HM3 – REPORT

Stokes and Navier-Stokes

Finite Element in Fluids

Álvaro Rodríguez Luis

1. Stokes problem

The problem to be solved is finding the steady flow of a highly viscous (the convective term can be neglected) isotropic incompressible fluid in a 2D domain, where certain boundary conditions are applied. This is done solving the Stokes steady equation:

$$\begin{cases} -\nu\nabla^2\boldsymbol{v} + \boldsymbol{\nabla} p = \boldsymbol{f} & \text{in } \Omega \\ \boldsymbol{\nabla}\cdot\boldsymbol{v} = 0 & \text{in } \Omega \\ \boldsymbol{v} = \boldsymbol{v}_D & \text{on } \partial\Omega \end{cases}$$

using the provided code. Based on the code, the dynamic viscosity for this problem is $\nu = 1$ and the domain Ω is a square of size 1×1, with corners $\{(0,0), (1,0), (1,1), (0,1)\}$. The source term, is set to zero in the MATLAB function "`SourceTerm`". The boundary conditions are set in the MATLAB function "`BC_red`", where the velocity is fixed to zero in all sides except for the side $y = 1$, where the velocity is set to $\boldsymbol{v} = (1,0)$. The velocity in the four corners is set to zero. These boundary conditions introduce a discontinuity in the corners $(0,1)$ and $(1,1)$. Also, the pressure is set to zero in the corner $(0,0)$.

First, the problem is solved using different element types:

- **Q1Q1**: quadrilateral mesh, linear in velocity and pressure. In the code:

```
elemV = 0; degreeV = 1; degreeP = 1;
```

- **Q2Q1**: quadrilateral mesh, quadratic in velocity, linear in pressure.

```
elemV = 0; degreeV = 1; degreeP = 1;
```

- **P1P1**: triangular mesh, linear in velocity and pressure.

```
elemV = 0; degreeV = 1; degreeP = 1;
```

- **P2P1**: triangular mesh, quadratic in velocity, linear inpressure.

```
elemV = 0; degreeV = 1; degreeP = 1;
```

And the following results are obtained for the pressure fields:

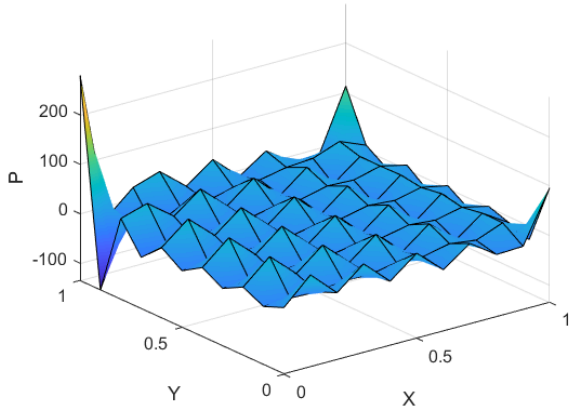

Figure 1. Mesh type Q1Q1

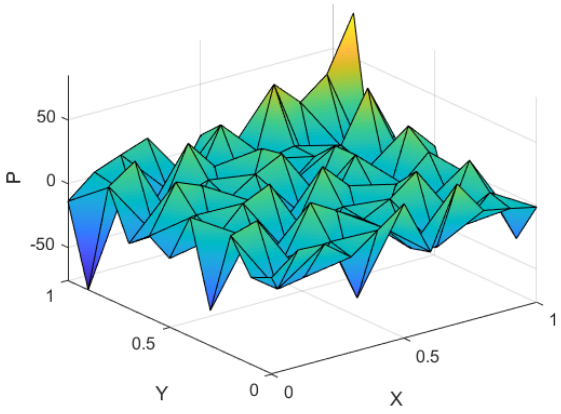

Figure 3. Mesh type P1P1

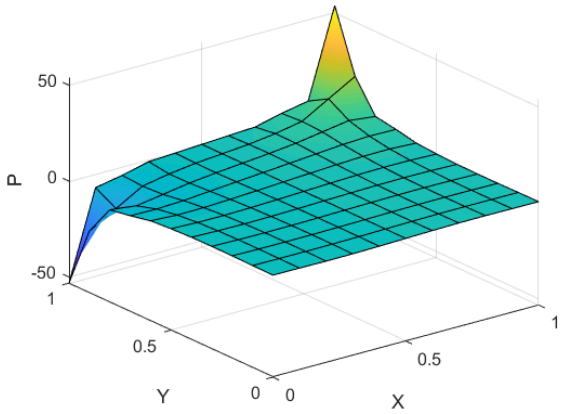

Figure 2. Mesh type Q2Q1

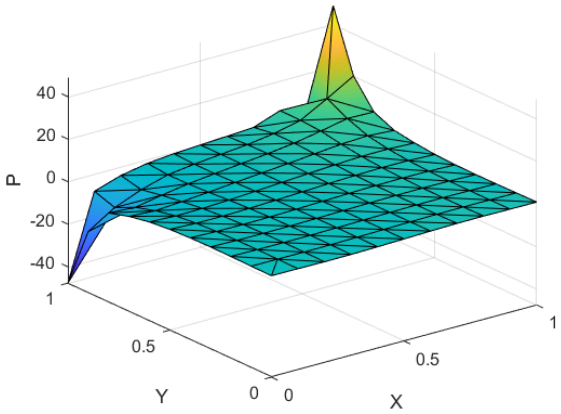

Figure 4. Mesh type P2P1

The results show that only meshes Q2Q1 and P2P1 are stable. Also, a singularity in the pressure is observed in corners $(0,1)$ and $(1,1)$, where the discontinuity on the velocity field is imposed with the boundary conditions.

In order to obtain stable results with the Q1Q1 and the P1P1 meshes, GLS stabilization is implemented. With no stabilization, the discretization with FEM of the weak form of the stokes equation yields the following system:

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

Where the matrices expressions are detailed in the class notes, and are already implemented in the code.

For linear elements in velocity and pressure, with GLS stabilization, the original system is modified as follows

$$\begin{pmatrix} \mathbf{K} & \mathbf{G}^T \\ -\mathbf{G} & \overline{\mathbf{L}} \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \overline{\mathbf{f}}_q \end{pmatrix}$$

$$\overline{\mathbf{L}} \leftarrow \sum_e \int_{\Omega_e} \tau_1 (\boldsymbol{\nabla} q) \cdot (\boldsymbol{\nabla} p) d\Omega$$

$$\overline{\mathbf{f}}_q \leftarrow \sum_e \int_{\Omega_e} \tau_1 (\boldsymbol{\nabla} q) \cdot (-\boldsymbol{f}) d\Omega$$

$$\tau_1 = \alpha_0 \frac{h^2}{4\nu}$$

Where h is the element side size, and $\alpha_0 = 1/3$ is optimal for linear elements.

These modifications are implemented in the provided code as shown in the Appendix, where the changes are highlighted in yellow, and the following results for the pressure field are obtained:

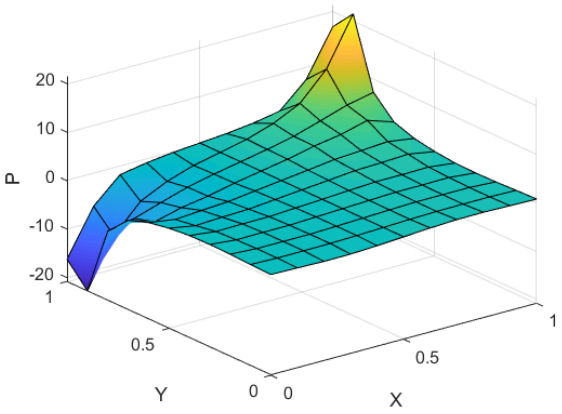

Figure 5. Mesh type Q1Q1 with GLS stabilization.

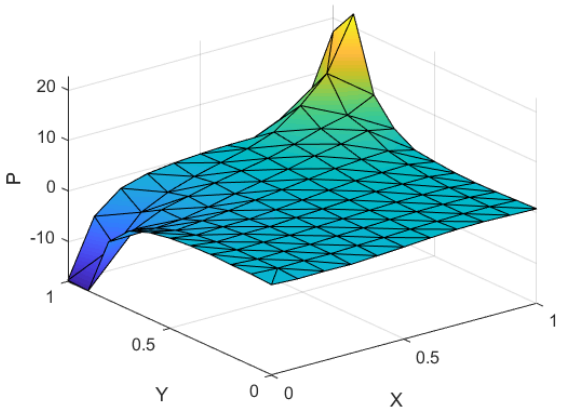

Figure 6. Mesh type P1P1 with GLS stabilization.

Figures 5 and 6 show how GLS stabilization worked properly stabilizing the results, but the obtained results are not equal to the results with Q2Q1 and P2P1 meshes.

2. Navier-Stokes problem

The problem to be solved is finding the steady flow of an isotropic incompressible fluid in a 2D domain, where certain boundary conditions are applied. This is done solving the Navier-Stokes steady equation:

$$\begin{cases} -\nu\nabla^2\boldsymbol{v} + (\boldsymbol{v}\cdot\boldsymbol{\nabla})\boldsymbol{v} + \boldsymbol{\nabla}p = \boldsymbol{f} & \text{in } \Omega \\ \boldsymbol{\nabla}\cdot\boldsymbol{v} = 0 & \text{in } \Omega \\ \boldsymbol{v} = \boldsymbol{v}_D & \text{on } \partial\Omega \end{cases}$$

using the provided code. Based on the code, the dynamic viscosity for this problem is $\nu = 1$, and the domain and the boundary conditions are the same as the described for the Stokes problem in previous section. The discretization with FEM of the weak form of the stokes equation yields the following non-linear system:

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

Where the matrices expressions are detailed in the class notes, and are already implemented in the code, except for the convective term, $\mathbf{C}(\mathbf{v})$, that had to be included, using the following expression:

$$\mathbf{C}(\mathbf{a})\cdot\mathbf{v} = \int_{\boldsymbol{\Omega}} \boldsymbol{\omega}\cdot(\mathbf{a}\cdot\boldsymbol{\nabla})\mathbf{v}\ \mathbf{d}\boldsymbol{\Omega}$$

The modified code is shown in the Appendix, with the modifications highlighted in yellow. Due to the convective term, the system is non-linear, thus an iterative method must be used to solve it. In the given code the Picard method was already implemented, and the following results are obtained solving the problem with this method and a Q2Q1 mesh:

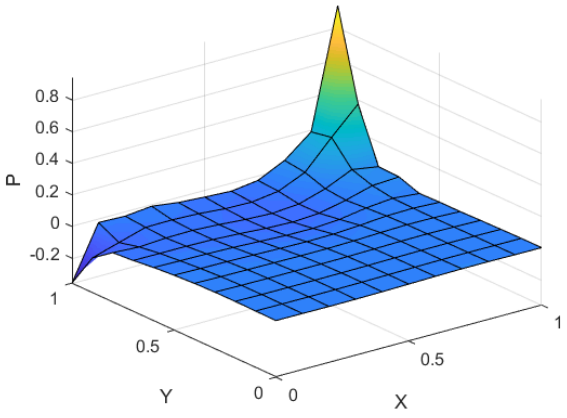

Figure 7. Pressure field results with Picard method and Q2Q1 mesh.

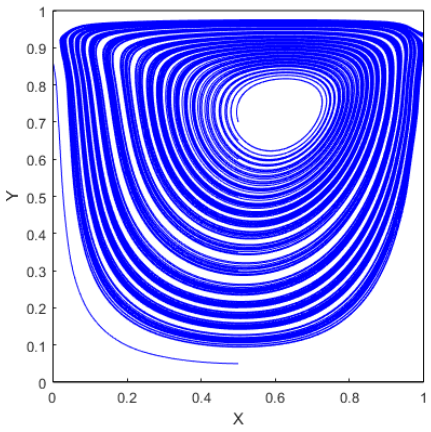

Figure 8. Stream line results with Picard method and Q2Q1 mesh.

Also, with the Picard method, the convergence was found to be linear, and for this case, 13 iterations were required to solve the problem, as shown in Figure 9.

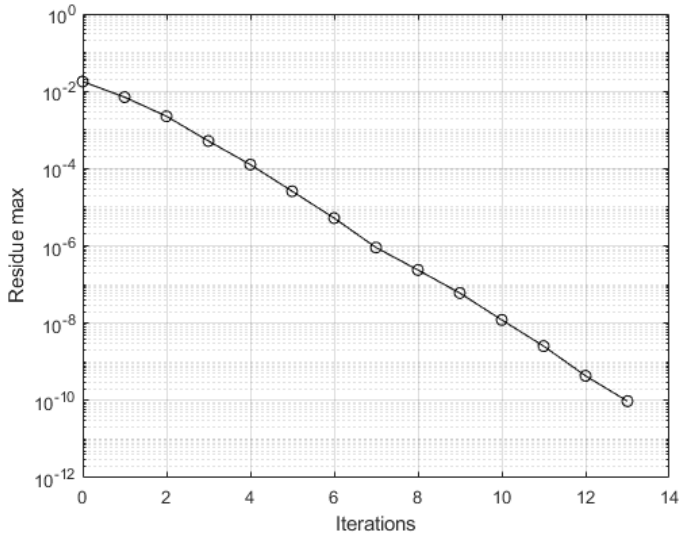

Figure 9. Convergence with Picard method and Q2Q1 mesh.

Finally, the problem can be solved with a Newton-Raphson method which was not implemented in the provided code. Newton-Raphson is an iterative method used for solving non-linear systems, $\mathbf{r}(\mathbf{x}) = \mathbf{0}$, given an initial estimation of the solution, $\mathbf{x^0}$. At each iteration, a linear system is solved and then the solution approximation is updated until the convergence criterium is met:

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

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}^{k+1}$$

where **J** is the Jacobian matrix of the function **r**. In this case:

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

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

where order to compute the matrix $\mathbf{C}^*(\mathbf{v}) = \frac{\partial \mathbf{C}(\mathbf{v})}{\partial \mathbf{v}} \cdot \mathbf{v}$, the following expression is used:

$$\mathbf{C}^*(\mathbf{a}) \cdot \mathbf{v} = \int_{\Omega} \boldsymbol{\omega} \cdot \boldsymbol{\nabla}\mathbf{v} \cdot \mathbf{a} \; d\boldsymbol{\Omega}$$

Once again, the modified code is shown in the Appendix, with the modifications highlighted in yellow.

In this case, the results were exactly the same to the results obtained for the Picard method, but the convergence was worse.

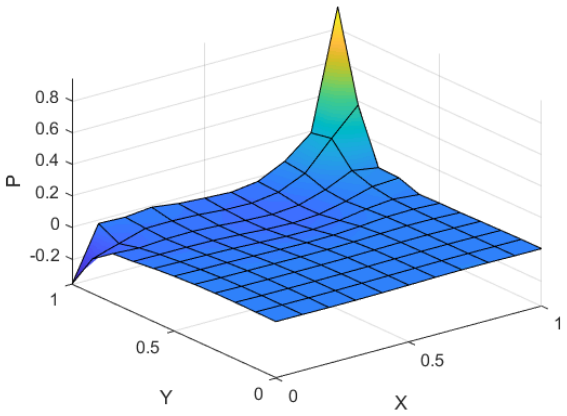

Figure 10. Pressure field results with NR method and Q2Q1 mesh.

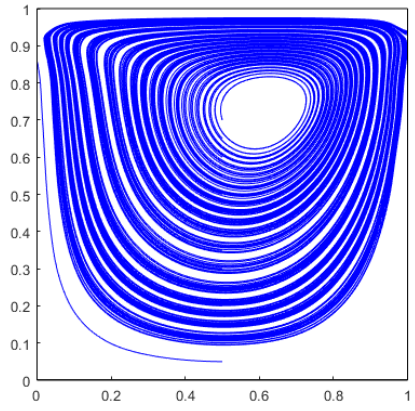

Figure 11. Stream line results with NR method and Q2Q1 mesh.

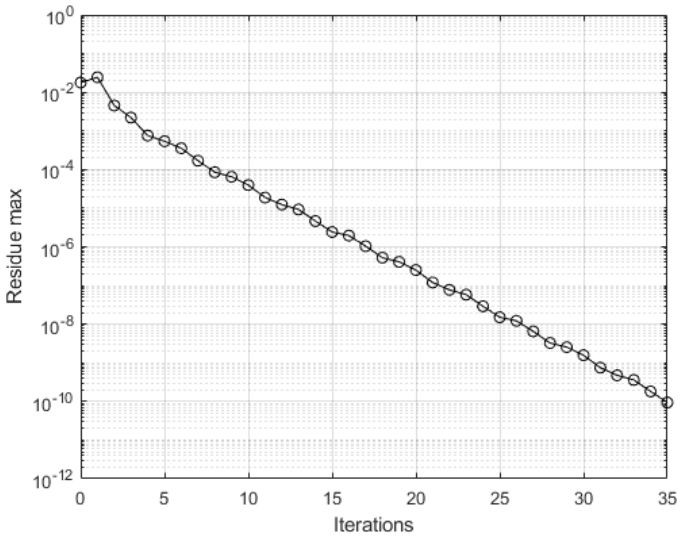

Figure 12. Convergence with NR method and Q2Q1 mesh.

Annex – Code modifications

Stokes problem

`mainStokes.m`

```
%... (previous code not changed)

% Matrices arising from the discretization
[K,G,L,f,fq] = StokesSystem(X,T,XP,TP,referenceElement);
K = mu*K;
[ndofP,ndofV] = size(G);

h = 0.5*(abs(dom(2)-dom(2))/nx + abs(dom(4)-dom(3))/nx);
tau = (1/3)*h^2/(4*mu);
L = tau*L;
fq = tau*fq;


% Prescribed velocity degrees of freedom
[dofDir,valDir,dofUnk,confined] = BC_red(X,dom,ndofV);
nunkV = length(dofUnk);

% Total system of equations
if confined
   nunkP = ndofP-1;
   disp(' ')
   disp('Confined flow. Pressure on lower left corner is set to zero');
   G(1,:) = [];
   L(1,:) = [];
   L(:,1) = [];
   fq(1) = [];
else
   nunkP = ndofP;
end

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

A = [Kred   Gred';
     Gred   L];
b = [fred;  fq];

%... (following code not changed)
```

StokesSystem.m

```
function [K,G,L,f,fq] = 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;
degV = referenceElement.degreeV;
degP = referenceElement.degreeP;
flag_GLS = 0;
if degV==1 && degP==1
    flag_GLS = 1;
end
% 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);
L = zeros(ndofP,ndofP);
f = zeros(ndofV,1);
fq = zeros(ndofP,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,Le,fe,fqe] = EleMatStokes(Xe,ngeom,nedofV,nedofP, ...
                                     ...ngaus,wgp,N,Nxi,Neta,NP,flag_GLS);
    % 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;
    L(TPe_dof,TPe_dof) = L(TPe_dof,TPe_dof) + Le;
    f(Te_dof) = f(Te_dof) + fe;
    fq(TPe_dof) = fq(TPe_dof) + fqe;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Ke,Ge,Le,fe,fqe] = EleMatStokes(Xe,ngeom,nedofV,nedofP, ...
                                          ... ngaus,wgp,N,Nxi,Neta,NP,flag_GLS)
Ke = zeros(nedofV,nedofV);
Ge = zeros(nedofP,nedofV);
Le = zeros(nedofP,nedofP);
fe = zeros(nedofV,1);
fqe = zeros(nedofP,1);
if flag_GLS~=0 && flag_GLS~=1
    error('GLS flag must be either 0 or 1')
end
% 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;
    if flag_GLS == 1  % GLS only for linear elements in velocity and pressure
        Le = Le - (nx'*nx+ny'*ny)*dvolu;
        fqe = fqe + [nx; ny]'*f_igaus*dvolu;
    end
end
```

Navier-Stokes problem

mainNavierStokes.m

```
%...previous code (omitted)
method = 1; %[0: Piccard, 1:NR]
if method == 0
    %...Provided Picard algorithm (omitted)
elseif method == 1
    while iter < 100
        fprintf('Iteration = %d\n',iter);
        iterVec = [iterVec iter];

        [C,dC] = ConvectionMatrix(X,T,referenceElement,velo);
        Cred = C(dofUnk,dofUnk);
        dCred = dC(dofUnk,dofUnk);

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

        J = Atot;
        J(1:nunkV,1:nunkV) = J(1:nunkV,1:nunkV) + dCred;

        % Computation of residual
        res = Atot*sol0-btot;
        % Computation of velocity and pressure increment
        solInc = -J\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);
        resVec = [resVec 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
else
    error('Iterative method not available')
end
%...following code (omitted, includes new lines to plot the convergence graph)
```

`ConvectionMatrix.m` (the whole function is new)

```
function [C,dC] = ConvectionMatrix(X,T,referenceElement,velo)
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);
dC = 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 at the nodes in element ielem
    Ve = velo(Te(1:ngeom),:);
    % Degrees of freedom in element ielem
    Te_dof = reshape([2*Te-1; 2*Te],1,nedofV);
    % Element matrices
    [Ce,dCe] = EleMatC(Xe,Ve,ngeom,nedofV,ngaus,wgp,N,Nxi,Neta);
    % Assemble the element matrices
    C(Te_dof, Te_dof) = C(Te_dof, Te_dof) + Ce;
    dC(Te_dof, Te_dof) = dC(Te_dof, Te_dof) + dCe;end

end
```

```
function [Ce,dCe] = EleMatC(Xe,Ve,ngeom,nedofV,ngaus,wgp,N,Nxi,Neta)

Ce = zeros(nedofV,nedofV);
dCe = zeros(nedofV,nedofV);

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)];
    % 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)];

    v_ig = N_ig(1:ngeom)*Ve;
    Ce = Ce + Ngp'*(v_ig(1)*Nx+v_ig(2)*Ny)*dvolu;
    dCe = dCe + Ngp'*[nx; ny]*Ve*Ngp*dvolu;

end

end
```