FEF Assignment

Bruno Aguirre Tessaro

May 23, 2016

1. Steady Convection-Diffusion-Reaction

1.a. The weak form/FEM approximation derivation

The derivation of the weak form of the convection-diffusion-reaction problem starts with the strong form of the equations

$$\begin{cases} \mathbf{a}\cdot\nabla u - \nabla\cdot(\nu\nabla u) + \sigma u = s & \text{in } \Omega, \\ u = u_d & \text{in } \Gamma_D, \\ \mathbf{n}\cdot\nu\nabla u = \nu\dfrac{\partial u}{\partial \mathbf{n}} = h & \text{in } \Gamma_N, \end{cases}$$

being Ω, Γ_D, Γ_N and h respectively the domain, the Dirichlet boundary, the Neumann boundary and the prescribed flux. With this, the first equation is integrated over the domain in its residual form and multiplied by the test functions w as

$$\int_\Omega w(\mathbf{a}\cdot\nabla u)d\Omega - \int_\Omega w\nabla\cdot(\nu\nabla u)d\Omega + \int_\Omega w(\sigma u)d\Omega - \int_\Omega wsd\Omega. = 0. \tag{1.1}$$

The diffusion term (second term of Equation 1.1) is integrated by parts in order to weaken its continuity requirements. The residual equations takes the from of

$$\int_\Omega w(\mathbf{a}\cdot\nabla u)d\Omega - \int_{\cancel{\Gamma_D}+\Gamma_N} w\cdot(\mathbf{n}\cdot\nu\nabla u)d\Omega + \int_\Omega \nabla w\cdot(\nu\nabla u)d\Omega + \int_\Omega w(\sigma u)d\Omega - \int_\Omega wsd\Omega, = 0,$$

where the Dirichlet integration of the second term is excluded because the test functions were chosen to be $w = 0$ on Γ_D. Finally, the weak form of the convection-diffusion-reaction problem reads

$$\int_\Omega w(\mathbf{a}\cdot\nabla u)d\Omega + \int_\Omega \nabla w\cdot(\nu\nabla u)d\Omega + \int_\Omega w(\sigma u)d\Omega = \int_\Omega wsd\Omega + \int_{\Gamma_N} whd\Omega,$$

which can be expressed in a compact version as

$$a(w,u) + c(\mathbf{a};w,u) + (w,\sigma u) = (w,s) + (w,h)_{\Gamma_N}.$$

Now using the discretization $u = u^h = N_j u_j$ (summation implied on the repeated indexes) and the Galerkin approximation $w = N_i$, the weak form becomes

$$\Big[a(N_i, N_j) + c(\mathbf{a}; N_i, N_j) + (N_i, \sigma N_j)\Big] u_j = (N_i, s) + (N_i, h)_{\Gamma_N}.$$

With this, the following system of equations can be presented

$$(\mathbf{K} + \mathbf{C} + \mathbf{M})\mathbf{u} = \mathbf{f}$$

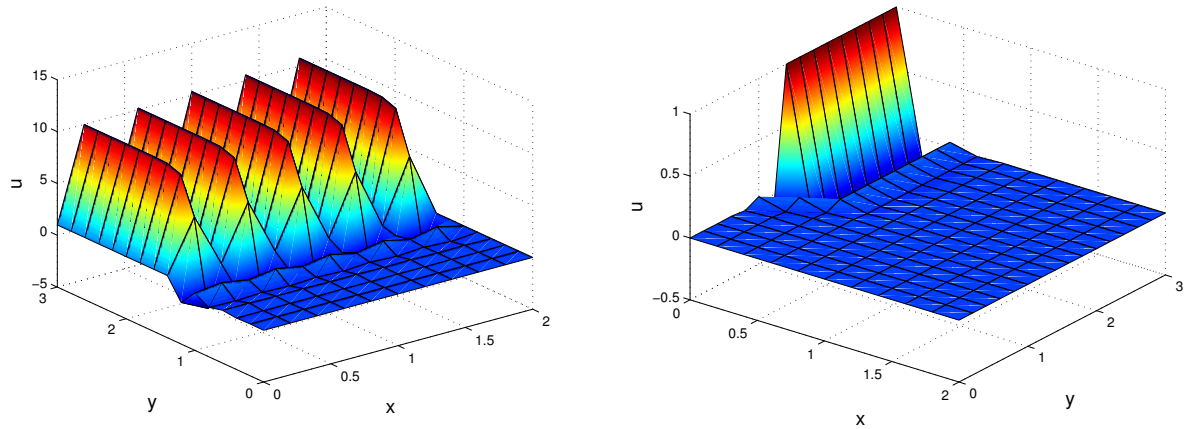
where $\mathbf{K}$, $\mathbf{C}$, $\mathbf{M}$, $\mathbf{u}$ and $\mathbf{f}$ are respectively the diffusion matrix, the convection matrix, the reaction matrix, the unknowns vector and the force vector. This matrices are obtained by the assembling of the finite element method.

1.b. Results For Galerkin Aproximation

Tests are carried out for the Convection-Diffusion-Reaction problem using the Galerkin aproximation to check the implementation of the code supplied. For all the tests, the space discretization h is going to be equal 0.2 with $u = 1$ in Γ_2 and $u = 0$ in Γ_4. The following test cases are being executed:

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

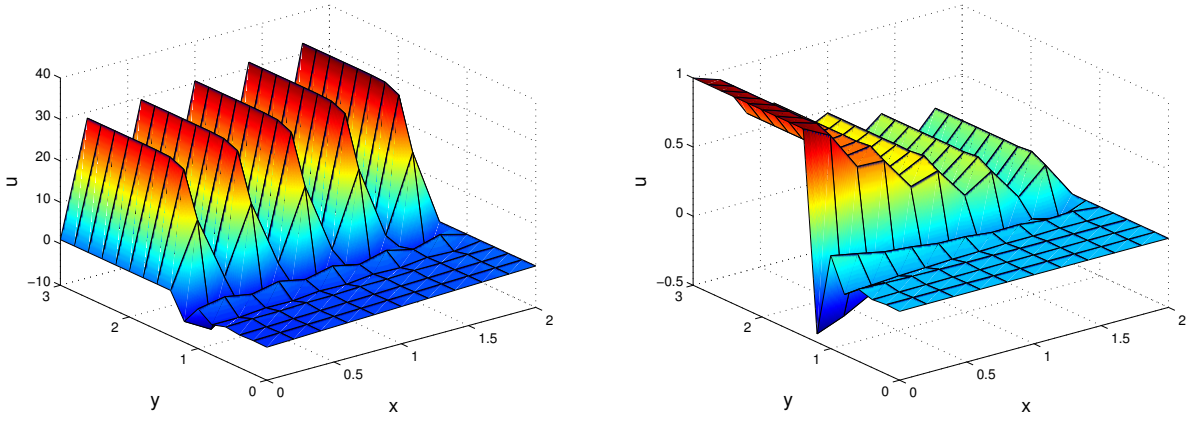
Results obtained using the above parameters can be seem in Figures 1.1 and 1.2. It can be observed that the first, third and fourth set of parameters present oscillatory behaviour. This happens because the Peclet number of this simulations were bigger them 1. The second simulation, which has a Peclet number of 0.01 does not present any oscillation.



(a) First set of parameters, $Pe = 100$.

(b) Second set of parameters, $Pe = 0.01$.

Figure 1.1: Results for the first and second set of parameters using Galerkin



(a) Third set of parameters, $Pe = 100$.

(b) Fourth set of parameters, $Pe = 100$.

Figure 1.2: Results for the third and fourth set of parameters using Galerkin

Now considering the same Galerkin approach without modifying the parameters a, ν, σ and s, one can diminish this oscillations by decreasing the space discretizaion h and consequently, decreasing the Peclet number. The exact value of the discretization can be obtained by

$$Pe = \frac{ha}{\nu} = 1, \qquad\qquad h = \frac{\nu}{a} = 10^{-3}.$$

The use of this discretization size may solve the oscillations problem but gives rise to new ones. For instance, the computational cost is highly increased, making the simulation time to raise exponentially. Another problem encountered is that the code has to be prepared to work with really large matrices, otherwise it cannot handle the allocations. For instance, the Matlab code provided can only handle the discretization $h = 10^{-3}$ with he implementation of sparse matrices, otherwise, it returns an error.

1.c. SUPG and GLS Stabilization

1.c.1. Method Derivation

The fourth set of parameters (together with the first and third sets) present oscillatory results. As mentioned, this problem can be fixed using a smaller discretization size, however, it is easier and less costly to use stabilization techniques like SUPG and GLS.

The application of stabilization techniques for convection-dominated problems consists in adding an extra term (function of the differential equation residual) on the Galerkin formulation. This term will not only bring stability but also produced accurate results. The Galerkin formulation with the stability term can be expressed as

$$a(w,u) + c(\mathbf{a};w,u) + (w,\sigma u) + \sum_e \int_{\Omega_e} P(w)\tau R(u) d\Omega = (w,s) + (w,h)_{\Gamma_N},$$

where $P(w)$ is the term related to the stabilization method, τ is the stabilization parameter and $R(u)$ is the differential equation's residual. The stabilization parameter is calculated using

$$\tau = \frac{\bar{\nu}}{||\mathbf{a}||^2}, \qquad\qquad \bar{\nu} = \frac{\beta a h}{2},$$

where β is a free parameter. Now the Streamline Upwind Petrov-Galerkin stabilization technique can be defined. It consists in applying the SU test function to all the terms of the Galerkin weak form. The SUPG stabilization term takes the form of

$$P(w) = \mathbf{a} \cdot \nabla w,$$

resulting on the following weak form expression

$$a(w,u) + c(\mathbf{a}; w, u) + (w, \sigma u) + \sum_e \int_{\Omega_e} (\mathbf{a} \cdot \nabla w) \tau R(u) d\Omega = (w, s) + (w, h)_{\Gamma_N}.$$

The Galerkin Least Squares stabilization technique is defined by a weighted least-squares formulation of the original Galerkin weak form. The stabilization term takes the form of

$$P(w) = \mathbf{a} \cdot \nabla w - \nabla \cdot (\nu \nabla u) + \sigma w,$$

resulting on the following weak form expression

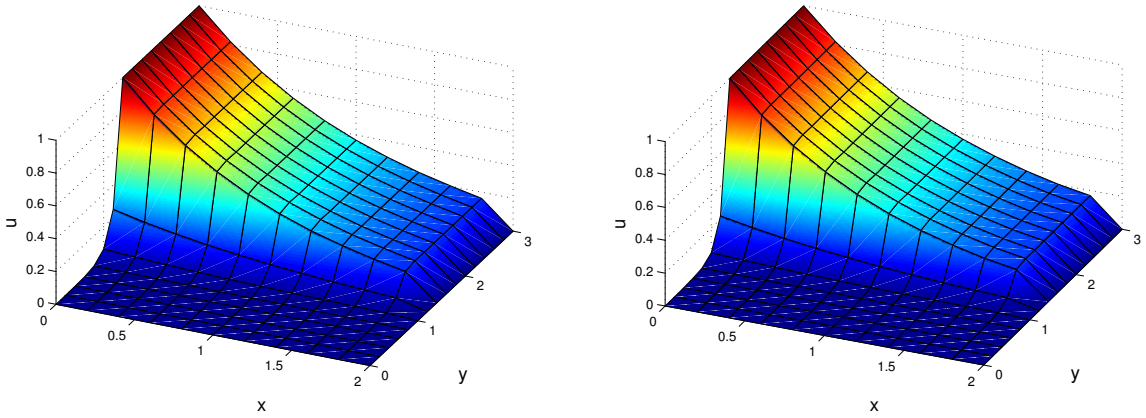
$$a(w,u) + c(\mathbf{a}; w, u) + (w, \sigma u) + \sum_e \int_{\Omega_e} (\mathbf{a} \cdot \nabla w - \nabla \cdot (\nu \nabla u) + \sigma w) \tau R(u) d\Omega = (w, s) + (w, h)_{\Gamma_N}.$$

Its important to remark that when linear elements are used, the second derivative term of the stabilization vanishes in both SUPG and GLS stabilization techniques. The implementation of this techniques on the Matlab code consisted in adding the stabilization term to the *FEM_system.m* file. The code implementation can be seem as following

```
    elseif method == 2 % SUPG
        Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) + N_ig'*(ax*Nx+ay*Ny) ...
            + sigma*(N_ig)'*N_ig + tau*((ax*Nx+ay*Ny)'*(ax*Nx+ay*Ny)...
            + (ax*Nx+ay*Ny)'*sigma*N_ig))*dvolu;
        fe = fe + (N_ig+tau*(ax*Nx+ay*Ny))'*(f_ig*dvolu);

    elseif method == 3  % GLS
        Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) + N_ig'*(ax*Nx+ay*Ny) ...
            + sigma*(N_ig)'*N_ig + tau*(((ax*Nx+ay*Ny)'+sigma*N_ig')...
            *(ax*Nx+ay*Ny) + ((ax*Nx+ay*Ny)'+sigma*N_ig')*sigma*N_ig))...
            *dvolu;
        fe = fe + (N_ig+tau*(ax*Nx+ ay*Ny + sigma*N_ig))'*(f_ig*dvolu);
```

1.c.2. Results

Results obtained for the fourth set of parameters using SUPG and GLS can be seem in Figure 1.3. Even though the Peclet number is high, no oscillations are present in any of the graphs. Also, no major difference between the methods can be spotted. However, mathematically, the GLS formulation for linear elements amplifies the oscillations given by the Galerkin formulation by a factor of $1 + \sigma\tau$ more them SUPG.



(a) SUPG (b) GLS

Figure 1.3: Results for the fourth set of parameters using stabilization tecniques.

1.d. Changing of Boundary Conditions

The fourth set of parameters with the use of SUPG was chosen for this problem. First, the boundary conditions of $u = 2$ in Γ_2 and $u = 1$ in Γ_4 were applied in the same way as in the problems before. After obtaining a solution using this B.C's, the reaction fluxes corresponding with the Γ_4 were obtained and stored in a file, as shown in the following code.

```
% Calculate and store reaction fluxed provided by Gamma4 = 1
RF_Gamma4 = zeros(neq,1);
for i=1:length(nodesDir0)
    RF_Gamma4(nodesDir0(i,1),1) = K(nodesDir0(i,1),:)*Temp(:);
end
fid=fopen('RF_Gamma4.txt','w');
fprintf(fid, '%20d \n', RF_Gamma4);
fclose(fid);
```

After that, a second calculation was done but without the Dirichlet conditions in Γ_4. Instead, the reaction fluxes calculated on the previous simulation were summed to the force vector (imposition of the Neumann B.C's). The results obtained for both methods can be seem in Figure 1.4. It can be observed that even for different implementation of the boundary conditions the results are the same meaning that the implementation used was accurate.

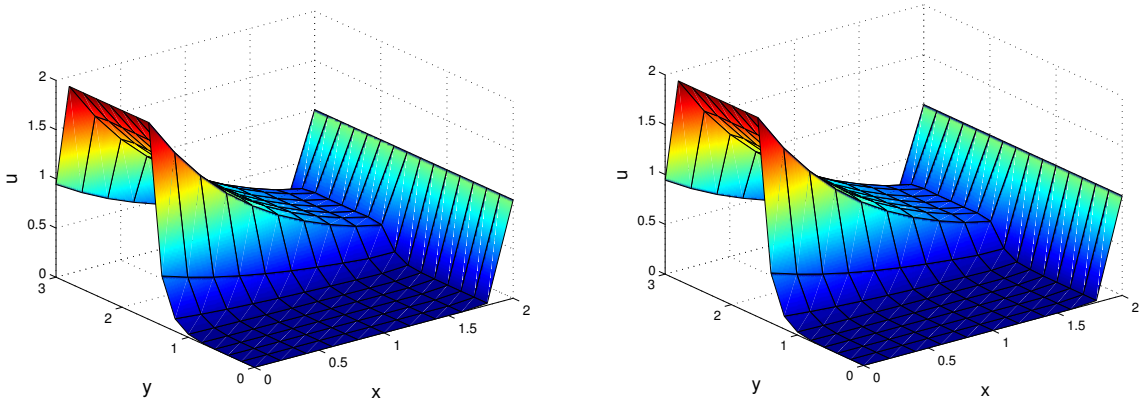
2. Transient Convection-Diffusion-Reaction

2.a. Method Derivation

2.a.1. Crank-Nicholson

The Crank-Nicholson time integration method can be expressed as a Pade method of approximation $R_{1,1}$. For a linear problem, the general implicit Pade methods can be me expressed as

$$\frac{\Delta \mathbf{u}}{\Delta t} + \mathbf{W}L(\Delta \mathbf{u}) = \mathbf{w}[s^n - L(u^n)] + \mathbf{W}\Delta \mathbf{s}, \tag{2.1}$$



(a) Results with Dirichlet Γ_4 (b) Results with Neumann Γ_4

Figure 1.4: Results obtained with different implementation for Γ_4 boundary conditions.

being $\mathbf{W}$ and $\mathbf{w}$ parameters that characterize the method used. The problem to be solved will not consider a source term or non-zero Neumman Boundary conditions, so their corresponding equation terms are going to be omitted from the derivation. The L operator is the spatial part of the differential equation and has the form of

$$L(\Delta\mathbf{u}) = \mathbf{a}\cdot\nabla(\Delta\mathbf{u}) - \nabla\cdot(\nu\nabla(\Delta\mathbf{u})) + \sigma(\Delta\mathbf{u})$$
$$L(u^n) = \mathbf{a}\cdot\nabla u^n - \nabla\cdot(\nu\nabla u^n) + \sigma(\Delta u^n).$$

For Crank-Nicholson, the method parameters are described as

$$\Delta\mathbf{u} = \Delta u = u^{n+1} - u^n, \qquad \mathbf{W} = 1/2, \qquad \mathbf{w} = 1,$$

resulting on the final CN integration method

$$\frac{\Delta u}{\Delta t} + \frac{1}{2}\big[\mathbf{a}\cdot\nabla(\Delta\mathbf{u}) - \nabla\cdot(\nu\nabla(\Delta\mathbf{u})) + \sigma(\Delta\mathbf{u})\big] = -\big[\mathbf{a}\cdot\nabla u^n + \nabla\cdot(\nu\nabla u^n) + \sigma u^n\big].$$

The space discretization follows the same procedure as in Section 1.a, and takes the form of

$$(w, \frac{\Delta u}{\Delta t}) + \frac{1}{2}\big[c(\mathbf{a}; w, \Delta u) + a(w, \Delta u) + (w, \sigma\Delta u)\big] = -\big[c(\mathbf{a}; w, u^n) + a(w, u^n) + (w, \sigma u^n)\big],$$

$$(N_i, N_j)\frac{\Delta u_j}{\Delta t} + \frac{1}{2}\big[c(\mathbf{a}; N_i, N_j) + a(N_i, N_j) + (N_i, \sigma N_j)\big]\Delta u_j = -\big[c(\mathbf{a}; N_i, N_j) + a(N_i, N_j) + (N_i, \sigma N_j)\big]u_j^n.$$

Finally, the discretization proposed can be solved using the following system of equations

$$\big[\mathbf{M} + \frac{\Delta t}{2}(\mathbf{K} + \mathbf{C} + \mathbf{M})\big]\Delta\mathbf{u} = -\Delta t\big[\mathbf{K} + \mathbf{C} + \mathbf{M}\big]\mathbf{u}^n. \tag{2.2}$$

2.a.2. Pade $R_{2,2}$

The Pade $R_{2,2}$ time integration method derivation follows a similar procedure as the Crank-Nicholson method. It starts with the method coefficients

$$\Delta \mathbf{u} = \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix} = \begin{bmatrix} u^{n+1/2} - u^n \\ u^{n+1} - u^{n+1/2} \end{bmatrix} \qquad \mathbf{W} = \frac{1}{24} \begin{bmatrix} 7 & -1 \\ 13 & 5 \end{bmatrix}, \qquad \mathbf{w} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

being plugged in the Equation 2.a.1. Considering the same L operator, the resulting equations can be be expressed as

$$\begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix} \frac{1}{\Delta t} + \frac{1}{24} \begin{bmatrix} 7 & -1 \\ 13 & 5 \end{bmatrix} \begin{bmatrix} a \cdot \nabla(\Delta u_1) - \nabla \cdot (\nu \nabla(\Delta u_1)) + \sigma(\Delta u_1) \\ a \cdot \nabla(\Delta u_2) - \nabla \cdot (\nu \nabla(\Delta u_2)) + \sigma(\Delta u_2) \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} (a \cdot \nabla(u^n) - \nabla \cdot (\nu \nabla(u^n)) + \sigma(u^n)).$$

Following the same procedure for the weak form as the in Section 1.a, the $R_{2,2}$ Pade method has the form of

$$\begin{bmatrix} (w, \Delta u_1) \\ (w, \Delta u_2) \end{bmatrix} \frac{1}{\Delta t} + \frac{1}{24} \begin{bmatrix} 7 & -1 \\ 13 & 5 \end{bmatrix} \begin{bmatrix} c(\mathbf{a}; w, \Delta u_1) + a(w, \Delta u_1) + (w, \sigma \Delta u_1) \\ c(\mathbf{a}; w, \Delta u_2) + a(w, \Delta u_2) + (w, \sigma \Delta u_2) \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} c(\mathbf{a}; w, u^n) + a(w, u^n) + (w, \sigma u^n).$$

Finally, with the FEM discretization together with Galerkin, the resulting system of equations can be expressed as

$$\left[\begin{bmatrix} \mathbf{M} \\ \mathbf{M} \end{bmatrix} + \frac{\Delta t}{24} \begin{bmatrix} 7 & -1 \\ 13 & 5 \end{bmatrix} \begin{bmatrix} \mathbf{C} + \mathbf{K} + \mathbf{M} \\ \mathbf{C} + \mathbf{K} + \mathbf{M} \end{bmatrix} \right] \begin{bmatrix} \Delta \mathbf{u}_1 \\ \Delta \mathbf{u}_2 \end{bmatrix} = \frac{\Delta t}{2} \begin{bmatrix} \mathbf{C} + \mathbf{K} + \mathbf{M} \\ \mathbf{C} + \mathbf{K} + \mathbf{M} \end{bmatrix} \begin{bmatrix} \mathbf{u}^n \\ \mathbf{u}^n \end{bmatrix}. \tag{2.3}$$

2.a.3. SUPG Derivation

In the case that problem is convection-dominated, is likely that stabilization is required in order to diminish oscillations. The weak form of the general implicit Pade method with stabilization can be expressed as

$$(w, \frac{\Delta \mathbf{u}}{\Delta t}) - (w, \mathbf{W} \Delta \mathbf{u}_t) + \sum_e (\tau P(w), R(\Delta \mathbf{u})) = (w, \mathbf{w} u_t^n),$$

where

$$P(w) = \mathbf{W}(\mathbf{a} \cdot \nabla) w, \qquad \tau = \Big[\frac{\mathbf{W}^{-1}}{\Delta t} + (\frac{2a}{h} + \frac{4\nu}{h^2} + \sigma) \mathbf{I} \Big] \mathbf{W}^{-1},$$

for the SUPG method. Now following the same procedure as before but carrying the stabilization term, the stabilized Crank Nicholson system of equations can be expressed as

$$\big[\mathbf{M} + \frac{\Delta t}{2} (\mathbf{K} + \mathbf{C} + \mathbf{M}) + \Delta t \mathbf{S} \big] \Delta \mathbf{u} = -\Delta t \big[\mathbf{K} + \mathbf{C} + \mathbf{M} + \mathbf{S}^n \big] \mathbf{u}^n,$$

where

$$\mathbf{S} = \tau \mathbf{W}(\mathbf{a}\cdot\nabla)w\Big[\frac{1}{\Delta t} + \mathbf{W}L\Big], \qquad\qquad \mathbf{S}^n = \tau \mathbf{W}(\mathbf{a}\cdot\nabla)w[\mathbf{w}L].$$

Using the same procedure, the final system of equations of the SUPG Pade $R_{2,2}$ method with has the form of

$$\left[\begin{bmatrix}\mathbf{M}\\ \mathbf{M}\end{bmatrix} + \frac{\Delta t}{24}\begin{bmatrix}7 & -1\\ 13 & 5\end{bmatrix}\begin{bmatrix}\mathbf{C}+\mathbf{K}+\mathbf{M}\\ \mathbf{C}+\mathbf{K}+\mathbf{M}\end{bmatrix}\Delta t\begin{bmatrix}\mathbf{S}\\ \mathbf{S}\end{bmatrix}\right]\begin{bmatrix}\Delta\mathbf{u}_1\\ \Delta\mathbf{u}_2\end{bmatrix} = \frac{\Delta t}{2}\begin{bmatrix}\mathbf{C}+\mathbf{K}+\mathbf{M}+\mathbf{S}^n\\ \mathbf{C}+\mathbf{K}+\mathbf{M}+\mathbf{S}^n\end{bmatrix}\begin{bmatrix}\mathbf{u}^n\\ \mathbf{u}^n\end{bmatrix}.$$

2.b. Code Implementation

The code implementation starts with the completion of the code provided. The matrices **K**, **M**, and **C** together with the boundary condition matrices were already computed in the code, meaning that the time integration method and the system solving would have to be implemented.

2.b.1. Crank-Nicholson Implementation

In order to implement the method, the function *System_CN.m* was created. This function receives the assembled matrices **K**, **M**, and **C** and grouped it up to return the system described in Equation 2.a.1 as following

```
    case 1 % Crank-Nicholson + Galerkin
    A = M + 1/2*dt*(C + nu*K + sigma*M);
    B = -dt*(C + nu*K + sigma*M);

    case 2 % Crank-Nicholson + SUPG
    A = M + 1/2*dt*(C + nu*K + sigma*M) + dt*S;
    B = -dt*(C + nu*K + sigma*M) - dt*Su;
```

The matrices A and B are returned from this function to the main, where they will be used to solve the linear system of equations together with the lagrange multipliers as

```
for n=1:nStep % Solution of the Crank-Nicholson system
    du = [A , ADir' ; ADir  zDir]\[B*u(:,n) ; bDir];
    u(:,n+1) = u(:,n) + du(1:nNodes,1);
end
```

2.b.2. Pade $R_{2,2}$ Implementation

The Pade implementation is a more involved, because it has to work with the 2-step system proposed in the previous section. The resulting multiplication of the second term of Equation 2.a.2 result in a crossed system, meaning that the matrices have to be separated in blocks depending on their corresponding coefficients. In the elemental level, this matrices are constructed with the following code

```
% K matrix for Pade method
Ke_Top = Ke_Top + Nxi_ig'*(W(1,1)*Nxi_ig + W(1,2)*Nxi_ig)*dvolu;
Ke_Bot = Ke_Bot + Nxi_ig'*(W(2,1)*Nxi_ig + W(2,2)*Nxi_ig)*dvolu;

% M matrix for Pade method
Me_Top = Me_Top + N_ig'*(W(1,1)*N_ig + W(1,2)*N_ig)*dvolu;
Me_Bot = Me_Bot + N_ig'*(W(2,1)*N_ig + W(2,2)*N_ig)*dvolu;

% C matrix for Pade method
Ce_Top = Ce_Top + N_ig'*(W(1,1)*aGradN + W(1,2)*aGr1adN)*dvolu;
Ce_Bot = Ce_Bot + N_ig'*(W(2,1)*aGradN + W(2,2)*aGradN)*dvolu;

% Stabilization matrix for delta u
Se_Top = Se_Top + (tau(1,1) + tau(1,2))*(W(1,1)*aGradN + ...
         W(1,2)*aGradN)'*(N_ig/dt - (W(1,1)*nu*Lapla(ngaus,:) + ...
         W(1,2)*nu*Lapla(ngaus,:)) + (W(1,1)*aGradN + W(1,2)*aGradN)...
         + (sigma*W(1,1)*N_ig + sigma*W(1,2)*N_ig))*dvolu;
Se_Bot = Se_Bot + (tau(2,1) + tau(2,2))*(W(2,1)*aGradN + ...
         W(2,2)*aGradN)'*(N_ig/dt - (W(2,1)*nu*Lapla(ngaus,:) + ...
         W(2,2)*nu*Lapla(ngaus,:)) + (W(2,1)*aGradN + W(2,2)*aGradN)...
         + sigma*(W(2,1)*N_ig + sigma*W(2,2)*N_ig))*dvolu;

% Stabilization matrix for u_n
Sue_Top = Sue_Top + (tau(1,1) + tau(1,2))*(W(1,1)*aGradN + ...
         W(1,2)*aGradN)'*(aGradN - nu*Lapla(ngaus,:) + sigma*N_ig)*dvolu;
Sue_Bot = Sue_Bot + (tau(2,1) + tau(2,2))*(W(2,1)*aGradN + ...
         W(2,2)*aGradN)'*(aGradN - nu*Lapla(ngaus,:) + sigma*N_ig)*dvolu;
```

With the assembled matrices, the function *System_Pade* was created to assemble the linear system of equations as in Equation 2.a.2. Its important to remark that the set of equations has to be solved simultaneously so the matrices have to be concatenated in blocks as shown in the following code

```
        case 3 % Pade R22
        A = [M zeros(size(M)) ; zeros(size(M)) M] ...
             + dt*([C_Top C_Bot ; C_Top C_Bot]    ...
             + nu*[K_Top K_Bot ; K_Top K_Bot]     ...
             + sigma*[M_Top M_Bot ; M_Top M_Bot]);

        B = -dt*([C, zeros(size(C)) ; zeros(size(C)), C] ...
            + nu*[K, zeros(size(K)) ; zeros(size(K)), K] ...
            + sigma*[M, zeros(size(M)) ; zeros(size(M)), M]);

        case 4 % Pade R22 + SUPG
        A = [M zeros(size(M)) ; zeros(size(M)) M] ...
             + dt*([C_Top C_Bot ; C_Top C_Bot]    ...
             + nu*[K_Top K_Bot ; K_Top K_Bot]     ...
             + sigma*[M_Top M_Bot ; M_Top M_Bot]  ...
             + [S_Top S_Bot ; S_Top S_Bot]);

        B = -dt*([C, zeros(size(C)) ; zeros(size(C)), C] ...
            + nu*[K, zeros(size(K)) ; zeros(size(K)), K] ...
            + sigma*[M, zeros(size(M)) ; zeros(size(M)), M] ...
            + [Su_Top Su_Top ; Su_Top Su_Bot]);
```

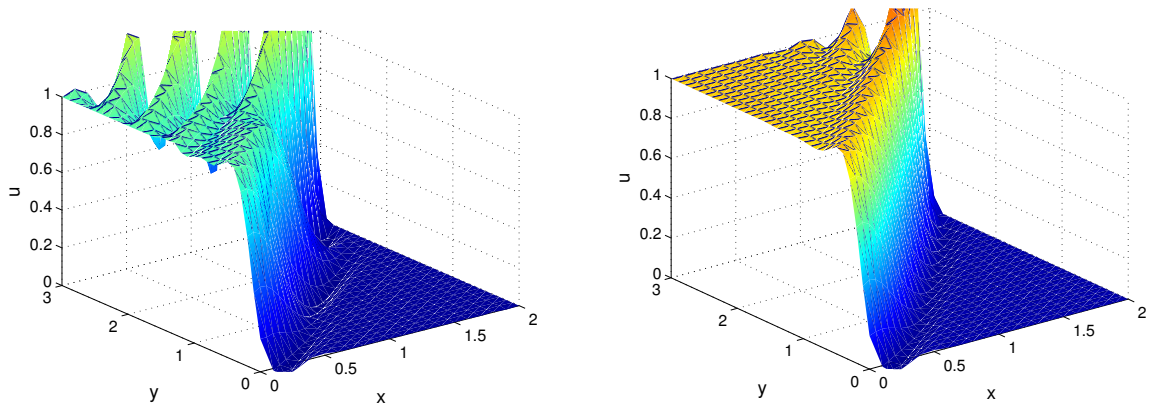
Now with A and B, the system of equations can be solved together with the Lagrange multipliers for the boundary conditions imposition. The solution of the next step is now computed making $\Delta u = \Delta u_1 + \Delta u_2$ as the following code

```
for n=1:nStep
        f = [B*1/2*[u(:,n) ; u(:,n)] ; bDir];
        K = [A , ADir' ; ADir, zDir];

        du_aux = K \ f;
        du1 = du_vec(1:nNodes,1);

        du2 = du_vec(nNodes+1:2*nNodes,1);
        du = du1+du2;
        u(:,n+1) = u(:,n) + du;

end
```

2.c. Results

The code provided was adapted to solve for the proposed domain with a initial condition of $x_0 = x(2-x)$. Considering that the formulation works with increments, it is impossible to solve the problem with the proposed boundary conditions because the initial condition do not respect the boundaries. So the boundary conditions in Γ_2 and Γ_4 had to applied directly to the initial condition

Now choosing $t^n = 5$ (to achieve a steady solution), a time discretization $h_t = 20$, $\nu = 0.1$, $\sigma = 0.001$, $a = (-x, -y)$ and $h = 0.1$, the results for the Galerkin Crank-Nicholson and Galerkin Pade can be seem in Figures 2.1.



(a) Crank-Nicholson

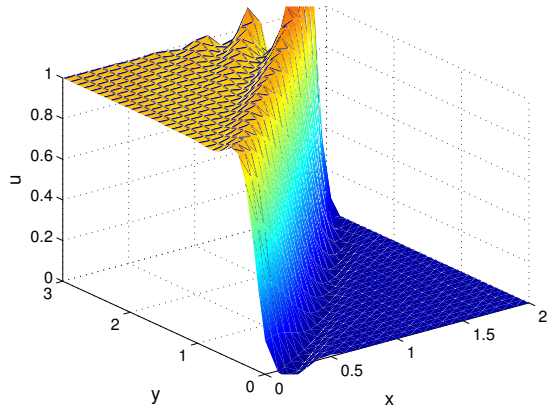
(b) Pade $R_{2,2}$

Figure 2.1: Results obtained for Crank-Nicholson and Pade $R_{2,2}$ using Galerkin for $h_t = 20$.

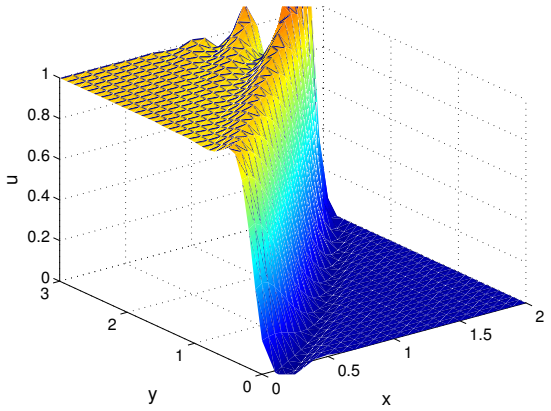
.

It can be observed that the Crank-Nicholson method experience a lot of oscillations, differently from Pade. This happens because Pade is a higher order scheme, so it can handle a smaller time discretizaion. With the increase of the time discretizaion, the Crank-Nicholson method behaves like Pade, as can be seem in Figure 2.2.

Its important to notice that although Pade can handle smaller discretizations, its significantly slower, becauase it has to solve for twice as many nodes as the CN method. Now the time discretization is going to be fixed to $h_t = 110$ and the space discretization is going to be devided by 2, results for this case can be seen in Figure 2.3.

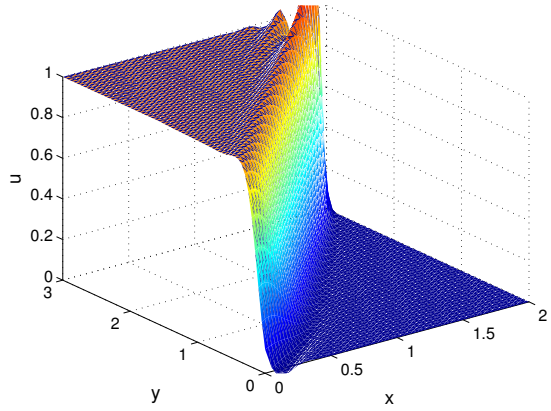


(a) Crank-Nicholson

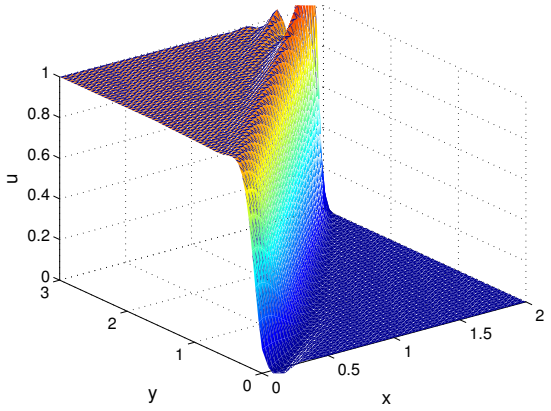


(b) Pade $R_{2,2}$

Figure 2.2: Results obtained for Crank-Nicholson and Pade $R_{2,2}$ using Galerkin for $h_t = 110$.
.



(a) Crank-Nicholson



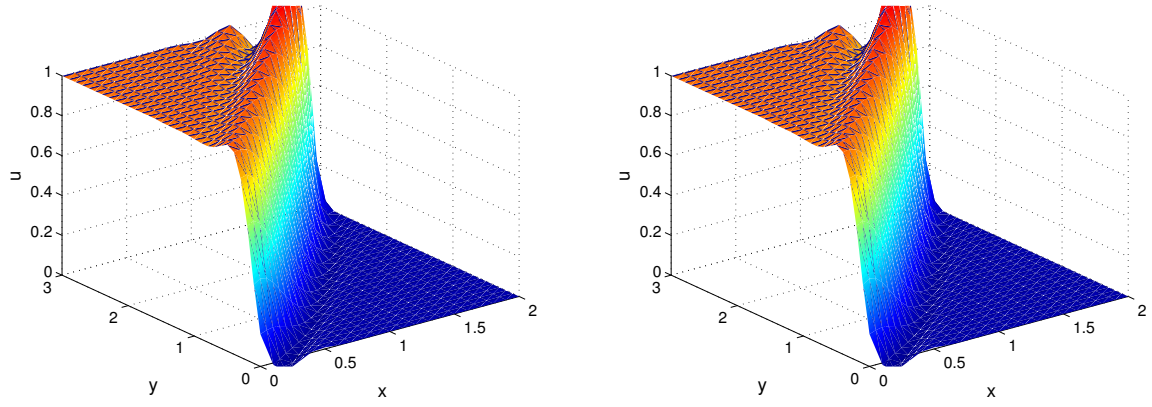
(b) Pade $R_{2,2}$

Figure 2.3: Results obtained for Crank-Nicholson and Pade $R_{2,2}$ using Galerkin for $h = 0.05$.
.

It can be observed that a finer space discretization diminishes the oscillations present on the boundary layer. This also can be achieved using the stabilizations methods, where oscilations are diminished by the use of an up-wind formulation and artificial diffusion. Results for the space discretiztion $h = 0.1$ with SUPG can be seem in Figure 2.4. It can be observed that the stabilization methods diminish the oscillations, even for a courser mesh.

2.c.1. Quadratic Elements

In order to check the effects of quadratic elements on the results a course mesh will be chosen and a simulation will be carried out with Crank Nicholson method using linear and quadratic elements. Results for the same parameters as in the previous section with the use of a space discretization $h = 0.1$ can be seem in Figure 2.5.

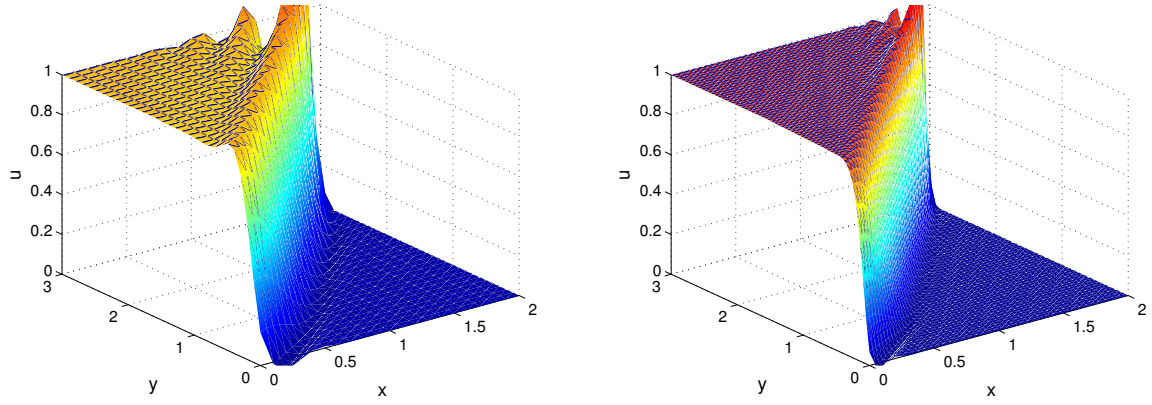


(a) Crank-Nicholson

(b) Pade $R_{2,2}$

Figure 2.4: Results obtained for Crank-Nicholson and Pade $R_{2,2}$ using Galerkin for $h = 0.05$.

.



(a) Linear Elements

(b) Quadratic Elements

Figure 2.5: Results obtained for Crank-Nicholson using linear and quadratic elements with $h = 0.1$

.

As expected, results show that for the same discretization quadratic elements provide more accurate results then linear elements. This expected because more nodes are being used and the solution has a higher order approximation. However, this increase of number of nodes also brings computational cost, with the quadratic elements being way more slow then the linear elements.

Remark: The other available methods are not presented in this section because they all show the same behaviour as Crank Nicholson for quadratic elements. However the code is also prepared to solve for Pade, SUPG Crank Nicholson and SUPG Pade with quadratic elements.

3. Stokes and Navier-Stokes

3.a. Stokes Problem

3.a.1. Problem Derivation

First, the weak form and finite element approximation of the problem is going to be presented. The derivation starts with the strong form of the steady state Stokes Equations using the Linear Stokes Law as

$$\begin{cases} -\nu\nabla^2\mathbf{v} + \nabla p = \mathbf{b} & \text{in } \Omega, \\ \nabla \cdot \mathbf{v} = 0 & \text{in } \Omega, \\ \mathbf{v} = \mathbf{v}_d & \text{in } \Gamma_D, \\ -p\mathbf{n} + \nu(\mathbf{n}\cdot\nabla)\mathbf{v} = \mathbf{t} & \text{in } \Gamma_N, \end{cases}$$

where p is the dynamic pressure, ν is the kinematic viscosity and $\mathbf{t}$ is the prescribed tractions. The derivation of the weak form goes on integrating the first and second equation over the domain and multiplying by test functions $\mathbf{w}$ and q as

$$\int_\Omega \mathbf{w}\cdot(-\nu\nabla^2\mathbf{v})d\Omega + \int_\Omega \mathbf{w}\cdot(\nabla p)d\Omega - \int_\Omega \mathbf{w}\cdot\mathbf{b}d\Omega = 0,$$
$$\int_\Omega q(\nabla\cdot\mathbf{v})d\Omega = 0.$$

Now the viscous and pressure terms are integrated by parts in order to weaken their continuity requirements resulting in

$$\int_\Omega \nabla\mathbf{w} : \nu\nabla\mathbf{v}d\Omega - \int_{\cancel{\Gamma_D}+\Gamma_N} \mathbf{w}\cdot\nu(\mathbf{n}\cdot\nabla\mathbf{v})d\Gamma - \int_\Omega p\nabla\cdot\mathbf{w}d\Omega + \int_{\cancel{\Gamma_D}+\Gamma_N} \mathbf{n}p\cdot\mathbf{w}d\Omega - \int_\Omega \mathbf{w}\cdot\mathbf{b}d\Omega = 0,$$
$$\int_\Omega q(\nabla\cdot\mathbf{v})d\Omega = 0,$$

where the Dirichelet boundary integration is cancelled because $\mathbf{w} = \mathbf{0}$ in Γ_D. Now grouping the Γ_N terms into $\mathbf{t}$ and rearranging the equations, the weak form of the Stokes Equations take the form of

$$\int_\Omega \nabla\mathbf{w} : \nu\nabla\mathbf{v}d\Omega - \int_\Omega p\nabla\cdot\mathbf{w}d\Omega = \int_\Omega \mathbf{w}\cdot\mathbf{b}d\Omega + \int_{\Gamma_N} \mathbf{w}\cdot\mathbf{t}d\Gamma,$$
$$\int_\Omega q(\nabla\cdot\mathbf{v})d\Omega = 0,$$

which can be rewritten in compact form as

$$a(\mathbf{w},\mathbf{v}) + b(\mathbf{w},p) + b(v,q) = b(\mathbf{w},\mathbf{b}) + b(\mathbf{w},\mathbf{t})_{\Gamma_N}.$$

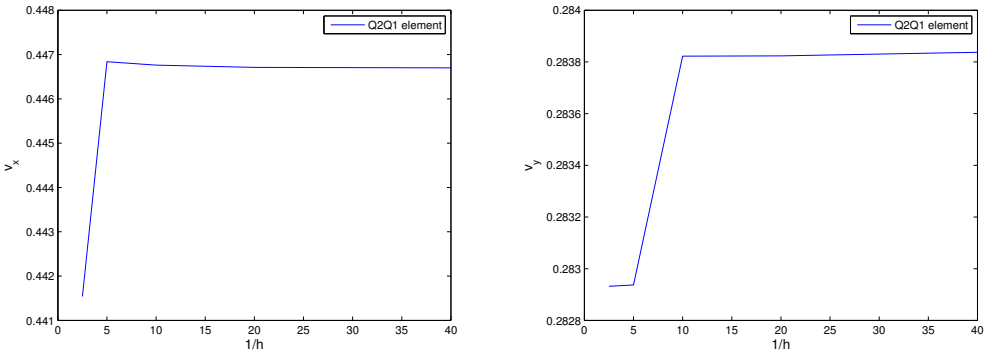
Now using the discretization $v = u^h = N_A u_{iA}$ and $p = p^h = \hat{N}_{\hat{A}} p_{\hat{A}}$ together with Galerkin the equations result in

$$a(N_A\mathbf{e}_i, N_B)u_{jB} + b(N_A\mathbf{e}_i, \hat{N}_{\hat{A}})p_{\hat{A}} + b(N_B\mathbf{e}_i, \hat{N}_{\hat{A}})u_{iB} = b(N_A\mathbf{e}_i,\mathbf{b}) + b(N_A\mathbf{e}_i,\mathbf{t})_{\Gamma_N}.$$

3.a.2. Convergence Analysis

The code provided was modified to solve for the required domain and boundary conditions. A convergence analysis will be carried out in order to find the optimal discretization. The Q2Q1 element is chosen for this simulations due its to quadratic convergence and stability (satisfies the LBB condition).

The convergence analysis is carried out starting with a discretization of $h = 0.4$. The maximum values of velocity in the x and y direction are computed as reference. Results for the convergence analysis can be seem in Figure 3.1.



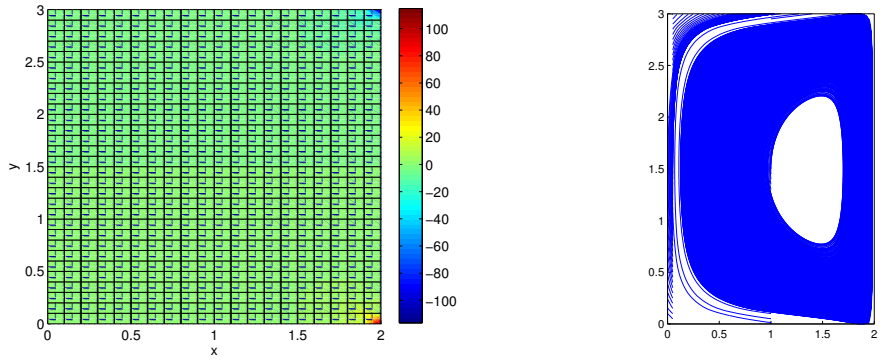
(a) Convergence of v_x (b) Convergence of v_y

Figure 3.1: Results obtained for the convergence analysis using the Q2Q1 element.

It can be seem that results converge in a discretization size of of $h = 0.1$, all the following simulations are going to be computed using this mesh size.

3.a.3. Results

The results obtained for the cavity problem using a viscosity of 1 can be observed in Figure 3.2. It can be seem that the main features of the classical cavity problem are being followed, with the centred circulating fluid and the pressure singularities on the corners.

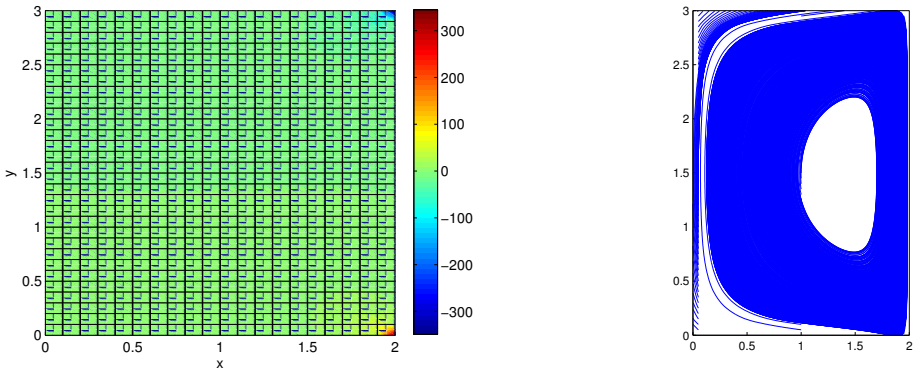


(a) Pressure iso-contours (b) Velocity streamlines

Figure 3.2: Results obtained for pressure and velocity of the cavity problem using Stokes Equations.

3.b. Navier-Stokes Problem

Now the Navier-Stokes Matlab Code is runned for Reynolds numbers of $1, 100, 1000$ and 2000 with the same parameters used in the Stokes Problem. Results can be seem in Figures 3.3, 3.4, 3.5, and 3.6.

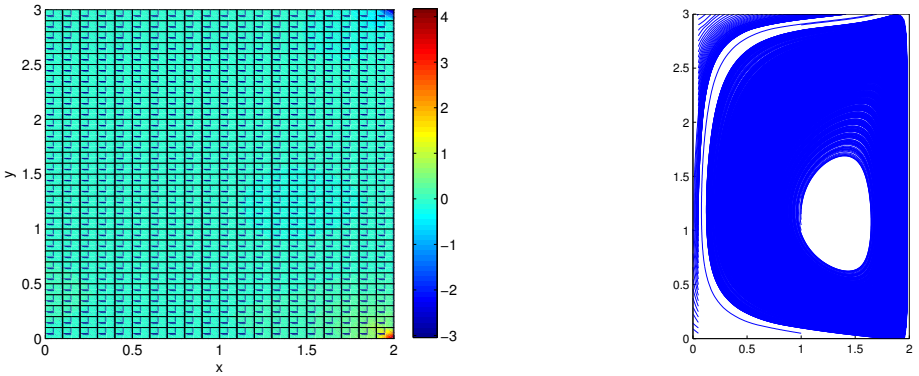


(a) Pressure iso-contours

(b) Velocity streamlines

Figure 3.3: Results obtained for pressure and velocity of the cavity problem using Navier-Stokes equations with $Re = 1$.

The results obtained in Figure 3.3 are very similar to the ones obtained in the Stokes Problem. This happens because at $Re = 1$ the flow is highly viscous and the convective term do not plays a big role. Also, this problem only took 2 newton iterations to solve the non-linearity, which also reflects the small effect of the inertial forces.



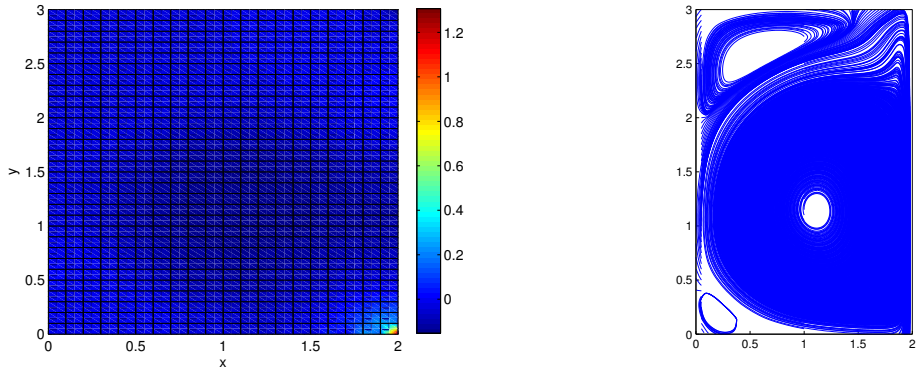
(a) Pressure iso-contours

(b) Velocity streamlines

Figure 3.4: Results obtained for pressure and velocity of the cavity problem using Navier-Stokes equations with $Re = 100$.

In Figure 3.4, more significant changes can be observed. The convective term now has a bigger significance, which is pushing the velocity stream-lines down together with the sliding plate. Also, a little vortex is likely to be created on the top left corner due to the velocity gradient present on this region. This simulation took 7 newton iterations to be solved, which is a reflects a more significant presence of the inertial forces.

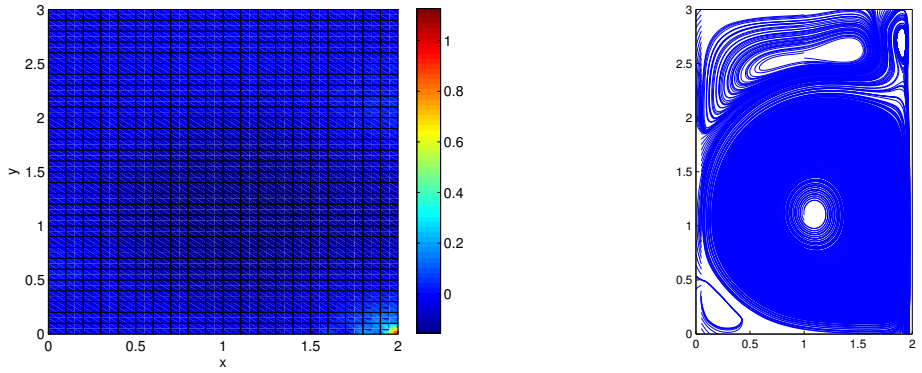
For the simulation correspondent to Figure 3.5, the convective forces are very high and at least



(a) Pressure iso-contours (b) Velocity streamlines

Figure 3.5: Results obtained for pressure and velocity of the cavity problem using Navier-Stokes equations with $Re = 1000$.

3 vortexes can be spotted in the velocity streamlines. The pressure contours are not symmetric and the singularities on the corners are diminishing. This simulation took 25 newton iterations to be solved, showing a high importance of the inertial forces.



(a) Pressure iso-contours (b) Velocity streamlines

Figure 3.6: Results obtained for pressure and velocity of the cavity problem using Navier-Stokes equations with $Re = 2000$.

Now in Figure 3.6, the maximum number of iterations was reached (99 iterations), meaning that the result is not fully converged. Still, the results do make physical sense, with more vortexes being created throughout the domain with the increase of the Reynolds number.