UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Study for the numerical resolution of conservation equations of mass, momentum and energy and a first approach to large problems using computational performance enhancement techniques

Document:
Report

Author:
Pau Romeu Llordella

Director / Co-director:
Carlos David Pérez Segarra
Asensio Oliva Llena
Àdel Alsalti Baldellou

Degree:
Bachelor's Degree in Aerospace Technologies Engineering

Examination session:
Spring, 2021

BACHELOR'S FINAL THESIS

Abstract

The conservation equations of mass, momentum and energy are of great importance in comprehending a physical system. From these equations derives, for example, the Navier Stokes system, one of the most studied in the field of physics to determine the motion and properties of any fluid.

The great complexity of these systems of non-linear partial differential equations and the lack of analytical solutions has forced scientists and engineers to focus their study using numerical methods. That is, to convert a physical reality of a continuous environment into a discrete system that can be efficiently processed by a computer.

The first part of this project presents a set of academic thermophysical problems of heat transfer and pressure and velocity distribution in a fluid. The equations governing these properties, including Navier-Stokes equations, are discretised to be implemented in C++ programs. The results obtained are analysed, contrasted and verified by making use of the existing literature.

The second half of the thesis presents an approach towards solving larger and more complex problems that can describe reality more accurately. Scenarios where the discretisation meshes are of such magnitude that a computer cannot process the result in an acceptable period of time. In these cases, it is necessary to be aware of how calculations are carried out by a computer in order to maximise the use of its resources. Some of the optimisation techniques used today are discussed and the performance improvement in solving real problems is evaluated.

Resum

Les equacions de conservació de la massa, quantitat de moviment i energia són de gran importància a l'hora de comprendre un sistema físic. D'aquestes equacions en deriva, per exemple, el sistema de Navier-Stokes, un dels més estudiats en el camp de la física per entendre el comportament de qualsevol fluid.

La gran complexitat d'aquests sistemes d'equacions en derivades parcials no lineals i la falta de solucions analítiques ha obligat a científics i enginyers a enfocar el seu estudi fent ús de mètodes numèrics. És a dir, convertir una realitat física de medi continu en un sistema discret que pot ser processat eficientment per un ordinador.

En la primera part d'aquest projecte es presenten un seguit de problemes termofísics acadèmics de transmissió de calor i distribucions de pressions i velocitat en un fluid. Les equacions que regeixen el comportament d'aquestes propietats, incloent les equacions de Navier-Stokes, són discretitzades per ser implementades en programes de C++. Els resultats obtinguts són analitzats, contrastats i verificats.

El segon bloc del treball presenta un enfocament cap a la resolució de problemes molt més grans i complexos. Escenaris on les discretitzacions són de tal magnitud que un ordinador no pot processar el resultat en un període de temps acceptable. En aquests casos cal conèixer el funcionament del càlcul en un ordinador per poder maximitzar l'ús dels seus recursos. Es presenten algunes de les tècniques d'optimització emprades avui en dia i es valora la millora de rendiment en resolucions de problemes reals.

Pau Romeu Llordella September 28, 2021

Study: Study for the numerical resolution of conservation equations of mass, momentum and energy and a first approach to large problems using computational performance enhancement techniques

List of contents

List of figures

List of tables

1

Project introduction

This first section presents the points on which the project is based. It introduces the objectives, the scope and the requirements to be met. The motivation for this work and the current state of the art are also presented.

1.1 Aim

The aim of this project is to approach the use of numerical methods to solve the governing equations of heat transfer and computational fluid dynamics (CFD). The purpose is to study and program C++ codes to solve concrete cases involving the resolution of the equations of conservation of mass, momentum and energy.

In a second part, the focus is on the computational techniques used. Different optimisation strategies required when working with real and complex CFD projects where high computational power is required are studied. Different performance-enhancing algorithms are presented and tested with real matrices obtained from real problems.

1.2 Scope

To achieve the aims of the project, the following list presents the activities that need to be carried out. They may be thought of as a to-do list.

- Present and study the analytical expressions that describe the conservation equations.
- Discretise analytical equations so they can be solved numerically.
- Develop C++ program to numerically solve the following problems:
    - Heat conduction between solid bodies.
    - Convection – diffusion effect in fluids
    - Laminar Navier-Stokes equations in a cavity
- Verify and validate all the programs developed making use of literature and results from other researches.
- Present how scientific problem solving is approached from the perspective of the computer and what are the problems that affect its performance.
- Describe and implement in C++ performance-enhancing algorithms
- Test the efficiency of the studied methods and verify that the computation time is reduced.

1.3 Basic requirements

The requirements describe the actions, processes and results the project needs to meet. For the case that concerns this project, the requirements are both for the solutions acquired and for the quality. Not only must the project achieve the appropriate results, but they must also be presented, discussed and concluded correctly. The requirements can be stated as follows.

- **Functional solver of unsteady 2D heat conduction problems.** The project delivery must include a numerical solver implemented in C++ language. Although the solver is presented for a specific problem case, it must be easily adapted to solve any kind of heat conduction problem in two-dimensions under unsteady conditions

- **Functional solver of unsteady 2D convection-diffusion problems.** Similar to the first requirement, a convection-diffusion problems C++ solver must be created. The program has to be capable of solving different problems with little changes in the initial conditions code.
- **Functional solver of Navier-Stokes (NS) simple flow problems.** A NS problems solver using the Fractional Step Method is being developed. This method is being applied to the lid driven cavity problem, a common case to start approaching NS numerical resolution.
- **Program to study computational performance upgrade.** A C++ program has to be created to test the effectiveness of the studied optimisation methods and to check if they reduce the computation time of real problems.
- **Good programming practices.** The C++ program codes developed are all of self-creation and they must be easily readable, modular and reusable. Furthermore, the use of object-oriented programming practices is required.
- **Verification and validation**. The solvers developed must be both verified and validated. The verification is done through tests to prove that the program meets the requirements it had. The validation must certify that the developed test accomplishes the expectations of the project from a more general point of view.
- **Conclusions**. The presentation of some results without their corresponding conclusions cannot be accepted.

1.4 Background and justification

The realization of this project may be perceived as a continuity of the *Gases Dynamics and Heat and Mass Transfer* subject of the bachelor's degree. This subject is a first approach to the study of the three main methods of heat and mass transfer: conduction, convection and radiation. As it is a huge field of engineering and physics, only a small part can be studied during the course of the subject.

Therefore, this project is a great opportunity to continue focusing and studying this branch of science and engineering. More specifically, it is an introduction to the computational numerical methods for solving physical and thermodynamical problems using a wide variety of resolution schemes.

It is also intended to reflect a keen interest in the computational techniques that are used to solve such problems. This study provides an introduction to real optimisation methods used in the world of high-performance computing.

1.4.1 State of the art

During the last decades, the evolution of the engineering world has been closely related to the exponential growth of computing power. Although not many years ago having access to a computer (with extremely poor resources) was only possible for a few people, in these years almost everyone has access to high-performance computers, smartphones and high-speed Internet.

These enormous advances in computing power have made it possible to achieve solutions to many scientific or engineering problems whose analytical solutions have yet to be found and may never be reached. In the old days, this type of problems could only be tackled with experimentation, which used to be a very expensive and inaccurate way of arriving at an approximate solution to a problem.

This project focuses on the study of heat transfer and fluid dynamics. These fields are some of the main beneficiaries of computing power. Analytical solutions are rarely found in this branch or can simply be used to find approximate solutions. Therefore, developing, understanding or optimising the algorithms used to solve problems is of great importance.

An appropriate example is the Navier-Stokes Equations. These nonlinear partial differential equations that describes the motion of fluids do not yet have an accepted analytical solution. Due to the great importance that these equations have in the field of fluid dynamics, numerical and computational methods have become a fundamental part of the study. Such has been its importance that has given rise to the recognition of a new discipline: the computational fluid dynamics, also known as CFD [1].

Researchers working in the field of CFD thus face two major challenges. On the one hand, as its name suggests, its object of study is fluid dynamics and its aim is to understand, predict and calculate how a fluid behaves. On the other hand, they have to adapt their studies to the current computing paradigm. For today's scientists and engineers, it is essential to know how the calculation process works on a computer.

Although computers have evolved exponentially over the last decades, so have the requirements of the studies that are carried out using this tool. Many problems studied today are of large magnitude and require a great deal of computational power. Hence, improving the hardware is not enough and the software needs to be adapted by means of optimisation techniques. Understanding how the CPU works, optimising the use of memory access or parallelising the solving code are some of the strategies that make it possible to deal with large problems.

1.5 Project structure

The first thing presented in the project, in Section 2, is a theoretical introduction on which the problems solved in this report are based. Section 3 deals with numerical methods applied to heat conduction while Section 4 focuses on the equations describing the convection-diffusion of a property in a fluid.

Section 5 introduces the Navier Stokes solution using the Fractional Step Method to solve the lid-driven cavity problem. Section 6 aims to focus on how larger real-world problems would be treated, where importance has to be given to computational optimisation techniques. Finally, in Section 7, the conclusions and the remaining work to be done are exposed.

2

Theoretical approach

The objective of this section is to present the theoretical background of this project. The conservation equations of mass and momentum are presented. The Navier Stokes system is obtained from the momentum formulation. Heat conduction and convection-diffusion are studied as part of the energy conservation equation.

Numerical methods are introduced in a general way. The schemes, the types of discretisation and the different solvers used in the problems of this work are presented.

2.1 Governing equations

In the field of fluid mechanics, the conservation equations determine the state and motion of the properties of matter. The governing principles are the laws of conservation of mass, momentum and energy. The latter can also be understood as the first law of thermodynamics. Furthermore, although it cannot be considered as a conservation law, the second law of thermodynamics must also be taken into account as a constraint. [2]

The three conservation laws mentioned above are a consequence of the **Reynolds Transport Theorem**. This theorem makes it possible to study the velocity of change of a fluid property by studying the flow through a control volume.

To present the Reynolds theorem, any specific scalar property per unit mass ϕ is assumed. Therefore, the quantity of this property Φ in a volume V_t at a time t is:

$$\Phi(\mathrm{t}) = \int_{V_t} \rho\phi \mathrm{d}V \tag{2.1}$$

Now it can be studied how the property changes as it is derived with respect to time. Note that since the property not only depends on time but also depends on its position, the material derivative is used.

$$\frac{\mathrm{D}\Phi}{\mathrm{D}t} = \frac{\partial\Phi}{\partial t} + \vec{v}\cdot\nabla\Phi = \int_{V_t} \frac{\partial(\rho\phi)}{\partial t}\,\mathrm{d}V + \int_{V_t} \nabla\cdot(\rho\phi\vec{v})\,\mathrm{d}V \tag{2.2}$$

Thus, using the divergence theorem to obtain a surface integral, the final form of the Reynolds transport theorem is obtained.

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{V_t} \rho\phi\,\mathrm{d}V = \int_{V_t} \frac{\partial(\rho\phi)}{\partial t}\mathrm{d}V + \oint_{S_t} (\rho\phi)\vec{v}\cdot\vec{n}\,\mathrm{d}S \tag{2.3}$$

This equation should be interpreted as follows:

Variation per time unit of the content of property Φ in control volume V_t	=	Variation due to the change of the content of property Φ in the particles inside the volume V_t	+	Variation due to the net convective flow of Φ through the surface of the control volume S_t

From this principle, all conservation laws can be deduced. This can be achieved by changing the generic property Φ by the one that defines each equation. In the following pages of this report, conservation equations will be deduced and discussed.

The equations are presented in differential and integral formulation, since both nomenclatures are sometimes useful. While the first one provides great information about the problem and only requires the boundary conditions, the other usually require hypothesis or simplifications but is easier to be solved both analytically and numerically [3].

The first can be understood as the formal definition and the second as the applied. In this report the equations of mass and momentum are presented and discussed from the integral to the differential formulation.

2.1.1 Mass conservation equation

The mass conservation is the most elementary conservation equation. It is also known as the continuity equation. Starting from the integral form of the Reynolds equation, the property Φ becomes the mass m and, therefore, the specific property is $\phi = 1$. Since the system volume is defined as a fixed identity system, the overall mass will not change during time. A static control volume is also assumed. [3]

$$\int_{V_t} \frac{\partial \rho}{\partial t}\, \mathrm{d}V + \oint_{S_t} \rho \vec{v} \cdot \vec{n}\, \mathrm{dS} = 0 \tag{2.4}$$

To obtain the differential form of the equation a Cartesian coordinate system is being used where the volume differential is formed by $\mathrm{d}V = \mathrm{d}x\mathrm{d}y\mathrm{d}z$. Thus, a cube with six faces is studied. The mass inside this cube is defined as follows:

$$\frac{\partial(\rho \mathrm{d}V)}{\partial t} + \frac{\partial \dot{m}_x}{\partial x}\mathrm{d}x + \frac{\partial \dot{m}_y}{\partial y}\mathrm{d}y + \frac{\partial \dot{m}_z}{\partial z}\mathrm{d}z = 0 \tag{2.5}$$

Assuming that the flow rate $\dot{m}_x$ in the face x is defined as $\dot{m}_x = \rho u \mathrm{d}y\mathrm{d}z$ where u is the flow velocity through x face (and being equal for faces y (v) and z (w)). The equation evolves to:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = 0 \tag{2.6}$$

Finally, this equation can also be rearranged as:

$$\frac{\partial \rho}{\partial t} + \nabla(\rho \vec{v}) = 0 \tag{2.7}$$

2.1.2 Momentum conservation equation

The momentum conservation equation combines the Reynolds theory with Newton's second law of motion. In this case, the property Φ becomes momentum $m\vec{v}$. On one hand, the derivative of the momentum is equal to the sum of the forces of the system. On the other hand, it can also be treated with the transport theorem with $\phi = \vec{v}$. Therefore,

$$\sum F_{sys} = \int_{V_t} \frac{\partial(\vec{v}\rho)}{\partial t}\, \mathrm{d}V + \oint_{S_t} \vec{v}\rho\vec{v} \cdot \vec{n}\, \mathrm{d}S \tag{2.8}$$

The interpretation of this equation indicates that the sum of forces acting on a system F_{sys} is equal to the velocity of accumulation of momentum within the control volume plus the net flux of momentum across the boundaries.

Let's first focus on the forces that may act on the system. Following the structure of the Reynolds theorem, they can be divided into mass (or volumetric) forces and surface forces. To simplify the problem, other forces such as the electromagnetic forces are not taken into account.

$$\sum \vec{F}_{sys} = \sum \vec{F}_{mass} + \sum \vec{F}_{surface} \tag{2.9}$$

Thus, the only mass force that affects the control volume is the caused by gravity.

$$\sum \vec{F}_{mass} = \int_{V_t} \rho \vec{g}\, \mathrm{d}V \tag{2.10}$$

The surface forces can be divided into pressure forces and viscous forces. The first are applied the opposite direction to the normal vector $\vec{n}$ of the entire surface of the control volume.

$$\sum \vec{F}_{pressure} = -\oint_{S_t} p\vec{n}\, \mathrm{d}S \tag{2.11}$$

Viscous or shear forces are applied both normal and tangential to the surface. Thus, to be fully taken into account the stress tensor $\bar{\bar{\tau}}$ is used.

$$\sum \vec{F}_{shear} = \oint_{S_t} \bar{\bar{\tau}} \cdot \vec{n}\, \mathrm{d}S \tag{2.12}$$

Therefore, combining the expressions of the forces with Equation (2.8), the equation in integral form of momentum can be expressed as follows:

$$\int_{V_t} \rho \vec{g}\, \mathrm{d}V - \oint_{S_t} p\vec{n}\, \mathrm{d}S + \oint_{S_t} \bar{\bar{\tau}} \cdot \vec{n}\, \mathrm{d}S = \frac{\partial}{\partial t}\int_{V_t} \rho\vec{v}\, \mathrm{d}V + \oint_{S_t} \vec{v}\rho\vec{v}\vec{n}\, \mathrm{d}S \tag{2.13}$$

Applying the divergence theorem to the surface integrals, using the continuity Equation (2.7) and manipulating the mathematical expressions, the equation in differential formulation in obtained.

$$\rho \frac{\mathrm{D}\vec{v}}{\mathrm{D}t} = -\nabla p + \rho\vec{g} + \nabla \cdot \bar{\bar{\tau}} \tag{2.14}$$

This result is of great importance in the field of fluid mechanics and is known as the Cauchy equation. In fact, by developing the shear forces term, one can easily obtain the Navier Stokes equations, the governing and most used equations in the area of computational fluid dynamics [2].

2.1.3 Navier-Stokes equations

Assuming an incompressible flow for a Newtonian fluid, and by means of the Stokes's stress constitutive equation, the shear forces term may be expressed as $\nabla\bar{\bar{\tau}} = \mu\nabla^2\vec{v}$ where μ is the dynamic viscosity. In a simplified form, where density and viscosity are considered as constant values (ρ_0 and μ_0 respectively), the following expression obtained from Equation (2.14) describes the incompressible Navier-Stokes equations [3].

$$\rho_0 \frac{\partial\vec{v}}{\partial t} + \rho_0\nabla \cdot (\vec{v}\vec{v}) = \mu_0\nabla^2\vec{v} - \nabla p + \rho_0\vec{g} \tag{2.15}$$

The terms to the left of the equality represent the material derivative of the velocity. The first is the variation with respect to time. The second is the convective term and is the most conflicting when solving the system of equations due to its non-linearity. It can be understood intuitively as the difficulty in finding the velocity lies in the fact that it transports itself.

The sum of forces is shown on the right-hand side. The first term is the one coming from the viscous forces and is understood as the diffusive term which does not depend on the velocity directly but on its variation. The other two elements show the pressure gradient and the effect of the mass forces due to the acceleration of the velocity.

2.1.4 Heat conduction equation

Heat conduction is the heat transfer method that occurs in a physical environment due to the existence of a gradient of temperature. It is presented in this section as one of the main terms of the energy conservation equation along with the convection-diffusion one. The rest of the energy equation is not presented as it has no relevance to the report.

Any material has a thermal conductivity λ associated. This coefficient expresses the ability to transfer heat by conduction. Higher values indicate more capacity to conduct. Solid materials are the best conductors while gases have much lower λ coefficients. The conductivity coefficient also depends on the temperature [4]. However, it will be considered as constant for simplification.

For any isotropic material the following equation describes the phenomenon.

$$\frac{\rho C_p}{\lambda}\frac{\partial T}{\partial t} = \frac{\dot{q_v}}{\lambda} + \nabla^2 T \tag{2.16}$$

Note that, to simplify the study, the material density ρ and the thermal conductivity λ are considered constant. The specific heat capacity C_p and C_v are considered equals and the internal energy is given as $\mathrm{d}u = C_p \mathrm{d}T$.

The first term of the equation describes the temporary evolution of the temperature. In steady problems it becomes null.

The second term represents the internal heat sources $\dot{q_v}$, the heat generated inside the control volume. Again, in simple problems where there are no internal sources, this term will be neglected.

Finally, the last term stands for the Fourier's thermal conduction law. This equation relates the heat flow with the negative local temperature gradient as it can be seen here.

$$\vec{q} = -\lambda \nabla T \tag{2.17}$$

2.1.5 Convection-diffusion equation

The convection-diffusion equations describe different phenomena involving the transport of a property in a physical system. Following on from heat transfer, the equations can study how temperature is distributed in a moving fluid. Convection-diffusion can also be derived in the Navier-Stokes equations when the transported property is the velocity. For now, the equations are presented for a generic scalar field property ϕ [5].

$$\frac{\partial \rho \phi}{\partial t} = \nabla \cdot (\Gamma\, \nabla \phi) - \nabla \cdot (\rho \vec{v} \phi) + S \tag{2.18}$$

The first term represents the accumulation of ϕ through time. It is equal to the difference between the diffusion and the convection term plus a parameter S depending on what property ϕ is being studied.

The diffusion term $\nabla \cdot (\Gamma\, \nabla \phi)$ describes the motion through gradients of ϕ. Depending on the specific meaning of ϕ, it can represent heat flux, viscous stress, chemical species diffusion flux from higher to lower concentrations, etc. Γ is the diffusion factor and may intensify or decrease the phenomenon as a function of its value.

The convection term $\nabla \cdot (\rho \vec{v} \phi)$ presents the net convective flow of ϕ. Note that forced convection and constant physical properties are supposed and, therefore, the velocity field does not depend on ϕ. It is also assumed incompressible flow such that the continuity Equation (*2.7*) is supposed to be as follows.

$$\nabla \cdot \vec{v} = 0 \tag{2.19}$$

Table 1 obtained from [6] lists the appropriate parameters in order to reproduce the governing equations.

Although the calculation will be performed in a generalist way, the main objective of this report is to study the heat transfer in order to find the resulting temperature field. Therefore, the concrete cases will be performed with the last equation.

Equation	ϕ	Γ	S
Continuity	1	0	0
Momentum in x direction	u	μ	$-\partial p_d/\partial x$
Energy (constant c_P)	T	λ/c_P	Φ/c_P

Table 1. Parameters to replace in convection-diffusion equation

As a last theoretical note, it is essential to present the Peclet number P. This is the ratio between advection and diffusion.

$$P = \frac{|\vec{v}|\rho L}{\Gamma} \tag{2.20}$$

where L is the characteristic length.

Thus, for great values of the Peclet number, advection will prevail, while for lower values the phenomenon of diffusion will gain importance.

2.2 Numerical methods

Numerical methods are a major branch of mathematics that has gained great importance with the development of computers. In scientific and engineering studies they are of great use in solving problems that otherwise could not be solved.

Mathematical models describing physical phenomena usually use continuous variables. A computer needs to work with discrete representations of information because calculations using continuous variables are much less efficient. Numerical methods aim to adapt a continuous variable model to a discrete variable model.

In this way, computational power can be used in conjunction with numerical methods to solve problems whose analytical solution is not known or which are too large to be dealt with manually [7].

2.2.1 Finite Volume Method

There are different types of numerical analysis methods to represent physical problems. Among the best known are the Finite Element Method, the Finite Volume Method and the Finite Difference Method.

The Finite Volume Method (FVM) is the one studied and applied to the problems posed in this report. Currently, it is one of the most widely used methods for CFD studies and simulations that require large regular meshes.

This method consists of imposing flow balances on control volumes by means of surface integrals [8]. This is a very efficient model on structured and rectangular meshes that can easily be implemented and is one of the most intuitive ones.

As a weakness, it can be more difficult to design a scheme to obtain a sufficiently high accuracy than, for instance, using finite elements. However, the robustness of the method and the relative simplicity of the problems presented in this paper make it a convenient method.

2.2.2 Discretisation of the domain

The first step in applying the finite volume method is to divide the study body into small portions. Each portion is a control volume with an associated node. Each node represents the value of a thermophysical property (e.g., temperature, pressure, ...) at that point in space. The property value can be considered to be the average of the whole control volume [9].

Algebraic equations are used to define the relationships between adjacent nodes. In this project, the nodes belonging to the boundary of the domain have known properties imposed by the boundary conditions.

The number of nodes required to solve the problem adequately is not a fixed or standard value. On the one hand, a large number of nodes allows more accurate solutions to be reached but with a higher computational cost, especially in terms of solving time. On the other hand, a small number of nodes can give rise to solutions that are too roughly approximated. One of the objectives of this project is to determine in each case what is the optimal number of nodes to balance the performance and the quality of the solution.

This paper deals with problems in one and two dimensions. For two dimensions, the control volume becomes the surface. Below is part of a regular mesh used in this type of problem. In the centre, the studied node *P* is shown. To refer to neighbour nodes, cardinal coordinates are used: north (*N*), south (*S*), east (*E*) or west (*W*).

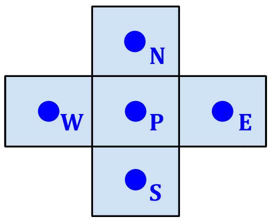


Figure 1. Nomenclature of the relative position of the nodes

The objective is to obtain the value of a property of node Φ_P as a function of its neighbouring nodes with an equation of the following form.

$$a_P\Phi_P = a_E\Phi_E + a_W\Phi_W + a_N\Phi_N + a_S\Phi_S + b_P \tag{2.21}$$

This is achieved by approximating the equations with partial derivatives describing a certain physical phenomenon to algebraic equations using FDM. The coefficients accompanying the property of each node a_I and the independent term b_P are obtained in different ways depending on the physical problem. In this project, the coefficients for each type of problem will be presented.

Once this equation has been found for each of the N nodes of the domain, a system of algebraic equations with N variables and N equations is available. This system is easily solved by a computer using one of the solvers explained in the section 0.

Returning to the geometry of the problem, there are two different ways to discretise the problem when distributing the set of nodes and control volumes. Although the internal nodes are treated in the same way, it affects the treatment of the nodes close to the boundary.

The two methods are node-centred and face-centred. The choice of one method or the other is arbitrary. Thus, in this project are presented and studied both: the heat conduction problems with node-centred discretisation and the convection-diffusion problems with face-centred discretisation.

2.2.2.1 Centred-nodes

The domain is divided into N_{CV} control volumes of equal size and a node is placed in the centre of each. In addition, a node is added to each of the boundary walls. Thus, there are a total of $N_{VC} + 2$ nodes. Note that the nodes on the boundary do not have an associated control volume. Figure 2presents the corresponding scheme

2.2.2.2 Centred-faces

The domain is divided into N_N nodes from one boundary to the opposite one. The faces of the control volumes are traced between the nodes. This way, there are $N_N + 1$ control volumes.

In this case, boundary nodes do have an associated control volume although, in regular meshes, their volume is half that of the internal CVs.

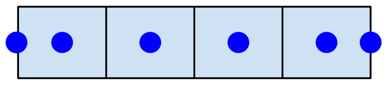


Figure 2. One-dimensional centred-nodes scheme

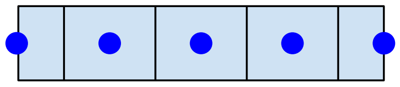


Figure 3. One-dimensional centred-faces scheme

2.2.3 Time schemes of resolution

When dealing with transient problems in numerical methods, it is common to work with variables of two different time instants in the same equation. For example, the property in a point of a solid body at instant t^{n+1} depends on the value of this same property at t^n.

This kind of problems are approached by solving all variables for $t = t_1$ before starting to calculate the value of the property at the immediately following time step $t = t_1 + \Delta t$. Thus, in each time iteration the unknowns are Φ^{n+1} while Φ^n are assumed to be known.

There are three different methods presented by Patankar in [5] to deal with unsteady problems: explicit, fully implicit and Crank-Nicholson. A one-dimensional problem where the property Φ_P^{n+1} is unknown is taken as example.

- Explicit method. This scheme assumes that the property Φ_P^n for a point P maintains its value during the entire time step except just at $t = t + \Delta t$.This means that Φ_P^{n+1} is not depending on their neighbours Φ_E^{n+1} and Φ_W^{n+1} but only on the already known variables Φ_P^n, Φ_E^n and Φ_W^n. This way, Φ_P^{n+1} can be *explicitly* and no systems of equations need to be solved.

$$a_P\Phi_P^{n+1} = a_E\Phi_E^n + a_W\Phi_W^n + b_P \tag{2.22}$$

  This method, however, presents a severe constraint. The coefficient b_P depending on Φ_P^n could become negative. This situation must be avoided or physically unrealistic results could be achieved. Therefore, the time step Δt is restricted to this condition leading to require much most time steps than probably desired.

- Fully implicit method. Any method that in order to found Φ_P^{n+1} requires the properties of its neighbours at t^n and t^{n+1} is implicit. If it only demands Φ_E^{n+1} and Φ_W^{n+1}, then is fully implicit. This is the only unconditionally stable method not only mathematically but also physically. The expression of Φ_P^{n+1} is as follows.

$$a_P\Phi_P^{n+1} = a_E\Phi_E^{n+1} + a_W\Phi_W^{n+1} + b_P \tag{2.23}$$

  Now, a system of equations has to be solved. Note that b_P still depends only on known values.

- Crank Nicholson method. This scheme assumes that Φ_P^n lineally varies from t^n to t^{n+1}. For small time steps, CN is more accurate than a fully implicit method. However, implies more complex notation.

$$a_P \Phi_P^{n+1} = \frac{1}{2}(a_E \Phi_E^n + a_E \Phi_E^{n+1} a_W \Phi_W^n + a_W \Phi_W^{n+1}) + b_P \tag{2.24}$$

A general expression depending on a factor f can describe the general equation of Φ_P^{n+1}. In this way, $f = 0$ stands for the explicit scheme, $f = 0.5$ represents the Crank-Nicholson method and $f = 1$ leads to a fully implicit scheme.

$$a_P \Phi_P^{n+1} = a_E[f \Phi_E^{n+1} + (1-f)\Phi_E^n] + a_W[f \Phi_W^{n+1} + (1-f)\Phi_W^n] + b_P \tag{2.25}$$

The following figure adapted from [5] illustrates the evolution of the property over the time step.

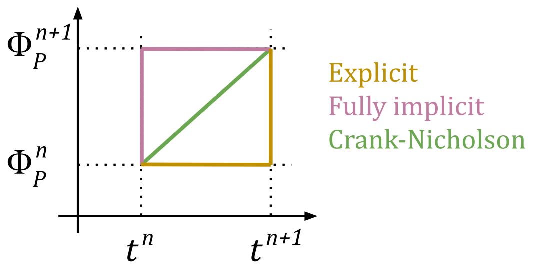


Figure 4. Time schemes comparison

2.2.4 System of linear equations

As already mentioned, the aim of discretising the domain and applying the Finite Volumes Method is to convert a continuous variable equation to a system of linear algebraic equations. This system is made up of as many equations as nodes have been studied.

An example is given below. A domain of square geometry is assumed and is discretised by means of the face-centred scheme with a total of 9 nodes. The nodes have been numbered from 1 to 9 from left to right and from top to bottom.

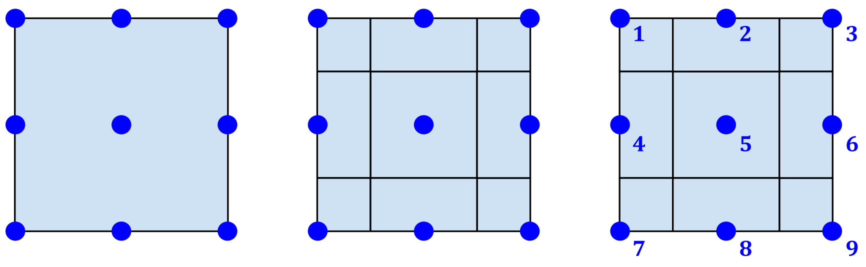


Figure 5. Centred-faces discretisation step by step

Now, nine equations of the same form as Equation (2.21) can be obtained. In this case, not all nodes have four neighbouring nodes. For example, node 1 is neighbouring node 2 on the east and node 4 on the south. Therefore, the equation defining the property Φ_1 will only depend on two adjacent nodes. The same applies to the rest of the nodes. If these equations are arranged in matrix form, the following expression is obtained [5].

$$\begin{pmatrix} a_{P1} & a_{E1} & 0 & a_{S1} & 0 & 0 & 0 & 0 & 0 \\ a_{W2} & a_{P2} & a_{E2} & 0 & a_{S2} & 0 & 0 & 0 & 0 \\ 0 & a_{W3} & a_{P3} & 0 & 0 & a_{S3} & 0 & 0 & 0 \\ a_{N4} & 0 & 0 & a_{P4} & a_{E4} & 0 & a_{S4} & 0 & 0 \\ 0 & a_{N5} & 0 & a_{W5} & a_{P5} & a_{E5} & 0 & a_{S5} & 0 \\ 0 & 0 & a_{N6} & 0 & a_{W6} & a_{P6} & 0 & 0 & a_{S6} \\ 0 & 0 & 0 & a_{N7} & 0 & 0 & a_{P7} & a_{E7} & 0 \\ 0 & 0 & 0 & 0 & a_{N8} & 0 & a_{W8} & a_{P8} & a_{E8} \\ 0 & 0 & 0 & 0 & 0 & a_{N9} & 0 & a_{W9} & a_{P9} \end{pmatrix} \begin{pmatrix} \Phi_{P1} \\ \Phi_{P2} \\ \Phi_{P3} \\ \Phi_{P4} \\ \Phi_{P5} \\ \Phi_{P6} \\ \Phi_{P7} \\ \Phi_{P8} \\ \Phi_{P9} \end{pmatrix} = \begin{pmatrix} b_{P1} \\ b_{P2} \\ b_{P3} \\ b_{P4} \\ b_{P5} \\ b_{P6} \\ b_{P7} \\ b_{P8} \\ b_{P9} \end{pmatrix} \quad (2.26)$$

This matrix format is a good way to synthesise the problem and is useful for solving it using some solvers. However, in the problem solving of this project, a separate matrix has been used for each coefficient and the value is referenced in the equations according to the relative position of rows and columns.

$$\begin{pmatrix} a_{P1} & a_{P2} & a_{P3} \\ a_{P4} & a_{P5} & a_{P6} \\ a_{P7} & a_{P8} & a_{P9} \end{pmatrix}; \begin{pmatrix} a_{E1} & a_{E2} & a_{E3} \\ a_{E4} & a_{E5} & a_{E6} \\ a_{E7} & a_{E8} & a_{E9} \end{pmatrix}; \dots; \begin{pmatrix} b_{P1} & b_{P2} & b_{P3} \\ b_{P4} & b_{P5} & b_{P6} \\ b_{P7} & b_{P8} & b_{P9} \end{pmatrix} \quad (2.27)$$

This format is more intuitive for code development as the matrix follows the format of the discretisation geometry.

2.2.5 Solvers

A linear equation system solver is an algorithm whose objective is to find the result of the unknown variables of the system. There is a wide variety of solvers that use all kinds of resolution methods. The equation to be solved has the same form as Equation (2.26).

$$A\Phi = B \quad (2.28)$$

The analytical solution to this system is easily obtained by using the inverse of the A matrix.

$$\Phi = A^{-1}B \quad (2.29)$$

However, performing this operation is a process that is too costly in terms of computing time. This is where the different strategies come in to attempt to solve this system with maximum efficiency.

Linear equation system solvers can be divided into two main groups.

- Direct solvers. These are algorithms that allow the exact solution of the system to be found by means of algebraic manipulations of the expressions. They are very fast and efficient but are difficult to implement and have severe limitations in problems that do not meet very specific conditions.

- Iterative solvers. These algorithms obtain an approximation of the solution through a process of iterations in the equations of the system. They start from an arbitrary initial value of the unknown values to calculate the result of these same variables. The process is repeated until an acceptable approximation is reached, which will never be exact. However, they are easy to implement and useful for very large matrices. The time required to solve will depend on the desired accuracy of the solution, which is usually measured by a convergence factor δ.

The solvers used in this project are presented below.

2.2.5.1 TDMA

The tri-diagonal matrix algorithm (TDMA) is a direct solver algorithm [10].

It can be easily applied by running through the N equations that make up the system twice. First, from equation 0 to N, the vectors P and Q are found.

$$P_i = \frac{a_{E\,i}}{a_{P\,i} - a_{W\,i}P_{i-1}} \qquad Q_i = \frac{b_{P\,i} + a_{W\,i}Q_{i-1}}{a_{P\,i} - a_{W\,i}P_{i-1}} \tag{2.30}$$

Then, running through the equations from N to 0, the results of the Φvariables are obtained.

$$\Phi_i = P_i\Phi_{i+1} + Q_i \tag{2.31}$$

This algorithm presents an exceptional ease of implementation and speed of computation. However, since this direct solver can only be used in tridiagonal matrices, in practice it can only be used on its own in very simple cases. In this project, it is only used to solve one-dimensional problems.

2.2.5.2 Gauss-Seidel

The Gauss-Seidel algorithm is an iterative method for solving linear systems. It can be applied to all types of diagonally dominant matrices [11] (the sum of the elements of a row is less than or equal to the diagonal element) or to symmetric and positive definite matrices.

The algorithm starts from an arbitrary initial Φ^0 value. The program runs through the equations from 0 to N calculating the values of Φ^1 using the value of Φ^1 for the variables already calculated and Φ^0 for those still unknown. The process is repeated until convergence. The following expression defines the computation of Φ^{n+1} in each iteration.

$$\Phi_i^{n+1} = \frac{1}{a_{ii}}(b_i + \sum_{j=1}^{i-1} a_{ij}\Phi_j^{n+1} + \sum_{j=i+1}^{N} a_{ij}\Phi_j^{n}) \tag{2.32}$$

This is a good method for solving large matrices that are not tri-diagonal and cannot use TDMA. However, it is slow. In this project it is used for solving two-dimensional problem systems.

2.2.5.3 Line-by-line

The line-by-line solver is an iterative solver that uses a direct solver for row and column resolution. In particular, it uses part of the Gauss-Seidel and TDMA solver.

It is useful for solving problems whose coefficient matrix A is diagonally dominant. For example, a two-dimensional physical problem with a structured mesh, where the property of each node depends on a maximum of 4 neighbouring nodes (N, W, S, E). The objective is to convert A into a tri-diagonal matrix.

This can be achieved by solving the problem first row by row and then column by column. In this way, neighbouring nodes that are not in the row or column become part of the independent term of the equation.

The equation to solve row-by-row is as follows.

$$a_P \Phi_P^{n+1} = a_E \Phi_E^{n+1} + a_W \Phi_W^{n+1} + b_P \quad \rightarrow \quad b_P = b_{P_0} + a_S \Phi_S^n + a_N \Phi_N^n \tag{2.33}$$

And the equation column-by-column:

$$a_P \Phi_P^{n+1} = a_N \Phi_N^{n+1} + a_S \Phi_S^{n+1} + b_P \quad \rightarrow \quad b_P = b_{P_0} + a_E \Phi_E^n + a_W \Phi_W^n \tag{2.34}$$

In other words, a two-dimensional problem is converted to a one-dimensional problem. This process is repeated until convergence is reached.

This solver, although a bit more complicated to implement than the Gauss-Seidel solver, is usually more efficient and faster. In this project it will also be used for solving two-dimensional problems.

3

Heat conduction

This part of the project includes the discretisation of the heat conduction equation introduced in its analytical form in the previous section. The numerical methods described are applied to a problem where the temperature field in a solid made up of different materials has to be determined. It is a two-dimensional and transient problem.

3.1 Heat conduction equations

The purpose of solving the heat conduction equation is to determine the temperature field of the element under study. That is, solving the problem consists of solving a system of equations where the unknowns are the value of the temperature at each point of the body.

To achieve this objective, the body is discretised into nodes and control volumes as explained in the previous section. Thus, the set of equations to be solved will be formed by linear combinations of temperatures between related nodes accompanied by coefficients that incorporate the physical properties of the problem.

3.1.1 One-dimensional steady problem

A first simple case as an example is being presented. A steady problem in one dimension. In this case, it is studied any node P that has as neighbors a node W to its left and node E to its right. Therefore, the temperature of node P will only depend on the temperature of nodes E and W as follows.

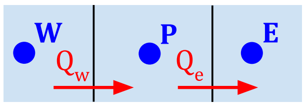


Figure 6. Heat conduction 1D discretisation

$$a_P T_P = a_E T_E + a_W T_W + b_P \tag{3.1}$$

Where a_E and a_W are the coefficients of the nodes E and W respectively and b_p is the independent term of the equation. Now it is necessary to relate the coefficients to the physical properties of the heat transfer problem. The heat conduction can be expressed like the following [5].

$$-\lambda_w \frac{T_P - T_W}{d_{PW}} S_w + \lambda_e \frac{T_E - T_P}{d_{PE}} S_e + q_v V_P = 0 \tag{3.2}$$

Here the volume of the control volume is V_P, the heat intern sources are q_v, the surfaces between nodes are S_i and d_{PI} represents the distance between the node P and node I. Also note that λ_i represents the heat conduction of the material found between the node P and the node I. This last parameter is computed as the harmonic mean of λ_I and λ_P like follows.

$$\lambda_i = \frac{2\lambda_I \lambda_P}{\lambda_I + \lambda_P} \tag{3.3}$$

Once the physical terms have been clarified, the expression can be rearranged to have the same form as Equation (*3.1*).

$$\left(\frac{\lambda_w S_w}{d_{PW}} + \frac{\lambda_e S_e}{d_{PE}}\right) T_P = \left(\frac{\lambda_w S_w}{d_{PW}}\right) T_W + \left(\frac{\lambda_e S_e}{d_{PE}}\right) T_E + (q_v V_P) \tag{3.4}$$

Here the coefficients are clearly distinguishable. They can be expressed as:

$$a_P = \frac{\lambda_w S_w}{d_{PW}} + \frac{\lambda_e S_e}{d_{PE}} \qquad a_W = \frac{\lambda_w S_w}{d_{PW}} \qquad a_E = \frac{\lambda_e S_e}{d_{PE}} \qquad b_P = q_v V_P \tag{3.5}$$

3.1.2 General case

After a first simpler case, a general case can be studied more easily. That is, with more than one dimension and in a transient regime. In this case, a two-dimensional case is presented. However, it could be applied in the same way to 3D bodies.

One starts again from the heat conduction equation applied to any P node. Now, together with the neighbours on the W and E sides, the one on the top N and the one on the bottom S are added. In addition, the temperature of the node P one instant of time before the one studied at time T_P^0 is also taken into account. This temperature is known.

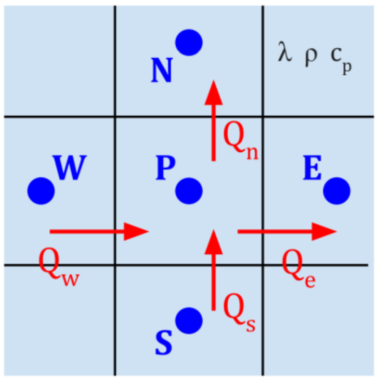


Figure 7. Heat conduction 2D discretisation

At this point, to formulate the equations it is necessary to decide the type of resolution scheme to be used: explicit, implicit or Crank-Nicolson. The general form of the problem is as follows.

$$\rho_P V_P c_{p_P}(T_P - T_P^0) = \beta \sum \dot{Q} \Delta t + (1-\beta)\dot{Q}^0 \Delta t \tag{3.6}$$

Depending on the method chosen, the value of β will be 0, 0.5 or 1. In this case, in order to work with an unconditionally stable method and with a simple notation, the implicit method will be used. That is, $\beta = 1$. In addition, it is assumed that there are no internal heat sources. Thus, developing the heat summation of the previous equation we arrive at:

$$\sum \dot{Q} = -\lambda_w \frac{T_P - T_W}{d_{PW}} S_w + \lambda_e \frac{T_E - T_P}{d_{PE}} S_e - \lambda_s \frac{T_P - T_S}{d_{PS}} S_s + \lambda_n \frac{T_N - T_P}{d_{PN}} S_n \tag{3.7}$$

And therefore, the equation to be solved is as follows.

$$\sum \dot{Q} = \frac{\rho_P V_P c_{p_P}(T_P - T_P^0)}{\Delta t} \tag{3.8}$$

Following the same steps than the previous section, the heat equation has this form:

$$a_P T_P = a_E T_E + a_W T_W + a_S T_S + a_N T_N + b_P \tag{3.9}$$

And the coefficients are the following.

$$\begin{gathered} a_W = \frac{\lambda_w S_w}{d_{PW}} \qquad a_E = \frac{\lambda_e S_e}{d_{PE}} \qquad a_S = \frac{\lambda_s S_s}{d_{PS}} \qquad a_N = \frac{\lambda_n S_n}{d_{PN}} \\ a_P = a_W + a_E + a_S + a_N + \frac{\rho_P V_P c_{p_P}}{\Delta t} \qquad b_P = \frac{\rho_P V_P c_{p_P}}{\Delta t} T_P^0 \end{gathered} \tag{3.10}$$

This system is obtained for each instant of time at which the solution is required to be known. The coefficients matrix is always diagonally dominant. Therefore, either the Gauss-Seidel solver or line-by-line can be used.

3.2 Four materials conduction problem

This study aims to document the solution of a heat conduction problem through a solid made of four different materials. The boundary conditions evolve with time and this implies the use of a transient treatment.

The solid to be studied is made up of four materials. This solid can be assumed to be a prism with a depth large enough to allow the problem to be dealt in two dimensions. Thus, the object of the study is the section of the prism. Figure 8 shows a scale diagram of the problem to be studied.

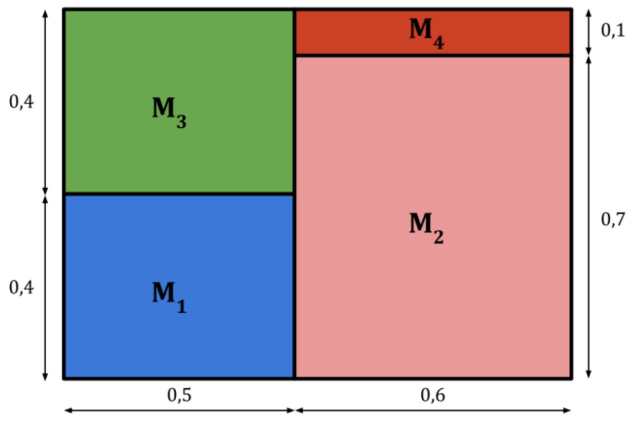


Figure 8. General scheme of the problem

At instant $t = 0$, the whole solid is at an initial temperature of $T_0 = 8°C$. The study consists of determining the temperature map for any given instant of time and position during the course of the problem.

The materials that constitute the solid are distributed as can be seen in Figure X. Each material has different physical properties. To simplify the problem, it is assumed that the properties of the materials do not vary with temperature and are thus constant over time. The table below shows the properties of each material.

	$\rho\ [kg/m^3]$	$c_p\ [J/kgK]$	$\lambda\ [W/mK]$
Material 1	1500.0	750.0	170.0
Material 2	1600.0	770.0	140.0
Material 3	1900.0	810.0	200.0
Material 4	2500.0	930.0	140.0

Table 2. Root-mean-square error for different resolution schemes

3.2.1 Boundary conditions

This section presents the boundary conditions of the solid object of study. Each wall of the solid is faced with different conditions. The top, bottom, left and right walls are considered by referring to the relative position of the wall to the solid from the reader's point of view. Thus,

- Bottom. The lower wall is isothermal at temperature $T = 23.0$°C.
- Top. The upper wall absorbs (from outside to inside) a heat flow $q_{flow} = 60.0\ W/m$.
- Left. The left wall is in contact with a fluid at $T_g = 33.0$°C and with a heat transfer coefficient $\alpha_g = 9.00 W/m^2K$.
- Right. The right wall is isothermal at temperature $T = 8.0 + 0.005t$ °C where t is the time expressed in seconds.

3.2.2 Discretisation of the problem

In order to solve the problem numerically, it is necessary to discretize the domain of study as explained above. In this case, a discretisation of centred nodes is used. The solid has been divided in such a way that the boundaries of the control volumes coincide with the boundaries of the materials. Once all the sections are divided, a node is placed at the centre of each volume and at the contour surfaces of the solid. A schematic is shown.

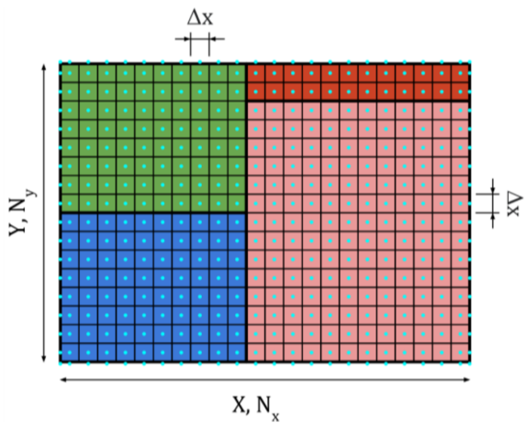


Figure 9. Centred-nodes body discretisation

The aim of discretising the solid is to apply an energy balance at each node to know its temperature at the instant of time studied. The internal nodes have an associated control volume while the wall nodes do not and are studied by applying the boundary conditions. Now, the latter are presented, since the former have already been introduced in the previous theoretical development. An implicit treatment is used ($\beta = 1$).

3.2.2.1 Top nodes

The nodes at the top wall of the solid are studied according to the heat boundary condition. In this case, there is a constant heat flow q_{flow} from the outside to the inside of the solid.

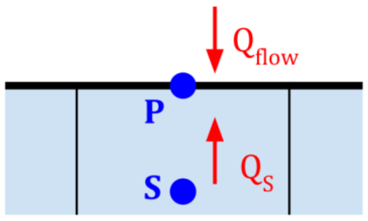


Figure 10. Top nodes scheme

$$\sum Q^{n+1} = 0$$

$$-\lambda_S \frac{T_P^{n+1} - T_S^{n+1}}{d_{PS}} - \dot{q}_{flow} = 0 \tag{3.11}$$

$$a_P T_P^{n+1} = a_S T_S^{n+1} + b_P$$

$$a_S = \frac{\lambda_S}{d_{PS}} \qquad a_P = a_S \qquad b_P = -\dot{q}_{flow} \tag{3.12}$$

3.2.2.2 Bottom nodes

The lowest wall is isotherm and therefore the temperature of the nodes is the temperature of the wall. Thus, no energy balance is required and coefficients are only used to impose the desired temperature.

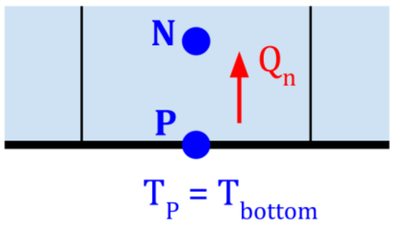


Figure 11. Bottom nodes scheme

$$T_P^{n+1} = T_{bottom}$$
$$a_P T_P^{n+1} = b_P \tag{3.13}$$

$$a_P = 1 \qquad b_P = T_{bottom} \tag{3.14}$$

3.2.2.3 Left nodes

Left nodes remain in contact with a fluid at a known temperature T_g and with a heat transfer coefficient α_g. This way, one imposes that the convection heat is equal to the conduction heat as boundary condition.

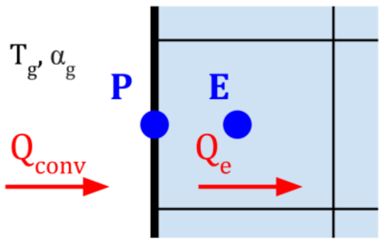


Figure 12. Left nodes scheme

$$\sum Q^{n+1} = 0$$
$$\alpha_g\left(T_g - T_P^{n+1}\right) + \lambda_E \frac{T_E^{n+1} - T_P^{n+1}}{d_{PE}} = 0 \tag{3.15}$$
$$a_P T_P^{n+1} = a_E T_E^{n+1} + b_P$$

$$a_E = \frac{\lambda_E}{d_{PE}} \qquad a_P = a_E + \alpha_g \qquad b_P = \alpha_g T_g \tag{3.16}$$

3.2.2.4 Right nodes

Finally, right wall is also considered as isotherm. However, in this case the wall temperature varies as a function of time according to the expression $T_{right}(t) = 8.0 + 0.005t$ °C. Thus, node will acquire the same temperature.

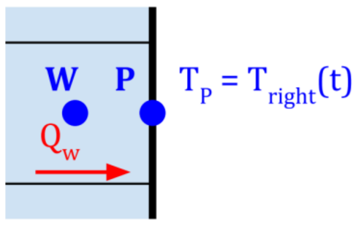


Figure 13. Right nodes scheme

$$T_P^{n+1} = T_{right}^{n+1}$$
$$a_P T_P^{n+1} = b_P \tag{3.17}$$

$$a_P = 1 \qquad b_P = T_{right}^{n+1} \tag{3.18}$$

3.2.2.5 Internal nodes between materials

When the control volume of a node is in contact with a control volume corresponding to a different material, special treatment is required with respect to the other internal nodes. Although the equations are the same, the λ coefficient changes.

For example, if a node P is a neighbour with a node K and they have a different associated material, the heat flux $\dot{q}_{PK}$ is evaluated with the harmonic mean of the respective coefficients.

$$\lambda_{PK} = \frac{d_{PK}}{\frac{d_{Pk}}{\lambda_P} + \frac{d_{kK}}{\lambda_K}} \tag{3.19}$$

3.2.3 Results

The time domain is defined between $t = 0$ and $t = 10000s$. During this time, the only boundary condition that varies is the temperature of the right wall, which evolves from $T = 8°C$ to $T = 58°C$ linearly. Here, the evolution of the temperature with respect to time is studied for two different points of the domain.

$$P_1(x, y) = (0.65; 0.56) \qquad P_2(x, y) = (0.74; 0.72)$$

The following figure presents the evolution of these two points. P_1 is represented in blue and P_2 in red.

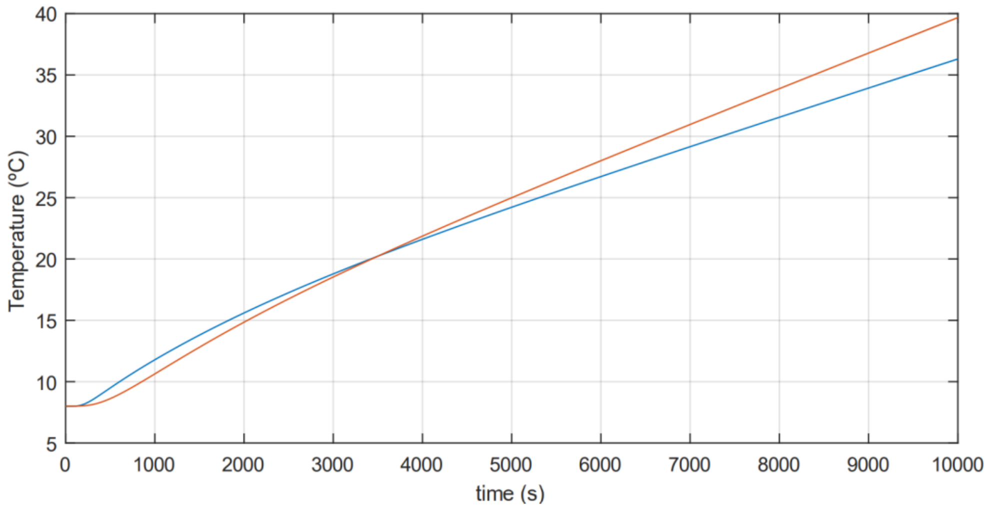


Figure 14. Evolution of the temperature in the two nodes

At first glance, the result is consistent. Both nodes have an initial temperature $T_0 = 8°C$ which has been imposed on the whole solid. As time goes on, the temperature increases almost linearly, that is because the nodes are close to the wall that varies the temperature. These results have been verified using those obtained by the Heat and Mass Transfer Technological Centre (CTTC) at UPC [12].

The following is a global representation of the solution to the problem. The temperature maps are presented every $\Delta t = 1250s$. For $t = 0$, all the nodes of the solid are at $T_0 = 8°C$ and, therefore, the block is isothermal. It starts to be plotted for $t = 1250s$.

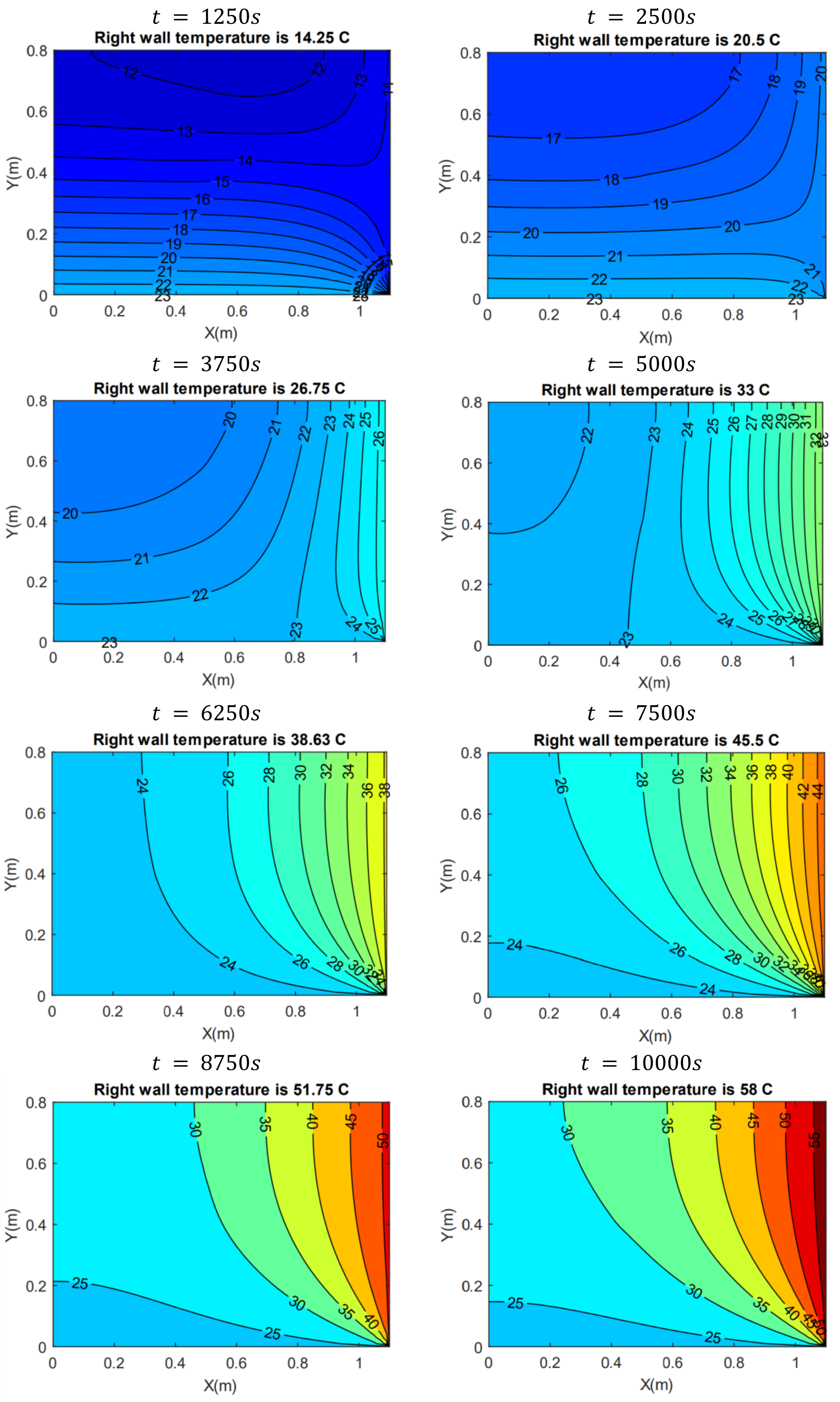


Figure 15. Evolution of the temperature map over time

The results obtained are consistent with the expected solution. As a first conclusion, it is observed that the isothermal walls have a stronger influence on the temperature distribution, i.e. the right and bottom wall.

At initial instants ($t \leq 1250s$) where a large part of the solid and the right wall still has a temperature close to T_0, a strong gradient is created close to the low wall.

After a while, for $2500s < t < 3750s$, practically all the solid is at the same temperature. That is because for this period of time T_{right} is very close to T_{bottom}.

From this moment on, T_{right} becomes higher than T_{bottom} and a gradient appears between these two walls. At points close to the shared corner the gradient is especially steep.

3.2.4 Code verification

A fundamental aspect to take into account when carrying out a programme that solves a problem numerically is to carry out its verification and validation. That is to say, to check that the program works correctly.

A first verification has already been performed on Figure 15 in the image for $t = 5000s$. It can be noticed that the map obtained is practically identical to the one obtained by the CTTC [12] shown below.

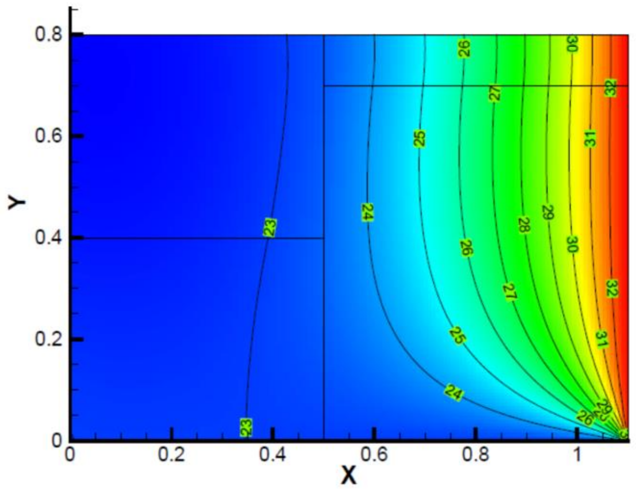


Figure 16. Temperature map for t = 5000s by CTTC

Another possible way to verify a code is through 0-dimensional analysis. That is, imposing an isothermal initial temperature map at $T = T_{ver}$ and imposing the same temperature boundary conditions. Thus, both walls are isothermal at $T_{wall} = T_{ver}$, the fluid on the left wall also at $T_g = T_{ver}$ and the top wall adiabatic ($\dot{q}_{flow} = 0$). If, as time evolves, the body temperature is maintained, the verification criterion is fulfilled. In the following figure, the body is shown for $t = 10000s$ and $T_{ver} = 20$°C.

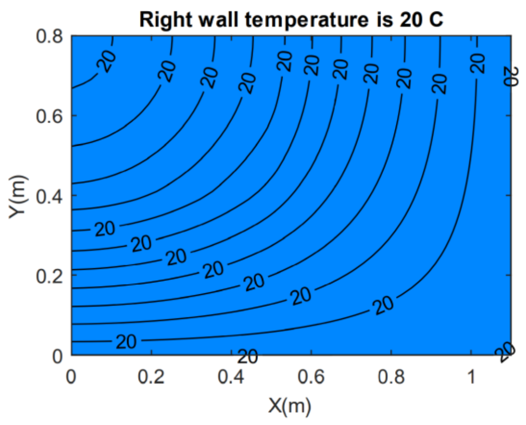


Figure 17. Temperature map for 0-dimensional analysis

Indeed, the whole body is at the same temperature after a considerable time.

Although these checks are necessary, they are not sufficient to prove that a code is correct. However, they do make it possible to state with some certainty that the results are close to reality.

3.3 Conclusions

The resolution of this first problem is a great introduction to numerical methods applied to a physical system. The four materials problem involves a two-dimensional, non-stationary treatment of the heat conduction equation with all kinds of boundary conditions and differences in the properties of the materials within the domain.

The first step is to discretise the heat conduction equations using the finite volume method. A fully implicit time scheme has been used to make the problem unconditionally stable. This has meant having to solve the system of equations presented for each time instant. As many systems have been solved as time increments up to the time of resolution.

A very important point is the treatment of boundary conditions. It has been explained how these constraints are translated into the composition of the system of equations. Furthermore, a strategic discretisation has been made to match the boundaries of the materials with those of the control volumes. In this way, the treatment of the volume properties has been made much simpler. The only property that has been taken into account in the material change has been the heat conduction coefficient using the harmonic mean.

Verification of the code is absolutely necessary to validate its use. In cases where the analytical or experimental solution is unknown, which in fact is the vast majority of numerically solved problems, it is difficult to perform such verification. In this case a solution verified by another author is provided and the result obtained with the developed code is practically identical. Therefore, it is assumed that the C++ program is validated. In cases where such a check is not possible, the code can be checked modularly with known solutions from other problems.

4

Convection-Diffusion

In this case, the methodology of numerical methods is applied to discretise the equations describing the convection-diffusion property of a fluid. In this report the resolution of these equations is focused on temperature fields. However, it is a great introduction to the Navier-Stokes equations when the property transported in the fluid is the velocity as will be treated in the following chapter.

A trivial one-dimensional stationary problem is presented to introduce and evaluate different numerical schemes. Next, two similar two-dimensional problems are presented to apply the equations to more complex cases.

4.1 Convection-diffusion equations

The equations describing the convection-diffusion phenomenon of a property ϕ can be discretised in much the same way as the heat conduction equations. Again, the objective is to find a system of linear equations where the unknowns are the quantity of ϕ at each node. In this case, Patankar's discretisation is being used described in [5].

4.1.1 Steady one-dimensional convection and diffusion

Again, the simplest problem for numerical discretisation is the steady, one-dimensional problem. Assuming incompressible flow and no external sources S. The governing equation is the following.

$$\frac{\mathrm{d}}{\mathrm{d}x}(\rho u\phi) = \frac{\mathrm{d}}{\mathrm{d}x}\left(\Gamma\frac{\mathrm{d}\phi}{\mathrm{d}x}\right) \tag{4.1}$$

Using the useful term $J = \rho u\phi - \Gamma\frac{\mathrm{d}\phi}{\mathrm{d}x}$ both convection and diffusion can be expressed as:

$$\frac{\mathrm{d}J}{\mathrm{d}x} = 0 \tag{4.2}$$

Integrating over the same control volume used to define the one-dimensional heat conduction discretisation,

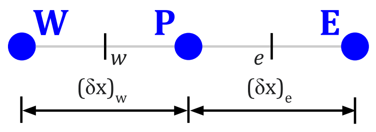


Figure 18. One-dimensional control volume dimensions

the result obtained is the following:

$$\begin{gathered} J_e - J_w = 0 \\ (\rho u)_e\phi_e - (\rho u)_w\phi_w = \left(\frac{\Gamma_e}{\delta x_e}\right)(\phi_E - \phi_P) - \left(\frac{\Gamma_w}{\delta x_w}\right)(\phi_P - \phi_W) \end{gathered} \tag{4.3}$$

In order to simplify the expression, let's use $F = \rho u$ and $D = \Gamma/\delta x$:

$$F_e\phi_e - F_w\phi_w = D_e(\phi_E - \phi_P) - D_w(\phi_P - \phi_W) \tag{4.4}$$

The main problem with this equation, compared to the conduction equation, is the need to work with the boundary value of the variable being searched (ϕ_e and ϕ_w). While for known values such as Γ_e this is not an inconvenient since the harmonic mean can be used, it can be problematic when it affects the main variable. This is where various resolution schemes come into play.

At first, the easiest solution may appear to be to make use of a simple average. Then, for example, ϕ_e would become $\phi_e = 1/2\ (\phi_E + \phi_P)$. This is a second order scheme known as Central Difference Scheme (CDS). However, this method is highly inaccurate and may tend to physically unrealistic values.

Another intuitive method is the first order Upwind Difference Scheme (UDS). In this case, as a function of the flow directions, ϕ_e would become $\phi_e = \phi_E$ for $F_e < 0$ or $\phi_e = \phi_P$ for $F_e > 0$. This way, the Equation (*4.4*) would become

$$a_P \phi_P = a_E \phi_E + a_W \phi_W \tag{4.5}$$

where

$$a_E = D_e + [\![-F_e, 0]\!] \qquad a_W = D_w + [\![F_w, 0]\!] \qquad a_P = a_E + a_W + (F_e - F_w) \tag{4.6}$$

Here, the operator $[\![A, B]\!]$ is used to denote the greater of A and B.

Even though this method does not tend to unrealistic values such as CDS, the results can also be too inaccurate. In order to obtain better solutions, some other methods are being presented. To keep simplifying expressions, the Peclet number P is used.

$$P = \frac{F}{D} = \frac{\rho u \delta x}{\Gamma} \tag{4.7}$$

Now, a general discretised equation that is used by several schemes can be reached using a function of Peclet $A(|P|)$.

$$a_P \phi_P = a_E \phi_E + a_W \phi_W \tag{4.8}$$

where

$$\begin{aligned} a_E &= D_e A(|P_e|) + [\![-F_e, 0]\!] \\ a_W &= D_w A(|P_w|) + [\![F_w, 0]\!] \\ a_P &= a_E + a_W + (F_e - F_w) \end{aligned} \tag{4.9}$$

Depending on the chosen scheme, the $A(|P|)$ function will take one of the following values:

Scheme	$A(\|P\|)$
Central difference	$1 - \|P\|/2$
Upwind difference	1
Hybrid difference	$[\![0, 1 - \|P\|/2]\!]$
Power law difference	$[\![0, (1 - 0.1\|P\|)^5]\!]$
Exponential (exact)	$\|P\|/[\exp(\|P\|) - 1]$

Table 3. Values of Peclet's function for different schemes

The schemes that have not yet been explained and are listed in the table are defined below.

- Hybrid Difference Scheme (HDS): This second order scheme is a combination between CDS for low velocities and UDS for high values of velocity.
- Power Law Difference Scheme (PLDS): It is a second order scheme that calculates the value of the boundary variable with a polynomial of fifth degree that is an approximation of the analytical solution.
- Exponential Difference Scheme (EDS): It is a second order scheme that provides the exact result of the problem. However, this scheme can only be used for one-dimensional, null source term and steady problems.

Apart from EDS, all the other schemes listed above can also be used to solve general transient and multidimensional problems. All of them are being utilized and compared in this report.

4.1.2 General case

To obtain the general discretised equations, one starts from the concepts explained for the one-dimensional case. Now the problem is transient and is presented in two dimensions.

Using Cartesian coordinates the velocity field becomes $\vec{v} = f(u, v)$. Therefore, the convection-diffusion equation is the following.

$$\frac{\partial(\rho\phi)}{\partial t} = -\frac{\partial J_x}{\partial x} - \frac{\partial J_y}{\partial y} + S \tag{4.10}$$

Where J_x and J_y are the convection-diffusion fluxes in each direction.

$$J_x = \rho u\phi - \Gamma\frac{\partial\phi}{\partial x} \qquad J_y = \rho v\phi - \Gamma\frac{\partial\phi}{\partial y} \tag{4.11}$$

The following figure illustrates a generic two-dimensional control volume.

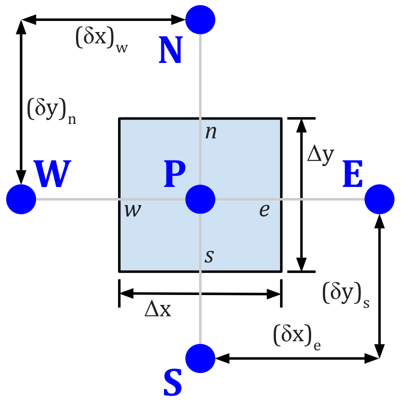


Figure 19. Two-dimensional control volume dimensions

Integrating over the control volume shown in Figure 19, this expression is reached.

$$\frac{(\rho_P\phi_P - \rho_P^0\phi_P^0)\Delta x\Delta y}{\Delta t} = -J_e + J_w - J_n + J_s + S_P\Delta x\Delta y \tag{4.12}$$

In a similar way, the continuity equation can also be integrated over the same volume.

$$\frac{(\rho_P - \rho_P^0)\Delta x\Delta y}{\Delta t} + F_e - F_w + F_n - F_s = 0 \tag{4.13}$$

where

$$F_e = (\rho u)_e\Delta y \qquad F_w = (\rho u)_w\Delta y \qquad F_n = (\rho v)_n\Delta x \qquad F_s = (\rho v)_s\Delta \mathrm{x} \tag{4.14}$$

Multiplying Equation (*4.13*) by ϕ_P and subtracting it from Equation (*4.12*), one obtains:

$$(\phi_P - \phi_P^0)\frac{\rho_P^0\Delta x\Delta y}{\Delta t} + (J_e - F_e\phi_P) - (J_w - F_w\phi_P) + (J_n - F_n\phi_P) - (J_s - F_s\phi_P) = S_P\Delta x\Delta y \tag{4.15}$$

Following a similar procedure than the one-dimensional method, the final discretisation equation is obtained. As before, the coefficients will depend on the resolution scheme chosen. Also note that a fully implicit methodology is used to study the transient problem.

$$a_P\phi_P = a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + b_P \tag{4.16}$$

The coefficients can be evaluated as:

$$\begin{aligned} a_E &= D_e A(|P_e|) + [\![-F_e, 0]\!] \\ a_W &= D_w A(|P_w|) + [\![F_w, 0]\!] \\ a_N &= D_n A(|P_n|) + [\![-F_n, 0]\!] \\ a_S &= D_s A(|P_s|) + [\![F_s, 0]\!] \\ a_P &= a_E + a_W + a_N + a_S + \frac{\rho_P^0 \Delta x \Delta y}{\Delta t} \\ b_P &= \left(S_P + \frac{\rho_P^0}{\Delta t}\right)\Delta x \Delta y \end{aligned} \tag{4.17}$$

where

$$D_e = \frac{\Gamma_e \Delta y}{\delta x_e} \qquad D_w = \frac{\Gamma_w \Delta y}{\delta x_w} \qquad D_n = \frac{\Gamma_n \Delta x}{\delta y_n} \qquad D_s = \frac{\Gamma_s \Delta \text{x}}{\delta y_s} \tag{4.18}$$

Table 3 presents the values of $A(|P|)$ for each resolution scheme.

4.2 1D convection-diffusion flow

This is the first convection-diffusion problem. It considers a one-dimensional flow moving from point X_0 to point X_1 at different temperatures. The velocity, density and diffusion factor are considered constant.

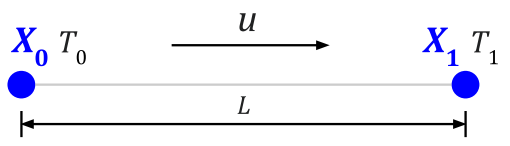


Figure 20. One-dimensional flow scheme

The numerical solution of this problem may seem to be of no importance since the exact analytical solution of the problem is known. The following expressions describes how the temperature evolves along the x-axis.

$$T(x) = T_0 + (T_1 - T_0)\frac{\exp(Px/L) - 1}{\exp(P) - 1} \tag{4.19}$$

Where P is the Peclet number defined as $P = \rho u L/\Gamma$.

In fact, since the exact solution is available, it is the ideal case to validate, verify and evaluate the different numerical methods presented in Table 3 used to solve this kind of problems.

4.2.1 The exact solution

The analytical solution of this trivial problem is also useful helping to understand how the temperature distribution is affected by changes in the defining variables. As seen in Equation (*4.19*), the temperature distribution only depends on one parameter: the Peclet number.

Below are shown the different solutions obtained for different Peclet numbers. The distance between the two points is considered to be $L = 1m$ and the boundary temperatures are $T_0 = 0°C$ and $T_1 = 100°C$.

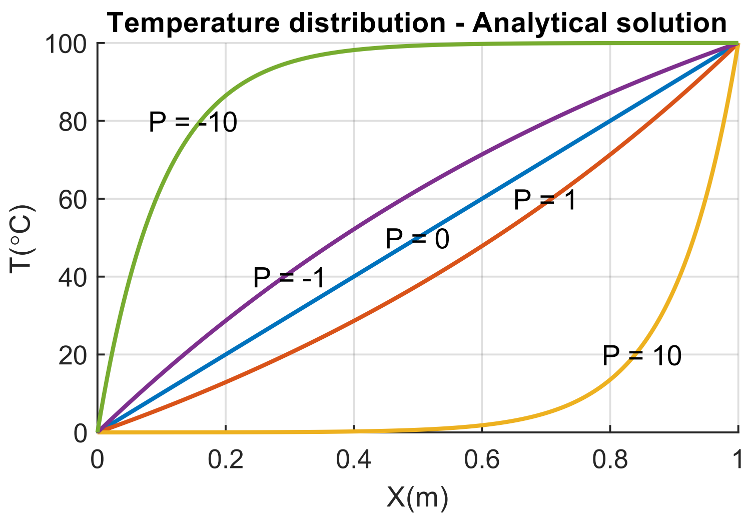


Figure 21. One-dimensional convection-diffusion problem solution for different P values

For low absolute values of P the diffusion phenomenon is stronger than the advection produced by the velocity and the temperature transition is quite uniform. Indeed, for P=0, where the velocity can be considered to be null, the temperature evolution is completely linear.

In contrast, for more extreme values of P, the advection effect becomes much more relevant. In this case, the prevailing temperature in the domain becomes that of the point from which the flow originates. For example, for $P = 10$, where the velocity flow is positive (from point X_0 to X_1), the temperature remains almost constant at T_0 until past the middle of the x-axis where it starts to increase sharply to T_1.

4.2.2 Numerical schemes comparison

Once the analytical problem is solved, one can proceed to the solution of the problem by means of the four presented schemes: Upwind (UDS), Central difference (CDS), Hybrid (HDS) and Power law (PLDS). The main objective is to find out the best performing scheme so it can be used for the following problems.

In order to carry out the study, the problem will be solved for $P = 30$ as the different methods diverge and vary between them in greater proportion for higher values of P. In addition, the aim is to observe how the different methods evolve as a function of the number of nodes. Thus, discretisations of 10, 20, and 40 nodes are used.

The results obtained are presented below together with the analytical solution in dashed line. The analytical solution has been computed using the Peclet's Function formulation presented above. This way, the temperature has been calculated node-by-node and not as a continuous function for a better comparison. Also note that the domain is the same than before (from $x = 0m$ to $x = 1m$ and from $T_0 = 0°C$ to $T_1 = 100°C$) but the plot is zoomed in the critical part of the curve where the temperature varies more sharply.

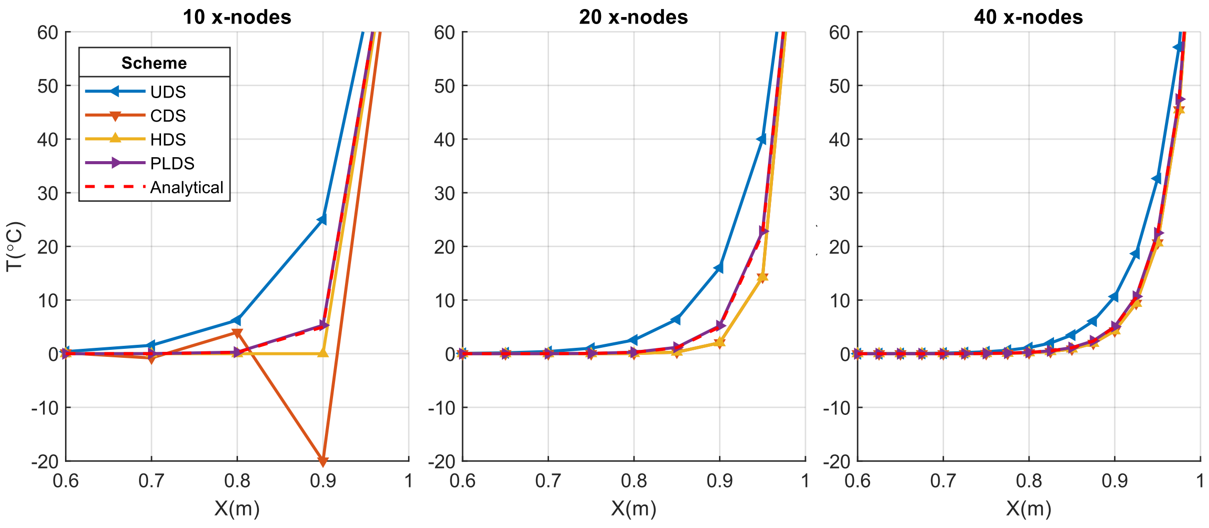


Figure 22. Comparison of convection-diffusion resolution schemes

It can be observed that the results are significantly different for each scheme. However, there is a common factor in all of them: the more nodes are used, the more they converge and tend to the analytical solution. This table presents the root-mean-square error for each scheme with regard to the analytical solution so that the differences can be quantified.

Number of	RMS error in relation to analytical solution			
x-nodes	UDS	CDS	HDS	PLDS
10	2.5539	1.9915	0.4763	0.0330
20	1.8069	0.5740	0.5740	0.0409
40	1.0685	0.1510	0.1510	0.0187

Table 4. Root-mean-square error for different resolution schemes

4.2.3 Conclusion and analysis of the different schemes

Major conclusions can be drawn from both the error values and the graphical plots for each type of scheme.

- The Upwind difference scheme (UDS) presents a physically realistic but also very inaccurate results. This scheme is the one more different and distant from the analytical solution where the number of nodes is increased and presents the largest errors. Therefore, it is not considered a recommended method for complex or detailed cases.
- The Central difference scheme (CDS) provides the most surprising result. For a temperature distribution between 0°C and 100°C there is one node with a lower temperature than the minimum ($T_i \approx -20$°C). In fact, this is the main issue of this method, although for an increase of nodes the results may seem accurate, there is a danger of obtaining completely physically unrealistic results. Thus, this method is not recommended either, especially if few nodes are used.
- The Hybrid difference scheme (HDS) is a reliable and realistic method. The solution obtained is satisfactory for both few and many nodes. An important point to note is that for the solutions with 20 and 40 nodes the solution is identical to the result obtained by CDS. This result is completely reasonable since the hybrid method uses the same formulation as the central difference method for low local Peclet numbers or, in other words, for low δx. This way, it achieves CDS accuracy but avoids physically impossible results.
- The Power law difference scheme (PLDS) clearly achieves the best results. Even with the minimum number of nodes, the solution is practically identical to the analytical expression. The errors obtained are an order of magnitude lower than any other scheme. This is due to the fifth-degree polynomial which closely matches the exponential evolution of temperature that this method uses.

Hence, the conclusion is clear. The scheme to be used to solve the rest of the convection-diffusion problems must be PLDS. It is not only the most accurate but also does not require extra computational resources.

4.3 Diagonal flow

The first two-dimensional convection-diffusion problem is the study of a diagonal flow within a rectangular control volume.

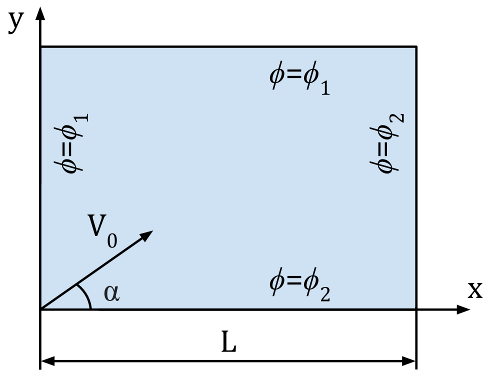


Figure 23. Diagonal flow scheme

The direction of the flow is parallel to the diagonal of the rectangle going from bottom left to top right. Therefore, the value of the angle α is the one that satisfies this condition. The velocity field is uniform throughout the control volume and constant in time. The velocity vector $\vec{v}$ can be stated as follows.

$$\vec{v} = [V_0 \cos(\alpha)\,,\ V_0 \sin(\alpha)] \tag{4.20}$$

The boundary conditions are simple: all walls have an imposed value of the generic property ϕ. The walls above the diagonal (left and top) have a value $\phi = \phi_1$ while those below (right and bottom) satisfy that $\phi = \phi_2$.

To solve a problem numerically the variables must have numerical values. Thus, the convection-diffusion equation is being applied to the energy conservation equation and the generic property ϕ becomes temperature T. All numerical values used in this problem are proposed below.

$T_1 = 100.0°\text{C}$	$X = 3.0m$	$V_0 = 10m/s$	$\Gamma = 10\ kg/ms$
$T_2 = 0.0°\text{C}$	$Y = 2.0m$	$\alpha = 33.69°$	$\rho = 1.225\ kg/m^3$

Table 5. Numerical values for Diagonal Flow Problem

4.3.1 Resolution equations

This is a two-dimensional convection-diffusion problem so the general discretised equation obtained in (4.16) is used. However, note that this a steady state problem without any internal source. Hence, the general equation of an internal node is considerably reduced.

The Power Law Difference Scheme is used since it is the most accurate scheme that has been studied. Therefore, the value of $A(|P|)$ must be replaced by the one described in Table 3. The resulting equation for a generic internal node is the following.

$$a_P T_P = a_E T_E + a_W T_W + a_N T_N + a_S T_S \tag{4.21}$$

The coefficients can be evaluated as:

$$
\begin{aligned}
a_E &= D_e[\![0,(1-0.1|P_e|)^5]\!] + [\![-F_e,0]\!]\\
a_W &= D_w[\![0,(1-0.1|P_w|)^5]\!] + [\![F_w,0]\!]\\
a_N &= D_n[\![0,(1-0.1|P_n|)^5]\!] + [\![-F_n,0]\!]\\
a_S &= D_s[\![0,(1-0.1|P_s|)^5]\!] + [\![F_s,0]\!]\\
a_P &= a_E + a_W + a_N + a_S
\end{aligned}
\tag{4.22}
$$

4.3.1.1 Wall nodes

The value of temperature at the wall nodes T_P is imposed. Thus, the equation to be introduced in the program is reduced to $a_P T_P = b_P$ where:

$$b_P = T_{wall} \qquad a_P = 1 \tag{4.23}$$

4.3.2 Results

Once the coefficients that are constant over time have been calculated, a line-by-line solver is used to solve the problem.

The most relevant numerical data is the mesh size. For this case a mesh of 180 $\times$ 120 nodes has been chosen following the proportions of the rectangle. In this way, all the control volumes are squared, slightly simplifying the calculations, For the convergence factor $\delta = 0.001$ is used.

The first result presented is for the most general case. It assumes the physical data presented in the introduction of the problem. This data permits to work with a Peclet number close to one. Thus, advection and diffusion effects act in the same proportion.

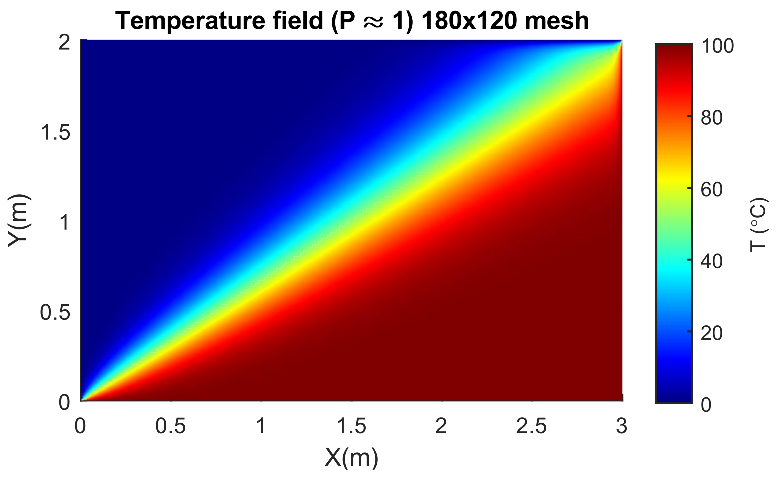


Figure 24. Temperature field distribution for a general diagonal flow problem

Two clear regions can be observed in the rectangle. The temperature of the one above the diagonal significantly tends to the wall temperature ($T_1 = 0$°C) and so does the temperature for the inferior region, tends to $T_2 = 100$°C.

Another aspect to be noted is how the transition between regions evolves in the diagonal. While in the lower left points the transition is abrupt, in the higher right points the diffusion is easily observed.

4.3.3 False diffusion for high Peclet

In convection-diffusion problems there are two extreme cases: those in which Peclet number tends to infinity (the is only advection) and those in which it is zero (only diffusion). In this case, the influence of mesh size for $P = \infty$ is being studied.

The false diffusion phenomenon appears in the numerical solution of convection-diffusion equations. It is due to the approximation made to the convection term typically when upwind scheme is used. It is a drawback to be taken into account when solving this type of problem as it can lead to erroneous results.

The analytical (and intuitive) solution of the problem states that all points above the diagonal will be at temperature T_1 while all points below will be at temperature T_2. Results for the diagonal flow problem with $P = \infty$ with different mesh densities are presented.

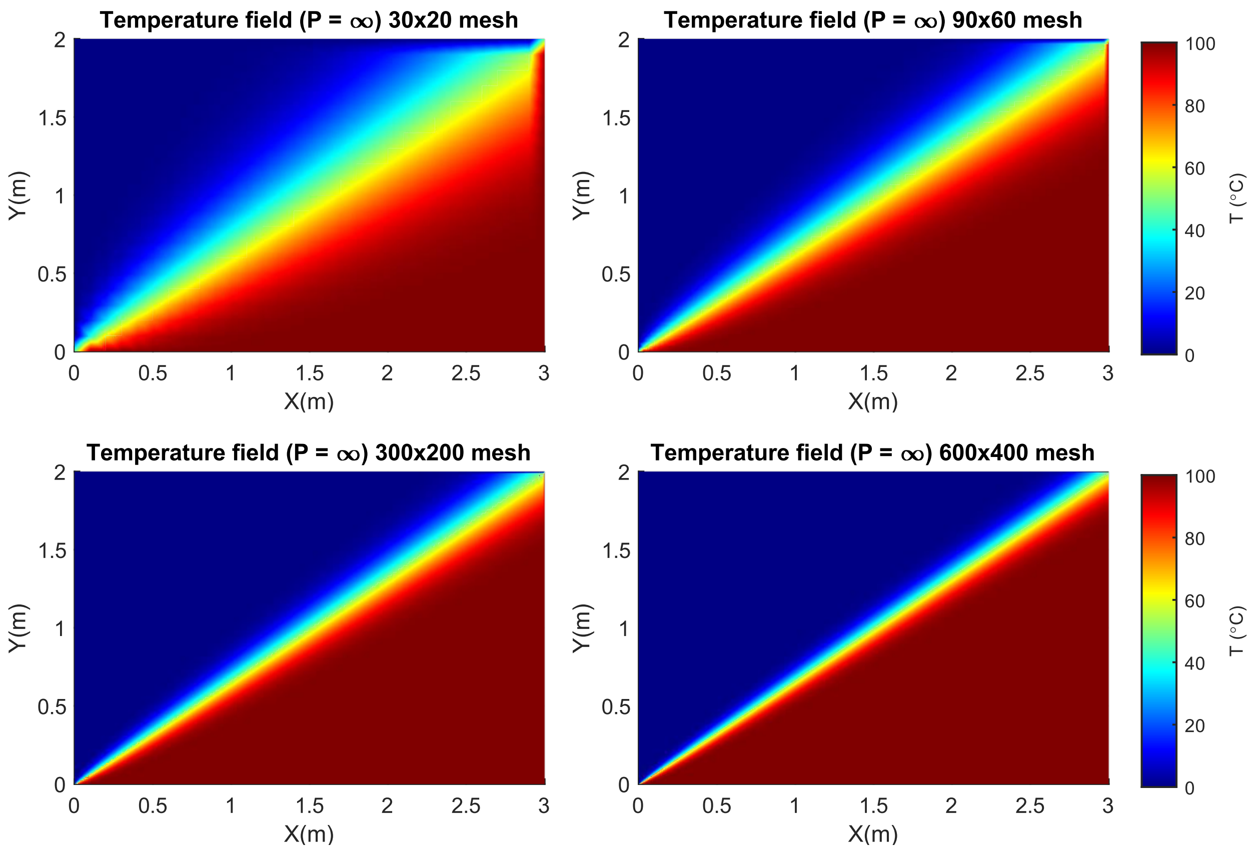


Figure 25. Mesh size comparison for infinite Peclet number

It can be clearly observed how that in the first figure (30×20 mesh) the diffusion phenomenon is much more noticeable than in the last one (600×400 mesh). What is really appreciated is a false diffusion caused by the truncation errors when the result is computed numerically since they are all representing the same physical problem.

To make these results even clearer, Figure 26 plots the temperature of each of each of these results for the straight line $y = 1m$. It can be noticed that at the central point all the curves are at the average temperature ($T_m = 50°\text{C}$). However, while the temperature of the meshes with many nodes tends to wall temperatures quickly, the curve for the less dense meshes recovers much more slowly.

This result highlights the importance of using a properly sized mesh. Even though this case is particularly critical due to its parameters of boundary conditions, it demonstrates that choosing too few nodes may result in a gain in computational speed but can also lead to incorrect or inaccurate results. This is why verification and validation processes are so important.

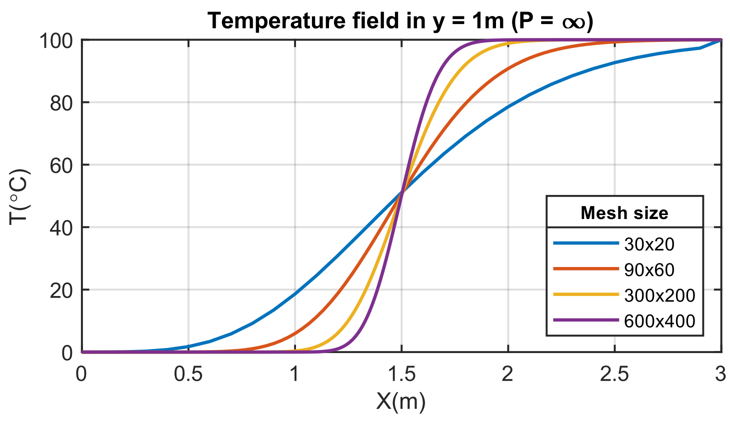


Figure 26. Temperature distribution for y = 1m for different mesh sizes

4.3.4 Natural convection

Finally, the result for the natural convection problem is presented. In this case, the flow velocity is zero ($V_0 = 0$). Thus, the Peclet number is also null ($P = 0$).

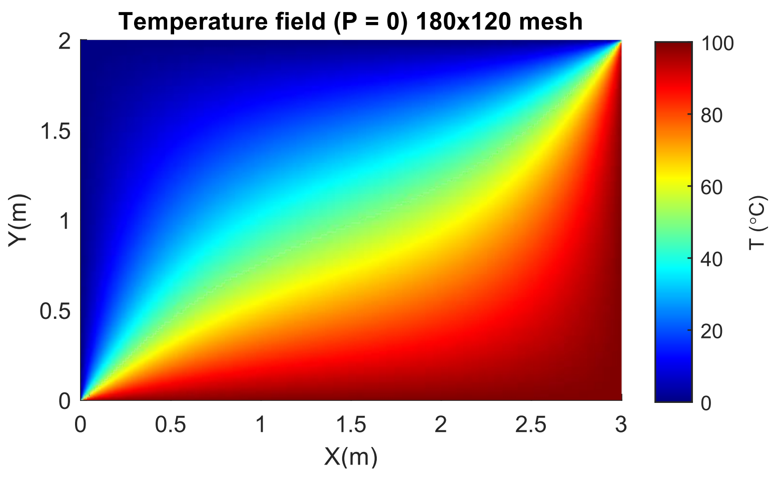


Figure 27. Temperature field for null velocity

The lack of velocity permits a much smoother temperature transition between the walls.

The most important conclusion of this problem is the requirement of good judgment needed to choose the numerical parameters when solving this type of problem. It has been clearly demonstrated how the choice of an incorrect mesh density leads to results that are far from reality.

It has also been shown in a two-dimensional case how the Peclet number affects the problem in the two most extreme cases, $P = 0$ and $P = \infty$. It must be noted how the behavior of temperature and heat is completely different.

4.4 Smith-Hutton problem

The last convection-diffusion problem is the study of a solenoidal flow. This exercise is also known as the Smith-Hutton problem, since it was developed by these two authors in order to study the performance of different numerical methods [13].

The domain of study is again rectangular and is steady with respect to time. Inside the control volume, where the third dimension can be neglected, the velocity field if given by the following expression.

$$\vec{v}(x,y) = [2y(1-x^2), -2x(1-y^2)] \tag{4.24}$$

The following figure illustrates the physical dimensions of the problem and the behaviour of the flow described by the expression.

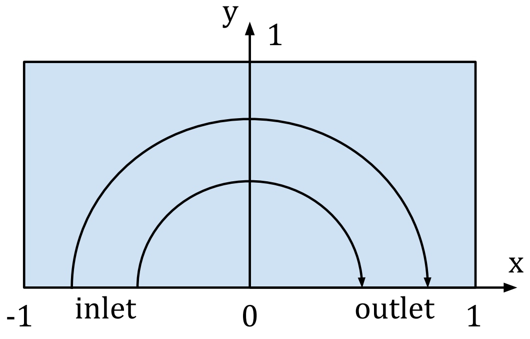


Figure 28. Smith-Hutton problem representation

Boundary conditions can be divided into three main parts. The side ($x = \pm 1m$) and top ($y = 1m$) walls follow the below expression which is designed to impose a temperature value very close to zero.

$$T_w = 1 - \tanh[\alpha] \tag{4.25}$$

The bottom horizontal wall ($y = 0$) is divided into the two remaining parts: the inlet for $x < 0m$ and the outlet for $x > 0m$. The expression for the inlet describes a temperature variation in the horizontal direction such that at the left end the temperature has a value tending to zero while in the center it tends to $T = 2$°C. In addition, the objective of using this equation is that the temperature change occurs abruptly for $x = -0.5m$.

$$T_{inlet} = 1 + \tanh[(2x+1)\alpha] \tag{4.26}$$

The outlet wall ($y = 0m$ and $x > 0m$) is considered adiabatic, therefore the expression is

$$\frac{\partial T_{outlet}}{\partial y} = 0 \tag{4.27}$$

In fact, the boundary conditions presented by Smith-Hutton were using a generic unknown ϕ. However, as has already been discussed in other sections, the use of a real physical incognita such as temperature is preferable when solving and interpreting the problem. Apart from this, the only numerical value required is the factor $\alpha = 10$ and the relation ρ/Γ that will be the only variable of the problem.

4.4.1 Resolution equations

The distribution of the mesh nodes is absolutely the same as in the diagonal flow problem. Thus, the equations describing the temperature of the internal nodes is identical to the one described in Section 4.3.1. Again, the Power Law Difference Scheme is being employed for the resolution.

4.4.1.1 Side and top walls

The temperature remains constant both in time and position for these nodes. Therefore, the expression of the coefficients gets reduced to:

$$a_P = 1 \qquad b_P = T_w = 1 - \tanh[\alpha] \tag{4.28}$$

4.4.1.2 Inlet

The temperature of the inlet is also imposed but its value depends on the x-axis position.

$$a_P = 1 \qquad b_P = T_{inlet}(x_P) = 1 + \tanh[(2x_P + 1)\alpha] \tag{4.29}$$

4.4.1.3 Outlet

The outlet zone of the domain is considered adiabatic in the vertical direction. Therefore, all the coefficients will be the same than the ones for internal nodes except for the a_S coefficient which describes the effect of the outside temperature. As there is no heat flux,

$$a_S = 0 \tag{4.30}$$

4.4.2 Results

As mentioned above, once all constraints have been imposed, the only factor that varies the result of the temperature field is the ρ/Γ ratio. This relation is equivalent to the Peclet number but, however, in this problem it is preferable the use of the ratio due to the geometry of the velocity field.

The first result is presented here for $\rho/\Gamma = 10$. A 100×50 nodes mesh has been employed respecting the proportions of the domain with a convergence factor $\delta = 0.0001$.

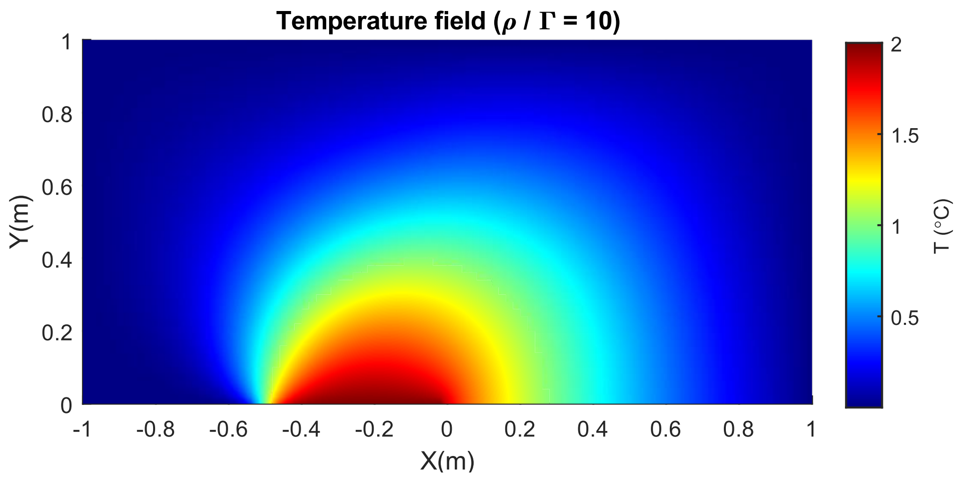


Figure 29. Temperature field for a dominant diffusion in solenoidal flow

With this temperature distribution, the tendency of the flow to rotate can be observer since a circular shape can be perceived. Nodes near the side and top walls have a temperature that distinctly tends to that of wall $T_{wall} = 0°C$.

As is evident, the inlet exhibits the imposed distribution: starting from $x = -1m$,temperature tends to zero up to $x = -0.5m$ and tends to 2°C up to $x = 0$. However, the behavior in the outlet is clearly different. Temperature decays smoothly and progressively. No symmetry is observed on the horizontal axis where it could be expected.

It must be noted that the ratio ρ/Γ utilized is relatively low and, thus, diffusion dominates over advection. The case shown now below is for a ratio one hundred times higher. In this case, it is assumed that advection gains a very important role. The same numerical parameters for mesh and convergence have been used.

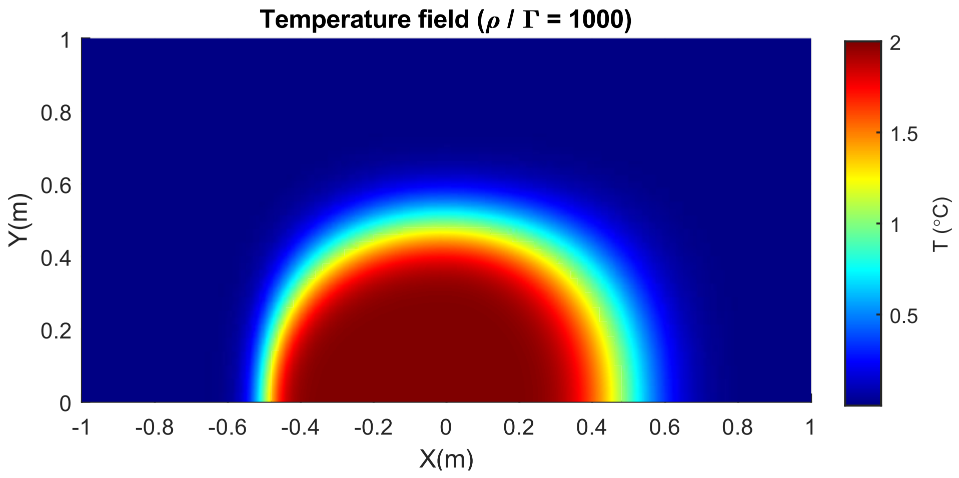


Figure 30. Temperature field for a dominant advection in solenoidal flow

The differences are quite visible. In this solution one can observe a semi circumference centred at origin with a unitary radius where a strong temperature gradient appears in radial direction. Inside the semicircle almost all node temperatures tend to 2°C while outside they tend to 0°C (wall temperature).

In this case, a radial symmetry and an axial symmetry to the $x = 0m$ axis is observed. However, the temperature gradient becomes weaker as it moves away from its origin at the inlet. This is due to the effect of diffusion, which, although much reduced, still affects the temperature distribution.

The relevance of the relationship between convection and advection is again emphasized. Although they are two concepts that are usually studied together, the fact that one is imposed on the other completely affects the result.

The objective of the Smith-Hutton case was to compare different numerical resolution methods in a problem "academically" perfect like this one. Therefore, they do not present a single result with which to compare the one obtained in this paper. However, it can be stated that the results obtained are very similar and consistent with those expected.

4.5 Conclusions

The convection-diffusion equations describe how the quantity of a property is transmitted within a physical system by convection and diffusion. The property being treated can be represented either by a scalar field, as would be the case for temperature distributions, or by a vector field, such as velocity.

In this section, the discretised equations have been presented using Patankar's nomenclature for any scalar property. The problems consisted of solving the temperature field in a stationary fluid flow.

The one-dimensional problem has made it possible to verify part of the code used to solve the other two, as its analytical solution is known. In addition, different first and second order numerical discretisation schemes have been compared in order to work with the one that has given the best results.

The diagonal flow and Smith-Hutton problems have produced the expected results. In these problems, the importance of choosing the right mesh size has been emphasised, as it has been observed that the diffusion effect is highly sensitive to the used accuracy.

It has also been observed that the lack of a conductive term in the treatment of the temperature of a fluid results in a natural convection phenomenon. In this case, the results obtained are very similar to those of heat conduction in solids.

Now the next challenge is to study the convection-diffusion equations when the transported variable is the velocity. In this section, the temperature field has been solved assuming that the velocity field is imposed and does not vary. When attempting to solve the velocity field, one tries to find a variable that is transported by itself. This is where the Navier-Stokes equations appear and where their great difficulty in being solved lies.

5

Navier-Stokes

Navier-Stokes is a system of non-linear partial differential equations describing the motion of a fluid. Its analytical solution has not yet been obtained due to its complexity. It is therefore the focus of all CFD research.

There are several methods for approaching the numerical solution of these equations. The Fractional Step Method is presented in this report. It is an algorithm to solve the equations for incompressible flows that has demonstrated high performance and simplicity of implementation.

The scheme of resolution is applied to the lid-driven cavity problem. A canonical case for which the results are known and which is therefore of great use in verifying the code.

5.1 Fractional Step Method

There exist many numerical methods to solve or simulate the Navier Stoke equations. When it comes to solving the incompressible problem, the Fractional Step Method (FSM) is one of the best performing in spite of its simplicity when applied.

The FSM uses a projection of the velocity field onto a divergence-free space. The objective is to obtain a velocity predictor as an approximation of the solution of the momentum equations. This predictor is found using the Helmholtz-Hodge theorem without taking into account the pressure gradient and therefore cannot satisfy the incompressibility constraint at the next time level. The Poisson equation solves this problem by determining the minimum perturbation that will make the velocity predictor incompressible.

5.1.1 Helmholtz-Hodge theorem into NS equations

First of all, the governing equations are presented for incompressible Newtonian fluids. This equation has been previously discussed in Equation (2.15).

$$\frac{\partial \vec{v}}{\partial t} = \frac{1}{Re}\nabla^2\vec{v} - (\vec{v}\cdot\nabla)\vec{v} - \nabla p \tag{5.1}$$

$$\nabla\cdot\vec{v} = 0 \tag{5.2}$$

where Re is the Reynolds number $Re = \rho V_0 L/\mu$ using characteristic length L and velocity V_0. By putting together the convective and diffusive terms from Equation (5.1) one obtains:

$$R(\vec{v}) = \frac{1}{Re}\nabla^2\vec{v} - (\vec{v}\cdot\nabla)\vec{v} \tag{5.3}$$

Now, Equations (5.1) and (5.2) can be integrated over time as follows.

$$\int_{t^n}^{t^{n+1}} \frac{\partial \vec{v}}{\partial t}\mathrm{d}t = \int_{t^n}^{t^{n+1}} R(\vec{v})\mathrm{d}t - \int_{t^n}^{t^{n+1}} \nabla p\mathrm{d}t \tag{5.4}$$

$$\int_{t^n}^{t^{n+1}} \nabla\cdot\vec{v}\,\mathrm{d}t = 0 \tag{5.5}$$

The $R(\vec{v})$ term is being integrated using a fully explicit second-order Adams-Bashforth scheme.

$$R^{n+1/2}(\vec{v}) = \frac{3}{2}R(\vec{v}^n) - \frac{1}{2}R(\vec{v}^{n-1}) \tag{5.6}$$

For the time derivative term, a simple central difference scheme is employed and for the pressure term a Euler scheme is used. The incompressibility constraint is integrated using an implicit scheme. The following result is obtained.

$$\frac{\vec{v}^{n+1} - \vec{v}^n}{\Delta t} = \frac{3}{2}R(\vec{v}^n) - \frac{1}{2}R(\vec{v}^{n-1}) - \nabla p^{n+1} \tag{5.7}$$

$$\nabla\cdot\vec{v}^{n+1} = 0 \tag{5.8}$$

This is where the FSM projection come into play. This method is based on the Helmoltz-Hodge theorem which is, in a slightly simplified version, defined as follows [12]: *A given vector field* ω*, defined in a bounded domain* Ω *with smooth boundary* $\partial\Omega$*, is uniquely decomposed in a pure gradient field and a divergence-free vector parallel to* $\partial\Omega$*.*

$$\vec{\omega} = \vec{a} + \nabla\varphi \tag{5.9}$$

where

$$\begin{aligned} \nabla \cdot \vec{a} = 0 \quad & \vec{a} \in \Omega \\ \vec{a} \cdot \vec{n} = 0 \quad & \vec{a} \in \partial\Omega \end{aligned} \tag{5.10}$$

For the case concerning this report, the projected vector $\vec{\omega}$ is the velocity predictor $\vec{v}^P$. It becomes the combination of the divergence-free velocity vector $\vec{v}^{n+1}$ and the gradient of a scalar field $\nabla\tilde{p}$.

$$\vec{v}^P = \vec{v}^{n+1} + \nabla\tilde{p} \tag{5.11}$$

In this equation $\tilde{p}$ stands for pseudo-pressure as $\tilde{p} = \Delta t p^{n+1}$. Since the flow is incompressible, the H-H theorem is satisfied as $\nabla \cdot \vec{v}^{n+1} = 0$. By substituting $\vec{v}^{n+1}$ obtained in Equation (5.11) into Equation (5.7) the following result is obtained where the pressure terms disappear.

$$\vec{v}^P = \vec{v}^n + \Delta t \left(\frac{3}{2} R(\vec{v}^n) - \frac{1}{2} R(\vec{v}^{n-1}) \right) \tag{5.12}$$

This result has a great relevance, because it allows to find $\vec{v}^P$ without the need to solve any system because it is the only unknown.

Finally, by deriving the Equation (5.11) the Poisson equation to find pressure is achieved.

$$\Delta\tilde{p} = \nabla \cdot \vec{v}^P \tag{5.13}$$

Once both $\vec{v}^P$ and $\tilde{p}$ are reached, the actual velocity field $\vec{v}^{n+1}$ can be find taking pressure gradient correction into account using Equation (5.11) again.

$$\vec{v}^{n+1} = \vec{v}^P + \nabla\tilde{p} \tag{5.14}$$

To solve the problem, these steps will be followed:

1. Compute $R(\vec{v}^n)$ using Equation (5.3)
2. Evaluate $\vec{v}^P$ from Equation (5.12)
3. Compute $\nabla \cdot \vec{v}^P$and solve the discrete Poisson Equation (5.13)
4. Get the velocity field $\vec{v}^{n+1}$ using Equation (5.14)

It must be noted that the only step that require to solve a system is the third with the resolution of the Poisson equation. All the other formulation is fully explicit. Therefore, is of great importance to solve efficiently the Poisson equation.

5.1.2 Staggered meshes

There is a problem in calculating the velocity field $\vec{v}^{n+1}$ in the fourth step presented. Let's assume the following one-dimensional scenario where finite volumes at node P have been applied.

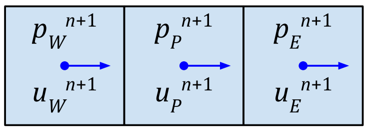


Figure 31. One-dimensional finite volumes discretisation

The following expression shows how the velocity coordinate u^{n+1} on the x-axis is computed.

$$u^{n+1} = u^P - \frac{\Delta t}{\rho}\left(\frac{p_E^{n+1} - p_W^{n+1}}{2\Delta x}\right) \tag{5.15}$$

It can be noticed how the discretisation of $\nabla\tilde{p}^{n+1}$ does not depend on the pressure p_P^{n+1} at node P. This setting can lead to totally unrealistic results such as the one presented in the following figure [14].

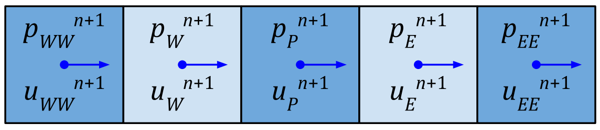


Figure 32. One-dimensional unrealistic scenario

$$p_{WW}{}^{n+1} = 100$$
$$p_W{}^{n+1} = 0$$
$$p_P{}^{n+1} = 100$$
$$p_E{}^{n+1} = 0$$
$$p_{EE}{}^{n+1} = 100$$

Although this result is not physically possible, it fulfils the condition$\nabla p^{n+1} = 0$. Thus, a better strategy is required to address the problem of finding $\vec{v}^{n+1}$. This inconvenience is known as *the checkerboard problem* and it is here where staggered meshes come into play.

Staggered meshes are widely used to solve academic problems since they perform quite well and are easy to implement on structured meshes. These meshes add new nodes to the control volume walls of the main mesh as shown in the following illustration.

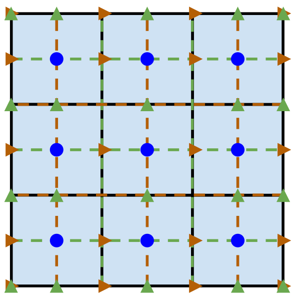


Figure 33. Main (blue) and staggered (green and brown) meshes

In this way, the pressure field will be evaluated in the main mesh while velocity field will make use of the staggered meshes. The x component of velocity is computed in the brown mesh while the green mesh is for the y component. The steps to be followed to solve this problem by means of this method are described in the following section.

5.1.3 FSM discretised equations

Following the steps described in Section 5.1.1, the first thing to do is to find $R(\vec{v}^n)$. Thus, Equation (5.3) is integrated into the control volume Ω_{xP} and Ω_{yP} for each direction of the staggered meshes.

$$R(u)\Omega_{xP} = -(\dot{m}_e u_e - \dot{m}_w u_w + \dot{m}_n u_n - \dot{m}_s u_s) + \left(\mu_e \frac{u_E - u_P}{d_{EP}} S_e - \mu_w \frac{u_P - u_W}{d_{WP}} S_w + \mu_n \frac{u_N - u_P}{d_{NP}} S_n - \mu_s \frac{u_P - u_S}{d_{SP}} S_s\right) \tag{5.16}$$

$$R(v)\Omega_{yP} = -(\dot{m}_e v_e - \dot{m}_w v_w + \dot{m}_n v_n - \dot{m}_s v_s) + \left(\mu_e \frac{v_E - v_P}{d_{EP}} S_e - \mu_w \frac{v_P - v_W}{d_{WP}} S_w + \mu_n \frac{v_N - v_P}{d_{NP}} S_n - \mu_s \frac{v_P - v_S}{d_{SP}} S_s\right) \tag{5.17}$$

where

$$\dot{m}_e = (\rho u)_e S_e \qquad \dot{m}_n = (\rho u)_n S_n \qquad \dot{m}_w = (\rho u)_w S_w \qquad \dot{m}_s = (\rho u)_s S_s \tag{5.18}$$

Here, the first question that arises is how the volumetric flow rate and the transport property can be evaluated. In other words, and focusing on the x-axis direction, which values of $(\rho u)_i$ and u_i are to be taken for the calculation? The following figure obtained and modified from [14] presents a scheme where these variables are represented.

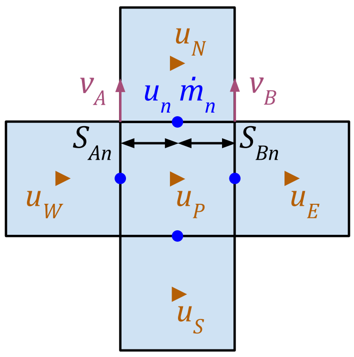


Figure 34. Velocities in the stagg-x mesh

The velocity through all the walls u_i is being evaluated as it was for the convection-diffusion equations. Several schemes can be used such as UDS or CDS for low-order schemes or QUICK or SMART for high-order schemes.

For the direction parallel to the studied axis, in this case the horizontal direction parallel to the x-axis, the mass flow rate can be calculated by averaging through the nodes as follows.

$$\dot{m}_e = \frac{(\rho u)_E + (\rho u)_P}{2} S_e \tag{5.19}$$

Note that the same formulation can be extrapolated to compute $\dot{m}_w$. For the nodes above and below, the vertical component of the velocity in the staggered mesh will be used and the mass flow will have this form. Again, this is equivalent for $\dot{m}_s$.

$$\dot{m}_n = (\rho v)_A S_{An} + (\rho v)_B S_{Bn} \tag{5.20}$$

All these procedures can also be easily extrapolated to y-axis and therefore the explanation is omitted.

Once $R(u)$ and $R(v)$ are reached, the following step is to obtain the velocity predictors for both axes. This can be done by integrating in the staggered meshes the Equation (5.12). The resulting expressions for each direction are here presented.

$$u^P = u^n + \frac{\Delta t}{\rho}\left(\frac{3}{2}R(u^n) - \frac{1}{2}R(u^{n-1})\right) \tag{5.21}$$

$$v^P = v^n + \frac{\Delta t}{\rho}\left(\frac{3}{2}R(v^n) - \frac{1}{2}R(v^{n-1})\right) \tag{5.22}$$

Next, the objective is to evaluate the pressure field. As it has been already stated, in this case the main grid will be employed. Thus, the Poisson Equation (5.13) is discretised by integrating over the control volume of the main grid Ω. This expression is obtained.

$$\begin{aligned} \frac{p_E^{n+1} - p_P^{n+1}}{d_{EP}}S_e - \frac{p_P^{n+1} - p_W^{n+1}}{d_{WP}}S_w + \frac{p_N^{n+1} - p_P^{n+1}}{d_{NP}}S_n - \frac{p_P^{n+1} - p_S^{n+1}}{d_{SP}}S_s \\ = \frac{1}{\Delta t}((\rho u^P)_e S_e - (\rho u^P)_w S_w + (\rho u^P)_n S_n - (\rho u^P)_s S_s) \end{aligned} \tag{5.23}$$

To find the pressure at each node one uses the structure of coefficients applied before to calculate temperatures. Therefore, Equation (5.23) becomes:

$$a_P p_P^{n+1} = a_E p_E^{n+1} + a_W p_W^{n+1} + a_N p_N^{n+1} + a_S p_S^{n+1} + b_P \tag{5.24}$$

where

$$\begin{gathered} a_E = \frac{S_e}{d_{EP}} \qquad a_W = \frac{S_w}{d_{WP}} \qquad a_N = \frac{S_n}{d_{NP}} \qquad a_S = \frac{S_s}{d_{SP}} \\ a_P = a_E + a_W + a_N + a_S \\ b_P = -\frac{1}{\Delta t}((\rho u^P)_e S_e - (\rho u^P)_w S_w + (\rho u^P)_n S_n - (\rho u^P)_s S_s) \end{gathered} \tag{5.25}$$

This linear system may be solved with any of the already known methods such as Gauss-Seidel or line-by-line solvers. It can be observed that in a fixed and constant mesh, the only term that is updated at each iteration is b_P.

After obtaining the value of the pressure p^{n+1}, the velocity field $\vec{v}^{n+1}$can be obtained by Equation (5.15) making use of the velocity predictor.

$$u_P^{n+1} = u_P^P - \frac{\Delta t}{\rho}\frac{p_B^{n+1} - p_A^{n+1}}{d_{BA}} \tag{5.26}$$

$$v_P^{n+1} = v_P^P - \frac{\Delta t}{\rho}\frac{p_B^{n+1} - p_A^{n+1}}{d_{BA}} \tag{5.27}$$

As already mentioned, the velocity is calculated at the nodes of the staggered mesh. The pressure, on the other hand, has been computed at the nodes of the main mesh. Thus, the suffixes p_A^{n+1} and p_B^{n+1} denote that the nodes of the main grid are located on the walls of the control volumes if the staggered mesh. This can be observed in the following illustration obtained and modified from [14]. In this way, the checkerboard problem is avoided.

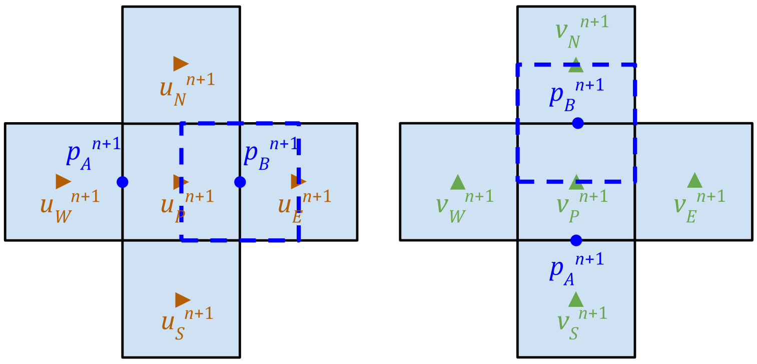


Figure 35. Velocity and pressure field distribution in staggered and main mesh

Finally, as the type of problem posed is transient in time, it requires choosing an increment of time Δt for the time iterations until reaching steady state. For the solution to converge, since an explicit scheme is being used, Trias and Lehmkuhl state that it is necessary to use the following formulation in [15].

$$\begin{gathered}\Delta t_c = \min\left(0.35\frac{\Delta x}{|v|}\right) \\ \Delta t_d = \min\,(0.20\frac{\rho \Delta x^2}{\mu}) \\ \Delta t = \min\,(\Delta t_c, \Delta t_d)\end{gathered} \tag{5.28}$$

5.2 Lid-driven cavity

The most widely used problem for the validation of a solution code for Navier Stokes equations is the one presented in this section. The lid-driven cavity problem consists of a square cavity with three rigid walls and a lid moving in a tangential direction to the fluid at a unit velocity. The object of study is the velocity and pressure fields create in the internal viscous and incompressible fluid flow.

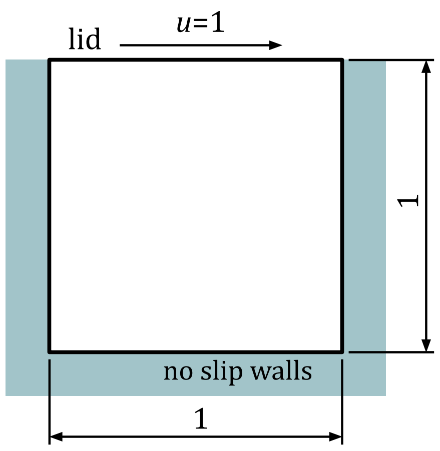


Figure 36. Geometry of the lid-driven cavity

Under initial conditions, both the velocity and pressure fields are considered to be zero. Starting from this point, the evolution of the fluid flow is studied as time elapses. The study is developed using different values of the Reynolds number.

5.2.1 Boundary conditions

The wall velocity values are defined in this table

Top wall	$u = 1m/s$	$v = 0$
Left, right and bottom wall	$u = 0$	$v = 0$

Table 6. Wall velocities for lid-driven cavity

The prescribed velocity must be taken into account both when calculating the actual speed field $\vec{v}^n$ and when using the velocity predictor $\vec{v}^P$. Thus, if the velocity in the next time step $\vec{v}^{n+1}$ is known like in this case, the predictor at a node P becomes:

$$\vec{v}_P^P = \vec{v}_P^{n+1} \tag{5.29}$$

As the cavity is considered to be rigid-walled, the pressure derivative is zero in the direction normal to each wall.

$$\frac{\partial p}{\partial n} = 0 \tag{5.30}$$

Therefore, the most external nodes of the main grid – which are no on the wall of the control volume – will have zero coefficient a_i when solving Poisson's equation. In this case, the coefficient a_i refers to the direction in which the wall of the control volume is located.

5.2.2 Results

The code used to solve the lid-driven cavity problem is an implementation of the Fractional Step Method. Results have been obtained for different Reynolds numbers and, in this section, are compared for $Re = 100$, $Re = 1000$ and $Re = 10000$.

The square domain of the problem has been discretised with a staggered-mesh with $N = N_x = N_y = 300$ nodes in the main grid. Thus, the stagg-x and stagg-y meshes have a total of $N_s = N + 2 = 302$ nodes in each direction. In order to obtain an acceptable accuracy in the results, a convergence factor $\delta_P = 10^{-5}$ has been used for the resolution of the Poisson equation and $\delta_t = 10^{-6}$ for the time convergence of the velocity.

The goal of the problem is to find the steady velocity field that is generated from the initial conditions. To reach such a steady state, a different time is required for each Reynolds number as will be shown in a later section. Below, the steady state velocity modulus field at each respective $t = t_{steady}$ is shown.

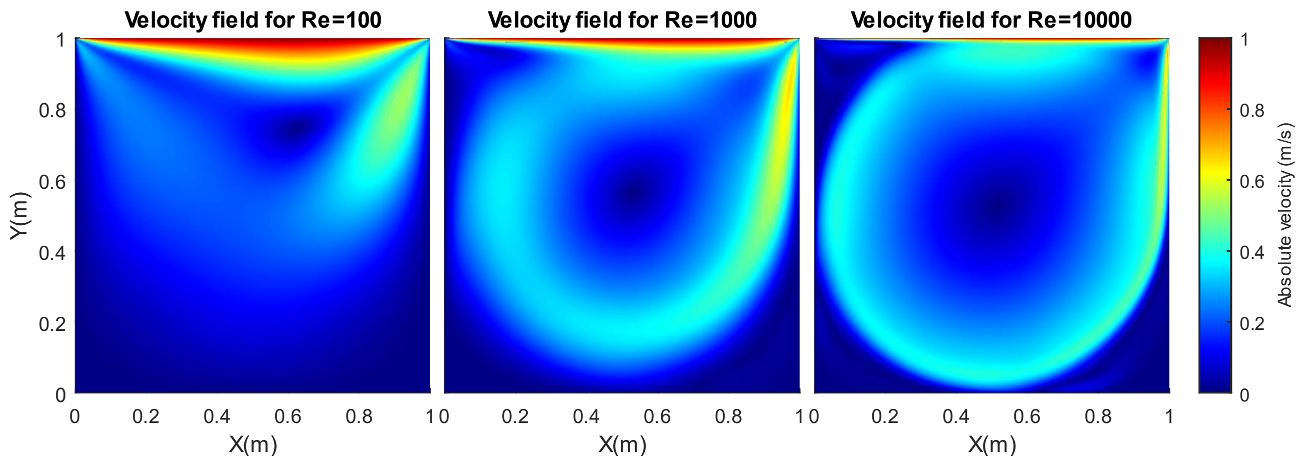


Figure 37. Velocity modulus field comparison for different Re values

A clear difference can be observed between the results according to the Reynolds number. For $Re = 100$, the velocity gradients are not so steep and more points in the domain are close to the maximum velocity. For higher Reynolds the gradient with the upper wall moving is much higher. In addition, a circular flow around the central coordinate can be appreciated.

An important point to note is that the Fractional Step Method performs better for low Re as it is difficult to deal with turbulent flow. Thus, while for $Re = 100$ the result is accurate, for $Re = 10000$ the results may not be that reliable. For even higher values, the result would hardly converge and its accuracy would be even lower.

The appearance of turbulent flow is preceded by the presence of vortices in the fluid flow. In the three cases presented, a clockwise vorticity clearly appears due to the condition of movement of the upper wall. However, as the Reynolds number increases, a higher number of other vortices appear. The streamline representation of the three cases is shown below, where it can be seen qualitatively.

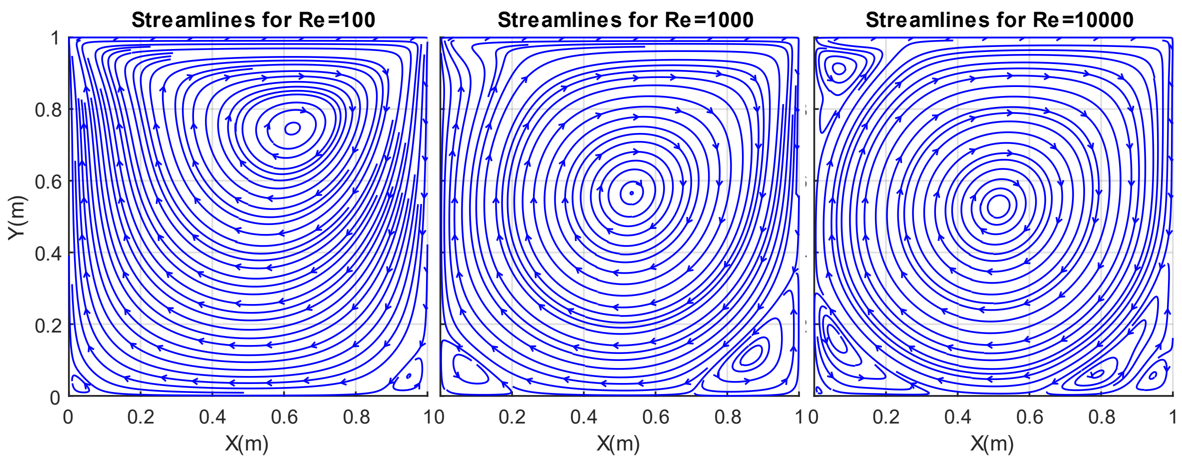


Figure 38. Streamlines comparison for different Re values

In all cases, it can be seen that in the lower corners there are vortices with a rotation in the opposite direction to the main vortex. Their vorticity increases with the Reynolds number. In addition, the rotation axis of the main vortex is also displaced and tends to the centre of the square domain.

5.2.3 Verification

One of the reasons for studying the lid-driven cavity problem is because it is considered a canonical case with available results for code verification. In this section, a comparison is made between the expected results and those obtained.

The reference results are those of the benchmark in [16]. The authors of the study present a table with velocity values. The values of the u component for the central vertical line ($x = 0.5$) and of the v component for the horizontal line ($y = 0.5$) are given. The following figure shows the benchmark data using squares and the curve obtained by the programme. The horizontal component of the velocity is shown in blue and the vertical component in orange. The mesh size and convergence factors are the same as in the previous section.

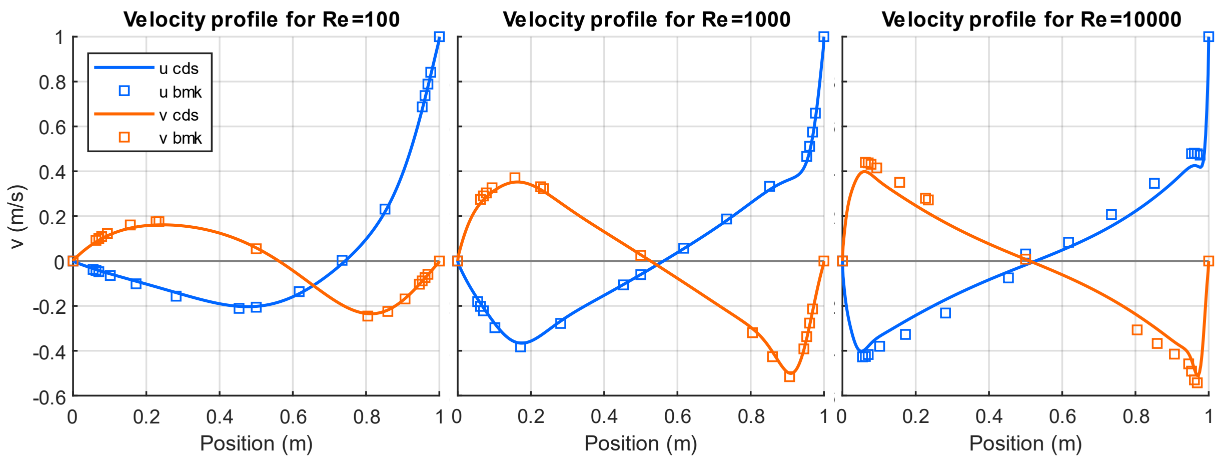


Figure 39. Horizontal and vertical central line velocity profile

At first sight, the results are satisfactory and sufficiently close to what was expected. For $Re = 100$ and $Re = 1000$ the curve follows the benchmark points quite accurately. The points furthest away from the solution correspond to the most extreme values, at the relative maximum and minimum of the curve.

For $Re = 10000$, however, the results are not as good. In this case, the error with respect to the benchmark points is higher. In any case, an acceptable behaviour can be appreciated that does follow the correct tendency. The lack of precision of these results is analysed in the following section.

5.2.4 Mesh size analysis

The mesh size is a key factor when it comes to obtaining an accurate result. In addition, mesh refinement becomes more important as the Reynolds number increases, as the gradients in the properties become larger. Too large control volumes can lead to a too coarse approximation of the result.

On the other hand, obtaining the result for a high Reynolds number involves many iterations until the steady time is reached. This fact implies that a large number of nodes means a long compilation time. This is the reason why a maximum mesh size of 300×300 has been used in this project.

The following shows, for $Re = 1000$, how the number of nodes used affects the accuracy of the result. The same graph is used as in the previous section, but the meshes with a smaller number of nodes are presented in lighter colours. The convergence factors have remained constant for all computations.

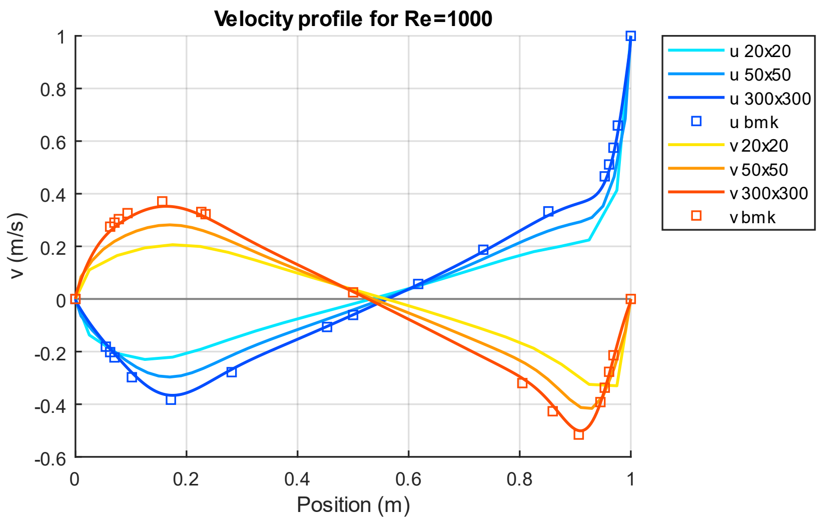


Figure 40. Comparison of the velocity profiles for different number of nodes

Clearly, the result is closer to the benchmark as the mesh size increases. Also, note that for meshes with few nodes, although the shape of the curve is similar, the values of the result tend to have a lower absolute value. This is the reason why in the solution with the maximum number of nodes the values with a higher relative error are those corresponding to the relative maximum and minimum. In this way, the most extreme values could be obtained by further increasing the mesh size.

This result also justifies the solution presented for $Re = 10000$ in Figure 39. As is the case here for mesh sizes that are too small, the absolute value of the curve points is smaller than those of the benchmark in the whole domain. Therefore, it can be deduced that a mesh with a larger number of modes is required to obtain a higher accuracy.

An alternative to increasing the number of mesh nodes with the corresponding increase in computational time is to use another type of mesh. The results presented use a uniform grid where all control volumes have the same size. A good solution would be to find an alternative mesh that concentrates the control volumes where the flow behaviour is most critical.

Some authors propose a non-uniform mesh with hyperbolic node concentration [14]. In this way, by means of a stretching factor, a higher concentration of nodes close to the walls is imposed. As can be seen in the representations of the velocity modulus and streamlines, the largest gradients and jumps in the property values take place at the walls of the domain and, more specifically, at the corners. Therefore, this would be a suitable solution that would allow more accurate results to be obtained with the same mesh size.

5.2.5 Time to reach steady state

The lid-driven cavity problem starts from an initial state where for t = 0 the velocity field is 0 throughout the domain. As soon as the top wall starts to move, so does the flow directly in contact, thus creating a non-zero velocity field over the whole domain. The boundary conditions remain constant with respect to time, so the initially accelerated fluid tends to a steady state.

In the case of the programme used, the stationary time can be defined as the instant of time when the difference of the value of the velocity at any point in the domain with respect to the value at the same point at an earlier instant is less than the convergence factor δ_t. In other words, max $(|\vec{v}^{n+1}(x,y) - \vec{v}^{n}(x,y)|) < \delta_t$.

A larger change with respect to the initial conditions implies a longer time to reach steady state. Therefore, for higher Reynolds numbers, t_{steady} is greater. Below is shown the steady state time for different Reynolds numbers.

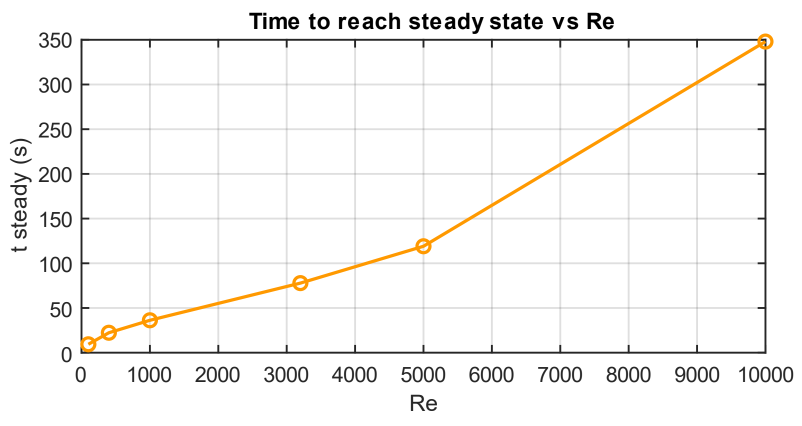


Figure 41. Time to reach steady time as a function of Re

These results have been obtained for a uniform mesh with $N = 300$ nodes in each direction. It can be observed that the time increase is more abrupt as the Reynolds number increases. Therefore, this method, apart from giving inaccurate results for high Re values, also requires many time increments to reach a solution.

Moreover, the Reynolds number is not the only parameter that affects t_{steady}. In this case, we compare how t_{steady} evolves as a function of the number of mesh nodes in each direction. $Re = 100$ is taken in all cases. The time taken by the computer to calculate the results is also presented.

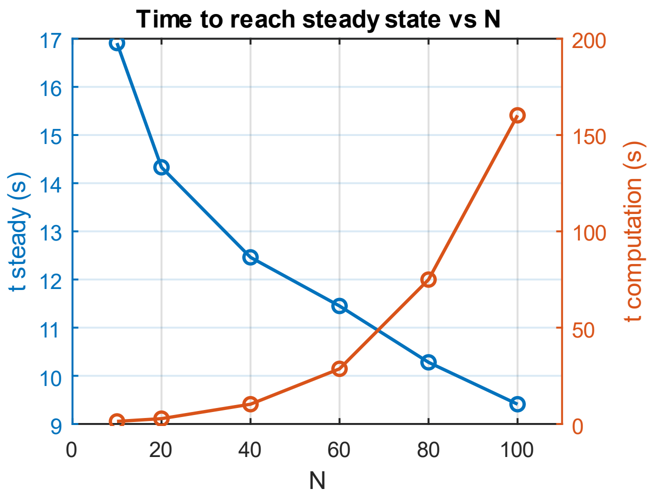


Figure 42. Computation time and time to reach steady state

The conclusion is clear, on the one hand the number of nodes N is inversely proportional to t_{steady}. On the other hand, as the number of nodes N increases, computing time $t_{computation}$ increases exponentially.

5.3 Conclusions

The aim of this section is to become aware of the great complexity involved in solving the Navier-Stokes equations numerically. Although it is one of the fundamental equations for current fluid mechanics, there is still no known analytical solution. Thus, the numerical solution of these equations is currently the object of study in the field of Computational Fluid Dynamics.

The approach used to tackle a first Navier-Stokes problem is the Fractional Step Method. It is a technique for the solution of the unsteady and incompressible equations. Its great performance and the code simplicity are some of the main reasons to use this method.

The discretisation of the problem by means of the Finite Volume Method has required a new concept: the staggered meshes. This kind of meshes where each property of the fluid is studied in different positions is required in order to avoid the pressure checkerboard problem. An issue that may lead to converged velocity fields for unphysical pressure distributions. The implementation of a problem using a staggered mesh had been a new challenge with regard to previous problems.

The described algorithm has been applied to the lid-driven cavity problem. This is a very popular and typical case when starting to work with the Navier-Stokes equations because of its simplicity and because the results obtained by other authors are known. In this way, the code used can be validated.

The verification carried out on the developed code has been satisfactory. Better results have been observed for low Reynolds numbers. For very high Reynolds numbers the result is not as close to the benchmark values although a correct tendency to approach it is observed. This error is due to an insufficient number of nodes and accuracy factor to obtain a satisfactory solution due to the long computational times that higher values entail.

In fact, the times involved in solving the problem have been studied in the last section. From a physical point of view, the time required to reach steady state starting from the initial conditions has been studied. It has been shown that this time increases in large proportions as the Reynolds number increases. However, for the same Reynolds number, it decreases as the accuracy of the mesh increases.

From a computational perspective, an increase in accuracy, which translates into larger meshes or smaller convergence factors, implies a large increase in computational time. The large number of iterations, the big size of the objects used and the quality of the code are determining factors in trying to reduce this time.

Thus, it is concluded that solving the Navier-Stokes equations is very expensive to solve both in terms of time and computational energy. This is one of the great challenges facing scientists and engineers working in the field of CFD where solving times for much more complex problems can take hours, days or even weeks.

6

Towards larger real problems

The exercises studied in this project are of great academic value in introducing the key concepts for the numerical solution of physical problems. However, they are ideal cases that can be solved with small meshes and low amounts of data requiring relatively little computational power. This last section of the report aims to introduce the new kind of computational approach required for the larger, real problems.

The limitations of current hardware are presented along with various techniques that scientists and engineers use to optimise computational resources. One of the main objectives of large simulations is to reduce the time required to compute them. This last section presents some of the most used algorithms to achieve so.

6.1 Larger problems

The problems presented in this report aim to study a physical problem that cannot be solved analytically due to its magnitude. To tackle the problem, numerical methods that discretise the domain of study into a large number of small control volumes are used, which can be easily studied by a computer. Although the nature of the studied cases is different, in all of them a more precise solution is obtained when more elements are used for the discretisation.

However, making use of more elements or greater meshes comes with a high computational cost. As presented in Figure 42, the computing time required to achieve an accurate solution gets sharply increased as the number of elements grows.

The problems treated in this report need relatively small meshes due to their ideal nature. Nevertheless, more complex real and three-dimensional cases would require huge amounts of computing time if they were solved the same way it has been done in this report. The time to reach an acceptable solution could get increased to hours, days or even months. Furthermore, the required memory to allocate all the data could be much bigger than the available in a personal computer.

For example, in case of studying the lid-driven cavity problem again, but now using three dimensions, the mesh requirements would become much higher. It is not just enough to add one dimension to the mesh to obtain a mesh of N^3 elements instead of N^2, but the appearance of the turbulence phenomenon requires a much greater refinement. In other words, if in the ideal 2D case studied the Reynolds number was already of great importance when choosing the mesh size, it is now a critical factor.

Moreover, if one works with real geometry, far from the ideal cases where regular meshes are enough, the computational requirement is even higher. Figure 43 obtained from [17] is a great example of how a massive mesh look like in real CFD problems. The authors required a high-resolution grid of approximately 10.8 million cells to numerically solve the Navier-Stokes equation in all the domain.

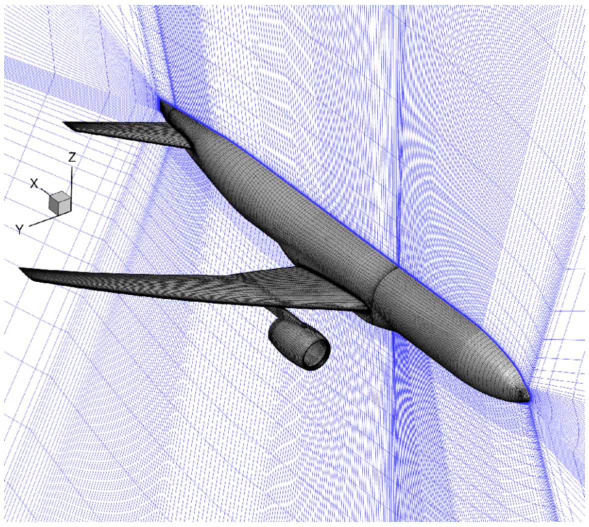


Figure 43. CFD surface mesh of an aircraft and its plane of symmetry

Of course, such problems can only be solved by using computers with high computing power. Nowadays, however, it is not enough to dispose of the most powerful and modern hardware. It is equally or even more important to make use of ingenious techniques to optimise computing time through the programmed solving code used.

In fact, the study of these new optimisation methods has given rise to what could be considered new disciplines of research such as *high-performance computing* or *computational science and engineering*.

This last section of the report aims to give a first approach to these disciplines and techniques so necessary in modern day problems.

First, the most critical elements concerning computational time in the field of physical problem solving are identified. It is also presented how the computer processor deals with these crucial elements. Then, different methods are presented to improve CPU performance, from the use of sparse matrices to the reordering of these to obtain better efficiency in certain operations.

Finally, the strategies are tested by conducting operations and measuring the calculation time on a problem coefficients matrix of this report and on much larger ones obtained from a repository. The results are commented and conclusions are reached.

6.2 Scientific computing performance

The computer CPU is the unit responsible for performing basic arithmetic, controlling the equipment and handling input and output operations specified by program instructions. This is the main element to take into account when studying the time and performance behaviour in dealing with the numerical resolution of large problems.

It is important to note that other technologies are currently being used to obtain high computing power, such as GPUs or processors using different architectures. However, these are outside the scope of this report.

A CPU has a large number of characteristics that define the performance and processing rate it is capable of providing. Determining factors are the clock speed or the type of processor used, the number of cores available or the size of the cache memory.

In terms of scientific computing, high performance is provided by the speed of the processor and memory access. Parallel programming is also one of the greatest allies but is out of the scope of this project.

6.2.1 Processor speed

The effective processor speed can be adequately measured by estimating the floating-point operations per second ($FLOPS$). This measure is one of the most representative in this kind of computation because practically all the operations required are of floating type. Other measures such as processor instructions per second would not be as accurate.

Modern supercomputers in 2021 measure they speed with a magnitude in the order of petaflops, i.e. around 10^{15} floating-point operations per seconds [18]. In ordinary personal computers the velocity is around $10^{12} FLOPS$. In fact, the improvement of this technology has been exponential over the last few years as can be seen in the following graph obtained from [19].

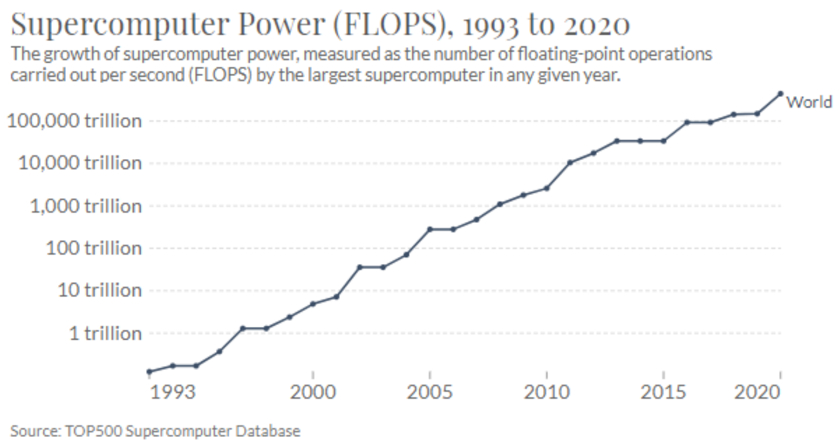


Figure 44. FLOPS performed by the most powerful supercomputer each year from 1993 to 2020

6.2.2 CPU memory access

Even though processor speed is a determinant factor when it comes to defining the performance of a calculation, memory accesses usually become the main issue to be considered.

The following scheme obtained from [20] and adapted represents in simplified form the memory hierarchy that the CPU can access to obtain the required data for the computation.

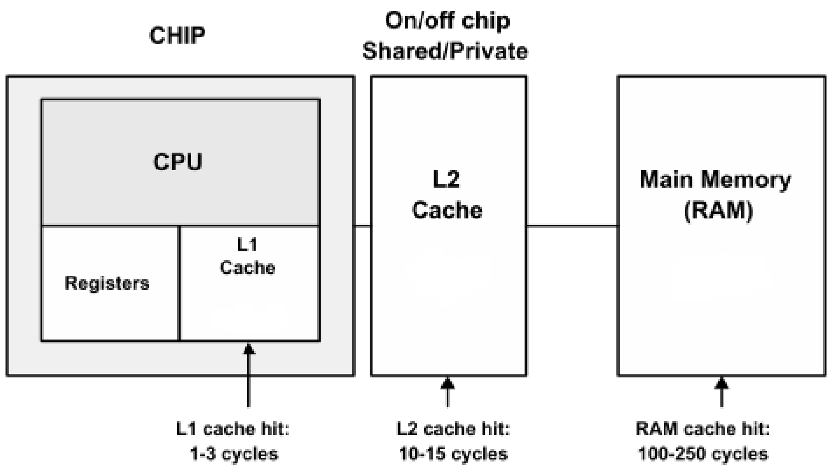


Figure 45. Simplified scheme of CPU access to memory

The processor can either get data from *fast* or *slow* memory. In this case, fast memory refers to *L1* and *L2* cache memory. In contrast, the main memory is considered the slow one and as can be seen in the scheme requires from 10 to 100 times more to achieve a hit.

The capacity of these three memories is inversely proportional to their speed. In 2021, for ordinary computer devices, the L1 cache has a memory of hundreds of kB, the L2 a few MB and the RAM memories have a magnitude of GBs. For instance, and to use real data, the computer used to solve the problems in this report uses an *AMD Ryzen 7 5800H* processor with 3,20GHz speed and the following memory specifications.

Memory	Size
L1 cache	512 kB
L2 cache	4 MB
RAM	16 GB

Table 7. Memory specs for AMD Ryzen 7 5800H processor

Ideally, all memory in the computer should be as fast as the L1 cache, so that a large amount of data can be fit into and accessed from this memory. However, this is not the case in reality. Fast memories are expensive and consume large amounts of power.

The aim of the memory hierarchy presented is to combine the amount of data that slower memories can store with the speed of the cache memory. To achieve this, the CPU allocates the data being used by the running program from the main memory to the cache. Therefore, most of the accesses of the CPU to memory take place in the cache.

When the processor requires and gets one element from the cache memory, a *cache hit* occurs. As seen in Figure 45, from 1 to 15 processor cycles are required for a cache hit, depending on the cache level.

Instead, if the element required to perform a calculation is not available in the cache when the processor demands it, a *cache miss* results. Main memory must then be accessed with the corresponding delay in the computation. Figure 45 quantifies this delay. A RAM cache hit requires between 100 and 250 processor cycles. In contrast to the L1 cache, it is a hundred times slower.

Indeed, this is a common problem in large real problems, as the matrices and vectors used are too massive to be allocated in the cache. Thus, the RAM memory accesses are higher than desired and computation time is greatly increased.

6.2.3 Low arithmetic intensity and slow memory access

Arithmetic intensity describes the relation between the floating-point operations realised in a piece of code and the amount of memory accesses required. Its units are $FLOP/Byte$ and it is a reliable measure for quantifying where the focus is on importance, processor speed or memory access [21].

The problems dealt with in this report have a low arithmetic intensity. In the resolution loops, the vast majority of operations require new data from memory. Moreover, the operations performed are very basic and have a low computational cost as they are simple linear combinations.

Indeed, many of the problems addressed in computational science and engineering are of this nature. They are known as *memory-bound* problems. The term makes reference to those problems or functions where the resolution time is mostly determined by the amount of memory accesses to handle the required data [22]. The opposite to this situation is given by a *CPU-bound* problem, where the bottleneck is given by the number of basic operations.

Another reason for the crucial importance of memory access when looking at performance is the hardware technology limitations. While in last years CPUs have improved exponentially in computational speed as can be seen in Figure 44, memory access time has hardly decreased in recent years. Figure 46 obtained from [23] shows a clear comparison of the evolution of both technologies from the 1980s to 2010.

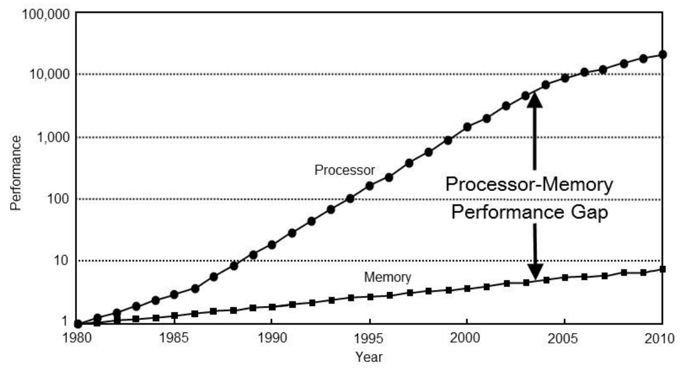


Figure 46. Processor vs memory performance gap

In the figure, performance makes reference to access speed for memory and FLOPS for the processor. While microprocessor industry has focused on improving speed, the memory field has concentrated on increasing storage capacity. This division of objectives has led to an exponentially growing gap between these two technologies when it comes to time of execution.

These two reasons are why RAM access by the CPU is a critical step in increasing computing time. The following section aims to present which typical scientific computing operations have the greatest impact on poor performance due to memory accesses.

6.2.4 Basic arithmetic operations

Modern computers have great computing power when it comes to linear algebraic operations. As has been presented throughout this report, the vast majority of scientific problems and physical systems that are studied are represented by a continuous environment. However, by making use of discretisation techniques, a continuous system of equations is converted into a discrete system.

This way, systems with complex mathematical operations can be reduced to schemes with the most basic arithmetic operations that can easily and quickly be solved by a computer. The CPU can promptly compute these as computational kernels.

For example, the Navier Stokes system of equations is one of the most widely studied and used in the field of computational fluid dynamics. These equations clearly describe a continuous environment. It can be expressed as follows as already been presented.

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} = \frac{1}{Re}\nabla^2 \vec{v} - \nabla p \tag{6.1}$$

Many authors have proposed a discretisation for this function. One of the most commonly used and convenient when it comes to being solved numerically is the vector matrix notation. That is, to convert all operators. into operations between vectors and matrices. For example, the discretisation of the Navier Stokes equations on an arbitrary collocated mesh using finite-volume method proposed by Trias in [24] is presented below. Forgetting the physical sense, elements in upper case represent matrices and in lower case represent vectors.

$$\boldsymbol{\Omega}\frac{\mathrm{d}\vec{v}_c}{\mathrm{d}t} + \boldsymbol{C}(\vec{v}_s)\vec{v}_c + \boldsymbol{D}\vec{v}_c + \boldsymbol{\Omega}\boldsymbol{G}_c\vec{p}_c = 0_c \tag{6.2}$$

Now this system is expressed by making use of only three simple operations: linear combinations, dot product of vectors and matrix-vector multiplication. Note how relatively complex operators such as the divergence of the velocity or the pressure term ∇p have become simple matrix-vector multiplications, $\boldsymbol{D}\vec{u}_c$ and $\boldsymbol{G}_c\vec{p}_c$ respectively.

In fact, all the cases treated in this document have been approached by solving a linear system of equations expressed using matrix-vector notation $Ax = b$. This kind of systems are commonly solved with solvers that make use of these same three operations. As seen, final coefficient matrices have a great number of zeros which make them ideal to be treated as sparse matrices. This concept is treated in detail in Section 6.3. For now, it can simply be stated that the multiplication between a matrix with a large number of zeros within its values and a vector is a SpMV (sparse matrix-vector multiplication) computational kernel [25].

For example, the Conjugate Gradient is a commonly used iterative solver which makes use of the following computational kernels:

- Dot product (DOT)
- Vector linear combination (AXPY)
- Sparse matrix vector multiplication (SpMV)

In the idealized pseudo-code for Conjugate Gradient presented in [26], three AXPY, two DOT and one SpMV are performed in each iteration. While AXPY and DOT have a computational complexity of $O(n)$, the SpMV complexity is bounded by $O(n \times nnz)$. Therefore, most of the time in each iteration is spent on the resolution of the latter kernel although it is solved only once.

Furthermore, the SpMV is clearly the most memory-bound kernel among the three. While for non-parallel DOT and AXPY the required data is accessed contiguously in memory, the SpMV vector values are not accessed in the same order as they are in memory. As previously discussed, this situation leads to a poor CPU and cache performance requiring even more computational time. The reason for this phenomenon in the SpMV calculation is fully explained in Section 6.4.1.

Concluding, the SpMV is one of the most commonly used and critical operations at same time in many scientific algorithms. The following sections discuss the reasons and how to focus on improving its performance. First, however, the sparse matrices are presented in detail so that the following steps can be better comprehended.

6.3 Sparse Matrices

A sparse matrix is a matrix containing a low proportion of non-zero elements. In a computer, these matrices can be treated differently by more efficient techniques that take advantage of the number of zeros present. [27]

6.3.1 Justification

Many problems in the field of science and technology end up making use of matrices to be solved by computation. In fact, this project is a clear example of the intensive use of matrices for this very reason.

A shared characteristic in all the problems is that the matrices contain a large relative number of zeros and the significant values are a minority.

For example, in the lid driven cavity problem, the algorithm used must solve a Poisson system of equations to solve the pressure field. In this case, the non-zero values of the matrix are those describing the pressure at the nodes. Taking into account that a structured and regular grid has been used, a node only depends on a maximum of four neighbouring nodes. When implementing the matrix of the pressure system to be solved, this means that in each row there will be a maximum of five non-zero elements. In other words, a 300×300 nodes mesh is equivalent to obtaining a matrix where less than 1.67% of the elements are non-zero.

Although there is no exact criterion, a matrix is considered to be of sparse type when the non-zero elements do not exceed 10%. [27]

There are a large number of disciplines related to science and engineering that make use of sparse matrices. Some examples are the study of structures, electromagnetic fields, electrical circuits, graphs, etc. All kinds of sparse matrices can be found in [28], a repository created by the University of Florida.

A sparse matrix can be defined by various criteria such as the level of its sparsity, bandwidth, symmetry, etc. Depending on the characteristics of the sparse matrices, different algorithms can be used to treat and manipulate them and not all of them will be suitable [29]. However, there is one obvious advantage that all sparse matrices share over dense matrices: the space required in computer memory to store their content is much smaller.

6.3.2 Data storage

In a sparse matrix the only significant values for the problem to be solved are those other than zero. In this way, a large memory saving can be produced by storing only these values.

Below there is a matrix of size 9×9 which will be treated as an example of a sparse matrix to represent memory allocation. Non-zero values are represented by a letter while zero values are shown with an empty white position.

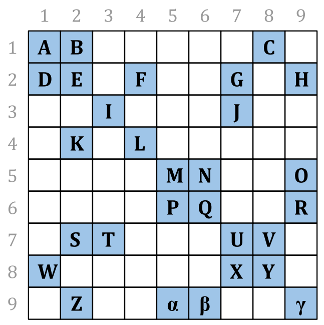


Figure 47. Example of sparse matrix

This square matrix contains a total of 81 values of which 29 are non-zero ($nnz = 29$). Note that the percentage of non-zero values is higher than desired for a sparse matrix but is useful for the examples in this report.

If the matrix was treated as a dense matrix, the storage in the computer's memory would occur as shown in the following scheme.

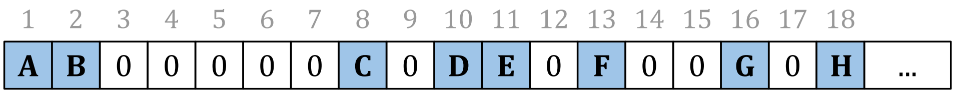


Figure 48. Dense matrix memory distribution

The computer's memory is buffered linearly. Therefore, although in this case it is a two-dimensional matrix, the values are stored one after the other row by row. To access a particular value in the array, one must know the memory address of the first element and the relative position of the searched element within the matrix [30].

For example, it is assumed that one wants to access the value J of the matrix at position (3, 7). In this case, you must access the memory address of $A(0,0)$ and add to it the relative address value of the searched position.

$$\mathrm{dir}(J) = \mathrm{dir}(A) + ts\,(Ni_J + j_J) \tag{6.3}$$

Here the relative direction is determined by the number of rows i_J multiplied by the number of elements in a row N plus the number of columns j_J. The value obtained must be multiplied by the space required in memory ts according to the type of variable used. In case of integer variables it is usually 4 bytes while for double type variables a minimum of 8 bytes are required [31].

This type of memory access requires all the elements of the matrix to be stored in memory in order to preserve the relative position. Thus, the amount of memory used is equal to $N \times N$. In matrices where a large number of their values are irrelevant (such as zeros in sparse matrices), this results in a large amount of wasted memory.

Two methods of storing sparse matrices in a space-saving manner are presented below. Both methods store only non-zero values. However, in this way the possibility to find the position of the values by means of the relative direction discussed in this section is lost. Therefore, it is required to store the indices of each value.

6.3.2.1 Coordinate Ordered

The Coordinate Ordered method stores non-zero values in a vector of size equal to the number of non-zeros (nnz). In turn, it uses two vectors of the same size to store the row and column index [32].

The following diagram presents the memory allocation for the sparse matrix in Figure 47. The first two vectors refer to the position in the row and column. The last one stores the value. The relative address within the memory is shown above the arrays in light grey.

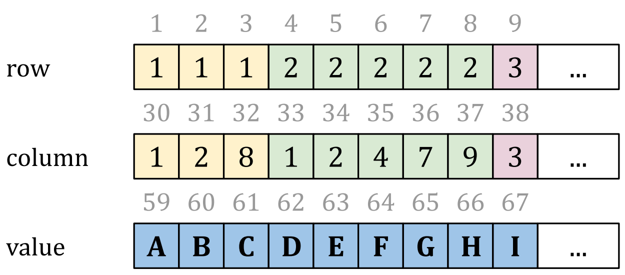


Figure 49. CO memory distribution

Storing an array using this scheme occupies a total of $3 \times nnz$ memory locations. This is why it is only used for sparse matrices, as the method is efficient as long as the space required is much smaller than with classical ordering. That is to say,

$$3 \times nnz \ll N \times N \tag{6.4}$$

6.3.2.2 Compressed Sparse Row

Compressed Sparse Row is a method that uses even less memory space than Coordinate Ordered. In this case, the value vector also only stores the non-zero values. However, the index vectors consist of a row offset vector and a column vector.

The column vector still represents the position of the column where the searched value is located and its size is nnz. However, the row offset array indicates the position of the column vector at which a row change occurs. Thus, the row offset vector has a size of $N + 1$ [33]. It can be illustrated by the saving scheme using the previous matrix.

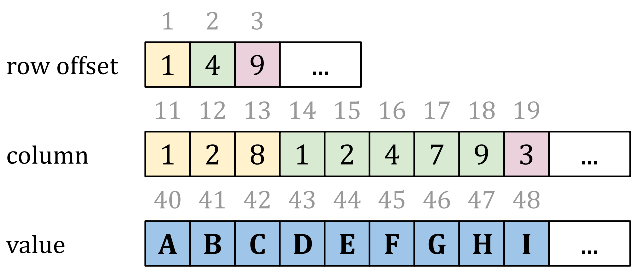


Figure 50. CSR memory distribution

It can be seen by the use of the colours which values are part of each row and how the row offset points to the start of a new row.

This method makes use of $2 \times nnz + N + 1$ memory locations. When the sparse matrix definition is met, it is almost always more efficient than the Coordinate Ordered method.

Therefore, it is the most widely used method in this field and is the one that will be used in this report.

6.3.3 Graph representation

The main purpose of using the sparse format to work with matrices with a large number of zeros is to save computer memory space. However, there are also a variety of algorithms that take advantage of the format to optimise the calculations to be performed with these matrices.

At this point, the graph as a memory representation format is a particularly useful tool. A sparse matrix can be represented as a graph on which algorithms can be applied more easily [20].

This project deals with symmetric sparse matrices in order to simplify the study. Furthermore, the emphasis is on the structure of the matrix rather than the value of the non-zero elements. Therefore, following the example presented above, the sparse matrix is converted to a graph in this way.

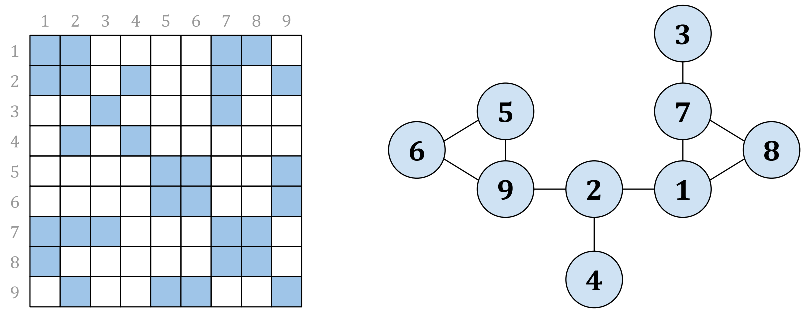


Figure 51. Example sparse matrix turned into a graph

Each node in the graph represents a row or a column - which are equal due to the symmetry of the matrix. The edges between nodes stands for the position of the non-zero values within the sparse matrix. For example, the first row has non-zero values in columns 1, 2, 7 and 8. Therefore, node 1 is connected to all these nodes. Note that as all rows are connected to each equivalent column, all nodes would have an edge from and to them that is omitted.

If non-zero values were required in the graph, the edge weight would contain the value of that position in the matrix. Similarly, if the matrix was not symmetric, a directed graph should be used and the edges weight would be different depending on their direction.

6.4 Optimisation by bandwidth minimisation

Most computer operations and algorithms using sparse matrices are most efficient when they have their non-zero elements close to the diagonal. This property is defined by the matrix semibandwidth β which for a symmetric matrix A of dimension $N \times N$ is given as follows [34].

$$\beta(A) = \max \{|i - j| : a_{ij} \neq 0\} \tag{6.5}$$

The following section describes why it is of interest that the matrices used in computational operations have a minimum β. Then, a method to reduce the matrix bandwidth by permutations of the matrix is presented.

6.4.1 Cache-based performance in SpMV

The sparse matrix-vector multiplication is the most critical kernel operation when CPU requires to access memory presented in this report. Apart from dealing with the largest data structures (a matrix and a vector), it can have very poor cache-based performance. However, by reducing the bandwidth of the sparse matrix the computing time of this operation can be greatly reduced. This section explains the reasons for this poor performance and how reducing the bandwidth of the matrix can solve it quite well.

The result of a matrix-vector multiplication is a vector $\vec{y} = A\vec{v}$. The i-th component of the destination vector $\vec{y}$ is the result of the dot product between each i row of the matrix A and the source vector $\vec{v}$. The following scheme illustrates this procedure for a multiplication between the example sparse matrix in the seventh row and a dense vector [32].

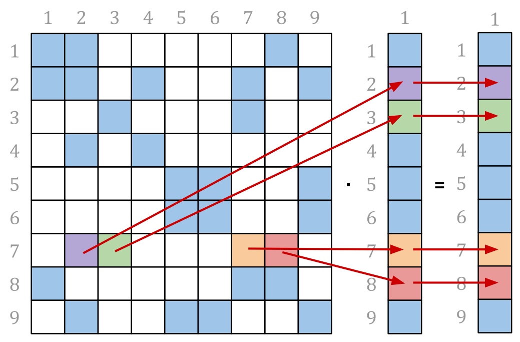


Figure 52. Scheme of a sparse matrix-vector multiplication

The logical implementation of a SpMV operation is to iterate the sparse matrix row-by-row. In each iteration, all the non-zero values of the row and the correspondent vector values are accessed. Then, the dot product can be easily performed and assigned to the component of the destination vector.

The illustration below obtained from [25] describes the process that takes places when SpMV kernel is calculated. In this case, as usual, the Compressed Sparse Row (CSR) is used as a method of storing the sparse matrix.

The *row pointer* indicates the row on which the operation is performed. Each *nz value* is multiplied by a value from the *source vector* allocated at the position denoted by the *nz* *column index*. All these products are summed and loaded into the *destination vector* at the corresponding position of the current *row pointer*.

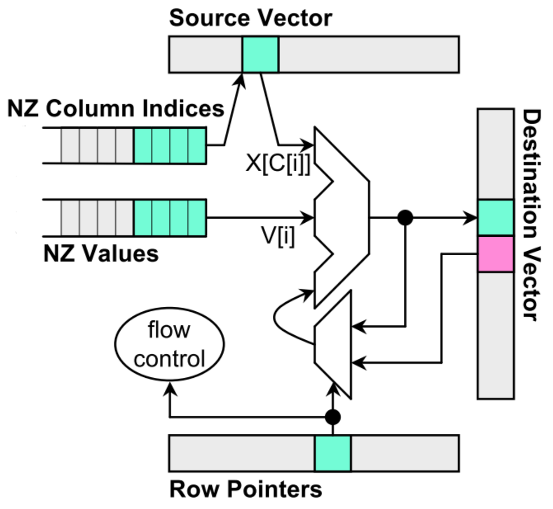


Figure 53. SpMV kernel memory access

As mentioned, non-zero values of the sparse matrix are consecutively allocated in arrays. Following the procedure just exposed, these elements are accessed by the CPU in the same order they are found in memory when SpMV is being calculated. Therefore, the required data of the matrix is almost always available in cache memory and the ratio of cache misses to cache hits is low.

The elements of the vector are also allocated contiguously in memory. However, in this case they are not accessed in the same order as in memory. In each iteration, the accessed values of the vector are the ones in the position of the columns where non-zero values of the matrix are located, indicated by the *nz column indices*.

For example, in the last row of the matrix in Figure 52 the *nz* values of the sparse matrix are found in the first and last column. Even though the values of the matrix are placed contiguously in memory, the vector will be accessed at its first and last position. If the vector is relatively large, a cache miss is likely to occur.

Note that, due to sparsity of the matrix, very few elements of the vector are required at each iteration. Thus, the probability of experiencing a cache miss increases depending on the distance between the values being accessed. This distance is shortened when bandwidth of the matrix is minimised since the values of columns *j* gets closer to the value of the row *i*. As a consequence, the accessed vector values also decrease their distance between them.

Concluding, to reduce the bandwidth of a sparse matrix from a SpMV can considerably enhance the performance of the kernel operation. This improvement only has an effect on the memory access time of the source vector as the rest of the values are already accessed in the order in which they are found. Still, in cases where the vector (and thus the matrix) is very large, the efficiency gain is very noticeable as demonstrated in the last part of this section.

Once the benefits of sparse matrix bandwidth reduction have been presented, an algorithm for achieving this goal is presented.

6.4.2 Reverse Cuthill-McKee algorithm

The Cuthill-McKee algorithm was proposed by the authors of the same name in [35] in 1969. This method reduces the bandwidth of a symmetric sparse matrix through permutations.

The great usefulness of this algorithm has led many authors to propose modifications to improve its efficiency. Among the most popular is the contribution of Alan George and Joseph Liu in [36] where they propose the Reverse Cuthill-McKee (RCM) algorithm. In this case the order of the result for the original one is simply inverted to improve the performance. This is the algorithm discussed in this report.

Although it is implemented to reorder sparse matrices, it is executed in its transformation to an equivalent graph.

The aim of the scheme is to renumber the nodes of the graph so that neighbouring nodes have a close numbering. That is, to make the columns *j* contained in a row *i* have *j* values close to *i*. Obviously, this would lead to a reduction of the matrix bandwidth.

The following scheme represents the algorithm used to achieve such an objective [29].

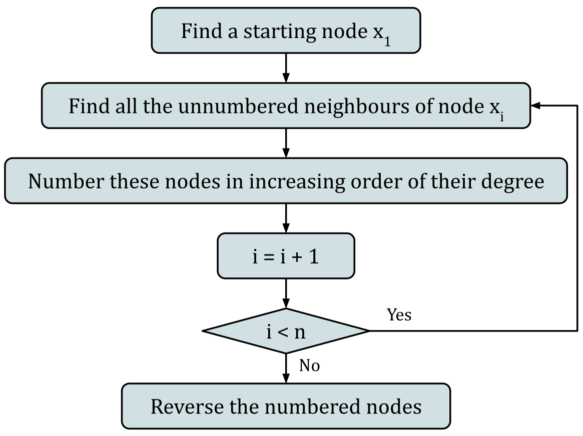

Find a starting node x_1
Find all the unnumbered neighbours of node x_i
Number these nodes in increasing order of their degree
$i = i + 1$
$i < n$
Yes
No
Reverse the numbered nodes

Figure 54. Scheme of the Reverse Cuthill-McKee algorithm

Note that the last step is the only difference between RCM and the original Cuthill-McKee algorithm, where the numbering is reordered. It is also important to highlight that in case the studied graph was disconnected, the algorithm would be applied to each of the connected parts of the graph.

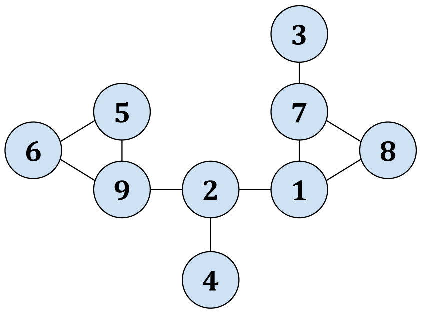
The presented scheme makes use of an arbitrary initial node. In the next section a method for finding the best initial node for optimal reordering is presented.

6.4.2.1 RCM applied to the example

To observe the operation and results of the RCM method, it is applied to the example used in the previous sections.

The original sparse matrix, which has already been converted to a network, is used as a starting point. In this case, node 5 is chosen as the initial node using the method explained in section 6.4.3. However, for the example presented, it is not necessary to know how the initial node has been chosen. In fact, RCM could be used with any initial node.

The method used to apply RCM is presented below. Starting with the first node, a queue of nodes is created in ascending order of degree. These nodes will be the next to be numbered. As they are numbered, they create their queue of neighbours. This procedure can be seen in the second and third column of the presented table. The first column simply sets out the relationship to the current inverted ordering.



Current order	New order	Queue
9	5	6, 9
8	6	
7	9	2
6	2	4, 1
5	4	
4	1	8, 7
3	8	
2	7	3
1	3	

Figure 55. Reordering of the graph nodes using RCM

Once the new reordering has been obtained, the new labelling is simply applied to a graph of the same form. It is important to note that the only thing that changes is the ordering, as the connections remain the same. That is, no connections can appear or disappear between rows and columns of the represented matrix. In the case of the example, the new graph is as follows.

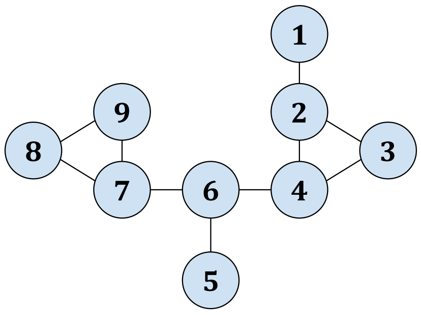


Figure 56. Resulting reordered graph

With the new ordering, it can be appreciated how neighbour nodes are assigned a close numbering.

In order to verify the results obtained, graph bandwidth may be calculated. In a graph G with N vertices v_i with distinct labels i where E is the edge set of the graph, the bandwidth β_G can be defined as follows [37].

$$\beta_G(G) = \max\{|i-j| : (v_i, v_j) \in E\} \tag{6.6}$$

Note that the graph bandwidth is exactly equivalent to the matrix semibandwidth presented in (6.5). Therefore, for a matrix A and its equivalent graph G_A bandwidths must be equal $\beta(A) = \beta_G(G_A)$.

For the example in this report, on one hand it can be seen at first sight that for the original graph G_0 the bandwidth is $\beta_G(G_0) = 7$ because the furthest neighbouring nodes are 1 and 8 or 2 and 9. On the other hand, the reordered graph G_r presents $\beta_G(G_r) = 2$ as there is no pair of nodes with a labelled value difference greater than or equal to 3. Thus, graph bandwidth clearly has been reduced.

Finally, it is the turn to reconvert the obtained graph to a sparse matrix again. As already mentioned, the use of graphs is very useful for reordering algorithms, but the sparse format is preferable when it comes to operating with the matrix. So, below, the original starting matrix on the left is presented together with the reordered one on the right. Diagonal bandwidth has been represented using a red line.

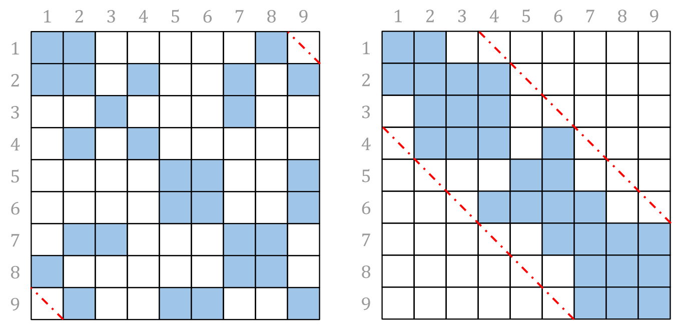


Figure 57. Matrix bandwidth comparison before and after reordering

At this point, the reduction in matrix bandwidth can be seen even more clearly. While in the first matrix the values are randomly sorted, the new one presents the non-zeros as close to the diagonal as possible.

Again, bandwidth can be calculated just by observing the scheme. For the first matrix, the distance between the diagonal and the furthest element is $\beta(A_0) = 7$ while for the reordered one the semibandwidth is $\beta(A_r) = 2$. Naturally, the same values are obtained than the ones calculated for the graphs.

6.4.3 Finding a starting vertex

The RCM algorithm requires an input node to create the reordering. This section presents an approach for deciding potential optimal starting nodes. It is presented in a separate section because it is an independent method from RCM that can be used by other reordering algorithms or even in other types of graph problems due to its great usefulness.

6.4.3.1 Pseudo-peripheral nodes

The optimal initial node for RCM is the one that produces a reordering with the lowest matrix bandwidth. This can be understood as looking for the pair of nodes separated by a maximum distance.

These nodes should be at opposite ends of the reordering and therefore be the beginning and the end of the new numbering. The ones that satisfy this condition are known as peripheral nodes. A mathematical description of this type of node is provided below in order to facilitate the understanding of the algorithm.

Using the terminology of [38], consider a connected graph $G(X, E)$ where X defines the groups of nodes and E the group of edges. The distance between two nodes $d(x, y)$ in G is measured by the minimum number of nodes through which it is required to pass to get from one to the other.

Thus, the eccentricity $\ell(x)$ of a node x is the highest minimum distance to another node.

$$\ell(x) = \max\{d(x, y) : y \in X\} \tag{6.7}$$

The diameter of G is given by the highest eccentricity between all the nodes.

$$\delta(G) = \max\{\ell(x) : x \in X\}$$

This way, a peripheral node is a node $x \in X$ whose eccentricity is equal to the diameter of the graph $\ell(x) = \delta(G)$.

Finding a peripheral node is a classic problem in graph theory. However, the computing effort required is very high due to its elevated computational complexity. That is the reason why George and Liu proposed a much simpler algorithm to find a pseudo-peripheral node in [29].

A pseudo-peripheral node resulting from such an algorithm has a high eccentricity within the network. Although its eccentricity does not have to be equal to the diameter of the graph, experience shows that it is still a very good starting node for RCM. Moreover, avoiding such computational effort is much more desirable.

The algorithm makes use of the rooted level structure of a graph. In the following, this concept is presented using the same example as before.

6.4.3.2 Rooted level structure

The rooted level structure of a graph is an arrangement of its nodes based on a tree-like structure proposed by Arany et al. in [39].

To structure the graph by means of this method, an initial node $x \in X$ is chosen. This node is the root node r and it is the only node in the first level of the structure $L_0(x) = \{x\}$. The next level contains the direct neighbours of the first node $Adj(x)$ and it can be expressed as $L_1(x) = Adj(L_0(x))$. The next levels contain the adjacent nodes of the previous level following the same procedure. However, nodes that has been already put in a previous level are not included again. The expression below describes a generic level.

$$L_i(x) = Adj\big(L_{i-1}(x)\big) - L_{i-2} \tag{6.8}$$

By using this structure, the last level of a node x is $L_{\ell(x)}$ where $\ell(x)$ is the previously defined eccentricity of the node. Therefore, the level rooted structure $\mathcal{L}$ rooted at x is defined as follows.

$$\mathcal{L}(x) = \{L_0(x), L_1(x), \dots, L_{\ell(x)}(x)\} \tag{6.9}$$

The eccentricity of x can be understood as the length of $\mathcal{L}(x)$.

For a better understanding of the concepts just exposed, they are now applied to the example of this section. The following scheme presents the root level structure of the graph presented in Figure 51 rooted at node 1. Note that it is exactly the same graph than before but its nodes have been visually redistributed to present the level structure.

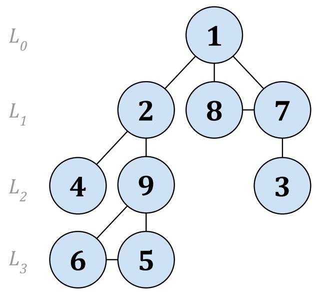


Figure 58. Level structure of the example graph rooted at node 1

The procedure is simple, the direct neighbours of 1 are nodes 2, 8 and 7; therefore, these make up level 1. The rest of the levels are equally conformed. Since the eccentricity of node 1 is $\ell(x_1) = 3$, the root level structure has 3 levels.

Note that in this case, node 1 is clearly not a peripheral node. The three nodes with maximum eccentricity are nodes 3, 5 and 6 with $\ell(x_3) = \ell(x_5) = \ell(x_6) = 5$. Therefore, the graph diameter is $\delta(G) = 5$. In this case, due to the simplicity of the graph, these results can be found visually. However, this is clearly not an option for huge systems like the ones used these days. For these cases, it is required to make use of algorithms such as the one presented.

6.4.3.3 Algorithm to find a pseudo-peripheral node

After the relevant background notes on graph theory, the algorithm for finding a pseudo-peripheral node that is suitable to be an initial node in RCM can be presented.

The algorithm consists of constructing level structures rooted at different nodes and comparing their eccentricity. The objective is to find a node with high eccentricity. In this case, the starting root node is completely arbitrary.

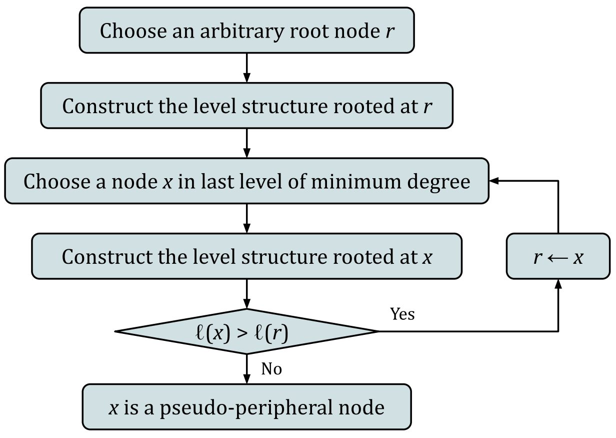

Choose an arbitrary root node r
Construct the level structure rooted at r
Choose a node x in last level of minimum degree
Construct the level structure rooted at x
$r \leftarrow x$
Yes
$\ell(x) > \ell(r)$
No
x is a pseudo-peripheral node

Figure 59. Scheme of the algorithm to find a pseudo-peripheral node in a graph

In the experiments conducted by the authors with all types of matrices, the algorithm never required more than two iterations to find a pseudo-peripheral node. This makes it a fast and effective algorithm for finding such nodes.

The algorithm is now applied to the example graph.

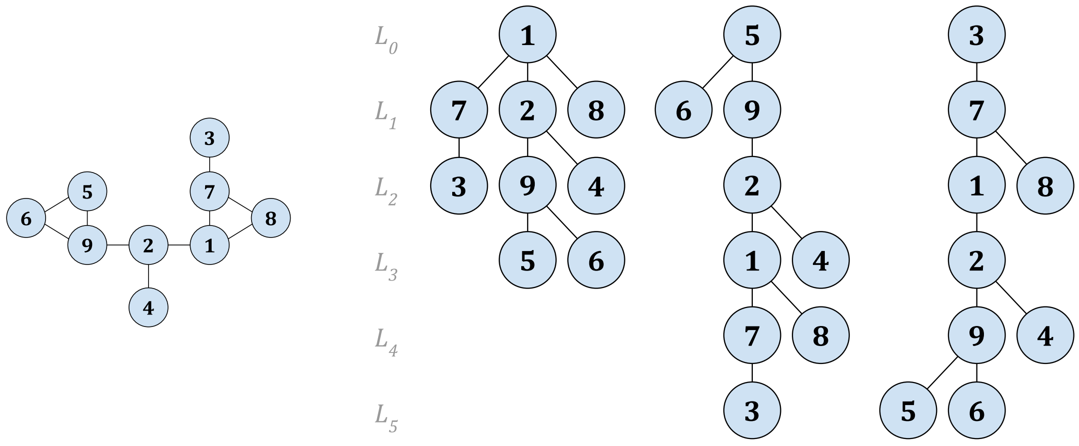


Figure 60. Example of the application of the algorithm for finding a pseudo-peripheral node

Again, node 1 is chosen as the root. The root level structure is constructed and the nodes in the last level are $L_3(x_1) = \{x_5, x_6\}$. Both have the same degree, so node 5 is arbitrarily chosen. Thus, a new level structure rooted at node 5 is created

The eccentricity of the new node is greater than the eccentricity of the root $\ell(x_5) > \ell(x_1)$ so the end condition is not satisfied and the loop starts again. Now, the length of the structure is $\ell(x_5) = 5$. Node 3 is the only node remaining in this last level. The level structure is constructed for node 3 and its eccentricity is the same than the eccentricity of node 5.

The process ends and, therefore, node 3 is a pseudo-peripheral node. As the length is the same, nodes 5 and 6 would also be feasible pseudo-peripheral nodes. Node 5 has been chosen as initial node in RCM example in Figure 55.

6.5 Performance analysis

The objective of this section is to study the magnitude of improvement in computational performance that occurs when applying the algorithms and techniques presented in this report.

C++ and MATLAB code is used to test the performance of the SpMV computational kernel on different matrices. The aim of the test is to measure the time it takes to calculate the operation before and after a rearrangement using the Reverse Cuthill-Mckee algorithm.

The sparse matrices used to carry out the experiment are obtained from *The SuiteSparse Matrix Collection* in [28]. This repository contains sparse matrices of any kind and nature. They are the result of problems of different nature (heat and temperature, Navier-Stokes, electromagnetism, graph problems, etc.). This demonstrates the great usefulness of these optimisation techniques in all kinds of fields.

First, the most important elements of the code that has been programmed to carry out the experimentation are presented. Then, the matrices used and their characteristics are presented. The method used to carry out the test and the results of the computation times are presented. Finally, the results are analysed and the pertinent conclusions are drawn.

6.5.1 Code implementation

The C++ program developed to perform the experiment includes the following steps.

1. A sparse matrix is imported from the repository into a **`SparseMatrix`** object.
2. The sparse matrix is turned into a graph as a **`Graph`** object.
3. A pseudo-peripheral vertex of this graph is found.
4. The RCM is applied to the graph using as starting node the pseudo-peripheral vertex.
5. A new order is obtained as an array of integers.
6. A new sparse matrix is created by reordering the original one using the RCM order.
7. A new sparse matrix is created by reordering the original one using random order.
8. SpMV kernel is performed in the original and randomly and RCM reordered matrices while time of computation is being measured.
9. Temporal data is exported to MATLAB so it can be represented.

The following sections describe the self-created most important classes and methods used to implement the main program just described.

6.5.1.1 Sparse Matrix

The class **`SparseMatrix`** has been created to represent a sparse matrix. It allocates the non-zero values of symmetric sparse matrices making use of the CSR storage method. Thus, this class has three main attributes.

- **`row_off`**. Array of integers that stores the offset of each row. It has the length of N, the number of rows and columns of the matrix.
- **`col`**. Array of integers that stores the index of the column where non-zero values are allocated. Its length is nnz, the number of non-zeros.

- **`val`**. Array of doubles that allocates the non-zero values of the sparse matrix. Its length is also *nnz*.

This class also have other attributes and methods. For example, a special class constructor has been required to import sparse matrices from the repository using a *.mtx* format.

6.5.1.2 Graph

In order to apply the reordering to the sparse matrices using RCM, a class **`Graph`** is also created. An array of adjacency lists is used to represent a graph in the program as the main attribute of the class. The adjacency list of a node $x \in X$ of a graph G is a list containing all its direct neighbour nodes. There are as many adjacency lists as there are nodes in the graph, so the length of the array is N. Figure 61 presents an example to expose the concept and the storage method used in the program. This data structure has been obtained and partially modified from [36].

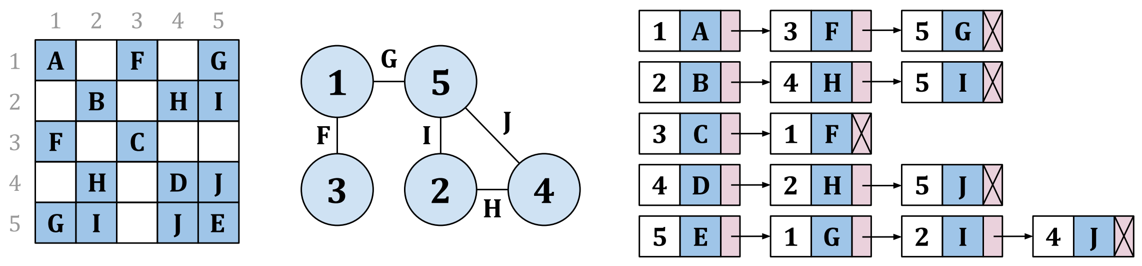


Figure 61. Implementation of a graph as an array of adjacency lists

The sparse matrix of the left is turned into the graph at the centre as explained in Section 6.3.3. Note that, in this case, the values of the edges representing the value between the nodes are shown.

Unsorted lists are used to obtain the adjacency list of each node. In this case, there exists an array of length $N = 5$ with five unsorted lists. Each element of the unsorted list is a C++ *struct* that contains the number of the node (presented in the figure in a white background), the correspondent value (presented in blue) and a pointer to the following *struct* (shown in purple). In the last element of each unsorted list, the pointer is set to *null* (represented by an *X*).

For example, row 1 (or column, as the matrix is symmetric) is represented by node 1. As in the first row of the sparse matrix there are values in the columns 1, 3 and 5, these are the resulting neighbour nodes as can be appreciated. The unsorted list of the node 1 is the first element of the array of unsorted lists. Thus, the *struct* referring to the node 1 is the first element of this unsorted list. This *struct* contains the number of the node 1, its own value *A* and a pointer to the *struct* of node 3. The *struct* of the third node points to the *struct* of node 5. As there are no more neighbour nodes of the first, this last *struct* points to *null*. The same procedure is applied to the other four nodes.

The constructor of this class requires an object of the **`SparseMatrix`** as input as it has been developed for the sole purpose of representing sparse matrices.

6.5.1.3 Algorithms

Two main algorithms have been programmed to reorder the sparse matrices.

- **`pseudo_peripheral_vertex(Graph g)`**. This method is the implementation of the algorithm discussed in Section 6.4.3.3. The function receives a graph as input and returns the number (int) of a pseudo-peripheral vertex. This node is the starting node of the RCM algorithm.
- **`order_RCM(Graph g, int starting_node)`**. The function implements the Reverse Cuthill Mckee algorithm to reorder the input graph. The algorithm makes use of the starting node obtained in the previous algorithm. It returns an array of integers of length N with the new order.

Moreover, other subroutines required to perform the algorithms have been needed. For example, a function that finds the degree of each node or a function to reorder a sparse matrix with the node.

6.5.1.4 SpMV

The SpMV is a computational kernel. Its implementation depends on the kind of storage of the sparse matrix. In this case, as CSR is the method used, the operation can be performed the following way in a C++ program [32].

```
for (int i = 0; i < N; i++){
   y[i] = 0.0;
   for (int j = row_off[i]; j < row_off[i + 1]; j++)
      y[i] += val[j] * v[col[j]];
}
```

In this algorithm, **`row_off`**, **`val`** and **`col`** are the attribute arrays of a **`SparseMatrix`** object, **`v`** stands for the source vector and **`y`** for the destination one. The logic of this method is the described in Section 6.4.1.

6.5.2 Studied sparse matrices

The aim of this last part of the report is to test the performance improvement that occurs when applying reordering algorithms to different types of sparse matrices. To do so, a SpMV operation $y = Ax$ is realized if different matrices.

The sparse matrices A used to carry out the experiment are actual matrices obtained from *The SuiteSparse Matrix Collection* in [28]. These matrices are the result of any kind of problem, from CFD to electromagnetics or graph problems. This fact also demonstrates the great usefulness of these optimisation techniques.

The table below presents the list of the nine matrices chosen. The name of the matrices is the one extracted from the repository and it is the way they are being referenced in this report. All matrices are symmetric and have only one strongly connected component.

From left to right, N stands for the number of rows and columns, nnz denotes the number of non-zeros, and nnz/row represents the average number of non-zeros in each row or column. The matrices are ordered according nnz.

Sparse matrix name	$N\ (\cdot 10^4)$	$nnz\ (\cdot 10^6)$	nnz/row
p_driven_cav	0.25	0.0124	4.96
Pres_Poisson	1.48	0.7158	48.36
2cubes_sphere	10.15	1.6473	16.23
poisson3Db	8.56	2.3749	27.74
cfd2	12.34	3.0854	25.00
ecology1	100.00	4.8825	4.88
netherlands_osm	221.67	4.9960	2.25
adaptive	681.57	27.2486	4.00
road_usa	2394.73	57.7086	2.41

Table 8. Sparse matrices used to test the performance improvement

The first matrix *p_driven_cav* is the only one that has not been obtained from the repository. This is the coefficients matrix obtained from the Poisson's problem for the pressure in the lid driven cavity case (Section 5.2) for a regular 50×50 mesh. This way, it can also be studied how an ideal case responds to an optimisation algorithm.

Finally, in order to have a proper perspective of the results obtained, the spy plot of the nine matrices is presented in Figure 62. The spy plot is a representation of the location of the non-zero values in a sparse matrix. It is of great utility in qualitatively evaluating the nature of the sparse matrix. The symmetry, the sparsity and the approximate matrix bandwidth can be clearly appreciated.

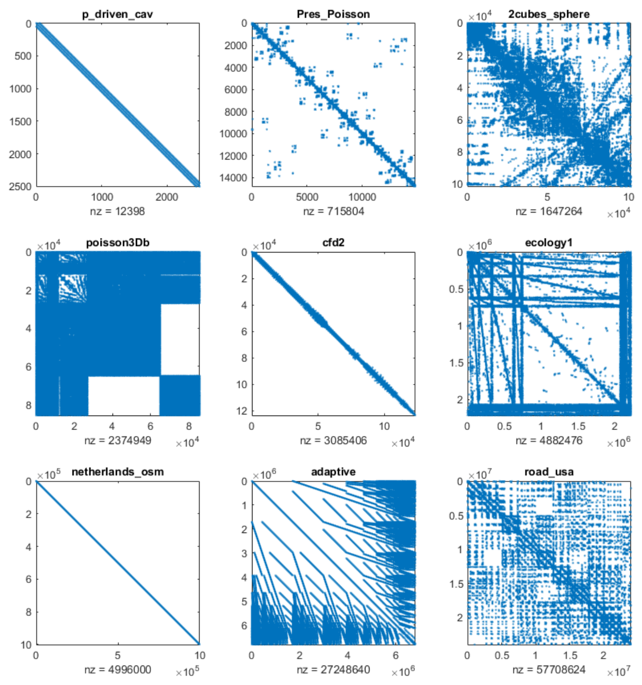


Figure 62. Spy graph for the 9 studied sparse matrices

6.5.3 Results

To measure the reduction of time after the RCM optimisation, the matrices have been tested in three different forms.

- Original matrix. The test has been performed directly on the matrix obtained from the repository.
- RCM reordered matrix. RCM algorithm has been applied to each matrix in the manner exposed in this report (Section 6.4.2) and then tested.
- Randomly sorted matrix. A random order has been generated and applied to the tested original matrix.

Figure 63 illustrate the appearance of the sparse matrices after RCM and random reordering by using spy plot again.

For the RCM reordered matrices, all matrix bandwidths have been clearly reduced, as all values are concentrated near the diagonal. Obviously, this reordering has a greater impact in matrices that had its non-zero values more dispersed such as *road_usa* or *adaptive*. Other matrices like *cfd2* or *netherlands_osm* remain almost the same as their bandwidth was already narrow. Finally, some bandwidths cannot be reduced like others due to the nature of the problem as in *poisson3Db*.

In contrast, the randomly sorted matrices have increased the matrix bandwidth to the maximum. With the exception of the first one, the values are so widely dispersed that no shape is visible in the spy plot. This rearrangement only makes sense for the purpose of the comparative study as it cannot be efficient in any case.

Of course, the matrices maintain their initial descriptive characteristics as *N* or *nnz*. They also remain symmetric.

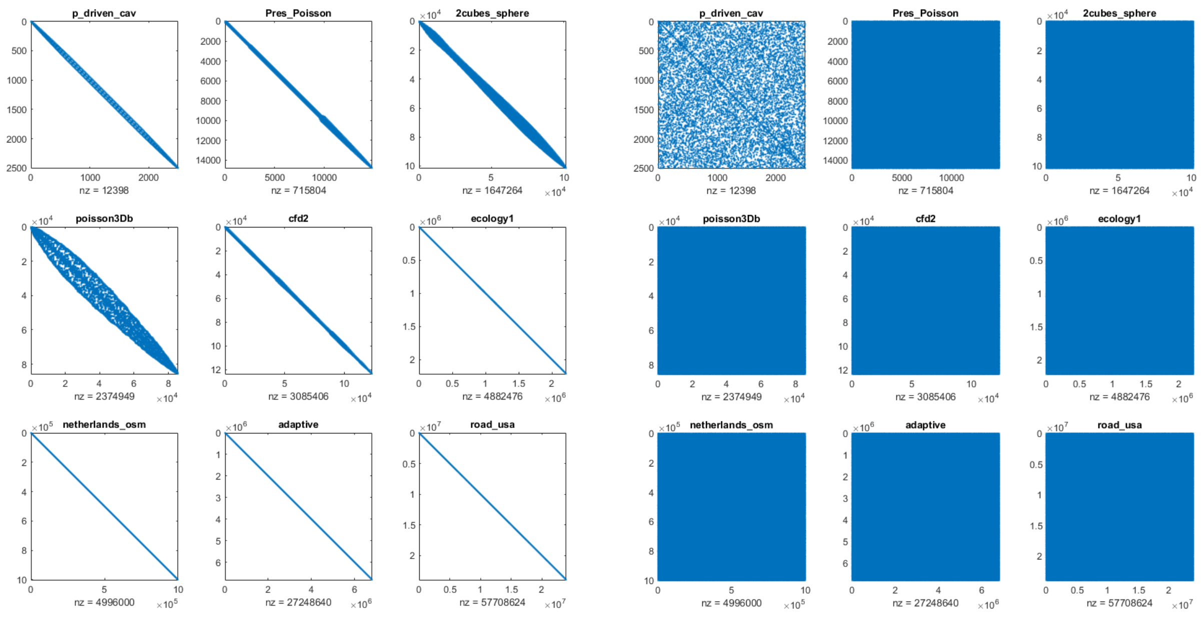


Figure 63. RCM (left) and randomly (right) reordered sparse matrices

In order to test the improvement in computing time, the SpMV has been applied several times to each matrix. In fact, the operation has been performed 50 times in the first seven matrices for each ordering to get a more accurate average time. That is to say, 150 operations per matrix. The time of computation have been measured separately for each ordering and compared to the time for the original matrix. SpMV for the last two matrices has just been applied twice for each ordering due to their massive size.

The vector utilized to perform the multiplication is a randomly generated N-size dense vector, i.e., all its elements are non-zero. The elements of this vector are allocated in consecutive order in the computer's memory.

The absolute time that is required to compute the SpMV is not as relevant as the difference between the original matrices and the reordered ones. That is the reason why all the values of time presented are referenced to the time required by the original matrix to perform the operation. Results are presented in Table 9.

Sparse matrix	Original	RCM	Random
p_driven_cav	1.000	1.014	1.178
Pres_Poisson	1.000	0.976	1.389
2cubes_sphere	1.000	0.869	1.341
poisson3Db	1.000	0.685	1.128
cfd2	1.000	1.066	1.937
ecology1	1.000	0.599	2.038
netherlands_osm	1.000	0.945	2.191
adaptive	1.000	0.397	3.701
road_usa	1.000	0.626	2.836

Table 9. Relative computing time depending on the reordering

The results are also graphically presented in Figure 64. This plot allows a better qualitative comparison of the outcomes.

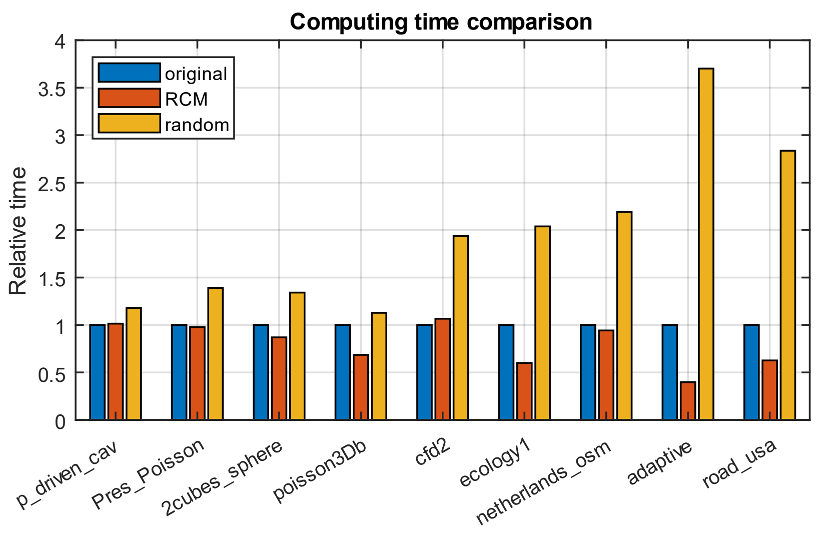


Figure 64. Relative computing time comparison

The first impression is that results differ considerably depending on the matrix. Nevertheless, some general conclusions can be reached.

The most obvious conclusion is that the randomly reordered matrices perform much worse in any case. That is an expected result as the values are spread across the matrix (and therefore memory) leading to an extremely poor CPU and cache performance. However, note that not all matrices are affected the same way. This is where the point of having conducted such an experiment lies and is discussed later, as a random rearrangement has no real value.

A second point to focus is the effectiveness of the RCM reordering. At first glance, the majority of matrices clearly performed better after this rearranging. Except for *p_driven_cav* and *cfd2*, all the other operations have been computed in less time. Once again, though, it can be seen that some matrices have exhibited a much greater improvement.

The aim of the following sections is to discuss why these differences occur. The matrices presented can be classified using two criteria: their shape and their size

6.5.3.1 Shape analysis

By looking at Figure 62, two main groups can be easily be done in terms of shape. Specifically, focusing on the bandwidth of the matrices. On one hand, the three relatively *narrow matrices* are *p_driven_cav*, *cfd2* and *netherlands_osm*. On the other hand, the three most *dispersed matrices* could be *poisson3Db, ecology1* and *adaptive*.

Based on the results Table 9, one can compare *netherlands_osm* as narrow and *ecology1* as dispersed. The choice of these two matrices is convenient as they have a similar size and that is not a factor that may misrepresent results. Size effect is discussed in the following section.

The results of the performance upgrade for these two matrices when RCM is applied are clearly differenced. While the narrower one presents a slight improvement of 5.55% time reduction, the dispersed matrix reduced the calculation time by 40.1%. The same pattern takes place with matrices of same nature.

The reason of this difference is evident, narrowest matrices already had a great operation performance based on the cache usage. Therefore, reordering the matrix has a low effect on improving or indeed worsening performance. In fact, that is the case for the two other narrow matrices: *cfd2* and *p_driven_cav*. The order of the latter was already almost perfect since it was the resulting matrix of an ideal case (the lid driven cavity problem) with a regular mesh. This way, its performance has decreased when RCM has been applied.

In contrast, matrices that are more dispersed had a much worse performance before reordering because of the explained poor cache performance. In those cases, applying RCM reordering really makes a difference.

Finally, analyzing the effects on the random order, the two example matrices present similar results. The narrower one, *netherlands_osm*, performs worse as the contrast with the original ordering is greater. However, both cases reaffirm that this type of ordering is not even close to optimal, as computing time more than doubles.

6.5.3.2 Size analysis

The size of the matrix can be defined both by the number of rows and columns N or the number of non-zero elements nnz. In this case, the matrices are ordered both in Figure 62 and in Table 9 by nnz.

Focusing now only on dispersed matrices (so the shape does not affect anymore), one can compare *2cubes_sphere* and *poisson3Db* as relatively small matrices ($nnz \approx 2 \cdot 10^6$) with *adaptive* and *road_usa* as the hugest with $nnz_a = 27 \cdot 10^6$ and $nnz_r = 58 \cdot 10^6$ respectively.

While the biggest matrices show the greatest improvement when RCM is used, they are also worst performers when the random order is applied. The most sharply contrasted case is for *adaptive*. When computing the SpMV in this matrix with the rearrangement obtained from RCM, the time of computation is 39.7% of the original time. In contrast, when randomly reordered, the time increases about 3.7 times. That is, from the optimum reordering to the worst, the time gets increased more than 9.4 times. This is a great example of the time savings that can be achieved.

Again, the reason is how the cache and the CPU performs in small or big matrices. For smaller matrices, the reordering does not have such an effect because the rows fit better in the cache. For instance, if vector the size of a matrix row could be fully allocated in the cache memory, the cache misses that cause the greatest delays would simply not occur. The same can be translated to bigger small matrices that, even though they cannot fit the entire vector into cache, there are very few cache misses in the access to its values.

In contrast, for larger matrices whose vector clearly cannot be fully allocated in cache memory, cache misses will happen much more frequently without proper ordering. In fact, when the matrix is randomly sorted, almost every memory access may be a cache miss. This situation leads to extremely low performance. In these cases, a reordering such as the one provided by RCM can increase the number of cache hits, significantly reducing the computing time.

6.6 Conclusions

Nowadays, the numerical resolution of real problems requires great computational power. Elements such as turbulence or boundary layer in three-dimensional CFD problems require very fine grids that involve solving large systems of equations. The computational time becomes enormous and a simulation of a seemingly simple element can take hours or days.

The main bottleneck in solving scientific and engineering computing problems is the processor's access to data in memory. The vast majority of problems are solved by memory-bound functions, i.e., methods whose execution time is mostly determined by the number of memory accesses as opposed to the computation time. This hardware constraint forces researchers to think of techniques to improve their programs and reduce computation time and costs.

One of the most widely used kernels in the field of numerical resolution is the Sparse Matrix Vector Multiplication (SpMV) and, due to its nature, the access to the vector elements in the operation has a very poor performance. The objective of the last section was to test whether sparse matrix reordering methods could be used to reduce the matrix bandwidth and consequently reduce the computational time required to realise the SpMV computational kernel.

The Reverse Cuthill Mckee algorithm has proved to be an efficient method for matrix bandwidth reduction as can be seen in the sparse matrix representations before and after reordering. The original version of this algorithm was presented in [35] in 1969 by the authors of the same name. Since then, other more sophisticated versions obtaining similar results in less time have been developed, for example, the GPS algorithm presented in [40]. Nevertheless, RCM remains today one of the most studied and widely used reordering algorithms in more complex parallel distributions.

The results in terms of computational time improvement have varied depending on the type of matrix treated. Matrices that already had a narrow bandwidth did not show an appreciable time reduction as their initial sorting was already good enough. Similarly, reordering does not offer much performance gain for small matrices as much of the corresponding SpMV vector could be allocated in the cache memory.

In contrast, operations on very large matrices with poor initial orderings have shown great improvements in efficiency, reducing their computation time by up to 60%.

In the random reordering of these matrices, it has been demonstrated the great importance of creating so-called cache-friendly programmes that take into account the functioning of this memory and avoid the number of failed accesses. In random reordering, where the worst-case scenario for cache performance is used, computation times have increased by as much as 370% of the original time.

6.7 Next steps towards parallelisation

The techniques presented have been only a minimal part of the methods used today in computational science and engineering.

When the problem size is so large, solving it by processing a single instruction at a time is inefficient and slow, no matter how fast the processor is. Parallel computing makes use of multi-core processors to perform calculations or processes simultaneously. In the last decades, high performance computing has made use of parallelism to solve problems that would be intractable in serial programming. This is done by dividing a huge problem into smaller ones that can be solved at the same time [41].

This method of computation, which is often the only one available, brings with it great challenges for the elaboration of the resolution codes. A simple operation in serial programming can be quite complex in parallel.

For example, a parallel SpMV shares with its serial equivalent the efficiency problems associated with irregular CPU access to memory. However, the same re-ordering methods presented in this report may not be a solution for improving computing time since the memory allocation scheme changes drastically.

A distributed-memory parallel SpMV splits the rows of the sparse matrix into as many subsets as cores are available to perform the computation. This technique is known as matrix partitioning and is a common preprocessing step. In this case, some new strategies are required in order to keep the number of elements close to same value in each core so computation load is balanced [42].

Another typical drawback of this scheme is dealing with communications. Frequently, an element of a sparse matrix that has been partitioned among the cache memory of the different process cores requires an element located in another core to perform an operation. Minimising these communications is critical to good performance and, again, can be achieved by reordering the elements [43].

The conclusion that can be drawn is that code parallelisation opens up a potentially huge improvement in computational time that, however, requires a sophistication of optimisation techniques. Thus, a wide range of researchers are focusing their efforts on improving and developing new techniques for parallel coding enhancement.

7

Conclusions and future work

These last sections present the conclusions of the project and the work that remains to be done in the future. This is followed by an estimate of the budget and the social and environmental impact of this work.

7.1 Project conclusions

The main objective of this project was to introduce numerical methods as a tool for solving heat transfer and fluid dynamics problems. In order to implement C++ programs capable of solving problems of this nature, it has been necessary to study the corresponding analytical equations and discretise them by means of the finite volume method.

Three major problems have been studied: heat conduction in a solid, convection-diffusion of a property in a fluid and the pressure and velocity field of a fluid in a cavity through the Navier-Stokes equations. Each of these problems has been solved using a system of linear equations that was obtained according to the concrete conditions. In the different sections it has been discussed both the physical aspects that influenced the result of the problem and the treatment given to it by the computer.

The cases studied are a good introduction to the field of Computational Fluid Dynamics, a discipline that has proven to be very useful in approaching and providing a good model of the solution to great variety of problems.

During the development of these three problems, it has been verified that as a more precise solution was required, the number of unknowns grew and, consequently, the computation time of the computer increased exponentially. These are academic problems that can be fairly solved without having to deal with long times of computation due to their simplicity. However, reducing this time is a major challenge when dealing with real problems that require much more processing.

The last part of the project has focused on an introduction to the treatment of large problems. The first step has been to identify what is the main issue facing the processor when performing operations needed in scientific computing.

The performance of the Sparse Matrix-Vector Multiplication kernel (very common in solving problems such as those studied) has proved to be of great importance in determining the computation time. This operation is memory-bound, i.e. its performance is limited by the access of the CPU to the cache memory.

The Reverse Cuthill Mckee algorithm has been programmed. An algorithm that reorders a sparse matrix by decreasing its bandwidth. This reordering allows much more regular and predictable memory access to the elements of the multiplied vector when performing a SpMV.

By means of real problem matrices obtained from a repository, it has been verified that the application of this technique allows to reduce the computation time in most cases. The effectiveness of the method depended to a large extent on the size and initial arrangement of the matrix. Larger and originally unordered matrices have reduced their computing time in SpMV by up to 60%, proof of the great effectiveness of this method.

7.2 Future work

This project has been a brief introduction to *Computational Fluids Dynamics*, *Computational Science and Engineering* and *High-Performance Computing* fields. Three major disciplines that encompass much of today's engineering research. Therefore, there is still much to learn and work on.

On the fluid mechanics side, the learning process will be continued through problem solving. The next step is the treatment of the Navier-Stokes equations in turbulent regime. Further complications such as complex geometries, compressible flow at high Mach numbers, etc. will also be studied.

On the purely computer side, there is also a wide range of options. First of all, the aim is to improve programming practices in C++, one of the most widely used languages in scientific programming today, in order to make codes cleaner, more efficient and faster. Starting to use Linux as an operating system in an efficient way is also key to be able to optimise the processes that interact closer to the compiler.

Finally, on the scientific computation side, the first imminent step is to get introduced into parallel computing and all its associated techniques. This step is intimately related to the fact of dealing with more complex problems as it is where the benefit of parallel implementation is realised.

Environmental and social impact

This project has a low impact both environmentally and socially due to its academic character. However, the concepts presented have great potential in the development of technologies with a major improvement in people's lives.

Scientific supercomputing makes it possible to create models to forecast weather on a large scale, the evolution of global warming or, recently, even the prediction of COVID-19 cases within a society. These are just a few examples of how such simulations can be of great help to everyone.

On a smaller scale, heat transfer and CFD programs like the ones discussed in this report are of great utility to simulate the operation of new technologies or to make the existing ones more efficient and less resource-consuming. If the power of computation was not available, many technical advances would not have occurred because the high cost of experimentation would have made it impossible.

On the other hand, it is also a fact that today's large computing centres consume a large amount of energy resources as they grow exponentially in size. This is one of the reasons why research is being done into cleaner and more efficient computing technologies, both for greener and cheaper systems. In large computational complexes, optimisation algorithms that improve performance and computational time such as the ones introduced here also can save enormous amounts of resources.

References

[1] R. Temam, *Navier-Stokes equations : theory and numerical analysis*. Amsterdam [etc.] : North-Holland Pub. Co., 1979.

[2] J. Oliver and C. Agelet de Saracibar, *Mecànica de medis continus per a enginyers*. Barcelona : Edicions UPC, 2003.

[3] J. M. Bergadà Granyó, *Mecánica de fluidos : breve introducción teórica con problemas resueltos*. Iniciativa Digital Politècnica, Oficina de Publicacions Acadèmiques Digitals de la UPC, 2017.

[4] E. R. G. Eckert and R. M. Drake Jr, "Analysis of Heat and Mass Transfer," 1972.

[5] S. V. Patankar, "Numerical heat transfer and fluid flow." Taylor & Francis, Boca Ratón, 1980.

[6] W. M. Rohsenow, J. P. Hartnett, and E. N. Ganic, "Handbook of Heat Transfer Fundamentals," 1985.

[7] J. F. Grcar, "John von Neumann's analysis of Gaussian elimination and the origins of modern numerical analysis," *SIAM Rev.*, vol. 53, no. 4, pp. 607–682, 2011, doi: 10.1137/080734716.

[8] J. Boudet, "Finite volume methods," *Comput. Fluid Dyn.*, no. January, pp. 1–24, 2011, doi: 10.1007/978-3-319-99693-6_4.

[9] W. J. Minkowycz, "Handbook of numerical heat transfer," 1988.

[10] L. Davidson, "TDMA Solver," *Numer. Methods Turbul. Flow*, 2008.

[11] J. Dongarra, "The Gauss-Seidel Method," *Stationary Iterative Methods*, 1995. http://www.netlib.org/linalg/html_templates/node14.html#figgs (accessed Aug. 14, 2021).

[12] CTTC, "A Two-dimensional Transient Conduction Problem," 2013.

[13] A. G. Smith and R. M. Hutton, "The numerical treatment of advection: a performance comparison of current methods," *Numer. Heat Transf.*

[14] CTTC, "Fractional Step Method - Staggered and Collocated Meshes," *Course Numer. Methods Heat Transf. Fluid Dyn.*, 2013.

[15] F. X. Trias and O. Lehmkuhl, "A self-adaptive strategy for the time integration of Navier-Stokes equations," *Numer. Heat Transf.*, vol. Part B: Fu, pp. 116–134, 2011.

[16] U. Ghia, K. N. Ghia, and C. T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *J. Comput. Phys.*, vol. 48, no. 3, pp. 387–411, Dec. 1982, doi: 10.1016/0021-9991(82)90058-4.

[17] C. Rozov, Vladyslav Stuhlpfarrer, Marco Fernández Osma, Mario Breitsamter, "Small-Disturbance-CFD-based Aircraft Flutter Investigation Including Powered Engine Model," 2019.

[18] Top500.org, "Top 500 supercomputers | June 2021," 2021. https://www.top500.org/lists/top500/2021/06/ (accessed Sep. 02, 2021).

[19] Our World In Data, "Supercomputer Power (FLOPS), 1993 to 2020," 2020. https://ourworldindata.org/grapher/supercomputer-power-flops (accessed Sep. 02, 2021).

[20] A. Buluç, J. Gilbert, and V. B. Shah, "Implementing Sparse Matrices for Graph Algorithms," *Graph Algorithms Lang. Linear Algebr.*, vol. 94720, pp. 287–313, 2011,

doi: 10.1137/1.9780898719918.ch13.

[21] J. L. Hennessy, D. A. Patterson, and C. Kunia, *The Architecture of ComputerHardware, System Software, and Networking*. 2013.

[22] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," *ACM Trans. Internet Technol.*, vol. 5, no. 2, pp. 299–327, 2005, doi: 10.1145/1064340.1064341.

[23] C. Carvalho, "The gap between processor and memory speeds," *Icca*, pp. 27–34, 2002, [Online]. Available: http://gec.di.uminho.pt/discip/minf/ac0102/1000gap_proc-mem_speed.pdf.

[24] F. X. Trias, O. Lehmkuhl, A. Oliva, C. D. Pérez-Segarra, and R. W. C. P. Verstappen, "Symmetry-preserving discretization of Navier–Stokes equations on collocated unstructured grids," *J. Comput. Phys.*, vol. 258, pp. 246–267, Feb. 2014, doi: 10.1016/J.JCP.2013.10.031.

[25] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, "Scientific Computing Kernels on the Cell Processor," *Lawrence Berkeley Natl. Lab.*, 2007.

[26] J. González-Domínguez, O. A. Marques, M. J. Martín, and J. Touriño, "A 2D algorithm with asymmetric workload for the UPC conjugate gradient method," *J. Supercomput.*, vol. 70, no. 2, pp. 816–829, 2014, doi: 10.1007/s11227-014-1300-0.

[27] L. O. Mafteiu-Scai, "The Bandwidths of a Matrix. A Survey of Algorithms," *Ann. West Univ. Timisoara - Math.*, vol. 52, no. 2, 2015, doi: 10.2478/awutm-2014-0019.

[28] U. of Florida, "SuiteSparse Matrix Collection," 1970. https://sparse.tamu.edu/ (accessed Aug. 12, 2021).

[29] A. George, J. Liu, and E. Ng, "Computer Solution of Sparse Linear Systems," 1994.

[30] J. Pitt-Francis and J. Whiteley, *Guide to Scientific Computing in C++*. 2018.

[31] "Built-in types (C++) | Microsoft Docs," *Microsoft documentation*, 2020. https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=msvc-160 (accessed Aug. 16, 2021).

[32] Dimitar Lukarski, "Sparse Matrix-Vector Multiplication and Matrix Formats," *Uppsala Univ.*, 2013.

[33] U. Borštnik, J. Vandevondele, V. Weber, and J. Hutter, "Sparse matrix multiplication: The distributed block-compressed sparse row library," *Parallel Comput.*, vol. 40, no. 5–6, pp. 47–58, May 2014, doi: 10.1016/J.PARCO.2014.03.012.

[34] P. R. Almeida Benítez and J. R. Franco Brañas, "Reducción del Ancho de Banda de Matrices en el Algoritmo Go-Away para Mallas Regulares," *Divulg. Matemáticas*, vol. 7, no. 1, pp. 1–12, 1999.

[35] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," *Proc. 1969 24th Natl. Conf. ACM 1969*, pp. 157–172, 1969, doi: 10.1145/800195.805928.

[36] A. George and J. Liu, "Computer solution of large sparse positive definite systems." Prentice-Hall, Englewood Cliffs, N.J, 1981.

[37] C. Wang, C. Xu, and A. Lisser, "Bandwidth Minimization Problem," pp. 190–203, 2014.

[38] C. Berge, "The theory of graphs and its applications," *Bull. Math. Biophys.*, vol. 24, pp. 441–443, 1962, [Online]. Available: https://doi.org/10.1007/BF02478000.

[39] I. Arany, W. F. Smyth, and L. Szoda, "An improved method for reducing the bandwidth of sparse symmetric matrices," *Inf. Process. 71*, 1972.

[40] N. E. . Gibbs, J. . William G . Poole, and P. K. . Stockmeyer, “An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix,” vol. 13, no. 2, pp. 236–250, 1976.

[41] G. S. Almasi and A. Gottlieb, “Highly Parallel Computing,” *Benjamin/Cummings Publ. Company, Inc*, 1989.

[42] A.-J. N. Yzelman and D. Roose, “High-Level Strategies for Parallel Shared-Memory Sparse Matrix-Vector Multiplication,” *IEEE Trans. parallel Distrib. Syst.*, vol. 25, no. 1, pp. 116–125, 2014, doi: 10.1109/TPDS.2013.31.

[43] A. Bienz, L. Olson, and W. D. Gropp, “Reducing Communication Costs in the Parallel SpMV,” 2015, Accessed: Sep. 15, 2021. [Online]. Available: http://www.llnl.gov/CASC/hypre/.