# CBZ/BurnupChainGeneratorの使用マニュアル

千葉豪

2021年1月15日

## 1 はじめに

原子炉の炉心では、炉心内をとびかう中性子が媒質に含まれる原子核と相互作用を起こすことによって異なる核種に変換される。中性子と反応してガンマ線を放出する (n,g) 反応（正確には (n,$\gamma$) と記載すべきであるが簡略化して記述）では反応後の原子核の質量数が一つ増え、中性子と反応してふたつの中性子を放出する (n,2n) 反応では反応後の原子核の質量数が一つ減る。例えば、U-235 が (n,g) 反応した場合には U-236 になり、(n,2n) 反応した場合には U-234 となる。また、原子核の大部分は不安定であり、固有の半減期で崩壊し、異なる原子核となる。崩壊の様式には、$\alpha$ 崩壊、$\beta$ 崩壊等、様々あり、それらによって崩壊後の原子核が規定される。例えば、Cm-242 が $\alpha$ 崩壊した場合、陽子数が 2、中性子数が 2 小さくなるため、崩壊後は Pu-238 に変換される。

このように、中性子と原子核との反応、また崩壊に伴う原子核の変換を記述するものが「燃焼チェーン」である。

ウラン、プルトニウムといった重核種 heavy nuclides の数はそれほど多くはない（30 程度）が、核分裂により発生する核分裂生成物（fission product, FP）核種は、その数が 1,000 を優に超える。中性子束の空間、エネルギー依存性を計算において直接考慮しない、いわゆる一点炉計算では、それら全ての核種を取り扱った燃焼計算を行うことは比較的容易であり、そのようなコードとしては米国オークリッジ国立研究所（ORNL）が開発している ORIGEN 等が有名である。このような計算では、対象として想定した原子炉体系の中性子束エネルギースペクトルから一群断面積セットを予め計算しておき、それらを燃焼計算で用いる。また、中性子束エネルギースペクトルが燃焼に伴い変動していく効果を考慮するため、重要核種の一群断面積は燃焼度に応じたテーブル形式で与えられる。ORIGEN コードには、軽水炉の $UO_2$ 燃料、MOX 燃料、また高速炉 MOX 燃料など、多様な原子炉の型、燃料種類に対する一群断面積セットが用意されており、ユーザは計算対象に応じて適切な一群断面積セットを選択し、使用することになる。従って、ユーザがこれから計算しようとしている体系と、使用する一群断面積セットがその適用を想定している体系との差異は、そのまま計算誤差となる。

一方、CBZ/Burner のように、計算対象、燃焼の進展に伴う中性子束エネルギースペクトルの変動を考慮する場合には、燃焼の進み具合に応じて一群断面積を計算するため、上記のような計算誤差の懸念は無い。ただし、取り扱う核種の多群データ（例えば Burner で軽水炉系を計算する場合は 107 群）を保持する必要があるため、取り扱う核種数に比例して必要となる計算負荷、容量が増大する。このような場合には、取り扱う核種を限定するのが一般的であり、特に短寿命の FP 核種が燃焼チェーンから除去されることになる。

例えば、FP 核種として A、B を考え、それぞれの核分裂収率を $y_A$、$y_B$ とする。核種 A が 1[ms] の半減期で崩壊し核種 B になるとした場合、見ている時間スケールが「day」オーダーであると

すると、核分裂して生成される核種 A は「瞬時に」核種 B に崩壊する、すなわち、核分裂により「瞬時に」核種 B が生成される、と考えることが出来る。この場合、核種 A と核種 B の収率はそれぞれ 0、$y_A + y_B$ となり、核種 A は燃焼チェーンから除去されることになる。

Burner ではいくつかの燃焼チェーンを使うことが出来るが、FP として 197 核種を考慮するものが一般的に使われるものの中で最も詳細なチェーンとなっている [1]。このチェーンは SRAC-2006 の「開発者用チェーン」と呼ばれるものと殆ど同一と考えてよい。このチェーンを用いた場合、燃焼後 1 年経過以降は崩壊熱や放射能などの計算を比較的精度良く計算することができる（そのように燃焼チェーンが作成されているとも言える）。ただし、燃焼後の比較的短い時間においては、燃焼チェーンにおいて短半減期の FP が無視されているため、それらの崩壊熱や放射能などを正確に評価することができない（すなわち過小評価となる）。

CBZ には、燃焼チェーンにおいて考慮する核種を指定すれば、自動的に燃焼チェーンを構築する機能を有する BurnupChainGenerator モジュールが実装されており、197 核種以上の FP を扱うチェーンも比較的容易に作成することが出来る。このメモでは、BurnupChainGenerator の使用法について述べる。

[1] 原子炉停止直後の崩壊熱を計算する場合などに用いる詳細なチェーンもある。

# 2 燃焼計算に関連する核データ

燃焼チェーンには、(n,g)、(n,2n) といった中性子核反応、また崩壊によって核種がどのように変わっていくかの情報が含まれる。それらのもとになる核データとしては以下のものが挙げられる。

- 崩壊データ（decay data）：核種がどのような様式（$\alpha$ 崩壊、$\beta$ 崩壊、等）で崩壊するかという情報。半減期も含まれる。核種によってはいくつかの異なる様式で崩壊する場合があるため、分岐比 branching ratio が与えられている。これらのデータは一般的に「Decay data file」に収納されており、代表的なものとしては、JENDL FP Decay Data File 2011(JENDL/FPD-2011)[1]、ENDF/B-VII.1、JEFF-3.1 が挙げられる。

- 核分裂収率データ（fission yield data）：核分裂性核種が核分裂した際に生成される FP 核種の生成率に関する情報。いくつかの中性子入射エネルギーに対して与えられる（高速中性子による核分裂と熱中性子による核分裂とでは生成される FP 核種の分布が微妙に異なるため）。これらのデータは一般的に「Fission yield data file」に収納されており、代表的なものとしては、JENDL FP Fission Yields Data File 2011(JENDL/FPY-2011)[1]、ENDF/B-VII.1、JEFF-3.1 が挙げられる。

- 中性子核反応の分岐比データ：(n,g) や (n,2n) 反応の後に準安定状態 meta-stable の核種が生成される場合がある。基底状態 ground と meta-stable の核種の生成比率が中性子核反応の分岐比 branching ratio データである。分岐比データは評価済み核データファイルに入射中性子エネルギー依存で与えられるが、汎用の核データファイルには重要な核種（例えば Am-241）に対してのみ与えられている。多くの FP 核種に分岐比データが与えられているファイルとしては、JEFF-3.1 activation file（JEFF-3.1A）がある。また、Japan Nuclear Data Comittee が評価した「JNDC Nuclear Data Library of Fission Products - Second Version -」（通称 JNDC Ver.2）にも、入射中性子エネルギーに依存しない分岐比データが与えられている。

燃焼チェーンを構築するためにはこれらのデータファイルから情報を取り出す必要がある。また、核種を限定したチェーンを構築する場合には、チェーンから除去する核種の崩壊を考慮したチェーンを再構築する作業が必要となる。CBZ には、燃焼チェーンにおいて考慮する核種を指定すれば、自動的に燃焼チェーンを構築する機能を有する BurnupChainGenerator モジュールが実装されている。

また、燃焼計算では、燃焼チェーンに加えて、以下の情報が用いられる。

- 発生熱データ：燃焼計算において熱出力を指定する場合、体系の中性子束の大きさはその熱出力の値によって規格化される。一般的な燃焼計算では、核分裂反応あたり、中性子捕獲反応あたりの発熱率をそれぞれ定義し、反応率にその発熱率を乗じ、空間積分をとることで体系としての総発熱量を計算する。Burner では SRAC-2006 と同様に、全ての熱が核分裂反応で発生すると見做し、核分裂反応のみに対して発熱率を定義している。

- 原子質量データ：燃焼計算では燃焼度の指標として GWd/tHM がよく用いられる。Burner では初期燃料組成を数密度で与えるため、燃料の初期重量を計算する際などに原子質量データが必要となる。

# 3 CBZ/BurnupChainGenerator の利用例

## 3.1 BurnupChainGenerator へのデータの入力

CBZ の BurnupChainGenerator は、個々の核種の崩壊データ、核分裂収率データを保持する BCGNuclide クラスと、BCGNuclide クラスのインスタンス群を保持し、燃焼チェーンの作成などを行う BCGManager クラスからなる。ここでは、BCGManager クラスのインスタンスへの、崩壊データ、核分裂収率データの入力方法と、入力されたデータを取り出す（画面に出力させる）方法について説明する。

**Listing 1** に BCGManager のインスタンスにデータを取り込む例を示す。1 行目では BCGManager クラスのインスタンス man を生成している。また、4 行目では Decay Data を、7 行目では Branching ratio Data を、11、12 行目では Fission yield データを、それぞれ外部ファイルから読み込んでいる。なお、Fission yield データを読み込むメソッド ReadFPYieldDataFromFile であるが、二つ目の引数では読み込む入射核分裂エネルギーを指定する。核分裂収率は核分裂を起こした中性子のエネルギーに依存するため、それらのデータはファイルにおいてはいくつかの入射エネルギー点に対して与えられている。この引数は、そのうち何番目の入射エネルギー点のものを読み込むかを指定している。この例ではゼロが与えられているため、一番低い入射エネルギー（一般的には熱中性子核分裂）のデータが取り込まれている。また、三つ目の引数では、そのメソッドで格納した核分裂収率データを識別するための名前を与えている。この例では、ファイル「fy.u5.j33」から読み込まれたデータが「u235」という名前で識別され、インスタンス「man」に格納されることになる。その他については、後ほど詳述する [2] 。

Listing 1: BCGManager へのデータの取り込み

```
1  BCGManager man;
2
3  // +++ Decay Data input
4  man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.endf71");
5
6  // +++ brancing ratio input
7  man.ReadNGBranchingRatioDataFromFile("./BR_DATA/ng_br_jndc_v2");
8  // JNDC Ver.2 for thermal reactor
9
10 // +++ Fission yield data input
11 man.ReadFPYieldDataFromFile("../../CBGLIB/FPYfile/fy.u5.j33",0,"u235");
12 man.ReadFPYieldDataFromFile("../../CBGLIB/FPYfile/fy.p9.j33",0,"pu239");
```

次に、各種データを格納した BCGManager クラスのインスタンスからデータを取り出す方法について説明する。サンプルを **Listing 2** に示す。

Listing 2: BCGManager に格納されているデータの取り出し

```
1  // +++ For showing information on screen
2  //man.ShowNuclideList(); // Nuclides included BCGManager are shown.
3  //man.ShowYieldData("u235",false); // Fission yield data are shown.
4
5  // +++ For plotting
6  //man.ShowYieldDataForXYPlot("u235"); // Fission yield data are shown.
7  //man.ShowDNDataForXYPlot(); // Delayed neutron emission per a decay
```

以下、それぞれのメソッドについて説明する。

[2] BurnupChainGenerator が読み込み可能な崩壊データ、核分裂収率データは、木綿豆腐コードが論理機番 7 に出力するテキストデータに対応している。

- ShowNuclideList：格納されている核種データの一覧が崩壊チャンネル数、半減期とともに示される。
- ShowYieldData：一つ目の引数で与えられた名前で識別される収率データが示される（二つ目の引数である Boolean 変数については、ここでは説明を省略する）。
- ShowYieldDataForXYPlot：引数で与えられた名前で識別される収率データが gnuplot に使える形式で出力される。
- ShowDNDataForXYPlot：崩壊あたりに発生する平均中性子数が gnuplot に使える形式で出力される。

ShowYieldDataForXYPlot により取り出したデータファイルから gnuplot を用いて作成したプロット例を **Fig. 1** に示す。また、gnuplot のシェルスクリプトのサンプルを**リスト 3** に示す。

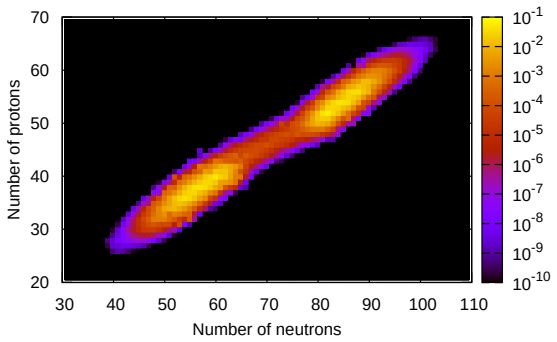


Fig. 1: 核分裂収率データのプロット例

Listing 3: gnuplot による核分裂収率データのプロットのためのシェルスクリプト

```
#!/bin/sh

gnuplot -persist <<EOF
set size 0.7,0.7
set pm3d map
set xrange[30:110]
set yrange[20:70]
set xlabel 'Number of neutrons'
set ylabel 'Number of protons'
set logscale cb
set cbrange[1e-10:0.1]
set format cb "10^{%L}"
set terminal postscript eps enhanced color
set output 'y_p9ff.eps'
splot "y_p9ff"
EOF
```

### 3.2 燃焼チェーンの作成

ここでは CBZ の BurnupChainGenerator を用いた燃焼チェーンの作成方法について述べる。

BurnupChainGenerator を利用したチェーン作成プログラム（main.make_cbg_chain.cxx）の最初の部分を以下に示す。

Listing 4: 燃焼チェーン作成ツール (1)

```
1  string filename_yield="./cbgdat_yield";
2  string filename_chain="./cbgdat_chain";
3
4  real half_life_limit=0; // half-life upper limit [s] (0:no limit)
5  //real half_life_limit=10*24*60*60; // half-life upper limit [s]
6
7  bool n2n_ignor=true; // ignoring (n,2n) reaction in FP burnup chain
```

このプログラムは、FP の核分裂収率データと FP のチェーンデータをそれぞれ異なる形式のテキストファイルに出力する。1、2 行目では、それぞれの出力ファイル名を指定する。この例では、このプログラムを走らせたディレクトリ上に、これらのデータファイルが生成されることになる。

燃焼チェーンを作成する際、チェーン上で考慮するように指定された核種以外はチェーンから無視されることになる。無視される核種の崩壊は一般的に「バイパス」されて、陰的に考慮されることになる [3] が、無視される核種の半減期が着目する時間に対して非常に長い場合には、崩壊チェーンをバイパスする（核種 A が崩壊により核種 C になるとする）ことは適切ではない。そこで、BurnupChainGenerator では崩壊チェーンをバイパスする半減期の上限値を指定することが出来る。すなわち、半減期がこの上限値を越える核種を無視する場合は、その崩壊を考慮しないこととなる。上の例では 4 行目に対応しているが、ここではゼロが指定されているため、上限を設定しないことになっており、全ての無視した核種の崩壊が考慮されることになる。

また、FP のチェーンにおいて (n,2n) 反応はそれほど重要ではないため、SRAC-2006 の燃焼チェーンでは無視されている（反応自体は考慮されるが、それによる核種の生成は考慮されていない）。CBZ/BurnupChainGenerator では、(n,2n) 反応の取扱いをユーザが指定することが出来、当該反応を無視するか、考慮するかを上の例の 7 行目の boolean 変数「n2n_ignor」で指定している。

次に、崩壊データ、分岐比データを指定する部分について示す。

Listing 5: 燃焼チェーン作成ツール (2)

```
1  // +++ Decay Data input
2  man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.endf71",true);
3  // When jendl library is used, endf data should be read in advance
4  // because the jendl library does NOT include stable nuclide data.
5  man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.jendl2000");
6  //man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.jendl2011");
7  //man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.jeff311");
8  //man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.endf71");
9
10 // +++ brancing ratio input
11 //man.ReadBranchingRatioDataFromFile("./BR_DATA/br_jeff31a_uo2");
12 //man.ReadBranchingRatioDataFromFile("./BR_DATA/br_origen_pwru");
13 man.ReadNGBranchingRatioDataFromFile("./BR_DATA/ng_br_jndc_v2");
14       // JNDC Ver.2 for thermal reactor
15 //man.SetNGBranchingRatioDataForFR(); // fast reactor
```

[3] A→B→C という崩壊の流れがあるときに核種 B をチェーンにおいて無視する場合、A→C という「バイパスされた」チェーンが構築される。

崩壊データとしては、ENDF/B-VII.1、JENDL/FPD-2000、JENDL/FPD-2011、JEFF-3.1.1 が指定できる。なお、JENDL を指定する場合には、JENDL には安定核のデータが含まれていないことから、予め 2 行目のように安定核を含むファイルを読み込んでおく必要がある（ReadDecayDataFromFile メソッドの最後の boolean 型の引数が true の場合には安定核のデータのみが抽出される）。

分岐比データとしては、軽水炉解析用のチェーンについては、JNDC Ver.2 のデータ、JEFF-3.1A（JEFF-3.1 Activation file）、ORIGEN-2.2 のデータを使うことが出来る。なお、JEFF-3.1A のデータについては、(n,g) 反応に加えて (n,2n) 反応の分岐比も与えられている。また、JEFF-3.1A の分岐比は入射エネルギー依存で与えられているため、ここでは UO2、MOX 燃料セルの中性子束エネルギースペクトルを用いて平均化した値を用いている。**ここで注意すべきなのが、CBZ の ChainGenerator は一般的に FP 核種のチェーンを作成することに用いられる点である。重核種のチェーンは CBZ の BurnupChain クラスに内蔵されているデータが用いられ、そこでは分岐比（例えば Am-241(n,g) 等) は SRAC-2K6 等のデータを用いている。**

核分裂収率データの読み込みについては前述の通りである。

最後に燃焼チェーンを作成する部分について示す。

Listing 6: 燃焼チェーン作成ツール (3)

```
1  int num=49;
2  string nuc_nam[]={
3    "Kr083","Zr095","Nb095","Mo095","Tc099", "Ru101","Ru103","Rh103","Rh105","Pd105",
4    "Pd107","Pd108","Ag107","Ag109","I135",  "Xe131","Xe133","Xe135","Cs133","Cs134",
5    "Cs135","Cs137","Ba140","La140","Pr143", "Nd143","Nd145","Nd147","Nd148","Pm147",
6    "Pm148m","Pm148","Pm149","Sm147","Sm148", "Sm149","Sm150","Sm151","Sm152","Eu153",
7    "Eu154","Eu155","Eu156","Gd154","Gd155", "Gd156","Gd157","Gd158","Gd160",
8  };
9
10 man.MakeFlagTrue(num,nuc_nam);
11 man.CalCumulativeYield(half_life_limit);
12 man.CalDecayBranch(half_life_limit);
13 man.CalReactionBranch();
14
15 man.ShowYieldSum();
16
17 // +++ Write down the data on file with CBG format
18 man.WriteFileYieldDataCBGFormat(num,nuc_nam,filename_yield);
19 man.WriteFileChainDataCBGFormat(num,nuc_nam,filename_chain,n2n_ignor);
```

1 行目の int 型変数「num」は燃焼チェーンにおいて考慮する FP の核種数に対応し、その後の 2 行目から 8 行目までは考慮する FP の核種名に対応している。10 行目から 13 行目は BurnupChainGenerator 内でチェーンを「再構築」する作業に対応し、18、19 行目において、作成した燃焼チェーン、累積核分裂収率データを指定した名前のファイルに書き出している。

核種毎の崩壊、収率データを保持する BCGNuclide クラスは boolean 型変数 flag をメンバーとして保持しており、燃焼チェーンの作成ではこれが利用される。すなわち、燃焼チェーンを作成する際に、考慮すべき核種については flag を「true」に、そうでない核種については「false」に設定し、最終的に flag が「true」のもののみを抽出する、等が行われている。この例では、10 行目の BCGManager クラスの「MakeFlagTrue」メソッドにより、string 型配列変数 nuc_nam に指定された 49 核種について、flag を「true」に設定している。

### 3.3 燃焼チェーンで考慮すべき核種を選別するアルゴリズム

前節では、指定した核種で構成される燃焼チェーンを作成する方法について述べたが、考慮すべき核種を抽出することも重要である。

以下には、短半減期核種の崩壊熱を精度良く計算するために必要となる核種を抽出するプログラムの例を示す。9 行目の BCGManager のメソッド SearchShortHalflivedNuclide では、燃焼チェーンで無視する核種について flag を true にする。まずは、一つ目の引数で与えられている値よりも大きな値を半減期に持つ核種については必ず flag を false にする（すなわち必ず考慮する）。また、それらの核種のチェーン上流にある核種についても flag を false にし、考慮させる。ただし、二つ目の引数で与えられている値よりも小さな値を半減期に持つ核種については必ず flag を true にする（必ず無視する）。これは、半減期があまりに短いものを燃焼チェーンに含めると、計算負荷が大きくなってしまうためである。この例では 50 時間以下の半減期を持つ核種とその親核種のうち、半減期が 1 秒以上のものが抽出されている [4]。17、18 行目は、flag が false、すなわち燃料チェーンから「落とさない」核種の ENDF-ID と名前が一覧で出力させる。これを用いることにより、前節のツールにより、燃焼チェーンを作成することが出来る。

Listing 7: 短半減期核種の崩壊熱を計算するための核種の抽出アルゴリズム

```
1  BCGManager man;
2
3  // +++ Decay Data input
4  man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.endf71",true);
5  man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.jendl2000");
6
7  man.ReadFPYieldDataFromFile("../../CBGLIB/FPYfile/fy.u5.j33",0,"u235");
8
9  man.SearchShortHalflivedNuclide(50*60*60,1);
10
11 man.MakeFlagTrue(68,145,0);
12 //  *Note*
13 //   Er145 is stable nuclide in ENDF/B-VII.1 decay data,
14 //   but ENDF-ID for this nuclide cannot given under the format rule.
15 //   So, this nuclide is omitted from burnup chain
16
17 man.ShowFlagedNuclideMatID(false);
18 man.ShowFlagedNuclideName(false);
19
20 man.ShowFlagedNuclideShortHalflife(1.,false);
```

一方、以下のサンプルは特定の元素のインベントリを計算するために必要となる核種を抽出するプログラムである。13 行目のメソッド SearchShortHalflivedNuclideForSpecificNuclides は、前述の SearchShortHalfLivedNuclide メソッドに対して、3、4 つ目の引数で指定される元素とその親核種のみを考慮させるというアルゴリズムを加えたものである。この例では Cs 元素とその親核種のうち、1、2 つ目の引数で指定した半減期をもつものが抽出（flag が false に設定）される。また、41 行目では、原子炉計算で考慮すべき 193 核種について flag を false にして、チェーンから落とされないような処理を施している。

Listing 8: 特定の元素のインベントリを計算するための核種の抽出アルゴリズム

```
1  BCGManager man;
2
3  // +++ Decay Data input
4  man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.endf71",true);
5  man.ReadDecayDataFromFile("../../CBGLIB/DDfile/dd.jendl2000");
```

[4] 11 行目の「MakeFlatTrue メソッドは、ENDF/B-VII.1 崩壊データファイルに与えられていない安定核 Eu-145 もチェーンに組み込むために行っているものであるが、ここでは特に理解する必要は無い。

```
6
7   man.ReadFPYieldDataFromFile("../../CBGLIB/FPYfile/fy.u5.j33",0,"u235");
8
9   // +++
10
11  int nucnum=1;
12  int nucid[]={55}; // Cs
13  man.SearchShortHalflivedNuclideForSpecificNuclides(1.,0.,nucnum,nucid);
14
15  // +++
16
17  int num_include_193chain=193;
18  string nucname_include_193chain[]={
19      "Ge073","Ge074","Ge076","As075","Se076","Se077","Se078","Se079","Se080","Se082",
20      "Br081","Kr082","Kr083","Kr084","Kr085","Kr086","Rb085","Rb086","Rb087","Sr086",
21      "Sr087","Sr088","Sr089","Sr090","Y089","Y090","Y091","Zr090","Zr091","Zr092",
22      "Zr093","Zr094","Zr095","Zr096","Nb093","Nb093m","Nb094","Nb095","Mo092","Mo094",
23      "Mo095","Mo096","Mo097","Mo098","Mo099","Mo100","Tc099","Ru100","Ru101","Ru102",
24      "Ru103","Ru104","Ru105","Ru106","Rh103","Rh105","Rh106","Pd104","Pd105","Pd106",
25      "Pd107","Pd108","Pd110","Ag107","Ag109","Ag110m","Cd110","Cd111","Cd112","Cd113",
26      "Cd113m","Cd114","Cd116","In113","In115","Sn116","Sn117","Sn118","Sn119","Sn119m",
27      "Sn120","Sn121","Sn121m","Sn122","Sn123","Sn124","Sn126","Sb121","Sb123","Sb124",
28      "Sb125","Sb126","Sb126m","Te122","Te123","Te123m","Te124","Te125","Te125m","Te126",
29      "Te127m","Te128","Te129m","Te130","Te132","I127","I129","I130","I131","I135",
30      "Xe126","Xe128","Xe129","Xe130","Xe131","Xe132","Xe133","Xe134","Xe135","Xe136",
31      "Cs133","Cs134","Cs135","Cs136","Cs137","Ba134","Ba135","Ba136","Ba137","Ba137m",
32      "Ba138","Ba140","La139","La140","Ce140","Ce141","Ce142","Ce143","Ce144","Pr141",
33      "Pr143","Pr144","Nd142","Nd143","Nd144","Nd145","Nd146","Nd147","Nd148","Nd150",
34      "Pm147","Pm148","Pm148m","Pm149","Pm151","Sm147","Sm148","Sm149","Sm150","Sm151",
35      "Sm152","Sm153","Sm154","Eu151","Eu152","Eu153","Eu154","Eu155","Eu156","Eu157",
36      "Gd152","Gd154","Gd155","Gd156","Gd157","Gd158","Gd160","Tb159","Tb160","Dy160",
37      "Dy161","Dy162","Dy163","Dy164","Ho163","Ho165","Ho166m","Er162","Er164","Er166",
38      "Er167","Er168","Er170"
39  };//チェFP193ンー
40
41  man.MakeFlagFalse(num_include_193chain,nucname_include_193chain);
42
43  // +++
44
45  man.MakeFlagTrue(68,145,0);
46  //  *Note*
47  //   Er145 is stable nuclide in ENDF/B-VII.1 decay data,
48  //   but ENDF-ID for this nuclide cannot given under the format rule.
49  //   So, this nuclide is omitted from burnup chain
50
51  // +++
52
53  man.ShowFlagedNuclideMatID(false);
54  man.ShowFlagedNuclideName(false);
55
56  man.ShowFlagedNuclideShortHalflife(1.,false);
```

# 4 BCGManager からの直接的なデータの取り出し

以下に、BCGManager クラスのヘッダファイルの一部を示す。BCGNuclide クラスのインスタンスがベクターで格納されており、それらを取り出すためのメソッド「GetNuclide」が実装されている。GetNuclide メソッドは、引数として 3 つの整数を指定する場合は、原子番号、質量数、励起準位で取り出す核種を指定し、引数として 1 つの整数を指定する場合は、ベクター配列の位置で取り出す核種を指定する。

Listing 9: BCGManager クラスのヘッダファイル（一部）

```
class BCGManager{
 private:
  vector<BCGNuclide> nuc;
  vector<string> yield_tagname;
 public:
  BCGNuclide &GetNuclide(int atm,int mas,int lev=0);
  BCGNuclide &GetNuclide(int i){return nuc[i];};
};
```

次に、BCGNuclide クラスのヘッダファイルの一部を示す。BCGNuclide クラスのインスタンスが保持している原子番号、質量数、励起準位、崩壊チャンネルの数（「channel」）、半減期とその絶対標準偏差等のアクセス関数が準備されていることが分かるであろう。

Listing 10: BCGNuclide クラスのヘッダファイル（一部）

```
class BCGNuclide{
 private:
  int atomic_number;
  int mass_number;
  int ex_level;
  int r_num; // reaction number
  real awr;
  real half_life;
  real delta_half_life; // (standard deviation of half life)
  int channel;
 public:
  int GetAtomicNumber(){return atomic_number;};
  int GetMassNumber(){return mass_number;};
  int GetExLevel(){return ex_level;};
  int GetID(){return midt.GetMATID(atomic_number,mass_number,ex_level);};//kawamoto
  int GetChannel(){return channel;};
  real GetHalflife(){return half_life;};
  real GetDeltaHalflife(){return delta_half_life;};
  real GetAWR(){return awr;};
};
```

以下には、BCGNuclide クラスのヘッダファイルのうち、崩壊データに関連する部分を抜き出したものを示す。崩壊チャンネル毎に、崩壊後の原子核の原子番号（atn_next）、質量数（mas_next）、励起準位（lev_next）、崩壊様式（decay_type）、中性子放出数（dn）、そのチャンネルへの分岐比（br）とその絶対標準偏差（delta_br）が与えられていることが分かるであろう。

Listing 11: BCGNuclide クラスのヘッダファイル（崩壊データ関連）

```
class BCGNuclide{
 private:
  // (decay)
  int channel;
  vector<int> atn_next;
  vector<int> mas_next;
  vector<int> lev_next;
  vector<int> decay_type; // 0:Beta-, 1:EC, 2:IT, 3:Alpha
  vector<int> dn;
  vector<real> br;
```

```
11    vector<real> delta_br;
12   public:
13    int GetChannel(){return channel;};
14    int GetAtomicNumberNext(int i){return atn_next[i];};
15    int GetMassNumberNext(int i){return mas_next[i];};
16    int GetExLevelNext(int i){return lev_next[i];};
17    int GetDecayType(int i){return decay_type[i];};
18    int GetEmittedNeutron(int i){return dn[i];};
19    real GetBr(int i){return br[i];};
20    real GetDeltaBr(int i){return delta_br[i];};
21  };
```

核反応データに関連するデータは、崩壊データと同様に定義される。以下にヘッダファイルの該当部分を示す。現時点では、(n,g) 反応と (n,2n) 反応のみが考慮可能であり、「r_atn_next」等のような配列は、その要素の一つ目が反応様式（「0」ならば (n,g)、「1」ならば (n,2n)）に対応する。

Listing 12: BCGNuclide クラスのヘッダファイル（核反応データ関連）

```
1   class BCGNuclide{
2    private:
3     // (reaction) 0:ng / 1:n2n
4     vector<int> r_channel;
5     vector< vector<int> > r_atn_next;
6     vector< vector<int> > r_mas_next;
7     vector< vector<int> > r_lev_next;
8     vector< vector<real> > r_br;
9    public:
10    int GetReactionChannel(int i){return r_channel[i];};
11    int GetReactionAtomicNumberNext(int i,int j){return r_atn_next[i][j];};
12    int GetReactionMassNumberNext(int i,int j){return r_mas_next[i][j];};
13    int GetReactionExLevelNext(int i,int j){return r_lev_next[i][j];};
14    real GetReactionBr(int i,int j){return r_br[i][j];};
15  };
```

さらに、BCGNuclide クラスのヘッダファイルのうち、核分裂収率データに関連する部分を抜き出したものを示す。BCGNuclide クラスは、複数の核分裂性核種の核分裂反応による生成収率とその絶対標準偏差を保持する。核分裂性核種の種類は文字型のタグ「yield_tag」で管理し、例えば「GetYield(“u235”);」というメソッドがユーザにより行われた場合には、配列 yield_tag 内を検索し、「“u235”」なる文字列を検出した場合、その位置に対応するデータを実数型の配列「yield」から取り出してユーザに返す、という手続きを行う。

Listing 13: BCGNuclide クラスのヘッダファイル（核分裂収率データ関連）

```
1   class BCGNuclide{
2    private:
3     vector<real> yield;
4     vector<real> delta_yield; // (standard deviation of fission yield)
5     vector<string> yield_tag;
6    public:
7     real GetYield(string tagname);
8     real GetDeltaYield(string tagname);
9   };
```

最後に、遅発中性子放出関連のデータに該当する部分を以下に示す。ある核種が崩壊したときに放出される平均遅発中性子数は、分岐比と放出中性子数の積を全ての崩壊チャンネルについて足し合わせることによって得ることが出来る。また、この崩壊あたりの平均遅発中性子放出数と累積核分裂収率の積を全ての核分裂生成物核種について和をとれば核分裂あたりの遅発中性子発生数を計算することが出来る。一方、変数「n_per_nuc」は、この原子核が 1 回崩壊した場合、娘核種も含めて、平均何個の中性子を放出するかを示す。この娘核種を含めた平均中性子発生数と独立収率の積を全ての核分裂生成物核種について和をとることで、やはり核分裂あたりの遅発中性子

発生数を計算することが出来る。また、変数「n_per_nuc_ch」は、崩壊チャンネル毎の、崩壊あたりの総中性子放出数である。なお、「n_per_nuc」「n_per_nuc_ch」の値は、BCGNuclide クラスのインスタンス群を保持する BCGManager クラスのインスタンスが「CalTotalEmittedNeutrons」メソッドを行わないと計算されないことに注意されたい。

Listing 14: BCGNuclide クラスのヘッダファイル（遅発中性子放出データ関連）

```
class BCGNuclide{
 private:
  // (decay)
  int channel;
  vector<int> dn;
  vector<real> br;
  real n_per_nuc; // number of total emitted neutrons including daughter nuclides
  vector<real> n_per_nuc_ch; // channel-wise
 public:
  int GetEmittedNeutron(int i){return dn[i];};
  real GetBr(int i){return br[i];};
  real GetTotalNumberEmittedNeutrons(){return n_per_nuc;};
  real GetTotalNumberEmittedNeutronsCh(int i){return n_per_nuc_ch[i];};
};
```

BCGManager クラスの「CalTotalEmittedNeutrons」メソッドは、そのインスタンスが保持する BCGNuclide クラスのインスタンス全てについて、「n_per_nuc」を計算する。そのメソッドのソースを以下に示す。BCGNuclide、BCGManager クラスの良い使用例となるであろう。

Listing 15: CanTotalEmittedNeutrons メソッドのソース

```
void BCGManager::CalTotalEmittedNeutrons()
{
  // Total number of emitted neutrons is calculated.
  // Neutron emittions of its daughter nuclides are also considered.

  int sz=nuc.size();
  for(int i=0;i<sz;i++){

    int at_org=nuc[i].GetAtomicNumber();
    int ms_org=nuc[i].GetMassNumber();
    int lv_org=nuc[i].GetExLevel();

    bool end=false;
    int iter=0;

    real n_emit=0.;

    // +++ initialize
    int ngc=nuc[i].GetChannel();
    int ngc_org=ngc;
    vector<real> n_emit_ch(ngc,0.);
    vector<int> atn(ngc);
    vector<int> mas(ngc);
    vector<int> lev(ngc);
    vector<real> bri(ngc);
    vector<int> ch_id(ngc);
    for(int j=0;j<ngc;j++){
      atn[j]=nuc[i].GetAtomicNumberNext(j);
      mas[j]=nuc[i].GetMassNumberNext(j);
      lev[j]=nuc[i].GetExLevelNext(j);
      bri[j]=nuc[i].GetBr(j);
      ch_id[j]=j;
      n_emit+=bri[j]*nuc[i].GetEmittedNeutron(j);
      n_emit_ch[j]+=bri[j]*nuc[i].GetEmittedNeutron(j);
    };

    while(!end){
      int ngc2=0;
      vector<int> atn2;
```

```
40        vector<int> mas2;
41        vector<int> lev2;
42        vector<real> bri2;
43        vector<int> ch_id2;
44
45        end=true;
46        for(int j=0;j<ngc;j++){
47          int id=GetNuclideIndex(atn[j],mas[j],lev[j]);
48          int ch_org=ch_id[j];
49          if(id!=-1){
50            int ngc3=nuc[id].GetChannel();
51            for(int l=0;l<ngc3;l++){
52              int am=nuc[id].GetAtomicNumberNext(l);
53              int ms=nuc[id].GetMassNumberNext(l);
54              int lv=nuc[id].GetExLevelNext(l);
55              if(am!=atn[j]||ms!=mas[j]||lv!=lev[j]){
56                end=false;
57                real br=nuc[id].GetBr(l);
58                real dn=nuc[id].GetEmittedNeutron(l);
59                n_emit+=(bri[j]*br)*dn;
60                n_emit_ch[ch_org]+=(bri[j]*br)*dn;
61                bool exist=false;
62                for(int m=0;m<ngc2;m++){
63                  if(am==atn2[m]&&ms==mas2[m]&&lv==lev2[m]){
64                    exist=true;
65                    bri2[m]+=bri[j]*br;
66                  };
67                };
68                if(!exist){
69                  atn2.push_back(am);
70                  mas2.push_back(ms);
71                  lev2.push_back(lv);
72                  bri2.push_back(bri[j]*br);
73                  ch_id2.push_back(ch_org);
74                  ngc2++;
75                };
76              };
77            };
78          };
79        };
80        ngc=ngc2;
81        atn.resize(ngc);
82        mas.resize(ngc);
83        lev.resize(ngc);
84        bri.resize(ngc);
85        ch_id.resize(ngc);
86        for(int j=0;j<ngc;j++){
87          atn[j]=atn2[j];
88          mas[j]=mas2[j];
89          lev[j]=lev2[j];
90          bri[j]=bri2[j];
91          ch_id[j]=ch_id2[j];
92        };
93        iter++;
94      };
95
96      nuc[i].PutTotalNumberEmittedNeutrons(n_emit);
97      real sum=0.;
98      for(int j=0;j<ngc_org;j++){
99        sum+=n_emit_ch[j];
100     };
101     if(fabs(sum-n_emit)>1e-5){
102       cout<<sum<<" "<<n_emit<<" "<<ngc<<"\n";
103       exit(0);
104     };
105     nuc[i].PutTotalNumberEmittedNeutronsCh(ngc_org,n_emit_ch);
106 };
```

# 5 擬似 FP 核種

限られた数の FP 核種で構成される簡略燃焼チェーンを用いる場合、無視される FP 核種の一部については、その収率がチェーン下流に位置する核種の収率に足されることになるため、「陰に」その存在が考慮されることになる。その一方で、陽にも陰にも考慮されない FP 核種も存在し、一般的に簡略化された燃焼チェーンを用いた場合、それらの中性子吸収への寄与が考慮されず、燃焼の進展に伴い反応度が過大評価されることになる。そのような計算誤差を回避するため、擬似的な FP 核種を導入することが多い。これは、簡略チェーンで無視される複数の FP 核種を擬似的な「単一の」FP 核種と見做す方法である。

# 6 おわりに

燃焼チェーンで考慮すべき核種を選別するアルゴリズムについては川本洋右君の仕事に負うところが大きいことを付記する。

## 参考文献

[1] J. Katakura, “JENDL FP Decay Data File 2011 and Fission Yields Data File 2011,” JAEA-Data/Code 2011-025, Japan Atomic Energy Agency, (2012).