UNIVERSITAT POLITÈCNICA DE CATALUNYA

BARCELONATECH

MSc. in Computational Mechanics
Finite Element Methods in Fluids (FEF)
Spring Semester 2017/2018

FINAL ASSIGNMENT

Submitted By:
Luan Malikoski Vieira

June 1, 2018
Barcelona, Spain

Contents

1 Introduction **1**

2 Problem I **1**
2.1 Weak Formulation and solution method 1
2.2 Results 2

3 Problem II **4**
3.1 Weak Formulation and solution method 4
3.1.1 Stabilized formulation for P1P1 interpolation 5
3.2 Results 6

4 Problem III **7**
4.1 Weak Formulation and solution method 7
4.1.1 General Procedure: Iterative Method to solve the coupling $\mathbf{u}$/F 8
4.1.2 Simplified Procedure to solve the coupling $\mathbf{u}$/F 9
4.2 Results 10

5 Annex - MatLab Routines **13**

1 Introduction

This report will cover the solution of the three proposed problems. Each problem will be discussed in different sections. Each section will be composed by a brief explanation on the derivation of the weak form of the governing equations and solution method. In last part, some results will be discussed.

Routines implemented in Matlab are in the Annex section. Only new created routines or routines that had important modifications are present in the Annex in order to shorten the report.

2 Problem I

The following set of unsteady advection-diffusion-reaction partial differential equations governs the problem in the time-space domain (0,T) x Ω:

$$\frac{\partial F}{\partial t} = -\mathbf{u}\cdot\nabla F + D_F\nabla^2 F - \sigma_F F$$

$$\frac{\partial G}{\partial t} = D_G\nabla^2 G - \sigma_G G + \hat{\sigma}_{GF} F$$

with $\mathbf{u} = -1/1500(rx, ry)\mu m/s$, $D_F = 5\mu m/s$, $\sigma_F = 0.25s^{-1}$, $D_G = 15\mu m/s$, $\sigma_G = 2s^{-1}$ and the coupling constant $\hat{\sigma}_{GF} = 0.5s^{-1}$. For F, Dirichlet boundary condition at the domain upper boundary is imposed such that $F = 80\mu N$ in Γ_D. Homogeneous Neumann boundary condition are imposed for all other boundaries. For G, all boundary conditions are of Homogeneous Neumann type.

2.1 Weak Formulation and solution method

Here, the time integration technique implemented is the general formulation of the implicit Padè approximations. After discretizing in time, applying the Galerkin Weighted residual method and integrating the higher order derivative terms by parts (neglecting boundary terms), the weak form to be solved at each time-step for F is found.

$$\begin{aligned}\int_\Omega \omega\frac{\Delta\mathbf{F}}{\Delta t}d\Omega + \mathbf{W}\left[\int_\Omega \omega(\mathbf{u}\cdot\nabla)\Delta\mathbf{F}d\Omega + \int_\Omega \nabla\omega\cdot(D_F\nabla\Delta\mathbf{F})d\Omega + \int_\Omega \omega\sigma_F\Delta\mathbf{F}d\Omega\right] = \\ -\mathbf{w}\left[\int_\Omega \omega(\mathbf{u}\cdot\nabla)F^n d\Omega + \int_\Omega \nabla\omega\cdot(D_F\nabla F^n)d\Omega + \int_\Omega \omega\sigma_F F^n d\Omega\right]\end{aligned} \tag{1}$$

Following the same procedure the weak form for G is found.

$$\begin{aligned}\int_\Omega \omega\frac{\Delta\mathbf{G}}{\Delta t}d\Omega + \mathbf{W}\left[\int_\Omega \nabla\omega\cdot(D_G\nabla\Delta\mathbf{G})d\Omega + \int_\Omega \omega\sigma_G\Delta\mathbf{G}d\Omega\right] = \\ -\mathbf{w}\left[\int_\Omega \nabla\omega\cdot(D_G\nabla G^n)d\Omega + \int_\Omega \omega\sigma_G G^n d\Omega\right] + \int_\Omega \omega\hat{\sigma_{GF}}\left(\mathbf{w}F^n + \mathbf{W}\Delta\mathbf{F}\right)d\Omega\end{aligned} \tag{2}$$

Where ω is the test function. $\mathbf{W}$ and $\mathbf{w}$ are the Padé matrix and vector respectively.

After interpolating ω, F and G with chosen shape functions and integrating in the appropriate gauss points, the following system of linear equations, which will be solved in each time-step, is found from equations (1) and (2)

$$\left[\mathbf{M} + \Delta t\mathbf{W}\left(\mathbf{C} + D_F\mathbf{K} + \sigma_F\mathbf{M}\right)\right]\Delta\mathbf{F} = -\Delta t\mathbf{w}\left(\mathbf{C} + D_F\mathbf{K} + \sigma_F\mathbf{M}\right)F^n \tag{3}$$

$$\left[\mathbf{M} + \Delta t\mathbf{W}\left(D_G\mathbf{K} + \sigma_G\mathbf{M}\right)\right]\Delta\mathbf{G} = -\Delta t\mathbf{w}\left(D_G\mathbf{K} + \sigma_G\mathbf{M}\right)G^n + \Delta t\hat{\sigma_{GF}}\left(\mathbf{w}\mathbf{M}F^n + \mathbf{W}\mathbf{M}\Delta\mathbf{F}\right) \tag{4}$$

The imposition of boundary conditions for both equations is done by means of Lagrange Multipliers technique as follows:

$$\begin{bmatrix} \mathbf{A} & ADir' \\ ADir & 0 \end{bmatrix} \begin{pmatrix} \Delta\mathbf{F} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{B}F^n \\ bDir \end{pmatrix} \tag{5}$$

Where $\mathbf{A}$ and $\mathbf{B}$ are the l.h.s. and r.h.s. of equation (4) respectively. For G the procedure is similar.

As the coupling is in only one direction, in the sense that G depends on F but the opposite is not true, the two equations can be solved separately within each time-step. In the practical point of view, equation (3), or system 5, is solved for $\Delta\mathbf{F}$ which is inserted, along with F^n, into the rhs of equation (4) leading to the solution for $\Delta\mathbf{G}$. This is done step by step until the end time T.

2.2 Results

The problem was solved using a 30x30 Q1Q1 mesh. Figure (1) shows the convective velocity vector and Figure(2) shows the surface for the convective velocity in directions y and x.

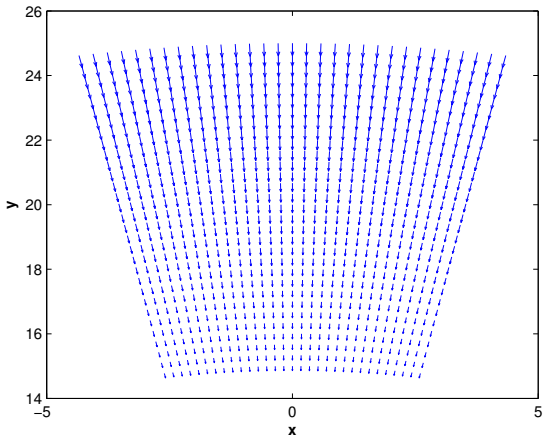
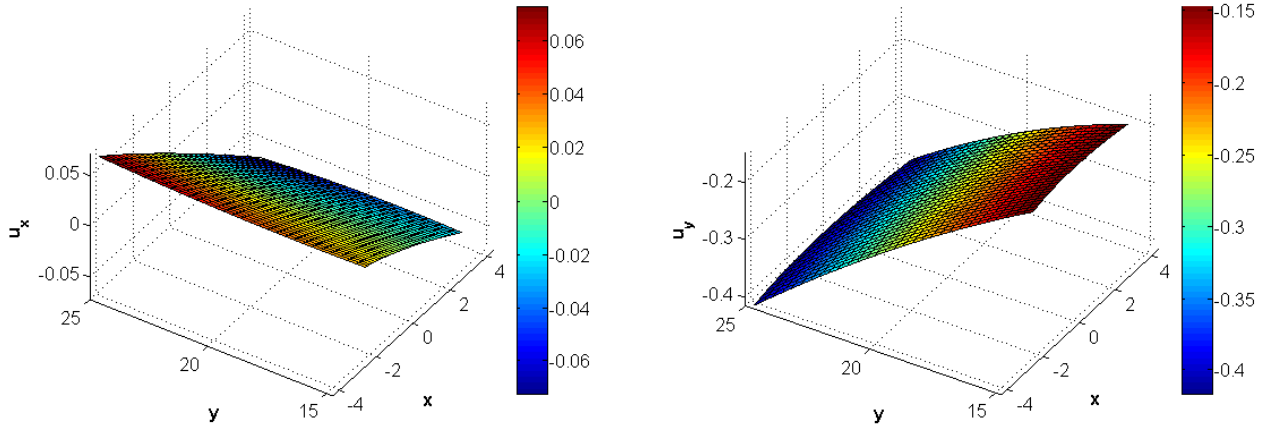


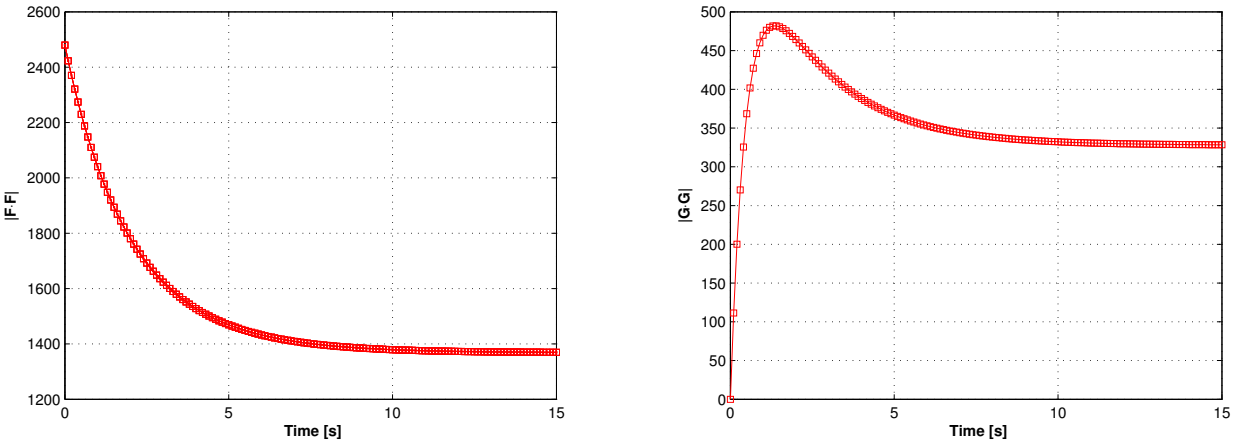
Figure 1: Convective velocity vector field.



(a) (b)

Figure 2: Velocity - (a) x direction; (b) y direction.

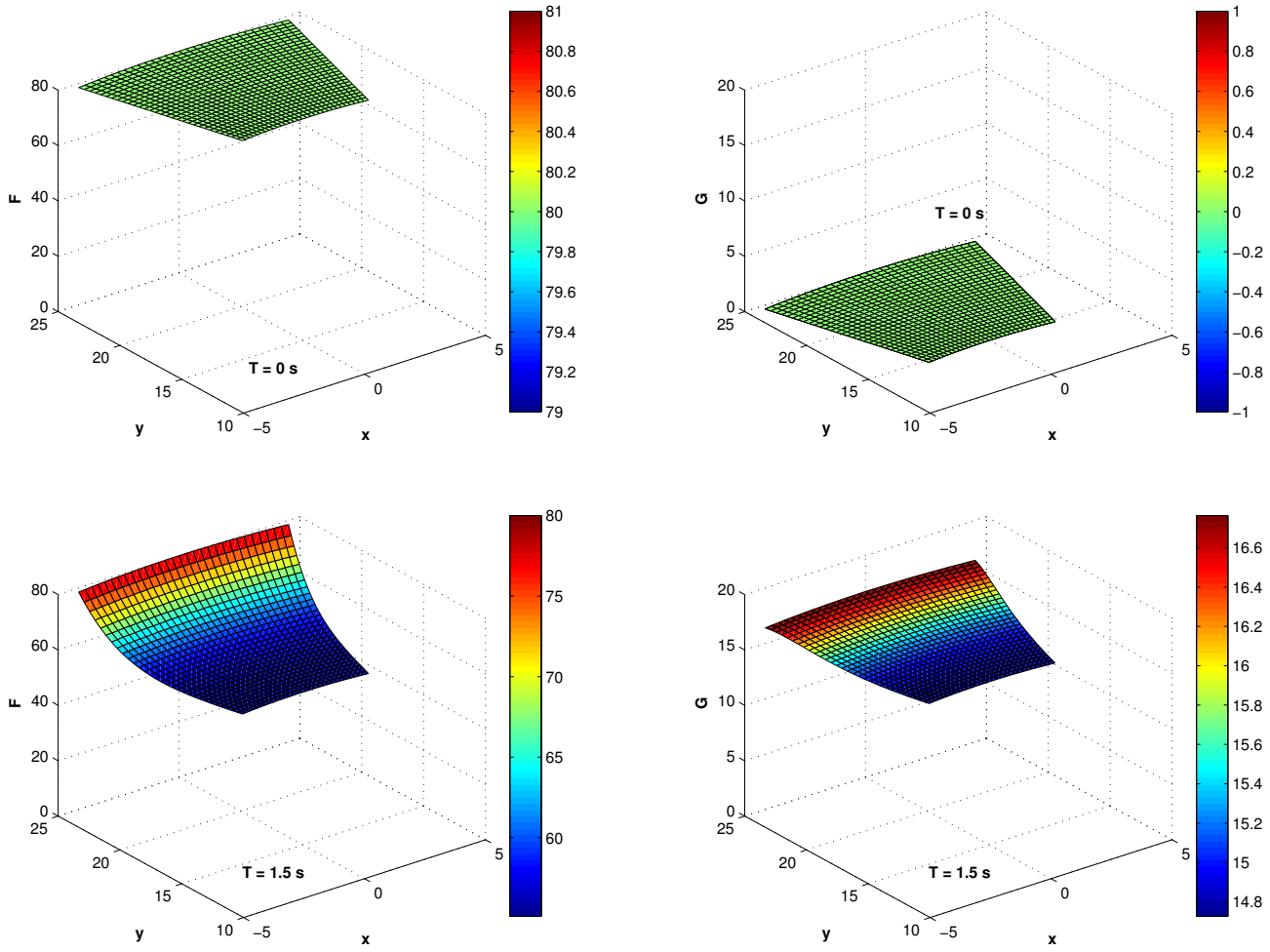
The time integration was performed using the Crank-Nicholson method (R22 and R33 padé are also available) with time-step $\Delta t = 0.1$. In order to define the stationary solution, the norm of the FEM approximated solution for F and G, given by $\sqrt{F^h \cdot F^h}$ and $\sqrt{G^h \cdot G^h}$ respectively, were tracked during the integration. Figure (3) show the evolution of the norm of each approximation.

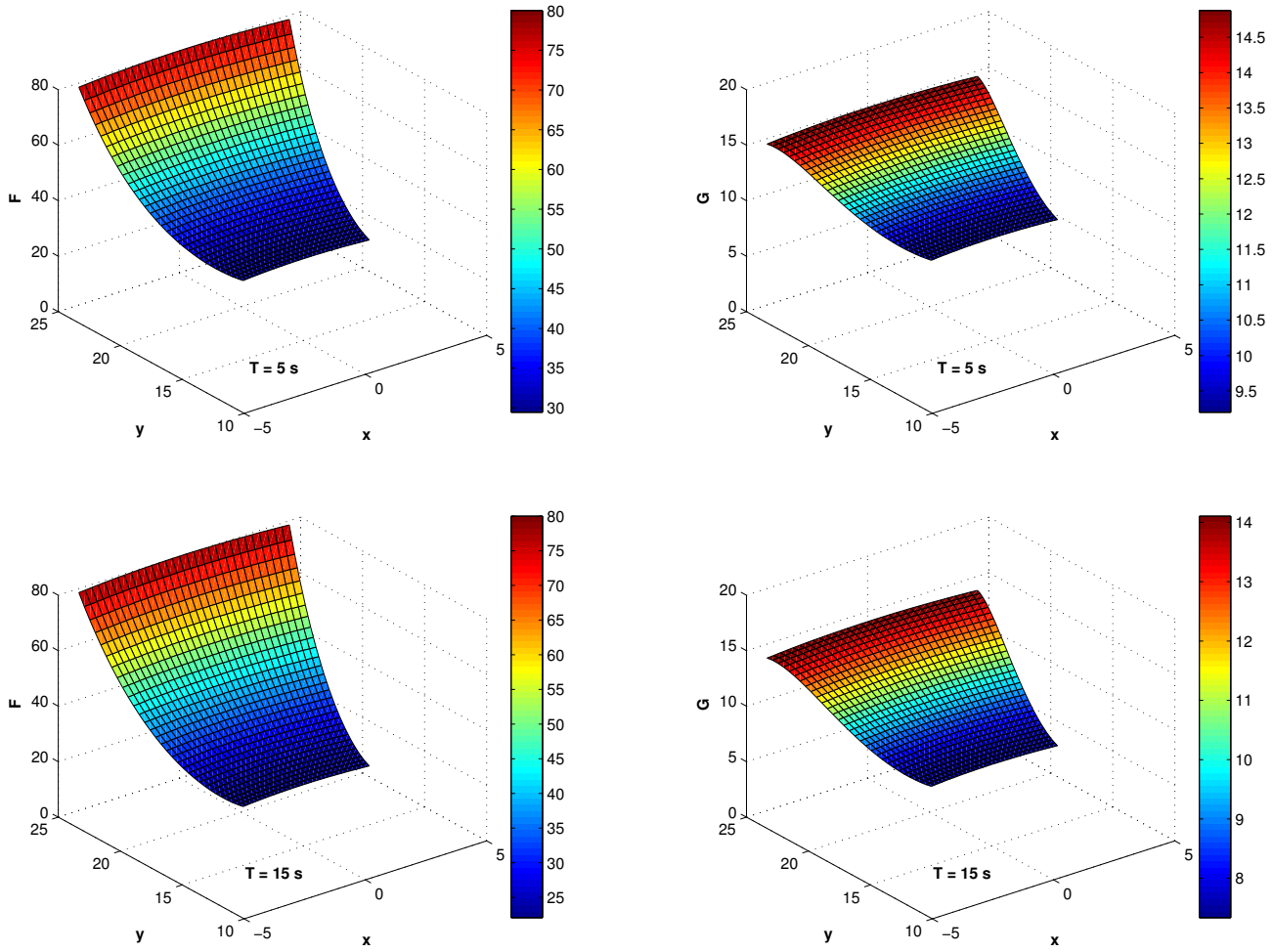


(a) (b)

Figure 3: Norm of approximated soluiton - (a) F; (b) G.

It can be noticed that for time around 15 s, stationary solution is reached by both variables. The time evolution of F and G, from initial condition until stationry state at T = 15 s is show in next figures. Initial condition for both F and G were chosen as uniform, fulfilling the boundary conditions. For G zero initial condition is imposed.





From the time evolution figures it can be noted that undergoes a overshoot in the solution between T = 0s and T = 5s, as also observed in the norm graph of figure (3). This is caused by the sudden imposition of the force F in such a way that reaction can not countermeasure causing the overshoot. However after a time, amplitude is alleviated by reaction and diffusion effects.

3 Problem II

The following set of equations governs the stokes problem in the space domain Ω:

$$\nabla \cdot \sigma = \mathbf{0}$$

$$\nabla \cdot u = 0$$

With Dirichlet boundary condition such as $u_r(r = 15) = -0.15$, $u_\theta(r = 15) = 0$,$u_r(r = 25) = -0.30$ and $u_\theta(r = 25) = 0$. Homogeneous Neumann boundary condition is imposed in all other boundaries.

3.1 Weak Formulation and solution method

For this particular stokes problem, the weak formulation for finite element approximation reads:

$$a(w^h, v^h) + b(w^h, p^h) = -a(w^h, v_D^h) \tag{6}$$

$$b(v^h, q^h) = -b(v_D^h, q^h) \tag{7}$$

Where:

$$a(w, v) = \int_\Omega \dot{\epsilon}(w)^T C_v \dot{\epsilon}(v) d\Omega = \mathbf{K} u^h$$

$$b(w, p) = \int_\Omega p \nabla \cdot w d\Omega = \mathbf{G^T} p^h$$

Using Lagrange multipliers technique, where p is the multiplier, the system of equations to be solved becomes:

$$\begin{bmatrix} K & G^T \\ G & 0 \end{bmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix} \tag{8}$$

Using Lagrange multipliers technique to impose boundary conditions, the final system of equations is found. With $ADir$ and $bDir$ are the boundary matrix and vector with BC values respectively.

$$\begin{bmatrix} K & ADir' & G^T \\ ADir & 0 & 0 \\ G & 0 & 0 \end{bmatrix} \begin{pmatrix} u \\ \lambda \\ p \end{pmatrix} = \begin{pmatrix} 0 \\ bDir \\ 0 \end{pmatrix} \tag{9}$$

3.1.1 Stabilized formulation for P1P1 interpolation

The stabilized Galerkin formulation by a Least Square approach (GLS), for this particular problem of P1P1 interpolation is found in equations (10) and (11). Where $\tau_e = 1/3$ is used as an optimal value for bilinear quadrilateral elements.

$$a(w^h, v^h) + b(w^h, p^h) = -a(w^h, v_D^h) \tag{10}$$

$$b(v^h, q^h) - \sum_{e=1}^{n_{el}} \tau_e (\nabla q^h, \nabla p^h)_{\Omega_e} = -b(v_D^h, q^h) - \sum_{e=1}^{n_{el}} \tau_e (\nabla q^h, b^h)_{\Omega_e} \tag{11}$$

As linear elements are used for both velocity and pressure interpolation, the weak form of momentum equation is not changed. Only the compressibility constrains is changed. Thus, two new tensors are added to the problem corresponding to the two terms in the equation (12), named here as A_e and h_e. It the present case h_e is identically zero as body forces $\mathbf{b}$ are inexistent in the model.

Consequently the system of equations to be solved becomes:

$$\begin{bmatrix} K & G^T \\ G & A_e \end{bmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ h + h_e \end{pmatrix} \tag{12}$$

Using Lagrange multipliers technique to impose boundary conditions, the final system of equations becomes:

$$\begin{bmatrix} K & ADir' & G^T \\ ADir & 0 & 0 \\ G & 0 & Ae \end{bmatrix} \begin{pmatrix} u \\ \lambda \\ p \end{pmatrix} = \begin{pmatrix} 0 \\ bDir \\ h_e \end{pmatrix} \tag{13}$$

3.2 Results

The problem was solved using a 30x30 Q1Q1 mesh, which is not a LBB compliant coupling. Standard and stabilized Galerkin formulations were tested. Figure (4) show the results for the vector velocity field.

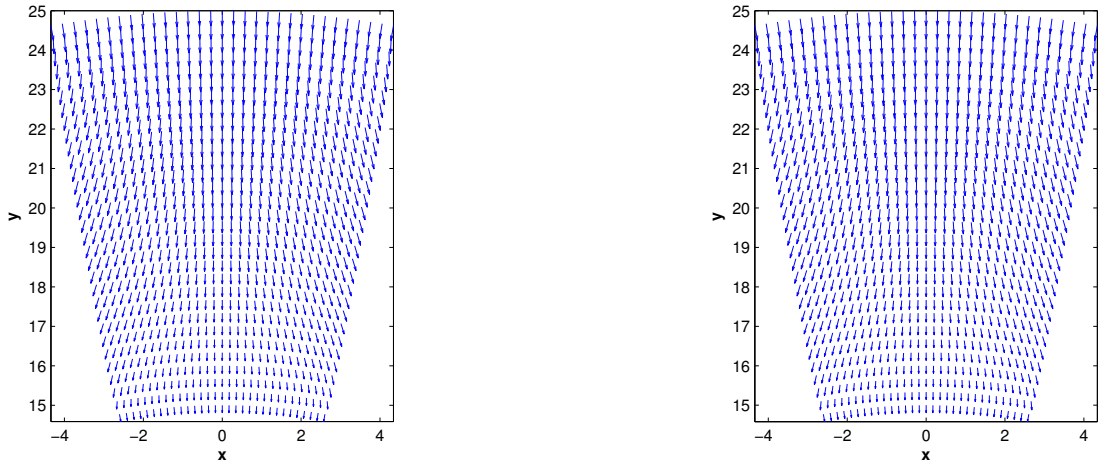


Figure 4: Velocity vector field - (a) Standard; (b) Stabilized

Surfaces for the velocity in x and y, for both formulations are shown in Figures (5) and (6). From the results in velocity, as expected, both formulations presents similar results.

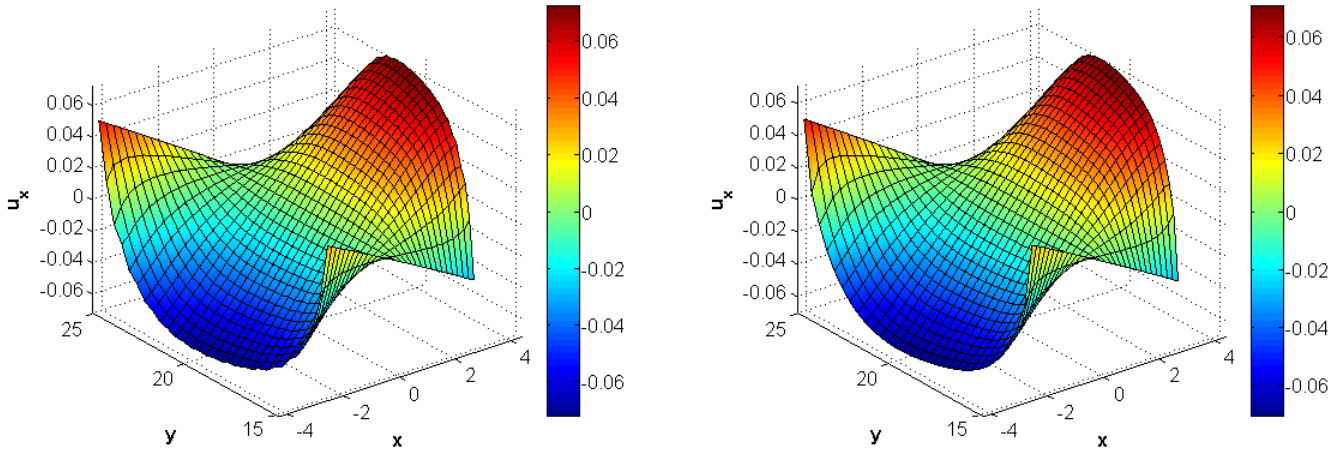


Figure 5: Velocity in x direction - (a) Standard; (b) Stabilized

The stabilization in pressure to eliminate spurious nodes, which is seek here, is observed in Figure (7). Peaks in the pressure appears at the domain corners which are the regions of interface between Dirichlet and homogeneous Neumann boundaries. This do not represent a discontinuity, but given the problem formulation and parameters, high gradients appear in both pressure and velocity.

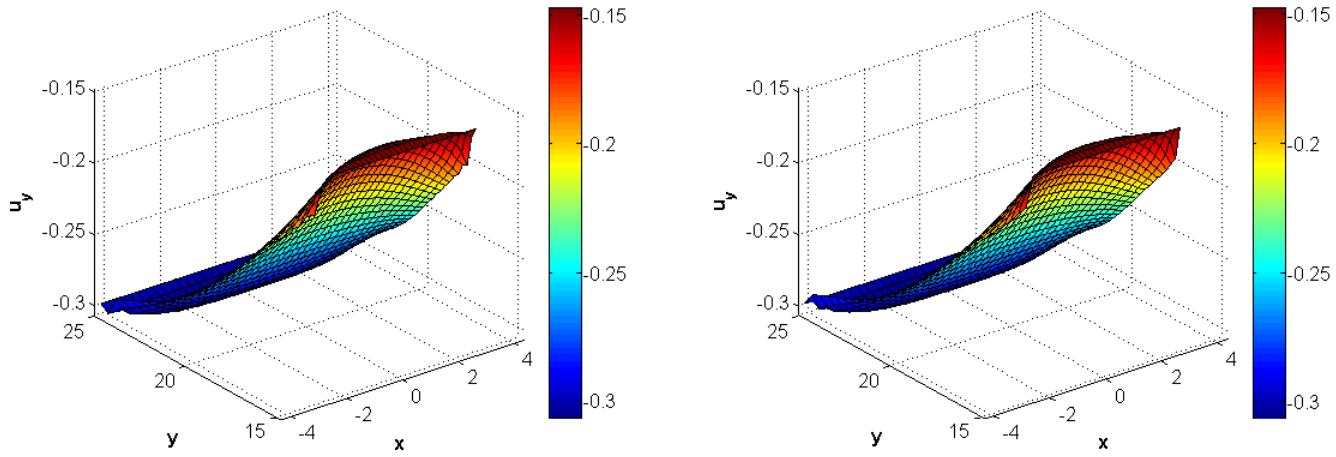


Figure 6: Velocity in y direction - (a) Standard; (b) Stabilized

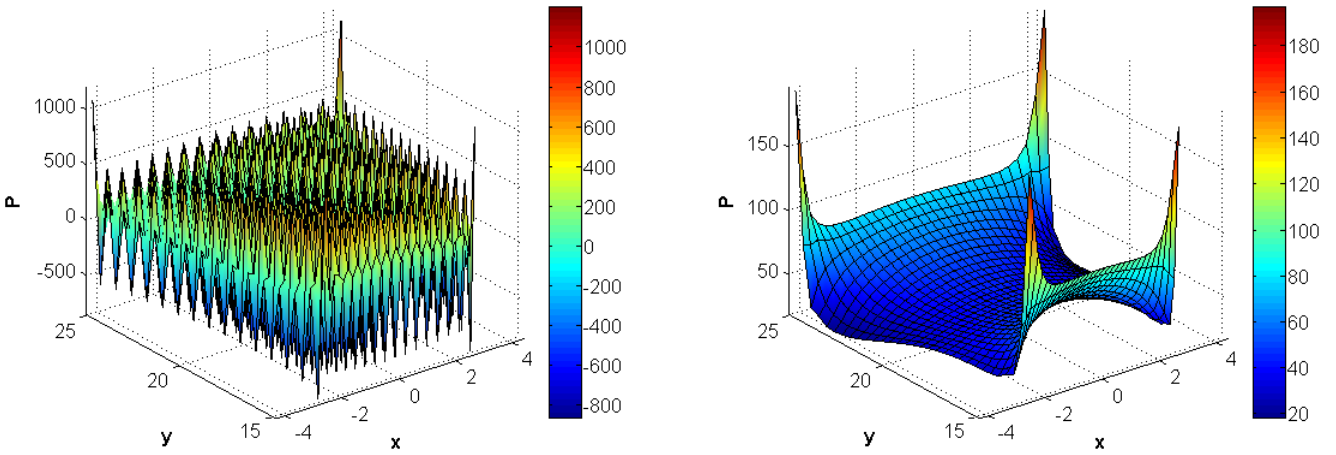


Figure 7: Pressure field - (a) Standard; (b) Stabilized

4 Problem III

The following set of partial differential equations governs the problem in the time-space domain (0,T) x Ω:

$$\nu\nabla\cdot(\nabla^S\mathbf{u})+\nabla\cdot\sigma_m(F)+\mathbf{T}_m(\mathbf{u})=0$$

$$\frac{\partial F}{\partial t}=-\mathbf{u}\cdot\nabla F+D_F\nabla^2F-\sigma_F F$$

$$\frac{\partial G}{\partial t}=D_G\nabla^2G-\sigma_G G+\hat{\sigma}_{GF}F$$

Where all constants and boundary conditions are taken from previous problems.

4.1 Weak Formulation and solution method

After applying the Galerkin Weighted residual method and integrating the higher order derivative terms by parts (neglecting boundary terms) of the stokes equation for the velocity field u, the weak form to

be solved is found. Time subscripts are present as $F = F(t)$, thus $\mathbf{u}$ is implicitly time dependent and can be evaluated at any time n given F^n.

$$\nu \mathbf{K_u u^n} - \mathbf{T_u u^n} = \mathbf{T_f} F^n \tag{14}$$

The boundary conditon constrain is imposed by lagrange multipliers and the final system of equation for $\mathbf{u^n}$ is found:

$$\begin{bmatrix} \mathbf{K_u} - \mathbf{T_u} & ADir' \\ ADir & 0 \end{bmatrix} \begin{pmatrix} \mathbf{u^n} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{T_f} F^n \\ bDir \end{pmatrix} \tag{15}$$

The time discretization of F is done by means of θ method, which can be seen as a particular case of the Padé approximations when $\mathbf{W} = \theta$ and $\mathbf{w} = 1$. Thus equation (1) can be rewritten for the time dependent convective velocity $\mathbf{u}$ case as:

$$\begin{aligned} \int_\Omega \omega \frac{F^{n+1}}{\Delta t} d\Omega + \theta \left[\int_\Omega \omega (\mathbf{u^{n+1}} \cdot \nabla) F^{n+1} d\Omega + \int_\Omega \nabla\omega \cdot (D_F \nabla F^{n+1}) d\Omega + \int_\Omega \omega \sigma_F F^{n+1} d\Omega \right] = \\ \int_\Omega \omega \frac{F^n}{\Delta t} d\Omega + (\theta - 1) \left[\int_\Omega \omega (\mathbf{u^n} \cdot \nabla) F^n d\Omega + \int_\Omega \nabla\omega \cdot (D_F \nabla F^n) d\Omega + \int_\Omega \omega \sigma_F F^n d\Omega \right] \end{aligned} \tag{16}$$

The weak form for G is similar to the one found in equation (2) and reads as follows:

$$\begin{aligned} \int_\Omega \omega \frac{\Delta G}{\Delta t} d\Omega + \theta \left[\int_\Omega \nabla\omega \cdot (D_G \nabla \Delta G) d\Omega + \int_\Omega \omega \sigma_G \Delta G d\Omega \right] = \\ - \left[\int_\Omega \nabla\omega \cdot (D_G \nabla G^n) d\Omega + \int_\Omega \omega \sigma_G G^n d\Omega \right] + \int_\Omega \omega \hat{\sigma_{GF}} \left(F^n + \theta \Delta F \right) d\Omega \end{aligned} \tag{17}$$

After interpolating ω, F and G with chosen shape functions and integrating in the appropriate gauss points, the following system of linear equations, which will be solved in each time-step, is found from equations (16) and (17).

$$\left[\mathbf{M} + \Delta t \theta \left(\mathbf{C}(\mathbf{u^{n+1}}) + D_F \mathbf{K} + \sigma_F \mathbf{M} \right) \right] F^{n+1} = \left[M + (\theta - 1) \Delta t \left(\mathbf{C}(\mathbf{u^n}) + D_F \mathbf{K} + \sigma_F \mathbf{M} \right) \right] F^n \tag{18}$$

$$\left[\mathbf{M} + \Delta t \theta \left(D_G \mathbf{K} + \sigma_G \mathbf{M} \right) \right] \Delta \mathbf{G} = -\Delta t \left(D_G \mathbf{K} + \sigma_G \mathbf{M} \right) G^n + \Delta t \hat{\sigma_{GF}} \left(\mathbf{M} F^n + \theta \mathbf{M} \Delta F \right) \tag{19}$$

Here, solution for F^{n+1} is not directly computed if an implicit method ($\theta \neq 0$) is chosen to integrate equation (18) as u^{n+1} is needed to compute the convective matrix $\mathbf{C}(\mathbf{u^{n+1}})$ at the l.h.s. of the equation. Thus, in such a case, a iterative method needs to take place involving equations (18) and 14) to find the variables $\mathbf{u}$ and F ant $t = n + 1$.

Solution for G follows the same procedure as used in Problem 1, section 2.1, no matter the parameter θ used for time integration. Lagrange multipliers are also employed to solve equations (18) and (19) at each time-step as described in section 2.1.

4.1.1 General Procedure: Iterative Method to solve the coupling u/F

As explained previously, a iterate process is necessary when implicit time integration method is employed. This iterative algorithm was implemented as follows:

BEGIN

. **FOR** {n = 1 to N° timesteps}

- $k = 0$;
- $\mathbf{Au^n} = \mathbf{T_f} F^n$; % Computation of $\mathbf{u^n}$ from F^n
- $\mathbf{C}(\mathbf{u^n})$; % Computation of $\mathbf{C^n}$;
- $\mathbf{u_k^{n+1}} = \mathbf{u^n}$; % First approximation for $\mathbf{u^{n+1}}$;
- $\mathbf{C}(\mathbf{u_k^{n+1}})$; % First approximation for $\mathbf{C^{n+1}}$;
- $[\mathbf{B} + \mathbf{C}(\mathbf{u_k^{n+1}})] F_k^{n+1} = [\mathbf{B} + \mathbf{C}(\mathbf{u^n})] F^n$; % First approximation for F^{n+1}

. **WHILE** {Criteria not met}

- $\mathbf{Au_{k+1}^{n+1}} = \mathbf{T_f} F_k^{n+1}$; % k+1 th approximation for $\mathbf{u^{n+1}}$;
- $\mathbf{C}(\mathbf{u_{k+1}^{n+1}})$; % k+1 th approximation for $\mathbf{C}(\mathbf{u^{n+1}})$
- $[\mathbf{B} + \mathbf{C}(\mathbf{u_{k+1}^{n+1}})] F_{k+1}^{n+1} = [\mathbf{B} + \mathbf{C}(\mathbf{u^n})] F^n$; % k+1 th approximation for F^{n+1}
- $\Delta F^{n+1} = F_{k+1}^{n+1} - F_k^{n+1}$; % Check convergence of F^{n+1}
- $\Delta \mathbf{u^{n+1}} = \mathbf{u_{k+1}^{n+1}} - \mathbf{u_k^{n+1}}$; % Check convergence of $\mathbf{u^{n+1}}$
- $k = k + 1$;

. **END**

- $\mathbf{u^{n+1}} = \mathbf{u_{k+1}^{n+1}}$; % Solution for $\mathbf{u^{n+1}}$
- $F^{n+1} = F_{k+1}^{n+1}$; % Solution for F^{n+1}

. **END**
END

4.1.2 Simplified Procedure to solve the coupling u/F

As seen previously for each time step, a new $\mathbf{u}$ needs to be computed at t = n + 1. However, the $\mathbf{T_f}$ matrix, which relates $\mathbf{u}$ with F, only operates in the boundary terms of F. More precisely, it operates in the terms correspondent to the upper boundary where F is constant, specified by a Dirichlet boundary condition. Consequently, the term $T_f F$ will be constant after the first computation of it using a initial condition for $F = F^1$ that fulfills the Boundary condition. As a result $\mathbf{u^1}$ is found and $\mathbf{u}$ will also be constant from this point throughout the time steps as the r.h.s. term $T_f F^n$ is time-invariant in this specific problem.

From this conclusion, the iterative process can be dropped and $\mathbf{u}$ and $C(u)$ can be both computed only once, outside the time-step loop. The algorithm reduces to the following

BEGIN
- $\mathbf{Au^1} = \mathbf{T_f} F^1$; % Computation of $\mathbf{u^1}$ from F^1
- $\mathbf{C}(\mathbf{u^1})$; % Computation of $\mathbf{C} = \mathbf{C}(\mathbf{u^1})$;

. **FOR** {n = 1 to N° timesteps}

- $[\mathbf{B} + \mathbf{C}] \mathbf{F^{n+1}} = [\mathbf{B} + \mathbf{C}] F^n$; % Solution for F^{n+1}

. **END**

END

This simplified algorithm to solve F is similar to the one used in Problem 1, where the FEM matrices are computed before the time-stepping loop. The only difference relies on the velocity field ($\mathbf{u}$) applied.

4.2 Results

As explained previously, given the form of the coupling $\mathbf{u}/F$, the convection velocity field will be constant over time, thus both algorithms, direct and iterative will present same results. Thus, no distinction is made from here. The problem was solved using a 30x30 Q1Q1 mesh. Figure (1) shows the convective velocity vector field, which was evaluated using the inital condition for F.

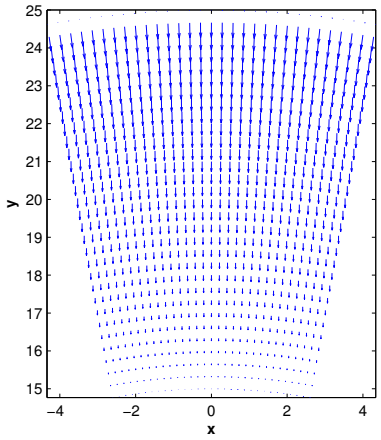
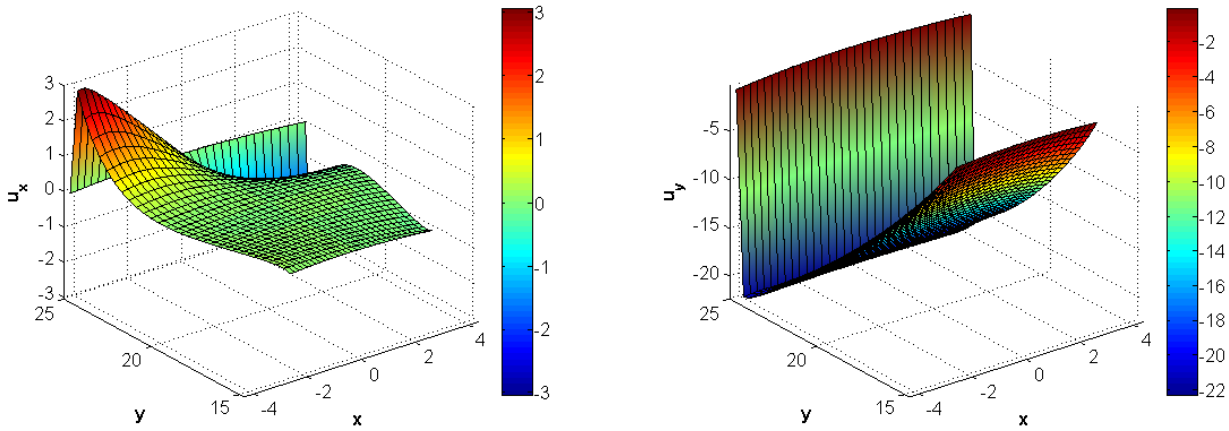


Figure 8: Convective velocity vector field.

From the velocity components surface, shown in Figure (9), it can be observed a high gradient for the component in y direction. It means that the force provided by F in $\mathbf{u}$ at the boundary is relatively high. An reduction in the model coefficients, specially a reduction in the coefficient of the matrix $\mathbf{T_f}$ which couples $\mathbf{u}$ and F may generate a more smooth and feasible profile.

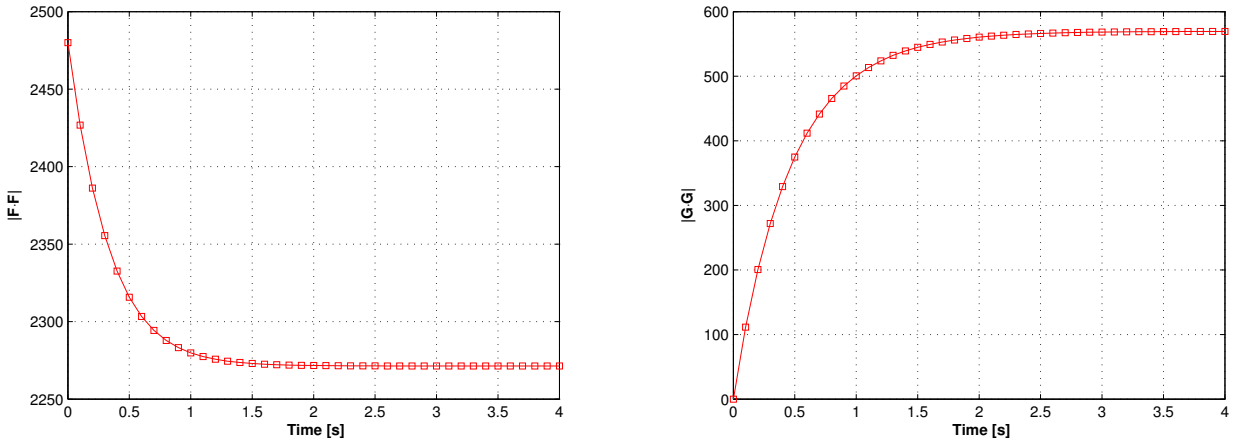


(a) (b)

Figure 9: Velocity - (a) x direction; (b) y direction.

The time integration was performed using the Crank-Nicholson method (R22 and R33 padé are also available) with time-step $\Delta t = 0.1$. In order to define the stationary solution, the norm of the FEM

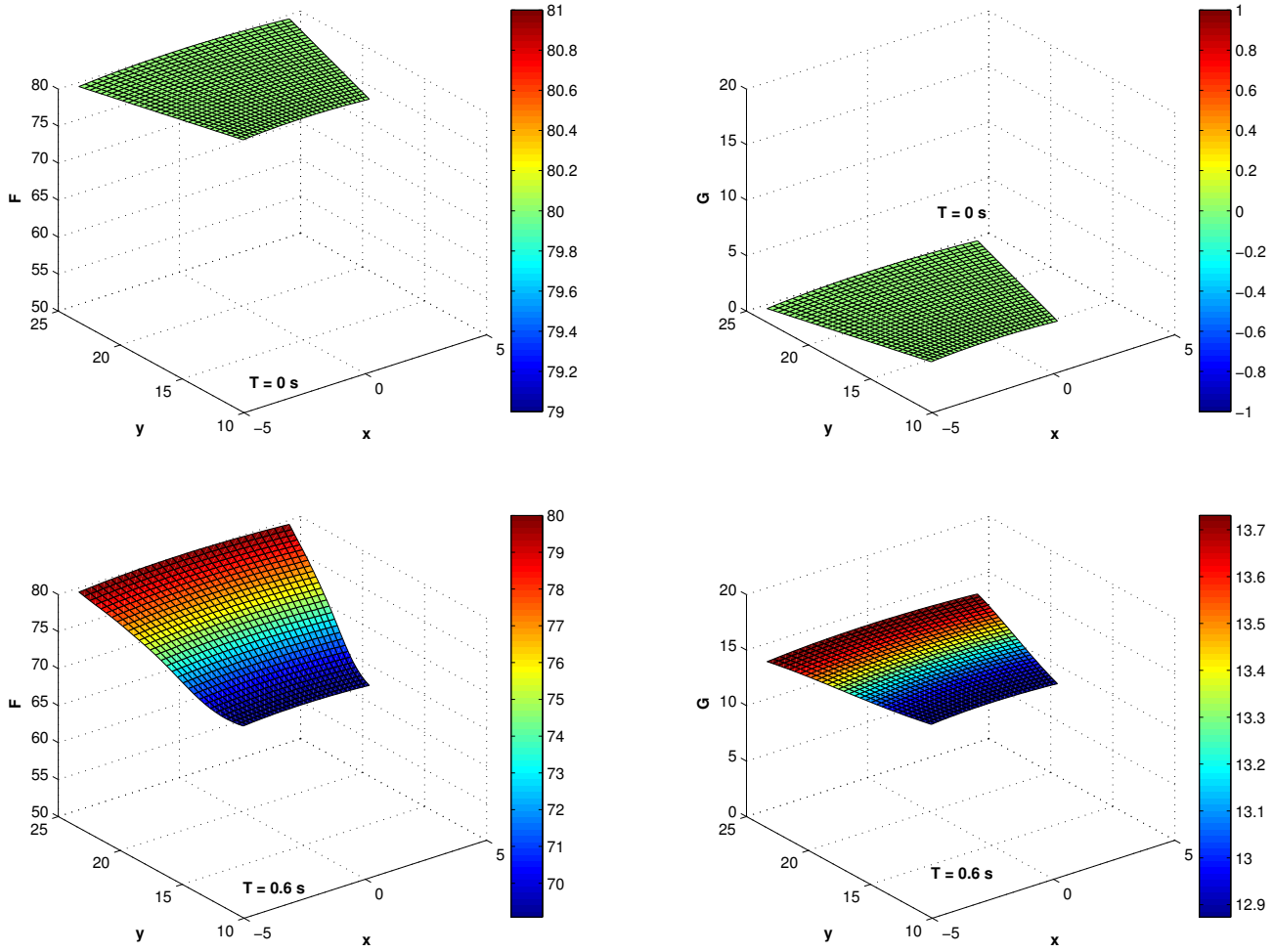
approximated solution for F and G, given by $\sqrt{F^h \cdot F^h}$ and $\sqrt{G^h \cdot G^h}$ respectively, were tracked during the integration. Figure (3) show the evolution of the norm of each approximation.

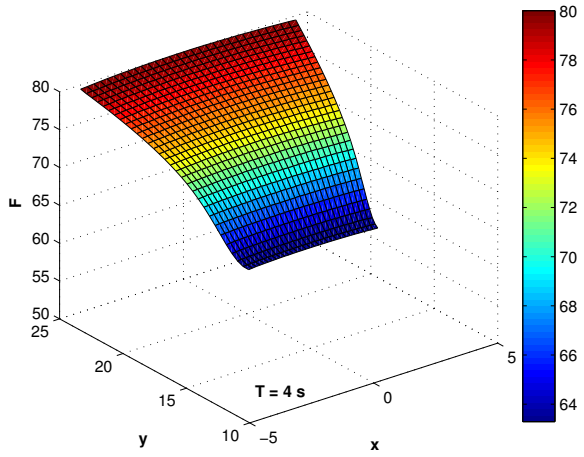


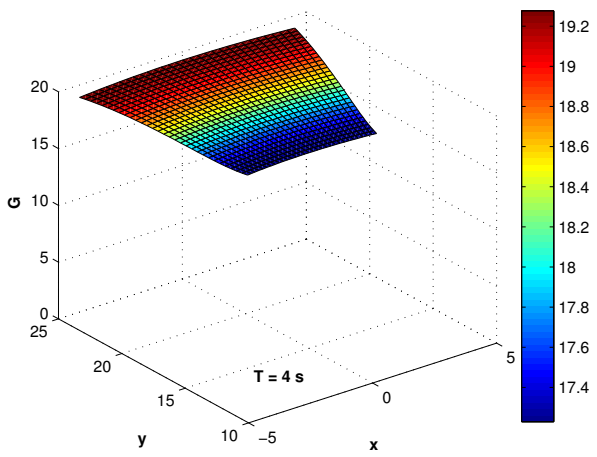
(a) (b)

Figure 10: Norm of approximated solution - (a) F; (b) G.

The evolution of F and G until stationary state is show in next figures. Initial condition for both F and G were chosen as uniform, fulfilling the boundary conditions.







It can be noticed that for time around 2 s, stationary solution is reached by both variables, as demonstred in Figure (10). The stationary solution is reached faster than the one found in the Problem I, this has to do with the high convective velocity imposed in this problem, as seen previously. This way F decreases asymptotically while G do the opposite without presenting any oscillation of overshoot as seen in Problem I. The time evolution of F and G, from initial condition until stationary state at T = 2 s is show in next figures. Initial condition for both F and G were chosen as uniform, fulfilling the boundary conditions.

5 Annex - MatLab Routines

5.1 Problem I

Main routine

```
clear, close all, home
clc
format long
disp(' ')
disp('This program solves a coupled transient convection-diffusion problem:')
disp('Ft + a GRAD F - NuF DIV(GRAD)F + SigF*F = 0')
disp('Gt - NuG DIV(GRAD)G + SigG*G = c*F')
disp('on a circular sector domain  r[10, 25] x theta[-10°, 10°].')
disp(' ')
disp('Velocity field is -k*(y,x)/r^2')
% ---------------
% COEFFICIENTS
% ---------------
% F
PR.NuF = 5;%5;   %[m^2*s^-1]
PR.SigF = 0.25;%0.25;  %[s^-1]
% G
PR.NuG = 15; %15 %[m^2*s^-1]
PR.SigG = 2;  %2   %[s^-1]
% COUPLING CONSTANT
PR.c = 0.5;      %[s^-1]
%-----------------------------
% MESH GENERATION
%-----------------------------
nx = 30;
ny = 30;
% Matrices of nodal coordinates and connectivities
elem = 0;
[X,T] = createMesh(ny,nx);
%[X T] = CreateMesh_Q(-5,5,15,25,nx,ny)
% figure(1)
plotMesh(T,X,elem)
numnp = size(X,1);
% -----------------------------
% CONVECTION VELOCITY FIELD
% -----------------------------
Conv = zeros(size(X));
R2 = X(:,1).^2 + X(:,2).^2;
R = sqrt(R2);
k =1/1500;
Conv(:,1) = -k*R.*X(:,1);
Conv(:,2) = -k*R.*X(:,2);
figure(15)
quiver(X(:,1),X(:,2),Conv(:,1),Conv(:,2))
xlabel('\bfx')
ylabel('\bfy')
%-------------------------------------------------
% BOUNDARY CONDITIONS  (LAGRANGE MULTIPLIERS METHOD)
%-------------------------------------------------
%F
BCF = BC_F(nx,ny,numnp);
%G
BCG = BC_G(nx,ny,numnp);
%--------------------
% INITIAL CONDITION
%--------------------
% F EQUATION
Fb = 80;
in = 1;
Rmax = 25;
Ro = 15;
if in == 1
```

```
% UNIFORM (compatible with Dirichlet BC)
cF = Fb*ones(numnp,1);
elseif in == 2
% LINEAR  (compatible with Dirichlet BC)
cF = Fb*((R-Ro)/(Rmax-Ro));
%cF = Fb*((X(:,2)-Ro)/(Rmax-Ro));
elseif in == 3 %Discontinuous
%cF = Fb*((R > 24)+(R == 25));
end
% G EQUATION
cG = zeros(numnp,1);
% -------------------------
% TIME INTEGRATION METHOD
% -------------------------
disp(' ')
disp('There are three integration schemes available ')
disp('       [0] = Crank-Nicolson');
disp('       [1] = R22');
disp('       [2] = R33');
d_temp = input('Choose a method to perform time integration = ');
% Matrices for the time integration
if d_temp == 0
    method = 'CN + ';
    W = 1/2;
    w = 1;
elseif d_temp == 1
    method = 'R22 + ';
    W = (1/24)*[7 -1; 13 5];
    w = [1/2; 1/2];
elseif d_temp == 2
    method = 'R33 + ';
    r5 = sqrt(5);
    W = [49-13*r5       12*(2-r5)       r5-1
         26*r5          12*r5           -2*r5
         61-13*r5       36              11+r5]/120;
    w = [12*(5-r5); 24*r5; 12*(5-r5)]/120;
else
    error('Unavailable time integration scheme')
end
%---------------------------
% INTEGRATION PARAMETERS
%---------------------------
dt = 0.1;
endt = 15;
nstep = ceil(endt/dt);
%-----------
% SOLUTION
%-----------
[SolF, SolG] = Galerkin1(X,T,Conv,PR,BCF,BCG,cF,cG,W,w,dt,nstep,numnp);
```

Subroutine: BC_F (Boundary Condition F)

```
function BCF =  BC_F(nx,ny,nunk)
% [Accd, bccd] =  BC(nx,ny,nunk)
% This function creates matrices Accd and bccd to impose homogeneous
% Dirichlet boudary conditions using Lagrange multipliers method.
%
%   nx,ny:  number of elements on each direction
%   nunk:   number of degrees of freedom for the velocity field
%


% Nodes on the Dirichlet boundary
nodesD = [(ny+1)*(nx+1) : -1 : ny*(nx+1)+1]';     % r = 25
% Imposed boundary conditions

C = [nodesD, zeros(size(nodesD))];

% Boundary conditions' matrix
```

```
nDir = size(C,1);
Accd = zeros(nDir,nunk);
Accd(:,C(:,1)) = eye(nDir);
% Boundary conditions' vector
bccd = C(:,2);
BCF.Accd = Accd;
BCF.bccd = bccd;
```

Subroutine: BC_G (Boundary Condition G)

```
function BCG =  BC_G(nx,ny,nunk)
% [Accd, bccd] =  BC(nx,ny,nunk)
% This function creates matrices Accd and bccd to impose homogeneous
% Dirichlet boudary conditions using Lagrange multipliers method.
%
%   nx,ny:  number of elements on each direction
%   nunk:   number of degrees of freedom for the velocity field
Accd = [];
% Boundary conditions' vector
bccd = [];

BCG.Accd = Accd;
BCG.bccd = bccd;
```

Subroutine: Galerkin1 (Time integration)

```
function [SolF , SolG] = Galerkin1(X,T,Conv,PR,BCF,BCG,cF,cG,W,w,dt,nstep,npoin)
% Sol = Galerkin(X,IEN,Conv,nu,f,c,Accd1,bccd1, T,s,beta,dt,nstep)
% This function computes solution for a transient convection-diffusion equation
% at different time instants.
% Galerkin method is used to perform spatial discretization.
%
% Input:
%   X: nodal coordinates
%   IEN: connectivities
%   Conv: velocity field
%   nu: diffusion
%   f: source term
%   c: initial condition
%   Accd1, bccd1: matrices to impose boundary conditions using Lagrange multipliers
%   T,s,beta: matrices for the time-integration scheme
%   dt: time-step
%   nstep: number of time steps
%
% F PARAMENTERS
Nu = PR.NuF;
Sig = PR.SigF;
% BC MATRICES
Accd1 = BCF.Accd;
bccd1 = BCF.bccd;
nccd = size(Accd1,1);
%-------------------------------------------------------------------------
% COMPUTATION OF MATRICES FOR F and G DISCRETIZATION
%-------------------------------------------------------------------------
% Number of Gauss points (numerical quadrature)
ngaus = 4;
% Quadrature
[pospg,wpg] = Quadrature(ngaus);
% Shape Functions
[N,Nxi,Neta] = ShapeFunc(pospg);
% Matrices obtained by discretizing the Galerkin weak-form
M = CreMassMat(X,T,pospg,wpg,N,Nxi,Neta);
C = CreConvMat(X,T,Conv,pospg,wpg,N,Nxi,Neta);
K = CreStiffMat(X,T,pospg,wpg,N,Nxi,Neta);
% Integration matrix
[n,m] = size(W);
Id = eye(n,m);
%------------------------------
```

```
% LEFT HAND SIDE SYSTEM FOR F
%------------------------------
% Computation of the matrix necessary to obtain solution at each time-step: A du = F
Kt  = C + Nu*K + Sig*M;  %REACTION TERM INCORPORATED
A = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*M  + dt*W(i,j)*Kt];
    end
    A = [A; row];
end

Accd = []; bccd = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*Accd1];
    end
    Accd = [Accd; row];
    bccd = [bccd; bccd1];
end
nccd = n*nccd;
Atot = [A Accd'; Accd zeros(nccd)];
Atot = sparse(Atot);
% Factorization of matrix Atot
[L,U] = lu(Atot);
L = sparse(L);
U = sparse(U);
% Initial condition
SolF = cF;
SolG = cG;
dSolF = [];
% Loop to compute the transient solution
disp(' ')
disp('Transient analysis: computation of the solution at each time step')
for i=1:nstep
    %----------------------------------
    % SOLUTION OF F
    %----------------------------------
    aux = -dt*Kt*cF;
    F = [];
    for h =1:n
         F = [F; w(h)*aux];
    end
    F = [F;bccd];
    dc  = U\(L\F);
    dSolF = [dSolF dc(1:n*npoin)];
    dc = reshape(dc(1:n*npoin),npoin,n);
    cF = cF + sum(dc,2);
    SolF = [SolF cF];
    %----------------------------------
    % SOLUTION OF G
    %----------------------------------
    [cG] = solve_G1(M,K,PR,cG,SolF,dSolF,BCG,i,dt,W,w,npoin);
    SolG = [SolG cG];
end
```

Subroutine: Solve_G1 (time integration of G)

```
function [cG] = solve_G1(M,K,PR,cG,f1,f2,BCG,i,dt,W,w,npoin)
% G PARAMENTERS
Nu = PR.NuG;
Sig = PR.SigG;
k = PR.c;
% BC MATRICES
Accd1 = BCG.Accd;
bccd1 = BCG.bccd;
nccd = size(Accd1,1);
```

```
%-----------------------------
% LEFT HAND SIDE SYSTEM FOR G
%-----------------------------
% Integration matrix
[n,m] = size(W);
Id = eye(n,m);
% Computation of the matrix necessary to obtain solution at each time-step: A du = F
Kt  =  Nu*K + Sig*M;  %REACTION TERM INCORPORATED and NO C MATRIX
A = [];
B = [];
for l = 1:n
    row = [];
    col = [];
    for j = 1:m
        row = [row, Id(l,j)*M  + dt*W(l,j)*Kt];
        col = [col,  k*dt*W(l,j)*M];           %FORCE TERM FROM F
    end
    A = [A; row];
    B = [B;col];
end
Accd = []; bccd = [];
for l = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(l,j)*Accd1];
    end
    Accd = [Accd; row];
    bccd = [bccd; bccd1];
end
nccd = n*nccd;
Atot = [A Accd'; Accd zeros(nccd)];
Atot = sparse(Atot);
% Factorization of matrix Atot
[L,U] = lu(Atot);
L = sparse(L);
U = sparse(U);
% Source term
Mf = k*M*f1; %%FORCE TERM FROM F
% Initial condition
aux = dt*(-Kt*cG + Mf(:,i));
F = [];
for h = 1:n
    F = [F; w(h)*aux];
end
F = F + B*f2(:,i);
F = [F;bccd];
dc  = U\(L\F);
dc = reshape(dc(1:n*npoin),npoin,n);
cG = cG + sum(dc,2);
```

5.2 Problem II

Main routine

```
clear; close all; clc
addpath('Func_ReferenceElement')

dom = [-4.5,4.5,14.5,25];
Rmax = 25;
Rmin = 15;
mu = 10^3;
% Element type and interpolation degree
% (0: quadrilaterals, 1: triangles, 11: triangles with bubble function)
elemV = 0;
degreeV = 1;
degreeP = 1;
elemP = elemV;
referenceElement = SetReferenceElementStokes(elemV,degreeV,elemP,degreeP);
nx = 30;
ny = 30;
[X,T] = createMesh(ny,nx);
XP = X;
TP = T;
figure; PlotMesh(T,X,elemV,'b-');
figure; PlotMesh(TP,XP,elemP,'r-');
%%
% Matrices arising from the discretization
met = 2;   % Galerkin: met = 1; Estbilized: met = 2
if met == 1
[K,G,f] = StokesSystem2(X,T,XP,TP,referenceElement);
[ndofP,ndofV] = size(G);
K = mu*K;
A = zeros(ndofP,ndofP);
h = zeros(ndofP,1);
elseif met == 2
[K,G,f,A,h] = StokesSystem_GLS2(X,T,XP,TP,referenceElement);
[ndofP,ndofV] = size(G);
dx = (Rmax - Rmin)/ny;
tau = (1/3)*dx^2/(4*mu);
K = mu*K;
A = tau*A;
h = tau*h;
end
% Matrix and r.h.s vector to impose Dirichlet boundary conditions using
% Lagrange multipliers
[A_DirBC, b_DirBC, nDir, confined] = BC_m(X,nx,ny,ndofV);
%%
% Total system of equations
if confined
   nunkP = ndofP-1;
   disp(' ')
   disp('Confined flow. Pressure on lower left corner is set to zero');
   G(1,:) = [];
else
   nunkP = ndofP;
end
Atot = [K          A_DirBC'             G'
        A_DirBC    zeros(nDir,nDir)     zeros(nDir,nunkP)
        G          zeros(nunkP,nDir)    A(1:nunkP,1:nunkP)];
btot = [f ; b_DirBC ; h(1:nunkP,1)];

sol = Atot\btot;
```

Subroutine: StokesSystem_GLS2 (GLS formulation Cauchy tensor)

```
 function [K,G,f,A,h] = StokesSystem_GLS2(X,T,XP,TP,referenceElement)
% [K,G,f] = StokesSystem2(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;
NPxi = referenceElement.NPxi;
NPeta = referenceElement.NPeta;
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);
A = zeros(ndofP,ndofP);
f = zeros(ndofV,1);
h = 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),:);
    Xp = XP(TPe(1:nenP),:);
    % Degrees of freedom in element ielem
    Te_dof = reshape([2*Te-1; 2*Te],1,nedofV);
    TPe_dof = TPe;
    % Element matrices
    [Ke,Ge,fe,Ae,he] =
EleMatStokes2(Xe,Xp,ngeom,nedofV,nedofP,ngaus,wgp,N,Nxi,Neta,NP,NPxi,NPeta);
    % 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;
    A(TPe_dof,TPe_dof) =  A(TPe_dof,TPe_dof) + Ae;
    f(Te_dof) = f(Te_dof) + fe;
    h(TPe_dof) = h(TPe_dof) + he;
end

function [Ke,Ge,fe,Ae,he] =
EleMatStokes2(Xe,Xp,ngeom,nedofV,nedofP,ngaus,wgp,N,Nxi,Neta,NP,NPxi,NPeta)
Ke = zeros(nedofV,nedofV);
Ge = zeros(nedofP,nedofV);
Ae = zeros(nedofP,nedofP);
fe = zeros(nedofV,1);
he = zeros(nedofP,1);
B = zeros(3,nedofV);
% Loop on Gauss points
```

```
for ig = 1:ngaus
    N_ig    = N(ig,:);
    Nxi_ig  = Nxi(ig,:);
    Neta_ig = Neta(ig,:);
    NP_ig = NP(ig,:);
    NPxi_ig = NPxi(ig,:);
    NPeta_ig = NPeta(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))
        ];
    JacobP =[
        NPxi_ig(1:nedofP)*(Xp(:,1)) NPxi_ig(1:nedofP)*(Xp(:,2))
        NPeta_ig(1:nedofP)*(Xp(:,1))    NPeta_ig(1:nedofP)*(Xp(:,2))
        ];
    dvolu = wgp(ig)*det(Jacob);
    res = Jacob\[Nxi_ig;Neta_ig];
    nx = res(1,:);
    ny = res(2,:);
    resP = JacobP\[NPxi_ig;NPeta_ig];
    Nx_P = resP(1,:);
    Ny_P = resP(2,:);
    Ngp = [reshape([1;0]*N_ig,1,nedofV); reshape([0;1]*N_ig,1,nedofV)];
    dN = reshape(res,1,nedofV);
    C = diag([2,2,1]);
    B(1,1:2:end) = nx;
    B(2,2:2:end) = ny;
    B(3,1:2:end) = ny; B(3,2:2:end) = nx;
    Ke = Ke + B'*C*B*dvolu;
    Ge = Ge - NP_ig'*dN*dvolu;
    Ae = Ae - (Nx_P'*Nx_P + Ny_P'*Ny_P)*dvolu;
    x_ig = N_ig(1:ngeom)*Xe;
    f_igaus = SourceTerm(x_ig);
    fe = fe + Ngp'*f_igaus*dvolu;
    he = he - resP'*f_igaus*dvolu;
end
```

Subroutine: BC_m (Boundary Condition)

```
function [A, b, nDir, confined] = BC_m(X,nx,ny,n)
% [A, b, nDir, confined] = BC(X,dom,n)
% Matrices to impose Dirichlet boundary conditions using Lagrange
% multipliers on a rectangular domain
% Input:
%    X: nodal coordinates
%    dom: domain description [x1,x2,y1,y2]
%    n: number of velocity degrees of freedom
% Output:
%    A,b: matrix and r.h.s. vector to impose the boundary conditions using
%         Lagrange multipliers
%    nDir: number of prescribed degrees of freedom
%    confined:
nodesLower = [1 : nx+1]';
nodesUpper = [(nx+1)*ny+1 : 1 : (ny+1)*(nx+1)]';
nodesDirBC = [nodesLower; nodesUpper];
% confined flow (velocity is imposed for all the nodes in the boundary)
confined = 0; %NO CONFINED FLOW
% number of prescribed degrees of freedom
nDir = 2*length(nodesDirBC);

% Degrees of freedom where the velocity is prescribed
% (horizontal and vertical component)
C = [2*nodesDirBC - 1; 2*nodesDirBC];
A = zeros(nDir,n);
A(:,C) = eye(nDir);
Th1 = atan2(X(nodesLower,2),X(nodesLower,1));
Th2 = atan2(X(nodesUpper,2),X(nodesUpper,1));
% Imposed value (velocity).
% at r = 15 (or Y1) --> Ur = -0.15 and Utheta = 0
```

```
% at r = 25 (or Y2) --> Ur = -0.30 and Utheta = 0
Ulower = -0.15*cos(Th1);
Vlower = -0.15*sin(Th1);
Uupper = -0.3*cos(Th2);
Vupper = -0.3*sin(Th2);

b = [
      Ulower; Uupper
      Vlower; Vupper
    ];
```

5.3 Problem III

Main routine

```
clear, close all, home
clc
format long
addpath('EX3_u')
addpath('EX3_u\Func_ReferenceElement')
disp(' ')
disp('This program solves a coupled transient convection-diffusion problem:')
disp('Ft + a GRAD F - NuF DIV(GRAD)F + SigF*F = 0')
disp('Gt - NuG DIV(GRAD)G + SigG*G = c*F')
disp('on a circular sector domain  r[10, 25] x theta[-10°, 10°].')
%-----------------------------
% PARAMETERS - COEFFICIENTS
%-----------------------------
%F
PR.NuF = 5;
PR.SigF = 0.25;
%G
PR.NuG = 15;
PR.SigG = 2;
% COUPLING CONSTANT
PR.c = 0.5;
% VISCOSITY (u)
PR.NuU = 1000;
%-----------------------------
% MESH GENERATION
%-----------------------------
nx = 30;
ny = 30;
% Matrices of nodal coordinates and connectivities
elem = 0;
[X,T] = createMesh(ny,nx);
numnp = size(X,1);
plotMesh(T,X,elem)
%-----------------------------------------------------
% BOUNDARY CONDITIONS  (LAGRANGE MULTIPLIERS METHOD)
%-----------------------------------------------------
% F
BCF = BC_F(nx,ny,numnp);
% G
BCG = BC_G(nx,ny,numnp);
% U
BCu = BC_m(X,nx,ny,2*numnp);
%--------------------
% INITIAL CONDITION
%--------------------
% F (EQUATION)
Fb = 80;
Rmax = 25;
Ro = 15;
R2 = X(:,1).^2 + X(:,2).^2;
R = sqrt(R2);
in = 1;
if in == 1
% UNIFORM (compatible with Dirichlet BC)
cF = Fb*ones(numnp,1);
elseif in == 2
% LINEAR  (compatible with Dirichlet BC)
cF = Fb*((R-Ro)/(Rmax-Ro));
%cF = Fb*((X(:,2)-Ro)/(Rmax-Ro));
elseif in == 3 %Discontinuous
%cF = Fb*((R > 24)+(R == 25));
end
% G (EQUATION)
cG = zeros(numnp,1);
%---------------------------
```

```
% INTEGRATION PARAMETERS
%--------------------------
dt = 0.1;
endt = 4;
nstep = ceil(endt/dt);
%-----------------------------------------------
% SOLUTION (TIME INTEGRATION BY CRANK-NICHOLSON)
%-----------------------------------------------
method = 'CN + ';
W = 1/2;
w = 1;
% Solution
s = 1;
if s == 1
% DIRECT
[SolF , SolG] = Galerkin1_3(X,T,PR,BCu,BCF,BCG,cF,cG,W,w,dt,nstep,numnp);
elseif s == 2
% ITERATIVE METHOD
BCF = BC_F2(nx,ny,numnp); % NEW DEFINITION OF BC (SOLVING FOR F NOT DF)
[SolF , SolG] = Galerkin1_3IT(X,T,PR,BCu,BCF,BCG,cF,cG,W,w,dt,nstep,numnp);
end
```

Subroutine: BC_m (Boundary Condition U)

```
function BC = BC_m(X,nx,ny,n)
% [A, b, nDir, confined] = BC(X,dom,n)
% Matrices to impose Dirichlet boundary conditions using Lagrange
% multipliers on a rectangular domain
% Input:
%    X: nodal coordinates
%    dom: domain description [x1,x2,y1,y2]
%    n: number of velocity degrees of freedom
% Output:
%    A,b: matrix and r.h.s. vector to impose the boundary conditions using
%         Lagrange multipliers
%    nDir: number of prescribed degrees of freedom
%    confined:
nodesLower = [1 : nx+1]';
nodesUpper = [(nx+1)*ny+1 : 1 : (ny+1)*(nx+1)]';
nodesDirBC = [nodesLower; nodesUpper];
% number of prescribed degrees of freedom
nDir = 2*length(nodesDirBC);

% Degrees of freedom where the velocity is prescribed
% (horizontal and vertical component)
C = [2*nodesDirBC - 1; 2*nodesDirBC];
A = zeros(nDir,n);
A(:,C) = eye(nDir);
Th1 = atan2(X(nodesLower,2),X(nodesLower,1));
Th2 = atan2(X(nodesUpper,2),X(nodesUpper,1));
% Imposed value (velocity).
% at r = 15 (or Y1) --> Ur = -0.15 and Utheta = 0
% at r = 25 (or Y2) --> Ur = -0.30 and Utheta = 0
Ulower = -0.15*cos(Th1);
Vlower = -0.15*sin(Th1);
Uupper = -0.3*cos(Th2);
Vupper = -0.3*sin(Th2);

b = [
      Ulower; Uupper
      Vlower; Vupper
    ];

BC.A = A;
BC.b = b;
BC.nDir = nDir;
```

Subroutine: BC_F (Boundary Condition F)

```
function BCF =  BC_F(nx,ny,nunk)
% [Accd, bccd] =  BC(nx,ny,nunk)
% This function creates matrices Accd and bccd to impose homogeneous
% Dirichlet boudary conditions using Lagrange multipliers method.
%
%   nx,ny:  number of elements on each direction
%   nunk:   number of degrees of freedom for the velocity field
%

% Nodes on the Dirichlet boundary
nodesD = [(ny+1)*(nx+1) : -1 : ny*(nx+1)+1]';     % r = 25
% Imposed boundary conditions

C = [nodesD, zeros(size(nodesD))];

% Boundary conditions' matrix
nDir = size(C,1);
Accd = zeros(nDir,nunk);
Accd(:,C(:,1)) = eye(nDir);
% Boundary conditions' vector
bccd = C(:,2);
BCF.Accd = Accd;
BCF.bccd = bccd;
```

Subroutine: BC_G (Boundary Condition G)

```
function BCG =  BC_G(nx,ny,nunk)
% [Accd, bccd] =  BC(nx,ny,nunk)
% This function creates matrices Accd and bccd to impose homogeneous
% Dirichlet boudary conditions using Lagrange multipliers method.
%
%   nx,ny:  number of elements on each direction
%   nunk:   number of degrees of freedom for the velocity field
Accd = [];
% Boundary conditions' vector
bccd = [];

BCG.Accd = Accd;
BCG.bccd = bccd;
```

Subroutine: Galerkin1_3 (Time integration - Direct)

```
function [SolF , SolG] = Galerkin1_3(X,T,PR,BCu,BCF,BCG,cF,cG,W,w,dt,nstep,npoin)
% Sol = Galerkin(X,IEN,Conv,nu,f,c,Accd1,bccd1, T,s,beta,dt,nstep)
% This function computes solution for a transient convection-diffusion equation
% at different time instants.
% Galerkin method is used to perform spatial discretization.
%
% Input:
%   X: nodal coordinates
%   T: connectivities
%   Conv: velocity field
%   nu: diffusion
%   f: source term
%   c: initial condition
%   Accd1, bccd1: matrices to impose boundary conditions using Lagrange multipliers
%   W,w,beta: matrices for the time-integration scheme
%   dt: time-step
%   nstep: number of time steps
% F PARAMENTERS
Nu = PR.NuF;
Sig = PR.SigF;
% VISCOSITY (u)
NuU = PR.NuU;
% BC MATRICES
```

```
Accd = BCF.Accd;
bccd = BCF.bccd;
nccd = size(Accd,1);
%-------------------------------------------------------------------------
% COMPUTATION OF MATRICES FOR VELOCITY FIELD (u)DISCRETIZATION
%-------------------------------------------------------------------------
elemV = 0; degreeV = 1;
RefE = SetReferenceElementStokes_3(elemV,degreeV);
[Ku,fu] = StokesSystem_3(X,T,RefE);
[Tf,Tu] = boundaryMatrices(X,T,RefE);
%-------------------------------------------------------------------------
% COMPUTATION OF MATRICES FOR F and G DISCRETIZATION
%-------------------------------------------------------------------------
N = RefE.N;
Nxi = RefE.Nxi;
Neta = RefE.Neta;
pospg = RefE.GaussPoints;
wpg = RefE.GaussWeights;
% Matrices obtained by discretizing the Galerkin weak-form
M = CreMassMat(X,T,pospg,wpg,N,Nxi,Neta);
K = CreStiffMat(X,T,pospg,wpg,N,Nxi,Neta);
%------------------------------
% CONSTANT MATRICES FOR F
%------------------------------
K1  = Nu*K + Sig*M;  %REACTION TERM INCORPORATED
% Initial condition
SolF = cF;
SolG = cG;
dSolF = [];
%Conv = solve_U(cF,Ku,fu,Tf,Tu,NuU,npoin,BCu);
% Loop to compute the transient solution
%WG = 0;
    %---------------------------------
    % SOLUTION OF U
    %---------------------------------
    Conv = solve_U(cF,Ku,fu,Tf,Tu,NuU,npoin,BCu);
    C = CreConvMat(X,T,Conv,pospg,wpg,N,Nxi,Neta);
    Kt = K1 + C;
for i=1:nstep
    %---------------------------------
    % SOLUTION OF F
    %---------------------------------
    A = M + dt*W*Kt;
    Atot = [A Accd'; Accd zeros(nccd)];
    Atot = sparse(Atot);
    % Factorization of matrix Atot
    [L,U] = lu(Atot);
    L = sparse(L);
    U = sparse(U);
    F = -dt*w*Kt*cF ;
    F = [F;bccd];
    dc  = U\(L\F);
    dc = dc(1:npoin);
    dSolF = [dSolF dc];
    cF = cF + dc;
    SolF = [SolF cF];
    %---------------------------------
    % SOLUTION OF G
    %---------------------------------
    [cG] = solve_G(M,K,PR,cG,SolF,dSolF,BCG,i,dt,W,w,npoin);
    SolG = [SolG cG];
end
```

Subroutine: Galerkin1_3IT (Time integration - Iterative)

```
function [SolF , SolG] = Galerkin1_3IT(X,T,PR,BCu,BCF,BCG,cF,cG,W,w,dt,nstep,npoin)
% Sol = Galerkin(X,IEN,Conv,nu,f,c,Accd1,bccd1, T,s,beta,dt,nstep)
% This function computes solution for a transient convection-diffusion equation
% at different time instants.
```

```
% Galerkin method is used to perform spatial discretization.
%
% Input:
%   X: nodal coordinates
%   T: connectivities
%   Conv: velocity field
%   nu: diffusion
%   f: source term
%   c: initial condition
%   Accd1, bccd1: matrices to impose boundary conditions using Lagrange multipliers
%   W,w,beta: matrices for the time-integration scheme
%   dt: time-step
%   nstep: number of time steps
% F PARAMENTERS
Nu = PR.NuF;
Sig = PR.SigF;
% VISCOSITY (u)
NuU = PR.NuU;
% BC MATRICES
Accd = BCF.Accd;
bccd = BCF.bccd;
nccd = size(Accd,1);
%-------------------------------------------------------------------------
% COMPUTATION OF MATRICES FOR VELOCITY FIELD (u)DISCRETIZATION
%-------------------------------------------------------------------------
elemV = 0; degreeV = 1;
RefE = SetReferenceElementStokes_3(elemV,degreeV);
[Ku,fu] = StokesSystem_3(X,T,RefE);
[Tf,Tu] = boundaryMatrices(X,T,RefE);
%-------------------------------------------------------------------------
% COMPUTATION OF MATRICES FOR F and G DISCRETIZATION
%-------------------------------------------------------------------------
N = RefE.N;
Nxi = RefE.Nxi;
Neta = RefE.Neta;
pospg = RefE.GaussPoints;
wpg = RefE.GaussWeights;
% Matrices obtained by discretizing the Galerkin weak-form
M = CreMassMat(X,T,pospg,wpg,N,Nxi,Neta);
K = CreStiffMat(X,T,pospg,wpg,N,Nxi,Neta);
%------------------------------
% CONSTANT MATRICES FOR F
%------------------------------
K1  = Nu*K + Sig*M;  %REACTION TERM INCORPORATED
% Initial condition
SolF = cF;
SolG = cG;
dSolF = [];
tol = 1e-8;
% Loop to compute the transient solution
for i=1:nstep
    %----------------------------------
    % SOLUTION OF U and F (ITERATIVE)
    %----------------------------------
    Convo = solve_U(cF,Ku,fu,Tf,Tu,NuU,npoin,BCu);
    %-------------------------------------------
    % SOLUTION OF F^(n+1) (INITIAL APPROXIMATION)
    %-------------------------------------------
    C = CreConvMat(X,T,Convo,pospg,wpg,N,Nxi,Neta);
    Kt = K1 + C;     % FIRST APPROXIMATION OF L.H.S MATRICES
    Ko = Kt;         % R.H.S MATRICES (CONSTANT INSIDE ITERATIVE PROCESS)
    A = M + dt*W*Kt; % TOTAL L.H.S MATRICES
    Atot = [A Accd'; Accd zeros(nccd)];
    Atot = sparse(Atot);
    % Factorization of matrix Atot
    [L,U] = lu(Atot);
    L = sparse(L);
    U = sparse(U);
    F = (M + dt*(W-w)*Ko)*cF ; % cF is FIXED (F^n)
    F = [F;bccd];
```

```
    sol  = U\(L\F);
    cFo = sol(1:npoin);     % FIRST  APPROXIMATION FOR F^(n+1)
    it = 1;
    while it < 100
          disp(it)
    %---------------------------------
    % SOLUTION OF U
    %---------------------------------
    Conv = solve_U(cFo,Ku,fu,Tf,Tu,NuU,npoin,BCu);% k+1 APPROX. FOR U^(n+1)
    DelU = max(max(abs(Convo-Conv)))  % MAX ABSOLUT INCREMENT IN U at n+1
    Convo = Conv;                     % UPDATE U
    % ----------------------------------------
    % SOLUTION OF F^(n+1) (ith APPROXIMATION)
    %----------------------------------------
    C = CreConvMat(X,T,Convo,pospg,wpg,N,Nxi,Neta);% k+1 APPROX FOR C^(n+1)
    Kt = K1 + C;                      % UPDATED OF L.H.S MATRICES
    A = M + dt*W*Kt;                  % UPDATED TOTAL L.H.S. MATRICES
    Atot = [A Accd'; Accd zeros(nccd)];
    Atot = sparse(Atot);
    % Factorization of matrix Atot
    [L,U] = lu(Atot);
    L = sparse(L);
    U = sparse(U);
    F = (M + dt*(W-w)*Ko)*cF ; % cF is FIXED (F^n)
    F = [F;bccd];
    sol  = U\(L\F);
    cF1 = sol(1:npoin);
    DelF = max(abs(cF1-cFo)); % MAX ABS. INCREMENT F(k+1) - F(k) at t = n+1
    cFo = cF1;                % k+1 APPROXIMATION FOR F^(n+1)
    %---------------------------------
    % CONVERGENCE CHECK
    %---------------------------------
    if DelU < tol && DelF < tol
        fprintf('\nConvergence achieved in iteration number %g\n',it);
        break
    end
    it = it + 1;
    end
    dc = cFo - cF;        % INCREMENT IN THE SOLUTION AT i+1 (DF^(i+1))
    dSolF = [dSolF dc];   % VECTOR OF INCREMENTS
    cF = cFo;             % SOLUTION AT i+1 (F^(i+1))
    SolF = [SolF cF];     % VECTOR OF SOLUTIONS (F)

    %---------------------------------
    % SOLUTION OF G
    %---------------------------------
    [cG] = solve_G(M,K,PR,cG,SolF,dSolF,BCG,i,dt,W,w,npoin);
    SolG = [SolG cG];
end
```

Subroutine: Solve_G (Time integration of G)

```
function [cG] = solve_G(M,K,PR,cG,f1,f2,BCG,i,dt,W,w,npoin)
% G PARAMENTERS
Nu = PR.NuG;
Sig = PR.SigG;
k = PR.c;
% BC MATRICES
Accd = BCG.Accd;
bccd = BCG.bccd;
nccd = size(Accd,1);
%------------------------------
% LEFT HAND SIDE SYSTEM FOR G
%------------------------------
Kt  =  Nu*K + Sig*M;  %REACTION TERM INCORPORATED and NO C MATRIX
A = M + dt*W*Kt;
B = k*dt*W*M;
Atot = [A Accd'; Accd zeros(nccd)];
Atot = sparse(Atot);
```

```
% Factorization of matrix Atot
[L,U] = lu(Atot);
L = sparse(L);
U = sparse(U);
% Source term
Mf = k*M*f1; %%FORCE TERM FROM F
F = [];
F = w*dt*(-Kt*cG + Mf(:,i));
F = F + B*f2(:,i);
F = [F;bccd];
dc  = U\(L\F);
dc = dc(1:npoin);
cG = cG + dc;
```

Subroutine: Solve_U (Solve for U)

```
function  velo = solve_U(FF,K,f,Tf,Tu,NuU,nPt,BC)
A_DirBC = BC.A;
b_DirBC = BC.b;
nDir = BC.nDir;

B = NuU*K + Tu;
fa = f - Tf*FF;

%% Total System of Equations
Atot = [B         A_DirBC'
        A_DirBC    zeros(nDir,nDir)];
btot = [fa ; b_DirBC ];

sol = Atot\btot;
velo = reshape(sol(1:2*nPt), 2, [])';
```