

# FreeFem++ un logiciel de résolution EDP en 2 et 3d

F. Hecht

Laboratoire Jacques-Louis Lions

Université Pierre et Marie Curie

Paris, France

ANR-07-CIS7-002-01

<http://www.freefem.org>

<mailto:hecht@ann.jussieu.fr>



## PLAN

- Introduction [Freefem++](#)
- Mathematical formulation of Poisson with Neuman condition
- Poisson equation in 3D
- Poisson equation with matrix
- varitionnal form (Matrix and vector )
- mesh generation in 3d
- an academic exercise problem of minimal surface
- Stokes Problem
- Navier Stokes Incompressible in 3D
- Dynamic Link example (hard)

<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.

## 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, vectorial element , mini-element, ...
- **Automatic interpolation** of data from a mesh to an other one, so a finite element function is view as a function of  $(x, y)$  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.

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

- Analytic description of boundaries, with specification by the user of the intersection of boundaries.
- **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** sparse linear solver ; **eigenvalue** and eigenvector computation with ARPACK.
- Online graphics, C++ like syntax.
- Link with other soft : modulef, emc2, medit, gnuplot, ...
- 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 I/II

```
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,P1) ; // a 2d finite element space ;
fespace Vh3(Th3,P13d) ; // a 3d finite element space ;

Vh u=x ; // a finite element function or array
Vh<complex> uc = x+ 1.i *y ; // complex valued FE function or array
u(.5,0.6) ; // value of FE function u at point (.5,.6)
u[] ; // the array associated to FE function u
u[][5] ; // 6th value of the array ( numbering begin at 0 like in C)
func f=x+y ; // a formal line function
func real g(int i, real a) { ..... ; return i+a ;}
```

## Build Mesh 2d

First a  $10 \times 10$  grid mesh of unit square  $]0, 1[{}^2$

```
mesh Th1 = square(10,10) ; // boundary label:  
plot(Th1,wait=1) ; // 1 bottom, 2 right, 3 top, 4, left
```

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

```
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 Neumann Poisson Problem with 1D lagrange multiplier

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

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

```
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 systeme
plot(uh,wait=1); // set the value
```

Execute Laplace-lagrange-mult.edp

# The plot of the Finite Basis Function

```
load "Element_P3"
mesh Th=square(1,1);
mesh th=square(150,150);
fespace Vh(Th,P2);

Vh vi=0;
for (int i=0;i<vi[].n;++i)
{
    vi[][i]=1;

    plot(vi,wait=0,cmm=" v"+i);
    savemesh(th,"mm",[x,y,vi*0.5]); // save files for medit
    exec("medit mm;rm mm.faces mm.points");
    vi[]=0;
}
}
```

Execute [plot-fb.edp](#)

# Laplace equation

The 3d FreeFem++ code :

```
mesh3 Th("dodecaedre.mesh");           // read th mesh from a file
fespace Vh(Th,P13d);                    // 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);                             // on  $\gamma_2$ 
```

# Matrix and vector

The first 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

## Metric / unit Mesh

In Euclidean geometry the length  $|\gamma|$  of a curve  $\gamma$  of  $\mathbb{R}^d$  parametrized by  $\gamma(t)_{t=0..1}$  is

$$|\gamma| = \int_0^1 \sqrt{\langle \gamma'(t), \gamma'(t) \rangle} dt$$

We introduce the metric  $\mathcal{M}(x)$  as a field of  $d \times d$  symmetric positive definite matrices, and the length  $\ell$  of  $\Gamma$  w.r.t  $\mathcal{M}$  is :

$$\ell = \int_0^1 \sqrt{\langle \gamma'(t), \mathcal{M}(\gamma(t)) \gamma'(t) \rangle} dt$$

The key-idea is to construct a mesh where the lengths of the edges are close to 1 accordingly to  $\mathcal{M}$ .

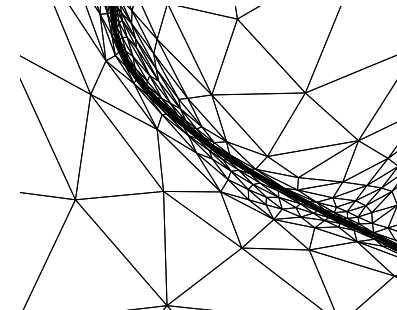
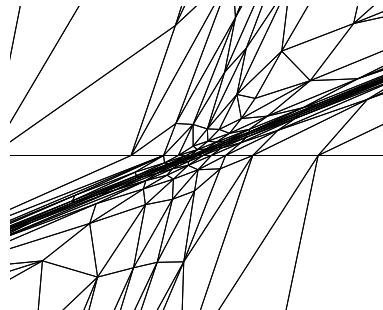
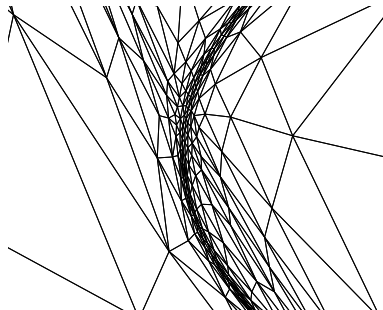
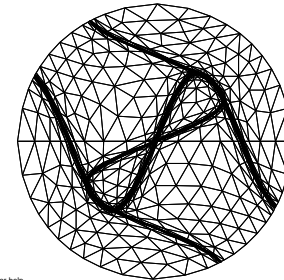
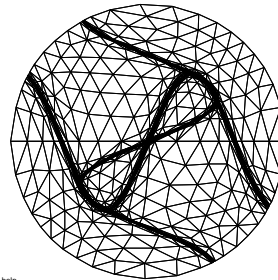
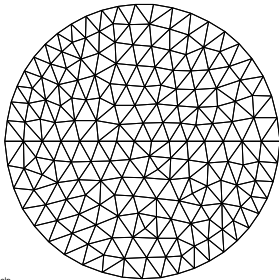
## The main IDEA for mesh generation

- The difficulty is to find a tradeoff between the error estimate and the mesh generation, because these two works are strongly different.
- To do that, we propose a way based on a metric  $\mathcal{M}$  and unit mesh w.r.t  $\mathcal{M}$
- The metric is a way to control the mesh size.
- remark : The class of the mesh which can be created by the metric, is very large.

## Example of mesh

$$f_1 = (10 * x^3 + y^3) + atan2(0.001, (\sin(5 * y) - 2 * x))$$

$$f_2 = (10 * y^3 + x^3) + atan2(0.01, (\sin(5 * x) - 2 * y)).$$



## A cube

```
load "msh3" // buildlayer
int nn=10 ;
mesh Th2=square(nn,nn) ;
int[int] rup=[0,2], rdown=[0,1], rmid=[1,1,2,1,3,1,4,1] ;
real zmin=0,zmax=1 ;

mesh3 Th=buildlayers(Th2,nn,
                    zbound=[zmin,zmax],
                    // refacet=r1,
                    reffacemid=rmid,
                    reffaceup = rup,
                    reffacedown = rdown) ;

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

Execute Cube.edp

## 3D mesh of a ball

```
load "tetgen"
mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]); // ] $-\frac{\pi}{2}, \frac{\pi}{2}$ [ $\times$ ]0,2 $\pi$ [
// a parametrization of a sphere
func f1 =cos(x)*cos(y); func f2 =cos(x)*sin(y); func f3 = sin(x);
// the partial 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;
func m11=f1x^2+f2x^2+f3x^2; //  $M = DF^t DF$ 
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, vv= 1;/square(hh);
// build a good mesh of the sphere
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
mesh3 Th3=tetgtransfo(Th,transfo=[f1,f2,f3]);
```

Execute Sphere.edp

## 3D layer mesh of a Lac / II

```
load "msh3"//      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,
                    reffacedown = rdown);           //      label def
savemesh(Th,"Th.meshb");
exec("medit Th;rm Th.meshb");
```

Execute Lac.edp

# Stokes equation

The Stokes equation is find a velocity field  $\mathbf{u} = (u_1, \dots, u_d)$  and the pressure  $p$  on domain  $\Omega$  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  $\mathbf{u}_\Gamma$  is a given velocity on boundary  $\Gamma$ .

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

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

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

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

# Stokes equation in FreeFem++

```
int nn=10 ;                mesh Th2=square(nn,nn) ;
fespace Vh2(Th2,P2) ;      Vh2 ux,uz,p2 ;
mesh3 Th=buildlayers(Th2,nn) ; savemesh(Th,"c10x10x10.mesh") ;
fespace VVh(Th,[P23d,P23d,P23d,P13d]) ; // 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(1,2,3,4,6,u1=0,u2=0,u3=0) + on(5,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 ;
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") ;
```

Execute Stokes3d.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

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

The term  $X^n(x) \approx x - u^n(x)\tau$  will be computed by the interpolation operator.

## 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 )) ;
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

## An 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

$$\arg \min J(u) = \int_{\Omega} \left\| \begin{pmatrix} 1 \\ 0 \\ \partial_x u \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ \partial_y u \end{pmatrix} \right\| d\Omega = \int_{\Omega} \sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2} d\Omega$$

So the Euler equation associated to the minimisation is :

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

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 NLCG like in the example : `algo.edp` in `examples++-tutorial`.

# Tools

Example of use of NLCG function :

```
func real J(real[int] & xx) // the fonctionnal 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 fonctionnal
{ .... // 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=matId) ;
```

Two useful operators on array :real[int]

```
real[int] a(10),b(10) ;
...
a = b? 1. : 0; // a[i] = 1 if b[i] else a[i]=0.  $\forall i$ 
```

To see the 3D plot of the surface with medit

```
savemesh(Th,"mm",[x,y,u]); // save mm.points and mm.faces file for medit
exec("medit mm;rm mm.bb mm.faces mm.points"); // call medit
```

## 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 au(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= au(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
savemesh(Th,"mm",[x,y,u]) ; // save surface mesh for medit
exec("medit mm;rm mm.bb mm.faces mm.points") ;
```

Execute minimal-surf.edp

## Conclusion and Future

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

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

Thank, for your attention ?