FreeFem++

F. Hecht, O. Pironneau

Laboratoire Jacques-Louis Lions

Université Pierre et Marie Curie

Paris, France

Olivier Pantz

http://www.freefem.org

mailto:hecht@ann.jussieu.fr

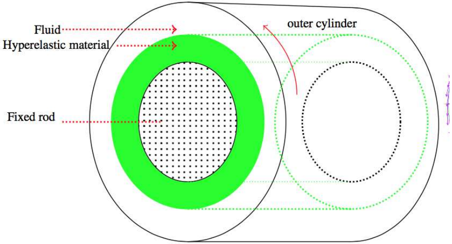
Introduction I suggest you choose, according to your level

- Beginners : Follow the slides
- Others 1 : Follow today's example
- Others 2 : Suggest an example and receive help to program it.

Today's example :

$$\rho\partial_t v - \frac{1}{r}\partial_r[\xi^f r\partial_r v + \xi^s r\partial_r d] = 0, \quad \partial_t d = v, \ r \in (R_0, R_1), \quad v_{|R_0} = 0, \ v_{|R_1} = 3,$$

with $\rho = \rho^s \mathbf{1}_{r\le R} + \rho^f \mathbf{1}_{r>R}$, $\xi^s = 2c_1\mathbf{1}_{r\le R}$, $\xi^f = \mu\mathbf{1}_{r>R}$, and with $d(r,0) = 0$.



Example

A cylinder contains a fixed rigid cylindrical rod in its center, a cylindrical layer of hyperelastic material around the rod and the rest is filled with a fluid (see figure 3). First the system is at rest and then a constant rotation is given to the outer cylinder. This cause the fluid to rotate with an angular velocity which depends on the distance r to the main axis; in turn because the friction of the fluid at the interface the hyperelastic material will be dragged into a angular velocity ω which is also only a function of r and time . Due to elasticity ω will oscillate with time until numerical dissipation and fluid viscosity damps it.

In a two dimensional cut perpendicular to the main axis, the velocities and displacements are two dimensional as well. Hence the geometry is a ring of inner and outer radii, R_0 and R_1, with hyperelastic material between R_0 and R and fluid between R and R_1. Because of the incompressibility of the fluid and axial symmetry, R is constant.

In this test $R_0 = 3$, $R = 4$, $R_1 = 5$. The solid is an hyperelastic incompressible material with $c_1 = 2$ and $\rho^s = 2$. The Newtonian fluid has $\mu = 2$, $\rho^f = 1$. The velocity of the outer cylinder has magnitude 3. As everything is axisymmetric the computation can be done in polar coordinates r, θ, and the fluid-solid system reduces to

$$\rho\partial_t v - \frac{1}{r}\partial_r[\xi^f r\partial_r v + \xi^s r\partial_r d] = 0, \quad \partial_t d = v, \; r \in (R_0, R_1), \quad v_{|R_0} = 0, \; v_{|R_1} = 3, \tag{24}$$

with $\rho = \rho^s \mathbf{1}_{r\le R} + \rho^f \mathbf{1}_{r>R}$, $\xi^s = 2c_1\mathbf{1}_{r\le R}$, $\xi^f = \mu\mathbf{1}_{r>R}$, and with $d(r,0) = 0$.

Example

```
load "pipe"
real R0=1,R1=2,R2=3,rhof=1,rhos=2,nu=1,kappa=5,v=3,T=0.8,dt=0.025;
          //   / semi-analytic solution by solving a 1d problem //
mesh Th=square(100,5,[R0+(R2-R0)*x,0.1*y]);
fespace Wh(Th,P1,periodic=[[1,x],[3,x]]);
fespace W0(Th,P0);
Wh d=0,w,wh,wold=0;
W0 nnu=nu*(x>R1)+2*kappa*dt*(x<=R1), Rho=rhof*(x>R1)+rhos*(x<=R1);
problem AA(w,wh) = int2d(Th)(Rho*x*w*wh/dt+x*nnu/2*dx(w)*dx(wh) )
+ int2d(Th)(-Rho*x*wold*wh/dt +x*nnu/2*dx(wold)*dx(wh)
+ 2*kappa*(x<=R1)*(x*dx(d)*dx(wh) ))
+on(2,w=-v)+on(4,w=0); //    this is the one-d axisymmetric problem
pstream pgnuplot("gnuplot" );        //   //// prepare gnuplot ////
int J=40;  real dr = (R2-R0)/(J-1);
for(int i=0;i<T/dt;i++){
    AA; d=d+(w+wold)*dt; wold=w;
    ofstream f("aux.gp");
    for(int j=0;j<J;j++) f << j*dr <<"   " << w(R0+j*dr,0.05) << endl;
    pgnuplot << "plot 'aux.gp' u 1:2 w l "<< endl;
    sleep(1); flush(pgnuplot);
}
```

Outline

Introduction

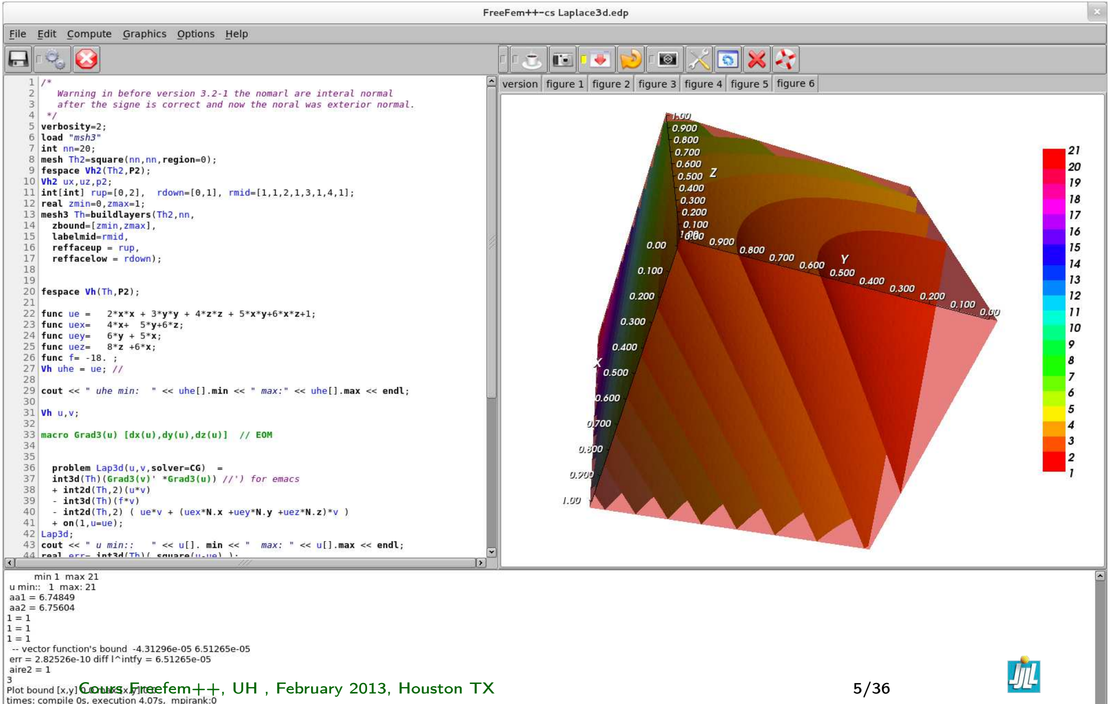
- Except `COMSOL` software to solve PDE are application oriented, like `NASTRAN,PamCrash,Abaqus,Fluent,OpenFOAM` etc.
- `FreeFem++` is a software born in 2000 to solve numerically partial differential equations (PDE) in $\mathbb{R}^2$) and in $\mathbb{R}^3$) with finite elements methods.
- It has its own language, as close to the mathematics as possible : the `FreeFem++ script` which overwrites C++.
- All PDEs are specified in variational form.
- At the root `FreeFem++` solves linear steady state problem and convection problems.
- For time depend, nonlinear and coupled problem the user must specify the algorithm.
- It can do mesh adaptation , compute error indicator, etc ...

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

The main characteristics (2D)(3D)

- Wide range of finite elements : continuous P1,P2 elements, discontinuous P0, P1, RT0,RT1,BDM1, elements ,Discontinuous-Galerkin, ...
- 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 on the degree of Freedom.
- Complex or real functions with access to vectors and the matrices.
- LU, Cholesky, Crout, CG, GMRES, UMFPack, SuperLU, MUMPS, HIPS , SUPERLU_DIST, PASTIX. ... sparse linear solver ; eigenvalue and eigenvector computation with ARPACK.
- Optimization Algorithms : GMRES, IPOPT, MEWUOA, CMAES etc.
- Automatic mesh generator, based on Delaunay-Voronoi. (2d,3d (tetgen))
- Mesh adaptation based on automatic metric, possibly anisotropic (only in 2d).
- Link with other soft : paraview, gmsh , vtk, medit, gnuplot
- Dynamic linking to add plugin. Full MPI interface
- Online graphics with OpenGL/GLUT/VTK, C++ like syntax.
- An integrated development environment FreeFem++-cs (ann.jussieu.fr/lehyaric/ffcs)

Freefem++-CS integrated environment



Element of syntax 1/2

```
x,y,z , label, N.x, N.y, N.z,        //    some reserved variables
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;                                          //    a 2d mesh
fespace Vh(Th,P2);           //   Def. of 2d finite element space;
                 Vh u=x;  //    a finite element function or array
fespace V3h(Th,[P2,P2,P1]); V3h u;
                                         u(.5,.6,.7); u[];
Vh3<complex> uc = x+  1i *y;                 //    complex valued FE
V3h [u1,u2,p]=[x,y,z];              //   a vectorial finite element
macro div(u,v) (dx(u)+dy(v))                      //   EOM a macro
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 syntax 2/2

```
 func Heaveside=(x>0);                 //    a formal line function
 func real g(int i, real a) { .....; return i+a;}
 A = A + A'; A = A'*A; A = [ [ A,0],[0,A'] ];
int[int] I(15),J(15);                  //    two array for renumbering
matrix B;
B = 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
 int[int] I(1),J(1); real[int] C(1);
[I,J,C]=A;           //    get of the sparse term of the matrix A
A=[I,J,C];                             //    set a new matrix
matrix D=[diagofA];                    //    set a diagonal matrix D
real[int] a=2:12;                      //    set a[i]=i+2; i=0 to 10.
```

a formal array is `[ exp1, exp1, ..., expn]`

```
complex a=1,b=2,c=3i;
func va=[ a,b,c];                      //    is a formal array in [ ]
a =[ 1,2,3i]'*va ; cout « a « endl;    //    hermien product
matrix<complex>  A=va*[ 1,2,3i]'; cout « A « endl;
a =[ 1,2,3i]'*va*2.;    a =(va+[ 1,2,3i])'*va*2.;
va./va; va*/va;             //    term to term / and term to term *
```

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 :

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

Denote $V_g = \{v \in H^1(\Omega) / v_{|\Gamma_2} = g\}$.
The Basic variationnal formulation with is : find $u \in V_g(\Omega)$, such that

$$\int_\Omega \nabla u . \nabla v = \int_\Omega f v + \int_\Gamma h v, \quad \forall v \in V_0(\Omega) \tag{2}$$

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

Poisson equation in a fish with FreeFem++

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

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

 Vh u,v;              //    set test and unknown function in Vh.
macro Grad(u) [dx(u),dy(u),dz(u)]                    //    EOM Grad def
func f=1;
solve laplace(u,v,solver=CG) =
   int3d(Th)(  Grad(u)'*Grad(v)    )
   - int3d(Th) ( f*v)
   + on(2,u=2);                                        //    int on γ2
plot(u,fill=1,wait=1,value=0,wait=1);
```

Run:fish.edp   Run:fish3d.edp

Build Mesh 2d

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

boundary mesh of a Sphere

```
load "tetgen"
mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]);        //    ]-pi/2, -pi/2[×]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);
int[int] ref=[0,L];    //    the label of the Sphere to L ( 0 -> L)
mesh3  ThS= movemesh23(Th,transfo=[f1*R,f2*R,f3*R],orientation=1,
   label=ref);
```

Run:Sphere.edp Run:sphere6.edp

Mesh tools

- `change` to change label and region numbering in 2d and 3d.
- `movemesh checkmovemesh movemesh23 movemesh3`
- `triangulate` (2d) , `tetgconvexhull` (3d) build mesh mesh for a set of point
- `emptymesh` (2d) built a empty mesh for Lagrange multiplier
- `freeyams` to optimize surface mesh
- `mmg3d` to optimize volume mesh with constant surface mesh
- `mshmet` to compute metric
- `isoline` to extract isoline (2d)
- `trunc` to remove peace of mesh and split all element (2d,3d)
- `splitmesh` to split 2d mesh in no regular way.

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

```
int[int] lab=[1,1,1,1];
mesh Th = square(6,6,label=lab);
Th=trunc(Th,x<0.5 | y<0.5, label=1);

fespace Vh(Th,P1);          Vh u,v;          real error=0.1;
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,u=0);
for (int i=0;i< 7;i++)
{  Probem1;                                  //    solving the pde
   Th=adaptmesh(Th,u,err=error,nbvx=100000);
                           //    the adaptation with Hessian of u
   plot(Th,u,wait=1,fill=1);           u=u;
   error = error/ (1000.^(1./7.)) ;   };
```

Run:CornerLap.edp

Poisson equation with 3d mesh adaptation

```
load "msh3" load "tetgen" load "mshmet" load "medit"
int nn  = 6;
int[int] l1111=[1,1,1,1],l01=[0,1],l11=[1,1];                   //   label numbering
mesh3 Th3=buildlayers(square(nn,nn,region=0,label=l1111),
      nn,  zbound=[0,1], labelmid=l11,labelup = l01,labeldown = l01);
Th3=trunc(Th3,(x<0.5)|(y < 0.5)|(z < 0.5) ,label=1);            //   remove ]0.5,1[^3

fespace Vh(Th3,P1);       Vh u,v;                 //   FE. space definition
macro Grad(u) [dx(u),dy(u),dz(u)]                                 //   EOM
problem Poisson(u,v,solver=CG) =
  int3d(Th3)( Grad(u)'*Grad(v) )  -int3d(Th3)( 1*v ) + on(1,u=0);

real errm=1e-2;                                        //   level of error
for(int ii=0; ii<5; ii++)
{
  Poisson;        Vh h;
  h[]=mshmet(Th3,u,normalization=1,aniso=0,nbregul=1,hmin=1e-3,
             hmax=0.3,err=errm);
  errm*= 0.8;                              //   change the level of error
  Th3=tetgreconstruction(Th3,switch="raAQ"
        ,sizeofvolume=h*h*h/6.);
  medit("U-adap-iso-"+ii,Th3,u,wait=1);
}
```

Run:Laplace-Adapt-3d.edp

Linear Lame equation, weak form

Let a domain $\Omega \subset \mathbb{R}^d$ with a partition of $\partial\Omega$ in Γ_d, Γ_n.
Find the displacement $\boldsymbol{u}$ field such that :

$$-\nabla.\sigma(\boldsymbol{u}) = \boldsymbol{f} \text{ in } \Omega, \quad \mathbf{u} = \mathbf{0} \text{ on } \mathbf{\Gamma_d}, \quad \sigma(\mathbf{u}).\mathbf{n} = \mathbf{0} \text{ on } \mathbf{\Gamma_n} \tag{3}$$

Where $\varepsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla\boldsymbol{u} + {}^t\nabla\boldsymbol{u})$ and $\sigma(\boldsymbol{u}) = \boldsymbol{A}\varepsilon(\boldsymbol{u})$ with $\boldsymbol{A}$ the linear positif operator on symmetric $d \times d$ matrix corresponding to the material propriety. Denote $V_{\boldsymbol{g}} = \{\boldsymbol{v} \in H^1(\Omega)^d / \boldsymbol{v}_{|\Gamma_d} = \boldsymbol{g}\}$.
The Basic displacement variational formulation is : find $\boldsymbol{u} \in V_0(\Omega)$, such that :

$$\int_\Omega \varepsilon(\boldsymbol{v}) : \boldsymbol{A}\varepsilon(\boldsymbol{u}) = \int_\Omega \boldsymbol{v}.\boldsymbol{f} + \int_\Gamma ((\boldsymbol{A}\varepsilon(\boldsymbol{u})).n).v, \quad \forall \boldsymbol{v} \in V_0(\Omega) \tag{4}$$

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

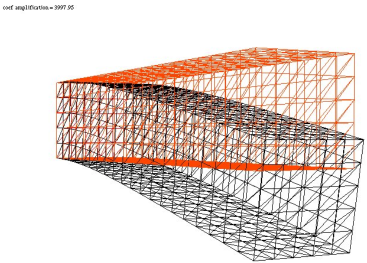
```
load "medit"   include "cube.idp"
int[int]  Nxyz=[20,5,5];
real [int,int]  Bxyz=[[0.,5.],[0.,1.],[0.,1.]];
int [int,int]  Lxyz=[[1,2],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);
                                                          //    Alu ...
real rhoAlu = 2600, alu11=  1.11e11 , alu12 = 0.61e11;
real alu44= (alu11-alu12)*0.5;
func Aalu = [  [alu11, alu12,alu12,   0.   ,0.   ,0.  ],
               [alu12, alu11,alu12,   0.   ,0.   ,0.   ],
               [alu12, alu12,alu11,   0.   ,0.   ,0.   ],
               [0. ,   0. ,   0. ,    alu44,0.   ,0.   ],
               [0. ,   0. ,   0. ,    0.   ,alu44,0.   ],
               [0. ,   0. ,   0. ,    0.   ,0.   ,alu44]    ];
real gravity = -9.81;
```

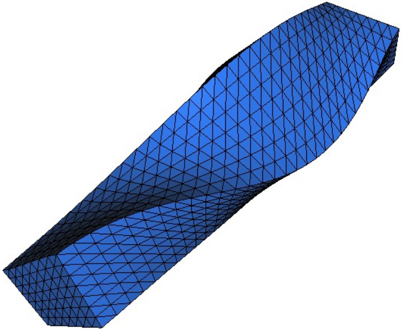
Linear elasticity equation, in FreeFem++

```
fespace Vh(Th,[P1,P1,P1]);
Vh [u1,u2,u3], [v1,v2,v3];
macro Strain(u1,u2,u3)
  [ dx(u1), dy(u2),dz(u3),
  (dz(u2) +dy(u3)), (dz(u1)+dx(u3)),
   (dy(u1)+dx(u2)) ]                                                  //    EOM
solve Lame([u1,u2,u3],[v1,v2,v3])=
  int3d(Th)(
      Strain(v1,v2,v3)'*(Aalu*Strain(u1,u2,u3))   )
- int3d(Th) ( rhoAlu*gravity*v3)
      + on(1,u1=0,u2=0,u3=0) ;

real coef= 0.1/u1[].linfty;  int[int] ref2=[1,0,2,0];
mesh3 Thm=movemesh3(Th,
     transfo=[x+u1*coef,y+u2*coef,z+u3*coef],
     label=ref2);
plot(Th,Thm, wait=1,cmm="coef  amplification = "+coef );
medit("Th-Thm",Th,Thm);
```

Lame equation / figure





Run:beam-3d.edp Run:beam-EV-3d.edp Run:beam-3d-Adapt.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

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

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

$$\forall \boldsymbol{v} \in H^1_0(\Omega)^d,\ \forall q \in L^2(\Omega)/\mathbb{R}, \qquad \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - p \nabla . v - q \nabla . u = 0$$

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

$$\forall \boldsymbol{v} \in H^1_0(\Omega)^d,\ \forall q \in L^2(\Omega), \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - p \nabla . v - q \nabla . u + \varepsilon p q = 0$$

Stokes equation in FreeFem++

```
 ...  build mesh .... Th (3d) T2d ( 2d)
fespace VVh(Th,[P2,P2,P2,P1]);          //    Taylor Hood FE.
macro Grad(u) [dx(u),dy(u),dz(u)]          //    EOM
macro div(u1,u2,u3) (dx(u1)+dy(u2)+dz(u3))      //   EOM
VVh [u1,u2,u3,p],[v1,v2,v3,q];
solve 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);
```

Run:Stokes3d.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

$$w = A^{-1} B v$$

Otherwise we get spurious mode.
Arpack interface :

```
int k=EigenValue(A,B,sym=true,value=ev,vector=eV);
```

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 BC
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);
k=min(k,nev);
for (int i=0;i<k;i++)
   plot(eV[i],cmm="ev  "+i+" v =" + ev[i],wait=1,value=1);
```

Execute Lap3dEigenValue.edp   Execute LapEigenValue.edp

Ipopt optimizer

The IPOPT optimizer in a FreeFem++ script is done with the `IPOPT` function included in the `ff-Ipopt` dynamic library. IPOPT is designed to solve constrained minimization problem in the form :

$$\begin{array}{ll} \text{find} & x_0 = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} f(x) \\ \text{s.t.} & \left\{ \begin{array}{ll} \forall i \leq n, \; x_i^{\mathrm{lb}} \leq x_i \leq x_i^{\mathrm{ub}} & \text{(simple bounds)} \\ \forall i \leq m, \; c_i^{\mathrm{lb}} \leq c_i(x) \leq c_i^{\mathrm{ub}} & \text{(constraints functions)} \end{array} \right. \end{array}$$

Where ub and lb stand for "upper bound" and "lower bound". If for some $i, 1 \leq i \leq m$ we have $c_i^{\mathrm{lb}} = c_i^{\mathrm{ub}}$, it means that c_i is an equality constraint, and an inequality one if $c_i^{\mathrm{lb}} < c_i^{\mathrm{ub}}$.

```
func real J(real[int] &X) {...}                          //   Fitness Function,
func real[int] gradJ(real[int] &X) {...}                           //   Gradient
func real[int] C(real[int] &X) {...}                            //   Constraints
func matrix jacC(real[int] &X) {...}                   //   Constraints jacobian
matrix jacCBuffer;                       //   just declare, no need to define yet
func matrix jacC(real[int] &X)
  {...  return jacCBuffer; }                               //   fill jacCBuffer
```

Stochastic Optimizer

This algorithm works with a normal multivariate distribution in the parameters space and try to adapt its covariance matrix using the information provides by the successive function evaluations. Syntaxe : `cmaes(J,u[],..)` ( )
From `http://www.lri.fr/~hansen/javadoc/fr/inria/optimization/cmaes/package-summary.html`

```
load "mpi-cmaes"

real mini = cmaesMPI(J,start,stopMaxFunEval=10000*(al+1),
                stopTolX=1.e-4/(10*(al+1)),
                initialStdDev=(0.025/(pow(100.,al))));
SSPToFEF(best1[],best2[],start);
```

Run:cmaes-VarIneq.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

$$\begin{array}{cl} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{5}$$

The term $X^n(x) \approx x - \tau u^n(x)$ will be computed by the interpolation operator or convect operator.
Or better we use an order 2 schema, BDF1

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u \approx \frac{(3u^{n+1} - 4u^n \circ X_1^n + u^{n-1} \circ X_2^n)}{2\tau}$$

with $u^* = 2u^n - u^{n-1}$, and $X_1^n(x) \approx x - \tau u^*(x), X_2^n(x) \approx x - 2\tau u^*(x)$
Run:NSCaraCyl-100-mpi.edp

Natural Convection

The coupling of natural convection modeled by the Boussinesq approximation and liquid to solid phase change in $\Omega =]0,1[^2$, No slip condition for the fluid are applied at the boundary and adiabatic condition on upper and lower boundary and given temperature θ_r (resp θ_l) at the right and left boundaries.
The model is : find the field : the velocity $\boldsymbol{u} = (u_1, u_2)$, the pressure p and temperature θ :

$$\left\{\begin{array}{rll} \partial_t \boldsymbol{u} + (\boldsymbol{u}.\nabla)\boldsymbol{u} + \nabla.\mu\nabla\boldsymbol{u} + \nabla p & = -C_T(\theta - \theta_0)\boldsymbol{e}_2 & \text{in } \Omega \\ \nabla.\boldsymbol{u} & = 0 & \text{in } \Omega \\ \partial_t\theta + (\boldsymbol{u}.\nabla)\theta + \nabla.k_T\nabla\theta & = 0 & \text{in } \Omega \end{array}\right. \tag{6}$$

Where $C_T(\theta - \theta_0)\boldsymbol{e}_2$ correspond to the Archimedes forces due to the affine dependence of density with the temperature and where θ_0 is temperature of the reference density.

Run:boussinesq2d.edp Run:boussinesq3d.edp Run:boussinesq3d-mpi.edp

Phase change with Natural Convection The starting point of the

problem is Brainstorming session (part I) of the third FreeFem++ days in december 2011, this is almost the Orange Problem is describe in web page `http://www.ljll.math.upmc.fr/~hecht/ftp/ff++days/2011/Orange-problem.pdf`.
The coupling of natural convection modeled by the Boussinesq approximation and liquid to solid phase change in $\Omega =]0,1[^2$, No slip condition for the fluid are applied at the boundary and adiabatic condition on upper and lower boundary and given temperature θ_r (resp θ_l) at the right and left boundaries.
The model is : find the field : the velocity $\boldsymbol{u} = (u_1, u_2)$, the pressure p and temperature θ :

$$\left\{\begin{array}{rll} \boldsymbol{u} & \text{given} & \text{in } \Omega_s \\ \partial_t \boldsymbol{u} + (\boldsymbol{u}\nabla)\boldsymbol{u} + \nabla.\mu\nabla\boldsymbol{u} + \nabla p & = -c_T \boldsymbol{e}_2 & \text{in } \Omega_f \\ \nabla.\boldsymbol{u} & = 0 & \text{in } \Omega_f \\ \partial_t \theta + (\boldsymbol{u}\nabla)\theta + \nabla.k_T\nabla\theta & = \partial_t S(T) & \text{in } \Omega \end{array}\right. \quad (7)$$

Where Ω_f is the fluid domain and the solid domain is $\Omega_s = \Omega \setminus \Omega_f$.

Phase change with Natural Convection

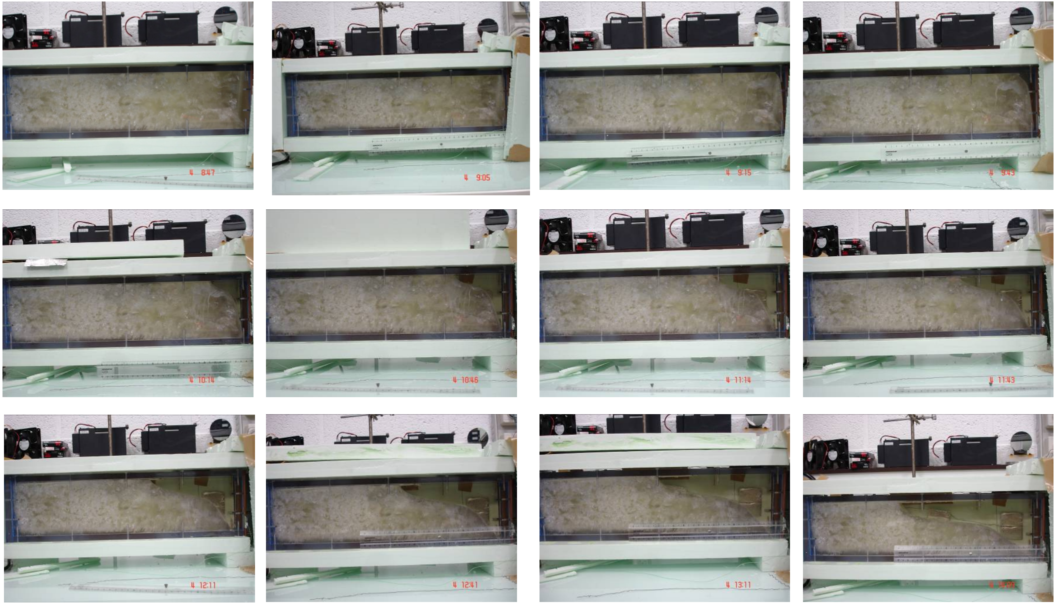
The enthalpy of the change of phase is given by the function S ; μ is the relative viscosity, k_T the thermal diffusivity.
In $\Omega_f = \{x \in \Omega; \theta > \theta_f\}$, with θ_m the melting temperature the solid has melt.
We modeled, the solid phase as a fluid with huge viscosity, so :

$$\mu = \left\{ \begin{array}{ccc} \theta < \theta_f & \sim & 10^6 \\ \theta \geq \theta_m & \sim & \frac{1}{\mathrm{Re}} \end{array} \right. ,$$

The Stefan enthalpy S_c with defined by $S_c(\theta) = H(\theta)/S_{th}$ where S_{the} is the stefan number, and H is the Heaviside function with use the following smooth the enthalpy :

$$S(\theta) = \frac{\tanh(50(\theta - \theta_m)))}{2S_{te}}.$$

The Physical Device



the Algorithm

We apply a fixed point algorithm for the phase change part (the domain Ω_f is fixed at each iteration) and a full no-linear Euler implicit scheme with a fixed domain for the rest. We use a Newton method to solve the non-linearity.

- if we don't make mesh adaptation, the Newton method do not converge
- if we use explicit method diverge too,
- if we implicit the dependance in Ω_s the method also diverge.

Implementation The finite element space to approximate $u1, u2, p, \theta$ is defined by

```
fespace Wh(Th,[P2,P2,P1,P1]);
```

We do mesh adaptation a each time step, with the following code :

```
Ph ph = S(T), pph=S(Tp);
Th= adaptmesh(Th,T,Tp,ph,pph,[u1,u2],err=errh,
              hmax=hmax,hmin=hmax/100,ratio = 1.2);
```

This mean, we adapt all variables, the 2 melting phase a time $n+1$ and n and smooth the metric with a ratio of 1.2 to account for the motion of the melting front.

The Newton loop

the fixed point are implemented as follows

```
real err=1e100,errp ;
 for(int kk=0;kk<2;++kk) // 2 step of fixe point on Ωs
 { nu = nuT; // recompute the viscosity in Ωs,Ωf
   for(int niter=0;niter<20; ++ niter) // newton loop
    { BoussinesqNL;
      err = u1w[].linfty;
      cout << niter << " err NL " << err <<endl;
      u1[] -= u1w[];
      if(err < tolNewton) break;  }// convergence ..
  }
```

The linearized problem

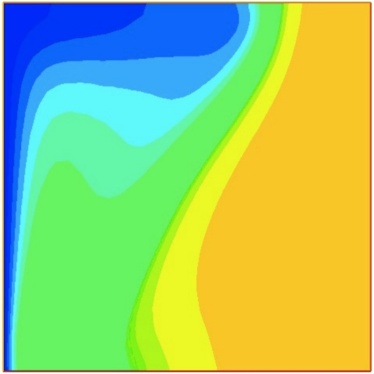
```
problem BoussinesqNL([u1w,u2w,pw,Tw],[v1,v2,q,TT])
= int2d(Th) (
     [u1w,u2w,Tw]'*[v1,v2,TT]*cdt
     + UgradV(u1,u2,u1w,u2w,Tw)' * [v1,v2,TT]
     + UgradV(u1w,u2w,u1,u2,T)' * [v1,v2,TT]
     + ( Grad(u1w,u2w)'*Grad(v1,v2)) * nu
     + ( Grad(u1,u2)'*Grad(v1,v2)) * dnu* Tw
     + cmT*Tw*v2  + grad(Tw)'*grad(TT)*kT
     - div(u1w,u2w)*q -div(v1,v2)*pw - eps*pw*q
     + dS(T)*Tw*TT*cdt )
   - int2d(Th)(
     [u1,u2,T]'*[v1,v2,TT]*cdt
     + UgradV(u1,u2,u1,u2,T)' * [v1,v2,TT]
     + ( Grad(u1,u2)'*Grad(v1,v2)) * nu
     + cmT*T*v2 - eps*p*q  +  grad(T)'*grad(TT)*kT
     - div(u1,u2)*q -div(v1,v2)*p
     +  S(T)*TT*cdt - [u1p,u2p,Tp]'*[v1,v2,TT]*cdt
     -  S(Tp)*cdt*TT)
  + on(1,2,3,4, u1w=0,u2w=0)+on(2,Tw=0)+on(4,Tw=0) ;
```

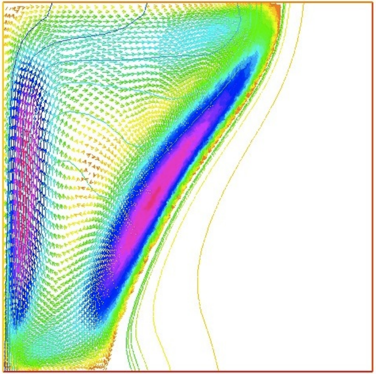
The parameters of the computation

take case 2 from
Shimin Wang, Amir Faghri, and Theodore L. Bergman. A comprehensive numerical model for melting with natural convection. *International Journal of Heat and Mass Transfer*, January 2010.

$\theta_m = 0$, $\mathrm{Re} = 1$, $S_{te} = 0.045$, $P_r = 56.2$, $R_a = 3.27\ 10^5$, $\theta_l = 1, \theta_r = -0.1$ so in this case $\mathrm{cmT} = c_T = -R_a/P_r$, $\mathrm{kT} = k_T = 1/P_r$, $\mathrm{eps} = 10^{-6}$, time step $\delta t = 10^{-1}$, $\mathrm{cdt} = 1/\delta t$, at time $t = 80$ and we get a good agreement with the article.

Phase change with Natural Convection





So now, a real problem, get the physical parameter of the real experiment.

Run:Orange-Newton.edp

Conclusion/Future

Freefem++ v3.20 is

- very good tool to solve non standard PDE in 2D/3D
- to try new domain decomposition domain algorithm

The the future we try to do :

- Build more graphic with VTK, paraview , ... (in progress)
- Add Finite volume facility for hyperbolic PDE (just begin C.F. FreeVol Projet)
- 3d anisotrope mesh adaptation
- automate the parallel tool

Thank for you attention.