© *Jean-Baptiste APOUNG KAMGA <jean-baptiste.apoung@math.u-psud.fr>*

Equation de Stokes, mise en oeuvre avec **FreeFem++**

On se propose dans cette fiche d'associer deux éléments finis, P1-Lagrange et P2-Lagrange, dans la discrétisation d'un problème aux limites. Cette approche est couramment adoptée pour résoudre des problèmes à inconnues vectorielles. Nous nous intéressons ici au problème de Stokes.

Thème - 1 *Problème de Stokes*

Soit $\Omega \subset \mathbb{R}^2$ et $\mathbf{f}, \mathbf{u}_D$ deux fonctions de $\mathbb{R}^2 \to \mathbb{R}$. On considère le problème :

Chercher deux fonctions $\mathbf{u} : \mathbb{R}^2 \to \mathbb{R}^2, p : \mathbb{R}^2 \to \mathbb{R}$ telles que

$$\left\{ \begin{array}{rcll} -\Delta \mathbf{u} + \nabla p = & \mathbf{f} & \text{dans} & \Omega, \\ \nabla \cdot \mathbf{u} = & 0 & \text{dans} & \Omega, \\ \mathbf{u} = & \mathbf{u}_D & \text{sur} & \partial\Omega. \end{array} \right. \tag{1}$$

Où, pour $\mathbf{u} = (u_1, u_2)^t$, on a posé $\Delta \mathbf{u} = (\Delta u_1, \Delta u_2)^t$, $\nabla \cdot \mathbf{u} = \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}$.
Et pour toute fonction $v : \mathbb{R}^2 \to \mathbb{R}$, $\Delta v = \frac{\partial^2 v}{\partial^2 x} + \frac{\partial^2 v}{\partial^2 y}$.

Exercice-1 **:** **Formulation variationnelle**

On pose

$$H^1(\Omega) = \{v \in L^2(\Omega); \nabla v \in (L^2(\Omega))^2\}, \quad H^1_0(\Omega) = \{v \in H^1(\Omega); v = 0 \text{ sur } \partial\Omega\},$$
$$V = H^1_0(\Omega) \times H^1_0(\Omega), \quad M = \left\{ q \in L^2(\Omega); \int_\Omega q \, dx \, dy = 0 \right\},$$
$$a(\mathbf{u}, \mathbf{v}) = \int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v} \, dx \, dy \equiv \sum_{i=1}^{2} \int_\Omega \nabla u_i \cdot \nabla v_i \, dx, \quad b(\mathbf{v}, p) = -\int_\Omega p \, \nabla \cdot \mathbf{v} \, dx \, dy,$$
$$l(\mathbf{v}) = \int_\Omega \mathbf{f} \cdot \mathbf{v} \, dx \, dy.$$

On admet qu'il existe une *unique* fonction $\bar{\mathbf{u}}_D \in \left(H^1(\Omega)\right)^2$ dont la restriction sur $\partial\Omega$ coïncide avec $\mathbf{u}_D$.
(*C'est l'image de* $(\mathbf{u}_D)|_{\partial\Omega}$ *par le relèvement continu des traces. Les détails sortent du cadre du présent TP.*)

Montrer que la formulation variationnelle du problème (1) est alors donnée par :

$$\left\{ \begin{array}{l} \text{Chercher } \mathbf{u} \in \bar{\mathbf{u}}_D + V \text{ et } p \in M \text{ tels que} \\ \quad a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = l(v) \quad \forall \mathbf{v} \in V, \\ \quad b(\mathbf{u}, q) = 0 \qquad\qquad\quad \forall q \in M. \end{array} \right. \tag{2}$$

On admettra que ce problème possède une unique solution, dotée des régularités nécessaires, permettant son évaluation ponctuelle.

Thème - 2 *Discrétisation par éléments finis de Taylor-Hood*

Pour discrétiser le problème de Stokes, on va introduire deux espaces éléments finis, l'un pour la vitesse et l'autre pour la pression. L'association de ces espaces doit être judicieusement menée, car en effet, il faudra s'assurer que : à vitesse donnée, le problème en pression est résoluble. Le choix des éléments finis est ainsi soumis à une restriction qu'on appelle couramment *condition inf-sup ou BNB (Banach-Nečas-Babuška) discrète*. Plusieurs couples d'éléments finis ont franchi cet obstacle, nous nous intéressons ici à celui de Taylor-Hood, appelé aussi éléments finis P2/P1. Il consiste en l'utilisation des éléments finis P2-Lagrange dans la discrétisation de chaque composante de la vitesse et des éléments finis P1-Lagrange dans la discrétisation de la pression.

Exercice-1 : Discrétisation par éléments finis P2 / P1

On introduit τ_h un maillage conforme de Ω, formé des triangles.

Espaces et problème discret. On introduit $V_h \subset V$ et $M_h \subset M$ des sous-espaces vectoriels de dimension finie définis de la manière suivante :

$$X_h^r = \{p_h \in C^0(\Omega) : p_h|_T \in P_r(T)\ \forall T \in \tau_h\},$$

où $P_r(T)$ est l'espace vectoriel des polynômes de degré total r sur le triangle T, (X_h^r n'est rien d'autre que l'espace éléments finis Pr-Lagrange)

$$V_h = \{\mathbf{v}_h \in X_h^2 \times X_h^2 \text{ tel que } : \mathbf{v}_h = 0 \text{ sur } \partial\Omega\},$$

$$M_h = \{q_h \in X_h^1 \text{ tel que } \int_\Omega q_h\, dx = 0\}.$$

Le problème discret s'écrit alors

$$\left\{ \begin{array}{l} \text{Chercher } \mathbf{u}_h \in \bar{\mathbf{u}}_{hD} + V_h \text{ et } p_h \in M_h \text{ tels que} \\ a(\mathbf{u}_h, \mathbf{v}_h) + b(\mathbf{v}_h, p_h) = l(\mathbf{v}_h) \quad \forall \mathbf{v} \in V_h, \\ b(\mathbf{u}_h, q_h) = 0 \qquad\qquad\quad \forall q_h \in M_h, \end{array} \right. \tag{3}$$

où $\bar{\mathbf{u}}_{hD}$ est un interpolé de $\bar{\mathbf{u}}_D$ dans $X_h^2 \times X_h^2$.

1. On suppose que le maillage a N_{so} sommets, N_{ma} triangles, N_a arêtes, dont N_{ab} arêtes frontières. Donner en fonction de ces quantités les dimensions de $X_h^r, r = 1, 2,\ V_h$, et M_h.
2. On admettra que ce problème admet une unique solution. (*Il est important de s'en assurer*).

Ecrire un script avec l'outil **`FreeFem++`** pour résoudre ce problème.

Exercice-2 : Post-traitement. On affiche la solution et on calcule si possible les erreurs numériques.

1. **Représentation de la vitesse calculée**. Représenter graphiquement la vitesse.(*la vitesse est représentée par un vecteur en chaque degré de liberté*).
2. **Représentation de la pression calculée**. La pression est une grandeur scalaire, on la représente par ses isovaleurs (*comme dans le cas de la résolution du laplacien*).
3. **Exemple : cavité entraînée** On prend $\Omega =]0,1[\times]0,1[$, $f = (f_1, f_2)^t, u_D = (g_1, g_2)^t$, avec

$$\left\| \begin{array}{ll} f_1(x,y) = 1, & g_1(x,y) = 1 \text{ si } y = 1, 0 \text{ sinon}, \\ f_2(x,y) = 1, & g_2(x,y) = 0. \end{array} \right.$$

   Représenter graphique la vitesse et la pression solutions du problème de Stokes, sur le maillage du carrée unité
4. **Estimation de l'erreur numérique.** On rappelle que l'erreur dans une certaine norme $\|u - u_h\|$ entre la solution exacte u et la solution calculée u_h est majorée par

$$\|u - u_h\| \leq \|u - \pi_h(u)\| + \|\pi_h(u) - u_h\|.$$

   La première à droite est l'erreur d'interpolation ; c'est la meilleure qu'on puisse obtenir puisque selon son principe même, la méthode de Galerkin consiste à injecter "un interpolé (à coefficients inconnus)" u_h dans la formulation variationnelle. Il est clair que ce faisant, on ne peut espérer une erreur $\|u - u_h\|$ meilleure que $\|u - \pi_h(u)\|$. C'est à ce titre que l'on peut quelque fois se contenter du calcul de $\|\pi_h(u) - u_h\|$ et à le considérer comme erreur numérique.

   On considère les données suivantes : $\Omega =]0,1[\times]0,1[$, $f = (f_1, f_2)^t, u_D = (g_1, g_2)^t$, avec

$$\left\| \begin{array}{ll} f_1(x,y) = 2\sin(\pi(x+y)) + \frac{1}{\pi}\cos(\pi(x+y)), & g_1(x,y) = \frac{1}{\pi^2}\sin(\pi(x+y)), \\ f_2(x,y) = -2\sin(\pi(x+y)) + \frac{1}{\pi}\cos(\pi(x+y)), & g_2(x,y) = -\frac{1}{\pi^2}\sin(\pi(x+y)). \end{array} \right.$$

Pour lesquelles la solution exacte est

$$u_1(x,y) = \frac{1}{\pi^2}\sin(\pi(x+y)), \quad u_2(x,y) = \frac{1}{\pi^2}\sin(\pi(x+y)), \quad p(x,y) = \frac{1}{\pi^2}\sin(\pi(x+y)).$$

On se propose de calculer les erreurs en norme L^2. On veut pour ainsi dire évaluer

$$\|\pi_h^2(u_1) - u_{1h}\|_{L^2(\Omega)}, \quad \|\pi_h^2(u_2) - u_{2h}\|_{L^2(\Omega)}, \quad \|\pi_h^1(p) - p_h\|_{L^2(\Omega)},$$

où π_h^2 est l'opérateur d'interpolation $P2-$Lagrange, π_h^1 est l'opérateur d'interpolation P1-Lagrange, (u_{1h}, u_{2h}) sont les composantes de la vitesse calculée, et p_h est la pression calculée.

(a) En utilisant ces fonctions, calculer en norme L^2 les erreurs numériques commises dans la discrétisation $P2/P1$ de l'équation de Stokes.

(b) Faites varier le maillage en le prenant de plus en plus fin, et représenter les courbes de variation des erreurs L^2 en fonction du pas h du maillage. On rappelle que $h = \max_{T\in\tau_h} h_T$ où h_T est le diamètre du triangle T. du maillage. Quel ordre de convergence observez-vous, pour la vitesse ? pour la pression ?.

Thème - 3 *Applications avancées*

Considérer la formulation matricielle du problème de Stokes discrétisé par les éléments finis $P2/P1$.

Exercice-1 **:** Proposer et justifier un algorithme itératif pour sa résolution.

Exercice-2 **:** Cherchez-en un préconditionneur qui soit scalable dans le sens où le nombre d'itérations de l'algorithme itératif soit indépendant de la finesse du maillage. Proposer une justification de ce choix.

Exercice-3 **:** Implémenter ce préconditionneur avec FreeFem++. On se renseignera sur la commande *LinearCG*.

Guides pour la réponse aux questions

Illustration de la condensation

On peut résoudre le problème de Stokes par une méthode itérative, dite Uzawa gradient conjugué.

Cela revient à condensé le problème de Stokes par rapport à la variable pression et de résoudre le problème résultant par un algorithme de gradient conjugué.

Plus précisément, on part du problème (4),

$$\begin{bmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ B_1^t & B_1^t & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ P \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ 0 \end{bmatrix}. \tag{4}$$

que l'on transforme en les problèmes (5)-(6)

$$\begin{cases} U_1 = A_1^{-1}(F_1 - B_1\, P), \\ U_2 = A_2^{-1}(F_2 - B_2\, P), \end{cases} \tag{5}$$

$$(B_1^t A_1^{-1} B_1 + B_2^t A_2^{-1} B_2)P = B_1^t A_1^{-1} F_1 + B_2^t A_2^{-1} F_2. \tag{6}$$

On résout alors le problème (6), par un algorithme itératif du type gradient conjugué, en prenant bien soin d'initialiser $P = 0$, afin d'obtenir une pression à moyenne nulle. Puis on détermine la vitesse grâce à la formule (5).

Remarque *:*

— *Il est déconseiller en pratique de construire explicitement la matrice et le vecteur*

$$A = B_1^t A_1^{-1} B_1 + B_2^t A_2^{-1} B_2 \quad \textit{et} \quad F = B_1^t A_1^{-1} F_1 + B_2^t A_2^{-1} F_2.$$

En effet dans les algorithmes itératifs seul les produits matrices-vecteurs et les produits scalaires sont utilisés. On calculer le produit Ax avec A fournie ci-dessus, sans nécessairement calculer explicitement la matrice A.

— *Afin de réduire le nombre d'itérations, on est conseiller de recourir à un préconditionneur, c'est-à-dire à la recherche d'une matrice (ou opérateur) M, facilement inversible, telle que : $cond(M^{-1}A) \leq cond(A)$ où $cond(A)$ désigne le conditionnement de la matrice A.*

Illustration d'utilisation de LinearCG

On considère le problème elliptique de la fiche précédente

Code Listing 1 – Problème elliptique avec LinearCG

```
//@Jean-Baptiste APOUNG
//Definition du maillage
mesh Th=square(10,10);
//Contrction de l'espace elements finis
fespace Xh(Th,P1);
//Inconnue et function test
Xh u,v;
//variableq pour linear CG
Xh F, /*Vecteur qui contiendra le second membre int2d(Th)(  f*v ) */
   bc, /* Vecteur qui contiendra la condition aux limites*/
   ppp; /*Variable éutilise dans la édfinition de la fonction r ->  A*x - b */

func uD = x*x+y*y;
func f  = -4;
//C'est ici que la énouveaut apparait
//On édfinit une forme bilineaire qui nous permetra d'extraire la matrice
// et les conditions aux limites
varf aA(u,v)= int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v) )
                    +  on(1,2,3,4,u=uD);
//Idem pour le second membre
varf aF(u,v)= int2d(Th)(  f*v );

//Extraction de la matrice avec choix du solveur pour son inversion
matrix A= aA(Xh,Xh,solver=CG);
//Extraction de la condition aux limites
bc[] = aA(0,Xh);
//éReprsentation graphique de la condition aux limites
plot(bc, value=true, wait=1);

//Recuperation du second membre sous forme de vecteur
 F[] = aF(0,Xh);

 //Definition du èproblme lineaire àé rsoudre:
 // i.e  on érsout residus(x) = 0
func real[int] residus(real[int] & pp)
{
  //on doit construire et retourner
  // b - A * pp
  int verb=verbosity;
      verbosity=0;  //éprsvention des affichages inutiles
   ppp[]= F[] + bc[];
   ppp[] *=-1.;
   ppp[] += A * pp ;
    verbosity=verb; //restoration des affichage
   pp = ppp[];
   return pp;
};

//Construction du éprconditionneur
Xh ppm;
func real[int] precM(real[int] & pp)
{

   ppm[] = pp;
   pp = A^-1*ppm[];  //Ceci doit être assoupli
   return pp;
};
//Initialisation de gradient éconjugu
 u=0;
//Appel du gradient éconjugu
```

```
 LinearCG(residus,u[],precon = precM,eps=1.e-6,nbiter=50);
//LinearGMRES(residus,u[],nbiter=50,eps=1.e-6);

//Dessin de la solution
plot(u,wait=1,value=true);
```

Script d'aide pour Stokes dans FreeFem++

Code Listing 2 – Script FreeFem++ pour Stokes avec cavité entraînée

```
//@ Jean-Baptiste APOUNG
//Maillage du domaine
mesh Th=square(10,10);
//Expace d'approximation de chaque composante de la vitess
fespace Vh(Th,P2);
//Expace d'approximation de la pression
fespace Wh(Th,P1);

Vh u1,u2; //composantes de la vitesse
Vh v1,v2; //fonctions tests
Wh  p;    //pression à calculer
Wh  q;    //fonction test

//èparamtre de éLam
real  nu=1;

//éEntres du èproblme pour la écavit îéentrane
func g=(x)*(1-x)*4;
func f1 =1;
func f2 =1;

//èProblme variationnel
problem Stokes ([u1,u2,p],[v1,v2,q],solver=Crout) =
    int2d(Th)(
              nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
             +         dx(u2)*dx(v2) + dy(u2)*dy(v2) )
             + p*q*(0.000001) /* ne pas oublier ce terme */
             - p*dx(v1) - p*dy(v2)
             - dx(u1)*q - dy(u2)*q
            )
  + int2d(Th) ( -f1*v1 - f2*v2 )
  + on(3,u1=g,u2=0)
  + on(1,2,4,u1=0,u2=0);

//éRsolution
Stokes;
//postprocessing
plot(coef=0.2,cmm=" [u1,u2] et p  ",p,[u1,u2],ps="StokesP2P1.eps",value=1,wait=1);
```

Code Listing 3 – Script FreeFem++ pour Stokes Uzawa

```
//@ Jean-Baptiste APOUNG
assert(version>1.18);
real s0=clock();
mesh Th=square(20,20);
fespace Xh(Th,P2),Mh(Th,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;

varf bx(u1,q) = int2d(Th)( (dx(u1)*q));
varf by(u1,q) = int2d(Th)( (dy(u1)*q));
varf mp(p,q)= int2d(Th)( p*q );
varf a(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                     +  on(1,2,4,u1=0)  +  on(3,u1=1) ;

Xh bc1; bc1[] = a(0,Xh);
Xh b;
```

```
//matrix A= a(Xh,Xh,solver=CG);
matrix A= a(Xh,Xh,solver=UMFPACK);
matrix Bx= bx(Xh,Mh);
matrix By= by(Xh,Mh);
matrix M= mp(Mh,Mh,solver=UMFPACK);

Xh bcx=1,bcy=0;

func real[int] divup(real[int] & pp)
{
  int verb=verbosity;
   verbosity=0;
   b[]  = Bx'*pp; b[] += bc1[] .*bcx[];
   u1[] = A^-1*b[];
   b[]  = By'*pp; b[] += bc1[] .*bcy[];
   u2[] = A^-1*b[];
   ppp[] =   Bx*u1[];
   ppp[] +=  By*u2[];
   verbosity=verb;
   return ppp[] ;
};


//Pressure Mass preconditionner
func real[int] precM(real[int] & pp)
{
   ppp[] = M^-1*pp;
   return ppp[];
};

p=0;q=0;u1=0;v1=0;

LinearCG(divup,p[],q[],eps=1.e-6,nbiter=50);

//LinearCG(divup,p[],precon=precM,eps=1.e-6,nbiter=50);

divup(p[]);

plot([u1,u2],p,wait=1,value=true,coef=0.1);
```