Kraken workshop
Group constant generation

Ville Valtavirta

04/2024 VTT – beyond the obvious

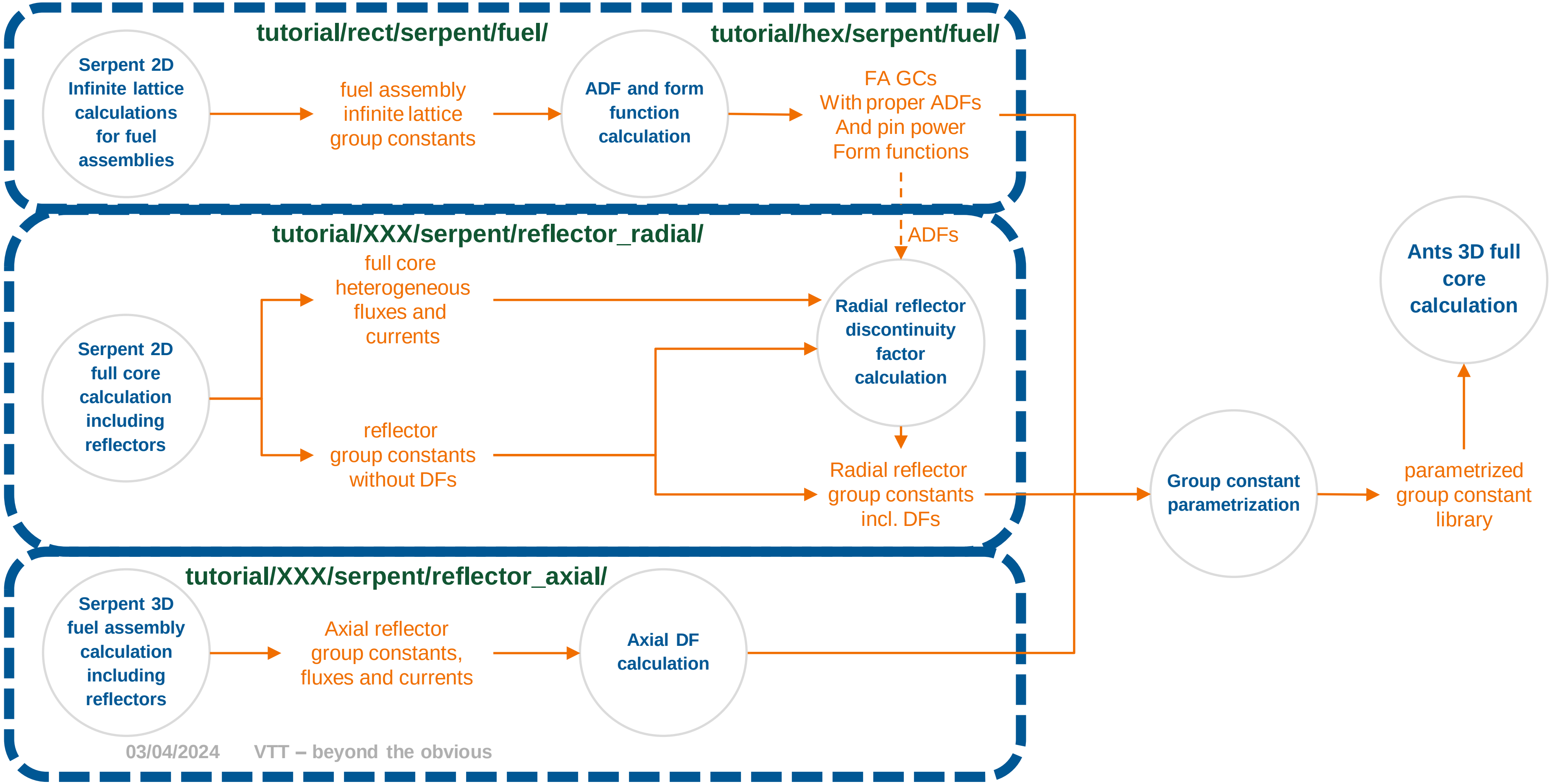
Contents

- Group constant generation with Serpent for Ants.
- Current work on VVER benchmarks.
- Summary and next steps

VTT

Using Serpent to generate group constants for Ants in the Kraken framework

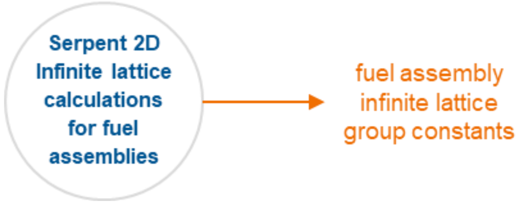
Group constant generation

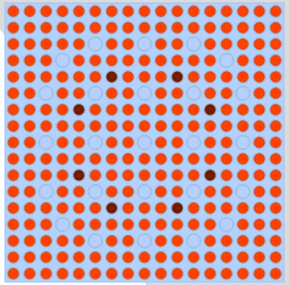


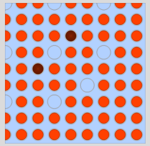
tutorial/rect/serpent/fuel/
tutorial/hex/serpent/fuel/
Serpent 2D Infinite lattice calculations for fuel assemblies
fuel assembly infinite lattice group constants
ADF and form function calculation
FA GCs With proper ADFs And pin power Form functions
tutorial/XXX/serpent/reflector_radial/
ADFs
full core heterogeneous fluxes and currents
Serpent 2D full core calculation including reflectors
Radial reflector discontinuity factor calculation
reflector group constants without DFs
Radial reflector group constants incl. DFs
Ants 3D full core calculation
Group constant parametrization
parametrized group constant library
tutorial/XXX/serpent/reflector_axial/
Serpent 3D fuel assembly calculation including reflectors
Axial reflector group constants, fluxes and currents
Axial DF calculation

Best practices calculation chain fuel GCs

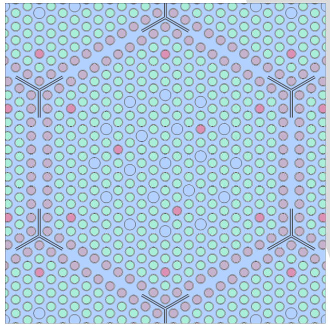
tutorial/???/serpent/fuel/
tutorial/???/serpent/includes/


Serpent 2D Infinite lattice calculations for fuel assemblies
fuel assembly infinite lattice group constants





- Full assembly in infinite lattice.
- Depletion calculations with nominal and off-nominal conditions.
- Branch calculations with momentary variations:
  - Different (T_{fuel} , T_{cool} , ρ_{cool} , C_B) variations.
  - Control rod variations.
  - Spacer grid variations.
  - Instrument tube variations.
- Can use an intermediate multigroup structure and apply leakage correction / critical spectrum in condensation to a few group structure.
- Typically produce CMM[7] diffusion coefficients.



[7] Z. Liu et al. “Cumulative migration method for computing rigorous diffusion coefficients and transport cross sections from Monte Carlo”. *Annals of Nuclear Energy*, 112 (2018), pp. 507–516.

Practical things about fuel GCs

- Full assembly in infinite lattice (set bc) (input example)
  - ADF setup
  - Pin power setup
  - Poison constants, microdepletion setup.
- Depletion calculations with nominal and off-nominal conditions.
- Branch calculations with momentary variations:
  - Different (T_{fuel} , T_{cool} , ρ_{cool} , C_B) variations.
  - Control rod variations.
  - Spacer grid variations.
  - Instrument tube variations.
- Can use an intermediate multigroup structure and apply leakage correction / critical spectrum in condensation to a few group structure.
- Typically produce CMM[7] or transport corrected diffusion coefficients.

```
set cmm 1
set trc cool "s2v0_endfb71.h_in_h2o.trcdata" 1.000000E-11 10010
```

Use of:
branch-card
casematrix-card

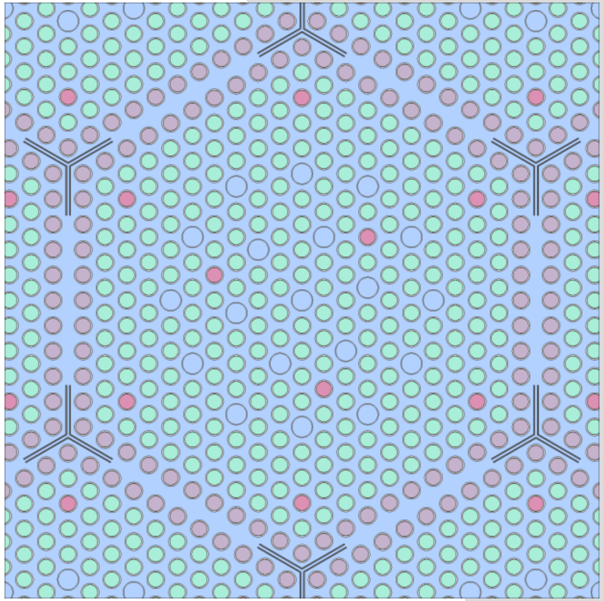
Running Serpent from command line

his, coe, ln -s

```
set fum cas70_ext 2 f 3

set micro cas70_ext

set nfg cas8_ext

set repro 0

set shbuf 0 0
```

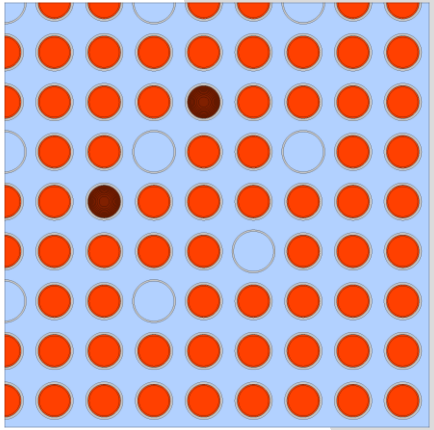
Setting up ADF and pin power evaluation

```
% --- Include all of the common data related to hexagonal models

include "../includes/constants.inc"
include "../includes/tutorial_assemblies.inc"
include "../includes/powdens_FP.inc"
include "../includes/A1_good.mvol"

% --- Outer boundary of geometry

surf sOuter hexxc 0 0 11.8

% --- Construct infinite lattice 2D model

cell c1 0 fill A1s -sOuter sAngle
cell c2 0 E635     -sAngle
cell c3 0 outside   sOuter

% --- Evaluate pin power form functions

set ppw 0 lA1s

% --- Geometry plots

plot 1 2000 2000  0 -15 15   0 30
plot 3 2000 2000 50 -15 15 -15 15
```



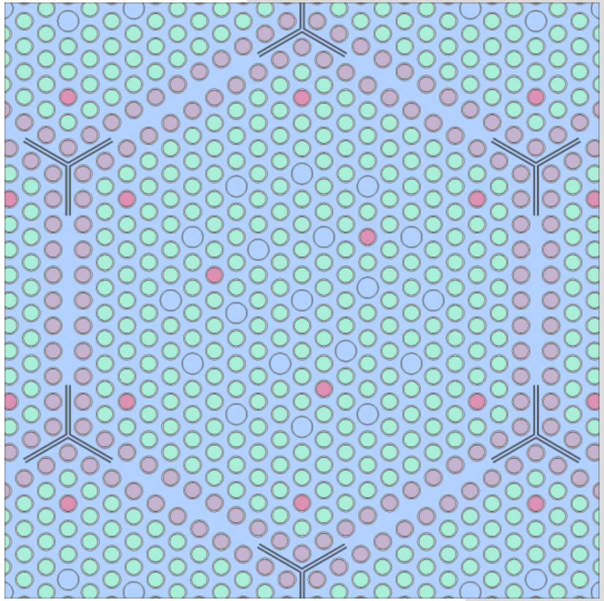
Setting up ADF and pin power evaluation

```
% --- Include all constant data

include "../includes/constants.inc"
include "../includes/assemblies.inc"
include "../includes/A1_good.mvol"

% --- Set up the calculation geometry (SE corner of fuel assembly)

surf sBound2D sqc  5.37591 -5.37591  5.37591

% --- Fill a symmetric version of the A1 lattice

cell c1 0 fill uA1s   -sBound2D
cell c2 0 outside      sBound2D

% --- Pin power form function calculation

set ppw 0 lA1s

% --- Geometry plot

plot 33 1000 1000 0.1 -12 12 -12 12
```



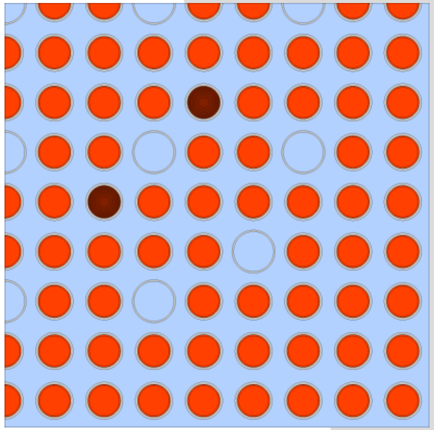
Poison constants and microdepletion data

```
% --- Fission poison data
%     system total area 482.341509 cm2

set poi 1 482.341509

% --- Microdepletion data for the plutonium chain
% set mdep UNI VOL N MAT1 MAT2 ... MATN
%          ZAI1 MT1
%          ZAI2 MT2
%          ...

set mdep 0 482.341509 0
922380 16 922380 18 922380 102
932390 16 932390 18 932390 102
942390 16 942390 18 942390 102
```



Using `branch` and `casematrix` to set up history and branch calculations

```
branch hi_tfu
stp F1400 original 800.0
stp F1500 original 800.0
stp F1800 original 800.0
stp F2400 original 800.0
stp F1506 original 800.0
stp F1806 original 800.0
stp F2405 original 800.0
stp F2409 original 800.0
stp EBOC original 404.15
stp INC original 404.15
stp air original 404.15
stp zirc original 404.15
stp steel original 404.15
repm cool cool_0000B_0404T_0934D
var TFU 800.0
var TMO 404.15
var DMO 0.9342
```

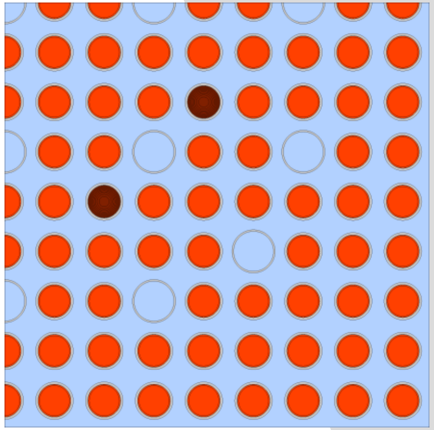


KrakenTools/tests/*GC_generator*

tutorial/rect/serpent/includes/scripts/generate_branches.py

Using `branch` and `casematrix` to set up history and branch calculations

```
branch cr_0_none
var CR 0

branch cr_1_boc
repu nocr boccr
var CR 1

branch cr_2_inc
repu nocr inccr
var CR 2

branch spa_0_none
var SPA 0

branch spa_1_grid
repu bare spa_zirc
repu uNosleeve uSleeve
var SPA 1
```

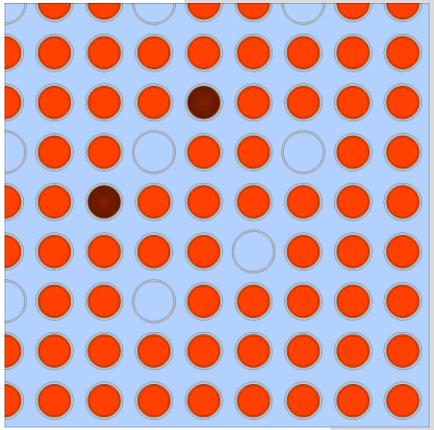


KrakenTools/tests/*GC_generator*

tutorial/rect/serpent/includes/scripts/generate_branches.py

Using `branch` and `casematrix` to set up history and branch calculations

```
branch hnomhis
stp F1400 original 561.0
stp F1500 original 561.0
stp F1800 original 561.0
stp F2400 original 561.0
stp F1506 original 561.0
stp F1806 original 561.0
stp F2405 original 561.0
stp F2409 original 561.0
stp EBOC original 404.15
stp INC original 404.15
stp air original 404.15
stp zirc original 404.15
stp steel original 404.15
repm cool cool_0000B_0404T_0934D
var hTFU 561.0
var hTMO 404.15
var hDMO 0.9342
var hCR 0
var hSPA 0
```

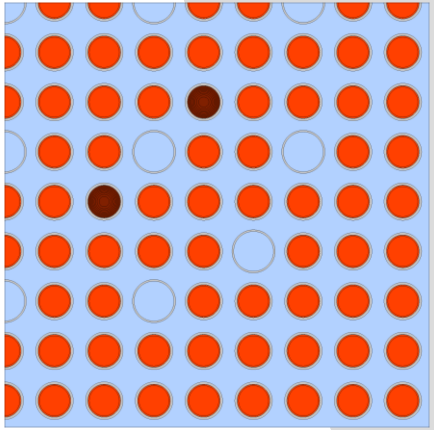


KrakenTools/tests/*GC_generator*

tutorial/rect/serpent/includes/scripts/generate_branches.py

Using `branch` and `casematrix` to set up history and branch calculations

```
casematrix nominals
2 hnomhis hoffhis
34 0 0.1 0.2 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 23 25 27 29 33 37 41
1 nominal
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix variations
2 hnomhis hoffhis
9 0 1 4 8 12 16 23 33 41
4 lo_tmo hi_tmo lo_tmo hi_tfu
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix reflector
1 hnomhis
1 0
3 nominal lo_tmo hi_tmo
1 cr_0_none
1 spa_0_none
```

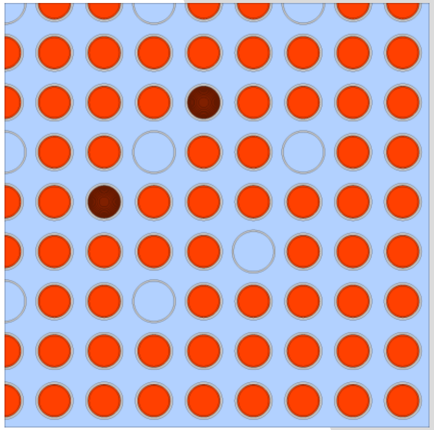


KrakenTools/tests/*GC_generator*

tutorial/rect/serpent/includes/scripts/generate_branches.py

Using `branch` and `casematrix` to set up history and branch calculations

```
# First run histories (burnup calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 -casematrix nominals 1 -1 A1
# ^Produces A1_nominals_h1.wrk binary restart
# (nominal history)

sss2 -omp 20 -casematrix nominals 2 -1 A1
# ^Produces A1_nominals_h2.wrk binary restart
# (off-nominal history)

# We can use the same restarts for coefficient calculations

ln -s A1_nominals_h1.wrk 390GO_variations_h1.wrk
ln -s A1_nominals_h2.wrk 390GO_variations_h2.wrk
```



Using `branch` and `casematrix` to set up history and branch calculations

```
# Then run branches (coefficient calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 -casematrix nominals 1 0 A1
# Runs all branches for nominal history based on
# A1_nominals_h1.wrk binary restart
# Has 1x3x2=6 branches
# (x15 burnups = 90 transport solutions)

sss2 -omp 20 -casematrix nominals 2 0 A1

sss2 -omp 20 -casematrix variations 1 0 A1
# Has 4x3x2=48 branches
# (x9 burnups = 432 transport solutions)

sss2 -omp 20 -casematrix coefficients 2 0 A1

# These 4 calculations could be distributed across 4
# calculation nodes on a cluster
```

```
casematrix nominals
2 hnomhis hoffhis
34 0 0.1 0.2 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 23 25 27 29 33 37 41
1 nominal
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix variations
2 hnomhis hoffhis
9 0 1 4 8 12 16 23 33 41
4 lo_tmo hi_tmo lo_tmo hi_tfu
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix reflector
1 hnomhis
1 0
3 nominal lo_tmo hi_tmo
1 cr_0_none
1 spa_0_none
```

Using `branch` and `casematrix` to set up history and branch calculations

```
# Then run branches (coefficient calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 -casematrix nominals 1 1 A1
sss2 -omp 20 -casematrix nominals 1 2 A1
sss2 -omp 20 -casematrix nominals 1 3 A1
sss2 -omp 20 -casematrix nominals 1 4 A1
sss2 -omp 20 -casematrix nominals 1 5 A1
sss2 -omp 20 -casematrix nominals 1 6 A1

# Runs single branches for nominal history based on
# A1_nominals_h1.wrk binary restart
# Has 1x3x2=6 branches
# (x15 burnups = 90 transport solutions)
```

```
casematrix nominals
2 hnomhis hoffhis
34 0 0.1 0.2 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 23 25 27 29 33 37 41
1 nominal
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix variations
2 hnomhis hoffhis
9 0 1 4 8 12 16 23 33 41
4 lo_tmo hi_tmo lo_tmo hi_tfu
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix reflector
1 hnomhis
1 0
3 nominal lo_tmo hi_tmo
1 cr_0_none
1 spa_0_none
```

Using branch and casematrix to set up history and branch calculations

```
# Then run branches (coefficient calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 -casematrix variations 2 1 A1
sss2 -omp 20 -casematrix variations 2 2 A1
sss2 -omp 20 -casematrix variations 2 3 A1
...
sss2 -omp 20 -casematrix variations 2 47 A1
sss2 -omp 20 -casematrix variations 2 48 A1

sss2 -omp 20 -casematrix variations 1 0 A1

# Runs single branches for off-nominal history based on
# A1_coefficients_h2.wrk binary restart
# Has 4x3x2=48 branches
# (x9 burnups = 432 transport solutions)

# Running branches separately yields 6*2+48*2 = 108
# separate Serpent runs which can be distributed across
# a computational cluster
```

```
casematrix nominals
2 hnomhis hoffhis
34 0 0.1 0.2 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 23 25 27 29 33 37 41
1 nominal
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix variations
2 hnomhis hoffhis
9 0 1 4 8 12 16 23 33 41
4 lo_tmo hi_tmo lo_tmo hi_tfu
3 cr_0_none cr_1_boc cr_2_inc
2 spa_0_none spa_1_grid

casematrix reflector
1 hnomhis
1 0
3 nominal lo_tmo hi_tmo
1 cr_0_none
1 spa_0_none
```

Output data from fuel GC calculations

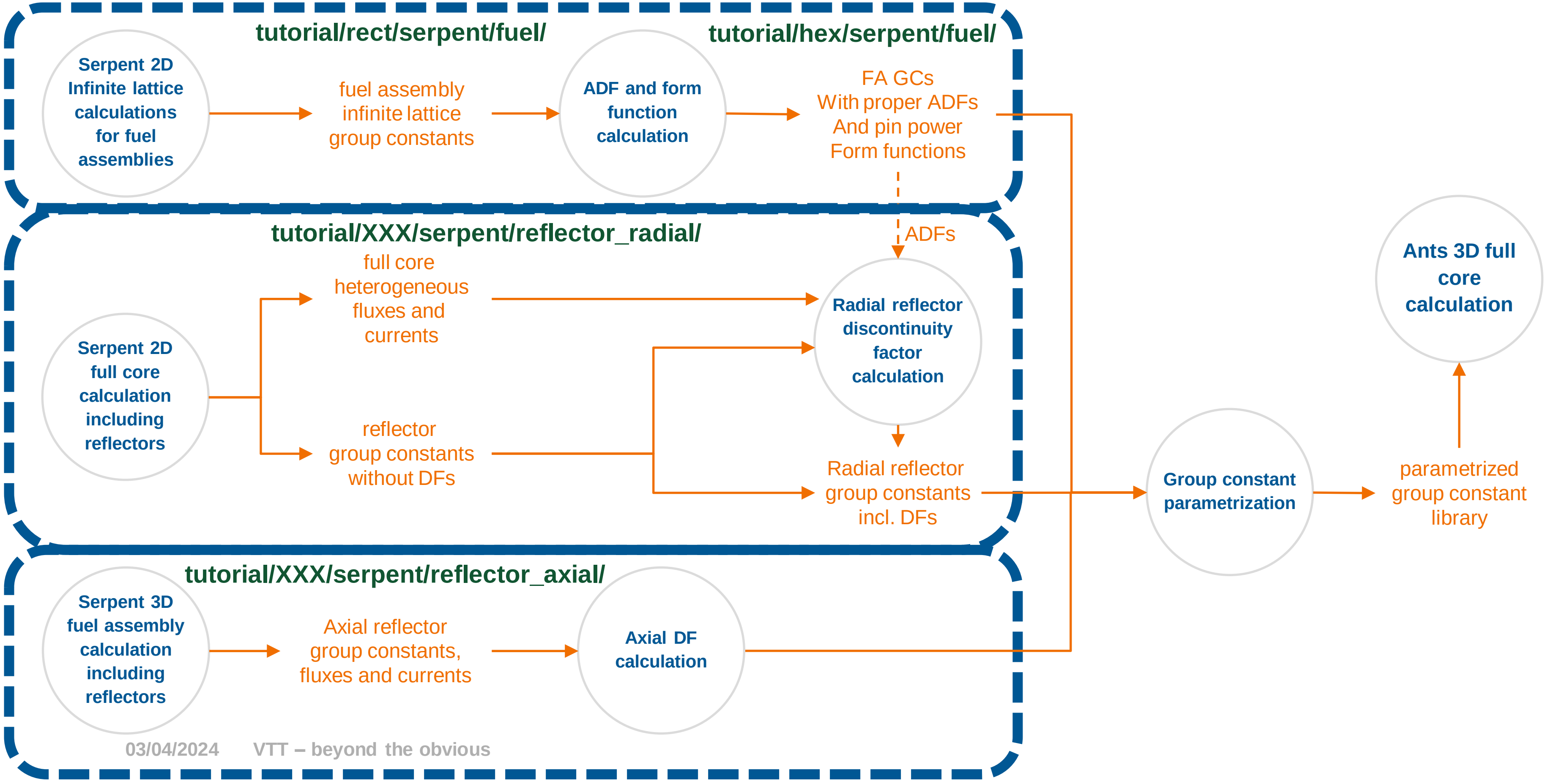
```
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>
```

tutorial/???/serpent/fit_fuel.py

- `A1_<case_name>_h<his_idx>_r<coe_idx>`**`.coe`** files
  - Contain homogenized few group constants for homogenized universes.
    - Includes cross sections, discontinuity factor data, pin power form function data, poison constants, basic time constants, microdepletion data etc.
    - var definitions from branch cards show up in .coe files to help identify, which file contains which data.
- `A1_<case_name>_h<his_idx>_r<coe_idx>`**`_res.m`** files
  - Contain some other important data not directly bound to homogenized universes.
- `A1_<case_name>_h<his_idx>_r<coe_idx>`**`_mdxb<coe_idx>.m`** files
  - Contain important data for microdepletion:
    - Fission spectra.
    - Decay reactions (decay constants, targets, branching ratios).
    - Neutron induced reactions (MTs, reaction products, Q-values).

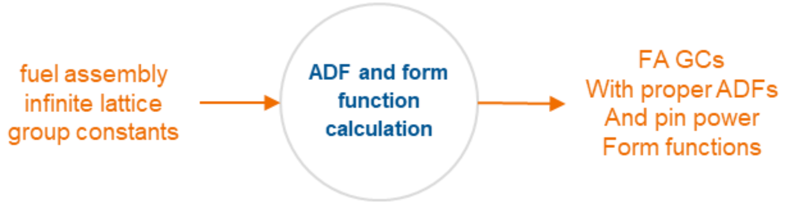
Can be read into Python objects with `serpentTools` and `KrakenTools`

Group constant generation



tutorial/rect/serpent/fuel/
tutorial/hex/serpent/fuel/
Serpent 2D Infinite lattice calculations for fuel assemblies
fuel assembly infinite lattice group constants
ADF and form function calculation
FA GCs With proper ADFs And pin power Form functions
ADFs
tutorial/XXX/serpent/reflector_radial/
full core heterogeneous fluxes and currents
Serpent 2D full core calculation including reflectors
reflector group constants without DFs
Radial reflector discontinuity factor calculation
Radial reflector group constants incl. DFs
tutorial/XXX/serpent/reflector_axial/
Serpent 3D fuel assembly calculation including reflectors
Axial reflector group constants, fluxes and currents
Axial DF calculation
Group constant parametrization
parametrized group constant library
Ants 3D full core calculation

Fuel ADFs and pin power form functions

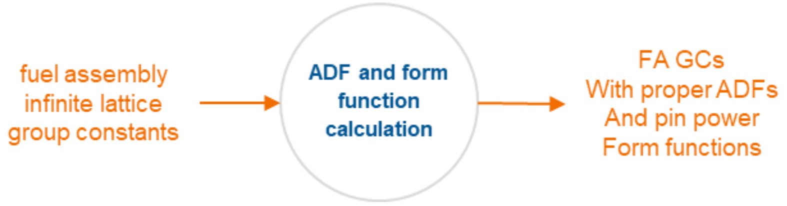

fuel assembly infinite lattice group constants
ADF and form function calculation
FA GCs With proper ADFs And pin power Form functions

tutorial/???/serpent/fit_fuel.py

- Assembly discontinuity factors and pin power form functions (FFs) are by definition dependent on the homogeneous flux solution.
  - In some simple cases, the homogeneous flux is constant inside the assembly and equal to the mean heterogeneous flux.
  - In general, an actual solution to the homogeneous problem is required.
  - Serpent has an internal diffusion flux solver, but as the homogeneous solution is dependent on the nodal model, using the Serpent calculated ADFs and form functions is **wrong** in general.

- Instead, Ants single node 2D simulations are executed using each set of generated group constants (and boundary conditions) to provide the corresponding homogeneous surface fluxes and homogeneous pin-cell fluxes.
  - The process is heavily automated: `krakentools.ants.evaluate_ffs_and_adfs_with_ants()`
  - ADFs and FFs can be evaluated based on known heterogeneous and homogeneous data.

Fuel ADFs and pin power form functions

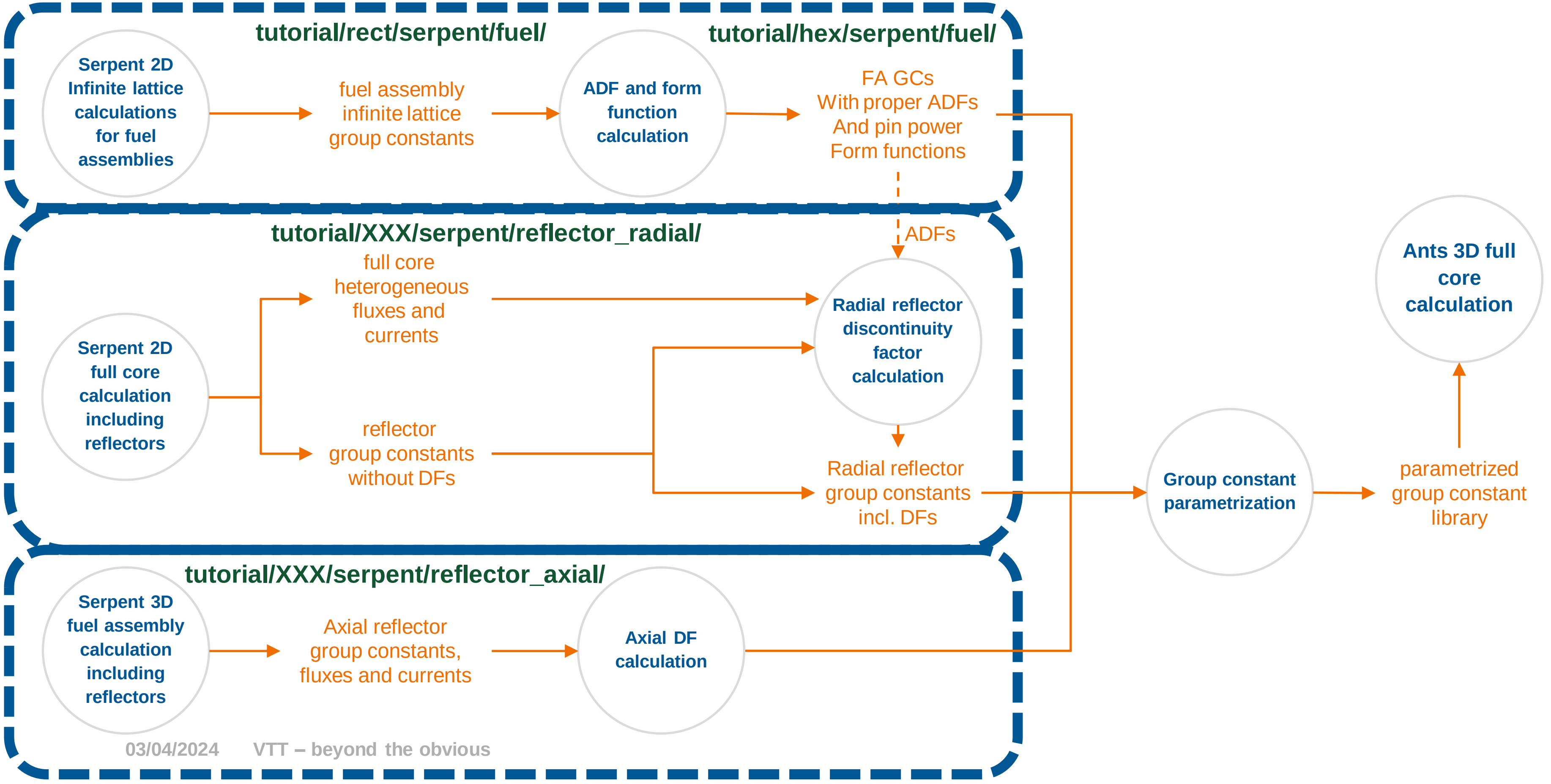


tutorial/???/serpent/fit_fuel.py

- Instead, Ants single node 2D simulations are executed using each set of generated group constants (and boundary conditions) to provide the corresponding homogeneous surface fluxes and homogeneous pin-cell fluxes.
  - The process is heavily automated: `krakentools.ants.evaluate_ffs_and_adfs_with_ants()`
  - ADFs and FFs can be evaluated based on known heterogeneous and homogeneous data.

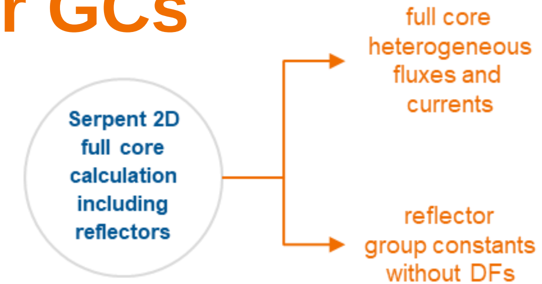
Heterogeneous data utilized from .coe files:
DF_HET_SURF_FLUX
PPW_POW

Group constant generation

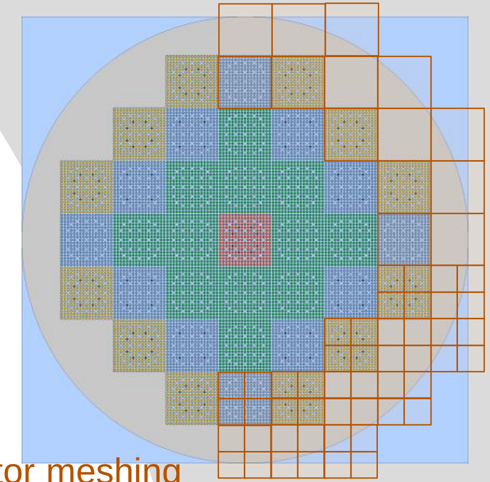


tutorial/rect/serpent/fuel/
tutorial/hex/serpent/fuel/
Serpent 2D Infinite lattice calculations for fuel assemblies
fuel assembly infinite lattice group constants
ADF and form function calculation
FA GCs With proper ADFs And pin power Form functions
ADFs
tutorial/XXX/serpent/reflector_radial/
Serpent 2D full core calculation including reflectors
full core heterogeneous fluxes and currents
reflector group constants without DFs
Radial reflector discontinuity factor calculation
Radial reflector group constants incl. DFs
tutorial/XXX/serpent/reflector_axial/
Serpent 3D fuel assembly calculation including reflectors
Axial reflector group constants, fluxes and currents
Axial DF calculation
Group constant parametrization
parametrized group constant library
Ants 3D full core calculation

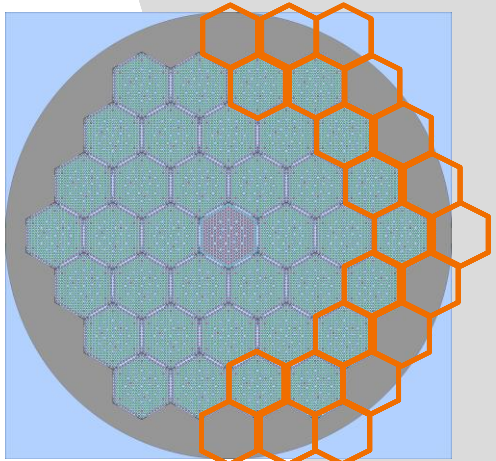
Best practices calculation chain reflector GCs


Serpent 2D full core calculation including reflectors
full core heterogeneous fluxes and currents
reflector group constants without DFs

1x1 reflector meshing



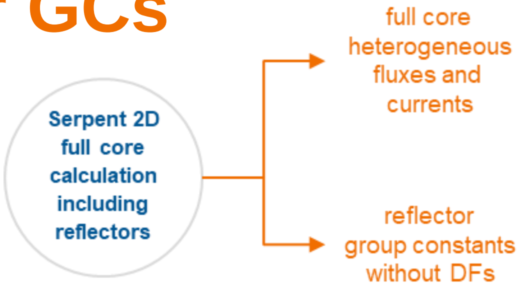
2x2 reflector meshing

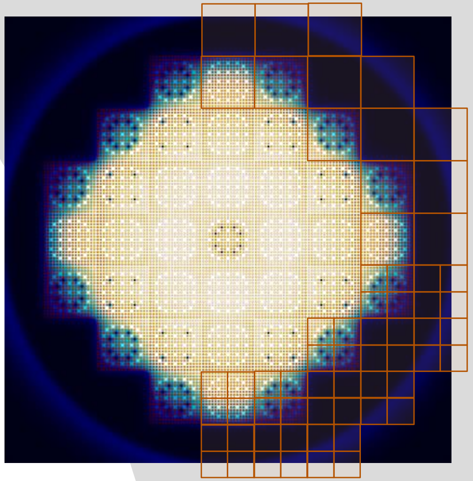


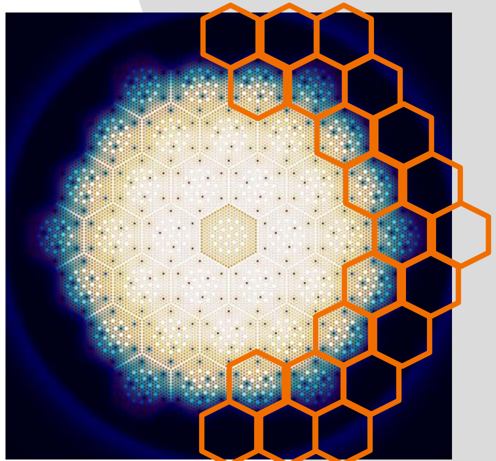
- Radial reflector constants are generated using a 2D full core geometry.
- Multiple transport solutions at zero burnup:
  - Cover different (T_{cool} , ρ_{cool} , C_B) variations.
- Diffusion coefficients transport corrected for H-1 in water.
  - set trc cool "s2v0_endfb71.h_in_h2o.trcdata" 1.000000E-11 10010

tutorial/???/serpent/preprocess_radial.py

Best practices calculation chain reflector GCs


Serpent 2D full core calculation including reflectors
full core heterogeneous fluxes and currents
reflector group constants without DFs





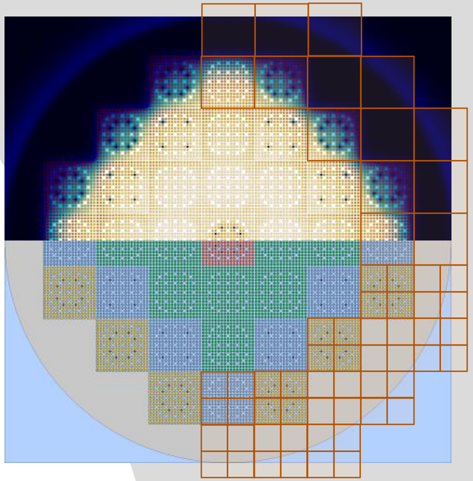
- Radial reflector constants are generated using a 2D full core geometry.
- Multiple transport solutions at zero burnup:
  - Cover different (T_{cool} , ρ_{cool} , C_B) variations.
- Diffusion coefficients transport corrected for H-1 in water.
  - set trc cool "s2v0_endfb71.h_in_h2o.trcdata" 1.000000E-11 10010
- Hexagonal lattice radial reflector currently homogenized using hexagonal nodes. In the future, also with triangular nodes.

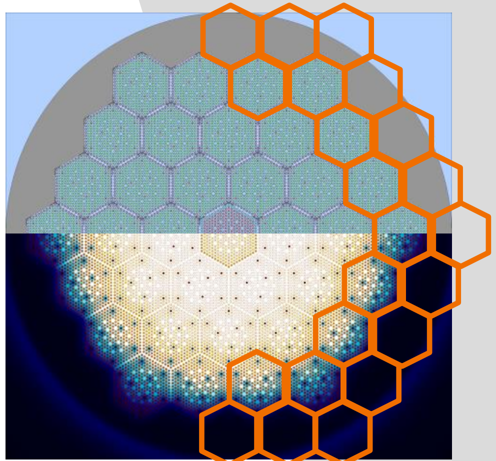
tutorial/???/serpent/preprocess_radial.py

Superimposed universes for reflector group constants

```
% --- First define bounding surfaces for superimposed universes
%     (must not overlap)

% --- Surface bounding node RR01

surf s_bound_RR01 hexxprism 165.20000000000002 0.0 11.8 30 50

% --- Surface bounding node RR02

surf s_bound_RR02 hexxprism 177.0 20.43819952931275 11.8 30 50

% --- Surface bounding node RR03

surf s_bound_RR03 hexxprism 165.20000000000002 40.8763990586255 11.8 30 50

...
```

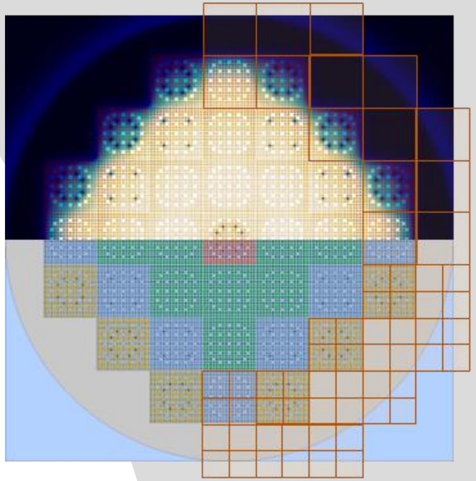
tutorial/???/serpent/preprocess_radial.py

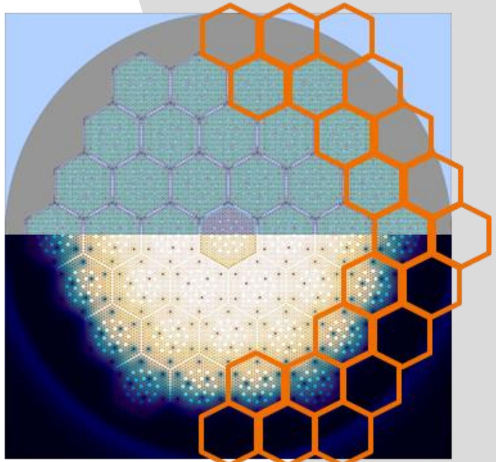




Superimposed universes for reflector group constants

```
% --- Then define (superimposed) universes based on the surfaces

% --- Superimposed universe for node RR01

cell  c_SI_RR01  -u_SI_RR01  void  -s_bound_RR01

% --- Superimposed universe for node RR02

cell  c_SI_RR02  -u_SI_RR02  void  -s_bound_RR02

% --- Superimposed universe for node RR03

cell  c_SI_RR03  -u_SI_RR03  void  -s_bound_RR03

...
```



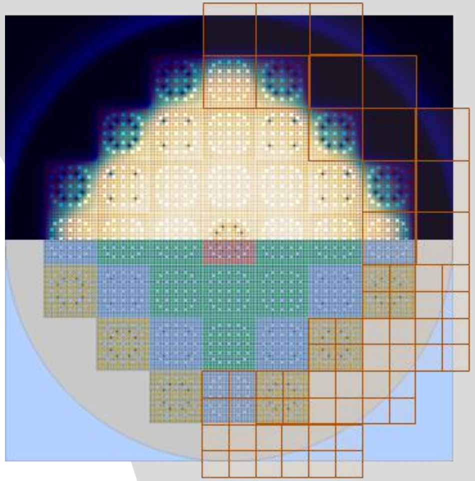


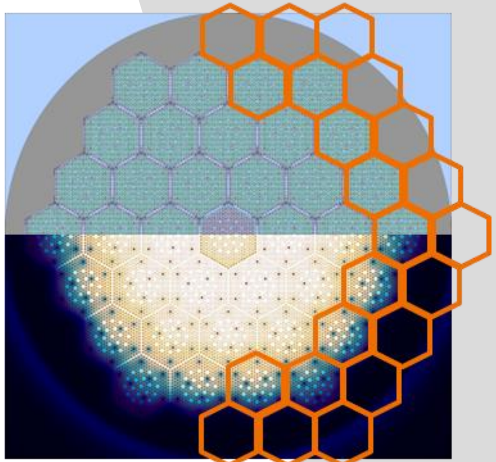
tutorial/???/serpent/preprocess_radial.py

Superimposed universes for reflector group constants

```
% --- Finally setup gcu and adf cards for the superimposed universes

set gcu  -u_SI_RR01
set adf  -u_SI_RR01 s_bound_RR01 0

set gcu  -u_SI_RR02
set adf  -u_SI_RR02 s_bound_RR02 0

set gcu  -u_SI_RR03
set adf  -u_SI_RR03 s_bound_RR03 0

...
```

- Universes linked to gcu or adf cards, but that are not part of the geometry are treated by Serpent as superimposed on top of the geometry.
- Some slowdown to simulations due to (additional) checking if collision is in a superimposed universe or crosses the boundary of one at each interaction site.





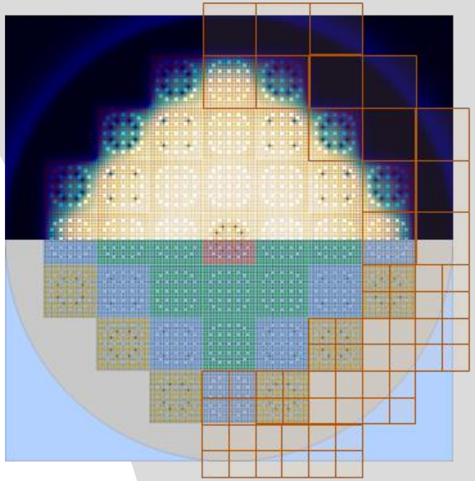
Output data

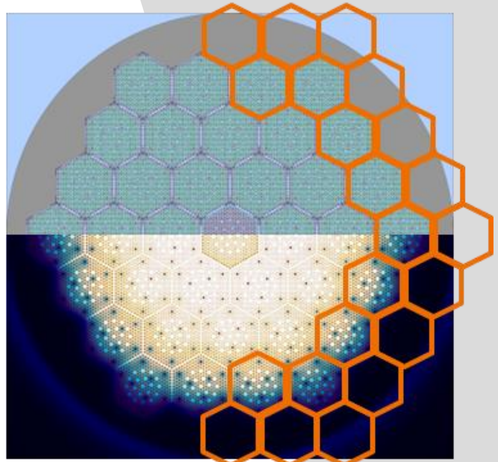
```
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>
# Can run with

sss2 -omp 20 -casematrix reflector -1 <coe_idx> fullcore
```

- Reflector casematrix may not need > 0 burnups or fuel temperature branches.
- `fullcore_<case_name>_h<his_idx>_r<coe_idx>.coe` files
  - Contain homogenized few group constants for homogenized universes.
    - Includes group constants and heterogeneous node boundary fluxes and currents.
    - var definitions from branch cards show up in .coe files to help identify, which file contains which data.
- `fullcore_<case_name>_h<his_idx>_r<coe_idx>_res.m` files
  - Contain some other important data not directly bound to homogenized universes.

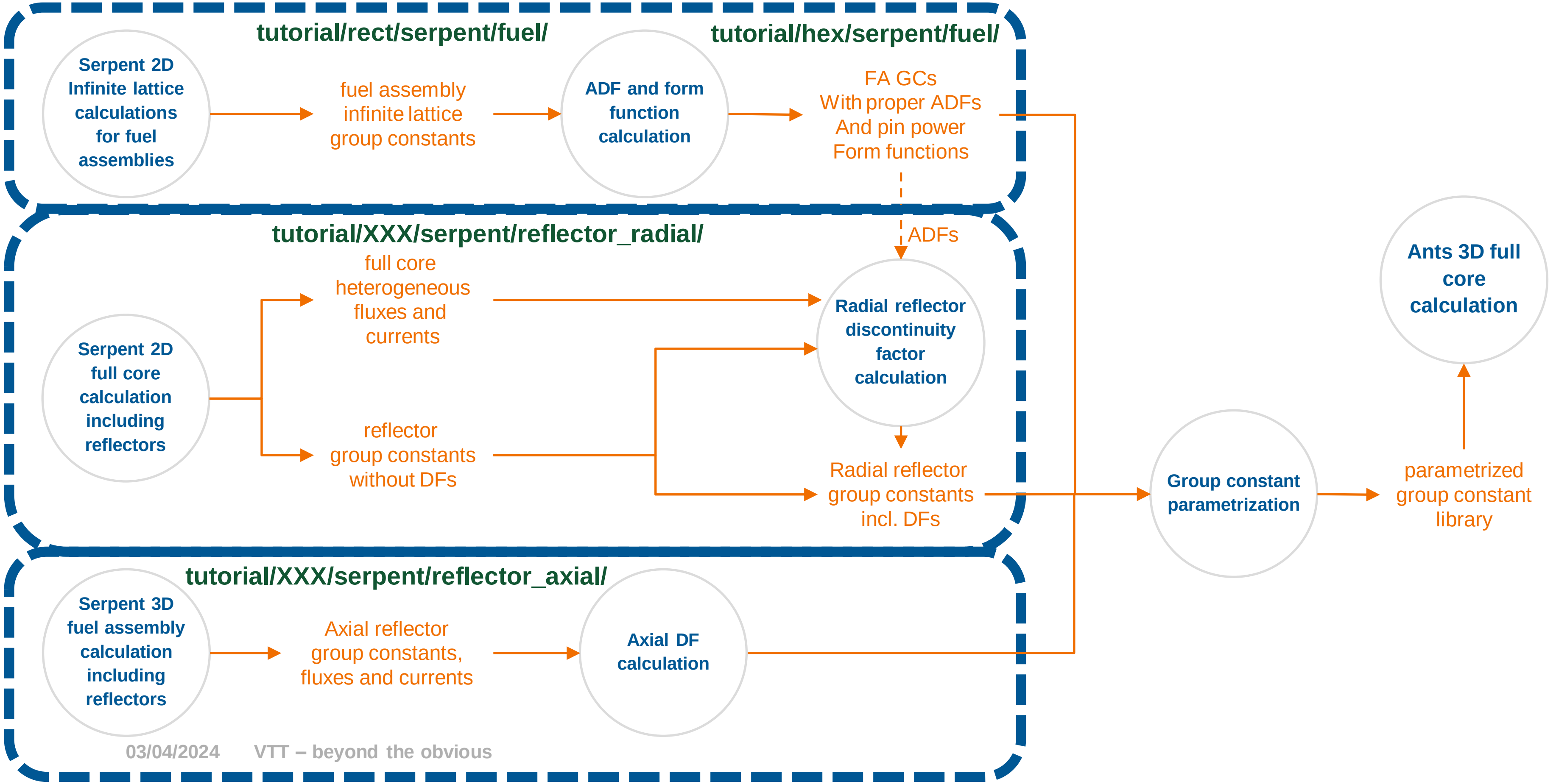
KrakenTools collects results from 360 degree core and averages results over symmetric positions

tutorial/???/serpent/process_radial.py





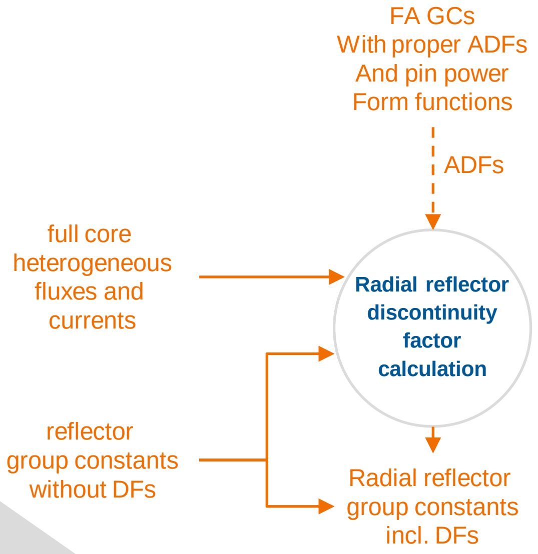
Group constant generation



tutorial/rect/serpent/fuel/
tutorial/hex/serpent/fuel/
Serpent 2D Infinite lattice calculations for fuel assemblies
fuel assembly infinite lattice group constants
ADF and form function calculation
FA GCs With proper ADFs And pin power Form functions
ADFs
tutorial/XXX/serpent/reflector_radial/
full core heterogeneous fluxes and currents
Serpent 2D full core calculation including reflectors
reflector group constants without DFs
Radial reflector discontinuity factor calculation
Radial reflector group constants incl. DFs
Ants 3D full core calculation
Group constant parametrization
parametrized group constant library
tutorial/XXX/serpent/reflector_axial/
Serpent 3D fuel assembly calculation including reflectors
Axial reflector group constants, fluxes and currents
Axial DF calculation

Best practices calculation chain reflector discontinuity factors

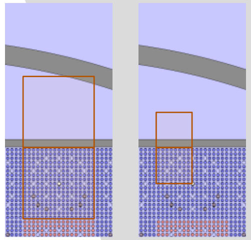
`krakentools.reflectorhg.solve_ants_2d_nodes()`



FA GCs
With proper ADFs
And pin power
Form functions
ADFs
full core heterogeneous fluxes and currents
Radial reflector discontinuity factor calculation
reflector group constants without DFs
Radial reflector group constants incl. DFs

1. The reflector side DF is first evaluated simply as the ratio of the heterogeneous surface flux from the Serpent 3D solution and the homogeneous surface flux from a single node Ants calculation using group constants and boundary condition currents from the Serpent3D solution:

$$f_{\text{refl.}}^{\text{Ants}} = \frac{\phi_{\text{refl.}}^{\text{Serpent3D}}}{\mathbf{\Phi}_{\text{refl.}}^{\text{Ants}}}$$



2. The fuel side DF is similarly evaluated

$$f_{\text{fuel}}^{\text{Ants}} = \frac{\phi_{\text{fuel}}^{\text{Serpent3D}}}{\mathbf{\Phi}_{\text{fuel}}^{\text{Ants}}}$$

3. This DF is then corrected[8] by the ratio of the assembly discontinuity factor $f_{\text{fuel}}^{\text{ADF}}$ evaluated for the fuel assembly in the infinite lattice 2D Serpent calculation and $f_{\text{fuel}}^{\text{Ants}}$:

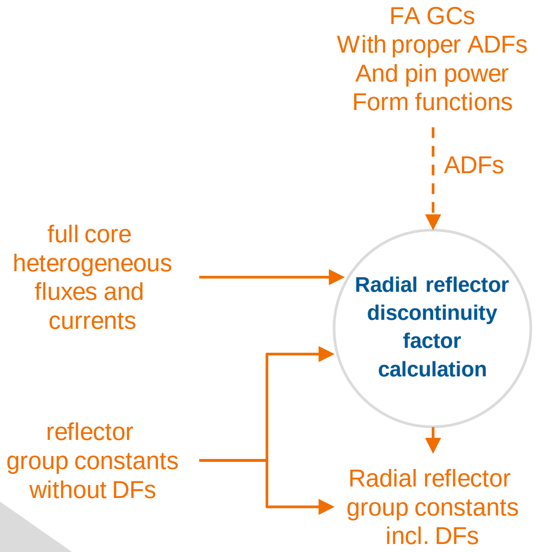
$$f_{\text{refl.}} = f_{\text{refl.}}^{\text{Ants}} \times \frac{f_{\text{fuel}}^{\text{ADF}}}{f_{\text{fuel}}^{\text{Ants}}}$$

[8] K. S. Smith. "Nodal diffusion methods and lattice physics data in LWR analyses: Understanding numerous subtle details". *Progress in Nuclear Energy* 101 (2017), pp. 360–369

tutorial/???/serpent/process_radial.py

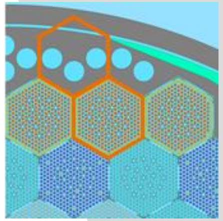
Best practices calculation chain reflector discontinuity factors

`krakentools.reflectorhg.solve_ants_2d_nodes()`


FA GCs
With proper ADFs
And pin power
Form functions
ADFs
full core heterogeneous fluxes and currents
Radial reflector discontinuity factor calculation
reflector group constants without DFs
Radial reflector group constants incl. DFs

1. The reflector side DF is first evaluated simply as the ratio of the heterogeneous surface flux from the Serpent 3D solution and the homogeneous surface flux from a single node Ants calculation using group constants and boundary condition currents from the Serpent3D solution:

$$f_{\text{refl.}}^{\text{Ants}} = \frac{\phi_{\text{refl.}}^{\text{Serpent3D}}}{\boldsymbol{\Phi}_{\text{refl.}}^{\text{Ants}}}$$



2. The fuel side DF is similarly evaluated

$$f_{\text{fuel}}^{\text{Ants}} = \frac{\phi_{\text{fuel}}^{\text{Serpent3D}}}{\boldsymbol{\Phi}_{\text{fuel}}^{\text{Ants}}}$$

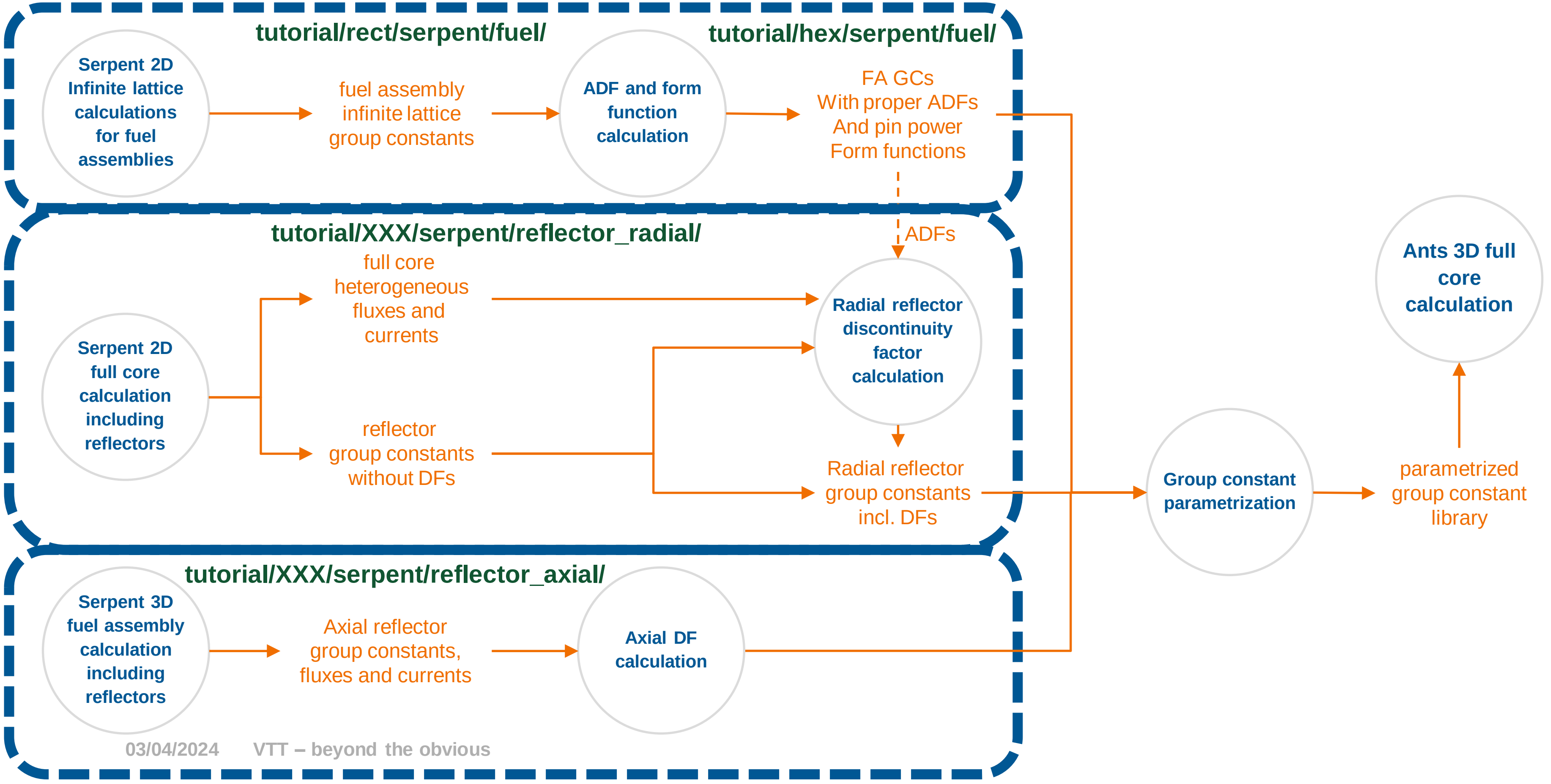
3. This DF is then corrected[8] by the ratio of the assembly discontinuity factor $f_{\text{fuel}}^{\text{ADF}}$ evaluated for the fuel assembly in the infinite lattice 2D Serpent calculation and $f_{\text{fuel}}^{\text{Ants}}$:

$$f_{\text{refl.}} = f_{\text{refl.}}^{\text{Ants}} \times \frac{f_{\text{fuel}}^{\text{ADF}}}{f_{\text{fuel}}^{\text{Ants}}}$$

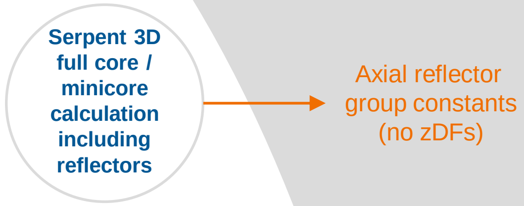
[8] K. S. Smith. "Nodal diffusion methods and lattice physics data in LWR analyses: Understanding numerous subtle details". *Progress in Nuclear Energy* 101 (2017), pp. 360–369

tutorial/???/serpent/process_radial.py

Group constant generation


tutorial/rect/serpent/fuel/
tutorial/hex/serpent/fuel/
Serpent 2D Infinite lattice calculations for fuel assemblies
fuel assembly infinite lattice group constants
ADF and form function calculation
FA GCs With proper ADFs And pin power Form functions
ADFs
tutorial/XXX/serpent/reflector_radial/
Serpent 2D full core calculation including reflectors
full core heterogeneous fluxes and currents
reflector group constants without DFs
Radial reflector discontinuity factor calculation
Radial reflector group constants incl. DFs
tutorial/XXX/serpent/reflector_axial/
Serpent 3D fuel assembly calculation including reflectors
Axial reflector group constants, fluxes and currents
Axial DF calculation
Group constant parametrization
parametrized group constant library
Ants 3D full core calculation

Best practices calculation chain Axial reflector homogenization

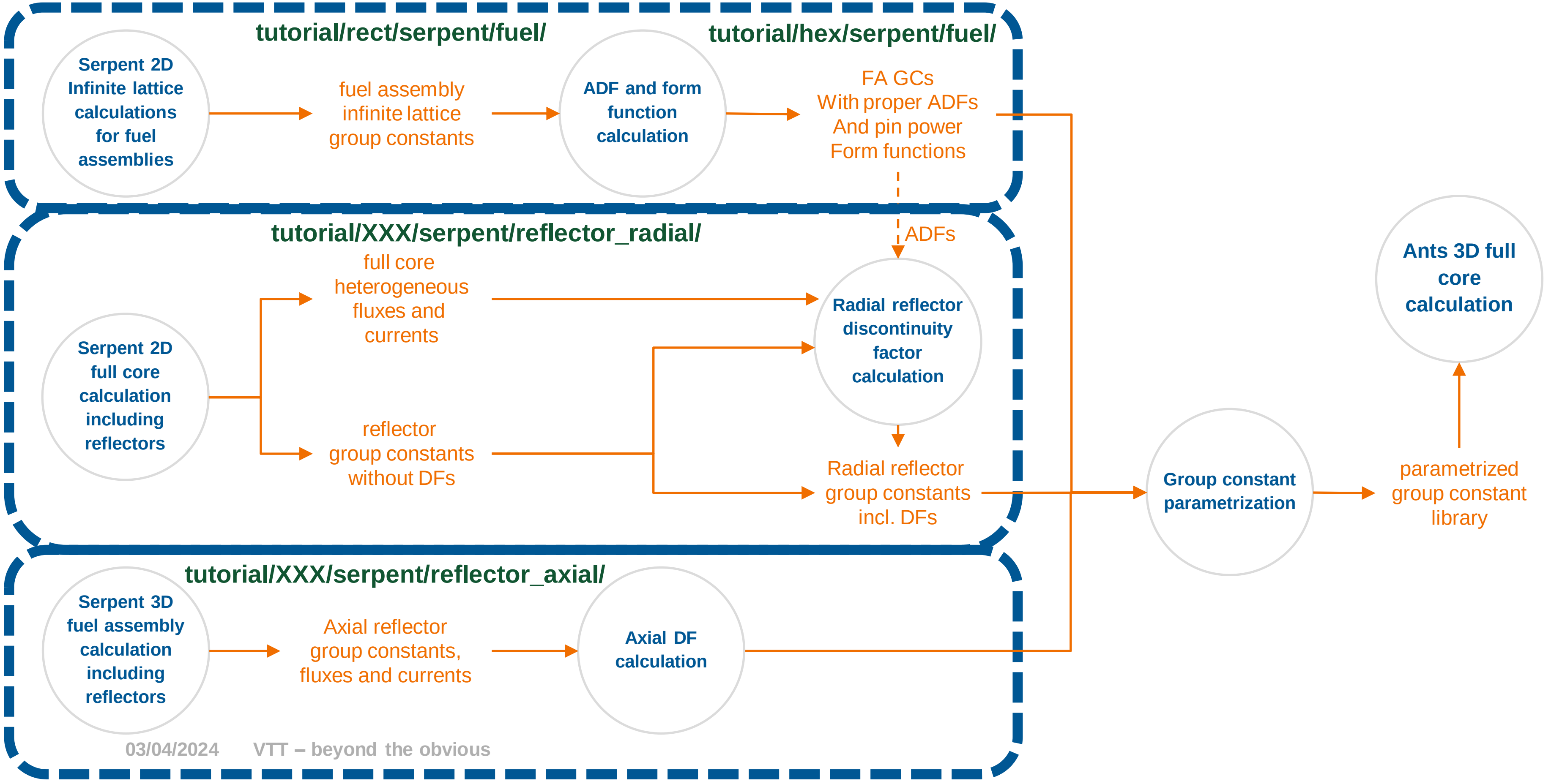

Serpent 3D full core / minicore calculation including reflectors
Axial reflector group constants (no zDFs)

- Rather similar to radial reflector homogenization, but typically uses a 3D model :
  - Single assembly.
  - Colorset.
  - Full core.
- May need control rod branches?
- Superimposed universes set up similar to radial reflector.
- Axial discontinuity factors calculated similar to radial ones.
  - Except no correction with ADF, naturally.

tutorial/???/serpent/preprocess_axial.py

tutorial/???/serpent/process_axial.py

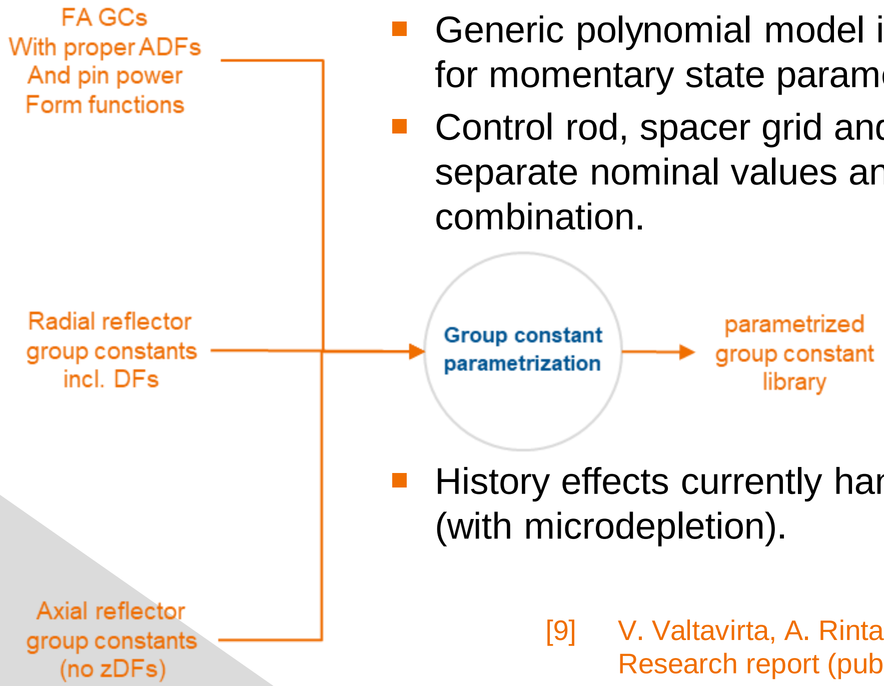
Group constant generation


tutorial/rect/serpent/fuel/
tutorial/hex/serpent/fuel/
Serpent 2D Infinite lattice calculations for fuel assemblies
fuel assembly infinite lattice group constants
ADF and form function calculation
FA GCs With proper ADFs And pin power Form functions
ADFs
tutorial/XXX/serpent/reflector_radial/
full core heterogeneous fluxes and currents
Serpent 2D full core calculation including reflectors
reflector group constants without DFs
Radial reflector discontinuity factor calculation
Radial reflector group constants incl. DFs
Ants 3D full core calculation
Group constant parametrization
parametrized group constant library
tutorial/XXX/serpent/reflector_axial/
Serpent 3D fuel assembly calculation including reflectors
Axial reflector group constants, fluxes and currents
Axial DF calculation

Group constant parametrization

`krakentools.groupconstants.genpoly`

- Generic polynomial model implemented in Ants[9] with a polynomial fit for momentary state parameters. (T_{fuel} , T_{cool} , ρ_{cool} , C_B).
- Control rod, spacer grid and instrumentation tube are treated as select variables with separate nominal values and polynomial coefficients tabulated for each possible combination.



tutorial/???/serpent/fit_fuel.py
tutorial/???/serpent/process_radial.py
tutorial/???/serpent/process_axial.py
tutorial/???/serpent/combine_gcs.py

- History effects currently handled using a plutonium history approach[10] (with microdepletion).

[9] V. Valtavirta, A. Rintala. “Specifications for the generic polynomial group constant model of Ants”, Research report (public), VTT-R-00154-21, 2021.
[10] Y. Bilodid. “Spectral history modelling in the reactor dynamics code DYN3D”, PhD thesis, Technical University of Dresden, 2014 (HZDR-051).

Summary

Summary

- Serpent has been developed for group constant generation from the start.
- In the recent years, the application of Serpent for such tasks at VTT has been made more routine.
- The process of generating group constants for fuel cycle simulations is quite clear:
  - Fuel assemblies, reflector regions, proper DFs and form functions.
  - Use of `branch` cards for setting up history and branch conditions.
  - Use of `casematrix` to set up the calculation matrix and run it efficiently.
  - Group constants for Ants parametrized using KrakenTools.

VTT

beyond the obvious

Ville Valtavirta
ville.valtavirta@vtt.fi