MSc in Computational Mechanics

Finite Elements in Fluids

MATLAB Assignment 3: Incompressible viscous flow: Stokes and Navier-Stokes problems

Submitted By:
Mario Alberto Méndez Soto

Submitted To:
Prof. Pablo Saez
Prof. Antonio Huerta

Spring Semester, 2019

1 Stokes problem

1.1 Governing equations and stabilization technique

Incompressible flows in which advective inertial forces are small compared with viscous forces, i.e. Reynolds number is low, are governed by Stokes equations:

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

This system of PDE's can be discretized using a Galerkin formulation and resulting in the following linear system of equations:

$$\begin{pmatrix} \mathbf{K} & \mathbf{G}^T \\ -\mathbf{G} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{h} \end{pmatrix}$$

Existence of an smooth and unique solution requires the finite elements used for the discretizations to satisfy the so-called *LBB condition*. This condition states that velocity and pressure cannot be chosen arbitrarily, but instead a link between them is necessary. Otherwise, although a convergent velocity field might be obtained, the pressure could present with spurious and oscillatory results.

Even though several elements can be used to solve the Stokes problem, some of the elements satisfy the LBB conditions and others do not (see Figure (1))

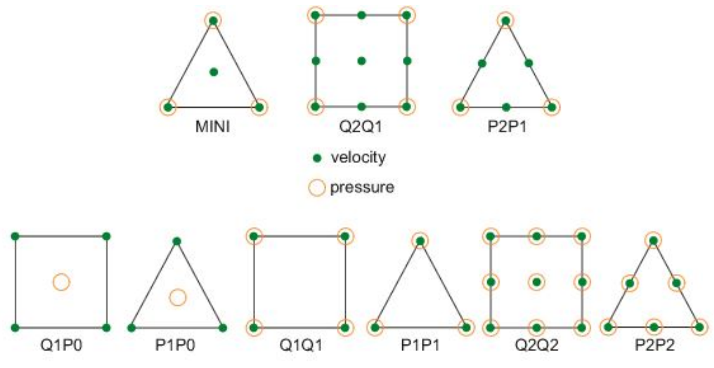


Fig. 1 – Example of 2D stable (upper row) and unstable (lower row) velocity and pressure elements

In analogy to the convection-diffusion problem, stabilization techniques can be used for unstable elements. An implementation of the SUPG stabilization technique for the Stokes problem results in the following stabilized system of equations:

$$\begin{pmatrix} \mathbf{K} + \bar{\mathbf{K}} & \mathbf{G}^T + \bar{\mathbf{G}}^T \\ -\mathbf{G} + \bar{\mathbf{G}} & \mathbf{0} + \bar{\mathbf{L}} \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} + \bar{\mathbf{f}}_{\mathbf{w}} \\ \mathbf{h} + \bar{\mathbf{f}}_{\mathbf{q}} \end{pmatrix}$$

where the extra-terms correspond to the SUPG added matrices, whose mathematical definitions can be found in specialized literature.

1.2 Implementation of the stabilization technique

The initial given code had a fully-operational implementation of the Galerkin formulation for a classical cavity problem, in which, a confined incompressible isothermal flow is located in a square lid-driven cavity (see Figure (2)). The lid can move with a velocity 1, while the rest of the sides are fixed. Considering that the two upper corners belong to the fixed vertical walls causes the introduction of singularities in the pressure field as it will be noticeable later on. Moreover, the lower part of the boundary is prescribed with zero pressure field.

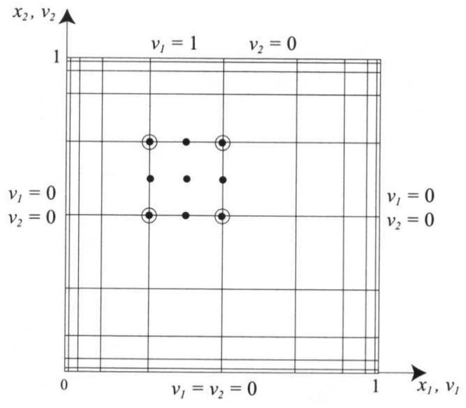


Fig. 2 – Cavity problem: problem statement and boundary conditions

The code allows the user to choose quadrilateral and triangular elements with linear and quadratic interpolations for the pressure and velocity fields.

It is worth mentioning that all the solutions presented in this report were computed using a structured 10x10 mesh for both the pressure and velocity fields. The discretized domains in the case of P2P1 elements are depicted in Figure (3).

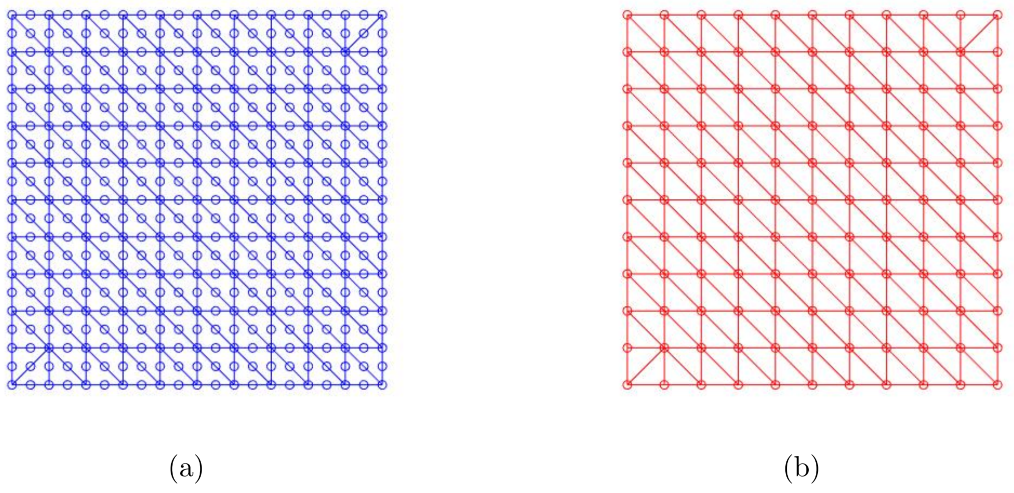


Fig. 3 – Discretization for the velocity (a) and (b) for elements P2P1

As expected, for elements Q2Q1 and P2P1 the solution is stable and no oscillations are present in the results (see Figure (4) and Figure (5)).

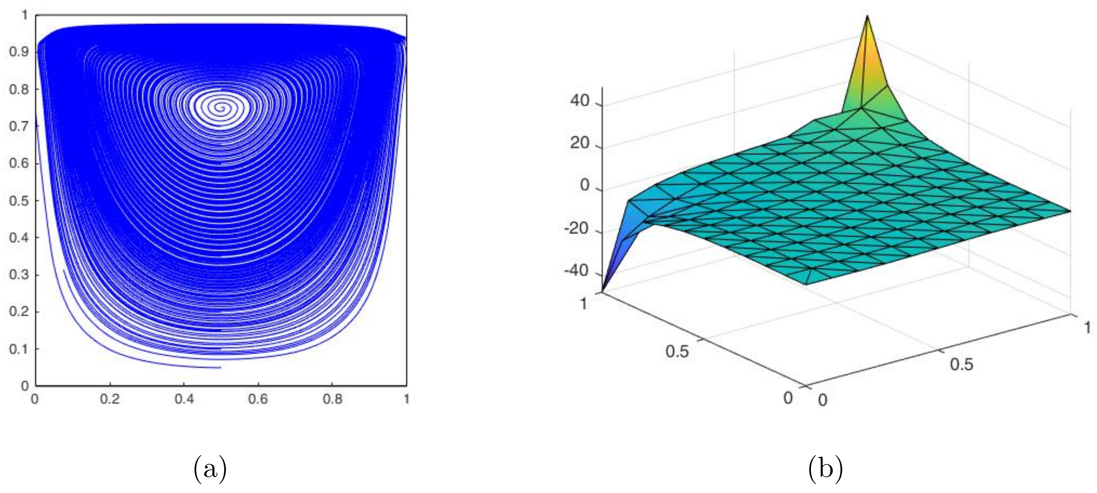


Fig. 4 – Streamlines (a) and pressure field (b) obtained using P2P1 elements

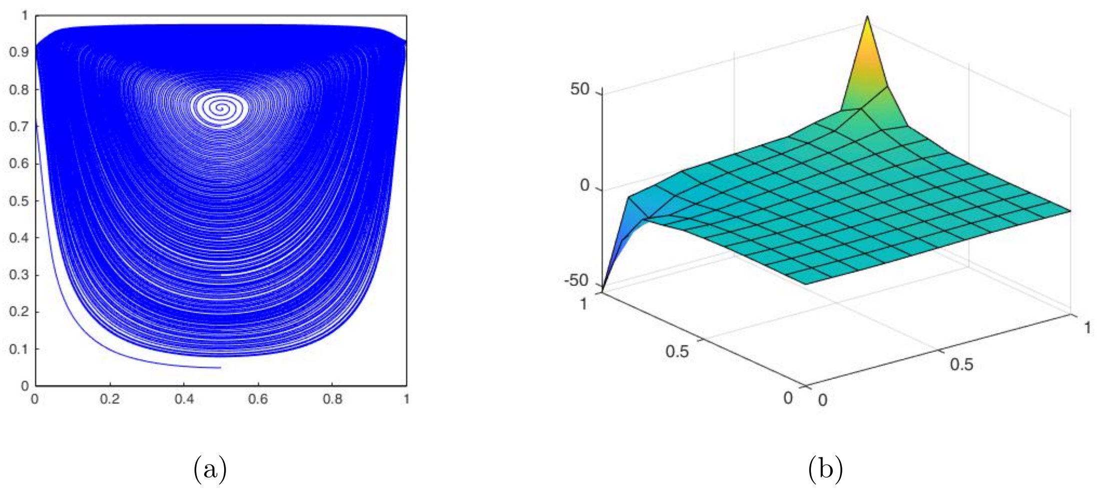
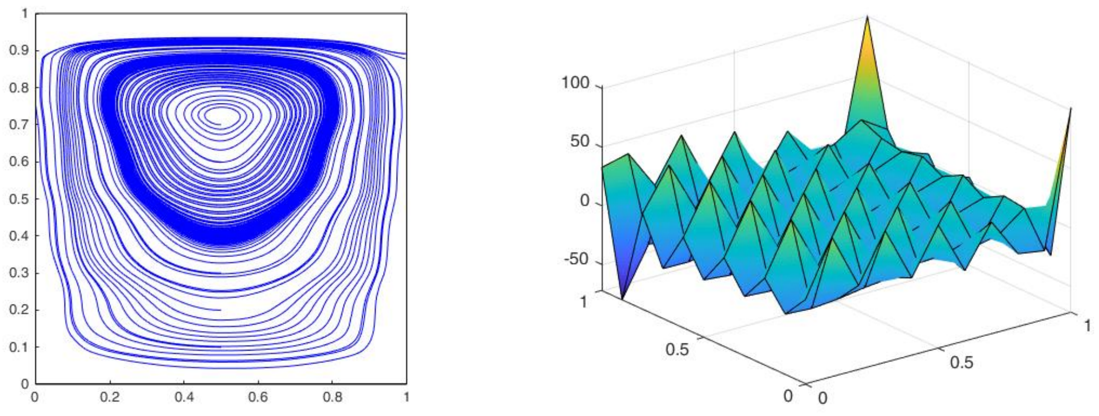


Fig. 5 – Streamlines (a) and pressure field (b) obtained using Q2Q1 elements

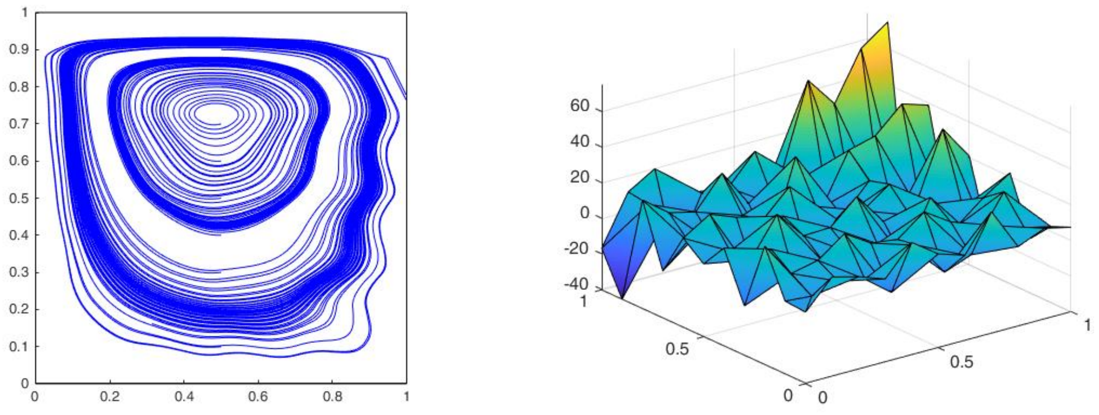
On the other hand, solutions computed using unstable elements are spurious with significant instabilities in the pressure field. For instance, Figure (6) shows the results computed using Q1Q1 elements. Moreover, the streamlines and pressure fields obtained with P1P1 elements are shown in Figure (7).

Using the stabilization technique introduced in the previous section, the objective of the present work was to code a SUPG stabilization for unstable elements with linear interpolations. Firstly, since the elements considered have linear interpolation, the only non-zero stabilizing matricial expressions of the technique will be $\bar{\mathbf{L}}$ and $\bar{\mathbf{f}}_{\mathbf{q}}$.



(a) (b)

Fig. 6 – Streamlines (a) and pressure field (b) obtained using Q1Q1 elements



(a) (b)

Fig. 7 – Streamlines (a) and pressure field (b) obtained using P1P1 elements

Thus, given the mathematical definition of $\bar{\mathbf{L}}$

$$\bar{\mathbf{L}} = \sum_e \int_{\Omega_e} \tau_1 (\nabla q) \cdot (\nabla p) d\Omega_e$$

for a 2D problem the following transformation can be done:

$$(\nabla q) \cdot (\nabla p) = \begin{bmatrix} \frac{\partial q}{\partial x} & \frac{\partial q}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$

Considering the discretization of the pressure field, the previous equation becomes:

$$\begin{aligned} \frac{\partial p}{\partial x} &= \frac{\partial N_1}{\partial x} p_1 + \frac{\partial N_2}{\partial x} p_2 + ... \\ &= \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & ... & \frac{\partial N_m}{\partial x} \end{bmatrix} \begin{bmatrix} p_1 & p_2 & ... & p_m \end{bmatrix} \end{aligned}$$

in which the first vector on the left is defined within the program as vector $\mathbf{n_x}$ whereas the second one is the vector of unknowns nodal values $\mathbf{p}$. Thus, the dot product of the gradients yields:

$$(\nabla q) \cdot (\nabla p) = (\mathbf{n_x q})^T (\mathbf{n_x p}) + (\mathbf{n_y q})^T (\mathbf{n_y p})$$

Since the weak formulation holds for any arbitrary test function q, the expression becomes:

$$(\nabla q) \cdot (\nabla p) = (\mathbf{n_x}^T \mathbf{n_x} + \mathbf{n_y}^T \mathbf{n_y}) \mathbf{p}$$

Similarly, for the vector $\bar{\mathbf{f}}_\mathbf{q}$ the following expression is valid:

$$\begin{aligned} (\nabla q)(-\mathbf{f}) &= - \begin{bmatrix} \frac{\partial q}{\partial x} & \frac{\partial q}{\partial y} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \\ &= - \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & ... & \frac{\partial N_m}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & ... & \frac{\partial N_m}{\partial y} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \end{aligned}$$

where the first matrix on the left can be defined using the program's notations as $[\mathbf{n_x}; \mathbf{n_y}]^T$.

As a result, the stabilizing terms $\bar{\mathbf{L}}$ and $\bar{\mathbf{f}}_\mathbf{q}$ can be coded as follows:

```
Le = Le + tau1*(nx'*nx+ny'*ny)*dvolu;
f_qe = f_qe - tau1*([nx; ny]'*f_igaus)*dvolu;
```

Based on the previously derived expressions, the stabilization technique was introduced by creating an additional function *StokesSystemStable.m* which is used when linear unstable elements are chosen. The structure of the code is similar to the *StokesSystem.m* original program with additional lines for the new matricial stabilizing expressions.

Using this implementation, the solutions obtained with Q1Q1 and P1P1 elements are stable producing reliable results (see Figures (8) and (9)).

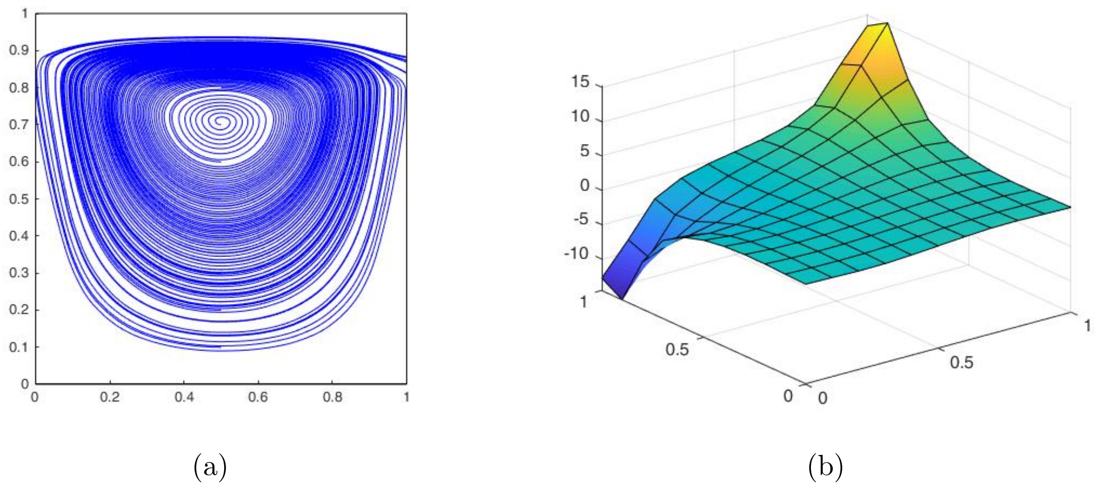


Fig. 8 – Streamlines (a) and pressure field (b) obtained using Q1Q1 stabilized elements

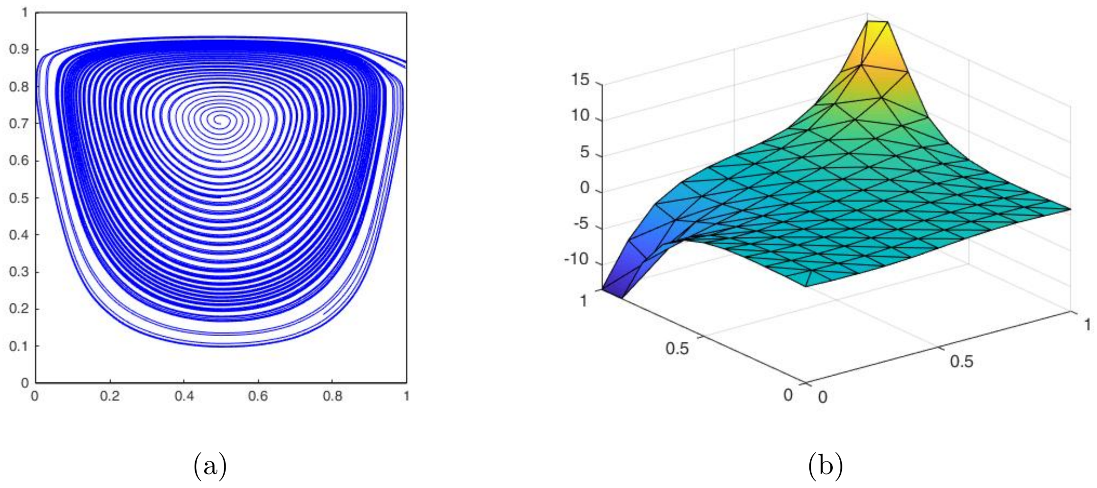


Fig. 9 – Streamlines (a) and pressure field (b) obtained using stabilized P1P1 elements

2 Navier-Stokes problem

2.1 Governing equations

Viscous fluid phenomena where inertial forces become important cannot longer by modeled using Stokes equations. Instead, the inertial forces are driven by pressure and viscous terms and the phenomena can be mathematically described using the so-called Navier Stokes equations:

$$\begin{cases} -\nu\nabla^2\mathbf{v} + (\mathbf{v}\cdot\nabla)\mathbf{v} + \nabla p = \mathbf{b} & \text{in } \Omega \\ \nabla\cdot\mathbf{v} = 0 & \text{in } \Omega \\ \mathbf{v} = \mathbf{v}_D & \text{on } \Gamma_D \\ \mathbf{n}\cdot\sigma = \mathbf{t} & \text{on } \Gamma_N \end{cases}$$

The stationary Navier-Stokes solution can be entirely characterized by the Reynolds Number $Re = \dfrac{V_{ref}L_{ref}}{\nu}$.

A Galerkin formulation of the Navier-Stokes equation allows for the problem to be reduced to the following system of equations:

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

which is by definition a non-linear system of equations that requires the use of iterative techniques such as Picard's or Newton-Raphson.

2.2 Convection matrix

The lid-driven cavity problem (see Figure (2)) introduced in the previous section is once again analyzed and solved, but in this case considering non-negligible inertial forces.

Since the solution is entirely defined by the Reynolds number and the reference velocity and length are given by the boundary condition ($V_{ref} = 1$) and domain dimension ($L_{ref} = 1$), any change in the Reynolds numbers is a direct consequence of an increase or decrease of the viscous forces through the viscosity parameter ν.

To solve this problem, firstly the code for the convection matrix $C(\mathbf{v})$ needed to be written. Thus, considering the definition of the convection matrix:

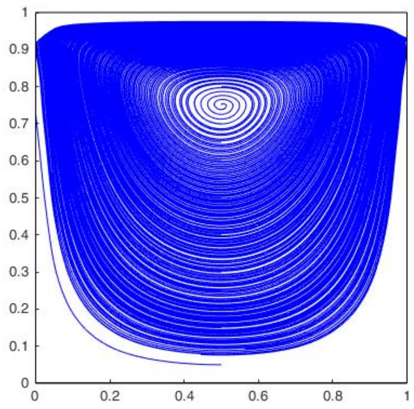
$$[C(\mathbf{v^h})]_{ij} = \Big(\mathbf{N}_i, \Big(\Big(\sum_{j=1}^{n} \mathbf{v}_j N_j(\mathbf{x})\Big)\cdot\nabla\Big), \mathbf{v^h}\Big)$$

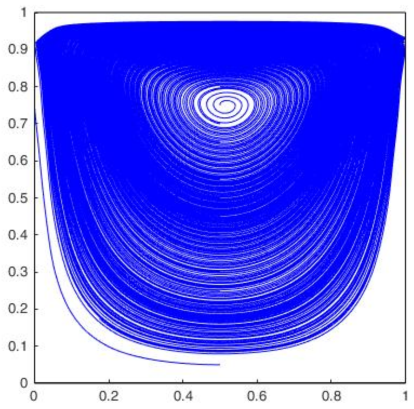
and the given notations used within the program, the implementation for the element convection matrix can be written as follows:

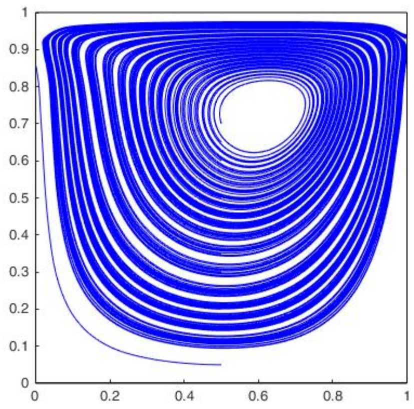
```
v_igaus = N_ig*u_e;
Ce = Ce + Ngp'*(v_igaus(1)*Nx+v_igaus(2)*Ny)*dvolu;
```

As a result, the convection matrix is computed using a newly written script called *Convection-Matrix.m* that is embedded into the main body of the program.

Using this implementation and since Picard's method was initially given, solutions for different Reynolds number can be computed. Figures (10) shows the velocity streamlines obtained for three different Reynolds numbers using Q2Q2 elements.


(a) Re = 1


(b) Re = 10


(c) Re = 100

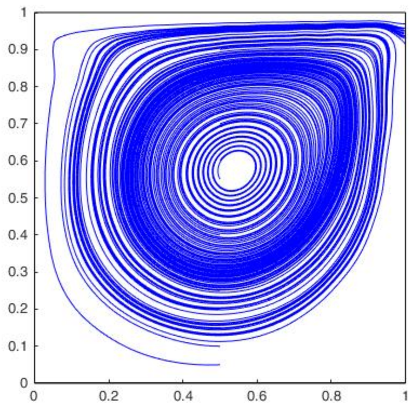

(d) Re = 1000

Fig. 10 – Streamlines solutions for different Reynolds numbers using Q2Q2 elements with Picard's method

As it can be seen in the figures above, for low Reynolds numbers the solution is very close to the one obtained using the Stokes equations. On the other hand, as Reynolds number increases, the main vortex of the solution drifts and its position gets closer to the geometric center of the domain.

2.3 Picard and Newton-Raphson methods for non-linear system of equations

After implementing the computation for the convection matrix, as shown above, the non-linear system of equations can be solved using a Picard's method. For this method, having an initial solution guess $\mathbf{x^0}$, the solution after k iterations can be computed by solving the linear system:

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

Thus, the method requires the solution of one linear system of equations per iteration and the convergence rate is usually linear.

An alternative methodology to solve the Navier-Stokes non-linear system of equations is the Newton-Raphson method. This method solves an equation of the form $\mathbf{r}(\mathbf{x}) = 0$ where, for the Navier-Stokes system, $\mathbf{r}(\mathbf{x})$ is defined as follows:

$$\mathbf{r} = \begin{bmatrix} \Big(\mathbf{K} + \mathbf{C}(\mathbf{v})\Big)\mathbf{v} + \mathbf{G}^T\mathbf{p} - \mathbf{f} \\ \mathbf{G}\mathbf{v} \end{bmatrix}$$

Thus, the solution after k iterations can be found by solving the linear system:

$$\begin{cases} \mathbf{J}(\mathbf{x}^k)\Delta\mathbf{x}^{k+1} = -r(\mathbf{x}^k) \\ \mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}^{k+1} \end{cases}$$

where $\mathbf{J}(\mathbf{x})$ is called the Jacobian matrix and it is mathematically defined as:

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial \mathbf{r}_1}{\partial \mathbf{v}} & \dfrac{\partial \mathbf{r}_1}{\partial \mathbf{p}} \\ \dfrac{\partial \mathbf{r}_2}{\partial \mathbf{v}} & \dfrac{\partial \mathbf{r}_2}{\partial \mathbf{p}} \end{pmatrix}$$

Even though the expressions $\dfrac{\partial \mathbf{r}_1}{\partial \mathbf{p}}$, $\dfrac{\partial \mathbf{r}_2}{\partial \mathbf{v}}$, and $\dfrac{\partial \mathbf{r}_2}{\partial \mathbf{p}}$ are trivial, the term $\dfrac{\partial \mathbf{r}_1}{\partial \mathbf{v}}$ requires more work. Considering the original bilinear form from which the matrix $\mathbf{C}(\mathbf{v})$ comes from, allows us to find the expression $\dfrac{\partial \mathbf{r}_1}{\partial \mathbf{v}}$ as follows:

$$\frac{\partial \mathbf{r}_1}{\partial \mathbf{v}} = \mathbf{K} + \underbrace{\Big(\mathbf{N}_i, (\mathbf{v}\cdot\nabla), \mathbf{N}_j\Big)}_{\mathbf{C_2}(\mathbf{v})} + \underbrace{\Big(\mathbf{N}_i, (\mathbf{N}_j\cdot\nabla), \mathbf{v}\Big)}_{\mathbf{C_1}(\mathbf{v})}$$

For the initial given code, it can be noticed that the firm matrix $\mathbf{C_1}(\mathbf{v})$ matches the definition of the convection matrix $\mathbf{C}(\mathbf{v})$, whereas, for the second term $\mathbf{C_2}(\mathbf{v})$, the following can be deduced:

$$\begin{aligned} \mathbf{w}\cdot(\mathbf{v}\cdot\nabla)\mathbf{v} &= \mathbf{w}\begin{bmatrix} v_x & v_y \end{bmatrix}\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix}\begin{bmatrix} v_x & v_y \end{bmatrix} \\ &= \mathbf{w}\begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{bmatrix}\begin{bmatrix} v_x & v_y \end{bmatrix} \end{aligned}$$

which can be differentiated with respect to $\mathbf{v}$ and using the notations originally given in the code, can be expressed as:

```
Ce2 = Ce2 + Ngp'*([nx ; ny]*u_e)'*Ngp*dvolu;
```

As a result, the $\dfrac{\partial \mathbf{r}_1}{\partial \mathbf{v}}$ can be found by adding the three expressions for $\mathbf{K}$, $\mathbf{C_1}(\mathbf{v})$, and $\mathbf{C_2}(\mathbf{v})$.

Lastly, to implement the Newton-Raphson methodology, a new function script *ConvectionMatrixNR.m* was created that computes matrices $\mathbf{C_1}(\mathbf{v})$ and $\mathbf{C_2}(\mathbf{v})$. Subsequently, the body of the algorithm for the Newton-Raphson method was introduced as follows:

```
[C1,C2] = ConvectionMatrixNR(X,T,referenceElement,velo); %Newly created
    function computes C_1 and C_2
Cred = C1(dofUnk,dofUnk); % Reduced matrix considering BC's
Cred2= C2(dofUnk,dofUnk); % Reduced matrix considering BC's

%Reduced system of equations
Atot = A;
Atot(1:nunkV,1:nunkV) = A(1:nunkV,1:nunkV) + Cred;
btot = [fred - C1(dofUnk,dofDir)*valDir; zeros(nunkP,1)];

%Jacobian matrix
J = A;
J(1:nunkV,1:nunkV) = A(1:nunkV,1:nunkV) + Cred + Cred2;

%Computation of r(x)
res = Atot*sol0 - btot;

% Computation of velocity and pressure increment
solInc = -J\res;
```

Correspondingly the final code was written and the results of the cavity problem with Reynolds number $Re = 500$ using Q2Q1 elements are shown in Figure (11).

For the same problem ($Re = 500$), the convergence rate curves for Picard's and Newton-Raphson methods are depicted in Figure (12). As it can be seen, the Newton-Raphson reaches the sought minimum error after 8 iterations with a quadratic law. On the other hand, Picard's method requires 40 iterations to achieve the same error value converging, conversely, in a linear pattern.

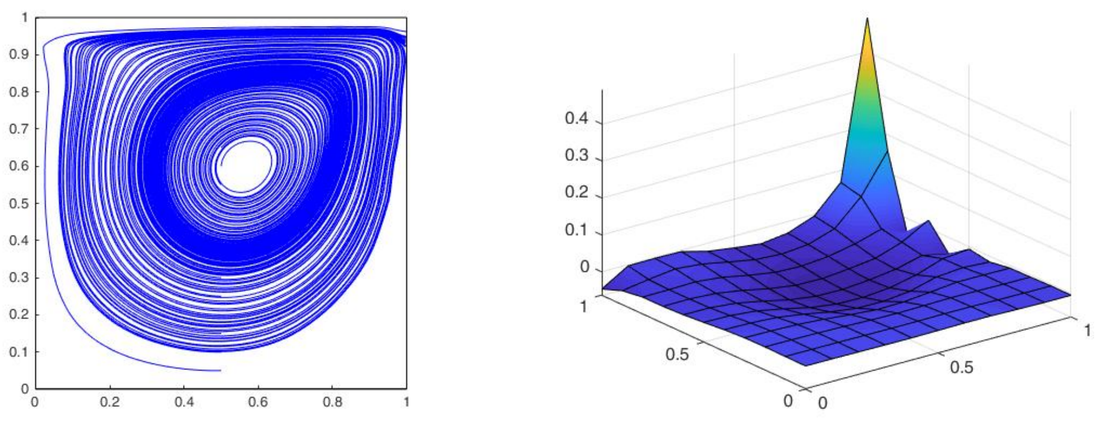


Fig. 11 – Streamlines (a) and pressure field (b) obtained using Q2Q1 elements with a Newton-Raphson method

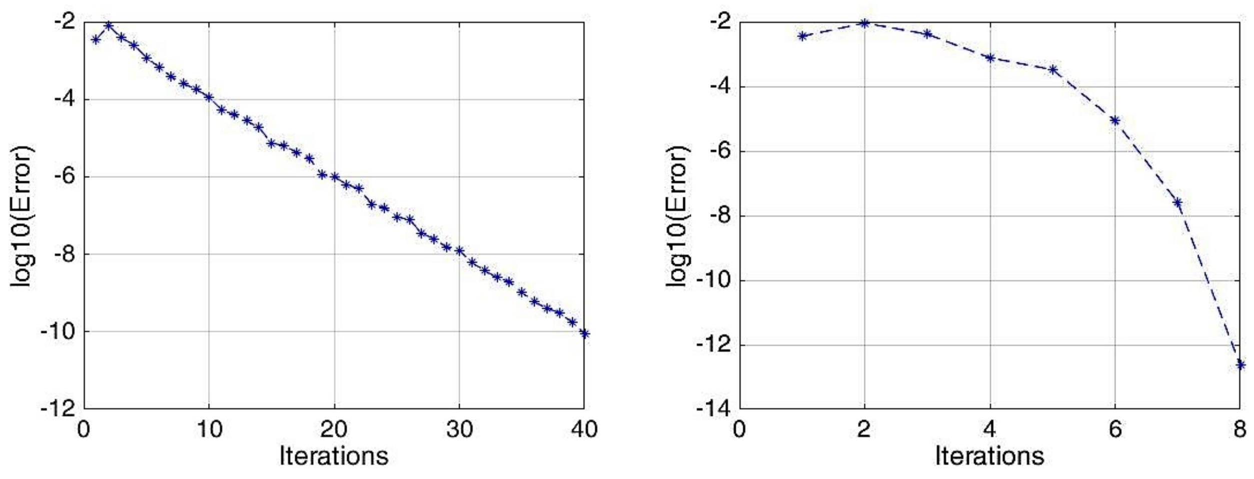


Fig. 12 – Convergence rate using Picard's (a) and Newton Raphson (b) methods