Finite Elements in Fluids - Course Assignment

Albert Taulera Campos - MSc Computational Mechanics

April 2016

1 Introduction

In this assignment, several exercises are going to be solved. They are all based on the same 2 dimensional domain, defined by the following characteristics:

A domain $\Omega = (0,2) \times (0,3) \in \mathbb{R}^2$. The boundary Γ, with Dirichlet and Neumann boundary conditions such that $\Gamma = \Gamma_d \cup \Gamma_n$, is defined by the following closed set as

$$\Gamma_1 = (0,0) \times (0,3/2)$$
$$\Gamma_2 = (0,0) \times (3/2,3)$$
$$\Gamma_3 = (0,2) \times (3,3)$$
$$\Gamma_4 = (2,2) \times (0,3)$$
$$\Gamma_5 = (0,2) \times (0,0)$$

Figure 1: Problem Statement

The codes used in all the sections are the ones that were given in the practical sessions of the subject or downloaded from the reference Antonio Huerta Lacàn webpage, widely complimented to cover all the subproblems proposed, all the new boundary and initial conditions and the new methodologies for both time and space integration.

2 Exercise 1

It is remarkable that the code came with a default velocity direction of v= [cos(pi/3),sin(pi/3)]*a, and I have worked with it as nothing else had been stated in the assignment.

**Remark: All the work had already been done when Pablo sent the in-campus message changing the velocity direction. For obvious reasons, I am not going to do the full assignment again.

- **a) Weak form and Discretization. Weighted residuals and Galerkin Method.**

  The Weak Form is obtained from the Srong Form in the following way:

  $$\int_\Omega w(\mathbf{a}\nabla u)d\Omega + \int_\Omega \nabla w(\nu\nabla u)d\Omega + \int_\Omega w\sigma u d\Omega = \int_\Omega w s d\Omega$$

  And basically the discretization relies on obtaining the next equation:

  $$Ku = f$$

  from our weak form. This is done through the following equalities:

  $$K_{ij} = \int_\Omega N_i(\mathbf{a}\nabla N_j)d\Omega + \int_\Omega \nabla N_i(\nu\nabla N_j)d\Omega$$

  and

  $$f_i = \int_\Omega N_i s d\Omega$$

- **b) Solution of the problem using the Galerkin approach. Overcoming the not acceptable solutions and SUPG, GLS approaches.**

  In this section, different values of convection, source, diffusion and reaction have been considered to study the different behavior of Galerkin, SUPG and GLS under different conditions, especially under different Péclet numbers. Péclet can be defined as:

  $$Pé = \frac{ah}{2\nu}$$

  and, as we will see, has a lot of influence in the results obtained and the node-to-node spurious oscillations that can be generated in our 2D analysis domain. The conditions contemplated in this section are:

1. $a = 1,\ \nu = 10^{-3},\ \sigma = 10^{-3},\ s = 0$

2. $a = 10^{-3},\ \nu = 10^{-3},\ \sigma = 1,\ s = 0$

3. $a = 1,\ \nu = 10^{-3},\ \sigma = 0,\ s = 1$

4. $a = 1,\ \nu = 10^{-3},\ \sigma = 1,\ s = 0$

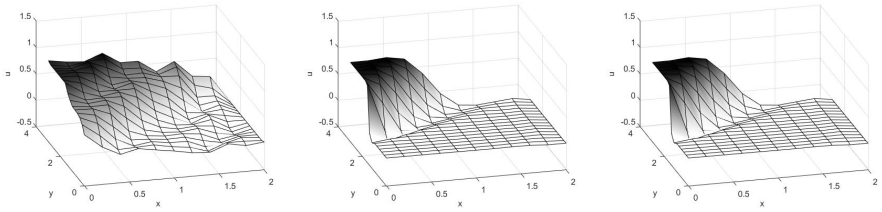
The BC are

$$\rho = 1\ in\ \Gamma_2$$

$$\rho = 0\ in\ \Gamma_4$$

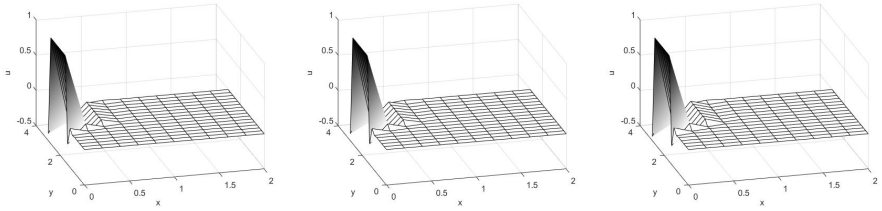
Figure 2: Subcases to be tested and Boundary Conditions

For each of the 4 combinations of relevant independent variables the code has been run and the results are shown next (Galerkin, SUPG and GLS)



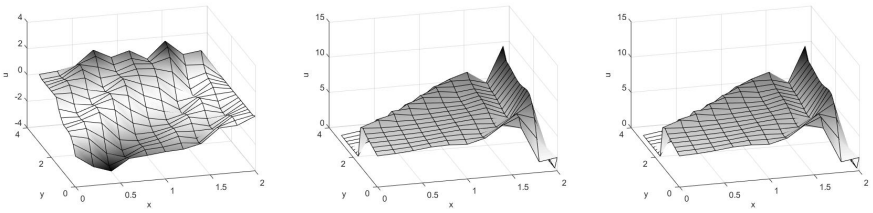
(a) Galerkin (b) SUPG (c) GLS

Figure 3: Case 1: a=1, $\nu = 10^{-3}$, $\sigma = 10^{-3}$, s=0



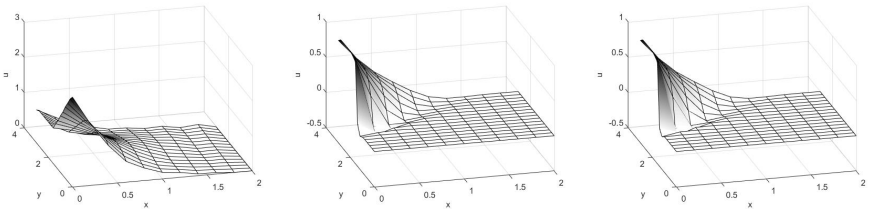
(a) Galerkin (b) SUPG (c) GLS

Figure 4: Case 2: a=10^{-3}, $\nu = 10^{-3}$, $\sigma = 1$, s=0



(a) Galerkin (b) SUPG (c) GLS

Figure 5: Case 3: a=1, $\nu = 10^{-3}$, $\sigma = 0$, s=1



(a) Galerkin (b) SUPG (c) GLS

Figure 6: Case 4: a=1, $\nu = 10^{-3}$, $\sigma = 1$, s=0

After a first check to all the results plotted here, it can easily be seen that Galerkin does not behave well in most cases, showing node-to-node oscillations. Recalling the slides from Finite Elements in Fluids, oscillations may appear for Pe>1.
It have been taken, for the following example for the Galerkin approach:

$$a = 1 \quad \nu = 0.1 \quad \sigma = 10^{-3} \quad s = 0 \quad Pé = 2 \quad h = 0.2$$

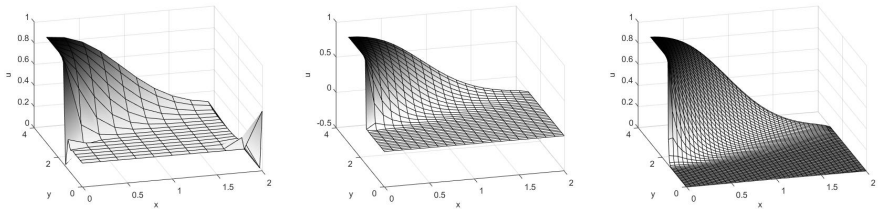
and the Péclet number has been subsequently reduced only by reducing the size of the elements. As the problem statment requires all the other variables to remain constant, the size of the element, h, is the only magnitude that we can change.
Subsequent iterations lead to:

$$a = 1 \quad \nu = 0.1 \quad \sigma = 10^{-3} \quad s = 0 \quad Pé = 1 \quad h = 0.1$$

and

$$a = 1 \quad \nu = 0.1 \quad \sigma = 10^{-3} \quad s = 0 \quad Pé = 0.5 \quad h = 0.05$$

The results would be:



(a) Pé=2, h=0.2 (b) Pé=1, h=0.1 (c) Pé=0.5, h=0.05

Figure 7: Different Pé and h values for the same Galerkin problem. Detection of node-to-node oscillations

It can be seen that in the first image,7a , some oscillations appear. Not many, because we are pretty close to the limit of stability, but they extist. In the previous examples Pé values were so high that the instabilities were easier to see.

On the other hand, images 7b and 7c show smooth solutions, which can also be obtained using the alternative methods proposed, SUPG and GLS. As it can be seen in all the previous sets of images, they offer a concise and enough smooth solution for every case without requiring a refined mesh.

These two methodologies (and others not mentioned here) allow us to benefit from the fact that they do not require fine meshes to find an adequate solution. As a result, Galerkin should not be the aproach to be prioritized in this kind of problems.

- **c) SUPG and GLS. Step-by-step explanation of the methodologies and Code Implementation.**

  Basically, these two methodologies are based on new parameters added to the weak form of the problem. These new parameters are "stabilizers", which make the solution much more consistent and without oscillations due to the Péclet number limitations. The basic weak form for the Galerkin approach was:

$$\int_\Omega w(\mathbf{a}\nabla u)d\Omega + \int_\Omega \nabla w(\nu\nabla u)d\Omega + \int_\Omega w\sigma u d\Omega = \int_\Omega w s d\Omega$$

  Basically, the SUPG approach to the weak form is the following, with stabilizer terms added, shown in red.

$$\int_\Omega w(\mathbf{a}\nabla u)d\Omega + \int_\Omega \nabla w(\nu\nabla u)d\Omega + \int_\Omega w\sigma u d\Omega + \sum_E \int_{\Omega^e} \mathbf{a}\nabla w\tau((\mathbf{a}\nabla u) - \nabla(\nu\nabla u) + \sigma u)d\Omega$$

$$= \int_\Omega w s d\Omega + \sum_E \int_{\Omega^e} (\mathbf{a}\nabla w)\tau s d\Omega$$

This stabilizer term is not the same that is used in GLS, they both differ a bit. For GLS, the terms used is:

$$\int_\Omega w(\mathbf{a}\nabla u)d\Omega + \int_\Omega \nabla w(\nu\nabla u)d\Omega + \int_\Omega w\sigma u d\Omega +$$
$$\sum_E \int_{\Omega^e} \mathbf{a}\nabla w - \nabla(\nu\nabla w) + \sigma w)\tau(\mathbf{a}\nabla u - \nabla(\nu\nabla u) + \sigma u)d\Omega$$
$$= \int_\Omega w s d\Omega + \sum_E \int_{\Omega^e} (\mathbf{a}\nabla w - \nabla(\nu\nabla w) + \sigma w)\tau s d\Omega$$

When it comes to the code implementation, it is quite easy to implement. First of all, the τ parameter has to be determined. It can be changed, but as default it is, for both formulations, the following one:

```
tau_p = h*(1 + 9/Pe^2 +(h*sigma/(2*a))^2)^(-1/2)/(2*a);
```

And, for both formulations the tau is included in the stiffness matrices and nodal force vectors of the element in the following way:

- SUPG

```
Ke = Ke + (N_igaus'*convi + nu*(Nx'*Nx+Ny'*Ny) +
sigma*N_igaus'*N_igaus  ...
        + tau*(convi'*convi + convi'*sigma*N_igaus))*dvolu ;


fe = fe + (N_igaus + tau*(ax*Nx+ay*Ny))'*(f_igaus*dvolu);
```

- GLS

```
Ke = Ke + (N_igaus'*convi + nu*(Nx'*Nx+Ny'*Ny) +
sigma*N_igaus'*N_igaus ...
        + tau*((convi + sigma*N_igaus)'*(convi +
        sigma*N_igaus)))*dvolu ;

fe = fe + (N_igaus + tau*(convi+sigma*N_igaus))'*(f_igaus*dvolu);
```

As it is remarked in the slides of the subject, SUPG and GLS are completely identical for convection-diffusion with linear elements. They become different when we introduce the reaction term. GLS obtains higher

contribution from the reaction term than SUPG, meaning that the instabilities that can be introduced by Galerkin approach are a bit more amplified in GLS if we compare them to SUPG.

To make differences appear (they are so little that can not be easily appreciated in the images shown before) we have used a huge reaction term value ($\sigma = 1000$). Here we can appreciate that GLS makes oscillations bigger, tracking the position of one of the nodes of our solution.

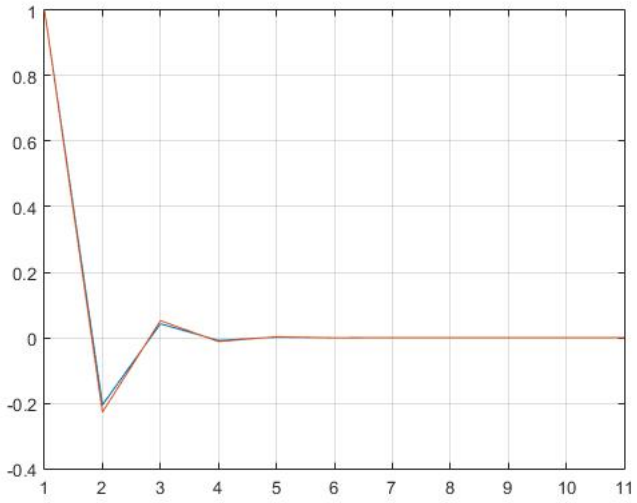
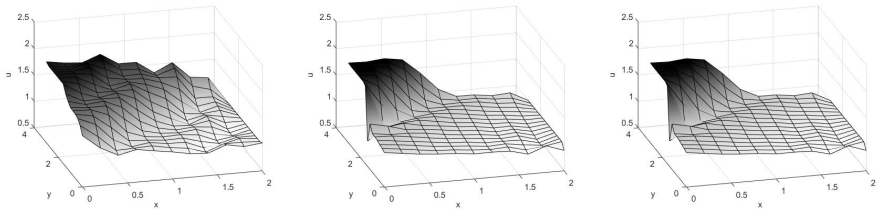


Figure 8: Comparative. Node 13. Analysis of SUPG and GLS with a=1, $\nu = 1$, $\sigma = 1000$ and s=0. GLS, in red, shows higher oscillations

- **d) Change in the boundary conditions and Neumann BC.**

  In this section, all boundary conditions (Dirichlet) have been raised in 1 point, having now $\rho = 2$ and $\rho = 1$ instead of $\rho = 1$ and $\rho = 0$, respectively. The solution should be similar, just with a relative translation in the ρ axis. The results, which are, indeed, very similar to the ones in 23:



(a) Galerkin (b) SUPG (c) GLS

Figure 9: Case 1: a=1, $\nu = 10^{-3}$, $\sigma = 10^{-3}$, s=0

3 Exercise 2

In this section, the transient term is included, together with new initial conditions and velocity vector. The new initial conditions are:

$$x_0(x,0) = x(2-x)$$

in Ω and the boundary conditions are going to be the same than in exercise 1. The convective term is:

$$a(x,y) = (-x,-y)$$

- **Discretization of the problem. Choosing a space discretization. Time discretization using 2 out of:**
  - Crank-Nicolson method
  - Two step third order TG method.
  - Padé approximation of order R22

  The systems have been worked separately, Crank-Nicolson and the Padé aproximmation of order R22, which are the ones chosen to work with.

  Moreover, all the methods have been implemented in the same way, using the theoric matrices of the implicit padé implementation. Implicit Padé aproxximation lies:

$$\frac{\triangle \boldsymbol{u}}{\triangle t} - \mathbf{W}\,\triangle \boldsymbol{u}_t = \mathbf{w}\,u_t^n,$$

$$\frac{\triangle \boldsymbol{u}}{\triangle t} + \mathbf{W}\mathcal{L}(\triangle \boldsymbol{u}) = \mathbf{w}\big[s^n - \mathcal{L}(u^n)\big] + \mathbf{W}\triangle \boldsymbol{s}.$$

Figure 10: Compact form for Padé aproxximation problems

where

$$\triangle \boldsymbol{u} = \begin{Bmatrix} u^{n+\beta_2} - u^n \\ u^{n+\beta_3} - u^{n+\beta_2} \\ \vdots \\ u^{n+1} - u^{n+\beta_{n_{tg}-1}} \end{Bmatrix}, \quad \text{and } \triangle \boldsymbol{u}_t = \begin{Bmatrix} u_t^{n+\beta_2} - u_t^n \\ u_t^{n+\beta_3} - u_t^{n+\beta_2} \\ \vdots \\ u_t^{n+1} - u_t^{n+\beta_{n_{tg}-1}} \end{Bmatrix}.$$

where W is a $(n_{tg}-1)x(n_{tg}-1)$ matrix and w a $(n_{tg}-1)$ vector. Their definition depends on every particular method and will be adapted for both Crank-Nicolson and R22.

– Crank-Nicolson method
The adaptation for W and w for the Crank-Nicolson method is the following one:

Second-order Padé approximation: $\boldsymbol{R}_{1,1}$ **(Crank–Nicolson)**

$$\Delta \boldsymbol{u} = u^{n+1} - u^n, \qquad \Delta \boldsymbol{s} = s^{n+1} - s^n,$$
$$\mathbf{W} = \frac{1}{2}, \qquad \mathbf{w} = 1.$$

Figure 11: C-N W and w coefficients

Obviously, for the C-N approach, both matrix and vector become scalar, obtaining the usual description of this methodology.

– Padé approximation of order R22

Fourth-order Padé approximation: $\boldsymbol{R}_{2,2}$

$$\Delta \boldsymbol{u} = \begin{Bmatrix} u^{n+1/2} - u^n \\ u^{n+1} - u^{n+1/2} \end{Bmatrix}, \qquad \Delta \boldsymbol{s} = \begin{Bmatrix} s^{n+1/2} - s^n \\ s^{n+1} - s^{n+1/2} \end{Bmatrix},$$
$$\mathbf{W} = \frac{1}{24}\begin{pmatrix} 7 & -1 \\ 13 & 5 \end{pmatrix}, \qquad \mathbf{w} = \frac{1}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}.$$

Figure 12: R22 W and w coefficients

These coefficients of w and W will be implemented in the code as different options so the solution can be computed with both methodologies. They are applied in the following way:

```
d_temp = input('Choose a method to perform time integration = ');
    if d_temp == 0
    method = 'CN + ';
    W = 1/2;
    w = 1;
    beta = [0,1];
elseif d_temp == 1
    method = 'R22 + ';
    W = (1/24)*[7 -1; 13 5];
    w = [1/2; 1/2];
    beta = [0,1/2,1];
```

and then used to compute the matrices of the system to solve, in each of the different methods available (Galerkin, LGS, SUPG), for example:

```
disp('Computation of total matrices for the time-integration scheme')
```

```
Kt  = C + nu*K;

A = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*M  + dt*\textbf{W(i,j)}*Kt];
    end
    A = [A; row];
end

nccd = size(Accd1,1);
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);
%Totality of the code available in the files%
```

Even though having different methods available, taking some conclusions from the exercise 1, as it is asked, the best time-step integrating methodologies to be used is the SUPG one, with the best stabilizing techniques and the lower error propagation.

- **Code programming**

  The full code used in this assignment is available as an attachment to this project.

  The interesting part to remark in this section is the imposition of boundary and initial conditions at the same time. We have:

$$\rho = 1 \; in \; \Gamma_2$$
$$\rho = 0 \; in \; \Gamma_4$$

Figure 13: Boundary Conditions

$$\mathbf{x}_0(\mathbf{x}, 0) = x(2 - x) \text{ in } \Omega.$$

Figure 14: Initial Conditions

which would be coded in the following way:

```
% INITIAL CONDITION
c = zeros(numnp,1);
for i=1:numnp;
    c(i,1)=X(i,1)*(2-X(i,1));
end


%BOUNDARY CONDITIONS
nodes_y0 = [2:nx]'                              ; % y=0
nodes_x1 = [(nx+1):nx+1:(ny+1)*(nx+1)]'     ;% x=1
nodes_y1 = [ny*(nx+1)+nx:-1:ny*(nx+1)+2]'     ;%  y=1
nodes_x0 = [(ny)*(nx+1)+1:-(nx+1):1]'     ;% x=0

%we require the division so we can have the different specified areas of
%boundary conditions.
nodes_x0_1 = nodes_x0(1:length(nodes_x0)/2) ;    % half x=0
nodes_x0_2 = nodes_x0(length(nodes_x0)/2+1:end);     % 2half x=0
% nodes_DC = [nodes_y0; nodes_x1; nodes_y1; nodes_x0];
nodes_DC=nodes_x1;
%new
nodes_DC_1=nodes_x0_1;
nodes=[nodes_DC;nodes_DC_1];
x1 = X(nodes_DC,1); y1 = X(nodes_DC,2);
x2 = X(nodes_DC_1,1); y2 = X(nodes_DC_1,2);
c=[nodes_DC,0*x1];
d=[nodes_DC_1,ones(length(x2),1)];
C=[c;d];
```

in this image we can see how does it behave in the first timestep of the code execution, when we have huge discontinuities due to this non-continuous conditions.

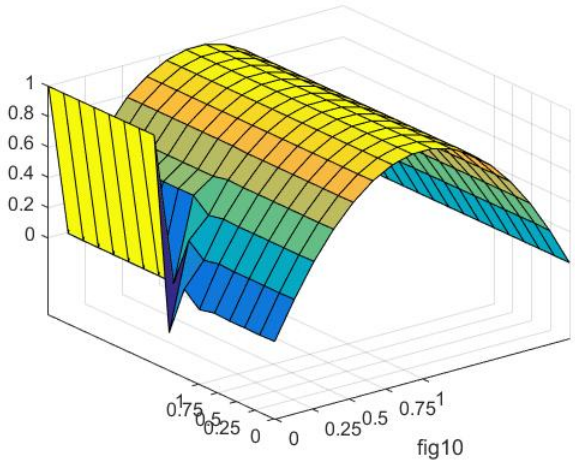


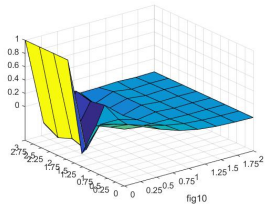
Figure 15: Initial Conditions + Boundary conditions

We expect to see a smoothing all along the timesteps so our problem shows realistic behavior.

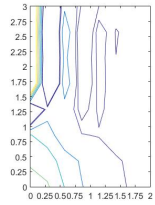
- **Mesh discretization and time discretization**

  For the sensitivity analysis on the mesh size we have used the following problem characteristics (with these values we guarantee Pé, C stability in most cases and and study of the full transient)

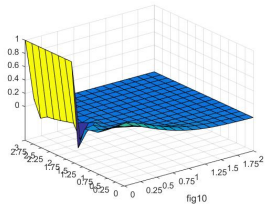
  $\nu = 0.1 \quad tstep = 0.02 \quad t_{end} = 2 \quad s = 0 \quad Crank{-}Nicolson \quad SUPG$



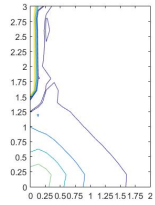
(a) Final tstep 7x7 mesh



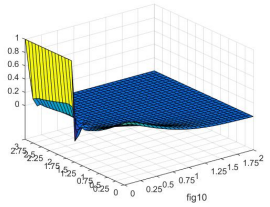
(b) Final tstep 7x7 mesh linear diagram



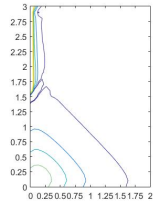
(c) Final tstep 15x15 mesh



(d) Final tstep 15x15 mesh linear diagram



(e) Final tstep 30x30 mesh



(f) Final tstep 30x30 mesh linear diagram

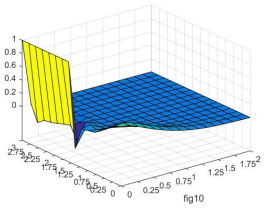
Figure 16: Differences between mesh discretization sizes

We can appreciate how the mesh refinements help improve our results. The 7x7 mesh gives pretty bad results and early oscillations that grow over 1, the maximum that we can achieve in our problem, so maybe that mesh is not the most adequate. From 15x15 and above all results seem to pretty much behave the same.

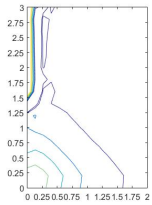
The computational cost has not been numerically determined, but as for every finite element analysis case, it is exponential and growing the number of elements from 15 to 30 in each direction highly increments our computation cost.

When it comes to timesteps and time discretization, the sensibility analysis will be the next. We have used a $t^n = 2$ because we can assume that the problem near to stationarity when we are close to that value. Therefore, we are going to work with the same final time and different timesteps.

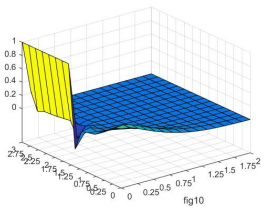
After trying with different values, it is obvious that as we go up in the timestep, higher errors are made in the solution of our problem. See how as we increase the Courant number, increasint the timestep, the solution every time becomes stranger and more distant to the usual solution showed before.



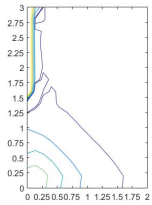
(a) Final tstep=0.01 mesh



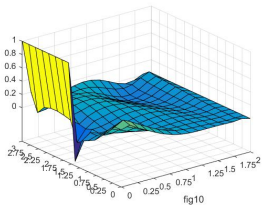
(b) Final tstep=0.01 mesh linear diagram



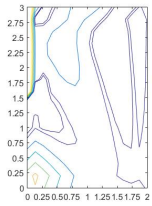
(c) Tstep=0.1 mesh



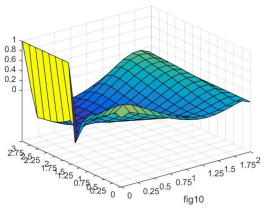
(d) Tstep=0.1 mesh linear diagram



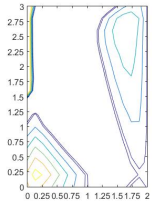
(e) Tstep=0.5 mesh



(f) Tstep=0.5 mesh linear diagram



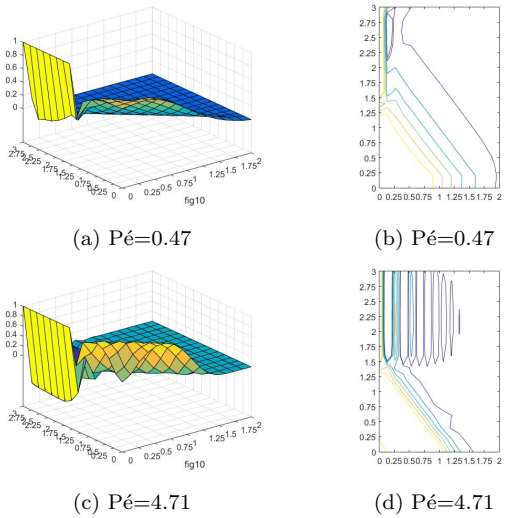
(g) Tstep=1 mesh



(h) Tstep=1 mesh linear diagram

Figure 17: Differences between time discretization sizes

They have also been studied the differences between Péclet numbers. Not in extension, but as an example we can see the differences using ν factors that are different, one stable and the other instable for the stability Péclet condition:



(a) Pé=0.47 (b) Pé=0.47

(c) Pé=4.71 (d) Pé=4.71

Figure 18: Differences between Péclet numbers

As one could expect, the high Péclet numbers above the limits stablished before (ex.1) show oscillations on the solution.

- **Quadratic elements implementation**

  Unfortunately, this part has not been done succesfully. It has been tried, though. To achive that purpose, several changes have been implemented; they are described next.

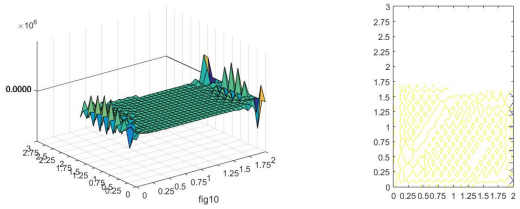
  First of all, shape functions for the quadratic elements have been included. The code for them is:

```
N    = [xi.*(xi-1).*eta.*(eta-1)/4, xi.*(xi+1).*eta.*(eta-1)/4, ...
           xi.*(xi+1).*eta.*(eta+1)/4, xi.*(xi-1).*eta.*(eta+1)/4, ...
           (1-xi.^2).*eta.*(eta-1)/2,  xi.*(xi+1).*(1-eta.^2)/2,   ...
           (1-xi.^2).*eta.*(eta+1)/2,  xi.*(xi-1).*(1-eta.^2)/2,   ...
           (1-xi.^2).*(1-eta.^2)];
```

  with their consequent derivatives. But that is not all what is required. The fact of inserting new nodes between the linear elements that we had requires the creation of a new mesh. For example, the new nodes coordinates are computed as:

```
% nodes coordinates
   for i=1:ny+1
       ys = ((i-1)*hy+y1)*vect_ones;
       posi = [(i-1)*(nx+1)+1:i*(nx+1)];
       X(posi,:)=[xs,ys];
   end
```

which completely differs from the previous implementation. New boundary conditions have also been implemented to fullfill the new node numbering. All the process has been carefully implemented but, finally, the solution matrix became singular, as it can be seen here:



(a) Solution obtained (b) Solution obtained

Figure 19: Quadratic elements solution. 15x15 $\nu = 0.1$

The solution matrix at each timestep, *sol*, gives a lot of results as NaN, which mean that they are divided by zero or infinite. The involved matrices; L, U and F have been reviewed and no source of singularity has been found. This probably is due to an implementation problem or some mistake in some part of the code. This would have been further tested or further checked if the amount of time available wasn't so limited.

4 Exercise 3

In this exercise, the following boundary conditions have been taken into account for the stokes problem:

$$\mathbf{v} = 0 \; in \; \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_5$$
$$\mathbf{v}_y = -1 \; in \; \Gamma_4$$

Figure 20: Problem Statement

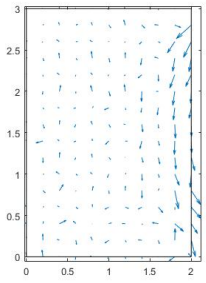
It is notorious that now there is a side of the rectangle that has different boundary conditions in the X and Y directions, only being constrained in the Y one. Therefore, every side has been constrained but the X direction of that side, which is not included as a Dirichlet boundary condition in the definitive code.

In this first section, the different element types that came implemented in the given code are going to be tested under different conditions to assess their suitability in the solution of the full problem.

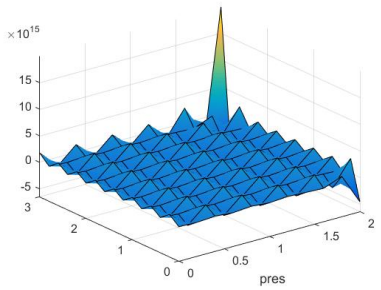
The different options are:

- Element Types : Triangular, Quadrilateral, Bubble
- Degree Velocity: 1, 2
- Degree Presure

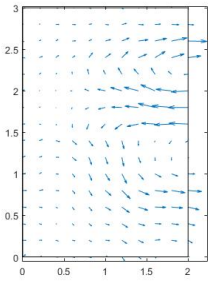
All subcases will be compared under the same conditions: 10x15 mesh.



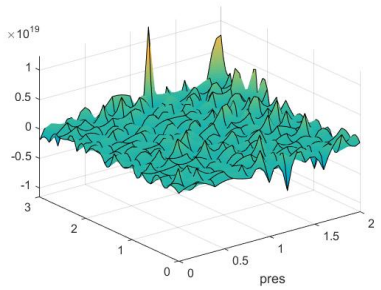
(a) Velocity Distribution - Linear/Linear



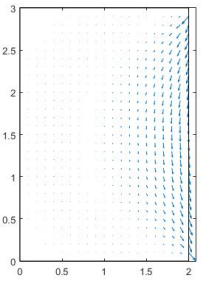
(b) Pressure Distribution Linear/Linear



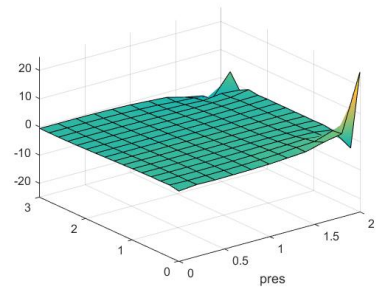
(c) Velocity Distribution - Linear/Quadratic



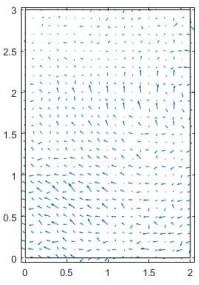
(d) Pressure Distribution Linear/Quadratic



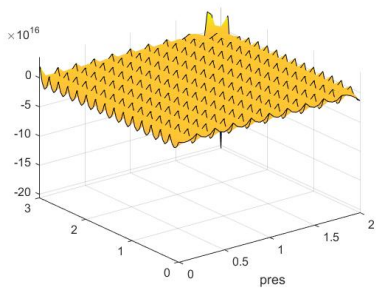
(e) Velocity Distribution - Quadratic/Linear



(f) Pressure Distribution Quadratic/Linear

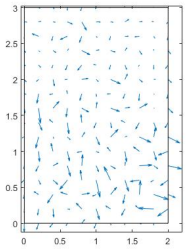


(g) Velocity Distribution - Quadratic/Quadratic

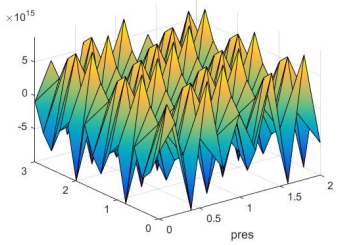


(h) Pressure Distribution Quadratic/Quadratic

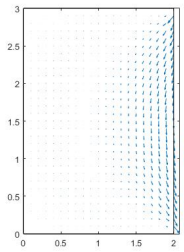
Figure 21: Quadrilateral elements



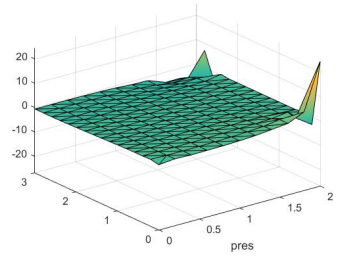
(a) Velocity Distribution - Linear/-Linear



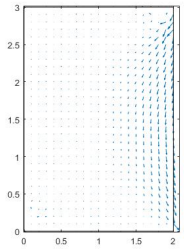
(b) Pressure Distribution Linear/Linear



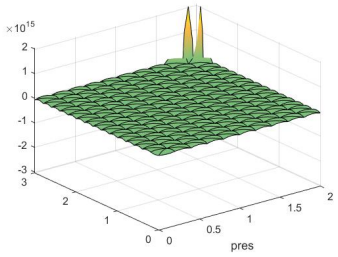
(c) Velocity Distribution - Quadratic/Linear



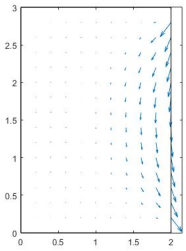
(d) Pressure Distribution Quadratic/Linear



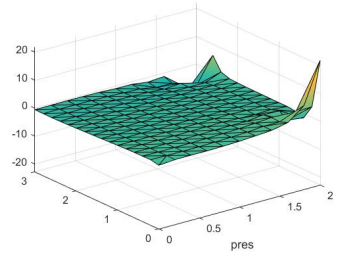
(e) Velocity Distribution - Quadratic/Quadratic



(f) Pressure Distribution Quadratic/Quadratic



(g) Bubble Triangles - Velocity Distribution - Linear/Linear



(h) Bubble Triangles -Pressure Distribution Linear/Linear

Figure 22: Triangular elements and Triangular with bubble function

Figure 23: Triangle with Bubble function elements

After a close look at all the combinations shown above we can easily state that the combination of elements and shape functions for velocity / pressure that work properly are:

- Triangular or Quadrilaterals with 2nd degree in Velocity and 1st degree in Pressure.
- Triangular with Bubble function for both 1st degree in Velocity and Pressure

The other ones were subject to several Matlab Warnings of singular matrices and won't be considered to give good solutions of the problem. Some of them are giving results that are close to the smooth ones obtained but with local oscillations that make them unrealistic.

- **Stokes. Weak Form and Discretization. Mesh size comparison and choosing criteria. Description of velocity and pressure fields.**

  The weak form of the Stokes problem can be written in the following way: Find the values of v and p that for all w functions accomplish:

$$\int_\Omega \nabla w : \sigma d\Omega = \int_\Omega w \cdot b d\Omega + \int_\Gamma w \cdot t d\Gamma$$

$$\int_\Omega q \nabla \cdot v d\Omega = 0$$

  After integration by parts and divergence theorem usage we can get to the expression:

$$0 = \int_\Omega \boldsymbol{w} \cdot \underbrace{(\boldsymbol{\nabla \cdot \sigma + b})}_{\text{equilibrium}} \, d\Omega + \int_\Omega q \, \underbrace{\boldsymbol{\nabla \cdot v}}_{\text{incomp.}} \, d\Omega - \int_{\Gamma_N} \boldsymbol{w} \cdot \underbrace{(\boldsymbol{n \cdot \sigma - t})}_{\text{Neumann b.c.}} \, d\Gamma.$$

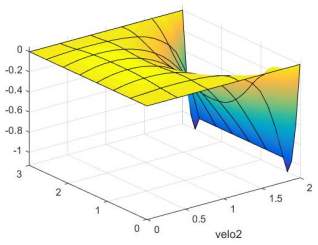
  where relevant terms influence can be seen. It can be rewritten in the following form, involving velocity and pressure:

$$\begin{cases} \boldsymbol{a(w,v) + b(w,p) = (w,b) + (w,t)_{\Gamma_N}} & \forall \boldsymbol{w} \in \boldsymbol{\mathcal{V}} \\ \boldsymbol{b(v,q) = 0} & \forall \boldsymbol{q} \in \boldsymbol{\mathcal{Q}} \end{cases}$$

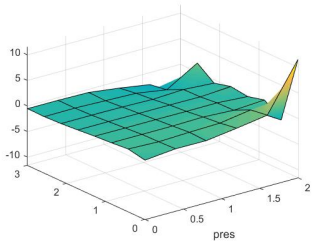
  with the matching discretization for the Galerkin Approach :

$$\sum_{j=1}^{n_{sd}} \Big\{ \sum_{B \in \eta \setminus \eta_{Dj}} a(N_A\, e_i, N_B\, e_j)\, u_{jB} \Big\} + \sum_{\hat{A} \in \hat{\eta}} b(N_A\, e_i, \hat{N}_{\hat{A}})\, p_{\hat{A}}$$
$$= (N_A\, e_i, b^h) + (N_A\, e_i, t^h)_{\Gamma_N} - \sum_{j=1}^{n_{sd}} \Big\{ \sum_{B \in \eta_{Dj}} a(N_A\, e_i, N_B\, e_j)\, v_{Dj} \Big\}.$$

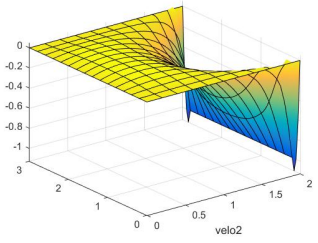
When it comes to mesh size, one of the most important and limiting factors is the computing performance of the computer. As this computations have been done in a laptop, the Matlab Code can not run meshes finer than 20x30 for V-Quadratic and P-Linear Quadrilateral elements.



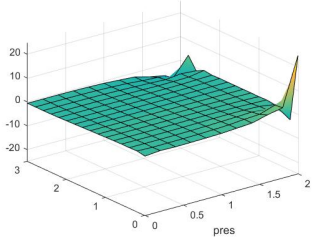
(a) Y Velocity Distribution - Quadratic/Linear



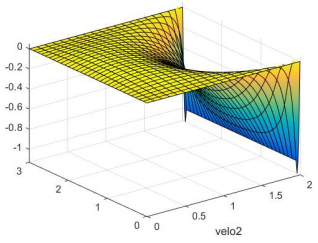
(b) Pressure Distribution Quadratic/Linear



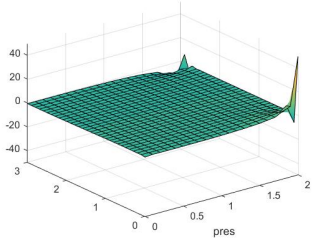
(c) Y Velocity Distribution - Quadratic/Linear



(d) Pressure Distribution Quadratic/Linear



(e) Y Velocity Distribution - Quadratic/Linear



(f) Pressure Distribution Quadratic/Linear

Figure 24: Differences in Y-Velocity and Pressure distributions using different refinement level meshes

As it can be appreciated, in the little range where we can study the behavior of the problem, almost all refinement mesh levels work fine.
It is true, though, that for the pressure distribution the singular points that can be seen in the limits of the rectangular domain grow and grow as the refinement increases. These two points should be studied further and with a much more refined meshes to extract some relevant conclusions regarding them. They can be related to some boundary conditions or some code issues.

In the images can be really well appreciated how the values of velocity stick to -1 in the imposed boundary condition on that side of the domain, while the rest of the rectangle adapts to this with a transition until it gets the value in all the other sides, 0. When it comes to pressure, we can appreciate singular pressure points in the corners of the -1 imposed velocity boundary. Their values can not be believed, as it grows with refinement and would require, as said, further study.

- **Navier-Stokes. Convergence depending on Re**

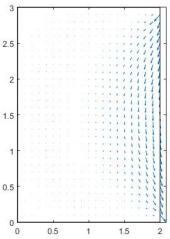
  Taking into account the given initial conditions, the code has been tested under the following specified conditions:

  - Re=1
  - Re=100
  - Re=1000
  - Re=2000

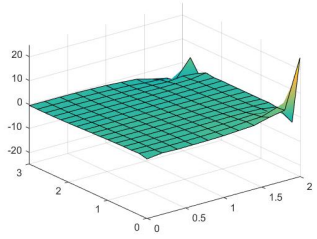
  only being it convergent in the two first subcases. The number of iterations were:

  - Re=1 - 3 iterations
  - Re=100 - 103 iterations
  - Re=1000 - Oscillating solution
  - Re=2000 - Oscillating solution

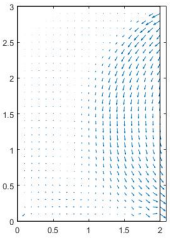
  The appearance of the solutions, velocity and pressure were:



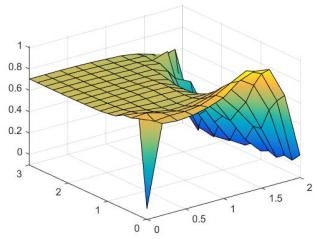
(a) Velocity Distribution - Re=1



(b) Pressure Distribution Re=1



(c) Velocity Distribution Re=100



(d) Pressure Distribution Re=100

Figure 25: Differences in Velocity and Pressure distributions depending on Reynolds number

Huge differences can be seen regarding the velocity and pressure fields, thing that is completely logical taking into account that the problem is completely different. The mesh used is 10x15, as was the one that gave better relation between performance/results in the last section of the exercise. Quadratic in velocity and Linear in pressure elements have been used also, as they were the ones that gave better results in the previous computations.

It can be seen how the velocity and pressure field suffer slower transitions as the Reynolds number goes up. The area of influence of the -1 velocity boundary is much higher when Re=100 than when it is equal to 1.

The problem has also been tested under different conditions when it comes to element types, showing that triangular bubble function elements have higher difficulties when it comes to convergence. Quadrilaterals and triangular elements work more or less at the same convergence ratio.