ANL/RTR/TM-24/25 Rev. 1

ADDER v1.1.0 User Guide

The Advanced Dimensional Depletion for Engineering of Reactors (ADDER) Software for Depletion and Fuel Management

Nuclear Science & Engineering Division

ANL/RTR/TM-24/25 Rev. 1

ADDER v1.1.0 User Guide

The Advanced Dimensional Depletion for Engineering of Reactors (ADDER) Software for Depletion and Fuel Management

prepared by
V. Mascolino, K. Anderson, C. Castagna, M. Elsawi, E. Wilson

Nuclear Science & Engineering Division, Argonne National Laboratory

July 2025

(This page left intentionally blank)

Abstract

The Advanced Dimensional Depletion for Engineering of Reactors (ADDER) software is being developed in the Research and Test Reactor (RTR) Program at Argonne National Laboratory to meet the reactor design and analysis needs of the Conversion Program. ADDER is a flexible tool that (1) provides a depletion capability through coupling external neutronics codes with a built-in CRAM solver or external depletion code and (2) provides a user-friendly interface to perform fuel management and criticality search operations. The ADDER software is a Python 3 application written using modern software development practices subject to a compliant implementation of NQA-1 and applicable Department of Energy software quality assurance standards. This report is the user guide for the software release referred to as ADDER v1.1.0. The motivation for a software to have flexible capabilities that ADDER possesses is the need to support a wide variety of geometries that are commonly required in analysis of research and test reactors. These reactors can have complex fuel, experiment, or control material shuffling patterns that persist over several years with many fuel management and partial refueling intervals. The scale of fuel management analysis can require tracking of an inventory that is multiple times the core loading. Many reactors, both power and non-power reactors of various types, will find the features of ADDER useful to facilitate key tasks that a fuel or core design engineer must perform with the convenience of concise input and validated functionality.

Table of Contents

List of Figures

(This page left intentionally blank)

1 ADDER Primer

The Advanced Dimensional Depletion for Engineering of Reactors (ADDER) software is a flexible, performant, and modern fuel management and depletion tool. The target audience of this tool is first the Research and Test Reactor (RTR) Program at Argonne National Laboratory (Argonne) and experts outside of Argonne. This report is the user guide for the software release referred to as ADDER v1.1.0.

This software is fundamentally an interface between the user, an external neutron diffusion or transport theory solver, and a depletion solver. The user will define the reactor with the necessary input to the neutron diffusion/transport solver, and ADDER will provide the user with the ability to deplete the reactor for a given power history, shuffle fuel in the core and load or remove fuel from the core, perform criticality searches, and stochastic volume computations. ADDER will eventually be able to perform other analyses necessary for the design of a reactor, such as branch calculations for multiple xenon conditions or operating temperatures.

At its current stage, ADDER utilizes the MCNP6 [1] (or MCNP5 [2]) software for neutron transport and either an internal Chebyshev rational approximation method (CRAM [4]) solver or ORIGEN2 [3] for depletion and decay functionality.

1.1 Installing ADDER

ADDER is written in the Python 3 programming language and requires that a Python 3.8 compatible interpreter be installed locally.

In addition to Python itself, ADDER relies on third-party packages. All prerequisites can be installed using Anaconda [5] (recommended), Python's pip command, or through the package manager in most Linux distributions.

Required

NumPy NumPy [6] is used extensively within the ADDER for its powerful N-dimensional array.

SciPy SciPy's [7] sparse linear algebra capabilities are used by ADDER's CRAM solver.

h5py h5py [8] provides Python bindings to the HDF5-formatted [9] depletion data library. The h5py package is needed to provide access to data within these files from Python.

Configobj Configobj [10] is a simple but powerful configuration file reader with syntax similar to the classic Windows .INI file format.

MCNPTools MCNPTools [11] includes binary utilities that facilitate common tasks or querying MCNP output files. ADDER uses MCNPTools to extract MCNP tallies.

pyparsing pyparsing [12] is used to parse strings in the initial MCNP input to retrieve information related to MCNP tally cards that are processed by ADDER.

Optional

pytest The pytest [13] framework is used for unit testing the ADDER code.

pytest-cov The pytest-cov [14] package is a plugin to pytest which reports code coverage of the ADDER test suite.

Matplotlib Matplotlib [15] is used to plot certain results for testing purposes.

These prerequisites, with the exception of MCNPTools, will be installed automatically from the pip repository when installing ADDER if they are not available already on the base Operating System or Anaconda environment.

Anaconda users can easily install all the required libraries and compilers for the installation of both ADDER and MCNPTools by setting up the following environment:

```
conda create -p <ENVS_FOLDER>/adder -c conda-forge python=3.8 make cmake compilers mpi
libtiff "h5py>=3.0=mpi*"
```

where `<ENVS_FOLDER>` is the path to the folder in which you want to create the conda environment (e.g., `~/envs`). Once the Anaconda environment is created using the command above, it can be activated by running the following command:

```
conda activate <ENVS_FOLDER>/adder
```

External users can find the ADDER source code at the following GitHub address: https://github.com/anl-rtr/adder. The repository can be cloned to a local Linux machine by running the following command:

```
git clone https://github.com/anl-rtr/adder.git
```

This will copy the ADDER root directory into the `adder` folder in the current working directory.

The installation of ADDER is performed by executing the following command from the root directory of the ADDER distribution/repository:

```
pip install .
```

To install ADDER for a single user, the following command should be executed by that user from the root directory of the ADDER distribution/repository:

```
pip install --user .
```

The ADDER executable, adder, can now be executed in the working directory of interest.

Note: If the command to execute Python 3 on the installation system is python3 instead of python, then pip in the above installation steps should be replaced with pip3.

1.1.1 Installing MCNPTools

To execute ADDER, users will need to compile and install MCNPTools. Users utilizing Anaconda as recommended above should make sure that the adder environment is active before performing the installation. The utility software is available with an open-source license from the GitHub repository located at: https://github.com/lanl/mcnptools. ADDER has been tested with MCNPTools version 5.3.1.

Users need to make sure to have a C++ compiler supporting C++11 features, the CMake tool set (version 3.21 or higher), and HDF5 (version 1.10.2 or above) installed on their system. The use of an Anaconda environment installed following the directions provided in the previous chapter is strongly recommended, as this ensures that all the dependencies necessary to install MCNPTools are in place.

The MCNPTools repository can be cloned from the GitHub repository using the following command:

```
git clone https://github.com/lanl/mcnptools.git
```

This will create a `mcnptools` folder in the current working directory. After navigating into the directory, users should run the following commands to compile and install MCNPTools:

```
cmake -D CMAKE_INSTALL_PREFIX=./install -D Python_EXECUTABLE=`which python3` .
cmake --build . --config Release -j 3
ctest --build-config Release
cmake --install . --config Release
```

Please refer to the MCNPTools documentation on the GitHub repository for further information about the software and its installation.

1.2 Executing ADDER

ADDER can be executed from any directory the user wishes. The simplest usage of ADDER includes providing one input argument, the name of the already-generated ADDER input file. The format of this input file is described in Section 4, Input File Format. Input flags are discussed in the block below.

In addition to the ADDER input file, an HDF5 file containing the depletion data library is utilized. The name of this file is specified by the *depletion_library_file* parameter in the ADDER input file. The depletion data library file format is described in the Section 6, Depletion Data Library.

When executed, ADDER creates the following output files:

1. `adder.log`: This is a continually-updated log of all messages from ADDER. Most messages in this file are echoes of the same information displayed to screen, however the adder.log file also includes additional information such as the configuration settings after validation and initial processing.

2. `results.h5` or an output file name as described by the *output_hdf5* parameter in the input

file: This file contains all results from the execution including the material inventories at each state point, the neutronics solver output files, etc.

3. Automatically-generated neutronics solver input and output files: ADDER leaves the neutronics solver input and output files generated during ADDER execution for easy verification and progress monitoring.

ADDER offers the following command line flags:

1. `-n or --no-deplete`: This flag instructs ADDER to perform fuel management operations but does not execute the neutronics or depletion solvers. This is used to produce typical fuel management output and logs which can be used to interrogate the names ADDER has applied to copied objects.

2. `-for --fast-forward`: This flag instructs ADDER to not execute neutronics calculations performed if the output files from previous ADDER runs are in the current working directory. When using this option, the user must be sure that the neutronics output files are consistent with the current case. Note that this only skips the neutronics calculations that are performed as part of deplete and geom_sweep operation blocks. This flag can be useful as a sort of restart capability, allowing the costliest computations to be skipped if they have already been performed in a previous ADDER execution.

3. `–h or --help`: This flag prints the above options and descriptions.

Finally, when using depletion solvers that must be interacted with via textual input and output files (currently only ORIGEN2), these inputs and outputs will be placed according to the following hierarchical rules:

1. The directory named by the TMPDIR environment variable.

2. The directory named by the TEMP environment variable.

3. The directory named by the TMP environment variable.

4. A platform-specific location:

   a. On Windows, the directories C:\TEMP, C:\TMP, TEMP, and TMP, in that order.

   b. On all other platforms, the directories /tmp, /var/tmp, and /usr/tmp, in that order.

5. As a last resort, the current working directory.

Understanding this hierarchy is useful for utilizing fast I/O resources such as RAM-based or local file systems as available. It should be noted that if the depletion solver fails to execute, ADDER will copy the working directory to ADDER's main execution directory for later user inspection.

1.3 ADDER Data Sources

Depletion solvers require knowledge of the transmutation chain, the reaction rate cross-sections, decay rates, and branching ratios. These are all defined for ADDER in a depletion library HDF5 file (discussed in Section 6, Depletion Data Library). This library can include either one-group or multigroup cross-section data. These cross-sections can either be used directly or the neutronics solver can be used to directly obtain the corresponding reaction rates in each depleting material. The former will lead to faster neutronics computations (with Monte Carlo solvers) while the latter will lead to a more accurate solution. This is controlled with the *use_depletion_library_xs* flag discussed later.

For simplicity of user input for feed and removal rates for molten-salt reactor (MSR) analyses, ADDER also uses isotopic natural abundances to expand user-provided elements. The natural abundances for this data are stored in the `adder/data.py` module. This data is sourced from the 2013 IUPAC Technical Report [16].

Note: If different natural abundance data is desired, the user must modify the `adder/data.py` file directly.

Next, atomic mass data is used internally by ADDER to convert any material concentrations provided in the neutronics input from weight percent to atom percent. This data is sourced directly from the `adder/mass16.txt` file. This file is an official ASCII reformatting of the Atomic Mass Evaluation 2016 (AME2016) data [17] from the IAEA's Nuclear Data Services. If the user wishes to modify this data, they simply need to modify/overwrite the adder/mass16.txt file.

Note: If different atomic mass data is desired, the user must modify the `adder/mass16.txt` file directly.

1.4 Tips to Navigate the ADDER User Guide

Although users are strongly encouraged to read the entirety of the User Guide, it is important to highlight the importance of specific sections to users who may need to acquire the minimum amount of information to run the software. One of ADDER's strongest suits is the ability to perform complex fuel management operations. These operations rely on a few concepts that need to be understood before setting up an ADDER input file. Users should thoroughly read Section 3, Fuel Management Methodology. Once the concepts discussed there are understood, users should refer to Section 4, Input File Format, for guidance on the ADDER input file structure and syntax. Section 5, Output Data Format, provides details about the structure of the output HDF5 file and how to navigate it.

Users who do not have access to an ADDER depletion library in the HDF5 format, should refer to Section 6, Depletion Data Library, for guidance on how to generate one.

Finally, users can find information for using specific neutronics software with ADDER in Section 7, Neutronics Solver Guidance, and are encouraged to familiarize themselves with the depletion calculations implementation by reading Section 2, Depletion Methodology.

1.5 ADDER Flowchart

A simplified flowchart for the ADDER software execution is depicted in Figure 1.1.

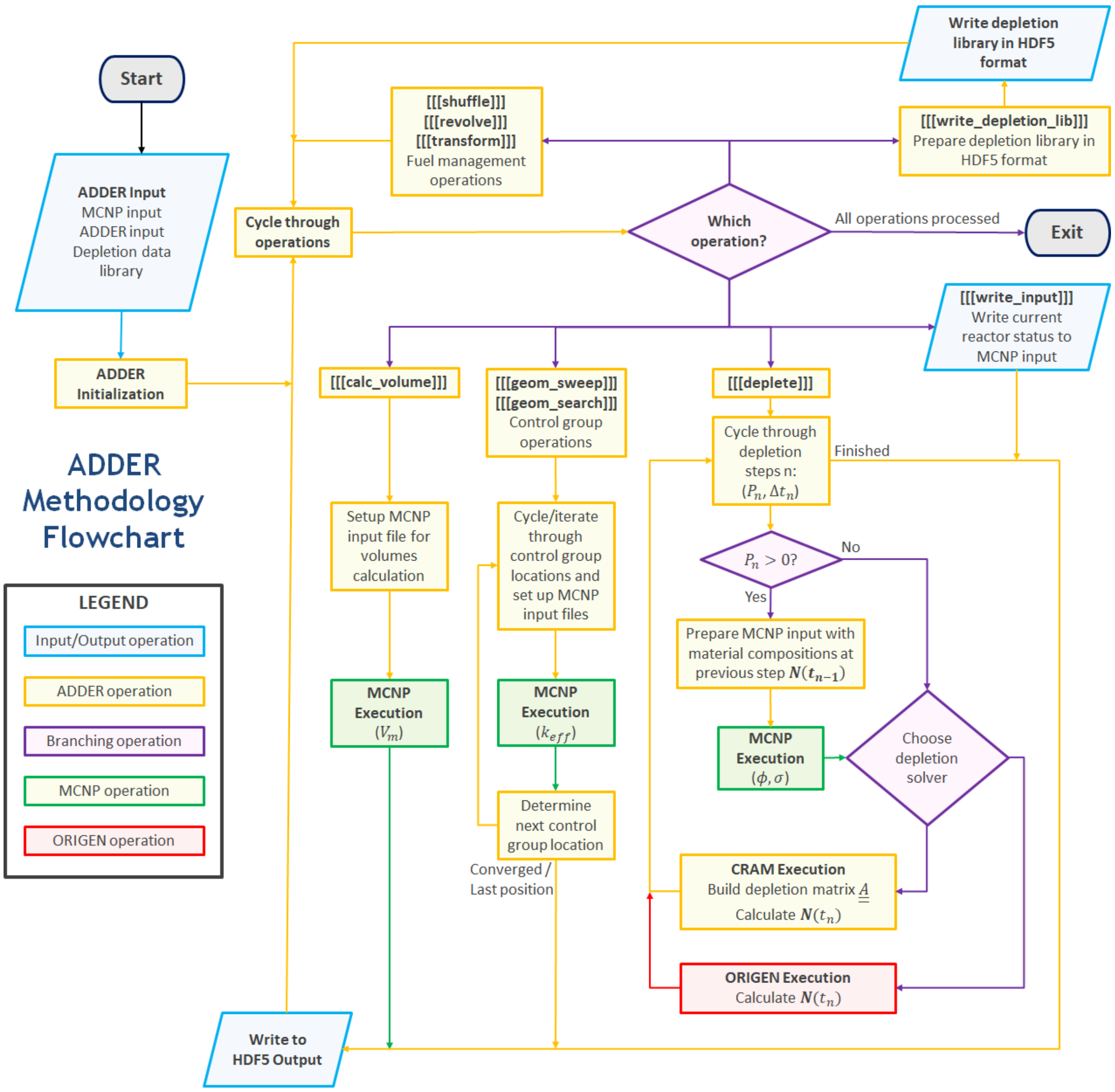
Start
ADDER Input
MCNP input
ADDER input
Depletion data library
ADDER Initialization
ADDER Methodology Flowchart
LEGEND
Input/Output operation
ADDER operation
Branching operation
MCNP operation
ORIGEN operation
Cycle through operations
[[[shuffle]]]
[[[revolve]]]
[[[transform]]]
Fuel management operations
Write depletion library in HDF5 format
[[[write_depletion_lib]]]
Prepare depletion library in HDF5 format
Which operation?
All operations processed
Exit
[[[write_input]]]
Write current reactor status to MCNP input
[[[calc_volume]]]
Setup MCNP input file for volumes calculation
MCNP Execution
(V_m)
[[[geom_sweep]]]
[[[geom_search]]]
Control group operations
Cycle/iterate through control group locations and set up MCNP input files
MCNP Execution
(k_{eff})
Determine next control group location
Converged / Last position
[[[deplete]]]
Cycle through depletion steps n:
$(P_n, \Delta t_n)$
Finished
$P_n > 0?$
No
Yes
Prepare MCNP input with material compositions at previous step $\boldsymbol{N}(\boldsymbol{t}_{n-1})$
MCNP Execution
(ϕ, σ)
Choose depletion solver
CRAM Execution
Build depletion matrix $\underline{\underline{A}}$
Calculate $\boldsymbol{N}(t_n)$
ORIGEN Execution
Calculate $\boldsymbol{N}(t_n)$
Write to HDF5 Output

Figure 1.1 ADDER simplified flowchart

2 Depletion Methodology

ADDER calculates the evolution of the material isotopics in every depleting region (as identified via the ADDER and neutronics software input files) by providing an interface between a neutronics solver and a depletion solver. The model reactor irradiation history is divided into several time steps, using the `[[[deplete]]]` operations (Section 4.7.3). In addition to the `[[[deplete]]]` operations, users can perform fuel management, control group, and data writing operations, as detailed in Section 4.7. This section discusses the methodology used by ADDER for performing the `[[[deplete]]]` operations.

It should be noted that the depletion methodology and equations that are presented throughout this Section and its subsections are based on the way the depletion problem is solved by ADDER's internal CRAM methodology. As such, they may be different if an external depletion solver, such as ORIGEN2 [3], is used. The equations presented here correspond to ADDER's implementation, hence users should refer to the User Guide specific to the depletion solver they are using to gain additional insight on the depletion algorithm and methodology. If an external depletion solver is used, ADDER prepares the necessary input files containing the material isotopics for the depleting materials and provides the path and library names to the necessary depletion and cross-section data libraries.

For each `[[[deplete]]]` operation, users need to define every irradiation step n in the irradiation history by providing the time step, Δt_n, and by setting either an average 1-group flux level for the system, $\overline{\phi}_n$, or the total power, P_n. The scaling factor for the flux normalization is calculated according to which option is selected. Both options are discussed in Section 2.2, Flux Normalization.

Each depleting region in ADDER is handled by identifying a separate material. ADDER is capable of automatically assigning a duplicated material to each separate region (e.g., cell in MCNP, see Section 7.1.1, Automatic Duplication of Materials), the material of which is identified as depleting. Since the ADDER implementation refers to materials rather than depleting regions, the term material is used in this methodology section and its subsections. As such, it is important for users to be aware that when referring to a material in this section (identified by the subscript m), it is in the ADDER sense. Hence, if the same fresh material is used in two (or more) different depleting regions in the base neutronics input file, these correspond to multiple separate ADDER materials that will be renamed and depleted separately. Additional information regarding how materials are handled by ADDER and what constitutes a material in ADDER is found in Section 3.1, Materials and Universes.

Each `[[[deplete]]]` operation can be expressed by the following set of depletion equations for each isotope, i, in each depleting material:

$$\begin{aligned} \frac{dN_{i,m}}{dt} &= -\left(\lambda_i + \sum_g \sum_x \sigma_{i,g,m,x}\,\phi_{g,m}\right) N_{i,m} \\ &+ \sum_j \left(\sum_g \sum_x \gamma_{i,j,x}\sigma_{j,g,m,x}\phi_{g,m} + f_{i,j}\lambda_j\right) N_{j,m} \end{aligned} \tag{1}$$

Equation (1) consists of the following parameters:

- $N_{i,m}$ is the atom density of isotope i in material m
- λ_i is the decay constant of isotope i
- $\sigma_{i,g,m,x}$ is the microscopic neutron (energy group g) cross-section of isotope i in material m for reaction channel x

- $\phi_{g,m}$ is the energy group g flux in material m, appropriately normalized as detailed in Section 2.2, Flux Normalization
- $\gamma_{i,j,x}$ is the yield of isotope i following the interaction of a neutron with isotope j through reaction channel x
- $f_{i,j}$ is the branch ratio for the radioactive decay of isotope j into isotope i

The Equation can be re-written in matrix form, as follows:

$$\frac{d\boldsymbol{N_m}}{dt} = \underline{\underline{A}}_m \cdot \boldsymbol{N_m} \tag{2}$$

where the matrix $\underline{\underline{A}}$ is the depletion matrix, the generic element of which is expressed as follows:

$$a_{m;\,i,j} = \begin{cases} \sum_g \sum_x \gamma_{i,j,x}\sigma_{j,g,m,x}\phi_{g,m} + f_{i,j}\lambda_j & i \neq j \\ -\left(\lambda_i + \sum_g \sum_x \sigma_{i,g,mg,x}\phi_{g,m}\right) & i = j \end{cases} \tag{3}$$

The parameters in the depletion matrix generic element, $a_{m;\,i,j}$, have been color-coded to better identify and distinguish the data sources for the depletion calculation, which are identified in Section 1.3, ADDER Data Sources. The color coding is explained in Section 2.1.1, Building the Depletion Matrix, which provides a more in-depth explanation of how the depletion matrix is built.

It should be noted that, although Equation (2) provides a general formalism for the depletion problem, external depletion solvers (such as ORIGEN2 [3]) may or may not use the exact same set of equations and the relevant user guide should be referred to when they are used.

The internal CRAM solver, based on the work from Pusa [4], directly solves Equation (2) via a numerical approximation by building the depletion matrix consistently with Equation (3). Additional detail about the CRAM algorithm is given in 2.1, Internal Algorithm for the Solution of the Depletion Equations, and its subsections.

2.1 Internal Algorithm for the Solution of the Depletion Equations

If the CRAM internal solver is selected in the ADDER input file as the depletion solver of choice, the depletion equation, Equation (2), is solved for every depletion time step defined within the `[[[deplete]]]` operations. This involves the following steps:

1. First, the depletion matrix, $\underline{\underline{A}}$, needs to be populated by ADDER. This includes pulling different data from various sources, including the neutron flux in every material from calculation using the neutronics solver, appropriately normalized.
2. After populating the depletion matrix, the depletion equation is solved using the internal CRAM algorithm. There are two options for solving one iteration (i.e., one depletion time step):
    a. Predictor method: the neutronics and depletion calculations are performed only once within the time step.
    b. Predictor/corrector method: the neutronics and depletion calculation are performed twice, using an intermediate solution as a correction to calculate the solution at the

end of the time step. In this case, the depletion matrix needs to be re-calculated after the predictor step to solve for the corrector.

Irrespective of the selected iteration scheme, for a given time step of duration $\Delta\tau$, starting at time τ_0, the depletion equation for material m, Equation (2), is solved as:

$$\boldsymbol{N_m}(\tau_0 + \Delta\tau) = \boldsymbol{N_m}(\tau_0) \cdot \exp\left(\underline{A}_m \Delta\tau\right) \tag{4}$$

To solve Equation (4), it is necessary to compute the matrix exponential, $\exp\left(\underline{A}_m \Delta\tau\right)$. Section 2.1.2, Implementation of the CRAM Method for Solution of the Matrix Exponential, provides additional information on how ADDER's internal CRAM solver, based on Pusa's work [4], computes the matrix exponential.

2.1.1 Building the Depletion Matrix

The depletion matrix is populated by ADDER at every depletion step if the internal CRAM solver is used (either in its predictor or predictor/corrector formulation). If the predictor/corrector method is used, the matrix is populated twice per depletion step, as discussed in Section 2.1.3, The Predictor and Predictor/Corrector Implementations.

Equation (3) highlights all the terms of the generic depletion matrix coefficient, $a_{m;\,i,j}$, for each material m. A generic physical interpretation of $a_{m;\,i,j}$ can be expressed as the effective probability per unit time for an isotope i to be produced as the result of the decay or transmutation of isotope j. If $i = j$, this is a negative value as there is actually a reduction of atom density for isotope i due to its decay or transmutation.

The terms in Equation (3) have been color-coded to better identify the data source for the various terms. Specifically, the data can be grouped as follows:

- Decay constants (λ_i), decay branch ratios ($f_{i,j}$), and neutron reaction yields ($\gamma_{i,j,x}$) are in green. These data do not change during the depletion calculation and are read directly from the HDF5 depletion data library used by ADDER (see Section 1.3, ADDER Data Sources). If the data library was generated from an ORIGEN2 library, these data correspond to that included in the decay library file (e.g., DECAY.LIB, see Section 6.1.1, ORIGEN2.2 Conversion).
- Neutron cross-section data ($\sigma_{i,g,m,x}$) are in blue. If the `use_depletion_library_xs` flag for the specific depletion step or for the entire calculation is set to `True`, then the cross-section data are taken directly from the HDF5 depletion library. If the data library was generated from an ORIGEN2 library, the data correspond to that included in the cross-section library file (e.g., PWRU.LIB). The latter can include several libraries, which can be selected in the ADDER metadata using the `depletion_library_name` parameter (see Section 4.2, Metadata). The number of neutron energy groups is consistent with the division of the specific library in the HDF5 depletion library file. If a `[[[deplete]]]` operation features multiple time steps, defined via the `powers` (or `fluxes`) and `durations` parameters (see Section 4.7.3, deplete), and `use_depletion_library_xs` is set to `False`, then the cross-section data are only calculated for the first of the irradiation steps. Any successive step defined in the same `[[[deplete]]]` (i.e., multiple `powers` or `fluxes` and `durations` parameters) operation uses the cross-section data as calculated during the first step.

- Neutron flux ($\phi_{g,m}$) is in yellow. The flux is calculated by ADDER using a transport solver, such as MCNP5 or MCNP6. The energy group division is consistent with that of the depletion library (identified by the `depletion_library_name` parameter).

The depletion matrix is then fed to the internal CRAM solver. The ORIGEN2 methodology for solving the matrix exponential is beyond the scope of this user guide, and users should refer to the ORIGEN2 manual [3]. The internal CRAM solver is discussed briefly in Section 2.1.2, Implementation of the CRAM Method for Solution of the Matrix Exponential, and is based from the work from Pusa et al. [4]

2.1.2 Implementation of the CRAM Method for Solution of the Matrix Exponential

Once the depletion matrix, as shown in Equation (3), is computed for the specific depletion time step or predictor step, the matrix exponential in Equation (4) is solved by a depletion solver. ADDER implements an internal CRAM solver based on the work from Pusa et al. [4] that can be selected by appropriately setting the `depletion_solver` parameter in the metadata section of the ADDER input file (see Section 4.2, Metadata). This section provides a very brief overview of the methodology, as implemented into ADDER; however, users are invited to read the work from Pusa for additional background information.

The matrix exponential, $\exp\left(\underline{\underline{A}}_m \Delta\tau\right)$, can be expanded as an infinite series as follows:

$$\exp\left(\underline{\underline{A}}_m \Delta\tau\right) = \sum_{k}^{\infty} \frac{1}{k!}\left(\underline{\underline{A}}_m \Delta\tau\right)^k \tag{5}$$

where $\underline{\underline{A}}_m$ is the depletion matrix and $\Delta\tau$ is the duration of the specific irradiation step or predictor step. Based on the above formulation, the solution of the depletion equation, Equation (4), can be approximated as:

$$\begin{gathered}\boldsymbol{N_m}(\tau_0 + \Delta\tau) = \boldsymbol{N_m}(\tau_0) \cdot \exp\left(\underline{\underline{A}}_m \Delta\tau\right) \approx \\ \alpha_0 \prod_{l=1}^{k/2} \left(\underline{\underline{I}} + 2\,Re\left\{\left(\underline{\underline{A}}_m \Delta\tau - \theta_l \underline{\underline{I}}\right)^{-1}\right\}\right) \cdot \boldsymbol{N_m}(\tau_0)\end{gathered} \tag{6}$$

In Equation (6), the following factors are identified:

- $\underline{\underline{I}}$ is the identity matrix
- k is the order of the CRAM solver
- α_0 and θ_l are coefficients that are given in [4]

ADDER implements the following two CRAM algorithm orders: 16 and 64. A higher CRAM order reduces the approximation error in the calculation of the matrix exponential at the cost of a higher computation time. The algorithm for implementing the computation of the matrix exponential, Equation (6), is identical to that implemented into the OpenMC Monte Carlo neutronics software [18]. The CRAM method has been shown to be particularly well suited for depletion problems and to provide accurate results even with large depletion steps [4].

2.1.3 The Predictor and Predictor/Corrector Implementations

ADDER implements two different time iteration schemes to solve for a depletion step, as formalized in Equation (4): a constant extrapolation (CE) predictor method and a more accurate (second order) but twice as expensive constant extrapolation, constant midpoint (CE/CM) predictor/corrector method.

In the CE implementation, if t_n is the time at the end of the n'th irradiation step, and h_n is the duration of the time step, then Equation (4) is re-written as:

$$\boldsymbol{N_m}(t_n) = \boldsymbol{N_m}(t_{n-1}) \cdot \exp\left(h_n \underline{\underline{A}}_m\right) \tag{7}$$

The CE algorithm requires the solution of a single transport calculation to calculate the flux and neutron cross-section data (as requested by the user) to compile the depletion matrix (as discussed in Section 2.1.1, Building the Depletion Matrix) and a single solution of the matrix exponential (as discussed in Section 2.1.2, Implementation of the CRAM Method for Solution of the Matrix Exponential).

The CE/CM algorithm effectively adds an intermediate step at the midpoint of the original time step and computes the material isotopics at the midpoint and at the end of the time step as follows:

$$\begin{aligned}
&\text{a.} \quad \boldsymbol{N_m^*}\left(t_{n-1/2}\right) = \boldsymbol{N_m}(t_{n-1}) \cdot \exp\left(\frac{h_n}{2} \underline{\underline{A}}_m\right) \quad &&\text{PREDICTOR} \\
&\text{b.} \quad \boldsymbol{N_m}(t_n) = \boldsymbol{N_m}(t_{n-1}) \cdot \exp\left(h_n \underline{\underline{A}}_m^*\right) \quad &&\text{CORRECTOR}
\end{aligned} \tag{8}$$

where $\underline{\underline{A}}_m^*$ is the corrected depletion matrix, evaluated by updating the flux and neutron cross-section data based on the isotopics calculated during the predictor step, $\boldsymbol{N_m^*}\left(t_{n-1/2}\right)$. This effectively entails the solution of two transport calculations to populate the predictor and corrector depletion matrices, as well as the solution of two matrix exponentials to solve for the predictor and corrector steps.

The implementation of the CE and CE/CM algorithms in ADDER is identical to that of the OpenMC Monte Carlo neutronics software [18].

2.2 Flux Normalization

ADDER utilizes an external neutron transport solver to calculate the neutron flux in every material. Currently, ADDER sets up the neutron transport calculations as criticality eigenvalue calculations, in which the fission source is iterated on to converge on the distribution that characterizes the critical system.

To obtain the neutron flux in physical units of `n/cm`^{2}`-sec`, it is necessary to normalize the flux tallies at each depletion step n. ADDER can perform this normalization in two different ways, based on which parameter is used to define the irradiation history in the `[[[deplete]]]` operation (see Section 4.7.3):

1. By setting the total reactor power, P_n, in the units of `MW` using the `powers` parameter.
2. By setting the average 1-group flux level, $\overline{\phi}_n$, in the units of `n/cm`^{2}`-sec`, using the `fluxes` parameter. Note that the provided flux level is intended as the flux averaged over all the *depleting* regions, not the entire system.

Note that the discussion below is relevant to the use of the MCNP software. Different normalization strategies will be added to this user guide as they become relevant for new ADDER interfaces that may be developed for alternative neutronics solvers.

For the normalization based on the total reactor power, it is important to be aware of the arbitrary normalization in the neutronics solver. MCNP normalizes the flux tallies per unit fission source [2]. As such, it is necessary to correlate the number of source fission neutrons to the energy released per unit time in the reactor (i.e., the reactor power). A more rigorous derivation of the normalization based on the reactor power can be found in [19].

The flux tallies scored by MCNP are normalized in such a way that the total number of generated fission neutrons (i.e., the fission source) at the end of one cycle is equal to the calculated k_{eff}. As such, under the assumption of a critical reactor, the MCNP tallies need to be divided by this value. Since MCNP normalizes its tallies to the fission source, they need to be multiplied by the expected total fission source of a critical reactor producing a thermal power equal to P_n. For each fission event, an average of $\bar{\nu}$ secondary neutrons are generated. If on average every fission deposits an energy equal to the average recoverable energy per fission, Q_{rec}, the total number of fission events per second is related to the total reactor power by the ratio P_n/Q_{rec}. As such, the total fission source per second corresponds to the product $\bar{\nu} \cdot P_n/Q_{rec}$. Based on the above, the scaling factor to multiply the MCNP-calculated fluxes can be calculated as:

$$C_P = \frac{P_n\,\bar{\nu}}{Q_{rec}\,k_{eff}} \tag{9}$$

The formula for calculating Q_{rec} is discussed in Section 2.2.1, Calculation of Average Recoverable Energy per Fission (Qrec). $\bar{\nu}$ is calculated as the ratio of the fission source divided by the fission rate for the entire system. These numbers are directly extracted from the MCNP output.

Alternatively, if the average 1-group flux level for the system, $\bar{\phi}_n$, is provided, the scaling factor used for normalizing the material fluxes becomes trivial, and is calculated as follows:

$$C_{\bar{\phi}} = \frac{\bar{\phi}_n \sum_m V_m}{\sum_g \sum_m \varphi_{g,m;\,MCNP} V_m} \tag{10}$$

where V_m is the volume of depleting material m and $\varphi_{g,m;\,MCNP}$ is the MCNP energy group g flux tally for material m.

Finally, for every energy group g in every material m, the flux in absolute units of `n/cm`^{2}`-sec` is calculated by ADDER as:

$$\phi_{g,m} = C\varphi_{g,m;MCNP} \tag{11}$$

where C, the scaling factor to normalize the flux, is either equal to C_P or $C_{\bar{\phi}}$ based on whether the irradiation history in the specific `[[[deplete]]]` operation was defined using the `powers` or `fluxes` parameter (see Section 4.7.3).

2.2.1 Calculation of Average Recoverable Energy per Fission (Q_{rec})

The methodology implemented in ADDER for calculation of the average recoverable energy per fission, Q_{rec}, is based on a correlation that is used by ORIGEN2 [3] to estimate an isotope-specific recoverable energy for every fissionable isotope. These isotope-specific values are then weighted by the concentration of the specific isotope within each material to calculate a material-averaged value. Finally, the material averaged-values are weighted by their fission rate to calculate a model-specific average recoverable energy per fission.

The details of this methodology, as implemented in ADDER, are as follows. First, for every fissionable isotope i in every material m, the *recoverable* energy per fission of that specific isotope is calculated as follows:

$$Q_i = 0.00129927 \cdot Z_i^2\sqrt{A_i} + 33.12 \quad \text{MeV} \tag{12}$$

where Z_i is the atomic number of isotope i and A_i is the mass number of isotope i. It is worth noting that the *recoverable* energy per fission in Equation (12) assumes contributions from the fission event (e.g., energy carried by fission products and primary gammas) and secondary reactions, such as, e.g., (n,γ) reactions inside and outside of the fuel. As such, Equation (12) represents an estimate that has been consistently used for depletion calculations, but that may be enhanced in the future to include more recent data and correlations. At a minimum, users should be aware of these underlying assumptions.

Next, the average fission recoverable energy for each material m is calculated by multiplying each isotope's Q_i by its fission rate, as follows:

$$\bar{E}_m = V_m \sum_i Q_i N_{i,m} \sum_g \sigma_{i,g,m,f} \cdot \varphi_{g,m;MCNP} \tag{13}$$

where $\sigma_{i,g,m,f}$ is the microscopic fission (energy group g) cross-section of isotope i in material m. Similarly, the total fission rate for the material is calculated as:

$$\bar{F}_m = V_m \sum_i N_{i,m} \sum_g \sigma_{i,g,m,f} \cdot \varphi_{g,m;MCNP} \tag{14}$$

Finally, the average recoverable energy per fission for the entire system is calculated by dividing the total (summed over every material) fission energy release by the total fission rate, as follows:

$$Q_{rec} = \frac{\sum_m \bar{E}_m}{\sum_m \bar{F}_m} \tag{15}$$

Alternative methodologies for calculation of Q_{rec} could be investigated to add additional capabilities into the software.

2.3 Calculation of Power Density, Fission Density, and Burnup

As part of the neutronics calculations that are performed during `[[[deplete]]]` operations, ADDER includes tallies for reaction channels, x, for isotope, i, in every depleting material, m. This includes fission cross-sections for all the fissile isotopes. Note that reaction rates are not calculated for isotopes that are identified as non-depleting or that do not meet the reactivity threshold, as selected by the user (see Section 4). Additionally, the relevant reaction channels need to be present in the initial depletion library for them to be included as tallies in the neutronics inputs. These cross-sections are used by ADDER to calculate the power density, $\dot{p}_m$, (in `W/cm`^{3}), fission density, $\dot{F}_m$, (in `fissions/cm`^{3}), and burnup, BU_m, (in `MWd`) for every material, m.

The power density in material m is calculated, at every depletion step n, as follows:

$$\dot{p}_m = \sum_i Q_i N_{i,m} \sum_g \sigma_{i,g,m,f} \phi_{g,m} \tag{16}$$

where Q_i is the recoverable energy from fission for isotope i (converted to units of `J`) calculated using Equation (12), $N_{i,m}$ is the atom density of isotope i in material m (in units of `atoms/barn-cm`), $\sigma_{i,g,m,f}$ is the energy group g microscopic fission cross-section for isotope i in material m (in units of `barn`), and $\phi_{g,m}$ is the energy group g neutron flux in material m (in units of $\mathtt{n/cm^2\text{-}sec}$) calculated via the neutronics solver and appropriately normalized as detailed in Section 2.2, Flux Normalization. It should be noted that the approach used in Equation (16) is conservative as it assumes that the entire recoverable energy from the fission events, including that from fission-induced photons and (n,γ) reactions, is deposited within the fissionable materials.

The fission density in material m is calculated at every depletion step n as follows:

$$\dot{F}_m = \sum_{n'=1}^{n} \Delta t_{n'} \left(\sum_i N_{i,m} \sum_g \sigma_{i,g,m,f} \phi_{g,m} \right)_{(n')} \tag{17}$$

where V_m is the volume of material m and $\Delta t_{n'}$ is the duration of depletion step n'. The subscript *(n')* indicates that the term in parentheses is evaluated at depletion step *n'*. The summation in the parenthetical is evaluated at every time step n' from the beginning of the calculation up to the current time step for which the fission density is evaluated, n. As such, $\dot{F}_m$, is the cumulative fission density from the beginning of the calculation until the end of the current step, for every time step n.

The burnup of material m is calculated at every depletion step n as follows:

$$BU_m = V_m \sum_{n'=1}^{n} \Delta t_{n'} \left(\sum_i Q_i N_{i,m} \sum_g \sigma_{i,g,m,f} \phi_{g,m} \right)_{(n')} \tag{18}$$

Burnup is then converted from units of `J` to units of `MWd`. Similar to the fission density, the burnup is cumulative and integrated over all the past depletion steps, n', up to the depletion step of interest, n. It is evident from Equations (16) to (18) that these quantities are all intimately related and the methodology for calculating them is analogous.

Finally, note that the power density, $\dot{p}_m$, is assumed to be constant over a depletion time step and it is calculated based on the flux and cross-sections obtained from the neutronics solver *at the start* of the depletion step (for both the predictor and predictor/corrector schemes). The fission density and burnup contributions from each time step are calculated using the same flux and cross-sections and multiplied times the time step duration, $\Delta t_{n'}$, hence the calculated values refer to the *end* of the depletion time step. Consequently, in the output file, for a given depletion step, the reported burnup and fission density correspond to the isotopic concentrations at the start of the next reported *[[[deplete]]]* time step (see Section 4.7.3). Note that, when using the predictor/corrector scheme (as discussed in Section 2.1.3), the flux is updated at the depletion step midpoint to generate the midpoint depletion matrix. This flux, however, is not used for the fission density and burnup calculation at this stage.

For consistency with the MCNP neutron transport software, a comparison of the ADDER methodology for the calculation of the above quantities – particularly the power density – to the MCNP tallies is detailed in Section 7.1.4, Equivalence of ADDER-calculated Power Density to MCNP f4 and f7 Tallies.

Note that minor discrepancies may arise between the sum of the power density over all the depleting materials and the input reactor power for step n, P_n. ADDER renormalizes the power density and burnup values to eliminate these discrepancies, however it is important that users are aware of the implications. This feature can be turned off in the ADDER input file. The source of the discrepancies as well as the re-normalization approach is discussed in Section 2.3.1, Re-normalization of Power Density. Note that the renormalization is not applied to the fission density, but only to the energy-related quantities (power density and burnup).

2.3.1 Re-normalization of Power Density

Some minor discrepancies may occur in the calculation of the power densities and can be attributed to two main causes.

The first cause is directly related to the use of a stochastic neutronics solver (such as MCNP). Multiple effects may contribute to this and an extensive quantitative evaluation would be impractical as it pertains the specific neutronics solver. One of the contributors is the statistical uncertainty of the neutronics parameters (k_{eff} and $\bar{\nu}$) and its propagation to the calculated Q_{rec} value (see Equation (15)) that is used to calculate the flux normalization scaling factor, C_P (see Equation (9)). Another effect is related to ADDER using the fission cross-sections extracted from the neutronics output to calculate the power density. These values are reported up to the sixth significant figure, which may cause round-off errors when a large number of depletion regions are defined in the model. Finally, it should be noted that some differences may arise if the approaches for tallying the cross-section results and the neutronics parameters are based on slightly different methodologies within the neutronics solver. The discrepancy between the calculated and input power due to the effects discussed above is however expected to be small, and ADDER will trigger a warning if a value greater than 1% is encountered. This value is based on empirical testing of the software but depends on the model statistics and complexity. As such, users should be aware of this behavior and inspect their results accordingly, based on whether the power normalization feature is turned on or off. If power normalization is performed (default behavior), users should review the `power_normalization` output value for the relevant `deplete` operations. Otherwise, users should manually multiply the power density values by the volumes and sum over all materials to obtain the total reactor power, and this should then be compared to the input value (in `MW`).

The second cause only occurs in cases in which the depletion library is not re-calculated for the specific depletion step. This happens when the neutron cross-section data is *not* recomputed using the neutronics solver and causes the summation performed to calculate the power density – Equation (16) – to be inconsistent with the fission source and fission rate as calculated by MCNP. The latter is then used to calculate the scaling factor, C_P, as discussed in Section 2.2, Flux Normalization. The discrepancy magnitude depends on how different the cross-section data used by ADDER to calculate the power distribution is with respect to that used in the neutronics solver. The above scenario can happen in any of these cases:

1. The `use_depletion_library_xs` flag is set to `True`. In this case, the default cross-section data is used from the HDF5 depletion data library (see Section 1.3, ADDER Data Sources). This can be done for the entire model (see Section 4.2, Metadata) or on a per material basis (see Section 4.3.1, [materials] Metadata). Based on the assumptions used for generating the depletion library, the cross-section data can be substantially different from that used in MCNP and can lead to substantial differences in the calculated total reactor power.

2. Multiple steps are defined using the `powers` parameter in the same `[[[deplete]]]` operation (see Section 4.7.3). In this case, the cross-section data is only recalculated at the first of these depletion steps. As such, for the following depletion steps in the same operation, there may be slight discrepancies due to the depletion of the materials. This effect is less impactful, however, it can be very significant for fresh materials if the first `[[[deplete]]]` operation defined in the ADDER input file after the material is inserted in the reactor involves multiple sub-steps, as the cross-section data for the fission products is set to the value in the depletion library default because the specific isotopes are not present in the first MCNP calculation. As such, it is recommended to only define a single irradiation step in the first `[[[deplete]]]` operation of any ADDER calculation after a fresh depleting material is inserted in the reactor.

Although these discrepancies are primarily related to user choices (e.g., using pre-built cross-section data, reducing the computation time by not re-computing the cross-section at every irradiation step), the user should be aware of them and understand their implications and the mitigation strategies discussed above.

ADDER offers the possibility to re-normalize the power density to ensure that the calculated reactor power is exactly equal to the value provided in the `powers` parameter of the specific `[[[deplete]]]` operation. In this case, the weight of each material power density to the total power remains unchanged, however, the volume-integrated power density (i.e., the calculated power) is forced to be equal to the input power. Formally, the renormalized power density is hence calculated as follows:

$$\dot{p}_m^* = \dot{p}_m \frac{P_n}{\sum_m V_m \dot{p}_m} = \dot{p}_m A_P \tag{19}$$

where $\dot{p}_m$is the power density in material m as calculated by ADDER using Equation (16), $\dot{p}_m^*$ is the renormalized power density, V_m is the volume of material m, and $P_n$is the user-provided reactor power for irradiation step n. The scaling factor for the normalization, A_P, is retained in the ADDER HDF5 output (see Section 5, Output Data Format).

It should be noted that the renormalization scaling factor is applied to the calculated burnup as well but *not* to the fission density values, to maintain the latter consistent with the depletion calculation.

The renormalization option can be set for the entire ADDER calculation or for specific `[[[deplete]]]` operations. By the default, this option is enabled for all the `[[[deplete]]]` operations (i.e., the `renormalize_power_density` flag is set to `True`). Instructions on how to turn the re-normalization of the power density off are provided in Section 4, Input File Format.

2.4 Considerations on Statistical Uncertainties of Monte Carlo Neutronics Solvers

It is important to note that if a Monte Carlo software (e.g., MCNP) is used as the neutronics solver for ADDER, the results extracted from the output are associated with a certain level of statistical confidence. Specifically, for ADDER, these solvers are used to calculate the neutronics quantities of interest needed to perform the depletion calculation, i.e., neutron flux, cross-sections, k_{eff}, and $\bar{\nu}$, and their standard deviations.

The quality of the ADDER results is inherently dependent on the level of statistical precision of the results obtained from the neutronics solver. However, it is beyond the scope of ADDER to perform statistical uncertainty checks on all these results and imposing a statistical uncertainty requirement on them. This is especially true as the uncertainties of the calculated local quantities (e.g., cross-sections) depend on the size of the depletion regions identified in the neutronics input file.

Given the considerations above, users should make sure to inspect the Monte Carlo software outputs generated during the ADDER calculation to make sure that the statistical uncertainties of the cross-section tallies are satisfactory. A good rule of thumb is to have statistical uncertainties around 1% or lower for the absorption cross-section of major actinides. It should also be noted that this is highly model-dependent and any quantitative threshold should be based on the user's knowledge of the model under analysis, including the number of different depletion regions and their size. To decrease the statistical uncertainty on these tallies while keeping the same number/size of depletion regions, users should increase the Monte Carlo criticality calculation parameters (e.g., `kcode` card parameters for MCNP). Alternatively, the size of the depletion regions can be increased if reasonable accuracy can be achieved with coarser regions.

3 Fuel Management Methodology

As stated, ADDER is a fuel management and depletion interface tool intended to simplify as much as possible these operations on complex, multi-cycle, reactor analyses. The overall execution process is to read the ADDER input, parse the neutronics solver model, setup operations, and execute those operations while storing results along the way. Before defining the commands utilized in the input file format, some background is necessary. In the next sections, several concepts, central to understanding ADDER's fuel management methodology, definitions, and input, are presented.

3.1 Materials and Universes

The basic component of fuel management is a "material"; such materials are as one would expect: collections of isotopes with specified atom fractions and density. These are the depleting and shuffling components of the model. In most neutronics solvers, these materials are referenced by integer IDs. Since a reactor model can contain millions of materials, ADDER provides capabilities that simplify the handling of all these material IDs. These are the following, generally in order of increasing aggregation:

- User-specified material names can be used instead of the integer IDs.
  - Naming allows the user to define, and later understand, the ADDER operations more simply with the use of sensical names.
  - If the user does not specify a name, then the name is assumed to simply be the integer ID.
  - ADDER will automatically duplicate materials assigned to multiple regions (cells, in MCNP) in the model if certain conditions are met. At the highest level, depleting materials and explicitly shuffled materials are duplicated. This allows the user to not expend effort modifying the neutronics model to create the necessary unique regions.
    - ∗ When duplicated, the cloned materials are re-named according to `oldname_reg_#` where # is the region (cell in MCNP) numerical ID. These name changes can be seen by

inspection of the ADDER log file (`adder.log`).

    * When a material duplicated by ADDER is later shuffled, the user should recognize that the duplicated name (discussed in the previous bullet) must be used, and that shuffling a duplicated material only shuffles that instance, not all of the original (pre-duplicated) instances of the material.

- User-specified material aliases which can be used for sets of material names.

  - These material aliases are simply one name to be used in place of a set of material names. For example, if one does not want to write "material_1", "material_2", …, "material_1000" every time, they can instead define an alias, named "materials" that contains "material_1" through "material_1000". ADDER will then replace the usage of this alias with the intended set of materials during input processing. The results of this are incorporated in the input echo reproduced in the ADDER log file.

  - When an alias is used for operations, the shuffle will be performed on each of the items in the set. For example, if the "materials" alias is shuffled to the locations of "materials_alias_2", then the first material in the "materials" set will move to the location of the first material in the "materials_alias_2" set, the second material in the "materials" set will move to the location of the second material in the "materials_alias_2", and so on. For this reason, when using aliases for shuffling, all aliases within that operation must have the same length.

  - Naturally, ADDER needs to be able to decipher between an alias name and a material name. Therefore, ADDER enforces a rule that aliases cannot be named the same as the material names. Further, the alias names must not be integers to minimize the chances for confusion during input file creation.

  - These aliases are purely optional as their benefits depend on specific attributes of the reactor and the size of the model.

- Repeated geometries are sets of geometric entities, and thus are sets of materials. Such repeated geometries are commonly referred to in Monte Carlo codes as universes, and so they are also referred to as universes in ADDER. Note that not all neutronics solvers are guaranteed to support these universe constructs. Currently, the only solver supported by ADDER is MCNP, which does support universes.

  - There is no specific way to define a universe as a collection of geometry and materials in ADDER; the equivalent universe construct in the neutronics solver's model is simply read and understood by ADDER.

  - However, ADDER does recognize the presence of these universes and can take them into account to simplify the user interface for fuel management.

  - As an example of the utility of ADDER understanding universes, consider a single reactor assembly composed of thousands of materials as it typically would be for a depleting model. Moving that assembly would require a complicated and risk-prone input file specifying the specific material moves to be performed for the thousands of materials. The material alias discussed above would reduce this risk by requiring the thousands of materials to only be delineated once (in the definition of the alias) instead of every time the assembly is moved. However, if that reactor assembly is mapped (on a one-to-one basis) to a universe, then ADDER only needs to move that one universe instead of the thousands of materials and the user does not need to specify the alias with all of its components.

  - Therefore, ADDER recognizes these universes and their manipulations and definitions are discussed below.

- User-specified universe names can also be supplied instead of the neutronics solver's integer IDs.
  - These are specified in a manner similar to the naming of materials discussed above. Like the materials, these names are required to be used for fuel management operations, however, the names default to the integral universe id if the user does not otherwise specify them.
- Finally, user-specified universe aliases are also possible.
  - These are analogous to the material aliases discussed above, allowing sets of universes to be operated on at once instead of through explicit operations for each.

3.2 Components (in-core, supply, storage)

Next, the user should be aware of the concepts of a component (i.e., a specific material or universe) being considered "in-core", "supply", or "storage".

- An "in-core" component is one that exists within the model and is modeled such that particles can eventually reach them (even if improbable).
- A "supply" component is one considered as a template for components to be brought into the core in the future.
  - If a component is included in the initial neutronics model (i.e., the model as supplied to ADDER), but not assigned to any spatial region, then it is not possible for any particles in the neutronics solver to reach that component. Such components are automatically labeled as "supply".
  - If a "supply" component is shuffled into the core, then ADDER will make a clone of that component, and introduce the clone into the core. This clone will then be treated as "in-core" and can be further shuffled within the core or to "storage".
  - The clone will have a new name. ADDER will append a [#] to the original name, where # is the number of clones made thus far. For example, the first clone of an initial storage material named `fresh_fuel` will need to subsequently be referred to as `fresh_fuel[1]` .

A "storage" component is one that is considered ex-core but is not to be treated as a template. In other words, when that "storage" component is shuffled into the core, no copy is made, the component is simply moved.

As stated, both materials and universes have this in-core, supply, and storage distinction, and both can be set with the `[[storage]]` and `[[supply]]` blocks as discussed below. This allows the user to tell ADDER that a component which defaulted to being in "supply" (since it is not in the reactor per the initial neutronics input file) should actually be considered in "storage". This also allows the user, for example, to make a copy of any "in-core" or "storage" component to be used as a template by using the `[[supply]][[[copy]]]` delineator.

Note: The universe storage/supply definition occurs after the material definitions and thus will overwrite any user re-definitions of materials as being in supply or storage.

Note: ADDER does not automatically duplicate universes and their constituents that are assigned to multiple regions. However, it will duplicate these universes and their constituent regions and materials upon cloning or copying of the universe.

3.3 Control Groups

ADDER also provides the user with the ability to pre-define "control groups":

- A control group is a named set of surfaces, cells, or universes that are to be moved together as one entity to represent the motion of control devices, for example. The control group definition must include all surfaces/cells/universes that undergo coordinate transformations together to represent the desired motion.

  Control group designators are defined in the `control_group` section and are specifically for usage with the `transform`, `geometry_sweep`, and `geometry_search` operations.

- A control group will have its displacement, relative to the initial model conditions, tracked throughout the entire analysis process. This position will be included in ADDER's results HDF5 file at each case/operation/step point.

- A control group is not necessary to perform the coordinate transformation with `transform` operation, however, a control group should be used if the user wishes for the displacement to be tracked.

Note: If a control group is defined, but the constituent components are moved with a `transform` operation that does not use the control group, then this tracked relative displacement can become out of sync.

4 Input File Format

This section of the User Guide discusses the input interactions the user should expect with ADDER, providing details on the ADDER input file sections, subsections, and their parameters. If a particular section or variable is optional, it will be denoted as such and the default value provided. It is recommended that users read and familiarize themselves with Section 3 to fully understand the philosophy and methodology that serves as the basis for the fuel management operations discussed here. Commented example input files can be found in the `examples/` sub-folder of the ADDER distribution/repository.

Note that ADDER can be executed with the `-n` or `--no-deplete` command line flag. With this command, ADDER performs fuel management operations, but does not execute the neutronics or depletion solvers, thereby producing the typical fuel management output and logs which can be used to interrogate the names ADDER has applied. A different naming convention is used for depleting and non-depleting materials.

4.1 ADDER Execution and General Input File Information

Before discussing the specifics of the format, some preliminary information should be discussed. Further information can be found via the ConfigObj documentation as this package is used to parse the input file.

1. The ADDER input file is an input file with a format similar to the classic INI configuration file format wherein sections are defined via the section names that are each enclosed in square brackets ([and]) and the values of parameters set by an equal sign. An important distinction between ADDER input files and INI files is that the ADDER format utilizes subsections and the order of sections and parameters within sections are preserved.

2. Sections are denoted by stating the section name within square brackets. For example, a section named "operations" would be declared as `[operations]`.

3. Sections can be nested within other sections, thereby creating subsections. An arbitrary number of nested subsection levels can be defined. Since denoting a section with a single-pair of square brackets would imply a new section is started, subsections are instead denoted by adding an additional pair of square brackets around the subsection name. For example, a `deplete` subsection that is within the `operations` section would be defined as follows:

```
[operations]
  [[deplete]]
    ...
```

   A subsubsection could then be defined within `[[deplete]]` by using triple square-brackets.

4. Comments can be denoted with the pound character (#) anywhere within the document. When a pound is encountered, all text to the right of that # is ignored.

5. Variables can be set with values using a `key = value` approach. The value will be type-cast to the correct type by both ConfigObj and ADDER-specific code. Further, values that are strings can optionally be enclosed in single or double quotes. Quotes around string values are required if the user wishes a string to include a blank space, a comment character (#) or a comma (used for lists). Multi-line strings are currently not supported.

6. Some input parameters require a list of values. Lists can be defined as values by utilizing commas to separate the individual values. If a list has only one entry, no trailing comma is necessary.

7. The names of parameters and sections are case-sensitive, i.e., **I**tem is not equivalent to **i**tem. The values for parameters may or may not be case-sensitive, depending on the context. For example, the value of a variable describing a filename should be case-sensitive when ADDER will be executed on an operating system that utilizes a case-sensitive file system.

8. ADDER uses the GND-like naming convention for nuclides/isotopes and metastable states:

   Isotopes `SymA` where "A" is the mass number (e.g., "Fe56")

   Metastable states `SymA_mN` (e.g., Am242_m1 for the first excited state of Americium-242).

9. Boolean (i.e., True or False) input values in ADDER can either be provided as typical Boolean values of `True` or `False`, but ADDER also interprets the following case-insensitive values as one would expect: A value of `True` can also be specified as: `true`, `yes`, `on`, and `1`. A value of `False` can also be specified as: `false, no, off`, and `0`. Any other strings will raise an error.

10. The only limit on input file string lengths is the memory available on the system.

11. Parameters which are not expected input are ignored.

4.2 Metadata

The metadata is information regarding the overall ADDER execution. Metadata has no section header and should be provided before any sections.

The description of all the parameters within this section are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

case_name [required, string] The name of the particular case for user convenience. This information will be propagated to the ADDER output files as relevant.

neutronics_solver [required, string] The name of the neutronics solver to use. Since MCNP (MCNP versions 5 and 6) is currently the only supported solver, this variable must be set to a string of `mcnp`. The specific version of MCNP is determined by the executable passed in `neutronics_exec`.

neutronics_exec [required, string] The command to run when executing the `neutronics_solver` program.

neutronics_input_file [required, string] The absolute or relative path of the base input file to use as the starting point for reactor depletion and manipulation.

neutronics_library_file [optional, string] The absolute or relative path of the neutronics library file. The default value of the parameter depends on the neutronics solver used. For the case of MCNP, this library file is used to determine which isotopes should be included in the neutronics calculation (and thus, this file provides the list of isotopes where nuclear data is available). Even with MCNP, this is an optional parameter, as if no value is provided, then the `$DATAPATH` environmental variable will be used instead with the MCNP default of `$DATAPATH\xsdir`.

neutronics_reactivity_threshold [optional, float] The reactivity threshold (in absolute units) when determining if an isotope should be incorporated in the neutronics model. Use this to remove isotopes with negligible reactivity effect (where negligible is defined as being below this threshold) from the neutronics calculations. This is an optional parameter that defaults to a threshold of 0.0, indicating that all isotopes will be included in the neutronics calculation.

The following algorithm is used to identify the reactivity worth of each of the isotopes (denoted with subscript *i*) in material *m* for the case of a non-zero flux and fissionable isotopes present. This method is based on identifying the reactivity worth of each isotope in an infinite medium composed of this material with the flux spectrum determined from the latest neutronics solver calculation:

$$\nu\Sigma_{f,m}\phi_m = \sum_{i\ in\ m}\sum_{g}\nu\Sigma_{f,i\in m}\Phi_{\mathrm{g,m}}$$

$$\Sigma_{a,m}\phi_m = \sum_{i\ in\ m}\sum_{g}\Sigma_{a,i\in m}\Phi_{\mathrm{g,m}}$$

$$\mathrm{k}_{\infty,\mathrm{unperturbed}} = \frac{\nu\Sigma_{f,m}\phi_m}{\Sigma_{a,m}\phi_m}$$

$$k_{\infty,perturbed\ i} = \left(\nu\Sigma_{f,m}\phi_m - \sum_{g}\nu\Sigma_{f,i}\phi_{g,m}\right)\Bigg/\left(\Sigma_{a,m}\phi_m - \sum_{g}\Sigma_{a,i}\phi_{g,m}\right)$$

$$\rho_{perturbed\ i} = \left|\frac{k_{\infty,unperturbed} - k_{\infty,perturbed\ i}}{k_{\infty,perturbed\ i}}\right|$$

When a material is evaluated that does not contain fissionable isotopes, the infinite medium reactivity computation cannot be performed. Instead, the fractional absorption rate is used:

$$\rho_{perturbed\ i} = \left(\sum\nolimits_{g}\Sigma_{a,i}\phi_{g,m}\right)\Big/\Sigma_{a,m}\phi_m$$

In both of the above, the cross sections are obtained from the depletion library. Further, the absorption reaction rate is based on an absorption cross section that is the sum of all neutron disappearance/removal reactions available in the depletion library.

With the reactivity worth of each isotope generated, the next step is to sort the isotopes by reactivity worth and identify the set of isotopes whose total reactivity worth is less than the `neutronics_reactivity_threshold`. This set of isotopes are then filtered out from being written to the neutronics input model.

If a material's flux is zero (e.g., after a zero-power depletion) and the depletion library contains only one group, then the filtering will be performed as stated above. However, if a material's flux is zero and the depletion library contains cross sections with more than one-group, then ADDER cannot perform neutron energy-weighting of the cross sections. In this case, ADDER will simply filter out all isotopes with concentrations of < 1E-10 atom/b-cm so long as a non-zero value is provided to this parameter.

apply_reactivity_threshold_to_initial_inventory [optional, boolean] This flag specifies whether the neutronics_reactivity_threshold is applied to the *initial* isotopic/elemental inventory or not. This option has no effect on the material's isotopes/elements produced at later times via depletion. This value defaults to `False`, indicating the reactivity threshold is not applied to the initial inventory.

In addition to globally setting this option, the user can control this on a per-material basis with the `apply_reactivity_threshold_to_initial_inventory` option in the `[materials][[metadata]]` subsection. The options in the `[materials][[metadata]]` subsection will override this value for the materials they are applied to.

depletion_solver [required, string] Similar to `neutronics_solver`, this is the choice for the depletion solver software. The currently supported options are `origen2.2, cram16, cram48, msr16`, and `msr48`. The first uses ORIGEN2.2 to solve the depletion system of equations. The cram## options use ADDER's internal CRAM solver with the 16th and 48th order approximations. Finally, the `msr##` options use the internal MSR depletion solver, also while applying the 16th and 48th order CRAM approximation.

depletion_exec [optional, string] The same as `neutronics_exec` except for the depletion solver. Note this is required if `depletion_solver` is origen2.2 but not needed for the internal CRAM solvers.

depletion_library_file [required, string] The absolute or relative path e of the default/initial depletion data library HDF5 file.

depletion_library_name [required, string] Since the depletion data library HDF5 file can contain multiple independent data sets, the `depletion_library_name` provides the name of the specific data set to use as the initial depletion data library for this ADDER execution.

mpi_command [optional, string] If it is desired to execute the neutronics solver via the Message Passing Interface (MPI), and the neutronics solver supports it, this value provides the MPI command (e.g., `mpirun`) which should be used. The user should ensure that this command can be called from a typical shell environment, and if necessary, include the path to the MPI command (e.g., `/usr/bin/mpirun` if necessary). If the value is not provided, then MPI will not be used.

num_mpi_processes [optional, integer] This provides the number of MPI processes to execute the neutronics solver with. This value is interpreted as an integer. This defaults to a value of 1.

num_threads [optional, integer] The number of shared memory threads to use for the neutronics and depletion solvers. This value is interpreted as an integer. This defaults to a value of 1. If provided, this is applied to both the neutronics and depletion solvers.

num_neutronics_threads [optional, integer] The number of shared memory threads to use for just the neutronics solver. This value is interpreted as an integer. This defaults to a value of 1. If num_threads is provided, then this is ignored.

num_depletion_threads [optional, integer] The number of shared memory threads to use for just the depletion solver. This value is interpreted as an integer. This defaults to a value of 1. If num_threads is provided, then this is ignored.

depletion_chunksize [optional, integer] This parameter is used to tune the parallel depletion performance. Specifically, it sets the number of materials passed at a time to a depletion thread. ADDER will automatically calculate a chunksize that is optimized for calculation speed if this parameter is set to zero. This parameter defaults to a value of 0, which is the recommended value.

output_hdf5 [optional, string] This sets the absolute or relative path of the desired HDF5 results file. This defaults to a value of `results.h5`.

depletion_method [optional, string] This sets which type of depletion method to use for depletion problems. The possible options are strings of either "predictor" (for the predictor method) or "cecm" (for predictor/corrector or CE/CM method) as defined in [20]. This value defaults to a value of the more accurate but costly "cecm". Finally, this is a global default which can be overridden for individual `[[[deplete]]]` sections by the value also named `depletion_method`.

use_depletion_library_xs [optional, Boolean] This sets whether the cross sections provided in the default depletion library will be used, or if the neutronics solver will be relied upon to provide as much of these as possible. This defaults to a value of `True`, indicating that the cross sections specified in the library will be used directly.

If this parameter is assigned a value of `False`, then the neutronics solver will be used to compute the cross sections which will be used for depletion. In this case, the depletion library data will be used to determine which isotopes, and reaction channels should be computed, as well as to identify the number of energy groups to use when computing these cross sections. This cross section update will be performed for the first neutronics solve in each of the `[[[deplete]]]` sections. In this way, the user can use these sections to control the frequency of cross section updates.

In addition to globally setting this option, when this flag is set to `False`, the user can specify which materials will use the default depletion library, as-is. This is enabled by the `use_default_depletion_library` option in the `[materials][[metadata]]` subsection. This feature allows the user to expend computational power generating (e.g., tallying) cross sections only in the materials for which the enhanced accuracy is necessary.

depletion_substeps [optional, integer] This value provides the number of intervals to sub-divide an operating history step to potentially increase the accuracy of the final result. This value defaults to a value of 1. This is a global default which can be over-ridden for individual `[[[deplete]]]` sections by the value also named `depletion_substeps`.

renormalize_power_density [optional, Boolean] This flag sets whether the material-wise power density and burnup values are to be renormalized following any `[[[deplete]]]` operation to ensure consistency with the `powers` indicated for the operation in the relevant subsection, if applicable. If this flag is set to `True` (default), the material-wise power density and burnup as calculated by ADDER are renormalized to be consistent with the power input. On the other hand, the total fission density is not renormalized. If this flag is set to `False` the renormalization is not performed. In either case, ADDER will provide warnings when the calculated power densities sum to a total power that is more than 1% different with respect to the `powers` values provided for the `[[[deplete]]]` operation. The renormalization constant is stored in the ADDER output.

`[[[deplete]]]` operations that are defined by setting the powers attribute (see Section 4.7.3) and for which more than one `powers` and `durations` values are provided may incur inconsistencies between the reactor power provided as input and the calculated power density values. In fact, when multiple `powers` values are provided, ADDER does not recompute the fission cross-section library and, instead, utilizes the ones from the previous depletion step. If these are not available (i.e., for isotopes not included in the calculation at the previous steps), the default cross-section library entry is used instead. These cross-sections may result in a total fission source that is inconsistent with the one calculated by the neutronics solver to normalize the flux. As such, the power density calculated by ADDER as a sum of the fissions of each isotope multiplied by the specific recoverable energy may be different from the power provided as input.

It is recommended that the users either perform the very first `[[[deplete]]]` step defining a single `powers` value, effectively re-calculating the cross-sections for all the fission and decay products at the second depletion step; or that trace amounts of all of the relevant isotopes are included in the base MCNP input in the isotopic composition of depleting materials, which results in calculating microscopic cross-sections for all the isotopes at the first step.

In addition to globally setting this option, this behavior can be set independently for each `[[[deplete]]]` operation by individually setting the parameter of the same name within it.

4.3 Materials

ADDER obtains as much information as possible about the materials used in the evaluation from the neutronics solver's input file specified by the *neutronics_input_file* parameter. This includes the isotopic or elemental constituents, densities, etc. The `[materials]` section is used to provide additional information that is not typically contained within a generic neutronics input file such as the name of the material for use within ADDER, the optional material volume, whether the material is to be treated as depleting, and providing the user the option to specify which specific isotopes within the material are not to be depleted.

Within the `[materials]` section are subsections for `[[metadata]]`, `[[aliases]]`, `[[storage]]`, and `[[supply]]`. These are described separately in the discussion which follows.

The usage of this `[material]` section of the input file is optional.

4.3.1 [materials] Metadata

Since a model can contain a large number of materials and many of them can have similar attributes, ADDER allows the user to provide this materials identification on individual materials via the [[metadata]] subsection and its own subsections: [[[item]]], [[[list]]], and [[[range]]]. The format and purpose of each subsection are described below.

Any material from the neutronics input file that is not specified via the means discussed below will be assigned a name that is the same as its identifying integer in the neutronics input file (neutronics_id), the material will be depleted, and no isotopes will be set to non-depleting. As such, the [materials][[metadata]] subsection is optional.

The [[[item]]], [[[list]]], and [[[range]]] subsections provide facilities for setting the volumes of materials. The user needs to be aware that if the volume is supplied here, then the calc_volume operation will not overwrite this value. Therefore, this should only be used for volumes which will be constant throughout the simulated operating history. For example, this parameter should not be used for moving components such as control devices if their modeled volume varies.

Some neutronics solvers only provide a density to a material if it is assigned to a geometric region in the core (for example, in MCNP, density is specified on cell cards, not on material definitions). In this case, a material that is not assigned to a cell will not have a density assigned based on the neutronics input file alone. Therefore, ADDER must have an alternative means to provide the density. These [[[item]]], [[[list]]], and [[[range]]] subsections provide that capability.

Finally, the [[[item]]], [[[list]]], and [[[range]]] subsections are applied in the order that the user defines them. This allows the user to define attributes for a wide range and then add exceptions to that range as they are encountered. For example, if materials 1 through 100 are intended to have the same attributes, except for material 42, then a [[[range]]] can be used to define the attributes of 1 through 100 followed by an [[[item]]] subsection specifically for material 42. Note that this capability is limited in that subsections which act on an already set material must have all properties set to the expected defaults. In other words, the attributes set in the [[[range]]] subsection for material 42 in the example above will all be over-written when the [[[item]]] subsection is applied, including with the default values for an [[[item]]] subsection.

4.3.1.1 [[[item]]]

Individual materials can be described by use of an [[[item]]] subsection. The actual subsection header for an item does not need to be exactly [[[item]]], but the text within the brackets must start with item. This allows for multiple item subsections to be specified so long as they have unique text after the specific phrase item. For example, if the user wishes to individually specify three items, their subsection names can be: [[[item]]], [[[item1]]], and [[[item_last]]], but they cannot be: [[[item]]], [[[item]]], and [[[item]]].

The description of all the parameters within this item subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

neutronics_id [required, integer] Which material in the neutronics model specified by the *neutronics_input_file* parameter that the `[[[item]]]` subsection is referring to.

name [optional, string] The name to apply within ADDER to the material with the `neutronics_id` provided above. If no value is provided, then the name will be a string representation of the `neutronics_id`. Note that this name will *replace* the neutronics ID and needs to be used in the rest of the input file to identify the material.

volume [optional, float] The volume of this material in the model in units of cubic centimeters. Note that this volume must be the volume of each instance of this material, as ADDER will duplicate depleting and shuffled materials.

depleting [optional, Boolean] whether this material is to be depleted during any depletion operations specified later. Since additional depleting materials generally increase the memory and computation cost of a calculation, this is often used so that only materials whose depleted inventories are important to downstream parameters of interest are depleted, reducing the overall computational costs. All materials without a value of `depleting` specified will default to a value of `True,` indicating the material will be depleted.

non_depleting_isotopes [optional, list of strings] Some materials should be depleted; however, it would be a wasteful use of computational resources to track the reaction rates and decays of every isotope within that material. The `non_depleting_isotopes` parameter allows the user to specify the list of isotope names within this material that should not be treated as variant during the depletion simulation. The isotope names should be named according to the GND naming convention described earlier.

density [optional, float] This `density` parameter provides this value in units of `atoms/b-cm.` This parameter is technically optional; however, the user should be aware that if a material in the neutronics model has no density since it is not assigned to a geometry region, and it has no density assigned here, then ADDER will exit with an error if the status of the material or any of its duplicates is changed to `in-core` or `storage`. For materials redefined as storage in the ADDER input file, this means that ADDER will exit before performing any operation; whereas for supply materials, ADDER will exit once an instance of it is moved into the core via a `[[[shuffle]]]` operation. When the `density` parameter is set here for a material that is duplicated during the ADDER calculation, the duplicated materials will inherit this value, irrespective of the value assigned to the geometric region (e.g., cell) in the neutronics input file.

use_default_depletion_library [optional, Boolean] This flag specifies whether this material is to use the default depletion library or to instead have the neutronics solver generate updated cross sections. Note that this option only has an effect if the `use_depletion_library_xs` option in the `[metadata]` section is `False`. This value defaults to `False`.

apply_reactivity_threshold_to_initial_inventory [optional, Boolean] This flag specifies whether the `neutronics_reactivity_threshold` is applied to this material's *initial* isotopic/elemental inventory or not. This option has no effect on the material's isotopes/elements produced at later times via depletion. This value defaults to `False`, indicating the reactivity threshold is not applied to the initial inventory. This option overrides the global version of this parameter.

4.3.1.2 [[[list]]]

Sets of materials can be described by use of either a `[[[list]]]` subsection or `[[[range]]]` subsection. This section describes the `[[[list]]]` subsection.

Like `[[[item]]]`, the actual subsection header for a list does not need to be `[[[list]]]`, but the text within the brackets must start with list.

Unlike the `[[[item]]]` subsection, here, a list of specific `neutronics_ids` from the neutronics input file are acted on with common parameters.

The description of all the parameters within this subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

neutronics_ids [required, list of integers] A comma or space-separated list of the specific `neutronics_ids` which specify the list of materials in the neutronics model specified by the *neutronics_input_file* parameter the `[[[list]]]` subsection refers to.

names [required, list of strings] The names to apply within ADDER to the material with the `neutronics_ids` provided above. If this parameter is not provided, then the names for all of the `neutronics_ids` will be a string representation of the `neutronics_id`. If only one name is provided, then that name will be applied to each entry in the list with a _# appended to the end of that name (where # is the material's `neutronics_id`). Otherwise, one name should be provided for each `neutronics_id` in the `neutronics_ids` list.

volume [optional, float] This is the same as the volume parameter described earlier for the `[[[item]]]` subsection. Only one value is provided; this value is applied to all `neutronics_ids` in the list. Note that this volume must be the volume, in cubic centimeters, of each instance of this material, as ADDER will duplicate depleting and shuffled materials.

depleting [optional, Boolean] This is the same as the depleting parameter described earlier for the `[[[item]]]` subsection. Only one value is provided; this value is applied to all `neutronics_ids` in the list.

non_depleting_isotopes [optional, list of strings] This is the same as the *non_depleting_isotopes* parameter described earlier for the `[[[item]]]` subsection. Only one list is provided; this value is applied to all `neutronics_ids`.

densities [optional, list of floats] This `densities` parameter provides this value in units of `atoms/b-cm`. This parameter is technically optional; however, the user should be aware that if a material in the neutronics model has no density since it is not assigned to a geometry region, and it has no density assigned here, then ADDER will exit with an error if the status of the material or any of its duplicates is changed to `in-core` or `storage`. For materials redefined as storage in the ADDER input file, this means that ADDER will exit before performing any operation; whereas for supply materials, ADDER will exit once an instance of it is moved into the core via a `[[[shuffle]]]` operation. When the `densities` parameter is set here for a material that is duplicated during the ADDER calculation, the duplicated materials will inherit this value, irrespective of the value assigned to the geometric region (e.g., cell) in the neutronics input file.

use_default_depletion_library [optional, Boolean] This flag specifies whether these materials are to use the default depletion library or to instead have the neutronics solver generate updated cross sections. Note that this option only has an effect if the `use_depletion_library_xs` option in the `[metadata]` section is `False`. This value defaults to `False`.

apply_reactivity_threshold_to_initial_inventory [optional, Boolean] This flag specifies whether the `neutronics_reactivity_threshold` is applied to these materials' *initial* isotopic/elemental inventory or not. This option has no effect on the isotopes/elements produced at later times via depletion. This value defaults to `False`, indicating the reactivity threshold is not applied to the initial inventory. This option overrides the global version of this parameter.

4.3.1.3 [[[range]]]

Sets of materials can be described by use of either a `[[[list]]]` subsection or `[[[range]]]` subsection. This section describes the `[[[range]]]` subsection.

Like `[[[item]]]`, the actual subsection header for a list does not need to be `[[[range]]]`, but the text within the brackets must start with `range`.

Unlike the `[[[item]]]` subsection, here, a user-specified range of `neutronics_ids` from the neutronics input file are acted on with common parameters. This range is inclusive, meaning the start and end points are in the range.

The description of all the parameters within this subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

neutronics_id_start [required, integer] A single `neutronics_id` integer which denotes the starting integer of the range that this subsection applies to. The start integer is inclusive. This value is required.

neutronics_id_end [required, integer] A single `neutronics_id` integer which denotes the ending integer of the range that this subsection applies to. The ending integer is inclusive. This value is required.

exclude_neutronics_ids [optional, list of integers] This parameter is a list of `neutronics_id` integers within the range specified by `neutronics_id_start` and `neutronics_id_end` that are not to be included in this subsection's definition. If no values are provided, then no `neutronics_ids` are excluded.

names [optional, list of strings] The names to apply within ADDER to the material with the `neutronics_ids` provided above. If no value is provided, then the names for all of the `neutronics_ids` will be a string representation of the `neutronics_id`. If only one name is provided, then that name will be applied to each entry in the range with a _# appended to the end of that name (where # is the material's `neutronics_id`). Otherwise, one name should be provided for each `neutronics_id` in the range (except the excluded ids).

volume [optional, float] This is the same as the volume parameter described earlier for the `[[[item]]]` subsection. Only one value is provided; this value is applied to all `neutronics_ids` in the range. Note that this volume must be the volume, in cubic centimeters, of each instance of this material, as ADDER will duplicate depleting and shuffled materials.

depleting [optional, Boolean] This is the same as the *depleting* parameter described earlier for the `[[[item]]]` subsection. Only one value is provided; this value is applied to all `neutronics_ids` in the range.

non_depleting_isotopes [optional, list of strings] This is the same as the *non_depleting_isotopes* parameter described earlier for the `[[[item]]]` subsection. Only one list is provided; this value is applied to all `neutronics_ids` in the range.

density [optional, float] This `density` parameter provides this value in units of `atoms/b-cm`. The same density will be supplied to every material in the range. This parameter is technically optional; however, the user should be aware that if a material in the neutronics model has no density since it is not assigned to a geometry region, and it has no density assigned here, then ADDER will exit with an error if the status of the material or any of its duplicates is changed to `in-core` or `storage`. For materials redefined as storage in the ADDER input file, this means that ADDER will exit before performing any operation; whereas for supply materials, ADDER will exit once an instance of it is moved into the core via a `[[[shuffle]]]` operation. When the `density` parameter is set here for a material that is duplicated during the ADDER calculation, the duplicated materials will inherit this value, irrespective of the value assigned to the geometric region (e.g., cell) in the neutronics input file.

use_default_depletion_library [optional, Boolean] This flag specifies whether these materials are to use the default depletion library or to instead have the neutronics solver generate updated cross sections. Note that this option only has an effect if the `use_depletion_library_xs` option in the `[metadata]` section is `False`. This value defaults to `False`.

apply_reactivity_threshold_to_initial_inventory [optional, Boolean] This flag specifies whether the `neutronics_reactivity_threshold` is applied to these materials' *initial* isotopic/elemental inventory or not. This option has no effect on the isotopes/elements produced at later times via depletion. This value defaults to `False`, indicating the reactivity threshold is not applied to the initial inventory. This option overrides the global version of this parameter.

4.3.2 Aliases

Since there can be many materials within a model, it may be useful to refer to sets of them with just one name. This name is referred to as an alias. For example, all materials within a reactor's fuel assembly can be assigned to an alias instead of repeating the set of materials set every time.

More specifically, if an alias is defined, it will be replaced by the ADDER input processor by the set of materials it refers to for all operations defined in the `[operations]` section, thereby significantly reducing the input the user must specify.

The `[[aliases]]` subsection is composed of any number of arbitrarily named subsubsections, each of which is described via the following parameters:

name [optional, string] The name of the alias; this is the name that should be used instead of the individual materials in the set. If this parameter is not provided, then the name of the subsubsection will be the name. The name must *not* be an integer number as that could conflict with neutronics ids.

set [required, list of strings] The list of material names which compose this set. Note that if any material names include a comma, then that name must be enclosed in quotations to tell ADDER that the comma does not denote the next entry in the list, but rather is part of a name. This parameter must have at least one entry.

4.3.3 Storage and Supply

If a material is included in the neutronics model but not assigned to a region, then ADDER will treat it as material that is "supply" (a material that copies of it can be reintroduced later as if it was from a factory or other supply chain). However, the user may wish to reassign that material from supply to being an "in-storage" material (a material that exists outside of the core). Finally, the user may wish to make a copy of a material that is in the core.

All of these options are handled by the `[[storage]]` and `[[supply]]` subsections. Both subsections are optional and only included if the user wishes to perform one of the actions detailed above. Both are discussed together in this manual since the only difference in the input format is if the `[[storage]]` or `[[supply]]` subsection header is used.

If present, the `[[storage]]` and `[[supply]]` subsections contain one of the following subsubsections: `[[[copy]]]` (copy a material) or `[[[redefine]]]` (redefine a material not in the reactor from the default of supply to storage). Note that the `[[[redefine]]]` subsection redefines a material to be assigned the category ("storage" or "supply") of the parent section under which `[[[redefine]]]` is included. In other words, when the `[[[redefine]]]` subsection is under the `[[storage]]` section, the listed universe will be redefined from "supply" to "storage".

Within the `[[[copy]]]` and/or `[[[redefine]]]` subsubsections are subsubsubsections named `[[[[item]]]]`, `[[[[list]]]]`, and/or `[[[[range]]]]`. These contain the same data as defined earlier at *item*, *list*, and *range* respectively, except for the following modifications:

depleting [optional, Boolean] This is not relevant in the storage and supply sections and so is not needed in [[storage]] and [[supply]].

non_depleting_isotopes [optional, list of strings] This is not relevant in the storage and supply sections and so is not needed in [[storage]] and [[supply]].

name or names [required, string or list of strings] A name variable in the `[[[[item]]]]` subsubsections, or names variable in the `[[[[list]]]]` or `[[[[range]]]]` subsubsections is only necessary in the `[[[copy]]]` subsection; in this case this value (or list of values) provides the name of the material(s) that has been copied. This parameter is required when using the `[[[copy]]]` block.

4.4 Universes

In ADDER, a universe is defined in the same way as in most Monte Carlo particle transport codes: a universe is a repeatable/modular geometry pattern. Since ADDER provides a facility to perform fuel shuffling options on these universes, in the neutronics solvers which support this capability, ADDER also allows the user to provide these universes with a name. This is to increase the human-readability of full-core ADDER models.

Specifically, the user can name universes (using items, lists, or ranges as above) as well as providing aliases for a set of universes. These capabilities are provided within the `[universes]` section. As for `[materials]`, the names are provided within the `[[metadata]]` subsection and the aliases within the `[[aliases]]` subsection. These are described separately in the discussion which follows.

This section of this `[universes]` section of the input file is optional. Further, the `[universes]` section is only recognized by neutronics solvers with support for universes.

4.4.1 [universes] Metadata

Since a model can contain a large number of universes and many of them can have similar attributes, ADDER allows the user to provide identification of individual universes via the `[[metadata]]` subsection and its own subsections: `[[[item]]], [[[list]]]`, and `[[[range]]]`. The format and purpose of each are described below.

Any universe from the neutronics input file that is not specified via the means discussed below will be assigned a name that is the same as its' identifying integer in the neutronics input file but cast to a string. As such, the `[universes][[metadata]]` subsection is optional.

Finally, the `[[[item]]], [[[list]]]`, and `[[[range]]]` subsections are applied in the order that the user defines them. This allows the user to define attributes for a wide range and then add exceptions to that range as they are encountered. For example, if universes 1 through 100 are intended to have the same attributes, except for universe 42, then a `[[[range]]]` can be used to define the attributes of 1 through 100 followed by an `[[[item]]]` subsection specifically for universe 42. This is the same behavior as discussed within the `[materials][[metadata]]` section discussion.

4.4.1.1 [[[item]]]

Individual universes can be described by the use of an `[[[item]]]` subsection. The actual subsection header for an item does not need to be exactly `[[[item]]],` but the text within the brackets must start with item. This allows for multiple item subsections to be specified so long as they have unique text after the specific phrase item. For example, if the user wishes to individually specify three items, their subsection names can be: `[[[item]]]`, `[[[item1]]]`, and `[[[item_last]]]`, but they cannot be: `[[[item]]]`, `[[[item]]]`, and `[[[item]]]`.

The description of all the parameters within this item subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

neutronics_id [required, integer] Which universe in the neutronics model specified by the *neutronics_input_file* parameter that the `[[[item]]]` subsection is referring to.

name [optional, string] The name to apply within ADDER to the universe with the `neutronics_id` provided above. If no value is provided, then the name will be a string representation of the `neutronics_id`. Note that this name will *replace* the neutronics ID and needs to be used in the rest of the input file to identify the universe.

4.4.1.2 [[[list]]]

Sets of universes can be described by use of either a `[[[list]]]` subsection or `[[[range]]]` subsection. This section describes the `[[[list]]]` subsection.

Like `[[[item]]]`, the actual subsection header for a list does not need to be `[[[list]]]`, but the text within the brackets must start with `list`.

Unlike the `[[[item]]]` subsection, here a list of specific `neutronics_ids` from the neutronics input file are acted on with common parameters.

The description of all the parameters within this subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

neutronics_ids [required, list of integers] A comma or space-separated list of the specific `neutronics_ids` which specify the list of universes in the neutronics model specified by the *neutronics_input_file* parameter the `[[[list]]]` subsection refers to.

names [required, list of strings] The names to apply within ADDER to the universes with the `neutronics_ids` provided above. If this parameter is not provided, then the names for all of the `neutronics_ids` will be a string representation of each `neutronics_id`. If only one name is provided, then that name will be applied to each entry in the list with a _# appended to the end of that name (where # is the universe's `neutronics_id`). Otherwise, one name should be provided for each `neutronics_id` in the `neutronics_ids` list.

4.4.1.3 [[[range]]]

Sets of universes can be described by use of either a `[[[list]]]` subsection or `[[[range]]]` subsection. This section describes the `[[[range]]]` subsection.

Like `[[[item]]]`, the actual subsection header for a list does not need to be `[[[range]]]`, but the text within the brackets must start with `range`.

Unlike the `[[[item]]]` subsection, here, a user-specified range of `neutronics_ids` from the neutronics input file are acted on with common parameters. This range is inclusive, meaning the start and end points are in range.

The description of all the parameters within this subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

neutronics_id_start [required, integer] A single `neutronics_id` integer which denotes the starting integer of the range that this subsection applies to. The start integer is inclusive. This value is required.

neutronics_id_end [required, integer] A single `neutronics_id` integer which denotes the ending integer of the range that this subsection applies to. The ending integer is inclusive. This value is required.

exclude_neutronics_ids [optional, list of integers] This parameter is a list of `neutronics_id` integers within the range specified by `neutronics_id_start` and `neutronics_id_end` that are not to be included in this subsection's definition. If no values are provided then no `neutronics_ids` are excluded.

names [optional, list of strings] The names to apply within ADDER to the universes with the `neutronics_ids` provided above. If no value is provided, then the names for all of the `neutronics_ids` will be a string representation of the `neutronics_id`. If only one name is provided, then that name will be applied to each entry in the range with a _# appended to the end of that name (where # is the universe's `neutronics_id`). Otherwise, one name should be provided for each `neutronics_id` in the range (except the excluded ids).

4.4.2 Aliases

Since there can be many universes within a model, it may be useful to refer to sets of them with just one name. This name is referred to as an alias. For example, all universes within a reactor's fuel assembly can be assigned to an alias instead of repeating this alias set every time.

More specifically, if an alias is defined, it will be replaced by the ADDER input processor by the set of universes it refers to for all operations defined in the `[operations]` section, thereby significantly reducing the input the user must specify.

The `[[aliases]]` subsection is composed of any number of arbitrarily named subsubsections, each of which is described via the following parameters:

name [optional, string] The name of the alias; this is the name that should be used instead of the individual universes in the set. If this parameter is not provided, then the name of the subsubsection will be the name. The name must *not* be an integer number as that could conflict with neutronics ids.

set [required, list of strings] The list of universe names which compose this set. Note that if any universe names include a comma, then that name must be enclosed in quotations to tell ADDER that the comma does not denote the next entry in the list, but rather is part of a name. This parameter must have at least one entry.

4.4.3 Storage and Supply

If a universe is included in the neutronics model but not assigned to a region, then ADDER will treat it as a universe that is "in supply" (a universe that copies of it can be reintroduced later as if it was from a factory or other supply chain). However, the user may wish to reassign that universe from supply to being a storage universe (a universe that is taken out of core to be introduced into the reactor later). Finally, the user may wish to make a copy of a universe that is in the core.

All of these options are handled by the `[[storage]]` and `[[supply]]` sections. Both sections are optional and only included if the user wishes to perform one of the actions detailed above. Both are discussed together in this manual since the only difference in the input format is if the `[[storage]]` or `[[supply]]` section header is used. Note that this input is very similar to the equivalent in the `[materials]` section, however, this `[universes]` equivalent provides no facilities to define the densities or to set the depleting status of materials. Therefore, the `[materials][[storage]]` or `[materials][[supply]]` blocks must also be specified as necessary to define those material-specific attributes.

If present, the `[[storage]]` and `[[supply]]` sections contain one of the following subsections: `[[[copy]]]` (copy a universe) or `[[[redefine]]]` (redefine a universe not in the reactor from the default of "supply" to "storage"). Note that the `[[[redefine]]]` subsection redefines a universe to be assigned the category ("storage" or "supply") of the parent section under which `[[[redefine]]]` is included. In other words, when the `[[[redefine]]]` subsection is under the `[[storage]]` section, the listed universe will be redefined from "supply" to "storage".

Within the `[[[copy]]]` and/or `[[[redefine]]]` subsections are subsections named `[[[[item]]]]`, `[[[[list]]]]`, and/or `[[[[range]]]]`. These contain the same data as defined earlier in the universe metadata section, except for the following modifications:

name or names [required, string or list of strings] For `[[[copy]]]` only. A name variable in the `[[[[item]]]]` subsubsections, or names variable in the `[[[[list]]]]` or `[[[[range]]]]` subsubsections is only necessary in the `[[[copy]]]` subsection; in this case this value (or list of values) provides the name of the universes that have been copied. This parameter is required when using the `[[[copy]]]` block.

Warning: These universe-wide storage or supply definitions are applied *after* the material storage/supply definitions and therefore since materials are within universes, the material definitions will be over- written by these universe definitions, if the materials modified by the `[materials][[storage]]` or `[materials][[supply]]` blocks are within the universes modified here.

4.5 Tallies

In ADDER, it is possible to associate user-defined tallies from the original neutronics input to specific universes or materials that are moved into and out of the core, including in-core components as well as storage and supply components. Via this capability, the tallies and their additional specifications are handled by ADDER to follow specific components (universes/materials) according to the their movement.

As for [materials] and [universes], ADDER allows the user to provide identification of individual user tallies to be processed and handled by ADDER during fuel management operations via the `[[item]]`,`[[list]]`, and `[[range]]` subsections. The format and purpose of each are described below. Section 5.1.3 provides a specific description of the management performed for tally objects in MCNP. The `[tallies]` section is optional and only required if users want for the tallies defined in the initial neutronics input file to "follow" identified components (materials or universes) throughout the fuel management operations.

Any user-defined tallies from the neutronics input file that is not specified via the means discussed below will not undergo processing, and ADDER will retain these tallies unchanged. Moreover, the user has also the possibility to explicitly define specific tallies as "unprocessed". During fuel management operations, each user tally (processed or not) is included in the neutronics input generated for the initial computations that ADDER performs (e.g., the predictor step of a predictor/corrector depletion). For tallies that are not defined using the `[tallies]` section, users should be aware that if they are associated to a component that is moved out of the reactor as a result of fuel management operations (e.g., put in storage) the neutronics calculations may fail and the ADDER run may be aborted. As such, in models with fuel management, these "unprocessed" (i.e., not managed by ADDER) tallies are best used for global or mesh-based tally results.

The default tally cards, defined via a tally number of 0 (E0, TM0, etc.), can be included in the initial neutronics input. They are automatically assigned by ADDER to each user-defined tally and rewritten by replacing the identifier 0 with the specific user-tally identifier. The default tally cards are not simply retained with their designations as-is (e.g., E0) in the ADDER-generated input files because they can potentially alter the tallies generated by ADDER for depletion. On the other hand, they are assigned to user tallies one by one. The user should ensure that the processing of these default tally cards results in the desired modifiers for the user-defined tallies.

Finally, the `[[item]]`, `[[list]]`, and `[[range]]` subsections are applied in the order that the user defines them. This allows the user to define attributes for a wide range and then add exceptions to that range as they are encountered. For example, if tallies 1 through 100 are intended to have the same attributes, except for tally 42, then a `[[range]]` can be used to define the attributes of 1 through 100 followed by an `[[item]]` subsection specifically for tallies 42 that could be set as "unprocessed". This is the same behavior as discussed within the `[materials][[metadata]]` and `[universes][[metadata]]` sections discussion.

The implementation of the tally feature is highly dependent on the neutronics solver. Currently, ADDER is tested for the use of MCNP5 v1.60 and MCNP6.2. Additional information about the methodology and implementation of the tally feature for these solvers is found in Section 7.1.2.

4.5.1.1 [[item]]

Individual tallies can be referred by use of an `[[item]]` subsection.

The description of all the parameters within this item subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

tally_id [required, integer] The tally card identifier in the neutronics model.

type [required, string] The type, i.e., "universe", "material", or "unprocessed", to be linked to the tally object and referring to the `tally_id`.

4.5.1.2 [[list]]

Sets of tallies can be described by use of either a `[[list]]` subsection or `[[range]]` subsection. This section describes the `[[list]` subsection.

Like `[[item]]`, the actual subsection header for a list does not need to be `[[list]]`, but the text within the brackets must start with `list`.

Unlike the `[[item]]` subsection, here a list of specific `tally_ids` from the neutronics input file are acted on with common parameters.

The description of all the parameters within this subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

tally_ids [required, list of integers] A comma or space-separated list of the `tally_ids` which specify the tally card identifiers in the neutronics model.

type [required, list of strings] The type, i.e., "universe", "material", or "unprocessed", to be linked to the tally objects with the `tally_ids` provided above.

4.5.1.3 [[range]]

Sets of tallies can be described by use of either a `[[list]]` subsection or `[[range]]` subsection. This section describes the `[[range]]` subsection.

Like `[[item]]`, the actual subsection header for a range does not need to be `[[range]]`, but the text within the brackets must start with `range`.

Unlike the `[[item]]` subsection, here, a user-specified range of `tally_ids` from the neutronics input file are acted on with common parameters. This range is inclusive, meaning the start and end points are in the range.

The description of all the parameters within this subsection are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

tally_id_start [required, integer] A single `tally_id` integer which denotes the starting tally card identifier in the neutronics model. The start integer is inclusive. This value is required.

tally_id_end [required, integer] A single `tally_id` integer which denotes the ending starting tally card identifier in the neutronics model . The ending integer is inclusive. This value is required.

exclude_tally_ids [optional, list of integers] This parameter is a list of `tally_id` integers within the range specified by `tally_id_start` and `tally_id_end` that are not to be included in this range. If no values are provided then no `tally_ids` are excluded.

type [required, list of strings] The type, i.e., "universe", "material", or "unprocessed", to be linked to the tally object with the `tally_ids` provided above.

Note: Users should be aware that "unprocessed" tallies are copied "as is" if the `include_user_tallies` flag for the specific operation is set to `True`. As a result, these tallies may become ill-defined as a result of fuel management operations. Users are advised to remove these tallies from the initial neutronics input file to prevent ADDER from exiting calculations prematurely due to errors in the neutronics input files.

Note: If indicated explicitly in the [tallies] subsection, surface-based or mesh-based tallies are retained "as is" in the neutronics input files generated by ADDER. These tallies are not modified by ADDER and, as such, are treated as tallies of the "unprocessed" type.

4.6 Control Groups

As described above, the user can define "control groups" which offer the user a simpler interface for moving control devices during an analysis. These are defined within an optional `[control_groups]` section.

Within the `[control_groups]` section are subsections for each group. The name of each group is defined by the name of the subsection. That is, a control group named `bank_1` will be located within a subsection of `[control_groups]` named `[[bank_1]]`.

Within this `[[<group_name>]]` subsection are the following attributes:

type [required, string] The type of objects in this group. Valid options are `surface`, `cell`, and `universes`.

Note: Only the surface type has been implemented at this time, an error will be raised if other types are provided.

set [required, list of string] The list of component IDs (names or integer identifiers) included in this control group. For example, this would include the surfaces for all the rods and the tops, bottoms, etc., of each. This list must have at least one entry.

axis [required, string] The single axis of motion the control group can move across. Valid values are: `x`, `y`, `z`, `roll`, `pitch`, and `yaw`. These correspond to the `[[[transform]]]` block options.

angle_units [optional, string] Whether `roll, pitch`, and `yaw` are provided in units of `degrees` or `radians`. If not provided, the default is `degrees`.

4.7 Operations

All previous sections have been for assigning and defining information needed to properly understand and perform an operation or series of operations. The `[operations]` section is where the specific operations to perform are described to ADDER. ADDER currently supports the following operations: `calc_volume, deplete, shuffle, revolve, transform, geometry_sweep, write_input` and `write_depletion_lib`. These are described in detail below.

These operations are encapsulated within a particular case subsection, allowing the user to separate operations into completely arbitrary phases of operation, such as a portion of reactor operations between fuel movements. **For consistency, the usage of this subsection is required, even if there is only one operation to perform.**

The cases and the operations within the cases are performed in the same order as specified by the user. Further, each case can contain multiple operations, even of the same type. These cases, operations, and operation steps are all output to the log and the output HDF5 file in the order they are presented by the user in the input file.

The remaining portion of this input guide will present the description of all the parameters within this subsection, including the meaning of the parameters, the expected type of the data, and whether it is optional.

Finally, Section 4.8, Examples of Fuel Management Operations, provides simple input and visual examples of the operations described in Sections 4.7.2 to 4.7.10.

4.7.1 case (parent subsection)

The case subsection of the operations section has no required name, allowing the user to name it whatever descriptive name is desired.

Whatever it is named, the case subsection has only one parameter:

label [optional, string] The label parameter is a name for this operations case block. If provided, this is the label of this operational block which will be referenced throughout the ADDER output and results files. If this value is not present, then the case subsection name is used as the label instead. This strategy allows the user to use consistent case block naming throughout all of their models but apply labels to provide more specifics as needed.

The following subsubsections, if provided, present ADDER with the set of operations to execute, in the order of their appearance within the case. If more than one of each of the following subsubsection operation types exist within a case, then each must start with the subsubsection names, but be appended by some unique identifier to differentiate. For example, the user cannot specify two `[[[deplete]]]` subsubsections within a case, but they can specify a `[[[deplete]]]` and `[[[deplete-1]]]` subsubsection. Note that it is perfectly acceptable to have a subsubsection named `[[[deplete]]]` in one case block and another subsubsection named `[[[deplete]]]` in a different case block.

Finally, the user is reminded that the operations that are described in the following subsections (Sections 4.7.2 to 4.7.10) must be included within a `[[case]]` block, even if a single operation per case is defined.

4.7.2 calc_volume

The `[[[calc_volume]]]` subsubsection allows the user to compute the volumes of all materials present in the core. This may be useful in the case of large models with millions of regions or when the volumes may change in time.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

only_depleting [optional, Boolean] Whether the volume should be determined for only the depleting materials (`True`) or not (`False`). Selecting `True` can significantly reduce the runtime of stochastic volume computations. Defaults to a value of `True`.

target_uncertainty [optional, float] The target uncertainty (in units of percent) to achieve with the stochastic volume calculation. Defaults to a value of 10^{-3} %.

Note: This is only necessary when using a Monte Carlo-based neutronics solver.

maximum_histories [optional, int] The maximum number of histories to run in the stochastic volume calculation; the volume calculation will stop when this number of histories are met, even if the target uncertainty is not reached. Defaults to a value of ten billion (10^{10}) histories.

Note: This is only necessary when using a Monte Carlo-based neutronics solver.

lower_left [optional, list of three floats] The x, y, z coordinates of the lower-left corner of a rectangular parallelepiped bounding box to use when sampling volumes. Provided in units of cm.

upper_right [optional, list of three floats] The x, y, z coordinates of the upper-right corner of a rectangular parallelepiped bounding box to use when sampling volumes. Provided in units of cm.

cylinder_bottom [optional, list of three floats] The x, y, z coordinates of center of the bottom of the z-oriented cylinder bounding volume to use when sampling volumes. Provided in units of cm.

cylinder_radius [optional, float] The radius of the cylindrical sampling volume to use in units of cm.

cylinder_height [optional, float] The height of the cylindrical sampling volume to use in units of cm.

Note: If a parallelepiped bounding volume is used, then *both* the `lower_left` and `upper_right` parameters must be provided. If a z-oriented cylinder or a z-oriented cylindrical shell sampling volume is desired, then the `cylinder_bottom`, cylinder_radius, and cylinder_height parameters all must be provided. If the required values are not provided, an error will be raised.

4.7.3 deplete

The [[[deplete]]] subsubsection provides the reactor power history.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

powers [optional, list of floats] The powers parameter is a list of floating point numbers providing the power for each of the time intervals provided in the durations parameter. The powers are interpreted as having units of MW. One of the fluxes or powers parameters must be provided. If present, one or more positive values must be provided. For the reasons discussed in Section 2.3.1, Re-normalization of Power Density, it is recommended that a single value (i.e., time interval) is used for the first [[[deplete]]] operation after a fresh material is inserted in the reactor.

fluxes [optional, list of floats] The fluxes parameter is a list of floating point numbers providing the one-group flux level volume-averaged over all spatial depleting domains for each of the time intervals provided in the durations parameter. The fluxes are interpreted as having units of n/cm^2-sec. One of the fluxes or powers parameters must be provided. If present, one or more positive values must be provided. For the reasons discussed in Section 2.3.1, Re-normalization of Power Density, it is recommended that a single value (i.e., time interval) is used for the first [[[deplete]]] operation after a fresh material is inserted in the reactor.

durations [required, list of floats] The durations parameter is a list of floating point numbers providing the time duration for each of the powers provided in the powers parameter. The durations are interpreted as having units of days. One or more positive values must be provided.

execute_endpoint [optional, Boolean] This flag sets whether ADDER shall execute the neutronics solver at the final time of this depletion block. A value of True indicates that ADDER will execute the final point of the block, whereas a value of False will not. If not provided, this parameter defaults to True. The execution status of the endpoint has no bearing on future depletion steps within the same ADDER calculation, it is provided as a way for the user to obtain the endpoint of a particular state (including e.g., reactivity, isotopic compositions). As such, to obtain the end-of-timestep material inventories in the HDF5 file, such as when fuel management requires the end-of-cycle isotopic compositions, this must be set to True. A value of False is generally preferred for computational efficiency for deplete operations. Note, however, that if the flag is not set to True for the last deplete operation in an ADDER calculation, the final reactor state (including isotopic compositions) would not be recorded in the HDF5 output file. As such, users who need this information should always set the flag to True for the last deplete operation.

depletion_method [optional, string] This is an override of the global *depletion_method* defined in the metadata section. The description is the same for this parameter as before.

depletion_substeps [optional, integer] This is an override of the global *depletion_substeps* defined in the metadata section. The description is the same for this parameter as before.

renormalize_power_density [optional, Boolean] This flag sets whether the material-wise power density and burnup values are to be renormalized following the specific `[[[deplete]]]` operation to ensure consistency with the `powers` indicated for the operation in this subsection, if applicable. If this flag is set to `True` (default), the material-wise power density and burnup as calculated by ADDER are renormalized to be consistent with the power input. On the other hand, the total fission density is not renormalized. If this flag is set to `False` the renormalization is not performed. In either case, ADDER will provide warnings when the calculated power densities sum to a total power that is more than 1% different with respect to the `powers` values provided for the `[[[deplete]]]` operation. The renormalization constant is stored in the ADDER output. For further explanation on the reasons why a renormalization may be necessary and for best practices approaches to avoid inconsistencies, please refer to the description of the parameter of the same name in the Metadata (Section 4.2).

include_user_tallies [optional, Boolean] A Boolean value denoting whether the created neutronics input file should include the tallies (if applicable) the user requested in the initial `neutronics_input_file`. This includes both "processed" tallies (i.e., `material` or `universe` type, as defined in the `[tallies]` subsection) and "unprocessed" tallies. The former are modified by ADDER as a result of fuel management operations, while the latter are copied "as is". This parameter defaults to `True`.

4.7.4 shuffle

The `[[[shuffle]]]` subsubsection provides information necessary to perform a movement ("shuffle") of a piece within the reactor.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

type [optional, string] The type of shuffle to make, i.e., whether it is a movement of a `material` or `universe`. If this parameter is not provided, then a `material` shuffle is assumed.

Note: A material move is valid for any neutronics solver; the universe type is only valid in neutronics solvers which provides facilities for nested geometries, commonly referred to as universes. This, therefore, is only currently a valid option for the MCNP neutronics solver.

moves [required, list of strings] A list of the names (or aliases) of the objects (whose type is defined via the `type` parameter) to move. The first entry of this movement list will displace the second, the second will displace the third, and so on and so forth. If the first object in the list is currently in the reactor, then the last object in the list will replace the first object in its original location. If the first object is in storage or is supply, then the last object will be placed in storage. If the move set includes a supply object, it must be located in the first position of the list. Finally, if the first object is a supply object, then a copy of it will be created and be given a new name of name[#] where name is the name in storage, and the # is the number of copies of this material introduced into the reactor model. This list must have at least two entries.

If an alias name is used, then all other entries in the list must also be aliases, each with the same number of objects to move. Using aliases in this list allows for each entry of the alias to move to the position of its counterpart in the next alias.

Note: If a `shuffle` operation is performed on a `universe` on which a `transform` operation was previously applied, the `transform` is carried to the next location. As such, if an element was, for example, rotated and flipped before the `shuffle`, it will be rotated and flipped by the same angles after the `shuffle`. The `transform` is applied on top of any transformation that is specified for that location in the initial neutronics input file (e.g., using a `tr` card in MCNP). In other words, after a `shuffle`, all the `transform` operations performed by ADDER are carried by the `universe` on which they are performed and are applied on top of transformations specified in the initial neutronics input file for the target locations.

4.7.5 revolve

The `[[[revolve]]]` subsubsection provides information necessary to perform a rearrangement of a set of pieces within the reactor such that it represents a revolution of a provided number of degrees. More specifically, this operation is condensed input that allows for a `[[[shuffle]]]` operation that mimics a revolution of assembly positions. This is different from the `[[[transform]]]` operation in that the `[[[revolve]]]` operation only shuffles the assignments of materials/universes without actually changing the orientation of each material/universe. The `[[[transform]]]` operation on the other hand actually changes these orientations. In other words, the `[[[revolve]]]` can be thought as the equivalent of "rotating" a set of elements through different positions. For example, for a 3D Cartesian set of positions, this is equivalent to "rotating" a matrix by a given number of degrees by moving all of the elements identified in the matrix to a different location. Since not all the combinations of geometries, set sizes, and angles make sense, the parameters of this operation only allow certain combination of values.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

type [optional, string] The type of revolution to make, i.e., whether it is a movement of a `material` or `universe`. If this parameter is not provided, then a `material` revolution is assumed.

Note: A `material` move is valid for any neutronics solver; the `universe` type is only valid in neutronics solvers which provides facilities for nested geometries, commonly referred to as universes. This, therefore, is only currently a valid option for the MCNP neutronics solver.

geometry [optional, string] whether this is a 3D Cartesian (`cartesian`) or hexagonal (`hexagonal`) revolution to perform. Only 3D Cartesian geometries are currently supported and error is raised if a value of `hexagonal` is provided. This parameter defaults to a `cartesian` geometry.

shape [required, list of integers] This parameter is a list of two or three integers, depending on the value of the geometry parameter. Either way, it provides the dimensions of the flattened array provided in `set`. If geometry is `cartesian`, then the first value is the number of entries in the z-dimension, the second is the same for the y-dimension, and the final is the same for the x-dimension. If `geometry` is `hexagonal`, then the first value is the number of rings, and the second is the number of values in the z-dimension.

Note: Since a revolution in the XY-plane of a non-square set is not physically possible, the y- and x-dimension shapes must be the same if xy_degrees is not 0 or 180. ADDER will check for this and raise an error if this is not followed.

set [required, list of strings] The list of the names (or aliases) of the objects (whose type is defined via the `type` parameter). This parameter must have at least one value. The set is a flattened list of the 2D or 3D matrix of items (with the dimensionality depending on `geometry`). If `cartesian` value of geometry is specified, then the first entry is located at the upper-left position of the top-plane and the last entry is in the lower-right position of the bottom plane. If `hexagonal` value of `geometry` is specified, then the first entry is located in the central position of the top-plane, the second entry is located in the first position (aligned with the positive x axis) of the first ring in the top-plane, etc. The last entry therefore is the last entry in the outer ring of the bottom plane. Despite this list being provided as a 1D list, it will be interpreted to a 3D array via the dimensions provided in the `shape` parameter.

xy_degrees [required, integer] The above options define what to revolve, `xy_degrees` and `z_flip` describe how to revolve it. `xy_degrees` is a single integer describing how many degrees to revolve the set in the x-y plane. If geometry is `cartesian`, then the allowed values are `0, 90, 180, 270`, or `360`. If `geometry` is `hexagonal`, then the allowed values are `0, 60, 120, 180, 240, 300`, or `360`.

z_flip [optional, Boolean] This parameter is a Boolean describing whether the set should be flipped in the z-dimension (i.e., top plane becomes bottom plane, etc.). If this value is not provided, then a value of `False` will be assumed, implying no flipping is to be done.

4.7.6 transform

The `[[[transform]]]` subsubsection provides information necessary to perform a rotation and translation of a requested object. This operation is not supported by all neutronics solvers nor for all geometry object types. For example, in MCNP, this operation is only supported on universes and cells. It is not supported for materials since a rotation or translation of a material does not make physical sense as a space-less quantity.

A `transform` operation may be defined with either a control group, or by an explicit selection of the components, their axes, etc. The former is defined with the `group_name` and `value` parameters, while the latter is defined with the `type, set, roll, pitch, yaw, angle_units`, and `displacement` parameters.

ADDER will not perform transform operations on storage or supply universes. If a transform operation is requested on a supply or storage universe, ADDER will exit with an error. Users should review their input file to ensure that transform operations are being performed on universes that are in-core, which may also include unique instances of a supply universe (rather than the supply universe itself).

Note: ADDER will apply the supplied rotations in a manner that is mathematically equivalent to first applying roll, then pitch, and finally, yaw.

If using the control groups for motion, then the axis of motion will be determined by the axis defined in the corresponding control group definition.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

group_name [optional, string] The name of the control group to use, if using this functionality.

reset [optional, Boolean] If this flag is set to `False` (default), the `transform` operation is performed starting from the latest position of the relevant object. If set to `True`, the position of the object is reset to its initial position from the neutronics (e.g., MCNP5) input before the `transform` is performed.

value [required if group_name is present, float] The displacement along the control group named `group_name`'s axis that this move will occur.

type [required if group_name is not present, string] The type of transform to make. Currently, the only valid types are `surface`, `universe`, and `cell` transform in MCNP.

Note: As specified for the `shuffle` operation, if the `transform` is applied to a `universe`, the `transform` is carried by the universe to the next location. As such, if a `transform` operation, for example, rotates an `universe` which is then shuffled into a different location, the same rotation will be applied to the `universe` in this new location (on top of any transformation defined for the location in the initial neutronics input file).

set [required if group_name is not present, list of strings] The list of the names (or aliases) of the objects (whose `type` is defined via the `type` parameter). This parameter must have at least one value. The transform will be performed for each of the entries in this set.

Note: Since ADDER does not provide a facility for naming MCNP Cells, when using this with MCNP to transform cells (per the `type` parameter), the cell id should be provided as the name.

roll [optional if group_name is not present, float] The angle of rotation about the x-axis. This should be provided in the units specified by `angle_units`. This is the first rotation performed. The default is 0.0 degrees.

pitch [optional if group_name is not present, float] The angle of rotation about the y-axis. This should be provided in the units specified by `angle_units`. This is the second rotation performed. The default is 0.0 degrees.

yaw [optional if group_name is not present, float] The angle of rotation about the z-axis. This should be provided in the units specified by `angle_units`. This is the third rotation performed. The default is 0.0 degrees.

matrix [optional if group_name is not present, and roll, pitch, or yaw is not provided, list of nine floats]

The explicit rotation matrix to apply. The 3 x 3 matrix is provided as a flattened (i.e., 1D) array where the first three entries are the first row, the second three entries are the second row, etc. If this matrix is provided, then yaw, pitch, and roll are ignored and this matrix is used.

matrix_notation [optional if group_name is not present, string]

When the `matrix` attribute is provided, the rotation matrix can be expressed using the notation implemented within the MCNP code, corresponding to the transpose of the rotation matrix used in the common mathematical notation. The two corresponding options are `mcnp` and `common`. The `common` notation, which is usually found in algebra textbooks, is transposed with respect to the `mcnp` notation. If not provided, the default is `mcnp`.

angle_units [optional if group_name is not present, string] Whether `roll`, `pitch`, and `yaw` are provided in units of `degrees` or `radians`. If not provided, the default is `degrees`. This has no effect if the `matrix` parameter is provided.

displacement [optional if group_name is not present, list of floats] An optional displacement vector to include when transforming the universes. As a spatial vector, three values are required. The default is no displacement (i.e., a zero vector).

4.7.7 geometry_sweep

The `[[[geometry_sweep]]]` subsubsection provides the user with the ability to perform multiple transform operations in a single operation on a control group, each followed by a neutronics calculation. This is especially useful for performing integral rod worth or spent fuel criticality analyses sensitivity studies. Importantly, the `[[[geometry_sweep]]]` operation leaves the reactor in the same state that it was before the sweep operation began. That is, e.g., if the control rods were placed at 20 cm above the fuel bottom before an integral rod-worth `[[[geometry_sweep]]]` operation, then they will be at 20 cm above the fuel bottom after that operation as well, even if the `[[[geometry_sweep]]]` evaluated these control rods at varying heights with separate neutronics calculations. This operation does not include user tallies during the neutronics calculation.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

group_name [required, string] The name of the control group to use.

Note: Only the surface type has been tested at this time.

The above does not yet define the set of values to perform the sweep over. This can be defined with either `[[[[range]]]]` or `[[[[list]]]]` subsubsubsections within a `[[[geometry_sweep]]]` subsubsection. The `[[[[range]]]]` is used to define a range of values, whereas `[[[[list]]]]` provides specific values to evaluate.

There can be multiple of each of these subsubsubsections. To do so, the user simply shall make sure the text within the square brackets begins with range or list. That is `[[[[range]]]]` and `[[[[range_2]]]]` can be used. Note that ADDER will evaluate the set of values in the order provided.

Note: In the `[[[[range]]]]` and `[[[[list]]]]` subsubsubsection below, the transformation values provided are in reference to the positions before this transformation begins. That is, a value of 0 will result in no change to the model.

4.7.7.1 range

The `[[[[range]]]]` subsubsubsection includes parameters similar to the Python numpy or MATLAB line space function. This block is defined by the following parameters:

start [required, float] The starting point of the range to evaluate. This will be included in the set.

end [required, float] The final point of the range to evaluate. This will be included in the set if `endpoint` is `True`.

number [required, integer] The number of points to include in the resultant list of values.

endpoint [optional, Boolean] A Boolean value denoting whether the end point is included in the resultant list of values. This parameter defaults to True.

4.7.7.2 list

The `[[[[list]]]]` subsubsubsection simply provides the list of values to include:

values [required, list of float] The specific values to evaluate.

4.7.8 geometry_search

The `[[[geometry_search]]]` subsubsection provides the user with the ability to identify the position that results in a specified k-eigenvalue to a specified tolerance. This is therefore useful for performing a criticality search with control rods. This method utilizes the Regula Falsi search method with rejection and convergence as described by *Gill*, et al. [21]. This operation does not include user tallies during the neutronics calculation.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

group_name [required, string] The name of the control group to use.

k_target [required, float] The target k-eigenvalue to obtain. This must be a positive value.

bracket_interval [required, list of two floats] This parameter is a list of two floats. The first is the lower end of the bracket range and the second is the upper end. The interval bounds are relative to the `reference_position` attribute, as defined below.

target_interval [required, float] The half-width range around the `k_target` that is considered an acceptable result. That is an acceptable result that will fall within `k_target +/- target_interval`. This must be a positive value.

reference_position [optional, string or float] This parameter specifies the reference position for the `bracket_interval`, i.e., the "zero" that the `bracket_interval` refers to. If set to "initial" (default), the reference position is the initial position from the neutronics (e.g., MCNP5) input file. If set to "last", the last position of the control group (as set by a `transform` or a previous `geometry_search`) is used instead. If set to a specific float value, that value is used.

initial_guess [optional, float or string] This parameter is an optional initial guess for the critical position. The value can be set to "last", in which case the `initial_guess` will correspond to the last position of the control group, as set by a `transform` or a previous `geometry_search`. If this is not provided, the initial guess will be the upper bracket position.

min_active_batches [optional, integer] This parameter provides the number of data-accumulating batches to perform, at a minimum, for each iterate. After this many batches, the result will be evaluated and if there is no chance to obtain a result in the `target_interval`, then the iterate will be rejected without expending additional computational time. This value, therefore, should be large enough to obtain significant batches for the estimate of k-effective and its uncertainty to be reasonably determined. Defaults to 30 active batches. This parameter only applies to Monte Carlo solvers.

uncertainty_fraction [optional, float] This parameter sets the stochastic uncertainty to target when the initial computation of an iteration reveals that a viable solution may exist. Specifically, the Monte Carlo solver will be executed with a number of batches that are estimated to achieve an apparent standard deviation on k-effective that `uncertainty_fraction` times the `target_interval`. This parameter must have a value between 0 and 1. Increasing this parameter will require more search iterations but can reduce the Monte Carlo runtime per iteration. Defaults to a recommended value of 0.6. This parameter only applies to Monte Carlo solvers.

max_iterations [optional, integer] This parameter is the maximum number of iterations to perform before aborting the search. Defaults to 30 iterations.

4.7.9 write_input

The `[[[write_input]]]` subsubsection provides the user with the ability to write to disk a version of the reactor neutronics model that will be solved as it exists at the point in the operations history when the `write_input` operation is performed. This is mostly useful for verifying the reactor model after performing fuel management movements before time consuming depletion calculations are performed. This operation also updates the HDF5 file with the most current information.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

filename [required, string] The name of the neutronics input file to write to.

include_user_tallies [optional, Boolean] A Boolean value denoting whether the created neutronics input file should include the tallies (if applicable) the user requested in the initial `neutronics_input_file`. This parameter defaults to `True`.

include_user_output [optional, Boolean] A Boolean value denoting whether the created neutronics input file should include the output tables (as applicable) the user requested in the initial `neutronics_input_file`. This parameter defaults to `True`.

include_adder_tallies [optional, Boolean] A Boolean value denoting whether the created neutronics input file should include the tallies (if applicable) that ADDER will use to perform the depletion simulation. This parameter defaults to `True`.

4.7.10 write_depletion_lib

The `[[[write_depletion_lib]]]` subsubsection provides the user with the ability to write to disk the depletion data library for a material or set of materials. This is useful for creating a depletion library useful for reactors of similar spectra to the one currently under analysis.

Note that ADDER utilizes the ENDF notation to identify reaction channel types, e.g., (n,alpha), which includes the lightest of the reaction products within the parentheses, as discussed in the introduction to Section 6.

The following parameters are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

filename [required, string] The name of the hdf5 file to write.

materials [required, list of strings] The list of material names (or aliases) to include in the library file. If "all_modified" is given, then the depletion libraries for all depleting materials that do not use the default depletion library will be printed. In this case, ``lib_names`` will be ignored and the material names will be the library names.

lib_names [optional, list of strings] The names to apply to the corresponding material library when the depletion library is written to the HDF5 file. There must be one entry in `lib_names` for each entry in materials, including after expanding any aliases used. If this `lib_names` parameter is not provided, then the material names will be used to name the library.

mode [optional, one of ‘r’, ‘r+’, ‘w’, ‘w-‘, ‘x’, ‘a’] The write mode to use when creating the HDF5 file; defaults to w, indicating the file will be created and truncated if it exists.

4.8 Examples of Fuel Management Operations

It is helpful to have a visual representation of the fuel management operations that involve movement of components across the reactor. For the sake of the example, four fuel elements, corresponding to universes 1, 4, 5, 7 and 99 in the base neutronics input file, are named `F1`, `F2`, `F3`, `F4` and `FS`, respectively, in the ADDER input file. FS is outside the geometry of the neutronics model at the beginning of the ADDER execution and hence is treated as supply. Based on the above, the `[universes]` section of the input file will be as follows:

```
[universes]
      [[metadata]]
            [[[list]]]
                  neutronics_ids = 1, 4, 5, 7, 99
                  names = F1, F2, F3, F4, FS
```

A control group is identified by the top and bottom surfaces of a fuel blade inserted around the center of the system. These two surfaces have IDs 41 and 42, respectively, in the neutronics input file. This control blade is assigned the name `control_blade` in the ADDER input file. The `[control_groups]` section of the input file looks as follows:

```
[control_groups]
      [[control_blade]]
            set = 41, 42
            type = surface
            axis = z
```

A graphical representation of the fuel elements' locations at the beginning of the ADDER execution is shown in Figure 4.1.

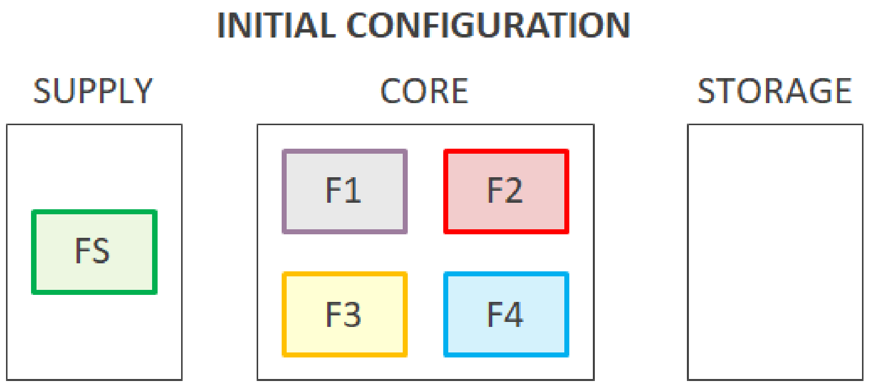

Figure 4.1 Initial configuration of sample ADDER model for fuel management operations

At this point, the following three operations are performed under the first `[[case]]`:

1. `F2` is rotated by 90 degrees counterclockwise using a `[[[transform]]]` operation. In the transform operation, the rotation angle is set via the `matrix` parameter.
2. An instance of the supply element `FS` is moved to replace `F3` using a `[[[shuffle]]]` operation. The `F3` element is moved to storage.
3. The `F1` element is flipped axially using a `[[[transform]]]` operation, via a rotation around the y axis (pitch). In the transform operation, the rotation angle is set via the `pitch` parameter.

The section of the input file for the above operation is as follows:

```
[operations]
      [[case 1]]
            [[[transform_1]]]
                  type = universe
                  set = F2
                  matrix = 0.0, -1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0
            [[[shuffle]]]
                  type = universe
                  moves = FS, F3
            [[[transform-flip]]]
                  type = universe
                  set = F1
                  pitch = 180
                  angle_units = degrees
```

Note that, since two `[[[transform]]]` operations are performed under the same `[[case 1]]`, it is necessary to have different operation section names. Note that it does not matter what follows the "transform" substring, as long as the section name starts with it. A depiction of the resulting fuel elements' locations after `[case 1]` is provided in Figure 4.2.

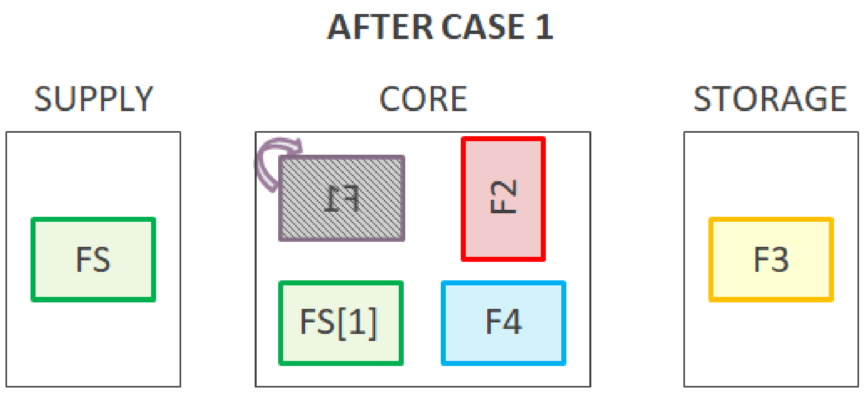

Figure 4.2 Case 1 configuration of sample ADDER model for fuel management operations

A few details are worth highlighting in Figure 4.2:

- The patterned parallel lines and rotating arrow indicate a flipped element.
- The `F3` element has been moved to storage since the first entry of the `moves` parameter in the `[[[shuffle]]]` operation is a supply element.
- The supply element, `FS`, is still a supply element and a unique instance of it, named `FS[1]`, has been created and put into the location that used to be occupied by `F3`.

After this a new `[case 2]` section of the input file is used to perform a single operation: the replacement of the F4 element with a supply element. The relevant section of the ADDER input file looks as follows:

```
      [[case 2]]
            [[[shuffle]]]
                  type = universe
                  moves = FS, F4
```

It should also be noted that, since this `[[[shuffle]]]` operation was included under a different `[[case 2]]`, the operation section name can be identical to the previous shuffle from `[[case 1]]`. A depiction of the resulting fuel elements' locations is provided in Figure 4.3.

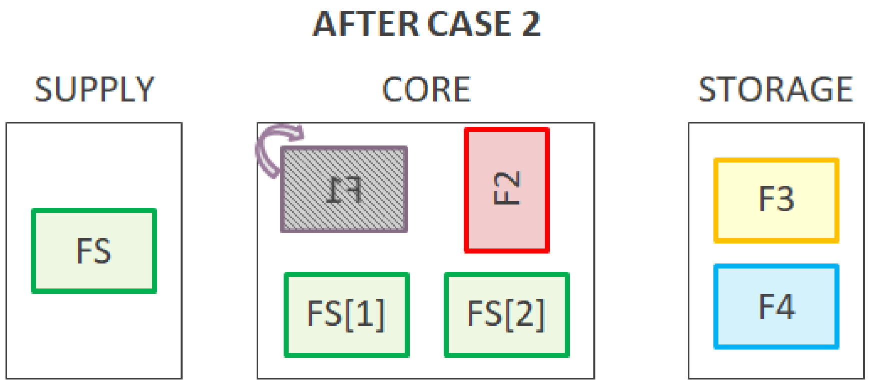

Figure 4.3 Case 2 configuration of sample ADDER model for fuel management operations

`[[case 2]]` shows how FS continues to exist in its unaltered state as a supply element, whereas a new unique instance of it named `FS[2]` and completely separate from `FS[1]` is inserted into the core. `FS[2]` replaces F4 that is moved to storage since it is the last entry of a `[[[shuffle]]]` operation in which the first entry is a supply component.

Finally, an additional case, `[[case 3]]`, is used to revolve all of the fuel elements currently present in the core. The relevant section of the ADDER input file looks as follows:

```
      [[case 3]]
            [[[revolve]]]
                  type = universe
                  geometry = cartesian
                  shape = 1, 2, 2
                  set = F1, F2, FS[1], FS[2]
                  xy_degrees = 180
```

The `[[[revolve]]]` operation effectively treats the components provided within the set parameters as locations in a grid. The number of degrees provided indicates how the entries are re-arranged via a revolution within that grid. In this case, the grid is a 2x2 square grid, which allows for rotations of 90°, 180°, and 270°. This facilitates the fuel management operations input as it allows for multiple `[[[shuffle]]]` operations to be performed under the same `[[[revolve]]]` operation. In this specific case, the components in the set parameter are a flattened array representing the 2x2 matrix of components in their physical xy plane position (as seen in Figure 4.3). Figure 4.4 shows the result of the above 180° `[[[revolve]]]` operation.

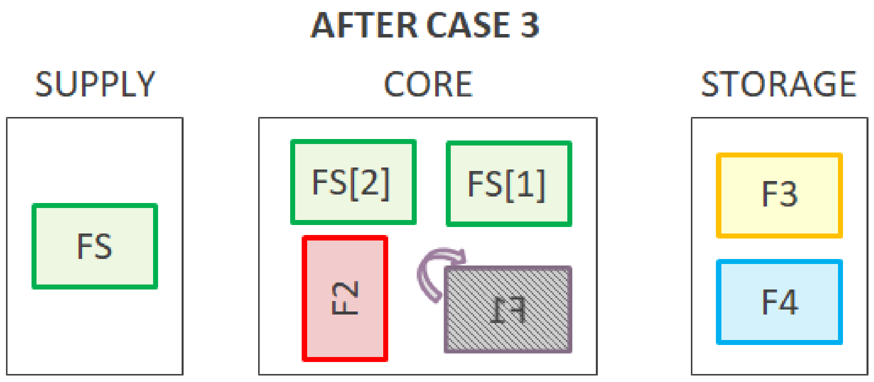

Figure 4.4 Case 3 configuration of sample ADDER model for fuel management operations

The `[[[revolve]]]` operation defined under `[[[case 3]]]` effectively shuffled `F1` into the location of `FS[2]` (and vice versa) and `F2` into the location of `FS[1]` (and vice versa). The same outcome could logically be achieved by setting up two separate `[[[shuffle]]]` operations for each of these shuffles. Although such a simple case is helpful to visualize the behavior of the `[[[revolve]]]` operation, the real strength of using `[[[revolve]]]` over `[[[shuffle]]]` becomes apparent once several components need to be rotated at the same time in a large grid, e.g., to replicate the refueling scheme and reshuffling of fuel elements following a shutdown.

Cases 1 to 3 have provided examples of fuel management operations that do not directly involve any neutronics calculation, in that the neutronics (and depletion) calculations are not depleted as part of those operations directly but instead they are used to set up successive [[[deplete]]] operations. On the other hand, the `[[[geometry_sweep]]]` and `[[[geometry_search]]]` operations allow users to perform operations pertaining to `control_groups` that involve performing several neutronics calculation. In the `[control_groups]` input section above, surfaces 41 and 42 have been identified as the bottom and top surfaces of a single control blade in the middle of the core. This control group was aptly named `control_blade`.

A `[[[geometry_sweep]]]` operation can be set up to calculate the k_{eff} value at several blade extractions by moving the `control_blade` to several different withdrawn positions. This can be helpful to calculate the control blade worth curve. Note that all the values used to indicate positions of a control group are relative displacements with respect to their position in the base neutronics input file. For this example, the control blade was set to its fully inserted position. The blade has a travel range of 58.42 cm. To calculate k_{eff} at each 1/16th of the blade travel length, the following `[[[case 4]]]` is used with a single `[[[geometry_sweep]]]` operation:

```
        [[case 4]]
                [[[geometry_sweep]]]
                        group_name = control_blade
                        [[[[range]]]]
                                start = 0.0
                                end = 58.42
                                number = 17
```

In the above input, the start and end points (again, referring to the position of `control_blade` in the base neutronics input file, which was fully inserted) are identified as 0.0 cm (fully inserted) and 58.42 cm (fully withdrawn). Since the default behavior for a `[[[geometry_sweep]]]` operation is to perform a neutronics calculation at the endpoint of the range, then the operation as defined above sets the `control_blade` control group at 17 different locations and performs a neutronics calculation at all those locations to calculate k_{eff}. The locations are as follows (in cm): 0.0, 3.65125, 7.3025, 10.95375, 14.605, 18.25265, 21.9075, 25.55875, 29.21, 32.86125, 36.5125, 40.16375, 43.815, 47.46625, 51.1175, 54.76875, and 58.42.

The calculated k_{eff} values are retained in the ADDER HDF5 output file. After the `[[[geometry_sweep]]]` operation, the control group is moved back to its location prior to the operation. As such, this type of operation allows a user to calculate the control blades worth at various points during a depletion calculation, without altering the following steps of the depletion.

The other operation that acts on `control_groups` is the `[[[geometry_search]]]` operation. In this case, ADDER performs an iteration to find the location of the desired control group that results in a k_{eff} of a specific input value. For example, this can be used to set control blades at their critical position before performing deplete operations. Unlike the `[[[geometry_sweep]]]` operation, a `[[[geometry_search]]]` does not return the control group to its previous location after completing. Using the same example as above, if users need to calculate the critical blade position for a blade that can travel from 0.0 cm (fully inserted) to 58.42 cm (fully withdrawn) the following `[[[case 5]]]` with a single `[[[geometry_search]]]` operation can be defined:

```
      [[case 5]]
            [[[geometry_search]]]
                  group_name = control_blade
                  k_target = 1.0
                  bracket_interval = 0.0, 58.42
                  target_interval = 0.0005
                  initial_guess = 25.4
```

In the above `[[[geometry_search]]]` operation, the search interval is provided with the `bracket_interval` parameter, that indicates the fully inserted and fully withdrawn positions. Since the user is interested in the critical blade position, the `k_target` parameter is set to exactly 1. The provided `target_interval` indicates that a solution is accepted if it falls within $\pm$50 pcm of the target k_{eff}. The initial guess for the search is provided as 25.4 cm withdrawn. Extensive detail on the search algorithm is provided in Section 7.1.6, Critical Geometry Search Specifics, for MCNP as the neutronics solver.

`[[[geometry_search]]]` operations can be used to deplete the system always starting from its critical configuration and, hence, account for spatial depletion effects that would not otherwise be accurate if the reactor was depleted at fixed blade height in a rundown simulation. Additionally, they are helpful to identify the critical control mechanisms position to make sure that a system is actually able to achieve criticality within the set range of movement and to compare to operational data.

4.9 Flowing Fuel Reactor Analysis

Warning: This molten-salt reactor (MSR) functionality is in the early developmental and validation stage; the user should anticipate changes to the user input while this warning exists. While tested equivalently to the solid-fuel portions of ADDER, the MSR capability was not reviewed with the same scrutiny as the typical solid-fuel capability. If a user requests this functionality, a warning will be written to the screen and log stating that the calculation will exercise non-QA compliant portions of the software.

ADDER provides support for fuel depletion and processing analysis of liquid-fueled reactors (i.e., molten salt reactors, or MSRs). The analysis of such reactors requires additional information which will be provided within this input section.

The `[msr]` section is optional; if it is not used, then ADDER will assume that the reactor is a fixed-fuel reactor. If it is present, then ADDER will treat the reactor like a flowing-fuel reactor. For full MSR functionality, the user must also define either the `msr16` or `msr48` depletion solvers in the `depletion_solver` parameter.

ADDER allows the flowing system to include a user-definable number of flowing systems or flowpaths (e.g., one for each a fuel salt and a blanket salt). These systems can then be defined with custom flowpaths from component to component to mimic the decay, transmutation, and removal of species in the real system.

Materials that are denoted in the `[materials]` block as depleting, but not included in the flowing systems will be depleted via traditional fixed-fuel depletion approaches. If the `msr16` depletion solver is used, then the 16th-order CRAM solver will be used for these materials. Similarly, the use of `msr48` will use the 48th-order CRAM solver for the materials.

Warning: The user shall not perform fuel shuffling with the flowing fuel materials; ADDER currently performs no checks for this.

The `[msr]` section is organized so that the problem metadata is at the base level. This `[msr]` section also includes subsections for each of the flowing systems (the fuel salt and blanket salt from the earlier example). These system sub-blocks contain system-specific metadata and multiple subsubsections, one for each component within the system.

4.9.1 [msr] Metadata

The description of all the parameters within this `[msr]` section are described below, including the meaning of the parameters, the expected type of the data, and whether it is optional:

solve_method [required, string] The methodology to use when determining the evolution of fuel constituents during the depletion and fuel processing. Valid options include `brute`, `tmatrix`, and `rxn_rate_avg`. The `brute` method is a brute-forced approach evaluating the inventory with a separate depletion solve for each component in each system. This methodology is quite slow, with depletion steps taking hours, depending on flowrates and the depletion duration. The `tmatrix` method runs significantly faster than `brute` at equivalent accuracy, requiring up to 30 minutes to run, also depending on the flowrates and depletion duration. Finally, the `rxn_rate_avg` method is an approximate approach which depletes the flowing material as if it was stationary but with reaction rates averaged over the in-core and ex-core portions of the fluid's transit through the system. This `rxn_rate_avg` method is significantly faster than the others, requiring less than a minute to execute.

flux_smoothing_method [optional, string] The flux smoothing methodology to apply when the fuel fluid flows from one region to another in the core. The current valid options are `average` and `histogram.` `average` computes a volume-time-weighted flux to all the in-core regions provided in `in_reactor_flowpath` and uses that flux for depletion. The `histogram` method applies the appropriate flux to each of the in-core regions; however, no smoothing is done to represent the constant flow of the fuel fluid between regions. This parameter defaults to a flux averaging approach.

4.9.2 MSR Systems

Within the `[msr]` block are subsections which define each section. These are denoted with the section header `[[system_<anything>]]` where `<anything>` can be anything (number, name, etc.).

This `[[system_<anything>]]` block contains system metadata, defined next. It is important to note that the feed will be injected to the component defined in the `flow_start` parameter.

name [required, string] The name of the system.

flow_start [required, string] The name of the component that is the 'starting point' for the system flow. This selection is somewhat arbitrary; however, the volume of this component will be used to homogenize the feed. This component must also be one for which there are no bypass paths (i.e., the component's flowrate must be the same as the total system flowrate).

flowrate [required, float] The system mass flowrate, in units of kg/sec.

4.9.3 Feed

The optional feed vector is described for each system with the `[[[feed]]]` subsubsection within the corresponding, `[[system_<anything>]]` block. Within the `[[[feed]]]` block are material subsubsubsections that contains the isotopic composition of materials used in the feed. These are denoted with the section heading `[[[[material_<anything>]]]]`. In `[[system_<anything>]]` and `[[[[material_<anything>]]]],` `<anything>` can be anything (number, name, etc.).

feed_rate [required, list of floats] A comma-separated list of feed rate values ordered to correspond to each depletion step. Each value is the total feed rate captured by the feed stream at each depletion step. The units of these rates are those provided in the `feed_rate_units` parameter. A constant `feed_rate` is defined by providing a single float value.

feed_rate_units [required, string] The units for the feed rate. Valid options are "kg/sec", "kg/day", or "atoms/sec"

feed_material [required, list of strings] A comma-separated list of the feed materials ordered to correspond to each depletion step. If more than one feed material is desired for a specific depletion step, a bracketed space-separated list within the comma-separated list should be used. For example, if there are three depletion steps, and the user wants to use material `f1` in the first step, material `f2` in the second step, and a mixture of `f1` and `f2` in the third step, `feed_material = f2, f1, (f1 f2)`. The user must be mindful that there is no comma between f1 and f2 in `(f1 f2)`. The ratio of f1 and f2 material used in the third depletion step is defined in the `feed_mixture` parameter. The `f1` and `f2` material compositions must be defined in the `[[[[material_<anything>]]]]` blocks. If a constant `feed_rate` is defined, only a single `feed_material` must be provided (This could be a mixture of two materials, e.g.: `(f1 f2)`).

feed_mixture [required, list of strings] A comma-separated list of the feed mixtures ordered to correspond to each depletion step. If more than one feed material is defined for a depletion step, the feed mixture defines the proportion of each feed material defined in `feed_material`. Therefore, following the example described in feed_material, if the user wanted a 3:7 ratio of `f1` and `f2` in the third depletion step, `feed_mixture = 1, 1, (3 7)`. The user must be mindful that there is no comma between 3 and 7 in `(3 7)`. If a constant `feed_rate` is defined, only a single `feed_mixture` must be provided.

density [required, list of floats] A comma-separated list of density values ordered to correspond to each depletion step. The density of the feed fluid. The units of this rate are "g/cc". If a constant `feed_rate` is defined, only a single `density` must be provided.

vector_units [required, string] The units for the feed vector attribute. Valid options are "ao" for atom-percent or "wo" for weight-percent.

Note: Only one `[[[feed]]]` subsubsection can be defined in a `[[system_<anything>]]` subsection.

Note: Providing a single `feed_rate` value triggers ADDER's constant `feed_rate` mode. The user must also define a single value for `feed_material, feed_mixture`, and `density`.

Note: feed_rate, feed_material, feed_mixture, and density input lists must have the same total number of depletion time steps in all the for all the `[[operations]] [[[deplete]]]` blocks. Each of the feed parameters in the list will correspond to a new depletion step whose power/flux and duration is defined in the `[[operations]] [[[deplete]]]` blocks.

4.9.4 Feed Material

The isotopic compositions of the materials used in the feed are defined in this subsubsubsection. Each material is denoted with the section heading `[[[[material_<anything>]]]]` where

`<anything>` can be anything (number, name, etc.) but must correspond to the feed material names used in the `feed_material` parameter.

names [required, list of strings] A comma or space-separated list of the specific elemental or isotopic names, in GND format. The ordering of these elements/isotopes must correspond exactly to the values in the `[[[vector]]]` list. Note that if elemental names are used, then the feed rates will be assigned to each of the naturally occurring isotopes within this element.

vector [required, list of floats] The specific values of the feed vector to use for each of the entries in `[[[names]]]`. The units of these feed rate values are those provided by the `vector_units` attribute. The values provided for elements will be applied to every isotope of that element.

Note: If the feed rate for an isotope is included multiple times, including as an isotope and an element, the last value included will be the one used.

4.9.5 Component

The `[[[component_<anything>]]]` subsubsections are used to define each component within a system. These subsubsections include meta-data and a subsubsubsection with the removal rates, should this be a component where chemical extraction is occurring.

name [required, string] The name of this component.

type [required, one of "generic", "in-core", or "tank"] The type of this component. A `generic` type means no flux-induced reactions treated, but decay and removal are present. `in-core` means that the component is subject to a high enough flux to require the treatment of flux-induced reactions. Finally, a `tank` component is equivalent to a `generic` component, however, the volume of the `tank` component increases after every depletion sub-step as needed based on the feed and removal rates. This increases the duration of time a fluid spends in the tank, mimicking the behavior of a tank. Note that this model does not factor in mixing of constituents of the tank, and some error therefore will result.

volume [required, float] The volume of the component, used for determining the duration of time that a volume of fluid requires to travel through this component. This volume must be provided in units of cubic meters.

mat_name [optional, string] If the `type` of this component is `in-core`, then a material name is required. This material name refers to the names of the materials defined in ADDER's `[materials]` block. This ties the material to a specific region of the neutronics model, which will be used to obtain the flux for this `in-core` region's transmutations.

density [optional, float] The density of this fluid in the component, also used to determine the duration of time that a volume of fluid requires to travel through this component. This density must be provided in units of grams per cubic centimeter. Note that this is not required for `in-core` components because ADDER will instead use the neutronics model's density instead of the value provided by this parameter.

downstream_components [required, list of strings] A list of the component names which are connected to the fluid output of this component.

downstream_mass_fractions [required, list of floats] Each of the downstream components described in the `downstream_components` parameter will receive the mass fraction of flow as described by this parameter. The ordering of these mass fractions must match the ordering of the components in `downstream_components`.

removal_names [required, list of strings] A comma or space-separated list of the specific elemental or isotopic names, in GND format. The ordering of these elements/isotopes must correspond exactly to the values in the `[[[removal_rates]]]` list.

removal_rates [required, list of floats] The specific values of the removal vector to use for each of the entries in `[[[removal_names]]]`. The units of these removal rate values are to be provided as the fraction of that element or isotope that is removed per second. The rates provided for elements will be applied to every isotope of that element.

Note: If the removal rate for an isotope is included multiple times, then the last value included will be the one used.

Note: The elemental removal rates are assigned to every element, natural or not.

Note: If elemental and isotopic removal rates are provided, the isotopic is preferentially used.

5 Output Data Format

During and after execution, ADDER produces an HDF5 file that contains the eigenvalues, material inventories, etc., as a function of simulated reactor time. The following describes the scripts available to access this data from the HDF5 files as well as providing the documentation of the format of this file.

5.1 Output File Interaction

At this point in the ADDER development, only rudimentary scripts are available to ease parsing of the output HDF5 file.

Specifically, a script exists which is named `adder_extract_materials.py` that is located in the `scripts/` directory. This script can create a comma-separated-value (CSV) file from the HDF5 results file with information for a requested material. These CSV files are readily parsed by Excel and many other tools, thus making it a useful format for generic interfacing.

The CSV file produced by the `adder_extract_materials.py` contains the power, k_{eff}, one-group fluxes, Q-recoverable, and isotopic data for a requested material as a function of time.

This script has the following **required** arguments which must be provided in the order introduced below:

results_file: Specifies the path and filename of the output results HDF5 to parse.

csv_file: Specifies the path and filename of the CSV file to write. This file will be overwritten if it exists.

material_name: The name of the material within the model that will be extracted.

The above information is also accessible with the **-h** or **–help** argument to the `adder_extract_materials.py` script.

The following is an example usage pattern for this script:

```
$ adder_extract_materials.py ./results.h5 ./test.csv fuel_1
```

This command will perform the conversion of the data located within `results.h5` and write the resultant data to `test.csv` both in the present working directory. The test.csv file will contain information for the material named `fuel_1` for each case block, operation block, and step block.

5.2 Output File Format

The output HDF5 file format is specified as is provided below. Each attribute and dataset include the expected type, and if it is an array, the expected number of entries where possible. The listed types are those used by the h5py package to write the HDF5 format; of interest boolean values are stored as an HDF5 enum type.

The current version of the output library file format is 2.0.

In general, the file format is such that the root level of the file contains general execution parameters as attributes. The root level also includes a series of groups named `/case_<X>/` where `<X>` is the index of the input block within the `[operations]` block for which this output corresponds to. These numbers are consistent with the user's ordering of the case blocks in the input.

Within each `/case_<X>/` group is a sub-group for each operation. These are named as `/case_<X>/ operation_<Y>/` where the `<Y>` is the number of the operation (e.g., deplete, etc.) within that case block from the input. These numbers are consistent with the user's ordering of the operations in the input. This `operation_<Y>/` group contains no attributes, only subgroups for each state/step of results.

The `/case_<X>/operation_<Y>/` sub-groups are named `step_<Z>` where `<Z>` is the step of the operation of interest. For example, depletion blocks will have a step for each time-step analyzed. This `step_<Z>` block includes the pertinent information for that state point such as the material information, fluxes, powers, eigenvalues, etc. The remaining sections will discuss the specifics of the file contents as necessary.

/

Attributes

- **filetype** (*char[]*) – String indicating the type of file; for this library it will be 'output'.
- **version** (*int[2]*) – Major and minor version of the output file format.
- **name** (*char[]*) – The name of the case modeled.
- **neutronics_solver** (*char[]*) – The name of the neutronics solver.
- **neutronics_exec** (*char[]*) – The neutronics solver executable command including any MPI parameters, for example.
- **neutronics_mpi_cmd** (*char[]*) – The MPI executor used with the neutronics solver.
- **base_neutronics_input_filename** (*char[]*) – The path and filename of the original neutronics input to ADDER.
- **reactivity_threshold** (*double*) – The reactivity threshold used to filter out nuclides present in the neutronics solver model.
- **reactivity_threshold_initial** (bool) – This flag specifies whether the reactivity_threshold is applied to the model's materials initial isotopic/elemental inventory or not.
- **use_depletion_library_xs** (*bool*) – Whether to use the depletion library's cross sections (True) or to have the neutronics solver collect the required cross sections (False).
- **depletion_solver** (*char[]*) – The name of the depletion solver.

- **depletion_exec** (*char[]*) – The depletion solver executable command.
- **num_neutronics_threads** (*int*) – The number of CPU threads used for neutronics solver execution.
- **num_depletion_threads** (*int*) – The number of CPU threads used for depletion solver execution.
- **num_mpi_procs** (*int*) – The number of processors used for neutronics solver execution, with MPI, for example.
- **depletion_chunksize** (*int*) – The number of chunks passed to the depletion threads at a time.
- **begin_time** (*char[]*) – The wall-clock time that ADDER began execution.
- **end_time** (*char[]*) – The wall-clock time that ADDER completed execution.

Datasets No datasets are present

/case_<X>/

The `case_<X>` group contains no datasets or attributes. It only contains the groups for each operation within the case block from the user's input.

Attributes No attributes are present

Datasets No datasets are present

/case_<X>/operation_<Y>/

The `case_<X>/operation_<Y>` group contains no datasets or attributes. It only contains the groups for each step within the operation requested by the user.

Attributes No attributes are present

Datasets No datasets are present

/case_<X>/operation_<Y>/step_<Z>

The `case_<X>/operation_<Y>/step_<Z>` group contains the results for each of the steps within the case and operation solved by ADDER.

Attributes

- **case_label** (*char[]*) – The user-provided label for this case.
- **operation_label** (*char[]*) – The user-provided label for this operation.
- **step_idx** (*int*) – The index of this step (the same as <Z> in the group name).

- **time** (*double*) – The time since beginning-of-life the state represents in units of days.
- **power** (*double*) – The power of this operating interval, in units of `MW`; this is currently only provided if the user specifies the power level for a depletion step.
- **flux_level** (*double*) – The one group flux integrated over the depleting domain of the model for this operating interval, in units of `n/(cm`^{2}`-sec)`.
- **Q_recoverable** (*double*) – The value of the average recoverable energy from fission (in units of `MeV`) determined at this state point.
- **keff** (*double*) – The value of neutron multiplication factor for this state point as computed by the neutronics solver.
- **keff_stddev** (*double*) – The value of neutron multiplication factor standard deviation for this state point, if using a deterministic neutronics solver this value will be 0.
- **nu** *(double)* – The value of the average number of secondary neutrons per fission computed by the neutronics solver.
- **flux_normalization** (*double*) – The value of the flux normalization factor determined at this state point.
- **power_renormalization** (*double*) – The value of the power renormalization factor used for renormalizing power density and burnup values. It is equal to 1.0 if *renormalize_power_density* is set to *False* for the specific deplete operation.
- **step_end_time** (*char[]*) – The wall-clock time that ADDER completed execution of this step.

Datasets No datasets are present

/case_<X>/operation_<Y>/step_<Z>/materials/

Within each `step_<Z>` group there is a `materials` group. This group contains a dataset for each and every material in the model. Each of these material-wise datasets is named with the name of the material for the data it contains. Within these datasets is a 1-D array of a C-struct containing the following data:

Struct Members

- **id** (*int*) – The integral identifier in the neutronics model of this material.
- **is_depleting** (*bool*) – Whether or not this material is to be treated as depleting.
- **status** (*int*) – Whether or not the material is in core (0), in storage (1), or is supply (2).

- **num_copies** (*int*) – The number of copies of supply materials that have been brought into the core; if this material is not a supply, then this will be 0.
- **density** (*double[]*) – The density of this material in `atoms/(barn-cm)`.
- **volume** (*double[]*) – The volume of this material in $\mathtt{cm}^3$.
- **flux** (*double[num_groups]*) – The flux of this material in each energy group, in $\mathtt{n/(cm^2\text{-}sec)}$.
- **power_density** (*double[]*) – The power density for the material in $\mathtt{W/cm}^3$. This number may be renormalized based on the `renormalize_power_density` flag. Power density values refer to the *start* of the *step* (for both the predictor and predictor/corrector schemes), and are assumed constant over it. As such, the reported power density value corresponds directly to the *density* and *atom_fractions* (i.e., isotopic concentrations) reported for the same *step*.
- **burnup** (*double []*) – The cumulative burnup of the material over the irradiation history from the start of the calculation in units of `MWd`. This number may be renormalized based on the `renormalize_power_density` flag. Burnup values refer to the *end* of the depletion step, and assume a constant power density calculated at the *start* (for both the predictor and predictor/corrector schemes). As such the reported value corresponds to the *density* and *atom_fractions* (i.e., isotopic concentrations) at the start of the next *[[[deplete]]] step.*
- **fission_density** (*double []*) – The cumulative fission density of the material over the irradiation history from the start of the calculation in $\mathtt{fissions/cm}^3$. Fission density values refer to the *end* of the depletion step, and assume a constant flux calculated at the *start* (for both the predictor and predictor/corrector schemes). As such the reported value corresponds to the *density* and *atom_fractions* (i.e., isotopic concentrations) at the start of the next *[[[deplete]]] step.*
- **default_lib** (*char[6]*) – The default cross section library for isotopes in this material which is used if any individual isotopes do not have their own library assigned.
- **thermal_libs** (*char[][20]*) – The thermal cross section library identifiers of this material.
- **atom_fractions** (*double[]*) – The list of atom fractions of the isotopes in the material. These values are ordered the same as the isotopes dataset.
- **iso_data** (*Isotope Struct[]*) – An array of isotopic metadata for each isotope in the material. This struct contains the following data:
  - **name** (*char[9]*) – The isotope name, in GND format, of this isotope.
  - **is_depleting** (*bool*) – Whether or not this isotope is to be treated as depleting or not.

– **xs_lib** (*char[6]*) – The cross section library identifier for this isotope.

/case_<X>/operation_<Y>/step_<Z>/control_groups/

Within each `step_<Z>` group there is a `control_groups` group. This group contains a dataset for each and every control group in the model. Each of these control group-wise datasets is named with the name of the control group for the data it contains. Within these datasets is a 1-D array of a C-struct containing the following data:

Struct Members

- **id** (*int*) – The integral identifier in the neutronics model of this material.
- **name** (*char[]*) – The name of the control group.
- **type** (*char[]*) – The type of objects that constitute the control group, can either be `surface`, `cell`, or `universes`.
- **set** (*char[][]*) – The list of component IDs (names or integer identifiers) included in this control group.
- **axis** (*char[]*) – The single axis of motion the control group can move across. Possible values are: `x`, `y`, `z`, `roll`, `pitch`, and `yaw`. These correspond to the `[[[transform]]]` block options.
- **angle_units** (*char[]*) – Whether `roll, pitch`, and `yaw` are provided in units of `degrees` or `radians`.
- **displacement** (*double[]*) – The displacement along the control group `axis` with respect to its position as per the base neutronics input, as a result of `[[[transform]]]` and `[[[geometry_search]]]` operations.

/case_<X>/operation_<Y>/step_<Z>/user_tallies/

Within each `step_<Z>` group there is a `tallies` group. This group contains a datasets for each user tally in the model, Each of these tally-wise datasets are named with the identifier of the tally for the data it contains. Within these datasets is a 1-D array of a C-struct containing the following data:

Struct Members

- **id** (*int*) – The integral identifier in the neutronics model of this tally.
- **type** (*char[]*) – Whether or not the tally follows materials (material) or universes (universe). Tallies not processed by ADDER are indicated as “"unprocessed"
- **input_specification** (*[char[]]*) –The list of strings defining the Fn card and its tally card specifications. Tallies not simulated for the current operation are indicated as “no facet type assigned".

- **material_names** (*[char[]]*) – The list of materials by names linked to the tally.
- **universe_names** (*[char[]]*)– The list of the universes by names linked to the tally.
- **facet_ids** (*int[]*) – The list of the cell identifiers linked to the tally, defined within the Fn card in the neutronics input. For cells grouped within parentheses (General Form), the identifier defaults to 0 for each set of parentheses.
- **tally_matrix** (*float[][][][][][][][][]*) – The tally results represented as a nine-dimensional array, where each array index describes a bin structure of the tally. The tally results are normalized through the flux normalization factor. The array structure is consistent with that generated by the MCNPTools library [11].
- **tally_matrix_err** (*float[][][][][][][][][]*) – The MCNP tally results relative errors represented as a nine-dimensional array, where each array index describes a bin structure of the tally. The array structure is consistent with that generated by the MCNPTools library [11].

If a user tally is not simulated during `operation_<Y>`, it is not linked to facet identifiers in the geometry and results are not generated. Then, all `facet_ids`, `tally_matrix` and `tally_matrix_err` values are set to zero.

6 Depletion Data Library

The following describes methods of producing the depletion data library as well as documenting the format of this file.

It should be noted that ADDER uses reaction identifiers, e.g., (n,alpha), that are consistent with the convention of the ENDF data libraries. According to the ENDF convention, reaction identifiers should indicate the lightest of the products in the parentheses. For example, the Li-6 reaction with a neutron produces an atom of He-4 and an atom of H-3. This reaction could be identified by either the (n,alpha) or the (n,t) identifier. Since H-3 is lighter, the ENDF convention assigns the reaction to channel (n,t). ADDER follows the same convention. ORIGEN2.2 libraries may be inconsistent with this convention [3]. For example, the above-mentioned Li-6 reaction is identified as a (n,alpha) in ORIGEN2.2 data libraries. ADDER automatically converts between the two notations when interfacing with ORIGEN2.2, including when generating HDF5 depletion library files from ORIGEN2.2 libraries.

6.1 Library Generation

ADDER depletion data libraries can be generated either by using the supplied script which converts the libraries supplied with RSICC ORIGEN2.2 distribution, or through manual generation of the depletion data library via either directly writing the HDF5 library or using the DepletionLibrary Python API.

This document will discuss the usage of the ORIGEN2.2 conversion scripts and will document the HDF5 format of these files. The documentation for using the Python API directly will be the subject of future work.

6.1.1 ORIGEN2.2 Conversion

ORIGEN2.2-formatted ASCII libraries can be converted with two formats, each of which are described in the following sub-sections:

1. The user may convert the whole suite of libraries distributed with the Radiation Safety Information Computational Center (RSICC) distribution of ORIGEN2.2 in a single command, OR

2. The user may convert individual ORIGEN2.2 libraries which may have been created to support any specific needs.

Conversion of RSICC Suite

The ORIGEN2.2 libraries are distributed from the RISCC with a set of precomputed one group depletion libraries for a variety of reactor types, including various forms of PWRs, BWRs, and LMFBRs. The `adder_convert_origen22_rsicc_libraries.py` script distributed with ADDER (located in the `scripts/` directory of the ADDER distribution's root directory) can be used to convert all of these official ORIGEN2.2 libraries to an HDF5-formatted library in the format expected by ADDER.

This script has the following arguments:

-d, or –destination: Specifies the path and filename of the HDF5 library file for which the ORIGEN2.2 libraries will be written to. This value defaults to a value of `origen_lib.h5` being written in the present working directory.

-r, or –rsicc: Specifies the path to the RSICC distributed libraries which will be processed.

-o, or –overwrite: Specifies, via a `True` or `False`, whether the HDF5 file specified by the **-d** argument will be overwritten `(True)` or appended to `(False)`. This value defaults to `False`.

-h or –help: Prints help similar to the above to the standard output.

Therefore, the most likely usage pattern for this script is as follows:

```
$ adder_convert_origen22_rsicc_libraries.py -r /ORIGEN/data/
```

This command will perform the conversion of the RSICC data located in `/ORIGEN/data/`, (requiring on the order of minutes) and write the library to `origen_lib.h5` in the present working directory. If `origen_lib.h5` exists, then it will be appended to.

Conversion of Individual Library

It is not uncommon for the user to have created custom ORIGEN2.2 libraries. To convert these custom libraries to the ADDER HDF5 format, the `adder_convert_origen22_library.py` script distributed with ADDER (located in the `scripts/` directory of the ADDER distribution's root directory) can be used to convert all of these official ORIGEN2.2 libraries to an HDF5-formatted library in the format expected by ADDER.

This script has the following **required** arguments which must be provided in the order introduced below:

xslib_filename: Specifies the path and filename of the ORIGEN2.2 cross section and yield library to convert.

decay_filename: Specifies the path and filename of the ORIGEN2.2 decay library to convert.

xslib_ids: A set of three integral identifiers corresponding to the activation, actinide, and fission product cross section and yield library identifiers that should be converted from within the file specified by the `xslib_filename` parameter. Three integer values are required, separated by a space.

decay_ids: A set of three integral identifiers corresponding to the activation, actinide, and fission product decay library identifiers that should be converted from within the file specified by the `decay_filename` parameter. Three integer values are required, separated by a space.

new_name: Specifies the name of the library within the resultant HDF5 file (e.g., `PWRU`). This name can be a string of any manageable length, but it is recommended that spaces are not included in this name to minimize the chances for user-error in referring to this name later.

The script also has the following **default** arguments which, if necessary, are to be provided after the required arguments:

-d, or –destination: Specifies the path and filename of the HDF5 library file for which the ORIGEN2.2 library will be written to. This value defaults to a value of `origen_lib.h5` being written in the present working directory.

-o, or –overwrite: Specifies, via a True or False, whether the HDF5 file specified by the **-d** argument will be overwritten `(True)` or appended to `(False)`. This value defaults to `False`.

-h or –help: Prints help similar to the above to the standard output.

The following is an example usage pattern for this script:

```
$ adder_convert_origen22_library.py ./PWRU.LIB ./DECAY.LIB 204 205 206 1 2 3 PWRU -d
./test.h5
```

This command will perform the conversion of the data located within `PWRU.LIB` and `DECAY.LIB` located in the present working directory and write the library to `test.h5` in the present working directory in a group named `PWRU`. If `test.h5` exists, then it will be appended to.

6.2 Depletion Library Format

The depletion library HDF5 file format is specified as is provided below. Each attribute and dataset include the expected type, and if it is an array, the expected number of entries. The listed types are those used by the h5py package to write the HDF5 format; of interest boolean values are stored as an HDF5 enum type. This information will be useful if manually modifying libraries or creating new libraries.

The current version of the depletion data library file format is 1.0.

The top-level structure of the file is simply a set of library groups, each named by the library name.

/

Attributes

- **filetype** (*char[]*) – String indicating the type of file; for this library it will be 'depletion_data'.
- **version** (*int[2]*) – Major and minor version of the depletion library file format.

Datasets No datasets are present

/<library_name>/

The data within `<library_name>` contains the multi-group data for each of the isotopes within the library. This library is described as follows:

Attributes No attributes are present

Datasets

- **neutron_group_structure** (*double[]*) – Monotonically increasing list of group boundaries, including the lower boundary, in units of MeV.

/<library_name>/isotopic_data/

Within each `<library_name>` group is a group named `isotopic_data` that contains the isotope-specific information for this library. This group includes no attributes or datasets but includes a group for each isotope in the library.

Attributes No attributes are present

Datasets No datasets are present

/<library_name>/isotopic_data/<isotope_name>/

Within each `<isotope_name>` group is the information about the isotope named by `<isotope_name>`, where the isotope name is provided in GND format. This group contains no attributes or datasets, but contains the following groups:

/<library_name>/isotopic_data/<isotope_name>/decay

This group provides the decay data for the isotope `<isotope_name>`. This block is optional and only exists if decay data is available for this isotope. If present, this group contains the following attributes:

Attributes

- **half_life** (*"None" or double*) – The half-life, provided with units specified by the `half_life_units` attribute.

- **half_life_units** (*char[]*) – The units of time used for the decay data. The allowed values are `s, m, hr, d, yr, kyr, Myr`, and `Gyr`.

- **decay_energy** (*double*) – The decay energy in units of MeV.

/<library_name>/isotopic_data/<isotope_name>/decay/<decay type>

This group provides the branching ratio, target isotopes, and yields of those target isotopes for the decay type provided in <decay type>. This decay type can be one of: `alpha, n, n,n, p, p,p, beta-, beta-,n, beta-,n, n, beta-,n,n,n, beta-,n,n,n,n, beta-,alpha, beta-,beta-, ec_beta+, ec_beta+,alpha, ec_beta+,p, ec_beta+,p,p, ec_beta+,sf, IT, sf`. Note that the / in the `ec/beta+` keys have been replaced with _ as / in HDF5 denotes a subdirectory. When the decay type includes spontaneous fission (i.e., `ec_beta+,sf` and `sf` decay types), the targets Dataset should be set to the string *fission* and the yields Dataset should be set to 1.0. Fission yield data should then be provided under the `neutron_fission_yield` group.

Attributes

- **branching_ratio** (*double*) – The branching ratio of this decay.

Datasets

- **targets** (*char[][]*) – The list of isotopes produced by this decay; this includes the secondary particles (e.g., the `He4` particle produced in an `alpha` decay).

- **yields** (*double[len(targets)]*) – The yield of each of the targets in targets list. For example, an `p,p` decay which produces `Zr90` will have a yield for `Zr90` of 1.0 and 2.0 for `H1` to represent the protons.

/<library_name>/isotopic_data/<isotope_name>/neutron_xs

This group provides the neutron cross section data for the isotope `<isotope_name>`. This block is optional and only exists if neutron cross section data is available for this isotope. If present, this group contains the following attributes:

Attributes

- **xs_units** (*char[]*) – The units used for the neutron cross section data. The allowed values are `b, bE24`. The former is the typical barns, and the latter is barns multiplied by 10^{24}.

/<library_name>/isotopic_data/<isotope_name>/neutron_xs/<reaction type>

This group provides the cross section, target isotopes, yields of those target isotopes, and a Q-value for the reaction type provided in <reaction type>. This reaction type can be one of: `(n,gamma)`, `(n,2n)`, `(n,3n)`, `(n,4n)`, `fission`, `(n,p)`, `(n,d)`, `(n,t)`, `(n,3He)`, `(n,a)`, `(n,2nd)`, `(n,na)`, `(n,3na)`, `(n,n3a)`, `(n,2na)`, `(n,np)`, `(n,n2a)`, `(n,2n2a)`, `(n,nd)`, `(n,nt)`, `(n,nHe-3)`, `(n,nd2a)`, `(n,nt2a)`, `(n,2np)`, `(n,3np)`, `(n,n2p)`, `(n,2a)`, `(n,3a)`, `(n,2p)`, `(n,pa)`, `(n,t2a)`, `(n,d2a)`, `(n,pd)`, `(n,pt)`, `(n,da)`. The ENDF notation is used for the reaction type, which includes the lightest of the reaction products within the parentheses, as discussed in the introduction to Section 6. When the reaction type is `fission`, the targets Dataset should be set to the string *fission* and the yields Dataset should be set to 1.0. Fission yield data should then be provided under the `neutron_fission_yield` group.

Attributes

- **q_value** (*double[]*) – The Q-value of the reaction in MeV.

Datasets

- **xs** (*double[]*) – The group-wise cross section values.

- **targets** (*char[][]*) – The isotopes produced by this reaction; this includes the secondary particles (e.g., the He4 particle produced in an (`n,alpha`) reaction).

- **yields** (*double[len(targets)]*) – The yield of each of the targets in `targets` list.

/<library_name>/isotopic_data/<isotope_name>/neutron_fission_yield

This group provides the fission yield data for the isotope `<isotope_name>`. This block is optional and only exists if fission yield data is available for this isotope. This data is needed if fission is included as a neutron `<reaction_type>` or spontaneous fission is included as a `<decay_type>`. If present, this group contains the following datasets:

Datasets

- **isotopes** (*char[][]*) – A list of strings denoting the names of the fission products produced by this fissionable isotope. The order of data in `isotopes` matches that of `yields`. The allowed values are any GND-formatted isotope name.

- **yields** (*double[len(isotopes)]*) – The fission yield values themselves.

7 Neutronics Solver Guidance

This section provides the user with guidance on the neutronics solvers supported by ADDER and any useful techniques for using these solvers with ADDER.

At present, only two neutronics solvers are supported by ADDER: MCNP5 v1.60 and MCNP6.2. The input and output and behavior of these solvers are similar and therefore are discussed together.

7.1 MCNP

The following guidance is provided for using either supported version of MCNP with ADDER.

7.1.1 Automatic Duplication of Materials

ADDER will automatically duplicate all depleting and shuffled materials for each instance used in the MCNP model. This allows each depleting composition to deplete according to the local neutron flux conditions yielding a more accurate solution. Shuffled materials are duplicated to ensure volume and density information is correct as the shuffled material is moved throughout the core. ADDER will similarly duplicate materials used across universes, regardless of their depleting or shuffled status. This is to ensure the status conditions of materials within a universe are correct (i.e., that a supply universe has all of its constituent materials labeled as supply as well).

All of these duplications will increase ADDER's memory usage. To reduce this memory penalty at the moment is to modify the initial MCNP model to decrease the number of unique depleting material regions. This can be done by reducing the spatial extent of the model or making each depleting region larger (effectively applying a coarser depletion mesh).

Further, if ADDER is using the neutronics solver to generate unique depletion libraries (i.e., the `use_depletion_library_xs` is set to `False`), then unique versions of the library are also created for each of the depleting materials. This will increase ADDER's memory usage significantly. To reduce this effect, the user can create a multi-group depletion library of sufficient group structure and use that with `use_depletion_library_xs` set to `True`, and/or only use the neutronics solver to create unique depletion cross sections in regions which require it by usage of the `use_default_depletion_library` parameter for each material that does not need a unique library in the `[materials]` block.

7.1.2 MCNP User Tallies

User tallies are tallies present in the initial MCNP input file provided to ADDER. In ADDER, it is possible to associate user tallies from the neutronics input file to specific universes or materials that are moved during fuel management operations. This capability includes in-core components as well as storage and supply components. During fuel management operations, each user tally is written in the neutronics input generated for the initial computations that ADDER performs (e.g., the predictor step of a predictor/corrector depletion)

Tallies can be associated with tally card specifications (e.g. FMn, En, CFn, …), where n is a unique identifier for the tally. In simple form definition (see MCNP manual), a tally card Fn card and its tally specification cards are defined as follows:

```
Fn:pl C1 C2 … Ck
FMn …
En …
CFn …
…
```

The facet identification numbers (C1 C2 … Ck) of the entities of interest (corresponding to surfaces, cells and geometric points) are listed in the tally card Fn (*pl* refers to the particle to be tracked, i.e., neutron, proton or electron). ADDER is capable of processing tallies defined for facets representing cells (i.e., cell-based tally) as the cells themselves or the materials filling them are moved during fuel management of universes and materials, respectively. Moreover, tallies are cloned with new identifiers `n` if the components being followed are cloned during the ADDER run (e.g., if a copy is made or for supply components). On the other hand, surfaces or geometric points of the original neutronics input file are not changed by ADDER during fuel management operations, and, as such, tallies associated to these entities are not processed by ADDER.

The management of user-defined tallies is explained here below, with different strategies for components identified using universes and materials:

- `universe`: a single universe includes multiple cells. When universes are moved inside and outside the core, the cells belonging to these universes are moved accordingly. The tally card type and its tally card specifications are included in the ADDER-generated neutronics input if all the cells to which they are linked, are included in the simulated geometry. If during the ADDER simulation any cell indicated in the tally card is not included in the simulated geometry (e.g., cells for a component that is put in storage) the tally object is not included in the ADDER-generated neutronics input file for that specific operation. The tally is added back when all of the cells in the original neutronics input file are included in the simulated geometry. As such, it is required that users set up the MCNP input file in such a way that all the cells for a tally identified by the `universe` type in the ADDER input file belong to the same universe. If a universe is cloned (using the `[[copy]]` operation) and a tally object is linked to the cells of this universe, a new tally is created with a new ID and is associated to the cells of the cloned universe.
- `material`: in MCNP, tallies cannot be defined through materials identifiers as the facets of a Fn card. In this case, ADDER internally links tallies to the specific materials filling the cells indicated in the Fn tally card of the initial MCNP input. ADDER tracks the materials of the cells indicated for the tally in the initial MCNP input. As the materials are moved as a result of fuel management operations, ADDER changes the MCNP tally cards based on the cells that are filled by the linked materials. For example, if the material of interest is associated to tally f14 located in cell 101 in the initial MCNP input file and is then shuffled into cell 307, tally f14 will change cell facet 101 to 307 after the shuffle. It is important to clarify that, if a tally is generated and linked to specific materials, no additional tallies are generated if these are moved to cells that are with different Fn tally cards. The tally card and its tally card specifications are written to the ADDER-generated neutronics input file for the relevant operation if all the materials that are associated with the tally (through their cell facets) in the base neutronics input file are included in the simulated geometry. If any material associated with the cells indicated in the tally card in the base neutronics input file is not included in the simulated geometry, the specific tally cards are not written. If at a later point during the ADDER simulation all of these materials are added back into the simulated geometry, the tally is included in the ADDER-generated neutronics input file for that operation.

The ability to associate tallies to components (universes or materials) is designed to “follow” components and allow tracking of nuclear quantities of interest (e.g., fast flux in a specific fuel element’s cladding) as the component is moved throughout the core as a result of fuel management operations. As indicated above, all the facets (e.g., cell IDs) associated with the components in the initial MCNP input file need be present in the simulated geometry at the same time. If one or more facets (e.g., cells) are absent from the simulated geometry (e.g., because an element was moved to storage), the corresponding tally will also be excluded. As such, it is recommended that users make sure that the tally cards in the initial MCNP input only include facets of elements that are moved together as part of fuel management operations (e.g., all the cells are part of the same universe that identifies a fuel element).

7.1.2.1 MCNP Tallies in General Form

ADDER is able to process user tallies defined through the Fn card in General Form (as defined in the MCNP user guide) and General From through repeated structure via the appropriate facet cells identifiers:

General Form: Fn:pl *S1 (S2 ... S3) (S4 ... S5) S6 S7*

General Form: Fn:pl *C1 (C2... C3) ((C4 C5) < (C6 C7[I1 ... I2]) < (C8 C9 C10)) ...*
(repeated structure)

Here are some important considerations that users should be aware of regarding how tally management is handled:

- For the General From with repeated structure, ADDER does not process compound definitions of the Fn card with surface identifiers (i.e., surface-based tallies). These tallies are simply copied and pasted. Only tallies defined through cell identifiers (i.e., cell-based tally) are processed and followed through fuel management operations.
- Shorthand definitions through the universe format (U=#) are not supported, including those in Simple Form.
- When managing tallies within repeated structures, the tallies should be defined with type "`universe`" rather than "`material`". In fact, repeated structures involve the definition of a lattice filled with universes. If a tally is associated with a universe that fills a specific position in a lattice cloned by ADDER, it is recommended to:
  - Assign a unique universe for each position in the lattice.
  - Assign the tally to a lattice structure that is not split up during fuel management operations
  - Define the cell IDs in the MCNP tally card using single lattice coordinate (e.g., F4:n (100<110[-1 0 0])) rather than multiple lattice coordinates (e.g., F4:n (100<110[-1 0 0, 1 0 0]), F4:n (100<110[-1:1 0 0]))

7.1.3 Cross Sections

MCNP assigns cross section libraries (and their processed temperatures) to nuclides by an explicit specification (e.g., the `.72c` in `92235.72c`), an `nlib` keyword on a material card, or a global default library with the `nlib` keyword on the default material card, `m0`. This latter feature is only supported by MCNP6.2. In MCNP, if none of the above are present for a given nuclide, then the first entry in the neutronics cross section library (the `xsdir` file) will be used instead.

During depletion, ADDER introduces new nuclides to an MCNP material and assigns the material default or the global (`m0`) default cross section library identifier.

Since the accurate analysis of an at-power reactor requires that new nuclides be modeled at the right temperature, ADDER requires the MCNP input to include an indication of the default libraries so that new nuclides are modeled with the correct cross sections at the right temperature during the MCNP simulation.

If a starting model does not include this information, it can be included either by adding an `nlib` keyword to every material card, or more simply by including the `m0` card with a specified `nlib` keyword. This `m0` card is not supported by MCNP5, however, it can still be used for this purpose as ADDER will not include it explicitly in the input that it writes for MCNP.

Note: ADDER does not process ZAID numbers (e.g., `92235` for U-235) of nuclides with metastable state higher than 1.

7.1.4 Equivalence of ADDER-calculated Power Density to MCNP f4 and f7 Tallies

It is helpful to establish an equivalence between the calculation of the power density performed within ADDER and the MCNP tally structures normally used to calculate the same quantity. Two tally types that can be used to calculate power density in MCNP are the f4 (flux type) and f7 (energy deposition type) tallies. Assuming that the neutronics cell ID for the depleting region of interest is 42, the corresponding material ID is 238, and the region volume is 1.23 cm^3, the power density can be calculated using the following f4 tally and associated fm4 tally multiplier card and sd4 segment divisor card:

```
f4:n  42
fm4   -1.0 238 -6 -8
sd4   1.23
```

In the f4 tally above, the fm4 multiplier card multiplies the volume-integrated flux obtained by card f4 by the prompt energy release from fission (-8), the microscopic fission cross-section (-6), and finally the cell 42's atom density (-1.0). As the tally is integrated over the volume, this results in a quantity in the units of power. The segment divisor card sd4, used in conjunction with a flux tally such as f4, is used to divide the tallied quantity by the volume, obtaining the final tally in the units of a power per unit volume, i.e., power density.

Similarly, the power density can be calculated using the following f7 tally in conjunction with an sd7 segment divisor card:

```
F7:n  42
sd7   1.23
```

It should be noted that the multiplication times the prompt recoverable energy from fission, microscopic fission cross-section, and atom density is performed by the default energy deposition tally, f7, hence a tally multiplier card is not defined in this case.

Finally, it should be noted that – unlike ADDER – MCNP uses the *prompt released* energy from fission, not the *total recoverable* energy from fission. As such, if one were to obtain the same quantities from MCNP instead of ADDER directly, they would need to use a different flux normalization constant that considers the average *prompt released* energy in the system instead of the *total recoverable* energy.

7.1.5 Transforms, Geometry Sweeps, and Critical Geometry Searches

ADDER allows the user to perform individual transforms, sweeps of transforms, and critical searches using transforms. All of these operate on surfaces, cells, and universes in an MCNP model. In all of these cases, the user must be fully aware of the implications of moving the surface, cell, or universe.

For example, consider the case of using these capabilities to move a model's control rod position with surface transformations. In this case, the user must verify that the surfaces being transformed are not also used in the region specification for cells that are not intended to be transformed.

Further, consider the same above example but being performed with a cell transformation. In this case, the user again must ensure that when a cell is transformed that the MCNP model will correctly fill the space that is now empty after that move. In effect, this means that these cell transformations of control rods generally should be performed within a universe that is filling another cell.

Note: MCNP limits the number of coordinate transformation cards (TR cards) to 999. ADDER will combine equivalent TR cards to efficiently utilize this limited resource; however, this merging is performed at the end of each transform operation. The user should be aware of this limitation when developing their models.

7.1.6 Critical Geometry Search Specifics

1. The component of interest is moved to the lower position specified in the `bracket_interval` parameter. The k-effective and uncertainty are determined as follows:

    a. The calculation is run first using the number of inactive batches and histories per batch as specified in the original MCNP input file. The total batches, however, are modified such that the number of active batches is those specified by the `min_active_batches` parameter.

    b. After MCNP execution, the k-effective and its' uncertainty are obtained from the MCNP output.

    c. This case is then evaluated to see if it includes the target k-effective. Specifically, a 99.5% Confidence Interval (CI) is used, based on the MCNP-reported k-effective uncertainty, to evaluate if the current component position contains a valid solution. If it does, then proceed to step d; otherwise, proceed to the next point of interest (step 2).

    d. Estimate the number of batches necessary to achieve a 95% CI on k-effective that is equivalent to `uncertainty_fraction * target_interval`. If this `uncertainty_fraction` has a value of 1, then the number of batches will yield a k-effective uncertainty that has a 95% CI equivalent to the `target_interval` parameter.

The user should note that this is an a priori estimate of the number of batches, and the resultant k-effective uncertainty will not exactly match this estimate.

   e. Execute an MCNP "continue-run", starting from the previous results, for the number of batches determined in step d. Extract the new k-effective and its' uncertainty.

2. Move the component of interest to the `initial_guess` position. Repeat step 1 for this configuration.

3. Using the previous two points, perform the following:

   a. Increment the number of iterations by 1.

   b. Determine if the last point evaluated is a converged solution. This is done by analyzing if k-effective exists within `k_target +/- target_interval` to a 95% CI. If so, the search is complete.

   c. Otherwise, make sure the number of iterations is less than the user-specified maximum number of iterations (`max_iterations`). If it is, continue, and if not, the search is complete.

   d. If we do not yet have a solution, utilize the previous two iterations positions and k-effective values to compute a differential worth. Use this worth to estimate the position that yields a k-effective equal to the `k_target` parameter. Move the geometry to that position.

   e. Perform the computation as described in steps 1.a through 1.e.

   f. Loop through step 3 until convergence or the maximum iterations are achieved.

One of the main benefits of defining control groups is the ability to perform `geometry_search` operations. This kind of operation iteratively moves the selected control group and performs neutron transport calculations until a user-specified target k_{eff} value is obtained. This allows, e.g., to find the critical (k_{eff}) control blades height at any point during the depletion and fuel management calculation. Note that the control group and target value are not limited to those in the latter example, but rather can be any value and group of surfaces, cells, or universes that are to be moved together, as defined in Section 3.3, Control Groups.

A `geometry_search` operation is defined, at a minimum, by the following parameters (see Section 4.7.8, `geometry_search` for a complete list of input parameters):

- A search range (`bracket_interval`), which is used to set the boundaries for the successive `transform` operations that are performed as part of the `geometry_search`. The range consists of the maximum (z_{max}) and minimum (z_{min}) search values. The reference value (i.e., the "zero") of the search can be set in the specific input file subsection and defaults to the initial position of the surfaces, cells, or universes that are grouped into the specific `control_group`.
- The target k_{eff} value (`k_target`). This value can be any number greater than 0. It should be noted that if the k_{eff} value can't be found within the indicated interval, the ADDER calculation exits with an error.

- The half-width interval around the target k_{eff}, Δk_{target}, for which a `geometry_search` operation is considered converged (`target_interval`).

The convergence iteration for the geometry_search operations relies on the Regula Falsi search method [21]. This method is a typical search approach, with modifications to 1) not waste computational power on superfluous histories, and 2) to accommodate statistical uncertainty in the result. The algorithm can be described via the following steps and:

1. The algorithm is initialized differently, depending on the iteration number:
    a. If this is the first iteration, the `control_group` location, z_{new}, is set to the lower end of the provided search range, z_{min}.
    b. If this is the second iteration, the `control_group` location, z_{new}, is set to its guess location, z_{guess}, (`initial_guess`, if provided) or to the upper end of the provided search range, z_{max}.
    c. Otherwise, if this is any other iteration, the `control_group` location is set to the updated value of z_{new} calculated from the previous iteration.
2. A transport calculation is performed with a low number of histories to get the value of k_{eff} with the control group at the z_{new} location, k^*_{new}, and its associated statistical standard deviation, $\sigma_{k^*_{new}}$. This low-history neutronics calculation allows to save time on guesses that are far from the target by performing an early rejection check.
3. After the low-history neutronics calculation, ADDER checks whether the calculated 99.5% confidence range (expressed as $k^*_{new} \pm 2.807 \times \sigma_{k^*_{new}}$) overlaps with the target interval range (calculated as $k_{target} \pm \Delta k_{target}$). A graphic depiction of the early rejection check is provided in Figure 7.1. If the early rejection check is passed, then the algorithm continues with Step 4, otherwise it skips to Step 6.
4. If the rejection check is passed, then additional histories are run. ADDER estimates the number of history batches necessary to achieve a 95% confidence interval (CI) on k-effective that is equivalent to `uncertainty_fraction * target_interval`. If the `uncertainty_fraction` provided for the `geometry_search` has a value of 1, then the number of batches will yield a k-effective uncertainty that has a 95% CI equivalent to the `target_interval` parameter. The user should note that this is an a priori estimate, and the resultant k-effective uncertainty will not exactly match this estimate. This high-history calculation results in new estimates for the k_{eff} and its statistical standard deviation with the control group at location z_{new}: k_{new} and $\sigma_{k_{new}}$.
5. At this point, the final convergence check is performed, by comparing the calculated 95% confidence range (expressed as $k_{new} \pm 1.960 \times \sigma_{k_{new}}$) to the target interval range (calculated as $k_{target} \pm \Delta k_{target}$), and making sure that it falls entirely within it. A graphic depiction of the early convergence check is provided in Figure 7.1.
    a. If the ranges fall within one another, then the search is converged: z_{new} is the control group location for which the search converges and *the algorithm exits*.
    b. Otherwise, if the convergence check does not pass, then *the algorithm proceeds with Step 6.*
6. Three different scenarios can happen depending on the iteration number:
    a. If the iteration number is equal to the maximum number of allowed iterations, as indicated for the specific `geometry_search` operation, then z_{new} is taken as the converged value from the search. This value can be set by the user (`max_iterations`) and defaults to 30. *The algorithm exits.*
    b. If this is the first iteration, then location z_{old} is set to the value of location z_{new}. The value k_{old} is set equal to the value of k_{new} if the high-history neutronics calculation

was performed; otherwise, if the early rejection check in Step 4 failed, then k_{old} is set equal to the value of k_{new}^{*} from the low-history neutronics calculation. *The algorithm then restarts from Step 1.*

c. If the iteration is not equal to the maximum number of iterations nor is it the first iteration, then *the algorithm proceeds to Step 7*.

7. Location $z_{current}$ is set to the value of location z_{new}. The value $k_{current}$ is set equal to the value of k_{new} if the high-history neutronics calculation was performed; otherwise, if the early rejection check in Step 4 failed, then $k_{current}$ is set equal to the value of k_{new}^{*} from the low-history neutronics calculation.
8. A new guess location, z_{new}, is calculated via linear interpolation, using the Regula Falsi method, as follows:

$$z_{new} = z_{current} + \frac{k_{target} - k_{current}}{k_{current} - k_{old}} \cdot (z_{current} - z_{old}) \tag{20}$$

9. z_{new} is checked against the provided search range bounds:
   a. If z_{new} is larger than z_{max}, then z_{new} is set to z_{max}.
   b. If z_{new} is smaller than z_{min}, then z_{new} is set to z_{min}.
10. The z_{new} value is checked against $z_{current}$.
    a. If the two values are identical, that the target k_{eff} can't be found within the specified search range and *ADDER exits with an error*.
    b. Otherwise, initialize next iteration by setting z_{old} equal to $z_{current}$ and k_{old} equal to $k_{current}$. *The algorithm restarts from Step 1.*

A flowchart of the entire algorithm is provided in Figure 7.2.

In the algorithm above, the user has parameters they can tweak to alter the search process. Specifically, the user can alter the `target_interval`, `min_active_batches`, and `uncertainty_fraction` parameters in order to modify and potentially optimize the search process. Each of these is described below:

- The `target_interval` parameter sets the desired range of the result. Naturally, the larger this range, the more easily the desired `k_target +/- target_interval` is found, and thus fewer iterations will be required. Additionally, if this `target_interval` is increased, then the number of batches that must be simulated will decrease. The user must balance this runtime improvement with the engineering needs of the reactor being analyzed when selecting this parameter.
- The `min_active_batches` parameter is the number of batches to use when performing the first trial calculation. Ideally, this should be as small as possible to obtain a normally distributed (by batch) estimate of k-effective, but large enough so the uncertainty on the differential worth (step 3.b) is still useful. Even if this differential worth uncertainty is large, however, the solution will still converge; it just may require additional iterations to do so.
- The `uncertainty_fraction` parameter represents the fraction of the solution's `target_interval` that should be used to estimate the number of batches. If this has a value of 0.5, then the 95% CI of k-effective from MCNP will be half as large as the target interval. The smaller this value, the more batches are required; however, it makes it more likely that a valid solution is found. Another way to think of this is that `uncertainty_fraction` sets the size of a ball to be thrown through a hole. The size of the hole is set by `target_interval`. The larger the `uncertainty_fraction`, the easier the ball is to throw, but the less likely it is for the thrower to hit the mark.

Note: There is currently no way to perform a `geometry_search` operation to compute the corrector step when using the CE/CM time advancement scheme for a `[[[deplete]]]` operation (see Section 2.1.3, The Predictor and Predictor/Corrector Implementations).

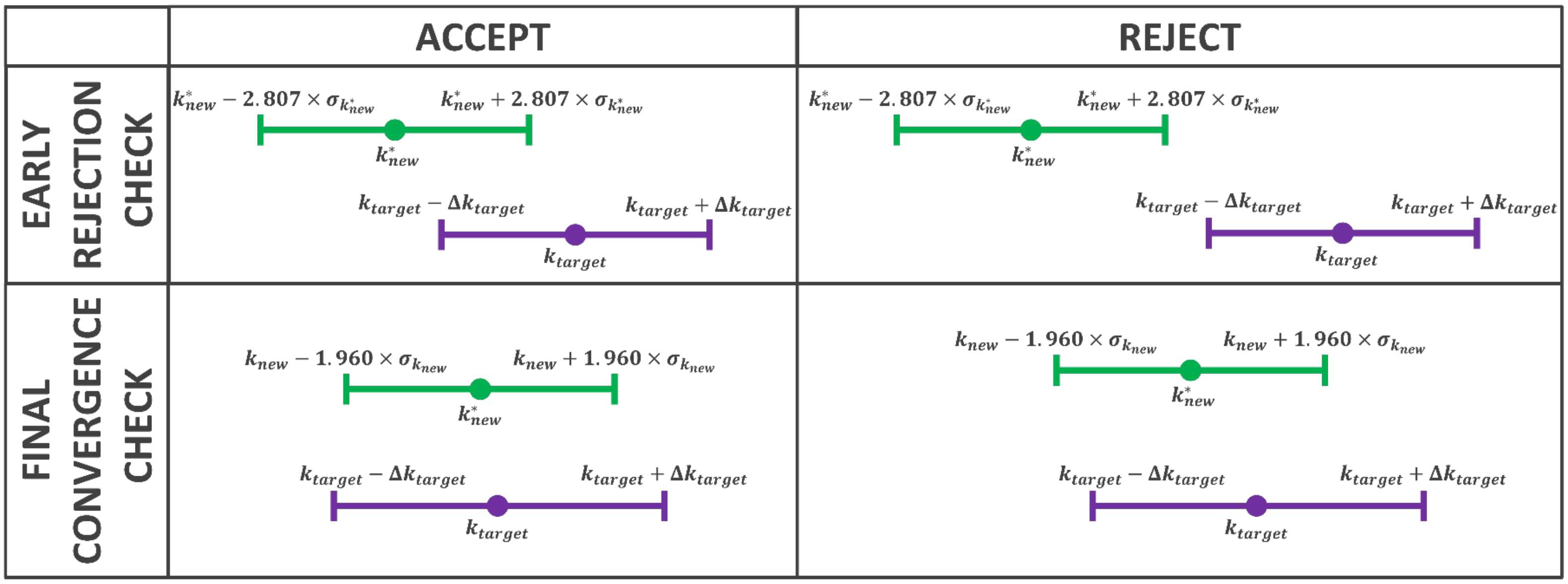
ACCEPT
REJECT
EARLY REJECTION CHECK
FINAL CONVERGENCE CHECK
$k^*_{new} - 2.807 \times \sigma_{k^*_{new}}$
$k^*_{new} + 2.807 \times \sigma_{k^*_{new}}$
k^*_{new}
$k_{target} - \Delta k_{target}$
$k_{target} + \Delta k_{target}$
k_{target}
$k_{new} - 1.960 \times \sigma_{k_{new}}$
$k_{new} + 1.960 \times \sigma_{k_{new}}$

Figure 7.1 Depiction of early rejection check and final convergence check for geometry search operations. Note that the early rejection check only checks for overlap of the two ranges, whereas the final convergence check makes sure that the calculated value and 95% confidence range falls entirely within the target and target tolerance range. As such, the final convergence check is more stringent.

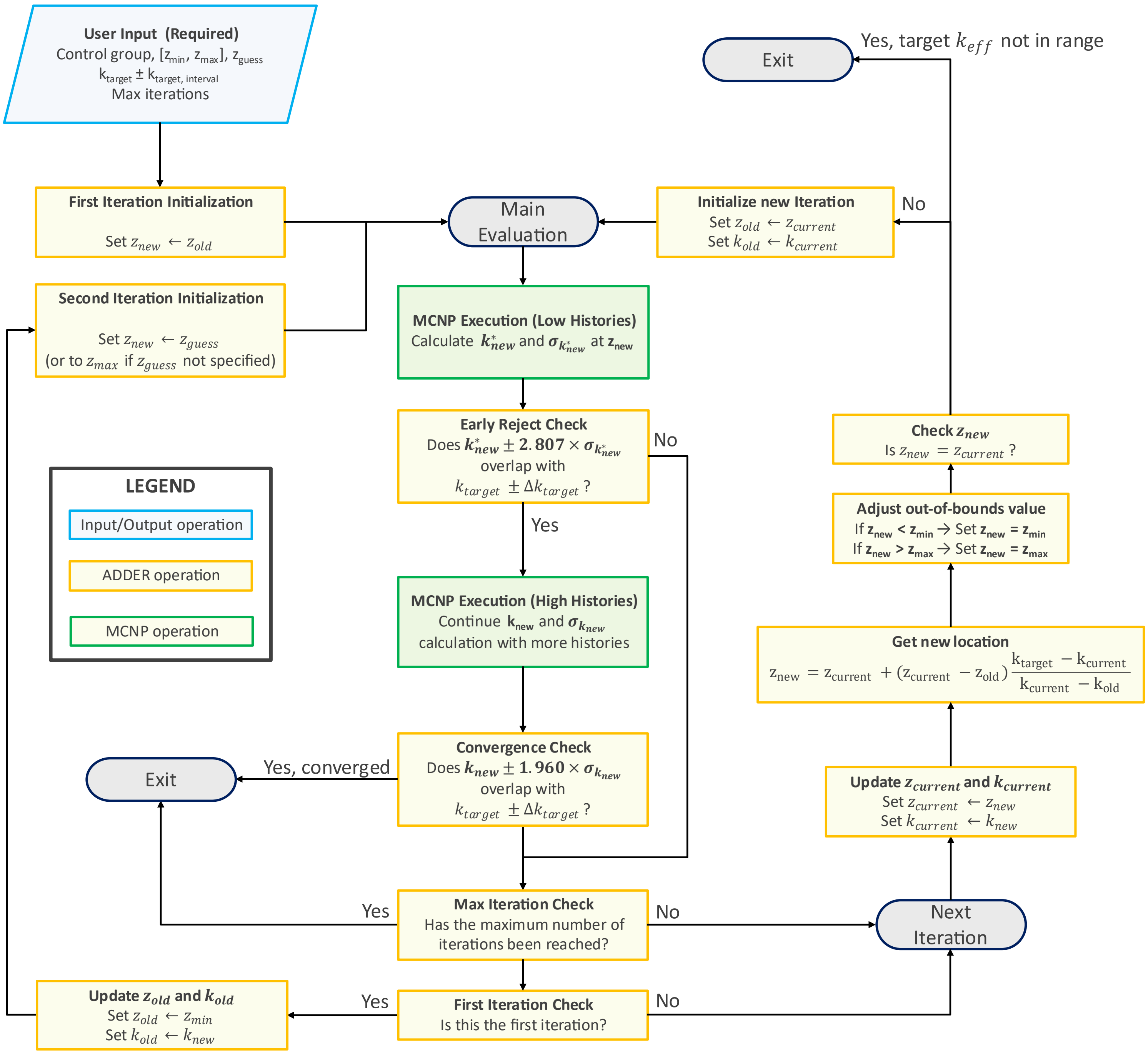
User Input (Required)
Control group, [z_{min}, z_{max}], z_{guess}
$k_{target} \pm k_{target, interval}$
Max iterations
First Iteration Initialization
Set $z_{new} \leftarrow z_{old}$
Second Iteration Initialization
Set $z_{new} \leftarrow z_{guess}$
(or to z_{max} if z_{guess} not specified)
Main Evaluation
Initialize new Iteration
Set $z_{old} \leftarrow z_{current}$
Set $k_{old} \leftarrow k_{current}$
No
Exit
Yes, target k_{eff} not in range
MCNP Execution (Low Histories)
Calculate k^*_{new} and $\sigma_{k^*_{new}}$ at z_{new}
Early Reject Check
Does $k^*_{new} \pm 2.807 \times \sigma_{k^*_{new}}$ overlap with $k_{target} \pm \Delta k_{target}$?
No
Yes
MCNP Execution (High Histories)
Continue k_{new} and $\sigma_{k_{new}}$ calculation with more histories
Convergence Check
Does $k_{new} \pm 1.960 \times \sigma_{k_{new}}$ overlap with $k_{target} \pm \Delta k_{target}$?
Yes, converged
Exit
Max Iteration Check
Has the maximum number of iterations been reached?
Yes
No
Next Iteration
First Iteration Check
Is this the first iteration?
Yes
No
Update z_{old} and k_{old}
Set $z_{old} \leftarrow z_{min}$
Set $k_{old} \leftarrow k_{new}$
Check z_{new}
Is $z_{new} = z_{current}$?
Adjust out-of-bounds value
If $z_{new} < z_{min}$ → Set $z_{new} = z_{min}$
If $z_{new} > z_{max}$ → Set $z_{new} = z_{max}$
Get new location
$z_{new} = z_{current} + (z_{current} - z_{old}) \frac{k_{target} - k_{current}}{k_{current} - k_{old}}$
Update $z_{current}$ and $k_{current}$
Set $z_{current} \leftarrow z_{new}$
Set $k_{current} \leftarrow k_{new}$
LEGEND
Input/Output operation
ADDER operation
MCNP operation

Figure 7.2 Flowchart for geometry search methodology in ADDER

Acknowledgement

The authors of this revised User Guide for Version 1.1.0 of the software would like to thank the original software developer and author of the original User Guide, Dr. Adam Nelson, as well as Dr. Florent Heidet for helping to define the requirements and design of the software. Dr. John Stillman and Dr. Aurelien Bergeron of Argonne National Laboratory are also thanked for their precious review of the ADDER User Guide.

This work was sponsored by the U.S. Department of Energy, Office of Material Management and Minimization in the U.S. National Nuclear Security Administration Office of Defense Nuclear Nonproliferation under Contract DE-AC02-06CH11357.

References

1. C.J. Werner (editor), *MCNP User's Manual - Code Version 6.2*, LA-UR-17-29981, Los Alamos National Laboratory, Los Alamos, USA (2017).

2. X-5 Monte Carlo Team, *MCNP – A General Monte Carlo N-Particle Transport Code, Version 5*, Volume I (LA-UR-03-1987), Volume II (LA-CP-03-0245), Volume III (LA-CP-03-0284), Los Alamos National Laboratory, Los Alamos, USA (2003).

3. A.G. Croff, *A User's Manual for the ORIGEN2 Computer Code*, ORNL/TM-7175, Oak Ridge National Laboratory, Oak Ridge, USA (1980).

4. M. Pusa, "Higher-Order Chebyshev Rational Approximation Method and Application to Burnup Equations", Nucl. Sci. Eng., 182:3, 297-318 (2016).

5. Anaconda Software Distribution, *Anaconda Documentation*, Anaconda Inc., retrieved from https://docs.anaconda.com/ (2025).

6. C.R. Harris, et al., "Array programming with NumPy", Nature 585, 357–362 (2020).

7. P. Virtanen, et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," Nature Methods, 17:3, 261-272 (2020).

8. A. Collette et al., "HDF5 for Python Documentation", https://docs.h5py.org (2020).

9. The HDF Group, HDF5 Support Page, retrieved from https://portal.hdfgroup.org/display/HDF5/HDF5 (2025).

10. M. Foord, N. Larosa, R. Dennis, E. Courtwright, "ConfigObj Documentation", retrieved from https://configobj.readthedocs.io/en/stable/ (2025).

11. C. R. Bates, S. R. Bolding, C. J. Josey, J. A. Kulesza, C. J. Solomon Jr., and A. J. Zukaitis, "The MCNPTools Package: Installation and Use", Los Alamos National Laboratory Tech. Report. LA-UR-22-28935, Los Alamos, NM, USA, August 2022, doi:10.2172/1884737.

12. P. McGuire "pyparsing: A Python parsing library." Retrieved from https://github.com/pyparsing/pyparsing (2025).

13. Krekel et al., *Pytest Repository*, retrieved from https://github.com/pytest-dev/pytest (2025).

14. Pytest-cov team, *Pytest-cov Repository*, retrieved from https://github.com/pytest-dev/pytest-cov (2025).

15. J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95 (2007).

16. J. Meija, et al., "Isotopic Compositions of the Elements 2013 (IUPAC Technical Report)", Pure Appl. Chem., 88:3, 293-306 (2016).

17. W.J. Huang, G. Audi, M. Wang, F.G. Kondev, S. Naimi and X. Xu, "The AME2016 Atomic Mass Evaluation (I)", Chinese Physics C, 41:3 03002 (2017).

18. P. K. Romano, C. J. Josey, A. E. Johnson, J. Liang, "Depletion capabilities in the OpenMC Monte Carlo particle transport code", Annals of Nuclear Energy, Volume 152 (2021), 107989, https://doi.org/10.1016/j.anucene.2020.107989.

19. G. Žerovnik, M. Podvratnik, L. Snoj, "On normalization of fluxes and reaction rates in MCNP criticality calculations", Annals of Nuclear Energy, Volume 63 (2014), Pages 126-128.

20. C. Josey, "Development and analysis of high order neutron transport-depletion coupling algorithms", Doctoral dissertation, Massachusetts Institute of Technology (2017).

21. D.F. Gill, B.R. Nease, and D.P. Griesheimer, "Movable geometry and eigenvalue search capability in the MC21 Monte Carlo Code," Proc. Intl. Conf. on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Sun Valley, USA (2013).