UPC, CIMNE, ETSECCP

Finite Element in Fluids - Assignment 2

Rafael Pacheco
77128580N

June 5, 2017

Universitat Politècnica De Catalunya BARCELONATECH
Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona
Centre Internacional de Mètodes Numèrics en Enginyeria

1 Assignment 2

1.1 a) Stokes, $[Q_2Q_0, Q_2Q_1, P_1P_1, P_1^+P_1]$, 20 elements per side, uniform structured mesh.

In order to have a uniquely defined solution, the following system will be solved:

$$\begin{bmatrix} K & G \\ G^T & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ h \end{bmatrix} \tag{1.1}$$

This system is non-singular if $kernel(G) = 0$, to fulfil this, the pressure and velocity have to accomplish the LBB condition:

$$\inf_{q^h \in \mathcal{Q}^h} \inf_{\omega^h \in \mathcal{V}^h} \frac{q^h, \nabla \cdot \omega^h}{||q||_0 ||\omega^h||_1} \geq \alpha > 0 \tag{1.2}$$

where $(\mathcal{Q}, \mathcal{V})$ are the pair of spaces for the approximate solution (u^h, p^h)

1.1.1 Q_2Q_0

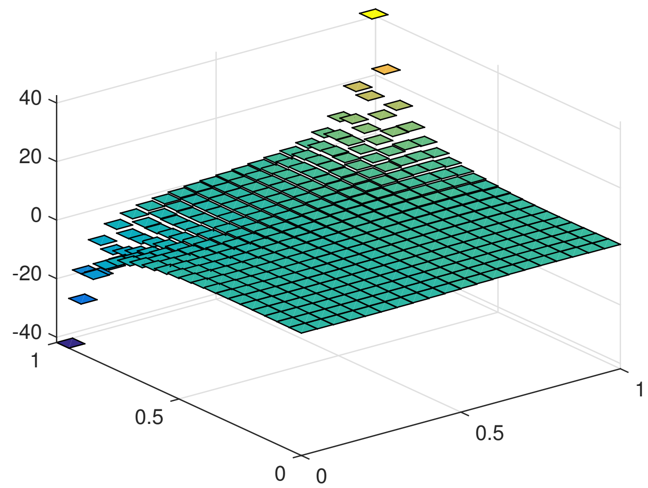

Figure 1.1: Pressure field for Q_2Q_0

This element does fulfil the LBB compatibility condition. But it cannot capture correctly the behaviour of the pressure since it is discretized by discontinuous bilinear elements.

1.1.2 Q_2Q_1

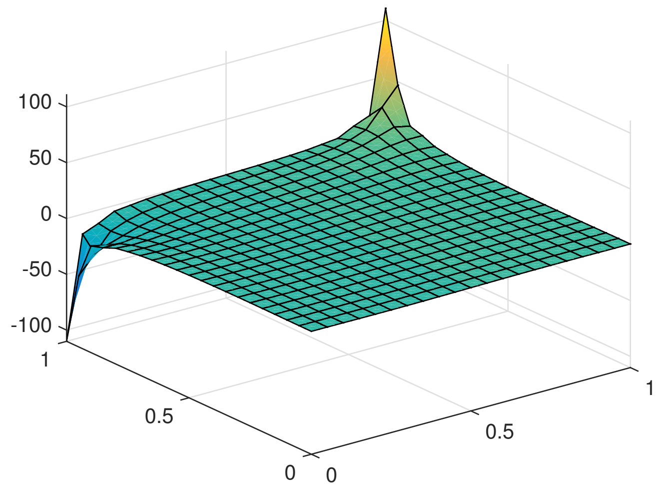

Figure 1.2: Pressure field for Q_2Q_1

This element does fulfil the LBB compatibility condition. The difference with the previous one is that it captures correctly the behaviour of the pressure since it is discretized by continuous bilinear elements.

1.1.3 P_1P_1

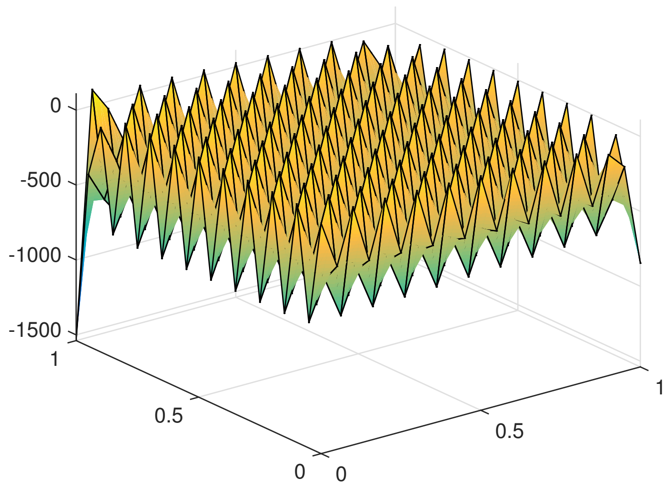

Figure 1.3: Pressure field for P_1P_1

This element does not fulfil the LBB compatibility condition.

1.1.4 $P_1^+P_1$

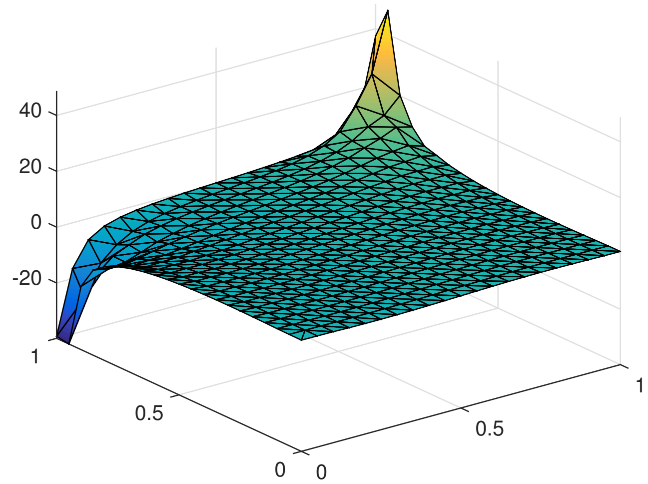

Figure 1.4: Pressure field for Mini($P_1^+P_1$)

This element does fulfil the LBB compatibility condition. This element uses continuous linear elements by using a cubic bubble function for the velocity so it can fulfil so-called LBB compatibility condition.

The main difference with the Q_2Q_1 element resides in that the convergence is linear instead of quadratic, which means it is more time-consuming to perform an analysis with this element.

1.2 B) STOKES, Q_2Q_1 , 20 ELEMENTS PER SIDE, UNIFORM STRUCTURED NON-REFINED MESH VS REFINED MESH.

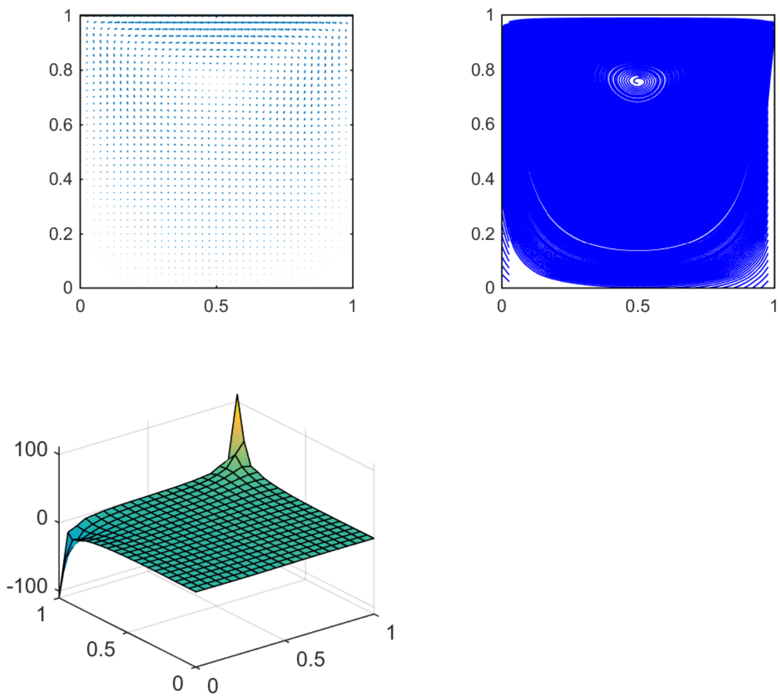

Figure 1.5: Non-refined mesh.

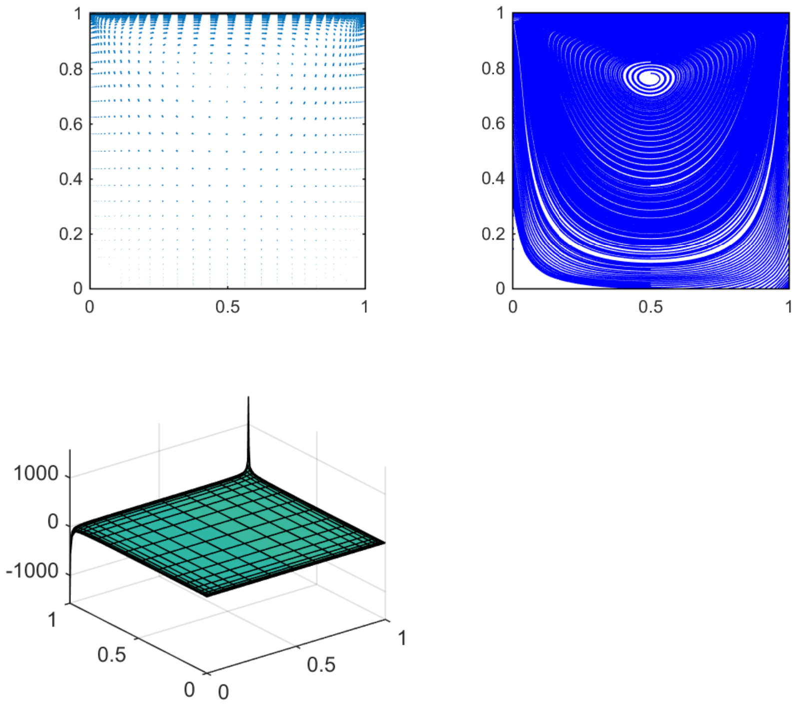

Figure 1.6: Refined mesh.

The best results due to the singularity of the case are shown by the refined mesh.

1.3 C) STOKES + GLS

Velocity trial solution space $\mathscr{S}$ and weighting functions space $\mathscr{V}$, pressure $\mathscr{Q}$.

$$\begin{aligned} &\mathscr{S} := \{u \in \mathscr{H} \quad | \quad u = u_D \quad \text{on} \quad \partial\Omega_D\} \\ &\mathscr{V} := \{\omega \in \mathscr{H} \quad | \quad \omega = 0 \quad \text{on} \quad \partial\Omega_D\} \\ &\mathscr{Q} := \mathscr{L}_2(\Omega) \end{aligned} \tag{1.3}$$

The stokes problem consists on finding the velocity and pressure such that:

$$\begin{aligned} -\nu\Delta u + \nabla p = f \quad &\text{in} \quad \Omega \\ \nabla \cdot u = 0 \quad &\text{in} \quad \Omega \\ u = u_D \quad &\text{on} \quad \partial\Omega_D \\ -pn + \nu(n \cdot \nabla)u = t \quad &\text{on} \quad \partial\Omega_N \end{aligned} \tag{1.4}$$

The discretized weak form would be:

$$\begin{aligned} &a(\omega^h, u^h) + b(\omega^h, q^h) = (\omega^h, b^h) + (\omega^h, t^h)_{\partial\Omega_N} \\ &b(u^h, q^h) - \textstyle\sum_{e=1}^{n_{el}} \tau_e (\nabla q^h, \nabla p^h)_{\Omega^e} = -\textstyle\sum_{e=1}^{n_{el}} \tau_e (\nabla q^h, \nabla b^h)_{\Omega^e} \end{aligned} \tag{1.5}$$

And assuming no body forces and no Neumann B.C. , the weark form is:

$$\begin{aligned} &a(\omega^h, u^h) + b(\omega^h, q^h) = 0 \\ &b(u^h, q^h) - \textstyle\sum_{e=1}^{n_{el}} \tau_e (\nabla q^h, \nabla p^h)_{\Omega^e} = 0 \end{aligned} \tag{1.6}$$

Now the system to solve is:

$$\begin{bmatrix} K & G \\ G^T & D \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{1.7}$$

where D is the stabilization term matrix:

$$D_{ij} = \int_{\Omega}^{e} \tau \nabla N_i \nabla N_j d\Omega \tag{1.8}$$

And regarding the stabilization factor:

$$\tau = \alpha_0 \frac{h_e^2}{4\nu} \tag{1.9}$$

The optimal choice for linear elements as requested is for $\alpha = 1/3$.

Same order interpolation element with GLS method, no longer are unstable.

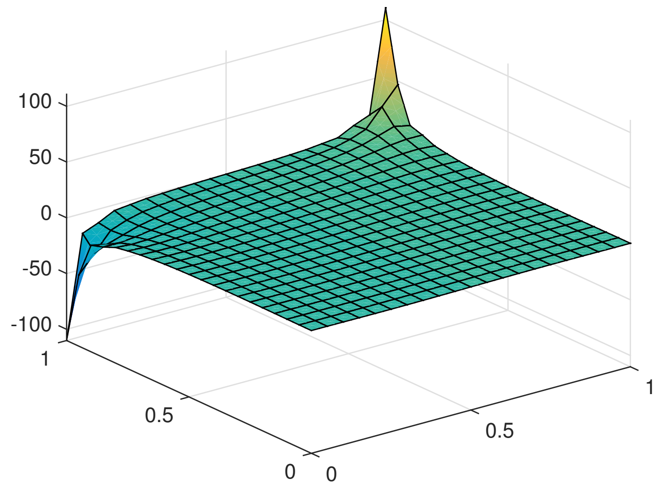

Figure 1.7: $P_1 P_1$ with GLS.

1.4 D) NAVIER STOKES

The element chosen (Q_2Q_1) fulfils the LBB condition. However, the greater the Reynolds number, the more iterations needed because of the dominance from the convective term increases.

Re	Iterations
100	13
500	26
1000	68
2000	100

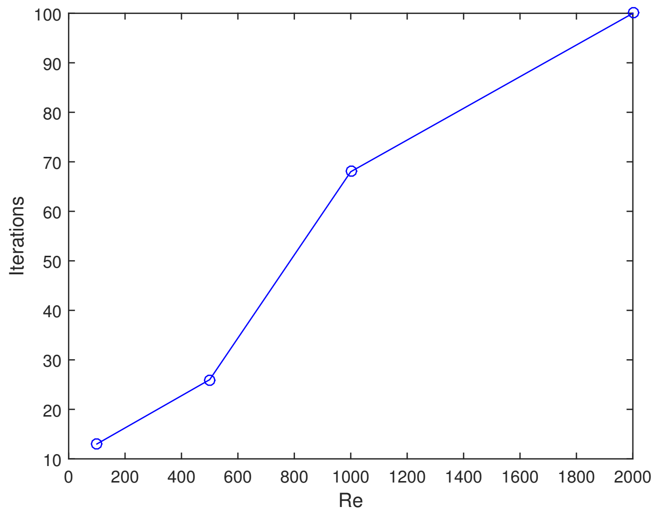

Figure 1.8: Iterations vs Reynolds number.

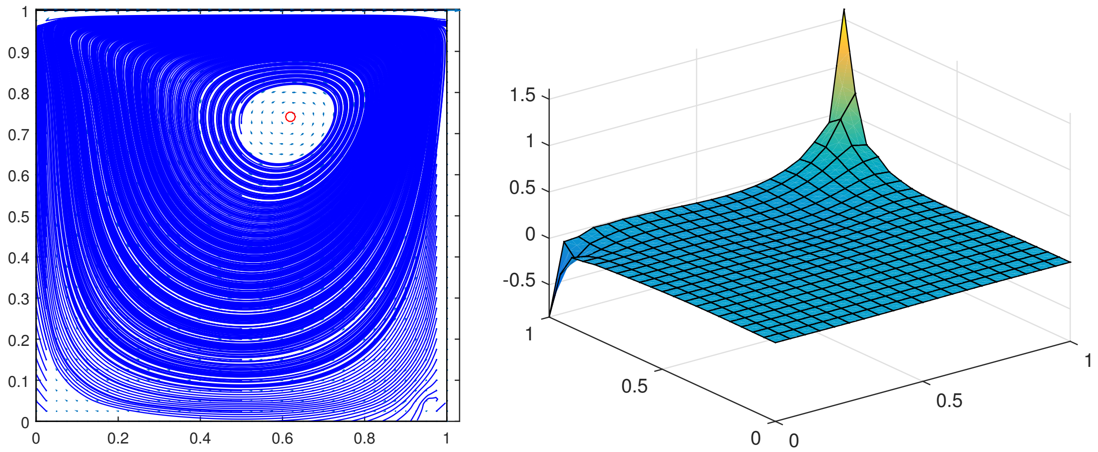

Figure 1.9: Re = 100 .

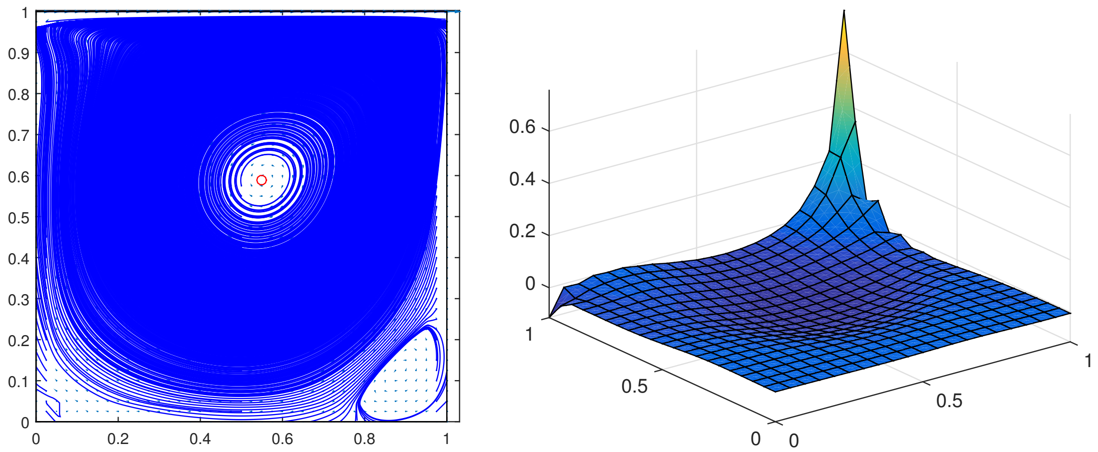

Figure 1.10: Re = 500 .

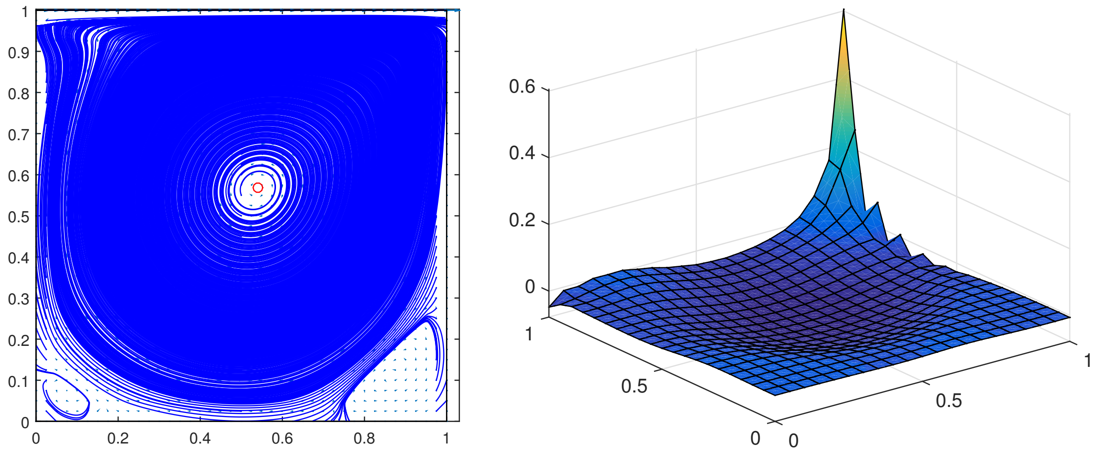

Figure 1.11: Re = 1000 .

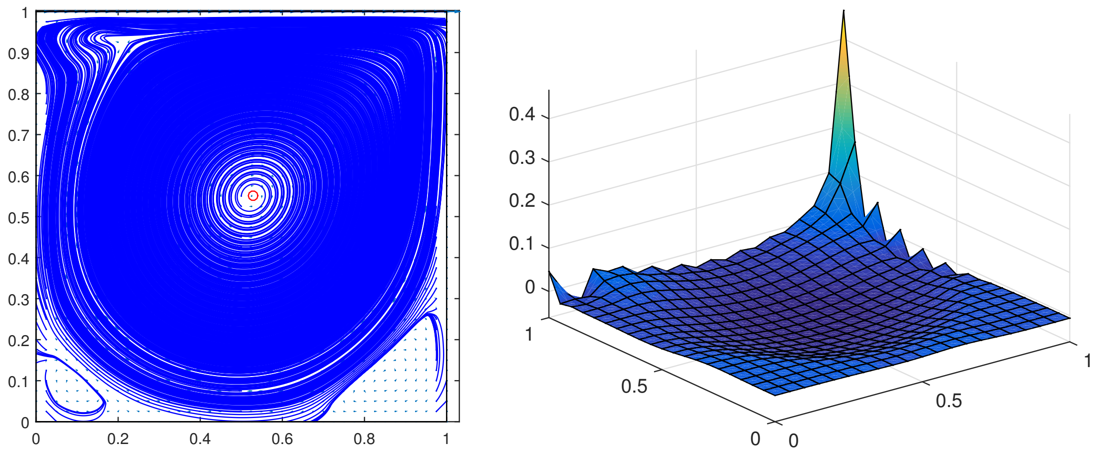

Figure 1.12: Re = 2000 .

Note that the vortex is displacing to the center of the domain and some other vortices generate on the bottom.

Remark that the higher the Reynolds the clearer the boundary layers will be. This means that the variation on the velocity becomes specially insignificant and therefore the velocity gradient will introduce some instability. Therefore for large Reynolds numbers, some stabilization has to be taken into account.

Comparing with the literature, the results obtained by optical adjustment of the vortex:

Re	Source	x_1	x_2
100	Present Simulation	0.62	0.72
	Donea&Huerta(2003)	0.62	0.74
	Burggraf(1996)	0.62	0.74
	Tuann&Olson(1978)	0.61	0.722
1000	Present Simulation	0.54	0.568
	Donea&Huerta(2003)	0.54	0.573
	Ozawa(1975)	0.533	0.569
	Goda(1979)	0.538	0.575

The results are similar to the literature.

APPENDIX

```
function [dofDir, valDir, dofUnk, confined] = BC_red(X,dom, ndofV)

x1 = dom(1);   x2 = dom(2);
y1 = dom(3);   y2 = dom(4);
tol = 1e-6;
nodesX1 = find(abs(X(:,1)-x1) < tol & abs(X(:,2)-y1) > tol & abs(X(:,2)-
    y2) > tol);
nodesX2 = find(abs(X(:,1)-x2) < tol & abs(X(:,2)-y1) > tol & abs(X(:,2)-
    y2) > tol);
nodesY1 = find(abs(X(:,2)-y1) < tol);
nodesY2 = find(abs(X(:,2)-y2) < tol);

confined = 1;
dofDir = [
    2*nodesX1-1; 2*nodesX1
    2*nodesX2-1; 2*nodesX2
    2*nodesY1-1; 2*nodesY1
    2*nodesY2-1; 2*nodesY2
     ];
valDir = [
    zeros(size(nodesX1)); zeros(size(nodesX1))
    zeros(size(nodesX2)); zeros(size(nodesX2))
    zeros(size(nodesY1)); zeros(size(nodesY1))
    ones(size(nodesY2));   zeros(size(nodesY2))
    ];
dofUnk = setdiff(1:ndofV, dofDir);
```

```
function res = cinput(s, DefaultValue)
%
% res = cinput(s, DefaultValue)
% User input. If no value is given, res takes the DefaultValue.

text = [s ' (default ' num2str(DefaultValue) ')= '];
res = input(text);
if isempty(res)
    res = DefaultValue;
end
```

```
function C = ConvectionMatrix(X,T, referenceElement, velo)
% C = ConvectionMatrix(X,T, referenceElement, velo)
% Convection Matrix for a 2D Navier-Stokes problem
%
```

```
% 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;

C = zeros(ndofV,ndofV);

% Loop on elements
for ielem = 1:nElem
    % Global number of the nodes in element ielem
    Te = T(ielem,:);
    % Degrees of freedom in element ielem
    Te_dof = reshape([2*Te-1; 2*Te],1,nedofV);
    % Coordinates of the nodes in element ielem
    Xe = X(Te(1:ngeom),:);
    % Velocity at the element's nodes
    Ve = velo(Te,:);
    % Element matrix
    Ce = EleConvMatrix(Ve,Xe,ngeom,nedofV,ngaus,wgp,N,Nxi,Neta);
    % Assemble the contrinbution of the element matrix
    C(Te_dof, Te_dof) = C(Te_dof, Te_dof) + Ce;
end
```

```matlab
function Ce = EleConvMatrix(Ve,Xe,ngeom,nedofV,ngaus,wgp,N,Nxi,Neta)
%

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,:);

        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
```

```matlab
function [X,T] = CreateAdaptedMesh(dom,nx,ny,elem,degree)
% [X,T] = CreateAdaptedMesh(dom,nx,ny,elem,degree)
% Structured mesh, refined close to the boundaries, in a rectangular domain
% Input:
%   dom = [x1,x2,y1,y2]:  vertices' coordinates
%   nx,ny: number of elements in each direction
%   elem: type of element (0:quadrilateral, 1:triangle, 11: triangle with bubble function)
%   degree: interpolation degree
```

```
% Output:
%     X:   nodal coordinates
%     T:   connectivities

x1 = dom(1); x2 = dom(2);
y1 = dom(3); y2 = dom(4);

npx = degree*nx + 1;
npy = degree*ny + 1;

npt = npx*npy;
x = linspace(x1,x2,npx);
x(2:npx-1) = (x2-x1)*(tanh(5/2)+tanh(5*(x(2:npx-1)-(x1+x2)/2)))/(2*tanh
    (5/2))+x1;
y = linspace(y1,y2,npy);
y(2:npy-1) = (y2-y1)*(tanh(5/2)+tanh(5*(y(2:npy-1)-(y1+y2)/2)))/(2*tanh
    (5/2))+y1;
[x,y] = meshgrid(x,y);
X = [reshape(x',npt,1), reshape(y',npt,1)];

if elem == 0
    nen = (degree+1)^2;
    T = zeros(nx*ny,nen);
    if degree == 1
        for i=1:ny
            for j=1:nx
                ielem = (i-1)*nx+j;
                inode = (i-1)*(npx)+j;
                T(ielem,:) = [inode   inode+1   inode+npx+1   inode+npx
                    ];
            end
        end
    elseif degree == 2
        for i=1:ny
            for j=1:nx
                ielem = (i-1)*nx + j;
                inode = (i-1)*2*npx + 2*(j-1) + 1;
                nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx
                    +(0:2)];
                T(ielem,:) = nodes_aux([1  3  9  7  2  6  8  4  5]);
            end
        end
    else
        error('not available element')
```

```
    end
elseif elem == 1
    nen = (degree+1)*(degree+2)/2;
    T = zeros(2*nx*ny,nen);
    if degree == 1
        for i=1:ny
            for j=1:nx
                ielem = 2*((i-1)*nx+j)-1;
                inode = (i-1)*(npx)+j;
                T(ielem,:) = [inode   inode+1   inode+(npx)];
                T(ielem+1,:) = [inode+1   inode+1+npx   inode+npx];
            end
        end
        % Modification of left lower and right upper corner elements to avoid them
        % having all their nodes on the boundary
        if npx > 2
            T(1,:) = [1   npx+2   npx+1];
            T(2,:) = [1    2     npx+2];
            aux = size(T,1);
            T(aux-1,:) = [npx*ny-1    npx*npy   npx*npy-1];
            T(aux,:)   = [npx*ny-1    npx*ny    npx*npy];
        end
    elseif degree == 2
        for i=1:ny
            for j=1:nx
                ielem=2*((i-1)*nx+j)-1;
                inode=(i-1)*2*(npx)+2*(j-1)+1;
                nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)];
                T(ielem,:) = nodes_aux([1  3  7  2  5  4]);
                T(ielem+1,:) = nodes_aux([3  9  7  6  8  5]);
            end
        end
        % Modification of left lower and right upper corner elements to avoid them
        % having all their nodes on the boundary
        if npx > 3
            inode = 1;
            nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)];
            T(1,:) = nodes_aux([1  9  7  5  8  4]);
            T(2,:) = nodes_aux([1  3  9  2  6  5]);

```

```
            ielem   = size(T,1)-1;
            inode = npx*(npy-2)-2;
            nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)];
            T(ielem,:) = nodes_aux([1  9  7  5  8  4]);
            T(ielem+1,:) = nodes_aux([1  3  9  2  6  5]);
        end
    else
        error('not available element')
    end
elseif elem == 11
    if degree == 1
        T = zeros(2*nx*ny,4);
        npt = size(X,1);
        for i=1:ny
            for j=1:nx
                ielem = 2*((i-1)*nx+j)-1;
                inode = (i-1)*(npx)+j;
                n_ad = npt + 2*((i-1)*nx+j)-1;
                T(ielem,:) = [inode   inode+1   inode+(npx)   n_ad];
                T(ielem+1,:) = [inode+1   inode+1+npx   inode+npx   n_ad+1];
            end
        end
        % Modification of left lower and right upper corner elements to avoid them
        % having all their nodes on the boundary
        if npx > 2
            T(1,:) = [1  npx+2   npx+1   npt+1];
            T(2,:) = [1    2     npx+2   npt+2];
            aux = size(T,1);
            T(aux-1,:) = [npx*ny-1    npx*npy    npx*npy-1    npt+aux-1];
            T(aux,:)   = [npx*ny-1    npx*ny     npx*npy      npt+aux];
        end
    else
        error('not available element')
    end
else
    error('not available element')
end
```

```
% elseif elem == 1
%     nen = (degree+1)*(degree+2)/2;
%     T = zeros(2*nx*ny,nen);
%     nx_2 = round(nx/2); ny_2 = round(ny/2);
%     if degree == 1
%         for i=1:ny
%             for j=1:nx
%                 ielem = 2*((i-1)*nx+j)-1;
%                 inode = (i-1)*(npx)+j;
%                 nodes = [inode   inode+1   inode+npx+1    inode+npx];
%                 if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
%                     T(ielem,:) = nodes([1,2,3]);
%                     T(ielem+1,:) = nodes([1,3,4]);
%                 else
%                     T(ielem,:) = nodes([1,2,4]);
%                     T(ielem+1,:) = nodes([2,3,4]);
%                 end
%             end
%         end
%     elseif degree == 2
%         for i=1:ny
%             for j=1:nx
%                 ielem=2*((i-1)*nx+j)-1;
%                 inode=(i-1)*2*(npx)+2*(j-1)+1;
%                 nodes = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)];
%                 if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
%                     T(ielem,:)   = nodes([1  3  9  2  6  5]);
%                     T(ielem+1,:) = nodes([1  9  7  5  8  4]);
%                 else
%                     T(ielem,:)   = nodes([1  3  7  2  5  4]);
%                     T(ielem+1,:) = nodes([3  9  7  6  8  5]);
%                 end
%             end
%         end
%
%     else
%         error('not available element')
%     end
% elseif elem == 11
%     if degree == 1
%         T = zeros(2*nx*ny,4);
%         nx_2 = round(nx/2); ny_2 = round(ny/2);
%         for i=1:ny
```

```
%                for  j=1:nx
%                    ielem  =  2*((i-1)*nx+j)-1;
%                    inode  =  (i-1)*(npx)+j;
%                    nodes = [inode    inode+1    inode+npx+1     inode+npx];
%                    n_ad  =  npx*npy  +  2*((i-1)*nx+j)-1;
%                    if  (i<=ny_2 && j<=nx_2)  ||  (i>ny_2 && j>nx_2)
%                        T(ielem ,:)  =  [nodes([1,2,3]),  n_ad];
%                        T(ielem+1,:)  =  [nodes([1,3,4]),  n_ad+1];
%                    else
%                        T(ielem ,:)  =  [nodes([1,2,4]),  n_ad];
%                        T(ielem+1,:)  =  [nodes([2,3,4]),  n_ad+1];
%                    end
%                end
%            end
%        else
%            error('not available element')
%        end
```

```
function  [X,T,XP,TP]  =  CreateMeshes(dom,nx,ny,referenceElement,adapted)
% Uniform meshes in a rectangular domain
% Input:
%   dom = [x1,x2,y1,y2]:  vertices' coordinates
%   nx,ny: number of elements in each direction
%   referenceElement: reference element's properties
%   adapted = 1 for a mesh which is refined near the boundary
% Output:
%   X,T:    nodal coordinates and connectivities of the velocity mesh
%   XP,TP: nodal coordinates and connectivities of the pressure mesh

elemV = referenceElement.elemV;
degreeV = referenceElement.degreeV;
elemP = referenceElement.elemP;
degreeP = referenceElement.degreeP;

if adapted == 1
    [X,T] = CreateAdaptedMesh(dom,nx,ny,elemV,degreeV);
else
    [X,T] = CreateUniformMesh(dom,nx,ny,elemV,degreeV);
end

if degreeP == 0
    nElem = size(T,1);
    TP = (1:nElem)';
    XP = zeros(nElem,2);
```

```
    for i = 1:nElem
        Te = T(i,:);
        Xe = X(Te,:);
        XP(i,:) = [mean(Xe(:,1)), mean(Xe(:,2))];
    end
elseif elemV == 11
    warning('only linear elements')
    XP = X;
    TP = T(:,1:3);
else
    if adapted == 1
        [XP,TP] = CreateAdaptedMesh(dom,nx,ny,elemP,degreeP);
    else
        [XP,TP] = CreateUniformMesh(dom,nx,ny,elemP,degreeP);
    end
end
```

```
function [X,T] = CreateUniformMesh(dom,nx,ny,elem,degree)
% [X,T] = CreateUniformMesh(dom,nx,ny,elem,degree)
% Uniform mesh in a rectangular domain
% Input:
%    dom = [x1,x2,y1,y2]:  vertices' coordinates
%    nx,ny: number of elements in each direction
%    elem: type of element (0:quadrilateral, 1:triangle, 11: triangle
     with bubble function)
%    degree: interpolation degree
% Output:
%    X:  nodal coordinates
%    T:  connectivities

x1 = dom(1); x2 = dom(2);
y1 = dom(3); y2 = dom(4);

npx = degree*nx + 1;
npy = degree*ny + 1;

npt = npx*npy;
x = linspace(x1,x2,npx);
y = linspace(y1,y2,npy);
[x,y] = meshgrid(x,y);
X = [reshape(x',npt,1), reshape(y',npt,1)];

if elem == 0
    nen = (degree+1)^2;
```

```
    T = zeros(nx*ny,nen);
    if degree == 1
        for i=1:ny
            for j=1:nx
                ielem = (i-1)*nx+j;
                inode = (i-1)*(npx)+j;
                T(ielem,:) = [inode   inode+1   inode+npx+1   inode+npx];
            end
        end
    elseif degree == 2
        for i=1:ny
            for j=1:nx
                ielem = (i-1)*nx + j;
                inode = (i-1)*2*npx + 2*(j-1) + 1;
                nodes_aux = [inode+(0:2)   inode+npx+(0:2)   inode+2*npx+(0:2)];
                T(ielem,:) = nodes_aux([1  3  9  7  2  6  8  4  5]);
            end
        end
    else
        error('not available element')
    end
elseif elem == 1
    nen = (degree+1)*(degree+2)/2;
    T = zeros(2*nx*ny,nen);
    if degree == 1
        for i=1:ny
            for j=1:nx
                ielem = 2*((i-1)*nx+j)-1;
                inode = (i-1)*(npx)+j;
                T(ielem,:) = [inode   inode+1   inode+(npx)];
                T(ielem+1,:) = [inode+1   inode+1+npx   inode+npx];
            end
        end
        % Modification of left lower and right upper corner elements to avoid them
        % having all their nodes on the boundary
        if npx > 2
            T(1,:) = [1   npx+2   npx+1];
            T(2,:) = [1     2     npx+2];
            aux = size(T,1);
            T(aux-1,:) = [npx*ny-1    npx*npy    npx*npy-1];
            T(aux,:)   = [npx*ny-1    npx*ny     npx*npy];
```

```
            end
        elseif degree == 2
            for i=1:ny
                for j=1:nx
                    ielem=2*((i-1)*nx+j)-1;
                    inode=(i-1)*2*(npx)+2*(j-1)+1;
                    nodes_aux = [inode+(0:2) inode+npx+(0:2) inode+2*npx+(0:2)];
                    T(ielem,:) = nodes_aux([1 3 7 2 5 4]);
                    T(ielem+1,:) = nodes_aux([3 9 7 6 8 5]);
                end
            end
            % Modification of left lower and right upper corner elements to avoid them
            % having all their nodes on the boundary
            if npx > 3
                inode = 1;
                nodes_aux = [inode+(0:2) inode+npx+(0:2) inode+2*npx+(0:2)];
                T(1,:) = nodes_aux([1 9 7 5 8 4]);
                T(2,:) = nodes_aux([1 3 9 2 6 5]);

                ielem = size(T,1)-1;
                inode = npx*(npy-2)-2;
                nodes_aux = [inode+(0:2) inode+npx+(0:2) inode+2*npx+(0:2)];
                T(ielem,:) = nodes_aux([1 9 7 5 8 4]);
                T(ielem+1,:) = nodes_aux([1 3 9 2 6 5]);
            end
        else
            error('not available element')
        end
    elseif elem == 11
        if degree == 1
            T = zeros(2*nx*ny,4);
            npt = size(X,1);
            for i=1:ny
                for j=1:nx
                    ielem = 2*((i-1)*nx+j)-1;
                    inode = (i-1)*(npx)+j;
                    n_ad = npt + 2*((i-1)*nx+j)-1;
                    T(ielem,:) = [inode inode+1 inode+(npx) n_ad];
                    T(ielem+1,:) = [inode+1 inode+1+npx inode+npx n_ad+1];
```

```
            end
        end
        % Modification of left lower and right upper corner elements to avoid them
        % having all their nodes on the boundary
        if npx > 2
            T(1,:) = [1   npx+2   npx+1   npt+1];
            T(2,:) = [1     2     npx+2   npt+2];
            aux = size(T,1);
            T(aux-1,:) = [npx*ny-1    npx*npy    npx*npy-1    npt+aux-1];
            T(aux,:)   = [npx*ny-1    npx*ny     npx*npy      npt+aux];
        end
    else
        error('not available element')
    end
else
    error('not available element')
end




% elseif elem == 1
%      nen = (degree+1)*(degree+2)/2;
%      T = zeros(2*nx*ny,nen);
%      nx_2 = round(nx/2); ny_2 = round(ny/2);
%      if degree == 1
%          for i=1:ny
%              for j=1:nx
%                  ielem = 2*((i-1)*nx+j)-1;
%                  inode = (i-1)*(npx)+j;
%                  nodes = [inode   inode+1   inode+npx+1    inode+npx];
%                  if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
%                      T(ielem,:) = nodes([1,2,3]);
%                      T(ielem+1,:) = nodes([1,3,4]);
%                  else
%                      T(ielem,:) = nodes([1,2,4]);
%                      T(ielem+1,:) = nodes([2,3,4]);
%                  end
%              end
%          end
%      elseif degree == 2
%          for i=1:ny
%              for j=1:nx
```

```
%                   ielem=2*((i-1)*nx+j)-1;
%                   inode=(i-1)*2*(npx)+2*(j-1)+1;
%                   nodes = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx
    +(0:2)];
%                   if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
%                       T(ielem,:)   = nodes([1  3  9  2  6  5]);
%                       T(ielem+1,:) = nodes([1  9  7  5  8  4]);
%                   else
%                       T(ielem,:)   = nodes([1  3  7  2  5  4]);
%                       T(ielem+1,:) = nodes([3  9  7  6  8  5]);
%                   end
%               end
%           end
%
%       else
%           error('not available element')
%       end
% elseif elem == 11
%       if degree == 1
%           T = zeros(2*nx*ny,4);
%           nx_2 = round(nx/2); ny_2 = round(ny/2);
%           for i=1:ny
%               for j=1:nx
%                   ielem = 2*((i-1)*nx+j)-1;
%                   inode = (i-1)*(npx)+j;
%                   nodes = [inode   inode+1   inode+npx+1    inode+npx];
%                   n_ad = npx*npy + 2*((i-1)*nx+j)-1;
%                   if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
%                       T(ielem,:) = [nodes([1,2,3]), n_ad];
%                       T(ielem+1,:) = [nodes([1,3,4]), n_ad+1];
%                   else
%                       T(ielem,:) = [nodes([1,2,4]), n_ad];
%                       T(ielem+1,:) = [nodes([2,3,4]), n_ad+1];
%                   end
%               end
%           end
%       else
%           error('not available element')
%       end
```

```
% This program solves a Navier-Stokes problem in a square domain
% The non-linear problem is solved using Picard iteration

clear; close all; clc
```

```
addpath('Func_ReferenceElement')

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

Re_v = [100];%,500,1000,2000];
x=[0.62,0.74;0.55,0.59;0.54,0.568;0.53,0.55];
for i=1:length(Re_v)
Re=Re_v(i)
nu = 1/Re;

% Element type and interpolation degree
% (0: quadrilaterals, 1: triangles, 11: triangles with bubble function)
elemV = 0; degreeV = 2; 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 =20;% cinput('Number of elements in each direction',20);
ny = nx;
adapted = 0;
[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
[K,G,f] = StokesSystem(X,T,XP,TP,referenceElement);
K = K*nu;
[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,:) = [];
```

```
else
    nunkP = ndofP;
end

Kred = K(dofUnk,dofUnk);
Gred = G(:,dofUnk);
fred = f - K(:,dofDir)*valDir;
fred = fred(dofUnk);
A = [Kred    Gred'
     Gred    zeros(nunkP)];

% Initial solution
disp(' ')
IniVelo_file = 0;%input('.mat file with the initial velocity = ','s');
if IniVelo_file==0%isempty(IniVelo_file)
    velo = zeros(ndofV/2,2);
    y2 = dom(4);
    nodesY2 = find(abs(X(:,2)-y2) < 1e-6);
    velo(nodesY2,1) = 1;
else
    load(IniVelo_file);
end
pres = zeros(nunkP,1);
veloVect = reshape(velo',ndofV,1);
sol0  = [veloVect(dofUnk);pres(1:nunkP)];

iter = 0; tol = 0.5e-8;
while iter < 100
    fprintf('Iteration = %d\n',iter);

    C = ConvectionMatrix(X,T,referenceElement,velo);
    Cred = C(dofUnk,dofUnk);

    Atot = A;
    Atot(1:nunkV,1:nunkV) = A(1:nunkV,1:nunkV) + Cred;
    btot = [fred - C(dofUnk,dofDir)*valDir; zeros(nunkP,1)];

    % Computation of residual
    res = btot - Atot*sol0;
    % Computation of velocity and pressure increment
    solInc = Atot\res;

    % Update the solution
    veloInc = zeros(ndofV,1);
```

```
    veloInc(dofUnk) = solInc(1:nunkV);
    presInc = solInc(nunkV+1:end);
    velo = velo + reshape(veloInc,2,[])';
    pres = pres + presInc;

    % Check convergence
    delta1 = max(abs(veloInc));
    delta2 = max(abs(res));
    fprintf('Velocity increment=%8.6e, Residue max=%8.6e\n',delta1, delta2);
    if delta1 < tol*max(max(abs(velo))) && delta2 < tol
        fprintf('\nConvergence achieved in iteration number %g\n',iter);
        break
    end

    % Update variables for next iteration
    veloVect = reshape(velo',ndofV,1);
    sol0 = [veloVect(dofUnk); pres];
    iter = iter + 1;
end

% Postprocess
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);
[p col]=min(pres);
plot(XP(col,1),XP(col,2),'xg');
plot(x(i,1),x(i,2),'or');
if degreeP == 0
    PlotResults(X,T,pres,referenceElement.elemP,referenceElement.degreeP)
else
    PlotResults(XP,TP,pres,referenceElement.elemP,referenceElement.degreeP)
end
```

```
iterres(i)=iter;
end
figure
plot(Re_v,iterres,'-ob');
```

```
% 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 = 0; degreeV = 2; degreeP = 1;
% elemV = 1; degreeV = 2; degreeP = 1;
 elemV = 0; 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 = 20;%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
[K,G,f] = StokesSystem(X,T,XP,TP,referenceElement);
[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,:) = [];
else
    nunkP = ndofP;
end

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

A = [Kred    Gred';
     Gred    zeros(nunkP)];
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;
subplot(2,2,1)
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
xlim([0,1]);
subplot(2,2,2)
PlotStreamlines(X,velo,dom);
xlim([0,1]);
if degreeP == 0
    PlotResults(X,T,pres,referenceElement.elemP,referenceElement.degreeP)
else
    subplot(2,2,3)
    PlotResults(XP,TP,pres,referenceElement.elemP,referenceElement.
```

```
        degreeP)
end
```

```
function PlotMesh(T,X,elem,str,nonum)
% PlotMesh(T,X,str,nonum)
% X:    nodal coordinates
% T:    connectivities
% str:  linestyle, color and marker used in the plot (optional)
% nonum = 1 to show nodes' number(optional)


% Line style and color
if nargin == 3
  str1 = 'yo';
  str2 = 'y-';
else
  if str(1) == ':' | str(1) == '-'
    str1 = 'yo';
    str2 = ['y' str];
  else
    str1 = [str(1) 'o'];
    str2 = str;
  end
end

nen = size(T,2);


if elem == 0
    if nen <= 4
        order = [1:nen,1];
    elseif nen == 9
        order = [1,5,2,6,3,7,4,8,1];
    end
elseif elem == 1
    if nen <= 3
        order = [1:nen,1];
    elseif nen == 6
        order = [1,4,2,5,3,6,1];
    end
elseif elem == 11
    order = [1:3,1];
end

```

```

% Nodes
plot(X(:,1),X(:,2),str1)
hold on
% Elements
for j = 1:size(T,1)
    plot(X(T(j,order),1),X(T(j,order),2),str2)
end


% nodes number
if nargin==5
    if nonum==1
        for I=1:size(X,1)
            text(X(I,1)+0.02,X(I,2)+0.03,int2str(I),'FontSize',16)
        end
    end
end

axis('equal')
axis('off')

hold off
```

```
function PlotResults(X,T,sol,elem,degree)
figure;hold on
if elem == 1 && degree == 1
    figure;
    trisurf(T,X(:,1),X(:,2),sol,'FaceColor','interp');
elseif degree == 0
    nElem = size(T,1);
    if elem == 0
        nen = 4;
    else
        nen = 3;
    end
    figure; hold on
    for ielem = 1:nElem
        Te = T(ielem,1:nen);
        Xe = X(Te,:);
        zz = sol(ielem)*ones(nen,1);
        patch(Xe(:,1),Xe(:,2),zz,zz)
    end
else
```

```
    [nElem,nen] = size(T);
    if elem == 11
        ngeom = nen-1;
    else
        ngeom = nen;
    end
    if elem == 0
        npt = 2*degree+1;
        x = linspace(-1,1,npt);
        [x,y] = meshgrid(x,x);
        pts = [reshape(x,npt^2,1), reshape(y,npt^2,1)];
        ptsEdge = [
            linspace(-1,1,npt)'              -ones(npt,1)
            ones(npt,1)                      linspace(-1,1,npt)'
            flipud(linspace(-1,1,npt)')      ones(npt,1)
            -ones(npt,1)                     flipud(linspace(-1,1,npt)')
            ];
    else
        npt = 2*degree+1;
        x = linspace(0,1,npt);
        [x,y] = meshgrid(x,x);
        pts = [reshape(x,npt^2,1), reshape(y,npt^2,1)];
        ind = find(pts(:,2) <= 1 - pts(:,1));
        pts = pts(ind,:);
        ptsEdge = [
            linspace(0,1,npt)'             1-linspace(0,1,npt)'
            flipud(linspace(0,1,npt)')     zeros(npt,1)
            zeros(npt,1)                   linspace(0,1,npt)'
            ];
    end
    tri = delaunay(pts(:,1), pts(:,2));
    N = ShapeFunc(elem,degree,pts);
    NEdge = ShapeFunc(elem,degree,ptsEdge);
    hold on;
    for ielem = 1:nElem
        Te = T(ielem,:);
        Xe = X(Te(1:ngeom),:);
        sol_e = sol(Te);
        xx = N(:,1:ngeom)*Xe(:,1);
        yy = N(:,1:ngeom)*Xe(:,2);
        zz = N*sol_e;
        xEdge = NEdge(:,1:ngeom)*Xe(:,1);
        yEdge = NEdge(:,1:ngeom)*Xe(:,2);
        zEdge = NEdge*sol_e;
```

```
        trisurf(tri,xx, yy, zz,'FaceColor','interp','EdgeColor','none')
        plot3(xEdge,yEdge,zEdge,'k');
    end
end
grid on
view(3); axis tight
set(gca,'FontSize',12)
```

```
function PlotStreamlines(X,velo,dom)

% Define a grid
xGrid = linspace(dom(1),dom(2),25);
yGrid = linspace(dom(3),dom(4),25);
% Interpolate the solution
tri = delaunay(X(:,1),X(:,2));
uGrid = tri2grid(X',tri',velo(:,1),xGrid,yGrid);    % x,y,f are column
    vectors.
vGrid = tri2grid(X',tri',velo(:,2),xGrid,yGrid);    % x,y,f are column
    vectors.
[xGrid,yGrid] = meshgrid(xGrid,yGrid);

% Points where the streamlines start
y1 = min(X(:,2));
aux = find(abs(X(:,2)-y1) < 1e-6);
xx = X(aux(round(length(aux)/2)),1);
ind1 = find(abs(X(:,1)-xx) < 1e-6);
xx = X(aux(2),1);
ind2 = find(abs(X(:,1)-xx) < 1e-6);
xx = X(aux(end-1),1);
ind3 = find(abs(X(:,1)-xx) < 1e-6);
sx = [X(ind1,1); X(ind2,1); X(ind3,1)];
sy = [X(ind1,2); X(ind2,2); X(ind3,2)];



%figure;
streamline(xGrid,yGrid,uGrid,vGrid,sx,sy)
hold on
%plot(sx,sy,'r*')
plot(dom([1,2,2,1,1]),dom([3,3,4,4,3]),'k')
axis equal; axis tight
```

```
function s = SourceTerm(pt)
% s = SourceTerm(pt)
```

```
% x = pt(1); y = pt(2);
s = zeros(2,1);
```

```
function [K,G,f] = StokesSystem(X,T,XP,TP,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...)


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);

% 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,fe] = EleMatStokes(Xe,ngeom,nedofV,nedofP,ngaus,wgp,N,Nxi,
        Neta,NP);

    % Assemble the element matrices
    K(Te_dof, Te_dof) = K(Te_dof, Te_dof) + Ke;
    G(TPe_dof,Te_dof) = G(TPe_dof,Te_dof) + Ge;
    f(Te_dof) = f(Te_dof) + fe;
end






function [Ke,Ge,fe] = EleMatStokes(Xe,ngeom,nedofV,nedofP,ngaus,wgp,N,
    Nxi,Neta,NP)
% [Ke,Ge,fe] = EleMatStokes(Xe,ngeom,nedofV,nedofP,ngaus,wgp,N,Nxi,Neta,
    NP)

Ke = zeros(nedofV,nedofV);
Ge = zeros(nedofP,nedofV);
fe = zeros(nedofV,1);
% 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,:);

            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;
      x_ig = N_ig(1:ngeom)*Xe;
      f_igaus = SourceTerm(x_ig);
      fe = fe + Ngp'*f_igaus*dvolu;
end
```