Methodology for spatial homogenization in Serpent 2

Jaakko Leppänen

Memo 2014/05/26

1 Background

Spatial homogenization has been one of the main motivations for developing Serpent since the beginning of the project in 2004. This task has taken much longer, and proven considerably more complicated than originally expected. After ten years, the methodology is finally coming together, but the result of the long development period with a trial and error type of approach has lead to a situation where the output consists of a number of seemingly identical parameters, some of which are based on less than solid theoretical foundation. The purpose of this memo is to clarify the methods available in Serpent 2, and introduce the features currently under development.[1]

The goal set for 2014 is to have the methodology needed for generating homogenized group constants for nodal diffusion codes ready for routine calculations by the end of the year. This methodology covers all parameters needed for steady-state calculations performed using fuel cycle simulator codes, as well as those required for transient calculations with dynamic codes. A major part of this work is an automated burnup sequence, capable of performing branch calculations according to a user-specified coefficient matrix. Since this capability affects the way the calculations are run, it is important to discuss the best practices with developers of utility and processing codes, such as SerpentXS, STOP, PyNE and GenPMAXS.

2 Methods used for group constant generation

The methodology used for group constant generation in Serpent 2 is based on a two-stage approach:

i) A number of multi-group homogenized reaction cross sections are calculated using standard Monte Carlo tallies and a few analog estimators. By default this is done using

[1] It should be noted that the methods described here are implemented only in Serpent 2, as the work on Serpent 1 is now limited to bug fixes and minor modifications.

the WIMS 69-group structure, which can be changed using the "set micro" or "set fum" cards.

ii) The multi-group cross sections are condensed into few-group cross sections using the infinite and the B_1-leakage corrected critical spectrum. By default this is done using a two-group structure (changed using the "set nfg" card), and the B_1 method is optional.

The reason for this two-stage approach is somewhat technical – the calculation involves a large number of parameters, and using the same calculation routine for both infinite-spectrum and leakage-corrected cross sections simplifies the coding. The only difference between the two methods is that in the first case the flux spectrum used for condensing the multi-group cross sections is obtained directly from the Monte Carlo calculation, and in the second case from the solution of the B_1 equations [1].

2.1 Diffusion coefficients

Whether the cross sections are homogenized using the infinite- or leakage-corrected spectrum also affects the calculation of the diffusion coefficient. In infinite spectrum, the value is obtained from the transport cross section [2]:

$$\Sigma_{\mathrm{tr},g} = \frac{\sum_{h\in g} \Phi_h}{\sum_{h\in g} \frac{\Phi_h}{\Sigma_{\mathrm{tot},h} - \overline{\mu}_h \Sigma_{\mathrm{s},h}}} \;\Rightarrow\; D_g = \frac{1}{3\Sigma_{\mathrm{tr},g}}\,, \tag{1}$$

where index h refers to the intermediate multi-group structure and g to the final few-group structure, Σ_{tot} and Σ_{s} are the total and scattering cross sections obtained using standard Monte Carlo tallies and $\overline{\mu}$ is the average scattering cosine obtained using an analog estimator (average over the scattering reactions in the corresponding energy group). The leakage-corrected diffusion coefficient is obtained from the B_1 calculation:

$$D_g = \frac{\sum_{h\in g} J'_h}{|B| \sum_{h\in g} \Phi'_h}\,, \tag{2}$$

where B is the critical buckling, and J'_h and Φ'_h are the corresponding current and flux spectra, respectively.

The infinite-spectrum and leakage-corrected group constants are named INF_XXX and B1_XXX in the _res.m output file. Code versions up to 2.1.17 (Feb. 24, 2014) also include other group constants, generated using methods adopted from Serpent 1. These parameters were removed from the more recent updates, mainly because the methods used for calculating the infinite-spectrum diffusion coefficient are fundamentally wrong (or at least inconsistent with the methods used in deterministic lattice codes), and any processing codes and scripts should be modified to use the new parameters instead.

2.2 Time constants

Serpent 2 calculates various time constants by default, and the calculation of adjoint-weighted point-kinetics parameters and delayed neutron fractions was recently revised by adopting the iterated fission probability method (IFP) [3]. This was done mainly for the purpose of research reactor modeling, and even if the results could be used as input for transient simulator calculations, the methodology is limited by the fact that the time constants are averaged over the entire geometry. This is not always the intention in homogenization, for example, when the calculation is performed in an assembly colorset configuration.

Instead of using the IFP method, we are planning to implement an entirely separate solution for reactor time constants for the purpose of group constant generation.

2.3 Assembly discontinuity factors

Even though group constant generation is traditionally carried out using a 2D model of the homogenized fuel assembly surrounded by reflective boundary conditions, there are several situations where this approach does not work. The most typical example is a reflector node, which requires a source of neutrons outside the homogenized region. Modeling the surroundings can also become important when the assembly is located at the core-reflector boundary (see Ref. [4]), or other region where the flux gradients are steep.

The geometry model used for homogenization affects the calculation of assembly discontinuity factors (ADF's). By definition the ADF for a given node boundary is calculated as the ratio of the local heterogeneous to the homogeneous integral flux:

$$F_g = \frac{\frac{1}{S}\int_S d^2r \int_{E_g}^{E_{g-1}} dE\phi(E,\mathbf{r})}{\frac{1}{S}\int_S d^2r \hat{\phi}_g(\mathbf{r})}\,, \tag{3}$$

where the integrations are carried over the boundary surface. When the homogenized region is surrounded by reflective boundary conditions, the net currents over the boundary surfaces are reduced to zero. The solution of diffusion equation in the corresponding homogeneous system then takes a constant form, and from the preservation of the volume-integrated reaction rates it results that this constant is equal to the average heterogeneous flux over the homogenized region. This, in turn, implies that the assembly discontinuity factors can be calculated using the heterogeneous flux solution alone, as:

$$F_g = \frac{\frac{1}{S}\int_S d^2r \int_{E_g}^{E_{g-1}} dE\phi(E,\mathbf{r})}{\frac{1}{V}\int_V d^3r \int_{E_g}^{E_{g-1}} dE\phi(E,\mathbf{r})}\,. \tag{4}$$

When the "set adf" card is defined, Serpent 2 calculates the integrals in Eq. (4) for the

given boundary using standard surface flux tallies, and prints the ADF's in the _res.m output (variable DF_SURF_DF for node boundary surfaces and variable DF_CORN_DF for corners).

If the boundary of the homogenized region does not coincide with the reflective boundary of the geometry, the net currents become non-zero, and Eq. (4) no longer holds. Instead, the ADF's must be calculated from Eq. (3), using an explicit solution for the homogeneous diffusion flux $\hat{\phi}_g$. In Ref. [4] this was done using a Matlab script that reads the homogenized group constants and net currents from the Serpent output[2]. The same homogeneous diffusion flux is also needed for calculating form functions for pin-power reconstruction.

The diffusion flux solver will be implemented in Serpent 2 as an internal subroutine. In order to get the best values for the nodal diffusion calculation, the solution should be obtained as it is done in the nodal code for which the group constants are generated, most importantly, using similar coupling at the node boundaries. At the first stage the methods will be limited to rectangular and hexagonal 2D geometries. Since our own experience is limited to codes developed at VTT, it is important to get feedback from the users to make sure that the developed methodology is sufficient for their applications.

3 Automated burnup sequence

Generating group constants for a fuel cycle simulator calculation requires covering all operating conditions within the reactor core throughout the simulated cycle, such as fuel temperature, coolant void fraction (BWR) and the concentration of soluble absorber (PWR). The group constant data is parametrized according to discrete state-points, from which the core simulator interpolates the values corresponding to burnup and thermal hydraulic state. Since the local operating conditions inside a fuel assembly also affect how the assembly is depleted, the state-points by which the group constant data is parametrized are not completely independent. The calculations are instead divided into:

i) History calculations, i.e. burnup calculations representing a single combination of state variables, which takes into account operating conditions that persist for an extended period of time and affect how the fuel is depleted.

ii) Branch or calculations, which account for momentary changes in the operating conditions, for example, insertion of control rods or increase or decrease in fuel temperature.

In practice, the process of group constant generation involves running assembly burnup calculations covering all assembly types and history variables. Branch variables are accounted for by performing restart calculations for each burnup point, varying the local operating conditions accordingly. The procedure is illustrated by a simplified example in Figure 1. The result is that the Monte Carlo transport simulation is performed hundreds or even thousands

[2]The net and partial currents are printed in the _res.m output when the "set adf" card is defined

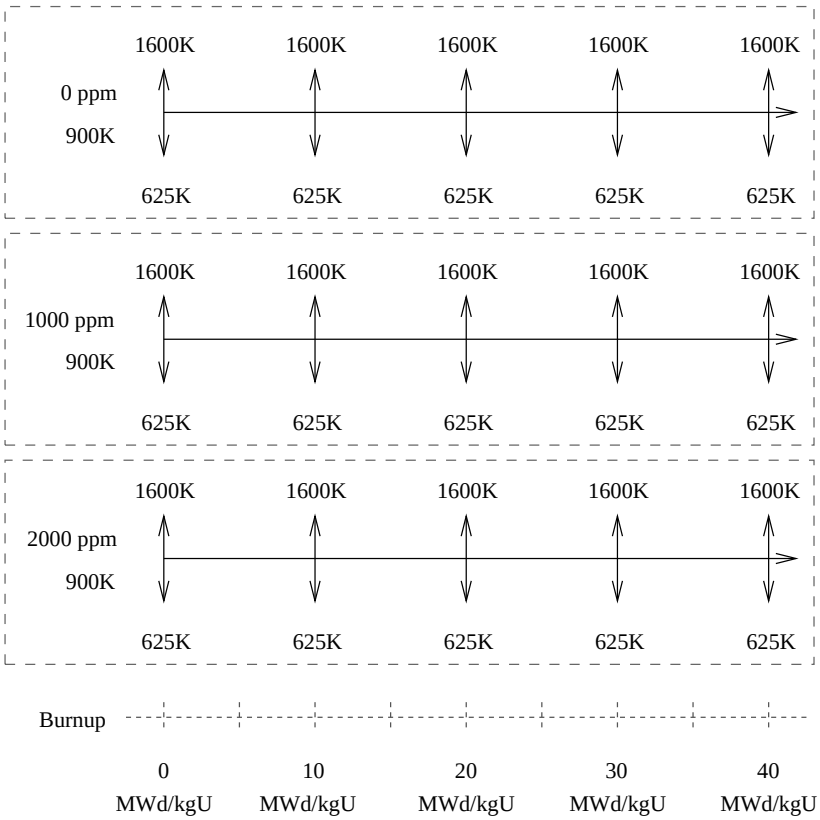

Figure 1: Illustration of boron history and fuel temperature branch calculations. Three burnup calculations are run with different coolant boron concentration. Branch calculations are run at 0, 10, 20, 30 and 40 MWd/kgU burnup, varying fuel temperature from the nominal value.

of times. Performing the calculations in the most efficient way, applying the correct changes in the input models and managing the output become major challenges for the code user, and in practice this task requires using automated driver and processing scripts.

To simplify the process, some of the effort is taken from the user and scripts by introducing an automated burnup sequence capable of performing the branch calculations automatically, based on variations defined in the Serpent input. The work was started in early 2014, and the methodology is made available in update 2.1.21. The calculation sequence is based on a new "branch" card, that can be used to perform the following variations in the input model:

stp – Change the temperature and density of a material

repm – Replace an entire material with another material

repu – Replace an entire universe with another universe

The number of variations in a single branch card is unlimited, as is the total number of branch cards. These variations can be used for things like:

- Changing the fuel temperature (invoked by reloading the cross section libraries at a different temperature)
- Changing the coolant void fraction or boron concentration (invoked by replacing one coolant material with another)
- Insertion of a control rod (invoked by replacing empty guide tubes with rodded tubes)

When the calculation is run, the code first performs the burnup calculation from beginning to end as it is defined in the input file (nominal state), without invoking any branches. For each burnup step the compositions of burnable materials are stored in a binary work file. After the burnup calculation is completed, the code starts to execute the branches, as they are listed in a coefficient matrix. This matrix, defined using a new "coef" card, lists the burnup points for which the group constants are requested, followed by the combination of branches. Before each transport calculation is executed, the corresponding changes are made in the input model and the compositions of burnable materials retrieved from the work file.

The input example in Figure 2 illustrates the procedure. Six branches are defined: one for the nominal state, two for elevated and reduced fuel temperature, two for increased and decreased coolant boron concentration and one for the insertion of control rods. These branches are used to define a $3\times3\times2$ coefficient matrix. Group constants are requested for 13 burnup points, which results in a total of 234 restart calculations. The group constant data is written in a separate output file in a format easily read by post-processing scripts for the construction of data libraries for the nodal code.

The advantage of using the automated burnup sequence for group constant generation is that the changes in the model can be made without editing the input file during the calculation. All variations, such as material definitions corresponding to different thermal hydraulic states, can be done beforehand, in the same input files describing the nominal state. What is left for the driver script is the execution of history calculations, which is done from beginning to end without interruptions. Another major advantage is that all burnup points for a single branch are run consecutively, without reloading the cross sections before each calculation.[3] This removes the unnecessary computational overhead and reduces overall calculation time.

[3]Before the new methodology, the execution of branch calculations could be done by writing the material compositions from a burnup calculation into new Serpent-format material cards using the "set printm 1" option, and constructing the inputs for branch calculations manually or using an automated script. Since the branch cases are executed as independente runs, the cross sections are loaded separately for each burnup point, even though the data remains the same.

```
% --- Nominal state branch (do nothing):

branch nom

% --- Fuel temperature branches (change in material temperature):

branch fueH stp fuel24 sum 1600
branch fueC stp fuel24 sum 625

% --- Coolant boron concentration branches (replace material):

branch borL  repm cool cool_loB
branch borH  repm cool cool_hiB

% --- Control rod insertion branch (replace universe):

branch CR  repu T R

% --- Coefficient matrix (13 burnup points, 3x3x2 branch combinations):

coef 13
0 5 10 15 20 25 30 35 40 45 50 55 60
3 nom fueH fueC
3 nom borL borH
2 nom CR
```

Figure 2: Serpent 2 input example of a coefficient calculation with fuel temperature, coolant boron concentration and control rod insertion branches. The new “branch” and “coef” cards are available in version 2.1.21.

4 Discussion

In order to make the most out of the new features and capabilities and to ensure that all necessary parameters are calculated, it is necessary to get feedback from Serpent users and the developers of processing and utility codes. New suggestions are welcome, and there are several specific topics with open questions:

1. Are all parameters needed by (steady-state) fuel cycle simulator codes calculated by Serpent 2, and is the methodology consistent with what is used in the nodal diffusion calculation? As mentioned in the text, the code is still missing time constants for transient calculations. In addition, the calculation of fission product poison cross sections needs to checked and possibly revised. We are also planning to add albedos and partial albedos in the output.

2. Is the methodology planned for the calculation of assembly discontinuity factors sufficient for all nodal diffusion codes? Do we need additional options for the deterministic diffusion flux solver?

3. What are the time constants needed for dynamic calculations? Are the parameters the same for all transient simulator codes?

4. Is the methodology planned for the automated burnup sequence with branch capability

sufficient for covering all state points? Do we need additional options for varying the operating conditions?

5. The _res.m output file may not be the optimal way to pass group constant data to processing codes. The automated burnup sequence will produce a separate output in a format that is easily read by post-processors. This format has not yet been decided, and the details should be discussed with the developers of processing codes.

At this point the goal is to complete the "conventional" methodology for spatial homogenization by the end of 2014, meaning that Serpent 2 should be able to perform the same tasks as currently-used deterministic lattice transport codes. It is equally important to provide a sufficient set of additional tools (tallies, etc.) to enable group constant generation for more advanced applications, for example, 3D homogenization with axial discontinuity factors, even though such options will not yet be included in the built-in routines.

References

[1] E. Fridman and J. Leppänen, *"On the use of the Serpent Monte Carlo code for few-group cross section generation."* Ann. Nucl. Energy, **38** (2011) 1399-1405.

[2] E. Fridman, J. Leppänen, and C. Wemple, *"Comparison of Serpent and HELIOS-2 as applied for the PWR few-group cross section generation."* In proc. M&C 2013, Sun Valley, ID, May 5-9, 2013.

[3] J. Leppänen, M. Aufiero, E. Fridman, R. Rachamin and S. van der Marck, *"Calculation of effective point kinetics parameters in the Serpent 2 Monte Carlo code."* Ann. Nucl. Energy, **65** (2014) 272-279.

[4] J. Leppänen, R. Mattila and M. Pusa, *"Validation of the Serpent-ARES code sequence using the MIT BEAVRS benchmark - Initial core at HZP conditions."* Ann. Nucl. Energy, **69** (2014) 212-225.