Rattlesnake User Manual

Yaqi Wang¹, Sebastian Schunert¹, Benjamin A. Baker², Vincent Labouré²

¹ Nuclear Engineering Methods Development
Idaho National Laboratory
P.O. Box 1625
Idaho Falls, ID 83415-3840

² Reactor Physics Design and Analysis
Idaho National Laboratory
P.O. Box 1625
Idaho Falls, ID 83415-3840

April, 2019
DISCLAIMER

This information was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness, of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.
Rattlesnake User Manual

Yaqi Wang, Sebastian Schunert, Benjamin Baker, Vincent Labouré

April, 2019

Idaho National Laboratory
Nuclear Systems Design and Analysis
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517
## Contents

1 Introduction  

2 Getting started  
   2.1 Obtain access  
   2.2 Install and test  
   2.3 Use tutorials as templates  
   2.4 Instruction on using this manual  
   2.5 General Rattlesnake inputs  
   2.6 When encountering problems  
   2.7 Get involved  

3 Tutorial: Example input files  
   3.1 Kobayashi benchmark  
      3.1.1 Problem description  
      3.1.2 Mesh  
      3.1.3 Transport system with SAAF-CFEM-SN  
      3.1.4 Materials  
      3.1.5 Postprocessors  
      3.1.6 Solver  
      3.1.7 Outputs  
      3.1.8 Results  
   3.2 Benchmark 16A1-1 (Eigenvalue Problem)  
      3.2.1 Problem description  
      3.2.2 Mesh  
      3.2.3 Transport system  
      3.2.4 Materials  
      3.2.5 Executioner  
      3.2.6 Outputs  
      3.2.7 Optional - (Postprocessors, AuxVariables, AuxScalarKernels and Functions)  
      3.2.8 Run the file  
   3.3 Benchmark 16A1-1 (Transient Problem)  
      3.3.1 Changes from Eigenvalue to Transient File  
      3.3.2 MultiApp and Transfers  
      3.3.3 Transient Code and Results
### 3.4 Takeda benchmark Mode 4

---

### 3.5 LRA benchmark (14-A1)

- **3.5.1 Problem description**
- **3.5.2 Mesh**
- **3.5.3 Transport System**
- **3.5.4 Materials**
- **3.5.5 Initial conditions on scalar fluxes**
- **3.5.6 Temperature equation**
- **3.5.7 Postprocessors and core map**
- **3.5.8 Executioner**
- **3.5.9 Preconditioning**
- **3.5.10 Outputs**
- **3.5.11 Primary results**
- **3.5.12 Further Exploration**
- **3.5.13 Using IQS**

---

### 3.6 LRA PKE

- **3.6.1 Dump PKE Parameters for LRA Benchmark**
- **3.6.2 Create a PKE Model for LRA Benchmark to Reproduce the Power History**
- **3.6.3 Fit Reactivity and Generation Time with Averaged Temperature and Control-Rod Fraction**
- **3.6.4 Create PKE Model for LRA Benchmark with the Fitted Functions**
- **3.6.5 Results with the PKE Model with the Fitted Functions**

---

### 3.7 C5G7-2D using First Order NDA solver

- **3.7.1 Problem Description**
- **3.7.2 Mesh Generation**
- **3.7.3 Nonlinear Diffusion Acceleration and the MultiApp System**
- **3.7.4 SN Mesh Block**
- **3.7.5 SN Transport System Block**
- **3.7.6 Transport Materials**
- **3.7.7 SN Postprocessor Block**
- **3.7.8 SN Executioner Block**
- **3.7.9 SN Outputs Block**
- **3.7.10 Diffusion Mesh Block**
- **3.7.11 Diffusion TransportSystems Block**
- **3.7.12 Diffusion Materials Block**
- **3.7.13 Diffusion Executioner Block**

---

3
7 Functions

7.1 MOOSE Functions ................................................................. 214
   7.1.1 MOOSE Functions in MOOSE Framework ............................. 214
   7.1.2 SlopeFunction ............................................................... 214
   7.1.3 StepFunction ............................................................... 215

7.2 Transport Functions ........................................................... 216
   7.2.1 ConstantSourceFunction ............................................... 216
   7.2.2 PulsedSourceFunction .................................................. 217
   7.2.3 DirectionalSourceFunction ............................................ 218
   7.2.4 FilePNTransportFunction ............................................. 218
   7.2.5 Customized TransportFunction (Advanced) ......................... 221

7.3 Adjustable Function .......................................................... 221

7.4 Phase Functions ................................................................... 222
   7.4.1 IsotropicPhaseFunction ................................................. 222
   7.4.2 RayleighPhaseFunction .................................................. 223
   7.4.3 TruncatedDiracPhaseFunction ........................................ 223
   7.4.4 HenyeyGreensteinPhaseFunction ..................................... 224
   7.4.5 HeavisidePhaseFunction ................................................ 225

8 Transport Materials ................................................................ 226

8.1 type ................................................................................. 226

8.2 Neutronics materials .......................................................... 227
   8.2.1 Brief introduction to YAKXS ......................................... 227
   8.2.2 Material properties declared by neutronics materials .......... 229
   8.2.3 Common parameters for all neutronics materials ............... 229
   8.2.4 ConstantNeutronicsMaterial ........................................... 238
   8.2.5 ConstantMatIDNeutronicsMaterial ................................... 243
   8.2.6 FunctionNeutronicsMaterial .......................................... 243
   8.2.7 MixedNeutronicsMaterial .............................................. 246
   8.2.8 MixedMatIDNeutronicsMaterial ....................................... 248
   8.2.9 CoupledFeedbackNeutronicsMaterial ............................... 249
   8.2.10 CoupledFeedbackMatIDNeutronicsMaterial ..................... 250
   8.2.11 CRoddedNeutronicsMaterial .......................................... 251
   8.2.12 MixedRoddedNeutronicsMaterial .................................... 253

8.3 Thermal radiation materials ............................................... 255
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.1</td>
<td>Material properties declared by thermal radiation materials</td>
<td>255</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Common parameters for all thermal radiation materials</td>
<td>255</td>
</tr>
<tr>
<td>8.3.3</td>
<td>ConstantTRMaterial</td>
<td>258</td>
</tr>
<tr>
<td>8.3.4</td>
<td>FunctionTRMaterial</td>
<td>260</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9</th>
<th>Executioner</th>
<th>261</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Steady</td>
<td>261</td>
</tr>
<tr>
<td>9.2</td>
<td>Preconditioning</td>
<td>268</td>
</tr>
<tr>
<td>9.3</td>
<td>Transient</td>
<td>274</td>
</tr>
<tr>
<td>9.4</td>
<td>InversePowerMethod</td>
<td>278</td>
</tr>
<tr>
<td>9.5</td>
<td>NonlinearEigen</td>
<td>283</td>
</tr>
<tr>
<td>9.6</td>
<td>CriticalitySearch</td>
<td>286</td>
</tr>
<tr>
<td>9.7</td>
<td>PicardSteady</td>
<td>291</td>
</tr>
<tr>
<td>9.8</td>
<td>PicardEigen</td>
<td>292</td>
</tr>
<tr>
<td>9.9</td>
<td>Richardson</td>
<td>293</td>
</tr>
<tr>
<td>9.10</td>
<td>AMGUpdate</td>
<td>295</td>
</tr>
<tr>
<td>9.11</td>
<td>SweepUpdate</td>
<td>297</td>
</tr>
<tr>
<td>9.12</td>
<td>IQS (improved quasi-static)</td>
<td>298</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10</th>
<th>Postprocessors and User Objects</th>
<th>305</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1</td>
<td>Common Parameters for Postprocessors</td>
<td>305</td>
</tr>
<tr>
<td>10.1.1</td>
<td>execute_on</td>
<td>305</td>
</tr>
<tr>
<td>10.1.2</td>
<td>use_displaced_mesh</td>
<td>305</td>
</tr>
<tr>
<td>10.1.3</td>
<td>outputs</td>
<td>305</td>
</tr>
<tr>
<td>10.2</td>
<td>MOOSE Postprocessors</td>
<td>306</td>
</tr>
<tr>
<td>10.3</td>
<td>MOOSE Vector Postprocessors and User Objects</td>
<td>306</td>
</tr>
<tr>
<td>10.4</td>
<td>Rattlesnake General Postprocessors</td>
<td>306</td>
</tr>
<tr>
<td>10.4.1</td>
<td>PostprocessorReduction</td>
<td>306</td>
</tr>
<tr>
<td>10.4.2</td>
<td>DynamicalScale</td>
<td>309</td>
</tr>
<tr>
<td>10.4.3</td>
<td>FindMaximumError</td>
<td>309</td>
</tr>
<tr>
<td>10.4.4</td>
<td>PostprocessorRatio</td>
<td>309</td>
</tr>
<tr>
<td>10.4.5</td>
<td>ReportBalance</td>
<td>309</td>
</tr>
<tr>
<td>10.4.6</td>
<td>SPHFactor</td>
<td>309</td>
</tr>
<tr>
<td>10.4.7</td>
<td>PrintIQSAmplitude</td>
<td>309</td>
</tr>
<tr>
<td>10.5</td>
<td>Rattlesnake Elemental Postprocessors</td>
<td>310</td>
</tr>
<tr>
<td>10.5.1</td>
<td>FluxReactionIntegral</td>
<td>310</td>
</tr>
</tbody>
</table>
10.5.2 ElementAverageMaterialProperty ................................................. 310
10.5.3 ElementL2Diff ................................................................. 310
10.5.4 ElementMultiVarL2Diff ......................................................... 310
10.5.5 FieldDynamicalScale ......................................................... 310
10.5.6 InnerProduct ................................................................. 310
10.5.7 FluxMomentL2Error ......................................................... 311
10.5.8 MaterialGroupAverage ......................................................... 312
10.5.9 MaterialL2Diff ................................................................. 312
10.5.10 MaterialL2Comparison ....................................................... 313
10.5.11 PrintIQSPower ................................................................. 313
10.6 Rattlesnake Nodal Postprocessors ............................................... 313
10.6.1 NodalInnerProduct ........................................................... 313
10.7 Rattlesnake Side Postprocessors ............................................... 313
10.7.1 PartialSurfaceCurrent ........................................................ 313
10.8 Rattlesnake Postprocessors for Pebble-Tracking-Transport .............. 314
10.8.1 PebbleReactionRate .......................................................... 314
10.9 Rattlesnake Vector Postprocessors .............................................. 316
10.9.1 ElementVariableValuesAlongLine .......................................... 317
10.9.2 SPHFactors .................................................................. 318
10.9.3 PrintIQSAmplitudeVector ..................................................... 318
10.9.4 IntegralVariableElemIDVPP .................................................. 318
10.10 Rattlesnake User Objects ....................................................... 318
10.10.1 VariableCartesianCoreMap .................................................. 318
10.10.2 FluxCartesianCoreMap ....................................................... 323
10.10.3 SAAFWrapper ................................................................. 325
10.10.4 LSWrapper ................................................................. 326
10.10.5 RattlesnakeVTKWriter ....................................................... 326
10.10.6 SolutionVectorFile ........................................................... 327
10.10.7 TransportSolutionVectorFile ............................................... 328

11 Auxiliary variables and Kernels ............................................... 330
11.1 Common Parameters for Auxiliary Kernels ..................................... 330
11.1.1 variable .................................................................. 330
11.1.2 execute_on ................................................................. 330
11.1.3 use_displaced_mesh .......................................................... 330
11.2 MOOSE Auxiliary Kernels ................................................................. 331
11.3 Rattlesnake Auxiliary Kernels ............................................................. 331
  11.3.1 CopyAux .................................................................................. 332
  11.3.2 CoupledAux ............................................................................ 332
  11.3.3 DivergenceAux ....................................................................... 332
  11.3.4 ElemIDAux .............................................................................. 332
  11.3.5 MaterialCSRAux .................................................................. 333
  11.3.6 NonlinearAdiabaticHeatAux ...................................................... 333
  11.3.7 NonlinearAdiabaticHeatIQS ....................................................... 333
  11.3.8 OrthogonalizationAux .............................................................. 333
  11.3.9 ReactionRate .......................................................................... 333
  11.3.10 VectorReactionRate ............................................................... 333
  11.3.11 SelfSubtractionAux ................................................................. 333
  11.3.12 SubtractionAux ..................................................................... 333
  11.3.13 ShowExtraMeshInfo .............................................................. 334
  11.3.14 SweepOrderAux ................................................................... 334
  11.3.15 VariableTimeIntegrationIQS ................................................... 334
  11.3.16 WeightedVariableSum ............................................................ 334
  11.3.17 FluxWithoutSPHFactor ........................................................ 334

12 Outputs ......................................................................................... 335

13 Other MOOSE syntax .................................................................... 336
  13.1 Rattlesnake Kernels .................................................................... 336
    13.1.1 SplitDGScalarDiffusion .......................................................... 336
  13.2 Rattlesnake Initial Conditions ...................................................... 338
  13.3 Rattlesnake Markers .................................................................... 338
  13.4 Rattlesnake Materials .................................................................. 338
  13.5 Rattlesnake MultiApps .................................................................. 338
    13.5.1 QuasiStaticSolveMultiApp ...................................................... 339
    13.5.2 RayTracerMultiApp ............................................................... 339
  13.6 Rattlesnake Transfers .................................................................. 339
    13.6.1 TransportSystemVariableTransfer ........................................... 339
    13.6.2 MultiAppAdjointFluxTransfer ................................................. 340
    13.6.3 EigenvalueTransfer ............................................................... 341
    13.6.4 MultiAppNumericVectorTransfer ............................................ 341
    13.6.5 MultiAppVariableTransfer .................................................... 341
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 of 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, developments in multiple disciplines etc. The number of shape functions on the same element support is determined by the local expansion of the polynomial order. The solution in Rattlesnake, i.e. the primal variables, are represented by continuous or discontinuous FEM shape functions depending on the solution scheme. When using continuous shape functions, the FEM basis functions of choice are often the Lagrangian shape functions. Auxiliary variables such as reaction rates or power should be represented by discontinuous shape functions such as monomials, because their definition often involves material properties that can be discontinuous across mesh elements.

The key architectural components within Rattlesnake are:

1. Particle type - The particle refers to the phenomena to be modeled. Currently, this is either neutrons or thermal radiation (i.e. photons in \( \sim 0.1\text{-}1000\mu\text{m} \) wavelength). There is also an option "common" that uses the common portions of all radiation transport equations.

2. Equation type - The type of equation can either be the primal equation with the flux as the variable or the adjoint equation with the adjoint or importance flux as the variable.

   - A \( k \)-eigenvalue problem solves for the multiplication factor and the (self-sustained) fundamental mode in a case where no external sources of particles are present. \( k \)-eigenvalue problem is only for neutron. In a \( k \)-eigenvalue problem the neutron fission sources are balanced to the losses by the inverse of the multiplication factor \( 1/k \). No time derivative term is present in the eigenvalue calculations.
   - The steady-state source problem solves a problem where the flux is driven by a distributed source and/or inflow of particle over the boundary. No time derivative term is present in steady-state calculations.
   - The time-dependent transient problem is solved that can comprise distributed sources or inflow boundary conditions.

   The equation that is solved depends on the problem type. \( k \)-eigenvalue problems are only permissible with particle type neutron.

4. Scheme for solve - The schemes are explained below. The particle type, equation type and equation problem type are capabilities within the scheme type. A scheme is usually a combination of an angular and a spatial discretization method applied to a particular form of the particle transport equation, e.g. SAAF-CFEM-SN: continuous FEM discretizing the self-adjoint angular flux equations using the \( S_N \) angular discretization.

5. Multischeme - Multischeme is a capability to deploy different discretization schemes with varying resolutions on different subdomains of the same problem without fixed-point iterations on subdomain interface variables. Regions of different levels of homogenization can be treated most efficiently with the multischeme 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.
When setting up a Rattlesnake input file, the selection of the right discretization scheme is very important. The schemes within Rattlesnake are listed below along with characteristics and additional capabilities:

- **SAAF-CFEM-SN**: stands for SAAF (self-adjoint angular flux)\(^1\) formulation with CFEM (continuous finite element methods) and SN (discrete ordinates methods). SAAF is a 2\(^{\text{nd}}\) order formulation, which is a transformation of the Boltzmann transport equation or 1\(^{\text{st}}\) order formulation. CFEM is a method for discretizing the transport equation in space that yields 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 scales roughly linearly with increasing the number of streaming directions. Thus, SN is typically more suitable for heterogeneous calculations, where higher angular resolution is desired, than for problems with significant spatial homogenization. This is true for all SN schemes.

  **Characteristics**
  
  - Global particle conservation: global particle conservation guarantees that the sum of all leakage, source and absorption terms is zero when integrating over the entire domain; locally or element-wise particles are not conserved. 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 \(\Sigma_t\) is zero or close to zero, respectively. The SAAF-CFEM-SN scheme implemented in Rattlesnake has a void treatment to overcome this issue.
  
  - Lagrangian shape functions are supported up to a polynomial order of two (limitation imposed by MOOSE (libMesh)). Other shape function families should have a similar maximum order. The disadvantage to increasing the polynomial order is a significant increase in computing cost. However, increasing the FEM order also 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 if SN is used for the angular discretization. The acceleration is accomplished by using Picard iterations between the transport-corrected low-order (i.e. diffusion) solves and the high-order (i.e. 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.

\(^1\)Do not confuse SAAF with the adjoint equation. SAAF is a scheme for solving equations and can be applied to any equation type.
There is a near-void/void treatment for SAAF-CFEM-PN (which is a hybrid method between SAAF and LS) that preserves the global conservation of the scheme at the cost of an additional asymmetric streaming term.

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 too 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 2nd 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.
  - 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 discretization can be conservative improving 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 a direct discretization of the Boltzmann transport or 1st order equation.

  Characteristics

  - It does not require void treatment.
  - DFEM can use MONOMIAL or L2_LAGRANGE shape functions. There is no order limitation on the Monomial shape function, but similar to the CFEM schemes L2_LAGRANGE are capped at 2nd 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, i.e. problems with void or near-void regions, when compared to CFEM schemes. DFEM is both locally and globally 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 lowering its memory footprint.
  - The down side is that we do not have an efficient transport 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 ($\hat{\Omega}$). It has the advantages and disadvantages of DFEM and PN.

Characteristics
- It does not need void treatment.
- In particular, this scheme is typically preferred over second-order PN schemes when the solution is rapidly varying, e.g. for transient calculations in advection-dominated regimes. In that case, the tendency for PN to give negative a scalar flux can become problematic.
- A spatially-dependent and energy-dependent filter can be added to damp the angular derivatives and mitigate this negativity issue. More details can be found in the theory manual.
- Thermal Radiation Transport is also available for that scheme.

CFEM-Diffusion: Diffusion calculation with CFEM.

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

DFEM-Diffusion: 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.
- Typical, the number of unknowns is higher 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. In addition, AMG can perform poorly for diffusion solves in NDA iterations because the closure terms (drift term, face closure) are convective in nature. In this case, a recommended course of action is to increase the number of V-cycles AMG uses.

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. SAAF-SN can be used for void or near void regions because Rattlesnake contains a special void treatment. A fairly similar void treatment has also been developed for SAAF-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 to any order for the shape functions when using monomial shape functions. CFEM methods are restricted to 2nd order for Lagrangian shape functions. When measured against the number of unknowns, higher order shape functions typically produce more accurate solutions. A trade-off exists between the order of the shape functions and the computational efficiency. The original transport formulation (i.e. DFEM-SN or DFEM-PN) avoids the problems associated with SAAF and LS. However, at the present moment DFEM-SN does not have an efficient sweeper with parallel domain decomposition. DFEM-PN tends to be slower as efficient preconditioning remains an open question. The user should select schemes based on the particular application.

Multischeme transport capability [1], 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, ISOXML [2], 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. Cross sections are accessed by importing files containing flexible interpolation tables.
Now, let us briefly introduce to the MOOSE/Rattlesnake syntax. Rattlesnake takes text-based input files, which are parsed by HIT parser provided by MOOSE. Advantages of using HIT 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
  </sn>
</>

# 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
  </nm>
</>
```
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 (== block in the input jargon) 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 output occurs,
# which quantities are outputted and which file types are used to save the data.
# 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]
exodus = true

This input deck is copied from the file "test/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 enclosed in brackets [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 start with a block name prepended with a [ ./sub-block name] enclosed with brackets; blocks must conclude with [../]. The sub-block names are 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 =" 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 

More sub-blocks could have been defined 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.

In the following some useful command line options that are commonly used are listed. 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 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 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. Access can be requested by contacting the Idaho National Laboratory.

2.2 Install and test

1. The first step depends on where you will install Rattlesnake
   - Local desktop:
     It is noted that user must comply with the export-control rules for this installation. 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 name.

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
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. More options on running the regression tests can be seen with

   >./run_tests --help

   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 from each other. 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; ISOXML 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; ISOXML 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 ISOXML**: Criticality search; ISOXML 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 in this manual. All parameters in blue color are advanced and red are basic. Most parameters have default values or are allowed 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. But true/false are preferred for clarity. The types can also be vectors in which case they must be enclosed in quotes and separated by spaces.

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]
  ...
  ...
  ...
  [.../scheme-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 please 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 typically 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.

```bash
# 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 Rattlesnake 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.
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

The Mesh block is used to specify the geometry representation in the finite element model, i.e., by the mesh 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 identical to the subdomains 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 [3] 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 subdomains 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/quadruples/octuples the number of mesh elements in the file by using the uniform_refine = 1 option.
2.5.2 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 depend on space you need to use [ScalarVariables] and [ScalarKernels] instead of [Variables] and [Kernels].
2. If the quantity you are after is a derived quantity where the spatial dependence has been integrated out, you might need to use a post-processor.
3. If the variable is dependent on space but is not itself part of the nonlinear system of equations, it should be defined as an aux variable. Reaction rates are typically defined as aux variables as they can be computed from the primal flux solution. Because of the discontinuity of material properties in space they use discontinuous FEM representations.
4. The Variables block is only for primal variables that are part of the nonlinear problem.

For several discretization schemes such as CFEM-Diffusion and SAAF-CFEM-SN must be continuous. A common type is Lagrange. MOOSE allows for multiple variables to be coupled with little effort, this is known as multiphysics 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 at least one particular application.

Below is an example of a variable being defined. Note that units are up to the user and should work out 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 (i.e. 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: $\frac{dn(t)}{dt} - \rho(t) - \beta - \sum_{i=1}^{6} C_i(t) \lambda_i = 0$. Where each of the the terms $\frac{dn(t)}{dt}$, $\rho(t) - \beta$, $\sum_{i=1}^{6} C_i(t) \lambda_i$ 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 first transforms the equation into the weak form by multiplication with a test function and integration over the domain. Subsequently integration by parts is used to reduce the order of the derivative on the solution yielding a term in the domain (volume term) and a term on the boundary. The volume term is integrated as a kernel while the boundary terms is useful to weakly impose boundary conditions via MOOSE IntegratedBC. Neumann boundary conditions are typically implemented this way. In case Dirichlet boundary conditions are used, the degree of freedom associated with the boundary is removed from the nonlinear system and the boundary term previously obtained by the integration by parts does not need to be computed. 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 = -\(\nabla \cdot k \nabla T\) - q + \(C_p\rho \frac{dT}{dt}\) = 0

# Use Adiabatic Equation = \(C_p\rho \frac{dT}{dt} - q = 0\) (q is the source term)

```
[./HeatConduction]
  type = HeatConduction
  variable = Tfuel
  block = '101 102 103 104 105 106'
```

```
[./HeatStorage]
  type = HeatConductionTimeDerivative
  variable = Tfuel
  block = '101 102 103 104 105 106'
```

```
[./HeatSource]
  type = CoupledForce
  variable = ScaledPowerDensity
  block = '106'
```

**[BCs]**

# This is a DirichletBC that sets the value of the solution at the boundary
# The associated degrees of freedom are removed and never enter the
# nonlinear system. If no BC are specified the default is Neumann BC with zero flow

# The DirichletBC
```
[./TempBC]
  type = DirichletBC
  variable = Tfuel
  boundary = '1 2 3 4 5 6'
  value = 300.0
```

Note that each of the kernels for the heat equation involves the \(T_{fuel}\) variable even if temperature was not part of the kernel, such as the HeatSource term. The variable parameter specifies which nonlinear variable this kernel operates on.

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 are quantities that are dependent on space and that can be computed without solving a nonlinear system, i.e. they can be computed using available primal variables, postprocessors, material properties, etc. For Rattlesnake reaction rates are frequently represented by AuxVariables and due to potential discontinuities in the material properties, discontinuous shape functions must be used. Usually, Monomial shape functions are 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".
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.
Included before the functions block are blocks for the AuxVariables and AuxKernels. The input 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 and then remains constant. 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

The Materials block is used 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'
```

29
# 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_spectrum is the delayed neutron spectrum 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

Postprocessor is a system in MOOSE to represent single Real numbers either computed by reduction operations on spatially dependent quantities (Variables, Materials, AuxVariables) or any other general Real number such as execution time. The Postprocessors block provides the user a location to perform calculations for computing derived quantities from variables, other post-processors, material properties, or functions. Some of the common post-processors are: Integrals with respect to space, functions, maximum/minimum value, receiver (a receiver simply saves a single value provided e.g. by a MultiApp).

Below shows several post-processors and associated blocks that communicate with it.
In the above example, the first two post-processors are performing integrals with respect to space on the scalar flux primal variables $f_{\text{moment}}$ for group 0 and 1. The third post-processor points to the function $\text{TotalFlx\_function}$ and saves the output value. The function uses the other three post-processors to calculate its value. The fourth post-processor stores a single value that is coming from a transfer between another application. *MultiApps* and *Transfers* are explained in section 2.5.10. For this example, the value is a normalizing constant determined from an eigenvalue calculation.

### 2.5.9 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* verb 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.
2.5.10 MultiApps and Transfers

[MultiApps] and [Transfers]

The multi-app blocks provide the syntax by which subapplications can be spawned by the master process. The transfers block allows specifying how data is transferred from and to the subapplication. In Sec. 3.3.2 the multi-app was used to calculate the eigenvalue as a subapplication and the transfers block was used to transfer the desired results to the transient problem master. This capability is leveraged in facilitating Picard iterations in multiphysics modeling.

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

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

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

[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]
```
In the above example, the multi-app initiates the eigenvalue solve for the specified input file "SS-Rev2.i" before the first time step and uses the Rattlesnake application. After the solve the data is transferred using the transfers specified in 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 solution. 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 receive and store the value and hold it so that other calculations could be normalized to it.

2.5.11 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, .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 spatial 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.
2.5.12 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 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.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Mathematical adjoint</th>
<th>Neutron</th>
<th>Thermal radiation</th>
<th>Transient</th>
<th>Multischeme</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFEM-Diffusion</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DFEM-Diffusion</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>SAAF-CFEM-SN</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SAAF-CFEM-PN</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LS-CFEM-SN</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LS-CFEM-PN</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>DFEM-SN</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>DFEM-PN</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Below are examples of TransportSystems inputs for 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
```
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'

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

  n_delay_groups = 6  # added because of "particle = neutron"
fission_source_as_material = true  # added because of "particle = 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
```

35
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 to make sure all elements are connected properly through node merging.

2.6.3 Run your problem in debug mode

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

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

If you are using Mac, run your problem with

```bash
lldb -- ./rattlesnake-dbgs -i <InputFile>
```

For Linux, use

```bash
gdb --args ./rattlesnake-dbgs -i <InputFile>
```

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

```bash
b libmesh_terminate_handler
```

then run

```bash
r
```

To obtain a back trace of the calling stack use

```bash
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

A user forum managed by INL email solver, GRP-mammoth-rattlesnake-users@inl.gov, has been set up for discussing issues when using Rattlesnake/MAMMOTH. Licensed users can be added to the accessing list per request.

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 strict rules/SQA (software quality assurance) procedures for how Rattlesnake can be changed. 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 that was originally developed to probe $P_N$ and $S_N$ method’s performance in the presence of voids. Under these circumstances, second order methods require special treatment and $S_N$ methods have to cope with the problem of ray effects. 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 Kobayashi problem [4] are shown as the third problem in Fig. 1. This configuration 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 scattering or with the scattering cross sections being set to half of the corresponding total cross sections.

<table>
<thead>
<tr>
<th>Region</th>
<th>$S$ ($n \cdot cm^{-3} \cdot s^{-1}$)</th>
<th>$\Sigma_t$ ($cm^{-1}$)</th>
<th>$\Sigma_s$ ($cm^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>$10^{-4}$</td>
<td>$0.5 \times 10^{-4}$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

3.1.2 Mesh

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

```python
<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'
ix = '2 8 2 8'
ix = '2 4 2 4'
subdomain_id = '
```

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

Figure 1  Geometry of the Kobayashi benchmark.
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, $\{1,x\} \times \{1,x\} \times \{1,z\}$).

```plaintext
[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 for this problem sufficiently. Typically for SN calculations, we do not care much about the angular flux, so we use hide_angular_flux preventing angular fluxes to be written to 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.

```plaintext
[Materials]
[./region13]
  type = ConstantNeutronicsMaterial
```
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 as the cross sections are not subject to feedback/change due to changing composition (depletion) or temperature.

[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'
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 iterations instead of using the full PFJNK solver. We still leave the parameters for the full PFJNK solver commented in the Executioner block. Because the discretization is CFEM based, we choose the special executioner AMGUpdate. BoomerAMG [5] is used in this executioner with the strong threshold equal to 0.7, which is the recommended value for three-dimensional calculations. 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 [6] output for this benchmark for visualization. The Exodus file can be viewed by VisIt [7] or ParaView [8]. 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 with the execute_on parameter.

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. It appears that the utilized Gauss-Chebyshev quadrature yields good accuracy in contrast to the $S_{24}$ quadrature.

3.2 Benchmark 16A1-1 (Eigenvalue Problem)

All files necessary for running this tutorials with Rattlesnake are under 'rattlesnake/tutorials/16A1' folder.
Figure 2  Kobayashi benchmark results. $N_p$: polar angles, $N_\phi$: azimuthal angles.
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. [9]. 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 Super-critical 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>
<thead>
<tr>
<th>Zone</th>
<th>Length(cm)</th>
<th>Number of Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>47.374</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>47.374</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>20</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Zone</th>
<th>Group</th>
<th>(v\Sigma_f^g)</th>
<th>(\Sigma_i^g)</th>
<th>(\Sigma_{s-s}^g)</th>
<th>(\Sigma_{s-s}^{g-g'})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,7</td>
<td>1</td>
<td>8.3441E-4</td>
<td>2.411E-1</td>
<td>2.33644E-1</td>
<td>3.598E-3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.2776E-4</td>
<td>4.172E-1</td>
<td>4.07004E-1</td>
<td>0.0</td>
</tr>
<tr>
<td>2,4,6</td>
<td>1</td>
<td>7.4518E-3</td>
<td>1.849E-1</td>
<td>1.77711E-1</td>
<td>2.085E-3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.1061E-2</td>
<td>3.668E-1</td>
<td>3.53721E-1</td>
<td>0.0</td>
</tr>
<tr>
<td>3,5</td>
<td>1</td>
<td>0.0</td>
<td>9.432E-2</td>
<td>8.571E-2</td>
<td>1.717E-3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.0</td>
<td>1.876E-1</td>
<td>1.7131E-1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

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 \(1/v_1 = 1.851 \times 10^{-09} s/cm, 1/v_2 = 1.088 \times 10^{-08} s/cm\). The prompt and delayed neutron spectra are identical with \(\chi_1 = 1.0\) and \(\chi_2 = 0.0\).
Table 6  Delayed Neutron Parameters

<table>
<thead>
<tr>
<th>Group</th>
<th>$\beta_i$</th>
<th>$\lambda_i$ (sec$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.81E-4</td>
<td>0.0129</td>
</tr>
<tr>
<td>2</td>
<td>6.87E-4</td>
<td>0.0311</td>
</tr>
<tr>
<td>3</td>
<td>6.12E-4</td>
<td>0.134</td>
</tr>
<tr>
<td>4</td>
<td>11.38E-4</td>
<td>0.331</td>
</tr>
<tr>
<td>5</td>
<td>5.12E-4</td>
<td>1.26</td>
</tr>
<tr>
<td>6</td>
<td>1.70E-4</td>
<td>3.21</td>
</tr>
</tbody>
</table>

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.

```plaintext
[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 refine each elements in a mesh by subdividing it into 2/4/8 elements in 1D/2D/3D. 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 elements are doubled/quadrupled/octupled in 1D/2D/3D. When refinement is set to 2 the refinement doubles again and there are four times the number of mesh elements than the original mesh. Modifying the `uniform_refine` option is a convenient method to check for spatial 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.
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 for 1-D geometry the names of the sidesets are left and right.

The discretization method is set to `scheme = SAAF-CFEM-SN` which means that the problem is solved using the 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_groups` and `fission_source_as_material`. The user can choose the shape function family and order. The most common is Lagrange, which is a continuous shape 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 sets up the cross-sections and determines in which zones the cross-sections exist. Below is the input syntax to generate one of the cross-section sets; the remaining sets will be added after this part of the input syntax is explained.
We are using the ConstantNeutronicsMaterial sub-block within Rattlesnake naming it '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 has two indices g and then g'. The convention is that the from group index runs faster than the to group index. 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 iterative tolerances. To perform eigenvalue calculations a simple input like the following would suffice. The most important part for the executioner is for the line type=NonlinearEigen. This command specifies the type of solve we are performing. Notice that the specification for eigenvalue is in two locations: the executioner and TransportSystems; if an eigenvalue problem is specified in the TransportSystems block the executioner must be an executioner for solving an eigenvalue problem.

```
[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 input the solver type is the pre-conditioned Jacobian-Free Newton Krylov. PETSc options are specified by the two vectors petsc_options_iname and petsc_options_value which have to match in length. The examples specifies to use the algebraic multigrid solver (boomeramg) included in the HYPRE package (hypre) and specifies the maximum Krylov subspace size to be 100. Also, the NonlinearEigen executioner uses four free power iterations. NonlinearEigen is not guaranteed to converge to the fundamental mode of the problem. Free power iterations are performed to get the initial guess closer to the fundamental mode to avoid convergence to a higher harmonic. Using 2-6 free power iterations is a fairly common setting.

3.2.6 Outputs

In the outputs block the user can choose how to save the data and where to save them. Since the described problem is an eigenvalue calculation all that is required is the final eigenvalue. The value will be printed on the screen even if there are no output modes specified in the Outputs block. We can save all the information for every mesh point using the exodus = 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'.

The reason for adding the `execute_on` option will be explained further on but the meaning is when within the MOOSE execution order 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 choose 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 `execute_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.
In the input above a new variable \texttt{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 parameter order takes on a new meaning. In case the variable is dependent on space (i.e. a standard FEM variable) 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 \texttt{AuxScalarKernels} is setup to be a function and apply to the variable \texttt{TotalFlux}. The kernel must also know which function is going to be applied. That is why the user defined name is placed after \texttt{function =}. The time step option was placed in the kernel to control when the calculation is performed.

The \texttt{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 \texttt{F1} and \texttt{F2} not \texttt{Flux1} and \texttt{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 \texttt{execute_on} feature. If you comment out the \texttt{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 \texttt{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 \texttt{timestep_end} could have also been included). While for the TotalFlux it was calculated at \texttt{timestep_end}.

Now we know that once we get to the transient calculation we will want to normalize by \texttt{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 \texttt{ParsedFunction} to divide by the total flux from the eigenvalue calculation. There is a problem if we try to mix the postprocessor values with variable types. Conversion can take place but there is really no reason why we should make it this complicated. We can perform the calculation in an alternative way.

Below are two post-processors that we could use to either transfer the \texttt{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. 
[./TotalFlux_PP]
  type = ScalarVariable
  variable = TotalFlux
[../]
#
#
# This PP will use the function TotalFlux_function to obtain its value. 
[./TotalFlux_Fun_PP]
  type = FunctionValuePostprocessor
  function = TotalFlux_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 $\text{Flux}_1 + \text{Flux}_2$ equals $\text{TotalFlx}_\text{Fun}_\text{PP}$.

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.

```plaintext
[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
```

54
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>
```

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 during the transient. Notice that the GlobalParams block was added. This block is a convenient way to place options common to multiple blocks or values that will be included many times over and shorten the input file. Local parameter take precedence over global parameters. 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 out of zone 4 into their own materials. That way we can easily copy and modify the cross-sections for the transient.
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'
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 0'

[../]

Mat6

[.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 0'

[../]

Mat3

[.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]
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 input given in 3.2.8 with additional input 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 input to have it perform the transient.

The initial eigenvalue solution is to find the multiplication factor defined as the value by which to divide the fission neutron production term to achieve a steady-state solution. By applying it to the fission source within the transient calculation we can enforce a transient starting from this ‘forced’ steady state conditions. An example would be to let the problem heat up during the transient which changes the cross section leading to the problem going into slow, feed-back driven transient. If we did not apply the multiplication factor, the solution would go into a fast transient because the configuration is truly sub- or super-critical. This is useful for simulating isolated parts of reactors, e.g. a pin or assembly, that are by themselves not critical.

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'
```

58
[Postprocessors]

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 *NormalizationFlux* is type *Receiver* and does nothing else. The point here is to store the value from the *TotalFlx_Fun_PP* in the eigenvalue calculation.

Next, modify the function to be divided by the normalization flux.
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 relative tolerance for the nonlinear solves. The \( timestep\_tolerance \) is added because we will demand the data to be output only at certain times. An issue arises when the previous time step is very close to but not exactly equal to the demanded time. This causes that the time stepper attempts an unrealistically small time step which can cause numerical problems. The \( timestep\_tolerance \) is used to decide if the current time step is close enough to the time at which the user forces output. The FunctionDT sub-block creates a function for the \( dt \) value. This particular problem demands a very small \( dt \) for the first couple of time steps and then allows \( dt \) to increase. This function follows a piece-wise linear formulate. Where the \( time\_dt \) vector is the list of \( dt \) values, to interpolate between and \( time\_t \) is the associated times.

```plaintext
[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'

[../]
```

```plaintext
[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'

[../]
```

```plaintext
[./exodus]
  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
```
The Outputs sub-block was also modified by adding a sub-block for CSV and Exodus 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 too much 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 file name for the eigenvalue file in the MultiApps block.

```plaintext
[MultiApps]
[./initial_solve]
  type = FullSolveMultiApp
  app_type = RattlesnakeApp
  execute_on = initial
  positions = '0 0 0'
  input_files = SS_16A1.i
[../]

[Transfers]
[./copy_solution]
  type = TransportSystemVariableTransfer
  direction = from_multiapp
  multi_app = initial_solve
  execute_on = initial
  from_transport_system = SN
  to_transport_system = SN
  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
```

61
In the input the first block defines MultiApps. This is a block that allows the master to spawn several sub problems being executed. For this problem, we are using the FullSolveMultiApp spawning a Rattlesnake sub-application. If Rattlesnake is just a sub-module to an encompassing application, the encompassing application’s name would be placed here, e.g. MAMMOTH. 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 TransportSystemVariableTransfer. 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 the post-processors and variables in the eigenvalue and transient problems 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).

```plaintext

[Mesh]
type = CartesianMesh
dim = 1
dx = '40 47.374 9 34 9 47.374 40'
ix = '20 24 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
```

62
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 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'

[./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'

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

[./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
0.001 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
0.001 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 shows the normalized power history (i.e., 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 \texttt{vlux\_moment\_g0\_L0\_M0} and \texttt{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)](image)
Figure 4  Flux Profile at 0.0, 0.01 and 1.0 sec (Benchmark 16A1-1)

Figure 5  Enable scalar fluxes
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 based on a thermal reactor. It is a super prompt-critical transient. To have better understanding on the cross sections given
The geometry is illustrated in Fig. 8.

The initial flux distribution shall be normalized such that the averaged power density

\[-\frac{1}{v_1} \frac{\partial \phi_1}{\partial t} = -\vec{\nabla} D_1 \vec{\nabla} \phi_1 + (\Sigma_{a,1} + \Sigma_{a,1\rightarrow2})\phi_1(\vec{r}, t) - v(1 - \beta) f(\vec{r}, t) - \frac{2}{k} \lambda_i C_i(\vec{r}, t), \tag{1a}\]

\[-\frac{1}{v_2} \frac{\partial \phi_2}{\partial t} = -\vec{\nabla} D_2 \vec{\nabla} \phi_1 + \Sigma_{a,2}\phi_2(\vec{r}, t) - \Sigma_{a,1\rightarrow2}\phi_1, \tag{1b}\]

\[f(\vec{r}, t) = \sum_{g=1}^{2} \Sigma_{f,g}\phi_g, \tag{1c}\]

\[\frac{\partial C_i}{\partial t} = v \beta_i f - \lambda_i C_i(\vec{r}, t), i = 1, 2, \tag{1d}\]

\[\frac{\partial T(\vec{r}, t)}{\partial t} = \alpha f, \tag{1e}\]

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

\[P(\vec{r}, t) = \kappa f, \tag{1g}\]

where \(\phi_1, \phi_2\) 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; \(\Sigma_{a,1\rightarrow2}\) is the fast-to-thermal scattering cross section; \(\Sigma_{f,1}, \Sigma_{f,2}\) are the fission cross sections; \(v\) is the averaged number of neutrons emitted per fission; \(\beta_1, \beta_2\) are the delayed neutron precursor fractions and \(\beta = \beta_1 + \beta_2\); \(C_1, C_2\) are the delayed neutron precursor concentrations; \(\lambda_1, \lambda_2\) are the decay constants of the delayed neutron precursors; \(f\) is the fission reaction rate; \(P\) is the power density; \(T\) is the temperature; \(\kappa\) is the averaged power released per fission; \(\alpha\) is the combination of \(\kappa\) and the specific heat capacity; \(\gamma\) 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

\[-\vec{\nabla} D_1 \vec{\nabla} \phi_1 + (\Sigma_{a,1} + \Sigma_{a,1\rightarrow2})\phi_1(\vec{r}, t) = \frac{1}{k} \sum_{g=1}^{2} v \Sigma_{f,g}\phi_g, \tag{1h}\]

\[-\vec{\nabla} D_2 \vec{\nabla} \phi_1 + \Sigma_{a,2}\phi_2(\vec{r}, t) = \Sigma_{a,1\rightarrow2}\phi_1. \tag{1i}\]

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

\[\bar{P} = \frac{\int_{V_{\text{core}}} P(\vec{r}, t = 0) d\vec{r}}{\int_{V_{\text{core}}} d\vec{r}}, \tag{1j}\]

where \(V_{\text{core}}\) 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.
Initial two-group constants are presented in Table 7. $\nu$ 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.

<table>
<thead>
<tr>
<th>Region</th>
<th>Material</th>
<th>Group</th>
<th>$D_g$ (cm)</th>
<th>$\Sigma_{a,g}$ (cm$^{-1}$)</th>
<th>$\nu \Sigma_{f,g}$ (cm$^{-1}$)</th>
<th>$\Sigma_{s,1\rightarrow2}$ (cm$^{-1}$)</th>
<th>$\chi_g$</th>
<th>$\nu_g$ (cm$\cdot$s$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fuel 1 with rod</td>
<td>1, 2</td>
<td>1.255</td>
<td>0.008252</td>
<td>0.004602</td>
<td>0.02533</td>
<td>1</td>
<td>$3.0 \times 10^7$ $3.0 \times 10^5$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.211</td>
<td>0.1003</td>
<td>0.1091</td>
<td>0.02533</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Fuel 1 without rod</td>
<td>1, 2</td>
<td>1.268</td>
<td>0.007181</td>
<td>0.004609</td>
<td>0.02767</td>
<td>1</td>
<td>$3.0 \times 10^7$ $3.0 \times 10^5$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.1902</td>
<td>0.07047</td>
<td>0.08675</td>
<td>0.02767</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Fuel 2 with rod</td>
<td>1, 2</td>
<td>1.259</td>
<td>0.008002</td>
<td>0.004663</td>
<td>0.02617</td>
<td>1</td>
<td>$3.0 \times 10^7$ $3.0 \times 10^5$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2091</td>
<td>0.08344</td>
<td>0.1021</td>
<td>0.02617</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Fuel 2 without rod</td>
<td>1, 2</td>
<td>1.259</td>
<td>0.008002</td>
<td>0.004663</td>
<td>0.02617</td>
<td>1</td>
<td>$3.0 \times 10^7$ $3.0 \times 10^5$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2091</td>
<td>0.073324</td>
<td>0.1021</td>
<td>0.02617</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Reflector</td>
<td>1, 2</td>
<td>1.257</td>
<td>0.006034</td>
<td>-</td>
<td>0.04754</td>
<td>-</td>
<td>$3.0 \times 10^7$ $3.0 \times 10^5$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.1592</td>
<td>0.01911</td>
<td>-</td>
<td>0.04754</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Group i</td>
<td>( \beta_i )</td>
<td>( \lambda_i , (s^{-1}) )</td>
<td>( \chi_{d,i,1} )</td>
<td>( \chi_{d,i,2} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.0054</td>
<td>0.0654</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.001087</td>
<td>1.35</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8 LRA benchmark delayed neutron data.

Table 9 LRA benchmark scalar values.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Notation</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axial buckling for both energy groups</td>
<td>( B_{g}^{2} )</td>
<td>( 10^{-4} , (cm^{-2}) )</td>
</tr>
<tr>
<td>Mean number of neutrons per fission</td>
<td>( \nu )</td>
<td>2.43</td>
</tr>
<tr>
<td>Conversion factor</td>
<td>( \alpha )</td>
<td>( 3.83 \times 10^{-11} , (K \cdot cm^{3}) )</td>
</tr>
<tr>
<td>Feedback constant</td>
<td>( \gamma )</td>
<td>( 3.034 \times 10^{-3} , (K^{1/2}) )</td>
</tr>
<tr>
<td>Energy released per fission</td>
<td>( \kappa )</td>
<td>( 3.204 \times 10^{-11} , (J / fission) )</td>
</tr>
<tr>
<td>Initial and reference temperature</td>
<td>( T_{0} )</td>
<td>300 (K)</td>
</tr>
<tr>
<td>Active core volume</td>
<td>( V_{core} )</td>
<td>17550 (cm^{2})</td>
</tr>
</tbody>
</table>

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

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

where \( t \) is time in seconds.

3.5.2 Mesh

The geometry can be meshed using a regular grids and a mesh spacing of 15 cm can be used as the maximum element size to match all material boundaries. So we can use the Rattlesnake built-in mesh generator \texttt{GeneratedBIDMesh} to generate the mesh. The Mesh block in the input would be:

```plaintext
[Mesh]
type = GeneratedBIDMesh
dim = 2
xmin = 0
xmax = 165
ymin = 0
ymax = 165
nx = 11
ny = 11
subdomain='2 1 1 1 1 2 2 3 3 5 5 1 1 1 1 1 1 3 3 5 5 1 1 1 1 1 1 3 3 5 5 1 1 1 1 1 1 3 3 5 5 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 4 5 5 5 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'``
Block 1 to 5 correspond to region 1 to 5 in Fig. 8. Block 6 is the region where control rod are ejected. It is worth mentioning that the parameter uniform_refine can be used in the Mesh block for convergence study. The refinement parameter uniform_refine is used to set the number of successive uniform refinement steps. It is noted that elem_type needs to be set to QUAD9 if second order Lagrange shape functions are used.

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
  [...]
```

We set particle as ‘neutron’ to include fission reactions. We are solving a two-group transient problem. Top and right boundaries are homogeneous Dirichlet. Bottom and left boundaries are reflecting. We only have one single diffusion system applied on the entire solution domain. The number of delayed neutron precursor groups is two. family and order are not specified, which means their default values, i.e. first-order Lagrange shape functions, are to be used.

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 \( B^2 D(\vec{r}, t) \) 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 Doppler 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]
  [.fuell_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 = '6.0678222e-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

[/..]

[/..]

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

[/..]

[/..]
chi = '1.0 0.0'
sigma_r = '0.034851 0.07047'
bz_sqrd = 0.0001

neutron_speed = '3.0e7 3.0e5'
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 = '3.0e7 3.0e5'
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 = '3.0e7 3.0e5'
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 also does not give their values. They are evaluated by $\frac{1}{\pi g} - \Sigma_{r,g}$, 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:

```plaintext
[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:

```plaintext
[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]
We do not put delayed neutron data into *GlobalParams* because the non-fissile material ‘reflect’ 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 conditions for the scalar fluxes. Because fission source and delayed neutron precursor concentrations are treated as material properties so their values are evaluated on the fly based on the most recent scalar flux and 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:

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

[Transfers]
./copy_solution
  type = TransportSystemVariableTransfer
  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 exactly the same variables, we can simply use \texttt{MultiAppSystemCopyTransfer} to transfer all solutions. This transfer will also transfer k-effective if there is one on the transport system. Because the \textit{Mesh}, \textit{Materials} and the \textit{AuxVariables} of the initial eigenvalue problem will be exactly the same as those in the transient problem, we will only provide the rest of the input for the initial calculation here:

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

  G = 2

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

  [.../diff]
    scheme = CFEM-Diffusion
    n_delay_groups = 2
  [.../]

[AuxKernels]
  [.../power]
    type = VectorReactionRate
    scalar_flux = 'sflux_g0 sflux_g1'
    cross_section = kappa_sigma_fission
    variable = power
    block = '1 2 3 4 6'
    execute_on = timestep_end
  [.../]

[Executioner]
  type = NonlinearEigen

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

    free_power_iterations = 2
    source_abs_tol = 1e-12
    source_rel_tol = 1e-12
```

The only difference in the \textit{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 \textit{linear} so we make it executed on 'timestep_end' to save a little computing time. \texttt{NonlinearEigen} is chosen for solving the problem. Their parameters can be found in Sec. 9.5. By default, \texttt{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 in the transient. 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 for most common use cases.

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 has been 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, hence we cannot use LAGRANGE shape functions as for the 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 but not lower, so that no accuracy will be lost. 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]
[./power]
    type = VectorReactionRate
    scalar_flux = 'sflux_g0 sflux_g1'
    cross_section = kappa_sigma_fission
    variable = power
    # due to the initial normalization, we need to have the coefficient 1e-6*vol*nu/kappa
    scale_factor = 1.331039325843e9
    block = '1 2 3 4 6'
    execute_on = 'initial linear'
[../]
[./temperature]
    type = VariableTimeIntegrationAux
    variable = T
    variable_to_integrate = power
    # alpha / kappa
    coefficient = 1.195380774032459
    block = '1 2 3 4 6'
    execute_on = linear
[../]
[
```

We have put the initial condition for the temperature in its declaration block because it is uniformly set to 300K. We also make the two auxiliary kernels execute on *linear* in order to have up-to-date values at each residual evaluation and thus perform strongly-coupled multiphysics calculations. If however, we set the ‘execute_on’ to *timestep_end*, we will basically performing operator split calculations. It is also noted that the coefficient applied in *temperature*
auxiliary kernel is $\frac{\alpha V_{\text{core}} 10^{-6}}{\kappa^2}$ instead of $\frac{\alpha}{\kappa}$ due to the initial normalization. The name of the scalar fluxes and kappa fission cross section can be found in Sec. 4.13 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 specifies 10 quantities to be reported. The most important two are item 4 (power) and 6 (temperature). It is noted that due to our choice of the initial normalization, we need to apply a scaling factor $\nu V_{\text{core}} 10^{-6} \approx 1.331 \times 10^9$ 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.

We do want the initial value of the two postprocessors, so the ‘execute_on’ parameter also contains initial.

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

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. The control parameters are listed in Sec. 9.3. We will not exercise 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 = 1e-2
  l_tol = 1e-2
  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-10
[]
```

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 it can be performed. There are two pieces to preconditioning: (1) the terms that are included in the preconditioning matrix and (2) the PETSc solver used to invert the preconditioning matrix. This section deals with item (1). Best convergence is achieved if the preconditioning matrix is very close to the true Jacobian of the problem. However, including too many (block) off-diagonal terms first increases memory consumption and second these off-diagonal terms might not be implemented. Item (2) is also very important as the inversion of the pre-conditioning matrix uses iterative methods that can fail if the matrix exhibits characteristics making it unsuitable for the selected PETSc method: an example are multigrid solvers applied to convection dominated problems. An insufficiently inverted pre-conditioning matrix will lead to poor convergence even if the original matrix is close to the actual Jacobian.

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:
3.5.10 Outputs

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

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 [10].

<table>
<thead>
<tr>
<th>Table 10</th>
<th>LRA benchmark reference results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial k-effective</td>
<td>0.996368</td>
</tr>
<tr>
<td>k-effective with the control rod completely out</td>
<td>1.015445</td>
</tr>
<tr>
<td>The first averaged peak power ($W/cm^3$)</td>
<td>5455.46</td>
</tr>
<tr>
<td>The occurrence of the first peak (s)</td>
<td>1.44112</td>
</tr>
<tr>
<td>The second normalized peak power ($W/cm^3$)</td>
<td>795.498</td>
</tr>
<tr>
<td>The occurrence of the second peak (s)</td>
<td>2.00164</td>
</tr>
<tr>
<td>Averaged power density at 0.4s ($W/cm^3$)</td>
<td>$1.38437 \times 10^{-6}$</td>
</tr>
<tr>
<td>Averaged power density at 0.8s ($W/cm^3$)</td>
<td>$3.07545 \times 10^{-6}$</td>
</tr>
<tr>
<td>Averaged power density at 1.2s ($W/cm^3$)</td>
<td>$6.67668 \times 10^{-6}$</td>
</tr>
<tr>
<td>Averaged power density at 1.4s ($W/cm^3$)</td>
<td>432.518</td>
</tr>
<tr>
<td>Averaged power density at 2.0s ($W/cm^3$)</td>
<td>795.126</td>
</tr>
<tr>
<td>Averaged power density at 3.0s ($W/cm^3$)</td>
<td>98.4276</td>
</tr>
<tr>
<td>Averaged fuel temperature at 3.0s (K)</td>
<td>1094.15</td>
</tr>
</tbody>
</table>

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.
t
t
t
t
t
t

= 0.4
= 0.8
= 1.2
= 1.4
= 2.0
= 3.0

0.8470
0.7370
0.5987
0.5415
0.4656
0.5067

0.8030
0.7116
0.5957
0.5464
0.4756
0.5102

0.7827
0.7198
0.6378
0.6002
0.5341
0.5580

0.8321
0.8024
0.7598
0.7354
0.6717
0.6823

0.9378
0.9454
0.9476
0.9391
0.8818
0.8765

1.0034
1.0485
1.0978
1.1086
1.0752
1.0541

0.8965
0.9668
1.0501
1.0781
1.0924
1.0621

1.3560
1.1779
0.9549
0.8634
0.7384
0.8039

1.1851
1.0485
0.8761
0.8037
0.6957
0.7464

1.1105
1.0217
0.9068
0.8549
0.7576
0.7903

1.1943
1.1561
1.1014
1.0699
0.9747
0.9868

1.4369
1.4587
1.4765
1.4708
1.3808
1.3661

1.7475
1.8444
1.9558
1.9879
1.9374
1.8896

1.7397
1.9077
2.1147
2.1950
2.2628
2.1880

1.4801
1.6955
1.9721
2.0946
2.3328
2.2461

1.5188
1.3168
1.0641
0.9608
0.8225
0.8973

1.0632
0.9384
0.7816
0.7166
0.6199
0.6664

0.9158
0.8432
0.7500
0.7086
0.6287
0.6556

1.0032
0.9759
0.9369
0.9144
0.8363
0.8441

1.3626
1.3955
1.4290
1.4321
1.3547
1.3348

2.1496
2.2897
2.4552
2.5090
2.4746
2.4059

2.3475
2.6122
2.9456
3.0851
3.2563
3.1399

1.8878
2.2674
2.7762
3.0300
3.6249
3.4795

1.0247
1.2829
1.6392
1.8257
2.3476
2.2532

1.2662
1.0970
0.8859
0.8002
0.6888
0.7532

0.8678
0.7653
0.6371
0.5845
0.5087
0.5482

0.7414
0.6827
0.6079
0.5753
0.5140
0.5369

0.8282
0.8066
0.7764
0.7594
0.6998
0.7070

1.1731
1.2029
1.2346
1.2398
1.1813
1.1646

1.9417
2.0691
2.2214
2.2737
2.2561
2.1949

2.2272
2.4771
2.7942
2.9301
3.1068
2.9978

1.9472
2.3281
2.8399
3.0978
3.6995
3.5541

1.1598
1.4317
1.8061
2.0023
2.5363
2.4370

0.7227
0.6266
0.5073
0.4597
0.3998
0.4384

0.6193
0.5466
0.4562
0.4196
0.3697
0.3995

0.5846
0.5369
0.4767
0.4510
0.4069
0.4272

0.6625
0.6402
0.6102
0.5951
0.5515
0.5611

0.8716
0.8820
0.8918
0.8911
0.8493
0.8438

1.1940
1.2534
1.3246
1.3492
1.3307
1.3027

1.4254
1.5475
1.7020
1.7671
1.8312
1.7774

1.5513
1.7360
1.9769
2.0888
2.2905
2.2155

1.0311
1.1723
1.3590
1.4486
1.6505
1.5970

0.4678
0.4051
0.3278
0.2976
0.2635
0.2911

0.4525
0.3985
0.3317
0.3052
0.2731
0.2973

0.4630
0.4218
0.3701
0.3486
0.3173
0.3366

0.5332
0.5060
0.4708
0.4548
0.4218
0.4352

0.6707
0.6601
0.6447
0.6356
0.6014
0.6070

0.8509
0.8616
0.8728
0.8743
0.8470
0.8425

1.0482
1.0850
1.1297
1.1463
1.1435
1.1277

1.2652
1.3281
1.4066
1.4386
1.4760
1.4508

0.8886
0.9395
1.0036
1.0305
1.0816
1.0632

0.3762
0.3236
0.2593
0.2347
0.2120
0.2371

0.3735
0.3262
0.2681
0.2455
0.2228
0.2456

0.3956
0.3552
0.3050
0.2847
0.2604
0.2804

0.4676
0.4329
0.3889
0.3700
0.3412
0.3586

0.5966
0.5660
0.5262
0.5079
0.4724
0.4873

0.7642
0.7379
0.7027
0.6852
0.6443
0.6565

0.9525
0.9322
0.9039
0.8885
0.8478
0.8568

1.1624
1.1476
1.1253
1.1115
1.0774
1.0848

0.8222
0.8158
0.8047
0.7966
0.7839
0.7887

0.3988
0.3397
0.2677
0.2405
0.2206
0.2502

0.3645
0.3147
0.2538
0.2304
0.2111
0.2362

0.3761
0.3327
0.2789
0.2575
0.2356
0.2575

0.4599
0.4167
0.3621
0.3392
0.3097
0.3310

0.6348
0.5843
0.5193
0.4906
0.4469
0.4703

0.8949
0.8310
0.7476
0.7094
0.6459
0.6737

1.1012
1.0306
0.9375
0.8938
0.8177
0.8474

1.2314
1.1609
1.0669
1.0217
0.9445
0.9744

0.8360
0.7918
0.7319
0.7024
0.6573
0.6771

0.5524
0.4676
0.3642
0.3252
0.3008
0.3439

0.4000
0.3426
0.2725
0.2459
0.2263
0.2555

0.3801
0.3332
0.2752
0.2523
0.2307
0.2543

0.4770
0.4274
0.3648
0.3388
0.3071
0.3311

0.7422
0.6739
0.5860
0.5474
0.4924
85
0.5233

1.3056
1.1927
1.0453
0.9781
0.8773
0.9256

1.5701
1.4414
1.2722
1.1937
1.0712
1.1244

1.4046
1.2961
1.1528
1.0858
0.9790
1.0238

0.8785
0.8136
0.7272
0.6863
0.6250
0.6529


Table 12  Temperature map at different times.

<table>
<thead>
<tr>
<th>Time</th>
<th>300.00</th>
<th>300.00</th>
<th>300.00</th>
<th>300.00</th>
<th>300.00</th>
<th>300.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>3.0</td>
<td>2.0</td>
<td>1.4</td>
<td>1.2</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>0.8</td>
<td>3.0</td>
<td>2.0</td>
<td>1.4</td>
<td>1.2</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>1.2</td>
<td>3.0</td>
<td>2.0</td>
<td>1.4</td>
<td>1.2</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>1.4</td>
<td>3.0</td>
<td>2.0</td>
<td>1.4</td>
<td>1.2</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>2.0</td>
<td>3.0</td>
<td>2.0</td>
<td>1.4</td>
<td>1.2</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>3.0</td>
<td>3.0</td>
<td>2.0</td>
<td>1.4</td>
<td>1.2</td>
<td>0.8</td>
<td>0.4</td>
</tr>
</tbody>
</table>
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 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 computed final power when using 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 418 seconds.

3.5.12 Further Exploration

Several changes are made on the original input file, and included in the folder. It is noted that we changed the integration order to 1 in temperature auxiliary kernel in this exploration. It is used only to make comparison with backward Euler scheme when temperature is treated as a primal variable.

3.5.12.1 Change the ‘execute_on’ of the temperature auxiliary kernel from ‘linear’ to ‘timestep_end’

In the input file, we now have:

```plaintext
[AuxKernels]
[./temperature]
  type = VariableTimeIntegrationAux
  ...
  order = 1
  execute_on = timestep_end
[../]
```

These modifications change the results: we list result from time 1.2s to 1.5s as an example using a constant time step of 0.01s and zero level of uniform refinement. The reason for this is when the temperature is computed on linear,
we have up-to-date residual evaluation, and can perform a strongly-coupled multiphysics calculation. After setting ‘execute_on’ to ‘timestep_end’, we are solving operator split calculation. From the table we can see the maximum relative error in average temperature and average power can be as much as 22.85% and 115.7%, respectively. It is noted that the reference execute-on-linear results are also generated with integration order 1.

<table>
<thead>
<tr>
<th>Time</th>
<th>Execute_on = linear</th>
<th>Execute_on = timestep_end</th>
<th>Error in Ave. T</th>
<th>Error in Ave. P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.20</td>
<td>300.02732</td>
<td>1.41632</td>
<td>300.02734</td>
<td>1.41723</td>
</tr>
<tr>
<td>1.21</td>
<td>300.07803</td>
<td>4.24158</td>
<td>300.07814</td>
<td>4.25005</td>
</tr>
<tr>
<td>1.22</td>
<td>300.24315</td>
<td>13.81365</td>
<td>300.24400</td>
<td>13.90815</td>
</tr>
<tr>
<td>1.23</td>
<td>300.82508</td>
<td>48.68161</td>
<td>300.84126</td>
<td>49.93109</td>
</tr>
<tr>
<td>1.24</td>
<td>302.94850</td>
<td>177.63483</td>
<td>303.18006</td>
<td>195.65272</td>
</tr>
<tr>
<td>1.25</td>
<td>309.82508</td>
<td>48.68161</td>
<td>300.24400</td>
<td>49.93109</td>
</tr>
<tr>
<td>1.26</td>
<td>326.50075</td>
<td>139.21559</td>
<td>344.03194</td>
<td>262.66122</td>
</tr>
<tr>
<td>1.27</td>
<td>354.03691</td>
<td>2303.54743</td>
<td>472.23240</td>
<td>591.24636</td>
</tr>
<tr>
<td>1.28</td>
<td>412.92581</td>
<td>2803.4594</td>
<td>550.08932</td>
<td>719.46136</td>
</tr>
<tr>
<td>1.29</td>
<td>450.33437</td>
<td>2429.15562</td>
<td>569.76985</td>
<td>1646.38128</td>
</tr>
<tr>
<td>1.30</td>
<td>473.48583</td>
<td>1936.8191</td>
<td>569.76985</td>
<td>1646.38128</td>
</tr>
<tr>
<td>1.31</td>
<td>491.11847</td>
<td>1475.0648</td>
<td>581.48982</td>
<td>980.43849</td>
</tr>
<tr>
<td>1.32</td>
<td>504.29428</td>
<td>1102.2269</td>
<td>588.64775</td>
<td>980.43849</td>
</tr>
<tr>
<td>1.33</td>
<td>514.15026</td>
<td>824.50561</td>
<td>593.28608</td>
<td>388.02159</td>
</tr>
<tr>
<td>1.34</td>
<td>521.64645</td>
<td>627.09619</td>
<td>596.55713</td>
<td>273.64027</td>
</tr>
<tr>
<td>1.35</td>
<td>527.51261</td>
<td>490.73596</td>
<td>599.09419</td>
<td>212.23846</td>
</tr>
<tr>
<td>1.36</td>
<td>532.27432</td>
<td>398.34279</td>
<td>601.24263</td>
<td>179.72848</td>
</tr>
<tr>
<td>1.37</td>
<td>536.29960</td>
<td>367.3594</td>
<td>603.19127</td>
<td>163.01469</td>
</tr>
<tr>
<td>1.38</td>
<td>539.84282</td>
<td>296.40920</td>
<td>605.04437</td>
<td>155.02142</td>
</tr>
<tr>
<td>1.39</td>
<td>543.07930</td>
<td>246.97491</td>
<td>569.76985</td>
<td>1646.38128</td>
</tr>
<tr>
<td>1.40</td>
<td>546.13044</td>
<td>255.24377</td>
<td>560.87631</td>
<td>1646.38128</td>
</tr>
<tr>
<td>1.41</td>
<td>549.08119</td>
<td>246.84635</td>
<td>610.50255</td>
<td>153.04275</td>
</tr>
<tr>
<td>1.42</td>
<td>551.99206</td>
<td>243.50929</td>
<td>612.36147</td>
<td>155.50909</td>
</tr>
<tr>
<td>1.43</td>
<td>554.90710</td>
<td>243.85905</td>
<td>614.25856</td>
<td>158.70151</td>
</tr>
<tr>
<td>1.44</td>
<td>557.85939</td>
<td>246.97427</td>
<td>616.20032</td>
<td>162.43830</td>
</tr>
<tr>
<td>1.45</td>
<td>560.87460</td>
<td>252.23892</td>
<td>618.19213</td>
<td>166.62582</td>
</tr>
<tr>
<td>1.46</td>
<td>563.97349</td>
<td>259.23804</td>
<td>620.23887</td>
<td>171.22108</td>
</tr>
<tr>
<td>1.47</td>
<td>567.17346</td>
<td>267.69523</td>
<td>622.34526</td>
<td>176.21090</td>
</tr>
<tr>
<td>1.48</td>
<td>570.48975</td>
<td>277.42508</td>
<td>624.51608</td>
<td>181.60045</td>
</tr>
<tr>
<td>1.49</td>
<td>573.93606</td>
<td>288.30267</td>
<td>626.75631</td>
<td>187.40670</td>
</tr>
</tbody>
</table>

Table 13  Relative error in average temperature and average power

3.5.12.2  Add Picard iteration

Picard Iteration is added with temperature auxiliary kernel ‘execute_on = timestep_end’. In the input file, we now include:

```plaintext
[Executioner]
  type = Transient
  ...
  picard_max_its = 3
  picard_rel_tol = 1e-3
  picard_abs_tol = 1e-3
  ...
```

88
The maximum relative error in average temperature and power of all time steps to Picard iteration are listed in the following table using the same parameters for calculation as in Sec. 3.5.12.1:

<table>
<thead>
<tr>
<th>Picard Iteration</th>
<th>Error in Ave. T</th>
<th>Error in Ave. P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.28E-1</td>
<td>1.15E+0</td>
</tr>
<tr>
<td>2</td>
<td>5.59E-2</td>
<td>3.30E-1</td>
</tr>
<tr>
<td>3</td>
<td>1.84E-2</td>
<td>9.29E-2</td>
</tr>
<tr>
<td>4</td>
<td>5.40E-2</td>
<td>2.83E-2</td>
</tr>
<tr>
<td>5</td>
<td>2.00E-2</td>
<td>1.08E-2</td>
</tr>
</tbody>
</table>

Table 14  Maximum Relative Error in Average Temperature and Power to Picard Iteration

From the table, the relative error in average temperature and power decreases with Picard iterations. With more Picard iteration, we are able to reproduce the result of the fully-coupled case.

3.5.12.3 Turn the temperature to a primal variable

In this multi-variable problem, the residuals of temperature and scalar flux have very different magnitudes before 1 s, which will make the system hard to solve. Therefore, a scaling factor is added to the temperature residual. In the input file, we add the following block to set up the temperature equation:

```plaintext
[Variables]
[./T]
  order = First
  family = L2_LAGRANGE
  initial_condition = 300
  scaling = 1e-3
[../]

[Kernels]
[./time_derive]
  type = TimeDerivative
  variable = T
[../]
[./coupledforce]
  type = CoupledForce
  variable = T
  v = power
  coef = 1.591098860816678e+009
[../]
```

It is noted that 'TransportSystemVariableTransfer' will only transfer primal variables of the transport system, thus the temperature is not transferred, which is expected. Now the T auxiliary variable and 'temperature' auxiliary kernel are not needed for temperature. The results for temperature as aux variable and primal variable agree to within iterative tolerance. The maximum relative error in average temperature and power with different scaling on variable temperature is listed in the following table:
Table 15  Maximum Relative Error in Average Temperature and Power to Picard Iteration

<table>
<thead>
<tr>
<th>Scaling</th>
<th>Error in Ave. T</th>
<th>Error in Ave. P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E-2</td>
<td>5.21E-10</td>
<td>7.51E-09</td>
</tr>
<tr>
<td>1E-3</td>
<td>1.66E-11</td>
<td>5.07E-09</td>
</tr>
<tr>
<td>1E-4</td>
<td>1.73E-11</td>
<td>5.07E-09</td>
</tr>
</tbody>
</table>

3.5.12.4  Added a conduction term in temperature equation

A conduction term with constant thermal conductivity = 0.01W/cm/K is added to the temperature by including the kernel ‘DiffusionApproximation’:

```
[Kernels]
  ...[
    ./diffusion
    type = DiffusionApproximation
    variable = T
    D = conductivity
  ]

```

Also, for the LRA core, the density is approximately 1.05E4, heat capacity is 300 J/kg/K [11]. The diffusion coefficient is calculated as 0.00315 cm, and added into the input file by adding ‘GenericConstantMaterial’:

```
[Materials]
  ...
  ./add_conductivity
    type = GenericConstantMaterial
    prop_names = conductivity
    prop_values = 0.00315
  ]

```

The maximum temperature at time 3s is listed in Table 16. In order to see the effect clearly, results with a 200 cm diffusion coefficient are also included.

<table>
<thead>
<tr>
<th>Diffusion Coefficient</th>
<th>Max. T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3884.3</td>
</tr>
<tr>
<td>0.00315</td>
<td>3884.1</td>
</tr>
<tr>
<td>200</td>
<td>3127.1</td>
</tr>
</tbody>
</table>

Table 16  Maximum Temperature at Time 3s

3.5.12.5  Change the ‘execute_on’ of the temperature auxiliary kernel from ‘linear’ to ‘nonlinear’

In the input file, we now have:
Because the problem is linear when temperature is fixed, updating temperature on nonlinear is essentially equivalent with execute on timestep_end with Picard iterations. Because the nonlinear residual evaluation at the end of each linear iteration is done with the updated temperature, its norm reflects the overall convergence of the nonlinear system. Thus the converged solution of each time step is almost equivalent with the solution with temperature updating on linear. Results between 1.2s and 1.5s presented in Table 17 agree with our expectation. It is also observed that the linear iteration is easier to converge with the price of larger numbers of nonlinear iterations.

<table>
<thead>
<tr>
<th>Time</th>
<th>Execute_on = linear</th>
<th>Execute_on = nonlinear</th>
<th>Error in Ave. T</th>
<th>Error in Ave. P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.20</td>
<td>300.02732</td>
<td>1.41632</td>
<td>300.02732</td>
<td>1.41632</td>
</tr>
<tr>
<td>1.21</td>
<td>300.07803</td>
<td>4.24158</td>
<td>300.07803</td>
<td>4.24158</td>
</tr>
<tr>
<td>1.22</td>
<td>300.24315</td>
<td>13.81365</td>
<td>300.24315</td>
<td>13.81364</td>
</tr>
<tr>
<td>1.23</td>
<td>300.82508</td>
<td>48.68161</td>
<td>300.82508</td>
<td>48.68159</td>
</tr>
<tr>
<td>1.24</td>
<td>302.94850</td>
<td>177.63483</td>
<td>302.94846</td>
<td>177.63505</td>
</tr>
<tr>
<td>1.25</td>
<td>309.88238</td>
<td>580.05637</td>
<td>309.88242</td>
<td>580.05636</td>
</tr>
<tr>
<td>1.26</td>
<td>326.50075</td>
<td>1390.21559</td>
<td>326.50084</td>
<td>1390.21273</td>
</tr>
<tr>
<td>1.27</td>
<td>354.03691</td>
<td>2303.54743</td>
<td>354.03690</td>
<td>2303.54394</td>
</tr>
<tr>
<td>1.28</td>
<td>387.78379</td>
<td>2823.10701</td>
<td>387.78375</td>
<td>2823.10315</td>
</tr>
<tr>
<td>1.29</td>
<td>421.29581</td>
<td>2803.45946</td>
<td>421.29573</td>
<td>2803.45694</td>
</tr>
<tr>
<td>1.30</td>
<td>450.3347</td>
<td>1475.06478</td>
<td>450.33336</td>
<td>1475.06541</td>
</tr>
<tr>
<td>1.31</td>
<td>473.48583</td>
<td>1936.81909</td>
<td>473.48573</td>
<td>1936.81914</td>
</tr>
<tr>
<td>1.32</td>
<td>491.11847</td>
<td>2429.15564</td>
<td>491.11838</td>
<td>2429.15470</td>
</tr>
<tr>
<td>1.33</td>
<td>504.29428</td>
<td>1102.22669</td>
<td>504.29420</td>
<td>1102.22731</td>
</tr>
<tr>
<td>1.34</td>
<td>514.15026</td>
<td>824.50561</td>
<td>514.15020</td>
<td>824.50635</td>
</tr>
<tr>
<td>1.35</td>
<td>521.64645</td>
<td>627.09619</td>
<td>521.64639</td>
<td>627.09692</td>
</tr>
<tr>
<td>1.36</td>
<td>527.51261</td>
<td>490.73596</td>
<td>527.51255</td>
<td>490.73657</td>
</tr>
<tr>
<td>1.37</td>
<td>532.27432</td>
<td>398.34279</td>
<td>532.27427</td>
<td>398.34319</td>
</tr>
<tr>
<td>1.38</td>
<td>536.29960</td>
<td>336.73594</td>
<td>536.29954</td>
<td>336.73623</td>
</tr>
<tr>
<td>1.39</td>
<td>539.84282</td>
<td>296.40920</td>
<td>539.84276</td>
<td>296.40943</td>
</tr>
<tr>
<td>1.40</td>
<td>543.07930</td>
<td>270.74911</td>
<td>543.07925</td>
<td>270.74929</td>
</tr>
<tr>
<td>1.41</td>
<td>546.13044</td>
<td>255.23477</td>
<td>546.13093</td>
<td>255.23491</td>
</tr>
<tr>
<td>1.42</td>
<td>549.08119</td>
<td>246.84635</td>
<td>549.08114</td>
<td>246.84647</td>
</tr>
<tr>
<td>1.43</td>
<td>551.99206</td>
<td>243.50929</td>
<td>551.99201</td>
<td>243.50939</td>
</tr>
<tr>
<td>1.44</td>
<td>554.90710</td>
<td>243.85905</td>
<td>554.90706</td>
<td>243.85915</td>
</tr>
<tr>
<td>1.45</td>
<td>557.85939</td>
<td>246.97472</td>
<td>557.85935</td>
<td>246.97480</td>
</tr>
<tr>
<td>1.46</td>
<td>560.87460</td>
<td>252.23892</td>
<td>560.87456</td>
<td>252.23899</td>
</tr>
<tr>
<td>1.47</td>
<td>563.97349</td>
<td>259.23804</td>
<td>563.97344</td>
<td>259.23811</td>
</tr>
<tr>
<td>1.48</td>
<td>567.17346</td>
<td>267.69523</td>
<td>567.17342</td>
<td>267.69529</td>
</tr>
<tr>
<td>1.49</td>
<td>570.48795</td>
<td>277.42508</td>
<td>570.48791</td>
<td>277.42514</td>
</tr>
<tr>
<td>1.50</td>
<td>573.93606</td>
<td>288.30267</td>
<td>573.93603</td>
<td>288.30272</td>
</tr>
</tbody>
</table>

Table 17 Relative error in average temperature and average power

3.5.13 Using IQS

Finally, we use IQS to solve the LRA benchmark.
3.5.13.1 Another way of preparing the transport initial condition

The original input uses MultiApp/Transfer to set the initial condition for the transport system. We give another way of preparing the transport initial condition here. We first run the steady-state initial eigenvalue input with ‘Output/exodus=true’ on command line to generate the Exodus file. The Exodus file contains the mesh and the solutions. We can then use FileMesh to load the mesh from the file with

```plaintext
[Mesh]
  type = FileMesh
  file = gold/lra_trans_initial_out.e
[]
```

It is noted that we have stored the Exodus file under sub-directory ‘gold’. Finally we load the solution from the Exodus file by adding one line in TransportSystem block as follows

```plaintext
[TransportSystems]
...
  restart_transport_system = true
...
```

This parameter will cause the solution in the mesh file to be loaded for setting initial conditions for all the primal variables of the transport system. The eigenvalue, if present in the mesh file under the name ‘eigenvalue’, will also be loaded for properly scaling the fission terms. GeneratedBIDMesh and MultiApp/Transfer used in the original input can be removed.

3.5.13.2 Preparing the weighting function

IQS requires a weighting function (a function as scalar fluxes) for evaluating the PKE (point-kinetics equation) parameters. This weighting function can be generated by an adjoint calculation. We choose to use the adjoint solution at the beginning time as the weighting function, which can be generated by simply running the initial eigenvalue input with ‘TransportSystems/for_adjoint=true Outputs/file_base=lra_trans_adjoint_out Output/exodus=true’ on command line. It is noted that the mathematical adjoint is the same as the physical adjoint for multigroup diffusion problems. We then store the generated Exodus file in the same ‘gold’ sub-directory and use it as the weighting function with an extra block RestartVariables in the input:

```plaintext
[RestartVariables]
[./adjoint]
  exodus_filename = gold/lra_trans_adjoint_out.e
  variable_names = 'sflux_g0 sflux_g1'
  target_variable_names = 'adjoint_flux_g0 adjoint_flux_g1'
  to_system = AUXILIARY
[../]
```

This block will create two auxiliary variables described with first-order Lagrange shape functions (default), ‘adjoint_flux_g0, adjoint_flux_g1’ from the two variables, ‘sflux_g0, sflux_g1’, in the solution file. It is noted we must follow the naming convention, ‘adjoint_flux_g#’, where ‘#’ is the group index, for weighting functions. IQS will look for these variables for evaluating PKE parameters.
3.5.13.3 IQS executioner

Now the Executioner block becomes

```plaintext
[Executioner]
type = IQS
start_time = 0
end_time = 3
l_tol = 1e-2
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-10
picard_max_its = 5
picard_abs_tol = 1e-5
```

Most parameters remain the same as the original input except that the executioner type is changed to IQS and two more parameters are added for controlling the Picard IQS.

3.5.13.4 Evaluating the temperature with amplitude on micro-steps

The original power auxiliary variable actually becomes the power without amplitude. To evaluate the temperature properly and consider the amplitude variation on all micro time steps within a macro-step, we need to replace the temperature auxiliary kernel with a different one:

```plaintext
[./temperature]
type = VariableTimeIntegrationIQS
variable = T
variable_to_integrate = power
# alpha / kappa
coefficient = 1.195380774032459
block = '1 2 3 4 6'
execute_on = 'linear timestep_end'
[../]
```

This auxiliary kernel needs to be executed on timestep_end as well so that the temperature will be properly updated during PKE parameter evaluations. Similarly, the postprocessor power in the original input needs to be replaced with a postprocessor in a particular type for IQS to take the amplitude into account:

```plaintext
[./power]
type = PrintIQSPower
execute_on = 'initial timestep_end'
variable = power
block = '1 2 3 4 6'
volume_average = true
[../]
```
3.5.13.5 Convergence study

We will only check the time convergence with the fixed mesh. To make the calculations less time-consuming, we will use the original mesh without any refinement in space. The reference solution is generated by a series of calculations with successive uniform time refinements with the Crank-Nicolson time integration scheme. IQS is not used in these calculations for obtaining the reference solution. The time step sizes we used are ‘0.01s, 0.005s, 0.0025s, 0.00125s, 0.000625s, 0.0003125s, 0.00015625s, 0.000078125s, 0.0000390625s’. We run the following command under the tutorial folder for each time step size:

```
../../rattlesnake-opt -i lra_trans.i Executioner/scheme=crank-nicolson Executioner/dt=[time step size]
```

The reference values are extrapolated from the results of the last two time step sizes with a proper convergence order (2 for most of quantities, 3 for the second peak time):

<table>
<thead>
<tr>
<th></th>
<th>Reference Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first averaged peak power ($W/cm^3$)</td>
<td>5678.342825</td>
</tr>
<tr>
<td>The occurrence of the first peak (s)</td>
<td>1.351476967</td>
</tr>
<tr>
<td>The second averaged peak power ($W/cm^3$)</td>
<td>834.4622094</td>
</tr>
<tr>
<td>The occurrence of the second peak (s)</td>
<td>2.001771936</td>
</tr>
<tr>
<td>Averaged power density at 1.2s ($W/cm^3$)</td>
<td>0.072845654</td>
</tr>
<tr>
<td>Averaged power density at 1.4s ($W/cm^3$)</td>
<td>543.5490014</td>
</tr>
<tr>
<td>Averaged power density at 2.0s ($W/cm^3$)</td>
<td>834.0295228</td>
</tr>
<tr>
<td>Averaged power density at 3.0s ($W/cm^3$)</td>
<td>109.6875978</td>
</tr>
<tr>
<td>Averaged fuel temperature at 3.0s (K)</td>
<td>1220.299731</td>
</tr>
</tbody>
</table>

We then plot the errors of quantities of interest with respect to the reference values obtained above in Fig. 11. The number of time steps for a refinement level $n$ is $300 \times 2^n$. The smooth second order convergence can be seen in Fig. 11.
Because we know the shape changes significantly with temperature feedback due to sufficient amount of energy deposition, we choose to use a function time stepper with non-uniform time step sizes to reduce the number of time steps with IQS. The function used for time stepping is

```plaintext
./dts
  type = PiecewiseConstant
  x = '0 1.2 1.42 1.7 2.2 2.5 3'
  y = '0 0.2 0.005 0.02 0.01 0.05 0.1'
  direction = right
  scale_factor = 1
[../]
```

The non-uniform step size specified by this function captures the significant flux redistribution during the transient. This time stepper is by no means optimal. A better time stepper with time adaptation is desired in the future. We can adjust the scale_factor as 1, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125, 0.00390625 to perform a series of uniform refinements in time. We only consider the Picard version of IQS here. We also tune the convergence tolerance a little to reduce the number of linear iterations per time step with IQS. The executioner block with all these changes is:

```plaintext
[Executioner]
  type = IQS
  start_time = 0
  end_time = 3
  l_tol = 1e-4
  nl_rel_tol = 1e-3
  nl_abs_tol = 2e-11
  line_search = none
  petsc_options = '-snes_converged_reason'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart'
  petsc_options_value = 'hypre boomeramg 100'
  timestep_tolerance = 1e-10
  picard_max_its = 5
  picard_abs_tol = 1e-10
  IQS_error_tol = 1e-8
  output_micro_csv = iqs_power.csv
  shape_postprocessor = shape_power
[../TimeStepper]
  type = FunctionDT
  function = dts
[../]
```

Users can output the averaged power density on all micro time steps into a CSV file with `output_micro_csv` and `shape_postprocessor`. These two parameters do not affect the convergence study performed here.

Convergence for the Picard version of IQS with backward Euler scheme and Crank-Nicolson scheme are plotted in Fig. 12 and Fig. 13 respectively. The number of time steps for a refinement level \( n \) is \( 125 \times 2^n \). We can see almost second order convergence initially and then slower convergence rate to the first order with backward Euler scheme and one PKE parameter update per macro step. On the other hand, Crank-Nicolson exhibits consistent second order convergence.
Figure 12  Time convergence of the LRA benchmark with IQS backward Euler.

Figure 13  Time convergence of the LRA benchmark with IQS Crank-Nicolson.
order convergence with one PKE parameter update per macro step. With multiple (six) PKE parameter updates per macro step, the expected convergence rates, first order for backward Euler and second order for Crank-Nicolson, are obtained. For the first peak power to reach an error of $1 \, \text{W/cm}^3$, we need three uniform refinement with IQS while need more than 4 refinements without in Fig. 11. It takes about four times more time steps for Crank-Nicolson without IQS to reach the same level of accuracy than backward Euler with IQS. With the same number of time steps, multiple PKE parameter updates per macro step can reduce the error particularly for Crank-Nicolson. These can become apparent by plotting the errors of first peak power and the power at time 1.4s for all four sets of calculations in Fig. 14. It can be seen that backward Euler and Crank-Nicolson obtain almost similar accuracy with the same number of refinement levels with one update. Crank-Nicolson outperforms backward Euler in terms of accuracy. Crank-Nicolson performs significantly better with smaller time steps. While multiple updates do not bring much with backward Euler, it does help significantly for Crank-Nicolson. To reach the same accuracy, the number of time steps can be about four times smaller. We also plot the error vs CPU time in Fig. 15. Because a PKE parameter update is fairly costly with the current implementation, six updates make the calculation about 4 times slower than the calculations with one update thus making single update more attractive. We hope to reduce the cost per PKE parameter update in the future and make multiple PKE parameter updates per macro step a better option. Backward Euler and Crank-Nicolson almost have the same performance. It is noted that Crank-Nicolson gives the expected second order convergence. At the current stage, we recommend to use Crank-Nicolson with one PKE parameter update for all IQS calculations.

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.
Figure 15  Time convergence of the LRA benchmark.

3.6.1 Dump PKE Parameters for LRA Benchmark

For making the constant function a valid weighting function within the FEM framework, we first change the benchmark input slightly by switching the Dirichlet boundary condition to vacuum boundary condition. In the transient input file and the initial eigenvalue input file, we now have

```plaintext
[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:

```plaintext
[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, and these two steps can be skipped in these cases.

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

```plaintext
[Executioner]
type = IQS
do_iqsTransient = false
pkeParamCsv = 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. We intentionally to exercise another transfer as the following:

```plaintext
[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.

```plaintext
[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
[./]
```
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 constant weighting functions and $\lambda_i, \beta_i, i = 1, 2$ are constant in the fuel, the parameters dumped to file are identical to the parameters provided in the input file. The dynamic reactivity $\rho$ and the generation time $\Lambda$ are plotted in Fig. 16. We also plotted fits of the dumped PKE parameters. Power (normalized at initial)

![Graph showing dynamic reactivity and generation time](image)

Figure 16  PKE parameters of LRA benchmark.

history from the spatial kinetics calculation is plotted in Fig. 17a. Temperature history is plotted in Fig. 17b.

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

The PKE input file is very simple and given by

```
[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
```
We have initial equilibrium DNP concentration. It is noted that the generation time changes with time. The PKE solve actually gives $\frac{\Delta}{\Delta_0} n$. The normalized power history from this PKE calculation with the dumped PKE parameters is also plotted in Fig. 17a.

### 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

\[ \rho(2 \leq t \leq 3) = \rho_{rod}(t = 2) - \alpha_\rho(\sqrt{T} - \sqrt{T_0}), \quad (3) \]

\[ \Lambda(2 \leq t \leq 3) = \Lambda_{rod}(t = 2) - \alpha_\Lambda(\sqrt{T} - \sqrt{T_0}), \quad (4) \]

where \( \rho_{rod} \) and \( \Lambda_{rod} \) 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:

\[ \rho_{rod}(t) = \rho + \alpha_\rho(\sqrt{T} - \sqrt{T_0}) \]

\[ \Lambda_{rod}(t) = \Lambda + \alpha_\Lambda(\sqrt{T} - \sqrt{T_0}) \]

in Fig. 16. 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

\[ \rho_{rod}(0 \leq t \leq 2) = \sum_{i=1}^{3} \beta_{\rho,i}(1 - c(t))^i, \quad (5) \]

\[ \Lambda_{rod}(0 \leq t \leq 2) = \Lambda(t = 0) + \sum_{i=1}^{3} \beta_{\Lambda,i}(1 - c(t))^i, \quad (6) \]

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

\[ c(t) = \begin{cases} 1 - t/2, & 0 \leq t \leq 2 \\ 0, & t > 2 \end{cases} \quad (7) \]

The fitted coefficients are listed in Table 19.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \beta_{\rho,i} )</th>
<th>( \beta_{\Lambda,i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 7.65201825 \times 10^{-3} )</td>
<td>( 1.32442742 \times 10^{-6} )</td>
</tr>
<tr>
<td>2</td>
<td>( 1.04302712 \times 10^{-2} )</td>
<td>( 1.69547914 \times 10^{-6} )</td>
</tr>
<tr>
<td>3</td>
<td>( 6.18254986 \times 10^{-4} )</td>
<td>( -2.25423161 \times 10^{-7} )</td>
</tr>
</tbody>
</table>

The final reactivity function would be

\[ \rho(c, T) = \sum_{i=1}^{3} \beta_{\rho,i}(1 - c(t))^i - \alpha_\rho(\sqrt{T} - \sqrt{T_0}), \quad (8) \]

\[ \Lambda(c, T) = \Lambda(t = 0) + \sum_{i=1}^{3} \beta_{\Lambda,i}(1 - c(t))^i - \alpha_\Lambda(\sqrt{T} - \sqrt{T_0}). \quad (9) \]

They are plotted in Fig. 16 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 cause the peak power happen later.

If we integrate Eq. (1e) over the core, and divide by the core volume, we obtain

\[ \frac{\partial \bar{T}(t)}{\partial t} = \frac{\alpha}{\nu} \bar{F}(t) = \frac{\alpha}{\nu} \bar{P}_0 n(t) = \frac{\alpha}{\nu} \bar{P}_0 n = \frac{\alpha \bar{P}_0}{\kappa} n \quad (10) \]

where \( \bar{F} \) is the core-averaged fission reaction rate in unit of \( \text{fission cm}^{-3} \) and \( \bar{T} \) is the core-averaged temperature. \( \bar{P}_0 \) is the initial core-averaged fission reaction rate in unit of \( \text{fission cm}^{-3} \). The value of \( \frac{\alpha \bar{P}_0}{\kappa} \) is \( 1.19538077 \times 10^{-6} \) in unit of \( \frac{\text{K}}{s} \).
Now we have set up a PKE equations with the fitted reactivity and generation-time functions:

\[
\begin{align*}
\frac{\partial n(t)}{\partial t} &= \rho(c,T) - \frac{\beta}{\Lambda(c,T)} n(t) + \sum_{i=1}^{2} \lambda_i c_i(t) \\
\frac{\partial c_i(t)}{\partial t} &= \frac{\beta_i}{\Lambda(c,T)} n(t) - \lambda_i c_i(t), i = 1, 2, \\
\frac{\partial \bar{T}(t)}{\partial t} &= \frac{\alpha \tilde{P}_0}{\kappa} \frac{\Lambda_0}{\Lambda(c,T)} n(t),
\end{align*}
\]

with

\[
\begin{align*}
n(t = 0) &= 1, \\
\bar{T}(t = 0) &= T_0,
\end{align*}
\]

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 \( \beta \) and decay constants \( \lambda \) can be set with the initial condition. They will remain constant while no auxiliary kernels are added for them. Initial condition of generation time \( \Lambda \) 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'
[../]
```

103
We will need two auxiliary scalar kernels to evaluate the reactivity and generation time.

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. (8) and Eq. (9) 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.
Its initial condition is added inline.
Then we add an auxiliary variable for the coupling coefficient from the deposited power to the temperature.

```plaintext
[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:

```plaintext
[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. 17. 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 despite strong thermal feedback.

### 3.7 C5G7-2D using First Order NDA solver

This tutorial covers setting up Rattlesnake inputs for solving the two-dimensional C5G7 benchmark problem using the first-order formulation and nonlinear diffusion acceleration (NDA). All files necessary for running this tutorials with Rattlesnake are under `rattlesnake/tutorials/C5G72D/fiso-nda` folder.
3.7.1 Problem Description

The C5G7 benchmark is a MOX fueled, pressurized water reactor (PWR) minicore configuration. The C5G7 geometry depicted in Fig. 18 comprises a 2-by-2 array of UO$_2$ and MOX assemblies surrounded by reflector. In Fig. 18 vacuum and reflective boundary conditions are denoted by V and R, respectively. The fuel cells are not homogenized, each pin cell is comprised of a fuel pin, fission chamber, or guide tube surrounded by moderator as depicted in Fig. 19. 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 [12]. The energy group boundaries are provided in Table 20; three of the seven energy groups are fast, four are thermal. A detailed description of the two-dimensional C5G7 benchmark can be found in [12].

Figure 18  Geometry of the two-dimensional C5G7 benchmark problem. Boundary conditions are vacuum (V) or reflective (R).
3.7.2 Mesh Generation

A triangular unstructured mesh is created using the INSTANT mesh generator, [13], [14]. INSTANT reads a (hierarchical) geometry description in XML format and calls Triangle to triangulate the geometry. The input file instant_c5g7.xml for the INSTANT mesh generator is listed below:

```xml
<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>
      107
    </Pins>
  </Geometry>
</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.

3.7.3 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.

3.7.4 $S_N$ Mesh Block

The $S_N$ problem’s Mesh block is simple because it simply loads the INSTANT created Exodus mesh file.

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

```

3.7.5 $S_N$ 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 to indicate that the transport system is set up for updating angular fluxes for the full low-order diffusion solve. 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 (default), then computed cross sections will be NaN.

```plaintext
[TransportSystems]
  particle = neutron
  G = 7
  VacuumBoundary = 'vacuum'
```

```
ReflectingBoundary = 'reflecting'
equation_type = steady-state

scheme = DFEM-SN
family = MONOMIAL
order = FIRST
AQorder = 8
AQtype = Level-Symmetric
NA = 1
for_transport_update = true
initialize_angular_flux = true

3.7.6 Transport Materials

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

```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>2.00599E-02 2.02730E-03 1.57059E-02 4.51830E-02 4.33421E-02 2.02090E-01 5.25710E-01</NuFissionXS>
      <ChiXS>5.87910E-01 4.11176E-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 1.57099E-06 1.63140E-03 4.50940E-01 5.51630E-09</FissionXS>
      <Profile>
        1 1
        1 2
        1 3
        2 4
        6 5
        7 7
        5 7
      </Profile>
      <ScatteringXS>
        1.27537E-01 4.25780E-02 3.24456E-01
        9.43740E-06 1.63140E-03 4.50940E-01
        5.51630E-09 2.67920E-03 4.52565E-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 4.84472E-01 5.66922E-01 4.26732E-01 6.78997E-01 6.62852E-01</TotalXS>
      <NuFissionXS>2.17533E-02 2.53510E-03 1.26799E-03 5.45769E-03 2.07240E-03 6.64509E-01 5.13999E-01</NuFissionXS>
      <ChiXS>5.87910E-01 4.11176E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiXS>
      <FissionXS>7.62704E-03 8.76898E-04 6.45320E-03 1.57099E-06 1.63140E-03 4.50940E-01 5.51630E-09</FissionXS>
      <Profile>
        1 1
        1 2
        1 3
        2 4
        6 5
        7 7
      </Profile>
      <ScatteringXS>
        1.28876E-01 4.14130E-02 3.24456E-01
        8.22900E-06 1.63140E-03 4.51830E-01
        5.04050E-09 2.67920E-03 4.57173E-01 1.60460E-04
        5.53940E-03 2.76814E-01 2.00510E-03
        9.31270E-03 2.52962E-01 8.49840E-03
        9.16560E-09 1.48500E-02 2.65070E-01
      </ScatteringXS>
    </material>
  </Macros>
</Materials>
```
<material ID="3" NA="0" fissile="true">
  <name>7.0 percent MOX fuel-clad</name>
  <TotalXS>1.81323E-01 3.34368E-01 4.93600E-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>
</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>
</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 1.17214E+00</TotalXS>
  <Profile>
    1 1 1 2 1 3 1 5 1 6 2 7 2 7
  </Profile>
</material>

<material ID="6" NA="0" fissile="true">
  <name>fission chamber</name>
  <TotalXS>1.26032E-01 2.93160E-01 2.84240E-01 2.80960E-01 3.34440E-01 5.65640E-01 1.17214E+00</TotalXS>
  <Profile>
    1 1 1 2 1 3 1 5 1 6 2 7 2 7
  </Profile>
</material>
It is noted that the benchmark provides the averaged neutrons emitted per fission $\nu$ and fission cross section $\Sigma_f$, $g = 1, \cdots, 7$. INSTANT format requires $\nu \Sigma_f$, which needs more precision than 6 to keep the data consistent. We also provide fake $\kappa \Sigma_f$ 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
  library_file = ../c5g7_materials.xml
  material_id = 1
[../]
[./mox4.3]
  type = ConstantNeutronicsMaterial
  block = 'M-2-TRI'
  fromFile = true
  library_file = ../c5g7_materials.xml
  material_id = 2
[../]
[./mox7.0]
  type = ConstantNeutronicsMaterial
  block = 'M-3-TRI'
  fromFile = true
  library_file = ../c5g7_materials.xml
  material_id = 3
[../]
[./mox8.7]
  type = ConstantNeutronicsMaterial
  block = 'M-4-TRI'
  fromFile = true
  library_file = ../c5g7_materials.xml
  material_id = 4
[../]
[./GuideTube]
  type = ConstantNeutronicsMaterial
  block = 'M-5-TRI'
  fromFile = true
  library_file = ../c5g7_materials.xml
  material_id = 5
[../]
[./FissionChamber]
  type = ConstantNeutronicsMaterial
  block = 'M-6-TRI'
  fromFile = true
  library_file = ../c5g7_materials.xml
  material_id = 6
[../]
[./moderator]
  type = ConstantNeutronicsMaterial
  block = 'M-7-TRI'
  fromFile = true
  library_file = ../c5g7_materials.xml
  material_id = 7
[../]
[]

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

[GlobalParams]
  fromFile = true
  library_file = ../c5g7_materials.xml
[]
3.7.7 $S_N$ 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.
We also add a postprocessor to monitor the CPU time during the nonlinear Picard iteration. \textit{PerformanceData} is one of MOOSE postprocessors.

3.7.8 \textit{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. \texttt{xdiff} takes the name of a postprocessor for evaluating the progress of the iteration; in this case the postprocessor defined in 3.7.7 is used. Note: The first order \textit{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
[]
```

3.7.9 \textit{S_N} Outputs Block

Nothing special needs to be considered in the Outputs block. \texttt{csv} output is set to true to record the convergence history.

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

3.7.10 Diffusion Mesh Block

NDA allows the diffusion mesh to be coarser than the \textit{S_N} mesh as long as it is nested within it. A mesh is nested in another mesh if each \textit{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 \textit{S_N} meshes, while coarse mesh (cm) uses a structured quadrilateral mesh for the diffusion calculation where each element comprises a single pin cell. Note: for this to work the \textit{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:

```
3.7.11 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 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:

```plaintext
[TransportSystems]
particle = neutron
equation_type = eigenvalue
G = 7
VacuumBoundary = 'vacuum'
ReflectingBoundary = 'reflecting'
[./diff_dfem]
scheme = DFEM-Diffusion
family = MONOMIAL
order = FIRST
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 1 1 1'
DGtype = IIP
[../]
[/]
```

Coarse mesh TransportSystems block:

```plaintext
[TransportSystems]
particle = neutron
equation_type = eigenvalue
G = 7
VacuumBoundary = '0 3'
ReflectingBoundary = '1 2'
```
3.7.12 Diffusion Materials Block

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

3.7.13 Diffusion Executioner Block

The PicardEigen executioner is used that triggers a Picard iteration scheme for the solution of the NDA eigenvalue equations. The parameter picard_max_its determines the maximum number of Picard iterations and wrapped_app_tol is the convergence tolerance that the iterative error given by the sub app postprocessor specified in section 3.7.7.

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 coarse 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:

```plaintext
[Executioner]
  type = PicardEigen

  # NDA params
  picard_max_its = 50
  wrapped_app_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

  petsc_options_iname = '-pc_type -ksp_gmres_restart -sub_pc_type -pc_asm_overlap'
  petsc_options_value = 'asm 50 1 lu 1'
```

118
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

  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'

3.7.14 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 enabled for visualization.

[Outputs]
  print_linear_residuals = true
  exodus = true
  csv = true

3.7.15 Executing the Input
The two calculations with coarse and fine meshes can be executed as

```
../../../rattlesnake-opt -i c5g7_master_cm.i
../../../rattlesnake-opt -i c5g7_master_fm.i
```

### 3.7.16 Results

Results for this tutorial can be found in Ref. [15] and [16].

### 3.8 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.

#### 3.8.1 Problem Description

Refer to Section 3.7.1.

#### 3.8.2 Mesh Generation

Refer to Section 3.7.2.

#### 3.8.3 Transport Materials

Refer to Section 3.7.6.

#### 3.8.4 Postprocessing and Outputs

We use the following inputs to generate the flux map.

```plaintext
[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. In the remainder of this tutorial we will use the transport system name ‘saaf’, but the user is allowed to use their own preferred names. It is noted that FluxCartesianCoreMap currently does not support distributed mesh. If we turn on the distributed mesh in command line with ‘–distributed-mesh’, the calculation will be performed normally but the flux map will be messed up due to the scrambled IDs of elements.

We also cares the run-time with:

```
120
```
PerformanceData is one of MOOSE postprocessors. The outputs is simple as

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

### 3.8.5 Direct Transport Solve with SAAF-SN-CFEM

To perform a calculation with NDA, we first want to create an input for direct transport solve. The input can be run stand-alone to provide a reference transport solution and be used as a template for an input file for transport update. What we need other than previous sections are two blocks.

One is for the transport system:

```plaintext
[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

AQtype = Bickley3-Optimized
NPolar = 2
NAzimuthl = 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 disables the void treatment.

The executioner block is given by:
We want to use BoomerAMG as the preconditioner because it is more parallel scalable with domain decomposition.

### 3.8.6 Converting the Direct Transport Solve for Transport Update

We essentially need to do four things:

1. Set `equation_type` in `TransportSystems` block to `steady-state`;
2. Set `for_transport_update` in `TransportSystems/saaf` block to `true`;
3. Change the executioner to:

   ```
   [Executioner]
   type = AMGUpdate
   amg_tol = 1e-3
   debug = false
   
   [Outputs]
   exodus = false
csv = true
   print_perf_log = true
   
   and remove the user object for flux map. We do not care about 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.

### 3.8.7 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 change the `TransportSystems` block to:
We basically switch the discretization scheme from SAAF-CFEM-SN to CFEM-Diffusion. And we tell the diffusion scheme the application wrapper, that are used for providing drift vectors and vacuum boundary coefficients on the quadrature points, with transport_wrapper. The value of the transport_wrapper is simply the file name for the transport update.

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

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
3.8.8 Results

With TransportSystems/saaf/NPolar=4 and TransportSystems/saaf/NAzmthl=16 and 72 cores with 2 nodes on Falcon, we got the following Picard convergence

```
wrapped_app: +----------------+----------------+----------------+----------------+
wrapped_app: | time | flux_error | flux_error_lo | runtime |         |
wrapped_app: +----------------+----------------+----------------+----------------+         |
wrapped_app: | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |         |
wrapped_app: | 1.000000e+00 | 1.969775e-01 | 1.969775e-01 | 1.882939e+02 |         |
wrapped_app: | 2.000000e+00 | 3.426955e-02 | 8.199061e-02 | 1.882939e+02 |         |
wrapped_app: | 3.000000e+00 | 5.279768e-03 | 3.753288e-02 | 2.563198e+02 |         |
wrapped_app: | 4.000000e+00 | 3.535617e-03 | 1.213774e-02 | 3.232737e+02 |         |
wrapped_app: | 5.000000e+00 | 7.478282e-04 | 1.238493e-03 | 3.923014e+02 |         |
wrapped_app: | 6.000000e+00 | 1.319686e-04 | 8.410504e-04 | 4.603781e+02 |         |
wrapped_app: | 7.000000e+00 | 8.789139e-05 | 3.030952e-04 | 5.280636e+02 |         |
wrapped_app: | 8.000000e+00 | 1.015141e-07 | 9.413686e-07 | 5.966746e+02 |         |
wrapped_app: | 9.000000e+00 | 1.015141e-07 | 6.420881e-07 | 8.702829e+02 |         |
wrapped_app: +----------------+----------------+----------------+----------------+         |
```

where time is actually considered as iteration index by Rattlesnake. flux_error and flux_error_lo are the two post-processors on the high-order transport system added by Rattlesnake for monitoring the convergence of the Picard iteration. flux_error is max$_{1 \leq g \leq G} \| \phi_g^{(n+1/2)} - \phi_g^{(n)} \|$ and flux_error_lo is max$_{1 \leq g \leq G} \| \phi_g^{(n-1)} - \phi_g^{(n)} \|$. where $\phi_g^{(n)}$ is the flux of energy group $g$ after the $n$-th diffusion solve and $\phi_g^{(n+1/2)}$ is the flux of energy group $g$ after the $n$-th transport update. Because the low-order diffusion system is completely consistent with the high-order transport system, flux_error keeps dropping. Saturation of flux_error typically means the inconsistency between the two systems. On the other hand, flux_error_lo should be always dropping, which otherwise means un converged Picard iterations. The calculation can be done in about 14.5min. In 7 Picard iterations, error in k-eff drops bellow 1pcm (1.18569). If we use this as a convergence criteria, the calculation can be done in about 8min. More results of the convergence study for this tutorial can be found in Ref. [17].

It is noted that we can simply change the discretization scheme in high-order input file from SAAF-CFEM-SN to LS-CFEM-SN to perform NDA calculations with least-square methods. If we chose the weight_type = 1/\sigma_t and weak_bc_type = 7 in the LS-CFEM-SN scheme, i.e. using the weighted least-square method, we will get the expected exactly same solution as SAAF-CFEM-SN. We can set TransportSystems/saaf/tau=0.5 in the high-order input file to activate the void treatment. This, however, requires changing two parameters for the underlining BoomerAMG solver used by AMGUpdate executioner. One is by setting Executioner/amg_strong_threshold equal to 0.9 (default 0.7). Another is by adding an extra PETSc parameter in the Executioner block:

```
petsc_options_iname = '-pc_hypre_boomeramg_no_CF'
petsc_options_value = 'true'
```
3.9 Coupled reactor calculation

3.9.1 Problem description

Geometry of this tutorial problem is illustrated in Fig. 20. Sizes of all regions are marked on the figure and are in units of centimeter. The region featuring the fast spectrum is surrounded by a thermal neutron filter only allowing leakage of neutrons with thermal energies from the fast to the thermal region. The 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 21. All cross sections in the table are in units of 1/cm. The group averaged neutron velocities are constant for all four regions. The fast velocity is 4.34311E+09 cm/s and the thermal velocity is 2.82595E+06 cm/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 present 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.

![Figure 20 Geometry of the coupled reactor.](image)

<table>
<thead>
<tr>
<th>Region</th>
<th>( g )</th>
<th>( \Sigma_{t,g} )</th>
<th>( \Sigma_{s,g} )</th>
<th>( \Sigma_{g,0,g'\rightarrow g} )</th>
<th>( \Sigma_{g,1,g'\rightarrow g} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td>1</td>
<td>3.79586E-01</td>
<td>3.4788E-01</td>
<td>0.0000E+00</td>
<td>1.0628E-01</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.41032E-01</td>
<td>3.8229E-09</td>
<td>3.1800E-01</td>
<td>0.8246E-09</td>
</tr>
<tr>
<td>Thermal</td>
<td>1</td>
<td>5.68247E-01</td>
<td>5.1603E-01</td>
<td>0.0000E+00</td>
<td>3.1445E-01</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.88134E+00</td>
<td>5.0241E-02</td>
<td>1.7937E+00</td>
<td>0.9285E-02</td>
</tr>
<tr>
<td>Filter</td>
<td>1</td>
<td>3.16675E-01</td>
<td>2.4923E-01</td>
<td>0.0000E+00</td>
<td>3.9400E-02</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.90484E+01</td>
<td>2.8582E-11</td>
<td>2.0710E+00</td>
<td>-0.7873E-12</td>
</tr>
<tr>
<td>Reflector</td>
<td>1</td>
<td>3.77860E-01</td>
<td>3.7174E-01</td>
<td>0.0000E+00</td>
<td>2.8487E-02</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.90661E-01</td>
<td>6.1056E-03</td>
<td>4.9052E-01</td>
<td>-1.4008E-03</td>
</tr>
</tbody>
</table>

Table 21 Cross sections of the coupled reactor.
3.9.2 The stand-alone input file

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

```plaintext
[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
```

126
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'
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 postprocessor to obtain the integral of the fast flux over the entire domain for a sanction check that will be 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 using the relation $\frac{1}{3(\Sigma_{t,1} - \Sigma_{s,1})}$. Removal cross sections are evaluated with $\Sigma_{t,1} - \sum_{g'=1}^{G} \Sigma_{s,0,\rightarrow g'}$. The remaining input parameters are fairly straightforward 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.

```plaintext
[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.

```plaintext
[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. \textit{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 \textit{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 \textit{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 \textit{Richardson} executioner instead the \textit{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
[

129
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 material’s parameter `disable_fission` 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
******************************************************

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>rho</th>
<th>Lambda</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.66447997</td>
<td>8.23966654e-09</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.11528978</td>
<td>3.97927783e-08</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.54174692</td>
<td>4.64094897e-06</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.81384753</td>
<td>5.32602341e-06</td>
</tr>
</tbody>
</table>
```

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

3.10 Restarting a Rattlesnake Calculation

Rattlesnake supports several modes of restarting:

1. Restarting a stand-alone eigenvalue or transient calculation.
2. Restarting an eigenvalue calculation that is used to establish the initial steady state of a transient calculation.
3. Loading the transport system’s variables from an Exodus file.

It is noted that restart is different from recover. Restart can be considered as a way of setting initial conditions with the data from a previous simulation. The latter, on the other hand, by definition is resuming an existing simulation either due to a fault or other premature termination, which is typically done through using the `--recover` command-line flag. Example inputs can be found under `test/tests/restart` folder.

3.10.1 Restarting a stand-alone eigenvalue & transient calculation

More detailed information about this can be found in MOOSE documentation at [http://www.mooseframework.org/wiki/R restart/]. This section adds that eigenvalue calculation can be restarted in the same way as the transient calculations. This syntax is directly inherited from MOOSE’s restart capabilities. In order for this mode to be available, the user needs to enable check-points in the original input file. This is done by simply specifying:

```
[Outputs]
   ...  
   checkpoint = true
   ...
```

You can create a subblock in the input file that will allow you to change the file format, suffix, frequency of output, the number of checkpoint files to keep, etc. After running an input file with checkpoint enabled, a folder will be created whose name ends on "_cp". This folder contains the restart files.

It is very easy to change a standard input file to a restarted input file. The first change that needs to be made is to change the Mesh block to read the correct mesh file that is saved in the checkpoint folder:

```
[Mesh]
   file = eigen_out_cp/0002_mesh.cpr
```

131
In this case, the folder’s name is "eigen_out_cp", in which "eigen_out" is the output file base. "0002" is the last checkpoint file generated by the original eigenvalue calculation. The second change is to add a Problem block specifying the restart_file_base:

```
[Problem]
  restart_file_base = eigen_out_cp/0002
[]
```

Instead of specifying the number 0002 the user can also use the keyword "LATEST" picking the largest serial number:

```
[Problem]
  restart_file_base = eigen_cp_cp/LATEST
[]
```

### 3.10.2 Restarting an eigenvalue calculation before a transient calculation

In this case, the transient calculation is the master app and the eigenvalue calculation is the sub app. In this mode we only restart the sub app while the master app starts normally, i.e. no restart. To achieve this, the transient input file is not changed, and the eigenvalue (sub app) restart file is set up for restart. The only difference to a standard restart is to use the parameter force_restart in the Problem block:

```
[Mesh]
  file = transient_out_initial_solve0_cp/0002_mesh.cpr
[]

[Problem]
  restart_file_base = transient_out_initial_solve0_cp/LATEST
  force_restart = true
[]
```

An example can be found in /test/tests/restart/restart_transient/transient_by_sub.i and /test/tests/restart/restart_transient/restart_eigen.i. The parameter force_restart is required whenever only the MultiApp subtree below a certain sub app is restarted but not the master and/or potential sister branches.

### 3.10.3 Loading transport solution from Exodus file

The calculation from which to restart needs to set the exodus flag equal to true and ensure that primal variables are not hidden, i.e. ensure that for example:

```
[TransportSystems]
  [./ts]
    hide_angular_flux = false
  [../]
[]

[Outputs]
  exodus = true
[]
```
If the Exodus file written by this calculation is named "eigen_out.e", then the mesh and transport system blocks should contain the following entries:

```plaintext
[Mesh]
  type = FileMesh
  file = eigen_out.e
[

[TransportSystems]
  restart_transport_system = true

[./ts]
  scheme = ...
  ...
[../]
[
```

Note in particular that `restart_transport_system` triggers the transport variables to be read from the Exodus file. This example is from `/yak/tests/restart/restart_transient_from_eig/start_transient_from_eig.i`.

Warning: This feature does not work for discontinuous shape functions of order greater than or equal to one. The reason is that Exodus format does not support high-order elemental variables and Rattlesnake saves these variables as continuous FEM shape functions typically. Also this feature can only restart the transport system. Variables of other physics in the same input file will not be restarted.

3.11 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.)

3.12 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.

Note that advanced parameters for which a concise explanation would require some theoretical background are only mentioned in this user manual and detailed in the Rattlesnake theory manual.

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/$G$
Note: $G$ must be greater than 0.

4.6 scaling_eigenkernels

Description: The factor used for dividing the eigen kernels
Data type: Real
Default value: 1.0
Syntax: TransportSystems/scaling_eigenkernels
Note: This parameter is active only for non-eigenvalue calculations. Its value must be greater than 0.

4.7 show_interaction_with_material_executioner

Description: Whether or not to show how the transport system interacts with materials and executioner
Data type: Logical
Default value: false
Syntax: TransportSystems/show_interaction_with_material_executioner

4.8 restart_transport_system

Description: Whether or not to initialize the primal variables from the provided Exodus file.
Data type: Logical
Default value: false
Syntax: TransportSystems/restart_transport_system
Note: The mesh must be a FileMesh. If the provided Exodus file comes from an eigenvalue calculation, the computed eigenvalue is automatically read from the file but the corresponding global variable holding that value must be called `eigenvalue`. For other cases (e.g. if the FileMesh is not an Exodus or if the Exodus file does not hold such a global variable), the eigenvalue must be manually set using scaling_eigenkernels to start with a self-sustaining initial condition (or null-transient). If scaling_eigenkernels is provided by the user, the eigenvalue will not be read from the FileMesh.

## 4.9 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.

### 4.9.1 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.

### 4.9.2 DirichletFunction

Description: Transport functions to be set on the Dirichlet boundaries

Data type: Vector of function names (or vector of real)

Default value: <empty>

Syntax: TransportSystems/DirichletFunction

Note: Two options are given to the user to specify the boundary source function. The first one is to provide one TransportFunction per side set in DirichletBoundary. The second – designed for constant sources – is to provide a number of values equal to the number of side sets in DirichletBoundary times G. The values for a given side set are grouped together and separated with commas (for example, if DirichletBoundary = 'left right' and DirichletFunction = '1,1 0,0', both groups will have the same boundary condition, 1 and 0 on the left and right boundaries, respectively). The functions thus created are later referred to as in-line transport functions.

When using the first option, make sure that the provided functions actually derive from TransportFunction, or else the code will crash while retrieving the input without a very self-explanatory error message.

### 4.9.3 VacuumBoundary

Description: the vacuum or surface source boundaries

Data type: Vector of boundary names

Default value: <empty>
4.9.4  **SurfaceSource**

Description: Transport functions to be set on the vacuum boundaries
Data type: Vector of function names (or vector of real)
Default value: `<empty>`

Syntax: `TransportSystems/SurfaceSource`

Note: Two options are given to the user to specify the boundary source function. The first one is to provide one `TransportFunction` per side set in `VacuumBoundary`. The second – designed for constant sources – is to provide a number of values equal to the number of side sets in `VacuumBoundary` times `G`. The values for a given side set are grouped together and separated with commas (for example, if `VacuumBoundary = 'left right'` and `SurfaceSource = '1,1 0,0'`, both groups will have the same boundary condition, 1 and 0 on the left and right boundaries, respectively). The functions thus created are later referred to as in-line transport functions.

When using the first option, make sure that the provided functions actually derive from `TransportFunction`, or else the code will crash while retrieving the input without a very self-explanatory error message.

4.9.5  **ReflectingBoundary**

Description: Specular reflecting boundaries
Data type: Vector of boundary names
Default value: `<empty>`

Syntax: `TransportSystems/ReflectingBoundary`

4.9.6  **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`.

4.10  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.
4.10.1 *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

4.10.2 *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.

4.10.3 *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 of type TransportFunction. Only LS-CFEM-PN, DFEM-SN and DFEM-PN are currently using this parameter.

4.10.4 *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 entries needs to be consistent with the dimension of the mesh. Only SAAF-CFEM-PN is currently using this parameter.

4.10.5 *PointSourceValue*

Description: Strength of the volumetric point source
Data type: Vector of real
Default value: `<empty>`

4.11 Multischeme transport

Rattlesnake supports multischeme transport simulations. For multischeme 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. It is noted that if users do not provide a custom partitioner for parallel calculations with multiple processors, Rattlesnake will add a default partitioner that assumes the load on elements is proportional to the number of primal variables (number of groups for diffusion, number of groups times number of streaming directions for SN and number of groups times number of angular flux moments for PN) and tries balancing the load of all processors based on this assumption.

4.11.1 multischeme_option

Description: Options for multischeme calculations

Data type: Enumeration (/mortar-split/mortar/penalty/)

Default value: mortar-split

Syntax: TransportSystems/multischeme_option

Note: If users are providing a conforming mesh for the multischeme simulation with more than one subdomain, this parameter must be mortar-split or penalty. If the mesh has been properly split for the multischeme simulation, this parameter must be mortar but this option is not currently supported.

4.11.2 show_multischeme_actions

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

Data type: Logical

Default value: False

Syntax: TransportSystems/show_multischeme_actions

4.11.3 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

Note: This parameter is valid only for mortar multischeme.
4.11.4  *evaluate_mortar_aux*

Description: Whether or not to evaluate mortar auxiliary variables

Data type: Logical

Default value: false

Syntax: TransportSystems/evaluate_mortar_aux

Note: This parameter is valid only for mortar multischeme with $S_N$ as the master scheme.

4.11.5  *strict_equal_angular_flux*

Description: True to add equal value angular flux constraint for mortar SN-SN

Data type: Logical

Default value: false

Syntax: TransportSystems/strict_equal_angular_flux

Note: This parameter is valid only for mortar multischeme coupling $S_N$ and $S_N$. This parameter will ensure that the angular fluxes on the two side of interface for any streaming direction are exactly equal for mortar SN-SN, even if the interface is parallel with the streaming direction. This parameter must be turned on for multischeme nonlinear diffusion acceleration (NDA) calculations.

4.11.6  *decouple_mortar_coupling*

Description: True to decouple the interface coupling for mortar SN-PN

Data type: Logical

Default value: false

Syntax: TransportSystems/decouple_mortar_coupling

Note: This parameter is valid only for mortar multischeme coupling $S_N$ and $P_N$ and coupling $S_N$ and $S_N$. This parameter will cause the off-diagonal Jacobian due to the mortar SN-PN to be not assembled and will cause the diagonal Jacobian of the mortar variables to be non-zero. As a result, the Jacobian will no longer have the so-called saddle pattern, i.e. zero entries on diagonal.

4.11.7  *Riemann_interface_upwind*

Description: Perform eigenstructure based or original interface upwind for PN

Data type: Logical

Default value: false

Syntax: TransportSystems/Riemann_interface_upwind

Note: Advanced parameter. Users should use default unless they really know what they are doing. It is only for penalty multischeme coupling $P_N$ and $P_N$. 

140
4.11.8  *diffusion_to_sn_penalty_factor*

Description: The factor for better interfacing diffusion and SN with penalty method

Data type: Real

Default value: 0.5

Syntax: TransportSystems/diffusion_to_sn_penalty_factor

Note: Advanced parameter. Users should use default unless they really know what they are doing. It is only for penalty multischeme coupling $S_N$ and diffusion. It must be greater than or equal to 0.2 and less than or equal to 1.2. This free parameter is used to adjust the coupling between SN and diffusion with the penalty method. 0.5 corresponds to the transport-consistent coefficient while 1.0 is preferred if the problem is diffusive.

4.11.9  *interface_diffusion_penalty*

Description: Penalty type for interface diffusion

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

Default value: SIP

Syntax: TransportSystems/interface_diffusion_penalty

Note: Advanced parameter. Users should use default unless they really know what they are doing. It is only for penalty multischeme coupling diffusion and diffusion.

4.11.10  *sigma*

Description: Diffusion penalty coefficient

Data type: Real

Default value: 4.0

Syntax: TransportSystems/sigma

Note: This parameter must be greater than 0. Advanced parameter. Users should use default unless they really know what they are doing. It is only for penalty multischeme coupling diffusion and diffusion.

4.11.11  *multischeme_partitioning_package*

Description: Partitioning option for multischeme with element weighting

Data type: Enumeration (/parmetis/hierarch/noweighting/)

Default value: parmetis

Syntax: TransportSystems/multischeme_partitioning_package

Note: Computing loads on elements could vary significantly across schemes. Rattlesnake thus applies different weightings for different schemes when partitioning the domain. Rattlesnake uses the number of primal variables of transport schemes as the element weighting. Users can use this parameter to turn off the weighting (noweighting) or switch packages for partitioning the domains with the weighting. If users provide a custom partitioner in the input through Mesh/Partitioner syntax, Rattlesnake honors the custom partitioner and this parameter becomes inactive.
4.12 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 multischeme transport capability.

4.12.1 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.

4.12.2 Neutron

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

4.12.2.1 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. If the number of delayed groups in the material data is greater than zero, it must be equal to this number.

4.12.2.2 fission_source_as_material

Description: Fission source is treated as a material property (true) or an auxiliary variable (false)

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.

### 4.12.2.3 `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`

Note: This parameter only has effect on transient calculations.

### 4.12.2.4 `dnp_integration_scheme`

Description: Integration scheme for DNP materials.

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

Default value: exp

Syntax: `TransportSystems/*/dnp_integration_scheme`

### 4.12.2.5 `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.

### 4.12.2.6 `assemble_fission_jacobian`

Description: True to assemble fission Jacobian.

Data type: Logical

Default value: false

Syntax: `TransportSystems/*/assemble_fission_jacobian`

Note: This parameter only has effect on diffusion calculations.
4.12.2.7 assemble_delay_jacobian

Description: True to assemble Jacobian from delayed kernels
Data type: Logical
Default value: false
Syntax: TransportSystems/*/assemble_delay_jacobian
Note: This parameter only has effect on transient diffusion calculations.

4.12.3 Thermal

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

4.12.3.1 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 in ascending order.

4.12.3.2 setup_temperature_equation

Description: True to set up the temperature equation
Data type: Logical
Default value: True
Syntax: TransportSystems/*/setup_temperature_equation

4.12.3.3 T_family

Description: Family of FE shape functions for temperature
Data type: Enumeration (refer to MOOSE syntax about Variables)
Default value: LAGRANGE
Syntax: TransportSystems/*/T_family
Note: This parameter is activated only when setup_temperature_equation is true.
4.12.3.4  \textit{T\_order}

Description: Order of FE shape functions for temperature
Data type: Enumeration (refer to MOOSE syntax about Variables)
Default value: FIRST
Syntax: TransportSystems/*/T\_order
Note: This parameter is activated only when \textit{setup\_temperature\_equation} is true.

4.12.3.5  \textit{T\_option}

Description: Heat equation primal variable option
Data type: enumeration (/T/T4/acT4/)
Default value: T
Syntax: TransportSystems/*/T\_option
Note: This parameter is activated only if \textit{setup\_temperature\_equation} is true.

4.12.3.6  \textit{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 \textit{setup\_temperature\_equation} is true.

4.12.3.7  \textit{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 \textit{setup\_temperature\_equation} is true.

4.12.3.8  \textit{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 \textit{setup\_temperature\_equation} is true.
4.12.3.9  *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.

4.12.3.10  *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.

4.12.3.11  *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.

4.12.3.12  *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.

4.12.3.13  *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.

4.12.3.14 use_cgs

Description: True to use cgs otherwise metric units
Data type: Logical
Default value: False
Syntax: TransportSystems/*/use_cgs

4.12.4 CFEM-Diffusion

This discretization scheme supports X-Y-Z, R-spherical and R-Z coordinates. 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 worry about them.

4.12.4.1 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.

4.12.4.2 family

Description: Family of FE shape functions for primal variables
Data type: Enumeration (refer to 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.

4.12.4.3 order

Description: Order of FE shape functions for primal variables
Data type: Enumeration (refer to MOOSE syntax about Variables)
Default value: FIRST
Syntax: TransportSystems/*/order
4.12.4.4  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 numbering 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.

4.12.4.5  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.

4.12.4.6  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
Note: This parameter does and should not change the calculation. It is used by Rattlesnake for switching between two implementations of the scattering kernels. Experience shows that setting this parameter to true can reduce the run-time slightly.

4.12.4.7  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.
4.12.4.8  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

4.12.4.9  fixed_jacobian

Description: To signal Rattlesnake that 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.

4.12.4.10  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 screen output is provided. 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.

4.12.4.11  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.
4.12.4.12  

**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.

4.12.4.13  

**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.

4.12.4.14  

**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.

4.12.4.15  

**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 when 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.
4.12.4.16 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

4.12.4.17 diffusion_kernel_type

Description: Type of diffusion kernel used
Data type: Enumeration (/scalar/vector/tensor/)
Default value: scalar
Syntax: TransportSystems/*/diffusion_kernel_type

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

\[
D = \begin{cases} 
D_s, & \text{scalar} \\
\frac{1}{3} \sum_{i=1}^{3} D_i, & \text{vector} \\
\frac{1}{3} \sum_{i=1}^{3} \sum_{j=1}^{3} D_{ij}, & \text{tensor} 
\end{cases}
\]  \hspace{1cm} (16)

\[
D_i = \begin{cases} 
D_i, & \text{scalar} \\
\sum_{j=1}^{3} D_{ij}, & \text{vector} \\
\sum_{j=1}^{3} \sum_{k=1}^{3} D_{ijk}, & \text{tensor} 
\end{cases}
\]  \hspace{1cm} (17)

\[
D_{ij} = \begin{cases} 
D_{ij}, & \text{scalar} \\
D_{ij}, & \text{vector} \\
D_{ij}, & \text{tensor} 
\end{cases}
\]  \hspace{1cm} (18)

NDA using the Application wrapper needs to use tensor (will be set automatically)

4.12.4.18 diffusion_coefficient_scheme

Description: Source of the diffusion coefficient, if supplied by input, calculated from the total cross section or supplied from a transport calculation
Data type: Enumeration (/user-supplied/local/nonlocal/)
Default value: user-supplied
Syntax: TransportSystems/*/diffusion_coefficient_scheme

Note: NDA calculations use local per default if not specified otherwise. If a nonlocal_diffusion_multiapp_file is supplied, this parameter will be set to nonlocal, if not set by input.

4.12.4.19 nonlocal_diffusion_multiapp_file

Description: Input file for the transport solver for the nonlocal diffusion coefficient
Data type: Input File
Default value: <empty>
4.12.4.20  **initialize_iqs_adjoint_flux**

Description: True to initialize adjoint fluxes with one for IQS
Data type: Logical
Default value: False
Syntax: TransportSystems/*/initialize_iqs_adjoint_flux
Note: This parameter is only used for IQS executioner. Setting it to true can provide a constant adjoint fluxes as the weighting functions for IQS.

4.12.4.21  **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 [18] for more explanations.

4.12.4.22  **transport_consistent_scalar_flux_name**

Description: True to use transport consistent scalar flux names
Data type: Logical
Default value: False
Syntax: TransportSystems/*/transport_consistent_scalar_flux_name
Note: This parameter changes the names of the scalar flux variables to be consistent with their names for $P_N$ or $S_N$ calculations.

4.12.4.23  **p1_consistent_output**

Description: True to add the currents as auxiliary variables
Data type: Logical
Default value: False
Syntax: TransportSystems/*/p1_consistent_output
Note: For the time being, the currents are saved as CONSTANT MONOMIAL (i.e. piecewise constant) auxvariables regardless of the order and family of the scalar flux. This is because mesh formats such as Exodus do not support high order discontinuous variables. The names of the currents follow the naming convention in transport calculations, i.e. \textit{flux\_moment\_g\{g\}\_L1\_M\{m\}} , where \{g\} is the group index and \{m\} is the secondary spherical harmonics index (0/1/-1/ correspond to x/y/z). When this parameter is true, Rattlesnake expects \textit{transport\_consistent\_scalar\_flux\_name} to be true as well, otherwise it will issue a warning about the inconsistent naming convention of scalar fluxes and currents. In particular, when the scalar fluxes and currents are consistent with transport naming, the solution can be loaded by a \texttt{FilePNTransportFunction} with \texttt{max\_moment\_degree\_to\_read}=1 and reconstructed with a linearly anisotropic representation.

4.12.5 DFEM-Diffusion

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

- The default value for \textit{family} is changed to \textit{L2\_LAGRANGE}.
- \textit{transport\_wrapper} is not valid.
- there are more parameters: \textit{DGType, penalty, transport\_multiapp\_file} and \textit{NDA\_type}.

This discretization scheme supports X-Y-Z, R-spherical and R-Z coordinates.

4.12.5.1 \textit{block}

Refer to \textit{block} in CFEM-Diffusion.

4.12.5.2 \textit{family}

Refer to \textit{family} in CFEM-Diffusion.

Note: The default value for \textit{family} is changed to \textit{L2\_LAGRANGE}.

4.12.5.3 \textit{order}

Refer to \textit{order} in CFEM-Diffusion.

4.12.5.4 \textit{group\_collapsing}

Refer to \textit{group\_collapsing} in CFEM-Diffusion.

4.12.5.5 \textit{group\_weights}

Refer to \textit{group\_weights} in CFEM-Diffusion.
4.12.5.6  *collapse_scattering*

Refer to *collapse_scattering* in CFEM-Diffusion.

4.12.5.7  *balance_table*

Refer to *balance_table* in CFEM-Diffusion.

4.12.5.8  *balance_table_on*

Refer to *balance_table_on* in CFEM-Diffusion.

4.12.5.9  *fixed_jacobian*

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.12.5.10  *verbose*

Refer to *verbose* in CFEM-Diffusion.

4.12.5.11  *prefix*

Refer to *prefix* in CFEM-Diffusion.

4.12.5.12  *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.12.5.13  *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.12.5.14  *save_residual*

Refer to *save_residual* in CFEM-Diffusion.

4.12.5.15  *assemble_scattering_jacobian*

Refer to *assemble_scattering_jacobian* in CFEM-Diffusion.

4.12.5.16  *vacuum_extrapolation_factor*

Refer to *vacuum_extrapolation_factor* in CFEM-Diffusion.
4.12.5.17  diffusion_kernel_type

Refer to diffusion_kernel_type in CFEM-Diffusion.

4.12.5.18  diffusion_coefficient_scheme

Refer to diffusion_coefficient_scheme in CFEM-Diffusion.

4.12.5.19  initialize_iqs_adjoint_flux

Refer to initialize_iqs_adjoint_flux in CFEM-Diffusion.

4.12.5.20  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

4.12.5.21  penalty

Description: Penalty function type
Data type: Enumeration (/std/)
Default value: std
Syntax: TransportSystems/*/penalty

4.12.5.22  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.

4.12.5.23  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.

4.12.6 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, larsen_trahan and larsen_trahan_mode. This discretization scheme supports X-Y-Z, R-Z cylindrical and R-spherical coordinates.

4.12.6.1 block

Refer to block in CFEM-Diffusion.

4.12.6.2 family

Refer to family in CFEM-Diffusion.

4.12.6.3 order

Refer to order in CFEM-Diffusion.

4.12.6.4 group_collapsing

Refer to group_collapsing in CFEM-Diffusion.

4.12.6.5 group_weights

Refer to group_weights in CFEM-Diffusion.

4.12.6.6 collapse_scattering

Refer to collapse_scattering in CFEM-Diffusion.

4.12.6.7 balance_table

Refer to balance_table in CFEM-Diffusion.

4.12.6.8 balance_table_on

Refer to balance_table_on in CFEM-Diffusion.
4.12.6.9 fixed_jacobian

Refer to *fixed_jacobian* in CFEM-Diffusion.

4.12.6.10 verbose

Refer to *verbose* in CFEM-Diffusion.

4.12.6.11 prefix

Refer to *prefix* in CFEM-Diffusion.

4.12.6.12 prefix_variables

Refer to *prefix_variables* in CFEM-Diffusion.

4.12.6.13 material_prop_namespace

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.12.6.14 initialize_iqs_adjoint_flux

Refer to *initialize_iqs_adjoint_flux* in CFEM-Diffusion.

4.12.6.15 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 [19] for more explanations.

4.12.6.16 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 [19] for more explanations. It is noted that this parameter is invalid for diffusion approximation.
4.12.6.17  **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 × (AQorder+2)/2 in 2D and AQorder × (AQorder+2) in 3D.

- Gauss-Chebyshev supports one-dimensional (1D), 2D and 3D calculations. The number of directions is equal to NPolar × 2 in 1D, NPolar × NAzimuthl × 4 in 2D and NPolar × NAzimuthl × 8 in 3D. If the input is for 1D with R-spherical coordinate and angle_derivative_scheme is set to SNSweep, the number of directions is equal to (NPolar+1) × 2 because two dummy directions at μ = 1 and μ = -1 are added. This quadrature is simply the Gaussian quadrature in 1D.

- Bickley3-Optimized only supports 2D calculations. The number of directions is equal to NPolar × NAzimuthl × 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 \(\vec{n} = \pm \vec{e}_x, \vec{n} = \pm \vec{e}_y\) or \(\vec{n} = \pm \vec{e}_z\), which covers most of practical calculations. Gauss-Chebyshev and Bickley3-Optimized can also be used if the z-component of \(\vec{n}\) is zero. In such cases, users can adjust NAzimuthl to satisfy the constraint. Both angular quadratures have in common that the azimuthal angles are distributed evenly. The angles are given by \((2i + 1) \times \pi/NAzimuthl/4, i = 0, \cdots, 4 \times NAzimuthl - 1\), where 4 is the number of octant. For example, if NAzimuthl 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 NAzimuthl being equal to \(3n\), where \(n\) is any natural number, satisfy the constraint for these surfaces as well.

4.12.6.18  **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.

4.12.6.19  **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. It will be ignored for other quadrature types even if provided.
**4.12.6.20 NAzmthl**

Description: The number of azimuthal angles of a product 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.

**4.12.6.21 angle_derivative_scheme**

Description: The scheme used for evaluating the angular derivative  
Data type: Enumeration (/PN/SNSweep/)  
Default value: PN  
Syntax: TransportSystems/*/angle_derivative_scheme  
Note: This parameter is active only when mu-derivative or omega-derivative evaluations are required, such as with curvilinear coordinates. The SNSweep option will result into the construction of dummy directions (only available with the mu-derivative at the moment).

**4.12.6.22 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 dimensions and (NA+1)^2 in three dimensions.

**4.12.6.23 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.
4.12.6.24  *hide-angular-flux*

Description: True to not output angular flux
Data type: Logical
Default value: False
Syntax: TransportSystems/*/hide-angular-flux

4.12.6.25  *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; 0 means that only scalar fluxes are reported.

4.12.6.26  *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 [18] for more explanations. Some executioner require this parameter to be true.

4.12.6.27  *explicit-on-boundary*

Description: True to use old angular fluxes for evaluating implicit BCs (reflecting or white etc.) during transport update
Data type: Logical
Default value: False
Syntax: TransportSystems/*/explicit-on-boundary
Note: users are referred to [18, 20] for more explanations.

4.12.6.28  *assemble-scattering-jacobian*

Refer to *assemble-scattering-jacobian* in CFEM-Diffusion.
4.12.6.29  \textit{tau}

Description: SUPG stabilization parameter for void or near-void

Data type: Real

Default value: 0.5

Syntax: TransportSystems/*/tau

Note: When the 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. In particular, \( \tau = 0 \) does not provide any additional stabilization other than the inverse of the cross-section used by the standard SAAF method (i.e. without the void treatment). Users are referred to [21] for more explanations.

4.12.6.30  \textit{stable_factor}

Description: Another SUPG stabilization parameter

Data type: Real

Default value: 1.0

Syntax: TransportSystems/*/stable_factor

Note: The lower this parameter is, the more stabilization is applied to the problem. When it is lower than \( \tau \) divided by the local element size in terms of mean free path, a fixed stabilization equal to the local element size divided by \( \tau \) is used. Thus setting this parameter to zero makes the stabilization only depend on the mesh, i.e. always equal to the local element size divided by \( \tau \). In practice, this parameter can be used for schemes such as SAAF-CFEM-SN in RZ cylindrical coordinates where stronger stabilization could be desired for better convergence. Users are referred to [21] for more explanations.

4.12.6.31  \textit{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.

4.12.6.32  \textit{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
4.12.6.33  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. There are two distinct modes that can be selected using the parameter larsen_trahan_mode that must be either “region” or “pointwise”. In case the default “region” is selected, region averaged diffusion coefficients are created and later written to a cross section file. Output cross sections files are created/overridden. The file names are formed as

\(<\text{input file base}> + \_\text{larsen_trahan\_material\_out}_ + \_\text{nm} + \_.\text{xml}\)

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.

If “pointwise” is selected, AuxKernels and AuxVariables are added to compute and store the diffusion tensors pointwise. This option is used for NDA calculation with Larsen-Trahan diffusion tensors.

Data type: Logical

Default value: False

Syntax: TransportSystems/*/larsen_trahan

Note: Can only be used with particle type common. Must use steady executioner or executioner supporting _for_transport_update. Supports ConstantNeutronicsMaterial, MixedNeutronicsMaterial, and MixedMatIDNeutronicsMaterial.

4.12.6.34  larsen_trahan_mode

Description: Switches between region averaged and pointwise computation of Larsen-Trahan diffusion tensors.

Data type: MooseEnum = ("region", "pointwise")

Default value: region

Syntax: TransportSystems/*/larsen_trahan_mode

Note: Must be used with keyword larsen_trahan.

4.12.6.35  fcs_input_file

Description: The name of the input file used to enable first collision source treatment.

Data type: String

Default value: <empty>

Syntax: TransportSystems/*/fcs_input_file

Note: The file must contain the uncollided flux solution as variables named group_0, group_1, ... , group_n, for energy groups 0, 1, ... , n.

When using first collision source treatment, the provided uncollided solution is used to create a first collision source. This source replaces the fixed sources in the \(S_N\) problem and the net scalar flux is then the sum of the provided uncollided scalar flux and the transport scalar flux. This net scalar flux is outputted in variables following the convention fcs_total_scalar_flux_g<\text{g}>. 

162
4.12.6.36  \textit{fcs\_app\_one\_proc}

Description: Whether or not to execute the MultiApp that imports the uncollided solution on a single processor.

Data type: Logical

Default value: True

Syntax: TransportSystems/*/fcs\_app\_one\_proc

Note: If false, the MultiApp executes on the number of processors chosen when running RattleSnake. Enabling this could potentially decrease the time needed to import the uncollided solution (thus speeding up the process), but will likely require more memory.

4.12.6.37  \textit{iqs\_constant\_time\_kernel}

Description: True to add a dummy kernel to ensure a constant time kernel for IQS.

Data type: Logical

Default value: False

Syntax: TransportSystems/*/iqs\_constant\_time\_kernel

Note: True to ensure a constant time kernel when evaluating the reactivity of the point-kinetics equation. It is noted that true will make the stabilization parameter stateful.

4.12.7  \textit{SAAF-CFEM-PN}

This discretization scheme shares the following parameters in CFEM-Diffusion: \textit{block}, \textit{family}, \textit{order}, \textit{group\_collapsing}, \textit{group\_weights}, \textit{collapse\_scattering}, \textit{balance\_table}, \textit{balance\_table\_on}, \textit{fixed\_jacobian}, \textit{verbose}, \textit{prefix}, \textit{prefix\_variables} and \textit{material\_prop\_namespace}. It also shares the following parameters in SAAF-CFEM-SN: \textit{vacuum\_bc\_type}, \textit{reflecting\_bc\_type} and \textit{hide\_higher\_flux\_moment}. This discretization scheme supports X-Y-Z and R-Z coordinates.

4.12.7.1  \textit{block}

Refer to \textit{block} in CFEM-Diffusion.

4.12.7.2  \textit{family}

Refer to \textit{family} in CFEM-Diffusion.

4.12.7.3  \textit{order}

Refer to \textit{order} in CFEM-Diffusion.

4.12.7.4  \textit{group\_collapsing}

Refer to \textit{group\_collapsing} in CFEM-Diffusion.
4.12.7.5  group_weights

Refer to group_weights in CFEM-Diffusion.

4.12.7.6  collapse_scattering

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

4.12.7.7  balance_table

Refer to balance_table in CFEM-Diffusion.

4.12.7.8  balance_table_on

Refer to balance_table_on in CFEM-Diffusion.

4.12.7.9  fixed_jacobian

Refer to fixed_jacobian in CFEM-Diffusion.

4.12.7.10  verbose

Refer to verbose in CFEM-Diffusion.

4.12.7.11  prefix

Refer to prefix in CFEM-Diffusion.

4.12.7.12  prefix_variables

Refer to prefix_variables in CFEM-Diffusion.

4.12.7.13  material_prop_namespace

Refer to material_prop_namespace in CFEM-Diffusion.

4.12.7.14  initialize_iqs_adjoint_flux

Refer to initialize_iqs_adjoint_flux in CFEM-Diffusion.

4.12.7.15  vacuum_bc_type

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

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

4.12.7.17  *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

4.12.7.18  *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.

4.12.7.19  *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.

4.12.7.20  *hide_higher_flux_moment*

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

4.12.7.21  *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
4.12.7.22  initialize_flux_moment

Description: True to initialize flux moments with one
Data type: Logical
Default value: False
Syntax: TransportSystems/*/initialize_flux_moment

4.12.7.23  void_blocks

Description: The subdomains corresponding to the void/near-void regions if the SAAF-CLS-PN method is to be used (SAAF and LS hybrid method providing a void treatment while preserving the global conservation of the scheme)
Data type: Vector of subdomain names
Default value: <empty>
Syntax: TransportSystems/*/void_blocks
Note: If this parameter is left empty, the void treatment is not activated.

4.12.7.24  partial_current_required_on

Description: Names of the side sets where the partial current is requested
Data type: Vector of boundary names
Default value: <empty>
Syntax: TransportSystems/*/partial_current_required_on
Note: The vacuum boundary are included by default and need not be added. This parameter is only used for evaluating partial currents with PartialSurfaceCurrent.

4.12.7.25  saaf_ls_scaling

Description: Scaling term (useful if SAAF-CLS scheme is used): units of a cross-section (but no physical meaning)
Data type: Real
Default value: 1
Syntax: TransportSystems/*/saaf_ls_scaling
Note: This parameter is used to adjust the scaling between the LS and SAAF kernels, respectively used in the void region and elsewhere. The value of this parameter typically has little impact on the solution (especially outside the void region) but may affect convergence. As a general guideline, it is encouraged to use a value close to that of the cross-section on the interface between the void region(s) and the rest of the problem.

4.12.8  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_jacobians. It also shares the following parameters in SAAF-CFEM-SN: AQtype, AQorder, NPolar, NAzmthl, angle_derivative_scheme, NA, initialize_angular_flux, hide_angular_flux, hide_higher_flux_moment, for_transport_update, explicit_on_boundary and show_drift. This discretization scheme supports only X-Y-Z coordinates.

4.12.8.1 block
Refer to block in CFEM-Diffusion.

4.12.8.2 family
Refer to family in CFEM-Diffusion.

4.12.8.3 order
Refer to order in CFEM-Diffusion.

4.12.8.4 group_collapsing
Refer to group_collapsing in CFEM-Diffusion.

4.12.8.5 group_weights
Refer to group_weights in CFEM-Diffusion.

4.12.8.6 balance_table
Refer to balance_table in CFEM-Diffusion.

4.12.8.7 balance_table_on
Refer to balance_table_on in CFEM-Diffusion.

4.12.8.8 fixed_jacobian
Refer to fixed_jacobian in CFEM-Diffusion.

4.12.8.9 verbose
Refer to verbose in CFEM-Diffusion.

4.12.8.10 prefix
Refer to prefix in CFEM-Diffusion.
4.12.8.11  *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.12.8.12  *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.12.8.13  *initialize_iqs_adjoint_flux*

Refer to *initialize_iqs_adjoint_flux* in CFEM-Diffusion.

4.12.8.14  *AQtype*

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

4.12.8.15  *AQorder*

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

4.12.8.16  *NPolar*

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

4.12.8.17  *NAzmthl*

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

4.12.8.18  *angle_derivative_scheme*

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

4.12.8.19  *NA*

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

4.12.8.20  *initialize_angular_flux*

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

4.12.8.21  *hide_angular_flux*

Refer to *hide_angular_flux* in SAAF-CFEM-SN.
4.12.8.22  *hide_higher_flux_moment*

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

4.12.8.23  *for_transport_update*

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

4.12.8.24  *explicit_on_boundary*

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

4.12.8.25  *assemble_scattering_jacobian*

Refer to *assemble_scattering_jacobian* in CFEM-Diffusion.

4.12.8.26  *show_drift*

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

4.12.8.27  *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 [22] for more details.

4.12.8.28  *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 [22] for more details.

4.12.8.29  *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 [22] for more details.
4.12.8.30  \texttt{fcs\_input\_file}

Refer to \texttt{fcs\_input\_file} in SAAF-CFEM-SN.

4.12.8.31  \texttt{fcs\_app\_one\_proc}

Refer to \texttt{fcs\_app\_one\_proc} in SAAF-CFEM-SN.

4.12.9  LS-CFEM-PN

This discretization scheme shares the following parameters in CFEM-Diffusion: \texttt{block}, \texttt{family}, \texttt{order}, \texttt{group\_collapsing}, \texttt{group\_weights}, \texttt{collapse\_scattering}, \texttt{balance\_table}, \texttt{balance\_table\_on}, \texttt{fixed\_jacobian}, \texttt{verbose}, \texttt{prefix}, \texttt{prefix\_variables} and \texttt{material\_prop\_namespace}. It also shares the following parameters in SAAF-CFEM-SN: \texttt{vacuum\_bc\_type}, \texttt{reflecting\_bc\_type} and \texttt{hide\_higher\_flux\_moment}. It also shares the following parameters in SAAF-CFEM-PN: \texttt{PN}, \texttt{NA}, \texttt{force\_secondary\_parity}, \texttt{initialize\_flux\_moment}.

This scheme does not work with \textit{transient equation\_type}. It supports only X-Y-Z coordinates.

4.12.9.1  \texttt{block}

Refer to \texttt{block} in CFEM-Diffusion.

4.12.9.2  \texttt{family}

Refer to \texttt{family} in CFEM-Diffusion.

4.12.9.3  \texttt{order}

Refer to \texttt{order} in CFEM-Diffusion.

4.12.9.4  \texttt{group\_collapsing}

Refer to \texttt{group\_collapsing} in CFEM-Diffusion.

4.12.9.5  \texttt{group\_weights}

Refer to \texttt{group\_weights} in CFEM-Diffusion.

4.12.9.6  \texttt{collapse\_scattering}

Refer to \texttt{collapse\_scattering} in CFEM-Diffusion. It is noted that the default value of \texttt{collapse\_scattering} is changed to true.
4.12.9.7  balance_table
Refer to balance_table in CFEM-Diffusion.

4.12.9.8  balance_table_on
Refer to balance_table_on in CFEM-Diffusion.

4.12.9.9  fixed_jacobian
Refer to fixed_jacobian in CFEM-Diffusion.

4.12.9.10  verbose
Refer to verbose in CFEM-Diffusion.

4.12.9.11  prefix
Refer to prefix in CFEM-Diffusion.

4.12.9.12  prefix_variables
Refer to prefix_variables in CFEM-Diffusion.

4.12.9.13  material_prop_namespace
Refer to material_prop_namespace in CFEM-Diffusion.

4.12.9.14  initialize_iqs_adjoint_flux
Refer to initialize_iqs_adjoint_flux in CFEM-Diffusion.

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

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

4.12.9.17  parity_option
Refer to parity_option in SAAF-CFEM-PN.
4.12.9.18  PN
Refer to PN in SAAF-CFEM-PN.

4.12.9.19  NA
Refer to NA in SAAF-CFEM-PN.

4.12.9.20  hide_higher_flux_moment
Refer to hide_higher_flux_moment in SAAF-CFEM-SN.

4.12.9.21  force_secondaryparity
Refer to force_secondaryparity in SAAF-CFEM-PN.

4.12.9.22  initialize_flux_moment
Refer to initialize_flux_moment in SAAF-CFEM-PN.

4.12.9.23  weak_bc_type
Refer to weak_bc_type in LS-CFEM-SN.
The default value is changed to 7, for consistency with the SAAF-CFEM-SN void treatment.

4.12.9.24  weak_bc_constant
Refer to weak_bc_constant in LS-CFEM-SN.

4.12.9.25  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 retained.

4.12.10 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, angle_derivative_scheme, NA, initialize_angular_flux, hide_angular_flux, hide_higher_flux_moment, for_transport_update, explicit_on_boundary and larsen_trahan. This discretization scheme supports X-Y-Z and R-Z cylindrical coordinates.

4.12.10.1 block

Refer to block in CFEM-Diffusion.

4.12.10.2 family

Refer to family in CFEM-Diffusion. It is noted that the default value of family is changed to L2_LAGRANGE.

4.12.10.3 order

Refer to order in CFEM-Diffusion.

4.12.10.4 group_collapsing

Refer to group_collapsing in CFEM-Diffusion.

4.12.10.5 group_weights

Refer to group_weights in CFEM-Diffusion.

4.12.10.6 balance_table

Refer to balance_table in CFEM-Diffusion.

4.12.10.7 balance_table_on

Refer to balance_table_on in CFEM-Diffusion.

4.12.10.8 fixed_jacobian

Refer to fixed_jacobian in CFEM-Diffusion.

4.12.10.9 verbose

Refer to verbose in CFEM-Diffusion.

4.12.10.10 prefix

Refer to prefix in CFEM-Diffusion.
4.12.10.11  *prefix_variables*

Refer to *prefix_variables* in CFEM-Diffusion.

4.12.10.12  *material_prop_namespace*

Refer to *material_prop_namespace* in CFEM-Diffusion.

4.12.10.13  *initialize_iqs_adjoint_flux*

Refer to *initialize_iqs_adjoint_flux* in CFEM-Diffusion.

4.12.10.14  *AQtype*

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

4.12.10.15  *AQorder*

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

4.12.10.16  *NPolar*

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

4.12.10.17  *NAzmthl*

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

4.12.10.18  *angle_derivative_scheme*

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

4.12.10.19  *NA*

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

4.12.10.20  *initialize_angular_flux*

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

4.12.10.21  *hide_angular_flux*

Refer to *hide_angular_flux* in SAAF-CFEM-SN.
4.12.10.22  *hide_higher_flux_moment*

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

4.12.10.23  *for_transport_update*

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

4.12.10.24  *explicit_on_boundary*

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

4.12.10.25  *larsen_trahan*

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

4.12.10.26  *fcs_input_file*

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

4.12.10.27  *fcs_app_one_proc*

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

4.12.11  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*. This discretization scheme supports only X-Y-Z coordinates.

4.12.11.1  *block*

Refer to *block* in CFEM-Diffusion.

4.12.11.2  *family*

Refer to *family* in CFEM-Diffusion. It is noted that the default value of *family* is changed to L2_LAGRANGE.

4.12.11.3  *order*

Refer to *order* in CFEM-Diffusion.
4.12.11.4  group_collapsing
Refer to group_collapsing in CFEM-Diffusion.

4.12.11.5  group_weights
Refer to group_weights in CFEM-Diffusion.

4.12.11.6  balance_table
Refer to balance_table in CFEM-Diffusion.

4.12.11.7  balance_table_on
Refer to balance_table_on in CFEM-Diffusion.

4.12.11.8  fixed_jacobian
Refer to fixed_jacobian in CFEM-Diffusion.

4.12.11.9  verbose
Refer to verbose in CFEM-Diffusion.

4.12.11.10  prefix
Refer to prefix in CFEM-Diffusion.

4.12.11.11  prefix_variables
Refer to prefix_variables in CFEM-Diffusion.

4.12.11.12  material_prop_namespace
Refer to material_prop_namespace in CFEM-Diffusion.

4.12.11.13  initialize_iqs_adjoint_flux
Refer to initialize_iqs_adjoint_flux in CFEM-Diffusion.

4.12.11.14  vacuum_bc_type
Refer to vacuum_bc_type in SAAF-CFEM-SN.
4.12.11.15  *reflecting_bc_type*

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

4.12.11.16  *parity_option*

Refer to *parity_option* in SAAF-CFEM-PN.

4.12.11.17  *PN*

Refer to *PN* in SAAF-CFEM-PN.

4.12.11.18  *NA*

Refer to *NA* in SAAF-CFEM-PN.

4.12.11.19  *hide_higher_flux_moment*

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

4.12.11.20  *force_secondary_parity*

Refer to *force_secondary_parity* in SAAF-CFEM-PN.

4.12.11.21  *initialize_flux_moment*

Refer to *initialize_flux_moment* in SAAF-CFEM-PN.

4.12.11.22  *NS*

Refer to *NS* in LS-CFEM-PN.

4.12.11.23  *filter_type*

Description: The type of filter to use  
Data type: Enumeration (/none/Lanczos/SSpline/Exp)  
Default value: none  
Syntax: TransportSystems/*/filter_type

4.12.11.24  *filter_strength_func*

Description: The function to use as the filter strength  
Data type: String  
Default value: <empty>  
Syntax: TransportSystems/*/filter_strength_func
4.12.11.25  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.

4.12.11.26  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.

4.12.11.27  removal_lumping

Description: True to lump the removal terms
Data type: Logical
Default value: false
Syntax: TransportSystems/*/removal_lumping

4.13 Summary of MOOSE objects added by *TransportSystems*

Rattlesnake is an open system in the sense that users can add additional MOOSE objects, including variables, materials, postprocessors etc., into the input file and can retrieve all MOOSE objects contained in Rattlesnake for post-processing purpose. Therefore, it is necessary to summarize the MOOSE objects added by Rattlesnake. All these objects are listed 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 we expect to be useful to users are listed.

4.13.1 Primal variables

The primal variables added with transport systems are listed in Table 22 and Table 23. 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. The temperature variable is added for Thermal if setup_temperature_equation is true.
Table 22  Primal variables of neutron transport systems.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Name</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar flux</td>
<td>sflux_g&lt;g&gt;</td>
<td>CFEM-Diffusion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFEM-Diffusion</td>
</tr>
<tr>
<td>Angular flux moment</td>
<td>flux_moment_g&lt;g&gt;_L&lt;_l&gt;_M&lt;_m&gt;</td>
<td>SAAF-CFEM-PN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LS-CFEM-PN</td>
</tr>
<tr>
<td>Angular flux</td>
<td>aflux_g&lt;g&gt;_d&lt;d&gt;</td>
<td>SAAF-CFEM-SN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LS-CFEM-SN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFEM-SN</td>
</tr>
</tbody>
</table>

Table 23  Primal variables of thermal radiation transport systems.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Name</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radiation density</td>
<td>E_g&lt;g&gt;</td>
<td>CFEM-Diffusion</td>
</tr>
<tr>
<td>Angular flux moment</td>
<td>E_g&lt;g&gt;_L&lt;_l&gt;_M&lt;_m&gt;</td>
<td>DFEM-PN</td>
</tr>
<tr>
<td>Specific intensity</td>
<td>specific_intensity_g&lt;g&gt;_d&lt;d&gt;</td>
<td>SAAF-CFEM-SN</td>
</tr>
<tr>
<td>Temperature</td>
<td>T_option</td>
<td></td>
</tr>
</tbody>
</table>

4.13.2  Auxiliary variables

The auxiliary variables added with transport systems are listed in Table 24 and Table 25. The delayed neutron group index \( i \) 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 for SN schemes with the maximum order being \( NA \). Temperature variable is added for Thermal only if \( setup_temperature_equation \) is true. Prompt fission source and delayed neutron precursors are added for Neutron only if \( fission_source_as_material \) 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 24  Auxiliary variables of neutron transport systems.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Name</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angular flux moment</td>
<td>flux_moment_g&lt;g&gt;_L&lt;_l&gt;_M&lt;_m&gt;</td>
<td>SAAF-CFEM-SN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LS-CFEM-SN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFEM-SN</td>
</tr>
<tr>
<td>First collision source</td>
<td>fcs_total_scalar_flux_g&lt;g&gt;</td>
<td>SAAF-CFEM-SN</td>
</tr>
<tr>
<td>total scalar flux</td>
<td></td>
<td>LS-CFEM-SN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFEM-SN</td>
</tr>
<tr>
<td>Prompt fission source</td>
<td>fission_source</td>
<td></td>
</tr>
<tr>
<td>Delayed neutron precursor</td>
<td>dnp_i&lt;_i&gt;</td>
<td></td>
</tr>
<tr>
<td>Adjoint fission source</td>
<td>fission_source_adj</td>
<td></td>
</tr>
</tbody>
</table>

4.13.3  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 \((l, m)\)
Table 25 Auxiliary variables of thermal radiation transport systems.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Name</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angular flux moment</td>
<td>$E_{g&gt;L&lt;M}^{&lt;g&gt;}$</td>
<td>SAAF-CFEM-SN</td>
</tr>
<tr>
<td>First collision source</td>
<td>fcs_total_scalar_flux_g^{&lt;g&gt;}</td>
<td>SAAF-CFEM-SN</td>
</tr>
<tr>
<td>total scalar flux</td>
<td></td>
<td>LS-CFEM-SN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFEM-SN</td>
</tr>
<tr>
<td>Temperature</td>
<td>temperature</td>
<td></td>
</tr>
<tr>
<td>Planck emission</td>
<td>total_emission</td>
<td></td>
</tr>
<tr>
<td>Radiation heating</td>
<td>radiation_heat_source</td>
<td></td>
</tr>
<tr>
<td>Radiative equilibrium</td>
<td>radiative_equilibrium</td>
<td></td>
</tr>
</tbody>
</table>

The total number of the scattering sources $P$ is

$$P = G \times \begin{cases} l, & 1D \\ \frac{l(l+1)}{2} + m, & 2D \\ l^2 + l + m, & 3D \end{cases}$$

where $L$ is $NA$. $G$ is the number of coarse groups specified by $G$ and $group\_collapsing$. Typically scattering sources are evaluated using the current solution. If $for\_transport\_update$ is true for the SAAF-CFEM-SN scheme, they are evaluated using the old solution.

Material properties of prompt fission source, delayed neutron precursor (DNP) concentrations are evaluated when $fission\_source\_as\_material$ is true for Neutron transport. 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 outflow partial currents are only evaluated for SN schemes, SAAF-CFEM-SN, LS-CFEM-SN and DFEM-SN, on the white boundaries specified by $WhiteBoundary$.

4.14 Supporting coordinate types

Rattlesnake allows three different coordinates: X-Y-Z, R-Z (axial-symmetric) and R-spherical. The supported coordinates by all schemes are summarized in Table 27.
### Table 26  Material properties of transport systems.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Name</th>
<th>Scheme</th>
<th>Stateful</th>
<th>Volumetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-group scattering source</td>
<td>scattering_source_p&lt;p&gt;_g&lt;g&gt;</td>
<td>CFEM-Diffusion, DFE-M-Diffusion, SAAF-CFEM-SN, SAAF-CFEM-PN, LS-CFEM-PN</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Prompt fission source</td>
<td>fission_source</td>
<td>Neutron</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Delayed neutron precursor</td>
<td>dnp_i&lt;i&gt;</td>
<td>Neutron</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Secondary parity moments</td>
<td>flux_moment_g&lt;g&gt;_L&lt;l&gt;_M&lt;m&gt;</td>
<td>SAAF-CFEM-PN, LS-CFEM-PN</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SAAF SN stabilization parameter $\tau$</td>
<td>tau_g&lt;g&gt;</td>
<td>SAAF-CFEM-SN</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>$</td>
<td>\vec{n}</td>
<td>_1$ of all sides</td>
<td>l1norm_normals</td>
<td>DFM-PN</td>
</tr>
<tr>
<td>$\vec{n}_{\text{crp}} = \frac{1}{\int ds} \int \vec{n}_n ds$ of all sides</td>
<td>averaged_normal</td>
<td>DFM-Diffusion</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Boundary out-going partial current</td>
<td>out_current_g&lt;g&gt;</td>
<td>SAAF-CFEM-SN, LS-CFEM-SN, DFM-SN</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### Table 27  The supported coordinates by schemes.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>X-Y-Z</th>
<th>R-Z</th>
<th>R-spherical</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFEM-Diffusion</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DFE-M-Diffusion</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SAAF-CFEM-SN</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SAAF-CFEM-PN</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LS-CFEM-SN</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>LS-CFEM-PN</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>DFE-M-SN</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>DFE-M-PN</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

The following parameters can be used for calculations other than X-Y-Z:

#### 4.14.1  block

Description: Block IDs for the coordinate systems

Data type: Vector of strings

Default value: <empty>

Syntax: Problem/block

Note: Default to all blocks.

#### 4.14.2  coord_type

Description: Type of the coordinate system per block param

Data type: Enumeration (/XYZ/RZ/RSPHERICAL/)
Default value: XYZ
Syntax: Problem/coord_type

4.14.3 \textit{rz\_coord\_axis}

Description: The rotation axis (R) for axisymmetric coordinates
Data type: Enumeration (/X/Y/)
Default value: Y
Syntax: Problem/rz\_coord\_axis
The point-kinetics equations (PKE) are given by:

\[
\frac{dn(t)}{dt} = \rho - \frac{\beta}{\Lambda} n + \sum_{i=1}^{I} \lambda_i c_i, \quad (21)
\]

\[
\frac{dc_i(t)}{dt} = \frac{\beta_i}{\Lambda} n - \lambda_i c_i; \quad i = 1, \cdots, I, \quad (22)
\]

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. \(\rho, \Lambda, \beta_i\) and \(\lambda_i\) are the reactivity, neutron generation time, DNP fractions and DNP decay constants, which are all potentially time dependent. The total DNP fraction \(\bar{\beta} = \sum_{i=1}^{I} \beta_i\). The initial condition for PKE is

\[
n(t = t_0) = n_0 \quad (23)
\]

\[
c_i(t = t_0) = c_{i,0}; \quad i = 1, \cdots, I, \quad (24)
\]

where \(n_0\) and \(c_{i,0}\) are the solutions at the initial time \(t_0\). Typically we assume the DNP concentrations reach the equilibrium condition at the starting time, i.e.

\[
c_i(t = t_0) = \frac{\beta_i}{\Lambda \lambda_i} n_0; \quad i = 1, \cdots, I. \quad (25)
\]

The PKE problem is set up using the PKE input block in 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 the user’s 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 \(c_i(t), i = 1, \cdots, I\). \(\rho, \Lambda, \beta_i, i = 1, \cdots, I\) and \(\lambda_i, i = 1, \cdots, I\) are treated as the auxiliary scalar variables of the correct order. It is the user’s responsibility to provide initial conditions, and/or auxiliary scalar kernels (if they change with time) for setting values for these variables. MOOSE contains several initial conditions for scalar variables that are shown in Fig. 21. MOOSE contains several scalar auxiliary kernels to set values for the auxiliary scalar variables as shown in Fig. 22.

Rattlesnake’s IQS executioner has the capability of dumping the lumped PKE parameters \((\rho, \Lambda, \beta_i, i = 1, \cdots, I\) and \(\lambda_i, i = 1, \cdots, I\)) while performing the spatial kinetics calculations. Users can use these dumped parameters to reproduce the transient history of the neutron production rate using the PKE. The PKE input block contains an optional parameter indicating in which file the dumped PKE parameters are stored. 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 \textit{n\_delayed\_groups}

Description: Number of groups of delayed neutron precursors
Data type: Integer
Default value: \texttt{<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 \textit{amplitude\_variable}

Description: Amplitude primal scalar variable
Data type: String
Default value: \texttt{<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 \textit{DNP\_variable}

Description: Delayed neutron precursor primal scalar variable
Data type: String
Default value: \texttt{<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 \textit{DNP\_fraction\_aux}

Description: Auxiliary variable for the delayed neutron precursor fractions
Data type: String
Default value: \texttt{<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 $\beta_i, i = 1, \cdots, I$. 

Figure 22  Scalar auxiliary kernels in MOOSE.
5.1.5  \textit{DNP\_decay\_constant\_aux}

Description: Auxiliary variable for the delayed neutron precursor decay constants
Data type: String
Default value: \texttt{<required>}
Syntax: PKE/DNP\_decay\_constant\_aux

Note: An auxiliary scalar variable with order of \texttt{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  \textit{generation\_time\_aux}

Description: Auxiliary variable for the neutron generation time
Data type: String
Default value: \texttt{<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 $\Lambda$.

5.1.7  \textit{reactivity\_aux}

Description: Auxiliary variable for the reactivity
Data type: String
Default value: \texttt{<required>}
Syntax: PKE/reactivity\_aux

Note: An first-order auxiliary scalar variable will be added by Rattlesnake with this name for the reactivity $\rho$.

5.1.8  \textit{has\_initial\_equilibrium}

Description: Whether or not DNP concentrations are in 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. (25) will added by Rattlesnake. Thus, no initial conditions for DNP concentrations will be needed from users.

5.1.9  \textit{pke\_parameter\_csv}

Description: The CSV file containing dumped PKE parameters used for setting up PKE
Data type: String
Default value: \texttt{<empty>}
Syntax: PKE/pke\_parameter\_csv
Note: If this parameter is provided, the CSV file will be used to set up interpolation functions for PKE parameters $\rho$, $\Lambda$, $\beta$ and $\lambda$. 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*

Rattlesnake allows multiple loosely coupled regions to coexist in a transport system; a prominent example is the coupled reactor system given in the tutorials. This input block makes Rattlesnake evaluate the coupled parameters including the generation time, reactivity, etc. for these multiple regions. Rattlesnake uses *MultiApp/Transfer system* to accomplish the calculations and outputs the parameters on screen at the end. Detailed explanations on these parameters can be found in the 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 multischeme.

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 (one for a region)
Data type: Vector of strings
Default value: `<required>`
Syntax: MultiRegion/forward_partial_multiapp_files
Note: Names need to be in the same order as the ones provided for the `adjoint_partial_multiapp_files` parameter. 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 (one for a region)
Data type: Vector of strings
Default value: `<required>`
Syntax: MultiRegion/adjoint_partial_multiapp_files
Note: Names need to be in the same order as the ones for the `forward_partial_multiapp_files` parameter. 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
5.3  *TransportSolutionError*

This input block is designed to compute the angular flux L2-error for PN calculations, for which the primal variables are the angular moments. Detailed explanations on how this can be done using the orthogonality property of the spherical harmonics can be found in the Rattlesnake theory manual. It particularly relies on the FluxMomentL2Error postprocessor.

As detailed below, the reference solution can either be given as a function or be read from xda files, generated for instance by running the same calculation on a more (angularly, spatially or temporally) refined mesh.

5.3.1  *transport_system*

Description: Name of the transport system

Data type: String

Default value: `<required>`

Syntax: TransportSolutionError/transport_system

5.3.2  *N_infinity*

Description: Maximum degree of moments used to compute the error norm of the angular flux.

Data type: Unsigned integer

Default value: 0

Syntax: TransportSolutionError/N_infinity

Note: If the reference solution comes from a PN calculation with \( N = N_\infty \), \( N_{\infty} \) should typically be set to \( N_\infty \). If the L2-error of the scalar flux is desired (as opposed to that of the angular flux), it should be set to 0.

5.3.3  *reference_solution*

Description: Reference solution deriving from TransportFunction to compute the L2-norm of the angular flux.

Data type: Function name

Default value: `<empty>`

Syntax: TransportSolutionError/reference_solution

5.3.4  *xda_file_name*

Description: Name of the xda file to read from (if the reference solution is not provided)

Data type: String

Default value: `<empty>`

Syntax: TransportSolutionError/xda_file_name
Note: If reference_solution is not provided, xda_file_name must be provided. Otherwise, it will be ignored.

To generate these xda files, one can for instance run a similar input file on a more (angularly, spatially or temporally) refined mesh without the TransportSolutionError block and Outputs/xda set to true. If Outputs/file_base is set to my_refined_solution, then two files will be created: my_refined_solution_XXXX.xda and my_refined_solution_XXXX_mesh.xda where XXXX is the time index (0000 for the zeroth time step etc.). In that case, xda_file_name should be set to my_refined_solution_XXXX. Examples upon which the implementation was based can be found in the following MOOSE examples: moose/examples/ex14_pps/ex14_compare_solutions_1.i and ex14_compare_solutions_2.i.

## 5.4 RestartVariables

This input block can be used to initialize variables with the solutions in Exodus mesh files. It can contain multiple sub-blocks each one loading solutions from a specified Exodus mesh file.

### 5.4.1 exodus_filename

Description: The name of the Exodus file  
Data type: String  
Default value: <required>  
Syntax: RestartVariables/*/exodus_filename

### 5.4.2 variable_names

Description: The names of the variables read from file  
Data type: Vector of strings  
Default value: <required>  
Syntax: RestartVariables/*/variable_names

### 5.4.3 target_variable_names

Description: The corresponding names of the variables in the system  
Data type: Vector of strings  
Default value: <empty>  
Syntax: RestartVariables/*/target_variable_names

Note: If this parameter is not set, the names in variable_names will be used to create variables in the system. Otherwise, the size of this parameter must be equal to the size of variable_names. Target variable names must not conflict with names of existing variables.
5.4.4 scaling

Description: Scaling factor for each of the variables
Data type: Vector of reals
Default value: <empty>
Syntax: RestartVariables/*/target_variable_names

Note: If this parameter is not set, a default scaling of one is applied to all variables. Otherwise, the size of this parameter must be equal to the size of variable_names.

5.4.5 to_system

Description: The system where the variables are added
Data type: Enumeration (/NONLINEAR/AUXILIARY/)
Default value: NONLINEAR
Syntax: RestartVariables/*/to_system

5.4.6 family

Description: The FE families of the variables
Data type: Enumeration (refer to MOOSE syntax about Variables)
Default value: LAGRANGE
Syntax: RestartVariables/*/family

5.4.7 order

Description: The FE orders of the variables
Data type: Enumeration (refer to MOOSE syntax about Variables)
Default value: FIRST
Syntax: RestartVariables/*/order

5.4.8 eigen

Description: True to make variables eigen variables
Data type: Logical
Default value: False
Syntax: RestartVariables/*/eigen

Note: This parameter is active only when the target variables is on the primal nonlinear eigenvalue system.
5.4.9 \textit{block}

Description: The block IDs where these variables live
Data type: Vector of subdomain names
Default value: `<empty>`
Syntax: RestartVariables/*\text{/}block

Note: Default means that the variables are defined on the entire domain.
Mesh can be either generated by one of the built-ins 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 mesh 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: If the simulation is using second order shape functions with CFEM (continuous finite element method), the mesh must be of 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 \textit{subdomain} to let the user assign the block IDs of all generated elements.

6.2.1 \textit{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 \textit{nx}

Description: Number of elements in the X direction
Data type: Integer
Default value: 1
Syntax: Mesh/nx

6.2.3 \textit{xmin}

Description: Lower X Coordinate of the generated mesh
Data type: Real
Default value: 0
Syntax: Mesh/xmin

6.2.4 \textit{xmax}

Description: Upper X Coordinate of the generated mesh
Data type: Real
Default value: 1
Syntax: Mesh/xmax
6.2.5  \textit{ny}

Description: Number of elements in the Y direction
Data type: Integer
Default value: 1
Syntax: Mesh/\textit{ny}
Note: This parameter will be ignored when \textit{dim} is 1.

6.2.6  \textit{ymin}

Description: Lower Y Coordinate of the generated mesh
Data type: Real
Default value: 0
Syntax: Mesh/\textit{ymin}
Note: This parameter will be ignored when \textit{dim} is 1.

6.2.7  \textit{ymax}

Description: Upper Y Coordinate of the generated mesh
Data type: Real
Default value: 1
Syntax: Mesh/\textit{ymax}
Note: This parameter will be ignored when \textit{dim} is 1.

6.2.8  \textit{nz}

Description: Number of elements in the Z direction
Data type: Integer
Default value: 1
Syntax: Mesh/\textit{nz}
Note: This parameter will be ignored when \textit{dim} is 1 or 2.

6.2.9  \textit{zmin}

Description: Lower Z Coordinate of the generated mesh
Data type: Real
Default value: 0
Syntax: Mesh/\textit{zmin}
Note: This parameter will be ignored when \textit{dim} is 1 or 2.
6.2.10  \textit{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 \textit{dim} is 1 or 2.

6.2.11  \textit{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 \textit{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  \textit{distribution}  

Description: Whether or not to distribute storage of the mesh among processors (Do all processors own the whole mesh or just a small fraction of it).  
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 do not distribute 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. This only controls if the mesh is storage is distributed. Variables, Jacobians and other applicable quantities are always broken up and stored in parallel.

6.2.13  \textit{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: Subdomain IDs of elements
Data type: Vector of integers
Default value: <empty>
Syntax: Mesh/subdomain

Note: If this parameter is empty, the subdomain 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 $nx \times ny$. The number of elements in 3D is equal to $nx \times ny \times nz$. Subdomain IDs are ordered as $(i, j, k), i = 1, \cdots, nx; j = 1, \cdots, ny; k = 1, \cdots, nz$, where $(i, j, k)$ is the grid index.

6.3 **GeneratedIDMesh**

This mesh generator generates a regular mesh with uniformly distributed elements. This mesh generator is very similar to GeneratedIDMesh except that a few additional parameters to let the user also assign the material, depletion and equivalence IDs of all generated elements.

6.3.1 **subdomain**

Refer to subdomain in GeneratedIDMesh.

6.3.2 **material_id**

Description: Material IDs of elements
Data type: Vector of integers
Default value: <empty>
Syntax: Mesh/material_id

Note: If this parameter is empty, the material IDs of all elements are assumed to be equal to the block IDs set through subdomain. 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 $nx \times ny$. The number of elements in 3D is equal to $nx \times ny \times nz$. 

197
6.3.3 depletion_id

Description: Depletion IDs of elements
Data type: Vector of integers
Default value: <empty>
Syntax: Mesh/depletion_id

Note: If this parameter is empty, the depletion IDs of all elements are assigned to an invalid ID. 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 $n_x$. The number of elements in 2D is equal to $n_x \times n_y$. The number of elements in 3D is equal to $n_x \times n_y \times n_z$.

6.3.4 equivalence_id

Description: Equivalence IDs of elements
Data type: Vector of integers
Default value: <empty>
Syntax: Mesh/equivalence_id

Note: If this parameter is empty, the material IDs of all elements are assumed to be equal to the material IDs set through material_id. 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 $n_x$. The number of elements in 2D is equal to $n_x \times n_y$. The number of elements in 3D is equal to $n_x \times n_y \times n_z$.

6.3.5 dim

Refer to dim in GeneratedBIDMesh.

6.3.6 nx

Refer to nx in GeneratedBIDMesh.

6.3.7 xmin

Refer to xmin in GeneratedBIDMesh.

6.3.8 xmax

Refer to xmax in GeneratedBIDMesh.

6.3.9 ny

Refer to ny in GeneratedBIDMesh.

6.3.10 ymin

Refer to ymin in GeneratedBIDMesh.
6.3.11 \textit{ymax} \\
Refer to \textit{ymax} in GeneratedBIDMesh.

6.3.12 \textit{nz} \\
Refer to \textit{nz} in GeneratedBIDMesh.

6.3.13 \textit{zmin} \\
Refer to \textit{zmin} in GeneratedBIDMesh.

6.3.14 \textit{zmax} \\
Refer to \textit{zmax} in GeneratedBIDMesh.

6.3.15 \textit{elem\_type} \\
Refer to \textit{elem\_type} in GeneratedBIDMesh.

6.3.16 \textit{distribution} \\
Refer to \textit{distribution} in GeneratedBIDMesh.

6.3.17 \textit{partitioner} \\
Refer to \textit{partitioner} in GeneratedBIDMesh.

6.3.18 \textit{centroid\_partitioner\_direction} \\
Refer to \textit{centroid\_partitioner\_direction} in GeneratedBIDMesh.

6.4 \textit{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 \textit{dim} is equal to 1, 2 and 3 respectively.

6.4.1 \textit{dim} \\
Description: The dimension of the mesh to be generated \\
Data type: Enumeration (/1/2/3/) \\
Default value: \textless\textit{required}\textgreater \\
Syntax: Mesh/dim
6.4.2 \( dx \)
Description: Intervals in the X direction
Data type: Vector of reals
Default value: \(<\text{required}>\>
Syntax: Mesh/dx

6.4.3 \( ix \)
Description: Number of grids in all intervals in the X direction (default to all one)
Data type: Vector of integers
Default value: \(<\text{empty}>\>
Syntax: Mesh/ix
Note: If provided the size of this parameter must be equal to the size of \( dx \).

6.4.4 \( dy \)
Description: Intervals in the Y direction
Data type: Vector of reals
Default value: \(<\text{empty}>\>
Syntax: Mesh/dy
Note: This parameter is required for 2D and 3D. It will be ignored for 1D.

6.4.5 \( iy \)
Description: Number of grids in all intervals in the Y direction (default to all one)
Data type: Vector of integers
Default value: \(<\text{empty}>\>
Syntax: Mesh/iy
Note: If provided the size of this parameter must be equal to the size of \( dy \). It will be ignored for 1D.

6.4.6 \( dz \)
Description: Intervals in the Z direction
Data type: Vector of reals
Default value: \(<\text{empty}>\>
Syntax: Mesh/dz
Note: This parameter is required for 3D. It will be ignored for 1D and 2D.
6.4.7  \textit{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: If provided the size of this parameter must be equal to the size of \textit{dz}. It will be ignored for 1D and 2D. \\

6.4.8  \textit{subdomain_id} \\
Description: Block IDs (default to all zero) \\
Data type: Vector of integers \\
Default value: <empty> \\
Syntax: Mesh/subdomain_id \\
Note: if both \textit{dy} and \textit{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 \textit{dx}, \textit{dy} and \textit{dz} respectively. Otherwise, it must be equal to \( nx \times ny \) if \textit{dy} is provided, or \( nx \) when \textit{dy} is not provided. \\

6.4.9  \textit{distribution} \\
Refer to \textit{distribution} in GeneratedBIDMesh. \\

6.4.10  \textit{partitioner} \\
Refer to \textit{partitioner} in GeneratedBIDMesh. \\

6.4.11  \textit{centroid_partitioner_direction} \\
Refer to \textit{centroid_partitioner_direction} in GeneratedBIDMesh. \\

6.5 Hexagonal meshes \\
(tobe added.) \\

6.6 \textit{FileMesh}
This option is used to load a mesh from 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 28 by libMesh.

<table>
<thead>
<tr>
<th>File extension</th>
<th>Mesh format</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.e</td>
<td>Sandia’s ExodusII format</td>
</tr>
<tr>
<td>*.exd</td>
<td>Sandia’s ExodusII format</td>
</tr>
<tr>
<td>*.gmv</td>
<td>LANL’s General Mesh Viewer format</td>
</tr>
<tr>
<td>*.mat</td>
<td>Matlab triangular ASCII file</td>
</tr>
<tr>
<td>*.n</td>
<td>Sandia’s Nemesis format</td>
</tr>
<tr>
<td>*.nem</td>
<td>Sandia’s Nemesis format</td>
</tr>
<tr>
<td>*.off</td>
<td>OOGL OFF surface format</td>
</tr>
<tr>
<td>*.ucd</td>
<td>AVS’s ASCII UCD format</td>
</tr>
<tr>
<td>*.unv</td>
<td>I-deas Universal format</td>
</tr>
<tr>
<td>*.vtu</td>
<td>Paraview VTK format</td>
</tr>
<tr>
<td>*.inp</td>
<td>Abaqus .inp format</td>
</tr>
<tr>
<td>*.xda</td>
<td>libMesh ASCII format</td>
</tr>
<tr>
<td>*.xdr</td>
<td>libMesh binary format</td>
</tr>
<tr>
<td>*.gz</td>
<td>any above format gzipped</td>
</tr>
<tr>
<td>*.bz2</td>
<td>any above format bzip2’ed</td>
</tr>
<tr>
<td>*.xz</td>
<td>any above format xzipped</td>
</tr>
<tr>
<td>*.cpa</td>
<td>libMesh Checkpoint ASCII format</td>
</tr>
<tr>
<td>*.cpr</td>
<td>libMesh Checkpoint binary format</td>
</tr>
</tbody>
</table>

6.6.1 distribution
Refer to distribution in GeneratedBIDMesh.

6.6.2 partitioner
Refer to partitioner in GeneratedBIDMesh.

6.6.3 centroid_partitioner_direction
Refer to centroid_partitioner_direction in GeneratedBIDMesh.

6.6.4 file
Description: The name of the mesh file to read
Data type: String
Default value: <required>
Syntax: Mesh/file
6.7 **FileIDMesh**

Conceptually, this option is to `FileMesh` as `GeneratedIDMesh` is to `GeneratedBIDMesh`, in the sense that it extends it to assign material, depletion and equivalence IDs. However, unlike `FileMesh`, only Exodus files are currently supported. Furthermore, each of these IDs may be assigned either through elemental variables stored in the Exodus file or manually through input parameters (or not at all). For a given ID type (material, depletion or equivalence), it is not allowed to assign the IDs through both a variable from the mesh and in the input. It is however possible to assign one type with a variable and another through the input file.

In addition, if the material ID is not somehow provided, it will be assumed to be equal to the block ID. Likewise, if the equivalence ID is not somehow provided, it will be assumed to be equal to the material ID.

### 6.7.1 material_id_name

*Description:* Name of the variable providing the material IDs (from the mesh file)

*Data type:* String

*Default value:* `<empty>`

*Syntax:* `Mesh/material_id_name`

*Note:* The variable provided must be an elemental variable. It cannot be provided at the same time as `material_ids`.

### 6.7.2 depletion_id_name

*Description:* Name of the variable providing the depletion IDs (from the mesh file)

*Data type:* String

*Default value:* `<empty>`

*Syntax:* `Mesh/depletion_id_name`

*Note:* The variable provided must be an elemental variable. It cannot be provided at the same time as `depletion_ids`.

### 6.7.3 equivalence_id_name

*Description:* Name of the variable providing the equivalence IDs (from the mesh file)

*Data type:* String

*Default value:* `<empty>`

*Syntax:* `Mesh/equivalence_id_name`

*Note:* The variable provided must be an elemental variable. It cannot be provided at the same time as `equivalence_ids`. 
6.7.4  **subdomain_ids**

Description: Subdomain IDs present in the mesh

Data type: Vector of integers

Default value: `<empty>`

Syntax: `Mesh/subdomain_ids`

Note: This parameter should only be provided if the material, depletion and/or equivalence IDs should be set directly through the input file (and not from the mesh). In other words, it must be provided if (and only if) at least of the following three parameters is provided: `material_ids`, `depletion_ids` and `equivalence_ids`. In that case, it does not have to list all the block IDs (or subdomain IDs) contained in the mesh but invalid IDs will be assigned to the missing blocks.

6.7.5  **material_ids**

Description: Material IDs corresponding to ’subdomain_ids’ in the same order (ignored if material_id_name is provided)

Data type: Vector of integers

Default value: `<empty>`

Syntax: `Mesh/material_ids`

Note: If `material_id_name` is provided, this parameter should not be provided. Otherwise, it may be provided, in which case `subdomain_ids` should also be provided and have the same size.

6.7.6  **depletion_ids**

Description: Depletion IDs corresponding to ’subdomain_ids’ in the same order (ignored if depletion_id_name is provided)

Data type: Vector of integers

Default value: `<empty>`

Syntax: `Mesh/depletion_ids`

Note: If `depletion_id_name` is provided, this parameter should not be provided. Otherwise, it may be provided, in which case `subdomain_ids` should also be provided and have the same size.

6.7.7  **equivalence_ids**

Description: Equivalence IDs corresponding to ’subdomain_ids’ in the same order (ignored if equivalence_id_name is provided)

Data type: Vector of integers

Default value: `<empty>`

Syntax: `Mesh/equivalence_ids`

Note: If `equivalence_id_name` is provided, should not be provided. Otherwise, it may be provided, in which case `subdomain_ids` should also be provided and have the same size.
6.7.8  distribution

Refer to distribution in GeneratedBIDMesh.

6.7.9  partitioner

Refer to partitioner in GeneratedBIDMesh.

6.7.10  centroid_partitioner_direction

Refer to centroid_partitioner_direction in GeneratedBIDMesh.

6.7.11  file

Refer to file in FileMesh. One limitation is that only Exodus files are currently supported (unlike FileMesh).

6.8  PebbleMesh

PebbleMesh is used to support pebble tracking transport (PTT) calculations. Currently, the mesh needs to be pre-generated and saved into a mesh file. With this mesh, the user can indicate which subdomain in the mesh is the region to be filled with pebbles. The nodes in pebble subdomain will be associated with pebbles. Pebbles are differentiated by the cross sections assigned. Currently only the DFEM-SN transport scheme works with PebbleMesh. Pebble subdomains do not need multigroup transport materials.

6.8.1  distribution

Refer to distribution in GeneratedBIDMesh.

6.8.2  partitioner

Refer to partitioner in GeneratedBIDMesh.

6.8.3  centroid_partitioner_direction

Refer to centroid_partitioner_direction in GeneratedBIDMesh.

6.8.4  file

Refer to file in FileMesh.
6.8.5 *subdomains_with_pebble*

Description: Subdomains where pebbles are contained
Data type: Vector of subdomain names
Default value: <required>
Syntax: Mesh/subdomains_with_pebble

6.8.6 *non_pebble_node_set*

Description: Node set to indicate no pebbles for these nodes
Data type: Node set name
Default value: <empty>
Syntax: Mesh/non_pebble_node_set
Note: This parameter is needed currently, for example, for not considering boundary nodes of the pebble subdomain as pebbles. Users are required to build the node set. We might change this parameter into a logical value in the future if we can check the nodes with the code internally. If this parameter is empty, all nodes in the pebble subdomain including the nodes on the subdomain boundary will be associated with pebbles.

6.8.7 *pebble_radius*

Description: Pebble radius
Data type: Real
Default value: <required>
Syntax: Mesh/pebble_radius
Note: Currently all pebbles have the same radius.

6.8.8 *pebble_material_file*

Description: Pebble material file, each type of pebble has an entry
Data type: String
Default value: <required>
Syntax: Mesh/pebble_material_file
Note: A cross section file in INSTANT XML format for all types of pebbles. The IDs of the materials will be used as pebble types in *pebble_packing_expression* to assign pebbles with IDs. Cross sections for gap need to be included in this file as well.

6.8.9 *gap_material_id*

Description: Gap material ID to access the gap material in the pebble_material_file
Data type: Integer
Default value: <required>
Syntax: Mesh/gap_material_id
Note: Currently the gap material is uniformly applied to gaps everywhere in the pebble subdomain.
6.8.10  verbose

Description: To control screen outputs on how the pebble mesh is set up
Data type: Enumeration (/0/1/2/3/4/)
Default value: 0
Syntax: Mesh/verbose

6.8.11  pebble_packing_expression

Description: Expression to return pebble ID based on pebble coordinates
Data type: String
Default value: <required>
Syntax: Mesh/pebble_packing_expression
Note: This packing expression can be as simple as a constant value or an arbitrary expression returning the distribution of pebble IDs in the pebble domain. packing_constant_names and packing_constant_expressions can be used for building complicated expressions.

6.8.12  packing_constant_names

Description: Vector of constants used in the packing expression
Data type: Vector of strings
Default value: <empty>
Syntax: Mesh/packing_constant_names

6.8.13  packing_constant_expressions

Description: Vector of values for the constants in packing_constant_names (can be an FParser expression)
Data type: Vector of strings
Default value: <empty>
Syntax: Mesh/packing_constant_expressions

6.8.14  numerical_integration_on_element

Description: True to do numerical integration to evaluate the total elemental matrices
Data type: Logical
Default value: True
Syntax: Mesh/numerical_integration_on_element
Note: This parameter does not affect the calculation but only the setup.
6.9 *TiledMesh*

6.10 *ImageMesh*

6.11 **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, etc.

6.11.1 **MOOSE mesh modifiers**

![Figure 23 Moose mesh modifiers.](image-url)
The complete list of MOOSE mesh modifiers can be found in Fig. 23. Their parameters are fairly straight-forward and can be found in MOOSE documents. We found that the mesh extruder is frequently used so we also document it here.

6.11.2 NonuniformMeshExtruder

This modifier takes a 1D or 2D mesh and extrudes it to 2D or 3D, respectively. Triangles are extruded to (triangular) 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 the original number of sides. This modifier is an extension of MOOSE MeshExtruder whose parameter \( \text{num_layers} \) is replaced with two parameters, \( \text{layer_thickness} \) and \( \text{layer_subdivisions} \), to allow the extruded layers having non-uniform lengths. It has one extra constraint than the MOOSE MeshExtruder that the mesh to be extruded must have the same spatial and mesh dimension (which can be different e.g. meshed curve in 3D). This modifier works with distributed mesh.

6.11.2.1 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 important.

6.11.2.2 extrusion_vector

Description: The direction 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 second or third component must not be equal to zero for 1D or 2D mesh respectively. The L2 norm of this parameter must be equal to one but unlike the one in MOOSE MeshExtruder, which indicates how far the mesh is going to be extruded.

6.11.2.3 layer_thickness

Description: The thickness of all layers in the extruded mesh
Data type: Vector of reals
Default value: <required>
Syntax: MeshModifiers/*/layer_thickness
Note: All elements of this parameter must be greater than zero.
6.11.2.4  \textit{layer_subdivisions}

Description: The number of even subdivisions of all layers
Data type: Vector of integers
Default value: <empty>
Syntax: MeshModifiers/*/layer_subdivisions

Note: The size of this parameter must be equal to the size of \textit{layer_thickness} if it is provided. If this parameter is not provided, it is assumed to be a vector with all elements being one and with size of \textit{layer_thickness}. The summation of all elements in this parameter is equal to \( L \), the number of extruded layers.

6.11.2.5  \textit{bottom_sideset}

Description: The boundary name that will be applied to the bottom of the extruded mesh
Data type: String
Default value: <empty>
Syntax: MeshModifiers/*/bottom_sideset

6.11.2.6  \textit{top_sideset}

Description: The boundary name that will be applied to the top of the extruded mesh
Data type: String
Default value: <empty>
Syntax: MeshModifiers/*/top_sideset

6.11.2.7  \textit{existing_subdomains}

Description: The subdomains that will be remapped for specific layers
Data type: Vector of integers
Default value: <empty>
Syntax: MeshModifiers/*/existing_subdomains

6.11.2.8  \textit{layers}

Description: The layers where the \textit{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.
6.11.2.9  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×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.

6.11.3  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 \( x_i + r c h_i \), 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 \( (x_i + r_1 ch_i \cos(r_22\pi), y_i + r_1 ch_i \sin(r_22\pi)) \), 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 \( (x_i + r_1 ch_i \cos(r_22\pi) \sqrt{1 - r_2^2}, y_i + r_1 ch_i \sin(r_22\pi) \sqrt{1 - r_2^2}, z_i + r_1 ch_i r_3) \), for each node. Users can easily create truly unstructured meshes from a structured mesh using this mesh modifier. This modifier works with distributed mesh.

6.11.3.1  depends_on

Refer to depends_on in NonuniformMeshExtruder.

6.11.3.2  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 \( c \).

6.11.3.3  perturb_boundary

Description: Whether the boundary nodes are displaced
Data type: Logical
Default value: false
Syntax: MeshModifiers/*/perturb_boundary
6.11.3  seed

Description: Random seed for initializing random number sequence
Data type: Integer
Default value: 1
Syntax: MeshModifiers/*/seed

6.11.4 SplitConformingMeshForMortar

This mesh modifier splits a conforming mesh, i.e. a mesh without hanging nodes, for calculations with mortar FEM. This modifier will create (d-1)-dimensional blocks used as 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 (commonly referred to as block). This modifier 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 multischeme_option is mortar-split and a multischeme calculation, i.e. multiple transport systems, is pursued.

6.11.4.1 depends_on

Refer to depends_on in NonuniformMeshExtruder.

6.11.4.2 num_subdomains

Description: Number of subdomains for mortar FEM
Data type: Integer
Default value: <required>
Syntax: MeshModifiers/*/num_subdomains

6.11.4.3 subdomain_blocks

Description: Multiple semicolon separated groups of space 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 names. 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.
6.11.4.4 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.

6.11.5 AddSideSetsForSplitBoundary

This mesh modifier creates new side sets by splitting the existing side sets based on their connectivity to mesh subdomains. All sides in an existing side set whose attaching elements or master elements belong to the same subdomain will form a new side set with a name as '<existing side set name>_<subdomain name>_bnd'. If the existing side set does not have a name, its ID will be used. Subdomain names are indicated by the parameter subdomain_names. If all subdomains specified by the parameter subdomain_blocks do not cover the entire mesh, a new subdomain will be assumed with name being 'subdomain_<N>', where N is the number of subdomains in subdomain_names. If a side set is not connected with a subdomain, its sub side set will not be created. This modifier works with distributed mesh.

6.11.5.1 depends_on

Refer to depends_on in NonuniformMeshExtruder.

6.11.5.2 subdomain_blocks

Description: Multiple semicolon separated groups of space separated block names
Data type: Vector of strings
Default value: <required>
Syntax: MeshModifiers/*/subdomain_blocks

Note: Number of elements of this parameter is the number of subdomains N. Elements of this parameter are the comma separated block names. 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.

6.11.5.3 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 the size of subdomain_blocks. If this parameter is not given, all subdomains will be assigned with a default name as 'subdomain_<i>', i = 0, · · · , N − 1.
7 Functions

7.1 MOOSE Functions

A MOOSE function is a function in space and in time. It can be as simple as a parsed expression or as complicated as a solution function defined on an unstructured mesh.

7.1.1 MOOSE Functions in MOOSE Framework

The complete list of MOOSE functions in MOOSE can be found in Fig. 24. Their parameters can be found in MOOSE documents. Because the functionalities of these functions and their parameters are fairly straightforward, 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. 25.
7.1.2.1 \textit{timep}

Description: The time points
Data type: Vector of reals
Default value: \texttt{<required>}
Syntax: \texttt{Functions/*/timep}

7.1.2.2 \textit{value}

Description: The function values at all time points
Data type: Vector of reals
Default value: \texttt{<required>}
Syntax: \texttt{Functions/*/value}
Note: The size of this parameter must be equal to the size of \textit{timep}.

7.1.3 \textit{StepFunction}

This function is a simpler version of \textit{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. 26.

Figure 26  
Piece-wise constant function by \textit{StepFunction}. 

(a) left continuous  
(b) right continuous

215
7.1.3.1 \textit{timep}

Description: The time points where steps happen
Data type: Vector of reals
Default value: \texttt{<required>}
Syntax: Functions/*/timep

7.1.3.2 \textit{value}

Description: The constant values of all time steps
Data type: Vector of reals
Default value: \texttt{<required>}
Syntax: Functions/*/value

7.1.3.3 \textit{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 Functions

Transport 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 \textit{Functions} input block. Rattlesnake has several built-in transport functions that users can directly use.

7.2.1 \textit{ConstantSourceFunction}

This function specifies a function which is constant in space, time, angle and in every individual energy group. The function could have different value in different energy groups.

\[
f_g(\vec{r}, \vec{\Omega}, t) = \frac{a_g}{c_d}, \quad (26)
\]

where \(a_g\) is the scalar value of energy group \(g\); \(c_d\) is the dimension-dependent normalization:

\[
c_d = \begin{cases} 
2, & 1D \\
2\pi, & 2D \\
4\pi, & 3D
\end{cases} \quad (27)
\]

This function can be typically constructed and used inline with the syntax \(a_1, a_2, \cdots, a_G\).
7.2.1.1  \textit{NG}

Description: Number of energy groups
Data type: Integer
Default value: <required>
Syntax: Functions/*/NG

7.2.1.2  \textit{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 \textit{NG}.

7.2.2  \textit{PulsedSourceFunction}

This function specifies an 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.

\[ f_g(\vec{r}, \Omega, t) = \frac{1}{c_d} a_g e^{-\frac{|\vec{r} - \vec{r}_0|^2}{2c^2}} \delta(t - t_0), \] (28)

where $a_g$ is the strength of energy group $g$; $c_d$ is the dimension-dependent normalization; $c$ is the standard deviation of the Gaussian distribution; $t_0$ is the time at which the pulse occurs.

7.2.2.1  \textit{NG}

Refer to \textit{NG} in \textit{ConstantSourceFunction}.

7.2.2.2  \textit{value}

Refer to \textit{value} in \textit{ConstantSourceFunction}.

7.2.2.3  \textit{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.
7.2.2.4 constant

Description: The standard deviation of the gaussian
Data type: Real
Default value: 1
Syntax: Functions/*/constant

7.2.2.5 pulse_time

Description: The time $t_0$ when the pulse occurs.
Data type: Real
Default value: 0 (for numerical purposes set to 1.0e-15)
Syntax: Functions/*/pulse_time

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.

$$f_g(\vec{r}, \vec{\Omega}, t) = a_g \delta(\vec{\Omega} - \vec{\Omega}_m),$$

(29)

where $a_g$ 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 NG

Refer to NG in ConstantSourceFunction.

7.2.3.2 strength

Description: Strength of the source in each energy group
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 FilePNTransportFunction

This function turns a $P_N$ solution into a transport function.

$$f_g(\vec{r}, \vec{\Omega}, t) = \begin{cases} 
\sum_{\ell=0}^{P_N} \frac{2\ell+1}{2} P_\ell(\mu) \phi_{g,\ell}(\vec{r}, t), & 1D \\
\sum_{\ell=0}^{P_N} \sum_{m=0}^{\ell} \frac{2\ell+1}{4\pi} Y_{\ell,m}(\vec{\Omega}) \phi_{g,\ell,m}(\vec{r}, t), & 2D \\
\sum_{\ell=0}^{P_N} \sum_{m=-\ell}^{\ell} \frac{2\ell+1}{4\pi} Y_{\ell,m}(\vec{\Omega}) \phi_{g,\ell,m}(\vec{r}, t), & 3D 
\end{cases}$$

(30)

where $P_N$ is the PN order; $P_\ell$ are the Legendre polynomials; $\mu$ is the cosine of the polar angle; $Y_{\ell,m}$ are the real spherical harmonics; $\phi_{\ell}$ or $\phi_{\ell,m}$ are the angular flux moments.

218
7.2.4.1  \textit{file\_name}

Description: The solution file (either XDA or Exodus)
Data type: String
Default value: \textlt{required}\textgt
Syntax: Functions/*/file\_name

Note: This parameter specifies the file in which the solution is stored. The file is typically in XDA or Exodus format that support the storage of the solutions.

7.2.4.2  \textit{es}

Description: The name of the file holding the equation system info in XDA format (XDA only)
Data type: String
Default value: \textlt{empty}\textgt
Syntax: Functions/*/es

Note: this parameter is only required if \textit{file\_name} is an XDA file. Refer to MOOSE documentation about \textit{SolutionUser-Object} for more information.

7.2.4.3  \textit{NG}

Refer to \textit{NG} in \textit{ConstantSourceFunction}.

7.2.4.4  \textit{weighting\_type}

Description: Type of weighting for discontinuous variables on an element side
Data type: Enumeration (/none/inside/outside/average/)
Default value: none
Syntax: TransportSystems/*/weighting\_type

Note:
This option controls where the data provided in \textit{file\_name} is evaluated. The default value is the most efficient if the function is continuous where the evaluation is to take place. In most cases, this parameter should therefore not be modified. If the solution is discontinuous where it is to be evaluated (e.g. across an interface), the other options are available. The option \textit{weighting\_type} = inside means that the function evaluates the variables contained in \textit{file\_name} in the vicinity of the interface within the current element. The option \textit{weighting\_type} = outside is similar but within the neighboring element. Finally, \textit{weighting\_type} = average takes the average of the previous two options.

7.2.4.5  \textit{eps}

Description: Tolerance to evaluate the function slightly outside the boundary
Data type: Real
Default value: $2 \times 10^{-6}$
Syntax: TransportSystems/*/eps
Note: This parameter should be strictly positive. It is used if \textit{weighting\_type} is different from \textit{none}. It specifies how far away from the boundary the values of the function should be taken: in practice, that distance is equal to $\epsilon$ times the distance of the point to the centroid element. It was noticed that setting it to $10^{-6}$ or less may not be enough to see a difference when changing \textit{weighting\_type}.

\subsection*{7.2.4.6 \textit{max\_moment\_degree\_to\_read}}

Description: Degree up to which the moments from the (exodus or xda) file should be read

Data type: Positive integer

Default value: 1

Syntax: \texttt{TransportSystems/*/max\_moment\_degree\_to\_read}

Note: This option forces moments up to this degree to be read from the exodus file to reconstruct a more accurate angular flux on the boundary (potentially useful for $S_N$ calculations). It requires those moments to be present in \texttt{file\_name} (refer to \texttt{NA} to see how to force it in the case the solution is generated by a $S_N$ calculation).

\subsection*{7.2.4.7 \textit{prefix}}

Description: The prefix for naming the flux moments

Data type: String

Default value: <empty>

Syntax: \texttt{TransportSystems/*/prefix}

Note: This option is only used if the flux moment variables in \texttt{file\_name} have a prefix (to avoid naming conflicts). The prefix must then be manually inputted through this parameter.

\subsection*{7.2.4.8 \textit{csphase}}

Description: Whether or not the Condon-Shortley phase is included in the function

Data type: Logical

Default value: False

Syntax: \texttt{Functions/*/csphase}

Note: If the solution is generated by Rattlesnake, the Condon-Shortley phase is not included for the evaluation of angular flux moments, which is why the default value of this parameter is false.

\subsection*{7.2.4.9 \textit{scale\_factor}}

Description: the scale factor (a) is applied to the solution (x) as follows: $ax + b$

Data type: Real

Default value: 1

Syntax: \texttt{Functions/*/scale\_factor}
7.2.4.10  add_factor

Description: the value (b) is added to the solution (x) as follows: \( ax + b \)

Data type: Real

Default value: 0

Syntax: Functions/*/add_factor

7.2.4.11  timestep

Description: Index of the single timestep used

Data type: String

Default value: <empty>

Syntax: Functions/*/timestep

Note: LATEST can be used for the last timestep (ExodusII file only). If not supplied, time interpolation will occur. When the ExodusII file to be loaded is generated by a steady-state calculation, there could be multiple timesteps in the file and the associated times are only for outputting purpose rather than the real constant time. It is thus desired to load the solution at a particular timestep directly instead of going through the time interpolation.

7.2.5  Customized TransportFunction (Advanced)

1. Determine the appropriate base class: PN, SN transport function;
2. Implement your function: using Larsen_2D as an example;
3. Register your function in RattleSnakeApp;
4. Use your function as other transport functions;
5. Consider adding it into Rattlesnake officially.

7.3  Adjustable Function

AdjustableFunction is a MOOSE function, identical to MOOSE ConstantFunction except that

- Parameter ‘value’ is renamed as ‘InitialParam’;
- It has two additional methods:

```cpp
Real AdjustableFunction::getParameter();
void AdjustableFunction::setParameter(Real v);
```
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 \( \theta \) or the cosine of the scattering angle \( \mu = \cos(\theta) \). It satisfies the normalization condition:

\[
\frac{1}{2\pi} \int_{2\pi} \int_{-1}^{1} p(\mu) d\mu d\phi = 1, \tag{31}
\]

or

\[
\int_{-1}^{1} p(\mu) d\mu = 1, \tag{32}
\]

There are two ways of using a phase function: either by taking its moments or by evaluating it at particular directions. These are respectively referred to as the \( P_N \) and \( S_N \) treatments. While the \( P_N \) treatment is the natural choice for \( P_N \) methods, both can make sense for \( S_N \) methods (the former is typically more appropriate for weakly anisotropic scattering, the latter for strongly anisotropic scattering). The choice between both treatments is enforced through Materials/*\texttt{/phase_function_treatment}.

More phase functions can be implemented, by having them derive from \texttt{PhaseFunction}.

7.4.1 IsotropicPhaseFunction

This phase function is independent of \( \mu \), i.e. \( p(\mu) = \frac{1}{2} \).

7.4.1.1 integration_option

Description: Option for the integration of the phase function
Data type: Enumeration (/sampling/patch/)
Default value: sampling
Syntax: Functions/*/integration_option

Note: This option only makes sense for the \( S_N \) treatment (i.e. is ignored for the \( P_N \) treatment). In that case, the simplest way is to evaluate the phase function along specific directions. A potential problem is that it may not be conservative if the angular quadrature does not exactly integrate the phase function. An alternative is then to perform the integration of the phase function over a small patch of the unit sphere. These first and second ways are obtained with the \texttt{sampling} and \texttt{patch} options, respectively.

7.4.1.2 aqdata

Description: Angular quadrature user object (only needed to store the values of the phase functions)
Data type: String
Default value: <empty>
Syntax: Functions/*/aqdata
Note: This option only makes sense for the $S_N$ treatment (i.e. is ignored for the $P_N$ treatment). In that case, it can be useful to precompute the direction-to-direction matrix to avoid having to recompute the (potentially costly, especially in 1-D where an integration of the phase function over the azimuthal angle is performed) evaluation of the phase function multiple times. In order to do so, the $S_N$ angular quadrature need somehow be provided. This can be done in two ways: either by directly setting this parameter or, if the action system is used, by setting $precompute$ to true.

### 7.4.1.3 precompute

**Description:** Set to true to have the action system automatically set the $aqdata$

**Data type:** Boolean

**Default value:** False

**Syntax:** Functions/*\precompute

Note: This option only makes sense for the $S_N$ treatment (i.e. is ignored for the $P_N$ treatment) and while using the action system. In that case, if the phase function values are to be precomputed, the user may not have access to $aqdata$, which can be resolved by setting this parameter to true.

### 7.4.2 RayleighPhaseFunction

This Rayleigh phase function is $p(\mu) = \frac{3}{4}(1 + \mu^2)$.

#### 7.4.2.1 integration_option

Refer to $integration_option$ in IsotropicPhaseFunction.

#### 7.4.2.2 aqdata

Refer to $aqdata$ in IsotropicPhaseFunction.

#### 7.4.2.3 precompute

Refer to $precompute$ in IsotropicPhaseFunction.

### 7.4.3 TruncatedDiracPhaseFunction

This phase function corresponds to the Dirac phase function truncated at a given degree $L$ along a particular direction $\tilde{\mu}$ given by $p(\mu) = \delta(\tilde{\mu} - \mu) \approx \sum_{\ell=0}^{L} \frac{2\ell+1}{2} P_\ell(\tilde{\mu}) P_\ell(\mu)$.

#### 7.4.3.1 integration_option

Refer to $integration_option$ in IsotropicPhaseFunction.

#### 7.4.3.2 aqdata

Refer to $aqdata$ in IsotropicPhaseFunction.
7.4.3.3 **precompute**

Refer to *precompute* in IsotropicPhaseFunction.

7.4.3.4 **Lmax**

Description: Truncation degree for approximating the moments with Legendre polynomials (i.e. higher moments set to zero)

Data type: Unsigned integer

Default value: `<required>`

Syntax: Functions/*/Lmax

7.4.3.5 **mu_tilde**

Description: Value of $\tilde{\mu}$, i.e. the value of $\mu$ for which the (non-truncated) Dirac delta function is non-zero

Data type: Real

Default value: `<required>`

Syntax: Functions/*/mu_tilde

7.4.4 **HenyeyGreensteinPhaseFunction**

This Henyey-Greenstein phase function is

\[ p(\mu) = \frac{(1-g^2)}{2(1+g^2-2g\mu)^{\frac{3}{2}}} \]

Its Legendre expansion is given by

\[ p(\mu) = \sum_{n=0}^{\infty} \frac{2n+1}{2} g^n P_n(\mu) \]

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.

7.4.4.1 **g**

Description: Henyey-Greenstein parameter $g$

Data type: Function name

Default value: 0

Syntax: Functions/*/g

Note: This parameter can be the name of a general function of time and space.

7.4.4.2 **integration_option**

Refer to *integration_option* in IsotropicPhaseFunction.

7.4.4.3 **aqdata**

Refer to *aqdata* in IsotropicPhaseFunction.
7.4.4.4 **precompute**

Refer to *precompute* in *IsotropicPhaseFunction*.

7.4.4.5 **Lmax**

Description: Maximum degree to be used for the $P_N$ treatment of the phase function

Data type: Unsigned integer

Default value: `<required>`

Syntax: Functions/*/Lmax

Note: All the scattering moments of degree higher than Lmax are ignored by the action system.

7.4.5 **HeavisidePhaseFunction**

This HeavisidePhaseFunction phase function is $p(\mu) = 1$ if $\mu > 0$ and 0 otherwise.

7.4.5.1 **integration_option**

Refer to *integration_option* in *IsotropicPhaseFunction*.

7.4.5.2 **aqdata**

Refer to *aqdata* in *IsotropicPhaseFunction*.

7.4.5.3 **precompute**

Refer to *precompute* in *IsotropicPhaseFunction*.

7.4.5.4 **Lmax**

Description: Maximum degree to be used for the $P_N$ treatment of the phase function

Data type: Unsigned integer

Default value: `<required>`

Syntax: Functions/*/Lmax

Note: All the scattering moments of degree higher than Lmax are ignored by the action system.
8 Transport 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, commonly encountered in benchmark problems where cross sections are given in the problem description.
- **ConstantMatIDNeutronicsMaterial**: same as **ConstantNeutronicsMaterial** where the material IDs are read from the mesh, allowing in particular to have one material assigned to multiple material IDs.
- **FunctionNeutronicsMaterial**: suitable for calculation where macroscopic cross sections are functions of space and time. It is typically used for MMS (method of manufactured solutions) or for transient benchmark problems.
- **MixedNeutronicsMaterial**: Allow the user to select a single entry (state point) from a YAKXS table and use it for the problem. Useful for checking cross sections in YAKXS format.
- **MixedMatIDNeutronicsMaterial**: same as **MixedNeutronicsMaterial** where the material IDs are read from the mesh, allowing in particular to have one material assigned to multiple material IDs.
- **CoupledFeedbackNeutronicsMaterial**: suitable for multi-physics or depletion calculations where tabulated cross sections are required.
- **CoupledFeedbackMatIDNeutronicsMaterial**: same as **CoupledFeedbackNeutronicsMaterial** where the material IDs are read from the mesh, allowing in particular to have one material assigned to multiple material IDs.
- **CRoddedNeutronicsMaterial**: suitable for modeling control-rod movements if some other constant neutronics materials are present.
- **MixedRoddedNeutronicsMaterial**: suitable for modeling control-rod movements if single state points from a YAKXS file are used; no interpolation of cross sections between state points.
- **ConstantTRMaterial**: suitable for calculation with fixed macroscopic cross sections, like in benchmark problems where cross sections are given by the problem description.
- **FunctionTRMaterial**: suitable for calculation with macroscopic cross sections being spatial and time functions. It is typically used for MMS (method of manufactured solutions) or for transient benchmark problems.

Several types of materials can be used alongside each other in a single discretization scheme. For computing Larsen Trahan diffusion coefficients some materials are forbidden. 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. In multischeme settings a single neutronics materials cannot be shared by two discretization schemes; this is even true if the two schemes are identical.
2. Neutronics materials cannot be used without defining a transport system.

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 Rattlesnake. It provides an XML cross section library format for storing multigroup cross sections. YAKXS provides operations like cross section interpolation, mixing, fitting, collapsing, etc. Users and developers are insulated from the complexity of these operations and will interact with YAKXS’ high-level interfaces.

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 an 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. However, mixture 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. If a 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 constant macroscopic cross sections to transport solvers. 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.
Figure 27  Graphical view of YAKXS.
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. 27. Rattlesnake neutronics material interact with YAKXS for loading the cross sections and evaluate the material properties required for neutron transport calculations.

8.2.2 Material properties declared by neutronics materials

All neutronics materials declare a set of material properties, which are listed in Table 29. 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 if 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<1> and neutron velocities are evaluated on element faces for SAAF-CFEM-PN and LS-CFEM-PN.

8.2.3 Common parameters for all neutronics materials

All neutronics materials share the following common parameters.

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. In the remainder we will refer to this discretization scheme simply as discretization scheme.

8.2.3.2 group_bounds

Description: Group bounds
Data type: Vector of reals
Default value: <empty>
Syntax: Materials/*/group_bounds
Note: The size of group bounds must be equal to the number of groups plus one. Group bounds are also required for flux-corrected group condensation, i.e. coarse_group_index is provided with size being greater than the number of groups and flux-corrected condensation is enabled by flux_corrected_condensation. Group bounds must be strictly descending for neutronics materials.
Table 29 Neutronics material properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Type</th>
<th>Declared if</th>
</tr>
</thead>
<tbody>
<tr>
<td>neutron_speed_g</td>
<td>$v_g$</td>
<td>Real</td>
<td>Fissile</td>
</tr>
<tr>
<td>neutron_speed</td>
<td>$v_{g_i}$</td>
<td>Real</td>
<td>Fissile</td>
</tr>
<tr>
<td>diffusion_coefficient_g</td>
<td>$D_g$</td>
<td>Real</td>
<td>Diffusion</td>
</tr>
<tr>
<td>vector_diffusion_coefficient_g</td>
<td>$D_{g_i}$</td>
<td>Real</td>
<td>Diffusion</td>
</tr>
<tr>
<td>tensor_diffusion_coefficient_g</td>
<td>$D_{g_{ij}}$</td>
<td>RealTensor</td>
<td>Diffusion</td>
</tr>
<tr>
<td>sigma_removal_g</td>
<td>$\sigma_{g_i}$</td>
<td>Real</td>
<td>Diffusion</td>
</tr>
<tr>
<td>sigma_total_g</td>
<td>$\sigma_{g_i}$</td>
<td>Real</td>
<td>NonDiffusion</td>
</tr>
<tr>
<td>sigma_total</td>
<td>$\sigma_{g_i}$</td>
<td>Real</td>
<td>NonDiffusion</td>
</tr>
<tr>
<td>sigma_scattering_g</td>
<td>$\sigma_{g_i,G_l}$</td>
<td>Real</td>
<td>ScatteringPattern #</td>
</tr>
<tr>
<td>sigma_scattering_total</td>
<td>$\sigma_{g_i,G}$</td>
<td>Real</td>
<td>NonzeroScattering #</td>
</tr>
<tr>
<td>phase_function_g</td>
<td>$p_{p-g}()$</td>
<td>PhaseFunction pointer</td>
<td>UsingPhaseFunction #</td>
</tr>
<tr>
<td>phase_func_g</td>
<td>$\sigma_{g_i,g_l}$</td>
<td>Real</td>
<td>UsingPhaseFunction #</td>
</tr>
<tr>
<td>&lt;MaterialName&gt;_density</td>
<td>$\rho = \frac{1}{\text{det}(V u + 1)}$</td>
<td>Real</td>
<td>Always #</td>
</tr>
</tbody>
</table>

# Transient means that equation_type is equal to transient;
# Diffusion means that scheme is equal to CFEM-Diffusion or DFEM-Diffusion;
# NonDiffusion means that scheme is equal to any other than CFEM-Diffusion and DFEM-Diffusion;
# ScatteringPattern means that the property will be declared for the non-zero scattering entries;
# NonzeroScattering means that there is scattering for this material. For diffusion calculations, this means that there is cross-group scatterings because in-group scatterings are merged into total as removal cross sections and thus never used;
# If phase_function_names is not empty and phase_function_treatment is SN;
# <MaterialName> is the name of the material; if displacement variables are not coupled in, this material property is equal to one;
# Fissile means that the material is fissile;
# Plus means that plus is set to true;
# FissionPattern means that the property will be declared for the group with non-zero fission neutron yields;
# $I > 0$ means that n_delay_groups is greater than 0; Note that the material has to be fissile to provide delayed neutron data;
# Provided means that the cross section data is provided by the input;
8.2.3.3  **scalar_fluxes**

Description: Scalar fluxes of all groups  
Data type: Vector of strings  
Default value: `<empty>`  
Syntax: Materials/*/scalar_fluxes  

Note: This parameter is required by /sum/Larsen/LP/max/ *diffusion_limiter* and flux-corrected group condensation (*coarse_group_index* is provided with size being greater than the number of groups and flux-corrected condensation is enabled by *flux_corrected_condensation*). Scalar fluxes must be ordered with the group index. The size of this parameter must be equal to the number of groups.

8.2.3.4  **volume_adjuster**

Description: User data object that applies the volume adjustment factor calculation on subdomain-base  
Data type: string  
Default value: `<empty>`  
Syntax: Materials/*/volume_adjuster  

Note: Meshes do not perfectly conserve the volume of the geometric objects they represent. To conserve total fissionable mass in a geometry, these correction factors can be used.

8.2.3.5  **speed_required**

Description: True to force the evaluation of radiation speed  
Data type: Logical  
Default value: False  
Syntax: Materials/*/speed_required  

Note: If the cross section data do not have radiation speed and this parameter is set to true, an error will be thrown by Rattlesnake.

8.2.3.6  **is_meter**

Description: Whether or not mesh is in units of meter  
Data type: Logical  
Default value: False  
Syntax: Materials/*/is_meter

8.2.3.7  **dump_prop**

Description: Whether or not to dump materials on QPs  
Data type: Logical  
Default value: False
Syntax: Materials/*/dump_prop

Note: This parameter should only be used for debugging purpose. Dumping material properties on quadrature points will write files and slow down the overall calculation. It is highly recommended to use dump_prop_at_time, dump_prop_at_residual, dump_prop_on_elements, dump_prop_on_point or dump_prop_on_first_element to restrict the dumping. If dumping happens, the cross sections will be dumped into a file named as <output>.xml, where output is the input file name without its file type. Also a strict check on the cross sections will be performed during dumping, which provides a way to investigate potential issues in cross section evaluations. Rattlesnake will output a screen message when dumping happens. When Rattlesnake runs with multiple processors, only processors having any of elements where the material is defined will output the message. Because Rattlesnake by default only performs outputs on master processor, you may not see the message if Rattlesnake runs with multiple processors.

8.2.3.8 dump_prop_at_time

Description: Dump materials on QPs at a particular time
Data type: Real
Default value: <empty>
Syntax: Materials/*/dump_prop_at_time
Note: This parameter restricts the dumping at a particular time. Empty means no time restriction on dumping, i.e. dumping at all times. It is active only when dump_prop is true.

8.2.3.9 dump_prop_at_residual

Description: Dump materials on QPs for a particular residual evaluation on a time step. This parameter is the residual evaluation that the dumping should occur for.
Data type: integer
Default value: <empty>
Syntax: Materials/*/dump_prop_at_residual
Note: This parameter restricts the dumping at a particular residual evaluation, which is numbered starting from 1. Residual evaluation counter will be reset at the beginning of each time step. Empty means no residual restriction on dumping, i.e. dumping at all residual evaluations. It is active only when dump_prop is true.

8.2.3.10 dump_prop_on_elements

Description: Dump materials on a list of elements with their IDs.
Data type: Vector of integers
Default value: <empty>
Syntax: Materials/*/dump_prop_on_elements
Note: This parameter is active only when dump_prop is true.

8.2.3.11 dump_prop_on_point

Description: Dump materials on the element close to a point for the residual evaluation specified in dump_prop_at_residual and the time specified in dump_prop_at_time.
Data type: Vector of reals
8.2.3.12  dump_prop_on_first_element

Description: True to dump material properties on the first element of this material
Data type: Logical
Default value: false
Syntax: Materials/*/dump_prop_on_first_element

Note: Elements where the material is defined could be distributed on multiple processors. In such a case, cross sections of all the first local elements of all processors of this material will be dumped.

8.2.3.13  output

Description: The file base used to dump debug information
Data type: String
Default value: ‘mat’
Syntax: Materials/*/output

8.2.3.14  dbgmat

Description: To turn on the debug info of reading the data
Data type: Logical
Default value: false
Syntax: Materials/*/dbgmat

8.2.3.15  diffusion_coefficient_scheme

Description: Scheme to obtain diffusion coefficient
Data type: Enumeration (/user_supplied/local/nonlocal/)
Default value: user_supplied
Syntax: TransportSystems/diffusion_coefficient_scheme

Note: This parameter is used for diffusion calculations. In transport update for doing nonlinear diffusion acceleration, this parameter affects the evaluation of drift vectors. This parameter must be consistent for diffusion calculations and transport updates when doing nonlinear diffusion acceleration. Diffusion coefficients must be available when user_supplied option is selected with the only exception of those rodded neutronics materials. In those rodded materials, it is desired that the materials evaluate the derived diffusion coefficients first before doing the weighted averaged for elements containing multiple rod segments.
8.2.3.16  nonlocal_diffusion_threshold

Description: Nonlocal diffusion coefficient is used only when the element local mean free path ($\Sigma_t g \ast h_{max}$) is below this threshold

Data type: Real

Default value: Maximum real number

Syntax: TransportSystems/nonlocal_diffusion_threshold

Note: This parameter is active only when diffusion_coefficient_scheme is set to nonlocal. $h_{max}$ is the maximum vertex separation of the local element. Otherwise, local derived diffusion coefficients will be evaluated assuming diffusion_coefficient_scheme to be local. Default value basically means that non-local diffusion coefficients will be always used.

8.2.3.17  larsen_trahan_fluxes

Description: The fluxes used for computing nonlocal diffusion coefficients

Data type: Vector of variable names

Default value: <Empty>

Syntax: TransportSystems/larsen_trahan_fluxes

Note: This parameter is active and must be provided only when diffusion_coefficient_scheme is set to nonlocal. The size of this parameter must be equal to $G \times d \times (d + 1)/2$, where $G$ is the number of energy groups and $d$ is the mesh dimension.

8.2.3.18  diffusion_limiter

Description: Flux limiter on evaluating the local diffusion coefficients

Data type: Enumeration (/none/sum/Larsen/LP/max/)

Default value: none

Syntax: TransportSystems/diffusion_limiter

Note: This parameter is active only when diffusion_coefficient_scheme is set to local or non-local diffusion coefficients are switched to local due to nonlocal_diffusion_threshold. Equation of diffusion coefficient $D_g$ is

$$D_g = \begin{cases} 
\min(D_{g,\text{raw}}, c_1), & \text{none} \\
D_{g,\text{raw}} + c_2 s_g, & \text{sum} \\
\left(D_{g,\text{raw}} - c_3 s_g \right)^{-1/c_3}, & \text{Larsen} \\
\left(\frac{1}{\tanh(3s_g D_{g,\text{raw}})} - \frac{1}{3s_g D_{g,\text{raw}}} \right) \frac{1}{s_g}, & \text{LP, } s_g = 0 \\
\frac{1}{\max(s_g, 1/D_{g,\text{raw}})}, & \text{LP, } s_g > 0 \\
\max(s_g, 1/D_{g,\text{raw}}), & \text{max}
\end{cases}$$  \hspace{1cm} (33a)

where

$$D_{g,\text{raw}} = \frac{c_4}{\Sigma_{tr,g}},$$  \hspace{1cm} (33b)

$$\Sigma_{tr,g} = \Sigma_{t,g} - \Sigma_{n_1,g}^{-g},$$  \hspace{1cm} (33c)

$$s_g = \begin{cases} 
\|\nabla \Phi_g\|, & \Phi_g > 0 \\
\frac{\Phi_g}{\Phi_g}, & \Phi_g \leq 0
\end{cases}$$  \hspace{1cm} (33d)
$c_1$ is the maximum diffusion coefficient, which can be set by \textit{maximum_diffusion_coefficient}. $c_2$ is the flux slope factor, which can be set by \textit{slope_factor}. $c_3$ is called as Larsen limit factor, which can be set by \textit{larsen_limit_factor}. $c_4$ is the factor applied to the inverse of the transport cross section, which can be set by \textit{diffusion_coefficient_factor}.

\subsection*{8.2.3.19 \textit{diffusion_coefficient_factor}}

Description: Factor applied to the inverse of the transport cross sections  
Data type: Real  
Default value: 1/3  
Syntax: TransportSystems/diffusion_coefficient_factor  
Note: This parameter is active only when \textit{diffusion_coefficient_scheme} is set to \textit{local}. This parameter must be greater than and equal to 0.1 and must be less than and equal to 10.

\subsection*{8.2.3.20 \textit{maximum_diffusion_coefficient}}

Description: The value of the local-derived diffusion coefficient is limited to this value  
Data type: Real  
Default value: Maximum real number  
Syntax: TransportSystems/maximum_diffusion_coefficient  
Note: This parameter is active only when \textit{diffusion_coefficient_scheme} is set to \textit{local} and \textit{diffusion_limiter} is \textit{none}.

\subsection*{8.2.3.21 \textit{slope_factor}}

Description: Factor on the slope for sum limitter  
Data type: Real  
Default value: 1  
Syntax: TransportSystems/slope_factor  
Note: This parameter is active only when \textit{diffusion_coefficient_scheme} is set to \textit{local} and \textit{diffusion_limiter} is \textit{sum}.

\subsection*{8.2.3.22 \textit{larsen_limit_factor}}

Description: Factor for Larsen limitter  
Data type: Enumeration (/2/4/)  
Default value: 2  
Syntax: TransportSystems/larsen_limit_factor  
Note: This parameter is active only when \textit{diffusion_coefficient_scheme} is set to \textit{local} and \textit{diffusion_limiter} is \textit{Larsen}.  

235
8.2.3.23  *phase_function_names*

Description: Phase function names  
Data type: Vector of strings  
Default value: `<empty>`  
Syntax: `Materials/*/phase_function_names`

Note: If this parameter is not empty, scattering with phase functions will be included in the calculation. This is in addition to (i.e. does not exclude) the traditional scattering treatment. Refer to Phase Functions for more information about scattering phase functions.

8.2.3.24  *phase_function_treatment*

Description: Which treatment should be used using the phase functions  
Data type: Enumeration (`/PN/SN/`)  
Default value: PN  
Syntax: `Materials/*/phase_function_treatment`

Note: The `S_N` and `P_N` treatment are chosen by setting this parameter to SN or PN, respectively. Refer to Phase Functions for more information. This parameter is ignored if `phase_function_names` is empty. To date, the SN treatment is only available with the SAAF-CFEM-SN scheme.

8.2.3.25  *phase_function_departing_groups*

Description: Departing groups of phase functions  
Data type: Vector of integers  
Default value: `<empty>`  
Syntax: `Materials/*/phase_function_departing_groups`

Note: The size of this parameter must be equal to the size of `phase_function_names`.

8.2.3.26  *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_departing_groups`, and `phase_function_arriving_groups` are lined up. The pairs which are not specified in `phase_function_departing_groups` and `phase_function_arriving_groups` are assumed to be represented by isotropic scattering (This should be zero?).
8.2.3.27  **coarse_group_index**

Description: Coarse group index for all fine groups for group condensation

Data type: Vector of integers

Default value: `<empty>`

Syntax: Materials/*/coarse_group_index

Note: The size of this parameter is the number of fine groups in the cross section data and must be greater than the number of groups. Group indices in this parameter must be strictly ascending by at most one. Last value in this parameter must be equal to the number of groups. When this parameter is given and its size is greater than the number of groups, on-the-fly group condensation will be enabled.

8.2.3.28  **flux_corrected_condensation**

Description: True to use fluxes to correct the condensation spectrum

Data type: Logical

Default value: True

Syntax: Materials/*/flux_corrected_condensation

Note: This parameter is activated only when group condensation is enabled, i.e. `coarse_group_index` is provided with size being greater than the number of groups. When this parameter is false, the flux spectrum provided by the data will be used to condense the cross sections. The flux spectrum is not altered by the scalar fluxes, thus referred as the static group condensation.

8.2.3.29  **spectrum_prolongation_option**

Description: Spectrum prolongation option for on-the-fly flux-corrected group condensation

Data type: Enumeration (`/LinearESimple/LogESpline/`)

Default value: LogESpline

Syntax: Materials/*/spectrum_prolongation_option

Note: This parameter is activated only when group condensation is enabled (`coarse_group_index` is provided with size being greater than the number of groups) and `flux_corrected_condensation` is set to true. This parameter controls how the coarse-group fluxes are prolonged onto fine group structures and thus affects how the static flux spectrum is altered by the fluxes.

8.2.3.30  **plus**

Description: To indicate if absorption, fission and kappa fission are to be evaluated

Data type: Logical

Default value: False

Syntax: Materials/*/plus
8.2.3.3 | **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 e.g. evaluating the partial solutions from the fission event in a part of the solution domain.

8.2.4 | **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.

8.2.4.1 | **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_coeff, sigma_r, sigma_f, kappa_sigma_f, L* and *fissile*, while *library_file* is deactivated. The activated/deactivated parameters are reversed if this parameter is true.

8.2.4.2 | **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 if *fromFile* is true. It is used for dumping the material into a INSTANT XML file if *fromFile* is false and *dbgmat* is true.

8.2.4.3 | **library_file**

Description: The INSTANT XML XS file name  
Data type: string  
Default value: <empty>
Syntax: Materials/*/library_file

Note: This parameter must be provided if fromFile is true. Refer to [2] 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.

8.2.4.4 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 if fromFile is false and one of the following is true:

- and Discretization Schemes of the subdomains where this material is defined on is not CFEM-Diffusion
- and Discretization Schemes of the subdomains where this material is defined on is not DFEM-Diffusion
- parameter transport_wrapper is supplied
- parameter transport_multiapp_file is supplied.

The size of this parameter must be equal to the number of coarse groups of the discretization scheme given by G. diffusion_coef and sigma_r will be ignored if this parameter is in action.

8.2.4.5 diffusion_coef

Description: Constant diffusion coefficients
Data type: Vector of reals
Default value: <empty>
Syntax: Materials/*/diffusion_coef
Note: This parameter must be provided if 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 if this parameter is in action.

8.2.4.6 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 in lieu of the total cross section if diffusion calculations are performed.
8.2.4.7  $L$

Description: Order of scattering anisotropy

Data type: Integer

Default value: 0

Syntax: Materials/*/L

Note: This parameter is activated if $fromFile$ is false. If this parameter is larger than $NA$ provided in the discretization scheme ($NA$ default is zero), expansion terms beyond $L$ are ignored. Higher order scattering (beyond order 0) will always be ignored for diffusion schemes, CFEM-Diffusion and DFEM-Diffusion.

8.2.4.8  $sigma_s$

Description: Constant scattering cross section

Data type: Vector of reals

Default value: <empty>

Syntax: Materials/*/sigma_s

Note: This parameter must be provided if $fromFile$ is false. The size of this parameter must be equal to $G^2 \times (L+1)$. Values are ordered as $\{\sigma_{g\rightarrow g}^{s \rightarrow s}, g = 1, \cdots, G; s = 1, \cdots, G; l = 0, \cdots, L\}$. In-group scatterings are ignored for diffusion schemes including CFEM-Diffusion and DFEM-Diffusion.

8.2.4.9  fissile

Description: To indicate if the material is fissile

Data type: Logical

Default value: False

Syntax: Materials/*/fissile

Note: This parameter is activated if $fromFile$ is false.

8.2.4.10  $nu_sigma_f$

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 if $fromFile$ is false and $fissile$ is true. The size of this parameter must be equal to $G$. 

240
8.2.4.11  \textit{chi}

Description: Fission spectrum
Data type: Vector of reals
Default value: $<$empty$>$
Syntax: Materials/*/\textit{chi}

Note: This parameter must be provided if $\textit{fromFile}$ is false and $\textit{fissile}$ 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 if $n_{\text{delay\_groups}}$ is non-zero. Otherwise, it contains the prompt spectrum. This convention is adapted by YAKXS.

8.2.4.12  \textit{neutron\_speed}

Description: Neutron speed
Data type: Vector of reals
Default value: $<$empty$>$
Syntax: Materials/*/\textit{neutron\_speed}

Note: This parameter must be provided if $\textit{fromFile}$ is false and $\textit{equation\_type}$ is transient. The size of this parameter must be equal to $G$. This parameter is ignored if $\textit{equation\_type}$ is not transient.

8.2.4.13  \textit{decay\_constant}

Description: Decay constants of delayed neutron precursors
Data type: Vector of reals
Default value: $<$empty$>$
Syntax: Materials/*/\textit{decay\_constant}

Note: This parameter must be provided if $\textit{fromFile}$ is false, $\textit{fissile}$ is true and $n_{\text{delay\_groups}}$ is non-zero. This parameter does not affect the calculation if $\textit{equation\_type}$ is not transient.

8.2.4.14  \textit{delay\_fraction}

Description: Delayed neutron fractions
Data type: Vector of reals
Default value: $<$empty$>$
Syntax: Materials/*/\textit{delay\_fraction}

Note: This parameter must be provided if $\textit{fromFile}$ is false, $\textit{fissile}$ is true and $n_{\text{delay\_groups}}$ is non-zero. The size of this parameter must be equal to $n_{\text{delay\_groups}}$. This parameter is used for generating weighted fission spectrum and affects the calculation in turn if $\textit{equation\_type}$ is not transient.
8.2.4.15  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 if 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 × G. Values are ordered as \( \chi_{d,\delta,ir \delta}, \delta = 1, \cdots, G; i = 1, \cdots, I \). This parameter is used for generating weighted fission spectrum and affects the calculation in turn if equation_type is not transient.

8.2.4.16  sigma_capture

Description: Constant Capture cross section
Data type: Vector of reals
Default value: <empty>
Syntax: Materials/*/sigma_capture

Note: This parameter is optional if fromFile is false. If it is provided, its size must be equal to G.

8.2.4.17  sigma_nalpha

Description: Constant Nalpha cross section
Data type: Vector of reals
Default value: <empty>
Syntax: Materials/*/sigma_nalpha

Note: This parameter is optional if fromFile is false. If it is provided, its size must be equal to G.

8.2.4.18  sigma_f

Description: Constant fission cross section
Data type: Vector of reals
Default value: <empty>
Syntax: Materials/*/sigma_f

Note: This parameter is optional if fromFile is false. If it is provided, its size must be equal to G.

8.2.4.19  kappa_sigma_f

Description: Constant kappa fission cross section
Data type: Vector of reals
Default value: <empty>
8.2.4.20  \textit{sigma_s_for_phase_function}

Description: Constant scattering cross section to be treated using a phase function
Data type: Vector of reals
Default value: <empty>
Syntax: Materials/*/sigma_s_for_phase_function

Note: This parameter must be provided if \textit{phase_function_names} is not empty. The size of this parameter must be equal to $G^2$. Values are ordered as $\{\sigma_{s,PF}^{g'\rightarrow g}, g' = 1, \cdots, G; g = 1, \cdots, G\}$. In-group scatterings are ignored for diffusion schemes including CFEM-Diffusion and DFEM-Diffusion.

8.2.4.21  \textit{keff_for_flux}

Description: For searching suitable flux spectrum by performing a buckling search to match this k-effective
Data type: Real
Default value: <empty>
Syntax: Materials/*/keff_for_flux

Note: This parameter is activated when material is fissile and flux spectrum is not provided. The value of this parameter must be smaller than the k-infinity of this material otherwise an error will be thrown. This parameter can be used to provide a default flux spectrum for on-the-fly group condensation.

8.2.5  \textbf{ConstantMatIDNeutronicsMaterial}

This material is the same as \textbf{ConstantNeutronicsMaterial} where the material IDs are read from the mesh, allowing in particular to have one material assign to multiple material IDs. It is noted that using this material requires the mesh to be either \texttt{GeneratedIDMesh} or \texttt{FileIDMesh}.

8.2.5.1  \textit{library_file}

Refer to \textit{library_file} in \textbf{ConstantNeutronicsMaterial}.

This parameter is required. File must be of xml type. In addition, this material requires the mesh to be either of type \texttt{FileIDMesh} or \texttt{GeneratedIDMesh}.

8.2.6  \textbf{FunctionNeutronicsMaterial}
This material uses *FunctionInputXS* in YAKXS for managing constant macroscopic cross sections. It differs from *ConstantNeutronicsMaterial* only as follows:

1. *is_meter* must be false in *FunctionNeutronicsMaterial*.
2. it does not have *sigma_capture* and *sigma_nalpha*;
3. it does not read cross sections from 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.

### 8.2.6.1 *material_id*

Refer to *material_id* in *ConstantNeutronicsMaterial*.

It is only used for dumping the material into a INSTANT XML file if *dbgmat* is true.

### 8.2.6.2 *sigma_t*

Refer to *sigma_t* in *ConstantNeutronicsMaterial*.

Every element can be either a constant value or a MOOSE function.

### 8.2.6.3 *diffusion_coef*

Refer to *diffusion_coef* in *ConstantNeutronicsMaterial*.

Every element can be either a constant value or a MOOSE function.

### 8.2.6.4 *sigma_r*

Refer to *sigma_r* in *ConstantNeutronicsMaterial*.

Every element can be either a constant value or a MOOSE function.

### 8.2.6.5 *L*

Refer to *L* in *ConstantNeutronicsMaterial*.

### 8.2.6.6 *sigma_s*

Refer to *sigma_s* in *ConstantNeutronicsMaterial*.

Every element can be either a constant value or a MOOSE function.

### 8.2.7 *fissile*

Refer to *fissile* in *ConstantNeutronicsMaterial*.
8.2.6.8 \textit{nu\_sigma\_f}

Refer to \textit{nu\_sigma\_f} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.

8.2.6.9 \textit{chi}

Refer to \textit{chi} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.

8.2.6.10 \textit{neutron\_speed}

Refer to \textit{neutron\_speed} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.

8.2.6.11 \textit{decay\_constant}

Refer to \textit{decay\_constant} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.

8.2.6.12 \textit{delay\_fraction}

Refer to \textit{delay\_fraction} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.

8.2.6.13 \textit{delay\_spectrum}

Refer to \textit{delay\_spectrum} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.

8.2.6.14 \textit{sigma\_f}

Refer to \textit{sigma\_f} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.

8.2.6.15 \textit{kappa\_sigma\_f}

Refer to \textit{kappa\_sigma\_f} in \texttt{ConstantNeutronicsMaterial}.
Every element can be either a constant value or a MOOSE function.
8.2.6.16  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. If neither is 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.

8.2.6.17  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. If neither is 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.

8.2.7  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. The user selects a single state point at which this material is evaluated. No feedback effects can be considered.

8.2.7.1  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 if dbgmat is true.

8.2.7.2  library_file

Description: File name containing the YAKXS libraries
Data type: String
Default value: <required>
Syntax: Materials/*/library_file

Note: The file must be in YAKXS XML format.
8.2.7.3  library_name

Description: Name of the multigroup library
Data type: string
Default value: <required>
Syntax: Materials/*/library_name
Note: Refer to [2] for details about the YAKXS format.

8.2.7.4  grid_names

Description: Names for all library parameters (used for tabulating the XS)
Data type: Vector of strings
Default value: <required>
Syntax: Materials/*/grid_names
Note: The reference grid value will be used for the missing grids.

8.2.7.5  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.

8.2.7.6  isotopes

Description: Name of isotopes
Data type: Vector of strings
Default value: <required>
Syntax: Materials/*/isotopes
Note: This parameter must not be empty.

8.2.7.7  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.
8.2.8 **MixedMatIDNeutronicsMaterial**

This material is similar to MixedNeutronicsMaterial except that the material IDs are read from the mesh. As such, the mesh must be either GeneratedIDMesh or FileIDMesh.

8.2.8.1 **library_file**

Refer to *library_file* in MixedNeutronicsMaterial.

8.2.8.2 **library_name**

Refer to *library_name* in MixedNeutronicsMaterial.

8.2.8.3 **grid_names**

Refer to *grid_names* in MixedNeutronicsMaterial.

8.2.8.4 **grid**

Refer to *grid* in MixedNeutronicsMaterial.

8.2.8.5 **isotopes**

Refer to *isotopes* in MixedNeutronicsMaterial.

8.2.8.6 **densities**

Refer to *densities* in MixedNeutronicsMaterial.

8.2.8.7 **replace_transport_with_derived**

Description: To replace provided transport cross sections with the one derived from total and scattering cross sections

Data type: Logical

Default value: False

Syntax: Materials/*/replace_transport_with_derived
8.2.8 starting_groups_for_precondensation

Description: The starting groups for the condensation of the libraries
Data type: Vector of integers
Default value: <empty>
Syntax: Materials/*/starting_groups_for_precondensation

Note: If this parameter is provided and valid, multigroup library will be condensed immediately after loading for further calculations. Otherwise, no condensation will be performed. If this parameter is provided, Flux reaction is required in the multigroup library as the spectrum for condensation. The size of this parameter will be the number of coarse groups, which must be smaller than or equal to the number of groups in the library. The elements in this parameter must be ordered strictly ascendingly. For example, if we have 10 groups in the original library, ‘1 3 4’ of this parameter will collapse 1-2 into the first coarse group, 3 into the second coarse group and 4-10 into the third coarse group.

8.2.9 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.

8.2.9.1 library_file

Refer to library_file in MixedNeutronicsMaterial.

8.2.9.2 library_name

Refer to library_name in MixedNeutronicsMaterial.

8.2.9.3 material_id

Refer to material_id in ConstantNeutronicsMaterial.

8.2.9.4 replace_transport_with_derived

Refer to replace_transport_with_derived in MixedMatIDNeutronicsMaterial.

8.2.9.5 starting_groups_for_precondensation

Refer to starting_groups_for_precondensation in MixedMatIDNeutronicsMaterial.
8.2.9.6  *stateful_properties*

Description: Determines which quantities are stateful
Data type: Enumeration (/none/mixing_table/all/)
Default value: none
Syntax: Materials/*/stateful_properties

Note: mixing_table will cause mixing tables constructed on quadrature points to be stored to avoid repeatedly constructing them on-the-fly when variable values are not changed. all will cause both mixing tables and mixtures on quadrature points to be stored. It is noted that storing mixing tables and mixtures increases memory usage.

8.2.9.7  *grid_names*

Refer to grid_names in MixedNeutronicsMaterial.

8.2.9.8  *grid_variables*

Description: Interpolation variables for all library parameters
Data type: Vector of MOOSE variable names
Default value: <empty>
Syntax: Materials/*/grid_variables

Note: The size of this parameter must be equal to the size of grid_names. grid_names are the names of the tabulation parameters in the YAKXS file, while grid_names is the name of MOOSE variables evaluated in the problem used to interpolate cross sections.

8.2.9.9  *isotopes*

Refer to isotopes in MixedNeutronicsMaterial.

8.2.9.10  *densities*

Refer to densities in MixedNeutronicsMaterial.

8.2.10  *CoupledFeedbackMatIDNeutronicsMaterial*

This material is similar to CoupledFeedbackNeutronicsMaterial except that the material IDs are read from the mesh. As such, the mesh must be either GeneratedIDMesh or FileIDMesh.

8.2.10.1  *library_file*

Refer to library_name in CoupledFeedbackNeutronicsMaterial.
8.2.10.2  *library_file*

Refer to *library_file* in *CoupledFeedbackNeutronicsMaterial*.

8.2.10.3  *replace_transport_with_derived*

Refer to *replace_transport_with_derived* in *MixedMatIDNeutronicsMaterial*.

8.2.10.4  *starting_groups_for_precondensation*

Refer to *starting_groups_for_precondensation* in *MixedMatIDNeutronicsMaterial*.

8.2.10.5  *stateful_properties*

Refer to *stateful_properties* in *CoupledFeedbackNeutronicsMaterial*.

8.2.10.6  *grid_names*

Refer to *grid_names* in *MixedNeutronicsMaterial*.

8.2.10.7  *grid_variables*

Refer to *grid_variables* in *CoupledFeedbackNeutronicsMaterial*.

8.2.10.8  *isotopes*

Refer to *isotopes* in *MixedNeutronicsMaterial*.

8.2.10.9  *densities*

Refer to *densities* in *MixedNeutronicsMaterial*.

8.2.11  *CRoddedNeutronicsMaterial*

This material is used to model the control-rod movement within a block. A control rod is illustrated in Fig. 28 that depicts a typical control rod. It illustrates the case with 3 regions typically labeled unrodded, rodded, and driver. *CRoddedNeutronicsMaterial* supports more general control rods with more regions. In the withdrawn direction, a control rod is divided into at least three parts: one region below the front, and then \( n + 1 \) positions above the front with \( n \) being the number of entries in the *rod_segment_length* parameter. In case of Fig. 28 \( n = 1 \). Control rod lengths are specified starting from the front moving up assuming that the first and last regions extend to infinity. Control rod front refers to the interface between rodded material and unrodded material. It is noted that the in case that diffusion coefficients are not available and *diffusion_coefficient_scheme* is *user-supplied*, rodded neutronics material will ask for the derived diffusion coefficient to do the weighting. User can set *diffusion_coefficient_scheme* to *local* to let rodded neutronics material do the weighting for total and scattering cross sections first and then derive the diffusion coefficients from them.
8.2.11.1 \textit{front\_position\_function}

Description: The front position of the control rod as a function of time
Data type: string
Default value: \textlt{required}\textgt
Syntax: Materials/*/front\_position\_function

8.2.11.2 \textit{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

8.2.11.3 \textit{rod\_segment\_length}

Description: The length of each rod segments.
Data type: Vector of reals
Default value: 10000
Syntax: Materials/*/rod\_segment\_length

Note: Default is a single entry of 10000 (assumed to be large compared with reactor). This means only three materials are present: driver, control material and un-rodded material. The length of each rod segments except the first and last ones (always -infinity/infinity). Order by starting at control rod front and follow in direction of rod withdrawal.

8.2.11.4 \textit{library\_file}

Refer to \textit{library\_file} in ConstantNeutronicsMaterial.
This parameter is required. File must be in xml type.
8.2.11.5  *segment_material_ids*

Description: The material ids for each segment
Data type: Vector of integers
Default value: <required>
Syntax: Materials/*/material_ids

8.2.11.6  *projection_order*

Description: The projection order used for decusping
Data type: unsigned integer
Default value: 0
Syntax: Materials/*/projection_order

Note: The projection order $p$ controls the expansion order of the Legendre polynomial series used for decusping. For everything but the diffusion coefficient in the withdrawal direction the order is $2p$ ($2p + 1$ terms in expansion), while for the diffusion coefficient in the withdrawal direction it is $2p - 2$ ($2p - 1$ terms in expansion). $p$ should be chosen equal to the order of the scalar flux expansion.

8.2.12  *MixedRoddedNeutronicsMaterial*

This is the *MixedNeutronicsMaterial* equivalent for rodded regions. The segment cross sections are retrieved from YAKXS cross sections files in exactly the same way that *MixedNeutronicsMaterial* obtains cross sections. The comments for *CRoddedNeutronicsMaterial* apply.

8.2.12.1  *front_position_function*

Refer to *front_position_function* in *CRoddedNeutronicsMaterial*.

8.2.12.2  *rod_withdrawn_direction*

Refer to *rod_withdrawn_direction* in *CRoddedNeutronicsMaterial*.

8.2.12.3  *rod_segment_length*

Refer to *rod_segment_length* in *CRoddedNeutronicsMaterial*.

8.2.12.4  *segment_material_ids*

Refer to *segment_material_ids* in *CRoddedNeutronicsMaterial*.
8.2.12.5  projection_order

Refer to projection_order in CRoddedNeutronicsMaterial.

8.2.12.6  library_file

Refer to library_file in MixedNeutronicsMaterial.

8.2.12.7  library_name

Refer to library_name in MixedNeutronicsMaterial.

8.2.12.8  grid_names

Description: Names for all library parameters
Data type: Vector of vector of strings.
Default value: <required>
Syntax: Materials/*/grid_names
Note: Leading index is the segment index and size must match segment_material_ids.

8.2.12.9  grid

Description: Integer index of the grid point cross sections associated with the grid name in the grid_names parameter.
Data type: Vector of vectors of integers
Default value: <required>
Syntax: Materials/*/grid
Note: Leading index is the segment index and size must match segment_material_ids. The size of the vectors at each segment id must be equal to the corresponding size of grid_names.

8.2.12.10  isotopes

Description: Name of isotopes
Data type: Vector of vectors of strings
Default value: <required>
Syntax: Materials/*/isotopes
Note: Leading index is the segment index and size must match segment_material_ids. This parameter must not be empty.
8.2.12.11 densities

Description: Atomic densities of isotopes
Data type: Vector of vector reals
Default value: <required>
Syntax: Materials/*/densities

Note: Leading index is the segment index and size must match segment_material_ids.
Note: The parameters grid_names, grid, isotopes, and densities are double vectors allowing to enter separate values for each segment. The size of the leading dimension of these vectors must be equal to the size of the parameter segment_material_ids. Each subvector corresponds to the entry in MixedNeutronicsMaterial of the same name.

8.3 Thermal radiation materials

Thermal radiation materials are special because they are interacting with TransportSystems in the following ways:

1. In multischeme settings thermal radiation materials cannot be defined across two discretization schemes even these are identical.
2. Thermal radiation materials cannot be used without defining a transport system.

8.3.1 Material properties declared by thermal radiation materials

All TR materials declare a set of material properties, which are listed in Table 30. 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.

8.3.2 Common parameters for all thermal radiation materials

The following parameters are shared by all thermal radiation materials.

8.3.2.1 block

Refer to block in Sec. 8.2.3.

8.3.2.2 group_bounds

Refer to group_bounds in Sec. 8.2.3. The default value is set by transport system to be consistent with frequency_bounds in Thermal.
<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Type</th>
<th>Declared if</th>
</tr>
</thead>
<tbody>
<tr>
<td>light_speed_g&lt;g&gt;</td>
<td>(c_g)</td>
<td>Real</td>
<td>Always</td>
</tr>
<tr>
<td>light_speed</td>
<td>(c_g, g = 1, \ldots, G)</td>
<td>std::vector&lt;Real&gt;</td>
<td>Always</td>
</tr>
<tr>
<td>diffusion_coefficient_g&lt;g&gt;</td>
<td>(D_g)</td>
<td>Real</td>
<td>Diffusion</td>
</tr>
<tr>
<td>diffusion_coefficient</td>
<td>(D_g)</td>
<td>RealVector</td>
<td>Diffusion</td>
</tr>
<tr>
<td>vector_diffusion_coefficient_g&lt;g&gt;</td>
<td>(D_g)</td>
<td>RealVector</td>
<td>Diffusion</td>
</tr>
<tr>
<td>tensor_diffusion_coefficient_g&lt;g&gt;</td>
<td>(D_g)</td>
<td>RealTensor</td>
<td>Diffusion</td>
</tr>
<tr>
<td>sigma_removal_g&lt;g&gt;</td>
<td>(σ_{r,g})</td>
<td>Real</td>
<td>Diffusion</td>
</tr>
<tr>
<td>opacity_g&lt;g&gt;</td>
<td>(σ_{i,g})</td>
<td>Real</td>
<td>always</td>
</tr>
<tr>
<td>scattering_g&lt;p&gt;_g&lt;g&gt;</td>
<td>(σ_{p}→g, l = 0, \ldots, L)</td>
<td>std::vector&lt;Real&gt;</td>
<td>ScatteringPattern</td>
</tr>
<tr>
<td>scattering</td>
<td>(p_{p→g}(\mu))</td>
<td>PhaseFunction pointer</td>
<td>NonzeroScattering</td>
</tr>
<tr>
<td>phase_function_g&lt;p&gt;_g&lt;g&gt;</td>
<td>(σ_{p}→g)</td>
<td>Real</td>
<td>UsingPhaseFunction</td>
</tr>
<tr>
<td>phase_func_g&lt;p&gt;_g&lt;g&gt;</td>
<td>(s_{p,phase})</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>grad_light_speed_g&lt;g&gt;</td>
<td>(\frac{\nabla c_g}{c_g})</td>
<td>RealVector</td>
<td>VaryingLightSpeed</td>
</tr>
<tr>
<td>grad_light_speed</td>
<td>(\frac{\nabla c_g}{c_g}, g = 1, \ldots, G)</td>
<td>std::vector&lt;RealVector&gt;</td>
<td>VaryingLightSpeed</td>
</tr>
<tr>
<td>refractive_index_g&lt;g&gt;</td>
<td>(c_g)</td>
<td>Real</td>
<td>Always</td>
</tr>
<tr>
<td>emissivity_g&lt;g&gt;</td>
<td>(ε_g, g = 1, \ldots, G)</td>
<td>std::vector&lt;Real&gt;</td>
<td>Always</td>
</tr>
<tr>
<td>mean_emissivity</td>
<td>(\bar{ε})</td>
<td>Real</td>
<td>Always</td>
</tr>
<tr>
<td>rosseland_mean_emissivity</td>
<td>(\bar{ε})</td>
<td>Real</td>
<td>Always</td>
</tr>
<tr>
<td>temperature_matprop</td>
<td>(T)</td>
<td>Real</td>
<td>Always</td>
</tr>
<tr>
<td>absorptivity_g&lt;g&gt;</td>
<td>(α_g)</td>
<td>Real</td>
<td>Always</td>
</tr>
</tbody>
</table>

\(^{a}\) Transient means that equation\_type is equal to transient;

\(^{b}\) Diffusion means that scheme is equal to CFEM-Diffusion or DFEM-Diffusion;

\(^{c}\) NotDiffusion means that scheme is equal to any other than CFEM-Diffusion and DFEM-Diffusion;

\(^{d}\) ScatteringPattern means that the property will be declared for the non-zero scattering entries;

\(^{e}\) NonzeroScattering means that there is scattering for this material. For diffusion calculations, this means that there is cross-group scatterings because in-group scatterings are merged into total as removal cross sections and thus never used;

\(^{f}\) If phase\_function\_names is not empty and phase\_function\_treatment is SN;

\(^{g}\) If the material type is FunctionTRMaterial and the light speed is not a ConstantFunction.
8.3.2.3  scalar_fluxes
Refer to scalar_fluxes in Sec. 8.2.3.

8.3.2.4  volume_adjuster
Refer to volume_adjuster in Sec. 8.2.3.

8.3.2.5  is_meter
Refer to is_meter in Sec. 8.2.3.

8.3.2.6  dump_prop_at_time
Refer to dump_prop_at_time in Sec. 8.2.3.

8.3.2.7  dump_prop_at_residual
Refer to dump_prop_at_residual in Sec. 8.2.3.

8.3.2.8  dump_prop_on_elements
Refer to dump_prop_on_elements in Sec. 8.2.3.

8.3.2.9  dump_prop_on_point
Refer to dump_prop_on_point in Sec. 8.2.3.

8.3.2.10  output
Refer to output in Sec. 8.2.3.

8.3.2.11  dbgmat
Refer to dbgmat in Sec. 8.2.3.

8.3.2.12  diffusion_limiter
Refer to diffusion_limiter in Sec. 8.2.3. It is noted that diffusion coefficients are always locally derived.

8.3.2.13  maximum_diffusion_coefficient
Refer to maximum_diffusion_coefficient in Sec. 8.2.3.
8.3.2.14  *slope_factor*

Refer to *slope_factor* in Sec. 8.2.3.

8.3.2.15  *larsen_limit_factor*

Refer to *larsen_limit_factor* in Sec. 8.2.3.

8.3.2.16  *phase_function_names*

Refer to *phase_function_names* in Sec. 8.2.3.

8.3.2.17  *phase_function_treatment*

Refer to *phase_function_treatment* in Sec. 8.2.3.

8.3.2.18  *phase_function_departing_groups*

Refer to *phase_function_departing_groups* in Sec. 8.2.3.

8.3.2.19  *phase_function_arriving_groups*

Refer to *phase_function_arriving_groups* in Sec. 8.2.3.

8.3.2.20  *coarse_group_index*

Refer to *coarse_group_index* in Sec. 8.2.3.

8.3.2.21  *flux_corrected_condensation*

Refer to *flux_corrected_condensation* in Sec. 8.2.3.

8.3.2.22  *spectrum_prolongation_option*

Refer to *spectrum_prolongation_option* in Sec. 8.2.3.

8.3.3  ConstantTRMaterial

This material provides the basic capability of doing multigroup thermal-radiation transport calculations.
8.3.3.1 **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.

8.3.3.2 **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).

8.3.3.3 **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.

8.3.3.4 **L**

Description: Order of scattering anisotropy
Data type: Integer
Default value: 0
Syntax: Materials/*/L

Note: If this parameter is larger than NA provided in the discretization scheme (NA default is zero), expansion terms beyond L are ignored. Higher order scattering (beyond order 0) will always be ignored for diffusion schemes, CFEM-Diffusion and DFEM-Diffusion.

8.3.3.5 **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 \( G^2 \times (L+1) \).

8.3.3.6  \textit{material\_id}

Description: ID of the material
Data type: integer
Default value: 1
Syntax: Materials/*/material\_id

Note: It is used for dumping the material into a INSTANT XML file when \textit{dbgmat} is true.

8.3.4  Function\textit{TRMaterial}

8.3.4.1  \textit{absorptivity}

Refer to \textit{absorptivity} in Constant\textit{TRMaterial}. Every entry can be a MOOSE function.

8.3.4.2  \textit{emissivity}

Refer to \textit{emissivity} in Constant\textit{TRMaterial}. Every entry can be a MOOSE function.

8.3.4.3  \textit{light\_speed}

Refer to \textit{light\_speed} in Constant\textit{TRMaterial}. Every entry can be a MOOSE function.

8.3.4.4  \textit{L}

Refer to \textit{L} in Constant\textit{TRMaterial}.

8.3.4.5  \textit{scattering}

Refer to \textit{scattering} in Constant\textit{TRMaterial}. Every entry can be a MOOSE function.

8.3.4.6  \textit{material\_id}

Refer to \textit{material\_id} in Constant\textit{TRMaterial}.
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 enhance users' understanding of the relevant control parameters. Most of parameters have default values and do not need users' input for standard calculations.

9.1 Steady

It is a general solver provided by MOOSE to solve the discrete steady-state nonlinear or linear problem:

$$\mathbf{F}(\mathbf{x}) = 0.$$  \hspace{1cm} (34)

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

$$\mathbf{J}(\mathbf{x}^{(n-1)}) \delta \mathbf{x}^{(n)} = \mathbf{F}(\mathbf{x}^{(n-1)}),$$  \hspace{1cm} (35)

where

$$\mathbf{J}(\mathbf{x}^{(n-1)}) \equiv \mathbf{F}_x|_{x=x^{(n-1)}}$$  \hspace{1cm} (36)

is the Jacobian matrix evaluated at $x^{(n-1)}$. Jacobian matrix depends on $x^{(n-1)}$ for general nonlinear problems while it is constant for linear problems. The right hand side is also typically referred to as the residual at $x^{(n-1)}$ of Eq. (34). The Krylov methods are employed for solving the above linear equation, which requires only the evaluation of the matrix-vector product $\mathbf{J}(x^{(n-1)})y$. Within the MOOSE framework, the Jacobian-Free Newton Krylov method is used that approximates matrix vector products by the Finite-Difference like approximation:

$$\mathbf{J}(x^{(n-1)})y \approx \frac{\mathbf{F}(x^{(n-1)}+\epsilon y) - \mathbf{F}(x^{(n-1)})}{\epsilon},$$  \hspace{1cm} (37)

where the scalar value $\epsilon$ is chosen by PETSc automatically to approximate $\mathbf{J}(x^{(n-1)})y$ accurately for the linear solve. It is noted that for a linear problem of which $\mathbf{F}(x)$ can be expressed as $\mathbf{A}x - \mathbf{b}$, where matrix $\mathbf{A}$ is the Jacobian independent on $\mathbf{x}$ and $\mathbf{b}$ is the right-hand-side vector, the right hand side of Eq. (37) is independent on $\epsilon$. Section 5.5 of PETSc user’s manual on matrix-free methods details the algorithm for choosing the value of $\epsilon$. It is actually the PETSc option ‘-mat_mffd_err’ controls the $\epsilon$ but not ‘-snes_mf_err’ unless we set ‘-snes_mf_version’ to 2 other than the default 1. This could be changed in future PETSc updates.

The Krylov methods typically also require an approximation of the actual Jacobian $\mathbf{J}(x^{(n-1)}) \approx \mathbf{J}(x^{(n-1)})$ for preconditioning the Krylov solution at each linear iteration. Note, the preconditioning matrix is seldom the exact Jacobian $\mathbf{J}$ because it would require too much computational time and memory to compute, and in some cases is simply impossible to compute. By default the type of Krylov method in use is GMRES because it does not have assumptions on the underlying Jacobian. The initial guess for each linear solve is always set to zero, which implies 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 the Hessenberg matrix if GMRES method is used. At the conclusion of the nonlinear iteration, the solution is updated as follows

\[ x^n = x^{n-1} + \alpha \delta x^{(n)} \]  

(38)

where \( \alpha \) is determined by the line-search algorithm. We can see that at each nonlinear or Newton iteration, we will need to update the preconditioning matrix and evaluate the residual 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. It is noted that the default preconditioner type\(^2\) depends on the number of processors and also depends on the assembled preconditioning matrix \( P \). 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. Note, there are two approximations in play here: (1) the Jacobian \( J \) is approximated by a matrix \( P \) that is easier to compute, and (2) the matrix \( P \) is inverted approximately.

The preconditioning matrix \( P \) can be viewed with the PETSc option ‘-ksp_view_pmat’.

### 9.1.1 \textit{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. (35). It is 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 \textit{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. (35). It is 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

\[ \| F(x^{(n-1)}) - J(x^{(n-1)})\delta x^{(n)} \|_2 < \max(rtol \| F(x^{(n-1)}) \|_2 , atol), \]  

(39)

where \( rtol \) is this parameter.

### 9.1.3 \textit{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

---

\(^{2}\)In PETSc the preconditioner type refers to the method to obtain an approximation of the inverse of \( P \) and not a means to compute the elements of \( P \).
Note: This parameter is used to control the solving of Eq. (35). It is the absolute convergence tolerance on the change of residual norm of linear iterations with respect to the previous one, Default value −1 means this parameter will not take 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 \( \alpha \) in Eq. (38). It is used to set the default value of a PETSc control parameter -snes_linesearch_type. The default value of default means the default PETSc line search algorithm. Users are allowed to directly use the PETSc control parameter -snes_linesearch_type to overwrite this parameter. Users can play with this parameter, for example setting it to none, when slow convergence of Newton (or nonlinear) iteration is observed.

9.1.5 solve_type

Description: Controls the PETSc solve type
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 \( P \). By default, the preconditioning matrix is block-diagonal with each block corresponding to a single MOOSE variable without custom preconditioning, refer to Preconditioner. Off-diagonal Jacobian terms are ignored. 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 \( P(x^{n-1}) \) 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 petc_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. (38) 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 setting 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. ‘-snes_test_err’ and ‘-mat_fd_type’ are the two PETSc options affecting how the Jacobian is evaluated with finite differences. Preconditioning input block will be ignored.
9.1.6  \textit{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  \textit{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 is used to set the default value of a PETSc control parameter \textit{-snes\_atol}. Users are allowed to directly use the PETSc control parameter \textit{-snes\_atol} to overwrite this parameter.

9.1.8  \textit{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 is used to set the default value of a PETSc control parameter \textit{-snes\_max\_funcs}. Users are allowed to directly use the PETSc control parameter \textit{-snes\_max\_funcs} to overwrite this parameter.

9.1.9  \textit{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 is used to set the default value of a PETSc control parameter \textit{-snes\_max\_it}. Users are allowed to directly use the PETSc control parameter \textit{-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 is 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 is 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  *picard_max_its*

Description: Specifies the maximum number of Picard iterations
Data type: Integer
Default value: 1
Syntax: Executioner/picard_max_its

Note: The default value of one turns off the Picard iterations. This parameter is mainly used when wanting to do Picard iterations with MultiApps that are set to execute_on timestep_end or timestep_begin.

9.1.13  *accept_on_max_picard_iteration*

Description: True to treat reaching the maximum number of Picard iterations as converged
Data type: Logical
Default value: False
Syntax: Executioner/accept_on_max_picardIteration

Note: This parameter is active only when *picard_max_its* is greater than 1.
9.1.14  picard_rel_tol

Description: The relative nonlinear residual drop required for convergence 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 norm, or the Picard residual norm. Before entering the Picard iteration, a Picard residual norm evaluation is performed and the result is set as the initial Picard residual norm. This parameter determines the convergence based on the relative residual drop with respect to the initial Picard residual norm. The Picard residual norm is re-evaluated at the end of each Picard iteration when all variable transfers on timestep_end have been accomplished. These transferred variables participate in the residual evaluation making the residual norm different from the nonlinear residual norm right after the master Newton solve. The Picard residual norm is also re-evaluated after all variable transfers on timestep_begin are accomplished within a Picard iteration. Both the Picard residual norms at the timestep_begin and the timestep_end are considered for convergence check. This parameter is active only when picard_max_its is greater than 1.

9.1.15  picard_abs_tol

Description: The absolute nonlinear residual required for convergence during Picard iterations
Data type: Real
Default value: \(10^{-50}\)

Syntax: Executioner/picard_abs_tol

Note: This check is performed based on the Picard residual norm (refer to picard_rel_tol for more details). By default, this tolerance is too small to take effect (meaning that the convergence is controlled by other criteria, such as relative tolerance etc.). This parameter is active only when picard_max_its is greater than 1.

9.1.16  picard_force_norms

Description: Force the evaluation of both the TIMESTEP_BEGIN and TIMESTEP_END norms
Data type: Logical
Default value: False

Syntax: Executioner/picard_force_norms

Note: When this parameter is false and there are not active MultiApps on TIMESTEP_BEGIN or TIMESTEP_END, Rattlesnake will not perform the Picard residual norm evaluation accordingly. However, there could be cases, for example IQS, the Picard residual norm evaluation is desired even without MultiApps. In these cases, this parameter needs to be set to true to get the proper convergence control. This parameter is active only when picard_max_its is greater than 1.

9.1.17  disable_picard_residual_norm_check

Description: Disable the Picard residual norm evaluation
Data type: Logical
Default value: False

Syntax: Executioner/disable_picard_residual_norm_check
Note: Setting this parameter to true disables the Picard residual norm evaluation, thus making \textit{picard\_rel\_tol}, \textit{picard\_abs\_tol} and \textit{picard\_force\_norms} inactive. Used in conjunction with \textit{accept\_on\_max\_picard\_iteration} equal to true, users can perform a fixed number of Picard iterations. This parameter is active only when \textit{picard\_max\_its} is greater than 1.

9.1.18 \textit{relaxation\_factor}

Description: Fraction of newly computed value to keep during Picard iteration

Data type: Real

Default value: 1.0

Syntax: Executioner/relaxation\_factor

Note: This parameter must be greater than 0 and less than 2. It provides the weighting (\textit{relaxation\_factor}) of the current solution of a Picard iteration and the weighting (1-\textit{relaxation\_factor}) of the solution at the previous Picard iteration to update the solution at the end of the Picard iteration. Default value 1 means no relaxation. This parameter is active only when \textit{picard\_max\_its} is greater than 1.

9.1.19 \textit{relaxed\_variables}

Description: List of variables to relax during Picard Iteration

Data type: Vector of strings

Default value: <empty>

Syntax: Executioner/relaxed\_variables

Note: This parameter must be provided when \textit{relaxation\_factor} is set to a value different from 1, otherwise no variables are to be relaxed. This parameter is active only when \textit{picard\_max\_its} is greater than 1.

9.1.20 \textit{petsc\_options}

Description: Used to set singleton PETSc options

Data type: Vector of strings

Default value: <empty>

Syntax: Executioner/petsc\_options

9.1.21 \textit{petsc\_options\_iname}

Description: Names of PETSc name/value pairs

Data type: Vector of strings

Default value: <empty>

Syntax: Executioner/petsc\_options\_iname
9.1.22  *petsc_options_value*

Description: Values of PETSc name/value pairs  
Data type: Vector of strings  
Default value: `<empty>`  
Syntax: Executioner/petsc_options_value  

Note: Each parameter must be matched with an entry in *petsc_options_iname, 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 31.

<table>
<thead>
<tr>
<th>PETSc parameter name</th>
<th>PETSc parameter value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pc_type</td>
<td>hypre</td>
<td>To set the preconditioner type to HYPRE</td>
</tr>
<tr>
<td>-pc_hypre_type</td>
<td>boomeramg</td>
<td>To use HYPRE BoomerAMG for preconditioning</td>
</tr>
<tr>
<td>-pc_hypre_boomeramg_tol</td>
<td>1e-4</td>
<td>To set the tolerance of multiple cycles with BoomerAMG</td>
</tr>
<tr>
<td>-pc_hypre_boomeramg_max_iter</td>
<td>100</td>
<td>To set the maximum number of cycles with BoomerAMG</td>
</tr>
<tr>
<td>-ksp_gmres_restart</td>
<td>100</td>
<td>To set the number of GMRes iterations for restart</td>
</tr>
</tbody>
</table>

More to be added when necessary...

9.2  *Preconditioning*

Preconditioner is closely related to the executioner, and therefore we organize it as a subsection of *Executioner*. *Preconditioning* block controls how the preconditioning matrix is built, i.e. which off-diagonal terms blocks are included in the preconditioner matrix, 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: Preconditioning/*/type  

Note: Currently available preconditioners are listed in Table 32:

<table>
<thead>
<tr>
<th>Type</th>
<th>Full name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP</td>
<td>Single matrix preconditioner</td>
</tr>
<tr>
<td>FDP</td>
<td>Finite difference preconditioner</td>
</tr>
<tr>
<td>PBP</td>
<td>Physics-based preconditioner</td>
</tr>
<tr>
<td>BDP</td>
<td>Block-diagonal preconditioner</td>
</tr>
</tbody>
</table>

268
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. Rattlesnake is responsible for evaluating this preconditioning matrix.

9.2.2.1 **pc_side**

Description: Preconditioning side  
Data type: Enumeration (/left/right/symmetric/)  
Default value: right  
Syntax: Preconditioning/*/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 to construct all off-diagonal blocks of the Jacobian  
Data type: Logical  
Default value: false  
Syntax: Preconditioning/*/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: Preconditioning/*/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: Preconditioning/*/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: Preconditioning/*/coupled_groups

Note: Off-diagonal jacobiands 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:

\[
\begin{bmatrix}
    u1 & u2 & u3 & u4 & u5 \\
    u2 & \times & \times & \times & \\
    u3 & \times & \times & \times & \\
    u4 & \times & \times & \times & \\
    u5 & \times & \times & \times & \\
\end{bmatrix}
\]

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 always be 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 on a single processor, i.e. in serial execution. It is costly and should be used only for testing purpose. One difference of this preconditioner with respect to FD solve type in `solve_type` is that it provides more control on how the preconditioning matrix is structured and it can work with PJFNK `solve_type`. Because MOOSE uses coloring schemes for evaluating the Jacobian, which is different from the default scheme used by FD solve type, different PETSc options are controlling the behavior of the evaluation. Those options, such as `-mat_fd_coloring_err`, can be found in Section 5.6 of PETSc user’s manual.

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:

\[ b_i = x_i - \sum_{j=1}^{N} c_{i,j} P_{i,j} y_j, \]  

(40)

where \( x \) is the global vector passed into the preconditioner and \( y \) is the global vector outputted by the preconditioner. \( P \) 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: Preconditioning/* 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: Preconditioning/*/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 cycles if you like).

9.2.5  **BDP (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 formed 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 loops through all variables once and apply the local preconditioners during preconditioning. The preconditioning type is hardcoded as HYPRE BoomerAMG. In this regard, it is a special case of PBP. However, it provides some convenient parameters. If this preconditioner is used solve_type is no longer active ands 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: Preconditioning/*/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: Preconditioning/*/rel_tol

9.2.5.4  **verbose**

Description: True to output progress on screen
Data type: Integer
Default value: 1
Syntax: Preconditioning/*/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: Preconditioning/*\-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: Preconditioning/*\-fix_jacobian  

Note: This parameter will be ignored if *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.

9.2.6.2  *new*

Description: True to use the new implementation of sweeper  
Data type: Logical  
Default value: False  
Syntax: Preconditioning/*\-new  

Note: We re-implemented the sweeper and decided to let the old and new implementations co-exist temporarily. This parameter can be used to switch between the old and the new implementation. The new implementation pre-assembles the elemental matrices and is significantly faster. Thus it is encouraged to set this parameter to true.
9.2.6.3  show_cpu_time

Description: True to show CPU time in sweeping
Data type: Logical
Default value: False
Syntax: Preconditioning/*show_cpu_time

9.3  Transient

It is a general solver provided by MOOSE to solve the finite-dimensional transient nonlinear or linear problem:

\[
\frac{dT(x(t), t)}{dt} = -R(x(t), t),
\]

with the initial condition

\[x(t_0) = x_0,\]

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

\[
\frac{T(x, t) - T(x^{old}, t^{old})}{\Delta t} = -R(x, t),
\]

where \(\Delta t = t - t^{old}\). If we define \(F(x)\) as

\[F(x) \equiv R(x, t) + \frac{1}{\Delta t} T(x, t) - \frac{T(x^{old}, t^{old})}{\Delta t},\]

where \(t, t^{old}\), and \(x^{old}\) are given, at every time step we are solving a finite-dimensional steady-state nonlinear or linear problem as given in Eq. (34). We can use our machinery implemented in the Steady executioner to solve it at every time step and then march 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 steppers for adjusting 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  \textit{start\_time}

Description: The start time of the simulation
Data type: Real
Default value: 0
Syntax: Executioner/start\_time

9.3.4  \textit{end\_time}

Description: The end time of the simulation
Data type: Real
Default value: $10^{30}$
Syntax: Executioner/end\_time

9.3.5  \textit{dt}

Description: The timestep size between solves
Data type: Real
Default value: 1
Syntax: Executioner/dt

9.3.6  \textit{dt\_min}

Description: The minimum timestep size in an adaptive run
Data type: Real
Default value: $2 \times 10^{-14}$
Syntax: Executioner/dt\_min

9.3.7  \textit{dt\_max}

Description: The maximum timestep size in an adaptive run
Data type: Real
Default value: $10^{30}$
Syntax: Executioner/dt\_max

9.3.8  \textit{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: $2 \times 10^{-14}$
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  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.

9.3.16  verbose

Description: Print detailed diagnostics on timestep calculation
Data type: Logical
Default value: False
Syntax: Executioner/verbose

9.3.17  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.
9.3.18 **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 $\Delta t$ to get $\hat{u}_{n+1}$ from $u_n$;
  2. Take two time steps of size $\Delta t^2$ 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 $\Delta t$;
  6. Parameters $e_{TOL}$ and $e_{MAX}$ can be specified in the input file as e_tol and e_max (in the Executioner block).

9.3.19 **Quadrature**

(advanced topic, to be added.)

9.4 **InversePowerMethod**

The operator form of the steady-state eigenvalue problem of the neutron transport equation is given by

$$A(x) = \frac{1}{k} B(x), \quad (45)$$

where $A$ represents all the events neutrons undergo in a background media, including collision, streaming, scattering, absorption, etc. except fission. $B$ represents the fission events. $A$, $B$ can be nonlinear but for neutron transport problems can be linear. $x$ is the solution that has the following properties: (1) $x = 0$ is a solution but is usually discarded, and (2) if $x$ is a solution you can multiply it with an arbitrary real number to obtain another solution. It could be the angular fluxes of some selected directions in 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 in reality self-sustained neutron populations are possible because the averaged neutrons emitted from fission after forming a compound nucleus is larger than one, actually about 2.5 for Uranium and
larger for Plutonium. Neutronics usually cares only about the smallest adjustment of the system or the maximum $k$ to achieve criticality, which has the physical meaning of the multiplication factor. If $k > 1$, the system is super-critical, it $k < 1$ the system is sub-critical and if $k = 1$, 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 in the remainder of this document.

The algorithm for doing inverse power iterations is given in Algorithm 1. We notice immediately that $|\frac{B(\mathbf{x})}{k}|$ remains constant during the iteration, so if we make $|\frac{B(\mathbf{x}^{(0)})}{k(0)}|$ equal to one, the algorithm can be simplified significantly into Algorithm 2.

### 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 $\dagger$ and update $k$:
   \[
   \mathbf{A}(\mathbf{x}^{(n)}) = \frac{1}{k^{(n-1)}} \mathbf{B}(\mathbf{x}^{(n-1)})
   \]
   \[
   k^{(n)} = k^{(n-1)} \frac{|\mathbf{B}(\mathbf{x}^{(n)})|}{|\mathbf{B}(\mathbf{x}^{(n-1)})|}
   \]  

3. Check the convergence
   \[
   \frac{|k^{(n)} - k^{(n-1)}|}{|k^{(n)}|} < \epsilon_k
   \]
   and
   \[
   \frac{|\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}|}{|\mathbf{x}^{(n)}|} < \epsilon_x
   \]
   When either of them is not true, return Step 2, otherwise exit.

$\dagger$ Note that we typically do not need to have a full solve. The currently code will only perform one Newton iteration.

Also in this simplified algorithm, the solution is automatically normalized making $|\mathbf{B}(\mathbf{x})| = k$. We can do post-processing to normalize the solution so that $|\mathbf{x}| = c$, where $|.|$ can be any norm and $c$ is a scalar constant. If the smallest and second smallest eigenvalue are close to each other, i.e. the dominance ratio is close to one, inverse power iterations converge very slowly. In such a case, we can apply accelerations, such as Chebyshev acceleration, based on the on-the-fly estimation of the dominance ratio. The details of Chebyshev 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 $\mathbf{A}$ for preconditioning purposes.

#### 9.4.1 \texttt{l\_max\_its}

Refer to \texttt{l\_max\_its} in \texttt{Steady}.

#### 9.4.2 \texttt{pfactor}

Description: Relative linear tolerance
Algorithm 2 Modified inverse power iteration

1. Initialization:
   \[
   k^{(0)} = k_0 \\
   x^{(0)} = k_0 \frac{x_0}{|B(x_0)|}.
   \]

2. Update \( x \) by performing a steady-state solve and update \( k \):
   \[
   A(x^{(n)}) = \frac{1}{k^{(n-1)}} B(x^{(n-1)}) \\
   k^{(n)} = |B(x^{(n)})|
   \]

3. Check the convergence
   \[
   \frac{|k^{(n)} - k^{(n-1)}|}{|k^{(n)}|} < \epsilon_k
   \]
   and
   \[
   \frac{|x^{(n)} - x^{(n-1)}|}{|x^{(n)}|} < \epsilon_x.
   \]

   When either of them is not true, return Step 2, otherwise exit.

Data type: Real

Default value: 0.01

Syntax: Executioner/pfactor

Note: This parameter is used to control the solution of Eq. (46). It is 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
   \[
   \|F(x^{(n-1)}) - J(x^{(n-1)}) \delta x^{(n)}\|_2 < \max(\text{rtol} \|F(x^{(n-1)})\|_2, \text{atol}), \tag{48}
   \]

where \( \text{rtol} \) is this parameter and \( \text{atol} \) is \( l_{\text{abs\_step\_tol}} \).

9.4.3 \( l_{\text{abs\_step\_tol}} \)

Refer to \( l_{\text{abs\_step\_tol}} \) in Steady.

9.4.4 \( \text{line\_search} \)

Refer to \( \text{line\_search} \) in Steady.

9.4.5 \( \text{solve\_type} \)

Refer to \( \text{solve\_type} \) in Steady.

9.4.6 \( \text{petsc\_options} \)

Refer to \( \text{petsc\_options} \) 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 $x_0$
Data type: Logical
Default value: true
Syntax: Executioner/auto_initialization

9.4.10  eig_check_tol

Description: Eigenvalue convergence tolerance $\epsilon_k$
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

281
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 \(|x^{(n)} - x^{(n-1)}|\)
Data type: String
Default value: <empty>
Syntax: Executioner/xdiff
Note: If this processor is not given, no convergence check with Eq. (47) will be performed.

9.4.16  *sol_check_tol*

Description: Relative tolerance on the solution \(\epsilon_x\)
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 \(|x|\)
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 x to make |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 intermediate 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

\[ A(x) = \frac{1}{k} B(x), \]

\[ k = |B(x)|. \]

This new viewpoint allows using the Newton method to solve the eigenvalue problem. However, as the eigenvalue problem has an infinite number of solutions (i.e. the higher harmonics) apart from the fundamental mode, we have to ensure that the nonlinear solution does not converge to either of the higher harmonics but to the fundamental mode. To this end we need to ensure that the initial guess supplied to NonlinearEigen is sufficiently close to the fundamental mode. This can be achieved by performing 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 described in "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

\[ A(x) = \frac{1}{|B(x)|} B(x). \]

Again we can use PJFNK (preconditioned Jacobian-free Newton Krylov) method to solve this nonlinear problem.
9.5.1 \textit{l\_max\_its}

Refer to \textit{l\_max\_its} in \textit{Steady}.

9.5.2 \textit{pfactor}

Refer to \textit{pfactor} in \textit{InversePowerMethod}.

9.5.3 \textit{l\_abs\_step\_tol}

Refer to \textit{l\_abs\_step\_tol} in \textit{Steady}.

9.5.4 \textit{line\_search}

Refer to \textit{line\_search} in \textit{Steady}.

9.5.5 \textit{solve\_type}

Refer to \textit{solve\_type} in \textit{Steady}.

9.5.6 \textit{nl\_abs\_step\_tol}

Refer to \textit{nl\_abs\_step\_tol} in \textit{Steady}.

9.5.7 \textit{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 is used to set the default value of a PETSc control parameter \texttt{-snes\_atol}. Users are allowed to directly use the PETSc control parameter \texttt{-snes\_atol} to overwrite this parameter.

9.5.8 \textit{nl\_max\_funcs}

Refer to \textit{nl\_max\_funcs} in \textit{Steady}.

9.5.9 \textit{nl\_max\_its}

Refer to \textit{nl\_max\_its} in \textit{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 is 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 soluble boron concentration, control rod bank position, etc., (which is affecting the operators $A(p,x)$ and/or $B(p,x)$), to force the eigenvalue $k$ of Eq. (45) to assume a specific value, usually $k = 1$. In case $k = 1$, the system is referred to as critical and typically therefore one is the targeted eigenvalue. For this reason the described kind of calculation is often called criticality search. We can think of 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

$$p^{(n+1)} = p^{(n)} - \left(\frac{dk}{dp}\bigg|_{p=p^{(n)}}\right)^{-1} (k(p^{(n)}) - k_{target}), \quad (52)$$

with $\frac{dk}{dp}\bigg|_{p=p^{(n)}}$ is approximated by

$$\frac{dk}{dp}\bigg|_{p=p^{(n)}} = \frac{k(p^{(n)}(1+\epsilon)) - k(p^{(n)})}{\epsilon p^{(n)}}, \quad (53)$$

where $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ should be. On the other hand, if large $\epsilon$ is used, we
ded up with secant method, which converges slower than Newton’s method. Newton’s method requires a fairly
close initial guess of $p$ if the dependence of $k$ on $p$ is complicated. In such a case, we can perform few bisection
evaluations to obtain an initial guess.

9.6.1  \textit{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 \textit{CriticalitySearch} to adjust the parameter. This function must be used by the system or be participating in the
residual evaluation, thus adjusting the parameter provided by this function changes the eigenvalue of the system.

9.6.2  \textit{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  \textit{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 \textit{bisection} is greater than
zero. It is also used to check if the Newton update is out of bound.

9.6.4  \textit{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 \textit{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  *abs_eig_tol*

Description: Absolute tolerance on eigenvalue for convergence during Newton iterations
Data type: Real
Default value: $10^{-50}$
Syntax: Executioner/abs_eig_tol
Note: Parameter has no effect on the bisection performed before the Newton solve.

9.6.7  *target_eigenvalue*

Description: Targetted eigenvalue $k_{\text{target}}$
Data type: Real
Default value: 1
Syntax: Executioner/target_eigenvalue

9.6.8  *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.

9.6.9  *newton_pi*

Description: Number of free power iterations for Newton search
Data type: Integer
Default value: 0
Syntax: Executioner/newton_pi
9.6.10  \textit{pre}_pi

Description: Number of starting power iterations
Data type: Integer
Default value: 4
Syntax: Executioner/pre.pi

9.6.11  \textit{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.

9.6.12  \textit{perturbation}

Description: Perturbation strength $\epsilon$ on the parameter $p$ for evaluating $dk/dp$
Data type: Real
Default value: $10^{-6}$
Syntax: Executioner/perturbation

9.6.13  \textit{l_max_its}

Refer to \textit{l_max_its} in \textit{Steady}.

9.6.14  \textit{pfactor}

Refer to \textit{pfactor} in \textit{InversePowerMethod}.

9.6.15  \textit{l_abs_step_tol}

Refer to \textit{l_abs_step_tol} in \textit{Steady}.

9.6.16  \textit{line_search}

Refer to \textit{line_search} in \textit{Steady}.

9.6.17  \textit{solve_type}

Refer to \textit{solve_type} in \textit{Steady}.
9.6.18  \textit{nl\_abs\_step\_tol}

Refer to \textit{nl\_abs\_step\_tol} in \textit{Steady}.

9.6.19  \textit{source\_abs\_tol}

Refer to \textit{source\_abs\_tol} in \textit{NonlinearEigen}.

9.6.20  \textit{nl\_max\_funcs}

Refer to \textit{nl\_max\_funcs} in \textit{Steady}.

9.6.21  \textit{nl\_max\_its}

Refer to \textit{nl\_max\_its} in \textit{Steady}.

9.6.22  \textit{nl\_rel\_step\_tol}

Refer to \textit{nl\_rel\_step\_tol} in \textit{Steady}.

9.6.23  \textit{petsc\_options}

Refer to \textit{petsc\_options} in \textit{Steady}.

9.6.24  \textit{petsc\_options\_iname}

Refer to \textit{petsc\_options\_iname} in \textit{Steady}.

9.6.25  \textit{petsc\_options\_value}

Refer to \textit{petsc\_options\_value} in \textit{Steady}.

9.6.26  \textit{auto\_initialization}

Refer to \textit{auto\_initialization} in \textit{InversePowerMethod}.

9.6.27  \textit{k0}

Refer to \textit{k0} in \textit{InversePowerMethod}.

9.6.28  \textit{time}

Refer to \textit{time} in \textit{InversePowerMethod}.

9.6.29  \textit{normalization}

Refer to \textit{normalization} in \textit{InversePowerMethod}.
9.6.30  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, but we partition it into two parts $F$ and $G$:

\[
\begin{align*}
F_y(x) &= 0 \\
G_x(y) &= 0.
\end{align*}
\] (54)

But the operator $F_x$ is made to depend on $x$ explicitly though auxiliary variables, material properties, postprocessors, multiapps 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

\[
F_{y(n-1)}(x^{(n)}) = 0,
\] (55)

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 to 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 $x$ is problem dependent. The convergence can be checked with the norm of the non-lagged residual

\[
\left\| F_x(x^{(n)}) \right\| < \epsilon.
\] (56)

One more residual evaluation is needed for this check. The convergence can also be checked based on any postprocessors which depend on $x$ either directly or indirectly. As an example, we demonstrate how this executioner are used to drive NDA (nonlinear diffusion acceleration) calculations in Sec. 3.7.
9.7.1 Steady parameters

All parameters of Steady are valid parameters. Refer to Steady for more information. It is noted that disable_picard_residual_norm_check is set to true internally by PicardSteady, thus disable_picard_residual_norm_check, picard_rel_tol, picard_abs_tol and picard_force_norms are all suppressed and invalid.

9.7.2 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 performs 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.

9.7.3 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 identical 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. 3.7.

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 Steady.
9.8.3  *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

9.8.4  *picard_rel_tol*

Refer to *picard_rel_tol* in Steady.

9.8.5  *picard_abs_tol*

Refer to *picard_abs_tol* in Steady.

9.8.6  *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 iteration.

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
Date type: Integer
Default value: 0
Syntax: Executioner/extra_free_pi

9.9  *Richardson*
There are cases the general nonlinear problem Eq. (34) can be written into
\[ L(x) = R(x). \]  
(57)

Then we can perform an iteration
\[ L(x^{(n)}) = R(x^{(n-1)}). \]  
(58)

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, a fraction of kernels representing operator \( R \) are switched to 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 a measure of the iteration error for determining the convergence. The postprocessor must be executed on linear when more than one Richardson is going to be 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.
\[
\left\| x^{(n-1)} - x^{(n)} \right\| < \max(rtol \left\| x^{(0)} - x^{(1)} \right\|, atol),
\]  
(59)

where \( \left\| x^{(n-1)} - x^{(n)} \right\| \) 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  \textit{richardson\_abs\_tol}

Description: Absolute tolerance for Richardson iterations

Data type: Real

Default value: $10^{-4}$

Syntax: Executioner/richardson\_abs\_tol

9.9.6  \textit{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  \textit{debug}

Description: True to add more screen print-outs for debugging purpose

Data type: Logical

Default value: true

Syntax: Executioner/debug

9.10  \textit{AMGUpdate}

This executioner is a special Richardson executioner for CFEM SN. At each iteration, it first evaluates the residual of the old solution

$$r(x^{(n-1)}) = -L(x^{(n-1)}) + R(x^{(n-1)}).$$  \hfill (60)

Then it solves the linear problem

$$\hat{L}\delta x = r(x^{(n-1)}),$$  \hfill (61)

where $\hat{L}x$ is an approximation of $L(x)$, and then it updates the solution

$$x^{(n)} = x^{(n-1)} + \delta x.$$  \hfill (62)

Currently we use the block-diagonal Jacobian, where each block corresponds to a primal variable. This executioner currently uses the hard-coded BoomerAMG PETSc solver for Eq. (61) 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 \texttt{petsc\_options}, \texttt{petsc\_options\_iname} and \texttt{petsc\_options\_value}. The rest of parameters in Richardson are valid parameters. This executioner works only when \texttt{scheme} is SAAF-CFEM-SN or LS-CFEM-SN.

9.10.1  \textit{xdiff}

Refer to \textit{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 below 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  *amg_strong_threshold*

Description: Setting BoomerAMG strong threshold
Data type: Real
Default value: 0.7
Syntax: Executioner/amg_strong_threshold
Note: This parameter must be greater than or equal to 0.25 and less than or equal to 1.

9.10.12  *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*

Like *AMGUpdate* this executioner is also a special Richardson executioner developed for DFEM SN. It is a matrix-free scheme on solving Eq. (61). 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. (61) on local elements. We also call this solver a sweeper. As *AMGUpdate*, all the parameters in *Steady* are invalid. The rest of parameters in *Richardson* are valid parameters. This executioner works only if *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.

9.11.7 *use_new_sweeper*

Description: True to use the new sweeper  
Data type: Logical  
Default value: False  
Syntax: Executioner/use_new_sweeper  
Note: This parameter allows users to use the new transport sweeper, which is more efficient than the default one because of the pre-assembled element matrices.

9.12 *IQS (improved quasi-static)*

This is an executioner derived from Transient for solving transient problems. All the parameters in Transient are also valid. It is noted that currently only CFEM-Diffusion, DFEM-SN and SAAF-CFEM-SN supports IQS. Multi-scheme transient calculations with the above three schemes can use IQS executioner. This executioner is based on the observation that the amplitude of the solution changes faster than the angular, spatial and energy distribution of the solution (*shape*) in radiation transport simulations. The shape of the solution thus can be solved with much larger time steps (macro-steps) along with amplitude update in lots of micro-steps within one macro-step. Because the amplitude equation is zero-dimensional in space, often referred to as the point-kinetics equation (PKE) for neutron transport, amplitude updates with lots of micro-steps are inexpensive compared with shape solves. The cost on a shape solve is about the same as a normal time step solve in Transient executioner. Thus, this executioner can significantly reduce the CPU time without loss of the accuracy for certain applications.

Users can use *do_iqs_transient* to switch on and off IQS. When *do_iqs_transient* is false, no IQS is performed and the executioner does a normal transient as Transient, but it still can output PKE parameters into a CSV file indicated by *pke_param_csv* at the time when PKE parameters are updated. There are two approaches to handle the coupling between the shape solve and the amplitude update. One considers them as a coupled nonlinear system and solve them iteratively with a Picard iteration along with other multiphysics couplings if there are any. The other uses the predictor-corrector method to first solve for the shape and then modulate the shape with the amplitude from a following micro-step calculation within the same macro-step. Users can switch between Picard iteration and
predictor-corrector with the parameter predictor_corrector. It is believed that the better choice is problem dependent. The shape calculation is the main solve while the amplitude update happens at the beginning of the time step in Picard iteration. The convergence of Picard iteration is controlled by picard_max_its, picard_rel_tol, picard_abs_tol in Transient in addition to IQS_error_tol provided by this executioner.

We typically assume that PKE parameters vary linearly in a macro-step and interpolate them with the PKE parameters at the beginning and the end times of macro-steps. It is sometimes desired to update PKE parameters more frequently in a macro-step particular due to multiphysics feedback effects. Users can either use num_param_updates to set the number of parameter updates per macro-step uniformly distributed in the step or set param_adaptive to true to let the executioner decide when PKE parameters are to be updated automatically. When param_adaptive is true, a PKE parameter update is triggered when the amplitude or power exceeds the param_adapt_fac factor from the last update. Although the shape does not change during this PKE parameter update, the affected amplitude can change quantities of other physics in multiphysics settings which can in turn change PKE parameters due to feedback. Users can use param_max_pke_iter and param_pke_iter_tol to set the desired number of PKE-multiphysics iterations for all PKE parameter updates.

Rattlesnake provides three time integration schemes for PKE solve, backward Euler, Crank-Nicolson and SDIRK33, that users can choose with pke_scheme. The number of micro-steps within a macro-step can be either static with num_pke_steps or adaptive with pke_time_adaptation, pke_dt_initial, pke_time_adapt_tol, pke_max_dt, pke_safety_factor with the step doubling adaptation. The step doubling technique involves estimating the local truncation error $e_n$ for a certain time step by taking the difference between a solution with one full step and a solution with two half steps:

$$
ed_n = \frac{|p_{\text{half}} - p_{\text{full}}|}{\max(p_{\text{half}}, p_{\text{full}})}, \quad (63)$$

where $p$ is the power at the current time step. In an asymptotic range, the error converges with the time step size as $e_n \approx Ch^p$, where $h$ is the full step size and $p$ is the local convergence order. This order is 2, 3 and 4 for backward Euler, Crank-Nicolson and SDIRK33 respectively for smooth solutions in time. With step doubling, we first reduce the time step size so that the error is smaller than a prescribed tolerance $e_{\text{tol}}$. Then the magnitude of the next step will be calculated based on the error:

$$
\Delta t_{\text{new}} = s\Delta t \left(\frac{e_{\text{tol}}}{e_n}\right)^{\frac{1}{p}}, \quad (64)
$$

where $s$ is a safety factor typically smaller than 1. If the step is too large, the step will be repeated with a smaller step calculated with the same equation. $e_{\text{max}}$ is usually less than $e_{\text{tol}}$ to better guarantee that the calculated $\Delta t_{\text{new}}$ will pass the error criteria so time step repeats can be reduced or avoided.

### 9.12.1 Transient parameters

All parameters of Transient are valid parameters. Refer to Transient for more information. It is emphasized that picard_max_its, picard_rel_tol, and picard_abs_tol documented in Transient are used for the Picard IQS. It is noted that verbose can be used to output detailed diagnostics for IQS.

### 9.12.2 initial_amplitude

Description: Initial amplitude for PKE solve

Data type: Real

Default value: 1

Syntax: Executioner/initial_power

Note: Initial power will be initial shape power multiplied with this parameter.
9.12.3  
**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 parameters will be outputted in the CSV file when they are updated.

9.12.4  
**do_iqs_transient**

Description: True to perform IQS transient otherwise do 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 machinery is required but no IQS calculations is needed. For example, when **pke_param_csv** is provided for dumping the PKE parameters.

9.12.5  
**predictor_corrector**

Description: True if doing predictor corrector version of IQS

Data type: Logical

Default value: False

Syntax: Executioner/predictor_corrector

Note: Predictor corrector scheme is different from Picard IQS in that there is no Picard iteration for shape solve and amplitude update on macro-step. Thus the cost per macro-step is smaller.

9.12.6  
**IQS_error_tol**

Description: The relative tolerance on normalization factor required for convergence during Picard 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 with the normalization factor that enforce the uniqueness of the factorization. The default value means this parameter is probably not contributing to the convergence check.

9.12.7  
**output_micro_csv**

Description: A CSV file name for outputting power on all micro time steps

Data type: String

Default value: `<empty>`
Syntax: Executioner/output_micro_csv

Note: When this parameter is provided, amplitude of all micro-steps of all macro-steps will be outputted in the CSV file. The file will be updated at the end of every macro-step with all micro-steps in the latest macro-step.

9.12.8 shape_postprocessor

Description: A postprocessor used along with amplitude output
Data type: String
Default value: 1

Syntax: Executioner/shape_postprocessor

Note: The parameter is only active when output_micro_csv is provided. The postprocessor will be used to modulate the amplitude profile with a linear interpolation on micro-steps. The default value means that the constant value one will be used, i.e. only the amplitude will be outputted. This parameter is useful for outputting on micro-steps the real powers with the postprocessor being a shape power (i.e. the power evaluated from the shape only). The name of this parameter contains the word ‘shape’, but it can be any postprocessor and does not have to be related with shape. If do_iqs_transient is false, the CSV file will actually contain the postprocessor values on all macro-steps multiplied with initial_amplitude.

9.12.9 Subset of parameters for PKE solve

This section contains all the control parameters for amplitude update, or how PKE is solved.

9.12.9.1 pke_scheme

Description: PKE time discretization scheme
Data type: Enumeration (/BE/CN/RK/)
Default value: RK

Syntax: Executioner/pke_scheme

Note: BE, CN and RK stand for backward Euler, Crank-Nicolson and SDIRK33 respectively.

9.12.9.2 pke_time_adaptation

Description: True if doing time step doubling adaptation for PKE
Data type: Logical
Default value: True

Syntax: Executioner/pke_time_adaptation

9.12.9.3 num_pke_steps

Description: Number of micro timesteps for PKE solves in each macro timestep
Data type: Integer
Default value: 100
9.12.9.4  pke_dt_initial

Description: Initial micro step size for PKE as fraction of macro step size
Data type: Real
Default value: $10^{-5}$
Syntax: Executioner/pke_dt_initial
Note: This parameter must be greater than 0 and less than 1. This parameter is used when pke_time_adaptation is set to true.

9.12.9.5  pke_time_adapt_tol

Description: Tolerance criteria for time step doubling adaptation
Data type: Real
Default value: $10^{-12}$
Syntax: Executioner/pke_time_adapt_tol
Note: This parameter is $e_{\text{max}}$ and is used when pke_time_adaptation is set to true.

9.12.9.6  pke_max_dt

Description: Maximum step size change $\Delta t_{\text{new}}$ for step doubling adaptation
Data type: Real
Default value: 1000
Syntax: Executioner/pke_max_dt
Note: This parameter is used when pke_time_adaptation is set to true.

9.12.9.7  pke_safety_factor

Description: Safety factor for finding next time step size in step doubling adaptation
Data type: Real
Default value: 0.8
Syntax: Executioner/pke_safety_factor
Note: This parameter is $s$ and is used when pke_time_adaptation is set to true.
9.12.9.8  *pke_result_lines*

Description: Limit to the number of micro-steps output at the beginning and the end of each macro-step
Data type: Integer
Default value: 5
Syntax: Executioner/pke_result_lines

Note: This parameter is active when *do_iqs_transient* is true and *verbose* is true. It can be useful for reducing the amount of screen outputs when the number of micro-steps is large.

9.12.10  **Subset of parameters for PKE parameter update**

This section contains all the parameters controlling how PKE parameters are updated within a macro-step.

9.12.10.1  *param_adaptive*

Description: True if doing adaptive time steps for PKE parameter update
Data type: Logical
Default value: False
Syntax: Executioner/param_adaptive

9.12.10.2  *num_param_updates*

Description: Number of PKE parameter updates in each macro time step
Data type: Integer
Default value: 1
Syntax: Executioner/param_updates

Note: This parameter is active when *param_adaptive* is set to false.

9.12.10.3  *param_adapt_fac*

Description: Maximum increase of amplitude to initiate a PKE parameter update
Data type: Real
Default value: 10
Syntax: Executioner/param_adapt_fac

Note: This parameter is active when *param_adaptive* is set to true.
9.12.10.4 \textit{param\_max\_pke\_iter}

Description: Maximum number of PKE-multiphysics iterations
Data type: Integer
Default value: 1
Syntax: Executioner/param\_max\_pke\_iter
Note: The default value means no PKE-multiphysics iteration will be performed.

9.12.10.5 \textit{param\_pke\_iter\_tol}

Description: Tolerance on power for PKE-multiphysics iteration
Data type: Real
Default value: $10^{-12}$
Syntax: Executioner/param\_pke\_iter\_tol
Note: This parameter is active when \textit{param\_max\_pke\_iter} is greater than 1.
10 Postprocessors and User Objects

Postprocessors can be used to create useful output data e.g. by integrating/averaging/finding maximum values of variables and users need to know about. User objects in Rattlesnake serves more general purpose other than postprocessing and data condensation. Some UserObjects directly participate in residual evaluations, but users do not necessarily need to know about them for using Rattlesnake. Therefore, we only document those that are useful for postprocessing purpose. This section is organized as follows. Sec. 10.1 presents three common parameters shared by all processors and user objects. Sec. 10.2 and Sec. 10.3 introduce MOOSE postprocessors and MOOSE vector postprocessors and user objects respectively. Rattlesnake postprocessors are grouped into general postprocessors in Sec. 10.4, elemental postprocessors in Sec. 10.5, side postprocessors in Sec. 10.7, and nodal postprocessors in Sec. 10.6. Postprocessors for pebble tracking calculations in Rattlesnake are special and are documented under Sec. 10.8. Rattlesnake user objects are arranged into Sec. 10.10.

10.1 Common Parameters for Postprocessors

All postprocessors including vector postprocessors and user objects share few common parameters provided by MOOSE, which are documented as follows:

10.1.1 execute_on

Description: To determine when the object is evaluated
Data type: enumeration (/initial/linear/nonlinear/timestep_begin/timestep_end/custom/)
Default value: timestep_end
Syntax: UserObjects/*/execute_on, or Postprocessors/*/execute_on, or VectorPostprocessors/*/execute_on
Note: This parameter accepts multiple enumerations.

10.1.2 use_displaced_mesh

Description: Whether or not this object should use the displaced mesh for computation
Data type: Logical
Default value: false
Syntax: UserObjects/*/use_displaced_mesh, or Postprocessors/*/use_displaced_mesh, or VectorPostprocessors/*/use_displaced_mesh

All postprocessors including vector postprocessors share a common parameter:

10.1.3 outputs

Description: Vector of output names were you would like to restrict associated with this object
Data type: Vector of strings
Default value: <empty>
Syntax: Postprocessors/*/use_displaced_mesh, or VectorPostprocessors/*/use_displaced_mesh
Note: By default, postprocessors will be outputted to all supported outputs.

10.2 MOOSE Postprocessors

The complete list of MOOSE postprocessors can be found in Fig. 29.

10.3 MOOSE Vector Postprocessors and User Objects

The complete list of MOOSE vector postprocessors and user objects can be found in Fig. 30. Their parameters are fairly straightforward and can be found in MOOSE documents.

10.4 Rattlesnake General Postprocessors

10.4.1 PostprocessorReduction

This postprocessor does a reduction operation on multiple processors. This includes summation, product, taking the maximum or computing the L2 norm of a list of postprocessors.

10.4.1.1 postprocessors

Description: Postprocessors to be operated on
Data type: Vector of strings
Default value: <required>
Syntax: Postprocessors/*/postprocessors

10.4.1.2 reduction_type

Description: Reduction type
Data type: Enumeration (/sum/l2norm/max/product/)
Default value: sum
Syntax: Postprocessors/*/reduction_type
Figure 29  Moose postprocessors.
Figure 30  Moose vector postprocessors and user objects.
10.4.1.3  scaling_factor

Description: The scaling factor to be multiplied with

Data type: Real
Default value: 1
Syntax: Postprocessors/*/scaling_factor

10.4.2  DynamicalScale

This postprocessor evaluates \(\frac{1}{2} \left[ p^{(n)} + p^{(n-1)} \right] \) of a postprocessor \(p\), where \(p^{(n)}\) denotes the current value of the postprocessor and \(p^{(n-1)}\) is the old value of the postprocessor. When \(p^{(n)}\) is equal to \(p^{(n-1)}\), this postprocessor returns the maximum real number.

(Parameters are to be added.)

10.4.3  FindMaximumError

This postprocessor evaluates \(\max_{i=1}^{N} \frac{p_i}{v_i}\). Its name contains error word because typically \(p_i\) represents the error of certain quantity of \(v_i\).

(Parameters are to be added.)

10.4.4  PostprocessorRatio

This postprocessor simply evaluates the ratio of two any other postprocessors.

(Parameters are to be added.)

10.4.5  ReportBalance

This postprocessor extracts and reports the balance for a certain mesh block and a certain group. Balance is defined as the relative difference of the production and loss rates of radiation.

(Parameters are to be added.)

10.4.6  SPHFactor

This postprocessor extracts and reports the SPH (super-homogenization) factor for a particular equivalence zone and a particular group.

(Parameters are to be added.)

10.4.7  PrintIQSAmplitude

This postprocessor print amplitude information during transient with IQS executioner, refer to sectIQS.

(Parameters are to be added.)
10.5 Rattlesnake Elemental Postprocessors

10.5.1 FluxReactionIntegral

This postprocessor evaluates the reaction rates.
(Parameters are to be added.)

10.5.2 ElementAverageMaterialProperty

This postprocessor evaluates the average of a scalar material property.
(Parameters are to be added.)

10.5.3 ElementL2Diff

This postprocessor evaluates the L2 norm of the difference between the current and the old solution of a field variable.
(Parameters are to be added.)

10.5.4 ElementMultiVarL2Diff

This postprocessor evaluates the L2 norm of the difference between two sets of field variables as
\[ \sqrt{\int_V \sum_{i=1}^{n} (u_i(x) - v_i(x))^2 \, dx}, \]
where \( V \) is the integration domain; \( u \) and \( v \) are the names of the two sets of variables.
(Parameters are to be added.)

10.5.5 FieldDynamicalScale

This postprocessor evaluates
\[ \Delta t \frac{1}{2} \sqrt{\int_V \sum_{i=1}^{n} (u_i(x) + u_{i(n-1)})^2 \, dx} \]
\[ \sqrt{\int_V (u_i(x) - u_{i(n-1)})^2 \, dx}, \]
where \( u^{(n)} \) denotes the current solution of a variable and \( u^{(n-1)} \) is for the old solution of the variable. When \( u^{(n)} \) is equal to \( u^{(n-1)} \), this postprocessor returns the maximum real number.
(Parameters are to be added.)

10.5.6 InnerProduct

This postprocessor evaluates inner product of two variables optionally with a material property and a scaling factor:
\[ c (\nu, \sigma u)_V = c \int_V \sigma v u \, dx. \]
(Parameters are to be added.)
10.5.7 FluxMomentL2Error

This postprocessor computes the contribution of a given moment to the angular flux L2-error. It uses the moment of the reference solution and of the numerical solution. Since the reference solution is potentially defined using more moments than the numerical solution, the option to only compute the L2-norm of that particular moment of the reference solution is given. This postprocessor is used by the action set through `TransportSolutionError`. It derives from the MOOSE ElementIntegralPostprocessor class.

10.5.7.1 execute_on

Refer to `execute_on` in `Postprocessors`.

10.5.7.2 use_displaced_mesh

Refer to `use_displaced_mesh` in `Postprocessors`.

10.5.7.3 outputs

Refer to `outputs` in `Postprocessors`.

10.5.7.4 variable

Description: The name of the moment from the numerical solution
Data type: Vector of strings
Default value: `<required>`
Syntax: Postprocessors/*/variable

10.5.7.5 g

Description: Energy group index of the considered moment
Data type: Unsigned integer
Default value: `<required>`
Syntax: Postprocessors/*/g

10.5.7.6 l

Description: Degree of the considered moment
Data type: Unsigned integer
Default value: `<required>`
Syntax: Postprocessors/*/l
10.5.7.7  \( m \)

Description: Order of the considered moment
Data type: Integer
Default value: \(<\text{required}>\)
Syntax: Postprocessors/*/m

10.5.7.8  \( function \)

Description: The reference solution to compare against, it MUST derive from TransportSolutionFunction
Data type: Function name
Default value: \(<\text{required}>\)
Syntax: Postprocessors/*/function

10.5.7.9  \( \text{coefficient} \)

Description: Coefficient which the postprocessor must be multiplied with
Data type: Real
Default value: 1
Syntax: Postprocessors/*/coefficient
Note: If coefficient is left to its default value, the results will only be proportional to the actual contribution of a given moment to the angular flux L2-error. If one desires to use this postprocessor to compute the entire angular flux L2-error, \( \text{coefficient} \) needs to be set appropriately (depending on the degree and order of the considered moment). All of this is done automatically by \text{TransportSolutionError} (among other things).

10.5.7.10  \( use\_zero\_variable \)

Description: True to ignore the value of variable (effectively assuming that the considered moment of the numerical solution is zero)
Data type: bool
Default value: false
Syntax: Postprocessors/*/use_zero_variable

10.5.8  \text{MaterialGroupAverage}

This postprocessor evaluates the average of one component of a vector material property over a domain.
(Parameters are to be added.)

10.5.9  \text{MaterialL2Diff}

This postprocessor evaluates the L2 norm of the difference of a scalar stateful material property between current and old.
(Parameters are to be added.)
10.5.10 MaterialL2Comparison

This postprocessor evaluates the L2 norm of the difference of two scalar material properties.
(Parameters are to be added.)

10.5.11 PrintIQSPower

This postprocessor evaluates the integration of a variable and a material property and multiplies the integration with the amplitude. Volume average is an option provided by this postprocessor. This postprocessor must work with IQS executioner, refer to sectIQS.
(Parameters are to be added.)

10.6 Rattlesnake Nodal Postprocessors

10.6.1 NodalInnerProduct

This postprocessor simply sums the product of two variables on a set of node points.
(Parameters are to be added.)

10.7 Rattlesnake Side Postprocessors

10.7.1 PartialSurfaceCurrent

This postprocessor evaluates the surface currents on a side set. This postprocessor is currently only implemented for SAAF-CFEM-PN and all SN and diffusion schemes.

10.7.1.1 transport_system

Description: Name of the transport system

Data type: String

Default value: <required>

Syntax: Postprocessors/*/transport_system

Note: The postprocessor uses this parameter to interact with the transport system to determine which variables should be used to compute the partial current.

313
10.7.1.2 **boundary**

Description: The list of boundary names from the mesh where this boundary condition applies
Data type: Vector of strings
Default value: `<required>`
Syntax: `Postprocessors/*/boundary`

Note: If `transport_system` is a SAAF-CFEM-PN transport system, all the boundaries provided in this input parameter that are not vacuum boundaries should be added to `partial_current_required_on`.

10.7.1.3 **fastest_group_index**

Description: Index of the fastest group included in reduction
Data type: Unsigned integer
Default value: 0
Syntax: `Postprocessors/*/fastest_group_index`

Note: The default value means that no energy groups at the fast energy end are removed from the reduction.

10.7.1.4 **most_thermal_group_index**

Description: Blocks on which the reaction rate is to be evaluated
Data type: Index of the fastest group included in reduction
Default value: \( G - 1 \)
Syntax: `Postprocessors/*/most_thermal_group_index`

Note: The default value means that no energy groups at the thermal energy end are removed from the reduction.

10.7.1.5 **execute_on**

Refer to `execute_on` in `Postprocessors`.

---

10.8 **Rattlesnake Postprocessors for Pebble-Tracking-Transport**

10.8.1 **PebbleReactionRate**

This postprocessor evaluates a reaction rate over the pebble packing subdomain.
10.8.1.1  **block**

Description: Blocks on which the reaction rate is to be evaluated
Data type: Vector of block names
Default value: `<empty>`
Syntax: UserObjects/*/block
Note: Blocks must be one of the pebble subdomains.

10.8.1.2  **execute_on**

Refer to `execute_on` in *Postprocessors*.

10.8.1.3  **use_displaced_mesh**

Refer to `use_displaced_mesh` in *Postprocessors*.

10.8.1.4  **outputs**

Refer to `outputs` in *Postprocessors*.

10.8.1.5  **scalar_fluxes**

Description: Scalar fluxes to be used for evaluating the reaction rate
Data type: Vector of strings
Default value: `<required>`
Syntax: Postprocessors/*/scalar_fluxes

10.8.1.6  **group_indices**

Description: Group indices of all coupled scalar fluxes
Data type: Vector of integers
Default value: `<empty>`
Syntax: Postprocessors/*/scalar_fluxes
Note: Empty implies the natural ordering from 1 to the number of fluxes. If this parameter is given, its size must be equal to the size of `scalar_fluxes`.

10.8.1.7  **reaction_type**

Description: Neutron reaction type
Data type: Enumeration (/total/scattering/fission/nufission/kappafission/absorption/removal/
  gamma/alpha/twoalpha/twon/threen/fourn/nproton/proton/deuteron/triton/)
Default value: `<total>`
Syntax: Postprocessors/*/reaction_type
10.8.1.8  **bottom_left**

Description: The bottom left point (in x, y, z with spaces in-between) of the box to select the pebbles  
Data type: Vector of real  
Default value: <(-maximum real, -maximum real, -maximum real)>  
Syntax: Postprocessors/*/bottom_left  
Note: The size of this parameter is equal to the dimension of the mesh. This parameter combined with *top_right* forms a bounding box in which the reaction rate of all pebbles will be evaluated. Default value means almost no bound at bottom left for the bounding box.

10.8.1.9  **top_right**

Description: The top right point (in x, y, z with spaces in-between) of the box to select the pebbles  
Data type: Vector of real  
Default value: <(maximum real, maximum real, maximum real)>  
Syntax: Postprocessors/*/top_right  
Note: The size of this parameter is equal to the dimension of the mesh. This parameter combined with *bottom_left* forms a bounding box in which the reaction rate of all pebbles will be evaluated. Default value means almost no bound at top right for the bounding box.

10.8.1.10  **including_pebble_ids**

Description: The list of IDs of pebbles on which the reaction rate is to be evaluated  
Data type: Vector of integers  
Default value: <empty>  
Syntax: Postprocessors/*/including_pebble_ids  
Note: Empty means that all pebbles in the bounding box specified by *bottom_left* and *top_right* will be included for the reaction rate evaluation.

10.8.1.11  **evaluate_difference**

Description: True to evaluate the reaction rate difference from the previous time step  
Data type: Logical  
Default value: False  
Syntax: Postprocessors/*/evaluate_difference

10.9  **Rattlesnake Vector Postprocessors**

All Rattlesnake vector postprocessors are also MOOSE VectorPostprocessor. Thus, they share the few common MOOSE parameters.
10.9.1 ElementVariableValuesAlongLine

This vector postprocessor samples the variable values along a line. It is different from the MOOSE LineValueSampler in that the sampling points is created through the intersection points of the line with elements. This line sampler allows sampling the variable values on two adjacent elements at the intersection points, which could be different with elemental variables.

10.9.1.1 execute_on

Refer to execute_on in Postprocessors.

10.9.1.2 use_displaced_mesh

Refer to use_displaced_mesh in Postprocessors.

10.9.1.3 outputs

Refer to outputs in Postprocessors.

10.9.1.4 start

Description: The beginning of the line
Data type: Vector of reals
Default value: <required>
Syntax: VectorPostprocessors/*/start
Note: Size of this parameter must agree with the mesh dimension.

10.9.1.5 end

Description: The end of the line
Data type: Vector of reals
Default value: <required>
Syntax: VectorPostprocessors/*/end
Note: Size of this parameter must agree with the mesh dimension.

10.9.1.6 number_points_per_elements

Description: Number of sampling values per element
Data type: Integer
Default value: 2
Syntax: VectorPostprocessors/*/number_points_per_elements
Note: This parameter must be greater than or equal to 2.
10.9.1.7 \textit{variable} \\
Description: The names of the variables that this VectorPostprocessor operates on \\
Data type: Vector of strings \\
Default value: \texttt{<required>} \\
Syntax: VectorPostprocessors/*/\textit{variable}

10.9.2 SPHFactors \\
This vector postprocessor reveals SPH factors for all energy groups and all equivalence zones. \\
(Parameters are to be added.)

10.9.3 PrintIQSAmplitudeVector \\
This vector postprocessor reveals the PKE solutions within a macro time step. It must work with IQS executioner, \texttt{refer to sectIQS}. \\
(Parameters are to be added.)

10.9.4 IntegralVariableElemIDVPP \\
This vector postprocessor computes integrals of a given coupled variable over all elements sharing the same deple-
tion/material/equivalence id.

10.9.4.1 \textit{variable} \\
The variable that is to be integrated.

10.9.4.2 \textit{elem_id_type} \\
The type of element id (depletion, material, or equivalence).

10.10 Rattlesnake User Objects

10.10.1 \textit{VariableCartesianCoreMap} \\
This user object evaluates integrated values of variables on blocks, materials, regions, assemblies, pins of assemblies and sectors and rings of fuel pins etc. over the entire solution mesh. The integrated values are then outputted either on screen or into a file in a readable format. The evaluation is performed 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 axially for output. 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 require the users’ input and are 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 33. 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. 31. It is noted that the mesh in the water in Fig. 31 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 falling into the grids are assigned 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 flexibility of this approach is limited, because 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>
<thead>
<tr>
<th>Table 33 Element variables for IDs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID type</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Material ID</td>
</tr>
<tr>
<td>Region ID</td>
</tr>
<tr>
<td>Assembly ID</td>
</tr>
<tr>
<td>Assembly ID</td>
</tr>
<tr>
<td>Assembly ID</td>
</tr>
<tr>
<td>Pin ID</td>
</tr>
<tr>
<td>Pin ID</td>
</tr>
<tr>
<td>Ring ID</td>
</tr>
<tr>
<td>Sector ID</td>
</tr>
</tbody>
</table>

10.10.1.1 execute_on

Refer to execute_on in Postprocessors.

10.10.1.2 use_displaced_mesh

Refer to use_displaced_mesh in Postprocessors.

10.10.1.3 print

Description: Vector of keywords controlling the information to be printed
Figure 31  Element IDs for the 2D C5G7 benchmark generated by INSTANT mesh generator.
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.

10.10.1.4  *output_in*

Description: Name of the file that data will be outputted in
Data type: String
Default value: <empty>
Syntax: UserObjects/*/output_in
Note: If this parameter is not set, data will be outputted on screen.

10.10.1.5  *regular_grid*

Description: Whether or not the grid is regular and constructed from grid coordinates in the input
Data type: Logical
Default value: false
Syntax: UserObjects/*/regular_grid

10.10.1.6  *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 $[grid_coord_x[i-1], grid_coord_x[i]], i = 1, \cdots, nx$.

10.10.1.7  *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 $[grid_coord_y[i-1], grid_coord_y[i]], i = 1, \cdots, ny$.
10.10.1.8  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 \( n_z \) is the size of this parameter minus one. The layer \( i \) in z direction is within \( [grid\_coord\_z[i - 1], grid\_coord\_z[i]], i = 1, \cdots, n_z]. \)

10.10.1.9  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 \( n_x \), 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.

10.10.1.10  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 \( n_y \), 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.

10.10.1.11  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.
10.10.2 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 if required on assemblies. It generates the map with either the fission neutron production rate or the power density on pins.

10.10.2.1 execute_on

Refer to execute_on in Postprocessors.

10.10.2.2 use_displaced_mesh

Refer to use_displaced_mesh in Postprocessors.

10.10.2.3 print

Refer to print in VariableCartesianCoreMap.

10.10.2.4 output_in

Refer to output_in in VariableCartesianCoreMap.

10.10.2.5 regular_grid

Refer to regular_grid in VariableCartesianCoreMap.

10.10.2.6 grid_coord_x

Refer to grid_coord_x in VariableCartesianCoreMap.

10.10.2.7 grid_coord_y

Refer to grid_coord_y in VariableCartesianCoreMap.

10.10.2.8 grid_coord_z

Refer to grid_coord_z in VariableCartesianCoreMap.

10.10.2.9 num_sub_grids_x

Refer to num_sub_grids_x in VariableCartesianCoreMap.
10.10.2.10  *num_sub_grids_y*

Refer to *num_sub_grids_y* in *VariableCartesianCoreMap*.

10.10.2.11  *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 determine where this user object is defined on and what variables and material properties are to be coupled in.

10.10.2.12  *print_assemblywise_fluxes*

Description: True to print assembly-wise total scalar fluxes
Data type: Logical
Default value: False
Syntax: UserObjects/*/print_assemblywise_fluxes

10.10.2.13  *print_groupflux*

Description: True to print fluxes for each group on all except pins
Data type: Logical
Default value: False
Syntax: UserObjects/*/print_groupflux

10.10.2.14  *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

10.10.2.15  *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
10.10.3  **SAAFWrapper**

This is the user object wrapping a high order transport system with SAAF-CFEM-SN scheme for nonlinear diffusion acceleration.

10.10.3.1  **input_file**

Description: The input file for the high order SAAF-CFEM-SN system  
Data type: String  
Default value: *required*  
Syntax: UserObjects/*/input_file

10.10.3.2  **accelerate_high_moments**

Description: True to accelerate higher angular moments when they are present  
Data type: Logical  
Default value: False  
Syntax: UserObjects/*/accelerate_high_moments

10.10.3.3  **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.

10.10.3.4  **from_variables**

Description: A list of variables to be transferred from diffusion to transport on diffusion system  
Data type: Vector of strings  
Default value: *empty*  
Syntax: UserObjects/*/from_variables  
Note: The variables can be either primal or auxiliary variables of the diffusion system.
10.10.3.5  

**to_variables**

Description: A list of auxiliary variables to be transferred from diffusion to transport on transport system

Data type: Vector of strings

Default value: <empty>

Syntax: UserObjects/*/to_variables

Note: The variables must be auxiliary variables of the transport system. Size of this parameter must be the same as the size of from_variables.

10.10.4  

**LSWrapper**

This is the user object wrapping a high order transport system with LS-CFEM-SN scheme for nonlinear diffusion acceleration.

10.10.4.1  

**input_file**

Refer to input_file in SAAFWrapper.

10.10.4.2  

**accelerate_high_moments**

Refer to accelerate_high_moments in SAAFWrapper.

10.10.4.3  

**initial_correction**

Refer to initial_correction in SAAFWrapper.

10.10.4.4  

**from_variables**

Refer to from_variables in SAAFWrapper.

10.10.4.5  

**to_variables**

Refer to to_variables in SAAFWrapper.

10.10.5  

**RattlesnakeVTKWriter**

This user object writes variables into a file in the simple legacy VTK format ([http://www.vtk.org/wp-content/uploads/2015/04/file-formats.pdf](http://www.vtk.org/wp-content/uploads/2015/04/file-formats.pdf)). It will create a file named <name>.vtk, where <name> is the name of this user object. This writer is particularly useful for visualizing elemental variables that are discontinuous across element faces. Currently, this writer supports TET4, HEX8, PRISM6 types of elements in 3D and TRI3, QUAD4 in 2D. More element types can be supported in the future. Mixed types of elements in the mesh are allowed. Users can use execute_on to control when the outputting happens. Whenever the mesh is changed, for example of with mesh adaptation, a new file will be created with name being <name>_<counter>.vtk, where <counter> counts the number of times that the mesh has been changed starting from 1.
10.10.5.1  **var**

Description: Variables to be outputted
Data type: Vector of string
Default value: `<required>`
Syntax: UserObjects/*\*/var
Note: Any nodal or elemental field variable can be put in this list.

10.10.5.2  **block**

Description: Outputting on selected blocks
Data type: Vector of block names
Default value: `<empty>`
Syntax: UserObjects/*\*/block
Note: Empty means variables will be outputted on all blocks in the mesh.

10.10.5.3  **cut**

Description: Number of cuts for all elements
Data type: Integer
Default value: 1
Syntax: UserObjects/*\*/cut
Note: Increasing this number can increase the resolution of the visualization. But keep in mind that the amount of data in the output file is increased by about a factor $cut^2$ in 2D and $cut^3$ in 3D. Default means no cut to the elements.

10.10.5.4  **execute_on**

Refer to execute_on in Postprocessors.

10.10.6  **SolutionVectorFile**

This user object writes primal or auxiliary variables into a binary file. The binary file can be read by this user object to set primal or auxiliary variables later. When writing, this user object creates a file named `<name>.bin`, where `<name>` is the name of this user object. This user object is particularly useful for restarting a calculation with the solution of previous calculations with identical meshes and shape functions. It provides a simple way of loading variables without relying on Exodus file readers or the MOOSE restarting system. Users can use execute_on to control when the writing or reading happens. Typically the writing happens at final and the reading at initial. It is noted that the reading happens before initial conditions are applied when the reading is on initial. This means that when both reading on initial and an initial condition are on the same variables, the initial condition will win. This user object works with parallelization. However, the binary file must be generated with the same number of processors that are used during reading.
10.10.6.1  \textit{var}

Description: Variables to be written to or read from a binary file  
Data type: Vector of strings  
Default value: <required>  
Syntax: UserObjects/*/var  

Note: Any primal/auxiliary nodal/elemental field variable can be put in this list.

10.10.6.2  \textit{writing}

Description: True for writing, false for reading  
Data type: Logical  
Default value: True  
Syntax: UserObjects/*/writing

10.10.6.3  \textit{loading}\_\textit{var}

Description: Variables to be stored when reading  
Data type: Vector of strings  
Default value: <empty>  
Syntax: UserObjects/*/loading\_\textit{var}

Note: If not provided, the same variable names as in \textit{var} will be used as the names of primal/auxiliary variables. It is only active when \textit{writing} is false.

10.10.6.4  \textit{execute}\_\textit{on}

Refer to \textit{execute}\_\textit{on} in Postprocessors.

10.10.7  TransportSolutionVectorFile

This user object writes primal variables of a transport system into a binary file. The binary file can be read by this user object to set primal variables of a transport system later. The read-in variables can be also used to set adjoint auxiliary variables for IQS calculations. \textit{k-effective} is automatically written/read by this user object when doing neutron transport calculations. When loading for primal variables, users can control if the loaded \textit{k-effective} is to be pushed into the transport system and if the \textit{k-effective} is going to be used to scale the primal variables. When writing, this user object creates a file named \textit{<name>}.bin, where \textit{<name>} is the name of this user object. This user object is particularly useful for restarting a calculation with the solution of previous calculations with identical meshes and shape functions. It provides a simple way of loading variables without relying on Exodus file readers or the MOOSE restarting system. Users can use \textit{execute}\_\textit{on} to control when the writing or reading happens. Typically the writing happens at \textit{final} and the reading at \textit{initial}. It is noted that the reading happens before initial conditions are applied when the reading is on \textit{initial}. This means that when both reading on initial and an initial condition apply to the same variables, the initial condition will take precedence. This user object works with parallelization. However, the binary file must be generated with the same number of processors that are used during reading.
10.10.7.1  transport_system

Description: The transport system name
Data type: String
Default value: <required>
Syntax: UserObjects/*/transport_system

10.10.7.2  writing

Description: True for writing, false for reading
Data type: Logical
Default value: True
Syntax: UserObjects/*/writing

10.10.7.3  load_to_adjoint

Description: True to load variables in the file to adjoint variables for IQS
Data type: Logical
Default value: False
Syntax: UserObjects/*/load_to_adjoint
Note: It is only active when writing is false.

10.10.7.4  execute_on

Refer to execute_on in Postprocessors.

10.10.7.5  scale_with_keff

Description: True to divide solutions with keff
Data type: Logical
Default value: True
Syntax: UserObjects/*/scale_with_keff
Note: It is only active when writing is false and load_to_adjoint is false.

10.10.7.6  disable_eigenvalue_transfer

Description: Whether or not the eigenvalue is not to be pushed into the transport system
Data type: Logical
Default value: False
Syntax: UserObjects/*/disable_eigenvalue_transfer
Note: It is only active when writing is false and load_to_adjoint is false.
11 Auxiliary variables and Kernels

Auxiliary variables can be added in the input to store field variables. Auxiliary kernels are used to evaluate the auxiliary variables. Primal variables, material properties, postprocessors, etc. can be coupled in auxiliary kernels. Auxiliary variables can be coupled back to other MOOSE objects. One sample usage of auxiliary variable is the fission source, which is evaluated from scalar fluxes and the nu fission cross sections and is used by fission kernels. MOOSE auxiliary kernels can be found in MOOSE documents. A more exhaustive list of auxiliary kernels in Rattlesnake will be added in the future.

11.1 Common Parameters for Auxiliary Kernels

All auxiliary kernels share a few common parameters provided by MOOSE, which are documented below:

11.1.1 variable

Description: Name of the auxiliary variable the auxiliary kernel operates on
Data type: String
Default value: <required>
Syntax: AuxKernels/*/variable

11.1.2 execute_on

Description: To determine when the object is evaluated
Data type: enumeration (/initial/linear/nonlinear/timestep_begin/timestep_end/custom/)
Default value: linear
Syntax: AuxKernels/*/execute_on
Note: This parameter accepts multiple enumerations. Its default value is different from the default value of post-processors in Sec. 10.1.

11.1.3 use_displaced_mesh

Description: Whether or not this object should use the displaced mesh for computation
Data type: Logical
Default value: false
Syntax: AuxKernels/*/use_displaced_mesh
The complete list of MOOSE auxiliary kernels can be found in Fig. 32.
11.3.1 *CopyAux*

This auxiliary kernel is used to assign one variable to another with possibly different variable types.
(Parameters are to be added.)

11.3.2 *CoupledAux*

This auxiliary kernel does simple arithmetic operation with a source variable.
(Parameters are to be added.)

11.3.3 *DivergenceAux*

This auxiliary kernel evaluates the divergence of a vector field variable.
(Parameters are to be added.)

11.3.4 *ElemIDAux*

This auxiliary kernel is designed to help visualizing various IDs from the mesh (element, subdomain, material, depletion or equivalence ID).

11.3.4.1 *variable*

Refer to *variable* in Common Parameters for Auxiliary Kernels.

11.3.4.2 *execute_on*

Refer to *execute_on* in Common Parameters for Auxiliary Kernels.

11.3.4.3 *use_displaced_mesh*

Refer to *use_displaced_mesh* in Common Parameters for Auxiliary Kernels.

11.3.4.4 *id_type*

Description: The type of ID from the mesh requested by this Auxkernel

Data type: Enumeration (/elem_id/subdomain_id/material_id/depletion_id/equivalence_id/)

Default value: elem_id

Syntax: AuxKernels/*/id_type

Note: If the material, depletion or equivalence IDs are requested, the mesh must be either a GeneratedIDMesh or a FileIDMesh.
11.3.5 MaterialCSRAux

This auxiliary kernel converts one element of a material property in type of a vector of a sparse matrix in CSR (compressed sparse row) format into an auxiliary variable.

(Parameters are to be added.)

11.3.6 NonlinearAdiabaticHeatAux

This auxiliary kernel performs nonlinear solvers locally on quadrature points. The variable to be evaluated is typically a temperature.

(Parameters are to be added.)

11.3.7 NonlinearAdiabaticHeatIQS

This auxiliary kernel performs nonlinear solvers locally on quadrature points with IQS executioner, refer to sectIQS. The variable to be evaluated is typically a temperature.

(Parameters are to be added.)

11.3.8 OrthogonalizationAux

This auxiliary kernel evaluates the orthogonal components of one variable with respect to another variable. Inner products of the two variables with the adjoint of the other variable have been pre-computed into postprocessors, which are passed into this auxiliary kernel.

(Parameters are to be added.)

11.3.9 ReactionRate

This auxiliary kernel evaluates the reaction rates with scalar fluxes and cross sections.

(Parameters are to be added.)

11.3.10 VectorReactionRate

This auxiliary kernel evaluates the reaction rates with scalar fluxes and cross sections. It differs from ReactionRate in that the cross sections are in type of a vector material property instead of a vector of material properties.

(Parameters are to be added.)

11.3.11 SelfSubtractionAux

This auxiliary kernel evaluates the subtraction of two variables and stores the results into the first variable.

(Parameters are to be added.)

11.3.12 SubtractionAux

This auxiliary kernel evaluates the subtraction of two variables and stores the results into a separate variable.

(Parameters are to be added.)
11.3.13  **ShowExtraMeshInfo**

This auxiliary kernel supports `VariableCartesianCoreMap` to reveal additional mesh information.
(Parameters are to be added.)

11.3.14  **SweepOrderAux**

This auxiliary kernel shows the sweeping order of a variable.
(Parameters are to be added.)

11.3.15  **VariableTimeIntegrationIQS**

This auxiliary kernel performs time integration with IQS executioner, refer to sectIQS.
(Parameters are to be added.)

11.3.16  **WeightedVariableSum**

This auxiliary kernel performs weighted summation of a list of variables.
(Parameters are to be added.)

11.3.17  **FluxWithoutSPHFactor**

This auxiliary kernel removes the multiplicative SPH (super-homogenization) factors from a flux of a certain energy group.
(Parameters are to be added.)
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 Other 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. Additional variables, kernels, auxiliary variables, auxkernels, materials, user objects, multiapps, transfers, initial conditions etc. can be added. Rattlesnake provides some extra objects on top of MOOSE objects. 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 can be done either strongly (solving as a whole) or tightly (with Picard iterations).

13.1 Rattlesnake Kernels

13.1.1 SplitDGScalarDiffusion

This kernel is to be used for discretizing non-neutronics physics, for example of thermal conduction equation, with DFEM diffusion. It is supposed to be used together with MOOSE Diffusion kernel. This kernel allows the connectivity of the diffusion process to be broken on given internal side sets, where convection can be added.

13.1.1.1 variable

Description: The primal variable this kernel operates on
Data type: String
Default value: <required>
Syntax: Kernels/*/variable

13.1.1.2 block

Description: The block this kernel is restricted to
Data type: Vector of strings
Default value: <empty>
Syntax: Kernels/*/block
Note: Empty means that the kernel is defined on the entire mesh.

13.1.1.3 DGType

Description: DG diffusion type
Data type: Enumeration (/SIP/NIP/IIP/)
Default value: SIP
Syntax: Kernels/*/DGType
Note: SIP, NIP and IIP stands for symmetric, non-symmetric and incomplete interior penalty respectively.
13.1.1.4  $D$

Description: Name of the diffusion coefficient material property
Data type: String
Default value: diffusion_coefficient
Syntax: Kernels/*/D
Note: Diffusion coefficient must be available on the blocks that this kernel is defined on. Diffusion coefficient must be a scalar other than a vector or a tensor.

13.1.1.5  averaged_normal

Description: Name of averaged normal material property
Data type: String
Default value: averaged_normal
Syntax: Kernels/*/averaged_normal
Note: Averaged normal is typically defined on the entire mesh with a material without block or boundary restriction, thus made available on all internal sides. It is used for evaluating the penalty coefficients when the mesh is irregular.

13.1.1.6  $\sigma$

Description: Diffusion penalty factor
Data type: Real
Default value: 4
Syntax: Kernels/*/sigma
Note: This number must be greater than or equal to 1. It is a factor applied on evaluating the penalty coefficient for interior penalty methods. Larger value can ensure the positive definiteness but also result in higher condition number of the Jacobian.

13.1.1.7  broken_boundary

Description: The internal side sets to be broken
Data type: Vector of strings
Default value: <required>
Syntax: Kernels/*/broken_boundary

13.1.1.8  convection_coefficient

Description: The convection coefficient on broken boundary
Data type: Real
Default value: 0
Syntax: Kernels/*/convection_coefficient

Note: Combining bulk_variable, it provides a boundary condition on the broken boundary for both sides with

\[- D \nabla u \cdot \vec{n} = h(u - u_b), \tag{65}\]

where $D$ is the diffusion coefficient in $D$; $u$ is the primal variable this kernel is for; $\vec{n}$ is the outward unit normal on the broken boundary for either side; $h$ is the convection coefficient specified by this parameter; $u_b$ is the bulk variable, bulk_variable, applied to the convection. 0 means no sink on the broken boundary.

13.1.1.9 bulk_variable

Description: The bulk variable applied to the convection

Data type: String

Default value: 0

Syntax: Kernels/*/bulk_variable

Note: Refer to the note in convection_coefficient.

13.2 Rattlesnake Initial Conditions

(To be added.)

13.3 Rattlesnake Markers

(To be added.)

13.4 Rattlesnake Materials

(To be added.)

13.5 Rattlesnake MultiApps
13.5.1 *QuasiStaticSolveMultiApp*

This MultiApp differs from the MOOSE FullSolveMultiApp in that it allows the problem to be resolved when the solution of the multiapp changes due to changes in mesh, material properties, etc. For example, it can be used to wrap an eigenvalue problem during master transient calculations to model slow cross section changes at different time steps.

13.5.2 *RayTracerMultiApp*

This MultiApp derives from the MOOSE FullSolveMultiApp. It is used for importing the uncollided flux solution for the purpose of first collision source treatment. It checks that the correct uncollided flux variables are defined such that they can be properly imported as a scattering source term.

### 13.6 Rattlesnake Transfers

13.6.1 *TransportSystemVariableTransfer*

This Transfer allows to transfer the transport system solution from a MultiApp calculation to the solution on the master with identical solution degrees of freedoms. Thus the mesh from the source and target transport systems must be identical. It also scales the solution as requested and transfers the desired eigenvalue to the target transport system. One frequent application is when the MultiApp is an eigenvalue calculation used to set the initial condition of a transient on the master. The initial condition would constitute a null-transient if the material properties of the target transport system were identical to those of the source.

**13.6.1.1 from_transport_system**

Description: The source transport system name  
Data type: String  
Default value: `<required>`  
Syntax: Transfers/*/from_transport_system

**13.6.1.2 to_transport_system**

Description: The target transport system name  
Data type: String  
Default value: `<required>`  
Syntax: Transfers/*/to_transport_system
13.6.1.3 scale_with_keff

Description: Whether to divide the solution vectors with the computed multiplication factor.
Data type: Logical
Default value: true
Syntax: Transfers/*/scale_with_keff

13.6.1.4 disable_eigenvalue_transfer

Description: True if the eigenvalue transfer is to be disabled.
Data type: Logical
Default value: false
Syntax: Transfers/*/disable_eigenvalue_transfer
Note: In some instances, it is desired to transfer the reference eigenvalue rather than the computed eigenvalue from
the source transport system (e.g. when computing discontinuity or SPH factors). Providing the reference eigenvalue
through scaling_eigenkernels and setting disable_eigenvalue_transfer to true will do just that.

13.6.1.5 beta_effective

Description: Beta effective (a dollar) in pcm
Data type: Positive real
Default value: 600
Syntax: Transfers/*/beta_effective
Note: This parameter is only used if the source transport system is an eigenvalue calculation and if disable_eigenvalue_transfer
is false.

13.6.1.6 reactivity_insertion

Description: Reactivity in dollar to be inserted by modifying the multiplication factor to be transferred
Data type: Real
Default value: 0
Syntax: Transfers/*/reactivity_insertion
Note: This parameter is only used if the source transport system is an eigenvalue calculation and if disable_eigenvalue_transfer
is false.

13.6.2 MultiAppAdjointFluxTransfer

This Transfer transfers primal variables of a transport system to adjoint variables of a target transport system for
IQS.

13.6.2.1 from_transport_system

Refer to from_transport_system in TransportSystemVariableTransfer.
13.6.2.2  to_transport_system

Refer to to_transport_system in TransportSystemVariableTransfer.

13.6.3  EigenvalueTransfer

This Transfer transfers an eigenvalue postprocessor to/from a MultiApp.

13.6.4  MultiAppNumericVectorTransfer

This Transfer transfers a numerical vector to/from a MultiApp. The MultiApp must be using the same mesh as the master’s.

13.6.5  MultiAppVariableTransfer

This Transfer allows transferring a set of variables to/from a MultiApp. The MultiApp must be using the same mesh as the master’s.
References


