

FreeFem++, 3d tools for PDE simulation

F. Hecht

Laboratoire Jacques-Louis Lions
Université Pierre et Marie Curie
Paris, France

with O. Pironneau, J. Morice

<http://www.freefem.org>

<mailto:hecht@ann.jussieu.fr>

With the support of ANR (French gov.)

ANR-07-CIS7-002-01

<http://www.freefem.org/ff2a3/>

<http://www-anr-ci.cea.fr/>



PLAN

- Introduction [FreeFem++](#)
- some syntaxe
- Mesh generation
- Variational formulation
- Poisson equation in 3D
- Poisson equation with matrix
- Time dependent problem
- No linear problem
- some Trick
- Variational/Weak form (Matrix and vector)
- mesh generation in 3d
- Stokes variational Problem
- Navier-Stokes
- Dynamic Link example (hard)
- Conclusion / Future

<http://www.freefem.org/>

Introduction

FreeFem++ is a software to solve numerically partial differential equations (PDE) in \mathbb{R}^2 and in \mathbb{R}^3 with finite elements methods. We used a user language to set and control the problem. The FreeFem++ language allows for a quick specification of linear PDE's, with the variational formulation of a **linear steady state problem** and the user can write they own script to solve no linear problem and time depend problem. You can solve coupled problem or problem with moving domain or eigenvalue problem, do mesh adaptation , compute error indicator, etc ...

FreeFem++ is a freeware and this run on Mac, Unix and Window architecture, in parallele with MPI.

The main characteristics of FreeFem++ I/II (2D)

- Wide range of finite elements : linear (2d,3d) and quadratic Lagrangian (2d,3d) elements, discontinuous P1 and Raviart-Thomas elements (2d,3d), **3d Edge element** , vectorial element , mini-element(2d, 3d), ...
- **Automatic interpolation** of data from a mesh to an other one, so a finite element function is view as a function of (x, y, z) or as an array.
- Definition of the problem (**complex or real value**) with the variational form with access to the vectors and the matrix if needed
- Discontinuous Galerkin formulation (only 2d to day).

The main characteristics of FreeFem++ II/II (2D)

- Analytic description of boundaries, with specification by the user of the intersection of boundaries in 2d.
- **Automatic mesh generator**, based on the Delaunay-Voronoi algorithm. (2d, **3d**)
- load and save Mesh, solution
- **Mesh adaptation based on metric**, possibly anisotropic, with optional automatic computation of the metric from the Hessian of a solution.
- **LU, Cholesky, Crout, CG, GMRES, UMFPack, SuperLU, MUMPS, ...** sparse linear solver; **eigenvalue** and eigenvector computation with ARPACK.
- Online graphics, C++ like syntax.
- Link with other soft : modulef, emc2, medit, gnuplot, tetgen, superlu, mumps ...
- Dynamic linking to add functionality.
- Wide range of of examples : Navier-Stokes **3d**, elasticity **3d**, fluid structure, eigenvalue problem, Schwarz' domain decomposition algorithm, residual error indicator, ...

Element of syntaxe 1/2

```
x,y,z , label, N.x, N.y, N.z , // current coordinate, label, normal
int i = 0; // an integer
real a=2.5; // a reel
bool b=(a<3.);
real[int] array(10); // a real array of 10 value
mesh Th; mesh3 Th3; // a 2d mesh and a 3d mesh
fespace Vh(Th,P2); // a 2d finite element space;
fespace Vh3(Th3,P1); // a 3d finite element space;
Vh u=x; // a finite element function or array
Vh3<complex> uc = x+ 1.i *y; // complex valued FE function or array
u(.5,.6,.7); // value of FE function u at point (.5,.6,.7)
u[]; // the array associated to FE function u
u[][5]; // 6th value of the array ( numbering begin at 0 like in C)
```

Element of syntaxe 1/2

```
fespace V3h(Th, [P2,P2,P1]) ;
Vh [u1,u2,p]=[x,y,z] ; // a vectorial finite element function or array
// remark u1[] <==> u2[] <==> p[] same array of unknown.
macro div(u,v) (dx(u)+dy(v))// EOM
macro Grad(u) [dx(u),dy(u)]// EOM
varf a([u1,u2,p],[v1,v2,q])=
  int2d(Th)( Grad(u1)'*Grad(v1) +Grad(u2)'*Grad(v2)
           -div(u1,u1)*q -div(v1,v2)*p)
  +on(1,2)(u1=g1,u2=g2) ;

matrix A=a(V3h,V3h,solver=UMFPACK) ;
real[int] b=a(0,V3h) ;
u[] =A^-1*b ; //
func f=x+y ; // a formal line function
func real g(int i, real a) { ..... ; return i+a ;}
A = A + A' ; A = A'*A // matrix operation (only one by one operation)
A = [ A,0],[0,A'] ; // Block matrix.
```

Build Mesh or read mesh

```
include "MeshSurface.idp" // tool for 3d surfaces meshes
mesh3 Th ;
try { Th=readmesh3("Th-hex-sph.mesh") ; } // try to read
catch(...) { // catch an error to build the mesh...
  real hs = 0.2 ; // mesh size on sphere
  int[int] NN=[11,9,10] ;
  real [int,int] BB=[[-1.1,1.1],[-.9,.9],[-1,1]] ; // Mesh Box
  int [int,int] LL=[[1,2],[3,4],[5,6]] ; // Label Box
  mesh3 ThHS = SurfaceHex(NN,BB,LL,1)+Sphere(0.5,hs,7,1) ; // "gluing"
// surface meshes
  real voltet=(hs^3)/6. ; // volume mesh control.
  real[int] domaine = [0,0,0,1,voltet,0,0,0.7,2,voltet] ;
  Th = tetg(ThHS,switch="pqaAAYYQ",nbofregions=2,regionlist=domaine) ;
  savemesh(Th,"Th-hex-sph.mesh") ; }
```

Build form a extern file mesh

```
mesh3 Th2("Th-hex-sph.mesh") ;
```

build with emc2, bamg, modulef, etc...

Laplace equation, weak form

Let a domain Ω with a partition of $\partial\Omega$ in Γ_2, Γ_e .

Find u a solution in such that :

$$-\Delta u = 1 \text{ in } \Omega, \quad u = 2 \text{ on } \Gamma_2, \quad \frac{\partial u}{\partial \vec{n}} = 0 \text{ on } \Gamma_e \quad (1)$$

Denote $V_g = \{v \in H^1(\Omega) / v|_{\Gamma_2} = g\}$.

The Basic variationnall formulation with is : find $u \in V_2(\Omega)$, such that

$$\int_{\Omega} \nabla u \cdot \nabla v = \int_{\Omega} 1v + \int_{\Gamma} \frac{\partial u}{\partial n} v, \quad \forall v \in V_0(\Omega) \quad (2)$$

Laplace equation in FreeFem++

The finite element method is just : replace V_g with a finite element space, and the FreeFem++ code :

```
mesh Th("Th-hex-sph.msh") ;
fespace Vh(Th,P1) ; // define the P1 EF space

Vh u,v ;
macro Grad(u) [dx(u),dy(u),dz(u)] // EOM
solve laplace(u,v,solver=CG) =
  int3d(Th) ( Grad(u)'*Grad(v) ) - int3d(Th) ( 1*v)
  + on(2,u=2) ; // int on  $\gamma_2$ 
plot(u,fill=1,wait=1,value=0,wait=1) ;
```

Laplace equation 2d / figure



Execute fish.edp Execute Laplace3d.edp

The plot of the Finite Basis Function (3d plot)

```
load "Element_P3" // load P3 finite element
mesh Th=square(3,3); // a mesh with 2 elements
fespace Vh(Th,P3);

Vh vi=0;
for (int i=0;i<vi[].n;++i)
{
    vi[][i]=1; // def the  $i + 1^{th}$  basis function

    plot(vi,wait=0,cmm=" v"+i,dim=3);
    vi[]=0; // undef  $i + 1^{th}$  basis function
}
}
```

Execute plot-fb.edp

Matrix and vector

The 3d FreeFem++ code :

```
mesh3 Th("dodecaedre.mesh");
fespace Vh(Th,P13d); // define the P1 EF space

Vh u,v;

macro Grad(u) [dx(u),dy(u),dz(u)] // EOM

varf vlaplace(u,v,solver=CG) =
  int3d(Th)( Grad(u)'*Grad(v) ) + int3d(Th) ( 1*v)
  + on(2,u=2); // on  $\gamma_2$ 

matrix A= vlaplace(Vh,Vh,solver=CG); // bilinear part
real[int] b=vlaplace(0,Vh); // // linear part
u[] = A^-1*b;
```

Execute Poisson3d.edp



Remark on varf

The functions appearing in the variational form are formal and local to the `varf` definition, the only important thing in the order in the parameter list, like in

```
varf vb1([u1,u2],[q]) = int2d(Th)( (dy(u1)+dy(u2)) *q) + int2d(Th)(1*q) ;
varf vb2([v1,v2],[p]) = int2d(Th)( (dy(v1)+dy(v2)) *p) + int2d(Th)(1*p) ;
```

To build matrix A from the bilinear part of the variational form a of type `varf` do simply

```
matrix B1 = vb1(Vh,Wh [, ...] ) ;
matrix<complex> C1 = vb1(Vh,Wh [, ...] ) ;
// where
// the fespace have the correct number of component
// Vh is "fespace" for the unknown fields with 2 component
// ex fespace Vh(Th,[P2,P2]); or fespace Vh(Th,RT);
// Wh is "fespace" for the test fields with 1 component
```

To build matrix a vector, the $u1 = u2 = 0$.

```
real[int] b = vb2(0,Wh) ;
complex[int] c = vb2(0,Wh) ;
```

The boundary condition terms

- An "on" scalar form (for Dirichlet) : `on(1, u = g)`
The meaning is for all degree of freedom i of this associated boundary, the diagonal term of the matrix $a_{ii} = tgv$ with the *terrible geant value* `tgv` ($=10^{30}$ by default) and the right hand side $b[i] = "(\Pi_h g)[i]" \times tgv$, where the $"(\Pi_h g)[i]"$ is the boundary node value given by the interpolation of g .
- An "on" vectorial form (for Dirichlet) : `on(1, u1=g1, u2=g2)` If you have vectorial finite element like RT0, the 2 components are coupled, and so you have : $b[i] = "(\Pi_h(g1, g2))[i]" \times tgv$, where Π_h is the vectorial finite element interpolant.
- a linear form on Γ (for Neumann in 2d)
`-int1d(Th)(f*w)` or `-int1d(Th,3)(f*w)`
- a bilinear form on Γ or Γ_2 (for Robin in 2d)
`int1d(Th)(K*v*w)` or `int1d(Th,2)(K*v*w)`.
- a linear form on Γ (for Neumann in 3d)
`-int2d(Th)(f*w)` or `-int2d(Th,3)(f*w)`
- a bilinear form on Γ or Γ_2 (for Robin in 3d)
`int2d(Th)(K*v*w)` or `int2d(Th,2)(K*v*w)`.

a Neumann Poisson Problem with 1D Lagrange multiplier

The variational form is find $(u, \lambda) \in V_h \times \mathbb{R}$ such that

$$\forall (v, \mu) \in V_h \times \mathbb{R} \quad a(u, v) + b(u, \mu) + b(v, \lambda) = l(v), \quad \text{where } b(u, \mu) = \int \mu u dx$$

```
mesh Th=square(10,10);          fespace Vh(Th,P1);          // P1 FE space
int n = Vh.ndof, n1 = n+1;
func f=1+x-y;                   macro Grad(u) [dx(u),dy(u)]          // EOM
varf va(uh,vh) = int2d(Th)( Grad(uh)'*Grad(vh) );
varf vL(uh,vh) = int2d(Th)( f*vh ); varf vb(uh,vh)= int2d(Th)(1.*vh);
matrix A=va(Vh,Vh);
real[int] b=vL(0,Vh), B = vb(0,Vh);
real[int] bb(n1),x(n1),b1(1),l(1); b1=0;
matrix AA = [ [ A , B ] , [ B', 0 ] ]; bb = [ b, b1]; // blocks
set(AA,solver=UMFPACK); // set the type of linear solver.
x = AA^-1*bb; [uh[],l] = x; // solve the linear system
plot(uh,wait=1); // set the value
```

Execute Laplace-lagrange-mult.edp

Fast The Time depend Problem/ formulation

First, it is possible to define variational forms, and use this forms to build matrix and vector to make very fast script (4 times faster here).

For example solve the Thermal Conduction problem of section 3.4.

The variational formulation is in $L^2(0, T; H^1(\Omega))$; we shall seek u^n satisfying

$$\forall w \in V_0; \quad \int_{\Omega} \frac{u^n - u^{n-1}}{\delta t} w + \kappa \nabla u^n \nabla w + \int_{\Gamma} \alpha (u^n - u_{ue}) w = 0$$

where $V_0 = \{w \in H^1(\Omega) / w|_{\Gamma_{24}} = 0\}$.

Fast The Time depend Problem/ algo

So the to code the method with the matrices $A = (A_{ij})$, $M = (M_{ij})$, and the vectors $u^n, b^n, b', b'', b_{cl}$ (notation if w is a vector then w_i is a component of the vector).

$$u^n = A^{-1}b^n, \quad b' = b_0 + Mu^{n-1}, \quad b'' = \frac{1}{\varepsilon} b_{cl}, \quad b_i^n = \begin{cases} b''_i & \text{if } i \in \Gamma_{24} \\ b'_i & \text{else} \end{cases}$$

Where with $\frac{1}{\varepsilon} = \text{tgv} = 10^{30}$:

$$A_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{ and } j = i \\ \int_{\Omega} w_j w_i / dt + k(\nabla w_j \cdot \nabla w_i) + \int_{\Gamma_{13}} \alpha w_j w_i & \text{else} \end{cases}$$

$$M_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{ and } j = i \\ \int_{\Omega} w_j w_i / dt & \text{else} \end{cases}$$

$$b_{0,i} = \int_{\Gamma_{13}} \alpha u_{ue} w_i$$

$$b_{cl} = u^0 \quad \text{the initial data}$$

Fast The Time depend Problem/ edp

...

```
Vh u0=fu0,u=u0 ;
```

Create three variational formulation, and build the matrices A, M .

```
varf vthermic (u,v)= int2d(Th)(u*v/dt + k*(dx(u) * dx(v) + dy(u) * dy(v)))  
+ int1d(Th,1,3)(alpha*u*v) + on(2,4,u=1) ;
```

```
varf vthermic0(u,v) = int1d(Th,1,3)(alpha*ue*v) ;
```

```
varf vMass (u,v)= int2d(Th)( u*v/dt) + on(2,4,u=1) ;
```

```
real tgv = 1e30 ;
```

```
A= vthermic(Vh,Vh,tgv=tgv,solver=CG) ;
```

```
matrix M= vMass(Vh,Vh) ;
```

Fast The Time depend Problem/ edp

Now, to build the right hand side we need 4 vectors.

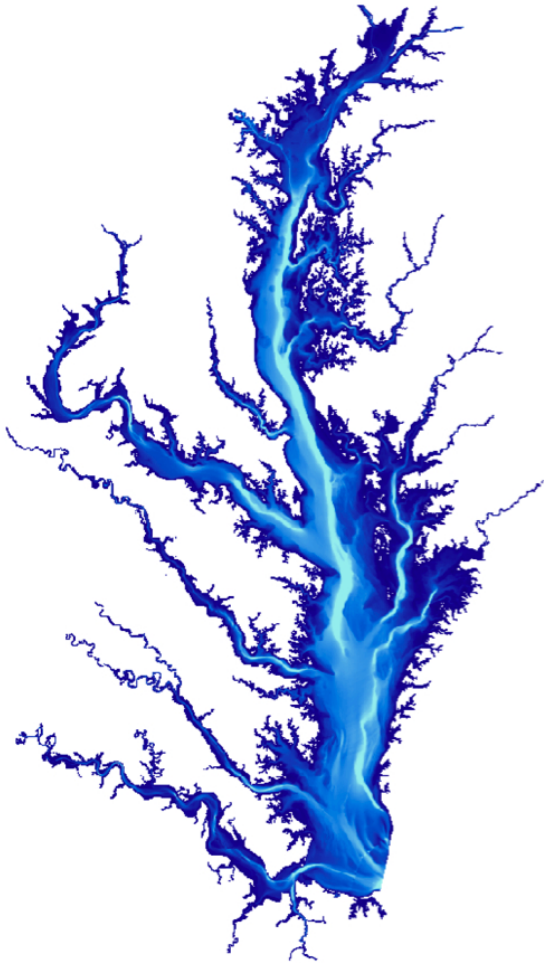
```
real[int]  b0  = vthermic0(0,Vh) ;           // constant part of the RHS
real[int]  bcn = vthermic(0,Vh) ; // tgv on Dirichlet boundary node(!=0)
           // we have for the node  $i : i \in \Gamma_{24} \Leftrightarrow bcn[i] \neq 0$ 
real[int]  bcl=tgv*u0[] ;           // the Dirichlet boundary condition part
```

The Fast algorithm :

```
for(real t=0 ;t<T ;t+=dt){
    real[int] b = b0 ;           // for the RHS
    b += M*u[] ;               // add the the time dependant part
    b = bcn? bcl : b ;         // do  $\forall i: b[i] = bcn[i]? bcl[i] : b[i]$  ;
    u[] = A^-1*b ;
    plot(u) ;
}
```

Some Idea to build meshes

The problem is to compute eigen value of a potential flow on the Chesapeake bay (Thank to Mme. Sonia Garcia, smg@usna.edu).



- Read the image in `freefem`, `adaptmesh`, `trunc` to build a first mesh of the bay and finally remove no connected component. We use : $\xi > 0.9\|\xi\|_\infty$ where ξ is solution of

$$10^{-10}\xi - \Delta\xi = 0 \quad \text{in } \Omega; \quad \frac{\partial\xi}{\partial n} = 1 \quad \text{on } \Gamma.$$

Remark, on each connect component ω of Ω , we have

$$\xi|_\omega \simeq 10^{10} \frac{\int_{\partial\omega} 1}{\int_\omega 1}.$$

Execute `Chesapeake/Chesapeake-mesh.edp`

- Solve the eigen value, on this mesh.
- Execute `Chesapeake/Chesapeake-flow.edp`

A cube

```
load "msh3" // buildlayer
int nn=10 ;
int[int] rup=[0,2], rdown=[0,1], rmid=[1,1,2,1,3,1,4,1],rtet[0,0] ;
real zmin=0,zmax=1 ;

mesh3 Th=buildlayers(square(nn,nn),nn,
                    zbound=[zmin,zmax],
                    reftet=rtet,
                    reffacemid=rmid,
                    reffaceup = rup,
                    reffacelow = rdown) ;

savemesh(Th,"c10x10x10.mesh") ;
exec("medit c10x10x10 ;rm c10x10x10.mesh") ;
```

Execute Cube.edp



3D layer mesh of a Lac

```
load "msh3"//      buildlayer
load "medit"//     buildlayer
int nn=5 ;
border cc(t=0,2*pi){x=cos(t) ;y=sin(t) ;label=1 ;}
mesh Th2= buildmesh(cc(100)) ;
fespace Vh2(Th2,P2) ;
Vh2 ux,uz,p2 ;
int[int] rup=[0,2],  rdown=[0,1], rmid=[1,1] ;
func zmin= 2-sqrt(4-(x*x+y*y)) ;   func zmax= 2-sqrt(3.) ;
//      we get nn*coef layers
mesh3 Th=buildlayers(Th2,nn,
                    coef= max((zmax-zmin)/zmax,1./nn),
                    zbound=[zmin,zmax],
                    reffacemid=rmid,  reffaceup = rup,
                    reffacelow = rdown) ;           //      label def
medit("lac",Th) ;
```

Execute Lac.edp Execute 3d-leman.edp

Stokes equation

The Stokes equation is find a velocity field $\mathbf{u} = (u_1, \dots, u_d)$ and the pressure p on domain Ω of \mathbb{R}^d , such that

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= 0 && \text{in } \Omega \\ \nabla \cdot \mathbf{u} &= 0 && \text{in } \Omega \\ \mathbf{u} &= \mathbf{u}_\Gamma && \text{on } \Gamma \end{aligned}$$

where \mathbf{u}_Γ is a given velocity on boundary Γ .

The classical variationnal formulation is : Find $\mathbf{u} \in H^1(\Omega)^d$ with $\mathbf{u}|_\Gamma = \mathbf{u}_\Gamma$, and $p \in L^2(\Omega)/\mathbb{R}$ such that

$$\forall \mathbf{v} \in H_0^1(\Omega)^d, \forall q \in L^2(\Omega)/\mathbb{R}, \quad \int_{\Omega} \nabla \mathbf{u} : \nabla \mathbf{v} - p \nabla \cdot \mathbf{v} - q \nabla \cdot \mathbf{u} = 0$$

or now find $p \in L^2(\Omega)$ such than (with $\varepsilon = 10^{-10}$)

$$\forall \mathbf{v} \in H_0^1(\Omega)^d, \forall q \in L^2(\Omega), \quad \int_{\Omega} \nabla \mathbf{u} : \nabla \mathbf{v} - p \nabla \cdot \mathbf{v} - q \nabla \cdot \mathbf{u} + \varepsilon p q = 0$$

Stokes equation in FreeFem++

```
... buidl mesh .... Th (3d) T2d ( 2d)
fespace VVh(Th, [P2,P2,P2,P1]) ; // Taylor Hood Finite element.
macro Grad(u) [dx(u),dy(u),dz(u)] // EOM
macro div(u1,u2,u3) (dx(u1)+dy(u2)+dz(u3)) // EOM
varf vStokes([u1,u2,u3,p],[v1,v2,v3,q]) = int3d(Th)(
    Grad(u1)'*Grad(v1) + Grad(u2)'*Grad(v2) + Grad(u3)'*Grad(v3)
    - div(u1,u2,u3)*q - div(v1,v2,v3)*p + 1e-10*q*p )
+ on(1,u1=0,u2=0,u3=0) + on(2,u1=1,u2=0,u3=0) ;
matrix A=vStokes(VVh,VVh) ; set(A,solver=UMFPACK) ;
real[int] b= vStokes(0,VVh) ;
VVh [u1,u2,u3,p] ; u1 []= A^-1 * b ;

// 2d intersection of plot
fespace V2d(T2d,P2) ; // 2d finite element space ..
V2d ux= u1(x,0.5,y) ; V2d uz= u3(x,0.5,y) ; V2d p2= p(x,0.5,y) ;
plot([ux,uz],p2,cmm=" cut y = 0.5") ;
```

Execute Stokes3d.edp

Stokes equation with Stabilization term / T. Chacòn

If you use Finite element P_2 in velocity and pressure, then we need a stabilisation term. The term to the classical variational formulation :

$$D = - \sum_{K \in Th} \int_K \tau_K R_h(\partial_x p) R_h(\partial_x q) + R_h(\partial_y p) (R_h \partial_y q) dx$$

To build R_h , first, denote

V_h P_1 continuous finite space

V_h^{dc} P_1 fully discontinuous finite space

I_h the trivial injection from V_h to V_h^{dc}

P_h an interpolation operator from V_h^{dc} to V_h

I_d the identity $V_h^{dc} \mapsto V_h^{dc}$.

$$R_h = Id - I_h P_h$$

Howto build R_h in FreeFem++

```
matrix Ih = interpolate(Vdch,Vh) ;
matrix Ph = interpolate(Vh,Vdch) ;
if( !scootzhang)
{
    // Clement's Operator or  $L_2$  projection with mass lumping
    varf vsigma(u,v)=int2d(Th)(v) ;
    Vh sigma ; sigma[]=vsigma(0,Vh) ; //  $\sigma_i = \int_{\Omega} w^i$ 
    varf vP2L(u,v)=int2d(Th,qft=qf1pTlump)(u*v/sigma) ; //  $P_1$  Mass Lump
    Ph=vP2L(Vdch,Vh) ;
}
matrix IPh = Ih*Ph ;
real[int] un(IPh.n) ; un=1 ;
matrix Id=un ;
Rh = Id + (-1.)*IPh ; //  $Id - Rh_h$ 
```

Howto build the D matrix in FreeFem++

```
.....
fespace Wh(Th, [P2,P2,P2]) ;                               // the Stokes FE Space
fespace Vh(Th,P1) ;                                         fespace Vdch(Th,P1dc) ;
.....
matrix D ;                                                  // the variable to store the matrix D
{ varf vMtk(p,q)=int2d(Th)(hTriangle*hTriangle*ctk*p*q) ;
  matrix Mtk=vMtk(Vdch,Vdch) ;
  int[int] c2=[2] ;                                         // take the 2 second component of Wh.
  matrix Dx = interpolate(Vdch,Wh,U2Vc=c2,op=1) ;           //  $\partial_x p$  discrete operator
  matrix Dy = interpolate(Vdch,Wh,U2Vc=c2,op=2) ;           //  $\partial_y p$  discrete operator
  matrix Rh ;
  ... add Build of Rh code here
  Dx = Rh*Dx ;      Dy = Rh*Dy ;

                                                                    // Sorry matrix operation is done one by one.
  matrix DDxx= Mtk*Dx ; DDxx = Dx'*DDxx ;
  matrix DDyy= Mtk*Dy ; DDyy = Dy'*DDyy ;
  D = DDxx + DDyy ;
}                                                            // cleaning all local matrix and array.
A = A + D ;                                                // add to the Stokes matrix
.....
```

Execute Stokes-tomas.edp

Coupling Finite element / BEM with FreeFem

Let Γ_b the straight border of length p // to x axis, with normal equal to $\vec{n} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. We need to **add border term** in the variational formulation

$$\int_{\Omega} \nabla u \cdot \nabla \bar{v} - \int_{\Gamma_b} \bar{v} \frac{\partial u}{\partial \vec{n}} = \int_{\Omega} f \bar{v}, \quad \forall v.. \quad (3)$$

We decompose w^i in the orthogonal de Floquet basic on border $\Gamma_b = \{x \in [0, p[, y = 0\}$

$$f_n = \exp(-2\pi(inx + |n|y)/p), \quad \int_{\Gamma_b} f_n \bar{f}_m = p \delta_{mn}$$

. Remark $-\Delta f_n = 0$, and we have

$$w_i = \sum_n c_n^i f_n \quad \text{and by orthogonality} \quad c_m^i = 1/p \int_{\Gamma_b} w_i \bar{f}_m$$

and $\frac{\partial f_n}{\partial \vec{n}} = -g_n f_n$ with $g_n = 2\pi|n|/p$ So we have :

$$- \int_{\Gamma_b} \bar{w}^i dn (w^j) ds = p \sum_n g_n \bar{c}_n^i c_n^j$$

Execute BEM.edp

incompressible Navier-Stokes equation with characteristics methods

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = 0, \quad \nabla \cdot u = 0$$

with the same boundary conditions and with initial conditions $u = 0$.

This is implemented by using the interpolation operator for the term $\frac{\partial u}{\partial t} + u \cdot \nabla u$, giving a discretization in time

$$\begin{aligned} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} &= 0, \\ \nabla \cdot u^{n+1} &= 0 \end{aligned} \tag{4}$$

The term $X^n(x) \approx x - u^n(x)\tau$ will be computed by the interpolation operator, or with convect operator (work form version 3.3)

The ff++ NSI 3d code

```
real alpha =1./dt ;
varf vNS( [uu1,uu2,uu3,p] , [v1,v2,v3,q] ) =
  int3d(Th)( alpha*(uu1*v1+uu2*v2+uu3*v3)
+ nu*(Grad(uu1)'*Grad(v1) + Grad(uu2)'*Grad(v2) + Grad(uu3)'*Grad(v3))
- div(uu1,uu2,uu3)*q - div(v1,v2,v3)*p + 1e-10*q*p )
+ on(1,2,3,4,5,uu1=0,uu2=0,uu3=0)
+ on(6,uu1=4*(1-x)*(x)*(y)*(1-y),uu2=0,uu3=0)
+ int3d(Th)( alpha*(
  u1(X1,X2,X3)*v1 + u2(X1,X2,X3)*v2 + u3(X1,X2,X3)*v3 )) ;
```

or with convect tools change the last line by

```
+ int3d(Th,optimize=1)( alpha*convect( [u1,u2,u3] , -dt,u1)*v1
+alpha*convect( [u1,u2,u3] , -dt,u2)*v2
+alpha*convect( [u1,u2,u3] , -dt,u2)*v3) ;
```

The ff++ NSI 3d code/ the loop in times

```
A = vNS(VVh,VVh) ;    set(A,solver=UMFPACK) ; //    build and factorize matrix
real t=0 ;
for(int i=0 ;i<50 ;++i)
  { t += dt ;  X1 []=XYZ []-u1 []*dt ;           //    set  $\chi=[X1,X2,X3]$  vector
    b=vNS(0,VVh) ;                               //    build NS rhs
    u1 []= A^-1 * b ;                             //    solve the linear systeme
    ux= u1(x,0.5,y) ;  uz= u3(x,0.5,y) ;  p2= p(x,0.5,y) ;
    plot([ux,uz],p2,cmm=" cut y = 0.5, time =" +t,wait=0) ;  }
```

Execute NSI3d.edp

Mortar Method

Let Ω be a partition without overlap $\Omega = \cup_{i=0,\dots,N} \Omega_i$.

Where Ω is the open set without the skeleton \mathcal{S} and the external boundary is Γ . So the Mortar problem is : Find $u \in H^1(\Omega)$ such that $u|_{\Gamma} = g$ and $\lambda \in H^{-\frac{1}{2}}(\mathcal{S})$ and

$$\forall v \in H^1(\Omega), \quad v|_{\Gamma} = 0, \quad \int_{\Omega} \nabla u \nabla v + \int_{\mathcal{S}} [v] \lambda = \int_{\Omega_i} f v$$

$$\forall \mu \in H^{-\frac{1}{2}}(\mathcal{S}), \quad \int_{\mathcal{S}} [u] \mu = 0$$

For each sub domain Ω_i ,

$$\forall v \in H^1(\Omega_i), \quad v|_{\Gamma} = 0, \quad \int_{\Omega_i} \nabla u \nabla v + \int_{\mathcal{S} \cap \partial \Omega_i} \varepsilon_i \lambda v = \int_{\Omega_i} f v$$

$$\forall \mu \in H^{-\frac{1}{2}}(\mathcal{S}), \quad \sum_i \int_{\mathcal{S} \cap \partial \Omega_i} \varepsilon_i \mu u = 0$$

Where $\varepsilon_i = \mathbf{n}_{\mathcal{S}} \cdot \mathbf{n}_i$, $\varepsilon_i = \pm 1$ and $\sum_i \varepsilon_i = 0$.

Mortar Method Precond

$$J'(\lambda)(\mu) = - \int_{\mathcal{S}} [u_\lambda] \mu = 0 \quad \forall \mu$$

where u_λ is solution of

$$\forall v \in H^1(\Omega_i), \quad v|_\Gamma = 0, \quad \int_{\Omega_i} \nabla u_\lambda \nabla v + \int_{\mathcal{S} \cap \partial\Omega_i} \varepsilon_i \lambda v = \int_{\Omega_i} f v$$

For each sub domain Ω_i ,

$$\forall v \in H^1(\Omega_i), \quad v|_\Gamma = 0, \quad \int_{\Omega_i} \nabla u \nabla v + \int_{\mathcal{S} \cap \partial\Omega_i} \varepsilon_i \lambda v = \int_{\Omega_i} f v$$

$$\forall \mu \in H^{-\frac{1}{2}}(\mathcal{S}), \quad \sum_i \int_{\mathcal{S} \cap \partial\Omega_i} \varepsilon_i \mu u = 0$$

Where $\varepsilon_i = \mathbf{n}_\mathcal{S} \cdot \mathbf{n}_i$, $\varepsilon_i = \pm 1$ and $\sum_i \varepsilon_i = 0$.

Mortar method, compute ε_i

```
func f=1+x+y;          real g=1;
...
fespace Lh(Thm,P1);    Lh  lh=0,rhsl=0;
fespace RTh(Tha,[P0edge,P0edge]); // Finite element constant on each edge
varf vNN([ux,uy],[nx,ny]) = int1d(Tha,1)(( nx*N.x + ny*N.y)/lenEdge);
Nmx[] = vNN(0,RTh); // Def of nS

// in a macro where i is the ssd number
macro defeps(i)
fespace Eh#i(Th#i,P0edge);
varf veps#i(u,v) = int1d(Th#i,1,qforder=5)(( Nmx*N.x + Nmy*N.y)*v/lenEdge);
Eh#i eps#i = 0;
eps#i[] = veps#i(0,Eh#i);
eps#i = -real(eps#i <-0.01) + real(eps#i >0.01); // a ±1
```

Mortar method

```
macro defspace(i)
  cout << " Domaine " << i<< " -----" << endl ;
  fespace Vh#i(Th#i,P1) ;
  Vh#i u#i ;                               Vh#i rhs#i ;
  defeps(i)
  varf vLapM#i([u#i],[v#i]) =
    int2d(Th#i)( Grad(u#i)'*Grad(v#i) )    + int2d(Th#i) (f*v#i)
  + on(labext,u#i=g) ;
  varf cc#i([l],[u]) = int1d( Thmm
  matrix C#i = cc#i(Lh,Vh#i) ;
  matrix A#i = vLapM#i(Vh#i,Vh#i,solver=GMRES) ;
  rhs#i []=vLapM#i(0,Vh#i) ;                // End of macro defspace(i)
defspace(0) defspace(1) defspace(2) defspace(3)
```

Mortar method/ Direct Method

```
varf vDD(u,v) = int2d(Thm)(u*v*1e-10) ;
matrix DD=vDD(Lh,Lh) ; // a trick to make a invertible matrix.
matrix A=[ [ A0 ,0 ,0 ,0 ,C0 ],
           [ 0 ,A1 ,0 ,0 ,C1 ],
           [ 0 ,0 ,A2 ,0 , C2 ],
           [ 0 ,0 ,0 ,A3, C3 ],
           [ C0',C1',C2',C3',DD ] ] ;
real[int] xx(M.n), bb=[rhs0[], rhs1[],rhs2[],rhs3[],rhs1[] ] ;
set(A,solver=UMFPACK) ; // choose the solver associated to M
xx = A^-1 * bb ;
[u0[],u1[],u2[],u3[],lh[]] = xx ; // dispatcher
```

Execute DDM18-Mortar.edp

Mortar method/ GC method

```
func real[int] SkPb(real[int] &l)
{  int verb=verbosity;  verbosity=0;  itera++;
   v0[] = rhs0[]; v0[]+= C0* l; u0[] = A0^-1*v0[];
   v1[] = rhs1[]; v1[]+= C1* l; u1[] = A1^-1*v1[];
   v2[] = rhs2[]; v2[]+= C2* l; u2[] = A2^-1*v2[];
   v3[] = rhs3[]; v3[]+= C3* l; u3[] = A3^-1*v3[];
   l = C1'*u1[];  l += C0'*u0[];  l += C2'*u2[];
   l += C3'*u3[];  l= lbc? lbc0: l;
   verbosity=verb;
   return l; };
verbosity=100; lh[]=0;
LinearCG(SkPb,lh[],eps=1.e-2,nbiter=30);
```

Mortar method/ in parallel : The functional

```
func real[int] SkPb(real[int] &l)
{
    int verb=verbosity;  verbosity=0;  itera++;
    broadcast(processor(0),what);
    if(what==2) return l;
    broadcast(processor(0),l);
    v0[] = rhs0[]; v0[]+= C0* l; u0[] = A0^-1*v0[];
    l = C0'*u0[];
    l= lbc? lbc0: l; // put zero on boundary
    if(mpirank==0) // on master process
        for (int i=1;i<4;++i)
            { processor(i) >> lw[];
              l += lw[]; }
    else processor(0) << l; // on slave process
    verbosity=verb; // restore level of output.
    return l; }
```

Mortar method/ CG in parallel

```
what=1 ;
verbosity=100 ;
if(mpirank==0)
{
    LinearCG(SkPb,1h[],eps=1.e-3,nbiter=20) ;           // on master process
    what=2 ; SkPb(1h[]) ;                               // future withprocess=[0,1,2,3]
}
else // on slave process
    while(what==1)
        SkPb(1h[]) ;
```

Execute DDM18-mortar-mpi.edp

Dynamics Load facility

Or How to add your C++ function in FreeFem++.

First, like in cooking, the first true difficulty is how to use the kitchen.

I suppose you can compile the first example for the `examples++-load`

```
numermac11:FH-Seville hecht# ff-c++ myppm2rnm.cpp
export MACOSX_DEPLOYMENT_TARGET=10.3
g++ -c -DNDEBUG -O3 -O3 -march=pentium4 -DDRAWING -DBAMG_LONG_LONG -DNCHECKPTR
-I/usr/X11/include -I/usr/local/lib/ff++/3.4/include 'myppm2rnm.cpp'
g++ -bundle -undefined dynamic_lookup -DNDEBUG -O3 -O3 -march=pentium4 -DDRAWING
-DBAMG_LONG_LONG -DNCHECKPTR -I/usr/X11/include 'myppm2rnm.o' -o myppm2rnm.dylib
```

add tools to read `pgm` image

The interesting code

```
#include "ff++.hpp"
typedef KNM<double> * pRnm;           // the freefem++ real[int,int] array variable type
typedef KN<double> * pRn;            // the freefem++ real[int] array variable type
typedef string ** string;           // the freefem++ string variable type

pRnm read_image( pRnm const & a,const pstring & b); // the function to read image
pRn seta( pRn const & a,const pRnm & b) // the function to set 2d array from 1d array
{ *a=*b;
  KN_<double> aa=*a;
  return a;}

class Init { public: Init(); }; // C++ trick to call a method at load time
Init init; // a global variable to enforce the initialisation by c++
Init::Init(){ // the like with FreeFem++ s
  // add ff++ operator "<-" constructor of real[int,int] form a string
  TheOperators->Add("<-",
    new OneOperator2_<KNM<double> *,KNM<double> *,string*>(&read_image));
  // add ff++ an affection "=" of real[int] form a real[int,int]
  TheOperators->Add("=",
    new OneOperator2_<KN<double> *,KN<double> *,KNM<double>* >(seta));
}
```

Remark, **TheOperators** is the ff++ variable to store all world operator, **Global** is to store function.

The prototype

```
OneOperator2_<returntype ,typearg1 ,typearg2>(& thefunction ));  
returntype thefunction(typearg1 const &, typearg2 const &)
```

To get the C++ type of all freefem++ type, method, operator : just do in examples++-tutorial directory

```
c++filt -t < lestable
```

```
Cmatrix 293 Matrice_Creuse<std::complex<double> >  
R3 293 Fem2D::R3  
bool 293 bool*  
complex 293 std::complex<double>*  
element 293 (anonymous namespace)::lgElement  
func 294 C_F0  
  ifstream 293 std::basic_istream<char, std::char_traits<char> >**  
int 293 long*  
matrix 293 Matrice_Creuse<double>  
mesh 293 Fem2D::Mesh**  
mesh3 293 Fem2D::Mesh3**  
ofstream 293 std::basic_ostream<char, std::char_traits<char> >**  
problem 294 Problem  
real 293 double*  
solve 294 Solve  
string 293 std::basic_string<char, std::char_traits<char>, std::allocator<char> >**  
varf 294 C_args  
vertex 293 (anonymous namespace)::lgVertex
```

FreeFem++ Triangle/Tet capability

```
Element::nv ;
const Element::Vertex & V = T[i] ;
double a = T.mesure() ;
Rd AB = T.Edge(2) ;
Rd hC = T.H(2) ; // gradient de la fonction de base associé au sommet 2
R l = T.lenEdge(i) ; // longueur de l'arête opposée au sommet i
(Label) T ; // la référence du triangle T
R2 G(T(R2(1./3,1./3))) ; // le barycentre de T in 3d

// soit T un Element de sommets A,B,C ∈ ℝ²
// -----
// nombre de sommets d'un triangle (ici 3)
// le sommet i de T (i ∈ 0,1,2)
// mesure de T
// "vecteur arête" de l'arete
```

FreeFem++ Mesh/Mesh3 capability

```
Mesh Th("filename"); // lit le maillage Th du fichier "filename"
Th.nt; // nombre de element (triangle or tet)
Th.nv; // nombre de sommets
Th.neb or Th.nbe; // nombre de éléments de bord (2d) or(3d)
Th.area; // aire du domaine de calcul
Th.peri; // perimetre du domaine de calcul
typedef Mesh::Rd Rd; // R2 or R3
Mesh2::Element & K = Th[i]; // triangle i , int i ∈ [0,nt[
Rd A=K[0]; // coordonnée du sommet 0 sur triangle K
Rd G=K(R2(1./3,1./3)); // le barycentre de K.
Rd DLambda[3];
K.Gradlambda(DLambda); // calcul des trois  $\nabla \lambda_i^K$  pour  $i = 0, 1, 2$ 
Mesh::Vertex & V = Th(j); // sommet j , int j ∈ [0,nv[
Mesh::BorderElement & BE=th.be(1); // Element du bord, int l ∈ [0,nbe[
Rd B=BE[1]; // coordonnée du sommet 1 sur Seg BE
Rd M=BE(0.5); // le milieu de BE.
int j = Th(i,k); // numéro global du sommet k ∈ [0,3[ du triangle i ∈ [0,nt[
Mesh::Vertex & W=Th[i][k]; // référence du sommet k ∈ [0,3[ du triangle i ∈ [0,nt[

int ii = Th(K); // numéro du triangle K
int jj = Th(V); // numéro du sommet V
int ll = Th(BE); // numéro de Seg de bord BE
assert( i == ii && j == jj); // vérification
```

Some Example (from the archive)

- Execute BlackScholes2D.edp
- Execute Poisson-mesh-adap.edp
- Execute Micro-wave.edp
- Execute wafer-heating-laser-axi.edp
- Execute nl-elast-neo-Hookean.edp
- Execute Stokes-eigen.edp
- Execute fluid-Struct-with-Adapt.edp
- Execute optim-control.edp
- Execute VI-2-membrane-adap.edp

An exercise in FreeFem++

The geometrical problem : Find a function $u : C^1(\Omega) \mapsto \mathbb{R}$ where u is given on $\Gamma = \partial\Omega$, (e.i. $u|_{\Gamma} = g$) such that the area of the surface S parametrize by $(x, y) \in \Omega \mapsto (x, y, u(x, y))$ is minimal.

So the problem is **arg min** $J(u)$ where

$$J(u) = \iint_{\Omega} \left\| \begin{pmatrix} 1 \\ 0 \\ \partial_x u \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ \partial_y u \end{pmatrix} \right\|_2 dx dy = \iint_{\Omega} \sqrt{1 + \nabla u \cdot \nabla u} dx dy$$

So the Euler equation associated to the minimisation is :

$$\forall v/v|_{\Gamma} = 0 \quad : \quad DJ(u)(v) = \iint_{\Omega} \frac{\nabla u \cdot \nabla v}{\sqrt{1 + \nabla u \cdot \nabla u}} = 0$$

So find the solution for $\Omega =]0, \pi[\times]0, \pi[$ and $g(x, y) = \cos(2 * x) * \cos(2 * y)$. by using the Non Linear Conjugate gradient NLGG like in the example : `algo.edp` in `examples++-tutorial`.

Tools

Example of use of NLCG function :

```
Vh u; // the finite to store the current value
func real J(real[int] & xx) // the fonctionnal to minimized
{ real s=0;
  u[]=xx; // add code to copy xx array of finite element function
  ... // /
  return s; }
```

Use the varf tools the vector $DJ(u)(w^i)_i$.

```
func real[int] DJ(real[int] &xx) // the grad of fonctionnal
{ u[]=xx; // add code to copy xx array of finite element function
  .... // /
  return xx; }; // return of an existing variable ok
...
u=g;
NLCG(DJ,u[],eps=1.e-6,nbiter=20);
```

To see the 3D plot of the surface

```
plot(u,dim=3,wait=1);
```


The C++ kernel / Dehli, (1992)

My early step in C++

```
typedef double R;
class Cvirt { public: virtual R operator()(R ) const =0;};

class Cfunc : public Cvirt { public:
  R (*f)(R) ; // a function C
  R operator()(R x) const { return (*f)(x) ;}
  Cfunc( R (*ff)(R)) : f(ff) {} };

class Coper : public Cvirt { public:
  const Cvirt *g, *d; // the 2 functions
  R (*op)(R,R) ; // l'opération
  R operator()(R x) const { return (*op)((*g)(x),(*d)(x)) ;}
  Coper( R (*opp)(R,R), const Cvirt *gg, const Cvirt *dd)
    : op(opp),g(gg),d(dd) {}
  ~Coper(){delete g,delete d;} };

static R Add(R a,R b) {return a+b;} static R Sub(R a,R b) {return a-b;}
static R Mul(R a,R b) {return a*b;} static R Div(R a,R b) {return a/b;}
static R Pow(R a,R b) {return pow(a,b) ;}
```

How to code differential operator

A differential expression in a PDE problem is a sum of product

$$f * [u_i, \partial_x u_i, \partial_y u_i, \dots] * [v_j, \partial_x v_j, \partial_y v_j, \dots]$$

where the unknown part is $[u_i, \partial_x u_i, \partial_y u_i, \dots] == [(0, i), (1, i), (2, i), \dots]$ is a pair of $i' \times i$

and same of the test part.

So the differential expression is a formal sum of :

$$\sum_k f_k \times (i'_k, i_k, j'_k, j_k)$$

So we can easily code this syntaxe :

```
varf a(u,v) = int2d(Th)(Grad(u)'*Grad(v)) - int2d(Th)(f*v) + on(1,u=0) ;  
matrix A=a(Vh,Vh,solver=UMFPACK) ;  
real[int] b=a(0,Vh) ;  
u[]=A^-1*b ;
```

Au boulot !



My solution First the fonctionnal

```
func g=cos(2*x)*cos(2*y) ; // valeur au bord
mesh Th=square(20,20,[x*pi,y*pi]) ; // mesh definition of  $\Omega$ 
fespace Vh(Th,P1) ;

func real J(real[int] & xx) // the fonctionnal to minimise
{ Vh u ; u[]=xx ; // to set finite element function u from xx array
  return int2d(Th)( sqrt(1 +dx(u)*dx(u) + dy(u)*dy(u) ) ) ; }

func real[int] dJ(real[int] & xx) // the grad of the J
{ Vh u ; u[]=xx ; // to set finite element function u from xx array
  varf vDJ(uh,vh) = int2d(Th)( ( dx(u)*dx(vh) + dy(u)*dy(vh) )
    / sqrt(1. +dx(u)*dx(u) + dy(u)*dy(u) ) )
    + on(1,2,3,4,u=0) ;
  return xx= vDJ(0,Vh) ; } // warning no return of local array
```

My solution

Second the call

```
Vh u=G ;  
verbosity=5 ; // to see the residual  
int conv=NLCG(dJ,u[],nbiter=500,eps=1e-5) ;  
cout << " the surface =" << J(u[]) << endl ;  
// so see the surface un 3D  
plot(u,dim=3,wait=1) ;
```

Execute minimal-surf.edp

An exercise in FreeFem++, next

The algorithm is too slow, To speed up the convergence after some few iterations of NLCG, you can use a Newton method and make also mesh adaptation.

The Newton algorithm to solve $F(u) = 0$ is to make the loop :

$$u^{n+1} = u^n - w^n; \text{ with } DF(u^n)(w^n) = F(u^n)$$

We have ;

$$D^2J(u)(v, w) = \int \frac{\nabla v \cdot \nabla w}{\sqrt{1 + \nabla u \cdot \nabla u}} - \int \frac{(\nabla u \cdot \nabla w)(\nabla u \cdot \nabla v)}{\sqrt{1 + \nabla u \cdot \nabla u}^3}$$

So $w^n \in H_0^1$ is solution of

$$\forall v \in H_0^1 : D^2J(u)(v, w^n) = DJ(u)(v)$$

Execute `minimal-surf-newton.edp`

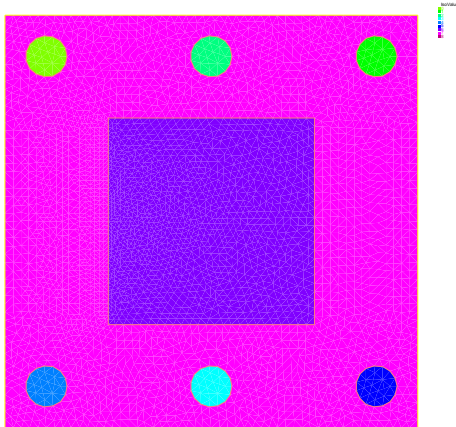
An exercice : Oven problem

Find the power on the 6 resistors of an oven such that the temperature is close as possible to a given temperature in the region 6.

The equation are the stationary Head equation in 2d with classical Fourier boundary condition. the mesh of the domain :

let call the u_p the solution of

$$\begin{aligned} -\nabla \cdot K \nabla u_p &= \sum_{i=0}^5 p_i * \chi_i \quad \text{in } \Omega \\ u + K \nabla u_p \cdot n &= 0 \quad \text{on } \Gamma = \partial\Omega \end{aligned}$$



"oven.msh"

where χ_i is the characteristics function of the resistance i , $K = 10$ in region 6, $K = 1$ over where.

The problem is find the array p such that

$$p = \operatorname{argmin} \int_{\Omega_6} (u_p - 100)^2 dx$$

Some remark

```
Xh[int] ur(6); // to store the 6 finite element functions Xh
```

To day, FreeFem++ as only linear solver on sparse matrix. so a way to solve a full matrix problem is for example :

```
real[int,int] AP(6,6); // a full matrix  
real[int] B(6),PR(6); // to array (vector of size 6)
```

```
... bla bla to compute AP and B
```

```
matrix A=AP; // build sparse data structure to store the full matrix  
set(A,solver=CG); // set linear solver to the Conjuguate Gradient  
PR=A^-1*B; // solve the linear system.
```

The file name of the mesh is oven.msh, and the region numbers are 0 to 5 for the resitor, 6 for Ω_6 and 7 for the rest of Ω and the label of Γ is 1.

My solution, build the 6 basics function u_{e_i}

```
int nbresitor=6;          mesh Th("oven.msh");
real[int] pr(nbresitor+2), K(nbresitor+2);
  K=1;      K[regi]=10;          //      def K
int regi=nbresitor, rege=nbresitor+1, lext=1;

macro Grad(u) [dx(u),dy(u)]          //      EOM
fespace Xh(Th,P2);          Xh u,v;
int iter=0;
problem Chaleur(u,v,init=iter)
  = int2d(Th)( Grad(u)'*Grad(v)* K[region]) + int1d(Th,lext)(u*v)
  + int2d(Th)(pr[region]*v);

Xh[int] ur(nbresitor);          //      to store the 6  $u_{e_i}$ 
for(iter=0;iter<nbresitor;++iter)
{ pr=0;pr[iter]=1;
  Chaleur;
  ur[iter][]=u[];
  plot(ur[iter],fill=1,wait=1); }
}
```

Computation of the optimal value

```
real[int,int] AP(nbresitor,nbresitor) ;
real[int] B(nbresitor),PR(nbresitor) ;

Xh   ui = 100 ;
for(int i=0 ;i<nbresitor ;++i)
{
    B[i]=int2d(Th,regi)(ur[i]*ui) ;
    for(int j=0 ;j<6 ;++j)
        AP(i,j)= int2d(Th,regi)(ur[i]*ur[j]) ;
}

matrix A=AP ; set(A,solver=UMFPACK) ;
PR=A^-1*B ;
cout << " P R = " << PR << endl ;
u[]=0 ;
for (int i=0 ;i<nbresitor ;++i)
    u[] += PR[i]*ur[i] [] ;
```

Execute oven-cimpa.edp

Conclusion and Future

It is a useful tool to teaches Finite Element Method, and to test some nontrivial algorithm.

- Optimization FreeFem++ in 3d
- All graphic with OpenGL (in construction)
- Galerkin discontinue (fait in 2d, à faire in 3d)
- complex problem (fait)
- 3D (under construction)
- automatic differentiation (under construction)
- // linear solver and build matrix //
- 3d mesh adaptation
- Suite et FIN. (L'avenir ne manque pas de future et lycée de Versailles)

Thank, for your attention ?