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

Stokes Equations. Implement in **FreeFem++**

Whe wish in this sequel to combine two finite elements, namely P1-Lagrange and P2-Lagrange, in the discretization of a boundary limits problem. This approach is often used to solve PDE with vectorial solution. We consider here the Stokes problem.

Thème - 1 *Stokes Problem*

Let $\Omega \subset \mathbb{R}^2$ be open, bounded and Lipschitz domain. Let $\mathbf{f}, \mathbf{u}_D$ be two functions $\mathbb{R}^2 \to \mathbb{R}$. We consider the following problem:

Find two functions $\mathbf{u} : \mathbb{R}^2 \to \mathbb{R}^2, p : \mathbb{R}^2 \to \mathbb{R}$ such that

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

Where, for $\mathbf{u} = (u_1, u_2)^t$, we have defined $\Delta \mathbf{u} = (\Delta u_1, \Delta u_2)^t$, $\nabla \cdot \mathbf{u} = \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}$.
And for any function $v : \mathbb{R}^2 \to \mathbb{R}$, $\Delta v = \frac{\partial^2 v}{\partial^2 x} + \frac{\partial^2 v}{\partial^2 y}$.

Exercice-1 : Variational Formulation

Let

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

We assume that there exists a *unique* function $\bar{\mathbf{u}}_D \in \left(H^1(\Omega)\right)^2$ whose restriction on $\partial\Omega$ coincides with $\mathbf{u}_D$. (*It is the image of* $(\mathbf{u}_D)|_{\partial\Omega}$ *by the boundary continuous lifting operator, whose details are beyond the scope of the present class.*)

Show that the variational formulation of problem (1) is given by:

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

We assume that this problem admits a unique solution with sufficient regularity such that it can be evaluated point-wise.

To discretize the Stokes problem, we are going to introduce two finite element spaces, one for the velocity and the other for the pressure. The association of the these two finite element spaces must be done judiciously since it should be make sure that for a given velocity, the problem over the pressure is well-posed. The selection of these two finite element spaces is thus subject to a constraint known as the *discrete inf-sup or BNB (Banach-Nečas-Babuška) condition*. Several pairs of finite element has overcome this constraint. Let us mention as example, the Taylor-Hood finite element, also referred to as P2/P1 finite elements. It consists of approximating the velocity by P2-Lagrange finite element and the pressure by P1-Lagrange finite element.

Exercice-1 **:** **Discretization with P2/P1 finite elements**

Let τ_h be a conformal mesh of Ω, made of triangles.

Finite elements spaces and discrete problem. Let $V_h \subset V$ and $M_h \subset M$ the two vector space of finite dimension defined as follows:

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

where $P_r(T)$ is the vector space of polynomials of total degree r on the triangle T, (recall that X_h^r is the Pr-Lagrange finite elements space),

$$V_h = \{\mathbf{v}_h \in X_h^2 \times X_h^2 \text{ such that } : \mathbf{v}_h = 0 \text{ on } \partial\Omega\},$$

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

The discrete problem reads:

$$\left\{ \begin{array}{ll} \text{Seek } \mathbf{u}_h \in \bar{\mathbf{u}}_{hD} + V_h \text{ and } p_h \in M_h \text{ such that} & \\ a(\mathbf{u}_h, \mathbf{v}_h) + b(\mathbf{v}_h, p_h) = l(\mathbf{v}_h) & \forall \mathbf{v} \in V_h, \\ b(\mathbf{u}_h, q_h) = 0 & \forall q_h \in M_h, \end{array} \right. \tag{3}$$

where $\bar{\mathbf{u}}_{hD}$ is the interpolate of $\bar{\mathbf{u}}_D$ in $X_h^2 \times X_h^2$.

1. Assuming that the mesh is made of N_{so} vertices, N_{ma} triangles, N_a edges with N_{ab} boundary edges. Gives in terms of those quantities the dimension of the spaces $X_h^r, r = 1, 2,\ V_h$, and M_h.
2. We admit that this discrete problem has a unique solution. (*It is important to check that*).

   Write a script with **`FreeFem++`** to solve the problem.

Exercice-2 **:** **Post-processing.** For the post-processing one can either display the solution or compute the numerical errors.

1. **Graphical display of the computed velocity**. Display in a graphic the computed velocity. (*The velocity can be display by plotting a vector at each position of the degree of freedom*).
2. **Graphical display of the pressure**. The pressure is a scalar quantity, we ca just display its isovalues just as for the Laplace problem .
3. **Example: driven cavity** Take $\Omega =]0,1[\times]0,1[$, $f = (f_1, f_2)^t, u_D = (g_1, g_2)^t$, with

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

   Display the the computed velocity and pressure on the unit square.

4. **Numerical estimation of the error.** Let us recall that an error between the exact solution u and the computed one u_h in a given norm, namely $\|u - u_h\|$ is bounded by

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

The first term in the right is the interpolation error. It is the best we can obtain, since in its principle, the Galerkin method consists in plugging "an interpolated (with unknown coefficients)" (i.e u_h) in the variational formulation. It is obvious that doing this one cannot expect an error $\|u-u_h\|$ which is better than $\|u - \pi_h(u)\|$. It is for this reason that one can just sometimes compute the quantity $\|\pi_h(u) - u_h\|$ and consider it as the numerical error.

Let us consider the following inputs for the Stokes problem: $\Omega =]0,1[\times]0,1[$, $f = (f_1, f_2)^t, u_D = (g_1, g_2)^t$, with

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

for which the exact solution is given by

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

We aim at computing the errors in the L^2 norm. We wish to evaluate

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

where π_h^2 is the $P2-$Lagrange interpolation operator, π_h^1 that of of the P1-Lagrange finite elements, (u_{1h}, u_{2h}) are the components of the computed velocity and p_h is the computed pressure.

(a) Using these functions, compute the numerical errors in the discretization of the stokes equation using the $P2/P1$ Finite elements.

(b) By making the mesh a more and more finer, display the evolution of the L^2 error with respect to the mesh size $h = \max_{T\in\tau_h} h_T$, where h_T denotes the diameter of the triangle T.
What is the observed convergence order for the velocity and for the pressure?

Thème - 3 *Advanced applications*

Consider the matrix formulation of the $P2/P1$ discretization of the Stokes problem.

Exercice-1 **:** Propose et justify an iterative algorithm to solve it.

Exercice-2 **:** Seek a preconditioner which is scalable in the sens that the number of iterations of the algorithm is independent of the mesh size. Try to justify your choice.

Exercice-3 **:** Implement your preconditioner in **`FreeFem++`**. You should check for the **`FreeFem++`** function *LinearCG*.

Tips to answer to the questions

Illustration of the condensation

One can solve the matricial formulation of the Stokes problem by an iterative, called Uzawa conjugate gradient.

This consists in condensing the discrete Stokes problem over the pressure variable and to solve the resulting problem using the conjugate gradient method.

More precisely, one goes from the problem (4),

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

and transform it into problems (5)-(6)

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

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

We then solve the problem (6), by an iterative algorithm of conjugate gradient type, ensuring that the starting value is $P = 0$, in order to obtain a pressure with zero mean value. Then we compute the velocity using formula (5).

Remark:

- *It is advised in practice, not to compute explicitly the matrix and the vector :*

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

  Because not only this matrix can be full, but also in much iterative algorithms only matrix-vector products and scalar products are required. So we only compute the matrix-vector product Ax with the given A without computing the matrix A explicitly.

- *In order to reduce the number of iterations, it is advised to use a preconditioner. This consists in seeking a matrix (or operator) M, easy to invert such that : $cond(M^{-1}A) \leq cond(A)$. Where $cond(A)$ is the condition number of the matrix A.*

Illustration of the usage of LinearCG

Let us consider the elliptic problem of the previous file.

Listing 1: Elliptic problem with 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);
```

Help script for Stokes problem with FreeFem++

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);
```

Listing 3: Script FreeFem++ for 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);
```