Initiation à FreeFem++

ALLA Abdellah *

Module: Analyse Numérique des équations aux dérivées partielles (ANEDP)

Option: Mathématiques d'Aide la Décision

Master: Mathématique et Applications, Semestre 3.

Département de Mathématiques

Université Mohammed V, Faculté des Sciences de Rabat.

2016-2017.

*UM5, Faculté des Sciences - Rabat, Maroc.
Email: a.alla@um5s.net.ma, abdellah.alla@gmail.com

FreeFem++

Développé au Laboratoire J-L Lions, Université Pierre et Marie Curie, Paris par F. Hecht en collaboration avec O. Pironneau, J. Morice, A. Le Hyaric et K. Ohtsuka.

Qu'est ce que c'est?

FreeFEM++ Est un solveur permettant de résoudre les équations aux dérivées partielles (EDP) en 2d et 3d.

- Il s'exécute en ligne de commande.
- Il permet de creer des maillages
- Résoudre des équations aux dérivées partielles par des méthodes de type éléments finis.

Méthode employé : Méthode d'éléments finis (Finite Element Method.):

- EF : P0, P1, P2,
- Raviart-Thomas, P1 diconstinue,

Comment utiliser FreeFEM++?

Télécharger

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

Documentation

FreeFEM++ Documentation

Comment ça marche?

- **Editer un fichier** nomfichier avec emacs, vi, etc...
- **Enregistrer** le fichier sous l'extension `.edp` nomfichier.edp;
- **Exécuter**:

FreeFem++ nomfichier.edp

sous Linux, ou Utiliser l'interface graphique de FreeFem++ :

FreeFem++-cs

sous Windows/Mac OSpuis cliquer sur Run.

Syntaxe

Savoir c'est quoi une Formulation Variationalle ?

Savoir des notions sur C++ ?

Manipulation FeeFem++ !

Outline 1

Les types de données

- Variables globales
- Les types basiques
- Les tableaux et les matrices

Les types de données: Variables globales

- **x**, **y** et **z**: les coordonnées du point courant.
- **dx**=$\frac{\partial}{\partial x}$, **dy**=$\frac{\partial}{\partial y}$, **dz**=$\frac{\partial}{\partial z}$, **dxy**=$\frac{\partial^2}{\partial x \partial y}$, **dxz**=$\frac{\partial^2}{\partial x \partial z}$, **dyz**=$\frac{\partial^2}{\partial y \partial z}$, **dxx**=$\frac{\partial^2}{\partial x^2}$, **dyy**=$\frac{\partial^2}{\partial y^2}$, **dzz**=$\frac{\partial^2}{\partial z^2}$,
- **label**: le numéro de référence de la frontière dans laquelle se situe le point courant, 0 sinon.
- **P** : le point courant.
- **N** : le vecteur normal sortant unitaire au point courant s'il se situe sur une frontière.
- **cin**, **cout** et **endl**: les commandes daffichage/récupération de données issues de C++, utilisées avec << et >>.
- **pi** : le nombre π .
- **true** et **false**: les booléens.
- **i**: le nombre imaginaire.

Les types de données: Les types basiques

Les opérateurs

Les opérateurs sont comme dans **C**:

```
-----------------------------------------------------
+, -, *, /, ^,
<, >, <=, >=,
&, |, // where a & b = a and b, a | b = a or b
=, +=, -=, /=, *=, !=, ==.
-----------------------------------------------------
```

Les types basiques

- Les entiers: Il s'agit du type **int**
- Les réels: Il s'agit du type **real**
- Les complexes: Il s'agit du type **complex**.
  Quelques fonctions élémentaires associées : **real**, **imag** et **conj**
- Les chaines de caractères: Il sagit de **string**
  string toto "this is a string"

Exemples:

```
-----------------------------------------------------------
int a=1,b; // entiers
real t=1.,z=0.5; // ne pas oublier le point
complex c=1.+3i; // nombre complexe 1+3i
string resultat="s"; // affichage graphique de s
-----------------------------------------------------------
```

Application:

1/ Afficher les valeurs de a, t, z, c et resultat.

2/ Afficher la partie réelle, imaginaire et le conjugué de c.

```
-----------------------------------------------------------
*
*
*
-----------------------------------------------------------
```

Les types de données: Les tableaux et les matrices

Les éléments d'un tableau sont de type **int**, **real** ou **complex**.

real $[int]$ $a(n)$;
Exemple: $a[3] = 45$;

real $[int, int]$ $A(n, m)$;
Exemple: $A[1][2] = 4$;
matrix $A = [\ [1, 1, 1, 1]\ ,\ [0, 1, 0, 0]\ ,\ [0, 0, 1, 0]\ ]$;

Exemples :

```
-----------------------------------------------------------
real[int] v(n); // vecteur de n composantes relles
complex[int] v(n); // vecteur de n composantes complexes
real[int,int] A(m,n); // matrice relles dordre mxn
matrix B=[[1,3],[2,-5]]; // dfinir la matrice B
complex[int,int] C(m,n); // matrice complexe dordre mxn
-----------------------------------------------------------
```

Manipulation :

```
-----------------------------------------------------------
real [int] u1 =[1 ,2 ,3] , u2 =2:4; // dfinir u1 et u2
real u1pu2 =u1'* u2; // donne le produit scalaire de u1 et u2,
                     // u1' est le transpos de u1;
real [int] u1du2 =u1 ./ u2; // division terme par terme
real [int] u1mu2 =u1 .* u2; // multiplication terme par terme
matrix A=u1*u2 '; // produit de u1 et le transpos de u2
matrix < complex > C=[ [1 ,1i ] ,[1+2i ,.5*1i] ];
-----------------------------------------------------------
```

Application sur les tableaux:

```
-----------------------------------------------------------
int n=10 ;
real[int] u(n) ; // Cration d?un vecteur de taille n
u=1 ; // la valeur 1 est affecte  tous les lments
u(0)=2 ; // la numrotation va de 0  n-1
u[1]=2 ; // crochets ou parenthses ...
cout<<"u="<<u<<endl ; // On affiche u
u=[0,1,2,3,4,5,6,7,8,9] ; // Dclaration de tous les termes
cout<<"u="<<u<<endl ;
cout<<"Taille="<<u.n<<endl ; // La taille du tableau
cout<<"Max="<<u.max<<" ; Min="<<u.min<<endl ; // Max et min
cout<<"Norme l1="<<u.l1<<endl ; // Norme l1
cout<<"Norme l2="<<u.l2<<endl ; // Norme l2
cout<<"Norme sup="<<u.linfty<<endl ; // Norme sup
cout<<"Somme des termes="<<u.sum<<endl ; // Somme
-----------------------------------------------------------
```

Application sur les matrices:

```
-----------------------------------------------------------
real[int,int] AA(4,4) ;
AA=[[1,2,3,4],[2,3,4,5],[0,0,1,1],[0,0,0,2]] ;
cout<<"AA="<<AA<<endl ;
int n1=3 ; int m1=4 ;
real[int,int] A(n1,m1) ; // matrice ou tableau n lignes, m colonnes
A=1 ; // tous les lments egaux a 1
cout<<"A="<<A<<endl ;
A(0,0)=2 ; // numerotation commence a 0
// A[0,0]=2 ; Ne fonctionne pas
cout<<"A="<<A<<endl ; // On affiche la matrice ou le tableau
A=[[1,1,1,1,4],[0,1,0,0,1],[0,0,1,0,-6]] ; // dclaration de la
//matrice ou du tableau
cout<<"A="<<A<<endl ;
cout<<"Nombre de lignes="<<A.n<<endl ; // nombre de lignes
cout<<"Nombre de colonnes="<<A.m<<endl ; // nombre de colonnes
// pas d'operateur max, min,l1,l2 ou linfty
-----------------------------------------------------------
```

Les types de données: Les fonctions

Fonctions élémentaires à une variable

```
pow(x,y), exp(x), log(x), log10(x), sin(x), cos(x), tan(x), acos(x),
asin(x), atan(x), sinh(x), cosh(x), tanh(x), asinh(x), acosh(x),
atanh(x)
```

Définir une fonction à une variable

```
func type nom_fct(type & var)
{
instruction 1 ;
...
...
instruction n ;
return outvar ;
}
```

Les formules:

Il s'agit de fonctions dépendant des deux variables d'espace x et y, et sont définies à partir des fonctions élémentaires.

func outvar = expression(x,y) ;

Exemples:

```
-----------------------------------------------------------
func f=2.*x+x^2;
func g=(x^2+3*y^2)*exp(1-x^2-y^2);
func h=max(-0.5,0.1*log(f^2+g^2));
func f = x+y ;
func g = imag(sqrt(z)) ;
-----------------------------------------------------------
```

Les boucles

La boucle for :

```
for (init;cond;incr) {
inst;
inst;
}
```

Exemple:

```
-----------------------------------------------------------
int sum =0;
for ( int i=1; i <=10; i++)
sum += i;
-----------------------------------------------------------
```

Les boucles

La boucle while :

```
while (cond) {
...
}
```

Exemple:

```
-----------------------------------------------------------
int i=1, sum =0;
while (i <=10) {
sum += i; i ++;
}
-----------------------------------------------------------
```

Les instructions de contrôle

```
Les instructions de controle :
if (cond) {
...
}
else {
...
}
```

Exemple:

```
-----------------------------------------------------------
int i=1, sum =0;
while (i <=10) {
sum += i; i ++;
if (sum >0) continue ;
if (i ==5) break ;
}
-----------------------------------------------------------
```

Les entrées / sorties

Pour ouvrir un fichier en lecture :

ifstream $name(nomfichier)$;

Pour ouvrir un fichier en écriture :

ofstream $name(nomfichier)$;

Lire/écrire dans un fichier >> / <<

Exemple:

```
-----------------------------------------------------------
int i;
cout << " std -out" << endl ;
cout << " enter i= ? ";
cin >> i ;
Vh uh=x+y;
ofstream f(" toto . txt "); f << uh []; // to save the solution
ifstream f(" toto . txt "); f >> uh []; // to read the solution
-----------------------------------------------------------
```

Définir un maillage
Maillage triangulaire régulier dans un domaine rectangulaire

On considère le domaine $]x_0, x_1[\times]y_0, y_1[$. Pour générer un maillage régulier $n \times m$:

mesh $nomMaillage =$ **square**$(n, m, [x_0+(x_1-x_0)\times x, y_0+(y_1-y_0)\times y])$;

Exemples:

```
-----------------------------------------------------------
mesh Th = square(4,5);
plot(Th);
mesh  Th1 = square(10,5,[1+(8-1)*x,2+(5-2)*y]);
plot(Th1);
-----------------------------------------------------------
```

Maillage triangulaire non structuré défini à partir de ses frontières

Pour définir les frontières, on utilise la commande border:

border $name(t = deb, fin)\{x = x(t); y = y(t); label = numlabel\};$

Pour définir un maillage à partir de ses frontières, on utilise buildmesh:

mesh $nomMaillage =$ **buildmesh**$(a1(n1) + a2(n2) + ... + ak(nk));$

Exemples:

```
-----------------------------------------------------------
border a(t=0,1) { x=t; y=0 ;}; // bottom
border b(t=0,1) { x=1; y=t ;}; //  rigth
border c(t=1,0) { x=t; y=1 ;}; // top
border d(t=1,0) { x=0; y=t;}; // left
mesh th = buildmesh(a(3)+b(5)+c(10)+d(8));
plot(th);
-----------------------------------------------------------
```

Divers autour le maillage

Lire un maillage externe

mesh NomMaillage (" NomFichier.msh");

avec emc2, bamg, modulef, etc...

Exemple: mesh Th2(" april-fish.msh") ;

Entrées/sorties fichiers

savemesh (NomMaillage," NomFichier.msh");

mesh NomMaillage = **readmesh** (" NomMaillage.msh");

Autres fonctions sur les maillage

mesh Mail2 = **movemesh** (mail1,[f1(x,y),f2(x,y)]) ;

mesh Mail2 = **adapmesh** (mail1,var) ;

Lire les données d'un maillage

- **Th.nt** (nombre de triangle),
- **Th.nv** (nombre de noeuds),
- **Th[i][j]** (sommet j du triangle i),
- ...

Exemple 1

```
-----------------------------------------------------------
real Dx =.2; // discretization space parameter
int aa=-5,bb =5, cc=-1,dd =1;
border AB (t = aa , bb){x = t ;y = cc; label = 1;};
border BC (t = cc , dd){x = bb;y = t ; label = 2;};
border CD (t = bb , aa){x = t ;y = dd; label = 3;};
border DA (t = dd , cc){x = aa;y = t ; label = 4;};
mesh Th = buildmesh ( AB( floor (abs(bb -aa)/Dx))
+ BC( floor ( abs (dd - cc)/Dx))
+ CD( floor ( abs (bb -aa)/Dx))
+ DA( floor (abs(dd -cc)/Dx)) );
plot ( AB( floor ( abs (bb -aa)/Dx)) + BC( floor ( abs (dd -cc)/Dx))
+ CD(floor ( abs (bb -aa)/Dx)) + DA( floor (abs(dd -cc)/Dx)) );
//to see the border
plot ( Th , ps=" mesh.eps "); // to see and save the mesh
-----------------------------------------------------------
```

Exemple 2

```
mesh Th= square (m,n ,[x,y]); // build a square with m point on x
//direction and n point on y direction
mesh Th1 = movemesh (Th ,[x+1,y *2]) ; // translate the square
//]0 ,1[*]0 ,1[ to a rectangle ]1 ,2[*]0 ,2[
savemesh (Th1 ,"Name.msh "); // to save the mesh
mesh Th2 (" mesh.msh "); // to load the mesh
```

Exemple 3

```
border C(t=0,2*pi){ x=cos(t);y=sin(t);label =1;};
mesh Mesh_Name = buildmesh(C (50) );
```

Exemple 4

```
-----------------------------------------------------------
border a(t=0 ,2*pi){ x= cos(t); y= sin(t); label =1;};
border b(t=0 ,2*pi){ x =0.3+0.3* cos(t); y =0.3* sin (t); label =2;};
mesh Th1 = buildmesh (a(50) +b(+30) );
mesh Th2 = buildmesh (a(50) +b(-30));
plot(Th1);
plot(Th1);
-----------------------------------------------------------
```

Remarquons que la frontière b peut avoir un argument positif ou négatif. Alors, que peut-on conclure de l'exemple 4?

Résoudre une EDP

Définition de l'espace d'approximation

fespace NomEspace(NomMaillage,TypeElementsFinis);

Exemple:

```
fespace Vh(Th,P1);
```

Les types d'éléments finis est un mot-clé dans la liste suivante:
$P0$, $P1$, $P1dc$, $P1b$, $P2$, $P2b$, $P2dc$, RT, $P1inc$.
L'espace ainsi défini est à son tour un type de données pour définir les variables de type éléments finis.
Exemple:

```
Vh u,vh;
```

Résoudre une EDP

Définir le problème variationnel

problem pbName(u,v,solver)=

```
a(u,v) - l(v)
+ (conditions aux limites) ;
```

Pour résoudre un problème variationnel, il suffit de taper la commande:

pbName ;

Résoudre une EDP

Formes bilinéaires

- **int1d**$(Th, n1, ..., nk)(A * u * v) \iff \sum_{T \in T_h} \int_{\partial T \cap (\cup_i \Gamma_{n_i})} Auv.$
- **int2d**$(Th[, k])(A * u * v) \iff \sum_{T \in T_h[\cap \Omega_k]} \int_T Auv.$
- **intalledges**$(Th[, k])(A * u * v) \iff \sum_{T \in T_h[\cap \Omega_k]} \int_{\partial T} Auv.$

Formes linéaires

- **int1d**$(Th, n1, ..., nk)(A * v) \iff \sum_{T \in T_h} \int_{\partial T \cap (\cup_i \Gamma_{n_i})} Av.$
- **intalledges**$(Th[, k])(A * v) \iff \sum_{T \in T_h[\cap \Omega_k]} \int_{\partial T} Av.$

Résoudre une EDP

Visualiser les résultats

Directement avec Freefem++: Pour afficher des maillages, les courbes d'isovaleurs et les champs de vecteurs:

$$\textbf{plot}(var1,[var2,var3],\ldots,[liste\ d'options]);$$

avec:

- $wait = true/false$,
- $value = true/false$,
- $fill = num$,
- $ps = "nomfichier"$,
- $\ldots$

Premier exemple: Equation de POISSON.

Les étapes à suivre:

- Définir le maillage
- Définir l'espace Elements Finis
- Déclaration des variables
- Résolution
- Sauvegarder $\cdots$

A travers d'un exemple !

Equation de Poisson

Trouver $u : \Omega :=]0,1[^2 \to \mathbb{R}$ tel que:

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

Formulation Variationnelle

Trouver $u \in H_0^1(\Omega)$ tq pour tout $v \in H_0^1(\Omega)$,

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

Approximation de Galerkin – En utilisant Elements finis P1

Trouver $u_h \in V_h$ tq pour tout $v_h \in V_h$,

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

$$V_h = \{v_h \in H_0^1(\Omega) \ : \ v_h|_T \in P_1, \forall T \in \mathcal{S}_h\},$$

Poisson.edp

```
mesh Sh= square(10,10);      //   génération du maillage d'un carré
fespace Vh(Sh,P1);                    //   Espace Elements finis P1
Vh u,v;                              //   u et v des éléments de Vh
func f=cos(x)*y;                   //    f est une fonction en x et y
problem Poisson(u,v)=                 //    Definition du problème
    int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v))      //    forme bilinéaire
    -int2d(Sh)(f*v)                        //    forme linéaire
+on(1,2,3,4,u=0);                     //    condition de Dirichlet
Poisson;                    //    Résolutionde l'équation de Poisson
plot(u);                              //    tracer la solutioin
```

Exercice:

Tester les options: value, wait, fill, ps $\cdots$

Sauvegarde et Lecture

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 :)");
```

Conditions aux Limites

Trouver $u : \Omega :=]0,1[^2 \to \mathbb{R}$ tq:

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

Formulation Variationnelle

Trouver $u \in V := \{v \in H^1(\Omega) \ : \ v = u_d \text{ sur } \Gamma_D\}$, telque pour tout $v \in V^D := \{v \in H^1(\Omega) \ : \ v = 0 \text{ sur } \Gamma_D\}$,

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

Approximation de Galerkin – P1 Elements finis P1

Trouver $u_h \in V_h$ tq pour tout $v_h \in V_h^D$,

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

$$V_h = \{v_h \in V \ : \ v_h|_T \in P_1, \forall T \in \mathcal{S}_h\},$$

$$V_h^D = \{v_h \in V^D \ : \ v_h|_T \in P_1, \forall T \in \mathcal{S}_h\}.$$

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);
```

Adaptation de maillage

Le maillage peut être adapté en utilisant `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);
```

Singularités des coins

```
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.)) ; } ;
```

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

Les éléments finis P3 ne sont pas disponibles par défaut. Ils doivent être chargés.

```
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);
                                                          //   ...
```

Résoudre et Tracer !

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");
```

Equation de Laplace (formulation mixte)

On cherche à résoudre:

$$\begin{cases} -\Delta p = f & \text{dans } \Omega \\ p = g & \text{sur } \partial\Omega. \end{cases}$$

avec: $\vec{u} = \nabla p$.

Ce problème reste équivalent à :

Trouver $\vec{u}$, p solutions dans un domaine Ω tq:

$$\begin{cases} -\nabla.\vec{u} = f & \text{dans } \Omega \\ \vec{u} - \nabla p = 0 & \text{dans } \Omega \\ p = g & \text{sur } \partial\Omega. \end{cases}$$

Formulation variationnelle Mixte :

Trouver $\vec{u} \in H(div, \Omega)$, $p \in L^2(\Omega)$:

$$\int_\Omega q\, \nabla.\vec{u} + \int_\Omega p\, \nabla.\vec{v} + \vec{u}.\vec{v} = \int_\Omega -fq + \int_\Gamma g\vec{v}.\vec{n} \quad \forall (\vec{v}, q) \in H(div) \times L^2$$

Equation de Laplace (formulation mixte)

```
mesh Th=square(10,10) ;
fespace Vh(Th,RT0) ;
fespace Ph(Th,P0) ;
Vh [u1,u2],[v1,v2] ;
Ph p,q ;
func f=1. ; func g=1 ;
problem laplaceMixte([u1,u2,p],[v1,v2,q],solver=LU) =          //
int2d(Th)( p*q*1e-10 + u1*v1 + u2*v2
+ p*(dx(v1)+dy(v2)) + (dx(u1)+dy(u2))*q )
- int2d(Th) ( -f*q)
- int1d(Th)( (v1*N.x +v2*N.y)*g) ;                  //    int on gamma
laplaceMixte ;                          //    the problem is now solved
plot([u1,u2],coef=0.1,wait=1,ps="lapRTuv.eps",value=true) ;
plot(p,fill=1,wait=1,ps="laRTp.eps",value=true) ;
```