Numerical Methods For PDE : Assignement 6

Leo-Paul BAUDET

1er janvier 2024

Table des matières

1 Navier-Stokes equation

1.1 Strong form

In that assignement, we want to solve the steady version of the Navier Stokes equation :

$$\begin{cases} \nu\nabla^2 v + (v.\nabla)v + \nabla p = f \text{ on } \Omega \\ \nabla.v = 0 \text{ on } \Omega \\ v = v_D \text{ on } \partial\Omega \end{cases} \tag{1}$$

1.2 Weak form

It can be shown that the associated weak form is the following :

Find p,v such that v=v_d on $\partial\Omega$ and

$$\begin{cases} a(w,v) + c(w,v,v) - b(w,p) = l(w,f) \\ b(v,q) = 0 \end{cases} \tag{2}$$

with :

$$a(w,v) = \int_\Omega \nabla w : (\nu\nabla v) d\Omega \tag{3}$$

$$l(w,f) = \int_\Omega w.f d\Omega \tag{4}$$

$$b(v,q) = \int_\Omega q\nabla.v d\Omega \tag{5}$$

$$c(w,v,a) = \int_\Omega w.(a.\nabla)v d\Omega \tag{6}$$

1.3 Discretization of the weak form

As for the Stokes equation, the discretization of the weak form yields a system of non-linear equations :

$$\begin{pmatrix} K + C(v) & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} v \\ p \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{7}$$

The difference is that the matrix of the system depends also on the speed, which is an unknow. In order to solve it we need to implement an iterative method(Picard method, Newton-Raphson method). In the following we have choosen the the Picard method

1.3.1 Picard method

Let's assume that we have the following system : A(x)x=b(x)

The algorithm to solve it is the following :

— Given an initial guess x_0

— Compute the approximations x^{k+1} until convergence (the criteria will be defined after)

$$A(x^k)x^{k+1} = b(x^k) \tag{8}$$

For the Navier-Stokes equation we get :

$$\begin{pmatrix} K + C(v^k) & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} v^{k+1} \\ p^{k+1} \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{9}$$

— First of all, we clearly that we need, in order to apply this method to recompute the matrix C at each iteration.
— The criteria of convergence choosen is the following : $||u_{k+1} - u_k|| < 10^{-3}||u_k||$

2 Numerical solving of the Navier-Stokes equation

2.1 Mesh and element

It can be shown that we need, in order get a stable method (LBB-stability), a mesh for the the speed which has elements one degree higher than those for pressure (if we don't want to use stabilization method. This is why we are going to use the Taylor-Hood elements (rectangular elements for each but with different degree) :

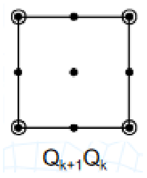

FIGURE 1 – Taylor-Hood Element

The points are the nodes for the speed, thr circle are the nodes for the pressure.

2.2 Strategy of solving

Given an initial viscosity (sufficiently high), we are going to apply to apply the Picard's method. As an initial guess ($x_0 = (0, 0)$), we get the solution of the Stokes equation (because $C(0) = 0$). Then, we this previous solution, we decrease the viscosity, and take as x_0 the previous solution. Indeed, we need to be not far enough of the solution in order that the Picard method converge.

For low Reynold number (high viscosity), the solution of the Stokes equation is close to the solution of the Navier Stokes equation. This is why we take as initial guess $x_0 = (0, 0)$.

Then, we reitirate this process. However, for viscosity under 0.01, the Picard method will not converge anymore. To solve that kind of problem, we need to implement the Newton-Raphson method.

2.3 Explanation of the code

```
degree=2; %Q2Q1 Taylor-Hood element
[X,T,Xp,Tp]=CreateMeshAdaptedCavityQ2Q1(40);
figure(1), aux=plotMesh(X,T); axis equal, title('Mesh for velocity')
hold on, plot(X(:,1),X(:,2),'o'), hold off
figure(2), aux=plotMesh(Xp,Tp); axis equal , title('Mesh for pressure')
hold on, plot(Xp(:,1),Xp(:,2),'o'), hold off
nOfNodes = size(X,1); nOfNodesp = size(Xp,1);

%Reference element
referenceElement = createReferenceElementStokesQua(degree);
```

```
11      viscosity=linspace(0.1,0.02,10);
12      u0=zeros(2*nOfNodes,1);
13
14      for i=1:length(viscosity)
15          disp(viscosity(i));
16          [u,p,j,error]=solve_NS(viscosity(i),X,T,Xp,Tp,referenceElement,u0);
17          disp(j);
18          disp(error);
19          u0=u;
20
21          ux=u(1:nOfNodes);
22          uy=u(nOfNodes+1:2*nOfNodes);
23          Tboundary=connectivityMatrixBoundary(T,referenceElement);
24          phi=computeStreamFunction(ux,uy,X,T,Tboundary,referenceElement);
```

Line 2, we take Taylor-Hood elements for the speed and the presure

Line 11, we take several viscosity for wich we are goin to compute the solution of the Navier Stokes equation.

Line 12, we take as initial guess $x_0 = 0$.

Line 17 for each viscosity we compute the solution and take as initial guess for the Picard method the previous solution (line 20)

2.3.1 *Solve_NS* function

The *solve_NS* function which is used in the previous algorithm is doing the Picard method. It returns the solution, the error and the number of iteration (in order to see if the method converge or not). It takes as parameters a viscosity, an initial guess and a mesh for the speed and for the pressure.

```
function [u,p,i,error]=solve_NS(viscosity,X,T,Xp,Tp,referenceElement,u0)
%__Definition of boundary conditions
    boundaryValue=@boundaryValueFunctionCavity; sourceFunction=@sourceCavity;
    nOfNodes = size(X,1); nOfNodesp = size(Xp,1);
    x = X(:,1); y = X(:,2); tol=1.e-10;
    nodesCCD = find(abs(x)<tol|abs(x-1)<tol|abs(y)<tol|abs(y-1)<tol);
    XnodesCCD = X(nodesCCD,:);  %coordinates of the nodes on the boundary
    coefficientsCCD = [nodesCCD; nodesCCD+nOfNodes];
    uCCD = boundaryValue(XnodesCCD); %boundary value %returns a (2*nOfNodes x 1) vector

    %__System computation
    [K,G,f]=computeSystemStokes(X,T,Xp,Tp,referenceElement,sourceFunction,viscosity);
    C=computeNSconvectionMatrix(u0,X,T,referenceElement);

    nK=size(K,1); A = spalloc(nK+nOfNodesp,nK+nOfNodesp,nnz(K)+2*nnz(G));
    A(1:nK,1:nK)=K+C; A(1:nK,nK+1:end)=G;
    A(nK+1:end,1:nK)=G';
    b = [f; zeros(nOfNodesp,1)];

    pend=0;
    prescribedValues =[uCCD;pend]; prescribedDOF = [coefficientsCCD;size(A,1)];

    %__Imposition of Dirichlet boundary conditions (system reduction)
    unknowns= setdiff(1:2*size(X,1)+nOfNodesp,prescribedDOF);

    b = b(unknowns)-A(unknowns,prescribedDOF)*prescribedValues;
    A=A(unknowns,unknowns);
    %__System solution

    x0=A\b;

    aux = zeros(2*size(X,1)+nOfNodesp,1);
```

```
33      aux(unknowns) = x0;
34      aux(prescribedDOF) = prescribedValues;
35      u1= aux(1:2*nOfNodes); p1 = aux(2*nOfNodes+1:end);
36      sol1=[u1;p1];
37
38      max_it=100;
39      i=0;
40      error=norm(u1-u0)/norm(u0);
41
42      while (norm(u1-u0)>10^-3*norm(u0) && i<max_it)
43          % disp(norm(u1-u0));
44          %disp(i);
45
46          sol0=sol1;
47          u0=sol0(1:2*nOfNodes);
48          i=i+1;
49
50          C=computeNSconvectionMatrix(u0,X,T,referenceElement);
51
52          nK=size(K,1); A = spalloc(nK+nOfNodesp,nK+nOfNodesp,nnz(K)+2*nnz(G));
53          A(1:nK,1:nK)=K+C; A(1:nK,nK+1:end)=G;
54          A(nK+1:end,1:nK)=G';
55          b = [f; zeros(nOfNodesp,1)];
56
57          pend=0;
58          prescribedValues =[uCCD;pend]; prescribedDOF = [coefficientsCCD;size(A,1)];
59
60          %__Imposition of Dirichlet boundary conditions (system reduction)
61          unknowns= setdiff(1:2*size(X,1)+nOfNodesp,prescribedDOF);
62
63          b = b(unknowns)-A(unknowns,prescribedDOF)*prescribedValues;
64          A=A(unknowns,unknowns);
65
66          x1=A\b;
67          %x1=fsolve(A*x0'-b);
68
69          aux = zeros(2*size(X,1)+nOfNodesp,1);
70          aux(unknowns) = x1;
71          aux(prescribedDOF) = prescribedValues;
72          u1 = aux(1:2*nOfNodes); p1 = aux(2*nOfNodes+1:end);
73
74          sol1=[u1;p1];
75          error=norm(u1-u0)/norm(u0);
76      end
77
78      sol=sol1;
79      u=sol(1:2*nOfNodes);
80      p=sol(2*nOfNodes:end);
```

Line 3-9, we define the boundary conditions

Line 12, we compute the matrix of the Stokes equation and line 13 the matrix C(v), given an initial guess.

Line 15-18, we build the global matrix of the system and the second member

Line 20-27, we impose Dirichlet condition

Line 30, we solve the system

Line 31-36, we get the solution of the equation. Then we reitirate (line 42) this process, until the method is converging (cyrtria explained before).

Remarks :

- At each iteration, we need to reimpose Dirichlet condition (line 57-64)
- Because only C depends on the speed, we just need no recompute C at each iteration.

2.4 Results

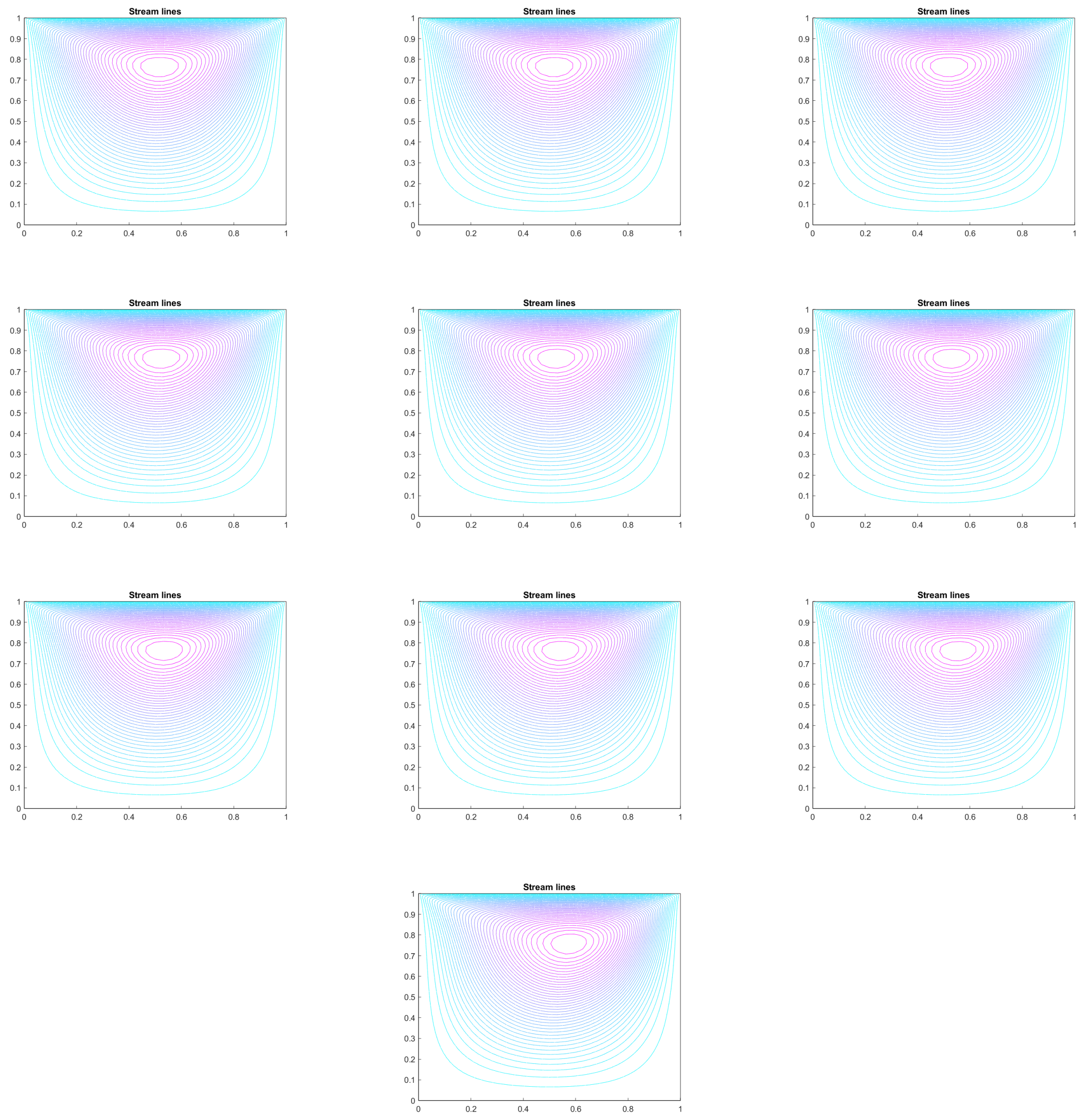

FIGURE 2 – Solutions for decreasing viscosity (here in case of the following animation does not work

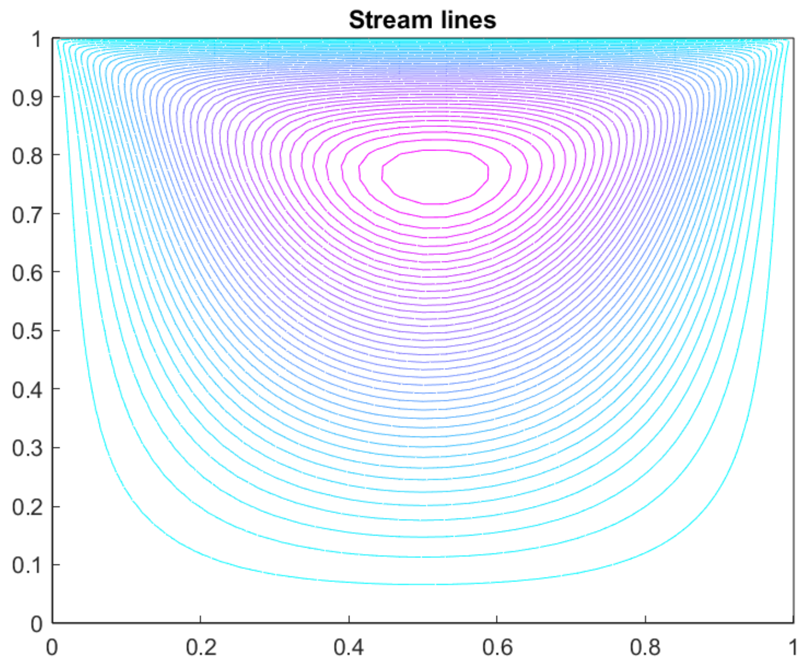

Figure 1: Solution from the higher viscosity to the lower

In order to see correctly the animation, open it in adobe reader

A Link to the code

All the code is available here
`https://github.com/leopaulbis/NMPDE_assignement_6`