FreeFem++, a tool to solve PDE's numerically

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++`
- Poisson Problem,Variational formulation
- Mesh 2d generation and adaptation
- Mesh generation 3d
- an academic example (minimal surface problem)
- Time dependent problem
- some Trick
- Mathematical formulation of Poisson with Neumann condition
- Schwarz, BEM methods
- Dynamic Link example (hard)
- Internal data structure
- 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 parallel with MPI.

History

1987 MacFem/PCFem les ancêtres (O. Pironneau en Pascal) payant.

1992 FreeFem réécriture de C++ (P1,P0 un maillage) O. Pironneau, D. Bernardi, F. Hecht , C. Prudhomme (adaptation Maillage, bamg).

1996 FreeFem+ réécriture de C++ (P1,P0 plusieurs maillages) O. Pironneau, D. Bernardi, F. Hecht (algèbre de fonction).

1998 FreeFem++ réécriture avec autre noyau élément fini, et un autre langage utilisateur ; F. Hecht, O. Pironneau, K.Ohtsuka.

1999 FreeFem 3d (S. Del Pino) , Une première version de freefem en 3d avec des méthodes de domaine fictif.

2008 FreeFem++ v3 réécriture du noyau élément fini pour prendre en compte les cas multidimensionnels : 1d,2d,3d...

For who, for what !

For what

1. R&D
2. Academic Research ,
3. Teaching of FEM, PDE, Weak form and variational form
4. Algorithmes prototyping
5. Numerical experimentation
6. Scientific computing and Parallel computing

For who : the researcher, engineer, professor, student...

The team to day and the users

1. Dev. team (ff2a3) : F. Hecht (Kernel) , A. le Hyaric (Graphic), J. Morice (PostDoc 2years, 3d Mesh, Solver ..), G. A. Atenekeng (PostDoc 1 an, Solver //), S. Auliac (Phd 3 years, automatic Diff.), O. Pironneau (Documentation), ...
2. Tester : LJLL : 4, IRMAR : 3, CMAP : 3, INRIA : 3, Seville : 3 Know users more than 300 in the world , in more than 20 country
3. Industry : EADS, Dassault, CEA, Andras, IFREMER, PME...
4. Teaching : Ecole Polytechnique, les Mines, Central, les Pont et Chaussées...
5. University : Orsay, UMPC, Rennes, Metz, Grenoble, Seville, Jyväskylä Finlande, Kénitra Maroc, Annaba, Setif Algérie, ENIT Tunisie, Belgique, USA, Japan, Chine , Chili,
6. FreeFem++ day at IHP (Paris) : 80 participants, 12 nationality, ... (for missing item, person, etc...)

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, with matrix construction if need.
- 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 in 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-Voronoï 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, HIPS, HYPRE, SUPERLU_DIST, PASTIX. ... sparse linear solver ; eigenvalue and eigenvector computation with ARPACK.
- Online graphics with OpenGL/GLUT, C++ like syntax.
- Link with other soft : modulef, emc2, medit, gnuplot, tetgen, superlu, mumps, ...
- Dynamic linking to add plugin.
- Wide range of of examples : Navier-Stokes 3d, elasticity 3d, fluid structure, eigenvalue problem, Schwarz' domain decomposition algorithm, residual error indicator, ...

How to use

on Unix build a "yours.edp" file with your favorite editor : emacs, vi, nedit, etc. Enter `FreeFem++ yours.edp` or `FreeFem++-nw yours.edp` to execute your script. Remark, this application `FreeFem++` must be in a directory of your `PATH` shell variable.

on Window, MacOs X build a "yours.edp" file with your favorite text editor (raw text, not word text) : emacs, winedit, wordpad, bbedit, ... and click on the icon of the application `FreeFem++` and load you file via de open file dialog box or drag and drop the icon of your built file on the application `FreeFem++` icon.

Element of syntaxe 1/3

```
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 2/3

```
fespace V3h(Th,[P2,P2,P1]);
V3h [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,u2)*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);
u2[] =A^-1*b;                          //    or you can put also u1[]= or p[].
```

Element of syntaxe 3/3

```
 func Heaveside=(x>0);                                          //    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.
int[int] I(15),J(15);                                          //    two array for renumbering
                                                                                        //
                                  //   the aim is to transform a matrix into a sparse matrix
matrix B;
B = A;                                                                    //    copie matrix A
B=A(I,J);                                                          //    B(i,j) = A(I(i),J(j))
B=A(I^-1,J^-1);                                                    //    B(I(i),J(j))= A(i,j)
B.resize(10,20);              //    resize the sparse matrix  and remove out of bound terms
int[int] I(1),J(1); real[int] C(1);
[I,J,C]=A;             //    get of the sparse term of the matrix A (the array are resized)
A=[I,J,C];                                                              //    set a new matrix
matrix D=[diagofA] ;                  //    set a diagonal matrix D from the array diagofA.
matlab, scilab   : operator to set array. like:
real[int] a(2:12);                        //    set an arry of 11 values a[i]=i+2; i=0 to 10.
```

Element of syntaxe : Like in C

The key words are reserved

The operator like in C exempt: ^ & |
+ - * / ^ // where a^b= a^b
== != < > <= >= & | // where a|b= a or b, a&b= a and b
= += -= /= *=

BOOLEAN: 0 <=> false , $\neq$ 0 <=> true $=$ 1

```
//   Automatic cast for numerical value : bool, int, reel, complex , so
func heavyside = real(x>0.);
```

```
for (int i=0;i<n;i++) { ...;}
if ( <bool exp> ) { ...;} else {  ...;};
while ( <bool exp> ) { ...;}
break  continue  key words
```

weakless: all local variables are almost static (????)
bug if break before variable declaration in same block.
bug for fespace argument or fespace function argument

The C++ kernel / Dehli, (1992)

My early step in C++

```
typedef double R;
 class Cvirt { public: virtual R operator()(R ) const =0;};
class Cfonc : public Cvirt { public:
  R (*f)(R);                                                    //    a function C
  R operator()(R x) const { return (*f)(x);}
  Cfonc( 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 on in a PDE problem is like

$$f * [u_i|\partial_x u_i|\partial_y u_i|\ldots] * [v_j, \partial_x v_j|\partial_y v_i|\ldots]$$

where $[f, |g, \ldots]$ mean f or g, or ..., and where the unknown part is $[u_i|\partial_x u_i|\partial_x u_i|\ldots] \equiv [(0,i)|(1,i)|(2,i)|\ldots]$ is a pair of $i' \times i$, if we do the same of the test part, the differential expression is a formally 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;
```

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 :

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

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

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

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

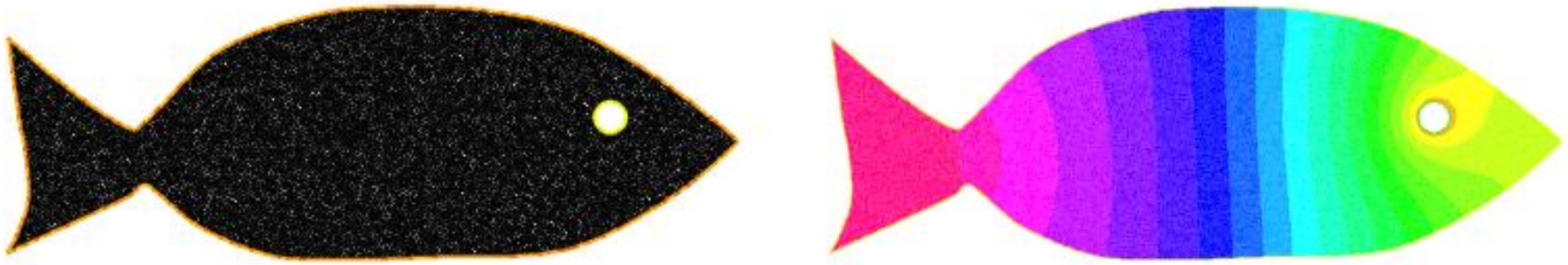
Laplace equation in FreeFem++

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

```
mesh3 Th("Th-hex-sph.msh");                                  //    read a mesh 3d
fespace Vh(Th,P1);                                       //    define the P1 EF space

 Vh u,v;                              //    set test and unkwow FE function in Vh.
macro Grad(u) [dx(u),dy(u),dz(u)]          //   End of Macro Grad definition
solve laplace(u,v,solver=CG) =
   int3d(Th)(  Grad(u)'*Grad(v)   )   -  int3d(Th) ( 1*v)
  + on(2,u=2);                                                  //    int on γ2
plot(u,fill=1,wait=1,value=0,wait=1);
```

Laplace equation 2d / figure



Execute fish.edp Execute Laplace3d.edp Execute EqPoisson.edp

Build Mesh 2d

First a 10×10 grid mesh of unit square $]0,1[^2$

```
int[int] labs=[10,20,30,40];          //   resp. bottom, right, top, left boundary labels
mesh Th1 =  square(10,10,label=labs,region=0);                    //    boundary label:
plot(Th1,wait=1);
int[int] old2newlabs=[10,11,    30,31];   //    the boundary labels 10 in 11 and 30 in 31
Th1=change(Th1,label=old2newlabs [,region= old2newregion]); //   do Change in 2d or in 3d.
```

second a L shape domain $]0,1[^2 \backslash [\frac{1}{2},1[^2$

```
border a(t=0,1.0){x=t;   y=0;  label=1;};
border b(t=0,0.5){x=1;   y=t;  label=2;};
border c(t=0,0.5){x=1-t; y=0.5;label=3;};
border d(t=0.5,1){x=0.5; y=t;  label=4;};
border e(t=0.5,1){x=1-t; y=1;  label=5;};
border f(t=0.0,1){x=0;   y=1-t;label=6;};
plot(a(6) + b(4) + c(4) +d(4) + e(4) + f(6),wait=1);          //    to see the 6 borders
mesh Th2 =  buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
```

Get a extern mesh

```
mesh Th2("april-fish.msh");
```

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

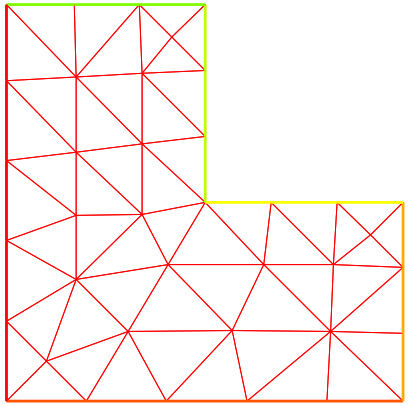
a corner singularity

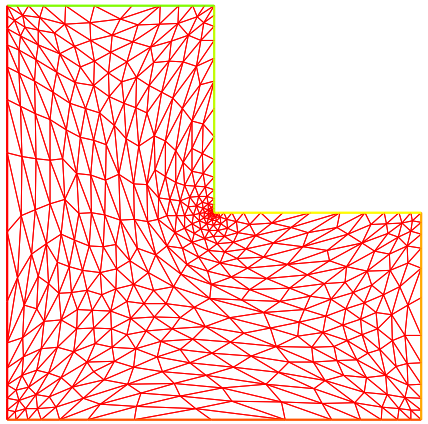
adaptation with metric

The domain is an L-shaped polygon $\Omega =]0,1[^2 \setminus [\frac{1}{2},1]^2$ and the PDE is

$$\text{Find } u \in H_0^1(\Omega) \text{ such that } \quad -\Delta u = 1 \text{ in } \Omega,$$

The solution has a singularity at the reentrant angle and we wish to capture it numerically.





example of Mesh adaptation

FreeFem++ corner singularity program

```
border a(t=0,1.0){x=t;   y=0;  label=1;};
border b(t=0,0.5){x=1;   y=t;  label=2;};
border c(t=0,0.5){x=1-t; y=0.5;label=3;};
border d(t=0.5,1){x=0.5; y=t;  label=4;};
border e(t=0.5,1){x=1-t; y=1;  label=5;};
border f(t=0.0,1){x=0;   y=1-t;label=6;};


mesh Th =  buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
fespace Vh(Th,P1);                  Vh u,v;          real error=0.01;
problem Probem1(u,v,solver=CG,eps=1.0e-6) =
     int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))  -   int2d(Th)( v)
     + on(1,2,3,4,5,6,u=0);
int i;
for (i=0;i< 7;i++)
{  Probem1;                                                  //    solving the pde problem
   Th=adaptmesh(Th,u,err=error);                          //    the adaptation with Hessian of u
   plot(Th,wait=1);            u=u;      error = error/ (1000^(1./7.)) ;   };
```

A cube with buildlayer (simple)

```
load "msh3"                                                    //    buildlayer
int nn=10;
int[int]

      rup=[0,2],      //    label of the upper face 0-> 2 (region -> label)
      rdown=[0,1],    //    label of the lower face 0-> 1 (region -> label)
      rmid=[1,1 ,2,1 ,3,1 ,4,1 ], //   4 Vert. faces: 2d label -> 3d label
      rtet[0,0];
real zmin=0,zmax=1;
mesh3 Th=buildlayers(square(nn,nn,),nn,
                     zbound=[zmin,zmax],
                     labeltet=rtet,
                     labelmid=rmid,
                     labelup = rup,
                     labeldown = rdown);
```

Execute Cube.edp

3D layer mesh of a Lac with buildlayer

```
load "msh3"//    buildlayer
load "medit"//    medit
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],
                     labelmid=rmid,  labelup = rup,
                     labeldown = rdown);                    //    label def
medit("lac",Th);
```

Execute Lac.edp Execute 3d-leman.edp

a Mesh of 3d fish with buildlayer

```
func f=2*((0.1+(((x/3))*(x-1)*(x-1)/1+x/100))^(1/3.)-(0.1)^(1/3.));
real yf=f(1.2,0);
border up(t=1.2,0.){ x=t;y=f;label=0;}
border axe2(t=0.2,1.15) { x=t;y=0;label=0;}
border hole(t=pi,0) { x= 0.15 + 0.05*cos(t);y= 0.05*sin(t); label=1;}
border axe1(t=0,0.1) { x=t;y=0;label=0;}
border queue(t=0,1) { x= 1.15 + 0.05*t; y = yf*t; label =0;}
int np= 100;
func bord= up(np)+axe1(np/10)+hole(np/10)+axe2(8*np/10)+ queue(np/10);
plot( bord);                                          //    plot the border ...
mesh Th2=buildmesh(bord);                             //    the 2d mesh axi mesh
plot(Th2,wait=1);
int[int] l23=[0,0,1,1];
Th=buildlayers(Th2,coef= max(.15,y/max(f,0.05)), 50 ,zbound=[0,2*pi]
  ,transfo=[x,y*cos(z),y*sin(z)],facemerge=1,labelmid=l23);
```

boundary mesh of a Sphere

```
load "tetgen"
mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]);                    //    ]-pi/2, -pi/2[x]0,2π[
func f1 =cos(x)*cos(y); func f2 =cos(x)*sin(y); func f3 = sin(x);
                                  //    the partiel derivative of the parametrization DF
func f1x=sin(x)*cos(y);    func f1y=-cos(x)*sin(y);
func f2x=-sin(x)*sin(y);   func f2y=cos(x)*cos(y);
func f3x=cos(x);           func f3y=0;
                                                                  //    M = DF^t DF
func m11=f1x^2+f2x^2+f3x^2;  func m21=f1x*f1y+f2x*f2y+f3x*f3y;
func m22=f1y^2+f2y^2+f3y^2;
func perio=[[4,y],[2,y],[1,x],[3,x]];
real hh=0.1/R; real vv= 1/square(hh);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);          //    4 times
int[int] ref=[0,L];                    //    to set the label of the Sphere to L ( 0 -> L)
mesh3  ThS= movemesh23(Th,transfo=[f1*R,f2*R,f3*R],orientation=1,label=ref);
```

Execute Sphere.edp Execute sphere6.edp

Build 3d Mesh from boundary 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 dering reading so 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"); }      //    save une mesh for next run
```

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

Build Matrix and vector of problem

The 3d FreeFem++ code :

```
mesh3 Th("dodecaedre.mesh");                          //    read a mesh from file
fespace Vh(Th,P13d);                                  //    define the P1 FE space

macro Grad(u) [dx(u),dy(u),dz(u)] //    End of Macro

varf vlaplace(u,v,solver=CG) =
   int3d(Th)( Grad(u)'*Grad(v) )  +  int3d(Th) ( 1*v)
  + on(2,u=2);                                                    //    on γ2

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

Remark on varf

The functions appearing in the variational form are formal and local to the `varf` definition, the only important think is 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 the 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 a vector, put $u1 = u2 = 0$ by setting 0 of on unknown part.

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

Remark : In this case the mesh use to defined , $\int, u, v$ can be different.

The boundary condition terms

First FreeFem use only the label number of the edge (2d) or the faces (3d).

- 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 giant value* `tgv` (=10^{30} by default) and the right hand side $b[i] =$ "$(\Pi_h g)[i]$" $\times\, tgv$, where the "$(\Pi_h g)g[i]$" 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)`.

An Not to simple exercice 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

$$J(u) = \iint_\Omega \left\| \begin{pmatrix} 1 \\ 0 \\ \partial_x u \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ \partial_y u \end{pmatrix} \right\|_2 dxdy = \iint_\Omega \sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2} dxdy$$

So the Euler equation associated to the minimization is :

$$\forall v / v_{|\Gamma} = 0 \quad : \quad DJ(u)v = -\int \frac{(\partial_x v \partial_x u + \partial_y v \partial_y u)}{\sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2}} = 0$$

So find the solution for $\Omega =]0, \pi[^2[$ and $g(x, y) = cos(2 * x) * cos(2 * y)$. by using the Non Linear Conjugate gradient `NLCG`. To speed up the algorithm you add a preconditionner to `NLCG`,

$$((\nabla J(u), v))_V = DJ(u)v, \quad \text{where} \quad ((u, v)) = \int \nabla u . \nabla v$$

or you can use a Newton method,

Tools

Example of use of NLCG function :

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

func real[int] DJ(real[int] &xx)                              //    the grad of functional
{ ....                         //    add code to copy xx array of finite element function
  ....                                                                          //
  return xx; };                                  //    return of an existing variable ok

func real[int] PreCon(real[int] &xx)                          //    the grad of functional
{ ....                         //    add code to copy xx array of finite element function
  ....                                                                          //
  return xx; };                                  //    return of an existing variable ok

NLCG(DJ,x,eps=1.e-6,nbiter=20,precon= PreCon);
```

Trick : if `uh` is a finite element function than `uh[]` is the vector of degree of freedom, so to convert `xx` on a finite element function do : `uh[]=xx;` .

Three solutions : First the functionnal

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


func real J(real[int] & xx)                                        //    the functionnal 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] & x)                                   //    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,uh=0);
    return xx= vDJ(0,Vh);  }


                                                                   //    the preconditionner
varf vlap(u,v)= int2d(Th)(  dx(v)*dx(u)+dy(v)*dy(u)) +on(1,u=0);
matrix Alap= vlap(Vh,Vh,solver=sparsesolver);
func real[int] lap(real[int] &u) {real[int] u1=Alap^-1*u; return u1;}
```

Three method

```
 Vh u=G;
if( case ==1)                                    //    very slow but no Gradient
  mincost=newuoa(J,u[],rhobeg=1,rhoend=1e-4,npt=2*n+1,maxfun=10000);
else if( case==2)
{
   NLCG(DJ,u[],eps=1.e-10,nbiter=200,precon=lap);
   mincost= J(u[]);
}
else {            //    by hand, gradient algorithm with precoditionner C.:
  .....
}
   plot(u,dim=3,wait=1);
```

Execute minimal-surf.edp Execute min-surf-3.edp

Fast method for 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

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

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

Fast method for Time depend Problem algorithm

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).

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

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

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

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

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

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

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 size 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 ∈ Γ_24   ⇔   bcn[i] ≠ 0
real[int]  bcl=tgv*u0[];          //    the Dirichlet boundary condition part
```

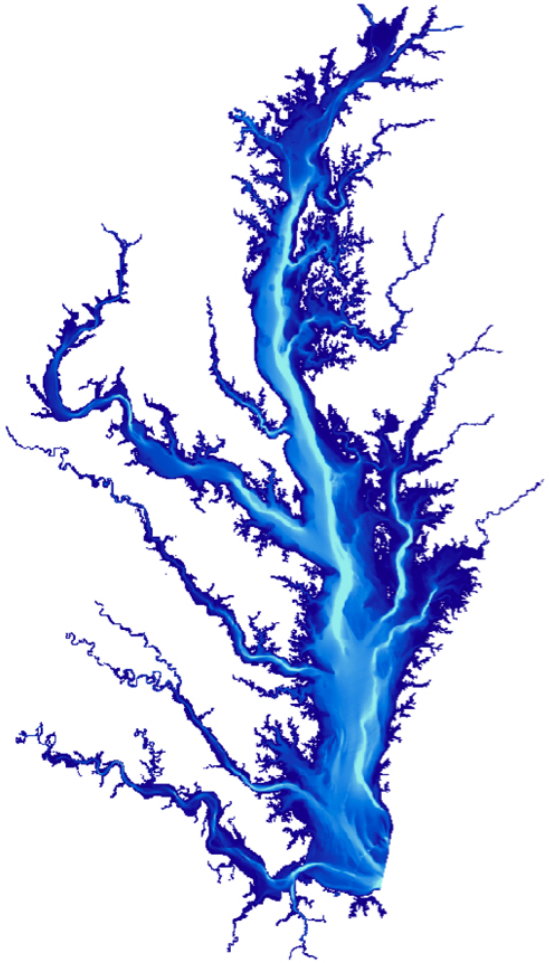
The Fast algorithm :

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

Execute Heat.edp

Some Trick to build meshes

The problem is to compute eigenvalue 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

$$10^{-5}\xi - \Delta\xi = 0 \quad \text{in } \Omega; \qquad \frac{\partial \xi}{\partial n} = 1 \quad \text{on } \Gamma.$$

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

$$\xi_{|\omega} \simeq 10^5 \frac{\int_{\partial\omega} 1}{\int_\omega 1}.$$

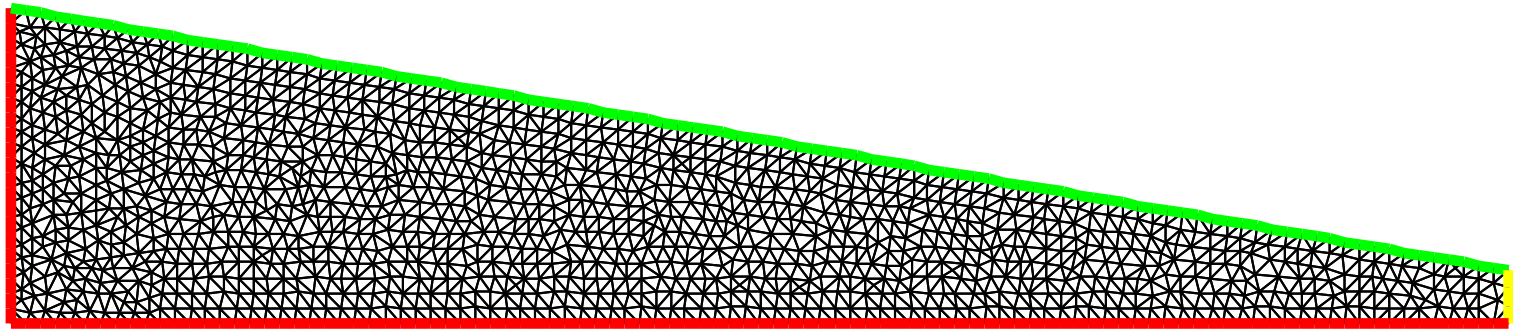
  Execute Chesapeake/Chesapeake-mesh.edp
- Solve the eigen value, on this mesh.
- Execute Chesapeake/Chesapeake-flow.edp

A Free Boundary problem , (phreatic water)

Let a trapezoidal domain Ω defined in FreeFem++ :

```
real L=10;                                                    //    Width
real h=2.1;                                                   //    Left height
real h1=0.35;                                                 //    Right height
border a(t=0,L){x=t;y=0;label=1;};    //    bottom impermeable Γ_a
border b(t=0,h1){x=L;y=t;label=2;};    //    right, the source Γ_b
border f(t=L,0){x=t;y=t*(h1-h)/L+h;label=3;}; //    the free surface Γ_f
border d(t=h,0){x=0;y=t;label=4;};    //    Left impermeable Γ_d
int n=10;
mesh Th=buildmesh (a(L*n)+b(h1*n)+f(sqrt(L^2+(h-h1)^2)*n)+d(h*n));
plot(Th,ps="dTh.eps");
```

The initial mesh



The problem is, find p and Ω such that :

$$\begin{cases} -\Delta p = 0 & \text{in } \Omega \\ p = y & \text{on } \Gamma_b \\ \dfrac{\partial p}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\ \dfrac{\partial p}{\partial n} = \frac{q}{K} n_x & \text{on } \Gamma_f \quad (Neumann) \\ p = y & \text{on } \Gamma_f \quad (Dirichlet) \end{cases}$$

where the input water flux is $q = 0.02$, and $K = 0.5$. The velocity u of the water is given by $u = -\nabla p$.

algorithm

We use the following fix point method : let be, $k = 0$, $\Omega^k = \Omega$. First step, we forgot the Neumann BC and we solve the problem : Find p in $V = H^1(\Omega^k)$, such $p = y$ onon Γ_b^k et on Γ_f^k

$$\int_{\Omega^k} \nabla p \nabla p' = 0, \quad \forall p' \in V \text{ with } p' = 0 \text{ on } \Gamma_b^k \cup \Gamma_f^k$$

With the residual of the Neumann boundary condition we build a domain transformation $\mathcal{F}(x, y) = [x, y - v(x)]$ where v is solution of : $v \in V$, such than $v = 0$ on Γ_a^k (bottom)

$$\int_{\Omega^k} \nabla v \nabla v' = \int_{\Gamma_f^k} (\frac{\partial p}{\partial n} - \frac{q}{K} n_x) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ sur } \Gamma_a^k$$

remark : we can use the previous equation to evaluate

$$\int_{\Gamma^k} \frac{\partial p}{\partial n} v' = -\int_{\Omega^k} \nabla p \nabla v'$$

The new domain is : $\Omega^{k+1} = \mathcal{F}(\Omega^k)$ Warning if is the movement is too large we can have triangle overlapping.

```
problem Pp(p,pp,solver=CG) = int2d(Th)( dx(p)*dx(pp)+dy(p)*dy(pp))
  + on(b,f,p=y) ;
problem Pv(v,vv,solver=CG) = int2d(Th)( dx(v)*dx(vv)+dy(v)*dy(vv))
  +  on (a, v=0) + int1d(Th,f)(vv*((Q/K)*N.y- (dx(p)*N.x+dy(p)*N.y)));
while(errv>1e-6)
{   j++;  Pp;  Pv;    errv=int1d(Th,f)(v*v);
     coef = 1;
    //    Here french cooking if overlapping see the example
    Th=movemesh(Th,[x,y-coef*v]);                          //    deformation
}
```

Execute freeboundary.edp

Eigenvalue/ Eigenvector example

The problem, Find the first λ, u_λ such that :

$$a(u_\lambda, v) = \int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega u_\lambda v = \lambda b(u_\lambda, v)$$

the boundary condition is make with exact penalization : we put $1e30 = tgv$ on the diagonal term of the lock degree of freedom. So take Dirichlet boundary condition only with a variational form and not on b variational form , because we compute eigenvalue of

$$w = A^{-1}Bv$$

Eigenvalue/ Eigenvector example code

```
...
fespace Vh(Th,P1);
macro Grad(u) [dx(u),dy(u),dz(u)]                                    //    EOM
varf  a(u1,u2)= int3d(Th)(  Grad(u1)'*Grad(u2) +  on(1,u1=0) ;
varf b([u1],[u2]) = int3d(Th)(  u1*u2 ) ;        //    no Boundary condition
matrix A= a(Vh,Vh,solver=UMFPACK),
        B= b(Vh,Vh,solver=CG,eps=1e-20);

int nev=40;                    //    number of computed eigenvalue close to 0
real[int] ev(nev);                                  //    to store nev eigenvalue
Vh[int] eV(nev);                                   //    to store nev eigenvector
int k=EigenValue(A,B,sym=true,value=ev,vector=eV,tol=1e-10);
k=min(k,nev);
for (int i=0;i<k;i++)
   plot(eV[i],cmm="Eigen 3d  Vector "+i+" valeur =" +
ev[i],wait=1,value=1);
```

Execute Lap3dEigenValue.edp

α periodic Finite element trick

Compute a Poisson problem in domain $]0,p[\times]-1,1[$ with α-periodic condition on vertical boundary Γ_p ($u(0,y) = \alpha u(p,y)$ where α is a complex number such that $\alpha = e^{2i\pi\gamma}$) and u is given on top and bottom border Γ_d.

Denote

$$V_\alpha = \{u \in H^1(\Omega); \quad u(0,y) = \alpha u(p,y)\}, \quad V_{0\alpha} = \{u \in V_\alpha; u_{\Gamma_d} = 0\}$$

Find $u \in V_\alpha$ and u given on Γ_d such that

$$\forall v \in V_{0\alpha}, \quad \int_\Omega \nabla u.\nabla \overline{v} - \int_{\Gamma_r} \overline{v}\frac{\partial u}{\partial \vec{n}} = \int_\Omega f\overline{v}, \tag{3}$$

denote $\varphi_\alpha = e^{2i\pi\gamma x/p}$ we have $u \in V_\alpha$ iff $\overline{\varphi_\alpha} u \in V_1$. So the problem is equivalent to find $u_\alpha = u\overline{\varphi_\alpha} \in V_1$ $u_\alpha\varphi_\alpha$ given on Γ_d such that

$$\forall v_\alpha \in V_{01}, \quad \int_\Omega \nabla_\alpha u_\alpha.\nabla_\alpha \overline{v_\alpha} - \int_{\Gamma_r} \overline{v_\alpha}\frac{\partial u_\alpha}{\partial \vec{n}} = \int_\Omega f\overline{v_\alpha\varphi_\alpha}, \tag{4}$$

where $\nabla_\alpha u = \nabla(\varphi_\alpha u) = \varphi_\alpha \nabla u + u\varphi_\alpha \Big|_0^{2i\pi\gamma/p}$

Coupling periodic Finite element / BEM

Compute a Poisson problem in semi ∞ domain $]0, p[\times]-1, \infty[$ with periodic condition on vertical boundary $u(0, y) = u(p, y)$.

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

$$\int_\Omega \nabla u.\nabla \overline{v} - \int_{\Gamma_b} \overline{v} \frac{\partial u}{\partial \vec{n}} = \int_\Omega f\overline{v}, \quad \forall v.. \tag{5}$$

The term contain the semi infinite modelization part.

Coupling periodic Finite element / BEM

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

$$f_n = exp(-2\pi(inx + |n|y)/p), \qquad \int_{\Gamma_b} f_n \overline{f_m} = p\delta_{mn}$$

. Remark $-\Delta f_n = 0,\ f_n(x, +\infty) = 0$, and we have

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

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

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

Trick to extract the degree of freedom on label 1 in know order

```
mesh Th=square(10,10);
fespace Vh(Th,[P2,P2]);                                    //    exemple of finite element
int[int] ib(0:Vh.ndof-1);                          //    arry ib[i]=i of size the number of DoF
{                          //    for memory management: all locals are remove at end of block
//    to get numbering in good order ..
//    for u1: x-10 alway <0
//    for u2: x-10 + 1e-10 alway <0 but a little gearter than u1 so after
//    after sort we have u1 u2 node 1,u1 u2 node 1, .... with node numbering x ↗
//    at the begin of the array

varf vbord([u1,u2],[v1,v2]) = on(1,u1=x-10,u2=x-10+1e-10);
real[int] xb=vbord(0,Vh,tgv=1);                    //    get the interpolation on border 1 .
sort(xb,ib);                                         //    sort of 2 array in parallel
//    so now the begin of the array is the dof on label 1 and in ib we have the numbering
xb = xb? 1 : 0;
int ndofon1 = xb.sum +0.5;                         //    to count the number of non zero.
ib.resize(ndofon1);                                          //    resize of the array
}
cout << ib << endl;
```

Execute BEM.edp

Schwaz algorithm

Let Ω_i $i \in I$ an overlaping of Ω, Let P_i a partition of the unite ($\sum P_i = 1$ on Ω) such that $supp(P_i) \subset \Omega_i$ To solve :

$$-\Delta u = f \text{ in } \Omega, \qquad u = 0 \text{ on } \partial\Omega$$

the algorithm is just : let u^0 a given solution, and $n = 0$

For all $i \in I$ solve :

$$\begin{aligned} -\Delta u_i^{n+1} &= f && \text{in } \Omega_i \\ u_i^{n+1} &= u^n && \text{on } \partial\Omega_i \cap \Omega \\ u_i^{n+1} &= 0 && \text{on } \partial\Omega_i \cap \partial\Omega \end{aligned}$$

$n = n + 1$

$$u^n = \sum_{i \in I} P_i u_i^n$$

To build the partition with metis, the overlapping is simply done with a loop n times of a successive L^2 projection $P_0 \mapsto P_1 \mapsto P_0$ to have n layer overlapping.

Execute schwarz-nm.edp

Execute schwarz-3.edp

Quelques exemples 3D

- Execute examples++-3d/beam-3d.edp
- Execute examples++-3d/3d-Leman.edp
- Execute examples++-3d/Lac.edp
- Execute examples++-3d/Laplace3d.edp
- Execute examples++-3d/LaplaceRT-3d.edp
- Execute examples++-3d/NSI3d.edp
- Execute examples++-3d/Period-Poisson-cube-ballon.edp
- Execute examples++-3d/Poisson-cube-ballon.edp
- Execute examples++-3d/Poisson3d.edp
- Execute examples++-3d/Stokes.edp
- Execute examples++-3d/convect-3d.edp
- Execute examples++-3d/cube-period.edp

Quelques exemples avec PLUGIN

- Execute examples++-load/metis.edp
- Execute examples++-load/LaplaceP4.edp
- Execute examples++-load/NSP2BRP0.edp
- Execute examples++-load/SuperLU.edp
- Execute examples++-load/bilapMorley.edp
- Execute examples++-load/buildlayermesh.edp
- Execute examples++-load/plot-fb-P4dc.edp
- Execute examples++-load/provadxw.edp
- Execute examples++-load/refinesphere.edp
- Execute examples++-load/tetgencube.edp
- Execute examples++-load/tetgenholeregion.edp
- Execute examples++-load/tetgenholeregion_rugby.edp

Quelques examples 2D

- Execute BlackScholes2D.edp
- Execute cavityNewtow.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-menbrane-adap.edp

Exemple //

Execute examples++-mpi/schwarz-3.edp

Execute schwarz-nm.edp

Stokes equation

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

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

where $\boldsymbol{u}_\Gamma$ is a given velocity on boundary Γ.

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

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

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

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

Stokes equation in FreeFem++

```
 ...  build 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 stabilization term. The term to be add to the classical variational formulation is :

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

with

$$R_h = Id - I_h P_h$$

and where

V_h P_1 continuous finite space

V_h^{dc} P_1 fully discontinuous finite space

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

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

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

How to build R_h in FreeFem++

```
matrix Ih = interpolate(Vdch,Vh);
matrix Ph = interpolate(Vh,Vdch);
if(!scootzhang)
   {            //    Clement's Operator or L2 projection with mass lumping
     varf vsigma(u,v)=int2d(Th)(v);
     Vh sigma; sigma[]=vsigma(0,Vh);                          //    σ_i = ∫_Ω w^i
     varf vP2L(u,v)=int2d(Th,qft=qf1pTlump)(u*v/sigma); //    P1 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
```

How to 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);              //    ∂_x p discrete operator
  matrix Dy = interpolate(Vdch,Wh,U2Vc=c2,op=2);              //    ∂_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

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

$$\begin{array}{ll} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{6}$$

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 χ=[X1,X2,X3] vector
    b=vNS(0,VVh);                                    //    build NS rhs
    u1[]= A^-1 * b;                             //    solve the linear system
    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

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 piece of 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; return a;}
double MyFunction( double const & p,KN_<double> const & x) { return x.lp(p); }

class Init { public: Init(); }; //    C++ trick to call a method at load time
Init init;                      //    a global variable to inforce the initialization 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));
   Global.Add("lp",new OneOperator2_<double, double, KN_<double> > (MyFunction);}
```

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

Usage : `real[int] b(10); ...; real norm=lp(10.,b);`

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 < lestables
     Cmatrix 293 Matrice_Creuse<std::complex<double> >
     R3 293 Fem2D::R3
     bool 293 bool*
     complex 293 std::complex<double>*
     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*
     string 293 std::basic_string<char, std::char_traits<char>, std::allocator<char> >**
```

FreeFem++ Triangle/Tet capability

```
                                                //    soit T un Element de sommets A, B, C ∈ ℝ²
                                                //    ----------------------------------
Element::nv ;                                   //    nombre de sommets d'un triangle (ici 3)
const Element::Vertex & V =  T[i];              //    le sommet i de T (i ∈ 0, 1, 2
double a = T.mesure() ;                         //    mesure de T
Rd AB = T.Edge(2);                              //    "vecteur arête" de l'arete
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
```

FreeFem++ Mesh/Mesh3 capabylity

```
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 ∇λ_i^K pour i = 0, 1, 2
Mesh::Vertex   & V = Th(j);                   //    sommet j , int j∈ [0, nv[
Mesh::BorderElement  &   BE=th.be(l) ;        //    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
```

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 (do in 2d, to do in 3d)
– 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 ?