Numerical Solution of the Incompressible Navier-Stokes Equations

Authors:

Altadill Llasat, Miquel
Tarroc Gil, Sergi
Granados de la Torre, Adrián
Garcia-Duran Castilla, Francesc

Professor name: Soria Guerrero, Manuel

Document: Final Report

Group: 11

Delivery Date: 11/11/2019

Master's degree in Aerospace Engineering - ESEIAAT
Universitat Politècnica de Catalunya (UPC)

Contents

Acronyms

DNS Direct Numerical Solution. 6

FVM Finnite Volume Method. 5

INS Incompressible Navier Stokes. 13, 14

MMS Method of Manufactured Solutions. 6

NS Navier Stokes. 5, 6, 13

List of Figures

Chapter 1

Numerical Solution of the Incompressible Navier-Stokes Equations

1.1 Introduction

The goal of this project is to find a numerical solution of the Navier Stokes equations by the application of the Finnite Volume Method discretization and using the following hyphotesis:

- Continuity of matter
- Continuum medium assumption
- Relativity effects negligible
- Inertial reference system
- Magnetic and electromagnetic forces negligible
- Two-Dimensional model
- Laminar flow
- Incompressible flow
- Newtonian fluid
- Boussinesq hypothesis[1]
- Negligible viscous dissipation
- Negligible compression or expansion work
- Non-participating medium in radiation
- Mono-component and mono-phase fluid
- Periodic domain.
- Time integration: Adams-Bashforth explicit method.

The NS equations used are the mass conservation equation and the momentum equation (the energy equation is not needed for incompressible flow):

$$\frac{du}{dx} + \frac{dv}{dy} = 0 \tag{1.1}$$

[1] Constant physical properties everywhere except in the body forces term

$$\rho\frac{du}{dx} + \rho u\frac{du}{dx} + \rho v\frac{du}{dy} = -\frac{dp}{dx} + \mu\Big(\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2}\Big) \tag{1.2}$$

$$\rho\frac{du}{dx} + \rho u\frac{dv}{dx} + \rho v\frac{dv}{dy} = -\frac{dp}{dy} + \mu\Big(\frac{d^2v}{dx^2} + \frac{d^2v}{dy^2}\Big) + \rho g\beta(T - T_\infty) \tag{1.3}$$

As said, it is used a finite-control volume discretization to solve the equations and find the convective and the diffusive terms. A control volume must be defined to calculate the velocity, so an uniform mesh is generated. The domain of this mesh must be periodic to avoid the boundary conditions. The function used to generate the mesh halo is halo_update.m. The mesh size is characterised by the element length and the number of elements and for the nature of the discretized equations it is set to be a Staggered Mesh [1] configuration.

1.1.1 General Algorithm

All the code developed to resolve numerically the NS equations is structured in each project part from a main structure of functions. This structure consists on:

1. InputData: This function gives the input parameters needed for the development of the code.
2. PreProcess: The Preprocessor is the one that has to manage the input parameters for generating a suitable environment for the development of the solver
3. SolverShell: This function has to solve the discretized equations obtained from applying a numerical analysis to our mathematical model. It is also called solver and for this project it was implemented a DNS
4. PostProcessors This function works with the data stored for turning it into visual data for a better interpretation of the results.

This is the code main structure. Other functions are applied in each part. Those will be named and explicated in each project part.

1.2 Part A

The goals of this part are to:

- Calculate the two components of the convective term.
- Calculate the two components of the diffusive term.
- Validate the convective and diffusive terms using the Method of Manufactured Solutions.

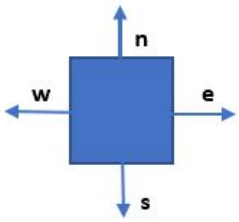


Figure 1.1: Mesh element orientation

1.2.1 Convective Term

Assuming that the velocities are known, from the advective term $\nabla \cdot (u_1 u)$ and using the conservative form for incompressible flow, integrating on each face surface, the equation becomes:

$$\int_V \nabla \cdot (u_1 u) dV = u_e F_e - u_w F_w + u_n F_n - u_s F_s \tag{1.4}$$

Where e, w, n and s are the surfaces of each control volume face and F_x is the product of the normal velocity at each face with the size of the control volume face x.

The u velocities from the transported flow at each face are obtained from the velocity field:

$$u_e = \frac{u_{i+1,j} + u_{i,j}}{2} \tag{1.5a}$$

$$u_w = \frac{u_{i-1,j} + u_{i,j}}{2} \tag{1.5b}$$

$$u_n = \frac{u_{i,j+1} + u_{i,j}}{2} \tag{1.5c}$$

$$u_s = \frac{u_{i,j-1} + u_{i,j}}{2} \tag{1.5d}$$

1.2.2 Diffusive Term

Assuming the velocities are known, from the viscous term $\nabla(\nabla \cdot (u_1) dV$ an expression for the diffusion is obtained at each face of the control volume. Integrating on each face surface, the following equation results:

$$\int_V \nabla(\nabla \cdot u_1) dV = \Delta_y \frac{\partial u_1}{\partial x}|_e - \Delta_y \frac{\partial u_1}{\partial x}|_w + \Delta_x \frac{\partial u_1}{\partial y}|_n - \Delta_x \frac{\partial u_1}{\partial y}|_s \tag{1.6}$$

Where the velocities at each face u_x are calculated from the velocity field u

$$\partial u_e = u_{i+1,j} - u_{i,j} \tag{1.7a}$$

$$\partial u_w = u_{i,j} - u_{i-,j} \tag{1.7b}$$

$$\partial u_n = u_{i,j+1} - u_{i,j} \tag{1.7c}$$

$$\partial u_s = u_{i,j} - u_{i,j-} \tag{1.7d}$$

Considering an uniform mesh, the mesh size values are the same

$$\Delta_x = \Delta_y = \partial x = \partial y = \Delta \tag{1.8}$$

And the diffusive term is hardly simplified

$$Diff_x = \Delta \frac{\partial u_1}{\Delta}|_x = \partial u_x \tag{1.9}$$

The diffusive term is obtained for each control volume. Figure 1.1 shows the mesh element orientation took into account for developing the discretisation.

1.2.3 Algorithm

This section gives a brief description of how each of the most important function works. A flowchart of the code is presented in Figure A.1.

Convective and Diffusive

The convective and the diffusive terms have been calculated with the function ConvDiff.m. This function structure consists on, for each control volume:

- Calculate the face velocities using the u velocity field. The v velocity field does not contribute on the transported velocity.
- Calculate the face forces with the following product: (Δ is the size of the control volume, which is constant.)

$$F_x = (u_N \Delta)_x \tag{1.10}$$

- Calculate the convective terms for each face. It is obtained as a result of the product of the force at each face with the face respective velocity:

$$conv = u_x * F_x \tag{1.11}$$

- Calculate the convective term of the whole control volume as the total value of the face convective values.

$$Conv(i,j) = conv_e – conv_w + conv_n – conv_s \tag{1.12}$$

- Calculate the diffusive term for each face. The diffusive term at one face is, with the previous simplification, the value of the face partial velocity.

$$Diff_x = \partial u_x \tag{1.13}$$

The partial face velocity is obtained from the u velocity field, as previously noted.

- Calculate the diffusive term of the whole control volume as the total value of the face diffusive values.

$$Diff(i,j) = diff_e – diff_w + diff_n – diff_s \tag{1.14}$$

- Finally, when all has been calculated, the application of halo_update.m to generate the periodic boundary of the mesh.

solution

A velocity field must be generated to solve analytically the convective and diffusive terms. The velocity field has to be divergence-free and has to verify the periodicity assumed as boundary conditions. In this case it is ($u = sinxcosy$). Deriving the velocity, the analytic convective term is found:

$$\int_V \nabla \cdot (u_1 u) dV = sinxcosx \tag{1.15}$$

And, calculating the Laplacian modifier for the velocity, the analytic diffusive term is found:

$$\nabla^2 \cdot u_1 = -2 \cdot sinxcosx \tag{1.16}$$

The steps and the functions the code follows consists on:

- Inputs introduction from the function InputData.m.
- Function SolverShell.m. This function calculates (are returned as outputs):
  - The numerical velocity field from the function VelocityField.m.
  - The numerical convective and diffusive terms from the function ConvDiff.m.
  - The analytic convective and diffusive terms from the function MMS.
  - The error between numerical and analytic terms.
- Function PostProcessors.m. This function prints the maximum error vs the element length and the squared element length.

1.2.4 Results and conclusions

Some error plots are presented for different mesh sizes. The mesh element length has to be a multiple of the 2*pi value, because the velocity field must assume divergence-free. As it can be observed in Fig. 1.2, there is parallelism between the Convective and Diffusive error and the squared mesh length.

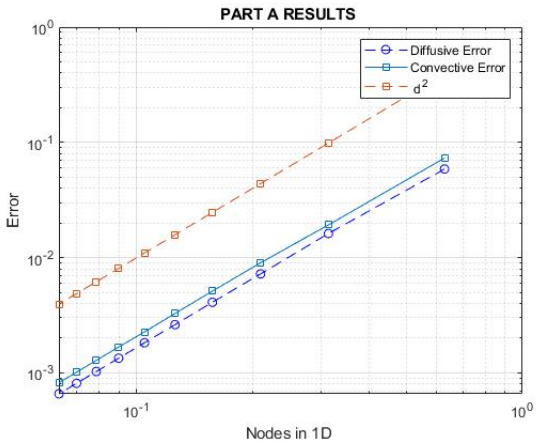


Figure 1.2: Part A results plot

1.3 Part B

1.3.1 Introduction

The goals of this part are to:

- Implement the pressure part of the algorithm
- Show that for an arbitrary field (satisfying overall mass conservation), it is possible to impose mass conservation in each control-volume with the previously described algorithm

Integrating all the terms from x-axis momentum's equation, remembering it'll be the same in y-axis changing the subindex as 2, considering V as the result of this integral of volume it's simple to arrive to:

$$\frac{du}{dt}V + C(u) = -\frac{V}{\rho}\nabla p + D(u) \tag{1.17}$$

Where C and B have been calculated in part A, u is a space-discrete velocity vector, p the space-discrete pressure (the one needed to solve) and V de volume of each control volume of the mesh Dividing all this expression between the V term:

$$\frac{du}{dt} + \frac{C}{V} = -\frac{1}{\rho}\nabla p + \frac{D}{V} \tag{1.18}$$

Working on this expression as seen in class, it comes to an expression that, in order to be solved, it must be imposed a predictor velocity. This velocity can be obtained from known velocity distributions at previous time steps, as was assumed in part A. With this new term, omitting the explanation about explicit (Adam-Bashfort) and implicit terms, the following equation is obtained:

$$u^{n+1} = u^p - \frac{\Delta t}{\rho}\nabla p^{n+1} \tag{1.19}$$

Imposing mass conservation to the left side of previous equation, or divergence-free condition:

$$\nabla u^p = \nabla^2(\frac{\Delta t}{\rho}p^{n+1}) \tag{1.20}$$

Defining an unknown pseudo-pressure, making easy manage the previous expression:

$$\tilde{p} = \frac{\Delta t}{\rho}p^{n+1} \tag{1.21}$$

Assuming the predictor velocity as known, uniform mesh and equal in both axis, after applying divergence theorem to the centered mesh discretising the derivatives as 1st order:

$$\Delta\frac{p_e - p_p}{\Delta} - \Delta\frac{p_p - p_w}{\Delta} + \Delta\frac{p_n - p_p}{\Delta} - \Delta\frac{p_p - p_s}{\Delta} = \Delta v_p - \Delta v_s + \Delta u_p - \Delta u_w \tag{1.22}$$

Simplifying both sides of this expression, defining an square, singular and symmetric A matrix of N squared components rows and columns, p as the unknown vector with the same number of components to compute and b as a known vector (predictor velocity) with N^2 components, it can be write as follows:

$$A \cdot p = b \tag{1.23}$$

1.3.2 Algorithm

The algorithm imposed to solve the equations seen before is:

a Generate a predictor velocity:

In order to create a known predictor velocity the function *rand_field* has been created. This function generates a random field. This random field will define the values of the variables u_p andv_p, forming the predictive velocity field term at expression 1.25.

b Solve the Poisson equation for the pseudo-pressure

To find the value of the pseudo-pressure as shown in 3.8, an A matrix and a b vector have to be generated.

The function *Amatrix* takes a null element matrix and impose the values needed at certain points. As seen in slides, A matrix it's full of -4 terms in it's main diagonal and, for each neighbour at north, south, east and west of the node studied, this row/column value is set to be 1. This function works defining each of the neighbours as 1 instead of 0. Furthermore this A matrix will have to be perturbed so as to be solved. In this function that can be done for example changing one of the -4 terms of the main diagonal. The reason behind that remains on the fact that a sparse matrix like that, with few non-zero elements, have its largest elements on the main diagonal, so every non null element has an immediate effect on all the domain. That's the way a non-singular matrix is generated. Before analysing the neighbours, the function makes a loop defining every position of the mesh with the variable K.

For each of the neighbours and algorithm based on if/else is imposed:

(a) Centered
Right now, this value is stablished as -4. That comes for the 4 negative pseudo-pressure terms we have when simplifying equation 1.27. It's imposed when both variables have the same value.

(b) West
A loop is done in x-axis, as the neighbours wanted are on the same row. That's for west and east nodes. In the west case, left-side value of each node is set to 1. In the special case the studied node is on the left limit, the value of the same row's right limit is stablished as 1 too.

(c) East
Right-side value of each node is set to 1. In the case the studied node is on the east limit, the value of the same row's left limit is stablished as 1 too.

(d) South
For this term and north's one, we work on the same column (j). The value, at same i,

but in the previous row is set to 1. Moreover, when the node is on the south limit, this 1 is imposed to the 1st column's term.

(e) North
The value, for the same column, but in the next row is set to 1. Additionally, when the node is on the north limit, this 1 is imposed to the last column's term.

The b vector comes from the predictive velocity generated as a random field with the function mentioned before. The function *bvector* work the expression 1.27, adding and substracting the different terms and multiplying them by the length of each control volume. In this case a matrix called B, with the same size as A, is created with a loop. Then, doing a loop again, it's converted into a vector

c Obtain the pressure

Once A matrix and b vector's algorithms have been calculated, pseudopressure can be obtained from the A matrix's inverse multiplied by b vector. That's done in the function called *pseudo_p*. With this operation a 1D vector is calculated, which is turned into a 2D field doing a loop.

Once the pseudo-pressure has been computed, looking at equation 3.5, it's easily seen that the gradient of the pseudo-pressure has to be calculated to find the 'n+1' velocity. For that, the function *grad* has been created. This function takes the values of the pseudo-pressure calculated at *pseudo_p* for 2 instants and calculate the gradient as the value's difference between nodes for the control volume's length. The same is done in both axis.

d Obtain the 'n+1' velocity

From equation 3.5, it's immediate to calculate the values for 'n+1' velocity, or *u/v_next* as called at code. For the computation are required the predictor velocity field generated by *rand_field*, and the values calculated in *grad*.

Finally, when calculate all the velocities and pseudo-pressure terms for each control volume, the application of the function haloupdate.m to generate consistency without applying directly boundary conditions. A flowchart of the code is presented in Figure B.1.

1.3.3 Results and conclusions

The last step in this part is to ensure the value calculated for the 'n+1' velocity is divergence-free. As imposed in predictor velocity and in order to satisfy overall mass conservation in each control-volume, every one of the variables has to be divergence-free.

In this case, the divergence is evaluated in the last variable computed (u/v_next), from the function div. This function takes the values of the variables mentioned before and find its divergence. The divergence is calculated as addition of both gradients from different axis.

If the procedure it's accurate, this value has to tend to 0 asymptotically, nearer to 0 as number of nodes and mesh size are risen. This value can be defined as the error committed in the computation of the 'n+1' velocity. With the max tool of Matlab, saving the maximum value of divergence, we can obtain the maximum error value of the global computation, for all the nodes of the mesh.

1.4 Part C

The goals of this part are to:

- Solve the NS equations with x and y periodicity.
- Test the code with the following u, v and p distributions:

$$\begin{bmatrix} u \\ v \end{bmatrix} = F \begin{bmatrix} cos(2\pi x)sin(2\pi y) \\ -cos(2\pi y)sin(2\pi x) \end{bmatrix} \tag{1.24}$$

$$F = e^{-8\pi^2 \upsilon t} \tag{1.25}$$

$$p = -F^2 \frac{\rho(cos^2(2\pi x) + cos^2(2\pi y))}{2} \tag{1.26}$$

1.4.1 Algorithm

For each time step the algorithm structure is similar than the Part B structure. Part C structure functions consists on:

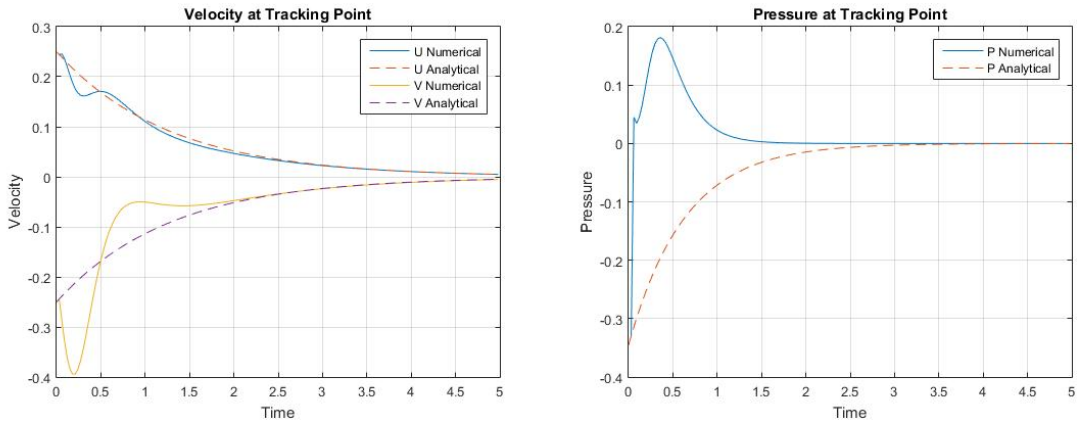
- Preprocess. This function define the initial conditions and get the solution of the analytical fields (u, v, p) at $t = 0$ and $t = 0 + \Delta t$. Those solutions are stored at analytical and numerical tracking.
- SolveINS. This function solve the INS problem and stores tracking values for each time step.
- Postprocess. This function generates two plots that compares numerical and analytical solution for pressure and velocity at tracking point.

1.4.2 Results and conclusions

The next plots represents the analytical Part C solution for the pressure and the velocity at the tracking point. Those plots are represented at each time step.

Fig. 1.3 shows that the numerical and analytic velocities start to converge at $t = [0.8 - 1.0]s$. It is clear that results are not correct because the curve does not exactly fit to the analytical result. But, instead of that, we find out that the results oscillate around the expected ones. Despite this fact, the code still follows a result with physical mean because it presents an aproximate behaviour. This error could be produced by an error inside the algorithm, which has not been yet debugged.

As seen at left side, the velocity's error at the first time steps of the code plummed till its maximum, recovering short time ago and being over the time converging stable to the final analytic solution. The explanation of that fact could remain in the initial conditions imposed. On the other side, the pressure suffers from something similar, being raised till its maximum few time steps after the beginning. As justified before, as velocity have fallen to its minimum, according to INS, the presurre might raise to its maximum.



(a) (b)

Figure 1.3: INS Results for
(a)Velocities at tracking point (b)Pressures at tracking point

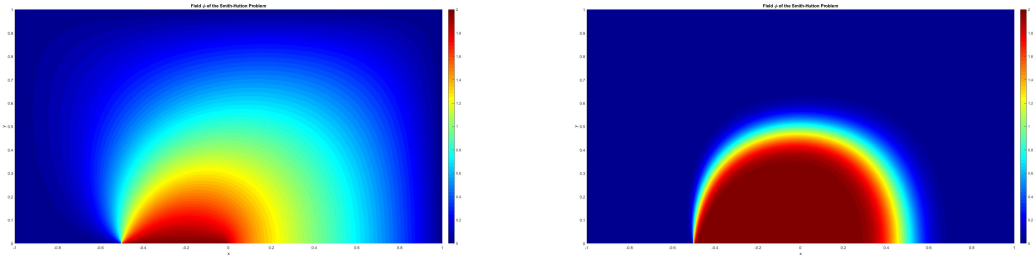
As seen at left side, the velocity's error at the first time steps of the code plummed till its maximum, recovering short time ago and being over the time converging stable to the final analytic solution. The explanation of that fact could remain in the boundary conditions imposed.

On the other side, the pressure suffers from something similar, being raised till its maximum few time steps after the beginning. As justified before, as velocity have fallen to its minimum, according to INS, the pressure might raise to its maximum.

1.5 Part D

This chapter shows the results for a two-dimensional application of the Part A that was developed during a Final Grade Thesis in order to show that the concepts applied has been fully understood. The results correspond to the convection-diffusion equation modelling for the application to the Smith-Hutton Problem[2].

The following figure shows the results obtained for a predefined ρ/Γ relation. The objective is to see the variation of the flow response over different values of this relation. As Fig. 1.4 shows, using graters values of ρ/Γ leads to more convective flow where the streamlines easily follows the solenoidal velocity field. A low value of ρ/Γ gives greater importance to the diffusive terms and so on, the flow is does not exactly match the velocity field.



(a) (b)

Figure 1.4: Field ϕ for
(a)$\rho/\Gamma = 10$ (b)$\rho/\Gamma = 1000$

[2]The document with a fully explanation of the case of study and the algorithm developed using MOO Programming is shown "here".

References

[1] S. V.Patankar, *Numerical Heat Transfer and Fluid Flow*, 1st ed., 1980.

Appendix A

Part A - Code

This appendix contains all the functions needed to solve the first part of this project. It is related to Chapter 1.2, there it is presented a flowchart of the algorithm developed and a general description of the problem and the code functions. The following sections shows each function by the order of its use in the algorithm.[1]

Flowchart

Figure A.1 show a flowchart of the structure developed for solving the Part A. Following this chart gives a brief idea about the code general structure and the function calling order. The following subsections explains the main concepts for the principal functions of this code.

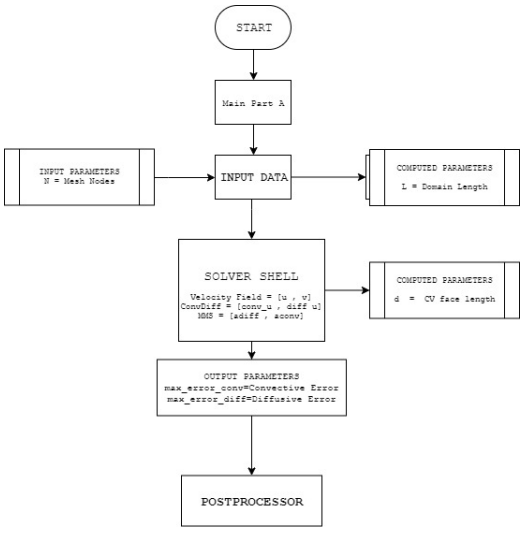


Figure A.1: Part A Flowchart

[1]This code can be downloaded by clicking "here".

A.1 Main

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  CODE A (Conv — Diff)  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Adrian Granados de la Torre  %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%  Miquel Altadill Llasat  %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%  Francesc Garcia—Duran   %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  Sergi Tarroc Gil   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ———————————————————————————  DESCRIPTION  ———————————————————————————————%
% This file contains all the functions needed to solve the case of
% study. This code section shows the function structure. You can
% find a description of how each function works inside each own file.

% The Final Report document shows the theorical development of the
% problem.

clear all
more off

InputData  %Shell parameters are defined

tic;
[max_error_diff,max_error_conv] = SolverShell(L,N);

PostProcessors (L,N,max_error_diff,max_error_conv);

toc;
fprintf('Solver Time %f\n',toc);
```

A.2 Input Data

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  INPUT DATA  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ———————————————————————————  DESCRIPTION  ———————————————————————————————%
% This file contains all the parameters needed to make the
% code run. Each parameter needed should be loaded into this
% file.
%
% ———————————————————  INPUT PARAMETERS  ———————————————————————————————%
% N  =  Number of Mesh nodes
%
% ——————————————————  COMPUTED PARAMETERS  ———————————————————————————————%
% L  =  Domai Length
```

```
14 %
15
16 N=[10:10:1e2];
17
18 L=2*pi;
```

A.3 SolverShell

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  SOLVER SHELL  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %
5  % ————————————————————————  DESCRIPTION  ————————————————————————————%
6  % This function iterates the solver over the different mesh
7  % sizes specifyed in the 'InputData.m'  file.
8  %
9  % ——————————————————  INPUT PARAMETERS  ——————————————————————————%
10 % N  =  Number of Mesh nodes
11 % L  =  Domai Length
12 %
13 % ——————————————————  OUTPUT PARAMETERS  —————————————————————————%
14 % max_error_conv  = Convective error
15 % max_error_diff  = Diffusive error
16 %
17 % ——————————————————  COMPUTE PARAMETERS  ————————————————————————%
18 % d  =  CV face length
19 %
20 % ————————————————————————  DATA STORED  ————————————————————————————%
21 %
22 % max_error_conv  = Saves  Convective error for postprocessing
23 % max_error_diff  = Saves  Diffusive error for postprocessing
24
25
26 function [max_error_diff, max_error_conv] = SolverShell(L,N)
27
28 max_error_diff=zeros(length(N),1);
29 max_error_conv=zeros(length(N),1);
30
31 for k=1:length(N)
32
33     d=L/N(k);
34
35     % —— NUMERICAL VELOCITY FIELD —— %
36     [u,v]  = VelocityField(N(k),L);
37
38     % —— NUMERICAL CONVECTIVE—DIFFUSIVE TERM —— %
39
```

```
40      [conv_u,diff_u] = ConvDiff (u,v,L);
41
42      % -- NUMERICAL CONVECTIVE TERM -- %
43      %conv_u = convective(u,v,L)./d^2;
44
45      % -- ANALYTIC CONV and DIFF THERMS -- %
46      [adiff, aconv]=MMS(N(k),L);
47
48      % -- ERROR COMPUTE -- %
49      conv_u = conv_u./d^2;
50      diff_u = diff_u./d^2;
51
52      error1  =  abs(diff_u-adiff);
53      max_error_diff(k)  =  max(max(error1));
54
55      % -- ERROR COMPUTE -- %
56      error2=abs(conv_u-aconv);
57      max_error_conv(k)=max(max(error2));
58
59  end
```

A.4 Convective - Diffusive

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  CONV-DIFF    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %
5  % ------------------------  DESCRIPTION  ------------------------------%
6  % This function computes the convective and diffusive term t
7  % values at each grid poin and saves it into the diff matrix.
8  %
9  % ---------------------  INPUT PARAMETERS  -------------------------%
10 % u  =  Velocity in X [m/s]
11 % v  =  Velocity in Y [m/s]
12 % L  =  Domai Length
13 %
14 % ---------------------  OUTPUT PARAMETERS  ------------------------%
15 % conv = convective term matrix
16 % diff = diffusive term matrix
17 %
18 % -------------------  COMPUTE PARAMETERS  -------------------------%
19 % d  =  CV face length
20 % N  =  Number of Mesh nodes
21 % u  =  Velocity in X [m/s]
22 %
23 % ------------------------  DATA STORED  ------------------------------%
24 %
```

```
25 % conv = stores convective term matrix [N+2][N+2]
26 % diff = stores diffusive term matrix [N+2][N+2]
27
28 function [conv,diff] = ConvDiff (u,v,L)
29
30     N = size(v,1)-2;
31
32     d = L/N;
33
34     conv = zeros(N+2,N+2);
35     diff=zeros(N+2,N+2);
36
37     for i=2:N+1
38         for j=2:N+1
39
40             %CONVECTIVE TERM
41             u_e = (u(i+1,j)+u(i,j))/2;
42             F_e = d*u_e;
43             conv_e = u_e*F_e;
44
45             u_w = (u(i-1,j)+u(i,j))/2;
46             F_w = d*u_w;
47             conv_w = u_w*F_w;
48
49             u_n = (u(i,j+1)+u(i,j))/2;
50             F_n = (d/2)*(v(i,j)+v(i+1,j));
51             conv_n = u_n*F_n;
52
53             u_s = (u(i,j-1)+u(i,j))/2;
54             F_s = (d/2)*(v(i,j-1)+v(i+1,j-1));
55             conv_s = u_s*F_s;
56
57             conv(i,j) = conv_e - conv_w + conv_n - conv_s;
58
59             %DIFFUSIVE TERM
60             diff_e=u(i+1,j)-u(i,j);
61             diff_w=u(i,j)-u(i-1,j);
62             diff_n=u(i,j+1)-u(i,j);
63             diff_s=u(i,j)-u(i,j-1);
64             diff(i,j)= diff_e -diff_w +diff_n -diff_s;
65
66         end
67     end
68
69     conv = halo_update (conv);
70     diff=halo_update(diff);
71
72 end
```

A.5 Mesh

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  MESH  %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ———————————————————— DESCRIPTION ————————————————————%
% This function generates the mesh geometry. The mesh
% choosen for the problem is uniform. For furure versions
% it could be improved to non—uniform meshes.
%

% ———————————————————— INPUT PARAMETERS ————————————————————%
% P = Initial Point
% O = Final Point

% ———————————————————— OUTPUT PARAMETERS ————————————————————%
% p = Point vector

function [p] = Mesh (P,O,dx)



    p = P : dx : O;

end
```

A.6 Velocity Field

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%  SOLVER SHELL  %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ————————————————————— DESCRIPTION ——————————————————————%
% This function iterates the solver over the different mesh
% sizes specifyed in the 'InputData.m'  file.
% ————————————————————— DESCRIPTION ——————————————————————%
% This function computes the velocity field for the case of
% study. It is defined inside the code.
%
% ———————————————————— INPUT PARAMETERS ————————————————————%
% N  =  Number of Mesh nodes
%
% ———————————————————— OUTPUT PARAMETERS ————————————————————%
% u  = Velocity field [m/s]
% L  =  Domai Length
```

```
18 %
19 % ------------------------- DATA STORED -------------------------%
20 % u = Data for the velocity field in a [N+2][N+2] matrix
21
22
23
24
25 function [u,v] = VelocityField (N,L)
26
27     d = L/N;
28
29     u = zeros(N+2, N+2);
30
31
32     x = Mesh (0 , L+d , d);
33     y = Mesh (-d/2, L+d/2, d);
34
35     for i=2:1:N+1
36         for j=2:1:N+1
37             u(i,j)=sin(x(i))*cos(y(j));
38         end
39     end
40
41     u=halo_update(u);
42
43     d = L/N;
44
45     v = zeros(N+2,N+2);
46
47     x = Mesh (-d/2 , L+d/2 , d);
48     y = Mesh (0, L+d, d);
49
50    for i=2:1:N+1
51         for j=2:1:N+1
52             v(i,j) = -cos(x(i))*sin(y(j));
53         end
54     end
55
56     v = halo_update(v);
57
58
59 end
```

A.7 Method of Manufactured Solutions

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%  MMS  %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----------------- DESCRIPTION -------------------%
% This functions computes the analytic result in order
% to use the Method of the Manufactured Solutions to
% check the validity of the code.
%
% ----------------- INPUT PARAMETERS --------------------%
% N  =  Number of Mesh nodes
% L  =  Domai Length
%
% ----------------- OUTPUT PARAMETERS -------------------%
% adiff  = Analitic Diffusive solution
% aconv  = Analitic Convective Solution
%
% ---------------------- DATA STORED ----------------------%
%
% adiff  = Analitic Diffusive solution matrix [N+2][N+2]
% aconv  = Analitic Convective Solution matrix [N+2][N+2]

function [adiff,aconv] = MMS(N,L)

    d = L/N;
    aconv = zeros(N+2,N+2);


    x = Mesh (0 , L+d , d);
    y = Mesh (-d/2, L+d/2, d);

     for i=2:1:N+1

        for j=2:1:N+1

            aconv(i,j)=sin(x(i))*cos(x(i));

        end
     end

     aconv = halo_update (aconv);

     adiff=zeros(N+2,N+2);


    for i=2:1:N+1

```

```
47          for j=2:1:N+1
48
49              adiff(i,j)=-2*sin(x(i))*cos(y(j));
50          end
51      end
52
53      adiff=halo_update(adiff);
54
55
56  end
```

A.8 Halo Update

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  HALO UPDATE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %
5  % ------------------------  DESCRIPTION  ------------------------------%
6  % This function updates the halo values of our case of study.
7  %
8  % ---------------------  INPUT PARAMETERS  ------------------------------%
9  % f  =  Parameter to update
10 %
11 % ---------------------  OUTPUT PARAMETERS  -----------------------------%
12 % F  =  Updated Parameter
13
14
15 function [F] = halo_update(F)
16
17     N = size(F, 1) - 2;
18     F(1, :) = F(N+1, :);
19     F(N+2, :) = F(2,:);
20     F(:, 1) = F(:, N+1);
21     F(:, N+2) = F(:, 2);
22
23 end
```

Appendix B

Part B - Code

This appendix contains all the functions needed to solve the first part of this project. It is related to Chapter 1.3, there it is presented a flowchart of the algorithm developed and a general description of the problem and the code functions. The following sections shows the new functions developed for solving the Part B.[1]

Flowchart

Figure B.1 show a flowchart of the structure developed for solving the Part A. Following this chart gives a brief idea about the general structure of the code and the function order. The following sections explains the main concepts for the principal functions of this code.

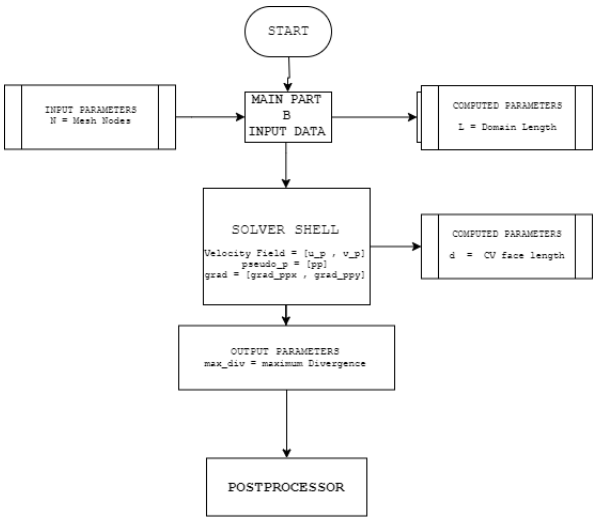


Figure B.1: Part B Flowchart

[1]This code can be downloaded by clicking "here".

B.1 Main

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%  CODE B (Pressure – Velocity)  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Adrian Granados de la Torre  %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%  Miquel Altadill Llasat  %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%  Francesc Garcia–Duran   %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%  Sergi Tarroc Gil   %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ———————————————————————  DESCRIPTION  ———————————————————————————%
% This file contains all the functions needed to solve the case of
% study. This code section shows the function structure. You can
% find a description of how each function works inside each own file.
%
% The Final Report document shows the theorical development of the
% problem.


clear all
more off

InputData

tic;
[u_next, v_next] = SolverShell(L,N);

PostProcessors (L,u_next,v_next);

toc;
fprintf('Solver Time %f\n',toc);
```

B.2 Input Data

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  INPUT DATA  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ———————————————————————  DESCRIPTION  ———————————————————————————%
% This file contains all the parameters needed to make the
% code run. Each parameter needed should be loaded into this
% file.
%
% ———————————————————  INPUT PARAMETERS  ——————————————————————————%
% N  =  Number of Mesh nodes
%
% ——————————————————  COMPUTED PARAMETERS  ——————————————————————————%
```

```
% L  =  Domai Length
%

N = 50;


L = 2*pi;
```

B.3 SolverShell

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  SOLVER SHELL  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ---------------------------  DESCRIPTION  ------------------------------%
% This function iterates the solver over the different mesh
% sizes specifyed in the 'InputData.m'  file.
%
% ---------------------  INPUT PARAMETERS  -------------------------------%
% N  =  Number of Mesh nodes
% L  =  Domai Length
%
% ---------------------  OUTPUT PARAMETERS  ------------------------------%
% u_next  =
% v_next  =
%
% -------------------  COMPUTE PARAMETERS  -------------------------------%
% d  =  CV face length
% L  =  Domai Length
%
% ---------------------------  DATA STORED  ------------------------------%
%
% max_error_conv  = Saves  Convective error for postprocessing
% max_error_diff  = Saves  Diffusive error for postprocessing


function [u_next, v_next] = SolverShell(L,N)

%----- Generate random predictive velocity field -----

% -- NUMERICAL VELOCITY FIELD -- %

u_p = VelocityField (N);
v_p = VelocityField (N);

%----- Solve predictive pressure -----
pp = pseudo_p(u_p,v_p,L);
```

```
38
39 %----- Solve u_n+1 for calculated pressure ---
40 [grad_ppx,grad_ppy] = grad(pp,L);
41
42 u_next = u_p-grad_ppx;
43 v_next = v_p-grad_ppy;
44
45 end
```

B.4 Velocity Field

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%  VELOCITY FIELD  %%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %
5  % ------------------------  DESCRIPTION  -------------------------------%
6  % This function computes the velocity field for the case of
7  % study. It is defined inside the code.
8  %
9  % ---------------------  INPUT PARAMETERS  -------------------------%
10 % N  =  Number of Mesh nodes
11 %
12 % ---------------------  OUTPUT PARAMETERS  ------------------------%
13 % u  = Velocity field [m/s]
14 %
15 %
16 % ------------------------  DATA STORED  -------------------------------%
17 % u = Data for the velocity field in a [N+2][N+2] matrix
18
19
20
21
22 function [u] = VelocityField (N)
23
24     u = zeros (N+2,N+2);
25
26     u (2:N+1,2:N+1) = 2*rand(N)-1; %generate a random field [-1,1]
27
28     u = halo_update (u);
29
30
31 end
```

B.5 Pseudo Pressure

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%  PSEUDO PRESSURE  %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ---------------------------  DESCRIPTION  ---------------------------------%
% This function computes Pseudo-Pressure
%
% ----------------------  INPUT PARAMETERS  ---------------------------------%
% u_p = Velocty X direction [m/s]
% v_p = Velocty Y direction [m/s]
% L  =  Domai Length [m]
% ----------------------  OUTPUT PARAMETERS  --------------------------------%
% max_error_conv  = Convective error
% max_error_diff  = Diffusive error
%
% -------------------  COMPUTE PARAMETERS  ----------------------------------%
% d  =  CV face length
%
% ---------------------------  DATA STORED  ---------------------------------%
% A = Coefficient Matrix


function [pp] = pseudo_p(u_p,v_p,L)

    N=size(u_p,1)-2;    % Number of nodes in a array
    d=L/N;              % Volume length

    % Coeficient matrix. A*pp_vect is the volume integral of the ...
       laplacian of
    % pseudo preassure
    A = Amatrix(N);

    % b is the volume integral of the divergence of de predictive velocity
    b = bvector(u_p,v_p,L);

    % Solve the equations system A*pp_vect=b
    pp_vect=A\b;

    %Transmform the 1D pp_vect to the 2D pp field form
    pp=zeros(N+2);

    for k=1:N^2
        j = 1+fix((k-1)/N)+1;
        i = 1+k-(j-2)*N;
        pp(i,j) = pp_vect(k);
    end

    pp=halo_update(pp);
```

```
49
50 end
```

B.6 Gradient

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  GRADIENT FUNCTION  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %
5  % ————————————————————————  DESCRIPTION  ————————————————————————————%
6  % This function calculate de dradient of a field pp in a
7  % presure centered mesh.
8  %
9  % ———————————————————  INPUT PARAMETERS  ————————————————————————%
10 % N  =  Number of Mesh nodes
11 % L  =  Domai Length
12 %
13 % ———————————————————  OUTPUT PARAMETERS  ———————————————————————%
14 % grad_ppx  = dpp/dx Pressure Gradient in X
15 % grad_ppy  = dpp/dy Pressure Gradient in Y
16 %
17 % ——————————————————  COMPUTE PARAMETERS  ———————————————————————%
18 % d  =  CV face length
19 % L  =  Domai Length
20 %
21 % ————————————————————————  DATA STORED  ————————————————————————————%
22 %
23 % grad_ppx  = Saves  X Gradient in a [N+2] vector
24 % grad_ppy  = Saves  Y Gradient in a [N+2] vector
25
26
27 function [grad_ppx,grad_ppy]=grad(pp,L)
28
29     N=size(pp,1)-2;
30     d=L/N;
31
32     grad_ppx=zeros(N+2);
33     grad_ppy=zeros(N+2);
34
35     for i=2:N+1
36         for j=2:N+1
37             grad_ppx(i,j)=(pp(i+1,j)-pp(i,j))/d;
38             grad_ppy(i,j)=(pp(i,j+1)-pp(i,j))/d;
39         end
40     end
41
42     grad_ppx=halo_update(grad_ppx);
```

```
43      grad_ppy=halo_update(grad_ppy);
44
45  end
```

B.7 Divergence

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %
5  % ————————————————————————  DESCRIPTION  ————————————————————————————%
6  % Calculate the divergence of a velocity field [u,v]
7  %
8  % ———————————————————  INPUT PARAMETERS  ———————————————————————————%
9  % u = Velocty X direction [m/s]
10 % v = Velocty Y direction [m/s]
11 % L  =  Domai Length [m]
12 %
13 % ———————————————————  OUTPUT PARAMETERS  ——————————————————————————%
14 % divergence = Velocity divergence value
15 %
16 % ———————————————————  COMPUTE PARAMETERS  —————————————————————————%
17 % d  =  CV face length
18 % N  =  Number of Mesh nodes
19 %
20 %
21 % ————————————————————————  DATA STORED  ————————————————————————————%
22 % divergene = Saves Divergence values in a [N+2] vector
23
24
25 function [divergence]=div(u,v,L)
26
27      N=size(u,1)-2;
28      d=L/N;
29
30      divergence=zeros(N+2);
31
32      for i=2:N+1
33          for j=2:N+1
34              divergence(i,j)=(u(i,j)-u(i-1,j))/d + (v(i,j)-v(i,j-1))/d;
35          end
36      end
37
38      divergence=halo_update(divergence);
39
40  end
```

B.8 Postprocessors

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  POSTPROCESOR %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ------------------------- DESCRIPTION ------------------------------%
% This function works with the data stored for turning it into
% visual data. (Exampe: Plots)
%

function PostProcessors (L, p1, p2)

    %----- Verify u_n+1 is divergence free -----
    div_next = div(p1,p2,L);

    max_div = max(max(abs(div_next)));

    fprintf('Maximum Divergence =  %e\n ',max_div);


end
```

B.9 Matrix 'A'

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%   A MATRIX     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ------------------------- DESCRIPTION ------------------------------%
% This function creates the A matrix with its corresponding
% coeficients.
%
% ---------------------- INPUT PARAMETERS ----------------------------%
% N  =  Number of Mesh nodes
%
% ---------------------- OUTPUT PARAMETERS ---------------------------%
% A  = Coefficient Matric
%
% --------------------- COMPUTE PARAMETERS ---------------------------%
% d  =  CV face length
% L  =  Domai Length
%
% ---------------------------- DATA STORED ---------------------------%
%
```

```
21 % A = Saves  Coefficient matrix in a [N^2][N^2] matrix
22
23
24
25 function [A] = Amatrix (N)
26
27     A = zeros (N^2,N^2);
28
29     K=zeros(N,N);
30     for i=1:N
31         for j=1:N
32             K(i,j)=i+(j-1)*N;
33         end
34     end
35
36     for k=1:N^2
37
38         j=fix((k-1)/N)+1;
39         i=k-(j-1)*N;
40
41         %Centered
42         A(k,k)=-4;
43
44         %West
45         if i==1 A(k,K(N,j))=1;
46         else A(k,k-1)=1;
47         end
48
49         %Est
50         if i==N A(k,K(1,j))=1;
51         else A(k,k+1)=1;
52         end
53
54         %South
55         if j==1 A(k,K(i,N))=1;
56         else A(k,k-N)=1;
57         end
58
59         %North
60         if j==N A(k,K(i,1))=1;
61         else A(k,k+N)=1;
62         end
63
64         A(5,5)=-4.1; % for avoiding singularity of A matrix
65
66     end
```

B.10 Vector 'b'

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ---------------------------  DESCRIPTION  ------------------------------%
% This function creates the b vector using the discretized
% equation corresponding values.
%
% ----------------------  INPUT PARAMETERS  ------------------------------%
%
%
% ----------------------  OUTPUT PARAMETERS  -----------------------------%
% u_p = Velocty X direction [m/s]
% v_p = Velocty Y direction [m/s]
% L  =  Domai Length
%

% ---------------------------  DATA STORED  ------------------------------%
%
% B = saves  matrix B [N+2][N+2]
% b = saves  vector B [N^2]


function [b] = bvector(u_p,v_p,L)

    N=size(u_p,1)-2;

    B = zeros (N+2,N+2);
    b = zeros (N^2,1);
    d = L/N;

    for i=2:N+1
        for j=2:N+1
            B(i,j) = (u_p (i,j) - u_p(i-1,j) + v_p(i,j) - v_p(i,j-1))*d;
        end
    end


   for j=2:N+1
       for i=2:N+1
       b((i-1)+(j-2)*(N)) = B(i,j);
       end
   end

end
```

Appendix C

Part C - Code

This appendix contains all the functions needed to solve the first part of this project. It is related to Chapter 1.4, there it is presented a flowchart of the algorithm developed and a general description of the problem and the code functions. The following sections shows the new functions developed for solving the Part C.[1]

Flowchart

Figure C.1 show a flowchart of the structure developed for solving the Part A. Following this chart gives a brief idea about the general structure of the code and the function order. The following sections explains the main concepts for the principal functions of this code.

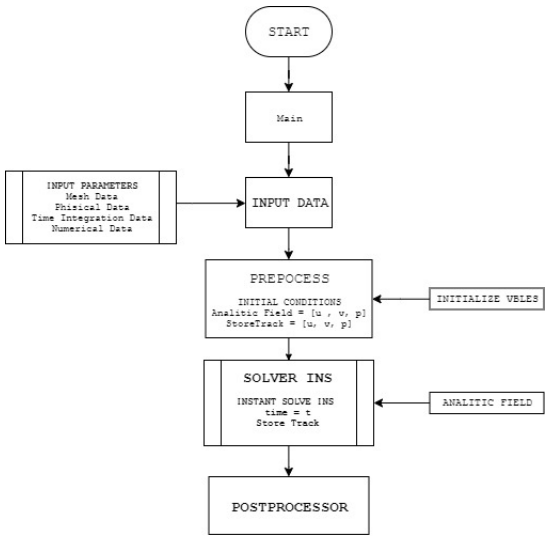


Figure C.1: Part C Flowchart

[1]This code can be downloaded by clicking "here".

C.1 Main

```
close all
clear all
clc

InputData

PreProcess

SolveINS

PostProcess
```

C.2 Input Data

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  INPUT DATA  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ------------------------- DESCRIPTION -------------------------%
% This file contains all the parameters needed to make the
% code run. Each parameter needed should be loaded into this
% file.
%
%% -------------------- MESH DATA ----------------------------------

N = 10;      % Number of elements per side
L = 1;       % Total mesh length
d = L/N;     % Element size

%% -------------------- PHYSICAL DATA ------------------------------

Re = 100;        % Reynolds number
u0 = 1;          % Reference velocity
nu = u0*L/Re;    % Cinematic viscosity
rho = 1;         % Density

%% -------------------- TIME INTEGRATION DATA ----------------------

T  = 5;                  % Total integration time
At = timestep(d,u0,Re); % Calculate required time step
t  = 0 : At : T;         % Time vector
S  = length(t);          % Number of time steps

%% -------------------- NUMERICAL DATA -----------------------
```

```
30
31 i_ref = 3;  % x_index for timetracking element
32 j_ref = 3;  % y_index for timetracking element
33
34 T_sample = 0.1; % Time sample for printing in command window
35
36 epsilon = 1e-3; % maximum error (only required if INPLICIT scheme is used)
37 lambda  = 0.05; % relax factor (only required if INPLICIT scheme is used)
38
39 %% ------------------- COMMAND WINDOW PRINT --------------------
40
41 fprintf('   Time step: %.3f \n',At);
42 fprintf('   Element size: %.3f \n',d);
43 fprintf('   Reynolds number: %.0f \n',Re);
44 fprintf('   Element (%.0f,%.0f) Velocity and Pressure will be tracked ...
       \n',i_ref,j_ref);
```

C.3 Mesh

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  MESH  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  % ---------------------  DESCRIPTION  ------------------------%
5  % This function generates the mesh geometry. The mesh
6  % choosen for the problem is uniform. For furure versions
7  % it could be improved to non-uniform meshes.
8  %
9  % ---------------------  INPUT PARAMETERS  ---------------------%
10 % P = Initial Point
11 % O = Final Point
12
13
14 function [p] = Mesh (P,O,dx)
15
16       p = P : dx : O;
17
18 end
```

C.4 Preprocess

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  PreProcess  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
4  % ------------------------  DESCRIPTION  --------------------------%
5  % This file prepares all the information required for solving
6  % the problem in the appropiate way for the solver
7  %
8
9  %% ---------------------- INITIALIZE VARIABLES ---------------------
10
11 % Temporal tracking numerical results for (i_ref,j_ref)
12 u_nt = zeros(S,1);  % X-Velocity numerical tracking
13 v_nt = zeros(S,1);  % Y-Velocity numerical tracking
14 p_nt = zeros(S,1);  % Pressure numerical tracking
15
16 % Temporal tracking analytic results for (i_ref,j_ref)
17 u_at = zeros(S,1);  % X-Velocity analytical tracking
18 v_at = zeros(S,1);  % Y-Velocity analytical tracking
19 p_at = zeros(S,1);  % Pressure analytical tracking
20
21 % Cumulative simulation time for printing is inizialized
22 tprint = T_sample;
23
24 %% ---------------------- INITIAL CONDITIONS ---------------------
25 % The analytical solution velocity fields at t=0 and t=0+At
26 % will be defined as initial conditions
27
28 % Notation
29 % *_1 -> *_n-1 (previous time step)
30 % *_2 -> *_n   (current time step)
31 % *_3 -> *_n+1 (following time step)
32
33
34 % Analytical fields are calculated at t=0...
35 [ u_1 , v_1 , p_1 ] = AnalyticField (N , L , t(1) , nu , rho);
36
37 % ... and stored in analytical and numerical tracking for t=0
38 [u_nt(1), v_nt(1), p_nt(1)] = StoreTrack ( u_1 , v_1 , p_1 , i_ref , ...
       j_ref );
39 [u_at(1), v_at(1), p_at(1)] = StoreTrack ( u_1 , v_1 , p_1 , i_ref , ...
       j_ref );
40
41
42 % Analytical fields are calculated at t=0+At...
43 [ u_2 , v_2 , p_2 ] = AnalyticField (N , L , t(2) , nu , rho);
44
45 % ... and stored in analytical and numerical tracking for t=0+At
46 [u_nt(2), v_nt(2), p_nt(2)] = StoreTrack ( u_2 , v_2 , p_2 , i_ref , ...
       j_ref );
47 [u_at(2), v_at(2), p_at(2)] = StoreTrack ( u_2 , v_2 , p_2 , i_ref , ...
       j_ref );
```

C.5 Solver INS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%  SolveINS  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ------------------------- DESCRIPTION -------------------------%
% This file solves de INS problem ans stores tracking values
% for the indicated control volume
%
% Notation
% *_1 -> *_n-1 (previous time step)
% *_2 -> *_n   (current time step)
% *_3 -> *_n+1 (following time step)

fprintf('\n ----------STARTING SIMULATION---------- \n');

tic;

for k=3:S

    % Analytical solution for t(k) is calculated...
    [ u_3 , v_3 , p_3 ] = AnalyticField (N , L , t(k) , nu , rho);

    % ... and stored in analytical solution tracking array for t(k)
    [u_at(k), v_at(k), p_at(k)] = StoreTrack ( u_3 , v_3 , p_3 , i_ref , ...
        j_ref );


    % Analytical solution for t(k) is calculated...
    [ u_3 , v_3 , p_3 ] = InstantSolveINS ( u_1, v_1, u_2, v_2, L, At, ...
        nu, rho, epsilon,lambda);

    % ... and stored in numerical solution tracking array for t(k)
    [u_nt(k), v_nt(k), p_nt(k)] = StoreTrack ( u_3 , v_3 , p_3 , i_ref , ...
        j_ref );

    % Window change
    u_1 = u_2; v_1 = v_2; p_1 = p_2;
    u_2 = u_3; v_2 = v_3; p_2 = p_3;


    if t(k) > tprint
        fprintf('  %.2f [s] simulated from %.3f [s] of total simulation ...
            \n',tprint,T);
        tprint = tprint + T_sample;
    end


```

```
end

fprintf('————————SIMULATION FINISHED———————— \n');

fprintf('   Total computation time: %.3f [s] \n',toc);
```

C.6 Instant Solve INS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%   InstantSolveINS     %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ————————————————————————  DESCRIPTION  ————————————————————————————%
% This function calculates vecocity and pressure fields at
% following (n+1) time instant from current (n) and previous (n—1)
% time instant fields.
%
% The numerical solution is computed with a EXPLICIT scheme,
% but a IMPLICIT scheme can be also used
%
% Notation
% *_1 —> *_n—1 (previous time step)
% *_2 —> *_n   (current time step)
% *_3 —> *_n+1 (following time step)
%
% ——————————————————  INPUT PARAMETERS  ————————————————————————%
% {u_1,v_1} = Velocity field at previous (n—1) instant
% {u_2,v_2} = Velocity field at current (n) instant
% L   =  Domain Length
% At  =  Time increment
% nu  =  Cinematic viscosity
% rho =  Density
% epsilon = Maximum error (only required if IMPLICIT scheme is used)
% lambda  = Relax factor (only required if IMPLICIT scheme is used)
%
% ——————————————————  OUTPUT PARAMETERS  ————————————————————————%
% {u_3,v_3,p_3} = Velocity and Pressure fields at following (n+1) instant
%
% ——————————————————  COMPUTE PARAMETERS  ————————————————————————%
%
% [Cu_1,Du_1] = CONVECTIVE and DIFFUSIVE term in x—momentum equation at n—1
% [Cu_2,Du_2] = CONVECTIVE and DIFFUSIVE term in x—momentum equation at n
% [Cv_1,Dv_1] = CONVECTIVE and DIFFUSIVE term in y—momentum equation at n—1
% [Cv_2,Dv_2] = CONVECTIVE and DIFFUSIVE term in y—momentum equation at n
%
% Ru_1 = Factor funcion of CONVECTIVE and DIFFUSIVE terms for u_1
% Ru_2 = Factor funcion of CONVECTIVE and DIFFUSIVE terms for u_2
```

```
40 % Rv_1 = Factor funcion of CONVECTIVE and DIFFUSIVE terms for v_1
41 % Rv_2 = Factor funcion of CONVECTIVE and DIFFUSIVE terms for v_2
42 %
43 % u_p = Velocty X direction [m/s]
44 % v_p = Velocty Y direction [m/s]
45 %
46 % pp  = Pseudo-Pressure field at n+1
47 % {grad_ppx,grad_ppy} = Pseudo-Pressure gradient field at n+1
48
49
50
51 function  [ u_3 , v_3 , p_3 ]= InstantSolveINS( u_1, v_1, u_2, v_2, L, ...
       At, nu, rho, epsilon, lambda)
52
53 N=size(u_1,1)-2;
54 d=L/N;
55
56 [Cu_1,Du_1] = ConvDiff(u_1,v_1,L);
57 [Cu_2,Du_2] = ConvDiff(u_2,v_2,L);
58
59 [Cv_1,Dv_1] = ConvDiff(v_1,u_1,L);
60 [Cv_2,Dv_2] = ConvDiff(v_2,u_2,L);
61
62 Ru_1 = At/d^2*( -Cu_1 + nu*Du_1 );
63 Ru_2 = At/d^2*( -Cu_2 + nu*Du_2 );
64
65 Rv_1 = At/d^2*( -Cv_1 + nu*Dv_1);
66 Rv_2 = At/d^2*( -Cv_2 + nu*Dv_2);
67
68 up = u_2 + 3/2*Ru_2 - 1/2*Ru_1;
69 vp = v_2 + 3/2*Rv_2 - 1/2*Rv_1;
70
71 pp=pseudo_p(up,vp,L);
72
73 [grad_ppx,grad_ppy]=grad(pp,L);
74
75 u_3 = up-grad_ppx;
76 v_3 = vp-grad_ppy;
77
78 % %---------------- IMLPICIT SCHEME SOLVER----------------------
79 % i=1;
80 % e=epsilon+1;
81 %
82 % while e > epsilon
83 %
84 %     Ru_3=At/d^2*(-convective(u_3,v_3,L)+nu*diffusive(u_3));
85 %     Rv_3=At/d^2*(-convective(v_3,u_3,L)+nu*diffusive(v_3));
86 %
87 %     up= u_3 + Ru_3;
88 %     vp= v_3 + Rv_3;
```

```
%
%     pp=pseudo_p(up,vp,L);
%     [grad_ppx,grad_ppy]=grad(pp,L);
%
%     u_3aux = u_2 + 3/2*Ru_2 - 1/2*Ru_1 - grad_ppx;
%     v_3aux = v_2 + 3/2*Rv_2 - 1/2*Rv_1 - grad_ppy;
%
%     error_u=abs(u_3aux-u_3);
%     error_v=abs(v_3aux-v_3);
%
%     e=max(max(max(error_u,error_v)));
%
%     u_3 = u_3+ lambda*(u_3aux-u_3);
%     v_3 = v_3+ lambda*(v_3aux-v_3);
%
%     i=i+1;
%     error(i)=e;
%
% end
% %figure; plot(error,'s');

p_3 = pp*rho/At;

end
```

C.7 Analytic Field

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%  AnalyticField  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ------------------------  DESCRIPTION  ------------------------------%
% This function calculates the analytic solution of the velocity
% and pressure field for all domain at a given time instant
%
% ---------------------  INPUT PARAMETERS  ----------------------------%
% N   =  Number of Mesh nodes
% L   =  Domain length
% t   =  Time instant
% nu  =  Cinematic viscosity
% rho =  Density
%
% ---------------------  OUTPUT PARAMETERS  ---------------------------%
% u  = X-Velocity field
% v  = Y-Velocity field
% p  = Pressure field
%
```

```
21 % ------------------  COMPUTE PARAMETERS  ------------------------%
22 % d  =  CV face length
23 % x  =  x position array
24 % y  =  y position array
25 % F  =  Time depended factor
26 %
27
28
29 function [ u , v , p ] = AnalyticField ( N , L , t , nu , rho )
30
31     d = L/N;
32     u = zeros(N+2, N+2);
33
34     % ------------ u field ----------------------------
35
36     x = Mesh (0 , L+d , d);
37     y = Mesh (-d/2, L+d/2, d);
38
39     F=exp(-8*pi^2*nu*t);
40
41     for i=2:1:N+1
42         for j=2:1:N+1
43             u(i,j)=cos(2*pi*x(i))*sin(2*pi*y(j))*F;
44         end
45     end
46
47     u=halo_update(u);
48
49
50     % ------------ v field ----------------------------
51
52     v = zeros(N+2,N+2);
53
54     x = Mesh (-d/2 , L+d/2 , d);
55     y = Mesh (0, L+d, d);
56
57    for i=2:1:N+1
58         for j=2:1:N+1
59             v(i,j) = -sin(2*pi*x(i))*cos(2*pi*y(j))*F;
60         end
61     end
62
63     v = halo_update(v);
64
65     % ------------ p field ----------------------------
66
67     p = zeros(N+2,N+2);
68
69     x = Mesh (-d/2 , L+d/2 , d);
70     y = Mesh (-d/2 , L+d/2 , d);
```

```
71
72      for i=2:1:N+1
73          for j=2:1:N+1
74              p(i,j) = -F^2 * rho/2 * ( (cos(2*pi*x(i)))^2 + ...
                    (cos(2*pi*y(j)))^2 );
75          end
76      end
77
78      p = halo_update(p);
79
80  end
```

C.8 Store Track

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%%%   StoreTrack    %%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %
5  % ------------------------  DESCRIPTION  -------------------------------%
6  % This function stores the velocity and pressure value of the
7  % CV (i,j) from given Velocity and Pressure fields
8  %
9  % ---------------------  INPUT PARAMETERS  ------------------------------%
10 % {U,V,P} = Velocity and Pressure fields
11 % {i,j} = index of the values to store
12 %
13 % ---------------------  OUTPUT PARAMETERS  -----------------------------%
14 % {u,v,p} = Values of Velocity and Pressure at (i,j)
15 %
16
17 function [ u , v , p ] = StoreTrack (U,V,P,i,j)
18
19 u = U(i,j);
20 v = V(i,j);
21 p = P(i,j);
22
23 end
```

C.9 Time Step

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%   timestep    %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ————————————————————————  DESCRIPTION  ————————————————————————————%
% This function determine if the fluid is domined by CONVECTIVE or
% DIFFUSIVE effects. Consequenctly, it defines the time step required
% for time integration
%
% ————————————————————  INPUT PARAMETERS  ————————————————————————%
% d  =  CV face length
% u0 =  Characteristic valocity
% Re =  Reynolds Number
%
% ————————————————————  OUTPUT PARAMETERS  ————————————————————————%
% At  = Characteristic time step of the problem
%
% ————————————————————  COMPUTE PARAMETERS  ————————————————————————%
% f  = minoration factor for time step
% Atc = characteristic time step for convective effects
% Atd = characteristic time step for diffusive effects
%

function [At] = timestep(d,u0,Re)
% Chose time step for time integration

f=0.4; % minoration factor for time step

Atc = d/u0;          % characteristic time step for convective effects
Atd = 0.5*d*Re/u0;   % characteristic time step for diffusive effects

At=min(Atc,Atd)*f;   % minorated characteristic time step of the problem

if At == Atc*f
    fprintf('The time integretion is dominated by CONVEITVE effects \n');

elseif At == Atd*f
    fprintf('The time integretion is dominated by DIFFUSIVE effects ...
        \n',toc);

end

end
```