Numerical Methods for Continuous Systems
Convection-Diffusion Project

Marangon Gaia
NM: 1242310

1 Description of the problem

1.1 Differential formulation

In this report we analyze the convection-diffusion problem, expressed in the following differential formulation:

$$\begin{aligned} -div(D\nabla u) + div(\beta u) &= f && (x,y) \in \Omega \\ u(x,y) &= 1 && (x,y) \in \Gamma_1 \\ u(x,y) &= 0 && (x,y) \in \Gamma_2 \end{aligned}$$

where D is the diffusion matrix, β is the vector of transport velocity and f is the forcing function. The domain is defined as:

$$\Omega = \{(x,y) : 0 \leq x \leq 1,\, 0 \leq y \leq 1\}$$

while Dirichlet boundary conditions are set in:

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

1.2 Weak formulation

We proceed by building the weak formulation of the problem: we multiply by the test function v, we integrate over the domain,and we apply the Green's Lemma, obtaining the following formulation:

$$\text{find } u \in H_0^1(\Omega) \text{ s. t.:} \qquad a(u,v) = F(v) \qquad \forall v \in H_0^1(\Omega)$$

where the bilinear function $a(u,v)$ and the functional $F(v)$ are defined as follows:

$$\begin{aligned} a(u,v) &= \int_\Omega D\nabla u \cdot \nabla v \, d\Omega + \int_\Omega div(\beta u) v \, d\Omega \\ F(v) &= \int_\Omega f \, v \, d\Omega \end{aligned}$$

1.3 Galerkin Finite Element formulation

Finally, we pass to Galerking Finite Element formulation. By calling $\{\phi_i\}_{i=1\dots N}$ the basis functions, we can expand:

$$u(x,y) = \sum_{j=1}^{N} u_j \phi_j(x,y) \qquad\qquad v(x,y) = \sum_{i=1}^{N} v_i \phi_i(x,y)$$

We substitute the expression for $v(x,y)$ and, since the equation must hold for all values of the coefficients $\{v_i\}_{i=1\dots N}$, we can split it into N equations:

$$\int_\Omega D\nabla u \cdot \nabla \phi_i \, d\Omega + \int_\Omega div(\beta u)\phi_i \, d\Omega = \int_\Omega f\phi_i \, d\Omega \qquad\qquad \forall i = 1 \dots N$$

Analogously, in each equation, we substitute the expression for $u(x,y)$:

$$\sum_{j=1}^{N} u_j \int_\Omega D\nabla\phi_j \cdot \nabla\phi_i \, d\Omega + \sum_{j=1}^{N} u_j \int_\Omega div(\beta\phi_j)\phi_i \, d\Omega = \int_\Omega f\phi_i \, d\Omega$$

This allows to express the discretized problem in matrix form:

$$(H + B)u = \mathcal{F}$$

with:

$$H = \{h_{ij}\} \qquad h_{ij}^{(T)} = \int_T D\nabla\phi_j \cdot \nabla\phi_i \, d\Omega = \int_T \left[D_{xx}\frac{\partial\phi_j}{\partial x}\frac{\partial\phi_i}{\partial x} + D_{yy}\frac{\partial\phi_j}{\partial y}\frac{\partial\phi_i}{\partial y} \right] d\Omega$$

$$B = \{b_{ij}\} \qquad b_{ij}^{(T)} = \int_T div(\beta\phi_j)\phi_i \, d\Omega = \int_T \left[\beta_x\frac{\partial\phi_j}{\partial x} + \beta_y\frac{\partial\phi_j}{\partial y} \right] \phi_i \, d\Omega$$

$$\mathcal{F} = \{f_i\} \qquad f_i^{(T)} = \int_T f\phi_i \, d\Omega$$

where we assumed the numerical coefficients of the problem to be constant all over the domain, with the following structure:

$$D = \begin{bmatrix} D_{xx} & 0 \\ 0 & D_{yy} \end{bmatrix} \qquad\qquad \beta = \begin{bmatrix} \beta_x \\ \beta_y \end{bmatrix}$$

Moreover, we assume zero forcing function: $\mathcal{F} = \{f_i\} = 0$.

1.4 Implementation notes

We can now specify the basis functions and proceed with actual implementation. We use the following definition [1]:

$$\phi_i^{(T)} = \frac{a_i^{(T)} + b_i^{(T)}x + c_i^{(T)}y}{2|T|}$$

[1] After the definition, we'll drop the $^{(T)}$ to simplify the notation.

where $|T|$ is the area of the triangle and the coefficients a_i, b_i, c_i are found by imposing the values of the function at the vertices. By calling i, j, k the local indexing of the vertices in the triangle, with anticlockwise order, we obtain:

$$a_i = x_j y_k - x_k y_j \qquad b_i = y_j - y_k \qquad c_i = x_k - x_j$$

while the elemental area can be computed through the Van Der Monde matrix, as:

$$VDM = \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix} \qquad |T| = \frac{|det(VDM)|}{2}$$

With these defintions, and by making use of some simple relations:

$$\frac{\partial \phi_i}{\partial x} = \frac{b_i}{2|T|} \qquad \frac{\partial \phi_i}{\partial y} = \frac{c_i}{2|T|} \qquad \int_T \phi_i \, d\Omega = \frac{|T|}{3}$$

we can compute explicitly the elemental contributions for each matrix of the system:

$$h_{ij} = \left[D_{xx} b_i b_j + D_{yy} c_i c_j \right] \frac{1}{4|T|}$$
$$b_{ij} = \left[\beta_x b_j + \beta_y c_j \right] \frac{1}{6}$$

This is the notation we used in the actual implementation of the problem. For the sake of completeness, we recall another common notation, which makes use of slightly different coefficients in the definition of the basis:

$$\phi_i^{(T)} = \tilde{a}_i^{(T)} + \tilde{b}_i^{(T)} x + \tilde{c}_i^{(T)} y$$

In this case, the elemental contribution would have been:

$$h_{ij} = \left[D_{xx} \tilde{b}_i \tilde{b}_j + D_{yy} \tilde{c}_i \tilde{c}_j \right] |T|$$
$$b_{ij} = \left[\beta_x \tilde{b}_j + \beta_y \tilde{c}_j \right] \frac{|T|}{3}$$

These descriptions are completely equivalent.

1.5 SUPG stabilization

In order to solve the problem of oscillations, we introduce a residual-based stabilization term. The weak formulation now reads:

$$\text{find } u \in H_0^1(\Omega) \text{ s. t.:} \qquad a(u,v) + a_S(u,v) = 0 \qquad \forall v \in H_0^1(\Omega)$$

where $a(\cdot,\cdot)$ is the bilinear form defined in the previous sections and the forcing function is assumed to be null. The stabilization term is of SUPG type:

$$a_S(u,v) = a_{SUPG}(u,v) := \sum_T \delta_T \int_T (-D \triangle u + \beta \cdot \nabla u)(\beta \cdot \nabla v) \, d\Omega \qquad \delta_T = \tau \frac{h^{(T)}}{|\beta|}$$

where $h^{(T)}$ is the length of the largest edge of the triangle, $|\beta| = \sqrt{\beta_x^2 + \beta_y^2}$ is the norm of the transport velocity vector (in our case, assumed to be constant for all the elements) and τ is the constant coefficient used to calibrate the amount of stabilization.
We notice that, since our basis function consists of first order piecewise polynomials, the term involving $\triangle u$ will disappear when passing to FEM formulation. In this way, the SUPG stabilization term will coincide with the steamline diffusion stabilization:

$$a_{SD}(u, v) := \sum_T \int_T \tau \frac{D^{(T)}}{|\beta|^2} \mathbb{P}_e^{(T)} (\beta \cdot \nabla u)(\beta \cdot \nabla v)\, d\Omega$$

where $D^{(T)} := max(D_{xx}, D_{yy})$ (in our case, assumed to be constant for all elements), and $\mathbb{P}_e^{(T)} := \frac{|\beta| h^{(T)}}{D^{(T)}}$ is the Peclet number, locally defined on each triangle.

Given this definition of the stabilization term, we can rewrite it in Galerkin Finite Element formulation, following the same procedure we used for the non-stabilized part. The discretized system now reads:

$$(H + B + S)u = 0$$

where the stabilization matrix is defined by the following elemental contributions:

$$S = \{s_{ij}\} \qquad\qquad s_{ij}^{(T)} = \delta_T \int_T (-D \triangle \phi_j + \beta \cdot \nabla \phi_j)(\beta \cdot \nabla \phi_i)\, d\Omega$$

which can be computed explicitly by using our specific choice of basis (since we mentioned to slighlty different definitions for the basis function, we espress these results in both ways: using coefficients a_i, b_i, c_i or using $\tilde{a}_i$, $\tilde{b}_i$, $\tilde{c}_i$. See previous sections for their definition):

$$s_{ij}^{(T)} = \delta_T |T| (\beta_x b_i + \beta_y c_i)(\beta_x b_j + \beta_y c_j) = \delta_T \frac{1}{4|T|} (\beta_x \tilde{b}_i + \beta_y \tilde{c}_i)(\beta_x \tilde{b}_j + \beta_y \tilde{c}_j)$$

2 Description of the code

The Matlab code implementing the problem has been structured as follows, making use of user-defined functions:

- *(inputData.m)* reading of the data from files (coordinates, triangulation, Dirichlet nodes and values) and definition of the constant coefficients (D, β, τ);
- *(localBasis.m)* element by element computation of the coefficients of the basis functions and of the elemental area;
- *(globBuild.m)* element by element computation and assembling of the matrices of the system (stiffness, transport and stabilization matrices);
- *(imposeBC.m)* imposition of the Dirichlet boundary conditions; both the penalty method and the standard method (by-hand modification of the total matrix and of the right hand side in correspondence of the Dirichlet nodes) have been tested;
- *(solveSys.m)* solution of the system; GMRES has been used, together with the lifting boundary operator;
- *(massBal.m)* computation of the diffusive and convective flux through the boundary, aimed at verifying mass balance and hence at testing the correctness of the solution;
- plotting of the numerical solution and computation of the norm of the residual error.

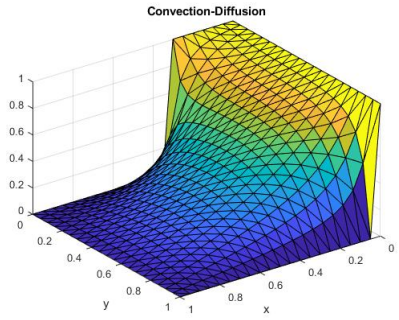
3 Numerical results

To begin with, we analyze the behavior of the numerical solution without stabilization ($\tau = 0$). The transport velocity vector is kept fixed: $\beta = (1, 3)$ and we vary the diffusion coefficient $D := D_{xx} = D_{yy}$ in the range [0.001, 1]. The results are analyzed in a graphical way. In the following we show some examples of plots.

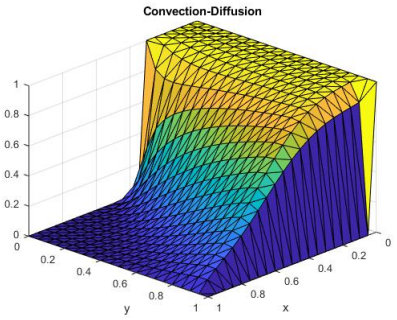
By observing the graphical results, it is immediate to notice the appearance of oscillations as the diffusion coefficient D grows small. The process is gradual: a rough sample of values for D shows that for $D = 1$ the graph is smooth; then at around $D = 0.1$ the gradient of the solution becomes higher and higher along the $y = 1$ axis, but the graph is still stable; then, starting from that region, some oscillations appear, which become quite evident at $D = 0.01$; at $D = 0.001$ the solution is completely ruined by oscillations.

A finer sample of values for D allows to define approximately the threshold $\tilde{D}$ below which the oscillations appear. The result can also be expressed by defining the corresponding pseudo Peclet number (as reference length, we use $h = 0.05$, edge of the triangles in the mesh):

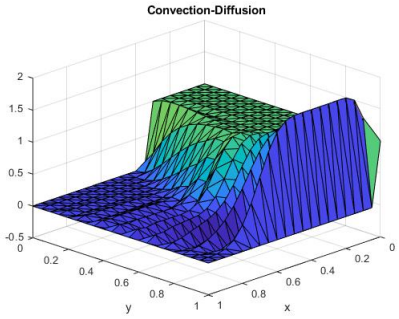
$$\tilde{D} = 0.075 \qquad\qquad \tilde{\mathbb{P}}_e = \frac{|\beta| h^{(T)}}{\tilde{D}} = 2.11$$



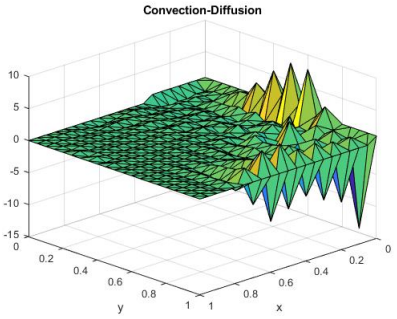
(a) $D = 1$, $\tau = 0$



(b) $D = 0.1$, $\tau = 0$

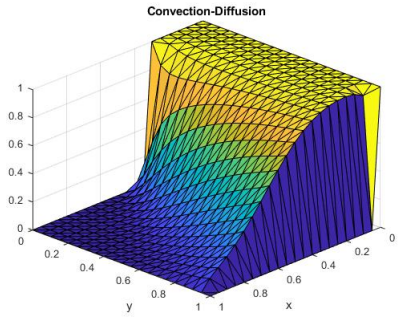


(c) $D = 0.01$, $\tau = 0$

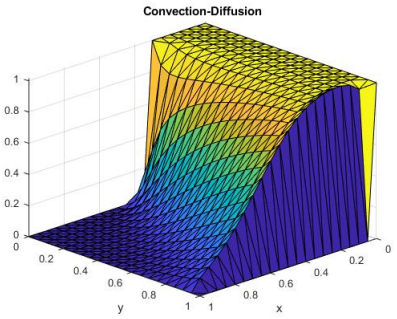


(d) $D = 0.001$, $\tau = 0$

Figure 1: Numerical solutions, no stabilization ($\tau = 0$)



(a) $D = 0.075$, $\tau = 0$



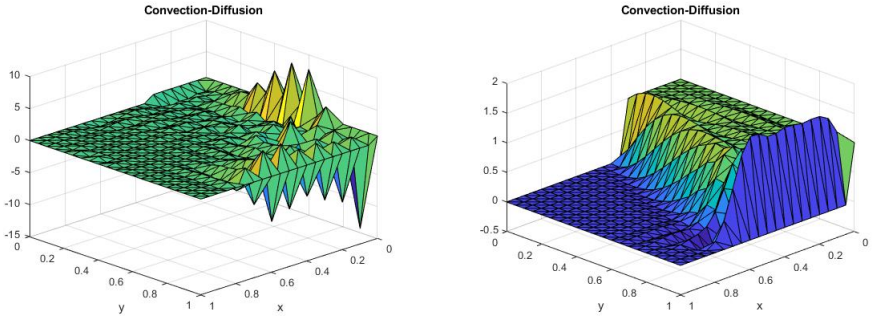
(b) $D = 0.07$, $\tau = 0$

Figure 2: Numerical solutions around threshold, no stabilization ($\tau = 0$)

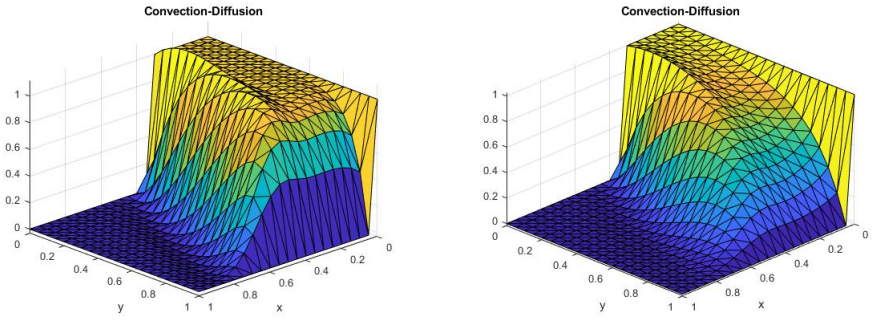
It is now possible to introduce the stabilization term and see what is the effect on the numerical solution. We plotted the solution for five different values of the diffusion coefficient ranging between $D = 0.001$ and $D = 1$: for each value, we let the stabilization coefficient vary from $\tau = 0$ (no stabilization) up to the point where the oscillations are disappeared (around $\tau = 3$). The results are shown below (figures 3 to figure 7).

We notice that, for each value of D, the solution undergoes the same evolution as the stabilization coefficient τ is gradually increased: first, wide oscillations (if present) are reduced to oscillations of smaller amplitude (see $D = 0.001$ and $D = 0.01$, with $\tau \sim 0.1$); then, oscillations are absorbed, but the gradient of the solution along the $y = 1$ axis is still high (see e.g. $D = 0.01$ and $D = 0.05$, with $\tau \sim 1$); finally, as $\tau \sim 5$, all solutions appear to be quite smoothed.

We also notice that, even though the the diffusion coefficient is ranging within three orders of magnitude ($[0.001, 1]$), the values of the stabilization coefficients employed to regularize the solutions are quite similar to each other (ranging approximately in $[1, 5]$ to have a stable solution).

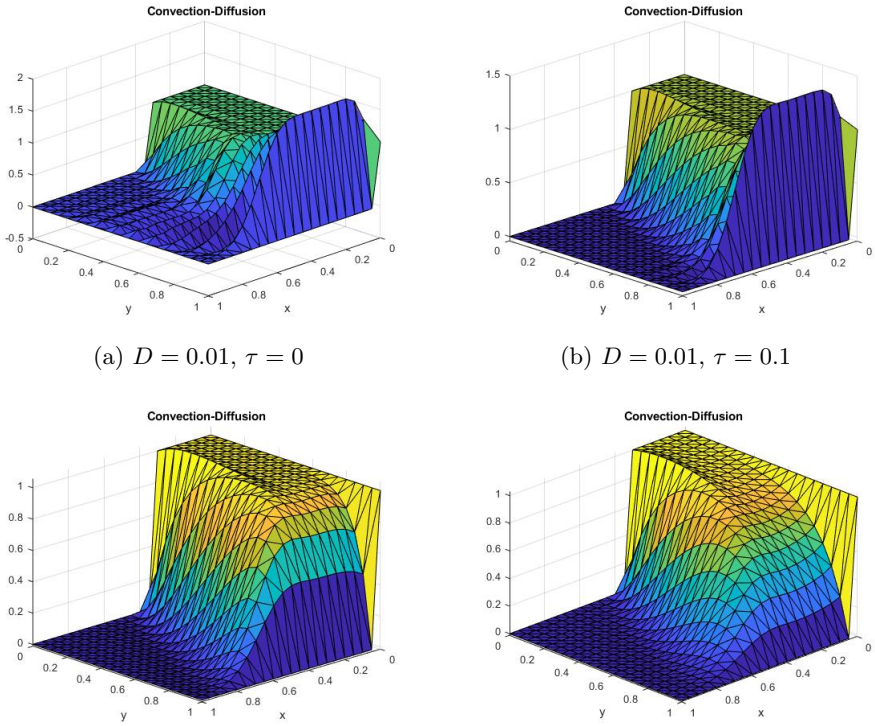


(a) $D = 0.001$, $\tau = 0$ (b) $D = 0.001$, $\tau = 0.1$



(c) $D = 0.001$, $\tau = 1$ (d) $D = 0.001$, $\tau = 5$

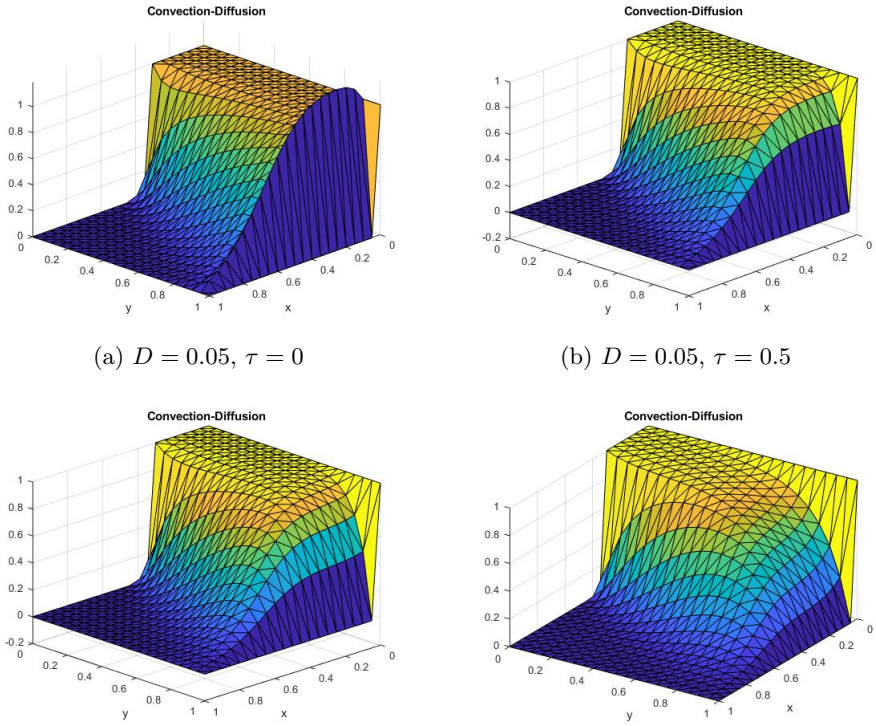
Figure 3: Numerical solutions, $D = 0.001$, with stabilization



(a) $D = 0.01$, $\tau = 0$ (b) $D = 0.01$, $\tau = 0.1$

(c) $D = 0.01$, $\tau = 1$ (d) $D = 0.01$, $\tau = 3$

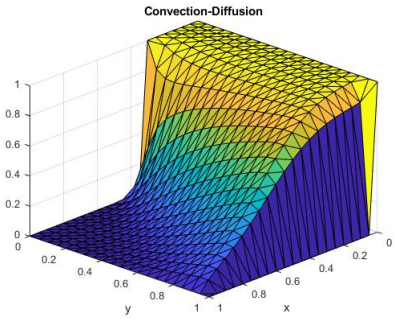
Figure 4: Numerical solutions, $D = 0.01$, with stabilization



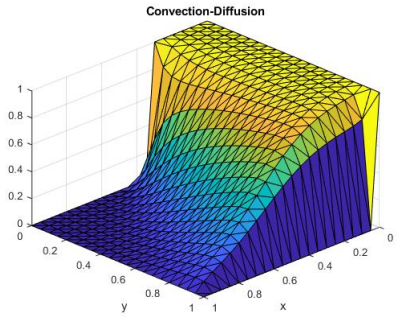
(a) $D = 0.05$, $\tau = 0$ (b) $D = 0.05$, $\tau = 0.5$

(c) $D = 0.05$, $\tau = 1$ (d) $D = 0.05$, $\tau = 3$

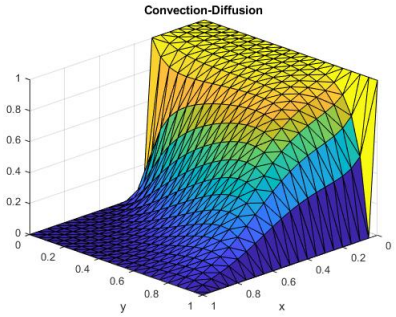
Figure 5: Numerical solutions, $D = 0.05$, with stabilization



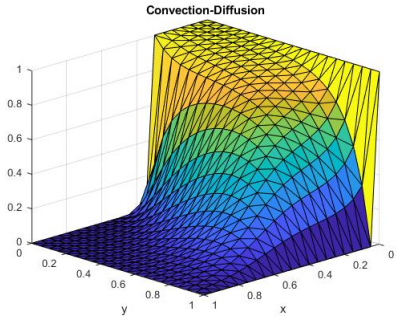
(a) $D = 0.1$, $\tau = 0$



(b) $D = 0.1$, $\tau = 0.1$

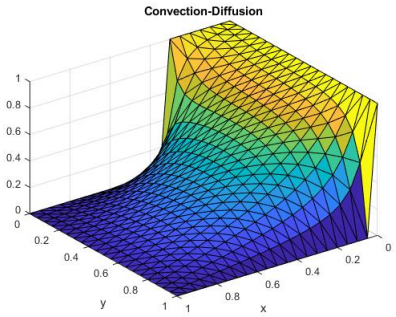


(c) $D = 0.1$, $\tau = 1$

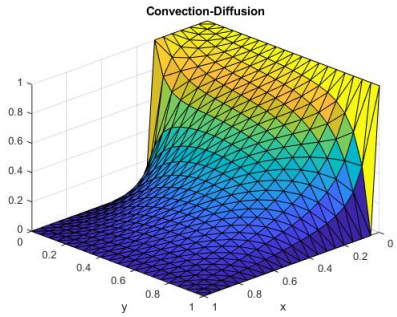


(d) $D = 0.1$, $\tau = 3$

Figure 6: Numerical solutions, $D = 0.1$, with stabilization



(a) $D = 1$, $\tau = 0$



(b) $D = 1$, $\tau = 3$

Figure 7: Numerical solutions, $D = 1$, with stabilization

We may also make some observations on the correctness of the numerical solution. First of all, we had to choose properly how to impose the Dirichlet boundary conditions and how to solve the system. As far as the boundary conditions are concerned, we tried two different methods: first, the Penalty method; then, the classical method involving direct impositions of the values. As for the solution of the system, we used the GMRES algorithm, which could be applied even with a non-symmetric system matrix.

Both methods yield good solutions: all residual norms are of order 10^{-9} or 10^{-10}, meaning that the discretized system is solved with the requested precision. However, when working with the penalty method, we had to make use of a boundary lifting function. This is due to the fact that for large values of the penalty ($\lambda \sim 10^{40}$) the algorithm fails to find the correct solution: the presence of the penalty in the right hand side causes the relative residual to be extremely small from the very beginning, and the algorithm stops at the first iteration, without any actual computation on the solution. The boundary lifting function allows to avoid this problem. Another possibility is to use relatively small values for the penalty ($\lambda \sim 10^{10}$): the solution is still good but less precise (residual norm $\sim 10^{-1}$). The third possibility is to avoid the penalty method and to impose the BCs in the classical way: in correspondence of the Dirichlet nodes, the diagonal term of the matrix is set to 1, while the other values in the corresponding column and row are set to 0. The right hand side is also modified, by imposing the Dirichlet value on the corresponding node and by adding corrections on the other terms in order not to lose any information. The result is equivalent to the penalty method, and doesn't need to make use of the boundary lifting function.

Another numerical check on the correctness of the solution is the mass balance. The idea is that, since we're dealing with a stationary solution, the total flux across the boundary should be zero. Hence, we computed both the convective and the diffusive flux across the boundary, making use of the obtained numerical solution, and we summed them. The results are shown in table 1.

Table 1: Total flux as D, τ vary

τ \ D	0.001	0.01	0.05	0.1	1
0	-1.94	-1.67	-1.13	-0.82	-0.28
0.1	-1.48	-1.35	-0.98	-0.74	-0.28
1	-0.56	-0.54	-0.49	-0.44	-0.27
3	-0.32	-0.32	-0.31	-0.30	-0.26

We notice that the convective flux generally predominates over the diffusive contribution. When using no stabilization, the difference is quite small for $D = 1$ and gradually increases as the diffusion coefficient becomes smaller and smaller; at $D = 0.001$, when the numerical solution is ruined by the oscillations, the total flux is at the highest (in absolute value), and consists mainly of convective flux. On the other side, as stabilization is added, the total flux is progressively reduced, as expected for a regularized solution.