Introduction of FreeFEM : a domain specific language and example on nonlinear elasticity solver

Atsushi Suzuki[1]

[1]R-CCS, Large-scale Parallel Numerical Computing Technology Research Team
atsushi.suzuki.aj@a.riken.jp

FreeFEM script to solve Poisson using matrix

bilinear form : $a(u,v) = \int_\Omega \nabla u \cdot \nabla v$, L^2 inner product : $(f,v) = \int_\Omega fv$

Find $u_h \in V_h(g)$ s.t. $a(u_h, v_h) = (f, v_h)\ \forall v_h \in V_h$.

```
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);
```

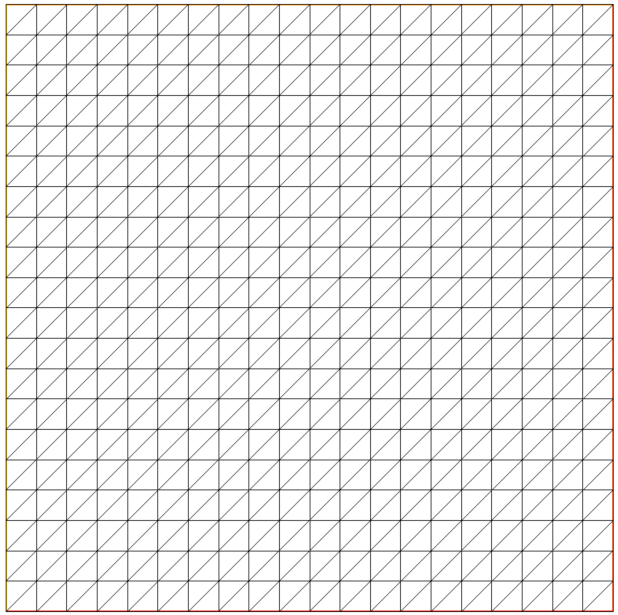
penalty term added to the linear system

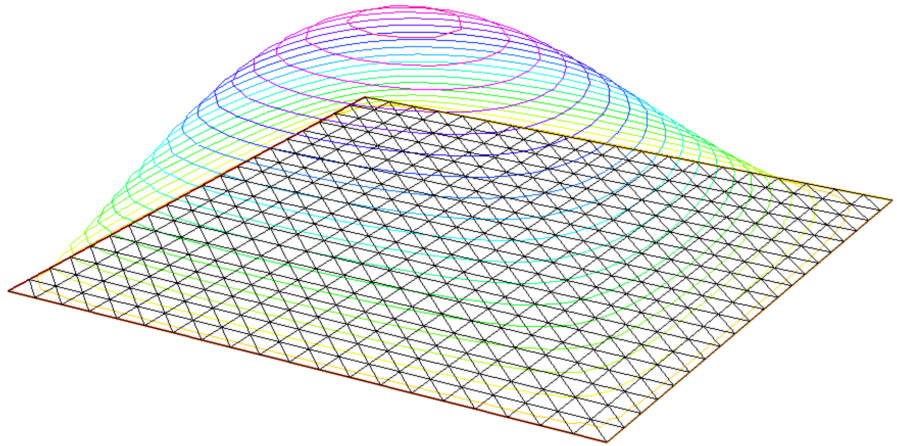
$$\tau u_k + \sum_{j\neq k} a_{kj} u_j = \tau g_k \quad \sum_j a_{ij} u_j = f_i \text{ for } i \neq k$$
$$\sum_j a_{ij} u_j = f_i \qquad \forall i \in \{1,\ldots,N\} \setminus \Lambda_D.$$

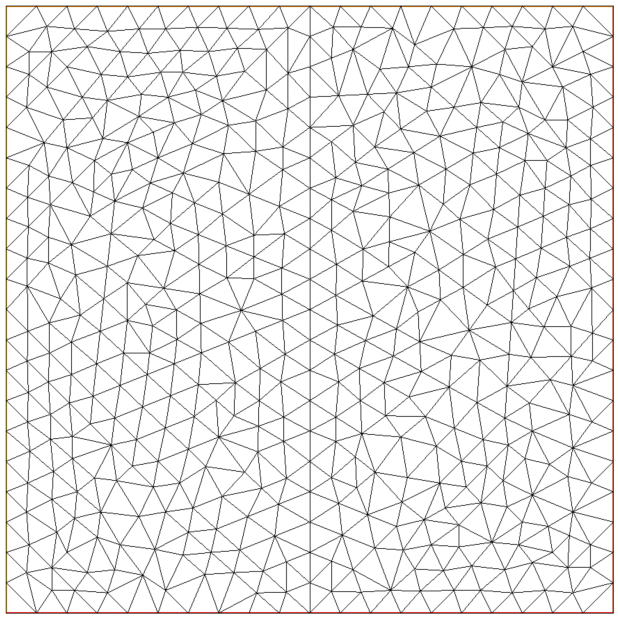
useful liner solver; `solver=`

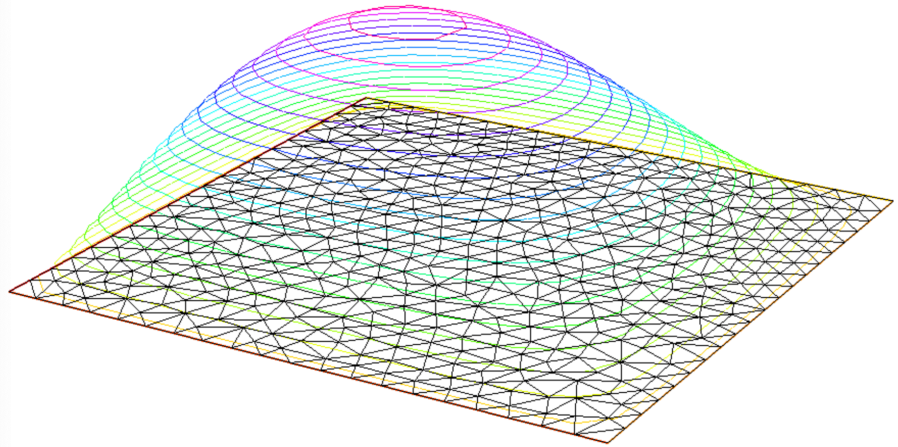
`CG` / `GMRES`	iterative solver for SPD / general matrix
`sparsesolver`	direct solvers to call PARDISO or MUMPS

structured/unstructured mesh









FreeFEM script to generate unstructured mesh

```
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;
...
```

It is better to look $\min_K h_K$, $\sum_K h_K/\#\mathcal{T}_h$, $\max_K h_K$, corresponding to mesh refinement.

FreeFEM script for error estimation

```
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;
```

theory on error estimation says, for the exact solution $u \in H^2(\Omega)$ and finite element solution $u_h \in S_h(P_k)$ with mesh size h

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

syntax of FreeFem++ script

loops

```
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++;
}
```

finite element space, variational form, and matrix

```
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
```

procedure (function)

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

array, vector, FEM data, sparse matrix, block data : 1/2

fundametal data types

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

array

```
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
```

There are other operations such as $\ell^1, \ell^2, \ell^\infty$-norms, max, min.

array, vector, FEM data, sparse matrix, block data : 2/2

FEM data

```
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
```

dense and sparse matrices

```
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
```

file I/O is same as C++,

```
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
```

linear elasticity problem in 2D : 1/2

$$\int_\Omega \Sigma(e(u)) : e(v) = \int_\Omega \lambda \mathsf{div}(u)\mathsf{div}(v) + 2\mu e(u) : e(v)$$

a trick to write a product of tensors by one of matrix and vectors

$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{12} & b_{22} \end{bmatrix}$.

$$\begin{aligned} \Sigma(A) : B &= (\lambda \mathsf{tr} A\, I + 2\mu A) : B \\ &= \begin{bmatrix} \lambda(a_{11}+a_{22}) + 2\mu a_{11} & 2\mu a_{12} \\ 2\mu a_{12} & \lambda(a_{11}+a_{22}) + 2\mu a_{22} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} \\ b_{12} & b_{22} \end{bmatrix} \\ &= \begin{bmatrix} \lambda+2\mu & 0 & \lambda \\ 0 & \mu & 0 \\ \lambda & 0 & \lambda+2\mu \end{bmatrix} \begin{bmatrix} a_{11} \\ 2a_{12} \\ a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} \\ 2b_{12} \\ b_{22} \end{bmatrix} \end{aligned}$$

$[e(u)] \leftrightarrow [\partial_1 u_1, (\partial_1 u_2 + \partial_2 u_1), \partial_2 u_2]$

$$\Sigma(e(u)) : e(v) \leftrightarrow \begin{bmatrix} \lambda+2\mu & 0 & \lambda \\ 0 & \mu & 0 \\ \lambda & 0 & \lambda+2\mu \end{bmatrix} \begin{bmatrix} \partial_1 u_1 \\ \partial_1 u_2 + \partial_2 u_1 \\ \partial_2 u_2 \end{bmatrix} \cdot \begin{bmatrix} \partial_1 v_1 \\ \partial_1 v_2 + \partial_2 v_1 \\ \partial_2 v_2 \end{bmatrix}$$

3×3 matrix with Lamé constants is symmetric

linear elasticity problem in 2D : 2/2

```
macro EL(u1, u2) [dx(u1), (dx(u2)+dy(u1)), dy(u2)] //
mesh Th = square(40, 10, [5*x, y]);
int[int] llabel = [1, 2, 2, 2, 3, 2, 4, 1];
Th = change(Th, label = llabel);
fespace Vh(Th, [P2, P2]);
real lambda = 1.0, mu = 1.0, gg = -0.5e-3;
func A = [ [lambda + 2.0 * mu, 0.0,              lambda],
           [              0.0, mu ,                 0.0],
           [lambda,            0.0, lambda + 2.0 * mu] ];
Vh [u1, u2], [v1, v2];

varf lelasticity([u1, u2], [v1, v2])
= int2d(Th)(EL(u1, u2)' * A * EL(v1, v2)) //'
+ on (1, u1 = 0.0, u2 = 0.0);

varf rhs([u1, u2], [v1, v2])
= int2d(Th)(v2 * gg)  + on (1, u1 = 0.0, u2 = 0.0);

matrix AA = lelasticity(Vh, Vh);
set(AA, solver = "PARDISO", sym = 1);
real[int] bb = rhs(0, Vh);
u1[] = AA^-1 * bb;

mesh Thm = movemesh(Th, [x + u1, y + u2]);
plot(Thm);
```

linear elasticity problem in 3D : 1/3

$$\int_\Omega \Sigma(e(u)) : e(v) = \int_\Omega \lambda \mathrm{div}(u)\mathrm{div}(v) + 2\mu e(u) : e(v)$$

a trick to write a product of tensors by one of matrix and vectors

$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{bmatrix}$.

$\Sigma(A) : B = (\lambda \mathrm{tr} A\, I + 2\mu A) : B$

$$= \begin{bmatrix} \lambda(\sum_k a_{kk}) + 2\mu a_{11} & 2\mu a_{12} & 2\mu a_{13} \\ 2\mu a_{12} & \lambda(\sum_k a_{kk}) + 2\mu a_{22} & 2\mu a_{23} \\ 2\mu a_{13} & 2\mu a_{23} & \lambda(\sum_k a_{kk}) + 2\mu a_{33} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{31} & b_{23} & b_{33} \end{bmatrix}$$

$$= \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & & & \\ & \mu & & & & \\ & & \mu & & & \\ \lambda & & & \lambda+2\mu & \lambda & \\ & & & & \mu & \\ \lambda & & & \lambda & & \lambda+2\mu \end{bmatrix} \begin{bmatrix} a_{11} \\ 2a_{12} \\ 2a_{13} \\ a_{22} \\ 2a_{23} \\ a_{33} \end{bmatrix} \cdot \begin{bmatrix} b_{11} \\ 2b_{12} \\ 2b_{13} \\ b_{22} \\ 2b_{23} \\ b_{33} \end{bmatrix}$$

$$[e(u)] \leftrightarrow \left[\partial_1 u_1, (\partial_1 u_2 + \partial_2 u_1), (\partial_1 u_3 + \partial_3 u_1), \partial_2 u_2, (\partial_2 u_3 + \partial_3 u_2), \partial_3 u_3\right]$$

linear elasticity problem in 3D : 2/4

```
macro EL(u1,u2,u3) [dx(u1),(dx(u2)+dy(u1)),(dx(u3)+dz(u1)),
                                  dy(u2),(dy(u3)+dz(u2)),dz(u3)] //
include "cube.idp"
int[int]  Nxyz=[40,10,10];
real [int,int]  Bxyz=[[0.,5.],[0.,1.],[0.,1.]];
int [int,int]  Lxyz=[[1,2],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);
fespace Vh(Th, [P2, P2, P2]);
real lambda = 1.0, mu = 1.0, gg = -0.5e-3;
real a1 = lambda + 2.0 * mu;
real a2 = mu;
real a3 = lambda;
func A  = [ [a1,  0.0, 0.0, a3,  0.0, a3],
            [0.0, a2,  0.0, 0.0, 0.0, 0.0],
            [0.0, 0.0, a2,  0.0, 0.0, 0.0],
            [a3,  0.0, 0.0, a1,  0.0, a3],
            [0.0, 0.0, 0.0, 0.0, a2,  0.0],
            [a3,  0.0, 0.0, a3,  0.0, a1]  ]; //

Vh [u1, u2, u3], [v1, v2, v3];
varf lelasticity([u1, u2, u3], [v1, v2, v3])
= int3d(Th,qfV=qfV1)(EL(u1, u2, u3)' * A * EL(v1, v2, v3)) //'
+ on (1, u1 = 0.0, u2 = 0.0, u3 = 0.0);

varf rhs([u1, u2, u3], [v1, v2, v3])
= int3d(Th,qfV=qfV1)(v3 * gg)
+ on(1, u1 = 0.0, u2 = 0.0, u3 = 0.0);
```

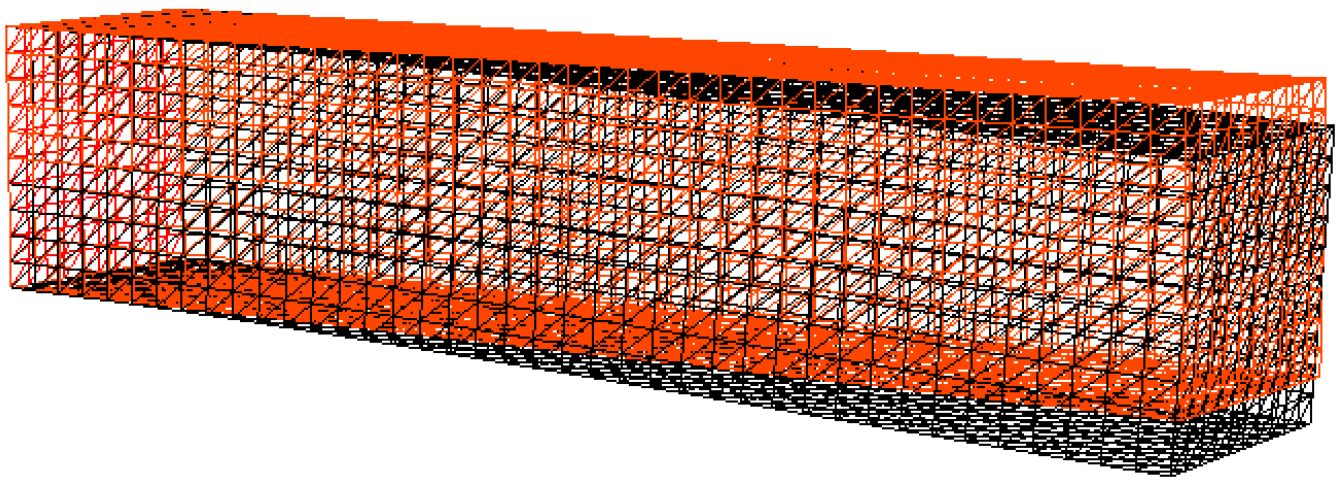
linear elasticity problem in 3D : 3/4

```
load "PARDISO"

matrix AA = lelasticity(Vh, Vh);
set(AA, solver = "PARDISO", sym = 1);
real[int] bb = rhs(0, Vh);
real[int] xx = AA^-1 * bb;
u1[] = xx;

mesh3 Thm = movemesh3(Th,transfo=[x + u1, y + u2, z + u3]);
plot(Th, Thm);
```

3D beam : displacement on the left wall is fixed



nonlinear elasticity : symmetric Jacobian matirx

$$\int_\Omega \Sigma(dE(u^n)[w]) : dE(u^n)[v] + \Sigma(E(u^n)) : d^2E(u^n)[v,w]$$

```
macro ENL(u1, u2, u3) [
  (dx(u1)*dx(u1)+dx(u2)*dx(u2)+dx(u3)*dx(u3))*0.5,
  (dx(u1)*dy(u1)+dx(u2)*dy(u2)+dx(u3)*dy(u3)),
  (dx(u1)*dz(u1)+dx(u2)*dz(u2)+dx(u3)*dz(u3)),
  (dy(u1)*dy(u1)+dy(u2)*dy(u2)+dy(u3)*dy(u3))*0.5, // ...
macro dENL(u1, u2, u3, v1, v2, v3) [
  (dx(u1)*dx(v1)+dx(u2)*dx(v2)+dx(u3)*dx(v3)),
  (dx(u1)*dy(v1)+dx(v1)*dy(u1)+
   dx(u2)*dy(v2)+dx(v2)*dy(u2)+
   dx(u3)*dy(v3)+dx(v3)*dy(u3)),
  (dx(u1)*dz(v1)+dx(v1)*dz(u1)+
   dx(u2)*dz(v2)+dx(v2)*dz(u2)+
   dx(u3)*dz(v3)+dx(v3)*dz(u3)),
  (dy(u1)*dy(v1)+dy(u2)*dy(v2)+dy(u3)*dy(v3)), //...
macro E(u1, u2, u3) (EL(u1, u2, u3) + ENL(u1, u2, u3)) //
macro dE(u1, u2, u3, v1, v2, v3) (EL(v1, v2, v3)+
                                  dENL(u1, u2, u3, v1, v2, v3)) //
varf nelasticity([w1, w2, w3], [v1, v2, v3])
= int3d(Th,qfV=qfV1)(dE(u1, u2, u3, w1, w2, w3)' * A *
                     dE(u1, u2, u3, v1, v2, v3)) +
                     E(u1, u2, u3)' * A *
                     dENL(w1, w2, w3, v1, v2, v3)) + on( ... //
```

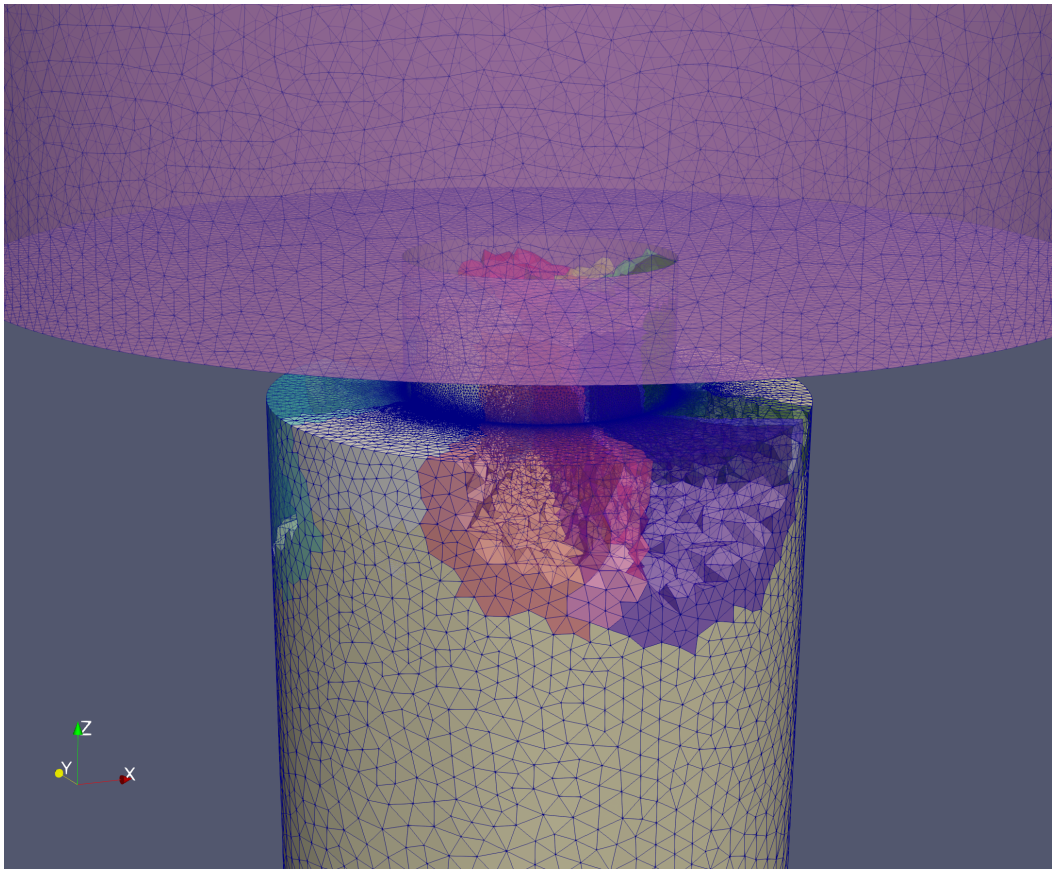
overlapping domain decomposition by METIS and HPDDM: 1/3

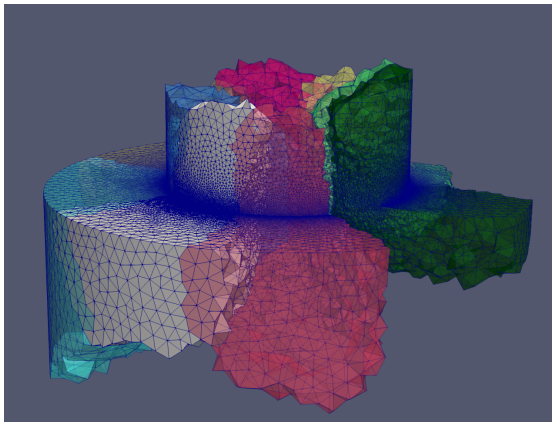
```
load "hpddm"
load "gmsh"
include "macro_ddm.idp"                    // additional DDM functions
mpiComm comm(mpiCommWorld,0,0);
mesh3 ThGlobal, Th;
ThGlobal = gmshload3("cylinders.msh"); // read Gmsh mesh
func Pk = [P1, P1, P1];
fespace Vhg(ThGlobal, Pk);      // FE space on global domain
fespace Vh(Th, Pk);  // FE space on local domain
int[int][int] intersection;
real[int] D;
{
  Th = ThGlobal;
  build(Th, 1, intersection, D, Pk, comm, 3)
}
matrix Rgl = interpolate(Vh, Vhg);
matrix Dh = [D]; // diagonal matrix
```

- `mesh3 Th` : $\widetilde{\Omega_p}$: one layer extended overlapping subdomain
- `matrix Rgl` : restriction operator R_p, global to local
- `matrix D` : diagonal matrix of discrete partition of the unity

overlapping domain decomposition by METIS and HPDDM: 2/3

12 subdomains are generated, visualized by ParaView:





- macro `build()` in HPDDM generates overlapped subdomains
- Since `METIS` provides dual-graph decomposition, interface between subodmains is not smooth in general
- additive Schwarz preconditioner is robust even with non-smooth interfaces

overlapping domain decomposition by METIS and HPDDM: 3/3

HPDDM is a C++ package for additive Schwarz domain decomposition including advanced preconditioner by Dr P. Jolivet @ LIP6, Sorbonne U.

```
load "hpddm"
mpiComm comm(mpiCommWorld,0,0);
mesh3 ThGlobal, Th;
fespace Vh(Th, Pk);  // FE space on local domain
// ...
totaldof = Vhg.ndof;
localdof = Vh.ndof;
real[int] xxg(totaldof), xdl(localdof), xxl(localdof);
matrix AA = lelasticity(Vh, Vh, tgv = -1, sym=1);
real[int] bb = rhs(0, Vh, tgv= -1);
schwarz dA(AA, intersection, D);
set(dA, sparams = "-hpddm_schwarz_method ras " +
  "-hpddm_variant right " +
  "-hpddm_gmres_restart 150 -hpddm_max_it 150" +
  "-hpddm_verbosity 10 -hpddm_tol 1.0e-10")
xxl = AA^-1 * bb;     // solution is distributed
xdl = Dh * xxl;       // adjust with weights for overlapping
xxg = Rgl' * xdl;      //'  local -> global
mpiAllReduce(wg, u1[], comm, mpiSUM); //
```

- stiffness matrix is generated from bilinear form locally `Th` and `Vh`
- global solution is obtained by mpi reduction for $\sum_p R_p^T D_p u_P$
- GeNEO frame work by Prof. F. Nataf provides efficient coarse space

interfacing to PETSc : 1/3

Interface between FreeFEM and PETSc is developed by Dr P. Jolivet

```
load "PETSc"
load "gmsh"
macro dimension()3// EOM                    // 2D or 3D
include "macro_ddm.idp"                     // additional DDM functions
macro def(u)[u, u#B, u#C]// EOM             // scalar field definition
macro init(i)[i, i, i]// EOM                // scalar field initialization
mesh3 Th = gmshload3("cylinders.msh");
func Pk = [P1, P1, P1];
Mat matA;
MatCreate(Th, matA, Pk);
fespace Vh(Th, Pk);
fespace Wh(Th, P0);
// ...
Vh [u1, u2, u3];
matrix AA = linearelas(Vh, Vh, tgv = -1);
real[int] bb = rhs(0, Vh, tgv= -1);
matA = AA;
set(matA, sparams = "-ksp_view -ksp_monitor -ksp_rtol 1e-6 " +
                    "-pc_type hypre");
u1[] = matA^-1 * bb;
```

- `Mat` : PETSc distributed matrix
- `MatCreate` generates domain decomposition of FreeFEM mesh and connects to decomposed matrix of PETSc
- `matA` is generated by `varf linearelas()` locally and solution is stored in local FEM data `[u1, u2, u3]`

computational efficiency

linear and nonlinear elasticity problem with $2,076,809$ DOF

solver	linear			memory
	matrix	RHS	solver	
`pardiso` symmetric	52.17	0.04	432.50	40.2GB
`hpddm` unsymmetric	20.30	0.01	352.69	5.1G$\times 12$
`hypre` unsymmetric	20.33	0.01	72.03	1.5G$\times 12$

solver	nonlinear			memory
	matrix	RHS	solver	
`pardiso` symmetric	233.34	129.33	430.60	40.2GB
`hpddm` unsymmetric	42.44	22.15	366.02	5.1G$\times 12$
`hypre` unsymmetric	45.01	20.1	87.42	1.5G$\times 12$

- generation of stiffness matrix for nonlinear elasticity problem consumes non-negligible time
- `hypre` multigrid preconditioner does not converge for more higher nonlinearity

References to opensoruce softwares

the latest version of FreeFem is 4.15.
installation guid is found at
https://doc.freefem.org/introduction/installation.html

- `FreeFEM` : A high level multiphysics finite element software
  `https://freefem.org`
- `gmsh` : 3d mesh generator, geometry discribed by own script language or STEP file with `OpenCASCADE` engine
  `https://gmsh.info`
- `mmg3d` : 3d mesh regenerator, for mesh refinement
  `https://www.mmgtools.org`
- `hpddm` : parallel linear solver with additive Schwarz preconditioner
  `https://github.com/hpddm/hpddm`
- `paraview` : 3d visualization software
  `https://www.paraview.org`

intrefaces to FreeFEM are developed in FreeFem-soucres/plugin/{seq,mpi}/
examples of FreeFEM scripts are located in FreeFem-soucres/examples/
interface to paraview in FreeFEM

```
load "iovtk"
mesh3 Th = ... ;
fespace Vh(Th, P1);
Vh u;
...
int[int] vtkorder = [1]; // FE order P1 or P2 of components
savevtk("output.vtk", Th, u, order = vtkorder);
```