Cavity flow

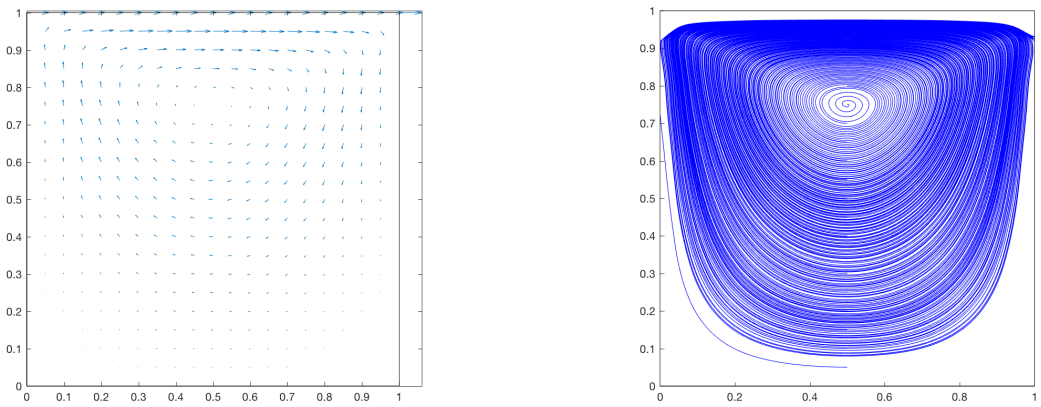
Shushu Qin

April 2, 2019

1 Stokes problem of cavity flow

1.1 Standard Galerkin

The streamlines and velocity vectors of the cavity flow problem are shown in Figure.1. This example has become a standard benchmark test for incompressible flows. The boundary conditions are indicated in Figure 2. As shown in the figures, we consider the plane flow of an isothermal fluid in a square lid-driven cavity. The upper side of the cavity moves in its own plane at unit speed, while the other sides are fixed. In this section, we simulate the stokes problem of the cavity flow with $\nu = 1$. First, we solve the lid-driven cavity for the stokes problem and the standard Galerkin formulation. The cavity is discretized with a uniform mesh of 10×10 T1T1 elements, 10×10 T2T1 elements, 10×10 Q1Q1 elements and 10×10 Q2Q1 elements. T: triangle; Q: quadrilateral. The detailed derivation of the linear system can be found at the end of the report as auxiliary files.



(a) (b)

Figure 1: Stokes cavity flow

Figure 3 and Figure 4 show that the solution using Q1Q1 and T1T1 elements is not LBB stable while Figure 5 and Figure 6 show Q2Q1 and T2T1 are LBB complicant elements. Both Q2Q1 and T2T1 elements present oscillations which are more pronounced in the corners.

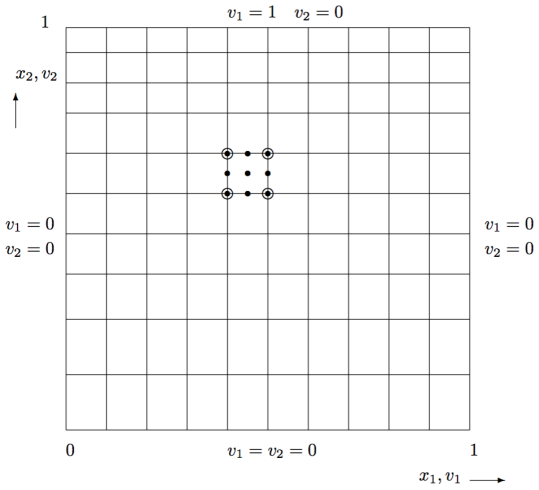
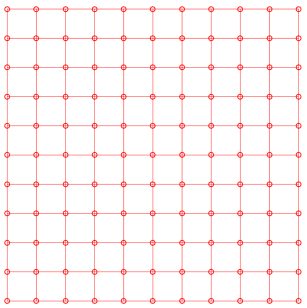
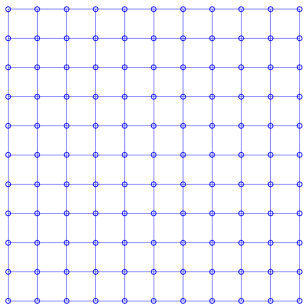


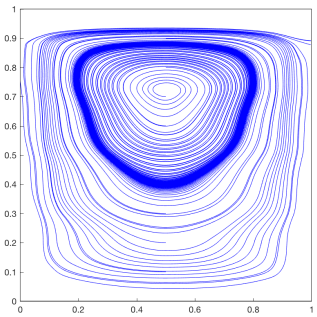
Figure 2: Boundary condition



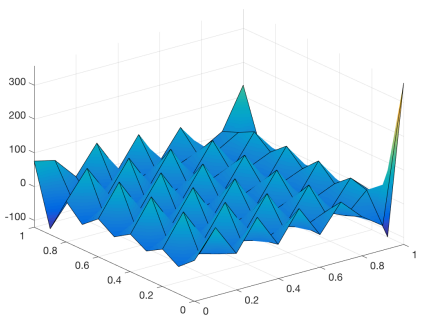
(a) Pressure nodes



(b) Velocity nodes

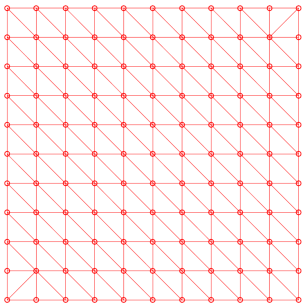


(c) Velocity streamline

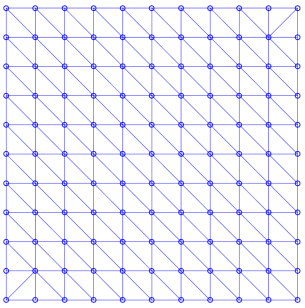


(d) Pressure field

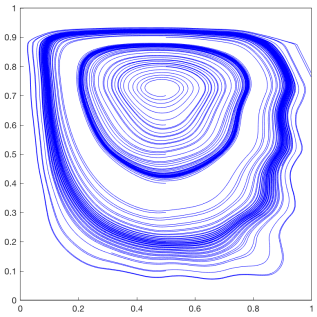
Figure 3: Q1Q1



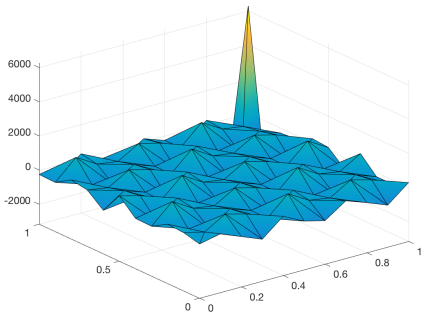
(a) Pressure nodes



(b) Velocity nodes

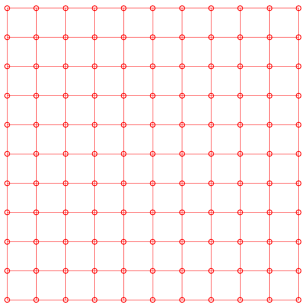


(c) Velocity streamline

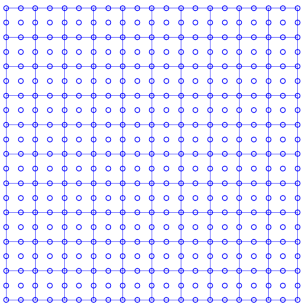


(d) Pressure field

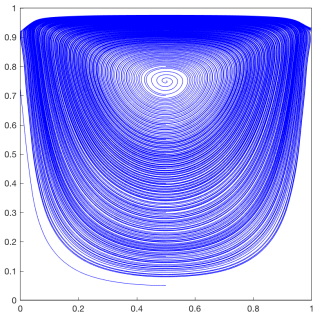
Figure 4: T1T1



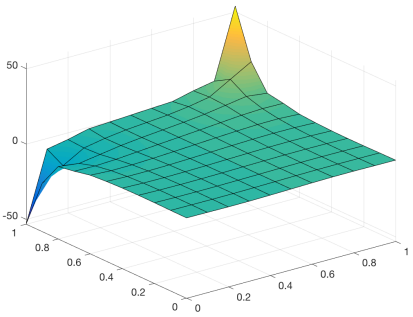
(a) Pressure nodes



(b) Velocity nodes

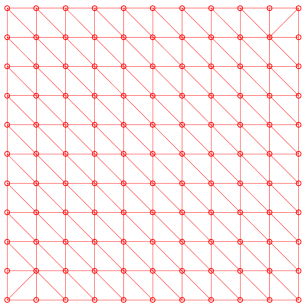


(c) Velocity streamline

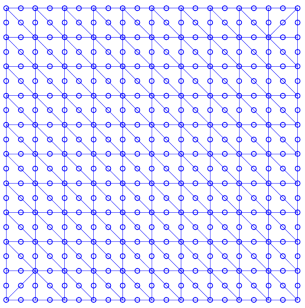


(d) Pressure field

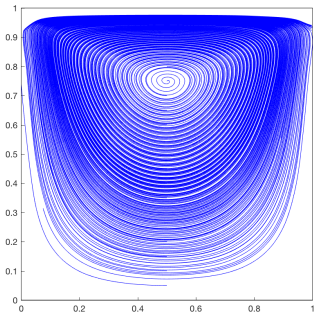
Figure 5: Q2Q1



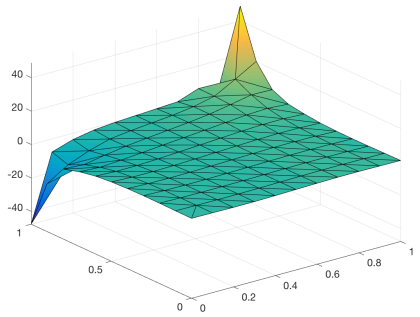
(a) Pressure nodes



(b) Velocity nodes



(c) Velocity streamline



(d) Pressure field

Figure 6: T2T1

1.2 Stabilization

In recent years, the efforts of the researchers in the area of mixed methods have been directed towards circumventing the LBB condition, thus opening the way to the use of velocity/pressure pairs which are not convergent in the standard Galerkin formulation. The basic idea behind the stabilization procedures is to render the matrix governing the velocity and pressure fields strictly positive-definite. This can, for instance, be accomplished through a modification of the weak form of the incompressibility condition, such as to render non-zero the diagonal term resulting from the incompressibility condition.
Several procedures were actually developed for stabilizing incompressible flow problems formulated in the primitive variables. Such procedures were essentially inspired from the stabilization techniques, such as SUPG and Galerkin/Least-squares (GLS).
The detailed derivation of the linear system can be found at the end of the report as auxiliary files. In the case of linear elements the GLS modification does not affect the weak form of the momentum equation. This is because the terms involving the second derivatives of the weighting function w do vanish in case of linear local approximations. The GLS matrix formulation then reduces to the following system of equations:

$$\begin{bmatrix} \mathbf{K} & \mathbf{G}^T \\ \mathbf{G} & \mathbf{L} \end{bmatrix} \begin{Bmatrix} \mathbf{v} \\ \mathbf{p} \end{Bmatrix} = \begin{Bmatrix} \mathbf{f} \\ \mathbf{f}_q \end{Bmatrix} \tag{1}$$

The elemental matrices at each loop on gaussian points are calculated as:

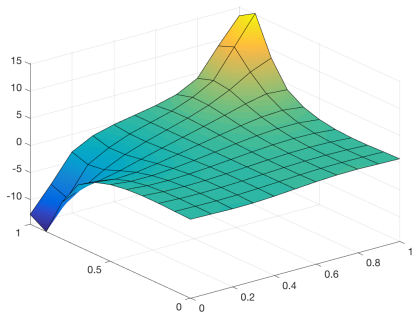
```
1     N_ig      = N(ig,:);
```

```
    Nxi_ig  = Nxi(ig,:);
    Neta_ig = Neta(ig,:);
    NP_ig = NP(ig,:);
    Jacob = [
        Nxi_ig(1:ngeom)*(Xe(:,1)) Nxi_ig(1:ngeom)*(Xe(:,2))
        Neta_ig(1:ngeom)*(Xe(:,1))  Neta_ig(1:ngeom)*(Xe(:,2))
        ];
    dvolu = wgp(ig)*det(Jacob);
    res = Jacob\[Nxi_ig;Neta_ig];
    nx = res(1,:);
    ny = res(2,:);

  Ngp = [reshape([1;0]*N_ig,1,nedofV); reshape([0;1]*N_ig,1,nedofV)];
    % Gradient
    Nx = [reshape([1;0]*nx,1,nedofV); reshape([0;1]*nx,1,nedofV)];
    Ny = [reshape([1;0]*ny,1,nedofV); reshape([0;1]*ny,1,nedofV)];
    % Divergence
    dN = reshape(res,1,nedofV);

    Ke = Ke + (Nx'*Nx+Ny'*Ny)*dvolu;
    Ge = Ge - NP_ig'*dN*dvolu;
    Le = Le - tau1*(nx'*nx+ny'*ny)*dvolu;
    x_ig = N_ig(1:ngeom)*Xe;
    f_igaus = SourceTerm(x_ig);
    fe = fe + Ngp'*f_igaus*dvolu;
    fqe = fqe -tau1* res'*f_igaus*dvolu;
```

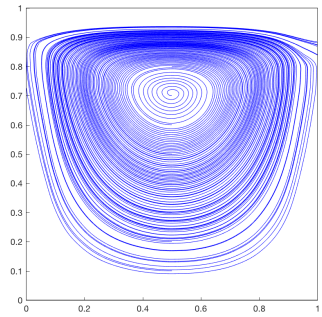
And in the main code, the left and right hand side is defined as

```
A = [Kred    Gred';
 Gred    L];
b = [fred; fqred];
```

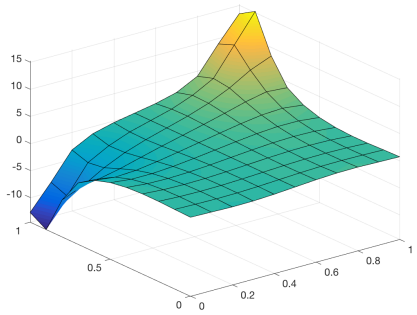
Figure 7 shows that with GLS stabilization, the stokes problem with equal order interpolations, which are unstable in the Galerkin formulation, now becomes stable. This is the case for quadrilateral element with continuous piecewise bilinear interpolations and for the linear three-node triangle.



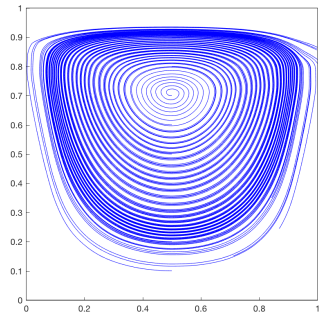
(a) Stabilized Q1Q1 pressure



(b) Stabilized T1T1 pressure



(c) Stabilized Q1Q1 velocity streamline



(d) Stabilized T1T1 velocity streamline

Figure 7: Stabilization of stokes problem

2 Incompressible Navier-Stokes problem

In the this section, we apply Newton-Raphson method to solve the incompressible Navier-Stokes problem.
The matrix form is

$$\begin{bmatrix} \mathbf{K}+\mathbf{C}(\boldsymbol{v}) & \mathbf{G}^T \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{v} \\ \mathbf{p} \end{Bmatrix} = \begin{Bmatrix} \mathbf{f} \\ \mathbf{h} \end{Bmatrix} \tag{2}$$

The residual is

$$\boldsymbol{r} = \begin{bmatrix} (\mathbf{K}+\mathbf{C}(\boldsymbol{v}))\mathbf{u}+\mathbf{G}^T\mathbf{p}-\mathbf{f} \\ \mathbf{G}\mathbf{u}-\mathbf{h} \end{bmatrix} \tag{3}$$

The Jacobian matrix of the global problem is:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \boldsymbol{r}_1}{\partial \boldsymbol{v}} & \mathbf{G}^T \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \tag{4}$$

where $\frac{\partial \boldsymbol{r}_1}{\partial \boldsymbol{v}} = \mathbf{K}+\mathbf{C}(\boldsymbol{v})+\frac{\partial \mathbf{C}}{\partial \boldsymbol{v}}\boldsymbol{v}$ The Newton Raphson method is firstly calculate $\triangle \boldsymbol{u}^{r+1}, \triangle \boldsymbol{p}^{r+1}$ for the following equation

$$\mathbf{J}^r \begin{Bmatrix} \triangle \mathbf{u}^{r+1} \\ \triangle \mathbf{p}^{r+1} \end{Bmatrix} = -\boldsymbol{r}^r \tag{5}$$

then $\mathbf{u}^{r+1} = \mathbf{u}^r + \triangle \mathbf{u}^{r+1}$ and $\mathbf{p}^{r+1} = \mathbf{p}^r + \triangle \mathbf{p}^{r+1}$. Loop until $\boldsymbol{r}$ satisfies the tolerance.
In order to apply Newton-Raphson's method, we need to calculate $\frac{\partial \mathbf{C}}{\partial \boldsymbol{v}}\mathbf{v}$. The detailed derivation can

be found in the auxiliary file.
The implementation of elemental convection matrix is

```
function C = ConvectionMatrix(X,T,referenceElement,velo)

% X,T: nodal coordinates and connectivities for velocity
% XP,TP: nodal coordinates and connectivities for pressure
% referenceElement: reference element properties (quadrature, shape functions...)


elem = referenceElement.elemV;
ngaus = referenceElement.ngaus;
wgp = referenceElement.GaussWeights;
N = referenceElement.N;
Nxi = referenceElement.Nxi;
Neta = referenceElement.Neta;
NP = referenceElement.NP;
ngeom = referenceElement.ngeom;

% Number of elements and number of nodes in each element
[nElem,nenV] = size(T);
% Number of nodes
nPt_V = size(X,1);
if elem == 11
    nPt_V = nPt_V + nElem;
end


% Number of degrees of freedom
nedofV = 2*nenV;
ndofV = 2*nPt_V;


C = zeros(ndofV,ndofV);

% Loop on elements
for ielem = 1:nElem
    % Global number of the nodes in element ielem
    Te = T(ielem,:);

    % Coordinates of the nodes in element ielem
    Xe = X(Te(1:ngeom),:);
    % Degrees of freedom in element ielem
    Te_dof = reshape([2*Te-1; 2*Te],1,nedofV);
    Ce = zeros(nedofV,nedofV);
    % Element matrices
    Ve = velo(Te,:);
    for ig = 1:ngaus
        N_ig    = N(ig,:);
        Nxi_ig  = Nxi(ig,:);
        Neta_ig = Neta(ig,:);
        Jacob = [
            Nxi_ig(1:ngeom)*(Xe(:,1)) Nxi_ig(1:ngeom)*(Xe(:,2))
            Neta_ig(1:ngeom)*(Xe(:,1))  Neta_ig(1:ngeom)*(Xe(:,2))
            ];
```

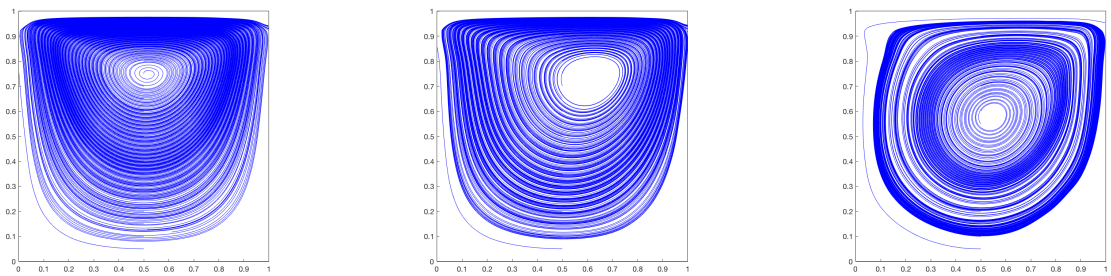
```
        dvolu = wgp(ig)*det(Jacob);
        res = Jacob\[Nxi_ig;Neta_ig];
        nx = res(1,:);
        ny = res(2,:);
        Ngp = [reshape([1;0]*N_ig,1,nedofV); reshape([0;1]*N_ig,1,nedofV)];
        % Gradient
        Nx = [reshape([1;0]*nx,1,nedofV); reshape([0;1]*nx,1,nedofV)];
        Ny = [reshape([1;0]*ny,1,nedofV); reshape([0;1]*ny,1,nedofV)];
        V_ig = N_ig*Ve;
        Vx = [V_ig(1) 0;0 V_ig(1)];
        Vy = [V_ig(2) 0;0 V_ig(2)];
        Ce = Ce + Ngp'*(Vx*Nx+Vy*Ny)*dvolu;
    end
    % Assemble the element matrices
    C(Te_dof, Te_dof) = C(Te_dof, Te_dof) + Ce;

end
```

The implementation of $\frac{\partial \mathbf{C}}{\partial \boldsymbol{v}}\mathbf{v}$ is

```
function G = dCdU_U(X,T,referenceElement,velo)

% X,T: nodal coordinates and connectivities for velocity
% XP,TP: nodal coordinates and connectivities for pressure
% referenceElement: reference element properties (quadrature, shape functions...)


elem = referenceElement.elemV;
ngaus = referenceElement.ngaus;
wgp = referenceElement.GaussWeights;
N = referenceElement.N;
Nxi = referenceElement.Nxi;
Neta = referenceElement.Neta;
NP = referenceElement.NP;
ngeom = referenceElement.ngeom;

% Number of elements and number of nodes in each element
[nElem,nenV] = size(T);
% Number of nodes
nPt_V = size(X,1);
if elem == 11
    nPt_V = nPt_V + nElem;
end


% Number of degrees of freedom
nedofV = 2*nenV;
ndofV = 2*nPt_V;


G = zeros(ndofV,ndofV);

% Loop on elements
for ielem = 1:nElem
    % Global number of the nodes in element ielem
    Te = T(ielem,:);
```

```
37
38      % Coordinates of the nodes in element ielem
39      Xe = X(Te(1:ngeom),:);
40      % Degrees of freedom in element ielem
41      Te_dof = reshape([2*Te-1; 2*Te],1,nedofV);
42      Ge = zeros(nedofV,nedofV);
43      % Element matrices
44      Ve = velo(Te,:);
45      xe = Xe(:,1); ye = Xe(:,2);
46      for ig = 1:ngaus
47          N_ig    = N(ig,:);
48          Nxi_ig  = Nxi(ig,:);
49          Neta_ig = Neta(ig,:);
50          Jacob = [
51              Nxi_ig(1:ngeom)*(Xe(:,1)) Nxi_ig(1:ngeom)*(Xe(:,2))
52              Neta_ig(1:ngeom)*(Xe(:,1))  Neta_ig(1:ngeom)*(Xe(:,2))
53              ];
54          dvolu = wgp(ig)*det(Jacob);
55          res = Jacob\[Nxi_ig;Neta_ig];
56          nx = res(1,:);
57          ny = res(2,:);
58          Ngp = [reshape([1;0]*N_ig,1,nedofV); reshape([0;1]*N_ig,1,nedofV)];
59
60
61          dudx = nx*Ve(:,1);
62          dudy = ny*Ve(:,1);
63          dvdx = nx*Ve(:,2);
64          dvdy = ny*Ve(:,2);
65          gradV = [dudx dudy;dvdx dvdy];
66          Ge = Ge + Ngp'*gradV*Ngp*dvolu;
67      end
68      % Assemble the element matrices
69      G(Te_dof, Te_dof) = G(Te_dof, Te_dof) + Ge;
70
71  end
```

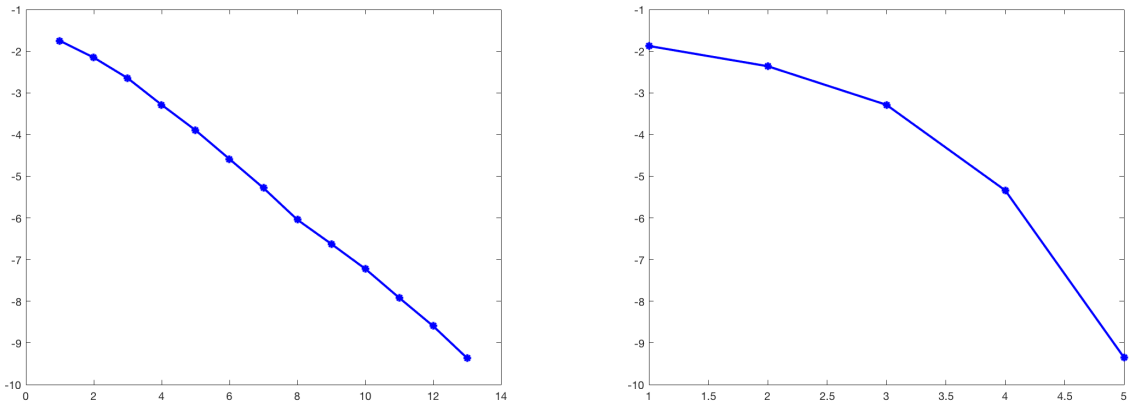
The cavity flow of Navier-Stokes problem is entirely characterized by the Reynolds number. Figure 8 allows us to visualize the streamlines under different Reynolds number. It can be observed that as Reynolds number increases boundary layers are more obvious and the position of the main vortex moves towards the center of the cavity.



(a) Re=10 (b) Re=100 (c) Re=1000

Figure 8: Cavity Navier-Stokes

For $R_e = 100$, the convergence achieved in iteration number 13 using Picard scheme with $tol = 0.5e - 8$. However, we only need 5 interations for Newton-Raphson method. As is shown in Figure 9, Picard gives linear convergence rate while Newton-Raphson is quadratic.



(a) Picard

(b) Newton-Raphson

Figure 9: Convergence

• Stokes equations

$$\begin{cases} -\nu\nabla^2\underline{v}+\nabla p=\underline{b} & \text{in } \Omega \quad ① \\ \nabla\cdot\underline{v}=0 & \text{in } \Omega \quad ② \end{cases}$$

• Weak form

① $\Rightarrow (\underline{w}, -\nu\nabla^2\underline{v}) + (\underline{w}, \nabla p) = (\underline{w}, \underline{b})$ Integration by part

$\Rightarrow a(\underline{w},\underline{v}) + b(\underline{w},p) = (\underline{w},\underline{b}) + (\underline{w},\underline{t})_{\Gamma_N}$ ($\underline{t}=0$) All Dirichlet boundary condition in this case.

② $\Rightarrow b(\underline{v},q)=0$

with $a(\underline{w},\underline{v}) = \nu(\nabla\underline{w},\nabla\underline{v})$

$b(\underline{v},q) = -(q,\nabla\cdot\underline{v})$

Therefore, the Galerkin weak form of the stokes problem is

$$\begin{cases} a(\underline{w},\underline{v})+b(\underline{w},p)=(\underline{w},\underline{b}) \\ b(\underline{v},q)=0 \end{cases} \quad ③$$

• Stabilization

We apply GLS stabilization method to solve the stokes problem with LBB non-compliant elements.

GLS derives from the minimization of the least-squares form

$$L_S(\underline{v},p) := (-\nu\nabla^2\underline{v}+\nabla p-\underline{b}, -\nu\nabla^2\underline{v}+\nabla p-\underline{b})$$

$$\Rightarrow \left.\frac{dL_S(\underline{v}+\varepsilon\underline{w},p+\varepsilon q)}{d\varepsilon}\right|_{\varepsilon=0}=0$$

$$\Rightarrow \begin{cases} (-\nu\nabla^2\underline{w}, -\nu\nabla^2\underline{v}+\nabla p-\underline{b}) = 0 & \forall \underline{w}\in\mathcal{U} \\ (\nabla q, -\nu\nabla^2\underline{v}+\nabla p-\underline{b})=0 & \forall q\in\mathcal{Q} \end{cases} \quad ④$$

The stabilization of the stokes problem is obtained by adding ④ to ③.

To avoid additional continuity requirements due to the presence of second spatial derivatives, the terms added to ③ act on the element interiors only. We use $\tau_1 = \alpha_0\frac{h^2}{4\nu}$, $\tau_2=0$ for this exercise.

This gives us ("h" is dropped off for simplicity)

$$\begin{cases} a(\underline{w},\underline{v})+b(\underline{w},p)+\sum_{e=1}^{n_{el}}\tau_1(-\nu\nabla^2\underline{w}, -\nu\nabla^2\underline{v}+\nabla p-\underline{b})_{\Omega_e} = (\underline{w},\underline{b}) \\ b(\underline{v},q)-\sum_{e=1}^{n_{el}}\tau_1(\nabla q, -\nu\nabla^2\underline{v}+\nabla p-\underline{b})_{\Omega_e}=0 \end{cases} \quad ⑤$$

We use $\tau_1=\frac{1}{3}\frac{h^2}{4\nu}$ $\tau_2=0$ for this exercise and linear elements. ⑤ is simplified to

$$\begin{cases} a(\underline{w},\underline{v})+b(\underline{w},p)=(\underline{w},\underline{b}) \\ b(\underline{v},q)-\sum_{e=1}^{n_{el}}\tau_1(\nabla q,\nabla p-\underline{b})_{\Omega_e}=0 \end{cases}$$

$$\Rightarrow \begin{cases} a(\underline{w},\underline{v}) + b(\underline{w},p) = (\underline{w},\underline{b}) \\ b(\underline{v},q) - \sum_{e=1}^{n_{el}} \tau_1 (\nabla q, \nabla p)_{\Omega e} = \sum_{e=1}^{n_{el}} \tau_1 (\nabla q, -\underline{b})_{\Omega e} \end{cases} \quad (6)$$

- Matrix

The discritization of the stabilized weak form yields.

$$\begin{pmatrix} \underset{\sim}{K} & \underset{\sim}{G}^T \\ \underset{\sim}{G} & \underset{\sim}{L} \end{pmatrix} \begin{pmatrix} \underline{v} \\ p \end{pmatrix} = \begin{pmatrix} \underline{f} \\ f_q \end{pmatrix}$$

Where

$\underset{\sim}{K}$	$a(\underline{w},\underline{v})$	$[\text{grad } N]^T [\text{grad } N]$
$\underset{\sim}{G}^T$	$b(\underline{w},p)$	$-\underset{\sim}{D}^T \hat{N}$
$\underset{\sim}{G}$	$b(\underline{v},q)$	$-\hat{N}^T \underset{\sim}{D}$
$\underset{\sim}{L}$	$-\sum_{e=1}^{n_{el}} \tau_1 (\nabla q, \nabla p)_{\Omega e}$	$-\tau_1 (\nabla \hat{N})^T \nabla \hat{N}$
$\underline{f}$	$(\underline{w},\underline{b})$	$\underline{N}^T \underline{b}$
f_q	$\sum_{e=1}^{n_{el}} \tau_1 (\nabla q, -\underline{b})_{\Omega e}$	$-\tau_1 (\nabla \hat{N})^T \underline{b}$

- Navier-stokes equations

$$\begin{cases} -\nu\Delta \underline{v} + (\underline{v}\cdot\nabla)\underline{v} + \nabla p = \underline{b} & \text{in } \Omega \\ \nabla\cdot\underline{v} = 0 & \text{in } \Omega \end{cases}$$

All Dirichlet boundary condition

- Weak form

Follow the same procedure as in stokes problem, we have

$$\begin{cases} a(\underline{w},\underline{v}) + c(\underline{w},\underline{v},\underline{v}) - b(\underline{w},p) = (\underline{w},\underline{b}) \\ b(\underline{v},q) = 0 \end{cases} \quad ①$$

with $c(\underline{w},\underline{v},\underline{a}) := \int_\Omega \underline{w}\cdot(\underline{a}\cdot\nabla)\underline{v}\, d\Omega$.

- Matrix form

The discritization of the weak form ① gives:

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

Where

$\underset{\sim}{K}$, $\underset{\sim}{G}$, $\underline{f}$ are the same as in stokes problem.

$\underset{\sim}{C}(\underline{v})$ $c(\underline{w},\underline{v},\underline{v}) = \int_\Omega \underline{w}\cdot(\underline{v}\cdot\nabla)\underline{v}\, d\Omega$

- Newton Raphson. scheme

$$\underset{\sim}{J} = \begin{pmatrix} \underset{\sim}{K} + \underset{\sim}{C} + \frac{\partial \underset{\sim}{C}}{\partial \underline{v}}\cdot\underline{v} & \underset{\sim}{G}^T \\ \underset{\sim}{G} & \underset{\sim}{0} \end{pmatrix} \qquad \underline{r} = \begin{bmatrix} (\underset{\sim}{K} + \underset{\sim}{C})\underline{v} + \underset{\sim}{G}^T p - \underline{b} \\ \underset{\sim}{G}\underline{v} \end{bmatrix}$$

$$\begin{cases} \underset{\sim}{J}(\underline{x}^k)\,\delta\underline{x}^{k+1} = -\underline{r}(\underline{x}^k) \\ \underline{x}^{k+1} = \underline{x}^k + \delta\underline{x}^{k+1} \\ \text{Loop until } \underline{r} \text{ satisfies tolerance.} \end{cases}$$

$$D(\underline{w}\cdot(\underline{v}\cdot\nabla)\underline{v})[\delta\underline{v}] = \underbrace{\underline{w}\cdot(\delta\underline{v}\cdot\nabla)\underline{v}}_{②\ \frac{\partial \underline{v}}{\partial \underline{v}}\delta\underline{v}} + \underbrace{\underline{w}\cdot(\underline{v}\cdot\nabla)\delta\underline{v}}_{③\ \underset{\sim}{C}}$$

$$③ \Rightarrow (\underline{v}\cdot\nabla)\delta\underline{v} = \begin{bmatrix} v_x \frac{\partial \delta v_x}{\partial x} + v_y \frac{\partial \delta v_x}{\partial y} \\ v_x \frac{\partial \delta v_y}{\partial x} + v_y \frac{\partial \delta v_y}{\partial y} \end{bmatrix} = \begin{bmatrix} v_x & 0 & v_y & 0 \\ 0 & v_x & 0 & v_y \end{bmatrix} \begin{bmatrix} \frac{\partial \delta v_x}{\partial x} \\ \frac{\partial \delta v_x}{\partial y} \\ \frac{\partial \delta v_y}{\partial x} \\ \frac{\partial \delta v_y}{\partial y} \end{bmatrix}$$

Therefore, $\underset{\sim}{C} \to [\text{matN}]^T \begin{bmatrix} v_x & 0 & v_y & 0 \\ 0 & v_x & 0 & v_y \end{bmatrix} [\text{grad N}]$

$$② \Rightarrow (\delta\underline{v}\cdot\nabla)\underline{v} = \begin{bmatrix} \delta v_x \frac{\partial v_x}{\partial x} + \delta v_y \frac{\partial v_x}{\partial y} \\ \delta v_x \frac{\partial v_y}{\partial x} + \delta v_y \frac{\partial v_y}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{bmatrix} \begin{bmatrix} \delta v_x \\ \delta v_y \end{bmatrix}$$

Therefore, $\frac{\partial \underset{\sim}{C}}{\partial \underline{v}}\underline{v} \to [\text{matN}]^T \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{bmatrix} [\text{mat N}]$