UNIVERSITY OF PADOVA

**Civil, Environmental and Architectural department DICEA**

Master's degree in Mathematical Engineering

# NMCS Project 1

Andrea Gorgi, Badge 2010658

Academic year 2020/2021

# Contents

# Introduction

We consider the convection-diffusion (or advection-diffusion) equation, governing, for example, the fate of a contaminant dissolved in a fluid moving with velocity $q(x,t)$ in a domain $\Omega \subset \mathbf{R}^d$, $d = 1,2,3$, as:

$$\begin{cases} \dfrac{\partial u}{\partial t} = div(D\nabla u) - div(\beta u) + f & in\, \Omega \times (0,T); \\ u(x,0) = u_0(x) & x \in \Omega, t = 0; \\ u(x,t) = U_D(x,t) & x \in \Gamma_D, t > 0; \\ D\nabla u \cdot \vec{n} = q_N(x,t) & x \in \Gamma_N, t > 0; \\ (\beta + D\nabla c) \cdot \vec{n} = q_c(x,c) & x \in \Gamma_C, t > 0; \end{cases} \tag{1}$$

where:

- $u[-]$: colute concentration (generally normalized to 1);
- $x[L]$: spatial coordinate with respect to a Eulerian reference system (for d = 2: $x = (x_1, x_2)$);
- $t[T]$: time;
- $\Omega \subset \mathbf{R}^d$ domain ($d = 1, 2,$ *or* $3$);
- $\Gamma = \Gamma_D \cup \Gamma_N \cup \Gamma_C$ boundary of $\Omega$
- $\Gamma_D$: Dirichlet boundary;
- $\Gamma_N$ : Neumann boundary;
- $\Gamma_C$: Robin (or Cauchy) boundary;
- $K[L^2/T]$: diffusion coefficient (assumed a scalar function of (x, t));
- $f[1/T]$: forcing function;
- $u_0(x)$: initial concentration;
- $u_D(x,t)$: value of the Dirichlet boundary condition;
- $q_N(x,t)$: value of the diffusive flux at Neumann boundary;
- $q_C(x,t)$: value of total flux at Cauchy boundary.

We want to solve the convection-diffusion equation in the stationary case ($\partial u/\partial t = 0$) in the domain $\Omega = \{(x,y) : 0 \le x \le 1, 0 \le y \le 1\}$, with constant diffusion coefficient and velocity, and with zero forcing. The data and boundary conditions are appropriately chosen so that a boundary layer is obtained. Thus we want to solve:

$$\begin{cases} \dfrac{\partial}{\partial x}(D\dfrac{\partial u}{\partial x}) + \dfrac{\partial}{\partial y}(D\dfrac{\partial u}{\partial y}) - \dfrac{\partial \beta_x u}{\partial x} - \dfrac{\partial \beta_y u}{\partial y} = 0 \quad (x,y) \in \Omega \\ u(x,t) = 1 \qquad x \in \Gamma_1, t > 0 \\ u(x,t) = 0 \qquad x \in \Gamma_2, t > 0 \end{cases}, \tag{2}$$

where the boundaries $\Gamma_1$ and $\Gamma_2$ are given by:

$$\begin{aligned} \Gamma_1 = \{(x,y) : [x = 0, 0 \le y \le 1] \cup [0 \le x \le 0.3, y = 0]\} \\ \Gamma_2 = \{(x,y) : [x = 1, 0 \le y \le 1] \cup [0.3 \le x \le 1, y = 0]\} \end{aligned} \tag{3}$$

## 1 Weak and Galerkin formulation

It is well known that to obtain the weak formulation of a problem we multiply both sides of the equation by a test function $v \in \mathcal{V}$ and integrate the result over the domain $\Omega$. In our case we choose $\mathcal{V} = H^1_{\Gamma_D}(\Omega) = \{v(x) : \Omega \to \mathbb{R}$ such that $v(x) \in L^2(\Omega), \frac{\partial v(x)}{\partial x_i} \in L^2(\Omega)$ for $i = 1, 2$, $v(x) = 0 \in \Gamma_D\}$, $\Gamma_D$ Dirichlet boundary. The weak formulation is then

$$-\int_\Omega div(D\nabla u)v d\Omega + \int_\Omega div(\beta u)v d\Omega = 0, \tag{4}$$

which, using Green's Lemma, and recalling that $v \in \mathcal{V}$ implies $v = 0$ in $\Gamma_D$, becomes

$$\int_\Omega (D\nabla u) \cdot \nabla v d\Omega - \int_{\Gamma_N} v(D\nabla u) \cdot n d\sigma + \int_\Omega div(\beta u)v d\Omega = 0, \tag{5}$$

where $n$ is the external normal to the domain. Now, to obtain the Galerkin formulation, we choose to discretize the test space with

$$\mathcal{V}_h = \{v \in C^0(\bar{\Omega}) : v|_T \in \mathcal{P}^1(T) \forall T \in \mathcal{T}_h\}, \tag{6}$$

where $\mathcal{T}_h$ is the given triangulation. Hence, each function in the basis of $\mathcal{V}_h$, will have the form

$$\phi_i^T(x, y) = a_i^T + b_i^T x + c_i^T y. \tag{7}$$

Considering the approximate solution $u_h$ given by a linear combination of the basis of $\mathcal{V}_h$,

$$u_h(x, y) = \sum_{i=1}^N u_i \phi_i(x, y), \tag{8}$$

$u_i$ nodal values of the solution, and N number of degrees of freedom in our problem, dimension of our test space, we can build the Galerkin formulation,

$$\int_\Omega (D\nabla u_h) \cdot \nabla v d\Omega - \int_{\Gamma_N} v(D\nabla u) \cdot n d\sigma + \int_\Omega div(\beta u)v d\Omega = 0, \quad \forall v \in \mathcal{V}_h, \tag{9}$$

which more specifically brings to

$$\int_\Omega (D\nabla u_h) \cdot \nabla \phi_i d\Omega - \int_{\Gamma_N} \phi_i(D\nabla u) \cdot n d\sigma + \int_\Omega div(\beta u)\phi_i d\Omega = 0, \quad i = 1, ..., N. \tag{10}$$

Usually the term $(D\nabla u) \cdot n$ on $\Gamma_N$ is set by the Neumann boundary conditions to some known value $q(x, y)$. Anyway in our case no Neumann BCs are specified, and $\Gamma_N = \oslash$, therefore we remove that term from the computations. Substituting $u_h(x, y) = \sum_{i=1}^N u_i \phi_i(x, y)$ in this formulation we can easily construct the linear system of the problem, which is in the form

$$(H + B)u = 0, \tag{11}$$

with

$$H = \{h_{ij}\} \qquad h_{ij} = \int_\Omega (D\nabla\phi_j) \cdot \nabla\phi_i d\Omega$$

$$B = \{b_{ij}\} \qquad b_{ij} = \int_\Omega div(\beta\phi_j)\phi_i d\Omega.$$

In particular, id $D = \begin{bmatrix} D_{xx} & 0 \\ 0 & D_{yy} \end{bmatrix}$ and $\beta = [v_x, v_y]^T$, the formulation is exactly the one in the assignment paper,

$$H = \{h_{ij}\} \qquad h_{ij} = \int_\Omega \left[D_{xx}\frac{\partial\phi_j}{\partial x}\frac{\partial\phi_i}{\partial x} + D_{yy}\frac{\partial\phi_j}{\partial y}\frac{\partial\phi_i}{\partial y}\right] d\Omega$$

$$B = \{b_{ij}\} \qquad b_{ij} = \int_\Omega \left(v_x\frac{\partial\phi_j}{\partial x} + v_y\frac{\partial\phi_j}{\partial y}\right)\phi_i d\Omega.$$

We here remark that on a practical point of view, the $phi_i$ are the weight functions of the nodal values of the solution, and do not completely vanish only in the triangles where such node is vertex, For this reason both the $h_{ij}$ and $b_{ij}$, once fixed the $\phi_i$ function of the basis, will be non zero only if node $j$ is connected to node with an edge of the triangulation, that is, only if there is at least one triangle where both $i$ and $j$ are vertices. In all the other cases the entries will automatically be zero for both the stiffness and the transport matrix, giving them the same sparse structure.

## 2 Implementation notes and first results

To solve this problem a MATLAB code [1] has been implemented. The solution is computed by running the "main.m" script, which contains all the calls to the functions which loads the mesh and data of the problem, build and solve the linear system. Below is reported the code of "main.m"

```
%%% Finite element code for 2D piecewise linear Galekrin
%%% -div(Diff grad u)(x) + div(\beta u) = 0 with Dirichlet BC
clear all; close all; clc;

%% PARAMETERS DEFINITION
%---------------------------------------------------------------------------
% Define the 2D mesh and equation BC's and coefficients
flag_mesh = 0;        % flag_mesh = 0 ---> PRESCRIBED MESH IN THE FOLDER
                      %           = 1 ---> UNIFORM MESH WITH NXN NODES
meshdir   = 'mesh';
N         = 21;
seeMesh   = true;     % to see the mesh

tau = 0; % SUPG stablization factor

%---------------------------------------------------------------------------
[Nelem,Nodes,triang,coord,NDir,DirNod,DirVal,h]=input_data(flag_mesh,meshdir,N, seeMesh);
%---------------------------------------------------------------------------
%------------------------------
% physical parameters
%------------------------------

Diff  = 1.0*ones(Nelem,1);
Vel   = [1,3];

if seeMesh
   nfig = 2;
else
   nfig = 1;
end
%--------------------------------------------------------------------------

%% LINEAR SYSTEM CONSTRUCTION
% calculate element area and elemental coefficients of basis functions
% (b,c)
[Bloc,Cloc,Area] = localBasis(Nelem,triang,coord);

% build stiffness matrix (without BCs)
[SYSMAT,H] = BuildLSys(Nelem,Nodes,triang,Bloc,Cloc,Area,Diff,Vel,h);
%---------------------------------------------------------------------------
%% IMPOSE THE BCs
%---------------------------------------------------------------------------
% impose BCs
[SYSMAT,rhs] = imposeBC(Nodes,NDir,DirNod,DirVal,SYSMAT);
%---------------------------------------------------------------------------
%% SOLVE THE LINEAR SYSTEM
%---------------------------------------------------------------------------
% GMRES
restart=10; tol=1e-9; maxit=20;
setup.type='nofill';
setup.milu='off';
[L,U]=ilu(SYSMAT,setup);
uh=gmres(SYSMAT,rhs,restart,tol,maxit,L,U);
%--------------------------------------------------------------------------
%% PLOT RESULTS
figure(nfig)
T=triangulation(triang,coord);
triplot(triang,coord(:,1),coord(:,2),'k');
triplot(triang,coord(:,1),coord(:,2));
triplot(triang,coord(:,1),uh,'c');
trisurf(triang,coord(:,1),coord(:,2),uh,'FaceColor','none');
```

The functions called pby the script are

- input_data: Loads the mesh from a ".dat" file in the speciefied directory, or builds a uniform triangulation with $N \times N$ nodes, N specified by the user. Then compute some geometric useful informations and eventually prints the mesh.
- localBasis: Compute the elemental coefficients of the basis functions.
- BuildLSys: Builds the $H$ and $B$ matrices for the linear system.
- imposeBC: Impose the boundary conditions to the linear system.

## 2.1 Input Data

A given mesh has been provided for this project in an appropriate ".dat" file. Anyway it seemed interesting to have the possibility to refine the mesh and check convergence of the scheme at the end of the algorithm implementation. For this reason some lines of code have been added to the function to build a uniform triangulation of the domain. Practically the program creates a uniform discretization of the space and then construct the delaunay triangulation with a Matlab function (lines 32-33 below) [2]. In such case it's also necessary to provide some methods to find the boundary nodes and assign the correct Dirichlet boundary values (lines 37-54). The last part of the function computes the max edge length of each triangle, which is going to be useful for the SUD stablization.

```
 1  function [Nelem,Nodes,triang,coord,NDir,DirNod,DirVal,h]=...
 2      input_data(flag_mesh,meshdir,N, seeMesh)
 3  %%% Collects the mesh information and define the values of the parameters.
 4
 5  %--------------------------------------------------------------------------
 6  %% MESH CONSTRUCTION
 7  %--------------------------------------------------------------------------
 8  if flag_mesh == 0
 9      %----------------------------------------------------------------------
10      if nargin < 2
11          error('not enough inpt variables in "input_data", specify a' + ...
12                'directory for the mesh');
13      end
14      %------------------------------------------
15      meshtriang = strcat(meshdir, '/triang.dat');
16      meshcoord  = strcat(meshdir, '/xy.dat');
17      meshDirNod = strcat(meshdir, '/dirnod.dat');
18      meshDirVal = strcat(meshdir, '/dirval.dat');
19      %-----------------------------------------
20      triang = [];
21      load(meshtriang,'triang');
22      load(meshcoord,'xy');
23      load(meshDirNod,'dirnod');
24      load(meshDirVal,'dirval');
```

```
    %-------------------------------------------
    triang = triang(:,1:3);
    coord  = xy;
    DirNod = dirnod;
    DirVal = dirval;
%---------------------------------------------------------------------------
else
    %-----------------------------------------------------------------------
    [x,y]  = meshgrid(linspace(0,1,N),linspace(0,1,N)); % uniform mesh NxN
    triang = delaunay(x,y);
    coord  = [reshape(x,[],1),reshape(y,[],1)];

    %Gamma_1
    j       = find(coord(:,1)≤0.3);
    k       = coord(j,:);
    k       = k(:,2)==0;
    gamma_1 =[find(coord(:,1)==0);j(k)];

    %Gamma_2
    j       = find(coord(:,1)>0.3);
    k       = coord(coord(:,1)>0.3,:);
    k       = k(:,2)==0;
    p       = coord(coord(:,2)==1,:);
    l       = find(coord(:,2)==1);
    p       = p(:,1)≠0;
    gamma_2 = [find(coord(:,1)==1);j(k);l(p)];
    %------------------------------------------
    DirNod  = [gamma_1;gamma_2];
    DirVal  = [ones(1,length(gamma_1)),zeros(1,length(gamma_2))]';
%---------------------------------------------------------------------------
end
%---------------------------------------------------------------------------
%% PARAMETERS DEFINITION
%---------------------------------------------------------------------------
Nelem = size(triang,1);
Nodes = size(coord,1);
NDir  = size(DirNod,1);

%---------------------
%max length of each triangle
h = zeros(Nelem,1);
P = zeros(3,2); d=zeros(3,1);
%----------------------------
for iel=1:Nelem
    %-----------
    for iloc=1:3
        xglob=triang(iel,iloc);
        P(iloc,:)=coord(xglob);
    end
    %-----------
    for i=1:2
        d(i)=norm(P(i,:)-P(i+1,:));
    end
    %----------
    d(3)=norm(P(1,:)-P(3,:));
    h(iel)=max(d);
%----------------------------
end
%----------------------------
if nargin == 4
    if seeMesh
        TR=triangulation(triang,coord);
        triplot(TR);
    end
end
%---------------------------------------------------------------------------
end
```

## 2.2 Build Linear System

Assume $D_{xx} = D_{yy}$. Each entry $h_{ij}$ or $b_{ij}$ can be computed by subdividing the integral over the whole domain $\Omega$ into the sum of the integrals over the triangulation triangles, and exploiting the chosen formulation of the basis functions inside each triangle (7), we have

$$\begin{aligned} h_{ij} &= D \sum_{T \in \mathcal{T}_h} \int_T \left[ \frac{\partial \phi_j^T}{\partial x} \frac{\partial \phi_i^T}{\partial x} + \frac{\partial \phi_j^T}{\partial y} \frac{\partial \phi_i^T}{\partial y} \right] dT = D \sum_{T \in \mathcal{T}_h} \int_T \left[ b_j^T b_i^T + c_j^T c_i^T \right] dT \\ b_{ij} &= \sum_{T \in \mathcal{T}_h} \int_T \left( v_x b_j^T + v_y c_j^T \right) \phi_i^T \, dT. \end{aligned} \tag{12}$$

Hence, the integrand of $h_{ij}$ is constant and the one of $b_{ij}$ is linear in each triangle, so Trapezi's rule is enough to compute the exact value:

$$\begin{aligned} h_{ij} &= D \sum_{T \in \mathcal{T}_h} \left[ b_j^T b_i^T + c_j^T c_i^T \right] Area(T) \\ b_{ij} &= \sum_{T \in \mathcal{T}_h} \left( v_x b_j^T + v_y c_j^T \right) \frac{Area(T)}{3}. \end{aligned} \tag{13}$$

In Matlab this is implemented with the standard loop over the mesh elements.

```matlab
function [SYSMAT]=BuildLSys(Nelem,Nodes,triang,Bloc,Cloc,Area,Diff,Vel,h)
% build system matrix (H+B+S)
%---------------------------------------------------------------------------
%%% INITIALIZE THE SPARSE VECTORS
%---------------------------------------------------------------------------
i_sparse = zeros(Nodes*9,1);
j_sparse = zeros(Nodes*9,1);
val_B    = zeros(Nodes*9,1);
val_H    = zeros(Nodes*9,1);it=1;
%---------------------------------------------------------------------------
for iel=1:Nelem % enter the triangle
    %---------------------------------------
    for iloc=1:3 %select node i of triangle
        %---------------------------------------------
        iglob=triang(iel,iloc); %select node j of triangle
        for jloc=1:3
            %---------------------------------------------
            jglob        = triang(iel,jloc);
            i_sparse(it) = iglob;
            j_sparse(it) = jglob;
            % stiffness matrix
            val_H(it) = (Bloc(iel,iloc)*Bloc(iel,jloc)+...
                         Cloc(iel,iloc)*Cloc(iel,jloc))*Area(iel)*Diff(iel);
            % transport matrix
            val_B(it) = (Vel(1)*Bloc(iel,jloc)+Vel(2)*Cloc(iel,jloc))*Area(iel)/3;

             it=it+1;
             %---------------------------------------------
        end % end jloc
    end % end iloc
end % end iel
%--------------------
% BUILD SPARSE MATRICES
%--------------------
H = sparse(i_sparse,j_sparse,val_H);
B = sparse(i_sparse,j_sparse,val_B);
SYSMAT = H + B;
%---------------------------------------------------------------------------
end
```

## 2.3 Impose BC

Below the implementation for the imposition of the Dirichlet BCs in the linear system. Two possibilities have been exploited, the penalty method and the lifting function approach.
We remark that the lifting operator provides a more precise and stable scheme, but is in general more complicated as requires higher CPU times, especially for large systems. On the other hand the penalty method is extremely fast, since it only requires the modification of one value in the system matrix and in the right hand side for each boundary node.
With some tests on the CPU time we find

| | $PENALTY\ (sec)$ | $LIFTING\ (sec)$ |
|---|---|---|
| $N = 21$ | $2.4e^{-2}$ | $1.9e^{-3}$ |
| $N = 100$ | $2.5e^{-1}$ | $2.1e^{-3}$ |
| $N = 1000$ | $3.1e^{-2}$ | 460 |

```
%% IMPOSE BOUNDARY CONDITIONS
function [sysMat,rhs] = imposeBC(Nodes,NDir,DirNod,DirVal,sysMat,flag)
    %------------------------------------------------------------
    % impose Dirichle BCs
    % flag: 0 => penalty method;
    %      : 1 => Lifting function;
    if nargin < 6
        flag = 0;
    end
    rhs = zeros(Nodes,1);
    %-----------------------
    %%% DIRICHLET
    %------------------------------------------------------------
    switch flag
        %--------------------------------------------------------
        case 0
            %----------------------------------------------------
            penalty = 1e10;
            for idir = 1:NDir
                %-----------------------------------------------
                iglob                = DirNod(idir);
                sysMat(iglob,iglob)  = penalty;
                rhs(iglob)           = penalty*DirVal(idir);
                %-----------------------------------------------
            end
            %----------------------------------------------------
        case 1
            %----------------------------------------------------
            for idir=1:NDir
                %-----------------------------------------------
                iglob            = DirNod(idir);
                sysMat(iglob,:)  = 0;
                rhs              = rhs-DirVal(idir)*sysMat(:,iglob);
                rhs(iglob)       = DirVal(idir);
                sysMat(:,iglob)  = 0;
                sysMat(iglob,iglob)  = 1;
                %-----------------------------------------------
            end
            %----------------------------------------------------
    end
end
%------------------------------------------------------------------
```

## 2.4 Case $v_x = v_y = 0$ and BCs problem

When $v_x = v_y = 0$ the code solves a diffusion problem, and using PCG for the resolution of the linear system (which has in this case a symmetric matrix, given that $B = 0$) it's easily found that the results are perfectly equivalent to the ones obtained with a code for a pure diffusion problem.
As written in the assignment paper, using the GMRES algorithm for solving the linear system we must be careful when imposing the Dirichlet BCs with a penalty approach, due to resulting high value of the right hand side, which gives problems at the termination method of GMRES (based on the norm $\frac{||SYSMAT*u_h - rhs||}{||rhs||}$, where $||rhs|| \sim penalty$, and when the penalty is high the relative exit norm is extremely low, forcing the algorithm to stop). Another possibility is to take benefit of the MATLAB library and use the function "mldivide" or "\", which automatically recognize the best solver algorithm for the specified matrix and use different stopping criteria for the check on the residual. This function is usually slower than a direct application of the best method with its appropriate preconditioners, but when we don't exactly know what is the best way to treat our linear system is an extremely powerful tool.
In our case, even with a a $\lambda \sim 1e40$ the "mldivide" function gives a final residual of order smaller than $1e^{-10}$, even if some warnings about the high conditioning number of the system matrix are in the code execution. The idea is that since the lifiting approach for the BCs constitutes a problem only for large systems, and when it can be applied the gmres is an optimal method for the solution of our problem, it makes no sense to employ the mldivide MATLAB function. On the opposite, when the system becomes large and the penalty method has to be applied, a different algorithm with a different stopping criteria should be imposed, and a fast and simple way to do that is to call this "mdivide" function.

# 3 Check on the solution: Mass balance

A good way to check the validity of our solution is to verify that the total flux across the boundary is zero in our case of no external forces. To do that we have to compute both the convective and diffusive flux. The convective flux is by definition

$$q_C = \int_{\partial\Omega} u\beta \cdot n d\sigma, \tag{14}$$

and since we have only Dirichlet BCs in the whole contour of the domain, we know exactly the values of u. Hence, substituting

$$q_C = \int_{\Gamma_1} \beta \cdot n d\sigma = \int_0^1 \beta \cdot \begin{bmatrix} -1 \\ 0 \end{bmatrix} dy + \int_0^{0.3} \beta \cdot \begin{bmatrix} 0 \\ -1 \end{bmatrix} dx = -v_x - 0.3 * v_y, \tag{15}$$

which is clearly zero in the Poisson case ($v_x = v_y = 0$). The diffusive flux is instead given by

$$q_D = \int_{\partial\Omega} -D\nabla u \cdot n d\sigma. \tag{16}$$

Now, following chapter 2.9 of [3], since the basis functions $\phi_i$ in such a way that

$$\sum_{i=1}^{N} \phi_i(x, y) = 1 \quad \forall (x, y) \in \Omega, \tag{17}$$

using the above definition for the entries of the diffusive matrix H, we have

$$h_{ij} = h(\phi_i, \phi_j) \quad \sum_{j=1}^{N} h_{ij} = h(\phi_i, \sum_{j=1}^{N} \phi_j) = 0 \implies h_{ii} = -\sum_{j=1, j\neq i}^{N} h_{ij}.$$

This moreover proves that when no BCs are imposed to a strict Poisson problem the kernel of the system matrix has dimension one and is generated by the constant vector.
Writing the $i_{th}$ element of the product $H \cdot u$ we have

$$\sum_{j=1}^{N} h_{ij}u_j = b_i \quad h_{ii}u_i + \sum_{j=1,j\neq i}^{N} h_{ij}u_j = b_i, \tag{18}$$

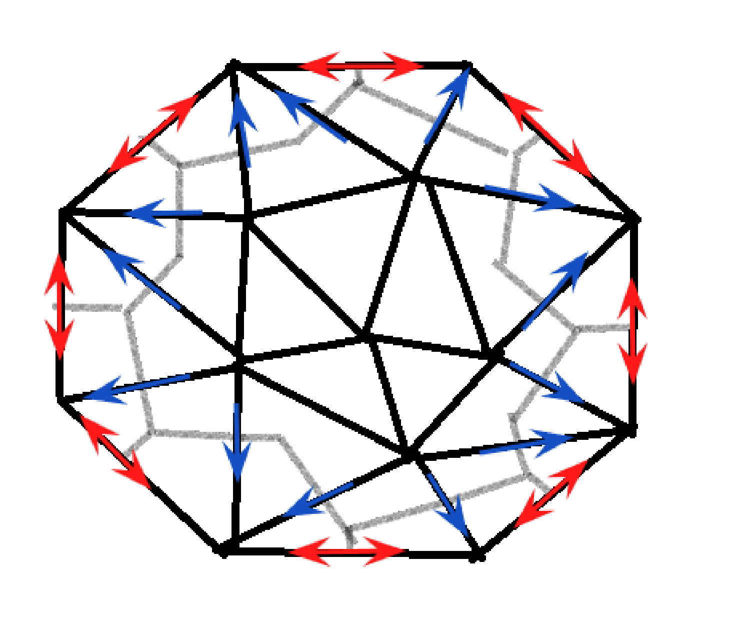
where $b_i$ may not be zero at the Dirichlet boundary nodes. Using (3) we obtain

$$\sum_{j=1,j\neq i}^{N} h_{ij}(u_j - u_i) = b_i, \quad \text{or} \quad \sum_{j=1,j\neq i}^{N} h_{ij}|\sigma_{ij}|\frac{(u_j-u_i)}{|\sigma_{ij}|} = b_i, \tag{19}$$

with $\sigma_{ij}$ length of edge of the triangulation with endpoints given by nodes $i$ and $j$. The term $\frac{u_j-u_i}{\sigma_{ij}}$ is a first order finite difference approximation of the gradient of $u$ along the edge $\sigma_{ij}$. It's easy to see how similar this result is to the diffusive flux going from node i to node j through the edge $\sigma_{ij}$

$$q_{ij} = -D|\sigma_{\alpha\beta}|\frac{u_j-u_i}{|\sigma_{ij}|},$$

where $\sigma_{\alpha\beta}$ is the edge of the Voronoi cell of node $i$ orthogonal to edge $\sigma_{ij}$. Hence, if $h_{ij} = -D\frac{|\sigma_{\alpha\beta}|}{|\sigma_{ij}|}$, and this can be proved to be exactly our case of Galerkin P1 formulation in two dimensional Delaunay triangulation and constant coefficients, the $i_{th}$ component of the product $H \cdot u$ is a first order approximation of the total diffusive flux flowing in the Voronoi cell of node $i$.
Summing all the values of $H \cdot u$ for the Dirichlet nodes, looking at figure (1) we are practically summing all



*Voronoi cells around the Dirichlet nodes and relative fluxes. The red ones are fluxes between two boundary nodes, which are opposite in two adjacent cells and thus cancel out in the sum. The blue ones are the ones computed with respect to the internal nodes, and their sum is what we use as the approximate flux outward*

the represented fluxes to each other. Actually the blue arrows are the opposites of the terms $\sum_{j=1}^{N} h_{ij}u_j$, since those represent the flux from node $i$ at the boundary to all the nodes j, internal, and with the sum $\sum_{i\in Dirichlet}\sum_{j=1}^{N} h_{ij}u_j$ we obtain the ingoing flux through the domain boundary(approximated by the grey contour of the Dirichlet Voronoi cells).
Therefore we want the sum

$$q_C + q_D \sim -v_x - 0.3 * v_y - \sum_{i\in Dirichlet} H_i \cdot u \sim 0, \tag{20}$$

where $H_i$ stands for row $i$ of H. Using the prescribed mesh in the zero velocity case we have an uncertainty of $10^{-12}$ to the mass balance.
Since this would be a first order approximation of the diffusive flux, is not guaranteed that with a coarse mesh the given result equalize the value of the convective flux. In general a more precise computation of the diffusive flux can arise from the variational formulation,

$$\int_\Omega (D\nabla u_h)\cdot\nabla\phi_i d\Omega - \int_{\partial\Omega}\phi_i(D\nabla u_h)\cdot n d\sigma + \int_\Omega div(\beta u)\phi_i d\Omega = 0, \quad i=1,...,N, \tag{21}$$

where before we sent to zero the integral over the boundary $\partial\Omega$, since we said that when only Dirichlet BCs are prescribed at the boundary we choose a test space with functions $v$ that goes to zero there, and this is practically obtained by basically substituting the equation of the linear system corresponding to a Dirichlet nodes with the direct imposition on the strong form of the Dirichlet boundary value (like $u_{idir} = DirVal_{idir}$). From the numerical point of view, anyway, we compute the values of coefficients $b_i^T$, $c_i^T$ for the Dirichlet nodes with the exact same rule used for the internal ones, since we have to use it in the lifting operator for the connection with the internal nodes. Considering $\phi_i$, for $i$ Dirichlet node, function of the basis of $\mathcal{V}_h$ ($\mathcal{V}_h$ subspace of $H^1$ generated by FEM basis functions that are nonzero on the boundary), space where the boundary operator is approximated, and looking at the variational formulation we have exactly (21), with the term over $\partial\Omega$ that does not vanish.
We note here how such term contains something extremely similar to what we are looking for, $\int_{\partial\Omega}(D\nabla u)\cdot n d\sigma$ just multiplied by a test function. Now, for the same reason of (17), we have that

$$\sum_{idir=1}^{NDir}\phi_{idir}(x,y) = 1, \quad \forall(x,y)\in\partial\Omega, \tag{22}$$

and summing (21) over all the $i$ of Dirichlet we have,

$$\begin{aligned}
&\sum_{idir=1}^{NDir}\left[\int_\Omega (D\nabla u_h)\cdot\nabla\phi_{idir}d\Omega - \int_{\partial\Omega}\phi_{idir}(D\nabla u_h)\cdot n d\sigma + \int_\Omega div(\beta u)\phi_{idir}d\Omega\right] = 0\\
\iff &\sum_{idir=1}^{NDir}\left[\int_\Omega (D\nabla u_h)\cdot\nabla\phi_{idir}d\Omega + \int_\Omega div(\beta u)\phi_{idir}d\Omega\right] = \int_{\partial\Omega}\sum_{idir=1}^{NDir}\phi_{idir}(D\nabla u_h)\cdot n d\sigma\\
\iff &\int_{\partial\Omega}(D\nabla u_h)\cdot n d\sigma = \sum_{idir=1}^{NDir}\left[\int_\Omega (D\nabla u_h)\cdot\nabla\phi_{idir}d\Omega + \int_\Omega div(\beta u)\phi_{idir}d\Omega\right]\\
\iff &q_D = \sum_{idir=1}^{NDir}\left[(H_{idir}+B_{idir})\cdot u_h\right].
\end{aligned} \tag{23}$$

We will see in the results below that this formulation gives a more precise result even for coarse meshes.

## 4 Oscillations nature and SUG stabilization

From Lax-Milgram theorem we know that our problem is well posed if its associated bilinear form $a : \mathcal{V}\times\mathcal{V}\to\mathbb{R}$ is continuous and coercive and if the linear form associated to the forcing term $F : \mathcal{V}\to\mathbb{R}$ is bounded and continuous.
In our case we don't have external force fields, so the last property is immediately satisfied, and the differential operator is defined as

$$a(u,v) = \int_\Omega D\nabla u\cdot\nabla v d\Omega + \int_\Omega div(\beta u)v d\Omega.$$

For the other two we have that for all $v\in\mathcal{V}$ (following chapterof [3])

$$\int_\Omega div(\beta u)v d\Omega = \int_{\partial\Omega}(\beta\cdot n)uv d\sigma - \int_\Omega(\beta u)\cdot\nabla v d\Omega = \int_\Omega(\beta\cdot\nabla u)v d\Omega + \int_\Omega(div\beta)uv d\Omega.$$

In this, the term over the boundary vanishes since we only have Dirichlet BCs and hence all test functions are identically zero on $\partial\Omega$. Taking $u = v$ we have

$$\int_\Omega (\beta \cdot \nabla v) v d\Omega = -\frac{1}{2} \int_\Omega div(\beta) v^2 d\Omega,$$

from which

$$\int_\Omega div(\beta v) v d\Omega = \int_\Omega div(\beta) v^2 d\Omega + \int_\Omega (\beta \cdot \nabla v) v d\Omega = \frac{1}{2} \int_\Omega div(\beta) v^2 d\Omega. \tag{24}$$

To prove coercivity we need to find a constant $\alpha > 0$ such that $a(v, v) \geq \alpha ||v||^2_{H^1(\Omega)}$ for all $v \in \mathcal{V}$, and from previous result

$$a(v, v) = \int_\Omega \left[ D|\nabla v|^2 + \frac{1}{2} div(\beta) v^2 \right] d\Omega. \tag{25}$$

In our case the advective field that transports $u$ is conservative, $div\beta = 0$, so the coercivity is completely controlled by the diffusive term, from which

$$a(v, v) = \int_\Omega D|\nabla v|^2 d\Omega \geq infD \int_\Omega |\nabla v|^2 d\Omega \geq \frac{infD}{1 + C_\Omega^2} ||v||_{H^1(\Omega)^2},$$

where $C_\Omega$ is the Poincaré constant. This proves coercivity.
For the continuity (need to find a $\gamma > 0$ for which $a(u, v) \leq \gamma ||u||_{H^1(\Omega)} ||v||_{H^1(\Omega)}$)we simply have

$$|a(u, v)| \leq ||D||_\infty ||\nabla u||_2 ||\nabla v||_2 + ||\beta||_\infty ||\nabla u||_2 ||v||_2 \leq (||D||_\infty + C_\Omega ||\beta||_\infty) ||u||_{H^1(\Omega)} ||v||_{H^1(\Omega)},$$

where we used again Poincaré lemma and we used the fact that if $\Gamma_D = \partial\Omega$ the $L^2$ and $H^1$ norm are equivalent. Therefore in our case

$$\alpha = \frac{infD}{1 + C_\Omega^2}$$
$$\gamma = ||D||_\infty + C_\Omega ||\beta||_\infty.$$

Now, from the Ceà Lemma:

$$||u - u_h||_{H^1} \leq \frac{\gamma}{\alpha} ||u - v||_{H^1} = (1 + C_\Omega^2) \frac{||D||_\infty + C_\Omega ||\beta||_\infty}{infD} ||u - v||_{H^1}, \quad \forall v \in \mathcal{V}_h. \tag{26}$$

From this result is clear that when $||\beta||_\infty >> ||D||_\infty$, the Ceà constant becomes indefinitely large and thus so could be the computational errors. This mathematical issue is observed by numerical oscillations in the solution, that needs to be stabilized when the equation is "convection-dominated". The idea is to add some "numerical" diffusion so that the Ceà constant becomes lower. A good and simple approach is the correction given by streamline upwind diffusion (SUD).

## 4.1 SUD

The streamline upwind diffusion introduce numerical diffusion in the streamline direction, basically

$$\int_\Omega (\beta \cdot \nabla u)(\beta \cdot v) d\Omega,$$

scaled by a factor for each element dependent on the mesh Pèclet number, which is defined by local ratio between advective and diffusive terms,

$$\mathbb{P}e_h = \frac{|\beta_k| h_k}{D_k},$$

with $D_k$ and $\beta_k$ mean diffusion coefficient and mean velocity vector in element $T_k$, and by a global factor $\tau$, which needs to be calibrated to tune the amount of numerical diffusion. The idea is that to maintain consistency in the problem we should add the smallest amount possible of extra stabilizing terms. Our weak formulation becomes than

$$\int_\Omega \left[(D\nabla u)\cdot\nabla v + div(\beta u)v + \tau\frac{\mathbb{P}e_h}{|\beta_k|^2}(\beta\cdot\nabla u_h)(\beta\cdot\nabla v)\right] d\Omega = 0, \tag{27}$$

which leads to the Galerkin formulation

$$(H+B+S)u = 0, \tag{28}$$

where $H$ and $B$ are the same matrices defined above, and $S$ is the additive stabilizing term with components

$$S = \{s_{ij}\}, \quad s_{ij} = \sum_{T\in\mathcal{T}_h}\int_T \tau\frac{h_k}{|\beta_k|D_k}(\beta\cdot\nabla\phi_j)(\beta\cdot\nabla\phi_i)dT.$$

The "BuildLSys.m" function is now modified as

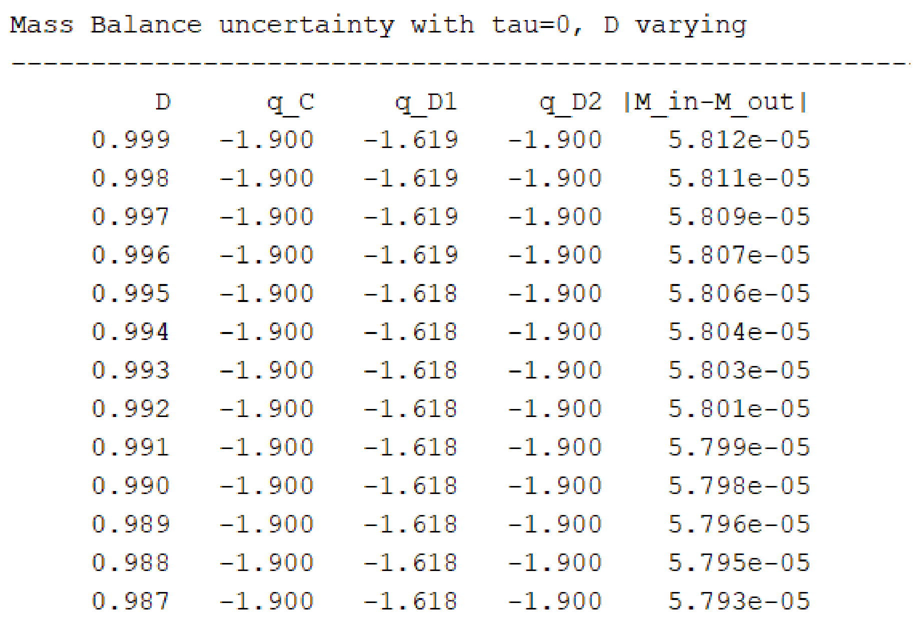
```
function [SYSMAT, H, B, S]=BuildLSys(Nelem,Nodes,triang,Bloc,Cloc,Area,Diff,Vel,h,tau)
% build system matrix (H+B+S)
%---------------------------------------------------------------------------
%%% INITIALIZE THE SPARSE VECTORS
%---------------------------------------------------------------------------
i_sparse = zeros(Nodes*9,1);
j_sparse = zeros(Nodes*9,1);
val_B    = zeros(Nodes*9,1);
val_H    = zeros(Nodes*9,1);
val_S    = zeros(Nodes*9,1);
it=1;
%---------------------------------------------------------------------------
for iel=1:Nelem % enter the triangle
    %---------------------------------------
    for iloc=1:3 %select node i of triangle
        %---------------------------------------------
        iglob=triang(iel,iloc); %select node j of triangle
        for jloc=1:3
            %---------------------------------------------
            jglob        = triang(iel,jloc);
            i_sparse(it) = iglob;
            j_sparse(it) = jglob;
            % stiffness matrix
            val_H(it) = (Bloc(iel,iloc)*Bloc(iel,jloc)+...
                         Cloc(iel,iloc)*Cloc(iel,jloc))*Area(iel)*Diff(iel);
            % transport matrix
            val_B(it) = (Vel(1)*Bloc(iel,jloc)+Vel(2)*Cloc(iel,jloc))*Area(iel)/3;

            % SUPG stablization
             if norm(Vel)==0
                 val_S(it) = 0;
             else
                 val_S(it) = ...
                     tau/Diff(iel)*h(iel)/sqrt(Vel(1)^2+Vel(2)^2)*(Vel(1)*Bloc(iel,iloc)+Vel(2)*...
                     Cloc(iel,iloc))*(Vel(1)*Bloc(iel,jloc)+Vel(2)*Cloc(iel,jloc))*Area(iel);
             end
             it=it+1;
             %---------------------------------------------
        end % end jloc
    end % end iloc
end % end iel
%---------------------
% BUILD SPARSE MATRICES
%---------------------
H = sparse(i_sparse,j_sparse,val_H);
B = sparse(i_sparse,j_sparse,val_B);
S = sparse(i_sparse,j_sparse,val_S);
SYSMAT = H + B + S;
%---------------------------------------------------------------------------
end
```

# 5 Results

We analyze here some results for an advective velocity of $\beta = (1,3)^T$. In such case the advective flux is equal to 1.9 pointing inside the domain.

## 5.1 Some tests: $\tau = 0,\ D = 10^{-3} \div 1$

The goal is to find the minimal value of D for which the oscillations occur.
We note here that the minimal value of $D$ at which oscillations occur depends strictly on the mesh size of the elements, from the local value of the Peclet number we have in fact $\mathbb{P}e_h = \frac{|\beta_k|h_k}{D_k} = \frac{|\beta|h}{D}$, where in last step we used the fact that the diffusion factor and convective velocity are constants along all the domain, and that the mesh is uniform (the provided mesh is actually a uniform mesh with $21 \times 21$ nodes). To compute it an idea is to solve the problem and compute its mass balance uncertainty in a while loop, starting from a diffusion constant $D = 1$ and decreasing its value by $10^{-3}$ at each iteration, until the error is greater than some chosen tolerance, say $10^{-6}$. If that is the case we stop the loop and check the result. This is surely not the best approach overall, but requires a simple implementation, and even if may need to run the solver several times the provided mesh has just $21 \times 21$ nodes, and the running times are quite low. Note furthermore that is not guaranteed that oscillations will appear as errors in the mass balance. Oscillations can be found in the solution values for $u_h$, the effects could cancel each other out in the $H \cdot u_h$ or $(H + B) \cdot u_h$ products for the fluxes. Heuristically we imagine that if the oscillations are large, so must be the errors in $u_h$ and at some point even the results for the fluxes must be influenced and show some errors.
We note that at least for the first first results, at high diffusion constant, the numerical mass residual is

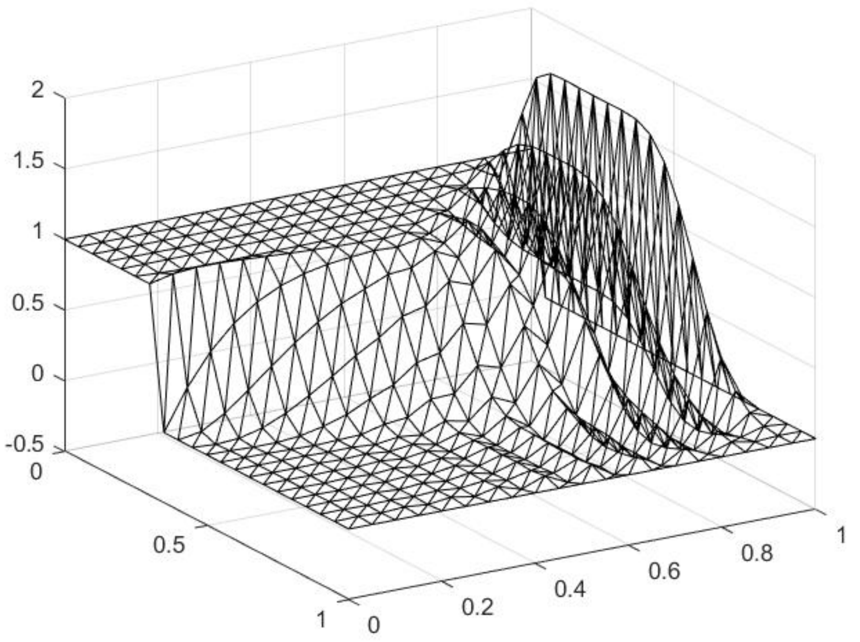


```
Mass Balance uncertainty with tau=0, D varying
-------------------------------------------------------
          D       q_C      q_D1      q_D2 |M_in-M_out|
      0.999    -1.900    -1.619    -1.900     5.812e-05
      0.998    -1.900    -1.619    -1.900     5.811e-05
      0.997    -1.900    -1.619    -1.900     5.809e-05
      0.996    -1.900    -1.619    -1.900     5.807e-05
      0.995    -1.900    -1.618    -1.900     5.806e-05
      0.994    -1.900    -1.618    -1.900     5.804e-05
      0.993    -1.900    -1.618    -1.900     5.803e-05
      0.992    -1.900    -1.618    -1.900     5.801e-05
      0.991    -1.900    -1.618    -1.900     5.799e-05
      0.990    -1.900    -1.618    -1.900     5.798e-05
      0.989    -1.900    -1.618    -1.900     5.796e-05
      0.988    -1.900    -1.618    -1.900     5.795e-05
      0.987    -1.900    -1.618    -1.900     5.793e-05
```

*First part of the table of results for the Mass balance with D varying and no stabilizing terms. $q_{D1}$ is the diffusive flux obtained with the first order approximation, whereas $q_{D2}$ is the one obtained with approach deriving from the weak formulation. The error $|M_{in} - M_{out}|$ is computed as $q_C - q_{D2}$, where $q_C$ is the convective flux, positive if outgoing, and $q_D$ is the diffusive flux, positive if ingoing.*

actually decreasing a little, even if the Peclet number increases. If we instead use the first order diffusive flux for the mass balance check we instead find it increasing. A possible explanation is that with such formulation the diffusive flux is strongly related to the discrete values of the gradient of $u$ along the edges connecting the internal nodes with the boundary ones, and hence if the solution oscillates even a little

the result feels the bias.
From the test described above we actually find that with the error in the mass conservation computed as $q_C - q_D$, with $q_D$ approximated as the sum of $(H + B)_i \cdot u_h$ over the Dirichlet nodes, gradually decreases up to a minimal value of order $1e^{-10}$ at $D = 0.093$, but then it starts to increase again, with some oscillations in the order $1e^{-8} \div 1e^{-10}$, overcoming the chosen tolerance of $1e^{-4}$ at $D = 0.012$. The result in this case is showed in figure (3)

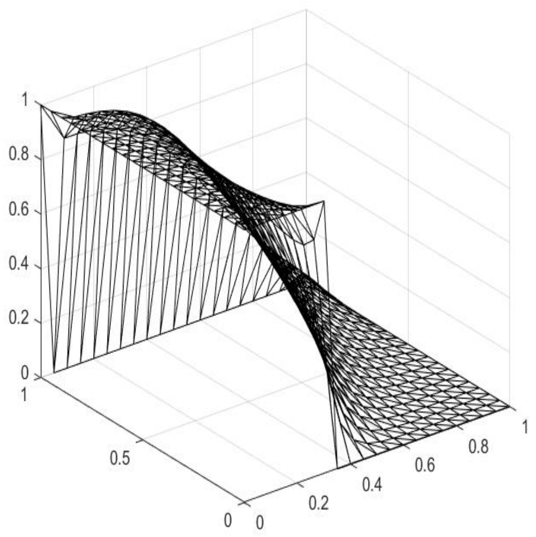


*Result with no stabilization and $D = 0.012$. The error in the mass balance is above the chosen tolerance of $1e^{-4}$, and oscillations in the solution are evident.*

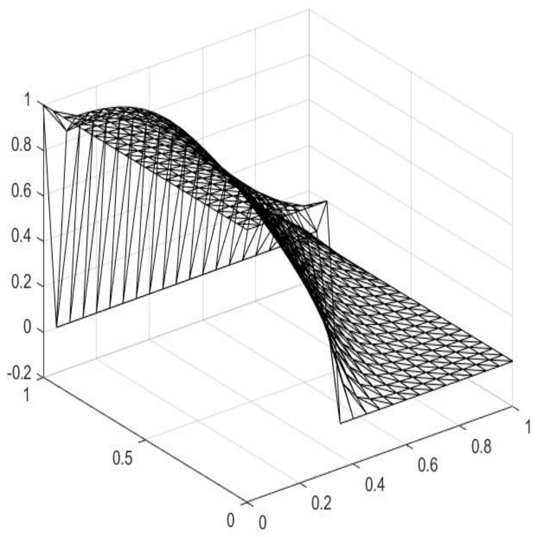
Actually we see that at this point the oscillations are already large, thus we expect that the instabilities actually start at an higher diffusion constant. It seems reasonable to firstly analyze the case where the error is minimal, $D = 0.093$. In such case the result seem actually oscillations free. Decreasing the diffusion constant instead we slowly see the appearance of small oscillations.

```
Mass Balance uncertainty with tau=0, D varying
------------------------------------------------------------
        D       q_C      q_D1      q_D2  |M_in-M_out|
    0.107    -1.900    -1.115    -1.900     1.895e-07
    0.106    -1.900    -1.110    -1.900     1.685e-07
    0.105    -1.900    -1.106    -1.900     1.487e-07
    0.104    -1.900    -1.102    -1.900     1.302e-07
    0.103    -1.900    -1.098    -1.900     1.129e-07
    0.102    -1.900    -1.094    -1.900     9.675e-08
    0.101    -1.900    -1.089    -1.900     8.179e-08
    0.100    -1.900    -1.085    -1.900     6.796e-08
    0.099    -1.900    -1.080    -1.900     5.524e-08
    0.098    -1.900    -1.076    -1.900     4.361e-08
    0.097    -1.900    -1.071    -1.900     3.302e-08
    0.096    -1.900    -1.067    -1.900     2.346e-08
    0.095    -1.900    -1.062    -1.900     1.488e-08
    0.094    -1.900    -1.057    -1.900     7.248e-09
    0.093    -1.900    -1.052    -1.900     5.359e-10
    0.092    -1.900    -1.048    -1.900     5.298e-09
    0.091    -1.900    -1.043    -1.900     1.029e-08
    0.090    -1.900    -1.038    -1.900     1.448e-08
    0.089    -1.900    -1.033    -1.900     1.792e-08
    0.088    -1.900    -1.027    -1.900     2.063e-08
```

*Table of results for the Mass balance with D varying from $D = 0.107$ to $D = 0.088$. We see a minimum in the error at the value $D = 0.093$, then reducing D the uncertainty raises.*



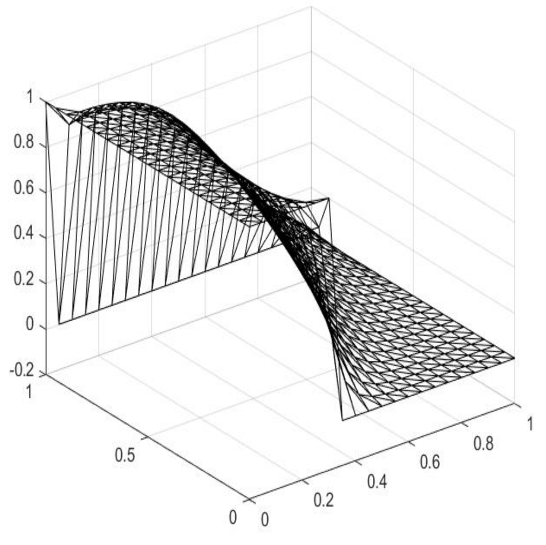
(a) *Case $D = 0.093$, error $= 5.34 \cdot 10^{-10}$*



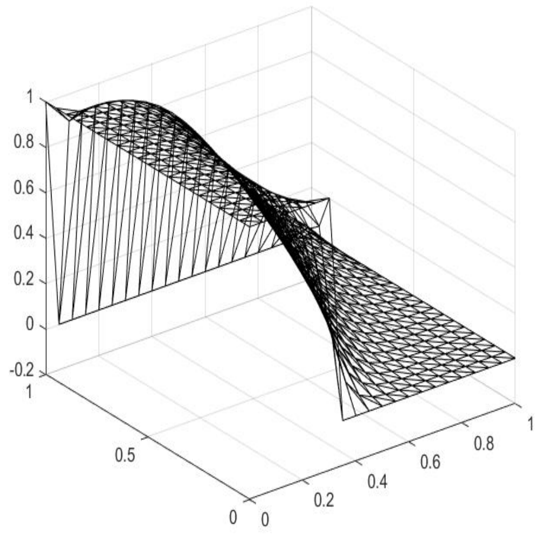
(b) *Case $D = 0.092$, error $= 5.30 \cdot 10^{-9}$*

*Solution comparison around the the minimal error case at $D = 0.093$. The scale off the two graphs is different, due to the fact that MATLAB automatically scales the axis depeding on the data. This means that for $D = 0.092$ there is at least one value of the solution $u_h < 0$ (actually two in this case).*

We note (5) that the more we reduce the value of $D$ below 0.093, the more appears in the approximated solutions values below the zero, or even above one in case of large oscillations. From a physical point of view, we are searching for a stationary solution of a convection-diffusion problem with zero external forces, with constant velocity field and diffusion constant, so the intuition is that the maximum and minimum values of the unknwon should be at the boundary of the domain, if otherwise a peak appears inside the domain we would still be in a transitory situation. From a mathematical point of view is possible to prove this intuition in a rigorous way, as done in [4] in the "Maximum principle". Hence in out case we shouldn't have any solution value above 1 or below 0, and since this happens for all the cases with $D < 0.93$, apart from the mass conservation and the actual possibility to see oscillations in the graph, we say that such value is the minimal possible one for the diffusion constant that allows to compute an acceptable solution without any stabilizing term. Below some graphs of the solution for different $D$. The pseudo Peclet number associated with $D = 0.093$ is then $\mathbb{P}e = \frac{||\beta||_\infty h}{D} = \frac{3 \cdot 0.0707}{0.093} \simeq 2.28$, with $h = 0.0707$ size of the elements of the prescribed mesh.
What is remarkable is that we now have an idea of the value of the Peclet number at which the oscillations should start, and from this we could refine the mesh, that is reduce the mesh size $h$ to the minimal value sufficient to avoid oscillations, or on the opposite side, predict what is the minimal possible value of $D$ which give stable solutions for each possible prescribed mesh. For example with a mesh of $51 \times 51$ nodes, $h = 0.0283$, and $D_{min} = 0.037$. Testing it with the solver we actually find that the oscillations starts slightly above, at $D = 0.039$, at a $\mathbb{P}e = 2.176$, but the result was still extremely close.



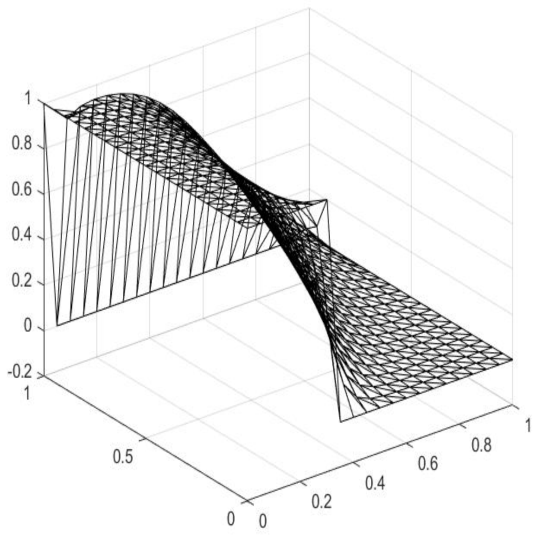
(a) *Case* $D = 0.085$, *error* $= 2.49 \cdot 10^{-8}$



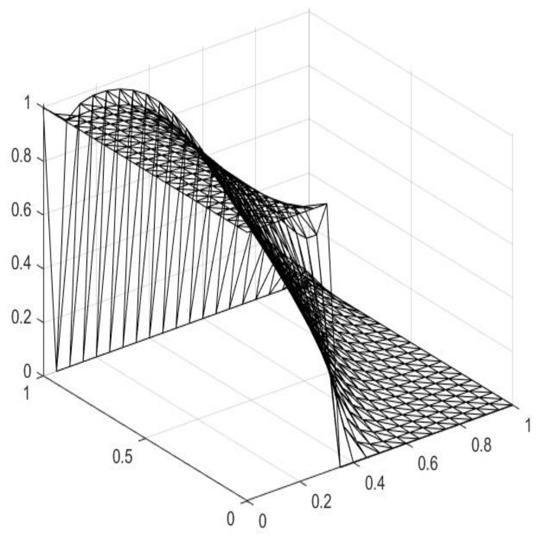
(b) *Case* $D = 0.080$, *error* $= 2.18 \cdot 10^{-8}$

## 5.2 Mass balance at low Peclet

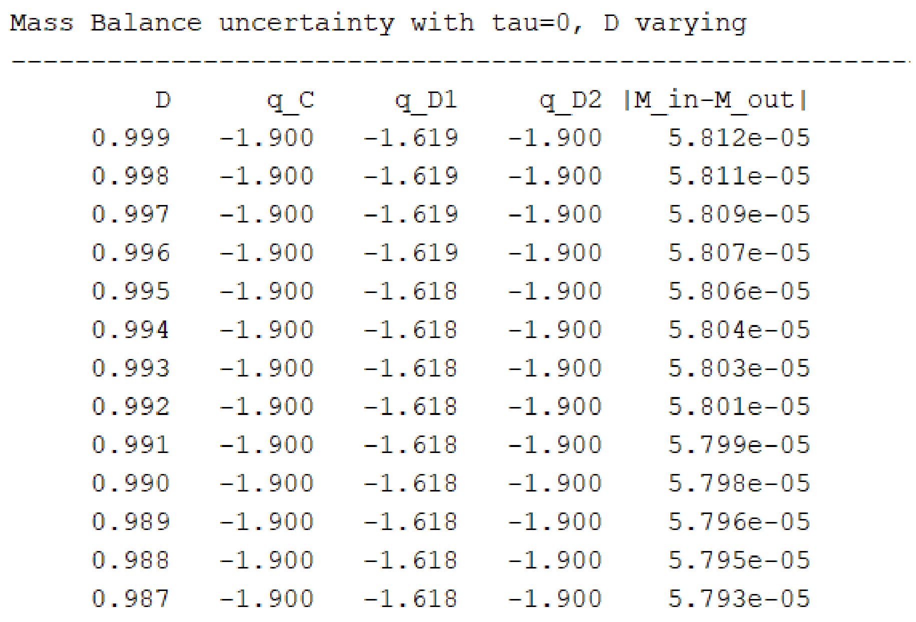
We here check that the two given approaches for the computation of the diffusive flux actually converge to the right value when the discretization tends to cover the whole domain and the solution becomes more accurate. For simplicity we first consider the case of low Peclet number, with $D = 1.0$ and $\beta = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$.



(a) *Case* $D = 0.075$, *error* $= 1.1 \cdot 10^{-8}$



(b) *Case* $D = 0.070$, *error* $= 1.9 \cdot 10^{-9}$



```
Mass Balance uncertainty with tau=0, D varying
---------------------------------------------------------
           D        q_C       q_D1       q_D2 |M_in-M_out|
       0.999     -1.900     -1.619     -1.900     5.812e-05
       0.998     -1.900     -1.619     -1.900     5.811e-05
       0.997     -1.900     -1.619     -1.900     5.809e-05
       0.996     -1.900     -1.619     -1.900     5.807e-05
       0.995     -1.900     -1.618     -1.900     5.806e-05
       0.994     -1.900     -1.618     -1.900     5.804e-05
       0.993     -1.900     -1.618     -1.900     5.803e-05
       0.992     -1.900     -1.618     -1.900     5.801e-05
       0.991     -1.900     -1.618     -1.900     5.799e-05
       0.990     -1.900     -1.618     -1.900     5.798e-05
       0.989     -1.900     -1.618     -1.900     5.796e-05
       0.988     -1.900     -1.618     -1.900     5.795e-05
       0.987     -1.900     -1.618     -1.900     5.793e-05
```

*Table of results for the Mass balance with a mesh of* $N \times N$ *points and no stabilizing terms.* $q_{D1}$ *is the diffusive flux obtained with the first order approximation, whereas* $q_{D2}$ *is the one obtained with approach deriving from the weak formulation. We see that the more we refine the mesh the more the two formulations for the computation of the diffusive flux become closer to what we expect as real solution.*

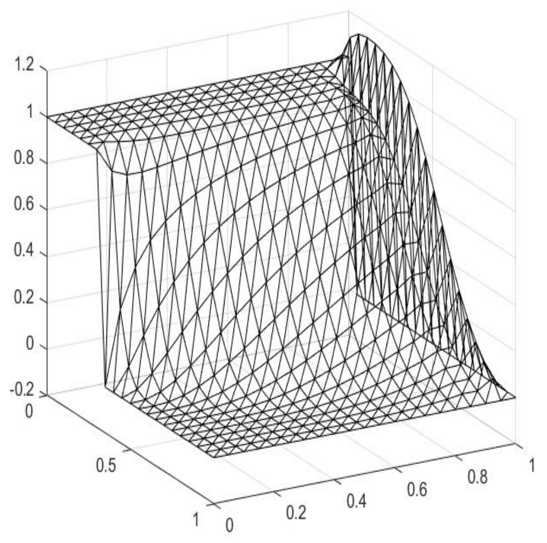
## 5.3 Effects of SUD for different values of $\tau$

As written above, the idea is to introduce the smallest possible amount of numerical diffusion to prevent the problem from oscillations in the solution. For this reason we recall our previous result from the maximal value of the Peclet number at which the solution is oscillations free, $\mathbb{P}e_h = 2.176$. We want to introduce the right amount of stabilization to have a numerical Peclet number lower than this even when the true one would be higher, that is at lower diffusion constants.
Let us work with the prescribed mesh, $h = 0.0707$. In this case we found that the minimum possible value

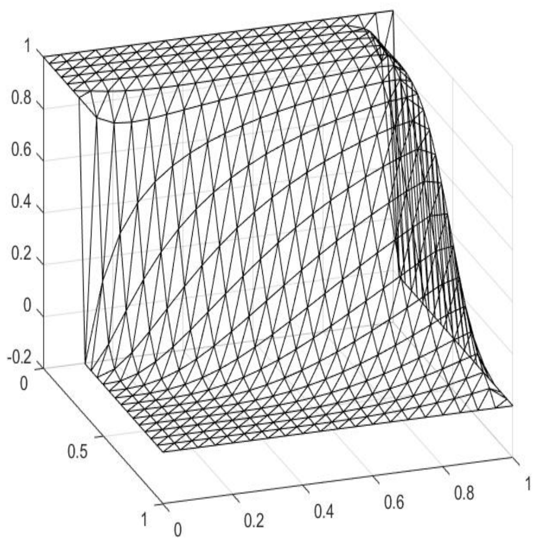
of $D$ is 0.093 to avoid instabilities. When $D$ is actually lower we calibrate the $\tau$ parameter to introduce a numerical diffusion such that $D + D_{num} = 0.093$. Just looking at the SUD formulation, it's clear that we are basically adding to the formulation a term exactly equal to the diffusion one in the streamline direction, but scaled by a factor $\tau\frac{h}{||\beta||D}$ when the coefficients are domain constants. Therefore, when $D < 0.093$ want

$$\tau\frac{h||\beta||}{D} = 0.093 - D \implies \tau = \frac{(0.093 - D)D}{h||\beta||}. \tag{29}$$

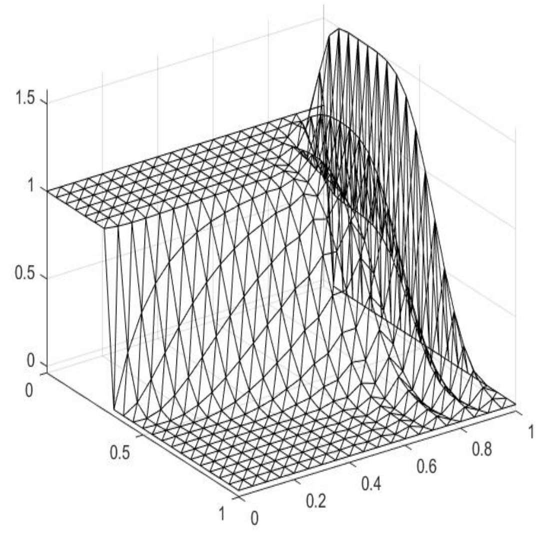
Below some results for the SUD stabilization. The effects of an excessive stabilization are shown in



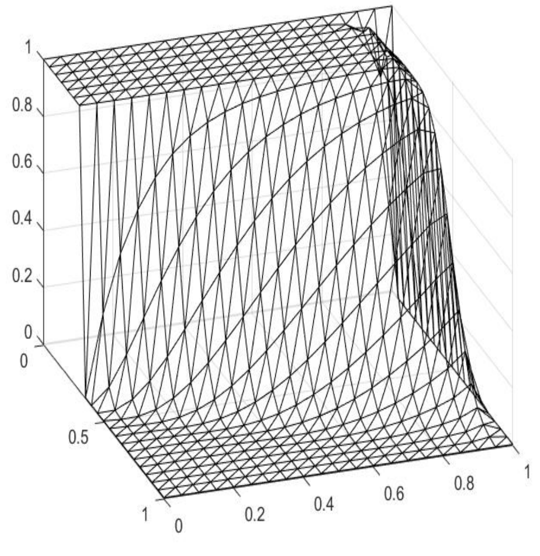
(a) *Case* $D = 0.05$, *not stabilized*



(b) *Case* $D = 0.05$, *stabilized*

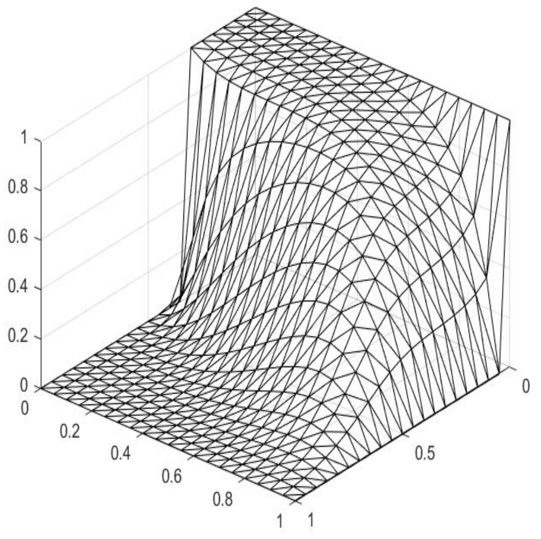


(a) *Case* $D = 0.02$, *not stabilized*

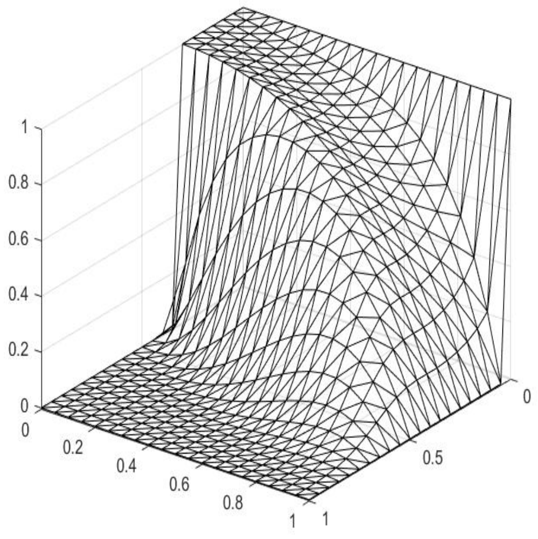


(b) *Case* $D = 0.02$, *stabilized*

the figure below. The more numerical diffusion we introduce in the system the less accurate will be the mass balance, since the mass conservation is actually done as $q_{Dtrue} + q_{Dnumerical} + q_C = 0$ and not as $q_{Dtrue} + q_C = 0$ . Actually, considering in (23) the modified version of the Galerkin formulation with the



(a) *Case* $D = 0.05$, $\tau = 0.1$



(b) *Case* $D = 0.02$, $\tau = 0.1$

extra stabilizing term we would obtain

$$q_D = \sum_{idir=1}^{NDir} \left[ (H_{idir} + B_{idir} + S_{idir}) \cdot u_h \right], \tag{30}$$

and hence in this case the total diffusive flux of the formulation is equal to the one of the correct version $q_{Dtrue} = \sum_{idir=1}^{NDir}(H_{idir} + B_{idir}) \cdot u_h$ plus the one deriving from the variational crime, $q_{Dnumerical} = \sum_{idir=1}^{NDir} S_{idir} \cdot u_h$.
In general we thus have two possible ways to avoid oscillations in the system: refine the mesh, or add numerical diffusion. The numerical diffusion is simple and doesn't practically increase the computational cost of the solution, but comes with a prize, the more of it we introduce, the less accurate is the mass balance of the solution. For this reason the best approach is to take advantage of both methods, reducing the mesh size when the required numerical diffusion is too high and would bring errors, and only then, when the new discretization allows a restrained numerical diffusion, apply the SUD to the system.

## Bibliography

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

[2] *Matlab, Delaunay triangulation.* `https://it.mathworks.com/help/matlab/ref/delaunay.html`.

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

[4] John Volker. *Convection-Diffusion-Reaction Equations and Maximum Principles*. `https://www.wias-berlin.de/people/john/LEHRE/NUMERIK_IV_21_22/num_konv_dom_prob_1.pdf`. 2021-22.