INL/EXT-15-37337

Rattlesnake: User Manual

Yaqi Wang, Sebastian Schunert,
Benjamin A. Baker

December 2015

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-15-37337

Rattlesnake: User Manual

Yaqi Wang, Sebastian Schunert, Benjamin A. Baker

December 2015

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Assistant Secretary for ____, OR Office of ____
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Rattlesnake: User Manual

Yaqi Wang, Sebastian Schunert, Benjamin A. Baker
(and whoever has made significant contributions on this manual)

March 9, 2016

Contents

1 Introduction

Rattlesnake is the radiation transport application built with MOOSE for modern multiphysics simulations. Rattlesnake uses finite element methods (FEM) to solve steady-state, transient and k-eigenvalue problems for the multigroup transport equations, the linear Boltzmann equation discretized with the multigroup approximation for the energy (or frequency) independent variable.

FEM is a numerical technique for finding approximations to partial differential equations (PDE) like the transport equation. To use FEM, the geometry is first meshed with an unstructured grid composed of smaller parts or finite elements. Then the solution is approximated with linear combinations of basis functions which have local supports on these elements. The number of basis functions is the number of unknowns in the solution. The variational form of the PDE can then be converted to an algebraic equation that can be numerically solved. The advantages of FEM include an accurate representation of complex geometries, capture of local effects, and etc. The number of shape functions on the same element support is determined by the local expansion of the polynomial order. It is common in Rattlesnake to use the Lagrangian shape functions to solve for variables that are continuous such as primal variables and monomials to solve for discontinuous variables.

The key architectural components within Rattlesnake are:

1. Particle type - The particle refers to the phenomena to be modeled. This is either neutrons or thermal radiation (i.e. $\sim$0.1-1000μm wavelength) currently. There is also an option "common" that uses the common portions of all radiation transport equations.
2. Equation type - Based on the particle, the type of equation can either be the primal equation with the angular flux as the variable or the adjoint equation with the adjoint or importance flux as the variable.
3. Equation problem type - k-eigenvalue, steady-state source and transient. The equation that is solved also depends on the problem type. For instance, in a k-eigenvalue problem the source is balanced to the loss term by 1/k. The steady-state source problem solves for the source term. The transient equation contains all a time derivative term.
4. Scheme for solve - The schemes are explained below. The particle type, equation type and equation problem type are capabilities within the scheme type.
5. Multiscale - Multiscale is a capability that non-uniform scales (or homogenization with corresponding discretization schemes) can be applied on the solution domain to enable first-principle simulation of a real-size problem without iterations on subdomain interface variables. Regions of different levels of homogenization can be treated most efficiently with the multiscale approach. It can be used to avoid sometimes inaccurate pre-homogenization of high-resolution regions.

Some schemes do not have all the possible options available at this point in time. Rattlesnake is in the development process and capabilities will be added in the future. The capability as of this writing for the schemes are summarized in Table 1.

Table 1 The capability of schemes.

Scheme	Mathematical adjoint	Neutron	Thermal radiation	Transient	Multiscale
CFEM-Diffusion	Y	Y	Y	Y	Y
DFEM-Diffusion	N	Y	N	Y	N
SAAF-CFEM-SN	Y	Y	Y	Y	Y
SAAF-CFEM-PN	N	Y	N	Y	Y
LS-CFEM-SN	N	Y	N	Y	N
LS-CFEM-PN	N	N	N	N	N
DFEM-SN	Y	Y	N	Y	N
DFEM-PN	N	N	Y	N	N

When setting up a Rattlesnake input file, the scheme choice is of utmost importance. The schemes within Rattlesnake are listed below along with characteristics and additional capabilities:

- SAAF-CFEM-SN: It stands for SAAF (self-adjoint angular flux)[1] formulation with CFEM (continuous finite element methods) and SN (discrete ordinates methods). SAAF is a 2^{nd} order formulation, which is a transformation of the Boltzmann transport equation or 1^{st} order formulation. CFEM is a method for solving the flux and provides a continuous flux solution. Built into the CFEM is the treatment for the variables: time (t) and space ($\vec{r}$). SN is a method for treating the angular variable ($\vec{\Omega}$). Energy (E) is accounted for by the multigroup approximation. The computing effort is increased about linearly with the increased number of streaming directions. Thus, SN is typically more suitable for heterogeneous calculations, where higher angular resolution is desired, than the calculations with the significant spatial homogenization. This is true for all SN schemes.

  Characteristics

  - Global particle conservation. Global particle conservation guarantees that the sum between leakage, source and absorption will be conserved, however conservation does not hold for each individually also known as element-wise conservation. Global conservation is important for obtaining accurate eigenvalues in eigenvalue problems.
  - The original SAAF formulation involves an inverse total cross-section ($1/\Sigma_t$) term which is problematic for voids or near voids where Σ_t is zero or close to zero. The SAAF-CFEM-SN scheme has a treatment to overcome this issue.
  - The Lagrangian shape function for finite element solves is supported up to a polynomial order two, which is imposed by MOOSE (libMesh). Other shape function families should have a similar maximum order. The disadvantage to increasing the polynomial order is a dramatic increase in computing cost but increases the solution accuracy. This is true for all CFEM schemes.

  Optional Built-in capabilities

  - The angular acceleration scheme, nonlinear diffusion acceleration (NDA) is available. NDA can only be implemented when SN is used for the angle characterization. The acceleration is accomplished by using picard iterations between the transport-corrected low-order (ie. diffusion) solves and the high-order (ie. transport) updates.

- SAAF-CFEM-PN: It differs from SAAF-CFEM-SN in the angular discretization ($\vec{\Omega}$) by using PN (spherical harmonics expansion method).

  Characteristics

  - Global particle conservation. Good for eigenvalue problems.
  - There is no near-void or void treatment for SAAF-CFEM-PN at this moment.
  - Computing cost increases fast with increasing PN order. Typically, low-order PN calculations are performed, which consequently restricts the application to the problems with significant homogenization of the materials and cross-sections, where the transport effect is not as strong. It is preferred in low-order PN cases over SAAF-CFEM-SN because the same numbers of unknowns typically renders smaller discretization errors. This is true for all PN schemes.

  Optional Built-in capabilities

  - We currently do not provide any angular acceleration.

- LS-CFEM-SN: It stands for LS (least-square) formulation with CFEM and SN. LS is also a 2^{nd} order formulation that is derived from the Boltzmann transport equation.

  Characteristics

  - The LS formulation does not include an inverse total cross-section ($1/\Sigma_t$) term, making it naturally work for void or near void regions.

[1] Do no confuse SAAF with the adjoint equation. SAAF is a scheme for solving equations and can be applied to any equation type.

- LS does not have global conservation.

  Optional Built-in capabilities

- The streaming plus collision operator is SPD (symmetric positive definite), which makes it suitable for CG (conjugate gradient) method. The CG generally improves the cpu time and memory usage.
- NDA is available. The low-order diffusion scheme can be conservative, which improves the accuracy of eigenvalue calculations.

- LS-CFEM-PN: It differs from LS-CFEM-SN by PN angular discretization ($\vec{\Omega}$).

  Characteristics

  - The LS formulation does not include an inverse total cross-section ($1/\Sigma_t$) term, making it naturally work for void or near void regions.
  - LS-CFEM-PN does not have global conservation.

- DFEM-SN: It stands for DFEM (discontinuous finite element methods) and SN. The DFEM is directly discretizing the Boltzmann transport or 1^{st} order equation.

  Characteristics

  - It does not need void treatment.
  - DFEM has no order limitation on the Lagrangian shape function, unlike the CFEM schemes which are capped at 2^{nd} order. Also with DFEM, increasing the mesh refinement and/or the shape function order increases the number of unknowns which scales almost linearly in computing cost. The spatial convergence is better for problems with solution singularity when compared to CFEM schemes. DFEM is local and global conservative. These are true for all DFEM schemes.
  - The solver is based on a mesh sweeper which does not impose limitations on the mesh quality.
  - It is matrix-free which lowers the memory usage.
  - The down side is that we do not have a good sweeper supporting parallelization with domain decomposition at this moment.

  Optional Built-in capabilities

  - Supports flexible NDA in the sense that a diffusion accelerator mesh can be coarser than the transport mesh and the diffusion accelerator shape function polynomial can be different than the transport shape function.

- DFEM-PN: It differs from DFEM-SN by PN angular discretization ($\vec{\Omega}$). It has the advantages and disadvantages owned by DFEM and PN.

  Characteristics

  - It does not need void treatment.

- CFEM-Diffuison: Diffusion calculation with CFEM.

  Characteristics

  - Diffusion approximation
  - Computing cost is much smaller than transport schemes.
  - Relies on good diffusion coefficients.

- DFEM-Diffuison: Diffusion calculation with DFEM.

  Characteristics

  - Diffusion approximation
  - Computing cost is much smaller than transport schemes.
  - Relies on good diffusion coefficients.

- Could be a better choice than the CFEM-Diffusion when a high order polynomial is desired for the shape function to increase the accuracy of the solve.
- The number of unknowns is higher for the shape functions to achieve the same level of accuracy with the same shape function order as CFEM-Diffusion.

It is recommended to use the algebraic multigrid method (AMG) in the executioner block for all the schemes except for the DFEM-SN scheme. As a cautionary note, AMG typically requires the mesh to not contain elements whose aspect ratio, or perimeter to volume ratio, is very high. There is an added burden on the user to be vigilant of the mesh quality. AMG is a current area of research with several unknowns for applicability.

To summarize the scheme options, one can choose between SAAF, LS, the original transport formulation and diffusion, CFEM or DFEM methods for solving and SN or PN angular discretization. The SAAF introduces a problem for void or near void regions but can be solved using a special treatment. As of this writing the special treatment has only been developed for the SN case and is not availabe for PN. SAAF has global particle conservation making it a good choice for k-eigenvalue calculations. The LS method does not have global particle conservation but can be used for k-eigenvalue calculations with the SN scheme and NDA. DFEM methods are not restricted on the order for the shape functions. CFEM methods are restricted to 2^{nd} order for Lagrangian shape functions. The higher the order of the shape function typically produces a more accurate solution in terms of the numbers of unknowns. A trade-off exists between the higher the order of the shape function and the computational efficiency. The original transport formulation (ie. DFEM-SN) avoids the problems associated with SAAF and LS. However, at the present moment DFEM-SN does not have a sweeper with parallel domain decomposition. The user should select schemes based on the particular application.

Multiscale transport capability [], along with different schemes can be applied to mesh subdomains simultaneously. This can reduce the computing cost and avoid possible difficult pre-homogenization in certain subdomains. A separate toolkit, YAKXS [1], for multigroup cross section management was developed to support Rattlesnake calculations with feedback both from changes in the field variables, such as fuel temperature, coolant density, etc., and are accessed from importing files with the field variables into the isotope inventory.

Now, for a brief introduction to the MOOSE/Rattlesnake syntax. Rattlesnake takes text-based input files, which are parsed by GetPot [2]. Advantages of using the GetPot format to manage the inputs are:

1. All inputs are naturally managed with a *tree* structure;
2. The input order is irrelevant in a sense that the blocks on the same level can be arbitrarily ordered;
3. Comments can be added anywhere with a leading pound sign #.

A sample input is,

```
#
# The Mesh block is a MOOSE block and is used to define the physical geometry, individual mesh zones (sub-domains)
# and side-sets (used to specify where boundary conditions are located).
# The mesh can also be imported from files for complex geometries.
#
[Mesh]
 type = GeneratedMesh
 dim = 2
 xmin = 0
 xmax = 60
 ymin = 0
 ymax = 60
 elem_type = QUAD4
 nx = 8
 ny = 8
 uniform_refine = 0
[]
```

```
#
# The TransportSystems block is the main block for Rattlesnake and will be covered in great detail.
# TransportSystems performs several built in actions to setup the Variables and Kernels blocks for transport.
# The Variables and Kernels blocks are MOOSE blocks and may still be used for other physics.
#
[TransportSystems]
 particle = neutron
 equation_type = eigenvalue
 G = 1
 ReflectingBoundary = 'right top'
 VacuumBoundary = 'left bottom'
 [./sn]
   scheme = SAAF-CFEM-SN
   family = LAGRANGE
   order = FIRST
   AQorder = 4
   AQtype = Level-Symmetric
   fission_source_as_material = true
   hide_angular_flux = true
 [../]
[]
#
# The Materials block is a general MOOSE block and is generally used to setup
# the material properties and link them to certain mesh zones (sub-domains).
#    Certain sub-blocks, in the Materials block, are used by Rattlesnake to define cross-sections
#    and setup field variables such as temperature dependent cross-sections.
#
[Materials]
  [./nm]
    type = ConstantNeutronicsMaterial
    block = 0
    sigma_t = 1.0
    sigma_s = 0.99
    fissile = true
    nu_sigma_f = 0.01
    chi = 1.0
  [../]
[]
#
# The Postprocessors block is a MOOSE block and is used to calculate derived quantities from primal variables
#   This example is taking the integral of the scalar flux for energy group 0.
#   The integral is for the entire mesh since no particular sub-domain is mentioned
#
[Postprocessors]
  [./fluxintegral]
    type = ElementIntegralVariablePostprocessor
    variable = flux_moment_g0_L0_M0
    execute_on = linear
  [../]
[]
#
# The Executioner is a MOOSE block. As the name implies it controls the execution of the solve. Options include
# solver types, solution tolerances, type of calculation to perform and many more options.
#      This particular example is performing a non-linear eigenvalue calculation.
#      Notice eigenvalue was also specified in the TransportSystems block above. Based on the scheme
#      (i.e. SAAF-CFEM-SN, etc. found in the TransportSystems block) options may become applicable
#      in the Executioner such as AMG.
#
[Executioner]
  type = NonlinearEigen
```

```
  free_power_iterations = 4
  source_abs_tol = 1e-6
  output_before_normalization = false
  output_after_power_iterations = false

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
[]
#
# The Outputs block is a MOOSE block and is used to determine when and files types to save the data.
#   "file_base =" specifies the file name without the extension.
#   Be mindful that choosing which data to print to the screen is controlled in other blocks. By default, all
#   variables, auxiliary variables and postprocessors are recorded.
#
[Outputs]
  file_base = out
  exodus = true
[]
```

This input deck is copied from the file "yak/tests/actions/neutron_saaf_sn_cfem/neutron_saaf_sn.i", with comments added for clarification.

From the example we can clearly see the tree structure of the inputs. The names of the zero-level blocks are embraced with bracket parenthesis [block name]. The zero-level blocks are concluded with empty parenthesis `[]`. The zero-level block names must match the names in Rattlesnake and MOOSE. All sub-blocks are given with a block name with leading `[./sub-block name]` embraced with bracket parenthesis and are concluded with `[../]`. The sub-block names are be chosen by the user, unless explicitly mentioned. Sub-blocks on the same level must be unique. To specify which sub-blocks to use within Rattlesnake/MOOSE use the "type =" then the sub-block name as listed in Rattlesnake/MOOSE. For instance, in the example above the user used the Rattlesnake sub-block "ConstantNeutronicsMaterial". First, the user gave the sub-block in the input file a new name called "[./nm]" and used "type = ConstantNeutronicsMaterial" to have access to all the options available within ConstantNeutronicsMaterial sub-block. The options set in this sub-block are only applicable to [./nm]. More sub-blocks could have been define using the same ConstantNeutronicsMaterial Rattlesnake sub-block but with different settings. We will term this tree structure as the *Rattlesnake input syntax* from now on. Currently all valid MOOSE syntax are also valid for Rattlesnake.

Following are some useful commands that are commonly used. However, it is required that Rattlesnake be compiled before performing these commands. To learn about compiling see Sec. 2.

All supported syntax can be seen by

```
./rattlesnake-opt --syntax
```

GetPot treats values of the parameters as strings. If the string contains spaces, the string must be quoted with the single quote sign `'`. All parameters of all syntax can be dumped using:

```
./rattlesnake-opt --dump
```

---

where *rattlesnake-opt* is the Rattlesnake executable generated in the optimized mode (opt). This dump takes a search string to filter the entire syntax, for instance, the following

---

```
./rattlesnake-opt --dump Outputs
```

dumps parameters for the *Outputs* block exclusively. The parameters have to be on the leaf level of the syntax. We are currently debating if we want to enable parameters on the parenting levels so that all the child levels can share those parameters when the parameters on the leaf level are not given. Invalid parameters can be put into input files. Rattlesnake will print a warning message on the screen for those invalid parameters. Users can use `-e` command line option to completely disable invalid parameters. Duplicated parameters are allowed by default to allow users overriding previously provided parameters. However this typically means there is a typo. Users can use `-o` command line option to disallow duplicated parameters.

Users can run Rattlesnake with,

```
./rattlesnake-opt -i <InputFile>
```

InputFile represents the file name of the input file. Command line option `--n-threads=<n>` enables multi-threading with *n* number of threads. Rattlesnake executable can also be invoked with MPI

```
mpirun -n <n> ./rattlesnake-opt -i <InputFile>
```

where *n* is the number of processors. A complete list of command line options can be seen with

```
./rattlesnake-opt
```

2 Getting started

2.1 Obtain access

Rattlesnake is currently an export-controlled software. (More to be added.)

2.2 Install and test

1. The first step depends on where you will install Rattlesnake

   - Local desktop:
     Follow the link in the MOOSE wiki page of your platform to install the redistributable packages. Then set up ssh tunnel with your password by

     ```
     >ssh -D 5555 hpclogin.inl.gov
     ```

     and change the proxy setting for socks with *localhost* as the proxy with port 5555. You will need the remote access to HPC to accomplish this step.
   - home directory on INL HPC:
     Load the development module.

     ```
     >module load moose-dev-gcc
     ```

2. Optionally create a fork of the main Rattlesnake repository at hpcgitlab.inl.gov.

3. Create a directory where you want to store Rattlesnake, for example, ∼/projects. Change directory into it and use git to clone the code.

   ```
   >mkdir ~/projects
   >cd ~/projects
   >git clone git@hpcgitlab.inl.gov:USERNAME/rattlesnake.git
   ```

   USERNAME is *idaholab* if you skipped the second step otherwise your high performance computing (HPC) user account.

4. Grab and initialize the submodule and build libMesh.

   ```
   >cd ~/projects/rattlesnake
   >git submodule --update init
   >cd ~/projects/rattlesnake/moose/scripts; ./update_and_rebuild_libmesh
   ```

5. Build Rattlesnake.

   ```
   >cd ~/projects/rattlesnake
   >make -j8
   ```

   We add -j8 to compile using 8 processors. You can use whatever number you have available. The executable *rattlesnake-opt* will be generated after a successful build. Extension *opt* means that this executable is an optimized version. Users can also build two other versions, *dbg* and *oprof* for debugging or profiling purpose by

```
>METHOD=dbg make -j8
```

or

```
>METHOD=oprof make -j8
```

These two builds are optional.

6. Run the regression tests to verify that the build was correct.

```
>./run_tests -j4
```

Here we use 4 processors with -j4 to make the tests run faster.

With the *rattlesnake-opt* built. It is now possible to run input files and other commands listed at the end of Sec. 1.

2.3 Use tutorials as templates

Sec. 3 provides several tutorial inputs. They can be viewed independently. They demonstrate the Rattlesnake capabilities and can be used as templates for users to create their own inputs. Keywords of these tutorials are listed:

- Kobayashi benchmark: One-group source problem; Regular Cartesian geometry; SN scheme; The source iteration.
- Benchmark 16A1-1 (Eigenvalue Problem): One-dimension; Two-group eigenvalue problem; PJFNK eigen solver.
- Benchmark 16A1-1 (Transient Problem): One-dimension; two-group transient problem; Initial condition from the eigenvalue calculation; Time integration and stepping with PJFNK.
- Takeda benchmark Mode 4: Four-group eigenvalue problem; Hexagonal geometry; Rotational periodic boundary condition.
- LRA benchmark (14-A1): Two-group transient problem; Adiabatic temperature model; Flux map; Custom neutronics materials.
- LRA PKE: PKE (point kinetics equation) for the LRA benchmark; PKE parameter dumping with spatial kinetics; Reactivity function fitting.
- C5G7-2D with SAAF-SN-CFEM NDA: LWR mesh generation; INSTANT XML cross section format; Flux map; NDA with the SAAF-SN-CFEM scheme; Transport update with AMG.
- C5G7-2D using First Order NDA solver: LWR mesh generation; INSTANT XML cross section format; Flexible NDA with the first-order SN scheme; Transport update with sweeper.
- Coupled reactor: multi-region calculation; coupled reactor.
- A problem demonstrates YAKXS: Criticality search; YAKXS XML cross section format.
- A thermal radiation benchmark: Thermal radiation transport.

2.4 Instruction on using this manual

Sec. 4 *TransportSystems* to Sec. 12 *Outputs* catalog the Rattlesnake input parameters.

The format structure for each of the input parameters will follow a basic pattern where a simple description of the option is provided, followed by the possible choices for the parameter, a default value and branch structure syntax. To illustrate, Sec. 4.1 is the particle option for the transport system and the catalog entry is shown below:

Description: Particle type of the transport system

Data type: Enumeration (/common/neutron/thermal/)

Default value: <required>

Syntax: TransportSystems/particle

The particle option changes the transport equation based on the particle of interest. The possible choices for particle are "common, neutron or thermal", where neutron is for neutrons, thermal is for thermal radiation and common uses the common portions of the transport equation between neutrons and thermal radiation. The syntax shows that the particle option is located within the *TransportSystems* branch.

An example input syntax would look like the following:

```
[TransportSystems]
  # particle set to neutron
  particle = neutron
  ...
[]
```

All parameter names are in italic font. All parameters in blue color are advanced and red are basic. Most parameters have default values or are allow to be empty. Users are not required to set the values of these parameters. If the parameter is empty, Rattlesnake will not activate the functionality represented by that parameter. If a parameters is required, it will be marked as *required* in the *default value* field. The data types are usually either a name, integer or logical. Logical inputs can be 1, 0 or true, false. The types can also be vectors and must be surrounded by single quotes.

The tree structure/block syntax of all the parameters are showed in the *syntax* field. The "$*$" in the syntax means wild-card and is used to represent the name given by the user for the sub-block. For instance, the user defines the sub-block scheme-name in the example code below, while one would go to the SAAF-CFEM-SN scheme to determine the available options.

```
[TransportSystems]
...
...
...
  [./sheme-name]
    scheme = SAAF-CFEM-SN
    family = LAGRANGE
    order = FIRST
    ...
    ...
    ...
  [../]
[]
```

2.5 General Rattlesnake inputs

The following sections will provide a brief description of common input blocks used for MOOSE and Rattlesnake with simple example blocks to show how they operate together. The code presented in the following sections are meant to explain how the input code is organized and will not function alone. To view full examples go to Sec. 3.

As was mentioned in the Introduction (Sec. 1), the Rattlesnake input file is structured in a tree system with branches or blocks beginning with brackets "[ ]" around the name and ending with a bracket. Sub-blocks also begin and end with brackets. The difference between the sub-block and the main block are observed by the `./` before the sub-block name and closing the sub-block with `[../]`. Sub-blocks are also named by the user, while the main blocks must have the same name as given by MOOSE or Rattlesnake.

Below is an example of a main block and two sub-blocks that have the same functionality. You will also notice that there is an option called `block = .` This usage of "block" is referring to the mesh, not the block syntax. Based on the context, the user should be able to distinguish between the two.

```
# The # symbol is a comment
#
# Postprocessors is the main block and gets its name from \MOOSE
[Postprocessors]
  # Flux1 and Flux2 are sub-blocks and are named by the user.
  # Each sub-block must begin with [./   ].
  [./Flux1]
    # Flux1 needs to know what sub-block within \RSN or \MOOSE to use. This is handled by "type = "
    # ElementIntegralVariablePostprocessor is the sub-block in Postprocessors to be used for this example.
    #
    # This particular post-processor calculates the integral of some variable with respect to space.
    # The variable being integrated is the scalar flux (flux_moment) for energy group 0.
    # The block = '1 2 3 4 5 6 7' specifies which regions within the mesh to apply the integral.
    #   If no block is specified the integral will apply to the entire mesh.
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g0_L0_M0
  [../]  # Flux1 is closed by the [../]
  #
  # Flux2 is a new sub-block
  # It is using the same sub-block type as Flux1. The difference is that the variable is now
  # the scalar flux (flux_moment) for energy group 1 instead of group 0.
  #
  [./Flux2]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g1_L0_M0
  [../]
  # Flux2 is closed by the [../]
[]
# The main block [Postprocessors] is closed with ending brackets [].
```

This example illustrates why each sub-block needs to be named by the user. By allowing the user to specify the name, the functionality (specified by "type =") can be duplicated multiple times. The sub-blocks within MOOSE or Rattlesnake dump files are given the same name as "type". There are a few exceptions, such as the sub-block for TransportSystems which uses "scheme =" instead of "type =". The names created by the user are also recognized in other parts of the input file.

The most commonly used blocks in MOOSE are shown below along with one for Rattlesnake. Since, these are main level blocks, the names need to be spelled exactly the same, including capitalization.

```
[Mesh]
[Variables]
[Kernels]
[BCs]
[AuxVariables]
[AuxKernels]
[Functions]
[Materials]
[Postprocessors]
[Executioner]
[MultiApps]
[Transfers]
[Outputs]

[TransportSystems] for RattleSnake
```

To view possible options for each of these blocks use

```
./rattlesnake-opt --dump BlockName
```

where BlockName is the block such as ”Mesh”. A file can also be saved with all the options from the dump by using the command

```
./rattlesnake-opt --dump >& filename.txt
```

where filename.txt is the name for the file to be created.

2.5.1 Mesh

[Mesh]

The Mesh block is used to specify the finite elements and define boundaries that are used for boundary conditions in the [BCs] block. A ”block”, as seen in the above example, is used by other parts of the input file to specify certain portions of the mesh. The block regions are defined in the subdomain of the mesh. One should note that the name block is not used in the mesh branch but is used in other parts of the input file to specify which sub-domain is being used. When specifying multiple blocks together it is required to use a vector represented by single quotes (’ ’). For example (block = ’4 7 10’).

Surfaces used for boundary conditions are defined automatically by mesh generators or can be manually created, such as side sets in an exodus file. When using the mesh generator the boundaries are located at the outside boundaries and are given names like ’left right top bottom front back’, however they can also be numbered ’1 2 etc.’.

For complicated geometries a mesh program like cubit might be required to create the finite element structure. For more information on building a mesh refer to Sec. 6.

Below are two example mesh blocks. The first uses a mesh generator to create a 1-D mesh used for the 16A1-1 benchmark. Notice that the `subdomain_id` contains elements 1-7. Other parts of the input file will refer to these sub domains by the ”block = ” option and will use 1-7 to identify the sub-domain to which the input file option applies to.

The second mesh block loads in a file with the mesh. It also doubles the number of mesh points in the file by using the `uniform_refine = 1` option.

```
[Mesh]
 type = CartesianMesh
 dim = 1
 dx = '40 47.374 9 34 9 47.374 40'
 ix = '20 24 5 16 5 24 20'
 # subdomain_id are the regions of the mesh.
 subdomain_id = '1 2 3 4 5 6 7'
 # dx is the width of each subdomain_id
 # ix is the number of mesh intervals to place within each dx.
 # Boundary surfaces 'right left' are automatically created.
 uniform_refine = 0
[]
```

```
[Mesh]
  file = ../../Filename.e
  uniform_refine = 1
[]
```

2.5.2 Variables

[Variables]

The Variables block is used to define the primal variables for the problem. These are the variables that are solved for in the differential equations and need to be spatially dependent.

Suggestions when classifying the variable for the input file:

1. If the problem you are solving does not have space as a variable you might need to use [ScalarVariables] and [ScalarKernels].
2. If the quantity you are after is a derived quantity where the spacial dependance has been integrated out you might need to use a post-processor.
3. If the variable is dependent on space but is not directly used in the main variable equation or depends on the primal variable or is discontinuous, it might need to be defined as an AuxVariable. Reaction rates are typically AuxVariables because of discontinuity in space.
4. The Variables block is only for primal variables that are part of the differential equation, meaning that there must be some differential of the variable in the equation.

In the input the primal variable shape function must be continuous. A common type is Lagrange. MOOSE allows for multiple variables to be coupled with little effort, this is known as multi-physics coupling. These variables can be solved for with various schemes. As a cautionary note, when multiple physics are coupled one might need to use the "scale=" option if the magnitudes of the variables are greatly different. It has been observed in Rattlesnake that scaling was required when coupling between the scalar flux and temperature for one particular application.

Below is an example of a variable being defined. Note that units are up to the user and should workout as long as the user maintains consistency based on the equations.

```
[Variables]
   [./Tfuel]
      order = FIRST
      family = LAGRANGE
```

```
      scaling = 1e-4
      block = '101 102'
        [./InitialCondition]
        type = ConstantIC
        value = 300.0
      [../]
   [../]
[]
# Tfuel was named by the user.
# family: sets the type of shape function to use for the solve over the finite elements in the mesh.
#   Most common family option is Lagrange for primal variables since they need to be continuous.
# order: sets the order of polynomial for the shape function.
# scaling: changes the tightness of the solve (ie. residuals) to other variables
#  scaling is usually not used and is defaulted to 1.
# block: sets the mesh locations to define the Tfuel variable.
# [./InitialCondition] is a sub-block named by the user
# The sub-block uses the ConstantIC functionality
# The Tfuel variable is set to 300 on every element within blocks 101 and 102 on the mesh.
#
# Units are arbitrary in MOOSE
# Users define the units and needs to stay consistent in definitions.
#
```

2.5.3 Kernels

[Kernels]

The Kernels block is where the physics/equations are created. The convention in MOOSE is to define a kernel to represents each part of the equation that can be separated by addition or subtraction operators. In this way, parts of the equation may be included/excluded very easily. Further, the source code is defined so that all kernels are on the left hand side of the equation. Thus, all terms sum to zero for the solve. An example of a simple point kinetics model would be: $\frac{dn(t)}{dt} - \frac{\rho(t)-\beta}{\Lambda} - \sum_{i=1}^{6} C_i(t)\lambda_i = 0$. Where each of the the terms $\frac{dn(t)}{dt}$, $\frac{\rho(t)-\beta}{\Lambda}$, $\sum_{i=1}^{6} C_i(t)\lambda_i$ are kernels. Kernels can be defined to include a grouping of several terms but by convention they are individual terms. The finite element method used in MOOSE is to transform the equation to the weak form. To keep things simple, the main implication here is that if there are any diffusion terms like $(\nabla k \nabla \psi)$ a boundary condition is created and needed in the input file. The equations can also be specified to apply to only certain parts of the mesh by using the blocks option. The kernels are pre-developed by MOOSE or other development teams such as Rattlesnake. The dump file can provide a listing of available kernels. One major disadvantage is that many of the kernels require opening the source C++ files to understand the operations it performs.

Example kernels are shown below in the Boundary Conditions section.

2.5.4 Boundary Conditions

[BCs]

The BCs block is used to specify the type of boundary condition, where the boundary condition is applied and the associated value. The most common boundary conditions are Dirichlet (constant value) and Neumann (constant derivative). The boundary conditions are specified on the mesh by the boundary which is defined in the side sets for an exodus mesh file.

Below is an example of several kernels being setup for the heat equations and a boundary condition.

```
[Kernels]
# Heat equation = -Del*k*Del*T - q + Cp*rho*dT/dt = 0
# Use Adiabatic Equation = Cp*rho*dT/dt -q = 0  (q is the source term)
#
   [./HeatConduction]           # Units [=] J/cm^3-sec = Watt/cm^3
      type = HeatConduction     # Kernel = Del*(thermal_conductivity)*Del*Temp
      variable = Tfuel
      block = '101 102 103 104 105 106'
   [../]

   [./HeatStorage]                          # Heat Storage, units [=] Watt/cm^3
      type = HeatConductionTimeDerivative   # Kernel = cp*rho*dT/dt
      variable =Tfuel
      block = '101 102 103 104 105 106'

   [../]

   [./HeatSource]            # Heat Source, units [=] Watt/cm^3
      type = CoupledForce    # Kernel = -q, where q is another variable
      v = ScaledPowerDensity
      block = '106'   # Heat generation only in the core
      variable = Tfuel
   [../]
[]

[BCs]
# The BC exists because of the diffusion term Del*k*Del*T
# If no BC is given the default is a NeumannBC condition (ie. dT = 0 at the boundary)
# The DirichletBC
   [./TempBC]
      type = DirichletBC
      variable = Tfuel
      boundary = '1 2 3 4 5 6'
      value = 300.0         # Units [=] Kelvin
   [../]
[]
```

Note that each of the kernels for the heat equation involves the Tfuel variable even if temperature was not part of the kernel, such as the HeatSource term. The variable is how MOOSE is able to know which kernels are linked together.

It is possible to define the kernel to apply to only certain parts of the mesh. For instance, the HeatSource term is defined in one region "block = 106" while the diffusion and time derivative terms apply to the sub-domain blocks 101-106. The HeatSource also depends on a user defined auxiliary variable "ScaledPowerDensity" which is defined elsewhere.

Notice that the thermal conductivity (k), specific heat (Cp) and density (rho) parts of the heat equation were not specified here. These are material properties and are specified in the [Materials] block. For this example, the boundary condition was added because of the diffusion term (HeatConduction).

The surfaces for the BC's are created in the mesh. When using a built in mesh generator the surfaces are automatically defined and are given names "left right top bottom front back". For this particular example, boundaries where created in a custom file and were assigned values 1-6 to represent the left, right, top, bottom, front and back surfaces or wherever surfaces were needed for the problem. This example creates a dirichlet BC which assigns a constant value at the boundaries 1-6.

Notice that the user applied comments to keep track of units.

One can easily modify this code to turn off the heat diffusion term to make the problem adiabatic or they can turn off the time dependent term for a steady-state calculation. If the heat diffusion term is turned off there is

no need for the boundary conditions.

2.5.5 AuxVariables and AuxKernels

[AuxVariables] and [AuxKernels]

The auxiliary variables and kernels are quantities that either do not have differentials and thus cannot be primary variables or the user wants to keep these variables from being explicitly coupled to other variables or the variable needs to be discontinuous. An example of discontinuous variables are reaction rates. The reaction rate is discontinuous because the cross-sections instantly change when crossing between materials. The Monomial shape functions are commonly used for discontinuous variables. Just like Variables and Kernels the block setup is similar for AuxVariables and AuxKernels.

```
[AuxVariables]

  [./CR_Boron]
      order = CONSTANT
      family = MONOMIAL
      [./InitialCondition]
        type = ConstantIC
        value = 6.700E-04
      [../]
  [../]

  [./ScaledPowerDensity]
    # ScaledPowerDensity [=] Watts/cm^3
    order = CONSTANT
    family = MONOMIAL
  [../]

  [./IntegralPower]
    # IntegralPower [=] Joules/cm^3
    order = CONSTANT
    family = MONOMIAL
  [../]

[]

[AuxKernels]

  [./ScaledPowerDensity]
    type = ReactionRateAux                    # (Variable is dependent on space)
    variable = ScaledPowerDensity             # Units [=] Watts/cm^3
    # This cross_section*scalar_flux is the UnscaledPowerDensity
    cross_section = kappa_sigma_fission
    scalar_flux = 'sflux_g0 sflux_g1 sflux_g2 sflux_g3
                   sflux_g4 sflux_g5 sflux_g6 sflux_g7
                   sflux_g8 sflux_g9 sflux_g10'
    normal_factor = 910.0                     # Starting Power [=] Watts
    normalization = UnscaledTotalPower        # Uses a post-processor not AuxVariable
    block ='106 107 6006 6007 6008'
    execute_on = linear
    #
    # ScaledPowerDensity gets the value of
    # "normal_factor" divided by the postprocessor value "UnscaledTotalPower".
    # ie. normal_facotor/normalization -> P0/integral(unscaledPowerDensity(vol,t)dV).
    #
    # Flux_g_i is the coupling variable for ScaledPowerDensity that is changing
```

```
    #   in the transient. The postprocessor UnscaledPower no longer changes for
    #   the transient and stays as its inherited value from the SS calc.
    #
  [../]

  [./Powerintegrator]
    # Units [=] Joules/cm^3   (Variable is dependent on space)
    # Integrates the variable ScaledPowerDensity w.r.t. time
    # ie. Integral(PowerDensity(Vol,t)*dt)
    # If the flag "use_as_density" is false then multiply by volume in the integral
    # ie. Integral(PowerDensity(Vol,t)*dV*dt) if "use_as_density" = False
    # No dV term if the flag "use_as_density" = true
    #
    type = VariableTimeIntegrator
    variable = IntegralPower                    # Units [=] Joules/cm^3
    variable_to_integrate = ScaledPowerDensity   # Units [=] W/cm^3
    coefficient = 1.0
    block ='106 107 6006 6007 6008'
    execute_on = linear
    use_as_density = true  # this makes the variable as energy density (J/cm^3)
    # Use_as_density: False multiplies by volume and True does not include it.
  [../]

  [./SetCRBoron]
     type = FunctionAux
     function = BContent
     variable = CR_Boron
     execute_on = timestep_end
   [../]
[]
```

In the above example, there are 3 AuxVariables. The AuxVariables are using the Monomial shape functions which are discontinuous as opposed to the Lagrange family which is continuous. The equations governing the AuxVariables are found in the AuxKernels portion. The AuxKernels may involve other variables or post-processors; for example the ScaledPowerDensity kernel depends on a post-processor (UnscaledTotalPower) and the scalar flux primal variables (`sflux_g0 through sfulx_g10`).

For every kernel, it must be told which variable it applies to, this is set with the "variable =" option. For this example, a diffusion calculation is being performed instead of transport and the variables for the scalar flux are given a different naming scheme then the scalar flux coming from a transport calculation. The naming scheme is `sflux` for diffusion and `flux_moment` for transport.

The `PowerIntegrator` kernel is integrating the auxiliary variable `ScaledPowerDensity` with respect to time, making the `IntegralPower` variable still a function of space. The `SetCRBoron` kernel is implementing a function to govern the value for the `CR_Boron` AuxVariable. The function usage will be explained in the next section. The code here tells the kernel to point to the function `BContent` and assign the variable `CR_Boron` the value coming from the function.

2.5.6 Functions

[Functions]

The Functions block is used to specify functions to be used by variables, materials and post-processors. There are several function types available. View the dump file for more options. Below is an example where three functions are setup.

```
[AuxVariable]
```

```
  [./CR_Boron]
      order = CONSTANT
      family = MONOMIAL
      block = '101 102'
      [./InitialCondition]
        type = ConstantIC
        value = 6.700E-04
      [../]
  [../]
[]

[AuxKernels]
  [./SetCRBoron]
     type = FunctionAux
     function = BContent
     variable = CR_Boron
     execute_on = timestep_end
   [../]
[]

[Functions]
  [./BContent]
     type = PiecewiseLinear
     x = '0.0 0.05 0.65 30.0'
     y = '6.700E-04 6.700E-04 1.3881E-07 1.3881E-07'
  [../]

  [./Set_k]
     type = ParsedFunction
     value = '8.21E-5*t*t-1.943E-04*t'   # Units [=] W/cm-K
     # ParsedFunction to set the value for k (thermal-conductivity) as a function of temperature
     # temp =Tfuel is used by HeatConductionMaterial to use in the function instead of t (time)
  [../]
   [./TotalFlx_function]
   type = ParsedFunction
   value = '(F1 + F2)/Norm'
   vars = 'F1 F2 Norm'
   vals = 'Flux1 Flux2 NormalizationFlux'
 [../]
[]
```

Included before the functions block are blocks for the `AuxVariables` and `AuxKernels`. The codes shows how they all rely on one another. The first function controls the boron concentration. For this example, the `CR_Boron` is the variable and is defined on the mesh at blocks/subdomain numbers 101 and 102. The value given to `CR_Boron` is initially given as 6.7E-4 but changes according to the function `BContent` which is a piecewise linear function where 'x' represents a point in time and "y" is the value for `CR_Boron`. The value for boron will linearly change between times steps 0.05 and 0.65 sec. The kernel `SetCRBoron` tells the variable `CR_Boron` which function to use and when to execute.

The second function is creating a polynomial to describe the thermal conductivity property, which is found in the material block. The `ParsedFunction` develops a polynomial that can depend on space (x,y and z) and time (t). For this particular case the developers had to use t, to represent temperature and a special adaptation was made in the source code for a sub-block within the Materials block to change the variable from time to temperature.

The third function shows how the `ParsedFunction` is normally used. In this example there are three inputs value, vars and vals. The value represents the equation to be built. The vals are the names of the variables/parameters coming from other parts of the input file. For this case the vals are coming from post-processors. The vars are a convenient way to set the variable name in the equation without using the variable name as defined

in other parts of the input file. The vals get the value of the corresponding vals based on order within the vector.

2.5.7 Materials

[Materials]

The Materials block is use to define materials, their properties and mesh blocks that use those properties. Another block called [YAKXSLibraries] is used to load in files with cross-section definitions to be used within the materials block.

An example block for thermal conductivity, specific heat and density is shown below.

```
[Materials]
  [./ThermalProperties]
     type = HeatConductionMaterial
     temp = Tfuel
     # temp tells the function to replace the time variable (t) with the variable it is set equal to (Tfuel)
     # temp is only used if we are using a function of temperature
     # Set_k is the function for thermal-conductivity
     thermal_conductivity_temperature_function = Set_k    # Units [=] W/cm-K
     specific_heat = 5.8    # Units [=] J/g-Kelvin
     block = '101 102 103'
  [../]

  [./density]
     type = Density
     density = 10.3             # Units [=] g/cm^3
     block = '101 102 103'
  [../]
[]
```

In this example the thermal conductivity is a function of temperature and specified in the functions block. The `temp = Tfuel` provided a special adjustment so that the function could take in temperature instead of time. The kernels used in the prior example had hard coded names for thermal conductivity, specific heat and density. The material sub-block had to use the exact same name as those hard-coded names in the kernels. The `HeatConductionMaterial` and `Density` materials sub-block provided these names.

Below is an example of cross-sections being created using the `ConstantNeutronicsMaterial` sub-block for a 2 energy group transport problem with delayed neutrons.

```
[Materials]
  [./Mat1]
    type = ConstantNeutronicsMaterial
    block = '1 7'

    fissile = true
    #
    # x-sections
    #
    nu_sigma_f = '8.3441E-4 3.2776E-4'
    # nu_sigma_f is the fission cross-section for each energy group
    sigma_t = '2.411E-1 4.172E-1'
    # sigma_t is the total cross-section for each energy group
    sigma_s = '2.33644E-1 0.0
               3.598E-3 4.07004E-1'
```

```
    # sigma_s is the scattering matrix: sigma_1->1, 2->1, 1->2, 2->2
    chi = '1 0'
    # chi is the fraction of prompt neutrons that appear in each energy group

    # Delayed properties
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0
                      1 0
                      1 0
                      1 0
                      1 0
                      1 0'

    # delay_sepectrum is the delayed neutron spcectrum for Xd_g_i = Xd_1_1, Xd_2_1 ... Xd_1_6, Xd_2_6
    # It's the spectrum at each energy group and for each delayed group
    # (energy group first followed by delayed group)
    neutron_speed = '540248514.31659 91911764.705882'
    # Units [=] cm/sec
  [../]
 []
```

2.5.8 Postprocessors

[Postprocessors]

The `Postprocessors` block provides the user a location to perform calculations for derived quantities from variables or other post-processors. Post processors may also use functions from the Functions block. They can also be used to catch a value. Some of the common post-processors are: Integrals with respect to space, functions, maximum/minimum value, receiver.

Below shows several post-processors and associated blocks that communicate with it.

```
[Postprocessors]
  [./Flux1]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g0_L0_M0
    execute_on = 'initial linear nonlinear timestep_end'
  [../]
  [./Flux2]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g1_L0_M0
    execute_on = 'initial linear nonlinear timestep_end'
  [../]
  [./TotalFlx_Fun_PP]
    type = FunctionValuePostprocessor
    function = TotalFlx_function
    execute_on = 'initial timestep_end'
  [../]
  [./NormalizationFlux]
    type = Receiver
    outputs = none
    execute_on = 'initial'
  [../]
[]
```

```
[Functions]
 [./TotalFlx_function]
   type = ParsedFunction
   value = '(F1 + F2)/Norm'
   vars = 'F1 F2 Norm'
   vals = 'Flux1 Flux2 NormalizationFlux'
 [../]
[]

[Transfers]
 [./Copy_NormFlux_PP]
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
   reduction_type = maximum
   from_postprocessor = TotalFlx_Fun_PP
   to_postprocessor = NormalizationFlux
   multi_app = initial_solve
   execute_on = initial
 [../]
[]
```

In the above example, the first two post-processors are performing integrals with respect to space on the scalar flux primal variables `flux_moment` for group 0 and 1. The third post-processor points to the function `TotalFlx_function` and catches the output value. The function uses the other three post-processors to calculate its value. The forth post-processor catches a single value that is coming from a transfer between another application. MultiApps and Transfers are explained in the coming sections. For this example, the value is a normalizing constant determined from an eigenvalue calculation.

2.5.9 Executioner

[Executioner]

The Executioner block is used to set tolerances for the solver, solver method, type of solve and other details related to the timestep and solver. When using Rattlesnake the most common types of solves are going to be NonlinearEigen or Transient. Below are two examples of the executioner, the first is for an eigenvalue calculation and the second is a time dependent/transient problem. The PETsc options are also specified in the executioner.

```
[Executioner]
  type = NonlinearEigen
  solve_type = 'PJFNK'
    #PJFNK - Preconditioned Jacobian-Free Newton Krylov
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
[]
```

```
[Executioner]
  type = Transient
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
#
  start_time = 0.0
  end_time = 10
```

```
#
  l_tol = 1e-8
  nl_rel_tol = 1e-6
  timestep_tolerance = 1E-5
#
 [./TimeStepper]
    type = FunctionDT
    time_dt = '2E-8 1E-6 1E-5 1E-4 1E-3'
    time_t = '0 1E-3 1E-1 1 10'
 [../]
[]
```

2.5.10 MultiApps and Transfers

[MultiApps] and [Transfers]

The multi-app and transfers blocks are the means by which other input files and applications can be started and then the data can be brought back and used by the main application. In Sec. 3.3.2 the multi-app was used to calculate the eigenvalue and the transfers block was used to transfer the desired results over to the transient problem.

Below is an example code for the multi-app, transfers and associated blocks.

```
[MultiApps]
 [./initial_solve]
   type = FullSolveMultiApp
   app_type = RattlesnakeApp
   execute_on = initial
   positions = '0 0 0'
   input_files = SS-Rev2.i
 [../]
[]

[Transfers]
 [./copy_solution]
   type = MultiAppSystemCopyTransfer
   direction = from_multiapp
   multi_app = initial_solve
   execute_on = initial
   scale_with_keff = false
 [../]
 [./Copy_NormFlux_PP]
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
   reduction_type = maximum
   from_postprocessor = TotalFlx_Fun_PP
   to_postprocessor = NormalizationFlux
   multi_app = initial_solve
   execute_on = initial
 [../]
[]

[Postprocessors]

  [./NormalizationFlux]
    type = Receiver
    outputs = none
    execute_on = 'initial'
```

```
  [../]
[]
```

In the above example, the multi-app initiates the eigenvalue solve for the specified input file "SS-Rev2.i" during the initial time step and used the Rattlesnake application. After the solve the data is transferred over with the Transfers block. The first sub-block is a system copy transfer and will transfer all the variables between the eigenvalue calculation and the transient. The transient uses the eigenvalue in the production term of the transport equation for its solves. The second transfer is used to record a post-processor value from the eigenvalue calculation to a post-processor in the transient calculation. The post-processor was created to simply catch the value and hold it so that other calculations could be normalized to it.

2.5.11 Outputs

[Outputs]

The Outputs block is used to determine the data file format and when to export data. The example codes below create two files with the name `Tr_out` with the difference being their extension. The first is a comma separated value (.csv) and will report the scalar values such as post-processor values and the second is an exodus (.e) file and will contain all the information from the calculation, including the spacial dependent variables. The csv files are universal and provide a convenient means of looking at integral data quickly. The exodus files can become very large but can be used to generate 3D views of the data. The second example shows how one could record the data at only specified moments in time. Other file formats also exist. Individual values are not turned off in the outputs block, that is controlled in the block where the value is defined. For instance, in the previous code the post-processor had the option "outputs = none" which turned off the reporting of the value.

```
[Outputs]
  file_base = Tr_out
  csv = true
  exodus = true
[]
```

```
[Outputs]
  execute_on = 'timestep_end'
  file_base = Tr_out
 [./csv]
   type = CSV
   sync_only = true
   sync_times = '0 2e-8 1e-7 1e-6 0.00001 0.00002 0.00004 0.00006 0.00008'
 [../]
 [./exedus]
   type = Exodus
   sync_only = true
   sync_times = '0 3e-8 3e-7 3e-6 0.00003 0.00005 0.00007 0.00009'
 [../]
[]
```

2.5.12 TransportSystems

[TransportSystems]

The TransportSystems is the block where a majority of the neutronics physics are setup. Inside of this block is where the number of energy groups are specified, boundary conditions are set, particle type is chosen and the scheme is setup. The scheme is a a sub-block of transport systems and the general differences were explained in Sec. 1. The capabilities of the schemes are repeated below in Table 2.

Table 2 The capability of schemes.

Scheme	Mathematical adjoint	Neutron	Thermal radiation	Transient	Multiscale
CFEM-Diffusion	Y	Y	Y	Y	Y
DFEM-Diffusion	N	Y	N	Y	N
SAAF-CFEM-SN	Y	Y	Y	Y	Y
SAAF-CFEM-PN	N	Y	N	Y	Y
LS-CFEM-SN	N	Y	N	Y	N
LS-CFEM-PN	N	N	N	N	N
DFEM-SN	Y	Y	N	Y	N
DFEM-PN	N	N	Y	N	N

Below are examples of TransportSystems codes for a transient and eigenvalue problems with two energy groups (ie. G = 2). The particle type was set to neutron which enables neutron specific options within the scheme (like `n_delay_groups`). The vacuum boundary is referring to surfaces that were setup in the mesh block. The scheme sub-block setup other options such as the number of angles.

```
[TransportSystems]
  particle = neutron
  equation_type = transient
  G = 2
  VacuumBoundary ='1 2'
  [./SN]
    scheme = SAAF-CFEM-SN
    family = LAGRANGE
    order = FIRST
    AQtype = Gauss-Chebyshev
    NPolar = 8
    n_delay_groups = 6
    fission_source_as_material = true
  [../]
[]
```

```
[TransportSystems]
  particle = neutron
  equation_type = eigenvalue
  G = 2
  VacuumBoundary ='1 2'
  [./SN]
    scheme = SAAF-CFEM-SN
    family = LAGRANGE
    order = FIRST
    AQtype = Gauss-Chebyshev
    NPolar = 8
    n_delay_groups = 6
    fission_source_as_material = true
  [../]
```

```
[]
```

The following example changes the scheme to a PN calculations instead of a SN calculation.

```
[TransportSystems]
  particle = neutron          # Solving the transport problem using the neutron equation
  equation_type = eigenvalue  # Doing an eigenvalue type problem for Steady State
  for_adjoint = false         # Non-adjoint weighted

  # Number of energy groups
  G = 2

  # Boundary Condition
  VacuumBoundary ='1 2'

  [./SN]
    scheme = SAAF-CFEM-PN
    family = LAGRANGE
    order = FIRST
    PN = 12  # PN option

    n_delay_groups = 6  # added because of "partical = neutron"
    fission_source_as_material = true  # added because of "partical = neutron"

  [../]
[]
```

2.6 When encountering problems

2.6.1 Debug input block

The following is generated by

```
>./rattlesnake-opt --dump Debug
```

```
[Debug]
  show_actions             = 0        # Print out the actions being executed
  show_material_props      = 0        # Print out the material properties supplied
                                      # for each block, face, neighbor, and/or sideset
  show_parser              = 0        # Shows parser block extraction and debugging
                                      # information
  show_top_residuals       = 0        # The number of top residuals to print out
                                      # (0 = no output)
  show_var_residual_norms  = 0        # Print the residual norms of the individual
                                      # solution variables at each nonlinear iteration
  show_var_residual        =          # Variables for which residuals will be sent to
                                      # the output file
  boundary                 =          # The side sets boundary converge check is
                                      # targeting for, none means all side sets
  check_boundary_coverage  = 0        # Check if all boundary sides are covered by side
```

```
                                    # sets
  domain                  =         # The blocks boundary coverage check is working
                                    # on, none means all blocks
  print_block_volume      = 0       # Print the volumes of blocks
  show_petsc_options      = 0       # Print PETSc options obtained via PetscOptionsGetAll
[]
```

2.6.2 Check your mesh

If your mesh is not generated by the build-in mesh generators, it is highly recommended to check your mesh with the mesh checker provided by INSTANT, which is under *yak/contrib/instant*.

You will need to first make INSTANT by doing

```
>cd yak
>make instant -j4
```

A few executables will be generated under *yak/contrib/instant*. One of them, *instant_mesh_generator-opt*, is the INSTANT mesh generator which can be used as a checker with the following input file:

```
<task type="modification" dim="$dim$">
    <input>$input_exodus_file$</input>
    <output>$output_exodus_file$</output>
</task>
```

Users are required to specify the dimension of the mesh `$dim$`, either 2 or 3, and the input exodus file `$input_exodus_file$` and the output exodus file `$output_exodus_file$`. The input and output file name need to be different. The mesh check will print messages on the screen showing the progress of checking and generate a text output file *Exodus_utilities.outp*, in which detected errors, if there is any, will be listed.

2.6.3 Run your problem in debug mode

First you will need to create an Rattlesnake executable, *rattlesnake-dbg*, in debug version with

```
>METHOD=dbg make -j8
```

If you are using Mac, run your problem with

```
lldb -- ./rattlesnake-dbg -i <InputFile>
```

For Linux, use

```
gdb --args ./rattlesnake-dbg -i <InputFile>
```

After the lldb or gdb prompt shows up, set a break point on the Rattlesnake internal error handler with

```
b libmesh_terminate_handler
```

then run

```
r
```

To obtain a back trace of the calling stack use

```
bt
```

The back trace can typically give you a clue on what could be wrong. We would typically ask you for the back trace when you cannot find the problem and need further assistance.

2.6.4 Ask questions on our user forum

(User forum is to be set up.)

2.7 Get involved

All users can create new issues or leave comments on the Rattlesnake *gitlab* page. To gain access to the source code, you need to have at least a *reporter* level. You can ask for this level of access by contacting the Rattlesnake developers and provide justification. If you made changes in your version of Rattlesnake and think these changes are useful, we encourage you to send these changes as merge requests (MR) for review and to be merged into the main Rattlesnake repository. You will have to gain *developer* access to do so. We have a strict rules/procedures for how Rattlesnake can be changed. We also enforce a set of coding styling rules for Rattlesnake. Getting familiar with these rules can save time during the reviewing process of your MR. We are basically following the MOOSE development procedure, which can be found at http://mooseframework.org/wiki/Contributing/.

3 Tutorial: Example input files

3.1 Kobayashi benchmark

This is a simple one-group source problem with void. All files necessary for running this tutorials with Rattlesnake are under *'rattlesnake/tutorials/Kobayashi'* folder.

3.1.1 Problem description

Plane geometries and a sketch of the problem are shown in Fig. 1 as the third problem in [3], which is called the dog leg void duct problem. Reflective boundary conditions are used at the boundary planes $x = 0$, $y = 0$ and $z = 0$, and vacuum boundary conditions at all outer boundaries. Cross sections and source strength in the three regions are given in Table 3. Results can be either generated without the scattering or with the scattering

Table 3 Cross sections and source strength.

Region	S $(n \cdot cm^{-3} \cdot s^{-1})$	Σ_t (cm^{-1})	Σ_s (cm^{-1})	
1	1	0.1	0	0.05
2	0	10^{-4}	0	0.5×10^{-4}
3	0	0.1	0	0.05

cross sections being the half of the corresponding total cross sections.

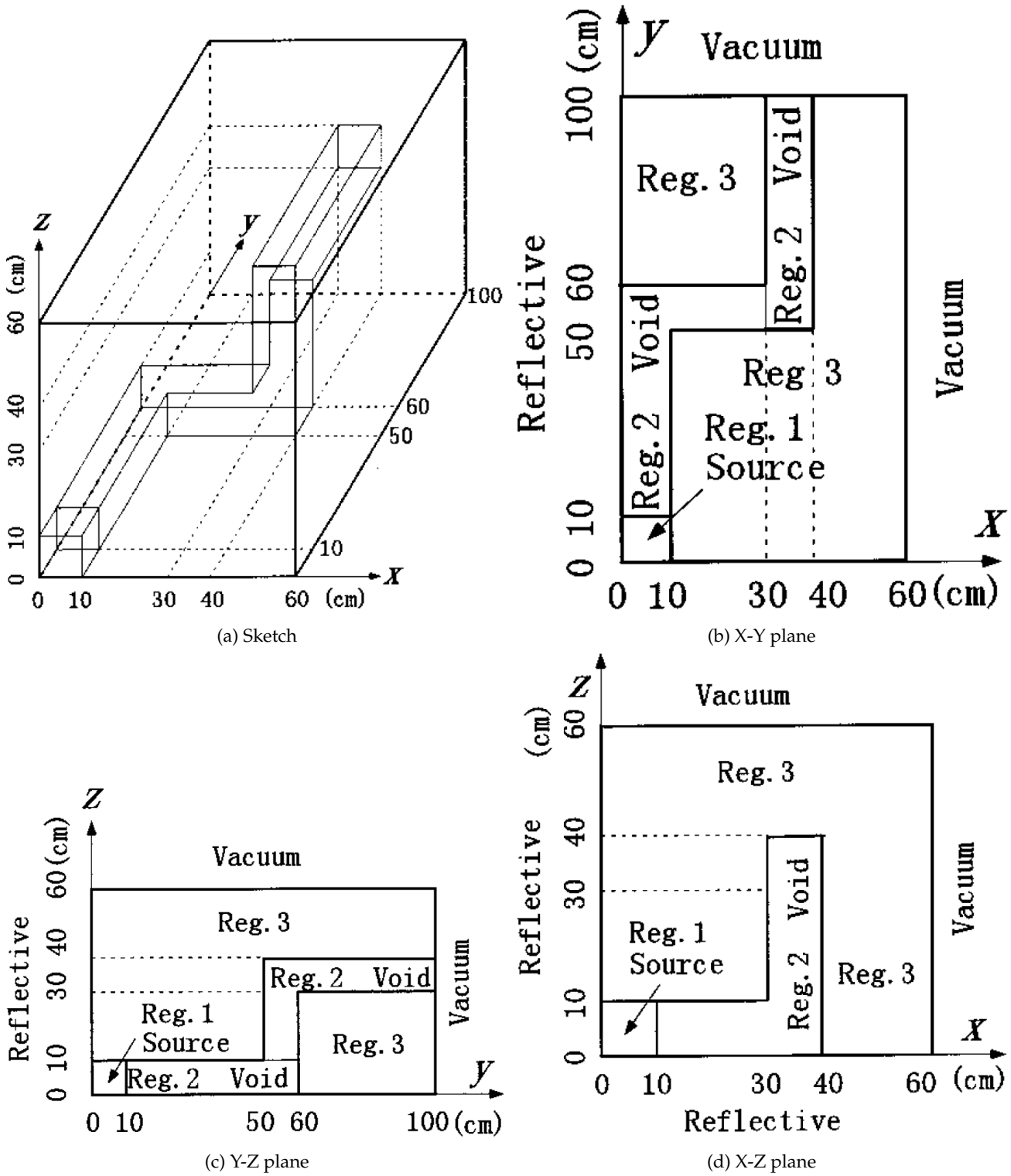
3.1.2 Mesh

The geometry of this problem is regular, so we can used *CartesianMesh* to generate a regular mesh covering the geometry and to assign block IDs for different regions.

```
[Mesh]
 type = CartesianMesh
 dim = 3
 dx = '10 20 10 20'
 dy = '10 40 10 40'
 dz = '10 20 10 20'
 ix = '2 4 2 4'
 iy = '2 8 2 8'
 iz = '2 4 2 4'
 subdomain_id = '
   1 3 3 3
   2 3 3 3
   2 2 2 3
   3 3 2 3

   3 3 3 3
   3 3 3 3
   3 3 2 3
   3 3 3 3

   3 3 3 3
   3 3 3 3
   3 3 2 3
   3 3 2 3
```



(a) Sketch

(b) X-Y plane

(c) Y-Z plane

(d) X-Z plane

Figure 1 Geometry of the Kobayashi benchmark.

```
   3 3 3 3
   3 3 3 3
   3 3 3 3
   3 3 3 3'
 uniform_refine = 0
 second_order = false
[]
```

We will need more uniform refinements and possibly quadratic shape functions to reduce the spatial discretization error.

3.1.3 Transport system with SAAF-CFEM-SN

There is no fission in this benchmark, so we use *common particle* for the transport system. We use level-symmetric angular quadrature for the calculation. The spatial polynomial *order* is set to tri-linear (i.e. FIRST).

```
[TransportSystems]
  particle = common
  equation_type = steady-state

  G = 1

  VolumetricSourceBlock = '1'
  VolumetricSource = '1.0'

  VacuumBoundary = 'right front top'
  ReflectingBoundary = 'left back bottom'

  [./saaf]
    scheme = SAAF-CFEM-SN
    AQtype = Level-Symmetric
    AQorder = 8
    order = FIRST
    hide_angular_flux = true
  [../]
[]
```

The default stabilization parameter $\tau = 0.5$ for treating void is used. We will definitely need higher SN order for reducing the angular discretization error of this problem. Typically for SN calculations, we do not care much about the angular flux, so we use *hide_angular_flux* to hide all angular fluxes in the applicable outputs, for example, in the Exodus output file.

3.1.4 Materials

It is proper to use the simple *ConstantNeutronicsMaterial* for this benchmark.

```
[Materials]
  [./region13]
    type = ConstantNeutronicsMaterial
    block = '1 3'
    sigma_t = 0.1
```

```
    sigma_s = 0.0
#    sigma_s = 0.05
  [../]
  [./region2]
    type = ConstantNeutronicsMaterial
    block = 2
    sigma_t = 0.0001
    sigma_s = 0.0
#    sigma_s = 5e-5
  [../]
[]
```

The commented lines are used for switching calculations with the scattering.

3.1.5 Postprocessors

We will evaluate the L2 norm of the scalar flux and the scalar flux value at bunch of points required by the benchmark.

```
[Postprocessors]
  [./norm]
    type = ElementL2Norm
    variable = flux_moment_g0_L0_M0
  [../]

  [./3A01]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '5 5 5'
  [../]
  [./3A02]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '5 15 5'
  [../]
  [./3A03]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '5 25 5'
  [../]
  [./3A04]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '5 35 5'
  [../]
  [./3A05]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '5 45 5'
  [../]
  [./3A06]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '5 55 5'
  [../]
  [./3A07]
    type = PointValue
```

```
  variable = flux_moment_g0_L0_M0
  point = '5 65 5'
[../]
[./3A08]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '5 75 5'
[../]
[./3A09]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '5 85 5'
[../]
[./3A10]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '5 95 5'
[../]

[./3B01]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '5 55 5'
[../]
[./3B02]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '15 55 5'
[../]
[./3B03]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '25 55 5'
[../]
[./3B04]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '35 55 5'
[../]
[./3B05]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '45 55 5'
[../]
[./3B06]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '55 55 5'
[../]

[./3C01]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '5 95 35'
[../]
[./3C02]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '15 95 35'
[../]
[./3C03]
```

```
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '25 95 35'
  [../]
  [./3C04]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '35 95 35'
  [../]
  [./3C05]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '45 95 35'
  [../]
  [./3C06]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '55 95 35'
  [../]
[]
```

3.1.6 Solver

Because there is no scattering or low scattering ratio (0.5) in this benchmark, it will be more efficient to use the executioner performing source iteration instead of using the full PJFNK solver. We still leave the parameters for the full PJFNK solver commented in the Executioner block. Because the discretization is CFEM based, we choose the special executioner *AMGUpdate*. BoomerAMG [4] is used in this executioner with the strong threshold equal to 0.7. Because the out-going angular fluxes on the reflecting boundaries are lagged by one iteration, we will need four source iterations to converge this problem even it does not have scattering.

```
[Executioner]
  type = AMGUpdate
  richardson_max_its = 10
  richardson_abs_tol = 1e-8
  debug = true
  amg_tol = 1e-3
  amg_abs_tol = 1e-9
  pre_pc_setup = false
#  type = Steady
#  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
#  petsc_options_value = 'hypre boomeramg 10'
#  l_max_its = 50
#  nl_rel_tol = 1e-12
[]
```

3.1.7 Outputs

We want the Exodus [5] output for this benchmark for visualization. The Exodus file can be viewed by VisIt [6] and ParaView [7]. We want the postprocessor values to be stored in a CSV (Comma Separated Values) file for record.

```
[Outputs]
  exodus = true
```

```
  print_perf_log = true
  [./csv]
    type = CSV
    file_base = kobayashi_out
    align = true
    precision = 6
    execute_on = timestep_end
  [../]
[]
```

We want more control on the CSV output here by creating a *csv* sub-block. We want the names and the values in the CSV file to be aligned for better readability. More precision than 6 digits is not needed. And we do not need the values on initial (just zeros) to be outputted.

3.1.8 Results

We only presents results without scattering. To reduce the spatial discretization error, we uniformly refine the mesh two times by adding the following command-line parameters

```
Mesh/uniform_refine=2
```

To reduce the angular discretization error, we use S24 level-symmetric quadrature with

```
TransportSystems/saaf/AQorder=24
```

The total number of unknowns with this setting is $121,356,144$. The calculation was conducted on INL Falcon cluster. 32 nodes each with 12 processors are required. The calculation can be finished in 9.67min with 4 source iterations. We expect that the CPU time to be smaller with a dedicated algebraical multigrid method for SAAF-CFEM-SN.

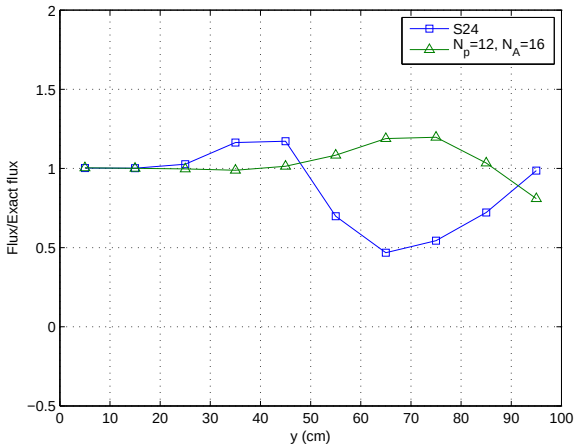
The results on $x = z = 5cm$, $y = 55cm, z = 5cm$ and $y = 95cm, z = 35cm$ are plotted in Fig. 2. Another angular quadrature is also tried:

```
[TransportSystems]
  ...
  [./saaf]
    ...
    AQtype = Gauss-Chebyshev
    NPolar = 12
    NAzmthl = 16
  [../]
[]
```

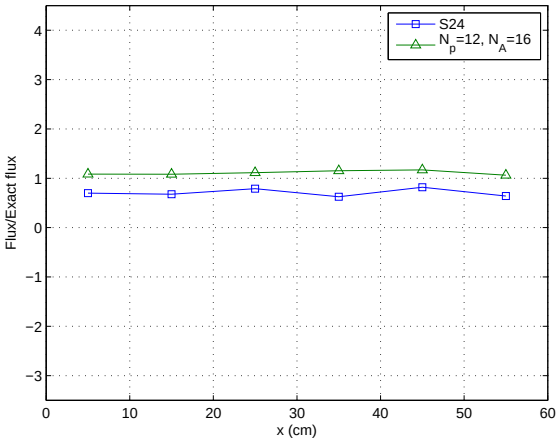
Results are also plotted in Fig. 2.

3.2 Benchmark 16A1-1 (Eigenvalue Problem)

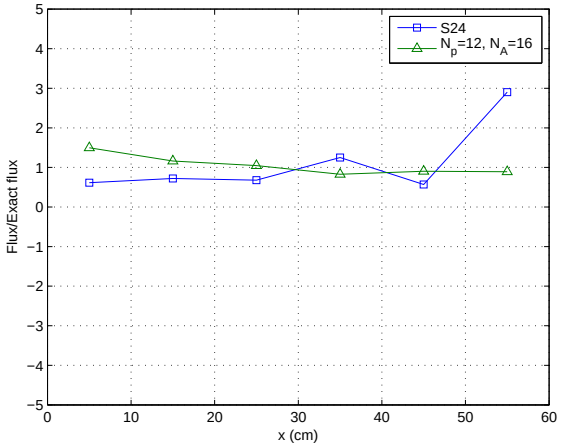
All files necessary for running this tutorials with Rattlesnake are under *'rattlesnake/tutorials/16A1'* folder.



(a) $x = z = 5cm$



(b) $y = 55cm, z = 5cm$



(c) $y = 95cm, z = 35cm$

Figure 2 Kobayashi benchmark results.

3.2.1 Problem description

This example covers the calculation of the eigenvalue for the problem described in the 16A-1 benchmark which can be found in Ref. [8]. The benchmark is a transient calculation. However, for illustrative purposes the problem is being broken-up into two parts one for the eigenvalue and one for the transient. See Sec. 3.3 for the transient part of the calculation.

Descriptive Title: Delayed Supercritical Transient; One-Dimensional, Two-Group Neutron Transport Problem in a Fast Reactor

(This example only covers the eigenvalue calculation)

Description:

1. One-dimension (slab with azimuthal symmetry), two-group neutron transport theory
2. Seven zones
3. Isotropic scattering
4. Zero return current boundary conditions on external surfaces
5. Steady-state critical initial conditions
6. Six delayed neutron precursor groups
7. At t=0 sec, the density of the material in zone 2 increases by 5% and the material in zone 6 is decreased by 5%. (This will not apply until the transient calculation in Sec. 3.3.)

Table 4 Zones

Zone	Length(cm)	Number of Intervals
1	40	20
2	47.374	24
3	9	5
4	34	16
5	9	5
6	47.374	24
7	40	20

On the sides of zone 1 and 7 are vacuum. The blanket material is in zones 1 and 7. The core material is in zones 2, 4 and 6. A mixture of sodium and control rod material is in zones 3 and 5.

Table 5 Cross-sections

Zone	Group	$\nu\Sigma_f^g$	Σ_t^g	$\Sigma_s^{g\to g}$	$\Sigma_s^{g\to g'}$
1,7	1	8.3441E-4	2.411E-1	2.33644E-1	3.598E-3
	2	3.2776E-4	4.172E-1	4.07004E-1	0.0
2,4,6	1	7.4518E-3	1.849E-1	1.77711E-1	2.085E-3
	2	1.1061E-2	3.668E-1	3.53721E-1	0.0
3,5	1	0.0	9.432E-2	8.571E-2	1.717E-3
	2	0.0	1.876E-1	1.7131E-1	0.0

In Table 5 the second to last column is the scattering within a group while the last column is the scattering between groups. For this case there is no up-scatter. The neutron speeds for this problem are given as $1/v_1 = 1.851x10^{-09}s/cm, 1/v_2 = 1.088x10^{-08}s/cm$. The prompt and delayed neutron spectra are identical with $x_1 = 1.0\, and\, x_2 = 0.0$.

Table 6 Delayed Neutron Parameters

Zone	β_i	$\lambda(sec^{-1})$
1	0.81E-4	0.0129
2	6.87E-4	0.0311
3	6.12E-4	0.134
4	11.38E-4	0.331
5	5.12E-4	1.26
6	1.70E-4	3.21

3.2.2 Mesh

The specifications for the desired mesh is given in Table 4. Since this problem is rather simple the *CartesianMesh* generator may be used to create the mesh.

```
[Mesh]
 type = CartesianMesh
 dim = 1
 dx = '40 47.374 9 34 9 47.374 40'
 ix = '20 24 5 16 5 24 20'
 subdomain_id = '1 2 3 4 5 6 7'

 # dx is the width of each subdomain_id
 # ix is the number of mesh intervals to place within each dx.
 # Boundary conditions are 'right left'
#
 uniform_refine = 0
 # modify uniform_refine to 1 or 2 to check for convergence
[]
```

In this code, the *dim* option refers to the number of Cartesian dimensions, *dx* is the spacing for each *subdomain_id* and *ix* is the number of intervals within each *dx* region. The *subdomain_id* is the specification for each region. These sub domains are referenced in other parts of the input file (such as in the materials section) as `block = ' ...'`. The mesh generator also automatically generates boundary surfaces. For a 1-D case the only boundary options are `'left right'`. These are used in the *TransportSystems* block.

The *uniform_refine* option, can be used to automatically increase the number of elements in a mesh by refining each region. When set to 0 refinement does not take place and the mesh is exactly the same as was specified in the mesh generator. When refinement is set to 1 the number of mesh points are doubled. When refinement is set to 2 the refinement doubles again and there are four times the number of mesh points then the original mesh. Modifying the *uniform_refine* option is a convenient method to check for mesh convergence.

Additional dimensions may be add with the *CartesianMesh* using *dy* and *dz* with *iy* and *iz* for 3-D. The boundary surface names used for boundary conditions are left, right, top, bottom, front and back. For more information on building a mesh refer to Sec. 6.

3.2.3 Transport system

Transport systems is the branch that is used to setup the transport physics. Below is the code to setup.

```
[TransportSystems]
```

```
  particle = neutron         # Solving the transport problem using the neutron equation
  equation_type = eigenvalue  # Doing an eigenvalue problem

  # Number of energy groups
  G = 2
  #
  # Boundary Condition
  # Boundary Conditions are created automatically by the mesh generator
  VacuumBoundary ='left right'     # Zero Return on External Surfaces
  #
  #
  [./SN]
    scheme = SAAF-CFEM-SN
    family = LAGRANGE
    order = FIRST
    #
    # The remaining options depend on the scheme used
    #
    AQtype = Gauss-Chebyshev   # Angular quadrature type (It needs to be gauss-cheby for 1-D)
    NPolar = 8                 # Number of polar angles (polar because its 1-D)
    NA = 0                     # NA is the maximum scattering anisotropy (0 for isotropic)
    verbose = 2  # verbose is printouts

    n_delay_groups = 6  # added because of "partical = neutron"
    fission_source_as_material = true  # added because of "partical = neutron"
  [../]
[]
```

This is a neutron transport problem so the `particle = neutron`. The equation type is eigenvalue, the number of groups is 2 and the vacuum boundary needs to be located to the left of zone 1 and to the right of zone 7. The options for `VacuumBoundary` can change based on the way the mesh was created. The mesh was generated using the *CartesianMesh* generator and the for 1-D the key words are left and right.

The sub-branch is set using the `scheme = SAAF-CFEM-SN` which means that the calculation for the solve will be using a self-adjoint angular flux formulation with the continuous finite element method and using the SN method for angular treatment. One will note that the sub-block type is defined by scheme, which is different than the normal MOOSE syntax `"type = "`. Other schemes could have been chosen as well such as SAAF-CFEM-PN, LS-CFEM-SN or DFEM-SN for this eigenvalue problems. Because the particle is neutron, extra options become available in the scheme. These options are `n_delay_group and fission_source_as_material`. The user can choose the shape function family and order. The most common is Lagrange, which is a continuous function. Since this is a 1-D problem it is required to use a Gauss-Chebyshev for the angular quadrature type and the number of polar angles needs to be chosen. For this problem, not many polar angles are required.

3.2.4 Materials

The materials block will setup the cross-sections and which zones will have those cross-sections. Below is the code to generate one of the cross-sections and the rest will be added after this part of the code is explained.

```
[Materials]
  [./Mat1]
    type = ConstantNeutronicsMaterial
    block = '1 7'

    fissile = true
    #
    # x-sections
```

```
    #
    nu_sigma_f = '8.3441E-4 3.2776E-4'
    # nu_sigma_f is the fission cross-section
    sigma_t = '2.411E-1 4.172E-1'
    # sigma_t is the total cross-section
    sigma_s = '2.33644E-1 0.0
               3.598E-3 4.07004E-1'

    # sigma_s is the scattering matrix: sigma_s: 1->1, 2->1, 1->2, 2->2
    chi = '1 0'
    # chi is the fraction of prompt neutrons that appear in each group

    # Delayed properties
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0
                      1 0
                      1 0
                      1 0
                      1 0
                      1 0'

    # delay_sepectrum is the delayed neutron spcectrum for Xd_g_i = Xd_1_1, Xd_2_1 ... Xd_1_6, Xd_2_6
    #   its the spectrum at each energy group and for each delayed group
    #   (energy group first followed by delayed group)
    neutron_speed = '540248514.31659 91911764.705882'
    # Neutron speed is the inverse of the numbers provided for 1/v1 and 1/v2.

  [../]
...
[]
```

We are using the `ConstantNeutronicsMaterial` sub-block within Rattlesnake and giving it the name Mat1. This material is going to be applied to the mesh at zones 1 and 7. The material contains fission cross-sections so we have `fissile = true`. We provide the cross-sections for `nu_sigma_f, sigma_t and sigma_s`, which are fission, total and scattering respectively. The order is to provide the cross-sections for the first energy group first. By convention the first energy group number is the highest energy group. Because C++ is a zero based program the first index is zero instead of one. The scattering matrix involves iterating between g and then g'. The option `chi` is the fraction of prompt neutrons that appear in each group. The `delayed_spectrum` also has the same spectrum as chi, but with one for each group. The delayed neutron fraction and decay constants are also defined. The delayed neutron options are not included for zones that do not include fissile material. The `neutron_speed` is also the neutron speed for each group.

These pattern can be followed to develop the materials for zones 2,3,4,5 and 6. Since, this is an eigenvalue calculation before the transient begins, zones 2,4,6 may be combined into one material definition. Or zones 2, 4 and 6 may be defined independently so that they can be copied and pasted into the transient file and all that needs done is a modification to the cross-sections for zones 2 and 6.

Below is the full input for the materials block. Note that zones 3 and 5 (ie. `block = '3 5'`) do not include delayed options. In reality the `neutron_speed` is not needed for the eigenvalue equation but will be required for the transient.

```
[Materials]
  [./Mat1]
    type = ConstantNeutronicsMaterial
    block = '1 7'

    fissile = true
```

```
    #
    # x-sections
    nu_sigma_f = '8.3441E-4 3.2776E-4'
    sigma_t = '2.411E-1 4.172E-1'
    sigma_s = '2.33644E-1 0.0
               3.598E-3 4.07004E-1'
    chi = '1 0'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0
                      1 0
                      1 0
                      1 0
                      1 0
                      1 0'
    neutron_speed = '540248514.31659 91911764.705882'
  [../]

  [./Mat4]
    type = ConstantNeutronicsMaterial
    block = '2 4 6'
    fissile = true
    #
    # x-sections
    nu_sigma_f = '7.4518E-3 1.10612E-2'
    sigma_t = '1.849E-1 3.668E-1'
     sigma_s = '1.77711E-1 0.0
               2.085E-3 3.53721E-1'
    chi = '1 0'
    neutron_speed = '540248514.31659 91911764.705882'

    # Delayed properties
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0
                      1 0
                      1 0
                      1 0
                      1 0
                      1 0'
  [../]

  [./Mat3]
    type = ConstantNeutronicsMaterial
    block = '3 5'
    fissile = false
    neutron_speed = '540248514.31659 91911764.705882'
    #
    # x-sections
    #
    sigma_t = '9.432E-2 1.8762E-1'
    sigma_s = '8.571E-2 0.0
               1.7168E-3 1.7131E-1'
  [../]
[]
```

3.2.5 Executioner

There are many options in the executioner to control tolerances. To perform the eigenvalue calculations a simple code like the following would suffice. The most important part for the executioner is for the line `type=NonlinearEign`. This command specifies the type of solve we are performing. Notice that the specification for eigenvalue is in two locations. The executioner and TransportSystems.

```
[Executioner]
  type = NonlinearEigen

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
  free_power_iterations = 4
[]
```

In the above code the solver type is the pre-conditioned Jacobian-Free Newton Krylov. The default is the Jacobian-Free Newton Krylov (JFNK). There are two PETSc options. PETSc stands for Portable, Extensible Toolkit for Scientific Computing and provides data structures and routines for parallel solutions and is built into MOOSE. One may wonder what petsc options to use. So, far the experience has been to just pick something that works. The above code for the PETSc options have worked for many applications using Rattlesnake. The `free_power_iterations = 4` option is useful for eigenvalue calculations. The default value is 4. The free power iteration method provides a convenient means of converging on an eigenvalue solution. The trade-off are the number of free power iterations increases the computational time but provides a better convergence. For calculations where the eigenvalue is not too important lower values like 2 may suffice. A typical value for free power iterations is 2 to 6.

3.2.6 Outputs

In the outputs block the user can choose how to save the data and where to save. Since this is an eigenvalue calculation all that is required is the ending eigenvalue. The value will be printed on the screen or to a job file even if there are no output files. We can save all the information for every mesh point using the `exedus = true` option. This will save an exodus file that can be opened using ParaView. ParaView is free to download from Sandia National Laboratory. The `csv` option will also create a comma separated file for concise numbers. For both the csv and exodus files they will be called by the `file_base` name test with the extension being .e for exodus and .csv for the comma separated file.

```
[Outputs]
  execute_on = 'timestep_end'
  file_base = test
  exodus = true
  csv = true
[]
```

3.2.7 Optional - (Postprocessors, AuxVariables, AuxScalarKernels and Functions)

This section is optional but may be very useful for learning many basic functions. As part of the official benchmark the normalized power history (ie. scalar flux vs time) is to be reported. This means that we need to integrate the scalar flux with respect to space for both energy groups, add the fluxes together and use the result from the eigenvalue calculation to normalize the values for the transient.

First we want to integrate the scalar flux with respect to space. This calls for a post-processor. Below is a code that will integrate the scalar flux (ie. flux moment) for both energy groups. The variables `flux_moment_g0_L0_M0` and `flux_moment_g1_L0_M0` are automatically created from the TransportSystems block whenever one of the transport schemes is chosen. The g0 or g1 portion refers to energy group0 or group1. The results are two scalar quantities Flux1 and Flux2. The integral is applying to mesh zones 1 to 7 which is indicated by `block='1 .. 7'`.

```
[Postprocessors]
  [./Flux1]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g0_L0_M0
    execute_on = 'linear nonlinear'
    #outputs = none
  [../]
  [./Flux2]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g1_L0_M0
    execute_on = 'linear nonlinear'
  [../]
[]
```

The reason for adding the `execute_on =` option will be explained further on but the meaning is when the post-processor will be calculated. Go ahead and change the option to others such as `linear`, `timestep_end`, `timestep_begin`, `'initial linear'` etc. Notice that the single quotes is not needed unless two or more options are used. Further, multiple execution time steps are allowed. Bear in mind that the sequence of execution is a switching between non-linear and linear solves. Therefore the execution process is `timestep_begin`, `nonlinear`, `linear` and `timestep end`. There are also an option for `initial`. The outputs option has temporarily been commented out to allow the user to understand how they can chose to have the post-processor displayed on the screen and on the output file. Go ahead and add these post-processors to the input file and run. Then change the `execution_on and outputs` options.

Now that we have the scalar flux for the entire core we need to combine the two values. There are multiple methods that can be used to do this. Below one method will be shown. The plan will be to define a new variable that will get the value of a function that adds the two post-processors together. The following code does just this.

```
[AuxVariables]
    [./TotalFlux]
        family = scalar
        order = first
        #outputs = none
    [../]
[]
[AuxScalarKernels]
    [./TotalScalarFlux_kern]
        type = FunctionScalarAux
        execute_on = timestep_end
        function = TotalFlx_function
        variable = TotalFlux
    [../]
[]

[Functions]
 [./TotalFlx_function]
   type = ParsedFunction
```

```
    value = 'F1 + F2'
    vars = 'F1 F2'
    vals = 'Flux1 Flux2'
  [../]
[]
```

In the code above a new variable `TotalFlux` is defined. The type is scalar, since it is a single value and does not depend on space. Given that the type is scalar the order takes on a new significance. When the variable is dependent on space the order refers to the polynomial order of the shape function. In this case, the order defines a vector size, which is one for our case. Just like the post-processors can be turned on/off by the outputs option, so can variables.

The `AuxScalarKernels` is setup to be a function and apply to the variable `TotalFlux`. The kernel must also know which function is going to be applied. That is why the user defined name is placed after `function =`. The time step option was placed in the kernel to control when the calculation is performed.

The `Functions` block is set to be a parsed function. This type requires three parameters (vals, vars, and value). The vals are the post-processors which have been named by the user Flux1 and Flux2 (see the Postprocessors block). The value option is where the user defines the equation to be performed. You will notice that the equation uses F1 and F2 not Flux1 and Flux2. The vars are used to define other variables names which are used in the actual equation instead of the original variables. One advantage to this operation is so that the equation can be defined by shorter variable names. The order in which the vars are entered corresponds to the order in which the vals were entered.

Add the above code to the input file and run. Go ahead an change the names and experiment with the operations.

Now, one maybe wondering why not just use the default values for the `execute_on` feature. If you comment out the `execute_on` feature, look at the values for Flux1, Flux2 and TotalFlux. Flux1 and Flux2 no longer add to TotalFlux. The reasoning is that when the fluxes are added together matters and when the values for Flux1 and Flux2 are calculated matters as well. For instance, if the post-processors and the variable, TotalFlux, were given an execution priority of time step end, then it would be possible for the post-processors to update after the TotalFlux variable is updated, leading to a TotalFlux value that is based on the previous step. The question becomes which blocks are executed first in MOOSE for a given `execute_on` command. For this particular situation we really do not need to know but can get a good answer by keeping the post-processors up-to-date as much as possible, instead of at the end of a solve where it might be updated after TotalFlux is updated. This is why the option 'linear nonlinear' was used (the option for `timestep_end` could have also been included). While for the TotalFlux it was calculated at `timestep_end`.

Now we know that once we get to the transient calculation we will want to normalize by `TotalFlux`. We will cover in the transient section how to transfer between the steady state eigenvalue calculation and the transient. But for now will mention that in the transient calculation we want to modify the ParsedFunction to divide by the total flux from the eigenvalue calculation. There is a problem if we try to mix the postprocessors values with variable types. Conversion can take place but why make it complicated. How about we perform the calculation in an alternative way.

Below are two post-processors that we could use to either transfer the `TotalFlux` variable to a post-processor value or calculate the total flux directly into the post-processor using the function that was setup.

```
[Postprocessors]
#...
#...
#
# This Postprocessor (PP) will copy the AuxVariable TotalFlux to a PP value.
  [./TotalFlx_PP]
    type = ScalarVariable
    variable = TotalFlux
```

```
  [../]
#
#
# This PP will use the function TotalFlx_function to obtain its value.
  [./TotalFlx_Fun_PP]
    type = FunctionValuePostprocessor
    function = TotalFlx_function
  [../]
[]
```

If we use latter post-processor there is no need to use the aux-variable or aux-kernel blocks from the previous code. Given that the second option is easier we will use that option. Keep in mind to check if the execution sequence is correct by running the input file and checking to make sure Flux1 + Flux2 equals `TotalFlx_Fun_PP`.

3.2.8 Run the file

With the blocks Mesh, TransportSystems, Materials, Executioner and Outputs it is possible to run the code for the eigenvalue. Below is the input with the required branches. You will notice the comments and a few lines have been deleted or changed from the code given in each individual section. This was meant to test the reader and shorten the text.

```
[Mesh]
 type = CartesianMesh
 dim = 1
 dx = '40 47.374 9 34 9 47.374 40'
 ix = '20 24 5 16 5 24 20'
 subdomain_id = '1 2 3 4 5 6 7'
 uniform_refine = 0
[]


[TransportSystems]
  particle = neutron
  equation_type = eigenvalue
  G = 2
  VacuumBoundary ='1 2'
  [./SN]
    scheme = SAAF-CFEM-SN
    family = LAGRANGE
    order = FIRST
    AQtype = Gauss-Chebyshev
    NPolar = 8
    n_delay_groups = 6
    fission_source_as_material = true
  [../]
[]

[Materials]
  [./Mat1]
    type = ConstantNeutronicsMaterial
    block = '1 7'
    fissile = true
    nu_sigma_f = '8.3441E-4 3.2776E-4'
    sigma_t = '2.411E-1 4.172E-1'
    sigma_s = '2.33644E-1 0.0 3.598E-3 4.07004E-1'
    chi = '1 0'
    neutron_speed = '540248514.31659 91911764.705882'
```

```
    # Delayed properties
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]

  [./Mat4]
    type = ConstantNeutronicsMaterial
    block = '2 4 6'
    fissile = true
    nu_sigma_f = '7.4518E-3 1.10612E-2'
    sigma_t = '1.849E-1 3.668E-1'
    sigma_s = '1.77711E-1 0.0 2.085E-3 3.53721E-1'
    chi = '1 0'
    neutron_speed = '540248514.31659 91911764.705882'
    # Delayed properties
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]

  [./Mat3]
    type = ConstantNeutronicsMaterial
    block = '3 5'
    fissile = false
    neutron_speed = '540248514.31659 91911764.705882'
    sigma_t = '9.432E-2 1.8762E-1'
    sigma_s = '8.571E-2 0.0
               1.7168E-3 1.7131E-1'
  [../]

[]
[Executioner]
  type = NonlinearEigen
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
[]

[Outputs]
  file_base = SS_out
  exodus = true
  csv = true
[]
```

Save the text file and give it a name with the extension ".i", such as test.i. To run the file use the command below while in the folder with the input file:

```
./rattlesnake-opt -i <InputFile>
## Comment: For the example input file test.i use
./rattlesnake-opt -i test.i
```

Below is the input file with the optional blocks to obtain the total flux using post-processors which will be used to normalize the flux from the transient. Notice that the `GlobalParams` block was added. This block is a convenient way to place options or values that will be included many times over and shorten the code. For this example the neutron speed is the only quantity that can be placed in every ConstantNeutronicsMaterial sub-block and thus can be used in the `GlobalParams` block. For this example it is hardly worth using the

GlobalParams block. But instead of copy and pasting many options over and over again this can be a good alternative. The one catch is that since the parameters become global, you have to be careful you do not have a sub-block that takes in the option and needs the value to be different than the one in the GlobalParams block. Also, you will observe that zones 2 and 6 (ie. block =) have been separated away from zone 4 into their own materials. That way we can easily copy and modify the cross-sections for the transient.

```
[Mesh]
 type = CartesianMesh
 dim = 1
 dx = '40 47.374 9 34 9 47.374 40'
 ix = '20 24 5 16 5 24 20'
 subdomain_id = '1 2 3 4 5 6 7'
 uniform_refine = 0
[]

[GlobalParams]
    neutron_speed = '540248514.31659 91911764.705882'
[]

[TransportSystems]
  particle = neutron
  equation_type = eigenvalue
  G = 2
  VacuumBoundary ='1 2'
  [./SN]
    scheme = SAAF-CFEM-SN
    family = LAGRANGE
    order = FIRST
    AQtype = Gauss-Chebyshev
    NPolar = 8
    n_delay_groups = 6
    fission_source_as_material = true
  [../]
[]

[Functions]
 [./TotalFlx_function]
   type = ParsedFunction
   value = 'F1 + F2'
   vars = 'F1 F2'
   vals = 'Flux1 Flux2'
 [../]
[]
[Materials]
  [./Mat1]
    type = ConstantNeutronicsMaterial
    block = '1 7'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '8.3441E-4 3.2776E-4'
    sigma_t = '2.411E-1 4.172E-1'
    sigma_s = '2.33644E-1 0.0 3.598E-3 4.07004E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]

  [./Mat4]
    type = ConstantNeutronicsMaterial
    block = '4'
```

```
    fissile = true
    chi = '1 0'
    nu_sigma_f = '7.4518E-3 1.10612E-2'
    sigma_t = '1.849E-1 3.668E-1'
    sigma_s = '1.77711E-1 0.0 2.085E-3 3.53721E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'

  [../]
  [./Mat2]
    type = ConstantNeutronicsMaterial
    block = '2'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '7.4518E-3 1.10612E-2'
    sigma_t = '1.849E-1 3.668E-1'
    sigma_s = '1.77711E-1 0.0 2.085E-3 3.53721E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]
  [./Mat6]
    type = ConstantNeutronicsMaterial
    block = '6'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '7.4518E-3 1.10612E-2'
    sigma_t = '1.849E-1 3.668E-1'
    sigma_s = '1.77711E-1 0.0 2.085E-3 3.53721E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]
  [./Mat3]
    type = ConstantNeutronicsMaterial
    block = '3 5'
    fissile = false
    sigma_t = '9.432E-2 1.8762E-1'
    sigma_s = '8.571E-2 0.0 1.7168E-3 1.7131E-1'
  [../]
[]

[Executioner]
  type = NonlinearEigen
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
[]

[Postprocessors]
  [./Flux1]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g0_L0_M0
    execute_on = 'linear nonlinear timestep_end'
  [../]
  [./Flux2]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g1_L0_M0
```

```
    execute_on = 'linear nonlinear timestep_end'
  [../]
  [./TotalFlx_Fun_PP]
    type = FunctionValuePostprocessor
    function = TotalFlx_function
    execute_on = 'linear nonlinear timestep_end'
  [../]
[]

[Outputs]
  execute_on = 'timestep_end'
  file_base = SS_out
  exodus = true
  csv = true
[]
```

The expected values to be obtained from the eigenvalue calculation are: eigenvalue = 0.999739, Flux1 = 1.345E+02, Flux2 = 2.280E+01 and TotalFlux = 1.573E+02. You may also modify the code to change the cross-sections for zones 2 and 6 to determine the change in the eigenvalue when the transient occurs. Zone 2 is increased by 5% and zone 6 is decreased by 5%. The result for the eigenvalue is about 1.000854.

3.3 Benchmark 16A1-1 (Transient Problem)

The transient problem for benchmark 16A1-1 was defined in 3.2. To setup this problem we start by using the same code given in 3.2.8 with the additional coding for post-processors and the function. We will need two files to perform the transient. The first has the code from 3.2.8 for the steady-state eigenvalue and second one is for the transient, both files should end with the ".i" extension. Go ahead and copy the code from the eigenvalue calculation into both files. We will now modify the code to have it perform the transient.

3.3.1 Changes from Eigenvalue to Transient File

First, change the equation type in the TransportSystems block to equal transient.

```
[TransportSystems]
# ...
  equation_type = transient
# ...
[]
```

Next, change the all the cross-sections in the materials block. If you do not have the materials for zones 2, 4 and 6 separated do so now in the transient file. The eigenvalue file can keep them together. Increase all the cross-sections for zone 2 by 5% and decrease all the cross-sections for zone 6 by 5%. This is simulating a rod movement. The code for the material changes is below.

```
 [Materials]
 # ...
 # ...
  [./Mat4]
    type = ConstantNeutronicsMaterial
    block = '4'
    fissile = true
    chi = '1 0'
```

```
    nu_sigma_f = '7.4518E-3 1.10612E-2'
    sigma_t = '1.849E-1 3.668E-1'
    sigma_s = '1.77711E-1 0.0 2.085E-3 3.53721E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'

  [../]
  [./Mat2]
    type = ConstantNeutronicsMaterial
    block = '2'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '7.82439E-3 1.161426E-2'
    sigma_t = '1.94145E-1 3.8514E-1'
    sigma_s = '1.8659655E-1 0.0 2.18925E-3 3.7140705E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]
  [./Mat6]
    type = ConstantNeutronicsMaterial
    block = '6'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '7.07921E-3 1.050814E-2'
    sigma_t = '1.75655E-1 3.4846E-1'
    sigma_s = '1.6882545E-1 0.0 1.98075E-3 3.3603495E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]
  # ...
  # ...
  []
```

Now, we will need to add a post-processor to become the normalizing flux value. The execution for the post-processor using the function also changed. This will be explained later. You will notice that the Normalization-Flux is type "Receiver" and does nothing else. The point here is to catch the value from the `TotalFlx_Fun_PP` in the eigenvalue calculation.

```
[Postprocessors]
# ...
# ...
  [./TotalFlx_Fun_PP]
    type = FunctionValuePostprocessor
    function = TotalFlx_function
    execute_on = 'initial timestep_end'
  [../]
  [./NormalizationFlux]
    type = Receiver
    #outputs = none
    execute_on = 'initial'
  [../]
# ...
# ...
[]
```

Next, modify the function to be divided by the normalization flux.

```
[Functions]
 [./TotalFlx_function]
   type = ParsedFunction
   value = '(F1 + F2)/Norm'
   vars = 'F1 F2 Norm'
   vals = 'Flux1 Flux2 NormalizationFlux'
 [../]
[]
```

The Executioner block now needs to be modified by changing the type to transient and adding start and end times for the calculation. More advanced features like tolerances and a time step function were added because of issues associated with this benchmark. The `l_tol =` is the tolerance for linear solves. The option `nl_rel_tol` is the tolerance for the nonlinear solves. The `timestep_tolerance` is added because we will demand the data to be output at only certain times. An issue arrises when the previous time step is very close to, but not exactly, on the demanded time. An unrealistic dt timestep is attempted which can cause the program to fail. The `timestep_tolerance` will accept the present time step as the demanded time if within this tolerance. The `FunctionDT` sub-block creates a function for the dt value. This particular problem demands a very small time step at first then increases. This function follows a piece-wise linear formate. Where the `time_dt` vector is the list of dt values, to interpolate between and `time_t` is the associated times.

```
[Executioner]
  type = Transient
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'

  start_time = 0.0
  end_time = 10

  l_tol = 1e-8
  nl_rel_tol = 1e-6
  timestep_tolerance = 1E-5

 [./TimeStepper]
    type = FunctionDT
    time_dt = '2E-8 1E-6 1E-5 1E-4 1E-3'
    time_t = '0 1E-3 1E-1 1 10'
 [../]
[]
#
#
#
[Outputs]
  execute_on = 'timestep_end'
  file_base = Tr_out
 [./csv]
   type = CSV
   sync_only = true
   sync_times = '0 2e-8 1e-7 1e-6 0.00001 0.00002 0.00004 0.00006 0.00008
                 0.0001 0.0002 0.0004 0.0006 0.0008 0.001 0.002 0.004 0.006 0.008
                 0.01 0.02 0.04 0.06 0.08 0.1 0.2 0.4 0.6 0.8 1 2 4 6 8 10'
 [../]
 [./exedus]
   type = Exodus
```

```
   sync_only = true
   sync_times = '0 2e-8 1e-7 1e-6 0.00001 0.00002 0.00004 0.00006 0.00008
                0.0001 0.0002 0.0004 0.0006 0.0008 0.001 0.002 0.004 0.006 0.008
                0.01 0.02 0.04 0.06 0.08 0.1 0.2 0.4 0.6 0.8 1 2 4 6 8 10'
 [../]
[]
```

The Outputs sub-block was also modified by adding a sub-block for csv and Exedus files to specify only certain points in time to record values instead of every time-step. To view the flux profile over time you will need to look at the exodus file using Paraview. Specific times are chosen for the exodus file so that it does not take up a lot of hard drive space.

3.3.2 MultiApp and Transfers

To be able to transfer information between the eigenvalue problem and transient problem we need to setup a MultiApp and Transfers branch. The code is provided below and should be added to the transient file. The only modification needed is to use your filename for the eigenvalue file in the MultiApps block.

```
[MultiApps]
 [./initial_solve]
   type = FullSolveMultiApp
   app_type = RattlesnakeApp
   execute_on = initial
   positions = '0 0 0'
   input_files = Eigenvalue_Filename.i
 [../]
[]

[Transfers]
 [./copy_solution]
   type = MultiAppSystemCopyTransfer
   direction = from_multiapp
   multi_app = initial_solve
   execute_on = initial
   scale_with_keff = false
 [../]
 [./Copy_pp_Flux0]
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
   reduction_type = minimum
   from_postprocessor = Flux1
   to_postprocessor = Flux1
   multi_app = initial_solve
   execute_on = initial
 [../]
 [./Copy_pp_Flux1]
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
   reduction_type = minimum
   from_postprocessor = Flux2
   to_postprocessor = Flux2
   multi_app = initial_solve
   execute_on = initial
 [../]
 [./Copy_NormFlux_PP]
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
```

```
   reduction_type = maximum
   from_postprocessor = TotalFlx_Fun_PP
   to_postprocessor = NormalizationFlux
   multi_app = initial_solve
   execute_on = initial
 [../]
[]
```

In the code the first branch is MultiApps. This is a branch that allows several other programs to be executed. For this problem, we are using the FullSolveMultiApp option and using the Rattlesnake application. If Rattlesnake is just a sub-module to a larger application the larger application's name would be placed here. The input file name for the eigenvalue calculation also needs to be specified by the user.

The next branch is Transfers. The first sub-branch is a MultiAppSystemCopyTransfer. This sub-branch will copy all the variables from the eigenvalue calculation to the transient model. The following sub-branches are all post-processor transfers. You will note that while the post-processors and variables may have the same names but they are not joined in the calculation until we use the Transfers block. With the system copy transfer all variables defined in the transient must also be defined in the eigenvalue calculation. If they are not defined in the eigenvalue file you will get a PETSc error saying that the vector local lengths are not equal. If you have a variable in a transient file that you need, but it is not needed in the eigenvalue calculation, you can simply define it but never perform any calculations with it.

The post-processor transfer, unlike the total system copy, allows us to specify which post-processor value from the eigenvalue calculation gets assigned to which post-processor in the transient calculation. You will notice that `TotalFlx_Fun_PP` post-processor in the transient calculation never gets a value from the eigenvalue calculation. We want this because it is now normalized and we don't want the first value to be un-normalized. That is also why the execution was set to initial for `TotalFlx_Fun_PP` in the transient calculation The NormalizationFlux however gets the value of the `TotalFlx_Fun_PP` post-processor in the eigenvalue calculation. There is a required option `reduction_type` which takes the possible inputs of "average, sum, maximum, and minimum". For our case the post-processors are single values and the choice does not matter.

3.3.3 Transient Code and Results

The entire transient code is found below. Make sure to change the name for the eigenvalue file (make sure there are no spaces in the name).

```
[Mesh]
 type = CartesianMesh
 dim = 1
 dx = '40 47.374 9 34 9 47.374 40'
 ix = '20 24 5 16 5 24 20'
 subdomain_id = '1 2 3 4 5 6 7'
 uniform_refine = 0
[]

[GlobalParams]
    neutron_speed = '540248514.31659 91911764.705882'
[]

[TransportSystems]
  particle = neutron
  equation_type = transient
  G = 2
  VacuumBoundary ='1 2'
  [./SN]
    scheme = SAAF-CFEM-SN
```

```
    family = LAGRANGE
    order = FIRST
    AQtype = Gauss-Chebyshev
    NPolar = 8
    n_delay_groups = 6
    fission_source_as_material = true
  [../]
[]


[Functions]
 [./TotalFlx_function]
   type = ParsedFunction
   value = '(F1 + F2)/Norm'
   vars = 'F1 F2 Norm'
   vals = 'Flux1 Flux2 NormalizationFlux'
 [../]
[]
[Materials]
  [./Mat1]
    type = ConstantNeutronicsMaterial
    block = '1 7'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '8.3441E-4 3.2776E-4'
    sigma_t = '2.411E-1 4.172E-1'
    sigma_s = '2.33644E-1 0.0 3.598E-3 4.07004E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]

  [./Mat4]
    type = ConstantNeutronicsMaterial
    block = '4'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '7.4518E-3 1.10612E-2'
    sigma_t = '1.849E-1 3.668E-1'
    sigma_s = '1.77711E-1 0.0 2.085E-3 3.53721E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'

  [../]
  [./Mat2]
    type = ConstantNeutronicsMaterial
    block = '2'
    fissile = true
    chi = '1 0'
    nu_sigma_f = '7.82439E-3 1.161426E-2'
    sigma_t = '1.94145E-1 3.8514E-1'
    sigma_s = '1.8659655E-1 0.0 2.18925E-3 3.7140705E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]
  [./Mat6]
    type = ConstantNeutronicsMaterial
    block = '6'
    fissile = true
```

```
    chi = '1 0'
    nu_sigma_f = '7.07921E-3 1.050814E-2'
    sigma_t = '1.75655E-1 3.4846E-1'
    sigma_s = '1.6882545E-1 0.0 1.98075E-3 3.3603495E-1'
    decay_constant = '0.0129 0.0311 0.134 0.331 1.26 3.21'
    delay_fraction = '0.81E-4 6.87E-4 6.12E-4 11.38E-4 5.12E-4 1.70E-4'
    delay_spectrum = '1 0 1 0 1 0 1 0 1 0 1 0'
  [../]
  [./Mat3]
    type = ConstantNeutronicsMaterial
    block = '3 5'
    fissile = false
    sigma_t = '9.432E-2 1.8762E-1'
    sigma_s = '8.571E-2 0.0 1.7168E-3 1.7131E-1'
  [../]
[]

[Postprocessors]
  [./Flux1]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g0_L0_M0
    execute_on = 'initial linear nonlinear timestep_end'
  [../]
  [./Flux2]
    type = ElementIntegralVariablePostprocessor
    block = '1 2 3 4 5 6 7'
    variable = flux_moment_g1_L0_M0
    execute_on = 'initial linear nonlinear timestep_end'
  [../]
  [./TotalFlx_Fun_PP]
    type = FunctionValuePostprocessor
    function = TotalFlx_function
    execute_on = 'initial timestep_end'
  [../]
  [./NormalizationFlux]
    type = Receiver
    #outputs = none
    execute_on = 'initial'
  [../]
[]

[MultiApps]
 [./initial_solve]
   type = FullSolveMultiApp
   app_type = RattlesnakeApp
   execute_on = initial
   positions = '0 0 0'
   input_files = SS-Filename.i
 [../]
[]

[Transfers]
 [./copy_solution]
   type = MultiAppSystemCopyTransfer
   direction = from_multiapp
   multi_app = initial_solve
   execute_on = initial
   scale_with_keff = false
 [../]
 [./Copy_pp_Flux0]
```

```
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
   reduction_type = minimum
   from_postprocessor = Flux1
   to_postprocessor = Flux1
   multi_app = initial_solve
   execute_on = initial
 [../]
 [./Copy_pp_Flux1]
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
   reduction_type = minimum
   from_postprocessor = Flux2
   to_postprocessor = Flux2
   multi_app = initial_solve
   execute_on = initial
 [../]
 [./Copy_NormFlux_PP]
   type = MultiAppPostprocessorTransfer
   direction = from_multiapp
   reduction_type = maximum
   from_postprocessor = TotalFlx_Fun_PP
   to_postprocessor = NormalizationFlux
   multi_app = initial_solve
   execute_on = initial
 [../]
[]

[Executioner]
  type = Transient
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
  start_time = 0.0
  end_time = 10
  l_tol = 1e-8
  nl_rel_tol = 1e-6
  timestep_tolerance = 1E-5
 [./TimeStepper]
    type = FunctionDT
    time_dt = '2E-8 1E-6 1E-5 1E-4 1E-3'
    time_t = '0 1E-3 1E-1 1 10'
 [../]
[]

[Outputs]
  execute_on = 'timestep_end'
  file_base = Tr_out

 [./csv]
   type = CSV
   sync_only = true
   sync_times = '0 2e-8 1e-7 1e-6 0.00001 0.00002 0.00004 0.00006 0.00008
                 0.0001 0.0002 0.0004 0.0006 0.0008 0.001 0.002 0.004 0.006 0.008
                 0.01 0.02 0.04 0.06 0.08 0.1 0.2 0.4 0.6 0.8 1 2 4 6 8 10'
 [../]
 [./exedus]
   type = Exodus
   sync_only = true
   sync_times = '0 2e-8 1e-7 1e-6 0.00001 0.00002 0.00004 0.00006 0.00008
                 0.0001 0.0002 0.0004 0.0006 0.0008 0.001 0.002 0.004 0.006 0.008
```

```
                0.01 0.02 0.04 0.06 0.08 0.1 0.2 0.4 0.6 0.8 1 2 4 6 8 10'
 [../]

[]
```

Figure 3 below, shows the normalized power history (ie. Normalized flux) from the benchmark and Figure 4 shows the flux profile at 0.0, 0.01 and 1.0 sec, normalized to 1 neutron/sec which is the same as dividing by the eigenvalue. The flux profile in Rattlesnake is automatically normalized. To compare the flux profile start by opening the exodus file in Paraview. Enable the variables on the `vlux_moment_g0_L0_M0` and `vlux_moment_g1_L0_M0` as shown in Figure 5. Then click on the "Plot over line" button which is shown in Figure 6. A graph with the scalar flux for group 0 and 1 should appear. Next, right-click in the blank space on the toolbar and click on "Time Inspector" and also "Animation View". The screen should look like Figure 7. You can now step through time to view the scalar flux profiles and compare with the benchmark.

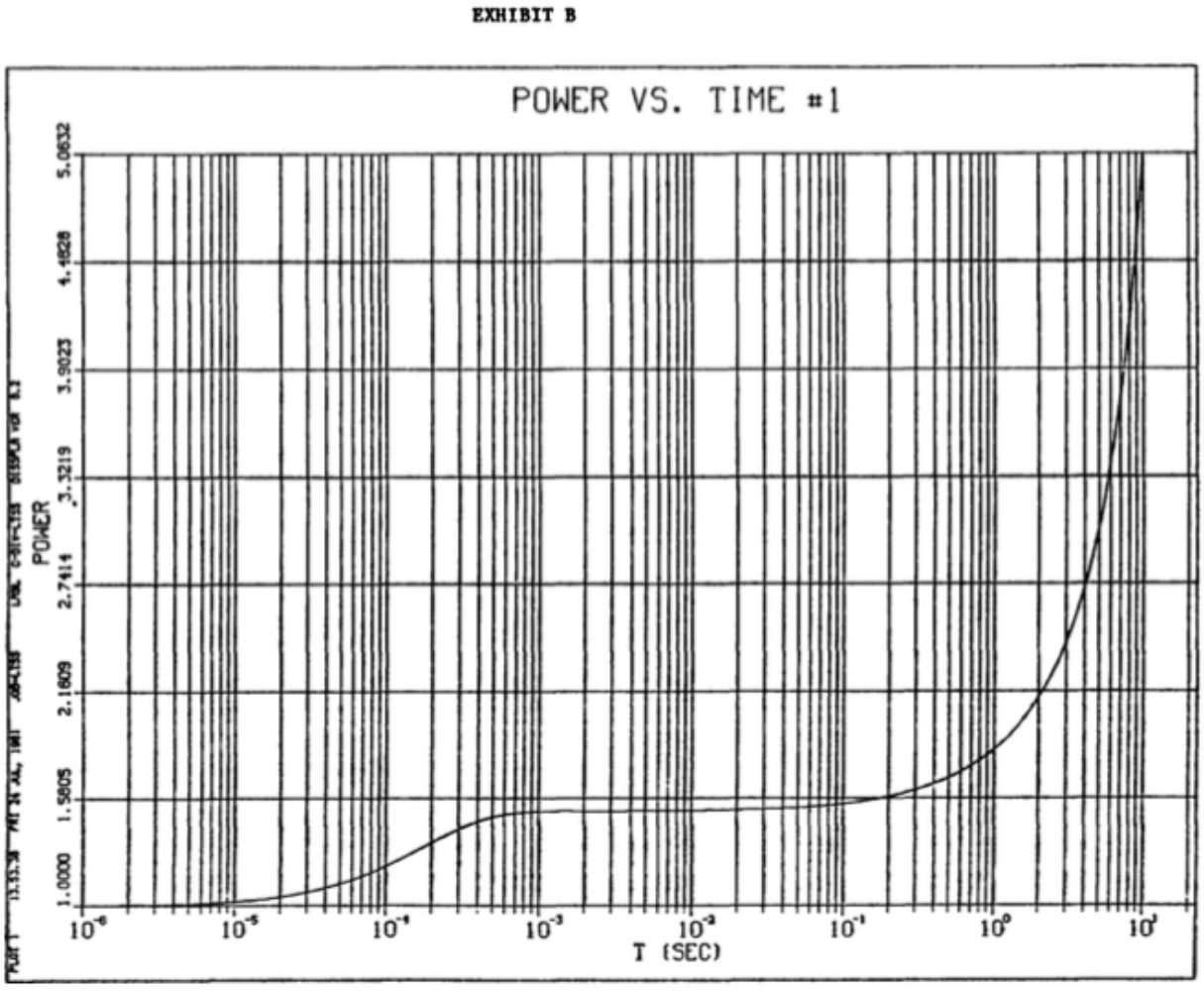


Figure 3 Normalized Power History (Benchmark 16A1-1)

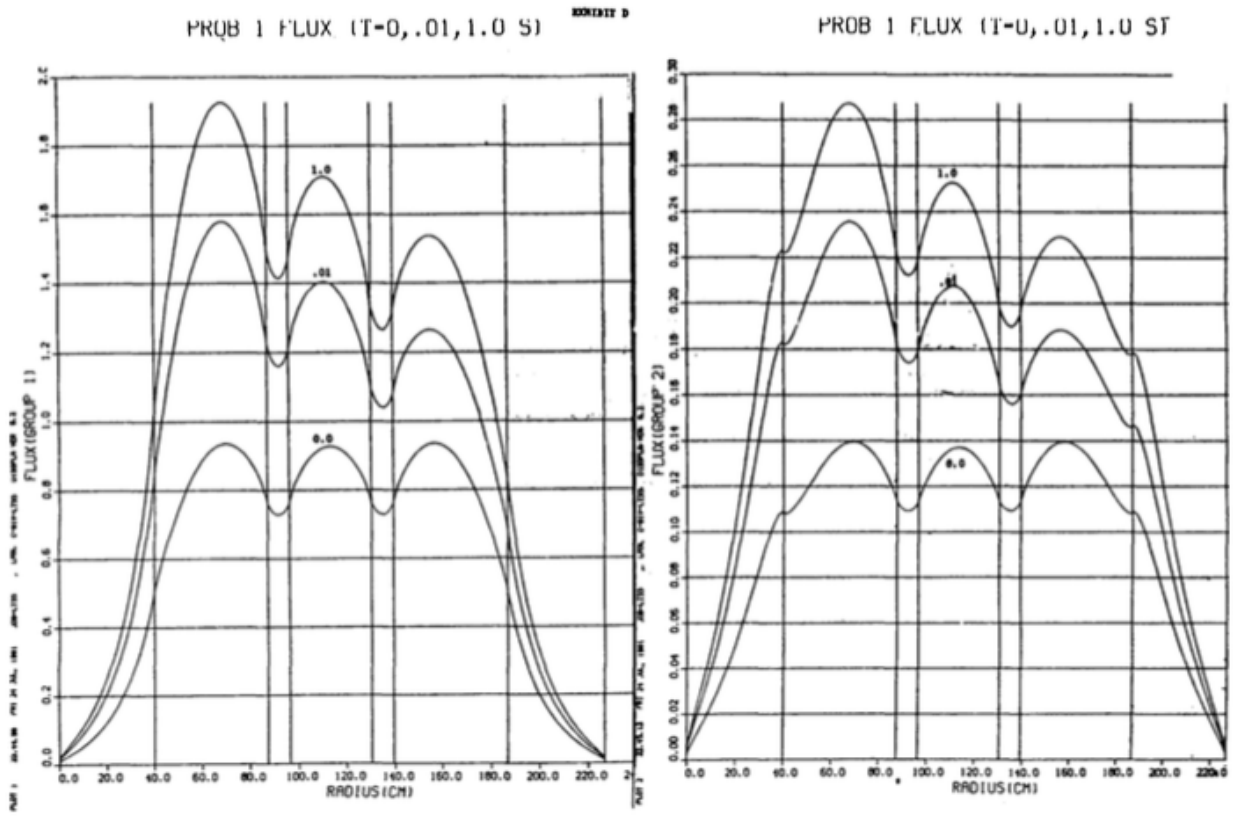


Figure 4 Flux Profile at 0.0, 0.01 and 1.0 sec (Benchmark 16A1-1)

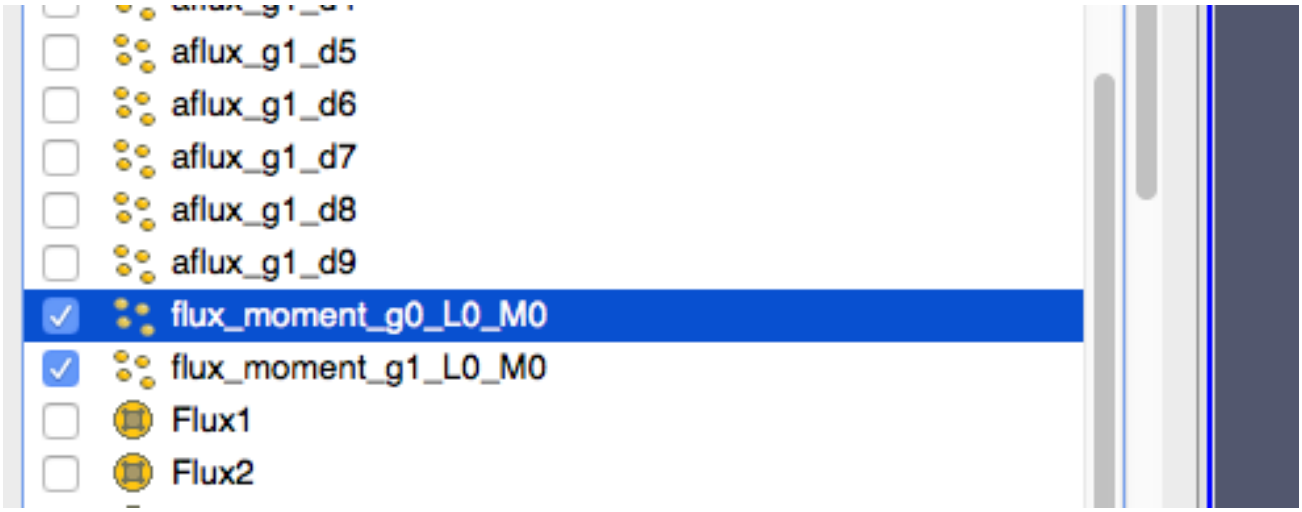


Figure 5 Enable scalar fluxes

Figure 6 Plot over line button

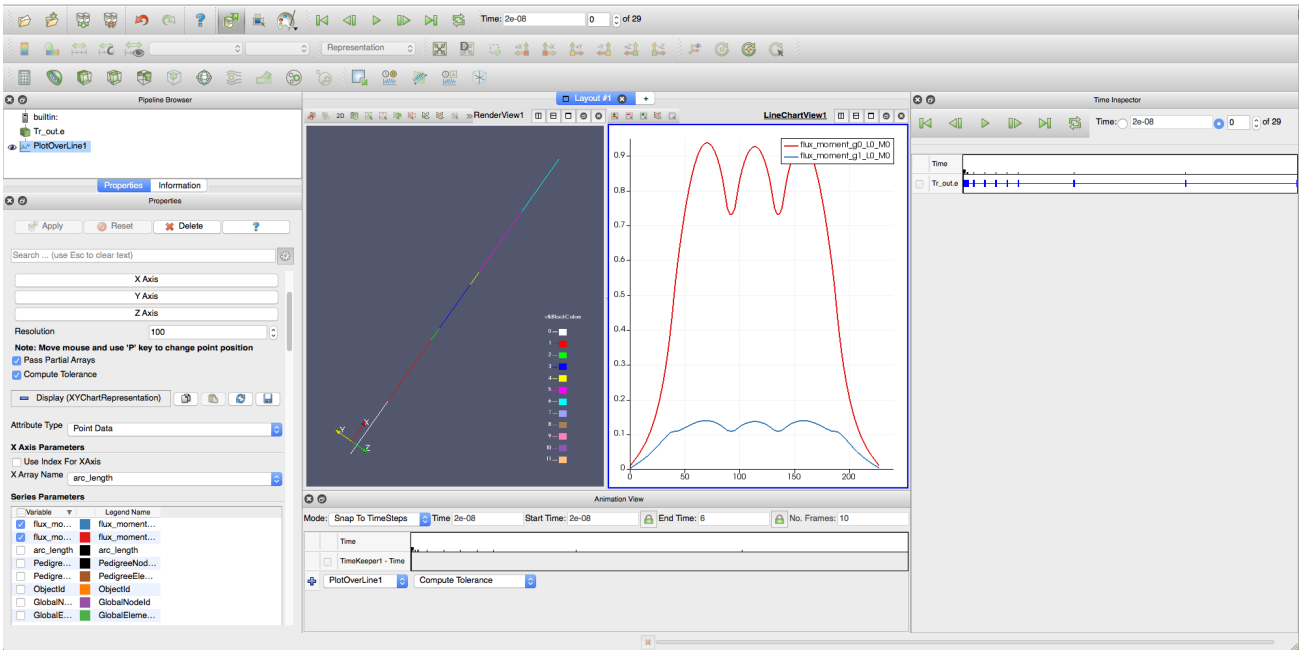


Figure 7 View scalar fluxes over time

3.4 Takeda benchmark Mode 4

This is a benchmark with hexagonal geometry. Periodic boundary condition will also be demonstrated. (To be added after HexagonalMesh is completed.)

3.5 LRA benchmark (14-A1)

This is a two-group transient benchmark with temperature feedback. All files necessary for running this tutorials with Rattlesnake are under *'rattlesnake/tutorials/LRA2D'* folder.

3.5.1 Problem description

It is a two-dimensional two-group neutron diffusion problem with adiabatic heat-up and Doppler feedback in thermal reactor. It is a super prompt-critical transient. To have better understanding on the cross sections given later, we present the equations here:

$$-\frac{1}{v_1}\frac{\partial \phi_1}{\partial t} = -\vec{\nabla} D_1 \vec{\nabla}\phi_1 + (\Sigma_{a,1} + \Sigma_{s,1\to 2})\phi_1(\vec{r},t) - \nu(1-\beta)f(\vec{r},t) - \sum_{i=1}^{2}\lambda_i C_i(\vec{r},t), \tag{1}$$

$$-\frac{1}{v_2}\frac{\partial \phi_2}{\partial t} = -\vec{\nabla} D_2 \vec{\nabla}\phi_1 + \Sigma_{a,2}\phi_2(\vec{r},t) - \Sigma_{s,1\to 2}\phi_1, \tag{2}$$

$$f(\vec{r},t) = \sum_{g=1}^{2}\Sigma_{f,g}\phi_g, \tag{3}$$

$$\frac{\partial C_i}{\partial t} = \nu\beta_i f - \lambda_i C_i(\vec{r},t), i = 1,2, \tag{4}$$

$$\frac{\partial T(\vec{r},t)}{\partial t} = \alpha f, \tag{5}$$

$$\Sigma_{a,1}(\vec{r},t) = \Sigma_{a,1}(\vec{r},t=0)\left[1+\gamma\left(\sqrt{T}-\sqrt{T_0}\right)\right], \tag{6}$$

$$P(\vec{r},t) = \kappa f, \tag{7}$$

where ϕ_1, ϕ_2 are the fast and thermal fluxes; v_1, v_2 are the averaged neutron velocities; $\Sigma_{a,1}, \Sigma_{a,2}$ are the absorption cross sections; $\Sigma_{s,1\to2}$ is the fast-to-thermal scattering cross section; $\Sigma_{f,1}, \Sigma_{f,2}$ are the fission cross sections; ν is the averaged number of neutrons emitted per fission; β_1, β_2 are the delayed neutron precursor fractions and $\beta = \beta_1 + \beta_2$; C_1, C_2 are the delayed neutron precursor concentrations; λ_1, λ_2 are the decay constants of the delayed neutron precursors; f is the fission reaction rate; P is the power density; T is the temperature; κ is the averaged power released per fission; α is the combination of κ and the specific heat capacity; γ is the Doppler feedback coefficient; $T_0 = T(\vec{r}, t = 0)$. The two-group diffusion equation are solved with zero flux boundary conditions on external surfaces, reflecting conditions at symmetry boundaries and steady state initial conditions which are obtained by solving

$$-\vec{\nabla} D_1 \vec{\nabla} \phi_1 + (\Sigma_{a,1} + \Sigma_{s,1\to2})\phi_1(\vec{r}, t) = \frac{1}{k} \sum_{g=1}^{2} \nu \Sigma_{f,g} \phi_g, \tag{8}$$

$$-\vec{\nabla} D_2 \vec{\nabla} \phi_1 + \Sigma_{a,2} \phi_2(\vec{r}, t) = \Sigma_{s,1\to2} \phi_1. \tag{9}$$

$$\tag{10}$$

The eigenvalue k is used to modify the fission cross section for the transient simulations with $\frac{1}{k}\Sigma_{f,g}, g = 1, 2$. The initial flux distribution shall be normalized such that the averaged power density

$$\bar{P} \equiv \frac{\int_{V_{core}} P(\vec{r}, t = 0) d\vec{r}}{\int_{V_{core}} d\vec{r}}, \tag{11}$$

where V_{core} is the core region with fuels, is equal to $10^{-6} W \cdot cm^{-3}$. The initial precursor concentrations are in equilibrium with the initial critical flux distribution.

The geometry is illustrated in Fig. 8.

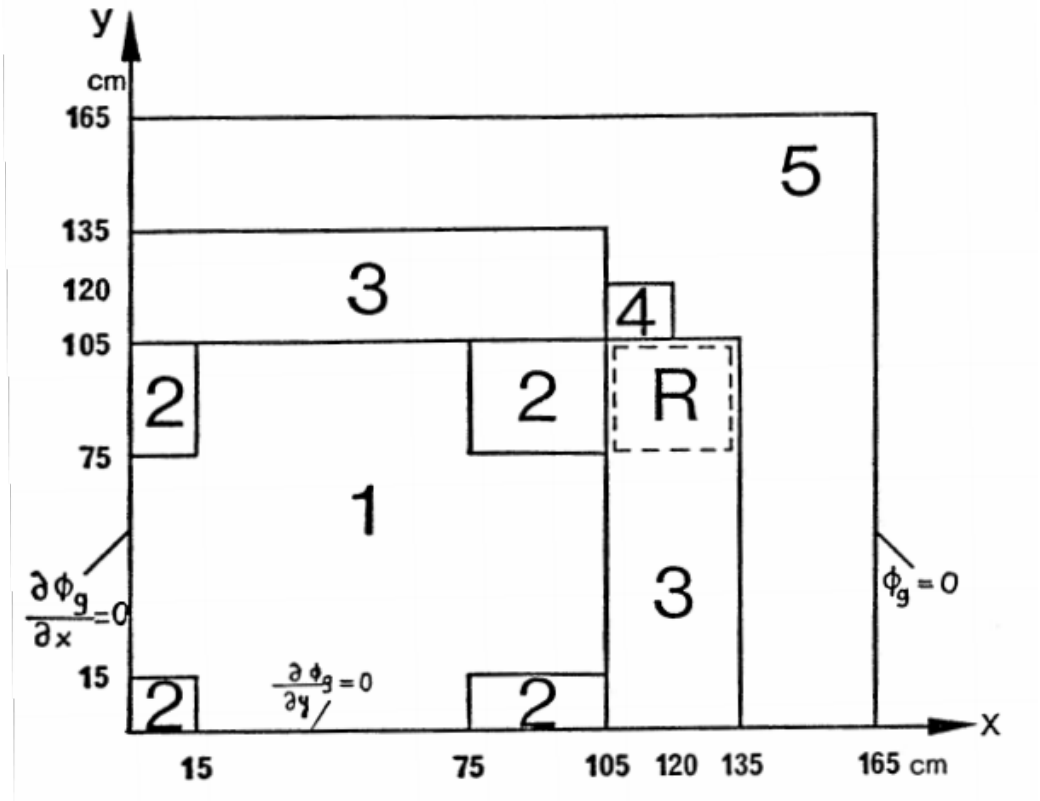


Figure 8 LRA benchmark geometry with region assignment.

Initial two-group constants are presented in Table 7. ν is equal to 2.43. Axial bulking $B^2 = 10^{-4}$ is applied for

both energy groups. Delayed neutron data are presented in Table 8. All fuel materials have the same delayed neutron data. Some scalar data are listed in Table 9.

Table 7 LRA benchmark initial two-group constants.

Region	Material	Group g	D_g (cm)	$\Sigma_{a,g}$ (cm^{-1})	$\nu\Sigma_{f,g}$ (cm^{-1})	$\Sigma_{s,1\to2}$ (cm^{-1})	χ_g	v_g $(cm \cdot s^{-1})$
1	Fuel 1 with rod	1 2	1.255 0.211	0.008252 0.1003	0.004602 0.1091	0.02533	1 0	3.0×10^7 3.0×10^5
2	Fuel 1 without rod	1 2	1.268 0.1902	0.007181 0.07047	0.004609 0.08675	0.02767	1 0	3.0×10^7 3.0×10^5
3	Fuel 2 with rod	1 2	1.259 0.2091	0.008002 0.08344	0.004663 0.1021	0.02617	1 0	3.0×10^7 3.0×10^5
4	Fuel 2 without rod	1 2	1.259 0.2091	0.008002 0.073324	0.004663 0.1021	0.02617	1 0	3.0×10^7 3.0×10^5
5	Reflector	1 2	1.257 0.1592	0.0006034 0.01911	- -	0.04754	- -	3.0×10^7 3.0×10^5

Table 8 LRA benchmark delayed neutron data.

Group i	β_i	λ_i (s^{-1})	$\chi_{d,i,1}$	$\chi_{d,i,2}$
1	0.0054	0.0654	1	0
2	0.001087	1.35	1	0

Table 9 LRA benchmark scalar values.

Meaning	Notation	value
Axial buckling for both energy groups	B_g^2	10^{-4} (cm^{-2})
Mean number of neutrons per fission	ν	2.43
Conversion factor	α	3.83×10^{-11} $(K \cdot cm^3)$
Feedback constant	γ	3.034×10^{-3} $(K^{1/2})$
Energy released per fission	κ	3.204×10^{-11} $(J/fission)$
Initial and reference temperature	T_0	300 (K)
Active core volume	V_{core}	17550 (cm^2)

The transient is initiated by changing the thermal absorption cross section as the following:

$$\Sigma_{a,2}(t) = \Sigma_{a,2}(t=0) \begin{cases} 1 - 0.0606184t, & t \leq 2 \\ 0.8787631, & t > 2 \end{cases} \tag{12}$$

where t is time in seconds.

3.5.2 Mesh

The geometry can be meshed with regular grids. And $15cm$ can be used as the maximum element size to match all material boundaries. So we can use the Rattlesnake built-in mesh generator GeneratedBIDMesh to generate the mesh. The mesh block in the input would be:

```
[Mesh]
 type = GeneratedBIDMesh
 dim = 2
 xmin = 0
 xmax = 165
 ymin = 0
 ymax = 165
 elem_type = QUAD4
 nx = 11
 ny = 11
 subdomain='2 1 1 1 1 2 2 3 3 5 5
            1 1 1 1 1 1 1 3 3 5 5
            1 1 1 1 1 1 1 3 3 5 5
            1 1 1 1 1 1 1 3 3 5 5
            1 1 1 1 1 1 1 3 3 5 5
            2 1 1 1 1 2 2 6 6 5 5
            2 1 1 1 1 2 2 6 6 5 5
            3 3 3 3 3 3 3 4 5 5 5
            3 3 3 3 3 3 3 5 5 5 5
            5 5 5 5 5 5 5 5 5 5 5
            5 5 5 5 5 5 5 5 5 5 5'
 uniform_refine = 1
 second_order = false
[]
```

Block 1 to 5 correspond to region 1 to 5 in Fig. 8. Block 6 is the region where control rod are ejected. We leave *uniform_refine* and *second_order* for performing convergence study when desired.

3.5.3 Transport System

We are solving the two-group diffusion problem throughout the geometry. The input block can be

```
[TransportSystems]
 particle = neutron
 equation_type = transient

 G = 2

 DirichletBoundary = 'top right'
 ReflectingBoundary = 'bottom left'

 [./diff]
  scheme = CFEM-Diffusion
  n_delay_groups = 2
  family = LAGRANGE
  order = FIRST
  fission_source_as_material = true
 [../]
[]
```

We set particle as 'neutron' to include fission reactions. We are solving a two-group transient problem. Top and right boundaries are homogneous Dirichlet. Bottom and left boundaries are reflecting. We only have one single diffusion system. The number of delayed neutron precursor groups is two. We want to use the classic linear Lagrange shape functions. We treat fission source as material to have faster residual evaluations.

3.5.4 Materials

Now we need to specify materials for the diffusion equation. We developed two neutronics materials for this benchmark: *BuckledConstantNeutronicsMaterial* and *BuckledFunctionTemperatureMaterial*. Although they are officially in Rattlesnake, we did not list them in Neutronics Materials because they are too specific. These two materials can be used only for diffusion calculations. *BuckledConstantNeutronicsMaterial* is derived from ConstantNeutronicsMaterial. It has one extra parameter 'bz_sqrd' with default value 0 for the axial buckling. The material property of the removal cross section is added by $B_g^2 D_g(\vec{r}, t)$ at every quadrature point at any time. *BuckledFunctionTemperatureMaterial* is derived from FunctionNeutronicsMaterial. It also has one extra parameter 'bz_sqrd' with default value 0 for the axial buckling. It accepts three more parameters 'gamma', 'temperature, 'temp0', which are the Doppler feedback coefficient, temperature variable and the reference temperature. It is more complicated than *BuckledConstantNeutronicsMaterial* because the Doppler feedback has to be applied on the absorption cross section. So this material will first evaluate the absorption cross section at every time step and at every spatial quadrature point. It then applies the Dopper feedback with the updated temperature variable and the axial buckling. Finally it will add back all the out-group scatterings to retrieve the removal cross section.

The material input block looks like

```
[Materials]
  [./fuel1_blade_in]
    type           = BuckledFunctionTemperatureMaterial
    block          = 1

    diffusion_coef = '1.255 0.211'
    sigma_s        = '0.232022 0.0
                      0.02533  1.479479'
    fissile        = true
    nu_sigma_f     = '0.004602 0.1091'
    kappa_sigma_f  = '6.06782222e-14 1.438503704e-12'
    chi            = '1.0 0.0'
    sigma_r        = '0.033582 0.1003'

    bz_sqrd        = 0.0001

    neutron_speed  = '3.e7 3.e5'
    delay_fraction = '0.0054 0.001087'
    decay_constant = '0.0654 1.35'
    delay_spectrum = '1.0 0.0
                      1.0 0.0'

    gamma          = '3.034e-3 0.0'
    temperature    = T
    temp0          = 300

    plus           = true
  [../]

  [./fuel1_blade_out]
    type           = BuckledFunctionTemperatureMaterial
    block          = 2

    diffusion_coef = '1.268 0.1902'
    sigma_s        = '0.22803 0.0
                      0.02767 1.682071'
    fissile        = true
    nu_sigma_f     = '0.004609 0.08675'
    kappa_sigma_f  = '6.07705185e-14 1.14381481e-12'
    chi            = '1.0 0.0'
```

```
  sigma_r        = '0.034851 0.07047'

  bz_sqrd        = 0.0001

  neutron_speed  = '3.e7 3.e5'
  delay_fraction = '0.0054 0.001087'
  decay_constant = '0.0654 1.35'
  delay_spectrum = '1.0 0.0
                    1.0 0.0'

  gamma          = '3.034e-3 0.0'
  temperature    = T
  temp0          = 300

  plus           = true
[../]

[./fuel1_blade_out]
  type           = BuckledFunctionTemperatureMaterial
  block          = 2

  diffusion_coef = '1.268 0.1902'
  sigma_s        = '0.22803 0.0
                    0.02767 1.682071'
  fissile        = true
  nu_sigma_f     = '0.004609 0.08675'
  kappa_sigma_f  = '6.07705185e-14 1.14381481e-12'
  chi            = '1.0 0.0'
  sigma_r        = '0.034851 0.07047'

  bz_sqrd        = 0.0001

  neutron_speed  = '3.e7 3.e5'
  delay_fraction = '0.0054 0.001087'
  decay_constant = '0.0654 1.35'
  delay_spectrum = '1.0 0.0
                    1.0 0.0'

  gamma          = '3.034e-3 0.0'
  temperature    = T
  temp0          = 300

  plus           = true
[../]

[./fuel2_blade_in]
  type           = BuckledFunctionTemperatureMaterial
  block          = 3

  diffusion_coef = '1.259 0.2091'
  sigma_s        = '0.2305884 0.0
                    0.02617   1.5106936'
  fissile        = true
  nu_sigma_f     = '0.004663 0.1021'
  kappa_sigma_f  = '6.14825185e-14 1.346207407e-12'
  chi            = '1.0 0.0'
  sigma_r        = '0.034172 0.08344'

  bz_sqrd        = 0.0001

  neutron_speed  = '3.e7 3.e5'
```

```
  delay_fraction = '0.0054 0.001087'
  decay_constant = '0.0654 1.35'
  delay_spectrum = '1.0 0.0
                    1.0 0.0'

  gamma          = '3.034e-3 0.0'
  temperature    = T
  temp0          = 300

  plus           = true
[../]

[./fuel2_blade_out]
  type           = BuckledFunctionTemperatureMaterial
  block          = 4

  diffusion_coef = '1.259 0.2091'
  sigma_s        = '0.2305884 0.0
                    0.02617   1.5208106'
  fissile        = true
  nu_sigma_f     = '0.004663 0.1021'
  kappa_sigma_f  = '6.14825185e-14 1.346207407e-12'
  chi            = '1.0 0.0'
  sigma_r        = '0.034172 0.073324'

  bz_sqrd        = 0.0001

  neutron_speed  = '3.e7 3.e5'
  delay_fraction = '0.0054 0.001087'
  decay_constant = '0.0654 1.35'
  delay_spectrum = '1.0 0.0
                    1.0 0.0'

  gamma          = '3.034e-3 0.0'
  temperature    = T
  temp0          = 300

  plus           = true
[../]

[./reflector]
  type           = BuckledConstantNeutronicsMaterial
  block          = 5

  diffusion_coef = '1.257 0.1592'
  sigma_s        = '0.2171793 0.0
                    0.04754   2.074692'
  fissile        = false
  sigma_r        = '0.0481434 0.01911'

  bz_sqrd        = 0.0001

  neutron_speed  = '3.e7 3.e5'
[../]

[./fuel2_blade_in_r]
  type           = BuckledFunctionTemperatureMaterial
  block          = 6

  diffusion_coef = '1.259 0.2091'
  sigma_s        = '0.2305884 0.0
```

```
                    0.02617   1.5106936'
    fissile        = true
    nu_sigma_f     = '0.004663 0.1021'
    kappa_sigma_f  = '6.14825185e-14 1.346207407e-12'
    chi            = '1.0 0.0'
    sigma_r        = '0.034172 move_blade'

    bz_sqrd        = 0.0001

    neutron_speed  = '3.e7 3.e5'
    delay_fraction = '0.0054 0.001087'
    decay_constant = '0.0654 1.35'
    delay_spectrum = '1.0 0.0
                      1.0 0.0'

    gamma          = '3.034e-3 0.0'
    temperature    = T
    temp0          = 300

    plus           = true
  [../]
[]
```

Materials, fuel1_blade_in, fuel1_blade_out, fuel2_blade_in, fuel2_blade_out, reflector, fuel2_blade_in_r are for the blocks 1, 2, 3, 4, 5, and 6 respectively as in the subdomain_id of the mesh input block. Parameters *type*, *block*, *diffusion_coef*, *sigma_s*, *fissile*, *nu_sigma_f*, *chi*, *kappa_sigma_f*, *sigma_r*, *neutron_speed*, *decay_constant*, *delay_fraction*, *delay_spectrum*, and *plus* of *BuckledFunctionTemperatureMaterial* can be found in FunctionNeutronicsMaterial. We turned *plus* to true because we want to use kappa fission cross section later for evaluating the power density. Same parameters of *BuckledConstantNeutronicsMaterial* can be found in ConstantNeutronicsMaterial. It is noted that the in-group scattering cross section is not needed for diffusion calculations. They are there for consistency. The benchmark specification does not give their values. They are evaluated with $1/3/D_g - \Sigma_{r,g}$, where $\Sigma_{r,g}$ is the removal cross section.

'bz_sqrd' is constant 0.0001 for all materials. The temperature variable is T, which will be discussed later, is set for all *BuckledFunctionTemperatureMaterial* materials. The same 'temp0' and 'gamma' are also set for all *BuckledFunctionTemperatureMaterial* materials. It is noted that the thermal gamma is set to zero, which means there is no Doppler feedback for thermal absorption.

The only function in the *BuckledFunctionTemperatureMaterial* materials is *move_blade* in material *fuel2_blade_in_r*. So it is convenient to give the function in the input here:

```
[Functions]
 [./move_blade]
   type = SlopeFunction
   timep = '0.0     2.0            3.0'
   value = '0.08344 0.073323993064 0.073323993064'
 [../]
[]
```

Parameters for *SlopeFunction* can be found in SlopeFunction.

We notice that there are lots of duplicated parameters in those materials. Rattlesnake provides a way to simplify the input. Users can add an input block *GlobalParams* in their input. It contains parameters, that can be substituted into any input blocks or sub input blocks in the rest of the input file when the blocks have them as the valid parameters and do not provide them. In this case, we can extract several parameters into *GlobalParams*, which are used by all materials:

```
[GlobalParams]
    bz_sqrd        = 0.0001

    gamma          = '3.034e-3 0.0'
    temperature    = T
    temp0          = 300

    plus           = true

    neutron_speed  = '3.e7 3.e5'
[]
[Materials]
  [./fuel1_blade_in]
    type           = BuckledFunctionTemperatureMaterial
    block          = 1

    diffusion_coef = '1.255 0.211'
    sigma_s        = '0.232022 0.0
                      0.02533  1.479479'
    fissile        = true
    nu_sigma_f     = '0.004602 0.1091'
    kappa_sigma_f  = '6.06782222e-14 1.438503704e-12'
    chi            = '1.0 0.0'
    sigma_r        = '0.033582 0.1003'

    delay_fraction = '0.0054 0.001087'
    decay_constant = '0.0654 1.35'
    delay_spectrum = '1.0 0.0
                      1.0 0.0'
  [../]

  [./fuel1_blade_out]
    type           = BuckledFunctionTemperatureMaterial
    block          = 2

    diffusion_coef = '1.268 0.1902'
    sigma_s        = '0.22803 0.0
                      0.02767 1.682071'
    fissile        = true
    nu_sigma_f     = '0.004609 0.08675'
    kappa_sigma_f  = '6.07705185e-14 1.14381481e-12'
    chi            = '1.0 0.0'
    sigma_r        = '0.034851 0.07047'

    delay_fraction = '0.0054 0.001087'
    decay_constant = '0.0654 1.35'
    delay_spectrum = '1.0 0.0
                      1.0 0.0'
  [../]

  [./fuel2_blade_in]
    type           = BuckledFunctionTemperatureMaterial
    block          = 3

    diffusion_coef = '1.259 0.2091'
    sigma_s        = '0.2305884 0.0
                      0.02617   1.5106936'
    fissile        = true
    nu_sigma_f     = '0.004663 0.1021'
    kappa_sigma_f  = '6.14825185e-14 1.346207407e-12'
    chi            = '1.0 0.0'
```

```
    sigma_r        = '0.034172 0.08344'

    delay_fraction = '0.0054 0.001087'
    decay_constant = '0.0654 1.35'
    delay_spectrum = '1.0 0.0
                      1.0 0.0'
  [../]

  [./fuel2_blade_out]
    type           = BuckledFunctionTemperatureMaterial
    block          = 4

    diffusion_coef = '1.259 0.2091'
    sigma_s        = '0.2305884 0.0
                      0.02617   1.5208106'
    fissile        = true
    nu_sigma_f     = '0.004663 0.1021'
    kappa_sigma_f  = '6.14825185e-14 1.346207407e-12'
    chi            = '1.0 0.0'
    sigma_r        = '0.034172 0.073324'

    delay_fraction = '0.0054 0.001087'
    decay_constant = '0.0654 1.35'
    delay_spectrum = '1.0 0.0
                      1.0 0.0'
  [../]

  [./reflector]
    type           = BuckledConstantNeutronicsMaterial
    block          = 5

    diffusion_coef = '1.257 0.1592'
    sigma_s        = '0.2171793 0.0
                      0.04754   2.074692'
    fissile        = false
    sigma_r        = '0.0481434 0.01911'
  [../]

  [./fuel2_blade_in_r]
    type           = BuckledFunctionTemperatureMaterial
    block          = 6

    diffusion_coef = '1.259 0.2091'
    sigma_s        = '0.2305884 0.0
                      0.02617   1.5106936'
    fissile        = true
    nu_sigma_f     = '0.004663 0.1021'
    kappa_sigma_f  = '6.14825185e-14 1.346207407e-12'
    chi            = '1.0 0.0'
    sigma_r        = '0.034172 move_blade'

    delay_fraction = '0.0054 0.001087'
    decay_constant = '0.0654 1.35'
    delay_spectrum = '1.0 0.0
                      1.0 0.0'
  [../]
[]
```

We do not put delayed neutron data into *GlobalParams* because the non-fissile material 'reflector' does not have them but will try to use them if they are in *GlobalParams*, which will result into a syntax error.

3.5.5 Initial conditions on scalar fluxes

We now need to provide the initial condition for the scalar fluxes. Because fission source and delayed neutron precursor concentrations are treated as material properties. Their values are evaluated on the fly based on the scalar flux, so their initial condition is automatically taken care of. To set the initial condition for the scalar fluxes, we use the same method, i.e. MultiApp and Transfer, like the one in 16A1 benchmark. Basically, we will need to use the same mesh and the same materials to set up an input for solving the initial eigenvalue problem. We let the problem run on *initial* and then transfer the scalar fluxes over to the transient problem. The input for adding the initial eigenvalue problem and transfer are as the following:

```
[MultiApps]
 [./initial_solve]
   type = FullSolveMultiApp
   execute_on = initial
   input_files = lra_trans_initial.i
 [../]
[]

[Transfers]
 [./copy_solution]
   type = MultiAppSystemCopyTransfer
   direction = from_multiapp
   multi_app = initial_solve
   execute_on = initial
 [../]
[]
```

The input for the initial eigenvalue problem is 'lra_trans_initial.i'. Because the nonlinear system and the auxiliary system of the initial eigenvalue problem and the transient problem contain the exactly same variables, we can simply use *MultiAppSystemCopyTransfer* to transfer all solutions. This transfer will also transfer k-effective if there is one on the transport system. Because the *Mesh*, *Materials* and the *AuxVariables* of the initial eigenvalue problem will be exactly the same as those in the transient problem, we will give the rest part of the input here:

```
[TransportSystems]
 particle = neutron
 equation_type = eigenvalue

# number of energy groups
 G = 2

# BC and external sources
 DirichletBoundary = 'top right'
 ReflectingBoundary = 'bottom left'

 [./diff]
  scheme = CFEM-Diffusion
  n_delay_groups = 2
  family = LAGRANGE
  order = FIRST
  fission_source_as_material = true
 [../]
[]

[AuxKernels]
  [./power]
    type = FissionSource
```

```
    variable = power
    scalar_flux = 'sflux_g0 sflux_g1'
    nusigf = kappa_sigma_fission
    block ='1 2 3 4 6'
    execute_on = timestep_end
  [../]
[]

[Executioner]
  type = NonlinearEigen

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'

  free_power_iterations = 5
  source_abs_tol = 1e-10

# we do not normalize to power due to numerical issues
# 1e-6 * volume
#  power = 0.01755
[]
```

The only difference in the *TransportSystems* is now the 'equation_type' is eigenvalue. We do not need the temperature auxiliary kernel because it does not change initially. We also do not need to evaluate power on *linear* so we make it executed on 'timestep_end' to same a little computing time. *NonlinearEigen* is chosen for solving the problem. Their parameters can be found in Sec. 9.5. By default, *NonlinearEigen* normalizes the solution so that the total fission source is equal to k-effective. We do not normalize to power, $10^{-6}V_{core}$, as specified in the benchmark, due to numerical issues. The solution changes several magnitude which makes Jacobian-free approach not numerically stable at the late of the transient if the initial solution is high. We note that the normalization does not change the transient solution if we scale the power properly. Scalar fluxes need to be interpreted properly with this different normalization though. We choose to disable the input for the initial problem because their solution is stored by the transient problem. It is also acknowledged that creating the extra input for the initial eigenvalue problem could be cumbersome for users. But it gives users the ultimate flexibilities on controlling how the initial problem to be solved. We are considering to automate the creation of the input of the initial eigenvalue problem.

3.5.6 Temperature equation

The adiabatic temperature simply says that the temperature linearly depends on the total energy deposited locally. So we can use the auxiliary kernel for the time integration of power density to get the temperature. The scalar fluxes have been added by the transport system and material property for the kappa fission cross section is added by neutronics materials. We can evaluate their product to get the power density with an another auxiliary kernel. It is noted that the power density and temperature could be discontinuous in space. So we can not use LAGRANGE shape function as the one used by scalar fluxes. Instead, we want to use the discontinuous version of the LAGRANGE shape function, L2_LAGRANGE for these two variables. We also want to keep the order or these two variables the same as those for the scalar fluxes, so that no accuracy will be lose. The input blocks for setting up these two variables are in the following:

```
[AuxVariables]
  [./T]
    order = FIRST
    family = L2_LAGRANGE
    initial_condition = 300
```

```
  [../]
  [./power]
    order = FIRST
    family = L2_LAGRANGE
  [../]
[]

[AuxKernels]
  [./temperature]
    type = VariableTimeIntegrationAux
    variable = T
    variable_to_integrate = power
    # originally should be alpha/kappa = 1.1953808
    # due to the normalization, we need to have the coefficient vol*nu/kappa = 1.331039e9
    # so the final coefficient is alpha*vol*nu/kappa/kappa
    #coefficient = 1.591098860816678e+009
    coefficient = 3.182197721633356e+009
    block = '1 2 3 4 6'
    execute_on = linear
  [../]
  [./power]
    type = FissionSource
    variable = power
    scalar_flux = 'sflux_g0 sflux_g1'
    nusigf = kappa_sigma_fission
    block ='1 2 3 4 6'
    execute_on = linear
  [../]
[]
```

We have put the initial condition for the temperature in its declaration block because the condition is constant with 300K. We also make the two auxiliary kernels executed on *linear* in order to make them participate the residual evaluation. This make us performing fully-coupled multiphysics calculations. If however, we set the 'execute_on' to *timestep_end*, we will basically performing operator splitting calculations. It is also noted that the coefficient applied in *temperature* auxiliary kernel is $\frac{\alpha\nu V_{core}10^{-6}}{\kappa^2}$ instead of α/κ due to the initial normalization. The name of the scalar fluxes and kappa fission cross section can be found in Sec. 4.10 and Sec. 8.2.2. Details about two auxiliary kernels *VariableTimeIntegrationAux* and *FissionSource* can be found in Sec. 13.

3.5.7 Postprocessors and core map

The benchmark asks for 10 items of the expected primary results. The most important two are item 4 and 6. It is noted that due to our choice of the initial normalization, we need to apply a scaling factor $\frac{\nu V_{core}10^{-6}}{\kappa} \approx 1.331 \times 10^9$ on the power. We accomplished this with three postprocessors: 'avg_temp' for the core averaged temperature, 'avg_power' for the unscaled power and 'power' for the properly scaled power.

```
[Postprocessors]
  [./avg_temp]
    type = ElementAverageValue
    execute_on = 'initial timestep_end'
    variable = T
    block = '1 2 3 4 6'
  [../]
  [./avg_power]
    type = ElementAverageValue
    execute_on = 'initial timestep_end'
    variable = power
```

```
    block = '1 2 3 4 6'
    outputs = none
  [../]
  [./power]
    type = ScalePostprocessor
    execute_on = 'initial timestep_end'
    value = avg_power
    scaling_factor = 1.331039325843e9
  [../]
[]
```

We set 'outputs' for 'avg_power' to none to prevent it from being outputted by Rattlesnake. We do want the initial value of them, so the 'execute_on' parameter also contains *initial*.

Power and flux maps can be generated with the core map user objects:

```
[UserObjects]
  [./flux_map]
   type = FluxCartesianCoreMap
   transport_system = diff
   output_in = flux
   regular_grid = true
   grid_coord_x = '0 15 30 45 60 75 90 105 120 135'
   grid_coord_y = '0 15 30 45 60 75 90 105 120 135'
   execute_on = 'initial timestep_end'
  [../]
  [./temp_map]
   type = VariableCartesianCoreMap
   variables = T
   output_in = temp
   regular_grid = true
   grid_coord_x = '0 15 30 45 60 75 90 105 120 135'
   grid_coord_y = '0 15 30 45 60 75 90 105 120 135'
   execute_on = 'initial timestep_end'
  [../]
[]
```

We can use the regular grids for generating the core maps for this benchmark. The maps are created into two files, *flux* and *temp*, for power and temperature respectively.

3.5.8 Executioner

We will use Transient executioner for solving this problem. Lots of control parameters can be seen in Sec. 9.3. We will not try all of them, but just use the constant time stepper and the backward-Euler scheme for this problem. The input block is:

```
[Executioner]
  type = Transient

  start_time = 0
  end_time = 3
  dt = 1.e-3

  l_tol = 1e-2
```

```
  nl_max_its = 200
  nl_rel_tol = 1e-6
  nl_abs_tol = 1e-8

  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart'
  petsc_options_value = 'hypre boomeramg 100'

  timestep_tolerance = 1e-12
[]
```

The tolerance on PJFNK are set to the values so that further reducing them will not affect the time convergence significantly. We use BoomerAMG for this diffusion solve because of its good parallel performance. We do not have a good time stepper for this benchmark currently, so we used the default time stepper with constant time steps specified by *dt*.

3.5.9 Preconditioning

Although preconditioning is not necessary for this small benchmark, we want to demonstrate how optional preconditioning can be performed.

We choose to use the full single matrix as the preconditioning matrix:

```
[Preconditioning]
 [./SMP_jfnk]
   type = SMP
   full = true
  [../]
[]
```

To make this single matrix effective, we will need to assemble Jacobian for more kernels, which is controlled by three additional parameters in the diffusion system:

```
[TransportSystems]
 .
 .
 .
 [./diff]
  .
  .
  .
  assemble_fission_jacobian = true
  assemble_scattering_jacobian = true
  assemble_delay_jacobian = true
[]
```

3.5.10 Outputs

Outputs are simple: we just need to indicate Rattlesnake that we want the Exodus and CSV output.

```
[Outputs]
  exodus = true
  csv = true
[]
```

3.5.11 Primary results

We first run the problem with four level of uniform refinements, quadratic shape functions and $7.8125 \times 10^{-5}s$ time step with Crank-Nicolson time integration scheme. The results are gathered in Table 10. The eigenvalue with the control rod completely out can be simply obtained by running the input file for the initial eigenvalue problem with 'Executioner/time=3' on the command line. The peak powers and their occurrence are obtained through quadratic interpolation of the three adjacent time steps whose powers show a turn-around. These results agree well with the results of SPANDEX presented in [9].

Table 10 LRA benchmark reference results.

Initial k-effective	0.996368
k-effective with the control rod completely out	1.015445
The first averaged peak power (W/cm^3)	5455.46
The occurrence of the first peak (s)	1.44112
The second normalized peak power (W/cm^3)	795.498
The occurrence of the second peak (s)	2.00164
Averaged power density at 0.4s (W/cm^3)	1.38437×10^{-6}
Averaged power density at 0.8s (W/cm^3)	3.07545×10^{-6}
Averaged power density at 1.2s (W/cm^3)	6.67668×10^{-6}
Averaged power density at 1.4s (W/cm^3)	432.518
Averaged power density at 2.0s (W/cm^3)	795.126
Averaged power density at 3.0s (W/cm^3)	98.4276
Averaged fuel temperature at 3.0s (K)	1094.15

The averaged power density and averaged temperature versus time are plotted in Fig. 9. Normalized power densities and temperature at $t = 0.4s$, $0.8s$, $1.2s$, $1.4s$, $2.0s$ and $3.0s$ for 78 assemblies are given in Table 11 and Table 12.

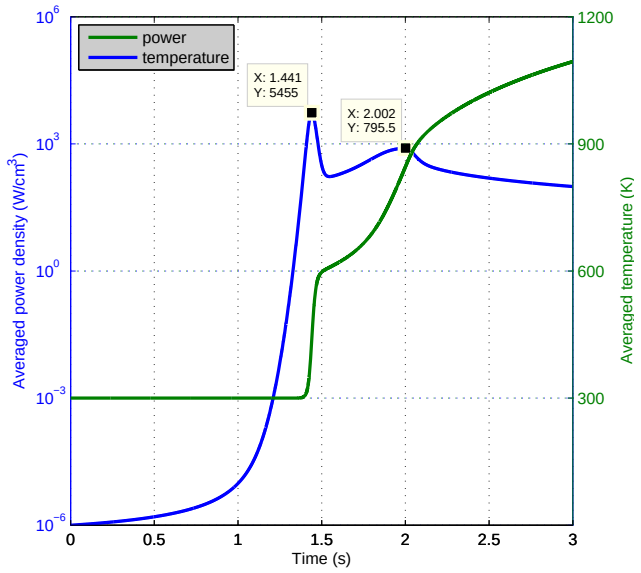


Figure 9 LRA benchmark results.

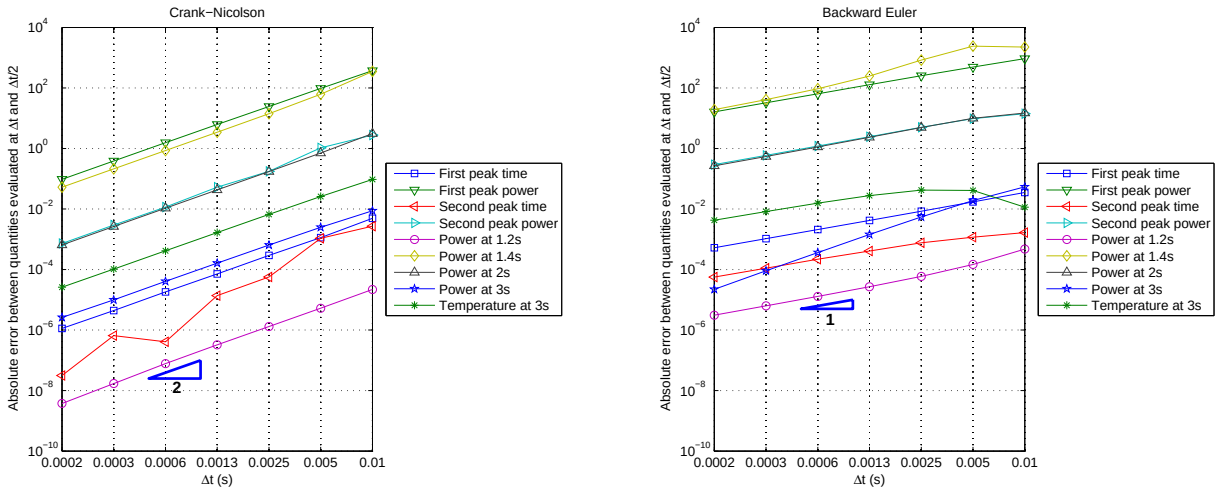
Table 11 Power map at different times.

0.8470	0.8030	0.7827	0.8321	0.9378	1.0034	0.8965		$t = 0.4$
0.7370	0.7116	0.7198	0.8024	0.9454	1.0485	0.9668		$t = 0.8$
0.5987	0.5957	0.6378	0.7598	0.9476	1.0978	1.0501		$t = 1.2$
0.5415	0.5464	0.6002	0.7354	0.9391	1.1086	1.0781		$t = 1.4$
0.4656	0.4756	0.5341	0.6717	0.8818	1.0752	1.0924		$t = 2.0$
0.5067	0.5102	0.5580	0.6823	0.8765	1.0541	1.0621		$t = 3.0$
1.3560	1.1851	1.1105	1.1943	1.4369	1.7475	1.7397	1.4801	
1.1779	1.0485	1.0217	1.1561	1.4587	1.8444	1.9077	1.6955	
0.9549	0.8761	0.9068	1.1014	1.4765	1.9558	2.1147	1.9721	
0.8634	0.8037	0.8549	1.0699	1.4708	1.9879	2.1950	2.0946	
0.7384	0.6957	0.7576	0.9747	1.3808	1.9374	2.2628	2.3328	
0.8039	0.7464	0.7903	0.9868	1.3661	1.8896	2.1880	2.2461	
1.5188	1.0632	0.9158	1.0032	1.3626	2.1496	2.3475	1.8878	1.0247
1.3168	0.9384	0.8432	0.9759	1.3955	2.2897	2.6122	2.2674	1.2829
1.0641	0.7816	0.7500	0.9369	1.4290	2.4552	2.9456	2.7762	1.6392
0.9608	0.7166	0.7086	0.9144	1.4321	2.5090	3.0851	3.0300	1.8257
0.8225	0.6199	0.6287	0.8363	1.3547	2.4746	3.2563	3.6249	2.3476
0.8973	0.6664	0.6556	0.8441	1.3348	2.4059	3.1399	3.4795	2.2532
1.2662	0.8678	0.7414	0.8282	1.1731	1.9417	2.2272	1.9472	1.1598
1.0970	0.7653	0.6827	0.8066	1.2029	2.0691	2.4771	2.3281	1.4317
0.8859	0.6371	0.6079	0.7764	1.2346	2.2214	2.7942	2.8399	1.8061
0.8002	0.5845	0.5753	0.7594	1.2398	2.2737	2.9301	3.0978	2.0023
0.6888	0.5087	0.5140	0.6998	1.1813	2.2561	3.1068	3.6995	2.5363
0.7532	0.5482	0.5369	0.7070	1.1646	2.1949	2.9978	3.5541	2.4370
0.7227	0.6193	0.5846	0.6625	0.8716	1.1940	1.4254	1.5513	1.0311
0.6266	0.5466	0.5369	0.6402	0.8820	1.2534	1.5475	1.7360	1.1723
0.5073	0.4562	0.4767	0.6102	0.8918	1.3246	1.7020	1.9769	1.3590
0.4597	0.4196	0.4510	0.5951	0.8911	1.3492	1.7671	2.0888	1.4486
0.3998	0.3697	0.4069	0.5515	0.8493	1.3307	1.8312	2.2905	1.6505
0.4384	0.3995	0.4272	0.5611	0.8438	1.3027	1.7774	2.2155	1.5970
0.4678	0.4525	0.4630	0.5332	0.6707	0.8509	1.0482	1.2652	0.8886
0.4051	0.3985	0.4218	0.5060	0.6601	0.8616	1.0850	1.3281	0.9395
0.3278	0.3317	0.3701	0.4708	0.6447	0.8728	1.1297	1.4066	1.0036
0.2976	0.3052	0.3486	0.4548	0.6356	0.8743	1.1463	1.4386	1.0305
0.2635	0.2731	0.3173	0.4218	0.6014	0.8470	1.1435	1.4760	1.0816
0.2911	0.2973	0.3366	0.4352	0.6070	0.8425	1.1277	1.4508	1.0632
0.3762	0.3735	0.3956	0.4676	0.5966	0.7642	0.9525	1.1624	0.8222
0.3236	0.3262	0.3552	0.4329	0.5660	0.7379	0.9322	1.1476	0.8158
0.2593	0.2681	0.3050	0.3889	0.5262	0.7027	0.9039	1.1253	0.8047
0.2347	0.2455	0.2847	0.3700	0.5079	0.6852	0.8885	1.1115	0.7966
0.2120	0.2228	0.2604	0.3412	0.4724	0.6443	0.8478	1.0774	0.7839
0.2371	0.2456	0.2804	0.3586	0.4873	0.6565	0.8568	1.0848	0.7887
0.3988	0.3645	0.3761	0.4599	0.6348	0.8949	1.1012	1.2314	0.8360
0.3397	0.3147	0.3327	0.4167	0.5843	0.8310	1.0306	1.1609	0.7918
0.2677	0.2538	0.2789	0.3621	0.5193	0.7476	0.9375	1.0669	0.7319
0.2405	0.2304	0.2575	0.3392	0.4906	0.7094	0.8938	1.0217	0.7024
0.2206	0.2111	0.2356	0.3097	0.4469	0.6459	0.8177	0.9445	0.6573
0.2502	0.2362	0.2575	0.3310	0.4703	0.6737	0.8474	0.9744	0.6771
0.5524	0.4000	0.3801	0.4770	0.7422	1.3056	1.5701	1.4046	0.8785
0.4676	0.3426	0.3332	0.4274	0.6739	1.1927	1.4414	1.2961	0.8136
0.3642	0.2725	0.2752	0.3648	0.5860	1.0453	1.2722	1.1528	0.7272
0.3252	0.2459	0.2523	0.3388	0.5474	0.9781	1.1937	1.0858	0.6863
0.3008	0.2263	0.2307	0.3071	0.4924	0.8773	1.0712	0.9790	0.6250
0.3439	0.2555	0.2543	0.3311	0.5233	0.9256	1.1244	1.0238	0.6529

Table 12 Temperature map at different times.

300.00	300.00	300.00	300.00	300.00	300.00	300.00		$t = 0.4$
300.00	300.00	300.00	300.00	300.00	300.00	300.00		$t = 0.8$
300.00	300.00	300.00	300.00	300.00	300.00	300.00		$t = 1.2$
303.14	303.17	303.48	304.26	305.44	306.41	306.23		$t = 1.4$
594.73	595.99	622.38	691.82	798.76	890.95	881.84		$t = 2.0$
715.91	718.78	758.31	860.20	1017.29	1155.48	1149.45		$t = 3.0$
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	
305.01	304.67	304.96	306.20	308.51	311.50	312.68	312.08	
768.68	734.02	757.45	867.58	1077.84	1357.01	1486.38	1465.07	
960.89	913.66	950.09	1111.48	1419.17	1832.37	2039.09	2033.68	
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
305.58	304.16	304.11	305.30	308.29	314.51	317.81	317.46	310.50
822.45	687.07	678.81	784.39	1057.06	1637.49	1979.04	2026.38	1372.59
1036.79	847.30	838.64	993.31	1391.20	2243.66	2773.28	2908.44	1943.86
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
304.65	303.39	303.34	304.40	307.17	313.14	316.92	317.85	311.52
736.39	616.67	608.41	703.42	957.26	1515.16	1897.98	2064.37	1469.42
916.11	748.32	739.18	878.32	1248.71	2067.97	2655.99	2964.93	2086.90
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
302.67	302.44	302.61	303.45	305.16	307.80	310.21	312.05	308.35
551.58	528.51	543.25	618.06	774.60	1021.73	1258.56	1456.48	1115.40
656.01	624.31	647.04	756.39	984.98	1348.82	1706.68	2015.99	1518.67
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
301.73	301.77	302.02	302.63	303.68	305.06	306.63	308.31	305.95
464.37	467.72	489.75	545.18	640.89	769.23	919.30	1085.00	868.08
533.46	538.76	571.14	651.76	791.09	979.17	1201.47	1448.58	1134.54
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
301.36	301.42	301.65	302.14	302.94	303.97	305.14	306.43	304.61
431.48	436.58	456.62	501.34	574.46	669.37	779.70	903.08	735.11
487.43	494.93	523.94	588.41	693.82	831.11	991.62	1171.89	930.63
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
301.40	301.34	301.49	301.97	302.84	304.11	305.18	305.92	304.07
436.79	429.84	443.02	485.93	566.60	683.87	782.95	853.20	682.22
495.46	485.57	504.43	565.70	680.75	848.09	990.14	1091.93	848.25
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00	300.00
301.89	301.43	301.46	301.96	303.17	305.67	306.92	306.30	303.98
486.81	439.67	440.90	486.29	598.14	830.90	946.43	887.68	672.93
567.18	499.70	501.31	565.76	724.60	1055.36	1219.76	1136.98	831.99

Convergence in time with quadratic shape functions in space and three-level uniform refinement for Crank-Nicolson and backward Euler time integration scheme are plotted in Fig. 10. It can be seen that the convergence



(a) Crank-Nicolson

(b) backward Euler

Figure 10 Time convergence of the LRA benchmark.

rates for LRA benchmark agree with the expected rates of both time integration schemes. The only exception is the convergence rate of the end power with the backward Euler scheme. Second order convergence is observed for the the end power with the backward Euler scheme. The similar convergence rates can be observed for linear shape functions and different level of uniform refinements and are not presented here. It is noted that with the fixed spatial discretization, we can do extrapolation to obtain the reference solution in time. The total CPU time taken for the case with quadratic shape functions, one-level uniform refinement, time step size being equal to 0.001s, and 24 processors on Falcon at INL is about 418seconds.

3.6 LRA PKE

This is a PKE tutorial using the LRA benchmark. All files necessary for running this tutorials with Rattlesnake are under '*rattlesnake/tutorials/LRA2D-PKE*' folder.

3.6.1 Dump PKE Parameters for LRA Benchmark

In order to make the constant function as a valid weighting function, we first change the benchmark a little by switch the Dirichlet boundary condition to vacuum boundary condition. In the transient input file and the initial eigenvalue input file, we have now

```
[TransportSystems]
 ...
 VacuumBoundary = 'top right'
 ...
[]
```

We are going to use backward Euler scheme in both spatial kinetics calculation and the PKE calculation in the next section. To make the evaluation of DNP concentrations consistent with the scheme so that we can exactly reproduce power history with PKE solve, we need to set two parameters for the discretization scheme in the transient input file:

```
[TransportSystems]
 ...
 VacuumBoundary = 'top right'
 ...
 [./diff]
   ...
  linear_fsrc_in_time = false
  dnp_integration_scheme = backward-Euler
 [../]
[]
```

These two steps are necessary to make the PKE calculation with the dumped PKE parameters exactly reproduce the power history of the spatial kinetics calculation. In reality, however, there are other sources causing much larger errors than skipping these two steps, these two steps can be skipped.

We need to use IQS executioner in order to dump the PKE parameters especially the dynamic reactivity during the spatial kinetics transient solve.

```
[Executioner]
   type = IQS
   do_iqs_transient = false
   pke_param_csv = lra_pke_params.csv
[]
```

The 'lra_pke_params.csv' will be used for storing the dumped PKE parameters.

The IQS executioner adds some auxiliary variables for storing the saved-in residuals for evaluating dynamic reactivity. This prevents us from using the simple system-copy transfer. We have to change the transfer block to the following:

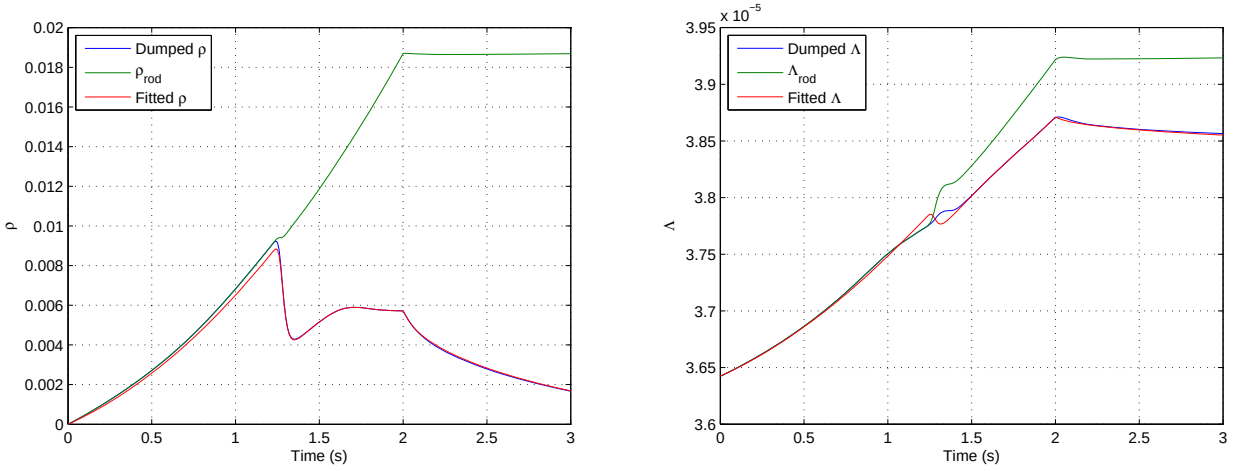
```
[Transfers]
 [./copy_vars]
   type = MultiAppVariableTransfer
   execute_on = initial
   direction = from_multiapp
   multi_app = initial_solve
   from_variables = 'sflux_g0 sflux_g1 power fission_source diff_dnp_i0 diff_dnp_i1'
   to_variables = 'sflux_g0 sflux_g1 power fission_source diff_dnp_i0 diff_dnp_i1'
 [../]
 [./copy_eigenvalue]
   type = EigenvalueTransfer
   execute_on = initial
   direction = from_multiapp
   multi_app = initial_solve
 [../]
[]
```

It is noted that along with the variables, eigenvalue needs to be transferred as well so that the transient system can use it for modifying the fission cross section.

We want to add four new postprocessors for monitoring the change of DNP concentration during the transient.

```
[Postprocessors]
  ...
  [./avg_dnp0]
    type = ElementAverageValue
    execute_on = 'initial timestep_end'
    variable = diff_dnp_i0
    block = '1 2 3 4 6'
    outputs = none
  [../]
  [./dnp0]
    type = ScalePostprocessor
    execute_on = 'initial timestep_end'
    value = avg_dnp0
    scaling_factor = 1.331039325843e9
  [../]
  [./avg_dnp1]
    type = ElementAverageValue
    execute_on = 'initial timestep_end'
    variable = diff_dnp_i1
    block = '1 2 3 4 6'
    outputs = none
  [../]
  [./dnp1]
    type = ScalePostprocessor
    execute_on = 'initial timestep_end'
    value = avg_dnp1
    scaling_factor = 1.331039325843e9
  [../]
[]
```

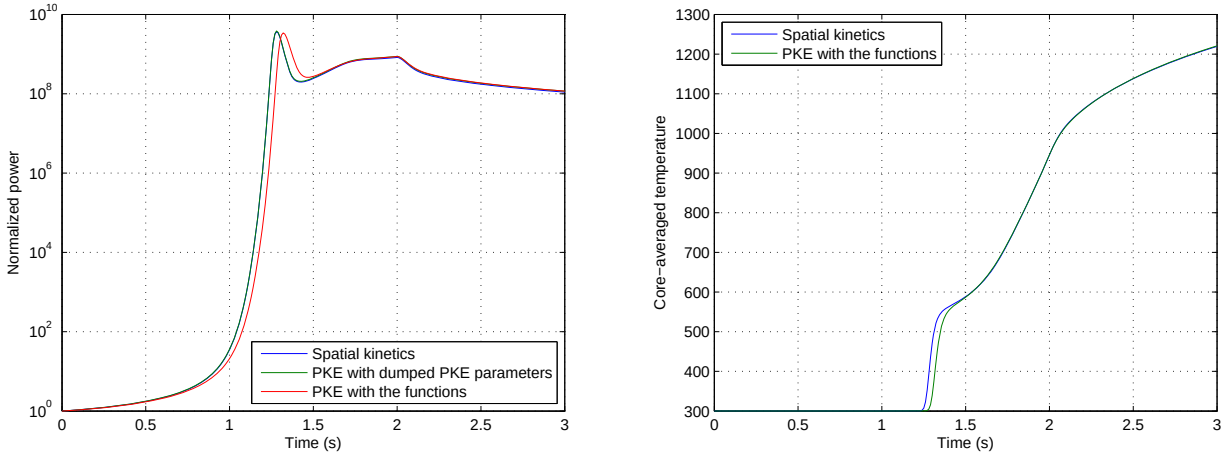
Once we made all of these changes, we can run the input to generate the PKE parameters in 'lra_pke_params.csv'. Because we are using the constant weighting function and $\lambda_i, \beta_i, i = 1, 2$ are constant in the fuel, we have the same values for them in the PKE parameters. The dynamic reactivity ρ and the generation time Λ are plotted in Fig. 11. Power (normalized at initial) history from the spatial kinetics calculation is plotted in Fig. 12a.



(a) ρ (b) Λ

Figure 11 PKE parameters of LRA benchmark.

Temperature history is plotted in Fig. 12b.



(a) Normalized power n

(b) Fuel averaged temperature T

Figure 12 PKE solutions of LRA benchmark.

3.6.2 Create a PKE Model for LRA Benchmark to Reproduce the Power History

The PKE input file can be

```
[PKE]
  n_delayed_groups = 2
  amplitude_variable = n
  DNP_variable = precursor
  DNP_fraction_aux = beta
  DNP_decay_constant_aux = lambda
  generation_time_aux = Lambda
  reactivity_aux = rho
  pke_parameter_csv = lra_pke_params.csv
[]

[ICs]
  [./ic_n]
    type = ScalarComponentIC
    variable = n
    values = 1
  [../]
[]

[Executioner]
  type = Transient
  nl_max_its = 100
  timestep_tolerance = 1e-12
  start_time = 0
  end_time = 3
  dt = 1.e-2
[]

[Outputs]
 csv = true
[]
```

We have initial equilibrium DNP concentration. It is noted that the generation time changes with time. The PKE solve actually gives $\frac{\Lambda}{\Lambda_0}n$. The normalized power history from this PKE calculation with the dumped PKE parameters is also plotted in Fig. 12a.

The PKE input file can be

```
[PKE]
  n_delayed_groups = 2
  amplitude_variable = n
  DNP_variable = precursor
  DNP_fraction_aux = beta
  DNP_decay_constant_aux = lambda
  generation_time_aux = Lambda
  reactivity_aux = rho
  pke_parameter_csv = lra_pke_params.csv
[]

[ICs]
  [./ic_n]
    type = ScalarComponentIC
    variable = n
    values = 1
  [../]
[]

[Executioner]
  type = Transient
  nl_max_its = 100
  timestep_tolerance = 1e-12
  start_time = 0
  end_time = 3
  dt = 1.e-2
[]

[Outputs]
 csv = true
[]
```

We have initial equilibrium DNP concentration. It is noted that the generation time changes with time. The PKE solve actually gives $\frac{\Lambda}{\Lambda_0}n$.

3.6.3 Fit Reactivity and Generation Time with Averaged Temperature and Control-Rod Fraction

The transient contains two stages: from 0 to 2 seconds with control-rod movements; from 2 to 3 seconds without control-rod movements. We first use least square method to fit the reactivity and generation time from 2 to 3 seconds where only thermal feedback takes effect. Because the Doppler feedback is introduced with the relative change of the square root of local fuel temperature, we would assume the temperature dependency of reactivity and generation time is

$$\rho(2 <= t <= 3) = \rho_{rod}(t = 2) - \alpha_\rho(\sqrt{T} - \sqrt{T_0}), \tag{13}$$

$$\Lambda(2 <= t <= 3) = \Lambda_{rod}(t = 2) - \alpha_\Lambda(\sqrt{T} - \sqrt{T_0}), \tag{14}$$

where ρ_{rod} and Λ_{rod} are the changes due to control-rod movement and remain constant from 2 to 3 seconds. The fitting gives $\alpha_{rho} = 3.79473349 \times 10^{-4}$ and $3.79473349 \times 10^{-8}$.

Now we can plot the change of reactivity and generation time caused by control-rod movements: $\rho_{rod}(t) = \rho + \alpha_\rho(\sqrt{T} - \sqrt{T_0})$ and $\Lambda_{rod}(t) = \Lambda + \alpha_\Lambda(\sqrt{T} - \sqrt{T_0})$ in Fig. 11. We can see that both $\rho_{rod}(t)$ and $\Lambda_{rod}(t)$ remains about constant after 2 second as expected. The dependency on rod fraction is not linear from 0 to 2 second, also there is a little non-smoothness around the time when the first peak power happens. Because the reactivity function after power distribution is established is more important for heat deposition, we will use the time period from 1.5 second to 2 second to fit the following cubic function

$$\rho_{rod}(0 \le t \le 2) = \sum_{i=1}^{3} \beta_{\rho,i}(1 - c(t))^i, \tag{15}$$

$$\Lambda_{rod}(0 \le t \le 2) = \Lambda(t = 0) + \sum_{i=1}^{3} \beta_{\Lambda,i}(1 - c(t))^i, \tag{16}$$

where $c(t)$ is the control-rod fraction

$$c(t) = \begin{cases} 1 - t/2, & 0 \le t \le 2 \\ 0, & t > 2 \end{cases}. \tag{17}$$

The fitted coefficients are listed in Table 13.

Table 13 Fitting coefficients.

i	$\beta_{\rho,i}$	$\beta_{\Lambda,i}$
1	$7.65201825 \times 10^{-3}$	$1.32442742 \times 10^{-6}$
2	$1.04302712 \times 10^{-2}$	$1.69547914 \times 10^{-6}$
3	$6.18254986 \times 10^{-4}$	$-2.25423161 \times 10^{-7}$

The final reactivity function would be

$$\rho(c, T) = \sum_{i=1}^{3} \beta_{\rho,i}(1 - c(t))^i - \alpha_\rho(\sqrt{T} - \sqrt{T_0}), \tag{18}$$

$$\Lambda(c, T) = \Lambda(t = 0) + \sum_{i=1}^{3} \beta_{\Lambda,i}(1 - c(t))^i - \alpha_\Lambda(\sqrt{T} - \sqrt{T_0}). \tag{19}$$

They are plotted in Fig. 11 with the given temperature history. It can be seen that the reactivity from 0 to 1.5 second is slightly lower than the dumped reactivity, which will possibly make the peak power happen later.

If we integrate Eq. (5) over the core, and divide by the core volume, we obtain

$$\frac{\partial \bar{T}(t)}{\partial t} = \frac{\alpha}{\nu}\bar{F}(t) = \frac{\alpha}{\nu}\bar{F}_0 n(t) = \frac{\alpha}{\nu}(\frac{\bar{P}_0}{\kappa}\nu)n = \frac{\alpha \bar{P}_0}{\kappa}n \tag{20}$$

where $\bar{F}$ is the core-averaged fission reaction rate in unit of $\frac{fission}{cm^3}$ and $\bar{T}$ is the core-averaged temperature. $\bar{F}_0$ is the initial core-averaged fission reaction rate in unit of $\frac{fission}{cm^3}$. The value of $\frac{\alpha \bar{P}_0}{\kappa}$ is $1.19538077 \times 10^{-6}$ in unit of $\frac{K}{s}$.

Now we have set up a PKE equation with the fitted reactivity and generation-time functions:

$$\frac{\partial n(t)}{\partial t} = \frac{\rho(c, T) - \beta}{\Lambda(c, T)}n(t) + \sum_{i=1}^{2} \lambda_i c_i(t) \tag{21}$$

$$\frac{\partial c_i(t)}{\partial t} = \frac{\beta_i}{\Lambda(c, T)}n(t) - \lambda_i c_i(t), i = 1, 2, \tag{22}$$

$$\frac{\partial \bar{T}(t)}{\partial t} = \frac{\alpha \bar{P}_0}{\kappa}\frac{\Lambda_0}{\Lambda(c, T)}n(t), \tag{23}$$

with

$$n(t=0) = 1, \tag{24}$$

$$\bar{T}(t=0) = T_0, \tag{25}$$

and initial equilibrium DNP concentrations.

3.6.4 Create PKE Model for LRA Benchmark with the Fitted Functions

We still have the PKE input block for the PKE equations, but now the PKE parameters will not be set up from the dumped file. The only change from the input for reproducing the power history is to delete the 'pke_parameter_csv' parameter.

```
[PKE]
  n_delayed_groups = 2
  amplitude_variable = n
  DNP_variable = precursor
  DNP_fraction_aux = beta
  DNP_decay_constant_aux = lambda
  generation_time_aux = Lambda
  reactivity_aux = rho
[]
```

The DNP fractions β and decay constants λ can be set with the initial condition. They will remain constant while no auxiliary kernels are added for them. Initial condition of generation time Λ needs to be set for the equilibrium condition of DNPs. Its initial value can be obtained from the dumped PKE parameters. Initial condition of reactivity is not used for calculation, but it is better to set it to zero to be consistent with the equilibrium condition. As the results, the *ICs* input block is now:

```
[ICs]
  [./ic_n]
    type = ScalarComponentIC
    variable = n
    values = 1
  [../]
  [./ic_beta]
    type = ScalarComponentIC
    variable = beta
    values = '0.0054 0.001087'
  [../]
  [./ic_lambda]
    type = ScalarComponentIC
    variable = lambda
    values = '0.0654 1.35'
  [../]
  [./ic_Lambda]
    type = ScalarComponentIC
    variable = Lambda
    values = 3.64255765123e-05  # value from the dumped PKE parameters
  [../]
  [./ic_rho]
    type = ScalarComponentIC
    variable = rho
    values = 0
  [../]
[]
```

We will need two auxiliary scalar kernels to evaluate the reactivity and generation time.

```
[Functions]
 [./fraction]
   type = SlopeFunction
   timep = '0 2 3'
   value = '0 1 1'
 [../]
[]

[AuxScalarKernels]
  [./rho]
    type = RoddedFeedbackAux
    variable = rho
    control_rod_fraction = fraction
    rod_fraction_monomial_coefficients = '0 7.65201825e-3 1.04302712e-2 6.18254986e-4'
    temperature = T
    temperature_coefficient = -9.66804214e-4
    reference_temperature = 300
  [../]
  [./Lambda]
    type = RoddedFeedbackAux
    variable = Lambda
    control_rod_fraction = fraction
    rod_fraction_monomial_coefficients = '3.64255765123e-05 1.32442742e-6 1.69547914e-6 -2.25423161e-7'
    temperature = T
    temperature_coefficient = -3.79473349e-8
    reference_temperature = 300
  [../]
[]
```

A function for one minus the control-rod fraction is also added and used by these two auxiliary kernels. *RoddedFeedbackAux* auxiliary kernel simply use Eq. (18) and Eq. (19) to evaluate the auxiliary variables. Although *RoddedFeedbackAux* auxiliary kernel is included in Rattlesnake, its task is too specific to be documented in this manual. On the other hand, its source can be used as an example to demonstrate that how Rattlesnake can be extended easily for special needs.

To set up the temperature equation, we first add the temperature variable.

```
[Variables]
  [./T]
    family = SCALAR
    order = FIRST
    initial_condition = 300
  [../]
[]
```

Its initial condition is added inline.

Then we add an auxiliary variable for the coupling coefficient from the deposited power to the temperature.

```
[AuxVariables]
  [./power_coef]
    family = SCALAR
```

```
    order = FIRST
    initial_condition = 4.35424338e-011
  [../]
[]
```

It is noted that the coefficient contains the initial generation time and remains constant throughout the transient.

We now need to have two kernels for the temperature variable:

```
[ScalarKernels]
  [./dT]
    type = ODETimeDerivative
    variable = T
  [../]
  [./power]
    type = ScalarPrecursorSource
    variable = T
    betas = power_coef
    Lambda = Lambda
    amplitude = n
  [../]
[]
```

Scalar kernel *ScalarPrecursorSource* is also used by Rattlesnake for setting up PKE.

3.6.5 Results with the PKE Model with the Fitted Functions

The normalized power and core-averaged temperature are plotted in Fig. 12. The first power peak is delayed while the final temperatures (total deposited energy) is very close with respect to the spatial kinetics results as expected. These results demonstrated that the LRA benchmark can be simplified into a PKE without significantly losing accuracy with strong thermal feedback.

3.7 C5G7-2D with SAAF-SN-CFEM NDA

This is a seven-group transport benchmark without homogenization. It is solved with SAAF-CFEM-SN scheme with nonlinear diffusion acceleration (NDA). All files necessary for running this tutorials with Rattlesnake are under *'rattlesnake/tutorials/C5G72D/saaf-nda'* folder.

3.7.1 Problem Description

Refer to Section 3.8.1.

3.7.2 Mesh Generation

Refer to Section 3.8.2.

3.7.3 Transport Materials

Refer to Section 3.8.6.

3.7.4 Postprocessing and Outputs

We use the following inputs to generate the flux map.

```
[UserObjects]
  [./flux_map]
    type = FluxCartesianCoreMap
    transport_system = saaf
    print = 'assembly pin'
    print_fission_absorption_ratio = false
    power_map_from = kappa_sigma_f
    execute_on = 'initial timestep_end'
 [../]
[]
```

Assembly IDs and pin IDs are presented as variables in the mesh file generated by INSTANT. We will use the fixed the transport system name *'saaf'* from now on. Users are allowed to use their own preferred names.

We also cares the run-time with:

```
[Postprocessors]
  [./runtime]
    type = RunTime
    time_type = alive
  [../]
[]
```

The outputs can be simply as

```
[Outputs]
  exodus = true
  print_perf_log = true
[]
```

3.7.5 Direct Transport Solve with SAAF-SN-CFEM

To perform a calculation with NDA, we first want to create an input for direct transport solve. What we need other than previous sections are two blocks.

One is for the transport system:

```
[TransportSystems]
  particle = neutron
  equation_type = eigenvalue

  G = 7

  VacuumBoundary = 'vacuum'
  ReflectingBoundary = 'reflecting'
```

```
  [./saaf]
    scheme = SAAF-CFEM-SN
    family = LAGRANGE
    order = FIRST
    fission_source_as_material = true
    n_delay_groups = 0

    AQtype = Bickley3-Optimized
    NPolar = 2
    NAzmthl = 8

    tau = 0
  [../]
[]
```

'vacuum' and *'reflecting'* are the names of the two side sets generated by INSTANT. We turn *fission_source_as_material* to true because we do not need fission source to be stored as variables thus do not waste time on converting back and forth between fission source values on quadrature points and expansion coefficients of shape functions. *Bickley3-Optimized* angular quadrature has been proved to be the best quadrature for this problem in term of accuracy with the least number of streaming directions. Setting *tau* to zero to turn off the void treatment.

Another is the executioner:

```
[Executioner]
  type = NonlinearEigen

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'

  free_power_iterations = 2
  l_max_its = 600

  source_abs_tol = 1e-8
[]
```

We want to use BoomerAMG as the preconditioner because it is more parallel scalable with domain decomposition.

3.7.6 Converting the Direct Transport Solve for Transport Update

We essentially need to do four things:

1. to set *equation_type* in *TransportSystems* block to *steady-state*;
2. to set *for_transport_update* in *TransportSystems/saaf* block to true;
3. to change the executioner to:

   ```
   [Executioner]
     type = AMGUpdate
     amg_tol = 1e-3
     debug = false
   []
   ```

We do not need to fully converge the AMG cycles for transport update. We reduce the residual norm by a factor 10^{-3} with *amg_tol*.

4. to change the output block to

```
[Outputs]
  exodus = false
  csv = true
  print_perf_log = true
[]
```

and remove the user object for flux map. We do not care the Exodus outputs on the transport system because scalar fluxes will be outputted on the low order diffusion system. The flux map will also be on the low order system. We do want to turn on CSV outputs to store our NDA convergence history.

3.7.7 The Low Order Diffusion System

Because the low order diffusion system will use the same mesh and materials as of the transport system, we start creating its input file from the input for the direct transport solve.

We first add an application wrapper (a user object) for transport update:

```
[UserObjects]
  [./flux_map]
    type = FluxCartesianCoreMap
    transport_system = saaf
    print = 'assembly pin'
    print_fission_absorption_ratio = false
    power_map_from = kappa_sigma_f
    execute_on = 'initial timestep_end'
 [../]
 [./saaf_transport]
    type = NewSAAFWrapper
    app_type = RattleSnakeApp
    input_file = saaf.i
 [../]
[]
```

We then change the *TransportSystems* block to:

```
[TransportSystems]
  particle = neutron
  equation_type = eigenvalue

  G = 7

  VacuumBoundary = 'vacuum'
  ReflectingBoundary = 'reflecting'

  [./saaf]
    scheme = CFEM-Diffusion
    family = LAGRANGE
    order = FIRST
    fission_source_as_material = true
```

```
    n_delay_groups = 0

    transport_wrapper = saaf_transport
  [../]
[]
```

We basically switch the discretization scheme from SAAF-CFEM-SN to CFEM-Diffusion. And we tells the diffusion scheme the application wrapper, that are used for providing drift vectors and vacuum boundary coefficients on the quadrature points, with *transport_wrapper*.

We will then use *PicardEigen* for the Picard nonlinear diffusion iteration:

```
[Executioner]
  type = PicardEigen

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'

  free_power_iterations = 2
  l_max_its = 600

  output_after_power_iterations = false
  source_abs_tol = 1e-50
  source_rel_tol = 1e-4

  picard_max_its = 12
  wrapped_app_tol = 1e-6
  output_on_final = false
[]
```

Five parameters have been kept from *NonlinearEigen*. The eigenvalue solve does not have to be complete. Instead, the convergence criteria on the relatively reducing the initial residual norm is set to 10^{-4} without adverse impact on the convergence of the nonlinear diffusion iteration. The nonlinear diffusion iteration converges fast. Typically 10 iterations can reduce the error on eigenvalue to less than 1pcm. To be more conservative, we set the maximum number of iterations to 12. The application wrapper provide the maximum relative error of scalar fluxes for checking the convergence.

Finally we want to turn both *exodus* and *csv* on in the outputs

---

```
[Outputs]
  exodus = true
  csv = true
  print_perf_log = true
[]
```

3.7.8 Results

Results for this tutorial can be found in Ref. [10].

3.8 C5G7-2D using First Order NDA solver

This example covers setting up Rattlesnake inputs for solving the two-dimensional C5G7 benchmark problem using the first order NDA method and Nonlinear Diffusion Acceleration. A detailed description of the two-dimensional C5G7 benchmark can be found in [11]. All files necessary for running this tutorials with Rattlesnake are under *'rattlesnake/tutorials/C5G72D/fiso-nda'* folder.

3.8.1 Problem Description

The C5G7 benchmark is a MOX fueled, pressurized water reactor (PWR) minicore configuration. The C5G7 geometry depicted in Fig. 13 comprises a 2-by-2 array of UO_2 and MOX assemblies surrounded by moderator. In Fig. 13 vacuum and reflective boundary conditions are denoted by V and R, respectively. The fuel pins are not homogenized, each pin cell is comprised of a fuel pin, fission chamber, or guide tube surrounded by moderator as depicted in Fig. 14. The cladding and gap are homogenized into the fuel and are not explicitly modeled. Each assembly is made up of a regular 17-by-17 grid of pin cells. Seven energy-group cross sections for the seven material regions are given in [11]. The energy group boundaries are provided in Table 14; three of the seven energy groups are fast, four are thermal.

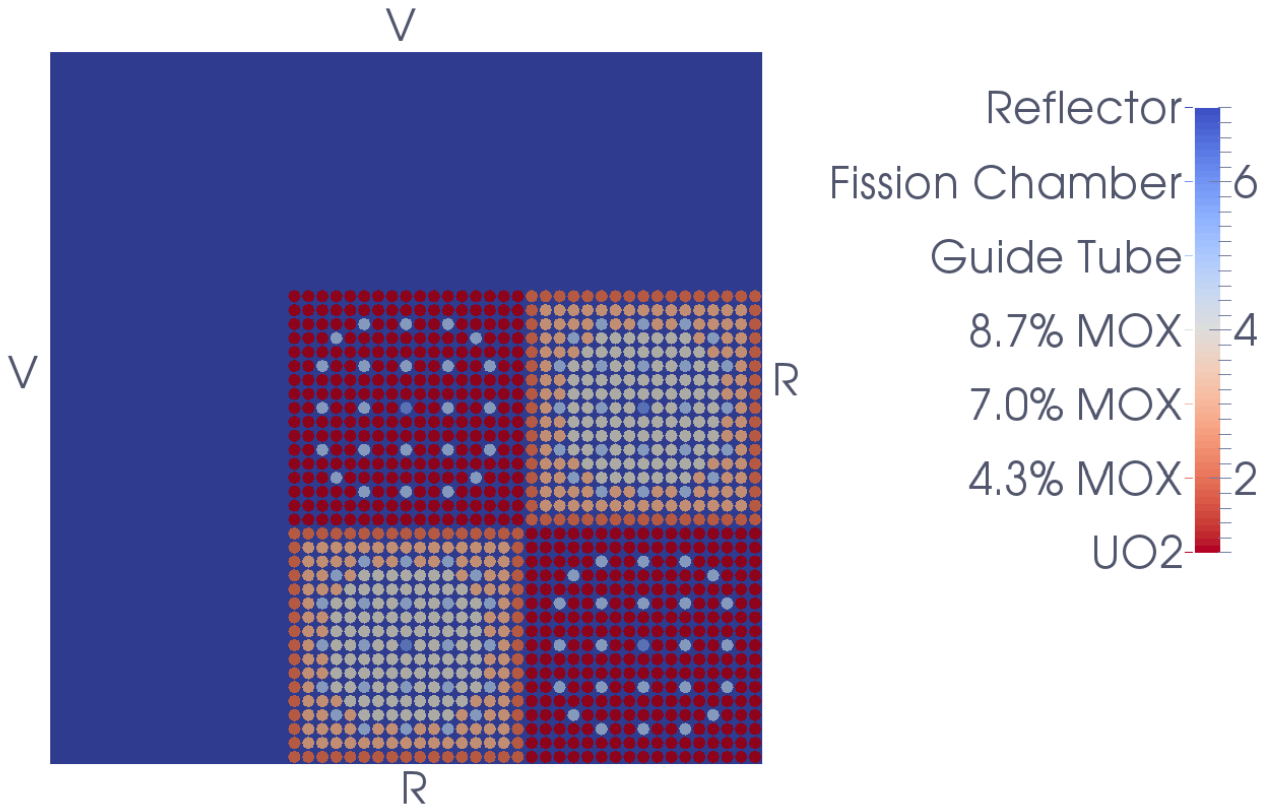


Figure 13 Geometry of the two-dimensional C5G7 benchmark problem. Boundary conditions are vacuum (V) or reflective (R).

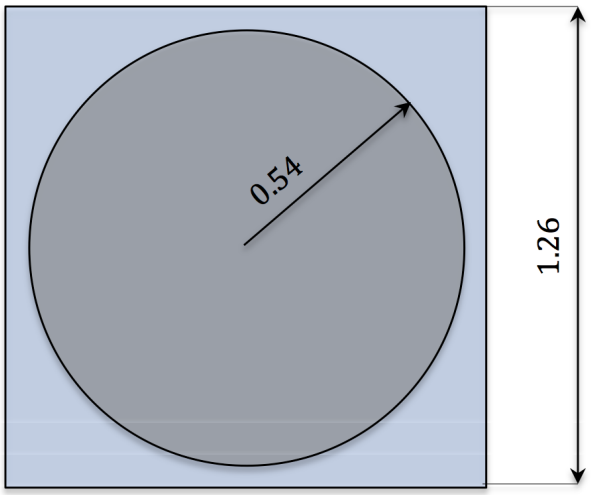


Figure 14 Geometry of a C5G7 pin-cell.

Table 14 Energy group boundaries for C5G7 MOX benchmark.

Group	Upper Energy
1	20 MeV
2	1 MeV
3	500 keV
4	3 eV
5	0.625 MeV
6	0.1 MeV
71	0.02 MeV

3.8.2 Mesh Generation

A triangular unstructured mesh is created using the INSTANT mesh generator, [12], [13]. INSTANT reads a (hierarchical) geometry description in XML format and calls Triangle to triangulate the geometry. INSTANT is compiled and executed using the commands:

```
  > Go to the Rattlesnake directory
  make instant
./yak/contrib/instant/instant_mesh_generator-opt /path/to/instant_c5g7.xml
```

The input file instant_c5g7.xml for the INSTANT mesh generator is listed below:

```
<task type="generation">

  <!--
    Description of the mesh
  -->
  <Geometry type="LWR">
```

```
<Controls>
  <MaxArea>0.064</MaxArea>
  <PinMaxArea>0.256</PinMaxArea>
  <AssemblyMaxArea>4.096</AssemblyMaxArea>
  <MinAngle>20</MinAngle>
  <DebugOutput>t</DebugOutput>
  <BlockOption>0</BlockOption>
</Controls>
<Pins>
  <Pin ID="1" shape="cylindrical" type="full" name="UO2">
    <Radius>0.54 0.63</Radius>
    <MaterialID>1 7</MaterialID>
    <NSides>8</NSides><Rotation>0</Rotation>
  </Pin>
  <Pin ID="2" shape="cylindrical" type="full" name="MOX4.3">
    <Radius>0.54 0.63</Radius>
    <MaterialID>2 7</MaterialID>
    <NSides>8</NSides><Rotation>0</Rotation>
  </Pin>
  <Pin ID="3" shape="cylindrical" type="full" name="MOX7.0">
    <Radius>0.54 0.63</Radius>
    <MaterialID>3 7</MaterialID>
    <NSides>8</NSides><Rotation>0</Rotation>
  </Pin>
  <Pin ID="4" shape="cylindrical" type="full" name="MOX8.7">
    <Radius>0.54 0.63</Radius>
    <MaterialID>4 7</MaterialID>
    <NSides>8</NSides><Rotation>0</Rotation>
  </Pin>
  <Pin ID="5" shape="cylindrical" type="full" name="Guide Tube">
    <Radius>0.54 0.63</Radius>
    <MaterialID>5 7</MaterialID>
    <NSides>8</NSides><Rotation>0</Rotation>
  </Pin>
  <Pin ID="6" shape="cylindrical" type="full" name="Fission Chamber">
    <Radius>0.54 0.63</Radius>
    <MaterialID>6 7</MaterialID>
    <NSides>8</NSides><Rotation>0</Rotation>
  </Pin>
  <Pin ID="7" shape="rectangular" name="Reflector">
    <NX>1</NX><NY>1</NY><IX>1</IX><IY>1</IY>
    <DX>1.26</DX><DY>1.26</DY>
    <MaterialID>7</MaterialID>
  </Pin>
</Pins>
<Assemblies>
  <Assembly ID="1" name="UO2">
    <NXPin>17</NXPin><NYPin>17</NYPin>
    <PinArrangement>
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 5 1 1 5 1 1 5 1 1 1 1 1
     1 1 1 5 1 1 1 1 1 1 1 1 1 5 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 5 1 1 5 1 1 5 1 1 5 1 1 5 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 5 1 1 5 1 1 6 1 1 5 1 1 5 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 5 1 1 5 1 1 5 1 1 5 1 1 5 1 1
```

```
      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      1 1 1 5 1 1 1 1 1 1 1 1 1 5 1 1 1
      1 1 1 1 1 5 1 1 5 1 1 5 1 1 1 1 1
      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     </PinArrangement>
     <XT>21.42</XT><YT>21.42</YT>
    </Assembly>
    <Assembly ID="2" name="MOX">
     <NXPin>17</NXPin><NYPin>17</NYPin>
     <PinArrangement>
      2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
      2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2
      2 3 3 3 3 5 3 3 5 3 3 5 3 3 3 3 2
      2 3 3 5 3 4 4 4 4 4 4 4 3 5 3 3 2
      2 3 3 3 4 4 4 4 4 4 4 4 4 3 3 3 2
      2 3 5 4 4 5 4 4 5 4 4 5 4 4 5 3 2
      2 3 3 4 4 4 4 4 4 4 4 4 4 4 3 3 2
      2 3 3 4 4 4 4 4 4 4 4 4 4 4 3 3 2
      2 3 5 4 4 5 4 4 6 4 4 5 4 4 5 3 2
      2 3 3 4 4 4 4 4 4 4 4 4 4 4 3 3 2
      2 3 3 4 4 4 4 4 4 4 4 4 4 4 3 3 2
      2 3 5 4 4 5 4 4 5 4 4 5 4 4 5 3 2
      2 3 3 3 4 4 4 4 4 4 4 4 4 3 3 3 2
      2 3 3 5 3 4 4 4 4 4 4 4 3 5 3 3 2
      2 3 3 3 3 5 3 3 5 3 3 5 3 3 3 3 2
      2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2
      2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
     </PinArrangement>
     <XT>21.42</XT><YT>21.42</YT>
    </Assembly>
    </Assembly>
    <Assembly ID="3" name="Reflector">
     <NXPin>17</NXPin><NYPin>17</NYPin>
     <PinArrangement>
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
      7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
     </PinArrangement>
     <XT>21.42</XT><YT>21.42</YT>
    </Assembly>
  </Assemblies>
  <Core name="C5G7">
    <BC>1 0 1 0</BC>
    <NX>3</NX><NY>3</NY>
    <Layout>
     1 2 3
```

```
        2 1 3
        3 3 3
      </Layout>
      <Homogenization>
        0 0 0
        0 0 0
        0 0 0
      </Homogenization>
    </Core>
  </Geometry>

  <!--
    1-15 for generating blocks
  -->
  <option>3</option>

  <output>c5g7-2d-pccm.e</output>
</task>
```

Three options *'task/Geometry/Controls/MaxArea*, *task/Geometry/Controls/PinMaxArea* and *task/Geometry/Controls/AssemblyMaxArea'* are used to control how fine the mesh is. The smaller this number is the more triangles will be generated. *'task/Geometry/Controls/MinAngle=20'* makes sure that no triangle has an interior angle smaller than 20 degrees. *'task/Geometry/Pins/Pin/NSides=8'* is used to indicate that we use 8 sides to approximate the circle of the fuel pin. *'task/option=3'* tells INSTANT that we want to created blocks based on the material ID and processor ID assigned to elements. The exodus mesh file is called c5g7-2d-pccm.e and can be used directly in Rattlesnake. To make the mesh embedded with the regular grid for diffusion calculation, the reflector assembly also has 17-by-17 regular rectangular pins. More details about the input can be found in INSTANT user manual.

3.8.3 Nonlinear Diffusion Acceleration and the MultiApp System

The first order S_N NDA method is implemented using the MultiApp system with the diffusion solve being the master app and the first order S_N solve being the sub app. The diffusion mesh and group structure can be coarser (fewer groups and fewer mesh elements) but both have to be nested within the S_N energy group structure and mesh, respectively. Cross sections are supplied to the sub app only, the system takes care of transferring the appropriate cross sections to the master solve. The user has to provide two input files: one for the S_N solve and one for the diffusion solve. Within this tutorial first the S_N input file c5g7_sub.i is discussed and then the diffusion input files are discussed.

3.8.4 S_N Mesh Block

The S_N problems mesh block is simple because it simply loads the INSTANT created exodus mesh file.

```
[Mesh]
 file = c5g7-2d-pccm.e
[]
```

3.8.5 S_N Transport System Block

The TransportSystems block is listed below. Most entries do not require any explanation, but several require specific attention as they are special for first order NDA mode. First the scheme must be DFEM-SN. Second, the computation of cross sections (in particular the diffusion coefficient) in the diffusion system requires the linearly anisotropic scattering rates requiring to set NA to one or larger even if the cross sections only comprise isotropic

scattering. Further, for_transport_update must be set to true because first order NDA can only be executed using the SweepUpdateExecutioner that requires for_transport_update=true. Finally, initialize_angular_flux must be set to true because NDA starts with a diffusion calculation that requires properly initialized cross sections. If all angular fluxes are set to zero, then computed cross sections will be NaN.

```
[TransportSystems]
  particle = neutron
  G = 7
  VacuumBoundary = 'vacuum'
  ReflectingBoundary = 'reflecting'
  equation_type = steady-state
  [./sn]
    scheme = DFEM-SN
    family = MONOMIAL
    order = FIRST
    AQorder = 8
    AQtype = Level-Symmetric
    NA = 1
    for_transport_update = true
    initialize_angular_flux = true
  [../]
[]
```

3.8.6 Transport Materials

Materials are loaded from the INSTANT-XS formatted file c5g7_materials.xml:

```
<Materials>
   <Macros NG="7">
      <material ID="1" NA="0" fissile="true">
        <name>UO2 fuel-clad</name>
        <TotalXS>1.77949E-01 3.29805E-01 4.80388E-01 5.54367E-01 3.11801E-01 3.95168E-01 5.64406E-01</TotalXS>
        <NuFissionXS>
          2.0059984287E-02 2.0273029734E-03 1.5705991756E-02 4.5183010240E-02 4.3342083920E-02 2.0209009624E-01 5.2571053520E-01
        </NuFissionXS>
        <ChiXS>5.87910E-01 4.11760E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiXS>
        <FissionXS>7.21206E-03 8.19301E-04 6.45320E-03 1.85648E-02 1.78084E-02 8.30348E-02 2.16004E-01</FissionXS>
        <KappaFissionXS>7.21206E-03 8.19301E-04 6.45320E-03 1.85648E-02 1.78084E-02 8.30348E-02 2.16004E-01</KappaFissionXS>
        <Profile>
          1 1
          1 2
          1 3
          1 5
          4 6
          5 7
          5 7
        </Profile>
        <ScatteringXS>
           1.27537E-01
           4.23780E-02 3.24456E-01
           9.43740E-06 1.63140E-03 4.50940E-01
           5.51630E-09 3.14270E-09 2.67920E-03 4.52565E-01 1.25250E-04
           5.56640E-03 2.71401E-01 1.29680E-03
           1.02550E-02 2.65802E-01 8.54580E-03
           1.00210E-08 1.68090E-02 2.73080E-01
        </ScatteringXS>
      </material>
      <material ID="2" NA="0" fissile="true">
        <name>4.3 percent MOX fuel-clad</name>
        <TotalXS>1.78731E-01 3.30849E-01 4.83772E-01 5.66922E-01 4.26227E-01 6.78997E-01 6.82852E-01</TotalXS>
        <NuFissionXS>
          2.1753004514E-02 2.5351033490E-03 1.6267991481E-02 6.5474099656E-02 3.0724087845E-02 6.6665096155E-01 7.1399043040E-01
        </NuFissionXS>
        <ChiXS>5.87910E-01 4.11760E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiXS>
        <FissionXS>7.62704E-03 8.76898E-04 5.69835E-03 2.28872E-02 1.07635E-02 2.32757E-01 2.48968E-01</FissionXS>
        <KappaFissionXS>7.62704E-03 8.76898E-04 5.69835E-03 2.28872E-02 1.07635E-02 2.32757E-01 2.48968E-01</KappaFissionXS>
        <Profile>
          1 1
```

```
    1 2
    1 3
    1 5
    4 6
    5 7
    5 7
  </Profile>
  <ScatteringXS>
     1.28876E-01
    4.14130E-02 3.25452E-01
    8.22900E-06 1.63950E-03 4.53188E-01
    5.04050E-09 1.59820E-09 2.61420E-03 4.57173E-01 1.60460E-04
    5.53940E-03 2.76814E-01 2.00510E-03
    9.31270E-03 2.52962E-01 8.49480E-03
    9.16560E-09 1.48500E-02 2.65007E-01
  </ScatteringXS>
</material>
<material ID="3" NA="0" fissile="true">
  <name>7.0 percent MOX fuel-clad</name>
  <TotalXS>1.81323E-01 3.34368E-01 4.93785E-01 5.91216E-01 4.74198E-01 8.33601E-01 8.53603E-01</TotalXS>
  <NuFissionXS>
    2.3813952011E-02 3.8586887635E-03 2.4134001354E-02 9.4366219990E-02 4.5769876104E-02 9.2818140452E-01 1.0432001182E+00
  </NuFissionXS>
  <ChiXS>5.87910E-01 4.11760E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiXS>
  <FissionXS>8.25446E-03 1.32565E-03 8.42156E-03 3.28730E-02 1.59636E-02 3.23794E-01 3.62803E-01</FissionXS>
  <KappaFissionXS>8.25446E-03 1.32565E-03 8.42156E-03 3.28730E-02 1.59636E-02 3.23794E-01 3.62803E-01</KappaFissionXS>
  <Profile>
    1 1
    1 2
    1 3
    1 5
    4 6
    5 7
    5 7
  </Profile>
  <ScatteringXS>
     1.30457E-01
    4.17920E-02 3.28428E-01
    8.51050E-06 1.64360E-03 4.58371E-01
    5.13290E-09 2.20170E-09 2.53310E-03 4.63709E-01 1.76190E-04
    5.47660E-03 2.82313E-01 2.27600E-03
    8.72890E-03 2.49751E-01 8.86450E-03
    9.00160E-09 1.31140E-02 2.59529E-01
  </ScatteringXS>
</material>
<material ID="4" NA="0" fissile="true">
  <name>8.7 percent MOX fuel-clad</name>
  <TotalXS>1.83045E-01 3.36705E-01 5.00507E-01 6.06174E-01 5.02754E-01 9.21028E-01 9.55231E-01</TotalXS>
  <NuFissionXS>
    2.5186004103E-02 4.7395094670E-03 2.9478053976E-02 1.1224999848E-01 5.5303012800E-02 1.0749988378E+00 1.2392983699E+00
  </NuFissionXS>
  <ChiXS>5.87910E-01 4.11760E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiXS>
  <FissionXS>8.67209E-03 1.62426E-03 1.02716E-02 3.90447E-02 1.92576E-02 3.74888E-01 4.30599E-01</FissionXS>
  <KappaFissionXS>8.67209E-03 1.62426E-03 1.02716E-02 3.90447E-02 1.92576E-02 3.74888E-01 4.30599E-01</KappaFissionXS>
  <Profile>
    1 1
    1 2
    1 3
    1 5
    4 6
    5 7
    5 7
  </Profile>
  <ScatteringXS>
     1.31504E-01
    4.20460E-02 3.30403E-01
    8.69720E-06 1.64630E-03 4.61792E-01
    5.19380E-09 2.60060E-09 2.47490E-03 4.68021E-01 1.85970E-04
    5.43300E-03 2.85771E-01 2.39160E-03
    8.39730E-03 2.47614E-01 8.96810E-03
    8.92800E-09 1.23220E-02 2.56093E-01
  </ScatteringXS>
</material>
<material ID="5" NA="0" fissile="false">
  <name>Guide tube</name>
  <TotalXS>1.26032E-01 2.93160E-01 2.84240E-01 2.80960E-01 3.34440E-01 5.65640E-01 1.17215E+00</TotalXS>
  <Profile>
    1 1
    1 2
    1 3
    1 5
    1 6
    2 7
    2 7
```

```
    </Profile>
    <ScatteringXS>
       6.61659E-02
       5.90700E-02 2.40377E-01
       2.83340E-04 5.24350E-02 1.83297E-01
       1.46220E-06 2.49900E-04 9.23970E-02 7.88511E-02 3.73330E-05
       2.06420E-08 1.92390E-05 6.94460E-03 1.70140E-01 9.97372E-02 9.17260E-04
       2.98750E-06 1.08030E-03 2.58810E-02 2.06790E-01 3.16765E-01 4.97920E-02
       4.21400E-07 2.05670E-04 4.92970E-03 2.44780E-02 2.38770E-01 1.09912E+00
    </ScatteringXS>
  </material>
  <material ID="6" NA="0" fissile="true">
    <name>fission chamber</name>
    <TotalXS>1.26032E-01 2.93160E-01 2.84250E-01 2.81020E-01 3.34460E-01 5.65640E-01 1.17214E+00</TotalXS>
    <NuFissionXS>
      1.3234010957E-08 1.4344997680E-08 1.1285993022E-06 1.2762993228E-05 3.5385018200E-07 1.7400988536E-06 5.0633018580E-06
    </NuFissionXS>
    <ChiXS>5.87910E-01 4.11760E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiXS>
    <FissionXS>4.79002E-09 5.82564E-09 4.63719E-07 5.24406E-06 1.45390E-07 7.14972E-07 2.08041E-06</FissionXS>
    <KappaFissionXS>4.79002E-09 5.82564E-09 4.63719E-07 5.24406E-06 1.45390E-07 7.14972E-07 2.08041E-06</KappaFissionXS>
    <Profile>
      1 1
      1 2
      1 3
      1 5
      1 6
      2 7
      2 7
    </Profile>
    <ScatteringXS>
       6.61659E-02
       5.90700E-02 2.40377E-01
       2.83340E-04 5.24350E-02 1.83425E-01
       1.46220E-06 2.49900E-04 9.22880E-02 7.90769E-02 3.73400E-05
       2.06420E-08 1.92390E-05 6.93650E-03 1.69990E-01 9.97570E-02 9.17420E-04
       2.98750E-06 1.07900E-03 2.58600E-02 2.06790E-01 3.16774E-01 4.97930E-02
       4.21400E-07 2.05430E-04 4.92560E-03 2.44780E-02 2.38760E-01 1.09910E+00
    </ScatteringXS>
  </material>
  <material ID="7" NA="0" fissile="false">
    <name>Moderator</name>
    <TotalXS>1.59206E-01 4.12970E-01 5.90310E-01 5.84350E-01 7.18000E-01 1.25445E+00 2.65038E+00</TotalXS>
    <Profile>
      1 1
      1 2
      1 3
      1 5
      1 6
      2 7
      2 7
    </Profile>
    <ScatteringXS>
      4.44777E-02
      1.13400E-01 2.82334E-01
      7.23470E-04 1.29940E-01 3.45256E-01
      3.74990E-06 6.23400E-04 2.24570E-01 9.10284E-02 7.14370E-05
      5.31840E-08 4.80020E-05 1.69990E-02 4.15510E-01 1.39138E-01 2.21570E-03
      7.44860E-06 2.64430E-03 6.37320E-02 5.11820E-01 6.99913E-01 1.32440E-01
      1.04550E-06 5.03440E-04 1.21390E-02 6.12290E-02 5.37320E-01 2.48070E+00
    </ScatteringXS>
  </material>
  <material ID="8" NA="0" fissile="false">
    <name>control rod</name>
    <TotalXS>2.16768E-01 4.80098E-01 8.86369E-01 9.70009E-01 9.10482E-01 1.13775E+00 1.84048E+00</TotalXS>
    <Profile>
      1 1
      1 2
      1 3
      1 5
      4 6
      5 7
      5 7
    </Profile>
    <ScatteringXS>
      1.7056E-01
      4.4401E-02 4.7105E-01
      9.8367E-05 6.8548E-04 8.0186E-01
      1.2779E-07 3.9140E-10 7.2013E-04 5.7075E-01 6.5556E-05
      1.4602E-03 2.0784E-01 1.0243E-03
      3.8149E-03 2.0247E-01 3.5304E-03
      3.6976E-09 4.7529E-03 6.5860E-01
    </ScatteringXS>
  </material>
</Macros>
```

```
</Materials>
```

It is noted that the benchmark provides the averaged neutrons emitted per fission ν and fission cross section $\Sigma_{f,g}, g = 1, \cdots, 7$. INSTANT format requires $\nu\Sigma_{f,g}$, which needs more precision than 6 to keep the data consistent. We also provide fake $\kappa\Sigma_{f,g}$ for making postprocessing easier. Note, INSTANT cross sections formats can be loaded into *ConstantNeutronicsMaterial* representing cross sections that do not depend on state variables such as temperature or burnup. The block names M-$< n >$-TRI are assigned automatically by the INSTANT mesh generator.

The Materials block is listed below:

```
[Materials]

  [./uo2]
    type = ConstantNeutronicsMaterial
    block = 'M-1-TRI'
    fromFile = true
    fileName = c5g7_materials.xml
    material_id = 1
  [../]
  [./mox4.3]
    type = ConstantNeutronicsMaterial
    block = 'M-2-TRI'
    fromFile = true
    fileName = c5g7_materials.xml
    material_id = 2
  [../]
  [./mox7.0]
    type = ConstantNeutronicsMaterial
    block = 'M-3-TRI'
    fromFile = true
    fileName = c5g7_materials.xml
    material_id = 3
  [../]
  [./mox8.7]
    type = ConstantNeutronicsMaterial
    block = 'M-4-TRI'
    fromFile = true
    fileName = c5g7_materials.xml
    material_id = 4
  [../]
  [./GuideTube]
    type = ConstantNeutronicsMaterial
    block = 'M-5-TRI'
    fromFile = true
    fileName = c5g7_materials.xml
    material_id = 5
  [../]
  [./FissionChamber]
    type = ConstantNeutronicsMaterial
    block = 'M-6-TRI'
    fromFile = true
    fileName = c5g7_materials.xml
    material_id = 6
  [../]
  [./moderator]
    type = ConstantNeutronicsMaterial
    block = 'M-7-TRI'
    fromFile = true
```

```
    fileName = c5g7_materials.xml
    material_id = 7
  [../]

[]
```

We can use GlobalParams to reduce the size of the materials block.

```
[GlobalParams]
  fromFile = true
  fileName = c5g7_materials.xml
[]

[Materials]
  [./uo2]
    type = ConstantNeutronicsMaterial
    block = 'M-1-TRI'
    material_id = 1
  [../]
  [./mox4.3]
    type = ConstantNeutronicsMaterial
    block = 'M-2-TRI'
    material_id = 2
  [../]
  [./mox7.0]
    type = ConstantNeutronicsMaterial
    block = 'M-3-TRI'
    material_id = 3
  [../]
  [./mox8.7]
    type = ConstantNeutronicsMaterial
    block = 'M-4-TRI'
    material_id = 4
  [../]
  [./GuideTube]
    type = ConstantNeutronicsMaterial
    block = 'M-5-TRI'
    material_id = 5
  [../]
  [./FissionChamber]
    type = ConstantNeutronicsMaterial
    block = 'M-6-TRI'
    material_id = 6
  [../]
  [./moderator]
    type = ConstantNeutronicsMaterial
    block = 'M-7-TRI'
    material_id = 7
  [../]
[]
```

3.8.7 S_N Postprocessor Block

The SweepUpdate Executioner requires the user to provide a postprocessor measuring the convergence as the norm of the difference between two iterates. In this example, we opt to use the fission source as the variable for checking converge. The ElementL2Diff postprocessor that computes an L2 norm of the difference of successive

iterates is used. As variable, the fission_source, that is added automatically if particle type neutron is selected, is provided. Execution must be on linear.

```
[Postprocessors]
  [./fsrc_diff]
    type = ElementL2Diff
    block = 'M-1-TRI M-2-TRI M-3-TRI M-4-TRI M-5-TRI M-6-TRI M-7-TRI'
    variable = fission_source
    execute_on = linear
  [../]
[]
```

3.8.8 S_N Executioner Block

The SweepUpdate executioner must be used. For running NDA efficiently, only a single richardson iteration (richardson_max_its = 1) should be used triggering only do a single transport update after each diffusion solve. For forcing the execution of a single transport update, the absolute tolerance should be set to a very small value. xdiff takes the name of a postprocessor for evaluating the progress of the iteration; in this case the postprocessor defined in 3.8.7 is used. Note: The first order S_N multiapp setup uses the postprocessor provided in the sub app's executioner block for determining convergence of the NDA Picard iteration.

```
[Executioner]
   type = SweepUpdate
   richardson_max_its = 1
   richardson_abs_tol = 1.0e-16
  xdiff = fsrc_diff
[]
```

3.8.9 S_N Outputs Block

Nothing special needs to be considered in the Outputs block. csv output is set to true to study the convergence history.

```
[Outputs]
  file_base = c5g7_2d_out
  exodus = false
  csv = true
[]
```

3.8.10 Diffusion Mesh Block

NDA allows the diffusion mesh to be coarser than the S_N mesh as long as it is nested within it. A mesh is nested in another mesh if each S_N mesh element is within a single diffusion mesh element. In this tutorial two diffusion meshing options are discussed: fine mesh (fm) uses identical diffusion and S_N meshes, while coarse mesh (cm) uses a structured quadrilaterial mesh for the diffusion calculation where each element comprises a single pin cell (Note: for this to work the S_N mesh was prepared so no triangle crosses pin-cell boundaries). The fine mesh input file is named c5g7_master_cm.i, and the coarse mesh input file is named c5g7_master_fm.i.

Fine mesh Mesh block:

```
[Mesh]
 file = c5g7-2d-pccm.e
[]
```

Coarse mesh Mesh block:

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  xmin = 0
  xmax = 64.26
  ymin = 0
  ymax = 64.26
  elem_type = QUAD4
  nx = 51
  ny = 51
  uniform_refine = 0
[]
```

3.8.11 Diffusion TransportSystems Block

The keyword triggering the execution of the first order NDA solve is transport_multiapp_file that must match the name of the sub app input file. The number of groups G should be equal to the number of groups in the sub app solve (G=7 in this case). The number of energy groups used in the diffusion solve is determined by the parameter group_collapsing. group_collapsing is an array of integers of length G. Using C++ indexing, the entry at position g is the diffusion energy group index of S_N energy group g. For using the same group structure, '0 1 2 3 4 5 6' is used, while for a two-group diffusion solve with the first 4 S_N groups being collapsed into diffusion group 0 and the last 3 groups being collapsed into group 1 '0 0 0 0 1 1 1' is used. DGtype must be set to IIP and NDA_type determines the type of NDA closure: pcmfd should usually be used. The only difference between the fine mesh and coarse mesh case are the boundary conditions. In the fine mesh case, the boundaries are named in the mesh generation process, while in the coarse mesh case the boundaries are named/numbered automatically by the GeneratedMesh object.

Fine mesh TransportSystems block:

```
[TransportSystems]
  particle = neutron
  G = 7
  VacuumBoundary = 'vacuum'
  ReflectingBoundary = 'reflecting'
  equation_type = eigenvalue
  [./diff_dfem]
    scheme = DFEM-Diffusion
    family = MONOMIAL
    order = FIRST
    NDA_type=pcmfd
    transport_multiapp_file = 'c5g7_sub.i'
    G = 7
    # same group structure
    group_collapsing = '0 1 2 3 4 5 6'
    # diffusion system uses two groups
    # group_collapsing = '0 0 0 0 1 1 1'
    DGtype = IIP
  [../]
```

```
[]
```

Coarse mesh TransportSystems block:

```
[TransportSystems]
  particle = neutron
  VacuumBoundary = '0 3'
  ReflectingBoundary = '1 2'
  equation_type = eigenvalue
  [./diff_dfem]
    scheme = DFEM-Diffusion
    family = MONOMIAL
    order = FIRST
    G = 7
    NDA_type=pcmfd
    transport_multiapp_file = 'c5g7_sub.i'
    # same group structure
    group_collapsing = '0 1 2 3 4 5 6'
    # diffusion system uses two groups
    # group_collapsing = '0 0 0 0 1 1 1'
    DGtype = IIP
  [../]
[]
```

3.8.12 Diffusion Materials Block

Do not provide a Materials block. Materials are added automatically and computed from the S_N solution.

3.8.13 Diffusion Executioner Block

The DFEMNDAEigenExecutioner is used that triggers a Picard iteration scheme for the solution of the NDA eigenvalue equations. The parameter max_steps determines the maximum number of Picard iterations and tol is the convergence tolerance that the iterative error given by the sub app postprocessor specified in section 3.8.7 must satisfy for convergence.

Differences exist in the fine mesh and coarse mesh executioner blocks as the convergence properties of these two cases are very different. The preconditioner of choice is usually algebraic multigrid, i.e. hypre boomeramg as in the coarse mesh case. However, it only works well for elliptic system, but the low order NDA system is an advection-diffusion system because of the advection-like closure terms. In the fine mesh case, the advective terms are too strong for AMG to work properly so the additive Schwartz method is used instead. In addition three free power iteration are required to guarantee convergence to the fundamental mode.

For the coase mesh case AMG can be used, but with increasing the number of concurrent parallel processes, more free power iteration and possibly more V-cycles are required to ensure appropriate convergence of the diffusion system. For both cases an alternative set of PETSc parameters is available for serial execution using the direct inversion of the linear system via LU decomposition.

Fine mesh TransportSystems block:

```
[Executioner]
  type = DFEMNDAEigenExecutioner
  # NDA params
  max_steps = 50
```

```
  tol = 1.0e-6
  # PJFNK diffusion solve
  nl_max_its = 20
  l_max_its  = 100
  l_tol = 1e-05
  source_abs_tol = 1.0e-9
  source_rel_tol = 1.0e-6
  free_power_iterations = 0
  extra_free_pi = 3
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -ksp_gmres_restart  -sub_pc_type -pc_asm_overlap'
  petsc_options_value = 'asm      50                   lu           1'
  # the following PETSc parameters only work in serial
  # petsc_options_iname = '-pc_type -ksp_type -ksp_gmres_restart'
  # petsc_options_value = ' lu       gmres     50'
[]
```

Coarse mesh TransportSystems block:

```
[Executioner]
  type = DFEMNDAEigenExecutioner
  # NDA params
  max_steps = 50
  tol = 1.0e-6
  # PJFNK diffusion solve
  nl_max_its = 20
  l_max_its  = 100
  l_tol = 1e-05
  source_abs_tol = 1.0e-9
  source_rel_tol = 1.0e-6
  free_power_iterations = 0
  extra_free_pi = 1
  # for execution on more processor these settings might be necessary
  # extra_free_pi = 3
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_type -ksp_gmres_restart'
  petsc_options_value = ' hypre boomeramg       gmres     50'
  # for execution on more processor these settings might be necessary
  # petsc_options_iname = '-pc_type -pc_hypre_type -ksp_type -ksp_gmres_restart
     -pc_hypre_boomeramg_max_iter -pc_hypre_boomeramg_rtol'
  # petsc_options_value = ' hypre boomeramg       gmres     50
     6                           0.0'
  # the following PETSc parameters only work in serial
  # petsc_options_iname = '-pc_type -ksp_type -ksp_gmres_restart'
  # petsc_options_value = ' lu       gmres     50'
[]
```

3.8.14 Diffusion Outputs Block

No special NDA parameters are required for diffusion Outputs block. print_linear_residuals is set to true for detecting problem with the linear convergence in the diffusion system. csv and exodus outputs are requested.

```
[Outputs]
  file_base = c5g7_cm_out
```

```
  print_linear_residuals = true
  exodus = true
  csv = true
[]
```

3.8.15 Executing the Input

You have to make sure that the yak library path is correctly set. Then the diffusion problem file is executed using Rattlesnake.

```
export MOOSE_LIBRARY_PATH=/path/to/libyak-opt.la
./rattlesnake-opt -i c5g7_master_cm.i
./rattlesnake-opt -i c5g7_master_fm.i
```

3.8.16 Results

Results for this tutorial can be found in Ref. [14] and [15].

3.9 Coupled reactor calculation

3.9.1 Problem description

Geometry of this tutorial problem is illustrated in Fig. 15. Sizes of all regions are marked on the figure and are

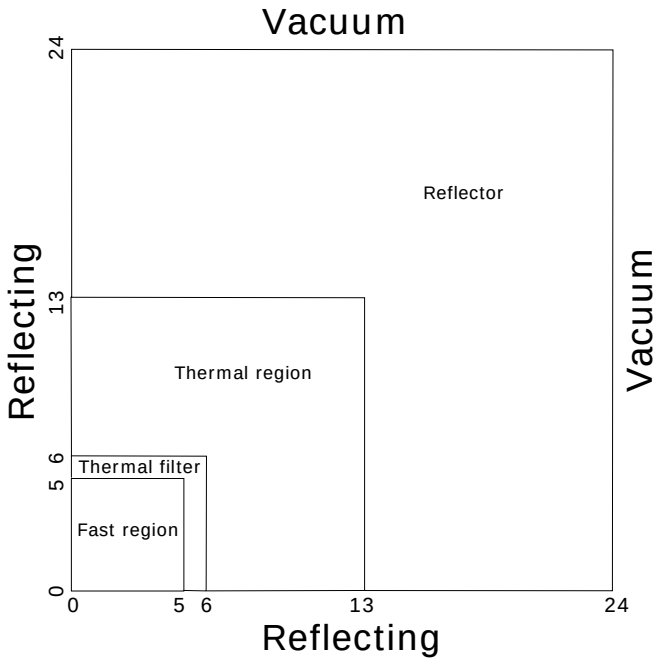


Figure 15 Geometry of the coupled reactor.

in unit of centimeter. The region with the fast spectrum is surrounded by a thermal neutron filter which filters the neutrons from the region next to it with the thermal spectrum. Thermal region is surrounded by the outer reflector. The simple two-group cross sections for demonstrating the capability of multi-region calculations

of these four regions are given in Table 15. All cross sections in the table are in unit of $1/cm$. The group averaged neutron velocities are constant for all four regions. The fast velocity is $4.34311E+09cm/s$ and the thermal velocity is $2.82595E+06cm/s$. Fission spectrum of the fast and thermal regions are the same, $\chi_1 = 1$ and $\chi_2 = 0$. Delayed neutron data can be included in the calculation although they are not presenting in this problem. The left and bottom boundaries are reflective and right and top boundaries have zero incoming neutron flux. We want to calculate the coupled parameters of the fast and thermal regions.

Table 15 Cross sections of the coupled reactor.

Region	g	$\Sigma_{t,g}$ $\Sigma_{t,g}$	$\Sigma_{s,0,g'\to g}$ $g'=1$	$g'=2$	$\Sigma_{s,1,g'\to g}$ $g'=1$	$g'=2$	$\nu\Sigma_{f,g}$
Fast	1	3.79586E-01	3.4788E-01	0.0000E+00	1.0628E-01	0.0000E+00	7.19278E-02
	2	6.41032E-01	3.8229E-09	3.1800E-01	0.8246E-09	1.0903E-03	3.89643E-01
Thermal	1	5.68247E-01	5.1603E-01	0.0000E+00	3.1445E-01	0.0000E+00	2.50614E-03
	2	1.88134E+00	5.0241E-02	1.7937E+00	0.9285E-02	0.7976E+00	1.32816E-01
Filter	1	3.16675E-01	2.4923E-01	0.0000E+00	3.9400E-02	0.0000E+00	-
	2	2.90484E+01	2.8582E-11	2.0710E+00	-0.7873E-12	1.9424E-02	-
Reflector	1	3.77860E-01	3.7174E-01	0.0000E+00	2.8487E-02	0.0000E+00	-
	2	4.90661E-01	6.1056E-03	4.9052E-01	-1.4008E-03	2.5143E-02	-

3.9.2 The stand-alone input file

We first create the stand-alone input file for the eigenvalue calculation of this problem.

```
[Mesh]
  type = CartesianMesh
  dim = 2
  dx = '5 1 7 11'
  ix = '5 1 7 11'
  dy = '5 1 7 11'
  iy = '5 1 7 11'
  uniform_refine = 1
  subdomain_id = '
   1 2 3 4
   2 2 3 4
   3 3 3 4
   4 4 4 4'
[]

[TransportSystems]
  particle = neutron
  equation_type = eigenvalue

  G = 2

  VacuumBoundary = 'right top'
  ReflectingBoundary = 'left bottom'

  [./diff]
    scheme = SAAF-CFEM-SN
    AQorder = 8
    AQtype = Level-Symmetric
    NA = 1
    family = LAGRANGE
    order = FIRST
    fission_source_as_material = true
```

```
  [../]
[]

[Materials]
  [./fast_reigon]
    type = ConstantNeutronicsMaterial
    block = 1

    sigma_t = '3.79586E-01 6.41032E-01'
    L = 1
    sigma_s = '3.4788E-01  0.0000E+00
               3.8229E-09  3.1800E-01
               1.0628E-01  0.0000E+00
               0.8246E-09  1.0903E-03'
    diffusion_coef = '1.2196 0.5209'
    fissile        = true
    nu_sigma_f     = '7.19278E-02 3.89643E-01'
    kappa_sigma_f  = '8.50985E-13 5.00750E-12'
    chi            = '1.0 0.0'
    sigma_r        = '3.1706E-02 3.23032E-01'
  [../]
  [./thermal_neutron_filter]
    type = ConstantNeutronicsMaterial
    block = 2

    sigma_t = '3.16675E-01 2.90484E+01'
    L = 1
    sigma_s = '2.4923E-01  0.0000E+00
               2.8582E-11  2.0710E+00
               3.9400E-02  0.0000E+00
              -0.7873E-12  1.9424E-02'
    diffusion_coef = '1.2022 0.0115'
    fissile        = false
    sigma_r        = '6.7445E-02 2.69774E+01'
  [../]
  [./thermal_reigon]
    type = ConstantNeutronicsMaterial
    block = 3

    sigma_t = '5.68247E-01 1.88134E+00'
    L = 1
    sigma_s = '5.1603E-01  0.0000E+00
               5.0241E-02  1.7937E+00
               3.1445E-01  0.0000E+00
               0.9285E-02  0.7976E+00'
    diffusion_coef = '1.3134 0.3076'
    fissile        = true
    nu_sigma_f     = '2.50614E-03 1.32816E-01'
    kappa_sigma_f  = '3.63645E-14 1.91727E-12'
    chi            = '1.0 0.0'
    sigma_r        = '5.2217E-02 8.7640E-02'
  [../]
  [./outer_reflector]
    type = ConstantNeutronicsMaterial
    block = 4

    sigma_t = '3.77860E-01 4.90661E-01'
    L = 1
    sigma_s = '3.7174E-01  0.0000E+00
               6.1056E-03  4.9052E-01
               2.8487E-02  0.0000E+00
```

```
            -1.4008E-03  2.5143E-02'
    diffusion_coef = '0.9541 0.7160'
    fissile        = false
    sigma_r        = '6.7445E-02 2.69774E+01'
  [../]
[]

[Postprocessors]
  [./fluxintegral]
    type = ElementIntegralVariablePostprocessor
    variable = flux_moment_g0_L0_M0
    execute_on = linear
  [../]
[]

[Executioner]
  type = NonlinearEigen

  free_power_iterations = 4
  source_abs_tol = 1e-12
  output_before_normalization = false
  output_after_power_iterations = false

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
[]

[Outputs]
  exodus = true
[]
```

We used the *CartesianMesh* mesh because the geometry of this tutorial is regular. We use S_8 level symmetric quadrature in the input. Finer angular resolution will be needed to reduce the angular discretization error. We need to set *NA* to 1 to accommodate the linear anisotropic scattering cross sections. We added one post-processor to obtain the integral of the fast flux over the entire domain for a sanction check explained later. Diffusion coefficients and removal cross sections in the materials are not necessary. We keep them only because it will make switching transport schemes to diffusion schemes easy. Diffusion coefficients are evaluated with $\frac{1}{3(\Sigma_{t,g}-\Sigma_{s,1,g\rightarrow g}}$). Removal cross sections are evaluated with $\Sigma_{t,g} - \sum_{g'=1}^{G} \Sigma_{s,0,g\rightarrow g'}$. The rest input parameters are fairly strait-forward and their meanings will not be repeated.

3.9.3 The master input file

Then we change the stand-alone input file to create the master input file for driving the multi-region calculation. The changes are quite simple.

We will first need to add the following three parameters for all materials. Because these parameters are the same for all materials, we use *GlobalParams* to add them.

```
[GlobalParams]
  # let neutronics material accept neutron speed
  forTransient = true
  neutron_speed  = '4.34311e9 2.82595e6'
  # let neutronics material evaluate delayed and prompt spectrum
  plus = true
```

```
[]
```

These parameters will ensure additional material properties are declared and evaluated by neutronics materials for the multi-region calculation.

We then change the executioner type to *PicardEigen* so that *MultiApps* for the full/partial adjoint and partial forward calculations on *timestep_begin* and *timestep_end* will be invoked.

```
[Executioner]
  type = PicardEigen
  ...
[]
```

PicardEigen shares most of parameters of *NonlinearEigen*, except the parameters for Picard iterations. In this calculation, we only need it to evaluate MultiApps on *timestep_begin* and *timestep_end* once and no Picard iterations are needed. So default values of all those parameters can be used.

Finally we use the *MultiRegion* input block to add the *MultiApps* and tell Rattlesnake to do the multi-region calculation.

```
[MultiRegion]
  transport_system = diff
  regions = '1 3'
  adjoint_multiapp_file = adjoint_2g.i
  adjoint_partial_multiapp_files = 'aslave1_2g.i aslave2_2g.i'
  forward_partial_multiapp_files = 'slave1_2g.i slave2_2g.i'
  csv_file = params.csv
[]
```

Block 1 and 3 are the two regions. *csv_file* makes the evaluated parameters dumped into the CSV file. It is used for automatic testing.

3.9.4 The input file for the full adjoint calculation

We need to create the input file for the full adjoint calculation. This can be done by modifying the stand-alone forward input file. We simply add the following two parameters in the *TransportSystems* block:

```
[TransportSystems]
  ...
  for_adjoint = true
  for_math_adjoint = true
  ...
```

We need to make sure we are indeed doing the *mathematical* adjoint calculation by setting *for_math_adjoint* to true.

3.9.5 The input file for the partial forward calculations

The partial forward calculations use the fission source from the master solve and evaluate the fluxes from this source with source problem calculations. To do this, starting from the stand-alone input file, we need to tell the transport system that

```
[TransportSystems]
  ...
  equation_type = steady-state
  ...
  [./diff]
    ...
    explicit_fission = true
  [../]
[]
```

We then need to use *Richardson* executioner instead the *NonlinearEigen*.

```
[Executioner]
  type = Richardson
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart '
  petsc_options_value = 'hypre boomeramg 100'
  nl_rel_tol = 1e-12
[]
```

We also need to disable the fission in the other regions for the partial calculation of one particular region. This can be done by setting the parameter *disable_fission* of the materials of the other regions to true. For example, for the partial solve of the fast region, we need to set

```
[Materials]
  ...
  [./thermal_region]
    ...
    disable_fission = true
  [../]
  ...
[]
```

3.9.6 The input file for the partial adjoint calculations

The partial adjoint calculations use the adjoint fission source from the full adjoint solve and evaluate the adjoint fluxes from this source with source problem calculations. These files can be created easily by modifying the existing input files for the partial forward calculations. We simply need to add

```
[TransportSystems]
  ...
  for_adjoint = true
  for_math_adjoint = true
  ...
```

for the corresponding partial forward input files.

3.9.7 Results

The mesh and angular quadrature need to be much refined to get a solution without too much discretization errors for this simple problem. However, we will just present the screen output generated with the above settings. One simple sanction check can be done by adding all fast flux postprocessors of all partial calculations, and checking if the result is equal to the full eigenvalue solve.

```
******************************
   Coupled reactor parameters
******************************

******************************
 rho and Lambda
******************************

  n  m         rho         Lambda
---------------------------------
  0  0  0.66447997 8.40803974e-09
  0  1  0.11528978 4.05670979e-08
  1  0  0.54174692 4.67042342e-06
  1  1  0.81384753 5.36178553e-06
---------------------------------

******************************
 rho derived from weak form
******************************

  n  m       (0,0)       (0,1)       (1,0)       (1,1)
-----------------------------------------------------
  0  0  0.70056919  0.68528197
  0  1                          0.12026290  0.10806838
  1  0  0.51949850  0.54380253
  1  1                          0.81192562  0.83183280
-----------------------------------------------------
```

The eigenvalues of both ρ matrices, $\rho_{n,m}, m = 1,2; n = 1,2$ is indeed 1.000000.

3.10 A problem demonstrates YAKXS

Simple criticality search with temperature change and control rod change with the tabulated cross sections in YAKXS format. (To be added.)

3.11 A thermal radiation benchmark

(To be added.)

4 *TransportSystems*

Parameters in this block and its sub-blocks are used to describe the transport equation and the employed type of discretization scheme. This block is the only one that does not follow the MOOSE syntax. It is meant to automatically setup many other MOOSE syntax blocks based on the options that are used within this block and sub-blocks.

4.1 *particle*

Description: Particle type of the transport system

Data type: Enumeration (/common/neutron/thermal/)

Default value: <required>

Syntax: TransportSystems/particle

Note: This parameter can only be common, neutron or thermal at this moment. We will add more particle types here for Rattlesnake in the future. If the particle is a type other than common, additional parameters in Neutron or Thermal determined by this parameter are available to append parameters specified in Discretization Schemes. When particle type is neutron, almost all schemes can be chosen except DFEM-PN. When particle type is thermal, only DFEM-PN, SAAF-CFEM-SN and CFEM-Diffusion schemes can be used.

4.2 *equation_type*

Description: Type of the transport equation

Data type: Enumeration (/steady-state/transient/eigenvalue/)

Default value: <required>

Syntax: TransportSystems/equation_type

4.3 *for_adjoint*

Description: Switch between (adjoint/primal) transport equation

Data type: Logical

Default value: False

Syntax: TransportSystems/for_adjoint

4.4 *for_math_adjoint*

Description: Switch between (mathematical/physical) adjoint.

Data type: Logical

Default value: True

Syntax: TransportSystems/for_math_adjoint

Note: Parameter is active only if for_adjoint is true. Mathematical adjoint of schemes, including CFEM-Diffusion, DFEM-SN and DFEM-Diffusion with *DGType* being equal to SIP, are the same as their physical adjoint. Mathematical adjoint of the rest schemes are different from the physical adjoint. This parameter can be set to true only for CFEM-Diffusion, DFEM-SN, DFEM-Diffusion with *DGType* being equal to SIP, or SAAF-CFEM-SN.

4.5 *G*

Description: Number of energy groups or bands

Data type: Integer

Default value: 1

Syntax: TransportSystems/G

4.6 Boundary condition

Rattlesnake uses mesh side sets to indicate where the boundary conditions are applied. This manual makes no differences between mesh side sets and boundaries. Users provide a list of names of side sets contained in the mesh for a set of supported boundary condition types, which are listed in the following subsections.

4.6.1 *DirichletBoundary*

Description: Dirichlet boundaries

Data type: Vector of boundary names

Default value: <empty>

Syntax: TransportSystems/DirichletBoundary

Note: This boundary condition type is *only* supported for diffusion approximations: CFEM-Diffusion or DFEM-Diffusion.

4.6.2 *DirichletValue*

Description: Constant scalar fluxes values on the Dirichlet boundaries

Data type: Vector of real

Default value: <empty>

Syntax: TransportSystems/DirichletValue

Note: This parameter depends on DirichletBoundary and G. The number of values must be equal to the number of side sets in DirichletBoundary times G. Values are ordered by groups then by boundaries. It is noted that we are considering to use function for setting Dirichlet values.

4.6.3 *VacuumBoundary*

Description: the vacuum or surface source boundaries

Data type: Vector of boundary names

Default value: <empty>

Syntax: TransportSystems/VacuumBoundary

4.6.4 *SourceDirection*

Description: Angular direction index of all non-zero surface source vacuum boundaries for all groups.

Data type: Vector of integers

Default value: <empty>

Syntax: TransportSystems/SourceDirection

Note: This parameter depends on VacuumBoundary and G. The number of values must be equal to the number of side sets in VacuumBoundary times G. Values are ordered by group then by boundaries. It is noted that only one non-zero surface source direction is currently allowed for a group. This parameter is *invalid* for diffusion approximations: CFEM-Diffusion or DFEM-Diffusion.

4.6.5 *SurfaceSource*

Description: Constant angular flux values on the vacuum boundaries

Data type: Vector of real

Default value: <empty>

Syntax: TransportSystems/SurfaceSource

Note: This parameter depends on VacuumBoundary, G and SourceDirection. The number of values must be equal to the number of side sets in VacuumBoundary times G. Values are ordered by group then by the boundaries. If SourceDirection is given, values here are treated as angular fluxes at those particular directions specified in SourceDirection, otherwise they are the value for all incoming directions. This might change in the future and allow functions for setting surface sources.

4.6.6 *ReflectingBoundary*

Description: Specular reflecting boundaries

Data type: Vector of boundary names

Default value: <empty>

Syntax: TransportSystems/ReflectingBoundary

4.6.7 *WhiteBoundary*

Description: White boundaries

Data type: Vector of boundary names

Default value: <empty>

Syntax: TransportSystems/WhiteBoundary

Note: This boundary condition type is *invalid* for diffusion approximations: CFEM-Diffusion or DFEM-Diffusion.

4.7 Volumetric source

Rattlesnake uses mesh blocks for specifying external volumetric sources. This manual makes no differences between mesh blocks and subdomains. Users provide a list of names of subdomains contained in the mesh and either a vector source strength values or a transport functions on each block.

4.7.1 *VolumetricSourceBlock*

Description: Subdomains on which non-homogeneous volumetric sources are to be specified

Data type: Vector of subdomain names

Default value: <empty>

Syntax: TransportSystems/VolumetricSourceBlock

4.7.2 *VolumetricSource*

Description: Strengths of volumetric sources

Data type: Vector of real

Default value: <empty>

Syntax: TransportSystems/VolumetricSource

Note: This parameter depends on VolumetricSourceBlock and G. The number of values must be equal to the number of blocks in VolumetricSourceBlock times G. Values are ordered by group then by the blocks. It is noted that we are considering to use function for setting sources and LS-CFEM-PN and DFEM-PN already switched to using VolumetricSourceFunc.

4.7.3 *VolumetricSourceFunc*

Description: Transport functions of volumetric sources

Data type: Vector of function names

Default value: <empty>

Syntax: TransportSystems/VolumetricSourceFunc

Note: This parameter depends on VolumetricSourceBlock. The number of values must be equal to the number of blocks in VolumetricSourceBlock. Functions used in this parameter must be in type of *TransportFunction*. Only LS-CFEM-PN is currently using this parameter.

4.7.4 *PointSourceLocation*

Description: X-Y-Z coordinates of a volumetric point source

Data type: Vector of real

Default value: <empty>

Syntax: TransportSystems/PointSourceLocation

Note: The number of real numbers needs to be consistent with the dimension of the mesh. Only SAAF-CFEM-PN is currently using this parameter.

4.7.5 *PointSourceValue*

Description: Strength of the volumetric point source

Data type: Vector of real

Default value: <empty>

Syntax: TransportSystems/PointSourceValue

Note: The number of values must be equal to the number of groups G. Only SAAF-CFEM-PN is currently using this parameter.

4.8 Multiscale transport

Rattlesnake supports multiscale transport simulations. For multiscale transport, different solution schemes can be assigned to different subdomains, including different angular discretizations, e.g. SN (discrete ordinates method) or PN (spherical harmonics expansion method), varying number of energy groups, and different levels of spatial resolution. Subdomain interfaces are automatically created by Rattlesnake and within this manual are sometimes referred to as mortar interfaces. We are in the early stage of this development and future changes are likely. Please always refer your latest manual for the multiscale parameters contained in this section.

4.8.1 *is_mesh_split*

Description: Whether or not the mesh is conforming

Data type: Logical

Default value: False

Syntax: TransportSystems/is_mesh_split

Note: If users are providing a conforming mesh for the multiscale simulation with more than one subdomain, this parameter must be true. If the mesh has been properly split for the multiscale simulation, this parameter must be false.

4.8.2 *show_multiscale_actions*

Description: Whether or not to show the actions setting up the multiscale system

Data type: Logical

Default value: False

Syntax: TransportSystems/show_multiscale_actions

4.8.3 *hide_mortar_variables*

Description: Whether or not to hide variables defined on mortar interfaces from output system

Data type: Logical

Default value: False

Syntax: TransportSystems/hide_mortar_variables

4.8.4 *angular_constraint_type*

Description: How SN-PN interface condition are imposed

Data type: Enumeration (/under/)

Default value: under

Note: Advanced parameter. Users should use default unless they really know what they are doing.

Syntax: TransportSystems/angular_constraint_type

4.8.5 *evaluate_mortar_aux*

Description: Whether or not to evaluate mortar auxiliary variables

Data type: Logical

Default value: false

Syntax: TransportSystems/evaluate_mortar_aux

Note: Advanced parameter. Users should use default unless they really know what they are doing.

4.9 Discretization schemes

The user must at least provide one discretization scheme sub-block to specify the discretization schemes applied for the parent transport system. More than one sub-blocks will invoke the multiscale transport capability.

4.9.1 *scheme*

Description: Scheme used to discretize the transport equation

Data type: Enumeration (see option in this note)

Default value: <required>

Syntax: TransportSystems/*/scheme

Note: The scheme parameter in the sub-block specifies the discretization scheme of the sub-block.. Valid schemes are SAAF-CFEM-SN, SAAF-CFEM-PN, LS-CFEM-SN, LS-CFEM-PN, DFEM-SN, DFEM-PN, CFEM-Diffusion and DFEM-Diffusion. The capability of all schemes are summarized in Table 1. Currently discretization schemes are a valid zero-level input syntax item. However, this syntax should only be used for the purpose of dumping valid parameters of schemes.

4.9.2 Neutron

If particle is *neutron*, parameters in this section can be included in the discretization scheme sub-block.

4.9.2.1 *n_delay_groups*

Description: Expected number of delayed neutron groups

Data type: Integer

Default value: 0

Syntax: TransportSystems/*/n_delay_groups

Note: Neutronic materials can either have delayed neutron data with this number of groups or have none.

4.9.2.2 *fission_source_as_material*

Description: Fission source is treated as a material property or an auxiliary variable

Data type: Logical

Default value: False

Syntax: TransportSystems/*/fission_source_as_material

Note: Fission source auxiliary variable will not be added if this parameter is true and fission source will be represented as a material (recalculated on quadrature points whenever necessary). It does not affect the calculation.

4.9.2.3 *linear_fsrc_in_time*

Description: True to use linear interpolation of current and old fission source to evaluate DNPs, otherwise to use the old fission source only

Data type: Logical

Default value: True

Syntax: TransportSystems/*/fission_source_as_material

4.9.2.4 *dnp_integration_scheme*

Description: Integration scheme for DNP materials

Data type: enumeration (/exp/backwardEuler/CrankNicolson/)

Default value: exp

Syntax: TransportSystems/*/dnp_integration_scheme

4.9.2.5 *explicit_fission*

Description: True to make fission kernels operate on the old solution

Data type: Logical

Default value: false

Syntax: TransportSystems/*/explicit_fission

Note: This parameter has the same effect on fission kernels as *for_transport_update*. However it does not affect the scattering kernels as *for_transport_update* does.

4.9.3 Thermal

When particle is *thermal*, parameters in this section can be included in the discretization scheme sub-block.

4.9.3.1 *frequency_bounds*

Description: Frequency bounds of all bands (or groups)

Data type: Vector of real

Default value: <required>

Syntax: TransportSystems/*/frequency_bounds

Note: The size of this parameter must be equal to G+1. The contained values must be ascending.

4.9.3.2 *setup_temperature_equation*

Description: True to set up the temperature equation

Data type: Logical

Default value: True

Syntax: TransportSystems/*/setup_temperature_equation

4.9.3.3 *T_family*

Description: Family of FE shape functions for temperature

Data type: Enumeration (refer MOOSE syntax about Variables)

Default value: LAGRANGE

Syntax: TransportSystems/*/T_family

Note: This parameter is activated only when setup_temperature_equation is true.

4.9.3.4 *T_order*

Description: Order of FE shape functions for temperature

Data type: Enumeration (refer MOOSE syntax about Variables)

Default value: FIRST

Syntax: TransportSystems/*/T_order

Note: This parameter is activated only when setup_temperature_equation is true.

4.9.3.5 *T_option*

Description: Temperature variable option

Data type: enumeration (/T/T4/acT4/)

Default value: T

Syntax: TransportSystems/*/T_option

Note: This parameter is activated only when setup_temperature_equation is true.

4.9.3.6 *T_scaling*

Description: Scaling of the temperature variable

Data type: Real

Default value: 1

Syntax: TransportSystems/*/T_scaling

Note: This parameter is activated only when setup_temperature_equation is true.

4.9.3.7 *thermal_conduction_eos*

Description: Thermal conduction EoS (equation of state)

Data type: Vector of user object names

Default value: <empty>

Syntax: TransportSystems/*/thermal_conduction_eos

Note: This parameter contains EoS user objects for setting up the temperature equation. Empty means that no time derivative and conduction term are added in the temperature equation. This parameter is activated only when setup_temperature_equation is true.

4.9.3.8 *has_thermal_conduction*

Description: True to include the thermal conduction

Data type: Logical

Default value: True

Syntax: TransportSystems/*/has_thermal_conduction

Note: This parameter is activated only when setup_temperature_equation is true.

4.9.3.9 *heating_blocks*

Description: Subdomains having an external heat source defined

Data type: Vector of subdomain names

Default value: <empty>

Syntax: TransportSystems/*/heating_blocks

Note: This parameter specifies mesh blocks which contain external heat sources. This parameter is activated only when setup_temperature_equation is true.

4.9.3.10 *heating_sources*

Description: External heat sources

Data type: Vector of function names

Default value: <empty>

Syntax: TransportSystems/*/heating_sources

Note: This parameter specifies strength of external heat sources. Its size must be equal to the size of heating_blocks. This parameter is activated only when setup_temperature_equation is true.

4.9.3.11 *heating_boundaries*

Description: Boundaries with the fixed temperatures

Data type: Vector of boundary names

Default value: <empty>

Syntax: TransportSystems/*/heating_boundaries

Note: This parameter specifies mesh side sets which have fixed temperatures. This parameter is activated only when setup_temperature_equation is true.

4.9.3.12 *boundary_temperatures*

Description: Boundary temperatures

Data type: Vector of function names

Default value: <empty>

Syntax: TransportSystems/*/boundary_temperatures

Note: This parameter specifies temperatures on boundaries. Its size must be equal to the size of heating_boundaries. This parameter is activated only when setup_temperature_equation is true.

4.9.3.13 *temperature_update_on*

Description: Determines when the temperature is updated within the MOOSE calculation cycle

Data type: enumeration (refer to MOOSE execute_on)

Default value: linear

Syntax: TransportSystems/*/temperature_update_on

Note: This parameter is activated only when setup_temperature_equation is true.

4.9.3.14 *use_cgs*

Description: True to use cgs otherwise metric units

Data type: Logical

Default value: False

Syntax: TransportSystems/*/use_cgs

4.9.4 CFEM-Diffusion

Among the following parameters, *block*, *family*, *order*, and *verbose* are the basic parameter that almost always will have to be set, while the rest are cataloged *advanced* and typically users do not have to wprry about them.

4.9.4.1 *block*

Description: Subdomains on which this discretization scheme is applied

Data type: Vector of subdomain names

Default value: <empty>

Syntax: TransportSystems/*/block

Note: if this parameter is missing, the scheme is defined over the entire domain.

4.9.4.2 *family*

Description: Family of FE shape functions for primal variables

Data type: Enumeration (refer MOOSE syntax about Variables)

Default value: LAGRANGE

Syntax: TransportSystems/*/family

Note: Primal variables are angular fluxes, angular flux moments and scalar fluxes for SN, PN and diffusion approximation, respectively.

4.9.4.3 *order*

Description: Order of FE shape functions for primal variables

Data type: Enumeration (refer MOOSE syntax about Variables)

Default value: FIRST

Syntax: TransportSystems/*/order

4.9.4.4 *group_collapsing*

Description: Coarse group IDs of all energy groups

Data type: Vector of integers

Default value: <empty>

Syntax: TransportSystems/*/group_collapsing

Note: If this parameter is given, its size must be equal to G. Group index starts from 0. Numbers in this vector must currently be non-decreasing. If this parameter is not given, number of coarse groups is equal to G and no group collapsing will be done.

4.9.4.5 *group_weights*

Description: To apply different weights for the equations associated with each group

Data type: Vector of real

Default value: <empty>

Syntax: TransportSystems/*/group_weights

Note: If this parameter is given, its size must be equal to G. If the parameter is not given, all groups have the default weight one.

4.9.4.6 *collapse_scattering*

Description: True to create in-group scattering source and use it in the scattering kernels

Data type: Logical

Default value: False

Syntax: TransportSystems/*/collapse_scattering

4.9.4.7 *balance_table*

Description: True to generate balance table

Data type: Logical

Default value: False

Syntax: TransportSystems/*/balance_table

Note: Balance table is for all blocks specified in block.

4.9.4.8 *balance_table_on*

Description: When the balance table is evaluated and printed

Data type: enumeration (/initial/timestep_begin/timestep_end/nonlinear/linear/custom/, refer MOOSE syntax about execute_on options)

Default value: timestep_end

Syntax: TransportSystems/*/balance_table_on

4.9.4.9 *fixed_jacobian*

Description: To indicate the Jacobian is constant throughout the simulation

Data type: Logical

Default value: False

Syntax: TransportSystems/*/fixed_jacobian

Note: If the Jacobian is constant throughout the simulation, setting this parameter to true can reduce the number of Jacobian evaluations.

4.9.4.10 *verbose*

Description: To control screen output related to the setup of the transport system

Data type: Enumeration (/0/1/2/3/)

Default value: 1

Syntax: TransportSystems/*/verbose

Note: The bigger the verbose number is, the more detailed outputs on screen are. Note that when the number is bigger than 1, angular quadrature and spherical harmonics data will be dumped into "aq.txt" and "shm.txt" respectively.

4.9.4.11 *prefix*

Description: Prefix for MOOSE objects used by the transport system

Data type: String

Default value: Name of the sub-block

Syntax: TransportSystems/*/prefix

Note: Here MOOSE objects include Kernels, AuxKernels, DiracKernels, DGKernels, BCs, Materials, MultiApps and Transfers. This parameter is exposed to users only for the purpose of avoiding name conflict in multiphysics simulations where users cannot change how these MOOSE objects in other physics are named.

4.9.4.12 *prefix_variables*

Description: Whether or not to add prefix for automatically added variables

Data type: Logical

Default value: False

Syntax: TransportSystems/*/prefix_variables

Note: This parameter is exposed to users only for the purpose of avoiding name conflict in multiphysics simulations where users cannot change how variables in other physics are named. If this parameter is true, prefix will be used as the prefix.

4.9.4.13 *material_prop_namespace*

Description: Prefix for transport material properties

Data type: string

Default value: empty string

Syntax: TransportSystems/*/material_prop_namespace

Note: This parameter is exposed to users only for the purpose of avoiding name conflict in multiphysics simulations where users cannot change how material properties in other physics are named.

4.9.4.14 *save_residual*

Description: True to save residual into auxiliary variables

Data type: Logical

Default value: False

Syntax: TransportSystems/*/save_residual

Note: This parameter is deprecated.

4.9.4.15 *assemble_scattering_jacobian*

Description: True for assembling Jacobian contributions from scattering

Data type: Logical

Default value: False

Syntax: TransportSystems/*/assemble_scattering_jacobian

Note: Whether or not the scattering term is considered during assembling the Jacobian. Note, that it also depends on what preconditioning matrix is used, since, by default, Rattlesnake only assembles the block-diagonal Jacobian, where each block corresponds to a variable. In this case cross-group scattering contributions to the Jacobian are not considered even if this parameter is true.

4.9.4.16 *vacuum_extrapolation_factor*

Description: Vacuum extrapolation factor for all vacuum side sets and all groups

Data type: Real

Default value: 2/3

Syntax: TransportSystems/*/vacuum_extrapolation_factor

4.9.4.17 *diffusion_coefficient_type*

Description: Diffusion coefficient type

Data type: Enumeration (/scalar/vector/tensor/)

Default value: scalar

Syntax: TransportSystems/*/diffusion_coefficient_type

Note: Users need to provide diffusion coefficients of this type in the transport materials, otherwise implicit conversion will be invoked:

$$D = \begin{cases} D, & scalr \\ \frac{1}{3}\sum_{i=1}^{3} D_i, & vector \\ \frac{1}{3}\sum_{i=1}^{3}\sum_{j=1}^{3} D_{i,j}, & tensor \end{cases} \tag{26}$$

$$D_i = \begin{cases} D, & scalar \\ D_i, & vector \\ \sum_{j=1}^{3} D_{i,j}, & tensor \end{cases}, i = 1,2,3 \tag{27}$$

$$D_{i,j} = \begin{cases} D\delta_{i,j}, & scalar \\ D_i\delta_{i,j}, & vector \\ D_{i,j}, & tensor \end{cases}, i = 1,2,3; j = 1,2,3. \tag{28}$$

4.9.4.18 *adjoint_fluxes*

Description: List of adjoint fluxes ordered by group index

Data type: Vector of strings

Default value: <empty>

Syntax: TransportSystems/*/adjoint_fluxes

Note: This parameter is only used for IQS executioner and is currently under development.

4.9.4.19 *transport_wrapper*

Description: The user object for adding derived material properties with a high-order system

Data type: string

Default value: <empty>

Syntax: TransportSystems/*/transport_wrapper

Note: This parameter is used for calculations with the nonlinear diffusion acceleration. It can be either an input file name or a valid user object name for the high order transport system. If the input file name is provided,

Rattlesnake will construct the user object on the fly. If users want to have more control on how the user object behaves, they can add the user object manually and pass the name to the diffusion system here. Refer to SAAFWrapper and LSWrapper for more details. This parameter will cause material properties of drift vector and vacuum boundary coefficient to be declared and corresponding terms be added in the equation. Users are referred to [16] for more explanations.

4.9.5 DFEM-Diffusion

This discretization scheme shares most of parameters in CFEM-Diffusion except the followings:

- The default value for *family* is changed to *L2_LAGRANGE*.
- *transport_wrapper* is not valid.
- there are more parameters: *DGType*, *penalty*, *transport_multiapp_file* and *NDA_type*.

4.9.5.1 *block*

Refer to *block* in CFEM-Diffusion.

4.9.5.2 *family*

Refer to *family* in CFEM-Diffusion.

Note: The default value for family is changed to *L2_LAGRANGE*.

4.9.5.3 *order*

Refer to *order* in CFEM-Diffusion.

4.9.5.4 *group_collapsing*

Refer to *group_collapsing* in CFEM-Diffusion.

4.9.5.5 *group_weights*

Refer to *group_weights* in CFEM-Diffusion.

4.9.5.6 *collapse_scattering*

Refer to *collapse_scattering* in CFEM-Diffusion.

4.9.5.7 *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.9.5.8 *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.9.5.9 *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.9.5.10 *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.9.5.11 *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.9.5.12 *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.9.5.13 *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.9.5.14 *save_residual*

Refer to *save_residual* in CFEM-Diffusion.

4.9.5.15 *assemble_scattering_jacobian*

Refer to *assemble_scattering_jacobian* in CFEM-Diffusion.

4.9.5.16 *vacuum_extrapolation_factor*

Refer to *vacuum_extrapolation_factor* in CFEM-Diffusion.

4.9.5.17 *diffusion_coefficient_type*

Refer to *diffusion_coefficient_type* in CFEM-Diffusion.

4.9.5.18 *adjoint_fluxes*

Refer to *adjoint_fluxes* in CFEM-Diffusion.

4.9.5.19 *DGType*

Description: DG diffusion type (SIP/NIP/IIP - symmetric interior penalty/non-symmetric IP/incomplete IP)

Data type: Enumeration (/SIP/NIP/IIP/)

Default value: SIP

Syntax: TransportSystems/*/DGType

4.9.5.20 *penalty*

Description: Penalty function type

Data type: Enumeration (/std/)

Default value: std

Syntax: TransportSystems/*/penalty

4.9.5.21 *transport_multiapp_file*

Description: The input file name of the SN sub problem

Data type: string

Default value: <empty>

Syntax: TransportSystems/*/transport_multiapp_file

Note: A valid file name will invoke the NDA calculation with a high-order transport system set up by this input file.

4.9.5.22 *NDA_type*

Description: NDA type

Data type: Enumeration (/syw/traditional_cmfd/pcmfd/)

Default value: traditional_cmfd

Syntax: TransportSystems/*/NDA_type

Note: Refer to the Rattlesnake theory manual for more information.

4.9.6 SAAF-CFEM-SN

This discretization scheme shares the following parameters in CFEM-Diffusion: *block*, *family*, *order*, *group_collapsing*, *group_weights*, *collapse_scattering*, *balance_table*, *balance_table_on*, *fixed_jacobian*, *verbose*, *prefix*, *prefix_variables*, *material_prop_namespace*, assemble_scattering_jacobian and larsen_trahan.

4.9.6.1 *block*

Refer to *block* in CFEM-Diffusion.

4.9.6.2 *family*

Refer to *family* in CFEM-Diffusion.

4.9.6.3 *order*

Refer to *order* in CFEM-Diffusion.

4.9.6.4 *group_collapsing*

Refer to *group_collapsing* in CFEM-Diffusion.

4.9.6.5 *group_weights*

Refer to *group_weights* in CFEM-Diffusion.

4.9.6.6 *collapse_scattering*

Refer to *collapse_scattering* in CFEM-Diffusion.

4.9.6.7 *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.9.6.8 *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.9.6.9 *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.9.6.10 *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.9.6.11 *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.9.6.12 *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.9.6.13 *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.9.6.14 *vacuum_bc_type*

Description: Vacuum or surface source boundary condition type

Data type: Enumeration (/saaf/even/odd/pe/)

Default value: saaf

Syntax: TransportSystems/*/vacuum_bc_type

Note: Users are referred to [17] for more explanations.

4.9.6.15 *reflecting_bc_type*

Description: Reflecting boundary condition type

Data type: Enumeration (/saaf/even/odd/symmetric/non-sym/)

Default value: saaf

Syntax: TransportSystems/*/reflecting_bc_type

Note: users are referred to [17] for more explanations. It is noted that this parameter is invalid for diffusion approximation.

4.9.6.16 *AQtype*

Description: Angular quadrature type

Data type: Enumeration (/Level-Symmetric/Gauss-Chebyshev/Bickley3-Optimized/)

Default value: Level-Symmetric

Syntax: TransportSystems/*/AQtype

Note:

- Level-Symmetric only support two-dimensional (2D) and three-dimensional (3D) calculations. The number of directions, depending on *AQorder*, is *AQorder*$\times$(*AQorder*+2)/2 in 2D and *AQorder*$\times$(*AQorder*+2) in 3D.
- Gauss-Chebyshev supports one-dimensional (1D), 2D and 3D calculations. The number of directions is equal to *NPolar*$\times$2 in 1D, *NPolar*$\times$*NAzmthl*$\times$4 in 2D and *NPolar*$\times$*NAzmthl*$\times$8 in 3D. This quadrature is simply the Gaussian quadrature in 1D.
- Bickley3-Optimized only supports 2D calculations. The number of directions is equal to *NPolar*$\times$*NAzmthl*$\times$4.

One constraint currently in our Sn calculations is that the reflective directions of all directions in the angular quadrature of all reflective boundary sides characterized with their unit perpendicular norm ($\vec{n}$) must be in the angular quadrature. All of the three quadratures satisfy this condition when $\vec{n} = \vec{e}_x$, $\vec{n} = \vec{e}_y$ or $\vec{n} = \vec{e}_z$, which covers most of practical calculations. Gauss-Chebyshev and Bickley3-Optimized can also find usages when the z-component of $\vec{n}$ is zero. In such cases, users can adjust *NAzmthl* to make the constraint satisfied. The azimuthal angles are distributed evenly with the two quadratures. The angles will be $(2i+1) \times \pi/NAzmthl/4$, $i = 0, \cdots, 4 \times NAzmthl$-1, where 4 is the number of octant. For example, if *NAzmthl* is equal to 3, there will be $3 \times 4 = 12$ azimuthal angles, started from $\pi/12$ with increment $\pi/6$. Three will be angles with 15, 45 and 75 degrees in the first octant. This will make all reflecting angles on a surface in parallel with z-axis but 30, 45 or 75 degree tilted with respect to x-axis, also in the quadrature. All quadratures with *NAzmthl* being equal to $3n$, where n is any natural number, satisfy the constraint for these surfaces as well.

4.9.6.17 *AQorder*

Description: The order of level-symmetric angular quadrature

Data type: Enumeration (/2/4/6/8/10/12/14/16/18/20/22/24/26/28/30/)

Default value: 8

Syntax: TransportSystems/*/AQorder

Note: This parameter is only valid for level symmetric quadrature. It will be ignored for other quadrature types even if provided.

4.9.6.18 *NPolar*

Description: The number of polar angles of a production angular quadrature

Data type: Integer

Default value: 2

Syntax: TransportSystems/*/NPolar

Note: This parameter must be greater than 0. It is used by production quadrature types including Gauss-Chebyshev and Bickley3-Optimized only. It will be ignored for other quadrature types even if provided.

4.9.6.19 *NAzmthl*

Description: The number of azimuthal angles of a production angular quadrature

Data type: Integer

Default value: 3

Syntax: TransportSystems/*/NAzmthl

Note: This parameter must be greater than 0. It is used by production quadrature types including Gauss-Chebyshev and Bickley3-Optimized only. It will be ignored for other quadrature types even if provided.

4.9.6.20 *NA*

Description: The maximum order of scattering anisotropy

Data type: Integer

Default value: 0

Syntax: TransportSystems/*/NA

Note: This parameter must be greater than or equal to zero with zero meaning isotropic scattering. This parameter also controls how many angular flux moments are to be evaluated: NA+1 in one-dimension, (NA+1)×(NA+2)/2 in two-dimension and $(NA+1)^2$ in three-dimension.

4.9.6.21 *initialize_angular_flux*

Description: True to initialize angular flux to one

Data type: Logical

Default value: False

Syntax: TransportSystems/*/initialize_angular_flux

Note: By default all angular flux will be initialized to zero. This parameter is used for ensuring correct evaluation of the closure terms in the diffusion equation when NDA is used.

4.9.6.22 *hide_angular_flux*

Description: True to not output angular flux

Data type: Logical

Default value: False

Syntax: TransportSystems/*/hide_angular_flux

4.9.6.23 *hide_higher_flux_moment*

Description: True to hide angular flux moments with order higher than the inputted value

Data type: Integer

Default value: Maximum value of an unsigned integer

Syntax: TransportSystems/*/hide_higher_flux_moment

Note: This parameter must be greater than or equal to 0, which means scalar fluxes are not allowed to be hidden.

4.9.6.24 *flux_moment_as_material*

Description: True to use material properties for flux moments

Data type: Logical

Default value: False

Syntax: TransportSystems/*/flux_moment_as_material

Note: This parameter does not affect the solver or the problem to be solved.

4.9.6.25 *for_transport_update*

Description: True to set up the transport system for transport update

Data type: Logical

Default value: False

Syntax: TransportSystems/*/for_transport_update

Note: users are referred to [16] for more explanations. Some executioner require this parameter to be true.

4.9.6.26 *explicit_on_boundary*

Description: True to use old angular fluxes for evaluating implicit BCs (reflecting or white and etc.) during transport update

Data type: Logical

Default value: False

Syntax: TransportSystems/*/explicit_on_boundary

Note: users are referred to [16, 18] for more explanations.

4.9.6.27 *assemble_scattering_jacobian*

Refer to *assemble_scattering_jacobian* in CFEM-Diffusion.

4.9.6.28 *tau*

Description: SUPG stabilization parameter for void or near-void

Data type: Real

Default value: 0.5

Syntax: TransportSystems/*/tau

Note: When total mean free path in an element evaluated with the local element size h is greater than this number, the inverse of the total cross section is used; otherwise h over this number is used for stabilization. Users should consider using smaller number for sufficient stabilization. Users are referred to [18] for more explanations.

4.9.6.29 *show_drift*

Description: True to show drift vectors and vacuum coefficients on quadrature points

Data type: Logical

Default value: false

Syntax: TransportSystems/*/show_drift

Note: This quantities are used as diffusion closure in NDA calculations.

4.9.6.30 *nda_damping*

Description: Damping factor (under-relaxation factor) for evaluating drift vectors and vacuum boundary coefficients for diffusion closure.

Data type: Real

Default value: 0

Syntax: TransportSystems/*/nda_damping

4.9.6.31 *larsen_trahan*

Description: Triggers computation of Larsen-Trahan mode for computation of tensor diffusion coefficients. For this purpose a problem is solved without scattering and fission, and with a unit source. Output cross sections files are created/overridden. The file names are formed as

<input file base> +_larsen_trahan_material_out_ + <cnm | mnm> +.xml

Existing files are read and the correct entry is modified. If the file does not exist a new MixedMultigroupLibrary is created and written to the file.

Data type: bool

Default value: 0

Syntax: TransportSystems/*/larsen_trahan

Note: Can only be used with particle type common. Must use steady executioner or executioner supporting _for_transport_update. Only supports ConstantNeutronicsMaterial or MixedNeutronicsMaterial.

4.9.7 SAAF-CFEM-PN

This discretization scheme shares the following parameters in CFEM-Diffusion: *block*, *family*, *order*, *group_collapsing*, *group_weights*, *collapse_scattering*, *balance_table*, *balance_table_on*, *fixed_jacobian*, *verbose*, *prefix*, *prefix_variables* and *material_prop_namespace*. It also shares the following parameters in SAAF-CFEM-SN: *vacuum_bc_type*, *reflecting_bc_type* and *hide_higher_flux_moment*.

4.9.7.1 *block*

Refer to *block* in CFEM-Diffusion.

4.9.7.2 *family*

Refer to *family* in CFEM-Diffusion.

4.9.7.3 *order*

Refer to *order* in CFEM-Diffusion.

4.9.7.4 *group_collapsing*

Refer to *group_collapsing* in CFEM-Diffusion.

4.9.7.5 *group_weights*

Refer to *group_weights* in CFEM-Diffusion.

4.9.7.6 *collapse_scattering*

Refer to *collapse_scattering* in CFEM-Diffusion. It is noted that the default value of collapse_scattering is changed to true.

4.9.7.7 *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.9.7.8 *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.9.7.9 *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.9.7.10 *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.9.7.11 *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.9.7.12 *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.9.7.13 *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.9.7.14 *vacuum_bc_type*

Refer to *vacuum_bc_type* in SAAF-CFEM-SN.

4.9.7.15 *reflecting_bc_type*

Refer to *reflecting_bc_type* in SAAF-CFEM-SN.

4.9.7.16 *parity_option*

Description: To select what the primal variable should be for PN

Data type: Enumeration (/all/even/)

Default value: all

Syntax: TransportSystems/*/parity_option

4.9.7.17 *PN*

Description: PN order

Data type: Integer

Default value: 0

Syntax: TransportSystems/*/PN

Note: When parity_option is all, this parameter can be any number greater than or equal to 0. When parity_option is even, this parameter must be an odd non-negative number.

4.9.7.18 *NA*

Description: the maximum order of scattering anisotropy

Data type: Integer

Default value: 0

Syntax: TransportSystems/*/NA

Note: This parameter must be greater than or equal to zero and less than or equal to PN.

4.9.7.19 *hide_higher_flux_moment*

Refer to *hide_higher_flux_moment* in SAAF-CFEM-SN.

4.9.7.20 *force_secondary_parity*

Description: True to force the evaluation of the secondary parity materials when parity_option is not all

Data type: Logical

Default value: False

Syntax: TransportSystems/*/force_secondary_parity

4.9.7.21 *initialize_flux_moment*

Description: True to initialize flux moments with one

Data type: Logical

Default value: False

Syntax: TransportSystems/*/initialize_flux_moment

4.9.8 LS-CFEM-SN

This discretization scheme shares the following parameters in CFEM-Diffusion: *block*, *family*, *order*, *group_collapsing*, *group_weights*, *balance_table*, *balance_table_on*, *fixed_jacobian*, *verbose*, *prefix*, *prefix_variables*, *material_prop_namespace* and assemble_scattering_jacobiansn. It also shares the following parameters in SAAF-CFEM-SN: *AQtype*, *AQorder*, *NPolar*, *NAzmthl*, *NA*, *initialize_angular_flux*, *hide_angular_flux*, *hide_higher_flux_moment*, *for_transport_update*, *explicit_on_boundary* and *show_drift*.

4.9.8.1 ***block***

Refer to *block* in CFEM-Diffusion.

4.9.8.2 ***family***

Refer to *family* in CFEM-Diffusion.

4.9.8.3 ***order***

Refer to *order* in CFEM-Diffusion.

4.9.8.4 *group_collapsing*

Refer to *group_collapsing* in CFEM-Diffusion.

4.9.8.5 *group_weights*

Refer to *group_weights* in CFEM-Diffusion.

4.9.8.6 *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.9.8.7 *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.9.8.8 *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.9.8.9 *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.9.8.10 *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.9.8.11 *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.9.8.12 *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.9.8.13 *AQtype*

Refer to AQtype in SAAF-CFEM-SN.

4.9.8.14 *AQorder*

Refer to AQorder in SAAF-CFEM-SN.

4.9.8.15 *NPolar*

Refer to NPolar in SAAF-CFEM-SN.

4.9.8.16 *NAzmthl*

Refer to NAzmthl in SAAF-CFEM-SN.

4.9.8.17 *NA*

Refer to NA in SAAF-CFEM-SN.

4.9.8.18 *initialize_angular_flux*

Refer to *initialize_angular_flux* in SAAF-CFEM-SN.

4.9.8.19 *hide_angular_flux*

Refer to hide_angular_flux in SAAF-CFEM-SN.

4.9.8.20 *hide_higher_flux_moment*

Refer to hide_higher_flux_moment in SAAF-CFEM-SN.

4.9.8.21 *flux_moment_as_material*

Refer to *flux_moment_as_material* in SAAF-CFEM-SN.

4.9.8.22 *for_transport_update*

Refer to *for_transport_update* in SAAF-CFEM-SN.

4.9.8.23 *explicit_on_boundary*

Refer to *explicit_on_boundary* in SAAF-CFEM-SN.

4.9.8.24 *assemble_scattering_jacobian*

Refer to *assemble_scattering_jacobian* in CFEM-Diffusion.

4.9.8.25 *show_drift*

Refer to *show_drift* in SAAF-CFEM-SN.

4.9.8.26 *strong_boundary_condition*

Description: True to impose the boundary condition strongly

Data type: Logical

Default value: False

Syntax: TransportSystems/*/strong_boundary_condition

Note: Users are referred to [19] for more details.

4.9.8.27 *weak_bc_type*

Description: To select different types of weakly imposed boundary condition

Data type: Enumeration (/1/2/3/4/5/6/7/)

Syntax: TransportSystems/*/weak_bc_type

Note: Users are referred to [19] for more details.

4.9.8.28 *weak_bc_constant*

Description: The constant factor for weak BCs

Data type: Real

Default value: 1

Syntax: TransportSystems/*/weak_bc_constant

Note: Users are referred to [19] for more details.

4.9.9 LS-CFEM-PN

This discretization scheme shares the following parameters in CFEM-Diffusion: *block*, *family*, *order*, *group_collapsing*, *group_weights*, *collapse_scattering*, *balance_table*, *balance_table_on*, *fixed_jacobian*, *verbose*, *prefix*, *prefix_variables* and *material_prop_namespace*. It also shares the following parameters in SAAF-CFEM-SN: *vacuum_bc_type*, *reflecting_bc_type* and *hide_higher_flux_moment*. It also shares the following parameters in SAAF-CFEM-PN: *PN*, *NA*, *force_secondary_parity*, *initialize_flux_moment*.

This scheme does not work with *transient* *equation_type* and with Neutron currently.

4.9.9.1 *block*

Refer to *block* in CFEM-Diffusion.

4.9.9.2 *family*

Refer to *family* in CFEM-Diffusion.

4.9.9.3 *order*

Refer to *order* in CFEM-Diffusion.

4.9.9.4 *group_collapsing*

Refer to *group_collapsing* in CFEM-Diffusion.

4.9.9.5 *group_weights*

Refer to *group_weights* in CFEM-Diffusion.

4.9.9.6 *collapse_scattering*

Refer to *collapse_scattering* in CFEM-Diffusion. It is noted that the default value of collapse_scattering is changed to true.

4.9.9.7 *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.9.9.8 *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.9.9.9 *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.9.9.10 *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.9.9.11 *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.9.9.12 *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.9.9.13 *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.9.9.14 *vacuum_bc_type*

Refer to *vacuum_bc_type* in SAAF-CFEM-SN.

4.9.9.15 *reflecting_bc_type*

Refer to *reflecting_bc_type* in SAAF-CFEM-SN.

4.9.9.16 *parity_option*

Refer to *parity_option* in SAAF-CFEM-PN.

4.9.9.17 *PN*

Refer to *PN* in SAAF-CFEM-PN.

4.9.9.18 *NA*

Refer to *NA* in SAAF-CFEM-PN.

4.9.9.19 *hide_higher_flux_moment*

Refer to *hide_higher_flux_moment* in SAAF-CFEM-SN.

4.9.9.20 *force_secondary_parity*

Refer to *force_secondary_parity* in SAAF-CFEM-PN.

4.9.9.21 *initialize_flux_moment*

Refer to *initialize_flux_moment* in SAAF-CFEM-PN.

4.9.9.22 *weak_bc_type*

Refer to *weak_bc_type* in LS-CFEM-SN.

The default value is changed to 7.

4.9.9.23 *weak_bc_constant*

Refer to *weak_bc_constant* in LS-CFEM-SN.

4.9.9.24 *NS*

Description: The maximum spherical harmonics order of external source moments to keep

Data type: Integer

Default value: 0

Syntax: TransportSystems/*/NS

Note: Zero means that only the isotropic part of the external source is kept.

4.9.10 DFEM-SN

This discretization scheme shares the following parameters in CFEM-Diffusion: *block*, *family*, *order*, *group_collapsing*, *group_weights*, *balance_table*, *balance_table_on*, *fixed_jacobian*, *verbose*, *prefix*, *prefix_variables* and *material_prop_namespace*. It also shares the following parameters in SAAF-CFEM-SN: *AQtype*, *AQorder*, *NPolar*, *NAzmthl*, *NA*, *initialize_angular_flux*, *hide_angular_flux*, *hide_higher_flux_moment*, *for_transport_update*, *explicit_on_boundary* and *larsen_trahan* .

4.9.10.1 *block*

Refer to *block* in CFEM-Diffusion.

4.9.10.2 *family*

Refer to *family* in CFEM-Diffusion. It is noted that the default value of family is changed to L2_LAGRANGE.

4.9.10.3 *order*

Refer to *order* in CFEM-Diffusion.

4.9.10.4 *group_collapsing*

Refer to *group_collapsing* in CFEM-Diffusion.

4.9.10.5 *group_weights*

Refer to *group_weights* in CFEM-Diffusion.

4.9.10.6 *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.9.10.7 *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.9.10.8 *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.9.10.9 *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.9.10.10 *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.9.10.11 *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.9.10.12 *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.9.10.13 *AQtype*

Refer to AQtype in SAAF-CFEM-SN.

4.9.10.14 *AQorder*

Refer to AQorder in SAAF-CFEM-SN.

4.9.10.15 *NPolar*

Refer to NPolar in SAAF-CFEM-SN.

4.9.10.16 *NAzmthl*

Refer to NAzmthl in SAAF-CFEM-SN.

4.9.10.17 *NA*

Refer to NA in SAAF-CFEM-SN.

4.9.10.18 *initialize_angular_flux*

Refer to *initialize_angular_flux* in SAAF-CFEM-SN.

4.9.10.19 *hide_angular_flux*

Refer to hide_angular_flux in SAAF-CFEM-SN.

4.9.10.20 *hide_higher_flux_moment*

Refer to hide_higher_flux_moment in SAAF-CFEM-SN.

4.9.10.21 *flux_moment_as_material*

Refer to *flux_moment_as_material* in SAAF-CFEM-SN.

4.9.10.22 *for_transport_update*

Refer to *for_transport_update* in SAAF-CFEM-SN.

4.9.10.23 *explicit_on_boundary*

Refer to *explicit_on_boundary* in SAAF-CFEM-SN.

4.9.10.24 *larsen_trahan*

Refer to *larsen_trahan* in SAAF-CFEM-SN.

4.9.11 DFEM-PN

This discretization scheme shares the following parameters in CFEM-Diffusion: *block*, *family*, *order*, *group_collapsing*, *group_weights*, *balance_table*, *balance_table_on*, *fixed_jacobian*, *verbose*, *prefix*, *prefix_variables* and *material_prop_namespace*.
It also shares the following parameters in SAAF-CFEM-SN: *vacuum_bc_type*, *reflecting_bc_type* and *hide_higher_flux_moment*.
It also shares the following parameters in SAAF-CFEM-PN: *PN*, *NA*, *force_secondary_parity*, *initialize_flux_moment*.

4.9.11.1 ***block***

Refer to *block* in CFEM-Diffusion.

4.9.11.2 ***family***

Refer to *family* in CFEM-Diffusion. It is noted that the default value of family is changed to L2_LAGRANGE.

4.9.11.3 ***order***

Refer to *order* in CFEM-Diffusion.

4.9.11.4 *group_collapsing*

Refer to *group_collapsing* in CFEM-Diffusion.

4.9.11.5 *group_weights*

Refer to *group_weights* in CFEM-Diffusion.

4.9.11.6 *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.9.11.7 *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.9.11.8 *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.9.11.9 *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.9.11.10 *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.9.11.11 *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.9.11.12 *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.9.11.13 *vacuum_bc_type*

Refer to *vacuum_bc_type* in SAAF-CFEM-SN.

4.9.11.14 *reflecting_bc_type*

Refer to *reflecting_bc_type* in SAAF-CFEM-SN.

4.9.11.15 *parity_option*

Refer to *parity_option* in SAAF-CFEM-PN.

4.9.11.16 *PN*

Refer to *PN* in SAAF-CFEM-PN.

4.9.11.17 *NA*

Refer to *NA* in SAAF-CFEM-PN.

4.9.11.18 *hide_higher_flux_moment*

Refer to *hide_higher_flux_moment* in SAAF-CFEM-SN.

4.9.11.19 *force_secondary_parity*

Refer to *force_secondary_parity* in SAAF-CFEM-PN.

4.9.11.20 *initialize_flux_moment*

Refer to *initialize_flux_moment* in SAAF-CFEM-PN.

4.9.11.21 *NS*

Refer to *NS* in LS-CFEM-PN.

4.9.11.22 *filter_type*

Description: The type of filter to use

Data type: Enumeration (/none/Lanczos/SSpline/Exp)

Default value: none

Syntax: TransportSystems/*/filter_type

4.9.11.23 *filter_strength_func*

Description: The function to use as the filter strength

Data type: String

Default value: <empty>

Syntax: TransportSystems/*/filter_strength_func

4.9.11.24 *exp_filter_order*

Description: The order of the exponential filter to use

Data type: Integer

Default value: 1

Syntax: TransportSystems/*/exp_filter_order

Note: This parameter is useful only if *filter_type* is Exp.

4.9.11.25 *exp_filter_const*

Description: The logarithm of the machine accuracy

Data type: Integer

Default value: -14

Syntax: TransportSystems/*/exp_filter_const

Note: This parameter is useful only if *filter_type* is Exp.

4.9.11.26 *removal_lumping*

Description: True to lump the removal terms

Data type: Logical

Default value: false

Syntax: TransportSystems/*/removal_lumping

4.10 Summary of MOOSE objects added by *TransportSystems*

Rattlesnake is an ***open*** system in a sense that users can inject more MOOSE objects, including variables, materials, postprocessors and etc., into the system and can retrieve all MOOSE objects contained by Rattlesnake for post-processing purpose. To this regard, it is needed to summarize the MOOSE objects added by Rattlesnake. All these objects can be seen on the screen print-out during the setup stage when *verbose* is greater than 1. Name of material properties can be obtained with *Debug/show_material_props* to true. It is noted that not all objects added by Rattlesnake are listed in this manual. Only those might useful to users are given.

4.10.1 Primal variables

The primal variables added with transport systems are listed in Table 16 and Table 17. It is noted that the energy group index g<g> means the variables are added for all groups with $g = 0, \cdots, G-1$, where G is the number of coarse energy groups specified by *G* and *group_collapsing*. The double spherical harmonics index L<l>_M<m> means the variable are added for all spherical harmonics with $m = low(l), \cdots, up(l); l = 0, \cdots, PN$, where $low(l)$ is 0, 0 and $-l$ and $up(l)$ is 0, l and l for one-dimension, two-dimension and three-dimension respectively and PN is the spherical harmonics order specified by *PN*. When parity_option is even, only variables with even l are added. The streaming direction index d<d> means the variables are added for all streaming directions specified by the angular quadrature. Temperature variable is added for Thermal when *setup_temperature_equation* is true.

Table 16 Primal variables of neutron transport systems.

Meaning	Name	Scheme
Scalar flux	sflux_g<g>	CFEM-Diffusion DFEM-Diffusion
Angular flux moment	flux_moment_g<g>_L<l>_M<m>	SAAF-CFEM-PN LS-CFEM-PN
Angular flux	aflux_g<g>_d<d>	SAAF-CFEM-SN LS-CFEM-SN DFEM-SN

4.10.2 Auxiliary variables

The auxiliary variables added with transport systems are listed in Table 18 and Table 19. The delayed neutron group index i<i> means the variables are added for all delayed neutron groups with $i = 0, \cdots, I-1$, where I is the number of delayed neutron groups specified by *n_delay_groups*. Angular flux moments are added

Table 17 Primal variables of thermal radiation transport systems.

Meaning	Name	Scheme
Radiation density	E_g<g>	CFEM-Diffusion
Angular flux moment	E_g<g>_L<l>_M<m>	DFEM-PN
Specific intensity	specific_intensity_g<g>_d<d>	SAAF-CFEM-SN
Temperature	*T_option*	

for SN schemes with the maximum order being *NA*. Temperature variable is added for Thermal only when *setup_temperature_equation* is true. Prompt fission source and delayed neutron precursors are added for Neutron only when *fission_source_as_material* is false. Adjoint fission source is added for Neutron only when SAAF-CFEM-SN scheme is used and both *for_adjoint* and *for_math_adjoint* are true.

Table 18 Auxiliary variables of neutron transport systems.

Meaning	Name	Scheme
Angular flux moment	flux_moment_g<g>_L<l>_M<m>	SAAF-CFEM-SN LS-CFEM-SN DFEM-SN
Prompt fission source	fission_source	
Delayed neutron precursor	dnp_i<i>	
Adjoint fission source	fission_source_adj	

Table 19 Auxiliary variables of thermal radiation transport systems.

Meaning	Name	Scheme
Angular flux moment	E_g<g>_L<l>_M<m>	SAAF-CFEM-SN
Temperature	temperature	
Planck emission	total_emission	
Radiation heating	radiation_heat_source	
Radiative equilibrium	radiative_equilibrium	

4.10.3 Material properties

In-group scattering sources for all groups and all spherical harmonics are evaluated when *collapse_scattering* is true. It is noted that the schemes supporting this parameter are CFEM-Diffusion, DFEM-Diffusion, SAAF-CFEM-SN, SAAF-CFEM-PN and LS-CFEM-PN. The single spherical harmonics index p is related with the double index (l, m) as

$$p = \begin{cases} l, & 1D \\ \frac{l(l+1)}{2} + m, & 2D \\ l^2 + l + m, & 3D \end{cases} \tag{29}$$

The total number of the scattering sources P is

$$P = G \times \begin{cases} L, & 1D \\ \frac{(L+1)(L+2)}{2}, & 2D \\ (L+1)^2, & 3D \end{cases}, \tag{30}$$

where L is *NA*. G is the number of coarse groups specified by G and *group_collapsing*. Typically scattering sources are evaluated with the current solution. But they are evaluated with the old solution when *for_transport_update* is true for SAAF-CFEM-SN scheme.

Material properties of prompt fission source, delayed neutron precursor (DNP) concentrations are evaluated when *fission_source_as_material* is true for Neutron transport. Material properties of angular flux moments present only for DFEM-SN scheme and *flux_moment_as_material* is true. Secondary parity moments of PN schemes, SAAF-CFEM-PN and LS-CFEM-PN, are evaluated as material properties when *parity_option* is set to even and when the scattering cross section of the corresponding order is nonzero or *force_secondary_parity* is set to true. Boundary out-going partial current is only evaluated for SN schemes, SAAF-CFEM-SN, LS-CFEM-SN and DFEM-SN, on the white boundaries specified by *WhiteBoundary*.

Table 20 Material properties of transport systems.

Meaning	Name	Scheme	Stateful	Volumetric
In-group scattering source	scattering_source_p<p>_g<g>	CFEM-Diffusion DFEM-Diffusion SAAF-CFEM-SN SAAF-CFEM-PN LS-CFEM-PN	No	Yes
Prompt fission source	fission_source	Neutron	No	Yes
Delayed neutron precursor	dnp_i<i>	Neutron	Yes	Yes
Angular flux moments	flux_moment_g<g>_L<l>_M<m>	DFEM-SN	No	Yes
Secondary parity moments	flux_moment_g<g>_L<l>_M<m>	SAAF-CFEM-PN LS-CFEM-PN	No	Yes
SAAF SN stabilization parameter τ	tau_g<g>	SAAF-CFEM-SN	No	Yes
$\lvert\vec{n}\rvert_1$ of all sides	l1norm_normals	DFEM-PN	No	No
$\bar{\vec{n}}_{e,qp} = \frac{\int_e \vec{n} ds}{\int_e ds}$ of all sides	averaged_normal	DFEM-Diffusion	No	No
Boundary out-going partial current	out_current_g<g>	SAAF-CFEM-SN LS-CFEM-SN DFEM-SN	No	No

5 Other Non-MOOSE Syntax

5.1 *PKE*

The point-kinetics equation (PKE) is

$$\frac{dn(t)}{dt} = \frac{\rho - \beta}{\Lambda} n + \sum_{i=1}^{I} \lambda_i c_i, \tag{31}$$

$$\frac{dc_i(t)}{dt} = \frac{\beta_i}{\Lambda} n - \lambda_i c_i; i = 1, \cdots, I, \tag{32}$$

where $n(t)$ is the neutron population and $c_i(t)$ are the delayed neutron precursor (DNP) concentrations; I is the total number of DNP groups. ρ, Λ, β_i and λ_i are the reactivity, neutron generation time, DNP fractions and DNP decay constants. They are all potentially time dependent. The total DNP fraction $\beta = \sum_{i=1}^{I} \beta_i$. The initial condition for PKE is

$$n(t = t_1) = n_0 \tag{33}$$

$$c_i(t = t_1) = c_{i,0}, i = 1, \cdots, I, \tag{34}$$

where n_0 and $c_{i,0}$ are the solutions at the starting time t_1. Typically we assume the DNP concentrations reach the equilibrium condition at the starting time, i.e.

$$c_i(t = t_1) = \frac{\beta_i}{\Lambda \lambda_i} n_0, i = 1, \cdots, I. \tag{35}$$

The equation is set up with the *PKE* input block with Rattlesnake. $n(t)$ and $c_i(t), i = 1, \cdots, I$ are treated as two primal scalar variables with order one and I respectively. It is users responsibility to provide initial conditions for these two scalar variables. Optionally, users can indicate the initial equilibrium condition to let Rattlesnake add the initial condition for DNPs $c_i(t), i = 1, \cdots, I$. ρ, Λ, $\beta_i, i = 1, \cdots, I$ and $\lambda_i, i = 1, \cdots, I$ are treated as the auxiliary scalar variables with the correct order. It is users responsibility to provide initial conditions, and/or auxiliary scalar kernels (if they change with time) for setting values for these variables. Few scalar initial conditions in MOOSE can be used for all these scalar variables as showed in Fig. 16. Few scalar auxiliary

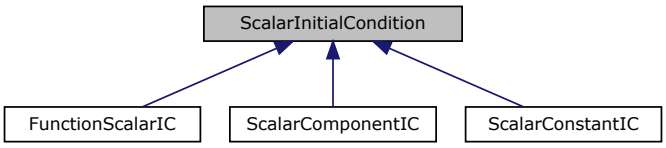


Figure 16 Scalar initial conditions in MOOSE.

kernels in MOOSE can be used to set values for the auxiliary scalar variables as showed in Fig. 17.

Rattlesnake IQS executioner has the capability of dumping the lumped PKE parameters (ρ, Λ, $\beta_i, i = 1, \cdots, I$ and $\lambda_i, i = 1, \cdots, I$) with the spatial kinetics calculations. Users can use these dumped parameters to reproduce

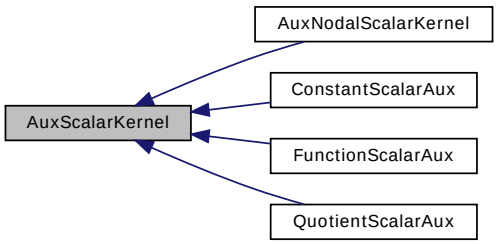


Figure 17 Scalar auxiliary kernels in MOOSE.

the transient history of the neutron production rate in PKE. The *PKE* input block contains an optional parameter, indicating in which file the dumped PKE parameters are stored, to facilitate calculations with the dumped parameters. If this parameter is set, the initial condition and auxiliary scalar kernels for these PKE parameters will be automatically added by Rattlesnake.

5.1.1 *n_delayed_groups*

Description: Number of groups of delayed neutron precursors

Data type: Integer

Default value: <required>

Syntax: PKE/n_delayed_groups

Note: This parameter will be used to specify the order of three scalar variables: $c_i, \beta_i, \lambda_i, i = 1, \cdots, I$.

5.1.2 *amplitude_variable*

Description: Amplitude primal scalar variable

Data type: String

Default value: <required>

Syntax: PKE/amplitude_variable

Note: A *first order* primal scalar variable will be added by Rattlesnake with this name for the amplitude variable n.

5.1.3 *DNP_variable*

Description: Delayed neutron precursor primal scalar variable

Data type: String

Default value: <required>

Syntax: PKE/DNP_variable

Note: A primal scalar variable with order of *n_delayed_groups* will be added by Rattlesnake with this name for the DNP concentrations $c_i, i = 1, \cdots, I$.

5.1.4 *DNP_fraction_aux*

Description: Auxiliary variable for the delayed neutron precursor fractions

Data type: String

Default value: <required>

Syntax: PKE/DNP_fraction_aux

Note: An auxiliary scalar variable with order of *n_delayed_groups* will be added by Rattlesnake with this name for the DNP fractions $\beta_i, i = 1, \cdots, I$.

5.1.5 *DNP_decay_constant_aux*

Description: Auxiliary variable for the delayed neutron precursor decay constants

Data type: String

Default value: <required>

Syntax: PKE/DNP_decay_constant_aux

Note: An auxiliary scalar variable with order of *n_delayed_groups* will be added by Rattlesnake with this name for the DNP decay constants $\lambda_i, i = 1, \cdots, I$.

5.1.6 *generation_time_aux*

Description: Auxiliary variable for the neutron generation time

Data type: String

Default value: <required>

Syntax: PKE/generation_time_aux

Note: An first-order auxiliary scalar variable will be added by Rattlesnake with this name for the generation time Λ.

5.1.7 *reactivity_aux*

Description: Auxiliary variable for the reactivity

Data type: String

Default value: <required>

Syntax: PKE/reactivity_aux

Note: An first-order auxiliary scalar variable will be added by Rattlesnake with this name for the reactivity ρ.

5.1.8 *has_initial_equilibrium*

Description: Whether or not DNP concentrations have the equilibrium on initial

Data type: Logical

Default value: True

Syntax: PKE/has_initial_equilibrium

Note: If this parameter is true, the equilibrium initial condition for DNP concentrations, Eq. (35) will added by Rattlesnake. Thus, no initial conditions for DNP concentrations will be needed from users.

5.1.9 *pke_parameter_csv*

Description: The CSV file containing dumped PKE parameters used for setting up PKE

Data type: String

Default value: <empty>

Syntax: PKE/pke_parameter_csv

Note: If this parameter is provided, the CSV file will be used to set up functions for PKE parameters ρ, Λ, β and λ. These functions are then used by Rattlesnake for adding the initial conditions and auxiliary kernels for those variables. Thus, no initial conditions and auxiliary kernels for these PKE parameters will be needed from users.

5.1.10 *verbose*

Description: Whether or not to show what objects are added by the action

Data type: Logical

Default value: False

Syntax: PKE/verbose

5.2 *MultiRegion*

There could be multiple regions in one transport system that are loosely coupled, for example the coupled reactor system. This input block makes Rattlesnake evaluate the coupled parameters including the generation time, reactivity, and so on for these multiple regions. Rattlesnake uses *MultiApp/Transfer* system to accomplish the calculations and outputs the parameters on screen at the end. Detailed explanation on these parameters can be found in Rattlesnake theory manual.

5.2.1 *transport_system*

Description: Name of the transport system which contains multiple reactor regions

Data type: String

Default value: <required>

Syntax: MultiRegion/transport_system

Note: Currently multi-region calculation does not support multi-scale.

5.2.2 *regions*

Description: Multiple space separated regions of comma separated subdomain names for the transport system

Data type: Vector of strings

Default value: <required>

Syntax: MultiRegion/regions

Note: The number of elements in this parameter separated by spaces is the number of coupled regions. Subdomains in this parameter must be a subset of the domain where the transport system is defined.

5.2.3 *adjoint_multiapp_file*

Description: MultiApp input file name for the adjoint problem

Data type: String

Default value: <required>

Syntax: MultiRegion/adjoint_multiapp_file

5.2.4 *forward_partial_multiapp_files*

Description: MultiApp input file names for all the partial forward problems each for a region.

Data type: Vector of strings

Default value: <required>

Syntax: MultiRegion/forward_partial_multiapp_files

Note: Names need to be in a certain order as the on for *adjoint_partial_multiapp_files*. The number of names need to be equal to the number of regions.

5.2.5 *adjoint_partial_multiapp_files*

Description: MultiApp input file names for all the partial adjoint problems each for a region.

Data type: Vector of strings

Default value: <required>

Syntax: MultiRegion/adjoint_partial_multiapp_files

Note: Names need to be in a certain order as the on for *forward_partial_multiapp_files*. The number of names need to be equal to the number of regions.

5.2.6 print_raw_pps

Description: True to print postprocessor values used for evaluating parameters

Data type: Logical

Default value: False

Syntax: MultiRegion/print_raw_pps

5.2.7 csv_file

Description: Output evaluated parameters in the CSV file if provided

Data type: String

Default value: <empty>

Syntax: MultiRegion/csv_file

6 *Mesh*

Mesh can be either generated from the mesh generators or loaded from a mesh file. The type of mesh generator or reader is specified by *type*. There are some parameters shared by all mesh generators, which are listed in this section. Parameters for individual mesh generators or readers are given in the subsections.

6.1 Common mesh parameters

6.1.1 *type*

Description: The type of mesh generator or reader

Data type: String

Default value: <required>

Syntax: Mesh/type

Note: Currently GeneratedBIDMesh, FileMesh, TiledMesh and ImageMesh are supported.

6.1.2 *second_order*

Description: Converts a first order mesh to a second order mesh

Data type: Logical

Default value: false

Syntax: Mesh/second_order

Note: When the simulation is using second order shape functions with CFEM (continuous finite element method), the mesh must be in the second order. This parameter need to be turned to true to use a first-order mesh for such a simulation.

6.1.3 *uniform_refine*

Description: Specify the level of uniform refinement applied to the initial mesh

Data type: Integer

Default value: 0

Syntax: Mesh/uniform_refine

Note: 0 means no uniform refinement. One level of uniform refinement generally increase the number of element by factor 2, 4 and 8 in 1D, 2D and 3D respectively.

6.1.4 *construct_side_list_from_node_list*

Description: Whether construct side lists from the nodesets in the mesh (i.e. if every node on a give side is in a nodeset then add that side to a sideset)

Data type: Logical

Default value: false

Syntax: Mesh/construct_side_list_from_node_list

6.1.5 *skip_partitioning*

Description: If true the mesh won't be partitioned

Data type: Logical

Default value: false

Syntax: Mesh/skip_partitioning

Note: This may cause large load imbalanced but is currently required if you have a simulation containing uniform refinement, adaptivity and stateful material properties.

6.1.6 *block_id*

Description: IDs of the block id/name pairs

Data type: Vector of integers

Default value: <empty>

Syntax: Mesh/block_id

6.1.7 *block_name*

Description: Names of the block id/name pairs

Data type: Vector of strings

Default value: <empty>

Syntax: Mesh/block_name

Note: This parameter must correspond with *block_id*. The assigned block names can be used throughout the input file. They will also be written to Exodus/XDA/XDR files.

6.1.8 *boundary_id*

Description: IDs of the boundary id/name pairs

Data type: Vector of integers

Default value: <empty>

Syntax: Mesh/boundary_id

6.2 *GeneratedBIDMesh*

This mesh generator generates a regular mesh with uniformly distributed elements. This mesh generator is the same as MOOSE *GeneratedMesh* except that it provides a parameter *subdomain* to let user assign the block IDs of all generated elements.

6.2.1 *dim*

Description: The dimension of the mesh to be generated

Data type: Enumeration (/1/2/3/)

Default value: <required>

Syntax: Mesh/dim

6.2.2 *nx*

Description: Number of elements in the X direction

Data type: Integer

Default value: 1

Syntax: Mesh/nx

6.2.3 *xmin*

Description: Lower X Coordinate of the generated mesh

Data type: Real

Default value: 0

Syntax: Mesh/xmin

6.2.4 *xmax*

Description: Upper X Coordinate of the generated mesh

Data type: Real

Default value: 1

Syntax: Mesh/xmax

6.2.5 *ny*

Description: Number of elements in the Y direction

Data type: Integer

Default value: 1

Syntax: Mesh/ny

Note: This parameter will be ignored when *dim* is 1.

6.2.6 *ymin*

Description: Lower Y Coordinate of the generated mesh

Data type: Real

Default value: 0

Syntax: Mesh/ymin

Note: This parameter will be ignored when *dim* is 1.

6.2.7 *ymax*

Description: Upper Y Coordinate of the generated mesh

Data type: Real

Default value: 1

Syntax: Mesh/ymax

Note: This parameter will be ignored when *dim* is 1.

6.2.8 *nz*

Description: Number of elements in the Z direction

Data type: Integer

Default value: 1

Syntax: Mesh/nz

Note: This parameter will be ignored when *dim* is 1 or 2.

6.2.9 *zmin*

Description: Lower Z Coordinate of the generated mesh

Data type: Real

Default value: 0

Syntax: Mesh/zmin

Note: This parameter will be ignored when *dim* is 1 or 2.

6.2.10 *zmax*

Description: Upper Z Coordinate of the generated mesh

Data type: Real

Default value: 1

Syntax: Mesh/zmax

Note: This parameter will be ignored when *dim* is 1 or 2.

6.2.11 *elem_type*

Description: The type of element from libMesh to generate

Data type: Enumeration (/EDGE EDGE2 EDGE3 EDGE4 QUAD QUAD4 QUAD8 QUAD9 TRI3 TRI6 HEX HEX8 HEX20 HEX27 TET4 TET10 PRISM6 PRISM15 PRISM18 PYRAMID5 PYRAMID13 PYRAMID14/)

Default value: EDGE2/QUAD4/HEX8

Syntax: Mesh/elem_type

Note: The default value varies with dimension *dim*. It is EDGE2, QUAD4 and HEX8 for 1D, 2D and 3D mesh respectively. It is noted that EDGE, EDGE2, EDGE3 and EDGE4 are the supported 1D element types; QUAD, QUAD4, QUAD8, QUAD9, TRI3 and TRI6 are the supported 2D element types; HEX, HEX8, HEX20, HEX27, TET4, TET10, PRISM6, PRISM15, PRISM18, PYRAMID5, PYRAMID13 and PYRAMID14 are the supported 3D element types.

6.2.12 *distribution*

Description: Whether or not to distribute the mesh among processors

Data type: Enumeration (/PARALLEL/SERIAL/DEFAULT/)

Default value: DEFAULT

Syntax: Mesh/distribution

Note: PARALLEL means always distributing the mesh by using libMesh::ParallelMesh; SERIAL means always not distributing the mesh by using libMesh::SerialMesh; DEFAULT means using libMesh::SerialMesh unless '–parallel-mesh' is specified on the command line. PARALLEL is recommended when the mesh contains more than 1 million elements.

6.2.13 *partitioner*

Description: Specifies a mesh partitioner to use when splitting the mesh for a parallel computation

Data type: Enumeration (/default/metis/parmetis/linear/centroid/hilbert_sfc/morton_sfc/)

Default value: default

Syntax: Mesh/partitioner

6.2.14 *centroid_partitioner_direction*

Description: Specifies the sort direction if using the centroid partitioner

Data type: Enumeration (/x/y/z/radial/)

Default value: <empty>

Syntax: Mesh/centroid_partitioner_direction

6.2.15 *subdomain*

Description: Block IDs of elements

Data type: Vector of integers

Default value: <empty>

Syntax: Mesh/subdomain

Note: If this parameter is empty, block IDs of all elements are assigned to 0. If this parameter is provided, the size of this parameter must agree with the number of generated elements. The number of elements in 1D is equal to *nx*. The number of elements in 2D is equal to $nx \times ny$ with *elem_type* being QUAD, QUAD4, QUAD8 or QUAD9 and is equal to $2nx \times ny$ with *elem_type* being TRI3 or TRI6. The number of elements in 3D is equal to $nx \times ny \times nz$ with *elem_type* being HEX, HEX8, HEX20 or HEX27 and is equal to $6nx \times ny$ with *elem_type* being TET4 or TET10, and is equal to $3nx \times ny$ with *elem_type* being PRISM6, PRISM15, PRISM18, PYRAMID5, PYRAMID13 or PYRAMID14.

6.3 *CartesianMesh*

This mesh generator generates a regular Cartesian mesh. Elements do not have to be uniformly distributed in the generated mesh. The element type will be EDGE2, QUAD4 and HEX8 when *dim* is equal to 1, 2 and 3 respectively.

6.3.1 *dim*

Description: The dimension of the mesh to be generated

Data type: Enumeration (/1/2/3/)

Default value: <required>

Syntax: Mesh/dim

6.3.2 *dx*

Description: Intervals in the X direction

Data type: Vector of reals

Default value: <required>

Syntax: Mesh/dx

6.3.3 *ix*

Description: Number of grids in all intervals in the X direction (default to all one)

Data type: Vector of integers

Default value: <empty>

Syntax: Mesh/ix

Note: This size of this parameter must be equal to the size of *dx*.

6.3.4 *dy*

Description: Intervals in the Y direction

Data type: Vector of reals

Default value: <empty>

Syntax: Mesh/dy

Note: This parameter is required for 2D and 3D. It will be ignored for 1D.

6.3.5 *iy*

Description: Number of grids in all intervals in the Y direction (default to all one)

Data type: Vector of integers

Default value: <empty>

Syntax: Mesh/iy

Note: This size of this parameter must be equal to the size of *dy*. It will be ignored for 1D.

6.3.6 *dz*

Description: Intervals in the Z direction

Data type: Vector of reals

Default value: <empty>

Syntax: Mesh/dz

Note: This parameter is required for 3D. It will be ignored for 1D and 2D.

6.3.7 *iz*

Description: Number of grids in all intervals in the Z direction (default to all one)

Data type: Vector of integers

Default value: <empty>

Syntax: Mesh/iz

Note: This size of this parameter must be equal to the size of *dz*. It will be ignored for 1D and 2D.

6.3.8 *subdomain_id*

Description: Block IDs (default to all zero)

Data type: Vector of integers

Default value: <empty>

Syntax: Mesh/subdomain_id

Note: when both *dy* and *dz* is provided, the size of this parameter must be equal to $nx \times ny \times nz$, where nx, ny and nz are the sizes of *dx*, *dy* and *dz* respectively. Otherwise, it must be equal to $nx \times ny$ when *dy* is provided, or nx when *dy* is not provided.

6.3.9 *distribution*

Refer to *distribution* in GeneratedBIDMesh.

6.3.10 *partitioner*

Refer to *partitioner* in GeneratedBIDMesh.

6.3.11 *centroid_partitioner_direction*

Refer to *centroid_partitioner_direction* in GeneratedBIDMesh.

6.4 Hexagonal meshes

(to be added.)

6.5 *FileMesh*

This will load a mesh file. The mesh file has to be pre-generated. For complicated geometries, we generally use CUBIT from Sandia National Laboratories. CUBIT can be licensed from CSimSoft for a fee depending on the type of organization you work for. Other mesh generators can work as long as they output a file format that libMesh reads. Currently the supported mesh format are listed in Table 21 by libMesh.

Table 21 Supported mesh format.

File extension	Mesh format
*.e	Sandia's ExodusII format
*.exd	Sandia's ExodusII format
*.gmv	LANL's General Mesh Viewer format
*.mat	Matlab triangular ASCII file
*.n	Sandia's Nemesis format
*.nem	Sandia's Nemesis format
*.off	OOGL OFF surface format
*.ucd	AVS's ASCII UCD format
*.unv	I-deas Universal format
*.vtu	Paraview VTK format
*.inp	Abaqus .inp format
*.xda	libMesh ASCII format
*.xdr	libMesh binary format
*.gz	any above format gzipped
*.bz2	any above format bzip2'ed
*.xz	any above format xzipped
*.cpa	libMesh Checkpoint ASCII format
*.cpr	libMesh Checkpoint binary format

6.5.1 *distribution*

Refer to *distribution* in GeneratedBIDMesh.

6.5.2 *partitioner*

Refer to *partitioner* in GeneratedBIDMesh.

6.5.3 *centroid_partitioner_direction*

Refer to *centroid_partitioner_direction* in GeneratedBIDMesh.

6.5.4 *file*

Description: The name of the mesh file to read

Data type: String

Default value: <required>

Syntax: Mesh/file

6.6 *TiledMesh*

6.7 *ImageMesh*

6.8 INSTANT mesh generators

This will invoke INSTANT to generate a mesh and write it into an Exodus file.

6.9 Mesh modifiers

Mesh modifiers further modify the mesh after it has been created. Possible modifications include: adding node sets and/or side sets, translating, rotating, and scaling the mesh points, assigning block IDs for elements, and etc.

6.9.1 MOOSE mesh modifiers

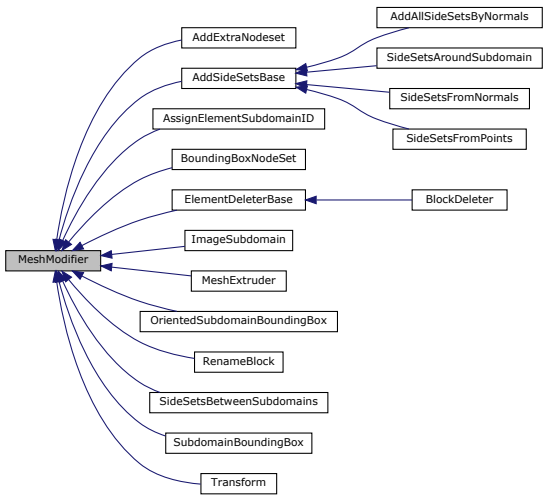


Figure 18 Moose mesh modifiers.

The complete list of MOOSE mesh modifiers can be found in Fig. 18. Their parameters are fairly strait-forward and can be found in MOOSE documents. We found the mesh extruder is frequently used so we also document it here.

6.9.2 *MeshExtruder*

This modifier takes a 1D or 2D mesh and extrudes it to 2D or 3D, respectively. Triangles are extruded to prisms (wedges). Quadrilaterals are extruded to hexahedra. The extruded mesh contains $N \times L$ elements, where N is the number of elements in the original mesh and L is the number of layers to be extruded. Newly-created top and bottom sidesets can be named by the user. Their numbers of sides are equal to N. Sidesets are extruded and preserved. Their number of sides is L times of their original number of sides.

6.9.2.1 *depends_on*

Description: The MeshModifiers that this modifier relies upon (i.e. must execute before this one)

Data type: Vector of strings

Default value: <empty>

Syntax: MeshModifiers/*/depends_on

Note: This parameter is useful when there are multiple mesh modifiers and the sequence of their runs are significant.

6.9.2.2 *num_layers*

Description: The number of layers in the extruded mesh (L)

Data type: Integer

Default value: <required>

Syntax: MeshModifiers/*/num_layers

6.9.2.3 *extrusion_vector*

Description: The direction and length of the extrusion

Data type: Vector of reals

Default value: <required>

Syntax: MeshModifiers/*/extrusion_vector

Note: The dimension of this parameter is 2 or 3 for 1D or 2D mesh respectively. The L2 norm of this parameter indicates how far the mesh is going to be extruded. All layers are distributed evenly at this moment.

6.9.2.4 *bottom_sideset*

Description: The boundary that will be applied to the bottom of the extruded mesh

Data type: String

Default value: <empty>

Syntax: MeshModifiers/*/bottom_sideset

6.9.2.5 *top_sideset*

Description: The boundary that will be applied to the top of the extruded mesh

Data type: String

Default value: <empty>

Syntax: MeshModifiers/*/top_sideset

6.9.2.6 *existing_subdomains*

Description: The subdomains that will be remapped for specific layers

Data type: Vector of integers

Default value: <empty>

Syntax: MeshModifiers/*/existing_subdomains

6.9.2.7 *layers*

Description: The layers where the *existing_subdomain* will be remapped to new ids

Data type: Vector of integers

Default value: <empty>

Syntax: MeshModifiers/*/layers

Note: The layer ID starts from 0.

6.9.2.8 *new_ids*

Description: The list of new ids

Data type: Vector of integers

Default value: <empty>

Syntax: MeshModifiers/*/new_ids

Note: This list should be either length *existing_subdomains* or *existing_subdomains*$\times$*layers*. In the former case, the new IDs will be assigned to all layers specified by *layers*. In the second case, the new IDs will be assigned to all layers individually according the ordering in *layers*.

6.9.3 *RandomNodeDisplacement*

This mesh modifier moves the nodes in the mesh randomly. The modifier will try to find the minimum distance h_i to any nodes for every node i. Then it generates one random number r from -1 to 1 in 1D and uses the number to obtain the new coordinate with $x_i + rch_i$, where c is a given fraction, for each node. Or it generates two random numbers r_1 and r_2 from 0 to 1 in 2D and sets the new coordinate with $(x_i + r_1 ch_i \cos(r_2 2\pi), y_i + r_1 ch_i \sin(r_2 2\pi))$, for each node. Or it generates two random numbers r_1 and r_2 from 0 to 1 and another random number r_3 from -1 to 1 in 3D and sets the new coordinate with $(x_i + r_1 ch_i \cos(r_2 2\pi)\sqrt{1-r_3^2}, y_i + r_1 ch_i \sin(r_2 2\pi)\sqrt{1-r_3^2}, z_i + r_1 ch_i r_3)$, for each node. Users can easily create an irregular mesh with a regular mesh generator and this mesh modifier.

6.9.3.1 *depends_on*

Refer to *depends_on* in MeshExtruder.

6.9.3.2 *max_perturb*

Description: Maximum fractional displacement of mesh nodes

Data type: Real

Default value: 0

Syntax: MeshModifiers/*/max_perturb

Note: This parameter is the fraction number c.

6.9.3.3 *perturb_boundary*

Description: Whether the boundary nodes are displaced

Data type: Logical

Default value: false

Syntax: MeshModifiers/*/perturb_boundary

6.9.3.4 *seed*

Description: Random seed for initializing random number sequence

Data type: Integer

Default value: 1

Syntax: MeshModifiers/*/seed

6.9.4 *SplitConformingMeshForMortar*

This mesh modifier basically splits a conforming mesh, i.e. a mesh without hanging nodes, for calculations with mortar FEM. This modifier will create blocks for mortar faces among subdomains with the naming convention '*interface-#1-to-#2*', where #1 is the number of the from subdomain ID and #2 is the number of the to subdomain ID. It is noted that subdomain here is different from libMesh subdomain, which is indeed just block. It will also disconnect all subdomains by duplicating interface nodes and reset the connectivity of all elements on subdomain interfaces to these nodes. It will also split existing side sets with respect to all subdomains into three parts with naming '*#ss_name-#1-boundary*', '*#ss_name-#1-interior*' and '*#ss_name-#1-outside*', where *#ss_name* is the name of the side set (if the name is missing, it will be the side set ID), #1 is the subdomain ID. It will also create new side sets between subdomains with naming '*interface-#1-to-#2*', where #1 is the number of the from subdomain ID and #2 is the number of the to subdomain ID. This modifier will be automatically added when *is_mesh_split* is true and multiscale calculation, i.e. multiple transport systems, is pursued.

6.9.4.1 *depends_on*

Refer to *depends_on* in MeshExtruder.

6.9.4.2 *num_subdomains*

Description: Number of subdomains for mortar FEM

Data type: Integer

Default value: <required>

Syntax: MeshModifiers/*/num_subdomains

6.9.4.3 *subdomain_blocks*

Description: Multiple space separated groups of comma separated block names.

Data type: Vector of strings

Default value: <required>

Syntax: MeshModifiers/*/subdomain_blocks

Note: Number of elements of this parameter must be equal to *num_subdomains*. Elements of this parameter are the comma separated block name. An example of this parameter could be '1,2 3,4', where block 1 and 2 will form the first subdomain and block 3 and 4 will form the second subdomain.

6.9.4.4 *subdomain_names*

Description: Subdomain names

Data type: Vector of strings

Default value: <empty>

Syntax: MeshModifiers/*/subdomain_names

Note: Users can optionally assign subdomain names with this parameter. If this parameter is given, the size of this parameter must be equal to *num_subdomains*.

6.9.5 *SplitSideSetsAndAddNormals*

This mesh modifier is useful to LS-CFEM-SN for imposing the surface source boundary condition strongly. (to be added)

7 *Functions*

7.1 MOOSE Functions

A MOOSE function is a function in space and in time. It can be as simple as an expression or as complicated as a solution function defined on an unstructure mesh.

7.1.1 MOOSE Functions in MOOSE Framework

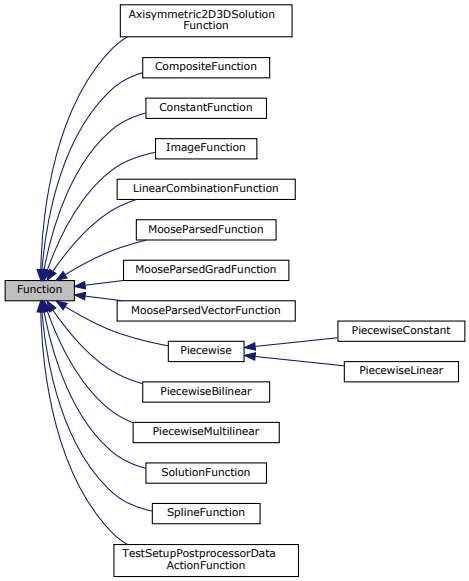


Figure 19 Moose functions.

The complete list of MOOSE functions in MOOSE can be found in Fig. 19. Their parameters can be found in MOOSE documents. Because the functionalities of these functions and their parameters are fairly straitforward, we will not replicate them here.

7.1.2 *SlopeFunction*

This function is a simpler version of *PiecewiseLinear* in that it only provides a function in time with the abscissa and ordinate data directly from the input. It is illustrated in Fig. 20.

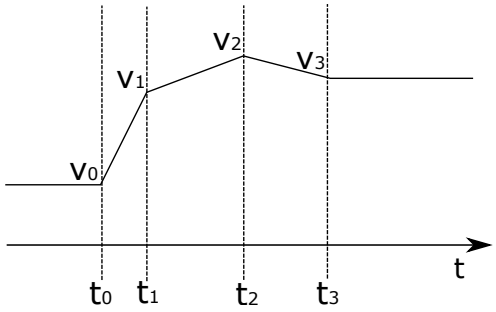


Figure 20 Piece-wise linear function by *SlopeFunction*.

7.1.2.1 *timep*

Description: The time points

Data type: Vector of reals

Default value: <required>

Syntax: Functions/*/timep

7.1.2.2 *value*

Description: The function values at all time points

Data type: Vector of reals

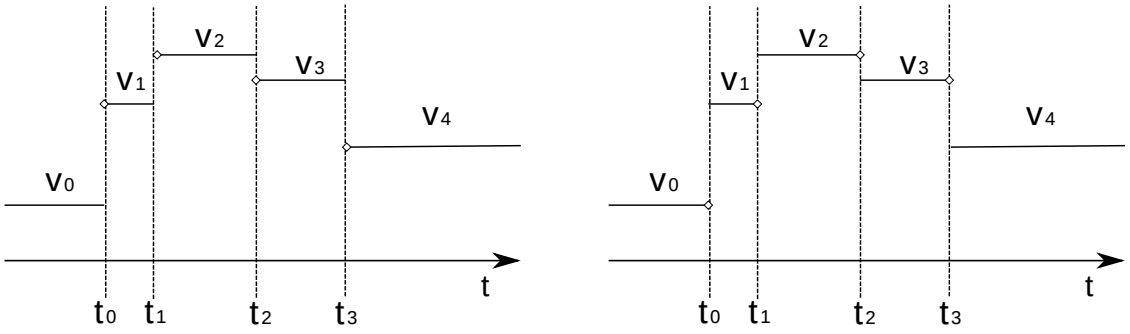
Default value: <required>

Syntax: Functions/*/value

Note: The size of this parameter must be equal to the size of *timep*.

7.1.3 *StepFunction*

This function is a simpler version of *PiecewiseConstant* in that it only provides a function in time with the abscissa and ordinate data directly from the input. It is illustrated in Fig. 21.



(a) left continuous (b) right continuous

Figure 21 Piece-wise constant function by *StepFunction*.

7.1.3.1 *timep*

Description: The time points where steps happen

Data type: Vector of reals

Default value: <required>

Syntax: Functions/*/timep

7.1.3.2 *value*

Description: The constant values of all time steps

Data type: Vector of reals

Default value: <required>

Syntax: Functions/*/value

7.1.3.3 *direction*

Description: Direction to look to find value

Data type: Enumeration (/left/right/)

Default value: left

Syntax: Functions/*/direction

Note: Direction left means the function is left continuous i.e. $\lim_{\Delta t \to 0^+} f(t - \Delta t) = f(t)$.

7.2 Transport Solution Functions

Transport solution functions are special MOOSE functions. They provide the multigroup radiation transport solutions in energy groups, space, time and angle. They can be used to specify the volumetric or boundary external sources. They can be added with the *Functions* input block. Rattlesnake has several built-in transport solution functions that users can directly use.

7.2.1 *ConstantSourceFunction*

This function specifies a function which is constant in space, time, angle and in every individual energy groups. The function could have different value in different energy groups.

$$f_g(\vec{r}, \vec{\Omega}, t) = \frac{v_g}{c_d}, \tag{36}$$

where v_g is the scalar value of energy group g; c_d is the dimension-dependent normalization:

$$c_d = \begin{cases} 2, & 1D \\ 2\pi, & 2D \\ 4\pi, & 3D \end{cases}. \tag{37}$$

This function can be typically constructed and used inline with the syntax $v_1, v_2, \cdots, v_G$.

7.2.1.1 *Dimension*

Description: Dimension

Data type: Integer

Default value: <required>

Syntax: Functions/*/Dimension

Note: This parameter must be the same as the mesh dimension.

7.2.1.2 *NG*

Description: Number of energy groups

Data type: Integer

Default value: <required>

Syntax: Functions/*/NG

7.2.1.3 *value*

Description: Group-dependent strengths

Data type: Vector of reals

Default value: <required>

Syntax: Functions/*/value

Note: The size of this parameter must be equal to *NG*.

7.2.2 *PulsedSourceFunction*

This function specifies a isotropic function pulsed in space at a given location $\vec{r}_0$ and in time at the first time step. It allows varying pulse strength in energy groups.

$$f_g(\vec{r}, \vec{\Omega}, t) = \frac{1}{c_d} v_g e^{\frac{|\vec{r}-\vec{r}_0|^2}{2c^2}} \delta(t - \Delta t), \tag{38}$$

where v_g is the strength of energy group g; c_d is the dimension-dependent normalization; c is the standard derivation of the Gaussian distribution.

7.2.2.1 *Dimension*

Refer to *Dimension* in ConstantSourceFunction.

7.2.2.2 *NG*

Refer to *NG* in ConstantSourceFunction.

7.2.2.3 *value*

Refer to *value* in ConstantSourceFunction.

7.2.2.4 *center*

Description: Position of the center of the Gaussian (same for each energy group)

Data type: Vector of reals

Default value: <required>

Syntax: Functions/*/center

Note: The size of this parameter must be equal to the mesh dimension.

7.2.2.5 *constant*

Description: The standard deviation of the gaussian

Data type: Real

Default value: 1

Syntax: Functions/*/constant

7.2.3 *DirectionalSourceFunction*

This function specifies a delta function in angle specifically in a direction in a given angular quadrature, constant in space and time with varying strength in energy groups.

$$f_g(\vec{r}, \vec{\Omega}, t) = v_g \delta(\vec{\Omega} - \vec{\Omega}_m), \tag{39}$$

where v_g is the strength of energy group g and $\vec{\Omega}_m$ is a direction with index m of a given angular quadrature.

7.2.3.1 *Dimension*

Refer to *Dimension* in ConstantSourceFunction.

7.2.3.2 *NG*

Refer to *NG* in ConstantSourceFunction.

7.2.3.3 *strength*

Description: Strength of the source of all energy groups

Data type: Vector of reals

Default value: <required>

Syntax: Functions/*/strength

Note: The size of this parameter must be equal to *NG*.

7.2.4 *FilePNTransportSolutionFunction*

This function turns a PN solution into a transport function.

$$f_g(\vec{r},\vec{\Omega},t) = \begin{cases} \sum_{l=0}^{PN} \frac{2l+1}{2} P_l(\mu)\phi_{g,l}(\vec{r},t), & 1D \\ \sum_{l=0}^{PN} \sum_{m=0}^{l} \frac{2l+1}{2\pi} Y_{l,m}(\vec{\Omega})\phi_{g,l,m}(\vec{r},t), & 2D \\ \sum_{l=0}^{PN} \sum_{m=-l}^{l} \frac{2l+1}{4\pi} Y_{l,m}(\vec{\Omega})\phi_{g,l,m}(\vec{r},t), & 3D \end{cases} \tag{40}$$

where PN is the PN order; P_l are the Legendre polynomials; μ is the cosine of the polar angle; $Y_{l,m}$ are the real normalized spherical harmonics; ϕ_l or $\phi_{l,m}$ are the angular flux moments. The solution must have been loaded from a file with a MOOSE *SolutionUserObject*.

7.2.4.1 *Dimension*

Refer to *Dimension* in ConstantSourceFunction.

7.2.4.2 *NG*

Refer to *NG* in ConstantSourceFunction.

7.2.4.3 *csphase*

Description: Whether or not Condon-Shortley phase is included in the function

Data type: Logical

Default value: False

Syntax: Functions/*/csphase

Note: If the solution is generated by Rattlesnake, Condon-Shortley phase is not included for the evaluation of angular flux moments, which is why the default value of this parameter is false.

7.2.4.4 *solution*

Description: The *SolutionUserObject* to extract data from

Data type: String

Default value: <required>

Syntax: Functions/*/solution

Note: The solution user object must load the angular flux moments in the certain order: $m = low(l), \cdots, up(l); l = 0, \cdots, PN; g = 1, \cdots, G$, where $low(l)$ is 0, 0 and $-l$ and $up(l)$ is 0, l and l for one-dimension, two-dimension and three-dimension respectively and PN is the spherical harmonics order and G is the number of energy groups.

7.2.4.5 *scale_factor*

Description: Scale factor (a) to be applied to the solution (x): $ax + b$

Data type: Real

Default value: 1

Syntax: Functions/*/scale_factor

7.2.4.6 *add_factor*

Description: Add this value (b) to the solution (x): $ax + b$

Data type: Real

Default value: 0

Syntax: Functions/*/add_factor

7.2.5 Customized *TransportSolutionFunction* (Advanced)

1. To determine the right base class: PN, SN transport solution function;
2. To implement your function: Using Larsen_2D as an example;
3. To register your function in *RattleSnakeApp*;
4. To use your function as other transport functions;
5. To consider adding it into Rattlesnake officially.

7.3 Adjustable Function

AdjustableFunction is a MOOSE function, the same as MOOSE *ConstantFunction* except that

- Parameter 'value' is renamed as 'InitialParam';
- It has two additional methods:

```
Real AdjustableFunction::getParameter();
void AdjustableFunction::setParameter(Real v);
```

AdjustableFunction is used by CriticalitySearch for finding the criticality parameter.

7.4 Phase Functions

The phase function p is the angular distribution of a particle scattered by a background medium. It is a function of the scattering angle θ or the cosine of the scattering angle $\mu = \cos(\theta)$. It satisfies the normalization condition:

$$\frac{1}{4\pi}\int_{2\pi}\int_{-1}^{1} p(\mu)d\mu d\varphi = 1, \tag{41}$$

or

$$\frac{1}{2}\int_{-1}^{1} p(\mu)d\mu = 1, \tag{42}$$

More phase functions can be implemented. Phase functions can be added as the normal MOOSE functions.

7.4.1 *IsotropicPhaseFunction*

This phase function is independent on μ, i.e. $p(\mu) = 1$. This function has no parameters.

7.4.2 *Rayleigh*

This Rayleigh phase function is $p(\mu) = \frac{3}{4}(1 + \mu^2)$. This function has no parameters.

7.4.3 *HenyeyGreenstein*

This Henyey-Greenstein phase function is $p(\mu) = \frac{(1-g^2}{1+g^2-2g\mu)^{\frac{2}{3}}}$. It is Legendre expansion is $p(\mu) = \sum_{n-1}^{\infty}(2n + 1)g^n P_n(\mu)$. This function has one parameter g, which is in range of $[-1, 1]$. When $g > 0$, forward scattering is dominant, while for $g < 0$, backward scattering predominates.

7.4.3.1 *g*

Description: Henyey-Greenstein parameter g

Data type: String

Default value: 0

Syntax: Functions/*/g

Note: This parameter can be a name of a general function.

8 *Materials*

8.1 *type*

Description: To specify the material type

Data type: string

Default value: <required>

Syntax: Materials/*/type

Note: The supported types include:

- ConstantNeutronicsMaterial: suitable for calculation with the fixed macroscopic cross sections, like in benchmark problems where cross sections are given by the problem description.
- FunctionNeutronicsMaterial: suitable for calculation with the macroscopic cross sections being spatial and time functions. It is typically used for MMS (method of manufactured solutions) or for transient benchmark problems.
- MixedNeutronicsMaterial: suitable for checking cross sections in YAKXS format.
- CoupledFeedbackNeutronicsMaterial: suitable for multiphysics or depletion calculations where tabulated cross sections are required.
- CRoddedNeutronicsMaterial: suitable for modeling control-rod movements with few different constant neutronics materials.
- ConstantTRMaterial: suitable for calculation with the fixed macroscopic cross sections, like in benchmark problems where cross sections are given by the problem description.
- FunctionTRMaterial: suitable for calculation with the macroscopic cross sections being spatial and time functions. It is typically used for MMS (method of manufactured solutions) or for transient benchmark problems.

Mixed types of materials can be used in one discretization scheme. All types of materials can be used for common particle transport. The material properties declared by materials not used by common particle transport will be ignored. Only neutronics materials can be used for neutron transport. Only thermal radiation (TR) materials can be used for thermal radiation transport.

8.2 Neutronics materials

Neutronics materials are special because they are interacting with TransportSystems in the following ways:

1. Neutronics materials cannot be defined across two discretization schemes even they are having exactly the same scheme.
2. The coverage of neutronics material of the entire domain can be checked by setting check_neutronics_material_coverage to true.
3. Neutronics materials cannot be used independently on discretization schemes

8.2.1 Brief introduction to YAKXS

YAKXS is a general toolkit for managing the multigroup cross sections for neutron transport calculations. It is developed under YAK, the MOOSE-based raidation transport module, so it is named as YAKXS. It provides a XML cross section library format for storing the multigroup cross sections. YAKXS provides operations like cross section interpolation, mixing, fitting, collapsing, and etc. Users and developers are shielded from the complexity of these operations.

YAKXS contains ten c++ classes and one *Utility* sub-namespace:

1. *Utility* sub-namespace collects a set of useful data structures and functions for cross section processing, like converting a isotope name to its mat number, getting the default isotope class of an isotope, etc.
2. *MultigroupLibrary* holds the raw data loaded from a library in YAKXS format. It comes along with manipulators, accessors and writers. *MultigroupLibrary* can be outputted into and be constructed from a data file in the YAKXS XML or binary format.
3. *MixingTable* regulates the raw data for the mixing operation. It holds the microscopic cross sections at a particular state. *MixingTable* can be constructed from *MultigroupLibrary* through a selection or interpolation operation.
4. *Mixture* contains the macroscopic cross sections, which can be directly used by the transport solvers. *Mixture* can be outputted into a data file in the INSTANT XML format. Mixture however is not constructed from a data file in the INSTANT XML format.
5. *MixedMultigroupLibrary* contains the macroscopic cross sections of all state points in *MultigroupLibrary*. *MixedMultigroupLibrary* can be obtained from *MultigroupLibrary* through mixing or folding operations. When folding operation is performed, new variables will be introduced and the atomic density dependency on these new variables is folded into the generated library. *MixedMultigroupLibrary* can be outputted into and be constructed from a data file in the YAKXS XML or binary format.
6. *InputXS* is one of the macroscopic cross section holders. *InputXS* provides to transport solvers constant macroscopic cross sections. *InputXS* can be constructed from *Mixture*. *InputXS* can be outputted into a data file in the INSTANT XML format and can be constructed from a data file in the INSTANT XML format.
7. *PerturbedInputXS* is one of the macroscopic cross section holders. *PerturbedInputXS* provides to transport solvers a simple model for cross sections with various feedback effects. *PerturbedInputXS* can be constructed from a RELAP-5 input file containing the cross sections data with all the perturbation coefficients or from a *MixedMultigroupLibrary* through fitting.
8. *FunctionInputXS* is one of the macroscopic cross section holders. Every cross section can be a function varying in space and time.
9. *YAKXSCreator* is designed for cross section generators to create the multigroup library in YAKXS format.
10. *TransmutationLibrary* holds the raw data loaded from a transmutation library.
11. *WorkingTransmutationLibrary* contains the preprocessed data from *TransmutationLibrary* which can be used for transmutation calculations directly. The number of isotopes in the *WorkingTransmutationLibrary* does not have to be the full list of isotopes in the *TransmutationLibrary*, from which the working library are generated.

All of them are included in *YAKXS namespace*. Cross section processing capabilities are built with the classes and utility functions. The relations among these classes are illustrated into Fig. 22. Rattlesnake neutronics material interact with YAKXS for loading the cross sections and evaluate the material properties required for neutron transport calculations.

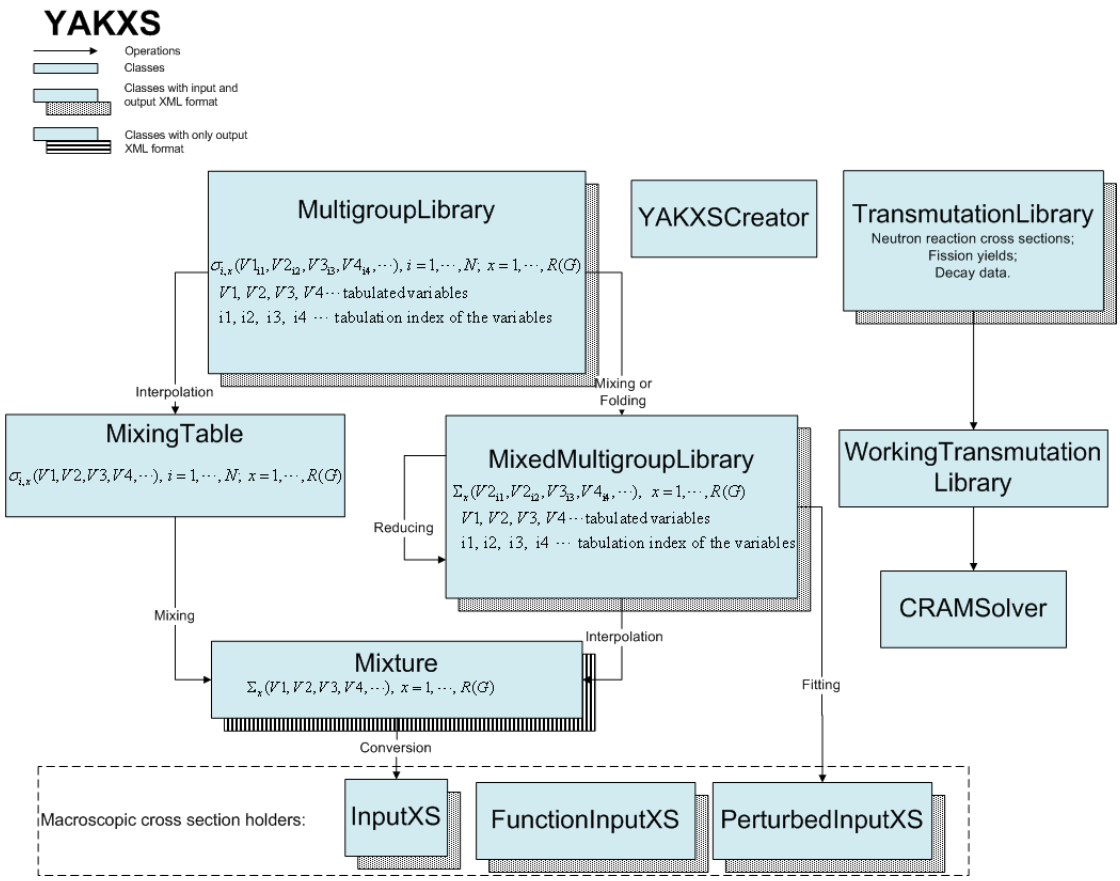


Figure 22 Graphical view of YAKXS.

8.2.2 Material properties declared by neutronics materials

All neutronics materials declare a set of material properties, which are listed in Table 22. This list could be useful for advanced users who want to utilize those properties for postprocessing. Those properties are declared on blocks on which the material are defined. The list of material properties can also be revealed during run-time by setting *show_material_props* in the *Debug* block of the input file to true. It is noted that not all of them are evaluated on element faces. Diffusion coefficients are evaluated on element faces for diffusion calculations when *scheme* is equal to CFEM-Diffusion or DFEM-Diffusion. Total cross sections are evaluated on element faces *scheme* is equal to SAAF-CFEM-PN, LS-CFEM-SN or LS-CFEM-PN. sigma_scattering_n<l> and neutron velocities are evaluated for SAAF-CFEM-PN and LS-CFEM-PN.

Table 22 Neutronics material properties.

Name	Notation	Type	When is declared
nu_sigma_fission	$\nu\sigma_{f,g}, g = 1, \cdots, G$	std::vector<Real>	Fissile[a]
nu_sigma_fission_g<g>	$\nu\sigma_{f,g}$	Real	Fissile
kappa_sigma_fission	$\kappa\sigma_{f,g}, g = 1, \cdots, G$	std::vector<Real>	Fissile & Plus[b]
kappa_sigma_fission_g<g>	$\kappa\sigma_{f,g}$	Real	Fissile & Plus
sigma_fission	$\sigma_{f,g}, g = 1, \cdots, G$	std::vector<Real>	Fissile & Plus
sigma_fission_g<g>	$\sigma_{f,g}$	Real	Fissile & Plus
fission_spectrum	$\chi_{p,g}, g = 1, \cdots, G$	std::vector<Real>	Fissile & Plus
beta_i<i>	β_i	Real	Fissile & $I > 0$[c]
lambda_i<i>	λ_i	Real	Fissile & $I > 0$
chi_delay_i<i>	$\chi_{g,i}, g = 1, \cdots, G$	std::vector<Real>	Fissile & Plus
neutron_speed_g<g>	v_g	Real	Transient[d]
diffusion_coefficient_g<g>	D_g	Real	Diffusion[e]
vector_diffusion_coefficient_g<g>	D_g	RealVector	Diffusion
tensor_diffusion_coefficient_g<g>	D_g	RealTensor	Diffusion
sigma_removal_g<g>	$\sigma_{r,g}$	Real	Diffusion
sigma_total_g<g>	$\sigma_{t,g}$	Real	NotDiffusion[f]
sigma_absorption_g<g>	$\sigma_{a,g}$	Real	Plus
sigma_nalpha_g<g>	$\sigma_{\alpha,g}$	Real	Provided[g]
sigma_capture_g<g>	$\sigma_{c,g}$	Real	Provided
chi_g<g>	χ_g	Real	Fissile & FissionPattern[h]
chi_delay_g<g>	$\chi_{g,i}, i = 1, \cdots, I$	std::vector<Real>	Fissile & FissionPattern
sigma_scattering_g<p>_g<g>	$\sigma_{s,l}^{p\to g}, l = 0, \cdots, L$	std::vector<Real>	ScatteringPattern[i]
sigma_scattering_n<l>	$\sigma_{s,l}^{p\to g}, p, g = 1, \cdots, G$	CSR<Real>	NonzeroScattering[j]

[a] Fissile means that the material is fissile;
[b] Plus means that *plus* is set to true;
[c] $I > 0$ means that *n_delay_groups* is greater than 0; Note that the material has to be fissile to provide delayed neutron data;
[d] Transient means that *equation_type* is equal to *transient*;
[e] Diffusion means that *scheme* is equal to CFEM-Diffusion or DFEM-Diffusion;
[f] NotDiffusion means that *scheme* is equal to any other than CFEM-Diffusion and DFEM-Diffusion;
[g] Provided means that the cross section data is provided by the input;
[h] FissionPattern means that the property will be declared for the group with non-zero fission neutron yields;
[i] ScatteringPattern means that the property will be declared for the non-zero scattering entries;
[j] NonzeroScattering means that there is scattering for this material.

8.2.3 ConstantNeutronicsMaterial

This material uses *InputXS* in YAKXS for managing the constant macroscopic cross sections. This material provides the basic capability of doing multigroup transport calculations.

8.2.3.1 *block*

Description: List of blocks where the material is defined on

Data type: Vector of strings

Default value: <required>

Syntax: Materials/*/block

Note: The blocks in the list must be a subset of a particular discretization scheme. We will later refer this discretization scheme as the discretization scheme.

8.2.3.2 *isMeter*

Description: Whether or not mesh is in unit of meter

Data type: Logical

Default value: False

Syntax: Materials/*/isMeter

8.2.3.3 *plus*

Description: To indicate if absorption, fission and kappa fission are to be evaluated

Data type: Logical

Default value: False

Syntax: Materials/*/plus

8.2.3.4 *AdjusterUO*

Description: User data object that performs the adjustment factor calculation on subdomain-base

Data type: string

Default value: <empty>

Syntax: Materials/*/AdjusterUO

8.2.3.5 *dumpMatAt*

Description: Dump materials on QPs for a particular residual evaluation

Data type: integer

Default value: Maximum unsigned integer

Syntax: Materials/*/dumpMatAt

Note: This parameter should only be used for debugging purpose. When this parameter is true, the cross sections will be dumped into a file named as *<output>-p<x>-t<y>.xml*, where *output* is the string given in output, *x* is the processor ID and *y* is the thread ID. If multiple processors or threads are used, multiple files will be generated.

8.2.3.6 *dumpMatOnElem*

Description: Dump materials on a list of elements for the residual evaluation specified in dumpMatAt; all will be dumped if empty

Data type: Vector of integers

Default value: <empty>

Syntax: Materials/*/dumpMatOnElem

8.2.3.7 *output*

Description: The file base used to dump debug information

Data type: String

Default value: 'mat'

Syntax: Materials/*/output

8.2.3.8 *dbgmat*

Description: To turn on the debug info of reading the file

Data type: Logical

Default value: false

Syntax: Materials/*/dbgmat

8.2.3.9 *disable_fission*

Description: To discard fission even it exists

Data type: Logical

Default value: False

Syntax: Materials/*/disable_fission

Note: This parameter provide a way to disable fission in neutronics materials, which is useful in some circumstances like evaluating the partial solutions from the fission event in a part of the solution domain.

8.2.3.10 *fromFile*

Description: To indicate the material will be read from a file

Data type: Logical

Default value: false

Syntax: Materials/*/fromFile

Note: If this parameter is false, the following parameters are activated: *sigma_t*, *sigma_s*, *nu_sigma_f*, *sigma_capture*, *sigma_nalpha*, *chi*, *neutron_speed*, *decay_constant*, *delay_fraction*, *delay_spectrum*, *diffusion_coef*, *sigma_r*, *sigma_f*, *kappa_sigma_f*, *L* and *fissile*. And *fileName* is deactivated. It is opposite when this parameter is true.

8.2.3.11 *material_id*

Description: ID of the material

Data type: integer

Default value: 1

Syntax: Materials/*/material_id

Note: This parameter is used for loading the correct material in the material file in INSTANT XML format when *fromFile* is true. It is used for dumping the material into a INSTANT XML file when *fromFile* is false and *dbgmat* is true.

8.2.3.12 *fileName*

Description: The INSTANT XML XS file name

Data type: string

Default value: <empty>

Syntax: Materials/*/fileName

Note: This parameter must be provided when *fromFile* is true. Refer to [1] for the INSTANT XML format. The number of groups in the file must be equal to *G*. If the material is fissile and have non-zero number of delayed neutron groups, the number must agree with *n_delay_groups*. If *equation_type* is transient, neutron speeds must be provided in the file.

8.2.3.13 *sigma_t*

Description: Constant total cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sigma_t

Note: This parameter must be provided and will be in action when the following conditions are met:

- *fromFile* is false;
- and Discretization Schemes of the subdomains where this material is defined on is not CFEM-Diffusion without *transport_wrapper*;

- and Discretization Schemes of the subdomains where this material is defined on is not DFEM-Diffusion without *transport_multiapp_file*.

The size of this parameter must be equal to the number of coarse groups of the discretization scheme from *G* and *group_collapsing*. *diffusion_coef* and *sigma_r* will be ignored when this parameter is in action.

8.2.3.14 *diffusion_coef*

Description: Constant diffusion coefficients

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/diffusion_coef

Note: This parameter must be provided when the following conditions are met:

- *fromFile* is false;
- and Discretization Schemes of the subdomains where this material is defined on is CFEM-Diffusion without *transport_wrapper*, or DFEM-Diffusion without *transport_multiapp_file*.

The size of this parameter must be equal to the number of coarse groups of the discretization scheme from *G* and *group_collapsing*. *sigma_t* will be ignored when this parameter is in action.

8.2.3.15 *sigma_r*

Description: Constant removal cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sigma_r

Note: This parameter must be either provided or not provided along with *diffusion_coef*.

8.2.3.16 *L*

Description: Order of scattering anisotropy

Data type: Integer

Default value: 0

Syntax: Materials/*/L

Note: This parameter is activated when *fromFile* is false. If this parameter is higher than *NA* in the discretization scheme (0 if the discretization scheme does not have), the higher order scattering will be ignored. To be a little more specific, higher order scattering than 0 will be ignored for diffusion schemes, CFEM-Diffusion and DFEM-Diffusion.

8.2.3.17 *sigma_s*

Description: Constant scattering cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sigma_s

Note: This parameter must be provided when *fromFile* is false. The size of this parameter must be equal to $G^2 \times (L+1)$. Values are ordered as $\left\{\sigma_{s,l}^{g' \to g}, g' = 1, \cdots, G; g = 1, \cdots, G; l = 0, \cdots, L\right\}$. In-group scatterings are ignored for diffusion schemes including CFEM-Diffusion and DFEM-Diffusion.

8.2.3.18 *fissile*

Description: To indicate if the material is fissile

Data type: Logical

Default value: False

Syntax: Materials/*/fissile

Note: This parameter is activated when *fromFile* is false.

8.2.3.19 *nu_sigma_f*

Description: Constant nu fission cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/nu_sigma_f

Note: This parameter must be provided when *fromFile* is false and *fissile* is true. The size of this parameter must be equal to G.

8.2.3.20 *chi*

Description: Fission spectrum

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/chi

Note: This parameter must be provided when *fromFile* is false and *fissile* is true. The size of this parameter must be equal to G. This parameter needs to contain the averaged fission spectrum of the prompt and delayed spectrum when *n_delay_groups* is non-zero. Otherwise, it contains the prompt spectrum. This convention is adapted by YAKXS.

8.2.3.21 *neutron_speed*

Description: Neutron speed

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/neutron_speed

Note: This parameter must be provided when *fromFile* is false and *equation_type* is transient. The size of this parameter must be equal to *G*. This parameter is ignored when *equation_type* is not transient.

8.2.3.22 *decay_constant*

Description: Decay constants of delayed neutron precursors

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/decay_constant

Note: This parameter must be provided when *fromFile* is false, *fissile* is true and *n_delay_groups* is non-zero. This parameter does not affect the calculation when *equation_type* is not transient.

8.2.3.23 *delay_fraction*

Description: Delayed neutron fractions

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/delay_fraction

Note: This parameter must be provided when *fromFile* is false, *fissile* is true and *n_delay_groups* is non-zero. The size of this parameter must be equal to *n_delay_groups*. This parameter is used for generating weighted fission spectrum and affects the calculation in turn when *equation_type* is not transient.

8.2.3.24 *delay_spectrum*

Description: Neutron spectrum of all delayed groups

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/delay_spectrum

Note: This parameter must be provided when *fromFile* is false, *fissile* is true and *n_delay_groups* is non-zero. The size of this parameter must be equal to *n_delay_groups*$\times$*G*. Values are ordered as $\left\{\chi_{d,g,i}, g = 1, \cdots, G; i = 1, \cdots, I\right\}$. This parameter is used for generating weighted fission spectrum and affects the calculation in turn when *equation_type* is not transient.

8.2.3.25 *sigma_capture*

Description: Constant Capture cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sigma_capture

Note: This parameter is optional when *fromFile* is false. When it is provided, its size must be equal to *G*.

8.2.3.26 *sigma_nalpha*

Description: Constant Nalpha cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sigma_nalpha

Note: This parameter is optional when *fromFile* is false. When it is provided, its size must be equal to *G*.

8.2.3.27 *sigma_f*

Description: Constant fission cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sigma_f

Note: This parameter is optional when *fromFile* is false. When it is provided, its size must be equal to *G*.

8.2.3.28 *kappa_sigma_f*

Description: Constant kappa fission cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/kappa_sigma_f

Note: This parameter is optional when *fromFile* is false. When it is provided, its size must be equal to *G*.

8.2.4 FunctionNeutronicsMaterial

This material uses *FunctionInputXS* in YAKXS for managing the constant macroscopic cross sections. It differs with ConstantNeutronicsMaterial only with the followings:

1. *isMeter* must be false in FunctionNeutronicsMaterial.
2. it does not have *sigma_capture* and *sigma_nalpha*;
3. it does not read cross sections in the file;
4. every entry in the input cross sections can be a MOOSE *function* possibly varying in space and/or time;
5. it uses a set of spatial points and a set of time to create the scattering and fission pattern.

This material is suitable for doing MMS (method of manufactured solutions) or doing some complicated spatial kinetics benchmark calculations.

8.2.4.1 *block*

Refer to *block* in ConstantNeutronicsMaterial.

8.2.4.2 *isMeter*

Refer to *isMeter* in ConstantNeutronicsMaterial.

isMeter must be false.

8.2.4.3 *plus*

Refer to *plus* in ConstantNeutronicsMaterial.

8.2.4.4 *AdjusterUO*

Refer to *AdjusterUO* in ConstantNeutronicsMaterial.

8.2.4.5 *dumpMatAt*

Refer to *dumpMatAt* in ConstantNeutronicsMaterial.

8.2.4.6 *dumpMatOnElem*

Refer to *dumpMatOnElem* in ConstantNeutronicsMaterial.

8.2.4.7 *output*

Refer to *output* in ConstantNeutronicsMaterial.

8.2.4.8 *dbgmat*

Refer to *dbgmat* in ConstantNeutronicsMaterial.

8.2.4.9 *disable_fission*

Refer to *disable_fission* in ConstantNeutronicsMaterial.

8.2.4.10 *material_id*

Refer to *material_id* in ConstantNeutronicsMaterial.

It is only used for dumping the material into a INSTANT XML file when *dbgmat* is true.

8.2.4.11 *sigma_t*

Refer to *sigma_t* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.12 *diffusion_coef*

Refer to *diffusion_coef* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.13 *sigma_r*

Refer to *sigma_r* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.14 *L*

Refer to *L* in ConstantNeutronicsMaterial.

8.2.4.15 *sigma_s*

Refer to *sigma_s* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.16 *fissile*

Refer to *fissile* in ConstantNeutronicsMaterial.

8.2.4.17 *nu_sigma_f*

Refer to *nu_sigma_f* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.18 *chi*

Refer to *chi* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.19 *neutron_speed*

Refer to *neutron_speed* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.20 *decay_constant*

Refer to *decay_constant* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.21 *delay_fraction*

Refer to *delay_fraction* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.22 *delay_spectrum*

Refer to *delay_spectrum* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.23 *sigma_f*

Refer to *sigma_f* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.24 *kappa_sigma_f*

Refer to *kappa_sigma_f* in ConstantNeutronicsMaterial.

Every element can be either a constant value or a MOOSE function.

8.2.4.25 *sample_t*

Description: Sampling time points for generating the material properties

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sample_t

Note: This parameter is used together with *sample_p*. When both are not empty, they are used to sample the cross sections and to create the fission and scattering pattern. If either one of them is empty, not sampling will be done and fission and scattering are considered dense.

8.2.4.26 *sample_p*

Description: Sampling space points for generating the material properties

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sample_p

Note: The size of this parameter has to be $3 \times n$, where n is the number of sampling points. Each point is inputted with x, y and z-coordinates. This parameter is used together with *sample_t*. When both are not empty, they are used to sample the cross sections and to create the fission and scattering pattern. If either one of them is empty, not sampling will be done and fission and scattering are considered dense.

8.2.5 *MixedNeutronicsMaterial*

This material creates a fixed set of macroscopic cross sections and use it from the interpolation and mixing of tabulated cross sections in YAKXS XML format. It does not bring in the feedback effect with the tabulated cross sections.

8.2.5.1 *block*

Refer to *block* in ConstantNeutronicsMaterial.

8.2.5.2 *isMeter*

Refer to *isMeter* in ConstantNeutronicsMaterial.

8.2.5.3 *plus*

Refer to *plus* in ConstantNeutronicsMaterial.

8.2.5.4 *AdjusterUO*

Refer to *AdjusterUO* in ConstantNeutronicsMaterial.

8.2.5.5 *dumpMatAt*

Refer to *dumpMatAt* in ConstantNeutronicsMaterial.

8.2.5.6 *dumpMatOnElem*

Refer to *dumpMatOnElem* in ConstantNeutronicsMaterial.

8.2.5.7 *output*

Refer to *output* in ConstantNeutronicsMaterial.

8.2.5.8 *dbgmat*

Refer to *dbgmat* in ConstantNeutronicsMaterial.

8.2.5.9 *disable_fission*

Refer to *disable_fission* in ConstantNeutronicsMaterial.

8.2.5.10 *material_id*

Refer to *material_id* in ConstantNeutronicsMaterial.

Note: this parameter is used to read the correct library and used to dump the generated macroscopic cross sections into INSTANT XML file when *dbgmat* is true.

8.2.5.11 *multigroup_library*

Description: File name of the multigroup library

Data type: string

Default value: <required>

Syntax: Materials/*/multigroup_library

Note: The file must be in YAKXS XML format.

8.2.5.12 *library_name*

Description: Name of the multigroup library

Data type: string

Default value: <required>

Syntax: Materials/*/library_name

Note: Refer to [1] for the detailed YAKXS format.

8.2.5.13 *grid_names*

Description: Names for all library parameters

Data type: Vector of strings

Default value: <required>

Syntax: Materials/*/grid_names

Note: The reference grid value will be used for the missing grids.

8.2.5.14 *grid*

Description: Grid names

Data type: Vector of integers

Default value: <required>

Syntax: Materials/*/grid

Note: The size of this parameter must be equal to the size of *grid_names*.

8.2.5.15 *isotopes*

Description: Name of isotopes

Data type: Vector of strings

Default value: <required>

Syntax: Materials/*/isotopes

Note: This parameter must not be empty.

8.2.5.16 *densities*

Description: Atomic densities of isotopes

Data type: Vector of reals

Default value: <required>

Syntax: Materials/*/densities

Note: The size of this parameter must be equal to the size of *isotopes*.

8.2.6 *CoupledFeedbackNeutronicsMaterial*

This material uses coupled variables to do the on-the-fly interpolation of the tabulated cross sections in YAKXS XML format. It is currently not using coupled variables for isotope densities, instead densities stay constant during the simulation.

8.2.6.1 *block*

Refer to *block* in ConstantNeutronicsMaterial.

8.2.6.2 *isMeter*

Refer to *isMeter* in ConstantNeutronicsMaterial.

8.2.6.3 *plus*

Refer to *plus* in ConstantNeutronicsMaterial.

8.2.6.4 *AdjusterUO*

Refer to *AdjusterUO* in ConstantNeutronicsMaterial.

8.2.6.5 *dumpMatAt*

Refer to *dumpMatAt* in ConstantNeutronicsMaterial.

8.2.6.6 *dumpMatOnElem*

Refer to *dumpMatOnElem* in ConstantNeutronicsMaterial.

8.2.6.7 *output*

Refer to *output* in ConstantNeutronicsMaterial.

8.2.6.8 *dbgmat*

Refer to *dbgmat* in ConstantNeutronicsMaterial.

8.2.6.9 *disable_fission*

Refer to *disable_fission* in ConstantNeutronicsMaterial.

8.2.6.10 *material_id*

Refer to *material_id* in ConstantNeutronicsMaterial.

Note: this parameter is used to get the correct library in library user object.

8.2.6.11 *MGLibObject*

Description: User object containing the desired XS library

Data type: string

Default value: <required>

Syntax: Materials/*/MGLibObject

Note: Refer to the MG library user object. We want to load the library in a separated object in order to avoid possible multiple load of the library in the memory.

8.2.6.12 *grid_names*

Refer to *grid_names* in MixedNeutronicsMaterial.

8.2.6.13 *grid_variables*

Description: Interpolation points for all library parameters

Data type: Vector of variable names

Default value: <empty>

Syntax: Materials/*/grid_variables

Note: The size of this parameter must be equal to the size of *grid_names*.

8.2.6.14 *isotopes*

Refer to *isotopes* in MixedNeutronicsMaterial.

8.2.6.15 *densities*

Refer to *densities* in MixedNeutronicsMaterial.

8.2.7 *CRoddedNeutronicsMaterial*

This material is used to model the control-rod movement within a block. A control rod is illustrated in Fig. 23. In the withdrawn direction, a control rod is divided into three parts: unrodded, rodded and driver. Control rod length is the length of the rodded material. Control rod front is at the interface between rodded material and unrodded material. Driver material are assumed to be infinite long while unrodded material will be filled in the places on the other side of control rod front.

8.2.7.1 *block*

Refer to *block* in ConstantNeutronicsMaterial.

Figure 23 Control rod sketch.

8.2.7.2 *isMeter*

Refer to *isMeter* in ConstantNeutronicsMaterial.

8.2.7.3 *plus*

Refer to *plus* in ConstantNeutronicsMaterial.

8.2.7.4 *AdjusterUO*

Refer to *AdjusterUO* in ConstantNeutronicsMaterial.

8.2.7.5 *dumpMatAt*

Refer to *dumpMatAt* in ConstantNeutronicsMaterial.

8.2.7.6 *dumpMatOnElem*

Refer to *dumpMatOnElem* in ConstantNeutronicsMaterial.

8.2.7.7 *output*

Refer to *output* in ConstantNeutronicsMaterial.

8.2.7.8 *dbgmat*

Refer to *dbgmat* in ConstantNeutronicsMaterial.

8.2.7.9 *disable_fission*

Refer to *disable_fission* in ConstantNeutronicsMaterial.

8.2.7.10 *front_position_function*

Description: The front position of the control rod as a function of time

Data type: string

Default value: <required>

Syntax: Materials/*/front_position_function

8.2.7.11 *rod_withdrawn_direction*

Description: In which direction the control rod is withdrawn

Data type: Enumeration (/x/y/z/-x/-y/-z/)

Default value: z

Syntax: Materials/*/rod_withdrawn_direction

8.2.7.12 *rod_length*

Description: The rod length

Data type: Real

Default value: Maximum real number

Syntax: Materials/*/rod_length

Note: The default value means no driver material.

8.2.7.13 *fileName*

Refer to *fileName* in ConstantNeutronicsMaterial.

This parameter is required. File must be in xml type.

8.2.7.14 *material_ids*

Description: IDs of the material in the file

Data type: Vector of integers

Default value: <required>

Syntax: Materials/*/material_ids

Note: The size of this parameter can be two or three. If it is in size two, roded and unrodded material IDs are given. If it is in size three, roded, unrodded and driver material IDs are given. It is also used for dumping the material into a INSTANT XML file when *fromFile* is false and *dbgmat* is true.

8.3 Thermal radiation materials

Thermal radiation materials are special because they are interacting with TransportSystems in the following ways:

1. Thermal radiation materials cannot be defined across two discretization schemes even they are having exactly the same scheme.
2. Thermal radiation materials cannot be used independently on discretization schemes.

8.3.1 Material properties declared by thermal radiation materials

All TR materials declare a set of material properties, which are listed in Table 23. This list could be useful for advanced users who want to utilize those properties for postprocessing. Those properties are declared on blocks on which the material are defined. The list of material properties can also be revealed during run-time by setting *show_material_props* in the *Debug* block of the input file to true. It is noted that not all of them are evaluated on element faces.

Table 23 Thermal radiation material properties.

Name	Notation	Type	When is declared
light_speed_g<g>	c_g	Real	Always
grad_light_speed_g<g>	$\vec{\nabla} c_g$	RealVector	Varying light speed[a]
refractive_index_g<g>	c_g	Real	Always
emissivity	$\epsilon_g, g = 1, \cdots, G$	std::vector<Real>	Always
emissivity_g<g>	ϵ_g	Real	Always
mean_emissivity	$\bar{\epsilon}$	Real	Always
rosseland_mean_emissivity	$\bar{\epsilon}$	Real	Always
temperature_matprop	T	Real	Always
diffusion_coefficient_g<g>	D_g	Real	Diffusion[b]
opacity_g<g>	σ_g	Real	Always
absorptivity_g<g>	α_g	Real	Always
scattering_g<p>_g<g>	$\sigma_{s,l}^{p\to g}, l = 0, \cdots, L$	std::vector<Real>	ScatteringPattern[c]
scattering_n<l>	$\sigma_{s,l}^{p\to g}, p, g = 1, \cdots, G$	CSR<Real>	NonzeroScattering[d]
phase_function_g<p>_g<g>	$p_{p\to g}(\mu)$	PhaseFunction pointer	Using phase function[e]

[a] When the material type is FunctionTRMaterial and the light speed is not a ConstantFunction;
[b] Diffusion means that *scheme* is equal to CFEM-Diffusion or DFEM-Diffusion;
[c] ScatteringPattern means that the property will be declared for the non-zero scattering entries;
[d] NonzeroScattering means that there is scattering for this material;
[e] When *use_phase_function* is set to true.

8.3.2 ConstantTRMaterial

This material provides the basic capability of doing multigroup thermal-radiation transport calculations.

8.3.2.1 *block*

Refer to *block* in ConstantNeutronicsMaterial.

8.3.2.2 *dumpMatAt*

Refer to *dumpMatAt* in ConstantNeutronicsMaterial.

8.3.2.3 *dumpMatOnElem*

Refer to *dumpMatOnElem* in ConstantNeutronicsMaterial.

8.3.2.4 *output*

Refer to *output* in ConstantNeutronicsMaterial.

8.3.2.5 *dbgmat*

Refer to *dbgmat* in ConstantNeutronicsMaterial.

8.3.2.6 *use_phase_function*

Description: True to use phase functions for the high-order scattering

Data type: Logical

Default value: <false>

Syntax: Materials/*/use_phase_function

Note: If this parameter is true, higher order scattering cross sections provided through other means will be ignored. Otherwise, *phase_function_names*, *phase_function_departuring_groups* and *phase_function_arriving_groups* should not be provided.

8.3.2.7 *phase_function_names*

Description: Phase function names

Data type: Vector of strings

Default value: <empty>

Syntax: Materials/*/phase_function_names

Note: Refer to Phase Functions for more information.

8.3.2.8 *phase_function_departuring_groups*

Description: Departuring groups of phase functions

Data type: Vector of integers

Default value: <empty>

Syntax: Materials/*/phase_function_departuring_groups

Note: The size of this parameter must be equal to the size of *phase_function_names*

8.3.2.9 *phase_function_arriving_groups*

Description: Arriving groups of phase functions

Data type: Vector of integers

Default value: <empty>

Syntax: Materials/*/phase_function_arriving_groups

Note: The size of this parameter must be equal to the size of *phase_function_names* if *phase_function_names*, *phase_function_departuring_groups* and *phase_function_arriving_groups* are lined up. The pairs which are not specified in *phase_function_departuring_groups* and *phase_function_arriving_groups* are assuming isotropic scattering.

8.3.2.10 *absorptivity*

Description: Absorptivities

Data type: Vector of reals

Default value: <required>

Syntax: Materials/*/absorptivity

Note: The size of this parameter must be equal to the size of *G*.

8.3.2.11 *emissivity*

Description: Emissivities

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/emissivity

Note: If this parameter is given, the size of this parameter must be equal to the size of *G*, otherwise emissivities are assumed to be the same as absorptivities based on LTE (local thermaldynamic equilibrium).

8.3.2.12 *light_speed*

Description: Light speeds in the medium

Data type: Vector of reals

Default value: <required>

Syntax: Materials/*/light_speed

Note: The size of this parameter must be equal to the size of *G*.

8.3.2.13 *L*

Refer to *L* in ConstantNeutronicsMaterial.

This number is also used to truncate the phase function scattering if *use_phase_function* is true.

8.3.2.14 *scattering*

Description: Scattering matrix

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/scattering

Note: The size of this parameter must be equal to G^2 when *use_phase_function* is true or must be equal to $G^2 \times (L+1)$ when *use_phase_function* is false.

8.3.2.15 *material_id*

Refer to *material_id* in ConstantNeutronicsMaterial.

8.3.3 FunctionTRMaterial

8.3.3.1 *block*

Refer to *block* in ConstantNeutronicsMaterial.

8.3.3.2 *dumpMatAt*

Refer to *dumpMatAt* in ConstantNeutronicsMaterial.

8.3.3.3 *dumpMatOnElem*

Refer to *dumpMatOnElem* in ConstantNeutronicsMaterial.

8.3.3.4 *output*

Refer to *output* in ConstantNeutronicsMaterial.

8.3.3.5 *dbgmat*

Refer to *dbgmat* in ConstantNeutronicsMaterial.

8.3.3.6 *use_phase_function*

Refer to *use_phase_function* in ConstantTRMaterial.

8.3.3.7 *phase_function_names*

Refer to *phase_function_names* in ConstantTRMaterial.

8.3.3.8 *phase_function_departuring_groups*

Refer to *phase_function_departuring_groups* in ConstantTRMaterial.

8.3.3.9 *phase_function_arriving_groups*

Refer to *phase_function_arriving_groups* in ConstantTRMaterial.

8.3.3.10 *absorptivity*

Refer to *absorptivity* in ConstantTRMaterial. Every entry can be a MOOSE function.

8.3.3.11 *emissivity*

Refer to *emissivity* in ConstantTRMaterial. Every entry can be a MOOSE function.

8.3.3.12 *light_speed*

Refer to *light_speed* in ConstantTRMaterial. Every entry can be a MOOSE function.

8.3.3.13 *L*

Refer to *L* in ConstantNeutronicsMaterial.

8.3.3.14 *scattering*

Refer to *scattering* in ConstantTRMaterial. Every entry can be a MOOSE function.

8.3.3.15 *material_id*

Refer to *material_id* in ConstantNeutronicsMaterial.

9 *Executioner*

Rattlesnake utilizes PETSc for solving the discretized transport system through the MOOSE framework. The following subsections detail the control parameters of all the executioners available in Rattlesnake. We will give an introduction on the methods used by the executioners in each subsection in order to let users have better understanding on those control parameters. Most of parameters have default values and do not need users' input for typical calculations.

9.1 *Steady*

It is a general solver provided by MOOSE to solve the finite-dimensional steady-state nonlinear or linear problem:

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}. \tag{43}$$

By default the line-search Newton in PETSc is used with the PJFNK (preconditioned Jacobian-free Newton Krylov) method. At each Newton iteration the executioner solves

$$\mathbf{J}(\mathbf{x}^{(n-1)})\delta\mathbf{x}^{(n)} = \mathbf{F}(\mathbf{x}^{(n-1)}), \tag{44}$$

where

$$\mathbf{J}(\mathbf{x}^{(n-1)}) \equiv \mathbf{F}_{\mathbf{x}}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^{(n-1)}} \tag{45}$$

is the Jacobian matrix evaluated at $\mathbf{x}^{(n-1)}$. Jacobian matrix depends on $\mathbf{x}^{(n-1)}$ for general nonlinear problems while it is constant for linear problems. The right hand side is also typically termed as the residual at $\mathbf{x}^{(n-1)}$ of Eq. (43). The Krylov methods are employed for solving the above linear equation, which essentially requires the matrix-vector product operation $\mathbf{J}(x^{(n-1)})\mathbf{y}$ possibly through a matrix-free operation

$$\frac{\mathbf{F}(\mathbf{x}^{n-1} + \epsilon\mathbf{y}) - \mathbf{F}(\mathbf{x}^{n-1})}{\epsilon}, \tag{46}$$

where the scalar value ϵ is chosen by PETSc automatically to approximate $\mathbf{J}(x^{(n-1)})\mathbf{y}$ accurately for the linear solve. Section 5.5 of PETSc user's manual on matrix-free methods details the algorithm for choosing the ϵ. The Krylov methods typically also require an approximation of the Jacobian $\mathbf{P}(x^{n-1})$ as the preconditioning matrix. Although the preconditioning matrix is seldom the exact Jacobian $\mathbf{J}$. By default the type of Krylov method in use is GMRes because it does not have assumptions on the underlining Jacobian. The initial guess of the linear solve is always set to zero, which also means that the initial linear residual is the same of the nonlinear residual. The residual norm at each linear iteration is evaluated by PETSc, for instance, during updating Hessenberg if GMRes method is used. The solution is updated after the linear solve with

$$\mathbf{x}^{n} = \mathbf{x}^{n-1} + \alpha\delta\mathbf{x}^{(n)} \tag{47}$$

where α is determined by the line-search algorithm. We can see that at each nonlinear or Newton iteration, we will need a preconditioning matrix evaluation and a residual evaluation with the updated solution. At each linear iteration, we simply need a residual evaluation and the operation of the preconditioner built from the

preconditioning matrix. In a linear problem, $\mathbf{F}(x)$ can be expressed as $\mathbf{A}\mathbf{x} - \mathbf{b}$, where matrix $\mathbf{A}$ is the Jacobian independent on $\mathbf{x}$. It is noted that the default preconditioner type depends on the number of processors and also depends on the assembled preconditioning matrix $\mathbf{P}$. Typically incomplete LU (PCILU) is the default type with one processor and block Jacobi (PCBJACOBI) is the type with multiple processors. Consequently, you will not see the same convergence with the different number of processors.

9.1.1 *l_max_its*

Description: Maximum linear iterations per nonlinear iteration

Data type: Integer

Default value: 10000

Syntax: Executioner/l_max_its

Note: This parameter is used to control the solving of Eq. (44). It essentially used to set the default value of a PETSc control parameter *-ksp_max_it*. Users are allowed to directly use the PETSc control parameter *-ksp_max_it* to overwrite this parameter.

9.1.2 *l_tol*

Description: Relative linear tolerance

Data type: Real

Default value: 10^{-5}

Syntax: Executioner/l_tol

Note: This parameter is used to control the solving of Eq. (44). It essentially used to set the default value of a PETSc control parameter *-ksp_rtol*. Users are allowed to directly use the PETSc control parameter *-ksp_rtol* to overwrite this parameter. The convergence criteria in PETSc is

$$\left\|\mathbf{F}(\mathbf{x}^{(n-1)}) - \mathbf{J}(\mathbf{x}^{(n-1)})\delta\mathbf{x}^{(n)}\right\|_2 < \max(rtol \left\|\mathbf{F}(\mathbf{x}^{(n-1)})\right\|_2, atol), \tag{48}$$

where *rtol* is this parameter and *atol* is *l_abs_step_tol*.

9.1.3 *l_abs_step_tol*

Description: Absolute linear tolerance on the residual change per linear step

Data type: real

Default value: -1

Syntax: Executioner/l_abs_step_tol

Note: This parameter is *NOT* active in this executioner. PETSc control parameter *-ksp_atol* can be used for setting this parameter. The default value of this PETSc parameter is 10^{-50}, which means *l_tol* will probably always take the effect.

9.1.4 *line_search*

Description: Specifies the line search type

Data type: Enumeration (/default/shell/none/basic/l2/bt/cp/)

Default value: default

Syntax: Executioner/line_search

Note: This parameter is used to control the evaluation of α in Eq. (47). It essentially used to set the default value of a PETSc control parameter *-snes_linesearch_type*. Users are allowed to directly use the PETSc control parameter *-snes_linesearch_type* to overwrite this parameter.

9.1.5 *solve_type*

Description: Control how solve are proceeded

Data type: Enumeration (/PJFNK/JFNK/NEWTON/FD/LINEAR/)

Default value: PJFNK

Syntax: Executioner/solve_type

Note:

- PJFNK is the default solve type. It makes the executioner perform Jacobian-free linear solves at each Newton iteration with the preconditioner built from the preconditioning matrix **P**. The preconditioning matrix is block-diagonal with each block corresponding to a variable by default. Off-diagonal Jacobian terms are ignored. Refer to ?? for more on how the preconditioning matrix is structured and assembled. It essentially activates the matrix-free Jacobian-vector products, and the preconditioning matrix, by setting the PETSc control parameter *-snes_mf_operator*.
- JFNK means there is no preconditioning during the Krylov solve. No Jacobian $\mathbf{P}(x^{n-1})$ will be assembled. It essentially activates the matrix-free Jacobian-vector products, and no preconditioning matrix, by setting the PETSc control parameter *-snes_mf*.
- LINEAR will use PETSc control parameter *-ksp_only* to set the type of SNES for solving the linear system. Note that it only works when you have an *exact* Jacobian because it is not activating matrix-free calculations. If the Jacobian is not exact, users need to set *-snes_mf* or *-snes_mf_operator* in his input explicitly through *pets_coptions*. All the control parameters *nl_abs_step_tol*, *nl_abs_tol*, *nl_max_funcs*, *nl_max_its*, *nl_rel_tol* and *nl_rel_step_tol* for the Newton iteration will be ignored.
- NEWTON means PETSc will use the Jacobian provided by Rattlesnake (typically not exact) to do the Krylov solve. If the Jacobian is not exact, Newton update in Eq. (47) will not reduce the residual effectively and typically results into an unconverged Newton iteration.
- FD means the real Jacobian will be assembled by calling residual evaluations at each Newton step by seting the PETSc control parameter *-snes_fd*. And then this Jacobian will be used during the Krylov solve. It is costly and should used only for testing purpose.

9.1.6 *nl_abs_step_tol*

Description: Nonlinear absolute step tolerance

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/solve_type

Note: This parameter is never used.

9.1.7 *nl_abs_tol*

Description: Nonlinear absolute tolerance

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/nl_abs_tol

Note: This parameter is used to control the termination of the Newton iteration. It essentially used to set the default value of a PETSc control parameter *-snes_atol*. Users are allowed to directly use the PETSc control parameter *-snes_atol* to overwrite this parameter.

9.1.8 *nl_max_funcs*

Description: Max nonlinear solver function evaluations

Data type: Integer

Default value: 10000

Syntax: Executioner/nl_max_funcs

Note: This parameter is used to control the termination of the Newton iteration. It essentially used to set the default value of a PETSc control parameter *-snes_max_funcs*. Users are allowed to directly use the PETSc control parameter *-snes_max_funcs* to overwrite this parameter.

9.1.9 *nl_max_its*

Description: Maximum nonlinear iterations

Data type: Integer

Default value: 50

Syntax: Executioner/nl_max_its

Note: This parameter is used to control the termination of the Newton iteration. It essentially used to set the default value of a PETSc control parameter *-snes_max_it*. Users are allowed to directly use the PETSc control parameter *-snes_max_it* to overwrite this parameter.

9.1.10 *nl_rel_tol*

Description: Nonlinear relative tolerance

Data type: Real

Default value: 10^{-8}

Syntax: Executioner/nl_rel_tol

Note: This parameter is used to control the termination of the Newton iteration. Be warned that if *nl_abs_tol* is not set to a meaningful value, the default value of this parameter may be overly tight and results into unconverged solve due to the machine round-off and a small initial residual. It essentially used to set the default value of a PETSc control parameter *-snes_rtol*. Users are allowed to directly use the PETSc control parameter *-snes_rtol* to overwrite this parameter.

9.1.11 *nl_rel_step_tol*

Description: Nonlinear relative step tolerance

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/nl_rel_step_tol

Note: This parameter is used to control the termination of the Newton iteration. It essentially used to set the default value of a PETSc control parameter *-snes_stol*. Users are allowed to directly use the PETSc control parameter *-snes_stol* to overwrite this parameter. Note that the default value of the PETSc control parameter is 10^{-8}.

9.1.12 *petsc_options*

Description: Used to set singleton PETSc options

Data type: Vector of strings

Default value: <empty>

Syntax: Executioner/petsc_options

9.1.13 *petsc_options_iname*

Description: Names of PETSc name/value pairs

Data type: Vector of strings

Default value: <empty>

Syntax: Executioner/petsc_options_iname

9.1.14 *petsc_options_value*

Description: Values of PETSc name/value pairs

Data type: Vector of strings

Default value: <empty>

Syntax: Executioner/petsc_options_value

Note: The parameter must correspond with *petsc_options_value*. *petsc_options*, *petsc_options_iname* and *petsc_options_value* give users the ultimate flexibility by directly interact with PETSc without providing them in the command-line. Every PETSc control parameters can be inputted. Several PETSc parameters that users could frequently use are listed in Table 24.

Table 24 Frequently used PETSc parameters.

PETSc parameter name	PETSc parameter value	Description
-pc_type	hypre	To set the preconditioner type to HYPRE
-pc_hypre_type	boomeramg	To use HYPRE BoomerAMG for preconditioning
-ksp_gmres_restart	100	To set the number of GMRes iterations for restart
More to be added when necessary...		

9.2 *Preconditioner*

Preconditioner is closely related with the executioner, so we make it as a subsection of *Executioner*. *Preconditioner* block controls how the preconditioning matrix is built and how the preconditioner is constructed. Several preconditioners are provided by Rattlesnake. The following parameter is used to choose which preconditioner is built.

9.2.1 *type*

Description: Type of the preconditioner

Data type: String

Default value: <required>

Syntax: Preconditioner/*/type

Note: Currently available preconditioners are listed in Table 25:

Table 25 Rattlesnake preconditioners.

Type	Full name
SMP	Single matrix preconditioner
FDP	Finite difference preconditioner
PBP	Physics-based preconditioner
BDP	Block-diagonal preconditioner

9.2.2 *SMP (single matrix preconditioner)*

This preconditioner is provided by MOOSE. It simply changes the default way (block-diagonal) of structuring the preconditioning matrix. Block diagonal submatrices will be always added. The preconditioner provide ways for adding more submatrices.

9.2.2.1 *pc_side*

Description: Preconditioning side

Data type: Enumeration (/left/right/symmetric/)

Default value: right

Syntax: Preconditioner/*/pc_side

Note: Directly setting PETSc parameters *-ksp_left_pc*, *-ksp_right_pc* and *-ksp_symmetric_pc* will overwrite this parameter.

9.2.2.2 *full*

Description: Set to true if you want the full set of couplings

Data type: Logical

Default value: false

Syntax: Preconditioner/*/full

Note: This parameter provides a convenient way of having the full preconditioning matrix.

9.2.2.3 *off_diag_row*

Description: The off diagonal row you want to add into the matrix

Data type: Vector of strings

Default value: <empty>

Syntax: Preconditioner/*/off_diag_row

Note: It need to be associated with an off diagonal column from the same position in *off_diag_column*. Elements of this parameter are the primal variable names.

9.2.2.4 *off_diag_column*

Description: The off diagonal column you want to add into the matrix

Data type: Vector of strings

Default value: <empty>

Syntax: Preconditioner/*/off_diag_column

Note: It need to be associated with an off diagonal row from the same position in *off_diag_row*. Elements of this parameter are the primal variable names.

9.2.2.5 *coupled_groups*

Description: Multiple space separated groups of comma separated variables.

Data type: Vector of strings

Default value: <empty>

Syntax: Preconditioner/*/coupled_groups

Note: Off-diagonal jacobians will be generated for all pairs within a group. Elements of this parameter are the primal variable names. An example of this parameter could be 'u1,u2 u3,u4' in a system with u1, u2, u3, u4 and u5 being the four primal variables, which will create a block matrix in the following pattern:

$$\left[\begin{array}{c|ccccc} & u1 & u2 & u3 & u4 & u5 \\ \hline u1 & \times & \times & & & \\ u2 & \times & \times & & & \\ u3 & & & \times & \times & \\ u4 & & & \times & \times & \\ u5 & & & & & \times \end{array}\right].$$

9.2.3 *FDP (finite difference preconditioner)*

This preconditioner is provided by MOOSE. It simply changes the default way (through MOOSE Jacobian evaluation, which may not be exact) of assembling the preconditioning matrix. Block diagonal submatrices will be always assembled. Other submatrices can be assembled through parameters *full*, *off_diag_row*, *off_diag_column* and *coupled_groups* as in SMP. It is noted that this preconditioner only works with one single processor. It is costly and should used only for testing purpose. One difference of this preconditioner with respect to FD solve type in *solve_type* is that it provides more controls on how the preconditioning matrix are structured.

9.2.3.1 *pc_side*

Refer to *pc_side* in SMP.

9.2.3.2 *full*

Refer to *full* in SMP.

9.2.3.3 *off_diag_row*

Refer to *off_diag_row* in SMP.

9.2.3.4 *off_diag_column*

Refer to *off_diag_column* in SMP.

9.2.3.5 *coupled_groups*

Refer to *coupled_groups* in SMP.

9.2.4 *PBP (physics-based preconditioner)*

This preconditioner is provided by MOOSE. It builds a shell, i.e. user-defined, preconditioner for PETSc. It builds systems for all primal variables, which is used for applying the local preconditioner of the linear system form by the diagonal block Jacobian of the variable and the right hand side formed by the following:

$$\mathbf{b}_i = \mathbf{x}_i - \sum_{j=1}^{N} c_{i,j}\mathbf{P}_{i,j}\mathbf{y}_j, \tag{49}$$

where $\mathbf{x}$ is the global vector passed into the preconditioner and $\mathbf{y}$ is the global vector outputted by the preconditioner. $\mathbf{P}$ is the preconditioning matrix. c specify the pattern of the preconditioning matrix determined by *full*, *off_diag_row* and *off_diag_column*. Its elements are either 0 or 1. Subscript i and j are used for indexing the variable. When this preconditioner is used *solve_type* is no longer active, the solve type is set to JFNK by PBP.

9.2.4.1 *pc_side*

Refer to *pc_side* in SMP.

9.2.4.2 *full*

Refer to *full* in SMP.

9.2.4.3 *off_diag_row*

Refer to *off_diag_row* in SMP.

9.2.4.4 *off_diag_column*

Refer to *off_diag_column* in SMP.

9.2.4.5 *preconditioner*

Description: To specify the preconditioning type of all variables

Data type: Vector of strings

Default value: <required>

Syntax: Preconditioner/*/preconditioner

Note: The current supported types are: IDENTITY, JACOBI, BLOCK_JACOBI, SOR, EISENSTAT, ASM, CHOLESKY, ICC, ILU, LU, SHELL and AMG. IDENTITY means no preconditioning and AMG is using HYPRE BoomerAMG. You can find mappings of the other types to PETSc preconditioner types. The size of the parameter must be equal to the number of primal variables. The type is assigned to variables based on their internal ordering.

9.2.4.6 *solve_order*

Description: The order the block rows will be solved in

Data type: Vector of strings

Default value: <required>

Syntax: Preconditioner/*/solve_order

Note: The elements of this parameter are the name of primal variables, which stand for solving that variable's block row. It is noted that a variable may appear more than once (to create cylces if you like).

When setup is called for this preconditioner?

9.2.5 *BDPreconditioner (block-diagonal preconditioner)*

This preconditioner is provided by Rattlesnake. It builds a shell, i.e. user-defined, preconditioner for PETSc. It builds a system for all primal variables, which is used for applying the local preconditioner of the linear system form by the diagonal block Jacobian of the variable and the right hand side directly from the global vector passed into the preconditioner. Because of this, all primal variables should be defined on the same mesh and using the same shape functions in the same order. It simply loop through all variables once and apply the local preconditioners when preconditioning. The preconditioning type is hardcoded as HYPRE BoomerAMG. In this regard, it is a special case of PBP. However it provide few convenient parameters as the following. When this preconditioner is used *solve_type* is no longer active, the solve type is set to JFNK.

9.2.5.1 *pc_side*

Refer to *pc_side* in SMP.

9.2.5.2 *rel_tol*

Description: Relative tolerance with respect to the initial residual for solving all variables

Data type: Real

Default value: 10^{-8}

Syntax: Preconditioner/*/rel_tol

Note: More than one multigrid cycles may be required to obtain the convergence.

9.2.5.3 *max_iter*

Description: Maximum number of AMG cycles

Data type: Integer

Default value: 100

Syntax: Preconditioner/*/rel_tol

9.2.5.4 *verbose*

Description: True to output progress on screen

Data type: Integer

Default value: 1

Syntax: Preconditioner/*/verbose

Note: The bigger this number is the more screen print-out will be.

9.2.5.5 *pre_assemble*

Description: Whether or not pre-assemble the Jacobian before starting the linear solve

Data type: Integer

Default value: 0

Syntax: Preconditioner/*/pre_assemble

Note: If this parameter is equal to 1, the Jacobian will be assembled at each linear solve, which is extremely slow although memory usage is significantly reduced.

9.2.5.6 *fix_jacobian*

Description: True to assemble matrices once

Data type: Logical

Default value: False

Syntax: Preconditioner/*/fix_jacobian

Note: This parameter will be ignored when *pre_assemble* is not equal to 1. Users should set this parameter to true only when the Jacobian is constant.

9.2.6 *SNSweepPreconditioner*

This preconditioner is provided by Rattlesnake. It only works with DFEM-SN. It informs transport system to set up a user object for mesh sweeping. It then uses the user object to perform the transport sweep as the preconditioner of the solve. The advantage of this preconditioner is that it is completely matrix-free. It does not requires the building of the Jacobian. Hence, the memory usage can be significantly reduced. When this preconditioner is used *solve_type* is no longer active, the solve type is set to JFNK. It only has one parameter:

9.2.6.1 *pc_side*

Refer to *pc_side* in SMP.

9.3 *Transient*

It is a general solver provided by MOOSE to solve the finite-dimensional transient nonlinear or linear problem:

$$\frac{d\mathbf{T}(\mathbf{x}(t),t)}{dt} = -\mathbf{R}(\mathbf{x}(t),t), \tag{50}$$

with the initial condition

$$\mathbf{x}(t_0) = \mathbf{x}_0, \tag{51}$$

This executioner uses method of lines to treat the time independent variable. Using implicit Euler scheme as an example, at every time step, we are solving

$$\frac{\mathbf{T}(\mathbf{x},t) - \mathbf{T}(\mathbf{x}^{old},t^{old})}{\Delta t} = -\mathbf{R}(\mathbf{x},t), \tag{52}$$

where $\Delta t = t - t^{old}$. If we define $\mathbf{F}(\mathbf{x})$ as

$$\mathbf{F}(\mathbf{x}) \equiv \mathbf{R}(\mathbf{x},t) + \frac{1}{\Delta t}\mathbf{T}(\mathbf{x},t) - \frac{\mathbf{T}(\mathbf{x}^{old},t^{old})}{\Delta t}, \tag{53}$$

where t, t^{old} and $\mathbf{x}^{old}$ are given, at every time step we are solving a finite-dimensional steady-state nonlinear or linear problem in Eq. (43). We can use our machinery implemented in the Steady executioner to solve it at every time step and then match forward. Because of this, all parameters of Steady are also valid parameters of Transient. Implicit Euler is just one of the integration scheme, the full list of integration schemes are given in *scheme*. It is seldom to step to the end time with the same time steps. MOOSE provides several time steps for adjust the time steps for reducing the number of time steps required for achieving a certain accuracy.

9.3.1 Steady parameters

All parameters of Steady are valid parameters. Refer to Steady for more information.

9.3.2 *scheme*

Description: Time integration scheme used

Data type: Enumeration (/implicit-euler/explicit-euler/crank-nicolson/bdf2/rk-2/dirk/)

Default value: implicit-euler

Syntax: Executioner/scheme

9.3.3 *start_time*

Description: The start time of the simulation

Data type: Real

Default value: 0

Syntax: Executioner/start_time

9.3.4 *end_time*

Description: The end time of the simulation

Data type: Real

Default value: 10^{30}

Syntax: Executioner/end_time

9.3.5 *dt*

Description: The timestep size between solves

Data type: Real

Default value: 1

Syntax: Executioner/dt

9.3.6 *dtmin*

Description: The minimum timestep size in an adaptive run

Data type: Real

Default value: 2×10^{-14}

Syntax: Executioner/dtmin

9.3.7 *dtmax*

Description: The maximum timestep size in an adaptive run

Data type: Real

Default value: 10^{30}

Syntax: Executioner/dtmax

9.3.8 *num_steps*

Description: The number of timesteps in a transient run

Data type: Integer

Default value: The maximum unsigned integer

Syntax: Executioner/num_steps

9.3.9 *abort_on_solve_fail*

Description: Abort if solve not converged rather than cut timestep

Data type: Logical

Default value: false

Syntax: Executioner/abort_on_solve_fail

9.3.10 *timestep_tolerance*

Description: The tolerance setting for final timestep size and sync times

Data type: Real

Default value: 2×10^{-14}

Syntax: Executioner/timestep_tolerance

9.3.11 *reset_dt*

Description: Use when restarting a calculation to force a change in dt

Data type: Logical

Default value: false

Syntax: Executioner/reset_dt

9.3.12 *n_startup_steps*

Description: The number of timesteps during startup

Data type: Logical

Default value: false

Syntax: Executioner/n_startup_steps

9.3.13 *trans_ss_check*

Description: Whether or not to check for steady state conditions

Data type: Logical

Default value: false

Syntax: Executioner/trans_ss_check

9.3.14 *ss_check_tol*

Description: Whenever the relative changes in the solution vector by less than this the solution will be considered to be at steady state

Data type: Real

Default value: 10^{-8}

Syntax: Executioner/ss_check_tol

9.3.15 *picard_max_its*

Description: Number of times each timestep will be solved.

Data type: Integer

Default value: 1

Syntax: Executioner/picard_max_its

Note: Default value one means no Picard iterations on time steps. This parameter is mainly used when wanting to do Picard iterations with MultiApps that are set to execute_on timestep_end or timestep_begin.

9.3.16 *picard_rel_tol*

Description: The relative nonlinear residual drop to shoot for during Picard iterations

Data type: Real

Default value: 10^{-8}

Syntax: Executioner/picard_rel_tol

Note: This check is performed based on the Master application's nonlinear residual. The nonlinear residual is re-evaluated at the end of each Picard iteration where all variable transfers has been accomplished or states are updated.

9.3.17 *picard_abs_tol*

Description: The absolute nonlinear residual to shoot for during Picard iterations

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/picard_abs_tol

Note: This check is performed based on the Master application's nonlinear residual. By default, this tolerance will not take effect.

9.3.18 *use_multiapp_dt*

Description: To determine how MultiApps affect the time steps

Data type: Logical

Default value: False

Syntax: Executioner/use_multiapp_dt

Note: If true then the dt for the simulation will be chosen by the MultiApps. If false (the default) then the minimum over the master dt and the MultiApps is used.

9.3.19 *TimeIntegrator*

(advanced topic, more to be added.)

MOOSE provides the following implicit TimeIntegrators:

- Backward Euler (default)
- BDF2
- Crank-Nicolson
- Implicit-Euler
- Implicit Midpoint (implemented as two-stage RK method)
- Diagonally-Implicit Runge-Kutta (DIRK) methods of order 2 and 3.

And these explicit TimeIntegrators:

- Explicit-Euler
- Explicit Midpoint
- Heun's Method
- Ralston's Method

Each one of these supports adaptive time stepping in *TimeStepper*.

9.3.20 *TimeStepper*

(advanced topic, from MOOSE wiki, more to be added.)

TimeSteppers are lightweight objects used for computing suggested time steps for transient executioners. MOOSE has several built-in TimeSteppers

- ConstantDT
- SolutionTimeAdaptiveDT
- IterationAdaptiveDT
- FunctionDT
- PostprocessorDT

- DT2:

  The steps of doing DT2:

  1. Take one time step of size Δt to get $\hat{u}_{n+1}$ from u_n;
  2. Take two time steps of size $\frac{\Delta t}{2}$ to get u_{n+1} from u_n;
  3. Calculate local relative time discretization error estimate: $\hat{e}_n \equiv \frac{\|u_{n+1}-\hat{u}_{n+1}\|_2}{\max(\|u_{n+1}\|_2,\|\hat{u}_{n+1}\|_2)}$;
  4. Obtain global relative time discretization error estimate $e_n \equiv \frac{\hat{e}_n}{\Delta t}$;
  5. Adaptivity is based on target error tolerance e_{TOL} and a maximum acceptable error tolerance e_{MAX}. If $e_n < e_{MAX}$, continue with a new time step size $\Delta t_{n+1} \equiv \Delta t_n \cdot \left(\frac{e_{TOL}}{e_n}\right)^{1/p}$ where p is the global convergence rate of the time stepping scheme. If $e_n \geq e_{MAX}$, or if the solver fails, shrink Δt;
  6. Parameters e_{TOL} and e_{MAX} can be specified in the input file as e_tol and e_max (in the Executioner block).

9.3.21 *Quadrature*

(advanced topic, to be added.)

9.4 *InversePowerMethod*

The operator form of the steady-state eigenvalue problem of the neutron transport equation is

$$\mathbf{A}(\mathbf{x}) = \frac{1}{k}\mathbf{B}(\mathbf{x}), \tag{54}$$

where $\mathbf{A}$ represents all the events neutrons undergo in a background media, including collision, streaming, scattering, absorption, and etc. except fission. $\mathbf{B}$ represents the fission event. $\mathbf{A}$, $\mathbf{B}$ can be linear or nonlinear. $\mathbf{x}$ is the solution. It could be the angular fluxes of some selected directions with SN calculations or the angular flux moments up to a certain order with PN calculations. For the details of the equation, readers are referred to textbooks such as "Nuclear Reactor Analysis by James J. Duderstadt and Louis J. Hamilton". We adjust the strength of fission by varying a scalar coefficient k so that the system can have self-sustained neutron populations without any external neutron sources. It is noted that we can have the self-sustained neutron populations in reality because the averaged neutrons emitted from fission after forming a compound nucleus is more than one, actually about 2.6. To this end, neutronics often cares only the minimum adjustment of the system or the maximum k, which has the physical meaning as the multiplication factor. When $k > 1$, the system is super-critical. Nuclear bomb is an example super-critical system. When $k < 1$, the system is sub-critical. When $k = 1$, the system is critical. It is also noted that k is always positive. $1/k$ is indeed the minimum eigenvalue of the system and always positive determined by the physics. Typically we are only interested in the absolute minimum eigenvalue and the corresponding eigenvector of the system, referred as the fundamental mode later.

The algorithm of doing inverse power iteration is given in Algorithm 1. We notice immediately that $\frac{|\mathbf{B}(\mathbf{x})|}{k}$ remains constant during the iteration, so if we make $\frac{|\mathbf{B}(\mathbf{x}^{(0)})|}{k^{(0)}}$ equal to one, the algorithm can be simplified a little into Algorithm 2.

Also in this simplified algorithm, the solution is automatically normalized making $|\mathbf{B}(\mathbf{x})| = k$. We can do postprocessing to normalize the solution so that $|\mathbf{x}| = c$, where $|.|$ can be any norm and c is a scalar constant. If the minimum eigenvalue and the second smallest eigenvalue are close, i.e. the dominant ratio is about to one, the inverse power iteration converges very slowly. In such a case, we can apply accelerations, such as Chebyshev acceleration, based on the on-the-fly estimation of the dominant ratio. The details of Chebyshev

Algorithm 1 Inverse power iteration

1: Initialization:

$$k^{(0)} = k_0$$
$$\mathbf{x}^{(0)} = \mathbf{x}_0.$$

2: Update $\mathbf{x}$ by performing a steady-state solve ‡ and update k:

$$\mathbf{A}(\mathbf{x}^{(n)}) = \frac{1}{k^{(n-1)}}\mathbf{B}(\mathbf{x}^{(n-1)}) \tag{55}$$
$$k^{(n)} = k^{(n-1)}\frac{|\mathbf{B}(\mathbf{x}^{(n)})|}{|\mathbf{B}(\mathbf{x}^{(n-1)})|}$$

3: Check the convergence

$$\frac{|k^{(n)} - k^{(n-1)}|}{|k^{(n)}|} < \epsilon_k$$

and

$$\frac{|\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}|}{|\mathbf{x}^{(n)}|} < \epsilon_x. \tag{56}$$

When either of them is not true, return Step 2, otherwise exit.

‡ Note that we typically do not need to have a full solve. The currently code will only perform one Newton iteration.

Algorithm 2 Inverse power iteration

1: Initialization:

$$k^{(0)} = k_0$$
$$\mathbf{x}^{(0)} = k_0\frac{\mathbf{x}_0}{|\mathbf{B}(\mathbf{x}_0)|}.$$

2: Update $\mathbf{x}$ by performing a steady-state solve and update k:

$$\mathbf{A}(\mathbf{x}^{(n)}) = \frac{1}{k^{(n-1)}}\mathbf{B}(\mathbf{x}^{(n-1)})$$
$$k^{(n)} = |\mathbf{B}(\mathbf{x}^{(n)})|$$

3: Check the convergence

$$\frac{|k^{(n)} - k^{(n-1)}|}{|k^{(n)}|} < \epsilon_k$$

and

$$\frac{|\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}|}{|\mathbf{x}^{(n)}|} < \epsilon_x.$$

When either of them is not true, return Step 2, otherwise exit.

acceleration can be found in paper "Optimized Iteration Strategies and Data Management Considerations for Fast Reactor Finite Difference Diffusion Theory Codes" by Ferguson, D. R. and Derstine, K. L. on Nuclear Science and Engineering, Vol.64, 1977. The inverse power method is appealing because we can use PJFNK (preconditioned Jacobian-free Newton Krylov) for solving for the updated solution and we do not have to exactly assemble matrix **A** for the preconditioning purpose.

9.4.1 *l_max_its*

Refer to *l_max_its* in Steady.

9.4.2 *pfactor*

Description: Relative linear tolerance

Data type: Real

Default value: 0.01

Syntax: Executioner/pfactor

Note: This parameter is used to control the solving of Eq. (55). It essentially used to set the default value of a PETSc control parameter *-ksp_rtol*. Users are allowed to directly use the PETSc control parameter *-ksp_rtol* to overwrite this parameter. The convergence criteria in PETSc is

$$\left\|\mathbf{F}(\mathbf{x}^{(n-1)}) - \mathbf{J}(\mathbf{x}^{(n-1)})\delta\mathbf{x}^{(n)}\right\|_2 < \max(rtol \left\|\mathbf{F}(\mathbf{x}^{(n-1)})\right\|_2, atol), \tag{57}$$

where *rtol* is this parameter and *atol* is *l_abs_step_tol*.

9.4.3 *l_abs_step_tol*

Refer to *l_abs_step_tol* in Steady.

9.4.4 *line_search*

Refer to *line_search* in Steady.

9.4.5 *solve_type*

Refer to *solve_type* in Steady.

9.4.6 *petsc_options*

Refer to *petsc_options* in Steady.

9.4.7 *petsc_options_iname*

Refer to *petsc_options_iname* in Steady.

9.4.8 *petsc_options_value*

Refer to *petsc_options_value* in Steady.

9.4.9 *auto_initialization*

Description: True to ask the solver to set a nonzero initial solution $\mathbf{x}_0$

Data type: Logical

Default value: true

Syntax: Executioner/auto_initialization

9.4.10 *eig_check_tol*

Description: Eigenvalue convergence tolerance ϵ_k

Data type: Real

Default value: 10^{-6}

Syntax: Executioner/eig_check_tol

9.4.11 *Chebyshev_acceleration_on*

Description: If Chebyshev acceleration is turned on

Data type: Logical

Default value: true

Syntax: Executioner/Chebyshev_acceleration_on

9.4.12 *k0*

Description: Initial guess of the eigenvalue k_0

Data type: Real

Default value: 1

Syntax: Executioner/k0

9.4.13 *max_power_iterations*

Description: The maximum number of power iterations

Data type: Integer

Default value: 300

Syntax: Executioner/max_power_iterations

9.4.14 *min_power_iterations*

Description: The minimum number of power iterations

Data type: Integer

Default value: 1

Syntax: Executioner/min_power_iterations

9.4.15 *xdiff*

Description: Name of the postprocessor evaluating $|\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}|$

Data type: String

Default value: <empty>

Syntax: Executioner/xdiff

Note: If this processor is not given, no convergence check with Eq. (56) will be performed.

9.4.16 *sol_check_tol*

Description: Relative tolerance on the solution ϵ_x

Data type: Real

Default value: Maximum real value

Syntax: Executioner/sol_check_tol

Note: This parameter is activated only when *xdiff* is provided.

9.4.17 *time*

Description: To set the system time with this value

Data type: Real

Default value: 0

Syntax: Executioner/time

Note: This parameter is useful when the default system time is not equal to zero.

9.4.18 *normalization*

Description: Name of the postprocessor evaluating $|\mathbf{x}|$

Data type: String

Default value: <empty>

Syntax: Executioner/normalization

Note: If this processor is not given, no final normalization after the power iteration will be performed.

9.4.19 *normal_factor*

Description: To normalize $\mathbf{x}$ to make $|\mathbf{x}|$ equal to this factor

Data type: Real

Default value: <empty>

Syntax: Executioner/normal_factor

Note: This parameter is required when *normalization* is provided.

9.4.20 *output_before_normalization*

Description: True to output a step before normalization

Data type: Logical

Default value: true

Syntax: Executioner/output_before_normalization

9.4.21 *output_on_final*

Description: True to disable all intemediate outputs

Data type: Logical

Default value: false

Syntax: Executioner/output_on_final

9.5 *NonlinearEigen*

Continuing the introduction part of InversePowerMethod, we can see the eigenvalue problem can be viewed as a nonlinear problem

$$\mathbf{A}(\mathbf{x}) = \frac{1}{k}\mathbf{B}(\mathbf{x}), \tag{58}$$

$$k = |\mathbf{B}(\mathbf{x})|. \tag{59}$$

So we can use the Newton method to solve it. However, this nonlinear problem has multiple number of solution other than the fundamental mode we are interested in. So to make the solving converge to the fundamental mode, we need to have a fairly close initial guess to the fundamental mode. This can be achieved with several free power iterations before the Newton iteration. We do not have to have k as part of the solution vector. Instead we can apply the elimination technique see "Acceleration of k-Eigenvalue/Criticality Calculations Using the Jacobian-Free Newton-Krylov Method" by D. A. Knoll and H. Park and C. Newman on Nuclear Science and Engineering, Vol. 167, 2011, i.e. solve the following

$$\mathbf{A}(\mathbf{x}) = \frac{1}{|\mathbf{B}(\mathbf{x})|}\mathbf{B}(\mathbf{x}). \tag{60}$$

Again we can use PJFNK (preconditioned Jacobian-free Newton Krylov) method to solve this nonlinear problem.

9.5.1 *l_max_its*

Refer to *l_max_its* in Steady.

9.5.2 *pfactor*

Refer to *pfactor* in InversePowerMethod.

9.5.3 *l_abs_step_tol*

Refer to *l_abs_step_tol* in Steady.

9.5.4 *line_search*

Refer to *line_search* in Steady.

9.5.5 *solve_type*

Refer to *solve_type* in Steady.

9.5.6 *nl_abs_step_tol*

Refer to *nl_abs_step_tol* in Steady.

9.5.7 *source_abs_tol*

Description: Nonlinear absolute tolerance

Data type: Real

Default value: 10^{-6}

Syntax: Executioner/source_abs_tol

Note: This parameter is used to control the termination of the Newton iteration. It essentially used to set the default value of a PETSc control parameter *-snes_atol*. Users are allowed to directly use the PETSc control parameter *-snes_atol* to overwrite this parameter.

9.5.8 *nl_max_funcs*

Refer to *nl_max_funcs* in Steady.

9.5.9 *nl_max_its*

Refer to *nl_max_its* in Steady.

9.5.10 *source_rel_tol*

Description: Nonlinear relative tolerance

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/source_rel_tol

Note: This parameter is used to control the termination of the Newton iteration. It essentially used to set the default value of a PETSc control parameter *-snes_rtol*. Users are allowed to directly use the PETSc control parameter *-snes_rtol* to overwrite this parameter.

9.5.11 *nl_rel_step_tol*

Refer to *nl_rel_step_tol* in Steady.

9.5.12 *petsc_options*

Refer to *petsc_options* in Steady.

9.5.13 *petsc_options_iname*

Refer to *petsc_options_iname* in Steady.

9.5.14 *petsc_options_value*

Refer to *petsc_options_value* in Steady.

9.5.15 *auto_initialization*

Refer to *auto_initialization* in InversePowerMethod.

9.5.16 *k0*

Refer to *k0* in InversePowerMethod.

9.5.17 *free_power_iterations*

Description: The number of free power iterations before the Newton iteration

Data type: Integer

Default value: 4

Syntax: Executioner/free_power_iterations

9.5.18 *time*

Refer to *time* in InversePowerMethod.

9.5.19 *normalization*

Refer to *normalization* in InversePowerMethod.

9.5.20 *normal_factor*

Refer to *normal_factor* in InversePowerMethod.

9.5.21 *output_before_normalization*

Refer to *output_before_normalization* in InversePowerMethod.

9.5.22 *output_on_final*

Refer to *output_on_final* in InversePowerMethod.

9.5.23 *output_after_power_iterations*

Description: True to output a step after free power iterations if there is any

Data type: Logical

Default value: true

Syntax: Executioner/output_after_power_iterations

9.6 *CriticalitySearch*

Occasionally, we'd like to determine a parameter p, such as the soluable boron concentration, control rod band position, and etc., which is affecting the results of $\mathbf{A}(p, \mathbf{x})$ and/or $\mathbf{B}(p, \mathbf{x})$, so that the eigenvalue k of Eq. (54) can be a specific value. Because when $k = 1$, the system is called critical and typically 1 is the targeted eigenvalue, we name this kind of calculation as the criticality search. We can think the eigenvalue $k(p)$ is a function of p. If the parameter p is close to the target parameter, we can do Newton search for $k(p) - k_{target} = 0$ with

$$p^{(n+1)} = p^{(n)} - \left(\left. \frac{dk}{dp} \right|_{p=p^{(n)}} \right)^{-1} (k(p^{(n)}) - k_{target}), \tag{61}$$

with $\left. \frac{dk}{dp} \right|_{p=p^{(n)}}$ is approximated by

$$\left. \frac{dk}{dp} \right|_{p=p^{(n)}} = \frac{k(p^{(n)}(1+\epsilon)) - k(p^{(n)})}{\epsilon p^{(n)}}, \tag{62}$$

where ϵ is a small scalar value used to perturb the parameter for evaluating the k-derivative with respect to p. So for every Newton step n, we perform two eigenvalue calculations one with the current parameter $p^{(n)}$ another with the perturbed parameter $p^{(n)}(1+\epsilon)$ with PJFNK with the free power iterations. It is noted that the perturbation with ϵ should be strong enough so that the numerical issue on evaluating $k(p^{(n)}(1+\epsilon)) - k(p^{(n)}$ does not cause the instability of the search. The smaller $\frac{dk}{dp}$ is, the larger ϵ should be. On the other hand, if large ϵ is used, we end up with secant method, which converges slower than Newton's method. Newton's method requires a fairly close initial guess of p when the $k(p)$ is complicated. In such a case, we can perform few bisection evaluations to obtain the initial guess.

9.6.1 *adjustable_function*

Description: The adjustable function coupled in the system having the adjustable parameter p

Data type: String

Default value: <required>

Syntax: Executioner/adjustable_function

Note: An adjustable function is a MOOSE function which has a parameter affect its evaluation. It provides a method for CriticalitySearch to adjust the parameter.

9.6.2 *bisection*

Description: Number of bisections before starting Newton

Data type: Integer

Default value: 0

Syntax: Executioner/bisection

Note: It must be greater than or equal to zero.

9.6.3 *lowerbound*

Description: The minimum value of the parameters

Data type: Real

Default value: <required>

Syntax: Executioner/lowerbound

Note: The estimated lower bound of the parameter. It is used for starting bisection search if *bisection* is greater than zero. It is also used to check if the Newton update is out of bound.

9.6.4 *upperbound*

Description: The maximum value of the parameters

Data type: Real

Default value: <required>

Syntax: Executioner/upperbound

Note: The estimated upper bound of the parameter. It is used for starting bisection search if *bisection* is greater than zero. It is also used to check if the Newton update is out of bound.

9.6.5 *rel_tol*

Description: Relative tolerance on the adjustable parameter

Data type: Real

Default value: 10^{-4}

Syntax: Executioner/rel_tol

9.6.6 *target_eigenvalue*

Description: Targetted eigenvalue k_{target}

Data type: Real

Default value: 1

Syntax: Executioner/target_eigenvalue

9.6.7 *bisection_pi*

Description: Number of free power iterations for bisection

Data type: Integer

Default value: 2

Syntax: Executioner/bisection_pi

Note: This parameter is activated only if *bisection* is greater than zero.

9.6.8 *newton_pi*

Description: Number of free power iterations for Newton search

Data type: Integer

Default value: 0

Syntax: Executioner/newton_pi

9.6.9 *pre_pi*

Description: Number of starting power iterations

Data type: Integer

Default value: 4

Syntax: Executioner/pre_pi

9.6.10 *echo*

Description: Used for controlling the screen printout

Data type: Integer

Default value: 1

Syntax: Executioner/echo

Note: The bigger this number is the more screen print-out will be seen.

9.6.11 *perturbation*

Description: Perturbation strength ϵ on the parameter p for evaluating dk/dp

Data type: Real

Default value: 10^{-6}

Syntax: Executioner/perturbation

9.6.12 *l_max_its*

Refer to *l_max_its* in Steady.

9.6.13 *pfactor*

Refer to *pfactor* in InversePowerMethod.

9.6.14 *l_abs_step_tol*

Refer to *l_abs_step_tol* in Steady.

9.6.15 *line_search*

Refer to *line_search* in Steady.

9.6.16 *solve_type*

Refer to *solve_type* in Steady.

9.6.17 *nl_abs_step_tol*

Refer to *nl_abs_step_tol* in Steady.

9.6.18 *source_abs_tol*

Refer to *source_abs_tol* in NonlinearEigen.

9.6.19 *nl_max_funcs*

Refer to *nl_max_funcs* in Steady.

9.6.20 *nl_max_its*

Refer to *nl_max_its* in Steady.

9.6.21 *nl_rel_step_tol*

Refer to *nl_rel_step_tol* in Steady.

9.6.22 *petsc_options*

Refer to *petsc_options* in Steady.

9.6.23 *petsc_options_iname*

Refer to *petsc_options_iname* in Steady.

9.6.24 *petsc_options_value*

Refer to *petsc_options_value* in Steady.

9.6.25 *auto_initialization*

Refer to *auto_initialization* in InversePowerMethod.

9.6.26 *k0*

Refer to *k0* in InversePowerMethod.

9.6.27 *time*

Refer to *time* in InversePowerMethod.

9.6.28 *normalization*

Refer to *normalization* in InversePowerMethod.

9.6.29 *normal_factor*

Refer to *normal_factor* in InversePowerMethod.

9.7 *PicardSteady*

This executioner essentially wraps the Steady executioner. It is still solving the general nonlinear problem

$$\mathbf{F}_{\mathbf{x}}(\mathbf{x}) = \mathbf{0}. \tag{63}$$

But the operator $F_{\mathbf{x}}$ is made to depend on $\mathbf{x}$ explicitly though auxiliary variables, material properties, post-processors, multiapps and etc.. There are cases that decoupling this dependency from the non-linearity of the

operator itself can make the nonlinear solves more efficient or can bypass difficulties in the nonlinear solves. We essentially lag the operator one Picard iteration behind by doing

$$\mathbf{F}_{\mathbf{x}^{(n-1)}}(\mathbf{x}^{(n)}) = \mathbf{0}, \tag{64}$$

where n is the Picard iteration index. At every Picard iteration, the executioner will perform the steps specified in Algorithm 3. The difference between one Picard iteration and the Steady executioner is that the later does not have Step 1, 3, 6, 7, 8 and 9 but has the mesh adaptation loop.

Algorithm 3 Picard iteration

1: Run all *MultiApps* and *Transfers* on *timestep_begin*. If any *MultiApp* fails in converging, Picard iteration will be aborted with an error message.
2: Evaluate all auxiliary variables, user objects (including postprocessors) on *timestep_begin*.
3: If Picard iteration is performed and the convergence is checked based on the residual, the residual L2 norm will be evaluated based on the current state.
4: Do a nonlinear solve essentially resolve all objects on *linear* for setting up the equation. If the solve fails in converging, Picard iteration will be aborted with an error message.
5: Evaluate all auxiliary variables, user objects (including postprocessors) on *timestep_end*.
6: Run all *MultiApps* and *Transfers* on *timestep_end*. If any *MultiApp* fails in converging, Picard iteration will be aborted with an error message.
7: Evaluate all auxiliary variables, user objects (including postprocessors) on *custom*.
8: Run all *MultiApps* and *Transfers* on *custom*. If any *MultiApp* fails in converging, Picard iteration will be aborted with an error message.
9: If Picard iteration is performed and the convergence is checked based on the residual, the residual L2 norm will be evaluated based on the current state.

The initial condition for $\mathbf{x}$ is problem dependent. The convergence can be checked with the norm of the non-lagged residual

$$\left\| \mathbf{F}_{\mathbf{x}^{(n)}}(\mathbf{x}^{(n)}) \right\| < \epsilon. \tag{65}$$

One more residual evaluation is needed for this check. The convergence can also be checked based on any postprocessors which depend on $\mathbf{x}$ either directly or indirectly. As an example, we demonstrate how this executioner are used to drive NDA (nonlinear diffusion acceleration) calculations in Sec. 3.8.

9.7.1 Steady parameters

All parameters of Steady are valid parameters. Refer to Steady for more information.

9.7.2 *picard_max_its*

Description: Maximum number of Picard iterations.

Data type: Integer

Default value: 1

Syntax: Executioner/picard_max_its

Note: Default value one means no Picard iterations.

9.7.3 *ignore_picard_tol*

Description: True to avoid extra residual norm evaluations and skip the check with *picard_rel_tol* and *picard_abs_tol*.

Date type: Logical

Default value: true

Syntax: Executioner/ignore_picard_tol

9.7.4 *picard_rel_tol*

Description: The relative nonlinear residual drop to shoot for during Picard iterations

Data type: Real

Default value: 10^{-8}

Syntax: Executioner/picard_rel_tol

Note: This check is performed based on the Master application's nonlinear residual. The nonlinear residual is re-evaluated at the end of each Picard iteration where all variable transfers has been accomplished or states are updated.

9.7.5 *picard_abs_tol*

Description: The absolute nonlinear residual to shoot for during Picard iterations

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/picard_abs_tol

Note: This check is performed based on the Master application's nonlinear residual. By default, this tolerance will not take effect.

9.7.6 *output_on_final*

Description: True to output only the final solution.

Date type: Logical

Default value: true

Syntax: Executioner/output_on_final

Note: When this parameter is false, solutions will be outputted after every Picard step of all Picard iterations is taken.

9.7.7 *wrapped_app_tol*

Description: Tolerance on the convergence of wrapped applications

Data type: Real

Default value: -1

Syntax: Executioner/wrapped_app_tol

Note: Typically the wrapped applications are asked to return the residual L2 norm or the difference of its solutions between before and after they are executed. This tolerance put a convergence check on these returned values from the wrapped applications. By default, this tolerance will not take effect. This parameter will be *deprecated* in the future in favor of convergence check with postprocessors.

9.7.8 *multi_app_name*

Description: MultiApp that executes with the *Richardson* executioner

Data type: String

Default value: Empty string

Syntax: Executioner/multi_app_name

Note: *TransportUpdateExecutioner* performs Richardson iteration and evaluates the difference of the solution before and after one iteration. This parameter indicates the executioner that the particular MultiApp is executed with *Richardson* and we can check the convergence of the MultiApp with *wrapped_app_tol*. This parameter will be *deprecated* in the future in favor of convergence check with postprocessors.

9.8 *PicardEigen*

This executioner is almost identitical to PicardSteady except that it wraps the NonlinearEigen executioner. Its parameters include the parameters of PicardSteady except those inherited from Steady executioner and all parameters for the NonlinearEigen executioner. As an example, we demonstrate how this executioner are used to drive NDA (nonlinear diffusion acceleration) calculations in Sec. 3.8.

9.8.1 NonlinearEigen parameters

All parameters of NonlinearEigen are valid parameters. Refer to NonlinearEigen for more information.

9.8.2 *picard_max_its*

Refer to *picard_max_its* in PicardSteady.

9.8.3 *ignore_picard_tol*

Refer to *ignore_picard_tol* in PicardSteady.

9.8.4 *picard_rel_tol*

Refer to *picard_rel_tol* in PicardSteady.

9.8.5 *picard_abs_tol*

Refer to *picard_abs_tol* in PicardSteady.

9.8.6 *output_on_final*

Refer to *output_on_final* in PicardSteady.

9.8.7 *wrapped_app_tol*

Refer to *wrapped_app_tol* in PicardSteady.

9.8.8 *multi_app_name*

Refer to *multi_app_name* in PicardSteady.

9.8.9 *extra_free_pi*

Description: Number of free power iterations at each Picard iteration

Data type: Integer

Default value: 0

Syntax: Executioner/extra_free_pi

9.9 *Richardson*

There are cases the general nonlinear problem Eq. (43) can be written into

$$\mathbf{L}(\mathbf{x}) = \mathbf{R}(\mathbf{x}). \tag{66}$$

Then we can perform an iteration

$$\mathbf{L}(\mathbf{x}^{(n)}) = \mathbf{R}(\mathbf{x}^{(n-1)}). \tag{67}$$

The above equation is solved with our PJFNK solver detailed in Steady. This executioner is not as general as the PicardSteady executioner in a sense that MultiApps, user objects, auxiliary variables on *timestep_begin* and *timestep_end* are not updated during iteration. Instead, typically only part of kernels representing operator R are made operating on the old solutions. The scattering source iteration well-known in radiation transport is a Richardson iteration.

9.9.1 Steady parameters

All parameters of Steady are valid parameters. Refer to Steady for more information.

9.9.2 *xdiff*

Description: Postprocessor evaluating the difference of solutions of two successive Richardson iterations

Data type: String

Default value: <empty>

Syntax: Executioner/xdiff

Note: If this postprocessor is given, it will be used as the iteration error for determining the convergence. The postprocessor must be executed on *linear* when more than one Richardson is going to performed, i.e. *richardson_max_its* is greater than 1. When *richardson_max_its* is equal to one, this postprocessor can be executed on *linear* or *timestep_end*. If this postprocessor is not given, the executioner will evaluate the L2 norm of the solution vectors of two successive Richardson iterations as the iteration error and use the result for determining the convergence.

9.9.3 *richardson_max_its*

Description: Maximum number of Richardson iterations

Data type: Integer

Default value: 1

Syntax: Executioner/richardson_max_its

9.9.4 *richardson_rel_tol*

Description: Relative tolerance for Richardson iterations

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/richardson_rel_tol

Note: The executioner will record the iteration error of the first iteration as the reference.

$$\left\|\mathbf{x}^{(n-1)}-\mathbf{x}^{(n)}\right\| < \max(rtol\left\|\mathbf{x}^{(0)}-\mathbf{x}^{(1)}\right\|, atol), \tag{68}$$

where $\left\|\mathbf{x}^{(n-1)}-\mathbf{x}^{(n)}\right\|$ is the iteration error of n^{th} iteration; $rtol$ is specified with this parameter and $atol$ is specified with *richardson_abs_tol*.

9.9.5 *richardson_abs_tol*

Description: Absolute tolerance for Richardson iterations

Data type: Real

Default value: 10^{-4}

Syntax: Executioner/richardson_abs_tol

9.9.6 *output_after_its*

Description: True to perform an output after Richardson iterations

Data type: Logical

Default value: true

Syntax: Executioner/output_after_its

9.9.7 *debug*

Description: True to add more screen print-outs for debugging purpose

Data type: Logical

Default value: true

Syntax: Executioner/debug

9.10 *AMGUpdate*

This executioner is a special Richardson executioner for CFEM SN. At each iteration, it first evaluate the residual of the old solution

$$\mathbf{r}(\mathbf{x}^{(n-1)}) = -\mathbf{L}(\mathbf{x}^{(n-1)}) + \mathbf{R}(\mathbf{x}^{(n-1)}). \tag{69}$$

Then it solves the linear problem

$$\bar{\mathbf{L}}\delta\mathbf{x} = \mathbf{r}(\mathbf{x}^{(n-1)}), \tag{70}$$

where $\bar{\mathbf{L}}\mathbf{x}$ is an approximation of $\mathbf{L}(\mathbf{x})$, and update solution

$$\mathbf{x}^{(n)}) = \mathbf{x}^{(n-1)}) + \delta\mathbf{x}. \tag{71}$$

Currently we use the block-diagonal Jacobian, where each block corresponds to a primal variable, as the approximation. This executioner is currently hard-coded BoomerAMG to solve Eq. (70) by solving each individual block sequentially. The Jacobian is constructed and stored. Because there is no PJFNK solver used, all the parameters in Steady are invalid except *petsc_options*, *petsc_options_iname* and *petsc_options_value*. The rest of parameters in Richardson are valid parameters. This executioner works only when *scheme* is SAAF-CFEM-SN or LS-CFEM-SN.

9.10.1 *xdiff*

Refer to *xdiff* in Richardson.

9.10.2 *richardson_max_its*

Refer to *richardson_max_its* in Richardson.

9.10.3 *richardson_rel_tol*

Refer to *richardson_rel_tol* in Richardson.

9.10.4 *richardson_abs_tol*

Refer to *richardson_abs_tol* in Richardson.

9.10.5 *output_after_its*

Refer to *output_after_its* in Richardson.

9.10.6 *debug*

Refer to *debug* in Richardson.

9.10.7 *fixed_jacobian*

Description: True to assemble the Jacobian during the initialization stage

Data type: Logical

Default value: true

Syntax: Executioner/fixed_jacobian

Note: When this parameter is false, the Jacobian will be assembled right before the Richardson iteration starts.

9.10.8 *amg_tol*

Description: Tolerance on the algebraic multigrid solve

Data type: Real

Default value: 10^{-6}

Syntax: Executioner/amg_tol

9.10.9 *amg_abs_tol*

Description: Absolute tolerance on the algebraic multigrid solve

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/amg_abs_tol

Note: AMG cycles will be terminated if either this parameter or *amg_tol* is met. There could be chances that the initial residual norm of a variable is already bellow this tolerance. In such a case, no correction will be done on the solution for this variable.

9.10.10 *amg_max_its*

Description: Maximum number of multigrid cycles

Data type: Integer

Default value: 1000

Syntax: Executioner/amg_max_its

9.10.11 *pre_pc_setup*

Description: True to setup Pre-Conditioner data during the matrix assembly

Data type: Logical

Default value: False

Syntax: Executioner/pre_pc_setup

Note: This parameter provides users an option that using more memory to store the data for AMG to avoid the CPU-time on evaluating them on-the-fly.

9.11 *SweepUpdate*

This executioner is like AMGUpdate, also a special Richardson executioner, developed for DFEM SN. It is a matrix-free scheme on solving Eq. (70). Because the Jacobian is block-wise lower-triangular, where each block corresponds a mesh element. So we can sweep through the mesh to solve Eq. (70) on local elements. We also call this solver as sweeper. As AMGUpdate, all the parameters in Steady are invalid. The rest of parameters in Richardson are valid parameters. This executioner works only when *scheme* is DFEM-SN.

9.11.1 *xdiff*

Refer to *xdiff* in Richardson.

9.11.2 *richardson_max_its*

Refer to *richardson_max_its* in Richardson.

9.11.3 *richardson_rel_tol*

Refer to *richardson_rel_tol* in Richardson.

9.11.4 *richardson_abs_tol*

Refer to *richardson_abs_tol* in Richardson.

9.11.5 *output_after_its*

Refer to *output_after_its* in Richardson.

9.11.6 *debug*

Refer to *debug* in Richardson.

9.12 *IQS* (improved quasi-static)

This is an executioner derived from Transient for solving transient problems. It is based on the observation that the magnitude of the solution (or power) changes faster than the angular, spatial and energy distribution of the solution in radiation transport simulations. The distribution or shape of the solution thus can be solved in much large time steps along with magnitude update in lots of micro-steps between two time steps. Because the magnitude equation is zero-dimensional, update with lots of micro-steps is trivial and can be done fast. The cost on shape update is about the same as the normal time step solves in Transient executioner. As the result, this executioner can significantly reduce the CPU time without loss the accuracy for certain applications. As an enhancement, we can repeat the shape update at every time step multiple times with the magnitude update because they are coupled. All the parameters in Transient are valid. Currently only CFEM-Diffusion supports IQS.

9.12.1 Transient parameters

All parameters of Transient are valid parameters. Refer to Transient for more information.

9.12.2 *n_micro*

Description: Number of small timesteps for pke solves in each macro timestep

Data type: Integer

Default value: 1

Syntax: Executioner/n_micro

9.12.3 *power_initial*

Description: Initial power for PRKE solve

Data type: Real

Default value: 1

Syntax: Executioner/power_initial

9.12.4 *IQS_error_tol*

Description: The absolute nonlinear residual to shoot for during IQS iterations

Data type: Real

Default value: 10^{-50}

Syntax: Executioner/IQS_error_tol

Note: This is used to control the convergence of the Picard iteration at every time step due to the power profile on micro-steps of this time step. Seems to me this is redundant and should be deprecated in the future.

9.12.5 *do_iqs_transient*

Description: True to perform IQS transient otherwise normal transient calculations without IQS

Data type: Logical

Default value: True

Syntax: Executioner/do_iqs_transient

Note: This parameter is necessary for situations where accessing the IQS machinary is required but no IQS calculations is needed. For example, when *pke_param_csv* is provided.

9.12.6 *pke_param_csv*

Description: The CSV file used to output PKE parameters

Data type: String

Default value: <empty>

Syntax: Executioner/pke_param_csv

Note: When this parameter is provided, PKE (point kinetics equation) parameters will be evaluated and outputted in the CSV file at every time step.

10 *Postprocessors and User Objects*

While postprocessors are generally for postprocessing and users need to know about. User objects in Rattlesnake serves more general purpose other than postprocessing. They can be directly participating into the residual evaluation, which users do not need to know for using Rattlesnake. So we only document user objects for postprocessing purpose here.

10.1 MOOSE Postprocessors

The complete list of MOOSE postprocessors can be found in Fig. 24. Their parameters are fairly strait-forward and can be found in MOOSE documents.

10.2 MOOSE Vector Postprocessors and User Objects

The complete list of MOOSE vector postprocessors and user objects can be found in Fig. 25. Their parameters are fairly strait-forward and can be found in MOOSE documents.

10.3 Rattlesnake Postprocessors

(to be added) Reaction rate.

10.4 Rattlesnake User Objects

10.4.1 BaseLibObject

This user object loads the multigroup cross section library in YAKXS format which can be used by *CoupledFeedbackNeutronicsMaterial*.

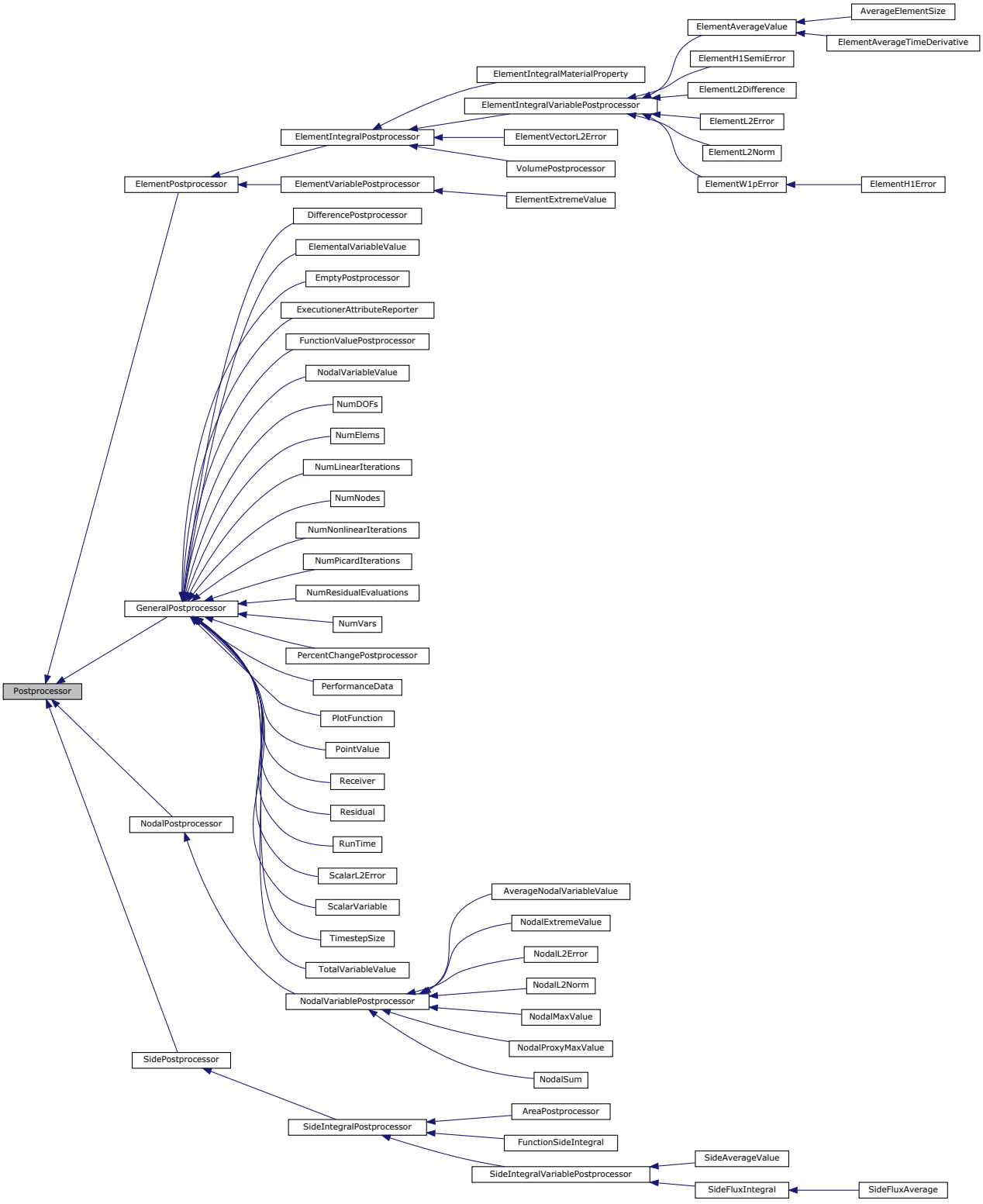


Figure 24 Moose postprocessors.

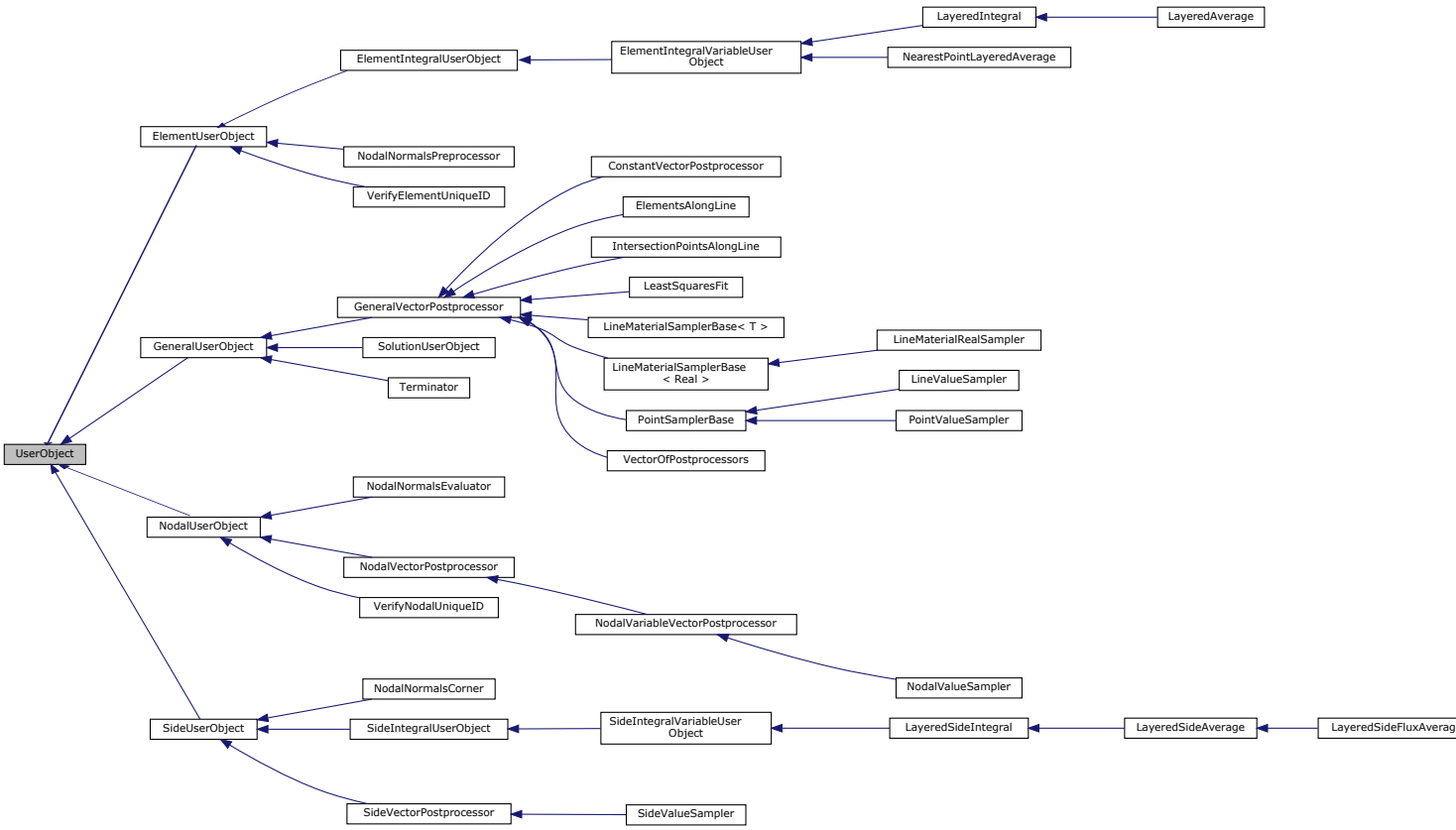


Figure 25 Moose vector postprocessors and user objects.

10.4.1.1 *block*

Description: Mesh blocks the multigroup libraries are for

Data type: Vector of block names

Default value: <empty>

Syntax: UserObjects/*/block

Note: This parameter is used to decide if the multigroup libraries are to be loaded when the mesh is distributed. Empty means that the libraries are to be loaded always.

10.4.1.2 *force_load*

Description: Force the loading of all libraries even if block ID is not in local mesh

Data type: Logical

Default value: false

Syntax: UserObjects/*/force_load

10.4.1.3 *library_file*

Description: File name where libraries are included

Data type: String

Default value: <required>

Syntax: UserObjects/*/library_file

10.4.1.4 *library_type*

Description: Library type

Data type: Enumeration (/MultigroupLibrary/GammaMGLibrary/TransmutationLibrary/)

Default value: <required>

Syntax: UserObjects/*/library_type

Note: Currently only *MultigroupLibrary* type is supported.

10.4.1.5 *isMeter*

Description: True to indicate the mesh is in unit of meter

Data type: Logical

Default value: false

Syntax: UserObjects/*/isMeter

10.4.1.6 *debug*

Description: True to report contents of the master library

Data type: Logical

Default value: false

Syntax: UserObjects/*/debug

10.4.1.7 *library_name*

Description: Master library name from which all sublibraries corresponding to the material IDs will be loaded

Data type: String

Default value: <required>

Syntax: UserObjects/*/library_name

10.4.1.8 *library_ids*

Description: Material IDs to load from the library

Data type: Vector of integers

Default value: <empty>

Syntax: UserObjects/*/library_ids

Note: If this parameter is not specied all material IDs contained in the master library are loaded.

10.4.2 *VariableCartesianCoreMap*

This user object evaluates integrated values of variables on blocks, materials, regions, assemblies, pins of assemblies and sectors and rings of fuel pins and etc. over the entire solution mesh. The integrated values are then outputted either on screen or into a file in a readable formatted manner. The evaluation is done by going through all elements of the mesh and accumulating the quantities based on the corresponding IDs assigned to the elements. Every element can have the block ID, material ID, region ID, assembly ID, fuel pin IDs, sector ID and ring ID. While block, material, region IDs are somewhat independent, assembly, fuel pin and sector and ring IDs are from the hierarchical structure of the geometry. Elements with the same pin ID may belong to different assemblies and elements with the same sector ID or ring ID may belong to different pins in different assemblies. To format the outputs, the single assembly ID is decoded into assembly x, y and z IDs with the number of assemblies in x, y and z directions in the core, which can be used to indicate where the assembly is located radially and axial for outputting. Similarly the single pin ID can be used for indicating the pin radial location. Elements aligned axially all must have the same pin ID. While the block ID is typically inherent to the mesh, like a mesh in the Exodus format, and is maintained by the mesh framework, other IDs requires the users' input and stored in this user object.

This user object provides two ways on how this IDs are inputted and assigned.

1. The most general way is to use elemental variables in the mesh file for all IDs. Currently this user object uses the fixed variable names for the IDs listed in Table 26. The number of assemblies in x, y and z direction of the entire mesh is automatically deferred while reading their x, y and z IDs and the x, y and z IDs are encoded into a single assembly ID. The number of pins in x and y direction of each individual assembly is also automatically deferred. It is noted that this way of ID input and assignment requires the mesh generator to create the IDs in the variables and store them in the mesh file. Currently INSTANT LWR

geometry mesh generator has the capability of generating all these IDs. Thus the C5G7 mesh generated with INSTANT is used for illustrating the ID assignment in Fig. 26. It is noted that the mesh in the water in Fig. 26 is removed for clarity. We are planning to have more built-in mesh generators to incorporate more geometry information in the generated mesh.

2. Through a regular Cartesian grid. Users provide the assembly boundaries in x y and z directions and number of pins in x and y direction. The elements falls into the grids are assigned with the corresponding assembly IDs and pin IDs. The element centroids are used for determining if the elements are completely inside a pin or completely outside. Elements covering two pins are not allowed. Although the functionality of this way is limited, i.e. only assembly and pin IDs can be assigned through this way, it provides a useful and quick way for generating the core maps due to its simplicity. It is actually quite useful because the geometry or at least the part of geometry we are mostly interested in are typically structured.

Table 26 Element variables for IDs.

ID type	Variable name
Material ID	material_id
Region ID	subdomain_id
Assembly ID	assembly_x_id
	assembly_y_id
	assembly_z_id
Pin ID	pin_x_id
	pin_y_id
Ring ID	ring_id
Sector ID	sector_id

10.4.2.1 *execute_on*

Description: Determines when the user object is evaluated and outputted

Data type: enumeration (refer to MOOSE execute_on)

Default value: timestep_end

Syntax: UserObjects/*/execute_on

10.4.2.2 *print*

Description: Vector of keywords controlling the information to be printed

Data type: Enumeration (/block/material/region/assembly/pin/pin_resolved/)

Default value: block + assembly

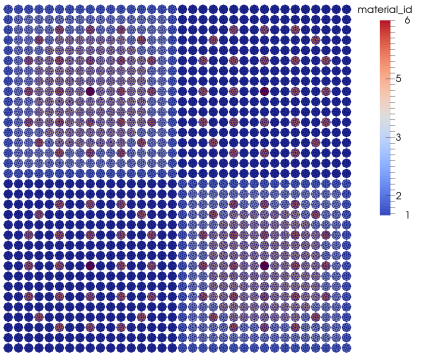
Syntax: UserObjects/*/print

Note: Multiple enumeration can be given for this parameter. If this parameter contains selections that the IDs that are not assigned to, a warning will be generated on screen.

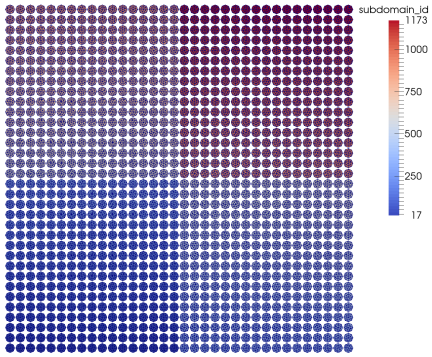
10.4.2.3 *output_in*

Description: Name of the file that data will be outputted in

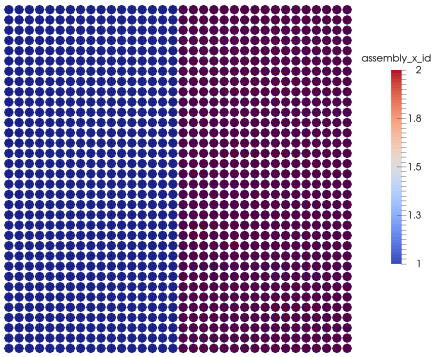
Data type: String



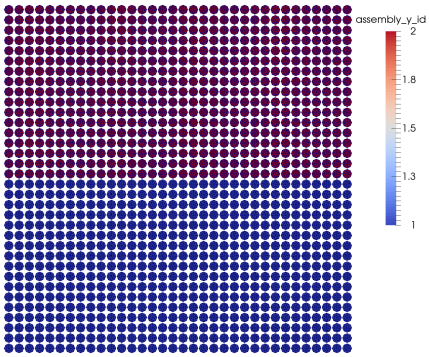
(a) Material



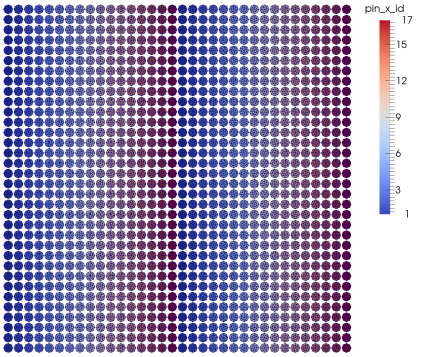
(b) Region



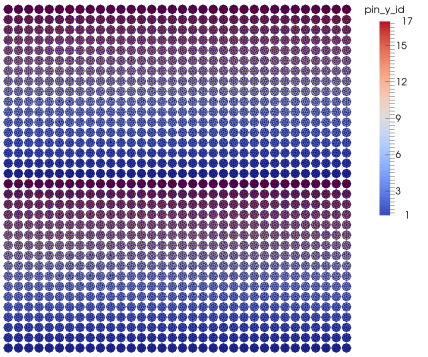
(c) Assembly in x direction



(d) Assembly in y direction



(e) Pin in x direction



(f) Pin in y direction

Figure 26 Element IDs for the 2D C5G7 benchmark generated by INSTANT mesh generator.

Default value: <empty>

Syntax: UserObjects/*/output_in

Note: If this parameter is not set, data will be outputted on screen.

10.4.2.4 *regular_grid*

Description: Wether or not the grid is regular and constructed from grid coordinates in the input

Data type: Logical

Default value: false

Syntax: UserObjects/*/regular_grid

10.4.2.5 *grid_coord_x*

Description: Grid or assembly coordinates in x direction

Data type: Vector of reals

Default value: <empty>

Syntax: UserObjects/*/grid_coord_x

Note: This parameter is required when *regular_grid* is true. The number of assemblies in x direction nx is the size of this parameter minus one. The assembly i in x direction is within $\left[grid\_coord\_x\left[i-1\right], grid\_coord\_x\left[i\right]\right], i = 1, \cdots, nx\right]$.

10.4.2.6 *grid_coord_y*

Description: Grid or assembly coordinates in y direction

Data type: Vector of reals

Default value: <empty>

Syntax: UserObjects/*/grid_coord_y

Note: This parameter is required when *regular_grid* is true and the mesh dimension is bigger than one. The number of assemblies in y direction ny is the size of this parameter minus one. The assembly i in y direction is within $\left[grid\_coord\_y\left[i-1\right], grid\_coord\_y\left[i\right]\right], i = 1, \cdots, ny\right]$.

10.4.2.7 *grid_coord_z*

Description: Grid or assembly coordinates in z direction

Data type: Vector of reals

Default value: <empty>

Syntax: UserObjects/*/grid_coord_z

Note: This parameter is required when *regular_grid* is true and the mesh dimension is bigger than two. The number of layers in z direction nz is the size of this parameter minus one. The layer i in z direction is within $\left[grid\_coord\_z\left[i-1\right], grid\_coord\_z\left[i\right]\right], i = 1, \cdots, nz\right]$.

10.4.2.8 *num_sub_grids_x*

Description: Number of sub-grids or pins in x direction

Data type: Vector of integers

Default value: <empty>

Syntax: UserObjects/*/num_sub_grids_x

Note: The size of this parameter must be equal to nx, i.e. the size of *grid_coord_x* if it is given. If it is not given and *regular_grid* is true then number of pins in x direction of all assemblies will be one.

10.4.2.9 *num_sub_grids_y*

Description: Number of sub-grids or pins in y direction

Data type: Vector of integers

Default value: <empty>

Syntax: UserObjects/*/num_sub_grids_y

Note: The size of this parameter must be equal to ny, i.e. the size of *grid_coord_y* if it is given. If it is not given and *regular_grid* is true then number of pins in y direction of all assemblies will be one.

10.4.2.10 *variables*

Description: Variables to be mapped

Data type: Vector of variable names

Default value: <required>

Syntax: UserObjects/*/variables

Note: Averaged values of the variables on the selections will be evaluated and outputted.

10.4.3 *FluxCartesianCoreMap*

This user object differs from the *VariableCartesianCoreMap* in that it interacts with the transport systems for automatically determining what variables are to be coupled for generating the core map. Thus it shares the same set of parameters as *VariableCartesianCoreMap* except *variables*. It generates the values with the fission neutron production rate, the power density, the neutron absorption rate, the total neutron flux and the group-wise neutron fluxes on blocks, materials, regions and pin-resolved. It generates the map with either the fission neutron production rate or the power density, and the total neutron flux when required on assemblies. It generates the map with either the fission neutron production rate or the power density on pins.

10.4.3.1 *execute_on*

Refer to *execute_on* in *VariableCartesianCoreMap*.

10.4.3.2 *print*

Refer to *print* in *VariableCartesianCoreMap*.

10.4.3.3 *output_in*

Refer to *output_in* in *VariableCartesianCoreMap*.

10.4.3.4 *regular_grid*

Refer to *regular_grid* in *VariableCartesianCoreMap*.

10.4.3.5 *grid_coord_x*

Refer to *grid_coord_x* in *VariableCartesianCoreMap*.

10.4.3.6 *grid_coord_y*

Refer to *grid_coord_y* in *VariableCartesianCoreMap*.

10.4.3.7 *grid_coord_z*

Refer to *grid_coord_z* in *VariableCartesianCoreMap*.

10.4.3.8 *num_sub_grids_x*

Refer to *num_sub_grids_x* in *VariableCartesianCoreMap*.

10.4.3.9 *num_sub_grids_y*

Refer to *num_sub_grids_y* in *VariableCartesianCoreMap*.

10.4.3.10 *transport_system*

Description: Name of the transport system

Data type: String

Default value: <required>

Syntax: UserObjects/*/transport_system

Note: The user object uses this parameter to interact with the transport system to determin where this user object is defined on and what variables and material properties are to be coupled in.

10.4.3.11 *print_assemblywise_fluxes*

Description: True to print assembly-wise total scalar fluxes

Data type: Logical

Default value: False

Syntax: UserObjects/*/print_assemblywise_fluxes

10.4.3.12 *print_groupflux*

Description: True to print fluxes for each group on all except pins

Data type: Logical

Default value: False

Syntax: UserObjects/*/print_groupflux

10.4.3.13 *power_map_from*

Description: What material property that normalized power map is generated from

Data type: Enumeration (/nu_sigma_f/kappa_sigma_f/)

Default value: nu_sigma_f

Syntax: UserObjects/*/power_map_from

10.4.3.14 *print_fission_absorption_ratio*

Description: True to print ratio between fission neutron production and absorption loss on assemblies and pins

Data type: Logical

Default value: False

Syntax: UserObjects/*/print_fission_absorption_ratio

10.4.4 *SAAFWrapper*

This is the user object wrapping a high order transport system with SAAF-CFEM-SN scheme for nonlinear diffusion acceleration.

10.4.4.1 *input_file*

Description: The input file for the high order SAAF-CFEM-SN system

Data type: String

Default value: <required>

Syntax: UserObjects/*/input_file

10.4.4.2 *accelerate_high_moments*

Description: True to accelerate higher angular moments when they present

Data type: Logical

Default value: False

Syntax: UserObjects/*/accelerate_high_moments

10.4.4.3 *initial_correction*

Description: True to do an initial correction

Data type: Logical

Default value: False

Syntax: UserObjects/*/initial_correction

Note: When this parameter is true, the angular fluxes on the high order system need to be initialized to non-zero values.

10.4.5 *LSWrapper*

This is the user object wrapping a high order transport system with LS-CFEM-SN scheme for nonlinear diffusion acceleration.

10.4.5.1 *input_file*

Refer to *input_file* in *SAAFWrapper*.

10.4.5.2 *accelerate_high_moments*

Refer to *accelerate_high_moments* in *SAAFWrapper*.

10.4.5.3 *initial_correction*

Refer to *initial_correction* in *SAAFWrapper*.

11 Auxiliary variables and Kernels

Reaction rates.

12 *Outputs*

Rattlesnake uses the MOOSE output system for the output. Variables, postprocessors can be outputted through the output system directly. Exodus outputs can contain both the variables and postprocessors. CSV (comma separated value) outputs are typically used for outputting scalar values like the postprocessors. Details about MOOSE output system can be found at http://mooseframework.org/wiki/MooseSystems/Outputs/.

13 Advanced Features with MOOSE syntax

Rattlesnake is an open system in a sense that users can add new objects through the MOOSE syntax and let them interact with the transport systems. For doing this, you will need to know the names of the objects already added by the transport systems including variables, auxiliary variables, material properties. Additional variables, kernels, auxiliary variables, auxkernels, materials, user objects, multiapps, transfers, initial conditions and etc. can be added. LRA benchmark actually demonstrates that how a simple temperature model can be added and used to affect the transport system. Rattlesnake can also interact with other MOOSE modules or applications easily for multiphysics simulations. Multiphysics coupling with MultiApp and Transfer (sprint problem)

13.1 Rattlesnake Transfers

References

[1] Yaqi Wang and Javier Ortensi. *YAKXS - The XML Multigroup Cross Section Library*. INL, 2016.

[2] Frank R. Schaefer. *GetPot Version 1.1.16, Powerful Input File and Command Line Parser*, 2007.

[3] Keisuke Kobayashi, Naoki Sugimura, and Yasunobu Nagaya. 3D radiation transport benchmark problems and results for simple geometries with void region. *Progress in Nuclear Energy*, 39(2):119–114, 2001.

[4] V.E. Henson and U.M. Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002.

[5] Gregory D. Sjaardema, Larry A. Schoof, and Victor R. Yarberry. *EXODUS: A Finite Element Data Model*. Sandia National Laboratories, 2008.

[6] Lawrence Livermore National Laboratory. *VisIt User's Manual*, 2005.

[7] Kitware. *ParaView Guide, A Parallel Visualization Application*.

[8] ANL. National energy software center: Benchmark problem book. Technical report, ANL-7416, 1985.

[9] T. M. Sutton and B. N. Aviles. Diffusion theory methods for spatial kinetics calculations. *Progress in Nuclear Energy*, 30(2):119–182, 1996.

[10] Yaqi Wang, Mark D. DeHart, Derek R. Gaston, Frederick N. Gleicher, Richard C. Martineau, Javier Ortensi, John W. Peterson, and Sebastian Schunert. Convergence study of Rattlesnake solutions for the two-dimensional C5G7 MOX benchmark. In *Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method*, Nashville, Tennessee, April 1–23 2015.

[11] E. E. Lewis, M. A. Smith, G. Palmiotti, T. A. Taiwo, and N. Tsoul-Fanidis. Benchmark specification for deterministic 2-D/3-D MOX fuel assembly transport calculations without spatial homogenisation (C5G7 MOX). Technical Report NEA/NSC/DOC(2001)4, OECD/NEA Expert Group on 3-D Radiation Transport Benchmarks, 2001.

[12] Yaqi Wang. *INSTANT: User Manual*. Idaho National Laboratory, Idaho Falls, ID, USA, 1. edition, June 2012.

[13] Yaqi Wang. *INSTANT Theory Manual - Part I. PN-hybrid-FEM for the Multigroup Transport Equation*. Idaho National Laboratory, Idaho Falls, ID, USA, 1. edition, February, 2012.

[14] S. Schunert, Y. Wang, J. Ortensi, F.N. Gleicher, B.A. Baker, M.D. DeHart, and R.C. Martineau. A flexible nonlinear diffusion acceleration method for the first order sn equations. In *PHYSOR 2016: Unifying Theory and Experiments in the 21st Century*, Sun Valley, ID, USA, May 1-5, 2016. American Nuclear Society.

[15] S. Schunert, Y. Wang, J. Ortensi, F.N. Gleicher, B.A. Baker, M.D. DeHart, and R.C. Martineau. On the degradation of the effectiveness of nonlinear diffusion acceleration with parallel block jacobi splitting. In *Nuclear Power: Leading the Supply of Clean, Carbon Free Energy,2016 ANS Annual Meeting*, New Orleans, LA, USA, 2016. American Nuclear Society.

[16] Yaqi Wang. Nonlinear diffusion acceleration for the multigroup transport equation discretized with s_n and continuous FEM with Rattlesnake. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, pages 2648–2665, Sun Valley, Idaho, May 5–9 2013.

[17] Yaqi Wang and Frederick N. Gleicher. Revisit boundary conditions for the self-adjoint angular flux formulation. In *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, The Westin Miyako, Kyoto, Japan, September 28 - October 3 2014.

[18] Yaqi Wang, Hongbin Zhang, and Richard Martineau. Diffusion acceleration schemes for the self-adjoint angular flux formulation with a void treatment. *Nuclear Science and Engineering*, 176:201–225, 2014.

[19] Jon Hansen, Jacob Peterson, Jim Morel, Jean Ragusa, and Yaqi Wang. A least-squares transport equation compatible with voids. *Journal of Computational and Theoretical Transport Special Issue: Papers from the 23rd International Conference on Transport Theory*, 43(1–7), 2014.