Mathematical methods for engineering.

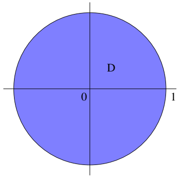
FreeFem: an example of finite element software

Doctoral School in Environmental Engineering - January 26, 2012

FreeFem

- Software to solve numerically partial differential equations based on FEM.
- Basically it is a compiler. We must write a program in which
  - create the mesh,
  - choose the kind of finite elements,
  - descrive the problem to solve,
  - solve the problem,
  - visualize the result.
- It is written in C++.
- It is free.

First Example



$$\begin{cases} -\Delta u = 1 & \text{in } D \\ u = 0 & \text{on } \partial D \end{cases}$$

$$u = (1 - x^2 - y^2)/4$$

```
border a(t=0,2*pi){x=cos(t);y=sin(t);label=1;};
mesh disk = buildmesh(a(30));
plot(disk,wait=true);

fespace femp1(disk,P1);
femp1 u,v;

solve laplace(u,v,solver=CG) =
   int2d(disk) (dx(u)*dx(v) +dy(u)*dy(v))
 + int2d(disk) (-1*v)
 + on(1,u=0);

plot(disk,u);
```

Creating the mesh

```
border a(t=0,2*pi){x=cos(t);y=sin(t);label=1;};
mesh disk =buildmesh(a(30));
plot(disk,wait=true);
```

All the boundary has the same label. $\rightsquigarrow$ We have only one kind of boundary condition.

- It is possible to save the mesh in a file
  `savemesh(disk,"disk.msh");`
- The file `disk.msh` contains
  - A first line with
    - the number of nodes `nnod`,
    - the number of elements `nelem`,
    - the number of edges on the boundary.
  - nnod lines with the coordinates and `label` (0 for the internal nodes).
  - nelem lines with the three nodes of each element and a label (for subdomains).

Choosing the finite elements

```
fespace femp1(disk,P1);
femp1 u,v;
```

P0	piecewise constant (discontinuous);
P1	piecewise lienar continuous;
P2	piecewise P_2 continuous;
RT0	Raviart-Thomas finite elements;
P1nc	piecewise linear continuous only in the middle point of each edge;

$\vdots$

Descriving and solving the problem

```
solve laplace(u,v) =
   int2d(disk) (dx(u)*dx(v) +dy(u)*dy(v))
  + int2d(disk) (-1*v)
  + on(1,u=0);
```

- The problem must be given in variational form.
- Notice the Dirichlet boundary condition.

Another way:

```
problem laplace(u,v) =
   int2d(disk) (dx(u)*dx(v) + dy(u)*dy(v))
  + int2d(disk) (-1*v)
  + on(1,u=0);
laplace;
```

- The first part descrive the problem laplace.
- Then the command laplace solve the problem.

Postprocesing

- Finally we plot the result:
  `plot(disk,u);`
- If we know the exact solution it is possible to evaluate the error

```
real errL2;
func ex=(1-x^2-y^2)/4;
errL2=sqrt(int2d(disk)((u-ex)^2));
cout << "errL2 " << "" << errL2 << endl;
```

Example - 1

```
border a(t=0,2){x=t;y=0;label=1;};
border b(t=0,1){x=2;y=t;label=2;};
border c(t=0,2){x=2-t;y=1;label=3;};
border d(t=0,1){x=0;y=1-t;label=4;};
int n,m,i;
n=5;
m=4;
real[int] errL2(m), errH1(m);
func ex=y*x^2;
func dxex=2*y*x;
func dyex=x^2;
func alpha = x+1;
func beta=(x^2-4*x-2)*y;
```

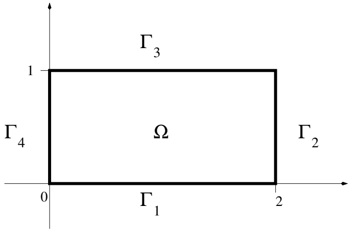
Example - 2

```
for (i=0; i<m;i++)
{
mesh rh = buildmesh(a(2^(i+1)*n)+b(2^i*n)+c(2^(i+1)*n)+d(2^i*n));
fespace femp2(rh,P2);
femp2 u,v;
problem myeq(u,v) =
   int2d(rh) (alpha*(dx(u)*dx(v) +dy(u)*dy(v)))
 + int2d(rh) (u*v)
 + int2d(rh) (-beta*v)
 + on(1,u=0) + on(2,u=4*y) +on(3,u=x^2);
myeq;
plot(rh,wait=true);
errL2[i]=sqrt(int2d(rh)((u-ex)^2));
errH1[i]=sqrt(int2d(rh)((dx(u)-dxex)^2+(dy(u)-dyex)^2));
plot(u,wait=true);
}
```

Example - 3

```
for (i=0; i<m;i++)
{
cout << " " << endl;
cout << "n=" << n*2^i << endl;
cout << "errL2" << " " << errL2[i] << endl;
cout << "errH1" << " " << errH1[i] << endl;
cout << " " << endl;
}
```

Example - 4



$$
\begin{array}{ll}
-\mathrm{div}\,[(x+1)\nabla u] + u = (x^2-4x-2)y & \text{in } \Omega \\
u = 0 & \text{on } \Gamma_1 \\
u = 4y & \text{on } \Gamma_2 \\
u = x^2 & \text{on } \Gamma_3 \\
(x+1)\nabla u \cdot \mathbf{n} = 0 & \text{on } \Gamma_4
\end{array}
$$

$$u(x,y) = x^2 y$$

Another example - 1

```
real theta=4.*pi/3.;
real a=1.,b=1.;
func f=-4*(cos(x^2+y^2-1)-(x^2+y^2)*sin(x^2+y^2-1));
func uex=sin(x^2+y^2-1);
real[int] errL2(2);

border Gamma1(t=0,theta){x=a*cos(t); y=b*sin(t);}
border Gamma2(t=theta,2*pi){x=a*cos(t); y=b*sin(t);}
```

Another example - 2

```
for (int n=0;n<2;n++)
{
mesh Th=buildmesh(Gamma1(20*(n+1))+Gamma2(10*(n+1)));

fespace Vh(Th,P2);
Vh u,v;
solve laplace(u,v)=int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
      -int2d(Th)(f*v) - int1d(Th,Gamma2)(2*v)+on(Gamma1,u=0);

plot(Th,u,wait=true);

errL2[n]=sqrt(int2d(Th)((u-uex)^2));
}
for(int n=0;n<2;n++)
cout << "errL2 " << n << " = " << errL2[n] << endl;
cout << "convergence rate = " << log(errL2[0]/errL2[1])/log(2.) << endl
```

An elasticity problem - 1

Consider the following elasticity problem:

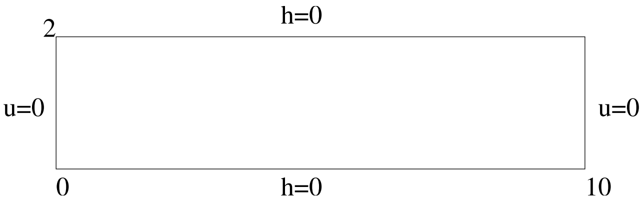
$$\begin{aligned}
&-\sum_{j=1}^{2} \frac{\partial}{\partial x_j}\sigma_{i,j}(\mathbf{u}) = g_i \quad i = 1,2 && \text{in } \Omega = [0,10] \times [0,2] \\
&\mathbf{u} = \mathbf{0} && \text{on } \Gamma_D \\
&\sum_{j=1}^{2} \sigma_{i,j}(\mathbf{u}) n_j = h_j && \text{on } \Gamma_N
\end{aligned}$$

Deformation tensor: $\epsilon_{i,j}(\mathbf{v}) = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)$

Stress tensor: $\sigma_{i,j}(\mathbf{v}) = \lambda \delta_{i,j} \text{div } \mathbf{v} + 2\mu \epsilon_{i,j}(\mathbf{v})$.

An elasticity problem - 2

The boundary stress is zero on lower and upper side and the two vertical sides of the beam are locked.



The associated bilinear form is:

$$a(\mathbf{u},\mathbf{v}) = 2\mu \sum_{i,j=1}^{2} \int_\Omega \epsilon_{i,j}(\mathbf{u})\epsilon_{i,j}(\mathbf{v})\, dx + \lambda \int_\Omega \text{div }\mathbf{u}\, \text{div }\mathbf{v}\, dx\,.$$

An elasticity problem - 3

```
border a(t=2,0)  { x=0; y=t ;label=1;};        //  left beam
border b(t=0,10) { x=t; y=0 ;label=2;};        //  bottom of beam
border c(t=0,2)  { x=10; y=t ;label=1;};       //  rigth beam
border d(t=0,10) { x=10-t; y=2; label=3;};     //  top beam
real E = 21.5, s = 0.29, g = -0.1;
real mu = E/(2*(1+s)), lambda = E*s/((1+s)*(1-2*s));
mesh th = buildmesh( b(40)+c(10)+d(40)+a(10));
plot(th,wait=1);
fespace Vh(th,[P1,P1]);
Vh [u1,u2], [v1,v2];

solve bb([u1,u2],[v1,v2],solver=Crout)=
int2d(th)(2*mu*(dx(u1)*dx(v1)+dy(u2)*dy(v2)
               + ((dx(u2)+dy(u1))*(dx(v2)+dy(v1)))/2 )
               + lambda*(dx(u1)+dy(u2))*(dx(v1)+dy(v2)))
      - int2d(th) (g*v2)
      + on(1,u1=0,u2=0);

plot([u1,u2],wait=1);
mesh th1 = movemesh(th, [x+u1, y+u2]);
plot(th1,wait=1);
```