POLYTECHNIC UNIVERSITY OF CATALONIA
MASTER OF SCIENCE IN COMPUTATIONAL MECHANICS

Finite Element in Fluids
Cavity and Stoke flow Assignment

Student: Chiluba Isaiah Nsofu

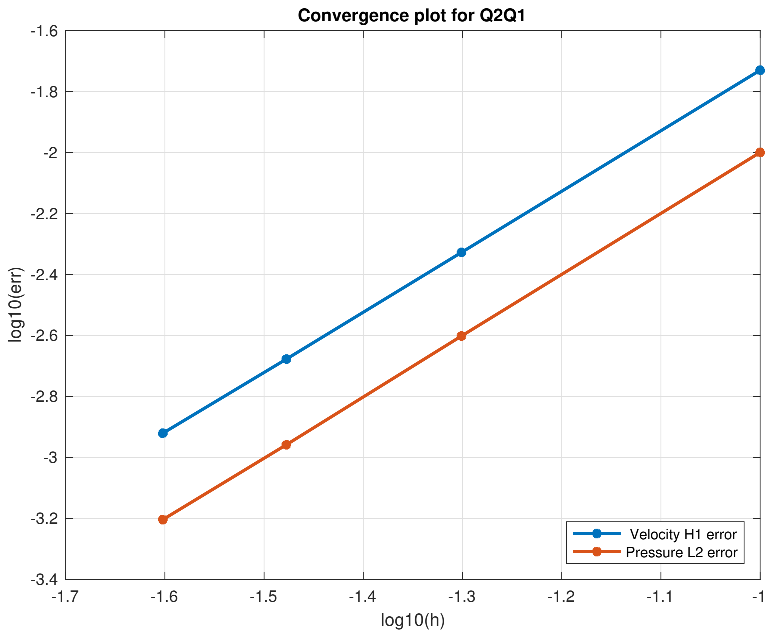
Date: 18 May, 2018

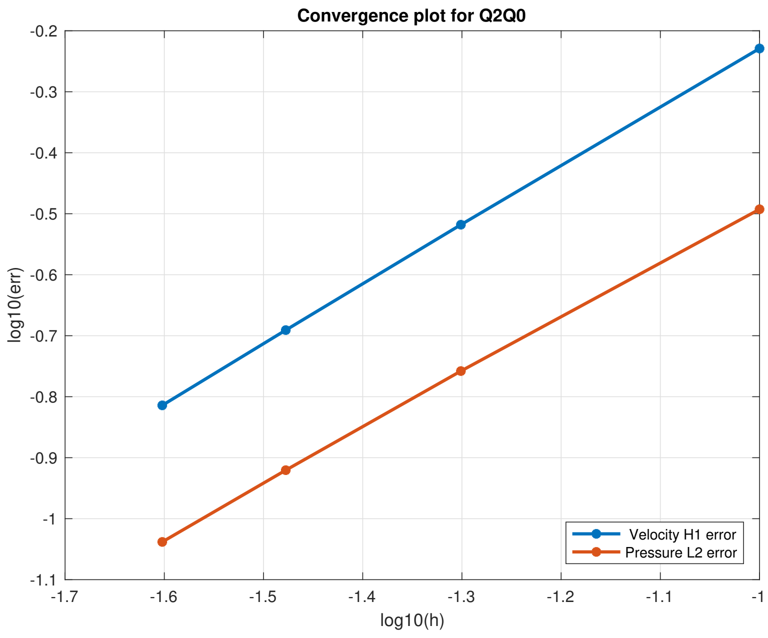
Contents

1 Question 1 . . . 1
1.1 b . . . 2
2 Question 2 . . . 4
3 Steady Navier-Stokes problem . . . 6
4 Appendix . . . 9
4.1 MATLAB codes . . . 9

1 Question 1

The matlab code for computing the velocity and pressure errors is attached in the appendix of this report. For all the convergence plots that are plotted in this report the following values were used for the element size, 1/10, 1/20,1,30 and 1/40. These values also corresponds to the convergence plots presented in the literature book for the same topic.





From the convergence plots we can see that the error plots are behaving as expected. This because the velocity converges to the exact solution for the Q2Q1. This is due to the fact that this element satisfy the LBB condition which allow it to deliver good solution. On the other the other the Q1Q0 does not satisfy the LBB condition hence a slight variation in the convergence plot as seen in figure 1.

1.1 b

Solve the problem using P1P1 elements with an stabilised formulation. Check the convergence of this approximation.

The stabilization formulation was implemented using GLS. The basic idea behind the stabilization procedures is the enforcement of positive definiteness of the matrix problem governing the Stokes flow in the Galerkin formulation. This is done by modifying the Galerkin weak form of the incompressibility condition in order to render non-zero the diagonal term resulting from the incompressiblity condition. Doing so allows for the circumventing of the inf-sup condition. Note that generally, the weak form is modified by adding the terms coming from a minimization of a least squares form. More detailed information about the use of GLS can be found on page 287-288 of the reference book. Nevertheless in this report we will illustrate the key points in terms of the implementation. With respect to the functional spaces we define the following spaces for both the weighting function $\boldsymbol{w}$ and velocity solution $\boldsymbol{v}$.

$$\boldsymbol{\mathcal{V}} := \left\{\boldsymbol{w} \in \boldsymbol{\mathcal{H}^1}(\Omega) \mid \boldsymbol{w} = \boldsymbol{0} \text{ on } \Gamma_D\right\} \quad \text{and} \quad \boldsymbol{S} := \left\{\boldsymbol{v} \in \boldsymbol{\mathcal{H}^1}(\Omega) \mid \boldsymbol{v} = \boldsymbol{v}_D \text{ on } \Gamma_D\right\} \tag{1.1}$$

Furthermore, we introduce a space of functions, denoted $\mathcal{Q}$, for the pressure as $\mathcal{Q} := \mathcal{L}_2(\Omega)$. Denoting that $\boldsymbol{\mathcal{S}}^h$ and $\boldsymbol{\mathcal{V}}^h$ be subspaces of $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{V}}$ and that $\mathcal{Q}^h$ be the subspace of $\mathcal{Q}$ then the Stokes problem to solved is stated as: Find $\boldsymbol{v}^h \in \boldsymbol{S}^h$ and $p^h \in \mathcal{Q}$, such that, for all $(\boldsymbol{w}^h, q^h) \in \boldsymbol{\mathcal{V}}^h \times \mathcal{Q}^h$,

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

For the case of linear elements, the GLS stabilization technique does not affect the weak form of the momentun equation because terms involving second derivatives cancel. So the weak form of the problem yields,

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

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:

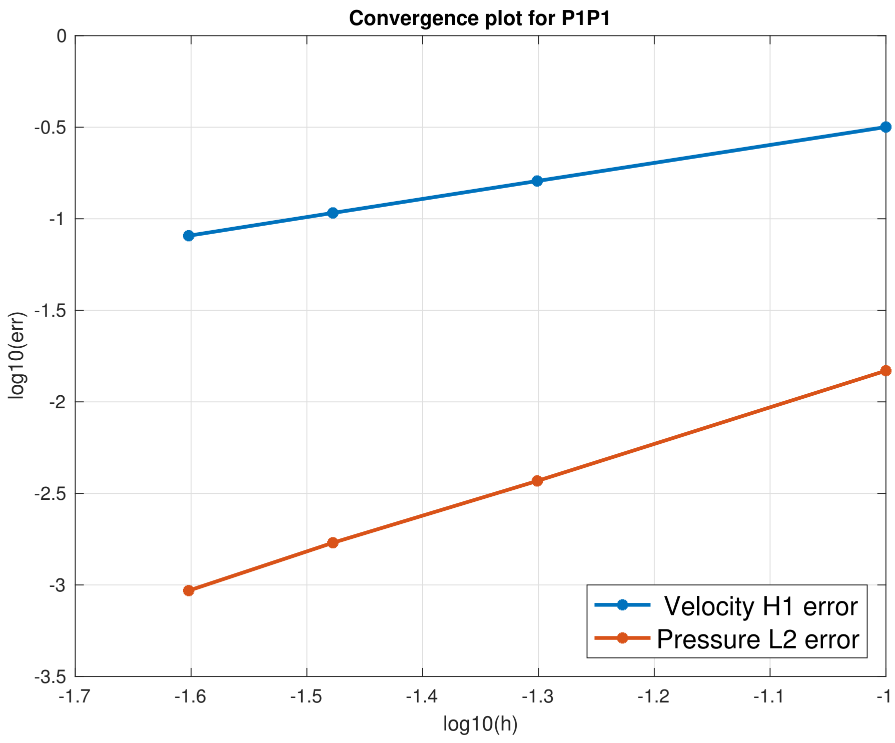
$$\boldsymbol{D} = \boldsymbol{A}^{(e)} \boldsymbol{D}^{(e)} \qquad \Rightarrow \qquad D_{ij}^{(e)} = \int_{\Omega^e} \tau_e \boldsymbol{\nabla} N_i \boldsymbol{\nabla} N_j \; d\Omega$$

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).

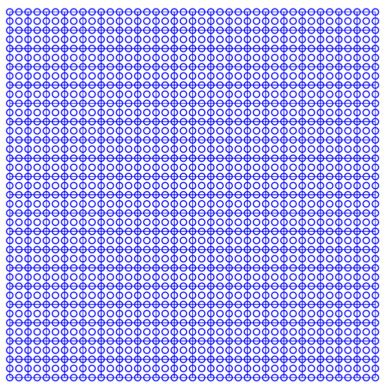
In appendix, there are the matlab codes pertaining to this section. Let us now show the obtained results when velocity and pressure fields ara discretized with linear continuous elements and using GLS as a stabilization technique.



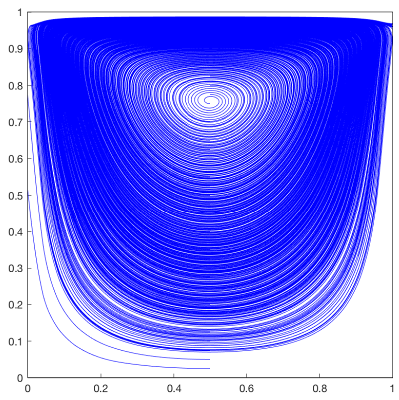
Both the velocity and pressure converged to a nice solutions as shown in the above figure. This indeed shows the importance of stabilized formulation. Generally this element will have difficulties in solving this problem if stabilization is not utilized. This is due to the fact this element does not satisfy the LBB condition, which guarantees the uniqueness and existence of solution. Therefore, when used without stabilization it presents a spurious node-to-node response for the pressure field. This can ultimately affect the convergence of the solution and it will not appear like the one shown in the above figure.

2 Question 2

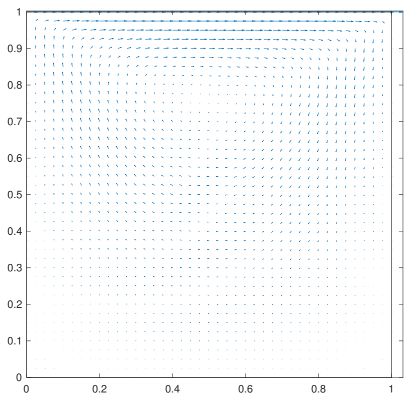
(b) Compute the solution of the Stokes problem considering (i) a structured, uniform mesh of Q_2Q_1 elements with 20 elements per side (ii) a structured mesh of 20x20 Q_2Q_1 elements refined near the walls. Comment on the results. Describe the main properties of the velocity and pressure fields. Are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



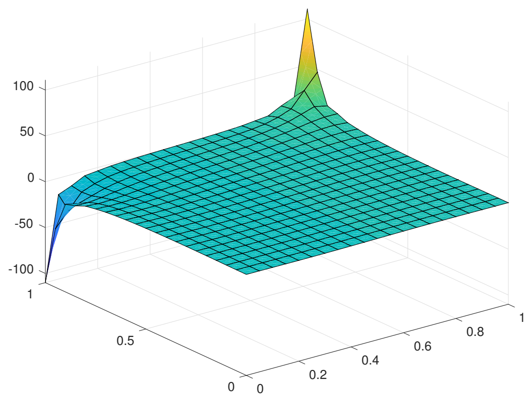
(a) Mesh



(b) Streamlines

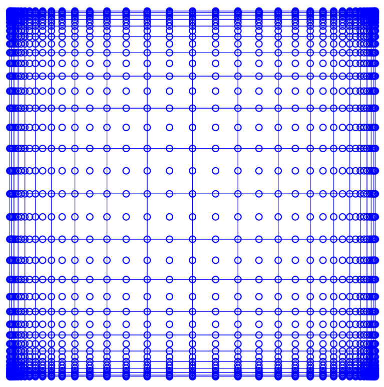


(c) Velocity field

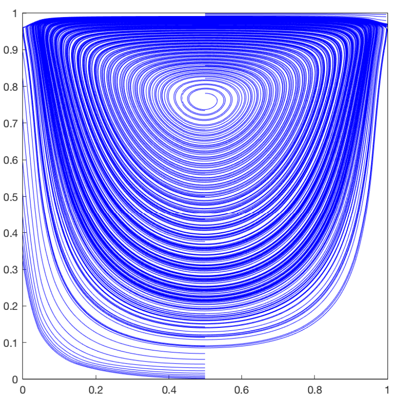


(d) Pressure field

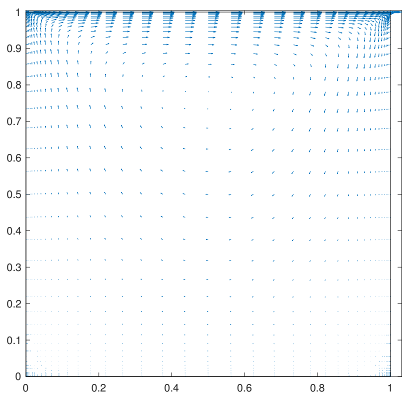
Figure 2.1: Results obtained for Q_2Q_1 element



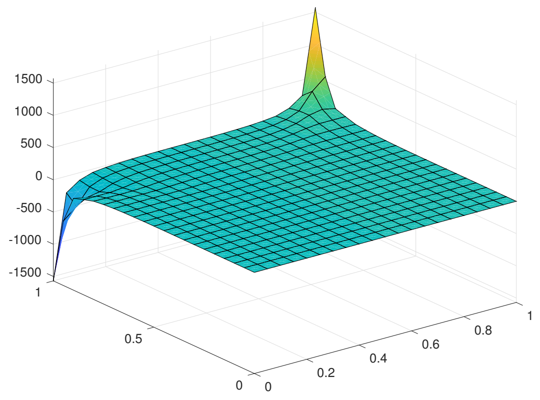
(a) Mesh



(b) Streamlines



(c) Velocity field



(d) Pressure field

Figure 2.2: Results obtained for Q_2Q_1 element with elements refined near the walls

As we can see by comparing between figure 2.1 and figure 2.2 both of them presents an almost identical distribution of streamlines and the solution for the pressure field is shown to be reliable in both figures as the element is LBB-conforming.On the other we also note the symmetric nature of the velocity solution in both figures. Furthermore, with regards 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.
From the results it easy to see that there is a discontinuity in the boundary conditions at the two upper corners of the cavity. This discontinuity will greatly affect the results it is not well handled. One way of solving this problem is mesh refinement around the affected part. Therefore the refined a refined mesh stands a better chance of producing better results the hence it will be a good choice.

3 Steady Navier-Stokes problem

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 code implementation for the *ConvectionMatrix.m* is attached in the appendix and the results for different Reynolds numbers are as follows.

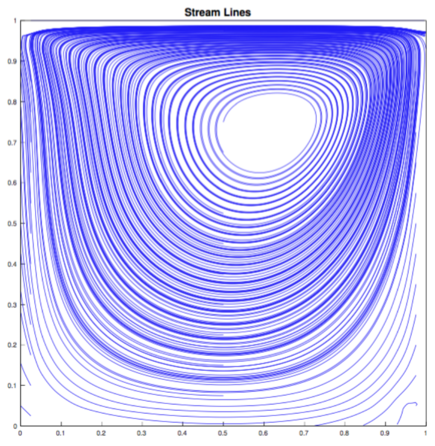


Figure 3.1: Streamlines

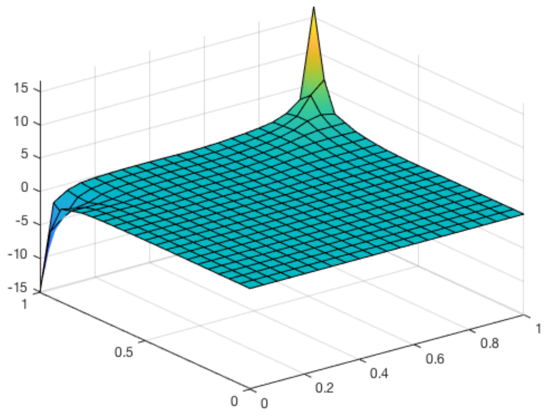
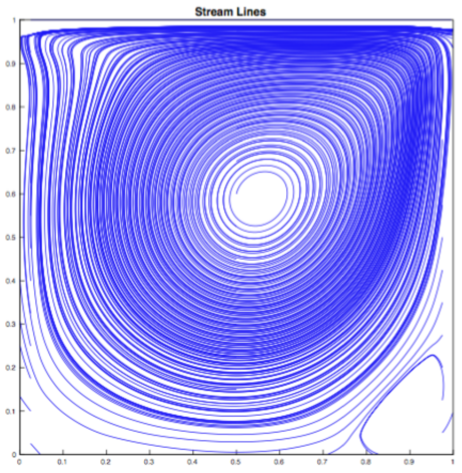
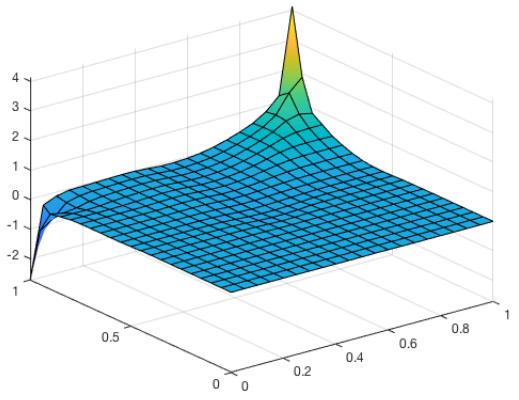


Figure 3.2: Pressure field

Figure 3.3: Results obtained for Q_2Q_1 element with elements refined near the walls with $R_e = 100$



(a) Streamlines



(b) Pressure field

Figure 3.4: Results obtained for Q_2Q_1 element with elements refined near the walls with $R_e = 500$

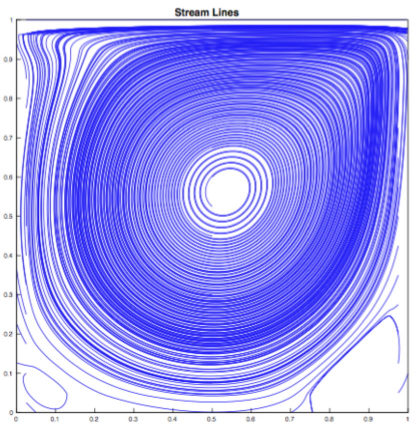


Figure 3.5: Streamlines

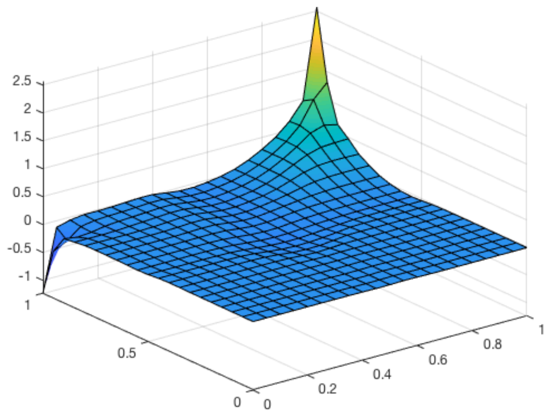


Figure 3.6: Pressure field

Figure 3.7: Results obtained for Q_2Q_1 element with elements refined near the walls with $R_e = 1000$

In the table below we present the Reynolds number and the corresponding number of iterations needed that are need to achieve the given tolerance:

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

From this table we can see that increasing the value of the Reynolds number also increases the number of iterations that are are needed to achieve convergence. Furthermore we can say that at higher is Reynolds number, the convective part becomes more dominant which leads to the which results in nonlinear and non-symmetric form of the final matrix is a situation which is difficult to solve. Since the Q_2Q_1 elements are used the pressure evolution is well satisfied. This because the Q_2Q_1 are LBB compliant and satisfy the compatibility condition as well. As for the pressure field increasing the Reynolds number results in a decreased value at the upper corners. With regards to the main vortex we 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 3.7. Finally when we compare the results produced with the literature (see the reference book page 312, Table6.2).based results the are similar.

Table 3.1: Comparison of results

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

References

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

4 Appendix

MATLAB codes

```matlab
% This Matlab function computes erros for both velocity and pressure
function [errV_L2,errV_H1,errP] = ComputeError(velo,pres,X,T,XP,TP,
   referenceElement)


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


errV_L2 = 0; normV_L2 = 0;
errV_H1 = 0; normV_H1 = 0;
errP = 0; normP = 0;
% Loop on elements
for ielem = 1:size(T,1)
    % 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),:);
    % Solution at the element's nodes
    velo_e = velo(Te,:);
    pres_e = pres(TPe);

    % Element matrices
    [errV_L2_e,errV_H1_e,errP_e,normV_L2_e,normV_H1_e,normP_e] = ...
        ElemError(velo_e,pres_e,Xe,ngeom,ngaus,wgp,N,Nxi,Neta,NP);

    % Add the element contribution to global error
    errV_L2 = errV_L2 + errV_L2_e; normV_L2 = normV_L2 + normV_L2_e;
    errV_H1 = errV_H1 + errV_H1_e; normV_H1 = normV_H1 + normV_H1_e;
    errP = errP + errP_e; normP = normP + normP_e;
end
errV_L2 = sqrt(errV_L2)/sqrt(normV_L2);
errV_H1 = sqrt(errV_H1)/sqrt(normV_H1);
errP = sqrt(errP)/sqrt(normP);
```

```
function [errV_L2_e,errV_H1_e,errP_e,normV_L2_e,normV_H1_e,normP_e] =
    ElemError(velo_e,pres_e,Xe,ngeom,ngaus,wgp,N,Nxi,Neta,NP)
%

errV_L2_e = 0; normV_L2_e = 0;
errV_H1_e = 0; normV_H1_e = 0;
errP_e = 0;  normP_e = 0;


% Loop on Gauss points
for ig = 1:ngaus
    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,:);

        % Exact solution
    pt = N_ig(1:ngeom)*Xe;
    [u,v,u_x,u_y,v_x,v_y,p] = ExactSol(pt);
    %
    normV_L2_e = normV_L2_e + (u^2+v^2)*dvolu;
    normV_H1_e = normV_H1_e + (u_x^2 + v_y^2)*dvolu;
    normP_e = normP_e + (p^2)*dvolu;
    % Computed solution
    uh = N_ig*velo_e(:,1); uh_x = nx*velo_e(:,1); uh_y = ny*velo_e
       (:,1);
    vh = N_ig*velo_e(:,2); vh_x = nx*velo_e(:,2); vh_y = ny*velo_e
       (:,2);
    ph = NP_ig*pres_e;
    % Error
    errV_L2_e = errV_L2_e + ((u-uh)^2 + (v-vh)^2)*dvolu;
    errV_H1_e = errV_H1_e + ( (u_x - uh_x)^2 + (v_y - vh_y)^2)*dvolu;
    errP_e = errP_e + ( (p - ph)^2 )*dvolu;
end
```

```
function [TL,TR] = GLSMethod(XP,TP,tau,referenceElement)
% [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...)

ngaus = referenceElement.ngaus;
wgp = referenceElement.GaussWeights;
NP = referenceElement.NP;
NPxi = referenceElement.NPxi;
NPeta = referenceElement.NPeta;

% Number of elements and number of nodes in each element
[nElem,nenP] = size(TP);

% Number of nodes
nPt_P = size(XP,1);

% Number of degrees of freedom
nedofP = nenP;
ndofP = nPt_P;

TL = zeros(ndofP,ndofP);
TR = zeros(ndofP,1);

% Loop on elements
for ielem = 1:nElem
    % Global number of the nodes in element ielem
    TPe = TP(ielem,:);
    % Coordinates of the nodes in element ielem
    XPe = XP(TPe,:);
    % Degrees of freedom in element ielem

    % Element matrices
    [TLe,TRe] = EleMatGLS(XPe,nedofP,ngaus,wgp,NP,NPxi,NPeta,tau);

    % Assemble the element matrices
    TL(TPe, TPe) = TL(TPe, TPe) + TLe;
    TR(TPe) = TR(TPe) + TRe;
end

function [TLe,TRe] = EleMatGLS(XPe,nedofP,ngaus,wgp,NP,NPxi,NPeta,tau
   )
% [TLe,TRe] = EleMatStokes(Xe,ngeom,nedofV,nedofP,ngaus,wgp,N,Nxi,
   Neta,NP)

TLe = zeros(nedofP,nedofP);
TRe = zeros(nedofP,1);
% Loop on Gauss points
for ig = 1:ngaus
    NP_ig = NP(ig,:);
    NPxi_ig = NPxi(ig,:);
    NPeta_ig = NPeta(ig,:);
    Jacob = [
        NPxi_ig*(XPe(:,1))       NPxi_ig*(XPe(:,2))
        NPeta_ig*(XPe(:,1))  NPeta_ig*(XPe(:,2))
        ];
    dvolu = wgp(ig)*det(Jacob);
    res = Jacob\[NPxi_ig;NPeta_ig];
    % Gradient
%     NPx = res(1,:);
%     NPy = res(2,:);

%       NPgp = [reshape([1;0]*NP_ig,1,nedofP); reshape([0;1]*NP_ig,1,
   nedofP)];
    % Gradient
%     NPx = [reshape([1;0]*nx,1,nedofP); reshape([0;1]*nx,1,nedofP)];
%     NPy = [reshape([1;0]*ny,1,nedofP); reshape([0;1]*ny,1,nedofP)];

    TLe = TLe + tau*(res'*res)*dvolu;
    x_ig = NP_ig*XPe;
    f_igaus = SourceTerm(x_ig);
    TRe = TRe + tau*res'*f_igaus*dvolu;
end
```

```
function C = ConvectionMatrix(X,T,referenceElement,velo)
% C = ConvectionMatrix(X,T,referenceElement,velo)
%
% X,T: nodal coordinates and connectivities for velocity
% referenceElement: reference element properties (quadrature, shape
   functions...)
% velo: velocity field

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

```matlab

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),:);
    % Velocities for the element
    Ve = velo(Te(1:ngeom),:);
    % Degrees of freedom in element ielem
    Te_dof = reshape([2*Te-1; 2*Te],1,nedofV);

    % Element matrices
    Ce = EleMatC(Xe,ngeom,nedofV,ngaus,wgp,N,Nxi,Neta,Ve);

    % Assemble the element matrices
    C(Te_dof, Te_dof) = C(Te_dof, Te_dof) + Ce;
end



function Ce = EleMatC(Xe,ngeom,nedofV,ngaus,wgp,N,Nxi,Neta,Ve)
% Ce = EleMatC(Xe,ngeom,nedofV,ngaus,wgp,N,Nxi,Neta,NP)

Ce = zeros(nedofV,nedofV);
% Loop on Gauss points
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,:);

        % Gradient
    Ngp = [reshape([1;0]*N_ig,1,nedofV); reshape([0;1]*N_ig,1,nedofV)
       ];
    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;

    Ce = Ce + Ngp'*(v_ig(1)*Nx+v_ig(2)*Ny)*dvolu;
end
```