CBZのXYZ座標系の炉心体系情報に関するクラス

千葉豪

平成27年10月6日

1 概要

CBZでは、XYZ座標系の炉心体系情報は、体系がよほど簡単でない場合にはCartCoreクラスで定義する（よほど簡単な場合については後述）。CartCoreクラスは、炉心内を上から見たときの燃料集合体配置情報と、各々の燃料集合体情報とで構成される。具体的には、燃料集合体の軸方向に関する物質組成情報はAssemblyクラスで定義され、Assemblyクラスのインスタンス群を格納するクラスAssemblySetクラスのインスタンスがCartCoreクラスのメンバ変数として定義される。また、燃料集合体の炉心内配置情報はCartCoreクラスで定義される。

一方、中性子輸送もしくは拡散ソルバー内で直接用いられる、XYZ座標系での計算メッシュ分割の情報は、CartMeshInfoクラスで定義される。CartMeshInfoクラスのインスタンスはCartCoreクラスのインスタンスから自動生成することが可能である。なお、体系がよほど簡単な場合には、直接CartMeshInfoクラスのインスタンスを作成したほうが手間が少ない。

2 クラスの解説

ここでは関連するクラスについての解説を行う。利用法を学ぶことが目的であるならば、この節はスキップしても問題ない。

2.1 燃料集合体クラス Assembly

燃料集合体の軸方向の組成情報を定義するのがAssemblyクラスである。Assemblyクラスのヘッダファイルを以下に示す。

Listing 1: Assembly.h

```
1  class Assembly{
2   protected:
3    vector<int> MaterialID;
4    vector<string> MaterialName;
5    vector<real> zdist; // Boundary point between different material along to z-axis
6    int     zdiv;      //Number of division along to z-axis
7    string AsmName;   //Assembly name
8    bool exist;
9   public:
10   Assembly(){exist=false;};
11   Assembly(int zdivi,string *zmapi,real *zdisti,string inp,string type="cumulative")
12     {Init(zdivi,zmapi,zdisti,inp,type);};
13   Assembly(int zdivi,vector<string> zmapi,vector<real> zdisti,string inp,
14            string type="cumulative")
15     {Init(zdivi,zmapi,zdisti,inp,type);};
16   ~Assembly(){};
17   void Init(int zdivi,string *zmapi,real *zdisti,string inp,string type="cumulative");
18   void Init(int zdivi,vector<string> zmapi,vector<real> zdisti,string inp,
```

```
19              string type="cumulative");
20    void show_self();
21    int GetZdiv(){return zdiv;};
22    void PutMaterialID(int i,int j){MaterialID[i]=j;};
23    int GetMaterialID(int i){return MaterialID[i];};
24    real GetZdist(int i){return zdist[i];};
25    void PutMaterialName(int i,string inp){MaterialName[i]=inp;};
26    string GetMaterialName(int i){return MaterialName[i];};
27    string GetName(){return AsmName;};
28    bool Exist(){return exist;};
29  };
```

メンバ変数は、燃料集合体の Z 軸に沿った構成物質領域数に対応する zdiv、Z 軸方向に沿って定義される物質 ID データ列を格納する MatarialID、異なる物質間の境界の Z 軸上の位置に対応する配列 zdist 等で構成される。

2.2 燃料集合体の集合クラス AssemblySet

Assembly クラスのインスタンス群を格納するクラスが AssemblySet である。AssemblySet クラスのヘッダファイルを以下に示す。

Listing 2: AssemblySet.h

```
class AssemblySet{
 protected:
  int AsmNum;                    //Number of included 'Assembly' instances
  int MatNum;                    //Number of included material
  Assembly Asm[MAX_ASM];         //Included 'Assembly' instances
  string MaterialName[MAX_MAT]; //Name of included material
  bool exist;
 public:
  AssemblySet(int matinp);
  ~AssemblySet(){};
  void PutAsm(Assembly &inp);  //Put instance of 'Assembly' into this class
  void PutMaterialName(vector<string> inp);
  void PutMaterialName(string *inp);
  void show_self();
  int GetAsmNum(){return AsmNum;};
  int GetMatNum(){return MatNum;};
  string GetMaterialName(int i){return MaterialName[i];};
  Assembly &GetAsm(int i){return Asm[i];};
  Assembly GetJointAssembly(int n, string *r);
};
```

メンバ変数 MatNum は、AssemblySet クラスが格納する Assembly クラスのインスタンス群が含む物質種類数を定義する。従って、Assembly クラスのメンバ変数 MaterialID で、MatNum を越えた ID が定義されるとエラーになる。

定数 MAX_ASM は、AssemblySet クラスのインスタンスが格納することができる Assembly クラスのインスタンスの最大数である。Assembly クラスのインスタンスは AssemblySet クラスの PutAsm メソッドでひとつづつ登録し、Assembly クラスのインスタンスの格納数、すなわち AsmNum が MAX_ASM を超過するとエラーが発生する。

2.3 炉心体系クラス CartCore

XYZ 座標系の炉心体系情報を定義する CartCore クラスのヘッダファイルを以下に示す。

Listing 3: CartCore.h

```
class CartCore{
 protected:
  int xr,yr;            //Region number along to x- and y-axis
```

```
  vector<int> AsmMap; //Assembly map on xy-plane
  vector<real> xwid;
  vector<real> ywid;
  int LeftBC,RightBC,BackBC,FrontBC,UpperBC,BottomBC;
        // Boundary condition (0/1:vacuum/reflective)
  AssemblySet *AsmSet; //'AssemblySet' instance
 public:
  /** AssemblyMap(AssemblyID is described in XY plane)   **/
  /** MaterialMap(MaterialID is described in XYZ system) **/
  CartCore(){};
  CartCore(int x,int y){Initialize(x,y);};
  ~CartCore();
  void Initialize(int x,int y);
  void PutAsmMap(vector<int> inp);
  void PutAsmMap(int *inp);
  void PutAsmMap(int x1,int x2,vector<int> inp);
  void PutAsmMap(int x1,int x2,int *inp);
  void PutAsmMap_Hex_to_XY(int *inp);
  void PutAsmSet(AssemblySet *inp){AsmSet=inp;};
  void PutWidthXY(vector<real> xinp,vector<real> yinp);
  void PutWidthXY(real *xinp,real *yinp);
  void PutWidthFromPitch(real pitch);
  void PutBC(vector<int> inp);
  void PutBC(int *inp);
  void PutBoundaryCondition
    (string xl="Vacuum",string xr="Vacuum",
     string yl="Vacuum",string yr="Vacuum",
     string zl="Vacuum",string zr="Vacuum");
  void ShowAssemblyMapNew();
  void ShowAssemblyMap();
  void ShowAssemblyMap(int ii, int ij=-1);
  void ShowMaterialMap();
  void MakeMaterialMap(int &zr,int *MaterialMap,real *zwid);
  void GetMaterialMapXY(real zinp,int *map); /** To get M.Map at z=zinp **/
  int GetUnifiedZMesh(); /** To get number of unified Z mesh **/
  // for XYLattice
  void PutLatticeMap(int x,int y,int *latmap,real *xwid,real *ywid,XYLatticeSet &latset,
                     bool print=false);
  //
  void ChangeAssembly(int x,int y,int asmid);
  void ChangeAssembly(int p,int* xp,int *yp,int asmid);
  void ChangeAssembly(int asmid1,int asmid2);
  //Output of member function
  int GetLeftBC(){return LeftBC;};
  int GetRightBC(){return RightBC;};
  int GetBackBC(){return BackBC;};
  int GetFrontBC(){return FrontBC;};
  int GetUpperBC(){return UpperBC;};
  int GetBottomBC(){return BottomBC;};
  int GetXr(){return xr;};
  int GetYr(){return yr;};
  real GetXwid(int i){return xwid[i];};
  real GetYwid(int i){return ywid[i];};
  int GetAsmMap(int i){return AsmMap[i];};
  int UsedOrNotAssembly(int i); // 0/1:No/Yes
  AssemblySet *GetAset(){return AsmSet;};
  void CountAssembly(int i);
};
```

前述したように、CartCore クラスでは、XY 平面上での燃料集合体の配置マップと燃料集合体クラス Assembly のインスタンス群を格納する AssemblySet クラスのインスタンスにより、XYZ 座標系における体系データを定義する。AssemblySet クラスのインスタンスはポインタとして保持される。境界条件もこのクラスで定義され、例えば X 軸負側の境界条件は LeftBC として定義されている (真空条件と反射条件、ゼロ中性子束条件が定義できる)。

ユーザは、AssemblySet クラスのインスタンスを準備し、X、Y 方向の領域分割数、各分割領域の長さ、燃料集合体マップを、PutAsmSet、PutAsmMap 等のメソッドにより入力する。その情報から炉心全体の物質マップが作成される。また、数値計算では、炉心を構成する各燃料集合体の Z 軸での物質領域分割から、統合的な領域分割情報（全ての燃料集合体の Z 軸方向の物質境界

が一致する領域分割）を作成する必要があるが、GetUnifiedZMesh メソッドで、統合 Z 軸領域分割数を計算出来る。さらに、MakeMaterialMap メソッドにより、Z 軸を統合物質領域分割したあとでの XYZ に関する物質 ID マップ（一次元に配列されている）が生成される。生成されたマップは整数へのポインタ MaterialMap により引数として渡されるため、このメソッドを呼び出す前に MaterialMap のためのメモリを確保する必要がある[1] 。

2.4 計算メッシュクラス CartMeshInfo

CBZ の拡散ソルバー PLOS、Sn 輸送ソルバー SNR、SNRZ、SNT、衝突確率法輸送ソルバー PJI での計算メッシュ分割情報は CartMeshInfo クラスで定義される (なお、PJI では一次元平板体系でのみ CartMeshInfo クラスを利用している)。CartMeshInfo クラスのヘッダファイルを**リスト 4** に示す。

Listing 4: CartMeshInfo.h

```
class CartMeshInfo{
  int FMesh[3],CMesh[3];
  int BC[6];
  vector< vector<real> > FMeshL;
  vector< vector<int> >  CMeshF;
  vector< vector<real> > CMeshL;
  vector<int> FMat; // including "-1" medium
  vector<int> FMat_par_mesh; // NOT including "-1" medium
  vector<int> CMat;
 public:
  CartMeshInfo(){};
  void PutMeshXY(int *xm,int *ym,real z,CartCore &core);
  void PutMeshXYZ(int *xm,int *ym,real zwidf,CartCore &core,real zwidc=20.);
  void PutMeshXYZ(int *xm,int *ym,int *zm,CartCore &core,real zwidc=20.);
  void PutMeshXY(vector<int> xm,vector<int> ym,real z,CartCore &core);
  void PutMeshXYZ(vector<int> xm,vector<int> ym,real zwidf,CartCore &core,real zwidc=20.);
  void PutMeshInfo(int xr,int yr,int zr,int *fmx,int *fmy,int *fmz,
                   real *xl,real *yl,real *zl,int *mat,string type="width");
  void PutMeshInfo(int xr,int yr,int *fmx,int *fmy,real *xl,real *yl,int *mat,
                   string type="width");
  void PutMeshInfo(int xr,int *fmx,real *xl,int *mat,string type="width");
  void PutMeshInfo(int xr,int yr,int zr,vector<int> fmx,vector<int> fmy,vector<int> fmz,
                   vector<real> xl,vector<real> yl,vector<real> zl,vector<int> mat,
                   string type="width");
  void PutMeshInfo(int xr,int yr,vector<int> fmx,vector<int> fmy,
                   vector<real> xl,vector<real> yl,vector<int> mat,string type="width");
  void PutMeshInfo(int xr,vector<int>fmx,vector<real>xl,vector<int>mat,
                   string type="width");
  int GetFMesh(int i){return FMesh[i];};
  int GetXF(){return FMesh[0];};
  int GetYF(){return FMesh[1];};
  int GetZF(){return FMesh[2];};
  int GetXC(){return CMesh[0];};
  int GetYC(){return CMesh[1];};
  int GetZC(){return CMesh[2];};
  int GetCMesh(int i){return CMesh[i];};
  real GetFMeshL(int i,int j){return FMeshL[i][j];};
  int GetCMeshF(int i,int j){return CMeshF[i][j];};
  real GetCMeshL(int i,int j){return CMeshL[i][j];};
  int GetFMat(int i){return FMat[i];};
  int GetFMatParMesh(int i){return FMat_par_mesh[i];};
  int GetCMat(int i){return CMat[i];};
  void PutBoundaryCondition(int *binp);
  void PutBoundaryCondition(vector<int> binp);
  void PutBoundaryCondition
    (string xl="Vacuum",string xr="Vacuum",
     string yl="Vacuum",string yr="Vacuum",
```

[1] MaterialMap のためのメモリ確保のためには、Z 軸での統合領域分割数がいくつになるかを事前に知る必要がある。従って、はじめに GetUnifiedZMesh メソッドにより Z 軸の領域分割数を求め、それに応じて MaterialMap のメモリを確保したのち、MakeMaterialMap メソッドを呼び出すことになる。

```
    string zl="Vacuum",string zr="Vacuum");
  int GetBC(int i){return BC[i];};

  void ReconstructCoarseMesh(real maxxl,real maxyl,real maxzl);
  bool CheckCoarseMeshMixedVacuum();

  int GetMeshPosition(int dir,real x);
  int GetXMeshPosition(real x){return GetMeshPosition(0,x);};
  int GetYMeshPosition(real x){return GetMeshPosition(1,x);};
  int GetZMeshPosition(real x){return GetMeshPosition(2,x);};
  real GetMeshLocation(int dir,int x);
  real GetXMeshLocation(int x){return GetMeshLocation(0,x);};
  real GetYMeshLocation(int x){return GetMeshLocation(1,x);};
  real GetZMeshLocation(int x){return GetMeshLocation(2,x);};

  void show_self();
  void GetPositionFromMeshID(int id);
};
```

各軸（X、Y、Z 軸）の詳細 (計算) メッシュ分割数 (FMesh) と各詳細メッシュの長さ (FMeshL)、粗メッシュ分割数 (CMesh) と各粗メッシュの長さ (CMeshL)、各粗メッシュに含まれる詳細メッシュ数 (CMeshF) がメンバ変数として格納される。また、物質 ID のマップが vector 型変数 FMat に格納される。この FMat での物質 ID は詳細メッシュの各メッシュで定義される。境界条件は整数の配列 BC として定義されている。CartCore のように「LeftBC」等のように定義する方法もあるが、こういった内部データにユーザーが直接アクセスすることはないので、このような簡潔な記述のほうがむしろ好ましいと考えている。

直接このクラスのインスタンスを作成して計算メッシュを構築する場合には PutMeshInfo メソッドを用いる。このメソッドでは、各軸の粗メッシュ数、各粗メッシュの詳細メッシュ分割数、各粗メッシュの長さ、粗メッシュ分割上での物質マップを引数として入力する。引数として string の「type」があるが、これは粗メッシュの長さを指定する際に、‘width’ としてやれば各メッシュの長さが入力となり、‘cumulative’ としてやれば原点からの各メッシュ端の長さが入力となる。

なお、CBZ では便宜的に球体系、円筒体系の記述もこのクラスを用いて行っている。

また、このクラスは CartCore クラスのインスタンスを引数として読み込み、計算メッシュを自動生成することが可能である。PutMeshXY は二次元計算用のメッシュを生成するメソッドであり、CartCore クラスのインスタンスの他に、X、Y 軸の各粗メッシュの詳細メッシュ分割数と、計算する XY 平面の Z 位置を指定する。PutMeshXYZ メソッドは三次元計算用であり、Z 軸方向の詳細メッシュの最大長さを指定する。

CartMeshInfo クラスのインスタンスのポインタは各ソルバーに保持される。

CartMeshInfo クラスでの粗メッシュデータは物質 ID を設定する際に有効であるが、それ以外の用途として、拡散ソルバー PLOS の粗メッシュ拡散加速でこのクラスで定義される粗メッシュをそのまま加速法における「粗メッシュ」として扱っている点が挙げられる。

3 利用の実際

3.1 媒質マップと物質配置、境界条件の関係

本節では、CartCore クラス、CartMeshInfo クラスで用いられている、XYZ 体系における物質配置情報を定義する方法について説明する。

はじめに、単純な体系を例にして概念を説明する。

3×3×3 の立方体を考え、中心の物質が「1」、それを囲む物質が「0」であるとする。CBZ ではこの三次元体系を一次元配列で以下のように記述する。

0,0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,0,0, 0,0,0,0,0,0,0,0,0

最初の 9 個の数字が Z 軸上でひとつめの「平面」(z=0) に対応し、次の 9 個の数字がふたつめ (z=1)、最後の 9 個の数字が三つめの平面 (z=2) に対応している。また、各々の平面は、(x,y)=(0,0) のものから (1,0)、(2,0) と続き、x が最大値に達すると、(0,1)、(1,1)、...、というように続く。

この一次元配列は、工夫すると以下のように記述できるであろう。

```
0,0,0,
0,0,0,
0,0,0, // z=0 plane

0,0,0,
0,1,0,
0,0,0, // z=1 plane

0,0,0,
0,0,0,
0,0,0 // z=2 plane
```

単に一行で示すよりも少しはイメージがしやすいものとなっているのではないだろうか。

さて、このように体系が対称性を有している場合は、全体系を計算するのではなく、適切な境界条件を用いてモデル化し、その一部のみを計算するようにすれば負荷を低減することが出来る。上のように 1/8 対称となっている場合は、全部で 6 面ある外部境界平面のうち 3 面を「反射条件 (Reflective boundary condition)」として設定してやれば良い。それでは X 軸に対して垂直となる 2 つの外部境界面について考えよう。このとき、2 つの外部境界を区別するため、X 軸に対して負側の境界面を「X 軸負側外部境界」、正側の境界面を「X 軸正側外部境界」として記述することとする。Y 軸に垂直となる外部境界面、Z 軸に垂直となる外部境界面についても同様である。

例えば、X 軸負側外部境界を「反射条件」とした場合、上記の体系は以下のように記述される。

```
0,0,
0,0,
0,0, // z=0 plane

0,0,
1,0,
0,0, // z=1 plane

0,0,
0,0,
0,0 // z=2 plane
```

ただし、この場合、X 軸方向のひとつめのメッシュの長さは半分となることに注意されたい。

また、Y 軸負側境界を反射条件とした場合は次のようになる。

```
0,0,0,
```

0,0,0, // z=0 plane

0,1,0,
0,0,0, // z=1 plane

0,0,0,
0,0,0 // z=2 plane

この場合、同様に Y 軸方向のひとつめのメッシュの長さは半分となる。
　さらに、Z 軸負側境界を反射条件とした場合は次のようになる。

0,0,0,
0,1,0,
0,0,0,

0,0,0,
0,0,0,
0,0,0

この場合、Z 軸方向のひとつめのメッシュの長さは半分となる。
　最後に、X、Y、Z 軸負側境界すべてを反射条件とした場合（1/8 体系）は次のようになる。

1,0,
0,0,

0,0,
0,0

　なお、外部境界条件は、CartCore、CartMeshInfo クラスともに、６つのキーワードの並びで定義する。並び順は、X 軸負側、X 軸正側、Y 軸負側、Y 軸正側、Z 軸負側、Z 軸正側、となっている。

3.2　CartCore による体系情報の作成

　さて、前節で示した三次元の体系情報を、Z 軸方向に関する情報と XY 平面に関する情報とに分離して記述することを考えよう。この三次元体系において、Z 軸に沿った物質配置 (これが燃料集合体情報に該当する) は、

- 「0」「0」「0」
- 「0」「1」「0」

の二種類に分類されることが分かるであろう。この一つ目を「A」、二つ目を「B」として、XY 平面上でこれを次のように配置する。

A,A,A,
A,B,A,
A,A,A,

この記述が上記の三次元体系の記述となる。

次に、実際の問題として竹田ベンチマーク [1] のモデル 2 を CartCore クラスで記述した例を**リスト 5** に示す (制御棒挿入体系。竹田ベンチマークの詳細は本メモの末尾を参照のこと)。

Listing 5: CartCore クラスによる竹田ベンチマークモデル 2 の記述

```
  int main()
{
  // *** Takeda Benchmark : Problem 2  Small FBR ***

  int asmnum = 3;
  int mednum = 5;

  // Assembly
  Assembly asmi[asmnum];
  float in0[]={20., 75., 130., 150.};
  int   mt0[]={ 2 , 0 ,  0 ,  2 };
  asmi[0].Init(4,mt0,in0,"Fuel");
  float in1[]={150.};
  int   mt1[]={1};
  asmi[1].Init(1,mt1,in1,"RadBla");
  float in2[]={75., 150.};
  int   mt2[]={ 3 ,  4 };
  asmi[2].Init(2,mt2,in2,"CR");

  // AssemblySet
  AssemblySet aset;
  char *name[]={"core","radb","axib","cr","crp"};
  aset.PutMaterialName(mednum,name);
  for(int i=0;i<asmnum;i++){
    aset.PutAsm(asmi[i]);
  };

  // CartCore
  CartCore cc(14,14);
  int mp[]={
    0,0,0,0,0,0,0,2,2,0,0,1,1,1,
    0,0,0,0,0,0,0,0,0,0,0,1,1,1,
    0,0,0,0,0,0,0,0,0,0,0,1,1,1,
    0,0,0,0,0,0,0,0,0,0,1,1,1,1,
    0,0,0,0,0,0,0,0,0,0,1,1,1,1,
    0,0,0,0,0,0,0,0,0,0,1,1,1,1,
    0,0,0,0,0,0,0,0,0,1,1,1,1,1,
    0,0,0,0,0,0,0,0,0,1,1,1,1,1,
    0,0,0,0,0,0,0,0,1,1,1,1,1,1,
    0,0,0,0,0,0,0,1,1,1,1,1,1,1,
    0,0,0,1,1,1,1,1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1};
  cc.PutAsmMap(mp);
  cc.PutBoundaryCondition
    ("Reflective","Vacuum","Reflective","Vacuum","Vacuum","Vacuum");
  float xmesh[]={5., 5., 5., 5., 5., 5., 5., 5., 5., 5., 5., 5., 5.};
  float ymesh[]={5., 5., 5., 5., 5., 5., 5., 5., 5., 5., 5., 5., 5.};
  cc.PutWidthXY(xmesh,ymesh);
  cc.PutAsmSet(&aset);

  // CartMeshInfo
  CartMeshInfo cmi;
  int xmf[]={1,1,1,1,1,1,1,1,1,1,1,1,1,1};
  cmi.PutMeshXYZ(xmf,xmf,5.,cc);
};
```

5 行目、6 行目ではこの体系を記述するための燃料集合体数 asmnum、媒質（物質）数 mednum を定義する。

8 行目から 18 行目までは三種類の燃料集合体情報を定義している。例えば 10 から 12 行目で定義している燃料集合体「0」(asmi[0]) の情報であるが、Z 軸方向の異なる物質間の境界位置を in0 で、Z 軸方向の物質 ID の並びを mt0 で定義してやり、その情報を Init メソッドにより asmi[0] に代入する。Init メソッドの一つ目の引数は Z 軸方向の物質数であり、四つ目の引数はその燃料集合体にユーザがつける名前である (この名前は重要ではない)。なお、この例での物質 ID とベンチマーク問題での物質との対応であるが、「0」がベンチマーク問題における「CORE」、「1」が「RADIAL BLANKET」、「2」が「AXIAL BLANKET」、「3」が「CR」、「4」が「CRP」となる。

20 行目から 26 行目までは、8 行目から 18 行目までで作成した Assembly クラスのインスタンス群を AssemblySet クラスに代入している。ここで、各物質に名前を与えているが、これはそれほど重要ではない。

29 行目では CartCore クラスのインスタンスを生成している。引数となっている整数は、それぞれ X 軸及び Y 軸の領域分割数である。30 行目から 44 行目で配列変数 mp に代入しているのは XY 平面上での燃料集合体の配置マップである。ここでの数値が、8 から 18 行目で作成した Assembly クラスのインスタンスの asmi[n] の n に該当する。この配列の長さは、CartCore クラスのインスタンスを生成した時点での指定値 (29 行目) と整合がとれなければならない (この例では 14×14 になっている)。45 行目では作成したマップを CartCore クラスのインスタンスに代入し、46、47 行目では境界条件を指定している。48 行目では X 軸、49 行目では Y 軸の各領域の長さを指定し、50 行目でそれらを PutWidthXY メソッドにより CartCore クラスのインスタンスに代入する。51 行目では燃料集合体群の情報を定義する AssemblySet クラスのインスタンスを CartCore クラスのインスタンスに代入し、体系情報の作成が完了する。

なお、53 行目以降では、CartCore クラスのインスタンスから CartMeshInfo クラスのインスタンスを作成している。

3.3 CartMeshInfo による体系情報の作成

物質配置が複雑ではない炉心や、一次元、二次元炉心の体系情報は、CartCore クラスを介さず直接 CartMeshInfo クラスで作成する。そこで、ZPPR-9 の円筒炉心モデルを CartMeshInfo クラスのインスタンスで記述した例を**リスト 6** に示す。

Listing 6: CartMeshInfo クラスによる ZPPR-9 二次元円筒モデルの記述

```
1   int main()
2 {
3   // *** ZPPR-9 RZ : Beff calculation ***
4
5   // *** CartMeshInfo ***
6   CartMeshInfo cmi;
7   float xl[]={88.872, 119.949, 142.594, 159.448};
8   float yl[]={50.876, 76.276, 91.516, 104.81};
9   int xm[]={18,6,5,3};
10  int ym[]={10,5,3,2};
11  int mat[]={
12    0,1,2,6,
13    4,4,2,6,
14    5,5,2,6,
15    7,7,7,8
16  };
17  cmi.PutMeshInfo(4,4,xm,ym,xl,yl,mat,"cumulative");
18  cmi.PutBoundaryCondition
19  ("Reflective","Vacuum","Reflective","Vacuum");
20 }
```

7、8 行目では、X(この場合は R)、Y（Z) 軸の粗メッシュの長さを定義しているが、ここでは軸の端からのメッシュ端までの距離で定義している。ここで各メッシュの長さを入力した場合には、17 行目の PutMeshInfo メソッドの最後の引数を ‘width’ とする（もしくは省略する）必要がある。9、10 行目では各軸の粗メッシュにおける詳細メッシュ分割数を定義し、11 から 16 行目までで粗メッシュ分割上での物質配置を定義する。18、19 行目では境界条件を指定している。

参考文献

[1] T.Takeda, H.Ikeda, ‘3-D neutron transport benchmark,’ NEACRP-L-330, OECD/NEA Committee on Reactor Physics (NEACRP), (1991).