Politecnico di Torino

Master's degree in Mathematical Engineering

Partial Different Equations Class Project

Professors
Claudio Canuto
Stefano Berrone

Candidates
Marta Di Ridolfi (LinkedIn profile)
Ludovica D'Incà (LinkedIn profile)
Biagio Torsello (LinkedIn profile)

Academic Year 2022-2023

Contents

1 Introduction

Mesh generation is a fundamental issue in scientific computing and in the simulation of all the phenomena described by partial differential equations. Finite elements are very often used in the numerical solutions of elliptic problem. Due to their flexibility in the discretization of complex domains and in giving strongly non uniform meshes, triangular grids are very common in the discretization of 2D problems. Moreover, 2D problems are the simplest 'real world problems' that can be shown.
The aim of this study was to implement, better understand and analyze the mathematical concept of finite element with respect to the discretization of the solution of partial difference equations. In particular, an unstructured triangular mesh generator called *BBTR* (see [1]) was used in order to construct a constrained triangulation of general polygonal domains in which we will find an approximation of the exact solution.
This report aims to describe an elliptical problem (and then its stabilization) in which a discretization with finite element was used. In particular for each paragraph you can find a theoretical description, followed by the Matlab implementation of each part of the code.
At the end of the theoretical display, a result analysis and discussion was done with the help of the graphs and figures given from the code.

2 Finite elements in triangular mesh

Let us consider a polygonal domain $\Omega \subset \mathbb{R}^2$ partitioned into N^E triangular elements E_n such that

- $\bigcup_{n=1}^{N^E} E_n = \bar{\Omega}$;
- $\mathring{E}_l \bigcap \mathring{E}_m = \emptyset,\ \forall l \neq m, \quad l, m = 1, \ldots, N^E$

where $\mathring{E}$ denotes the interior of E. The set of all such triangles is named *triangulation*, indicated by $\mathcal{T}$, and it is called *conforming* if for any two triangles E_l, E_m the intersection $E_l \bigcap E_m$ is either $\emptyset$, or a point, or a full edge for both; thus two adjacent triangles can only share a full edge.
The set of all vertices of the triangulation $\mathbf{a}_k$, $k = 1, \ldots, N_P$ is denoted by $P_{\mathcal{T}}$. We will enumerate each vertex with an integer number $k = 1, \ldots, N_P$ and we will consider a generic element of the partition, $E \in \mathcal{T}$, in which at every vertex corresponds a *global indices* in $\{k_1, k_2, k_3\}$; then we will introduce a surjective invertible mapping that maps the global indices k_i onto the set of *local indices* $\{1, 2, 3\}$.
For each element $E \in \mathcal{T}$, $h_E = \operatorname{diam}(E)$ denotes the diameter of the element so that we can define the meshsize of $\mathcal{T}$ as

$$h_{\mathcal{T}} := \max_{E \in \mathcal{T}} h_E;$$

we will set $h = h_{\mathcal{T}}$ for simplicity.

2.1 *The affine mapping on the reference element*

Definition 1 *[2] A **finite element** in $\mathbb{R}^d$ is a triple $(E, V_E, \mathcal{L}_E)$, where:*

- *E is a non-empty compact and connected set in $\mathbb{R}^d$, such that $E = \bar{\mathring{E}}$ and the boundary ∂E is Lipshitz-continuous;*
- *V_E is a linear space of functions defined in E, of finite dimension N_E;*

- $\mathcal{L}_E = \{l_j : 1 \le j \le N_E\}$ *is a set of linear forms* $l_j : V_E \to \mathbb{R}$ *called* ***degrees of freedom****, which is unisolvent for* V_E*, i.e.*

$$l_j(v) = 0 \qquad \forall\, j = 1, \dots, N_E \implies v = 0$$

Associated with any finite element, there is a canonical basis in V_E which allows us to represent any function in terms of its degrees of freedom.
The canonical basis $\mathbf{\Phi}_E$ is formed by the functions $\Phi_k \in V_E$, $1 \le k \le N_E$, *uniquely* defined by $l_j(\Phi_k) = \delta_{jk}$, $1 \le j \le N_E$. Any $v \in V_E$ can then be canonically represented as

$$v = \sum_{k=1}^{N_E} l_k(v)\, \Phi_k.$$

Moreover, we will assume that each finite element is the affine imagine of a **reference element** $(\hat{E}, \hat{V}, \hat{\mathcal{L}_E})$.
The reference element that has been chosen for this project are:

- $(E, \mathbb{P}_1, \mathcal{L}_E)$ with $\mathcal{L}_E = \{v(\mathbf{a}_1), v(\mathbf{a}_2), v(\mathbf{a}_3)\}$ also called **Courant element**;

- $(E, \mathbb{P}_2, \mathcal{L}_E)$ with $\mathcal{L}_E = \{v(\mathbf{a}_1), v(\mathbf{a}_2), v(\mathbf{a}_3), v(\mathbf{a}_{1,2}), v(\mathbf{a}_{2,3}), v(\mathbf{a}_{3,1})\}$ where $\mathbf{a}_{i,i+1}$ are the mid-points of each edge $L_i = [\mathbf{a}_i, \mathbf{a}_{i+1}]$ of E. This element is also called **quadratic Lagrangian element**.

The basis functions associated with the Courant element are:

$$\begin{aligned} \hat{N}_1(\hat{x}, \hat{y}) &= \hat{x}, \\ \hat{N}_2(\hat{x}, \hat{y}) &= \hat{y}, \\ \hat{N}_3(\hat{x}, \hat{y}) &= 1 - \hat{x} - \hat{y}. \end{aligned} \tag{1}$$

Whereas the basis functions associated with the quadratic Lagrangian element are (see figure 1):

$$\begin{aligned} \hat{\Phi}_1(\hat{x}, \hat{y}) &= 2\,\hat{N}_1(\hat{x}, \hat{y}) \left(\hat{N}_1(\hat{x}, \hat{y}) - \frac{1}{2}\right) = 2\,\hat{x}\left(\hat{x} - \frac{1}{2}\right), \\ \hat{\Phi}_2(\hat{x}, \hat{y}) &= 2\,\hat{N}_2(\hat{x}, \hat{y}) \left(\hat{N}_2(\hat{x}, \hat{y}) - \frac{1}{2}\right) = 2\,\hat{y}\left(\hat{y} - \frac{1}{2}\right), \\ \hat{\Phi}_3(\hat{x}, \hat{y}) &= 2\,\hat{N}_3(\hat{x}, \hat{y}) \left(\hat{N}_3(\hat{x}, \hat{y}) - \frac{1}{2}\right) = 2\,(1 - \hat{x} - \hat{y})\left(1 - \hat{x} - \hat{y} - \frac{1}{2}\right), \\ \hat{\Phi}_4(\hat{x}, \hat{y}) &= 4\,\hat{N}_2(\hat{x}, \hat{y})\,\hat{N}_1(\hat{x}, \hat{y}) = 4\,\hat{x}\,\hat{y}, \\ \hat{\Phi}_5(\hat{x}, \hat{y}) &= 4\,\hat{N}_2(\hat{x}, \hat{y})\,\hat{N}_3(\hat{x}, \hat{y}) = 4\,\hat{y}\,(1 - \hat{x} - \hat{y}), \\ \hat{\Phi}_6(\hat{x}, \hat{y}) &= 4\,\hat{N}_1(\hat{x}, \hat{y})\,\hat{N}_3(\hat{x}, \hat{y}) = 4\,\hat{x}\,(1 - \hat{x} - \hat{y}). \end{aligned} \tag{2}$$

where $\hat{N}_1$, $\hat{N}_2$, $\hat{N}_3$ are the Lagrange linear basis function described in (1).

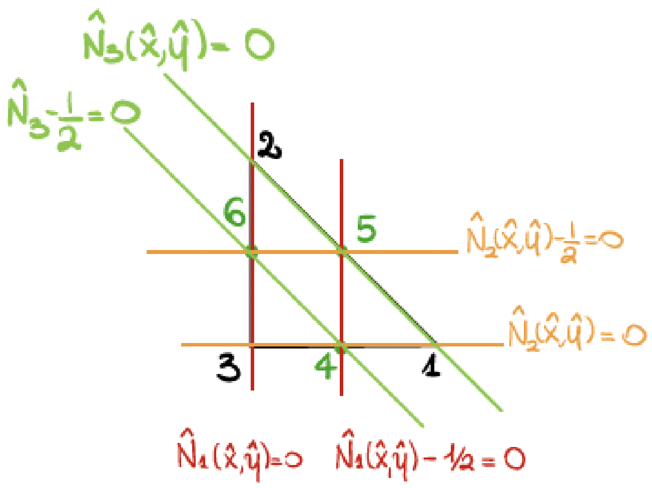

$\hat{N}_3(\hat{x},\hat{y}) - 0$
$\hat{N}_3 - \frac{1}{2} = 0$
2
6
5
$\hat{N}_2(\hat{x},\hat{y}) - \frac{1}{2} = 0$
$\hat{N}_2(\hat{x},\hat{y}) = 0$
3
4
1
$\hat{N}_1(\hat{x},\hat{y}) = 0$
$\hat{N}_1(\hat{x},\hat{y}) - 1/2 = 0$

Figure 1: Representation of the Lagrange linear basis function in $\mathbb{P}_2$ [3]

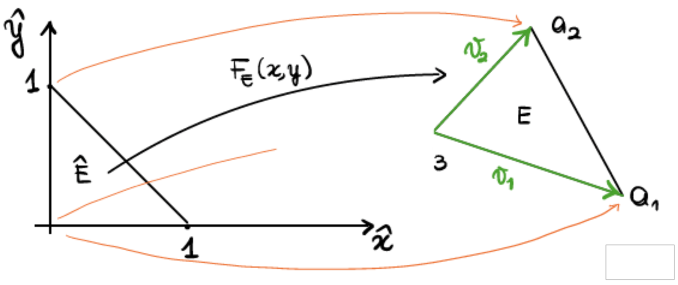


Figure 2: Affine iso-parametric mapping [3]

Let us introduce the reference triangular element $\hat{E}$ as

$$\hat{E} = \left\{ \hat{x} = \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} \subset \mathbb{R}^2 : 0 \le \hat{x}, \hat{y} \le 1,\ 0 \le 1 - \hat{x} - \hat{y} \le 1 \right\}.$$

Any triangle E in the (x, y) plane with vertices $a_1 = (x_1, y_1)$, $a_2 = (x_2, y_2)$, $a_3 = (x_3, y_3)$ can be considered as the image of the reference triangle $\hat{E}$ through the affine mapping

$$\mathbf{x} = \mathbf{F}(\hat{x}) = \mathbf{a}_3 + \mathbf{B}\hat{\mathbf{x}},$$

where

$$\mathbf{B} = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix}$$

is the matrix having as columns the triangle edges $\mathbf{v}_1 = \mathbf{a}_1 - \mathbf{a}_3$ and $\mathbf{v}_2 = \mathbf{a}_2 - \mathbf{a}_3$ (see Figure (2)).

In order to introduce the affine mapping, we can use the functions defined in (1) so that affine mapping can be written as:

$$\mathbf{F}(\hat{x}, \hat{y}) = \mathbf{a}_1\, \hat{N}_1\, (\hat{x}, \hat{y}) + \mathbf{a}_2\, \hat{N}_2\, (\hat{x}, \hat{y}) + \mathbf{a}_3\, \hat{N}_3\, (\hat{x}, \hat{y}). \tag{3}$$

Property 1 *[2] The mapping* **F** *is invertible if and only if the element E is non-degenerate.*

In other words, in order not to deal with degenerate triangles we have to have $\det(B) \neq 0$, so that the affine mapping **F** is invertible.

Assumption 1 *We assume that the vertices of any cell of our mesh are provided in a counter-clockwise versus.*

Thanks to this assumption we have that $\det(B) = +2A$, where A is the area of the triangle E considered; otherwise, if we would have considered a clockwise counting, we would have had that $\det(B) = -2A$.

Let us consider two differentiable functions $\hat{v} : \hat{E} \subset \mathbb{R}^2 \to \mathbb{R}$ and $v : E \subset \mathbb{R}^2 \to \mathbb{R}$ such that

$$\hat{v}(\hat{x}, \hat{y}) = v(\mathbf{F}(\hat{x}, \hat{y})), \quad \forall (\hat{x}, \hat{y}) \in \hat{E},$$

where $\mathbf{F}$ is the affine mapping described in (3) (Figure (2)). This being given, the derivative chain rule implies that the Jacobian matrix of the function $\hat{v}$ can be written as

$$\mathbf{J}_{\hat{v}}(\hat{x}, \hat{y}) = \mathbf{J}_v(\mathbf{F}(\hat{x}, \hat{y}))\, \mathbf{J}_{\mathbf{F}}(\hat{x}, \hat{y}) = \mathbf{J}_v(\mathbf{F}(\hat{x}, \hat{y}))\, \mathbf{B}.$$

Applying the transpose ^{T} to both sides we can find that

$$\left(\nabla_{x,y} v \circ \mathbf{F}\right)(\hat{x}, \hat{y}) = \nabla_{x,y} v\left(\mathbf{F}(\hat{x}, \hat{y})\right) = \mathbf{B}^{-T} \nabla_{\hat{x},\hat{y}} \hat{v}(\hat{x}, \hat{y}).$$

Notation 1 *[2] In the following, we will denote the gradient with respect to the coordinates x, y simply by $\nabla := \nabla_{x,y}$, whereas the gradient with respect to the coordinate $\hat{x}, \hat{y}$ of the reference element will be denoted by $\hat{\nabla} := \nabla_{\hat{x},\hat{y}}$.*

Furthermore, in the set-up of the discrete problems we will use the following integration rules, based on the affine change of variables (3):

$$\begin{aligned} \int_E f(x, y) dx\, dy &= \int_{\hat{E}} f(\mathbf{F}(\hat{x}, \hat{y}))\, |\det \mathbf{B}|\, d\hat{x}\, d\hat{y} \\ &= 2\,A \int_{\hat{E}} f(\mathbf{F}(\hat{x}, \hat{y}))\, d\hat{x}\, d\hat{y} \end{aligned} \tag{4}$$

$$\begin{aligned} \int_E \nabla f(x, y) \cdot \nabla g(x, y)\, dx\, dy &= \int_E (\nabla f(x, y))^T\, \nabla g(x, y)\, dx\, dy \\ &\int_{\hat{E}} (\nabla f(\mathbf{F}(\hat{x}, \hat{y})))^T\, \nabla g(\mathbf{F}(\hat{x}, \hat{y}))\, |\det \mathbf{B}|\, d\hat{x}\, d\hat{y} \\ &= 2\,A \int_{\hat{E}} (\hat{\nabla} \hat{f}(\hat{x}, \hat{y}))^T\, \mathbf{B}^{-1}\, \mathbf{B}{-}T\, \hat{\nabla} \hat{g}(\hat{x}, \hat{y})\, d\hat{x}\, d\hat{y} \end{aligned} \tag{5}$$

2.2 *The affine mapping in Matlab*

The affine mapping that has just being explained was translated into a function called *'Riferimento fisico'*.
The function takes as inputs the coordinates of the vertices of the physical triangle E and the $(\hat{x}, \hat{y})$ coordinates of some points of the reference triangle $\hat{E}$ (all given as column vectors) and its outputs are the (x, y) coordinates of the physical triangle E calculated using the formula (3). The function also computes and has as an output the $\mathbf{B}^{-T}$ matrix that will be used later to calculate integrals of the type described in (5).

```
function [x_punti_fisici, y_punti_fisici, B_meno_trasposto] =
    Riferimento_fisico(a_1, a_2, a_3, x_rif, y_rif)

% I define the basis functions on the reference triangle
Fe=@(x,y) [x, y, 1-x-y];

x_punti_fisici = zeros(length(x_rif),1);
```

```
y_punti_fisici = zeros(length(x_rif),1);

for i = 1:length(x_rif)

Fe_val=Fe(x_rif(i, 1), y_rif(i,1));

x_punti_fisici(i) = a_1(1,1)*Fe_val(1) + a_2(1,1)*Fe_val(2) + a_3(1,1)
    *Fe_val(3);
y_punti_fisici(i) = a_1(2,1)*Fe_val(1) + a_2(2,1)*Fe_val(2) + a_3(2,1)
    *Fe_val(3);

end

B_meno_trasposto = zeros(2,2);
B_meno_trasposto(1,1) = a_2(2,1) - a_3(2,1);
B_meno_trasposto(2,2) = a_1(1,1) - a_3(1,1);
B_meno_trasposto(1,2) = a_3(2,1) - a_1(2,1);
B_meno_trasposto(2,1) = a_3(1,1) - a_2(1,1);

end
```

For what concerns the Lagrange linear basis function (both for $\mathbb{P}_1$ and $\mathbb{P}_2$), those were written in the main code and made *global variables* which means that any change of value to that variable, in any function, is visible to all the functions that declare it as global. In the main code we checked whether we were working with $\mathbb{P}_1$ or $\mathbb{P}_2$ elements and then we defined the linear basis functions and their gradients, that we used later on in the code.

```
%Let's define the basis functions and their gradients
global fi grad_fi

if flag_P==1
    fi=@(x,y) [x, y, 1-x-y];

    gradfi_1 =@(x,y)  [1; 0];
    gradfi_2 =@(x,y)  [0; 1];
    gradfi_3 =@(x,y)  [-1;-1];
    grad_fi=@(x,y) [gradfi_1(x,y), gradfi_2(x,y), gradfi_3(x,y)];

elseif flag_P==2
    fi=@(x,y) [2*x.*(x-0.5), 2*y.*(y-0.5), 2*(1-x-y).*(0.5-x-y), 4*x.*
        y, 4*y.*(1-x-y), 4*x.*(1-x-y)];

    gradfi_1 = @(x,y) [4*x-1; 0];
    gradfi_2 = @(x,y) [0; 4*y-1];
    gradfi_3 = @(x,y) [-2*(1.5-2*x-2*y);-2*(1.5-2*x-2*y)];
    gradfi_4 = @(x,y) [4*y;4*x];
    gradfi_5 = @(x,y) [-4*y;4*(1-x-2*y)];
    gradfi_6 = @(x,y) [4*(1-2*x-y);-4*x];
    grad_fi= @(x,y)[gradfi_1(x,y), gradfi_2(x,y), gradfi_3(x,y),
        gradfi_4(x,y), gradfi_5(x,y), gradfi_6(x,y)];

end
```

3 Second-order elliptic problem

Now, let us consider the general second-order scalar elliptic equation in $\Omega \subset \mathbb{R}^2$:

$$\begin{cases} -\nabla \cdot (\nu \nabla u) + \beta \cdot \nabla u + \sigma u = f \quad \text{in } \Omega \\ \\ u = g_D \quad \text{on } \Gamma_D \\ \\ \nu \dfrac{\partial u}{\partial n} = g_N \quad \text{on } \Gamma_N \end{cases} \tag{6}$$

Assumption 2 *We want the solution to be written as $u = u_0 + Rg_D$ in which the term Rg_D is the finding of g_D in H^1*

Thank to this we can rewrite the problem as:

$$\begin{cases} -\nabla \cdot (\nu \nabla u_0) + \beta \cdot \nabla u_0 + \sigma u_0 = f + \nabla \cdot (\nu \nabla Rg_D) - \beta \cdot \nabla Rg_D - \sigma Rg_D \quad \text{in } \Omega \\ \\ u = g_D \quad \text{on } \Gamma_D \\ \\ \mu \dfrac{\partial u}{\partial n} = g_N \quad \text{on } \Gamma_N \end{cases}$$

The discrete variational formulation of the problem is:

$$\begin{gathered} \text{Finding } u_0 \in H^1_{0,\Gamma_D}(\Omega) : \\ \int_\Omega \nu \, \nabla u_0 \, \nabla v \, d\Omega \, + \int_\Omega \beta \cdot \nabla u_0 \, v \, d\Omega \, + \int_\Omega \sigma \, u_0 \, v \, d\Omega \; = \\ \int_\Omega f \, v \, d\Omega \, - \int_\Omega \nu \, \nabla Rg_D \, \nabla v \, d\Omega \, - \int_\Omega \beta \cdot \nabla Rg_D \, v \, d\Omega \, - \int_\Omega \sigma \, Rg_D \, v \, d\Omega \, + \int_{\Gamma_N} g_N \, \sigma_N \, v \, d\Gamma \quad \forall v \in H^1_{0,\Gamma_D}(\Omega) \end{gathered} \tag{7}$$

These equations are equivalent to the N_{dof} equations:

$$\begin{gathered} \int_\Omega \nu \, \nabla u_h \, \nabla \varphi_j \, d\Omega \, + \int_\Omega \beta \cdot \nabla u_h \, \varphi_j \, d\Omega \, + \int_\Omega \sigma \, u_h \, \varphi_j \, d\Omega \; = \\ \int_\Omega f \, \varphi_j \, d\Omega \, - \int_\Omega \nu \, \nabla Rg_D \, \nabla \varphi_j \, d\Omega \, - \int_\Omega \beta \cdot \nabla Rg_D \, \varphi_j \, d\Omega \, - \int_\Omega \sigma \, Rg_D \, \varphi_j \, d\Omega \, + \int_{\Gamma_N} g_N \, \sigma_N \, \varphi_j \, d\Gamma \\ \forall j = 1, \ldots, N_{\text{dof}} \end{gathered}$$

where N_{dof} is the number of elemental degrees of freedom.
The solution u_0 can be written as

$$u_0 = \sum_{k=1}^{N_{\text{dof}}} u_k \, \varphi_k = \sum_{E \in \mathcal{T}} \sum_{\hat{k}=1}^{N_E} u_{k_E(\hat{k})} \, \Phi_{\hat{k}}$$

The finding Rg_D can also be approximated by a piecewise linear function

$$\tilde{R}g_D = \sum_{k_D=1}^{N_D} \Phi_{k_D} \, g_D(a_{\text{vert}(k_D)})$$

so that the j-th equation can be written as:

$$\begin{aligned}\sum_{E\in\mathcal{T}_j}\sum_{\hat{k}=1}^{N_E}\left(\int_E \nu\,\nabla\Phi_{\hat{k}}\,\nabla\Phi_{\hat{j}}\,dx\,dy\;+\int_E \beta\cdot\nabla\Phi_{\hat{k}}\,\Phi_{\hat{j}}\,dx\,dy\;+\int_E \sigma\,\Phi_{\hat{k}}\,\Phi_{\hat{j}}\,dx\,dy\right)u_{k_E(\hat{k})} &=\\ =\sum_{E\in\mathcal{T}_j}\int_E f\,\Phi_{\hat{j}}\,dx\,dy\;+\sum_{e\in\Gamma_N}\int_e g_N\,\Phi_{\hat{j}}\,d\Gamma+ \\ -\sum_{k_D=1}^{N_D}\left(\sum_{E\in\mathcal{T}_j}\int_E \nu\,\nabla\Phi_{\hat{k}_D}\,\nabla\Phi_{\hat{j}}\,dx\,dy\;+\int_E \beta\cdot\nabla\Phi_{\hat{k}_D}\,\Phi_{\hat{j}}\,dx\,dy\;+\int_E \sigma\,\Phi_{\hat{k}_D}\,\Phi_{\hat{j}}\,dx\,dy\right)g_D(a_{\text{vert}(\hat{k}_D)})\end{aligned}\tag{8}$$

where N_D is the number of vertices in which it has been assigned a Dirichlet boundary condition, and k_D is the index that corresponds to the vertex.
We can also write this as:

$$\begin{aligned}\sum_{E\in\mathcal{T}_j}\left(\sum_{\hat{k}=1}^{N_E}\left(\int_{\hat{E}} \nu\,\hat{\nabla}\hat{\Phi}_{\hat{k}}\,\mathbf{B}^{-1}\,\mathbf{B}^{-T}\,\hat{\nabla}\hat{\Phi}_{\hat{j}}\;+\beta\cdot\hat{\nabla}\hat{\Phi}_{\hat{k}}\,\hat{\Phi}_{\hat{j}}+\sigma\,\hat{\Phi}_{\hat{k}}\,\hat{\Phi}_{\hat{j}}\right)d\hat{x}\,d\hat{y}\;2A\right)u_k &=\\ =\sum_{E\in\mathcal{T}_j}\int_{\hat{E}} f\,\hat{\Phi}_{\hat{j}}\,2A\;d\hat{x}\,d\hat{y}+\sum_{e\in\Gamma_N}\int_{\hat{e}} g_N(\gamma_e)\,\sigma_{\hat{e}}\,\hat{\Phi}_{\hat{j}}\,|e|\,d\hat{\Gamma}+ \\ -\sum_{k_D=1}^{N_D}\left(\sum_{E\in\mathcal{T}_j}\int_{\hat{E}}\left(\nu\,\hat{\nabla}\hat{\Phi}_{\hat{k}_D}\,\hat{\nabla}\hat{\Phi}_{\hat{j}}\;+\beta\cdot\hat{\nabla}\hat{\Phi}_{\hat{k}_D}\,\hat{\Phi}_{\hat{j}}\;+\sigma\,\hat{\Phi}_{\hat{k}_D}\,\hat{\Phi}_{\hat{j}}\right)2A\;d\hat{x}\,d\hat{y}\right)g_D(a_{\text{vert}(\hat{k}_D)})\end{aligned}$$

Which in a more compact form becomes

$$\begin{aligned}\sum_{E\in\mathcal{T}_j}\sum_{\hat{k}=1}^{N_E}\mathbf{A}^E_{\hat{j},\hat{k}}\,u_{k_E(\hat{k})} = \sum_{E\in\mathcal{T}_j}\sum_{\hat{k}=1}^{N_E}\left(\mathbf{D}^E_{\hat{j},\hat{k}}+\mathbf{C}^E_{\hat{j},\hat{k}}+\mathbf{R}^E_{\hat{j},\hat{k}}\right)u_{k_E(\hat{k})} &=\\ =\sum_{E\in\mathcal{T}_j} f^E_{\hat{j}}+|e|\sum_{e\in\Gamma_N}\sum_{k=1}^{N_{\text{quad}}}\omega_n\,g(\gamma(t_n))\,\Phi_j(\gamma(t)).\end{aligned}\tag{9}$$

Let us analyze the computation of the entries and the formal assembling process.
The entries of the **elemental diffusion matrix** for the homogeneous problem can be computed resorting to equation (5):

$$\begin{aligned}\mathbf{D}^E_{\hat{j},\hat{k}} &= 2\,A\int_{\hat{E}}\nu(\mathbf{F}(\hat{x},\hat{y}))\,\hat{\nabla}\hat{\Phi}^T_{\hat{k}}(\hat{x},\hat{y})\,\mathbf{B}^{-1}\,\mathbf{B}^{-T}\,\hat{\nabla}\hat{\Phi}_{\hat{j}}(\hat{x},\hat{y})\,d\hat{x}\,d\hat{y}\\ &\simeq 2\,A\sum_{q=1}^{N_q}\omega_q\,\nu(\mathbf{F}(\hat{x}_q,\hat{y}_q))\,\hat{\nabla}\hat{\Phi}^T_{\hat{k}}(\hat{x}_q,\hat{y}_q)\,\mathbf{B}^{-1}\,\mathbf{B}^{-T}\,\hat{\nabla}\hat{\Phi}_{\hat{j}}(\hat{x}_q,\hat{y}_q)\end{aligned}\tag{10}$$

In which we applied a quadrature formula on the reference triangle with nodes $\hat{\mathbf{x}}_q$ and weights ω_q in order to have the last approximation.
Similarly, the entries of the **elemental convection matrix** and **reaction matrix** so that:

$$\begin{aligned}\mathbf{C}^E_{\hat{j},\hat{k}} &= 2\,A\int_{\hat{E}}\beta(\mathbf{F}(\hat{x},\hat{y}))\,\mathbf{B}^{-T}\,\hat{\nabla}\hat{\Phi}_{\hat{k}}(\hat{x},\hat{y})\,\hat{\Phi}_{\hat{j}}(\hat{x},\hat{y})\,d\hat{x}\,d\hat{y}\\ &\simeq 2\,A\sum_{q=1}^{N_q}\omega_q\,\beta(\mathbf{F}(\hat{x}_q,\hat{y}_q))\,\mathbf{B}^{-T}\,\hat{\nabla}\hat{\Phi}_{\hat{k}}(\hat{x}_q,\hat{y}_q)\,\hat{\Phi}_{\hat{j}}(\hat{x}_q,\hat{y}_q)\end{aligned}\tag{11}$$

$$\begin{aligned}\mathbf{R}^E_{\hat{j},\hat{k}} &= 2\,A \int_{\hat{E}} \sigma(\mathbf{F}(\hat{x},\hat{y}))\,\hat{\Phi}_{\hat{k}}(\hat{x},\hat{y})\,\hat{\Phi}_{\hat{j}}(\hat{x},\hat{y})\,d\hat{x}\,d\hat{y} \\ &\simeq 2\,A \sum_{q=1}^{N_q} \omega_q\,\sigma(\mathbf{F}(\hat{x}_q,\hat{y}_q))\,\hat{\Phi}_{\hat{k}}(\hat{x}_q,\hat{y}_q)\,\hat{\Phi}_{\hat{j}}(\hat{x}_q,\hat{y}_q)\end{aligned} \tag{12}$$

The same discretization can be introduced in the **elemental diffusion, convection** and **reaction matrices** for the nodes with Dirichlet condition; the substantial difference for these ones is that the sum is done over the nodes with Dirichlet conditions, which means that the sum will result in $\sum_{q=1}^{N_D}$. Each entry $\mathbf{A}^E_{\hat{j},\hat{k}}$ is then added to the corresponding entry $\mathbf{A}_{j,k}$ of the global matrix, the same occurs for $f^E_{\hat{j}}$ added to f_j, resorting to the unique correspondence of the indices $\hat{j},\hat{k}$ and j,k for the element E.

3.1 *Second order elliptical problem in Matlab*

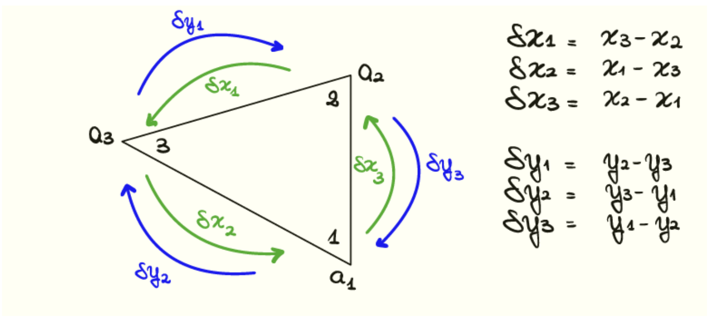
When writing the code to solve the problem we introduced the following notation:

Notation 2

$$\mathbf{B} = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix} = \begin{bmatrix} \delta x_2 & -\delta x_1 \\ -\delta y_2 & \delta y_1 \end{bmatrix}$$

$$\mathbf{B}^T = \begin{bmatrix} \delta x_2 & -\delta y_2 \\ -\delta x_1 & \delta y_1 \end{bmatrix}$$

$$\mathbf{B}^{-T} = \frac{1}{2A} \begin{bmatrix} \delta y_1 & \delta y_2 \\ \delta x_1 & \delta x_2 \end{bmatrix}$$



In order to compute the solution of the equations (9) using the formulas described in (10), (11), (12), we created a function called *'Sistema'* that takes in input the coefficients of the problem (ν, β, σ), the right-hand side (f), the functions related to the Neumann condition (gN1, gN2) and some flags value that determines whether we are dealing with a problem with $\mathbb{P}_1$, or $\mathbb{P}_2$ elements ('flag-P'), with homogeneous or non-homogeneous border conditions ('flag-DN'), with constant or non-constant coefficients ('flag-c'). The output of the function is given by the single elements of the discretization of the integrals (diffusion, convection and reaction), both for homogeneous and non-homogeneous boundary conditions and by the right hand side relative to the boundary with Neumann conditions. What happens, basically, is that the function visits each element E, builds the matrices $\mathbf{D}^E$, $\mathbf{C}^E$, $\mathbf{R}^E$ and the vector f^E, where $\mathbf{D}^E$, $\mathbf{C}^E$, $\mathbf{R}^E$ and f^E are the *elemental stiffness matrices* and the *elemental right hand-side* respectively; then it identifies which entries of $\mathbf{B}$, $\mathbf{C}$, $\mathbf{R}$ and f depend on the element E, and finally updates the current value of these entries. The resulting matrix $\mathbf{A} = \mathbf{B} + \mathbf{CR}$ is not assembled in the function *'Sistema'*, but in the main code.

Let's note that some global variable were given to the function; in particular, the number of nodes with a Dirichlet condition ('ND') which is necessary for computing the sum over the nodes. The weights related to the nodes were computed in a function called *'Nodi_Pesi´* in which a fifth order quadrature formula was chosen.
For what concerns the right hand side term, in the non-homogeneous problem other than Dirichlet condition, there is a term referring to the Neumann conditions, and thus the last integral of (7) is non zero. Particularly, in the code, for the discretization of the so said integral, we distinguished the situations in which we used $\mathbb{P}_1$ or $\mathbb{P}_2$ elements. Referring to the term $\sum_{e\in\Gamma_N}\int_e g_N\,\Phi_{\hat{j}}\,d\Gamma$ in equation (3), in the first scenario we interpolate the function g_N with linear functions on the edge e. Calling the value of g_N in the initial and final nodes of the so said edge, g_b and g_e respectively, we approximate the integral with

$$\begin{cases} \dfrac{2\,g_b+g_e}{6}\,|e| & \text{if } \Phi_j \text{ is the basis function related to the beginning node of } e \\[2ex] \dfrac{g_b+2\,g_e}{6}\,|e| & \text{if } \Phi_j \text{ is the basis function related to the ending node of } e \end{cases}$$

Whereas in the second scenario we interpolate the function g_N with quadratic functions on the edge e. Using the same notation as before for the beginning and the end of the edge, and denoting with g_m the value of g_N in the midpoint of e we approximate the integral with

$$\begin{cases} \left(2\,g_b+g_m-\dfrac{1}{2}\,g_e\right)\dfrac{|e|}{15} & \text{if } \Phi_j \text{ is the basis function related to the beginning node of } e \\[2ex] (g_b+8\,g_m+g_e)\,\dfrac{|e|}{15} & \text{if } \Phi_j \text{ is the basis function related to the midpoint of } e \\[2ex] \left(-\dfrac{1}{2}\,g_b+g_m+2\,g_e\right)\dfrac{|e|}{15} & \text{if } \Phi_j \text{ is the basis function related to the ending node of } e \end{cases}$$

```
function [D,C,R, A_D, b, b_N]=sistema(nu, beta, gamma,f, g_N1, g_N2,
    flag_P, flag_DN, flag_c)

global Ndof NE Ele XY Pivot ND Ne NN N_L Area G I fi grad_fi x_nodi
    y_nodi pesi

%let's initialize all the matrices: diffusion, convection and reaction
     (in both homogeneous and non-homogeneous cases)
D=zeros(Ndof, Ndof);
C=zeros(Ndof, Ndof);
R=zeros(Ndof, Ndof);
A_D=zeros(Ndof, ND(1));
b=zeros(Ndof,1);
b_N=zeros(Ndof,1);

%let's iterate for every element of the triangulation
for e=1:NE
    %let's compute all the delta elements in order to calculate the B-
        matrices
    d_x1=XY(Ele(e, 3),1)-XY(Ele(e, 2),1);
    d_x2=XY(Ele(e, 1),1)-XY(Ele(e, 3),1);
    d_x3=XY(Ele(e, 2),1)-XY(Ele(e, 1),1);
    dx=[d_x1, d_x2, d_x3];
    d_y1=XY(Ele(e, 2),2)-XY(Ele(e, 3),2);
    d_y2=XY(Ele(e, 3),2)-XY(Ele(e, 1),2);
    d_y3=XY(Ele(e, 1),2)-XY(Ele(e, 2),2);
    dy=[d_y1, d_y2, d_y3];
```

```matlab

    area = Area(e);

    B=[d_x2, -d_x1; -d_y2, d_y1];
    BT=[d_x2, -d_y2; -d_x1, d_y1];
    [x_punti_esatti, y_punti_esatti, B_T] = Riferimento_fisico(XY(Ele(e,1),:)',XY(Ele(e,2),:)', XY(Ele(e,3),:)', x_nodi', y_nodi');
    B_T = (1/(2*area))*(B_T);
    B_inv = B_T';
    %let's iterate for every local index (row index)
    for j=1:size(Ele,2) %3
        jj=Pivot(Ele(e,j));
        %jj is indicating the degree of freedom that corresponds to the index j of our triangle
        if(jj>0)
            %let's iterate for every local index (column index)
            for k=1:size(Ele,2) %3

                diff=0;
                conv=0;
                reaz=0;
                t_noto=0;

                for p = 1:length(x_nodi) %here we're summing every term over the squaring nodes

                    diff = diff + 2*area*pesi(p)*(nu(x_punti_esatti(p,1),y_punti_esatti(p,1)))*(grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,k))'*B_inv*B_T*(grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,j));
                    conv= conv+2*area*pesi(p)*(beta(x_punti_esatti(p),y_punti_esatti(p)))'*B_T*(grad_fi(x_nodi(p),y_nodi(p))*I(:,k))*(fi(x_nodi(p),y_nodi(p))*I(:,j));
                    reaz=reaz+2*area*pesi(p)*gamma(x_punti_esatti(p),y_punti_esatti(p))*(fi(x_nodi(p),y_nodi(p))*I(:,k))*(fi(x_nodi(p),y_nodi(p))*I(:,j));
                    t_noto=t_noto+2*area*pesi(p)*f(x_punti_esatti(p),y_punti_esatti(p))*(fi(x_nodi(p),y_nodi(p))*I(:,j));
                    end
                end
                kk=Pivot(Ele(e,k));
                if(kk>0)
                    D(jj,kk)=D(jj,kk)+diff;
                    C(jj,kk)=C(jj,kk)+conv;
                    R(jj,kk)=R(jj,kk)+reaz;
                else
                    A_D(jj,-kk)=A_D(jj,-kk)+diff+con+reaz;
                end
            end
            b(jj)=b(jj)+t_noto;
        end
    end
end
%----Known term related to Neumann border conditions----
```

```matlab
for e=1:NN(1)
    l=Ne(e,1);
    i_b=N_L(l,1); %index of the beginning vertex of the edge e
    i_e=N_L(l,2); %index of the end vertex of the edge e
    ii_b=Pivot(i_b); %degree of freedom of the index of the i_b node
    ii_e=Pivot(i_e); %degree of freedom of the index of the i_e node

    if(Ne(e,2)==2)
        g_N=g_N1;
    elseif(Ne(e,2)==4)
        g_N=g_N2;
    end
%let's check if we're dealing with a P1 or a P2 problem
    if flag_P==1
        gb=g_N(XY(i_b,1),XY(i_b,2));
        ge=g_N(XY(i_e,1),XY(i_e,2));
        length_lato=norm(abs(XY(i_e,:)-XY(i_b,:)),2);

        %let's check if whether the edge has or not a node with
           Dirichlet condition
        if(ii_b>0)
               b_N(ii_b)=b_N(ii_b)+((2*gb+ge)*length_lato)/6;
        end
        if(ii_e>0)
               b_N(ii_e)=b_N(ii_e)+((gb+2*ge)*length_lato)/6;
        end

    elseif flag_P==2
        i_m=N_L(l, 5); %index of the middle point of edge e
        ii_m=Pivot(i_m); %degree of freedom of the index of the i_m
           node
        gb=g_N(XY(i_b,1),XY(i_b,2));
        ge=g_N(XY(i_e,1),XY(i_e,2));
        gm=g_N(XY(i_m,1), XY(i_m,2));
        h_quad=norm(abs(XY(i_e,:)-XY(i_b,:)),2)/6;

        %let's check if whether the edge has or not a node with
           Dirichlet condition
        if(ii_b>0)
               b_N(ii_b)=b_N(ii_b)+((gb+4*gm*0+ge*0)*h_quad);
        end

        if(ii_m>0)
               b_N(ii_m)=b_N(ii_m)+((gb*0+4*gm+ge*0)*h_quad);
        end

        if(ii_e>0)
               b_N(ii_e)=b_N(ii_e)+((gb*0+4*gm*0+ge)*h_quad);
        end
    end
end
end
```

4 Error functions and convergence order

In order to verify if what we have done until now is correct, we computed the numerical solution of the problem with a given meshsize and the error of the discretization related to it. Then we refined the triangulation, reducing its meshsize, and we studied if and how the error improved.
To be more specific, the error that are computed are the L^2, H^1 and the L^∞ errors, given as:

$$\begin{aligned} \|u-u_h\|_{L^2(\Omega)} &= \int_\Omega (u-u_h)^2\,d\Omega = \sum_{E\in\mathcal{T}_j}\int_E (u-u_h)^2\,d\Omega = \\ &= A\sum_{E\in\mathcal{T}_j}\int_{\hat{E}}\Big(u(\mathbf{F}(\hat{x},\hat{y})) - \sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})}\,\hat{\Phi}_{\hat{k}}(\hat{x},\hat{y})\Big)^2\,d\hat{\Omega} = \\ &\simeq \sum_{E\in\mathcal{T}_j}\sum_{q=1}^{N_q}\omega_q\Big(u(\mathbf{F}(\hat{x}_q,\hat{y}_q)) - \sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})}\,\hat{\Phi}_{\hat{k}}(\hat{x}_q,\hat{y}_q)\Big)^2 \end{aligned} \tag{13}$$

$$\begin{aligned} \|\nabla u-\nabla u_h\|_{L^2(\Omega)} &= \int_\Omega (\nabla u-\nabla u_h)^T(\nabla u-\nabla u_h)\,d\Omega = \sum_{E\in\mathcal{T}_j}\int_E (\nabla u-\nabla u_h)^T(\nabla u-\nabla u_h)\,d\Omega = \\ &= \sum_{E\in\mathcal{T}_j}\int_{\hat{E}}\Big(\nabla u(\mathbf{F}(\hat{x},\hat{y})) - \nabla u_h(\mathbf{F}(\hat{x},\hat{y}))\Big)^T\Big(\nabla u(\mathbf{F}(\hat{x},\hat{y})) - \nabla u_h(\mathbf{F}(\hat{x},\hat{y}))\Big)\,|E|\,d\hat{\Omega} = \\ &= \sum_{E\in\mathcal{T}_j}\int_{\hat{E}}\Big(\nabla u(\mathbf{F}(\hat{x},\hat{y})) - \mathbf{B}^{-T}\sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})}\,\hat{\nabla}\hat{\Phi}_{\hat{k}}(\hat{x},\hat{y})\Big)^T\Big(\nabla u(\mathbf{F}(\hat{x},\hat{y})) - \mathbf{B}^{-T}\sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})}\,\hat{\nabla}\hat{\Phi}_{\hat{k}}(\hat{x},\hat{y})\Big)\,|E|\,d\hat{\Omega} = \\ &\simeq \sum_{E\in\mathcal{T}_j}\sum_{q=1}^{N_q}\omega_q\Big(\nabla u(\mathbf{F}(\hat{x}_q,\hat{y}_q)) - \mathbf{B}^{-T}\sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})}\,\hat{\nabla}\hat{\Phi}_{\hat{k}}(\hat{x}_q,\hat{y}_q)\Big)^T\Big(\nabla u(\mathbf{F}(\hat{x}_q,\hat{y}_q)) - \mathbf{B}^{-T}\sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})}\,\hat{\nabla}\hat{\Phi}_{\hat{k}}(\hat{x}_q,\hat{y}_q)\Big)\,|E| \end{aligned} \tag{14}$$

$$||u-u_h||^2_{H^1\Omega} = ||u-u_h||^2_{L^2(\Omega)} + \|\nabla u-\nabla u_h\|_{L^2(\Omega)} \tag{15}$$

and

$$\|u-u_h\|_\infty = \max_{x\in\Omega}|u(x)-u_h(x)| \tag{16}$$

in which with u_h we are referring to the discretized solution and the sum $\sum_{\hat{k}=1}^{N^E}$ is over the finding if non-homogeneous border conditions were given; on the other hand, if homogeneous border conditions were given, then the sum should have been $\sum_{\hat{k}=1}^{N_{\text{dof}}}$. (Let's highlight that this argument is going to be used also for the following error functions).

Several theoretical results (such as the Cea's lemma, which is a consequence of the Lax-Milgram Theorem) and hypotesis (e.g. shape regular family of triangulations) ensure that if $u \in H^r(\Omega)$, $r \le k+1$, one has the following error bound:

$$||u-u_h||_{1,\Omega} \le Ch^{r-1}|u|_{r,\Omega} \tag{17}$$

Having this error bound, we obtain the following estimates:

$$\mathcal{E}_0 = \|u-u_h\|_{L^2(\Omega)} \sim ch^2|u|_2 \tag{18}$$

$$\mathcal{E}_1 = \|u-u_h\|_{H^1(\Omega)} \sim ch^1|u|_2 \tag{19}$$

that if $\mathbb{P}_1$ elements were used.
Instead, if $\mathbb{P}_2$ elements were used, we have that

$$\mathcal{E}_0 = \|u-u_h\|_{L^2(\Omega)} \sim ch^3|u|_3 \tag{20}$$

$$\mathcal{E}_1 = \|u-u_h\|_{H^1(\Omega)} \sim ch^2|u|_3 \tag{21}$$

In all these estimates we considered the function u sufficiently regular in the sense of Sobolev regularity; namely $u \in H^2(\Omega)$ for $P1$ elements and $u \in H^3(\Omega)$ for $P2$.

4.1 *Error function and convergence error in Matlab*

In order to compute the errors and to understand if the written code was operating correctly, an error function called *'Errore'* was created. This function takes as inputs the exact solution, the discretized solution, the function u with its gradient and a flag value that specify whether the problem is being solved with $\mathbb{P}_1$or $\mathbb{P}_2$ elements and it gives as outputs the H^1, L^2 and L^∞ errors.

```matlab
function [err_H_1,  err_grad, err_l2, err_inf]=errore(u, u_esatta, U, gradU, flag_P)

global NE Ele XY Area I fi grad_fi x_nodi y_nodi pesi

err_l2 = 0;
err_grad = 0;

%---With P1 elements---
if flag_P==1
    for e = 1:NE
        area = Area(e);
        contributo = 0;
        contributo_grad = 0;

        [x_punti_esatti, y_punti_esatti, B_T] = Riferimento_fisico(XY(Ele(e,1),:)',...
            XY(Ele(e,2),:)', XY(Ele(e,3),:)', x_nodi', y_nodi');
        B_T = (1/(2*area))*(B_T);

        for p = 1:length(x_nodi)
            % summing all the quatities from the nodes of the grid.

            contributo = contributo + 2*area*pesi(p)*...
                (U(x_punti_esatti(p),y_punti_esatti(p))-u(Ele(e,1),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,1))-u(Ele(e,2),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,2))-u(Ele(e,3),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,3)))^2;

            contributo_grad = contributo_grad + pesi(p)*2*area*...
                (gradU(x_punti_esatti(p),y_punti_esatti(p))- B_T*...
                (u(Ele(e,1),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,1))+...
                u(Ele(e,2),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,2))+...
                u(Ele(e,3),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,3))))'*...
                (gradU(x_punti_esatti(p),y_punti_esatti(p))...
                - B_T*(u(Ele(e,1),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,1))+...
                u(Ele(e,2),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,2))+...
                u(Ele(e,3),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,3))));
        end
        err_l2 = err_l2 + contributo;
        err_grad = err_grad + contributo_grad;
    end
end

%---With P2 elements---
```

```
if flag_P==2
    for e = 1:NE
        area = Area(e);
        contributo = 0;
        contributo_grad = 0;

        [x_punti_esatti, y_punti_esatti, B_T] = Riferimento_fisico(XY(
            Ele(e,1),:)',...
            XY(Ele(e,2),:)', XY(Ele(e,3),:)',...
            x_nodi', y_nodi');
        B_T = (1/(2*area))*(B_T);

        for p = 1:length(x_nodi)
            % summing all the quatities from the nodes of the grid.

            contributo = contributo + 2*area*pesi(p)*...
                (U(x_punti_esatti(p),y_punti_esatti(p))-u(Ele(e,1),1)*
                    ...
                (fi(x_nodi(p),y_nodi(p))*I(:,1))-u(Ele(e,2),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,2))-u(Ele(e,3),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,3))-u(Ele(e,4),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,4))-u(Ele(e,5),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,5))-u(Ele(e,6),1)*...
                (fi(x_nodi(p),y_nodi(p))*I(:,6)))^2;

            contributo_grad = contributo_grad + pesi(p)*2*area*...
                (gradU(x_punti_esatti(p),y_punti_esatti(p))- B_T*...
                (u(Ele(e,1),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,1))+
                    ...
                u(Ele(e,2),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,2))+...
                u(Ele(e,3),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,3))+...
                u(Ele(e,4),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,4))+...
                u(Ele(e,5),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,5))+...
                u(Ele(e,6),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,6))))'*
                    ...
                (gradU(x_punti_esatti(p),y_punti_esatti(p))...
                - B_T*(u(Ele(e,1),1)*(grad_fi(x_nodi(p),y_nodi(p))*I
                    (:,1))+...
                u(Ele(e,2),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,2))+...
                u(Ele(e,3),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,3))+...
                u(Ele(e,4),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,4))+...
                u(Ele(e,5),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,5))+...
                u(Ele(e,6),1)*(grad_fi(x_nodi(p),y_nodi(p))*I(:,6))));
        end
        err_l2 = err_l2 + contributo;
        err_grad = err_grad + contributo_grad;
    end
end

err_H_1 = sqrt(err_l2 + err_grad);

%the H1 error should decrease with h^2, which means that divinding the
    area, we should get an error which is the previous error divided
    by \sqrt(2)
err_l2 = sqrt(err_l2);
err_grad = sqrt(err_grad);
```

```
err_inf=max(abs(u-u_esatta));
end
```

Another function, called *'Ordine_convergenza'* was then created to calculate the convergence orders. This function takes as inputs the error values calculated in *'Errore'*, the mesh size ('*h*'), and the size of the area of the elements ('element_area') and it gives as outputs an error matrix which will be plotted in the main code. Moreover, inside this function, plots of the errors, computed in *'Errore'*, were created using the Matlab function *'polyfit'* with respect to the mesh size and to the area of the elements.

```
function [error_matrix] = ordine_convergenza(element_area, h, err_l2,
    err_inf, err_H_1, err_grad, Np)

error_matrix = ones(4,3);

p_inf_h = polyfit(log(h), log(err_inf), 1);
p_l2_h = polyfit(log(h), log(err_l2), 1);
p_H_1_h = polyfit(log(h), log(err_H_1), 1);
p_H10_h = polyfit(log(h), log(err_grad), 1);

error_matrix(:,1) = [p_inf_h(1);p_l2_h(1);p_H_1_h(1);p_H10_h(1)];

p_inf_area = polyfit(log(element_area), log(err_inf), 1);
p_l2_area = polyfit(log(element_area), log(err_l2), 1);
p_H_1_area = polyfit(log(element_area), log(err_H_1), 1);
p_H10_area = polyfit(log(element_area), log(err_grad), 1);

error_matrix(:,2) = [p_inf_area(1);p_l2_area(1);p_H_1_area(1);
    p_H10_area(1)];

p_inf_vertices_number = polyfit(log(Np),log(err_inf), 1);
    p_l2__vertices_number = polyfit(log(Np), log(err_l2), 1);
    p_H_1_vertices_number = polyfit(log(Np), log(err_H_1), 1);
p_H10_vertices_number = polyfit(log(Np), log(err_grad), 1);

error_matrix(:,3) = [p_inf_vertices_number(1);p_l2_vertices_number(1);
    p_H_1_vertices_number(1);p_H10_vertices_number(1);

%---Graph of L2 and H1 errors---
%Errors with respect to the mesh size
figure(2)
subplot(1,2,1)
loglog(h, err_l2, 'b')

subplot(1,2,2)
loglog(h,err_H_1, 'r')

%Errors with respect to the area
figure(3)
subplot(1,2,1)
loglog(element_area,err_l2, 'b')

subplot(1,2,2)
loglog(element_area, err_H_1, 'r')
end
```

5 The problem

Everything that has just been said is general and could be used for any function and problem that follows the hypothesis mentioned in the prior sections. In particular, we worked with two different functions: one for the homogeneous case and one for the non-homogeneous one. Both the functions follow the **Lax-Milgram Theorem** (Appendix, I) which guarantees the existence and the uniqueness of the solution of the specific problem.
Moreover we chose to work both with constant and non-constant coefficients which are (essentially) bounded, i.e., $\nu, \sigma \in L^{\infty}(\Omega)$ and $\beta \in (L^{\infty})^d$, in particular we need to have $\nu \geq \nu_0 > 0$ in Ω and $-\frac{1}{2}\nabla \cdot \beta + \sigma \geq \sigma_0 > 0$ in Ω, where ν_0 and σ_0 are constants; thanks to these assumptions we obtain a coercivity result.

5.1 *The homogeneous problem*

Starting from system 6 (section 3) we defined the specific system:

$$\begin{cases} -\nabla \cdot (\nu \nabla u) + \beta \cdot \nabla u + \sigma u = f \quad \text{in } \Omega = [0,1] \times [0,1] \\ \\ u = 0 \quad \text{on } \Omega \end{cases} \tag{22}$$

where we set two different set of coefficient, as we previously mentioned.
The first one takes into account only constant coefficients which were set to

$$\nu = 1$$
$$\beta = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$
$$\sigma = 1.$$

The second one, instead, takes into account non-constant coefficients which were set to

$$\nu = y + 1$$
$$\beta = \begin{pmatrix} x+2 \\ 4x \end{pmatrix}$$
$$\sigma = x^2 + y^2 + 1.$$

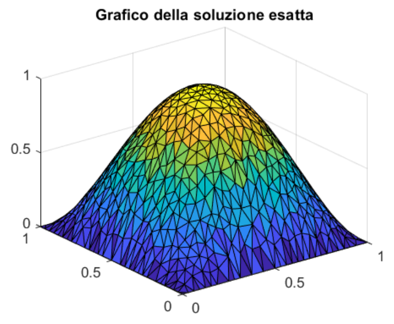
More importantly, the exact solution u of (6) that was chosen is (see figure 12)

$$u = 16\, x\, (1-x)\, y\, (1-y) \tag{23}$$

with gradient

$$\nabla u = \begin{pmatrix} 16\, y\, (1-y)\, (1-2x) \\ 16\, x\, (1-x)\, (1-2y) \end{pmatrix}$$

Let's note that the problem with constant coefficients is a diffusion-reaction problem only, since the convection term β is set to zero.



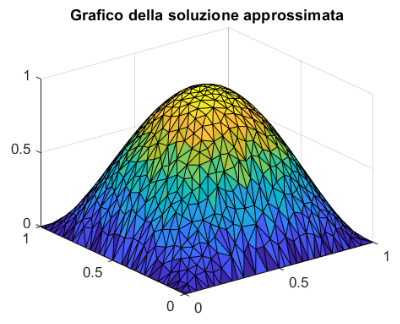


Figure 3: Exact solution and its approximation (with non-constant coefficients).

5.2 ***Non-homogeneous problem***

Again, starting from system 6 (section 3) we defined the specific system:

$$\begin{cases} -\nabla \cdot (\nu \nabla u) + \beta \cdot \nabla u + \sigma u = f \quad \text{in } \Omega = [0,1] \times [0,1] \\ \\ u = g_D \quad \text{on } \Gamma_D \\ \\ \nu \dfrac{\partial u}{\partial n} = g_N \quad \text{on } \Gamma_N \end{cases} \tag{24}$$

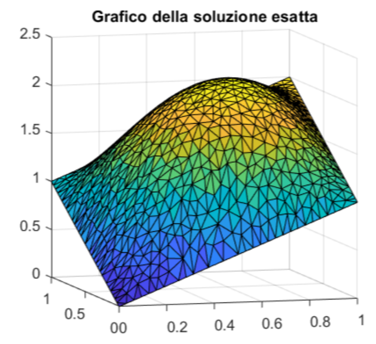
where we set two different set of coefficient, which are the same as the one set in the previous section. The exact solution u chosen for the non-homogeneous problem is (see figure 15)

$$u = 16\,x\,(1-x)\,y\,(1-y) + x + y \tag{25}$$

with gradient

$$\nabla u = \begin{pmatrix} 16\,y\,(1-y)\,(1-2x) + 1 \\ 16\,x\,(1-x)\,(1-2y) + 1 \end{pmatrix}$$

The function on the Dirichlet conditioned boundary g_D was set to $g_D = x + y$ while the function on the Neumann conditioned boundary was computed using the formula $\nu \dfrac{\partial u}{\partial n}$, which is the partial derivative with respect to the direction in which the normal of the considered edge points.



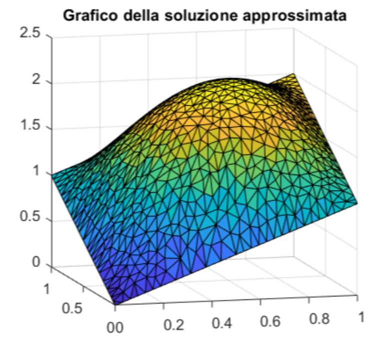


Figure 4: Exact solution and its approximation (with non-constant coefficients).

We also observe that the choice of non-constant coefficients was made in order to ensure sufficient conditions for the coercivity of the bilinear form of the variational problem. In particular, these conditions are the following ones:

$$\nu \geq \nu_0 > 0 \quad \text{in} \quad \Omega, \qquad -\frac{1}{2}\nabla\cdot\beta \geq 0 \quad \text{in} \quad \Omega, \qquad \beta\cdot\mathrm{n} \geq 0 \quad \text{in} \quad \Gamma_{\mathrm{N}}$$

It is easy to show that the chosen coefficients satisfy them.

Notation 3 *For both homogeneous and non-homogeneous problems the right-hand side was computed as*

$$f(x,y) = -\nabla u(x,y)\,\nabla\nu(x,y) + \nu(x,y)\,\triangle\,u(x,y) + \nabla u(x,y)\,\beta(x,y) + \sigma(x,y)\,u \tag{26}$$

6 Result analysis

We are now going to analyze the data and results of the problem introduced in the previous section, specifically taking into account the error and the convergence order for what concerns the use of $\mathbb{P}_1$ and $\mathbb{P}_2$ elements. There exists two main strategies to estimate the discretization error, which are somehow complementary to each other: **priori error estimation** and **posteriori error estimation**.
The estimation used in our code is the first one, which gives an estimate of the error before even computing the Galerkin solution, but by just knowing certain qualitative properties of the exact solution u. This strategy is more theoretically oriented, aiming at predicting the qualitative behaviour of the discretization error, such as its asymptotic decay as the discretization parameter h becomes smaller and smaller.

In order to see the decay of the error with the mesh size we used the **Solve-Estimate-Refine algorithm** which means that we computed the Galerkin solution (solve), then estimated the error (estimate) and then we generated a new partition by halving the greatest area of the partition (refine).

Starting from (18) and applying the logarithm function on both sides of the equation we have that

$$\log(\mathcal{E}_0) \sim log(c\,|u|_{k+1}) + (k+1)\,log(h) \tag{27}$$

So, plotting $\mathcal{E}_0$ with respect to h on a logarithmic base we're expecting a linear decay in which the slope $m = k + 1$ gives the error convergence order with respect to the mesh size.
Similarly, for the H^1 error we have

$$\log(\mathcal{E}_1) \sim log(c\,|u|_{k+1}) + k\,log(h) \tag{28}$$

Another way to analyze the error is taking into account the link between the error and the current greatest area of the triangles of the partition, which we called *'element area'* (A). Knowing that element area $\sim h^2$ (this is true thanks to the hypotesis of shape regularity of the family of triangulations), thanks to (27) and (28) we have:

$$\begin{aligned} \log(\mathcal{E}_0) &\sim log(c\,|u|_{k+1}) + \frac{k+1}{2}\,log(A) \\ \log(\mathcal{E}_1) &\sim log(c\,|u|_{k+1}) + \frac{k}{2}\,log(A) \end{aligned} \tag{29}$$

Furthermore, we can study the convergence order of the error with respect to the vertices number of the triangulation. Indeed, if Ω is a square domain and we introduce a partition for each edge in N_x and N_y points respectively, than we have totally $N = N_x\,N_y$ nodes and $h \sim \frac{1}{N_x} \sim \frac{1}{N_y}$. Thus $h \sim N^{-1/2}$ and

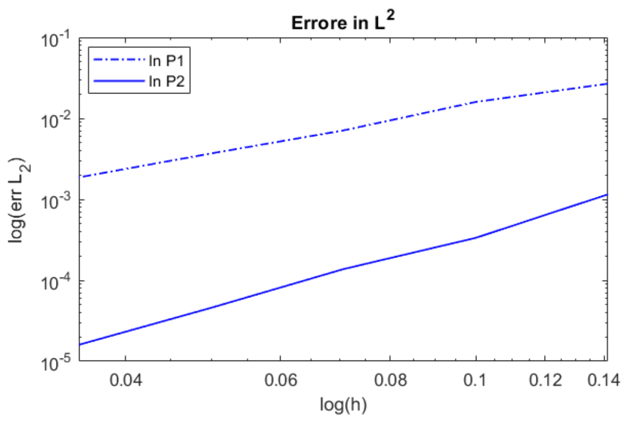
$$\mathcal{E} \sim h^{\alpha} \quad \Longrightarrow \quad \mathcal{E} \sim N^{-\frac{\alpha}{2}} \tag{30}$$

Lastly, one can prove the following error estimates for L^∞ norm:

$$||u - u_h||_{L^\infty(\Omega)} \leq Ch^2|log(h)|, \qquad \text{if we use } \mathbb{P}_1 \text{ elements} \tag{31}$$

$$||u - u_h||_{L^\infty(\Omega)} \leq Ch^3, \quad \text{if we use } \mathbb{P}_2 \text{ elements} \tag{32}$$

From this studies follows that using $\mathbb{P}_1$ elements the L^2 error decay quadratically with h and linearly with *'element area'*. The convergence order with respect to vertices number is -1. Moreover the H^1 error decay linearly with h, the convergence order with respect to *'element area'* is $\frac{1}{2}$, while the convergence order with respect to vertices number is $-\frac{1}{2}$. Instead, using $\mathbb{P}_2$ elements the L^2 error decay cubically with h and quadrically with *'element area'*, gaining in both regularity and precision. The convergence order with respect to vertices number is $-\frac{3}{2}$. Moreover the H^1 error decays quadratically with h, and linearly with *'element area'*, and its convergence order with respect to vertices number is -1.
Finally, in both cases the L^∞ error shows a behaviour which is similar to the L^2 error.



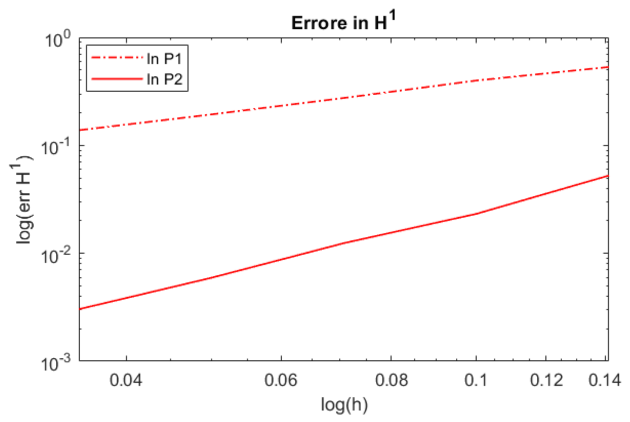
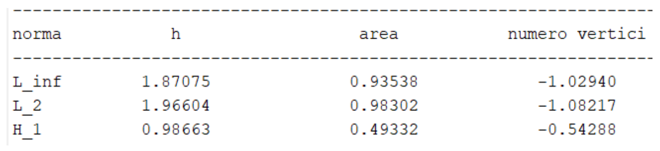
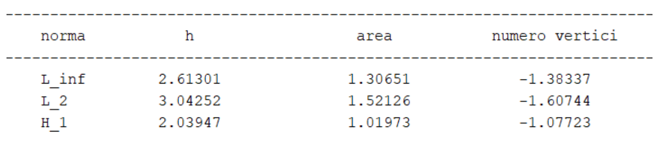


Figure 5: Error in norm L^2 and H^1 with respect to the mesh size h (homogeneous problem with non-constant coefficients).



```
--------------------------------------------------------------------
norma             h                   area                numero vertici
--------------------------------------------------------------------
L_inf         1.87075             0.93538             -1.02940
L_2           1.96604             0.98302             -1.08217
H_1           0.98663             0.49332             -0.54288
```

Figure 6: Convergence orders in the L^∞, L^2, H_1 norms, with respect to the maximal diameter of the triangles, the maximal area and the total number of vertices of the partition in $\mathbb{P}_1$ (homogeneous problem with non-constant coefficients).



```
--------------------------------------------------------------------
    norma            h                   area              numero vertici
--------------------------------------------------------------------
    L_inf        2.61301             1.30651             -1.38337
    L_2          3.04252             1.52126             -1.60744
    H_1          2.03947             1.01973             -1.07723
```

Figure 7: Convergence orders in the L^∞, L^2, H_1 norms, with respect to the maximal diameter of the triangles, the maximal area and the total number of vertices of the partition in $\mathbb{P}_2$ (homogeneous problem with non-constant coefficients).

From the graphs above we can see that with the code we implemented the a priori estimates are followed in both with the use of $\mathbb{P}_1$ and $\mathbb{P}_2$ elements. Furthermore, we want to highlight how the approximation given with $\mathbb{P}_2$ elements is far more precise and gives better results than the one given with $\mathbb{P}_1$ elements. Looking at Figure 5 it's obvious that using the same mesh size the two errors decay more rapidly with $\mathbb{P}_2$ than with $\mathbb{P}_1$ (as we expected to).

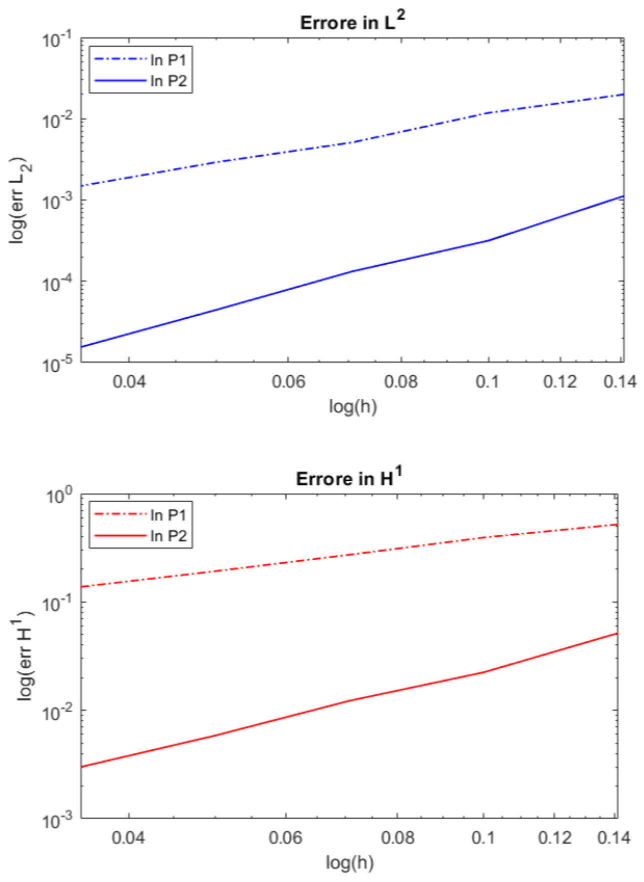


Figure 8: Error in norm L^2 and H^1 with respect to the mesh size h (non- homogeneous problem with non-constant coefficients).

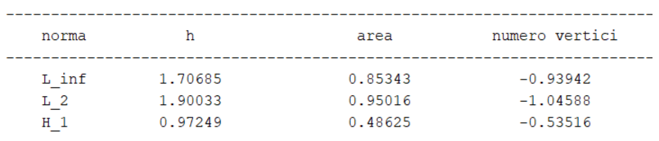

```
--------------------------------------------------------------------
  norma           h                    area             numero vertici
--------------------------------------------------------------------
  L_inf        1.70685              0.85343              -0.93942
  L_2          1.90033              0.95016              -1.04588
  H_1          0.97249              0.48625              -0.53516
```

Figure 9: Convergence orders in the L^∞, L^2, H_1 norms, with respect to the maximal diameter of the triangles, the maximal area and the total number of vertices of the partition in $\mathbb{P}_1$ (non-homogeneous problem with non-constant coefficients).

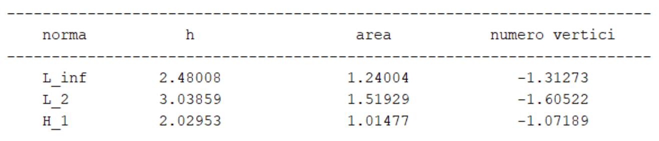

```
--------------------------------------------------------------------
  norma           h                    area             numero vertici
--------------------------------------------------------------------
  L_inf        2.48008              1.24004              -1.31273
  L_2          3.03859              1.51929              -1.60522
  H_1          2.02953              1.01477              -1.07189
```

Figure 10: Convergence orders in the L^∞, L^2, H_1 norms, with respect to the maximal diameter of the triangles, the maximal area and the total number of vertices of the partition in $\mathbb{P}_2$ (non-homogeneous problem with non-constant coefficients).

The analysis and examinations done for the homogeneous problem (Figure 5, 6, 7) could be also done in the graphs above, which refer to the non-homogeneous case.
We notice that the non-homogeneous problem error decays slightly less rapidly than the homogeneous one, which we thought could be related to the higher computational cost given from the greater number of operations the system has to compute (i.e. the quadrature formulas used in the integrals approximations).

Finally, in order to verify if our code is correct we can compute the condition number of the stiffness matrix A. As theoretical result we know that

$$\mathrm{cond}_2(A) \sim h^{-2}$$

Thus, plotting the number condition of A with respect to h^{-2} we see a linear behaviour. Our code complies this theoretical result as, for example, the next Figure shows:

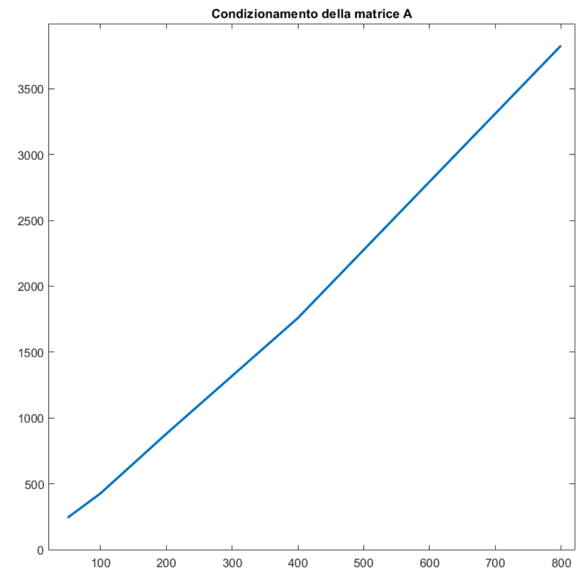


Figure 11: Condition number of A with respect to h^{-2} ($\mathbb{P}_2$ elements non-homogeneous problem with non-constant coefficients).

7 Stabilization for Convection Diffusion problems

7.1 *A brief theoretical introduction*

Let us consider the convection-diffusion equation with positive constant coefficients ν and β defined in the interval $(0, L) \subset \mathbb{R}$.

$$\begin{aligned} -\nu \frac{\partial^2 u}{\partial x^2} + \beta \frac{\partial u}{\partial x} = 0 \qquad 0 < x < L \\ u(0) = u_0 \\ u(L) = u_L \end{aligned} \tag{33}$$

and let us consider a *uniform partition* in N intervals with length $h = \frac{L}{N}$.
By using a finite element discretization with linear elements or a centered finite difference discretization we can approximate (33) with:

$$\begin{aligned} -\nu \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \beta \frac{u_{i+1} - u_{i-1}}{2h} = 0 \quad i = 1, \ldots, N-1 \\ u_0 = u_0 \\ u_N = u_L \end{aligned} \tag{34}$$

Considering the 'mesh *Peclet number*'

$$P_{e_E} = \frac{\beta h}{2\nu},$$

defined with the scale length of the cells, equation (34) can be written as

$$(P_{e_E} - 1)\, u_{i+1} + 2\, u_i - (P_{e_E} + 1)\, u_{i-1} = 0. \tag{35}$$

Now, looking for a solution of (35) with the structure $u_i = \mathrm{const}\, \xi^i$, the equation can be written as:

$$\xi^{i-1} \left[(P_{e_E} - 1)\, \xi^2 + 2\, \xi - (P_{e_E} + 1) \right] = 0,$$

that is satisfied by the values

$$\begin{aligned} \xi_0 &= 0 \\ \xi_1 &= 1 \\ \xi_2 &= \frac{1 + P_{e_E}}{1 - P_{e_E}} \end{aligned}$$

The general solution results in

$$u_i = C_1 + C_2 \left(\frac{1 + P_{e_E}}{1 - P_{e_E}} \right)^i = C_1 + C_2\, \xi_2^i.$$

The constants C_1 e C_2 are given by the imposition of the boundary condition given by 34. The exact solution to the continuous differential problem is

$$u = u_0 + (u_L - u_0) \frac{e^{P_{e_E} \frac{x}{L}} - 1}{e^{P_{e_E}} - 1}.$$

Let us note that the absolute values of ξ_2^i is increasing with i and the resulting solution displays an oscillatory spurious behaviour. In order to prevent this from happening, the easier solution is to increase the number of elements N in order to have $P_{e_E} < 1$.

7.1.1 Artificial viscosity and upwinding

Let us consider the origin of oscillations; in particular, let us consider a given mesh size h corresponding to $P_{e_E} >> 1$ and let us consider the diffusion parameter that would yield a mesh Peclet number equal to 1:

$$\frac{\beta h}{2(\nu + \tau)} = 1,$$

where the parameter τ is the *artificial viscosity* and is $\tau = \nu + \frac{\beta h}{2}$, but considering that $P_{e_E} >> 1$ implies that $\nu << \frac{\beta h}{2}$, which means that we can neglect ν so that $\tau = \frac{\beta h}{2}$.
With this parameter, the equation (35) becomes

$$-\nu \, \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \beta \, \frac{u_i - u_{i-1}}{h} = 0, \quad i = 1, \ldots, N-1. \tag{36}$$

In this equation the discretization of the convective term is a one side upwind discretization. Namely, the first derivative is approximated by a different term which takes into account only the point u_{i-1} that is “touched” by the wind. In this way, the stabilization has also a “physical” meaning.

Now we focus on the stabilization of a 2D Convection Diffusion problem. As we have seen, in order to prevent the oscillations we can reduce the mesh size (i.e. the Peclet number). In several cases, this is not possible and for this reason we now describe two techniques for the stabilization of the problem: *Streamline viscosity method* and *Streamline Upwind Petrov Galerkin.*

7.1.2 Stabilization in 2D: Streamline viscosity

The Streamline viscosity method consists on a generalization of the artificial viscosity: the key idea is to add a viscosity that acts only on the tangent direction of the convective velocity (not in the orthogonal one). For this purpose, we define a diffusion tensor τ; it is possible to show (with some mathematical steps) that the addition of a tangent viscosity is equal to the addition of the following term in the equation:

$$-\int_\Omega \nabla \cdot (\tau \nabla u)\, v \, d\Omega = \int_\Omega \frac{\tau}{||\beta||^2}\, \beta^T \nabla u \cdot \beta^T \nabla v \, d\Omega$$

where $\beta^T \nabla u$ is the directional derivative of u along β.
This approach is based on an inconsistent pertubation, in the sense that it does not preserve the rate of convergence when FEM of high order is considered. For this reason, we will use it in our example only for the stabilization of the problem with $\mathbb{P}_1$ elements and we introduce the following other stabilization techinque.

7.1.3 Stabilization in 2D: Streamline Upwind Petrov Galerkin (SUPG)

The formulation of this stabilization method is based on an artificial viscosity only in the direction of the convection velocity and in order to preserve the rate of the convergence of finite element methods of order higher than one it includes in the stabilization not only the streamline diffusion term, but the full residual of the equation:

$$\begin{aligned}&\int_\Omega \nu \nabla u_h \nabla \Phi_j \, d\Omega + \int_\Omega \beta \cdot \nabla u_h \, \Phi_j \, d\Omega + \sum_{E \in \mathcal{T}} \tau_E \int_E (\nabla \cdot (\nu \nabla u_h) + \beta^T \nabla u_h) \cdot \beta^T \nabla \Phi_j \, d\Omega \\ &= \int_\Omega f \, \Phi_j \, d\Omega + \sum_{E \in \mathcal{T}} \tau_E \int_E f \, \beta^T \nabla \Phi_j \, d\Omega, \quad \forall j = 1, \ldots, N_{\text{dof}}.\end{aligned} \tag{37}$$

Noting that the stabilization is computed cell-wise because the numerical solution is not sufficiently regular. The choice of the parameter τ_E is given by:

$$\tau_E == \begin{cases} m_k \dfrac{h_E^2}{4\nu}, & 0 \leq P_{e_E} \leq 1, \quad \text{diffusion dominant;} \\[2ex] \dfrac{h_E}{2\|\beta\|_E}, & P_{e_E} \geq 1, \quad \text{convection dominant.} \end{cases} \tag{38}$$

where $P_{e_E} = \frac{m_k ||\beta|| h_E}{2\nu}$ is the Peclet number associated to each element and the parameter m_k is related to an inverse inequality. If $k = 1$ ($\mathbb{P}_1$ elements), $m_k = \frac{1}{3}$; if $k = 2$ ($\mathbb{P}_2$ elements), $m_k = \frac{1}{24}$.

7.2 *Stabilization for a Convection Diffusion problem in Matlab*

The Matlab code we implemented for the stabilization of the Convection-Diffusion problem is an extension of the function "Sistema" previously mentioned, called *"Sistema stabilizzato"* which builds the linear system of our problem with the same procedure described in the previous sections. The main differences are related to the addition of the stabilization terms for the definition of the matrices D, C, R (used to obtain the stiffness matrix A) and of the right-hand side f. As described in the previous section, we define two procedures for the stabilization of the problem: the addition of a streamline viscosity and the Streamline Upwind Petrov Galerkin method.
The first one is used in the code when the condition *"if flagP == 1"* is true; indeed, it is a first order method for the stabilization of the problem and we use it with linear Lagrangian finite elements. In this case "Sistema stabilizzato" only modifies the matrix D by adding the contribution given from the streamline viscosity.
For the second one (*"if flagP == 2"*), the function modifies the matrices D, C, R and the right-hand side f in order to include the full residual of the equation (see (37)). In this case, we also define a new global variable called "Grad-fi-beta" in which we save all the quantities $\nabla(\beta^T \nabla \Phi_j)$, $j = 1, \ldots, \mathrm{N}_{\text{dof}}$ which are used in the integrals that define the matrix D.
Finally, in both cases we calculate the variables *Pe triangolo*, h_E, τ_E defined in (38) (from line 21, up to line 39 of the following code).

```
1 function [D,C,R, A_D, b, b_N, tau_E]=sistema_stabilizzato(nu, beta,
     gamma,f, g_N1, g_N2, flag_P, flag_DN, flag_c)
2 global Ndof NE Ele XY Pivot ND Ne NN N_L Area G I fi grad_fi x_nodi
     y_nodi pesi grad_fi_beta
3
4 %same code as the function "sistema" shown in section "Second-order
     elliptic problem" from line 6, up to line 13
```

```matlab

nu_const = 1/6/10^(6);
beta_const = [1;0];
norma_beta = norm(beta_const,2);
coefficiente_Peclet = 1/24;

 %same code as the function "sistema" shown in section "Second-order elliptic problem" from line 15, up to line 25

    a_1 = [XY(Ele(e, 1),1);XY(Ele(e, 1),2)];
    a_2 = [XY(Ele(e, 2),1);XY(Ele(e, 2),2)];
    a_3 = [XY(Ele(e, 3),1);XY(Ele(e, 3),2)];

    lato_1 = norm(a_1 - a_2);
    lato_2 = norm(a_2 - a_3);
    lato_3 = norm(a_1 - a_3);

    h_E = lato_1;

    if (lato_2 >= h_E)
        h_E = lato_2;
    end

    if (lato_3 >= h_E)
        h_E = lato_3;
    end

    area = Area(e);

    Pe_triangolo = coefficiente_Peclet*(norma_beta.*(h_E))/(2*nu_const);

    if (Pe_triangolo >= 1)
        tau_E = h_E/(2*norma_beta);
    else
        tau_E = (coefficiente_Peclet.*(h_E.^2))/(4.*nu_const);
    end

    %same code as the function "sistema" shown in section "Second-order elliptic problem" from line 29, up to line 33

    % First order stabilization.

    if flag_P==1
        for j=1:size(Ele,2) %3
        jj=Pivot(Ele(e,j));
        if(jj>0)
            for k=1:size(Ele,2) %3

                diff=0;
                conv=0;
                reaz=0;
                t_noto=0;

                for p = 1:length(x_nodi)
                    % Let's modify the matrix D by adding the
```

```
contribution due to the streamline viscosity.

                    diff = diff + 2*area*pesi(p)*...
                        (nu(x_punti_esatti(p,1),y_punti_esatti(p,1)))*...
                        (grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,k))'*B_inv*B_T*...
                        (grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,j)) +...
                        2*area*pesi(p)*(tau_E)/(norma_beta^2)*...
                        ((grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,k))'*beta_const)*...
                        (grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,j))'*beta_const;

                    conv= conv+2*area*pesi(p)*...
                        (beta(x_punti_esatti(p),y_punti_esatti(p)))'*B_T*...
                        (grad_fi(x_nodi(p),y_nodi(p))*I(:,k))*...
                        (fi(x_nodi(p),y_nodi(p))*I(:,j));

                    reaz=reaz+2*area*pesi(p)*...
                        gamma(x_punti_esatti(p),y_punti_esatti(p))*...
                        (fi(x_nodi(p),y_nodi(p))*I(:,k))*...
                        (fi(x_nodi(p),y_nodi(p))*I(:,j));

                    if flag_P==1 && flag_DN==1 && flag_c==0
                        t_noto=area*(f(G(e,1), G(e,2))/3);
                    else
                        t_noto=t_noto+2*area*pesi(p)*f(x_punti_esatti(p),y_punti_esatti(p))*...
                         (fi(x_nodi(p),y_nodi(p))*I(:,j));
                    end
                end
     %same code as the function "sistema" shown in section "Second-order elliptic problem" from line 630, up to line 74

    %Petrov Galerkin stablization.
    elseif flag_P==2
        for j=1:size(Ele,2) %3
        jj=Pivot(Ele(e,j));
        if(jj>0)
            for k=1:size(Ele,2) %3

                diff=0;
                conv=0;
                reaz=0;
                t_noto=0;

                for p = 1:length(x_nodi)

                    diff = diff + 2*area*pesi(p)*...
                        (nu(x_punti_esatti(p,1),y_punti_esatti(p,1)))*...
                        (grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,k))'*B_inv*B_T*...
                        (grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,j)) +
                           ... %stabilizzazione
                       2*area*pesi(p)*tau_E*(nu(x_punti_esatti(p,1),y_punti_esatti(p,1))*...
                        (grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,k))'*B_inv*B_T*...
                        (grad_fi_beta(x_nodi(1,p),y_nodi(1,p))*I(:,j)));

                    conv= conv+2*area*pesi(p)*...
                        (beta(x_punti_esatti(p),y_punti_esatti(p)))'*B_T*...
                        (grad_fi(x_nodi(p),y_nodi(p))*I(:,k))*...
                        (fi(x_nodi(p),y_nodi(p))*I(:,j)) +...
                        2*area*pesi(p)*tau_E*(((B_T*grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,k)))'*beta_const)*...
                        (((B_T*grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,j))))'*beta_const;

                    reaz=reaz+2*area*pesi(p)*...
                        gamma(x_punti_esatti(p),y_punti_esatti(p))*...
                        (fi(x_nodi(p),y_nodi(p))*I(:,k))*...
                        (fi(x_nodi(p),y_nodi(p))*I(:,j));
                        t_noto=t_noto+2*area*pesi(p)*f(x_punti_esatti(p),y_punti_esatti(p))*...
                            (fi(x_nodi(p),y_nodi(p))*I(:,j)) +... % stabilizzazione
                            2*area*pesi(p)*tau_E*f(x_punti_esatti(p),y_punti_esatti(p))*...
                            ((B_T*grad_fi(x_nodi(1,p),y_nodi(1,p))*I(:,j)))'*beta_const;
                    end

                end
        %same code as the function "sistema" shown in section "Second-order elliptic problem" from line 60, up to line 101

    elseif flag_P==2
        i_m=N_L(l, 5); %indice del punto medio del lato e
        ii_m=Pivot(i_m);
        gb=g_N(XY(i_b,1),XY(i_b,2));
        ge=g_N(XY(i_e,1),XY(i_e,2));
        gm=g_N(XY(i_m,1), XY(i_m,2));
        length_lato=norm(abs(XY(i_e,:)-XY(i_b,:)),2);

        gb_stab = g_N(XY(i_b,1),XY(i_b,2))*((beta_const')*(grad_fi(XY(i_b,1),XY(i_b,2))*I(:,j)));
        gm_stab = g_N(XY(i_m,1),XY(i_m,2))*((beta_const')*(grad_fi(XY(i_m,1),XY(i_m,2))*I(:,j)));
        ge_stab = g_N(XY(i_e,1),XY(i_e,2))*((beta_const')*(grad_fi(XY(i_e,1),XY(i_e,2))*I(:,j)));

        if(ii_b>0)

               b_N(ii_b)=b_N(ii_b)+((2*gb+gm-0.5*ge)*length_lato)/15;

               % stabilizzo
```

```
                b_N(ii_b) = b_N(ii_b)+tau_E*((2*gb_stab+gm_stab-0.5*
                    ge_stab)*length_lato)/15;
            end

            if(ii_m>0)
                b_N(ii_m)=b_N(ii_m)+((gb+8*gm+ge)*length_lato)/15;
                b_N(ii_m)=b_N(ii_m)+tau_E*((gb_stab+8*gm_stab+ge_stab)*
                    length_lato)/15;
            end

            if(ii_e>0)
                b_N(ii_e)=b_N(ii_e)+((gm+2*ge-(1/2)*gb)*length_lato)
                    /15;
                b_N(ii_e)=b_N(ii_e)+tau_E*((gm_stab+2*ge_stab-(1/2)*
                    gb_stab)*length_lato)/15;
            end
        end
end
```

8 Stabilization for Convection Diffusion problems: our example

We are now going to explain in detail the stabilization of a specific Convection Diffusion problem. Particularly, we considered the following mixed Dirichlet-Neumann boundary-value problem with constant coefficients:

$$\begin{cases} -\nabla \cdot (\nu \nabla u) + \beta \cdot \nabla u = f \quad \text{in } \Omega = [0,1] \times [0,1] \\ \\ u = g_D \quad \text{on } \Gamma_D \\ \\ \nu \dfrac{\partial u}{\partial n} = g_N \quad \text{on } \Gamma_N \end{cases} \tag{39}$$

with $\nu = \frac{1}{6}10^{-6}$, $\beta = [1;0]$, f such that:

$$u(x,y) = 16\,x\,(1-x)\,y\,(1-y)\,e^{ax} + x + y, \quad a > 0$$

is the solution of the problem (39). Namely, we consider a problem which is very similar to the one we studied previously: the solution u of the "old" problem was multiplied by the exponential function. In particular, the parameter a will be set to a specific value for each stabilization. The intuitive idea is that we are going to study a solution which is very steep near the boundary of the domain (specifically, in the region $[1-\epsilon] \times [0,1]$, for ϵ sufficiently small). This is a key aspect for the stabilization of the problem, as we have seen in the theoretical introduction above. Finally, we want to highlight the values chosen for the constant coefficients ν and β: in particular, the small value that the constant ν assumes is fundamental for the stabilization of the problem, since $P_{e_E} >> 1$ (see (38)). We will calculate the approximate solution with and without stabilization in order to show how the stabilization process contributes to remove the oscillations of the numerical solution.

8.1 *Streamline viscosity: solution of the problem with $\mathbb{P}_1$ elements*

In problem (39), we considered $a = 5$, so that the solution looks like the following Figure 12

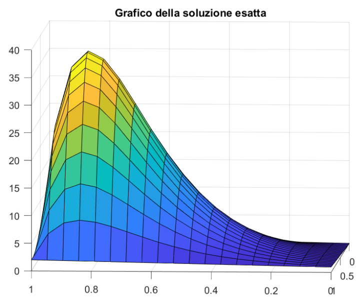


Figure 12: Exact solution of problem (39) (with $a = 5$). This function is considered as exact solution in the stabilization of the problem with streamline viscosity.

For the stabilization of the problem we used the streamline viscosity implemented in *"Sistema stabilizzato"* (flagP $= 1$). The solution is referred to four iterations in which we halve the parameter "element area" (the maximal value of the area of each triangle). The initial value of "element area" is set equal to 0.08. The following figures represent the numerical solutions obtained by solving the problem and the errors linked to each one of them:

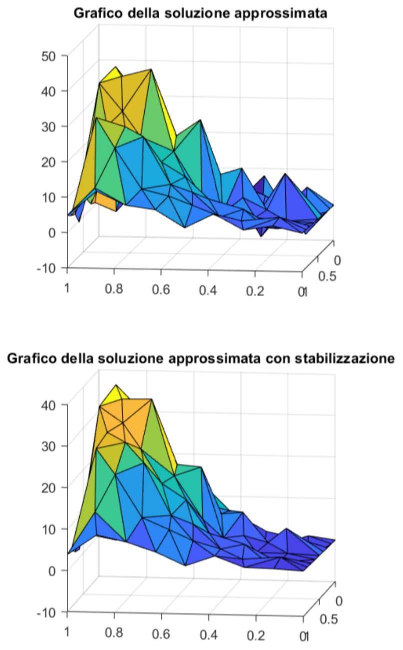
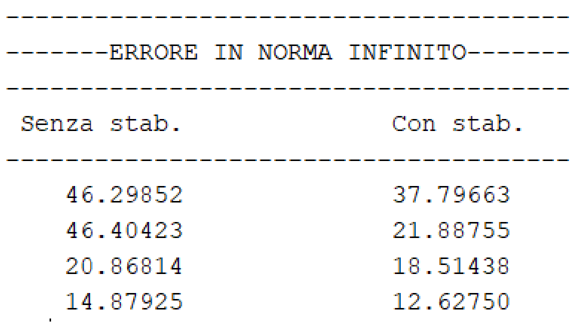


Figure 13: Numerical solutions obtained without stabilization (first graph) and with stabilization (second one, with streamline viscosity).



```
---------------------------------------
-------ERRORE IN NORMA INFINITO-------
---------------------------------------
 Senza stab.                 Con stab.
---------------------------------------
    46.29852                 37.79663
    46.40423                 21.88755
    20.86814                 18.51438
    14.87925                 12.62750
```

Figure 14: Norm L^∞ errors without stabilization (left) and with stabilization (right) in each iteration.

We observe that the stabilization seems to reduce the oscillations of the solution, both in the figure showing the graphs (see (13)) and in the L^∞ norm error, which for continuous functions is nothing but the maximum norm (see Figure (14)).

8.2 *Streamline Upwind Petrov Galerkin: solution of the problem with $\mathbb{P}_2$ elements*

In problem (39), we considered $a = \frac{13}{2}$, so that the solution looks like the following Figure 15 is :

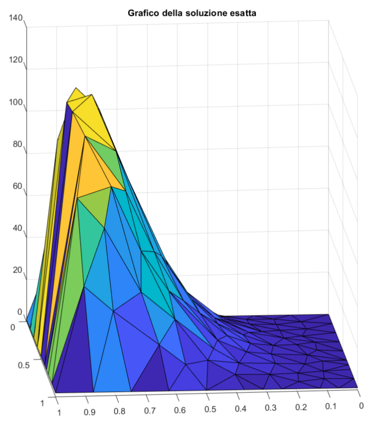


Figure 15: Exact solution of problem (39) (with $a = \frac{13}{2}$). The figure refers to the partition of the domain that we obtained at the fourth iteration. This function is considered as exact solution in the stabilization of the problem with the streamline upwind Petrov Galerkin method.

For the stabilization of the problem we use the streamline upwind Petrov Galerkin method implemented in *"Sistema stabilizzato"* (flagP = 2). The solution is referred to four iterations in which we halve the parameter "element area" . The initial value of "element area" was set equal to 0.06. The following figures represent the numerical solutions obtained by solving the problem and the errors linked to each one:

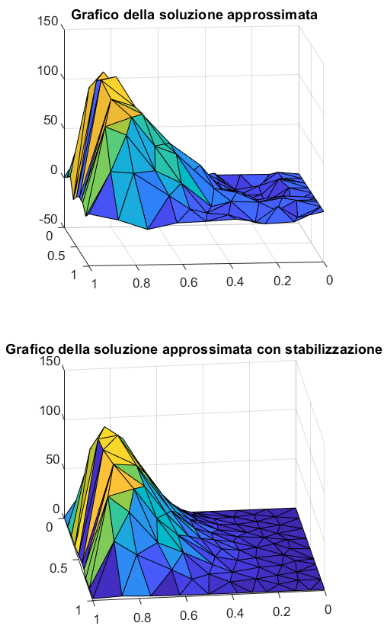
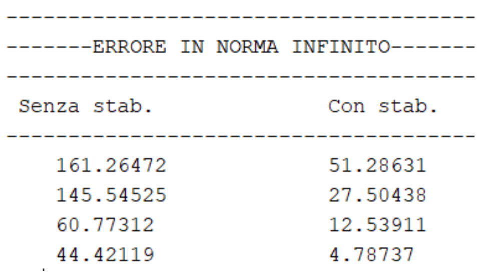


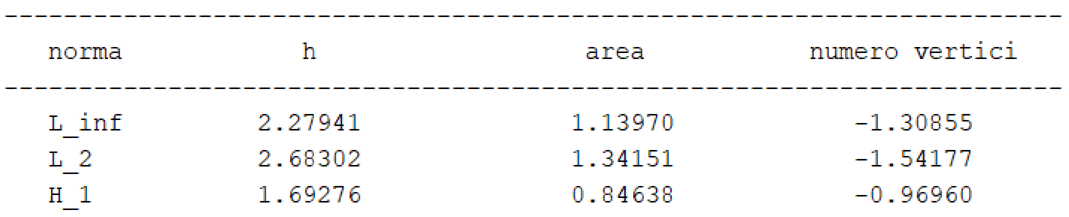
Figure 16: Numerical solutions obtained without stabilization (first graph) and with stabilization (second one, with SUPG).



```
---------------------------------------
-------ERRORE IN NORMA INFINITO-------
---------------------------------------
 Senza stab.                Con stab.
---------------------------------------
    161.26472               51.28631
    145.54525               27.50438
    60.77312                12.53911
    44.42119                4.78737
```

Figure 17: Norm L^∞ errors without stabilization (left) and with stabilization (right) for each iteration.

We observe that the stabilization seems to reduce the oscillations of the solution, both in the figure showing the graphs ((16)) and in the error one ((17)). This reduction is stronger than the previous one, as we could expect. In fact, this stabilization uses an higher polinomial degree ($p = 2$) and also it is consistent with second order FEM. We want to highlight again this key aspect by showing the convergence orders of the stabilized solution:



```
----------------------------------------------------------------------
   norma             h                    area             numero vertici
----------------------------------------------------------------------
   L_inf         2.27941              1.13970              -1.30855
   L_2           2.68302              1.34151              -1.54177
   H_1           1.69276              0.84638              -0.96960
```

Figure 18: Convergence orders in the L^∞, L^2, H_1 norms, with respect to the maximal diameter of the triangles, the maximal area and the total number of vertices of the partition.

As we can see from figure (18), including in the stabilization the full residual of the equation (as the Petrov Galerkin method does) allows us to preserve the rate of convergence for a FEM of order 2 (and of order k, in general), since the obtained numbers are similar to the theoretical convergence orders of the method.
Finally, we conclude this paragraph by mentioning the second paragraph in the appendix II for an equivalent interpretation of the SUPG method as an extension of the Galerkin framework.

9 Stabilization for Reaction Diffusion problems

9.1 *A brief theoretical introduction*

Let us consider the simple reaction diffusion equation with positive coefficients ν and σ defined in the interval $(0, L) \subset \mathbb{R}$.

$$\begin{cases} -\nu \dfrac{\partial^2 u}{\partial x^2} + \sigma u = 0 \quad 0 < x < L \\ u(0) = u_0 \\ u(L) = u_L. \end{cases} \tag{40}$$

Let us consider a uniform partition of the domain $(0, L)$ in N intervals with length $h = \frac{L}{N}$.
Given a finite element discretization with linear elements, we get:

$$\begin{aligned} -\nu \, \frac{u_{i+1} - 2u_i + u_{i-1}}{h} + \frac{\sigma \, h}{6} \, (u_{i+1} - 4u_i + u_{i-1}) = 0, \quad i = 1, \ldots, N-1 \\ u_0 = u_0 \\ u_N = u_L \end{aligned} \tag{41}$$

Considering the dimentionless number

$$S_{e_E} = \frac{\sigma h^2}{6\nu},$$

defined with the scale length of the cells, then (41) can be written as

$$(S_{e_E} - 1) \, u_{i+1} + 2 \, (2S_{e_E} + 1) \, u_i + (S_{e_E} - 1) \, u_{i-1} = 0.$$

Looking for a solution with the structure $u_i = \text{const} \, \xi^i$ the equation can also be written as

$$\xi^{i-1} \left[(S_{e_E} - 1) \, \xi^2 + 2 \, (2S_{e_E} + 1) \, \xi + (S_{e_E} - 1) \right] = 0, \tag{42}$$

that is satisfied by the values

$$\begin{aligned} \xi_0 &= 0, \\ \xi_1 &= \frac{(2 \, S_{e_E} + 1) + \sqrt{3 \, S_{e_E} \, (S_{e_E} + 2)}}{S_{e_E} - 1}, \\ \xi_2 &= \frac{-(2 \, S_{e_E} + 1) - \sqrt{3 \, S_{e_E} \, (S_{e_E} + 2)}}{S_{e_E} - 1} \end{aligned}$$

We expect the same oscillatory behaviour noticed previously in the convection diffusion equation when $P_{e_E} > 1$. The constant values are given by the boundary condition and the exact solution of the `discrete` difference problem comes to be

$$u_i = \frac{u_L \, (\xi_1^i - \xi_2^i) - u_0 \, (\xi_2^N \, \xi_1^i - \xi_1^N \, \xi_2^i)}{\xi_1^N - \xi_2^N} \tag{43}$$

whereas the exact solution to the `continuous` differential problem is

$$u(x) = \frac{u_L - u_0 \, e^{-\lambda L}}{e^{\lambda L} - e^{-\lambda L}} \, e^{\lambda x} + \frac{u_0 \, e^{-\lambda L} - u_L}{e^{\lambda L} - e^{-\lambda L}} \, e^{-\lambda x} = \frac{u_L \, (e^{\lambda x} - e^{-\lambda x}) - u_0 \, (e^{-\lambda L} e^{\lambda x} - e^{\lambda L} e^{-\lambda x})}{e^{\lambda L} - e^{-\lambda L}}.$$

Let us now consider a different discretization, in particular a finite difference dicretization with centered difference with the same boundary condition as in (41).

$$-\nu \, \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \sigma \, u_i = 0, \quad i = 1, \ldots, N-1$$

that can also be written as

$$\xi^{i-1} \left[\xi^2 - (S_{e_E} + 2) \, \xi + 1 \right] = 0$$

satisfied by the values

$$
\begin{aligned}
\xi_0 &= 0 \\
\xi_1 &= \frac{(S_{e_E}+2)+\sqrt{S_{e_E}\,(S_{e_E}+4)}}{2} \\
\xi_2 &= \frac{(S_{e_E}+2)-\sqrt{S_{e_E}\,(S_{e_E}+4)}}{2}
\end{aligned}
$$

that leads to the same solution (43).
The finite difference discretization does not suffer of spurious oscillations also for S_{e_E} very large thanks to the fact that the reaction term does not involve values of the solution in the neighboring nodes. The oscillations rise close to the boundary where the solution takes values different from the values inside the domain.
This good behaviour of the finite difference discretization can be introduced also in the finite element discretization by the **mass lumping** process in the discretization of the reaction term that provides the same difference equation of the finite difference discretization. The mass lumping can be considered a stabilization for the reaction-diffusion equation discretized by finite elements.

9.2 ***Stabilization for Reaction Diffusion problem in Matlab***

As it's just been explained in the previous section, the stabilization of the Reaction-Diffusion problem consists on applying the mass lumping technique to the reaction matrix R. This one is simply implemented as follows: for each row of the reaction matrix R, output of the function "Sistema" mentioned before, every row entry is summed and placed in the diagonal element of the row we're taking into account.

```
[D,C,R, A_D,b, b_N]=sistema(nu, beta, gamma,f, g_N1, g_N2,flag_P,
    flag_D, flag_c);

%  mass lumping
   r=sum(R')';
   R=diag(r);
```

10 Stabilization for Reaction Diffusion problems: our example

We now want to explain in detail the stabilization of a specific Reaction-Diffusion problem. We considered the following Dirichlet-Neumann boundary-value problem with constant coefficients:

$$
\begin{cases}
-\nabla\cdot(\nu\,\nabla u)+\sigma\,u=f \quad \text{in } \Omega=[0,1]\times[0,1] \\
\\
u=g_D \quad \text{on } \Gamma_D \\
\\
\nu\,\dfrac{\partial u}{\partial n}=g_N \quad \text{on } \Gamma_N
\end{cases}
\tag{44}
$$

with $\nu=\frac{1}{6}10^{-5}$, $\sigma=1$, f such that:

$$
u(x,y)=16\,x\,(1-x)\,y\,(1-y)\,e^{6x}+x+y
$$

is the solution of problem (44). Namely, we considered the same problem studied for the stabilization of a Convection-Diffusion problem: the idea and reasoning behind this particular choice in the exact solution is the same explained above (it is very steep near the boundary of the domain). In order to visualize the function, we can refer to figure (15) or (12) (even if the exponential function is different in both cases, the behaviour of the solution is substantially the same). We also want to highlight the values of the constant coefficients

ν and σ: as we have seen in the Convection-Diffusion problem and also in the theoretical introduction to Reaction-Diffusion one, the small value of ν ($\nu = 10^{-5}$ in this scenario) makes fundamental the stabilization of the problem with mass lumping technique, since $S_{e_E} >> 1$.
We considered the solution of this problem with the use of $\mathbb{P}_1$ elements; in this way the mass lumping makes the two approaches, FEM and finite differences, equivalent. The numerical solution is referred to four iterations in which we halve the parameter "element area" (the maximal value of the area of each triangle). The initial value of "element area" was set equal to 0.2.

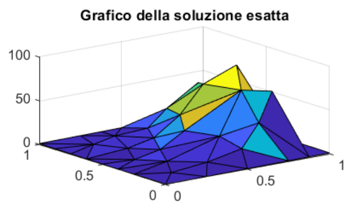


Figure 19: exact solution of the problem (44). The figure refers to the partition of the domain that we obtain at the fourth iteration

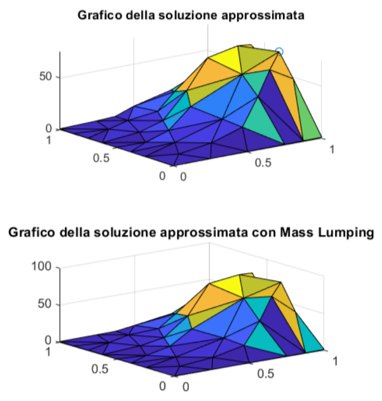
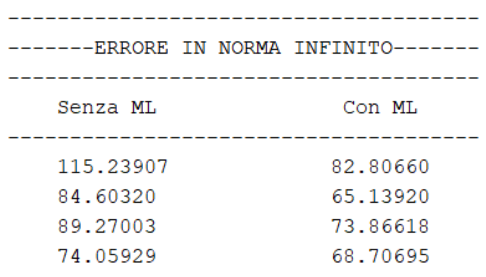


Figure 20: numerical solutions obtained without stabilization (the first figure) and with stabilization (the second one). $S_{e_E} = 2500$ at the fourth iteration



```
---------------------------------------
-------ERRORE IN NORMA INFINITO-------
---------------------------------------
    Senza ML                Con ML
---------------------------------------
    115.23907             82.80660
    84.60320              65.13920
    89.27003              73.86618
    74.05929              68.70695
```

Figure 21: error in the norm of L^∞ without stabilization (left) and with stabilization (right) for each iteration.

The behaviour of the stabilized numerical solution seems to be more similar to the exact solution one; furthermore, the errors in the maximum norm are lower than in the non-stabilized numerical solution. On the other hand, we also observe that the mass lumping technique is not fully able to prevent all the oscillations and it doesn't prevent from a great error in maximum norm to happen.

Appendices

Part I

Lax-Milgram theorem

Theorem 1 *Lax-Milgram[2] Assume that the bilinear form $a : V \times V \to \mathbb{R}$ is continuous and coercive, and that the linear form $F : V \to \mathbb{R}$ is continuous. Then the variational problem*

$$[VP]\begin{cases} Find\ u \in V such\ that \\ a(u,v) = F(v), \quad \forall v \in V \end{cases}$$

admits one and only one solution u, which satisfies

$$\|u\|_V \leq \frac{1}{\alpha}\|F\|_{V'}.$$

The Theorem states that $[VP]$ is well-posed in the sense of Hadamard.

Part II

Approximate bilinear and linear forms

In the applications to partial differential equations, the bilinear form a and the linear form F are defined by means of integrals. Often, it is not possible or convenient to compute these integrals exactly, and numerical integration formulas have to be used. This is what has happened in our discretized problem, in which we used fifth order quadrature formulas in order to compute the integrals. Since such formulas are not exact, they lead to approximate bilinear and linear forms, here indicated by a_δ and F_δ.
Thus, we assume that our bilinear form takes the form $a_\delta : V_\delta \times V_\delta \to \mathbb{R}$ and our linear form $F_\delta : V_\delta \to \mathbb{R}$ so that the approximate variational problem can be described by

$$[VP]_\delta \begin{cases} \text{Find } \mathrm{u}_\delta \in \mathrm{V}_\delta \text{ such that} \\ a_\delta(u_\delta, v_\delta) = F_\delta(v_\delta), \quad \forall v_\delta \in V_\delta. \end{cases} \tag{45}$$

This is the natural setting for a Galerkin method with *numerical integration* (see [2]). The variational formulation of a problem with an approximate bilinear form is accompained by another important hypotesis, which is the discrete coercivity:

$$a(u_\delta, u_\delta) \geq \alpha_\delta ||u_\delta||^2, \quad \forall u_\delta \in V_\delta$$

This condition is said to be *uniform* if exists a α_0 such that $\alpha_0 <= \alpha_\delta$, for all δ. The uniform discrete coercivity ensures the numerical stability of the problem; on the other hand, several other hypotesis on the approximate linear and bilinear forms ensure the consistency of the problem. From the Lax-Richmyer Theorem stability and consistency ensure the convergence of the method.
Now we focus on our problem showing that the uniform discrete coercivity is granted. In general, if we use finite elements of order k, this condition is satisfied if the quadrature formula has order $q \geq 2k - 2$. In our case, $q = 5$:

$$5 = q \geq 2k - 2 = 0, \quad \text{for} \quad \mathbb{P}_1 \quad \text{elements}$$

$$5 = q \geq 2k - 2 = 2, \quad \text{for} \quad \mathbb{P}_2 \quad \text{elements}$$

Non conforming methods

Sometimes one gains in flexibility and efficiency by removing the requirement that the finite dimensional space V_δ is contained in V: V_δ is just required to be a subspace of a larger space H, in such that $V \subset H$. In this case, the bilinear form a and the linear form F must be extended in such a way that they are defined on V_δ. Denoting by a_δ and F_δ these extensions, we formally end up with the same variational problem as (45).
In the finite element method, an external approximation occurs if one relaxes the continuity requirement between contiguous elements.
This formalism was necessary when we applied the Streamline Upwind Petrov Galerkin in which we used test functions defined as $v + \tau_E \, \beta \, \nabla v$ (see Figure 22).

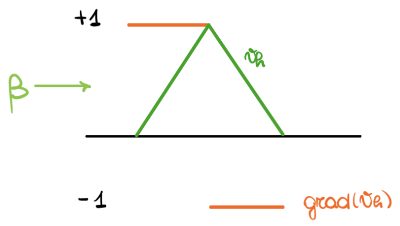


Figure 22: Test function in the Streamline Upwind Petrov Galerkin [3]

References

[1] A. Barbera and Stefano Berrone. “BBTR: an unstructured triangular mesh generator”. In: *Quaderni del Dipartimento di Matematica, Politecnico di Torino* (2008).

[2] Stefano Berrone and Claudio Canuto. *Notes of the Lecture Course, Numerical Methods for Partial Differential Equations*. 2022.

[3] Francesca Marino. *Metodi numerici per le PDE*. 2022-2023.