Introduction à FreeFem++ (suite)

O. Pantz et M. Kallel
CMAP, Ecole Polytechnique, Palaiseau, France
ENIT-LAMSIN, Tunis, Tunisie

Merci à F. Hecht...

Où télécharger les logiciels libres qui vont bien ?

- `http://www.freefem.org`
- Visualisation de maillages `Medit`
  `http://www-rocq.inria.fr/gamma/medit/medit.html`
- Éditeur de texte sous Windows `Notepad++`
  `http://notepad-plus.sourceforge.net`

Retrouver cette présentation (avec les scripts) sur
`http://www.cmap.polytechnique.fr/~pantz`

PLAN

- Matrices, tableaux et vecteurs
  - Vecteurs
  - Tableaux
  - Matrices
- Eléments finis et vecteurs, formulations variationnelles et matrices
  - Formulations Variationnelles
  - Conditions aux bords de Dirichlet
  - Laplacien avec CL de Neumann
  - Changements de maillage
  - Un exemple sur un problème d'évolution
- Formulations Galerkin discontinues
  - jump, average, nTonEdge, intallEdge, ...
  - Laplacien (encore...) P_1^{dc}
  - Cas d'un domaine avec fracture
- Vecteurs et valeurs propres
- Algorithmes
- Pour le "fun" : Vesicules dans un écoulement

Déclaration de vecteurs

```
int n=10 ;
real[int] u(n) ;                    //    Création d'un vecteur de taille n
u=1 ;                    //    la valeur 1 est affectée à tous les éléments
u(0)=2 ;                              //    la numérotation va de 0 à n-1
u[1]=2 ;                                //    crochets ou parentheses ...
cout<<"u="<<u<<endl ;                                    //    On affiche u
u=[0,1,2,3,4,5,6,7,8,9] ;              //   Déclaration 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
```

Execute declarationvecteurs.edp

Opérations sur les vecteurs

```
int n=10;
real[int] u(n),v(n),w(n),r(n);        //    Création de vecteurs de
taille n
u=[0,1,2,3,4,5,6,7,8,9];                    //    Déclaration de u
v=[9,8,7,6,5,4,3,2,1,9];                    //    Déclaration de v
w=2.*u+1./2.*v;      //    Combinaison linéaire de vecteurs v/2=non
autorisé
cout<<"u="<<u<<endl;
cout<<"v="<<v<<endl;
cout<<"w=2*u+1/2*v="<<w<<endl;
r=1;
r(0:5)=w(n-6,n-1);                    //    Extraction de tableau
cout<<"r="<<r<<endl;
cout<<"u'*u="<<u'*u<<endl;              //   u'= transposée de u;
*=multiplication matrices
```

Execute operationsvecteurs.edp

Tableaux I/II

```
int n=3 ; int m=4 ;
real[int,int] A(n,m) ;                    //    Tableau n lignes, m colonnes
A=1 ;                                      //    tous les elements egaux a 1
A(0,0)=2 ;                                   //    numerotation commence a 0
                                           //    A[0,0]=2 ; Ne fonctionne pas
cout<<"A="<<A<<endl ;                       //    On affiche le tableau
A=[[1,1,1,1],[0,1,0,0],[0,0,1,0]] ;       //    déclaration de 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'opérateur max, min,l1,l2 ou linfty
                                                          //    ...
```

Tableaux II/II

```
real[int] b(4),c(3) ;                    //    vecteurs de 4 elements
b=[0,1,2,3] ;
c=A*b ;                                  //    produit matrice vecteur
cout<<"b="<<b<<endl ;
cout<<"c=A*b="<<c<<endl ;
real[int,int] B(4,3) ;
B=b*c' ;
cout<<"B=b*c'="<<B<<endl ;
                                 //    A+B ; B*A ; 2.*B ; non definis!!!
```

Execute tableaux.edp

Déclarations Matrices I/II

```
real[int,int] A(3,3);
A=[[1,2,3],[1,0,0],[0,16,18]];
matrix B=A;                              //    B est une matrice "creuse"
cout<<"A="<<A<<endl;
cout<<"B=A="<<B<<endl;   //    Stockage: indices non nuls seulement
int[int] I(1),J(1);
real[int] C(1);
[I,J,C]=B;     //    On récupère les indices (i,j) et leur valeur c
cout<<"I="<<I<<endl<<"J="<<J<<endl
<<"C="<<C<<endl;
I=[0,1,3,2];J=[0,1,2,3];C=[pi,0.5,3.333,4.5];
B=[I,J,C];                               //    Matrice B(I(i),J(i))=C(i)
cout<<"I="<<I<<endl<<"J="<<J<<endl<<"C="<<C<<endl;
cout<<"B=[I,J,C]="<<B<<endl;
                                                              //    ...
```

Déclarations Matrices II/II

```
real[int] d(B.n);
d=B.diag;                          //    Diagonale de la matrice carrée B
cout<<"d=B.diag="<<d<<endl;
d=2.*d;
B.diag=d;
cout<<"Fixe la diagonale de B a 2.*d; B="<<B<<endl;
matrix D=[d];                                  //    Matrice diagonale
cout<<"D=[d]="<<D<<endl;
matrix E=[[B,D],[D',B]];                             //    Matrice bloc
cout<<"E=[[B,D],[D',B]]="<<E<<endl;
```

Execute declarationmatrices.edp

Algèbre matriciel

```
real[int,int] A(4,4);
A=[[1,2,3,4],[2,3,4,5],[0,0,1,1],[0,0,0,2]];
matrix B=A;
real[int] b=[1,1,1,1];
real[int] c(b.n);
c=B*b;                          //    Multiplication Matrice / vecteur
cout<<"B="<<A<<endl;
cout<<"b="<<b<<endl;
cout<<"c=B*b="<<c<<endl;
set(B,solver=UMFPACK);
real[int] d(4);
d=B^-1*c;                              //    Résolution du système Bc=d
cout<<"B^-1*c="<<d<<endl;
matrix C,D,E;
C=2.*B;                         //    Combinaison Linéaire de Matrices
D=2.*B+(-1)*C;                             //    D=2.*B-C; non autorisé
cout<<"D="<<D<<endl;
```

Execute algebrematriciel.edp

Formulations variationnelles I/V

On considère l'EDP

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

Cette EDP est équivalente au problème variationnel consistant à trouver $u \in H_0^1(\Omega)$ tel que et

$$a(u, v) = L(v)$$

pour tout $v \in H_0^1(\Omega)$ avec

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \text{ et } L(v) = \int_\Omega fv.$$

Une approximation de Galerkin consiste à déterminer $u_h \in V_h$ tel que

$$a(u_h, v_h) = L(v_h)$$

pour tout $v_h \in V_h$, où V_h est un sous espace de dimension finie de $H_0^1(\Omega)$.

Formulations variationnelles II/V

La méthode des éléments finis P_1 sur un maillage $\mathcal{S}_h$ consiste à choisir

$$V_h = \{u \in C(\overline{\Omega}; \mathbb{R}) \ : \ u = 0 \text{ sur } \partial\Omega \ , \ u_{|T} \in P_1$$
$$\text{pour tout } T \text{ triangle du maillage } \mathcal{S}_h\},$$

où P_1 désigne l'ensemble des polynômes de degré 1.

Formulations variationnelles III/V

Comme V_h est un espace vectoriel de dimension finie, on peut décomposer u_h et v_h dans une base de vecteurs (appelés ici fonctions de base) et noté (Φ_i). On note alors

$$u_h = \sum_i u_h^i \Phi_i, \text{ et } v_h = \sum_i v_h^i \Phi_i.$$

Ainsi, $a(u_h, v_h) = L(v_h)$ si et seulement si pour tout vecteur (v_h^i), on a

$$\sum_{i,j} a(\Phi_i, \Phi_j) u_h^i v_h^j = \sum_j L(\Phi_j) v_h^j.$$

En d'autres termes, on doit avoir

$$AU = b,$$

où U est le vecteur (u_h^i), A est la matrice $A_{ij} = a(\Phi_i, \Phi_j)$ et b est le vecteur $b_j = L(\Phi_j)$.

Formulations variationnelles IV/V

Application au Laplacien

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

```
mesh Sh=square(50,50) ;                    //    Définition du maillage
fespace Vh(Sh,P1) ;                       //    Définition de l'espace Vh
func f=10. ;
func g=cos(x) ;
varf a(u,v)=int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v))
+on(1,2,3,4)(u=g) ;   //    Définition de la forme bilineaire a(u,v)
varf L(u,v)=int2d(Sh)(f*v)+on(1,2,3,4,u=g) ;  //    Définition de la
forme linéaire b(v)
```

Formulations variationnelles V/V

```
matrix A=a(Vh,Vh) ;                    //    Matrice A_ij = a(Φ_i, Phi_j)
real[int] b=L(0,Vh) ;                  //    Le second membre b_j = L(Φ_j)
Vh uh=0 ;                                                  //    u_h ∈ V_h
uh[]=A^-1*b ;  //   uh[] est le vecteur U = (u_h^i) des coordonnées de
u dans la base de fonctions propres
plot(uh,wait=1) ;
```

Execute fv.edp

Au sujet des conditions aux bords

Les conditions aux bords de type Dirichlet sont imposées par pénalisation. Si i est un degré de liberté fixé à la valeur g_i, A_{ii} est incrémenté de `tgv` et b_i de `tgv`$*g_i$, où `tgv` prend une Très Grande Valeure.

```
mesh Sh=square(10,10) ;
fespace Vh(Sh,P1) ;
varf L(u,v)=on(1,2,3,4,u=1) ;
Vh u=0 ;
u[]=L(0,Vh,tgv=1) ;      //    On peut modifier tgv u=tgv sur le bord
plot(u,wait=1,value=1) ;
```

Execute tgv.edp

Laplacien avec conditions de Neumann I/III

On souhaite résoudre

$$\begin{cases} -\Delta u = f & \text{sur } \Omega \\ \frac{\partial u}{\partial n} = 0 & \text{sur } \partial\Omega \end{cases}$$

On introduit la formulation mixte consistant à déterminer $u \in H^1(\Omega)$ et $p \in \mathbb{R}$ tels que pour tout $v \in H^1(\Omega)$ et tout $q \in \mathbb{R}$,

$$\int_\Omega \nabla u \cdot \nabla v + p \int_\Omega u + q \int_\Omega v = \int_\Omega f v,$$

c'est à dire

$$a(u,v) + p s(v) + q s(u) = L(v),$$

avec

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v, \quad s(v) = \int_\Omega v \text{ et } L(v) = \int_\Omega f v.$$

Laplacien avec conditions de Neumann II/III

```
mesh Sh=square(50,50,[x-0.5,y-0.5]);                    //   Maillage
fespace Vh(Sh,P1);                          //   Vh est l'espace d'EF P1
func f=x*y;                                 //   forces appliquées
varf a(u,v)=int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v));
                          //   forme bilinéaire a(u,v) = ∫_Ω ∇u · ∇v
varf L(u,v)=int2d(Sh)(f*v);          //   forme linéaire L(v) = ∫_Ω fv
varf s(u,v)=int2d(Sh)(v);             //   forme linéaire s(v) = ∫_Ω v
                                                        //   ...
```

Laplacien avec conditions de Neumann III/III

```
matrix B=a(Vh,Vh) ;          //   Assemblage matrice B_{i,j} = a(Φ_i, Φ_j)
real[int] c=s(0,Vh) ;        //   Assemblage vecteur c_i = s(Φ_i)
real[int] d=L(0,Vh) ;        //   Assemblage vecteur d_i = L(Φ_i)
matrix A=[[B,c],[c',0]] ;    //   Définition de
```

$$A = \begin{pmatrix} B & c \\ c^* & 0 \end{pmatrix}$$

```
real[int] b=[d,0] ;          //   Définition du vecteur b = (d, 0)
set(A,solver=UMFPACK) ;      //   On doit choisir un solveur
real[int] res(Vh.ndof+1) ;   //   ndof = Number of Degree Of Freedom
r=A^-1*b ;                   //   r = A^{-1} b
Vh u=0 ; real p ;
[u[],p]=r ;                  //   u = Σ_i r_i Φ_i
plot(u,wait=1) ;
```

Execute Neumann.edp

Changements de maillages I/IV
Elasticité

```
mesh Sh=square(5,5,[x-0.5,y-0.5]);                    //   Maillage
fespace Vh(Sh,[P1,P1]);          //   Vh est l'espace d'EF P1 x P1
Vh [u,v]=[0,0];                                       //   (u,v) ∈ Vh
u[]=1.;                    //   (u,v) = ∑_i Φ_i où Φ_i est une base de Vh
plot([u,v],wait=1,value=1);      //   (u,v) = (1,1) et non (0,1)
v[]=-1.;                                              //   (u,v) = ∑_i 2Φ_i
plot([u,v],wait=1,value=1);      //   (u,v) = (2,2) et non (1,2)
```

Execute Vectoriel.edp

Changements de maillages II/IV
Elasticité

```
mesh Sh=square(1,1);                    //   Maillage très grossier
fespace Vh(Sh,P1);                      //   Vh est l'espace d'EF P1
Vh u=cos(pi*x)*cos(pi*y);
plot(Sh,u,wait=1);
Sh=adaptmesh(Sh,0.05,IsMetric=1);       //   Adaptation du maillage
Sh=adaptmesh(Sh,0.05,IsMetric=1);       //   Adaptation du maillage
plot(Sh,wait=1,cmm="Le nouveau maillage");
string legende="u est defini sur l'ANCIEN maillage u[] est de
taille "+u[].n;
plot(u,cmm=legende,wait=1);
u=cos(pi*x)*cos(pi*y);
legende="u est defini sur le nouveau maillage u[] est de taille
"+u[].n;
plot(Sh,u,cmm=legende,wait=1);
```

Execute AfterAdapt.edp

Changements de maillages III/IV
Elasticité

```
mesh Sh=square(10,10) ;
real mu=1.,lambda=1. ;
func f1=0 ;func f2=-2. ;
fespace Vh(Sh,[P2,P2]) ;          //    Vh est l'espace d'EF P1 × P1
Vh [u1,u2],[v1,v2] ;
macro e(u1,u2)
 [dx(u1),(sqrt(2.)*dx(u2)+dy(u1))/2.,dy(u2)]                    //
macro div(u1,u2) (dx(u1)+dy(u2))                               //
solve elasticity([u1,u2],[v1,v2])=
int2d(Sh)(2.*mu*(e(u1,u2)'*e(v1,v2))+lambda*div(u1,u2)*div(v1,v2))
-int2d(Sh)(f1*v1+f2*v2)+on(1,u1=0,u2=0) ;
```

Changements de maillages IV/IV
Elasticité

```
fespace Wh(Sh,P1) ;
Wh sigma,sigma2 ;
sigma=2.*mu*(e(u1,u2)'*e(u1,u2))+lambda*div(u1,u2)^2 ;
Sh=movemesh(Sh,[x+u1,y+u2]) ;
sigma2=0 ;      //    Déclare σ2 comme un EF sur le nouveau maillage
sigma2[]=sigma[] ;              //    σ2 a les meme coordonnées que σ
plot(sigma,fill=1,wait=1,cmm="contraintes dans la configuration de
référence") ;
plot(sigma2,fill=1,wait=1,cmm="contraintes dans la configuration
déformée") ;
```

Execute elasticite.edp

Un problème d'évolution I/IV

On souhaite déterminer l'évolution d'un solide élastique, solution du problème variationnel consistant à déterminer
$u \in C^1([0,T];L^2(\Omega)^2) \cap C^0([0,T];H^1(\Omega)^2)$ tel que pour tout $t \in]0,T[$ et tout $v \in H^1(\Omega)^2$,

$$\int_\Omega \rho \frac{d^2u}{dt^2} v dx + \int_\Omega 2\mu e(u) \cdot e(v) + \lambda (\mathrm{div}\, u)(\mathrm{div}\, v) = \int_\Omega f \cdot v,$$

où f sont les forces appliquées, μ et λ les coefficients de Lamé du solide, ρ la masse volumique et $e(\cdot)$ le tenseur symétrisé. Il faut éventuellement ajouter des conditions aux limites (temps et espace). On effectue une discrétisation de type différences finies en temps. On

peut (par exemple) considéré les schéma impicite, centré d'ordre 2 consistant à déterminer $u^n \simeq u(n\Delta t)$

$$\int_\Omega \rho \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} v + \theta \int_\Omega 2\mu e(u^{n+1}) \cdot e(v) + \lambda(\mathrm{div}\, u^{n+1})(\mathrm{div}\, v) + (1-\theta) \int_\Omega 2\mu e(u^{n-1}) \cdot e(v) + \lambda(\mathrm{div}\, u^{n-1})(\mathrm{div}\, v) = \int_\Omega f \cdot v.$$

Un problème d'évolution II/IV

```
real Lx=5., Ly=1., dn=7., mu=10., lambda=0., rho=1. ;
real Dt=0.5, T=100., exa=0.3 ;
func f1=0. ;func f2=0. ;

mesh Sh=square(Lx*dn,Ly*dn,[Lx*x,Ly*y]) ;

fespace Vh(Sh,[P2,P2]) ;
macro e(u1,u2)
 [dx(u1),(sqrt(2.)*dx(u2)+dy(u1))/2.,dy(u2)]                    //
macro div(u1,u2) (dx(u1)+dy(u2))                               //

Vh [u1,u2],[v1,v2] ;
Vh [up1,up2],[upp1,upp2] ;

[upp1,upp2]=[0.,0.] ;                          //    Déplacement initial
[v1,v2]=[0.,-1] ;                                 //    Vitesse initiale
[up1,up2]=[upp1+Dt*v1,upp2+Dt*v2] ;                     //    au temps Dt
```

Un problème d'évolution III/IV

```
real t=0;
real theta=0.6;                          //    theta >= 1/2 => c'est stable
problem elasticity([u1,u2],[v1,v2],init=t,solver=Cholesky)=
int2d(Sh)(rho*([u1,u2]'*[v1,v2])/(Dt)^2)
+int2d(Sh)(theta*(2.*mu*(e(u1,u2)'*e(v1,v2))
                                        +lambda*div(u1,u2)*div(v1,v2)))
+int2d(Sh)((1.-theta)*(2.*mu*(e(upp1,upp2)'*e(v1,v2))
                                      +lambda*div(upp1,upp2)*div(v1,v2)))
-int2d(Sh)(f1*v1+f2*v2)
-int2d(Sh)(rho*([2.*up1-upp1,2.*up2-upp2]'*[v1,v2])/(Dt)^2)
+on(4,u1=0,u2=0);
```

Un problème d'évolution IV/IV

```
mesh Th=Sh ;
mesh Th0=Sh ;
fespace Ph(Th,P1) ;
fespace Wh(Sh,P1) ;
Ph sigma2=0 ; Wh sigma ;

while(t<T){
elasticity ;
sigma=(2.*mu*(e(u1,u2)'*e(u1,u2))+lambda*div(u1,u2)*div(u1,u2)) ;
Th=movemesh(Sh,[x+exa*u1,y+exa*u2]) ;
sigma2=0 ;sigma2[]=sigma[] ;
plot(sigma2,fill=1,wait=0,bb=[[0,-Ly],[Lx+Ly,2.*Ly]],viso=viso,hsv=color
t+=Dt ; [upp1,upp2]=[up1,up2] ; [up1,up2]=[u1,u2] ;
} ;
```

Execute elasticite-evolution.edp

Formulations Galerkin discontinues

FreeFem++ permet d'utiliser des formulations de type Galerkin ou volumes finis à l'aide des mots clés jump, average, intalledges et nTonEdge.

$$\texttt{intalledges(Th)(u)} = \sum_{T\in\mathcal{T}_h} \int_{\partial T} u.$$

Sur chaque arête e d'un triangle T,

$$\texttt{jump(u)}(x) = \lim_{t\to 0+} u(x+tn) - u(x-tn),$$

où n est la normale extérieure au triangle T et la convention $u(x) = 0$ si $x \notin \Omega$.

$$\texttt{average(u)}(x) = \lim_{t\to 0+} (u(x+tn) + u(x-tn))/2,$$

nTonEdge= Nombre de triangles adjacents à l'arête.

Le Laplacien avec P_1^{dc} I/II

On peut résoudre le problème d'approximation du Laplacien à l'aide d'éléments finis P_1 discontinus. En effet, la résolution du problème consistant à minimiser

$$\frac{1}{2}\int_\Omega |\nabla u|^2 - \int_\Omega fu$$

sur l'ensemble des $u_h \in V_h^0$ (éléments finis P_1 nuls sur le bord du domaine) équivaut à minimiser

$$\frac{1}{2}\sum_{T\in\mathcal{S}_h}\int_T |\nabla u|^2 - \int_\Omega fu$$

sur l'ensemble des u, éléments finis P_1 discontinus de sauts nuls aux arêtes. A cet effet, on introduit des multiplicateurs de Lagrange.

Le Laplacien avec P_1^{dc} II/II

On cherche $(u,p) \in P_1^{dc} \times P_1^{dc}$ tels que

$$\sum_{T\in\mathcal{S}_h} \int_T \nabla u \cdot \nabla v - \int_\Omega fv + \int_{\partial T} [u][q] + [v][p] - \int_T \varepsilon pq = 0$$

pour tous $(v,q) \in P_1^{dc} \times P_1^{dc}$.

```
mesh Sh=square(50,50,[x,y]) ;          //   Définition du Maillage Sh
func f=-1 ;                                                //   Second membre
fespace Vhdc(Sh,P1dc) ;      //   Espace Vh^dc des EF P1 discontinues
Vhdc u,v,p,q ;                                             //   u,v,p et q ∈ Vh^dc
solve LaplacienP1dc(u,p,v,q)=
  int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v)+u*v)
 -intalledges(Sh)(jump(q)*jump(u)+jump(p)*jump(v))
 -intalledges(Sh)(1.e-10*(p*q))
 -int2d(Sh)(f*v) ;
plot(u,wait=1) ;
```

Execute P1dc.edp

Application aux domaine fracturés I/II

On peut utiliser le précédent script afin de traiter le cas des domaines fracturés. Il suffit de ne pas imposer la contrainte de saut nul le long de la fracture (et le long de la frontière du domaine si on impose des conditions aux limites de type Neumann).

```
border a(t=0,2*pi){x=cos(t);y=sin(t);label=1;};
border fracture(t=0,1){x=t;y=0;label=2;};
mesh Sh=buildmesh(a(50)+fracture(20));          //    Définition du
Maillage
func f=x+y;
fespace Vh(Sh,P1);
varf idfracture(u,v)=on(2,u=1.);        //    On repère la fracture
Vh onfracture=0;  onfracture[]=idfracture(0,Vh,tgv=1);
plot(Sh,onfracture,wait=1);
                                                        //    ...
```

Application aux domaine fracturés II/II

```
fespace Vhdc(Sh,P1dc) ;
Vh1dc u,v,p,q ;
solve LaplacienP1dc(u,p,v,q)=
  int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v)+u*v)
 -intalledges(Sh)((jump(q)*jump(u)+jump(p)*jump(v))
*(nTonEdge-1.)*(1.-onfracture))
          //    On impose la continuité sauf le long de la fracture
                                              //    et le long du bord
 -intalledges(Sh)(1.e-10*(p*q))-int2d(Sh)(f*v) ;
plot(u,wait=1) ;
```

Execute fracture.edp

Vecteurs propres et valeurs propres I/II

On souhaite déterminer $(u, \lambda) \in H_0^1(\Omega) \times \mathbb{R}$ tels que pour tout $v \in H_0^1(\Omega)$,

$$a(u, v) = \lambda m(u, v),$$

où

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \text{ et } m(u, v) = \int_\Omega uv$$

Vecteurs propres et valeurs propres II/II

```
mesh Sh=square(50,50);
fespace Vh(Sh,P1);                          //    Vh = Eléments finis P1
varf a(u,v)=int2d(Sh)(dx(u)*dx(v)+dy(u)*dy(v))+on(1,2,3,4,u=0);
                                            //    a(u,v) = ∫Ω ∇u · ∇v
varf m(u,v)=int2d(Sh)(u*v);                 //    m(u,v) = ∫Ω uv
matrix A=a(Vh,Vh);                          //    Aij = a(Φi, Φj)
matrix M=m(Vh,Vh);                          //    Mij = m(Φi, Φj)
int neV=20;     //    nombre de valeurs propres et vecteurs propres
stockés
real[int] lambda(neV);                      //    valeurs propres
Vh[int] u(neV);                             //    vecteurs propres
int k=EigenValue(A,M,sym=true,sigma=0,value=lambda,
vector=u,tol=1.e-10,ncv=0,maxit=0);    //    Calcul des vecteurs et
valeurs propres
for(int i=0;i<k;i++)
plot(u[i],cmm="Vecteur propre n."+(i+1)+"; valeur
propre="+lambda(i),wait=1);
```

Execute propres.edp

Algorithmes

Afin de résoudre des problèmes de type $J'(u) = 0$, divers algorithmes sont disponibles, à savoir

- Le Gradient Conjugué linéaire ou Non Linéaire (`LinearCG` et `NLCG`) pour les problèmes convexes.
- The Generalized minimal residual Method pour les problèmes non symétriques (`LinearGMRES`).
- La méthode de Newton (`Newton`)
- La méthode Broyden-Flechter-Goldfrab-Shanno (`BFGS`)

Les méthodes `Newton` et `BFGS` sont importées du package COOOL et utilisent des matrices pleines.

Gradient Conjugué Non Linéaire I/IV

On cherche $u : C^1(\Omega) \mapsto \mathbb{R}$ minimisant l'aire de la surface S paramétrée par $(x, y) \in \Omega \mapsto (x, y, u(x, y))$, sur l'ensemble des fonctions u tels que $u_{|\partial\Omega} = u^0_{|\partial\Omega}$.

Le problème consiste donc à minimiser

$$J(u) = \iint_\Omega \left\| \begin{pmatrix} 1 \\ 0 \\ \partial_x u \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ \partial_y u \end{pmatrix} \right\|_2 dxdy = \iint_\Omega \sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2} dxdy$$

d'équation d'Euler associée

$$\forall v / v_{|\partial\Omega} = 0 \quad : \quad DJ(u)v = -\int \frac{(\partial_x v \partial_x u + \partial_y v_y \partial_y u)}{\sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2}} = 0$$

Gradient Conjugué Non Linéaire II/IV

Exemple d'utlisation de NLCG

```
    func real J(real[int] & xx)                    //    La fonctionnelle à minimiser
    {  real s=0;
       ...                          //    Le code qui calcul J(u) en fonction de xx=u[]
       return s; }
  func real[int] DJ(real[int] &xx)           //    Le gradient de la fonctionnelle
    { ....                           //    calcul la dérivée J'(u) en fonction de u[]
  ....                                                //    stocke le resultat dans xx
       return xx; };                          //    Retourne une variable non locale=ok
  ...
  NLCG(DJ,x,eps=1.e-6,nbiter=20,precon=matId);
```

Pour visualiser le résultat avec medit

```
 savemesh(Th,"mm",[x,y,u]);   //   sauve mm.points et mm.faces pour
medit
 exec("medit mm;rm mm.bb mm.faces mm.points");  //    appel à medit
```

Gradient Conjugué Non Linéaire III/IV

```
func g=cos(2*x)*cos(2*y) ;                          //    valeur au bord
mesh Th=square(20,20,[x*pi,y*pi]) ;                 //    Maillage de Ω
fespace Vh(Th,P1) ;

func real J(real[int] & xx)       //    La fonctionnelle à minimiser
  { Vh u ;u[]=xx ;    //    Defini l'EF u à l'aide de ces coordonnées
xx
    return int2d(Th)( sqrt(1 +dx(u)*dx(u) + dy(u)*dy(u) ) ) ; }

func real[int] dJ(real[int] & x)                //    Le gradient de J
  { Vh u ;u[]=xx ;    //    Defini l'EF u à l'aide de ces coordonnées
xx
    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,uh=0) ;
    return xx= au(0,Vh) ;  }          //    Retourne une variable non
locale=ok
```

Gradient Conjugué Non Linéaire IV/IV

```
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]);          //    sauve la surface pour medit
exec("medit mm;rm mm.bb mm.faces mm.points");
```

Execute minimal-surf.edp

Juste pour le "fun"...
Vesicules dans un écoulement

On considère un système constitué de vesicules dans un écoulement. Chaque vesicule est définie comme un région du maillage global. A chaque itération, le domaine est remaillé complètement. On assure de plus le non-recouvrement des vesicules. Execute visualisation.edp