RIKEN Center for Computational Science

2025 年 8 月 1 日

有限要素解析のための並列ソルバーに関するスクール 2025

FreeFEM 入門 - 3 次元定常流れ問題を例に

鈴木 厚 [1]

[1] 理研計算科学研究センター 大規模並列数値計算技術研究チーム
atsushi.suzuki.aj@a.riken.jp

疎行列を用いて Posisson 方程式を解く FreeFEM スクリプト

双一次形式 $a(u,v)=\int_\Omega \nabla u\cdot\nabla v\,dx$, L^2 と内積 $(f,v)=\int_\Omega fv$

次を満たす $u_h\in V_h(g)$ を見付けよ $a(u_h,v_h)=(f,v_h)\ \forall v_h\in V_h$.

```
mesh Th=square(20,20);  // (0,1)x(0,1) square domain
fespace Vh(Th,P1);
Vh u,v;
func f = cos(x)*sin(y);
func g = 1.0;
varf poisson(u,v)=int2d(Th)( dx(u)*dx(v)+dy(u)*dy(v) )
                    + on(1,2,3,4,u=g);
varf external(u,v)=int2d(Th)( f*v );
real tgv=1.0e+40; // penalty parameter > (machine eps)^-2
matrix A = poisson(Vh,Vh, tgv=tgv,solver=CG);
real[int] ff = external(0, Vh);
real[int] bc = poisson(0, Vh, tgv=tgv);
ff = bc ? bc : ff; // penalty term for inhomogenise data
u[] = A^-1 * ff;
plot(u);
```

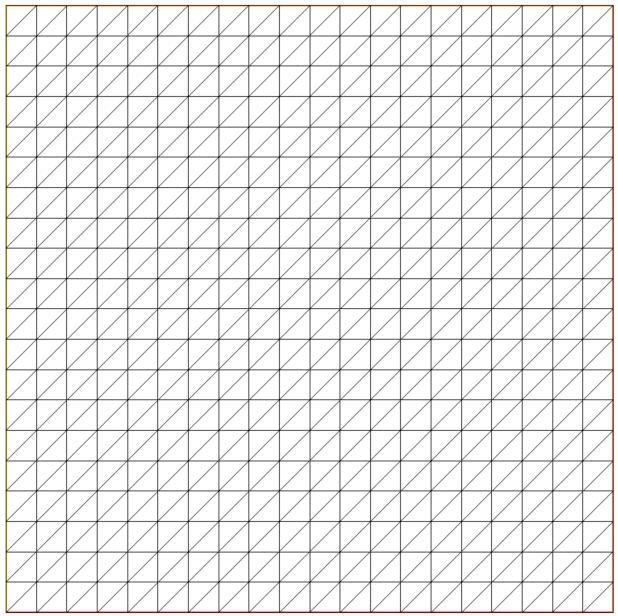
線形方程式に擾乱を加える

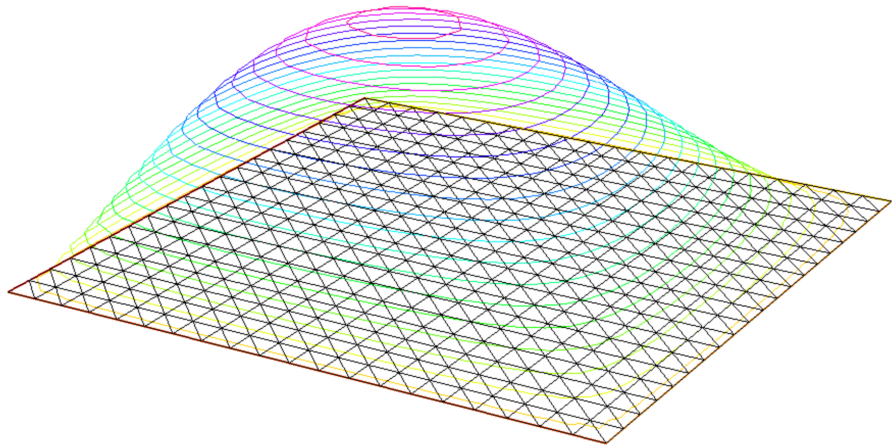
$$\tau u_k+\sum_{j\neq k}a_{kj}u_j=\tau g_k \quad \sum_j a_{ij}u_j=f_i \text{ for } i\neq k$$
$$\sum_j a_{ij}u_j=f_i \qquad \forall i\in\{1,\dots,N\}\setminus\Lambda_D.$$

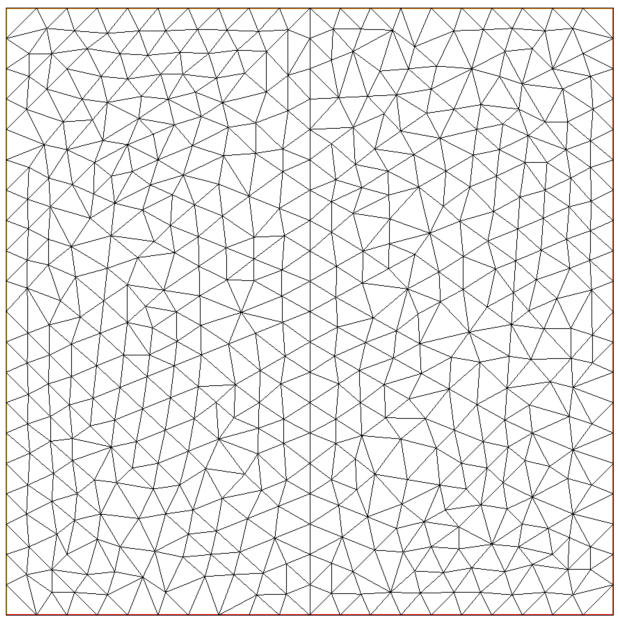
線形ソルバーの指定 `solver=`

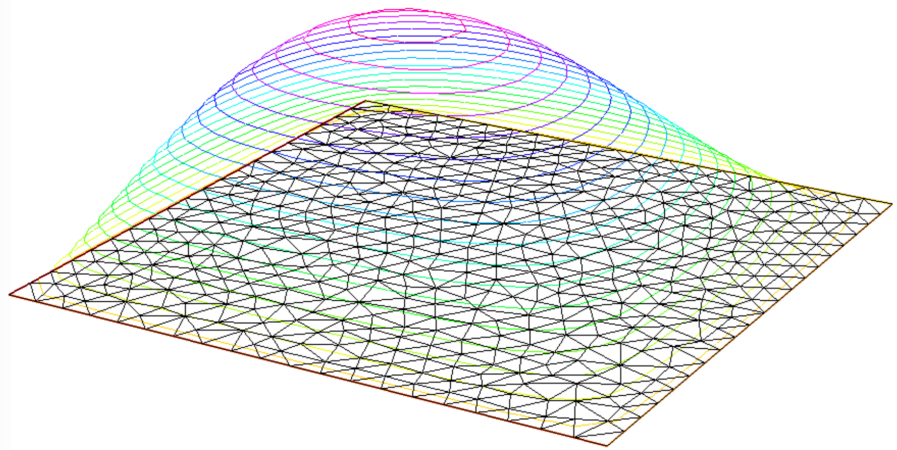
`CG` / `GMRES`	正定値対称行列 / 一般の行列
`sparsesolver`	Pardiso あるいは MUMPS を用いる

構造/非構造メッシュ









非構造メッシュを生成する FreeFEM スクリプト

```
int n1 = 20;
border bottom(t=0,1){x=t;y=0;   label=1;};
border right(t=0,1) {x=1;y=t;   label=2;};
border top(t=0,1)   {x=1-t;y=1; label=3;};
border left(t=0,1)  {x=0;y=1-t; label=4;};
mesh Th1=buildmesh(bottom(n1)+right(n1)+top(n1)
                   +left(n1));
...
fespace Vh10(Th1,P0);
Vh10 h1 = hTriangle;
real hmax = h1[].max;
...
```

メッシュ細分を用いる場合は分割の要素直径に何れを採用するかは議論が必要

$\min_K h_K$, $\sum_K h_K/\#\mathcal{T}_h$ と $\max_K h_K$ を計算しておくと良い

誤差評価を確認する FreeFEM スクリプト

```
int n1 = 20;
real hh1,hh2,err1,err2;
func sol = sin(pi*x)*sin(pi*y/2.0);
func solx = pi*cos(pi*x)*sin(pi*y/2.0);
func soly = (pi/2.0)*sin(pi*x)*cos(pi*y/2.0);
mesh Th1=square(n1,n1);
fespace Vh1(Th1,P2);
// calculation of a solution on Vh1 wth mesh Th1
matrix A = poisson(Vh1, Vh1, tgv=tgv,solver=CG);
real[int] ff = external(0, Vh1);
real[int] bc = poisson(0, Vh1, tgv=tgv);
ff = bc ? bc : ff;
u1[] = A^-1 * ff;
err1 = int2d(Th1)((dx(u1)-solx)*(dx(u1)-solx) +
                  (dy(u1)-soly)*(dy(u1)-soly) +
                  (u1-sol)*(u1-sol));
err1 = sqrt(err1);
// calculation of a solution on Vh2 with mesh Th2
hh1 = 1.0/n1*sqrt(2.0); hh2 = 1.0/n2*sqrt(2.0);
cout<<"O(h^2)="<<log(err1/err2)/log(hh1/hh2)<<endl;
```

誤差評価理論から有限要素解 $u_h \in S_h(Pk)$ の厳密解 $u \in H^2(\Omega)$ に対する誤差は要素サイズ h に関して次の評価がなりたつ

$$||u - u_h||_1 = c\,h^k, \quad \frac{||u - u_{h_1}||_1}{||u - u_{h_2}||_1} = \frac{c h_1^k}{c h_2^k} = \left(\frac{h_1}{h_2}\right)^k$$

FreeFEM の文法

繰り返し

```
for (int i=0; i<10; i++) {
  ...
  if (err < 1.0e-6) break;
}
//
int i = 0;
while (i < 10) {
  ...
  if (err < 1.0e-6) break;
  i++;
}
```

有限要素空間, 双一次形式と疎行列

```
fespace Xh(Th,P1)
Xh u,v;            // finite element data
varf a(u,v)=int2d(Th)( ... ) ;
matrix A = a(Xh,Xh,solver=UMFPACK);
real [int] v;  // array
v = A*u[];     // multiplication matrix to array
```

ユーザー定義関数

```
func real[int] ff(real[int] &pp) { // C++ reference
   ...
   return pp;                      // the same array
}
```

配列, ベクトル, FEM データ, 疎行列, ブロックデータ : 1/2

基本データ型

```
bool flag; // true or false
int i;
real w;
string st = "abc";
```

配列

```
real[int] v(10);  // real array whose size is 10
real[int] u;      // not yet allocated
u.resize(10);     // same as C++ STL vector
real[int] vv = v; // allocated as same size of v.n
a(2)=0.0  ;       // set value of 3rd index
a += b;           // a(i) = a(i) + b(i)
a = b .* c  ;      // a(i) = b(i) * c(i); element-wise
a = b < c ? b : c // a(i) = min(b(i), c(i)); C-syntax
a.sum;            // sum a(i);
a.n;              // size of array
```

ベクトルの $\ell^1, \ell^2, \ell^\infty$-ノルムを計算するための配列に対する操作が max, min に加えて利用できる.

配列, ベクトル, FEM データ, 疎行列, ブロックデータ : 2/2

有限要素オブジェクト

```
func fnc = sin(pi*x)*cos(pi*y); // function with x,y
mesh Th = ...;
fespace Vh(Th,P2);          // P2 space on mesh Th
Vh f;                       // FEM data on Th with P2
f[];                        // access data of FEM DOF
f = fnc;                    // interpolation onto FEM space
fespace Vh(Th,[P2,P2]); // 2 components P2 space
Vh [u1,u2];                 // u1[], u2[] is allocatd
u1[]=0.0;                   // access all data of [u1,u2];
real[int] uu([u1[].n+u2[].n);
u1[] = uu;                  // u1[], u2[] copied from uu
[u1[], u2[]] = uu;          // using correct block data
```

密行列と疎行列

```
real[int,int] B(10,10); // 2D array
varf aa(u,v)=int2d(Th)(u*v); // L2-inner prod. for mass
matrix A=aa(Vh,Vh,solver=sparsesolver); //sparse matrix
```

C++ と同様のファイル入出力,

```
Vh u;
{
  ofstream file("output.data");
  file << u; // with meta data
}  // scope of file object
```

```
Vh u;
{
  ifstream file("saved.data");
  for (int i=0; i<u[].n; i++) {
    file >> u[](i);// only number
  }
}  // scope of file object
```

3 次元での Stokes 方程式 : 1/2

拡散項の弱形式 $a(u,v) = 2\int_\Omega D(u) : D(v)$ は 3×3 テンソル $D(u)$ と $D(v)$ の内積から計算している. $A : B = \mathrm{tr}(AB^T)$ であること

$$A : B = \sum_{i,j} [A]_{i\,j}[B]_{i\,j} = \sum_{k=i}\sum_{j} [A]_{i\,k}[B]_{j\,k} = \mathrm{tr}(AB^T)$$

を用いる. FreeFEM はテンソルの記述するための固有の命令はないため, マクロ機能を用いて実現する.

```
macro Vec(uu) [uu#1, uu#2, uu#3] //
macro d11(uu) dx(uu[0]) //
macro d22(uu) dy(uu[1]) //
macro d33(uu) dz(uu[2]) //
macro d12(uu) ((dy(uu[0]) + dx(uu[1])) / 2.0) //
macro d13(uu) ((dz(uu[0]) + dx(uu[2])) / 2.0) //
macro d23(uu) ((dz(uu[1]) + dy(uu[2])) / 2.0) //
macro Deform(uu) [ [d11(uu), d12(uu), d13(uu)],
                   [d12(uu), d22(uu), d23(uu)],
                   [d13(uu), d23(uu), d33(uu)] ]//
macro DDOTS(aa, bb) trace(aa * bb')//
macro DOT(aa, bb) ((aa)' * (bb))//

varf a([u1, u2, u3], [v1, v2, v3]) =
int3d(Th,qfV=qfV2)( DDOTS(Deform(Vec(u)), Deform(Vec(v))) );
```

3 次元での Stokes 方程式 : 2/2

発散の弱形式 $b(v,p) = -\int_\Omega \nabla \cdot v\, p$ はベクトルの有限要素関数から発散を計算するマクロを定義して記述する.

```
macro Div(uu) (dx(uu[0]) + dy(uu[1]) + dz(uu[2]))//
varf b([v1, v2, v3], p) =
int3d(Th,qfV=qfV2)( - Div(Vec(v)) * p);
```

流速 3 成分と圧力の合計 4 成分をまとめて Stokes 方程式の弱形式は次のように記述される.

```
varf StokesStiff([u1, u2, u3, p], [v1, v2, v3, q])
= int3d(Th,qfV=qfV2)(
    DDOTS(Deform(Vec(u)), Deform(Vec(v)))
  - Div(Vec(v)) * p
  + Div(Vec(u)) * q
  + delta*hTriangle*hTriangle* grad(p)'*grad(q)) // stabilization
  + on(1, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // surface
  + on(2, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // top
  + on(3, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // bottom
  + on(4, u1 = 1.0, u2 = 0.0, u3 = 0.0)  // inlet
  + on(6, u1 = 0.0, u2 = 0.0, u3 = 0.0); // both sides
```

3 次元での Navier-Stokes 方程式 : 1/2

Navier-Stoks 方程式は Newton 法による反復計算を行なうために, Jacobi 行列を手計算して線形化して弱形式を得ているので, 前のステップでの流速 up を用いて, 次のように記述できる.

```
macro Convect(ww, uu) (ww[0] * dx(uu) +
                       ww[1] * dy(uu) + ww[2] * dz(uu)) //
macro ConvectVector(ww, uu) [Convect(ww, uu[0]),
                     Convect(ww, uu[1]), Convect(ww, uu[2])] //
varf NSNewtonStiff([u1, u2, u3, p], [v1, v2, v3, q])
= int3d(Th,qfV=qfV2)(
    DOT(ConvectVector(Vec(up), Vec(u)), Vec(v))
  + DOT(ConvectVector(Vec(u), Vec(up)), Vec(v))
  + nu * DDOTS(Deform(Vec(u)), Deform(Vec(v)))
  - Div(Vec(v)) * p
  + Div(Vec(u)) * q
  + delta*hTriangle*hTriangle* grad(p)'*grad(q)) // stabilization'
  + on(1, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // surface
  + on(2, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // top
  + on(3, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // bottom
  + on(4, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // inlet
  + on(6, u1 = 0.0, u2 = 0.0, u3 = 0.0); // both sides
```

Dirichlet 境界条件は流入部分 (4 番の添字) で斉次条件を課していることに注意する.

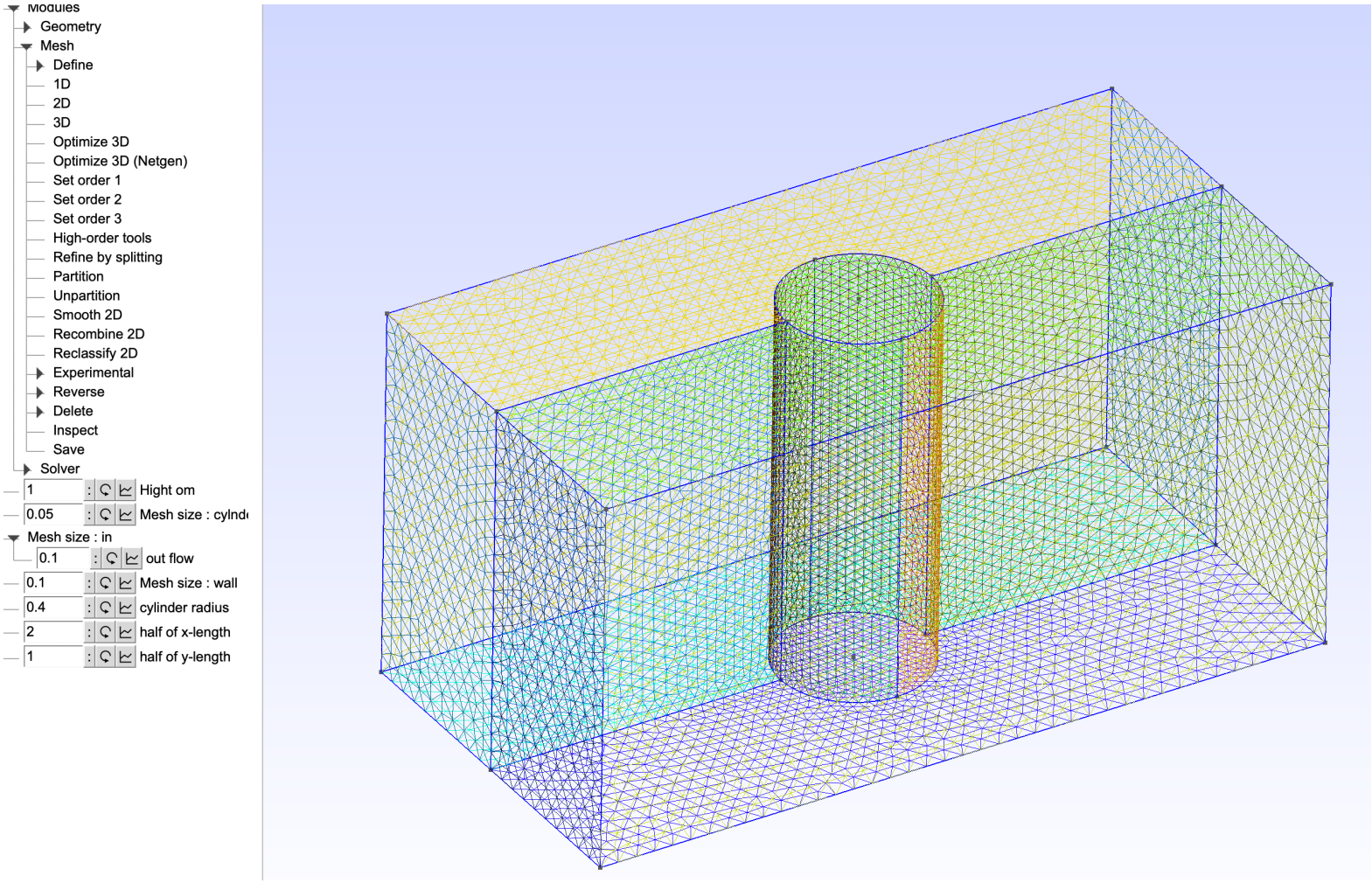
3 次元での Navier-Stokes 方程式 : 2/2

Newton 法による反復計算で右辺は前のステップ up と pp の残差となるので, 次のように記述できる.

```
varf NSNewtonRHS([u1, u2, u3, p], [v1, v2, v3, q])
= int3d(ThG,qfV=qfV2)(
    DOT(ConvectVector(Vec(up), Vec(up)), Vec(v))
  + nu * DDOTS(Deform(Vec(up)), Deform(Vec(v)))
  - Div(Vec(v)) * pp
  + Div(Vec(up)) * q
  + delta*hTriangle*hTriangle* grad(pp)'*grad(q)) // stabilization
  + on(1, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // surface
  + on(2, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // top
  + on(3, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // bottom
  + on(4, u1 = 0.0, u2 = 0.0, u3 = 0.0)  // inlet
  + on(6, u1 = 0.0, u2 = 0.0, u3 = 0.0); // both sides
```

Gmsh による計算領域の記述と四面体要素分割

3 次元領域の有限要素計算のための四面体要素分割は Gmsh ソフトウェアにより作製することができる. パラメトライズされた境界面によって領域を構成しその内部に四面体要素を生成する.



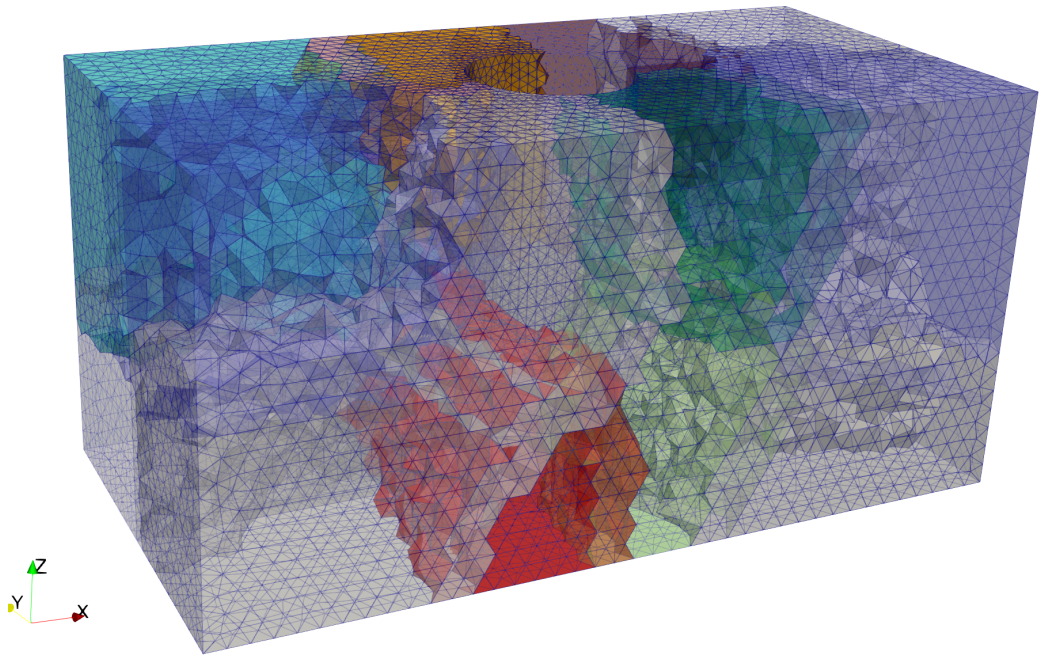
METIS による重なりのある領域分割と HPDDM による解法: 1/3

```
load "hpddm"
mpiComm comm(mpiCommWorld,0,0);
mesh3 ThGlobal, Th;
ThGlobal = gmshload3("cylinder.msh"); // read Gmsh mesh
func Pk4 = [P1, P1, P1, P1];
fespace Vhg(ThGlobal, Pk4);      // FE space on global domain
fespace Vh(Th, Pk4);  // FE space on local domain
int[int][int] intersection;
real[int] D;
{
  Th = ThGlobal;
  build(Th, 1, intersection, D, Pk, comm, 4)
}
matrix Rgl = interpolate(Vh, Vhg);
matrix Dh = [D]; // diagonal matrix
```

- `mesh3 Th`：$\widetilde{\Omega_p}$：一層の重なりのある領域分割
- `matrix Rgl`：全体自由度から部分自由度への制限作用素 R_p
- `matrix D`：離散的な単位の分解の重み

METIS による重なりのある領域分割と HPDDM による解法: 2/3

16 部分領域への分割, ParaView による可視化



- macro `build()`：HPDDM の組み込みマクロで重なりのある領域分割を生成する
- `METIS` は双対要素分割による有限要素の領域を実現するが, 領域間の境界は滑らかでなはい
- 加法的 Schwarz 前処理は滑らかな境界を持つ領域分割でなくても堅牢な収束を実現する

METIS による重なりのある領域分割と HPDDM による解法: 3/3

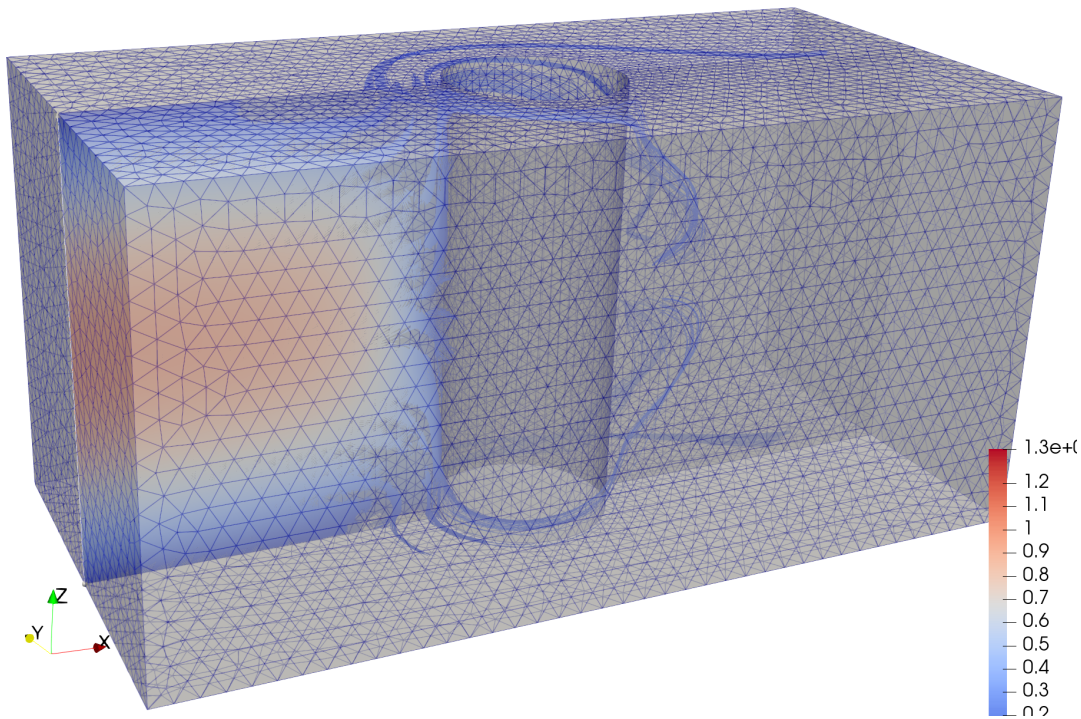
```
load "hpddm"
// ...
fespace Vh(Th, Pk4);  // FE space on local domain
// ...
int totaldof = VhG.ndof;
int localdof = Vh.ndof;
real[int] wwl(localdof), wl(localdof), wg(totaldof);
matrix AST = StokesStiff(Vh, Vh, tgv = -1);
real[int] bST = StokesRHS(0, Vh, tgv = -1);
schwarz dAST(AST, intersection, D);
set (dAST, sparams = "-hpddm_schwarz_method ras " +
                     "-hpddm_schwarz_coarse_correction balanced "+
                     "-hpddm_variant right "+
                     "-hpddm_verbosity 10 "+
                     "-hpddm_tol 1.0e-12 " +
                     "-hpddm_max_it 60");
DDM(dAST, bST, wwl);^^I^^I
wl = Dh * wwl;
wg = Rgl' * wl;
mpiAllReduce(wg, up1[], comm, mpiSUM);
```

- 剛性行列は双一次形式から部分領域の要素分割 `Th` と有限要素空間 `Vh` により並列に生成される
- 全体の解は MPI の総和演算 $\sum_p R_p^T D_p u_p$ により部分領域の解をまとめることで得られる

Paraview による可視化の例

3 次元問題では FreeFEM は, 入力した四面体要素分割のデータを用いスクリプト言語によって記述された弱形式から剛性行列を生成することに特化している. 出力結果は `vtk` ファイルに書き出し, `ParaView` により可視化する.

```
int[int] vtkorder = [1, 1 ,1];
if (mpirank == 0) {
  savevtk("NS.vtk", ThG, [up1, up2, up3], order = vtkorder);
```



FreeFEM とオープンソフトウェア

最新の FreeFem のバージョンは 4.15.
インストール方法は https://doc.freefem.org/introduction/installation.html

- `FreeFEM` : 高レベルの複合物理シミュレーションのための有限要素法ソフトウェア
  `https://freefem.org`
- `gmsh` : 3 次元のメッシュ生成ソフトウェア, 領域記述のためのスクリプト言語を用いるか, `STEP` ファイルを `OpenCASCADE` エンジンにより解釈
  `https://gmsh.info`
- `mmg3d` : 要素細分のための 3 次元のメッシュソフトウェア
  `https://www.mmgtools.org`
- `hpddm` : 加法的 Schwarz 前処理などの領域分割を用いるソルバー
  `https://github.com/hpddm/hpddm`
- `paraview` : 3 次元の可視化ソフトウェア
  `https://www.paraview.org`

外部ソフトウェアの FreeFEM へのインターフェースは
FreeFem-soucres/plugin/{seq,mpi}/
FreeFEM のスクリプトの例は FreeFem-soucres/examples/