Manual of CBZ/Burner

Go CHIBA

September 9, 2024

Contents

1 Introduction

Several isotopes of heavy nuclides such as uranium and plutonium can fission with high probability when they collide with neutrons, and an energy of approximately 200 MeV is generated per one fission reaction. As the results, the heavy nucleus is divided into two fragments known as *fission product* (FP). Most (approximately 80%) of the fission energy become kinetic energy of FP and these energy finally is converted to the heat energy and the electric energy in the nuclear power plants.

Since heavy nuclei are converted to FP by fission reactions, the amount of the heavy nuclei in nuclear fuel gradually decreases with the operation of the nuclear power plant. The heavy nuclides which can fission contribute to the fission chain reaction, so if the amount of them decreases, it becomes hard to maintain the fission chain reaction. Thus, in nuclear power plants, relatively large amount of nuclear fuels are loaded to a nuclear reactor when it starts its operation. If the large amount of nuclear fuels are loaded, the reactor core becomes a supercritical state, so the large positive reactivity is compensated by inserting control rods made of neutron absorbing materials, by diluting the boron, which is also strong neutron absorbing material, into the coolant, or by mixing the other neutron absorbing material gadolinium with fuels, and the critical state is maintained. As the power plant operation continues, the amount of fissioning materials in nuclear fuels is decreased, and reactivity is also decreased. Thus the control rod is gradually withdrawn or the boron concentration is reduced to maintain the critical state.

Uranium-238, which is main consituent of the uranium fuel, is converted to uranium-239 by absorbing a neutron, and this uranium-239 is converted to plutonium-238, which is a fissioning nuclide, via sequential two β decays. This means that the amounts of uranium-235 and plutonium-239 decreased with the power plant operation, but the same time, plutonium-239 is newly generated by the neutron absorption of uranium-238 and it can contribute to teh fission chain reaction.

Furthermore, the amount of FP generated through the fission reactions is also important. The kinds of FP are over 1,000, and some of them can emit quite strong radiation rays and some have long half-lives and remain in the spent nuclear fuels. When we consider the reprocessing, the intermediate storage, and the final disposal of the spent nuclear fuel, we need to grasp how much the radioactive FP are contained in the spent fuels. Also the amount of FP accumulated in the spent fuel is important in the accident analysis. Each of FP is generated by a fission reaction with a given probability, and most of FP are neutron-rich unstable nuclides and are transmuted to the different nuclide according to their intrinsic decay half-lives. Nuclides included in the nuclear fuel can also convert to the other nuclides via neutron-nuclide reactions.

As described below, the compositions of the nuclear fuels loaded to the nuclear reactor change with the plant operation. This is called *nuclear fuel burnup*, and changes in the nuclear fuel composition during the nuclear fuel burnup should be accurately predicted.

A reactor physics code system CBZ has several packeges to numerically simulate the nuclear fuel burnup, and this document is devoted to the brief description of the package of the nuclear fuel burnup calculations for a single nuclear fuel pin cell, CBZ/Burner.

2 Required data files

The Burner module requires several specific data for the nuclear fuel burnup calculations such as multi-group reaction cross section data, the FP decay data, the FP yield data, etc. The multi-group cross section data are stored in the `CBGLIB` directory, and the fuel burnup-relevant data are stored in the `CBGLIB_BURN` directory.

When one wants to use the Burner module, the following structure of the directories would be typical.

/home/
- `CBG/src/` (CBZ source program)
- `CBGCAL/Burner/` (CBZ calculation files)
- `CBGLIB/`
  - `j4.107g.iwt4/` (JENDL-4.0-based 107-group library)
  - `Bell-107g/`(Optimized Bell-factor for 107-group library)
- `CBGLIB_BURN/`
  - `CBG_CHAIN/` (Fuel burnup chain data)
  - `CBG_FY/` (Cumulative fission yield data)
  - `DDFile/` (Decay data)
  - `ReactionEnergy/` (Data of generated energy per one fission reaction)
  - `atomic_mass/` (Atomic mass data)
  - `decay_constant/` (Decay constant data)

The present tutorial will give explanations based on the sample file `main.burn_tutorial.cxx` in the directory `Burner`.

The fuel burnup calculation can be carried out by compiling and running this `main.burn_tutorial.cxx` file. If you fail to do this, please consult with Chiba.

3 Generation of input data

In light-water reactor fuel burnup analyses, **the single fuel pincell model**, which consists of the fuel pellet, the cladding, and the surronding coolant, is sometimes employed. Out of the coolant region, the specific boundary conditions are assigned, and this model is equivalent with the infinite array of this cell on the radial plane. The axial direction is assumed to have infinite length, this single fuel pincell model is a two-dimensional model. An example of a BWR fuel assembly and the associated single pincell model is shown in **Fig. 1**.

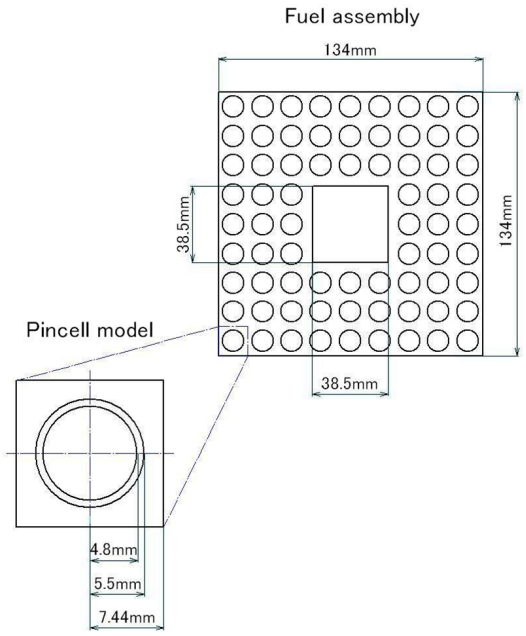

Fig. 1: BWR fuel assembly and single pincell model

In the following, how to use the Burner module is described by seeing a sample input file `main.burn_tutorial.cxx`.

3.1 Assignment of multi-group cross section data and fuel burnup chain data

In nuclear fuel burnup calculations, we have to know how many reactions which accompany the nuclide transmutation occur. Among these kinds of reactions, the fission reaction, the neutron capture reaction, and (n,2n) reactions are important. When the fission reaction occurs, the target nucleus becomes two other nucleus. When the neutron capture reaction occurs, the target nucleus is converted to an other nucleus whose number of neutrons is large by one than the original one. When the (n,2n) reaction occurs, the target nucleus becomes other one whose number of neutrons is small by one than the original. If we want to know how many such reactions occur, we require the corresponding reaction data. Furthermore, the level and energy spectrum of the neutron flux are also required [1], and these are obtained by the calculations with the single pincell model. The reaction cross section data are required also in this neutron flux calculation.

Nuclide transmutations occur through the various neutron-nuclide reactions, and the information on which nuclide is newly generated through the reaction is required for each reaction and each target nuclide. In addition, the nuclide transmutations also occur through the radioactive decay, and the information on which nuclide is generated through the decay is required for each decay path (or mode) and each concerned unclide. These information are summarized as the **nuclide transmutation (or burnup) chain** in the numerical calculations. The good example of the transmutation chain can be found in page 186 in the manual of the code system SRAC-2006 (JAEA-Data/Code 2007-004) [2] During this tutorial, the nuclide transmutation chain for heavy nuclides presented in Fig. 3.3-1 in page 186 and that for FP in page 190 would be useful. The nuclide transmutation chain for important heavy nuclides in SRAC-2006 is shown in **Fig. 2** as an example.

[1] The information on the neutron flux energy spectra are required since the reaction cross sections significantly depend on the neutron incident energy.

[2] http://jolissrch-inter.tokai-sc.jaea.go.jp/pdfdata/JAEA-Data-Code-2007-004.pdf

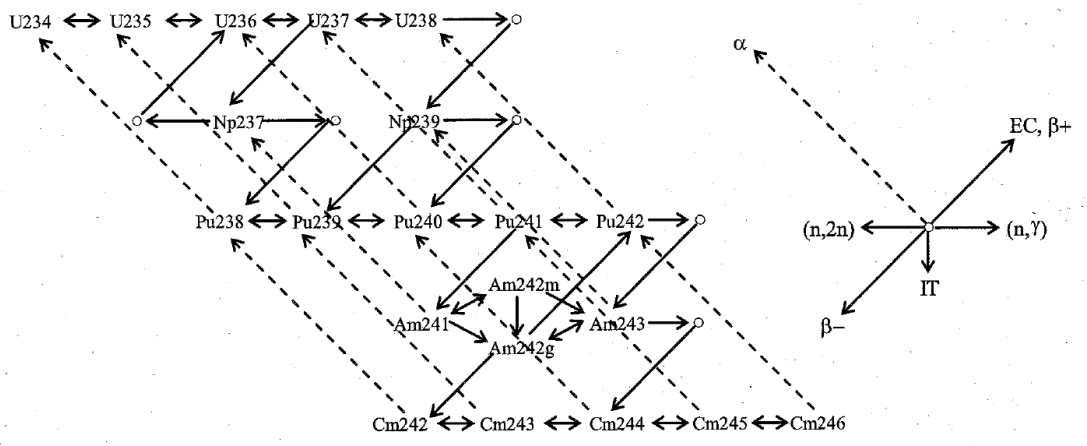

Fig. 2: Nuclide transmutation chain for important heavy nuclides in SRAC-2006

In CBZ, the data for the reaction cross section and the nuclide transmutation chain are defined as external files, so these data should be read in calculations. The following shows the part reading the external files on the cross section data and the transmutation chain.

Listing 1: Reading the cross section data and the transmutation chain data

```
1  Burner bn;
2  Burnup bu;
3
4  string cbglibdir("../../");
5
6  // (common)
7  bu.ReadReactionEnergyFromFile(cbglibdir,"sractype");
8  //bu.ReadReactionEnergyFromFile(cbglibdir,"fr_standard");
9  bu.GetBurnupChain().ReadDecayConstantFromFile(cbglibdir,"srac_org");
10 bu.ReadAtomicMassDataFromFile(cbglibdir,"jeff311");
11
12 // (FP197)
13 bn.SetLibrary(cbglibdir,"jendl-4.0");
14 bu.GetBurnupChain().Set21HeavyMetalChain();
15 bu.GetBurnupChain().ReadFPYieldDataFromFile(cbglibdir,"fp197.j2011t");
16 bu.GetBurnupChain().OverWritingChainData(cbglibdir,"fp197.j2011t");
17
18 // (Actinide decay heat)
19 bn.AddActinideDecayHeatDataToBurnupChain(cbglibdir,"dd.jeff311",bu);
20
21 // (Dose coefficient data)
22 bu.GetBurnupChain().ReadDoseCoefficientData(cbglibdir);
```

The module Burner to conduct the pin-cell burnup calculations is a class of C++, so the instance of the `Burner` class should be generated in the calculations. In line 1, the instance `bn` of the `Burner` class is generated. The following calculations are carried out with the instance `bn`.

Whereas the `Burner` class is to conduct the pin-cell burnup calculations, the actual nuclide transmutation calculations are carried out with the `Burnup` class. The `Burnup` class possesses the one-group cross section data and the instance of the `BurnupChain` class to define the nuclide transmutation chain, and carry out the burnup calculations. The instance `bu` of the `Burnup` class is generated in line 2.

In Burner, the reaction cross section data are located on the `CBGLIB` directory, and the transmutation data are on the `CBGLIB_BURN` directory. In line 4, the information on these directories are provided.

The multi-group cross section data are read through the method `bn.SetLibrary` in line 13. In this example, 107-group cross section library based on the evaluated nuclear data file JENDL-4.0 is read. When Burner is used, the `XSLibrary` class defines the multi-group cross section library is defined internally, so users do not have to treat it directly. If users want to use the methods of the `XSLibrary` class, they can retreive the instance of the `XSLibrary` class through the `GetXSLibrary` method of the `Burner` class.

The nuclide transmutation chain is defined as a member variable of the `Burnup` class as the instance of the `BurnupChain` class. In line 14, it declares to use the 21-heavy nuclide chain consisting of uranium to curium. In lines 15 and 16, fission product yield data and fission product chain data are read from the external files. The data contained in the nuclide transmutation chain are different among different evaluated nuclear data files, and this sample reads the data based on the JENDL FP decay data file 2011.

The energy data generated through the nuclide reaction, the decay data, and the atomic mass data are read in lines 7, 9, and 10, respectively.

3.2 Retreival of the nuclide transmutation chain data

The data described in the preceding subsection are stored in the instance `bu` of the `Burnup` class, and several methods have been implemented for users to access these data. This subsection describes these methods implemented into the `BurnupChain` class to define the nuclide transmutation chain. More datailed information can be found in the different manual for the `CBZ/BurnupChain` and `NuclideChainData`, so you can refer them.

The following is an example to print the nuclide transmutation chain data out.

Listing 2: Printing out of the nuclide transmutation chain data

```
1  //bu.GetBurnupChain().ShowHalfLife();
2  //bu.GetBurnupChain().ShowDoseCoefficient();
3  //bu.GetBurnupChain().ShowChainData();
4  //bu.GetBurnupChain().ShowChainDataToOrigin("Ba137");
5  //bu.GetBurnupChain().ShowNGBranchingRatio();
6  //bu.GetBurnupChain().ShowFPYieldForFP("Cs137");
7  //bu.GetBurnupChain().ShowFPYield("U235");
```

The following describes each of these methods:

- `ShowHalfLife`: The decay constants and half-lives of nuclides included in the chain are printed out.
- `ShowDoseCoefficient`: The dose coefficients of nuclides included in the chain are printed out.
- `ShowChainData`: The nuclide transmutation chain data are printed out. When no arguments are given, the information on which nuclides are newly generated through the neutron reactions and the decays are printed out for all nuclides contained in the chain are printed out; If users want to retreive the information of one specific nuclide, the character data or the index for the concerned nuclide should be given as the argument.
- `ShowChainDataToOrigin`: The information on which nuclide is origin of the nuclide given as the argument (Ba-137 in this example) is printed out. An example is below:

Listing 3: Example of the `ShowChainDataToOrigin` method

```
1 # Origin of Ba137 (561370)
2 #   Cs137 (551370) : Decay      0.053
3 #   Ba136 (561360) : Capture    0.972094
4 #   Ba137m (561371) : Decay      1
5 #   Ba138 (561380) : (n,2n)     0.875913
```

The numerical values at the end of each line are the brancing ratio. This example suggests that Ba-136 is converted to Ba-137 with 97.2% probability after the neutron capture reaction. The remaining 2.8% becomes Ba-137m (meta-stable Ba-137).

- `ShowNGBranchingRatio`: Branching ratio data after the (n,γ) reaction are listed.
- `ShowFPYieldForFP`: Fission product yield data for the fissile given as the argument are presented.
- `ShowFPYield`: Fission product yield data for a fission product given as the augument are presented.

When users want to retreive the specific data on the nuclide transmutation, they can use the following method in which the index of the concerned nuclide index is given as the argument.

Listing 4: Example to retreive the specific data on the nuclide transmutation

```
1  int id=942390;
2  real dc=bu.GetBurnupChain().GetDecayConstant(id);
3  real coef1=bu.GetBurnupChain().GetDoseCoefficientIngestion(id);
4  real coef2=bu.GetBurnupChain().GetDoseCoefficientInhalation(id);
5  cout<<id<<" : "<<dc<<" "<<coef1<<" "<<coef2<<"\n"; exit(0);
```

3.3 Material data input

As described below, the fuel pincell model consists of three media: the fuel pellet, the fuel cladding, and the coolant, and thus the nuclides and their number densities comprising these media should be inputted by users. In Burner, the index and the number density of the nuclides are inputted for each medium. The following is the part relevant with the material data input:

Listing 5: Material data input

```
1  // +++ fuel composition +++
2  // (UO2)
3  int nuc0=3;
4  int mat0[]={922350,922380,80160};
5  real den0[]={7.753e-4, 2.175e-2, 4.505e-2}; // 3.4% U5-enrichment
6  real temp0=968.8;
7   // (clading)
8  int nuc1=3;
9  int mat1[]={400000,260000,240000};
10 real den1[]={3.786e-2, 2.382e-4, 6.770e-5};
11 real temp1=604.;
12 // (moderator)
13 int nuc2=6;
14 int mat2[]={10010,80160,50100,280000,240000,260000};
15 real den2[]={5.572e-2, 2.786e-2, 4.592e-6, 3.688e-4, 1.609e-4, 1.306e-4};
16 real temp2=574.2;
17
18 bn.PutFuelData(nuc0,mat0,den0,temp0);
19 bn.PutCladData(nuc1,mat1,den1,temp1);
20 bn.PutModeratorData(nuc2,mat2,den2,temp2);
```

The fuel material data are assigned in line 2 to 6. In line 3, the number of nuclides comprising the medium is given in `nuc0`. In line 4, the index of nuclides in this medium is given in `mat0` which is the integer-value array. In line 5, the number densities of the nuclides are given in `den0` which is the real-value array. The order in `den0` corresponds to that in `mat0`. The unit of the number density is $10^{24}/\mathrm{cm}^3$. In line 6, the temperature of the fuel pellet region is given with the unit of K. These number density data of the fuel pellet region is inputted to the instance `bn` of the `Burner` class through the `PutFuelData` method in line 18. The number density data for the cladding and coolant regions are inputted with the same procedure.

3.4 Geometrical data input

The Burner module handles with the fuel pincell model. The fuel pin (or rod) consists of the pellet and cladding having the cicrular geometry, but the outer boundary of the surrounding coolant region is square (or hexagonal). The periodic boundary conditions are assumed to the outer boundaries. [3]

The fuel pincell model consists of three media, and the fuel pellet and coolant regions are divided into multiple spatial meshes in solving the transport equation. [4]

Figure 1 shows an example of the spatial mesh division in the square pincell model. In this example, three regions from the center are the fuel pellet region, the forth region is the cladding region, and the others are the coolant region.

The following shows the part to input the geometry information in `main.burn_tutorial.cxx`.

Listing 6: Geometry information input

```
1  // +++ Geometry Data +++
2  real pin_pitch=0.6325*2.; // [cm]
3  real rr[]={0.25,0.35,0.412,0.476,0.55,0.65,0.75};
4  int rmed[]={0,0,0,1,2,2,2};
5  bn.PutGeometryData(pin_pitch,7,rr,rmed);
```

The variable `pin_pitch` in line 2 is the pin pitch of the fuel rods; a distance between fuel rod centers adjacent to each other. In the fuel pincell model, the pin pitch is a length of the one plane of the outer square of the coolant region. Its unit is cm. In line 3, the radii of the rings mentioned above are assigned from the smaller one. In line 4, the correspondance of each ring region to the medium index is assigned from the center region. The medium index `0` is the fuel pellet, the index `1` is the cladding, and the index `2` is the coolant, respectively. In line 5, the data specified above are transfered to the instance `bn` of the `Burner` class through the `PutGeometryData` method of the `Burner` class. The second argument is the number of rings.

[3] If the fuel pincells are arranged periodically, this boundary condition is rigorous. Also it is possible to use the white reflection condition in which neutrons reaching to the boundary are reflected isotropically on the boundary. Calculations with the white reflection condition require lower computational costs than the periodic boundary, but those are approximations. If users want to switch from the periodic condition to the white condition, they can use the method `PutWhiteBoundary` of the `Burner` class.

[4] In the actual fuel rod, there is a gap between the fuel pellet region and the cladding region, but it is ignored in this example. It can be understood that the fuel region is smeared to the gap region.

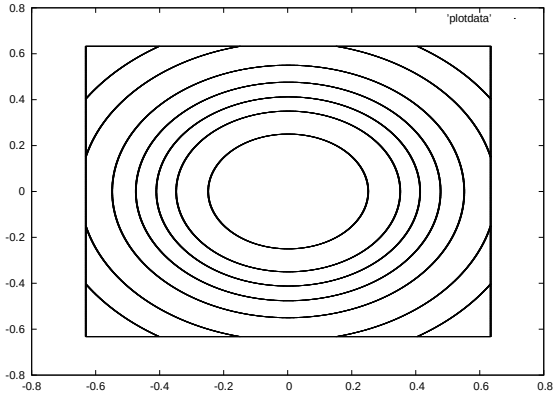

Fig. 3: Spatial mesh division in the fuel pincell model

3.5 Burnup (or irradiation) condition input

Nuclear fuel depletion (burnup) calculations require the imformation on the neutron flux level and the burnup period. Instead of the neutron flux level, the line power per 1 cm of the fuel rod is sometimes assigned. A part of the burnup condition input in `main.burn_tutorial.cxx` is shown below:

Listing 7: Burnup condition input

```
1   // +++ Burnup history data +++
2   bn.PutBurnStep(21);
3   real power_density_list[]={
4     179., 179., 179., 179., 179.,
5     179., 179., 179., 179., 179.,
6     179., 179., 179., 179., 179.,
7     179., 179., 179., 179., 179.,
8     0.
9   }; // [W/cm]
10  real burn_time[]={
11     0.1, 1.0, 2.5, 5.0, 7.5,  10., 12.5, 15., 17.5, 20.,
12     22.5, 25., 27.5, 30., 32.5,  35., 37.5, 40., 42.5, 45., // GWd/t
13     4.*365., // day
14  };
15  bn.PutPowerDensityList(power_density_list);
16  bn.PutBurnTime(burn_time,true,true); // [GWd_t/day][accumulate/not]
```

The fuel burnup calculations are carried out with the divided *steps*. In each step, neutron flux calculation is carried out, and the reaction rates for all nuclides in the fuel pin are calculated. Neutron flux spatial and energy distributions are dependent on the fuel composition, so they change with time and the fuel burnup period should be divided into steps properly. In line 2, the number of divided steps is assigned. In line 3 to line 9, the line power in each burnup step is specified with the unit of W/cm. It is also possible to specify the power as the rated power with MW/tU, and it can be realized by adding the string `''MW_t''` in the second argument of the `PutPowerDensityList` method as `PutPowerDensityList(power_density_list,''MW_t'')`. In this example, the line power in the 21th step is zero, and this means that the nuclear fuel is discharged from a reactor core during this step [5] In line 10 to line 14, the period of each step is assigned, and the unit is "GWd/t". [6] Please note that the accumulated values are given in this example. For example, a fact that 1.0 GWD/t at the 2nd step means that the burnup of 1.0 GWD/t at the end of the step 2. When the line power is zero, the unit for the step length becomes "day" and this is automatically considered by the `Burner` class. The reason why the fine burnup division is applied at the beginning of the fuel burnup is to take into account the behevior of the short-lived fission products.

In lines 15 and 16, the burnup condition information assigned above are transferred to the instance `bn` of the `Burner` class.

In the above example, the fuel burnup period of the steps is given as the fuel burnup with GWd/t, but it is also possible to be given with the day. In such cases, users should switch the second argument of the `boolean` variable to `false` in the `PutBurnTime` method of the `Burner` class. Furthermore, the length of the burnup step can be assigned

[5] During the outage period, there are not any neutron reactions but the radioactive decay, the fuel composition should change with time.
[6] This is the accumulated power per heavy nuclides weight initially loaded.

with not the cumulative value but the value of the step itself. In such case, please switch the third argument of th `boolean` variable to `false` in the `PutBurnTime` method. [7]

If the step length is constant over all burnup steps, the first argument of the `PutBurnTime` method can be defined with not the array but the single real variable.

In the fuel burnup calculations, the fuel burnup step is further divided into sub-steps. If the line power is given as the neutron flux normalization condition, the neutron flux should be normalized per every sub-step according to the line power. If the line power is constant, neutron flux level should be increased with time in the burnup step since the nuclear fuel inventory decreases with time. This can be taken into account by doing the neutron flux normalization at the sub-steps. The default number of sub-step divisions in Burner is 20, and it can be changed by using the `PutSubstep(int i)` method of the `Burner` class. The number of sub-step divisions is transferred as the argument. The concept of the burnup steps and sub steps is shown in **Fig. 4**.

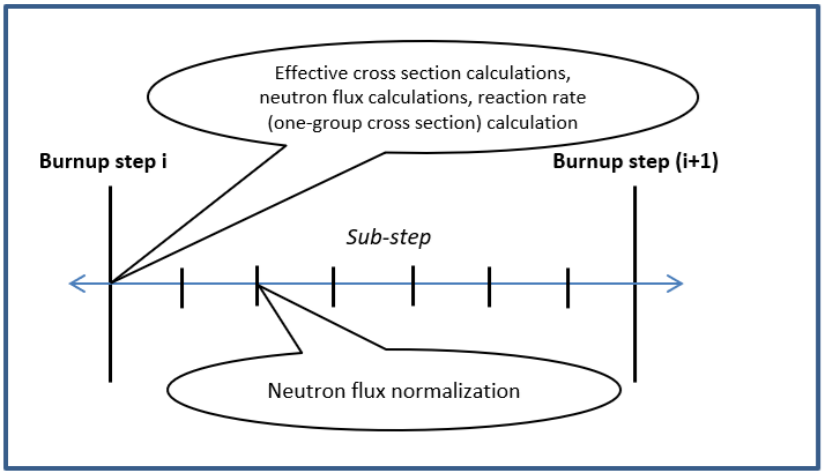

Fig. 4: Concept of burnup steps and sub steps

It is possible to use the neutron flux level itself as the neutron flux normalization condition instead of the line power. In such cases, not the `PutPowerDensityList` method but the `PutFluxLevelList` method is used. An example to assign the neutron flux level is shown below. The unit of the neutron flux level is /cm^2/s.

Listing 8: Burnup condition input with the neutron flux level assignment

```
1  // +++ Burnup history data +++
2  bn.PutBurnStep(10);
3  real flux_level_list[]={
4    5.5856e15, 0.,
5    5.5856e15, 0.,
6    5.5856e15, 0.,
7    5.5856e15, 0.,
8    5.5856e15, 0.
9  }; // [/cm2/s]
10 real burn_time[]={
11   148, 60,
12   148, 60,
13   148, 60,
14   148, 60,
15   148, 4*365
16 }; // [day]
17 bn.PutFluxLevelList(flux_level_list);
18 bn.PutBurnTime(burn_time,false,false); // [GWd_t/day][accumulate/not]
```

[7] If users want to do calculations with iterative burnup and coolant, please be careful that the coolant period should not be given with the cumulative vales and that the coolant period is not included in the cumulative burnup value.

4 Execution of fuel burnup calculation and output of calculation results

4.1 Execution of fuel burnup calculation

With the procedure above, all required information are transferred to the instance `bn`, so now it is ready to execute the burnup calculation.

A part of the execution of the calculation in `main.burn_tutorial.cxx` is shown below:

Listing 9: Execution of burnup calculation

```
1  bn.Calculation(bu);
```

With this method, a burnup calculation is carried out and results are stored in the instance `bn`.

4.2 Output of the results on the screen

To access the calculation rsults, users have to do relevant methods of the instance `bn` after executing the calculation. A part of outputing the results in `main.burn_tutorial.cxx` is shown below:

Listing 10: Output of the results on the screen

```
1  bn.ShowEigenvalue();
2  bn.ShowNeutronFluxHistory();
3  //bn.ShowNuclideList();
4
5  bn.ShowNumberDensityChange();
6  string nuc_name1[]={"Xe135","Xe137"};
7  bn.ShowNumberDensityChange(2,nuc_name1);
8
9  string prt_nuc_name[]={"Mo098","Mo099","Mo100"};
10 string prt_nuc_name2[]={"FP","HM","ALL"};
11 bn.ShowNumberDensityHistory(3,prt_nuc_name,bu,"nd");
12 bn.ShowNumberDensityHistoryAtomWise("Mo",bu,"nd");
13
14 //bn.ShowHeavyMetalWeightRatio(bu);
15 //bn.ShowCrossSection(3,prt_nuc_name);
16 //bn.ShowCrossSection(0); // for burnup step 0
17 //bn.ShowFuelNeutronFlux(0);
18 //bn.ShowFuelNeutronFluxExcel(0);
19 int prt_nuc2=5;
20 string prt_nuc_nam2[]={
21   "U235","U238","Pu239","Pu240","Pu241"
22 };
23 //bn.ShowNuclideWiseContributionForFission(prt_nuc2,prt_nuc_nam2);
24 //bn.ShowCaptureToDecayRatio(prt_nuc2,prt_nuc_nam2,bu);
25 //bn.ShowGroupDependentCrossSection("U235");
26 //bn.ShowGroupDependentCrossSectionExcel("U235");
```

In this example, the following methods are used to retreive the results:

- `ShowEigenvalue`: The infinite neutron multiplication factor, the conversion ratio, and the neutron flux level at the beginning of the first sub-step at every fuel burnup step are presented.
- `ShowNeutronFluxHistory`: The burnup step-wise neutron flux levels at the cladding and coolant regions are presented.
- `ShowNuclideList`: A list of nuclides in the fuel pellet region is presented.
- `ShowNumberDensityChange`: The number densities of nuclides before and after the fuel burnup are presented. Not that nuclides having zero number denzities are not presented. If the real argument is given, it presents the nuclide information whose number densities at the beginning or end of the burnup period are larger than this value. If the number of nuclides and a list of these nuclides are given in the arguments, the information only on these nuclides are presented.
- `ShowNumberDensityHistory`: Various data on the specific nuclides can be presented. The first argument is the number of the specfic nuclides, and the list of these nuclides is given as the second argument. In the list, nuclide can be specified with its name as "`Xe135`", and also element-wise input such as "`Xe`" is possible. Users can also use "`FP`", "`HM`", and "`ALL`" which are all fission products, all heavy nuclides, and all nuclides. [8] Uses can specify the type of output data with the keyword in the forth argument. The following is a list of the keyword and the correspoinding data type.

[8] `HM` is nuclide whose atomic number is larger than 80.

Keyword	Output data
nd	Nuclide number density over all volume [10^{24}]
nd_per_vol	Nuclide number density per unit volume [10^{24}/cm^3]
bq	Radioactivity over all volume [Bq]
bq_per_thm	Radioactivity per unit fuel mass [Bq/tHM]
kg_per_thm	Weight per unit fuel mass [kg/tHM]
w_per_thm	Decay heat per unit fuel mass [W/tHM]
w_gamma_per_thm	Decay heat per unit fuel mass (gammma-component) [W/tHM]
w_beta_per_thm	Decay heat per unit fuel mass (beta-component) [W/tHM]
sv_per_thm_ing	Ingested absorbed dose per unit fuel mass [Sv/tHM]
sv_per_thm_inh	Inhalated absorbed dose per unit fuel mass [Sv/tHM]

In the output yielded by this method, the value is given at the final column, and this corresponds to the sum of the values in all columns. If the fifth argument is `true`, only the sum is presented.

- `ShowNumberDensityHistoryAtomWise`: By assigning the name of the element, the results of all isotopes of this element are presented. The data type to be outputted is given in the third argument. The keyword is the same as the `ShowNumberDensityHistory` method. If users set the forth argument as `true`, only the sum is presented.
- `ShowHeavyMetalWeightRatio`: The weight of heavy nuclides before and after the burnup is presented.
- `ShowCrossSection`: One-group cross section in the fuel region is presented. If the number of the outputted nuclides and a list of these nuclides, the one-group cross sections dependent on the fuel burnup of these nuclides are presented. If only one integer value is given in the argument, the one-group cross sections of all nuclides at the burnup step corresponding to the argument are presented. If the third argument is `true`, not the one-group cross sections but the reaction rates are presented. If the forth argument in the integer value is given, users can change the number of precisions in the output.
- `ShowFuelNeutronFlux`: The neutron flux enregy spectrum of the fuel pellet region is presented. The concerned burnup step is assigned in the argument This example uses "`0`", so the neutron flux energy spectrum at the beginning of the fuel burnup is presented. When runnning this example, 107 pairs of vales which are energy group-wise neutron flux are presented, and the left value is the upper energy of the energy group and the right value is the neutron flux. Since neutron flux data are divided by the lethargy width of the corresponding energy group, $\ln(E_0/E)$, the X-axis should be the neutron lethargy or log of energy when plotting these data.
- `ShowFuelNeutronFluxExcel`: This is almost same as the `ShowFuelNeutronFlux` method, but it is suited to plotting with the histgram-type graph in Excel.
- `ShowNuclideWiseContributionForFission`: Nuclide-wise contributions to the total fission reactions in every burnup step are presented. Concerned nuclides are speficied in the augument, and the sum of the contributions is presented with the parenthesis.
- `ShowCaptureToDecayRatio`: Ratios of neutron capture reaction rates to decay constant are presented. [9]
- `ShowGroupDependentCrossSection`: Energy group-wise cross sections are presented at the end of the burnup for the specified nuclide.
- `ShowGroupDependentCrossSectionExcel`: This is the almost same with `ShowGroupDependentCrossSection`, but the output format is suited to the editting with Excel.

4.3 Storing calculation results in external files

Generally, calculation results by Burner are printed out on the screen, but it is convenient to store them in external files if users want to use calculation results with various cases such as uncertainty quantification of nuclide inventories after furl burnup using the random sampling method. For such cases, a method `WriteFileNumberDensity` is implemented. With this method, the infinite neutron multiplication factor and the nuclide number densities at the beginning of the specified burnup step can be stored in an external file whose name can be also assigned by users. An exampe to use the `WriteFileNumberDensity` method is shown below:

[9] On the number density of one nuclide, the decreasing rate by neutron capture is written as $N\sigma\phi$ and the decreasing rate by decay is as λN. This method outputs $\sigma\phi/\lambda$, so it provides the information of the contribution of neutron capture and decay to the decrease of the concerned nuclide.

Listing 11: Storing calculation results in an external file

```
1  int cyc_num=2;
2  int cyc[]={0,20};
3  bn.WriteFileNumberDensity(cyc_num,cyc,"./","out_nd",10);
```

The number of burnp steps to be outputted is assigned by `cyc_num` in line 2, and the burnup steps are assigned with `cyc` in line 2. Here "`0`" corresponds to the first step, and "`20`" does to the final step if the number of the user-specified burnup steps is 20. The third and forth arguments in the `WriteFileNumberDensity` method are the directory name where the external file is stored and the file name, respectively. The forth argument is the precision of the prented values.

The burnup step, the infinite neutron multiplication factor, and the number densities of the nuclides in the fuel pellet region are stored in the file. The nuclide index corresponding to the order in this file can be referred by doing the `ShowNumberDensityChange` method with the argument of `-1`.

5 Exercise

Please prepare the input data and perform the burnup calculations for the following condition:

- Pin pitch: 1.265 cm
- Radius of the fuel pellet: 0.412 cm
- Outer radius of the cladding region: 0.476 cm
- Temperatures of fuel, cladding, and coolent: 968.8 / 604.0 / 574.2 K
- Line power: 179 W/cm
- Burnup period: Burnup until 45 GWd/t and 4-year cooling

Please calculate two types of nuclear fuel; the UO_2 fuel and the MOX fuel. The U-235 enrichments of the former and latter cases are 4.1 wt% and 0.2 wt%, respectively, and the Pu content in the MOX fuel is 10.0 wt%. The number densities of the fuel pellet are shown in **Table 1**, and those of the cladding and coolant regions are in **Table 2**. "Index" in these tables are nuclide index used in CBZ. [10]

Table 1: Initial fuel composition

Nuclide	Index	Number density (10^{24}/cm^3)	
		UO_2	MOX
^{235}U	922350	9.349E-4	4.104E-5
^{238}U	922380	2.159E-2	2.022E-2
^{238}Pu	942380		4.731E-5
^{239}Pu	942390		1.223E-3
^{240}Pu	942400		5.586E-4
^{241}Pu	942410		2.069E-4
^{242}Pu	942420		1.418E-4
^{241}Am	952410		6.007E-5
^{16}O	80160	4.505E-2	4.500E-2

Table 2: Number densities of clading and moderator regions

Region	Nuclide	Index	Number density (10^{24}/cm^3)
Clading	Zr	400000	3.786E-2
	Fe	260000	2.382E-4
	Cr	240000	6.770E-5
Moderator	^{1}H	10010	5.572E-2
	^{16}O	80160	2.786E-2
	^{10}B	50100	4.592E-6
	Ni	280000	3.688E-4
	Cr	240000	1.609E-4
	Fe	260000	1.306E-4

Based on the results obtained by the fuel burnup calculations, please answer to the following questions. The nuclide transmutation chain in the SRAC manual would be mondatory. If you need more information, please let Chiba know.

Ex. 1: Plot and compare the infinite multiplication factor with the fuel burnup for the UO_2 and MOX fuels. Also, by using the `ShowFuelNeutronFlux` method, plot the energy spectra of the neutron flux in the fuel region of the UO_2 and MOX fuels. Please prepare two figures with the linear and log scales on the Y-axis.

[10] The rule of the nuclide index in CBZ is given as $Z \times 10000 + A \times 10 + L$ where Z is the atomic number, A is the mass number, and L is the excitation level. Generally, most of nuclides are in ground state so L is zero, but some nuclides are in meta-stable and L is larger than zero like Am-242m. For the nuclide having the natural abundance, the mass number is assumed zero, so the index of iron in the natural abundance is defined as 260000.

Ex. 2: Plot the number densities of U-235, -238, Pu-238, Pu-239, -240, -241, -242, and Am-241 with the fuel burnup for the UO_2 and MOX fuels, and briefly explain the trends.

Ex. 3: Plot the nuclide-wise contribution to the fission reaction with the fuel burnup for the UO_2 and MOX fuels by using the `ShowNuclideWiseContributionForFission` method of the `Burner` class.

Ex. 4: Plot the number densities of Cs-133, -134, and -137 with the fuel burnup for the UO_2 fuel. Also please try to fit them by the polynomial functions, and estimate how many orders in the polynomial are required for each. If there is a difference among these three, please discuss the reason.

Ex. 5: Xe-135 is strong neutron absorbing nuclide and is mainly generated by the beta decay of I-135. Plot the number densities of Xe-135 and I-135 with the burnup for the UO_2 fuel. The trend of them are very different from those of the Cs isotopes, and please explain the reason.

Ex. 6: Discuss the generaion mechanism of Pu-238 for the UO_2 and MOX fuels.

In Ex. 1, it can be easily found that there is a significant difference in the neutron flux energy spectra between the UO_2 and MOX fuels. This can be understood by observing the neutron absorption cross sections and the text which Chiba prepared before would be helpful. If necessary, please let Chiba know.

In Ex. 6, one method to investigate the mechanism of the nuclide generation is to cut the nuclide transmutation chain. If you feel one tranmutation path is important for the generation of the concenred nuclide, you can confirm it by cutting this path and doing the calculations. To cut the chain, you can use the `CutChainForDecay`, `CutChainForCapture`, and `CutChainForN2N` methods in the `BurnupChain` class. The argument for these methods should be the character-type variable, so please specify the nuclide name to cut the chain. For example, if you want to cut the decay path from Pu-241, you can do `bu.GetBurnupChain().CutChainForDecay("Pu241")`. This should be done before conducting the calculation with `bn.Calculation(bu)`.

Another way is to make the fission yield of the specific nuclide zero. To do this, please use the `SetZeroFissionYield` method in the `BurnupChain` class. This method requires two character-type arguments, and the first and second are the fissile and the fission product, respectively. If you want to make the Xe-135 fission yield from the U-235 fission zero, you can do `bu.GetBurnupChain().SetZeroFissionYield("U235","Xe135")`.