UNIVERSITÀ DEGLI STUDI DI PADOVA

UNIVERSITY OF PADOVA

Civil, Environmental and Architectural department DICEA

Master's degree in Mathematical Engineering

NMCS Project 2

Andrea Gorgi, Badge 2010658

Academic year 2021/2022

Contents

Introduction

Consider the "extended" Stokes equations:

$$cu - \mu\Delta u + \nabla p = f \qquad in\ \Omega, \tag{1}$$
$$\text{div } u = g \qquad in\ \Omega, \tag{2}$$
$$u = u_D \qquad in\ \Gamma, \tag{3}$$

where

- u: fluid velocity vector,
- p: fluid pressure,
- μ: fluid dynamic viscosity,
- c: constant,
- u_D: Dirichlet boundary values assigned for the velocity.

Working with 2-dimensional problems, $\Omega \subset \mathbb{R}^2$ and $\Gamma = \partial\Omega$.
If $c = 0$ here we have the ordinary Stokes problem, whereas a $c \neq 0$ could represent one term of an implicit Euler discretization of the time-dependent Stokes equation. In this case $c = 1/\Delta t$, where Δt is the time-step size.
The time-dependent Stokes problem is in fact

$$\partial_t u - \mu\Delta u + \nabla p = f \qquad in\ \Omega \times [0,T], \tag{4}$$
$$\text{div } u = g \qquad in\ \Omega \times [0,T], \tag{5}$$
$$u(x,t) = u_D(x,t) \qquad in\ \Gamma \times [0,T], \tag{6}$$
$$u(x,0) = u_0(x) \qquad in\ \Omega \times \{0\}, \tag{7}$$

where T is the maximum time. Applying a first order finite difference on the temporal term of the first equation, and using an implicit approach (all unknowns referred at the upper time level), we have

$$\frac{u^{n+1} - u^n}{\Delta t} - \mu\Delta u^{n+1} + \nabla p^{n+1} = f \implies \frac{1}{\Delta t}u^{n+1} - \mu\Delta u^{n+1} + \nabla p^{n+1} = f + \frac{1}{\Delta t}u^n, \tag{8}$$

with n time step index. So if $n = 0$ and $u_0 = 0$ in Ω, the formulation for the unknowns at the first time step (u^1 and p^1) is exactly the one showed above, with $c = \frac{1}{\Delta t}$. In the following we will actually study the time-dependent version of the problem, knowing that the same results could be obtained by choosing a zero initial condition for the velocity and a time step equal to $\frac{1}{c}$.

1 Weak and Galerkin formulation

For a generality scope, we derive the weak formulation of the time dependent Stokes equations even for Neumann boundary conditions (assume ρ density of the fluid equal to one).

$$\begin{cases} \partial_t u + \nabla p - \mu\Delta u = f & in\ \Omega \times [0,T] \\ \nabla \cdot u = g & in\ \Omega \times [0,T] \\ u(x) = u_D(x), & on\ \Sigma_D \times [0,T] \\ [pI - \mu\nabla u] \cdot n = g(x), & on\ \Sigma_N \times [0,T] \\ u(x,0) = u_0(x) & in\ \Omega \times \{0\}. \end{cases} \tag{9}$$

The solution to this system of PDEs describes a flow with a negligible convective term with respect to the viscous forces, thus a slow viscous flow $Re << 1$. To derive the weak formulation of eq.(9) we define the mappings $u(t) : \mathcal{I} \to \mathcal{V}$ and $p(t) : \mathcal{I} \to \mathcal{Q}$ so that we can look for solution $(u, p) \in L^2(\mathcal{I}, \mathcal{V}) \times L^2(\mathcal{I}, \mathcal{Q})$, where (chap. 5 [2]):

$$\begin{aligned} \mathcal{V} &= [H^1_\Gamma(\Omega)]^d \\ \mathcal{Q} &= L^2(\Omega), \end{aligned} \tag{10}$$

with d the space dimension of the problem (2 in our case), $H^1_\Gamma(\Omega) = \{v \in H^1(\Omega) \mid v = 0 \; in \; \Gamma \subset \partial\Omega\}$, H^1 Hilbert space and Γ portion of the boundary where we want to impose our Dirichlet boundary conditions. If the boundary condition are only of Dirichlet type for the velocity, the pressure appears only under a gradient sign in the equations and is then defined only up to a constant. In this case is thus necessary an additional condition to prevent oscillations (see (1 for the implementation),

$$\int_\Omega p dx = 0. \tag{11}$$

As first thing we compute the weak formulation of these equations using the standard variational approach, hence we multiply each term of the first equation by a test function $v \in [H^1_\Gamma(\Omega)]^d$, and each term of second equation by a $q \in L^2(\Omega)$ and integrate over Ω.

$$\begin{cases} \int_\Omega [(\partial_t u) \cdot v - \mu \Delta u \cdot v + \nabla p \cdot v] dx = \int_\Omega f \cdot v dx \\ \int_\Omega q \nabla \cdot u dx = \int_\Omega g \cdot v dx \end{cases} \tag{12}$$

Then we apply Gauss divergence theorem, obtaining

$$\begin{cases} \int_\Omega (\partial_t u) \cdot v dx + \int_\Omega \mu \nabla u : \nabla v dx - \int_\Omega p \nabla \cdot v dx - \int_{\partial\Omega} \mu \nabla u \cdot v \cdot n d\sigma + \int_{\partial\Omega} p v \cdot n d\sigma = \int_\Omega f \cdot v dx \\ \int_\Omega q \nabla \cdot u dx = \int_\Omega g \cdot v dx. \end{cases} \tag{13}$$

We need the test functions v to be zero in the Dirichlet boundaries, thus we choose $v \in [H^1_{\Sigma_D}(\Omega)]^d$. Simplifying the notation we end up with

$$\begin{cases} (\partial_t u, v) + a(u, v) + b(v, p) + \int_{\Sigma_N} [pI - \mu \nabla u] \cdot v \cdot n d\sigma = F(v) \\ b(u, q) = G(v). \end{cases} \tag{14}$$

The term integrated in the Neumann boundary Σ_N is automatically computed using the Neumann boundary conditions in (9), where we made use of the bilinear and linear forms (assuming μ constant in the domain)

$$\begin{aligned} (\partial_t u, v) &= \int_\Omega (\partial_t u) \cdot v dx \\ a(v, w) &= \mu \int_\Omega \nabla v : \nabla w dx \\ b(v, q) &= -\int_\Omega q \nabla \cdot v dx \\ F(v) &= \int_\Omega f \cdot v dx \\ G(v) &= -\int_\Omega g \cdot v dx. \end{aligned} \tag{15}$$

Since the Neumann b.c. are imposed automatically in the weak formulation of the problem, they are also referred as "natural" condition, whereas the Dirichlet ones, that have to be forced in the strong form, are called "essential".
We can now proceed and obtain the Galerkin formulation.

To this scope, let us define a regular triangulation $\mathcal{T}_h(\Omega)$ of the domain Ω, and a finite subsets $\mathcal{V}_h \subset \mathcal{V}$, $\mathcal{Q}_h \subset \mathcal{Q}$, dependent on the triangulation. calling $\{w_i\}$ the basis of $\mathcal{V}_h$ and $\{\phi_i\}$ the basis of $\mathcal{Q}_h$, the solution (u_h, p_h) will be expressed as

$$\begin{cases} u_h(x) = \sum_{j=1}^{N} u_j w_j(x), \\ p_h(x) = \sum_{k=1}^{M} p_k \phi_k(x). \end{cases} \tag{16}$$

The FEM discretization of the Stokes equation is then the finite dimensional counterpart of problem (9). In case of no Neumann boundary conditions with a backward Euler time stepping:

$$\frac{1}{\Delta t}(u_h^{n+1}, v) + a({u_h}^{n+1}, v) + b(v, p_h^{n+1}) = F^{n+1}(v) + \frac{1}{\Delta t}(u_h^n, v) \qquad \forall v \in \mathcal{V}_h(\mathcal{T}_h), \tag{17}$$

$$b({u_h}^{n+1}, q) = G^{n+1}(v) \qquad \forall q \in \mathcal{Q}_h(\mathcal{T}_h), \tag{18}$$

from which, equalizing the equations for each v in the basis for $\mathcal{V}_h$ and each q in the basis for $\mathcal{Q}_h$, we can build the linear system:

$$\begin{bmatrix} \frac{M}{\Delta t} + A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u^{n+1} \\ p^{n+1} \end{bmatrix} = \begin{bmatrix} \frac{M}{\Delta t} u^n + f \\ g \end{bmatrix}. \tag{19}$$

The matrix elements are

$$M = \{m_{ij}\} \qquad m_{ij} = m(w_i, w_j) = \int_\Omega w_i \cdot w_j dx \qquad i, j = 1, ..., N; \tag{20}$$

$$A = \{a_{ij}\} \qquad a_{ij} = a(w_i, w_j) = \mu \int_\Omega \nabla w_i : \nabla w_j dx \qquad i, j = 1, ..., 2N; \tag{21}$$

$$B = \{b_{ki}\} \qquad b_{ki} = b(w_i, J_k) = -\int_\Omega \phi_k \nabla \cdot w_i dx \qquad k = 1, ..., M; i = 1, ..., 2N; \tag{22}$$

$$f = \{f_i\} \qquad f_i = \int_\Omega f \cdot w_i dx \qquad i = 1, ..., 2N \tag{23}$$

$$g = \{g_i\} \qquad g_i = -\int_\Omega g \cdot \phi_k dx \qquad k = 1, ..., M. \tag{24}$$

It can be proved ([3]-[2]) that the above problem is well posed (i.e. the solution exists, is unique and is a continuous function of the data), if the following properties are satisfied:

1. the bilinear form $a(\cdot, \cdot)$ is continuous and coercive:

   $|a(v, w)| \leq \gamma ||v||_{\mathcal{V}_h} ||w||_{\mathcal{V}_h}, \qquad a(v, v) \geq \alpha ||v||_{\mathcal{V}_h}^2$ for all $v, w \in \mathcal{V}_h$, for some $\gamma, \alpha > 0$;

2. the bilinear form $b(\cdot, \cdot)$ is continuous and satisfies the inf-sup condition, i.e.:

   $b(v, q)| \leq \delta ||v||_{\mathcal{V}_h} ||q||_{\mathcal{Q}_h}, \qquad b(v, q) \geq \beta ||v||_{\mathcal{V}_h} ||q||_{\mathcal{Q}_h}$ for all $(v, q) \in \mathcal{V}_h \times \mathcal{Q}_h$, for some $\delta, \beta > 0$.

Pressure boundary condition

As written above, if no pressure condition in the boundary is prescribed for the problem the pressure solution can be obtained only up to a constant, and we have to introduce an additional constraint to fix the pressure to a precise value. To impose the given condition $\int_\Omega p dx = 0$ we use our discretization

$$\int_\Omega p dx = \sum_{T \in \mathcal{T}_h} \int_T \sum_{k=1}^{M} \phi_k p_k dx = \sum_{k=1}^{M} p_k \sum_{T \in \mathcal{T}_h} \int_T \phi_k dx = \sum_{K=1}^{M} p_k \sum_{T \in \mathcal{T}_h} \frac{Area(T)}{3} \delta(k, T),$$

where $\mathcal{T}_h$ is the pressure triangulation and $\delta(k,T) = 1$ if k is vertex of T, 0 otherwise. In this way we define an additional equation for the system, that we can either replace with one of the pressure related equations, thus forcing the solution to satisfy the Galerkin formulation for all the nodes expect the one associated to the substituted line, which will have a value dictated by this zero-mean condition, or add as additional constraint, and hence solve for example with least squares an over constrained problem.

1.1 Elements choice

The well-posedness for the FEM linear system of the above problem is guaranteed if the *discrete inf-sup condition* (also known as Babuska-Brezzi condition) holds: **For every** $p \neq 0$ **there must be a** v **such that**

$$p^T Bv \geq \beta ||v|| ||p|| \tag{25}$$

for a fixed $\beta > 0$. This condition guarantees that the system matrix $\mathcal{M} = \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix}$ is invertible. Such condition would immediately fail if a nonzero pressure p for which $Bp = 0$ exists. In the discrete case this is equivalent to impose that B has maximum rank (full column rank). In particular from (25) every possible nonzero pressure p must not be orthogonal to every Bv. Hence the space of all $Bv's$ must have at least the dimension of the pressure space. In our case B represent the divergence operator, and this is the reason why often the velocity elements are defined by polynomial with one degree higher than the ones for the pressure elements. Unfortunately, dimension of spaces are not sufficient to prove the discrete inf-sup condition (25), and it's necessary to check each particular couple of spaces $(\mathcal{V}_h, \mathcal{Q}_h)$. The general idea is then that the two spaces cannot be chosen independently and that, loosely speaking, we need to allow enough degrees of freedom in $\mathcal{V}_h$ with respect to $\mathcal{Q}_h$.
Here some typical results, writing P_0, P_1, P_2 for constant, linear and quadratic polynomials on triangles, and Q_0, Q_1, Q_2 for constant, bilinear, and biquadratic elements on quadrilaterals (from [5]):

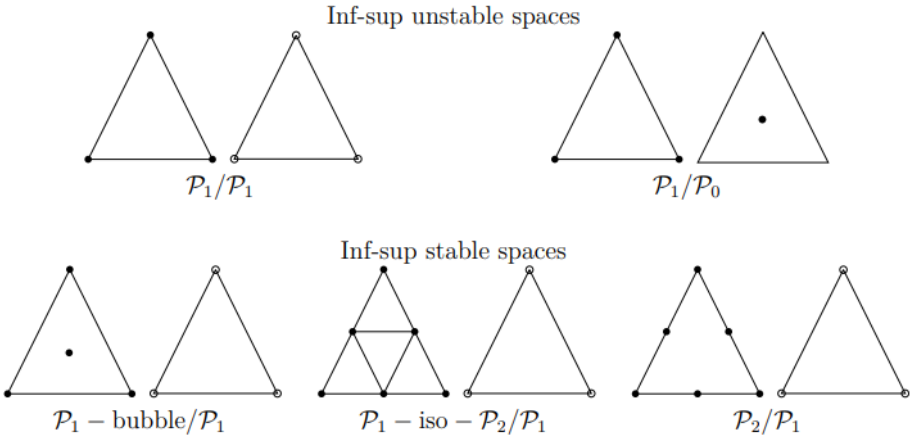
1. Velocities in P_1, pressures in P_0: failure
2. Velocities in P_1, pressures in P_1: failure
3. Velocities in P_2, pressures in P_1: **well posed**
4. Velocities in Q_1, pressures in Q_1: failure
5. Velocities in Q_2, pressures in Q_1: **well posed**.

The simplest choice for a triangular discretization would then be the P_2/P_1 approach, but since working with linear functions is way simpler than working with quadratic ones, we exploit the possibility of enriching the velocity space but starting from a P_1/P_1 approach, without moving to the next order.

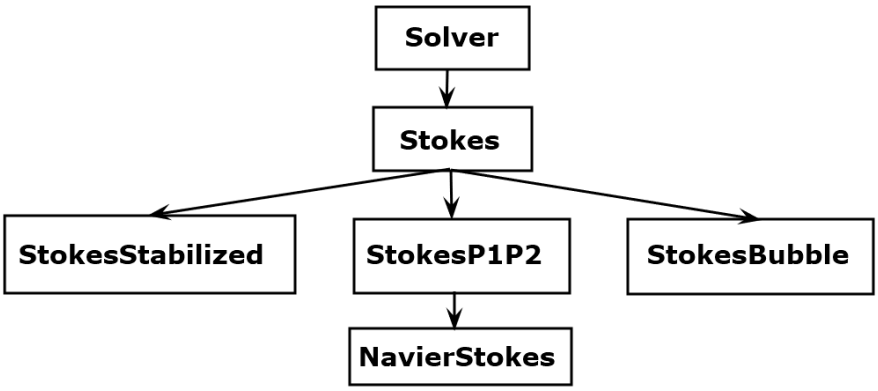
2 Implementation notes

The tests on this project has been conducted using codes built with an object oriented programming routine. The class hierarchy is showed in figure (2). Briefly:

- **Solver**: Has the most general methods for a solver, like the load of the mesh, the BCs definition or the results representation



Some inf-sup stable and unstable spaces



Class hierarchy of the code

- **Stokes**: Collects methods and properties for the Stokes problem resolution, involving functions for a time-dependent solver;
- **StokesStabilized**: Defines the functions needed to build the linear system associated to the Stokes problem with GLS stabilized P1/P1 elements;
- **StokesP1P2**: Defines the functions needed to build the linear system associated to the Stokes problem with P1 - isoP2/P1 elements;
- **StokesBubble**: Defines the functions needed to build the linear system associated to the Stokes problem with P1 - bubble P1 elements;
- **NavierStokes**: Collects the methods for a solver of the Navier Stokes equations with P1 - isoP2/P1 elements.

All this classes are used inside a "main.m" script where a user has the possibility to define the numerical values of the physical parameters.

2.1 Main script

We firstly assign a value to all the physical and numerical parameters.

```
 %%% Main script for the resolution of Stokes problem
clear all; close all; clc;

% Define the 2D mesh and equation BC's and coefficients
meshdir="mesh" + num2str(itmesh);
nfig   = 1;

%--------------------------------------------------------------------------
% SET PARAMETERS
%--------------------------------------------------------------------------
see_mesh  = false; % true to see the mesh
flag_BC   = 1; % BC: 0 for FAST penalty method
               %   : 1 for ACCURATE lifting function method
Δ     = 0.000001; % <--- ONLY FOR STOKESSTABILIZED
%-----------------------
%%% physical parameters
%-----------------------
mu           = 1;
Dt           = 100;
MaxTimeSteps = 1;

c = 1/Dt;
%-----------------------
%%% FIRST PROBLEM SOLUTION
%-----------------------
u_1 = @(x,y) -cos(2*pi.*x).*sin(2*pi.*y)+sin(2*pi.*y);
u_2 = @(x,y) sin(2*pi.*x).*cos(2*pi.*y)-sin(2*pi.*x);
p   = @(x,y) 2*pi.*(cos(2*pi.*y)-cos(2*pi.*x));
f_1 = @(x,y) c.*u_1(x,y)-4*mu*pi^2.*sin(2*pi.*y).*(2*cos(2*pi.*x)-1)+4*pi^2.*sin(2*pi.*x);
f_2 = @(x,y) c.*u_2(x,y)+4*mu*pi^2.*sin(2*pi.*x).*(2*cos(2*pi.*y)-1)-4*pi^2.*sin(2*pi.*y);
g   = @(x,y) 0;
```

Then we create an object of the class for the chosen stable approach, set the boundary conditions (automatically imposed as zero Dirichlet in each wall if not differently specified) and compute the matrices of the linear system.

```
%--------------------------------------------------------------------------
% INITIALIZE SOLVER
%--------------------------------------------------------------------------
tic;
disp('INITIALIZE SOLVER');
disp('-----------------------------------'); disp('-----------------------------------');
Stokes = StokesBubble();
Stokes = Stokes.initialize(meshdir, mu, f_1, f_2, g, see_mesh);

toc;
%--------------------------------------------------------------------------
% set BC
%--------------------------------------------------------------------------
tic;
disp('-----------------------------------')
disp('SET BC');
disp('-----------------------------------'); disp('-----------------------------------');

% Upper wall
% Stokes = Stokes.setDirBC(1, [0.0; 1.0],1.0,[1,0]);

% Lower wall

% Left wall

```

```
% Right wall

toc;
%------------------------------------------------------------------------
% PREPRO
%------------------------------------------------------------------------
tic;
disp('-----------------------------------')
disp('PREPRO');
disp('-----------------------------------'); disp('-----------------------------------');
Stokes = Stokes.prepro;
toc;
```

The code has been developed for a general time-dependent problem, thus we here have a loop over the chosen number of time steps of the solution (one in this case), with an initial condition for the velocity equal to zero everywhere. Finally we compute the approximation for the current time step and plot the results.

```
[uh1, uh2] = Stokes.zeroInitCond();

%%% !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
%%  LAUNCH SOLVER
for timestep = 1 : MaxTimeSteps
%------------------------------------------------------------------------
% SOLVE
%------------------------------------------------------------------------
tic;

disp('-----------------------------------')
disp('SOLVE MASTER PROBLEM');
disp('-----------------------------------');disp('-----------------------------------');
[uh1, uh2, ph] = Stokes.OneStepSolver(uh1, uh2, Dt, flag_BC);

toc;
%-----------------------
%%% PLOT VECTOR FIELDS/STREAMLINES
%-----------------------
figure(2)
for iedge = 1:size(Stokes.Bound.Edges,1)
    p1 = Stokes.V.coord(Stokes.Bound.Edges(iedge,1),:);
    p2 = Stokes.V.coord(Stokes.Bound.Edges(iedge,2),:);
    plot([p1(1), p2(1)], [p1(2), p2(2)], 'r-');
    hold on;
end

[x,y] = meshgrid(Stokes.X_limits(1):0.1:Stokes.X_limits(2), ...
                 Stokes.Y_limits(1):0.1:Stokes.Y_limits(2));
u = zeros(size(x)); v = zeros(size(x));
for i = 1:numel(x)
    [u(i), v(i)]= Stokes.getApproxValue_v([x(i);y(i)],uh1,uh2);
    if norm([u(i), v(i)]) < 1e-3
        u(i) = 0;
        v(i) = 0;
    end
end
quiver(x,y,u,v, 'Color', 'b');
streamslice(x,y,u,v);
hold off;

%-----------------------------
%%% MASS BALANCE CHECK
flux = 0;
for iedge = 1 : length(Stokes.Bound.Edges)
    for iloc = 1 : 2
        iglob = Stokes.Bound.Edges(iedge,iloc);
        flux = flux+Stokes.Bound.EdgesLength(iedge)*([uh1(iglob),uh2(iglob)]*...
        Stokes.Bound.ExtNormal(iedge))/2;
    end
```

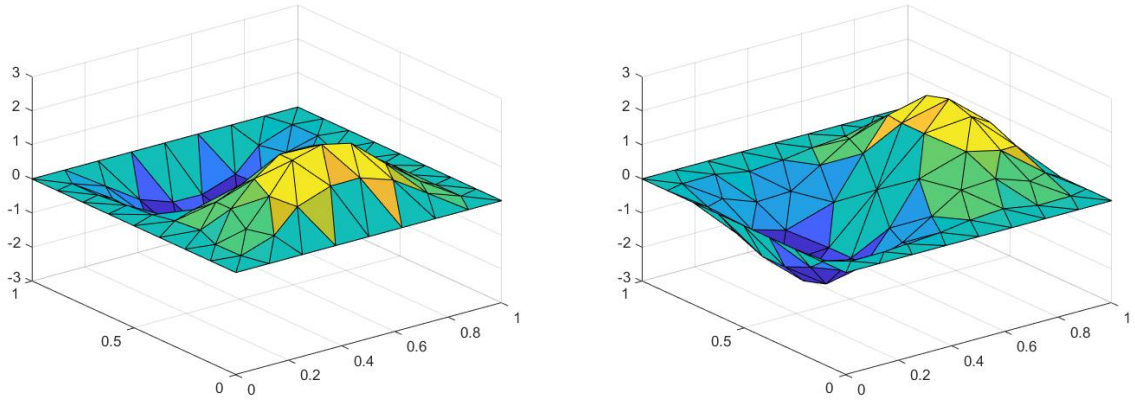
```
51  end
52  % Error:
53  fprintf('error err= %7.4e \n', norm(flux));
54  end
55
56  Stokes.PrintRes(uh1,uh2,ph,3);
57  Stokes.PrintResIso(uh1,uh2,ph,6);
```

3 P1/P1

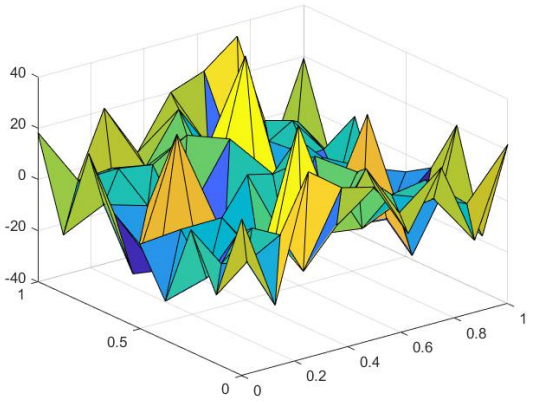
The P1/P1 approach makes use of linear shape functions for both the velocity and the pressure spaces, on the same nodes of the discretization. It's possible to prove that in this case $\dim(ker(B)) = 1 \neq 0$ (Chapter 3.3.3 of [2]) and therefore the inf sup condition is not satisfied. In particular we see that, calling for example $\bar{p}_h$ a pressure state in the kernel of B, we have in (18)

$$b(v,\bar{p}_h) = 0 \implies \frac{1}{\Delta t}(u_h^{n+1},v) + a({u_h}^{n+1},v) + b(v,p_h^{n+1}+\bar{p}_h) = \frac{1}{\Delta t}(u_h^{n+1},v) + a({u_h}^{n+1},v) + b(v,p_h^{n+1}) + b(v,\bar{p}_h)$$
$$= \frac{1}{\Delta t}(u_h^{n+1},v) + a({u_h}^{n+1},v) + b(v,p_h^{n+1}) = F^{n+1}(v) + \frac{1}{\Delta t}(u_h^n,v),$$

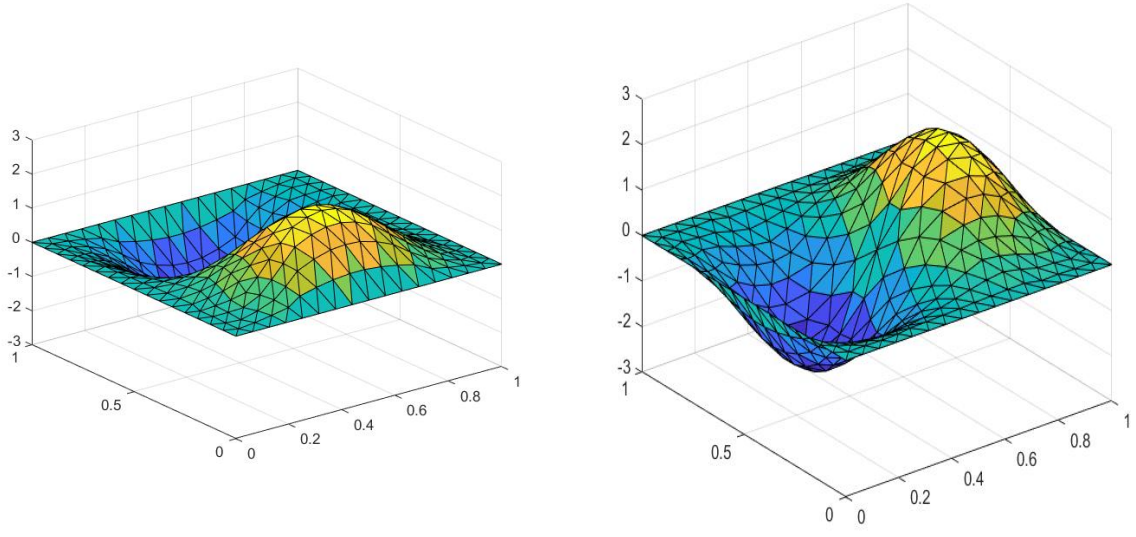
which means that if p_h^n is a solution for the n_{th} time step, also $p_h^n + const * \bar{p}_h$ will be an equally valid solution, causing the loss of uniqueness and the appearance of numerical instabilities. In this way, this elements choice allows the formation of spurious modes that result in oscillations in the pressure field.



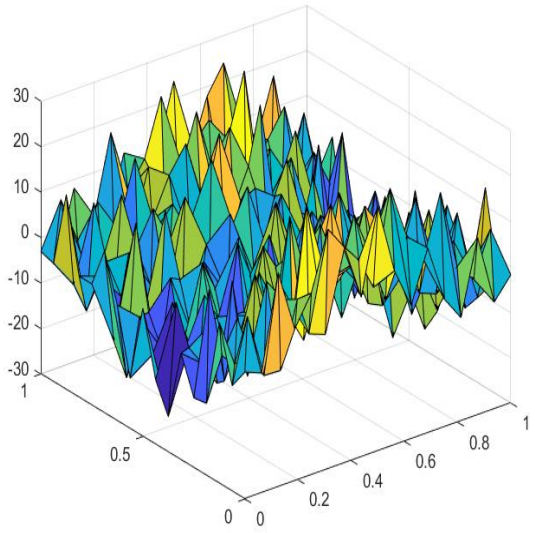
u_1 and u_2 approximated velocities with P1/P1 elements with provided mesh "mesh1".



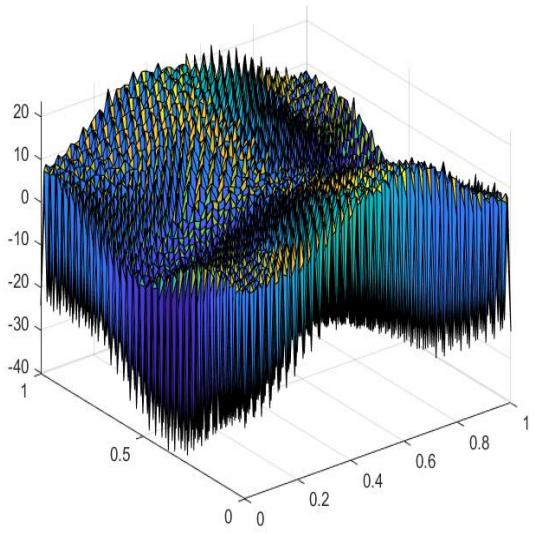
Approximated pressure field with P1/P1 elements with provided mesh "mesh1".



u_1 and u_2 approximated velocities with P1/P1 elements with provided mesh "mesh2".



(a) *Approximated pressure field with P1/P1 elements with provided mesh "mesh2".*



(b) *Approximated pressure field with P1/P1 elements with provided mesh "mesh4".*

We see that the more we decrease the mesh size the more oscillations appear in the pressure field, even if the velocities seem to stay stable.

4 GLS stabilized P1/P1

As discussed in section 1.1, stable spaces need to satisfy the "inf-sup" condition, which guarantees invertibility of the system matrix and well-posedness of the problem, and the starting point is to have a velocity space sufficiently "richer" than the pressure one, either increasing the number of nodes for the velocity or the degree of the polynomials used as basis functions. Anyway, in many practical problems, different approximating polynomials comes with both theoretical and implementation difficulties and the use of equal order polynomials would be much more advantageous.
The idea behind this stabilization is then to find another way (possibly consistent) to ensure the invertibility of the system matrix keeping the same structure for the basis of the velocity and pressure spaces. The matrix associated to our linear system is

$$\begin{bmatrix} \frac{M}{\Delta t} + A & B^T \\ B & 0 \end{bmatrix}.$$

This is a 2×2 block matrix, and it's clear that the most critical block is the empty one. In this case we know that if we replace this empty block an invertible matrix C the total matrix is always invertible given that also the first block $(\frac{M}{\Delta t} + A)$ is invertible (this is our case after the imposition of the boundary conditions).
With this approach we'd be modifying the original formulation just to reach a configuration with a stable convergence. The idea is then to make a consistent "variational crime" that generates an invertible matrix in the (2,2) position of the block matrix.
The GLS stabilized P1/P1 formulation for the time dependent Stokes problem is then:

Find $(u_h, p_h) \in \mathcal{V}_h(\mathcal{T}_h) \times \mathcal{Q}_h(\mathcal{T}_h)$ such that:

$$\begin{aligned}
&\frac{1}{\Delta t}(u_h^{n+1}, v) + a({u_h}^{n+1}, v) + b(v, p_h^{n+1}) = F^{n+1}(v) + \frac{1}{\Delta t}(u_h^n, v) && \forall v \in \mathcal{V}_h(\mathcal{T}_h),\\
&\quad b({u_h}^{n+1}, q) + c((u_h^{n+1}, p_h^{n+1}), (v, q)) = G^{n+1}(v) && \forall v \in \mathcal{V}_h(\mathcal{T}_h), \forall q \in \mathcal{Q}_h(\mathcal{T}_h),
\end{aligned}$$

where the new bilinear form is given by

$$c((u_h^{n+1}, p_h^{n+1}), (v, q)) = \delta \sum_{T \in \mathcal{T}_h} \int_T (\partial_t u_h^{n+1} - \mu \Delta u_h^{n+1} + \nabla p_h^{n+1} - f)(-\mu \Delta v + \nabla q) dx,$$

δ empirical tuning parameter for the stabilization. This formulation is clearly consistent with the time-dependent Stokes formulation since the term in the $c(\cdot, \cdot)$ operator is the residual and is equal to zero if we set $u_h = u$. Choosing

$$\begin{aligned}
\mathcal{V}_h(\mathcal{T}_h) &= \{v \in \mathcal{C}^0(\Omega) : v|T \in [\mathcal{P}_1(T)]^2\} = \text{Span}(w_1, ..., w_2 N) \subset \left[H^1_\Gamma(\Omega)\right]^2,\\
\mathcal{Q}_h(\mathcal{T}_h) &= \{q \in \mathcal{C}^0(\Omega) : q|_T \in \mathcal{P}_1(T)\} = \text{Span}(\phi_1, ..., \phi_N) \subset H^1_\Gamma(\Omega),
\end{aligned}$$

that is linear basis functions for both velocity and pressure spaces, and a first an implicit Euler scheme for the time discretization, we find the following linear system

$$\begin{bmatrix} A & B^T \\ B + B_C & -C \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ g + g_C \end{bmatrix},$$

with matrix entries defined as

$$\begin{aligned}
A = \{a_{ij}\} \quad & \text{same as above} && \\
B = \{b_{ki}\} \quad & \text{same as above} && \\
f = \{f_i\} \quad & \text{same as above} && \\
g = \{g_k\} \quad & \text{same as above} && \\
C = \{c_{km}\} \quad & c_{ki} = m(\phi_k, \phi_m) = \delta \sum_{T \in \mathcal{T}_h} h_T^2 \int_T \nabla \phi_k \cdot \nabla \phi_m dx && k, m = 1, ..., N\\
B_C = \{b_{C_{ki}}\} \quad & b_{C_{ki}} = -\frac{\delta}{\Delta t} \sum_{T \in \mathcal{T}_h} h_T^2 \int_T w_i \cdot \nabla \phi_k dx && k = 1, ..., N; i = 1, ..., 2N\\
g_C = \{g_{Ck}\} \quad & g_{Ck} = -\delta \sum_{T \in \mathcal{T}_h} h_T^2 \left[\frac{1}{\Delta t} \int_T u_{old} \cdot \nabla \phi_k dx + \int_T f \cdot \nabla \phi_k dx\right] && k = 1, ..., N.
\end{aligned}$$

Recalling that the velocity basis functions are

$$\begin{aligned}
w_i &= \begin{bmatrix} \phi_i \\ 0 \end{bmatrix} \quad i = 1, ..., N\\
&= \begin{bmatrix} 0 \\ \phi_i \end{bmatrix} \quad i = N+1, ..., 2N,
\end{aligned}$$

the entries of matrix B_C can be computed as

$$\begin{aligned}
b_{C_{ki}} &= -\frac{\delta}{\Delta t} \sum_{T \in \mathcal{T}_h} h_T^2 \int_T w_i \cdot \nabla \phi_k dx = -\frac{\delta}{\Delta t} \sum_{T \in \mathcal{T}_h} h_T^2 \int_T \phi_i \partial_x \phi_k dx\\
&= -\frac{\delta}{\Delta t} \sum_{T \in \mathcal{T}_h} h_T^2 \partial_x \phi_k \int_T \phi_i dx = -\frac{\delta}{\Delta t} \sum_{T \in \mathcal{T}_h} h_T^2 \frac{Area(T)}{3} \partial_x \phi_k, \quad i = 1, ..., N
\end{aligned}$$

and

$$b_{C_{ki}} = -\frac{\delta}{\Delta t} \sum_{T \in \mathcal{T}_h} h_T^2 \frac{Area(T)}{3} \partial_y \phi_k, \quad i = N+1, ..., 2N,$$

where we made use of the fact that since each basis function is linear in each element, their derivatives are constants. Moreover, given that u_{old} is the approximated solution obtained at the previous time step, thus $u_{old} = \sum_{i=1}^{2N} w_i u_i^{old}$ we have in g_C

$$\int_T u_{old} \cdot \nabla\phi_k dx = \sum_{i=1}^{2} N \int_T u_i^{old} w_i \cdot \nabla\phi_k dx = \sum_{i=1}^{2} N u_i^{old} \int_T w_i \cdot \nabla\phi_k dx,$$

and therefore

$$g_{C_k} = -\frac{\delta}{\Delta t} \sum_{T\in\mathcal{T}_h} h_T^2 \int_T u_{old} \cdot \nabla\phi_k dx = \sum_{i=1}^{2N} u_i^{old} \left[-\frac{\delta}{\Delta t} \sum_{T\in\mathcal{T}_\langle} h_T^2 \int_T w_i \cdot \nabla\phi_k dx \right] = \sum_{i=1}^{2N} u_i^{old} b_{C_{ki}}$$

$$\implies \left[g_C\right] = \left[B_C\right]\left[u_{old}\right].$$

Actually with this procedure we introduced unbalanced term in our system matrix, B_C, which will cause a loss of CPU time for the solver.

4.1 Code

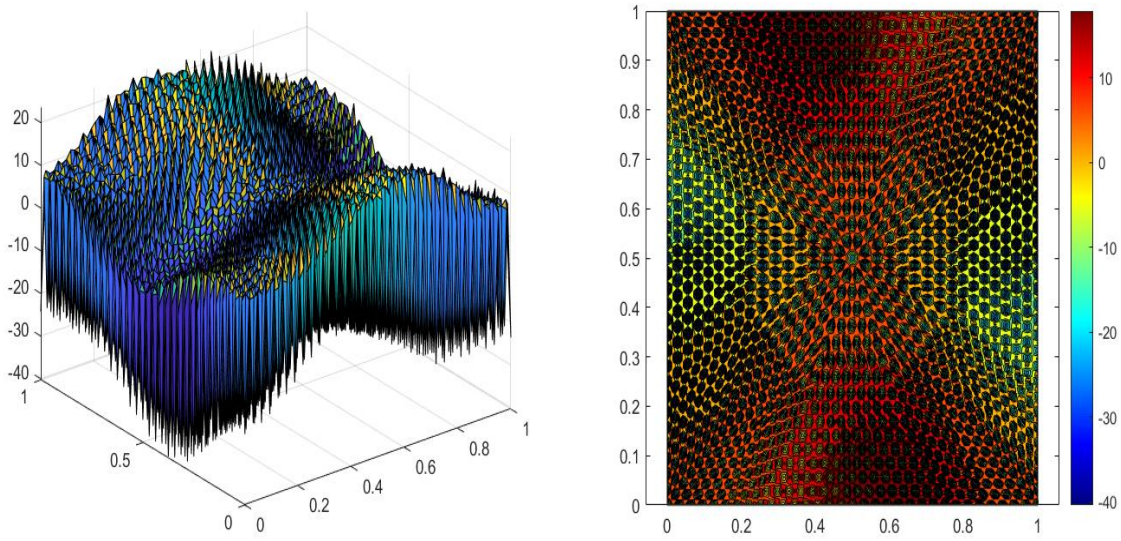
Here's the code for the construction of the matrices introduced above

```
%--------------------------------------------------------------------
        %% BUILD THE SYSTEM MATRICES
        function [M, A, B1, B2, C, BC1, BC2] = BuildStiffness(this)
            %----------------------------------------------------------------
            Nelem  = this.V.Nelem;
            triang = this.V.triang;
            Bloc   = this.V.Bloc;
            Cloc   = this.V.Cloc;
            Area   = this.V.Area;
            %----------------------------------------------------------------
            % initialize vectors for sparse allocation
            i_sparse = zeros(Nelem*9,1);
            j_sparse = zeros(Nelem*9,1);
            H_val    = zeros(Nelem*9,1);
            M_val    = zeros(Nelem*9,1);
            B1_val   = zeros(Nelem*9,1);
            BC1_val = zeros(Nelem*9,1);
            B2_val   = zeros(Nelem*9,1);
            BC2_val = zeros(Nelem*9,1);
            C_val    = zeros(Nelem*9,1);
            %----------------------------------------------------------------
            it=1;
            for iel=1:Nelem
                %------------------------------------------------------------
                for iloc=1:3
                    %--------------------------------------------------------
                    iglob=triang(iel,iloc);
                    for jloc=1:3
                        %----------------------------------------------------
                        jglob        = triang(iel,jloc);
                        i_sparse(it) = iglob;
                        j_sparse(it) = jglob;
                        %----------------------------------------------------
                        %%% DIFFUSION
                        %------------
                        H_val(it) = (Bloc(iel,iloc)*Bloc(iel,jloc)+...
                                     Cloc(iel,iloc)*Cloc(iel,jloc))*Area(iel);
                        %-----------------------------------------------
                        %%% MASS
                        %-------------
                        if iloc==jloc
```

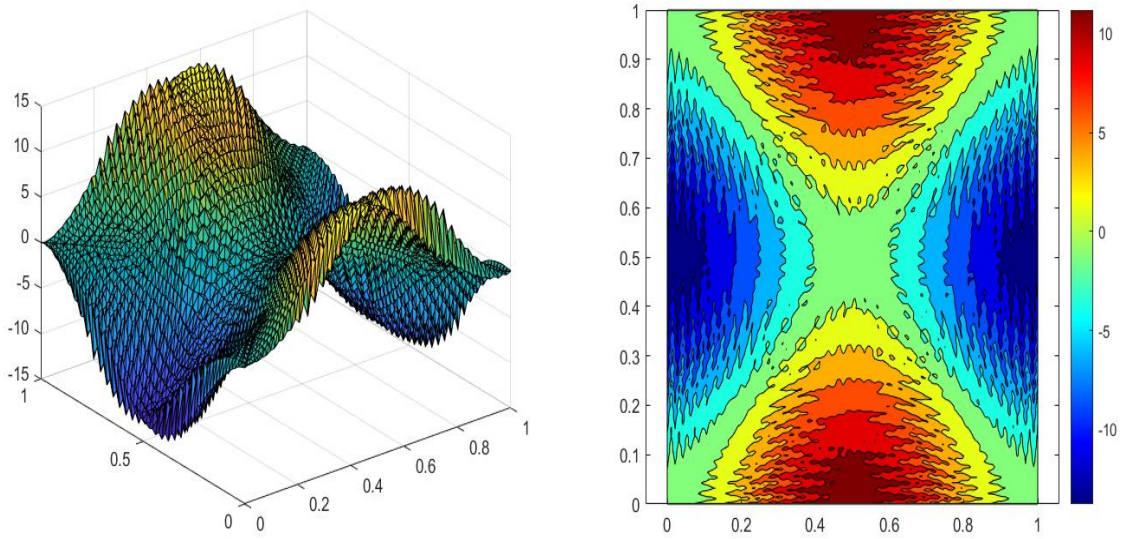
```
                            M_val(it) = Area(iel)/6;
                        else
                            M_val(it) = Area(iel)/12;
                        end
                        %-------------------------------------------
                        %%% DIVERGENCE/PRESSURE GRADIENT
                        %---------------
                        B1_val(it)=-1/3*Area(iel)*Bloc(iel,jloc);
                        B2_val(it)=-1/3*Area(iel)*Cloc(iel,jloc);
                        BC1_val(it) = B1_val(it)*this.h(iel)^2;
                        BC2_val(it) = B2_val(it)*this.h(iel)^2;
                        %-------------------------------------------
                        %%% STABILIZATION
                        %---------------
                        C_val(it)=this.Δ*this.h(iel)^2*(Bloc(iel,iloc)*Bloc(iel,jloc)+...
                                    Cloc(iel,iloc)*Cloc(iel,jloc))*Area(iel);
                        it=it+1;
                        %-------------------------------------------------
                    end % jloc
                end % iloc
            end % Nelem
            %-----------------------------------------------------------------
            A  = sparse(i_sparse,j_sparse,H_val);
            M  = this.rho*sparse(i_sparse,j_sparse,M_val);
            B1 = sparse(i_sparse,j_sparse,B1_val);
            B2 = sparse(i_sparse,j_sparse,B2_val);
            C  = sparse(i_sparse,j_sparse,C_val);

            BC1 = sparse(i_sparse,j_sparse,BC1_val);
            BC2 = sparse(i_sparse,j_sparse,BC2_val);
            %-----------------------------------------------------------------
        end
    end
```

4.2 Comparison for different value of δ

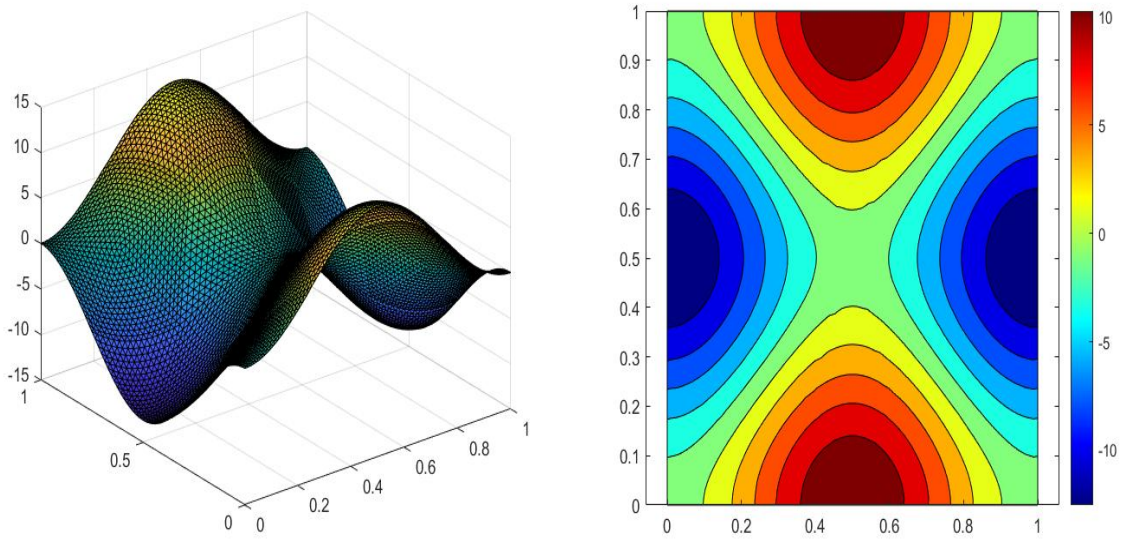
Below some results for different values of the stabilization parameter δ. The idea is to find the smallest possible value of δ which preserve the solution from oscillations, in order to commit the smallest possible variational crime.



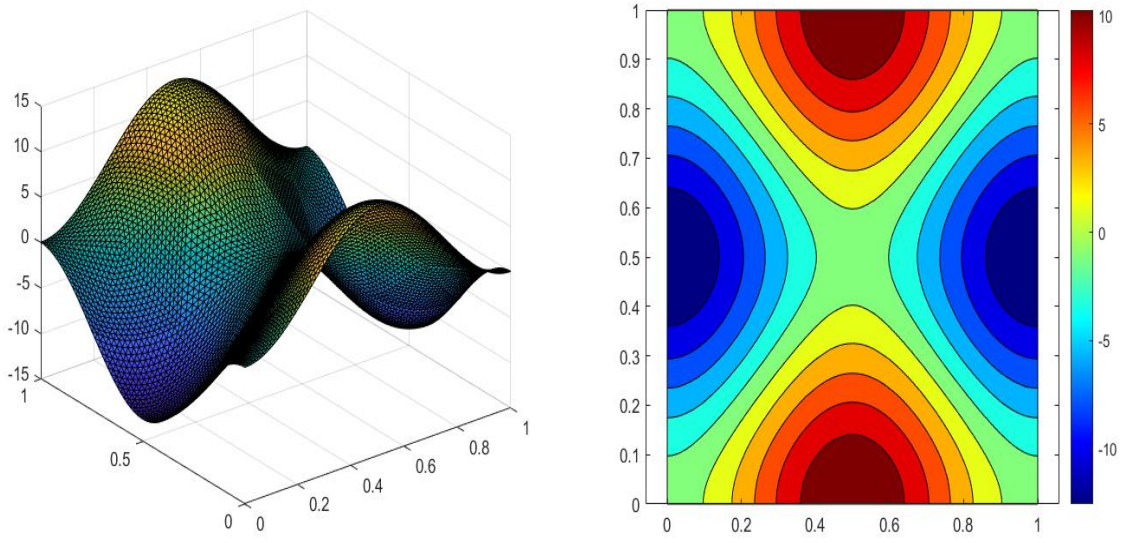
Approximated pressure state with "mesh4", $\delta = 0$. 3D plot (left), isosurface (right).



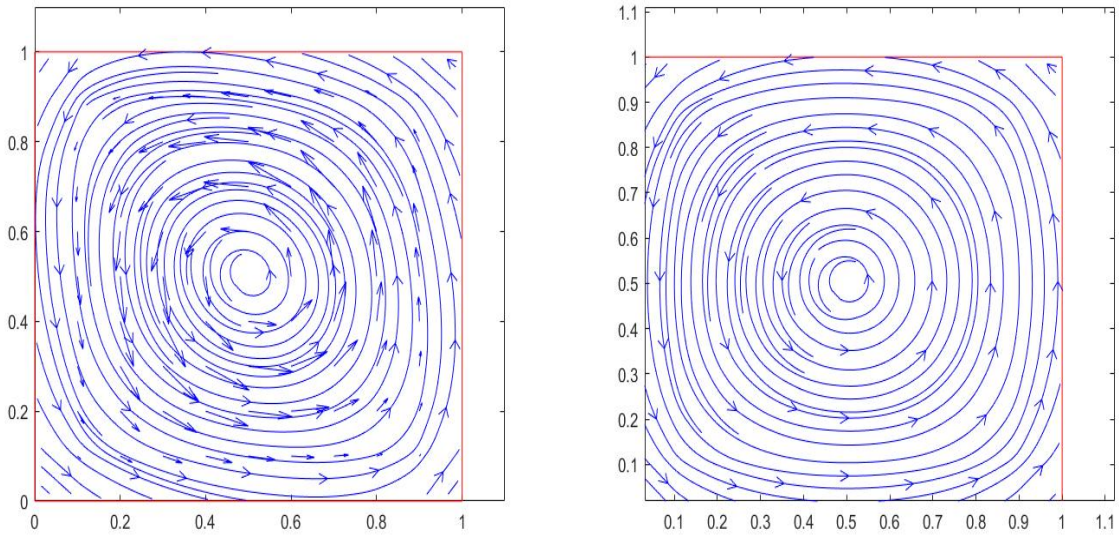
Approximated pressure state with "mesh4", $\delta = 0.0001$. 3D plot (left), isosurface (right).



Approximated pressure state with "mesh4", $\delta = 0.05$. 3D plot (left), isosurface (right).



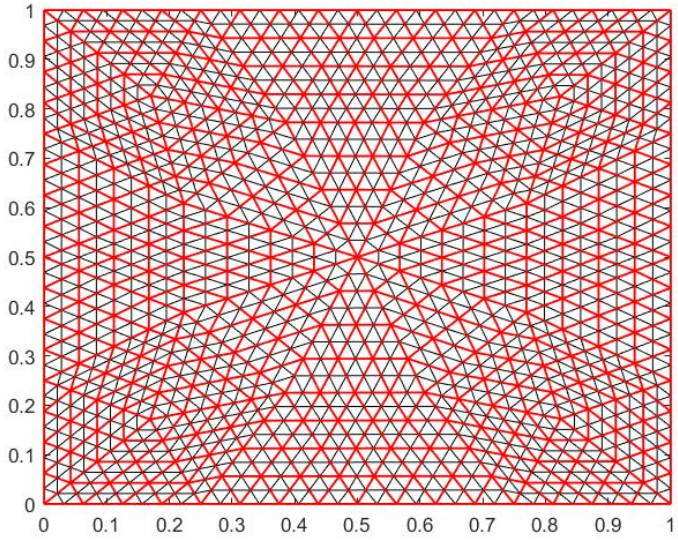
Approximated pressure state with "mesh4", $\delta = 10.0$. 3D plot (left), isosurface (right).



Approximated streamlines of the flow for $\delta \sim 10$ and $\delta = 0.05$. If the stabilization term is too high leads to miscalculations in the solution. Actually the u_1 and u_2 graphs show a a growing shift with respect to the real solution as we increase the value of δ.

5 P1 - iso P2/P1

The choice of P1-iso P2/P1 elements defines a a pressure and velocity spaces with linear polynomials as basis which satisfy the inf-sup condition by enriching the velocity space with a uniform refinement of the triangulation used for the pressure.
Following chapter 3.3.3 of [2], we thus have the following spaces



"mesh1" case for P1-isoP2/P1 elements. Pressure discretization and nodes (in red) and refined discretization for the velocity (in black).

$$\mathcal{V}_h = \{w_i \in \mathcal{C}^0(\mathcal{T}_h) : w_i|_T \in [\mathcal{P}_1(T)]^2 \forall T \in \mathcal{T}_h, \quad i = 1, ..., N_N^{(h)}\}$$
$$\mathcal{Q}_h = \{\psi_k \in \mathcal{C}^0(\mathcal{T}_{2h}) : \psi_k|_T \in \mathcal{P}_1(T) \forall T \in \mathcal{T}_{2h}, \quad k = 1, ..., N_N^{(2h)}\},$$

where $\mathcal{T}_h$ and $\mathcal{T}_{2h}$ are the refined and coarse triangulation respectively, $N_N^{(h)}$ is the number of nodes in the fine mesh and $N_N(2h)$ is the number of nodes in the coarse one.
From the computational point of view the construction of all the matrices and elements of the linear system which are affected only by the velocity test functions are built in the exact same way of the previous cases, considering the fine mesh in the computations. The only problem arises in the divergence and pressure gradient matrix B where the two basis functions interact with each other.
We know that

$$B_{i,k} = -\int_\Omega \psi_k \nabla \cdot w_i dx = \begin{cases} -\int_\Omega \psi_k \partial_x \phi_i dx, & i = 1, ..., N_N^{(h)} \\ -\int_\Omega \psi_k \partial_y \phi_i dx, & i = N_N^{(h)} + 1, ..., 2N_N^{(h)}. \end{cases}$$

In the following and in the code we will generally refer to $B_{i,k} = B1_{ik}$ if $i = 1, ..., N_N^{(h)}$ and $B_{ik} = B2_{i-N_N^{(k)}k}$ for the last $i = N_N^{(h)}+1, ..., 2N_N^{(h)}$ indices. This formulation introduces not irrelevant difficulties, since the ψ_k are defined on the pressure elements and the ϕ_i on the velocity ones, and we would have to work on both simultaneously. Luckily, given the smart construction of the refined mesh for the velocity, it's possible to leave apart the coarse discretization and work solely on the finest mesh, since it turns out that we can express the coarse mesh ($\mathcal{T}_{2h}$) basis functions ψ_k in terms of the fine mesh ($\mathcal{T}_h$) basis functions as follows

$$\psi_k(x) = \phi_k(x) + \frac{1}{2} \sum_{r \in \nu(k)} \phi_r(x), \tag{26}$$

where $\nu(k)$ is the set of indices connected to k via an edge of the fine mesh. The final formulation for the entries of B is then

$$B1_{ik} = \int_\Omega \phi_k \partial_x \phi_i dx + \frac{1}{2} \sum_{r \in \nu(k)} \int_\Omega \phi_r \partial_x \phi_i dx$$
$$B2_{ik} = \int_\Omega \phi_k \partial_y \phi_i dx + \frac{1}{2} \sum_{r \in \nu(k)} \int_\Omega \phi_r \partial_y \phi_i dx.$$

5.1 Implementation

In order to compute such entries for the $B1$ and $B2$ matrices a good idea is to split the integrals over the elements of the coarse mesh, and then each of these elements into the four elements of the fine mesh built inside of it. Basically for $B1$ ($B2$ is the same but with the derivative in the $y-$ direction)

$$\begin{aligned} B1_{ik} &= \sum_{T_{2h} \in \mathcal{T}_{2h}} \left[\int_{T_{2h}} \phi_k \partial_x \phi_i dx + \frac{1}{2} \sum_{r \in \nu(k)} \int_{T_{2h}} \phi_r \partial_x \phi_i dx \right] = \\ &= \sum_{T \in \mathcal{T}_{2h}} \sum_{T_h \in \varrho(T_{2h})} \left[\int_{T_h} \phi_k \partial_x \phi_i dx + \frac{1}{2} \sum_{r \in \nu(k)} \int_{T_h} \phi_r \partial_x \phi_i dx \right]. \end{aligned} \tag{27}$$

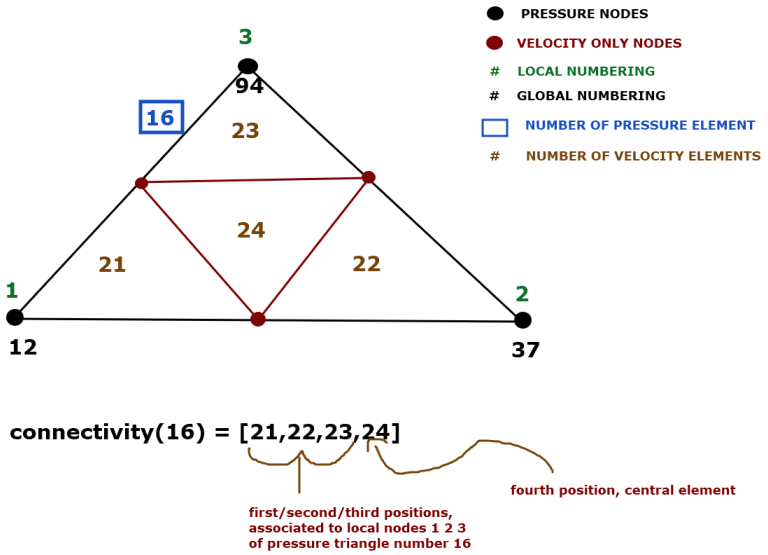
Now since the basis functions are linear in each element T_h their derivatives are constant and we can bring them out of the integrals to apply Trapezi's rule (since the remaining term in each integral is still a linear function),

$$\begin{aligned} B1_{ik} &= \sum_{T_{2h} \in \mathcal{T}_{2h}} \sum_{T_h \in \varrho(T_{2h})} \left[\partial_x \phi_i \delta(i, T_h) \int_{T_h} \phi_k dx + \frac{1}{2} \partial_x \phi_i \delta(i, T_h) \sum_{r \in \nu(k)} \int_{T_h} \phi_r dx \right] \\ &= \sum_{T_{2h} \in \mathcal{T}_{2h}} \sum_{T_h \in \varrho(T_{2h})} \frac{Area(T_h)}{3} \partial_x \phi_i \left[\delta(k, T_h) \delta(i, T_h) + \sum_{r \in \nu(k)} \frac{1}{2} \delta(r, T_h) \delta(i, T_h) \right], \end{aligned} \tag{28}$$

with

$$\delta(i, T) = \begin{cases} 1 & \text{if i vertex of } T \\ 0 & \text{otherwise} \end{cases}.$$

Fixing a triangle of a coarse mesh T_{2h} we have that the only possible k indices that produce a non null increment are the three associated to the vertices of the considered triangle. This simply because ψ_k, basis function associated to node k of the coarse mesh is identically null in the triangles where it is not one of the vertices. We therefore consider the three possible values for k referring to the local values from 1 to 3 of the vertices. Now, fixing one of the four elements of the fine mesh inside the chosen of the coarse mesh, the only i indices that contributes to the sum are the vertices of such triangle T_h. Hence in the code we will have a loop over the pressure elements, a loop over the three possible values of k of the vertices, then a loop over the four fine triangles and a final loop over the possible i indices of the fine triangle.
To do that we need to store in the memory the information about the indices of the triangles of the fine mesh built inside on each triangle of the coarse one. Since the fine mesh is constructed by hands starting from the coarse, the storing of these information is straightforward. In the code the matrix "connectivity" contains in each row (number of the pressure triangle) the four indices of the four velocity triangles. Moreover the four indices are inserted with reference to the local values of the nodes of the big triangle in the coarse mesh, that is, the first entry in the row refers to the small triangle which has the node of the big triangle with local numbering 1 as one of the vertices, the second entry points to the small triangle which has as one of the vertices the node of the big triangle with local number 2, and so the third for the remaining vertex. Last entry refers to the central element which doesn't share any vertex with the original triangle in the coarse mesh (13).



Construction of the connectivity matrix.

Calling $iel_{loc} = 1, ..., 4$ the local numbering of the fine inner triangles, $k_{loc} = 1, ..., 3$ the local indices of the coarse nodes, and rewriting

$$\delta(i, iel_{loc}) = \begin{cases} 1 & \text{if i vertex of local triangle } iel_{loc} \\ 0 & \text{otherwise} \end{cases}$$

we will clearly have $\delta(i, iel_{loc}) = 1$ for all the i vertices of the chosen local fine triangle. In this way the square brackets in (28) will be

$$\delta(k_{loc}, iel_{loc})\delta(i, iel_{loc}) = \delta(k_{loc}, iel_{loc}) = \begin{cases} 1 & if\ k_{loc} = iel_{loc} \\ 0 & otherwise \end{cases}$$

$$\sum_{r \in \nu(k)} \frac{1}{2}\delta(r, T_h)\delta(i, T_h) = \sum_{r \in \nu(k)} \frac{1}{2}\delta(r, T_h) = \begin{cases} 1 & if\ k_{loc} = iel_{loc}\ or\ iel_{loc} = 4 \\ 0.5 & otherwise \end{cases}$$

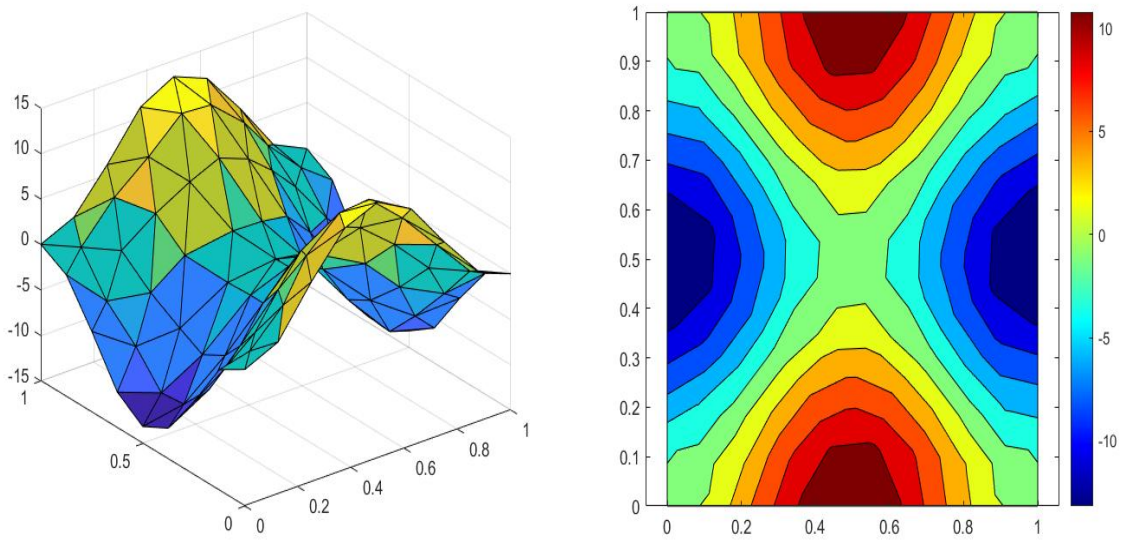
5.2 Code

Here's the code for the construction of the divergence/pressure gradient matrix

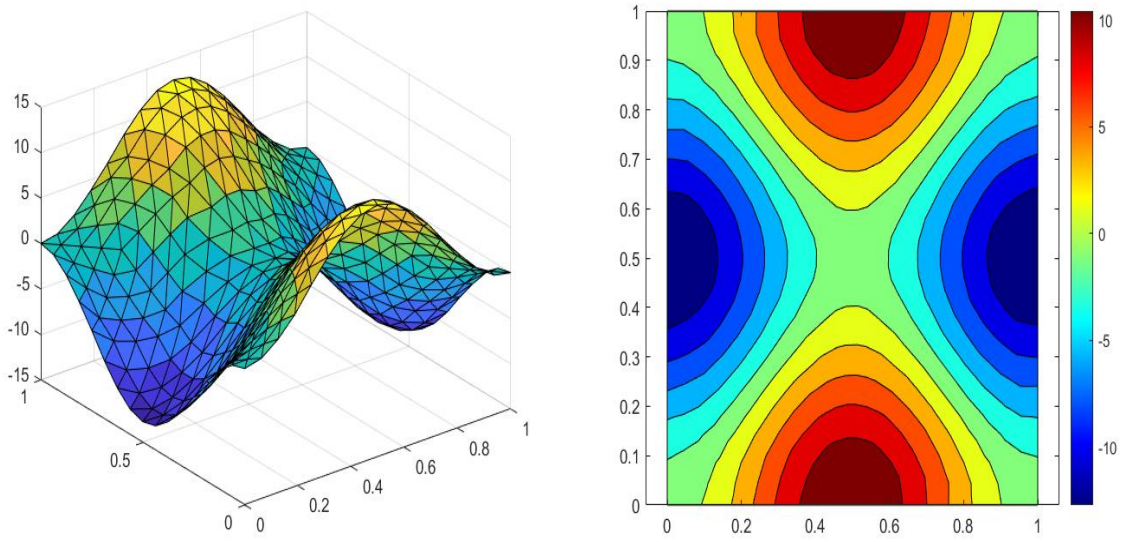
```
%-------------------------------------------------------------------
        %% BUILD BILINEAR FORM b(v,q)
        %-------------------------------------------------------------------
        function [B1, B2] = Build_b(this)
            %-----------------------------------------------------------
            triang_v = this.V.triang;
            Bloc_v   = this.V.Bloc;
            Cloc_v   = this.V.Cloc;
            Area_v   = this.V.Area;
            %-----------------------------------------------------------
            % initialize vectors for sparse allocation
            B1_val   = zeros(36*this.P.Nelem,1);
            B2_val   = zeros(36*this.P.Nelem,1);
            %
            i_p      = zeros(36*this.P.Nelem,1);
            k_p      = zeros(36*this.P.Nelem,1);
```

```
17              %
18              %-------------------------------------------------------------
19              % Build B1 and B2
20              it = 1;
21              for kel = 1 : this.P.Nelem
22                  triNB = this.connectivity(kel,:);
23                  for kloc = 1:3
24                      kglob = this.P.triang(kel,kloc);
25                      for iel_loc = 1 : 4
26                          iel = triNB(iel_loc);
27                          for iloc = 1 : 3
28                              iglob = triang_v(iel,iloc);
29                              B1 = 0;
30                              B2 = 0;
31                              %-----------------------------------------------
32                              if kloc == iel_loc
33                                  % first sum
34                                  B1=B1+Bloc_v(iel,iloc)*Area_v(iel)/3;
35                                  B2=B2+Cloc_v(iel,iloc)*Area_v(iel)/3;
36                              end
37                              %-----------------------------------------------
38                              if kloc == iel_loc || iel_loc == 4
39                                  % double second sum
40                                  B1=B1+0.5*2*Bloc_v(iel,iloc)*Area_v(iel)/3;
41                                  B2=B2+0.5*2*Cloc_v(iel,iloc)*Area_v(iel)/3;
42                              else
43                                  % just one second sum
44                                  B1=B1+0.5*Bloc_v(iel,iloc)*Area_v(iel)/3;
45                                  B2=B2+0.5*Cloc_v(iel,iloc)*Area_v(iel)/3;
46                              end
47                              %-----------------------------------------------
48                              % update
49                              i_p(it)=iglob;
50                              k_p(it)=kglob;
51                              B1_val(it)=-B1;
52                              B2_val(it)=-B2;
53                              it=it+1;
54                          end
55                      end
56                  end
57              end
58              %-------------------------------------------------------------
59              % BUILD MATRICES
60              B1=sparse(k_p,i_p,B1_val);
61              B2=sparse(k_p,i_p,B2_val);
62              %-------------------------------------------------------------
63          end
64      end
```

5.3 Some Results



Approximated pressure state with "mesh1". 3D plot (left), isosurface (right).



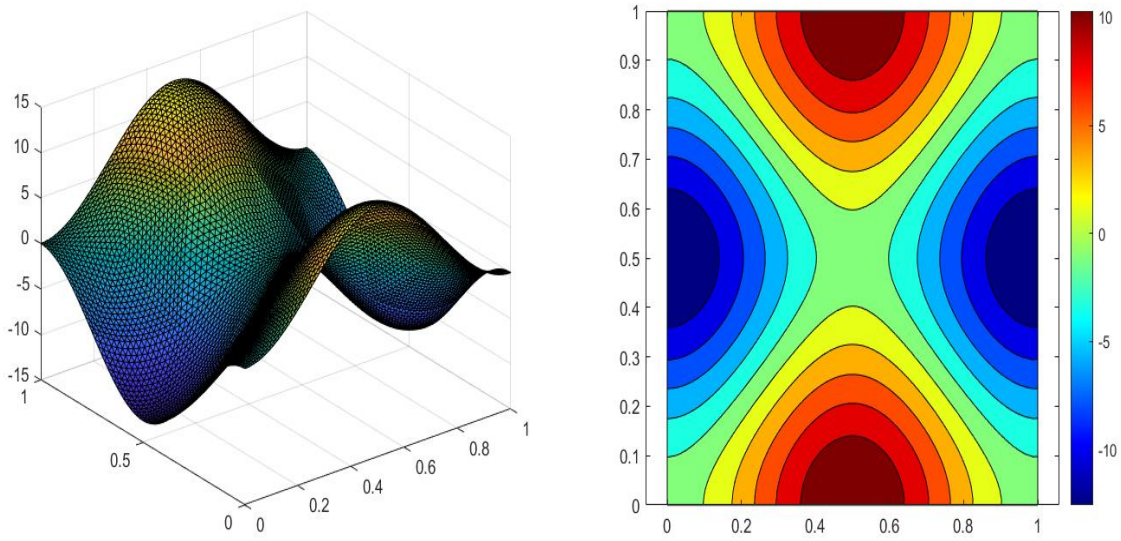
Approximated pressure state with "mesh2". 3D plot (left), isosurface (right).

6 P1 - bubble P1

Starting from linear polynomials as basis for both pressure and velocity spaces, we obtain a stable configuration by adding to each element a degree of freedom for the velocity in its center of gravity. In this case we have

$$\mathcal{P}_{h,1}^{\beta} = [\mathcal{P}_1(T) \oplus Span(\beta_T)]^2,$$

where $\beta_t(x)$ is the so called bubble function that has value 1 at the center of gravity of T and zero at the boundary, satisfying $0 \leq \beta_T \leq 1$ for each point in the domain.



Approximated pressure state with "mesh4". 3D plot (left), isosurface (right).

Considering a mesh triangle in the standard local configuration, with vertices in $P_1 = (0,0)$, $P_2 = (1,0)$ and $P_3 = (0,1)$, the four basis functions associated to this element has the form

$$\phi_1(x,y) = 1 - x - y; \quad \phi_2(x,y) = x; \quad \phi_3(x,y) = y; \quad \phi_\beta(x,y) = 27\phi_1(x,y)\phi_2(x,y)\phi_3(x,y). \tag{29}$$

Approximated velocities and pressure in the element will then be computed as

$$u_h(x,y) = \sum_{i=1}^{3} u_i\phi_i(x,y) + u_\beta\phi_\beta(x,y), \quad p_h(x,y) = \sum_{i=1}^{3} p_i\phi_i(x,y), \tag{30}$$

where u_i and u_β are two-dimensional vectors containing the x and y components of the nodal (baricentral) velocity vector [2].

6.1 Implementation

The idea for the implementation in this case is to transform the integrals over the various elements of the mesh, where we don't know the correct formulation for the bubble function and the integrals would be harder to compute, into integrals over the standard unitary triangle with a linear transformation of the coordinates.

In this way, the matrices entries introduced in (24) become

$$m_{ij} = \sum_{T\in\mathcal{T}_h} \int_T w_i(x,y)\cdot w_j(x,y)dxdy = \sum_{T\in\mathcal{T}_h} \int_{T^*} w_i^*(\xi,\eta)\cdot w_j^*(\xi,\eta)|det(J)|d\xi d\eta \quad i,j = 1,...,2N; \tag{31}$$

$$a_{ij} = \mu \sum_{T\in\mathcal{T}_h} \int_T \nabla_{x,y} w_i(x,y) : \nabla_{x,y} w_j(x,y)dxdy = \mu \sum_{T\in\mathcal{T}_h} \int_{T^*} \nabla_{x,y} w_i^*(\xi,\eta) : \nabla_{x,y} w_j^*(\xi,\eta)|det(J)|d\xi d\eta \quad i,j = 1,...,2N; \tag{32}$$

$$b_{ki} = -\sum_{T\in\mathcal{T}_h} \int_T \phi_k(x,y)\nabla_{x,y}\cdot w_i(x,y)dxdy = -\sum_{T\in\mathcal{T}_h} \int_{T^*} \phi_k^*(\xi,\eta)\nabla_{x,y}\cdot w_i^*(\xi,\eta)|det(J)|d\xi d\eta \quad k = 1,...,M; i = 1,...,2N, \tag{33}$$

where $|det(J)|$ is the determinant of the Jacobian matrix associated to the transformation from the global coordinates (x,y) to the local ones (ξ,η) and the $*$ apex refers to the basis functions in written in the

Change of coordinates

local system. T^* is the standard triangle with vertices $(0,0)$, $(1,0)$ and $(0,1)$, where the shape functions are respectively $\phi_1(\xi,\eta) = 1-\xi-\eta$, $\phi_2(\xi,\eta) = \xi$ and $\phi_3(\xi,\eta) = \eta$ for the three vertices, and the bubble function is $\phi_4 = \phi_\beta = 27\phi_1\phi_2\phi_3$.
The new local coordinates and the global ones have clearly a linear dependence from each other (no curvatures are introduced in the transformation) and thus we can compute the Jacobian matrix just using finite differences. Actually

$$Jac = \begin{bmatrix} \partial_\xi x & \partial_\eta x \\ \partial_\xi y & \partial_\eta y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}, \tag{34}$$

with the subscripts $1-3$ refers to the global values of the coordinates for the local vertices $1-3$. It is well known that the determinant of such Jacobian is $2Area(T)$. With small effort it's easy to find the Jacobian's inverse

$$Jac^{-1} = \begin{bmatrix} \partial_x \xi & \partial_y \xi \\ \partial_x \eta & \partial_y \eta \end{bmatrix} = \frac{1}{2Area(T)} \begin{bmatrix} y_3 - y_1 & -(x_3 - x_1) \\ -(y_2 - y_1) & x_2 - x_1 \end{bmatrix}. \tag{35}$$

We now have to transform the derivatives with respect to x and y in the above equations into the derivatives with respect to the local coordinates. Hence, instead of save in memory the values of the local derivatives of the basis functions for each element, we will save the values of the entries of the inverse of the Jacobian matrix for each triangle, and just once the known values of the gradient of the local basis functions in the local coordinates. We know that

$$\frac{\partial \phi}{\partial x} = \frac{\partial \phi}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial \phi}{\partial \eta}\frac{\partial \eta}{\partial x}$$
$$\frac{\partial \phi}{\partial y} = \frac{\partial \phi}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial \phi}{\partial \eta}\frac{\partial \eta}{\partial y}$$

which implies

$$\nabla_{x,y}\phi^*(\xi,\eta) = \begin{bmatrix} \partial_x \phi^* & \partial_y \phi^* \end{bmatrix} = \begin{bmatrix} \partial_\xi \phi^* & \partial_\eta \phi^* \end{bmatrix} Jac^{-1} = \nabla_{\xi,\eta}\phi^* Jac^{-1}, \tag{36}$$

and we can therefore compute all the matrix entries using only the local coordinates and the simple form of the basis functions in such local system.
We will then have for the mass matrix

$$m_{i,j}^{(T)} = \begin{cases} \dfrac{Area(T)}{6} & if\ i = j,\ i,j < 4 \\ \dfrac{Area(T)}{12} & if\ i \neq j\ i,j < 4 \\ 2Area(T)\displaystyle\int_{T^*} w_i^* \cdot w_j^* d\xi d\eta & if\ i = 4\ or\ j = 4. \end{cases}$$

In the last case we can compute exactly the integral for each case, obtaining

$$\int_{T^*} w_1^* \cdot w_4^* d\xi d\eta = \int_0^1 \int_0^{1-x} 27\xi\eta(1-\xi-\eta)^2 d\xi d\eta = \frac{3}{40}$$
$$\int_{T^*} w_2^* \cdot w_4^* d\xi d\eta = \int_0^1 \int_0^{1-x} 27\xi^2\eta(1-\xi-\eta) d\xi d\eta = \frac{3}{40}$$
$$\int_{T^*} w_3^* \cdot w_4^* d\xi d\eta = \int_0^1 \int_0^{1-x} 27\xi\eta^2(1-\xi-\eta) d\xi d\eta = \frac{3}{40}$$
$$\int_{T^*} w_4^* \cdot w_4^* d\xi d\eta = \int_0^1 \int_0^{1-x} (27\xi\eta(1-\xi-\eta))^2 d\xi d\eta = \frac{81}{560}.$$

For the stiffness matrix (assume $i = 1, ..., N$, so that $w_i = \begin{bmatrix} \phi_i \\ 0 \end{bmatrix}$

$$a_{ij}^{(T)} = 2 * Area(T) \int_T^* (\nabla_{\xi,\eta}\phi_i^* Jac^{-1}) \cdot (\nabla_{\xi,\eta}\phi_j^* Jac^{-1}) d\xi d\eta.$$

For what concerns the basis functions of the vertices, local indices i and j less then 4, the gradient is constant along the element end the entry becomes simply

$$a_{i,j}^{(T)} = 2 * Area(T) Area(T^*) (\nabla_{\xi,\eta}\phi_i^* Jac^{-1}) \cdot (\nabla_{\xi,\eta}\phi_j^* Jac^{-1}) = Area(T) (\nabla_{\xi,\eta}\phi_i^* Jac^{-1}) \cdot (\nabla_{\xi,\eta}\phi_j^* Jac^{-1}).$$

In all the other cases we directly compute the integral and considering the product

$$(\nabla_{\xi,\eta}\phi_i^* Jac^{-1}) \cdot (\nabla_{\xi,\eta}\phi_j^* Jac^{-1}) = \left(\frac{\partial\xi}{\partial x}\right)^2 \frac{\partial\phi_i}{\partial\xi}\frac{\partial\phi_j}{\partial\xi} + \frac{\partial\xi}{\partial x}\frac{\partial\eta}{\partial x}\left[\frac{\partial\phi_i}{\partial\xi}\frac{\partial\phi_j}{\partial\eta} + \frac{\partial\phi_i}{\partial\eta}\frac{\partial\phi_j}{\partial\xi}\right] + \left(\frac{\partial\eta}{\partial x}\right)^2 \frac{\partial\phi_i}{\partial\eta}\frac{\partial\phi_j}{\partial\eta} +$$
$$\left(\frac{\partial\xi}{\partial y}\right)^2 \frac{\partial\phi_i}{\partial\xi}\frac{\partial\phi_j}{\partial\xi} + \frac{\partial\xi}{\partial y}\frac{\partial\eta}{\partial y}\left[\frac{\partial\phi_i}{\partial\xi}\frac{\partial\phi_j}{\partial\eta} + \frac{\partial\phi_i}{\partial\eta}\frac{\partial\phi_j}{\partial\xi}\right] + \left(\frac{\partial\eta}{\partial y}\right)^2 \frac{\partial\phi_i}{\partial\eta}\frac{\partial\phi_j}{\partial\eta}$$

we reduce the calculus to the integration of

$$\int_{T^*} \frac{\partial\phi_i}{\partial\xi}\frac{\partial\phi_j}{\partial\xi} d\xi d\eta$$
$$\int_{T^*} \left[\frac{\partial\phi_i}{\partial\xi}\frac{\partial\phi_j}{\partial\eta} + \frac{\partial\phi_i}{\partial\eta}\frac{\partial\phi_j}{\partial\xi}\right] d\xi d\eta$$
$$\int_{T^*} \frac{\partial\phi_i}{\partial\eta}\frac{\partial\phi_j}{\partial\eta} d\xi d\eta,$$

due to the fact that all the components of the Jacobian matrix and its inverse are constants in the element. With straightforward computations we find that the three terms are all zero for $i = 1, 2, 3$ and $j = 4$ or vice versa and all $\frac{81}{20}$ if $i = j = 4$. In this way we have

$$a_{i,j}^{(T)} = \begin{cases} Area(T)(\nabla_{\xi,\eta}\phi_i^* Jac^{-1}) \cdot (\nabla_{\xi,\eta}\phi_j^* Jac^{-1}) & i,j \leq 3 \\ 0 & i = 1,2,3 \; j = 4 \; or \; i = 4, \; j = 1,2,3 \\ 2Area(T)\frac{81}{20}\left[\left(\frac{\partial\xi}{\partial x}\right)^2 + \frac{\partial\xi}{\partial x}\frac{\partial\eta}{\partial x} + \left(\frac{\partial\eta}{\partial x}\right)^2 + \left(\frac{\partial\xi}{\partial y}\right)^2 + \frac{\partial\xi}{\partial y}\frac{\partial\eta}{\partial y} + \left(\frac{\partial\eta}{\partial y}\right)^2\right] & i = j = 4 \end{cases} \tag{37}$$

With the same approach we derive the formulation for the divergence matrix

$$b_{ki}^{(T)} = -2 * Area(T) \int_{T^*} \phi_k^* \nabla_{\xi,\eta} \cdot w_i^* Jac^{-1} d\xi d\eta.$$

If $i \leq 3$ the gradient is constant and we can bring the term outside the integral,

$$b_{ki}^{(T)} = -2 * Area(T) \nabla_{\xi,\eta} \cdot Jac^{-1} w_i^* \int_{T^*} \phi_k^* d\xi d\eta = -2 * Area(T) \nabla_{\xi,\eta} \cdot Jac^{-1} w_i^* \frac{Area(T^*)}{3} = -\frac{Area(T)}{3} \nabla_{\xi,\eta} \cdot Jac^{-1} w_i^*$$

whereas if $i = 4$ we have to compute by hands the three possible scenarios for $k = 1, 2, 3$. We have for $i = 1, ..., N$

$$\begin{aligned}\int_{T^*} \phi_1^* \left(\frac{\partial \xi}{\partial x}\frac{\partial \phi_4}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial \phi_4}{\partial \eta} \right) d\xi d\eta =& \frac{\partial \xi}{\partial x} \int_{T^*} 27\left[\eta(1-\xi-\eta) - \xi\eta\right](1-\xi-\eta) d\xi d\eta \\ &+ \frac{\partial \eta}{\partial x} \int_{T^*} 27\left[\xi(1-\xi-\eta) - \xi\eta\right](1-\xi-\eta) d\xi d\eta \\ =& \frac{9}{40}\left(\frac{\partial \xi}{\partial x} + \frac{\partial \eta}{\partial x}\right),\end{aligned}$$

and similarly

$$\begin{aligned}\int_{T^*} \phi_2^* \left(\frac{\partial \xi}{\partial x}\frac{\partial \phi_4}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial \phi_4}{\partial \eta} \right) d\xi d\eta &= -\frac{9}{40}\frac{\partial \xi}{\partial x} + 0\frac{\partial \eta}{\partial x} \\ \int_{T^*} \phi_3^* \left(\frac{\partial \xi}{\partial x}\frac{\partial \phi_4}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial \phi_4}{\partial \eta} \right) d\xi d\eta &= 0\frac{\partial \xi}{\partial x} - \frac{9}{40}\frac{\partial \eta}{\partial x}.\end{aligned}$$

Finally

$$b_{1_{ij}}^{(T)} = \begin{cases} -\dfrac{Area(T)}{3}\left(\dfrac{\partial \phi_j}{\partial \xi}\dfrac{\partial \xi}{\partial x} + \dfrac{\partial \phi_j}{\partial \eta}\dfrac{\partial \eta}{\partial x}\right) & i \leq 3,\ k = 1,2,3 \\ -2Area(T)\dfrac{9}{40}\left(\dfrac{\partial \xi}{\partial x} + \dfrac{\partial \eta}{\partial x}\right) & i = 4\ k = 1 \\ 2Area(T)\dfrac{9}{40}\dfrac{\partial \xi}{\partial x} & i = 4\ k = 2 \\ 2Area(T)\dfrac{9}{40}\dfrac{\partial \eta}{\partial x} & i = 4\ k = 3 \end{cases}$$

$$b_{2_{ij}}^{(T)} = \begin{cases} -\dfrac{Area(T)}{3}\left(\dfrac{\partial \phi_j}{\partial \xi}\dfrac{\partial \xi}{\partial y} + \dfrac{\partial \phi_j}{\partial \eta}\dfrac{\partial \eta}{\partial y}\right) & i \leq 3,\ k = 1,2,3 \\ -2Area(T)\dfrac{9}{40}\left(\dfrac{\partial \xi}{\partial y} + \dfrac{\partial \eta}{\partial y}\right) & i = 4\ k = 1 \\ 2Area(T)\dfrac{9}{40}\dfrac{\partial \xi}{\partial y} & i = 4\ k = 2 \\ 2Area(T)\dfrac{9}{40}\dfrac{\partial \eta}{\partial y} & i = 4\ k = 3. \end{cases}$$

6.2 Code

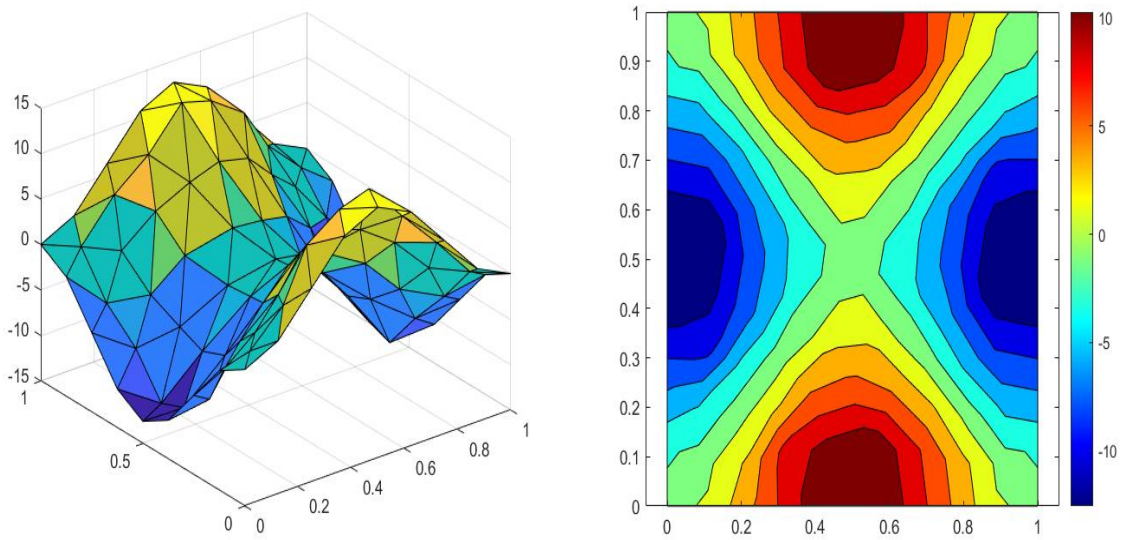
Here the code for the construction of the matrices of the linear system.

```
%--------------------------------------------------------------------
        %% BUILD THE SYSTEM MATRIX
        function [M, A, B1, B2] = BuildSystem(this)
            %----------------------------------------------------------------
            Nelem  = this.V.Nelem;
            Nodes  = this.V.Nodes;
            triang = this.V.triang;
            Bloc   = this.V.Bloc;
            Cloc   = this.V.Cloc;
            Area   = this.V.Area;
            Jac    = this.J;
            %----------------------------------------------------------------
            % initialize vectors for sparse allocation
            i_sparse = zeros(Nelem*16,1);i_sparse_B=zeros(Nelem*12,1);
            j_sparse = zeros(Nelem*16,1);j_sparse_B=zeros(Nelem*12,1);
            H_val    = zeros(Nelem*16,1);
            M_val    = zeros(Nelem*16,1);
```

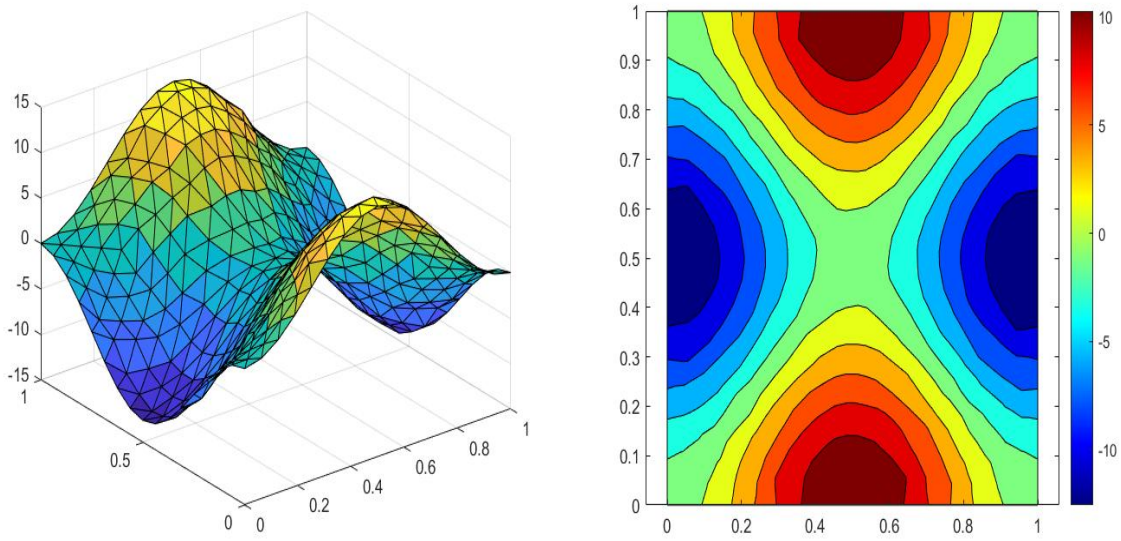
```
18              B1_val   = zeros(Nelem*12,1);
19              B2_val   = zeros(Nelem*12,1);
20              %-------------------------------------------------------------
21              it=1; it_B=1;
22              bubble = Nodes*ones(Nelem,1)+(1:Nelem)';
23              ELEM   = [triang,bubble];
24              for iel=1:Nelem
25                  %---------------------------------------------------------
26                  for iloc=1:4
27                      %-----------------------------------------------------
28                      iglob=ELEM(iel,iloc);
29                      for jloc=1:4
30                          %-------------------------------------------------
31                          jglob        = ELEM(iel,jloc);
32                          i_sparse(it) = iglob;
33                          j_sparse(it) = jglob;
34                          %-------------------------------------------------
35                          if iloc<4 && jloc<4 %Standard case
36                              H_val(it) = ([Bloc(iloc), Cloc(iloc)]*Jac(:,:,iel))*...
37                                          ([Bloc(jloc), Cloc(jloc)]*Jac(:,:,iel))'*Area(iel);
38                              %---------------------------------------------
39                              if iloc==jloc
40                                  M_val(it) = Area(iel)/6;
41                              else
42                                  M_val(it) = Area(iel)/12;
43                              end
44                              %---------------------------------------------
45                              i_sparse_B(it_B)=iglob;
46                              j_sparse_B(it_B)=jglob;
47                              %---------------------------------------------
48                              B1_val(it_B)=-1/3*Area(iel)*[Bloc(jloc),Cloc(jloc)]*Jac(:,1,iel);
49                              B2_val(it_B)=-1/3*Area(iel)*[Bloc(jloc),Cloc(jloc)]*Jac(:,2,iel);
50                              %---------------------------------------------
51                              it_B=it_B+1;
52              %                  BUBBLE
53                          %-------------------------------------------------
54                          elseif (iloc==1 && jloc==4) || (iloc==4 && jloc==1)
55                              H_val(it)=0+0+0;
56                              M_val(it)=3/40*2*Area(iel);
57                          %-------------------------------------------------
58                          elseif (iloc==2 && jloc==4) || (iloc==4 && jloc==2)
59                              H_val(it)=0+0+0;
60                              M_val(it)= 3/40*2*Area(iel);
61                          %-------------------------------------------------
62                          elseif (iloc==3 && jloc==4) || (iloc==4 && jloc==3)
63                              H_val(it)=0+0+0;
64                              M_val(it)=3/40*2*Area(iel);
65                          %-------------------------------------------------
66                          elseif (iloc==4 && jloc==4)
67                              H_val(it)=(81/20*Jac(1,1,iel)^2+Jac(1,1,iel)*Jac(2,1,iel)*...
68                              (81/40+81/40)+81/20*Jac(2,1,iel)^2+81/20*Jac(1,2,iel)^2+...
69                              Jac(1,2,iel)*Jac(2,2,iel)*(81/40+81/40)+Jac(2,2,iel)^2*81/20)*2*Area(iel);
70                              M_val(it)=81/560*2*Area(iel);
71                          %-------------------------------------------------
72                          end
73                          it=it+1;
74                          if iloc==4
75                               continue
76                          elseif jloc==4
77                              %---------------------------------------------
78                               i_sparse_B(it_B)=iglob;
79                               j_sparse_B(it_B)=jglob;
80                               %---------------------------------------------
81                               if     iloc==1
82                                   B1_val(it_B)=(-9/40*Jac(1,1,iel)-9/40*Jac(2,1,iel))*2*Area(iel);
83                                   B2_val(it_B)=(-9/40*Jac(1,2,iel)-9/40*Jac(2,2,iel))*2*Area(iel);
84                               elseif iloc==2
85                                   B1_val(it_B)=(9/40*Jac(1,1,iel)+0)*2*Area(iel);
86                                   B2_val(it_B)=(9/40*Jac(1,2,iel)+0)*2*Area(iel);
87                               elseif iloc==3
88                                   B1_val(it_B)=(0+9/40*Jac(2,1,iel))*2*Area(iel);
```

```
89                                  B2_val(it_B)=(9/40*Jac(2,2,iel)+0)*2*Area(iel);
90                              end
91                              %-----------------------------------------
92                              it_B=it_B+1;
93                          end
94                          %-------------------------------------------------
95                      end % jloc
96                  end % iloc
97              end % Nelem
98              %-------------------------------------------------------------
99              A  = sparse(i_sparse,j_sparse,H_val);
100             M  = this.rho*sparse(i_sparse,j_sparse,M_val);
101             B1 = sparse(i_sparse_B,j_sparse_B,B1_val);
102             B2 = sparse(i_sparse_B,j_sparse_B,B2_val);
103             %-------------------------------------------------------------
104         end
```

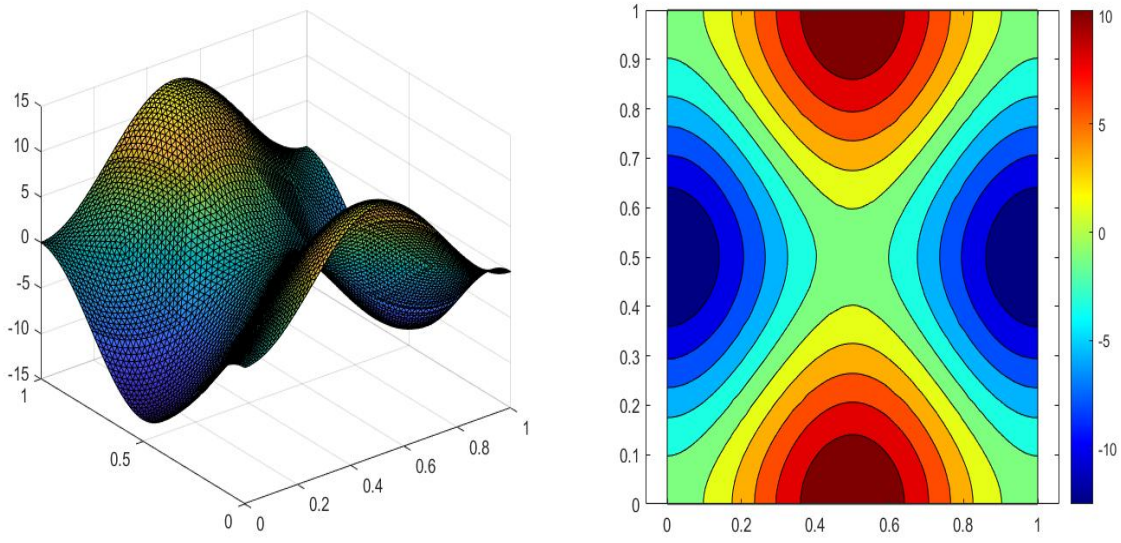
6.3 Some results



Approximated pressure state with "mesh1". 3D plot (left), isosurface (right).



Approximated pressure state with "mesh2". 3D plot (left), isosurface (right).



Approximated pressure state with "mesh4". 3D plot (left), isosurface (right).

7 Results

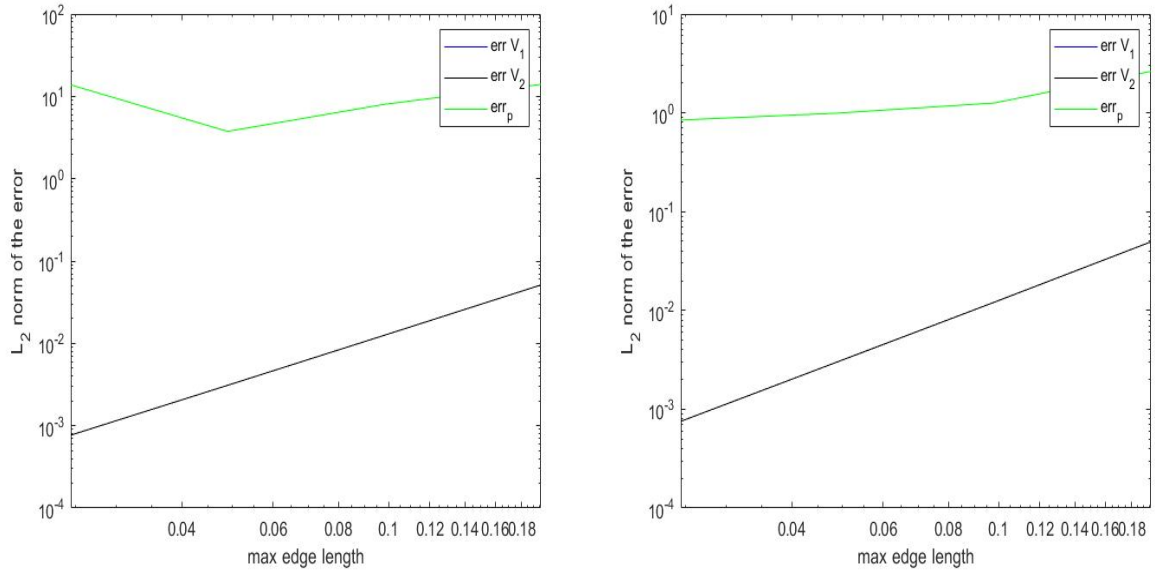
In this section we are going to compare the different methods in terms of convergence properties and computational effort required for each mesh. For this reason also manufactured cases with uniform mesh of $N \times N$ nodal points will be taken into consideration. Another remarkable point is that for the given problem (example 1.0.2 in the assignment paper) we can choose if impose or not the Dirichlet boundary conditions in the system, since they are not in principle required and form an over constrained problem. The considered errors have been computed as the norm $||u-u_h||_{L^2(\Omega)}$ using Trapezi's rule on each triangle.

```
%--------------------------------------------------------------------
      %% ERROR EVALUATION
      %--------------------------------------------------------------------
      function err=evalErr(fh, freal, meshStruct)
          %----------------
          % using the trapezoidal rule for evaluating the integral
          % (it is exact on linear functions)
          % has a convergence error compatible with linear P1 FEM
          % Trapezoidal is used here exploiting the fact that
          % in one of the nodes the phi is zero!!! (only 1 term)
          %---------------
          Area  = meshStruct.Area;
          triang = meshStruct.triang;
          err=0;
          %------
          if isa(freal,'function_handle')
              coord = meshStruct.coord;
              for iel=1:meshStruct.Nelem
                  for iloc=1:3
                      iglob=triang(iel,iloc);
                      err=err+...
                          (fh(iglob)-freal(coord(iglob,1),coord(iglob,2)))^2*Area(iel)/3;
                  end
              end
          else
              for iel=1:meshStruct.Nelem
                  for iloc=1:3
                      iglob=triang(iel,iloc);
                      err=err+...
                          (fh(iglob)-freal(iglob))^2*Area(iel)/3;
                  end
              end
          end
          %-----
          err=sqrt(err);
          %-------------
       end
```

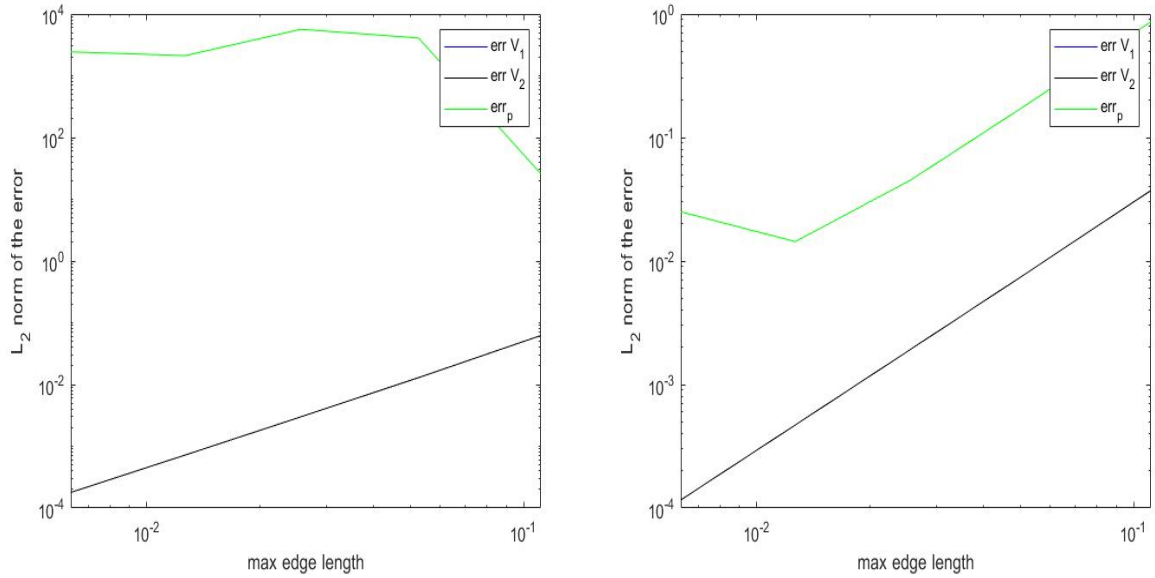
7.1 Convergence velocities

P1/P1 not stabilized

In some way the imposition of the pressure values at the boundary stabilizes a little the pressure solution even for the standard P1/P1 space, but still the method shows oscillations in the convergence plot and decreasing the mesh size the error on the pressure enlarge. We see that the worst case scenario is with uniform meshes and no Dirichlet BCs on the pressure, where the L_2 errors reach very high values.



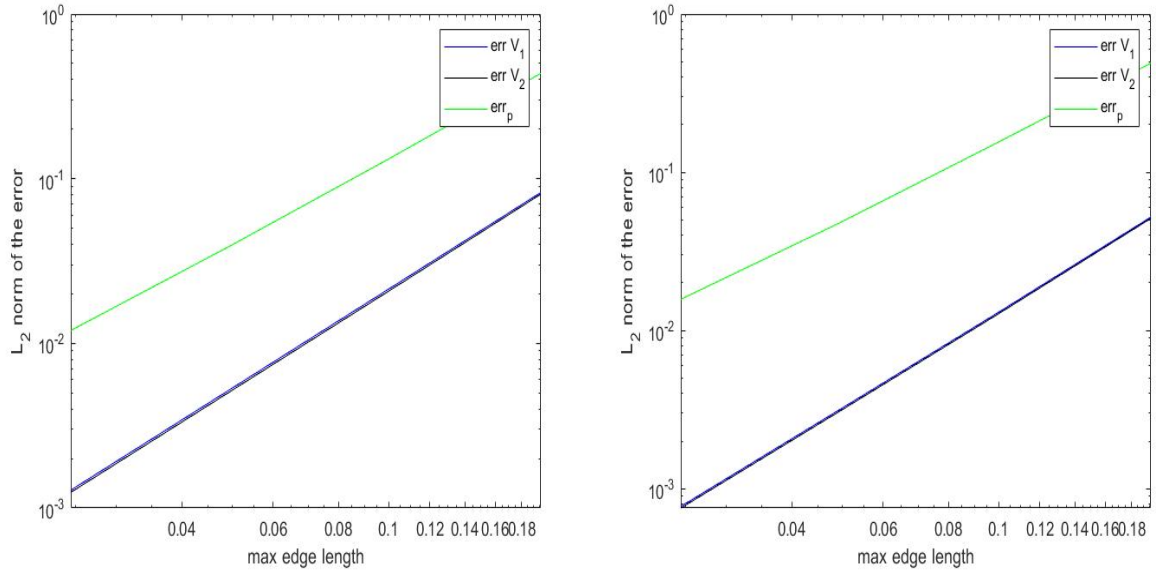
Provided non uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left) and with them (right).



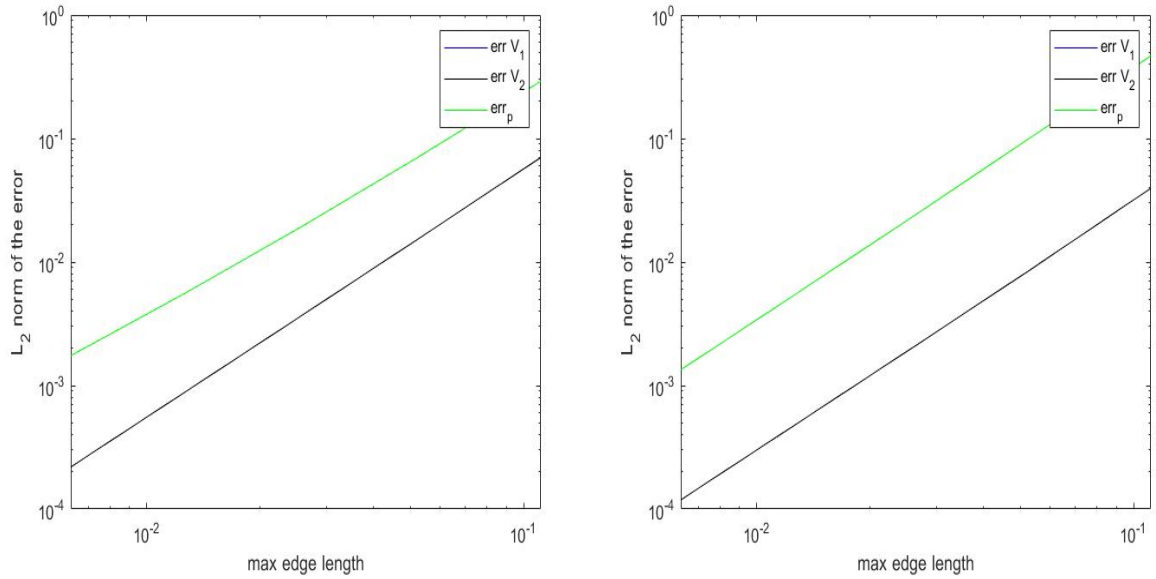
Uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left) and with them (right).

P1/P1 stabilized

In this case the method is clearly convergent in all the scenarios with or without pressure Dirichlet BCs and with or without uniform meshes. The convergence velocities are almost quadratic in each unknown, and some tests show that the more we increase the value of the stabilization factor (up to reasonable values) the more the convergence velocity on the pressure tends to increase and the ones for the velocity components tend to decrease, but always with values close the quadratic convergence (say $V = 1.7 \div 2.1$).

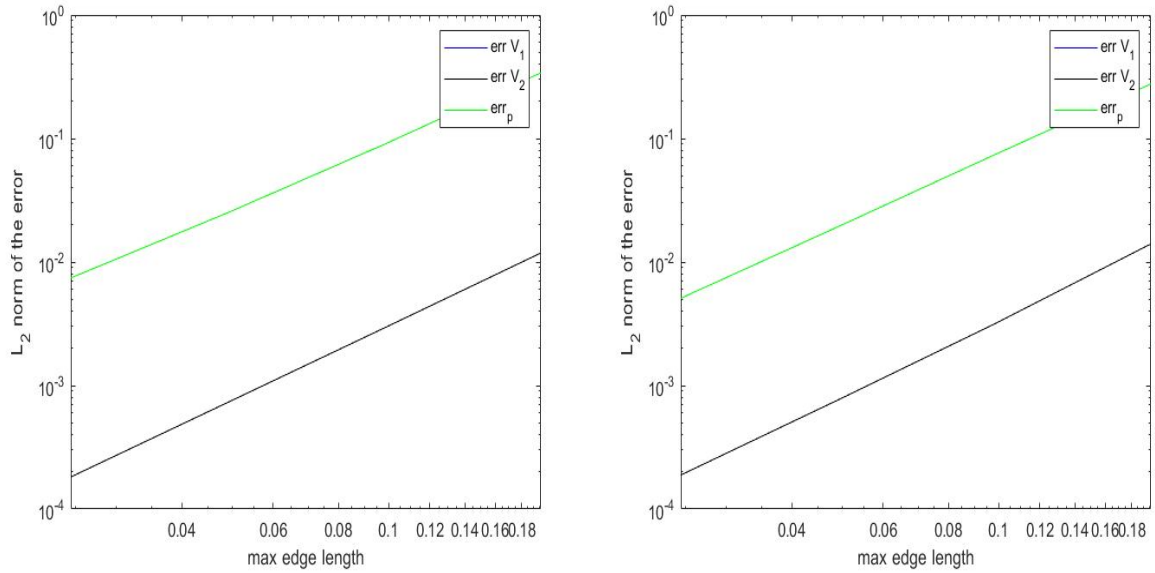


Provided non uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left) and $\delta = 0.08$*, and with them (right) and* $\delta = 0.01$*. Experimental convergence velocities are* $\begin{bmatrix} V_p \sim 1.75 \\ V_{v1} \sim 2.00 \\ V_{v2} \sim 2.00 \end{bmatrix}$ *for no pressure Dir BCs and* $\begin{bmatrix} V_p \sim 1.7 \\ V_{v1} \sim 2.03 \\ V_{v2} \sim 2.03 \end{bmatrix}$ *in the other case.*



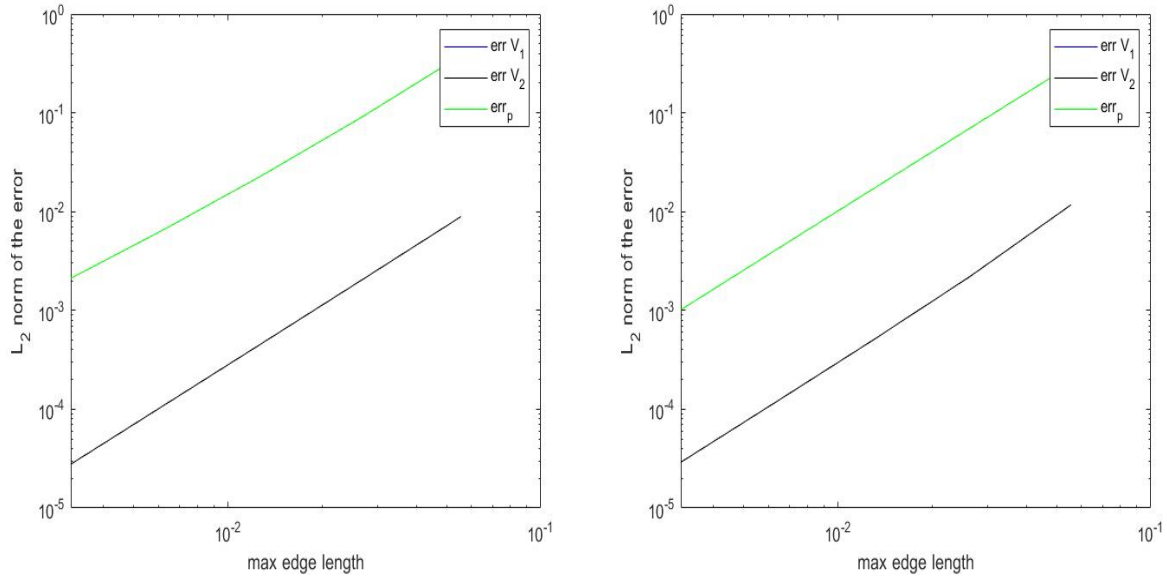
Uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left) and $\delta = 0.08$*, and with them (right) and* $\delta = 0.01$*. Experimental convergence velocities are* $\begin{bmatrix} V_p \sim 1.83 \\ V_{v1} \sim 2.01 \\ V_{v2} \sim 2.01 \end{bmatrix}$ *for no pressure Dir BCs and* $\begin{bmatrix} V_p \sim 2.05 \\ V_{v1} \sim 2.05 \\ V_{v2} \sim 2.05 \end{bmatrix}$ *in the other case.*

P1 - iso P2/P1



Provided non uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left), and with them (right). Experimental convergence velocities are $\begin{bmatrix} V_p \sim 1.86 \\ V_{v1} \sim 2.01 \\ V_{v2} \sim 2.01 \end{bmatrix}$ *for no pressure Dir BCs and* $\begin{bmatrix} V_p \sim 1.92 \\ V_{v1} \sim 2.1 \\ V_{v2} \sim 2.1 \end{bmatrix}$ *in the other case.*

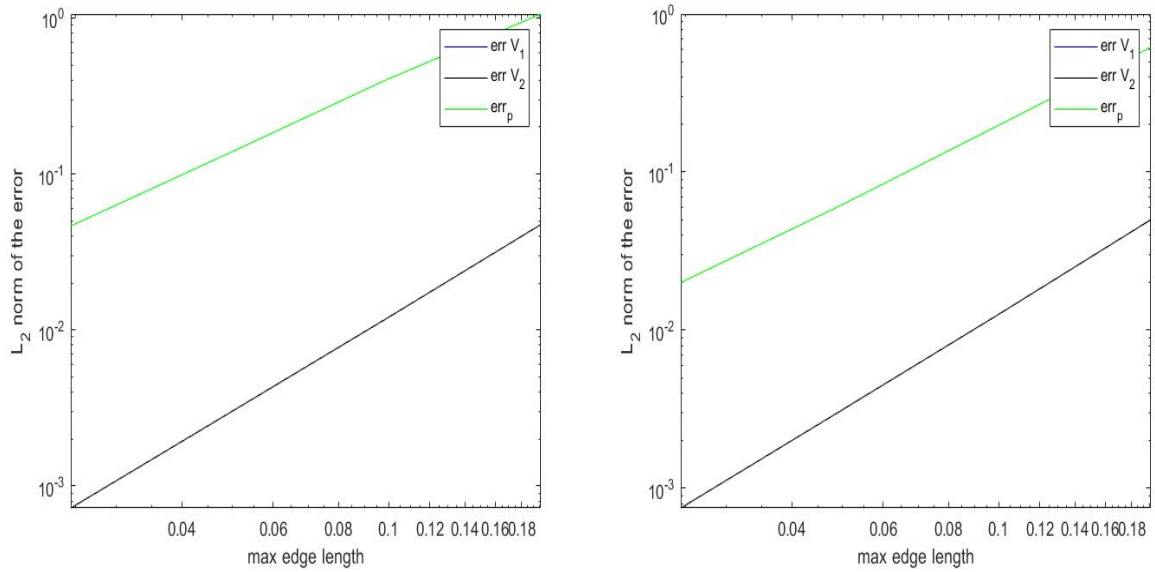
Also in this case the method seem to be stable for all the scenarios with quadratic convergence in each unknown, but the absolute value of all the errors is almost half of one order of magnitude below the ones obtained with the $P1/P1$ stabilized for the same discretizations.



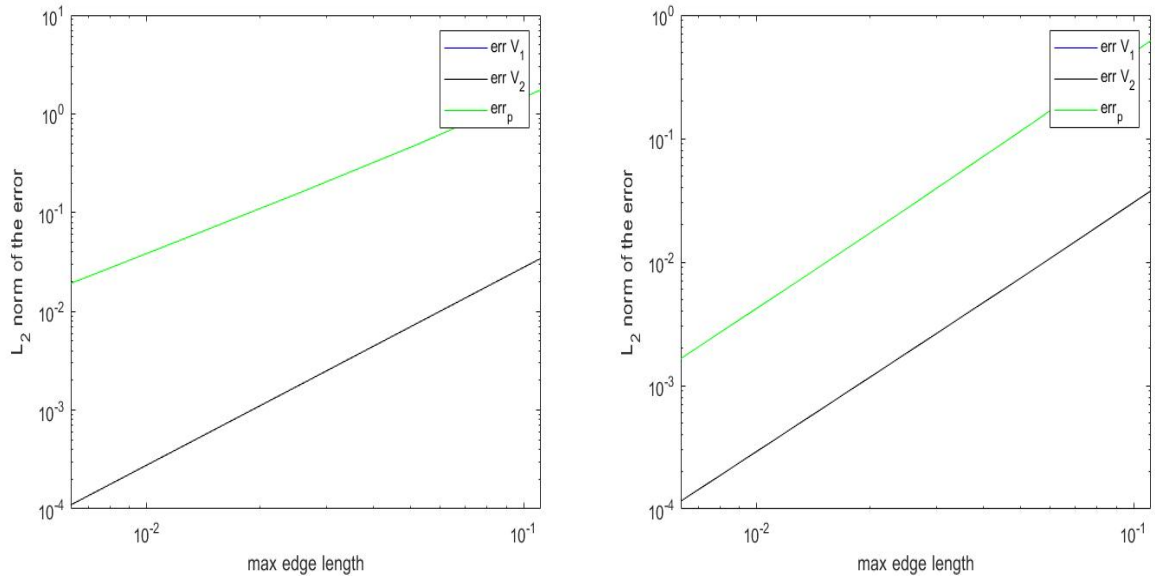
Uniform meshes. Provided non uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left), and with them (right). Experimental convergence velocities are $\begin{bmatrix} V_p \sim 1.9 \\ V_{v1} \sim 2.01 \\ V_{v2} \sim 2.01 \end{bmatrix}$ *for no pressure Dir BCs and* $\begin{bmatrix} V_p \sim 1.98 \\ V_{v1} \sim 2.15 \\ V_{v2} \sim 2.15 \end{bmatrix}$ *in the other case.*

P1 - bubble P1

Also in this case the method is stable for all scenarios, but we have some changes in the pressure velocity convergent between them. First of all it seems that the addition of the Dirichlet BCs has a non indifferent effect in terms of improvement in the convergence, and in general the uniform discretizations seem to have a better performance than the provided case with triangles of various size.



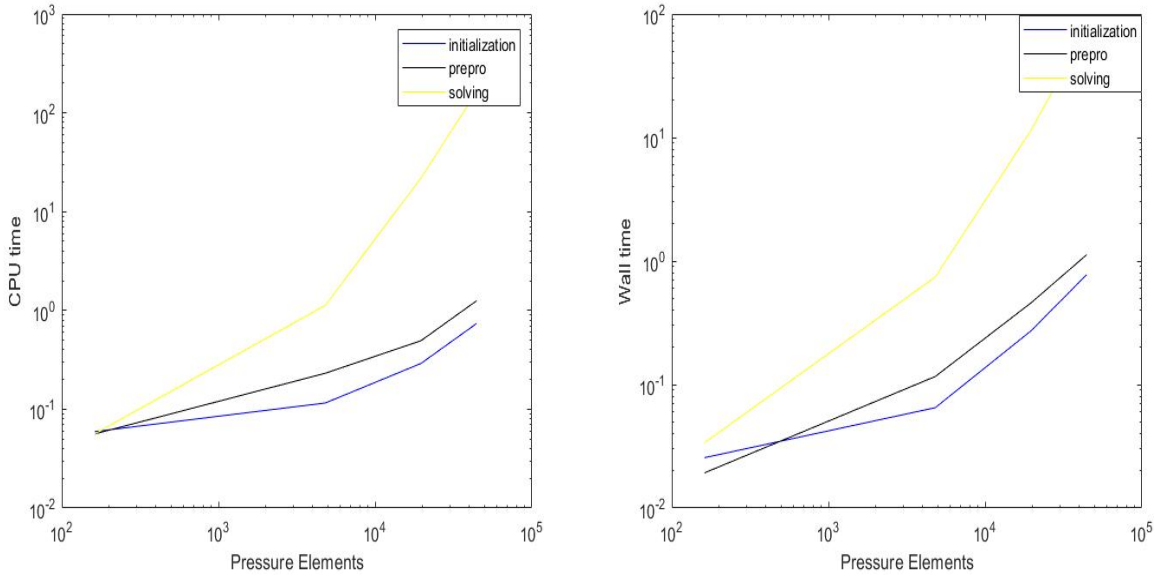
Provided non uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left), and with them (right). Experimental convergence velocities are $\begin{bmatrix} V_p \sim 1.45 \\ V_{v1} \sim 2.00 \\ V_{v2} \sim 2.00 \end{bmatrix}$ *for no pressure Dir BCs and* $\begin{bmatrix} V_p \sim 1.67 \\ V_{v1} \sim 2.01 \\ V_{v2} \sim 2.01 \end{bmatrix}$ *in the other case.*



Uniform meshes. Provided non uniform meshes. Convergence velocities with no Dirichlet BCs on the pressure(left), and with them (right). Experimental convergence velocities are $\begin{bmatrix} V_p \sim 1.61 \\ V_{v1} \sim 2.00 \\ V_{v2} \sim 2.00 \end{bmatrix}$ *for no pressure Dir BCs and* $\begin{bmatrix} V_p \sim 2.1 \\ V_{v1} \sim 2.01 \\ V_{v2} \sim 2.01 \end{bmatrix}$ *in the other case.*

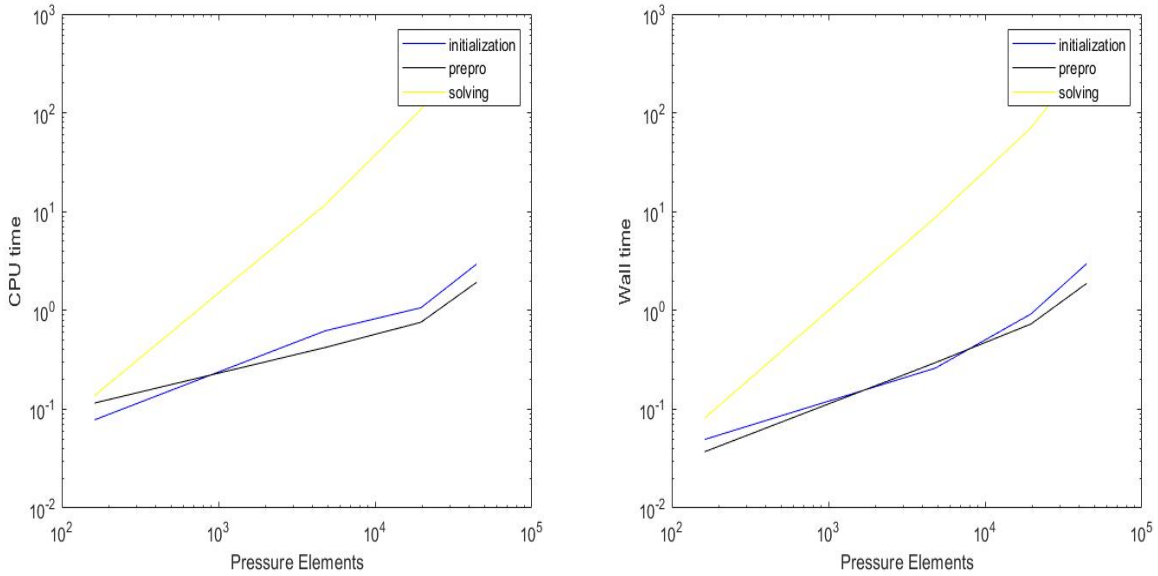
7.2 Computational time comparison

Here we report the result in terms of wall clock and CPU times required for each method for initialization (load of the mesh and creation of the main parameters), preprocessing (assembling of the matrices, imposition of the boundary condition) and solving of the system. Actually these results may depend on some random behaviour in the CPU processor, and to get a result free from random extreme good or bad cases it's usually necessary to take a mean of the time for several simulations. In this case we only take a mean over 5 simulations for each number of points in the discretization. The solution of the linear system is computed using the Matlab command "mldivide" [1] with automatically chooses the best numerical method for such resolution depending on the matrix features (like symmetry, tridiagonality...)

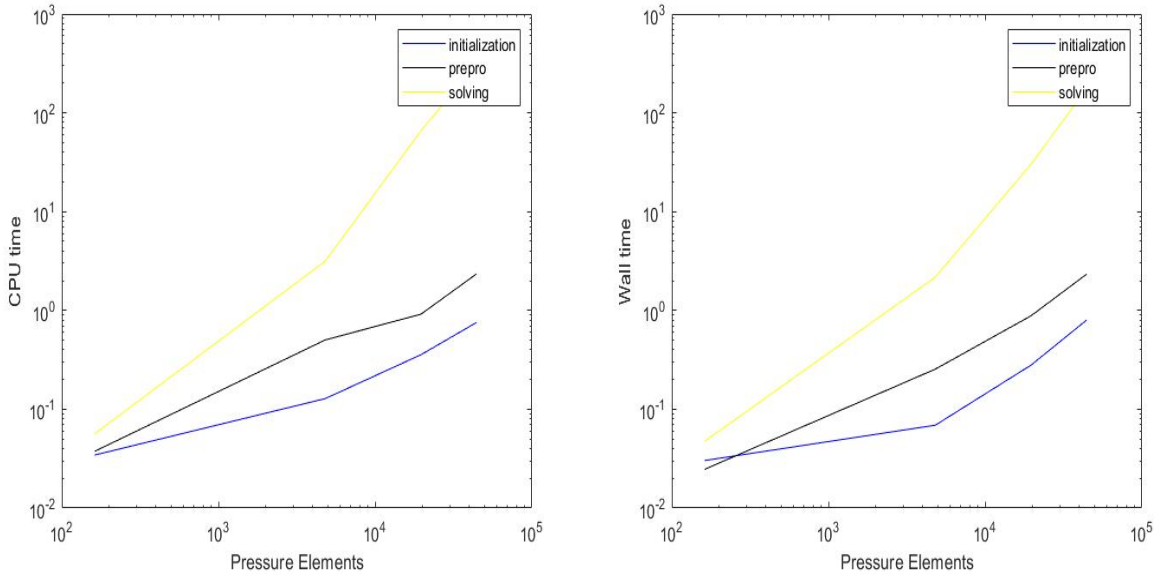


CPU time (left) and Wall clock time (right) for the P1/P1 stabilized method over the number of elements in a uniform mesh

Clearly the higher CPU and Wall clock times for the solver are found in the P1-iso P2/P1 case, since it is the formulation with the highest number of degrees of freedom (and thus largest matrix system) of the three starting from the same discretization for the pressure. On the other hand the best method in terms of time effort is evidently the P1/P1 stabilized, given the simple formulation that doesn't require any manipulation of the provided mesh and has the smallest number of nodes of the three.



CPU time (left) and Wall clock time (right) for the P1 - iso P2/P1 method over the number of elements in a uniform mesh



CPU time (left) and Wall clock time (right) for the P1 - bubble P1 method over the number of elements in a uniform mesh

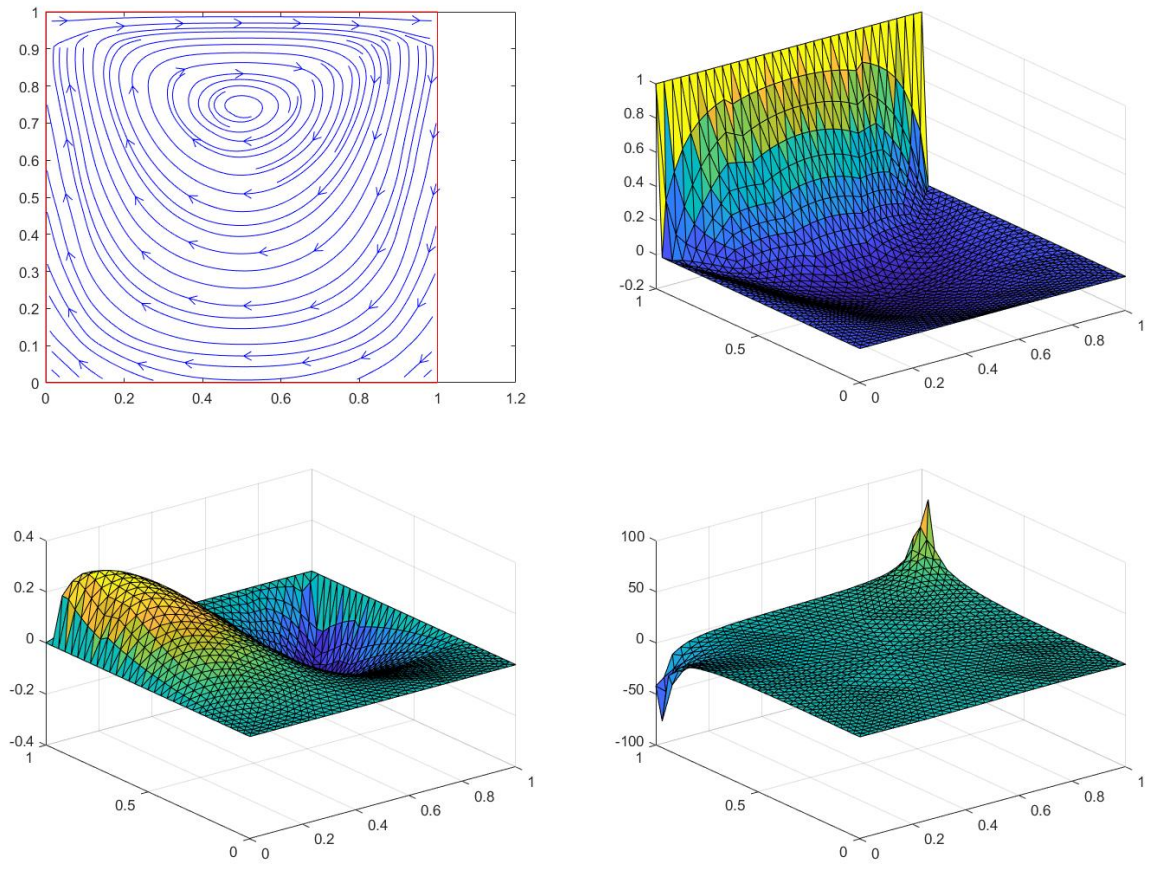
7.3 Lid-Driven Cavity problem

In this case we just fix $f_1 = f_2 = g = 0$ in the main code showed above, and set

```
1    Stokes = Stokes.setDirBC(1, [0.0; 1.0],1.0,[1,0]);
```

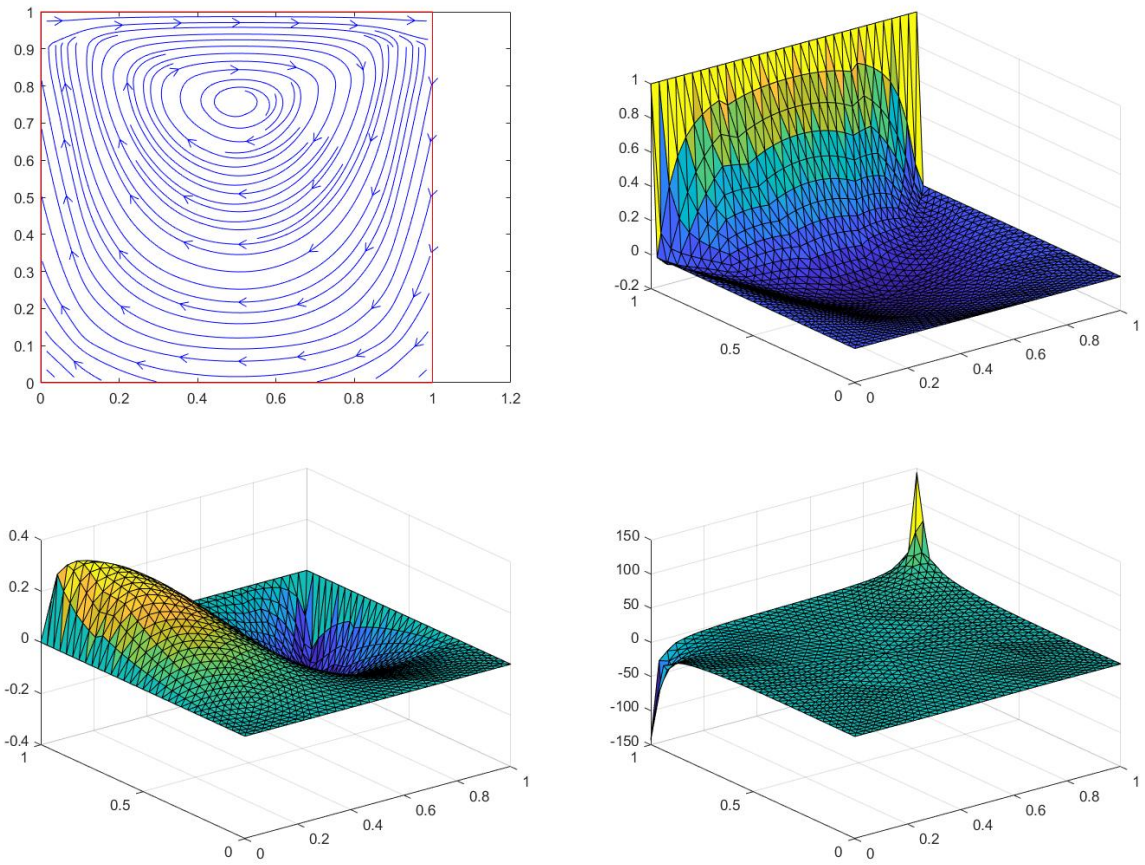
for the only non zero boundary conditions. The problem asks now for a stationary solution, so we can either set a huge time step like $\Delta t = 1e10$ or use the stationary solver

```
1    [uh1, uh2, ph] = Stokes.DirectStatSolver(flag_BC);
```

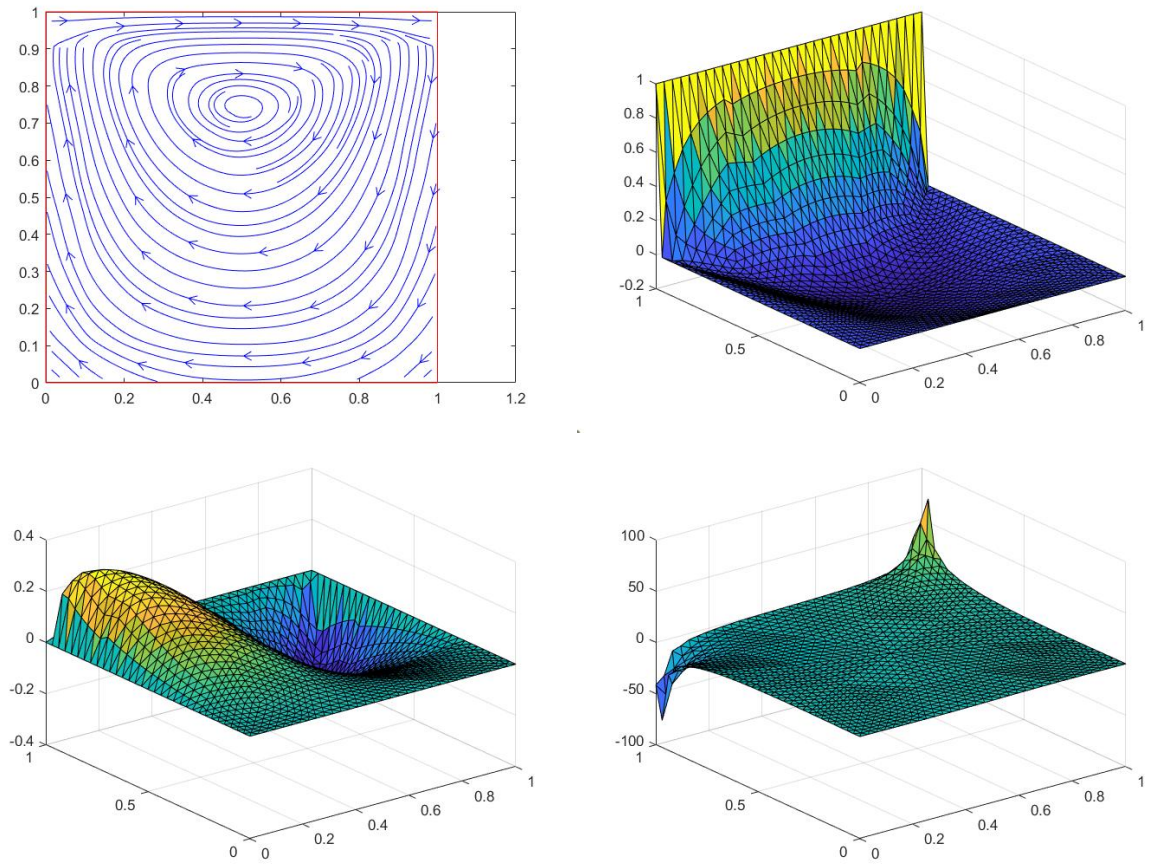


Streamlines, u_1, u_2 and pressure results with "mesh3" and P1/P1 stabilized

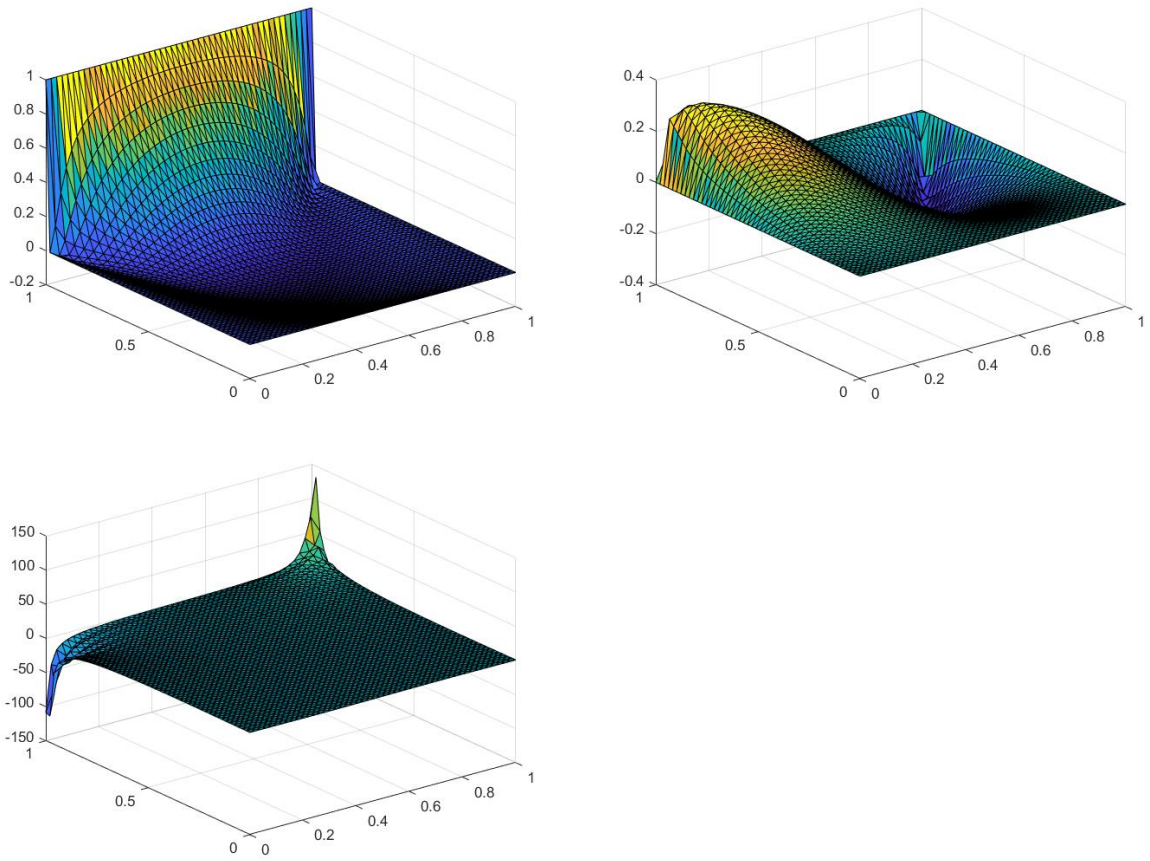
Some oscillations seems to appear in the u_1 graphs, but they are probably related to the non uniform mesh. In fact choosing for example a uniform mesh of 50×50 points we have figure (35).



Streamlines, u_1, u_2 and pressure results with "mesh3" and P1 - iso P2/P1



Streamlines, u_1, u_2 and pressure results with "mesh3" and P1 - bubble P1



u_1, u_2 and pressure results with uniform mesh of 50×50 nodes and P1 - bubble P1

7.3.1 Time-dependent Lid Driven

We could also check the results for a time-dependent version of the Lid Driven cavity problem,

$$\rho\frac{\partial u}{\partial t} - \mu\Delta u + \nabla p = g \qquad in\ \Omega \times [0,T], \tag{38}$$

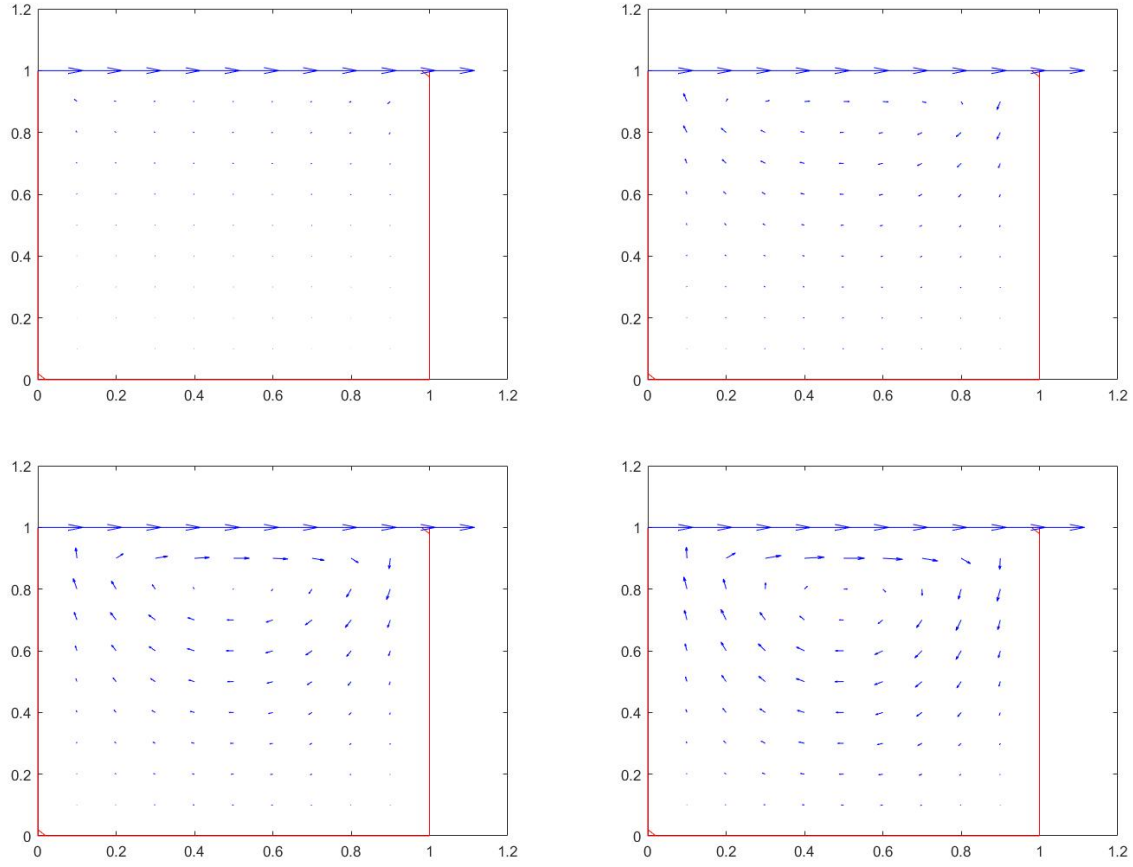
$$div(u) = g \qquad in\ \Omega \times [0,T], \tag{39}$$

$$u = (1,0)^T \qquad in\ \Gamma_1 = \{x : x_2 = 1\} \times [0,T], \tag{40}$$

$$u = (0,0)^T \qquad in\ \Gamma \setminus \Gamma_1 \times [0,T] \tag{41}$$

$$u = (0,0)^T \qquad in\ \Omega \times \{0\} \tag{42}$$

For $\rho = \mu = 1$, $g = 0$ in the domain, we find figure(36).



Velocity vector field for time dependent lid driven cavity problem at time $t = 0.0005$ (upper left), $t = 0.005$ (upper right), $t = 0.02$ (down left), $t = 1$ (down right).

7.4 Final considerations

All the analyzed stable methods, P1/P1 stabilized, P1 - iso P2/P1 and P1 - bubble P1, have shown to provide oscillations free results and good convergence properties in the chosen benchmarks. Of all three the best in terms of precision of the numerical solutions seems to be the P1 - iso P2/P1, even if it requires computational times slightly higher than the bubble method, and considerably higher than the P1/P1

stabilized algorithm.
In next chapter we will briefly cover the derivation of the weak and Galerkin formulation for the complete Navier Stokes problem for an incompressible fluid, and see some comparisons with the simple Stokes solver.

8 Extra: comparison with NS Solver

8.1 Navier Stokes: Weak and Galerkin formulation

With the same procedure used for the Stokes problem, we can derive the weak formulation for the time dependent Navier Stokes problem for an incompressible fluid, defined by the following system,

$$\begin{cases} \partial_t u + (u \cdot \nabla)u + \frac{1}{\rho}\nabla p - \nu \Delta u = f & in\ \Omega \times [0,T] \\ \nabla \cdot u = 0 & in\ \Omega \times [0,T] \\ u(r) = u_D(r), & on\ \Sigma_D \times [0,T] \\ [\frac{p}{\rho} I - \nu \nabla u] \cdot n = g, & on\ \Sigma_N \times [0,T] \\ u = u_0 & in\ \Omega \times \{0\}. \end{cases} \tag{43}$$

Now, multiplying all members by appropriate test functions, integrating over the domain Ω and apply Green's theorem, we end up with

$$\int_\Omega (\partial_t u) \cdot v dr + \int_\Omega [u \cdot \nabla u] \cdot v dr + \int_\Omega \nu \nabla u : \nabla v dr - \frac{1}{\rho}\int_\Omega \nabla \cdot v p dr = \int_\Omega f \cdot v dr \qquad \forall v \in \mathcal{V}, \tag{44}$$

$$\int_\Omega q \nabla \cdot u = 0 \qquad \forall q \in \mathcal{Q}, \tag{45}$$

where $\mathcal{V}$ and $\mathcal{Q}$ are the same used above and we implicitly assumed only Dirichlet boundary conditions. Using a short notation

$$\begin{aligned} (\partial_t u, v) + n(u,u,v) + a(u,v) + b(v,p) = F(v) \quad \forall v \in \mathcal{V}, \\ b(u,q) = 0 \quad \forall q \in \mathcal{Q}, \end{aligned} \tag{46}$$

with

$$(v,w) = \int_\Omega vw dr \qquad n(v,w,z) = \int_\Omega (v \cdot \nabla w) \cdot z dr$$

$$a(v,w) = \mu \int_\Omega \nabla v : \nabla w dr \qquad b(v,q) = -\int_\Omega q \nabla \cdot v dr$$

$$F(v) = \int_\Omega f \cdot v dr.$$

Note that the "tri-linear" form $n(\cdot,\cdot,\cdot)$ contains the nonlinear convective term, but here is considered for simplicity as a tri-linear form. All these linear, bi-linear and tri-linear forms are bounded and thus continuous, but in this case, even if it can be proved that a solution exists, uniqueness is completely proved only for two dimensional cases.
The Galerkin discretized formulation, using backward Euler time-stepping, is

$$(u_h^{n+1}, v) + \Delta t_n [n(u_h^{n+1}, u_h^{n+1}, v) + a(u_h^{n+1}, v) + b(v, p_h^{n+1})] = (u_h^n, v) + \Delta t_n (f(t_{n+1}, v) \qquad \forall v \in \mathcal{V}_h, \tag{47}$$

$$b(u_h^{n+1}, q) = 0 \qquad \forall q \in \mathcal{Q}_h. \tag{48}$$

As for the Stokes problem, we have

$$\begin{cases} u_h(r,t) = \sum_{j=1}^{N} u_j(t) w_j(r), \\ p_h(r,t) = \sum_{k=1}^{M} p_k(t) J_k(r). \end{cases}$$

In matrix form, we get

$$\begin{bmatrix} (\frac{M}{\Delta t} + N(u_h^{n+1}) + K), & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u^{n+1} \\ p^{n+1} \end{bmatrix} = \begin{bmatrix} \frac{1}{\Delta t} M u^n + f \\ 0 \end{bmatrix}, \tag{49}$$

where

$$\begin{aligned}
M = \{m_{ij}\} \qquad & m_{ij} = (w_i, w_j) = \int_\Omega w_i \cdot w_j dr && i,j = 1,...,N; \\
N(u_h) = \{n_{ij}(u_h)\} \quad & n_{ij}(u_h) = n(u_h, w_i, w_j) = \int_\Omega [(u_h \cdot \nabla) w_i] \cdot w_j dr && i,j = 1,...,N; \\
K = \{k_{ij}\} \qquad & k_{ij} = a(w_i, w_j) = \nu \int_\Omega \nabla w_i : \nabla w_j dr && i,j = 1,...,N; \\
B = \{b_{ki}\} \qquad & b_{ki} = b(w_i, J_k) = -\frac{1}{\rho} \int_\Omega \phi_k \nabla \cdot w_i dr && k = 1,...,M; i = 1,...,N; \\
f = \{f_i\} \qquad & f_i = \int_\Omega f \cdot w_i dr && i = 1,...,N.
\end{aligned}$$

Since this formulation include the same problem of the Stokes equations for the coupling of the pressure and velocity discretization spaces, we choose to use the $P_1 - iso - P_1/P_2$ elements among the three methods sdstudied.
The main concern in this case is the treatment of the Convection term $n(u_h^{n+1}, u_h^{n+1}, v)$. This is a standard convection term with u_h^{n+1} as velocity field that carries the momentum in and out of the system. In order to solve the flow with a linear system, we need to linearize such term, and the idea is to fix the value of the velocity field to something like $\beta = u_h^{n+1}$ in each step, hence

$$N(\beta) = \{n_{ij}(\beta)\} \qquad n_{ij}(\beta) = n(\beta, w_i, w_j) = \int_\Omega [(\beta \cdot \nabla) w_i] \cdot w_j dr \qquad i,j = 1,...,N. \tag{50}$$

For large Reynolds number (large convection term), is necessary to introduce numerical diffusion in the system to stabilize the algorithm. Briefly, the problem is that the shape functions "weight" in the same way all the nodes of the system for the solution computation, but if there is a strong convective field the upstream nodes will clearly have a higher impact on the downstream ones than the opposite. The standard is thus the SUPG ("Streamline Upwind Petrov Galerkin"), which is a residual based stabilization, practically adds diffusion in the same direction of the convection term.
We then clearly have to stabilize the method in the choice for β. Two possible approaches are the following.

1. **Oseen apprach.** In this case at each time step we use the velocity field of the last computed time, $\beta = u_h^n$. In this case the alogithm is simple, but it can be proved that some restrictions on the time step Δt have to be applied for stability. We choose instead the following method,
2. **Picard iteration.** This apprach is based on a standard nonlinear solver (root finding), and it practically works through an inner loop for the β stabilization. At each time step we firstly fix $\beta = u_h^{n+1,0} = u_h^n$ as the first tentative term, and then we keep solving the system, without increasing the time, and update it's value until convergence. Convergence in this case could be approximated by a test on the difference between the β of two following iterations.

For a complete analysis of the just stated terms look at chapter 5 of [2] or [3].

8.2 Stokes Vs NS: Channel

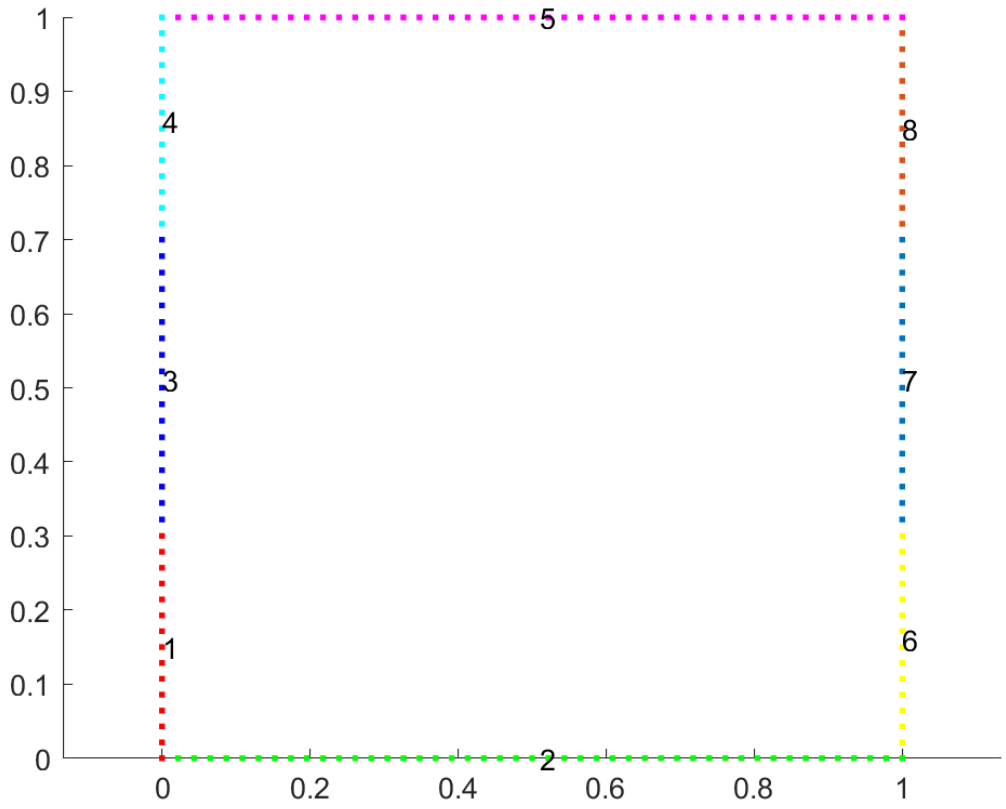
Let's consider the following problem,

$$\begin{cases} \partial_t u + (u \cdot \nabla)u + \frac{1}{\rho}\nabla p - \nu\Delta u = \begin{bmatrix}0\\0\end{bmatrix} & in\ \Omega \times [0,T] \\ \nabla \cdot u = 0 & in\ \Omega \times [0,T] \\ u(r) = u_{In}, & on\ \Sigma_3 \times [0,T] \\ u(r) = \begin{bmatrix}0\\0\end{bmatrix}, & on\ \partial\Omega \setminus (\Sigma_3 U \Sigma_7) \times [0,T] \\ [\frac{p}{\rho}I - \nu\nabla u] \cdot n = \begin{bmatrix}0\\0\end{bmatrix}, & on\ \Sigma_7 \times [0,T] \\ u = \begin{bmatrix}0\\0\end{bmatrix} & in\ \Omega \times \{0\}. \end{cases} \tag{51}$$

The boundaries of the problem are defined in figure (37). u_{In} is the chosen velocity at the inlet,

$$u_{In}(x,y) = 25(y-0.3)(0.7-y), \quad (x,y) \in \Sigma_3, \tag{52}$$

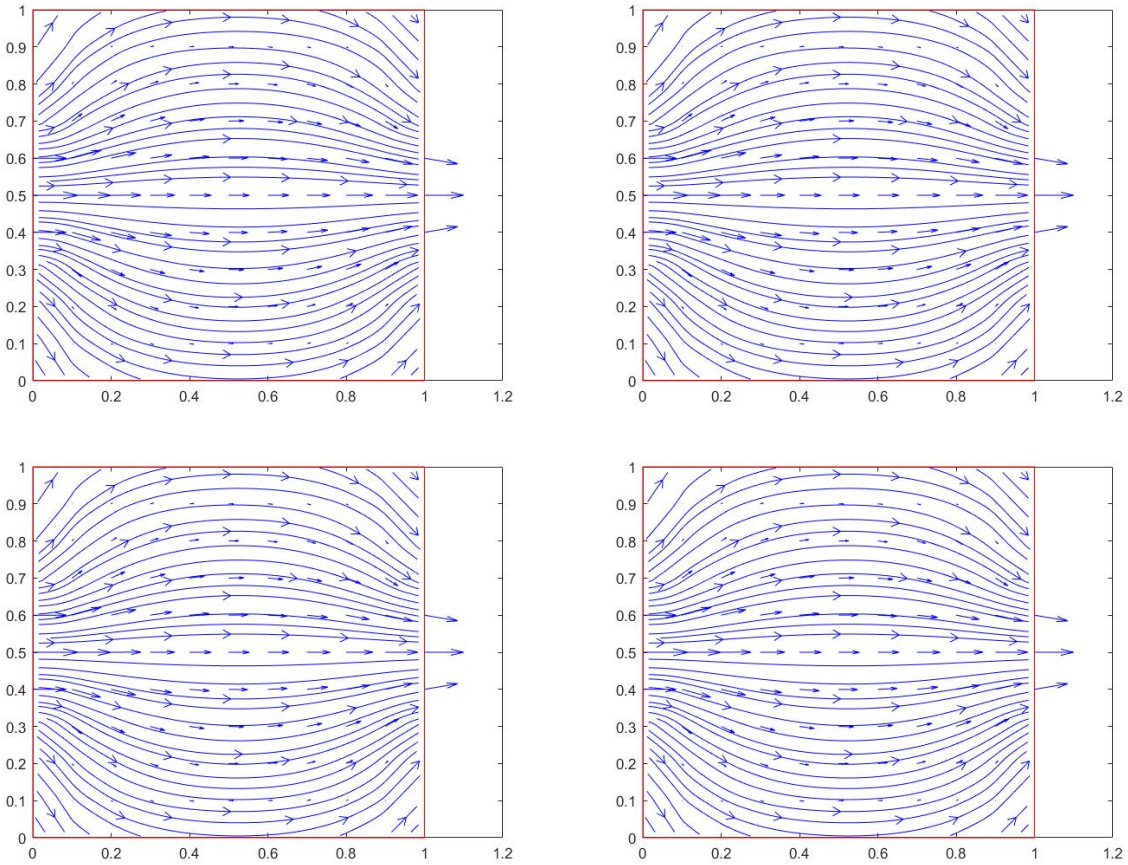
which is parabolic in the inlet $\Sigma_3 = \{(x,y) \in \partial\Omega : x = 0.3, 0 \le y \le 0.7\}$. The condition $[\frac{p}{\rho}I - \nu\nabla u]\cdot n = \begin{bmatrix}0\\0\end{bmatrix}$ is a typical Neumann BC for the outflow region, usually called "do nothing" boundary condition [4].



Channel problem boundaries

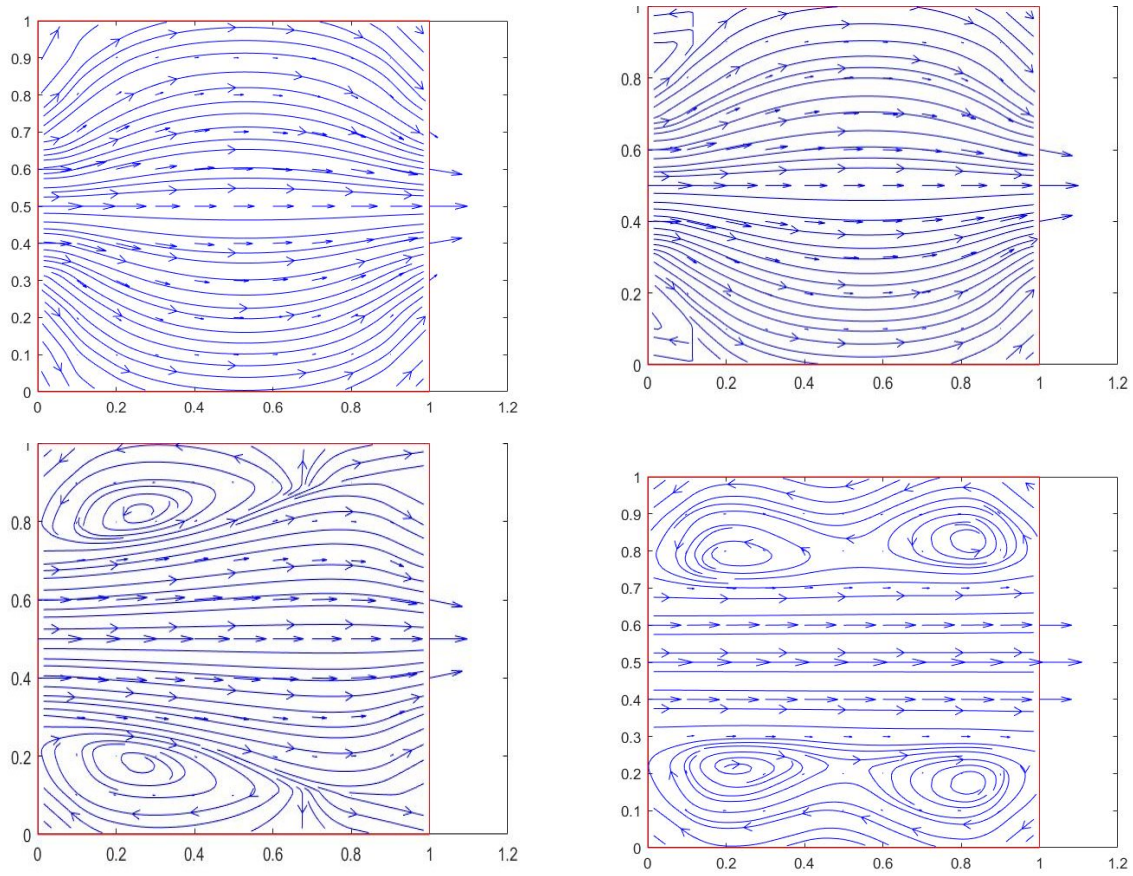
Consider the case $\rho = 1$. We compare the results for different values of the viscosity μ between the Stokes

and Navier Stokes problem to see the effects of the convective term. For simplicity we report here only the streamlines solution of the stationary state. We clearly see how the Stokes problem is unable to



Streamlines solution for the steady Stokes problem of (51), at $\mu = 1, 0.1, 0.01, 0.001$.

identify eddies in the solutions, due to the lack of the convective term in the formulation. The Navier-Stokes solver instead seem to detect in a proper way the eddies due to the sudden enlargement and in the sudden contraction of the channel.



Streamlines solution for the steady Navier Stokes problem (51), at $\mu = 1, 0.1, 0.01, 0.001$.

Bibliography

[1] MATLAB. *version R2021b*. Natick, Massachusetts: The MathWorks Inc., 2021.

[2] Mario Putti. *Notes on Numerical Methods for Differential Equations*. Department of Mathematics "Tullio Levi-Civita", University of Padua, 2021.

[3] Alfio Quarteroni. *Numerical Models for Differential Problems*. Springer, 2014.

[4] Rolf Rannacher. "Finite Element Methods for the Incompressible Navier-Stokes Equations". In: (1999).

[5] Gilbert Strang. *MATHEMATICAL METHODS FOR ENGINEERS II*. 2006.