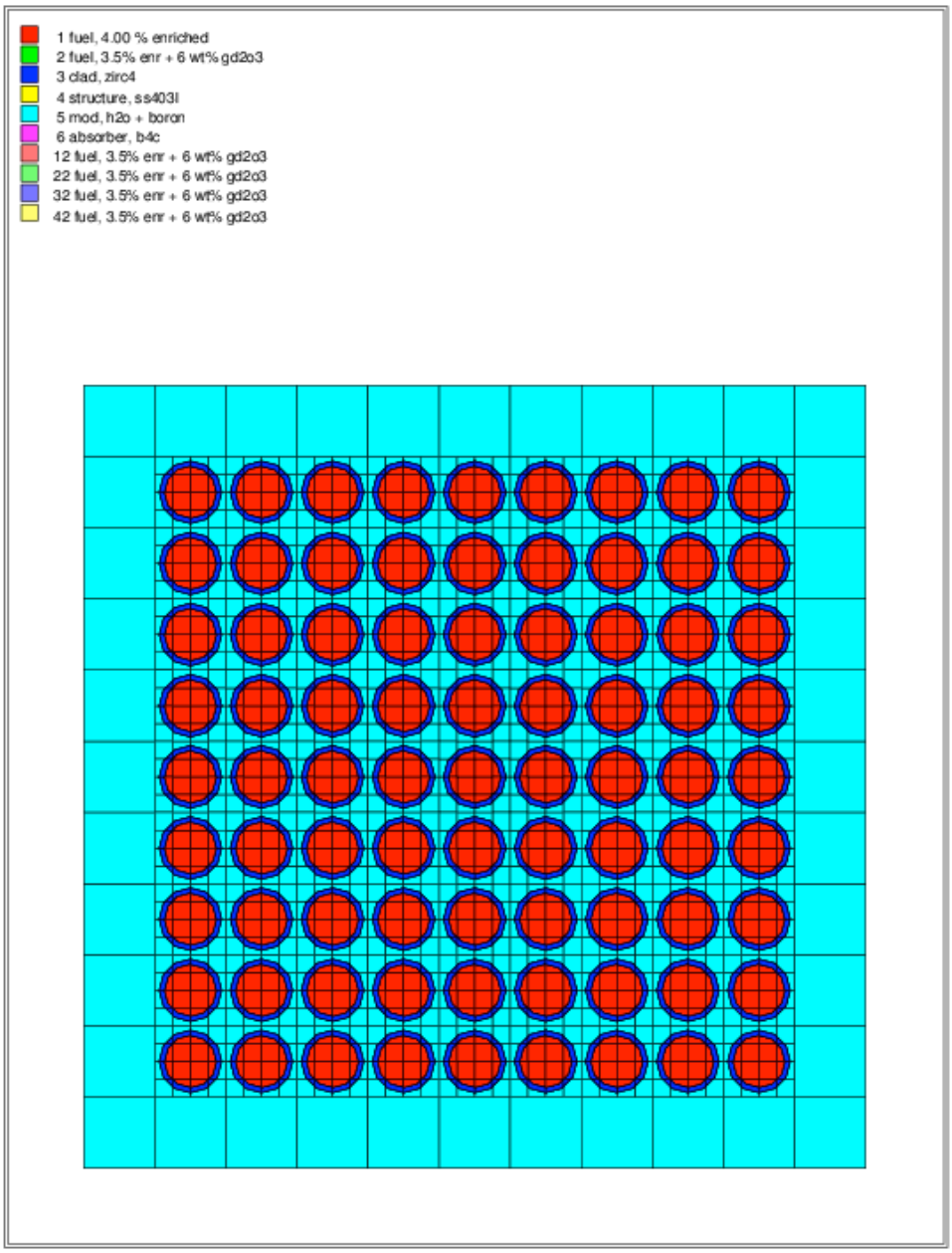


Fig. 5.11. TRITON/NEWT plot of the 9×9 lattice model.

5.6.2 Basic 9×9 Assembly—¼ Lattice

Fuel assemblies are often symmetric, especially in PWRs. It is possible to use symmetry to decrease the size of some lattices while maintaining accuracy. Many PWR lattices have 1/8 assembly symmetry. Using 1/8 symmetry would require use of a triangular outer boundary, which is difficult to build in TRITON/NEWT, so this primer focuses on 1/4 symmetry. If the model has 1/4 symmetry, the overall size of the model can be decreased by 75%, which decreases the number of cells and the NEWT transport solution runtime by approximately 75%.

Use **File>Save As** to save `assembly1.inp` with the filename `assembly1_quarter.inp`. For this problem, the northeastern ¼ of the assembly will be modeled, as seen in Fig. 5.12.

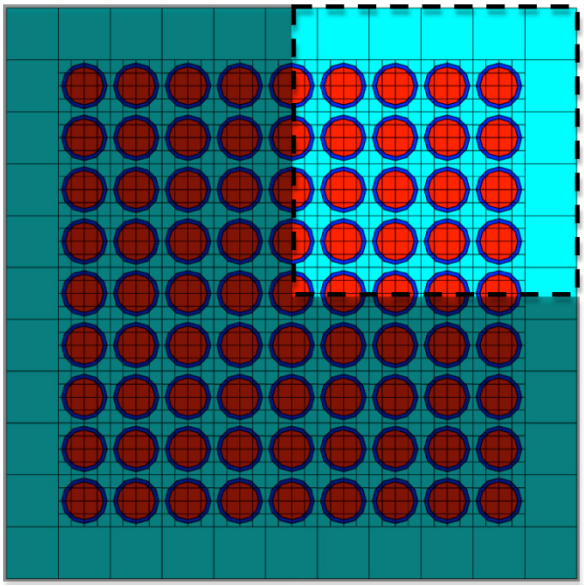


Fig. 5.12. Portion of the 9×9 lattice that will be modeled with the ¼ lattice model.

To construct this model, you need three new units: the +Y=0 side (north half) of a unit cell, the +X=0 side (east half) of a unit cell, and the +Y=0 and +X=0 sector of a unit cell (northeastern quadrant). These cells will form the southern boundary, western boundary, and southwestern corner of the model, respectively.

To add these units, click **Geometry** to open the **Geometry Form**. It is easiest to copy the base fuel unit to new unit numbers and then modify the units, rather than building three new units. Select `unit 1` from the **Select Unit** dropdown and then click **Copy Unit** along the horizontal toolbar. Insert a descriptive comment such as "`north half unit 1`". You can use whatever unit number you like; this example uses `101`. You will need to add a **Chord** to cylinder 10 and cylinder 20 to keep the north half (`+Y=0 chord`) as shown in Fig. 5.13.

Truncating a cuboid with a chord is not allowed in TRITON/NEWT so you need to modify the size of the cuboid to keep only the northern half. You will need to move the –Y plane from `-0.7` to `0.0` cm. You should also change the boundary grid structure. This unit cell is now halved in the *y*-direction, so you should divide the number of Y grids by 2. The final **Geometry Form** for the north half unit should look like Fig. 5.14.

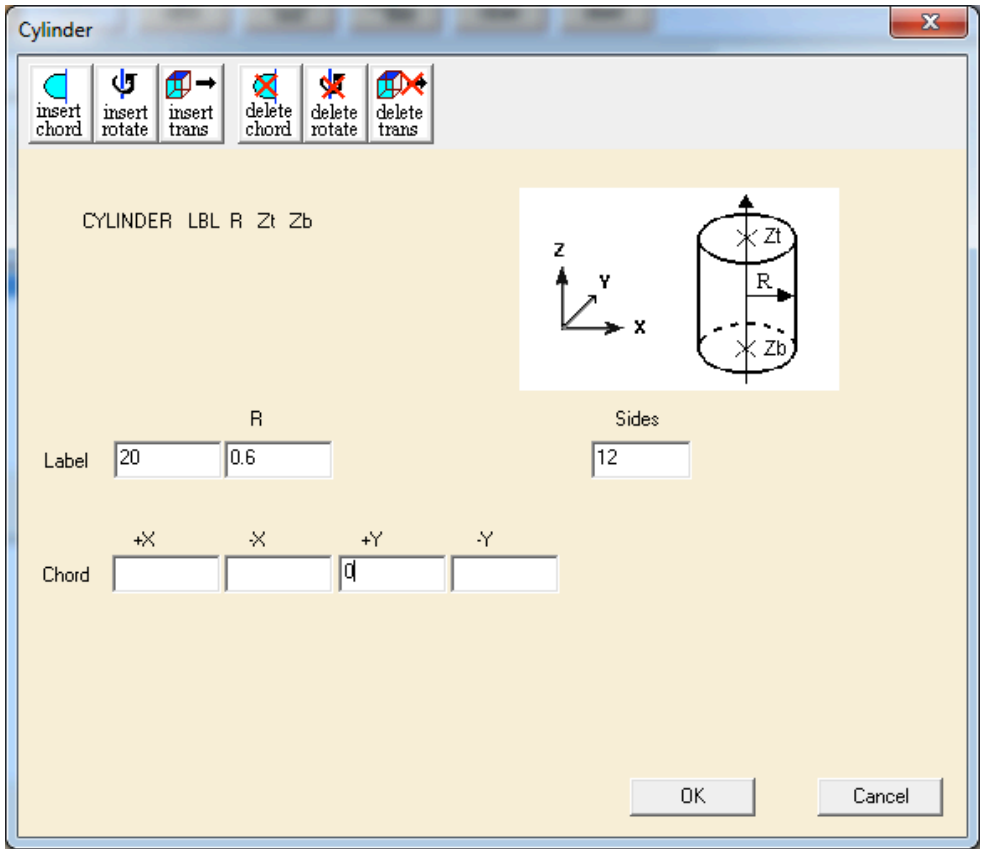


Fig. 5.13. Half fuel pin using +Y=0 chord.

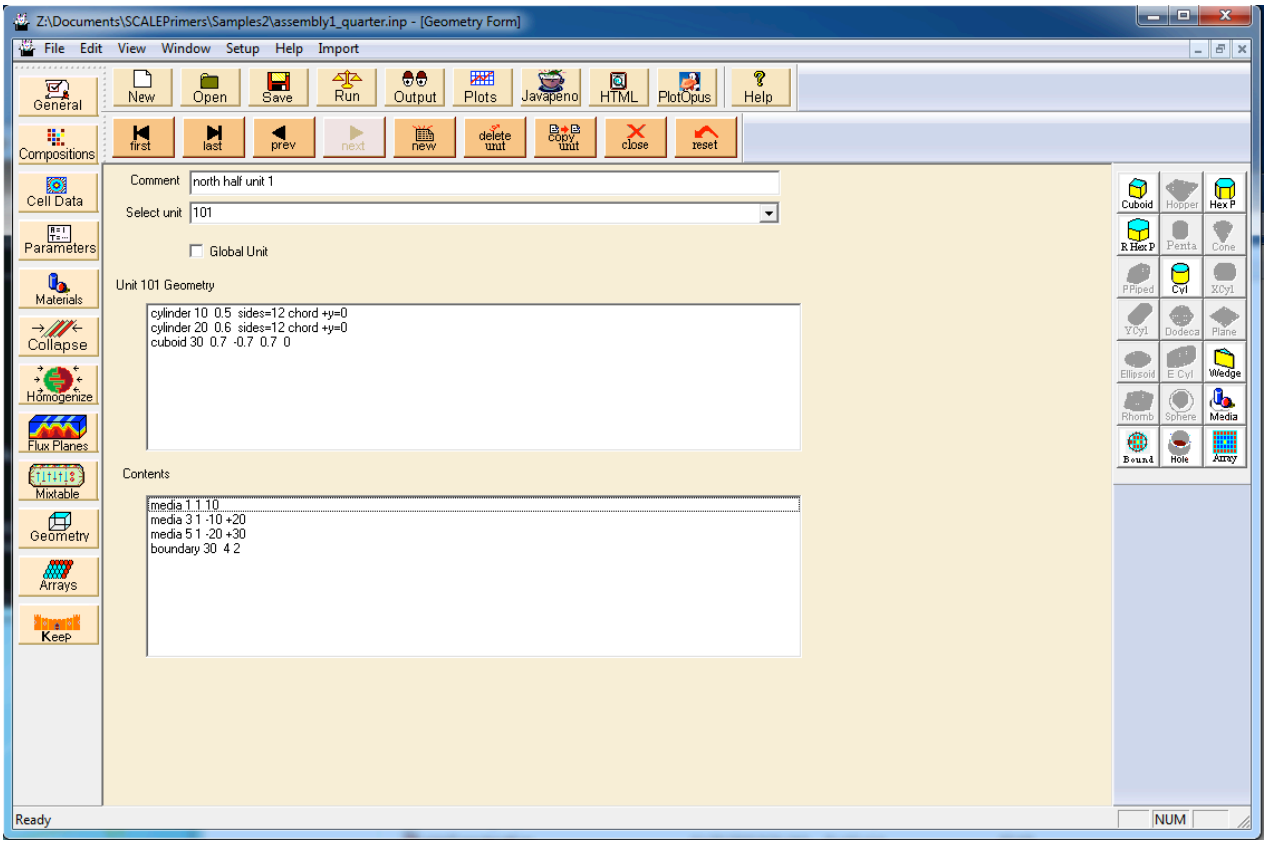


Fig. 5.14. Geometry form for north half pin cell.

Using the same procedure, add two more new units for the east half unit cell and the northeast quarter unit cell. The final **Geometry Forms** for these units can be found in Figs. 5.15 and 5.16.

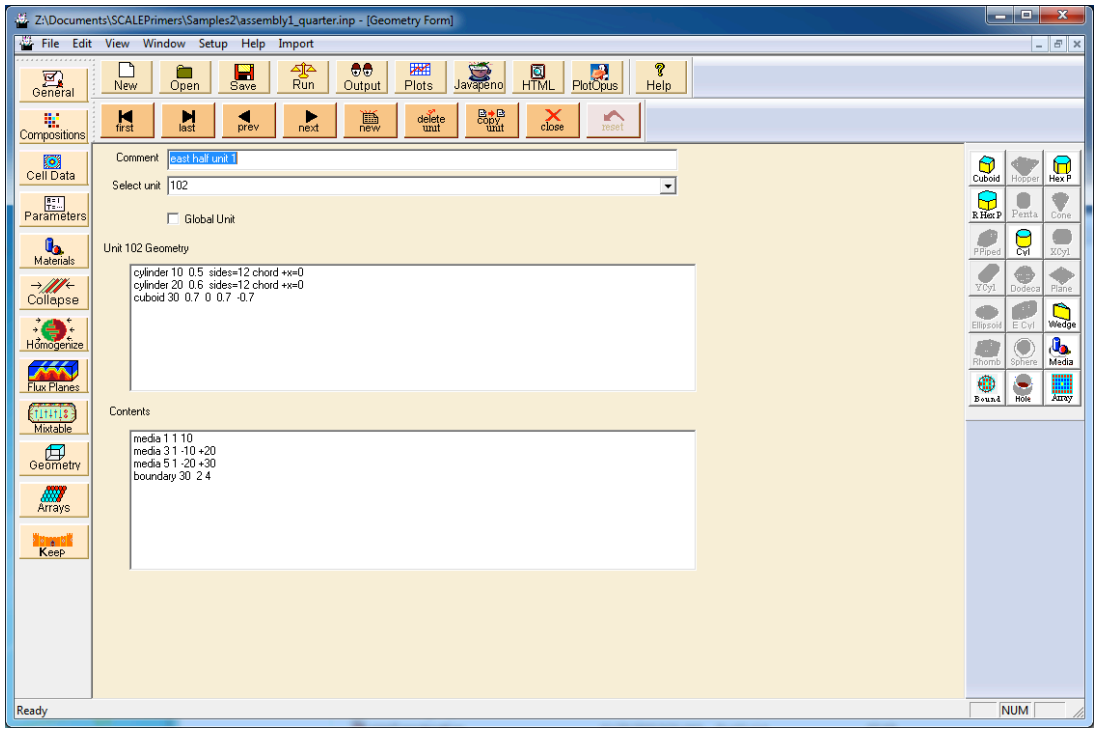


Fig. 5.15. Geometry form for east half pin cell.

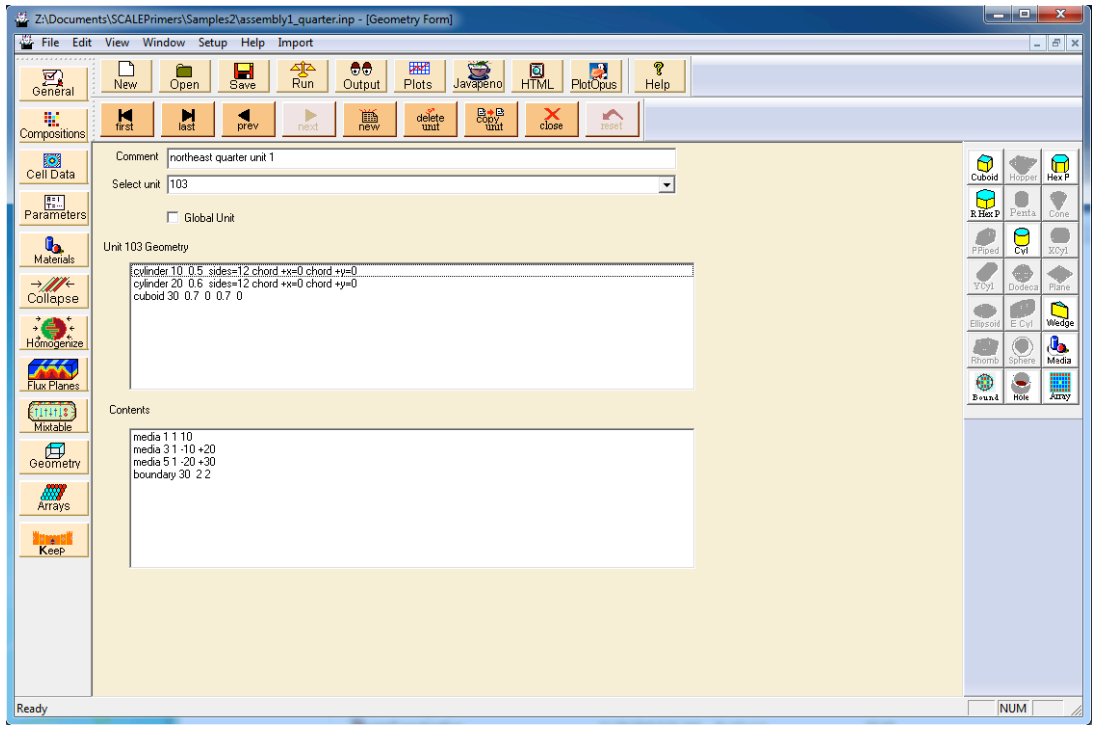


Fig. 5.16. Geometry form for northeast quarter pin cell.

Now that you have all the units to construct a quarter assembly model, you need to change size of the global unit to accommodate the smaller lattice. You can change all global unit cuboid dimensions so the center of the unit still lies at (0,0); however, since this is a quarter assembly, you might find it easier to move the -X and -Y cuboid boundaries to 0. This means that the southwestern corner of the global unit will lie at (0,0). Because the size of the array has changed, you also need to change the array placement so that array index (**nx,ny**)=(1,1) lies at (**X,Y**)=(0,0) of the global unit. You should also change the global unit grid to 6×6 to accommodate the smaller array. You do not need to change the boundary conditions—they are all set to reflective from the assembly1.inp problem. The final **Geometry Form** for 100 (global unit) can be found in Fig. 5.17.

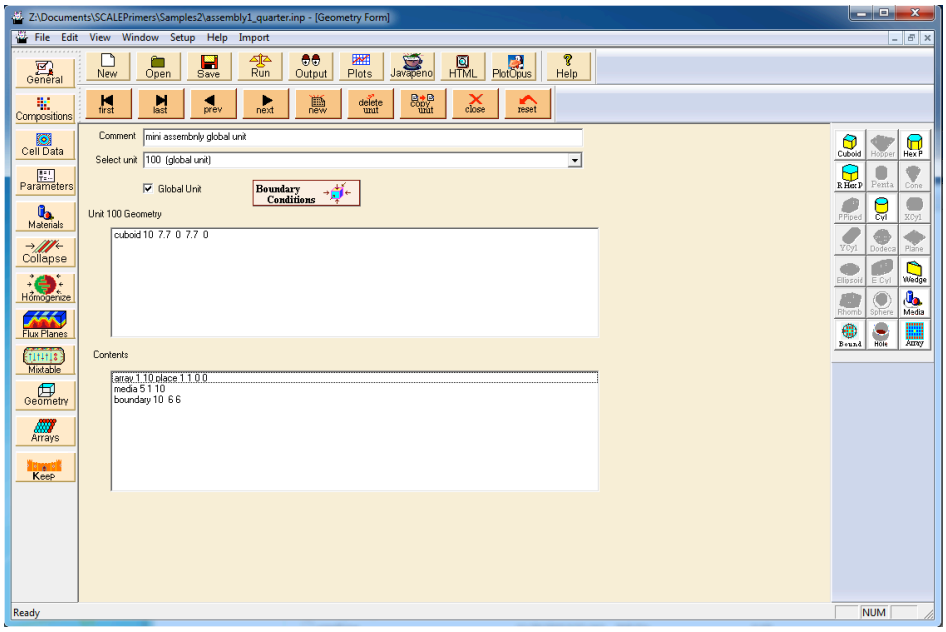


Fig. 5.17. Geometry form global unit of the quarter 9×9 assembly.

Now click the **Arrays** button on the left toolbar to change the fuel array. Click **Modify** along the top horizontal toolbar to open the **Array Properties** window. For **NUX** and **NUY** enter 5 and 5 to specify a 5×5 array; also initialize the array to `unit 1` and click **OK** (Fig. 5.18).

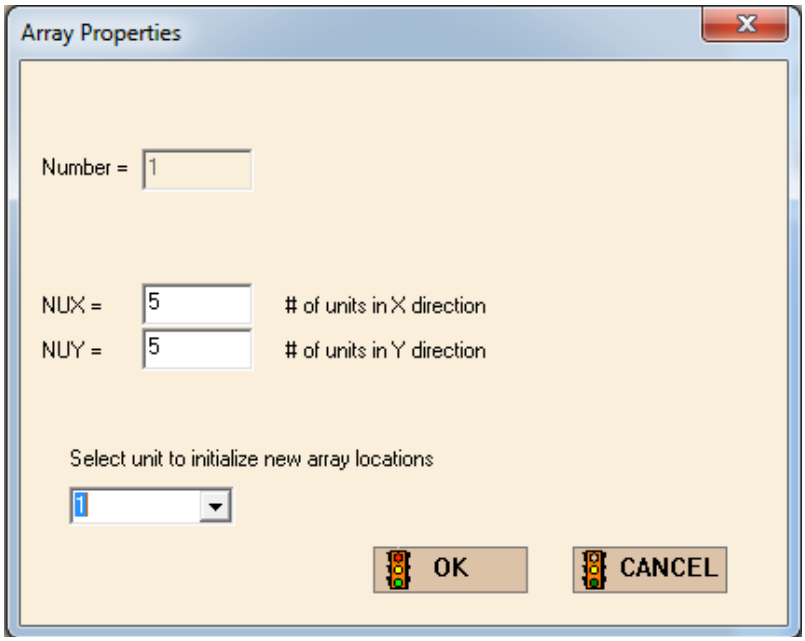


Fig. 5.18. Array properties window for the quarter 9×9 assembly.

The array should be initialized to unit 1 (all blue). You can use the **Finger** or **Line** tool along the right vertical toolbar to select unit `101`, `102`, or `103` and place them in the correct positions. Do not worry that all of the units in the array are not the same size. As long as all of the X widths are the same in a column and all the Y heights are the same in a row, then the array will be constructed properly. The final **Array Form** should have unit `103` at the southwestern corner, unit `101` along the south edge, and unit `102` along the west edge (Fig. 5.19). If everything looks correct, close the **Array Form**.

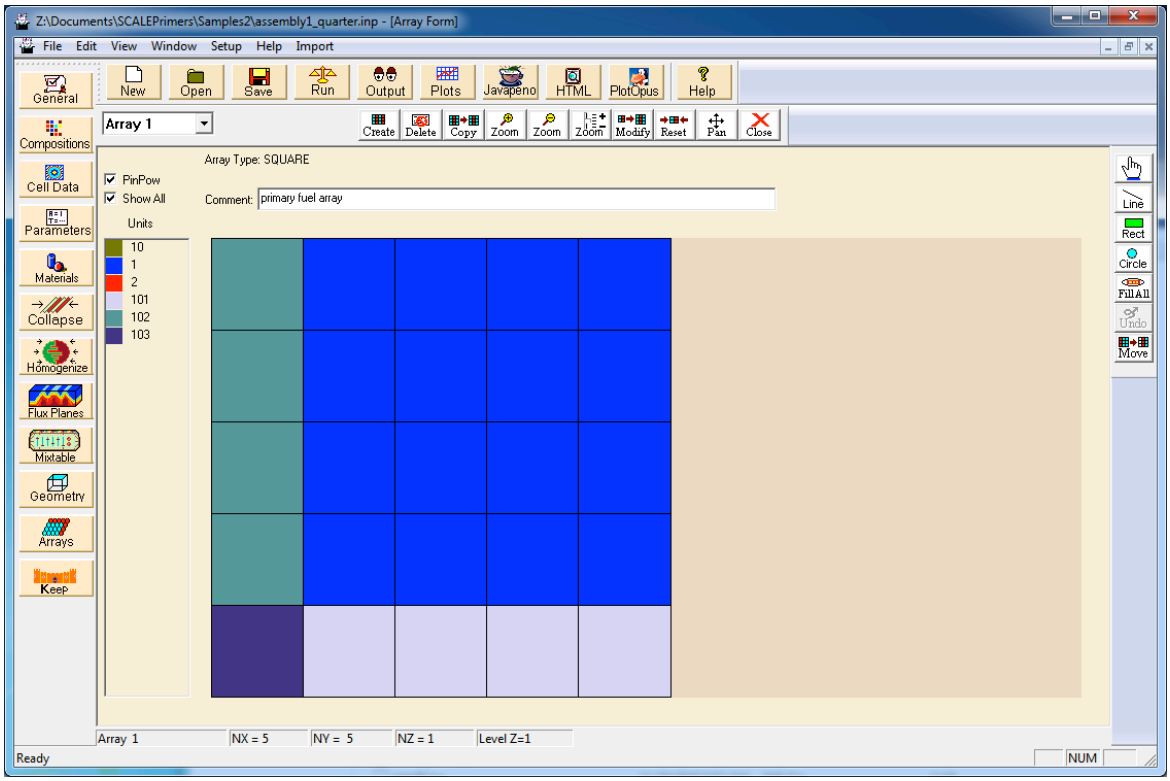


Fig. 5.19. Array form for the quarter 9×9 assembly.

Before running the problem, you should plot the model by turning checking the **Parm=Check** checkbox in the **General** settings. You can then **Run** the case. When TRITON/NEWT completes checking the model, open the PostScript file to view the plot of the geometry, which should resemble Fig. 5.20. If the model looks correct and the output file does not report any errors, return to the GeeWiz **General** settings and turn off **Parm=Check**, then rerun the problem. Once the problem completes, open the output file and make note of the total CPU time, the NEWT transport solution time, and the overall k_{inf}. This problem ran in 98.50 seconds of total CPU time, including 26.66 seconds for NEWT, and resulted in an overall k_{inf} of 1.19026604. The eigenvalue is slightly different than the full assembly case due to the global grid spacing. The global grid spacing changed slightly in the quarter assembly case because the model is now 5.5 unit cell pitches wide (we used 6×6 grid), rather than a whole number (11 cell pitches) as before. Also, the CPU time for the NEWT transport solution decreased from 72 seconds to 27 seconds—a 62.5% decrease in the CPU time required for the NEWT transport solution. Although it is not a full 75% decrease in CPU time, the computational cost savings is significant.

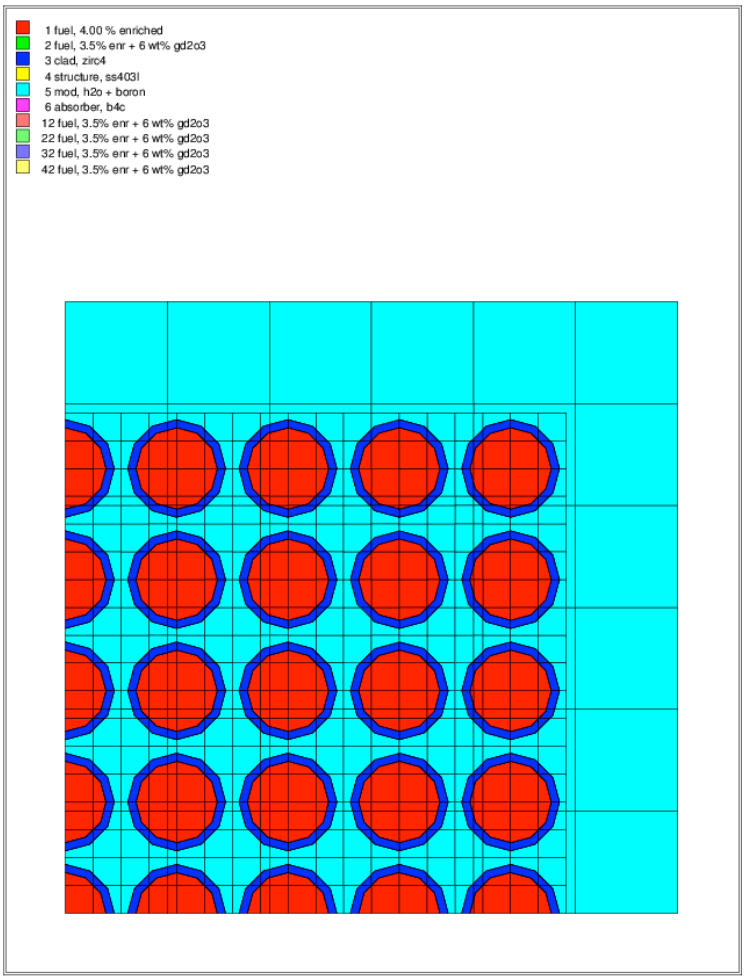


Fig. 5.20. TRITON/NEWT plot of the quarter 9×9 assembly.

5.6.3 Basic 9×9 Assembly with Water Rods

For added complexity, water rods will now be added to the full 9×9 fuel assembly. Reopen the `assembly1.inp` input file in GeeWiz and **Save As** `assembly1_wr.inp`. The water rods will have a 2.6 cm outer diameter and a wall thickness of 0.1 cm. The water rod structure should be Zircaloy-4 filled with borated water. The two water rods will be centered at (-1.5*P, -1.5*P) and (1.5*P, 1.5*P), where P is the fuel pin pitch, which in cm are at (`-2.1,-2.1`) and (`2.1, 2.1`).

Water rods are common in BWRs, and they often eliminate the symmetry of a lattice. So you will not be able to decrease the size of this lattice with symmetry. Using a **Hole** is the simplest way to insert a water rod into the lattice. You should specify lattice cells filled with water in the array to act as the background of the **Hole**, and then specify a **Hole** in the global unit to place the water rod at the correct location.

With `assembly1_wr.inp` open, click **Geometry** to open the **Geometry Form**. Click **New** to add a new unit for the water rod. Provide a descriptive comment in the **Comment** box and use `101` as the **unit** number. Now click **Cyl** from the right toolbox; add a cylinder with radius of `1.2` cm and another with radius of `1.3` cm. Fill the inside of the first cylinder with mixture `5 h2o,boron` and the ring inside

the second cylinder and outside the first cylinder with mixture `3 zirc4` as the structural material. The final **Geometry Form** for the new water rod unit should look like Fig. 5.21.

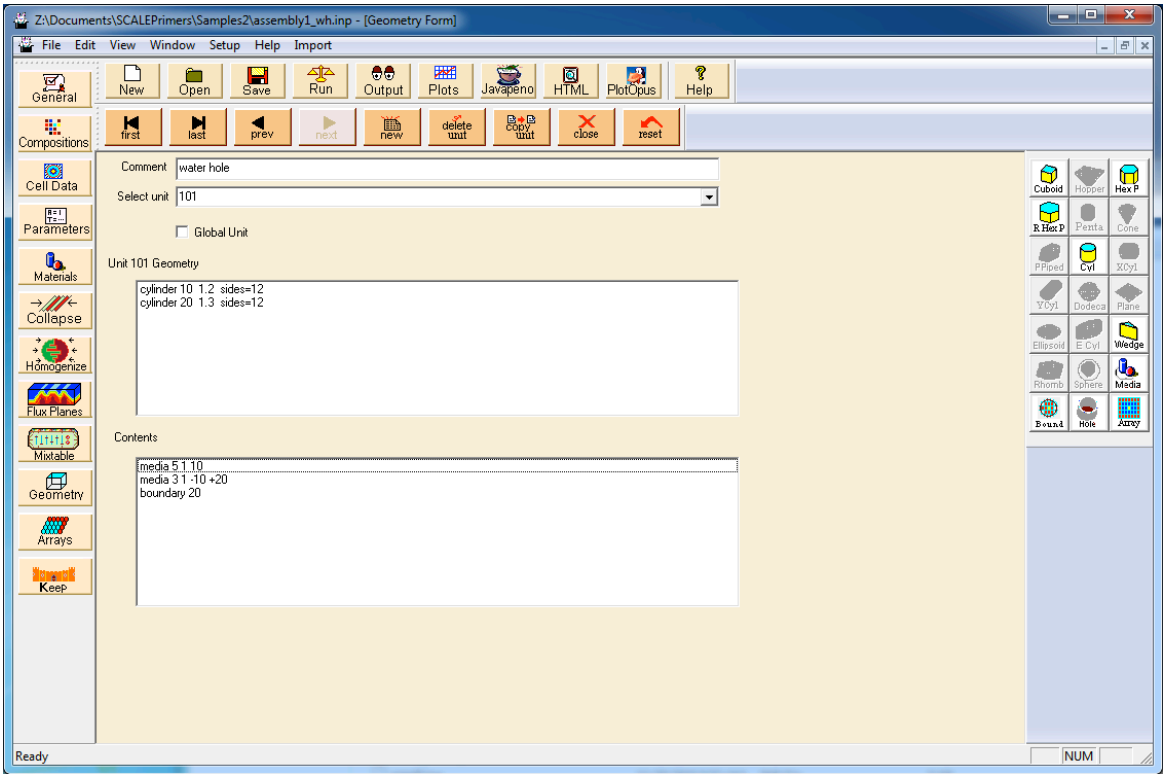


Fig. 5.21. Geometry form for the water rod unit in the 9×9 assembly.

Note that the boundary grid for this unit has been left blank, you can allow the underlying unit cell grid and the global grid to act as the grid for this unit. It is not possible to construct a unit grid for this circular unit that will fit well with the global grid. Adding a grid for this unit would result in a number of extra cells for which NEWT would calculate flux, resulting in increased runtime.

Click on **Arrays** to open the **Array Form**. You may use the **Finger** or **Line** tool on the right toolbar to select a block of units. The **Line** tool allows you to click and drag the mouse over a collection of units. Click `10` in the **Units** box to specify that unit `10` will be inserted into the array. Now select the four units that surround each of the two points where the centers of the water rods will be located (Fig. 5.22). Selecting these units specifies that the area of the array under the water rods (the background) should be water.

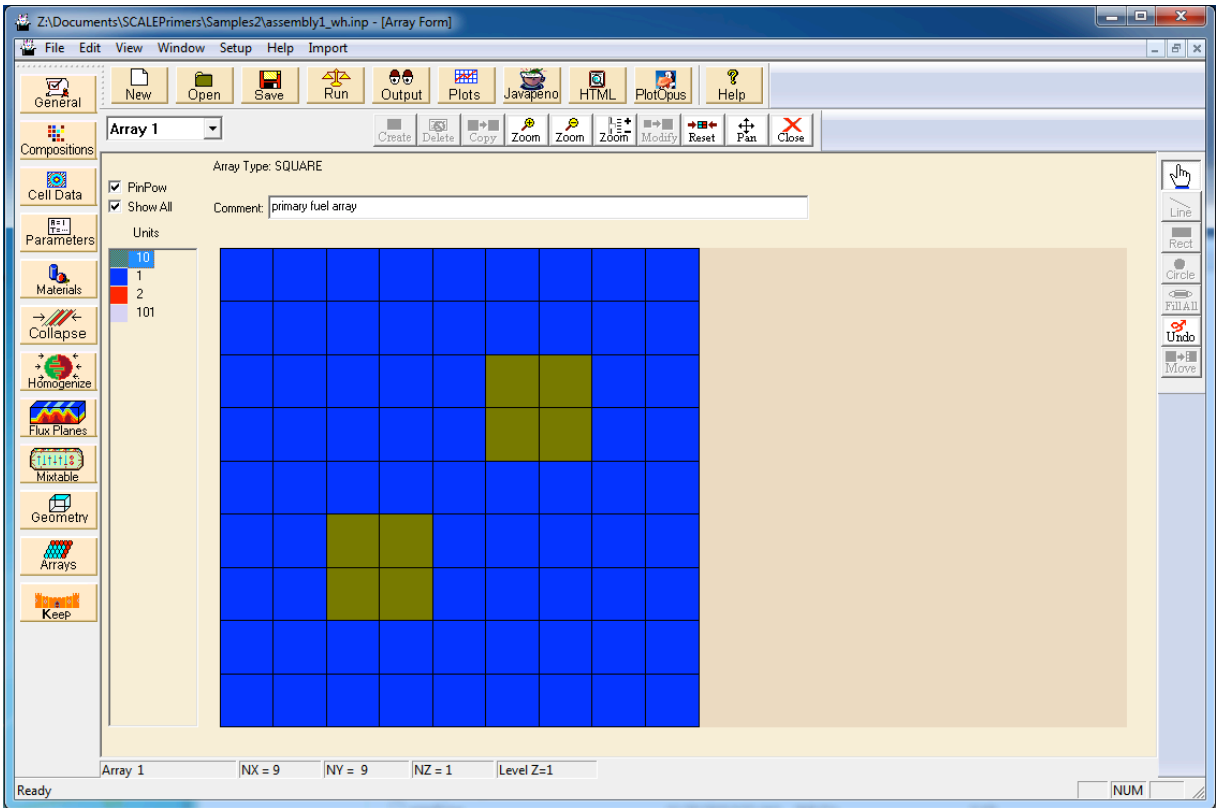


Fig. 5.22. Array form for the 9×9 assembly with water rods.

If the **Array Form** looks like Fig. 5.22, click **Close** and return to the **Geometry Form**. **Select** `100(global unit)` from the **Select Unit** dropdown. Now click **Hole** from the right toolbox to specify the two holes that are needed to place the water rods. In the **Hole** window, specify `101` for the **Hole Unit Number** and the **Origin** of the hole should be (**X,Y**)=(`-2.1,-2.1`) and then click **OK** (Fig. 5.23).

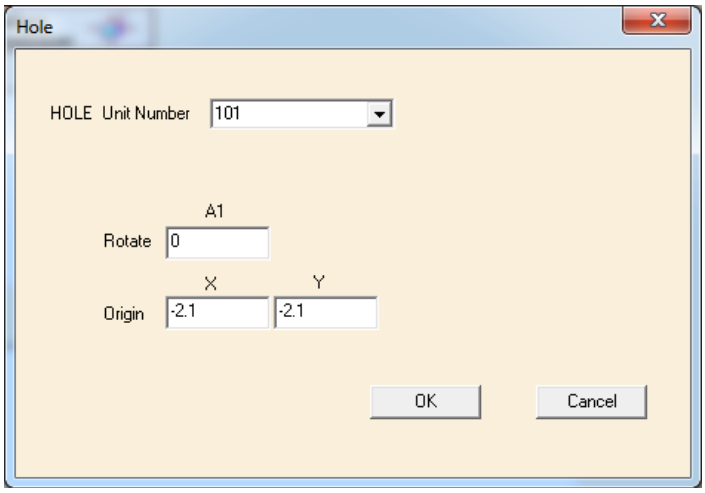


Fig. 5.23. Hole window for the lower left water rod in the 9×9 lattice with water rods.

Now click **Hole** again to add a second `unit 101` hole with `origin` located at **(X,Y)**=(`2.1,2.1`), and then click **OK**. Now you have two holes that are `unit 101` in the global unit. The final **Geometry Form** for the global unit should look like Fig. 5.24.

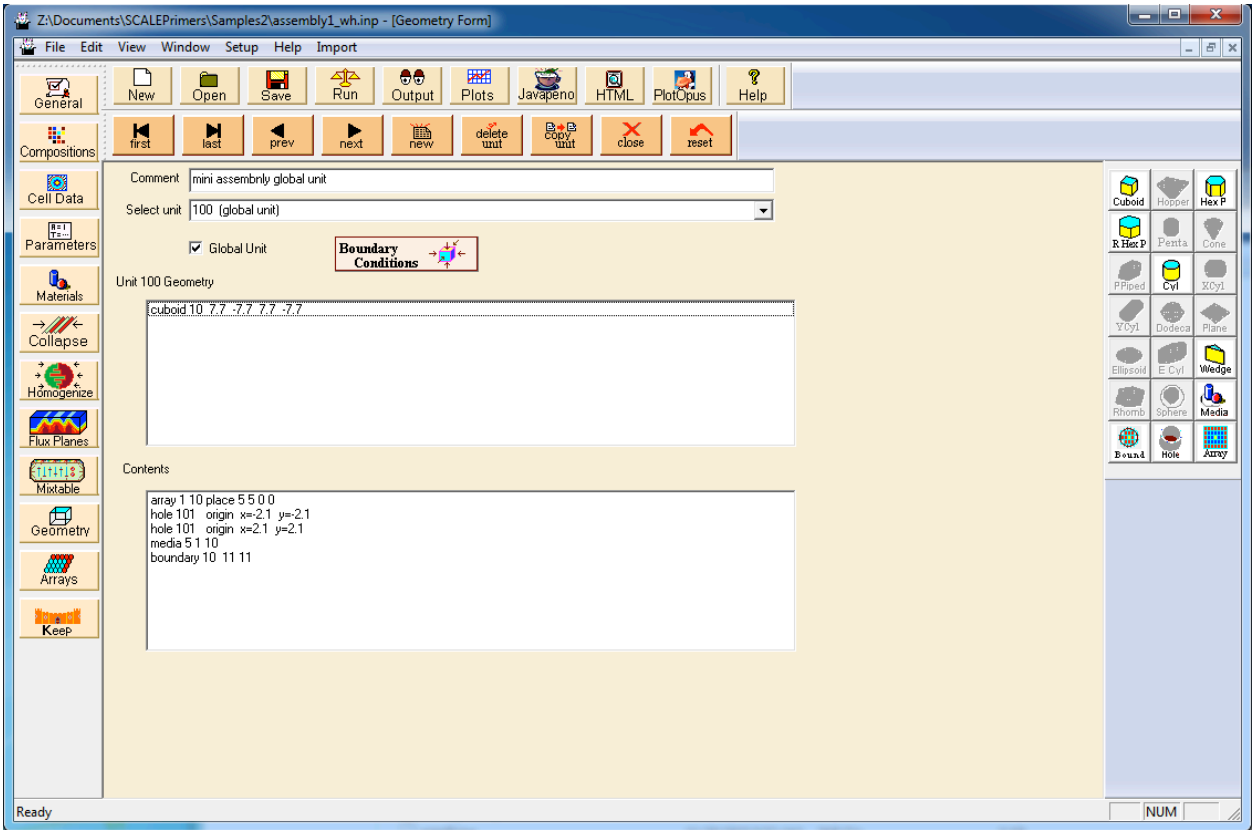


Fig. 5.24. Final geometry form for the 9×9 lattice with water rods.

Go to the **General** settings, check the **Parm=Check** checkbox to have TRITON/NEWT check and plot the geometry. Click **Run** to save the model and run the case. Open the output file to make sure the case completed without errors. Then open the PostScript file `assembly1_wr.newtmatl.ps`. You should see that the water rods have been placed in the correct location (Fig. 5.25). Turn off **Parm=Check** in the **General** settings and **Run** the problem again. Upon completion, record the CPU times used and the final k_{inf}. This example finished in 159.08 seconds with 78.90 seconds consumed by the NEWT transport solution. The calculated k_{inf} is 1.19639958.

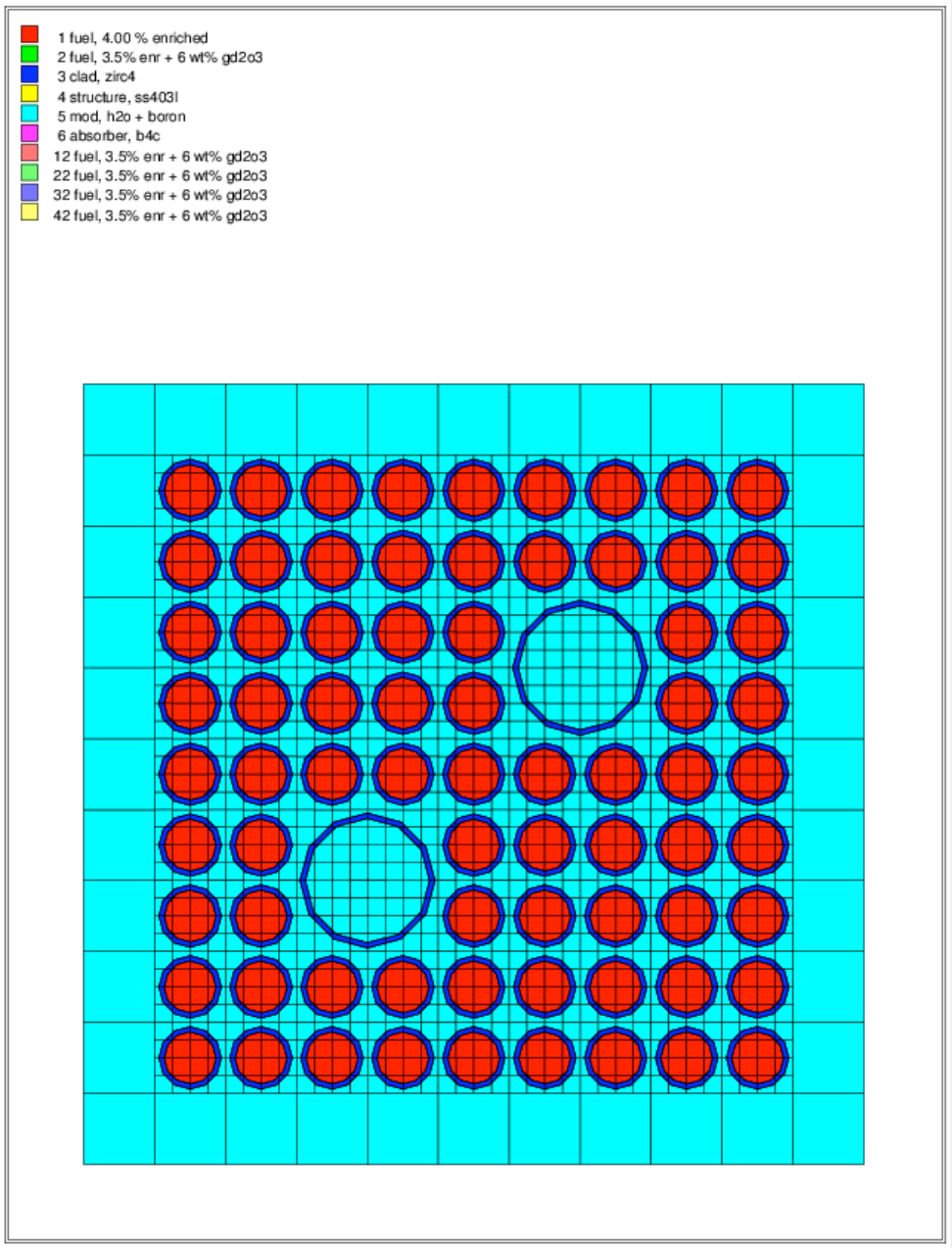


Fig. 5.25. TRITON/NEWT plot of the 9×9 lattice with water rods.

5.6.4 Basic 9×9 Lattice with Water Rods and Channel Box

BWR lattices generally contain a thin metal channel box to separate saturated liquid water from the two-phase boiling water that is in-channel. Another added complexity of BWR lattices are the corners of the channel box. The channel boxes normally have rounded corners that are difficult to model in TRITON/NEWT. For this problem, you will start with the previous model `assembly1_wr.inp` and add a channel box. You will also add an additional moderator definition to place inside the water rods and outside the channel box. The channel box will have an inner span of 12.8 cm and a thickness of 0.2 cm. The channel box rounded corners will have an inner radius of 0.8 cm. The extra moderator material will be unborated water.

Starting with `assembly1_wr.inp`, **Save As** `assembly1_wr_box.inp`. First add the new moderator material. Click **Compositions** from the left toolbar and then click **Create** from the **Standard Basic Compositions** window. Now select **Basic Composition** and input `7` for the **Mixture** number, `h2o` for the **Composition Name**, 1 for the **Density Multiplier**, 300 K for the **Temperature**, and 1.0 g/cm^3 for the **Density**. Click **OK**. Remember that you need to add material 7 to the list of NEWT materials. Click **Materials** on the left toolbar and add **MIX**=`7`, **PN**=`2`, and **COM**=`water outside channel`.

Now add the channel box. Click **Geometry** to open the **Geometry Form** and select `100 (global unit)`. You will need to add three new cuboids—one to enclose the fuel array, one to form the inside of the channel box, and one to form the outside of the channel box. It is less confusing if the surfaces are numbered from inside to outside, so these cuboids should have labels less than 10 (2, 4, and 6 are good choices). The first cuboid should be centered at (0,0) with a width of `12.6` cm (+/- `6.3` cm). The second cuboid should be centered at (0,0) with a width of `12.8` (+/- `6.4` cm). The third cuboid should be centered at (0,0) with a width of `13.2` (+/- `6.6` cm).

Once the cuboids have been added, change the array placement so that array 1 is inside cuboid 2, instead of inside cuboid 10. To do this, double click the `array` entry in the **Contents** box of the **Geometry Form** and the **Array** window will appear. Click on `cuboid 10` so that it is highlighted and then click **Remove From Media** on the right. Now click `cuboid 2` so that it is highlighted and then click **Inside** to specify that the array is inside cuboid 2 (Fig. 5.26).

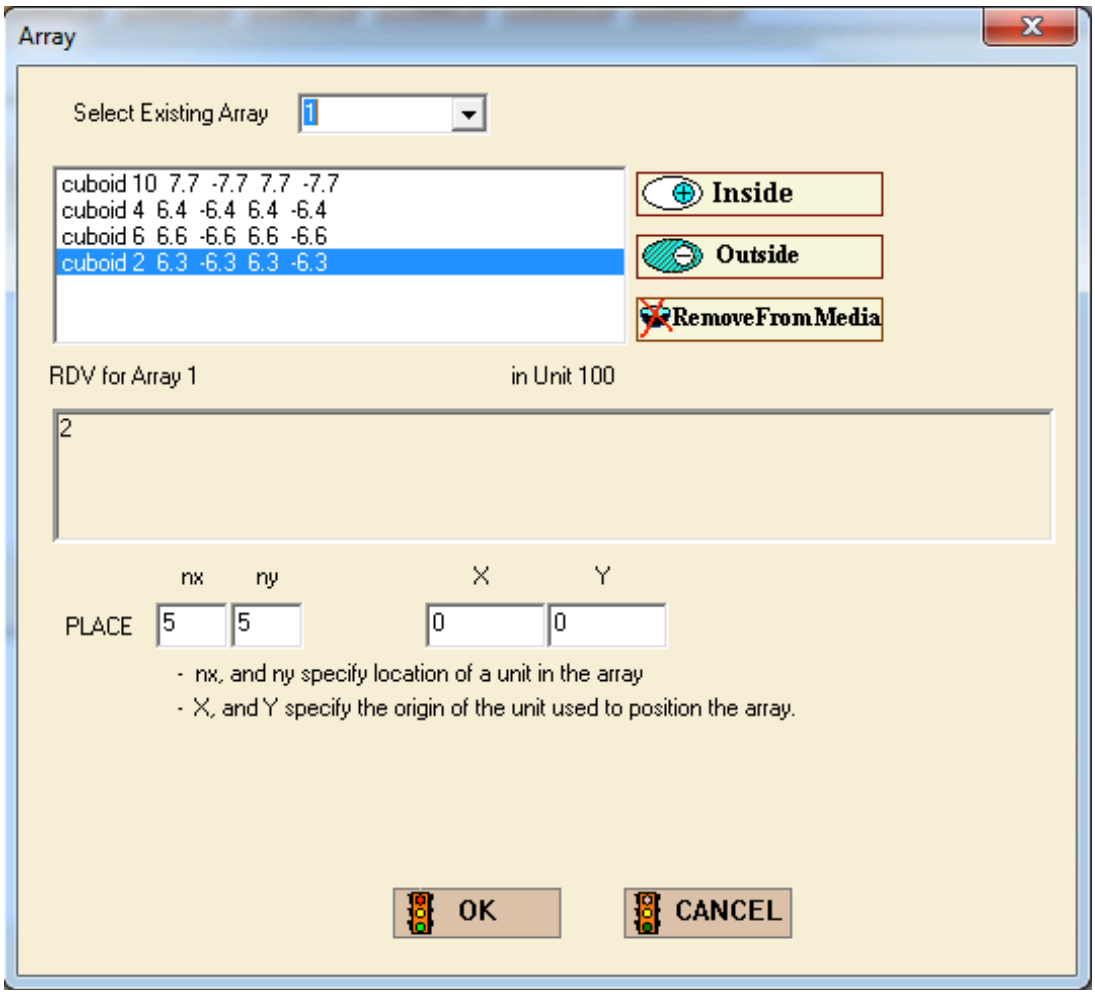


Fig. 5.26. Placement of fuel array inside cuboid 2.

Now add two new media corresponding to the new regions. The best way to do this is to just start fresh with `media` statements—it will be easier to understand this way. Double click the `media 10` statement in the **Contents** box. Highlight `cuboid 10` and click **Remove From Media** on the right. This removes the only media statement attached to the global unit.

Now add the three `media` statements that are needed. First select `5 h2o,boron` from the **Region Definition Vector for Media** dropdown; click and highlight `cuboid 2`; click **Outside**; highlight `cuboid 4` and click **Inside**. Then click **OK** (Fig. 5.27). This media fills the inside of the channel box (and outside the fuel array) with borated water.

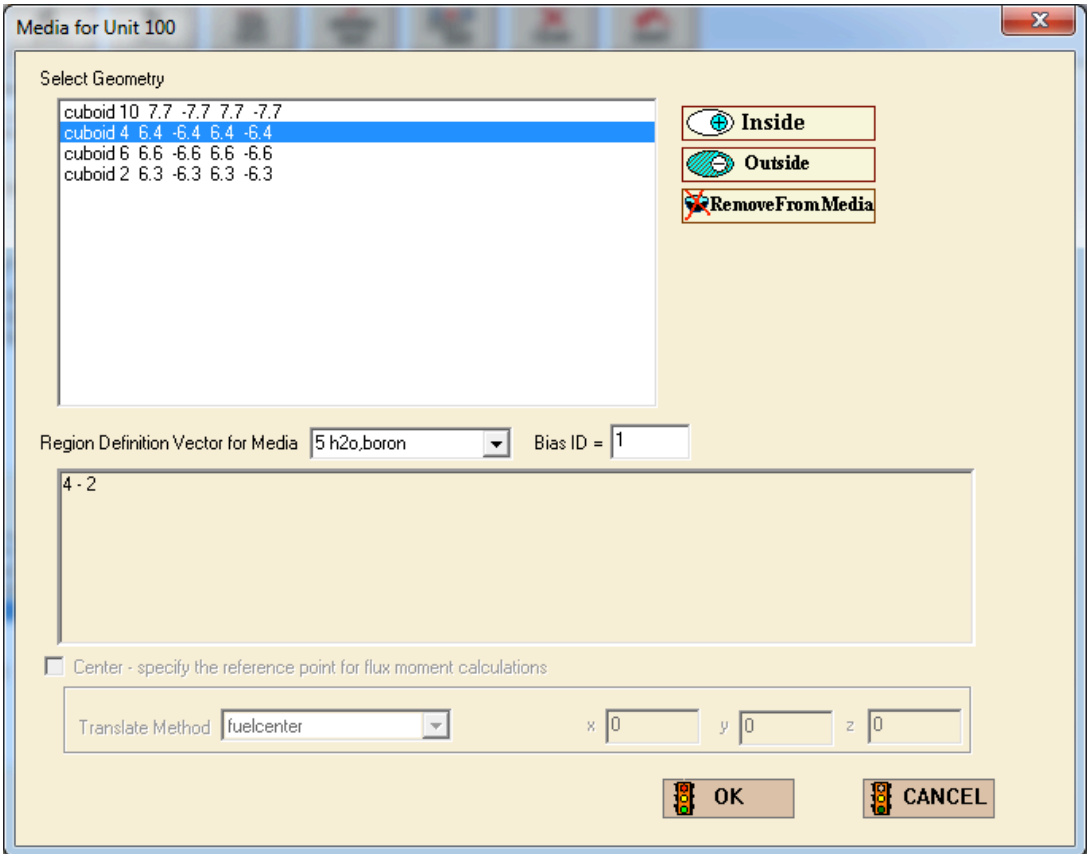


Fig. 5.27. Borated water media inside channel box and outside fuel array.

Now click **Media** again from the right toolbox, select `3 zirc4` from the **Region Definition Vector for Media** dropdown; highlight `cuboid 4` and click **Outside**; and then highlight `cuboid 6` and click **Inside** (Fig. 5.28). Click **OK**. This media fills the channel box shell with Zircaloy-4.

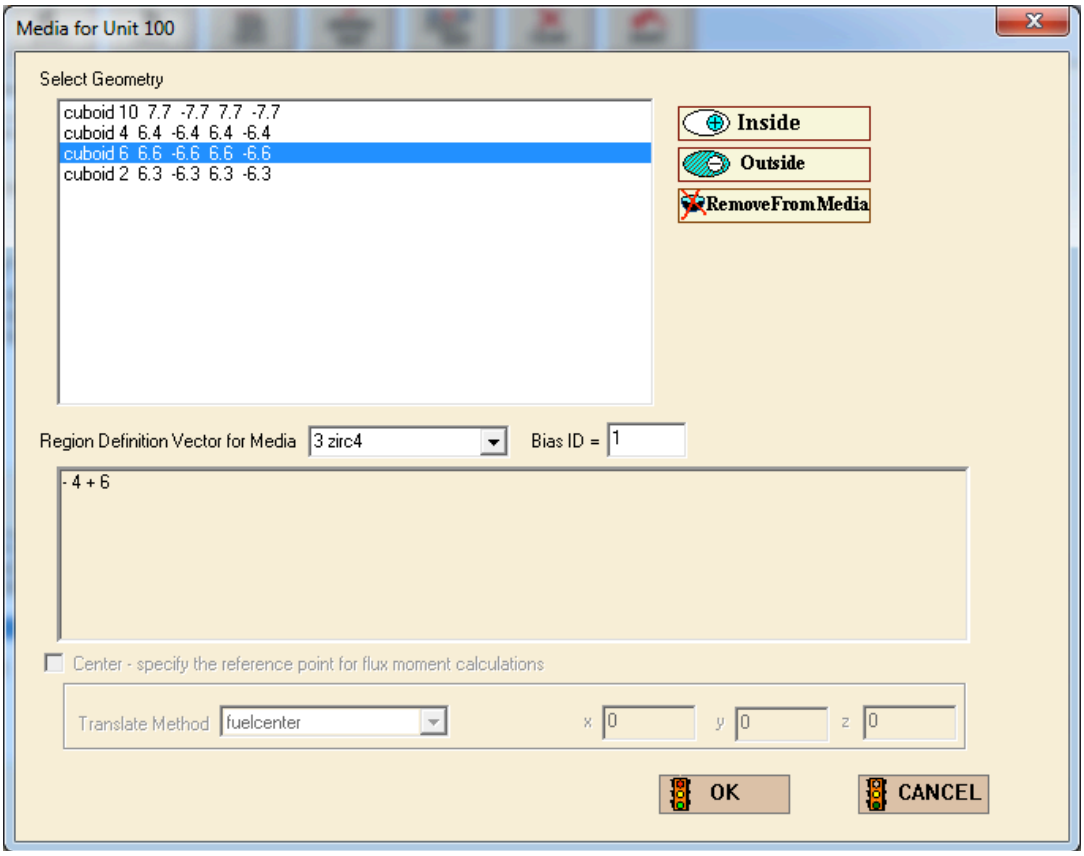


Fig. 5.28. Zircaloy-4 media for the channel box shell.

Lastly, click **Media** from the right toolbox; select `7,h2o` from the **Region Definition Vector for Media** dropdown; highlight `cuboid 6` and click **Outside**; and then highlight `cuboid 10` and click **Inside** (Fig. 5.29). Click **OK** when you are finished. This media is the unborated water outside the channel box.

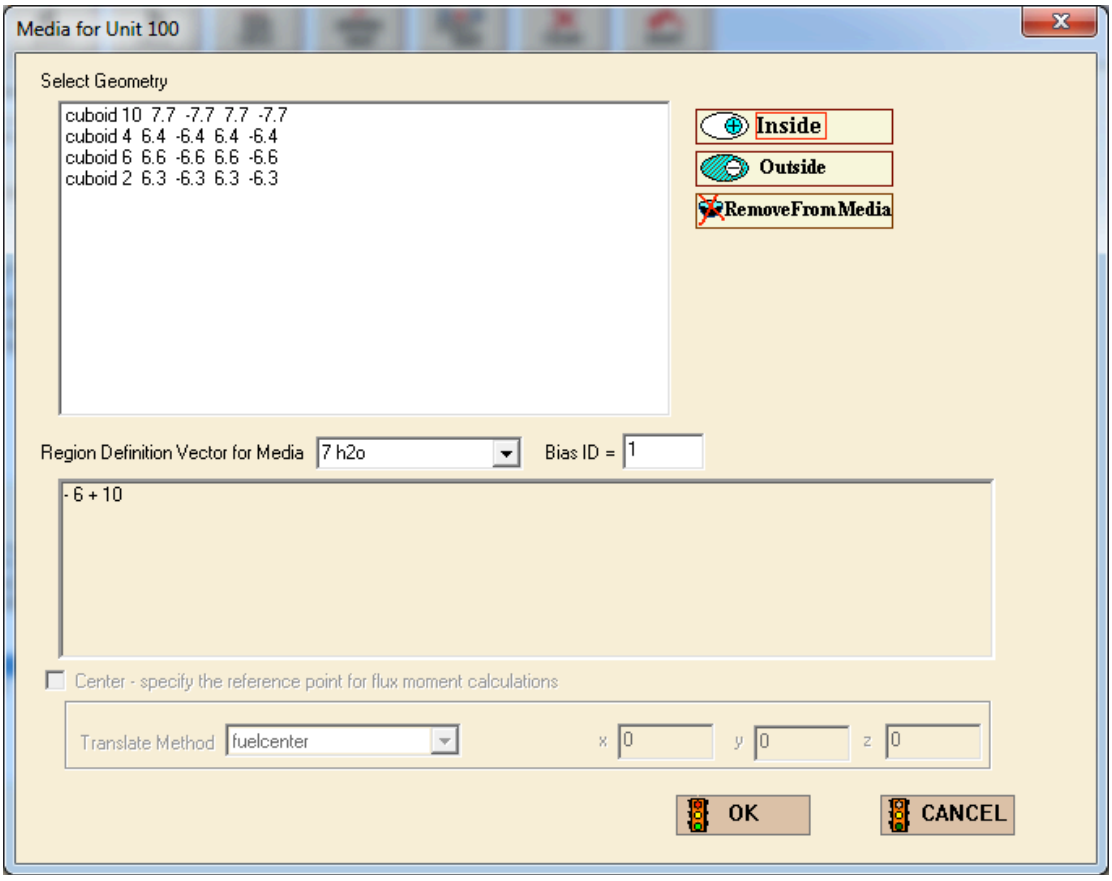


Fig. 5.29. Unborated water media window for outside channel box.

Note that there is no media specification that is inside cuboid 2. This is because array 1 fills cuboid 2 entirely—a media statement would be redundant in this case. The channel box is now complete with the exception of the rounded channel corners. The **Geometry Form** for the global unit should look like Fig. 5.30.

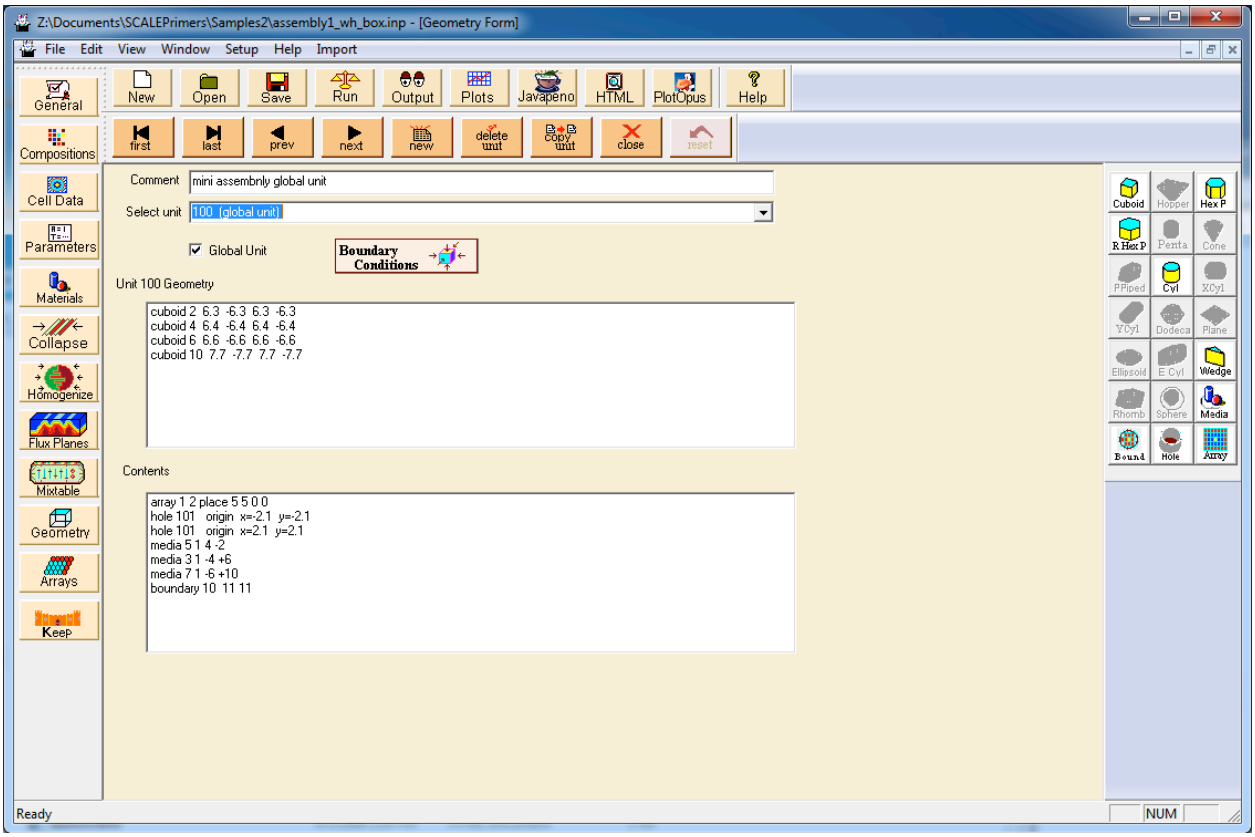


Fig. 5.30. Geometry form for 9×9 lattice with water rods and a channel box.

When changing the array, you might notice in the **Contents** box that the `array` statement is below the `hole` statements. Generally, you always want the holes to come after the arrays in a unit. TRITON/NEWT processes holes and arrays by the order in which they are read in each unit. If a `hole` statement comes before an `array` statement, the hole will be covered by the contents of the array. If the `array` statement does not appear first as shown in Fig. 5.30, you should reorder them by deleting the `hole` statements and adding them back to make them appear after the `array` statement. Alternatively, you can open the ASCII text input file and change the order in the global unit using a text editor. To open the text input file, go to **Edit**>**Edit File** on the top menu bar in GeeWiz. Once you save the changes in your text editor, close and re-open the file in GeeWiz.

Now plot the geometry without the rounded channel corners to make sure that there are no mistakes. Turn **Parm=Check** on and then **Run** the model. You should open `assembly1_wr_box.newtmatl.ps` to view the geometry (Fig. 5.31). Observation of the geometry shows that you have not placed the new moderator material, `7 h2o`, in the water rods. Small errors such as these are why using **Parm=Check** is recommended to check the model and view the TRITON/NEWT plot to find errors or inconsistencies. To change the water mixture in the water rods unit `101`, change the media to `7 h2o`. If you were to plot the geometry again, the inside of the water rods would be colored pink (same as the outside channel water).

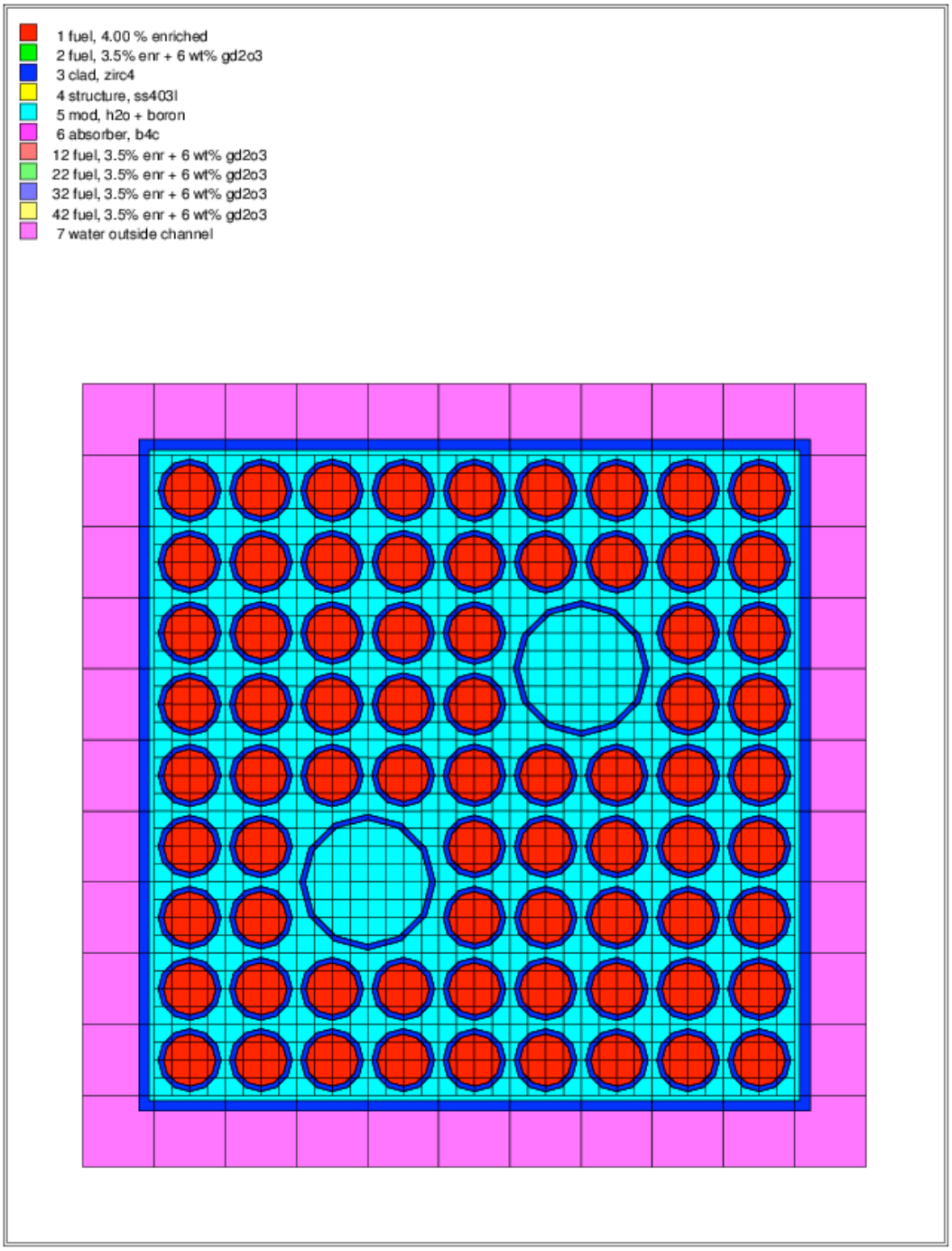


Fig. 5.31. TRITON/NEWT plot of 9×9 lattice with water rods and a channel box.

Now add the channel corners. To add the rounded channel corners, you will need to construct one additional unit. This unit will contain ¼ of a fuel pin (pin and clad), two cylinders that define the inner and outer radii of the channel corner, and a cuboid that bounds the unit. The cuboid is needed to cover the square channel corner that would not otherwise be covered by the cylinders. A visualization of the unit that is being constructed can be found in Fig. 5.32.

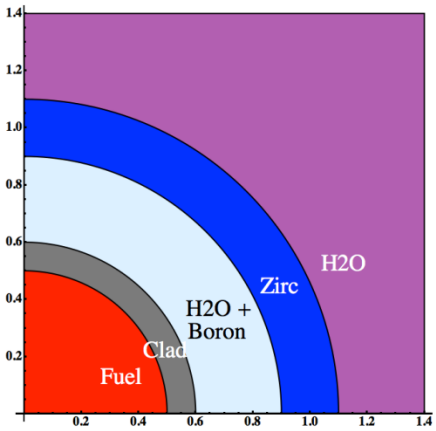


Fig. 5.32. Channel corner drawing of the 9×9 lattice.

For this unit, use four cylinders that are chorded at +X= and +Y=0.0 to keep the upper right quadrant of the cylinders. From the **Geometry Form**, click **New** to construct the new channel corner unit `201`. Specify four cylinders with chords at **+X**=`0` and **+Y**=`0` and radii of `0.5`, `0.6`, `0.8`, and `1.0` cm. The cuboid should span from `0` to `1.4` cm in the *x* and *y* directions. Once the cylinders are defined, add the five `media` statements. At this point, you should be familiar with how to enter media statements, so the steps will not be outlined in detail here. Also specify the bounding unit and a `4×4` grid. The final **Geometry Form** for this unit can be found in Fig. 5.33.

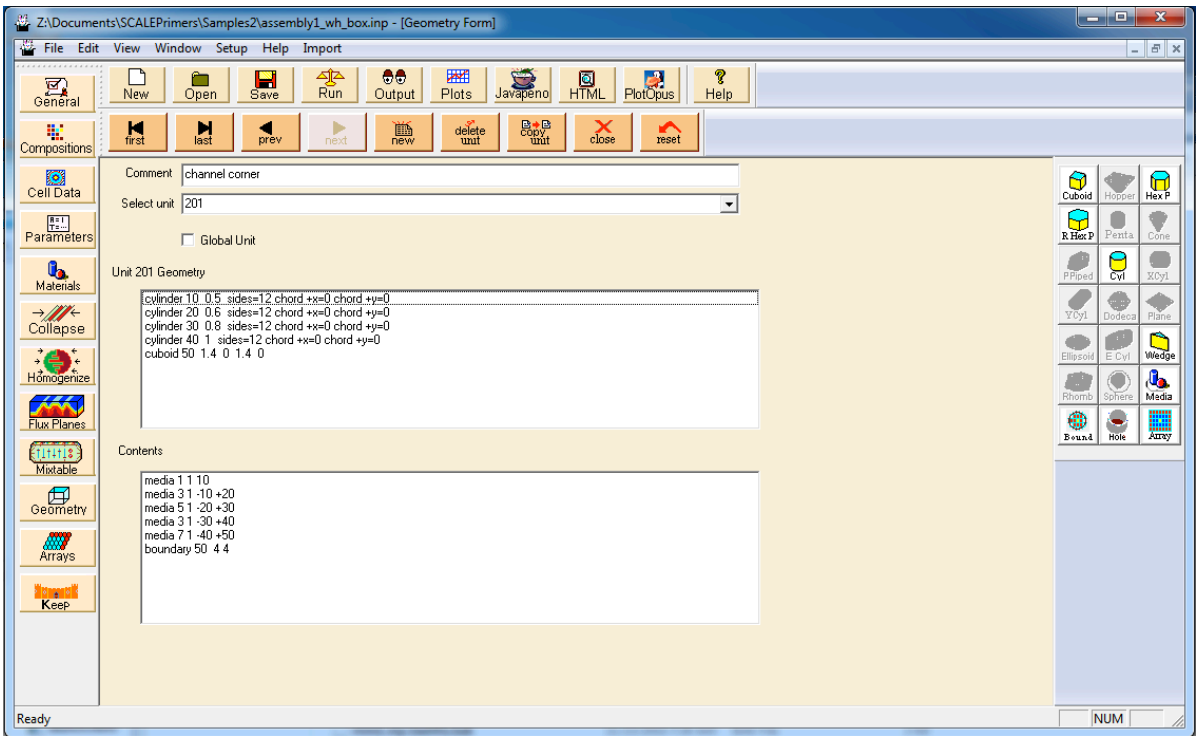


Fig. 5.33. Geometry form for the channel corner.

Now select `100 (global unit)` and add four holes containing the new unit `201`. The first hole should be placed at the (X,Y) location of the center of the most northeastern fuel pin, (`5.6`, `5.6`). The hole will not have to be rotated because it is already in the correct orientation for the northeastern corner (Fig. 5.34).

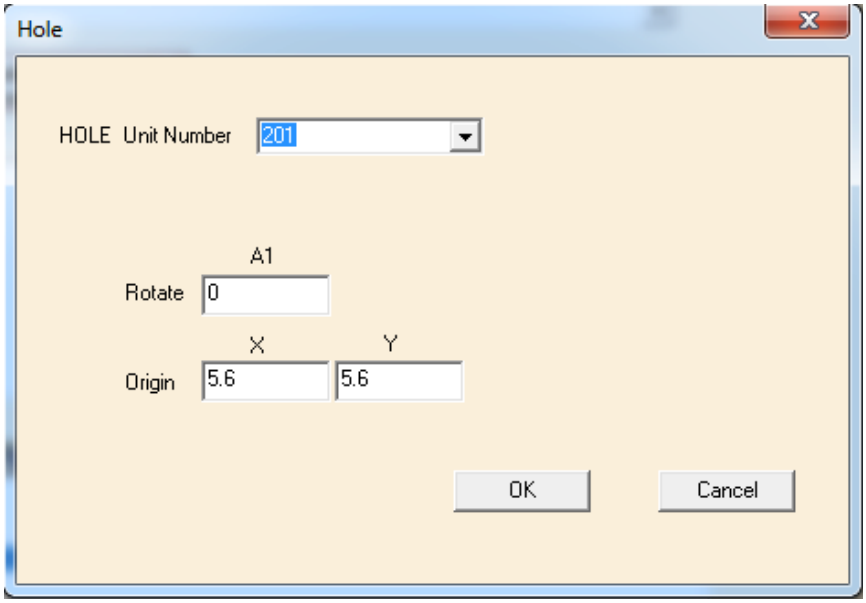


Fig. 5.34. Hole form for northeast channel corner of the 9×9 lattice.

Add three other holes at (`-5.6,5.6`), (`-5.6,-5.6`), and (`5.6,-5.6`) with rotations of `90`, `180`, and `270` degrees, respectively. The final global unit **Geometry Form** can be seen in Fig. 5.35.

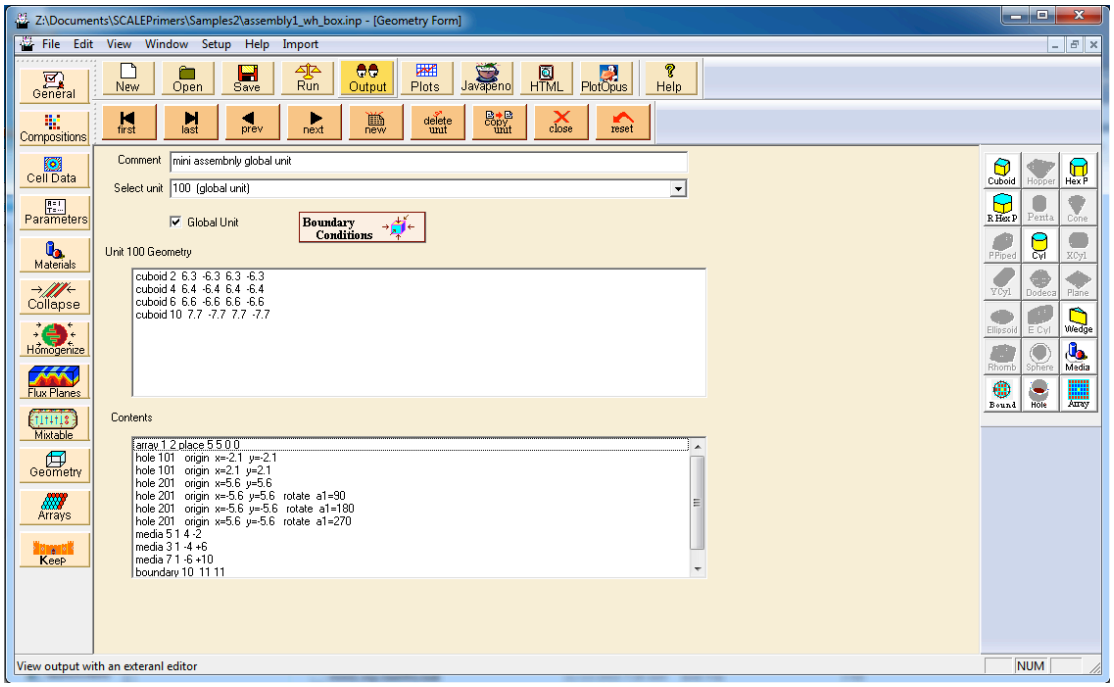


Fig. 5.35. Final geometry form for 9×9 lattice with water rods and channel box with rounded corners.

Now **Run** the model to generate a plot of the geometry (Fig. 5.36). If the channel corners look correct, turn off **Parm=Check** and **Run** the model again. This sample problem took 191.75 seconds of CPU time, with 110.26 seconds used by the NEWT transport solution. The calculated k_{inf} is 1.29441281. You will notice that the CPU time has been steadily increasing as with added complexity, and thus, more grid cells. Also, you might notice that the calculated k_{inf} increased for this assembly as compared to previous assemblies. A significant amount of unborated water was added in the water rod and bypass regions, which decreased parasitic absorption compared to the cases that had borated water in the water rods and bypass.

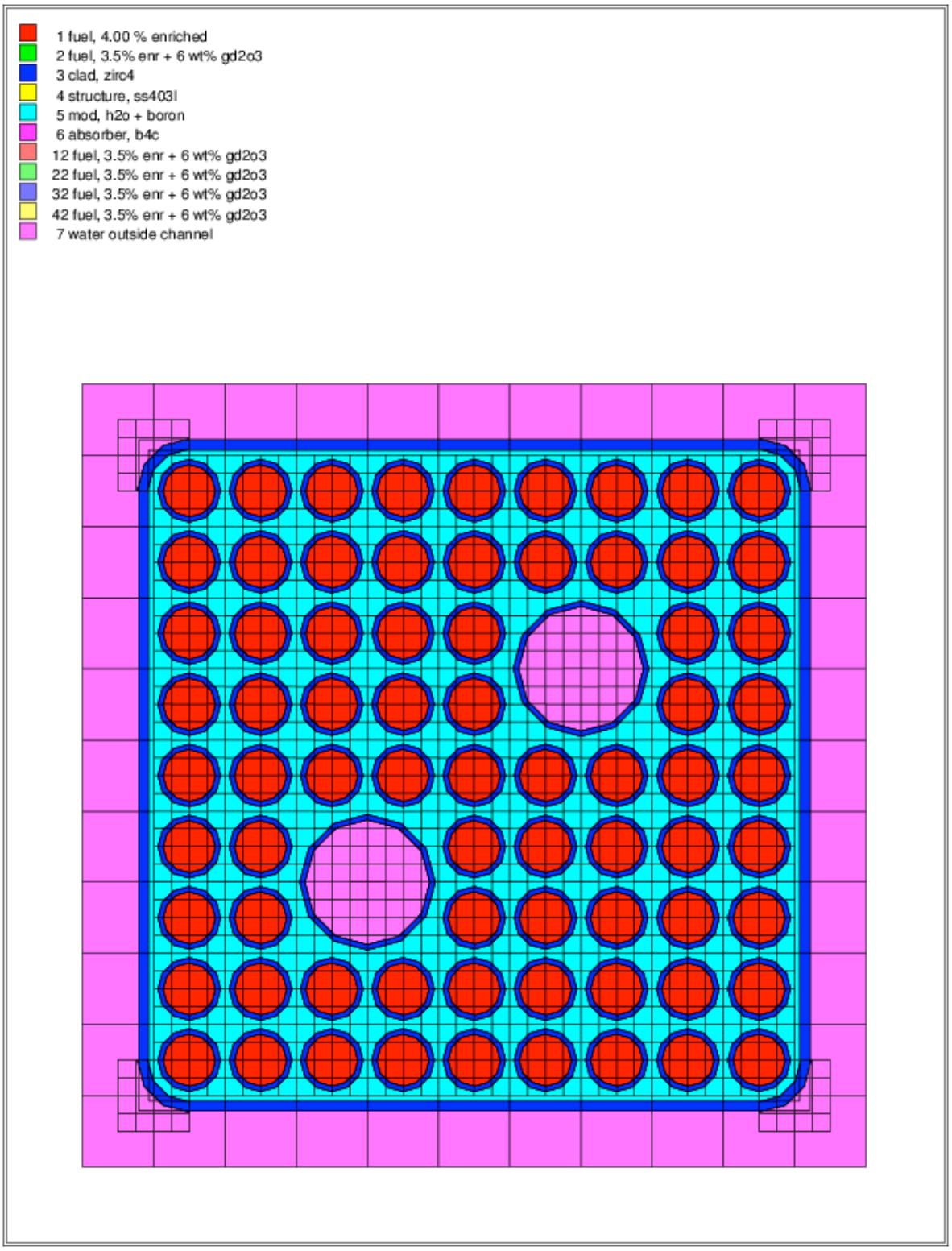


Fig. 5.36. TRITON/NEWT plot of the 9×9 lattice with water rods and a channel box with rounded corners.

5.6.5 Basic 9×9 Lattice with Water Rods, Channel Box, and Control Blades

Unlike PWRs that have control rods and soluble boron, the main reactivity control and shutdown mechanism in BWRs is control blades. These control blades are external to the fuel bundle and typically have somewhat complex geometry features that can make modeling difficult. Typical control blades have a cross shape or "cruciform" and are positioned in the gap between four fuel bundles. The four arms of the control blade are typically referred to as "wings". Modeling a single fuel bundle with a control blade requires modeling one-fourth of the control blade, as identified by the shaded region in Fig. 5.37.

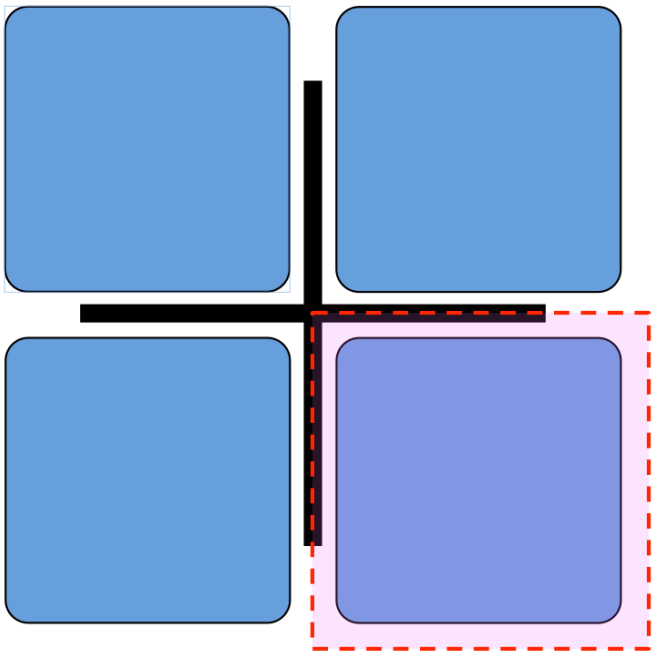


Fig. 5.37. Fuel bundle and control blade layout in a typical BWR.

In this sample problem, control blades will be added to the BWR assembly model. Each wing of the control blade has 21 absorber pins with a clad inner radius of 0.18 cm and a clad outer radius of 0.24 cm. The control blade has a sheath thickness of 0.16 cm and a tip-to-tip span of 24.88 cm. For this control blade, the central support and the wing have the same thickness, but this is not necessarily the case for all control blades. Typical BWR control blades have holes drilled in them so that coolant (water) fills the space between the absorber pins and the blade sheath. The blade structure is stainless steel 304L (SS304L) and the blade absorber material is B_4C at 70% theoretical density. A SCALE/NEWT representation of the southern half of the east control blade wing can be found in Fig. 5.38.

Note that in this example, the absorber pins are mounted vertically, as is the case for many reactors. However, there are some designs where absorber pins are mounted horizontally. Modeling a control blade with horizontal absorber pins is more difficult, but not impossible. In a 2D model, the absorber pins must be represented as vertical cylinders, but their geometry will need to be "adjusted" to give the correct absorber/fuel ratio. When modeling horizontal absorber pins as vertical pins, ensure that the overall absorber pin volume of the blade is correct.

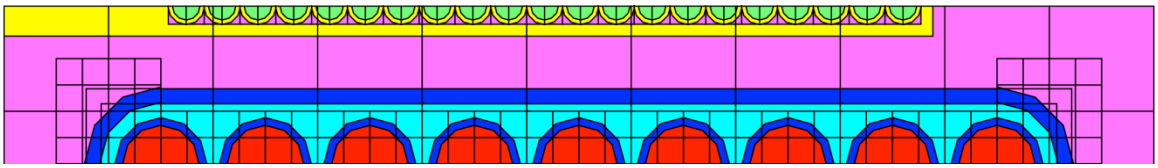


Fig. 5.38. Southern half of an east control blade wing.

In general, it is recommended that control blades be built in SCALE/NEWT as separate units and be placed in the global geometry using holes. By constructing the control blades in this manner, it is easy to remove or insert control blades by commenting or uncommenting the control blade holes in the global unit, and it is easy to reuse the same control blade design in a different bundle model by simply copying and pasting the control blade units into the new model.

For this problem, **Open** the BWR sample problem with water rods and a channel box (assembly1_wr_box.inp) and **Save As** assembly1_wr_box_cr.inp. No new mixtures are needed for this problem; use the SS304L and B_4C compositions that were previously defined. First, build the absorber tube units for the southern half of the east wing of the control blade by constructing two half-cylinders for the clad inner and outer radii, and a cuboid whose sides are tangent with the clad outer boundary. Go to the **Geometry Form** and click **New** to create an additional unit; give the unit a descriptive comment such as "east control wing half tube" and use 300 as the unit number. Then, add two cylinders for the clad inner radius (0.18 cm) and outer radius (0.24 cm) with a chord –y=0 to retain the bottom half of the cylinders. Now add a cuboid with boundaries of ±x=0.24 cm, +y=0 cm, and –y=0.24 cm. Fill the inner cylinder with B_4C (media 6), between the inner and outer radii with SS304L (media 4), and outside the outer clad radius and inside the cuboid with out-channel moderator (media 7). The final Geometry Form for the absorber tube unit can be found in Fig. 5.39.

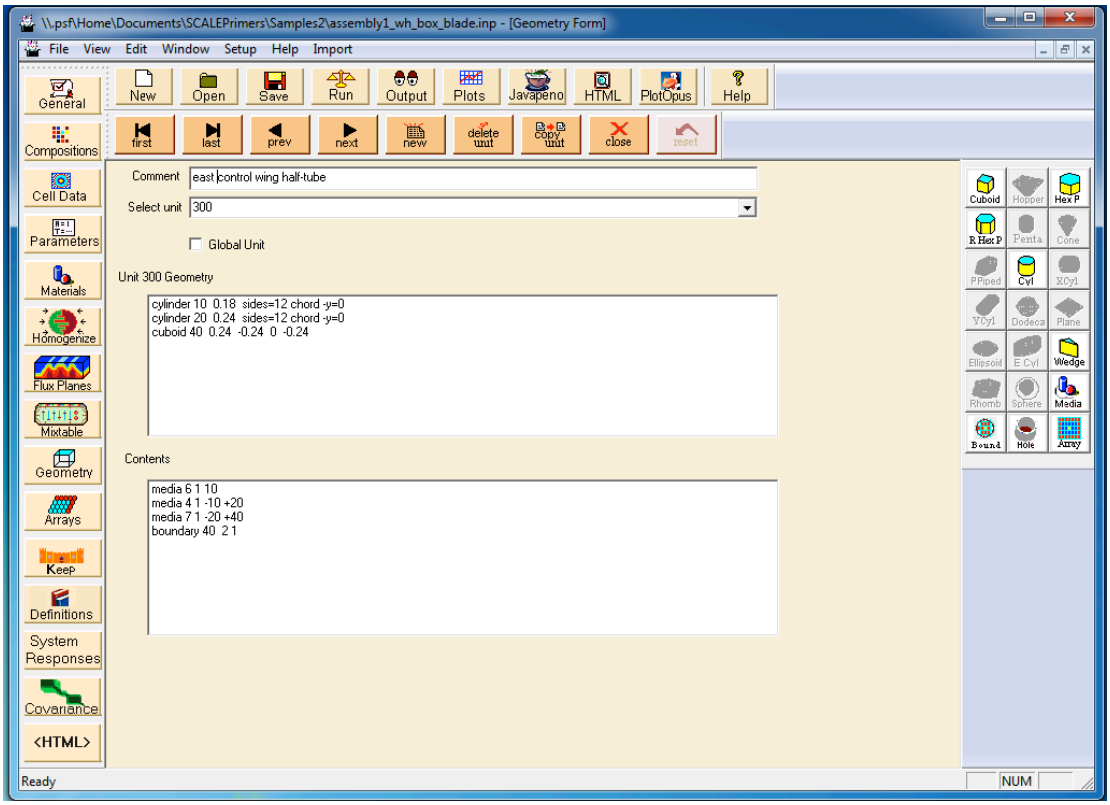


Fig. 5.39. Control blade eastern wing absorber tube unit.

The next step will add an array of the absorber tube units. Currently, the cuboid enclosing the absorber tubes has the same dimensions as the absorber tubes, essentially assuming no gap between absorber tubes and adjacent absorber tubes or the blade sheath. To construct the array of absorber tubes, first click **Arrays** to open the array geometry form, then click **Create** to create a new array. In the **Array Properties** window, enter 200 for the array **Number**, Square for the array **Type**, 21 for **NUX**, 1 for **NUY**, and initialize the array locations with unit 300; then click **OK**. The final **Array** form and **Array Properties** windows can be found in Fig. 5.40.

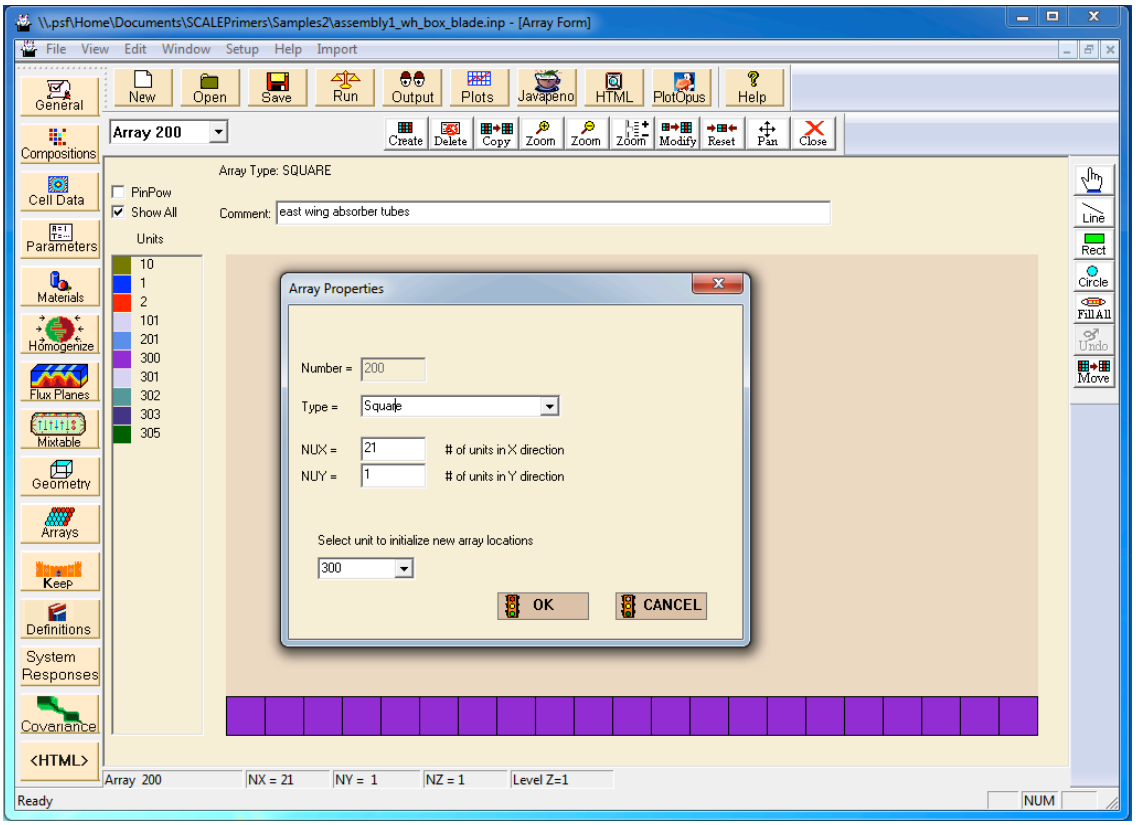


Fig. 5.40. East control blade array form.

To complete the southern half of the east wing of the control blade, the array should be placed in another unit that is the half-width of the blade wing in the y-direction and the half-span of the blade in the x-direction. Click **Geometry** to open the **Geometry Form**, then click **New** to create a new unit for the control blade. Name the unit "control blade east wing" and use a unit number of 302. Then add a **Cuboid** that is the half-span of the blade (12.44 cm) and the half thickness of the blade (0.4 cm). The media inside the cuboid should be SS304L (media 4). Then place the center of the (1, 1) unit of array 200 (the absorber tube array) at an x-position of 2.44 cm and a y-position of 0.0 cm. The array placement x-position of 2.44 cm corresponds to blade central support length (2.2 cm) plus the absorber tube outer radius (0.24 cm). The final Geometry Form for unit 302 is shown in Fig. 5.41.

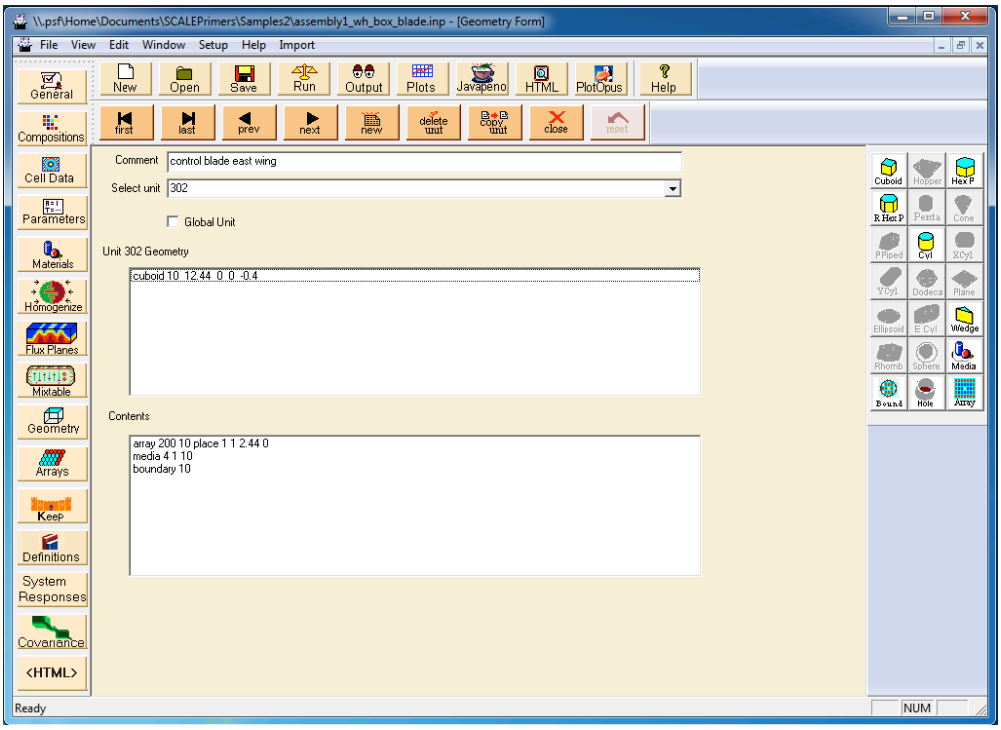


Fig. 5.41. Final geometry form for the southern half of the east control blade wing.

Now to place the control blade wing into the global geometry, go to the **Geometry Form** for the global unit. Insert a **Hole** that is unit 302 into the global geometry at x=-7.7 and y=7.7. The final Geometry Form for the global unit and the Hole dialog box can be found in Fig. 5.42.

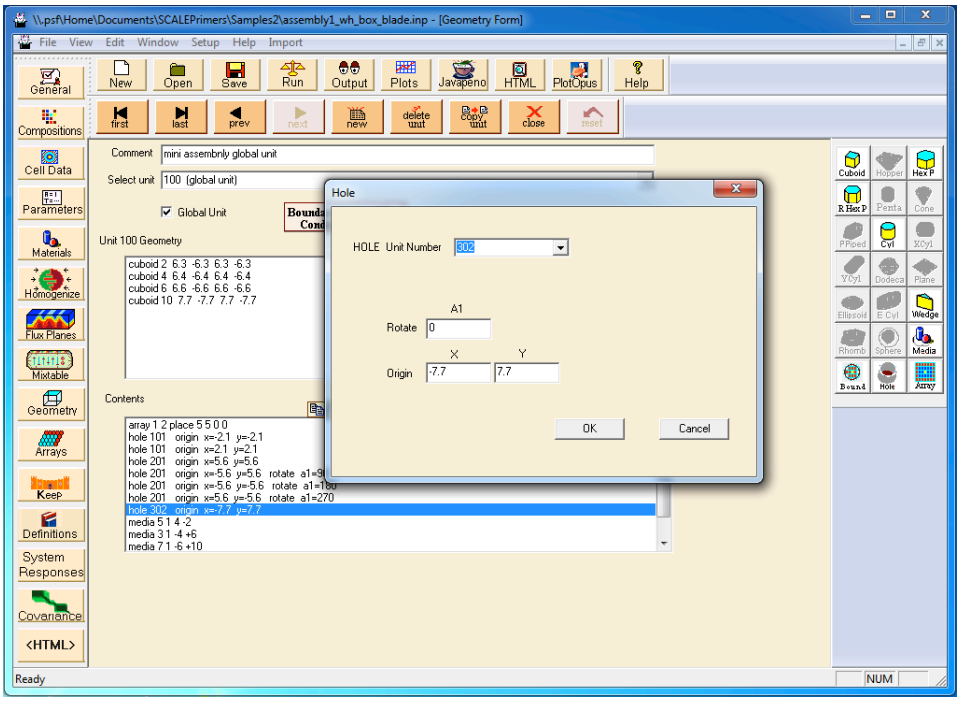


Fig. 5.42. Final geometry form and hole dialog box for the global unit and the southern half of the eastern control blade wing.

Although the southern wing of the control blade still needs to be added before the model is complete, it is a good idea to **Run** the model using **Parm=Check** to plot the model and confirm that the control blade wing has been properly constructed and positioned in the global unit. A SCALE/NEWT plot of the current model can be found in Fig. 5.43.

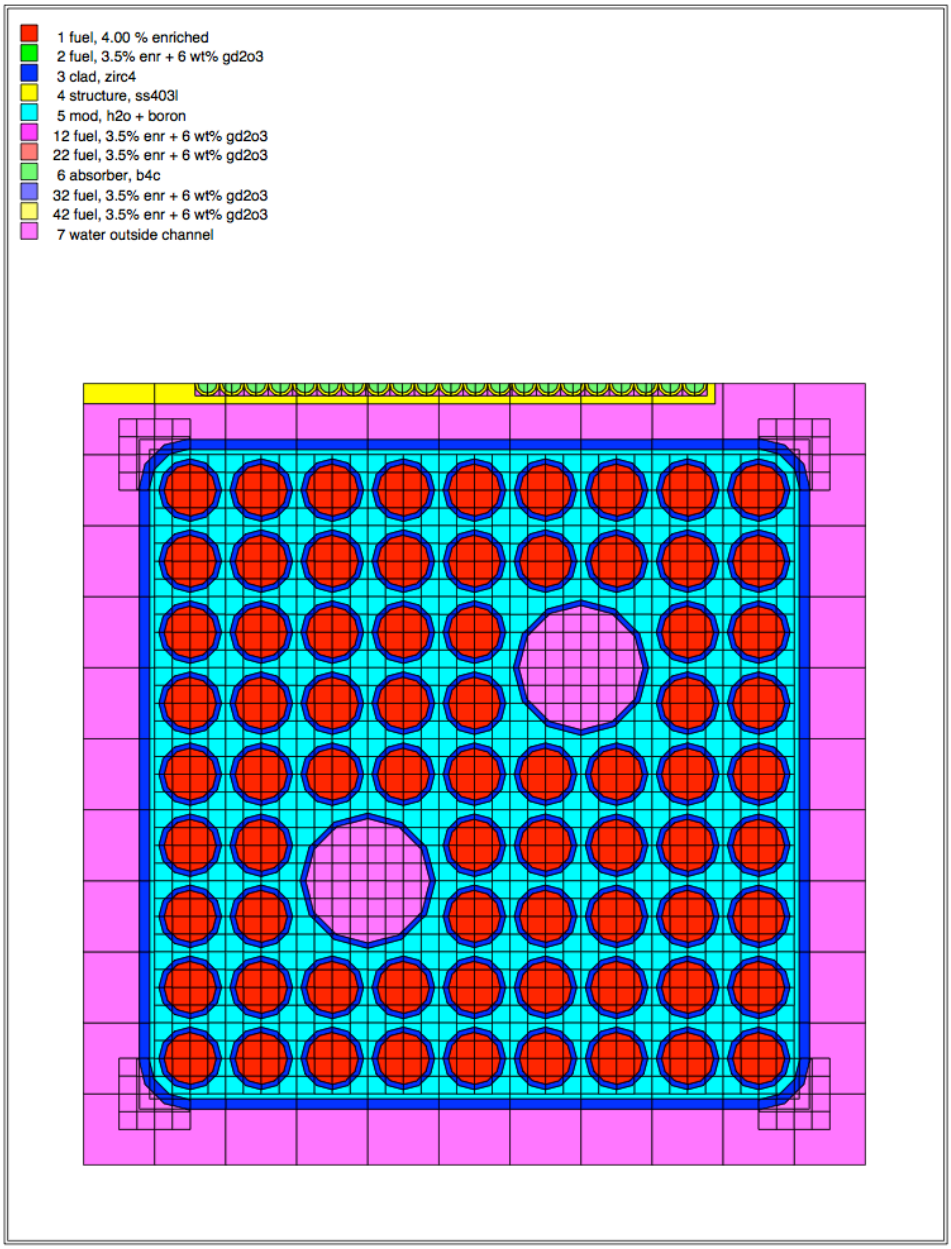


Fig. 5.43. SCALE/NEWT plot of the assembly and the southern half of the eastern control blade wing.

Using a similar strategy, add the eastern half of the southern control blade wing to the model. Use the identical dimensions as used for the eastern wing, construct the absorber tube unit, but use chords to retain the right side of the fuel pin. The final **Geometry Form** for the eastern half of the southern wing can be found in Fig. 5.44.

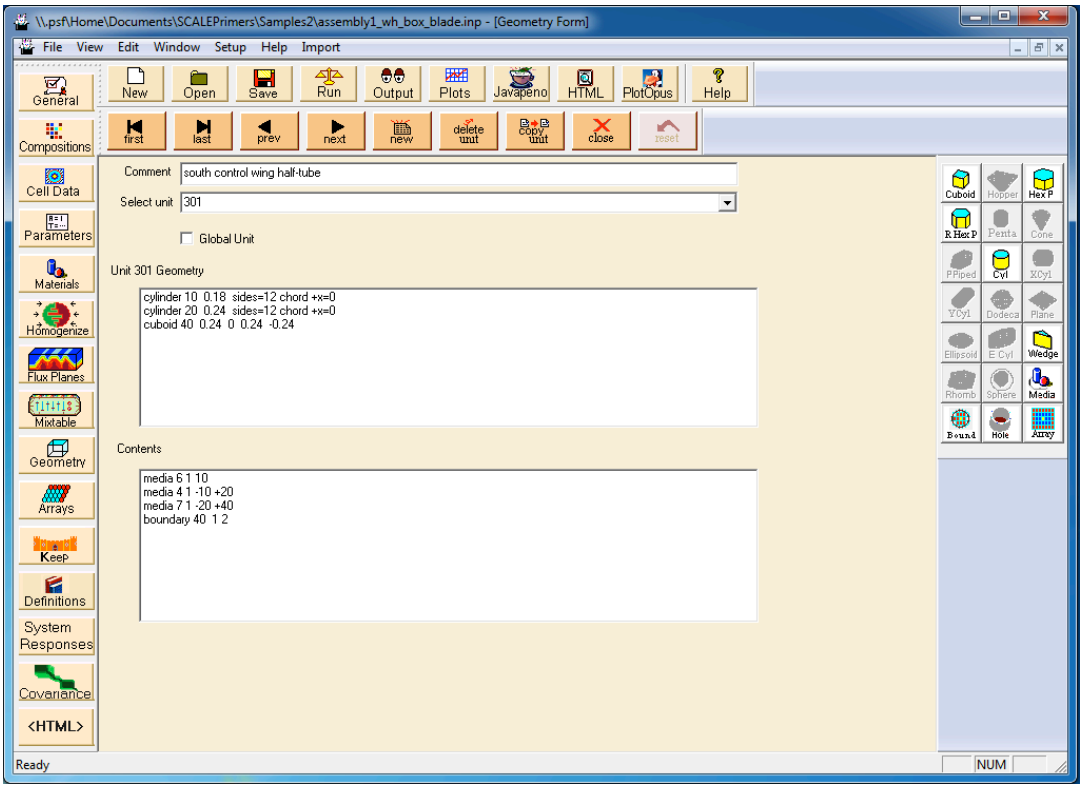


Fig. 5.44. Control blade southern wing absorber tube unit.

The main difference in the way the southern wing is constructed in comparison to the eastern wing is the array placement and the size of the wing unit. Overlapping holes are not a recommended practice, so the size of the southern wing central support should be smaller by the blade half-thickness so that the eastern wing and the southern wing do not overlap in the northwest corner of the model. This length of the southern wing should be 12.04 cm: the half-span (12.44 cm) minus the wing half-thickness (0.4 cm). The array placement is also slightly different in comparison to the eastern wing; now array position (1, 21) should be placed at x=0.0 and y=-2.04 cm (the central support length, 2.2 cm, minus the blade half thickness, 0.4 cm plus the absorber tube outer radius, 0.24). The final **Array Form** and **Geometry Form** for the eastern half of the southern control blade wing can be found in Figs. 5.45 and 5.46.

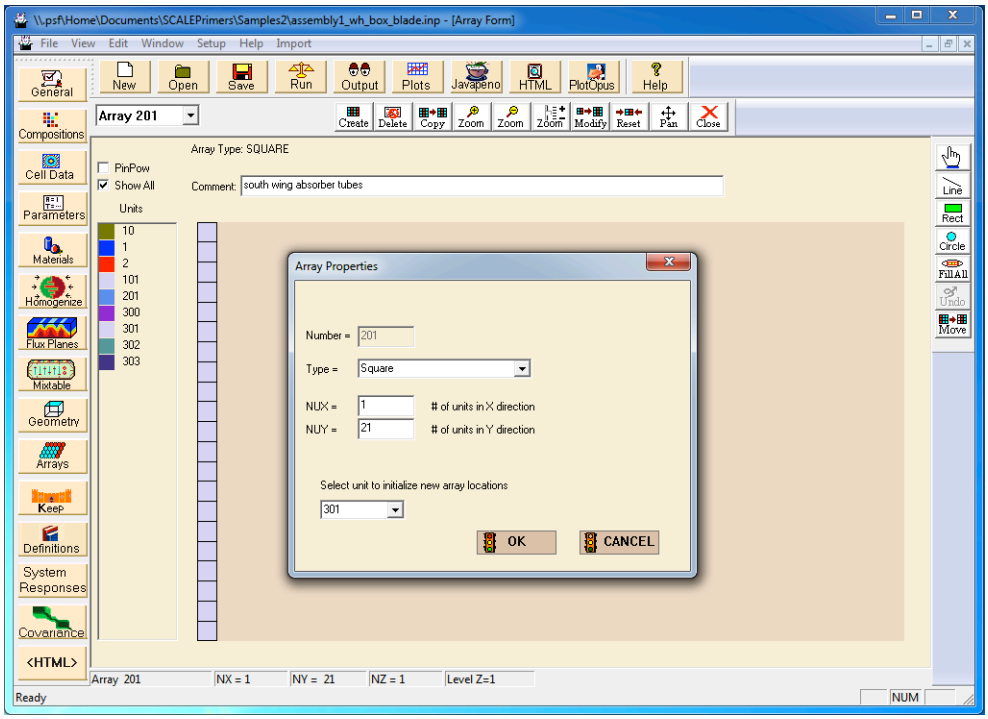


Fig. 5.45. South control blade array form.

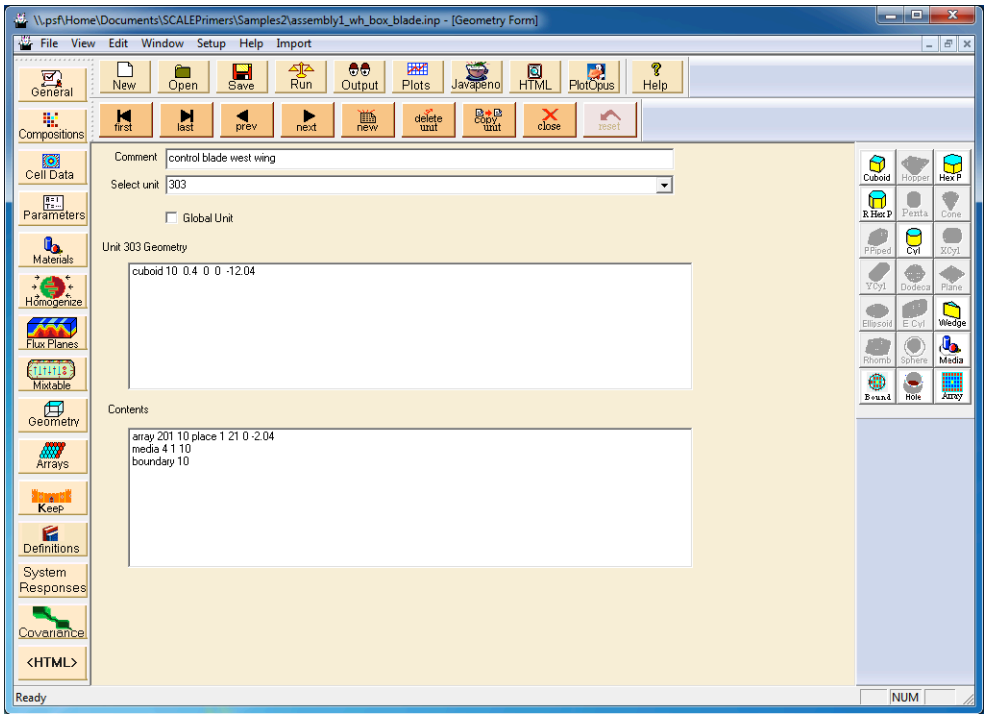


Fig. 5.46. Final geometry form for the eastern half of the south control blade wing.

In the global unit, the hole placement of the south wing is also slightly different from the eastern wing; the y-position has been shifted down one half-thickness of the control blade. The final Geometry Form for the global unit can be found in Fig. 5.47.

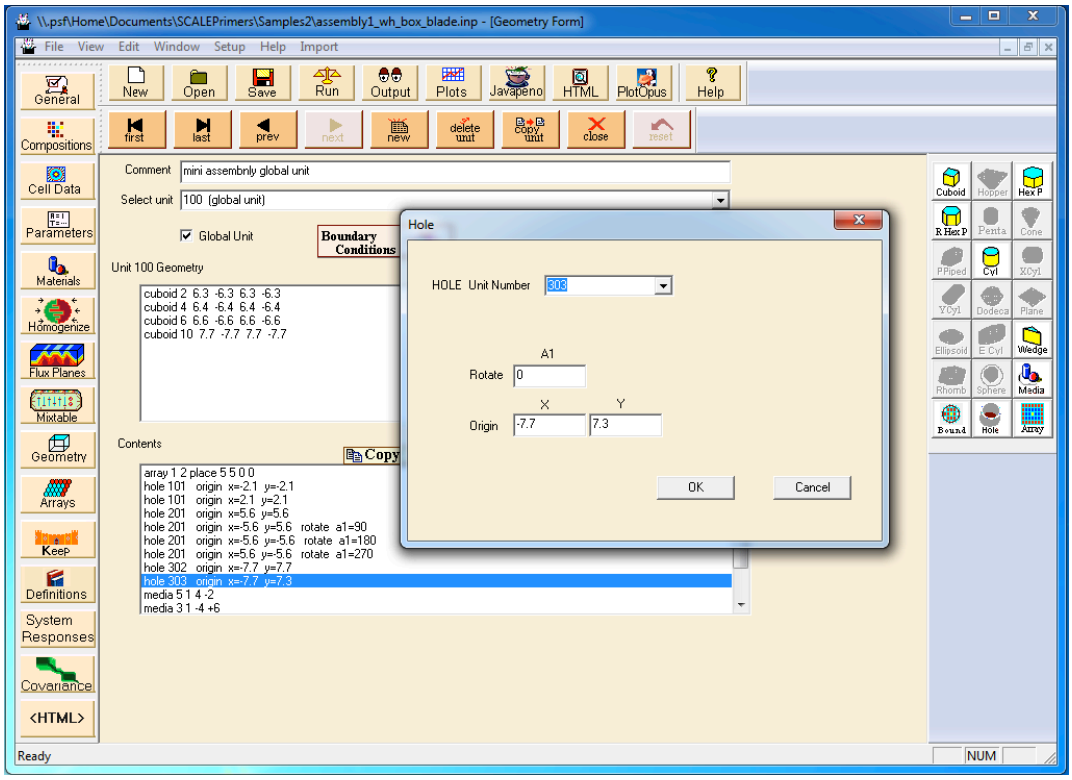


Fig. 5.47. Final geometry form and hole dialog box for the global unit for control blade placement.

Again, **Run** the model using **Parm=Check** to plot the model and confirm that the control blade wing has been properly constructed and positioned in the global unit. A SCALE/NEWT plot of the final fuel bundle with both the east and south wings of the control blade can be found in Fig. 5.48. If the model looks as it should, turn off **Parm=Check** and **Run** the model again. The SCALE/NEWT calculated k_{inf} is 1.07255008 for this problem. Note that the high multiplication factor is because this model is not a true BWR fuel bundle model and is intended for illustration purposes only; typical BWR fuel bundles have lower enrichment, contain fuel rods with integrated burnable absorbers, and operate at a lower coolant density.

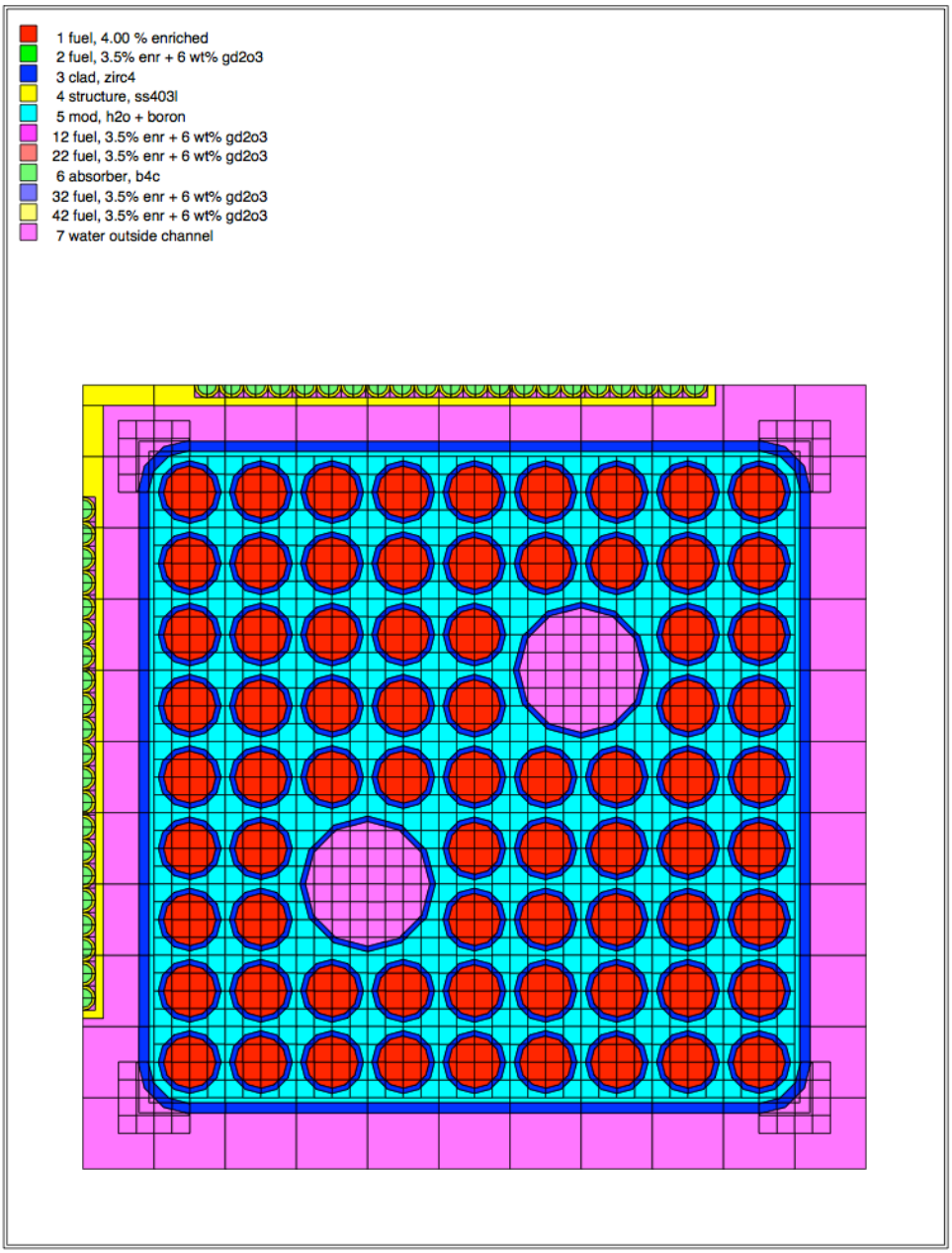


Fig. 5.48. SCALE/NEWT plot of the assembly and the southern and eastern control blade wings.

5.7 SUMMARY

This section has helped you to accomplish the following:

- use GeeWiz and TRITON/NEWT to describe more advanced geometry models using geometry modification data and holes
- use the CHORD modifier to truncate a body with a plane perpendicular to a major axis
- use the ORIGIN modifier to translate the location of a body (i.e., shape)

- use the ROTATE modifier to rotate a body to any angle
- use HOLE to place one unit inside another unit
- use the geometry modification data to model common features of PWR and BWR lattices

Next simple depletion calculations for pin cells and lattices will be introduced.

6. DEPLETION

In the previous section, you learned some advanced geometry features that are used to set up more complicated lattice models for PWRs and BWRs. In this section, you will learn to add time-dependent depletion to these models.

6.1 WHAT YOU WILL BE ABLE TO DO

- use GeeWiz and SCALE to perform standard depletion calculations for pin cells and lattices
- use the Parm=Weight option to generate problem-dependent 49-group cross-section libraries
- understand the advanced concepts required for detailed depletion of lattices
- use the Assign function to assign material cross sections to different materials

6.2 DEPLETION BASICS FOR LWR LATTICE PHYSICS

SCALE/TRITON has three different depletion sequences: T-DEPL, T5-DEPL, and T6-DEPL. The T5-DEPL and T6-DEPL sequences use the Monte Carlo codes KENO-V.a and KENO-VI, respectively, for the transport calculations. The material in this primer only covers how to use the T-DEPL sequence, which uses NEWT for the transport solution. Although this primer focuses only on the T-DEPL sequence, the depletion input formats for all three sequences are identical.

A good explanation of the T-DEPL sequence can be found in the SCALE manual[1]:

> *The T-DEPL sequence builds upon the cross-section processing and neutron transport solution of the T-NEWT sequence. The transport calculation with NEWT is followed by COUPLE (Sect. F6 of the SCALE Manual) and ORIGEN-S (Sect. F7 of the SCALE Manual) calculations. In a T-DEPL calculation, NEWT is used to create a three-group weighted library based on calculated volume-averaged fluxes for each mixture. COUPLE is used to update the ORIGEN-S cross-section library with cross-section data read from the weighted library. Three-group fluxes calculated by NEWT are supplied to ORIGEN-S for depletion calculations. COUPLE/ORIGEN calculations are repeated for each mixture being depleted, as specified in input, using mixture-specific cross-section data and fluxes.*
>
> *Although ORIGEN-S supports the use of time-dependent fluxes, T-DEPL uses a more rigorous iterative procedure. Because spatial fluxes are burnup dependent, changing with nuclide inventories, and because mixture cross sections will also change with burnup, the T-DEPL sequence uses a predictor-corrector approach to update both fluxes and cross sections as a function of burnup. T-DEPL calculations can be considered to consist of two components during this iterative phase: (1) transport calculations (cross-section processing and the neutron transport solution) and (2) depletion calculations (COUPLE and ORIGEN-S). Transport calculations are used to calculate fluxes and prepare weighted cross sections and other lattice physics parameters based on a given set of nuclide concentrations; depletion calculations are used to update nuclide concentrations, which can be used in the following transport calculation.*

So the SCALE/TRITON depletion calculation uses a transport calculation to calculate the spatial fluxes over the lattice. These space-dependent fluxes are then passed to COUPLE and ORIGEN-S to deplete the

mixtures. The new mixture concentrations are then passed back to the transport calculation—this iterative process continues until all depletion steps have been completed.

It is entirely up to the user which materials are depleted and how many depletion steps are used. These are two very important key parameters that the user must fully understand prior to attempting depletion calculations. All fuel materials need to be depleted. As a reactor operates, fissile nuclei undergo fission reactions, which creates two lighter isotopes. The consumption of fissile nuclei and production of lighter nuclei are modeled only if the user specifies the compositions that contain these fissile nuclei. Mixtures that are not specified by the user will not be depleted by SCALE/TRITON. It is also important to deplete materials that contain burnable poisons—gadolinium, boron, etc. The concentration of these burnable absorbers decreases over depletion, and so the user must also specify these materials. Users need not deplete structural materials, moderator, or any other material whose composition does not change significantly under neutron irradiation. You do not need to deplete moderator mixtures (water) that contain boron because reactor operators strictly control the concentration of boron during depletion. Boron is being consumed from neutron capture, but reactor operators are adding boron at the same rate. A boron letdown curve can be modeled using the SCALE/TRITON **Timetable** feature, but the **Timetable** feature is omitted from this primer because nodal simulators typically model the boron letdown curve by interpolating between different boron concentrations contained in the broad-group cross sections.

The other major consideration is the length of depletion steps. A reactor is under continuous operation—isotopic concentrations, fluxes, pin powers, multigroup cross sections, etc. change continuously as a function of time. SCALE/TRITON employs an iterative procedure to model this behavior. Error will be introduced if too few depletion steps are taken during a period of simulated operation. To make matters more complicated, assemblies that contain strong burnable absorbers require finer depletion steps than assemblies that contain only fissile fuel. There is no substitute for a firm understanding of your problem. Only you can determine what depletion step scheme is detailed enough for a certain assembly configuration. The most accurate transport solver can only be as accurate as the isotopic concentrations and the cross sections that the transport solver uses. For this reason, care must be taken to ensure that adequately fine time steps are used. With increasingly fine depletion step schemes, the solution should converge to some answer. If you have doubt that the depletion steps you have used are fine enough, run another simulation with finer depletion steps and observe the difference in the two solutions. If the difference is significant, then you probably need finer depletion steps. Generally, very fine depletion steps are needed for gadolinium-bearing lattices, 0.5 to 1.0 GWd/MTU before peak reactivity. After peak reactivity, the depletion steps can be lengthened. The recommended step size for non-gadolinium-bearing lattices is approximately 2 to 3 GWd/MTU or smaller.

6.2.1 Mini Lattice with Depletion

To become more familiar with the depletion inputs, use GeeWiz to open the previous problem with the file name `mini1_cmfd.inp` and **Save As** `mini1_depl1.inp`. Recall that this problem is the 2×2 array of 4.00% enriched fuel pins. In the **General** settings, change the **Sequence** from `T-NEWT` to `T-DEPL` in the **Sequence** dropdown. Making this change will result in the **General** settings window changing slightly. You should have some new options: **Orgnflux** and **AddNux**. The **Orgnflux** option will not be used, but the **AddNux** option will be. Change the **AddNux** option to `2` in the **AddNux** dropdown; click **OK**. On the main **T-DEPL form**, you will notice some new options on the left vertical toolbar: **Depl** and **Branch**. Click the **Depl** button on the left toolbar and you will see the **Depletion Data Blocks** window appear (Fig. 6.1).

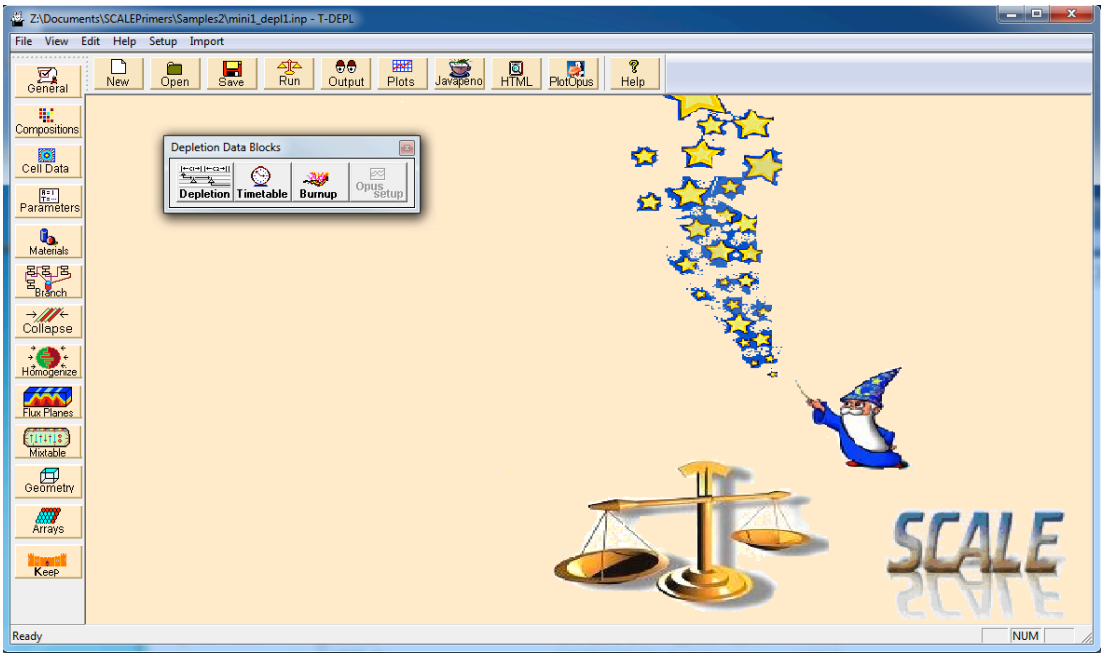


Fig. 6.1. GeeWiz window for the T-DEPL sequence.

In the **Depletion Data Blocks** window, the **Depletion** and **Burnup** options will be used quite extensively. Click the **Depletion** option from the **Depletion Data Blocks** window and the **Depletion Data** window will appear. The user can select which mixtures to deplete in the **Depletion Data** window. This simple problem only contains one material to be added: `1 uo2`, which can be selected from the **Mixtures** box in the **Depletion Data** window. Select `1 uo2` so that it is highlighted and then click **Add Mixture**. Mixture number 1 should be added to the **Mixtures** to **Deplete** box. The final **Depletion Data** window can be found in Fig. 6.2. Now click **OK**.

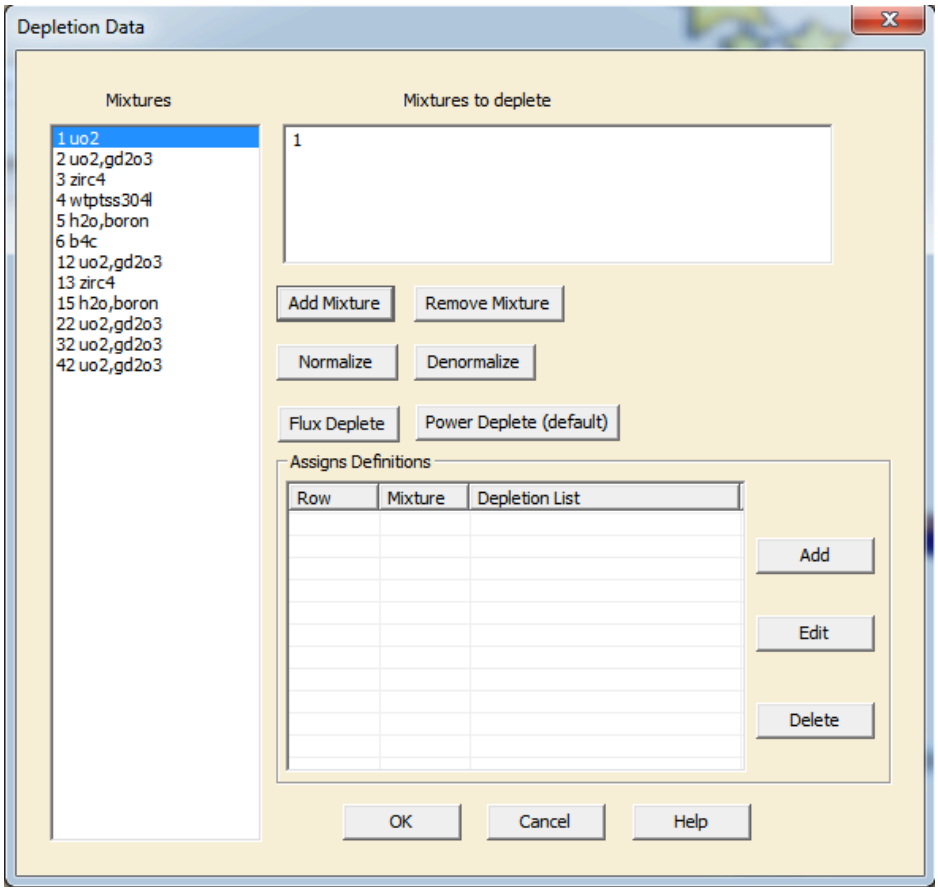


Fig. 6.2. Depletion data window.

In the **Depletion Data Blocks** window, click **Burnup**. The **Burn Data** window will appear, where a user can specify any combination of different depletion steps. The **Burn Data** inputs are **Power** (specific power), input in units of megawatts per metric ton uranium (MW/MTU); **Burn** (full-power irradiation time), input in days; **Libraries**, the number of cross-section updates in a certain step; and the **Downtime** (zero-power decay time). Enter `25` MW/MTU for **Power**, `1000` days for **Days**, `5` for **Libraries**, and `0` for **Downtime** (Fig. 6.2). Click **OK**. These parameters specify depletion to a burnup of 25,000 MWd/MTU with five updates to the cross sections.

Fig. 6.3. Burn data window.

Now you should have a complete depletion input file for the mini assembly. This problem should run a total of six transport calculations: the t=0 calculation and the five depletion steps that were specified using the **Libraries** option. SCALE/TRITON performs depletion calculations for the midpoints of the requested steps. Intuitively, a user might think that because this problem starts at time t=0 days and goes to time t=1000 days with five library updates, the transport calculations will be performed at t=0, 200, 400, 600, 800, and 1000 days. Because SCALE/TRITON performs depletion calculations at the midpoints of the steps, the transport calculations are actually performed at t=0, 100, 300, 500, 700, and 900 days. This input will take much longer to run since it must run six different transport calculations. During a SCALE/TRITON depletion calculation, a number of nuclides are added in trace amounts, making the cross section processing longer as well. You should turn on **Parm=Check** to make sure there are no errors in the input file. After the check completes without error, turn off **Parm=Check** and **Run** the problem.

In the return directory (where your input file is located) you will have the PostScript and output files, as in previous cases. In addition, you will also have a *.plt file that corresponds to each depletion material. The *.plt files contain isotopic concentration information for each burnup material at each depletion step. The eigenvalue trajectory for this sample problem has been plotted in Fig. 6.4. The two different trajectories use a different amount of library updates. The 6 points from this sample problem form the blue curve, while 16 depletion steps (**Libraries**=15) in the same total depletion time form the red curve. The **Libraries**=5 problem has 5 GWd/MTU depletion steps, while the **Libraries**=15 problem has 1.67 GWd/MTU depletion steps. The difference between the two solutions is plotted on the secondary axis. You can see that the difference between the 16-step and the 6-step problems increases with depletion.

This simple problem is designed to illustrate the need for relatively fine depletion steps. The total computer time for this six-step problem was 64.15 minutes. The NEWT calculations only took a small portion of time for this problem, averaging ~40 seconds at each step.

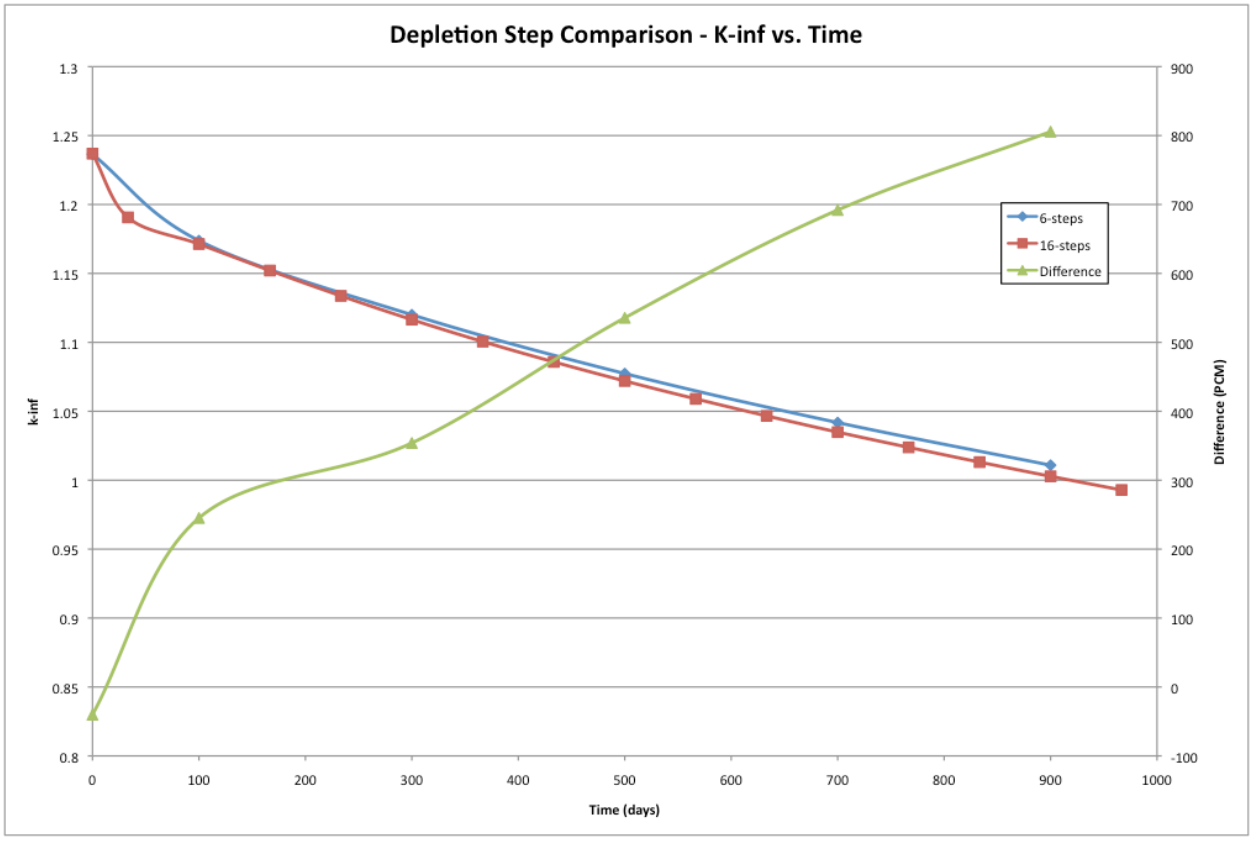


Fig. 6.4. Eigenvalue trajectories using two different depletion step schemes.

6.2.2 Using Parm=Weight

The Parm=Weight option can be very useful in lattice physics analysis. Running SCALE/TRITON depletion calculations requires a significant amount of computer resources. To make SCALE/TRITON run faster for depletion calculations, you may want to consider using the Parm=Weight option. This option allows the user to generate a problem-dependent 49-group cross-section library at the start of a depletion calculation. SCALE/TRITON will first run a beginning-of-life steady-state calculation with a 238-group cross-section library. SCALE/TRITON will use the neutron spectrum from steady-state solution to collapse the 238-group library down to a 49-group library that will be used for the subsequent depletion calculation. In effect, this makes the transport calculations run much faster and should result in a faster solution. To turn on Parm=Weight, go to the General settings and click in the checkbox next to Weight (Fig. 6.5).

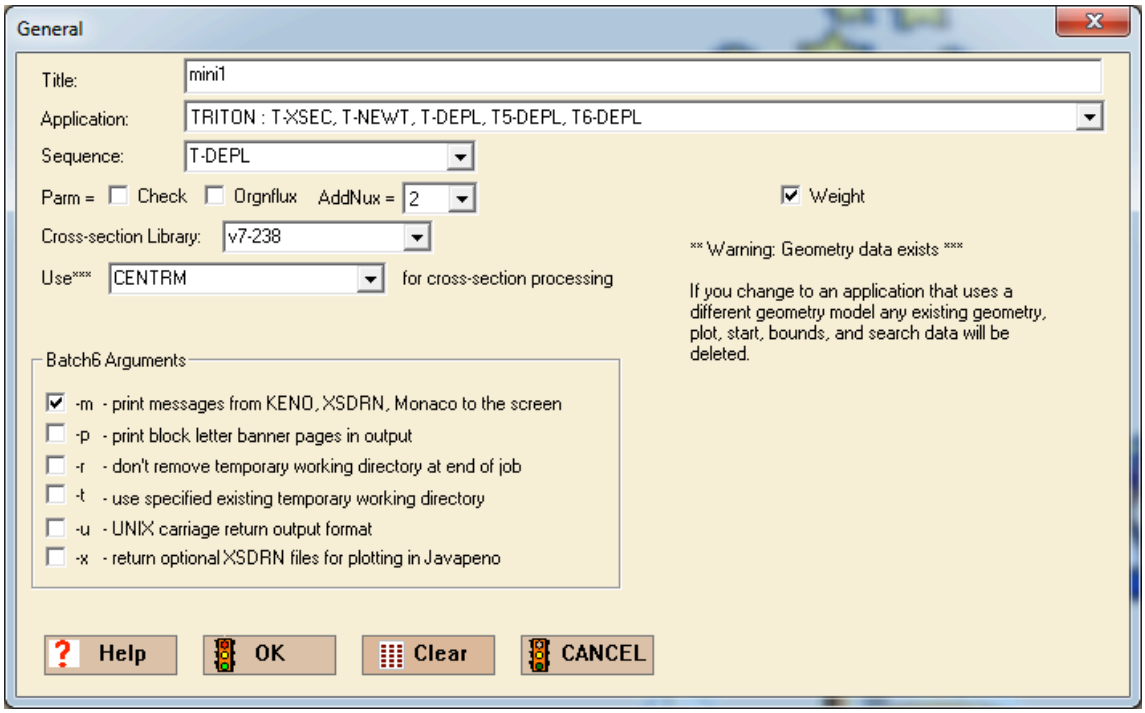


Fig. 6.5. General settings window that uses Parm=Weight.

The same six-step depletion problem took 27.71 minutes with Parm=Weight vs. 64.15 minutes without Parm=Weight. The NEWT transport calculations took ~10 seconds with Parm=Weight. Use of Parm=Weight is recommended for production runs where computational time is at a premium. The bias introduced in the library collapse should be investigated to make sure that the bias is acceptably small prior to running production calculations. This bias is typically less than 200 pcm for LWRs.

It is possible to save the cross-section library that is generated when using the Parm=Weight command for use in subsequent calculations. To save the cross-section library generated from a Parm=Weight run, copy the file named *newxnlib* in the temporary working directory to the return directory and rename the file. To use this file in subsequent calculations, add it to the input file using the text-based input format. In the text-based input, the first line signifies the sequence (T-NEWT, T-DEPL, etc.), the second line is a model title, and the third line is the cross-section library for the problem. On the third line of the input file, enter the name of the 49-group library instead of one of the SCALE-supplied cross-section libraries. Remember that this library was collapsed using a specific neutron spectrum. You will introduce error if you apply the 49-group library to a problem whose spectrum is significantly different than the spectrum that was used to generate the library. It is always safest to use a 238-group library, but in the interest of reducing computational cost, the Parm=Weight option can be applied to most depletion problems without significant error.

6.3 ADVANCED LWR LATTICE DEPLETION

Accurate depletion of advanced assembly designs in SCALE/TRITON may require that users specify a different fuel mixture for every unique fuel pin in a lattice. A different mixture must be specified for every unique fuel pin because the spatial flux distribution varies significantly throughout a lattice model. Figure 6.6 shows the thermal flux distribution for a detailed BWR lattice model.

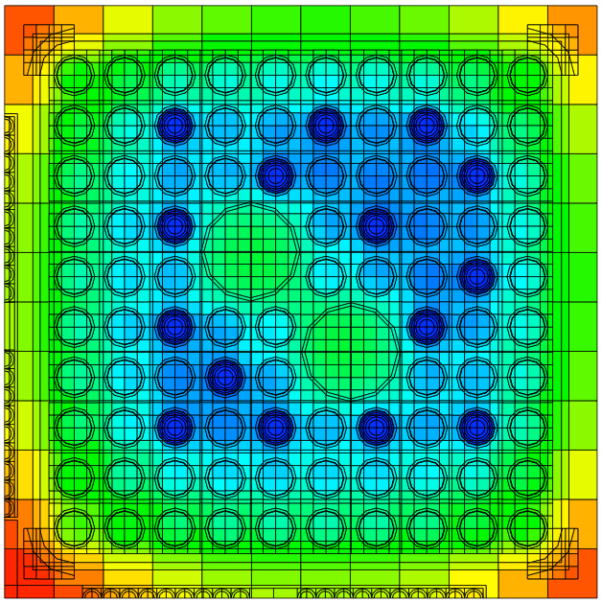


Fig. 6.6. Flux distribution for a detailed BWR lattice.

Observation of the flux distribution shows that the thermal flux changes significantly as a function of lattice position. In addition, the flux distribution changes as a function of depletion. Because the flux distribution is a function of time and space, fuel pin mixtures must be depleted independently in order to accurately track isotopic concentrations during depletion. A 10×10 BWR lattice with water rods, such as the one shown in Fig. 6.6, would require 49 different fuel materials. The lattice has a total of 92 fuel pins, but with diagonal symmetry, the number of different fuel pin locations can be reduced to 49. Users must specify different fuel pin materials for each unique fuel pin location in order to obtain pin-by-pin tracking of isotopic concentrations. Before performing a full lattice calculation, this effect can be seen in a smaller model. (*Hint: You can create flux plots like Fig. 6.6 for any geometry by setting "prtflux=yes" in the NEWT parameters. Be careful—NEWT makes a flux plot for every energy group so the files can be large if using a fine group library such as a 238-group library.)*

ORIGEN-S, the depletion module used by SCALE/TRITON, has two options for depletion of materials: depletion by constant power, and depletion by constant flux. Because ORIGEN is a point depletion code, it is used to independently model distinct depletion regions (mixtures) instead of the fuel assembly as a whole. Coupling between these regions is achieved using the transport solution for the assembly; relative fluxes across an assembly are used to determine the local flux or power in each individual ORIGEN calculation. ORIGEN supports two modes of input for depletion calculations: power or flux. When flux is input, that value is used directly in the ORIGEN solution. However, when power is input, ORIGEN will automatically adjust the flux to maintain a constant power, based on power-producing reactions to scale the flux. While depletion assuming a constant power is generally representative of a fuel element, this is not true for nonfuel elements. Targets, structural materials, and burnable poisons are generally "driven" by fluxes from neighboring fuel elements and do not contribute significantly to power production. Forcing a constant power in a burnable poison rod, for example, would result in a rapidly increasing flux due to depletion of the poison material, which is nonphysical. Allowing mixed-mode depletion within SCALE/TRITON better approximates the time-dependent flux behavior across an assembly.

For fuel pins that **do not** contain integral fuel burnable absorber, users should deplete by constant power. For fuel pins that **do** contain integral fuel burnable absorber, users should deplete by constant flux because the specific power of these regions could change significantly over a depletion step.

6.3.1 Mini-assembly with Different Fuel Pin Mixtures

Using the mini-assembly problem, you will add an extra fuel material and change the fuel array slightly. The mini-assembly problem that was set up previously has four fuel pins arranged in symmetric locations. This example will add some asymmetry to the model in the form of a vanished rod. The first problem you will run using three identical fuel materials. The second problem you will run using two different fuel materials: one for the northeast corner pin, and another for the northwest and southeast fuel pins (Fig. 6.7). The mini-assembly model on the left uses one fuel mixture, while the assembly on the right uses two fuel mixtures of identical composition. Because the flux will be more thermal for the northwest and southeast fuel pins (due to the vanished rod), the fuel in these locations will deplete faster than the fuel pin in the northeastern corner.

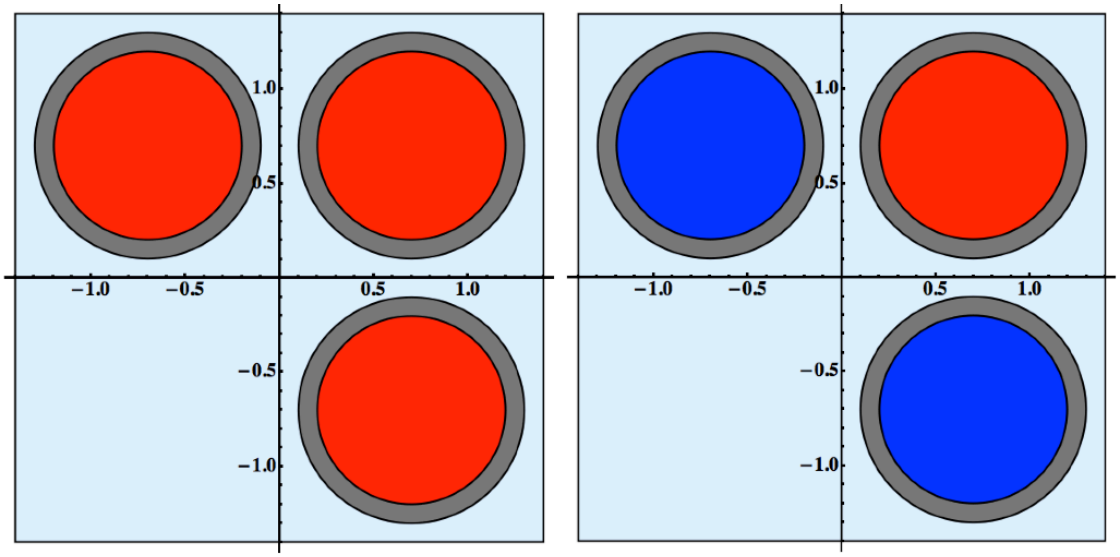


Fig. 6.7. Drawings of the one-fuel and two-fuel mini-assembly models.

To set up the first model, open `mini1_depl1.inp` and **Save** it to `mini1_depl_1fuel.inp`. Click **Arrays** to open the **Array Form**. Place unit `10` (empty water cell) in the southwestern corner of the model. The **Array Form** for this array can be found in Fig. 6.8.

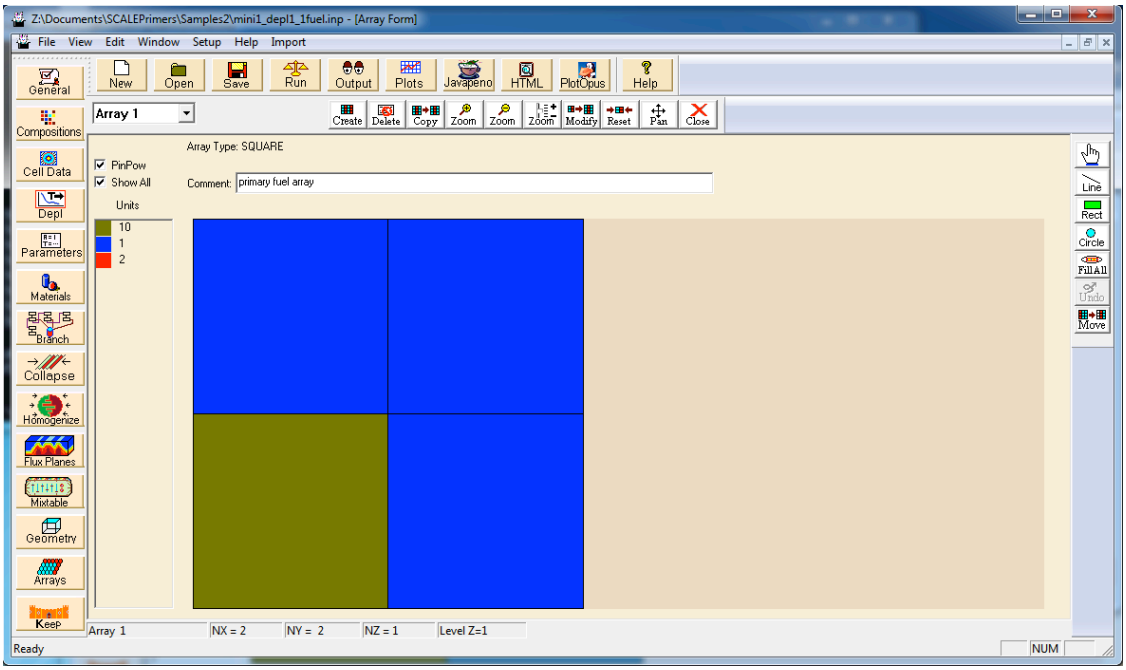


Fig. 6.8. Array form for the one-fuel lattice with a vanished rod.

This is the *only* change that needs to be made to this model. Click Save **and then click** Run**.**

With this model running in the background, you can open another GeeWiz window by navigating to the shortcut and double clicking the GeeWiz icon. Now Open `mini1_depl1_1fuel.inp` and Save As `mini1_2fuel_depl.inp`. You will need to add new fuel, clad, and moderator mixtures for this problem. Click Compositions from the left toolbar and, using the Copy Mix button, Copy mixture `1` to `201`, mixture `3` to `203`, and mixture `5` to `205`. You will also need to add a new Cell Data `latticecell`. Click Cell Data from the left vertical toolbar and then click New Cell. Add a new `latticecell` that is identical to the previous lattice cell, but with mixtures `201`, `203`, and `205` (Fig. 6.9).

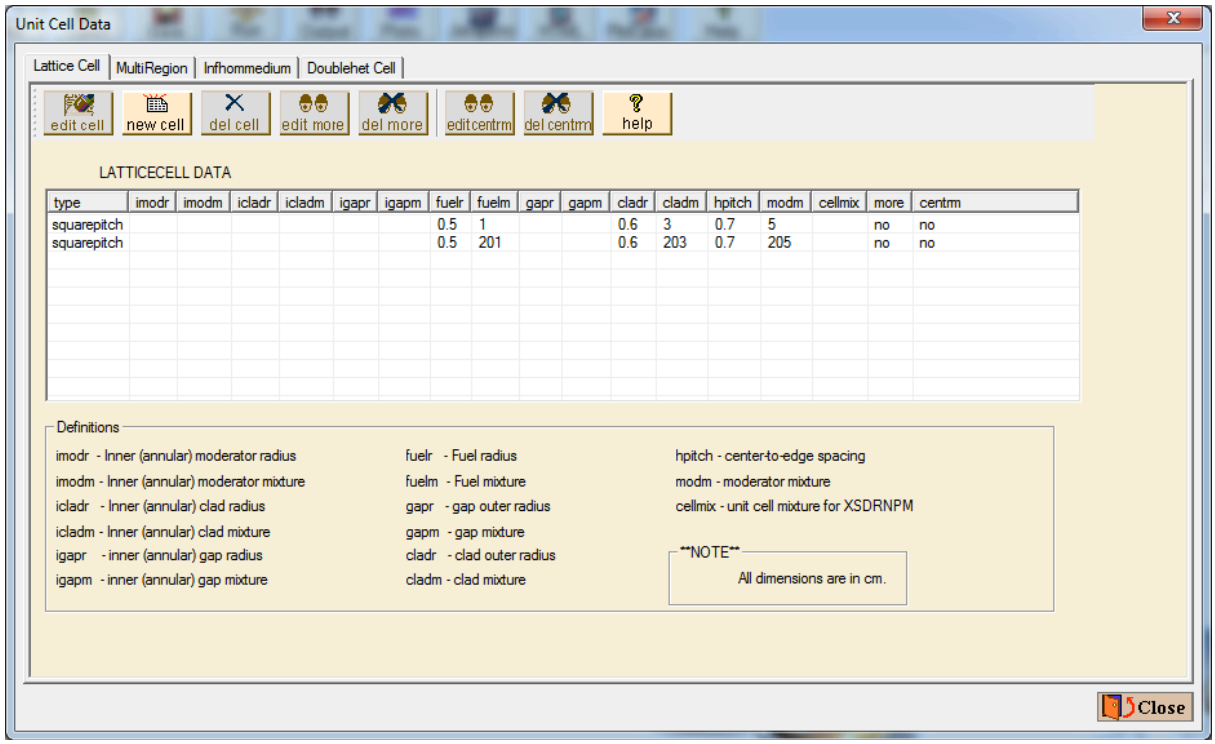


Fig. 6.9. Unit cell data for additional lattice cell in the two-fuel mini-assembly.

You should also add the new fuel material to the list of NEWT materials. Click Materials along the left vertical toolbar and add mixture `201` to the Material Mixtures. You will then need to add a new unit to the geometry that is identical to the other fuel pin cell, unit `1`. You will also need to add the new fuel mixtures as depletion mixtures. Navigate to unit `1` on the Geometry Form and click Copy Unit. This example uses unit `12`. Simply go into the `media` statements and change the fuel media from mixture `1` to mixture `201`. You will have to remove the media and then add it back using the new mixture `201` (Fig. 6.10). It is okay to leave the clad and moderator mixture as the same values in the NEWT model. Make sure that the grid spacing is still `4×4`—sometimes GeeWiz does not copy the grid spacing.

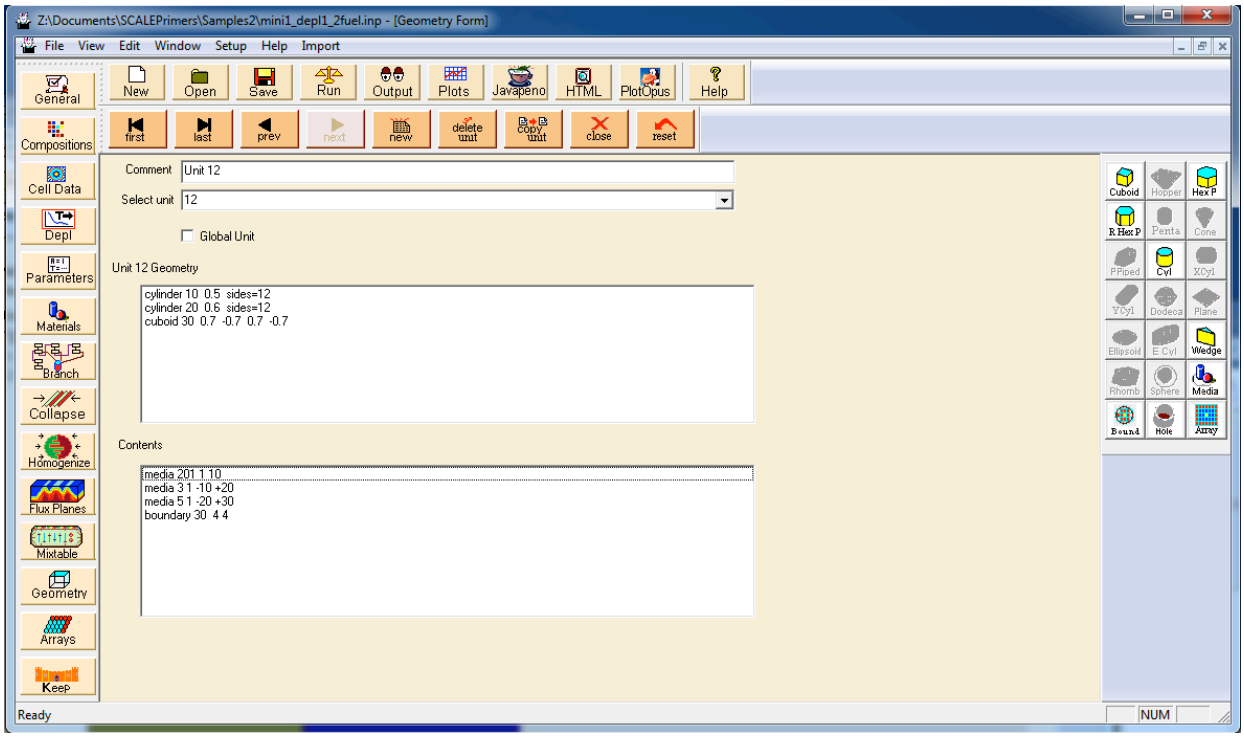


Fig. 6.10. Geometry form for additional unit required for two-fuel mini-assembly problem.

Now click **Arrays** on the left toolbar. You should change the northwest and southeast units to unit `12` (Fig. 6.11).

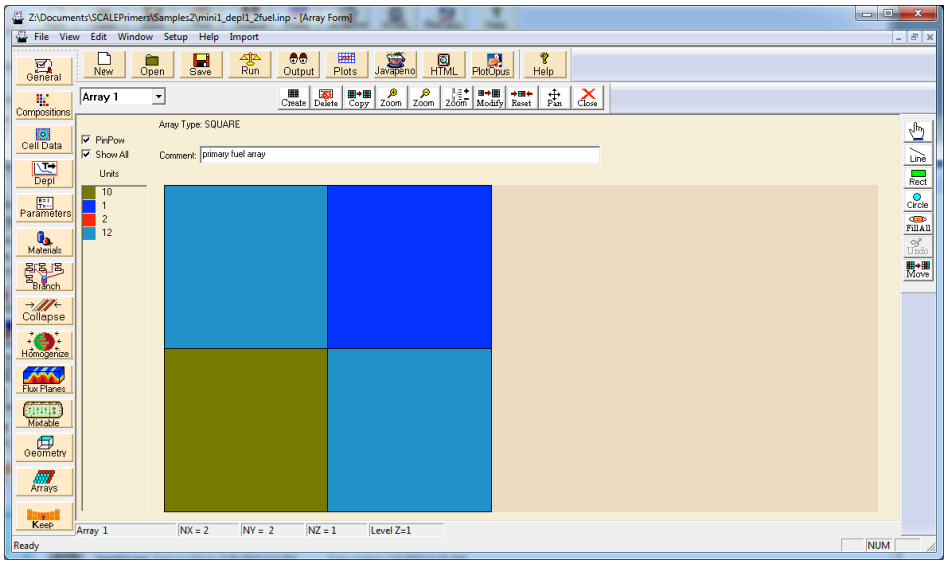


Fig. 6.11. Array form for the two-fuel mini-assembly problem.

If you compare the results from these runs, you will see diverging behavior between the two solutions. Figure 6.12 shows the difference between the one-fuel and the two-fuel problem ($k_{1\text{-fuel}} - k_{2\text{-fuel}}$) as a

function of time. The two-fuel depletion problem tracks isotopic concentrations for every symmetric pin, whereas the one-fuel depletion problem lumps all the isotopic concentrations from the fuel into one material. Note that for this three-pin problem, you do not need three different materials—only two are needed due to symmetry. The difference between a uniform fuel mixture and separate mixtures for each unique location is more significant for larger and more complicated lattices (especially in BWR nonuniform lattices). As a side note, the total runtimes were 23.04 and 32.68 minutes for the one-fuel and two-fuel problems, respectively.

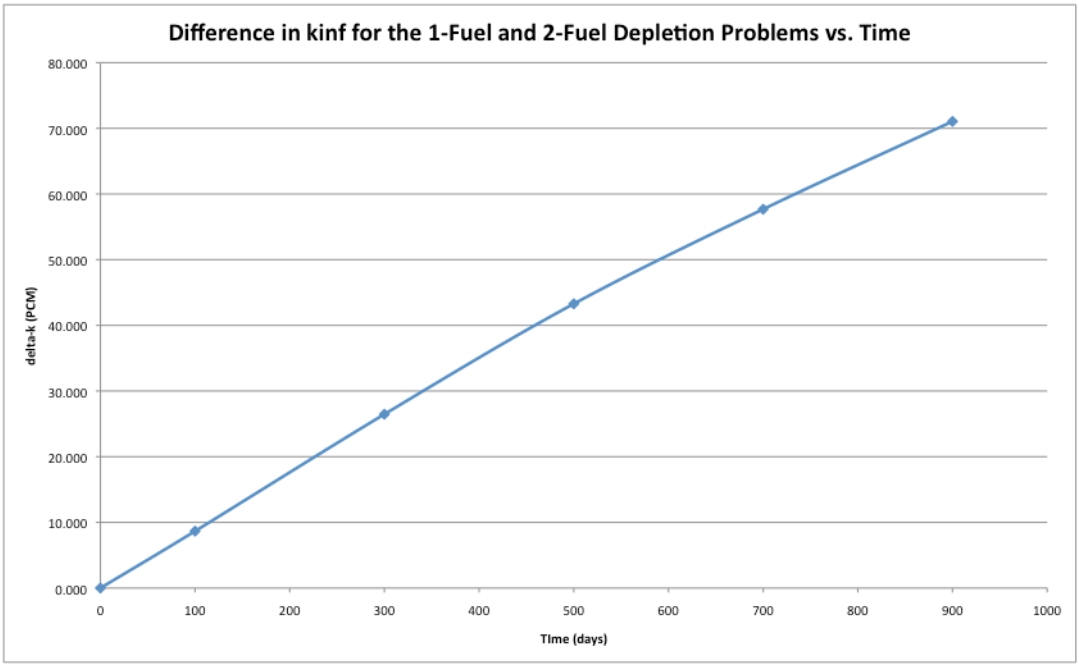


Fig. 6.12. Eigenvalue difference between one-fuel and two-fuel mini-assembly problems.

6.3.2 Mini-assembly with Separate Fuel Pins Using the Assign Function

Adding another fuel mixture to the model increased CPU time rather significantly, approximately 50 percent, from the one-fuel to the two-fuel problem. The primary reason for the increase is the additional SCALE/CENTRM self-shielding calculation for the extra fuel mixture. If you were to model every pin in a full lattice with different SCALE/CENTRM self-shielding calculations, the self-shielding portion of the calculation would dominate the total computer time used. In the two-fuel problem, there are two identical SCALE/CENTRM lattice cell calculations. The initial mixture compositions did not change—they were simply copied from the other fuel, clad, and moderator mixtures. The unit cell dimensions did not change either, so the two SCALE/CENTRM calculations are identical. Fortunately, there is a way in SCALE/TRITON to eliminate the redundant SCALE/CENTRM calculations. This is done using the Assign function in the depletion settings.

The Assign function simply copies the cross sections from one mixture to another. If each fuel has been defined as a different mixture for depletion, the isotopic concentrations for all fuel mixtures are tracked separately. When using the Assign function, SCALE/TRITON "averages" the mixture isotopic number densities in the assigned set of mixtures for the self-shielding calculation. The microscopic cross sections from that self-shielding calculation will then be copied back to the individual materials. The isotopic

number densities of the different fuel mixtures remain independent, but the microscopic cross sections for each fuel mixture are updated each depletion step using the average isotopic concentrations of the materials used in the Assign function.

To set up a simple problem that uses the Assign function, you will need to edit the ASCII text input file. The text input file is fairly simple and straightforward to use. Advanced features of the TRITON sequence are usually released before updated versions of GeeWiz, so GeeWiz development sometimes lags SCALE/TRITON development. Even though some features require use of the text-based input, you can use GeeWiz to set up the basics for most problems. You will eventually become familiar with the text-based input and may find that you can set up input files faster using the text-based input rather than GeeWiz. Be careful using the text-based input—the input process is much more error prone than using the GeeWiz GUI. To begin, use GeeWiz to do most of the problem setup, and then switch to the text-based input for adding advanced features.

To set up a sample problem using the Assign function, open `mini1_2fuel_depl.inp` in a text editor by clicking **Edit>Edit File.** Save the file with the new filename `mini1_2fuel_depl+assign.inp`. Because you will assign the cross sections from one self-shielding calculation to another material number, you do not need the extra fuel, clad, and moderator mixtures in the composition block. Highlight all the lines for mixtures 201, 203, and, 205 (seen below) and delete these lines.

```
read composition
 ...
 uo2          201 den=10.6312 1 900
                                92234 0.005407837
                                92235 4
                                92238 95.99459   end
 zirc4        203 1 540   end
 h2o          205 den=0.8458 1 500   end
 boron        205 den=0.8458 0.0009 500   end
end composition
```

Also remove the extra latticecell specification that was added to the celldata block, highlighted below.

```
read celldata
  latticecell squarepitch fuelr=0.5 1 cladr=0.6 3 hpitch=0.7 5 end
  latticecell squarepitch fuelr=0.5 201 cladr=0.6 203 hpitch=0.7 205 end
  multiregion cylindrical left_bdy=reflected right_bdy=white end
           2       0.2236
          12       0.3162
          22       0.3878
          32       0.4472
          42          0.5
          13          0.6
          15       0.7899
      end zone
end celldata
```

Now move to the depletion block of the input file. In the depletion block, you will see that mixtures 1 and 201 are being depleted. Do not change this; you still want both mixtures 1 and 201 to be depleted. You will need to add an END keyword to the end of the depletion materials. The Assign function works by assigning a source (S) material to target (T_n) material(s) and is terminated by an END keyword:

```
assign S  T1 T2 T3 … Tn  end
```

In order to assign the cross sections from mixture 1 to 201, add the elements highlighted in yellow to the depletion block:

```
read depletion
  1 201 end
  assign 1  1 201 end
end depletion
```

SCALE/TRITON assigns the cross sections from mixture `1` (source) to mixtures `1` and `201` (targets). Always include the source mixture in the list of targets; this is so SCALE/TRITON averages isotopic concentrations from mixtures 1 and 201 back to mixture 1 for self-shielding calculations. This is the only change needed for the input file. You should now save the input file. Note that you did not change anything in the mini-assembly geometry. From the previous problem, mixtures 1 and 201 are in the correct locations. However, the way self-shielding cross-section adjustments are accomplished for mixtures 1 and 201 during depletion is modified. Media 201 should still be used in the NEWT model.

This input file contains features that are unavailable in GeeWiz, so do not open this file in GeeWiz to run it. Opening the file in GeeWiz will cause the unrecognized text to be removed. Instead, open the SCALE6 shortcuts folder on the desktop or in the scale6 directory. Inside the shortcuts folder, there will be a shortcut named "Run SCALE6". Double clicking the "Run SCALE6" icon opens a DOS command prompt that has the correct links established for running SCALE. Using the command prompt, navigate to the directory in which your input file is located. You can run SCALE by typing:

```
batch6 –m filename
```

This command calls the batch script that runs SCALE 6. For the example shown here, the "–m" option is used to print all messages back to the screen, rather than in a messages file (Fig. 6.13). There are a number of options available to users for running SCALE from the command prompt. The command prompt and other useful runtime options can be found in the document titled "Getting Started" on the SCALE 6 installation DVD.

After the problem completed, the eigenvalues were extracted from the output file and compared to the previous results for the one-fuel and two-fuel models. These results are plotted in Fig. 6.14.

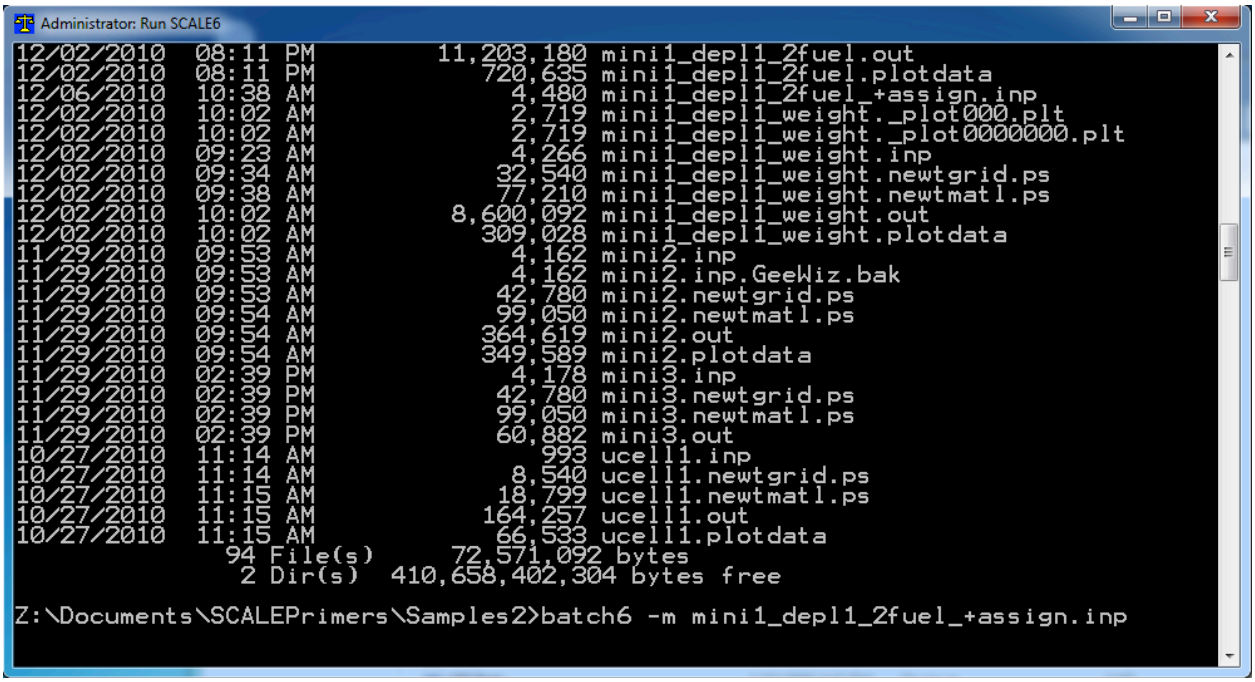


Fig. 6.13. Command prompt for running SCALE 6 with the batch script.

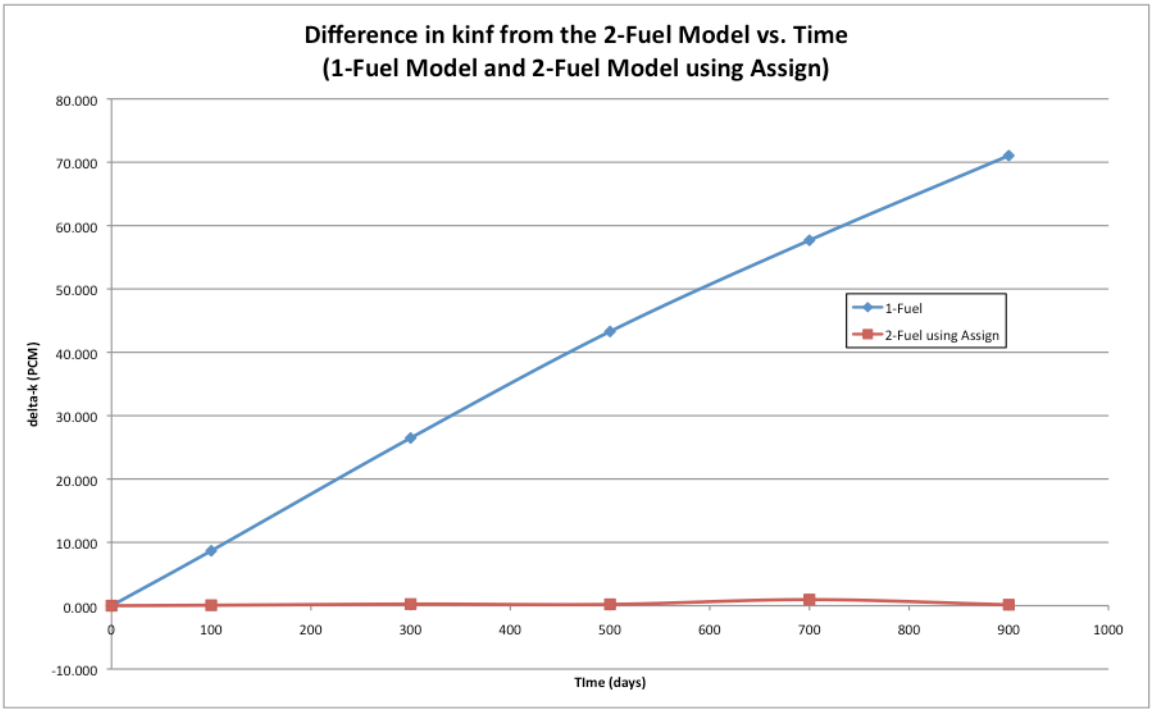


Fig. 6.14. Comparison of results using the Assign function.

From Fig. 6.14, you can see that the two-fuel model using the Assign function matches the reference two-fuel solution very well. Remember that the two-fuel reference solution uses two separate CENTRM unit cell calculations for self-shielding. This method is the most accurate but also takes the most computer

time. The previous sample problem runtimes were 23.04 and 32.68 minutes for the one-fuel and two-fuel models, respectively. The runtime for this problem was 24.19 minutes. By using the Assign function, the computational time was reduced by 8.5 minutes (26%) and with virtually no impact on the accuracy when compared to the two-fuel reference model. It is recommended that you use the Assign function in lattice physics problems in order to limit computational cost while maintaining accuracy of the problem solution. Remember that every unique fuel pin location should be depleted as a different material—using the Assign function will allow you to do so without a major impact on runtime and accuracy.

6.4 SUMMARY

This section has helped you accomplish the following:

- use GeeWiz and SCALE to perform standard depletion calculations for pin cells and lattices
- use the Parm=Weight option to generate problem-dependent 49-group cross section libraries
- understand the advanced concepts required for detailed depletion of lattices
- use the Assign function to assign material cross sections to different materials.

7. LWR BRANCH CALCULATIONS

7.1 WHAT YOU WILL BE ABLE TO DO

After completion of this section, you should be able to do the following:

- understand what branch calculations are and why they are needed for lattice physics
- use GeeWiz to add branch calculations to a model
- understand typical branch calculations required for PWRs and BWRs
- understand the need for simple test cases to verify branch conditions

7.2 BRANCH CALCULATIONS IN LWR LATTICE PHYSICS

The primary purpose of lattice physics calculations is to generate broad-group nodal diffusion data (cross sections, diffusion coefficients, discontinuity factors, scattering matrices, etc.) for reactor core simulators. The most current cutting-edge software on the largest computer systems is only now becoming capable of modeling full-core nuclear reactors in extreme detail. The computing resources and time required to generate such a solution would be prohibitive for typical users who do not have access to such software and computer systems nor do they have the time to wait for an answer of such detail. Users require a fast solution that is capable of running on a normal desktop computer. To this end, reactor core simulators based on nodal diffusion theory have been developed as a capable tool for full-core reactor analyses. Nodal simulators rely on "average" data for different lattices. Reactor cores are composed of a large number of fuel assemblies (or bundles), and each assembly is composed of an array of fuel pins. Each fuel assembly usually has some axial variation—changes in enrichment, types of fuel pins, number of fuel pins, etc. Each axial level of an assembly is generally called a lattice. So there might be a number of different lattices that make up one assembly model. In lattice physics, users build and run detailed models of each lattice in an assembly to generate data for that lattice. The lattice data are converted to a format that the core simulator can read. In SCALE/TRITON the lattice-average data are saved in a binary file named *xfile016* and a text-based file name *txtfile16*.

Typical reactors may contain many different types of assemblies. These assemblies experience depletion of fissile nuclides and buildup of fission products and transuranics over multiple operating cycles. In addition, reactors may undergo certain transients such as a change in moderator density, insertion or removal of control rods, a change in fuel or moderator temperature, and changes in moderator composition (soluble boron in PWRs). For a core simulator to model these changing conditions, the core simulator must have data for these conditions. Branch conditions are used to generate the data required for these transient conditions.

Branch conditions should be specified to account for all normal and off-normal operating conditions that need to be modeled for safety analyses. The nominal conditions, or branch 0 conditions, are specified in the base input file *and* specified in the first branch condition of the branch block. At each depletion step, SCALE/TRITON generates broad-group nodal diffusion branch data by adjusting the branch 0 model to the requested branch condition and rerunning the cross-section processing and transport modules. The broad-group nodal data are then written to the *xfile016* and the *txtfile16* files. So the *xfile016* and *txtfile16* files for any depletion calculation will contain nodal parameters for every depletion step and every branch condition specified in the model.

Nodal core simulators interpolate between a set of data points to obtain nodal parameters for instantaneous operational conditions. For this reason, a branch condition does not need to be specified for every possible operating condition. Rather, the envelope formed by the branch conditions should cover the range of possible operational conditions. If a core simulator predicts an operating condition between two branch conditions, the nodal parameters are interpolated based on available data. If a core simulator predicts an operating condition that lies outside the envelope of branch conditions, the nodal parameters are extrapolated to that condition-based available data. Extrapolation of nodal parameters is particularly risky, so all foreseeable operating conditions should lie inside the envelope formed by the chosen branch conditions.

7.3 BRANCH CONDITIONS FOR PWRS AND BWRS

For PWRs, the branch conditions should contain the following:

- a set of moderator densities above, below, and at the expected nominal conditions
- a sufficient number of soluble boron concentrations to simulate the boron letdown curve
- control rod branches, both in and out, that correspond to these aforementioned moderator densities and boron concentrations
- conditions that correspond to startup and shutdown

A sample list of operating conditions can be found in Table 7.1. This sample list should not be considered a complete list for your problem; rather, it is a sample of what branch conditions for a PWR might look like.

Table 7.1. Sample PWR branch conditions for nominal and limited transient conditions

Branch no.	Moderator density (g/cm^3)	Fuel temperature (K)	Moderator temperature (K)	Soluble boron (ppm)	Control rod state (0=out, 1=in)
0	0.7	925	575	900	0
1	0.6	925	575	900	0
2	0.8	925	575	900	0
3	0.9	925	575	900	0
4	1	925	575	900	0
5	0.7	925	575	900	1
6	0.6	925	575	900	1
7	0.8	925	575	900	1
8	0.9	925	575	900	1
9	1	925	575	900	1
10	0.7	925	575	0	0
11	0.6	925	575	0	0
12	0.8	925	575	0	0
13	0.9	925	575	0	0
14	1	925	575	0	0
15	0.7	925	575	0	1
16	0.6	925	575	0	1
17	0.8	925	575	0	1
18	0.9	925	575	0	1
19	1	925	575	0	1
20	0.7	925	575	2200	0
21	0.6	925	575	2200	0
22	0.8	925	575	2200	0
23	0.9	925	575	2200	0
24	1	925	575	2200	0
25	0.7	925	575	2200	1
26	0.6	925	575	2200	1
27	0.8	925	575	2200	1
28	0.9	925	575	2200	1
29	1	925	575	2200	1
30	0.7	294	575	900	0
31	0.7	1550	575	900	0
32	0.7	3000	575	900	0
33	0.7	294	294	900	0
34	0.7	294	294	0	0
35	0.7	294	294	0	1

For BWRs, the required branch conditions are generally less straightforward. Users will generally need branch conditions with moderator densities corresponding to 0, 40, and 80 percent void fraction, control rod in/out branches, and branches corresponding to startup and shutdown. You might need to add soluble boron branches depending on the transient(s) being analyzed. Rapidly changing moderator density (due to boiling) and control rod placement in BWRs necessitate the need to generate nodal data that has void and/or control rod "history." In BWRs, the void fraction has a strong effect on the neutron spectra. The changing spectra will result in different rates of depletion of fissile isotopes and buildup of fission products and transuranic isotopes. Because of changing spectra that are dependent on void fraction and control rod position, BWR nodal parameters should have some history associated with how the lattice was depleted. The lattices in the upper portions of a BWR bundle see a higher void fraction than the lattices located in the lower portion of the bundle. Users will likely need to run depletion calculations

whose branch 0 moderator conditions are 0, 40 and 80 percent void. So for every lattice you likely have to run three different depletion calculations depending on the void history alone. At times, BWRs operate with the control rod blades partially inserted—users might need history calculations for control rods in addition to void fraction. Because of the wide range of operating conditions that are possible with BWRs, sample branch conditions have been omitted from this document to avoid misleading users into thinking that a sample set of conditions given here would be adequate for their problem.

Once a model with branch conditions has been run and the *xfile016* and *txtfile16* files have been generated, the *xfile016* file is processed with a conversion utility that creates nodal parameters for core simulators. For the PARCS nodal core simulator, GenPMAXS (PARCS cross-section conversion utility) is used to generate PMAXS files that contain the required nodal parameters. ORNL has the T2N utility that performs the same conversion for the NESTLE nodal code. It is possible for a user to generate a set of branch conditions that is too complex to be processed with GenPMAXS or another conversion utility. The computer time that it takes to run a SCALE/TRITON depletion calculation with branches is very large. It is recommended that you set up a simple, fast running test case to avoid the risk of running a SCALE/TRITON depletion calculation with a incompatible set of branches. The test case can be a simplified lattice, such as 2×2 array or pin cell, which contains the full set of branches. You can then run this simple 2×2 case for just a couple of depletion steps. Once complete, use GenPMAXS or another conversion utility to convert the *xfile016* file to required format. If the conversion succeeds for the test case, you can feel confident in running the depletion calculations for a full lattice.

7.4 SAMPLE PROBLEM—MINI-ASSEMBLY WITH BRANCHES

For this example, the familiar mini-assembly problem will be used. In GeeWiz, open `mini1_2fuel_depl.inp` and Save As `mini1_2branches.inp`. Note that this example does not use the Assign function, so that GeeWiz can be used throughout the demonstration. Because branches will be added, the runtime will be much longer. The purpose of this problem is to show how the SCALE/TRITON branch structure works, so only a couple of depletion steps are needed. Click **Depl** on the left toolbar and then click **Burnup** from the **Depletion Data Blocks** window. Change the number of **Libraries** from `5` to `2` and then click **OK**. Note that the depletion step size will not yield accurate solution—it should be shorter in an accurate production calculation. Because this example simply illustrates the use of branches, you can use any step size, but remember to use appropriately smaller step sizes for production calculations.

Now navigate to the **Geometry** form for unit 10. Remember that unit 10 is the water cuboid in the southwestern corner of the model. Change the **Media** in this cuboid from mixture `5` to mixture `15`. This only changes the mixture in the southwestern corner to another mixture number. Mixtures 5 and 15 are identical, but this sample assumes that the empty cuboid in the southwestern corner will be replaced with a control rod via the branch specification (this assumes the control rod is a square that entirely fills unit 10).

Now click the Branch button from the left toolbar. The Branch Materials window will appear (Fig. 7.1).

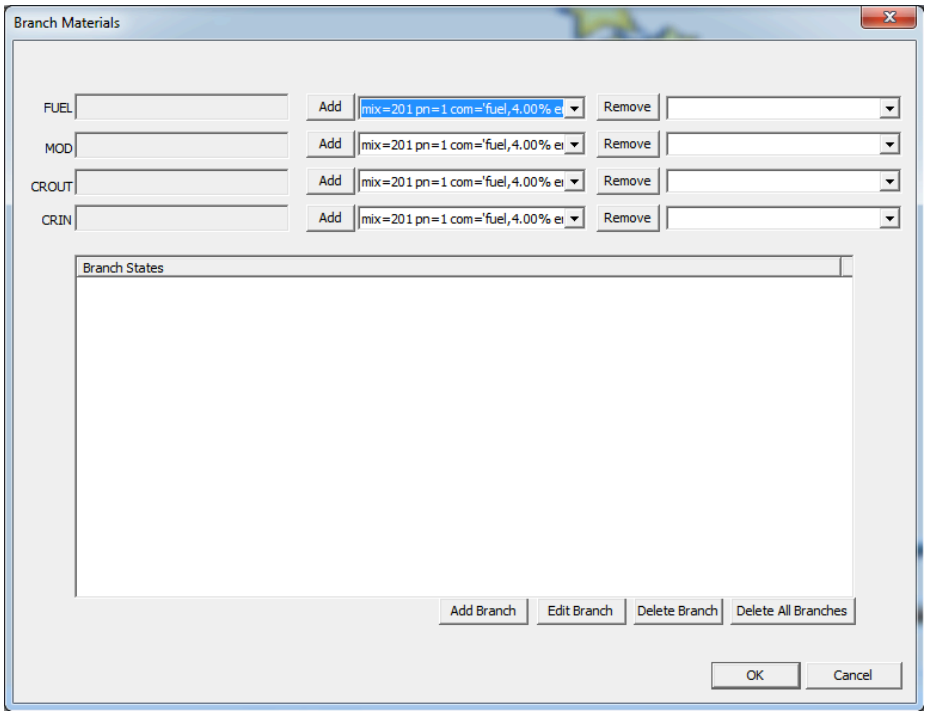


Fig. 7.1. Branch materials window.

In the Branch Materials window, first specify the materials that will be used for FUEL, MOD, CROUT, and CRIN. The CROUT and CRIN variables specify the control rod materials when the control rods are out and when they are in. SCALE/TRITON simply exchanges these materials for a controlled/uncontrolled lattice. The reason that the media for unit 10 was changed from 5 to 15 is that this example uses mixture 15 as the CROUT material. If mixture 5 was specified as the CROUT material, then mixture 5 would be exchanged for control rod material everywhere in the model (around the fuel pins in the other three corners), instead of only in the southwestern corner. You can add a material to the fuel definition by using the material dropdown (middle) and then clicking the Add button. In the Branch Materials window, specify mixtures `1` and `201` for FUEL, mixtures `5`, `15`, and `205` for MOD, mixture `15` for CROUT, and mixture `6` for CRIN. Note that mixture `15`, the CROUT mixture, is specified in CROUT and MOD. This is required for moderator perturbations (density, temperature) to occur to the CROUT material, because it acts as a normal moderator when the control rods are out. The updated Branch Materials window can be found in Fig. 7.2.

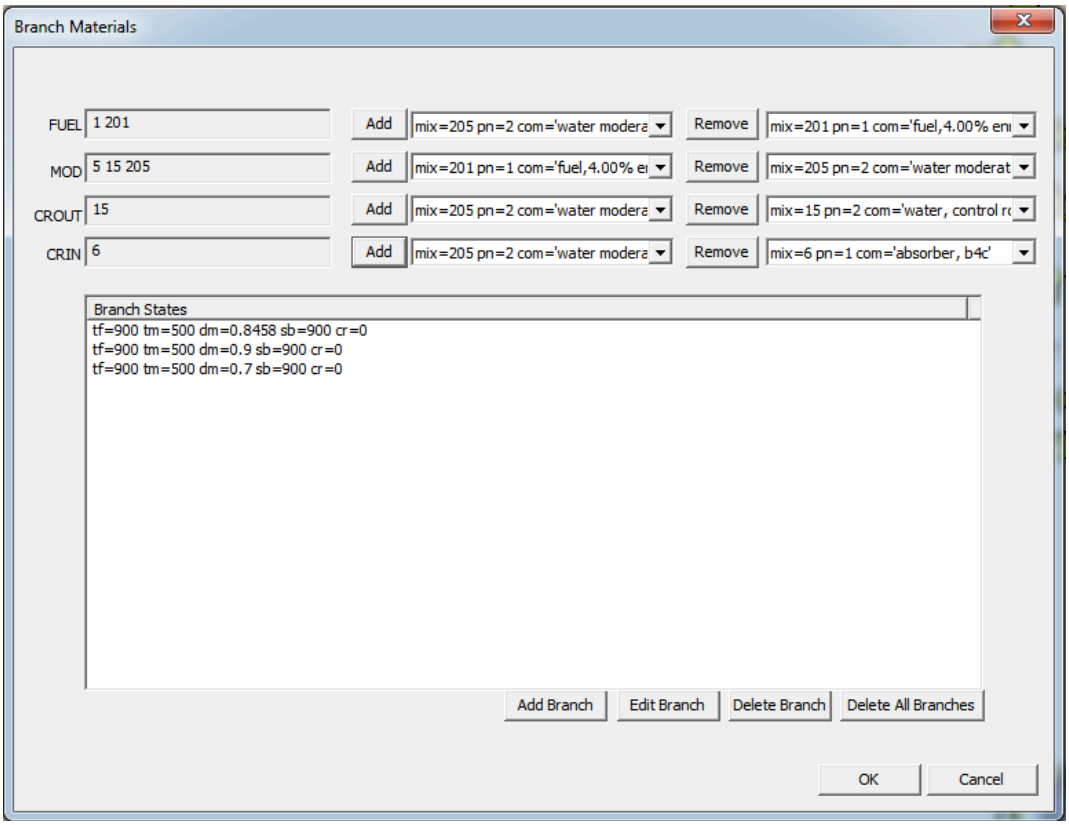


Fig. 7.2. Branch materials window with specified materials.

Now click the Add Branch button on the bottom of the form, and the Branch Mixture window will appear (Fig. 7.3).

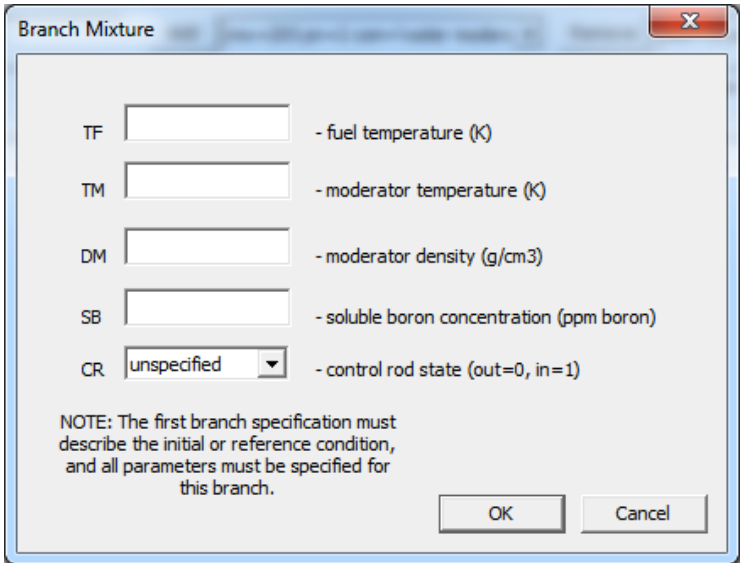


Fig. 7.3. Branch mixture window.

The first branch condition corresponds to the reference conditions: TF = `900` K, TM = `500` K, DM = `0.8458` g/cm^3, SB = `900` ppm, and CR = `out(0)`. The final Branch Mixture window for the branch 0 (nominal) conditions can be found in Fig. 7.4. If everything looks good, click OK.

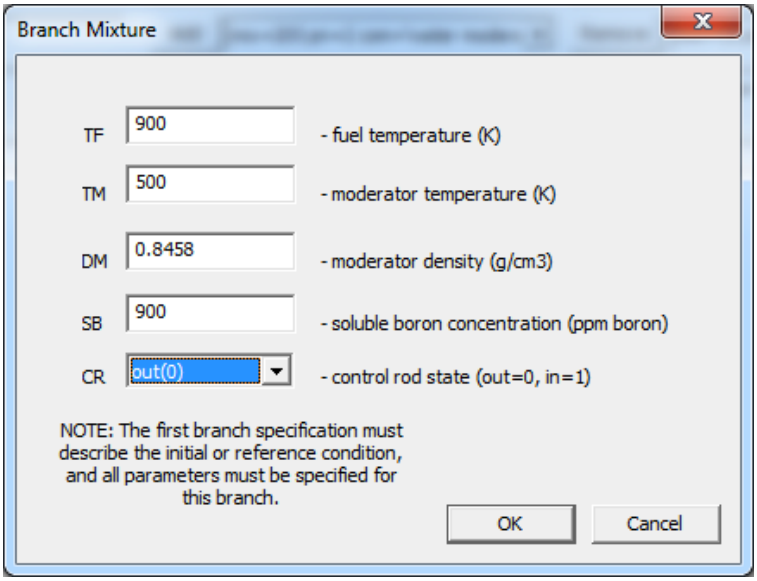


Fig. 7.4. Branch mixture window for nominal conditions.

After clicking OK on the Branch Mixture window, notice that the Branch Materials window has changed. You can now see that a branch has been added to the Branch States box. Add two more branch states by clicking Add Branch and following the same procedure. The next two branches are moderator density branches so everything but DM is identical to branch 0. Add branches for DM = `0.9` g/cm^3 and `0.7` g/cm^3. The final Branch Materials window can be found in Fig. 7.5.

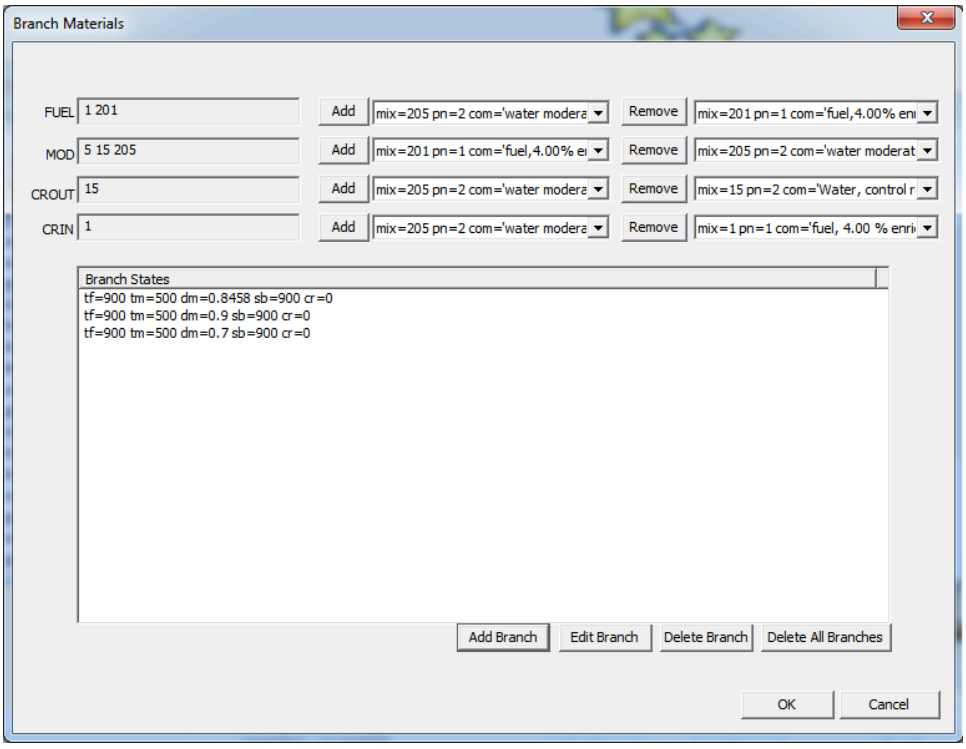


Fig. 7.5. Final branch materials window for the mini-assembly problem.

The model has three depletion steps: the first step plus two library updates between 0 and 1000 days. The model contains the nominal branch and two moderator density branches. This example has a total of nine NEWT transport calculations (three steps, each with three states); running this input file will take a significant amount of computer time on a typical PC. Before running this problem, you need to add a few other options, shown in the next section.

8. ADVANCED INPUT FOR LWR LATTICE PHYSICS

There are a few more features that users need to understand before performing lattice physics analyses. Most of these features will be applied to all lattice physics models, while one feature is primarily required for BWRs. After completion of this section, you should be able to do the following:

- use the Collapse block to collapse fine-group structures to broad-group structures
- use the Homogenize block to specify materials whose cross sections will be collapsed by the Collapse block
- use the ADF option to specify planes on which assembly discontinuity factors will be calculated
- apply user-defined Dancoff factors to a BWR model

8.1 THE COLLAPSE BLOCK

The **Collapse** block is required for generating broad-group nodal parameters for reactor core simulators. The **Collapse** block specifies energy group boundaries that SCALE/TRITON uses to collapse cross sections to a reduced group structure. By collapsing the cross sections with the flux in each mixture, SCALE/TRITON preserves the reaction rates of the fine-group solution in the broad-group solution. Generally, LWR nodal core simulators use a two-group (fast and thermal) structure. To add a user-specified energy group collapse to your problem, first enable Collapse in the NEWT logic parameters, then click **Collapse** along the left vertical toolbar (the **Collapse** button will only appear after it has been enabled in the NEWT logic parameters). To specify the typical 238G to 2G energy collapse, you will need to use the checkboxes along the left side of the **Collapse Block** window to uncheck all the group boxes excluding group `1`, `200`, and `239`. Only boxes `1`, `200`, and `239` should have checkmarks next to them, as seen in Fig. 8.1. Click OK. Users should note that if using the Parm=weight option in TRITON, a 49-group library will be used for the transport calculation, and as such, a 49-group collapse should be used rather than a 238-group collapse.

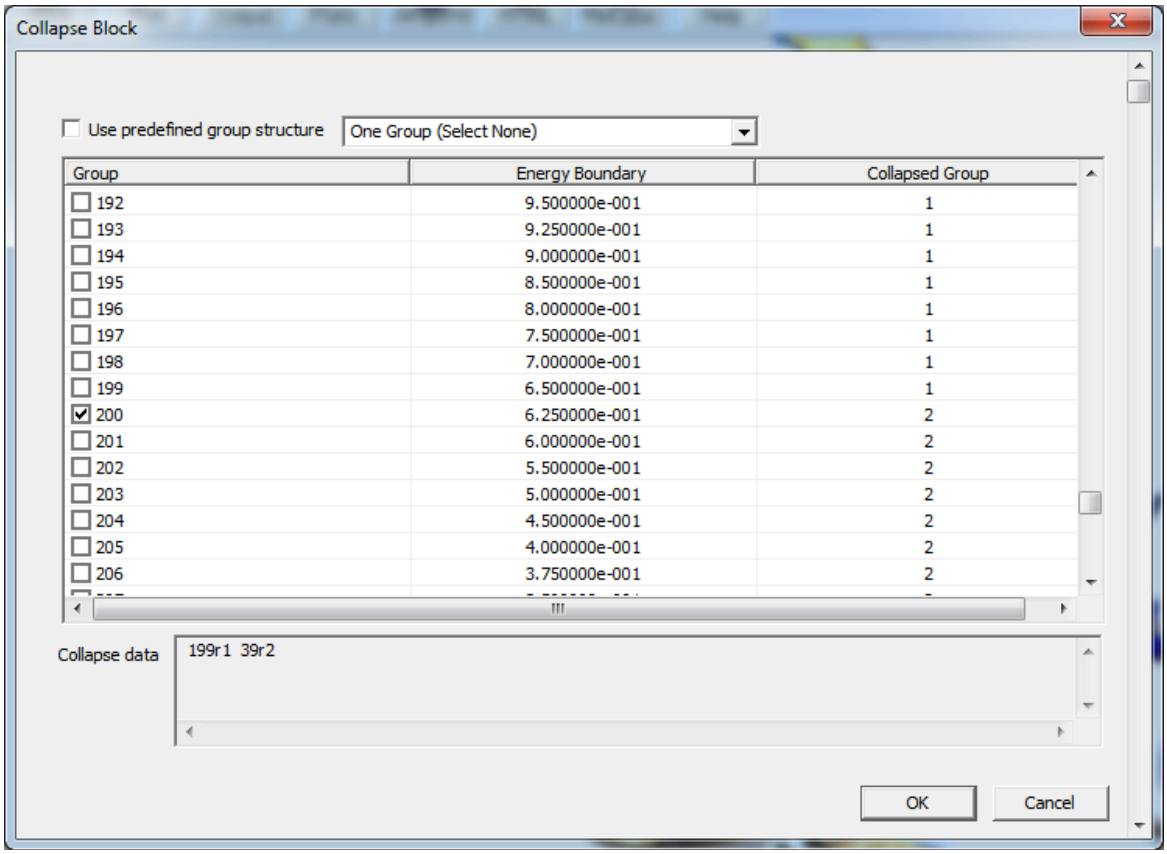


Fig. 8.1. The collapse window.

8.2 THE HOMOGENIZE BLOCK

SCALE/TRITON uses the instructions contained in the Homogenize block to homogenize cross sections over space, usually in conjunction with an energy group collapse. Using the Homogenize function, multiple mixtures can be homogenized into a single effective mixture that conserves the overall reaction rates of the detailed lattice model. The Homogenize block is required for generation of lattice physics cross sections. In the Homogenize block, the user specifies a homogenized mixture identifier, a homogenized mixture description, and the materials for which cross sections will be homogenized. When generating nodal parameters, all materials in the lattice model need to be specified in the Homogenize block to properly homogenize the cross sections over the lattice. The responsibility lies with the user to ensure that every material used in the model appears in the Homogenize block—SCALE/TRITON will not warn you if you omit a material.

For an example, you will add an Homogenize block to the mini-assembly model. If `mini1_2branches.inp` is not already open, open it in GeeWiz now. Click Homogenize along the left toolbar and the Homogenization Block window will appear. The mini-assembly problem contains five different mixtures: two fuel, one clad, and two moderator mixtures. To add these mixtures to the Homogenization Block, click Add on the Homogenization Block window. The Homogenize Record window will appear with a list of available mixtures on the left and the homogenized mixture ID and Label on the right. Leave the homogenized record ID as `500`, and change the Label to `mini2f` (meaning mini-assembly with two fuels). Now click the materials that are present in this mini-assembly problem: `1 uo2`, `201 uo2`, `3 zirc4`, `5 h2o`, and `15 h2o`. You do not need to specify the control

rod material (`6 b4c`) that will be exchanged for `15 h2o` in the model when control rods are inserted. SCALE/TRITON automatically performs this exchange in the case that a branch has control rods inserted. The final Homogenize Record window for this problem can be found in Fig. 8.2. Also, the final Homogenization Block window can be found in Fig. 8.3. Click OK on the Homogenize Record window and the Homogenization Block window when finished.

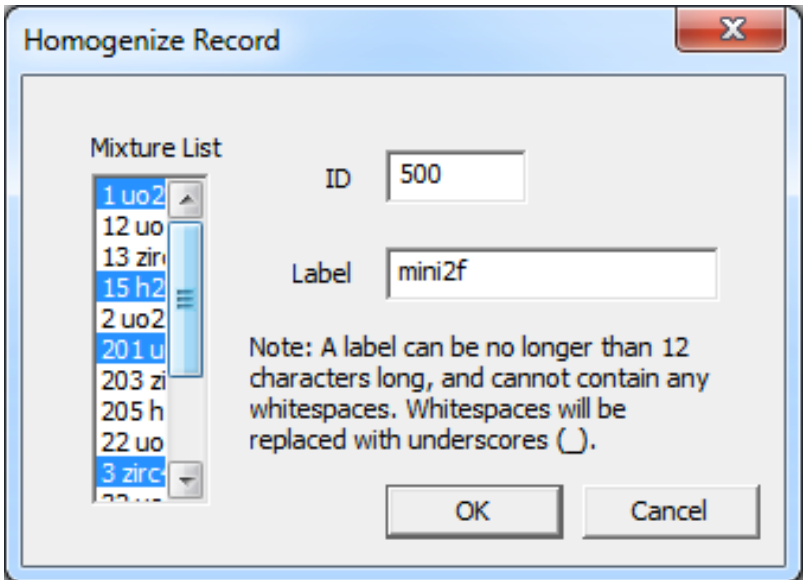


Fig. 8.2. Homogenize record window.

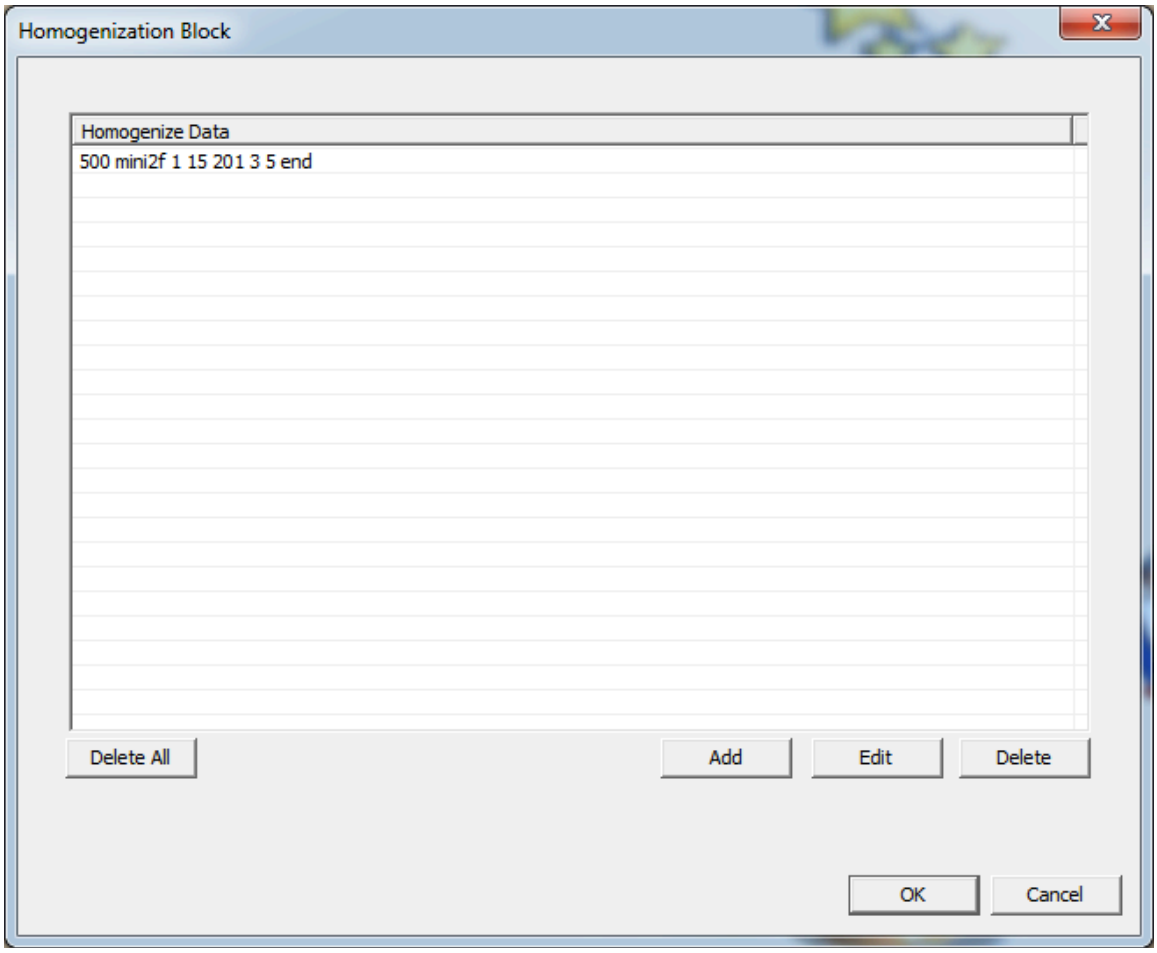


Fig. 8.3. Homogenization block window.

8.3 THE ADF BLOCK

The assembly discontinuity factor (ADF) is defined as the ratio of the heterogeneous flux along the assembly boundary to the nodal average flux. In addition to cross sections, the ADFs are passed to nodal core simulators to satisfy the boundary conditions of certain homogenized nodes. In an actual reactor, neutron flux is continuous between assemblies. In a nodal simulation, there is one homogeneous flux per node, so the flux between two adjoining nodes is discontinuous. General Equivalence Theory replaces the continuity of the homogeneous flux at the interface between two assemblies with a function of the ADFs of the two adjoining assemblies. SCALE/TRITON ADFs only have meaning in reference to homogenized regions that are specified in the Homogenize block. In order for SCALE/TRITON to calculate ADFs, the model must have a Homogenize block with associated materials for which the ADFs will be calculated.

To specify ADFs, click on the A.D. Factors button on the left toolbar and the Assembly Discontinuity Factors window will appear (Fig. 8.4). The inputs for the ADF block are ADF type (assembly or reflector), Assembly Hmog Id specified by ID and Label in the Homogenize block, Reflected Hmog Id that is specified in the Homogenize block (only for reflector problems), and the locations for which North, South, East, and West ADFs will be calculated. For a single assembly problem, the North, South, East, and West ADFs are simply the locations of the global boundary. For the mini-assembly problem, enter

`Single Assembly` for ADF Type; `500 mini2f` for the Assembly Hmog Id; and `1.4`, `-1.4`, `1.4`, `-1.4` for the North, South, East, and West ADFs (Fig. 8.4), and then click OK.

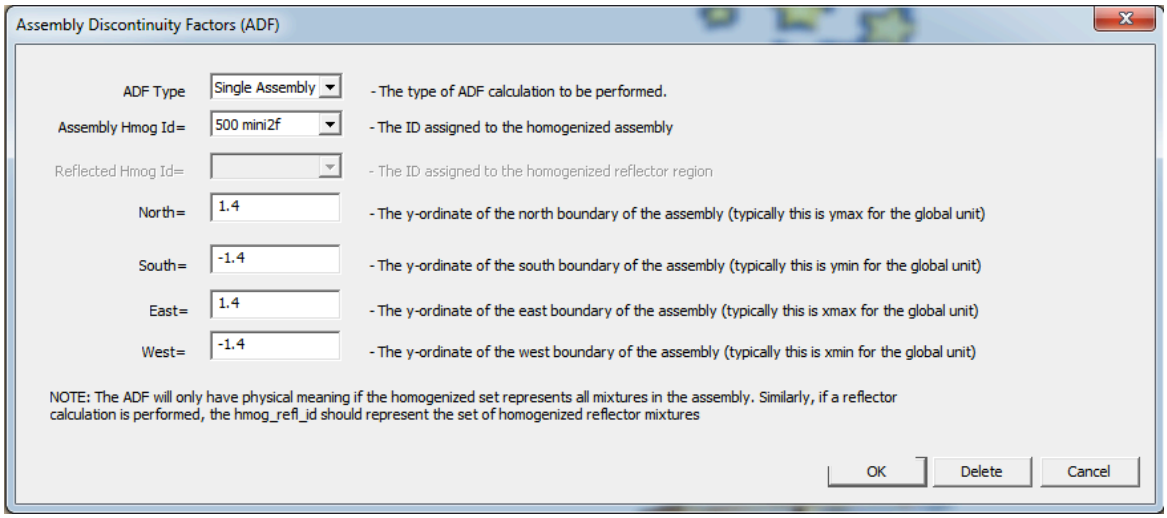


Fig. 8.4. Assembly discontinuity factors (ADF) window.

The mini-assembly model should now have Collapse, Homogenize, and ADF blocks. Execution of this input file with SCALE/TRITON will generate *xfile016* and *txtfile16* files that contain nodal data for the mini-assembly lattice.

Before running this input file, there is one more thing that you should add. By default, SCALE/TRITON does not copy the *xfile016* and *txtfile16* files back to the return directory. Using a Shell command, you can copy these files back to the return directory and rename them. Using the Shell command and SCALE-defined environment variables TMPDIR and RTNDIR, you can copy any file in the temporary directory back the runtime directory. To enter shell commands using GeeWiz, click Edit along the top of the GeeWiz window and then click Edit Shell Commands near the bottom of the dropdown. The Shell block will execute any DOS (Windows) or shell (Linux) system command that you specify before or after execution of the SCALE/TRITON sequence. To copy the *xfile016* and *txtfile16* files back to the runtime directory, enter the following commands in the After end box:

```
copy %TMPDIR%\xfile016 %RTNDIR%\mini1_2branches.x16
copy %TMPDIR%\txtfile16 %RTNDIR%\mini1_2branches.t16
```

These commands will execute after the run is complete and will copy the *xfile016* and *txtfile16* files from the temporary directory back to the return directory with the file names `mini1_2branches.x16` and `mini1_2branches.t16`, respectively. After you have entered the shell commands, you can click Run to execute the problem. Note that this problem can take a significant amount of time to complete, depending on the speed of your computer.

8.4 USER-DEFINED DANCOFF FACTORS FOR BWRS

Recent studies have shown that nonuniform lattice effects have a significant impact on the accuracy of the results (eigenvalues, collapsed cross sections, etc.) that can be obtained using standard SCALE/TRITON techniques for high-void BWR lattices. Fuel pins in the assembly corner, along the assembly edge, and

adjacent to liquid water rods in high-void BWR lattices receive significantly more neutron moderation than other fuel pins in the lattice.

The Dancoff factor (DF) provides an established means of quantifying resonance self-shielding effects between fuel pins. A high Dancoff factor (near 1.0) indicates that resonance energy neutrons can easily stream uncollided between fuel pins. A low Dancoff factor (near 0.0) indicates that most resonance energy neutrons undergo a collision when passing between lumps. The SCALE latticecell treatment assumes that a fuel pin lies in an infinite lattice of identical fuel pins with even spacing. With this assumption, an analytical expression for the Dancoff factor is employed and applied uniformly throughout the lattice [SCALE manual, Section M7.2.5.4]. For the aforementioned fuel pins, in an otherwise low-moderated lattice, the uniform Dancoff factor overestimates the actual Dancoff factor. The effect of this overestimation increases as the number of neutron energy groups is decreased. A more accurate, location-specific Dancoff factor can be calculated using the SCALE module MCDancoff, which computes Dancoff factors using Monte Carlo techniques with a simplified version of KENO-VI [SCALE manual, Section M24].

The analytically computed Dancoff factor can be overridden using the `dan2pitch` option in the `latticecell` input to apply user-defined Dancoff factors. However, in SCALE 6.0, the `dan2pitch` parameter only applies to static calculations in TRITON, without depletion steps. SCALE 6.1 will have the capability for a user to compute and apply user-defined Dancoff factors to TRITON depletion calculations, with or without branches. The Dancoff factor changes as a function of moderator density so the user-defined Dancoff factor should change for moderator density branches.

Figure 8.5 shows a Dancoff factor map for an 80% void BWR lattice. For this study, Dancoff factors for gadolinium-bearing pins were not calculated. Note that the standard latticecell Dancoff factor computed by SCALE for this 80% void lattice is 0.7063, where every fuel pin in this model actually has a lower Dancoff factor due to the nonuniformity of the lattice.

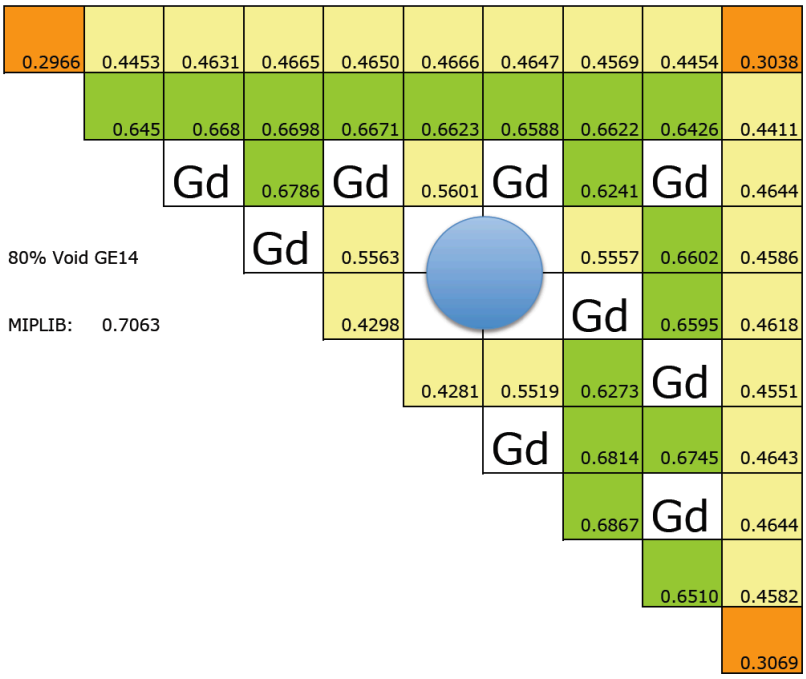


Fig. 8.5. Dancoff factor map for 80% void GE14 lattice.

After appropriate Dancoff factors are obtained, the user must edit the T-NEWT or T-DEPL input file and add separate CENTRM cross-section processing blocks for all materials for which special Dancoff factors will be used. Ideally, every fuel pin would have a separate Dancoff factor, but good results have been obtained by grouping similar enrichment and similar Dancoff factor fuel pins along the lattice edge, in the lattice corner, and near water rods. In each CENTRM cross-section processing block, the CENTRM option `dan2pitch(N)=DF` should be used, where `N` is the fuel material number and `DF` is the user-specified Dancoff factor.

After new DFs are applied to the CENTRM calculation, SCALE/TRITON can be run normally. A BWR lattice test case was run using a void fraction of 80% in the fuel channel and solid saturated water in the bypass and water rods. This test case used the TRITON *PARM=WEIGHT* option to generate a problem-dependent 49-group cross-section library prior to the calculation. The standard latticecell calculated Dancoff factors were used for one simulation while user-specified Dancoff factors were used for the other. Throughout most of the depletion range, the difference in k_{inf} was 800–1100 pcm, as plotted in Fig. 8.6.

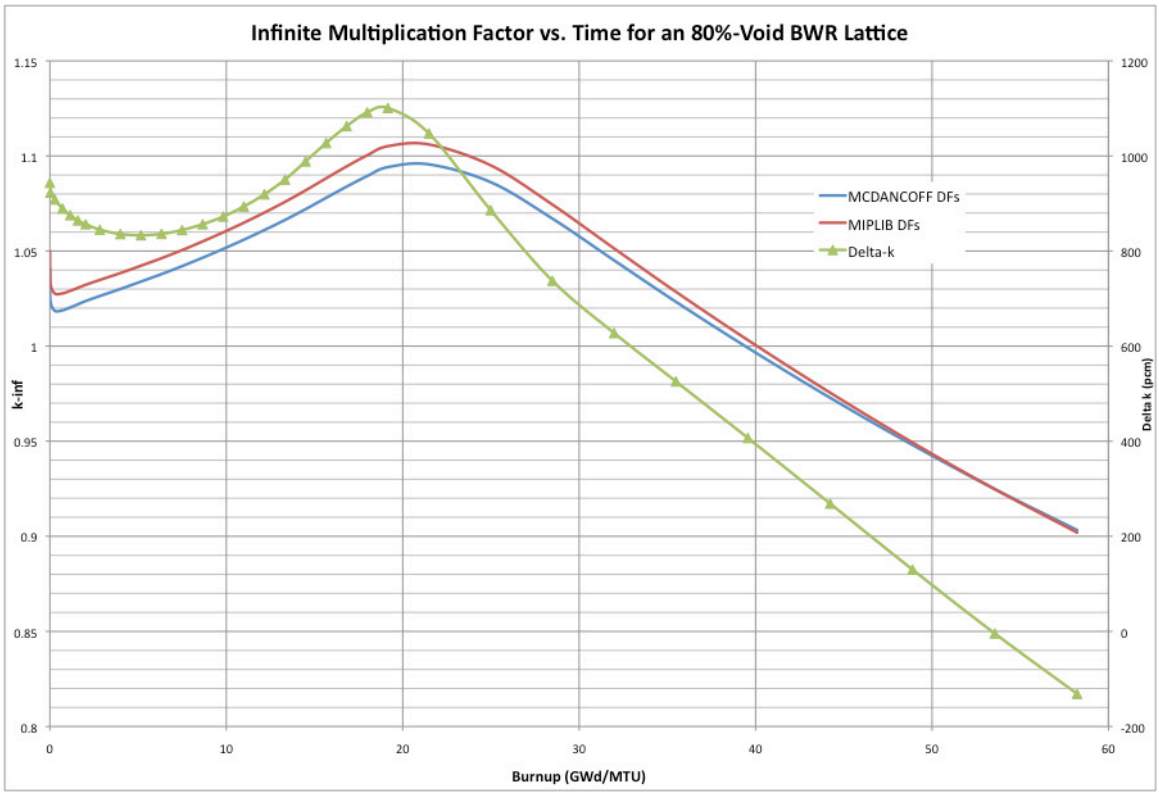


Fig. 8.6. Comparison of standard and user-specified Dancoff factors for a GE14 lattice.

Users can calculate accurate DFs using SCALE/MCDancoff. MCDancoff is a modified version of SCALE/KENO-VI that runs a fixed-source simulation to calculate Dancoff factors. Currently, there is not a good way to treat Dancoff factors of multiringed gadolinium-bearing pins; the current best practice it to use standard SCALE XS processing technics that were previously presented in this document. To use SCALE/MCDancoff, you should convert your TRITON/NEWT model to a SCALE/KENO-VI model.

For further directions on using SCALE/MCDancoff and converting a NEWT model to a KENO-VI model, see Appendix B.

Applying user-defined Dancoff factors to your model is relatively straightforward, but first you must decide which fuel pins need special DFs. Generally, it has been found that fuel pins have similar DFs in the lattice corners, along the lattice edge, and in the lattice interior (Fig. 8.7). The true DFs for fuel pins in the lattice corners and along the lattice edges tend to differ from the CENTRM infinite-lattice Dancoff factor by a relatively large amount. Instead of running a separate CENTRM latticecell calculation with a user-defined Dancoff factor for every pin, you should group the fuel pin locations together in some way. There are generally two levels of fuel pin grouping that are recommended for BWRs: (1) fuel enrichment and (2) Dancoff factor. For all BWR and PWR fuel assemblies there should be a corresponding CENTRM latticecell for each fuel enrichment. Within a same-enrichment group of fuel pins, you should divide into separate subgroups of fuel pins that are located in the lattice corner, on the lattice edge, and in the lattice interior. For example, if the lattice you are modeling contains a 2.00% enriched fuel pin in the lattice corner and a 2.00% enriched fuel pin on the lattice edge, you need separate CENTRM latticecells with user-specified DFs for both the corner and edge pins.

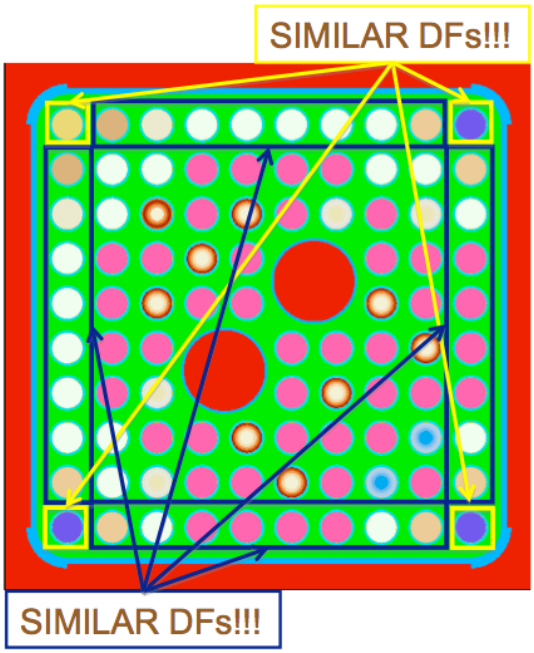


Fig. 8.7. Fuel pin locations with similar Dancoff factors.

To add user-specified Dancoff factors to your model without branches, you can use GeeWiz or the text-based input. To use GeeWiz to add user-specified Dancoff factors to a CENTRM calculation, first click Cell Data from the left vertical toolbar. The Unit Cell Data window will appear (Fig. 8.8). You should now click the edit centrm button along the top horizontal toolbar of the Unit Cell Data window. After clicking edit centrm, the Centrm Data window will appear. In the right bottom portion of the window, you will see a box named Dan2Pitch Factor. In the Dan2Pitch Factor box, you can select the Mixture (fuel) and specify a user-defined Dancoff Factor for that mixture.

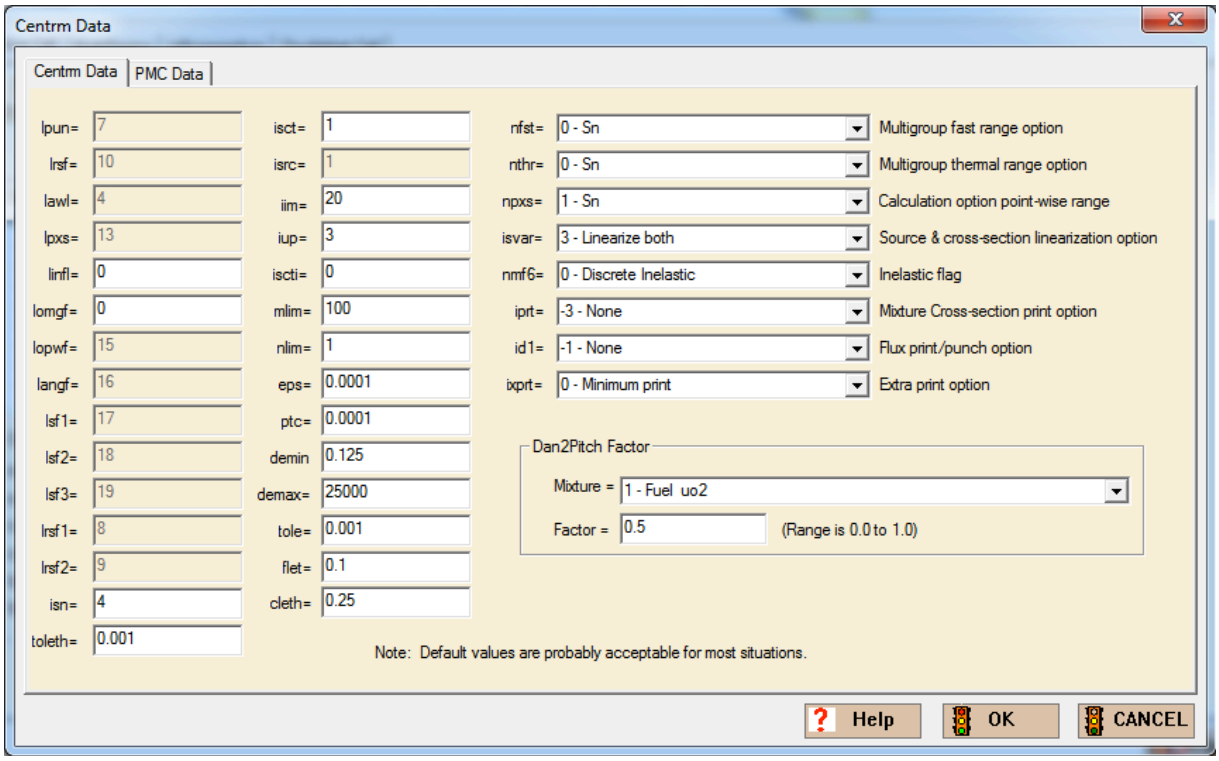


Fig. 8.8. Centrm data window.

If the problem of interest is a depletion case or has branches and requires user-defined Dancoff factors, you will have to use SCALE6.1 and edit the text input file to apply user-specified DFs that are different from the DFs in the nominal (branch 0) state. This occurs in BWR lattices that are depleted using a certain void fraction (moderator density) and have moderator density branches that differ from the nominal state. Dancoff factors for depletion calculations and DFs for different branches are both new features in SCALE 6.1.

An example of the new SCALE 6.1 Dancoff factor features being applied to the branches of a SCALE/TRITON input file is shown here:

```
read branch
  define fuel   $1fl240 $1fl280 $1fl320 $1fl360 $1fl395 $1fl440 $1fl490
                $fuel440 $fuel240 $fuel280 $fuel320 $fuel360 $fuel395
                $fuel490 $1f490gd700 $1f440gd700 $1f440gd800 $f490gd700
                $f440gd700 $f440gd800 $1fl440E $fuel440E  $1fl490E
                $fuel490E                end
  define mod    $mod                                 end
  define crin   $cbpelin $cbcladin $cbwingin         end
  define crout  $cbout                               end
  define d2pset 100   140 0.360  280 0.360  320 0.560 360 0.560 395 0.560
                      449 0.560  499 0.560 end
  define d2pset 80    140 0.300  280 0.300  320 0.460 360 0.460 395 0.460
                      449 0.460  499 0.460 end
  define d2pset 40    140 0.230  280 0.230  320 0.340 360 0.340 395 0.340
                      449 0.340  499 0.340 end
  define d2pset  1    140 0.175  280 0.175  320 0.260 360 0.260 395 0.260
                      449 0.260  499 0.260 end
```

```
    dm=0.7397    tf=863.15   tm=559.15  sb=1   cr=0     d2p=  1      end
    dm=0.458432  tf=863.15   tm=559.15  sb=1   cr=0     d2p= 40      end
    dm=0.177164  tf=863.15   tm=559.15  sb=1   cr=0     d2p= 80      end
    dm=0.03653   tf=863.15   tm=559.15  sb=1   cr=0     d2p=100      end
end branch
```

The branch block begins with a `read branch` statement, followed by definitions for fuel, mod, crin, and crout. Users can then define a "Dancoff factor set" (highlighted in yellow). The Dancoff factor set is defined using the following syntax:

```
define d2pset ID  Mix_n DF_n end.
```

where

ID	=	Dancoff factor set identifier—ID can be any number from 1 to 100 (-1 and 0 have special meanings),
Mix_n	=	fuel mixture number,
DF_n	=	Dancoff factor for fuel *n*.

Mix_n and DF_n are defined pairs that can be specified as many times as needed. Mix_n should be the fuel mixture number that is used in the Cell Data latticecell calculation. Different Dancoff factor sets can be defined for different moderator densities. These DF sets are then applied to branch conditions by using `d2p`=ID (highlighted in blue), in addition to the other conditions for a given branch.

8.5 GENERATING REFLECTOR CROSS SECTIONS FOR NODAL SIMULATORS USING SCALE/TRITON

In addition to nodal data for fuel bundles, nodal simulators also require data for the boundary of the problem in both the radial and axial directions. SCALE/TRITON has the capability to generate data needed for nodal simulator reflector cross sections. In order to generate reflector cross sections using SCALE/TRITON, a typical fuel bundle model is used with additional regions(s). It is recommended that a somewhat representative model be used (i.e., a PWR fuel bundle model should not be used to generate reflector cross sections for a BWR). However, the process to generate reflector cross sections is similar for both PWRs and BWRs.

The reflector model should be developed from a typical fuel bundle with an additional region or regions added to the east side of the model. A region that is at least one fuel bundle in width is typically added. Reflective boundary conditions are used for the north, west, and south boundaries, and a vacuum boundary condition is used for the east boundary, as illustrated in Fig. 8.9, where the solid red lines represent reflective boundary conditions and the dashed line on the east boundary represents a vacuum boundary condition.

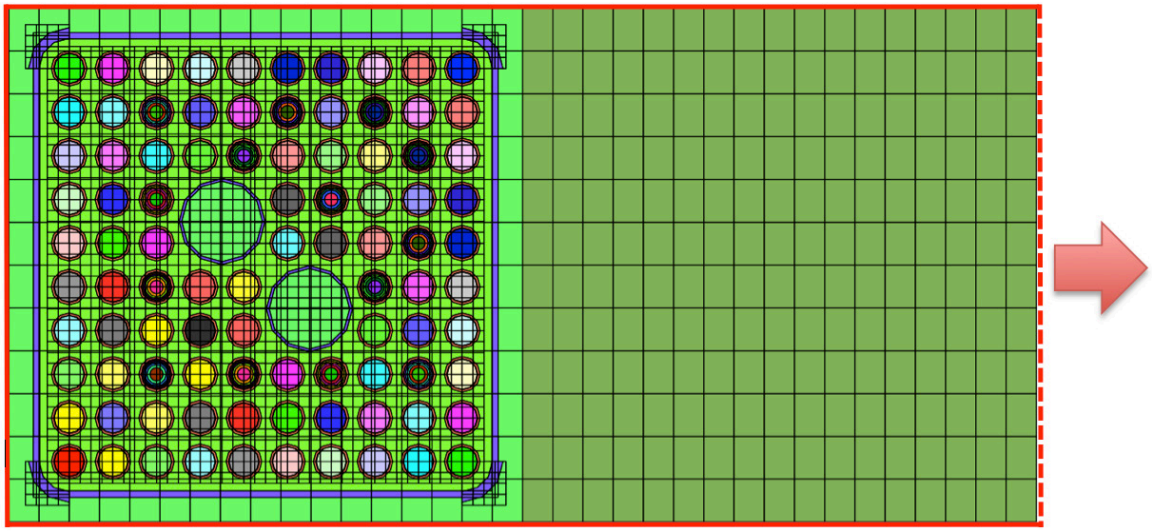


Fig. 8.9. SCALE/NEWT representation of a model used to generate reflector cross sections.

Because there is no fissile material in the reflector portion of the model, the model does not require depletion like typical fuel bundle models. One of the main difficulties in generating reflector cross sections is the definition of the problem. The model represented in Fig. 8.9 is similar to the geometric configuration of a fuel bundle on the core radial periphery. However, for core axial periphery, the physical configuration is that of a fuel bundle followed axially by a region of structural material and water, as illustrated on the left in Fig. 8.10. However, because NEWT is a 2-D code, the fuel bundle and axial reflector must be approximated by placing the water and structural material to the east side of the fuel bundle model, as seen in Fig. 8.9 and on the right side of Fig. 8.10.

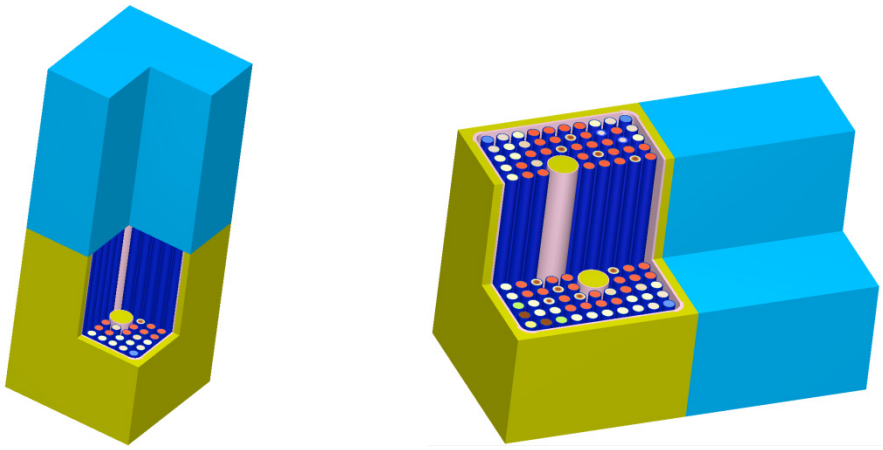


Fig. 8.10. SCALE/KENO-3D representation of the true top axial reflector configuration (left) and the approximated configuration in NEWT (right).

In addition to approximation of the geometry due to 2-D geometry limitations, the definition of the materials in the axial reflector region can be somewhat difficult. The physical structure of the materials above and below the active fuel regions is typically approximated by homogenizing structural material with water. Accurately defining the amount of structural material is difficult because the geometry is often complex and the information on the geometry can be somewhat difficult to find. The bottom axial

reflector might contain some material from a core support plate and a fuel bundle inlet nozzle. The top axial reflector might contain a region of fuel cladding material filled with gas to represent the fuel plenum, and other structural materials such as the bundle handle. It is standard practice to use a homogenized mixture of water and structural material for these regions.

8.5.1 Generating Homogenized Materials for Reflector Cross-Section Generation

In generating the reflector materials, the primary goal is to estimate the mass of the structural materials as accurately as possible. If the volume of various structural materials above and below the fuel bundle is unknown, a reasonable approximation is to assume that the fuel clad material (filled with void) and the other structure materials (water rods and channel box) extend the width of one fuel bundle. Control blades are not typically modeled for reflector cross sections, so they can be removed from the model. In order to generate a homogenized material for a BWR fuel bundle, the volume of four different regions should be calculated: the in-channel moderator, the out-channel moderator, the plenum gas volume (fuel volume), and the structure volume.

The easiest way to calculate these mixture volumes is to use the fuel bundle model that will be used as the neutron source for the reflector problem. Before any reflector geometry has been added, run the fuel bundle model using the NEWT parameter "run=no". By setting the run parameter to "no" in the NEWT parameter block, the model will run through all calculations required for the transport calculations, but will terminate without running the transport calculation. Then search for "Mixture Volumes" in the output file; below these keywords, a table of various mixture volumes in the NEWT model will be listed in units of cubic centimeters per unit length.

Using the previously developed BWR fuel bundle problem (Sect. 5.6.4), run the model using "run=no" in the NEWT parameters and then search for the mixture volumes table in the output file. For this model, the mixture volumes table has been reproduced below:

```
Mixture       Volume        Volume
    ID     (cc/unit length)  Fraction
     0       0.00000E+00      0.00000E+00
     1       5.73341E+01      2.41753E-01
     3       3.68888E+01      1.55544E-01
     5       7.01110E+01      2.95627E-01
     7       7.28262E+01      3.07076E-01
 Total       2.37160E+02      1.00000E+00
```

Using the information in the mixture volume table, there are a number of ways to construct a homogenized mixture. The most straightforward method is to simply use the volume fractions for each mixture and the actual mixtures used in the input file. Mixture 1 is the fuel mixture, so it should be changed to a mixture that is typical for a plenum; in this example, helium is used. Also, it is easiest to construct this particular mixture by using the text-based input, rather than GeeWiz. Below, the five mixtures that will be homogenized have been listed in the SCALE text-based input format:

```
he          1 den=2.2218E-4 1.0000  900  end
zirc4       3               1       540  end
h2o         5 den=0.8458    1       500  end
boron       5 den=0.8458    0.0009  500  end
h2o         7 den=1         1       300  end .
```

Currently, these mixtures all have different mixture numbers, and different temperatures, but they should all be changed to the same mixture number (mixture 100 in the example below) and same temperature

(the moderator temperature). The current volume fraction of each composition should be multiplied by the volume fraction specified in the mixture volume table from the output file:

```
he        100 den=2.2218E-4 2.41753E-01  500  end
zirc4     100               1.55544E-01  500  end
h2o       100 den=0.8458    2.95627E-01  500  end
boron     100 den=0.8458    2.66064E-04  500  end
h2o       100 den=1         3.07076E-01  500  end  .
```

This mixture specification can then be used as the homogenized mixture for an axial reflector. The one complication to remember is that the density of the water in the axial reflector can change for BWR and PWRs. For a PWR, the water density changes with temperature, and in a BWR the water density changes as a function of axial position. In a BWR, the bottom axial reflector water will have an in-channel moderator density corresponding to saturated liquid, however, for the top axial reflector, the in-channel moderator density corresponds to the exit void fraction.

Material specification is easier for radial reflectors. No homogenization or major approximation is typically needed, but the additional reflector region might require extra regions such as a baffle or core barrel. The geometry specification for radial reflectors can be more complicated, but the materials should be easier to define than axial reflectors. The following sections describe how to set up SCALE/TRITON reflector models for both a radial reflector and an axial reflector.

8.5.2 Adding a Radial Reflector to the BWR Fuel Bundle Model

Using the previously generated BWR fuel bundle model (Sect. 5.6.4), add an additional region on the east side of the model that will serve as a radial reflector. There is a core barrel separating the bypass and downcomer regions. This cylindrical barrel has an inner radius (R) of 260 cm and a thickness of 3 cm and is composed of stainless steel 304L. From the center of the core, there are 15 fuel bundles to the core edge (Y), and one flat side of the core contains 14 fuel bundles (W). The aforementioned dimensions are labeled in Fig. 8.11. The difficulty in generating the radial reflector model is adequately modeling the core barrel and any other structure that is cylindrical. To generate a reflector model that corresponds to the conditions that the "average" bundle on the periphery experiences, the average distance from the edge of the fuel bundles to these cylindrical surfaces should be calculated. The distance from the center of the reactor to the edge of the resulting flat surface is given by H in Fig. 8.11. To calculate H, which is the average distance to the cylindrical surface, for a given flat-side width (W) and structure radius (R), use the following equation:

$$H = \frac{W^3}{6R^2\left(2\sin^{-1}\left[\frac{W}{2R}\right]-\sin\left[2\sin^{-1}\left[\frac{W}{2R}\right]\right]\right)}.$$

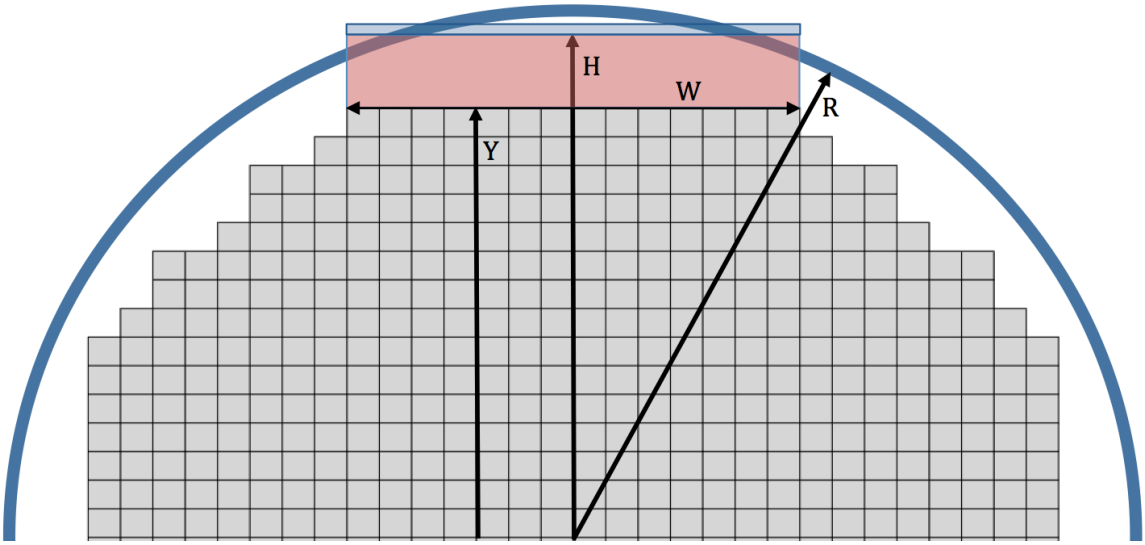


Fig. 8.11. Visual representation of the dimensions and procedure used to generate radial reflector dimensions.

The distance from the edge of the fuel bundle to the inner surface of the cylindrical structure is given by H–Y. Use the bundle dimensions for the BWR assembly model in Sect. 5.6.4, Y=231.0 cm, W=215.6 cm, and from the previous equation, H=246.0. The distance from the edge of the fuel bundle to the inner surface of the cylindrical structure is 15.0 cm (H–Y = 246.0 – 231.0 = 15.0).

To construct the reflector model, first **Open** the problem named `assembly1_wr_box.inp` from Sect. 5.6.4 and **Save as** `assembly1_wr_box_radref.inp`. Now add an additional water composition. The same mixture as the standard out-channel water cannot be used for reflector problems. **Copy** mixture number `7` to mixture number `102`. The mixture numbers are not important; any mixture number can be used as long as it is not used elsewhere in the model. For the core barrel stainless steel, the previously generated mixture for SS304L will be used. After successfully adding the radial reflector water mixture, add it to the list of NEWT materials. Click on the **Materials** button along the left vertical toolbar and **Add Mixture** `102` with a scattering order of PN=`2` (Fig. 8.12).

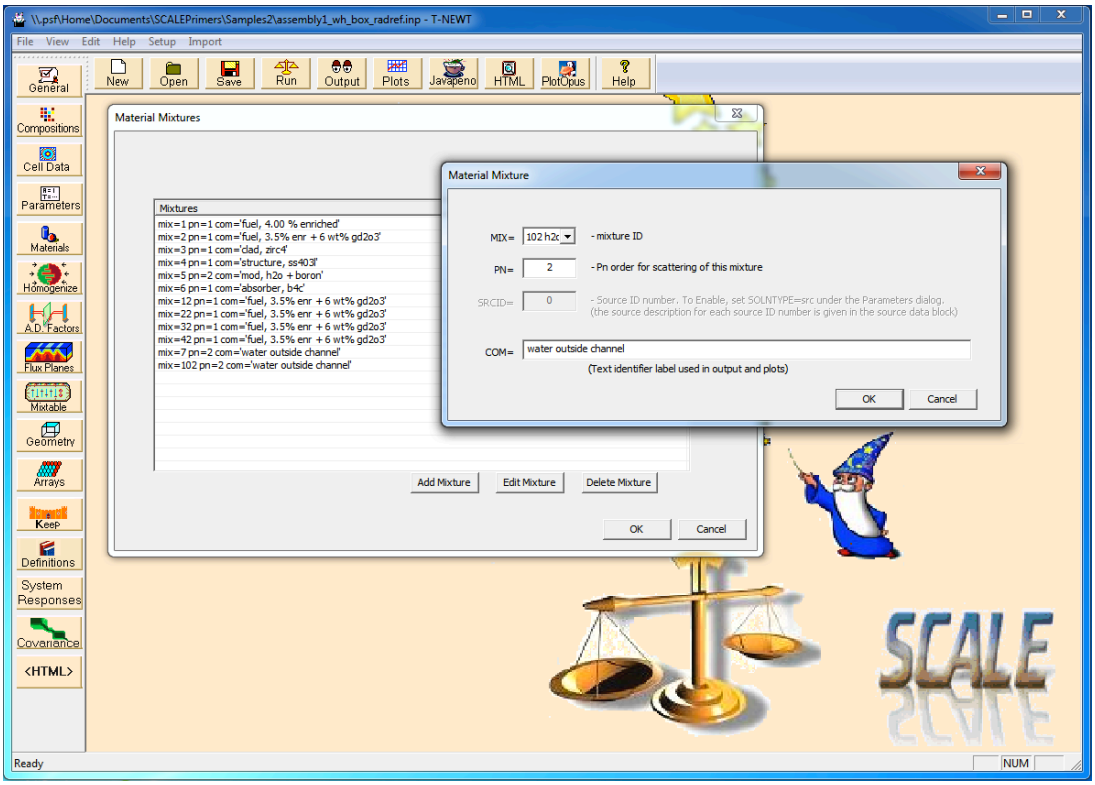


Fig. 8.12. Radial reflector example NEWT materials.

Now open the geometry form by clicking **Geometry** from the left vertical toolbar. Navigate to the global unit definition. Add an additional cuboid the to the global unit definition that will serve as the new boundary. There is no need to change the current fuel bundle boundary (cuboid 10); simply add an additional larger cuboid to bound the additional geometry. Click **Cuboid** from the right toolbar and add a cuboid with a label of `20`, and boundaries of `25.7`, `-7.7`, `7.7`, and `-7.7`. The positive x-dimension is now 25.7 cm, which includes the 15 cm from the fuel bundle boundary to the inner surface of the cylindrical structure, plus 3 cm for the thickness of the cylinder (Fig. 8.13).

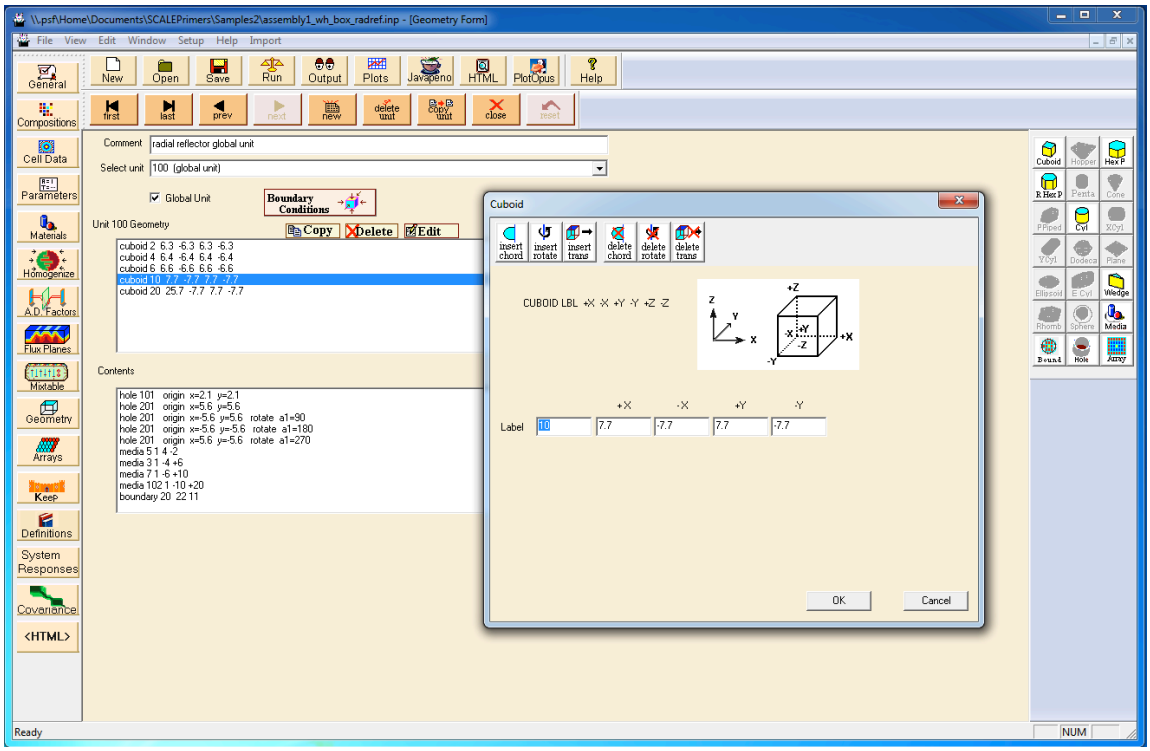


Fig. 8.13. Radial reflector example global unit definition.

Once the cuboid that will serve as the global unit boundary has been added, add a media specification for the new global unit, which should be outside body 10 and inside body 20 and filled with media 102 (the newly added out-channel water), as seen in Fig. 8.14. Then change the **Boundary Conditions** so that body 102 is the global unit boundary. Also change the global unit grid spacing to `22` in the x-direction and `11` in the y-direction. Additional grid spacing is needed in the x-direction because the model has more than doubled in size. Also change the global unit boundary so that the x-boundary has a `Vacuum` condition and other boundary conditions are `Mirror` (Fig. 8.14).

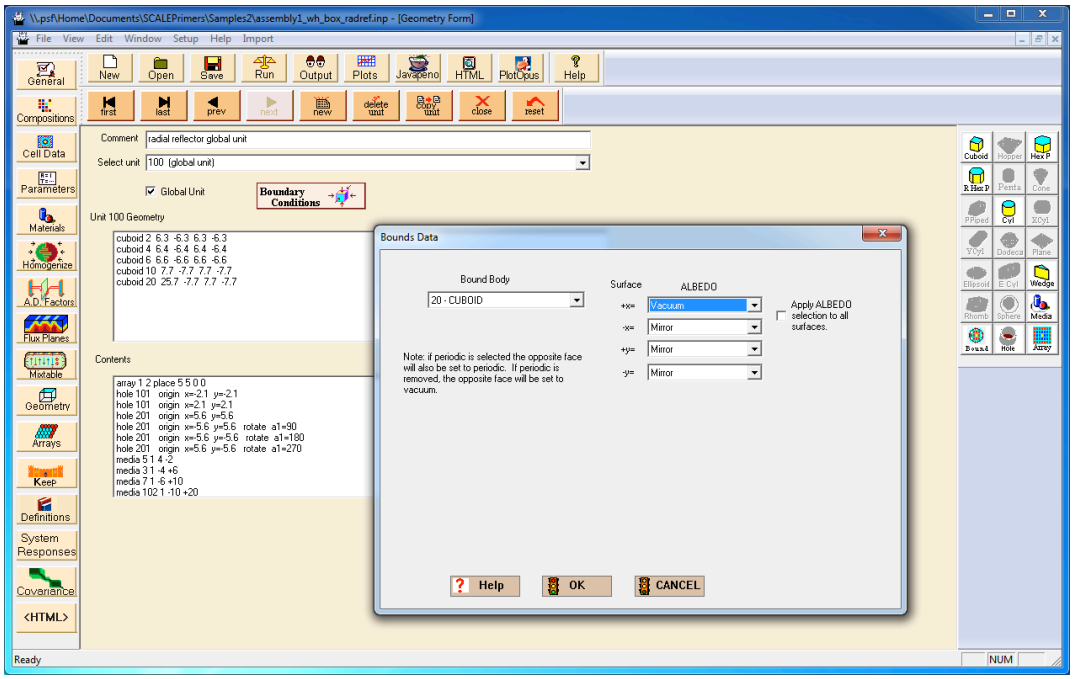


Fig. 8.14. Radial reflector example bounds data.

Before moving forward, it is a good idea to check that everything looks okay in the model. Add **Parm=check** to the problem parameters, then **Run** the model. Once the run is complete, open the output file to make sure there are no errors, then open `assembly1_wr_box_radref.newtmatl.ps` in GhostView. The model should have an additional block of water on the east side of the assembly as shown in Fig. 8.15).

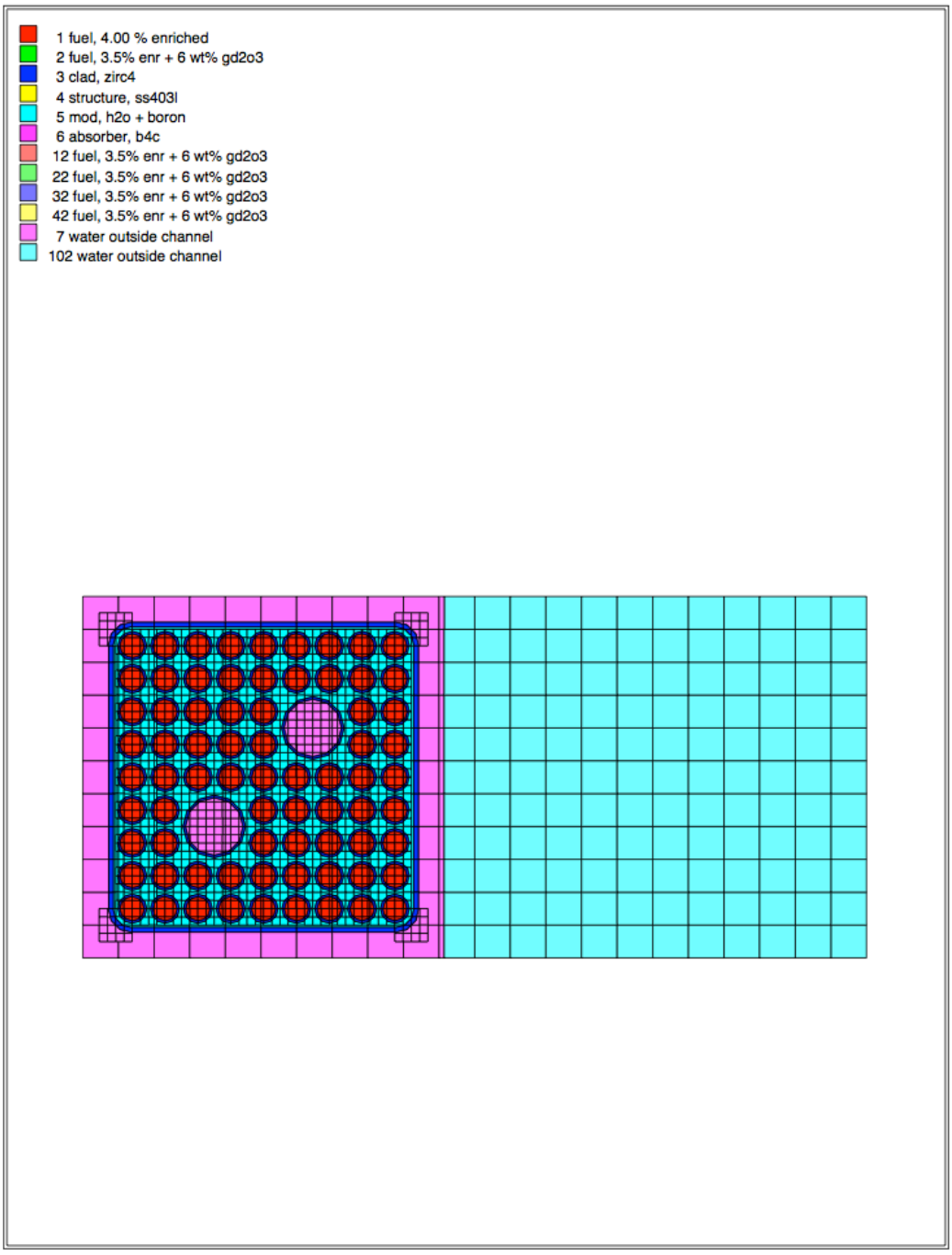


Fig. 8.15. SCALE/NEWT representation of the radial reflector model without the SS304L structure.

If the model appears correct and there are no errors noted in the output file, add the SS304L cylindrical structure. There are multiple ways to add the SS304L block to the model, but it is typically easiest to add it as a separate unit, and then place in the global unit using a hole. Using this technique, it will be easier to adapt this model to suit a different reactor type. With the model open in GeeWiz, click **Geometry** to open the geometry form. Then click **New** to add a new unit, give the unit a descriptive comment, and use `300` for the unit number. Add a cuboid whose label is `10`, and extends from `3.0` to `0` in the

x-dimension, and `7.7` to `-7.7` in the y-dimension. Then fill the unit with `media 4` (SS304L). The final **Geometry Form** for unit 300 can be found in Fig. 8.16.

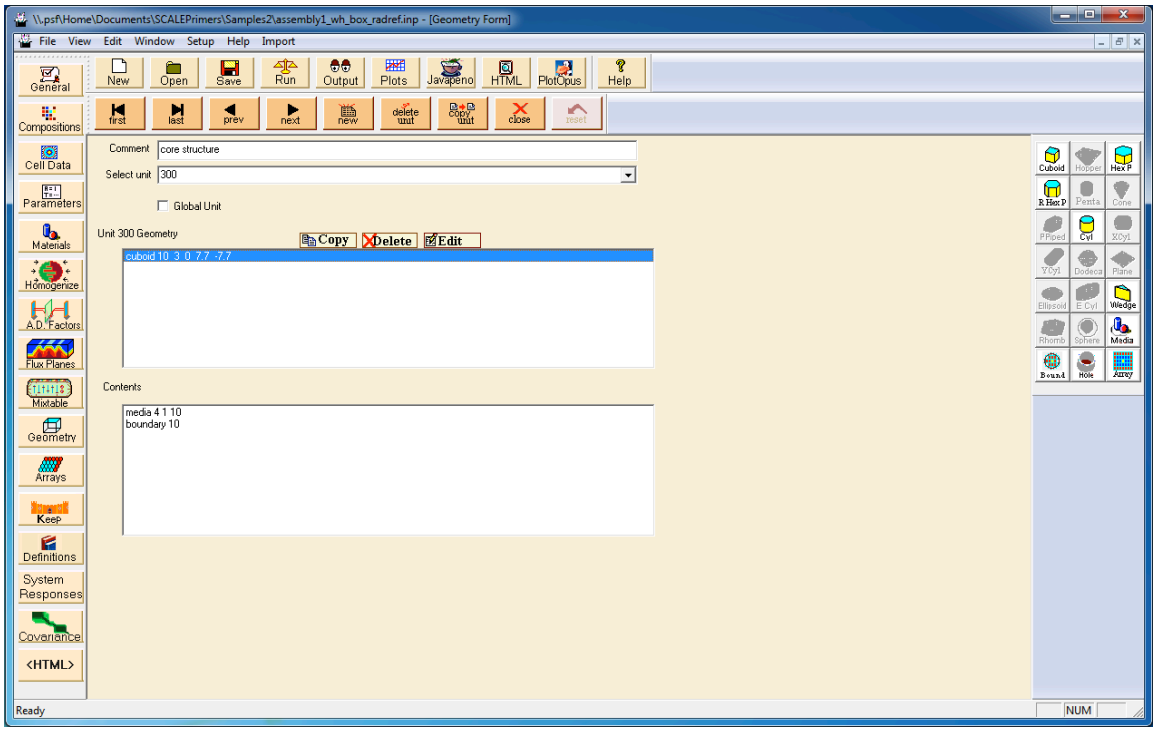


Fig. 8.16. Geometry form for the SS304L cylindrical structure in the radial reflector.

Now insert unit 300 as a hole into the global unit at a position of x=`22.7` (Fig. 8.17).

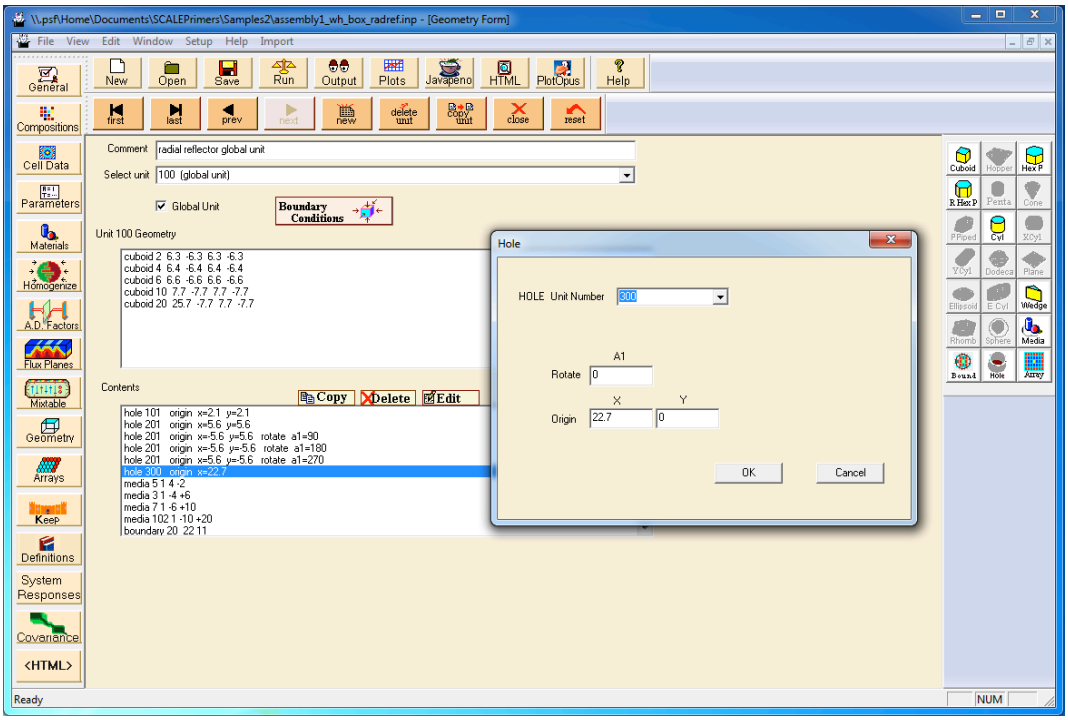


Fig. 8.17. Final global unit geometry from for the radial reflector model.

Before the radial reflector model is complete, some additional features need to be added. First click **Collapse** to collapse the data into two groups (outlined in Sect 8.1). Then click **Homogenize** to specify the needed homogenized regions. The homogenize function is outlined in Sect. 8.2, but this application of the homogenize block is slightly more complex than the example in Sect. 8.2. Two homogenized regions will now be specified; one region contains all the fuel bundle mixture numbers (`1`, `3`, `5`, and `7`), and the other contains all the reflector mixture numbers (`4` and `102`), as seen in Fig. 8.18. The homogenize ID can be any number that is not a mixture number in the problem and not another homogenize ID; this examples uses `998` for the reflector region and `999` for the fuel assembly. The final **Homogenize Block** dialog box is shown in Fig. 8.18.

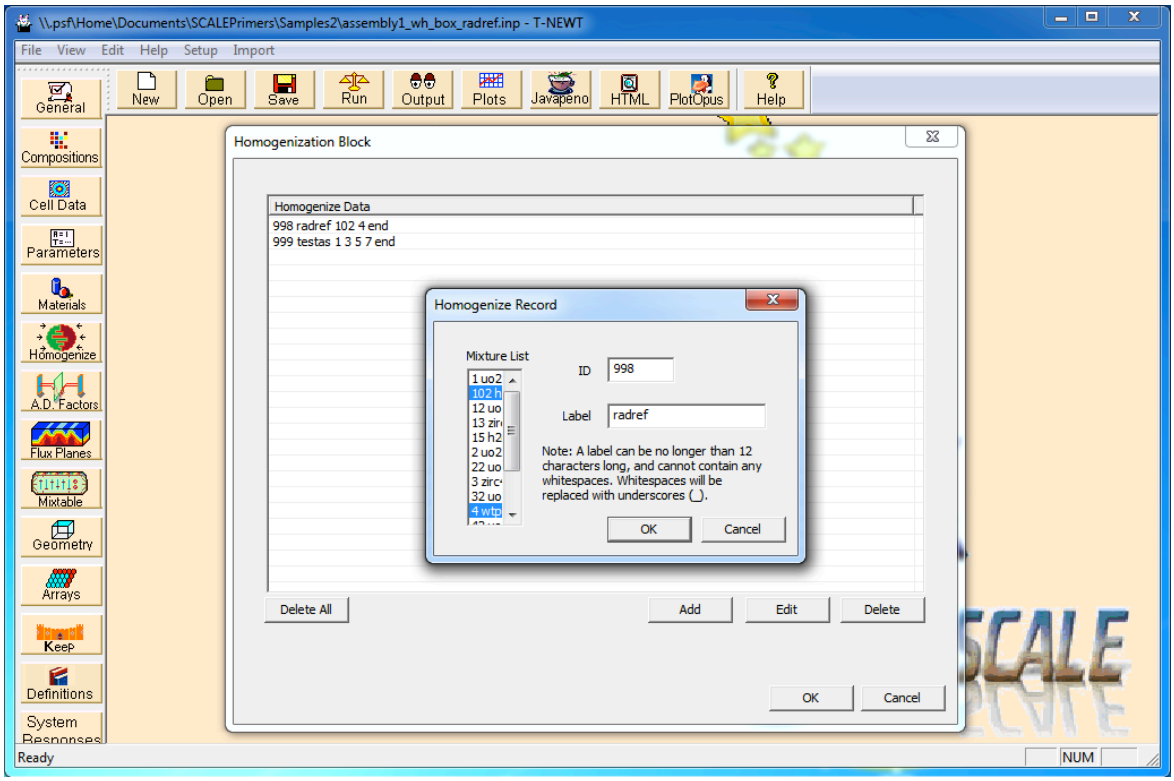


Fig. 8.18. Final Homogenize Block dialog box for the radial reflector example.

Now add assembly discontinuity factors to the model by clicking the **A.D. Factors** button along the left vertical toolbar. The **Assembly Discontinuity Factors (ADF)** dialog box will appear. Use `Reflected Assembly` for **ADF Type**, `999` for the **Assembly Hmog Id**, `998` for the **Reflected Hmog Id**, and `7.7` for the **East ADF** (Fig. 8.19).

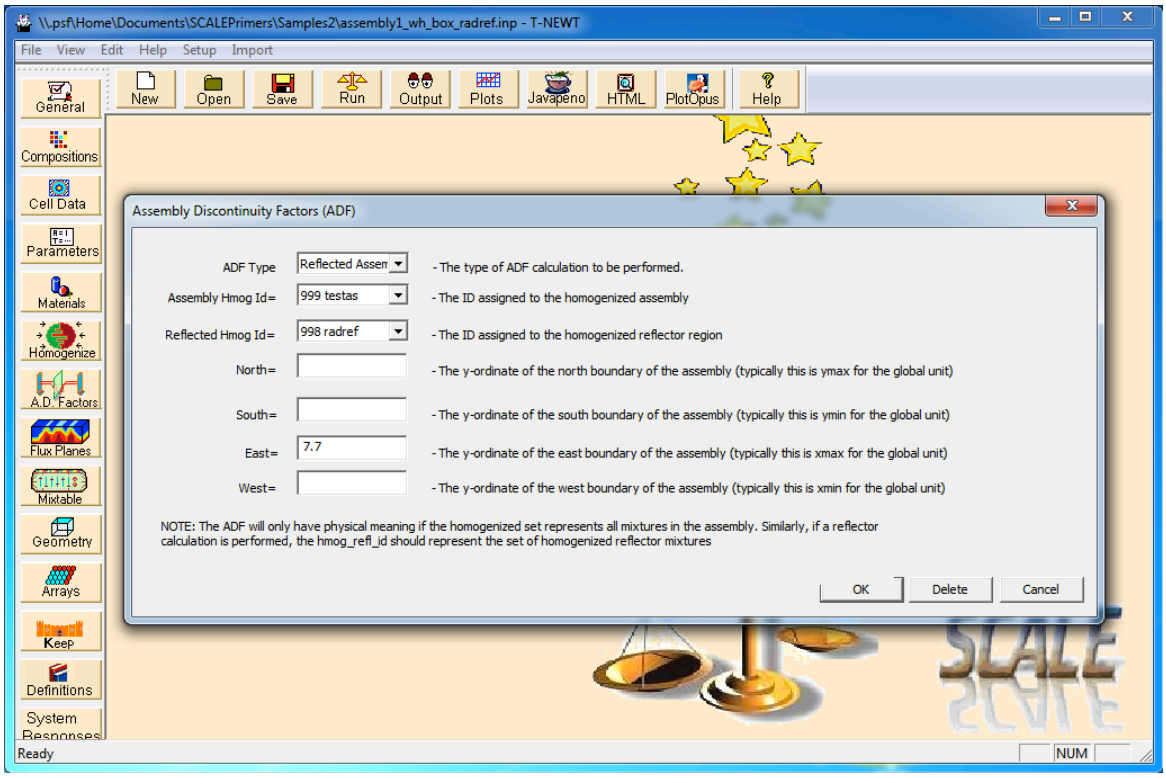


Fig. 8.19. Final Assembly Discontinuity Factor dialog box for the radial reflector example.

Now run the model with **Parm=check** to check and plot the model. The SCALE/NEWT representation of the radial reflector model can be found in Fig. 8.20.

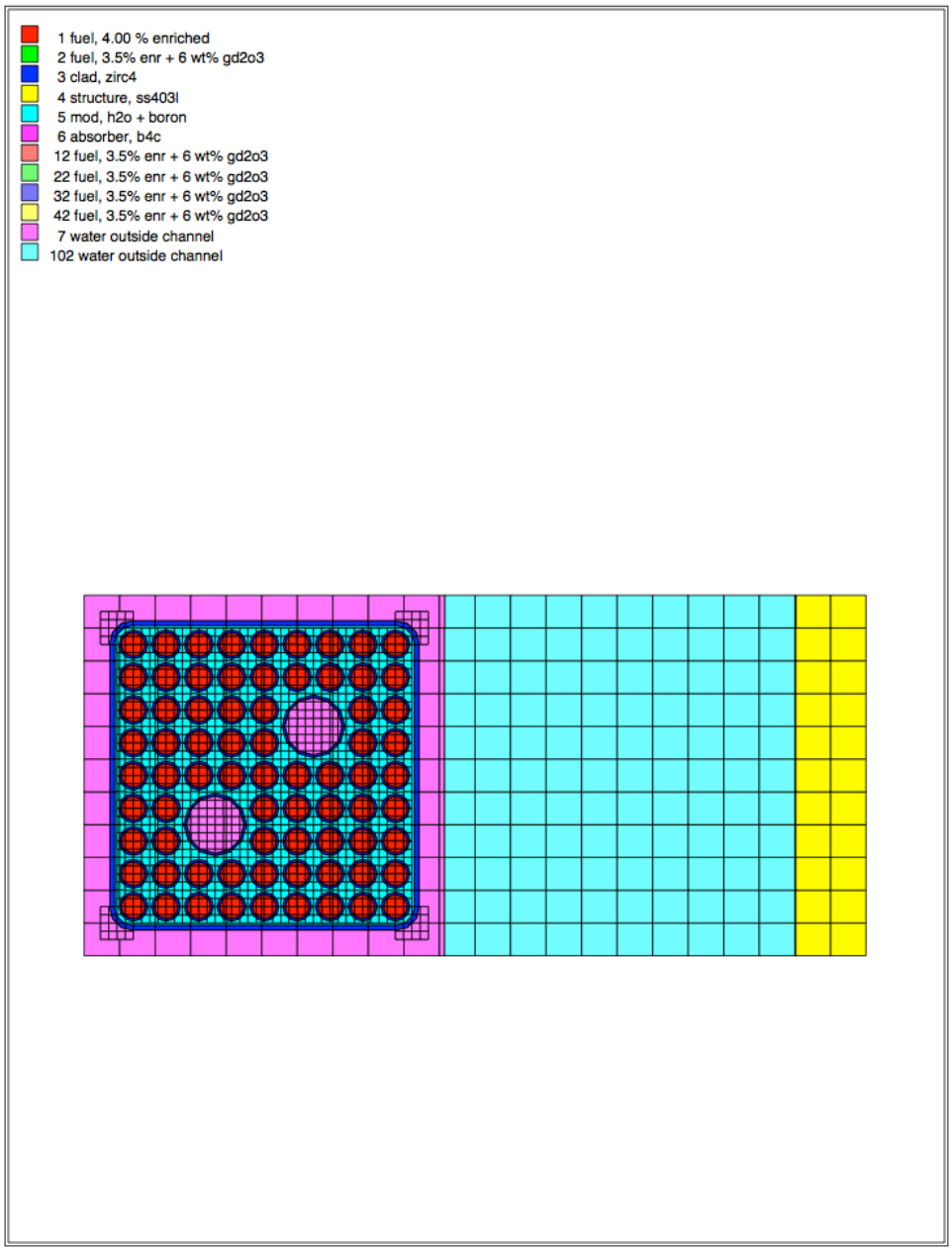


Fig. 8.20. SCALE/NEWT representation of the radial reflector model.

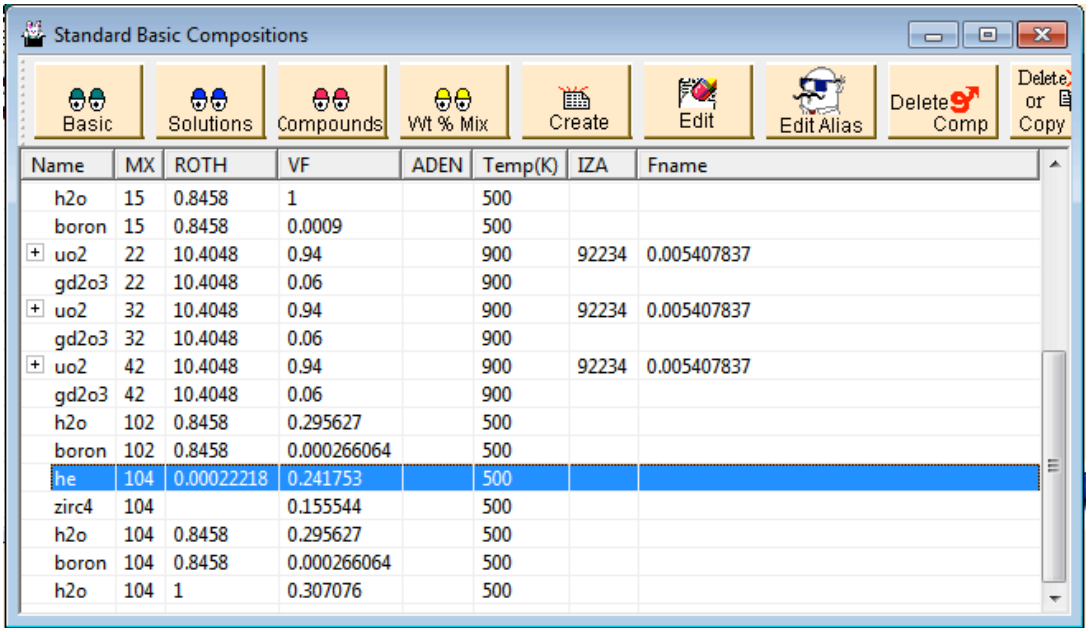
After running the model, the *xfile016* and *txtfile16* files will contain the nodal data for the reflector region in the first block of data. As with nodal data for a fuel lattice, the data can be processed and reformatted for use in nodal simulators.

8.5.3 Adding an Axial Reflector to the BWR Fuel Bundle Model

Generating nodal data for axial reflectors is very similar to the process used for the radial reflectors. Typically, at least the width of one fuel bundle is modeled on the east side of the assembly and a homogenized mixture similar to the one described in Sect. 8.5.1 is used to represent a mixture of

moderator and structural material. For this example, add a top axial reflector to the fuel bundle model. Start with the problem from Sect. 8.5.2, and modify the radial reflector to be the size of one fuel bundle in width, remove the stainless steel cylindrical structure, and change the reflector mixture to the mixture generated in Sect. 8.5.1.

The simplest way to add the homogenized mixture from Sect. 8.5.1 to the model is to copy and paste it into the model using the text-based input, but the same mixture can be constructed in GeeWiz. With `assembly1_wh_box_radref.inp` open in GeeWiz, click **Save As** and name the file `assembly1_wh_box_topref.inp`. Add the homogenized mixture to GeeWiz using the same mixture number for multiple standard compositions, and use the volume fraction to specify the correct volume of each separate standard composition. The final **Standard Basic Compositions** window is shown in Fig. 8.21 with the homogenized axial reflector material as mixture number `104`. Specification of multiple instances of a basic standard composition with the same mixture number is allowed, as seen in Fig. 8.21 where "h2o" is specified twice.



Name	MX	ROTH	VF	ADEN	Temp(K)	IZA	Fname
h2o	15	0.8458	1		500		
boron	15	0.8458	0.0009		500		
uo2	22	10.4048	0.94		900	92234	0.005407837
gd2o3	22	10.4048	0.06		900		
uo2	32	10.4048	0.94		900	92234	0.005407837
gd2o3	32	10.4048	0.06		900		
uo2	42	10.4048	0.94		900	92234	0.005407837
gd2o3	42	10.4048	0.06		900		
h2o	102	0.8458	0.295627		500		
boron	102	0.8458	0.000266064		500		
he	104	0.00022218	0.241753		500		
zirc4	104		0.155544		500		
h2o	104	0.8458	0.295627		500		
boron	104	0.8458	0.000266064		500		
h2o	104	1	0.307076		500		

Fig. 8.21. Standard basic composition mixture for the homogenize top axial reflector.

Now add the homogenized mixture to the list of NEWT materials by clicking the **Materials** button along the left vertical toolbar. Add mixture `104` to the list of materials with `PN=2`, as seen in the **Material Mixtures** dialog box in Fig. 8.22.

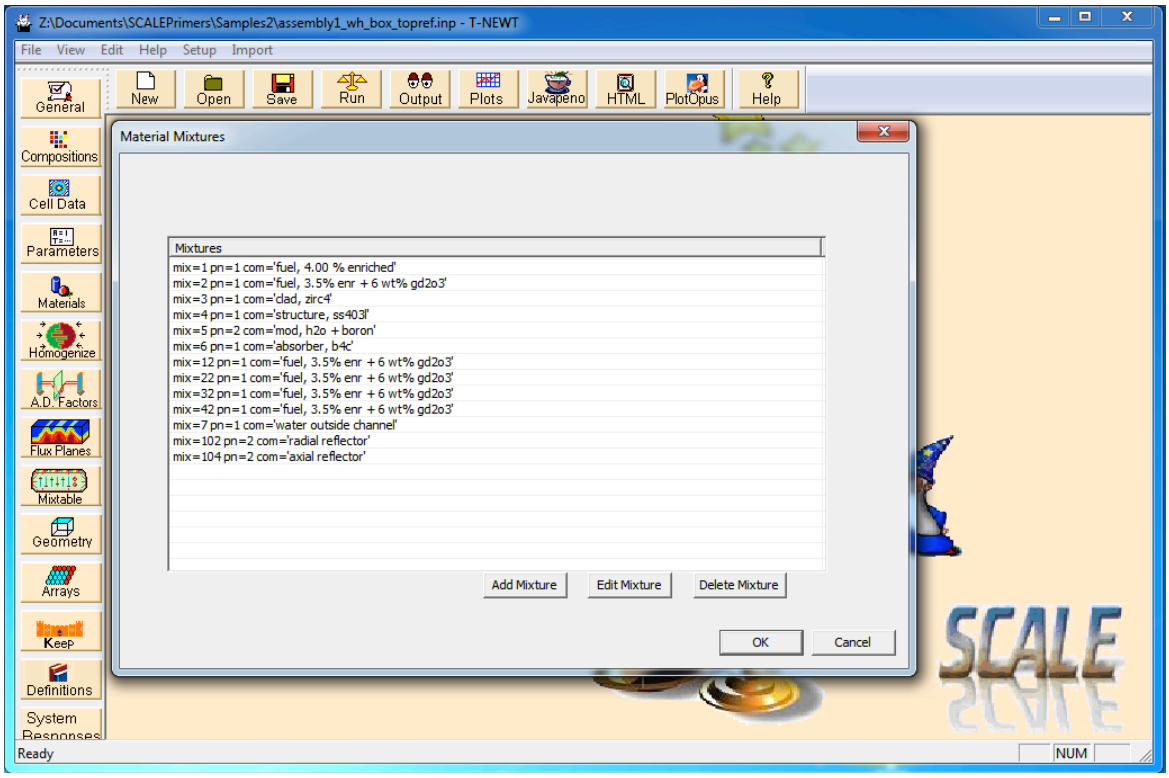


Fig. 8.22. Standard basic composition mixture for the homogenize top axial reflector.

Once the axial reflector material has been added, modify the geometry and add the mixture to the model. First, click **Geometry** from the left vertical toolbar and navigate to the global unit. Cuboid 20 should be the boundary cuboid – change the definition of cuboid 20 such that the positive x-boundary is `23.1` (7.7 + 2*7.7), which corresponds to one fuel bundle width. Then **Delete** hole 300 to remove the stainless steel cylindrical structure from the previous model. Unit 300 may remain in the input file because the unit only appears in the model if placed in the global unit. Now modify the media statements to include media 104, rather than media 102. The final global unit geometry form can be found in Fig. 8.23.

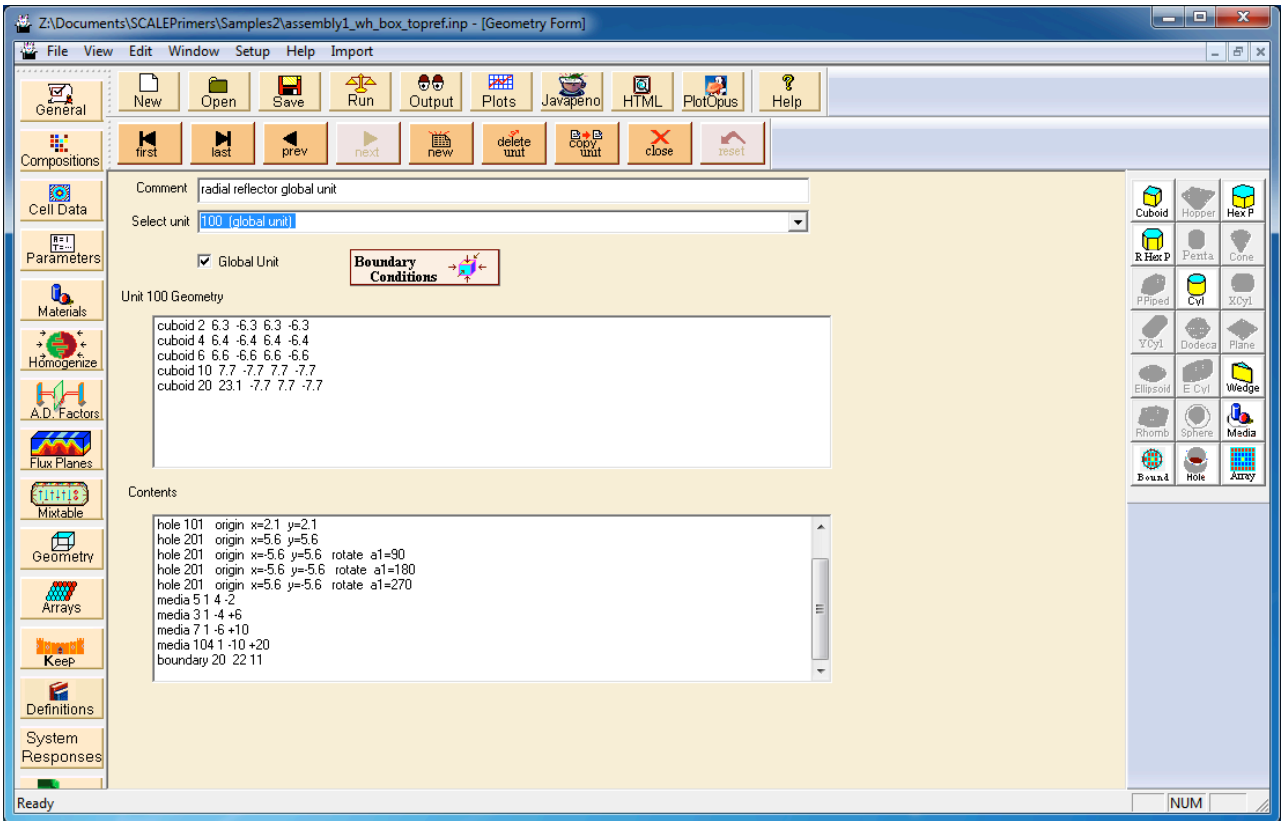


Fig. 8.23. Final global unit geometry form for the top axial reflector model.

Once the model is complete, **Run** the model using **Parm=check**. Then inspect the output file to ensure there are no errors. Also open the postscript files to plot the geometry. The model should look like the SCALE/NEWT representation in Fig. 8.24. After running the model, the *xfile016* and *txtfile16* files will contain nodal data required for the axial reflector.

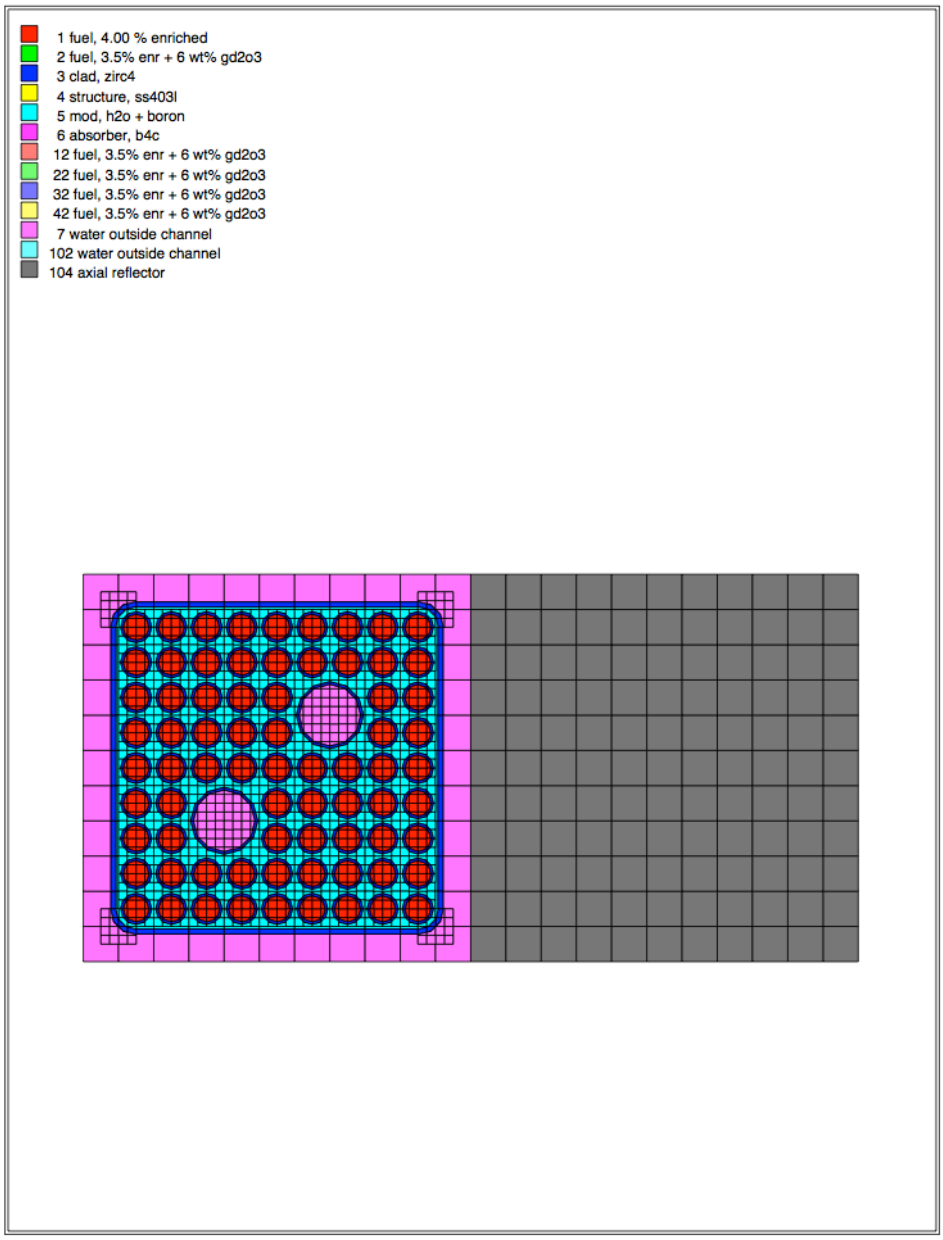


Fig. 8.24. SCALE/NEWT representation of the top axial reflector model.

8.6 SCALE/TRITON ACCELERATION TECHNIQUES FOR LWR LATTICES

Users of SCALE/TRITON have many acceleration options at their disposal. Some options have been previously mentioned in this primer, but others have not. This section of the SCALE/TRITON primer compiles the already-mentioned acceleration options with various other options that users can utilize to accelerate an LWR problem. Although this section and this primer focus on applications related to LWR lattices, these same techniques can be applied to other applications as well.

SCALE/TRITON depletion calculations consist of three parts: cross-section processing calculations (CENTRM/PMC or NITAWL), transport calculations (NEWT or KENO), and depletion calculations (COUPLE/ORIGEN). Because this primer is focused on the *t-depl* sequence of SCALE/TRITON, acceleration options for NEWT are discussed in depth. Also, NEWT typically requires more CPU time than other modules in the TRITON depletion sequence, so the most significant acceleration can be accomplished here. The acceleration techniques for NEWT can decrease CPU time in both NEWT steady-state calculations (*t-newt*) and depletion calculations (*t-depl*). Some acceleration techniques are only available in *t-depl* depletion calculations, but these techniques will often accelerate the NEWT transport portion of the depletion calculation. This section has been broken into two main sections: acceleration techniques for NEWT and acceleration options available in depletion calculations.

8.6.1 Acceleration Techniques in NEWT

A good way to test various acceleration strategies in NEWT is to construct an identical KENO MG model. The NEWT and KENO model should use the same cross-section libraries and identical cross-section processing. The only major difference between the models will be the transport solution. By comparing the NEWT solution to the KENO MG solution, users can evaluate the impact on accuracy of various acceleration techniques. Ideally, the solutions would be equivalent, within the convergence criteria of NEWT and estimated statistical deviation of KENO.

Grid spacing in NEWT

Decreasing the number of grid cells in NEWT is a possible way to decrease runtime. In general, at least 4x4 grid spacing, is recommended for each pin cell. NEWT requires that each body be cut by at least 1 grid line, so there is a limit on the coarseness of grid spacing. Using 4×4 grid spacing for each fuel pin cell and defaults for other parameters will generally result in a solution that is ~100 pcm from a corresponding KENO MG solution. Increasing the grid spacing (i.e., more cells) will decrease the bias, but at a cost to computer time. Doubling the grid spacing will slightly more than double the number of cells, and will therefore result in a CPU time increase of slightly more than double. Figure 8.25 shows a NEWT representation of a BWR mini-lattice problem with 4×4 (left), 8×8 (middle), and 12×12 (right) grid spacing. The mini-lattice problem ran in 99, 199, and 342 seconds with a bias from the KENO MG solution of 116, 81, and 25 pcm, respectively.

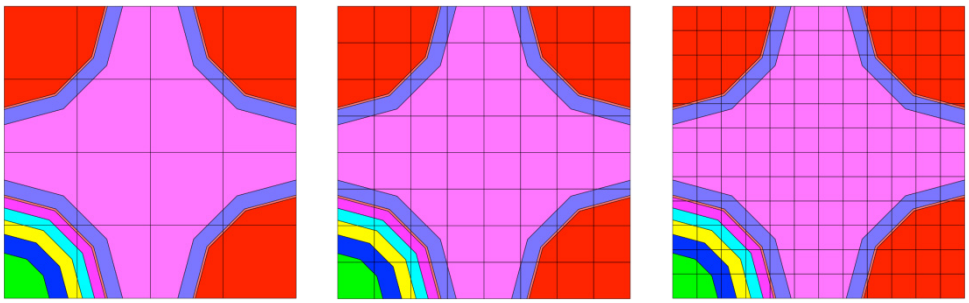


Fig. 8.25. Three examples of NEWT global unit spacing.

Even-numbered grid spacing for each pin cell is recommended; an odd number of grid cells for a single pin tend to result in a slightly larger bias than even-numbered grid spacing due to the way the grid lines segment the cell. This effect is reduced as the grid spacing is increased and is not present with grid spacing greater than 10×10 for each pin cell. As an example, using the model in Fig. 8.25, the grid spacing was refined from 2×2 to 13×13 and the results are plotted in Fig. 8.26 with the KENO MG solution plotted as a thick black line. In terms of k_{inf}, the results using 4×4 grid spacing are roughly equivalent to 7×7 grid spacing, but the 4×4 grid spacing case runs significantly faster than the 7×7 case.

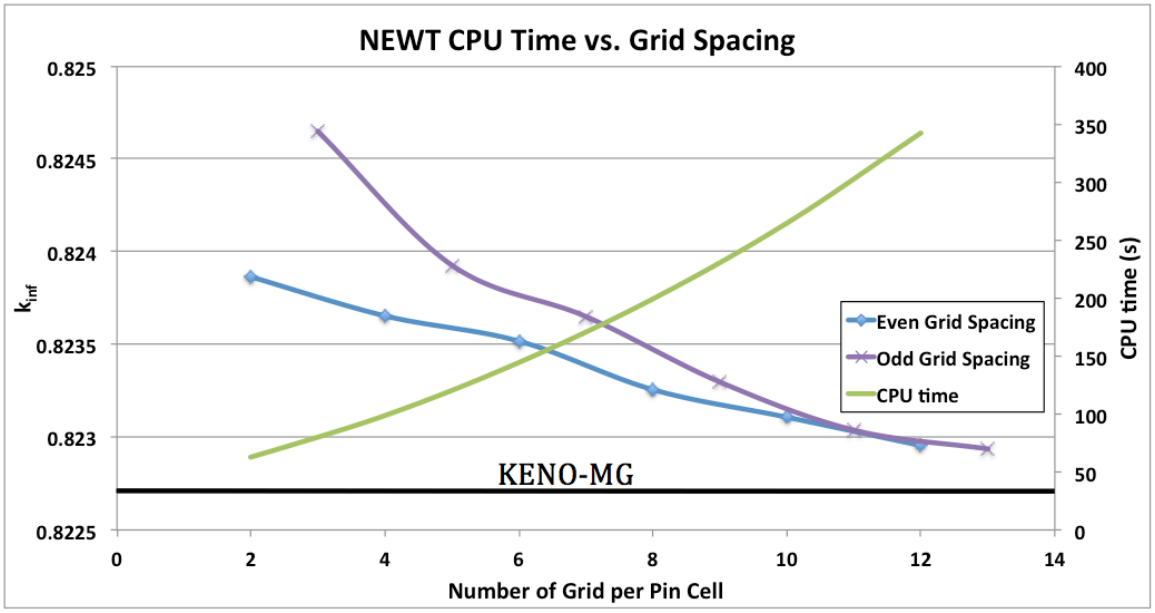


Fig. 8.26. NEWT CPU time as a function of grid spacing for the mini-assembly model.

Constructing an optimized NEWT grid

It is important that users understand how to construct an efficient and optimized NEWT grid. Extraneous grid cells can increase the amount of computational time required without making a significant contribution to the solution accuracy. For example, in NEWT it is possible for each unit in the model to have a subgrid structure in addition to the global unit grid. If this subgrid spacing is not optimized for the global unit grid, then many extra cells will be constructed in the model. Figure 8.27 shows a NEWT representation of a mini-assembly model using three types of subgrid spacing; the figure on the left shows a model with no subgrid, the figure in the center shows the northeast fuel pin with 4×4 subgrid, and the figure on the right shows the northeast fuel pin with poorly chosen subgrid.

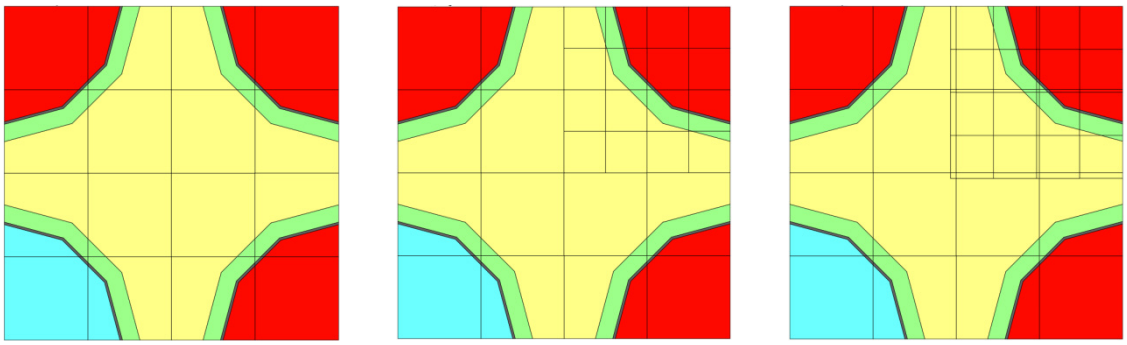


Fig. 8.27. Three examples of NEWT subgrid spacing.

If a subgrid is used on the fuel pins, it is preferable that it line up exactly with the global unit grid, as shown in the center of Fig. 8.27. The figure on the right has a poorly-chosen subgrid in which many small extraneous cells are created due to the subgrid. These small cells do not increase the solution fidelity and will only increase the computation time required. The model on the left with no subgrid required 32 seconds of CPU time, the model in the middle required 40 seconds of CPU time, and the model on the right required 51 seconds of CPU time, but all solutions were within a range of 20 pcm.

If a subgrid is used on each unit cell in a full lattice model, it is likely that many extraneous cells will be created. This is common for both BWR and PWR lattices whose lattice pitch is not an even multiple of the unit cell pitch. Using only a global unit grid is recommended unless higher solution fidelity is required in certain fuel pins. Using a grid only on the global unit will minimize extraneous small cells and will generally allow specification a more optimized CMFD mesh. For example, a good choice of global unit grid spacing for a full 17×17 PWR lattice would be 68×68 (17*4). Choosing a global unit grid for a BWR is slightly more difficult because the fuel pins do not extend to the boundary of the model. Generally, the distance between the boundary of the BWR lattice model and an edge fuel pin is approximately the width of one pin cell, so a user can specify a reasonable global unit grid using two plus the number of pins in one row of the lattice times four. Using a 10×10 BWR lattice as an example:

> (10 fuel pins in each row + 2 pin pitches for distance from the edge pin to the boundary of the model) × (4 grids per pin pitch) = 48 grid cells in the global unit.

A test case using a GE14 fuel bundle was constructed. In one model, the fuel pins were placed using a lattice and subgrid spacing of 4×4 for each pin cell and a global unit grid of 48×48. In the other model, each fuel pin (no square boundary on each pin) was placed using a hole in the global unit with global unit grid spacing of 48×48. A NEWT representation of the two models can be found in Fig. 8.28 with the model using the lattice on the left and the model using the holes on the right (only the northwest quadrant of each model has been plotted). The model using a lattice and subgrid spacing (on the left of Fig. 8.28) has many additional computational cells because the subgrid and global unit grid are not collinear. The reduction in computation cells in the model using holes to place fuel pins has a significant impact on CPU time; in this case, the model completed in approximately ½ the time required for the model using a lattice and subgrid spacing with a difference of ~90 pcm.

Caution should be used for multi-ring burnable absorber fuel pins as asymmetrically cutting these pins by grid lines could have a larger impact. Using subgrid spacing on only those pins will minimize this impact. If CPU time is a concern, use holes to place the fuel pins and a grid only on the global unit. Ensure that all multi-ringed burnable-absorber fuel pins have a grid line passing through the central region, and if not, apply subgrid spacing on that unit.

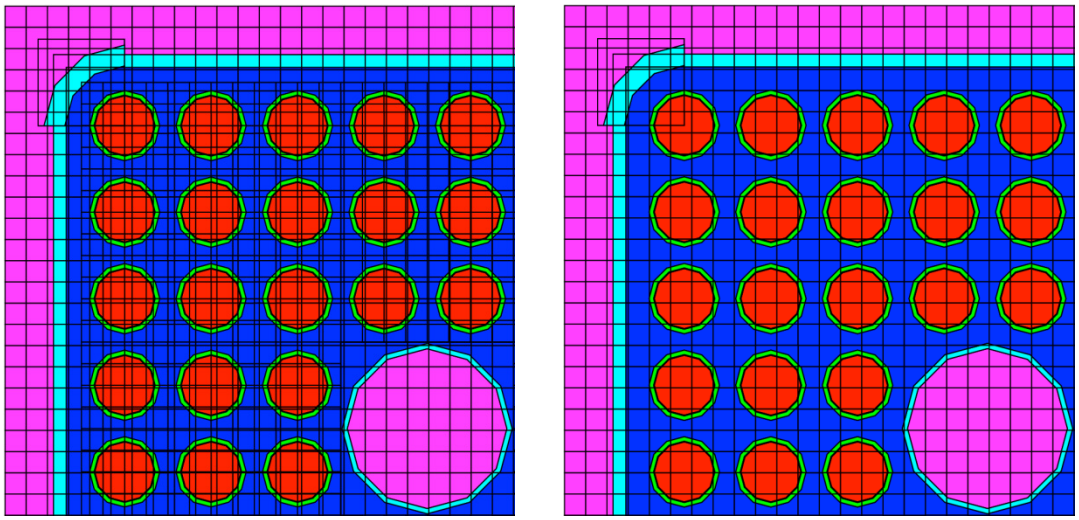


Fig. 8.28. SCALE/NEWT representation of the northwest quadrant of a GE14 fuel lattice using a lattice to place fuel pins with subgrid spacing and using holes to place pins with no subgrid spacing.

CMFD acceleration in NEWT (Sect. 4.6.2)

Coarse-mesh finite-difference acceleration is a common acceleration technique used in lattice physics codes. As the name suggests, it uses a low-fidelity solution to accelerate the high-fidelity solution. An example using the NEWT CMFD solver can be found in Section 4.6.2. CMFD acceleration should be used for all lattice physics calculations. Using CMFD will not make the problem converge to a different solution, it will only make convergence to the solution faster. In general, it is recommended that users apply CMFD acceleration on the global grid and make the CMFD grid cells approximately the size of one fuel pin unit cell. For example, in a full 17×17 PWR fuel lattice model with 4x4 grid spacing for each pin, a global unit grid would consist of 68×68 total grid cells (17 pins * 4 grid cells = 68 total grid cells on the global unit). Then the CMFD parameters, *xycmpfd*, should be set to *xycmfd=4*, specifying four fine mesh cells for every one coarse mesh cell in both the x- and y-direction.

Angular quadrature in NEWT

As with grid spacing, increasing the angular quadrature in NEWT will increase the CPU time to obtain a solution. The default in NEWT is a level symmetric quadrature set with 6 angles (*sn=6*), which is recommended for LWR calculations. This default will generally provide reasonable solutions compared to KENO MG calculations. Increasing angular quadrature will result in solutions that are closer to the KENO MG solution at a penalty to CPU time. Decreasing the angular quadrature to a level symmetric set with fewer than six angles generally yields poor results for LWR configurations. Separately specifying the number of polar and azimuthal angles is possible in NEWT, and could potentially be used to decrease the CPU time, but extensive studies for LWRs have not been performed, because the default (*sn*=6) typically works well for LWR calculations. Although extensive studies have not been done, higher order quadrature might be necessary for very detailed and accurate determination of pin powers.

Convergence Criteria in NEWT

Users can control the convergence criteria in NEWT, so it is possible to relax the convergence criteria in order to decrease the CPU time required. In general, the eigenvalue converges much faster than the flux. Relaxing the convergence criteria of the flux will result in faster solution convergence. However, relaxing the convergence criteria of the flux can have an impact on the accuracy of depletion calculations and nodal data generated with SCALE/TRITON. If only the eigenvalue is of concern, users can relax the flux convergence criteria to decrease CPU time. For depletion calculations, the default values in SCALE/TRITON should be used. One common way to decrease computer time is to use convergence-by-mixture (*converge=mix*) option instead of the SCALE/TRITON default, which is convergence-by-cell (*converge=cell*). This option can significantly decrease CPU time required for problems that have large regions of the same mixture, problems that have many small cells, or problems that have low-flux regions (reflector problems).

Multi-threading on multicore processors

The NEWT transport solver has the ability to utilize multiple threads on a multicore processor. This can significantly decrease the wall-clock time necessary to run NEWT calculations. This option is currently only available for NEWT, so CENTRM and other modules in the SCALE/TRITON sequence will run on only one thread. It is also currently only available when calling SCALE using the command line; this feature is not supported in the GeeWiz graphical user interface. To use this feature on the command line, use the "-I" command line option followed by the number of threads on which the calculation will run. The following command uses four threads and prints messages to the screen:

```
>>  batch6.1 -I 4 -m test1.inp
```

A full list of the command line options available in SCALE can be found by typing "batch6.1" at the command prompt with no additional options or input files. Remember that if using multiple threads on a multicore processor, other software will be competing with SCALE for resources, so other software may run slower than normal.

8.6.2 Acceleration Techniques for Depletion Calculations

SCALE/TRITON executes a number of COUPLE/ORIGEN-S calculations to perform depletion of fuel materials. These depletion calculations run quickly and require little CPU time, so there is little time to be gained in the depletion calculations themselves. However, there are additional options available in the SCALE/TRITON depletion sequence (*t-depl*) that are not available in the transport sequence (*t-newt*) that can accelerate the overall solution.

Parm=weight for reduced group structure calculations (Sect. 6.2.2)

The parameter option "*parm=weight*" is a commonly used option used to accelerate the NEWT transport calculations performed during SCALE/TRITON depletion. The *parm=weight* option utilizes a single 238-group transport calculation performed at BOL to collapse the 238-group cross-section library to a problem-dependent 49-group cross-section library. Subsequent transport calculations performed during depletion then use the 49-group library, thus decreasing the NEWT CPU time by 4-5 times. The significant decrease in CPU time typically comes with a slight penalty to solution fidelity, referred to as the collapsing bias. The collapsing bias can be checked by searching for "k-eff =" in the output file and recording the eigenvalues for the collapsing step and for the time=0 step as follows:

```
>> grep "k-eff =" testcollapse.out
```

```
 k-eff =       1.05156060      Collapsing Sol'n
 k-eff =       1.05182796      Time=      0.00d
...
```

In this case, the collapsing bias is 27 pcm, which would be acceptable for most problems. Generally, if proper self-shielding is applied to all fuel pins including user-defined Dancoff factors, the collapsing bias will be less than 100 pcm. For highly heterogeneous lattices in which accurate self-shielding is difficult, the collapsing bias can be larger than 100 pcm, especially for high-void BWRs. Using the *parm=weight* parameter for depletion calculations is recommended, but the collapsing bias should be checked to ensure that it is sufficiently small for the problem of interest. Because the 49-group structure is optimized for LWRs, larger biases could be observed for different spectra systems. Also, before extensively using the 49-group library for depletion calculations, isotopes of interest should be check at EOL to ensure that no major discrepencies are observed when compared to the 238-group library.

Assign function for reduced cross-section processing (Sect. 6.3.2)

Using the "assign" function available in SCALE/TRITON depletion calculations can significantly reduce the cross-section processing time required for accurate depletion calculations. In an LWR lattice, each fuel pin should be depleted separately; in other words, the isotopic concentration of each pin should be tracked separately from all other non-symmetric fuel pins. This can be done in SCALE/TRITON by specifying a different mixture number for each non-symmetric fuel pin. Before SCALE 5.1, each fuel mixture needed corresponding cross-section processing for accurate depletion calculations, making the cross-section processing step a significant contributor to the total CPU time. In SCALE 5.1, the *assign* feature was introduced that, when implemented properly, performs cross-section processing on an "average" mixture and updates microscopic cross sections of each fuel mixture accordingly. The assign function introduces almost no bias in depletion calculations and can significantly reduce CPU time required in lattices that have many fuel pin types. Using the assign function is always recommended for depletion calculations.

The number of depletion steps required for accurate calculations

The question of the number of depletion steps, or the depletion step size, is a common question to which a reliable answer is difficult to define. In terms of BWR lattices, the number of depletion steps required for accurate simulations is highly dependent on the number and content of gadolinia-bearing burnable absorber fuel pins in the lattice. For lattices that contain burnable absorber fuel pins, the depletion step size should be small at the start of the calculation when gadolinia concentration is changing rapidly, and increase in size as the calculation progresses past peak reactivity.

Using the mini-assembly model (Fig. 8.25), a depletion step convergence study was conducted. Constant depletion step sizes were used for the first 30 GWd/MTU, varying in size from 0.039 GWd/MTU to 5 GWd/MTU. The infinite multiplication factor at 20 GWd/MTU (near peak reactivity) was compared for each case. Generally, peak reactivity is the point at which the largest bias occurs. For this comparison, the simulation utilizing 0.039 GWd/MTU steps was used as a reference case. The difference in the infinite multiplication factor at 20 GWd/MTU has been plotted as a function of depletion step size on the primary axis in Fig. 8.29, and the total number of depletion steps has been plotted on the secondary axis. As seen in Fig. 8.29, the eigenvalue bias at peak reactivity is highly dependent on the depletion step size. Steps larger than 1.0 GWd/MTU should not be used before peak reactivity for gadolinia-bearing lattices. The burnup at which peak reactivity occurs varies as a function of gadolinia content, so it is impossible to give a depletion step scheme that is adequate for all scenarios. Decreasing the number of depletion steps can be used to accelerate depletion calculations, but it should be done with caution

because large depletion steps can result in significant biases. Generally, a depletion step size of 3.0 GWd/MTU is acceptable past peak reactivity burnup.

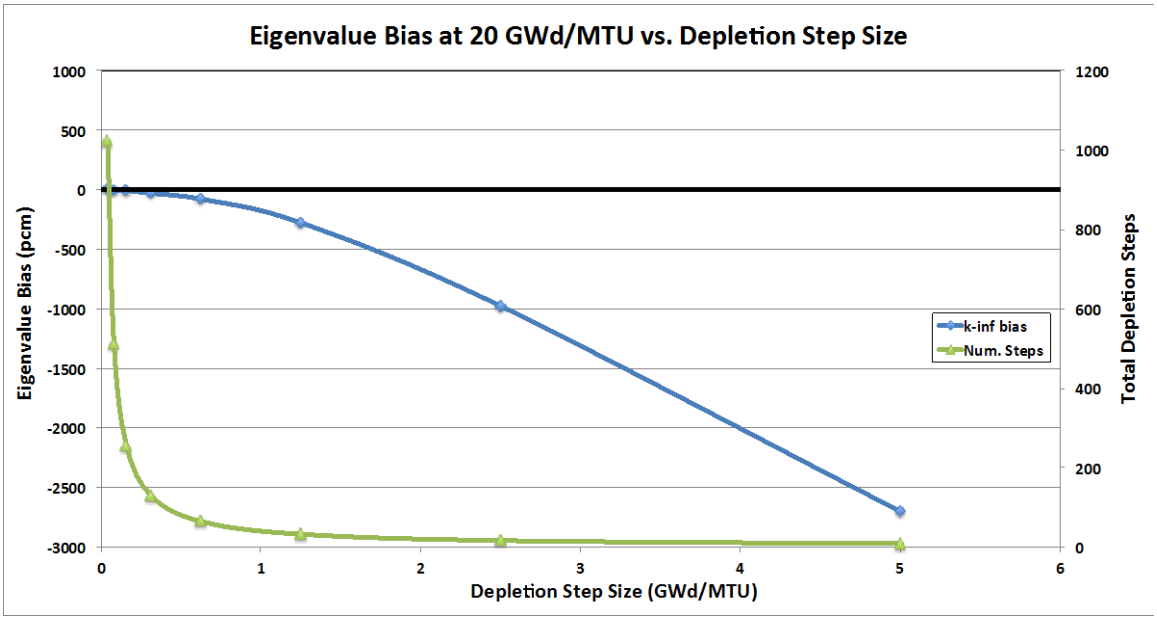


Fig. 8.29. Eigenvalue bias from the reference solution at 20 GWd/MTU for varying depletion step size.

The number of rings required for fuel pins containing burnable absorber

A burnable absorber fuel pin, like those common to BWR fuel bundles, should be depleted using multiple rings to properly capture the radial depletion of gadolinia. In order to illustrate the importance of the number of rings in burnable-absorber fuel pins, the number of rings used in the mini-assembly model (Fig. 8.25) was varied from 1 to 25 volume-equivalent rings. The solution using 25 rings was considered a reference solution. The difference in eigenvalue compared to the reference solution at a burnup of 20 GWd/MTU was plotted as a function of the number of rings in Fig. 8.30. As the number of rings increases, the bias from the reference solution decreases with diminishing returns past five rings. The CPU time increases linearly with the number of rings. Using five rings rather than 25 rings for burnable absorber fuel pins results in a maximum bias of ~100 pcm at peak reactivity for this configuration. Other problems using a different gadolinia concentration might have different results. Generally, at least five rings are recommended for burnable absorber fuel pins.

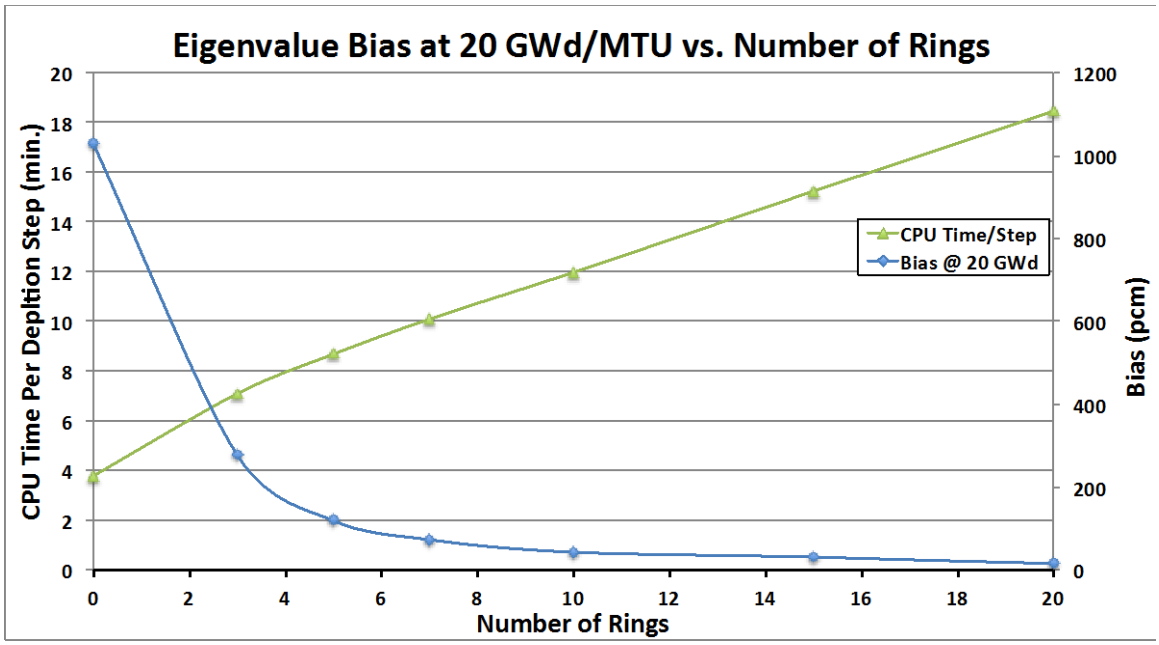


Fig. 8.30. Eigenvalue bias from the reference solution at 20 GWd/MTU for varying number of rings in burnable absorber fuel pins.

9. REFERENCES

1. *SCALE: A Comprehensive Modeling and Simulation Suite for Nuclear Safety Analysis and Design*, ORNL/TM-2005/39, Version 6.1, June 2011. Available from Radiation Safety Information Computational Center at Oak Ridge National Laboratory as CCC-785.
2. T. Downar, Y. Xu, and V. Seker, *PARCS V3.0, U.S. NRC Core Neutronics Simulator*, User Manual, Department of Nuclear Engineering and Radiological Sciences, University of Michigan, Ann Arbor, MI, March 2010.
3. G. I. Maldonado, J. Galloway, H. Hernandez, K. T. Clarno, E. L. Popov, M. A. Jessee, "Integration of the NESTLE Core Simulator with SCALE," *Trans. Am. Nucl. Soc*. **100**, 619-620 (2009).
4. T. Bahadir and S. Lindahl, "Studsvik's Next Generation Nodal Code SIMULATE-5," *Advances in Nuclear Fuel Management IV*, Hilton Head, SC, April 2009.
5. "SS 304L Technical Data Sheet," Hamilton Precision Metals, Lancaster, PA, March 22, 1991.

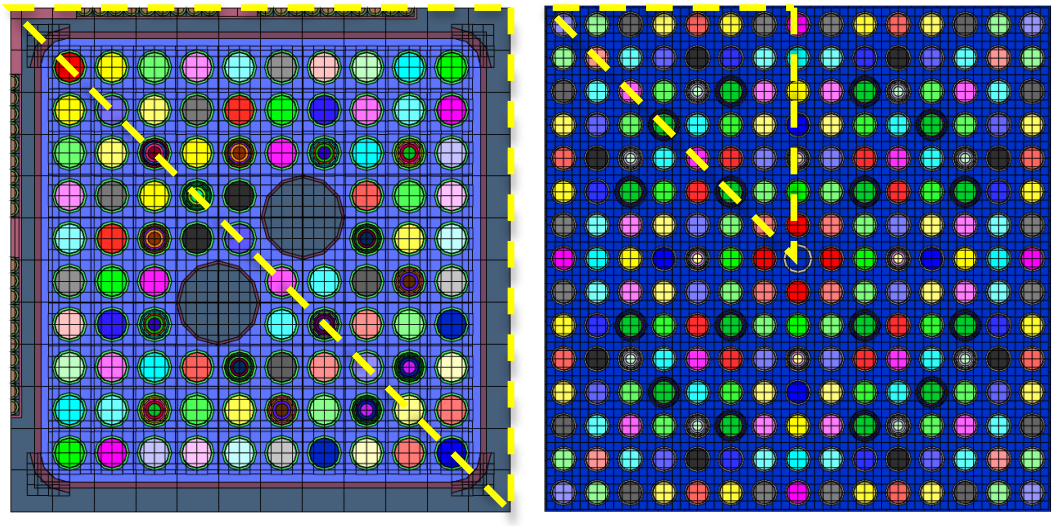
APPENDIX A

TRITON LWR LATTICE PHYSICS USER GUIDANCE

This appendix is designed such that a modeler with a significant amount of SCALE experience but little lattice physics experience could use this section as a reference for lattice physics model development. This section makes little reference to the specific setup of problems but addresses cruxes that a modeler might encounter during development of lattice physics models. In addition, a novice user could use this appendix as a guide after he or she has learned the basics of SCALE model development. The appendix is designed as a "Frequently Asked Questions" document that attempts to answer a number of common questions that modelers might ask themselves during model development.

How many compositions should I define?

- An accurate depletion model with many unique fuel locations requires the use of many different materials.
- BWRs generally require many more materials than PWRs.
    - BWR lattices typically have 1/2 diagonal symmetry, whereas PWR lattices generally have 1/8 assembly symmetry.
    - BWR and PWR lattices typically have many pins of different enrichment and gadolinium content.
- The user should find the smallest symmetric "subassembly," usually 1/2 (BWR) or 1/8 (PWR), and model each symmetric fuel pin as a different material.
- In addition, a user should define different fuel materials for each equal-volume ring (5–10 per fuel pin) in a gadolinium-bearing pin.
- A user will also need to define different gap, clad, and moderator mixtures for each CENTRM calculation, i.e., one for each enrichment and gadolinium content.
- If nonstandard Dancoff factors are being used, another set of mixtures (fuel, gap, clad, and moderator) must be specified for each separate pin.
- Control blade pins can all be modeled as the same material.
- A user should define different compositions for the water rod and outside channel water in BWRs as this water is likely a different density than the in-channel water.



How should I define my aliases?

- Some thought should be given to defining the aliases for any certain model. Using some foresight in defining the aliases will make subsequent modification of the model much simpler.
- The alias name should reflect some information about the fuel pin being specified—enrichment and gadolinium content. $fuel485 might specify a UO_2 pin that is 4.85 wt% enriched. $fl450gd350 might specify a fuel pin that is 4.5 wt% enriched with 3.5 wt% Gd_2O_3.
- Generally, it is wise to specify pins by a number that corresponds to that pin's position in the lattice. It is recommend to use the number scheme *yx*, where *y* is the *y*-position in the lattice and *x* is the *x*-position in the lattice. This means that fuel pin materials start with the number 11 for the northwestern most pin, and count up across the lattice:

```
$1fuel485    111 end
$fuel485      11  12  13  14  15  16  17  18
                  22  23  24  25  26  27  28
                          34  35      37  38
                          44  45  46  47  48
                              56  57  58
                                  67  68
                                  77  78 end
```

  - The above example specifies a lattice of 4.85 wt% enriched fuel pins with holes in the lattice left for guide tubes, instrumentation holes, and gadolinium-bearing fuel pins.
- It is also helpful to specify gadolinium-bearing pins in this way:

```
$f450gd350  166 266 366 466 566 end
```

  - This alias example specifies five rings of gadolinium-bearing fuel.
- Specifying aliases makes it easy to use TRITON's "assign" function that assigns a certain cross section to a material. Using the assign function eliminates the need for separate CENTRM calculations for every fuel mixture number. Cross-section processing can be performed for the $1fuel485 alias, then assigned to $fuel485 in the branch block.
- It might also be helpful to specify control rod/blade materials in the aliases. $CRin and $CRout could specify that control rods are in or out. This helps eliminate confusion that could occur.
- Below is a full alias block that could be used for a 17x17 PWR assembly—in this case there is no fuel pin 88 because it is the instrument tube location, other absent pins are guide tubes.

```
$onefuel495  111 end
$fuel495       11 12  13  14  15  16  17  18
                  22  23  24  25  26  27  28
                          34  35      37  38
                          44  45  46  47  48
                              56  57  58
                                  67  68
                                  77  78 end
$fuelgd3     566 666 766 866 966 end
$oneclad275  210 211 212 213 end
$onecool275  310 311 312 313 end
$onegap275   410 411 412 413 end
```

```
$crpelout    501 end
$crpelin     502 end
$crgapout    503 end
$crgapin     504 end
$crcladout   505 end
$crcladin    506 end
```

What cross-section processing options should I use?

- The cross-section processing options in SCALE are highly flexible. The most accurate solution is obtained using CENTRM and the ENDF-7 cross-section library. Faster, less accurate cross-section processing is available using NITAWL (only available with ENDF-5 cross-section libraries).
- For fuel pins without gadolinium, a one-region **latticecell** specification is sufficient. This assumes that the fuel pin lies in an infinite lattice of identical fuel pins—a good assumption in most PWR and some BWR cases.
- Some fuel pins in high-void BWR lattices might need to use a special option to modify the Dancoff factor for that fuel pin. In high-void BWR lattices, the infinite lattice assumption is a poor assumption for edge and corner fuel pins. This is discussed in depth in Appendix B.
- For gadolinium-bearing fuel pins, a **multiregion** specification should be used to process the cross sections. Because gadolinium is such a strong absorber of neutrons, the flux shape across the fuel pin changes significantly. This technique involves dividing the fuel pin into 5–10 equal-area rings. These rings must also be in the actual lattice geometry.
- Choosing the cross-section processing method depends strongly on the time constraints. If the answer is needed very quickly and accuracy need not be great, NITAWL might be a good choice. If accuracy is of great concern, then CENTRM should be used. A good tradeoff between speed and accuracy can be found by using the 2REGION option in CENTRM (parm=2region).
- If using the dan2pitch option, then CENTRM or 2REGION must be used.

What depletion strategies should I use for BWRs and PWRs?

- The strategy for depleting BWR and PWR lattices are fairly similar. In TRITON, a relatively simple depletion block is used.
- You can use aliases in the depletion block to specify groups of materials.
- Gadolinium-bearing fuel should be depleted by flux instead of by power.
- Also in this block, you should use the assign function to assign cross sections to applicable mixtures. Using this feature limits the amount of time spent on cross-section processing.

```
read depletion
  $fuel495 flux
  $fuelgd3 end
  assign $onefuel495 $fuel495 end
end depletion
```

- In TRITON, the specific power density must be input in megawatts per metric ton of heavy metal (MW/MTHM). The power density can be calculated from the total reactor power, the number of

assemblies, and the mass of uranium in each assembly. It can also be calculated from the linear heat rate, fuel pin dimensions, and fuel density.

- Because gadolinium burns out fairly quickly, relatively fine depletions steps are needed at the start of the depletion. After the peak reactivity is reached, the depletion steps can be lengthened.
- The TRITON "nlib" option is used in the burndata block to specify the number of cross-section updates per burn period.
- The user should specify a first step that is very small (~100 hours) in order to set xenon/samarium equilibrium concentrations.
- Remaining steps for gadolinium-bearing lattices should be kept lower than 2 GWd/MTHM for the entire burn range, and less than 1 GE/MTHM prior to peak reactivity.

What goes in the branch block?

- The branch block contains all data that TRITON needs to calculate branch conditions (higher/lower moderator density, fuel temperature, soluble boron, etc.).
- The branch block starts with a number of "define" statements. The "define fuel" example below specifies the mixtures for which fuel temperature branches will be done.

```
define fuel    $1fl395     $1fl440     $1fl490
               $fuel395    $fuel440    $fuel490
               $1f490gd700 $1f440gd700 $1f440gd800
               $f490gd700  $f440gd700  $f440gd800  end
```

- All fuel mixtures must be listed here, including the fuel materials that are used in the cross-section processing step and assigned to other fuel materials in the depletion block.
- The "define mod" block specifies the moderator materials for which moderator density, moderator temperature, and soluble boron branches will be performed. It is important to remember in the case of BWRs that if both the in-channel and outside-channel moderators are included in the "define mod" statement, they will have the same density, temperature, and soluble boron concentration in the branch calculations.
- Because of this limitation, it is impossible to perform soluble boron concentration branches on the outside-channel water. For BWRs, there is no soluble boron in the water under nominal conditions, so this is acceptable for most cases. For transient branches that have soluble boron injection, there will be no soluble boron included in the outside-channel water resulting in a bias in the reactivity of the model; TRITON will calculate a higher k_{inf} than it should. This limitation is being eliminated in future releases of SCALE.
- The "define crin" and "define crout" blocks specify control rod materials that are exchanged in control rod branches.
- The "define d2pset" block specifies a set of Dancoff factors that should be used during cross-section processing for a certain branch. Many times, a user will want to specify moderator density branches different than the nominal case. In the case of BWRs, a different Dancoff factor is likely needed for the edge and corner fuel pins of these branches. This is probably best shown by an example.

In Celldata:

```
latticecell squarepitch  pitch=1.23          913
                         fuelr=0.44          140
                          gapr=0.45          911
                         cladr=0.52          912          end
centrmdata dan2pitch(140)=0.360 end centrmdata

latticecell squarepitch  pitch=1.23          923
                         fuelr=0.44          280
                          gapr=0.45          921
                         cladr=0.52          922          end
centrmdata dan2pitch(280)=0.360 end centrmdata
```

In the branch specification:

```
  define d2pset 100   140 0.360   280 0.360   end
  define d2pset  80   140 0.300   280 0.300   end
  define d2pset  40   140 0.230   280 0.230   end
  define d2pset   1   140 0.175   280 0.175   end
```

- It is a good idea to name a d2pset based on the void fraction corresponding to those branches. Remember: -1 and 0 have special meanings, so a user should not use those numbers in "define d2pset" unless the options corresponding to those numbers are desired.
- Below are a few examples of actual branch specifications that use the Dancoff factor option:

```
dm=0.7397    tf=863.15   tm=559.15  sb=1  cr=0  d2p=  1  end
dm=0.458432  tf=863.15   tm=559.15  sb=1  cr=0  d2p= 40  end
dm=0.177164  tf=863.15   tm=559.15  sb=1  cr=0  d2p= 80  end
dm=0.03653   tf=863.15   tm=559.15  sb=1  cr=0  d2p=100  end
```

  - Where
    dm = density of the moderator,
    tf = temperature of the fuel,
    tm = temperature of the moderator,
    sb = soluble boron content (in ppm),
    cr = control rod state; 0 is out, 1 is in,
    d2p = Dancoff factor set to be used.

- **Watch out!!!** The soluble boron concentration specified in the base input file moderator mixture must be the concentration specified in the first branch. The soluble boron concentration in the base case should be greater than zero; otherwise it cannot be modified in branch cases. In this case, a user would specify 1 ppm of soluble boron in the composition block for the moderator. If a user specifies a boron concentration in the first branch unequal to what is specified in the moderator composition, the concentration will be incorrect for other soluble boron branches. The example below uses 1 ppm boron.

```
h2o      $mod     den=0.7397 1      559.15 end
boron    $mod     den=0.7397 1e-6   559.15 end
```

What branches should I have?

- The specific branches that a user will need are highly dependent on the analysis being performed. Steady-state analysis requires very few branches. On the other hand, transient analysis requires many different branches.
- If producing cross sections to be used in a nodal simulator (e.g., PARCS), the branch specifications will need to be wide enough to cover anticipated conditions without large cross-section extrapolation. It is much better to interpolate cross sections rather than extrapolate cross sections. When in doubt, add another branch to cover a possible operating condition.
- PWRs
    - A user should have branches corresponding to startup and shutdown.
    - A user should have branches that simulate a quick transient (fuel temperature only).
    - A user should have different moderator density branches that correspond to different operating temperatures.
    - A user should have soluble boron branches that cover possible nominal boron concentrations, including letdown; 0 to ~2500 ppm and various points between for proper interpolation (e.g., 0, 700, 1400, and 2200 ppm).
    - The user should also have control rod branches corresponding to control rods in and control rods out.
- BWRs
    - "History" calculations may be needed for BWRs. This means that a lattice is burned with different void fractions in order to obtain some information on the void history of a particular section of a lattice. It might be necessary to burn a lattice with 0, 40, 80, and 100% void fractions. It also might be necessary to include control blade history calculations for BWRs.
    - Other than history calculations, the branch specifications for BWRs are very similar to PWRs.
    - If high void moderator density branches are being run, "d2p" will likely need to be used for an accurate solution.
- When working with multiple input files that use the same branch structure, it might be helpful to specify a "tree" file that contains all the branches that are going to be used for different input files. This centralizes the branch specification so that if something needs to be changed, it is only changed in one file, rather than many different input files. This also ensures consistency in the branch structure across different input files and avoids associated problems when using these files to generate cross sections for a core simulator.
- This file needs to be in the user's runtime directory, or copied into the runtime or temporary directory during execution by using the "shell" module. Once the file has been copied to the correct location, it can simply be fed into the input file using "<filename":

```
=shell
 copy \mella+\rev02\tree.00V %TMPDIR%
end

=t-depl
…
read branch
<tree.00V
end branch
…
end
```

Contained in the tree.00V are the branches corresponding to 0% void nominal conditions.

- Before a user spends a significant amount of computer time on a simulation to generate cross sections for a nodal simulator, it is recommend that you start with a simple model that includes the full branch statements. This simulation does not have to be run for the entire burnup range. The main goal is to generate a SCALE/TRITON *xfile016* file that exercises the capability of GenPMAXS to process the branch structure being used. It is possible to generate a branch structure that cannot be processed by GenPMAXS, so it is important to test the proposed branch structure before using a significant amount of computer time generating the full cross section set.

What options should I use in my NEWT transport solution?

- Currently, the basic defaults for NEWT are recommended. Future investigation could lead to a more optimized set of parameters, but those investigations still need to be performed.
- Quadrature: S_6.
- Scattering: P_0 in the gap; P_1 in fuel, clad, and structural materials; and P_2 in water.
- Spatial mesh of 4×4 in fuel cells. The global grid is sized such that each grid cell is approximately 1 unit cell-sized.
- CMFD is performed on the global grid.
- Epsilon = 1×10^{-5}.
- Reflective boundary conditions.
- Recent studies have shown that adding a ring of water around fuel pins increases the accuracy of the solution by refining the NEWT solution mesh. Adding this ring increases the spatial accuracy of the flux entering the fuel pin. This adds some time to the NEWT solution that may be worthwhile depending on the accuracy needed.

Which cross-section library should I use?

- It depends on the required accuracy of the solution and the time available.
- If time allows, usage of the 238-group ENDFB-VII (v7-238) multigroup library is recommended for the most accurate results.
- If a quick estimate is needed, the 44-group ENDFB-V (v5-44) multigroup library can be used.

- A good compromise between accuracy and fast runtime can be accomplished using parm=weight.
    - Generates a problem-specific 49-group library that is used in subsequent transport calculations.
    - This library can be saved and used for other similar lattice calculations.

How do I know if all my assumptions are correct?

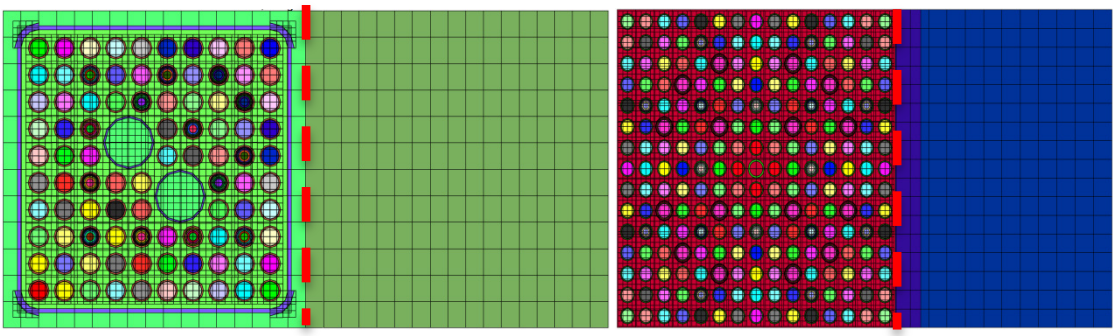
- It can be useful to run some KENO Monte Carlo simulations.
- CE-KENO vs. MG-KENO: validates the cross-section processing and multigroup library assumptions.
- MG-KENO vs. NEWT: validates grid structure and quadrature.
- A KENO-VI model is relatively simple to generate from an existing TRITON model.
- If you are working with a BWR, it is likely that you should calculate Dancoff factors using MCDANCOFF—you will need a KENO-VI model for this.

How do I generate reflector cross sections for PARCS using SCALE/TRITON?

- We recommend modeling at least one assembly width for a reflector.
- The model should include an actual fuel assembly to provide a neutron source.
- Axial reflectors can be modeled as a homogenized (smeared) region.
    - Sometimes the exact geometry of the axial reflector region is complex and three-dimensional.
    - Often the shape of structural materials in the axial reflector is not exactly known or not given.
    - As long as the volume and density (or mass) of materials are given, you can construct a homogenized region.
- The radial reflector baffle/shroud should be modeled as close to reality as possible.
    - Especially in PWRs—the baffle is close to the assemblies.
- You will need to specify another homogenize block with the reflector materials.
    - Goes before the fuel assembly homogenize block.
- A user will also need a special "read adf" block with an ADF for the fuel-reflector interface (red line in fuel+reflector plots on next page).

```
read homog
 2 RadRef 999 998  end
 1 21C           12  13      15  16  17  18
…
end homog
read adf
 2  1 2  e=10.90549
end adf
```

- In the "read adf" block, the first three numbers specify that:
    - This is a reflector ADF (2).
    - Identifying number of the fuel region (1).
    - Identifying number of the reflector region (2).
- Reflective north, south, west boundaries; vacuum east boundary.
- Use special input for GenPMAXS that specifies a corner reflector—hint: look for SFAC in the GenPMAXS manual.
    - SFAC is a variable that the user must calculate; the equation can be found in the GenPMAXS manual.
    - SFAC is only needed for corner reflectors.



The fuel bundle I am modeling is very heterogeneous, is there anything I need to look out for?

- For very heterogeneous fuel bundles (example: SVEA bundles), it is possible that even with very detailed Dancoff factor treatment for each fuel pin, the bias from KENO-CE calculations could be large.
- For high in-channel void conditions for BWR fuel bundles that have many saturated liquid water features, the library collapsing bias when using Parm=weight can be larger than 100 pcm. This is typically due to inadequate self-shielding of fuel pins due to the solid liquid water features.
- Similarly larger biases can also be seen for BWR lattices at high in-channel void fractions whose burnable absorber pins (gadolinia-bearing pins) are in close proximity to other burnable absorber pins. The self-shielding portion of the calculation can be inadequate due to the neighboring burnable absorber pins, resulting in a larger-than-normal bias.
- When generating nodal data for these cases, extrapolating lower-void data to higher-void conditions can result in nodal data that more closely corresponds to KENO-CE calculations than explicitly generating the data using SCALE/TRITON. For example, generating nodal data at 0%, 40%, and 80% void and extrapolating to 100% void can result in nodal data that is more accurate than explicitly running branch calculations for 100% void with SCALE/TRITON.

Is there anything else?

- Watch out for file size.
    - If there are many branches and many depletion steps, the output files can get very large, especially if the user is keeping or reusing the temporary directory. After the run is

complete, the user might want to delete the _tot00000001 file in the temporary directory—it is a copy of the output file.

  - If you have flux plots turned on (prtflux=yes) with many branches, your computer will try to copy many large postscript files—this can bog down your system.

- Use the postscript files—they can often give the user a clue about an error in the geometry.
- Check the results as they are being generated. If performing a material homogenization, the *txtfile016* file contains eigenvalues for every branch. A user can extract and plot these as needed.
- Have someone else check your work before running very long simulations. It is easy to make a mistake working with very large input files. Sometimes it is difficult to see your own mistakes because you are so familiar with the input files.

APPENDIX B

USING SCALE/MCDANCOFF FOR BWRS

Studies have shown that the infinite lattice approximation used in SCALE when calculating Dancoff factors for self-shielding calculations breaks down for some fuel pins in moderately high-void BWR lattices. The fuel pins of interest are located in the lattice corner, along the lattice edge, and near water rods. These pins are located near regions of solid liquid water, which greatly influence the neutron spectra seen by the pins. For this reason, the infinite lattice approximation is no longer valid. To this end, SCALE/MCDancoff (Monte Carlo Dancoff) was developed as a tool to calculate Dancoff factors using the Monte Carlo method. SCALE/MCDancoff is a modified version of SCALE/KENO-VI that uses a one-group neutron library to calculate Dancoff factors for requested materials in a model. SCALE/MCDancoff is a modified version of SCALE/KENO-VI, so the geometry setup is identical to SCALE/KENO-VI and very similar to SCALE/NEWT.

To use SCALE/MCDancoff to calculate Dancoff factors for the fuel pins in your model, first begin with an already developed SCALE/NEWT geometry model. Then follow these steps to convert the model to SCALE/MCDancoff or SCALE/KENO-VI:

- Redefine the sequence.
    - =t-depl or =t-newt to =mcdancoff or =csas6.
- Change XS library for MCDANCOFF.
    - v5-44, v5-238, v6-238, v7-238 => xn01.
- Add the *z*-axis.
    - Add +z and -z to all bodies. You should specify a *z*-axis corresponding to the length of the fuel bundle. +
    - Add nuz=1 terms in arrays.
    - Add z terms in array and hole placement.
- Remove all Aliases.
    - SCALE/MCDancoff is incompatible with aliases.
    - Expanded composition block can be copied and pasted from *input_t* file in SCALE temporary directory.
- Remove Depletion, Burndata, and Branch blocks.
    - Change the fuel mixtures in the geometry model so that aliases are not needed.
- Remove the Celldata block.
    - Leave the Celldata blocks to run SCALE/KENO-VI in multigroup mode, otherwise remove them.
- Change the Parameter block.
    - Remove all NEWT parameters.
    - Include KENO parameters.
- Remove spatial grid for cells and global unit.
- If your BWR model contains control rod blades, remove these from the model.
- Remove the SCALE/NEWT Material block.
- Add SCALE/KENO-VI plot data (not required, but highly recommended).

- Add a shape to enclose arrays placed in the global unit.
    - For SCALE/KENO-VI, all arrays placed in the global unit must have their own enclosing surface—your SCALE/NEWT models may or may not already have this.

Once geometry conversion is complete, you can use Parm=Check to check the model and generate plots of the geometry. Upon checking the model, you might encounter some geometry errors that you will need to resolve. Array placement errors are common to new SCALE/KENO-VI users—these errors usually arise from not having a bounding surface for the array in the global unit and not placing the array correctly inside the bounding surface. Detailed SCALE/KENO-VI geometry instruction can be found in the SCALE/KENO-VI primer and the SCALE/KENO-VI manual [SCALE manual, Section F17].

When the geometry runs through Parm=Check without errors, you can add the required input blocks for calculated Dancoff factors. There is no graphical user interface for MCDancoff, so you must add this data using the text-based input format. The input block that specifies the Dancoff factors that SCALE/MCDancoff will calculate is the `Start` block [SCALE manual, Section M24]. Inside the Start block, enter `dancoff` to signify the calculation of the Dancoff factor. Starting at the global unit, you must specify the `array` or `hole` that contains the unit for which a Dancoff factor will be calculated. You can use multiple instances of `array` or `hole` if your model has "arrays of arrays." Using the `array` and `hole` keywords, step down from the global unit until you have specified a unit. You must then specify the `region` that contains the mixture for which a Dancoff factor is calculated. This is best illustrated by an example:

```
read start
  dancoff
    array 1  4 2 1
    unit 10
    region 1
end start
```

This example begins with `read start`. On the next line, `dancoff` specifies that a Dancoff factor will be calculated. On the third line, `array 1  4 2 1` specifies the first array (array 1) in the global unit, and then specifies array position (x,y,z) = 4 2 1; `unit 10` on the fourth line is the unit contained in array position (4,2,1). The fifth line, `region 1`, specifies the first region (first media statement) of `unit 10`. The `end start` line signifies the end of the Start input block for the Dancoff factor calculation. Again, `region 1` is the first region specified by a media statement in `unit 10`, which is array position `4 2 1` in `array 1`, which is the first array listed in the global unit. The `dancoff` keyword and subordinate `array`, `hole`, `unit`, and `region` keywords can be repeated as many times as necessary to specify a mixture.

With the SCALE/MCDancoff model complete, you can run the case from the command prompt like any other SCALE input file. SCALE/MCDancoff will run a Dancoff factor simulation for every `dancoff` specified in the Start block with as many particles and generations as specified in the Parameters block. You do not need to specify a great amount of particles and generations to obtain adequate statistics for Dancoff factors. If the error of the Dancoff factors are ~0.01 or less, the Dancoff factors should be adequate. When the SCALE/MCDancoff run completes, you will have an inputfile.danc* file for each `dancoff` specified in the Start block. This input file contains the unit name and Dancoff factors for all the isotopes in the fuel material. You should use the Dancoff factor for U-235 (92235) in subsequent SCALE/TRITON calculations. The following is a sample output file from a SCALE/MCDancoff case that used the previous Start block example:

```
Unit 10 at global x -5.62500E+00  y -5.62500E+00  z  0.00000E+00
     index        nuclide        dancoff      deviation
         1          92235    1.41663E-01    2.24232E-03
         2          92234    1.41663E-01    2.24232E-03
         3          92236    1.41663E-01    2.24232E-03
         4          92238    1.41663E-01    2.24232E-03
         5           8016    9.59918E-01    3.68866E-04
```

For subsequent calculations, a user would specify dan2pitch(N)=1.41663E-01 for the CENTRM latticecells for fuel pins in a similar lattice position. Remember that different Dancoff factors should be used for corner, edge, and central lattice fuel pins—using the same Dancoff factor for all fuel pins is not adequate.

It is recommended that you use SCALE/MCDancoff to develop a Dancoff factor map for every fuel pin. This usually means that you need to calculate separate Dancoff factors for every fuel pin for half of the lattice. Once you have a Dancoff factor map, you can more clearly see where you can group fuel pins with similar Dancoff factors, as seen in Fig. B.1.

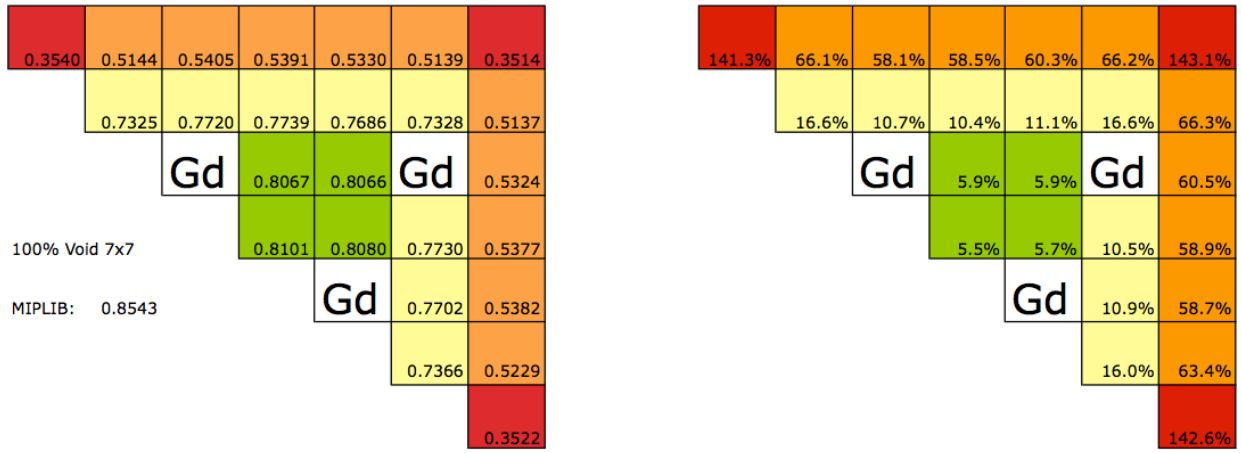


Fig. B.1. Dancoff factor map for a 7×7 fuel lattice. The map on the left shows the SCALE/ MCDancoff-calculated Dancoff factors, and the map on the right shows the relative difference between the infinite lattice Dancoff factor and the SCALE/MCDancoff-calculated Dancoff factor.

You should already have separate latticecells for all fuel pins that have different enrichment or gadolinium content. Once you have used SCALE/MCDancoff to calculate all the required Dancoff factors, you should add latticecell calculations for fuel pins that have the same enrichment but different Dancoff factors. Generally, fuel mixtures should be lumped in two levels—one level for fuel composition, and another level for Dancoff factor. Generally, the Dancoff factors for corner pins are similar, along with the Dancoff factors for fuel pins along the lattice edge. This fuel lumping strategy is illustrated in Figs. B.2 and B.3.

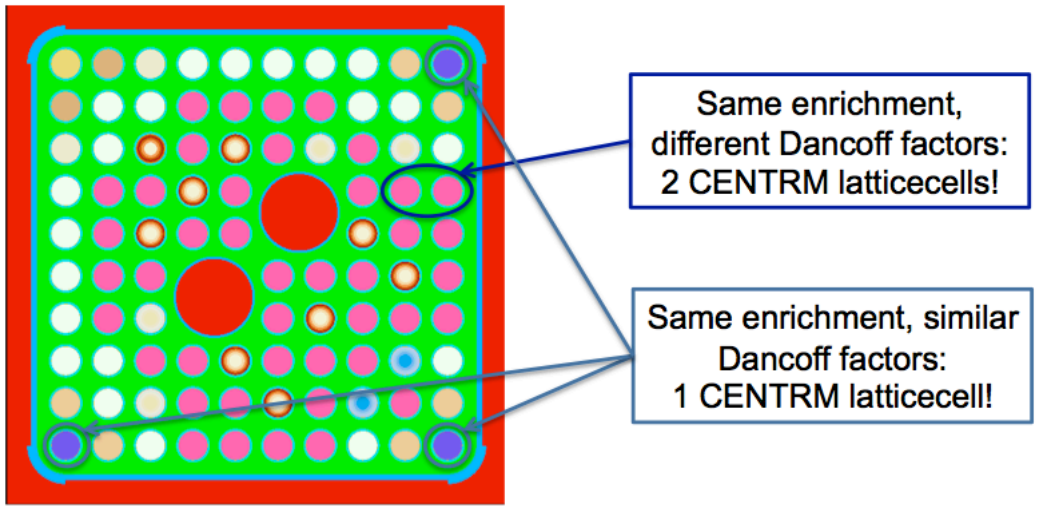


Fig. B.2. Illustration of the fuel lumping strategy with different Dancoff factors.

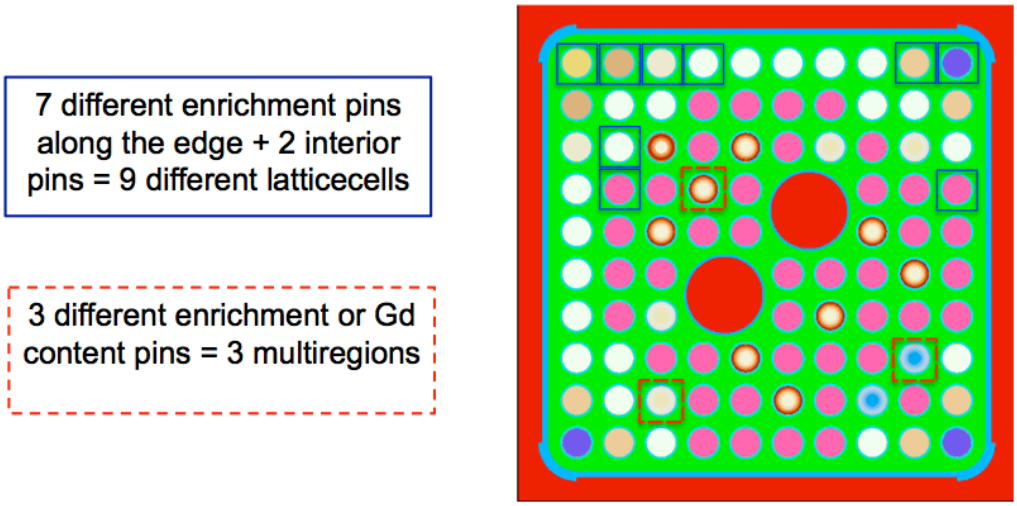


Fig. B.3. Illustration of the total number of latticecell and multiregion statements that are needed for a complex BWR model.

APPENDIX C

INSTALLING GHOSTSCRIPT AND GSVIEW ON WINDOWS

GhostScript is freeware that can interpret PostScript language and the Adobe Portable Document Format (PDF), meaning that GhostScript can read PostScript or PDF files and display the results. SCALE/NEWT displays the model geometry in the form of two PostScript files: filename.newtgrid.ps and filename.newtmatl.ps. To view these files, the user must have PostScript viewing software installed on their computer. GSview is the Windows graphical user interface for GhostScript. Although not required, installing GSview is highly recommended for Windows users.

To download GhostScript and GSview, navigate to http://pages.cs.wisc.edu/~ghost/ using a web browser. Download the latest version of Ghostscript; for this example the most recent version is GhostScript 9.00 (Fig C.1)

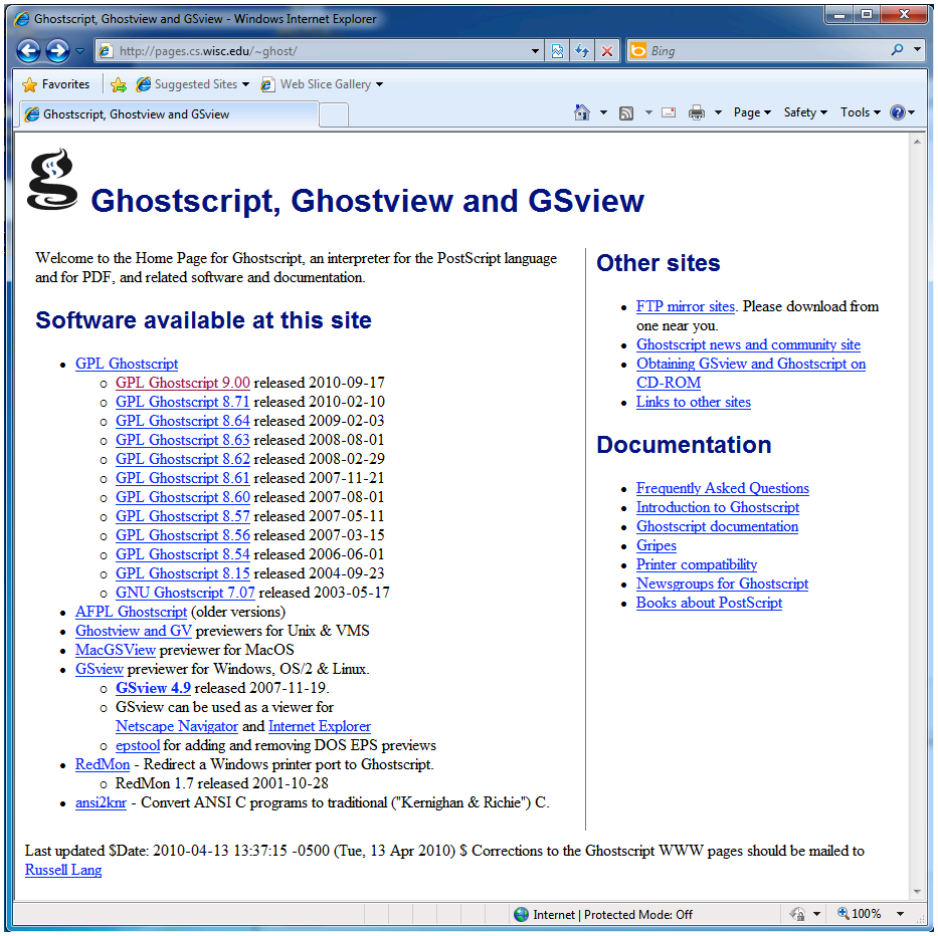


Fig. C.1. Ghostscript, Ghostview, and GSview download page.

Scroll to the bottom of the page and download the latest Microsoft Windows executable for GhostScript 9.00 (Fig. C.2)

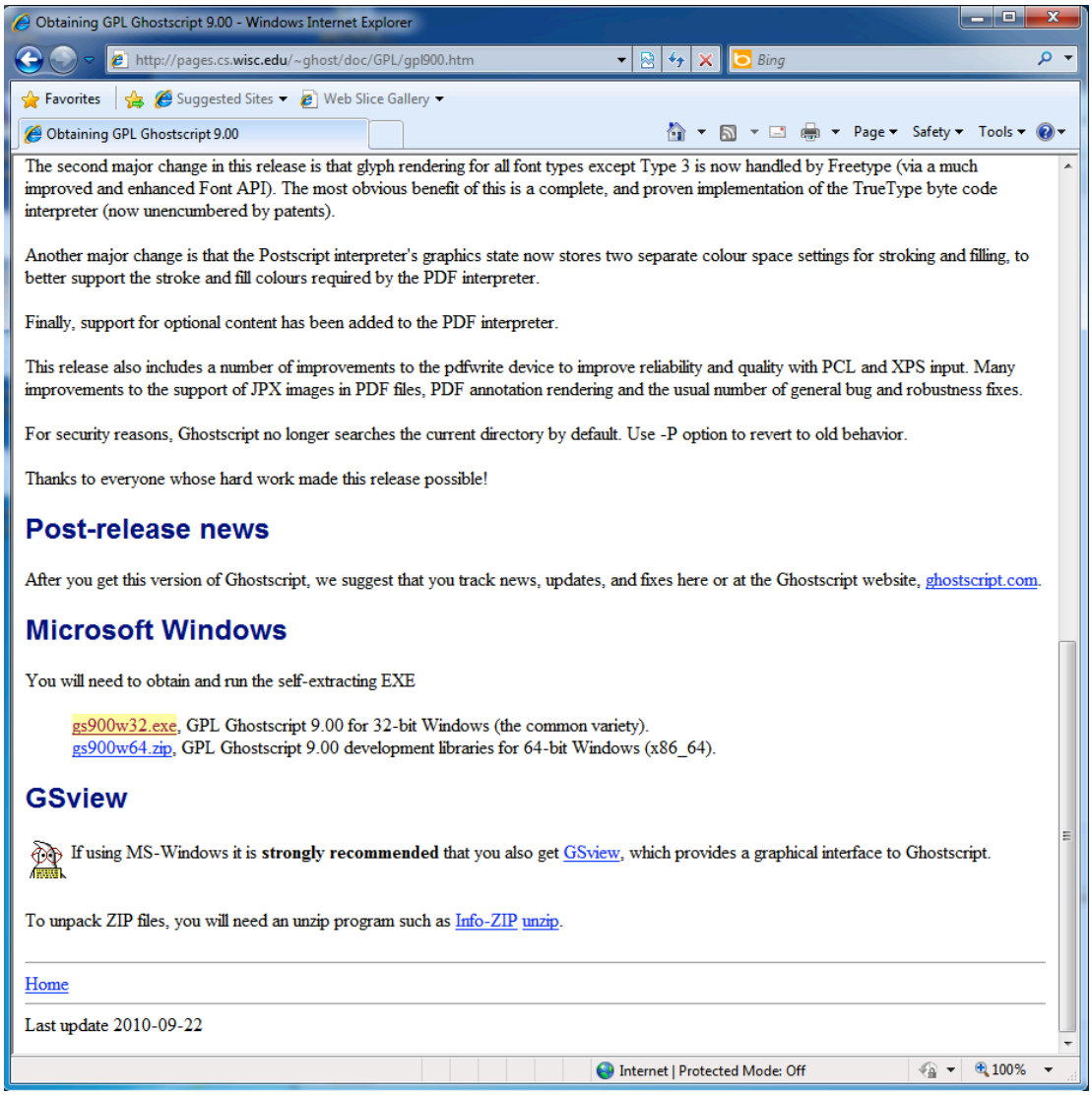


Fig. C.2. GhostScript 9.00 download page.

After clicking the link to the self-extracting executable, the **File Download** window will appear; choose **Run** (Fig. C.3).

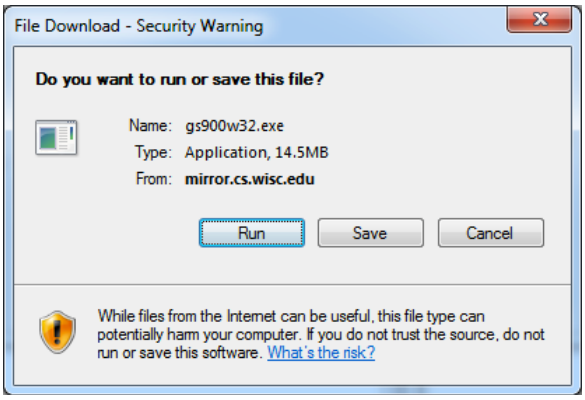


Fig. C.3. File Download window for the GhostView executable.

Click **Run** again on the **Internet Explorer – Security Warning** window (Fig. C.4).

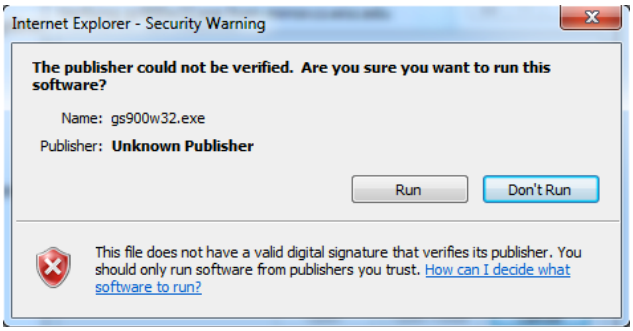


Fig. C.4. Internet Explorer – Security Warning window.

The self-extracting executable will then open; click **Setup** from the **WinZip Self-Extractor** window (Fig. C.5).

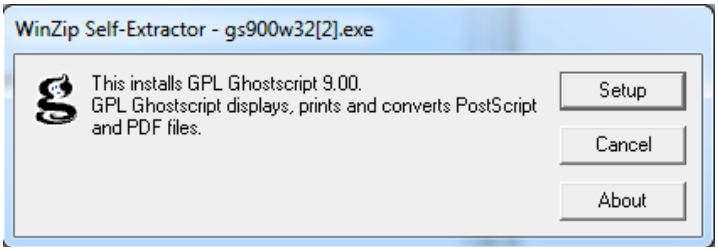


Fig. C.5. WinZip Self-Extractor window.

The **GPL Ghostscript Setup** window will appear; click **Install** on this window (Fig. C.6).

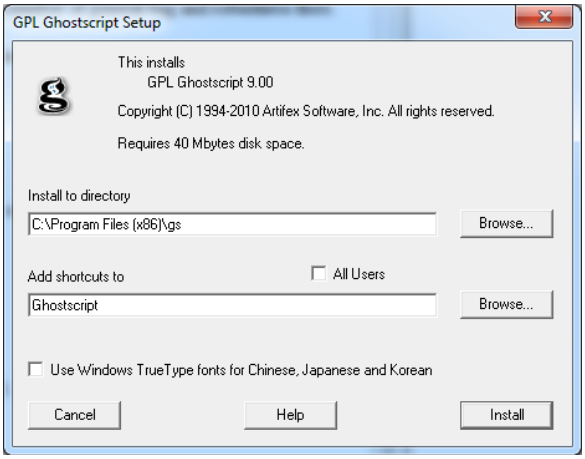


Fig. C.6. Ghostscript setup window.

Clicking **Install** runs software to install Ghostscript on your computer.

Now return to the web browser and go back to the main Ghostscript, Ghostview, and GSview page and choose to install the most recent version of GSview (Fig. C.1). The GSview download page will open. Choose the applicable Windows self-extracting archive (Fig. C.7).

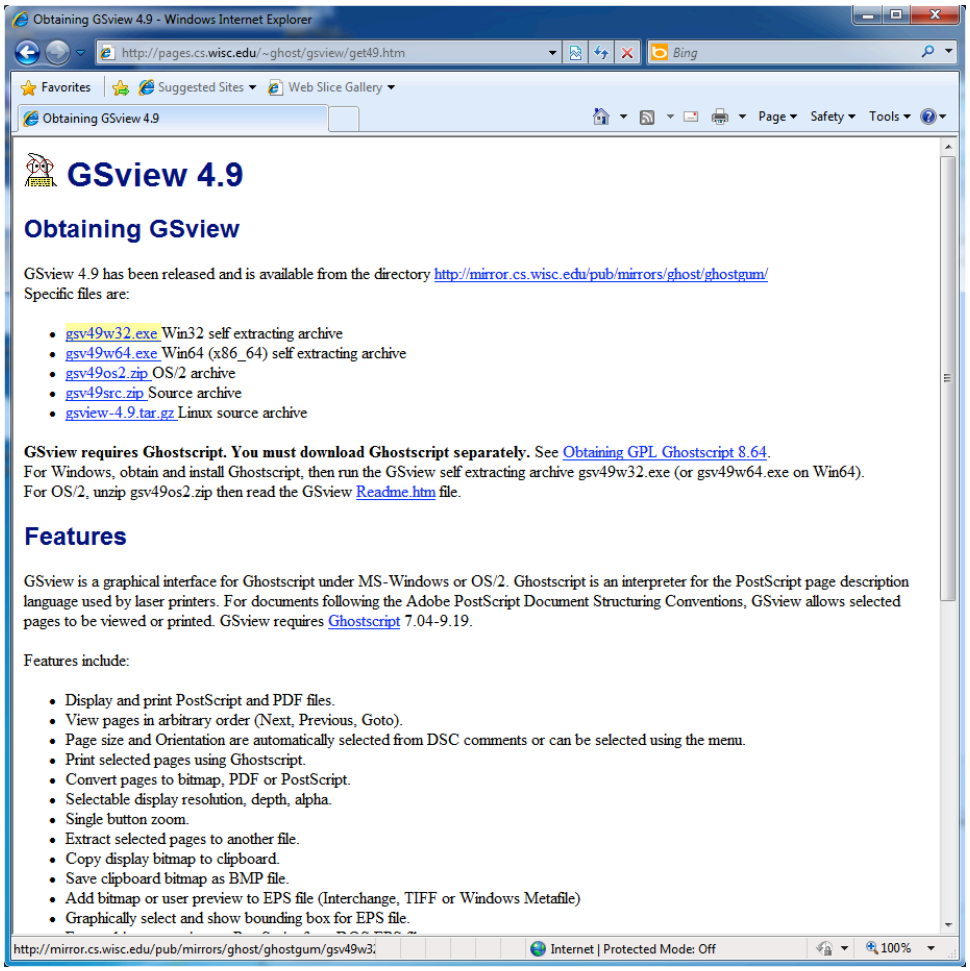


Fig. C.7. GSveiw download page.

The **File Download – Security Warning** window will again appear (Fig. C.3); again, click **Run**. The **Internet Explorer – Security Warning** will appear, and again choose **Run** (Fig. C.4). As before, click **Setup** on the **WinZip Self-Extractor** window (Fig. C.5). The **GSview Install** window will appear; follow the steps to install GSview (Fig. C.8).

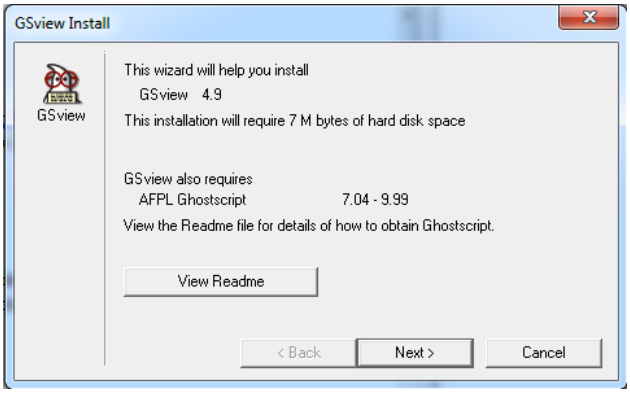


Fig. C.8. GSview Install window.

Choose to associate PostScript files with GSview from the GSview Install window (Fig. C.9).

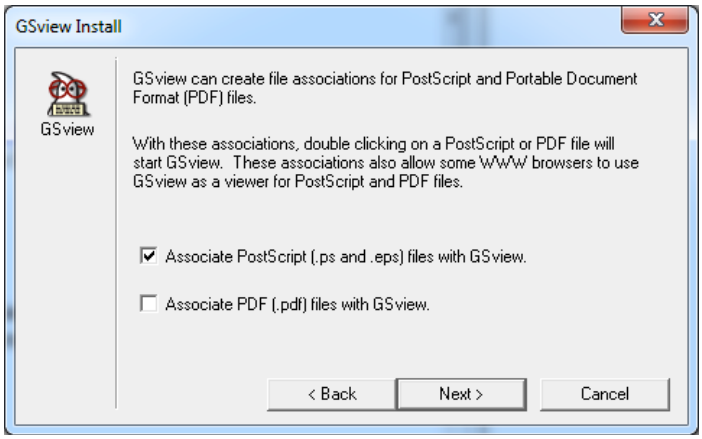


Fig. C.9. GSview Install window – setup options.

Choose a directory to install GSview and click **Next**; the default directory was chosen here (Fig. C.10).

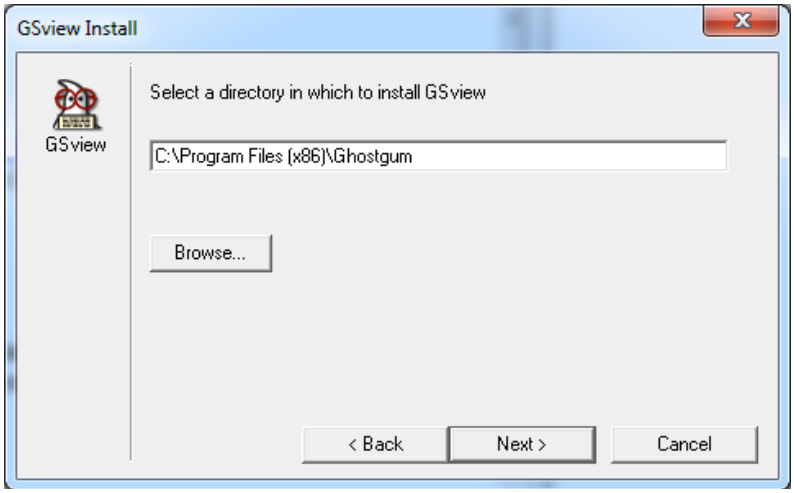


Fig. C.10. GSview install window – install directory.

Choose the defaults on the GSview install window for shortcut options (Fig. C.11).

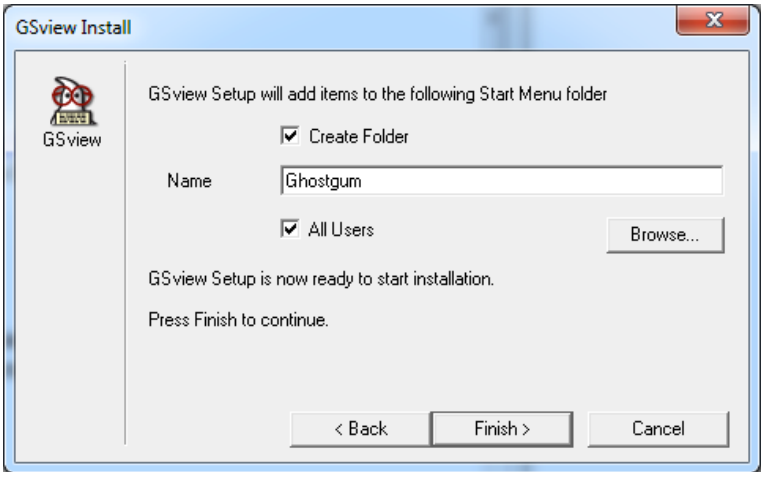


Fig. C.11. GSview Install window – shortcut options.

Once you click Finish, you should see that GSview has installed correctly. You are now ready to begin using GhostScript and GSview to view postscript files.

APPENDIX D

OVERVIEW OF GENPMAXS FOR LWR APPLICATIONS

GenPMAXS (Generation of Purdue Macroscopic XS) is the interface between lattice physics codes and PARCS core simulator. This document focuses on using GenPMAXS as the interface between SCALE/TRITON and PARCS. GenPMAXS reads the SCALE/TRITON-generated *xfile016* file and generates partial derivatives that are printed in the PMAXS file. Figure D.1 shows a basic flowchart of the interaction of SCALE/TRITON, GenPMAXS, and PARCS.

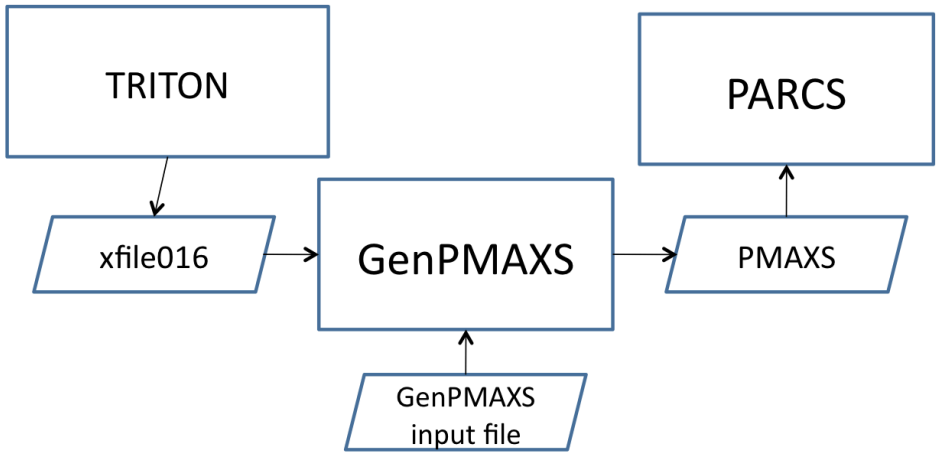


Fig. D.1. SCALE/TRITON-GenPMAXS-PARCS flowchart.

Using lattice physics input files, SCALE/TRITON generates *xfile016* files that contain cross-section data for the nominal condition and branch conditions. After completion of the SCALE/TRITON runs, GenPMAXS is used to read a single or multiple *xfile016* files, to process the cross-section data contained in the *xfile016* files, and to write the partial derivatives in the PMAXS file. The PMAXS file(s) are then read by PARCS during the full-core simulation.

Choosing branch states for the SCALE/TRITON simulation can be difficult. Different branch states are needed for BWRs than for PWRs, and different branches are needed for steady-state than for transient analyses. Nodal core simulators interpolate between a set of data points to obtain nodal parameters for instantaneous operational conditions. For this reason, a branch condition does not need to be specified for every possible operating condition. Rather, the envelope formed by the branch conditions should cover possible operational conditions that the user plans to simulate. If a core simulator predicts an operating condition between two branch conditions, the nodal parameters are interpolated based on available data. If a core simulator predicts an operating condition that lies outside the envelope of branch conditions, the nodal parameters are extrapolated to that condition based available data. Generally speaking, extrapolation of nodal parameters is particularly risky, so all foreseeable operating conditions should lie inside envelope formed by the chosen branch conditions. Users must specify an adequate number of branch conditions in order for PARCS to accurately interpolate between conditions.

In addition to specifying an adequate number of branch states, users must also take care to ensure that the set of branch conditions is valid for GenPMAXS. For GenPMAXS to properly calculate partial derivatives for the PMAXS file, the set of branches must be "orthogonally connected." Orthogonally connected means that for any state condition requested by PARCS, there must be an orthogonal path in the branch states back to the nominal state. This can be better understood with the illustration in Fig. D.2.

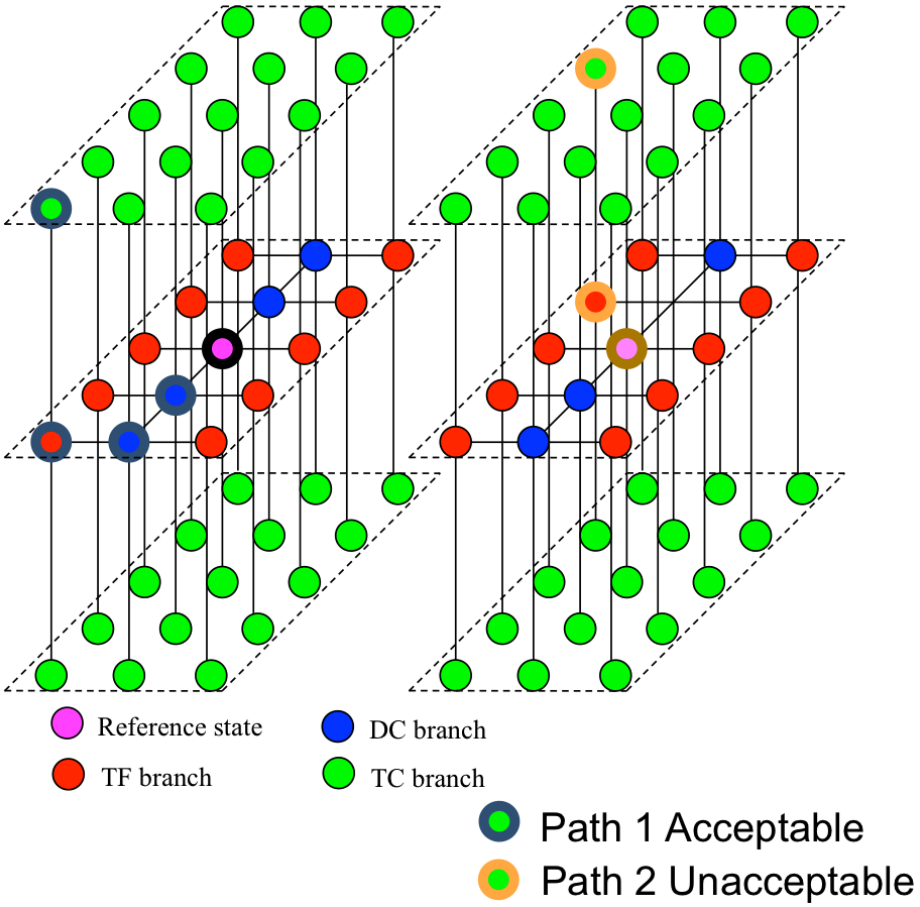


Fig. D.2. Acceptable (orthogonal) and unacceptable (nonorthogonal) paths from reference state to branch state.

In Fig. D.2, the path on the left is acceptable because the TC branch can be traced through the branch states along orthogonal paths to the reference state. The path on the right of Fig. D.2 is unacceptable because the required DC branch has not been calculated. In the unacceptable case, PARCS would have to simultaneously interpolate on two different state variables (DC and TF), and therefore, the path is nonorthogonal. In short, each branch state must only perturb one variable at a time. After the desired branch structure has been determined and before running the full SCALE/TRITON lattice calculations, it is recommend that users generate a simple TRITON model (e.g., simple pin-cell with two depletion steps) to test the branch structure in GenPMAXS. Use the *xfile016* file from the simple test case and run it through GenPMAXS to make sure there are no errors.

D.1 How to Run GenPMAXS

Running GenPMAXS is very straightforward. You should have the SCALE/TRITON *xfile016* files and the GenPMAXS input files in the working directory. A sample of a GenPMAXS input file for a SCALE/TRITON *xfile016* file can be found here:

```
%JOB_TIT
 'PWR1.PMAX'  F   3.0 !PWR1 PMAXS file
%JOB_OPT
  T  T  F  F  F  F  T  F  F  F  F  F  F  F   1
!ad,xe,de,j1,ch,Xd,iv,dt,yl,cd,gf,be,lb,dc,ups
%DAT_SRC
    4   1   1   1.0
%FIL_CNT
   1  'pwr1.x16' 1 1
%JOB_END
```

Users should specify the name of the final PMAXS file (`PWR1.PMAX`) and the name of the *xfile016* file (`pwr1.x16`). The number `4` under `%DAT_SRC` in the input files signifies that the cross-section data is in SCALE/TRITON *xfile016* format. To run this input file from the working directory, type the full path to the GenPMAXS executable followed by the GenPMAXS input filename, as seen in Fig. D.3. In this example, the GenPMAXS executable lies in the working directory.

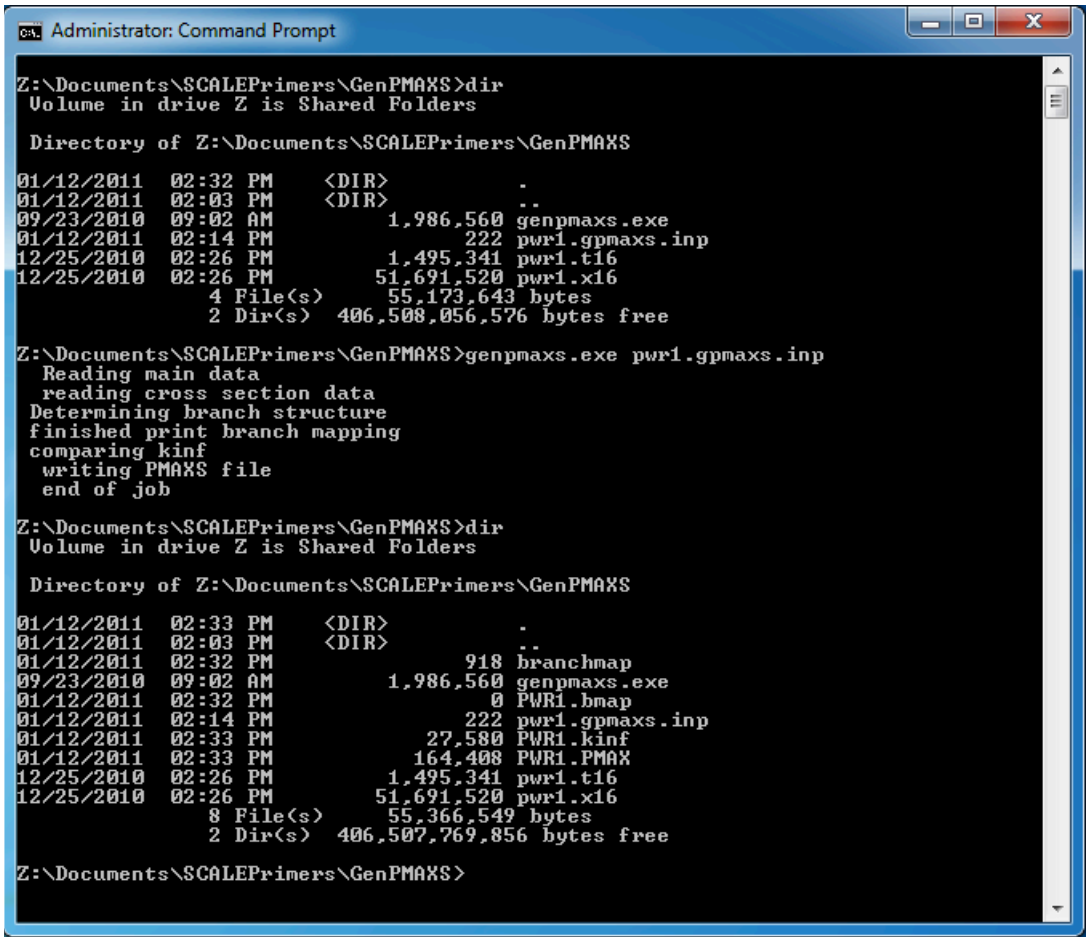


Fig. D.3. GenPMAXS I/O for Windows.

Upon completion, GenPMAXS will write the PMAXS file—in this case, PWR1.PMAX. GenPMAXS also generates PWR1.bmap and PWR1.kinf files. Of particular interest is the PWR1.kinf file; this file

contains a comparison of the SCALE/TRITON-calculated k_{inf} and the GenPMAXS-calculated k_{inf} that is based on the two-group cross sections. A sample of the PWR1.kinf file can be found here:

```
history    1    0.00000         0.70000         0.00000      900.00000
500.00000
 branch    1    0.00000         0.70000         0.00000      900.00000
500.00000
 burnup(GWD/MT),k-output,     k-XS,     difference(pcm)
    0.000000    1.139146    1.139179         3.30
    0.023037    1.112268    1.112293         2.48
    0.218852    1.104258    1.104281         2.26
    0.564407    1.103788    1.103811         2.24
    0.909962    1.105744    1.105767         2.30
    1.255517    1.108002    1.108026         2.39
    1.601072    1.110175    1.110199         2.44
    1.946627    1.112289    1.112314         2.53
    2.292182    1.114386    1.114412         2.60
    2.637737    1.116527    1.116554         2.72
```

The last column of the output file is the difference (in pcm) between *k*-output (SCALE/TRITON) and *k*-XS (two-group cross sections). It is common to observe differences of a few to 100 pcm in the difference column. Differences larger than 100 pcm could indicate an error somewhere in the calculation. The two most likely sources for the difference are not having all lattice materials in the homogenize block in the TRITON input file, and having the second value (Lxes, or xe) of the `%JOB_OPT` card in the GenPMAXS input file set to "F." This option controls the macroscopic xenon and samarium cross sections and will cause a difference between *k*-output and *k*-XS on the order of 2000 pcm. If the k_{inf} difference between *k*-output and *k*-XS is large, GenPMAXS prints a warning to the screen.

D.2 Branch Structures

To reiterate a previous statement, it is possible to generate a nonorthogonal branch structure that will fail for GenPMAXS. The pitfall is created when a user specifies a branch that changes two states simultaneously. Users could be tempted to specify multiple changes in states for a branch because it requires fewer branches to develop a perceived operating condition envelope, and therefore requires less computer time to generate cross sections. Users will find, however, that this practice will result in errors.

For example, for PWRs, users generally need branch states for coolant density and soluble boron concentration. In this fictitious example, assume the user needs states at a moderator density of 0.7 g/cm^3 with 2000 ppm boron, 0.8 g/cm^3 with 900 ppm boron, and 0.9 g/cm^3 at 900 ppm boron. For these states the fuel temperature is constant at 900 K; the moderator temperature is constant at 500 K, and control rods are out. The orthogonal branch structure for these states would be:

```
Tf=900     tm=500     dm=0.8    sb=900     cr=0
Tf=900     tm=500     dm=0.9    sb=900     cr=0
Tf=900     tm=500     dm=0.7    sb=900     cr=0
Tf=900     tm=500     dm=0.7    sb=2200    cr=0
```

The nonorthogonal branch structure would be:

```
Tf=900     tm=500     dm=0.8    sb=900     cr=0
Tf=900     tm=500     dm=0.9    sb=900     cr=0
```

```
Tf=900      tm=500      dm=0.7     sb=2200     cr=0
```

The nonorthogonal branch structure uses one less branch but modifies both the moderator density and the soluble boron concentration simultaneously in comparison to the other branches. Using a nonorthogonal branch structure can produce unexpected behavior when trying to convert *xfile016* files to PMAXS format using GenPMAXS.

NRC FORM 335
(12-2010)
NRCMD 3.7

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER
(Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, if any.)

NUREG/CR-7041
ORNL/TM-2011/21

2. TITLE AND SUBTITLE

SCALE/TRITON Primer: A Primer for Light Water Reactor Lattice Physics Calculations

3. DATE REPORT PUBLISHED

MONTH	YEAR
November	2012

4. FIN OR GRANT NUMBER

V6098 (former N7098)

5. AUTHOR(S)

B. J. Ade

6. TYPE OF REPORT

Technical

7. PERIOD COVERED (Inclusive Dates)

05/1/10 – 09/30/12

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)

Oak Ridge National Laboratory
Managed by UT-Battelle, LLC
Oak Ridge, TN 37831-6170

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above", if contractor, provide NRC Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address.)

Division of Systems Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

I. Frankl, NRC Project Manager

11. ABSTRACT (200 words or less)

The SCALE (Standardized Computer Analyses for Licensing Evaluation) computer software system developed at Oak Ridge National Laboratory is widely used and accepted around the world for many radiation transport applications. This primer focuses on use of the NEWT (New ESC-Based Weighting Transport) transport solver with SCALE/TRITON to generate cross sections for light water reactor nodal simulators. The primer is based on SCALE 6.1, which includes the Graphically Enhanced Editing Wizard (GeeWiz) Windows user interface. Each example in this primer uses GeeWiz to provide the framework for preparing input data and viewing output results. Beginning with a Quickstart section, the primer gives an overview of the basic requirements for SCALE/TRITON input and allows the user to quickly run a simple pin cell simulation with SCALE/TRITON. The sections that follow the Quickstart include a list of basic objectives at the beginning that identifies the goal of the section and the individual SCALE/TRITON features that are covered in detail in the sample problems in that section. Upon completion of the primer, a new user should be comfortable using GeeWiz to set up 2-D lattice physics problems in SCALE/TRITON. The primer provides a starting point for the reactor engineer who uses SCALE/TRITON for lattice physics. Complete descriptions are provided in the SCALE/TRITON manual. Although the primer is self-contained, it is intended as a companion volume to the SCALE/TRITON documentation—the SCALE manual is provided on the SCALE installation DVD. The primer provides specific examples of using SCALE/TRITON for lattice physics analyses; the SCALE/TRITON manual provides detailed information on the use of SCALE/TRITON and all its modules. The primer also contains an appendix with sample input files.

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)

SCALE, lattice physics, reactor analysis, core, TRITON, NEWT, depletion

13. AVAILABILITY STATEMENT

unlimited

14. SECURITY CLASSIFICATION

(This Page)
unclassified

(This Report)
unclassified

15. NUMBER OF PAGES

16. PRICE

Federal Recycling Program

UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, DC 20555-0001

OFFICIAL BUSINESS

NUREG/CR-7041

SCALE/TRITON Primer: A Primer for Light Water Reactor Lattice Physics Calculations

November 2012