

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:

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

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

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

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

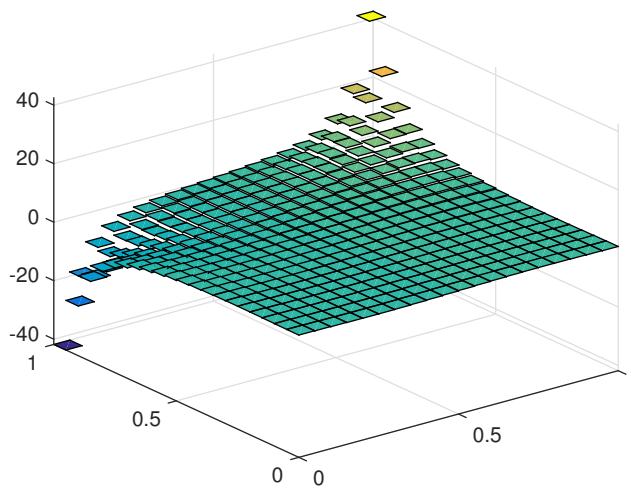
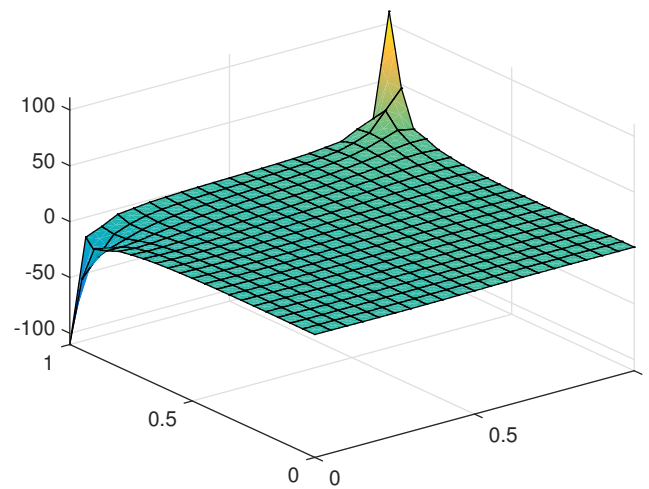
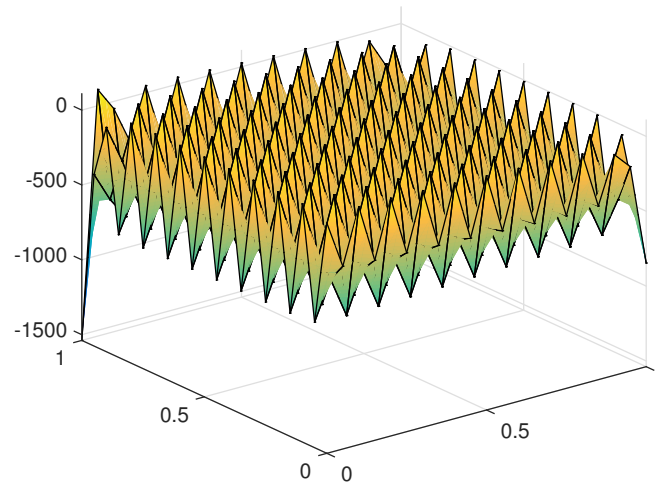
1.1.1 Q_2Q_0 

Figure 1.1: Pressure field for Q_2Q_0

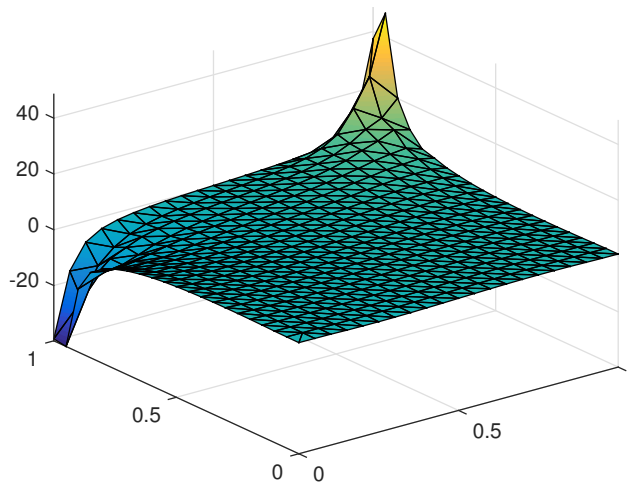
This element does fulfil the LBB compatibility condition. But it cannot capture correctly the behaviour of the pressure since it is discretized by discontinuous bilinear elements.

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 $P_1 P_1$ Figure 1.3: Pressure field for $P_1 P_1$

This element does not fulfil the LBB compatibility condition.

1.1.4 $P_1^+ P_1$ Figure 1.4: Pressure field for $\text{Mini}(P_1^+ P_1)$

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.

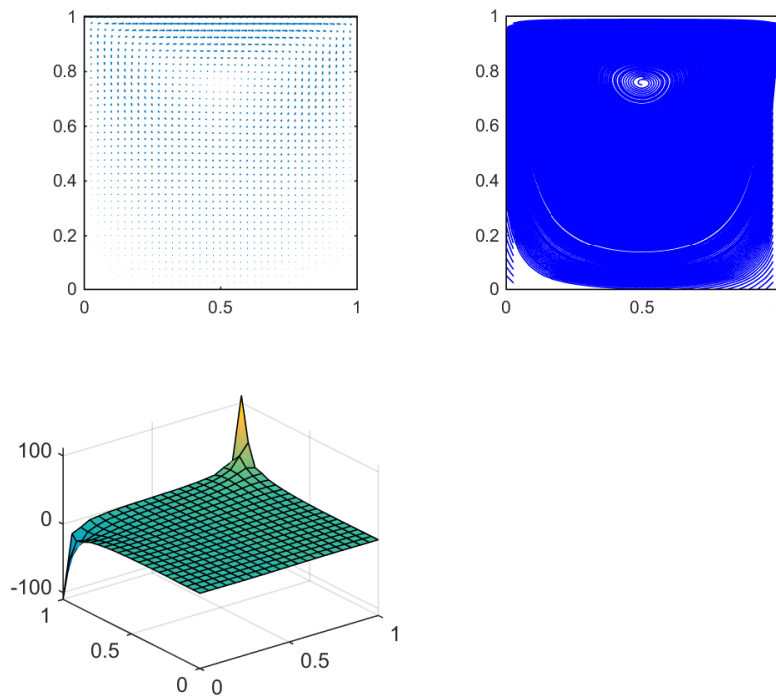


Figure 1.5: Non-refined mesh.

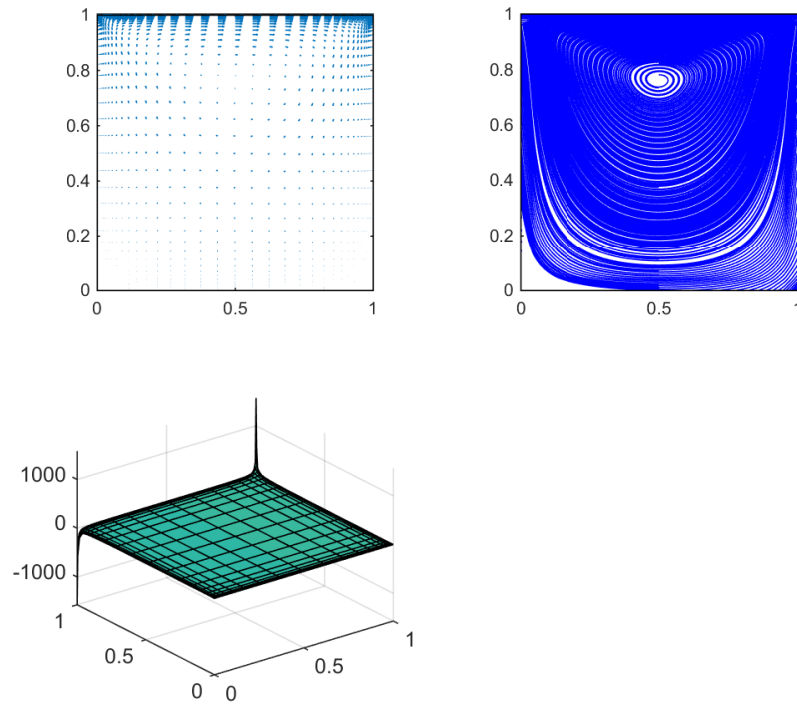


Figure 1.6: Refined mesh.

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 \mathcal{S} and weighting functions space \mathcal{V} , pressure \mathcal{Q} .

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

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

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

The discretized weak form would be:

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

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

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

Now the system to solve is:

$$\begin{bmatrix} K & G \\ G^T & D \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (1.7)$$

where D is the stabilization term matrix:

$$D_{ij} = \int_{\Omega} \tau \nabla N_i \nabla N_j d\Omega \quad (1.8)$$

And regarding the stabilization factor:

$$\tau = \alpha_0 \frac{h_e^2}{4\nu} \quad (1.9)$$

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

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

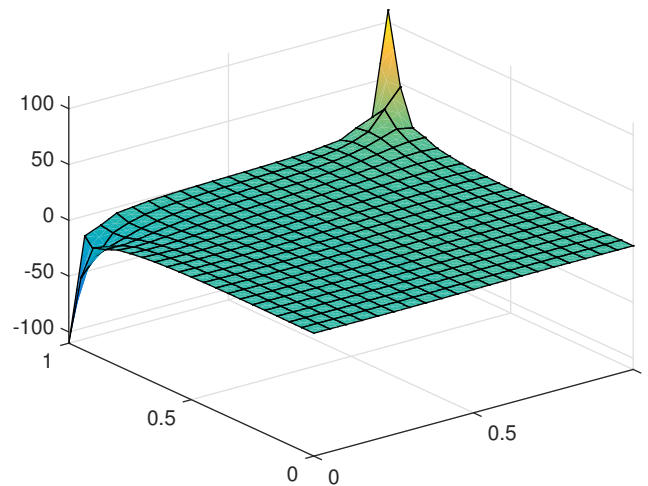


Figure 1.7: P_1P_1 with GLS.

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

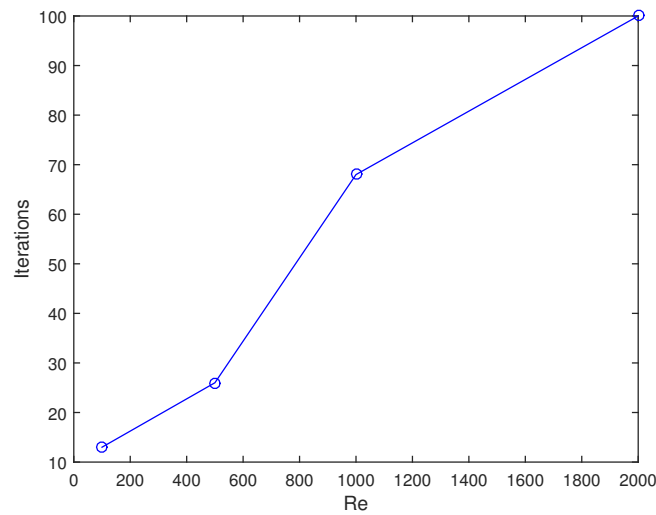


Figure 1.8: Iterations vs Reynolds number.

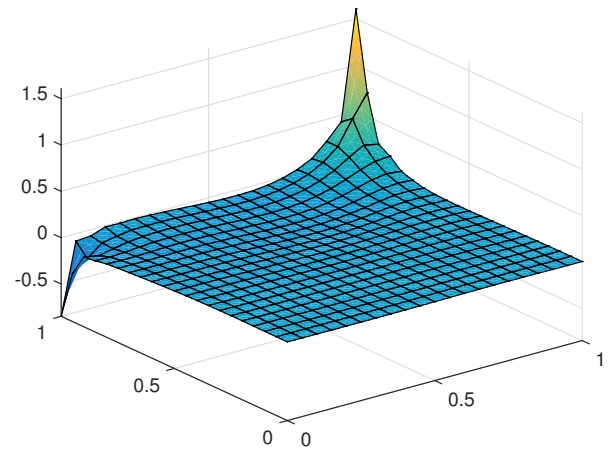
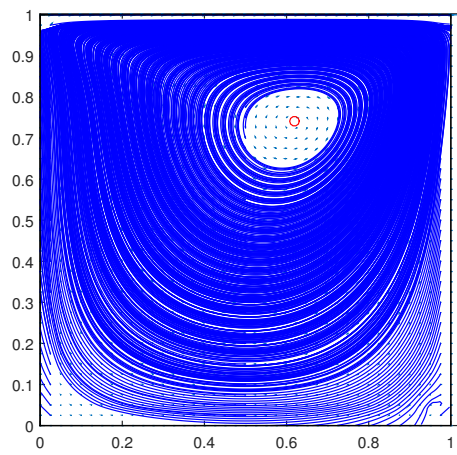


Figure 1.9: $Re = 100$.

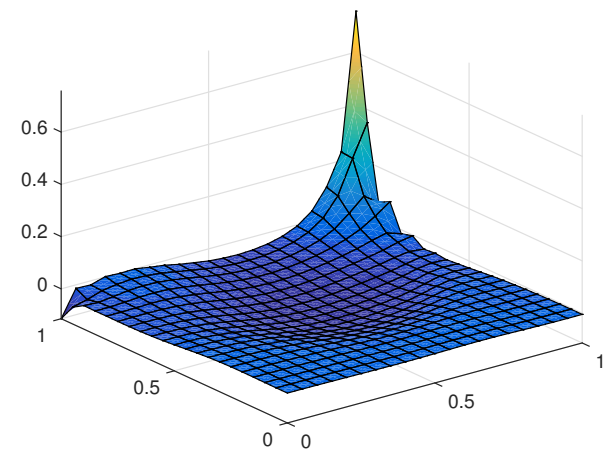
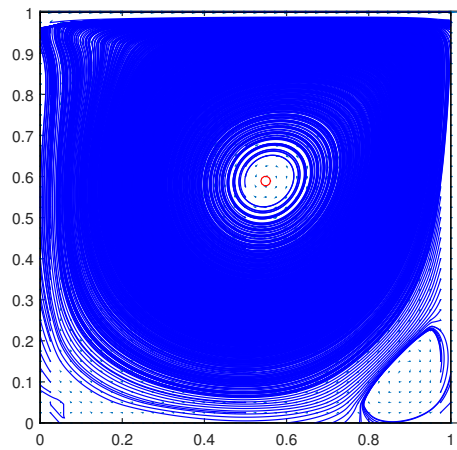
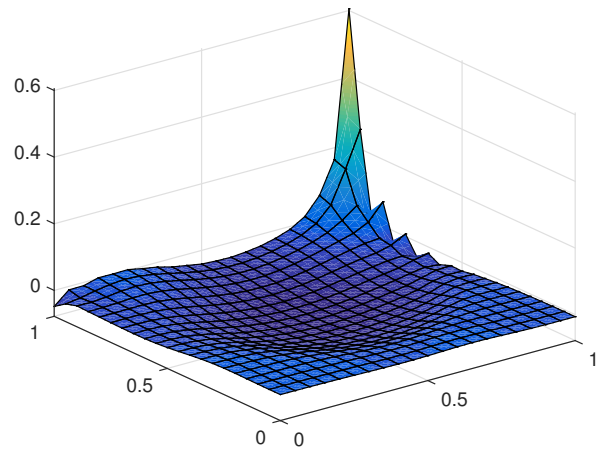
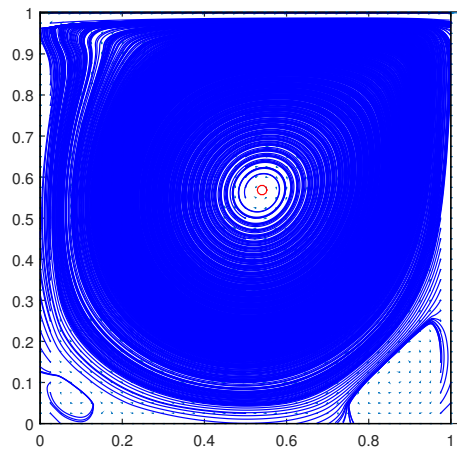
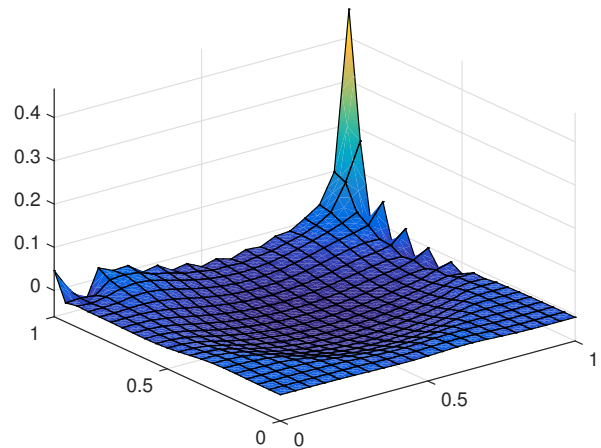
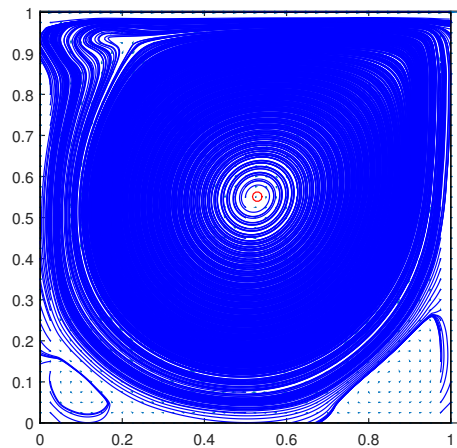


Figure 1.10: $Re = 500$.

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

```

1 function [dofDir, valDir, dofUnk, confined] = BC_red(X, dom, ndofV)
2
3 x1 = dom(1); x2 = dom(2);
4 y1 = dom(3); y2 = dom(4);
5 tol = 1e-6;
6 nodesX1 = find(abs(X(:,1)-x1) < tol & abs(X(:,2)-y1) > tol & abs(X(:,2)-
    y2) > tol);
7 nodesX2 = find(abs(X(:,1)-x2) < tol & abs(X(:,2)-y1) > tol & abs(X(:,2)-
    y2) > tol);
8 nodesY1 = find(abs(X(:,2)-y1) < tol);
9 nodesY2 = find(abs(X(:,2)-y2) < tol);
10
11 confined = 1;
12 dofDir = [
13     2*nodesX1-1; 2*nodesX1
14     2*nodesX2-1; 2*nodesX2
15     2*nodesY1-1; 2*nodesY1
16     2*nodesY2-1; 2*nodesY2
17     ];
18 valDir = [
19     zeros(size(nodesX1)); zeros(size(nodesX1))
20     zeros(size(nodesX2)); zeros(size(nodesX2))
21     zeros(size(nodesY1)); zeros(size(nodesY1))
22     ones(size(nodesY2)); zeros(size(nodesY2))
23     ];
24 dofUnk = setdiff(1:ndofV, dofDir);

1 function res = cinput(s, DefaultValue)
2 %
3 % res = cinput(s, DefaultValue)
4 % User input. If no value is given, res takes the DefaultValue.
5
6 text = [s ' (default ' num2str(DefaultValue) ') = '];
7 res = input(text);
8 if isempty(res)
9     res = DefaultValue;
10 end

1 function C = ConvectionMatrix(X, T, referenceElement, velo)
2 % C = ConvectionMatrix(X, T, referenceElement, velo)
3 % Convection Matrix for a 2D Navier-Stokes problem
4 %

```

```

5 % X,T: nodal coordinates and connectivities for velocity
6 % referenceElement: reference element properties (quadrature, shape
  functions...)
7 % velo: velocity field
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 C = zeros(ndofV,ndofV);
31
32 % Loop on elements
33 for ielem = 1:nElem
34     % Global number of the nodes in element ielem
35     Te = T(ielem,:);
36     % Degrees of freedom in element ielem
37     Te_dof = reshape([2*Te-1; 2*Te],1,ndofV);
38     % Coordinates of the nodes in element ielem
39     Xe = X(Te(1:ngeom),:);
40     % Velocity at the element's nodes
41     Ve = velo(Te,:);
42     % Element matrix
43     Ce = EleConvMatrix(Ve,Xe,ngeom,ndofV,ngaus,wgp,N,Nxi,Neta);
44     % Assemble the contribution of the element matrix
45     C(Te_dof, Te_dof) = C(Te_dof, Te_dof) + Ce;
46 end
47

```

```

48
49
50
51
52
53 function Ce = EleConvMatrix (Ve, Xe, ngeom, nedofV, ngaus, wgp, N, Nxi, Neta)
54 %
55
56 Ce = zeros (nedofV, nedofV);
57
58 % Loop on Gauss points
59 for ig = 1:ngaus
60     N_ig = N(ig, :);
61     Nxi_ig = Nxi(ig, :);
62     Neta_ig = Neta(ig, :);
63     Jacob = [
64         Nxi_ig(1:ngeom)*(Xe(:,1))      Nxi_ig(1:ngeom)*(Xe(:,2))
65         Neta_ig(1:ngeom)*(Xe(:,1))      Neta_ig(1:ngeom)*(Xe(:,2))
66     ];
67     dvolu = wgp(ig)*det(Jacob);
68     res = Jacob\[Nxi_ig;Neta_ig];
69     nx = res(1,:);
70     ny = res(2,:);
71
72     Ngp = [reshape ([1;0]*N_ig,1, nedofV); reshape ([0;1]*N_ig,1, nedofV)
73            ];
74     Nx = [reshape ([1;0]*nx,1, nedofV); reshape ([0;1]*nx,1, nedofV)];
75     Ny = [reshape ([1;0]*ny,1, nedofV); reshape ([0;1]*ny,1, nedofV)];
76
77     v_ig = N_ig*Ve;
78     Ce = Ce + Ngp'*(v_ig(1)*Nx + v_ig(2)*Ny)*dvolu;
79 end

```

```

1 function [X,T] = CreateAdaptedMesh(dom, nx, ny, elem, degree)
2 % [X,T] = CreateAdaptedMesh(dom, nx, ny, elem, degree)
3 % Structured mesh, refined close to the boundaries, in a rectangular
4 % domain
5 % Input:
6 % dom = [x1,x2,y1,y2]: vertices' coordinates
7 % nx,ny: number of elements in each direction
8 % elem: type of element (0:quadrilateral, 1:triangle, 11: triangle
9 % with bubble function)
10 % degree: interpolation degree

```

```

9 % Output:
10 % X: nodal coordinates
11 % T: connectivities
12
13 x1 = dom(1); x2 = dom(2);
14 y1 = dom(3); y2 = dom(4);
15
16 npx = degree*nx + 1;
17 npy = degree*ny + 1;
18
19 npt = npx*npy;
20 x = linspace(x1,x2,npx);
21 x(2:npx-1) = (x2-x1)*(tanh(5/2)+tanh(5*(x(2:npx-1)-(x1+x2)/2)))/(2*tanh
    (5/2))+x1;
22 y = linspace(y1,y2,npy);
23 y(2:npy-1) = (y2-y1)*(tanh(5/2)+tanh(5*(y(2:npy-1)-(y1+y2)/2)))/(2*tanh
    (5/2))+y1;
24 [x,y] = meshgrid(x,y);
25 X = [reshape(x',npt,1), reshape(y',npt,1)];
26
27 if elem == 0
28     nen = (degree+1)^2;
29     T = zeros(nx*ny,nen);
30     if degree == 1
31         for i=1:ny
32             for j=1:nx
33                 ielem = (i-1)*nx+j;
34                 inode = (i-1)*(npx)+j;
35                 T(ielem,:) = [inode inode+1 inode+npx+1 inode+npx
                    ];
36             end
37         end
38     elseif degree == 2
39         for i=1:ny
40             for j=1:nx
41                 ielem = (i-1)*nx + j;
42                 inode = (i-1)*2*npx + 2*(j-1) + 1;
43                 nodes_aux = [inode+(0:2) inode+npx+(0:2) inode+2*npx
                    +(0:2)];
44                 T(ielem,:) = nodes_aux([1 3 9 7 2 6 8 4 5]);
45             end
46         end
47     else
48         error('not available element')

```

```

49     end
50 elseif elem == 1
51     nen = (degree+1)*(degree+2)/2;
52     T = zeros(2*nx*ny, nen);
53     if degree == 1
54         for i=1:ny
55             for j=1:nx
56                 ielem = 2*((i-1)*nx+j)-1;
57                 inode = (i-1)*(npx)+j;
58                 T(ielem,:) = [inode  inode+1  inode+(npx)];
59                 T(ielem+1,:) = [inode+1  inode+1+npx  inode+npx];
60             end
61         end
62         % Modification of left lower and right upper corner elements to
63         % avoid them
64         % having all their nodes on the boundary
65         if npx > 2
66             T(1,:) = [1  npx+2  npx+1];
67             T(2,:) = [1  2  npx+2];
68             aux = size(T,1);
69             T(aux-1,:) = [npx*ny-1  npx*ny  npx*ny-1];
70             T(aux,:) = [npx*ny-1  npx*ny  npx*ny];
71         end
72     elseif degree == 2
73         for i=1:ny
74             for j=1:nx
75                 ielem=2*((i-1)*nx+j)-1;
76                 inode=(i-1)*2*(npx)+2*(j-1)+1;
77                 nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx
78                             +(0:2)];
79                 T(ielem,:) = nodes_aux([1  3  7  2  5  4]);
80                 T(ielem+1,:) = nodes_aux([3  9  7  6  8  5]);
81             end
82         end
83         % Modification of left lower and right upper corner elements to
84         % avoid them
85         % having all their nodes on the boundary
86         if npx > 3
87             inode = 1;
88             nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)
89                         ];
90             T(1,:) = nodes_aux([1  9  7  5  8  4]);
91             T(2,:) = nodes_aux([1  3  9  2  6  5]);
92         end
93     end

```



```

89         ielem = size(T,1)-1;
90         inode = npx*(npy-2)-2;
91         nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)
92                     ];
93         T(ielem,:) = nodes_aux([1  9  7  5  8  4]);
94         T(ielem+1,:) = nodes_aux([1  3  9  2  6  5]);
95     end
96 else
97     error('not available element')
98 end
99 elseif elem == 11
100 if degree == 1
101     T = zeros(2*npx*ny,4);
102     npt = size(X,1);
103     for i=1:ny
104         for j=1:npx
105             ielem = 2*((i-1)*npx+j)-1;
106             inode = (i-1)*(npx)+j;
107             n_ad = npt + 2*((i-1)*npx+j)-1;
108             T(ielem,:) = [inode  inode+1  inode+(npx)  n_ad];
109             T(ielem+1,:) = [inode+1  inode+1+npx  inode+npx  n_ad
110                             +1];
111         end
112     end
113     % Modification of left lower and right upper corner elements to
114     % avoid them
115     % having all their nodes on the boundary
116     if npx > 2
117         T(1,:) = [1  npx+2  npx+1  npt+1];
118         T(2,:) = [1  2  npx+2  npt+2];
119         aux = size(T,1);
120         T(aux-1,:) = [npx*ny-1  npx*ny  npx*ny-1  npt+aux-1];
121         T(aux,:) = [npx*ny-1  npx*ny  npx*ny  npt+aux];
122     end
123 else
124     error('not available element')
125 end
126
127
128
129

```

```

130 % elseif elem == 1
131 %     nen = (degree+1)*(degree+2)/2;
132 %     T = zeros(2*nx*ny, nen);
133 %     nx_2 = round(nx/2); ny_2 = round(ny/2);
134 %     if degree == 1
135 %         for i=1:ny
136 %             for j=1:nx
137 %                 ielem = 2*((i-1)*nx+j)-1;
138 %                 inode = (i-1)*(npx)+j;
139 %                 nodes = [inode  inode+1  inode+npx+1  inode+npx];
140 %                 if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
141 %                     T(ielem,:) = nodes([1,2,3]);
142 %                     T(ielem+1,:) = nodes([1,3,4]);
143 %                 else
144 %                     T(ielem,:) = nodes([1,2,4]);
145 %                     T(ielem+1,:) = nodes([2,3,4]);
146 %                 end
147 %             end
148 %         end
149 %     elseif degree == 2
150 %         for i=1:ny
151 %             for j=1:nx
152 %                 ielem=2*((i-1)*nx+j)-1;
153 %                 inode=(i-1)*2*(npx)+2*(j-1)+1;
154 %                 nodes = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx
+ (0:2)];
155 %                 if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
156 %                     T(ielem,:) = nodes([1 3 9 2 6 5]);
157 %                     T(ielem+1,:) = nodes([1 9 7 5 8 4]);
158 %                 else
159 %                     T(ielem,:) = nodes([1 3 7 2 5 4]);
160 %                     T(ielem+1,:) = nodes([3 9 7 6 8 5]);
161 %                 end
162 %             end
163 %         end
164 %     else
165 %         error('not available element')
166 %     end
167 %
168 % elseif elem == 11
169 %     if degree == 1
170 %         T = zeros(2*nx*ny, 4);
171 %         nx_2 = round(nx/2); ny_2 = round(ny/2);
172 %         for i=1:ny

```

```

173 %           for j=1:nx
174 %               ielem = 2*((i-1)*nx+j)-1;
175 %               inode = (i-1)*(npx)+j;
176 %               nodes = [inode  inode+1  inode+npx+1  inode+npx];
177 %               n_ad = npx*ny + 2*((i-1)*nx+j)-1;
178 %               if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
179 %                   T(ielem,:) = [nodes([1,2,3]), n_ad];
180 %                   T(ielem+1,:) = [nodes([1,3,4]), n_ad+1];
181 %               else
182 %                   T(ielem,:) = [nodes([1,2,4]), n_ad];
183 %                   T(ielem+1,:) = [nodes([2,3,4]), n_ad+1];
184 %               end
185 %           end
186 %       end
187 %   else
188 %       error('not available element')
189 %   end

```

```

1  function [X,T,XP,TP] = CreateMeshes(dom,nx,ny,referenceElement ,adaptd)
2  % Uniform meshes in a rectangular domain
3  % Input:
4  %   dom = [x1,x2,y1,y2]: vertices' coordinates
5  %   nx,ny: number of elements in each direction
6  %   referenceElement: reference element's properties
7  %   adaptd = 1 for a mesh which is refined near the boundary
8  % Output:
9  %   X,T: nodal coordinates and connectivities of the velocity mesh
10 %   XP,TP: nodal coordinates and connectivities of the pressure mesh
11
12 elemV = referenceElement.elemV;
13 degreeV = referenceElement.degreeV;
14 elemP = referenceElement.elemP;
15 degreeP = referenceElement.degreeP;
16
17 if adaptd == 1
18     [X,T] = CreateAdaptedMesh(dom,nx,ny,elemV,degreeV);
19 else
20     [X,T] = CreateUniformMesh(dom,nx,ny,elemV,degreeV);
21 end
22
23 if degreeP == 0
24     nElem = size(T,1);
25     TP = (1:nElem)';
26     XP = zeros(nElem,2);

```

```

27     for i = 1:nElem
28         Te = T(i,:);
29         Xe = X(Te,:);
30         XP(i,:) = [mean(Xe(:,1)), mean(Xe(:,2))];
31     end
32 elseif elemV == 11
33     warning('only linear elements')
34     XP = X;
35     TP = T(:,1:3);
36 else
37     if adapted == 1
38         [XP,TP] = CreateAdaptedMesh(dom,nx,ny,elemP,degreeP);
39     else
40         [XP,TP] = CreateUniformMesh(dom,nx,ny,elemP,degreeP);
41     end
42 end

1 function [X,T] = CreateUniformMesh(dom,nx,ny,elem,degree)
2 % [X,T] = CreateUniformMesh(dom,nx,ny,elem,degree)
3 % Uniform mesh in a rectangular domain
4 % Input:
5 % dom = [x1,x2,y1,y2]: vertices' coordinates
6 % nx,ny: number of elements in each direction
7 % elem: type of element (0:quadrilateral, 1:triangle, 11: triangle
8 % with bubble function)
9 % degree: interpolation degree
10 % Output:
11 % X: nodal coordinates
12 % T: connectivities
13 x1 = dom(1); x2 = dom(2);
14 y1 = dom(3); y2 = dom(4);
15
16 npx = degree*nx + 1;
17 npy = degree*ny + 1;
18
19 npt = npx*npy;
20 x = linspace(x1,x2,npx);
21 y = linspace(y1,y2,npy);
22 [x,y] = meshgrid(x,y);
23 X = [reshape(x',npt,1), reshape(y',npt,1)];
24
25 if elem == 0
26     nen = (degree+1)^2;

```

```

27     T = zeros(nx*ny, nen);
28     if degree == 1
29         for i=1:ny
30             for j=1:nx
31                 ielem = (i-1)*nx+j;
32                 inode = (i-1)*(npx)+j;
33                 T(ielem,:) = [inode    inode+1    inode+npx+1    inode+npx
                                ];
34             end
35         end
36     elseif degree == 2
37         for i=1:ny
38             for j=1:nx
39                 ielem = (i-1)*nx + j;
40                 inode = (i-1)*2*npx + 2*(j-1) + 1;
41                 nodes_aux = [inode+(0:2)    inode+npx+(0:2)    inode+2*npx
                                +(0:2)];
42                 T(ielem,:) = nodes_aux([1 3 9 7 2 6 8 4 5]);
43             end
44         end
45     else
46         error('not available element')
47     end
48 elseif elem == 1
49     nen = (degree+1)*(degree+2)/2;
50     T = zeros(2*nx*ny, nen);
51     if degree == 1
52         for i=1:ny
53             for j=1:nx
54                 ielem = 2*((i-1)*nx+j)-1;
55                 inode = (i-1)*(npx)+j;
56                 T(ielem,:) = [inode    inode+1    inode+(npx)];
57                 T(ielem+1,:) = [inode+1    inode+1+npx    inode+npx];
58             end
59         end
60         % Modification of left lower and right upper corner elements to
           avoid them
61         % having all their nodes on the boundary
62         if npx > 2
63             T(1,:) = [1    npx+2    npx+1];
64             T(2,:) = [1     2    npx+2];
65             aux = size(T,1);
66             T(aux-1,:) = [npx*ny-1    npx*npny    npx*npny-1];
67             T(aux,:) = [npx*ny-1    npx*ny    npx*npny];

```

```

68     end
69     elseif degree == 2
70         for i=1:ny
71             for j=1:nx
72                 ielem=2*((i-1)*nx+j)-1;
73                 inode=(i-1)*2*(npx)+2*(j-1)+1;
74                 nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx
75                               + (0:2) ];
76                 T(ielem,:) = nodes_aux([1  3  7  2  5  4]);
77                 T(ielem+1,:) = nodes_aux([3  9  7  6  8  5]);
78             end
79             % Modification of left lower and right upper corner elements to
80             % avoid them
81             % having all their nodes on the boundary
82             if npx > 3
83                 inode = 1;
84                 nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)
85                               ];
86                 T(1,:) = nodes_aux([1  9  7  5  8  4]);
87                 T(2,:) = nodes_aux([1  3  9  2  6  5]);
88
89                 ielem = size(T,1)-1;
90                 inode = npx*(npy-2)-2;
91                 nodes_aux = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx+(0:2)
92                               ];
93                 T(ielem,:) = nodes_aux([1  9  7  5  8  4]);
94                 T(ielem+1,:) = nodes_aux([1  3  9  2  6  5]);
95             end
96         else
97             error('not available element')
98         end
99     elseif elem == 11
100         if degree == 1
101             T = zeros(2*nx*ny,4);
102             npt = size(X,1);
103             for i=1:ny
104                 for j=1:nx
105                     ielem = 2*((i-1)*nx+j)-1;
106                     inode = (i-1)*(npx)+j;
107                     n_ad = npt + 2*((i-1)*nx+j)-1;
108                     T(ielem,:) = [inode  inode+1  inode+(npx)  n_ad];
109                     T(ielem+1,:) = [inode+1  inode+1+npx  inode+npx  n_ad
110                                       +1];

```

```

107         end
108     end
109     % Modification of left lower and right upper corner elements to
        avoid them
110     % having all their nodes on the boundary
111     if npx > 2
112         T(1,:) = [1 npx+2 npx+1 npt+1];
113         T(2,:) = [1 2 npx+2 npt+2];
114         aux = size(T,1);
115         T(aux-1,:) = [npx*ny-1 npx*ny npx*ny-1 npt+aux-1];
116         T(aux,:) = [npx*ny-1 npx*ny npx*ny npt+aux];
117     end
118     else
119         error('not available element')
120     end
121 else
122     error('not available element')
123 end
124
125
126
127
128 % elseif elem == 1
129 %     nen = (degree+1)*(degree+2)/2;
130 %     T = zeros(2*nx*ny, nen);
131 %     nx_2 = round(nx/2); ny_2 = round(ny/2);
132 %     if degree == 1
133 %         for i=1:ny
134 %             for j=1:nx
135 %                 ielem = 2*((i-1)*nx+j)-1;
136 %                 inode = (i-1)*(npx)+j;
137 %                 nodes = [inode inode+1 inode+npx+1 inode+npx];
138 %                 if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
139 %                     T(ielem,:) = nodes([1,2,3]);
140 %                     T(ielem+1,:) = nodes([1,3,4]);
141 %                 else
142 %                     T(ielem,:) = nodes([1,2,4]);
143 %                     T(ielem+1,:) = nodes([2,3,4]);
144 %                 end
145 %             end
146 %         end
147 %     elseif degree == 2
148 %         for i=1:ny
149 %             for j=1:nx

```

```

150 %           ielem=2*((i-1)*nx+j)-1;
151 %           inode=(i-1)*2*(npx)+2*(j-1)+1;
152 %           nodes = [inode+(0:2)  inode+npx+(0:2)  inode+2*npx
+ (0:2) ];
153 %           if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
154 %               T(ielem,:) = nodes([1 3 9 2 6 5]);
155 %               T(ielem+1,:) = nodes([1 9 7 5 8 4]);
156 %           else
157 %               T(ielem,:) = nodes([1 3 7 2 5 4]);
158 %               T(ielem+1,:) = nodes([3 9 7 6 8 5]);
159 %           end
160 %       end
161 %   end
162 %
163 %   else
164 %       error('not available element')
165 %   end
166 % elseif elem == 11
167 %   if degree == 1
168 %       T = zeros(2*nx*ny,4);
169 %       nx_2 = round(nx/2); ny_2 = round(ny/2);
170 %       for i=1:ny
171 %           for j=1:nx
172 %               ielem = 2*((i-1)*nx+j)-1;
173 %               inode = (i-1)*(npx)+j;
174 %               nodes = [inode  inode+1  inode+npx+1  inode+npx];
175 %               n_ad = npx*npy + 2*((i-1)*nx+j)-1;
176 %               if (i<=ny_2 && j<=nx_2) || (i>ny_2 && j>nx_2)
177 %                   T(ielem,:) = [nodes([1,2,3]), n_ad];
178 %                   T(ielem+1,:) = [nodes([1,3,4]), n_ad+1];
179 %               else
180 %                   T(ielem,:) = [nodes([1,2,4]), n_ad];
181 %                   T(ielem+1,:) = [nodes([2,3,4]), n_ad+1];
182 %               end
183 %           end
184 %       end
185 %   else
186 %       error('not available element')
187 %   end

```

1 % *This program solves a Navier–Stokes problem in a square domain*

2 % *The non–linear problem is solved using Picard iteration*

3

4 **clear; close all; clc**


```

5
6 addpath('Func_ReferenceElement')
7
8 dom = [0,1,0,1];
9
10 Re_v = [100];%500,1000,2000];
11 x=[0.62,0.74;0.55,0.59;0.54,0.568;0.53,0.55];
12 for i=1:length(Re_v)
13 Re=Re_v(i)
14 nu = 1/Re;
15
16 % Element type and interpolation degree
17 % (0: quadrilaterals, 1: triangles, 11: triangles with bubble function)
18 elemV = 0; degreeV = 2; degreeP = 1;
19 % elemV = 1; degreeV = 2; degreeP = 1;
20 % elemV = 11; degreeV = 1; degreeP = 1;
21 if elemV == 11
22     elemP = 1;
23 else
24     elemP = elemV;
25 end
26 referenceElement = SetReferenceElementStokes(elemV, degreeV, elemP, degreeP
    );
27
28 nx =20;% cinput('Number of elements in each direction ',20);
29 ny = nx;
30 adapted = 0;
31 [X,T,XP,TP] = CreateMeshes(dom,nx,ny,referenceElement,adapted);
32
33 figure; PlotMesh(T,X,elemV,'b-');
34 figure; PlotMesh(TP,XP,elemP,'r-');
35
36 % Matrices arising from the discretization
37 [K,G,f] = StokesSystem(X,T,XP,TP,referenceElement);
38 K = K*nu;
39 [ndofP,ndofV] = size(G);
40
41 [dofDir, valDir, dofUnk, confined] = BC_red(X,dom,ndofV);
42 nunkV = length(dofUnk);
43 if confined
44     nunkP = ndofP-1;
45     disp(' ')
46     disp('Confined flow. Pressure on lower left corner is set to zero');
47     G(1,:) = [];

```

```

48 else
49     nunkP = ndofP;
50 end
51
52 Kred = K(dofUnk,dofUnk);
53 Gred = G(:,dofUnk);
54 fred = f - K(:,dofDir)*valDir;
55 fred = fred(dofUnk);
56 A = [Kred    Gred'
57      Gred    zeros(nunkP)];
58
59 % Initial solution
60 disp(' ')
61 IniVelo_file = 0;%input('.mat file with the initial velocity = ','s');
62 if IniVelo_file==0%isempty(IniVelo_file)
63     velo = zeros(ndofV/2,2);
64     y2 = dom(4);
65     nodesY2 = find(abs(X(:,2)-y2) < 1e-6);
66     velo(nodesY2,1) = 1;
67 else
68     load(IniVelo_file);
69 end
70 pres = zeros(nunkP,1);
71 veloVect = reshape(velo',ndofV,1);
72 sol0 = [veloVect(dofUnk);pres(1:nunkP)];
73
74 iter = 0; tol = 0.5e-8;
75 while iter < 100
76     fprintf('Iteration = %d\n',iter);
77
78     C = ConvectionMatrix(X,T,referenceElement,velo);
79     Cred = C(dofUnk,dofUnk);
80
81     Atot = A;
82     Atot(1:nunkV,1:nunkV) = A(1:nunkV,1:nunkV) + Cred;
83     btot = [fred - C(dofUnk,dofDir)*valDir; zeros(nunkP,1)];
84
85     % Computation of residual
86     res = btot - Atot*sol0;
87     % Computation of velocity and pressure increment
88     solInc = Atot\res;
89
90     % Update the solution
91     veloInc = zeros(ndofV,1);

```

```

92     veloInc(dofUnk) = solInc(1:nunkV);
93     presInc = solInc(nunkV+1:end);
94     velo = velo + reshape(veloInc,2,[],)';
95     pres = pres + presInc;
96
97     % Check convergence
98     delta1 = max(abs(veloInc));
99     delta2 = max(abs(res));
100    fprintf('Velocity increment=%8.6e, Residue max=%8.6e\n',delta1,
        delta2);
101    if delta1 < tol*max(max(abs(velo))) && delta2 < tol
102        fprintf('\nConvergence achieved in iteration number %g\n',iter);
103        break
104    end
105
106    % Update variables for next iteration
107    veloVect = reshape(velo',ndofV,1);
108    sol0 = [veloVect(dofUnk); pres];
109    iter = iter + 1;
110 end
111
112 % Postprocess
113 if confined
114     pres = [0; pres];
115 end
116
117 nPt = size(X,1);
118 figure;
119 quiver(X(1:nPt,1),X(1:nPt,2),velo(1:nPt,1),velo(1:nPt,2));
120 hold on
121 plot(dom([1,2,2,1,1]),dom([3,3,4,4,3]),'k')
122 axis equal; axis tight
123
124 PlotStreamlines(X,velo,dom);
125 [p col]=min(pres);
126 plot(XP(col,1),XP(col,2),'xg');
127 plot(x(i,1),x(i,2),'or');
128 if degreeP == 0
129     PlotResults(X,T,pres,referenceElement.elemP,referenceElement.degreeP
        )
130 else
131     PlotResults(XP,TP,pres,referenceElement.elemP,referenceElement.
        degreeP)
132 end

```

```

133 iterres(i)=iter;
134 end
135 figure
136 plot(Re_v, iterres, '-ob');

1 % This program solves the 2D cavity flow Stokes problem
2
3
4 clear; close all; clc
5
6 addpath('Func_ReferenceElement')
7
8 dom = [0,1,0,1];
9
10 % Element type and interpolation degree
11 % (0: quadrilaterals, 1: triangles, 11: triangles with bubble function)
12 % elemV = 0; degreeV = 2; degreeP = 1;
13 % elemV = 1; degreeV = 2; degreeP = 1;
14 elemV = 0; degreeV = 2; degreeP = 1;
15 % elemV = 11; degreeV = 1; degreeP = 1;
16 if elemV == 11
17     elemP = 1;
18 else
19     elemP = elemV;
20 end
21 referenceElement = SetReferenceElementStokes(elemV, degreeV, elemP, degreeP
    );
22
23 nx = 20;%cinput('Number of elements in each direction',20);
24 ny = nx;
25 adapted = 1;
26 [X,T,XP,TP] = CreateMeshes(dom,nx,ny,referenceElement,adapted);
27
28 figure; PlotMesh(T,X,elemV,'b-');
29 figure; PlotMesh(TP,XP,elemP,'r-');
30
31 % Matrices arising from the discretization
32 [K,G,f] = StokesSystem(X,T,XP,TP,referenceElement);
33 [ndofP,ndofV] = size(G);
34
35 [dofDir, valDir, dofUnk, confined] = BC_red(X,dom,ndofV);
36 nunkV = length(dofUnk);
37 if confined
38     nunkP = ndofP-1;

```

```

39     disp(' ')
40     disp('Confined flow. Pressure on lower left corner is set to zero');
41     G(1,:) = [];
42     else
43         nunkP = ndofP;
44     end
45
46     f = f - K(:,dofDir)*valDir;
47     Kred = K(dofUnk,dofUnk);
48     Gred = G(:,dofUnk);
49     fred = f(dofUnk);
50
51     A = [Kred    Gred';
52         Gred    zeros(nunkP)];
53     b = [fred; zeros(nunkP,1)];
54
55     sol = A\b;
56
57     velo = zeros(ndofV,1);
58     velo(dofDir) = valDir;
59     velo(dofUnk) = sol(1:nunkV);
60     velo = reshape(velo,2,[],)';
61     pres = sol(nunkV+1:end);
62     if confined
63         pres = [0; pres];
64     end
65
66     nPt = size(X,1);
67     figure;
68     subplot(2,2,1)
69     quiver(X(1:nPt,1),X(1:nPt,2),velo(1:nPt,1),velo(1:nPt,2));
70     hold on
71     plot(dom([1,2,2,1,1]),dom([3,3,4,4,3]),'k')
72     axis equal; axis tight
73     xlim([0,1]);
74     subplot(2,2,2)
75     PlotStreamlines(X,velo,dom);
76     xlim([0,1]);
77     if degreeP == 0
78         PlotResults(X,T,pres,referenceElement.elemP,referenceElement.degreeP
79         )
80     else
81         subplot(2,2,3)
82         PlotResults(XP,TP,pres,referenceElement.elemP,referenceElement.

```

```

        degreeP)
82 end

1 function PlotMesh(T,X,elem, str ,nonum)
2 % PlotMesh(T,X, str ,nonum)
3 % X: nodal coordinates
4 % T: connectivities
5 % str: linestyle, color and marker used in the plot (optional)
6 % nonum = 1 to show nodes' number(optional)
7
8
9 % Line style and color
10 if nargin == 3
11     str1 = 'yo';
12     str2 = 'y-';
13 else
14     if str(1) == ':' | str(1) == '-'
15         str1 = 'yo';
16         str2 = ['y' str];
17     else
18         str1 = [str(1) 'o'];
19         str2 = str;
20     end
21 end
22
23 nen = size(T,2);
24
25
26 if elem == 0
27     if nen <= 4
28         order = [1:nen,1];
29     elseif nen == 9
30         order = [1,5,2,6,3,7,4,8,1];
31     end
32 elseif elem == 1
33     if nen <= 3
34         order = [1:nen,1];
35     elseif nen == 6
36         order = [1,4,2,5,3,6,1];
37     end
38 elseif elem == 11
39     order = [1:3,1];
40 end
41

```

```

42
43 % Nodes
44 plot(X(:,1),X(:,2),str1)
45 hold on
46 % Elements
47 for j = 1:size(T,1)
48     plot(X(T(j,order),1),X(T(j,order),2),str2)
49 end
50
51
52 % nodes number
53 if nargin==5
54     if nonum==1
55         for I=1:size(X,1)
56             text(X(I,1)+0.02,X(I,2)+0.03,int2str(I),'FontSize',16)
57         end
58     end
59 end
60
61 axis('equal')
62 axis('off')
63
64 hold off

1 function PlotResults(X,T,sol,elem,degree)
2 figure;hold on
3 if elem == 1 && degree == 1
4     figure;
5     trisurf(T,X(:,1),X(:,2),sol,'FaceColor','interp');
6 elseif degree == 0
7     nElem = size(T,1);
8     if elem == 0
9         nen = 4;
10    else
11        nen = 3;
12    end
13    figure; hold on
14    for ielem = 1:nElem
15        Te = T(ielem,1:nen);
16        Xe = X(Te,:);
17        zz = sol(ielem)*ones(nen,1);
18        patch(Xe(:,1),Xe(:,2),zz,zz)
19    end
20 else

```

```

21     [nElem, nen] = size(T);
22     if elem == 11
23         ngeom = nen-1;
24     else
25         ngeom = nen;
26     end
27     if elem == 0
28         npt = 2*degree+1;
29         x = linspace(-1,1,npt);
30         [x,y] = meshgrid(x,x);
31         pts = [reshape(x,npt^2,1), reshape(y,npt^2,1)];
32         ptsEdge = [
33             linspace(-1,1,npt)'           -ones(npt,1)
34             ones(npt,1)                   linspace(-1,1,npt)'
35             flipud(linspace(-1,1,npt)')   ones(npt,1)
36             -ones(npt,1)                   flipud(linspace(-1,1,npt)')
37         ];
38     else
39         npt = 2*degree+1;
40         x = linspace(0,1,npt);
41         [x,y] = meshgrid(x,x);
42         pts = [reshape(x,npt^2,1), reshape(y,npt^2,1)];
43         ind = find(pts(:,2) <= 1 - pts(:,1));
44         pts = pts(ind,:);
45         ptsEdge = [
46             linspace(0,1,npt)'           1-linspace(0,1,npt)'
47             flipud(linspace(0,1,npt)')   zeros(npt,1)
48             zeros(npt,1)                   linspace(0,1,npt)'
49         ];
50     end
51     tri = delaunay(pts(:,1), pts(:,2));
52     N = ShapeFunc(elem, degree, pts);
53     NEdge = ShapeFunc(elem, degree, ptsEdge);
54     hold on;
55     for ielem = 1:nElem
56         Te = T(ielem,:);
57         Xe = X(Te(1:ngeom),:);
58         sol_e = sol(Te);
59         xx = N(:,1:ngeom)*Xe(:,1);
60         yy = N(:,1:ngeom)*Xe(:,2);
61         zz = N*sol_e;
62         xEdge = NEdge(:,1:ngeom)*Xe(:,1);
63         yEdge = NEdge(:,1:ngeom)*Xe(:,2);
64         zEdge = NEdge*sol_e;

```



```

65     trisurf(tri ,xx, yy, zz, 'FaceColor', 'interp', 'EdgeColor', 'none')
66     plot3(xEdge,yEdge,zEdge, 'k');
67     end
68 end
69 grid on
70 view(3); axis tight
71 set(gca, 'FontSize',12)

1 function PlotStreamlines(X,velo ,dom)
2
3 % Define a grid
4 xGrid = linspace(dom(1),dom(2),25);
5 yGrid = linspace(dom(3),dom(4),25);
6 % Interpolate the solution
7 tri = delaunay(X(:,1),X(:,2));
8 uGrid = tri2grid(X',tri',velo(:,1),xGrid,yGrid); % x,y,f are column
   vectors.
9 vGrid = tri2grid(X',tri',velo(:,2),xGrid,yGrid); % x,y,f are column
   vectors.
10 [xGrid,yGrid] = meshgrid(xGrid,yGrid);
11
12 % Points where the streamlines start
13 y1 = min(X(:,2));
14 aux = find(abs(X(:,2)-y1) < 1e-6);
15 xx = X(aux(round(length(aux)/2)),1);
16 ind1 = find(abs(X(:,1)-xx) < 1e-6);
17 xx = X(aux(2),1);
18 ind2 = find(abs(X(:,1)-xx) < 1e-6);
19 xx = X(aux(end-1),1);
20 ind3 = find(abs(X(:,1)-xx) < 1e-6);
21 sx = [X(ind1,1); X(ind2,1); X(ind3,1)];
22 sy = [X(ind1,2); X(ind2,2); X(ind3,2)];
23
24
25
26 %figure;
27 streamline(xGrid,yGrid,uGrid,vGrid,sx,sy)
28 hold on
29 %plot(sx,sy,'r*')
30 plot(dom([1,2,2,1,1]),dom([3,3,4,4,3]),'k')
31 axis equal; axis tight

1 function s = SourceTerm(pt)
2 % s = SourceTerm(pt)

```

```

3
4
5 %  $x = pt(1)$ ;  $y = pt(2)$ ;
6 s = zeros(2,1);

1 function [K,G,f] = StokesSystem(X,T,XP,TP,referenceElement)
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 % XP,TP: nodal coordinates and connectivities for pressure
7 % referenceElement: reference element properties (quadrature, shape
   functions...)
8
9
10 elem = referenceElement.elemV;
11 nGaus = referenceElement.nGaus;
12 wgp = referenceElement.GaussWeights;
13 N = referenceElement.N;
14 Nxi = referenceElement.Nxi;
15 Neta = referenceElement.Neta;
16 NP = referenceElement.NP;
17 ngeom = referenceElement.ngeom;
18
19 % Number of elements and number of nodes in each element
20 [nElem,nenV] = size(T);
21 nenP = size(TP,2);
22
23 % Number of nodes
24 nPt_V = size(X,1);
25 if elem == 11
26     nPt_V = nPt_V + nElem;
27 end
28 nPt_P = size(XP,1);
29
30 % Number of degrees of freedom
31 nedofV = 2*nenV;
32 nedofP = nenP;
33 ndofV = 2*nPt_V;
34 ndofP = nPt_P;
35
36 K = zeros(ndofV,ndofV);
37 G = zeros(ndofP,ndofV);

```

```

38 f = zeros(ndofV,1);
39
40 % Loop on elements
41 for ielem = 1:nElem
42     % Global number of the nodes in element ielem
43     Te = T(ielem,:);
44     TPe = TP(ielem,:);
45     % Coordinates of the nodes in element ielem
46     Xe = X(Te(1:ngeom),:);
47     % Degrees of freedom in element ielem
48     Te_dof = reshape([2*Te-1; 2*Te],1,ndofV);
49     TPe_dof = TPe;
50
51     % Element matrices
52     [Ke,Ge,fe] = EleMatStokes(Xe,ngeom,ndofV,ndofP,ngaus,wgp,N,Nxi,
        Neta,NP);
53
54     % Assemble the element matrices
55     K(Te_dof, Te_dof) = K(Te_dof, Te_dof) + Ke;
56     G(TPe_dof,Te_dof) = G(TPe_dof,Te_dof) + Ge;
57     f(Te_dof) = f(Te_dof) + fe;
58 end
59
60
61
62
63
64
65 function [Ke,Ge,fe] = EleMatStokes(Xe,ngeom,ndofV,ndofP,ngaus,wgp,N,
    Nxi,Neta,NP)
66 % [Ke,Ge,fe] = EleMatStokes(Xe,ngeom,ndofV,ndofP,ngaus,wgp,N,Nxi,Neta,
    NP)
67
68 Ke = zeros(ndofV,ndofV);
69 Ge = zeros(ndofP,ndofV);
70 fe = zeros(ndofV,1);
71 % Loop on Gauss points
72 for ig = 1:ngaus
73     N_ig = N(ig,:);
74     Nxi_ig = Nxi(ig,:);
75     Neta_ig = Neta(ig,:);
76     NP_ig = NP(ig,:);
77     Jacob = [
78         Nxi_ig(1:ngeom)*(Xe(:,1))      Nxi_ig(1:ngeom)*(Xe(:,2))

```

```

79         Neta_ig(1:ngeom)*(Xe(:,1))      Neta_ig(1:ngeom)*(Xe(:,2))
80     ];
81     dvolu = wgp(ig)*det(Jacob);
82     res = Jacob\[Nxi_ig;Neta_ig];
83     nx = res(1,:);
84     ny = res(2,:);
85
86     Ngp = [reshape([1;0]*N_ig,1,nedofV); reshape([0;1]*N_ig,1,nedofV
87             )];
88     % Gradient
89     Nx = [reshape([1;0]*nx,1,nedofV); reshape([0;1]*nx,1,nedofV)];
90     Ny = [reshape([1;0]*ny,1,nedofV); reshape([0;1]*ny,1,nedofV)];
91     % Divergence
92     dN = reshape(res,1,nedofV);
93
94     Ke = Ke + (Nx'*Nx+Ny'*Ny)*dvolu;
95     Ge = Ge - NP_ig'*dN*dvolu;
96     x_ig = N_ig(1:ngeom)*Xe;
97     f_igaus = SourceTerm(x_ig);
98     fe = fe + Ngp'*f_igaus*dvolu;
99 end

```