An Introduction to FreeFem++

UPMC, December 2011

O. Pantz *

*CMAP, Ecole Polytechnique.

FreeFem++

What is it ?

FreeFEM++ is a Free software to solve PDE using the Finite Element Method.

It runs on Windows, Linux and Mac OS.

What does it do ?

- Mesh generation;
- Automatic Building of Mass/Rigid Matrices and second member;
- Diverse Linear Solvers integrated;
- Works for 2d or 3d problems;
- Generates Graphic/Text/File outputs;
- and a lot more ...

How to start ?

Download page

http://www.freefem.org/ff++

How does it work ?

• **Write your script** using a raw text editor – Notepad++ under Windows, plenty of choices under Linux(for iunstance kate);

• **Save it** with file extension .edp;

• **Run it** : Just clic on the file under Windows/Mac OS. Execute command FreeFem++ MyScript.edp under Linux.

Getting Help

Full documentation and many examples included in FeeFem++.

Subscribe to the mailing list.

Syntax

YOU KNOW what a Variational Formulation is ?

YOU KNOW some C++ ?

Then YOU KNOW FeeFem++ !

Outline...

First Example: Poisson Equation

- Mesh
- Finite Elements
- Save and Read

Second Example: Elasticity

- Make it simple: use Macros

Third Example: Stokes

- Matrices: build and solve

Forth Example: Navier Stokes

- Convection

Fifth Example: Coupled systems

- Boundary Elements

Sixth Example: Vector 2 Mesh (and Mesh 2 Vector)

- Usefull for free boundary problems.

First Example: POISSON.

Build a mesh, Define a problem, Solve and Save.

Let's Start !

Poisson Equation

Find $u : \Omega :=]0,1[^2 \to \mathbb{R}$ such that

$$\begin{cases} -\Delta u = f & \text{on } \Omega \\ u = 0 & \text{on } \partial\Omega. \end{cases}$$

Variational Formulation

Find $u \in H_0^1(\Omega)$ such that for all $v \in H_0^1(\Omega)$,

$$\int_\Omega \nabla u \cdot \nabla v \, dx - \int_\Omega f v \, dx = 0.$$

Galerkin Approximation – Using P1 Finite Elements

Find $u_h \in V_h$ such that for all $v_h \in V_h$,

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, dx - \int_\Omega f v_h \, dx = 0,$$

$$V_h = \{v_h \in H_0^1(\Omega) \ : \ v_h|_T \in \Pi_1, \text{ for any } T \in \mathcal{S}_h\},$$

where $\mathcal{S}_h$ is a regular mesh of Ω and $\Pi_1 :=$ space of polynomals whose degree is lower or equal to one.

Poisson.edp

```
mesh Sh= square(10,10);                    //   mesh generation of a square
fespace Vh(Sh,P1);                         //   space of P1 Finite Elements
Vh u,v;                                     //   u and v belongs to Vh
func f=cos(x)*y;                           //    f is a function of x and y
problem Poisson(u,v)=                       //    Definition of the problem
    int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v))            //    bilinear form
    -int2d(Sh)(f*v)                                 //    linear form
+on(1,2,3,4,u=0);                           //    Dirichlet Conditions
Poisson;                                  //    Solve Poisson Equation
plot(u);                                    //    Plot the result
```

Save and Read

Save.edp

```
include "Poisson.edp";                    //    include previous script
plot(u,ps="result.eps");                  //    Generate eps output
savemesh(Sh,"Sh.msh");                    //    Save the Mesh
ofstream file("potential.txt");
file<<u[];
```

Read.edp

```
mesh Sh=readmesh("Sh.msh");               //    Read the Mesh
fespace Vh(Sh,P1);
Vh u=0;
ifstream file("potential.txt");
file>>u[];                                //    Read the potential
plot(u,cmm="The result was correctly saved :)");
```

More mesh Generation

Keyword `square` only allows for the generation of structured meshes. More general meshes can be generated by using keywords `border` and `buildmesh`. It enables you to define mesh for open sets whose boundaries are described by parametrized curves.

Mesh.edp

```
border a(t=0,2.*pi){x=cos(t);y=sin(t);};
mesh Sh1=buildmesh(a(20));
plot(Sh1,wait=1,cmm="The unit disk");

border b(t=0,2.*pi){x=0.2*cos(t)+0.3;y=sin(t)*0.2+0.3;};
mesh Sh2=buildmesh(a(20)+b(-20));
plot(Sh2,wait=1,cmm="The unit disk with a hole");

mesh Sh3=buildmesh(a(20)+b(20));
plot(Sh3,wait=1,cmm="The unit disk with an inclusion");
```

Poisson2.edp

```
int Dirichlet=1;                                //    For label definition
border a(t=0,2.*pi){x=cos(t);y=sin(t);
                                                label=Dirichlet;};
border b(t=0,2.*pi){x=0.2*cos(t)+0.3;y=sin(t)*0.2+0.3;
                                                label=Dirichlet;};
mesh Sh=buildmesh(a(80)+b(-20));     //    The Unit Disk with a hole
fespace Vh(Sh,P1);
Vh u,v;
func f=cos(x)*y;
problem Poisson(u,v)=
    int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v))
    -int2d(Sh)(f*v)
+on(Dirichlet,u=0);                             //    u=0 label=Dirichlet=1
Poisson;
plot(u);
```

Limit Conditions

Find $u : \Omega :=]0,1[^2 \to \mathbb{R}$ such that

$$\begin{cases} -\Delta u = f & \text{on } \Omega, \\ \frac{\partial u}{\partial n} = g & \text{on } \Gamma_N, \\ u = u_D & \text{on } \partial\Gamma_D. \end{cases}$$

Variational Formulation

Find $u \in V := \{v \in H^1(\Omega) \ : \ v = u_d \text{ on } \Gamma_D\}$, such that for all $v \in V^D := \{v \in H^1(\Omega) \ : \ v = 0 \text{ on } \Gamma_D\}$,

$$\int_\Omega \nabla u \cdot \nabla v \, dx - \int_\Omega f v \, dx - \int_{\Gamma_N} g v \, ds = 0.$$

Galerkin Approximation – Using P1 Finite Elements

Find $u_h \in V_h$ such that for all $v_h \in V_h^D$,

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, dx - \int_\Omega f v_h - \int_{\Gamma_N} g v_h \, ds \, dx = 0,$$

$$V_h = \{v_h \in V \ : \ v_h|_T \in \Pi_1, \text{ for any } T \in \mathcal{S}_h\},$$

$$V_h^D = \{v_h \in V^D \ : \ v_h|_T \in \Pi_1, \text{ for any } T \in \mathcal{S}_h\}.$$

Poisson3.edp

```
int Dirichlet=1,Neumann=2;                    //    For label definition
border a(t=0,2.*pi){x=cos(t);y=sin(t);label=Dirichlet;};
border b(t=0,2.*pi){x=0.2*cos(t)+0.3;y=sin(t)*0.2+0.3;label=Neumann;};
mesh Sh=buildmesh(a(80)+b(-20));
fespace Vh(Sh,P1);
Vh u,v;
func f=cos(x)*y; func ud=x; func g=1.;
problem Poisson(u,v)=
    int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v))
    +int1d(Sh,Neumann)(g*v)
    -int2d(Sh)(f*v)
+on(Dirichlet,u=ud);                        //    u=ud on label=Dirichlet=1
Poisson;
plot(u);
```

Mesh adaptation

Mesh could be adapted with respect to a given finite element with the keyword `adaptmesh`.

Poisson4.edp

```
int Dirichlet=1,Neumann=2;
border a(t=0,2.*pi){x=cos(t);y=sin(t);label=Dirichlet;};
border b(t=0,2.*pi){x=0.2*cos(t)+0.3;y=sin(t)*0.2+0.3;label=Neumann;};
mesh Sh=buildmesh(a(80)+b(-20));
fespace Vh(Sh,P1); Vh u,v;
func f=cos(x)*y; func ud=x; func g=1.;
problem Poisson(u,v)=
    int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v))
    -int1d(Sh,Neumann)(g*v)-int2d(Sh)(f*v)
+on(Dirichlet,u=ud);
Poisson;plot(Sh,cmm="Initial Mesh",wait=1);plot(u,wait=1);
Sh=adaptmesh(Sh,u,err=1.e-3);
Poisson;plot(Sh,cmm="Adapted Mesh",wait=1);plot(u);
```

P1...

```
int Dirichlet=1,Neumann=2;
border a(t=0,2.*pi){x=cos(t);y=sin(t);label=Dirichlet;};
border b(t=0,2.*pi){x=0.2*cos(t)+0.3;y=sin(t)*0.2+0.3;label=Neumann;};
mesh Sh=buildmesh(a(10)+b(-5));
func f=cos(x)*y; func ud=x; func g=1.;

fespace Vh1(Sh,P1);Vh1 u1,v1;
problem Poisson1(u1,v1)=
    int2d(Sh)(dx(u1)*dx(v1)+dy(u1)*dy(v1))
    -int1d(Sh,Neumann)(g*v1)-int2d(Sh)(f*v1)
+on(Dirichlet,u1=ud);
                                                    //   ...
```

.. versus P2 ...

```
fespace Vh2(Sh,P2);Vh2 u2,v2;
problem Poisson2(u2,v2)=
     int2d(Sh)(dx(u2)*dx(v2)+dy(u2)*dy(v2))
     -int1d(Sh,Neumann)(g*v2)-int2d(Sh)(f*v2)
+on(Dirichlet,u2=ud);
                                                    //    ...
```

... versus P3

P3 finite elements are not avaiable by default. They have to be loaded first (in fact, you can even define your own elements).

```
load "Element_P3";                          //    load P3 finite elements
fespace Vh3(Sh,P3);Vh3 u3,v3;
problem Poisson3(u3,v3)=
int2d(Sh)(dx(u3)*dx(v3)+dy(u3)*dy(v3))
-int1d(Sh,Neumann)(g*v3)-int2d(Sh)(f*v3)
+on(Dirichlet,u3=ud);
                                                          //   ...
```

Solve it all and plot !

Poisson5.edp

```
Poisson1;Poisson2;Poisson3;
plot(u1,cmm="with P1 finite elements",wait=1);
plot(u2,cmm="with P2 finite elements",wait=1);
plot(u3,cmm="with P3 finite elements");
```

WARNING

VERY BAD SCRIPT !!!

As we will see there is smarter ways to do this by using `macro` or `varf`.

What have I learned

About Meshes

- `mesh` = Type for... meshes;
- `square` = To create structured meshes;
- `savemesh` = To save a mesh in a file;
- `readmesh` = To read mesh previously saved;
- `border` = Type for parametrized curves;
- `buildmesh` = Build a mesh with parametrized boundaries;
- `adaptmesh` = To refine a mesh.

About Variational Formulations

- `fespace` = Type for Finite Element Spaces;
- `problem` =Type for a Variational Problem;
- `P1,P2,P3` = Lagrange Elements;
- `int2d` = Integration over a domain;
- `int1d` = Integration over part of the boundary;
- `on` = To impose Boundary Conditions.

What have I learned

Misc...

- `func` = Type for a function of x and y;
- `int` = Type for integers;
- `plot` = Just guess;
- `include` = To include a script into another;
- `ofstream` = To direct the output to a file;
- `ifstream` = To get the input from a file.
- `cos` = All classical maths functions could be used;

Second Example: ELASTICITY.

Macros are beautiful.

More tricky: Linear Elasticity

The Variationnal Formulation

Find

$$u \in X := \{v \in H^1(\Omega)^2 \ : \ v = 0 \text{ on } \Gamma_D\},$$

such that for all $v \in X$,

$$\int_\Omega Ae(u) \cdot e(v)\, dx - \int_{\Gamma_N} v \cdot g\, ds = 0,$$

with

$$e(u) = (\nabla u + \nabla u^T)/2,$$

where for any symetric matrix ξ,

$$A\xi \cdot \xi = 2\mu|\xi|^2 + \lambda(\mathrm{Tr}(\xi))^2.$$

If you are a barbarian

elasticity1.edp

```
real L=10,H=1,dn=5;
mesh Sh=square(L*dn,H*dn,[x*L,y*H]);  //    square OK of rectangles
real mu=1,lambda=1;
fespace Vh(Sh,P1);
Vh u1,u2,v1,v2;                        //    One for each component
func g1=0;func g2=-1.;
problem elasticity(u1,u2,v1,v2)=
int2d(Sh)(2.*mu*
(dx(u1)*dx(v1)+dy(u2)*dy(v2)+(dy(u1)+dx(u2))*(dy(v1)+dx(v2))/2.)
+ lambda*(dx(u1)+dy(u2))*(dx(v1)+dy(v2)))
-int1d(Sh,2)(g1*v1+g2*v2)                          //    ARGH ....
+on(4,u1=0,u2=0);
plot([u1,u2],wait=1);                  //    you can plot vectors :)
```

If you are civilized: Use Macros

elasticity2.edp

```
real L=10,H=1,dn=5;
mesh Sh=square(L*dn,H*dn,[x*L,y*H]);
real mu=1,lambda=1;
fespace Vh(Sh,[P1,P1]);                          //    vectorial FE space
macro u [u1,u2]                                  //    end of macro
macro v [v1,v2]                                  //    end of macro
macro e(u) [dx(u[0]),(dx(u[1])+dy(u[0]))/ sqrt(2),dy(u[1])]   //
macro A [[2.*mu+lambda,0,lambda],[0,2.*mu,0],[lambda,0,2.*mu+lambda]]
//
macro g [0,-1.]                                                //
Vh u,v;                                      //    One for each component
problem elasticity(u,v)=
          int2d(Sh)((A*e(u))'*e(v))-int1d(Sh,2)(g'*v)
+on(4,u1=0,u2=0);
elasticity;plot(u,wait=1);
```

Don't like the output ?
Try This: elasticity3.edp

```
include "elasticity2.edp";
fespace Vh0(Sh,P0);
Vh0 sigma=(A*e(u))'*e(u);
plot(sigma,fill=1,cmm="Constraints",wait=1);

real delta=1.e-3;
mesh Th=movemesh(Sh,[x+delta*u1,y+delta*u2]);
plot(Th,cmm="Deformed configuration",wait=1);

fespace Wh0(Th,P0);
Wh0 tau=0;
tau[]=sigma[];

plot(tau,fill=1,cmm="Constraints on deformed configuration");
```

Time, it is Time

Wave equation

Looking for $u \in \mathsf{C}^1([0,T], L^2(\Omega)^2) \cap \mathsf{C}^0([0,T], X)$ such that for all $v \in X := \{v \in H^1(\Omega) \ : \ v = 0 \text{ on } \Gamma_D\}$,

$$\int_\Omega \frac{\partial^2 u}{\partial t^2} v \, dx + \int_\Omega Ae(u) \cdot e(v) \, dx = 0,$$

$$u(t=0) = u_0,$$

and

$$\frac{\partial u}{\partial t}(t=0) = v_0.$$

Time Discretization

$$\int_\Omega \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} v \, dx + \int_\Omega Ae(u^{n+1}) \cdot e(v) \, dx = 0.$$

Parameters and Macros

```
real T=5000.,dt=5.;                          //    Finale Time and time step
macro v0 [0,-1.]                                  //    Intial velocity

mesh Sh;                                        //    Building the mesh
real L=10,H=1,dn=5;Sh=square(L*dn,H*dn,[x*L,y*H]);

macro u [u1,u2]                                          //    state
macro up [up1,up2]                                  //    previous state
macro upp [upp1,upp2]                          //    previous previous state
macro v [v1,v2]                                        //    end of macro
macro e(u) [dx(u[0]),(dx(u[1])+dy(u[0]))/ sqrt(2),dy(u[1])]   //
macro A [[2.*mu+lambda,0,lambda],[0,2.*mu,0],[lambda,0,2.*mu+lambda]]
//   ...
```

Definition of the problem

```
fespace Vh(Sh,[P1,P1]);                    //    vectorial FE space
Vh u,up,upp,v;
real mu=1,lambda=1;
real t;
problem OneStepElasticity(u,v,init=t)=
          int2d(Sh)((u'*v)/(dt^2)+(A*e(u))'*e(v))
         +int2d(Sh)((upp'*v-2.*(up'*v))/(dt^2))
+on(4,u1=0,u2=0);

upp=[0.,0.];                               //    Initialization of states
up=upp+[dt*v0[0],dt*v0[1]];
```

Loops look like C++

Evolution.edp

```
real delta=0.05;                    //    exageration of the displacement

for(t=0;t<T;t+=dt){
    mesh Th=movemesh(Sh,[x+delta*u[0],y+delta*u[1]]);
    plot(Th,bb=[[0,-3.*H],[L*1.3,4.*H]]);
    OneStepElasticity;
    upp=up;up=u;
}
```

What have I learned

- `macro` = To define a macro, ends with a commentary;
- `for` = To make a loop;
- [`P1`,`P1`]= Product of FE spaces;
- `P0` = P0 Finite Elements;
- `movemesh` = To move a mesh;
- `init` = Build the matrix or Not ?;
- `real` = Type for reals;
- `fill` = Fill between isolines;
- `cmm` = Comments for plot;
- `wait` = Wait before next plot;
- `u[]` = If `u` is a Finite Element, return the values of `u` at each degree of freedom.

Third Example: STOKES.

Be Free: Use Matrices and Vectors.

Stokes

Let $V = H^1(\Omega)^2$, $V_0 = H_0^1(\Omega)^2$, and $R = L^2(\Omega)$. Stokes problem consists in finding $(u, p, c) \in V \times R \times \mathbb{R}$, such that

$$a(u, v) + b(p, v) = 0 \text{ for all } v \in V,$$

$$b(q, u) + cm(q) = 0, \text{ for all } q \in R,$$

$$m(p) = 0,$$

and $u = u_d$ on $\partial\Omega$, where

$$a(u, v) = \int_\Omega \mu e(u) \cdot e(v)\, dx,$$

$$b(p, u) = \int_\Omega \mathrm{div}(u) p\, dx,$$

$$m(p) = \int_\Omega p\, dx.$$

Let's rock...

```
real u1d=1.;                              //   velocity of the upper face

mesh Sh;                                  //    Definition of the mesh
{ int n=30; Sh=square(n,n); }

fespace Vh(Sh,[P1b,P1b]);                 //    Definition of the FE spaces
fespace Rh(Sh,P1);


macro u [u1,u2]                                        //    Some macros
macro v [v1,v2]                                                 //
macro e(u) [dx(u[0]),(dx(u[1])+dy(u[0]))/sqrt(2.),dy(u[1])]    //
macro div(u) (dx(u[0])+dy(u[1]))                              //   ...
```

```
real mu=1.;                                                    //    Viscosity
                          //    Definition of the linear/bilinear forms
varf a(u,v)=int2d(Sh)(mu*(e(u)'*e(v)))+on(1,2,3,4,u1=0,u2=0);
varf b([p],u)=int2d(Sh)(div(u)*p);
varf m(p,q)=int2d(Sh)(q);
varf bc(u,v)=on(3,u1=u1d,u2=0);

matrix M;                                          //    Building the System
real[int] L(Vh.ndof+Rh.ndof+1);
{
matrix A=a(Vh,Vh);
matrix B=b(Rh,Vh);
real[int] C=m(0,Rh);
M=[[A ,B,0],
   [B',0,C],
   [0,C',0]];
L=0;
L(0:Vh.ndof-1)=bc(0,Vh);}                                          //    ...
```

stokes.edp

```
Vh u; Rh p;                                    //    Solving the system
{
set(M,solver=UMFPACK);
real[int] sol=M^-1*L;
u=[0,0];p=0;
u1[]=sol(0:Vh.ndof-1);
p[]=sol(Vh.ndof:Vh.ndof+Rh.ndof-1);
}
plot(u,p);                                     //    Display the result
```

What have I learned

- `matrix` = Sparse Matrix;
- `real[int]` = array of reals;
- `M'` = Transposed Matrix;
- `M=[[A,B],[C,D]]` = Matrix defined by bloc;
- `M=a(Vh,Wh)` = Build Matrix Associated to the Bilinear part of `a`;
- `L=a(0,Wh)` = Build vector Associated to the Linear part of `a`;
- `varf` = bilinear form+linear form;
- `set(·,solver=·)` = Define the solver associate to a matrix;
- `s=M^-1*L` = Solve `Ms=L`.

Fourth Example: Navier Stokes.

Recipe for NS: Take Stokes, Add Convection.

Navier Stokes

Let's try to solve the NS equation

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \mu \nabla \cdot e(u) + \nabla p = 0 \text{ in } \Omega,$$

$$\nabla \cdot u = 0 \text{ in } \Omega,$$

$$u = u_D \text{ on } \partial\Omega$$

Time Discretization

$$\frac{u^{n+1} - u^n \circ X^n(-dt)}{dt} - \mu \nabla \cdot e(u^n) + \nabla p^n = 0 \text{ in } \Omega,$$

$$\nabla \cdot u^n = 0 \text{ in } \Omega,$$

$$u^n = u_D \text{ on } \partial\Omega,$$

with

$$\dot{X}^n = u^n \circ X^n \text{ and } X^n(t = 0) = \text{Id}\,.$$

Variational Fromulation

Using the same notation than for Stokes, N.S. problem consists in finding for each n, $(u^{n+1}, p^{n+1}, c^{n+1}) \in V \times R \times \mathbb{R}$, such that

$$r(\frac{u^{n+1} - u_c^n}{dt}, v) + a(u^{n+1}, v) + b(p^{n+1}, v) = 0 \text{ for all } v \in V,$$

with

$$u_c^n = u^n \circ X^n(-dt),$$

and

$$r(u, v) = \int_\Omega uv \, dx.$$

Parameters/mesh and macros

```
real T=0.1,dt=0.005,u1d=1.;
real mu=1.,rho=1.;
mesh Sh;
{ int n=30; Sh=square(n,n); }
fespace Vh(Sh,[P1b,P1b]);
fespace Rh(Sh,P1);
macro u [u1,u2]                                                    //
macro up [up1,up2]                                                 //
macro v [v1,v2]                                                    //
macro e(u) [dx(u[0]),(dx(u[1])+dy(u[0]))/sqrt(2.),dy(u[1])]         //
macro div(u) (dx(u[0])+dy(u[1]))                                   //
macro Convect(u,dt,v) [convect(u,dt,v[0]),convect(u,dt,v[1])]      //
Vector convection...
```

Linear and Bilinear forms

```
Vh u,up;Rh p;
        //    definition of the linear/bilinear forms (same than for
Stokes)
varf a(u,v)=int2d(Sh)(mu*(e(u)'*e(v)))+on(1,2,3,4,u1=0,u2=0);
varf b([p],u)=int2d(Sh)(div(u)*p);
varf m(p,q)=int2d(Sh)(q);
varf bc(u,v)=on(3,u1=u1d,u2=0);

varf r(u,v)=int2d(Sh)((rho/dt)*(u'*v));       //    For the inertial
terms
varf conv(u,v)=int2d(Sh)((rho/dt)*(Convect(up,(-dt),up)'*v));     //
...
```

Building the System

```
matrix M;
real[int] L(Vh.ndof+Rh.ndof+1);L=0;
real[int] L0=bc(0,Vh);
{
matrix A0=a(Vh,Vh);
matrix A1=r(Vh,Vh);
matrix A=A0+A1;
matrix B=b(Rh,Vh);
real[int] C=m(0,Rh);
M=[[A,B,0],
   [B',0,C],
   [0,C',0]];
}
set(M,solver=UMFPACK);                                    //   ...
```

NS.edp

```
real[int] viso=[0.,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.];      //
For the output
up=[0.,0.];u=[0,0];p=0;
for(real t=0;t<T;t+=dt){
        real[int] L1=conv(0,Vh);
        L(0:Vh.ndof-1)=L0+L1;
        real[int] sol=M^-1*L;                  //    Solving the system
        u1[]=sol(0:Vh.ndof-1);
        p[]=sol(Vh.ndof:Vh.ndof+Rh.ndof-1);
        plot(u,viso=viso);                     //    Display the result
        up=u;
}
```

What have I learned

- `convect` = Compute Characteristics;
- `viso` = Option in `plot` to set the values of the isolines displayed.

Fifth Example: COUPLED SYSTEMS.

"Boundary" Elements

Couples

Let's now try to couple two systems... Could be wathever you want, but let's consider two Poisson equations. For $i = 1, 2$,

$$-k_i \Delta u_i = f_i \text{ in } \Omega_i$$

$$u_1 = u_2 \text{ on } \Gamma = \partial\Omega_1 \cap \partial\Omega_2$$

$$u_i = 0 \text{ on } \Gamma_i = \partial\Omega_i \setminus \Gamma.$$

Variational Formulation

Find

$$u_i \in V_i := \{v \in H^1(\Omega_i) \ : \ u_1 = 0 \text{ on } \Gamma_i\},$$

and $p \in H^{-1/2}(\Gamma)$ such that for all $(v_1, v_2, q) \in V_1 \times V_2 \times H^{-1/2}(\Gamma)$,

$$a_1(u_1, v_1) + b(p, v_1) = L_1(v_1) \text{ and } a_2(u_2, v_2) - b(p, v_2) = L_2(v_2),$$

$$b(q, u_1 - u_2) = 0,$$

where

$$a_i(u_i, v_i) = \int_{\Omega_i} (k_i \nabla u_i \cdot \nabla v_i),$$

and

$$b(q, u) = \int_\Gamma qu \, ds.$$

Parameters, mesh and macro

```
int n1=10,n2=25;
real k1=1,k2=2;
real f1=1,f2=1;

mesh Sh1=square(n1,n1);
mesh Sh2=square(n2,n2,[x+1.,y]);
mesh Sh=emptymesh(Sh1);        //    mesh for the Lagrange Multiplier

macro grad(u) [dx(u),dy(u)]                                    //
```

Linear,Bilinear Forms,Spaces

```
varf a1(u,v)=int2d(Sh1)(k1*(grad(u)'*grad(v)))+on(1,3,4,u=0);
varf a2(u,v)=int2d(Sh2)(k2*(grad(u)'*grad(v)))+on(1,2,3,u=0);
varf b1(q,u)=int1d(Sh,2)(q*u);
varf b2(q,u)=int1d(Sh,2)(-q*u);
varf pen(p,q)=int1d(Sh,1,3,4)(-p*q);      //    penalization of the
unused terms

varf L1(u,v)=int2d(Sh1)(f1*v);
varf L2(u,v)=int2d(Sh2)(f2*v);

fespace Vh1(Sh1,P2),Vh2(Sh2,P1),Rh(Sh,P1);
```

Build Sytem

```
matrix M;{
     matrix A1=a1(Vh1,Vh1);
     matrix A2=a2(Vh2,Vh2);
     matrix B1=b1(Rh,Vh1);
     matrix B2=b2(Rh,Vh2);        //    Cool:  Automatic interpolation
     matrix P=pen(Rh,Rh);
     M=[[A1 ,0  ,B1],
        [0 ,A2  ,B2],
        [B1',B2',P ]];}
real[int] L(Vh1.ndof+Vh2.ndof+Rh.ndof);{
     real[int] L10=a1(0,Vh1);real[int] L11=L1(0,Vh1);
     real[int] L1=L10+L11;
     real[int] L20=a2(0,Vh2);real[int] L21=L2(0,Vh2);
     real[int] L2=L20+L21;
     L=0;L(0:Vh1.ndof-1)=L1;L(Vh1.ndof:Vh1.ndof+Vh2.ndof-1)=L2;}
                                                          //    ...
```

Coupled.edp

```
set(M,solver=UMFPACK);
real[int] sol=M^-1*L;
Vh1 u1=0; Vh2 u2=0;
u1[]=sol(0:Vh1.ndof-1);
u2[]=sol(Vh1.ndof:Vh1.ndof+Vh2.ndof-1);

plot(Sh1,Sh2,wait=1);
plot(u1,u2);
```

What have I learned

- `emptymesh` = To create a mesh with all nodes on the boundary;
- Interpolation between different mesh works !

Sixth Example: Vector 2 Mesh

Usefull for free boundary problems

vector2mesh.edp

```
int nb=10;                          //    nb of elements of the boundary
real[int] Cx(nb+1),Cy(nb+1);
for(int i=0;i<=nb;i++){
   real t=(1.*i)/nb;
   Cx(i)=cos(2.*pi*t);Cy(i)=sin(2.*pi*t);
}

plot([Cx,Cy],wait=1);               //    The boundary defined as a vector

                                    //    Build the mesh of border [Cx,Cy]
border a(t=0,nb){x=Cx(t);y=Cy(t);label=1;};
mesh Th=buildmesh(a(nb),fixeborder=1);
plot(Th,wait=1);
```

Border and CoBorder operators

```
int nb=50;
border a(t=0,2.*pi){x=cos(t);y=sin(t);label=1;};
mesh Th=buildmesh(a(nb));
cout<<"Number of elements of the boundary="<<Th.nbe<<endl;

macro b(nb,k) Th.be(nb)[k]                          //    border operator
int[int,int] cb(Th.nv,2);                          //    coborder operator
   int[int] nb(Th.nv); nb=0;
   for(int i=0;i<Th.nbe;i++){
      cb(b(i,0),nb(b(i,0))++)=i;
      cb(b(i,1),nb(b(i,1))++)=i;}
```

mesh2vector.edp

```
            //    Retreive the sorted list of points of the boundary
int[int] boundaryDOF(Th.nbe+1);
boundaryDOF(0)=b(0,0);boundaryDOF(1)=b(0,1);        //    first edge
for(int i=1;i<Th.nbe;i++){
   int j=boundaryDOF(i);
   int k0=(b(cb(j,0),0)==j)?(b(cb(j,0),1)):(b(cb(j,0),0));
   int k1=(b(cb(j,1),0)==j)?(b(cb(j,1),1)):(b(cb(j,1),0));
   boundaryDOF(i+1)=(k0==boundaryDOF(i-1))?k1:k0;}}

       //    Define the coordinates of the sorted boundary vertices
real[int] Cx(Th.nbe+1),Cy(Th.nbe+1);
   for(int i=0;i<Th.nbe+1;i++){
   Cx[i]=Th(boundaryDOF(i)).x;Cy[i]=Th(boundaryDOF(i)).y;}

                              //    plot the "vectorialized" boundary
   plot([Cx,Cy],wait=1);
```

Application: Moving meshes

How to movemesh under FreeFem++.

The basic problem

Let Ω be the mesh of boundary

$$\partial\Omega = \gamma.$$

Let $F : \gamma \to \mathbb{R}^2$. How to move definie move Ω according to F, that is construct Ω_F

$$\partial\Omega_F = F(\gamma).$$

FreeFem++ classical solution

Extend γ as an application from Ω into $\mathbb{R}^2$.

Standard Way

2bigMoveMesh.edp

```
int nb=50;                                        //    Definition of the mesh
border a(t=0,2.*pi){x=cos(t);y=sin(t);};
mesh Th=buildmesh(a(nb));
real exa;
                                                  //    Definition of the map F
func real F1(real x1,real x2){return x1+exa*0.2*(cos(4.*x2));}
func real F2(real x1real x2){return x2+exa*0.1*(sin(4.*x1));}
fespace Vh(Th,P1);Vh u1,u2;
real exaMax=2.,dexa=0.1;                          //    Moving the mesh
for(exa=0;exa<exaMax;exa+=dexa){
u1=F1(x,y);u2=F2(x,y);
mesh Sh=movemesh(Th,[u1,u2]);                     //    Failed for BIG exa
plot(Sh,wait=1,cmm="exa="+exa); }
```

mesh2vector → buildmesh → vector2mesh

Initalisation

```
                                        //    Definition of the mesh
                    //    and parametrization [Cx,Cy] of the boundary
include "mesh2vector.edp"
real exa;
func real F1(real x1,real x2){return x1+exa*0.2*(cos(4.*x2));}
func real F2(real x1,real x2){return x2+exa*0.1*(sin(4.*x1));}
```

movemesh.edp

Same as vector2mesh

```
real[int] Dx(Cx.n),Dy(Cy.n);
border A(t=0,Th.nbe){x=Dx(t);y=Dy(t);};
real exaMax=2.,dexa=0.1;
for(exa=0;exa<exaMax;exa+=dexa){
for(int i=0;i<Th.nbe+1;i++){
Dx(i)=F1(Cx(i),Cy(i));Dy(i)=F2(Cx(i),Cy(i));}
                                                    //    vector2mesh
mesh Sh=buildmesh(A(orientation*Th.nbe),fixeborder=1);
plot(Sh,wait=1,cmm="exa="+exa); }
```

What have I learned

- `movemesh` to move a mesh
- Sorry, 1D meshes not implemented...

THANK YOU FOR YOUR ATTENTION.