

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET
POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE

École Nationale Polytechnique
Département de Génie Mécanique
Laboratory of Green and Mechanical Development



Final year project dissertation

For obtaining State Engineer degree in Mechanical Engineering

Development and validation of a CFD code based on
the finite volume method with a graphical user
interface: Application to laminar, incompressible and
two-dimensional flows

Mohamed SEMMAD

Supervised by: **Dr. A BOUHELAL Pr. A SMAILI**

Publicly presented and defended on Sep 15, 2022

Committee board:

President :	Said RECHAK	Prof (ENP)
Supervisor :	Abdelhamid BOUHELAL	Dr, MCB (ENP)
Co-Supervisor :	Arezki SMAILI	Prof (ENP)
Examiner :	Mohammed Amokrane MAHDI	Dr, MCB (ENP)

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET
POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE

École Nationale Polytechnique
Département de Génie Mécanique
Laboratory of Green and Mechanical Development



Final year project dissertation

For obtaining State Engineer degree in Mechanical Engineering

Development and validation of a CFD code based on
the finite volume method with a graphical user
interface: Application to laminar, incompressible and
two-dimensional flows

Mohamed SEMMAD

Supervised by: **Dr. A BOUHELAL Pr. A SMAILI**

Publicly presented and defended on Sep 15, 2022

Committee board:

President :	Said RECHAK	Prof (ENP)
Supervisor :	Abdelhamid BOUHELAL	Dr, MCB (ENP)
Co-Supervisor :	Arezki SMAILI	Prof (ENP)
Examiner :	Mohammed Amokrane MAHDI	Dr, MCB (ENP)

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET
POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE

École Nationale Polytechnique
Département de Génie Mécanique
Laboratory of Green and Mechanical Development



Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'ingénieur d'état en Génie Mécanique

Développement et validation d'un code CFD basé sur la
méthode des volumes finis avec une interface graphique
: Application aux écoulements laminaires,
incompressibles et bidimensionnels

Mohamed SEMMAD

Encadrants: **Dr. A BOUHELAL Pr. A SMAILI**

Présenté et soutenu publiquement le 15 Septembre 2022

Composition du Jury:

Président :	Said RECHAK	Prof (ENP)
Encadrant :	Abdelhamid BOUHELAL	Dr, MCB (ENP)
Co-Encadrant :	Arezki SMAILI	Prof (ENP)
Examineur :	Mohammed Amokrane MAHDI	Dr, MCB (ENP)

ملخص

الغرض الرئيسي من هذا العمل هو تطوير برنامج لديناميك الموائع الحاسوبية (Computational Fluid Dynamic) وذلك باستعمال طريقة الحجوم المتهمة (Finite Volume Method) واستخدام خوارزمية (SIMPLE) الخاصة بمعادلات المحتوى على عامل الضغط. يحتوي هذا البرنامج على واجهة المستخدم (Graphical User Interface). ويهدف إلى أن يكون أداة تعليمية تستعمل في الدروس التمهيدية في ميكانيك الموائع وديناميك الموائع الحاسوبية (CFD). البرنامج مصمم لحل الحالات المستمرة (غير زمنية)، غير قابلة للانضغاط، ذات الجريان الصفائحي في مجال حساب ديكارتي ثنائي الأبعاد. يمكن للمستخدم من خلال الواجهة المرئية من تحديد خصائص المائع المستعمل، معاملات الشبكة الحاسوبية، ومعاملات برنامج الحساب، وتصور حقول التدفق الناتجة.

تتمثل فلسفة المشروع في تحقيق خطوة أولية صغيرة في مجال العلوم الحاسوبية، والتي نأمل أن تبناها الأجيال القادمة من الطلاب لجعلها مشروعاً مستمرًا للتعليم التطبيقي.

الكلمات المفتاحية: ديناميك الموائع الحاسوبية، برنامج تعليمي، واجهة المستخدم، طريقة الحجوم المتهمة، خورزمية SIMPLE .

Résumé

L'objectif principal de ce travail est de développer un code de calcul de la dynamique des fluides (CFD) basé sur la méthode des volumes finis (FVM) et utilisant la méthode SIMPLE (Semi-Implicit Method for Pressure Linked Equations). Ce logiciel a une interface graphique (GUI) et il est destiné à être un outil pédagogique dans les cours d'introduction à la mécanique des fluides et à la CFD. Le logiciel est capable de résoudre des problèmes laminaires incompressibles en régime permanent sur des grilles cartésiennes bidimensionnelles. L'interface graphique permet à l'utilisateur de définir les propriétés du fluide,

les caractéristiques du maillage, et les paramètres du l'algorithme du calcule, et de visualiser les champs d'écoulement résultants.

La philosophie du projet est de faire un petit pas dans la science computationnelle, qui sera - espérons-le - poursuivi par les prochaines générations d'étudiants pour en faire un projet d'apprentissage continu.

Mots clés : CFD, Logiciel Pédagogique, Interface Graphique, Méthode des Volumes Finis, Algorithme SIMPLE.

Abstract

The main purpose of this work is to develop a Computational Fluid Dynamic (CFD) code based on the Finite Volume Method (FVM) and using the Semi-Implicit Method for Pressure Linked Equations (SIMPLE algorithm). This software has a Graphical User Interface (GUI) and it is intended to be an educational tool in introductory-level fluid mechanics and CFD courses. The software is capable of solving steady-state, incompressible, Laminar problems on two-dimensional Cartesian grids. The GUI enables the user to define the fluid properties, the mesh parameters, the solver settings, and a visualization of the resulting flow fields.

The philosophy of the project is to make an initial, small leap into computational science, that will be -hopefully- carried out by the next generations of students to make it a continuous learning-by-doing project.

Keywords: CFD, Software-Based Education, Graphical User Interface, Finite Volume Method, SIMPLE Algorithm.

Acknowledgements

First and foremost, I thank **ALLAH**, for all his blessings, and for the causes he gave me to accomplish this work.

Secondly, I would like to place a special thanks to my parents, I am grateful for your support throughout my whole life. Sincerely, I found Writing those words for you the most appreciable part of this work.

Also, I am grateful to my supervisors, **Pr. A. BOUHELAL** (MCB, ENP, Algiers) and **Pr. SMAILI** (Professor, ENP, Algiers), for having entrusted me with this research work.

I also avail myself of this opportunity to thank members of my thesis committee: **Pr.Saïd RECHAK** (Professor, ENP, Algiers), and **Pr. M. A. MAHDI** (Professor, ENP, Algiers). I appreciate your time and efforts to evaluate this thesis.

Last but not least, an enormous thanks to all my colleagues at the ENP school, in particular to my *friends* **Macil TAMAZIRT** and **Hani MEHABA**. I am grateful to them for their help and valuable support.

I thank all those who helped me and supported me from near or far,

Thank you very much to all.

Dedication

I would like to dedicate this modest work:

***T**o my dear father: Toufik and **T**o my dear mother: Zohra.*

***T**o my brothers: Abdelkarim, and Walid and **T**o my sister: Fella*

***T**o all members of my family and **T**o all my dear and faithful
friends*

Contents

List of Figures

List of Tables

General Introduction	13
1 Overview	15
1.1 Introduction	16
1.2 CFD Overview	16
1.3 Components of a Numerical Solution	17
1.4 Mathematical Classification of Flows	19
1.4.1 Equations Types Significance	19
1.4.2 Classification Method For Simple PDEs	19
1.5 Numerical Grid	20
1.5.1 Structured (Regular) Grid	20
1.5.2 Block-structured Grid	20
1.5.3 Unstructured Grid	24
1.6 Discretization methods	24
1.6.1 Finite Difference Method	24
1.6.2 Finite Volume Method	26
1.6.3 Finite Element Method	26
1.7 CFD Techniques	26
1.8 Solution of Linear Equations System	27
1.8.1 Point-iterative Methods	28
1.8.2 Multigrid Techniques	28
1.9 Incompressible Flow	29
1.10 Motivation: Educational Software	29
1.11 Conclusion	30
2 Numerical Method	31
2.1 Introduction	32
2.2 Governing Equations	32
2.3 Finite Volume Method	32
2.3.1 The Diffusion Term	34

2.3.2	The Source Term	34
2.3.3	The Convective Term	34
2.3.4	The General Discretized Form	35
2.4	Geometry Discretization: Staggered Grid	36
2.5	The Pressure Correction Method: SIMPLE Algorithm	37
2.5.1	Discretized Momentum Equation on a Staggered Grid	38
2.5.2	The SIMPLE Algorithm	41
2.5.3	The Assembly of the Complete Method	43
2.6	Boundary Conditions	44
2.6.1	Inlet	45
2.6.2	Outlet	46
2.6.3	Wall	47
2.7	Iterative Solver: Gauss Seidel	48
2.8	Conclusion	48
3	The Code's Implementation	50
3.1	Introduction	51
3.2	The Computation Code: <i>ico_ns_solver</i>	51
3.2.1	Phases of Implementation	51
3.2.2	Code Structure	51
3.2.3	The Code's Working Principle	52
3.3	The Graphical User Interface and its Features	53
3.4	Conclusion	56
4	Results And Discussion	57
4.1	Introduction	58
4.2	Validation of the First Stage: Convection-diffusion 1D	58
4.3	Validation of the Second Stage: 1D SIMPLE	59
4.4	Validation of the Third Stage: The Complete Algorithm	59
4.4.1	Pipe Flow: Entry Length Plots	61
4.4.2	Pipe Flow: Contours Plots	61
4.5	Conclusion	63
	General Conclusion	65
	References	68
	The Code's repository	69

List of Figures

1.1	A schematic diagram representing the components of a numerical solution. [1]	18
1.2	Example of a 2D, structured, non-orthogonal grid, designed for calculation of flow in a symmetry segment of a staggered tube bank. [2]	21
1.3	Example of an O-type grid: Streamlines in cylinder vicinity at $Re = 20$. [3]	21
1.4	C-grid around a NACA 0012 airfoil. [4]	22
1.5	Example of a Block-structured grid, designed for calculation of flow around an airfoil. [5]	23
1.6	Example of a 2D block-structured grid which does not match at interfaces, designed for calculation of flow around a hydrofoil under a water surface. [2]	23
1.7	A composite 2D grid, used to calculate flow around a cylinder in a channel. [2]	24
1.8	Three examples of unstructured grids: tetrahedral (upper), polyhedral (middle) and trimmed hexahedral (lower) with prism layers along walls and local grid refinement. [3]	25
2.1	The arrangement for a two-dimensional flow calculation, using a staggered grid [6].	38
2.2	A ‘checker-board’ pressure distribution.[6]	39
2.3	The SIMPLE algorithm.[6]	44
2.4	Three emphasized u momentum cells in the vicinity of the inlet, outlet, and wall boundaries in a pipe flow configuration. [7] . . .	45
2.5	Three emphasized v momentum cells in the vicinity of the inlet, outlet, and wall boundaries in a pipe flow configuration. [7] . . .	46
2.6	Three emphasized p momentum cells in the vicinity of the inlet, outlet, and wall boundaries in a pipe flow configuration. [7] . . .	47
3.1	The principal window of CFDENP	54
3.2	Residuals printing in the command line window for a given simulation.	55
3.3	Visualization of the vector plot of a flow field in a pipe, $Re \approx 1100$. 56	

4.1	A schematic figure of the treated convection-diffusion problem. [6]	58
4.2	Comparison of the the numerical result with the analytical solution for the transported property for a convection-diffusion problem.	59
4.3	A planar two-dimensional nozzle. [6]	60
4.4	comparison of the converged pressure and velocity field after 19 iterations to the analytical solution for the nozzle flow case.	60
4.5	A comparative figure for a pipe flow where plots of the normalized u velocity component at the pipe centerline versus the normalized distance from the inlet for three Re numbers: (a) Re=1000, (b) Re=1500, (c) Re=2000 were presented	61
4.6	Contours plots for Pressure, u velocity, and v velocity fields for three Re number: (a) Re=1000, (b) Re=1500, (c) Re=2000. The plots are arranged so that the first row (top), second and third represent: u-velocities, v-velocities, and Pressure contours respectively	62
4.7	Contours plots for Pressure, u velocity, and v velocity fields for Re=2000 in a pipe flow simulation done using OpenFoam.	63

List of Tables

1.1	The type of the characteristic equation based on the discriminant value. [6]	20
2.1	The neighbor coefficients for common schemes. [6]	37

General Introduction

Background

Fluid flow applications are ubiquitous, to the point that it is even irrelevant to mention examples. Interestingly, most flows and related phenomena can be described by a small set of partial differential equations. Although dating from the 9th century, those equations cannot be solved analytically to this day¹, except for special cases. Part of that difficulty comes from the fact that the solution of the Navier-Stokes equations often includes one of the greatest unsolved problems in physics, namely turbulence. Yet, turbulence has immense importance in science and engineering.

In virtue of that, scientists found the urge to develop a reliable method for tackling fluid flow problems. The goal is to derive results that can be confidently used in engineering calculations where the smallest error can lead to catastrophic ramifications, all with making those methods affordable even with limited resources.

Numerical simulation was the most promising candidate. Relying on the growing power of computer performances and an active community of algorithm developers, numerical approximations of the governing equations of fluid flows were constantly enhanced. Both in the quality of the approximating and the cost in terms of time and resources, the new branch of numerical approximation applied to flow problems gained a reputation in the name of Computational Fluid Dynamics (CFD).

From the early days of CFD, new numerical simulation software (or CFD software) was massively developed. From in-house codes to famous commercial software, CFD is made accessible and ready for use without requiring the re-development of a computer program, hence shortcutting the way for industrials, and tool-users wanting to focus on the development of their project instead of the development of the CFD solution. Mainly, Two types of CFD codes are available for the large public: (i) Commercial software and (ii) Open-source software. Although both have great performances they present some disadvantages

¹To this day, no one has proved that a smooth solution for the Navier-Stokes equations three-dimensional system, and given some initial conditions always exist, nor has anyone found a counter-example. This fact has urged the Clay Mathematics Institute [8] in May 2000 to make the Navier-Stokes existence and smoothness problem one of its seven Millennium Prize problems in mathematics.

for some particular users. First, commercial software are well documented and provides expert assistance, but comes for a very expensive price making them accessible only to large companies and institutions. On the other hand, open-source programs are freely available. However, they present a steeper learning curve for new users, because of their relatively small documentation and assistance.

For educational purposes, generally, resources are limited, making commercial software hardly affordable (if not unaffordable), In addition, Open source software mostly demands larger learning time, which make them a bad choice for educational activity focusing on the understanding of physical phenomenon rather than mastering the software itself.

Objective

In the present work, we aim to develop the building block of a CFD code designed for educational purposes. We restrain our application to laminar, incompressible, and isothermal flow (without energy equation) for a Cartesian uniform grid. The code is based on the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) algorithm. Where we opted for a staggered grid arrangement for geometry discretization. Moreover, the finite volume method (FVM) is used for the discretization of the equations. Finally, since the simple mesh we are using does not require a multi-grid solver we opted for the Gauss-Seidel algorithm as a linear solver. The different solution components are coded in C++ to construct a single compact solver.

Since the project is intended for educational use, it comprises a Graphical User Interface (GUI). The GUI is simplified to one window that contains all the needed inputs, and some basic post-processing features. It is developed in Python using The Tkinter library.

The whole Project is made Open Source, where every interested person can be a contributor to upgrade the project along the targeted development path.

Chapter 1

Overview

1.1 Introduction

In this introductory chapter, we will conduct a brief presentation of the CFD as a scientific research field, then talk about the major components of a CFD study, and finally, we will focus on our theme which concerns only incompressible flows. Also we will present the motivations of this project.

The information given in this chapter pertain to the CFD field in its general picture, and are not specific to our current study. Therefore, we will present information like they are found in books and we will not be restrained to the elements that we will use in the present project.

1.2 CFD Overview

Nowadays numerical simulation is commonly described as the third branch of scientific research, next to the experimental development and mathematical theory [9]. Computational fluid Dynamic is the numerical approach for solving the governing equations of fundamental phenomena present in most engineering applications namely, transport phenomena related to flow motion. In the more general case, the governing equations are the continuity, the momentum, and the energy equations, which are commonly called in the CFD literature as the Navier-Stokes equations ¹ do not have an analytical solution nor a proof about its existence except for some special, enough simplified cases.

The first use of CFD, although this statement depends on the definition of a CFD study, can be traced back to the 1940s [7]. However, it was until the development of digital computers in the mid-1950s and early 1960s that the common knowledge uses of CFD had seen light, and flourished since then. It is a logical fact that the development era of computational resources in terms of storage resources and execution speed has been a catalyzer for the CFD research community for enabling new capabilities in the numerical simulation field. Indeed, a wide range of complex study cases presenting geometrical (e.g. crane of a ship, fuselage of an aircraft, automobile body), or physical (e.g. combustion, multiphase) difficulties becomes feasible after the hardware barrier was removed.

Although CFD was initially developed by some sections within the aeronautic and aerospace industries [9], it is today a crucial tool in many other sectors such as the automotive, power generation, chemical, nuclear, biomedical, and marine industries, to cite a few. The ranks of heavy CFD users have witnessed great growth over the past decades, as a general rule every product that is impacted by a fluid motion, heat transfer, or a phenomenon with similar governing equations, CFD presents a promising tool for the product's development cycle.

To give a brief idea about the power of CFD in contrast to the other dimensions of fluid dynamics (namely, theoretical and experimental approaches), The

¹Historically, the equation obtained by the two scientists (working independently) was the momentum equation. But in the CFD community the resolution of the complete closed system is described as the resolution of the Navier-Stokes equations.

following aspect of CFD are highlighted:

Accessibility As wind, tunnels have been used to obtain sets of data for given flow configuration CFD can be used to give analogous results. However, Computational Fluid Dynamic has the advantage of being a computer program that can be moved anywhere. Or, for a better case, a source program is accessed remotely.

Facility Limitation The physical world has drawbacks compared to CFD, that is nature is not perfectly controlled. For instance, studying a turbulent versus a laminar case using a CFD code is as straightforward as switching the model on and off, whereas in a wind tunnel it's not so simple to control things if it is not impossible in many cases.

Visualization Using preprocessing software, CFD results can be presented in a fashion that is impossible to obtain in an actual laboratory experiment. This advantage allows scientists to draw observations and conclusions more easily and reliably.

Finally, it is wise to note that even if the use of CFD as a design phase tool becomes abundant due to the aforementioned advantages, However, some applications still require a great deal of human and computer resources.

1.3 Components of a Numerical Solution

The numerical solution of a partial differential equation consists of finding the values of a dependent variable ϕ at specified points from which the distribution over the domain of interest can be constructed [1]. Solving a partial differential equation numerically, is finding the values of the dependents variables at some locations of the solution domain. The complete solution can be subsequently obtained by interpolating those points. These points are called grid elements, or grid nodes, and result from the meshing process, which is the discretization of the problem's geometry. These nodes are generally positioned at cell centroids or vertices depending on the adopted discretization procedure [1]. Domain discretization is essential for converting the governing equations to an algebraic system (Equation discretization), written for the variable of interest ϕ . This algebraic system is solved using a linear solver for obtaining the discrete solution of the governing equation. In the following subsections, we will dive into each of the aforementioned steps, which are illustrated in Figure 1.1.

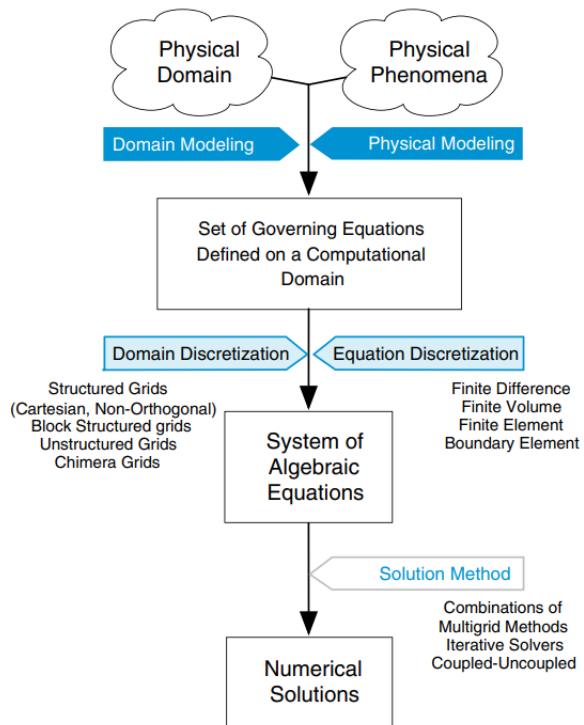


Figure 1.1: A schematic diagram representing the components of a numerical solution. [1]

1.4 Mathematical Classification of Flows

1.4.1 Equations Types Significance

For solving a CFD problem, not only the governing equation have to be known, but also its physical behavior. This condition is a requirement because the initial and boundary conditions –along the governing equation- are mandatory to construct a well-posed problem, and setting those conditions depends on the physical behaviors of a fluid flow. In general, two categories of problems may be encountered:

Equilibrium Problems Those problems are characterized by their steadiness. In the next subsection we will put the equations governing this type of phenomena in the elliptic type. The most representative equation for this category is the Laplace’s equation, which describe potential, incompressible flows for instance.

Marching Problems In this category, the solution can be marched along the flow direction. These problems are governed by parabolic or hyperbolic equations, which arises not only from time-dependent phenomena (like unsteady flows and wave propagations) but in seady flow described by parabolic or hyperbolic equations.

ere, a probe at any point of the solution domain can be only influenced by events at later times. For obtaining a well-posed problem initial and boundary conditions must be specified.

1.4.2 Classification Method For Simple PDEs

A general second order PDE in two co-ordinates x and y have the following form:

$$a \frac{\partial^2 \phi}{\partial x^2} + b \frac{\partial^2 \phi}{\partial x \partial y} + c \frac{\partial^2 \phi}{\partial y^2} + d \frac{\partial \phi}{\partial x} + e \frac{\partial \phi}{\partial y} + f \phi + g = 0 \quad (1.4.1)$$

Note that we are assuming that the equation is linear, and a, b, c, d, e, f, and g are constants. This classification method is based on searching for the roots of the characteristic equation. And it also applies to quasi-linear second-order partial differential equations.

For establishing our classification, we should consider the coefficients of the highest order derivatives. Based on those coefficients, we search for simple wave solutions. This is done by solving the characteristic equation bellow:

$$a \left(\frac{\partial y}{\partial x} \right)^2 - b \left(\frac{\partial y}{\partial x} \right) + c = 0 \quad (1.4.2)$$

Based on the discriminant of equation 1.4.2, table 1.4.2 will distinguish between three cases.

$b^2 - 4ac$	Equation type	characteristics
> 0	Hyperbolic	Two real characteristics
$= 0$	Parabolic	one real characteristics
< 0	Elliptic	No characteristics

Table 1.1: The type of the characteristic equation based on the discriminant value. [6]

It is important to note that the equation may have more the one type, and that is in dependence to the considered regions of the solution domain.

1.5 Numerical Grid

Before calculations are carried out, the calculation domain must be discretized, i.e. an amount of location is defined throughout the domain where flow variables will be evaluated. Many *Meshing* families exist depending on the treated geometry. Each family has its specific challenges, but the shared principle is that the calculation domain is divided into a finite number of subdomains.

1.5.1 Structured (Regular) Grid

Regular or structured grids are constructed by respecting the propriety that two lines of one single-family: (i) do not cross each other, and (ii) cross each line of the other families exactly once. This structure allows the numbering of the grid point by the mean of indices just like in a Cartesian grid.

Based on the shape of the grid line we can distinguish H-, O-, or C-type grids. Figure 1.2 shows an H-type grid, it has the property that when mapped into a rectangular grid, one cell will have distinct boundaries (four in the 2D case). An O-type grid is shown in figure 1.3, which corresponds to an flow around a cylinder. In the airfoil flow configuration of Figure 1.4, a C-type grid is used, we note that, in this case, an “artificial cut”, as commonly known is introduced, to allow the grid lines around the airfoil jumping from infinity to 0.

An important remark is that: to be able to map a structured grid into a rectangular one, the mathematical expression of the grid’s lines should be known.

1.5.2 Block-structured Grid

Many flow problems do not require mesh refinement throughout the calculation domain, but only in the area of interest. For this application a coarse level and a fine level are used, therefore the domain will contain two level (or more) of subdivisions. Certainly, special treatment is needed for the interferences. Figure 1.5 is an example of a block structured grid it contains six block with different grid type, block number 3 has a C-type grid, while the other blocks have an H-type grid.

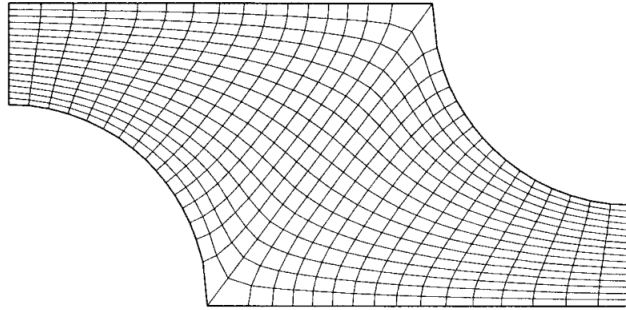


Figure 1.2: Example of a 2D, structured, non-orthogonal grid, designed for calculation of flow in a symmetry segment of a staggered tube bank. [2]

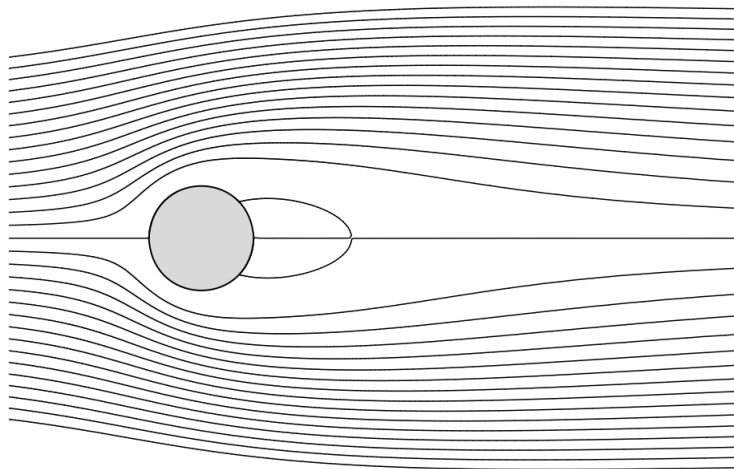


Figure 1.3: Example of an O-type grid: Streamlines in cylinder vicinity at $Re = 20$. [3]

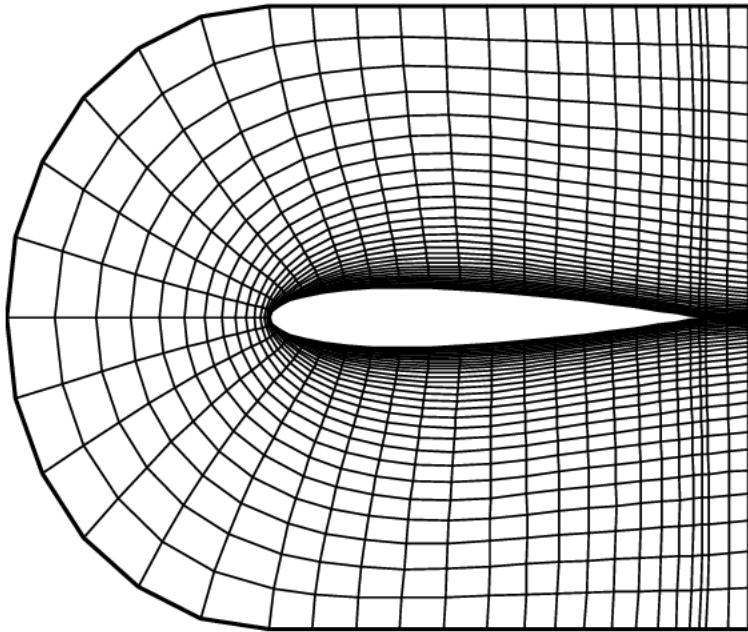


Figure 1.4: C-grid around a NACA 0012 airfoil. [4]

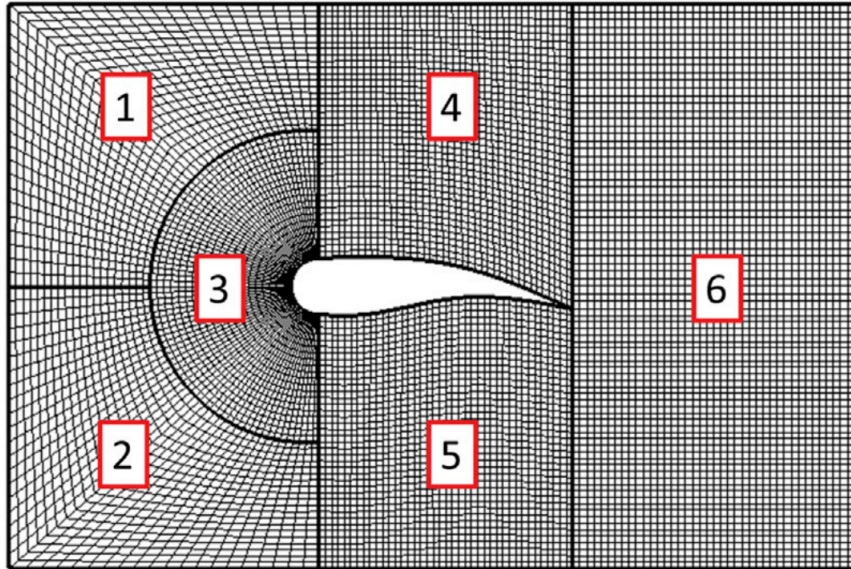


Figure 1.5: Example of a Block-structured grid, designed for calculation of flow around an airfoil. [5]

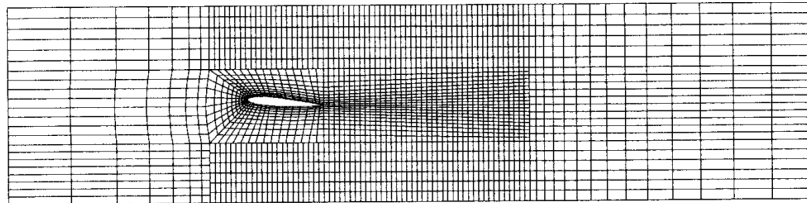


Figure 1.6: Example of a 2D block-structured grid which does not match at interfaces, designed for calculation of flow around a hydrofoil under a water surface. [2]

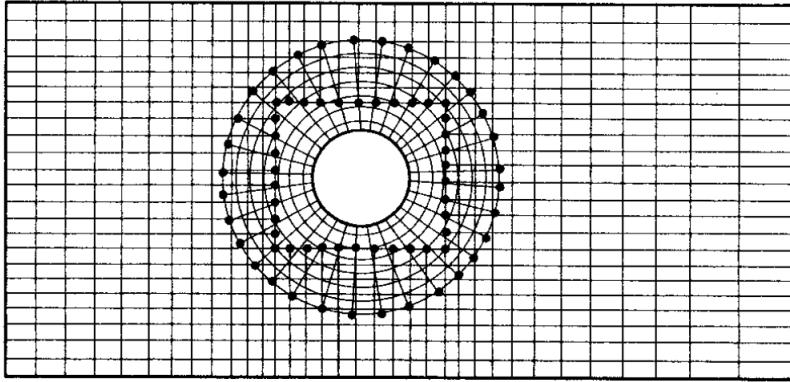


Figure 1.7: A composite 2D grid, used to calculate flow around a cylinder in a channel. [2]

1.5.3 Unstructured Grid

For arbitrary geometries, more flexibility for fitting the body shape is needed. This is the cause for using unstructured grids. In principle, such grids have no restriction on term of the discretization method, however the Finite Volume Method is clearly the best suited option.

Because the fluxes being the important quantity, no constraint is placed on the cells shape nor on the neighboring faces. Despite that any cell shape can be used, the commonly used 2D grid are made of triangles (tetrahedron in 3D), or quadrilaterals (hexahedra in 3D).

Unstructured Grids generation is research field in it self, many computer programs are developed specifically for that purpose with an uneven reliability. The aims for development is present because Unstructured Grids complicate the solving process. For instance since the corresponding matrix of the algebraic equations system no longer has a regular, diagonal structure, the solvers are usually slower than those for regular grids. In Figure 1.8 a set of three unstructured grids is presented.

1.6 Discretization methods

1.6.1 Finite Difference Method

Finite Difference Method was introduced by Euler in the 18th century. It has a basic mathematical foundation, and it is more adapted for simple geometries. Its mathematical foundation relies on replacing differentiation with finite differences, therefore, the differential form of the conservation equation is more suited for the FDM formulation. The general idea of the method is about approximating the nodes' values of the approximation field as unknown in a system of algebraic equations. Many finite difference representations of derivatives exist,

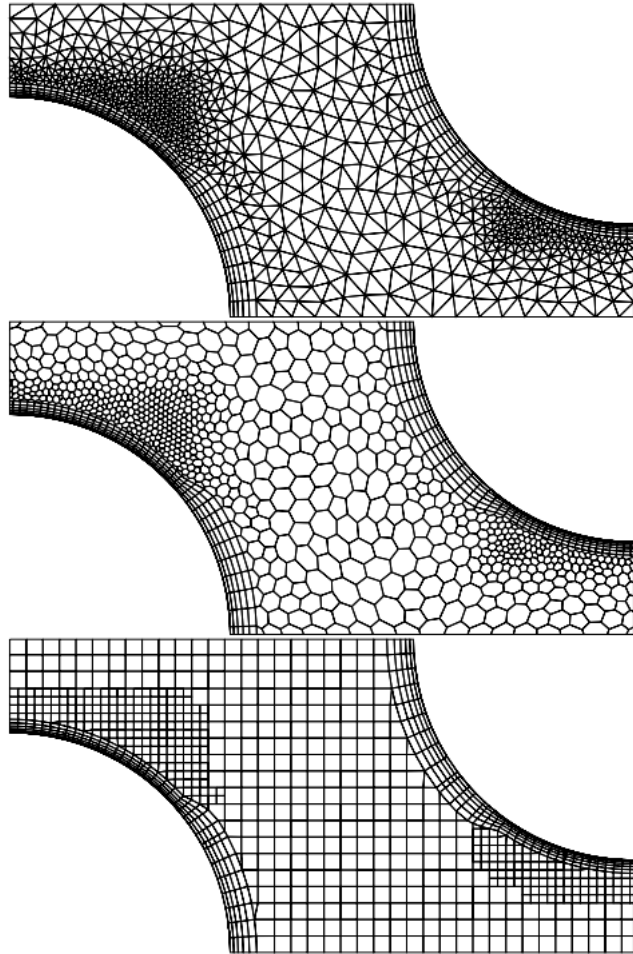


Figure 1.8: Three examples of unstructured grids: tetrahedral (upper), polyhedral (middle) and trimmed hexahedral (lower) with prism layers along walls and local grid refinement. [3]

with variations in the accuracy order and the number of the implicated grid point, known as the *stencil*. Two main approaches exist that are used for obtaining an FDM representation, (i) the Taylor Series Expansion, and the (ii) Polynomial Approach.

The first approach is the basic one; it allows an explicit evaluation of the accuracy order, which is defined as the order of the term of the highest magnitude in Taylor's series truncation error. The second approach is more suited for elaborating FDM representation of higher order of accuracy, which usually involves a larger stencil. The basic idea of this approach is to write the derivative in question as a linear combination of the chosen grid points and determine those coefficients subsequently.

Commonly, FDM is applied only to structured meshes; however, FDM is applicable to any type of network, but with more complexity. This is due to the simplicity and effectiveness of the method on structured meshes. The principal disadvantage of the method is the non-enforcement of conservation (unless with special treatment), furthermore, the restriction to simple geometry prevents the method from the use in complex flow.

1.6.2 Finite Volume Method

In contrast to the FDM, the discretization of the integral form of fluid flow governing equations is called finite volume. Most of the commercial solvers use this technique [7] (e.g Fluent, STAR-CD, CFX). One of the main advantages of FVM is that it can be used for unstructured grids without any transformation. Another advantage is related to a fundamental principle contained behind the integral form, which is the ability to handle discontinuities (most commonly induced by shock waves) in the flow field. The other form of the governing equations (the differential form) supposes the continuity of the solution letting the FDM formulation fundamentally not appropriate for such flows.

1.6.3 Finite Element Method

The method was initially developed in the structural analysis, and then generalized to the whole continuum mechanics [7]. It is based on the approximation of the solution with simple (linear or quadratic) functions each one valid only in one sub-region of the domain. This will reduce the complex partial differentials to simple piecewise functions. As stated earlier, most CFD packages use FVM, However, some FVM-based packages exist (e.g Ansys Flotran, CFdesign) [7].

1.7 CFD Techniques

CFD is about approximating fluid flow with the best available precision. And since flow phenomena are very diverted, fluid researchers have been continuously seeking more efficient methods to perform that approximation. Their work has conducted to a diverse classification of CFD techniques, where each method

has its own strength, weaknesses, and range of applicability. In the present discussion, We will focus on CFD techniques adapted for solving incompressible flow problems.

It is worth noting that an exhaustive review of all the techniques –if it is not an impossible task- is out of our present discussion (such reviews can be found in [10], [11]). However, a brief review of some techniques developed in the era of viscous incompressible flows is provided below, with more detail given on the pressure correction technique in section 2.5. There are two main categories of approaches for solving incompressible Navier-Stokes equations:

- Primitive Variable Approaches;
- Primitive Variable Approaches.

In the first formulation, the system of equations is written with pressure and velocity components as independent variables, and the resolution is based on pressure or on density [7]. For the second formulation, the governing equation is written in function of derived quantities such as the vorticity-stream function, and the primitive variables (i.e. pressure and velocity) are calculated later as dependent variables [7]. Some examples of methods using the first formulation are as follows:

- Pressure based
 - The Marker and Cell (MAC) method
 - Fractional Step Method
 - Semi-Implicit Method for Pressure Linked Equations (SIMPLE)
 - Density Based
 - * Artificial Compressibility Method

In pressure-based approaches, the incompressibility is satisfied directly by using the pressure as a mapping parameter. The density-based approach is based on compressible flow formulation; by taking advantage of the fact that equations are coupled through density, and recovering incompressibility as a limiting case [7].

1.8 Solution of Linear Equations System

Solving linear systems is a crucial part of a CFD analysis; indeed, it is alike for all numerical analysis in general. This is why numerical resolution of linear system has ever been an active research field.

In CFD, the discretization process approximates the analytical formulation of the physical phenomenon by a system of algebraic equations. Which are linear or non-linear according to the nature of the original PDE(s). Non-linear linear algebraic equations follow a three steps resolution method, which is about: i)

Guessing, ii) linearizing the solution about the guessed solution, iii) applying corrections, until convergence is achieved. Therefore, even non-linear algebraic equation need a linear solver, and depend on its performances.

Sparsity ¹. is what characterize the matrices of the linear systems. If the used grid is structured, the matrix will have a favorable form for the resolution process, in which the non-zero elements lie on a small number of well-defined diagonals. For the general case, more sophisticated and adapted method should be used for reaching a reasonable resolution time.

In the present section, we will restrict ourselves to iterative techniques, which have the advantage of requiring a lower storage overhead ², making them preferred over their counterpart: the direct techniques, especially for large systems. Moreover, we will just give a brief presentation of the most common iterative linear solver algorithms. For a more thorough and deep review, the reader is directed to the specialized bibliography.

1.8.1 Point-iterative Methods

The name Point-iterative come from the fact that the solution is updated sequentially node by node. A general rule in point-iterative methods is that the degree of explicitness is inversely proportional to the order or convergence. This is important to keep in mind when trying to make a compromise between these two factors.

Jacobi Method

In the Jacobi ³ update process, all off-diagonal (neighboring) terms are treated explicitly (i.e. their values are taken from the previous iteration). Since the method is a point wise method, it can be applied to both structured and unstructured meshes.

Gauss–Seidel Method

It is founded on the basis of the Jacobi method by adopting an important modification; at each iteration whenever an updated value is calculated, it will be use for the upcoming calculations. This implies an increase in the degree of implicitness and hence, an improvement in the convergence rate in expected.

1.8.2 Multigrid Techniques

Multigrid concept is designed to take advantage of the inherent differences of the error behavior by using iterations on meshes of different size. [6]

¹A sparse matrix is a matrix which most of its elements are zero.

²Storage overhead is the memory needed for performing a specific task. [12]

³*Carl Gustav Jacobi* (1804–1851) was a German mathematician. He is best known for his development of the theory of elliptic functions, and their applications to solving inverse problems involving periodicity. He also made significant contributions to the theory of differential equations and number theory[5]

1.9 Incompressible Flow

The compressibility of a fluid is related to the change in its density. The assumption of an incompressible flow implies a constant density throughout all the flow fields. It turns out that this assumption is justified not only for liquids but also for gases under certain conditions. This condition is explicitly derived from the relation of compressibility to the flow's Mach number. It is demonstrated that as the Mach number approaches zero, the compressibility becomes negligible (i. e. a constant flow density). In engineering the common limit for a flow to be incompressible is $M = 0.3$. It is important to note that most natural flows, and engineering applications flows fall under this condition.

From a CFD perspective, this assumption has both a positive and a negative effect on the numerical solution. By admitting constant density; (i) problems involving energy transfer will require a separate resolution of the energy equation on one side and the Navier Stokes equations on the other side, this is what we call decoupling.; On the other hand, (ii) Another decoupling is induced between the pressure and the velocity field, this is because no time evolution for the pressure is present in the continuity equation. This presents a major difficulty for the numerical resolution; as a result, many algorithms addressing this difficulty were developed.

In the present work, we have adopted a pressure correction technique based on the SIMPLE algorithm.

1.10 Motivation: Educational Software

In the majority of engineering curriculum, CFD is limited to graduate-level courses. This is because a mathematical background is needed to write a CFD programs. Although recent undergraduate-level fluid mechanics books involve CFD-related chapters, references indicating the use of CFD as a teaching aid are limited [7]. The aim of the present work is to develop a CFD code to be used as i) an educational support in fluid mechanics courses and ii) a practicing tool in CFD related courses.

Fluid mechanics in the National Polytechnic School is taught as a on semester course in the preparatory classes. After that, in the second cycle each department has a different approach for taking this course, with the Mechanical Engineering Department being the department which takes the courses in much details and for a longer period. Some related courses like *Turbomachinery*, flows, are made a separate courses that are taught over one semester or a half of a semester.

Due to the inherent complexity of fluid motion, fluid flow problems require a different viewpoint compared to solid mechanics problems. Understanding topics like continuum assumption and its validity, proper comprehension of the field concept such as the velocity or the pressure field, making the switch from the classical Lagrangian approach -which is taught in earlier statics and dynamics courses-, to the Eulerian approach, establishing the link between these

two different view points, developing mathematical and physical understanding of the convective derivatives; are some of the challenges that the students face with when they begin studying fluid mechanics [7].

Other than the aforementioned challenges facing the beginner students is; the fact that fluid mechanics phenomenon are hardly observed in everyday life in comparison to solid mechanic problem. For example, a deflected beam is a very common observation that the student can builds upon when studying bending, on the other hands, the control volume analysis may not find a concrete common phenomenon that the student uses as a visualization of the theoretical foundation. Other issues may arise when talking about space and time variability of fluid properties, or the presence of turbulence in most of fluid flow applications.

As a rule of thumb, visualization always facilitate the understanding of the physical phenomenon. In this respect, an educational experiment is very important. That is why most of the Fluid mechanics courses are scheduled to comprise a laboratory time. Unfortunately, this option depends on the equipment available in the lab, which is in most cases scarce and not reliable. Educational videos for laboratory experiment are available for the large public, and are very helpful, such as the ones prepared by NCFMF [13] and IIHR [14], but learning by doing remains the best option. Obviously, CFD is the state of the art alternative. Although experimentation remains valuable for understanding the physical phenomena and deriving mathematical models, CFD can give satisfying results to mathematical models that are impossible to solve. This is a great tool to put in the hand of student for pushing the boundary of their thinking beyond the analytical solutions. In addition, One simple advantage of this is its power in attracting the attention of today's computer oriented students [15].

1.11 Conclusion

CFD is a powerful tool that can be integrated not only in the industry sector but also as an educational tool in the engineering field. It helps the beginner student to understand complex phenomenon, and to widen their thinking beyond the analytical methods.

A user friendly CFD code is a valuable tool to put in the hand of engineering student, especially if it is an open source code that every interested individual can be part of it, by making it more reliable and multi-function.

Chapter 2

Numerical Method

2.1 Introduction

In the present chapter, the mathematical foundation of the flow solver will be presented. Starting by presenting the governing equations, and then discussing the discretization method of those equations, Next the geometric grid will be specified, also the SIMPLE algorithm will be presented and assembled, finally, we will talk about the implementation of the boundary conditions and the linear solver that we will use in our code.

2.2 Governing Equations

As described in the introduction CFD is about solving flow equations numerically, therefore it is necessary to state the governing equations of our study case clearly before getting into the resolution procedure. Here we are interested in the steady, incompressible, flow with constant viscosity and uniform temperature. The constant viscosity assumption is well justified since we are dealing with a uniform temperature flow. Also, under the last assumption, the energy equation is not a part of our governing equation system since no temperature field needs to be determined. A flow under the aforementioned assumption is governed by the Navier-Stokes and continuity equations are given by:

$$\frac{\partial}{\partial x}(\rho u u) + \frac{\partial}{\partial y}(\rho v u) = \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) - \frac{\partial p}{\partial x} + S_u \quad (2.2.1)$$

$$\frac{\partial}{\partial x}(\rho u v) + \frac{\partial}{\partial y}(\rho v v) = \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) - \frac{\partial p}{\partial y} + S_v \quad (2.2.2)$$

$$\frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) = 0 \quad (2.2.3)$$

2.3 Finite Volume Method

The discretization scheme adopted for the present work is the Finite Volume Method (FVM). In the present subsection, we will derive a general expression for FVM discretization that can be applied subsequently to our governing equations. To obtain a general expression, we need to start from the generic form of the transport equation for a scalar quantity ϕ , this equation is given by equation (2.3.1),:

$$\frac{\partial(\rho\phi)}{\partial t} + \text{div}(\rho\phi\mathbf{u}) = \text{div}(\Gamma \text{grad} \phi) + S_\phi \quad (2.3.1)$$

The transport equation for any flow variable is obtained from the substitution of ϕ by the corresponding term, for example for obtaining the u-momentum equation, ϕ is replaced by the u component of the velocity. The source term S_ϕ ,

the diffusion coefficient Γ , need also to be replaced appropriately to obtain the desired equation. It is important to note that in this section we suppose that the velocity field is known, which is not true for real problems. However, the complete technique for solving a CFD problem that takes care of the velocity field will be presented in the next section.

In our study case, the governing equations correspond to steady-state, diffusion-convection with a source term, therefore we will focus on that particular case for the derivation of the discretized equation. To that end, we will start from the following equation obtained from equation (2.3.2) by omitting the transient term.

$$\text{div}(\rho\phi\mathbf{u}) = \text{div}(\Gamma \text{grad} \phi) + S_\phi \quad (2.3.2)$$

For any space discretization, a computation grid must be predefined, for our example, we will use the grid presented in figure 2.1. The figure illustrates the convention adopted for noting the neighboring nodes of point P, which is the node of interest. The equation (2.3.2) will be integrated over the control volume represented by the shaded area in figure 2.1. The control volume boundaries are positioned midway between the adjacent nodes. Each adjacent node is noted with the capital letter of the first letter of its direction, the same is applicable for the control volume faces but using lowercase letter. The distances between each node can be noted as $\delta x_{\text{point1 point2}}$ but since our control volume faces are well centered between the nodes, we can simply denote the horizontal distance as Δx and the vertical distance as Δy .

After the computation domain is defined, an integration over a control volume of a nodal point P is performed as follows:

$$\int_{\Delta V} \text{div}(\rho\phi\mathbf{u}) dV = \int_{\Delta V} \text{div}(\Gamma \text{grad} \phi) dV + \int_{\Delta V} S_\phi dV \quad (2.3.3)$$

The application of the divergence theorem, which states that:

$$\int_{CV} \text{div}(\mathbf{a}) dV = \int_A \mathbf{n} \cdot \mathbf{a} dA \quad (2.3.4)$$

Yields:

$$\int_A \mathbf{n} \cdot (\rho\phi\mathbf{u}) dA = \int_A \mathbf{n} \cdot (\Gamma \text{grad} \phi) dA + \int_{CV} S_\phi dV \quad (2.3.5)$$

Here ΔV is the volume, A is the cross-sectional area of the control volume face, and S is the average value of source S over the control volume.

For simplicity's sake, we will perform the integration for a one-dimensional case; the result can be straightforwardly extended to a multidimensional case. By calculating the integral, the following expression is obtained:

The last step in the derivation is the calculation of the value of transported property ϕ at control volume faces and the convective fluxes across these boundaries. Also, the diffusion term and the source term need to be differentiated to obtain the discretized equation. Each term will be discussed in the following subsections.

2.3.1 The Diffusion Term

The derivatives of the diffusion term and the interface diffusion coefficient Γ are generally evaluated using a linear approximation between the neighboring nodes. This practice is called the central difference scheme. In a uniform one dimensional grid (equivalent to the one presented in figure 2.1 with ignoring the second dimension) the linear interpolation of Γ_e and Γ_w is given by:

$$\Gamma_w = \frac{\Gamma_W + \Gamma_P}{2} \quad (2.3.6)$$

$$\Gamma_e = \frac{\Gamma_P + \Gamma_E}{2} \quad (2.3.7)$$

And the diffusive fluxes are evaluated as:

$$\left(\Gamma A \frac{d\phi}{dx} \right)_e = \Gamma_e A_e \left(\frac{\phi_E - \phi_P}{\delta x_{PE}} \right) \quad (2.3.8)$$

$$\left(\Gamma A \frac{d\phi}{dx} \right)_w = \Gamma_w A_w \left(\frac{\phi_P - \phi_W}{\delta x_{WP}} \right) \quad (2.3.9)$$

2.3.2 The Source Term

For many situations, the source term may be a function of the dependent variable. In a finite volume method formalism, this situation is handled by approximating the source term in a linear form:

$$\bar{S}\Delta V = S_u + S_p\phi_P \quad (2.3.10)$$

2.3.3 The Convective Term

The principal problem of FVM discretization arises from the convective term. This problem is encountered when evaluating the transported property ϕ at control volume faces and the convective fluxes across these boundaries. It seems a logical intuition that the difference scheme adopted for the diffusion term can also be applied here, however, diffusion influence the distribution of the transported quantity in all direction along its gradient, whereas convection spreads influence only in the flow direction [6]. This fact presents a fundamental difference between the two transport modes and manifests itself in a stringent upper limit to the grid size leading to stable calculation with central differencing. This limit is dependent on the relative strength between convection and diffusion in the flow.

For that reason, many alternatives for the discretization of the convective term are developed. Each one has its order of accuracy, stability condition, and level of complexity. We represent here the central difference scheme and the upwind scheme which is the basic first-order difference base on the upstream node.

For the central difference scheme:

$$\phi_e = \frac{\phi_P + \phi_E}{2} \quad (2.3.11)$$

$$\phi_w = \frac{\phi_W + \phi_P}{2} \quad (2.3.12)$$

For the Upwind scheme: For the positive direction $u_w > 0$, $u_w > 0$ we set:

$$\phi_e = \phi_P \quad (2.3.13)$$

$$\phi_w = \phi_W \quad (2.3.14)$$

For the negative direction $u_w < 0$, $u_w < 0$ we set:

$$\phi_e = \phi_E \quad (2.3.15)$$

$$\phi_w = \phi_P \quad (2.3.16)$$

It is important to note that regardless of the used discretization scheme, the total convection-diffusion equation is arranged in a general form (equation 2.3.24) that its coefficient takes different expressions depending on the used scheme, we will present this form in the next subsection.

2.3.4 The General Discretized Form

We will derive the general discretized equation in details for a Central Difference evaluation of the convective term. After the general expression is derived, a summarizing table will be presented for the different coefficients values for the other schemes (Table 2.1). By replacing the discretized form of all the three terms in in equation 2.3.5 the above expression is obtained:

$$\frac{F_e}{2} (\phi_P + \phi_E) - \frac{F_w}{2} (\phi_W + \phi_P) = D_e (\phi_E - \phi_P) - D_w (\phi_P - \phi_W) \quad (2.3.17)$$

Where F and D , are two defined variables that represent the mass flux per unit area and diffusion conductance at cell faces respectively. Their value at the cell faces can be written as:

$$F_e = (\rho u)_e \quad (2.3.18)$$

$$F_w = (\rho u)_w \quad (2.3.19)$$

$$D_e = \frac{\Gamma_e}{\delta x_{PE}} \quad (2.3.20)$$

$$D_w = \frac{\Gamma_w}{\delta x_{WP}} \quad (2.3.21)$$

The equation 2.3.17 can be rearranged to give:

$$\left[\left(D_w - \frac{F_w}{2} \right) + \left(D_e + \frac{F_e}{2} \right) \right] \phi_P = \left(\left(D_w + \frac{F_w}{2} \right) \right) \phi_w + \left(\left(D_e - \frac{F_e}{2} \right) \right) \phi_E \quad (2.3.22)$$

$$\left[\left(D_w + \frac{F_w}{2} \right) + \left(D_e - \frac{F_e}{2} \right) + (F_e - F_w) \right] \phi_P = \left(D_w + \frac{F_w}{2} \right) \phi_w + \left(D_e - \frac{F_e}{2} \right) \phi_E \quad (2.3.23)$$

Setting the coefficient of ϕ_W , and ϕ_E equal to a_W and a_E respectively, the central differencing expression for the discretized convection-diffusion equation can be written as:

$$a_P \phi_P = a_W \phi_W + a_E \phi_E + S_u \quad (2.3.24)$$

Where:

$$a_P = a_W + a_E + (F_e - F_w) - S_p \quad (2.3.25)$$

$$a_w = \left(D_w + \frac{F_w}{2} \right) \quad (2.3.26)$$

$$a_e = \left(D_e - \frac{F_e}{2} \right) \quad (2.3.27)$$

In the same vein, the discretized expression for the other schemes can be derived. Furthermore, it can be shown that they all take the same form as equation 2.3.24. The only difference is in the values of the coefficients. The table 2.1 summarizes the values of the corresponding coefficient for some common schemes.

2.4 Geometry Discretization: Staggered Grid

The principle of discretization discussed in the introduction needs some pre-set points in the calculation domain, where the flow-fields values are calculated and stored. This system of points is called a grid (or a mesh). Here, for our two-dimensional problem a uniform Cartesian grid is adopted as shown in figure 2.1.

This grid is of a particular type called a staggered grid, which means that the velocities and pressure fields are stored at different grid points. A staggered grid is obtained by considering a separate control volume of each of the velocities components (u and v in 2D) and the scalar field (pressure field in our context).

Scheme	a_W	a_E
Centrale differencing	$D_w + F_w/2$	$D_w - F_w/2$
Upwind differencing	$D_w + \max(F_w, 0)$	$D_e + \max(0, -F_e)$
Hybrid differencing	$\max[f_w, (D_w + F_w/2), 0]$	$\max[-f_e, (D_e - F_e/2), 0]$
Power law	$D_w \max[0, (1 - 0.1 Pe_w ^5) + \max(F_w, 0)]$	$D_e \max[0, (1 - 0.1 Pe_e ^5) + \max(-F_e, 0)]$

Table 2.1: The neighbor coefficients for common schemes. [6]

these considerations allow us to address each node (filled or open nodes) by two coordinate systems. The first corresponds to the scalar field nodes, shown as filled points, linked by solid lines in the figure 2.1, these locations are addressed by the capital letter indices (I, J, I-1...etc). The second coordinate system corresponds to the velocity field nodes, shown as open points and placed at cell faces of the scalar grid, these locations are addressed by the lower letter indices (Namely: i, j, i-1...etc). A point situated on the intersection of two lines from different grids is simply addressed by a combination of the two notation (exp.: (I, j-1), (i,J)...etc). Two types of staggered grids can be adopted; a forward or a backward staggered grid. The one shown in fig. 2.1 is a backward one. Because the i-th row of the second grid is below the I-th row of the first grid, and the j-th column of the second grid is below the J-th column of the first grid.

The mean reason for introducing such a grid system is for avoiding the non-physical behavior of non-uniform fields in an ordinary grid where all flow fields are stored at the same nodes. This problem is illustrated by a checkerboard distribution (See Fig. 2.2) in an ordinary grid. In this case, the difference equations for pressure/velocities gradients give a uniform, null distribution, which is a non-physical behavior.

2.5 The Pressure Correction Method: SIMPLE Algorithm

In 1972 Patankar and Spalding introduce an iterative method based on pressure correction, for the calculation of the pressure field on a staggered grid [6]. SIMPLE stands for "Semi-Implicit Method for Pressure Linked Equations". It is called "semi-implicit" because the pressure correction formula -as introduced shortly- does not involve the pressure values at all grid nodes, but only at some neighboring nodes.

It is important to note that the FVM discretization scheme presented earlier, presents only a tool that can be used in a complete CFD technique as the one presented here. This is because in our derivation of the FVM discretization we

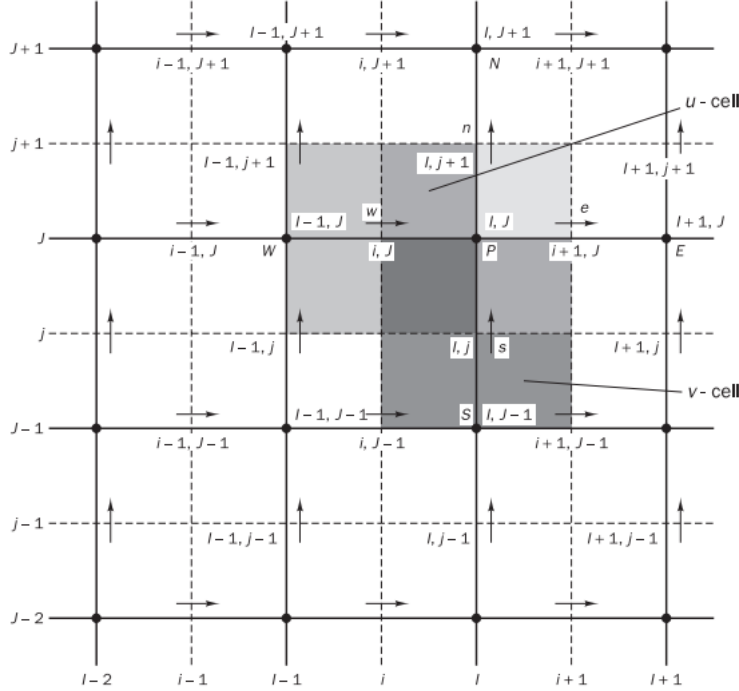


Figure 2.1: The arrangement for a two-dimensional flow calculation, using a staggered grid [6].

supposed a known velocity field, which is not the case for real problems. For that reason, the SIMPLE algorithm is set to solve the complete set of governing equations for all the independent variables.

2.5.1 Discretized Momentum Equation on a Staggered Grid

The algebraic equation 2.3.24 can be re-expressed in the staggered grid coordinate system. Here we replace the transported variable ϕ by u , and the pressure term is taken out of the source term and discretized by the mean of a linear interpolation between the pressure nodes on the u-control volume boundaries. This yields:

$$a_{i,J}u_{i,J} = \sum a_{nb}u_{nb} + \frac{(P_{I,J} - P_{I-1,J})}{\delta x_u} \Delta V_u + \bar{S} \Delta V_u \quad (2.5.1)$$

Or

$$a_{i,J}u_{i,J} = \sum a_{nb}u_{nb} - (P_{I-1,J} - P_{I,J}) A_{i,J} + b_{i,J} \quad (2.5.2)$$

Where ΔV_u is the volume of the u-cell, $b_{i,J} = \bar{S} \Delta V_u$ is the momentum

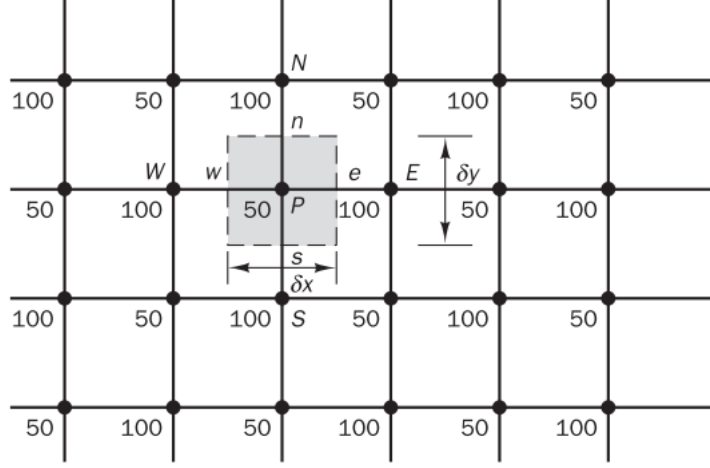


Figure 2.2: A ‘checker-board’ pressure distribution.[6]

source term, $A_{i,J}$ is the (east or west) cell face area of the u-control volume. The coefficients $a_{i,J}$, and a_{nb} may be calculated with any of the differencing methods (see table 2.1). The calculation of $a_{i,J}$, and a_{nb} involves the variables F and D , therefore the values of these two variables in the new coordinate system are given below:

$$\begin{aligned}
 F_w = (\rho u)_w &= \frac{F_{i,J} + F_{i-1,J}}{2} \\
 &= \frac{1}{2} \left[\left(\frac{\rho_{I,J} + \rho_{I-J,J}}{2} \right) u_{i,J} + \left(\frac{\rho_{I-1,J} + \rho_{I-2,J}}{2} \right) u_{i-1,J} \right] \quad (2.5.3)
 \end{aligned}$$

$$\begin{aligned}
 F_e = (\rho u)_e &= \frac{F_{i+1,J} + F_{i,J}}{2} \\
 &= \frac{1}{2} \left[\left(\frac{\rho_{I+1,J} + \rho_{I,J}}{2} \right) u_{i+1,J} + \left(\frac{\rho_{I,J} + \rho_{I-1,J}}{2} \right) u_{i,J} \right] \quad (2.5.4)
 \end{aligned}$$

$$\begin{aligned}
 F_s = (\rho v)_s &= \frac{F_{I,j} + F_{I-1,j}}{2} = \\
 &= \frac{1}{2} \left[\left(\frac{\rho_{I,J} + \rho_{I,J-1}}{2} \right) v_{I,j} + \left(\frac{\rho_{I-1,J} + \rho_{I-1,J-1}}{2} \right) v_{I-1,j} \right] \quad (2.5.5)
 \end{aligned}$$

$$\begin{aligned}
F_n = (\rho v)_n &= \frac{F_{i,J+1} + F_{i-1,J+1}}{2} \\
&= \frac{1}{2} \left[\left(\frac{\rho_{I,J+1} + \rho_{I,J}}{2} \right) v_{I,j+1} + \left(\frac{\rho_{I-1,J+1} + \rho_{I-1,J}}{2} \right) v_{I-1,J+1} \right] \quad (2.5.6)
\end{aligned}$$

$$D_w = \frac{\Gamma_{I-1,J}}{x_i - x_{i-1}} \quad (2.5.7)$$

$$D_e = \frac{\Gamma_{I,J}}{x_{i+1} - x_i} \quad (2.5.8)$$

$$D_s = \frac{\Gamma_{I-1,J} + \Gamma_{I,J} + \Gamma_{I-1,J-1} + \Gamma_{I,J-1}}{4(y_J - y_{J-1})} \quad (2.5.9)$$

$$D_n = \frac{\Gamma_{I-1,J+1} + \Gamma_{I,J+1} + \Gamma_{I-1,J} + \Gamma_{I,J}}{4(y_{J+1} - y_J)} \quad (2.5.10)$$

The values of the F and D are calculated based on the previous iteration of the velocity field. By analogy the discretized v-momentum equation becomes:

$$a_{i,J} v_{i,J} = \sum a_{nb} v_{nb} - (P_{I-1,J} - P_{I,J}) A_{i,J} + b_{i,J} \quad (2.5.11)$$

Similarly, The calculation of $a_{i,J}$, and a_{nb} are combinations of the variables F and D , therefore the values of these two variables in the new coordinate system are given below:

$$\begin{aligned}
F_w = (\rho u)_w &= \frac{F_{i,J} + F_{i,J-1}}{2} = \\
&= \frac{1}{2} \left[\left(\frac{\rho_{I,J} + \rho_{I-1,J}}{2} \right) u_{i,J} + \left(\frac{\rho_{I-1,J-1} + \rho_{I,J-1}}{2} \right) u_{i,J-1} \right] \quad (2.5.12)
\end{aligned}$$

$$\begin{aligned}
F_e = (\rho u)_e &= \frac{F_{i+1,J} + F_{i+1,J-1}}{2} = \\
&= \frac{1}{2} \left[\left(\frac{\rho_{I+1,J} + \rho_{I,J}}{2} \right) u_{i+1,J} + \left(\frac{\rho_{I,J-1} + \rho_{I+1,J-1}}{2} \right) u_{i+1,J-1} \right] \quad (2.5.13)
\end{aligned}$$

$$\begin{aligned}
F_s = (\rho v)_s &= \frac{F_{I,j-1} + F_{I,j}}{2} = \\
&= \frac{1}{2} \left[\left(\frac{\rho_{I,J-1} + \rho_{I,J-2}}{2} \right) v_{I,j-1} + \left(\frac{\rho_{I,J} + \rho_{I,J-1}}{2} \right) v_{I,j} \right] \quad (2.5.14)
\end{aligned}$$

$$\begin{aligned}
F_n = (\rho v)_n &= \frac{F_{I,j} + F_{I,j+1}}{2} = \\
&= \frac{1}{2} \left[\left(\frac{\rho_{I,J} + \rho_{I,J-1}}{2} \right) v_{I,j} + \left(\frac{\rho_{I,J+1} + \rho_{I,J}}{2} \right) v_{I,j+1} \right] \quad (2.5.15)
\end{aligned}$$

$$D_w = \frac{\Gamma_{I-1,J-1} + \Gamma_{I,J-1} + \Gamma_{I-1,J} + \Gamma_{I,J}}{4(x_I - x_{I-1})} \quad (2.5.16)$$

$$D_e = \frac{\Gamma_{I,J-1} + \Gamma_{I+1,J-1} + \Gamma_{I,J} + \Gamma_{I+1,J}}{4(x_{I+1} - x_I)} \quad (2.5.17)$$

$$D_s = \frac{\Gamma_{I,J-1}}{y_i - y_{i-1}} \quad (2.5.18)$$

$$D_n = \frac{\Gamma_{I,J}}{y_{i+1} - y_i} \quad (2.5.19)$$

Again these values are calculated from the u and v velocities components of the last iteration.

2.5.2 The SIMPLE Algorithm

The algorithm uses a guess-and-correct procedure. The derivation of the building blocks equations of the algorithm will be presented below, and a summarizing assembly is provided in the next subsection.

Derivation of the Gessed Velocities Expressions

The calculation of the gessed velocities fields: u^* and v^* based on the gessed pressure value are carried out using the equations:

$$a_{i,J}u_{i,J}^* = \sum a_{nb}u_{nb}^* + (P_{I-1,J}^* - P_{I,J}^*) A_{i,J} + b_{i,J} \quad (2.5.20)$$

$$a_{I,j}v_{I,j}^* = \sum a_{nb}v_{nb}^* + (P_{I,J-1}^* - P_{I,J}^*) A_{i,J} + b_{I,j} \quad (2.5.21)$$

Derivation of the Pressure Correction Formula

First, the pressure, and velocities corrections are defined as follows:

$$p = p^* + p' \quad (2.5.22)$$

$$u = u^* + u' \quad (2.5.23)$$

$$v = v^* + v' \quad (2.5.24)$$

By substituting the correct pressure field into the discretized momentum equations, we get equations linking the correct pressure field to the correct velocities fields (u, v), those are equations (2.5.2) and (2.5.11). By subtracting equation 2.5.20, 2.7.2 (gessed fields) from equations 2.5.2, 2.5.11 (correct field), and using equations 2.5.22-2.5.24 for writing the resulting equations in terms of the pressure, and velocities corrections $p', u',$ and v' , we get:

$$a_{i,J}u'_{i,J} = \sum a_{nb}u'_{nb} - (p'_{I-1,J} - p'_{I,J}) A_{i,J} \quad (2.5.25)$$

$$a_{i,J}v'_{J,i} = \sum a_{nb}v'_{nb} - (p'_{I,J-1} - p'_{I,J}) A_{I,j} \quad (2.5.26)$$

The SIMPLE algorithm as introduced in [6], use an approximation at that stage: $\sum a_{nb}u'_{nb}$, $\sum a_{nb}v'_{nb}$ are set to be zero. This is the approximation that allows the discretized equation to be semi-implicit. We obtain:

$$u'_{i,J} = d_{i,J} (p'_{I-1,J} - p'_{I,J}) \quad (2.5.27)$$

$$v'_{I,j} = d_{I,j} (p'_{I,J-1} - p'_{I,J}) \quad (2.5.28)$$

Where $d_{i,J} = A_{i,J}/a_{i,J}$ and $d_{I,j} = A_{I,j}/a_{I,j}$

Thus the velocities correction formulae 2.5.23 and 2.5.24 become:

$$u_{i,J} = u_{i,J}^* + d_{i,J} (p'_{I-1,J} - p'_{I,J}) \quad (2.5.29)$$

$$v_{I,j} = v_{I,j}^* + d_{I,j} (p'_{I,J-1} - p'_{I,J}) \quad (2.5.30)$$

Writing those equations for $u_{i+1,J}$ and $v_{I,j+1}$, we get:

$$u_{i+1,J} = u_{i+1,J}^* + d_{i+1,J} (p'_{I,J} - p'_{I+1,J}) \quad (2.5.31)$$

$$v_{I,j+1} = v_{I,j+1}^* + d_{I,j+1} (p'_{I,J} - p'_{I,J+1}) \quad (2.5.32)$$

Where $d_{i+1,J} = A_{i+1,J}/a_{i+1,J}$ and $d_{I,j+1} = A_{I,j+1}/a_{I,j+1}$

The second part of the derivation of the pressure correction formula is the application of the continuity constraint for the velocity field. In the discretized form, continuity is expressed for the scalar control volume (control volume of node p in figure 2.1) by:

$$\left[(\rho u A)_{i+1,J} - (\rho u A)_{i,J} \right] + \left[(\rho v A)_{I,j+1} - (\rho v A)_{I,j} \right] = 0 \quad (2.5.33)$$

By substituting the corrected velocities of equations 2.5.29-2.7.1, rearranging, and identifying the coefficients of p' , we get:

$$a_{I,J}p'_{I,J} = a_{I+1,J}p'_{I+1,J} + a_{I-1,J}p'_{I-1,J} + a_{I,J+1}p'_{I,J+1} + a_{I,J-1}p'_{I,J-1} + b'_{I,J} \quad (2.5.34)$$

Where $a_{I,J} = a_{I+1,J} + a_{I-1,J} + a_{I,J+1} + a_{I,J-1}$. The other coefficients are given as:

$a_{I+1,J}$	$a_{I-1,J}$	$a_{I,J+1}$	$a_{I,J-1}$	$b'_{I,J}$
$(\rho dA)_{i+1,J}$	$(\rho dA)_{i,J}$	$(\rho dA)_{I,j+1}$	$(\rho dA)_{I,j}$	$(\rho u^* A)_{i,J}$ -
				$(\rho u^* A)_{i+1,J}$ +
				$(\rho v^* A)_{I,j}$ -
				$(\rho v^* A)_{I,j+1}$

The equation 2.5.34 represents the discretized continuity equation as an equation for pressure correction p' . The term b' can be used as a convergence criterion because it represents the continuity imbalance arising from the incorrect velocity field u^* , v^* . The approximation introduced in the derivation does not affect the final result because the velocities corrections will converge to zero.

To avoid the divergence of the pressure correction equation, an under-relaxation is introduced during the iterative process [6], thus, the following formula can be used:

$$p^{new} = p^* + \alpha_p p' \quad (2.5.35)$$

Similarly, an under-relaxation step is introduced for the velocity's components:

$$u^{new} = \alpha_u u + (1 - \alpha_u) u^{(n-1)} \quad (2.5.36)$$

$$v^{new} = \alpha_v v + (1 - \alpha_v) v^{(n-1)} \quad (2.5.37)$$

Where α_u and α_v are the u - and v -velocity under-relaxation factors. The aim is to take the pressure and velocities under relaxation factor α_p , α_u , and α_v between 0 and 1, in a sense that the added fraction will be small enough to stabilize the computations, and large enough to accelerate the convergence. Unfortunately, the optimum choice must be sought on a case-by-case basis.

2.5.3 The Assembly of the Complete Method

The CFD procedure that employs the SIMPLE algorithm is given by the sequence of operations given below:

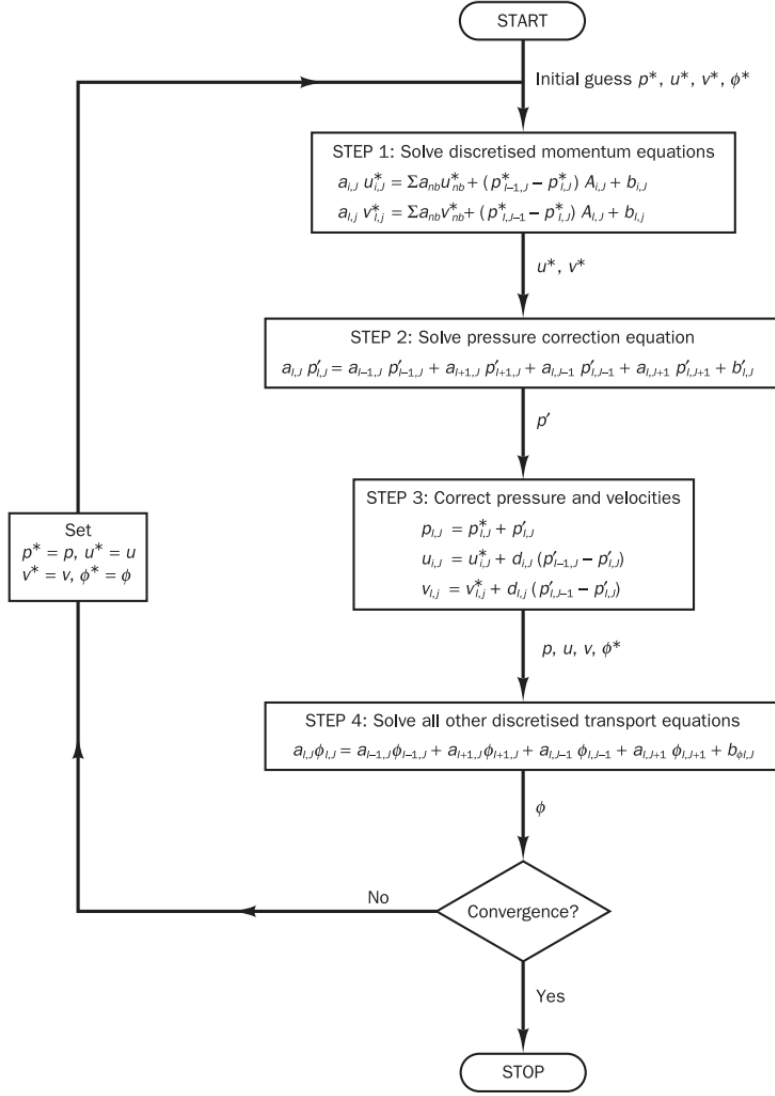


Figure 2.3: The SIMPLE algorithm.[6]

2.6 Boundary Conditions

The solution of the discrete form of the governing equations needs proper implementation of the boundary conditions and initial conditions. Although the initialization of the flow field variables is straightforward and is not critical for the solution results, the boundary conditions on the other hand need more attention. The implementation of the boundary condition begins with the choice

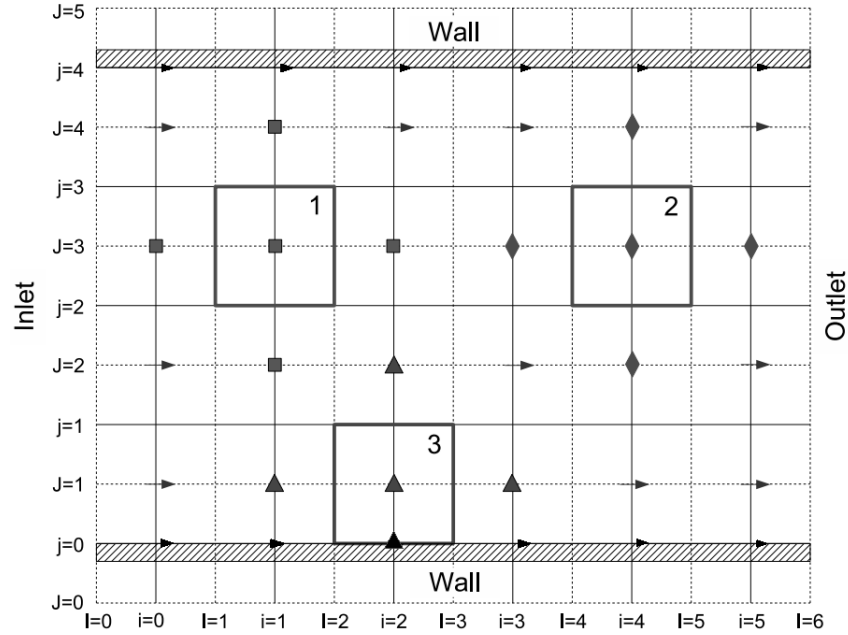


Figure 2.4: Three emphasized u momentum cells in the vicinity of the inlet, outlet, and wall boundaries in a pipe flow configuration. [7]

of the appropriate boundary condition type, which correctly models the physical world. Then, the discrete form of the governing equations for the boundary node must be treated separately to consider the boundary condition. In this subsection, the most important boundary conditions, namely: inlet, outlet, and wall are discussed in detail.

2.6.1 Inlet

At the inlet, u and v have to be prescribed. In our study, we are using a staggered grid system, which implies that the three flow variable are stored in different locations. We will illustrate our grid system by 3 figures: 2.4, 2.5, and 2.6.

Figure 2.4 represents the location where u velocities are stored: Non-boundary locations are represented by arrows, and three boundary nodes are highlighted. The first (labeled 1) is an inlet node it is represented as its neighboring nodes by solid squares; the second node type (labeled 2) is an outlet node, it is represented as its neighboring nodes by solid diamonds, the third node type is adjacent to the wall it is represented as for its neighboring nodes by solid triangles. The figures 2.5, and 2.6 are drawn for v and p nodes respectively following the same principle. The nodes laying in along the physical boundary (i.e. $i=0$) are used

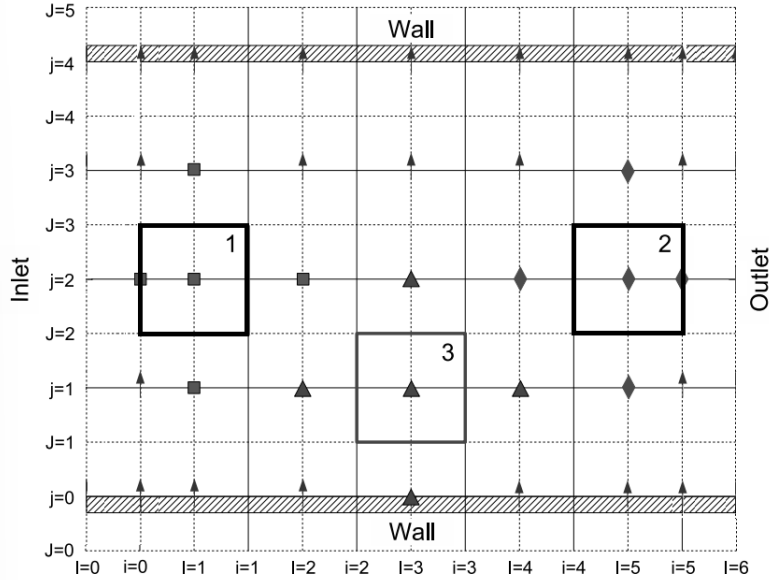


Figure 2.5: Three emphasized v momentum cells in the vicinity of the inlet, outlet, and wall boundaries in a pipe flow configuration. [7]

to store the inlet values of the flow variables. It is also important to note that for the v grid we have placed the inlet and outlet columns at columns $i=1$ and $i=5$ respectively, i.e. from a distance $dx/2$ from the succeeding and preceding columns respectively. This is done to make the boundary conditions applied directly to the grid nodes. It is clear that since the values of the boundary nodes will not change as the solution proceed, no velocity correction will be applied to these nodes i.e:

$$u_{i=0,J} = u_{i=0,J}^* \quad (2.6.1)$$

2.6.2 Outlet

The region far enough downstream does not present any disturbance in the flow fields. This physical fact is intuitively modeled by a zero gradient in the direction of the flow for u and v velocities. The condition is known as a Neuman condition (because it is set on the gradient of the variable). Numerically, this boundary condition is implemented by setting the variable laying in the exit column equal to the values of the variables of the preceding column. i.e. the diamond cells at the extreme right of the domain will take the value of the next iteration. The following expression summarizes the explained procedure:

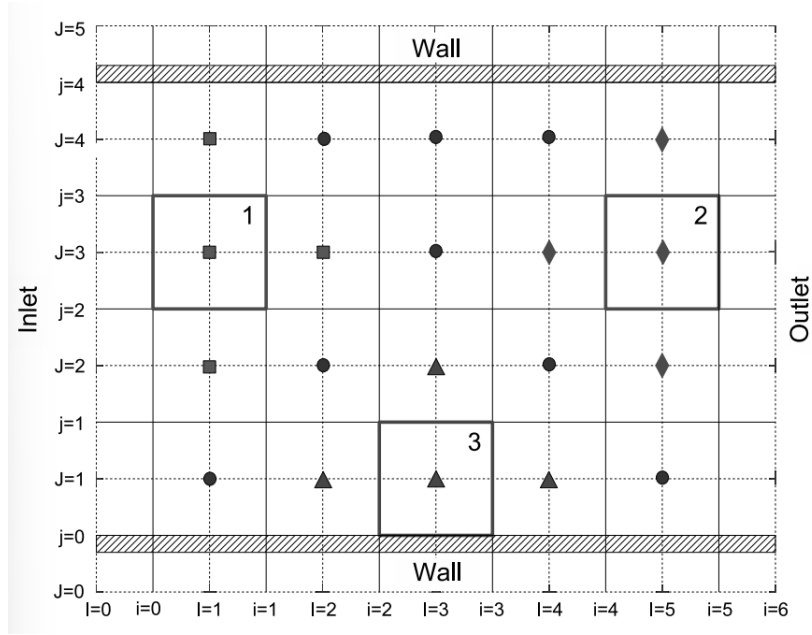


Figure 2.6: Three emphasized p momentum cells in the vicinity of the inlet, outlet, and wall boundaries in a pipe flow configuration. [7]

$$\begin{aligned}
 u_{i=5,J} &= u_{i=4,J} \\
 v_{I=6,j} &= v_{I=5,j} \\
 p_{I=6,j} &= p_{I=5,j}
 \end{aligned}
 \tag{2.6.2}$$

The choice of the dimensions of the calculation's domain has to be valid with the boundary condition. In another word, the domain must be set so that the exit will lay in the fully developed region far away from any disturbances. The pressure correction solution is solved with a zero gradient boundary condition at the inlet. Moreover, since the outlet node is directly deduced from the preceding nodes, no velocity correction is required.

2.6.3 Wall

The wall boundary is of special interest in fluid applications, particularly in viscous flows where the physical property of adherence, or more commonly named no-slip condition is observed [7]. This property is modeled by setting all the velocity components at the boundary equal to zero. Here, the added rows of u -velocity on the wall conditions become useful, because no special treatment is needed for the triangle cell labeled 3, we will just set the adjacent south cell equal to zero and take into account that the distance of this cell is $dy/2$ in the diffusive coefficient. For the v , we only set the south adjacent cell equal to zero.

2.7 Iterative Solver: Gauss Seidel

In our code, we have used the Gauss-Seidel method which is a point-wise stepping method. The interesting feature of this algorithm is that it is unconditionally stable for the coefficients matrix because this latter is always positive and definite. The algorithm for the Gauss-Seidel method is given as follows [5]:

Algorithm 1: Gauss Seidel Algorithm

- 1 Guess values of ϕ at all nodes. We denote these values as $\phi(0)$. The guessed solution for Dirichlet boundary conditions must be set equal to the prescribed value.
 - 2 Set $\phi^{n+1} = \phi^n$ and apply the Gauss-Seidel update formula, Eq. 2.7.1 for interior nodes. Special treatment must be done to non-Dirichlet BCs (i.e. calculation of the link coefficients based on the nodal equation at that boundary)
 - 3 Compute the residuals using $\phi(n+1)$, and then compute the norm for monitoring the overall residuals.
 - 4 Monitor convergence, by comparing the residuals to the tolerance. If it is achieved go to Step 5. If not, then go to Step 2.
 - 5 Quit the loop and post-process results.
-

The Gauss-Seidel update formula for interior nodes is given by:

$$\phi_{i,j}^{(n+1)} = \frac{S_{i,j} - a_E \phi_{i+1,j}^{(n)} - a_W \phi_{i-1,j}^{(n+1)} - a_N \phi_{i,j+1}^{(n)} - a_S \phi_{i,j-1}^{(n+1)}}{a_P} \quad (2.7.1)$$

Where the link coefficients are given by:

$$\begin{aligned} a_k = a_P &= - \left(\frac{2}{(\Delta x)^2} + \frac{2}{(\Delta y)^2} \right) \\ a_{k+1} = a_E &= \frac{1}{(\Delta x)^2} \\ a_{k-1} = a_W &= \frac{1}{(\Delta x)^2} \\ a_{k+N} = a_N &= \frac{1}{(\Delta y)^2} \\ a_{k-N} = a_S &= \frac{1}{(\Delta y)^2} \end{aligned} \quad (2.7.2)$$

It is important to note that it is not necessary to store values of ϕ for both previous and current iterations. The same array may be used to store both and the update formula will take the new values as soon as they become available.

2.8 Conclusion

In the present chapter, we have built the foundation of our CFD code. For each component of the numerical solution, we have chosen one method. For the discretization, the finite volume method was chosen, and the geometry was

discretized uniformly using a staggered grid configuration. For the solution technique, the SIMPLE algorithm was selected, and finally the Gauss-Seidel method is implemented for the iterative solver.

Chapter 3

The Code's Implementation

3.1 Introduction

So far we have presented the building blocks of our software, but we haven't talked yet about the implementation of those blocs and their interactions, this will be the subject of the present chapter. In addition, the other part of the software will be presented here, which is the graphical user interface (GUI). Therefore, this chapter will be divided into two sections the first one is about the computation code, named *ico_ns_solver*, and the second is about the Graphical user interface.

3.2 The Computation Code: *ico_ns_solver*

3.2.1 Phases of Implementation

The code development was divided into multiple stages, where at each stage a piece of complexity is added. This method is more suited for writing long codes, it allows the validation of each piece of the aggregate separately, thus facilitating debugging and maintenance. We will talk more about those stages in the next sub-section:

One-Dimension Transport Equation

The first stage consisted of solving the general one-dimension transport equation. The results given in example 5.1 of [6], and the analytical solution was used for the code validation. In this stage, the structure of the software was clarified, and some parts of the code were validated, for instance: The linear solver, and the upwind scheme (for 1d).

Pressure Coupling

The second stage consisted of pressure coupling; unlike the previous stage, here the velocity field is unknown. For that purpose, the SIMPLE algorithm was implemented to solve the planar two-dimensional nozzle problem presented in example 6.2 of [6]. Although the governing equations of the treated problem are different from the Navier-Stokes equations, our purpose here was to understand the logic of the SIMPLE algorithm and to be more comfortable when treating our actual 2-dimensional Navier-Stokes problem.

Two-dimension

In the third stage, the 2-dimensional Navier-Stokes equations were treated, this task was considerably facilitated by taking advantage of the previous stages.

3.2.2 Code Structure

The computation code was written in c++, using standard libraries and the jsoncpp library for manipulating a JSON file. The JSON file is used as the

data transfer main between the GUI and the computation code, The project structure is as follows:

CFD1.1.cpp: is the main() function file, it comprises a section for input data reading, and an outer iteration loop (SIMPLE iteration).

SIMPLE_2d.h: a header file, which executes one outer iteration (or SIMPLE iteration), by taking input data from the main file and returning the new values of u, v velocities, the pressure field, and the residual for that iteration.

util_2d_NS.h a header file containing all the function used in one iteration of the SIMPLE algorithm. For example, the construction of linear system u, v-momentum equations and the pressure correction is done by the functions `conv_diff2d_momentum_eqs_u`, `conv_diff_2d_momentum_eqs_u`, and `pressure_correction_eqs_2d` respectively. The iterative solution of that system is done by the function `gauss_seidel_2d`. The update operation and mass conservation residual are done by the functions `update_2d`, and `mass_conservation` respectively.

utilities.h: contains general utilities like functions for arrays printing in the screen and writing them in files.

nutril.h: this is part of the Numerical Recipe book's [16] library . It is used for the dynamic allocation and de-allocation of matrix and vector throughout the code.

3.2.3 The Code's Working Principle

The code takes the input data from a JSON file (named `solver_setup.json`) and reads the following input values:

`rho`: The fluid density.

`mu`: The dynamic viscosity of the fluid.

`nx`: The streamwise number of cells.

`ny`: The number of cells perpendicular to the streamwise direction.

`length`: The streamwise domain length.

`breadth`: The distance between horizontal boundaries.

`inlet_u`: The u velocity at the inlet boundary.

`inlet_v`: The v velocity at the inlet boundary.

`top_u`, `top_v` : The u, v velocity at the top boundary.

`bottom_u`, `bottom_v`: The u, v velocity at the bottom boundary.

`p_relax`, `u_relax`, `v_relax`: p, u, v under relaxation factors.

`outer_itr`: Number of outer iterations.

outer_tol: Stop criterion of outer iterations.

gs_itr: Gauss-Seidel solver number of iterations.

gs_tol: Gauss-Seidel solver tolerance.

The *main()* function located in the `CFD1.1.cpp` file feeds the `SIMPLE_2d_iteration` function with the input data by calling it in an outer loop and monitors its residuals at every iteration.

The `SIMPLE_2d_iteration` function contains a succession of function calls similar to the SIMPLE procedure, each function has an essential role in the SIMPLE algorithm for instance: The `mumuntum_eqs_u` call will construct the linear system for u component. Then, `Gauss_seidel_2d` call will solve the previous system. And the same is repeated for the v component and the pressure correction equation.

In the end, the mass imbalance is calculated (for convergence monitoring) and we update the variable fields by calling the following functions respectively: `mass_conservation`, and `update_2d`. All the used functions in the `SIMPLE_2d_iteration` function are present in the `util_2d_NS_Solver.h` file.

After the program exit the outer loop in the `CFD1.1.cpp` file, either by exceeding the iteration number, or if the residuals fall below the tolerance, the results are saved in a form of CSV files, where the:

P.csv: Contains the pressure distribution.

U.csv: Contains the u-velocity distribution.

V.csv: Contains the v-velocity distribution.

3.3 The Graphical User Interface and its Features

The GUI is a principal component of the present project, that is because it is designed for educational purposes. The GUI is aimed to be simple and suitable for an untrained user. It should contain input fields and a graphical window for basic post-processing plots. To that end, we have chosen the Tkinter Python library for developing the CFDENP GUI. In addition to Tkinter, some other python libraries were used for manipulating data and plotting, for instance, the JSON library was used for reading/ writing JSON files, the NumPy, and pandas libraries for handling and basic manipulation of data, and the matplotlib library for plotting different graphics.

Once the program is executed, a command line window and the principle CFDENP window (shown in Figure 3.1) will open. The command line is used for printing some messages from the computation code, and the principal widows contain all the other functionalities.

The left half of the window consists of the different input fields. The user should enter all the inputs manually, except for the fluid properties section,

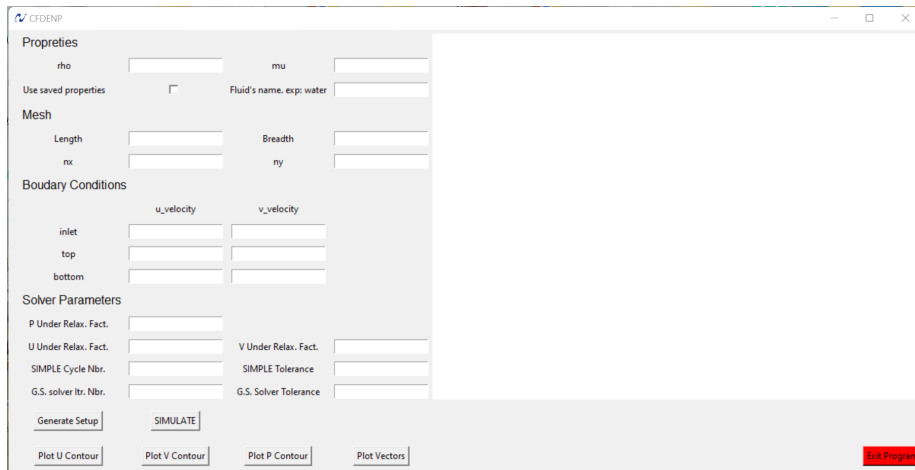


Figure 3.1: The principal window of CFDENP

where the program has a JSON file named “properties_db” that plays the role of a fluid properties database. The user can modify this file (with any text editor) to include/delete any element where he just needs to respect the file structure given as the example below:

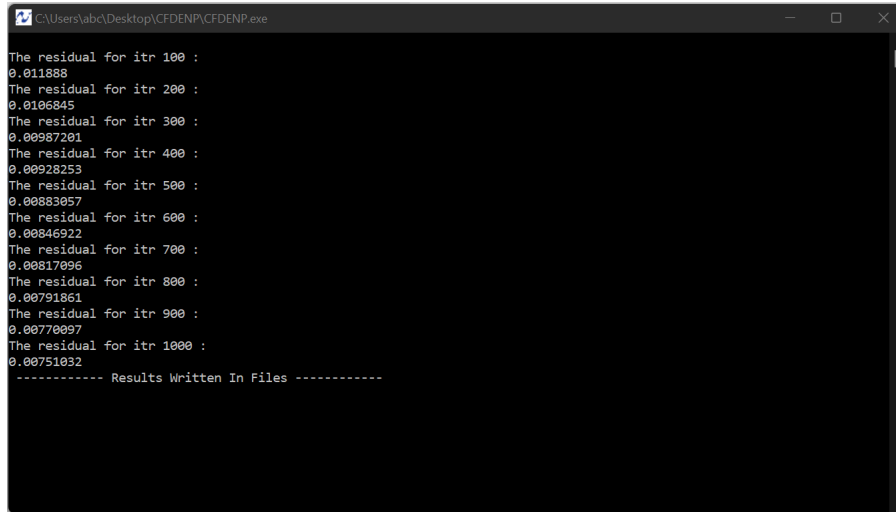
```
{
  "water":
  {
    "rho": 997.05,
    "mu": 0.89e-3
  }
,
  "air":
  {
    "rho": 1.1845,
    "mu": 1.8444e-5
  }
}
```

After an element is added to the database, it can be accessed by typing its name in the “Fluid’s name” input field after checking the “use saved properties” check button.

After filling in all the inputs, the user needs to press the “Generate Setup” button to create the setup file (discussed in section 3.2.3). That file remains saved and can be reused for running other simulations with the same parameters.

Next, the user needs to press the “SIMULATE” button, here the code executes the computation program named ico_ns_solver.exe, which will plot the

residuals and other messages in the Command Line Window that automatically opens with the GUI.



```
CFDENP.exe
The residual for itr 100 :
0.011888
The residual for itr 200 :
0.0106845
The residual for itr 300 :
0.00987201
The residual for itr 400 :
0.00928253
The residual for itr 500 :
0.00883057
The residual for itr 600 :
0.00846922
The residual for itr 700 :
0.00817096
The residual for itr 800 :
0.00791861
The residual for itr 900 :
0.00770097
The residual for itr 1000 :
0.00751032
----- Results Written In Files -----
```

Figure 3.2: Residuals printing in the command line window for a given simulation.

After the simulation finishes, the user can post-process the results by plotting u , v , and pressure contours, which are directly done via the corresponding buttons, also, the user can plot the velocities vectors by pressing the plot vectors button.

A toolbar is positioned below the plotting window. It allows the user to perform basic operations on the graphs. It contains the following buttons:

- The home button: it allows the user to return to the initial view after zooming or shifting the graph.
- Back to previous view/ Forward to next view buttons: They allow the user to return to a given state after modifying the graph.
- Graph displacement button: It allows the user to shift the graph in the axis frame.
- Zoom to rectangle button: It allows the user to zoom into a sub-rectangular area in the graph.
- Configure subplots button: It allows the user to modify the axis scale.
- Save the figure button: It allows the user to save the plots in a PNG format.

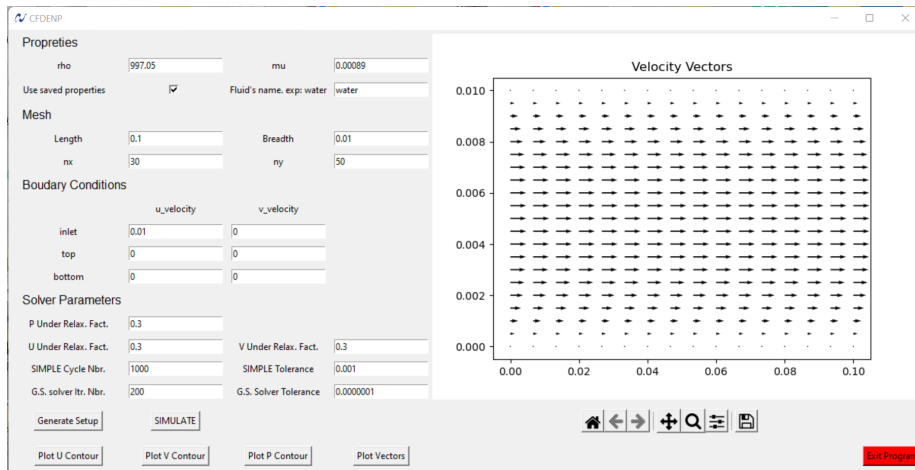


Figure 3.3: Visualization of the vector plot of a flow field in a pipe, $Re \approx 1100$.

3.4 Conclusion

This chapter discussed the code structure, the working principle, and the general organization of the code. That information is important not only for the users of the code but also for any person wanting to resume this work so that he can easily understand the code, and follow the same conventions.

Chapter 4

Results And Discussion

4.1 Introduction

Our project is divided into two sub-project, the first is (i) the computation code, and the second is (ii) the implementation of a Graphical User Interface (GUI). The GUI was tested and works properly, but the computation code, although complete has an issue that we can not fix due to the time constraint. However, many elements of the code were tested separately and worked as expected, therefore the future contributors willing to resume this work will be able to build upon the present code. The comments and coding style used to develop the code were also made to make it readable and easy to understand by another person.

In this section, we will present the results of our codes at a different stage of the implementation and compare them to the simulations made using OpenFoam. Due to the erroneous flow field obtained by the final developed solver (named *ico_ns_solver*), we will not present many comparisons of the obtained results (at the third stage of the development), since a complete verification is not relevant only for a solver validation.

4.2 Validation of the First Stage: Convection-diffusion 1D

In this stage, we have considered a one-dimensional convection-diffusion problem. Where a property ϕ is transported through convection and diffusion through the one-dimensional domain sketched in Figure 4.1. The governing equation is given by equation 4.2.1. And the boundary conditions are $\phi_0 = 1$ at $x = 0$ and $\phi_L = 0$ at $x = L$, finally, the velocity is constant throughout the domain and equal to $u = 0.1m/s$. Using five equally spaced cells and the central differencing scheme for convection and diffusion we have obtained the results presented in Figure 4.2.

$$\frac{d}{dx}(\rho u \phi) = \frac{d}{dx} \left(\Gamma \frac{d\phi}{dx} \right) \quad (4.2.1)$$

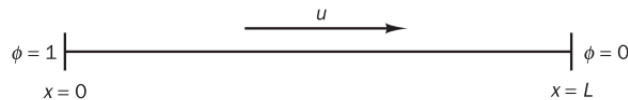


Figure 4.1: A schematic figure of the treated convection-diffusion problem. [6]

From figure 4.2 we see that the results are in full agreement, so we can say that our code for the one-dimensional convection-diffusion equation is validated. From this validation, we have ensured that the implementation of the central difference scheme and the linear solver is correct.

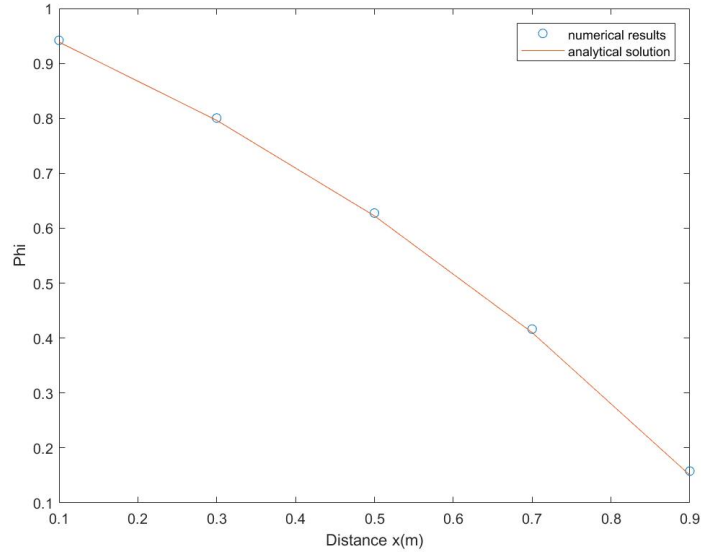


Figure 4.2: Comparison of the the numerical result with the analytical solution for the transported property for a convection-diffusion problem.

4.3 Validation of the Second Stage: 1D SIMPLE

In this stage, we have considered the frictionless, incompressible flow through a planar, converging nozzle represented in figure 4.3. For making the problem exhibit the same pressure-velocity coupling issues encountered in the Navier-Stokes equations, we assume that the flow is unidirectional, and the flow variables are constant throughout every cross-section perpendicular to the nozzle symmetry axis.

We have tested our code by solving this problem and checking the accuracy of the computed solution against the well-known Bernoulli equation, the results represented in figure 4.4 were obtained. The presented results show an acceptable correlation after only 19 iterations. For a large number of grid points, we have found that the computed solution converges to the analytical solution given by the Bernoulli equation. From the above elementary validation, we can conclude that the SIMPLE outer loop is correctly implemented inside the code.

4.4 Validation of the Third Stage: The Complete Algorithm

In this stage, we are dealing with the complete solver, i.e. 2-dimensional incompressible flow solver.

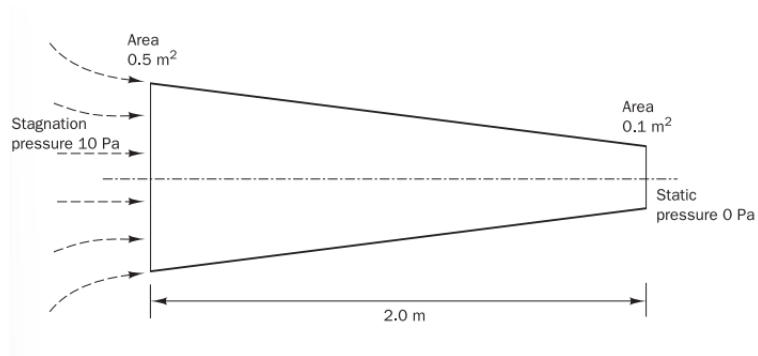


Figure 4.3: A planar two-dimensional nozzle. [6]

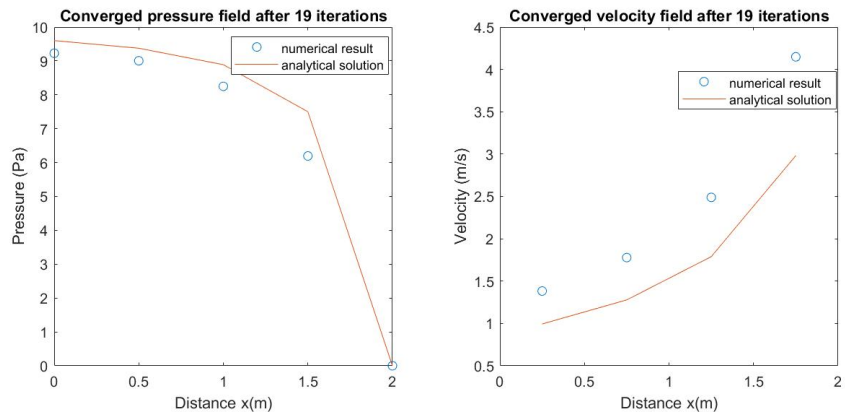


Figure 4.4: comparison of the converged pressure and velocity field after 19 iterations to the analytical solution for the nozzle flow case.

4.4.1 Pipe Flow: Entry Length Plots

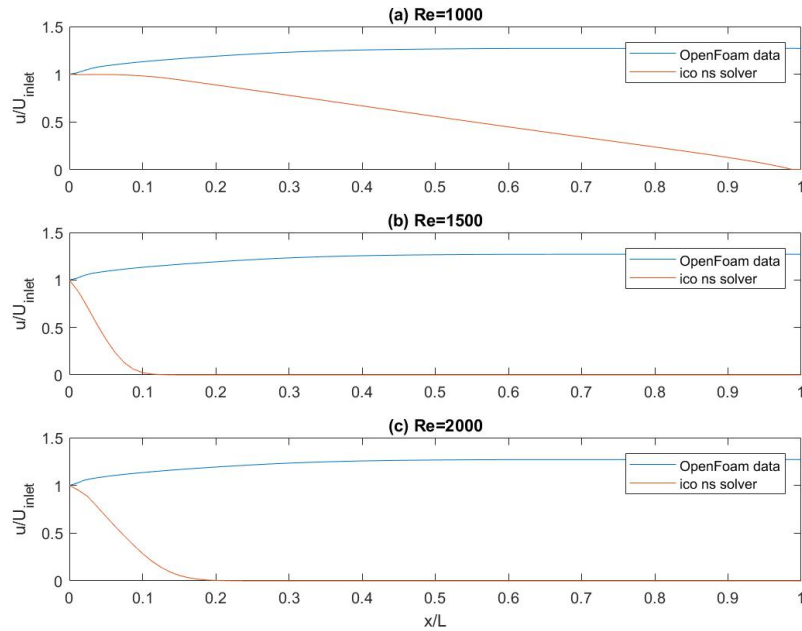


Figure 4.5: A comparative figure for a pipe flow where plots of the normalized u velocity component at the pipe centerline versus the normalized distance from the inlet for three Re numbers: (a) $Re=1000$, (b) $Re=1500$, (c) $Re=2000$ were presented

Figure 4.5 shows that the results of the two simulations do not correlate except at the inlet, where both plots start from the inlet velocity. Next to the inlet the red plot (*ico_ns_solver* results) drops to zero. this drop is sharper for larger Re as it can be shown if figures (b), and (c).

The comparative study between various Re numbers suggests that the issue does not depend on the fluid properties or the inlet velocity, but it is an issue in the code.

4.4.2 Pipe Flow: Contours Plots

Figure 4.6 shows the contour plots for the computed flow field, namely: pressure P , and the two velocity components u , and v . Those results were obtained using *ico_ns_solver* code for three different Reynolds Numbers, $Re=1000$, $Re=1500$, and $Re=2000$.

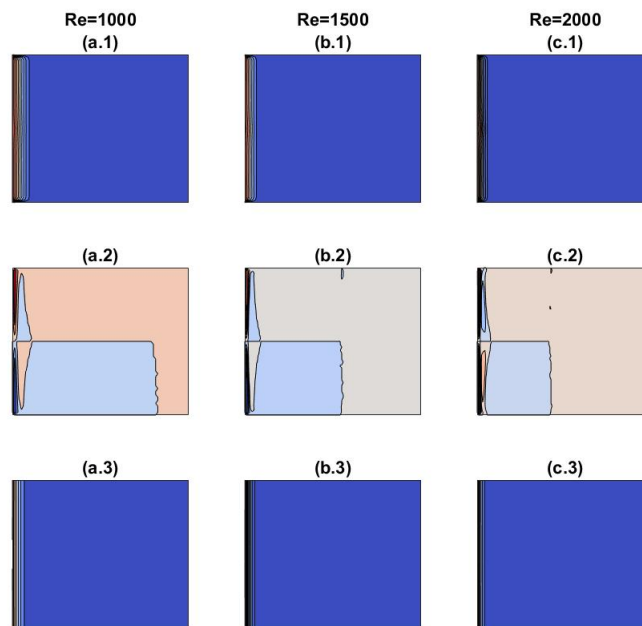


Figure 4.6: Contours plots for Pressure, u velocity, and v velocity fields for three Re number: (a) Re=1000, (b) Re=1500, (c) Re=2000. The plots are arranged so that the first row (top), second and third represent: u-velocities, v-velocities, and Pressure contours respectively

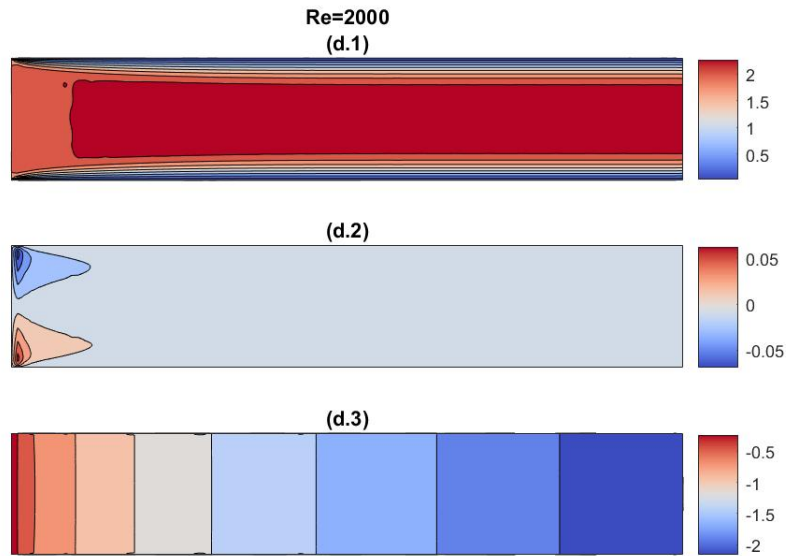


Figure 4.7: Contours plots for Pressure, u velocity, and v velocity fields for $Re=2000$ in a pipe flow simulation done using OpenFoam.

Those results did not pass the sanity check. Here, the flow is stagnating next to the inlet, and at the rest of the pipe, the fluid does not flow.

As it is well known in a pipe flow we get a developed u velocity field after an entry length, which means that due to shear forces the velocity profile changes next to the inlet and remains constant after a distance from the pipe's entry called *The Entry Length*. The resulting profile is a parabolic profile which has a maximum value in the pipe's centerline (slightly greater than the inlet velocity), and a null velocity due to the boundary layer effects at solid walls, this is shown in Figure 4.7 (d.1) obtained using OpenFoam (*icoFoam* solver). For the v velocity field, it should be almost null everywhere in the domain since the inlet has a pure horizontal velocity as shown in Figure 4.7 (d.2). For the pressure field, the right result will be a decreasing pressure gradient in the x direction and constant in the y direction, as it is presented in Figure 4.7 (d.3).

4.5 Conclusion

The present project has two separate sub-projects, the first is the GUI, and the second is the computation code *ico_ns_solver*.

The GUI was tested and works as expected. However, the results of the computation code were not validated yet due to an error in the code. we expect

that the error is in the implementation: of the outlet boundary condition; or the matrix construction.

Although the computational solver is not validated yet, the project can still be useful since the GUI is separated from the solver, and one needs just an executable file (after adding some line of code in the solver for reading the *setup_file* and writing the results in a CSV format) of a validated solver to be able to use the GUI to simulate and post-process study cases.

General Conclusion

Conclusion

This work aimed to develop a basic CFD code with a graphical user interface for educational purposes. The computational code called *ico.ns* solver is based on the SIMPLE algorithm that uses the FVM discretization on a staggered grid arrangement, and the Gauss-Seidel algorithm as a linear solver. The code is implemented in C++, where each element of the code is separated from the main body and used by a function call, this clean structure makes the code readable and easy to extend without the owner's help.

The Graphical User Interface is the second part of this project, it was written in python using the Tkinter library. It facilitates feeding the simulation data into the code, executing the simulation, and performing basic post-processing options. The GUI is independent of the solver and can be used with another solver with minor modification, for data reading and writing.

In the first part of the project, although all the solver parts were accomplished, the solver results were not valid due to an unexpected bug in the code. This problem could not be fixed before the graduation date due to the encountered technical difficulties. However, the second part of the project, namely the GUI was tested and validated. It is worth noting that an interesting feature is that the GUI can work with any other solver with minor modifications in the source code of this later.

Future Work

As stated earlier, this project is an initiation, that invites every interested student to add his contribution to this Open-Source code. Although the targeted outcome was not achieved, this project -as it was initially intended- remains open to further development. We suggest that this future development will be targeting the following features:

- Implementing an optimized Incompressible Flow solver.
- Implementing the Energy Equation in a new solver.
- Model for Turbulent flow (RANS).

- Three-dimensional flow.
- Multiphase Flow.

Bibliography

- [1] F. Moukalled, L. Mangani, and M. Darwish, *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM and Matlab*. Springer Publishing Company, Incorporated, 1st ed., 2015.
- [2] J. H. F. . P. . L. Street, *Computational Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg NewYork, 3th ed., 2002.
- [3] J. H. F. . P. . L. Street, *Computational Methods for Fluid Dynamics*. Springer Publishing Company, Incorporated, 4th ed., 2020.
- [4] A. Silva and J. Barata, “Grid generation with boundary point distribution control on heterogeneous parallel architectures,” *International Review of Aerospace Engineering*, vol. 4, p. 48, 02 2011.
- [5] S. Mazumder, *Numerical Methods for Partial Differential Equations Finite Difference and Finite Volume Methods*. Elsevier Inc, 2016.
- [6] W. M. H. K. Versteeg, *An Introduction to Computational Fluid Dynamics THE FINITE VOLUME METHOD, Secon edition*. Pearson Education, England, 2007.
- [7] G. N. IBOGLU, “Development of an educational cfd software for two dimensional incompressible flows,” 2007.
- [8] “Clay mathematics institute web page.” Last accessed 02 August 2022.
- [9] J. D. Anderson, *Computational Fluid Dynamics: The Basics with Applications*. New York: McGraw-Hill, 1995.
- [10] *Computational Techniques for Fluid Dynamics*. Springer-Verlag, Berlin, 1988.
- [11] C. A. Fletcher, “Computational techniques for fluid dynamics,” *Springer-Verlag, Berlin*, vol. vol. II: Specific Techniques for Different Flow Categories, 1988.
- [12] “Overhead (computing).” Last accessed 26 October 2022.

- [13] “Ncfmf fluid mechanics films.” Last accessed 04 September 2022.
- [14] “Hunter rouse (ihr) fluid mechanics films.” Last accessed 04 September 2022.
- [15] D. Pines, “Using computational fluid dynamics to excite undergraduate students about fluid mechanics,” *ASEE Annual Conference and Exposition, Session: 1055*, 2004.
- [16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*. Cambridge, USA: Cambridge University Press, second ed., 1992.

The Code's repository

The Developed codes can be accessed via the following GitHub repository:
<https://github.com/smdmohamed/CFDENP.git>