ANL/RTR/TM-21/8 Rev. 1

User Guide to the Advanced Dimensional Depletion for Engineering of Reactors (ADDER) Software

Release Version 1.0.1

Nuclear Science & Engineering Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free at OSTI.GOV (http://www.osti.gov/), a service of the U.S. Dept. of Energy's Office of Scientific and Technical Information.

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000 Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer

ANL/RTR/TM-21/8 Rev. 1

User Guide to the Advanced Dimensional Depletion for Engineering of Reactors (ADDER) Software

Release Version 1.0.1

prepared by
K. Anderson, V. Mascolino, and A. G. Nelson

Nuclear Science & Engineering Division, Argonne National Laboratory

April 2022

(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 v1.0.1. 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

(This page left intentionally blank)

1 A Beginner's Guide to ADDER

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 (ANL) and, eventually, to experts outside of ANL. This report is the user guide for the software release referred to as ADDER v1.0.1.

This software is fundamentally an interface between the user, external neutron diffusion or transport theory solvers, 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.

The initial application of ADDER will be to utilize the MCNP6 [1] (or MCNP5 [2]) software for neutron transport and either ORIGEN2 [3] for depletion and decay functionality or an internal Chebyshev rational approximation method (CRAM [4]) solver.

1.1 Installing ADDER

ADDER is written in the Python 3 programming language. It therefore requires that a Python 3.7 (or greater) 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.

Optional

pytest The pytest [11] framework is used for unit testing the ADDER code.

pytest-cov The pytest-cov [12] package is a plugin to pytest which reports code coverage of the ADDER test suite.

Matplotlib Matplotlib [13] is used to plot certain results for testing purposes.

These prerequisites 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.

The installation of ADDER is trivially 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 python 3 instead of python, then pip in the above installation steps should be replaced with pip3.

1.2 Executing ADDER

ADDER can be executed in whichever 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 Section2, 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 4, 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 most costly 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 4, 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 [14].

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 [15] 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/mass.16` file directly.

---

2 Input File Format

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. This section will discuss the input interactions the user should expect with ADDER.

Before defining the commands utilized in the input file format, some background is necessary. 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.

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.

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.

Finally, to help with understanding the name's ADDER has assigned to materials or universes, 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.

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). Finally, if necessary, multi-line strings can be denoted utilizing triple quotes (""") similar to any other Python program.

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.

The following sections of this document provide details on the sections, subsections, and their parameters. If a particular section or variable is optional, it will be denoted as such and the default value provided.

Commented example input files can be found in the examples/ sub-folder of the ADDER distribution/repository.

2.1 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:

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

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

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

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

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

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:

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

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. Increasing this number increases parallel efficiency at the expense of additional memory usage. This defaults to a value of 1000 and has no effect if num_depletion_threads is 1.

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 [16]. This value defaults to a value of the more accurate but costly “cecm”. Finally, this is a global default which can be over-ridden 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, 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`.

2.2 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.

2.2.1 [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.

[[[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.

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 it will default to a density of `1.0 atoms/b-cm` which is likely not intended.

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 affect if the `use_depletion_library_xs` option in the `[metadata]` section is `True`. 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.

[[[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 it will default to a density of `1.0 atoms/b-cm` which is likely not intended.

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 affect if the `use_depletion_library_xs` option in the `[metadata]` section is `True`. 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.

[[[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 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 it will default to a density of `1.0 atoms/b-cm` which is likely not intended.

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 affect if the `use_depletion_library_xs` option in the `[metadata]` section is `True`. 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.

2.2.2 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.

2.2.3 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 storage to supply).

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.

2.3 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.

2.3.1 [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.

[[[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`.

[[[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.

[[[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).

2.3.2 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.

2.3.3 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 storage" (a universe that is taken out of core to be introduced into the reactor later). However, the user may wish to reassign that universe from storage to being a supply universe (a universe that copies of it can be reintroduced later as if it was from a factory or other supply chain). 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 storage to supply).

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

2.4 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`.

2.5 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:

2.5.1 Case

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.

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 10 billion histories.

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

---

lower_left [optional, list of 3 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 3 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 3 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.

---

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.

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.

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, it is merely provided as a way to provide the user with reactivity (for example) for a particular state. Note, however, that to obtain the end-of-timestep material inventories in the HDF5 file, this must be set to `True`.

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.

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's 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.

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.

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. 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 1st 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.

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.

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.

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.

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. 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. 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. degrees.

matrix [optional if group_name is not present, and roll, pitch, or yaw is not provided, list of 9 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.

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).

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. 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, 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.

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.

---

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.

list

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

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

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. [17].

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 2 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.

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.

initial_guess [optional, float] This parameter is an optional initial guess for the critical position. 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.

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`.

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.

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.

2.6 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.

2.6.1 [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.

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.

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.

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.

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.

---

3 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.

3.1 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-recoverables, 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.

3.2 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; 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.
- **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.
- **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** (*char[]*) – Whether or not the material is in core (in_core), in storage (storage), or is supply (supply)
- **num_copies** (*int*) – The number of copies of supply materials that have been brought in to the core; if this material is not a supply, then this will be 0.
- **density** (*double[]*) – The density of this material.
- **volume** (*double[]*) – The volume of this material.
- **flux** (*double[num_groups]*) – The flux of this material in each energy group.
- **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.

4 Depletion Data Library

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

4.1 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.

4.1.1 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.

4.2 Depletion Library Format

The depletion library HDF5 file format is specified as is provided below. Each attribute and dataset includes 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.

Datasets

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

- **num_groups** (*int*) – The number of groups contained by this cross section.

/<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).`

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.

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

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

This group provides the neutron-induced fission yield data for the isotope <isotope_name>. This block is optional and only exists if neutron fission yield data is available for this isotope. 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.

5 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.

5.1 MCNP

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

5.1.1 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.

5.1.2 User Tallies

User tallies are tallies present in the MCNP input file when it is passed to ADDER. ADDER will include any such tallies in the non-intermediate computations that it performs (e.g., the corrector step of a predictor-corrector depletion) so that results are included in the relevant outputs.

ADDER does not process these user tallies for correctness or consistency with the model. These tallies may become inconsistent as a result of fuel management operations and potentially lead to an MCNP error.

Therefore, the user should consider which tallies should be included as User Tallies and ensure that they are not impacted by the fuel management operations. As such, in models with fuel management, these tallies are best used for global or mesh-based tally results.

5.1.3 Default 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.

5.1.4 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.

5.1.5 Critical Geometry Search Specifics

This method utilizes the Regula Falsi search method with rejection and convergence as described by Gill, et al. [17]. 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 process is summarized as follows:

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 2 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.

In this process, 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.

Acknowledgement

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/ (2020).
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 (2020).
10. M. Foord, N. Larosa, R. Dennis, E. Courtwright, "ConfigObj Documentation", retrieved from https://configobj.readthedocs.io/en/stable/ (2020).
11. Krekel et al., *Pytest Repository*, retrieved from https://github.com/pytest-dev/pytest (2020).
12. Pytest-cov team, *Pytest-cov Repository*, retrieved from https://github.com/pytest-dev/pytest-cov (2020).
13. J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95 (2007).
14. J. Meija, et al., "Isotopic Compositions of the Elements 2013 (IUPAC Technical Report)", Pure Appl. Chem., 88:3, 293-306 (2016).
15. 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).
16. C. Josey, "Development and analysis of high order neutron transport-depletion coupling algorithms", Doctoral dissertation, Massachusetts Institute of Technology (2017).
17. 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).

Nuclear Science & Engineering Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC