UPC, CIMNE, ETSECCP

FINITE ELEMENTS IN FLUIDS
Assignment 2 : Cavity flow problem

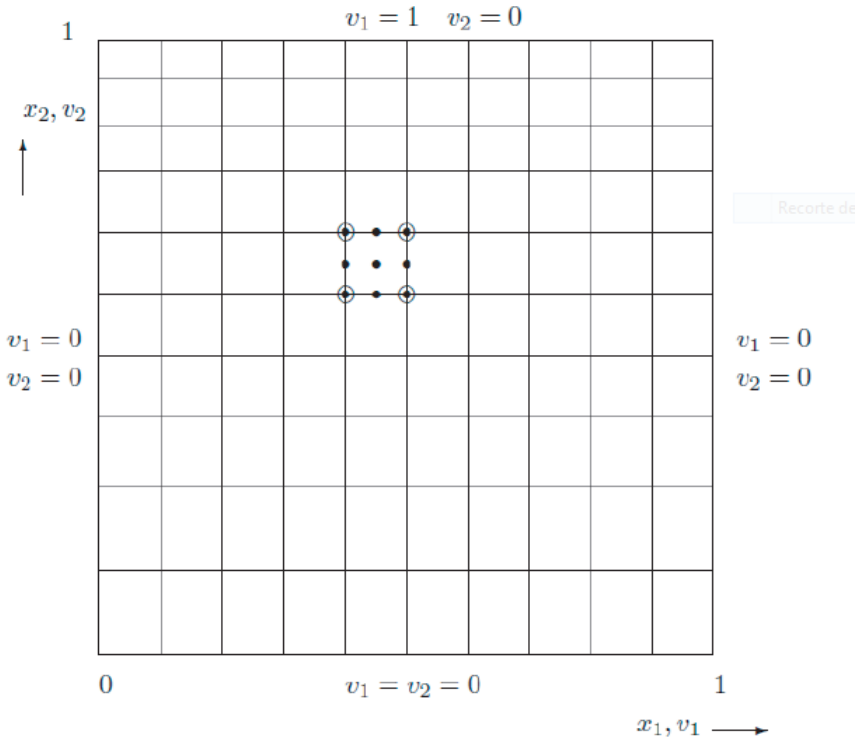
Inocencio Castañar

May 23, 2017

Contents

1 Statement

The cavity flow problem is a standard benchmark test for incompressible flows. The figure below shows a schematic representation of the problem setting. The goal of this exercise is to analyze the results obtained when adopting either the Stokes or the Navier-Stokes equations. Use the code in(*HW2Files_ Cavity*) to compute the finite elements approximation of these problems and answer the questions below.



2 Steady Stokes problem

(a) Use the script *mainStokes.m* to compute the solution of the Stokes problem using a uniform, structured mesh of Q_2Q_0, Q_2Q_1, P_1P_1 and MINI ($P_1^+P_1$) elements, with 20 elements per side. Comment on the results.

First of all, let us explain the solvability condition and the LBB compatibility condition that must be satisfied in the Stokes problem to guarantee that the solution is uniquely defined.

Consider the partiotioned matrix system governing the steady Stokes flow,

$$\begin{pmatrix} \boldsymbol{K} & \boldsymbol{G} \\ \boldsymbol{G}^T & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{h} \end{pmatrix}$$

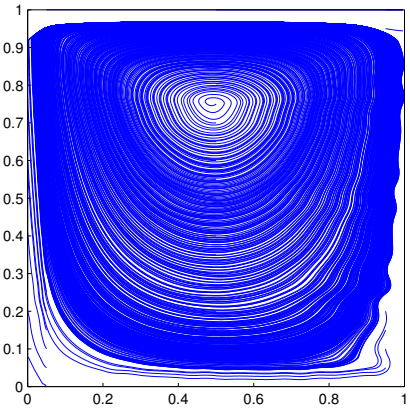
It can be shown that, provided the kernel of matrix $\boldsymbol{G}$ is zero, the global matrix is non-singular. This is the also known as solvability condition. In turn, to have $\ker \boldsymbol{G} = \{\boldsymbol{0}\}$, the velocity and pressure interpolations must satisfy a compatibility condition, called the LBB condition. Inappropriate combinations of velocity and pressure interpolations may render the discrete divergence matrix $\boldsymbol{G^T}$, rank deficient.

LBB compatibility condition states that: The existence of a stable finite element approximate solution $(\boldsymbol{u}^h, \boldsymbol{p}^h)$ to the steady Stokes problem depends on choosing a pair of spaces $\boldsymbol{\mathcal{V}^h}$ and $\mathcal{Q}^h$, such that the following inf-sup condition holds:

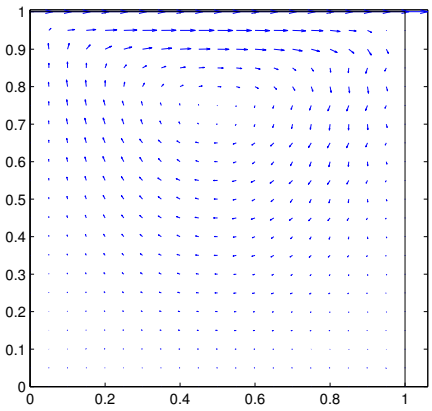
$$\inf_{q^h \in \mathcal{Q}^h} \sup_{\boldsymbol{w}^h \in \boldsymbol{\mathcal{V}}^h} \frac{(q^h, \boldsymbol{\nabla} \cdot \boldsymbol{w}^h)}{\|q\|_0 \|\boldsymbol{w}^h\|_1} \geq \alpha > 0,$$

where α is independent of the mesh size h.

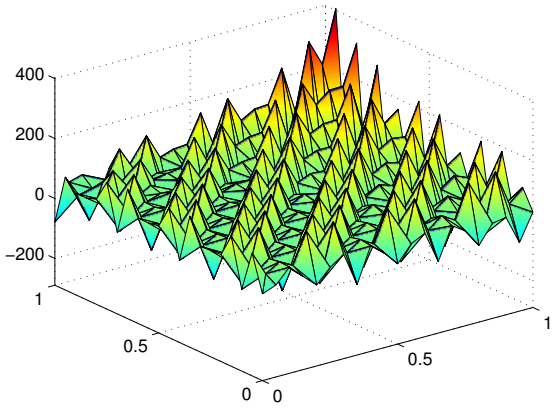
Let us now show firstly a combination of elements which does not satisfy this LBB condition:



(a) Streamlines



(b) Velocity field

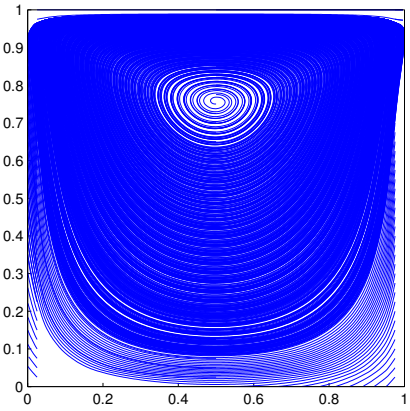


(c) Pressure field

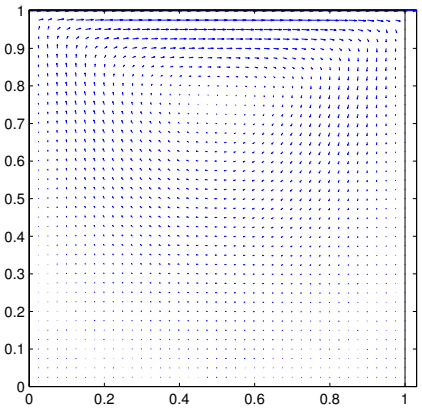
Figure 2.1: Results obtained for P_1P_1 element

Figure 2.1 shows the solution obtained for the steady Stokes problem when velocity is discretized with continuous linear elements (P_1) but also the pressure field. As expected as this element is non LBB compliant it presents inaccurate pressure results. It presents oscillations which are more pronounced in the corners where there is a discontinuity in the boundary conditions.

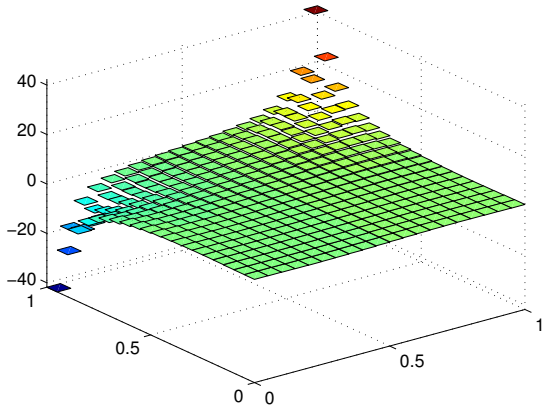
Let us now move the the Q_2Q_0 element and Q_2Q_1 element.



(a) Streamlines

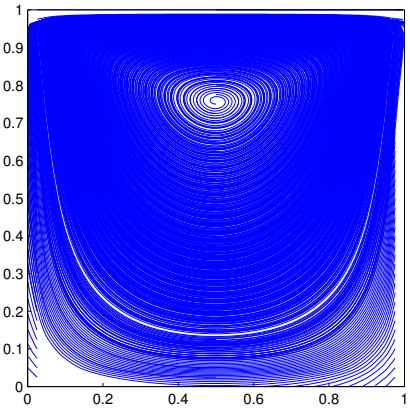


(b) Velocity field

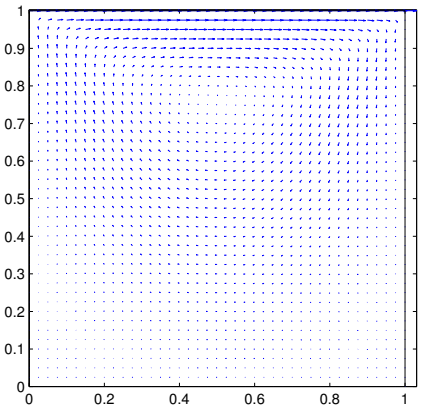


(c) Pressure field

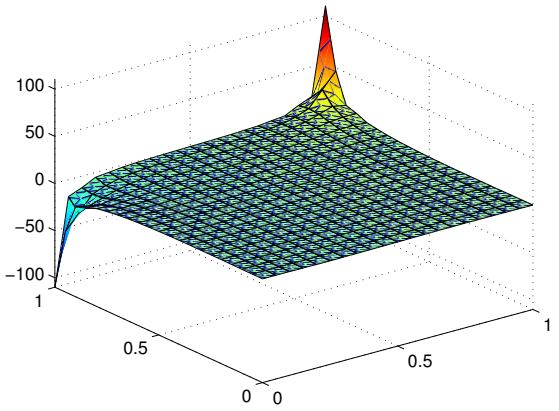
Figure 2.2: Results obtained for Q_2Q_0 element



(a) Streamlines



(b) Velocity field

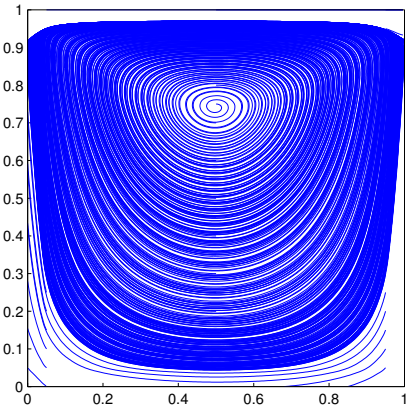


(c) Pressure field

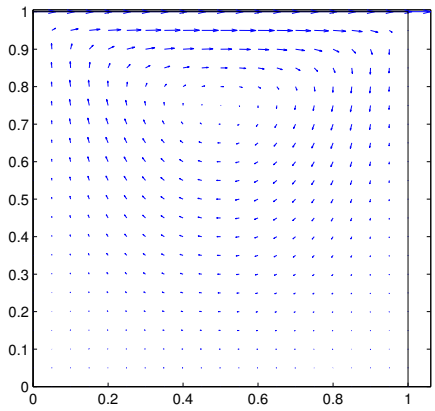
Figure 2.3: Results obtained for Q_2Q_1 element

Both of them are LBB compliant elements and for that reason we can see, as expected, reasonale results for pressure. Note that as pressure field is discretized with discontinuous bilinear elements in figure 2.2 it is not able to capture as well as it is needed the pressure at the upper corners. Figure 2.3 shows better and accurate results for the pressure field due to the fact that the pressure is discretized with continuous bilinear elements.

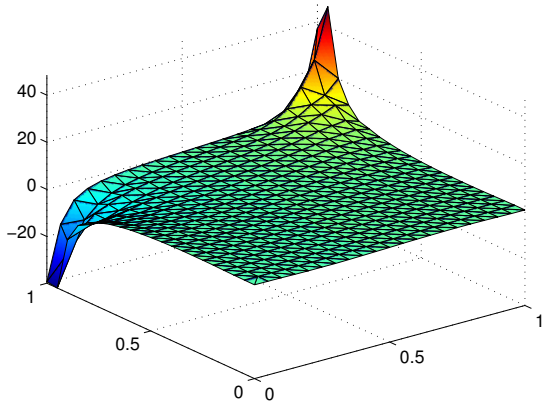
Finally let us recall that there is one interesting option to satisfy the LBB compatibility condition but keeping linear elements. This elements are the so-called Mini element, which discretizes both the velocity field and the pressure with continuous linear elements. The key is that it uses a cubic bubble function for the velocitity to satisfy this LBB condition:



(a) Streamlines



(b) Velocity field



(c) Pressure field

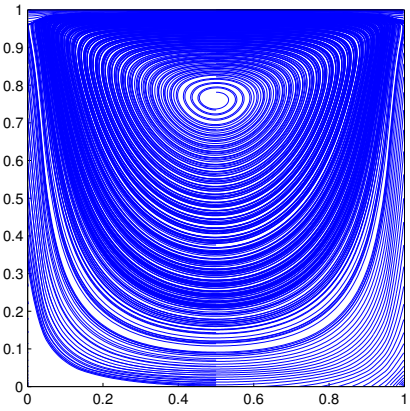
Figure 2.4: Results obtained for MINI($P_1^+P_1$) element

Figure 2.4 reproduces exactly what we have expected. This kind of elements are LBB compliants so it performs reasonable results even though we are working with linear elements. The drawback of using these elements is that the convergence is linear instead of quadratic as it is in the Q_2Q_1 elements (Taylor-Hood element)

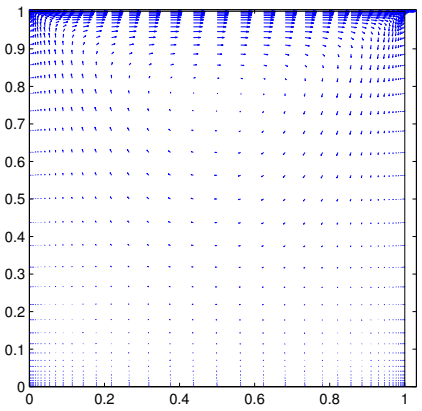
In addition, let us recall that the main features in this case are the symmetry with respect to the vertical centerline and the pressure singularity at the two upper corners as it can be shown in the previous pictures.

(b) Compute the solution of the Stokes problem considering (i) a structured, uniform mesh of Q_2Q_1 elements with 20 elements per side (ii) a structured mesh of 20x20 Q_2Q_1 elements refined near the walls. Comment on the results. Describe the main properties of the velocity and pressure fields. Are there any differences between the solutions obtained with these two meshes? Which one do you thinks the best? Why?

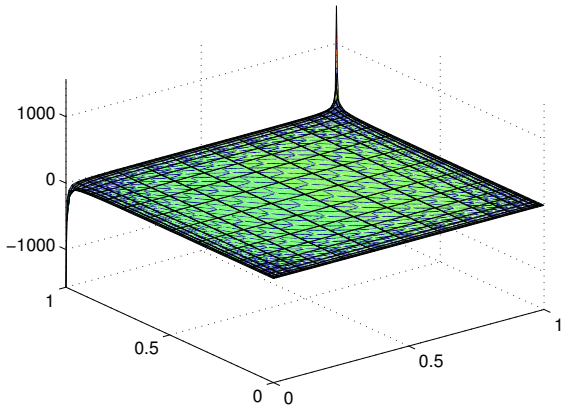
Solutions for the structured uniform mesh are already available in figure 2.3. Let us now show the results obtained for the second case, where the elements are refined near the walls.



(a) Streamlines



(b) Velocity field



(c) Pressure field

Figure 2.5: Results obtained for Q_2Q_0 element with elements refined near the walls

As we can see by comparing between figure 2.3 and figure 2.5 both of them present quasi-identical distribution of streamlines. With regard to both the velocity field and the pressure field, we can observe that adapted mesh is able to capture better what happens in the corners and edges. Moreover, we can see that the pressure field for adapted mesh is smoother and more regular than the one obtained with uniform mesh.

(c) Modify the Stokes code to solve the problem using a GLS stabilized formulation with P_1P_1 elements. Describe the formulation you are using and the choice of the stabilization parameter. Is the method behaving as expected?

The basic idea behind the stabilization procedures is to enforce the positive definiteness of the matrix problem governing the Stokes flow in the Galerkin formulation. Let us first of all define the spaces which play a rol in the problem:

The trial solution space $\boldsymbol{S}$ containing the approximating functions for the velocity:

$$\boldsymbol{S} := \left\{ \boldsymbol{v} \in \boldsymbol{\mathcal{H}^1}\left(\Omega\right) \mid \boldsymbol{v} = \boldsymbol{v}_D \text{ on } \Gamma_D \right\}$$

The weighting functions of the velocity, $\boldsymbol{w}$ belong to $\boldsymbol{\mathcal{V}}$. Functions in this class have the same characteristics as those in class $\boldsymbol{\mathcal{S}}$, except that the weighting functions are required to vanish on Γ_D where the velocity is prescribed. The class $\boldsymbol{\mathcal{V}}$ is thus defined by:

$$\boldsymbol{\mathcal{V}} := \left\{ \boldsymbol{w} \in \boldsymbol{\mathcal{H}^1}\left(\Omega\right) \mid \boldsymbol{w} = \boldsymbol{0} \text{ on } \Gamma_D \right\}$$

Finally, we introduce a space of functions, denoted $\mathcal{Q}$, for the pressure.This functions in $\mathcal{Q}$ are simply required to be square-integrable. So at the end,

$$\mathcal{Q} := \mathcal{L}_2\left(\Omega\right)$$

We denote by $\boldsymbol{\mathcal{S}}^h$ and $\boldsymbol{\mathcal{V}}^h$ th finite dimensional subspaces of $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{V}}$ and $\mathcal{Q}^h$ the finite dimensional subspace of $\mathcal{Q}$.

The problem states: Find $\boldsymbol{v}^h \in \boldsymbol{\mathcal{S}}^h$ and $p^h \in \mathcal{Q}$, such that, for all $(\boldsymbol{w}^h, q^h) \in \boldsymbol{\mathcal{V}}^h \times \mathcal{Q}^h$,

$$\begin{cases} \boldsymbol{a}(\boldsymbol{w}^h, \boldsymbol{v}^h) + \boldsymbol{b}(\boldsymbol{w}^h, q^h) = (\boldsymbol{w}^h, \boldsymbol{b}^h) + (\boldsymbol{w}^h, \boldsymbol{t}^h)_{\Gamma_N}, \\ \boldsymbol{b}(\boldsymbol{v}^h, q^h) - \sum_{e=1}^{n_e l} \tau_e (\boldsymbol{\nabla} q^h, \boldsymbol{\nabla} p^h)_{\Omega^e} = -\sum_{e=1}^{n_e l} \tau_e (\boldsymbol{\nabla} q^h, \boldsymbol{b}^h)_{\Omega^e}. \end{cases}$$

Note that for linear elements the GLS stabilization does not affect the weak form of the momentum equation because the terms involving the second derivatives of the weighting function $\boldsymbol{w}$ vanish. Note also that for the cavity problem we are facing there are neither body forces nor Neumann boundary conditions. So the weak form of the problem yields,

$$\begin{cases} \boldsymbol{a}(\boldsymbol{w}^h, \boldsymbol{v}^h) + \boldsymbol{b}(\boldsymbol{w}^h, q^h) = 0, \\ \boldsymbol{b}(\boldsymbol{v}^h, q^h) - \sum_{e=1}^{n_e l} \tau_e (\boldsymbol{\nabla} q^h, \boldsymbol{\nabla} p^h)_{\Omega^e} = 0. \end{cases}$$

By using the Galerkin discretization of the weak form, one finds that the matrix system which governs the discrete Stokes problem with GLS stabilization for linear elements assumes the following partioned form:

$$\begin{pmatrix} \boldsymbol{K} & \boldsymbol{G} \\ \boldsymbol{G}^T & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{pmatrix}$$

where $\boldsymbol{K}$ is the viscosity matrix, matrix $\boldsymbol{G}$ is the discrete gradient operator, and $\boldsymbol{G}^T$ the divergence gradient operator. All of them already implemented in the code. And finally the matrix $\boldsymbol{D}$ which arises from the discretization of the term $\tau_e(\boldsymbol{\nabla} q^h, \boldsymbol{\nabla} p^h)$ and is the also known stiffness matrix:

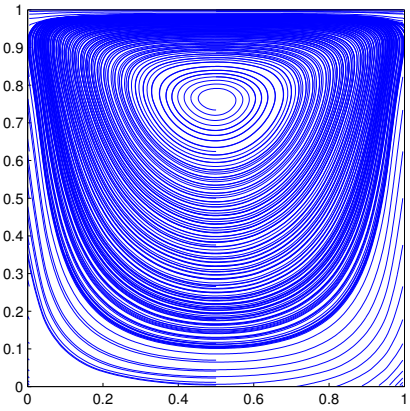
$$\boldsymbol{D} = \boldsymbol{A}^{(e)} \boldsymbol{D}^{(e)} \qquad \Rightarrow \qquad D_{ij}^{(e)} = \int_{\Omega^e} \tau_e \boldsymbol{\nabla} N_i \boldsymbol{\nabla} N_j \, d\Omega$$

The stabilization parameter is chosen as:

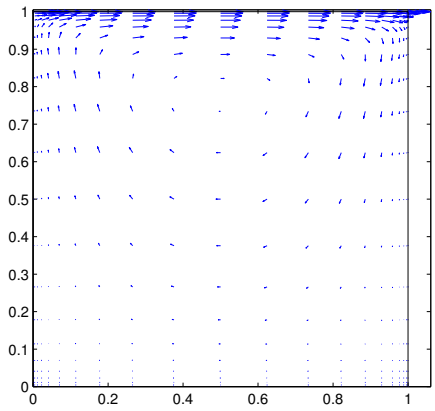
$$\tau_e = \alpha_0 \frac{h_e^2}{4\nu}$$

where h^e is a measure of the element size and ν the viscosity parameter. The parameter α_0 can be tuned but the choice $\alpha_0 = 1/3$ appears to be optimal for linear elements. (See [1] page 288).

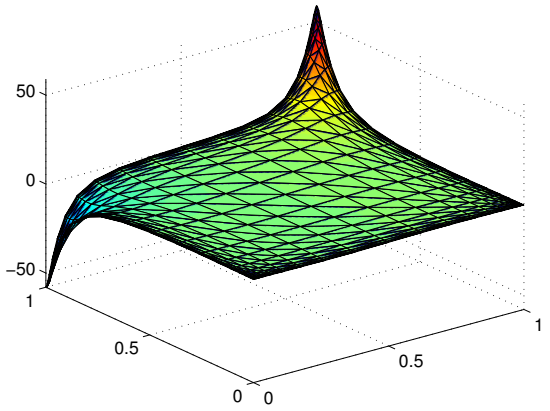
In appendix, there are the different modificed matlab codes which are involved in this section. Let us now show the obtained results when velocity and pressure fields ara discretized with linear continuous elements and using GLS as a stabilization technique.



(a) Streamlines



(b) Velocity field



(c) Pressure field

Figure 2.6: Results obtained for P_1P_1 element with elements refined near the walls and stabilizing the problem with GLS technique

The basic idea behind stabilization procedures is to enforce the positive definiteness of the matrix which governs de steady Stokes problem. An interesting consequence of the GLS stabilization of the Stokes problem is that elements with equal order interpolations, which are unstable in the Galerkin formulation (see figure 2.1) now become stable as it can be observed in figure 2.6.

So we can conclude that the method behaves as expected because the linear/linear three-node triangle has become stable even though it was unstable with the Galerkin formulation used at a).

d) The script *mainNavierStokes.m* can be used to solve the Navier-Stokes equations with Picard method. In order to be able to use it, you must first write a Matlab function *ConvectionMatrix.m* to evaluate the matrix arising from the discretization of the convective term:

$$\boldsymbol{c}(\boldsymbol{w}, \boldsymbol{v}, \boldsymbol{v}^*) = \int_\Omega \boldsymbol{w} \cdot (\boldsymbol{v}^* \cdot \boldsymbol{\nabla}) \boldsymbol{v} d\Omega$$

Solve the Navier-Stokes equations using a structured mesh of $\boldsymbol{Q_2Q_1}$ elements with 20 elements per side. Consider the Reynolds numbers $\boldsymbol{R_e}$ = 100; 500; 1000; 2000 and comment on the results. In particular, discuss the number of iterations needed to achieve convergence, the evolution of the pressure field, the position and strength of the main vortex of the velocity. Compare your results with the ones given in literature.

The matlab implementation of the code can be seen in appendix. First of all let us show a table with the number of iterations needed to achieve the given tolerance:

Re	Number of iterations
100	13
500	29
1000	35
2000	69

As we can see the higher is the Reynolds number, the more iterations are needed to achieve convergence. This fact can be easily explained by using the dimensionless form of the steady Navier-Stokes equation:

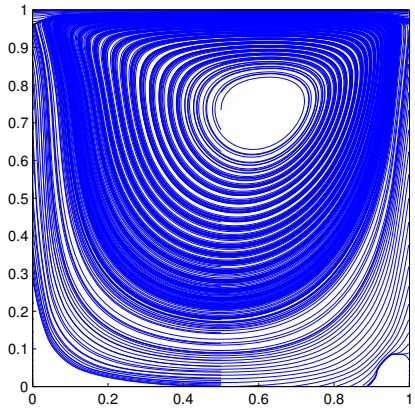
$$(\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{v} - \frac{1}{R_e}\boldsymbol{\nabla^2 v} + \boldsymbol{\nabla} p = 0$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$$

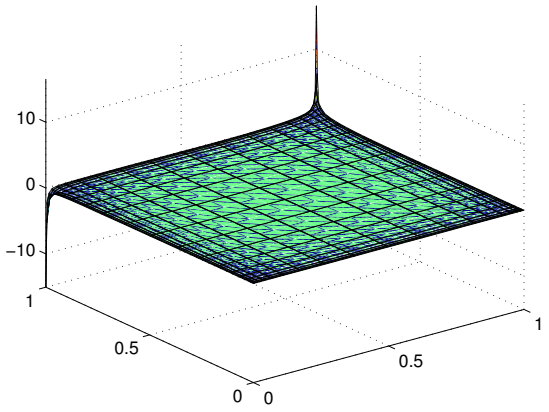
The higher is Reynolds number, the more dominant becomes the convective part, which is the part which causes that the final matrix is nonlinear and non-symmetric, so harder to solve.

Let us now recall that there are two potential sources of numerical instability in the Galerkin finite element solution of this problem, the first is due to the treatment of the convective term and manifests itself in high Reynolds number flows when unresolved internal or boundary layers are present in the solution. The second it the inappropriate combination of interpolation functions for velocity and pressure.

For this problem as Q_2Q_1 elements are chosen, we can assure that they are LBB compliant and satisfy the compatibility condition. Let us now show the results for different Reynolds numbers:

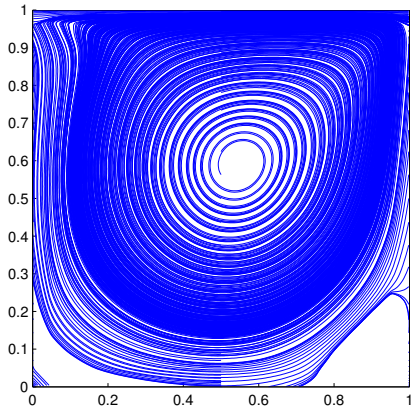


(a) Streamlines

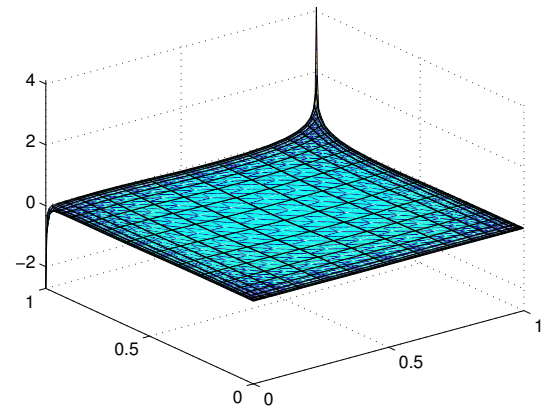


(b) Pressure field

Figure 3.1: Results obtained for Q_2Q_1 element with elements refined near the walls with $R_e = 100$

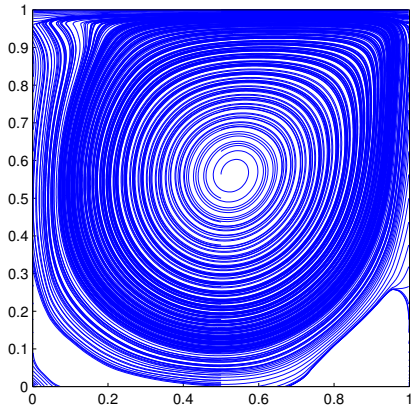


(a) Streamlines

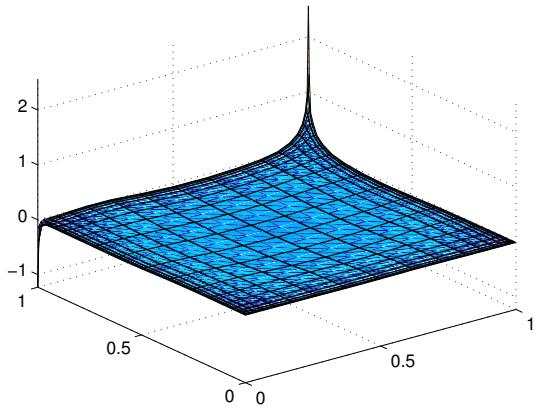


(b) Pressure field

Figure 3.2: Results obtained for Q_2Q_1 element with elements refined near the walls with $R_e = 500$

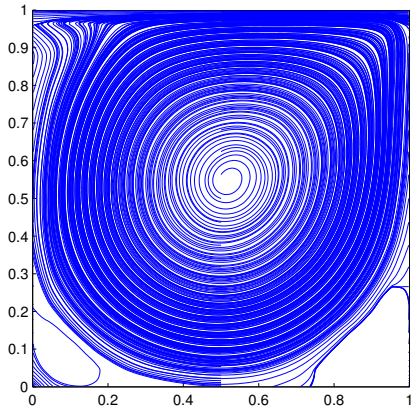


(a) Streamlines

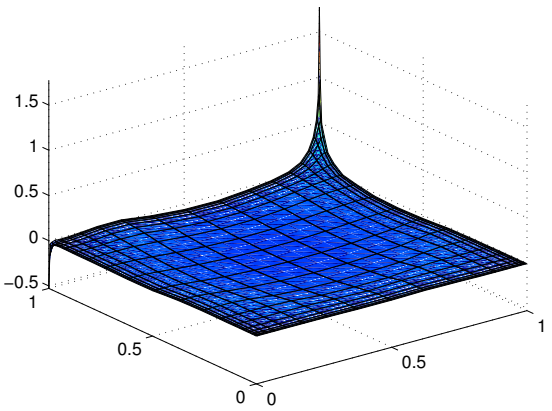


(b) Pressure field

Figure 3.3: Results obtained for Q_2Q_1 element with elements refined near the walls with $R_e = 1000$



(a) Streamlines



(b) Pressure field

Figure 3.4: Results obtained for Q_2Q_1 element with elements refined near the walls with $R_e = 2000$

The influence of the Reynolds number can be clearly seen along the different figures. Note that the position of the main vortex moves towards the center of the cavity when the Reynolds number increases. The development of a secondary vortex in the right bottom corner of the cavity becomes progressively apparent and a third vortex appears at the lower left corner as it can be seen in figure 3.4.

Note also for the pressure field that, while Reynolds number is increasing, its value at the upper corners is decreasing. Regarding to the velocity field, as Reynolds number increases boundary layers are more obvious and the variations of velocity become sharper. This elevated velocity gradient could develop non-physical oscillations for large values of the Reynolds number and a stabilization procedure will be needed.

To end up let us compare our results with the ones given in the literature:

Table 3.1: Add caption

Square cavity		x_1	x_2
Re=100	Present simulation	**0.62**	**0.74**
	Burggraf(1966)	0.62	0.74
	Tuann and Olson(1978)	0.61	0.722
Re=1000	Present simulation	**0.54**	**0.57**
	Ozawa(1975)	0.533	0.569
	Goda(1979)	0.538	0.575

As expected our results are the same that the ones obtained in [1] (see page 312, Table6.2).

References

[1] Donea, J., Huerta, A. *Finite Element Methods for Flow Problems*. Wiley, 2003.

4 Appendix

4.1 MATLAB codes

```
% This program solves the 2D cavity flow Stokes problem


clear; close all; clc

addpath('Func_ReferenceElement')

dom = [0,1,0,1];

% Element type and interpolation degree
% (0: quadrilaterals, 1: triangles, 11: triangles with bubble
   function)
elemV = 1; degreeV = 1; degreeP = 1;
% elemV = 1; degreeV = 2; degreeP = 1;
% elemV = 11; degreeV = 1;  degreeP = 1;
if elemV == 11
    elemP = 1;
else
    elemP = elemV;
end
referenceElement = SetReferenceElementStokes(elemV,degreeV,elemP,
   degreeP);

nx = cinput('Number of elements in each direction',20);
ny = nx;
adapted = 1;
[X,T,XP,TP] = CreateMeshes(dom,nx,ny,referenceElement,adapted);

figure; PlotMesh(T,X,elemV,'b-');
figure; PlotMesh(TP,XP,elemP,'r-');

% Matrices arising from the discretization and the GLS stabilization
[K,G,D,f] = StokesSystem(X,T,XP,TP,referenceElement,nx);
[ndofP,ndofV] = size(G);

[dofDir,valDir,dofUnk,confined] = BC_red(X,dom,ndofV);
nunkV = length(dofUnk);
if confined
   nunkP = ndofP-1;
   disp(' ')
   disp('Confined flow. Pressure on lower left corner is set to zero'
      );
   G(1,:) = [];
   D(1,:) = [];
   D(:,1) = [];
else
   nunkP = ndofP;
```

```
end

f = f - K(:,dofDir)*valDir;
Kred = K(dofUnk,dofUnk);
Gred = G(:,dofUnk);
fred = f(dofUnk);

GLS=1; % If is 1 GLS stabilization is activated
    if GLS==0
    A = [Kred   Gred';
     Gred   zeros(nunkP)];
    else
    A = [Kred   Gred';
     Gred   D];
    end
b = [fred; zeros(nunkP,1)];

sol = A\b;

velo = zeros(ndofV,1);
velo(dofDir) = valDir;
velo(dofUnk) = sol(1:nunkV);
velo = reshape(velo,2,[])';
pres = sol(nunkV+1:end);
if confined
    pres = [0; pres];
end

nPt = size(X,1);
figure;
quiver(X(1:nPt,1),X(1:nPt,2),velo(1:nPt,1),velo(1:nPt,2));
hold on
plot(dom([1,2,2,1,1]),dom([3,3,4,4,3]),'k')
axis equal; axis tight

PlotStreamlines(X,velo,dom);

if degreeP == 0
    PlotResults(X,T,pres,referenceElement.elemP,referenceElement.
       degreeP)
else
    PlotResults(XP,TP,pres,referenceElement.elemP,referenceElement.
       degreeP)
end
```

```
function [K,G,D,f] = StokesSystem(X,T,XP,TP,referenceElement,nx)
% [K,G,f] = StokesSystem(X,T,XP,TP,referenceElement)
% Matrices K, G and r.h.s vector f obtained after discretizing a
   Stokes problem
%
```

```
% 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);
nenP = size(TP,2);

% Number of nodes
nPt_V = size(X,1);
if elem == 11
    nPt_V = nPt_V + nElem;
end
nPt_P = size(XP,1);

% Number of degrees of freedom
nedofV = 2*nenV;
nedofP = nenP;
ndofV = 2*nPt_V;
ndofP = nPt_P;

K = zeros(ndofV,ndofV);
G = zeros(ndofP,ndofV);
f = zeros(ndofV,1);
D = zeros(ndofP,ndofP);

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

    % Element matrices
    [Ke,Ge,De,fe] = EleMatStokes(Xe,ngeom,nedofV,nedofP,ngaus,wgp,N,
        Nxi,Neta,NP,nx);
```

```
54
55      % Assemble the element matrices
56      K(Te_dof, Te_dof) = K(Te_dof, Te_dof) + Ke;
57      G(TPe_dof,Te_dof) = G(TPe_dof,Te_dof) + Ge;
58      D(TPe_dof,TPe_dof) = D(TPe_dof,TPe_dof) + De;
59      f(Te_dof) = f(Te_dof) + fe;
60  end
61
62
63
64
65
66
67  function [Ke,Ge,De,fe] = EleMatStokes(Xe,ngeom,nedofV,nedofP,ngaus,
        wgp,N,Nxi,Neta,NP,nx)
68  % [Ke,Ge,fe] = EleMatStokes(Xe,ngeom,nedofV,nedofP,ngaus,wgp,N,Nxi,
        Neta,NP)
69
70  Ke = zeros(nedofV,nedofV);
71  Ge = zeros(nedofP,nedofV);
72  De = zeros(nedofP,nedofP);
73  fe = zeros(nedofV,1);
74
75  %Compute tau for GLS
76  he=sqrt(2/nx^2);
77  alpha = 1/3;% optimal value for linear elements
78  nu =1;
79  tau_e = alpha *he ^2/(4* nu); % nu is 1.
80
81  % Loop on Gauss points
82  for ig = 1:ngaus
83      N_ig    = N(ig,:);
84      Nxi_ig  = Nxi(ig,:);
85      Neta_ig = Neta(ig,:);
86      NP_ig = NP(ig,:);
87      Jacob = [
88          Nxi_ig(1:ngeom)*(Xe(:,1))        Nxi_ig(1:ngeom)*(Xe(:,2))
89          Neta_ig(1:ngeom)*(Xe(:,1))       Neta_ig(1:ngeom)*(Xe(:,2))
90          ];
91      dvolu = wgp(ig)*det(Jacob);
92      res = Jacob\[Nxi_ig;Neta_ig];
93      nx = res(1,:);
94      ny = res(2,:);
95
96          Ngp = [reshape([1;0]*N_ig,1,nedofV); reshape([0;1]*N_ig,1,
              nedofV)];
97      % Gradient
98      Nx = [reshape([1;0]*nx,1,nedofV); reshape([0;1]*nx,1,nedofV)];
99      Ny = [reshape([1;0]*ny,1,nedofV); reshape([0;1]*ny,1,nedofV)];
100     % Divergence
101     dN = reshape(res,1,nedofV);
```

```
102
103     Ke = Ke + (Nx'*Nx+Ny'*Ny)*dvolu;
104     Ge = Ge - NP_ig'*dN*dvolu;
105     De = De -tau_e*(nx'*nx + ny'*ny)*dvolu;
106     x_ig = N_ig(1:ngeom)*Xe;
107     f_igaus = SourceTerm(x_ig);
108     fe = fe + Ngp'*f_igaus*dvolu;
109 end
```

```
1  function C = ConvectionMatrix(X,T,referenceElement,velo)
2  % [K,G,f] = StokesSystem(X,T,XP,TP,referenceElement)
3  % Matrices K, G and r.h.s vector f obtained after discretizing a
       Stokes problem
4  %
5  % X,T: nodal coordinates and connectivities for velocity
6  % referenceElement: reference element properties (quadrature, shape
       functions...)
7  % velo: convective velocity
8
9  elem = referenceElement.elemV;
10 ngaus = referenceElement.ngaus;
11 wgp = referenceElement.GaussWeights;
12 N = referenceElement.N;
13 Nxi = referenceElement.Nxi;
14 Neta = referenceElement.Neta;
15 ngeom = referenceElement.ngeom;
16
17 % Number of elements and number of nodes in each element
18 [nElem,nenV] = size(T);
19
20 % Number of nodes
21 nPt_V = size(X,1);
22 if elem == 11
23     nPt_V = nPt_V + nElem;
24 end
25
26 % Number of degrees of freedom
27 nedofV = 2*nenV;
28 ndofV = 2*nPt_V;
29
30 %Allocate
31 C = zeros(ndofV,ndofV);
32
33 % Loop on elements
34 for ielem = 1:nElem
35     % Global number of the nodes in element ielem
36     Te = T(ielem,:);
37     % Coordinates of the nodes in element ielem
38     Xe = X(Te(1:ngeom),:);
39     % Degrees of freedom in element ielem
```

```
    Te_dof = reshape([2*Te-1; 2*Te],1,nedofV);
    % Ve: element nodes' velocity
    Ve = velo(T(ielem,:),:);
    % Element matrices
    Ce = zeros(nedofV,nedofV);
        % Loop on Gauss points
        for igaus = 1:ngaus
            % Shape functions on Gauss point igaus
            N_ig    = N(igaus,:);
            Nxi_ig  = Nxi(igaus,:);
            Neta_ig = Neta(igaus,:);
            % Jacobian matrix on the Gauss point
            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(igaus)*det(Jacob);
            % Shape functions' derivatives in global coordinates
            res = Jacob\[Nxi_ig;Neta_ig];
            nx = res(1,:);
            ny = res(2,:);
            % Shape functions in 2D
            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)];
            % Velocity on point igaus
            v_igaus = N_ig*Ve;
            % Contribution to element matrix
            Ce = Ce + Ngp'*(v_igaus(1)*Nx+v_igaus(2)*Ny)*dvolu;
        end
    %Assembly
    C(Te_dof, Te_dof) = C(Te_dof, Te_dof) + Ce;
end
end
```