Local Preconditioning for Parallel Iterative Solvers

Ph.D. Thesis

Paula Córdoba Pañella

Advisor
Guillaume Houzeaux

Computer Applications in Science and Engineering department
Barcelona Supercomputing Center (BSC-CNS)

Ph.D. in Applied Mathematics
Universitat Politècnica de Catalunya

A papá, a mamá,
a Amanda
y a mis chicos, Víctor y Ángel.

Let the wild rumpus start!

Maurice Sendak,
Where the Wild Things Are

Acknowledgements

Tengo que empezar con una confesión. La idea que me rondaba por la cabeza al acabar el Máster era coger la mochila, un avión y volver a Suecia para continuar mi viaje donde lo había dejado unos meses antes. Mi intención por aquel entonces era conocer el norte de Europa, luego coger un tren dirección a Rusia y acabar en Corea. Pero bien, un día vi el *status* del Facebook de un buen amigo, dónde ponía algo así como: '*Se busca estudiante de doctorado para entrar en el Barcelona Supercomputing Center*', y me llamó la atención. Nunca había hecho nada relacionado con las *supercomputing-cosas*, pero oye, parecía interesante y además había escuchado que hacían no sé qué de simulación de un corazón humano, del cual quería saber más. Así que decidí enviar un mail para ver qué se cocía. Después unas cosas llevaron a las otras y al poco tiempo cambié mis planes de viaje por una tesis doctoral. Aún a día de hoy me pregunto si fue una buena decisión, pero sea como fuere, así han sido las cosas y en este largo camino hay muchísimas personas que han hecho que esto haya sido (por fin) posible.

Primero me gustaría dar las gracias a Mariano, él fue quién confió primero en mi y me ayudó a conseguir la beca para poder hacer esta tesis. Si está Mariano, también tiene que estar Guillaume, porque ellos siempre están juntos en la 'ofi', y porque Guillaume ha sido quién ha dirigido esta tesis.

No me quiero olvidar de Eva, un apoyo fundamental en mis primeros meses de tesis, y que sin ella, sin nuestra coca-cola y sin el vicio de ambas por los tomates cherry esto no hubiese sido posible. En la larga lista de compañeros del BSC están de Matías (el cuerdo), Georgios, Diana, Jordi, Sergio, Álex, Maria Cristina, Guido, Dani P., Raul, Xavi y Sergio.

Con el tiempo hay compañeros que acaban siendo amigos, como es el caso de Miguel, mi compañero de congresos. Sólo él y yo sabemos que un congreso sin chinches en el hotel, no es un congreso. Echaré de menos el: *Pau, Pau, ¡deja de joder!* o el *Pau, Pau, ¡esas no son horas de llegar!*. También me gustaría destacar aquí los *fruit time*, cafés y cenas con Cristóbal y Dani G., dónde hemos arreglado el mundo $n + 1$ veces. Y para el final dejo a Egar, que no sólo es un amigo, sino que ha acabado siendo alguien de la família.

Por último también quiero acordarme de mis otros compañeros de oficina, con los que compartí casi cuatro meses en Estados Unidos. Gracias a Seid, Ahmed, Erman, Madhu y Qiyue, por hacerlo fácil todos esos meses y ¡por animarme a presentar mi trabajo!

Fuera del mundo laboral, han sido muchas y muchos, con los que he podido compartir ratos y momentos inolvidables durante este tiempo. Empezaré por los que fueron mis compañeros de piso durante más de dos años Pau y Sergi (y Andrea, mi okupa preferida), a los que traumaticé lo suficiente como para no querer oir de tesis doctorales nunca más.

Conitnuo con los físicos, desde las cervezas con Toño y Albert los Viernes, dónde la dueño del bar casi le da un algo el día que pedimos agua, a las cenas

y excursiones con Mariona, Adri, Jorge, Joana y Eulalia. También están los otros físicos Sergio, Ana, Elia, el trio Sílvia-David-Axel, Ferré y Miquel, de los que no me he separado desde el segundo día de universidad. No quiero dejarme a Pacheco, o mejor dicho Facheco, el físico más revolucionario y que me obligó a entrar en la organización de la JIPI, simplemente porque a él le daba la gana. ¡Gracias por guardarme la sudadera que nunca vi! Ni a Irene, la física olvidada, porque las birras de 2016 ya van por 2022… algún día, algún día iremos Edgar y yo a verte, ¿en 2025 quizás? O a Víctor, ¡el físico más *empanado* que he visto jamás!

De las propiedades cuánticas de Edu mejor no hablar, prefiero quedarme con vermuts *afterwork* que decidimos hacerlos coincidir en Viernes porque siempre se alargaban más de la cuenta. Gracias por subirme la moral, por confiar en mi y por decirme que era capaz en los peores momentos.

Todos estos años no hubiesen sido lo mismo sin los reencuentros Erasmus, dónde cada encuentro hace que empieces la semana con una sonrisa en la cara. Gracias Rene, Jaume, Álvaro, Amaia, Borja, Alberto y Elena.

Y como no, a Judith. Sí, hubo flechazo desde el primer momento en aquel fin de semana en Barcelona. Estaba claro que teníamos que acabar siendo familia (¡no se como aguantas a Ángel!)

A los de Berlín, Núria, Pedro, Marcel, Olga, Roger y Laura porque todos los caminos llevan a Berlín y con la tontería ya es mi segunda casa and to the English ones, to Ahmad and to his roomates Rob and Andrew. Also to Jaimito, which finally moved to Barcelona and to the guiri non-guiri, ‘Lo Charlieo’ which is responsible for one of the quotes in this thesis.

A Mickey, por buscarte los congresos, workshops y summer schools siempre en Barcelona, por enseñarme la Bruselas no turística y por los 15 días en Cuba junto a John y Toño. We're scientists, but the cool ones, right?

A mes 'gabachos' préférés: Clement, Remi et Ethiene. La vida sería mucho más aburrida sin vosotros. Gracias por los momentos en Mende, Lyon, Donosti y Barcelona. ¡Sois únicos!

Son esos momentos en los que estás fuera de casa, donde te das cuenta de los lazos que puedes llegar a tener con completos desconocidos. Sin duda alguna Urbana - Champaign y Chicago no hubiesen sido lo mismo sin Laia, Xavi, Antonio, Ronak, Tom, Sofía y Sergio. Ni sin la *comunidad hindú* de Urbana-Champaign, que me acogió desde el principio y me enseñó a bailar al más puro estilo de Bollywood, en especial a Ankita, mi compañera de piso durante esos meses y una buena amiga ahora.

De los 'holandeses' me quedo con Laura, Gloria, Nerea y Laia, que han sido una familia fuera de casa y con Mónica (ex-BSC) y Julián que me ayudaron cuando buscaba un techo donde vivir. Tampoco podían faltar los CA-TASS-LANS porque Siemens sin ellos, no hubiese sido lo mismo, ni sin La Sombra, ni sin ¿ese tío habla?, ni sin el pindakaas de las 12, ni sin… vamos que sin Alfred, Joan, Xavi, Paco, y Jordan el trabajar a 2500 km de casa no hubiese sido lo mismo. Y de esta larga lista de expats no me quiero olvidar de Pietro Santagati, el mejor team lead que me he podido encontrar a lo largo de mi carrera profesional.

Tampoco me puedo olvidar de Berta y sus cafés/cervezas *express* de camino

entre Girona y Sant Boi o la excursión fallida al Turó del Home con la niebla que no nos dejaba ver a 2 centímetros de distancia y o a Vicente, con el que los encuentros en las cenas de Jaume ha acabado siendo un amigo de esos que los son para toda la vida.

Cuatro palabras para el aragonés más alemán no podían faltar, porque al final con los años y a pesar de la distáncia hemos encontrado esos cinco minutos para vernos, contarnos qué tal y darnos un buen abrazo. Eres ingeniero, sí y aun y así, se te aprecia. Gracias por ser siempre tan... bueno tan tú, Guille.

A Dani L., porque no consigo que me dejes de hablar a pesar de que te ignoro sistemáticamente, porque nos aguantamos desde que tenemos memoria y porque siempre tienes 5 minutos para lo que sea cuando estás en Barcelona.

La lista continua con las petardas con las que llevo desde los 15 años acumulando momentos. Gracias a Laura (la doctora, doctora, compañera de biblioteca y frustraciones de tesis), Raquel, Hannah, Celia, Sara, Clara y en especial a Amanda, que es como una hermana y ha sido un apoyo incondicional, en todos estos años.

Y ya por último no me quiero olvidar de las golfas de Madrid a Carla y a Elena (¡y a su Frufro!), que llevan dando guerra casi 2 décadas y a pesar de la distancia siempre están en todo y para todo.

A mi familia, a los Córdoba por ser tan, tan, taaaaaaan ellos (nótese el tono de voz elevado y la vena del cuello a punto de explotar) y a los Pañella,

‘perque sense les seves Nadales i sopars burgesos’ (¡que no se note la ironía por favor!) esta tesis no hubiese sido posible.

A Estrella y Alfonso, que también me han aguantado durante todo este tiempo y alimentado y cuidado, ¡simplemente gracias!

Y bueno el final se va acercando ya, pero no por eso deja de ser menos importante. Sin duda alguna no hay día desde hace 15 años que no me acuerde de ti, papá, no sabes como me hubiera gustado compartir contigo todo esto. Y como no a mi madre, mi apoyo incondicional desde siempre. Siempre seremos un *pack de 3*.

Al Víctor, el meu millor amic, l’amor de la meva vida. Encara no entenc com em vas continuar parlant després d’aquella primera trobada dotze anys després al Marenostrum, jo hagués fugit per potes! Gràcies per estimar-me com ho fas. Y como no, al mejor regalo que me ha dado la vida y que me llevo con el final de la tesis, el pequeñín de la casa, el otro amor de mi vida, Ángel. Ahora sí, lo hemos conseguido. Ho hem aconseguit.

Abstract

This thesis aims at improving the convergence of iterative solvers, used for algebraic systems coming from the discretization of partial differential equations (PDE), in the context of large scale simulations and high performance computing (HPC). The methodology followed consists in adapting some existing preconditiong techniques to the physics and numerics of convection-dominated transport and boundary layer problems in flows.

For convection-dominated flows, a physics-based permutation algorithm is presented, which consists in renumbering the mesh in the direction of convection. This renumbering is then used together with a Gauss-Seidel preconditioner to propagate the result of the matrix-vector products along the convection. The robutsness and effectiveness of this preconditioner is proved in several test cases solving the heat equation as well as the Navier-Stokes equations in both sequential and in parallel using the Message Passing Interface library MPI.

Additionally, the composition of preconditioners is proposed to solve cases where different local physical behaviors co-exist in the same flow. In particular, we focus on such problems where of a highly convective flow encounters an obstacle. Such problems involve a zone with high convection far from the obstacle and the development of a boundary layer in the vicinity of

the obstacle. In numerical terms, these local behaviors translate into specific matrix structures that we will take advantage of to adapt the preconditioner locally. On the one hand, the linelet preconditioner is a well-known efficient preconditioner for boundary layers where the mesh is highly anisotropic, in particular to solve the Poisson equation. On the other hand, the streamline linelet that we propose in this thesis (Gauss-Seidel together with a mesh renumbering in the convection direction) is well adapted for locally hyperbolic flows. Both preconditioners will be composed (combined) in different ways to investigate their robustness in terms of convergence as well as their costs to solve the proposed transport problems. We will study as well their performances in terms of parallelization.

Contents

Chapter 1

Introduction

'Begin at the beginning', the King said very gravely 'and go on till you come to the end: then stop.'

Lewis Carroll,
Alice in Wonderland

Improving convergence of the iterative solvers coming from the discretization of complex transport problems, in the context of large scale simulations and high performance computing (HPC), is a challenging topic. Efficiency is not only linked to a good implementation of the numerical method used to solve the problem, but to other constraints such as the physics of the problem that is solved.

The main objectives of this thesis are: First, to analyze thoroughly the heat equation and the physical phenomena associated and how this nature affects the numerics. Second to improve some of the existent preconditioning techniques adapting them to the nature of the physics and improve the convergence of the iterative solvers. Finally, to study these new preconditioning techniques in challenging complex cases using Marenostrum IV supercomputer.

1.1 Motivation

Complex physical problems for both, applied fields and basic research, such as fluid dynamics, heat transfer problems, solid dynamics or general transport equations, are often represented by partial differential equations (PDEs) which have to be discretized and solved numerically. This takes the continuum formulations of physics to systems of algebraic equations, and in order to obtain good approximations to such complex problems it is necessary to solve the discretized problem with a great number of unknowns. The resulting matrices obtained from these discretizations are often very sparse, that is, only a few entries of the matrix differ from zero. Sparse linear systems of equations can be solved either with direct or with iterative methods. Iterative solvers are often the ones preferred, as they are cheaper in terms of computer storage and CPU-time, but at the same time they are less robust than direct methods and often converge slowly to the desired solution. To cope with this problem, equivalent preconditioned systems can be solved instead of the original ones.

Finding a good preconditioner for solving sparse linear systems of equations is not an easy task and several aspects have to be taken into account. The values and distribution of the sparse matrix coefficients highly depend on the physics of the problem. Depending on the problem at hand, different patterns or dependencies (stucture) can be observed in the matrix. Adapting the preconditioner to the physics of the problem and detecting the different physical behaviours that a problem can have, by looking at the values of the coefficients of the sparse matrix, can improve convergence in many multiphysics problems.

1.2 Physics and Numerics in Computational Simulations

Computation is now regarded as an equal partner along with theoretical and experimental sciences in the advance of scientific knowledge and engineering. Numerical simulation enables the study of natural phenomena and other complex systems that would be difficult, expensive or even impossible to study by direct experimentation. The search for having high levels of detail in such simulations, requires enormous computational capacity and has been responsible for mathematicians, physicists and engineers to rack their brains in new breakthroughs in computer algorithms and architectures.

At software level, to build an efficient algorithm, there are many factors that have to be taken into account. First, the nature of the problem that one wants to study. For example, the complexities to study a laminar flow or a turbulent flow are different in terms of degrees of freedom; the study of convective heat transfer where the process of conduction is dominant requires different numerical techniques than the ones required to solve convective heat transfer with a dominant advection. These physical properties will also have an impact on the matrix properties of the discretized problem. This is, depending on the case that one wants to study, matrices can be symmetric, non-symmetric, dense, sparse etc., and then, based on these properties, different numerical methods can be selected to solve the algebraic system in an accurate way. Also, the discretization scheme used plays an important role in solving a problem, for example in terms of numerical method (finite differences, finite volumes or finite elements) or in terms of stabilization techniques. Finally, the way in which the mesh is numbered can affect the convergence and efficiency of the iterative solver used to solve

the corresponding algebraic system. In all these steps, one must also consider the fact that these problems are often very complex and that often a high performance computing environment is necessary to solve them, so these algorithms have to be also envisaged and implemented to eventually run in parallel efficiently.

1.2.1 Physics in Mathematical and Numerical Modeling

In general, the laws of physics define the rules for the dynamics of systems, such as heat transfer, flow motion or electromagnetic radiation. A possible approach to have a good comprehension of the behavior of the system is by looking at the solution of PDEs that describe such systems under different circumstances. PDEs describe the change of a system in space and time, where space coordinates and time are the so-called independent variables.

Mathematically, these PDEs express the conservation of certain physical quantities such as momentum, mass, energy, etc. When envisaging the solution of such PDEs, identifying and understanding physical phenomena beforehand enables to have a better understanding of the problem and also to have a first guess on the dynamics of the system under consideration.

1.2.2 Numerical methods in large scale simulations

Depending on the matrix obtained after a particular problem is discretized with an appropriate scheme [73, 79], different numerical approaches may be used to solve the resultant linear system of equations. For instance, for large sparse matrices, direct solvers such as LU decomposition lose efficiency since the arithmetical operations and memory requirements increase rapidly with the problem size and are thus far from being optimal from a computational point of view. In these cases, an iterative method

such as the conjugate gradient if the matrix is symmetric, or the GMRES and BiCGSTAB methods if the matrix is non-symmetric may be preferred [79]. Also, in case of opting for an iterative method, it is also important to choose a good preconditioning technique to accelerate or even to allow the convergence of the iterative solvers [36], as the matrices coming from the discretizations of PDEs are likeley to be ill-conditioned.

This thesis focuses on the developments of local preconditioning techniques, based on the physics and numerics, to enhance the robustness and performance of the iterative solvers. Deciding which numerical method fits better in each situation and which is the best way to have a fast and robust solver, requires a good understanding of the matrix properties as well as a good knowledge of the existing literature.

1.3 High Performance Computing Environment

This thesis has been developed in the Computer Applications of Science & Engineering (CASE) department of the Barcelona Supercomputing Center (BSC-CNS). The aim of this department is to develop numerical software to solve complex physical problems by means of HPC. In this context, the use of supercomputers becomes essential as ordinary computers do not have enough computational power to deal with large-scale cases. Many codes have been developed to tackle complex problems on supercomputers, such as Code_Saturne, FLASH or Alya [2, 20, 96], which is the one that has been used in this thesis.

1.3.1 Marenostrum IV Supercomputer

A supercomputer is composed of thousands of processors working in parallel. The computational performance of a computer is measured in floating point operations per second (flops) and to get an idea of the power, the performance of a supercomputer is measured in Petaflops as opposed to Gigaflops for a laptop, for instance.

In the last years, supercomputers have been used to model scenarios for tackling challenges that would be impossible otherwise. They change how meteorologists forecast the weather, how astrophysicists study the evolution of the universe or how engineers simulate aircrafts. There are many supercomputing centers world wide which are in charge of providing resources to enable such simulations. In the particular case of this thesis, the numerical experiments have been carried out on Marenostrum IV supercomputer. Marenostrum IV is the last version of the Marenostrum series which started with Marenostrum I in March 2004. MareNostrum IV is based on Intel Xeon Platinum processors. It consists of 48 racks housing 3456 nodes with a total of 165,888 processor cores. Compute nodes are equipped with 2 sockets Intel Xeon Platinum 8160 CPU with 24 cores each, for a total of 48 cores per node. Its current Linpack Rmax Performance is 6.23 Petaflops.

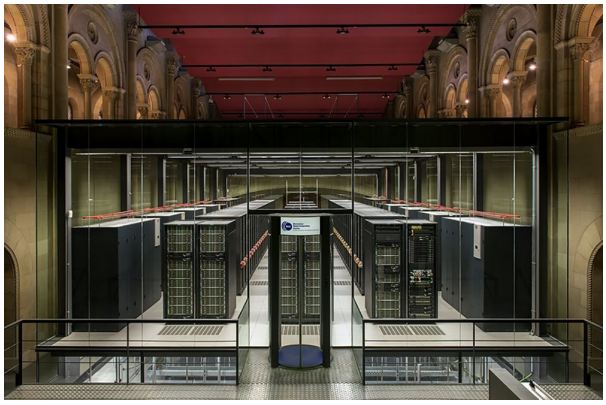


Figure 1.1: *Marenostrum IV supercomputer.*

1.3.2 Alya multi-physics code

Alya is a high-performance computational mechanics code developed at Barcelona Supercomputing Center. This code is designed for multiphysics and to be flexible to run parallel coupled problems using techniques for distributed (Message Passing Interface library, MPI) and shared memory (OpenMP and OmpSs) parallelization, together with vectorization to enhance performance at node level. The code uses mainly the finite element method for space discretization and was written from scratch to be run in parallel.

Alya has a modular architecture (see Figure 1.2). It consists of a kernel and some modules. The kernel contains the common tasks in which all the numerical problems solved in Alya need. It is responsible for reading data, performing operations and calling the different modules and services needed in each task. The modules solve each of the physical problems individually.

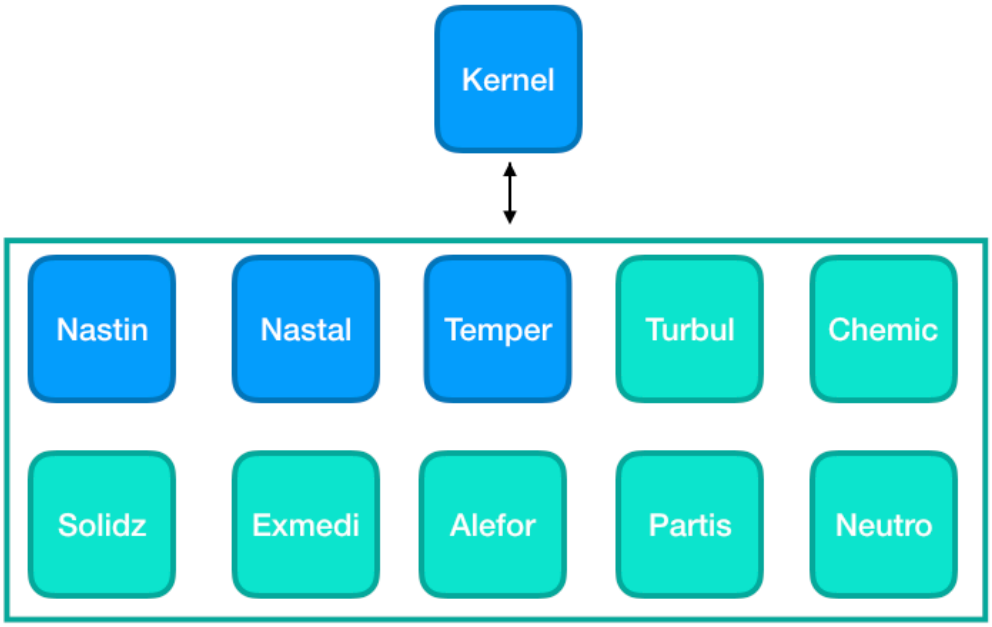


Figure 1.2: *Alya's modular structure.*

In the particular case of this thesis, all the code has been developed in the kernel, in particular in the algebraic solver subroutines. Its has been tested to solve the heat equation (Temper module) and also to solve the incompressible and compressible Navier-Stokes equations (Nastin and Nastal modules, respectively).

1.4 Objectives

A good example that involves complex physical phenomena is convection heat transfer in a fluid, which involves fluid motion as well as heat transfer. Generally, the problems governed by such phenomena are associated with complex geometries including obstacles, zones of high advection and zones of low advection, thus exhibiting different local behaviors. In this particular case, two main physics can be distinguished. On the one hand, there is a high advection away from the boundary layer developing near the obstacles; on the other hand, in the boundary layer attached to the obstacle, the problem is locally diffusion-dominant. The heat transport in such flows is modeled using a PDE, namely the advection-diffusion equation. This equation needs to be discretized to obtain a finite dimension solution to the problem. As it was said in the Motivation, solving the corresponding linear system arising from this discretization of the PDE is often complex and in some cases also difficult to converge. If an iterative solver is used, often preconditioning techniques are used to improve the convergence. The aim of this thesis is to adapt some existing preconditioning techniques to the physics of the problem, and also to derive different preconditioners to further enhance the convergence. For example, in the case of the presence of a boundary layer, there are two local phenomena, as stated above: strong advection and high diffusion. So in this case, one preconditioner would be used to solve the

strong advection and the other the high diffusion parts. This is what is called local preconditioning (a clear picture of this can be seen in Figure 1.3). To achieve these goals, the following objectives have been accomplished:

- Analyze the connection between the physics and numerical methods in some limiting situations found in the convection-diffusion equation.
- Study the impact of the numerical method on the algebraic system.
- Devise the different state-of-the art preconditioning techniques based on these analyses.
- Study the condition number of the preconditioned system and estimate the gain for simple problems.
- Validate the hypotheses using complex examples.
- Understand the effect of parallelization on the convergence of such preconditioners.

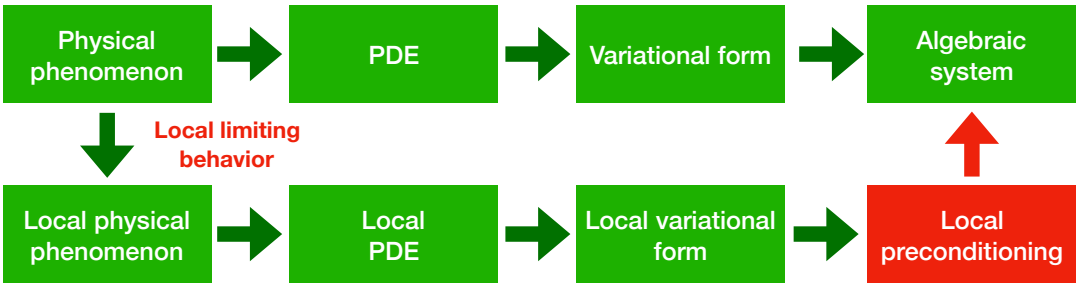


Figure 1.3: *Objectives general scheme.*

1.5 Thesis Outline

Apart from this introduction, the thesis is organized in six additional chapters:

- Chapter two introduces a finite element method to solve the advection-diffusion equation focusing on its limiting behaviors. This includes a full description of the advection-diffusion equation, a description of the variational formulation together with the SUPG stabilization method and finally, we introduce some case studies where we bring some theoretical backgrounds to the simulations that we will study along the thesis.
- Chapter three consists of a state-of-the-art of the different methods, either direct or iterative, to solve sparse linear systems. We will focus mainly on the GMRES, BiCGSTAB and conjugate gradient methods, as these will be the ones used in the different simulations carried in this thesis.
- Chapter four starts with a state-of-the-art of the existing preconditioning techniques and finally explains in detail the mesh renumbering proposed to be used together with the Gauss-Seidel method to precondition systems coming from convection-dominant problems.
- Chapter five focuses on the composition of preconditioners.
- Chapter six proves the results obtained in Chapters four and five, but for the compressible Euler and the incompressible Navier-Stokes equations.
- Chapter seven holds the conclusions of this PhD thesis and shows possible future work.

Chapter 2

Physical and Numerical Phenomena in Numerical Simulations

What is essential,
is invisible to the eye

Antoine de Saint-Exupéry,
The Little Prince

This chapter presents the finite element method used in this thesis to model the advection-diffusion (AD) equation. In the first section we explain briefly the advection-diffusion equation together with the characteristics behavior encountered under certain conditions such as high/low Péclet numbers or under some specific conditions such as boundary layers. Then the variational formulation of the AD equation is introduced and some of the stabilization techniques found in literature are explained. Finally the second section covers several examples found in the physical and numerical modeling of the AD equation. In particular we will study the cases of convection-dominated problems and anisotropic meshes.

2.1 The Advection-Diffusion Equation

This section describes a transport equation known as the AD equation. First we will briefly describe the AD equation, then we will put it in its non-dimensional form to study its applicability range and the boundary layer phenomenon. After, we will state the properties of the work spaces and finally we will derive the weak form of the AD equation together with the streamline upwind Petrov-Galerkin (SUPG) stabilization strategy [17].

2.1.1 Description of the AD Equation

The advection-diffusion equation is a transport equation that models many physical phenomena like heat transfer in a fluid or the transport of species through two transport mechanisms: advection (convection) and diffusion. As a model equation in this section the heat equation will be considered in order to establish analogies to illustrate mathematical behaviors:

$$Lu := \overbrace{\underbrace{\rho c_p \partial_t u}_{Hyperbolic} + \rho c_p \mathbf{a}\cdot\nabla u}^{Parabolic} \overbrace{\underbrace{-k\Delta u = F}_{Hyperbolic}}^{Parabolic} \quad \text{in } \Omega \tag{2.1}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{Elliptic}$$

where, u is temperature, Ω with boundary $\partial\Omega$, is a n-dimensional domain, k is the thermal conductivity, ρ is the density of the medium, c_p the specific heat, F the heat source term, t the time, and $\mathbf{a}$ the advection field. This equation should be supplied with specific initial and boundary $\partial\Omega$ conditions. This can be re-written as:

$$\frac{\partial u}{\partial t} + \mathbf{a}\cdot\nabla u - \alpha\Delta u = f \quad \text{where} \quad \alpha = \frac{k}{\rho c_p} \quad \text{and} \quad f = \frac{F}{\rho c_p} \tag{2.2}$$

where α is the thermal diffusivity.

As remarked, Equation (2.1) can admit different characteristics behaviors. These are elliptic parabolic and hyperbolic. For example a pure diffusive problem would be elliptic and a pure convective problem would be hyperbolic. In a hyperbolic PDE, a perturbation of the solution will follow the direction of convection. In an elliptic situation, for example, in the case of pure diffusion, the quantity will diffuse isotropically and the domain of influence eventually will be the complete domain Ω. The implications at mathematical and numerical level will be shown in the next sections of this chapter.

Also, according to the characteristics of the PDE's, different types of initial and boundary conditions will be needed to solve the problem in each particular case. To simplify, only three main types of boundary conditions depending on the coordinates $\mathbf{x}$ and time t will be considered. These are:

- **Dirichlet:** Prescription of temperature $u = f_d(\mathbf{x}, t)$.
- **Neumann:** Prescription of the heat flux $f_n(\mathbf{x}, t)$ and it is given by:

$$k\frac{\partial u}{\partial n} = k\nabla u \cdot \mathbf{n} = f_n(\mathbf{x}, t)$$

  where $\mathbf{n}$ is the outward unit vector to the boundary.

- **Robin:** These are weighted combinations of Dirichlet and Neumann boundary conditions and are defined by:

$$k\frac{\partial u}{\partial n} + \alpha_r u = f_r(\mathbf{x}, t)$$

  with α_r being the Robin factor.

Depending on the characteristic of the PDE, different combination of the boundaries may be applied or prohibited to get a well posed problem [89]. For example, for stationary hyperbolic flows a Dirichlet boundary condition must be imposed at inflow [76].

2.1.2 Asymptotic vs Characteristic Behaviors

In the following, when we refer to asymptotic and characteristic behaviors it will be considered purely elliptic, parabolic or hyperbolic cases. Studying the asymptotic and characteristic behaviors of the AD equation together with some of the physical phenomena that occur in determined situations, helps to predict the behaviour of the AD equation before it is solved numerically. In this section the non-dimensional form of the AD equation will be written, to study different regimes, and also the consequences of the boundary layer phenomenon. This study will then be used in Section 4.3 to derive suitable preconditioning techniques.

The asymptotic behaviour of the AD equation is better understood having the AD equation in a non-dimensional form. In this way taking (2.2), the following non-dimensional variables can be defined:

$$t^* = \frac{t}{t_0} \quad \mathbf{x}^* = \frac{\mathbf{x}}{x_0} \quad \mathbf{a}^* = \frac{\mathbf{a}}{a_0} \quad u^* = \frac{u}{u_0} \quad f^* = \frac{f}{f_0}$$

where t_0, x_0, a_0, u_0 and f_0 are the characteristic time, length, advection velocity, physical quantity studied (temperature in this particular case) and source term and t^*, $\mathbf{x}^*$, $\mathbf{a}^*$, u^*, f^* are the non-dimensional variables of the time, length, advection velocity, the physical quantity studied and the source term. Such characteristic values will enable to compare the different terms of the equations and are problem-dependent.

Then substituting in (2.2) the following is obtained:

$$\frac{a_0}{x_0}\frac{\partial u^*}{\partial t^*} + \frac{a_0 \mathbf{a}^*}{x_0} \cdot \nabla u^* - \frac{\alpha}{x_0^2}\Delta u^* = \frac{f_0 f^*}{u_0}$$

where in this case we have chosen $t_0 = \frac{x_0}{a_0}$.

$$\frac{\partial u^*}{\partial t^*} + \mathbf{a}^* \cdot \nabla u^* - \frac{1}{\text{Pe}}\Delta u^* = f' \tag{2.3}$$

$$\text{with } \text{Pe} = \frac{a_0 x_0}{\alpha} = \frac{a_0 x_0 \rho c_p}{k} \text{ and } f' = \frac{f_0 x_0 f^*}{a_0 u_0}$$

The non-dimensional number Pe is the Péclet number and gives the ratio of advection to the rate of diffusion, this is, in the particular case shown in Equation (2.3) for heat transfer, it measures the ratio between the heat transferred by convection to the diffusive heat transfer.

With all these, some questions arise: How does (2.3) behaves under high/low Péclet numbers? How these asymptotic and characteristic behaviours will affect the solution of the AD equation?

Asymptotic Behavior

Equations (2.4) and (2.5) show the behaviour of the non-dimensional AD equation under high and low Péclet numbers. As we can see with high Péclet numbers Equation (2.3) tends to an hyperbolic behaviour whereas under low Péclet numbers it has a parabolic behavior if the problem is time-dependent or elliptic behaviour if the problem is stationary. Then considering the non-dimensional form of the AD equation, the asymptotic behavior is given by:

$$\text{Pe} \to \infty \text{ then equation 2.3 tends to } \frac{\partial u^*}{\partial t^*} + \mathbf{a}^* \cdot \nabla u^* = f' \tag{2.4}$$

$$\text{Pe} \to 0 \text{ then equation 2.3 tends to } \frac{\partial u^*}{\partial t^*} - \frac{1}{\text{Pe}}\Delta u^* = f' \tag{2.5}$$

It is important to stress that equation that Equation (2.4) is only valid away from the boundaries. In fact, in the vicinity of walls, where the temperature is prescribed, diffusion takes over convection and the correct equation is the original one (2.3). This is what happens with boundary layer as we will show in section 2.1.3.

Characteristic Behavior

We will refer to characteristic behavior to pure elliptic, hyperbolic and parabolic cases. Considering Equation 2.3, these are:

- Pure elliptic:

$$-\frac{1}{\text{Pe}}\Delta u^* = f' \tag{2.6}$$

- Pure hyperbolic:

$$\frac{\partial u^*}{\partial t^*} + \mathbf{a}^* \cdot \nabla u^* = f' \tag{2.7}$$

- Pure parabolic:

$$\frac{\partial u^*}{\partial t^*} - \frac{1}{\text{Pe}}\Delta u^* = f' \tag{2.8}$$

The main difference between asymptotic and characteristic behaviors stems from the appropriate boundary conditions and the locality of the phenomenon in the case of asymptotic behavior.

From now on in this thesis u, t, $\mathbf{x}$, $\mathbf{a}$ are going to be considered non-dimensional variables and f' will be called f.

2.1.3 Boundary Layer Phenomena

In a flow, boundary layers are some restricted regions where sharp gradients of the solution (velocity or temperature) exist. Boundary layers occur when an incoming flow encounters an obstacle where the local conditions are different from the free stream conditions. They can be isothermal, thus affecting exclusively the velocity profile, or thermal, affecting both the velocity and temperature.

The main characteristic of a boundary layer is height, which typically depends on dimensionless numbers like the Reynolds number for the velocity flow and the Prandtl number for the temperature. The height of the boundary layer is by definition the extent through which the unknown goes from its value on the obstacle to its free stream value. As an example the height of the boundary layer decreases when the Reynolds number increases, leading to sharper gradients when convection dominates over diffusion.

In nature, we encounter two main types of boundary layers. On the one hand, parabolic boundary layers occur when a flow encounters an obstacle which walls are parallel to it. On the other hand, exponential boundary layers occur when the flow impinges an obstacle.

Let us define our boundary $\partial\Omega$ in three parts as:

$$\partial\Omega = \partial\Omega^{-} \cup \partial\Omega^{0} \cup \partial\Omega^{-}$$

where $\partial\Omega^-$, $\partial\Omega^0$ and $\partial\Omega^-$ are the inflow, wall and outflow boundaries respectively which are defined as:

$$\partial\Omega^- = \{\mathbf{u} \in \partial\Omega \text{ where } \mathbf{a}\cdot\mathbf{n} < 0\}$$
$$\partial\Omega^0 = \{\mathbf{u} \in \partial\Omega \text{ where } \mathbf{a} = 0\}$$
$$\partial\Omega^+ = \{\mathbf{u} \in \partial\Omega \text{ where } \mathbf{a}\cdot\mathbf{n} > 0\}$$

According to the place in which this sudden changes of velocity and temperature take place, different types of boundary layers may occur. So for example, if the changes of velocity and temperature occur in the inflow boundary ($\partial\Omega^-$) or along the characteristic boundary ($\partial\Omega^0$), then a parabolic boundary layer may arise, whereas if the same phenomena occurs in the outflow boundary ($\partial\Omega^+$), then an exponential boundary layer will happen.

Knowing this, the parabolic and exponential boundary layers will be explained in more details.

Parabolic Boundary Layer

Let us consider a flat plate at temperature T_w immersed in a fluid with free stream velocity V_∞ and temperature T_∞ when the flow hits the wall, velocity and temperature must accommodate to the plate boundary condition. In this situation, a boundary layer triggers and propagates along the plate as illustrated by Figure 2.1. Mathematically, the asymptotic behavior can be obtained through dimensional analysis [3, 10]:

$$\frac{\partial u}{\partial t} + a_x\frac{\partial u}{\partial x} + a_y\frac{\partial u}{\partial y} - \frac{1}{\mathrm{Pe}}\left(\cancel{\frac{\partial^2 u}{\partial x^2}} + \frac{\partial^2 u}{\partial y^2}\right) = f$$

$$\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} + a_y \frac{\partial u}{\partial y} - \frac{1}{\text{Pe}} \frac{\partial^2 u}{\partial y^2} = f$$

$$a_x \frac{\partial u}{\partial x} + a_y \frac{\partial u}{\partial y} \ll \frac{1}{\text{Pe}} \frac{\partial^2 u}{\partial y^2} \Rightarrow \boxed{\frac{\partial u}{\partial t} - \frac{1}{\text{Pe}} \frac{\partial^2 u}{\partial y^2} = f} \tag{2.9}$$

Equation (2.9) shows the thermal boundary layer equation. In this equation several approximations have been considered, first it has been assumed that the gradients across the boundary layer are larger than the ones across the main flow direction. Also it has been assumed that the direction of the flow is negligible if compared to the gradient across the boundary [3].

The approximation for the equations of the velocity boundary layer are similar to the thermal boundary layer ones but using Navier-Stokes equations instead. This thesis will not go into detail into Navier-Stokes equations or the velocity boundary layer, but a full demonstration can be found in the literature [3, 10].

Apart from the approximation of the equations it is also possible to obtain the thickness of both boundary layers, these are δ and δ_T for velocity and temperature respectively, which are defined as:

$$\delta \sim \sqrt{\frac{\nu L}{V_\infty}} \tag{2.10}$$

$$\delta_T \sim \delta \, \text{Pr}^{-1/3} \tag{2.11}$$

where Pr is the Prandtl number, which is defined to be the ratio between momentum diffusivity and thermal diffusivity:

$$\mathrm{Pr} = \frac{\nu}{\alpha} = \frac{\mu/\rho}{k/(c_p\rho)} = \frac{c_p\mu}{k}$$

with ν being the kinematic viscosity, which is also $\nu = \mu/\rho$ where μ is the dynamic viscosity [3, 10].

As it is shown in Figure 2.1 if the momentum diffusivity is greater than the thermal diffusion rate, the Prandtl number will be greater than one and thus the thickness for the velocity boundary layer δ will be greater than the thickness for the thermal boundary layer δ_T. The opposite will happen when the thermal diffusion rate dominates over the momentum diffusivity. Finally if the Prandtl number equals to one, both boundary layers will have the same thickness.

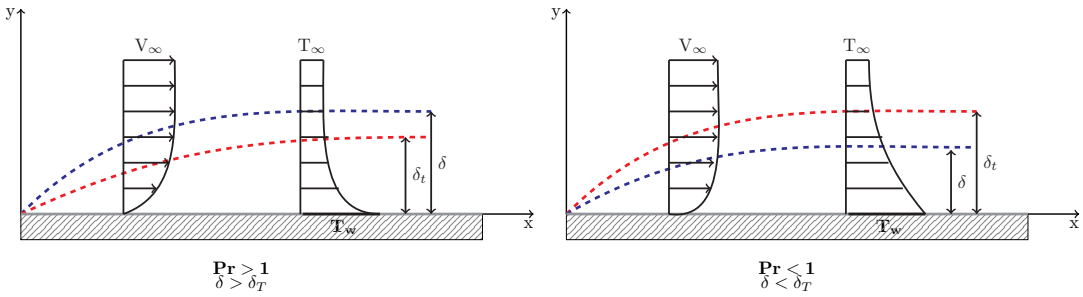


Figure 2.1: *Parabolic boundary layer in a flat plate. Right: The boundary layers from the cooling of a body with* $Pr < 1$. *Left: The boundary layers from the cooling of a body with* $Pr > 1$.

Exponential Boundary Layer

Impinging jets occurs when a flow impinges a surface, generally at high velocities (high Reynolds number). Such jets can be intentionally sought for convective heat transfer in many engineering applications such as cooling of turbine blades or cooling of hot steel plates. Figure 2.2 shows a general scheme of an impinging jet. An impinging jet can be divided into several regions. First we find the free jet region, where velocity remains

constant and equal to the nozzle exit (of width d) until the jet spreads (impinging region). Then, we find the stagnation region, in which the jet is deflected from the axial direction. Finally in the two sides we find the wall jet region, where the flow increases in thickness and boundary layer builds along the surface.

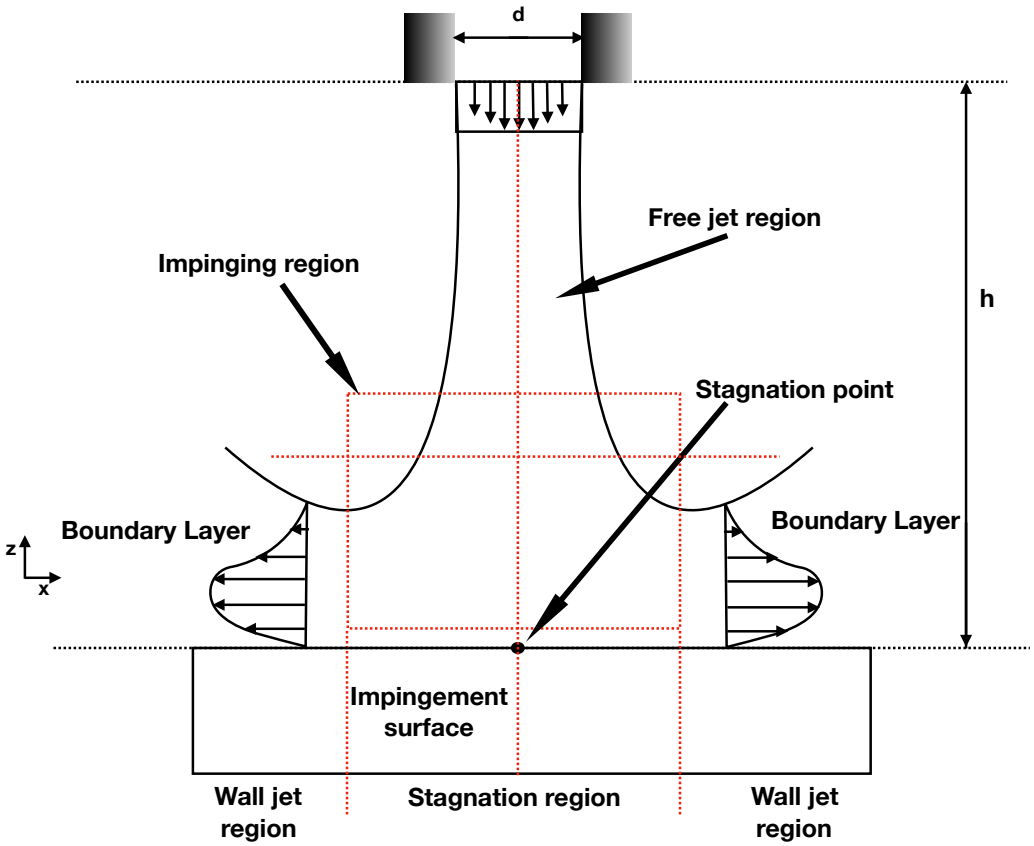


Figure 2.2: *Schematic of an impinging jet inspired in the work of [71, 77].*

As we stated in the introduction of this section, exponential boundary layers can be expressed mathematically when the temperature is prescribed at an outflow. This is situation is unlikely found in physical problems, but a similar behavior happens with impinging jets [65]. In fact, when the jet impinges, a zero velocity is enforced by the no-slip condition. For this reason, we will show two examples. We start with a 1D example, for which we find the exact solution, and then show the variation of the boundary

layer according to the Péclet number; then we calculate the numerical solution of an impinging jet for a 2D example. Through the comparison of a 2D impinging jet with the results of a 1D exponential boundary layer (see Figures 2.5 and 2.4), apart from the space dimension, the main difference stems from the boundary condition of the velocity which is a no-slip condition in the former case and an outflow condition in the latter.

Let us consider the following 1D example:

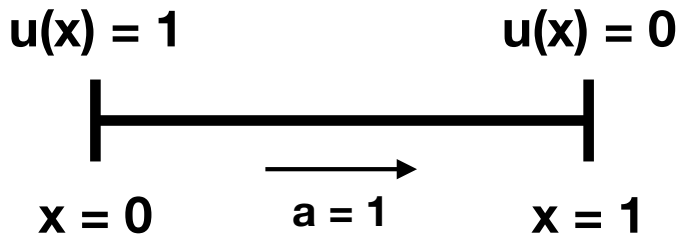


Figure 2.3: *Domain of length 1 in which the AD equation is solved.*

$$\begin{cases} -\dfrac{1}{\mathrm{Pe}}\dfrac{d^2u}{dx^2} + \dfrac{du}{dx} = 0 \quad \forall x \in \Omega = (0,1) \\ u(0) = 1 \\ u(1) = 0 \end{cases} \tag{2.12}$$

For the sake of simplicity, the advection velocity a and the source term f will be chosen as $\mathrm{a} = 1$ and $f = 0$. Then, it is easy to see that (2.12) has the following exact solution:

$$u(x) = 1 - \frac{e^{\mathrm{Pe}x} - 1}{e^{\mathrm{Pe}} - 1} \tag{2.13}$$

And plotting the solution for different Péclet numbers we obtain the profiles of Figure 2.4.

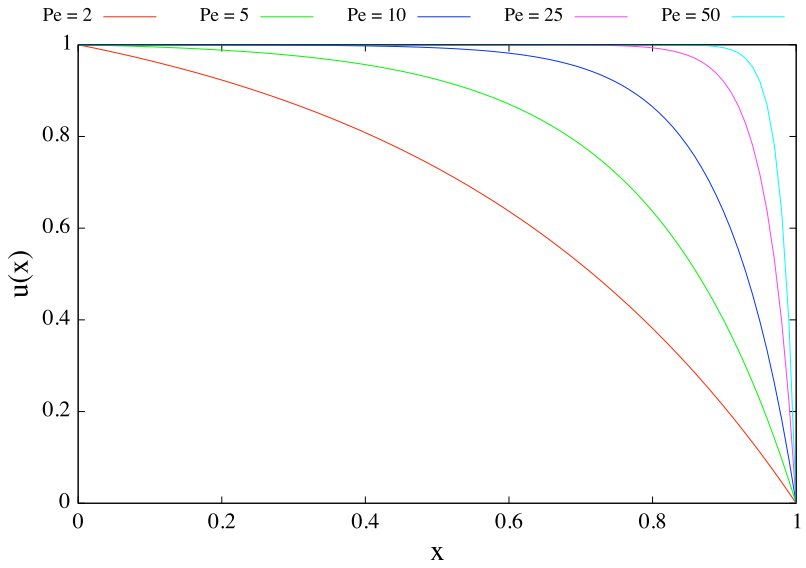


Figure 2.4: *Exponential boundary layer of the 1D advection-diffusion equation for different Pe.*

As it is shown in Figure 2.4, the width of the boundary layer increases as the Péclet number decreases, in this cases outflow boundary conditions need to be prescribed in order to guarantee the solution of the PDE.

Now let us consider the 2D example sketched in Figure 2.5. In a first step, the incompressible Navier-Stokes equations are solved in the computational domain shown in the figure. Constant inflow velocity is imposed at the inlet, a no-slip condition is imposed on the walls, and a symmetry condition on the left-hand boundary. The Reynolds number was set to 100, where the characteristic length is the inlet width. The left part of the figure shows the velocity **a** magnitude obtained. Then we solve the temperature equation, with a high Péclet number of 500. The temperature is prescribed at the inlet to a value of one, and to zero on the upper wall of the outflow channel. Elsewhere, a zero flux condition is imposed. The temperature magnitude contours are shown on the right part of the figure, where we can observe the parabolic boundary layer developing on the upper part of the outflow channel. Finally, the plot

compares the solution obtained with the 1D exact solution together with the temperature along a vertical cut in the inflow channel. We observe a very similar behavior, showing that the solution of an impinging jet is triggering an exponential boundary layer.

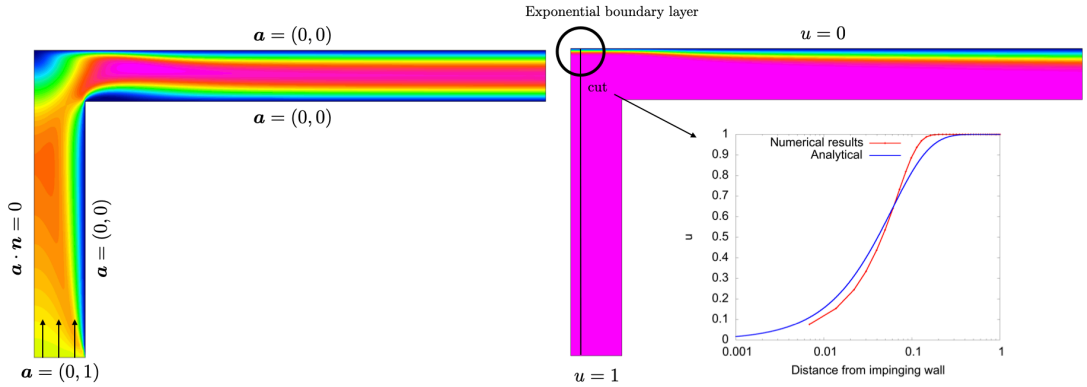


Figure 2.5: *Exponential boundary layer in an impinging jet. On the left, velocity contours. On the right, temperature contours and comparison between the 1D exact solution and the temperature obtained along a vertical cut (shown in the entrance channel).*

2.2 Numerical Treatment of the AD Equation

In general, partial differential equations such as Equation (2.3) only have an exact solution under certain conditions of some of its parameters. But in most cases and more when solving general problems, numerical methods are required to solve them. There are several techniques to solve a partial differential equation numerically, although the most common one consists in the discretization of partial differential equations using techniques such as finite differences, finite elements and finite volumes. This thesis will only focus on the finite element method, as it is the only one that has been used to carry the different simulations that will follow in the next chapters.

In this section the finite element formulation of the advection-diffusion

equation will be introduced. To do so, a continuous variational formulation will be introduced (weak formulation), after the same will be done but for a finite dimensional subspace, then a basis for the variational formulation in the finite dimensional subspace will be chosen to obtain a system of linear equations.

2.2.1 Weak Formulation

Before starting with the finite element spatial discretization of the AD equation, a weak formulation needs to be defined for the Equations in (2.1) and (2.3). First the boundary $\partial\Omega$ will be split into two components, these are, Γ_D and Γ_N. Γ_D is in general associated with the inflow $\partial\Omega^-$ and/or the wall $\partial\Omega^0$, while Γ_N is related to the outflow $\partial\Omega^+$. Then, to define the weak or variational formulation two types of spaces are needed, these are, natural spaces for second order PDE's and are defined as:

- *Test Spaces:* These will be denoted by V and are all the square integrable functions that have square integrable first derivatives in the domain Ω and that are zero on the Dirichlet boundary Γ_D.

$$V = \{v \in H^1(\Omega) \mid v = 0 \text{ on } \Gamma_D\} \equiv H^1_{\Gamma_D}(\Omega) \tag{2.14}$$

where $H^1(\Omega)$ is a Hilbert space defined by:

$$H^1(\Omega) := \{v \in L^2(\Omega) \mid \frac{\partial v}{\partial x_j} \in L^2(\Omega), j = 1, ..., n_d\} \tag{2.15}$$

with $L^2(\Omega)$ being the space of square integrable functions in Ω and n_d the space dimension. Also, we provide $H^1(\Omega)$ with the following scalar product:

$$(u,v)_1 := \int_\Omega uvd\Omega + \int_\Omega \nabla u \cdot \nabla v d\Omega$$

and we define the norms associated to $H^1(\Omega)$ and $L^2(\Omega)$ respectively as:

$$\|u\|_1 = \left(\int_\Omega u^2 d\Omega + \int_\Omega \nabla u \cdot \nabla u d\Omega\right)^{1/2}$$

$$\|u\| = \left(\int_\Omega u^2 d\Omega\right)^{1/2}$$

- *Trial Spaces:* These will be denoted by S. They are similar to test functions but with the difference that the functions need to satisfy the Dirichlet conditions at the boundary Γ_D.

$$S = \{u \in H^1(\Omega) \mid u = u_D \text{ on } \Gamma_D\} \equiv V + \{\bar{u}_D\} \tag{2.16}$$

where $\bar{u}_D$ is any function of $H^1(\Omega)$ which satisfies $\bar{u}_D = u_D$.

From the aforementioned definitions, it is easy to see that in case of having only homogeneous boundary conditions, this is $u_D = 0$, test, trial and Hilbert spaces will be the same and will be identified as V^0, S^0 and $H_0^1(\Omega)$ respectively and can be written as:

$$H_0^1(\Omega) := \{v \in H^1(\Omega) \mid v_{\partial\Omega} = 0\} \tag{2.17}$$

These will have the same scalar product and norm as H^1.

Then, the weak form of Equations (2.1) and (2.3) is achieved by multiplying them by a test function belonging to V and integrating over the domain Ω.

Then the variational formulation consists in finding $u \in S$ such that:

$$\int_\Omega v\frac{\partial u}{\partial t}\, d\Omega + \int_\Omega v\mathbf{a}\cdot\nabla u\, d\Omega - \frac{1}{\text{Pe}}\int_\Omega v\Delta u\, d\Omega = \int_\Omega vf\, d\Omega \;\; \forall v \in V \quad (2.18)$$

Applying the Gauss divergence theorem to the left-hand side of equation (2.18) the following is obtained:

$$\begin{aligned}\int_\Omega v\frac{\partial u}{\partial t}\, d\Omega + \int_\Omega v\mathbf{a}\cdot\nabla u\, d\Omega + \frac{1}{\text{Pe}}\int_\Omega \nabla v\cdot\nabla u\, d\Omega = \\ = \int_\Omega vf\, d\Omega + \frac{1}{\text{Pe}}\int_{\partial\Omega} v\nabla u\cdot\mathbf{n}\, d\partial\Omega \;\; \forall v \in V\end{aligned} \quad (2.19)$$

Before introducing the finite element formulation, the following bilinear and linear forms are introduced for the sake of clarity:

$$\begin{aligned}a(u,v) :&= \int_\Omega v\frac{\partial u}{\partial t}\, d\Omega + \int_\Omega v\mathbf{a}\cdot\nabla u\, d\Omega + \frac{1}{\text{Pe}}\int_\Omega \nabla v\cdot\nabla u\, d\Omega \\ f(v) :&= \int_\Omega vf\, d\Omega + \frac{1}{\text{Pe}}\int_{\partial\Omega} v\nabla u\cdot\mathbf{n}\, d\partial\Omega\end{aligned} \quad (2.20)$$

Finally, with these definitions, the weak form of the problem consists in finding $u \in S$ such that:

$$a(u,v) = f(v) \;\; \forall v \in V \quad (2.21)$$

2.2.2 Existence and Uniqueness of Solutions

Let us analyze the existence and uniqueness of solutions of Equation (2.21). To do so, we will consider the non-dimensional form of the advection-diffusion equation defined in Equation (2.3) with $\partial u/\partial t = 0$ and homogeneous boundary conditions;

$$\begin{cases} \mathbf{a} \cdot \nabla u - \dfrac{1}{\mathrm{Pe}} \Delta u = f \text{ in } \Omega \\ \qquad\qquad u = 0 \text{ on } \partial\Omega \end{cases} \tag{2.22}$$

where we have considered $\nabla u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \right)$, $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, $\mathbf{a} = (a_x, a_y)$ and $\nabla \cdot \mathbf{a} = 0$.

Lax-Milgram theorem [31], guarantees the existence and uniqueness of solutions of a non-symmetric bilinear form $a(\cdot, \cdot)$. The theorem states the following:

Theorem 2.2.1. *The variational problem defined in (2.21) has a unique solution $u \in V$ if the following conditions hold:*

(i) *$a(\cdot, \cdot)$ is continuous, this is, it exists a real constant N_a such that:*

$$\left| a(u, v) \right| \leq N_a \|u\|_V \|v\|_V \, \forall u, v \in V$$

(ii) *$a(\cdot, \cdot)$ is coercive, this is, it exists a real positive constant M_a such that:*

$$\left| a(v, v) \right| \geq M_a \|v\|_V^2 \, , \forall v \in V$$

(iii) *$f(\cdot)$ is a continuous linear functional on V, this is it exists a real constant N_l, such that:*

$$f(v) \leq N_l \|v\|_V \, \forall v \in V$$

Let us prove now the existence and uniqueness for this specific case of the AD equation using the problem defined in (2.22). In this case we have chosen homogeneous Dirichlet boundary conditions and no Neumann boundary

conditions, so our solution will be in $u \in V^0$.

Proof.

(i) *Continuity of* $a(\cdot,\cdot)$

From the bilinear form it is easy to see that:

$$a(u,v) \leq \left| \int_\Omega \frac{1}{Pe} \nabla u \cdot \nabla v d\Omega \right| + \left| \int_\Omega v\mathbf{a} \cdot \nabla u d\Omega \right| \tag{2.23}$$

Now, if we seek the bounds of each term separately:

$$\left| \int_\Omega \frac{1}{Pe} \nabla u \cdot \nabla v d\Omega \right| \leq \int_\Omega \left| \frac{1}{Pe} \nabla u \cdot \nabla v \right| d\Omega \leq \frac{1}{Pe} \|\nabla u\| \|\nabla v\| \leq \frac{1}{Pe} \|u\|_1 \|v\|_1$$

$$\left| \int_\Omega v\mathbf{a} \cdot \nabla u d\Omega \right| \leq \int_\Omega |v\mathbf{a} \cdot \nabla u| \, d\Omega \leq \|\mathbf{a}\|_\infty \|u\|_1 \|v\|_1$$

where $\|\mathbf{a}\|_\infty = \sup_i |a_i|$.

Finally putting together these two bounds into Equation (2.23), we get:

$$a(u,v) \leq N_a \|u\|_1 \|v\|_1 \quad \forall v \in V^0 \tag{2.24}$$

where $N_a = \frac{1}{Pe} + \|\mathbf{a}\|_\infty$.

(ii) *Coercivity of* $a(\cdot,\cdot)$

Here we note that $\forall \in v\Omega$,

$$\int_\Omega v\mathbf{a} \cdot \nabla v = \int_\Omega \mathbf{a} \cdot \nabla \left(\frac{v^2}{2} \right) = \int_{\partial\Omega} (\mathbf{a} \cdot \mathbf{n}) \left(\frac{v^2}{2} \right) - \int_\Omega \left(\frac{v^2}{2} \right) \nabla \cdot \mathbf{a}$$

The term $\int_{\partial\Omega}(\mathbf{a}\cdot\mathbf{n})\left(\frac{v^2}{2}\right)$ dissapears, as v vanishes in $\partial\Omega$. Then using that $\nabla\cdot\mathbf{a}=0$ and Poincaré-Friedrichs inequality we obtain:

$$a(v,v)=\frac{1}{\text{Pe}}\int_\Omega\|\nabla v\|^2+\int_\Omega v\mathbf{a}\cdot\nabla v=\frac{1}{\text{Pe}}\int_\Omega|\nabla v|^2\ d\Omega\ \geq M_a\|v\|_1^2 \tag{2.25}$$

where $M_a=\frac{C}{Pe}$ and C is the Poincaré constant [40].

- *Continuity of* $f(\cdot)$

$$f(v)\leq\int|vfd\Omega|\leq\int|vf|\,d\Omega\leq\|v\|\|f\|\leq N_l\|v\|_1$$

where $N_l=\|f\|$.

Since Lax-Milgram theorem holds for the advection-diffusion equation, we can conclude that there exists a unique solution $u\in V^0$ for this variational problem. □

2.2.3 Energy Norm and Stability

Let us now introduce the energy norm $\|\cdot\|_a$ in V, given by:

$$\|u\|_a:=a(u,u) \tag{2.26}$$

In the particular case of the problem defined in (2.22) this can be written as:

$$a(u,u)=\frac{1}{\text{Pe}}\|\nabla u\|^2 \tag{2.27}$$

From Equation (2.25) it is easy to see that:

$$M_a\|u\|_1^2\leq\frac{1}{\text{Pe}}\|\nabla u\|^2 \tag{2.28}$$

Also from this last equation and using Cauchy-Schwarz inequality, we know that:

$$\frac{1}{\text{Pe}}\|\nabla u\|^2 \leq \|f\|\|u\|_1 \tag{2.29}$$

And putting Equations (2.28) and (2.29) together we obtain:

$$\|u\|_1 \leq \frac{\text{Pe}}{C}\|f\| \tag{2.30}$$

where C is the Poincaré constant. This means that, for large Péclet numbers, or what is the same, for small values of the diffusion coefficient k, the right-hand-side of Equation (2.30) is high, meaning that small variations on the data f can lead to large variations on the solution u [15, 40, 41].

2.2.4 Discrete Galerkin Formulation

Let Ω_e be a finite element partition of the domain Ω, where the index e ranges from 1 to the total number of elements n_e and let h be the diameter of Ω_e. From the definition of (2.17) the functional space from the previous partition V_h^0 is considered such that $V_h^0 \subset V^0$, the finite element space under these circumstances is said to be conforming. Finally the finite element approximation or what is the same, the discrete Garlekin formulation of the problem consists in finding $u_h \in V_h^0$ such that:

$$a(u_h, v_h) = f(v_h) \;\; \forall v_h \in V_h^0 \tag{2.31}$$

where $a(u_h, v_h)$ is the same as in (2.20) but considering the discrete functional space V_h^0.

As it was proved in Subsection 2.2.2, Equation (2.31) has a unique solution

such that $u_h \in V_h^0$. Also the following error estimate can be derived (Cea's lemma [21]):

$$\|u - u_h\|_1 \leq \frac{N_a}{M_a} \inf_{v_h \in V_h^0} \|u - v_h\|_1 \tag{2.32}$$

and where u is the solution to the problem defined in Equation (2.21). From Equation (2.32) we can deduce that this error bound is suited if $\frac{N_a}{M_a}$ is not too large. To see this better let us use the results obtained in Equations (2.24) and (2.25) for N_a and M_a respectively. In this case the relationship between the two is given by:

$$\frac{N_a}{M_a} = \frac{1}{C} + \frac{\mathrm{Pe}}{C}\|a\|_\infty \tag{2.33}$$

This means that for large Péclet numbers the right-hand-side of Equation (2.32) can also be very large. In practice this leads to oscillations in the solution, as described in [15]. In these cases, a stabilization technique is needed.

2.2.5 Stabilized Finite Elements

As it has been shown, the Galerkin discretization is not appropriate to convection dominated problems. A remedy to this, is to use a stabilization technique. Before continuing, it must be clarified that the scope of this thesis is not about comparing the different existent stabilization strategies, there will be only used as a tool needed for the type of problems that will be solved. In this framework, a small overview of the streamline upwind Petrov-Galerkin (SUPG) method will be introduced, in order to support the results in the next chapters. For further details about stabilization techniques see [24].

In general for the stabilization methods that will be considered in this work, the discrete problem that needs to be solved is the following:

$$a(u_h, v_h) + S(u_h, v_h) = f(v_h) \quad \forall v_h \in V_h \tag{2.34}$$

where $S(u_h, v_h)$ is the stabilization term.

For solving the equation, several models can be considered as illustrated in Figure 2.6. Historically, artificial viscosity methods were intended to add a stabilising isotropic diffusion [59]. Schemes of different orders have been proposed and in general, the numerical viscosity introduced scales with the mesh size, to progressively disappear while the mesh is refined. Later, streamline upwind methods (SU) were adopted to introduce diffusion only in the streamline direction [16]. Such methods suffer from the fact that they are not consistent, that is an exact solution to the PDE does not satisfy the weak form. To correct this, the streamline upwind Petrov-Galerkin (SUPG) methods [17] are similar to their SU counterpart but are consistent, as the added term is proportional to the residual of the equation. Then, Galerkin-Least-Square (GLS) methods [57] were developed as a generalisation of stabilisation methods to any type of PDE and to systems of equations. In 1995, Hughes [56] reinterpreted the stabilization methods in the context of Variational Multiscale methods (VMS). In brief, the stability is provided by modelling the sub grid scale, that is the scale not resolved by the mesh, and including it in the resolved scale equation. In this context, Algebraic Subgrid Scale methods propose an algebraic equation for the subgrid scale that poses mathematical grounds to previous stabilisation methods. Once this subgrid scale has been identified and modelled, families of tracking methods for the subgrid scale have been proposed to follow the history of the subgrid scale

for unsteady problems and to consider the subgrid scale in the non-linear terms of the equation. In parallel, orthogonal subgrid scales method (OSS) or split OSS methods intend to increase the accuracy of the stabilized formulation by choosing the subgrid scale in an appropriate space, the space orthogonal to the space of the resolved scale [24, 59].

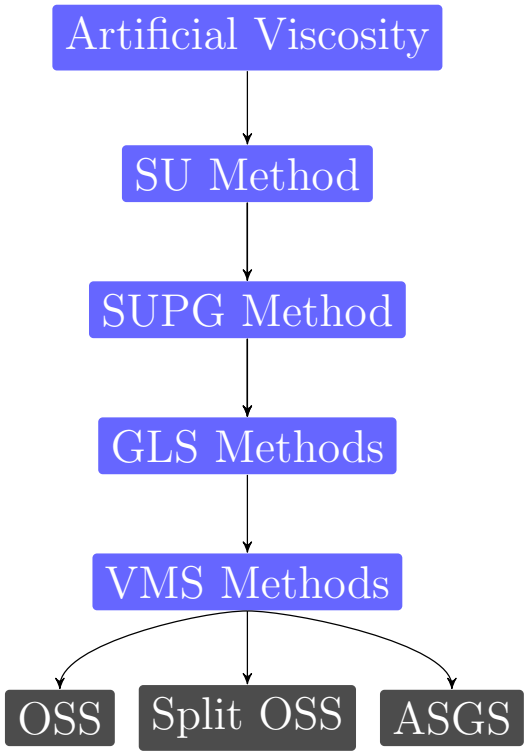


Figure 2.6: Evolution of the different stabilization methods

Streamline upwind Petrov-Galerkin

Before starting with the SUPG method, we need to define the following:

- h_e is a characteristic non-dimensional length of the element e (minimum edge size in the direction of the advection **a**)
- We simplify the integral over all elements as:

$$\int_{\Omega'} := \sum_{e=1}^{n_e} \int_{\Omega_e}$$

where Ω_e is the interior of each element e that the mesh has.

Then, in case of the SUPG method, the stabilization term is given by:

$$S(u_h, v_h) = \int_{\Omega'} \tau_e(\mathbf{a} \cdot \nabla v_h)(L(u_h) - f(v_h))d\Omega \tag{2.35}$$

where τ_e is the stability parameter at element level and it is given by [58, 62, 63]:

$$\tau_e = \frac{h_e}{2|\mathbf{a}|}\left(\coth \mathrm{Pe} - \frac{1}{\mathrm{Pe}}\right) \quad \text{with} \quad \mathrm{Pe} = \frac{|\mathbf{a}|\, h_e}{2k} \tag{2.36}$$

A different formulation is used if quadratic elements are used [25]. If the asymptotic behavior of this last equation is analyzed, then:

$$\tau_e = \begin{cases} \dfrac{h_e}{2|\mathbf{a}|} & \text{if } \mathrm{Pe} \to \infty \\ 0 & \text{if } \mathrm{Pe} \to 0 \end{cases} \tag{2.37}$$

From Equation (2.37) we observe that when $\mathrm{Pe} \to 0$, $\tau_e \to 0$ meaning that the term $S(u_h, v_h)$ will be negligible in Equation (2.34).

Let us analyze now Equation (2.34), taking $u_h = v_h$:

$$a(u_h, u_h) = \frac{1}{\mathrm{Pe}}\|\nabla u_h\|_1^2 + \left\|\tau_e^{1/2}\mathbf{a} \cdot \nabla u_h\right\|_1^2 \tag{2.38}$$

where τ_e is given by Equation (2.36). From this last equation we observe that when $Pe \to \infty$ we still have control over the gradient in the direction of the flux provided by the second term of Equation (2.38).

2.2.6 Matrix Form of the Finite Element Formulation

Since the objective of the finite element formulation is to build a linear system of equations, we build its matrix form, as this will be useful at

the end of this chapter to discuss the resultant matrices coming form the advection-diffusion equation. Also, this will give us a first idea on how to adapt preconditioning techniques to the physics of the problem.

Let us consider a basis of functions $\phi_i \in V_h^0$ with $1 \leq i \leq N$. And let us express u_h in this basis:

$$u_h = \sum_{j=1}^{N} u_j \phi_j \tag{2.39}$$

Then using it in Equation (2.31) (analogously the same can be done with Equation (2.34)) we obtain:

$$\sum_{j=1}^{N} u_j a(\phi_j, \phi_i) = f(\phi_i) \ \ \forall \ \ i = 1, ..., N \tag{2.40}$$

Defining then $A = a(\phi_j, \phi_i)$ and $f_i = f(\phi_i)$ the following linear system is obtained:

$$A\mathbf{u} = \mathbf{f} \tag{2.41}$$

where $\mathbf{u} = (u_1, ..., u_N)^T \in \mathbb{R}^N$ is the vector of coefficients of u_h, A is an $N \times N$ matrix and f $\in \mathbb{R}^N$ is the right-hand-side vector.

2.2.7 Transient Problem

To solve the transient problem let us consider the following:

$$\begin{cases} \partial_t u + Lu = f \text{ in } \ \Omega \times (0, T) \\ \qquad u = 0 \quad \text{ on } \ \partial\Omega \times (0, T) \\ \qquad u = u_0 \quad \text{ in } \ \Omega \ \times \{0\} \end{cases} \tag{2.42}$$

where the time interval goes from time 0 to time T and Lu is defined by (2.22) and where f can depend also explicitly of t, but for the sake of simplicity this will not be considered. To solve the transient problem, we use the generalized trapezoidal rule [6] as shown in Equation (2.43).

$$\begin{aligned} \partial_t u^{n+\theta} &:= \frac{u^{n+\theta} - u^n}{\theta \Delta t} \\ \Delta t &:= t^{n+1} - t^n \\ u^{n+\theta} &:= \theta u^{n+1} + (1-\theta) u^n \end{aligned} \tag{2.43}$$

where n is the time step number and θ is a parameter related to the method used in each case. It is always in the interval $[0, 1]$ and Δt is the time step size. Finally putting all these together and using (2.42), the equation to be solved for $u^{n+\theta}$ is:

$$\partial_t u^{n+\theta} + L^{n+\theta} u^{n+\theta} = f^{n+\theta} \tag{2.44}$$

Then the solution at time step $n+1$ will be given by:

$$u^{n+1} = u^n + \frac{u^{n+\theta} - u^n}{\theta} \tag{2.45}$$

By using the trapezoidal rule we obtain the formulation of the transient problems by carrying out the time discretization on 2.42 and then deriving the weak formulation.

There are mainly two strategies to select the time step size Δt. On the one hand, when implicit methods are considered and the time scales of the problems are known, the time step can be prescribed by the user. On the other hand, explicit schemes are constrained by the Courant-Friedrichs-

Lewy (CFL) condition [27] which imposes a critical (maximum) time step Δt_c to provide temporal stability of the scheme. Also, when considering implicit schemes and when no indication on the time scales of the problem are known a priori, it may be convenient to select the time step as a function of the critical time step. The multiply α is called the safety factor and the resulting time step is therefore $\Delta t = \alpha \Delta t_c$ [18, Chapter 9]. A classical expression for the critical time step for the AD equation is $\Delta t_c = \min_i \Delta t_{c_i}$, where Δt_{c_i} is the critical time step of element I such that:

$$\Delta t_{c_i} = \frac{1}{\frac{2|a|}{h} + \frac{4k}{h^2}}$$

In this section we will list some of the integration methods used in this thesis.

Backward Euler approximation

This is a first order implicit approximation and it is obtained by choosing $\theta = 1$. Then substituting in Equation (2.44):

$$\partial_t u^{n+1} + L^{n+1} u^{n+1} = f^{n+1}$$

And the Galerkin formulation, using a backward Euler approximation, of the problem defined in (2.42) will consist in finding for each time step n, $u_h^{n+1} \in V_h$ such that:

$$\begin{aligned} (\partial_t u_h^{n+1}, v_h) + a^{n+1}(u_h^{n+1}, v_h) &= f^{n+1}(v_h) \forall v_h \in V_h \\ (u_h^0, v_h) &= (u_0, v_h) \ \ \forall v_h \in V_h \end{aligned} \tag{2.46}$$

where again $a^{n+1}(u_h^{n+1}, v_h)$ is the bilinear form defined in (2.20) at time

step $n+1$.

Crank Nicolson approximation

Crank Nicolson approximation is a second order method implicit in time and it is obtained by choosing $\theta = 1/2$, this is:

$$\partial_t u^{n+1/2} + L^{n+1/2} u^{n+1/2} = f^{n+1/2}$$

And analogously as it happened with the backward Euler approximation, the Galerkin formulation, using a backward Euler approximation, of the problem defined in (2.42) will consist in finding for each time step n, $u_h^{n+1/2} \in V$ such that:

$$\begin{aligned} (\partial_t u_h^{n+1/2}, v_h) + a^{n+1/2}(u_h^{n+1/2}, v_h) &= f^{n+1/2}(v_h) \forall v_h \in V_h \\ (u_h^0, v_h) &= (u_0, v_h) \ \ \forall v_h \in V_h \end{aligned} \tag{2.47}$$

To obtain the solution at $n+1$, Equation (2.45) has to be used.

2.2.8 Integration Rules

In Subsection 2.2.6 we show that the finite element method requires the calculation of integrals over individual elements. In some occasions these integrals can be calculated analytically, but often they are too difficult and numerical integration methods are required. There are several integration methods such as the trapezoidal rule [6] or Simpson's rules [100], although Gaussian quadrature is one of the most frequently applied to numerical integration methods [75]. It is the one that we have used in this thesis.

Gaussian quadrature approximates an integral as the weighted sum of the values of its integrands as shown in Equation (2.48) for a one-dimensional integral, over the interval $-1 \leq \xi \leq 1$.

$$\int_{-1}^{1} f(\xi) d\xi \approx \sum_{q=1}^{N_q} \omega_q f(\xi_q) \tag{2.48}$$

where N_q is the total number of quadrature points, ξ_q is the location of the qth quadrature point in the domain, and ω_q is the corresponding quadrature weight. The extension to multi-dimensional integrals can be found in [72].

We will call it a closed rule if the chosen quadrature points are located at the bounds of the interval [-1,1] and an open rule if the quadrature nodes are interior nodes of the interval [-1,1]. As we will see in the next section, the way in which these integrals are calculated will have an influence on the structure of the matrix of the linear system defined in Equation (2.41).

2.3 Case Studies of the Advection-diffusion Equation

In this section a series of problems are selected to show the physical and numerical behaviors explained in this chapter. Also they will be the theoretical basis that will justify the renumbering algorithms developed in this thesis. In particular, we will evidence some specific structures of the matrix under different scenarios: strong advection and boundary layer mesh. When referring to the structure of a matrix, we refer to the original connectivity while excluding the null coefficients.

2.3.1 Example 1: Strong Advection

This example will consider the problem defined in (2.22) taking the velocity constant only in the y direction, so that $\mathbf{a} = (0, a_y)$. As the Péclet number is high, the convection term is dominant over the diffusive one. To study this problem let us consider a patch of a typical regular 2-dimensional mesh made of four linear quadratic elements and nine nodes. The example will solve the advection-diffusion equation for node 5, the other nodes being boundary nodes. It should be noted that the nodes have been ordered in a way that we can evidence some specific structure, and that the velocity has been chosen in the y direction.

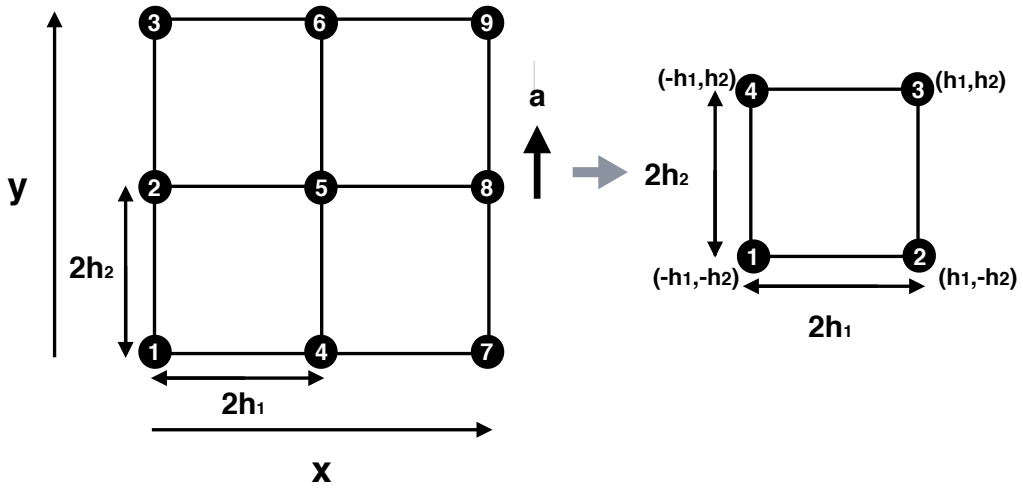


Figure 2.7: *Global mesh (left) and single element with local numbering (right).*

In Figure 2.7, h_1 and h_2 are the element sizes in the x and y axes respectively. Also, as it is shown in Figure 2.7 we have chosen $h_1 = h_2 = h$.

Considering just one element of the mesh given in Figure 2.7, the following

change of variables is done from cartesian to local coordinates as shown in Figure 2.8:

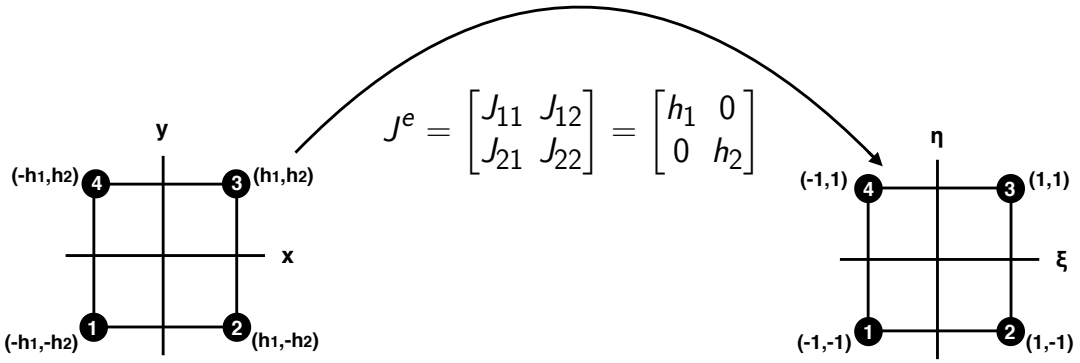


Figure 2.8: *Isoparametric transformation in the 2D mesh.*

where ξ and μ are the local coordinates. Then, the shape functions and local derivatives are given by:

$$\begin{aligned}
\Phi_1(\xi,\eta) &= \frac{1}{4}(1-\xi-\eta+\xi\eta) & \frac{\partial\Phi_1}{\partial\xi} &= \frac{1}{4}(\eta-1) & \frac{\partial\Phi_1}{\partial\eta} &= \frac{1}{4}(\xi-1) \\
\Phi_2(\xi,\eta) &= \frac{1}{4}(1+\xi-\eta-\xi\eta) & \frac{\partial\Phi_2}{\partial\xi} &= \frac{1}{4}(1-\eta) & \frac{\partial\Phi_2}{\partial\eta} &= -\frac{1}{4}(\xi+1) \\
\Phi_3(\xi,\eta) &= \frac{1}{4}(1+\xi+\eta+\xi\eta) & \frac{\partial\Phi_3}{\partial\xi} &= \frac{1}{4}(\eta+1) & \frac{\partial\Phi_3}{\partial\eta} &= \frac{1}{4}(\xi+1) \\
\Phi_4(\xi,\eta) &= \frac{1}{4}(1-\xi+\eta-\xi\eta) & \frac{\partial\Phi_4}{\partial\xi} &= -\frac{1}{4}(\eta+1) & \frac{\partial\Phi_4}{\partial\eta} &= \frac{1}{4}(1-\xi)
\end{aligned} \tag{2.49}$$

Substituting them into (2.31), the following local matrices are obtained for each element:

$$
\begin{aligned}
K_s^l &= \int_{\Omega_e} \left(J_{11}^{-1} \frac{\partial \Phi_i}{\partial \xi} + \cancel{J_{12}^{-1} \frac{\partial \Phi_i}{\partial \eta}} \right) \cdot \left(J_{11}^{-1} \frac{\partial \Phi_j}{\partial \xi} + \cancel{J_{12}^{-1} \frac{\partial \Phi_j}{\partial \eta}} \right) \left| J^e \right| d\xi d\eta \\
&+ \int_{\Omega_e} \left(\cancel{J_{21}^{-1} \frac{\partial \Phi_i}{\partial \xi}} + J_{22}^{-1} \frac{\partial \Phi_i}{\partial \eta} \right) \cdot \left(\cancel{J_{21}^{-1} \frac{\partial \Phi_j}{\partial \xi}} + J_{22}^{-1} \frac{\partial \Phi_j}{\partial \eta} \right) \left| J^e \right| d\xi d\eta \\
C_s^l &= \int_{\Omega_e} \Phi_i \left(\underbrace{a_x}_{=0} \left(J_{11}^{-1} \frac{\partial \Phi_j}{\partial \xi} + \cancel{J_{12}^{-1} \frac{\partial \Phi_j}{\partial \eta}} \right) + a_y \left(\cancel{J_{21}^{-1} \frac{\partial \Phi_j}{\partial \xi}} + J_{22}^{-1} \frac{\partial \Phi_j}{\partial \eta} \right) \right) \left| J^e \right| d\xi d\eta
\end{aligned}
\tag{2.50}
$$

where J^e is the Jacobian of the transformation and K_s^l and C_s^l correspond to the stiffness matrix that belongs to the diffusive term of (2.22) respectively. The subindex s will be either o if the integration is done with an open integration rule or c if the integration of the variational form is done with a closed integration rule.

Then if an open integration rule is considered, the following result is obtained for the local matrices:

$$
K_o^l = \frac{1}{6} \begin{bmatrix} 4 & -1 & -2 & -1 \\ -1 & 4 & -1 & -2 \\ -2 & -1 & 4 & -1 \\ -1 & -2 & -1 & 4 \end{bmatrix} \quad C_o^l = \frac{h a_y}{6} \begin{bmatrix} -2 & -1 & 1 & 2 \\ -1 & -2 & 2 & 1 \\ -1 & -2 & 2 & 1 \\ -2 & -1 & 1 & 2 \end{bmatrix}
$$

and looking at the global matrices of the local matrices above:

$$K_o^g = \frac{1}{6}\begin{bmatrix} 4 & -1 & 0 & -1 & -2 & 0 & 0 & 0 & 0 \\ -1 & 8 & -1 & -2 & -2 & -2 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & -2 & -1 & 0 & 0 & 0 \\ -1 & -2 & 0 & 8 & -2 & 0 & -1 & -2 & 0 \\ -2 & -2 & -2 & -2 & 16 & -2 & -2 & -2 & -2 \\ 0 & -2 & -1 & 0 & -2 & 8 & 0 & -2 & -1 \\ 0 & 0 & 0 & -1 & -2 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & -2 & -2 & -2 & -1 & 8 & -1 \\ 0 & 0 & 0 & 0 & -2 & -1 & 0 & -1 & 4 \end{bmatrix}$$

$$C_o^g = \frac{a_y h}{6}\begin{bmatrix} -2 & 2 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ -2 & 0 & 2 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -2 & 2 & 0 & -1 & 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & -4 & 4 & 0 & -1 & 1 & 0 \\ -1 & 0 & 1 & -4 & 0 & 4 & -1 & 0 & 1 \\ 0 & -1 & 1 & 0 & -4 & 4 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 & 1 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & -2 & 0 & 2 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & -2 & 2 \end{bmatrix}$$

Now if the same is done using a closed integration rule, then:

$$K_c^l = \frac{1}{2}\begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix} \quad C_c^l = \frac{a_y h}{2}\begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

and in this case the global matrices will be the following:

$$K_c^g = \frac{1}{2}\begin{bmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 0 \\ 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{bmatrix}$$

$$C_c^g = \frac{a_y h}{2}\begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Finally the local elemental matrices are presented for SUPG stabilization, where the coefficient τ_e has been used in its asymptotic form given by (2.37). Also we present, the resultant global matrix after summing the convective term C together with the SUPG stabilization term S. Both calculations are done with open and closed integration rules.

In the case of having an open integration rule, the local elementary matrices are the following:

$$S_o^l = a_y^2 \tau_e \frac{1}{6} \begin{bmatrix} 2 & 1 & -1 & -2 \\ 1 & 2 & -2 & -1 \\ -1 & -2 & 2 & 1 \\ -2 & -1 & 1 & 2 \end{bmatrix}$$

where τ_e is taken as the one defined in (2.36) and the resultant global matrix for the new convective term is:

$$C_o^g + S_o^g = \frac{a_y h}{6} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 4 & 0 & -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & -4 & 4 & 0 & -2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 2 & 0 & -8 & 8 & 0 & -2 & 2 & 0 \\ 0 & -2 & 2 & 0 & -8 & 8 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 2 & 0 & -4 & 4 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 & 0 & -4 & 4 \end{bmatrix}$$

If the same is done with a closed integration rule, the local elementary matrix is:

$$S_c^l = a_y^2 \tau_e \frac{1}{2} \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

and the resultant global matrix in this case is:

$$C_c^g + S_c^g = \frac{a_y h}{2} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -4 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 2 \end{bmatrix}$$

Remarks

Several conclusions come out from the derivations carried out:

- First, as it is shown in Figure 2.7 the mesh has been ordered in the same direction as the advection velocity, which is given in the y direction. This choice was made to evidence the structure of the matrix.

- Second, in order to solve the problem (or equivalently to invert the global matrix), boundary conditions should be prescribed. We observe that in the case of pure advection (stabilized with SUPG), the rows of 1, 4, 7 are null, suggesting that the unknown must be prescribed on these nodes. This is achieved by adding a non-zero in the diagonal of their respective rows. Otherwise zero flux is prescribed and the rows remain unchanged.

- The Galerkin terms C_o^g and C_c^g have no specific structure, although we observe in C_c^g a zero on the diagonal of node 5; this is a centered scheme [91].

- In the case of $C_c^g + S_c^g$ we have independent rows in the direction of the advection (independent streamlines). in this example we have 3 streamlines aligned along the nodes (1,2,3), (4,5,6) and (7,8,9). This is because there is only a dependence with the previous node (downstream node).

- This structure does not appear in the case of open integration rule. Figure 2.9 shows the domain of influence of both stabilized convection matrices. We observe that for both rules, the domain of influence is upstream, justifying the term streamline upwind for the SUPG method.

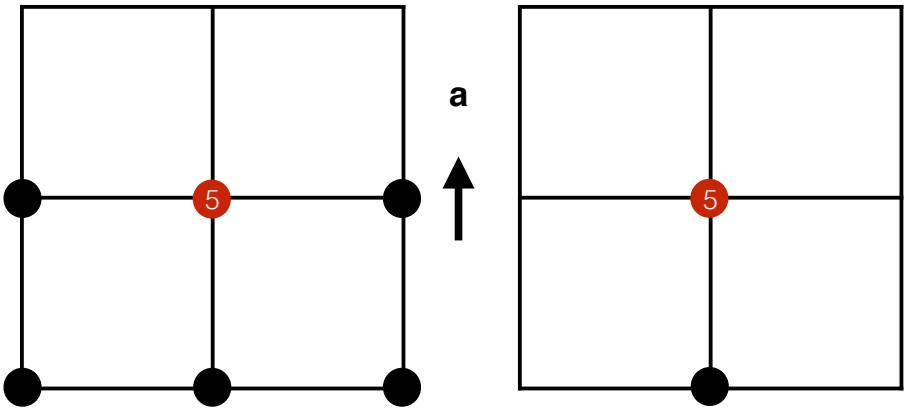


Figure 2.9: *Domain of influence of node 5 for convection and stabilization matrices with open integration rule (left) and closed integration rule (right).*

- Regarding diffusion, we observe that the close rule has a cross structure, similar to the one we obtain in finite differences. On the contrary, the open rule has a complete structure. This is sketched in Figure 2.10.

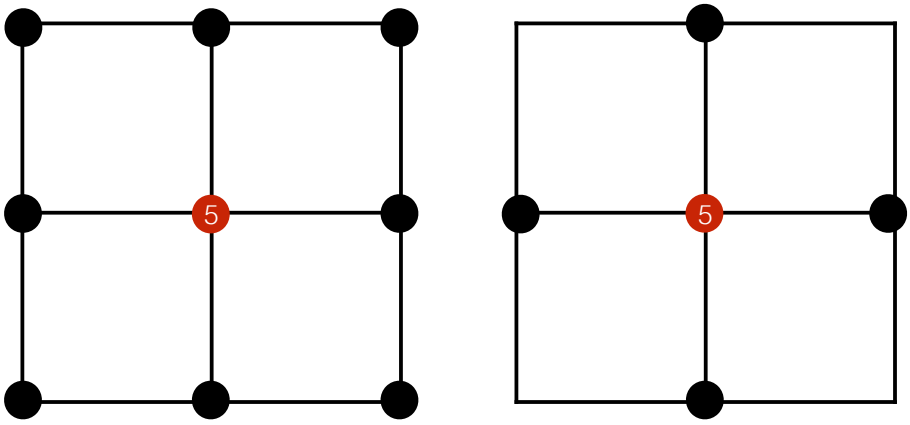


Figure 2.10: *Domain of influence of node 5 for diffusion matrices with open integration rule (left) and open integration rule (right).*

2.3.2 Example 2: Pure Diffusion with Mesh Anisotropy

In this case the same mesh as in Example 1 will be considered, but now the mesh will have an anisotropy. This example represents the typical mesh that can be found usually when solving boundary layer problems, shear layer problems etc., for which the variation of the solution is concentrated in very thin layers. In this example diffusion dominates advection, this is why only the diffusion matrix will be shown in this example.

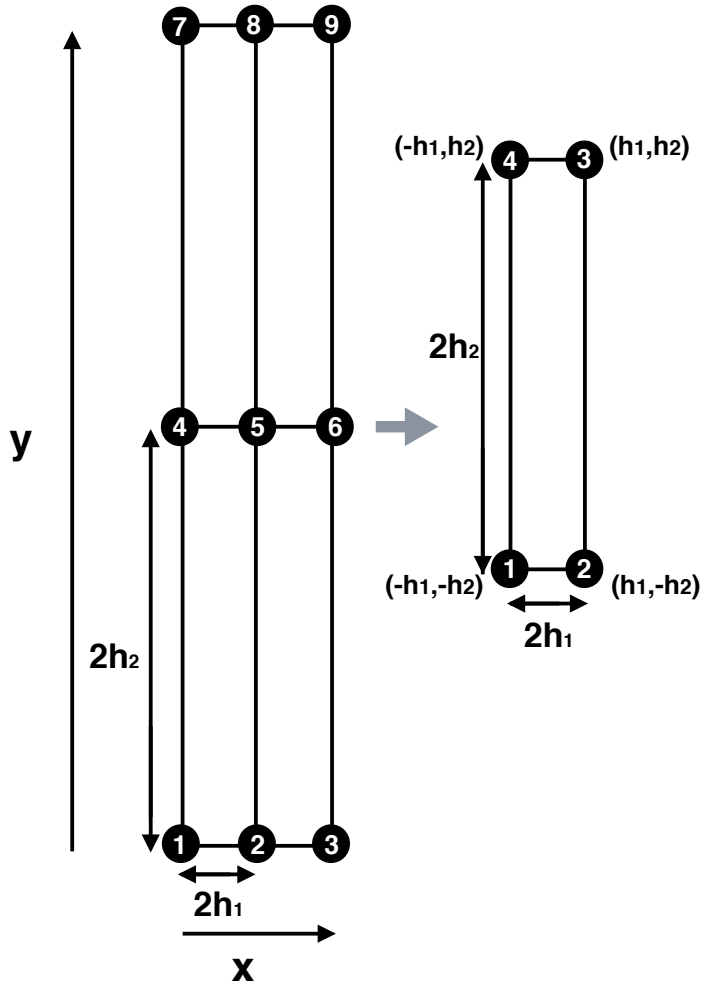


Figure 2.11: *Global mesh (left) and single element with local numbering (right).*

Like in Figure 2.7 h_1 and h_2 are the element sizes in the x and y axes respectively and as it is shown in Figure 2.11 in this case $h_1 \ll h_2$. This choice is used to neglect some terms of the matrices, in particular, the terms proportional to h_1/h_2. We define the aspect ratio as:

$$a_r = \frac{h_2}{h_1} \tag{2.51}$$

According to this definition, the situation in this example corresponds to a high aspect ratio. Then using the same change of coordinates as the one shown in Figure 2.8 of Example 1 and using the same shape functions defined in (2.49), the resultant matrices for each element are in case of using

an open integration rule:

$$K_o^l = \frac{h_2}{6h_1}\begin{bmatrix} 2 & -2 & -1 & 1 \\ -2 & 2 & 1 & -1 \\ -1 & 1 & 2 & -2 \\ 1 & -1 & -2 & 2 \end{bmatrix}$$

Finally the global matrix in this example for the stiffness (diffusion) matrix K will be:

$$K_o^g = \frac{h_2}{6h_1}\begin{bmatrix} 2 & -2 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -2 & 4 & -2 & -1 & -1 & 2 & -1 & 0 & 0 \\ 0 & -2 & 2 & 0 & -1 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 4 & -4 & 0 & 1 & -1 & 0 \\ -1 & 2 & -1 & -4 & 8 & -4 & -1 & 2 & -1 \\ 0 & -1 & 1 & 0 & -4 & 4 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 2 & -2 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & -2 & 4 & -2 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & -2 & 2 \end{bmatrix}$$

And if the same is done with an close integration rule, the result for the local and global stiffness matrix are the following:

$$K_c^l = \frac{h_2}{2h_1}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

for the local matrix and

$$K_c^g = \frac{h_2}{2h_1}\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 4 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

for the global one.

Remarks

Several things come out from the calculations done in this example:

- First, as it is shown in Figure 2.7 the mesh has been ordered in the direction of anisotropy of the mesh, which is given in the x direction. As it happened with the previous example, this choice was made to show the structure of the matrix.

- Second, this patch is done for row 5, but in order to solve the problem (or equivalently to invert the global matrix), we need to prescribe at least the value on one node.

- K_o^g has no specific structure, and the coefficients are of the same order. On the contrary in K_c^g we observe a tridiagonal structure. These domain of influence are shown in Figure 2.12. This Figure should be compared with Figure 2.10 for the case $h_1 = h_2$.

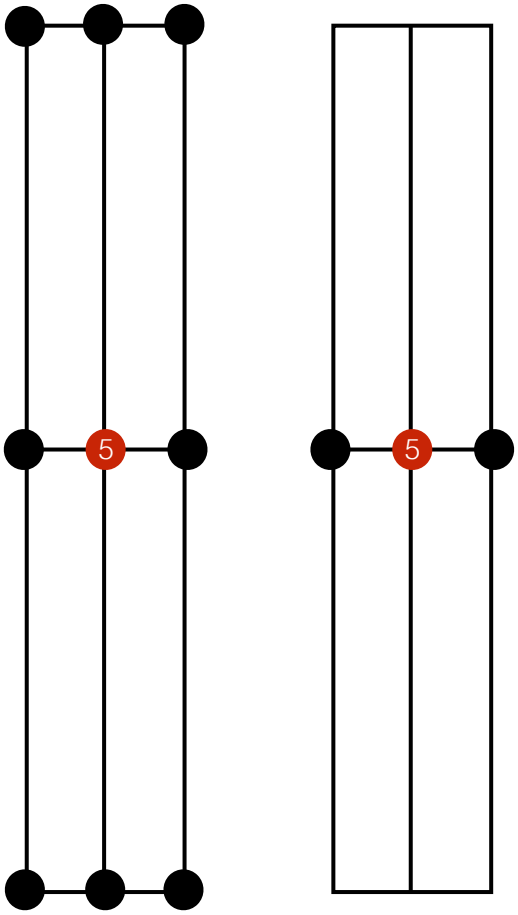


Figure 2.12: *Domain of influence of node 5 for the diffusion matrix with open integration rule (left) and closed integration rule (right).*

2.3.3 Conclusions

In this Section two canonical cases often encountered in CFD have been analyzed. The first example could correspond to any heat transfer problem with a high advection, whereas the second one could be associated to a parabolic boundary layer problem. In both cases and independently of the physics of the problem itself, we have demonstrated that the order in which the mesh is renumbered can exhibit some interesting structures (tridiagonal and lower diagonal). In more general problems, these limiting regimes can happen together. Then, by performing a careful renumbering according to the local physics or mesh anisotropy, we can in principle exhibit such structures in

the same way it was done for the simple cases. This opens the possibility of applying specific preconditioning techniques adapted to these structures.

Chapter 3

Sparse Linear Systems

'I'm not young enough to know everything'

J.M. Barrie,
Peter Pan

This chapter presents the state-of-the-art solvers used in this thesis to solve the advection-diffusion and Navier-Stokes equations. Section 3.1 shows how the assembly of an algebraic system is done and how matrices are stored. Several aspects on basic linear algebra are discussed in Section 3.2. Section 3.3 focuses on the different existing solvers used to solve algebraic linear systems of equations depending on the matrix properties described in Section 3.2. Finally, in the last section we explain the parallelization of the different operations involved.

3.1 Assembly and Storage

As we have seen with the cases of Section 2.3, the discretization of the partial differential equations lead to sparse matrices. By sparse matrix, we understand a matrix that only has a few entries different from zero. In this section we give a short overview on how the system of equations is assembled and which technique we will use to store sparse matrices, since the zero elements

in this matrices do not need to be stored.

3.1.1 Algebraic System Assembly

The discretization methods considered, Finite Element (FE), Finite Volume (FV) or Finite Difference (FD), transform a continuous problem into a discrete problem of finite dimension that results in solving a linear system of equations of the type:

$$A\mathbf{x} = \mathbf{b} \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a matrix representing the coefficients relating the unknowns in each equation, $\mathbf{x} \in \mathbb{R}^n$ is the vector of unknowns and $\mathbf{b} \in \mathbb{R}^n$ is the right-hand side term. Note that this equation is the same as Equation (2.41), obtained from the finite element discretization of the advection-diffusion equation.

In the context of classical finite element method as shown in Chapter 2, the coefficients of the matrix are obtained computing a loop over the element mesh, in each elemental matrix. Finally these coefficients are assembled into the global matrix.

The matrices resulting from these discretizations are often large and sparse, meaning that they have only a few entries different from zero. Thus solving the system defined in Equation (3.1) can be challenging.

3.1.2 Matrix Storage

Several storage schemes have been proposed for these sparse matrices. The matrix storage format is determined mainly by two constraints: First to keep the memory requirements as low as possible to store the coefficients,

and second to make the sparse matrix vector product (SpMV) as fast as possible. Different storage formats can be found in the literature, such as Coordinate Storage (COO) [49, 79], Jagged Diagonal Storage (JAD) [87], Compressed Sparse Row (CSR) [79], Compressed Sparse Column (CSC) [33, 79] or ELLPACK (ELL) [37]. In this document, only the CSR format will be explained in detail.

Let us consider the following example:

$$\begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix} \tag{3.2}$$

The previous matrix in CSR will be expressed with the following data structures:

$$\text{aa} = \begin{bmatrix} 10, -2, 3, 9, 3, 7, 8, 7, 3, 8, 7, 5, 8, 9, 9, 13, 4, 2, -1 \end{bmatrix}$$

$$\text{ja} = \begin{bmatrix} 1, 5, 1, 2, 6, 2, 3, 4, 1, 4, 5, 6, 2, 4, 5, 6, 2, 5, 6 \end{bmatrix}$$

$$\text{ia} = \begin{bmatrix} 1, 3, 6, 9, 13, 17, 20 \end{bmatrix}$$

The array aa stores the non-zero values of the sparse matrix, ja indicates the corresponding column indexes and ia contains the pointers of the previous arrays in which each row starts.

3.2 Basic Linear Algebra

Several aspects have to be taken into account when solving the system in Equation (3.1), such as matrix properties and matrix conditioning. The type of matrix obtained and its associated condition number are related directly to the physics of the discretized equation, these will determine eventually the appropriate solver and preconditioner used in each case. Also, we will explain matrix permutation and restriction. These will be of special interest in Chapters 4 and 5.

3.2.1 Matrix Properties

Depending on the nature of the physical problem and the numerical methods studied in each case, the resulting sparse matrix can have different properties after the discretization. In this section, some of them will be mentioned.

Let $A = [a_{ij}]$ be a real square matrix of dimension $n \times n$ with coefficients $a_{ij} \in \mathbb{R}$ such that $1 \leq i \leq n$ and $1 \leq j \leq n$ and where i and j represent the row and column index respectively. If we denote $A^t = [a_{ji}]$ as the transpose of matrix A, we can define the following properties:

- **Symmetry** if $A = A^t \Rightarrow a_{ij} = a_{ji}$ is symmetric, otherwise is **non-symmetric**.

  The discretization of the Laplacian in the convection-diffusion equation described in Chapter 2 for a pure diffusive problem, results in a symmetrix matrix. The presence of the convection term in Equation (2.1) makes it non-symmetric. These two statements have been shown in detail in Section 2.3.

- **Symmetric Positive Definite** A symmetric, square matrix A of dimension $n \times n$ with real coefficients is said to be positive definite, if it verifies one of the following equivalent properties [42]:

  (i) $\forall$ non-zero vectors $\mathbf{x}$ with n rows:

  $$(\mathbf{x}, \mathbf{x})_A = (A\mathbf{x}, \mathbf{x}) = \mathbf{x}^t A\mathbf{x} > 0$$

  (ii) All the eigenvalues $\lambda_i(A) \;\; i = 1, ..., n$ of the matrix A are strictly positive.

  (iii) The symmetric bilinear form $(\mathbf{x}, \mathbf{y})_A = (A\mathbf{x}, \mathbf{y}) = \mathbf{y}^t A\mathbf{x}$ is a scalar product on $\mathbb{R}^n$

  Theorem 3.2.1. *If A is positive definite, then:*

  $$(\boldsymbol{x}, \boldsymbol{y})_A = (A\boldsymbol{x}, \boldsymbol{y})$$

  defines an inner product, and

  $$\|\boldsymbol{x}\|_A = \sqrt{(\boldsymbol{x}, \boldsymbol{x})_A}$$

  defines a norm.

  For a pure diffusive problem the resultant matrix is symmetric and positive definite [13, 45, 52].

- **Orthogonality** $AA^T = A^T A = I \Rightarrow \sum_k a_{ki} a_{kj} = \sum_k a_{ik} a_{jk} = \delta_{ij}$ is an orthogonal matrix, where I is the identity matrix and δ_{ij} the Kronecker symbol.

3.2.2 Matrix Permutation

In the last chapter, it was shown that we could explicitly exhibit some convenient properties of the matrix by numbering the nodes of the mesh properly. Given an arbitrary ordering, the desired numbering can be obtained by permuting rows and columns of the original matrix. Such a permutation can be achieved using a permutation matrix P of size $n \times n$ and such that $PP^T = I$, therefore P is an orthogonal matrix. This matrix has one single non-null coefficient per row corresponding to the permutation to apply. To permute row and column i and row and column j, we simply set $P_{ij} = 1$, while if no permutation is desired, $P_{ii} = 1$. The permutation of matrix A to matrix A_P is then done in the following way:

$$A_p = PAP^T \tag{3.3}$$

For example, let us consider the following matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

If we permute the first and third rows and columns of a 3×3 matrix:

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad P^T = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.4}$$

And matrix A_p:

$$A_p = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

It should be noted that in practice, such a permutation is rarely explicitly needed. Rather, a node permutation array is used to put the unknowns in a given order. However, it is a useful tool to express formally a renumbering. It will be implicitly used to design linelet type preconditioners as well as composite preconditioners in next chapters.

3.2.3 Matrix Restriction

The restriction of a matrix of rank $n \times n$ enables to extract sub-matrices A_i of rank $m_i \times m_i$ with $m_i < n$. Let us consider for example the following matrix

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

To extract two square matrices from A, one corresponding to the first 2×2 block and the other with the second 2×2 block, let us set:

$$R_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad R_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

The restricted matrices A_i can then be computed as

$$A_i = R_i A R_i^T \tag{3.6}$$

so that we find:

$$A_1 = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 11 & 12 \\ 15 & 16 \end{bmatrix} \tag{3.7}$$

The restriction matrix also enables to permute rows and columns. For the sake of clarity, we will nevertheless maintain the permutation and restriction operations independently. Finally, let us remark that the restrictions can overlap, meaning that $\sum m_i > n$. For example, this kind of restriction is the one used by overlapping Schwarz methods.

3.2.4 Matrix Norm

Let $M_{n\times n}$ be the vector space of all the matrices of size $n \times n$ and let A and B be two matrices that belong to $M_{n\times n}$. Then, the transformation $\|\cdot\| : M_{n\times n} \to \mathbb{R}$ is a matrix norm if it satisfies the following properties:

- $\|\alpha A\| = |\alpha| \, \|A\| \;\; \forall A \in M_{n\times n} \; \alpha \in \mathbb{R}$
- $\|A + B\| \leq \|A\| + \|B\| \;\; \forall A, B \in M_{n\times n}$
- $\|A\| \geq 0 \;\; \forall A \in M_{n\times n}$ with $\|A\| = 0 \iff A = 0$
- $\|AB\| \leq \|A\|\|B\|$

Induced Matrix Norms:

The most frequent type of matrix norms are induced matrix norms, these come form vector norms which are defined in detail in [39, 42]. Then, for every p-vector norm, the matrix p-norm will be defined as:

$$\|A\|_p := \max_{\|\mathbf{x}\|_p \neq 0} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p} \tag{3.8}$$

This is, the matrix norm is, the largest size of $\|A\mathbf{x}\|_p$ relative to $\|\mathbf{x}\|_p$. Particularly it can be proved that for $p = 1, 2$ and ∞ this norm is given by [88]:

$$\|A\|_1 = \max_j \sum_i |a_{ij}|$$
$$\|A\|_2 = \sqrt{\text{largest singular value of } A^t A}$$
$$\|A\|_\infty = \max_i \sum_j |a_{ij}|$$

3.2.5 Matrix Conditioning

Chapter 2 shows how a partial differential equation is discretized using the finite element method. More precisely we apply the finite element method to the convection-diffusion equation. The matrices obtained with this technique are usually large, sparse, and ill-conditioned. As it will be shown in the next section, under these conditions, the linear system defined in Equation (3.1) is usually solved with iterative methods. Estimates for the convergence in these methods, among others, depend on the condition number of the resultant matrix [79]. In the case of iterative solvers, this condition number can be improved using the appropriate preconditioning technique.These statements will be developed in Chapter 4.

Then, the conditioning of a matrix A associated to the norm p, is defined as the real number:

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p \tag{3.9}$$

As shown in the following items, the condition number $\kappa_p(A)$ can be thought of as the rate at which the solution $\mathbf{x}$ changes with respect to a change in the

right-hand-side $\mathbf{b}$. This measure is strictly a property of the matrix and is independent of the algorithm or floating-point accuracy used to solve the system itself.

- From Equation (3.1) the following system can be now considered:

$$A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$$

where $\mathbf{x}$ is the solution to Equation (3.1) and $\delta\mathbf{x}$ is the error in the solution of Equation (3.1) and $\delta\mathbf{b}$ is a perturbation of $\mathbf{b}$. Then taking:

$$\delta\mathbf{x} = A^{-1}\delta\mathbf{b} \quad \text{and} \quad \mathbf{b} = A\mathbf{x}$$

leads to:

$$\|\delta\mathbf{x}\|_p \leq \|A^{-1}\|_p \|\delta\mathbf{b}\|_p$$

$$\|\mathbf{b}\|_p \leq \|A\|_p \|\mathbf{x}\|\|_p$$

And using these two last inequalities the following is obtained:

$$\frac{\|\delta\mathbf{x}\|_p}{\|\mathbf{x}\|_p} \leq \|A\|_p \|A^{-1}\|_p \frac{\|\delta\mathbf{b}\|_p}{\|\mathbf{b}\|_p} \quad \Rightarrow \frac{\|\delta\mathbf{x}\|_p}{\|\mathbf{x}\|_p} \leq \kappa_p(A) \frac{\|\delta\mathbf{b}\|_p}{\|\mathbf{b}\|_p} \tag{3.10}$$

- Again taking Equation (3.1) but now perturbing the matrix A:

$$(A + \delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b}$$

where δA is the perturbation in the sparse matrix A, and taking:

$$\delta\mathbf{x} = A^{-1}\delta A(\mathbf{x} + \delta\mathbf{x}) \quad \text{and} \quad \mathbf{b} = A\mathbf{x}$$

Doing the same procedure as it was done with Equation (3.10), the following relation finally is obtained:

$$\frac{\|\delta \mathbf{x}\|_p}{\|\mathbf{x}+\delta \mathbf{x}\|_p} \leq \|A\|_p \|A^{-1}\|_p \frac{\|\delta A\|_p}{\|A\|_p} \quad \Rightarrow \frac{\|\delta \mathbf{x}\|_p}{\|\mathbf{x}+\delta \mathbf{x}\|_p} \leq \kappa_p(A) \frac{\|\delta A\|_p}{\|A\|_p} \tag{3.11}$$

It is easy to see from the definitions above that $\kappa(A) \geq 1$, as $1 = \|I\| = \left\|AA^{-1}\right\| \leq \left\|A^{-1}\right\| \|A\|$. This means that when the condition number is large, even a small change in $\mathbf{b}$ or A may cause a large error in $\mathbf{x}$.

Let us show now how the condition number changes with the mesh size. To do so, we will consider the one-dimensional Poisson equation with homogeneous boundary conditions:

$$\begin{cases} -\dfrac{d^2u}{dx^2} = 1 \;\; \Omega \in [0,1] \\ \qquad u = 0 \;\; \text{in} \;\; \partial\Omega \end{cases} \tag{3.12}$$

We now discretize the equation on a Cartesian grid in a segment with $N+2$ nodes spaced with a distance (mesh size) $h = \frac{1}{N+1}$, from Section 2.3 in Chapter 2, we know that the resultant matrix of the linear system will be symmetric.

Also, we know that the condition number of a symmetric squared matrix in the Euclidean space is given by [42]:

$$\kappa_2(A) = \frac{\lambda_{max}}{\lambda_{min}} \tag{3.13}$$

where λ_{max} and λ_{min} are the maximum and minimum eigenvalue respec-

tively. In this case, the condition number is:

$$\kappa_2(h) = \frac{4}{\pi^2 h^2} \tag{3.14}$$

See [42] for more details on how to obtain Equation (3.14). The results of plotting Equation (3.14) are shown in Figure 3.1.

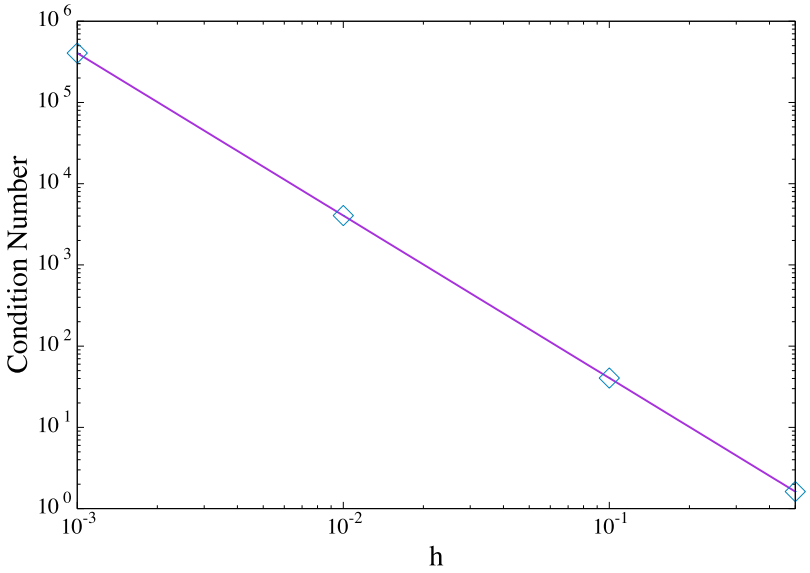


Figure 3.1: *Evolution of the condition number for different mesh sizes.*

Figure 3.1 shows that the condition number increases as the mesh gets more refined. As we will show in Subsection 3.3.2 this will affect the convergence of the iterative method chosen to solve the system.

3.3 Solution of Sparse Linear Systems of Equations

The linear system defined in Equation (3.1) can be solved with direct or iterative methods, although in most cases, iterative methods are the ones preferred to solve sparse matrices, as direct methods scale poorly with problem size n in terms of problem size n, arithmetic operations and memory requirements [68] (see Table 3.1).

	Arithmetic Operations	**Memory Demand**
Direct Methods	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$
Iterative Methods	$\mathcal{O}(n)$	$\mathcal{O}(n)$

Table 3.1: *A comparison of arithmetic operations and memory requirements between direct and iterative methods (considered in this work) when solving sparse linear systems of equations [29, 68].*

In this section, a small overview of some state-of-the-art methods will be given, focusing mainly on the Conjugate Gradient, GMRES, and BiCGSTAB methods as these are the ones used to carry out the test cases in this thesis. Stationary methods are explained briefly for two reasons: First, to have a general overview, as they are part of the state-of-the-art methods, and second, because some of them, will be mentioned in Chapter 4 as they are used as preconditioners.

3.3.1 Direct Methods

Direct methods provide the solution of the system defined in Equation (3.1) in a finite number of steps. In general, to solve large sparse linear systems, they are not a good option, as the number of operations needed to solve the system and memory requirements may be too demanding as shown in Table 3.1. However, in some cases, such as some non-linear problems in solid mechanics, which are often very ill-conditioned, they can be competitive with iterative solvers [95].

In general, direct methods are divided into two groups:

- Elimination methods, such as Gauss method.
- Factorization methods, such as LU or Cholesky factorizations.

In this document only factorization methods will be briefly presented, as they are more commonly used when solving linear systems.

LU Factorization

In this case the matrix A of Equation (3.1) is factorized as a product of a lower triangular matrix L and an upper triangular matrix U. Then Equation (3.1) is solved in the following way:

$$A\mathbf{x} = \mathrm{b} \Rightarrow LU\mathbf{x} = \mathbf{b} \text{ with } L\mathbf{y} = \mathbf{b} \text{ and } U\mathbf{x} = \mathbf{y}$$

Although the factorization is very computationally expensive, the operations to solve this last system are referred to as forward and backward substitutions respectively, these substitutions being seen as trivial [5]. Such methods can be advantageous when the matrix A does not change, as the factorization can be carried out only once, allowing the solution step to reduce to both substitutions.

Cholesky Factorization

If the matrix A is symmetric and positive definite, it can be factorized so that, $A = LL^T$, where L is the lower triangular matrix and L^T is the transpose of L. With respect to full LU factorization, Cholesky factorization takes advantage of the symmetry and thus involves less computation and memory requirements. In this case the sytem is solved as:

$$A\mathbf{x} = \mathbf{b} \Rightarrow LL^T\mathbf{x} = \mathbf{b} \text{ where } L\mathbf{y} = \mathbf{b} \text{ and } L^T\mathbf{x} = \mathbf{y}$$

3.3.2 Iterative methods

Contrary to direct methods, iterative methods require fewer operations and memory demand. However, they suffer from the ill-conditioned matrices coming from the discretized matrices, meaning that they can lead to large iteration numbers or even diverge. To deal with such a situation and improve its performance, preconditioning techniques are used [11, 79].

Let us define the $\mathbf{r}(\mathbf{x})$ the residual of Equation (3.1) as:

$$\mathbf{r}(\mathbf{x}) := \mathbf{b} - A\mathbf{x} \tag{3.15}$$

Then, iterative solvers search an approximation of the exact solution $\bar{\mathbf{x}}$, of the system defined in Equation (3.1) through a sequence of approximate solutions to make $\mathbf{r}(\mathbf{x}) \rightarrow 0$. And for the exact solution we have that $\mathbf{r}(\bar{\mathbf{x}}) = 0$.

Also, we define the error function as:

$$\mathbf{e}(\mathbf{x}) = \bar{\mathbf{x}} - \mathbf{x} \tag{3.16}$$

Finally, the iterative method is stopped when the norm of the residual is below a given criterion.

In this thesis we will present two types of iterative methods, these are:

- Stationary methods.
- Krylov subspace methods.

For both methods we present an overview of the most relevant ones inside

the context of this thesis. For more details about iterative methods see for example [50, 94, 99] in case of stationary methods and [43, 84] for Krylov subspace methods.

Stationary Methods

Stationary methods find the solution to a linear system finding the stationary point (fixed point) of a fixed point iteration given by:

$$\mathbf{x}^{k+1} = \mathrm{F}(\mathbf{x}^k) \tag{3.17}$$

If the system solved is like the one in Equation (3.1) then the fixed point iteration chosen is in such a way that the mapping F is affine, this means that if X and Y are affine spaces, then every affine transformation $\mathrm{F} : \mathrm{X} \to \mathrm{Y}$ is of the form $\mathbf{x}^{k+1} = \mathrm{G}\mathbf{x}^k + \mathbf{h}$, where G and $\mathbf{h}$ are a matrix and a vector that do not depend on the iteration k. This fixed point iteration then will generate a succession of vectors $\{\mathbf{x}^0, \mathbf{x}^1, ..., \mathbf{x}^k\}$ meant to converge to the solution of the linear system (3.1), or what is the same:

$$\lim_{k\to\infty} \mathbf{x}^k = \bar{\mathbf{x}} \tag{3.18}$$

With this definition, any such fixed point iteration of the matrix A can be written by 'splitting' the matrix A in the following way:

$$A = M - R \tag{3.19}$$

where M is a non-singular matrix known as the preconditioning matrix and R is a well-defined matrix.

Then Equation (3.1) can be re-written as:

$$M\mathbf{x} = R\mathbf{x} + b \Rightarrow \mathbf{x} = \mathbf{x} + M^{-1}(\mathbf{b} - A\mathbf{x}) \tag{3.20}$$

Starting with an initial guess $\mathbf{x}^0$, iteratively a new guess will be obtained in the following way:

$$\mathbf{x}^{k+1} = M^{-1}(R\mathbf{x}^k + \mathbf{b}) = \mathbf{x}^k + M^{-1}(\mathbf{b} - A\mathbf{x}^k) \tag{3.21}$$

Finally defining $G = M^{-1}R$ and $\mathbf{h} = M^{-1}\mathbf{b}$ (3.21) can be written as:

$$\mathbf{x}^{k+1} = G\mathbf{x}^k + \mathbf{h} \tag{3.22}$$

which is no more than the affine transformation defined above.

Several stationary methods exist, depending on the how the matrix M is chosen in Equation (3.21). Some of the most common methods found in the literature are the following:

Richardson Iteration

In this case the matrix M is chosen as the identity, and thus (3.21) is written as:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + (\mathbf{b} - A\mathbf{x}^k) \tag{3.23}$$

or written component by component:

$$x_i^{k+1} = x_i^k - \sum_{j=1}^{n} a_{ij}x_j^k + b_i, \quad i = 1, ..., n \quad k = 0, 1, .. \tag{3.24}$$

Jacobi

Considering the splitting of the matrix in an additive way:

$$A = L + D + U \tag{3.25}$$

where L and U are the strict lower and upper triangular parts of the matrix A, and D is the diagonal of A.
Then in the case of the Jacobi method $M = D$ so that:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + D^{-1}(\mathbf{b} - A\mathbf{x}^k) \tag{3.26}$$

This can be re-written as:

$$D\mathbf{x}^{k+1} = D\mathbf{x}^k + (\mathbf{b} - A\mathbf{x}^k) \tag{3.27}$$

and using the additive decomposition of A described above it is obtained:

$$\mathbf{x}^{k+1} = D^{-1}(\mathbf{b} - L\mathbf{x}^k - U\mathbf{x}^k) \tag{3.28}$$

and written component by component from Equation (3.28):

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^k - \sum_{j=i+1}^{n} a_{ij} x_j^k \right], \quad i = 1, ..., n \quad k = 0, 1, .. \tag{3.29}$$

Gauss-Seidel

Taking the matrix splitting defined in Equation (3.25), M will be taken as $M = D + L$, then considering Equation (3.21):

$$\mathbf{x}^{k+1} = \mathbf{x}^k + (D + L)^{-1}(\mathbf{b} - A\mathbf{x}^k) \tag{3.30}$$

and as it happened with the Jacobi method this can be re-written as:

$$(D + L)\mathbf{x}^{k+1} = (D + L)\mathbf{x}^k + (\mathbf{b} - A\mathbf{x}^k) \tag{3.31}$$

Then taking the decomposition of A given in (3.25):

$$(D + A)\mathbf{x}^{k+1} = (\mathbf{b} - U\mathbf{x}^k) \tag{3.32}$$

$$D\mathbf{x}^{k+1} = (\mathbf{b} - L\mathbf{x}^{k+1} - U\mathbf{x}^k) \tag{3.33}$$

Obtaining:

$$\mathbf{x}^{k+1} = D^{-1}(\mathbf{b} - L\mathbf{x}^{k+1} - U\mathbf{x}^k) \tag{3.34}$$

Note that if $U = 0$, then $A = L + D$, that is, if the matrix is lower triangular we have:

$$D\mathbf{x}^{k+1} = (\mathbf{b} - L\mathbf{x}^{k+1} - U\mathbf{x}^k) \tag{3.35}$$

$$(D + L)\mathbf{x}^{k+1} = \mathbf{b} \tag{3.36}$$

$$A\mathbf{x}^{k+1} = \mathbf{b} \tag{3.37}$$

Implying that the solution is obtained in one iteration. In this work, we will devise a renumbering strategy to make the matrix lower triangular in some asymptotic cases to make the Gauss-Seidel algorithm an efficient solver or preconditioner.

Finally if (3.34) is written component by component:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right], \quad i = 1, ..., n \quad k = 0, 1, .. \tag{3.38}$$

Here the components x_i are updated successively. Note that the sequence obtained is thus dependent on the ordering of the unknown. This property will be used to design suitable preconditioners in chapter 4.

Krylov Subspace Methods

Another approach for solving Equation (3.1) is to use Krylov subspace methods. These solve a minimization problem in a given subspace generating a sequence of approximate solutions. In general, Krylov subspace methods only need few matrix-vector products at each iteration. Also, a small number of other operations such as vector updates or dot products are required [8], meaning that low memory requirements are needed to solve a problem. For this reason, Krylov subspace methods are retained to carry out most of the simulations of this thesis.

Given the linear system of equations defined in Equation (3.1), an initial guess $\mathbf{x}^0$ and the subspaces K_k and L_k of dimension k, an approximation $\mathbf{x}^k$ can be defined by the following conditions:

$$\mathbf{x}^k - \mathbf{x}^0 \in \mathrm{K}_k$$

$$\mathbf{r}^k \perp \mathrm{L}_k$$

where $\mathbf{r}^k$ is the residual defined in Equation (3.15) at iteration k and the Krylov subspace K_k is defined as:

$$\mathrm{K}_k = \mathcal{K}_k(A, \mathbf{r}^0) = span\{\mathbf{r}^0, A\mathbf{r}^0, A^2\mathbf{r}^0, ..A^{k-1}\mathbf{r}^0\} \tag{3.39}$$

The choice of L_k depends on the type of Krylov subspace used in each case. This choice will be based on the properties of the matrix A. For example, we know that the diffusion equation leads to a symmetric positive definite (SPD) matrix, in this case the Conjugate Gradient (CG) is the appropriate method to solve the linear system [51, 82]. In Section 2.3 we also proved that the convective part of the advection-diffusion equation is non-symmetric. This means that in problems which present diffusion and convetion or only convection will lead to non-symmetric matrices. In these cases, other Krylov subspace methods have been proposed, such as the Generalized Minimal Residual (GMRES), the Bi-Conjugate Gradient or the Bi-Conjugate Gradient Stabilized [32, 78, 79]. Although this thesis focuses on strong convection problems, meaning that we will use the GMRES and BiCGStab methods in all our tests, the CG, due to its simplicity, is introduced to have an overview of how the convergence rate changes with the condition number of the matrix.

Conjugate Gradient

The conjugate gradient method is used to solve linear systems with SPD matrices and L_k is given by, $\mathrm{L}_k = \mathcal{K}_k(A, \mathbf{r}^0)$. This method minimizes the norm of the error at each iteration over the Krylov subspace. If the matrix A is symmetric and positive definite, solving Equation (3.1) is the same as minimizing the quadratic form [82]:

$$\mathbf{f}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b} \;\Rightarrow \nabla\mathbf{f}(\mathbf{x}) = \frac{1}{2}A^T\mathbf{x} + \frac{1}{2}A\mathbf{x} - \mathbf{b}$$

$$\text{If } \; A \; \text{ is symmetric then } \Rightarrow \nabla\mathbf{f}(\mathbf{x}) = A\mathbf{x} - \mathbf{b} \tag{3.40}$$

so the gradient of the quadratic form is no more than the opposite of the residual defined in (3.15).

Comparing Equation (3.40) with the definition given in (3.15) it is easy to see that the residual is the direction in which a small change in $\mathbf{x}^k$ will cause $\mathbf{f}(\mathbf{x})$ to decrease rapidly (negative gradient of $\mathbf{f}(\mathbf{x})$).

Considering this scheme, to find the next iterate $\mathbf{x}^{k+1}$ by the CG method two methods have to be introduced briefly [82]:

- **Method of the Steepest Descent:** This methods uses a line search strategy to find the new iterate $\mathbf{x}^{k+1}$ in the direction in which $\mathbf{f}$ decreases most quickly, this is the direction opposite to $\nabla\mathbf{f}(\mathbf{x})$, which according to (3.40) is $-\nabla\mathbf{f}(\mathbf{x}) = \mathbf{b} - A\mathbf{x}$ (known as search direction), at each iteration to reach the minimum value of $\mathbf{f}(\mathbf{x})$ in this specific direction. This direction changes at each iteration, as it depends on the residual defined in Equation (3.15) (for more details see [82]).

- **Method of the Conjugate Directions:** Two vectors $\mathbf{p}^k$ and $\mathbf{p}^l$ are said to be A-orthogonal if $(\mathbf{p}^k)^T A\mathbf{p}^l = 0$. A set of A-orthogonal vectors can be generated by the conjugate Gram-Schmidt process [82]. This method then searches solutions in such a way that the search direction $\mathbf{d}^k$ at the iteration $k+1$ starting at $\mathbf{x}^k$, is taken to be A-orthogonal to all previous search directions and thus the error $\mathbf{e}^{k+1}$ is also A-orthogonal to all previous search directions. This means that after k iterations, the error will be A-orthogonal to k linearly independent vectors, and the solution will be reached, or what is the same, at each iteration, a component of the error is removed.

The **Conjugate Gradient Method** is no more than the Method of the Conjugate Directions where the search directions are computed by conjugation of the residuals $\mathbf{r}^k$. We summarize how the CG method works in Algorithm 3.1.

Algorithm 3.1: Conjugate gradient algorithm.

Given an initial guess $\mathbf{x}^0$

Compute $\mathbf{r}^0 := \mathbf{b} - A\mathbf{x}^0$

Set $\mathbf{p}^0 := \mathbf{r}^0$

repeat

$$\beta^{k+1} := \frac{(\mathbf{r}^k)^T\mathbf{r}^k}{(\mathbf{r}^{k-1})^T\mathbf{r}^{k-1}}$$

$$\mathbf{p}^{k+1} := \beta^{k+1}\mathbf{p}^k + \mathbf{r}^k$$

$$\alpha^{k+1} := \frac{(\mathbf{r}^k)^T\mathbf{r}^k}{(\mathbf{p}^{k+1})^T A\mathbf{p}^{k+1}}$$

$$\mathbf{x}^{k+1} := \mathbf{x}^k + \alpha^{k+1}\mathbf{p}^{k+1}$$

$$\mathbf{r}^{k+1} := \mathbf{r}^k + \alpha^{k+1}A\mathbf{p}^{k+1}$$

until *stopping criterion is reached*;

Convergence rate

It can be shown that the CG method can be used as a direct method and that it converges after k iterations [51]. Although in practice, we use it as an iterative method, the algorithm is stopped when the converge criterion is reached (see Algorithm 3.1). It is for this reason that calculating the convergence rate of the CG is important to know the limitations of such a method.

From [82] we know that the convergence result for the conjugate gradient is:

$$\left\|\mathbf{e}^k\right\|_A = 2\tau^k \left\|\mathbf{e}^0\right\|_A \tag{3.41}$$

where $\mathbf{e}$ is the error function defined in Equation (3.16) and $\tau := \frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}$ is referred to as the convergence rate. In Figure 3.2 we plot the convergence rate of the CG in function of the condition number $\kappa(A)$.

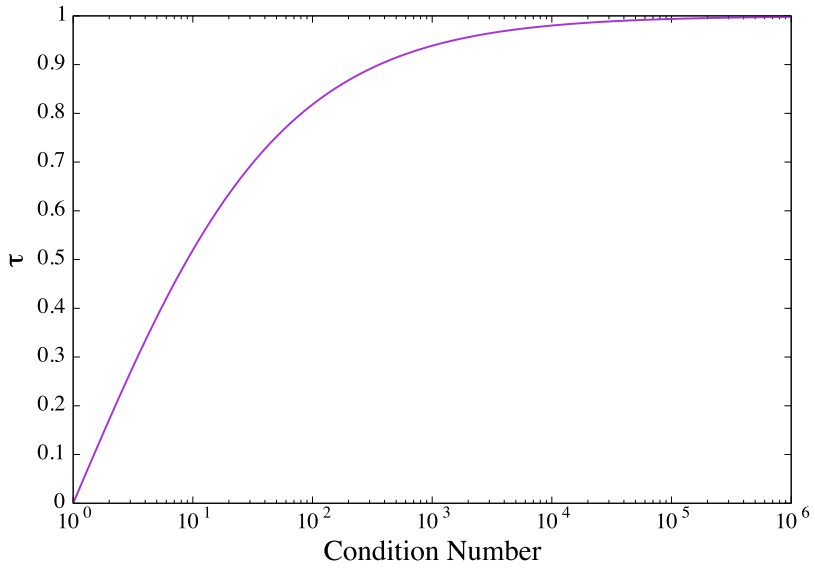


Figure 3.2: *Convergence rate of the conjugate gradient of one iteration in terms of the condition number* $\kappa(A)$.

For small condition numbers, we can see from Equation (3.41) that the CG method will converge fast, and that only a few iterations will be needed to reach the stopping criterion. The convergence of the method looses efficiency as the condition number increases, and gets stagnant for matrices with a condition numbers of an order of 10^6, where $\tau \approx 1$. Therefore we need to control the condition number for large systems, as the convergence of the CG worsens for large condition numbers.

Let now consider again the example of the Poisson equation defined in Equation (3.12). Given Equations (3.14) and (3.41) we can write τ as a function of the mesh size h. Figure 3.3 shows the tight relation between the mesh size and the rate of convergence of the CG method.

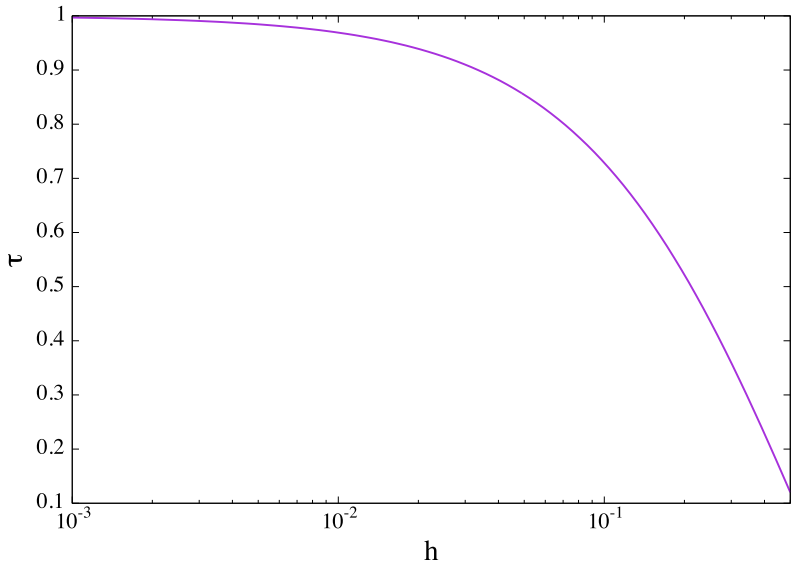


Figure 3.3: *Convergence rate of the conjugate gradient of one iteration in terms of the mesh size* h.

Generalized Minimal Residual

The Generalized Minimal Residual (GMRES) method is used to solve non-symmetric matrices, and in this case $L_k = A\mathcal{K}_k(A, \mathbf{r}^0)$ rather than minimizing the norm of the error each iteration over the Krylov subspace as CG does, GMRES minimises the Euclidean norm of the residual over this space. Following this definition the general procedure to solve the linear system in Equation (3.1) is:

- Because the matrices are non-symmetric the vectors defined for the Krylov subspace in (3.39) might almost be linearly independent, so in order to work with an orthonormal basis a 'kind of orthonormalization' similar to Gram-Schmidt procedure is applied. It is known as the Arnoldi iteration [78] and gives a set of orthonormal vectors $\{\mathbf{q}^1, \mathbf{q}^2, ...\mathbf{q}^k\}$ and the space $\mathrm{L}_k = A\mathcal{K}_k(A, \mathbf{r}^0)$. This is shown in Al-

gorithm 3.2.

Algorithm 3.2: Arnoldi iteration algorithm.

Given a vector $\mathbf{q}^1$ with $\|\mathbf{q}^1\| = 1$

for *j=1, ..., k* **do**

  $\mathbf{q}^{j+1} = A\mathbf{q}^j$

  for *i=1, ..., j* **do**

    $h^{ij} = \mathbf{q}^{j+1} - h^{ij}\mathbf{q}^i$

  $h^{j+1,j} = \|\mathbf{q}^{j+1}\|$

  $\mathbf{q}^{j+1} = \mathbf{q}^{j+1}/h^{j+1,j}$

- Once the orthonormal basis is found, an approximation of the solution $\mathbf{x}^k$ can be written as $\mathbf{x}^k = Q^k\mathbf{y}^k$, where $\mathbf{y}^k \in \mathbb{R}^k$ and Q^k is a matrix formed by $\{\mathbf{q}^1, \mathbf{q}^2, ..., \mathbf{q}^k\}$.

- The Arnoldi process will then produce the $(k+1)\times k$ upper Hessenberg matrix $\widetilde{\mathrm{H}}^k$(a detailed proof of this can be found in [78]) with:

$$AQ^k = Q^{k+1}\widetilde{H}^k \tag{3.42}$$

- With the equality in Equation (3.42) and following the details found in the bibliography in [78] the following equality between residual norms can be established:

$$\|\mathbf{b} - A\mathbf{x}^k\| = \|\beta\mathbf{e}^1 - \widetilde{H}^k\mathbf{y}^k\| \tag{3.43}$$

where $\mathbf{e}^1 = (1, 0, 0, ..., 0)^T$ is the first vector in the basis $\mathbb{R}^k$ and $\beta = \|\mathbf{b} - A\mathbf{x}^0\|$ with $\mathbf{x}^0$ the initial iterating vector.

- Once this is defined, the final solution $\mathbf{x}^k$ at the kth iterate is found taking the residual as $\mathbf{r}^k = \beta\mathbf{e}^1 - \widetilde{H}^k\mathbf{y}^k$ and minimising it until the

residual $\mathbf{r}^k$ is small enough.

In a general way the algorithm of GMRES is written as in Algorithm 3.3.

Algorithm 3.3: GMRES algorithm.

Given an initial guess $\mathbf{x}_0$

Compute $\mathbf{r}^0 := \mathbf{b} - A\mathbf{x}^0$, $\beta := \left\|\mathbf{r}^0\right\|$, and $\mathbf{q}^1 := \mathbf{r}^0/\beta$

Create $k+1$ orthogonal vectors $\{\mathbf{q}^1, \mathbf{q}^2, ..., \mathbf{q}^{k+1}\}$ as a basis for Q^{k+1} following algorithm 3.2

repeat

  Find $\mathbf{y}^k \in \mathbb{R}^k$ which minimizes $\|\mathbf{b} - A\mathbf{x}^k\|$

  Compute $\mathbf{x}^k = \mathbf{x}^0 + Q^k\mathbf{y}^k$

until *until stopping criterion is reached*;

In the GMRES method, each new iteration is more costly than the previous one, and the memory required grows as the algorithm progresses. Therefore, a new version of the GMRES called the Restarted GMRES deals with this issue by discarding after a given number of iterations all of its accumulated history and starting again from its current location [79]. In this work we will use the Restarted GMRES, that we will refer to as GMRES in the next chapters.

Bi-Conjugate Gradient (BiCG) and Bi-Conjugate Gradient Stabilized (BiCGSTAB)

The BiCG and BiCGSTAB methods are used to solve systems that are neither symmetric nor positive definite. Unlike CG, it cannot be described in terms of function minimization. As shown above, for the GMRES method, the residuals are retained orthogonally using long recurrences,

meaning that more storage will be needed. Instead, in the case of the BiCG, it replaces the orthogonal sequence of residuals by two mutually orthogonal sequences (the chosen subspaces will be $\mathrm{K}_k = \mathcal{K}_k(A, \mathbf{r}^0)$ and $\mathrm{L}_k = \mathcal{K}_k(A^T, \mathbf{r}^0)$ [38, 67]. The residuals generated within one path are not necessarily orthogonal to each other as they are within CG. However, each residual is orthogonal to all of the residual vectors generated within the other path. In other words, the residual generated within the other path on the same iteration. Similarly, the search directions are not orthogonal within one path, as they are in the CG algorithm, but each is orthogonal to all of the search directions used to generate the other path, except for the one used in the same iteration on the other path. A drawback of this method is that each step requires a matrix-by-vector product with both A and A^T [79]. The BiCGSTAB algorithm is a variant of the BiCG, developed by van der Vorst [93] and of the Conjugate Gradient Squared (CGS) developed by Sonneveld [84]. Compared to the BiCG method avoids using the transpose matrix A^T at each iteration and thus reducing one matrix-vector product. Also, it avoids the cases of irregular convergence that happen with the CGS under some circumstances (see [79, 93] for more details).

The classical algorithm for the BiCGSTAB algorithm is given by:

Algorithm 3.4: Bi-conjugate gradient stabilized algorithm.

Given an inital guess $\mathbf{x}^0$

Compute $\mathbf{r}^0 := \mathbf{b} - A\mathbf{x}^0$ such that $(\mathbf{r}^0)^T\mathbf{r}^0 \neq 0$

Set $\mathbf{p}^0 := \mathbf{r}^0$, $\bar{\mathbf{r}}^0 := \mathbf{r}^0$

repeat

$$\alpha^k := \frac{(\mathbf{r}^k)^T\bar{\mathbf{r}^0}}{(\bar{\mathbf{r}}^0)^T A\mathbf{p}^k}$$

$$\mathbf{s}^k := \mathbf{r}^k - \alpha^k A\mathbf{p}^k$$

$$\omega^k := \frac{(\mathbf{s}^k)^T A\bar{\mathbf{s}^k}}{(\mathbf{s}^k)^T A^2\mathbf{s}^k}$$

$$\mathbf{x}^{k+1} := \mathbf{x}^k + \alpha^k\mathbf{p}^k + \omega^k\mathbf{s}^k$$

$$\mathbf{r}^{k+1} := \mathbf{s}^k - \omega^k A\mathbf{a}^k$$

$$\beta^{k+1} := \frac{(\mathbf{r}^{k+1})^T\bar{\mathbf{r}}^0}{(\bar{\mathbf{r}}^k)^T\bar{\mathbf{r}}^0}\frac{\alpha^k}{\omega^k}$$

$$\mathbf{p}^{k+1} := \beta^{k+1}(\mathbf{p}^k - \omega^k A\mathbf{p}^k) + \mathbf{r}^{k+1}$$

until *stopping criterion is reached*;

3.4 Parallelization of the Iterative Solvers

This section focuses on how matrices are assembled in parallel and iterative solvers are parallelized in the context of Alya.

3.4.1 Parallel Assembly

In a parallel context, the division of the work is generally achieved through a mesh paritioning into disjoint sets of elements, referred to as subdomains.

At the computational level, these different sets of elements are treated by different processes. Each process then computes the contributions of the elements that have been assigned to it into local matrices and right-hand side vectors (local meaning specific to each subdomain).

The partitioning of the primal mesh into disjoint sets of elements defines two main categories of nodes: internal nodes are those belonging to one single subdomain; interface nodes are those that belong to more than one subdomain. Figure 3.4 shows an example of the partitioning of two subdomains Ω_1 and Ω_2. Γ is called the interface and it is defined as the set of nodes belonging to $\Omega_1 \cap \Omega_2$.

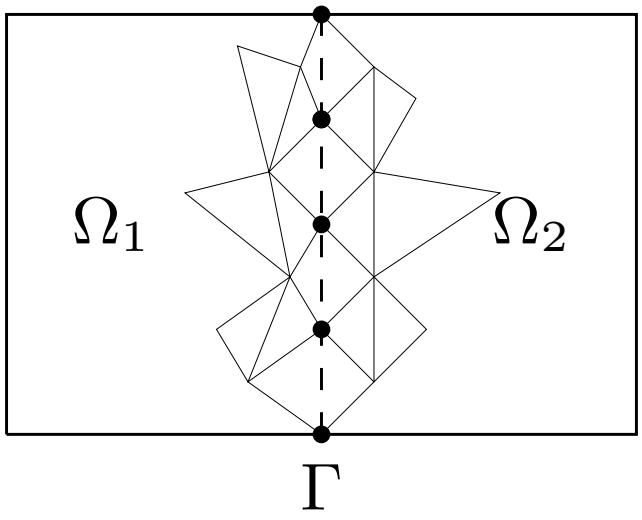


Figure 3.4: *Mesh Partition into two subdomains.*

By choosing this parallel assembly strategy, we note that the local matrices are not fully assembled for the interface unknowns, as the coefficients of the interface unknowns in the global matrix have contributions from neighboring subdomains. This has an implication on the matrix-vector product operations, as shown in the next subsection.

3.4.2 Solution with Iterative Methods

Stationary and Krylov methods require three types of operations: matrix-vector products, scalar products, and linear vector combination of two vectors. The parallelization of the matrix vector product is linked to the way the local matrices are assembled, as shown in the previous subsection. Note finally that in this thesis, the communications between the parallel processes are implemented through the MPI library [54].

- **Linear Combinations:** In this case, each process calculates the local vectors assigned to it.
- **Scalar Product:** For scalar products, each process calculates the contribution of its local vectors, and the final results consist of a reduction operation of the different local contributions. One has to take care of not duplicating the contributions of the interface nodes. There are several options as explained in [42].
- **Matrix-Vector Product:** This is the most tricky part from the implementation point of view and the one that will be discussed in detail.

  Let us focus on a two-subdomain partition, as depicted by Figure 3.4. Let us denote x_i the unknowns belonging to the interiors of subdomain $i = 1, 2$ and x_Γ the unknowns on the interface. A similar notation is employed for larger number of subdomains. Upon reordering following the partitioning, the matrix can be rewritten as:

$$A = \begin{pmatrix} A_{11} & 0 & A_{1\Gamma} \\ 0 & A_{22} & A_{2\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma\Gamma} \end{pmatrix} \tag{3.44}$$

so that a matrix vector product gives:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_\Gamma \end{pmatrix} = \begin{pmatrix} A_{11} & 0 & A_{1\Gamma} \\ 0 & A_{22} & A_{2\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_\Gamma \end{pmatrix} \tag{3.45}$$

$$= \begin{pmatrix} A_{11}x_1 + A_{1\Gamma}x_\Gamma \\ A_{22}x_2 + A_{2\Gamma}x_\Gamma \\ A_{\Gamma 1}x_1 + A_{\Gamma_2 2}x_2 + A_{\Gamma\Gamma}x_\Gamma \end{pmatrix} \tag{3.46}$$

Now let us consider two local matrices $A^{(1)}$ and $A^{(2)}$ coming from the independent contributions of subdomains 1 and 2, respectively:

$$A^{(1)} = \begin{pmatrix} A_{11} & A_{1\Gamma} \\ A_{\Gamma 1} & A_{\Gamma\Gamma}^{(1)} \end{pmatrix}, \quad A^{(2)} = \begin{pmatrix} A_{22} & A_{2\Gamma} \\ A_{\Gamma 2} & A_{\Gamma\Gamma}^{(2)} \end{pmatrix} \tag{3.47}$$

where

$$A_{\Gamma\Gamma} = A_{\Gamma\Gamma}^{(1)} + A_{\Gamma\Gamma}^{(2)}. \tag{3.48}$$

By performing two local matrix-vector products, we have

$$\begin{pmatrix} y_1 \\ y_\Gamma^{(1)} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{1\Gamma} \\ A_{\Gamma 1} & A_{\Gamma\Gamma}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_\Gamma \end{pmatrix} \tag{3.49}$$

$$\begin{pmatrix} y_2 \\ y_\Gamma^{(2)} \end{pmatrix} = \begin{pmatrix} A_{22} & A_{2\Gamma} \\ A_{\Gamma 2} & A_{\Gamma\Gamma}^{(2)} \end{pmatrix} \begin{pmatrix} x_2 \\ x_\Gamma \end{pmatrix}. \tag{3.50}$$

Here, we have denoted $y_\Gamma^{(i)}$ the different interface values obtained on

subdomains $i = 1, 2$. By using Equation 3.48, we observe by that computing

$$y_\Gamma = y_\Gamma^{(1)} + y_\Gamma^{(2)}, \tag{3.51}$$

we recover the interface value given by Equation (3.46). This means that the full matrix-vector product can be performed in two steps:

(i) Perform the local matrix-vector products as in Equations (3.49) and (3.50).

(ii) Assemble the local contribution on the interface as in Equation (3.51).

The first step involves local (subdomain) data only. The second requires an exchange of data between the processes with a common interface. This process can be easily generalized to multiple subdomains with arbitrary interfaces as explained in [96] in the context of Alya.

Note that this parallel implementation of the matrix-vector product for sparse matrices is not unique and several options exist, as explained in [42]. In general, the finite difference and finite volume methods rely on halos in order to produce full local matrices, while for the finite element method, the one presented here, appears naturally (no halo, and local matrices not fully assembled). In any case, a point-to-point communication is always necessary.

Chapter 4

Preconditioning

There's no earthly way of
knowing
Which direction they are going!
There's no knowing where
they're rowing,
or which way the river's
flowing!

Roald Dahl,
Charlie and the Chocolate
Factory

The discretization of the continuum equations in physics often leads to sparse linear systems of equations which are usually solved with iterative methods as they are cheaper than direct methods in terms of CPU-time and memory. But at the same time, iterative methods often converge slowly. To deal with this problem, instead of solving the system defined in Equation (3.1), an equivalent system, called preconditioned system, is solved. This is, Equation (3.1) is multiplied by a matrix M^{-1}, called preconditioner, that carries part of the information of the original matrix A in the sense that we seek M^{-1} such that $M^{-1}A \approx I$ (having in mind that $\kappa(I) = 1$), and taking

I as the identity matrix. Doing so, the condition number of the original matrix is improved as well as the convergence of the iterative solver used to solve the linear system of equations. The computational constraint imposed on the preconditioner is that its construction should be not too memory or time consuming.

Given an iterative solver, the design of a preconditioner is crucial to ensure fast convergence and robustness, but not at the expense of time-to-solution. A simple preconditioner (Jacobi) could a priori be disregarded because of its slow convergence. However, if one takes into account computing time, it may then be competitive compared to more complex preconditioners, such as Domain Decomposition Methods (DDM) which are much more robust and provide fast convergence in general. The choice is strongly case dependent, in terms of PDE to be solved, computational domain, partitioning and mesh size.

In this chapter, we explain the different preconditioning techniques techniques used, we review some of the different state-of-the-art preconditioners and finally we introduce what we understand by local preconditioning.

4.1 Deriving a Preconditioned System

According to the literature [11, 79], preconditioning can be done in three different ways:

(a) **Left Preconditioning**

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b} \tag{4.1}$$

This is the most 'natural' type of preconditioning as no extra step is required to compute $\mathbf{x}$ as the solution of such system is directly $\mathbf{x}$. In this case, the residual norm computed at each iteration to check the convergence of the solver, is naturally the preconditioned residual norm.

(b) **Right Preconditioning**

$$AM^{-1}\mathbf{y} = \mathbf{b} \tag{4.2}$$

and then:

$$M\mathbf{x} = \mathbf{y} \tag{4.3}$$

In this case the residual norm computed at each iteration of the solver is the non-preconditioned norm.

(c) **Left-Right (Mixed) Preconditioning:**

$$M_L^{-1}AM_R^{-1}\mathbf{y} = M_L^{-1}\mathbf{b} \tag{4.4}$$

and then:

$$M_R\mathbf{x} = \mathbf{y} \tag{4.5}$$

The type of preconditioning technique (right, left or mixed) chosen in each case, will depend on the properties of the resultant matrix obtained after the discretization of the partial differential equation. A good example to illustrate this, would be taking the advection diffusion equation (see Equation (2.3)) described in Chapter 2. As shown in Equations (2.4) and (2.5) under

high Péclet numbers the behavior will be parabolic and under low Péclet numbers the behavior will be hyperbolic. Also, as it is shown in Chapter 2, Equation (2.4) will give rise to a symmetric matrix A after its discretization, whereas Equation (2.5) will lead to a non-symmetric matrix. In the first case using the Conjugate Gradient method to solve the linear system would be the most suitable [79, 82], whereas in the second one, GMRES or BiCGSTAB would be selected [78, 79]. When choosing a preconditioner for these iterative solvers, something similar happens: in the case of symmetric matrices, when the linear system is preconditioned, the preconditioned system should preserve its symmetry, meaning that $M^{-1}A$ should be symmetric. The fact that M and A are symmetric does not guarantee that $M^{-1}A$ is symmetric. In fact, generally this is not the case. To solve this problem, one common approach is to find a Cholesky factorization of M into LL^T and solve the preconditioned system $L^{-1}AL^{-T}$ [79]. Doing this, the resultant preconditioning system, will be:

$$L^{-1}AL^{-T}\mathbf{y} = L^{-1} \text{ with } \mathbf{y} = L^T\mathbf{x}$$

which is no more than Equation (4.4) taking $L = M_L$ and $L^T = M_R$. Therefore, for symmetric systems, mixed preconditioners would be appropriate. In case of having non-symmetric matrices, as there are no restrictions, the three types of preconditioners described hereafter are suitable.

4.2 Preconditioning Techniques

Apart from the way chosen to precondition the linear system of equations (left, right or mixed), there are different methods by which the preconditioner can be obtained. In this section, the different state-of-the art preconditioning techniques will be briefly introduced, focusing more on stationary

methods rather than in domain decomposition and multigrid methods, as these are the ones that have been used in this thesis and that will be developed in more detail in Subsection 4.2.2 of this chapter.

4.2.1 Domain Decomposition Methods

In general Domain Decomposition Methods (DDM) are computationally more expensive than simple preconditioning methods and often more difficult to implement. But at the same time they tend to improve convergence of the iterative solvers. Usually, in a parallel context, domain decomposition preconditioners follow the partitioning of the mesh, meaning that the preconditioner subdomains coincide with the submeshes of the partitions. But this is not a prerequisite and such preconditioners would also make sense in sequential. Actually, Schwarz preconditioner was firstly designed at the beginning of last century to handle the solution on non-trivial geometries, out of any parallel context.

In the context of the PDE solutions, the stucture of the matrix of the linear system is linked to the underlying mesh. DDM preconditioners are based on the partitioning of this mesh into disjoint or overlapping subdomains, also named partitions, and the solution of the local problems using mainly direct solvers. Such methods have been mainly applied in the context of parallelization techniques, where one subdomain is associated to one partition, corresponding to one MPI process. This is not a requirement of the method. In fact, it is already mentioned at the previously, that this techniques were originally developed to solve problems on complex geometries, and more recently, as the mathematical basis of the Chimera method [55].

In general, the computational domain Ω is partitioned in N subsets that are either non-overlapped, Ω_i^0 or overlapped Ω_i^δ, where δ denotes the overlap partition. Then for each subdomain we define:

- A rectangular extension matrix, R_i^0 in the non-overlapping case and R_i^δ for the overlapping case, that extends by zero a vector of values defined at the mesh vertices associated with the elements contained in Ω_i^0 and Ω_i^δ respectively. The entries of R_i^0 and R_i^δ are zeros and ones and it considers in each subdomain the interior nodes and the nodes of the interface [47].

And the following subsets:

- The set of mesh vertices for the two cases considered (non-overlapping and overlapping) Γ_i^0 and Γ_i^δ that belong to the interfaces $\partial\Omega_i^0/\partial\Omega$ and $\partial\Omega_i^\delta/\partial\Omega$ respectively.

- The set of interior nodes also for the non-overlapping and overlapping subdomains I_i^0 and I_i^δ within the subdomains Ω_i^0 and Ω_i^δ.

Then, the discrete matrices coefficients obtained from a Finite Element discretization scheme in Ω_i^0 and Ω_i^δ can be obtained from the extension matrices defined earlier. We define them as A_i^0 and A_i^δ in each case:

$$A_i^0 = \begin{bmatrix} A_{I_iI_i}^0 & A_{I_i\Gamma_i}^0 \\ A_{\Gamma_iI_i}^0 & A_{\Gamma_i\Gamma_i}^0 \end{bmatrix} \text{ and } A_i^\delta = \begin{bmatrix} A_{I_iI_i}^\delta & A_{I_i\Gamma_i}^\delta \\ A_{\Gamma_iI_i}^\delta & A_{\Gamma_i\Gamma_i}^\delta \end{bmatrix} \tag{4.6}$$

According to the scheme presentented above, two kinds of DDM can be found, these are overlapping DDM and non-overlapping DDM. In this document we will give a short overview of overlapping DDM, for more details about DDM see [48].

Overlapping DDM

Often known as Schwarz methods, because of the work done by Swcharz in the XIXth century [92]. The most common ones used, as preconditioners are:

Additive Schwarz Preconditioner:

$$(M_{AS}^{\delta})^{-1} = \sum_{i=1}^{N} (\mathrm{R}_i^{\delta-1})^T (A_{I_i I_i}^{\delta})^{-1} \mathrm{R}_i^{\delta-1} \tag{4.7}$$

In this case if the original matrix is symmetric (or non-symmetric) the preconditioner will also be symmetric (or non-symmetric).

Restrictive Additive Schwarz Preconditioner

$$(M_{RAS}^{\delta})^{-1} = \sum_{i=1}^{N} \mathrm{R}_i^T (A_{I_i I_i}^{\delta})^{-1} \mathrm{R}_i^{\delta-1} \tag{4.8}$$

The Additive Schwarz (AS) preconditioner computes multiple solutions on the overlapping degrees of freedom. To remedy this, the Restrictive Additive Schwarz (RAS) preconditioner selects only one subdomain to update the interface degrees of freedom [92], through the action of a different restriction operator R_i. As a side effect, this enables one to avoid a communication of the unknown updates. The negative effect is that the resulting preconditioner is not symmetric even if the original matrix is symmetric [92].

Multiplicative Schwarz Preconditioner:

$$(M_{MULT})^{-1} = \prod_{i=N}^{1} (\mathrm{R}_i{}^{\delta-1})^T (A^{\delta}_{I_i I_i})^{-1} \mathrm{R}_i^{\delta-1} A \tag{4.9}$$

Contrary to the additive approach, that builds the solution in all the subdomains simultaneously, the multiplicative Schwarz preconditioner builds the solution of the system by switching successively through all the subdomains [83]. In general, a problem solved with a multiplicative Schwarz preconditioner will converge in less iterations, however its performance in a parallel environment can be affected by the data dependencies existent between the different subdomains [83, 92].

4.2.2 Stationary Iteration Methods

These preconditioners are especially interesting because they are computationally cheap, there are easy to implement and also can be adapted easily to the physics of the problem. In the cases hereafter D, L and U correspond to the diagonal, lower triangular and upper triangular terms of the matrix A respectively.

(i) **Jacobi**

In this case, the matrix M is taken as the diagonal of the original matrix A. Often this preconditioner is also called diagonal preconditioner.

(ii) **Gauss-Seidel Type:**

There are three different choices of the preconditioning matrix M and by which the inverse M^{-1} is found using Equation (3.34):

- **Forward Gauss-Seidel:** $M = D + L$

- **Backward Gauss-Seidel:** $M = D + U$
- **Symmetric Gauss-Seidel:** $M = (D + L)D^{-1}(D + U)$

(iii) **Bidiagonal**

The preconditioning matrix M is chosen either as $M = D + B$ or $M = D + K$, where B and K are the subdiagonal and the upper-diagonal of the original matrix A respectively.

(iv) **Tridiagonal**

Following the same notation as in the bidiagonal, in this case M is such that, $M = D + B + K$.

In the next section, we will show how efficient these preconditioners can be with an efficiently applied numbering strategy.

4.3 Mesh Renumbering for Local Preconditioning

In this section we explain how we can take advantage of the geometry (node coordinates) and topology (node connectivity) of the mesh to renumber the nodes and thus exhibit some matrix structures like the ones presented in Section 2.3. In this way, we will be able to devise two preconditioners: the classic anisotropy linelet preconditioner [85] and the streamline linelet preconditioner, which is the new one that has been developed in this thesis.

In Section 2.3 we have shown that the order in which the mesh is numbered can present some particular structures in some limited situations. Thus the stiffness matrix in an anisotropic mesh can present a tridiagonal structure or in a pure advection problem we observe independent streamlines in the direction of advection. This suggests us that the way in which a mesh is numbered can have an effect on the convergence of a given. This concept

was seen in the work of [23, 34]. In the case of [34] four different orderings are presented for the 1-dimensional convection-diffusion equation along the direction of the flow, showing how the performance of iterative solvers can be affected by the directions of the flow and by the orderings of the discrete grid points. The work of [85] presented a mesh renumbering or permutation in the direction of the mesh anisotropy in a boundary layer problem solving the pressure equation coming form the finite element discretization of the incompressible flows. Finally, [42] suggested that solving the 1-dimensional advection-diffusion equation with the appropriate boundary conditions, can be done in only one iteration using the Gauss-Seidel method. Following the work started in [42] we introduce in this chapter a mesh numbering that accounts for the advection to solve 2 and 3-dimensional problems with high advection.

Before explaining the renumbering algorithm for convection-dominated flows, a short overview of the anisotropy linelet preconditioner together with some results obtained with Alya will be given. Then the renumbering algorithm will be explained in detail together with some results that support the mathematical study first given in Chapter 2.

4.3.1 Anisotropy Linelet: Preconditioning and Results

The anisotropy linelet preconditioner is referred to as the linelet preconditioner in the literature [1]. We have added the adjective anisotropic in order to differenciate it from the streamline linelet preconditioner, which constructs linelets but in a different way and for a different purpose. The main idea of the linelet preconditioner is to construct ‘lines’ in the mesh e.g. in the parts of the mesh where the anisotropy becomes high, typical in boundary layer problems, as shown in Subsection 2.3.2 in the case of a close integra-

tion rule. In this case, the dominant terms are in the normal direction to the grid stretching. A linelet then is a group of nodes in that direction and the preconditioning matrix is built by assembling these linelets. One of the conditions to build a linelet is that a nodal point can only belong to one linelet, this means that two linelets will never cross each other. With this property, if a Laplacian is integrated in the mesh shown in Figure 4.1 and if the nodes are renumbered following the linelet structure, the corresponding preconditioning matrix is such that it has a tridiagonal structure, when anisotropy is sufficiently high, and if a close integration rule is used. This tridiagonal matrix, tends to the original matrix when the aspect ratio (anisotropy) as defined in Equation (2.51) tends to infinity, as shown in Subsection 2.3.2. For the nodes of the mesh which do not belong to any linelet the diagonal preconditioner is applied in our case [42]. We will also study the case of the open rule, although we could not witness any apparent structure, as shown in Subsection 2.3.2.

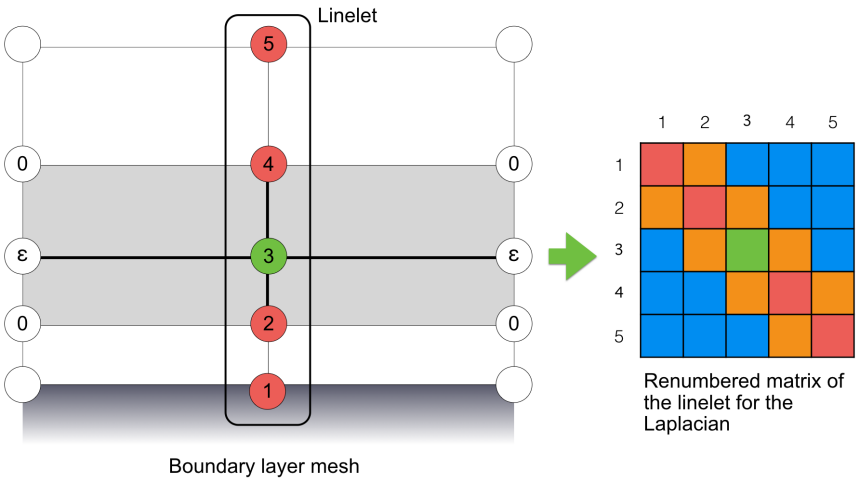


Figure 4.1: *Construction of the anisotropy linelet preconditioner (linelet ordering with its tridiagonal structure)*

Test Case

To prove that renumbering the mesh in the direction of the mesh anisotropy is efficient on more general cases, Figure 4.2 from [42], shows the convergence of the pressure equation (Poisson's equation) in the simulation of a thermal turbulent cavity. As it is illustrated in Figure 4.2 this problem does not converge with the Conjugate Gradient method when is used with the diagonal preconditioner. However, the convergence improves significatively if the linelet preconditioner is used together with a deflation preconditioning technique [7]. This shows that this renumbering works well in case of mesh anisotropy and leaves a door open to try different mesh numberings, mesh topologies and geometries with different physics.

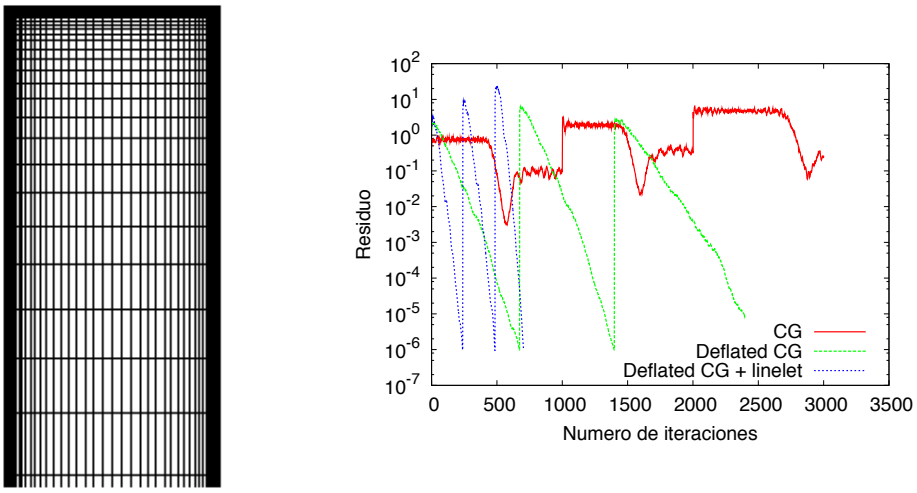


Figure 4.2: *Mesh and convergence of the pressure equation for a thermal turbulent cavity.*

4.3.2 Anisotropy Linelet: Matrix Conditioning

Chapter 3 showed that the convergence rate of the Conjugate Gradient method depends on the distribution of the eigenvalues of a matrix A, and that a good estimate of this convergence rate could be done using the condition number of the matrix. Also it is well known [64, 79] that precondition-

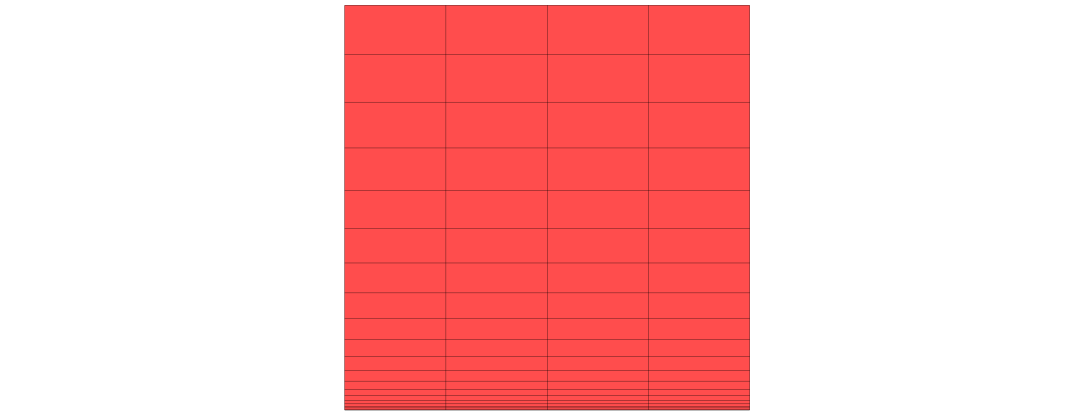
ing techniques help to accelerate the convergence of the Conjugate Gradient method. Estimating the condition number of preconditioned matrices is important to know how effective a preconditioner is. In the case of having a preconditioned matrix, its condition number is given by:

$$\kappa_p(M^{-1}A) = \|M^{-1}A\|_p \|(M^{-1}A)^{-1}\|_p \tag{4.10}$$

Case 1

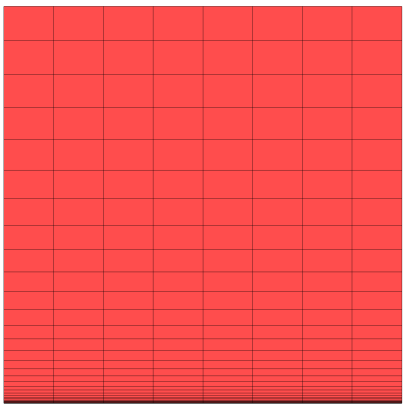
To show that the anisotropy linelet preconditioner is good, we study a case solving the Poisson equation for a temperature problem. The problem has been discretized using the finite element method and has been integrated using both open and close integration rules. To solve the linear system we use the CG method with and without the linelet preconditioner. The computational domain is a square and Dirichlet boundary conditions are imposed at the bottom wall and zero flux at the other walls. The meshes are artificially stretched in the bottom wall direction, to mimic the mesh required to capture all the phenomena in a boundary layer.

Three different meshes have been considered, e.g. Mesh 1,2,3 and shown in Figure 4.3. The aspect ratios of their first elements at the bottom wall are, respectively: 20.29, 127.50 and 6771.04. We understand by aspect ratio the relationship between $length/height$, where the $length$ is in the x dimension and $height$ in the y dimension (see Equation (2.51)). To perform the conditioning sensitivity to the aspect ratio, we have scaled in the horizontal direction the three meshes, four times. Results are shown in Figure 4.4 for the open and the close rules, respectively.



(a) *Mesh 1*

(b) *Mesh 2*



(c) *Mesh 3*

Figure 4.3: *Example meshes used to perform estimations of the condition number for the Laplacian solving the heat equation.*

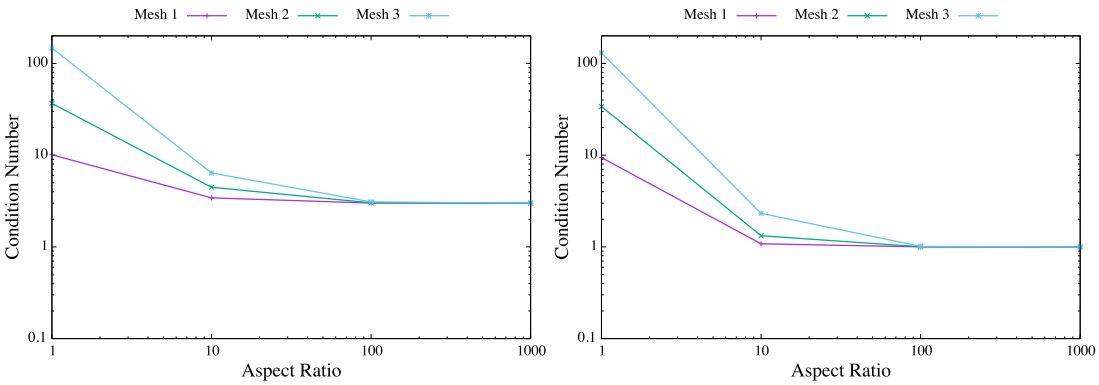


Figure 4.4: *Condition number of the preconditioned matrix with the anisotropy linelet preconditioner for different meshes, open integration rule (left) and close integration rule (right).*

Remarks

- Figure 4.4 shows the condition number of the preconditioned system $M^{-1}A$ for the meshes shown in Figure 4.3 and its four different aspect ratios. The left figure corresponds to the results of the open and the right figure to the close rule.
- When scaling in the horizontal direction the final condition number becomes almost independent of the aspect ratio of the initial mesh.
- In the case of the close rule, the condition number tends towards 1, as the matrix exhibits a tridiagonal structure as shown in Subsection 2.3.2.
- The case of the open rule is different. We observed in Subsection 2.3.2 that the structure of the matrix (removing zero coefficients) does not depend on the anisotropy. Despite this fact, Figure 4.4 left shows that the conditioning tends to a value of 3 with increasing aspect ratios. This shows that the linelet preconditioner is a good preconditioner for boundary layers disregarding the integration rule.

4.3.3 Streamline Linelet: Setup of the preconditioner

Following the idea of the anisotropy linelet renumbering, and for convection dominated problems, the matrix structure concentrates mainly in the direction of advection. As shown in Subsection 2.3.1, the main contributions for every row of a given node of the resultant matrix, apart from the diagonal term, comes from the closest neighbor in the opposite direction of the advection field, in case of the close integration rule and all the upstream nodes in case of the open rule. In these cases, the mesh

numbering is performed along the flow direction (streamwise direction). In this way, the main coefficient in each row, apart from the diagonal term happens to be the first lower-diagonal term.

To this end, we group the nodes into different non-crossing groups, corresponding to 'numerical streamlines'. The specific procedure to identify all nodes into groups is shown in Algorithm 4.1. The renumbering algorithm is based on two main ideas. First, the nodes are ordered by their velocity modules in an increasing way, starting with the inflow nodes, the interior nodes and finally the outflow nodes. This is clearly shown in Figure 4.5.

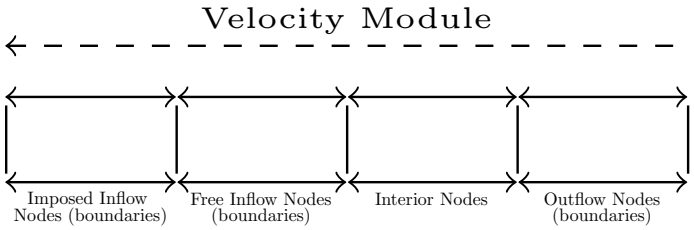


Figure 4.5: *Nodes ordered using their types and velocity modules.*

The next step consists in putting the nodes in different groups following what we call the *minimum angle criterion* achieving in this way the final ordering. This is done as it follows:

- Starting with an inflow node (e.g. starting point in Figure 4.6), the vector $\mathbf{v}_i$ (the black vectors in Figure 4.6) between this node and each of its neighbors i is computed.

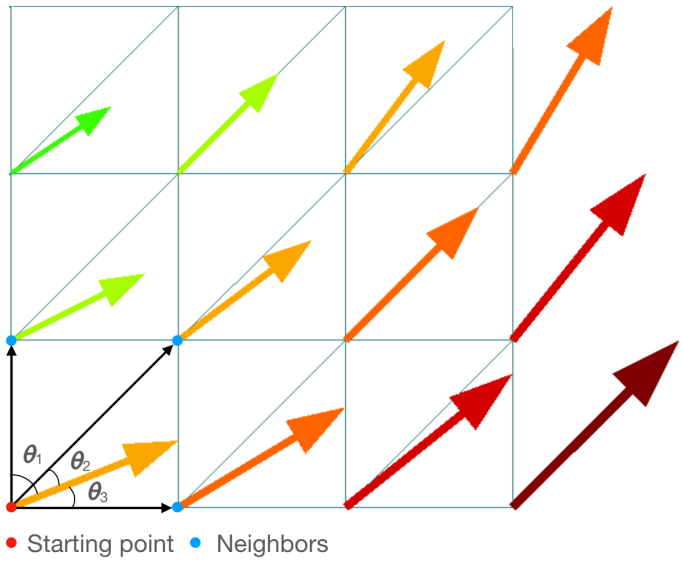


Figure 4.6: *Schematic of the streamline numbering.*

- Then, we compute the angle between these vectors and the representative velocity vector $\mathbf{u}_{ref}$ (in case of Figure 4.6, we have three angles, θ_1, θ_2 and θ_3 and the velocity vector is the one in orange starting in the starting point). The angle is computed in the following way:

$$\cos\theta_i = \frac{\mathbf{u}_{ref} \cdot \mathbf{v}_i}{\|\mathbf{u}_{ref}\| \, \|\mathbf{v}_i\|} \quad \text{with} \quad i = 1, \ldots, n \tag{4.11}$$

  where n are the number of neighbors of each node (the three blue dots in Figure 4.6).

- Discarding the nodes which have a negative or zero cosine, the next node in the group will be the one that forms the smallest angle with the velocity vector (maximum cosine), or what is the same, the one which is the closest to the direction of the advection velocity.

- This procedure is repeated recursively until no positive values of the cosine are found.

- When this happens the next node on the list is selected (as explained

in Figure 4.6), and the above process is repeated until all the nodes of the mesh are numbered.

Algorithm 4.1 summarizes the procedure. The outputs are: a list of group to which the nodes belong to (`lgrou`), a permutation array (`permu`) and the inverse permutation array (`invpr`) to perform the reordering from the new numbering to the old numbering. In this new numbering, nodes are numbered successively in terms of linelets, and, for each linelet, following the 'numerical streamline'.

Algorithm 4.1: *Streamline numbering along advection:* Construct groups following the direction of advection

Result: Permutation array `permu` (from new to old) and group list `lgrou`

`kpoin`=0

`igrou`=0

while *all nodes have not been assigned a group* **do**

  Look for next seed node `ipoin` not belonging to any group

  `kpoin` = `kpoin` + 1

  `igrou` = `igrou` + 1

  repeat

    `inext` = 0

    `cangm` = 0 **for all** *the neighbors* `jpoin` *of* `ipoin` **do**

      if `jpoin` *does not belong to any group* **then**

        `vpoin` = coord(`jpoin`) - coord(`ipoin`)

        `vrefe` = $\frac{1}{2}$ (advec(`ipoin`)+advec(`jpoin`))

        `cangl` = $\frac{\texttt{vpoin} \cdot \texttt{vrefe}}{\|\texttt{vpoin}\|\|\texttt{vrefe}\|}$

        if `cangl` $>$ `cangm` **then**

          `cangm` = `cangl`

          `inext` = `jpoin`

    if `inext` *!= 0* **then**

      `kpoin` = `kpoin` + 1

      `permu`(`kpoin`) = `inext`

      `lgrou`(`ipoin`) = `igrou`

      `ipoin` = `inext`

  until `inext = 0` *or* `inext` *is an outflow*;

Figure 4.7 shows an example of streamline linelets, constructed with a velocity field obtained from the solution of the Navier-Stokes equations for a NACA0012 geometry. The velocity vectors are shown in the (top) (left) part. The other plots show the vectors forming the linelets from node-to-node.

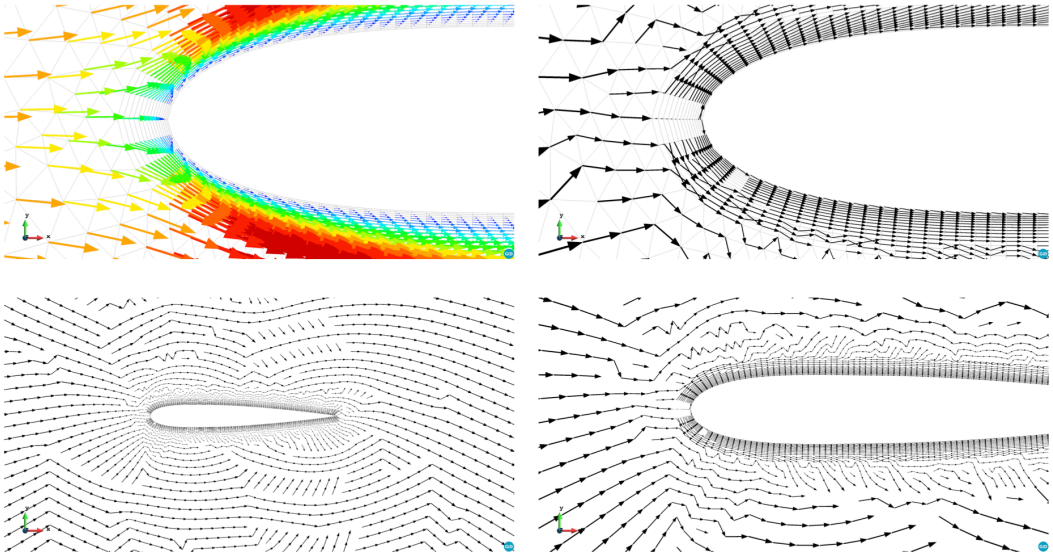


Figure 4.7: *Streamline linelets for a NACA0012. Velocity vectors used as a reference to construct the linelets (top) (left) and different views of the vectors forming the different linelets.*

Figure 4.8 shows the effect of the maximum angle cosine `cangm` on the streamline linelets. Although the quantity of linelets seems not be affected, we can observe that large cosine implies straighter linelets, as expected. The zone in red shows an example in this situation.

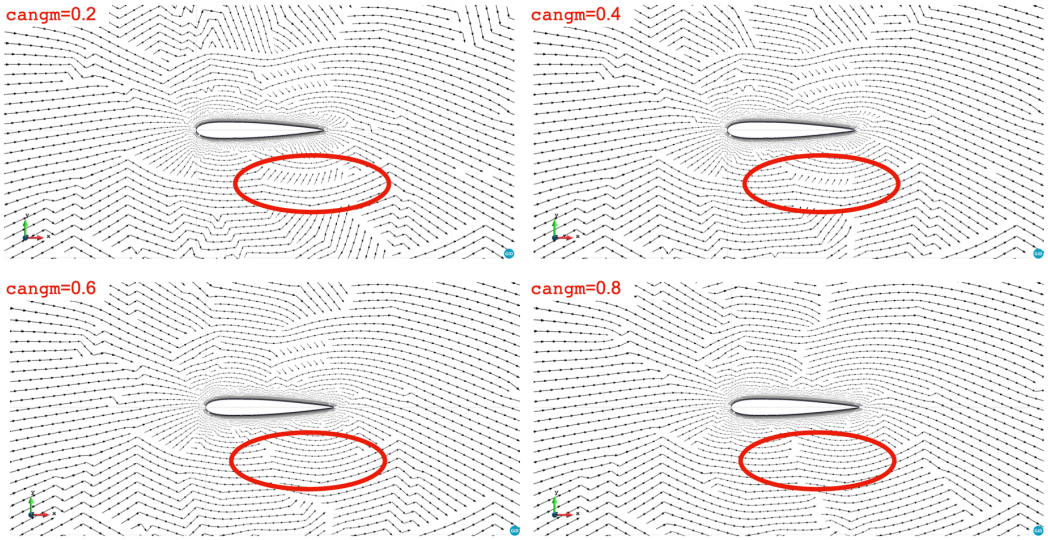


Figure 4.8: *Streamline linelets for a NACA0012 for different maximum angle cosines: 0.2 (top) (left), 0.4 (top) (right), 0.6 (bottom) (left), 0.8 (bottom) (right)* .

4.3.4 Streamline Linelet: Preconditioning

Once this renumbering is done, an appropriate preconditioning technique can be applied [26] to solve the preconditioning step:

$$M\mathbf{z} = \mathbf{r} \tag{4.12}$$

if left preconditioning is considered.

In the particular case of this thesis, the preconditioners are the Gauss-Seidel and the bidiagonal preconditioners. For which a matrix takes the diagonal and sub-diagonal entries from the permuted matrix using the permutation array coming from Algorithm 4.1. In Algorithms 4.2 and 4.3 we show how we apply this new reordering to the Gauss-Seidel and bidiagonal preconditioners respectively. The proposed preconditioners are thus based on a streamline renumbering together with a solver, namely the Gauss-Seidel

and bidiagonal solvers.

Note that in both algorithms `ia`, `ja` and `aa` are the row indices, column indices and matrix coefficients defined in Subsection 3.1.2 for the CSR format respectively.

Algorithm 4.2: *One iteration Gauss-Seidel in CSR format.*

Define `dd` in old numbering ii_{old}.

`dd`(ii_{old}) diagonal coefficient for row ii_{old}

Initial guess `z` = `0`

for all *the nodes of the mesh in the new numbering* ii_{new} **do**

  ii_{old} = `permu`(ii_{new})

  `z`(ii_{old}) = `r`(ii_{old})

  for `izdom` = `ia`(ii_{old}), `ia`$(ii_{old}+1)$-1 **do**

    `col` = `ja`(`izdom`)

    if `invpr`(`ja`) $\neq ii_{new}$ **then**

      `z`(ii_{old}) = `z`(ii_{old}) - aa(`izdom`) * `z`(`col`)

  `z`(ii_{old}) = `z`(ii_{old})/`dd`(ii_{old})

Algorithm 4.3: *One iteration Bidiagonal in CSR format.*

Define `dd`, `ll` and `ka` in old numbering ii_{old}.

`dd`(ii_{old}) diagonal coefficient for row ii_{old}

`ll`(ii_{old}) is the lower-diagonal column for row ii_{old}

`ka`(ii_{old}) is the matrix position of the lower-diagonal coefficient

for all *the interior nodes of the mesh in the new numbering* ii_{new} **do**

  ii_{old} = `permu`(ii_{new})

  `col` = `ll`(ii_{old})

  `izdom` = `ka`(ii_{old})

  `z`(ii_{old}) = (`r`(ii_{old}) - aa(`izdom`) * `z`(`col`))/`dd`(ii_{old})

We note that by doing one single Gauss-Seidel iteration in Algorithm 4.2

and considering a zero initial condition, Equation (3.31) reads

$$\mathbf{x}^1 = (D + L)^{-1}\mathbf{r} \tag{4.13}$$

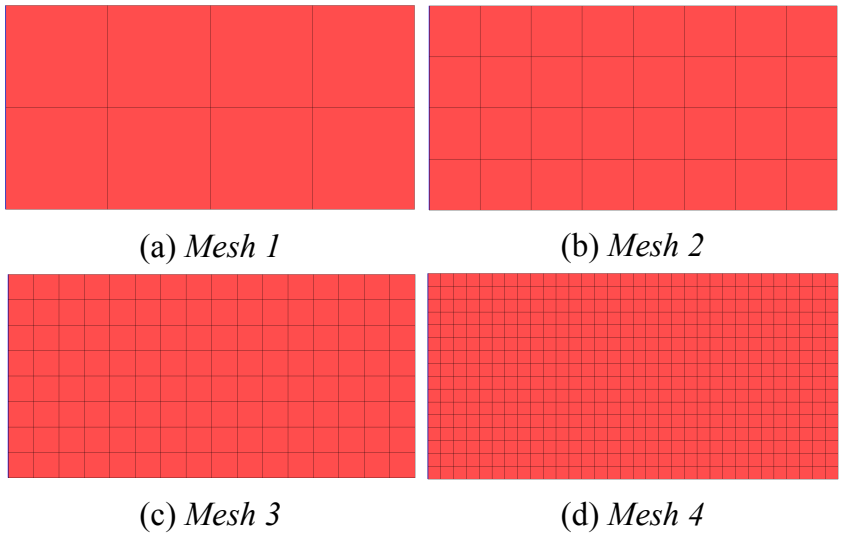
which is equivalent to approximating the matrix A by its lower part and selecting $M = L + D$ in the new renumbering. However, we prefer to maintain the Gauss-Seidel terminology to envisage the possibility of carrying out several iterations in the future.

4.3.5 Streamline Linelet: Matrix Conditioning

Two more cases are presented to show how the condition number decreases when the matrix is preconditioned appropriately. In the first case we show the ideal case, similar to the one considered in Subsection 2.3.1. The second case shows the same for a more challenging problem, where we prove again that the condition reduces after the matrix is preconditioned.

Case 1

For this first problem we consider pure advection equation. The problem has been discretized using the finite element method and integrated using both integration rules. Four different meshes have been considered, these are shown in Figure 4.9. The computational domain is a rectangle, where Dirichlet boundary conditions are imposed at the left wall and zero flux at the other walls. The results are shown in Figure 4.10.



(a) *Mesh 1* (b) *Mesh 2*

(c) *Mesh 3* (d) *Mesh 4*

Figure 4.9: *Example meshes for Case 1 to perform the estimations of the condition number for the convection equation with size* h, $h/2$, $h/4$ *and* $h/8$, *respectively.*

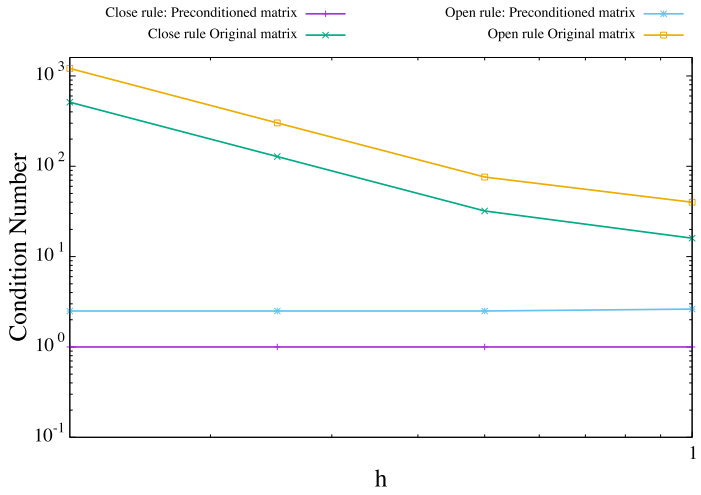


Figure 4.10: *Condition number of the unpreconditioned matrix and preconditioned matrix using the streamline linelet preconditioner.*

Remarks

- Figure 4.10 shows the condition number of the non-preconditioned system and of the preconditioned system for the meshes shown in Figure 4.9 and for both integration rules.

- In the case of the non-preconditioned system (for both integration rules), the condition number of the matrix increases as the mesh size h decreases. In the case of the preconditioned system and for both rules, the condition number does not depend on the size of the matrix.
- As we observe in Figure 4.10 using the close rule gives better results than using the open rule. It can be explained by the fact that the close rules have independent streamlines and there is only a dependence to the previous node as showed in Subsection 4.3.3.
- If the preconditioned system uses the close integration rule, the condition number, independently of the mesh size, is equal to one, as the stencil is one-dimensional in the direction of the flow and the matrix has a lower diagonal structure.
- In the case of the open rule, the condition number cannot be one because, apart from the upstream dependence, there is a cross-flow dependence between the nodes (see in Figure 2.9) which cannot be resolved with one single Gauss-Seidel iteration.

Case 2

We will now compute the condition number of the matrix coming from a more complex case. The reason for choosing this problem is to show that the condition number in this 'not-so-ideal case' (advection is not perfectly aligned with the mesh edges), we can also obtain a gain in the condition number. In this case, we will compute the condition numbers of the preconditioned matrix with the streamline linelet preconditioner (Gauss-Seidel and bidiagonal) and compare them to those of the diagonal preconditioned matrix and non-preconditioned matrix.

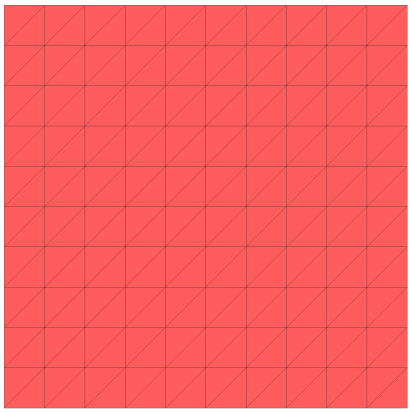
In this example, we consider a rotating advection field centered in $(0.5, 0.5)$ in a square domain $\Omega = (0,1) \times (0,1)$ such that $\mathbf{a} = (0.5 - y, x - 0.5)$ and solve the following problem:

$$\mathbf{a} \cdot \nabla u - \frac{1}{Pe} \Delta u = 0 \ \text{ in } \ \Omega = (0,1) \times (0,1)$$

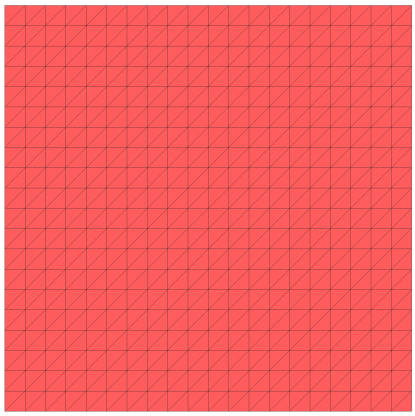
where we choose $Pe = 10^4$ and the following Dirichlet boundary conditions:

$$\begin{cases} u &= 1 \ \text{ at } \ 0 < x < 1 \ \text{ and } \ y = 0 \\ u &= 2 \ \text{ elsewhere} \end{cases}$$

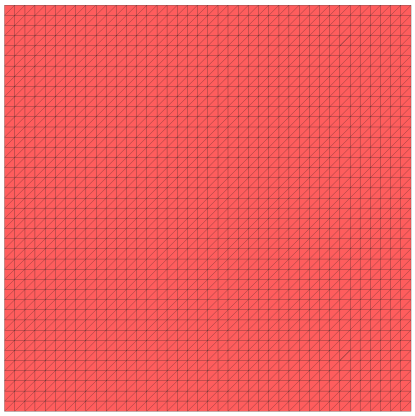
The test was performed on three different triangular meshes of 200, 800 and 3200 elements shown in Figure 4.11 respectively.



(a) *Mesh 1*



(b) *Mesh 2*



(c) *Mesh 3*

Figure 4.11: *Example meshes to estimate the condition number for the convection-diffusion equation with a rotating advection field.*

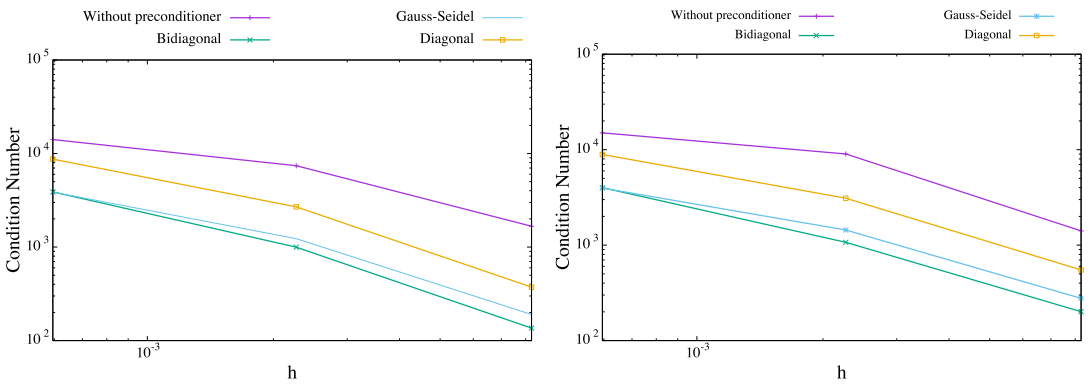


Figure 4.12: *Condition number of the unpreconditioned matrix and preconditioned matrix using the streamline linelet (Gauss-Seidel and bidiagonal) and diagonal preconditioners: close integration rule (left) and open integration rule (right).*

Remarks

- Figure 4.12 shows the condition number for matrices A and $M^{-1}A$ for a close (left) and open (right) integration rules. In both cases we have calculated the condition number for three different types of preconditioners, these are the diagonal, the streamline linelet (Gauss-Seidel and bidiagonal).
- The close rule gives slightly better results. Contrary to what happened with the ideal case,we do not have a perfect structure in this case due to the effect of diffusion and the missalignment of the advection with the mesh.
- As we observe in both figures, the condition number depends on the mesh size, and it increases as h decreases. Also we note, that if we renumber the mesh appropriately along the advection velocity and apply the Gauss-Seidel or bidiagonal preconditioners, the condition number is reduced by one order of magnitude, meaning that the convergence is accelerated at the time of solving the linear system of equations.

4.4 Parallelization of the Preconditioners

The parallelization of a preconditioning step $M\mathbf{z} = \mathbf{r}$ can result in a complex implementation, or in some cases even makes it serial. Considering the parallel implementation used in this work (and described in Section 3.4), the minimum required for the preconditioning is that after solving $M\mathbf{z} = \mathbf{r}$, interface values of $\mathbf{z}$ are the same on all neighbouring subdomains. For instance, the diagonal preconditioner is easy to compute, as the coefficient

of the diagonal matrix can be computed through the reduction of all the subdomain contributions on the interface nodes. The same strategy as the one described in Subsection 3.4.2 can then be used. The parallelization of the Gauss-Seidel method would make serial the preconditioning step. In fact, the values of $\mathbf{z}$ are obtained successively along the streamlines, so that downstream subdomains should wait for their upstream neighbours before solving their local solution to $M\mathbf{z} = \mathbf{r}$. Finally, the parallelization of the anisotropy linelet preconditioner is possible, but not easy to implement [74]. One workaround consists in preventing the mesh partitioning from cutting the linelet [85]. In this work, we have chosen an easier option for all the preconditioners, which consists of imposing a diagonal preconditioner on the interface nodes. Then, each preconditioner is only applied locally. The efficiency of the preconditioner thus depends on both the partitioning topology and the number of subdomains. This has obviously an impact on the performance of the preconditioning step, but greatly simplifies the implementation and limits the communication between neighbouring subdomains.

4.5 Convergence Results for the Streamline Linelet Solver and Preconditioner

This section tackles two different problems. In the first example, we will study the convergence of the Gauss-Seidel solver with streamline renumbering. In the second example, we will apply the streamline linelet preconditioner (Gauss-Seidel and bidiagonal) and study the convergence of several iterative solvers. For all the results we have chosen a close integration rule.

For the sake of clarity, before continuing with the results we will introduce

the following notation for this section:

Streamline linelet used with the Gauss-Seidel: Gauss-Seidel

Streamline linelet used with the bidiagonal: Bidiagonal

4.5.1 Simple Case: Gauss-Seidel as a Solver

In order to confirm the good behavior of the streamline renumbering, two meshes of 3 elements and 8 nodes are considered (see Figure 4.13). In both of them we solve the pure advection equation with a horizontal advection field with constant velocity $\mathbf{a} = (a, 0)$. We number the two meshes differently, the one at left-hand side of Figure 4.13 being numbered arbitrarily, whereas the one at the right-hand side of the figure has been numbered following the direction of the advection velocity, which in this particular case is along the x axis. In addition we show in Figure 4.14 the assembled matrices coming from the two meshes. The blue squares in the matrix are the zero coefficients, the orange ones the non-zero coefficients and the black ones correspond to the boundary conditions that, for this example, correspond to Dirichlet boundary conditions applied to nodes 1 and 5.

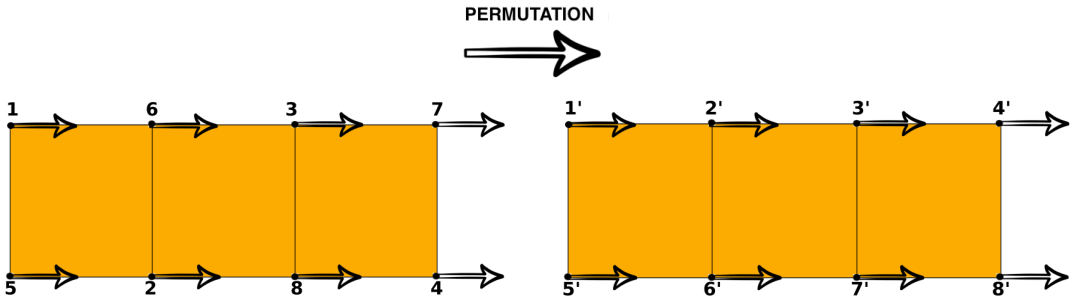


Figure 4.13: *Simple Mesh with arbitrary numbering (left) and numbered along advection (right)*

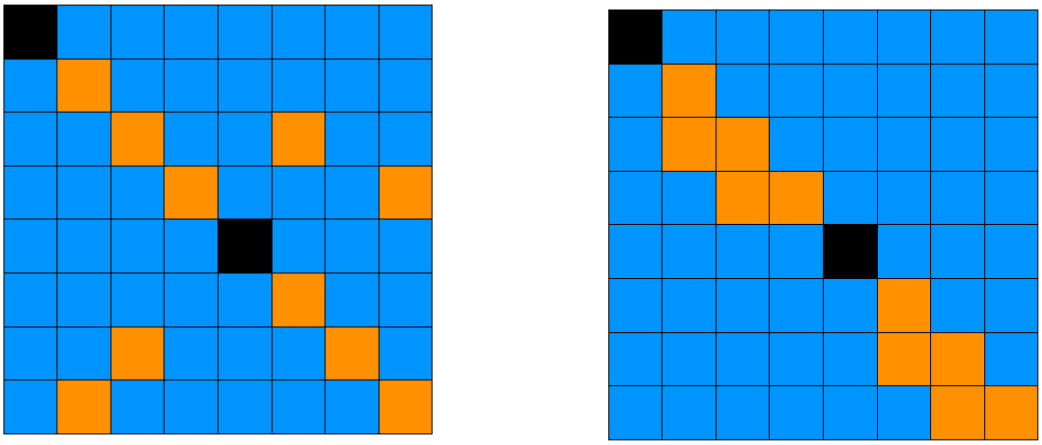


Figure 4.14: *Resultant matrices that correspond to the mesh showed in Figure 4.13 before renumbering (left) and after renumbering (right)*

We then apply the Gauss-Seidel Algorithm 3.38 to these two cases and count the number of iterations to reach convergence. The results of these two experiments are shown in Table 4.1.

	Number of iterations
Arbitrary numbering	3
Numbering along the advection	1

Table 4.1: *Comparison of the convergence between the mesh numbered randomly and the one numbered along the direction of advection.*

Remarks

- Without diffusion, the problem is hyperbolic and the information 'propagates' only in the direction of the advection. If we put this in terms of what we observe in the matrices of Figure 4.14, it means that for every row of the matrix, a part from the coefficients in the diagonal, there is only one coefficient different from zero, located down-

stream in terms of position in the mesh. This suggests that we can find the solution node to node starting from a node with a Dirichlet boundary condition, this is what the Gauss-Seidel method does [42]. If the numbering is performed in the direction of advection, then the non-zero coefficient corresponds to the previous node and we observe that in this case that we need one iteration to obtain the solution to the problem.

- The case with random numbering converges in 3 iterations, showing the impact of the renumbering even in this very simple case. Depending on the renumbering this number of iterations is likely to increase for larger meshes.

4.5.2 Swirl (Rotating Advection Field): Gauss-Seidel as a Preconditioner

We consider now the example considered in Subsection 4.3.3 where we showed the effect of preconditioning on the condition number for a non-ideal case. From the meshes considered in Figure 4.11, we consider Mesh 3, as shown in Figure 4.15, which is a 2-D mesh of 3200 elements. We will also consider a finer mesh made of 12800 elements. The figure also shows the velocity vectors and the streamline linelet, drawn as vectors from node to node.

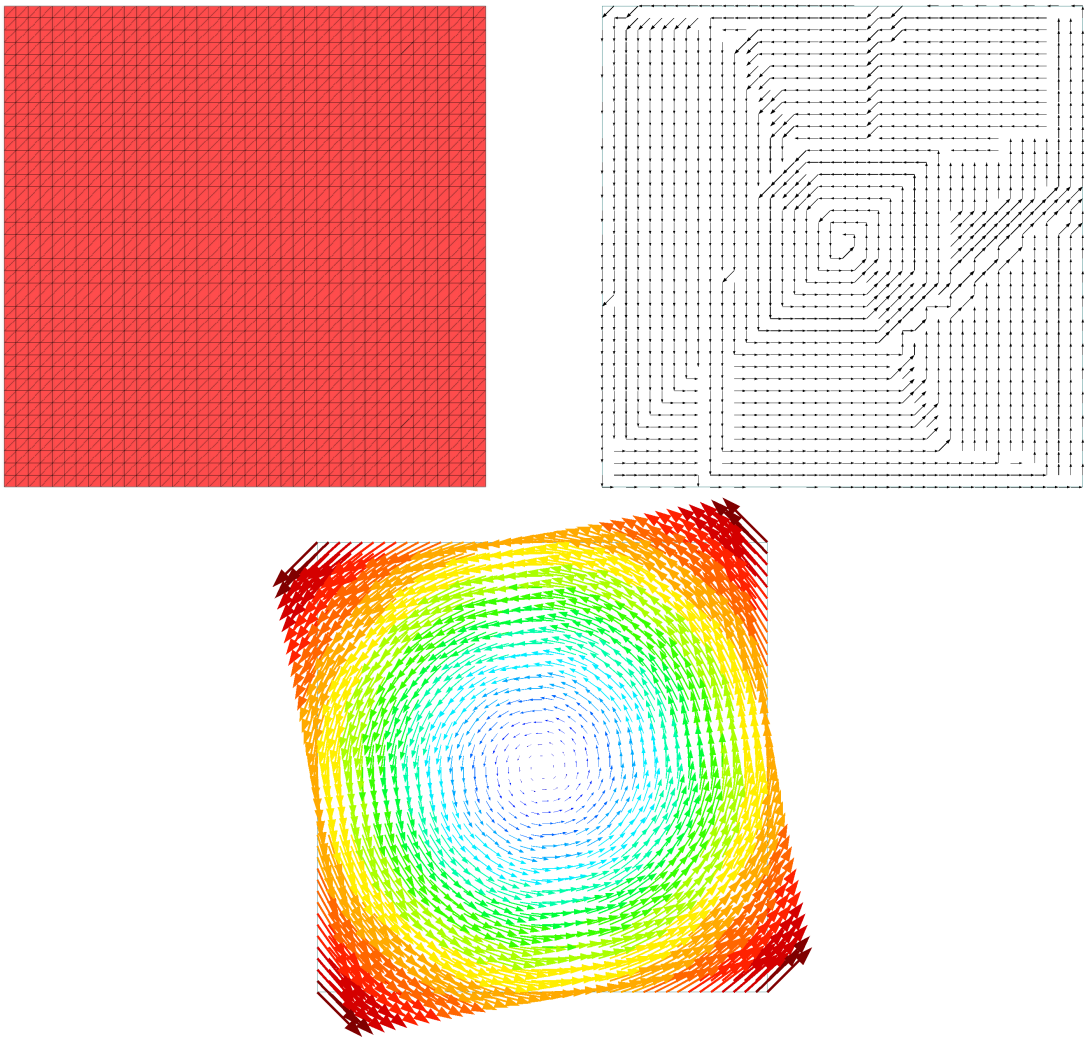


Figure 4.15: *Swirl: Mesh (top-left) and node-to-node streamline vectors computed by algorithm 4.1 (top-right) together with the velocity vectors (bottom).*

We also show in Figure 4.16 the original and renumbered matrices. Similarly as for the previous example, the dark blue parts of both matrices show the zero coefficients, and the rest of the color map, shows the distribution of the non-zero elements.

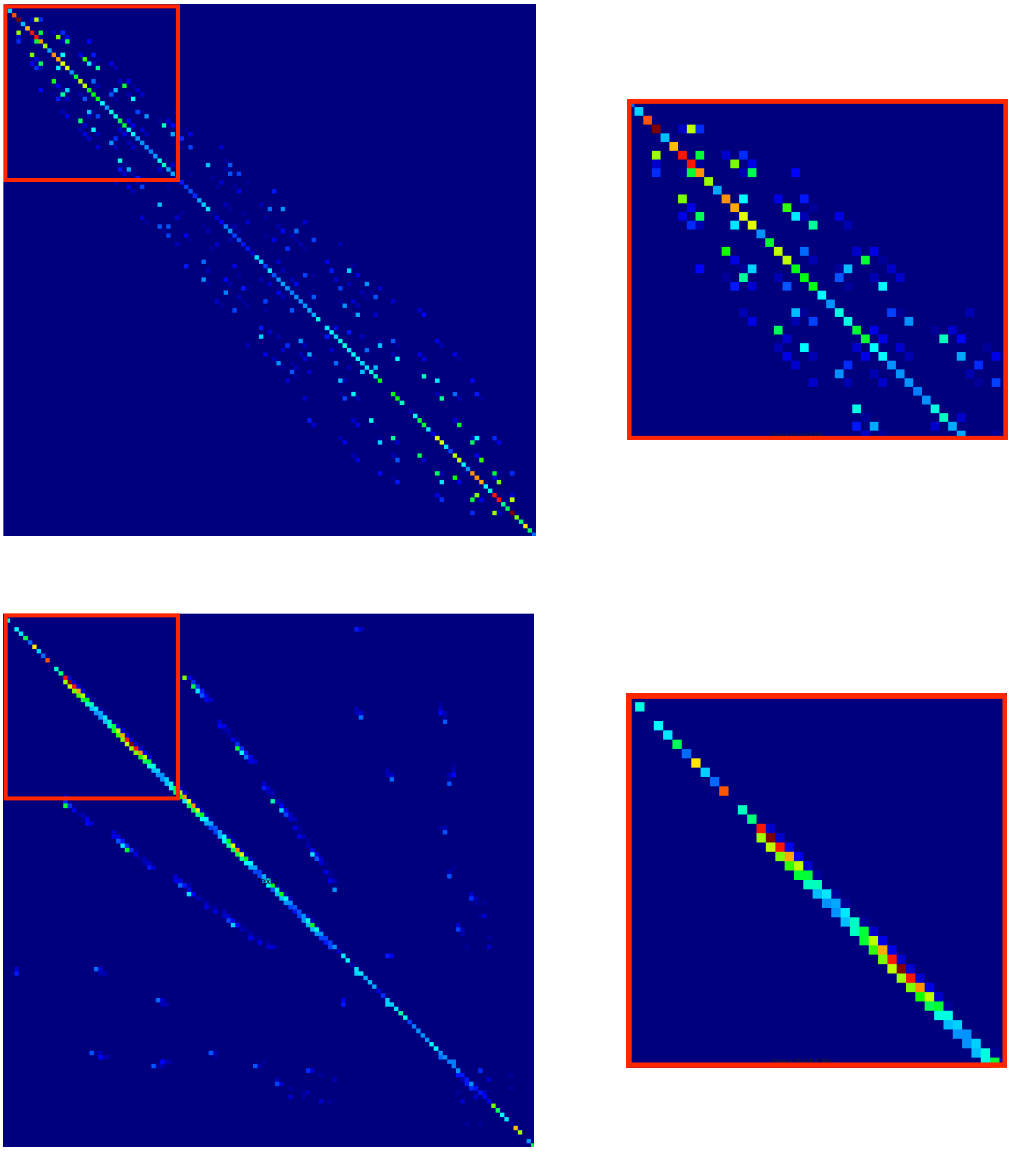


Figure 4.16: *Swirl: Original matrix (top) and renumbered matrix (bottom).*

In the ideal cases studied in Section 2.3.1 and in the previous example, the resulting matrix obtained after renumbering was lower diagonal ($U = 0$ in Equation 3.34). We observe that in the case of the streamline reordering, the greatest coefficients lie just below the diagonal, showing that the re-ordering make the structure of the matrix tend to a lower triangular structure.

Figures 4.17 and 4.18 show the results for convergence and convergence time of this case. We have carried out more tests with two different iterative

solvers, the GMRES with a Krylov dimension of 10 and BiCGSTAB. In both cases we have chosen a convergence tolerance of 10^{-12} and we have tried the diagonal, Gauss-Seidel and bidiagonal preconditioners respectively using the renumbering algorithm in all cases.

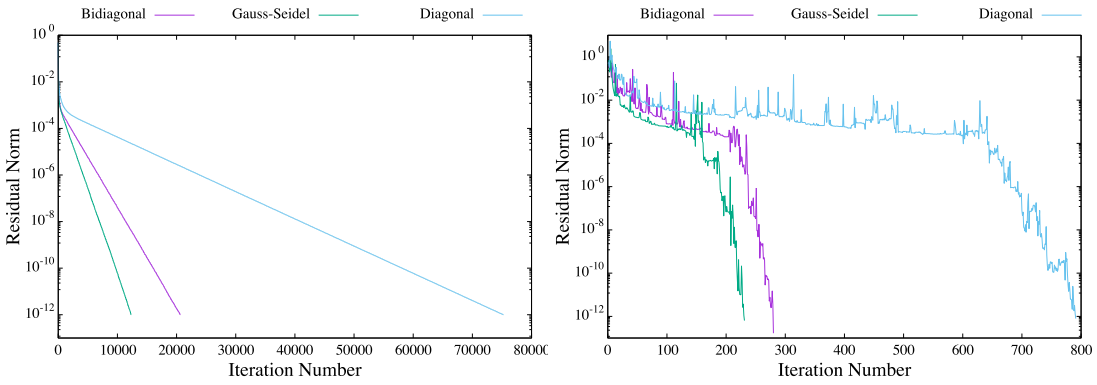


Figure 4.17: *Swirl: Results for convergence using a GMRES (left) and a BiCGSTAB (right).*

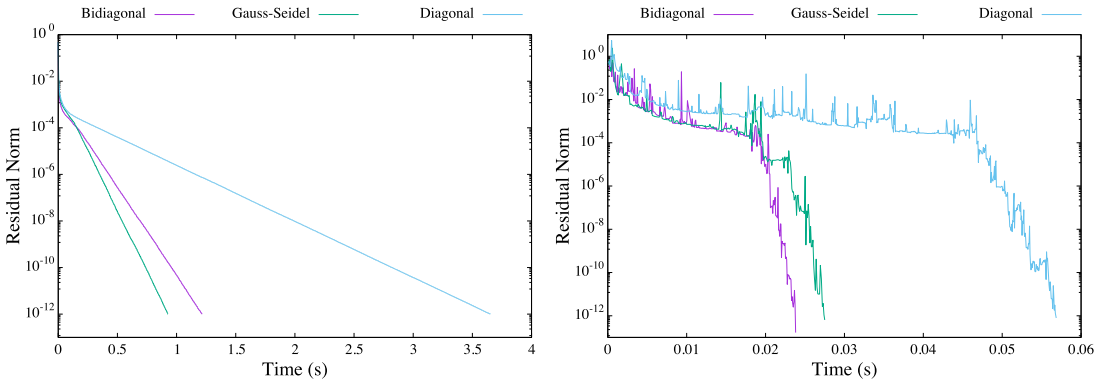


Figure 4.18: *Swirl: Results for convergence times using a GMRES (left) and a BiCGSTAB (right).*

Let us now consider the same problem, but with a triangular mesh of 12800 elements, in Figures 4.19 and 4.20 we show the convergence and convergence times using a GMRES and a BiCGSTAB in this case.

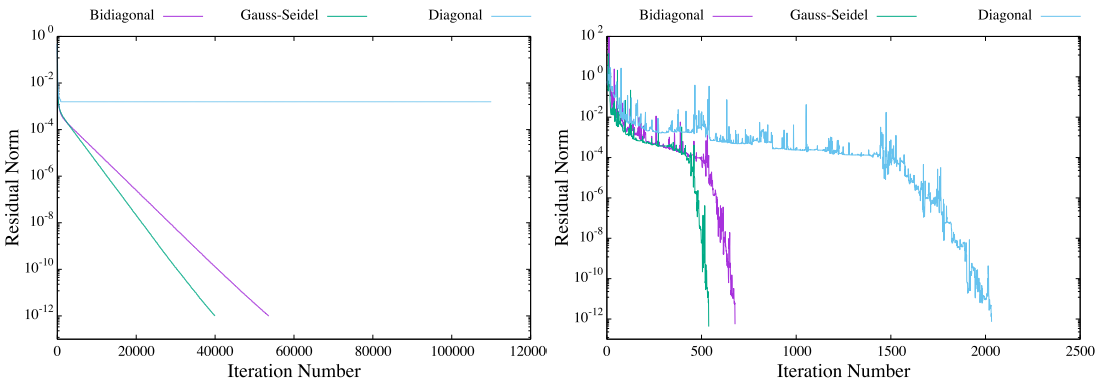


Figure 4.19: *Swirl: Results for convergence using a GMRES (left) and a BiCGSTAB (right).*

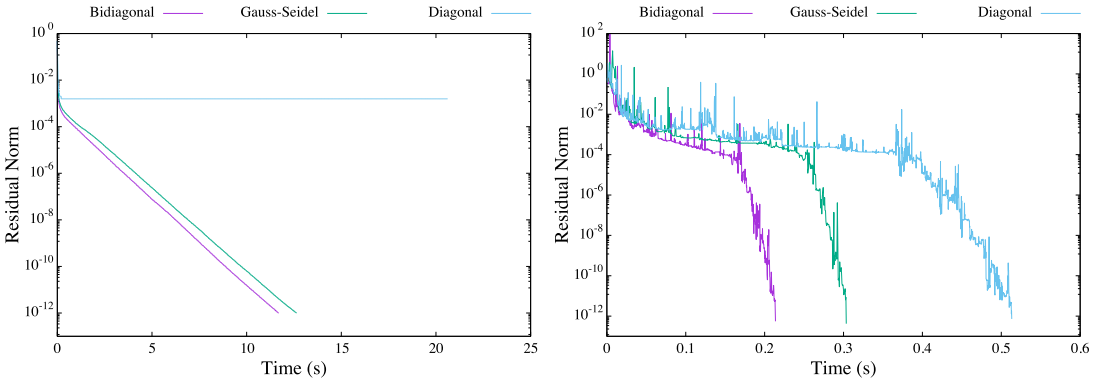


Figure 4.20: *Swirl: Results for convergence times using a GMRES (left) and a BiCGSTAB (right).*

We summarize the results obtained in Figures 4.17, 4.18, 4.19 and 4.20 in Tables 4.2 and 4.3, and we use the notation NC in the cases where the problem does not converge.

Preconditioner	**Coarse**	**Fine**
Bidiagonal	18766	48713
Gauss-Seidel	11202	36307
Diagonal	68397	NC

Preconditioner	**Coarse**	**Fine**
Bidiagonal	280	677
Gauss-Seidel	231	538
Diagonal	791	2073

Table 4.2: *Number of iterations of the different preconditioners in the two meshes considered with a GMRES (left) and a BiCGSTAB (right).*

Preconditioner	**Coarse**	**Fine**
Bidiagonal	1.21	11.6
Gauss-Seidel	0.92	12.6
Diagonal	3.65	NC

Preconditioner	**Coarse**	**Fine**
Bidiagonal	$2.3 \cdot 10^{-2}$	0.21
Gauss-Seidel	$2.8 \cdot 10^{-2}$	0.31
Diagonal	$5.6 \cdot 10^{-1}$	0.51

Table 4.3: *Convergence time (in seconds) of the different preconditioners in the two meshes considered with a GMRES (left) and a BiCGSTAB (right).*

Remarks

- Figure 4.16 shows the assembled matrix before and after the mesh renumbering is done. As we can see, the non-zero coefficients (red and green dots in Figure 4.16) lay closer and below to the diagonal, this is, the relevant coefficients of the problem are packed in the lower part of the diagonal.

- Thanks to the renumbering, we have succeded in making the matrix more lower triangular, on which the Gauss-Seidel algorithm is efficient (see remark in Subsection 3.3.2).

- Figure 4.18 and Tables 4.2 and 4.3 show the time that it takes for the problem to converge and it compares it with the three preconditioners. The fastest one, independently of the solver used is the Gauss-Seidel preconditioner for both iterative solvers. In this case, we observe that the bidiagonal preconditioner, although involving less computations than the Gauss-Seidel, does not compensate in terms of time to convergence.

- From Figures 4.19 and 4.20 and Tables 4.2 and 4.3, we confirm the robustness of the Gauss-Seidel and Bidiagonal preconditioners using the renumbering strategy. Also we observe that in this case, the prob-

lem does not converge when the diagonal preconditioner is used with a GMRES solver.

4.5.3 Temperature Transport over a Sphere

Let us consider a more complex problem, the temperature transport over a sphere. The geometry and boundary conditions are shown in Figure 4.21. We consider three meshes, coarse, middle and fine, composed of 8670, 69360 and 554880 tetrahedra, respectively. Some views are shown in Figure 4.22.

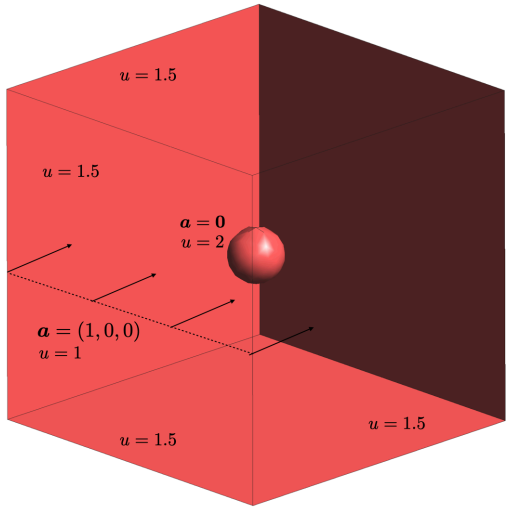


Figure 4.21: *Geometry and boundary conditions for the temperature transport over a sphere.*

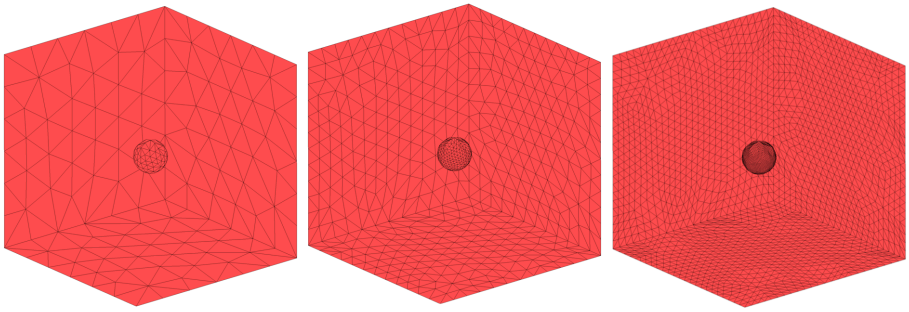


Figure 4.22: *Coarse, middle and fine meshes for the temperature transport over a sphere.*

In a first step we solve the incompressible Navier-Stokes equations at a Reynolds number 100, based on the sphere diameter. We then transport the temperature with a Péclet number of 1000 by solving the stationary convection-diffusion equation:

$$\mathbf{a} \cdot \nabla u - \frac{1}{Pe} \Delta u = 0 \text{ in } \Omega$$

Figure 4.23 shows the streamline linelets constructed on the middle mesh.

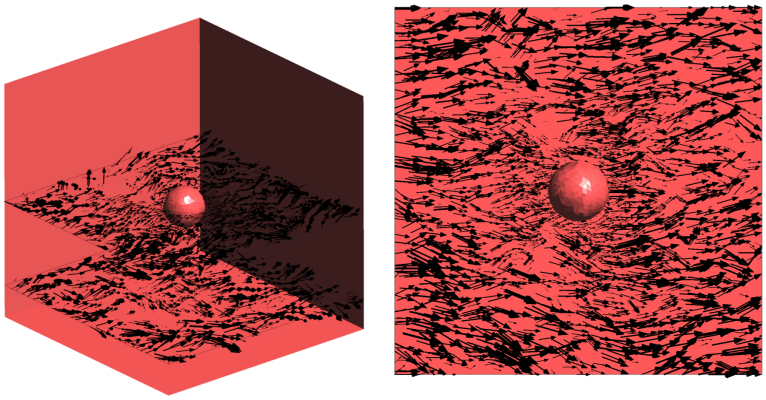


Figure 4.23: *Streamline linelets on the middle mesh for the temperature transport over a sphere.*

Figures 4.24, 4.25 and 4.26 show the results for convergence in terms of iterations and time for the three meshes considered. We use the GMRES method with a Krylov subspace of dimension 100 and a convergence tolerance of 10^{-12} using apply the diagonal, Gauss-Seidel and Bidiagonal preconditioners and the renumbering algorithm in all cases.

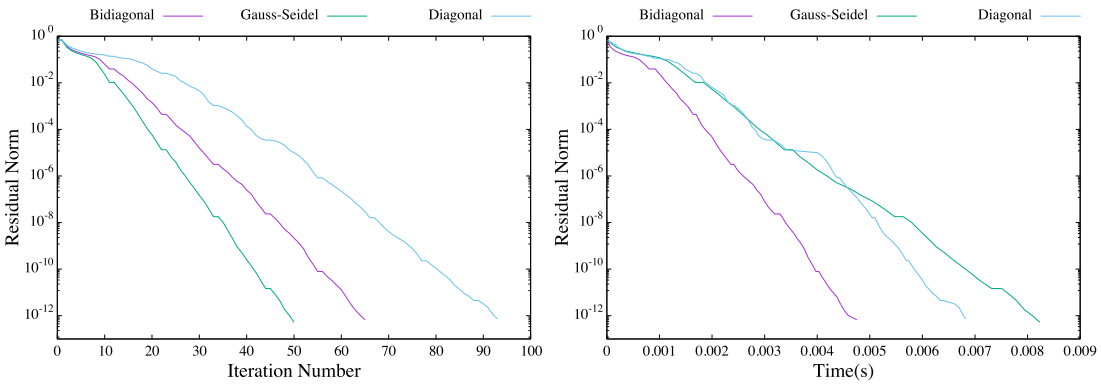


Figure 4.24: *Coarse Mesh: Results for convergence (left) and convergence times using a GMRES (right).*

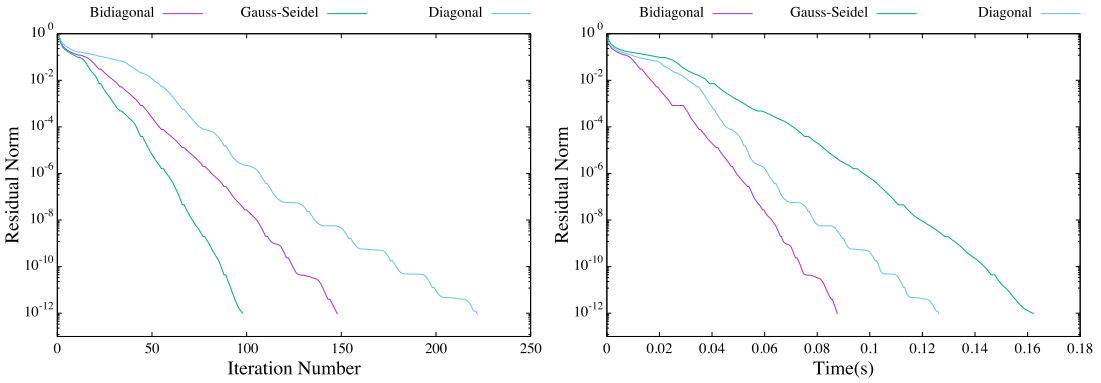


Figure 4.25: *Middle Mesh: Results for convergence (left) and convergence times using a GMRES (right).*

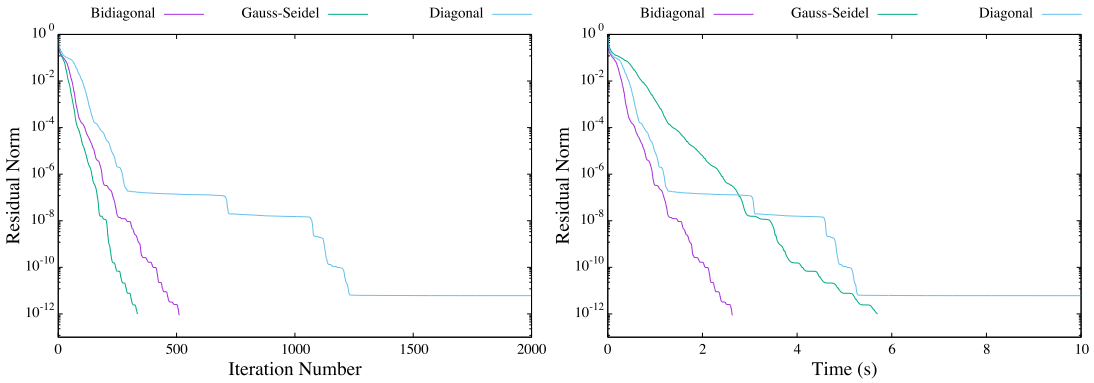


Figure 4.26: *Fine Mesh: Results for convergence (left) and convergence times using a GMRES (right).*

We summarize Figures 4.24, 4.25 and 4.26 in Tables 4.4 and 4.5. Like in the previous example we use NC in the cases where the problem has not converged.

Preconditioner	**Coarse**	**Middle**	**Fine**
Bidiagonal	59	134	464
Gauss-Seidel	45	89	304
Diagonal	84	201	NC

Table 4.4: *Number of iterations of the different preconditioners in the three meshes.*

Preconditioner	**Coarse**	**Middle**	**Fine**
Bidiagonal	$4.75 \cdot 10^{-3}$	$8.76 \cdot 10^{-2}$	$2.63 \cdot 10^{-1}$
Gauss-Seidel	$8.31 \cdot 10^{-3}$	$1.62 \cdot 10^{-1}$	$5.69 \cdot 10^{-1}$
Diagonal	$6.82 \cdot 10^{-3}$	$1.26 \cdot 10^{-1}$	NC

Table 4.5: *Convergence time (in seconds) of the different preconditioners in the three meshes.*

Figure 4.27 shows the results for the relationship of the different sizes of the meshes considered and the number of iterations.

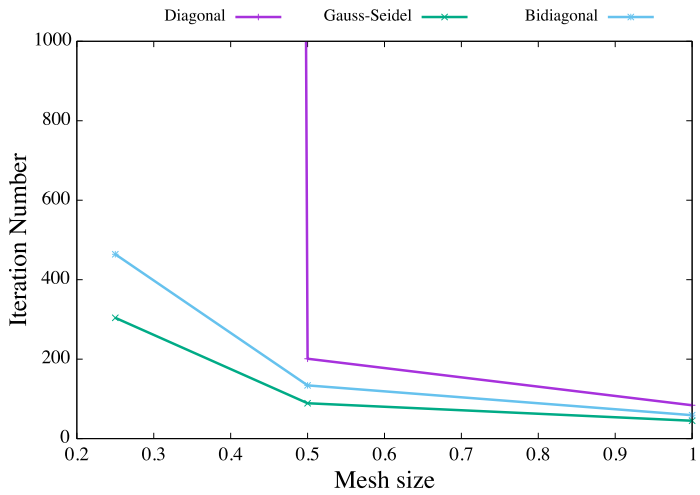


Figure 4.27: *Mesh Convergence: Relationship between the mesh size and the number of iterations required to converge.*

Remarks

- The left side of Figures 4.24, 4.25 and 4.26 show the convergence for the three meshes considered respectively using the Gauss-Seidel, bidiagonal and diagonal preconditioners together with a GMRES solver. The results in terms of convergence are similar to the ones obtained for the swirl test case. This is, the Gauss-Seidel is the one that needs less iterations to converge if compared with the other two preconditioners. We also observe in Figure 4.26, that the diagonal preconditioner does not converge in this case.

- The right side of Figures 4.24, 4.25 and 4.26 show the results for the convergence time for the problem using the same three preconditioners. In this case, differently as what happened with the swirl test case, the problem takes less time to converge using a bidiagonal preconditioner. In Figures 4.24 and 4.25 we also observe that the Gauss-Seidel is the one that takes more time to converge.

- In Figure 4.27 and Tables 4.4 and 4.5, we show the relationship be-

tween the size of the mesh and the convergence of the solver. Here we note that as the mesh gets finer, the converge slows down and as a consequence it takes more time to converge.

- The Gauss-Seidel preconditioner is the one that converges in less iterations, as it is the one that requires more information from the original matrix of the discretized problem. This does not mean that it is always faster in terms of time to converge, as one iteration using a Gauss-Seidel preconditioner is more expensive in terms of CPU time, that an iteration using a bidiagonal preconditioner. In this case we observe that the Gauss-Seidel multiplies by two the time to convergence of the diagonal preconditioner. As it has been shown convergence times depend on several aspects, such as, how the reordering is done and how many relevant terms are left away from the diagonal or the balance between number of iterations needed for the problem to converge versus the time that takes an iteration to complete its cycle.

- From the computational point of view, the way the Gauss-Seidel was implemented using permutation arrays involves non-contiguous memory accesses. This also contributes to increasing the time of one Gauss-Seidel iteration.

- All these results suggest the need for a more efficient algorithm to handle the preconditioning step when considering the Gauss-Seidel and bidiagonal preconditioners.

4.5.4 Temperature Transport over a NACA0012: Parallelization

With this example we study the effect of the parallelization on the performence of the streamline linelet preconditioner. We consider the temperature

transport over a NACA0012 profile. The 2-D mesh is composed of 21127 triangular elements and 3075 quadrilateral elements in the boundary layer (see Figure 4.28). We first solve the Navier-Stokes equations at a Reynolds number 500, and then transport the temperature with a Péclet number of 500 by solving the stationary advection-diffusion equation. To solve the algebraic system, we use the GMRES method with a Krylov subspace size of 10, and a tolerance of 10^{-12}. Here we choose the maximum angle cosine of 0.4.

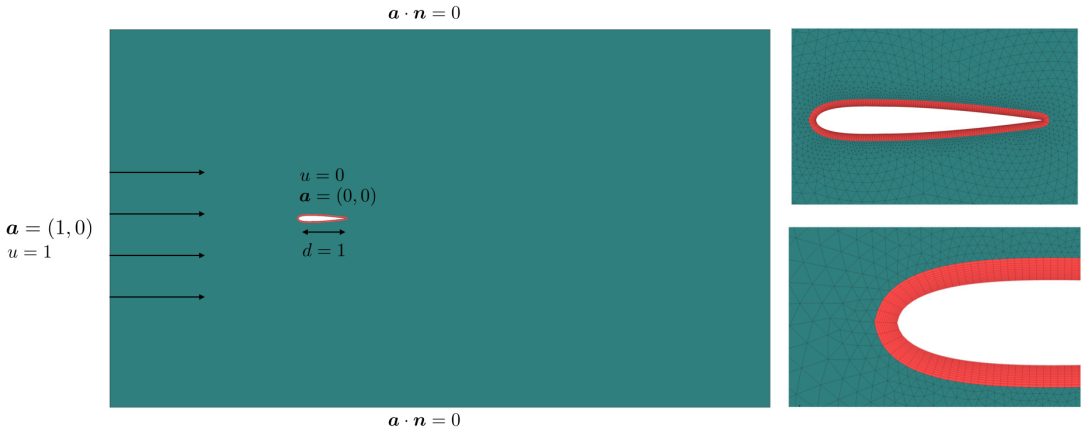


Figure 4.28: *Boundary conditions (left) and mesh (right) for NACA0012 airfoil.*

We first consider an algebraic partitioner (METIS [61]), and compare it to the sequential version and the diagonal preconditioner. Figure 4.29 shows the convergence histories obtained using different number of subdomains (indicated between parenthesis).

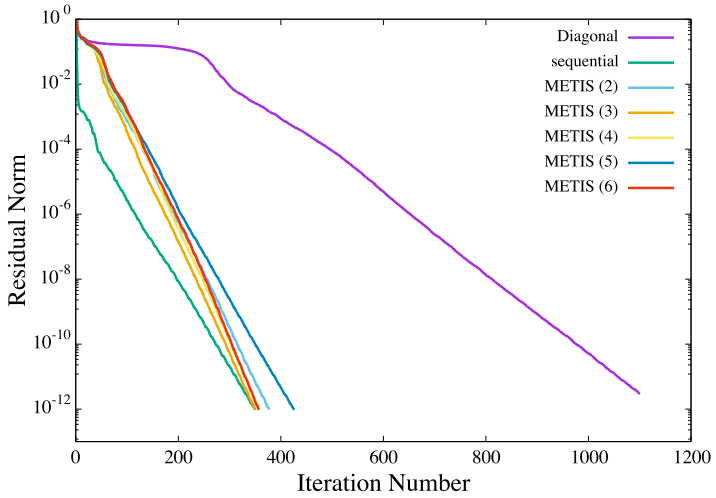


Figure 4.29: *Convergence histories: effect of the partitioning for NACA0012 airfoil.*

One can easily imagine the adverse effect of cutting the streamlines when considering running this problem in parallel. Intuitively, we expect that partitioning horizontally, streamlines would not be cut and thus the preconditioner performance would not be so much affected compared to the sequential version. Conversely, we expect that if streamlines are cut by a vertical partitioning, the convergence will be adversely affected. Let us examine this last example. In Figure 4.30, we show a vertical partitioning, together with streamline linelets.

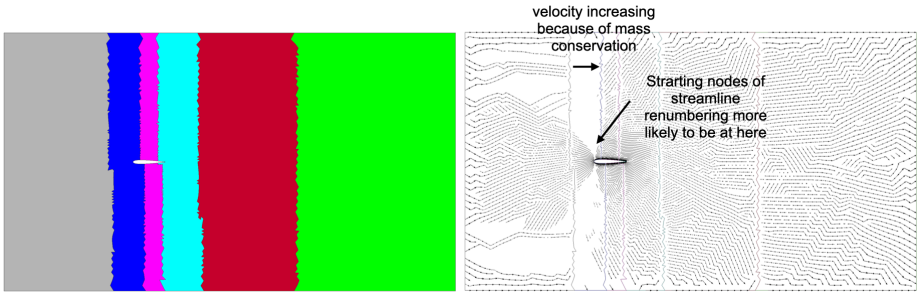


Figure 4.30: *Vertical partitioning for NACA0012 airfoil. Different subdomains (left) and streamline linelets (right).*

We first observe that almost no streamline linelets are present in the second (blue) subdomain from the left of the partitioning. How can we explain this?

In Subsection 4.3.3, we constructed the streamline linelets by choosing the seed nodes with higher values of velocity. In the present case, when the flow encounters the airfoil profile, the intensity of the velocity decreases along the x-axis to satisfy mass conservation. Therefore, the possible seeds are ordered from right to left of the subdomain. According to the algorithm described in Subsection 4.3.3, the streamlines are constructed following the flow, that is from left to right in this case. Thus, they cannot propagate when using to our algorithm. To fix this, we have slightly modified the algorithm to enable the upstream propagation of the streamline linelets. Figure 4.31 shows the linelets given by the original algorithm on the left and by the new algorithm on the right.

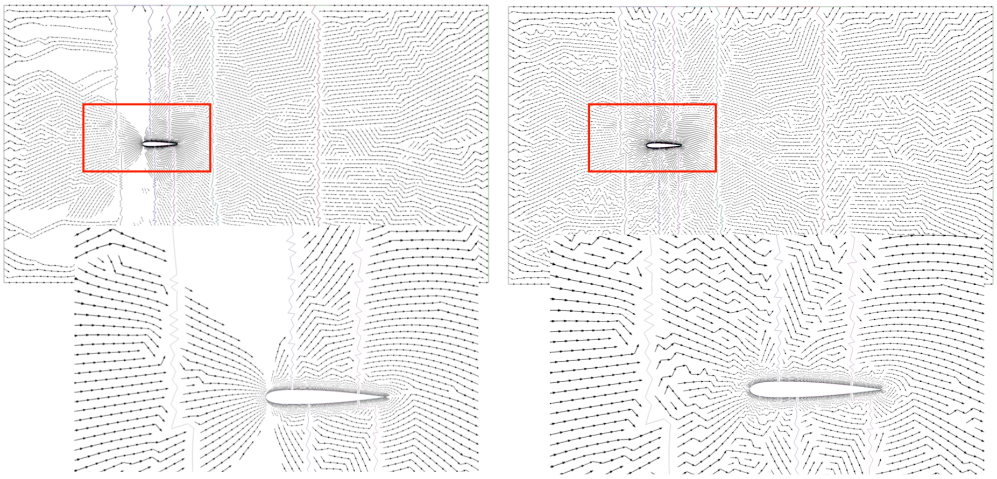


Figure 4.31: *Vertical partitioning for NACA0012 airfoil. Original algorithm (left) and modified algorithm enabling upstream propagation of linelets (right).*

Figure 4.32 compares the convergence obtained using the original algorithm, which considered only downstream propagation of streamline linelets, and the modified algorithm which accounts for both downstream and upstream propagations.

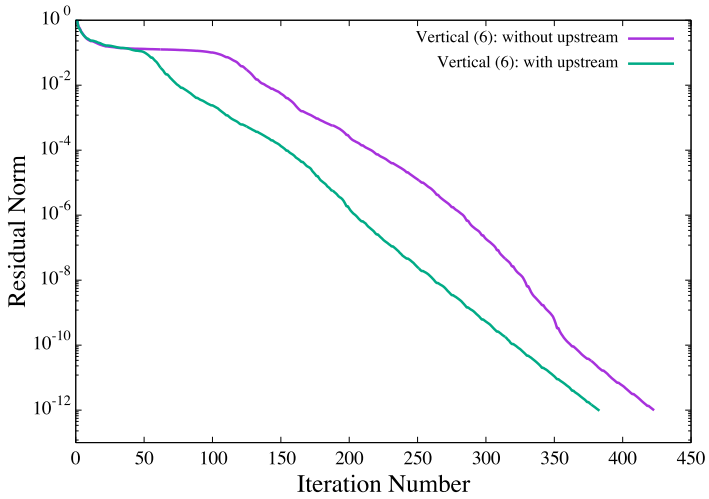


Figure 4.32: *Convergence histories of the original algorithm and modified algorithm enabling upstream linelets.*

Finally, let us compare this adverse partitioning together with METIS and a supposingly favorable horizontal partitioning, using 6 subdomains. Partitions as well as the streamline linelets are shown in Figure 4.33.

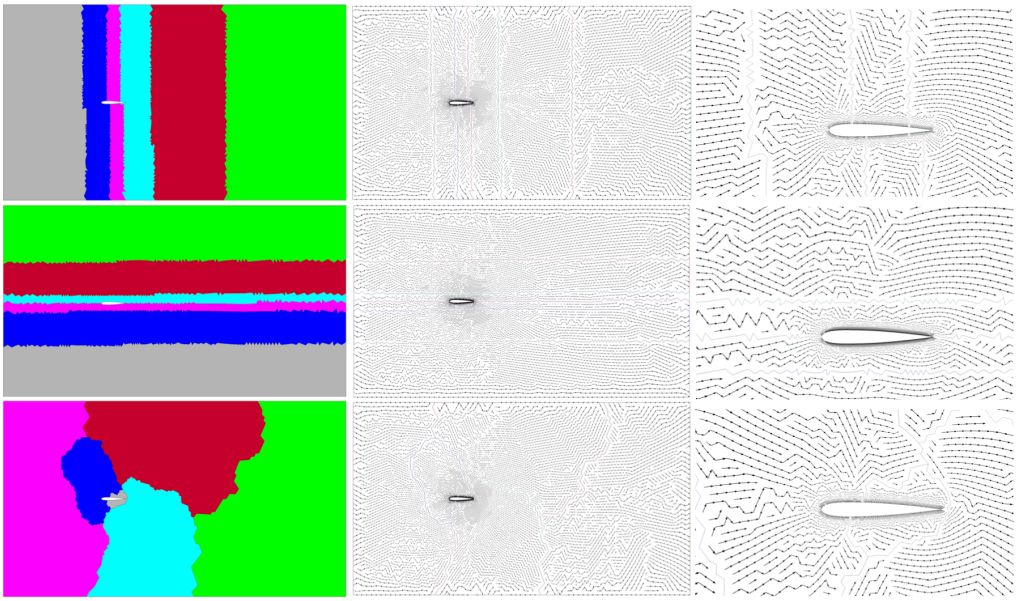


Figure 4.33: *Partitioning and streamline linelets for NACA0012 airfoil. Vertical (top), horizontal (middle) and metis (bottom).*

Figure 4.34 compares the corresponding convergences.

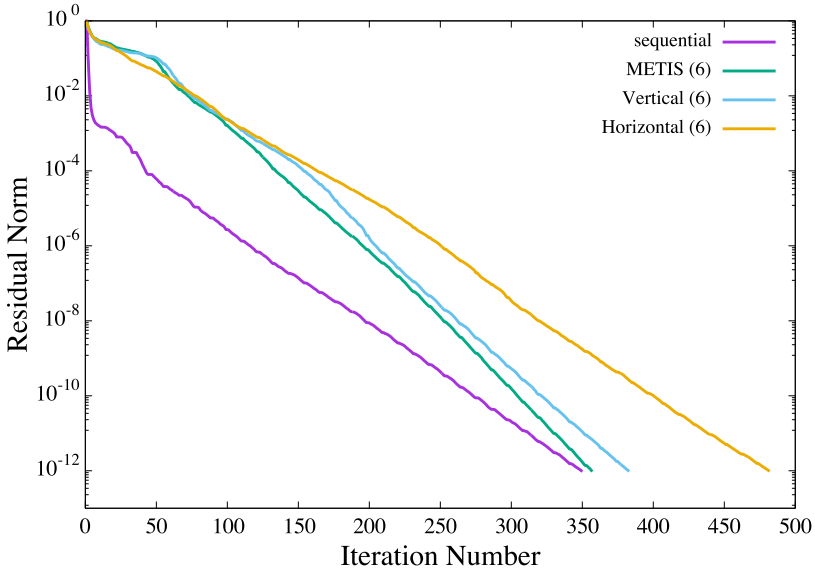


Figure 4.34: *Convergence histories using different partitionings: vertical, horizontal and METIS.*

Remarks

- In Figure 4.30 we show the effect of the partitioning for a NACA0012 airfoil. Here we observe non-negligible differences with respect to the sequential version, very clear in the first iterations of the solver. However, the impact of the number of subdomains seems to be limited.

- In Figure 4.31 the streamlines considering first only downstream propagation and then both downstream and of the linelets. It is important to use both upstream and downstream propagations as in Figure 4.32 we observe a gain in number of iterations achieved by this slight modification.

- Regarding the results for the vertical, horizontal and METIS shown in Figure 4.34, we observe that the vertical partitioning and METIS partitioning give similar convergences. We also note that their convergence rates are smaller when the residual gets small (around 10^{-4}). However, the horizontal partitioning give the worst convergence, which

was not expected at all, although the final convergence rate is quite similar to that of the sequential method. We do not have an explanation for the result obtained for the horizontal partition.

4.5.5 Temperature Transport over a NACA0012: Maximum Angle

To finish the study of the streamline linelet preconditioner, we investigate the effect of the maximum angle cosine `cangm` of Algorithm 4.1 on the convergence of the solver. To this aim, we consider the last example (NACA0012) in sequential mode, with the same four angles used in Figure 4.8. Figure 4.35 shows the convergence histories obtained for a Péclet of 500 and a Péclet of 5000. The convergences are compared with those obtained with a diagonal preconditioner and a Gauss-Seidel preconditioner using the original mesh node renumbering.

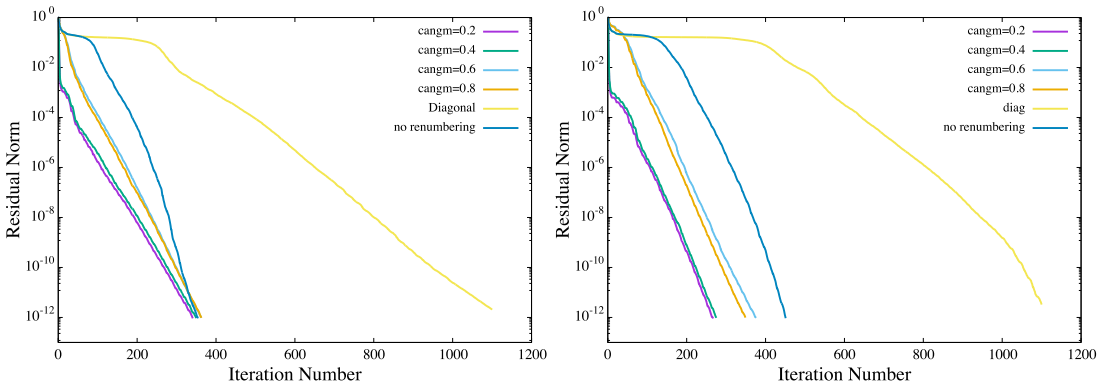


Figure 4.35: *Convergence histories using different maximum angle cosines* `cangm` *for Pe=500 (left) and Pe=5000 (right).*

Remarks

- The larger values of angle cosines 0.6 and 0.8 respectively, which lead to straight streamlines, converge in more iterations that the smaller angles, as the small angle cosines present a rapid decay of the residual

during the first iterations. However, they present a slightly better rate of convergence once a minimum residual (around 10^{-4}) is reached, in the case of smaller Péclet numbers. Therefore small angle cosines lead to faster convergence in the first phase of the solver.

- In Figure 4.35 (left) we observe that the Gauss-Seidel algorithm used without renumbering gives better results than the streamline linelet preconditioner with an angle cosine of 0.8. Although we observe this, the convergence is slower and obviously strongly depends on the node numbering, which does not depend on the direction of the flow.

- In all cases, the gain compared to a case using a diagonal preconditioner is huge, e.g. 2.5 to 3.

- We confirm the positive effects of the streamline renumbering in all cases.

4.6 Conclusions

Table 4.6 summarizes the ideal case of pure diffusion considered in Subsection 4.3.1 for the anisotropy linelet preconditioner case. It shows that renumbering the mesh appropriately, we can improve the convergence of the linear system. In this case, the condition number $\kappa(M^{-1}A)$ depends on the mesh size h, but it tends to one in case of using the close integration rule and three with the open rule as the aspect ratio increases. Table 4.7 summarizes the ideal case where we have pure advection as considered in Subsection 4.3.3. In this case, we observed, that if the system is preconditioned with the streamline linelet preconditioner (Gauss-Seidel), the condition number $\kappa(M^{-1}A)$ does not depend on the size of the mesh, and similarly to the results obtained for the pure diffusion case, it is one in case of using a close

rule and 2.5 if we use the open rule.

	open integration rule	close integration rule
Pure diffusion (Anisotropy linelet)	3	1

Table 4.6: *Condition number $\kappa(M^{-1}A)$ of the anisotropy linelet with close and open integration rule for an aspect ratio that tends to infinity.*

	open integration rule	close integration rule
Strong advection (Streamline linelet)	2.5	1

Table 4.7: *Condition number $\kappa(M^{-1}A)$ of the streamline linelet with close and open integration rule.*

From the study of these simple cases, we have proposed a renumbering strategy to generalize the application of simple preconditioners, namely Gauss-Seidel, tridiagonal and bidiagonal, to more complex and arbitrary meshes. The results show that the Gauss-Seidel and the bidiagonal preconditioners have a better performance than the diagonal preconditioner in terms of number of iterations. Also, they are more robust than the diagonal one when considering convection-dominated flows, as we have shown with the example of the swirl test case, that the diagonal preconditioner sometimes does not converge. Regarding the maximum angle cosine criterion, small angle cosines lead to a faster convergence in the first part of the solver, although this dependence is limited in terms of number of iterations. Finally, the partitioning affects the convergence of the iterative

solvers, although no general and clear tendency has been found in the examples considered.

So far we have been focusing on problems that present either a strong diffusion or a strong convection, but in most industrial problems several different physical regimes (strong diffusion and strong convection) might coexist. This is the reason why we are interested in composing the anisotropy and streamline linelet preconditioners in the same algebraic system.

Chapter 5

Composition of Preconditioners

I've always wanted to use that spell!

J.K. Rowling,
Harry Potter and the Deathly Hallows

The use of a single preconditioner, as described in the previous chapter, may not be an optimal option for general cases. In fact, the matrix can exhibit some local behaviors (meaning in some rows of the matrix) not well suited for the selected preconditioner, which can undermine the convergence of the solver. One typical example where the different numerical patterns coexist is the advection of a scalar subject to a high Péclet number with the presence of an obstacle, which may lead to a boundary layer. We actually showed in Chapter 2 the differences in the distribution of the coefficient of the matrix in zones of high Péclet and typical boundary layer involving anisotropic meshes. Also, in a parallel context, many implementations 'cut' the preconditioner at the subdomain interface nodes (as usually it is complex to construct) and the diagonal preconditioner (easy to construct on interface nodes) is used instead. In this chapter we show that this

kind of composing can be embedded in the context of some restricted preconditioners, also referred to herein as local preconditioners.

This chapter is organized as follows: In the first section, the general framework explaining preconditioner composition is set, in order to introduce the composition of the anisotropy and streamline linelet preconditioners in the following sections. Finally, several options showing how to mix these two preconditioners are presented.

5.1 Composition Methods

A general framework for the composition of preconditioners is explained. To do so, let M_1 and M_2 be two preconditioners in $\mathbb{R}^n$. By composition it is meant the use of M_1 and M_2 to build a new preconditioner M. In the following, additive, multiplicative and restricted operators are understood they are described in the literature [83, 47, 28, 69, 86, 90]. We also define two restricted preconditioners $\tilde{M}_1$ and $\tilde{M}_2$ with a rank lower than n. These two preconditioners are assumed to be good preconditioners of restricted matrices $\tilde{A}_1$ and $\tilde{A}_2$, obtained from combinations of permutations and restrictions of the global matrix as follows:

$$\tilde{A}_i = R_{pi} A R_{pi}^T, \quad \text{for } i = 1, 2 \tag{5.1}$$

where R_{pi} for $i = 1, 2$ are the restriction and permutation matrices for the non-overlapping form, as the one depicted in Figure 2.11.
We define three different types of compositions:

- **Additive**

$$M^{-1} = \alpha_1 M_1^{-1} + \alpha_2 M_2^{-1}. \tag{5.2}$$

where α_1 and α_2 are appropriate scaling factors.

- **Multiplicative**

$$M^{-1} = M_1^{-1} + M_2^{-1} - M_2^{-1} A M_1^{-1} \tag{5.3}$$

- **Additive restrictive**

$$M^{-1} = R_{p1}^T \tilde{M}_1^{-1} R_{p1} + R_{p2}^T \tilde{M}_2^{-1} R_{p2} \tag{5.4}$$

- **Multiplicative restrictive**

$$M^{-1} = (R_{p2}^T \tilde{M}_2^{-1} R_{p2})(R_{p1}^T \tilde{M}_1^{-1} R_{p1}) \tag{5.5}$$

5.1.1 Additive

The additive preconditioner defined in Equation (5.2) comes from the weighted sum of two preconditioners. It is noted that even if M_1 and M_2 are good preconditioners of A, the additive preconditioner is not necessarily a good preconditioner for A, and could even be singular. One example of such sum is the coarse level preconditioner mainly used in DDM type preconditioners or in deflation methods. In the latter case, setting D as the diagonal of A, yields (using the same notation as in [7]):

$$M^{-1} = D^{-1} - W \tilde{A}^{-1} W^T A D^{-1},$$

where $\tilde{A}$ is a coarse matrix and the columns of W are the basis of the deflation subspace (where coefficients are 1 and 0).

5.1.2 Multiplicative

The multiplicative preconditioner comes from the application of preconditioner M_1 followed by a preconditioned Richardson iteration using preconditioner M_2. Assuming the preconditioning step consists of solving $Mz = r$, then:

$$M_1 \mathbf{z}_1 = \mathbf{r} \tag{5.6}$$

$$\mathbf{z} = \mathbf{z}_1 + M_2^{-1}(\mathbf{r} - A\mathbf{z}_1) \tag{5.7}$$

By substituting the first equation into the second we obtain:

$$\begin{aligned} \mathbf{z} &= M_1^{-1}\mathbf{r} + M_2^{-1}(\mathbf{r} - AM_1^{-1}\mathbf{r}) \\ &= (M_1^{-1} + M_2^{-1} - M_2^{-1}AM_1^{-1})\mathbf{r} \end{aligned} \tag{5.8}$$

Contrary to the additive case, this two step preconditioning is necessarily well defined if M_1 and M_2 are invertible.

5.1.3 Restricted

The restricted composition is defined as the application of local preconditioners $\tilde{M}_1$ and $\tilde{M}_2$ designed for restricted matrices $\tilde{A}_1$ and $\tilde{A}_2$. These matrices can be obtained by permuting and restricting the original matrix, to isolate rows with similar behavior as formally indicated in Equation (5.1). Typical Schwarz preconditioners are obtained exactly in this way. As an example, the Restricted Additive Schwarz (RAS) preconditioner can be expressed as Equation (5.4), where the choice $\tilde{M}_i = \tilde{A}_i$ is made. That is, a local inverse of the matrix is chosen to precondition the system. It should be noted that in this case, the restriction matrices are not exactly the same on the left and right hand side of the preconditioner [19].

The linelet preconditioner can be represented as well by Equation (5.4), in the case of two linelets, and can be generalized for n linelets. First, a permutation is carried out to put the rows of the different linelets together, then a restricition is carried out to isolate each linelet and two tri-diagonal restricted preconditioners $\tilde{M}_1$ and $\tilde{M}_2$ are assembled and applied formally as given by Equation (5.4). Let us formalize the construction of the streamline linelet preconditioner for the example presented in Subsection 4.3.3, and using the definition of the permutation matrix in Equation (3.4). Let us assume the matrix is given by:

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, Dirichlet conditions on nodes 1 and 5 have been imposed by putting an arbitrary value on the diagonal 2 and 3, and by eliminating the corresponding contributions from the matrix (which are eventually cast to the right-hand side).

The permutation matrix is given by:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

so that the permuted matrix is $A_p = PAP^T$:

$$A_P = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Now, a restriction can be carried out to isolate the two linelets, as done in Subsection 3.2.3. In this case we have:

$$R_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad R_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.9}$$

Then the resulting restricted and permuted matrices $\tilde{A}_i = R_{pi} A R_{pi}^T$ are built as:

$$\tilde{A}_1 = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}, \quad \tilde{A}_2 = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix},$$

By defining $\tilde{L}_i$ and $\tilde{D}_i$ the lower and diagonal parts of $\tilde{A}_i$, the streamline linelet preconditioner can then be applied independently to each submatrix with:

$$\tilde{M}_1 = \tilde{L}_1 + \tilde{D}_1, \quad \tilde{M}_2 = \tilde{L}_2 + \tilde{D}_2$$

and formally expressed as a restricted composition as in Equation 5.4.

Remark

For the sake of clarity, we have intentionally simplified Equations (5.4) and (5.5), with respect to the way the restrictions are implemented in this work. We mentioned that the restrictions were non-overlapping in order to have one single possible update per row: rows are exclusively preconditioned by one single preconditioner. However, degrees of freedom can appear in both preconditioners (at the nodes they meet), meaning that their values will be taken into account twice (although updated once). This can be formally expressed by using different restrictions on the left and right hand sides of M_1 and M_2 in Equations (5.4) and (5.5), as in the case of RAS preconditioner.

5.2 Linelet Compostion: Streamline Linelet and Anisotropy Linelet

The main objective of this section is to blend the anisotropy and streamline linelet preconditioners to solve the same system, namely M_1 and M_2, and investigate their combined performance for problems exhibiting appropriate physical behaviors. For instance, this could be the case of a thermal boundary layer problem, where a region of strong advection is identified outside the boundary layer and a strong anisotropy is found close to the wall. A way of accomplishing this, is to compose the two preconditioners in a single one. To do so, the three ways of composing preconditioners explained in the previous section will be adapted to the particular case of considering the streamline linelet and the anisotropy linelet preconditioners.

Let us note that in this section, we consider M_1 and M_2 as two preconditioners of the original (not restricted) matrix, that is, the M_i's have the same size as the original matrix. On the nodes not belonging to any linelet, we will use the diagonal (Jacobi) preconditioner. This means that the M_i's are constructed in the same way as if they were to be applied as a single preconditioner.

5.2.1 Additive

In the case of this thesis, the additive preconditioner is defined as the sum of inverse of the two invertible preconditioners M_1^{-1} and M_2^{-1}. The streamline linelet and anisotropy linelet are inverted and added to obtain a new preconditioner that exploits the two different physics of the problem without distinguishing the different regions where the different physics dominate. This is exactly what the classical approach does [7].

Then formally, the additive preconditioner is given by:

$$M^{-1} = \alpha_1 M_1^{-1} + \alpha_2 M_2^{-1} \tag{5.10}$$

In practice only one parameter is needed. Noting that $\kappa(\alpha A) = \kappa(A)$ [79], the equation can be factorized by α_1, to lead to, if $\alpha = \alpha_2/\alpha_1$:

$$M^{-1} = M_1^{-1} + \alpha M_2^{-1} \tag{5.11}$$

In the examples of this thesis, we have chosen $\alpha = 1$.

5.2.2 Multiplicative

For the multiplicative version, we will use directly Equation (5.3). However, we will distinguish the order in which the preconditioners are applied. When composing anisotropy and streamline linelets, we will mention the first preconditioner that is applied. For example 'Mult. anisotropy' if the anisotropy linelet is applied first (M_1) and the streamline in a second step (M_2) or 'Mult. streamline' if we apply the streamline linelet first and then the anisotropy linelet. In addition, from numerical tests that we will analyze further on, we have found that the convergence of the preconditioner can be enhanced by relaxing the second preconditioning step using the relaxation parameter ω. This can be done by substituting Equation (5.7) by:

$$\mathbf{z} = \mathbf{z}_1 + \omega M_2^{-1}(\mathbf{r} - A\mathbf{z}_1) \tag{5.12}$$

5.2.3 Restricted

The restricted preconditioning is the new approach that this thesis introduces, if compared with the classical additve or multiplicative ones. The approach in this case, is different, as different regions are identified and different preconditioners are applied to these regions, depending on the physics that dominates in each case. Ideally, in the region where convection dominates, the streamline linelet preconditioner will be picked, whereas in the region where diffusion dominates, the anisotropy linelet will be the preferred one.

This could be achieved for example by using a criterion based on a nodal measure of the Péclet number to decide on the preconditioner to be used. However, in order to compute this nodal Péclet one would need to access nodal values for convection, diffusion and mesh size. To avoid introducing problem-dependent criteria in the solver, we thus chose another strategy. First, anisotropy linelet nodes are identified through the geometrical criterion used to initiate and grow the different linelets [85]. Then, the remaining nodes will be identified as streamline linelet nodes.

This selection procedure is illustrated in Figure 5.1. We present an example of a mesh in which there are two different regions: a region that presents a strong anisotropy in the y-direction (green nodes in Figure 5.1) and an outer region away from the one with strong anisotropy (blue nodes in Figure 5.1). We identify first, the nodes belonging to the anisotropic boundary layer (in green), based on a geometrical criterion. The remaining nodes are then identified as streamline linelets (in blue). This is a first level of permuation. The second level of permutation consists in constructing each individual linelet

(see bottom part in Figure 5.1 and Table 5.1). Finally, the two preconditioners are composed:

$$M^{-1} = R_{p1}^t \tilde{M}_1^{-1} R_{p1} + R_{p2}^t \tilde{M}_2^{-1} R_{p2} \tag{5.13}$$

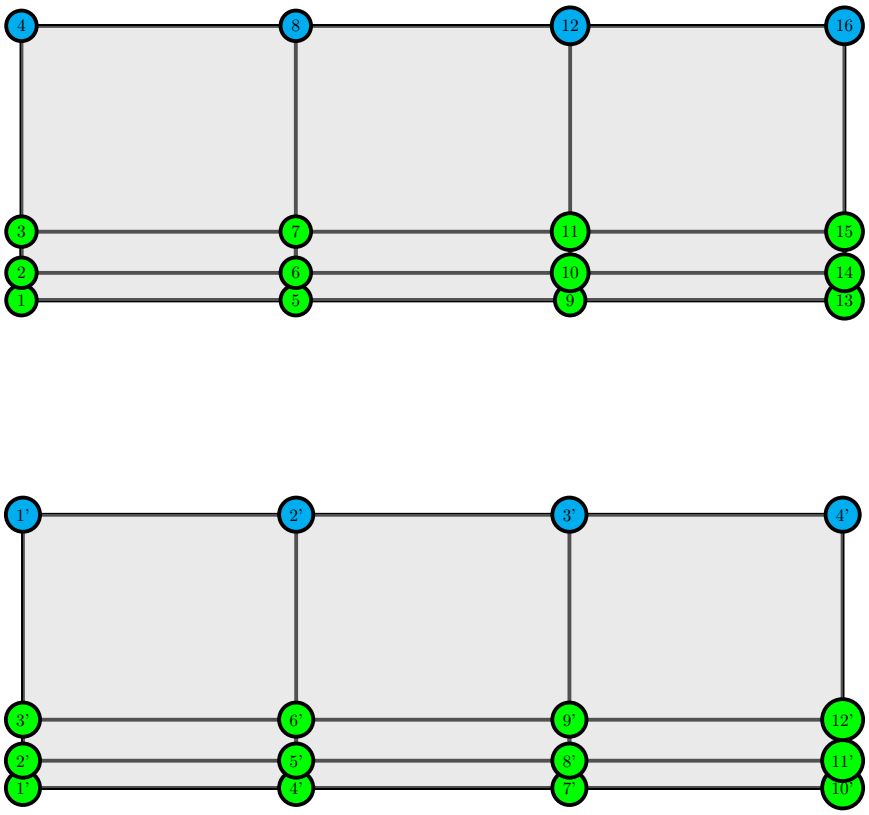


Figure 5.1: *Example anisotropic mesh - Global numbering (top) and local numbering (bottom).*

Global Numbering Anisotropy Linelet	Local Numbering Anisotropy Linelet	Linelet Number
1	1’	1
2	2’	1
3	3’	1
5	4’	2
6	5’	2
7	6’	2
9	7’	3
10	8’	3
11	9’	3
13	10’	4
14	11’	4
15	12’	4

Global Numbering Streamline Linelet	Local Numbering Streamline Linelet	Linelet Number
4	1’	1
8	2’	1
12	3’	1
16	4’	1

Table 5.1: *Global and local numberings for the anisotropy and streamline linelets.*

5.3 Results

In this section, we show two examples where we have tested the composition of the two preconditioners. In these examples we will study:

- The performance of the four different compositions described in this chapter, compared to the streamline linelet, the anisotropy linelet and the diagonal preconditioners.
- The convergence time of all the preconditioners we have mentioned so far.

For the sake of clarity, with the results of this section we will introduce the following notation:

- Composed preconditioners

  Additive: Addit.

  Restrictive: Restr.

Multiplicative if we apply the anisotropy linelet first: Mult. anisotropy

Multiplicative if we apply the streamline linelet first: Mult. streamline

- Single preconditioners

  Anisotropy linelet: Anisotropy lin.

  Streamline linelet: Streamline lin.

Also we have to mention that in the case of the streamline linelet preconditioner in this chapter we have only considered the Gauss-Seidel method.

5.3.1 Blasius - Thermal Boundary Layer Flow Over a Flat Plate

The Blasius boundary layer can be simulated considering a sufficiently long flat plate on which a constant flow impinges. In this context, two main regions can be clearly distinguished, the boundary layer region and the region away from the boundary. In the first region, diffusion dominates over convection, whereas in the second one, is the other way around. This configuration makes it ideal to test the compositions presented previously.

As in Subsection 4.5.3, we first solve an incompressible flow with prescribed velocity at the inflow. The Reynolds number is 500, based on the plate length. The mesh and the boundary conditions used for this simulation are shown in Figure 5.2. The mesh is composed of 6000 quadrilateral elements in the boundary layer and 10188 triangular elements in the core flow. Once the velocity is obtained, we then consider the temperature transport. Here, we select a Péclet number of 500. The problem is solved as unsteady using the Euler scheme.

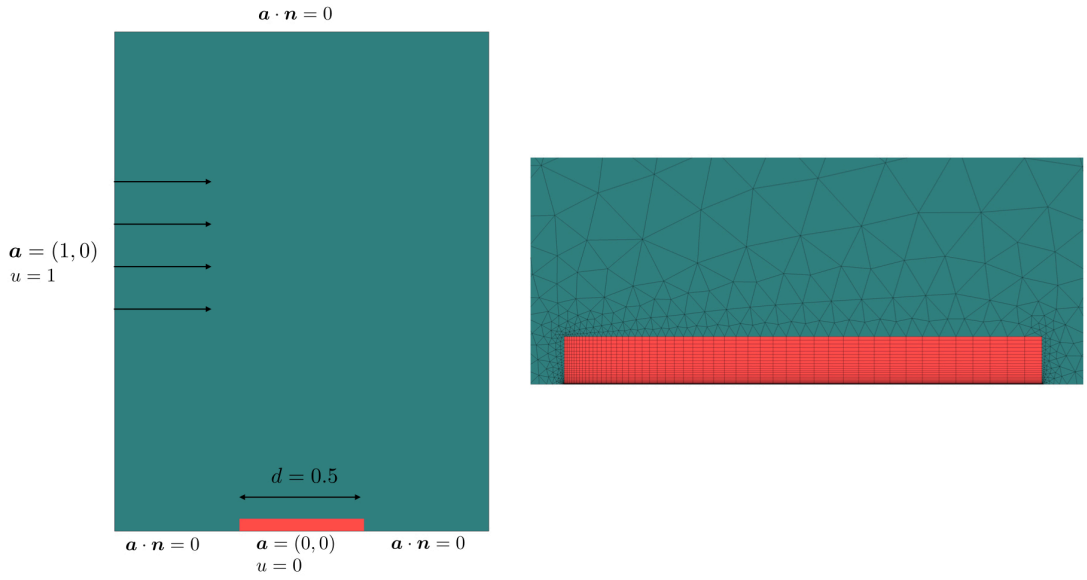


Figure 5.2: *Boundary conditions (left) and mesh (right).*

Figure 5.3 shows the anisotropy linelets constructed in the boundary layer, colored by their numbers. We observe that they are longer and longer as we move away from the plate, while anisotropy is increasing. This is due to the refinement at the start of the plate that limits the anisotropy in this region. When considering the restrictive preconditioners, streamline linelets are constructed in the rest of the domain.

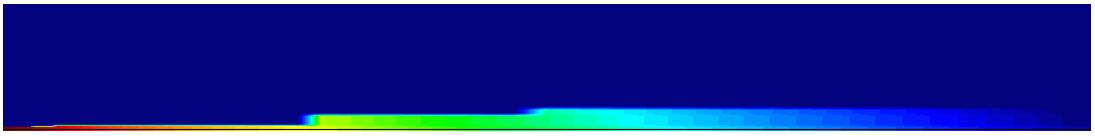


Figure 5.3: *Anisotropy linelets colored by their numbers.*

In Figures 5.4 and 5.5, we show the results obtained for the Blasius boundary layer. We present results for convergence and time. In all cases we have applied all the preconditioning strategies to a GMRES iterative solver with a Krylov dimension of 70. The convergence tolerance has been set to 10^{-10}.

Figure 5.4 compares the different composition strategies with the streamline linelet, anisotropy linelet and diagonal preconditioners.

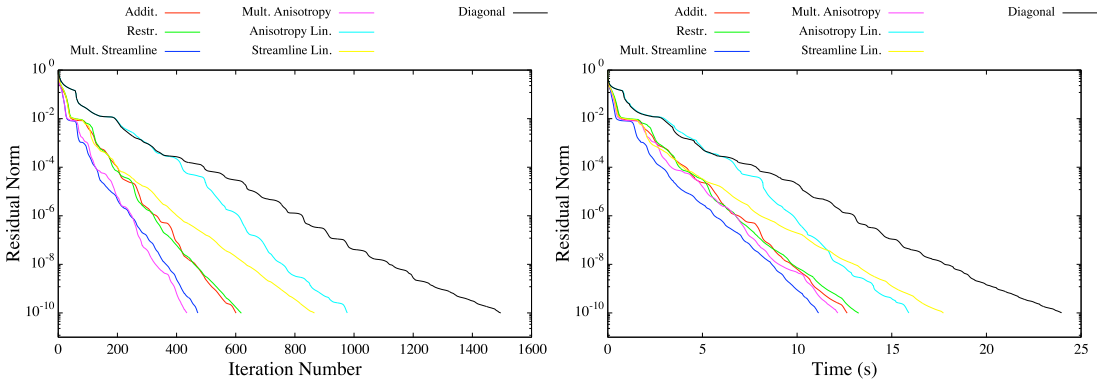


Figure 5.4: *Results for convergence (left) and convergence times (right). Comparison of the all composition strategies with the anisotroy linelet, streamline linelet and diagonal preconditioners.*

Figure 5.5 shows the convergence and time of the four different approaches that we have used for the composition of the preconditioners.

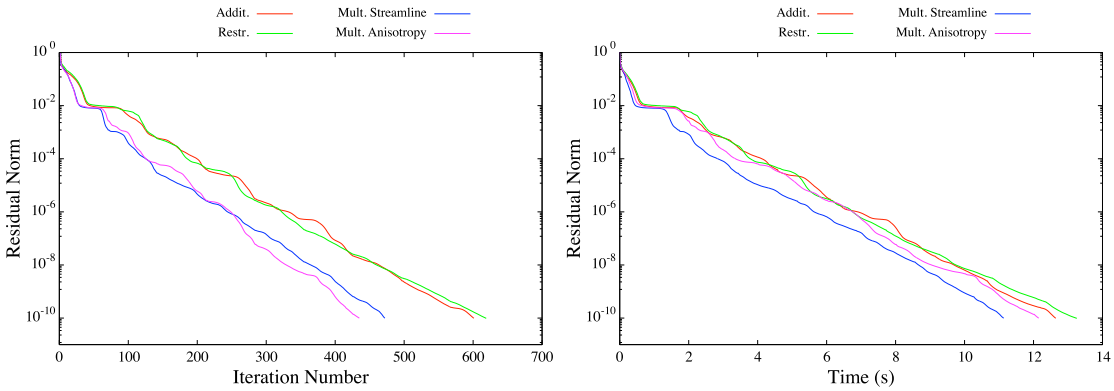


Figure 5.5: *Results for convergence (left) and convergence times. Comparison of the different composition strategies.*

Preconditioner	**Number of iterations**	**Time (s)**
Diagonal	1477	23.9
Anisotropy Linelet	993	15.9
Streamline Gauss-Seidel	853	17.7
Mult. Streamline	465	11.1
Mult. Anisotropy	428	12.1
Additive	769	12
Restrictive	610	12.6

Table 5.2: *Comparison of the preconditioners in terms of iterations and time.*

Remarks

- Table 5.2 shows the results for convergence and time for all the types of preconditioners considered in this thesis. The results clearly show that the compositions of preconditioners in any of its four variants are the ones with the best performance. IFor instance, the Mult. anisotropy is the one that is the best in terms of iterations and the Mult. streamline in terms of time (see Figure 5.5).

- In this example, the multiplicative preconditioner converges without relaxation ($\omega = 1$ in Equation (5.12)).

- The main goal of combining the anisotropy linelet and the streamline linelet into one single preconditioner is to take into account the physical characteristics that a general problem can exhibit (e.g. strong convection away from the boundary and diffusion dominant close to the boundary layer), so it makes sense that if the two preconditioners are used for the same problem, they will have a better performance than if they are used on their own.

- The additive and restrictive additive preconditioners behave equally, as they are similar by construction. In fact, the main difference is the additional diagonal preconditioning provided by the additive preconditioner and possibly the streamline linelets crossing the boundary layer.

5.3.2 NACA0012 - Thermal Boundary Layer

Temperature effects on airfoils such as icing has a negative impact on aerodynamic performance. For this reason a detailed understanding of convective heat transfer is necessary to, among others, develop de-icing

methods [97]. From a numerical point of view, these problems can be very challenging, as the temperature gradients created near the surface of the airfoil due to the difference of temperature between the fluid free stream and the surface, gives rise to steep boundary layers. Near the surface, heat transfer occurs only through heat conduction, whereas away from the surface heat transfer is done mainly by heat convection (see Figure 5.6). This situation of having two different physics in a given configuration, makes it difficult to choose an appropppriate method to solve the corresponding algebraic system. Using the local preconditioning approach that includes both the properties for high advection problems (heat convection away from the surface of the airfoil) and boundary layer problems (heat conduction close to the surface of the airfoil) can be a good option to improve the convergence results for a NACA0012 airfoil.

In the same way as we have done with the Blasius example, we consider an incompressible flow surrounding the NACA0012 airfoil and we use the resultant velocity field from solving the momentum and continuity equations as an input for the convection-diffusion equation for temperature. See Subsection 4.5.4 for the description of the test case. Here, we select a Péclet number of 100. As we have done with the previous example, we will precondition the linear system with the different composition methods that we have described in this chapter and compare them to the anisotropy linelet, the streamline linelet and the diagonal preconditioners.

Figure 5.6 (left), depicts the linelets colored by their numbers, while the right Figure shows the boundary layer developement around the airfoil.

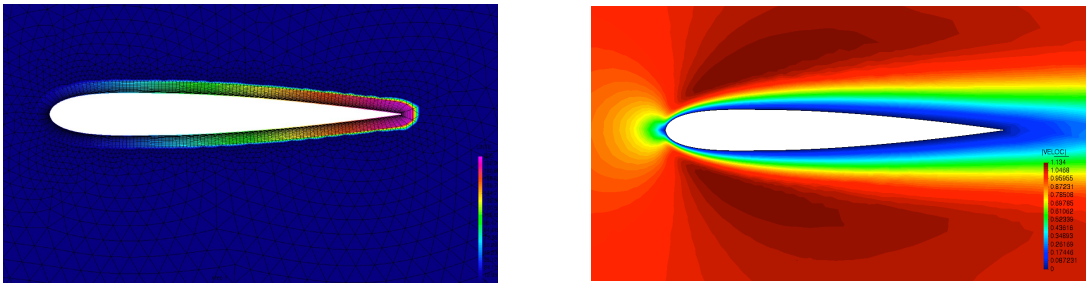


Figure 5.6: *Anisotropy linelets colored by their numbers (left) velocity magnitude in the boundary layer (right).*

Figures 5.7 and 5.8 show the results obtained for the NACA0012 airfoil for convergence and times. In all cases we have applied the different preconditioning strategies to a GMRES solver with a Krylov subspace dimension of 10 and a convergence tolerance of 10^{-12}.

Figure 5.7 shows the different convergence histories and time to solution obtained with the different preconditioners used in this thesis.

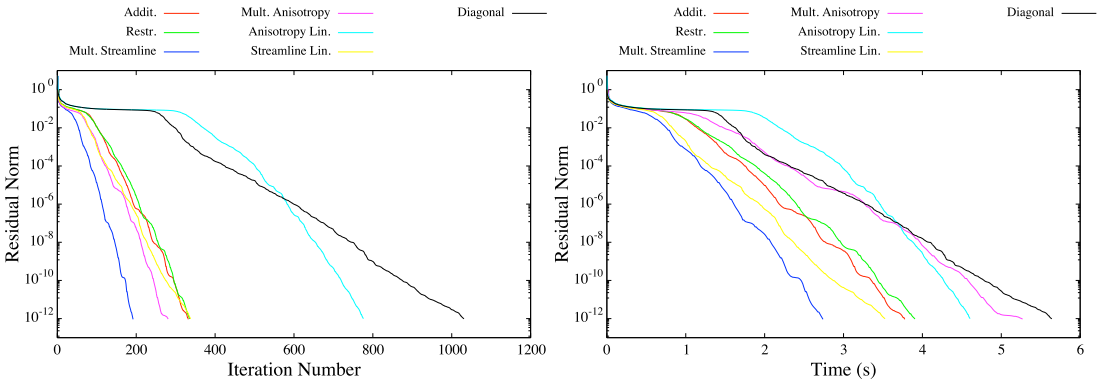


Figure 5.7: *Results for convergences (left) and convergences and times (right).*

To better appreciate the differences between the composed preconditioners, Figure 5.8 shows the convergence of a subset of these preconditioners.

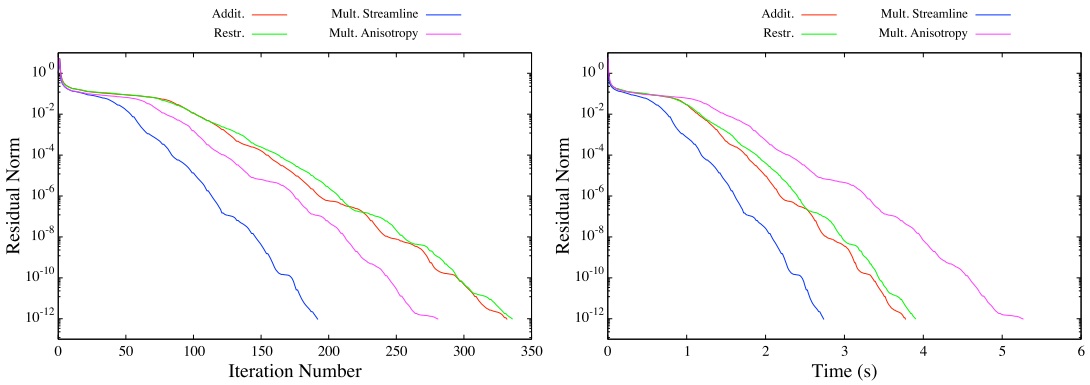


Figure 5.8: *Results for convergences (left) and times (right).*

Preconditioner	Number of iterations	Time (s)
Diagonal	937	5.6
Anisotropy Linelet	705	4.6
Streamline Linelet	307	1.1
Mult. Streamline	174	2.78
Mult. Anisotropy	255	5.2
Additive	301	3.8
Restrictive	305	3.9

Table 5.3: *Comparison between preconditioners.*

Remarks

- Table 5.3 shows the results for convergence and time of all the types of the streamline linelet and the anisotropy linelet and of the four different approaches that have been implemented containing the composition of both preconditioners. By comparing the methods, we observe similar time and convergence to the ones obtained with the Blasius case.

- All the composed preconditioners give better convergence and times than the single preconditioners.

- The anisotropy linelet preconditioner exhibits the worst performance

(excepting for the diagonal preconditioner), although after some iterations its rate of convergence seems to decrease down to the same value as the others (low rate of convergence means faster convergence see Equation (3.41) for τ). This can be due to the fact that the main errors at the beginning of the iterations are located in the core flow, which are easily removed by the Gauss-Seidel algorithm.

- The preconditioners involving the streamline linelets are the most efficient ones, as the rate of convergence is low very soon in the iterative process.

- Regarding the convergence of the multiplicative preconditioners, starting with the Gauss-Seidel gives a better rate of convergence than starting with the anisotropy linelet.

- No much difference is found between the additive and restrictive versions, which are quite similar in their constructions.

5.4 Conclusions

In this chapter we have presented several strategies to compose two different preconditioners, namely the anisotropy and streamline linelet ones. These preconditioners are tested on problems which offer regions with strong convection and regions with strong diffusion. Tables 5.5 and 5.3 of show that any of the composition strategies have a better performance in terms of convergence (iterations) and time compared to the case in where we solve our problem only with one of the two preconditioners. As we expected, when different regimes coexist, the composition of preconditioners is a suitable strategy to accelerate the convergence of the linear system. Also, it should be mentioned that the gain in convergence not always means a gain in time,

due to the additional operations involved in the composition with respect to using one single preconditioner. However, there remains room for improvement as in this thesis, we have concentrated more on the algorithmic performance than on the computational performance.

Chapter 6

Numerical Applications: Navier-Stokes Equations

The further he climbed,
the closer he got.
To the slumbering lion
reclining on top

Rachel Bright & Jim Field
The Lion Inside

In this chapter, the preconditioners presented in Chapters 4 and 5 are applied to the solution of the Navier-Stokes equations. First, a brief introduction to the Navier-Stokes equations is given. Then, we introduce two different sets of equations, namely the compressible Euler equations and the incompressible Navier-Stokes equations. The first set is considered to test the streamline linelet preconditioner which should be well suited, in principle, to hyperbolic problems. We finally end with the simulation of the incompressible Navier-Stokes equations on a test case involving both a boundary layer and a core flow to study the behavior of the different preconditioner compositions introduced in the previous chapter.

6.1 Navier-Stokes equations

In this section, a brief introduction to the Navier-Stokes equation is given.

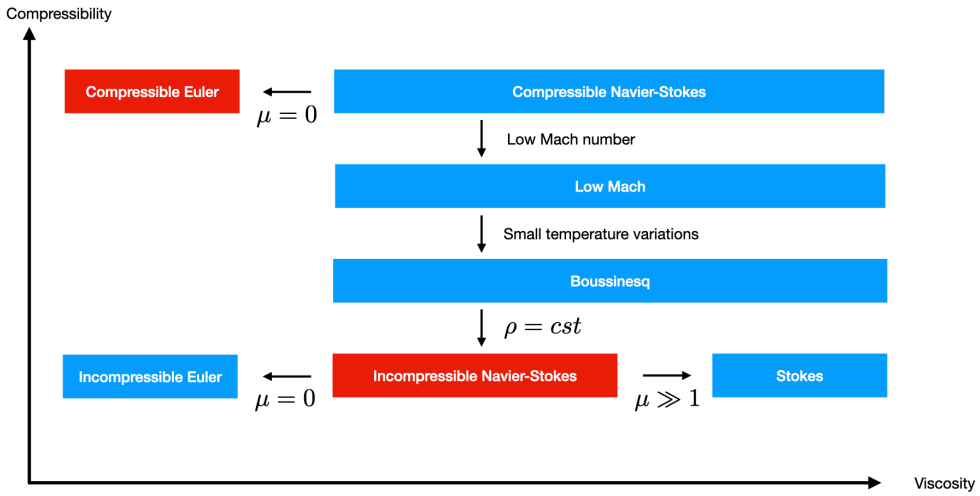


Figure 6.1: *Different approximations of the Navier-Stokes equations. The ones in red are the ones studied in this chapter.*

In particular, we will focus on two approximations of the general Navier-Stokes equations, namely the compressible Euler equations and the incompressible Navier-Stokes equations. Figure 6.1 shows the common approximations of the general Navier-Stokes equations, depending on the compressibility, which characterizes the dependence of the density on other flow variables (e.g. pressure and temperature) and on the viscosity. In fact, starting from the general governing equations, some simplifications can be made to decrease the complexity of the equations, making them more tractable numerically without compromising the accuracy.

The compressible Euler equations govern the motion of an inviscid (no viscosity) and adiabatic flow (no heat conduction). They express the conservations of mass, momentum and energy. If $\mathbf{u}$ is the fluid velocity, p the pressure, ρ the density, E its specific total energy, the equations in the non-conservative form read:

$$\begin{aligned} \partial_t \rho + \nabla(\rho \mathbf{u}) &= 0 \\ \rho \partial_t \mathbf{u} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p &= \mathbf{0} \\ \rho \partial_t E + \rho \mathbf{u} \cdot \nabla E + \nabla \cdot (p\mathbf{u}) &= 0 \end{aligned} \tag{6.1}$$

together with the following closure equations:

$$\begin{aligned} p &= (\gamma - 1)\rho e, \\ e &= E - \frac{|\mathbf{u}|^2}{2} \end{aligned}$$

where e is the internal energy and γ is the ratio of specific heats. One important number in the case of the compressible Euler equations is the Mach number M, which non-dimensionalizes the velocity with respect to the speed of sound. These Euler equations should be provided with initial and boundary conditions.

The incompressible Navier-Stokes equations consist of an approximation of the general set of equations. They are considered valid when the Mach number is sufficiently low (0.3 is an accepted value), and if the temperature influence on the density is negligible. Basically, the density is assumed to be constant. Finally, these assumptions remove the dependence of the mass and momentum conservations from the energy. They read:

$$\begin{aligned} \rho \partial_t \mathbf{u} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla \cdot [2\mu\epsilon(\mathbf{u})] + \nabla p &= \mathbf{0} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \tag{6.2}$$

where μ is the dynamic viscosity and ϵ is the rate of deformation tensor

defined as

$$\epsilon(\mathbf{u}) = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$$

These equations should be provided with initial and boundary conditions.

In a similar way as it was done in Chapter 2, the non-dimensional form of the Navier-Stokes equations is now presented, to easily analyze limiting behaviors. For the sake of simplicity, this is done with the incompressible form of the equations. For more details on how to find the non-dimensional form of the compressible Navier-Stokes equations see [12].

Let us consider u_0 and x_0 some characteristics measures for velocity and length, respectively. We assume constant density and viscosity. By non-dimensionalizing the variables, and redefining the original variables for the sake of clarity, we end up with the non-dimensional form of the incompressible Navier-Stokes equations:

$$\underbrace{\overbrace{\partial_t \mathbf{u}}^{Parabolic} + (\mathbf{u}\cdot\nabla)\mathbf{u}}_{Hyperbolic} \underbrace{\overbrace{-\frac{2}{Re}\nabla\cdot\epsilon(\mathbf{u}) + \nabla p}^{Parabolic} = \underbrace{\mathbf{0}}_{Hyperbolic}}_{Elliptic} \qquad (6.3)$$

$$\nabla\cdot\mathbf{u} = 0$$

where Re is the Reynolds number, given by:

$$Re = \frac{\rho u_0 x_0}{\mu} \qquad (6.4)$$

The Reynolds number Re can provide valuable information on the behavior of the flow, as it measures the ratio of the inertia over the viscous forces. It

thus indicates the global state of the flow, laminar, transitional or turbulent [35].

As a system of equations, the Navier-Stokes momentum equations present similar characteristics to the transport equation of a scalar, as studied in the case of the temperature equation (here the Reynolds number plays the role of the Péclet number, as explained in Chapter 2). In the case of the incompressible Navier-Stokes equations, the viscosity term is responsible for the so-called no-slip condition (a fluid sticks to a surface), which triggers boundary layers. Similarly, far from the walls, we may be in the presence of a core flow, almost uniform, where we expect the behavior to be locally hyperbolic. We will take advantage of these characteristics to test the different preconditioner compositions. We finally note that in the case of the Euler equations, due the absence of viscous and thermal dissipation terms, the equations are globally hyperbolic and the no-slip condition should be substituted to a slip condition together with a no-penetration condition on walls.

6.2 Streamline Linelet: Euler equations

In this section two different airfoils have been studied, these are the NACA0012 and the ONERA M6. For both of them the streamline numbering strategy has been tested using the Gauss-Seidel and bidiagonal preconditioners and these two, have been compared to the diagonal preconditioner. The approach has been exactly the same as in the test cases of Chapter 4, but this time it has been applied to the compressible Euler equations. For the NACA0012 test case, the effects of this renumbering are shown for a problem run in sequential, and in the ONERA M6 we study the effects of

parallelization of the streamline linelet algorithm.

6.2.1 NACA0012 Airfoil

The aerodynamic properties of a wing, a propeller or a turbine blade are determined by the precise of the airfoil that is used. In case of the NACA0012 (and other airfoils too), depending on the boundary conditions, shear layer interactions, Reynolds number for instance, complex flow patterns may appear which usually make the convergence of the non-linearity and of the iterative solvers challenging. For example at high Reynods number the convection term described in Equation (6.1) dominates over the diffusive one. In this case, and knowing the results obtained from the convection-diffusion equation, if the mesh is reordered following the direction of the velocity $\mathbf{u}$, the convergence of the iterative solver is expected to improve.

In the present case, we propose to solve the compressible Euler equations to simulate the inviscid flow over a NACA0012 airfoil. Therefore, no boundary layer is present near the profile where a slip condition is imposed. To do so, we consider a 2-D mesh of 4522 elements (see Figure 6.2). The problem is solved as unsteady using the Euler scheme in time. We solve the corresponding monolithic linear system using a GMRES algorithm with a Krylov subspace of dimension 30 and BiCGSTAB solvers and a convergence tolerance of 10^{-6} for both cases. We will precondition the system with the streamline linelet (using the Gauss-Seidel and bidiagonal) and the diagonal preconditioners.

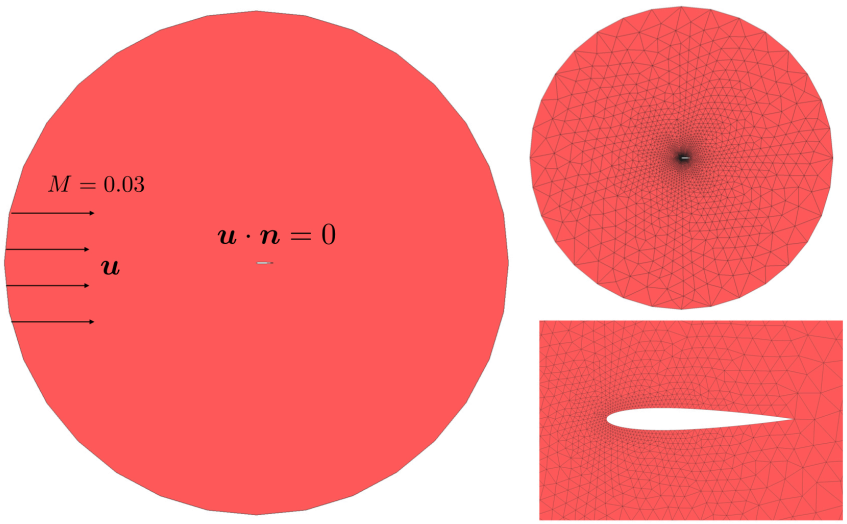


Figure 6.2: *Geometry and boundary conditions (left) mesh details (right).*

Figure 6.3 shows some views of the streamline linelets, representing the node-to-node vectors of each linelets. We observe, that they follow the flow direction.

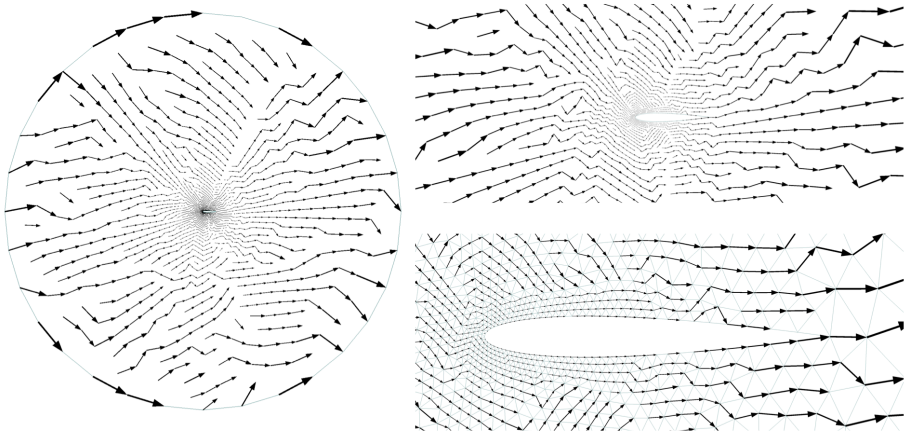


Figure 6.3: *NACA0012: Node to node vectors forming the streamline linelets.*

Figure 6.4 shows the results for the convergence of the streamline linelet and diagonal preconditioners using a GMRES (left) and BiCGSTAB (right).

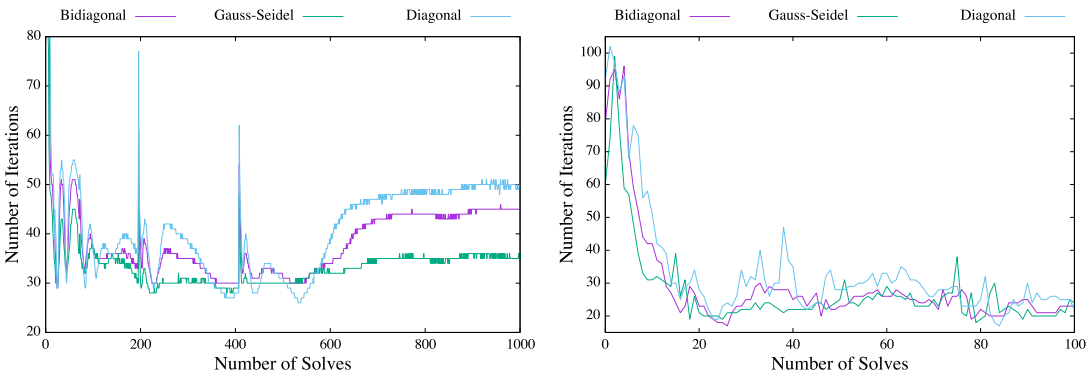


Figure 6.4: *NACA0012: Results for convergence with a GMRES (left) and BiCGSTAB (right).*

Remarks

- Figure 6.4 shows the convergence of the Gauss-Seidel, Bidiagonal and Diagonal preconditioners using GMRES and BiCGSTAB. In both cases (GMRES and BiCGSTAB), the Gauss-Seidel preconditioner is the one that requires less iterations to converge if compared with the other preconditioners.

- The right-hand-side of Figure 6.4 shows that in the case of the BiCGSTAB solver it takes less iterations to converge than with the GMRES. This behavior is similar to the test cases shown in Chapter 4 and it is due to the difference between the two iterative solvers [46].

- With this problem, we show that the renumbering also works for the inviscid compressible Euler equations, although the gain in number of iterations is less than for the scalar case (around 30% less iterations of the Gauss-Seidel preconditioner with respect to the diagonal preconditioner) in the case of GMRES.

6.2.2 ONERA M6 Airfoil

Let us know consider a more challenging problem. In this case we consider the ONERA M6 airfoil, which is a standard transonic test case frequently used to validate CFD codes [70, 81]. In this case we want to use the inviscid compressible Euler equations in 3-D to study the effect of the parallelization on the performance of the streamline linelet preconditioner. In Figure 6.5 we show the details of the mesh (composed of 472026 elements) and the boundary conditions used to solve the problem. The problem is solved as unsteady using the Euler scheme. We will solve the linear system using GMRES with a Krylov subspace of dimension 100 and we will precondition the system with the streamline linelet (using the Gauss-Seidel and bidiagonal) and the diagonal preconditioners. The convergence tolerance is set to 10^{-6}.

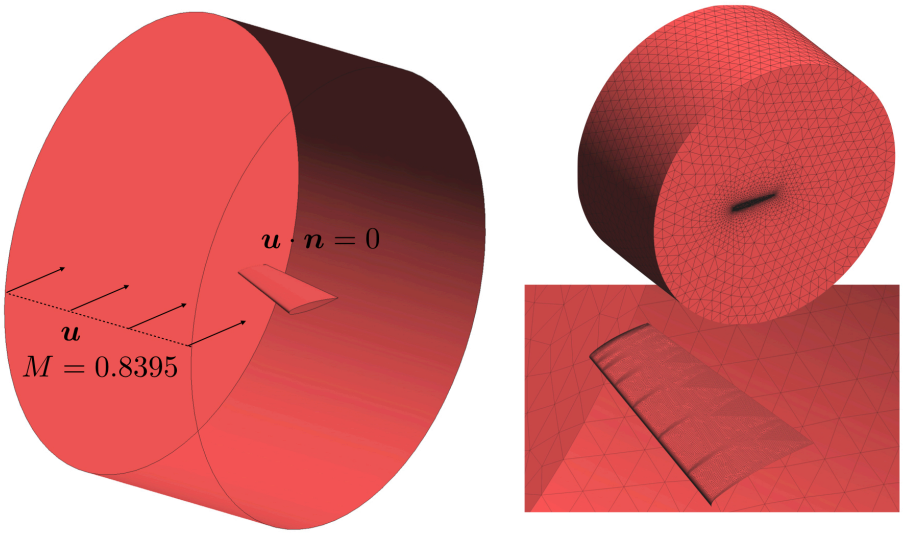


Figure 6.5: *Geometry and boundary conditions (left) mesh details (right).*

Figure 6.6 shows the velocity and temperature profiles around the airfoil, showing the presence of the so-called lambda shock [66].

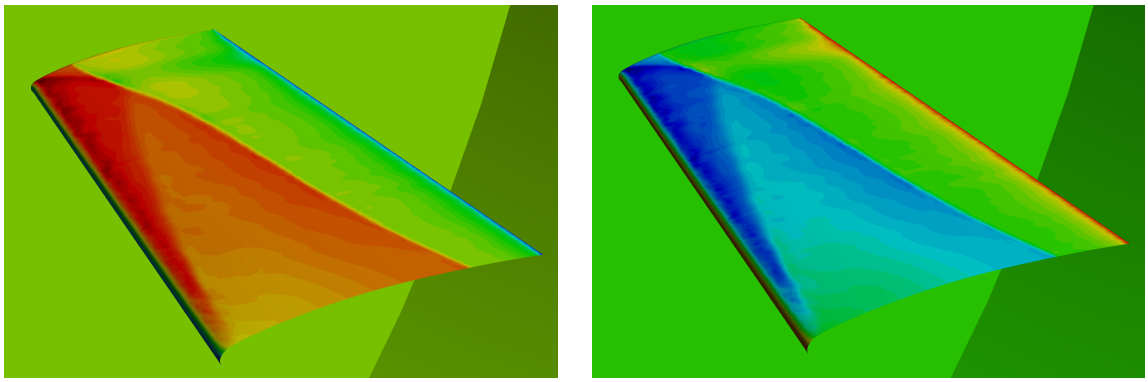


Figure 6.6: *Velocity contours (left) temperature contours (right).*

Figure 6.7 shows some snapshots of the streamline linelets. The left part of the figure shows the different streamline linelets in the near-airfoil region. The other view shows the vectors going from node-to-node and eventually forming the different streamline linelets. By construction, we can observe that they in general follow the flow direction.

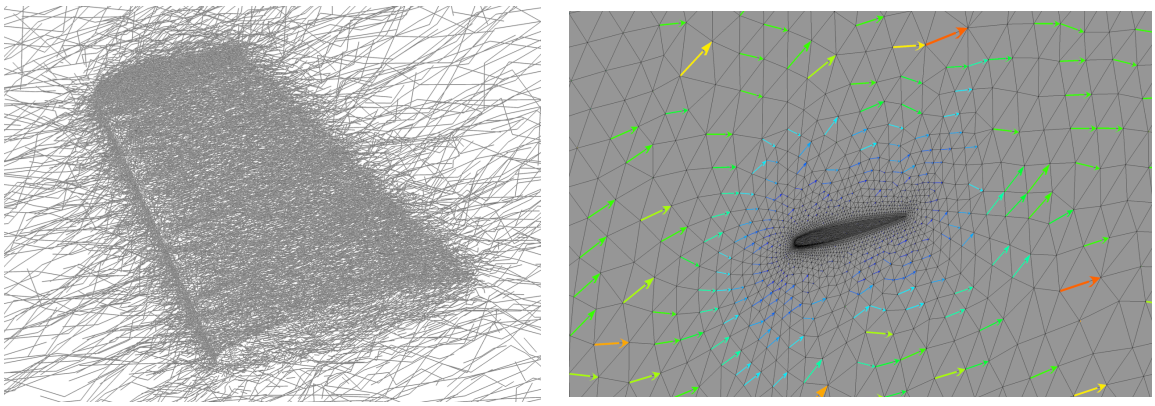


Figure 6.7: *Streamline linelets near the airfoil (left) and node-to-node vectors forming the streamlines (right).*

On the left-hand-side of Figure 6.8 we show the results for the convergence of the streamline linelet and diagonal preconditioners. The right-hand-side of the figure shows the convergence when we parallelize the problem with 10 and 50 processors in the case of using the streamline linelet in the Gauss-Seidel preconditioner case.

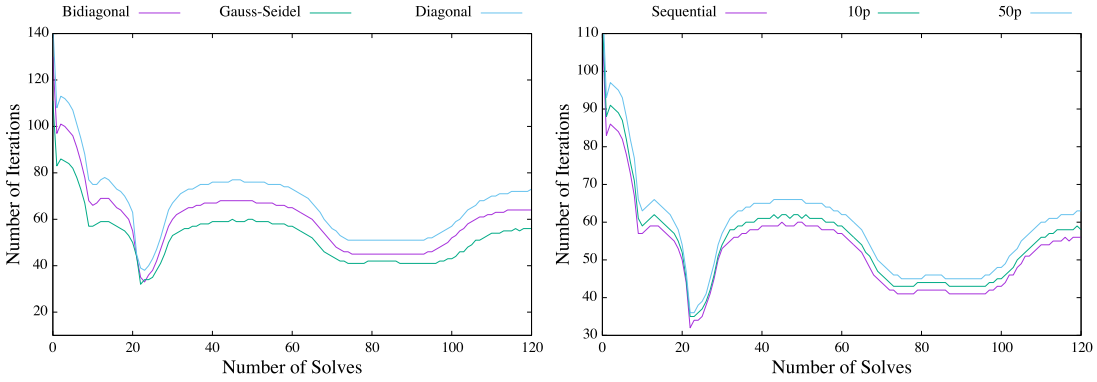


Figure 6.8: *Results for convergence in sequential (left) and paralllelization for the Gauss-Seidel (right).*

Remarks

- Figure 6.8 (left-hand-side) shows the convergence for the streamline linelet (Gauss-Seidel and bidiagonal cases) and for the diagonal preconditioners. The results obtained are similar to the ones obtained with the NACA0012 airfoil, this is, the streamline linelet in the Gauss-Seidel case is the one that converges in less iterations, followed by the streamline linelet in the bidiagonal case and finally the diagonal is the one that converges in more iterations. The Gauss-Seidel preconditioner provides a speedup of 1.4 with respect to the diagonal preconditioner in terms of total solver iterations.

- The gain is also limited compared to the scalar transport case studied in Chapter 4.

- The left-hand-side of Figure 6.8 shows the effect of the parallelization on the streamline linelet. To show this we have chosen the streamline linelet with the Gauss-Seidel algorithm and we have used 10 and 50 processors. As we can see, the algorithm looses efficiency as more processors are used, because we apply each preconditioner locally, meaning that the streamlines are cut.

6.3 Composition of Preconditioners: Incompressible Navier-Stokes Equations

This section tests the local preconditioning algorithms for the incompressible Navier-Stokes equations following the same approach as the one used in Chapter 4 for the case of the advection-diffusion equation. The solution of the incompressible Navier-Stokes equations is based on the segregation of the momentum and continuity equations. To summarize, the algorihtm extracts the pressure Schur complement from the monolithic Navier-Stokes algebraic system, and this Schur complement is solved using the Orthomin(1) method [53]. Eventually, at each time step, the momentum equation is solved twice and the continuity equation once. The preconditioners are applied here to the momentum equation, exclusively.

To carry out the study, we consider a NACA0012 airfoil similar to the one studied in Section 6.2. We first consider the sequential solution of a NACA0012, to compare the different preconditioners presented in Chapters 4 and 5. Then we study the same example to investigate the effects of the parallelization on the performance of one single preconditioner, the multiplicative streamline one.

6.3.1 NACA0012 Airfoil

In the previous section, we have used the streamline linelet algorithm to accelerate the convergence of the solvers in the case of having an inviscid compressible flow over two airfoils. In this example we consider the different composition approaches presented in this Chapter and we compare them with the anisotropy linelet and the streamline linelet approaches. The problem setup is the same as the one in Subsection 4.5.4 of

Chapter 4 but in this occasion the problem is solved as unsteady using the Euler scheme. Also we will consider two safety factors of 100 and 1000. The solver tolerance is set to 10^{-2} and the Krylov subspace dimension to 10.

Figure 6.9 shows the convergence for all the preconditioners studied (left) and the convergence for different relaxation parameters using the Mult. Streamline preconditioner (right). In all cases we choose a safety factor of 100.

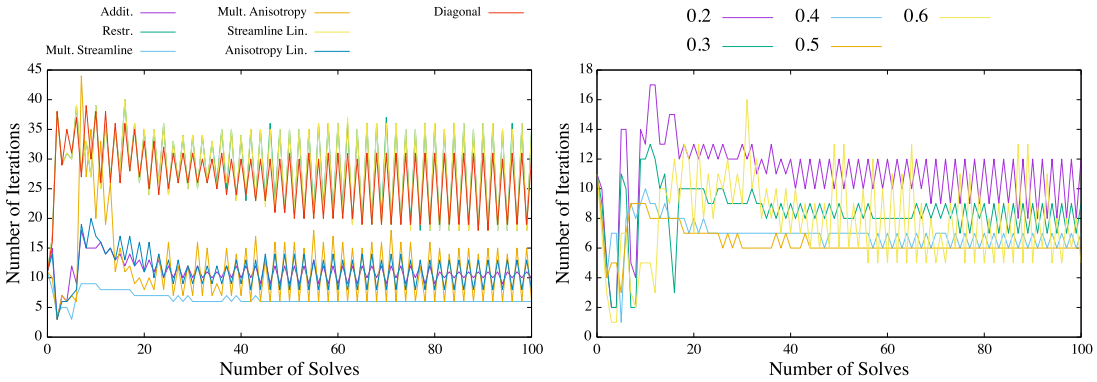


Figure 6.9: *NACA0012 NS: Results for convergence (left) and comparison of the relaxation parameter (right) for a factor of* 100.

We now consider a safety factor of 1000. We first point out that in some cases, the solution diverges if some preconditioners are activated from the first time step. This is a known problem when solving complex linear problems and when the tolerance is too tight during the first iterations. Thus we also consider using the diagonal preconditioner for the first 30 time steps (which corresponds to 60 momentum equations solves). Figure 6.10 shows the convergence for all the preconditioners studied (left) and the convergence for different relaxation parameters using the Mult. Streamline preconditioner (right).

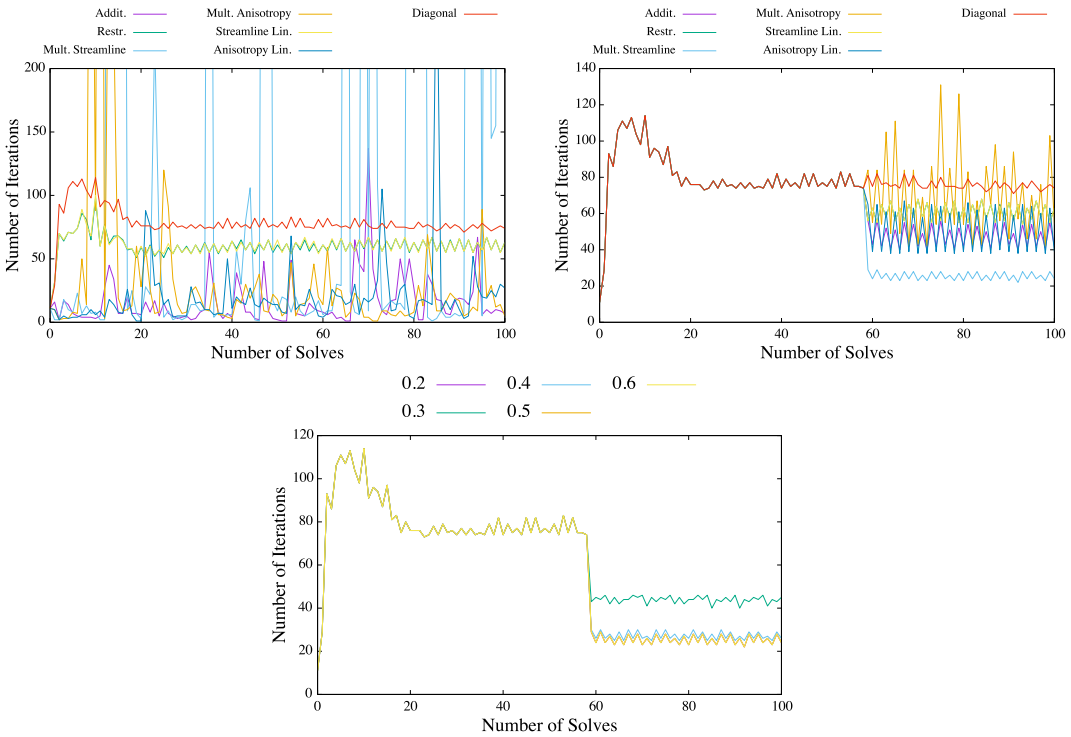


Figure 6.10: *NACA0012 NS: Convergences starting from the beginning (left), starting at time step 30 (right) and comparison of the relaxation parameter (bottom) for a factor of* 1000.

Remarks

- Figure 6.9 (left) shows the resuls for the convergence of all the preconditioners studied in this thesis using a safety factor of 100. In this case, Mult. Streamline is the one that the best performance.

- In Figure 6.9 (right) we compare the convergence of the Mult. Streamline preconditioner for different relaxation parameters. We observe that with $\omega = 0.5$, is where we have the best performance.

- Figure 6.10 (top) shows the results for the convergence for all the preconditioners using a safety factor of 1000. At the top left of the figure, we apply each preconditioner from the beginning. Here we notice that the Mult. Streamline preconditioner does not converge and that the Additive one is the one that has the best performance. At the

top right of the figure, we first start with 30 iterations of the diagonal preconditioner and then we apply the other preconditioners. In this case we see that the one that converges in less iterations is the Mult. Streamline one.

- In Figure 6.10 (bottom) we show the convergence results for different relaxation parameters taking the Mult. Streamline preconditioner. Again in this case we notice that the preconditioner has the best performance when we take $\omega = 0.5$.

- To conclude, the Mult. Streamline option gives the best results, with a speedup up to 5 compared to the diagonal preconditioner (for both safety factors). Also the number of iterations per time step is more stable than any other preconditioner and a relaxation parameter of 0.5 gives the best results.

6.3.2 NACA0012 Airfoil: parallelization

This last section studies the effect of the parallelization on the performance of the best preconditioner found in the previous example, namely the Multiplicative Streamline, with a relaxation of 0.5. To this end, we consider exactly the same problem and consider different partitioning using METIS, with 2, 3, 4, 5 and 6 subdomains. The safety factor is 100, the solver tolerance set to 10^{-2} and the Krylov subspace dimension to 10.

Figure 6.11 compares the number of iterations required to convergence, for the different partitionings, and compares them to the sequential version and the diagonal preconditioner.

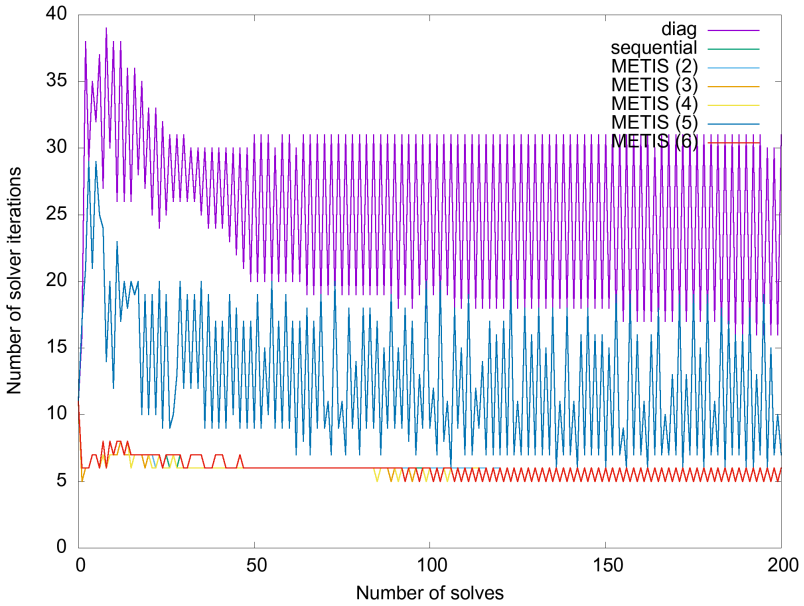


Figure 6.11: *NACA0012: effect of the parallelization on the number of solver iterations for different partitionings.*

Figure 6.12 shows the streamline linelets obtained for the last time step (50^{th}) when partitioning into 5 and 6 subdomains.

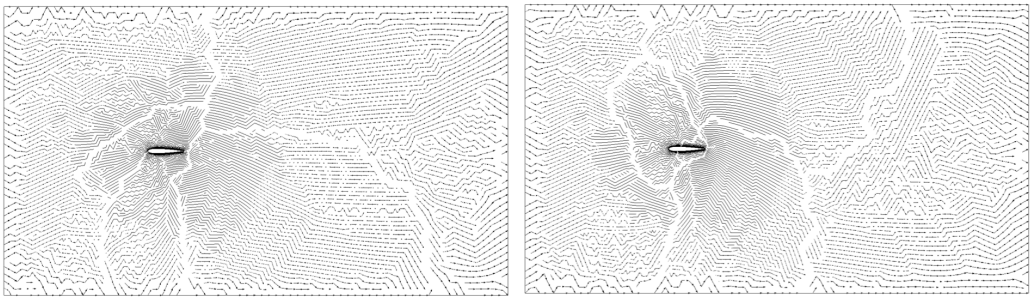


Figure 6.12: *NACA0012 NS: streamline linelets with 5 sudbomains (left) and 6 subdomains (right).*

Remarks

- In Figure 6.11 we observe a constant reduction of factor 5 with respect to the diagonal preconditioner, except for the case with 5 subdomains. This means that in this case, partitioning has a limited effect on the performance of the preconditioner.

- We cannot straightforwardly conclude why the streamline in the case of 6 subdomains is more favorable than in the case of 5. We thus cannot explain the loss of efficiency with 5 subdomains. Specific partitionings alter the performance (here using 5 subdomains), without a well-identified explaination.

6.4 Conclusions

In this chapter we have applied the preconditioners developed in Chapters 4 and 5 to the compressible Euler and the incompressible Navier-Stokes equations respectively. First we have tested the streamline linelet (both Gauss-Seidel and bidiagonal) preconditioner for a NACA0012 airfoil and an ONERA M6 airfoil using the compressible Euler equations. In both cases we observe that the convergence is accelerated but it has a minor impact compared to the case of the scalar transport. In the case of the ONERA M6 airfoil we notice that the streamline linelet preconditioner looses performance when parallelized. However, it still has a better performance if compared to the diagonal preconditioner. Then, we have tested the composition of preconditioners for solving the incompressible Navier-Stokes equations for a NACA0012 airfoil and studied the parallelization. In this case we show that the Mult. Streamline option is the one that has a better performance with a speed-up up to 5 if compared to the diagonal preconditioner. Regarding parallelization we observe that the partitioning has a limited effect on the performance of the preconditioner.

Chapter 7

Conclusions and Future Work

The less there is to justify a traditional custom, the harder it is to get rid of it

Mark Twain,
The Adventures of Tom Sawyer

In this chapter the conclusions reached in this dissertation together with the future work are presented. The main objective of this thesis was the developement of preconditioning techniques to accelerate the solution of algebraic systems, coming from the discretization of PDEs or systems of PDEs. To this end, the following sub-objectives were posed:

- Understanding and explaining the relations between the physics and the numerical methods in some given situations.
- Understanding the impact of the numerical method on the resulting algebraic system: stabilization, Péclet number, integration rule, mesh anisotropy for instance.
- Deriving preconditioning techniques based on these analyses.
- Estimating the gain by solving simple problems and studying the condition number of the preconditioned system.

- Validating our hypothesis by solving some practical examples.
- Understanding the effect of parallelization on the convergence properties of such preconditioners.

7.1 Achievements

Following the motivations and objectives set the following achievements have been obtained.

- By understanding the relations between the physics and the numerical methods, we have found that the order in which the mesh is renumbered in some well identified problems (strong advection or strong diffusion) can exhibit a particular structure (e.g. tridiagonal, lower diagonal etc.). In general, for advection-diffusion problems, these limiting behaviors are likely to co-exist.
- For simple cases, these structures depend on the stabilization technique, integration rule etc. used in each case. Hence, the stiffness matrix coming from an anisotropic mesh where the nodes have been ordered in the direction of anisotropy presents a tridiagonal structure if we use a close integration rule and no specific structure if we use an open integration rule. In the case of having a pure convection problem, using a SUPG stabilization technique leads to independent streamlines in the matrix in the direction of advection if we use a close integration rule, but no remarkable structure is observed if we use an open rule.
- We have also explored the different state-of-the art preconditioners, where we have chosen the Gauss-Seidel and bidiagonal preconditioners. Then by performing a renumbering along the advection direction or mesh anisotropy we have applied these preconditioning techniques

adapted to the structures observed (streamline linelet and anisotropy linelet preconditioners).

- We have calculated the condition number for a pure diffusive and a pure advective problem with SUPG stabilization and solved simple problems using the streamline linelet approach. From the results obtained we conclude that the matrix is more lower triangular and we confirm the positive impact of the streamline numbering.

- To validate our first results, we have built a composition of physics-based preconditioners and applied them to problems where different physics occur. Herein, when different regimes coexist the composition of preconditioners has proven to be a good strategy to accelerate convergence.

- Parallelizing the code with MPI to make it appropriate to an HPC environment has enabled us to solve more complex problems and to understand how the convergence of these preconditioners is affected when the code is run in parallel. From this, we conclude that, the preconditioners studied do not retain performance, although these has a limited effect on the performance of the preconditioners. Also, some specific partitions alter the performance of the preconditioners.

7.2 Future Work

Using physics-based preconditioners for large scale simulations remains an open problem. As future work we present different possibilities based on the results and the work done in this thesis.

In terms of convergence we have been successful in building physics-based preconditioners, but as we showed in our results they are still expensive in terms of convergence times. Efforts to make their implementation more efficient are required. Also, we have not presented the results of the timings for the preconditioner set-up. The algorithm costs should be studied in detail.

Concerning the streamline linelet renumbering combined with the Gauss-Seidel algorithm, we propose to study the impact of doing more than one iteration.

A deeper study of the mesh partition could be done, to understand why the case of the horizontal partition presented Subsection 4.5.4 is the one that exhibits a better performance in terms of convergence, opposite to what we thought in our initial hypothesis.

MPI has been chosen to parallelize the different problems targeting distributed-memory machines. As said in Chapter 4, parallelization has a counter-effect on the performance of the numbering algorithms, as it cuts the streamlines and thus the technique loses performance. One possibility to parallelize the Gauss-Seidel method would be to use OpenMP with a multi-coloring technique as the one presented in [80] to avoid race condition. In the case of the bidiagonal solver, one could parallelize the loop over the different linelets, as a node only belongs to one single linelet so that there is no race condition. By introducing OpenMP, we have the posibility to load balance our solution by using libraries such as DLB [44]. In fact, we may find that some compute nodes are overloaded while others are left inactive (e.g. due to the lack of linelets), which could explain the

relatively poor performance.

Finally, the main objective of this thesis has been to adapt mesh numbering to the physics of convection-dominated flows to improve the convergence of such problems. This mesh numbering in all cases has been done using a geometrical approach. As future work, the same could be done algebraically. This is, instead of finding a geometrical and physics-based condition (e.g. renumbering the mesh in the direction of the mesh anisotropy or renumbering the mesh along the direction of advection using the minimum angle criteria), finding a relationship between the coefficients of the assembled global matrix resulting from the discretization of the partial differential equation and from this condition reorder the mesh. A preliminary attempt of doing this was done at the end of this thesis using Gephi, an open-source network analysis and visualization software written in Java on the NetBeans platform [9], which gives the following result for a 2-D boundary layer problem. The input of Gephi is the matrix graph and coefficients coming from the discretization of the temperature equation assembled for a boundary layer flow with high Péclet number. The mesh considered is shown in Figure 7.1.

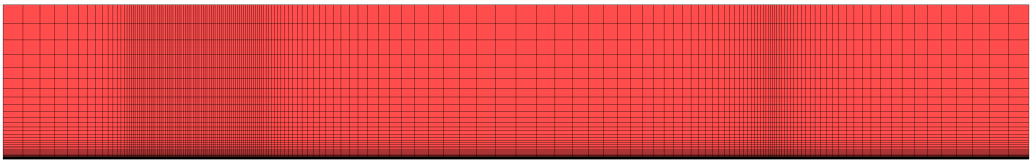


Figure 7.1: *Boundary layer mesh used to try Gephi.*

Figure 7.2 shows the groups given as output of Gephi, where the node are colored according to the assigned group.

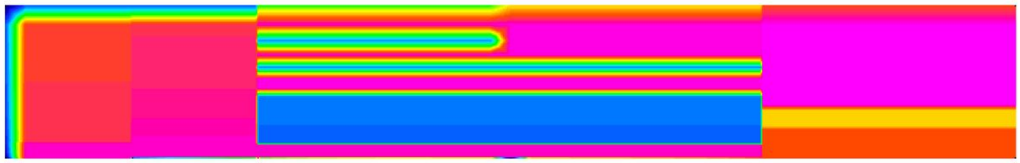


Figure 7.2: *Boundary layer problem renumbered with Gephi software.*

the pink color follows the zone of the mesh with higher anisotropy, including also the part of the core flow on the top right. The blue and the yellow colors in the core flow. The results show that the algorithm follows more the mesh than the physics in this case. It should be noted that a deeper analysis is required to understand deeper the relation between the groups and the stiffness matrix.

The result obtained in Figure 7.2 suggests that employing clustering techniques such as K-Means clustering [4, 60], Mean-shift clustering [22, 98] or Agglomerative hierarchical clustering [14, 30] could be a way to cluster the different physics existent in a given physical problem and once this clustering has done order the mesh in a convenient way to form streamline or anisotropy linelets.

List of Figures

List of Algorithms

List of Tables

Bibliography

[1] *An implicit, linelet-based solver for incompressible flows*, 1 1992. 98

[2] A massively parallel fractional step solver for incompressible flows. *Journal of Computational Physics*, 228(17):6316–6332, 2009. 5

[3] A.A. Dafa Alla, R.D. Harper, and M.M Gibson. Flat plate boundary layers. *Computation and Comparison of Efficient Turbulence Models for Aeronautics — European Research Project ETMA. Notes on Numerical Fluid Mechanics (NNFM)*, 5, 1998. 18, 19, 20

[4] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. *Electrical Engineering and Computer Science*, 42, 1997. 186

[5] P.R. Amestoy, S. de Ryhove, J.Y. L'Excellent, G. Moreau, and D. V. Shantsev. Efficient use of sparsity by direct solvers applied to 3d controlled-source em problems. *Computational Geosciences*, 23:1237–1258, 2019. 68

[6] K.E. Atkinson. *An Introduction to Numerical Analysis*. John Wiley & Sons, 1989. 37, 39

[7] R. Aubry, F. Mut, R. Löhner, and J.R. Cebral. Deflated preconditioned conjugate gradient solvers for the pressure–Poisson equation. *Journal of Computational Physics*, 227(14):10196–10208, 2008. 100, 143, 148

[8] Z.Z. Bai. Motivations and realizations of Krylov subspace methods for large sparse linear systems. *Journal of Computational and Applied Mathematics*, 1:71–78, 2015. 74

[9] M. Bastian, S. Heymann, and M. Jacomy. Gephi : An open source software for exploring and manipulating networks. *AAAI Publications, Third International AAAI Conference on Weblogs and Social Media*, 2011. 185

[10] A. Bejan. Convection heat transfer. *John Willey & Sons*, 2004. 18, 19, 20

[11] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182:418–477, 2002. 69, 90

[12] J. Berg and J. Nordström. Using the compressible navier-stokes equations as a model for heat transfer in solids. *Center for Turbulence Research*, 2011. 166

[13] R. Bhatia. Positive definite matrices. *Princeton Series in Applied Mathematics*, 2007. 59

[14] A. Bouguettaya, Q. Yu, X. Liu, X. zhou, and A. Song. Efficient agglomerative hierarchical clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(5):2785–2797, 2015. 186

[15] F. Brezzi and A. Russo. Stabilization techniques for the finite element method. *Applied and Insdustrial Mathematics*, (Venice-2):47–58, 1998. 31, 32

[16] A.N. Brooks and T.J.R. Hughes. A multidimensional upwind scheme with no crosswind diffusion. *Finite Element Methods for Convection Dominated Flows*, pages 19–35, 1979. 33

[17] A.N. Brooks and T.J.R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convective dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Eng.*, 32:199–259, 1982. 12, 33

[18] G. Capriz, V.N. Ghionna, and P. Giovine. *Modeling and Mechanics of Granular and Porous Materials*. Springer Science + Business Media, LLC, 2002. 38

[19] L. M. Carvalho, L. Giraud, and G. Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. Technical Report TR/PA/99/38, CRFACS, 1999. 144

[20] E. Casoni, A. Járusalem, C. Samaniego, B. Eguzkitza, P. Lafortune, D. D. Tjahjanto, X. Saáez, G. Houzeaux, and M. Vázquez. Alya: Computational solid mechanics for supercomputers. *Arch Computat Methods Eng*, 22:557–576, 2014. 5

[21] J. Cea. Approximation variationnelle des problèmes aux limites. *Annales de l'institut Fourier*, 14(2):345–444, 1964. 32

[22] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 1995. 186

[23] R. C. Y. Chin and T. A. Manteuffel. An analysis of block successive overrelaxation for a class of matrices with complex spectra. *SIAM J. Numer. Anal.*, 25:564–585, 1988. 98

[24] R. Codina. Comparison of some finite element methods for solving the diffusion-convection-reaction equation. *Comp. Meth. Appl. Mech. Eng.*, 156:185–210, 1998. 32, 34

[25] R. Codina and E. Oñate. The intrinsic time for the SUPG formulation using quadratic elements. *Comput. Methods Appl. Mech. Eng.*, 94(5):239–262, 1992. 35

[26] P. Córdoba, G. Houzeaux, and J.C. Cajas. Streamwise numbering for gauss-seidel and bidiagonal preconditioners in convection dominated flows. *PMAA16: The 9th International Workshop on Parallel Matrix Algorithms and Applications*, 2016. 109

[27] R. Courant, K.O. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. 38

[28] K. Davey, S. Bounds, I. Rosindale, and M.T. Alonso Rasgado. A coarse preconditioner for multi-domain boundary element equations. *Computers and Structures*, 80:643–658, 2002. 142

[29] T.A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006. 67

[30] W. H. E. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1:7–24, 1984. 186

[31] D.Bahuguna, V.Raghvendra, and B.V.R.Kumar. *Topics in sobolev spaces and applications*. Narosa Publishing House, 2002. 28

[32] H.A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 2(13):631–644, 2003. 75

[33] I. Duff, R.G. Grimes, and J.G. Lewis. Sparse matrix test problems. 15(1), 1989. 57

[34] H. C. Elman and M. P. Chernesky. Ordering effects on relaxation methods applied to the discrete one dimensional convection diffusion equation. *SIAM J. Numer. Anal.*, 30:1268–1290, 1993. 98

[35] Gregory Falkovich. *Fluid Mechanics*. Cambridge University Press, 2 edition, 2018. 167

[36] M. Ferronato. Preconditioning for sparse linear systems at the dawn of the 21st century: History, current developments, and future perspectives. *International Scholarly Research Notices*, 2012, 2012. 5

[37] S. Filippone, V. Cardellini, D. Barbieri, and A. Fanfarillo. Sparse matrix-vector multiplication on GPUs. *ACM Transactions on Mathematical Software*, 43(4):1–49, 2017. 57

[38] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. Alistair Watson, editor, *Numerical Analysis*, pages 73–89, Berlin, Heidelberg, 1976. Springer Berlin Heidelberg. 83

[39] W. Ford. Vector and matrix norms. *Numerical Linear Algebra with Applications*, pages 119–144, 2007. 62

[40] L.P. Franca, G. Hauke, and A. Masud. Revisting stabilized finite element methods for the advective-diffusive equation. *Comp. Meth. Appl. Mech. Eng.*, 195:1560–1572, 2006. 30, 31

[41] L.P. Franca, G. Hauke, and A. Masud. Stabilized finite element methods. *Finite Element Methods: 1970's and Beyond*, CIMNE,2003. 31

[42] Guillaume Houzeaux Frédéric Mogoules, François-Xavier Roux. Parallel scientific computing. *Wiley Computer Engineering Series*, 2015. 59, 62, 65, 66, 86, 88, 98, 99, 100, 120

[43] Roland W. Freund and Noël M. Nachtigal. An implementation of the QMR method based on coupled two-term recurrences. *SIAM Journal on Scientific Computing*, 15(2):313–337, 1994. 70

[44] M. Garcia, J. Corbalan, RM. Badia, and J. Labarta. A dynamic load balancing approach with SMPSuperscalar and MPI. In *Facing the Multicore - Challenge II*, volume 7174 of *Lecture Notes in Computer Science*, pages 10–23. Springer Berlin Heidelberg, 2012. 184

[45] A. Ghai and X. Jiao C. Lu. Fundamentals of the finite element method for heat and fluid flow. *John Wiley & Sons Ltd*, 2004. 59

[46] A. Ghai, C. Lu, and X. Jiao. A comparison of preconditioned Krylov subspace methods for large-scale nonsymmetric linear systems. *Numerical Linear Algebra with Applications*, 26(1):e2215, 2019. 170

[47] L. Giraud and R.S. Tuminaro. Algebraic domain decomposition preconditioners. *Technical Report ENSEEIHT-IRIT RT/APO/06/07*, October 2006. 94, 142

[48] P. Gosselet and C. Rey. Non-overlapping domain decomposition methods in structural mechanics. *Archives of Computational Methods in Engineering*, 13(4):359–392, 2007. 94

[49] D. Grewe and A. Lokhmotov. Automatically generating and tuning gpu code for sparse matrix-vector multiplication from a high-level representation. In *GPGPU-4*, 2011. 57

[50] L. Hageman and D. Young. Chapters 4, 5 and 9. In L. Hageman and D. Young, editors, *Applied Iterative Methods*, pages 209–258. Academic Press, San Diego, 1981. 70

[51] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau Standards*, 49:409–436, 1952. 75, 78

[52] R. Horn and C.R. Johnson. Matrix analysis (2nd. edition). *Cambridge University Press*, 2013. 59

[53] G. Houzeaux, R. Aubry, and M. Vázquez. Extension of fractional step techniques for incompressible flows: The preconditioned Orthomin(1) for the pressure Schur complement. *Computers & Fluids*, 44:297–313, 2011. 174

[54] G. Houzeaux, R. Borrell, Y. Fournier, M. Garcia-Gasulla, J.H. Göbbert, E. Hachem, V. Mehta, Y. Mesri, H. Owen, and M. Vázquez. *High-Performance Computing: Dos and Don'ts*. 02 2018. 86

[55] G. Houzeaux and R. Codina. A Chimera method based on a Dirichlet/Neumann(Robin) coupling for the Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 192(31-32):3343–3377, 2003. 93

[56] T.J.R. Hughes. Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. *Comp. Meth. Appl. Mech. Eng.*, 127(1):387–401, 1995. 33

[57] T.J.R. Hughes, L.P. Franca, and M. Balestra. A new finite element formulation for computational fluid dynamics: VII: The Galerkin/least-squares method for advective-diffusive equations. *Comp. Meth. Appl. Mech. Eng.*, 73:173–189, 1989. 33

[58] V. John and P. Knobloch. On spurious oscilations at layers diminishing (SOLD) methods for convextion-diffusion equations: Part i - a review. *Comp. Meth. Appl. Mech. Eng.*, 196:2197–2215, 2007. 35

[59] C. Johnson. Numerical solution of partial differential equations by the finite element method. *Cambridge University Press*, 1987. 33, 34

[60] T. Kanungo, D.M. Mount, N.S. Netanyahuand C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002. 186

[61] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998. 132

[62] P. Knobloch. On the choice of the SUPG parameter at outflow boundary layers. *Advances in Computational Mathematics*, 31:369–389, 2008. 35

[63] P. Knobloch. On the definition of the SUPG parameter. *Electronic transactions on numerical analysis ETNA*, 32:79–89, 2008. 35

[64] N. Kushida. Condition number estimation of preconditioned matrices. *SIAM J. Numer. Anal.*, 10(6):e0130920, 2015. 100

[65] N. Kushida. Impinging jets – a short review on strategies for heat transfer enhancement. *SIAM J. Numer. Anal.*, 10(6):e0130920, 2015. 21

[66] A. Kuzmin. On the lambda-shock formation on ONERA M6 wing. *International Journal of Applied Engineering Research*, 9:7029–7038, 2014. 171

[67] C. Lanczos. Solution of systems of linear equations by minimized iterations. *Journal of Research of the National Bureau of Standards*, (49):33–53, 1952. 83

[68] U. Langer and M. Neumüller. *Direct and Iterative Solvers*, pages 205–251. Springer International Publishing, Cham, 2018. 66, 67

[69] S. loisel and H. Nguyen. A comparison of additive Schwarz preconditioners for parallel adaptive finite elements. *Domain Decomposition Methods in Science and Engineering XXII*, 104:345–354, 2016. 142

[70] M. Mani, J. Ladd, A. Cain, R. Bush, M. Mani, J. Ladd, A. Cain, and R. Bush. An assessment of one- and two-equation turbulence models for internal and external flows. *American Institute of Aeronautics and Astrounautics*, 9, 1997. 171

[71] H. Martin. Heat and mass transfer between impinging jets. *Advances in Heat Transfer*, 13:1–60, 1977. 21

[72] Ryan G. McClarren. Chapter 16 - Gauss quadrature and multidimensional integrals. In Ryan G. McClarren, editor, *Computational Nuclear Engineering and Radiological Science Using Python*, pages 287–299. Academic Press, 2018. 40

[73] K.W. Morton and D.F. Mayers. Numerical solution of partial differential equations, an introduction. *Cambridge University Press*, 2005. 4

[74] R. Olazábal, O. Lehmkuhl, G. Houzeaux, and R. Borell. Algebraic linelet preconditioner for the solution of the poisson equation on boundary layer flows. Nice, France, 2021. 117

[75] S. P. Oliveira, A. L. Madureira, and F. VAlentin. Weighted quadrature rules for finite element methods. *Journal of Computational and Applied Mathematics*, 227(1):93–101, 2009. 39

[76] O.Pirenneau. The finite element method for hyperbolic systems. *Finite Elements. ICASE/NASA LaRC Series.*, pages 67–93, 1988. 14

[77] G.A. Rao, Y. Levy, and M. Kitron-Belinkov. Heat transfer characteristics of a multiple jet impingement system. *48th Israeli Aerospace Conference*, 13:1–60, March 1977. 21

[78] Y. Saad and M.H. Shultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. SCI STAT. COMPUT*, 1986. 75, 80, 81, 92

[79] Yousef Saad. *Iterative Methods for Sparse Linear Syatems*. SIAM, 2004. 4, 5, 57, 63, 69, 75, 82, 83, 90, 92, 100, 149

[80] Y. Sato, T. Hino, and K. Ohashi. Parallelization of an unstructured Navier–Stokes solver using a multi-color ordering method for OpenMP. *Computers & Fluids*, 88:496–509, 2013. 184

[81] V. Schmitt and F. Charpin. Pressure distributions on the ONERA M6 wing at transonic mach numbers. Technical Report Report of the Fluid Dynamics Panel. Working Group 4, AGARD AR 138, NATO-Experimental Data Base for Computer Program Assessment, 1979. 171

[82] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. *School of Computer Science Carnegie Mellon University Pittsburgh*, 1994. 75, 76, 77, 78, 92

[83] B. Smith, P. Bjorstad, and W. Gropp. Domain decomposition: Parallel multilevel methods for eliptic partial differential equations. *Cambridge University Press*, pages 19–23, March 2004. 96, 142

[84] P. Sonneveld. CGS, a fast lanczos-type solver for nonsymmetric linear systems. *SIAM. Journal on Scientific and Statistical Computing*, 10(1):36–52, 1989. 70, 83

[85] O. Soto, R. Löhner, and F. Camelli. A linelet preconditioner for incompressible flows. *International Journal of Numerical Methods for Heat and Fluid Flow*, 13(1):133–147, 2003. 97, 98, 117, 150

[86] A. St-Cyr, S. Thomas, and M. Gander. Multiplicative, additive and restricted additive optimized Schwarz preconditioning. *SIAM Journal on Scientific Computing*, 29(6):2402–2425, 2007. 142

[87] P.T. Stathis. *Sparse Matrix Vector Processing Formats*. Technische Universiteit Delft, 2004. 57

[88] J. L. Stensby. Analytical and computational methods (chapter 4 - matrix norms), Fall 1998. 63

[89] W.A. Strauss. *Partial Differential Equations. An Introduction.* Wiley, United States of America, 2008. 14

[90] H.S. Tang, R.D. Haynes, and G. Houzeaux. A review of domain decomposition methods for simulation of fluid flows: Concepts, algorithms, and applications. *Archives of Computational Methods in Engineering*, 80:643–658, 2020. 142

[91] V. Thomeé. From finite differences to finite elements. a short history of numerical analysis of partial differential equations. *Journal of Computational and Applied Mathematics*, 128(1-2):1–54, 2001. 47

[92] A. Toselli and O.Widlund. Domain decomposition methods-algorithms and theory. *Springer*, 2005. 95, 96

[93] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. s. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992. 83

[94] R. Varga. *Basic Iterative Methods and Comparison Theorems*, pages 63–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. 70

[95] M. Vargas-Felix and S. Botello. Parallel direct solvers for finite element problems. *Comunicaciones del CIMAT*, 10 2000. 67

[96] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, AJ. guado Sierra, D. Mira, H. Calmet, F. Cucchieti, H. Owen, A. Taha, and J.M. Cela. Alya: Towards exascale for engineering simulation codes. *International Supercomputing Conference*, April 2014. 5, 88

[97] X. Wang, G.F. Naterer, and E. Bibeau. Convective heat transfer from a NACA airfoil at varying angles of attack. *Journal of Thermophysics and Heat Transfer*, 22(3):457–463, 2008. 157

[98] C. Xiao and M. Liu. Efficient mean shift clustering using gaussian kd tree. *Computer Graphics Forum*, 29(7):2065–2073, 2010. 186

[99] D. Young. Chapters 3 and 4 - linear stationary iterative methods and convergence of the basic iterative methods. In D. Young, editor, *Iterative Solution of Large Linear Systems*, pages 106–139. Academic Press, 1971. 70

[100] D.M. Young and R.T. Gregory. *A survey of numerical mathematics*. Dover Publications, 1988. 39