CBZ/IndependentYieldCovariance における 相関を考慮した独立核分裂収率の共分散行列の計算方法 [1]

2018/6/13　千葉　豪

独立核分裂収率に対する誤差情報としては、分散のみが評価済み核データファイルに与えられている。これは評価済み核データファイルの標準的なフォーマットである ENDF フォーマットの制限によるものであり、物理的には同一の質量チェーンに属する FP 核種の独立収率には相関が存在するものと考えられる。また、独立核分裂収率の総和は、二体核分裂のみを考える場合には 2 となるべきであり、その点を考慮すると、同一の質量チェーンに限らず全ての FP 核種の独立収率に相関が存在する筈とも言える。

独立核分裂収率の核種間の相関を考慮する方法としてはいくつか挙げられるが、CBZ では以下で述べる Devillers の方法により同一の質量チェーンに属する FP 核種の独立収率の相関を考慮している [1, 2]。ある質量を有する FP 核種の総数を J とし、それらの核分裂収率の総和に対する標準偏差を σ とする。これはすなわち質量チェーンの最下流に位置する FP 核種の累積核分裂収率に対する標準偏差に相当する。これら同一の質量を有する FP 核種が構成する崩壊チェーン（質量チェーン）に属する FP 核種 i の独立核分裂収率に対する標準偏差が評価済み核データに σ_i と与えられているとき、Devillers の方法では FP 核種 i と j の独立核分裂収率に対する共分散 $\bar{V}_{ij}$ を以下で計算する。

$$\bar{V}_{ii} = \sigma_i^2 \left(1 - \frac{\sigma_i^2}{\sigma^2 + \sum_{j=1}^{J} \sigma_j^2} \right), \tag{1}$$

$$\bar{V}_{ij} = -\frac{\sigma_i^2 \sigma_j^2}{\sigma^2 + \sum_{j=1}^{J} \sigma_j^2}. \tag{2}$$

以下では、この方法に基づく核分裂収率の共分散行列の構築に関するコーディングの詳細について説明する [2]。

IndependentYieldCovariance クラスは「YieldDecayCovariance」という名前のファイルにおいて定義されており、独立核分裂収率の共分散行列を作成するのは MakeCovarianceMatrix メソッドである。このメソッドのソースを以下に示す。

Listing 1: MakeCovarianceMatrix メソッドのソース

```
void IndependentYieldCovariance::CalCovariance(BCGManager &bm, int num)
{
  int sz=size_array[num]; // # of FP nuclides

  int z;
  int a;
  int l;
  matrix[num].set_zero();

  vector<real> unc_idp_sqr(sz);
  for(int i=0;i<sz;i++){
    unc_idp_sqr[i]=pow(unc_idp[num][i],2.);
  };

  vector<int> z_array;
  vector<int> a_array;
  vector<int> l_array;
  int a_max=0;
  for(int i=0;i<sz;i++){
    eidt.GetParameterNew(id[num][i],z,a,l);
    z_array.push_back(z);
    a_array.push_back(a);
    l_array.push_back(l);
    if(a>a_max)a_max=a;
  };

  vector<bool> exist(a_max,false);
  for(int i=0;i<sz;i++){
    exist[a_array[i]]=true;
  };
```

[1] Document/CBG_Manual/FYCovariance
[2] なお、このコーディングを実際に行ったのは川本洋右氏である。

```
  cout<<"# FISSILE : "<<fisnucname_array[num]<<"\n";

  for(int i=1;i<=a_max;i++){
    if(exist[i]){

    //+++Searching Mass Yield+++
    vector<real> mass_unc;      // absolute variance
    vector<real> mass_unc_idp; // absolute variance
    vector<int> mass_nuc;
    vector<int> mass_z;
    int mass_nuc_num=0;
    vector<int> mass_chain_id;//1,2,3,...
    mass_chain_id.resize(sz,0);

    for(int j=0;j<sz;j++){

      z=z_array[j];
      a=a_array[j];
      l=l_array[j];

      if(bm.GetNuclideIndex(z,a,l)==-1){
        bm.AddNuclide(z,a,l);
      };

      if(a==i){
        int channel=bm.GetNuclide(z,a,l).GetChannel();
        real hl=bm.GetNuclide(z,a,l).GetHalflife();
        bool end_nuclide=false; // To detect the end nuclide in the mass chain
        if(hl==0.){
          end_nuclide=true;
        }else if(channel==1){
          int decay_type=bm.GetNuclide(z,a,l).GetDecayType(0);
          if(decay_type==3)end_nuclide=true; // (Alpha-decay)
        };
        if(end_nuclide){
          /*
          cout<<"# The final nuclide in the mass chain "<<a<<" is detected.\n";
          cout<<"#    (Z,A,L) of this nuclide : "<<z<<","<<a<<","<<l<<"\n";
          cout<<"#    Cumulative yield uncertainty : "<<unc_cum[num][j]<<"\n";
          */
          mass_unc.push_back(pow(unc_cum[num][j],2));
          mass_unc_idp.push_back(unc_idp_sqr[j]); // absolute standard deviation
          mass_nuc.push_back(id[num][j]);
          mass_z.push_back(z);
          mass_nuc_num++;
          mass_chain_id[j]=mass_nuc_num;
        };
      };

    };

    vector<real> sum_unc_square;
    sum_unc_square.resize(mass_nuc_num);

    if(mass_nuc_num>=1){

      real tmp2;
      for(int jj=0;jj<mass_nuc_num-1;jj++){
        tmp2=0.;
        for(int j=0;j<sz;j++){
          z=z_array[j];
          a=a_array[j];
          if(a==i&&mass_z[jj]>z&&mass_chain_id[j]==0){
            mass_chain_id[j]=jj+1;
            tmp2+=unc_idp_sqr[j];
          };
        };
        sum_unc_square[jj]=tmp2+mass_unc_idp[jj];
      };

      tmp2=0.;
      for(int j=0;j<sz;j++){
        a=a_array[j];
        if(a==i&&mass_chain_id[j]==0){
          mass_chain_id[j]=mass_nuc_num;
          tmp2+=unc_idp_sqr[j];
        };
      };
      sum_unc_square[mass_nuc_num-1]=tmp2+mass_unc_idp[mass_nuc_num-1];

      for(int ii=0;ii<mass_nuc_num;ii++){
        for(int k=0;k<sz;k++){
          if(mass_chain_id[k]==ii+1){
            for(int kk=0;kk<sz;kk++){
              if(kk==k){
                real tmp1=yield_idp[num][k]*yield_idp[num][kk];
                if(tmp1!=0.){
                  real tmp2=unc_idp_sqr[k];
                  real tmp=tmp2*(1.-tmp2/(mass_unc[ii]+sum_unc_square[ii]))/tmp1;
                  matrix[num].put_data(k,kk,tmp);
                };
              }else if(mass_chain_id[kk]==ii+1){
```

```
124                     real tmp1=yield_idp[num][k]*yield_idp[num][kk];
125                     if(tmp1!=0.){
126                       real tmp=-unc_idp_sqr[k]*unc_idp_sqr[kk]/
127                               (mass_unc[ii]+sum_unc_square[ii])/tmp1;
128                       matrix[num].put_data(k,kk,tmp);
129                     };
130                   };
131                 };
132               };
133             };
134           };
135
136         };
137
138       };
139     };
```

このメソッドでは、ある入射エネルギーの中性子によるある重核種の独立核分裂収率について共分散を計算する。

34 行目からの質量数のループにおいて、各質量数の FP 核種の独立核分裂収率の分散と共分散を計算する。

46 行目からは FP 核種のループになっており、ここで着目する質量数の FP 核種について、半減期がゼロ、もしくはアルファ崩壊しか崩壊経路がないものを特定する。この核種がこの質量数チェーン（mass chain）の最下流に位置する核種となる。なお、同一の質量数であってもこのような最下流核種が複数存在する場合があることに注意が必要である。

88 行目から 100 行目までの処理は、着目質量数について複数の最下流核種が存在するときのためのものである。例えばそのような核種が n 個存在するとした場合、最初の $n-1$ 個の核種については、原子核の崩壊が全て β^- 崩壊であると想定し、自らと同一の質量数でかつ自らより原子番号が小さく、さらにまだタグ付けされていない（配列 mass_chain_id がゼロとなっている）ものを同一の質量チェーンとみなしタグ付けする。従って、FP 核種のデータの並びが原子番号が小さい順（大きくなっていく順）に並んでいることが暗黙で想定されていることになる。この処理により、同一の質量数であっても、複数の異なる質量チェーンを考慮することが可能となっている。

最後に 112 行目以降でそれぞれの質量チェーンについて、Devillers の式に基づいて共分散行列を作成する。

なお、このメソッドでは、β^- 崩壊による系列と β^+ 崩壊による系列とは区別されていないこと、遅発中性子の放出を伴う崩壊を考慮していないことに注意が必要である。

このメソッドにより計算した質量数 160 の 5 つの FP 核種の独立収率に関する共分散行列を **Table 1** に示す。この質量数では Gd-160、Dy-160 が安定核であるため、Gd-160 とその親核である Sm-160、Eu-160 に関する共分散行列と、Dy-160、Tb-160 の共分散行列とに分けられる（ブロック対角行列となっている）ことが分かるであろう。

Table 1: Covariance matrix of fission yield

	Sm	Eu	Gd	Tb	Dy
Sm (62*)	+0.26	-0.17	-0.04	0	0
Eu (63)		+0.21	-0.04	0	0
Gd (64)			+0.40	0	0
Tb (65)				+0.20	-0.00
Dy (66)					+0.41

* Atomic number

参考文献

[1] Devillers C., ‘The importance of fission product nuclear data in reactor design and operation,’ IAEA Panel, Petten (1977).

[2] Katakura J., ‘JENDL FP decay data file 2011 and fission yields data file 2011,’ JAEA-Data/Code 2011-025 (2011).