



UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

Escola Superior d'Enginyeries Industrial,  
Aeroespacial i Audiovisual de Terrassa

# Study of machine learning techniques for accelerating finite element simulations of Stokes flows

**Document:** Report

**Author:**

Ferran de Miguel Blasco

**Director/Co-director:**

Joaquín Alberto Hernández Ortega  
Anastasios Drougkas

**Degree:**

Bachelor in Aerospace Technology Engineering

**Examination session:**

Spring

BACHELOR FINAL THESIS



# Acknowledgements

First and foremost I want to thank my advisor Joaquín Hernández for always being available to answer my doubts. His guidance and his patience have been crucial for the development of this thesis.

Secondly, I want to thank my family for their unconditional support and for listening to me (or at least pretending to) think out loud whenever I get stuck trying to get some result, which happens more often than I would like to admit.

---

*"Oh! I have slipped the surly bonds of Earth  
And danced the skies on laughter-silvered wings;  
Sunward I've climbed, and joined the tumbling mirth  
of sun-split clouds,—and done a hundred things  
You have not dreamed of—wheeled and soared and swung  
High in the sunlit silence. Hov'ring there,  
I've chased the shouting wind along, and flung  
My eager craft through footless halls of air."  
- John Gillespie Magee Jr.  
Letter to parents (September 3, 1941) ([1])*



# Contents

List of Figures	VI
List of Tables	VIII
List of algorithms	IX
Abbreviations	X
Nomenclature	XI
Abstract	1
<b>1 Introduction</b>	<b>2</b>
1.1 Objective . . . . .	2
1.2 Scope . . . . .	2
1.3 Requirements . . . . .	3
1.4 Justification . . . . .	3
1.5 Schedule . . . . .	3
1.6 State of the art . . . . .	4
1.6.1 Finite Element Method . . . . .	4
1.6.2 Machine learning techniques . . . . .	5
<b>2 Finite Element Method for incompressible steady state Navier-Stokes equations</b>	<b>6</b>
2.1 Strong form . . . . .	6
2.2 Weak form . . . . .	7
2.3 Finite element formulation . . . . .	8
2.4 Matrix problem . . . . .	10
2.4.1 Solvability conditions . . . . .	14
2.5 Element point of view . . . . .	16
2.5.1 Elemental matrices calculations . . . . .	17
2.6 Picard Iteration . . . . .	21
<b>3 Vectorization and code optimization</b>	<b>23</b>

3.1	Naive assembly . . . . .	23
3.2	Vectorization . . . . .	25
3.3	Time savings . . . . .	26
3.4	Profiling . . . . .	27
<b>4</b>	<b>Reduced order modeling</b>	<b>28</b>
4.1	Parametric problem . . . . .	28
4.2	Solution space . . . . .	29
4.3	Training stage . . . . .	30
4.4	Dimensionality reduction . . . . .	30
4.5	Angle between subspaces . . . . .	31
4.6	Projection onto the reduced space . . . . .	31
<b>5</b>	<b>Results</b>	<b>33</b>
5.1	Stokes flow simulations . . . . .	33
5.1.1	Lid driven cavity . . . . .	33
5.1.2	Flow around a cylinder . . . . .	35
5.2	Navier-Stokes simulations . . . . .	36
5.2.1	Lid driven cavity . . . . .	37
5.2.2	Flow around a cylinder . . . . .	43
5.2.3	NACA 0012 . . . . .	46
5.3	Reduced order modeling results . . . . .	49
5.3.1	Lid driven cavity . . . . .	50
5.3.2	Flow around a cylinder . . . . .	51
5.3.3	NACA 0012 . . . . .	53
<b>6</b>	<b>Environmental impact</b>	<b>56</b>
<b>7</b>	<b>Conclusions</b>	<b>57</b>
	<b>Bibliography</b>	<b>58</b>
<b>A</b>	<b>Profiling results</b>	<b>60</b>
<b>B</b>	<b>Matlab code</b>	<b>69</b>
<b>C</b>	<b>Meshing process</b>	<b>70</b>
<b>D</b>	<b>Kratos simulations</b>	<b>73</b>
<b>E</b>	<b>Principal angles combinations</b>	<b>75</b>
E.1	Lid driven cavity . . . . .	75
E.2	Cylinder . . . . .	77

E.3 NACA0012 . . . . . 77

# List of Figures

1.1	Gantt chart	4
1.2	CFD examples	5
2.1	Domain discretization	9
2.2	2D velocity and pressure interpolations	15
2.3	Coordinate transformation	19
3.1	Assembly visualization	25
5.1	Lid driven cavity case description	33
5.2	Lid driven Stokes comparison	34
5.3	Lid driven Stokes results	34
5.4	Flow around cylinder case description	35
5.5	Cylinder Stokes comparison	35
5.6	Cylinder Stokes results	36
5.7	Lid driven Navier-Stokes comparison Re=100	37
5.8	Lid driven Navier-Stokes comparison Re=100 pressure	38
5.9	Lid driven Navier-Stokes results Re=100	38
5.10	Lid driven Navier-Stokes comparison Re=400	39
5.11	Lid driven Navier-Stokes comparison Re=400 pressure	40
5.12	Lid driven Navier-Stokes results Re=400	40
5.13	Lid driven Navier-Stokes comparison Re=1000	41
5.14	Lid driven Navier-Stokes comparison Re=1000 pressure	41
5.15	Lid driven Navier-Stokes results Re=1000	42
5.16	Lid driven Navier-Stokes results Re=1000 Pressure	42
5.17	Cylinder Navier-Stokes comparison Re=50	43
5.18	Cylinder Navier-Stokes comparison Re=50 pressure	44
5.19	Cylinder Navier-Stokes results Re=50	45
5.20	Cylinder Navier-Stokes results Re=50 pressure	46
5.21	NACA 0012 case definition	47
5.22	NACA 0012 results Re=200	47

5.23	NACA 0012 results $Re=500$	48
5.24	NACA 0012 results $Re=500$ zoomed	49
5.25	Lid driven velocity modes	50
5.26	Singular values lid driven	51
5.27	Cylinder velocity modes	52
5.28	Singular values cylinder	53
5.29	Airfoil NACA0012 velocity modes	54
5.30	Singular values NACA 0012	55
C.1	NURBS surface	70
C.2	Example mesh	71
D.1	Kratos gui	73
D.2	Kratos guide results	74



# List of Tables

3.1	Performance improvements	26
6.1	CO <sub>2</sub> emissions	56

# List of Algorithms

1	Picard iteration method . . . . .	22
2	Assembly process . . . . .	24
3	Principal angles calculation . . . . .	31

# List of abbreviations

Abbreviation	Meaning
CFD	Computer Fluid Dynamics
DNS	Direct Numerical Simulation
DOF	Degrees of Freedom
FDM	Finite Difference Method
FEM	Finite Element Method
FVM	Finite Volume Method
LBB	Ladyzhenskaya Babuška Brezzi
NS	Navier-Stokes
PDE	Partial Differential Equation
POD	Proper Orthogonal Decomposition
SVD	Singular Value Decomposition
VMS	Variational MultiScale



# Nomenclature

$\mathbf{x}$	Physical coordinates $x, y$
$\mathbf{X}$	Set of all nodal coordinates
$\xi$	Elemental coordinates: $\xi, \eta$
$\Gamma$	Boundary of the problem $\Gamma = \Gamma_{\sigma} \cup \Gamma_{\mathbf{u}}$
$\Gamma_{\sigma}$	Domain with Von Neumann boundary conditions
$\Gamma_{\mathbf{u}}$	Domain with Dirichlet boundary conditions
$\bar{\Omega}^e$	Domain of element $e$
$\Omega$	Domain of the problem
$\hat{n}_{el}$	Total number of pressure elements
$\hat{n}_{nodE}$	Number of pressure nodes per element
$\mathbf{n}_{sd}$	Number of dimensions of the problem, generally 2
$n_m$	Number of Gauss points
$n_{nod}$	Total number of velocity nodes
$\hat{n}_{nod}$	Total number of pressure nodes
$n_{stp}$	Number of steps
$n_{el}$	Total number of velocity elements
$n_{nodE}$	Number of velocity nodes per element
$\hat{\mathbf{n}}$	Unit outward normal vector
$\nabla$	Nabla operator $\left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right)$
$\mathbf{J}$	Jacobian matrix
$\frac{D\mathbf{X}}{Dt}$	Material derivative: $\frac{\partial \mathbf{X}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{X}$
$\nabla^s \mathbf{X}$	Symmetric gradient operator: $\frac{\nabla \mathbf{X} + (\nabla \mathbf{X})^T}{2}$

---

$\mathbf{I}$	Identity matrix
$\nabla^2$	Laplacian operator
$\mathcal{L}^p$	Lebesgue function space of order $p$
$\mathcal{H}^p$	Sobolev vector space of functions of order $p$
$\mathbf{b}$	Volumetric forces
$\bar{\mathbf{t}}$	Tractions $\bar{\mathbf{t}} = \boldsymbol{\sigma} \hat{\mathbf{n}}$
$\lambda$	Volume viscosity
$\mu$	Dynamic viscosity
$\nu$	Fluid kinematic viscosity, $\nu = \frac{\mu}{\rho}$
$\rho$	Fluid density
$\boldsymbol{\sigma}$	Stress tensor
$\mathbf{u}$	Fluid velocity $(u_1, u_2, u_3)$
$p$	Fluid absolute pressure
$\mathbf{N}_A$	Shape function at point A for velocity nodes
$\mathbf{l}$	Set of unconstrained degrees of freedom
$\mathbf{r}$	Set of constrained degrees of freedom
$\hat{N}_A$	Shape function at point A for pressure nodes
$\bar{\mathbf{u}}$	Vector of prescribed velocities
$\mathbf{C}$	Global convection matrix
$\mathbf{F}$	Global external force vector
$\mathbf{G}$	Global gradient operator matrix
$\mathbf{K}$	Global viscosity matrix
$\mathbf{M}$	Global mass matrix
$\delta$	Convergence tolerance
$\psi$	Relaxation factor
$\boldsymbol{\alpha}(\zeta)$	Matrix containing the coefficients for $\boldsymbol{\Theta}$
$\mathbf{A}^u$	Snapshot matrix
$\Psi$	Basis matrix approximating the solution space for pressure
$\Phi$	Basis matrix approximating the solution space for velocity

---

$\chi$	Matrix of coefficients for the approximation of pressure
$\Theta$	Matrix containing columns representing spatial patterns of the external force vector $\mathbf{F}$
$\zeta$	Vector of input parameters
$\kappa$	Matrix of coefficients for the approximation of velocity
$\mathbf{D}^u$	Matrix containing columns representing spatial patterns of the prescribed displacement $\bar{\mathbf{u}}$
$\beta^u(\zeta)$	Matrix containing the coefficients for $\mathbf{D}^u$
$\mathbf{B}^e$	Elemental velocity shape functions derivatives matrix
$\hat{\mathbf{B}}^e$	Elemental pressure shape functions derivatives matrix
$\hat{\mathbf{N}}^e$	Elemental pressure shape functions matrix
$\mathbf{N}^e$	Elemental velocity shape functions matrix
$\mathcal{B}$	Stacked $\mathbf{B}$ matrix
$\mathcal{C}$	Global matrix of viscosity
$\mathcal{I}$	Global matrix of the mapping from pressure to velocity
$\mathcal{N}$	Stacked $\mathbf{N}$ matrix
$\mathcal{U}$	Stacked velocities matrix
$\mathcal{W}$	Matrix of Jacobians and Gauss weights.
$\mathbf{c}_A$	Vector of variations
$\mathbf{d}$	Global velocity vector
$\mathbf{v}$	Test functions for velocity
$\mathbf{u}^h$	Finite dimensional space of velocity trial functions
$\mathbf{v}^h$	Finite dimensional space of velocity test functions
$p^h$	Finite dimensional space of pressure trial functions
$q$	Test functions for pressure
$q^h$	Finite dimensional space of pressure test functions

# Abstract

[EN] This project starts by studying the finite element method for the steady state Navier-Stokes equations, afterwards it is implemented in Matlab and optimized via *Vectorization* achieving up to 5000x speed-up in some calculations. Then a reduced order model is studied to decrease the computational time of performing different simulations with slight modifications to the input parameters. Finally, the results obtained are compared against an already tested FEM code, *Kratos Multiphysics*, and against literature, and the performance of the developed solver for the equations is analyzed. It has been observed that the results obtained with the present work's solver are almost equal to those made by the reference alternatives.

Keywords: Finite element method, Computational fluid dynamics, Reduced Order Modelling, Coding, Fluid mechanics, Machine learning, Partial differential equations

[CA] Aquest projecte s'inicia estudiant el mètode dels elements finits per les equacions de Navier-Stokes en règim estacionari. Després s'implementa en Matlab i mitjançant *Vectorització* s'accelera el codi. Més endavant s'estudia la implementació d'un model d'ordre reduït per disminuir el cost computacional de realitzar diverses simulacions amb lleugers canvis als paràmetres d'entrada. Finalment, els resultats es comparen contra un codi validat d'elements finits, *Kratos Multiphysics*, contra resultats de la literatura i s'analitza el rendiment del solver. S'ha observat que els resultats del programa desenvolupat són gairebé iguals als de les altres alternatives.

Paraules clau: Mètode dels elements finits, Dinàmica de fluids computacional, Modelització d'ordre reduït, Programació, Mecànica de fluids, Aprenentatge de màquina, Equacions diiferencials en derivades parcials

# Chapter 1

## Introduction

### 1.1 Objective

The aim of this thesis is twofold, firstly to develop a Matlab Finite Element code able to solve steady state Navier-Stokes flows given some initial boundary conditions. Secondly, calculate different training cases and compare them to later develop a reduced order model and increase the simulation's performance.

### 1.2 Scope

This project will include:

- Derivation of Navier-Stokes' equations weak form.
- Finite element matrix formulation of Stokes equations.
- Development of a steady state Navier-Stokes Finite Element Matlab code.
- Validation and verification of the solver against available data.
- Generation of comparison data with *Kratos*, if needed.
- Optimization of the code via vectorization of the assembly process.
- Pre and post-processing functions to generate meshes and view the results in GiD.
- Development of machine learning training sets.
- Development of a reduced order model from the training data.
- Study of the feasibility of these techniques for accelerating Computational Fluid Dynamics.

The project will not include:

- The use of meshes with elements different than Q2Q1.
- The use of the variational multiscale method.

- The study of different stabilization schemes for solving finite elements CFD for elements that do not satisfy the LBB condition.
- The implementation for solving transient flows.
- The implementation of the code in a high performing programming language such as C.

## 1.3 Requirements

The restrictions and initial requirements are:

- The FEM code must be able to solve Stokes flow problems.
- The solutions must be verified against trusted data.
- The code must be accelerated by statistical techniques.
- Computer with Matlab, GiD and python installed.
- Knowledge on Mixed FEM, machine learning and statistics.
- The code must be well structured so that future users can use it or improve it with ease.

## 1.4 Justification

CFD studies are of utmost importance in the aeronautical industry as it is a quick and cheap method to gather data compared to experimental methods. Different approaches are available, such as Finite Difference Method, Finite Volume Method and Finite Element Method (FEM). It has been chosen to perform a FEM simulation because it gives more flexibility when meshing irregular geometries, although its mathematical formulation is more complex. Also it is well suited to analyze interfaces between different materials, for instance liquid-solid or liquid-liquid. The motivation behind accelerating FEM with machine learning and other statistical techniques is that finding the solution is computationally expensive in terms of both computer memory and execution time. Having the capacity of finding solutions faster with enough precision is critical for rapid prototyping in the aerospace industry.

## 1.5 Schedule

The tasks to be done are:

Task ID	Description
1.1	Research about FEM, specially for fluids.
1.2	Research about machine learning for accelerating simulations.
2.1	Create a Matlab code capable of solving Stokes flow.
2.2	Optimize the code and make it compliant to ISO 5055.
2.3	Generate the meshes and modify the code to import and export meshes to and from GiD.
2.4	Validate the code against some typical example cases, such as lid-driven cavity or flow around a cylinder.
3.1	Implement machine learning techniques for accelerating simulations.
3.2	Validate these techniques against some typical example cases, such as lid-driven cavity or flow around a cylinder.
3.3	Compare the results obtained by direct simulation and machine learning.
4.1	Elaborate the project charter.
4.2	Elaborate the report.
4.3	Elaborate the budget.

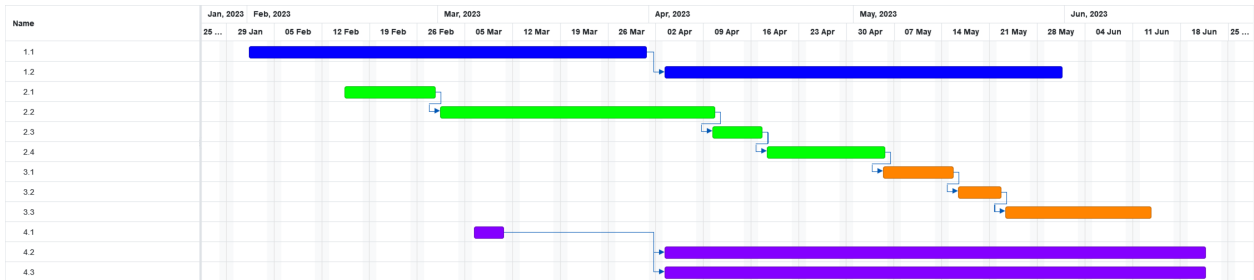


Figure 1.1: Gantt chart

## 1.6 State of the art

In this section some of the techniques for dealing with FEM and machine learning applied to CFD will be studied.

### 1.6.1 Finite Element Method

There are many variations to the FEM each one with its pros and cons. Some examples of the variations are: the discontinuous Galerkin method, mixed element methods, spectral methods and the variational multiscale approach. Only the latter will be studied, as the advantages it provides are of particular interest to incompressible fluids, the object of this thesis.

Since its introduction in the 90s the variational multiscale (VMS) framework has been applied to design stabilized finite element methods in problems where the standard Galerkin method stability is not ensured.

This can be caused by two main reasons. First, there are problems that require extremely fine meshes so the computational cost is unaffordable. The best example of this is a convection dominated problem. Second, in mixed problems such as Stokes flow, where compatibility conditions (LBB condition) must arise to stabilize the solution [2]. The VMS method decomposes the problem into resolvable and subgrid scales. The latter can be approximated to end up with a stable finite element problem that can deal with convective dominated flows and the use of the same pressure-velocity interpolations [3]. Although the VMS method explored here is thought mainly for fluids it can deal with any particular problem such as the Helmholtz equation, of particular interest in acoustics and electromagnetics [4]. A relevant example of a code that uses VMS is *Kratos* which uses the VMS method to solve fluid problems [5], [6], [7].

### 1.6.2 Machine learning techniques

The main drawback of CFD is the computational cost. A method to reduce the cost is to use machine learning to *compress* the data and obtain a reduced model which is cheaper and faster to simulate yet represents the original large-scale simulation [8].

The first example of such techniques that will be analyzed is the aeroelastic study of an F-16 fighter jet. Flutter prediction is essential to develop high performance and safe aircraft designs. To compute flutter at transonic regimes where the flow is nonlinear, CFD studies must be performed and coupled with the plane's structure, thus resulting in very costly simulations. However, to analyze the jet's performance at varying input parameters, a reduced order model can be used so as not to perform the whole CFD study of all the possible configurations. One method to obtain a reduced order model is to use a proper orthogonal decomposition on the solution, which gives a basis matrix into which the governing equations can be projected. Then, as this approach is limited by the variation of the free-stream Mach number, an interpolation of the angles between two POD subspaces is performed [9].

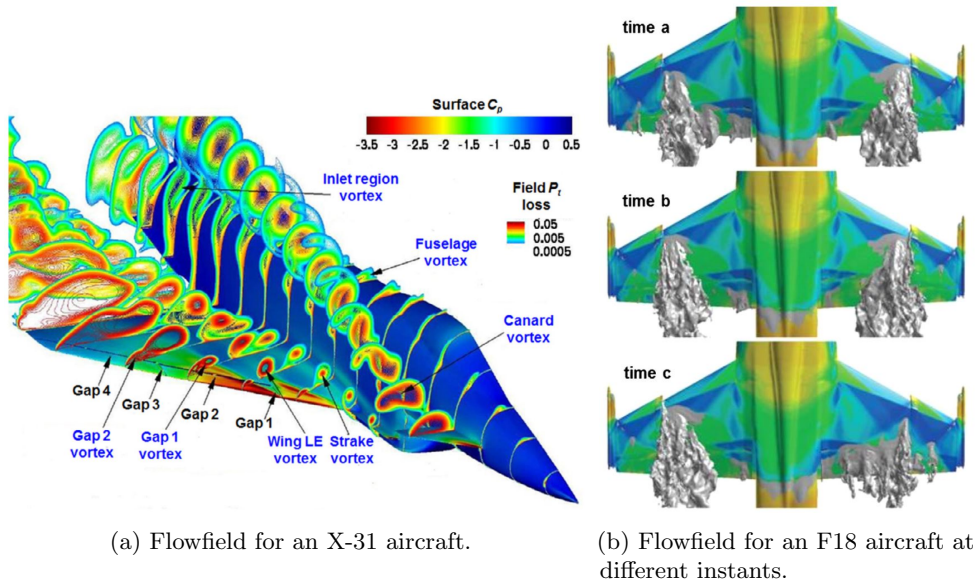


Figure 1.2: Examples of CFD over two different aircraft from [10].



## Chapter 2

# Finite Element Method for incompressible steady state Navier-Stokes equations

This chapter will begin with the strong form of the steady Navier-Stokes equations, then the weak form will be derived. Afterwards, the equations will be discretized and formulated in terms of the FEM, where the matrix problem of the NS equations will be found. Subsequently, the solvability of the equations will be discussed, as well as the obtention of said matrices. Finally, as the system of equations to be solved is nonlinear, a method based on Picard iteration will be presented to solve it.

### 2.1 Strong form

The starting point are the Navier-Stokes equations formulated in differential form which govern fluid motion:

Mass conservation (or continuity):

$$\frac{\partial \rho}{\partial t} + \rho \nabla \cdot \mathbf{u} = 0 \quad (2.1)$$

Momentum conservation:

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \rho \frac{D\mathbf{u}}{Dt} \quad (2.2)$$

Constitutive equation for a newtonian fluid:

$$\boldsymbol{\sigma} = -p\mathbf{I} + \lambda \nabla \cdot \mathbf{u} \mathbf{I} + 2\mu \nabla^s \mathbf{u} \quad (2.3)$$

Where:

- $\rho$  is the fluid's density
- $\mathbf{u}$  is the fluid's velocity

- $\boldsymbol{\sigma}$  is the fluid's stress tensor
- $p$  is the fluid's pressure
- $\lambda$  is the fluid's volume viscosity
- $\mu$  is the fluid's dynamic viscosity

Assuming steady state and incompressibility and given proper boundary conditions the problem can be written formally as [11]:

$$\text{Strong Form} \left\{ \begin{array}{ll} \text{Given } \mathbf{b} : \Omega \rightarrow \mathbb{R}^3, \bar{\mathbf{u}} : \Gamma_u \rightarrow \mathbb{R}^3, \bar{\mathbf{t}} : \Gamma_\sigma \rightarrow \mathbb{R}^3, \text{ find } \mathbf{u}, p : \bar{\Omega} \rightarrow \mathbb{R} \text{ such that} \\ -\nabla p + \nu \nabla^2 \mathbf{u} - (\mathbf{u} \cdot \nabla) \mathbf{u} + \mathbf{b} = \mathbf{0} & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0 & \text{in } \Omega \\ \mathbf{u} = \bar{\mathbf{u}} & \text{on } \Gamma_u \\ -p\mathbf{n} + \nu(\hat{\mathbf{n}} \cdot \nabla) \mathbf{u} = \bar{\mathbf{t}} & \text{on } \Gamma_\sigma \end{array} \right. \quad (2.4)$$

where we have introduced the fluid's kinematic viscosity  $\nu$ .

## 2.2 Weak form

To obtain the weak form of the NS equations, the proper test and trial functions are introduced for velocity and pressure. The trial solution space  $\mathcal{S}$  that contains the approximating functions for the velocity is:

$$\mathcal{S} := \{ \mathbf{u} \in \mathcal{H}^1(\Omega) \mid \mathbf{u} = \bar{\mathbf{u}} \text{ on } \Gamma_u \} \quad (2.5)$$

Then the test functions for the velocity are introduced:

$$\mathcal{V} := \mathcal{H}_{\Gamma_u}^1(\Omega) = \{ \mathbf{v} \in \mathcal{H}^1(\Omega) \mid \mathbf{v} = 0 \text{ on } \Gamma_u \} \quad (2.6)$$

Finally, a suitable space for pressure test functions is defined:

$$\mathcal{Q} := \{ q \in \mathcal{L}^2(\Omega) \} \quad (2.7)$$

Taking the momentum equation from Eq.(2.4), multiplying by test functions  $\mathbf{v}$  and integrating over the domain,  $\Omega$ , we arrive at:

$$\int_{\Omega} -\nabla p \cdot \mathbf{v} \, d\Omega + \int_{\Omega} \nu \nabla^2 \mathbf{u} \cdot \mathbf{v} \, d\Omega + \int_{\Omega} \mathbf{b} \cdot \mathbf{v} \, d\Omega - \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} \, d\Omega = 0 \quad (2.8)$$

By using Gauss theorem, which states that:

$$\int_{\Omega} (\nabla \cdot \mathbf{F}) d\Omega = \int_{\Gamma} (\mathbf{F} \cdot \hat{\mathbf{n}}) d\Gamma$$

in the first and second terms of Eq.(2.8) and using integration by parts, we get:

$$\int_{\Omega} p \nabla \cdot \mathbf{v} d\Omega = - \int_{\Omega} \nabla p \cdot \mathbf{v} d\Omega + \int_{\Gamma} p \mathbf{v} \cdot \hat{\mathbf{n}} d\Gamma \quad (2.9)$$

$$\int_{\Omega} \nu \nabla \mathbf{u} \cdot \nabla \mathbf{v} d\Omega = - \int_{\Omega} \nu \nabla^2 \mathbf{u} \cdot \mathbf{v} d\Omega - \int_{\Gamma} \nu \frac{\partial \mathbf{u}}{\partial \hat{\mathbf{n}}} \cdot \mathbf{v} d\Gamma \quad (2.10)$$

Considering the above definitions of test functions, Eq.(2.6), that vanishes on the Dirichlet boundary, the integrals on the boundary of Eq.(2.9) and Eq.(2.10) can be rewritten as:

$$\int_{\Gamma} \left( \nu \frac{\partial \mathbf{u}}{\partial \hat{\mathbf{n}}} - p \hat{\mathbf{n}} \right) \cdot \mathbf{v} d\Gamma = \underbrace{\int_{\Gamma_u} \left( \nu \frac{\partial \mathbf{u}}{\partial \hat{\mathbf{n}}} - p \hat{\mathbf{n}} \right) \cdot \mathbf{v} d\Gamma_u}_{\mathbf{v}=0 \text{ on } \Gamma_u} + \underbrace{\int_{\Gamma_{\sigma}} \left( \nu \frac{\partial \mathbf{u}}{\partial \hat{\mathbf{n}}} - p \hat{\mathbf{n}} \right) \cdot \mathbf{v} d\Gamma_{\sigma}}_{=\bar{\mathbf{t}} \text{ on } \Gamma_{\sigma}} = \int_{\Gamma_{\sigma}} \bar{\mathbf{t}} \cdot \mathbf{v} d\Gamma_{\sigma} \quad (2.11)$$

As for the continuity equation, using Eq.(2.4), multiplying by the pressure test functions  $q$  and integrating over the domain, we get:

$$\int_{\Omega} q \nabla \cdot \mathbf{u} d\Omega = 0 \quad (2.12)$$

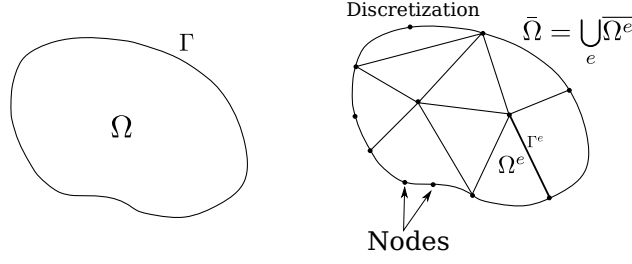
Substituting equations (2.9), (2.10) and (2.11) into Eq.(2.8) the weak form of the problem follows [12]:

$$\text{W.F.} \left\{ \begin{array}{l} \text{Given } \mathbf{b} : \Omega \rightarrow \mathbb{R}^3, \bar{\mathbf{t}} : \Gamma_{\sigma} \rightarrow \mathbb{R}^3, \text{ find } \mathbf{u} \in \mathcal{S}, p \in \mathcal{Q} \text{ such that} \\ \int_{\Omega} \nu \nabla \mathbf{u} \cdot \nabla \mathbf{v} d\Omega + \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} d\Omega - \int_{\Omega} p \nabla \cdot \mathbf{v} d\Omega = \int_{\Omega} \mathbf{b} \cdot \mathbf{v} d\Omega + \int_{\Gamma_{\sigma}} \bar{\mathbf{t}} \cdot \mathbf{v} d\Gamma_{\sigma} \quad \forall \mathbf{v} \in \mathcal{V} \\ \int_{\Omega} q \nabla \cdot \mathbf{u} d\Omega = 0 \quad \forall q \in \mathcal{Q} \end{array} \right. \quad (2.13)$$

## 2.3 Finite element formulation

Now that the weak form is obtained, the next step to obtain the finite element formulation is to discretize the domain,  $\bar{\Omega}$  which is divided into element domains  $\bar{\Omega}^e$  for  $e = 1, 2, \dots, n_{el}$

$$\bar{\Omega} = \bar{\Omega}^1 \cup \bar{\Omega}^2 \dots \cup \bar{\Omega}^{n_{el}} = \bigcup_{e=1}^{n_{el}} \bar{\Omega}^e \quad (2.14)$$

Figure 2.1: Discretization of the domain into  $n_{el}$  elements with corresponding nodes

Then, we define  $\mathbf{r}$  as the set of velocity nodes pertaining to the Dirichlet boundary  $\Gamma_{\mathbf{u}}$  and  $\mathbf{l}$  as the set of remaining velocities so that:

$$\mathbf{r} \cup \mathbf{l} = \{1, 2, \dots, n_{nod}\} \quad (2.15)$$

where  $n_{nod}$  is the total number of velocity nodes.

The vector of prescribed velocity  $\bar{\mathbf{u}}$  contains the prescribed velocity in each direction at each Dirichlet point.

Finally, the velocity shape functions are defined as  $N_A : \bar{\Omega} \rightarrow \mathbb{R}$  ( $A = 1, 2, \dots, n_{nod}$ ).

$$N_A(\mathbf{x}_B) = \delta_{AB} \quad (2.16)$$

where  $\delta_{AB}$  is the Kronecker delta, so they have a value of 1 when  $A=B$  and 0 in the other nodes. In the rest of the domain the interpolation function can be chosen. Typically it is assumed linear or quadratic, as higher orders decrease performance and similar results can be obtained through mesh refinement.

Now, the finite dimensional space of velocity test functions  $\mathbf{v}^h \in \mathbf{V}$  can be defined as:

$$\mathbf{v}^h = \sum_{A=1}^{n_{nod}} N_A \mathbf{c}_A, \text{ with } \mathbf{c}_A = \mathbf{0} \text{ if } A \in \mathbf{r} \quad (2.17)$$

In matrix notation:

$$\mathbf{v}^h := \begin{bmatrix} v_1^h \\ v_2^h \end{bmatrix}, \quad \mathbf{N}_A := N_A \mathbf{I} = \begin{bmatrix} N_A & 0 \\ 0 & N_A \end{bmatrix}, \quad \mathbf{c}_A := \begin{bmatrix} c_{1A} \\ c_{2A} \end{bmatrix} \quad (2.18)$$

Similarly, finite dimensional space of velocity trial functions  $\mathbf{s}^h \in \mathbf{S}$  is the space of all functions of the form:

$$\mathbf{u}^h = \sum_{A=1}^{n_{nod}} N_A \mathbf{d}_A, \text{ with } \mathbf{d}_A = \bar{\mathbf{u}}(\mathbf{x}_A) \text{ if } A \in \mathbf{r} \quad (2.19)$$

Notice how it is enforced that  $\mathbf{d}_A = \bar{\mathbf{u}}(\mathbf{x}_A)$  to meet the Dirichlet condition, and  $\mathbf{c}_A = \mathbf{0}$  if  $A \in \mathbf{r}$  to ensure the test functions vanish on the Dirichlet conditions.

For the pressure, an almost identical definition is used, the shape functions are defined as:  $\hat{N}_A : \bar{\Omega} \rightarrow \mathbb{R}$  ( $A =$

$1, 2, \dots, \hat{n}_{nod})$

$$\hat{N}_A(\hat{\mathbf{x}}_B) = \delta_{AB} \quad (2.20)$$

As done for the velocity, the finite dimensional space of trial functions for the pressure  $\mathcal{Q}^h \subset \mathcal{Q}$  is the space of all functions of the form:

$$p^h = \sum_{A=1}^{\hat{n}_{nod}} \hat{N}_A p_A \quad (2.21)$$

The same can be done for the test functions:

$$q^h = \sum_{A=1}^{\hat{n}_{nod}} \hat{N}_A q_A \quad (2.22)$$

Hence, equation Eq.(2.13) can be rewritten in terms of these new finite dimensional test and trial functions [13].

$$\text{W.F.} \left\{ \begin{array}{l} \text{Given } \mathbf{b} : \Omega \rightarrow \mathbb{R}^3, \bar{\mathbf{t}} : \Gamma_\sigma \rightarrow \mathbb{R}^3, \text{ find } \mathbf{u}^h \in \mathcal{S}^h, p^h \in \mathcal{Q}^h \text{ such that} \\ \int_{\Omega} \nu \nabla \mathbf{u}^h \cdot \nabla \mathbf{v}^h d\Omega + \int_{\Omega} (\mathbf{u}^h \cdot \nabla) \mathbf{u}^h \cdot \mathbf{v}^h d\Omega - \int_{\Omega} p^h \nabla \cdot \mathbf{v}^h d\Omega = \\ = \int_{\Omega} \mathbf{b} \cdot \mathbf{v}^h d\Omega + \int_{\Gamma_\sigma} \bar{\mathbf{t}} \cdot \mathbf{v}^h d\Gamma_\sigma \quad \forall \mathbf{v}^h \in \mathcal{V}^h \\ \int_{\Omega} q^h \nabla \cdot \mathbf{u}^h d\Omega = 0 \quad \forall q^h \in \mathcal{Q}^h \end{array} \right. \quad (2.23)$$

## 2.4 Matrix problem

Now, we need to derive a matrix formulation for the current problem. To arrive at the matrix problem, equations Eq.(2.17), Eq.(2.19) and Eq.(2.22) are substituted into equation Eq.(2.23), but firstly, the following linear forms will be introduced for the sake of clarity:

$$a(w, v) := \int_{\Omega} \nabla w \cdot \nu \nabla v d\Omega \quad (2.24)$$

$$\bar{a}(v, q) := - \int_{\Omega} q \nabla \cdot v d\Omega \quad (2.25)$$

$$c(\chi; w, v) := \int_{\Omega} w \cdot (\chi \cdot \nabla) v d\Omega \quad (2.26)$$

$$\langle v, b \rangle := \int_{\Omega} v^T b d\Omega \quad (2.27)$$

$$\langle v, b \rangle_{\Gamma_\sigma} := \int_{\Gamma_\sigma} v b d\Gamma \quad (2.28)$$

With these definitions at hand, the substitution can be made. For the momentum equation:

$$\begin{aligned} a \left( \sum_{l=1}^{n_{nod}} \mathbf{N}_l \mathbf{c}_l, \sum_{J=1}^{n_{nod}} \mathbf{N}_J \mathbf{d}_J \right) + c \left( \sum_{J=1}^{n_{nod}} \mathbf{N}_J \mathbf{d}_J; \sum_{l=1}^{n_{nod}} \mathbf{N}_l \mathbf{c}_l, \sum_{J=1}^{n_{nod}} \mathbf{N}_J \mathbf{d}_J \right) + \bar{a} \left( \sum_{l=1}^{n_{nod}} \mathbf{N}_l \mathbf{c}_l, \sum_{K=1}^{\hat{n}_{nod}} \hat{N}_K \mathbf{p}_K \right) \\ = \left\langle \sum_{l=1}^{n_{nod}} \mathbf{N}_l \mathbf{c}_l, \mathbf{b} \right\rangle + \left\langle \sum_{l=1}^{n_{nod}} \mathbf{N}_l \mathbf{c}_l, \bar{\mathbf{t}} \right\rangle_{\Gamma_\sigma} \end{aligned} \quad (2.29)$$

As for the continuity equation:

$$\bar{a} \left( \sum_{K=1}^{\hat{n}_{nod}} \hat{N}_K \mathbf{q}_K, \sum_{l=1}^{n_{nod}} \mathbf{N}_l \mathbf{d}_l \right) = 0 \quad (2.30)$$

Simplifying, the momentum equation becomes:

$$\sum_{l=1}^{n_{nod}} \mathbf{c}_l \left( \sum_{J=1}^{n_{nod}} \overbrace{a(\mathbf{N}_l, \mathbf{N}_J)}^{= \mathbf{K}_{lJ}} \mathbf{d}_J + \sum_{J=1}^{n_{nod}} \overbrace{c(\mathbf{N}_J \mathbf{d}_J; \mathbf{N}_l, \mathbf{N}_J)}^{= \mathbf{C}_{lJ}} \mathbf{d}_J + \sum_{K=1}^{\hat{n}_{nod}} \overbrace{\bar{a}(\mathbf{N}_l, \hat{N}_K) \mathbf{p}_K}^{= \mathbf{G}_{lK}} - \overbrace{(\langle \mathbf{N}_l, \mathbf{b} \rangle + \langle \mathbf{N}_l, \bar{\mathbf{t}} \rangle_{\Gamma_\sigma})}^{= \mathbf{F}_l} \right) = 0 \quad (2.31)$$

As for the pressure equation:

$$\sum_{K=1}^{\hat{n}_{nod}} \mathbf{q}_K \left( \sum_{l=1}^{n_{nod}} \overbrace{\bar{a}(\mathbf{N}_l, \hat{N}_K) \mathbf{d}_l}^{= [\mathbf{G}^T]_{KI}} \right) = 0 \quad (2.32)$$

Next, we define the matrices appearing in the preceding equation:

**Global viscosity matrix** : The block matrix  $\mathbf{K}_{lJ} \in \mathbb{R}^{n_{sd} \times n_{sd}}$  of the **global viscosity matrix**  $\mathbf{K} \in \mathbb{R}^{n_{sd} n_{nod} \times n_{sd} n_{nod}}$  corresponding to velocity nodes  $l, J$  is defined by

$$\mathbf{K}_{lJ} := a(\mathbf{N}_l, \mathbf{N}_J) \quad (2.33)$$

**Global external force vector** (body force + boundary tractions). The block vector  $\mathbf{F}_l \in \mathbb{R}^{n_{sd}}$  of the global external force vector  $\mathbf{F} \in \mathbb{R}^{n_{sd} n_{nod} \times 1}$  corresponding to velocity node  $l$  is defined by

$$\mathbf{F}_l := \langle \mathbf{N}_l, \mathbf{b} \rangle + \langle \mathbf{N}_l, \bar{\mathbf{t}} \rangle_{\Gamma_\sigma} \quad (2.34)$$

**Global gradient operator** : The block matrix  $\mathbf{G}_{lK} \in \mathbb{R}^{n_{sd} \times 1}$  of the **global gradient operator**  $\mathbf{G} \in \mathbb{R}^{n_{sd} n_{nod} \times \hat{n}_{nod}}$  corresponding to velocity node  $l$  and pressure node  $K$  is defined by

$$\mathbf{G}_{lK} := \bar{a}(\mathbf{N}_l, \hat{N}_K) \quad (2.35)$$

**Global convection matrix**: The block matrix  $\mathbf{C}_{lJ} \in \mathbb{R}^{n_{sd} \times n_{sd}}$  of the **global convection matrix**

$\mathbf{C} \in \mathbb{R}^{n_{sd}n_{nod} \times n_{sd}n_{nod}}$  corresponding to velocity nodes  $l, J$  is defined by

$$\mathbf{C}_{lJ} := c(\mathbf{N}_J \mathbf{d}_J; \mathbf{N}_l, \mathbf{N}_J) \quad (2.36)$$

which depends on the unknowns  $\mathbf{d}_J$ . This will make it so that a nonlinear equation will need to be solved. From now onward, the assumption that the problem is 2D will be made. The extension to 3D can be found in [11] or [13].

Then, defining

- Global vectors of velocities ( $\mathbf{d} \in \mathbb{R}^{n_{sd}n_{nod}}$ ) and variations ( $\mathbf{c} \in \mathbb{R}^{n_{sd}n_{nod}}$ )

$$\mathbf{d} := \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \vdots \\ \mathbf{d}_{n_{nod}} \end{bmatrix}, \quad \mathbf{c} := \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{n_{nod}} \end{bmatrix} \quad (2.37)$$

- Set of constrained degrees of freedom *global* degrees of freedom along which velocity is *known*)

$$\mathbf{r} := \begin{bmatrix} 2\mathbf{r}^1 - 1 \\ 2\mathbf{r}^2 \end{bmatrix} \quad (2.38)$$

Likewise, we define the set of unconstrained degrees of freedom for velocity (*global* degrees of freedom along which velocity is *unknown*) as

$$\mathbf{l} := \begin{bmatrix} 2\mathbf{l}^1 - 1 \\ 2\mathbf{l}^2 \end{bmatrix} \quad (2.39)$$

Note that  $\mathbf{l} \cup \mathbf{r} = \{1, 2, \dots, n_{sd}n_{nod}\}$  and these definitions extend those of 2.15 for multiple degrees of freedom, i. e. when the field is a vector field and has  $n_{sd}$  solutions for each node. So for a node  $A$ , the corresponding DOF are:

$$DOF_A = \begin{bmatrix} 2A - 1 \\ 2A \end{bmatrix} \quad (2.40)$$

- Global vector of prescribed velocities (constructed by arranging in a single vector the vectors defined in Eq.(2.41))

$$\bar{\mathbf{u}} := \begin{bmatrix} \bar{u}^1 \\ \bar{u}^2 \end{bmatrix} \quad (2.41)$$

- Global vector of nodal pressures ( $\mathbf{p} \in \mathbb{R}^{\hat{n}_{nod}}$ ) and variations ( $\mathbf{q} \in \mathbb{R}^{\hat{n}_{nod}}$ )

$$\mathbf{p} := \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{\hat{n}_{nod}} \end{bmatrix}, \quad \mathbf{q} := \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{\hat{n}_{nod}} \end{bmatrix} \quad (2.42)$$

With these definitions, equation Eq.(2.31) becomes:

$$\mathbf{c}^T((\mathbf{K} + \mathbf{C}(\mathbf{d}))\mathbf{d} + \mathbf{G}\mathbf{p} - \mathbf{F}) = \mathbf{0}, \quad \forall \mathbf{c}_1 \quad (2.43)$$

where  $\mathbf{c}_r = \mathbf{0}$  and  $\mathbf{d}_r = \bar{\mathbf{u}}$ .

Making the decomposition into block matrices, the preceding equation can be expressed:

$$\begin{aligned} & \begin{bmatrix} \mathbf{c}_1^T & \mathbf{c}_r^T \end{bmatrix} \left( \begin{bmatrix} \mathbf{K}_{ll} + \mathbf{C}_{ll} & \mathbf{K}_{lr} + \mathbf{C}_{lr} \\ \mathbf{K}_{rl} + \mathbf{C}_{rl} & \mathbf{K}_{rr} + \mathbf{C}_{rr} \end{bmatrix} \begin{bmatrix} \mathbf{d}_l \\ \mathbf{d}_r \end{bmatrix} + \begin{bmatrix} \mathbf{G}_l \\ \mathbf{G}_r \end{bmatrix} \mathbf{p} - \begin{bmatrix} \mathbf{F}_l \\ \mathbf{F}_r \end{bmatrix} \right) = \mathbf{0} \quad \forall \mathbf{c}_1 \\ & \Rightarrow \mathbf{c}_1^T ((\mathbf{K}_{ll} + \mathbf{C}_{ll})\mathbf{d}_l + (\mathbf{K}_{lr} + \mathbf{C}_{lr}) \overbrace{\mathbf{d}_r}^{\bar{\mathbf{u}}} + \mathbf{G}_l \mathbf{p} - \mathbf{F}_l) + \\ & \quad \underbrace{\mathbf{0}}_{\mathbf{c}_r^T} ((\mathbf{K}_{rl} + \mathbf{C}_{rl})\mathbf{d}_l + (\mathbf{K}_{rr} + \mathbf{C}_{rr})\mathbf{d}_r + \mathbf{G}_r \mathbf{p} - \mathbf{F}_r) = 0 \\ & \Rightarrow \mathbf{c}_1^T ((\mathbf{K}_{ll} + \mathbf{C}_{ll})\mathbf{d}_l + (\mathbf{K}_{lr} + \mathbf{C}_{lr})\bar{\mathbf{u}} + \mathbf{G}_l \mathbf{p} - \mathbf{F}_l) = 0. \end{aligned} \quad (2.44)$$

For the incompressibility equation:

$$\mathbf{q}^T (\mathbf{G}_l^T \mathbf{d}_l + \mathbf{G}_r^T \mathbf{d}_r) = 0 \quad \forall \mathbf{q} \quad (2.45)$$

Hence, these equations are fulfilled for all  $\mathbf{c}_1^T$  and  $\mathbf{q}$  if and only if the terms multiplying  $\mathbf{c}_1^T$  and  $\mathbf{q}$  vanish. So the equations in compact form can be rewritten as:

$$\begin{bmatrix} \mathbf{K}_{ll} + \mathbf{C}_{ll}(\mathbf{d}_l) & \mathbf{G}_l \\ \mathbf{G}_l^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{d}_l \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_l - (\mathbf{K}_{lr} + \mathbf{C}_{lr}(\mathbf{d}_l))\bar{\mathbf{u}} \\ -\mathbf{G}_r^T \bar{\mathbf{u}} \end{bmatrix} \quad (2.46)$$



Note that the coefficient matrix  $\mathbf{C}$  depends on the unknowns  $\mathbf{d}_1$ . This makes it a nonlinear system of equations that can be solved via Newton-Raphson method or via Picard iteration as described in section 2.6.

To obtain Stokes flow, the hypothesis that the convective term is much smaller than the viscous term is made  $(\mathbf{u} \cdot \nabla)\mathbf{u} \ll \nu \nabla^2 \mathbf{u}$ . Thus, Stokes flow arises when viscosity is the only effect on the flow. Then it is trivial to see that the system of equations 2.46 becomes:

$$\begin{bmatrix} \mathbf{K}_{11} & \mathbf{G}_1 \\ \mathbf{G}_1^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_1 - \mathbf{K}_{1r}\bar{\mathbf{u}} \\ -\mathbf{G}_r^T \bar{\mathbf{u}} \end{bmatrix} \quad (2.47)$$

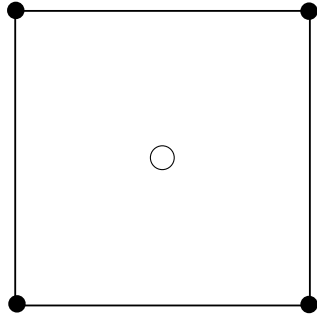
which is a linear system.

Notice that pressure acts as a Lagrangian multiplier of the incompressibility constraint, this means that the solution will exhibit a saddle point [11]. Practically this means that the pressure on some node must be fixed to some reference value to establish a base for the pressure on the other nodes.

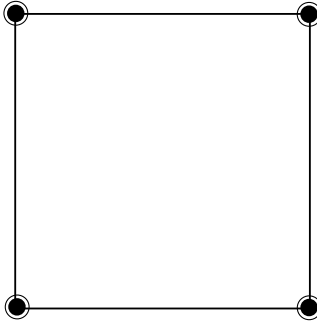
### 2.4.1 Solvability conditions

In Eq.(2.46) and Eq.(2.47) the null submatrix in the diagonal poses an interesting question: Under which conditions is the system solvable? The condition that must be satisfied is that the kernel of  $\mathbf{G}_1$  is zero, so then  $\mathbf{d}_1$  and  $\mathbf{p}$  are uniquely defined. To satisfy that  $\ker(\mathbf{G}_1) = 0$ , the LBB condition must be satisfied (Ladyzhenskaya (1969), Babuška (1970/71) and Brezzi (1974)). This condition is also called the inf sup condition for its formal mathematical description. This enforces a different discretization for the velocity and pressure, which makes the meshing process rather tedious [11].

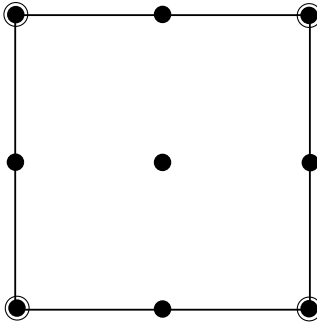
As seen on figure 2.2, there are multiple elements that satisfy the LBB condition. In this report, the Q2Q1 element has been chosen as it has quadratic convergence for the velocity and is easy to implement.



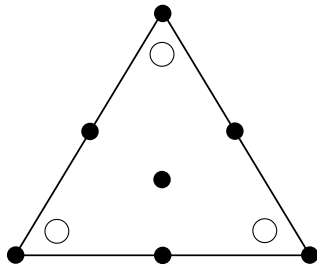
Q1P0 element:  
 Continuous bilinear velocity,  
 Discontinuous constant pressure,  
 Does not satisfy LBB condition.



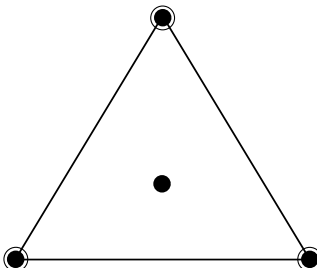
Q1Q1 element:  
 Continuous bilinear velocity,  
 Continuous bilinear pressure,  
 Does not satisfy LBB condition.



Q2Q1 element:  
 (Taylor-Hood element)  
 Continuous biquadratic velocity,  
 Continuous bilinear pressure,  
 Satisfies LBB condition.  
 Quadratic convergence.



Crouzeix-Raviart element:  
 Velocity: continuous quadratic  
 & Cubic bubble function,  
 Pressure: discontinuous linear,  
 Satisfies LBB condition  
 Quadratic convergence.



Mini element:  
 Velocity: continuous linear  
 & Cubic bubble function,  
 Pressure: continuous linear,  
 Satisfies LBB condition  
 Linear convergence.

● Velocity node

○ Pressure node

Figure 2.2: Types of mixed elements and whether they satisfy the LBB condition [11]

## 2.5 Element point of view

It has been seen that matrices  $\mathbf{K}$ ,  $\mathbf{C}$  and  $\mathbf{G}$  and the vector  $\mathbf{F}$  need to be constructed to be able to solve Eq.(2.46) and obtain the global velocity and pressure. In this section the global matrices will be obtained through the elemental matrices.

The velocity within an element is:

$$u^h|_{\Omega^e} = u^e = \sum_{a=1}^{n^e} N_a^e d_a^e = \begin{bmatrix} N_1^e & N_2^e & \cdots & N_{n^e}^e \end{bmatrix} \begin{bmatrix} d_1^e \\ d_2^e \\ \vdots \\ d_{n^e}^e \end{bmatrix} \quad (2.48)$$

$$\Rightarrow u^e = \mathbf{N}^e \mathbf{d}^e$$

where  $n^e$  is the number of nodes of the  $e$ -th element, and  $\mathbf{N}^e \in \mathbb{R}^{1 \times n^e}$  and  $\mathbf{d}^e \in \mathbb{R}^{n^e}$  are defined by

$$\mathbf{N}^e := \begin{bmatrix} N_1^e & N_2^e & \cdots & N_{n^e}^e \end{bmatrix} \quad (2.49)$$

and

$$\mathbf{d}^e := \begin{bmatrix} d_1^e \\ d_2^e \\ \vdots \\ d_{n^e}^e \end{bmatrix} \quad (2.50)$$

respectively. Similarly, for the gradient of velocities:

$$\nabla \mathbf{u}^e = \mathbf{B}^e \mathbf{d}^e \quad (2.51)$$

With this definitions at hand, the following element matrices can be known:

Element viscosity matrix:

$$\mathbf{K}^e := \int_{\Omega^e} \mathbf{B}^{eT} \nu \mathbf{B}^e d\Omega \quad (2.52)$$

Element gradient operator:

$$\mathbf{G}^e := - \int_{\Omega^e} \hat{\mathbf{N}}^{eT} [1 \ 0 \ 0 \ 1] \mathbf{B}^e d\Omega \quad (2.53)$$

Notice the appearance of the vector  $[1 \ 0 \ 0 \ 1]$ . This is due to the gradient operator and the difference in nodes per element between pressure and velocity interpolation.

Element convection matrix:

$$\mathbf{C}^e := \int_{\Omega^e} \mathbf{N}^{eT} \mathbf{u}^e \mathbf{B}^e d\Omega \quad (2.54)$$

The elemental body force:

$$\begin{aligned} \mathbf{F}_b^e &= \int_{\Omega^e} \mathbf{N}^{eT} \mathbf{b} d\Omega = \int_{\Omega^e} \mathbf{N}^{eT} (\mathbf{N}^e \mathbf{b}^e) d\Omega \\ \Rightarrow \mathbf{F}_b^e &= \int_{\Omega^e} \mathbf{N}^{eT} (\mathbf{N}^e \mathbf{b}^e) d\Omega \end{aligned} \quad (2.55)$$

where

$$\mathbf{b}^e := \begin{bmatrix} \mathbf{b}(\mathbf{x}_1^e) \\ \mathbf{b}(\mathbf{x}_2^e) \\ \vdots \\ \mathbf{b}(\mathbf{x}_{n^e}^e) \end{bmatrix} \quad (2.56)$$

is the vector containing the values of function  $\mathbf{b}$  at each node of element  $e$ . Finally the traction forces:

$$\mathbf{F}_{dis}^e = \int_{\Gamma_\sigma^e} \bar{\mathbf{N}}^{eT} \bar{\mathbf{t}} d\Gamma_\sigma = \int_{\Gamma_\sigma^e} \bar{\mathbf{N}}^{eT} (\bar{\mathbf{N}}^e \bar{\mathbf{t}}^e) d\Gamma_\sigma \quad (2.57)$$

And

$$\mathbf{F}^e = \mathbf{F}_b^e + \mathbf{F}_{dis}^e \quad (2.58)$$

These vectors and matrices are assembled by their corresponding assembly operator  $\mathbf{A}_{e=1}^{n_{el}}$  that assigns each local degree of freedom to the corresponding global DOF. For more details see section 3.1

### 2.5.1 Elemental matrices calculations

Now, the last step to compute Eq.(2.46) is to calculate each elemental matrix, equations (2.52), (2.53) and (2.54) and the elemental vector (2.55). First, some definitions are needed in order to compute them.

**Physical domain** :  $\mathbf{x} \in \bar{\Omega}^e$

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.59)$$

$\mathbf{x}$  is termed the vector of global or physical coordinates.

**Parent** or element domain:  $\boldsymbol{\xi} \in \bar{\Omega}_\xi$

$$\boldsymbol{\xi} = \begin{bmatrix} \xi \\ \eta \end{bmatrix} \quad (2.60)$$

$\boldsymbol{\xi}$  is termed the vector of local or element coordinates.

Isoparametric elements: The mapping from the parent domain  $\bar{\Omega}_\xi$  to the physical domain  $\bar{\Omega}^e$ :

$$\mathbf{x} : \bar{\Omega}_\xi \rightarrow \bar{\Omega}^e \quad (2.61)$$

is constructed using the same shape functions employed in the interpolation of  $\mathbf{u}^h$ , that is:

$$x = \tilde{\mathbf{N}}^e(\boldsymbol{\xi}) \mathbf{x}^e = \overbrace{\begin{bmatrix} N_1^e & N_2^e & \cdots & N_{n^e}^e \end{bmatrix}}^{:= \tilde{\mathbf{N}}^e} \begin{bmatrix} x_1^e \\ x_2^e \\ \vdots \\ x_{n^e}^e \end{bmatrix} \quad (2.62)$$

$$y = \tilde{\mathbf{N}}^e(\boldsymbol{\xi}) \mathbf{y}^e = \begin{bmatrix} N_1^e & N_2^e & \cdots & N_{n^e}^e \end{bmatrix} \begin{bmatrix} y_1^e \\ y_2^e \\ \vdots \\ y_{n^e}^e \end{bmatrix} \quad (2.63)$$

$$\begin{aligned} \begin{bmatrix} x & y \end{bmatrix} &= \mathbf{x}^T = \tilde{\mathbf{N}}^e \overbrace{\begin{bmatrix} x_1^e & y_1^e \\ x_2^e & y_2^e \\ \vdots & \vdots \\ x_{n^e}^e & y_{n^e}^e \end{bmatrix}}^{= \mathbf{X}^{eT}} \\ &= \tilde{\mathbf{N}}^e \mathbf{X}^{eT} \\ \Rightarrow x(\boldsymbol{\xi}) &= \mathbf{X}^e \tilde{\mathbf{N}}^{eT}(\boldsymbol{\xi}) \end{aligned} \quad (2.64)$$

where

$$\mathbf{X}^e := \begin{bmatrix} x_1^e & x_2^e & \cdots & x_{n^e}^e \\ y_1^e & y_2^e & \cdots & y_{n^e}^e \end{bmatrix} \quad (2.65)$$

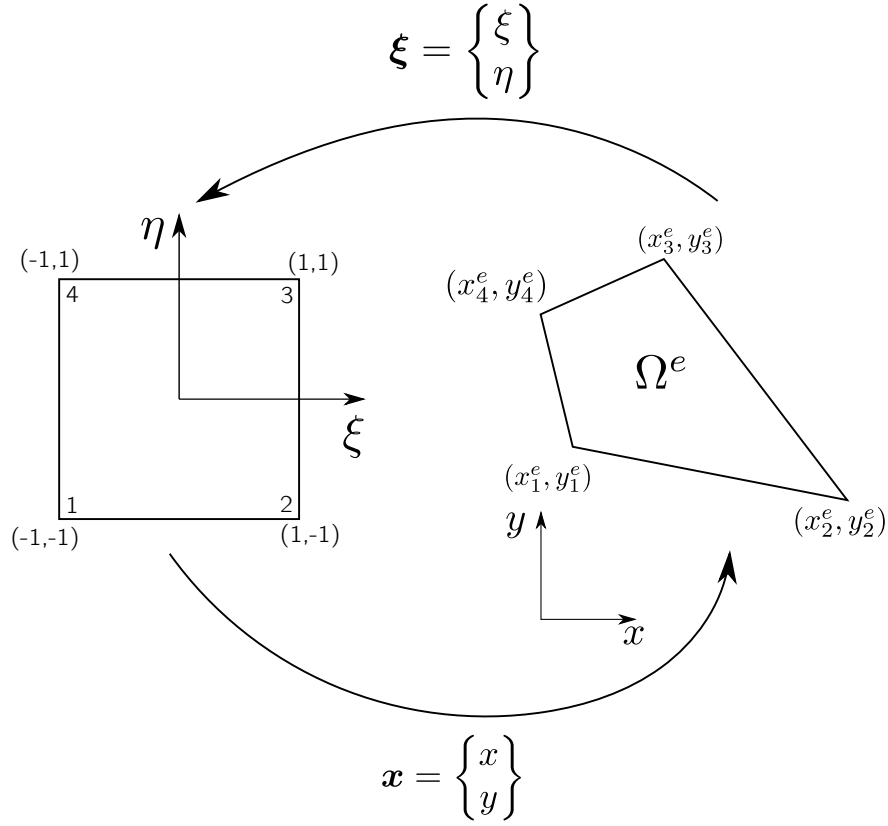


Figure 2.3: Transformation from the parent domain to the coordinate domain.

There are 2 matrices for shape functions, as  $\tilde{\mathbf{N}}^e$  is the matrix of shape functions for scalar based functions, and  $\mathbf{N}^e$  is the matrix of shape functions for vector based functions. They are defined respectively as:

$$\tilde{\mathbf{N}}^e := \begin{bmatrix} N_1^e & N_2^e & \cdots & N_{n^e}^e \end{bmatrix} \quad (2.66)$$

$$\mathbf{N}^e := \begin{bmatrix} \begin{bmatrix} N_1^e & 0 \\ 0 & N_1^e \end{bmatrix} & \begin{bmatrix} N_2^e & 0 \\ 0 & N_2^e \end{bmatrix} & \cdots & \begin{bmatrix} N_{n^e}^e & 0 \\ 0 & N_{n^e}^e \end{bmatrix} \end{bmatrix} \quad (2.67)$$

To go from one domain to another, the Jacobian matrix to change the basis is introduced:

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \overbrace{\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}}^{:= \mathbf{J}^e} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \Rightarrow d\mathbf{x} = \mathbf{J}^e d\boldsymbol{\xi} \quad (2.68)$$

The inverse of  $\mathbf{J}^e$  is the Jacobian matrix of the inverse mapping  $\boldsymbol{\xi} : \bar{\Omega}^e \rightarrow \bar{\Omega}_\xi$

$$\begin{bmatrix} d\xi \\ d\eta \end{bmatrix} = \overbrace{\begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix}}^{:= \mathbf{J}^{e-1}} \begin{bmatrix} dx \\ dy \end{bmatrix} \Rightarrow d\boldsymbol{\xi} = \mathbf{J}^{e-1} d\mathbf{x} \quad (2.69)$$

The determinant of the Jacobian matrix is called the Jacobian:

$$|\mathbf{J}^e| := \det(\mathbf{J}^e) \quad (2.70)$$

It can be shown that:

$$\mathbf{J}^e = \mathbf{X}^e \tilde{\mathbf{B}}_\xi^{eT} \quad (2.71)$$

where:

$$\tilde{\mathbf{B}}_\xi^e := \tilde{\nabla}_\xi \tilde{\mathbf{N}}^e = \begin{bmatrix} \frac{\partial \tilde{\mathbf{N}}^e}{\partial \xi} \\ \frac{\partial \tilde{\mathbf{N}}^e}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1^e}{\partial \xi} & \frac{\partial N_2^e}{\partial \xi} & \dots & \frac{\partial N_{n^e}^e}{\partial \xi} \\ \frac{\partial N_1^e}{\partial \eta} & \frac{\partial N_2^e}{\partial \eta} & \dots & \frac{\partial N_{n^e}^e}{\partial \eta} \end{bmatrix} \quad (2.72)$$

With the above definitions it's easy to prove that

$$\tilde{\mathbf{B}}^e = \mathbf{J}^{e-T} \tilde{\mathbf{B}}_\xi^e \quad (2.73)$$

Similar to  $\tilde{\mathbf{N}}^e$  and  $\mathbf{N}^e$ , we define the matrix of symmetric gradient  $\mathbf{B}^e$  as:

$$\mathbf{B}^e := \begin{bmatrix} \frac{\partial N_1^e}{\partial x} & 0 & \frac{\partial N_2^e}{\partial x} & 0 & \dots & \frac{\partial N_{n^e}^e}{\partial x} & 0 \\ 0 & \frac{\partial N_1^e}{\partial x} & 0 & \frac{\partial N_2^e}{\partial x} & 0 & \dots & \frac{\partial N_{n^e}^e}{\partial x} \\ \frac{\partial N_1^e}{\partial y} & 0 & \frac{\partial N_2^e}{\partial y} & 0 & \dots & \frac{\partial N_{n^e}^e}{\partial y} & 0 \\ 0 & \frac{\partial N_1^e}{\partial y} & 0 & \frac{\partial N_2^e}{\partial y} & 0 & \dots & \frac{\partial N_{n^e}^e}{\partial y} \end{bmatrix} \quad (2.74)$$

Moreover, the mapping from  $\tilde{\mathbf{B}}^e$  to  $\mathbf{B}^e$  is defined as:

$$\mathbf{B}^e(\boldsymbol{\xi}_g) = \tilde{\mathbf{Q}}(\tilde{\mathbf{B}}^e(\boldsymbol{\xi}_g)) \quad (2.75)$$

Then to calculate, Eq.(2.52) and similar integrals, Gauss quadrature will be used, which is a method to approximate integrals by evaluating the integrand at certain points (Gauss points). Afterwards, a weighted sum of these values is computed, multiplying each by an appropriate weight (Gauss weight), which gives the

approximation to the integral [14, p. 58-65].

$$I = \int_{\Omega^e} f \, d\Omega = \int_{\Omega_\xi} |\mathbf{J}^e|(\xi) f(\xi) \, d\Omega_\xi \approx \sum_{g=1}^{n_m} w_g |\mathbf{J}^e|(\xi_g) f(\xi_g) \quad (2.76)$$

Finally,  $\mathbf{K}^e$  is computed as:

$$\begin{aligned} \mathbf{K}^e &= \int_{\Omega^e} \mathbf{B}^{eT} \nu \mathbf{B}^e \, d\Omega = \int_{\Omega_\xi} |\mathbf{J}^e| \mathbf{B}^{eT} \nu \mathbf{B}^e \, d\Omega_\xi \\ \Rightarrow \mathbf{K}^e &= \sum_{g=1}^{n_m} w_g \left( |\mathbf{J}^e| \mathbf{B}^{eT} \nu \mathbf{B}^e \right)_{\xi=\xi_g} \end{aligned} \quad (2.77)$$

where  $|\mathbf{J}^e|(\xi_g)$  and  $\mathbf{B}^e(\xi_g)$  are given by

$$\mathbf{B}^e(\xi_g) = \tilde{\mathbf{Q}}(\tilde{\mathbf{B}}^e(\xi_g)) \quad (2.78)$$

$$\tilde{\mathbf{B}}^e(\xi_g) = \mathbf{J}^{e-T}(\xi_g) \tilde{\mathbf{B}}_\xi^e(\xi_g) \quad (2.79)$$

$$\mathbf{J}^e(\xi_g) = \mathbf{X}^e \tilde{\mathbf{B}}_\xi^{eT}(\xi_g) \quad (2.80)$$

To calculate  $\mathbf{G}^e$ ,  $\mathbf{C}^e$  and  $\mathbf{F}^e$  an identical procedure is followed, the only difference is that to compute  $\hat{\mathbf{N}}^e$  the pressure shape functions and nodes will be used.

## 2.6 Picard Iteration

To solve the nonlinear system of equations in (2.46) Picard iteration will be used. This algorithm to solve nonlinear systems is based on Banach's fixed point theorem.

**Definition.** Let  $(X, d)$  be a complete metric space. Then a map  $T : X \rightarrow X$  is called a contraction mapping on  $X$  if there exists  $q \in [0, 1)$  such that for all  $x, y \in X$

$$d(T(x), T(y)) \leq qd(x, y)$$

**Banach's fixed point theorem.** Let  $(X, d)$  be a non-empty complete metric space with a contraction mapping  $T : X \rightarrow X$ . Then  $T$  admits a unique fixed-point  $x^*$  in  $X$  (i.e.  $T(x^*) = x^*$ ). Furthermore,  $x^*$  can be found as follows: start with an arbitrary element  $x_0 \in X$  and define a sequence  $(x_n)_{n \in \mathbb{N}}$  by  $x_n = T(x_{n-1})$  for  $n \geq 1$ . Then  $\lim_{n \rightarrow \infty} x_n = x^*$ . [15]

The idea behind Picard's algorithm is that  $T$  is a contraction mapping induced by Eq.(2.46), and the point  $x^*$  is the desired solution of the nonlinear system. With this in mind, the algorithm adopts the following form:



---

**Algorithm 1:** Picard iteration for system Eq.(2.46)

---

```

1 Function  $[d_l, p] \leftarrow \text{Picard}(\mathbf{K}_{ll}, \mathbf{K}_{lr}, \mathbf{G}_l, \mathbf{G}_r, \mathbf{F}_l, \bar{\mathbf{u}}, \delta, \mathcal{N}, \mathcal{B}, \mathcal{W}, \psi)$ :
   Data:  $\mathbf{K}_{ll} \in \mathbb{R}^{n_{DOFl} \times n_{DOFl}}$ ,  $\mathbf{G}_l \in \mathbb{R}^{n_{DOFl} \times \hat{n}_{DOFl}}$ ,  $\mathbf{K}_{lr} \in \mathbb{R}^{n_{DOFl} \times n_{DOFr}}$ ,
            $\mathbf{G}_r \in \mathbb{R}^{n_{DOFr} \times \hat{n}_{DOFl}}$ ,  $\bar{\mathbf{u}} \in \mathbb{R}^{n_{DOFr} \times 1}$ , convergence tolerance  $\delta$ , Stacked matrices  $\mathcal{N}, \mathcal{B}$  and
           global Gauss and Jacobians matrix  $\mathcal{W}$ , relaxation factor  $\psi$ 
   Result:  $d_l$  and  $p$  which are the velocity field and the pressure field respectively
2    $d_l \leftarrow \mathbf{0}$ 
3    $\xi \leftarrow 1$ 
4   while  $\delta < \xi$  do
5        $\mathcal{U} \leftarrow \text{assembly}(d_l)$  // See 3.2 for more details
6        $\mathbf{C} \leftarrow \mathcal{N}^T \mathcal{U} \mathcal{W} \mathcal{B}$ 
7        $\mathbf{C}_{ll} \leftarrow \mathbf{C}(\text{DOFl}_v, \text{DOFl}_v)$ 
8        $\mathbf{C}_{lr} \leftarrow \mathbf{C}(\text{DOFl}_v, \text{DOFr}_v)$ 
9        $\mathbf{A} \leftarrow [\mathbf{K}_{ll} + \mathbf{C}_{ll} \mathbf{G}_l; \mathbf{G}_l^T \mathbf{0}]$ 
10       $\mathbf{B} \leftarrow [\mathbf{F}_l - \bar{\mathbf{u}}(\mathbf{K}_{lr} + \mathbf{C}_{lr}); -\mathbf{G}_r^T \bar{\mathbf{u}}]$ 
11       $(d_l^{new}, p) \leftarrow \text{sol}(\mathbf{A} \mathbf{x} = \mathbf{B})$ 
12       $\xi \leftarrow \max(||d_l^{new} - d_l||)$ 
13       $d_l \leftarrow (1 - \psi)d_l + \psi d_l^{new}$ 
14  end

```

---

## Chapter 3

# Vectorization and code optimization

In the previous chapter, it has been seen how to assemble the different matrices required to solve the steady state Navier-Stokes equations from a theoretical point of view. In this chapter, the Matlab implementation will be explained, which traditionally loops over all elements. As there can be on the order of  $10^5$  or even higher number of elements, it results in poor performance, especially as Matlab is an interpreted language. For this reason in the second part of the chapter a different approach will be taken to compute the FEM matrices, and the subsequent time saves will be analyzed.

### 3.1 Naive assembly

The typical assembly algorithm is performed as follows:

**Algorithm 2:** Assembly process for matrix  $\mathbf{K}$ 


---

```

1 Function  $K \leftarrow \text{AssemblyK}(COOR_v, CN_v)$ :
   Data:  $COOR_v$   $n_{nod} \times n_{sd}$  Velocity nodal coordinates matrix,  $CN_v$   $n_{el} \times n_{nodE}$  Velocity element
         connectivity matrix
   Result:  $\mathbf{K}$  Global viscosity matrix
2  $K \leftarrow \mathbf{0}$  // Allocate memory for matrix  $\mathbf{K}$ 
3 for  $e=1; e \leq n_{el}$  do
4     Nodes  $\leftarrow CN_v(e, :)$ 
5      $X_e \leftarrow COOR_v(\text{Nodes}, :)^T$ 
6      $K_e \leftarrow \text{ComputeKe}(X_e, \text{TypeElement})$  // See section 2.5.1 for more details on how to
         calculate the elemental matrix
7     for  $a=1; a \leq n_{nod}$  do
8         for  $b=1; b \leq n_{nod}$  do
9              $A \leftarrow \text{Nod2DOF}(CN_v(e, a))$  // Function Nod2DOF transforms the node number to a
                 global DOF according to (2.40).
10             $B \leftarrow \text{Nod2DOF}(CN_v(e, b))$ 
11             $a1 \leftarrow \text{Nod2DOF}(a)$ 
12             $b1 \leftarrow \text{Nod2DOF}(b)$ 
13             $K(A, B) \leftarrow K(A, B) + K_e(a1, b1)$ 
14        end
15    end
16 end

```

---

This assembly process has many nested loops, which are specially slow in an interpreted language such as Matlab. With the exterior for looping over the total number of elements, the computation time is really high.

$$\begin{aligned}
& \begin{bmatrix} [1] & [2] \\ K_1^e & K_2^e \\ K_3^e & K_4^e \end{bmatrix} \begin{matrix} [1] \\ [2] \end{matrix} \\
& \begin{bmatrix} [2] & [3] \\ K_1^e & K_2^e \\ K_3^e & K_4^e \end{bmatrix} \begin{matrix} [2] \\ [3] \end{matrix} \\
& \begin{bmatrix} [1] & [4] \\ K_1^e & K_2^e \\ K_3^e & K_4^e \end{bmatrix} \begin{matrix} [1] \\ [4] \end{matrix} \\
& \begin{bmatrix} [4] & [2] \\ K_1^e & K_2^e \\ K_3^e & K_4^e \end{bmatrix} \begin{matrix} [4] \\ [2] \end{matrix}
\end{aligned}
\quad
K = \begin{bmatrix}
\begin{matrix} [1] & [2] & [3] & [4] \end{matrix} \\
\begin{matrix} K_1^e + K_1^e & K_2^e & 0 & K_2^e \\ K_3^e & K_4^e + K_1^e + K_4^e & K_2^e & K_3^e \\ 0 & K_3^e & K_4^e & 0 \\ K_3^e & K_2^e & 0 & K_4^e + K_1^e \end{matrix}
\end{bmatrix} \begin{matrix} [1] \\ [2] \\ [3] \\ [4] \end{matrix}$$

Figure 3.1: Visualization of the assembly process from the elemental matrices to the global extracted from [16]. <sup>1</sup>

## 3.2 Vectorization

To speed up the code, vectorization will be used. This process consists in assembling the FEM matrices as the product of different matrices. The derivation for this method can be found in [14], but it mainly consists in finding  $\mathcal{N}$  and  $\mathcal{B}$ , which are the N-stacked and B-stacked matrices respectively. They are defined as:

$$\mathcal{N} := \text{diag} (N_1^1, N_2^1, \dots, N_1^e, N_2^e, \dots, N_{m-1}^{n_{el}}, N_m^{n_{el}}) L \quad (3.1)$$

and similarly:

$$\mathcal{B} := \text{diag} (B_1^1, B_2^1, \dots, B_1^e, B_2^e, \dots, B_{m-1}^{n_{el}}, B_m^{n_{el}}) L \quad (3.2)$$

where  $N_m^e$  and  $B_m^e$  are the shape functions and derivatives of shape functions matrices respectively, at element  $e$  at Gauss point  $m$ .  $L$  is the global boolean connectivity matrix (see [14] for more details). Matrices  $\widehat{\mathcal{N}}$  and  $\widehat{\mathcal{B}}$  have the same definitions but with pressure points and shape functions.

In this report, the viscosity will be assumed to be the same everywhere in the domain, but it can be generalized to different values for different elements like this:

$$\mathcal{C} := \text{diag} (\nu_1^1, \nu_2^1, \dots, \nu_1^e, \nu_2^e, \dots, \nu_{m-1}^{n_{el}}, \nu_m^{n_{el}}) \quad (3.3)$$

The product of all Jacobians and Gauss weights at each point is defined as:

$$\mathcal{W} := \text{diag} (w_1^1 |J|_1^1, w_2^1 |J|_2^1, \dots, w_1^e |J|_1^e, w_2^e |J|_2^e, \dots, w_{m-1}^{n_{el}} |J|_{m-1}^{n_{el}}, w_m^{n_{el}} |J|_m^{n_{el}}) \quad (3.4)$$

<sup>1</sup>In my particular case, as there are 9 nodes per element and it is a 2D problem, elemental matrices are 18x18. This is a simplification for small unidimensional problems.

The matrix  $\mathbf{U}$  containing the velocity of each element at each point is defined as:

$$\mathbf{U} := \text{diag}(\bar{\mathbf{u}}_1^1, \bar{\mathbf{u}}_2^1, \dots, \bar{\mathbf{u}}_1^e, \bar{\mathbf{u}}_2^e, \dots, \bar{\mathbf{u}}_{m-1}^{n_{el}}, \bar{\mathbf{u}}_m^{n_{el}}) \quad (3.5)$$

where  $\bar{\mathbf{u}}$  follows the mapping:

$$\bar{\mathbf{u}}_1^1 = \tilde{\mathbf{S}}(\mathbf{u}_1^1) = \begin{bmatrix} u_{1,x}^1 & 0 & u_{1,y}^1 & 0 \\ 0 & u_{1,x}^1 & 0 & u_{1,y}^1 \end{bmatrix} \quad (3.6)$$

The matrix  $\mathbf{I}$  that relates pressure shape functions to velocity derivatives of shape functions is defined as:

$$\mathbf{I} := \text{diag} \left( \overbrace{[1 \ 0 \ 0 \ 1], \dots, [1 \ 0 \ 0 \ 1]}^{n_m \times n_{el}} \right) \quad (3.7)$$

Finally the different global matrices can be calculated as the products of the corresponding stacked matrices, instead of the slow assembly process:

$$\mathbf{K} = \mathbf{B}^T \mathbf{C} \mathbf{W} \mathbf{B} \quad (3.8)$$

$$\mathbf{G} = \hat{\mathbf{N}}^T \mathbf{I} \mathbf{W} \mathbf{B} \quad (3.9)$$

$$\mathbf{C} = \mathbf{N}^T \mathbf{U} \mathbf{W} \mathbf{B} \quad (3.10)$$

### 3.3 Time savings

Now the performance of the code will be evaluated. Profiling is made in order to know the time saved to calculate the global FEM matrices as described in the previous section, 3.2.

Total DOF velocity & pressure	Assembly K [s]		Assembly G [s]	Assembly C [s]	
	Vect.	Naive <sup>2</sup>	Vect.	Naive	Vect.
3803	0.004	1.129	0.006	0.935	0.005
8450	0.021	4.677	0.005	4.03	0.01
18500	0.017	49.4	0.011	45.5	0.022
51378	0.030	451	0.027	392	0.086
73000	0.22	1179	0.037	1050	0.134
255000	0.96	-	0.121	-	0.398

Table 3.1: Performance difference of the naive assembly process versus the vectorized process.

First of all, an astonishing time difference of up to 5000x can be observed. This is, in part, because Matlab

<sup>2</sup>There is only one time for the assembly of  $\mathbf{K}$  and  $\mathbf{G}$  in the naive implementation because the loop over elements is the same in both cases, so to maximize performance the two assemblies are done together.

is an interpreted language so matrix multiplications are particularly fast while loops are particularly slow. Table 3.1 also shows how if the problem is too big and has too many DOF, it is not feasible to solve it by the standard approach. This speedup is important when solving Stokes flow, where matrices  $\mathbf{K}$  and  $\mathbf{G}$  only need to be computed once. However, it is crucial in Navier-Stokes flow where matrix  $\mathbf{C}$  needs to be computed at each iteration, because it becomes unreasonable to proceed with the standard assembly. For example, the case with 73000 DOF, needed 150 iterations. This means that matrix  $\mathbf{C}$  has to be assembled 150 times, thus it would take 157500 s or 1 day and 19h, without taking into account the rest of the program. It can be concluded that the only way to solve the Navier-Stokes equations in Matlab is with vectorization or else the performance would be too slow to do any meaningful simulation.

### 3.4 Profiling

To evaluate the performance of the code to know where it could be further optimized, one simulation is profiled. The case chosen is the lid driven cavity shown in sections 5.1.1 and 5.2.1, in particular this is the case with 51378 DOF that appears in table 3.1. Using Matlab's built in profiler tool, the results are obtained and attached in appendix A.

The results provide some interesting information:

- The total runtime is 96s.
- Most of the time, 84.78s or around 88% of the runtime, is spent solving the linear system of equations described in line 11 of Picard's algorithm, 1.
- The time spent assembling  $\mathbf{K}$  and  $\mathbf{G}$  matrices is negligible.
- The time spent assembling  $\mathbf{C}$  is around 8.4s.
- A considerable time, around 1s, is spent reading and writing data.

With these results it becomes obvious that the function that needs to be optimized is the linear solver. As this thesis focuses on the FEM, the function will not be optimized, nor will the rest of the code as it has an acceptable level of performance.

## Chapter 4

# Reduced order modeling

Imagine that an analysis of an airfoil needs to be computed at different velocities, let us say from 1 to 20 m/s. If a resolution of 0.5 m/s is wanted, 38 different simulations must be performed, 1m/s, 1.5m/s, ..., 19.5m/s and 20m/s. If the Reynolds number is high, this becomes quite a challenge as the computational cost increases with the Reynolds number, so a considerable time is needed. In this chapter a method for reducing this time by *compressing* the simulations is presented. First a velocity basis matrix is generated in the training stage, then from this velocity basis matrix subsequent simulations can be performed with much less computational cost. Returning to the above example, let's say only 3 simulations are performed at 1 m/s, 10 m/s and 20 m/s. A model is obtained with the results from the 3 simulations. Then, the intermediate values can be computed from the model, not having to interpolate the results directly which gives a worse result.

The main idea of this chapter is to assume a solution of the NS equations where space and time are independent so that:

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{a}(t)\Phi(\mathbf{x}) \quad (4.1)$$

where the variable  $\mathbf{a}(t)$  includes all the time dependence of the equations and  $\Phi(\mathbf{x})$  characterizes the spatial dependence. The objective of this chapter will be to find  $\Phi(\mathbf{x})$  [17], [18].

### 4.1 Parametric problem

First, the supposition is made that different simulations must be performed varying the input parameters, a typical case when analysing aircraft. Let us suppose that the geometry and fluid properties remain constant, so the only variations are on the prescribed velocities  $\bar{\mathbf{u}}$  and tractions  $\bar{\mathbf{t}}$  which affect the external force vector  $\mathbf{F}$ . Thus the following can be written,  $\mathbf{F} = \mathbf{F}(\boldsymbol{\zeta})$  and  $\bar{\mathbf{u}} = \bar{\mathbf{u}}(\boldsymbol{\zeta})$ , where  $\boldsymbol{\zeta}$  is the vector of input parameters,  $\boldsymbol{\zeta} \in \mathbb{R}^{m_\zeta}$

The intrinsic dimension of the input vectors  $\mathbf{F}$  and  $\boldsymbol{\zeta}$  can be inferred by introducing the decompositions:

$$\mathbf{F}(\boldsymbol{\zeta}) = \boldsymbol{\Theta}_1 \boldsymbol{\alpha}_1(\boldsymbol{\zeta}) + \boldsymbol{\Theta}_2 \boldsymbol{\alpha}_2(\boldsymbol{\zeta}) + \dots + \boldsymbol{\Theta}_n \boldsymbol{\alpha}_n(\boldsymbol{\zeta}) = \boldsymbol{\Theta} \boldsymbol{\alpha}(\boldsymbol{\zeta}) \quad (4.2)$$

$$\bar{\mathbf{u}}(\boldsymbol{\zeta}) = \mathbf{D}_1^u \beta_1^u(\boldsymbol{\zeta}) + \mathbf{D}_2^u \beta_2^u(\boldsymbol{\zeta}) + \dots + \mathbf{D}_n^u \beta_n^u(\boldsymbol{\zeta}) = \mathbf{D}^u \boldsymbol{\beta}^u(\boldsymbol{\zeta}) \quad (4.3)$$

where  $\boldsymbol{\Theta}$  and  $\mathbf{D}^u$  are matrices whose columns represent (nodal) spatial patterns and  $\boldsymbol{\beta}^u(\boldsymbol{\zeta})$  and  $\boldsymbol{\alpha}(\boldsymbol{\zeta})$  represent the coefficients of the patterns at an input  $\boldsymbol{\zeta}$ . Hence,  $\mathbf{F}(\boldsymbol{\zeta})$  and  $\bar{\mathbf{u}}(\boldsymbol{\zeta})$  must belong in the linear subspace spanned by the columns of these matrices. Therefore the dimensionality of the force and displacement input spaces is  $\text{rank}(\boldsymbol{\Theta})$  and  $\text{rank}(\mathbf{D}^u)$ , respectively.

## 4.2 Solution space

Generally, it would be expected that  $n = \mathcal{O}(m_\zeta) > m_\zeta$  (at least for moderate Reynolds), that is, the dimension of the solution space is expected to be of the same order of magnitude as the dimension of the input space. However, from this point onwards it will be assumed that the dimension of the input space is much smaller than the dimension of the finite element space:

$$m_\zeta \ll n \quad (4.4)$$

The dependence of the nodal velocities  $\mathbf{d}$  with respect to the inputs  $\mathbf{F}$  and  $\bar{\mathbf{u}}$  is in general nonlinear. It can be shown that the set of all solutions  $\mathbf{d}_1 = \mathbf{d}_1(\boldsymbol{\zeta})$  is, in general, a manifold of dimension  $m_\zeta$ . Finding this manifold is not a trivial problem and is left outside the scope of this thesis. Rather, a vector space of dimension  $n$  containing the above mentioned manifold will be determined [18].

The higher the degree of nonlinearity, the larger the difference between both dimensions. Notice that in the case of a linear problem  $n = m_\zeta$ . This leaves Stokes flow outside of the analysis as it is a linear problem thus it will not benefit from this approach as the dimensions of the reduced model are the same as the dimensions of the input space, so no reduction is possible.

The idea now is to find a basis matrix that represents the solution space and, hopefully, it is smaller than the full solution space so less data is needed and the problem can be compressed.

The solution space will be represented in what follows by a basis matrix  $\boldsymbol{\Phi} \in \mathbb{R}^{n \times n}$ :

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_1 & \boldsymbol{\Phi}_2 & \dots & \boldsymbol{\Phi}_n \end{bmatrix} \quad (4.5)$$

Some assumptions that will be made going forward:  $\boldsymbol{\Phi}^T \mathbf{M} \boldsymbol{\Phi} = \mathbf{I}$ , where  $\mathbf{M}$  is some symmetric positive definite matrix. Also,  $\mathbf{M} = \mathbf{I}$  so  $\boldsymbol{\Phi}^T \boldsymbol{\Phi} = \mathbf{I}$ . Nevertheless, it would be more accurate to use the geometric mass matrix  $\mathbf{M} = \mathcal{N}^T \mathcal{W} \rho \mathcal{N}$ , specially for irregular meshes.

Finally, the determination of the basis matrix is done in two subsequent stages:



1. Training
2. Dimensionality reduction

### 4.3 Training stage

Firstly, algorithm 1 must be modified. As the problem is a steady state case, a *fictional* time will be introduced. Instead of solving the whole problem at once, a loading parameter  $t \in [0, T]$ , that can be thought of as time, is introduced. Then, the interval  $[0, T]$  is divided into  $n_{stp}$  subintervals:

$$[0, T] = [0, t_1] \cup [t_1, t_2] \cup \dots \cup [t_n, t_{n+1}] \cup \dots \cup [t_{n_{stp}-1}, T] \quad (4.6)$$

Following, (2.46) will be solved at each step with  $\bar{\mathbf{u}}(t) = \frac{\bar{\mathbf{u}}}{n_{stp}}t$  and the velocity results will be stored in the snapshot matrix,  $\mathbf{A}^u$ . Pressure will not be studied as it is the same procedure than velocity so it does not provide new meaningful results.

The snapshot matrix is defined as:

$$\mathbf{A}^u = [\mathbf{d}_1(\boldsymbol{\zeta}^1, t_1), \mathbf{d}_1(\boldsymbol{\zeta}^1, t_2), \dots, \mathbf{d}_1(\boldsymbol{\zeta}^2, t_1), \mathbf{d}_1(\boldsymbol{\zeta}^2, t_2), \dots, \mathbf{d}_1(\boldsymbol{\zeta}^P, t_{n_{stp}-1}), \mathbf{d}_1(\boldsymbol{\zeta}^P, T)] \quad (4.7)$$

So for each input vector  $\boldsymbol{\zeta}$ , that is to say for each simulation with different boundary conditions, the information at several time steps will be stored. Once sufficient data is gathered and stored into the snapshot matrix, the next step can be performed.

### 4.4 Dimensionality reduction

To obtain the velocity basis matrix  $\Phi$ , it is computed as a linear combination of the columns of  $\mathbf{A}^u$ , so the column space of  $\Phi$  is a subspace of the column space of  $\mathbf{A}^u$

$$\text{span}(\Phi) \subset \text{span}(\mathbf{A}^u) \quad (4.8)$$

The idea being that the number of columns of  $\Phi$  is as small as possible while retaining all relevant information. Some error threshold is defined as  $0 \leq \epsilon^u < 1$  such that

$$\|\mathbf{A}^u - \Phi \Phi^T \mathbf{A}^u\| \leq \epsilon^u \|\mathbf{A}^u\| \quad (4.9)$$

where  $\|\bullet\|$  denotes a suitable matrix norm, such as the Frobenius norm defined as:

$$\|\mathbf{A}\|_F = \sqrt{\sum_i^m \sum_j^n |a_{ij}|^2} = \sqrt{\text{trace}(\mathbf{A}^T \mathbf{A})} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2(\mathbf{A})}$$

where  $\sigma_i$  are the singular values of  $\mathbf{A}$ .

To obtain these basis functions, the truncated singular value decomposition will be used. It must satisfy:

$$\tilde{\mathbf{A}}^u = \mathbf{Y}_h \mathbf{\Sigma}_h \mathbf{Z}_h \quad (4.10)$$

where  $\tilde{\mathbf{A}}^u$  is an approximation of  $\mathbf{A}^u$  with only the  $h$  singular values calculated such that it satisfies (4.9). Matrices  $\mathbf{Y}$ ,  $\mathbf{\Sigma}$  and  $\mathbf{Z}$  are the standard left singular vectors, singular values and right singular vectors respectively. Then it can be proven that the basis matrix is  $\mathbf{\Phi} = \mathbf{Y}_h$  [17].

## 4.5 Angle between subspaces

Now that the basis matrix is known for each case (different input parameters), we wish to compare basis matrices to know if, for example, the information for a low Reynolds number simulation is contained within a high Reynolds simulation. Thus only requiring one basis matrix instead of two. The first idea that comes to mind to compare 2 different matrices is to subtract one from another and check whether the result is 0. As the comparison is actually between the 2 subspaces represented by the matrices, this procedure makes no sense, so the principal angles, a generalization of the notion of angles between straight lines, will be computed instead. The following definitions and algorithm are based on [19].

Let  $\mathbf{\Lambda}$  and  $\mathbf{\Xi}$  be subspaces in  $\mathbb{R}^m$  whose dimensions satisfy

$$p = \dim(\mathbf{\Lambda}) \geq \dim(\mathbf{\Xi}) = q \geq 1 \quad (4.11)$$

The principal angles  $\{\theta_i\}_{i=1}^q$  between these two subspaces and the associated principal vectors  $\{\boldsymbol{\lambda}_i, \boldsymbol{\xi}_i\}_{i=1}^q$  are defined recursively by

$$\cos \theta_k = \boldsymbol{\lambda}_k^T \boldsymbol{\xi}_k = \max_{\substack{\boldsymbol{\lambda} \in \mathbf{\Lambda}, \|\boldsymbol{\lambda}\|_2=1 \\ \boldsymbol{\lambda}^T [\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_{k-1}] = 0}} \max_{\substack{\boldsymbol{\xi} \in \mathbf{\Xi}, \|\boldsymbol{\xi}\|_2=1 \\ \boldsymbol{\xi}^T [\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_{k-1}] = 0}} \boldsymbol{\lambda}^T \boldsymbol{\xi} \quad (4.12)$$

The algorithm to calculate the principal angles between two subspaces is:

---

**Algorithm 3:** Algorithm for computing the principal angles between subspaces.

---

```

1 Function  $\cos \theta_k \leftarrow (\mathbf{\Lambda}, \mathbf{\Xi})$ :
   | Data:  $(\mathbf{\Lambda}, \mathbf{\Xi})$  two subspaces with  $\mathbf{\Lambda} \in \mathbb{R}^{m \times p}$  and  $\mathbf{\Xi} \in \mathbb{R}^{n \times p}$ 
   | Result:  $\cos \theta_k$  cosines of principal angles
2    $[Q_1, R_1] \leftarrow \text{Thin QR decomposition}(\mathbf{\Lambda})$ 
3    $[Q_2, R_2] \leftarrow \text{Thin QR decomposition}(\mathbf{\Xi})$ 
4    $\mathbf{C} \leftarrow Q_1^T Q_2$ 
5    $[\mathbf{Y}, \mathbf{\Sigma}, \mathbf{Z}] \leftarrow \text{svd}(\mathbf{C})$ 
6    $\cos \theta_k \leftarrow \text{diag}(\mathbf{\Sigma})$ 

```

---

## 4.6 Projection onto the reduced space

In this section the procedure for constructing the reduced order model will be explained from a theoretical point of view, the Matlab implementation is left for future projects.

The construction of the reduced order model consists in projecting the vectors  $\mathbf{d}_1$  and  $\mathbf{c}_t$  onto the basis functions subspace  $\Phi$  with the goal of using the reduced space, not the larger finite dimensional space. Remember that  $\mathbf{d}$  admits the decomposition:

$$\mathbf{d} = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_r \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ D^u \beta^u(t) \end{bmatrix} \quad (4.13)$$

From the previous sections:

$$\mathbf{d}_1 = \Phi \kappa = \Phi_1 \kappa_1 + \Phi_2 \kappa_2 + \dots + \Phi_n \kappa_n \quad (4.14)$$

Similarly introducing a decomposition for the pressure:

$$\mathbf{p} = \Psi \chi = \Psi_1 \chi_1 + \Psi_2 \chi_2 + \dots + \Psi_m \chi_m \quad (4.15)$$

Then, it follows that:

$$\mathbf{d} = \begin{bmatrix} \Phi \kappa \\ D^u \beta^u \end{bmatrix} = \begin{bmatrix} \overbrace{\Phi \quad 0}^{=\bar{\Phi}} \\ \underbrace{0 \quad D^u}_{\bar{\kappa}} \end{bmatrix} \begin{bmatrix} \kappa \\ \beta^u \end{bmatrix} = \bar{\Phi} \bar{\kappa} \quad (4.16)$$

Following the same procedure as in [18], the reduced matrices can be found:

$$\mathcal{B}^* := \mathcal{B} \bar{\Phi} \quad (4.17)$$

$$\mathcal{N}^* := \mathcal{N} \bar{\Phi} \quad (4.18)$$

$$\hat{\mathcal{B}}^* := \hat{\mathcal{B}} \Psi \quad (4.19)$$

$$\hat{\mathcal{N}}^* := \hat{\mathcal{N}} \Psi \quad (4.20)$$

And the external force vector:

$$\mathbf{F}^* = \Theta^* \alpha = \bar{\Phi}^T \Theta \alpha \quad (4.21)$$

Note that these matrices will have the dimensions of the projection space which should be much smaller than the finite element space, thus reducing the computing power needed. So now it is a matter of constructing the reduced matrices  $\mathbf{K}^*$ ,  $\mathbf{G}^*$  and  $\mathbf{C}^*$  and the vector  $\mathbf{F}^*$  and solving equation (2.46) in the reduced space.

## Chapter 5

# Results

In this chapter the results of this thesis will be presented. The main result is the code per se, which can be found in appendix B. Here, some simulations performed with the code developed in the present work will be presented and compared against the same simulations using *Kratos Multiphysics*. Afterwards, the results of chapter 4 will be presented and the dimensionality of some problems will be analyzed.

### 5.1 Stokes flow simulations

First of all, Stokes flow will be studied, as represented by Eq.(2.47). Two different problems will be analyzed, the lid driven cavity and the flow around a cylinder.

#### 5.1.1 Lid driven cavity

The first case that will be studied is the lid driven cavity which is a famous benchmark for fluid codes [11]. The definition of the case is given in figure 5.1.

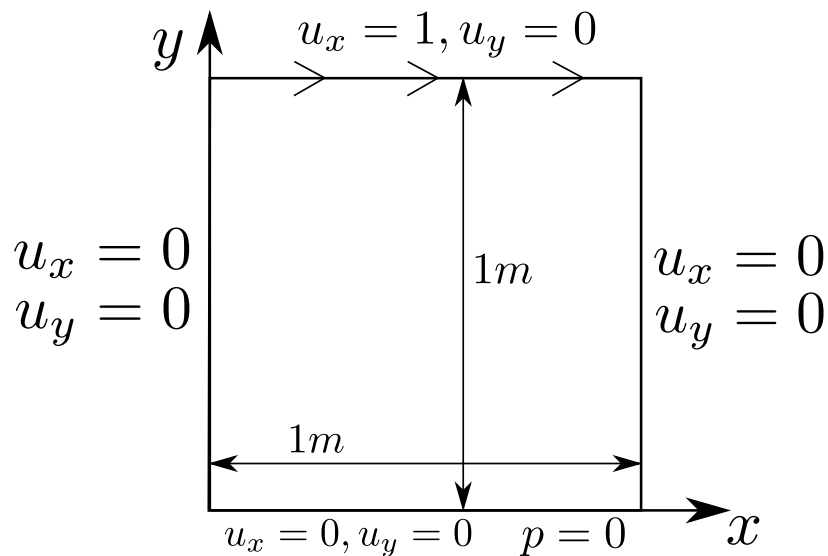


Figure 5.1: Definition of the lid driven problem

The boundary conditions in the two upper corners are discontinuous, so two options arise when deciding what to do with these nodes. They can belong either to the fixed vertical walls (non-leaky), or to the prescribed velocity top (leaky) [11, p. 319]. In this report the later condition is chosen.

Once the simulation is done, the x velocity component is compared against [11, Figure 6.11] as it is the only component available. The data has been extracted using [WebPlotDigitalizer](#) and the same simulation using *Kratos*. The results are:

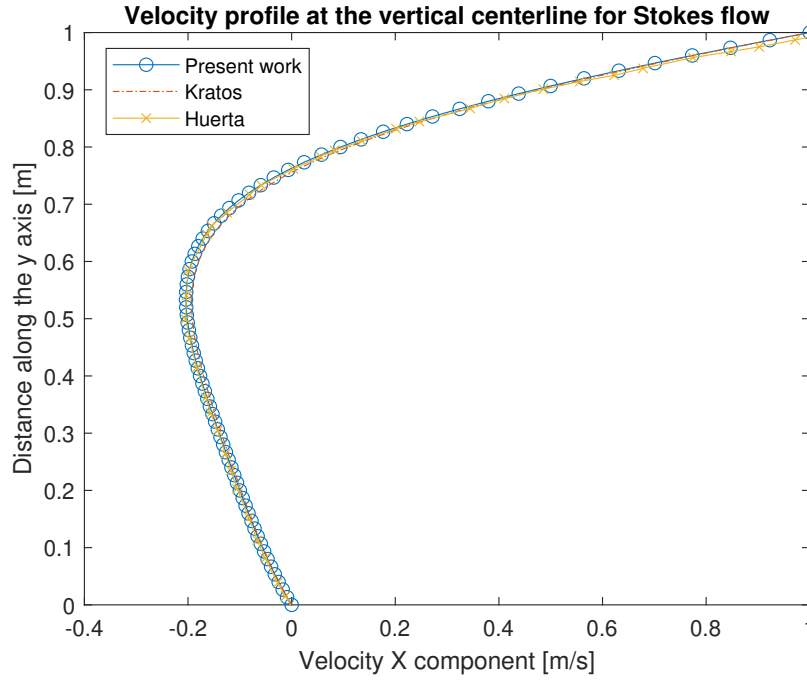


Figure 5.2: Comparison of the velocity x component at the vertical centerline between Kratos, Huerta [11] and the present work for stokes flow.

As for the plot of the velocity:

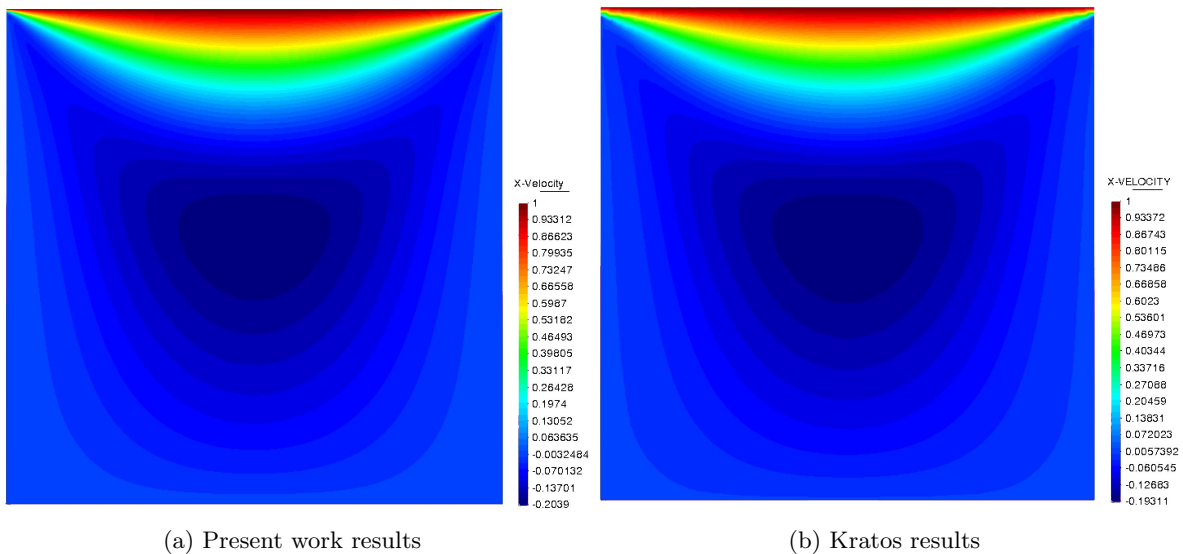


Figure 5.3: Comparisons of the x velocity results for Kratos and the present work's FEM fluid solvers

It can be seen on both figures the present results obtained are almost identical to those of Huerta [11], and *Kratos*.

### 5.1.2 Flow around a cylinder

In this section, the flow around a cylinder will be studied. Only results against *Kratos* will be compared. The definition for this flow is:

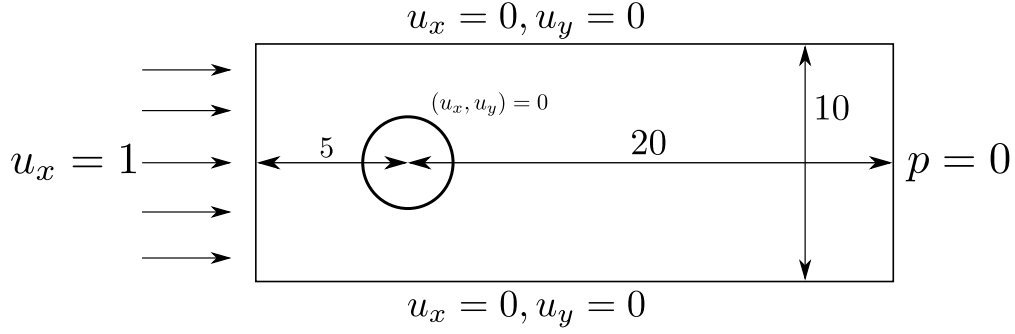


Figure 5.4: Definition of the case of flow around a cylinder. No slip condition assigned at the upper and lower walls and the cylinder.

As it can be seen, the upper and lower walls have imposed a zero velocity condition. This could represent the walls of a wind tunnel.

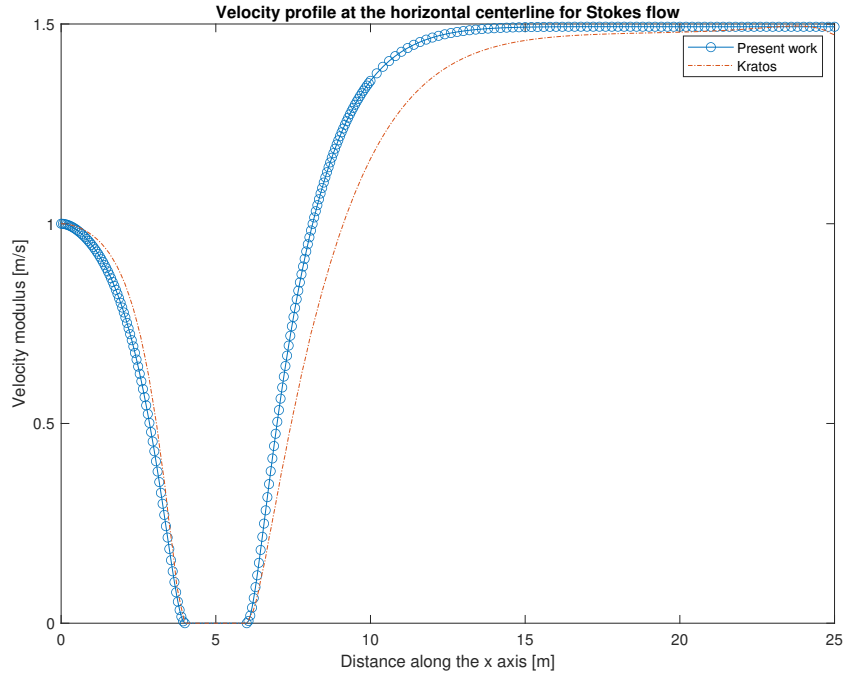


Figure 5.5: Comparison of the velocity modulus along the horizontal centerline for Stokes flow.

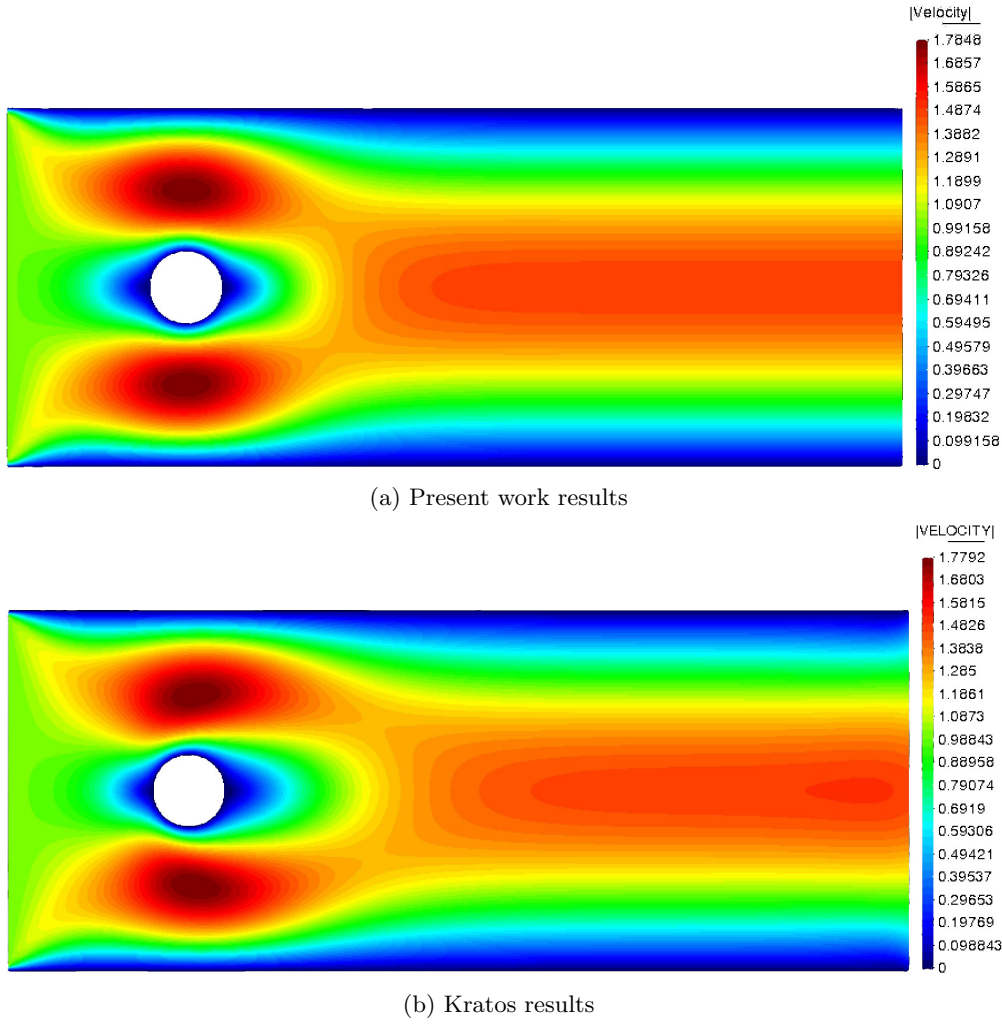


Figure 5.6: Velocity modulus plot for Stokes flow around a cylinder

Contrary to the lid driven cavity, there is a slight deviation from the results obtained by the present work's code to those obtained by *Kratos*. This deviation can occur because *Kratos* uses a different FEM formulation as described in 1.6, because the mesh is too coarse or because as *Kratos* does not have a Stokes flow option so the simulation has been performed with a high viscosity, as in the limit where the viscosity tends to infinity NS flow is equal to Stokes flow.

## 5.2 Navier-Stokes simulations

In this section, Eq.(2.46) will be solved to obtain the full Navier-Stokes steady state solution, then it will be compared against professional codes and literature to perform the validation. Different Reynolds numbers will be analyzed, as the difficulty of the simulation increases with the Reynolds numbers because of turbulence. As the code is a DNS code, not too high of a Reynolds will be tested as the computational time increases because all the fluid's scales must be represented, including Kolmogorov's scale. This means that element sizes must be very small, so the mesh gets very large very fast, and the problem can not be solved in a reasonable time by this report's code and on the author's computer [17].

On all these simulations the velocity imposed is of 1m/s and to change the Reynolds number the viscosity has been changed.

### 5.2.1 Lid driven cavity

The simulation is defined as in 5.1.1, only that now the nonlinear convective term will be calculated. Three different Reynolds numbers will be studied, 100, 400 and 1000.

#### Reynolds 100

First the velocity profile at the centerline is compared:

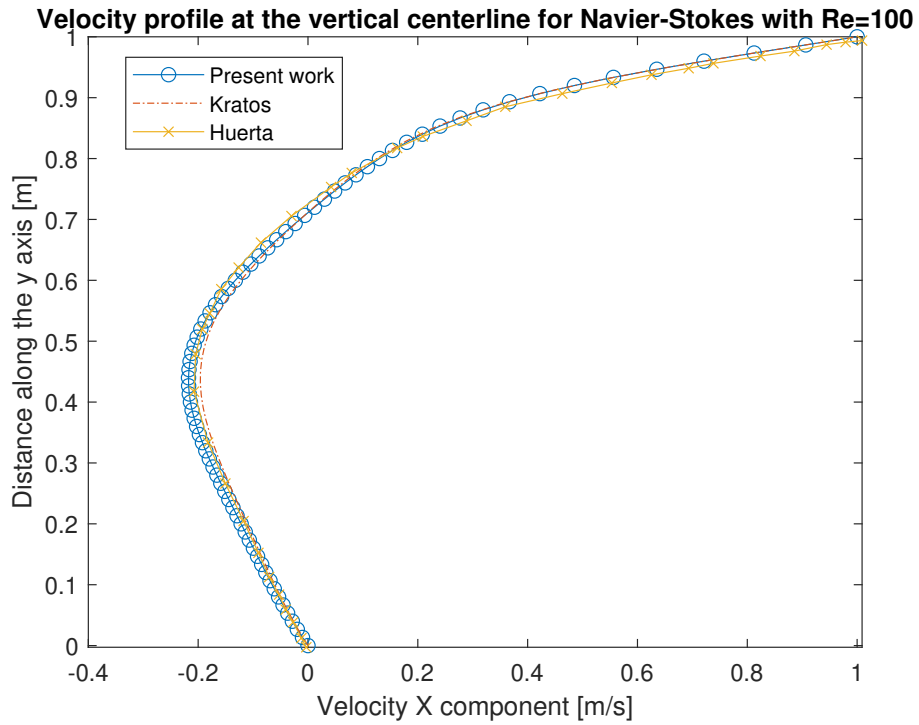


Figure 5.7: Comparison of the velocity X component at the vertical line between Kratos, Huerta [11] and the present work for Navier-stokes flow for a Reynolds number of 100.

Now, the pressure is compared:



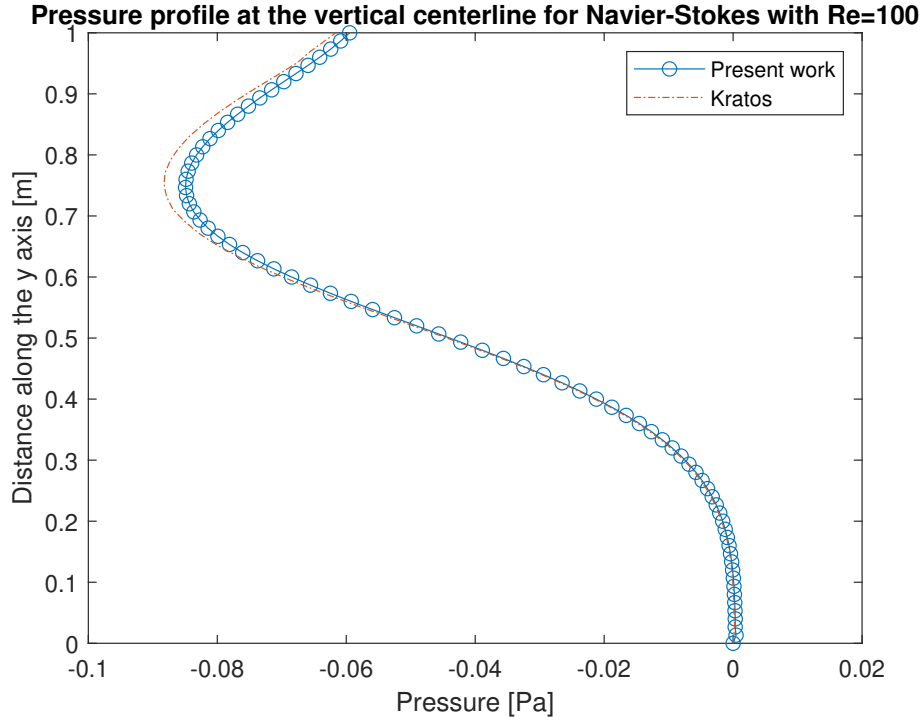


Figure 5.8: Comparison of the pressure at the vertical line between Kratos, and the present work for Navier-stokes flow for a Reynolds number of 100.

Finally the velocity can be plotted and compared against that obtained with *Kratos*.

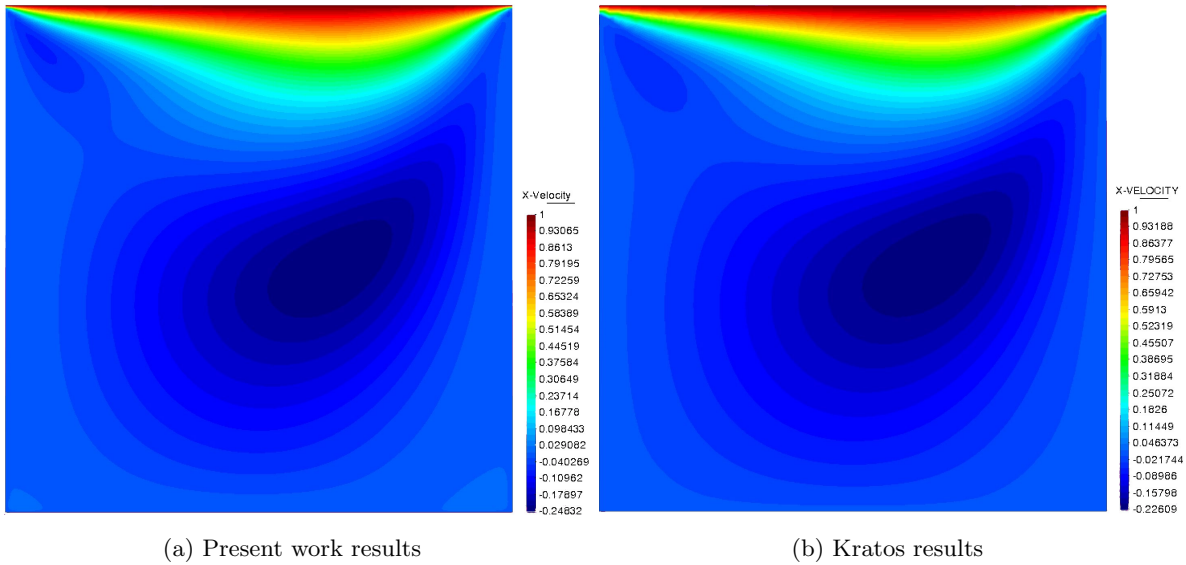


Figure 5.9: Comparisons of the X velocity results for Kratos and the present work's FEM fluid solvers

It can be seen that the velocity profiles match very well the literature and the *Kratos* simulation. However, the pressure does have some slight deviation that can be due to the size of the mesh, the FEM formulation used, or some other unknown factor.

**Reynolds 400**

As always, first the velocity comparisons:

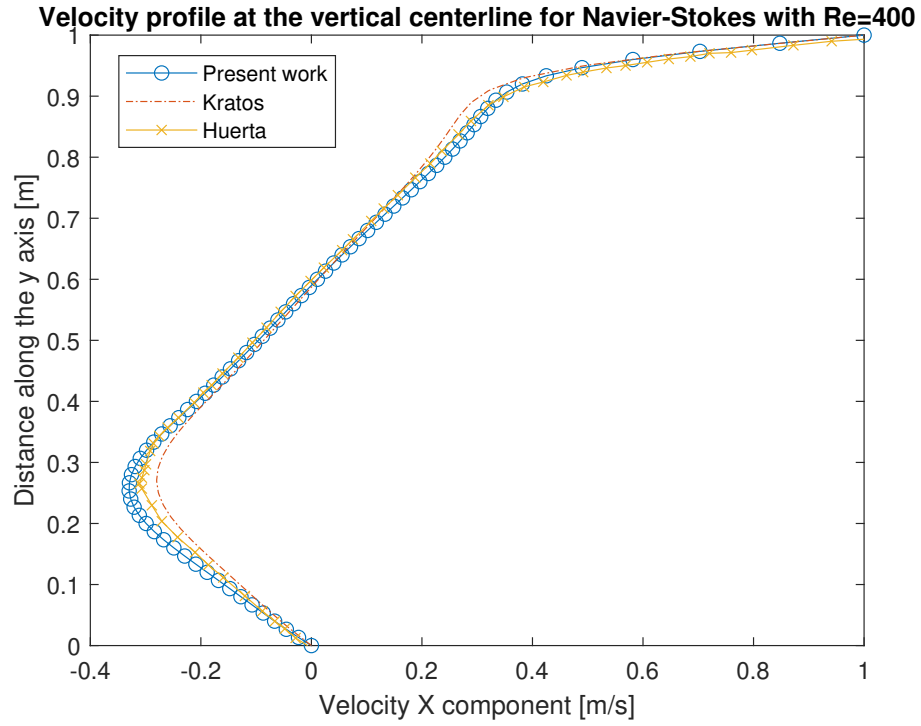


Figure 5.10: Comparison of the velocity X component at the vertical line between Kratos, Huerta [11] and the present work for Navier-stokes flow for a Reynolds number of 400.

Then the pressure profile is compared:

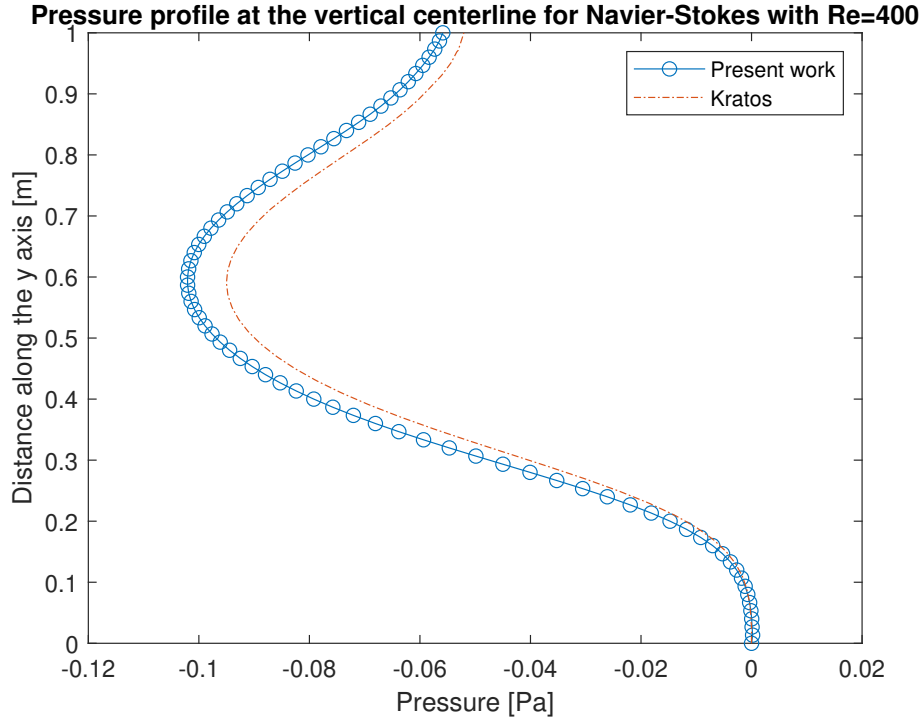


Figure 5.11: Comparison of the pressure at the vertical line between Kratos, and the present work for Navier-stokes flow for a Reynolds number of 400.

Finally, the modulus of the velocity can be visualized both for *Kratos* and for the code developed.

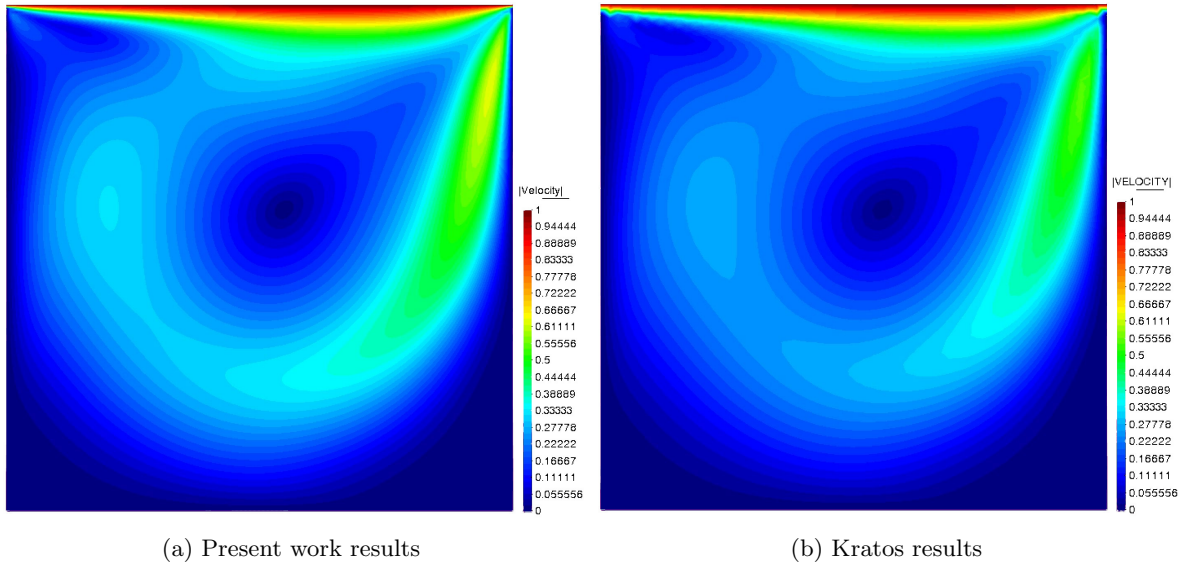


Figure 5.12: Comparisons of the velocity modulus results for Kratos and the present work's FEM fluid solvers

Once again the velocity profiles match quite well, but there is a deviation on the pressure. As the simulation is the same varying only the input parameters the reason for this deviation is the same as in section 5.2.1.

### Reynolds 1000

For this Reynolds number, the velocity profile and the pressure will be compared against literature and *Kratos*, and the Y velocity component and the pressure will be plotted.

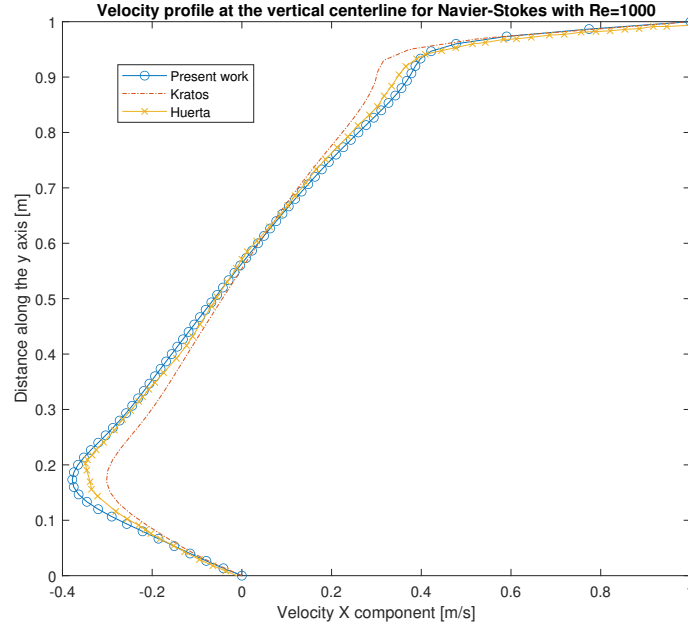


Figure 5.13: Comparison of the velocity X component at the vertical line between Kratos, Huerta [11] and the present work for Navier-stokes flow for a Reynolds number of 1000.

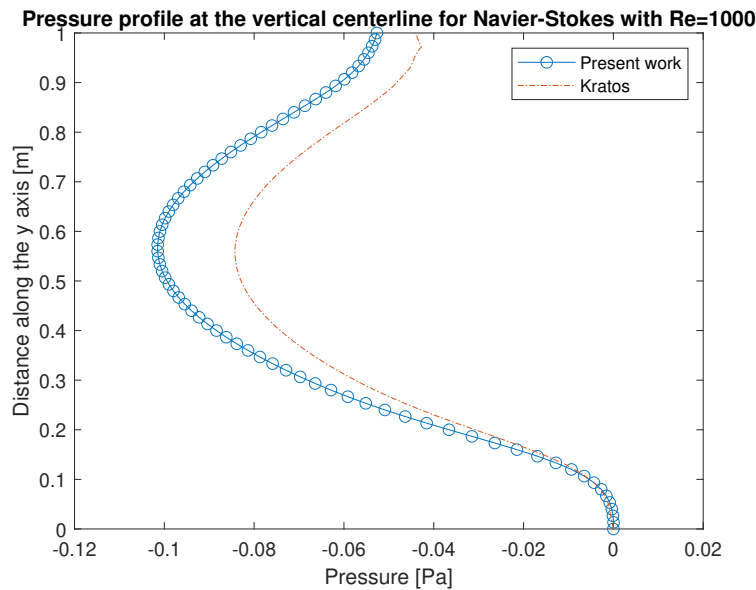


Figure 5.14: Comparison of the pressure at the vertical line between Kratos, and the present work for Navier-stokes flow for a Reynolds number of 1000.

The plots for the Y component of the velocity are:

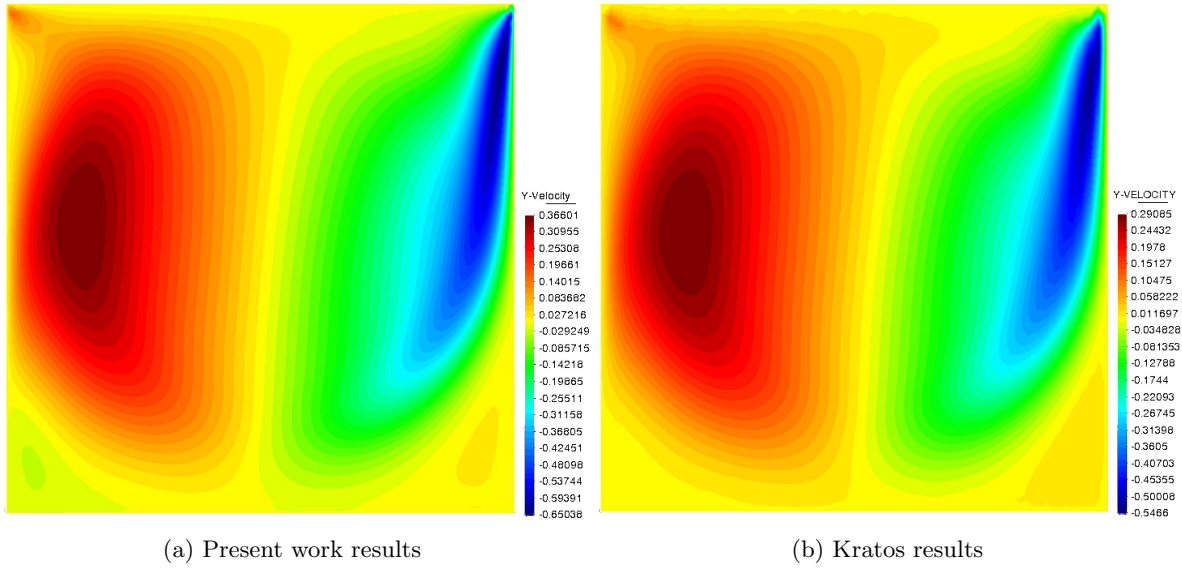


Figure 5.15: Comparisons of the velocity Y component results for Kratos and the present work's FEM fluid solvers

Finally, the plot for the pressure is:

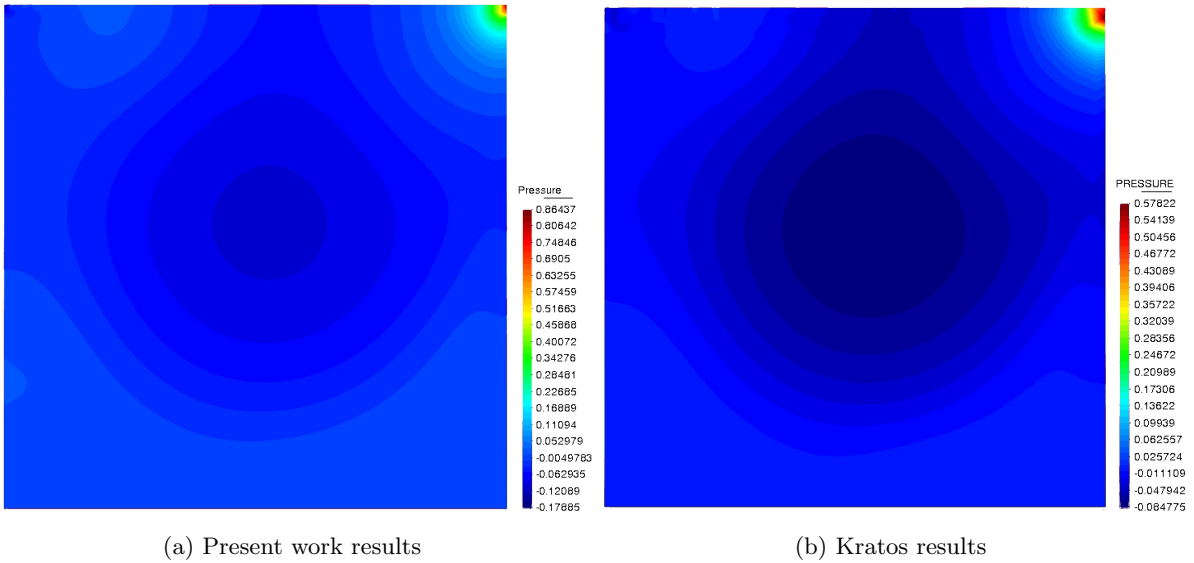


Figure 5.16: Comparisons of the pressure for Kratos and the present work's FEM fluid solvers

In this plots, several things can be observed:

- There is a singularity in the upper right corner, this is because as described in 5.1.1 the leaky cavity has been chosen.
- The value of this singularity is notably higher in my simulation than on *Kratos*'. This is because the mesh is finer in my case so the "jump" from  $u_x = 1$  to  $u_x = 0$  is done in a shorter space. In turn this means that a higher pressure gradient must occur to make this possible, explaining the difference in the peak pressure.

- The colour is very uniform except in the corner. This is because the spike in pressure in the corner "smoothens" the pressure in the rest of the domain. Then the postprocessor, GiD, assigns uniform colours. If the corner were removed, more detailed information could be seen.

### 5.2.2 Flow around a cylinder

Now the cylinder case will be revisited as described in 5.1.2 but with the full steady state Navier-Stokes equations. Only one Reynolds number has been represented as the higher Reynolds simulations do not give more insight into the code's performance.

#### Reynolds 50

Firstly, the velocity and pressure profiles have been plotted along the distance in the horizontal centerline and compared to those of *Kratos*

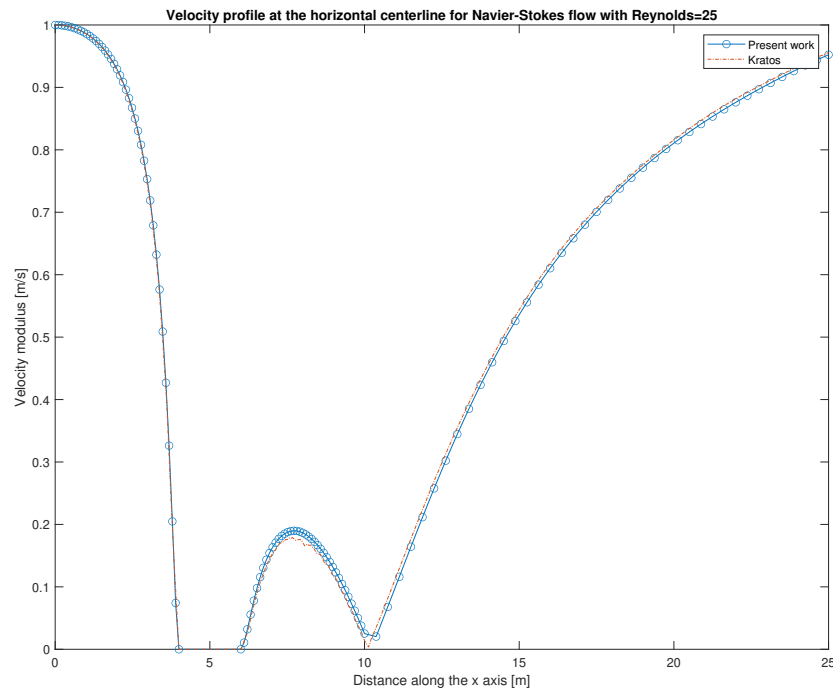


Figure 5.17: Comparison of the velocity modulus along the horizontal centerline for Navier-Stokes for a Reynolds number of 50.

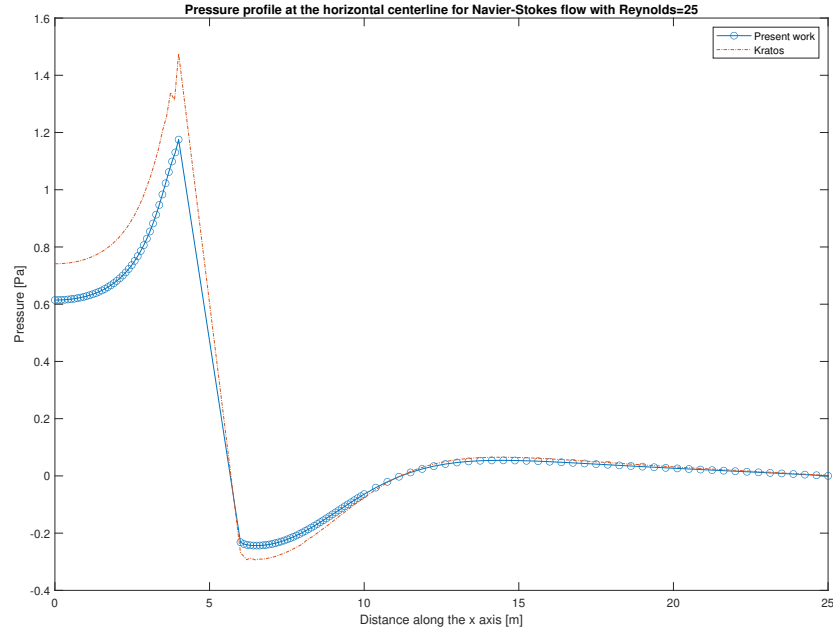
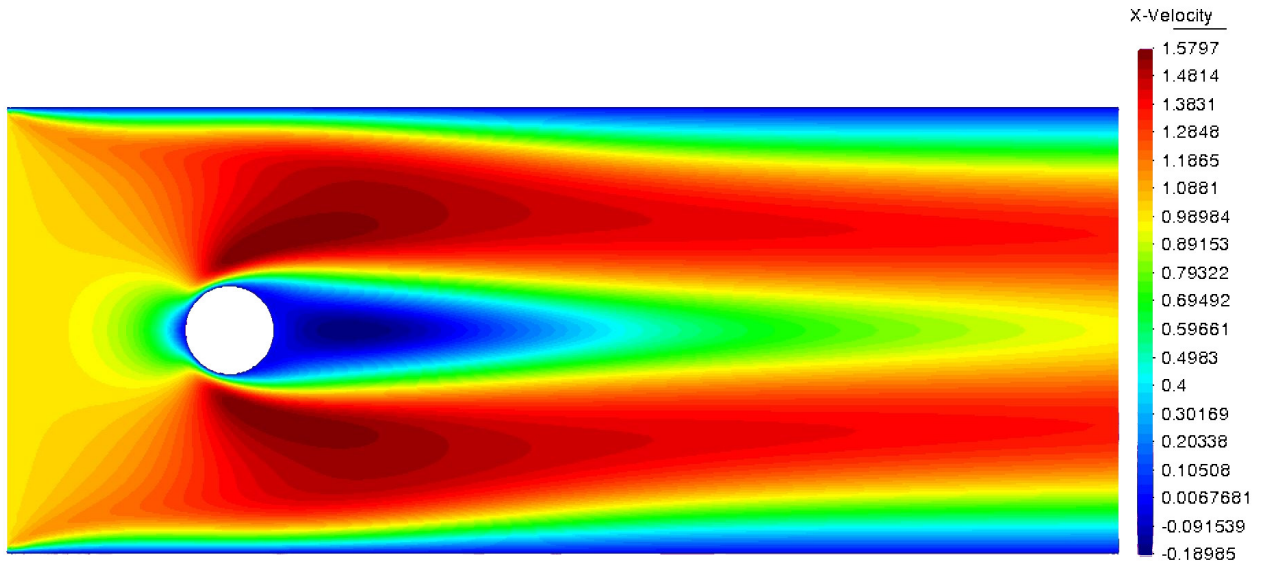


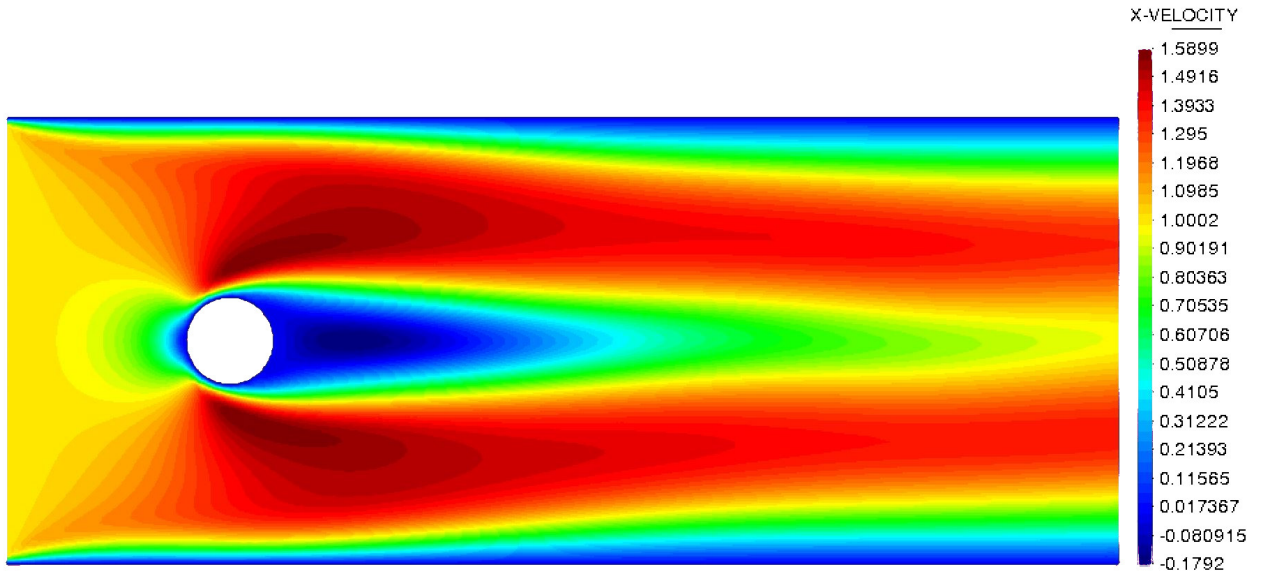
Figure 5.18: Comparison of the pressure profile along the horizontal centerline for Navier-Stokes for a Reynolds number of 50.

It can be observed that the velocity profile matches almost perfectly to that of *Kratos*. However, pressure does have some differences as in the lid driven cavity. The explanation for this difference is the same as in the lid driven cavity: the different meshes and the different formulations as *Kratos* uses VMS elements.

Secondly, plots of the velocity and pressure are represented:



(a) Present work results



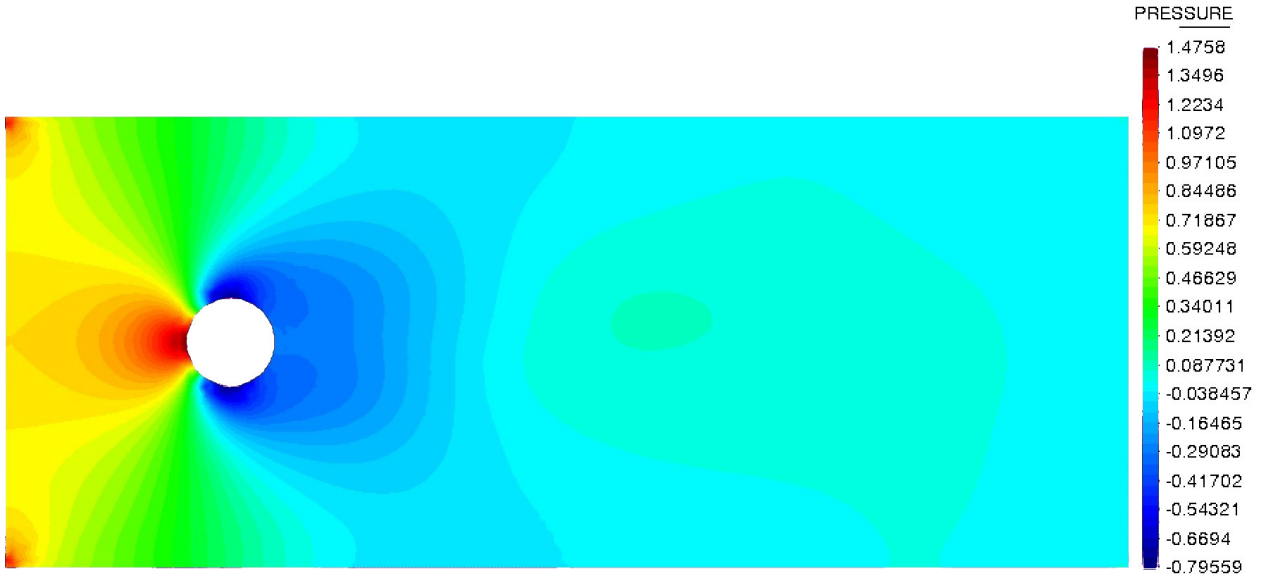
(b) Kratos results

Figure 5.19: Velocity profiles of the x velocity component for Navier-Stokes with a Reynolds number of 50.





(a) Present work results



(b) Kratos results

Figure 5.20: Pressure plot for Navier-Stokes with a Reynolds number of 50.

First of all, it can be observed on the pressure plots that there is a singularity on the left corners, this is the same case as in the leaky cavity where there is a discontinuity in the boundary. Secondly, observe that the profiles are almost identical.

As both this case and the lid driven case obtain good results when compared to literature and to professional codes, the code can be considered validated.

### 5.2.3 NACA 0012

Once the code has been validated with the above results, simulations with utility can be performed. To illustrate this, the flow around a NACA 0012 airfoil will be computed. Two Reynolds numbers have been

tested: 200 and 500. No higher Reynolds were tested because the computational time increase is too high. An angle of attack of  $5^\circ$  has been chosen arbitrarily. The definition for this case is in figure 5.21.

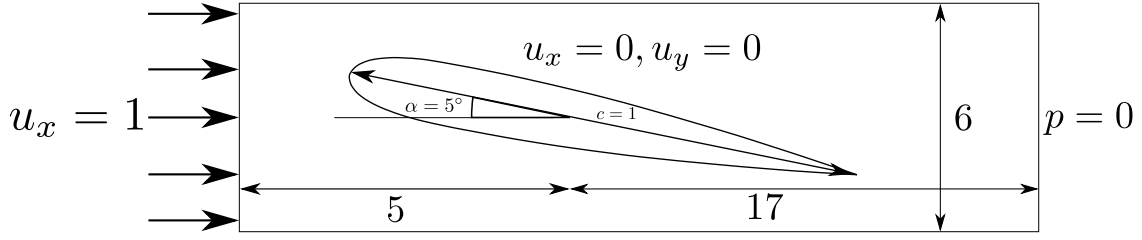


Figure 5.21: Case definition for the NACA 0012 simulation.

### Reynolds 200

The results for this Reynolds number are:

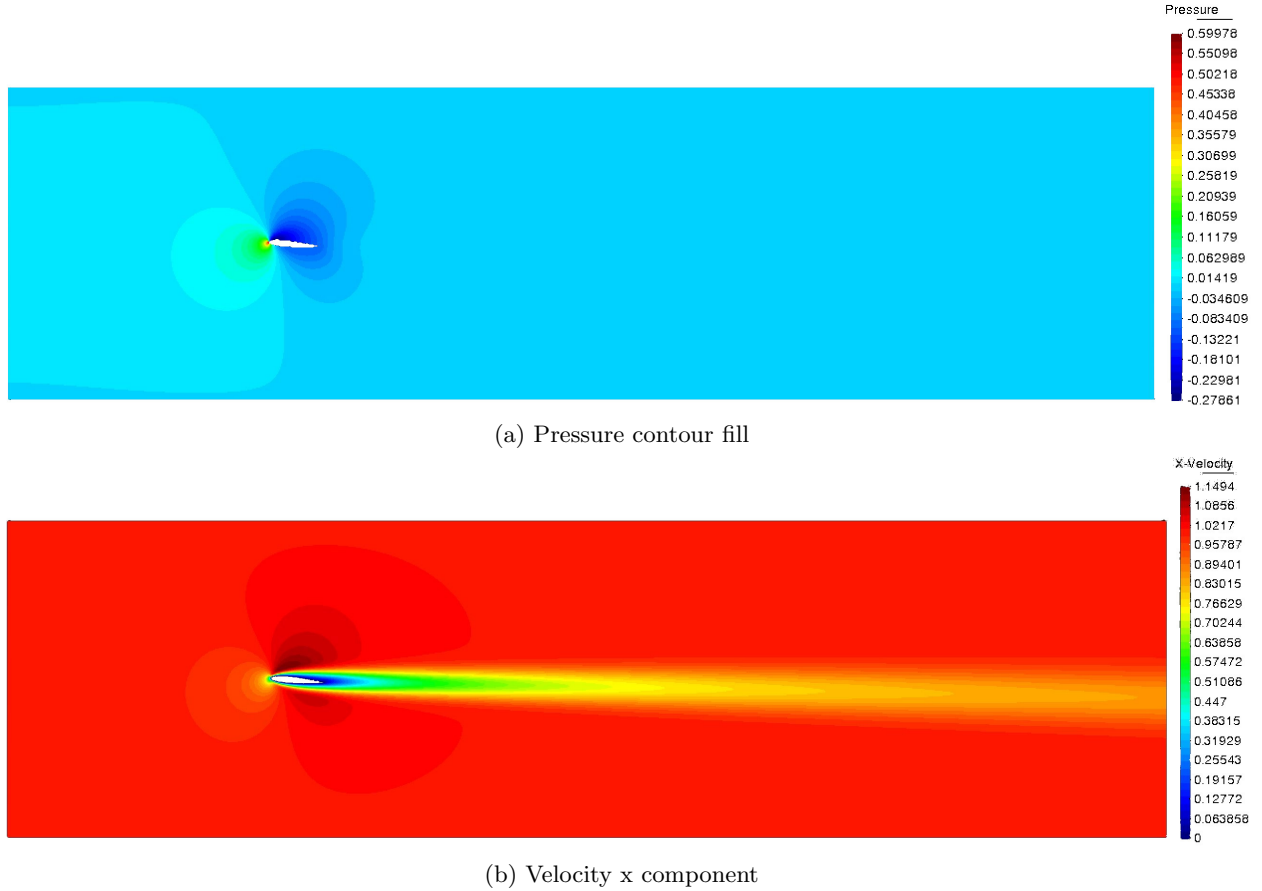
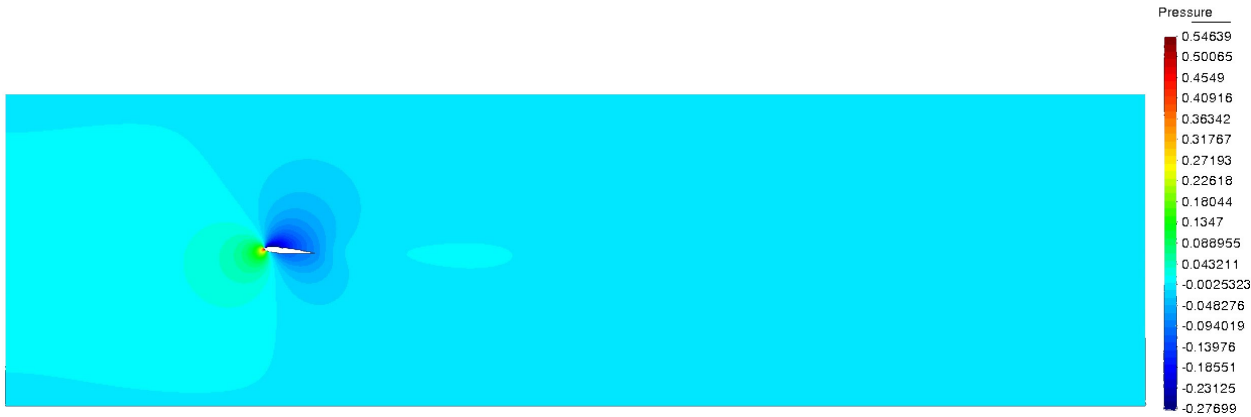
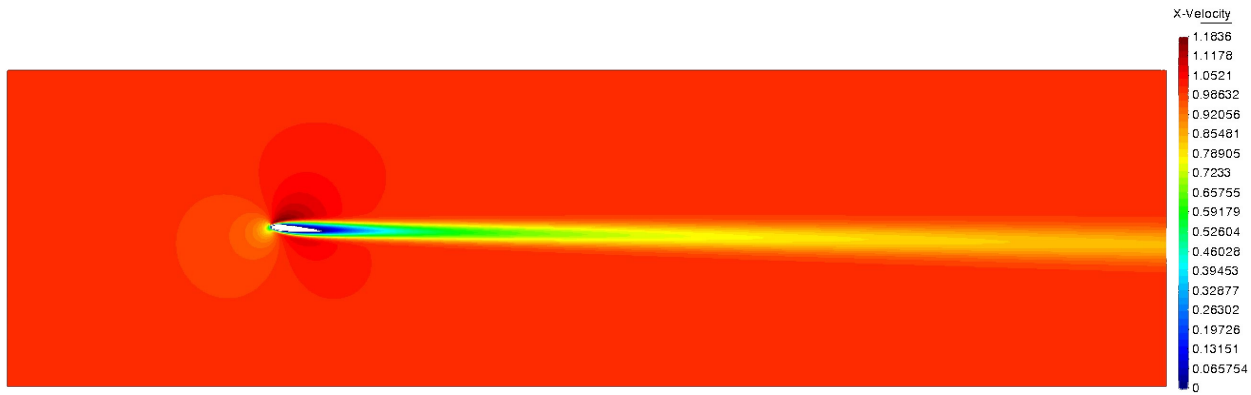


Figure 5.22: Pressure and velocity plots for a Reynolds number of 200.

Reynolds 500



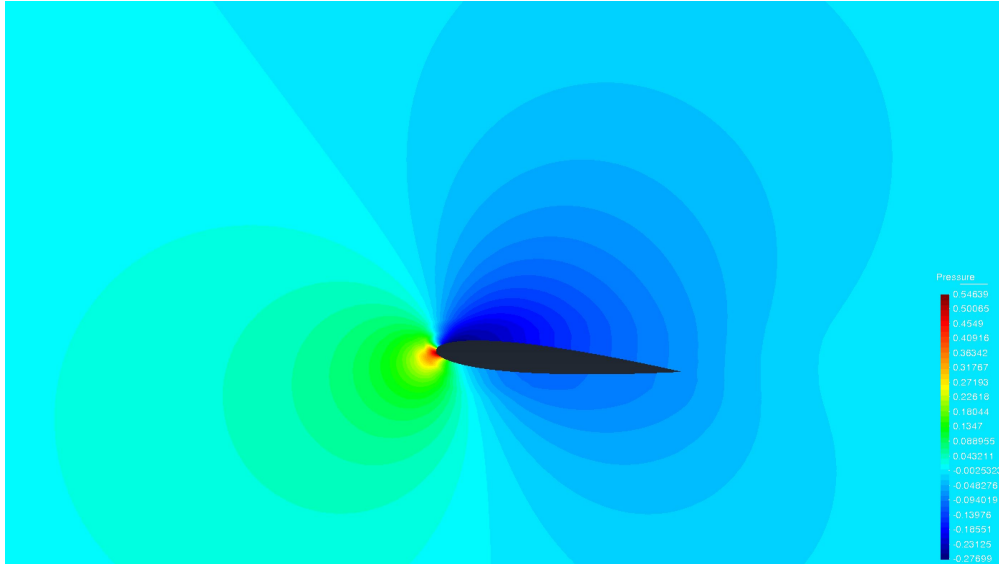
(a) Pressure contour fill



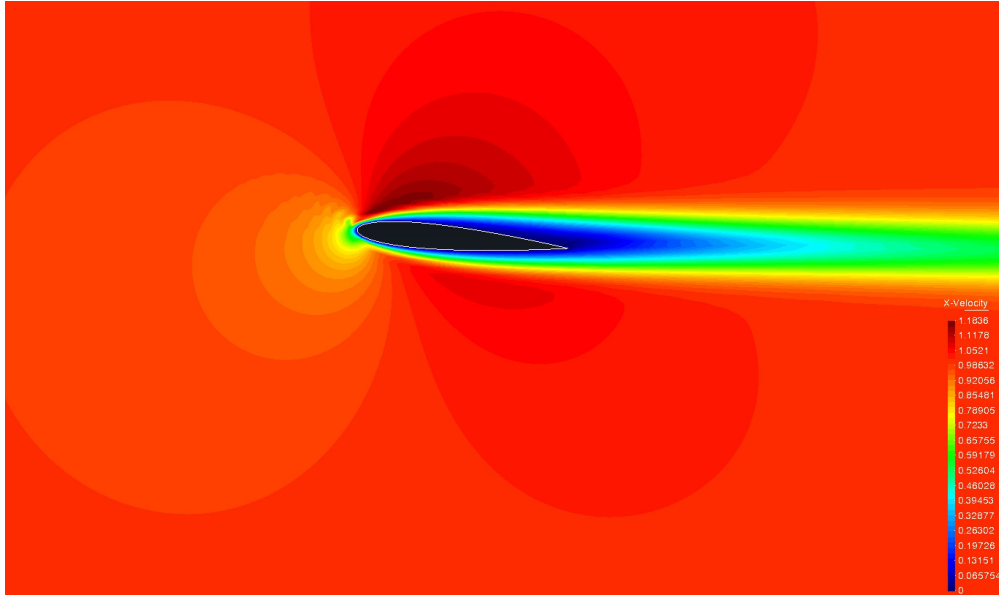
(b) Velocity x component

Figure 5.23: Pressure and velocity plots for a Reynolds number of 500.

Zooming in on the airfoil:



(a) Pressure contour fill



(b) Velocity x component

Figure 5.24: Pressure and velocity plots zoomed in on the airfoil.

Note that some singularities appear on the leading edge of the airfoil. This is due to how the mesh is done as it has high skewness and aspect ratio because of the curvature of the airfoil.

In both of these simulations a plausible solution is obtained. Now CFD studies of different airfoils could be made to maximize efficiency and study aerodynamics.

### 5.3 Reduced order modeling results

In this section, the results obtained for chapter 4 will be presented. Basis functions  $\Phi$  will be visualized. Afterwards, the principal angles between different simulations will be studied to know if there is potential for compressing the data and reducing computational costs. The problems will be solved as described in

4.3, with discrete steps and solving the system at each step. It is important to mention that by varying the number of steps the span of  $\Phi$  varies. Such effect will also be discussed. Finally, note that the full reduced model is not present as the time spent coding Navier-Stokes left no more time for the machine learning part. Nonetheless, with the matrix basis  $\Phi$  it should be straightforward.

### 5.3.1 Lid driven cavity

For the case described in 5.1.1, some of the modes, where a mode is a column of  $\Phi$ , obtained for a Reynolds number of 1000 with 20 steps are:

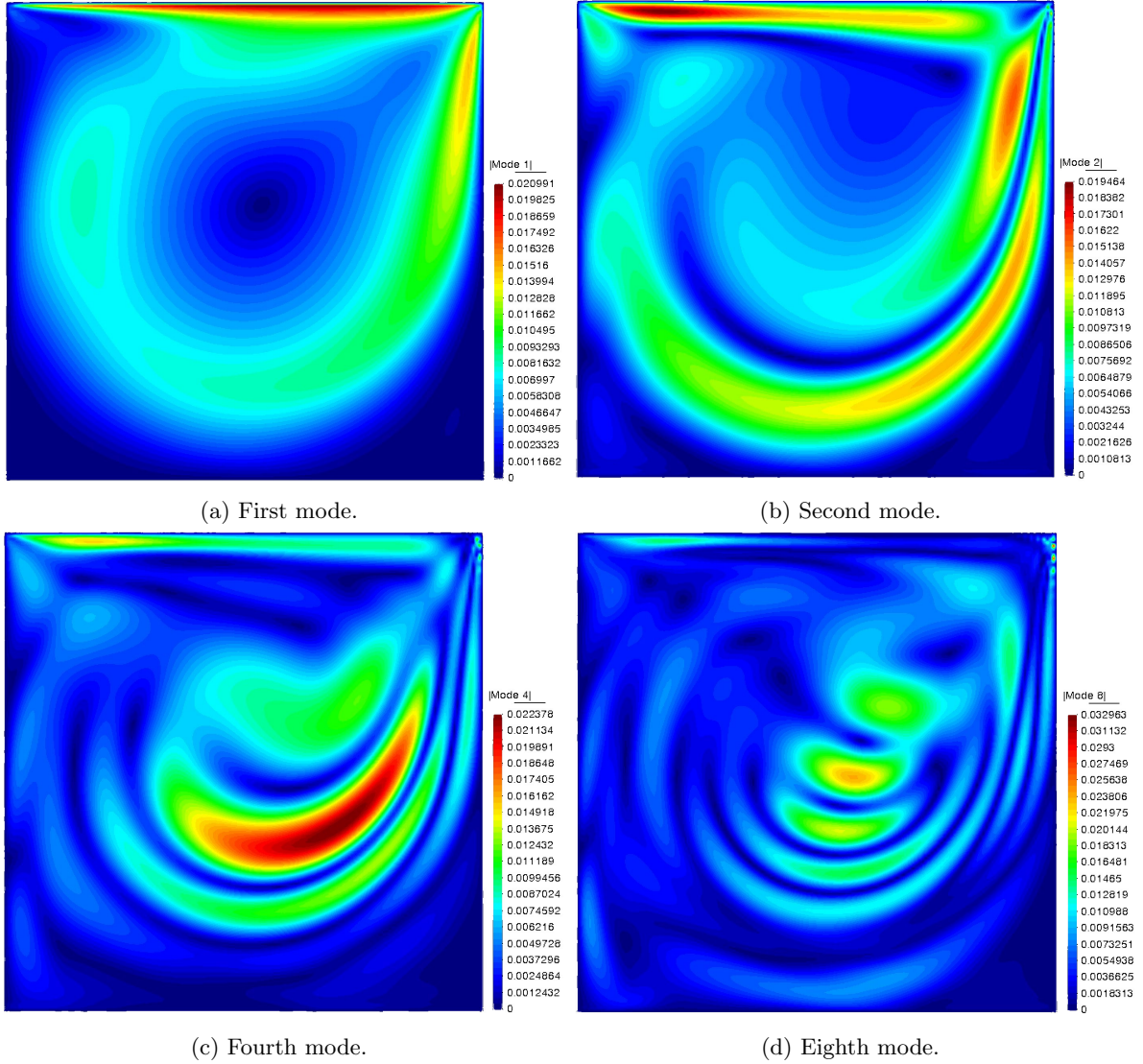


Figure 5.25: Different modes of the velocity modulus for a Reynolds number of 1000.

Although these modes do not have an intrinsic physical meaning, the patterns that arise in this case look like vibrations in a membrane.

The relevance of each mode decreases exponentially, as can be seen in the following plot:

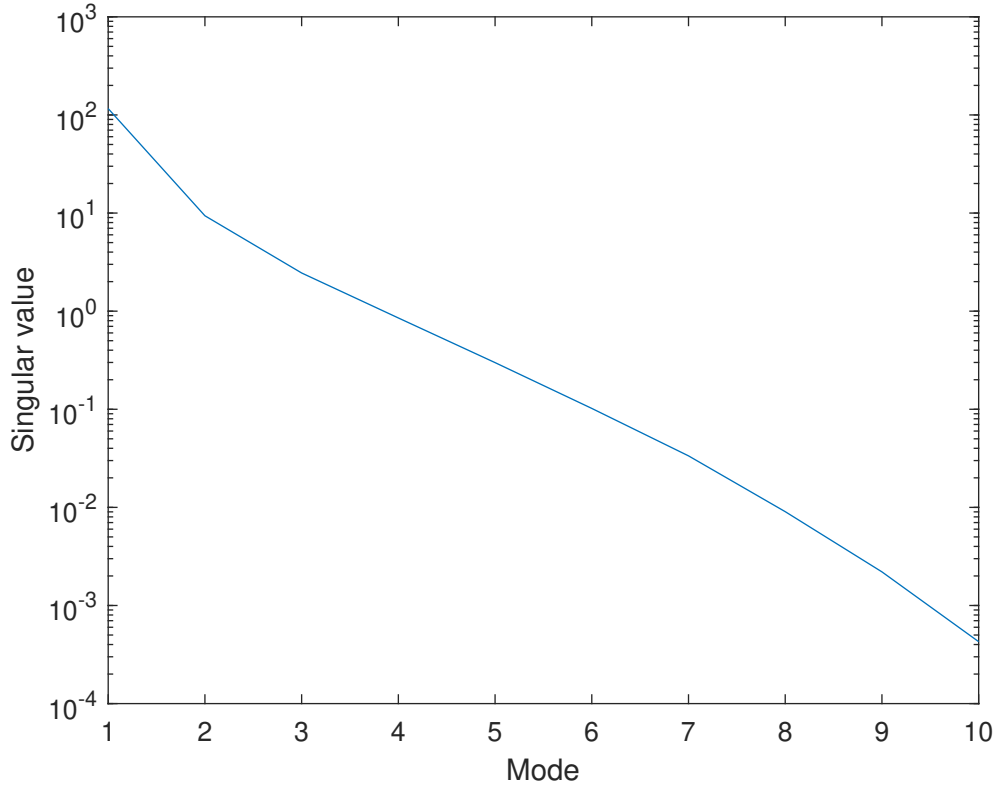


Figure 5.26: Plot of the singular value for each mode.

It is easy to see that the importance of the last values is negligible, so the basis matrix could be truncated and be a good approximation.

### Principal angles analysis

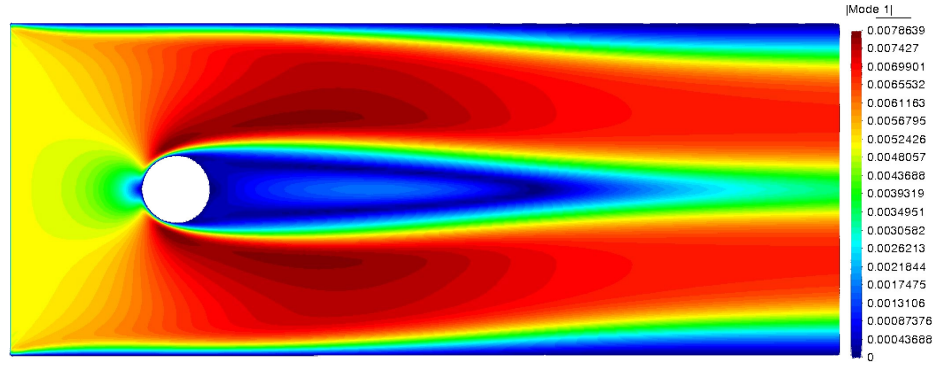
Between all the Reynolds numbers (100, 400 and 1000) and the number of steps used to perform the simulation (10 and 20), there are  $\binom{6}{2} = 15$  combinations possible. All the combinations and their respective angles can be found in appendix E.1. The most notable results are:

- Between two Reynolds numbers, varying the number of steps does not alter the subspace in a significant manner.
- The closer the Reynolds number, the more similar are the angles.

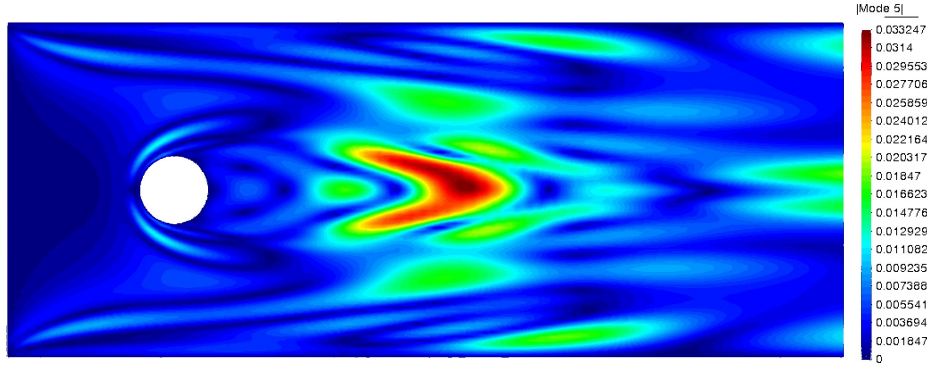
### 5.3.2 Flow around a cylinder

In the case of flow around a cylinder, see 5.1.2, the Reynolds number chosen to represent the modes is 200, and the number of steps to perform the simulation is 20.

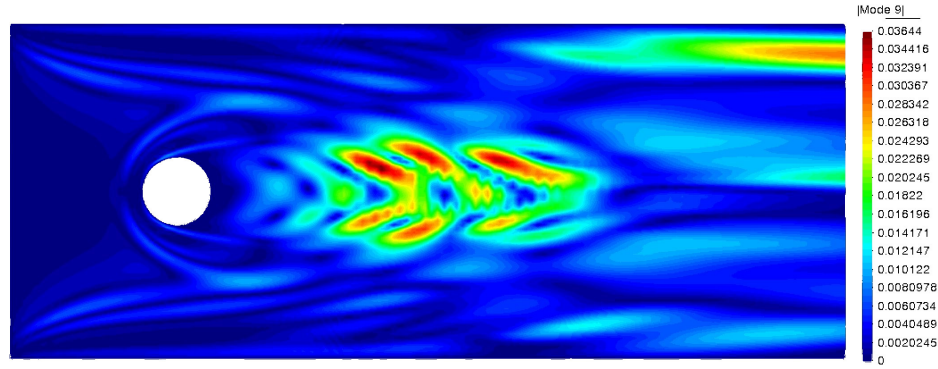




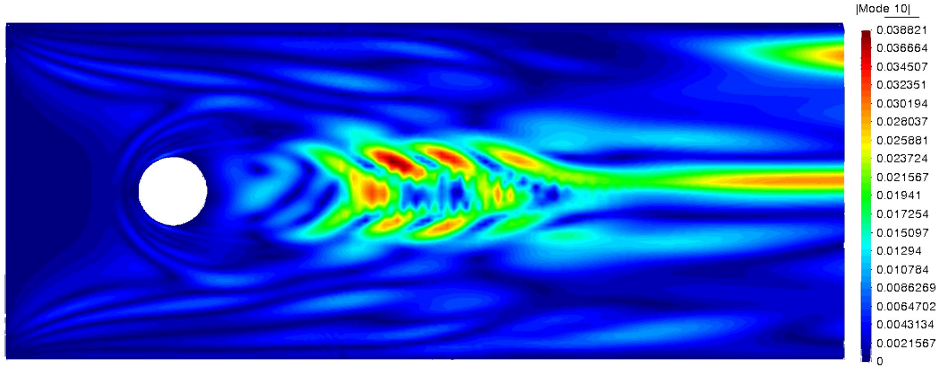
(a) First mode.



(b) Fifth mode.



(c) Ninth mode.



(d) Tenth mode.

Figure 5.27: Different modes of the velocity modulus for a Reynolds number of 200.

Note that on the ninth and tenth modes there is an asymmetry along the horizontal centerline. When viewing

the complete flow, figure 5.19a, no asymmetry can be observed, at least visually. The explanation for the asymmetry might be that this is the beginning of vortex shedding and, if the simulation were transient, vortices would start forming.

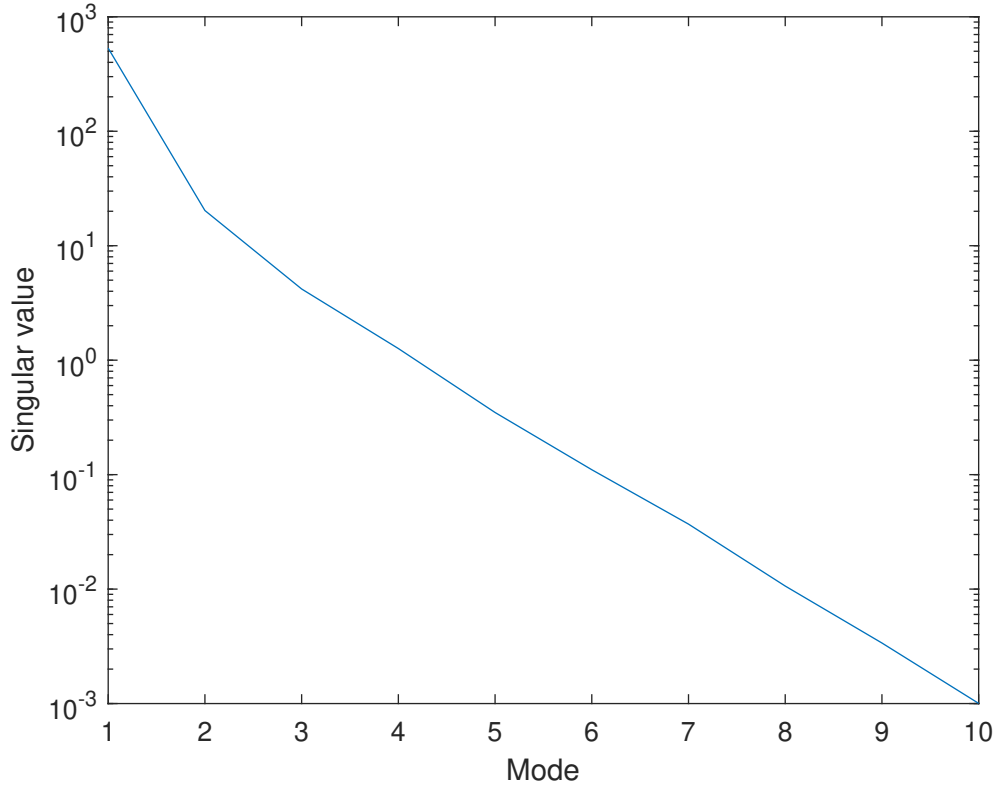


Figure 5.28: Plot of the singular value for each mode for the cylinder case.

The singular values decay even faster than on the lid driven case indicating that the last modes can almost be completely ignored.

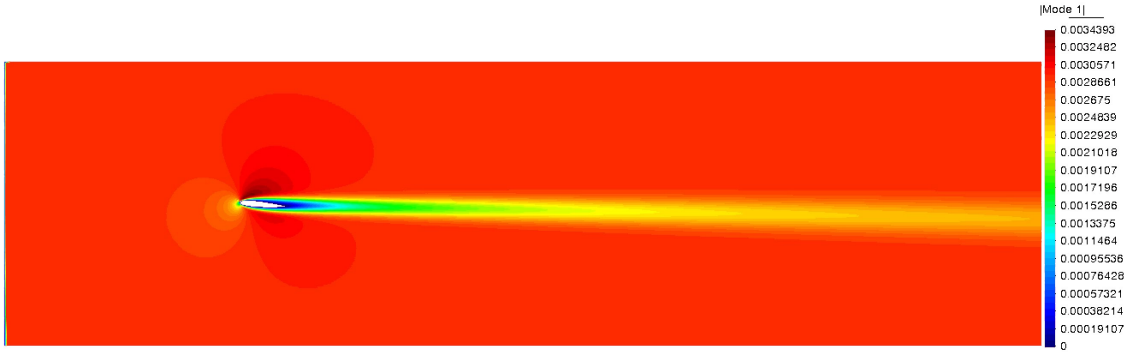
#### Principal angles analysis

Once again, the values for the principal angles can be found in appendix E.2. In this case the results found are identical to those for the lid driven cavity.

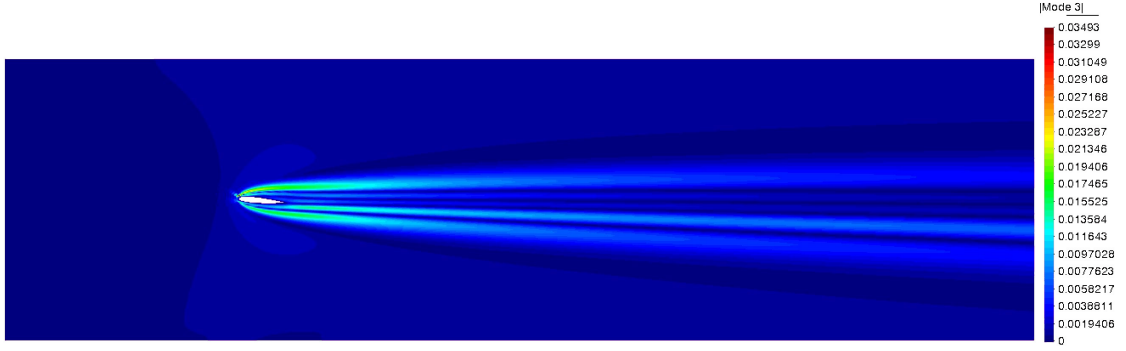
#### 5.3.3 NACA 0012

Finally, the flow around the NACA 0012 airfoil is studied. A Reynolds of 500 and 20 steps have been chosen for this results.

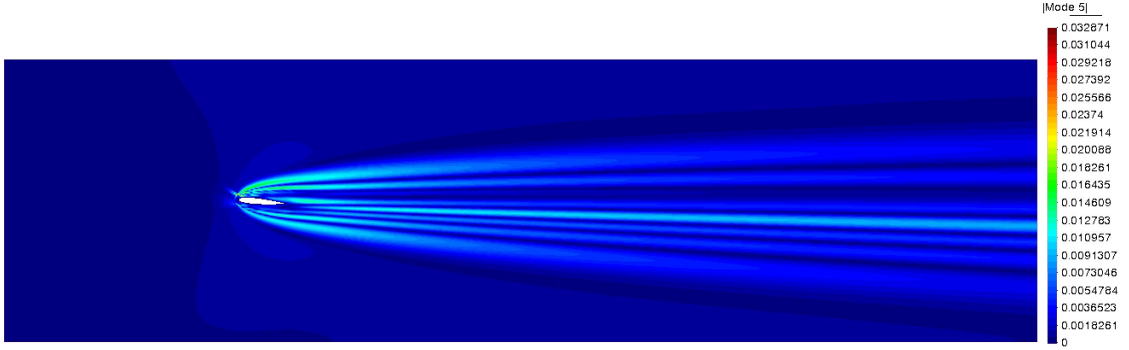




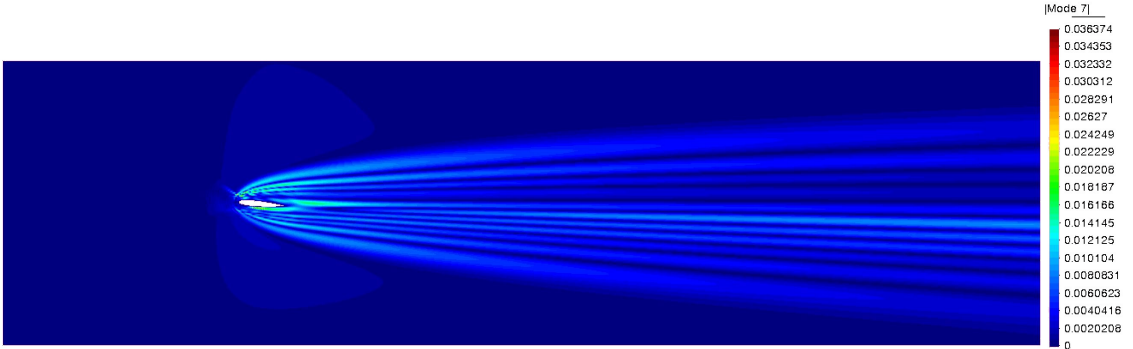
(a) First mode.



(b) Third mode.



(c) Fifth mode.



(d) Seventh mode.

Figure 5.29: Different modes of the velocity modulus for a Reynolds number of 500.

The singular values are:

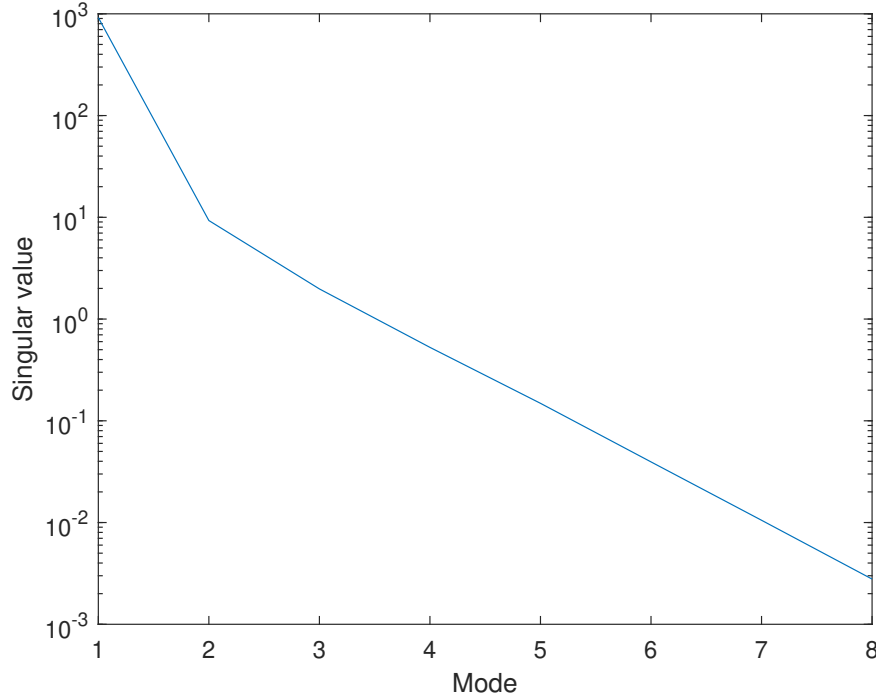


Figure 5.30: Plot of the singular value for each mode.

This particular case does not provide more insight, the results are what would be expected after visualizing the lid driven cavity and cylinder.

### Principal angles analysis

The principal angles can be found in appendix E.3. The results show that more steps provide the same basis, as the other cases, but it also shows something new. In this particular case, as the Reynolds numbers are closer together, performing the simulation with  $Re=200$  or  $Re=500$  give closer angles than when the Reynolds are varied in the lid driven case or the cylinder. It can be explained as the only modification between two Reynolds is in the wake of the airfoil leaving a bigger part of the computational domain the same. This might lead to closer bases.

Finally, regarding the potential for compressing data, it seems that there is potential for using the same basis for different Reynolds numbers, as there are a few angles that are 0. Nonetheless information and precision will be lost, as no 2 different simulations are perfectly contained within each other.

## Chapter 6

# Environmental impact

The environmental impact analysis aims to provide insight into how this project affects the environment and to provide possible solutions if deemed necessary.

First of all, the project only has one major aspect affecting the environment, that is energy consumption. As CFD simulations take a lot of computing power, the electricity consumption is huge, going as far as some Mega Watts in some super computers [20]. Thus the consumption of the computer used to make this project will be studied. On average, the consumption is of 200W (including screens). The total CO<sub>2</sub> emissions will be:

Table 6.1: Total CO<sub>2</sub> emissions in kg.

Time [h]	Power [kW]	Total consumption [kWh]	Impact Factor [21] [kg CO <sub>2</sub> /kWh]	CO <sub>2</sub> [kg]
250	0.2	50	0.259	13

To reduce the emissions renewable energy must be used. Then, other aspects like the materials used to build the computer could be looked at, but currently it is a negligible problem compared to the CO<sub>2</sub> emissions.

## Chapter 7

# Conclusions

In conclusion, the implementation of the finite element method for the Navier-Stokes equations has been a success and it is able to solve the governing equations of a newtonian incompressible fluid. It is able to capture the essential features of the flow field, such as velocity and pressure distributions. Moreover, the code has been successfully accelerated via the implementation of vectorization providing several orders of magnitude of speed-up. Afterwards, three different cases have been studied, both for Stokes flow and for Navier-Stokes flow, all with excellent results. The project proves that the finite element method is a suitable choice for performing CFD simulations, but the LBB condition must be circumvented or the meshing process becomes too tedious.

Furthermore, the ground work for increasing the computational efficiency with a reduced order model has been laid out, and the comparison between principal angles has provided us with useful information.

Analyzing the objectives of this project it can be seen that the first one is completed, as it solves the NS equations. However the second has not been met completely, which is: *Accelerate the code by statistical techniques*. This is because most of the time has been spent implementing the Navier-Stokes equations which originally was not planned. So although not every requirement has been met, one of them has been far exceeded, as the code solves NS, not only Stokes.

In summary, this project has built a Finite Element solver for the Navier-Stokes equations, and laid the foundation for implementing a reduced order model. In the future a transient solver can be implemented and there is much to do on the machine learning part, such as the implementation of the proper orthogonal decomposition.

# Bibliography

1. FOUNDATION, Poetry. *High flight* [<https://www.poetryfoundation.org/poems/157986/high-flight-627d3cfb1e9b7>]. 2023. [Online; accessed 11-June-2023].
2. CODINA, Ramon; BADIA, Santiago; BAIGES, Joan; PRINCIPE, Javier. Variational multiscale methods in computational fluid dynamics. *Encyclopedia of computational mechanics*. 2018, pp. 1–28.
3. CODINA, Ramon. Stabilized finite element approximation of transient incompressible flows using orthogonal subscales. *Computer methods in applied mechanics and engineering*. 2002, vol. 191, no. 39-40, pp. 4295–4321.
4. HUGHES, Thomas JR; FEIJÓO, Gonzalo R; MAZZEI, Luca; QUINCY, Jean-Baptiste. The variational multiscale method—a paradigm for computational mechanics. *Computer methods in applied mechanics and engineering*. 1998, vol. 166, no. 1-2, pp. 3–24.
5. DADVAND, Pooyan; ROSSI, Riccardo; OÑATE, Eugenio. An object-oriented environment for developing finite element codes for multi-disciplinary applications. *Archives of computational methods in engineering*. 2010, vol. 17, pp. 253–297.
6. DADVAND, Pooyan; ROSSI, Riccardo; GIL, Marisa; MARTORELL, Xavier; COTELA, Jordi; JUANPERE, Edgar; IDELSOHN, Sergio R; OÑATE, Eugenio. Migration of a generic multi-physics framework to HPC environments. *Computers & Fluids*. 2013, vol. 80, pp. 301–309.
7. FERRÁNDIZ, Vicente Mataix; BUCHER, Philipp; ZORRILLA, Rubén; ROSSI, Riccardo; CORNEJO, Alejandro; JCOTELA; CELIGUETA, Miguel Angel; MARIA, Josep; TTESCHEMACHER; ROIG, Carlos; MASÓ, Miguel; WARNAKULASURIYA, Suneth; CASAS, Guillermo; NÚÑEZ, Marc; DADVAND, Pooyan; LATORRE, Salva; POUPLANA, Ignasi de; GONZÁLEZ, Joaquín Irazábal; ARRUFAT, Ferran; RICCARDOTOSI; AFRANCI; GHANTASALA, Aditya; WILSON, Peter; DBAUMGAERTNER; CHANDRA, Bodhinanda; GEISER, Armin; SAUTTER, Klaus Bernd; LOPEZ, Inigo; LLUÍS; GÁRATE, Javi. KratosMultiphysics/Kratos: Release 9.3. 2023. Available from DOI: [10.5281/zenodo.7681287](https://doi.org/10.5281/zenodo.7681287).
8. BENNER, Peter; GUGERCIN, Serkan; WILLCOX, Karen. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM review*. 2015, vol. 57, no. 4, pp. 483–531.
9. LIEU, Thuan; FARHAT, Charbel; LESOINNE, Michel. Reduced-order fluid/structure modeling of a complete aircraft configuration. *Computer methods in applied mechanics and engineering*. 2006, vol. 195, no. 41-43, pp. 5730–5742.
10. RIZZI, Arthur; LUCKRING, James M. Historical development and use of CFD for separated flow simulations relevant to military aircraft. *Aerospace Science and Technology*. 2021, vol. 117, p. 106940.

11. DONEA, Jean; HUERTA, Antonio. *Finite element methods for flow problems*. John Wiley & Sons, 2003.
12. WIKIPEDIA CONTRIBUTORS. *Navier–Stokes equations* — *Wikipedia, The Free Encyclopedia* [[https://en.wikipedia.org/w/index.php?title=Navier%E2%80%93Stokes\\_equations&oldid=1149572892](https://en.wikipedia.org/w/index.php?title=Navier%E2%80%93Stokes_equations&oldid=1149572892)]. 2023. [Online; accessed 7-May-2023].
13. ORTEGA, Joaquín Hernández. *Stokes Flow*. 2023. Technical report.
14. ORTEGA, Joaquín Hernández. *Finite element method*. 2023. Technical report.
15. WIKIPEDIA CONTRIBUTORS. *Banach fixed-point theorem* — *Wikipedia, The Free Encyclopedia* [[https://en.wikipedia.org/w/index.php?title=Banach\\_fixed-point\\_theorem&oldid=1147555595](https://en.wikipedia.org/w/index.php?title=Banach_fixed-point_theorem&oldid=1147555595)]. 2023. [Online; accessed 28-May-2023].
16. CANTE, Juan Carlos. *Structural theory*. 2022. Technical report.
17. BRUNTON, Steven L; KUTZ, J Nathan. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.
18. ORTEGA, Joaquín Hernández. *Geometrically nonlinear elastostatics*. 2023. Technical report.
19. GOLUB, Gene H; VAN LOAN, Charles F. *Matrix computations*. JHU press, 2013.
20. STROHMAIER, Erich; DONGARRA, Jack; SIMON, Horst; MEUER, Martin. *Top 500* [<https://www.top500.org/>]. 2023. [Online; accessed 11-June-2023].
21. GENCAT. *Factor de emisión de la energía eléctrica: el mix eléctrico. Cambio climático* [[https://canviclimatic.gencat.cat/es/actua/factors\\_demissio\\_associats\\_a\\_lenergia/](https://canviclimatic.gencat.cat/es/actua/factors_demissio_associats_a_lenergia/)]. 2023. [Online; accessed 11-June-2023].

## Appendix A

# Profiling results

The profiled results as mentioned in [3.4](#) for the code's *main* are:

Profile Summary (Total time: 96.035 s)

Generated 06-jun-2023 17:23:35 using performance time.

Function Name	Calls	Total Time (s) ↓	Self Time* (s)	Total Time Plot (dark band = self time)
<a href="#">mainFLUID</a>	1	96.032	0.165	<div></div>
<a href="#">SolverNavierStokes</a>	1	93.839	85.365	<div></div>
<a href="#">assemblyC</a>	97	8.388	3.645	<div></div>
<a href="#">AssemblyUGlobal</a>	97	4.725	4.725	<div></div>
<a href="#">ComputeKGF</a>	1	0.708	0.090	<div></div>
<a href="#">ReadMeshFileStr_MULT</a>	2	0.657	0.049	<div></div>
<a href="#">ReadInputDataFile_v</a>	1	0.548	0.030	<div></div>
<a href="#">ReadUntCount</a>	6	0.359	0.359	
<a href="#">addpath</a>	6	0.322	0.007	
<a href="#">path</a>	6	0.313	0.179	
<a href="#">ComputeBelemALL</a>	1	0.266	0.210	
<a href="#">AssemblyMethodBCB</a>	1	0.263	0.065	
<a href="#">ReadInputDataFile_p</a>	1	0.240	0.005	
<a href="#">leer_fichero_colum</a>	6	0.214	0.214	
<a href="#">AssemblyBGlobal</a>	1	0.139	0.139	
<a href="#">GidPostProcess2DV</a>	1	0.121	0.003	
<a href="#">GidMesh2DFE</a>	2	0.108	0.107	
<a href="#">...l.mvm.eventmgr.MVMEvent.invokeListener(listener,eventTags,details)</a>	2	0.080	0.001	
<a href="#">MVMEvent&gt;MVMEvent.invokeListener</a>	2	0.080	0.004	
<a href="#">...spaceListener&gt;WorkspaceListener.workspaceUpdatedCorrectContext</a>	2	0.074	0.005	
<a href="#">...b.datatoolsservices.WorkspaceEventType.WORKSPACE_CLEARED)</a>	1	0.066	0.000	
<a href="#">WorkspaceListener&gt;WorkspaceListener.executeWorkspaceListeners</a>	1	0.061	0.005	
<a href="#">AssemblyNGlobalV</a>	1	0.058	0.058	
<a href="#">ListOfNodesLINE</a>	5	0.053	0.003	
<a href="#">NodesFacesLinesGID</a>	5	0.049	0.043	
<a href="#">general\private\parsedirs</a>	12	0.048	0.048	
<a href="#">FdisCOMP</a>	1	0.046	0.020	
<a href="#">GidPostProcess2DP</a>	1	0.043	0.002	
<a href="#">GidResults2DFEV</a>	1	0.041	0.040	
<a href="#">MLWorkspaceDataModel&gt;MLWorkspaceDataModel.workspaceUpdated</a>	1	0.040	0.005	
<a href="#">ConvertBlockDiag</a>	2	0.038	0.004	
<a href="#">AssemblyNIB</a>	1	0.036	0.027	
<a href="#">num2str</a>	172	0.033	0.011	
<a href="#">ReadNodesMsh</a>	1	0.031	0.030	
<a href="#">ComputeElementShapeFun</a>	6	0.026	0.010	
<a href="#">ComputeNelemALLV</a>	1	0.023	0.015	
<a href="#">QtransfBvect</a>	18	0.023	0.023	
<a href="#">ConvertCmatSparseMatrix</a>	2	0.022	0.022	
<a href="#">num2str&gt;handleNumericPrecision</a>	127	0.019	0.002	
<a href="#">FdisElem</a>	223	0.018	0.009	
<a href="#">genpath</a>	20	0.018	0.013	
<a href="#">num2str&gt;convertUsingRecycledSprintf</a>	127	0.018	0.018	
<a href="#">unique</a>	18	0.018	0.009	
<a href="#">DefineElastMatGLO</a>	1	0.017	0.015	
<a href="#">string.strcat</a>	1	0.016	0.007	



<a href="#">setdiff</a>	1	0.013	0.002	
<a href="#">IndicesCtang</a>	2	0.012	0.012	
<a href="#">RemoveREpeatedConnectivities</a>	2	0.012	0.006	
<a href="#">inverseTRANSvectorize</a>	9	0.012	0.012	
<a href="#">setdiff&gt;setdiffR2012a</a>	1	0.011	0.003	
<a href="#">MLArrayDataModel&gt;MLArrayDataModel.variableChanged</a>	1	0.011	0.003	
<a href="#">GidResults2DFEP</a>	1	0.010	0.009	
<a href="#">...edVariableObserver&gt;MLNamedVariableObserver.workspaceUpdated</a>	1	0.010	0.003	
<a href="#">ChangeCoordBnd</a>	669	0.009	0.009	
<a href="#">cell.strcat</a>	1	0.009	0.009	
<a href="#">unique&gt;uniqueR2012a</a>	18	0.009	0.009	
<a href="#">Quadrilateral9NInPoints</a>	3	0.009	0.005	
<a href="#">...atlab.datatoolsservices.WorkspaceEventType.VARIABLE_CHANGED)</a>	1	0.008	0.000	
<a href="#">determinantVECTORIZE</a>	9	0.008	0.008	
<a href="#">strcat</a>	8	0.008	0.008	
<a href="#">ObtInfMsh</a>	6	0.008	0.003	
<a href="#">ComputeNelemALL</a>	1	0.007	0.002	
<a href="#">ReadUntilToken</a>	16	0.007	0.004	
<a href="#">DetermineWeightsST</a>	9	0.007	0.007	
<a href="#">AssemblyNGlobalIP</a>	1	0.007	0.007	
<a href="#">cell.setdiff</a>	2	0.006	0.000	
<a href="#">cell2mat</a>	16	0.006	0.006	
<a href="#">WorkspaceListener&gt;WorkspaceListener.logEvent</a>	10	0.006	0.005	
<a href="#">cell.setdiff&gt;cellsetdiffR2012a</a>	2	0.006	0.002	
<a href="#">fullfile</a>	19	0.005	0.003	
<a href="#">Quadrilateral9NInPointsPressure</a>	2	0.005	0.003	
<a href="#">ElemBnd</a>	2	0.005	0.005	
<a href="#">strtok</a>	76	0.005	0.002	
<a href="#">cell.unique</a>	4	0.004	0.004	
<a href="#">WriteAuxFdNamesNEW</a>	2	0.004	0.001	
<a href="#">FindInArgOUT</a>	8	0.004	0.002	
<a href="#">close</a>	1	0.004	0.002	
<a href="#">strsplit</a>	1	0.004	0.002	
<a href="#">VarArginInput</a>	2	0.003	0.001	
<a href="#">COORailELEM</a>	1	0.003	0.003	
<a href="#">strtok&gt;doStrtok</a>	76	0.003	0.003	
<a href="#">int2str</a>	45	0.003	0.003	
<a href="#">MLArrayDataModel&gt;MLArrayDataModel.adapterChangeForSameClass</a>	1	0.003	0.001	
<a href="#">MLWorkspaceDataModel&gt;MLWorkspaceDataModel.updateData</a>	1	0.003	0.001	
<a href="#">WorkspaceListener&gt;WorkspaceListener.disableLXELListeners</a>	1	0.003	0.001	
<a href="#">str2num</a>	12	0.003	0.002	
<a href="#">ismember</a>	1	0.003	0.001	
<a href="#">...ariableObserver&gt;MLNamedVariableObserver.disableNaNInfBreakpoint</a>	2	0.003	0.003	
<a href="#">allchild</a>	1	0.003	0.002	
<a href="#">predictFromFeatureQueue</a>	1	0.003	0.001	
<a href="#">...^2).*(y-1/2).x.*(x+1).*(-y)).(1-x.^2).*(y+1/2).x.*(x-1).*(-y)).(1-x.^2).*(-2*y)]</a>	27	0.003	0.003	
<a href="#">split</a>	6	0.002	0.002	
<a href="#">MLWorkspaceDataModel&gt;MLWorkspaceDataModel.equalityCheck</a>	1	0.002	0.000	
<a href="#">FormatDataUtils&gt;FormatDataUtils.getCurrentNumericFormat</a>	1	0.002	0.001	
<a href="#">isprop</a>	1	0.002	0.002	

<a href="#">FndStrInCell</a>	8	0.002	0.002	
<a href="#">MLManager&gt;MLManager.getAdapterClassNameHelper</a>	3	0.002	0.002	
<a href="#">general\private\catdirs</a>	6	0.002	0.001	
<a href="#">ismember&gt;ismemberR2012a</a>	1	0.002	0.001	
<a href="#">join</a>	2	0.002	0.002	
<a href="#">kron</a>	1	0.002	0.002	
<a href="#">...+1).*(1-y.^2)/2.-(1-x.^2).*y.*(y+1)/2.x.*(x-1).*(1-y.^2)/2.-(1-x.^2).*(1-y.^2)]</a>	27	0.002	0.002	
<a href="#">Linear3NInPoints</a>	1	0.002	0.001	
<a href="#">WorkspaceListener&gt;WorkspaceListener.setupListeners</a>	2	0.001	0.001	
<a href="#">squeeze</a>	16	0.001	0.001	
<a href="#">str2num&gt;protected_conversion</a>	12	0.001	0.001	
<a href="#">...del&gt;@(es,ed)this.sendVariableEvent('VariablesChanged',ed.Variables)</a>	1	0.001	0.000	
<a href="#">fullfile&gt;ensureTrailingFilesep</a>	19	0.001	0.000	
<a href="#">setfield</a>	4	0.001	0.001	
<a href="#">...rkspaceViewModel&gt;RemoteWorkspaceViewModel.sendVariableEvent</a>	1	0.001	0.001	
<a href="#">WorkspaceListener&gt;@(varName)ismember(varName,varsInWS)</a>	1	0.001	0.000	
<a href="#">...ressure&gt;@(x,y)0.25*((1-x).*(1-y).*(1+x).*(1-y).*(1+x).*(1+y).*(1-x).*(1+y))</a>	18	0.001	0.001	
<a href="#">getBaseVariableName</a>	1	0.001	0.000	
<a href="#">strfun\private\isTextStrict</a>	8	0.001	0.001	
<a href="#">mustBeTextScalar</a>	2	0.001	0.001	
<a href="#">MVMEEvent&gt;MVMEEvent.subsref</a>	3	0.001	0.001	
<a href="#">strfun\private\strescape</a>	2	0.001	0.001	
<a href="#">partialMatchString</a>	4	0.001	0.001	
<a href="#">ismember&gt;ismemberClassTypes</a>	1	0.001	0.001	
<a href="#">StructureDataModel&gt;StructureDataModel.getData</a>	6	0.001	0.001	
<a href="#">...tatoollsservices,WorkspaceListener.enableLXEListeners(currentState)</a>	1	0.001	0.000	
<a href="#">CloneableVariable&gt;CloneableVariable.getCloneData</a>	2	0.001	0.000	
<a href="#">settings</a>	1	0.001	0.000	
<a href="#">fullfile&gt;refinePath</a>	19	0.001	0.001	
<a href="#">WorkspaceListenerList&gt;WorkspaceListenerList.getListener</a>	2	0.001	0.000	
<a href="#">mustBeText</a>	1	0.001	0.000	
<a href="#">WorkspaceListener&gt;WorkspaceListener.logEventsEnabled</a>	10	0.001	0.001	
<a href="#">...Pressure&gt;@(x,y)0.25*[-(1-y).*(1-y).*(1+y).-(1+y).-(1-x).-(1+x).*(1+x).*(1-x)]</a>	18	0.001	0.001	
<a href="#">fullfile&gt;addTrailingFileSep</a>	19	0.001	0.001	
<a href="#">cell.ismember</a>	1	0.001	0.001	
<a href="#">cellstr</a>	1	0.001	0.001	
<a href="#">validators\private\istext</a>	1	0.001	0.000	
<a href="#">settings</a>	1	0.001	0.001	
<a href="#">MLManager&gt;MLManager.getVariableAdapterClassTypeHelper</a>	1	0.001	0.000	
<a href="#">pathsep</a>	38	0.001	0.001	
<a href="#">WorkspaceListener&gt;WorkspaceListener.enableLXEListeners</a>	1	0.001	0.000	
<a href="#">MLStructureDataModel&gt;MLStructureDataModel.getData</a>	2	0.000	0.000	
<a href="#">NamedVariable&gt;NamedVariable.get.Workspace</a>	2	0.000	0.000	
<a href="#">NamedVariable&gt;NamedVariable.getName</a>	3	0.000	0.000	
<a href="#">MVMEEvent&gt;MVMEEvent.updateExecutionDepth</a>	4	0.000	0.000	
<a href="#">WorkspaceListener&gt;WorkspaceListener.getWorkspaceListenersList</a>	3	0.000	0.000	
<a href="#">WorkspaceListenerList&gt;WorkspaceListenerList.getListenerListSize</a>	3	0.000	0.000	
<a href="#">getSettingsRoot</a>	2	0.000	0.000	
<a href="#">blanks</a>	8	0.000	0.000	
<a href="#">MVMEEvent&gt;MVMEEvent.MVMEEvent</a>	2	0.000	0.000	

<a href="#">...paceDataModel&gt;@()this.reEnableNanInfBreakpoint(naninfBreakpoint)</a>	1	0.000	0.000	
<a href="#">validators\private\isCharRowVector</a>	3	0.000	0.000	
<a href="#">MLWorkspaceDocument&gt;MLWorkspaceDocument.variableChanged</a>	1	0.000	0.000	
<a href="#">...riableObserver&gt;@()this.reEnableNanInfBreakpoint(naninfBreakpoint)</a>	1	0.000	0.000	
<a href="#">..._internal.mvm.eventmgr.MVMEvent.updateExecutionDepth(listener,-1)</a>	2	0.000	0.000	
<a href="#">path&gt;isValidInput</a>	12	0.000	0.000	
<a href="#">...iableObserver&gt;MLNamedVariableObserver.reEnableNanInfBreakpoint</a>	3	0.000	0.000	
<a href="#">ismethod</a>	1	0.000	0.000	
<a href="#">StructureDataModel&gt;StructureDataModel.getData_I</a>	6	0.000	0.000	
<a href="#">Linear3N</a>	3	0.000	0.000	
<a href="#">...paceListener&gt;WorkspaceListener.getSetListenerExecutionInProgress</a>	2	0.000	0.000	
<a href="#">allchild&gt;@()set(rootobj,'ShowHiddenHandles',Temp)</a>	1	0.000	0.000	
<a href="#">...medVariableObserver&gt;MLNamedVariableObserver.getIgnoreUpdates</a>	2	0.000	0.000	
<a href="#">allchild&gt;getchildren</a>	1	0.000	0.000	
<a href="#">WorkspaceListener&gt;WorkspaceListener.getWorkspaceListenerEnabled</a>	2	0.000	0.000	
<a href="#">FormatDataUtils&gt;FormatDataUtils.checkIsString</a>	1	0.000	0.000	
<a href="#">MLNamedVariableObserver&gt;@()warning(w)</a>	2	0.000	0.000	
<a href="#">...ventData&gt;WorkspaceChangeEventData.WorkspaceChangeEventData</a>	1	0.000	0.000	
<a href="#">uitools\private\allchildRootHelper</a>	1	0.000	0.000	
<a href="#">ArrayDataModel&gt;ArrayDataModel.getClassType</a>	1	0.000	0.000	
<a href="#">WorkspaceListenerList&gt;WorkspaceListenerList.getDisabledListenerList</a>	1	0.000	0.000	
<a href="#">MLStructureAdapter&gt;MLStructureAdapter.getClassType</a>	1	0.000	0.000	
<a href="#">Document&gt;Document.getDataModel</a>	1	0.000	0.000	
<a href="#">...orkspaceListenerList&gt;WorkspaceListenerList.resetDisabledListenerList</a>	1	0.000	0.000	

\*Self time is the time spent in a function excluding any time spent in child functions. The time includes any overhead time resulting from the profiling process.

As for the *SolverNavierStokes* function:

SolverNavierStokes (Calls: 1, Time: 93.839 s)

Generated 06-jun-2023 17:25:56 using performance time.  
Function in file D:\Ferran\AA Universitat\8e Quatrimestre\TFG\_NS\_FEM\Code\Solver\SolverNavierStokes.m  
[Copy to new window for comparing multiple runs](#)

Parents (calling functions)

Function Name	Function Type	Calls
<a href="#">mainFLUID</a>	Script	1

Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
<a href="#">68</a>	d = [K11+C11 G1T; G1 L]\[-K1r*dR_v-C1r*dR_v; -Gr*dR_v...	97	84.784	90.4%	<div></div>
<a href="#">64</a>	C=assemblyC(COOR_v,CN_v,u,TypeElement_v,Bst,OmegaGlo,...	97	8.487	9.0%	<div></div>
<a href="#">65</a>	C11=C(DOFl_v,DOFl_v);	97	0.373	0.4%	<div></div>
<a href="#">51</a>	Nst=AssemblyNGlobalV(Nelem,nstrain,nelem_v,nnodeE_v,n...	1	0.059	0.1%	
<a href="#">66</a>	C1r=C(DOFl_v,DOFr_v);	97	0.028	0.0%	
All other lines			0.107	0.1%	
Totals			93.839	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time (s)	% Time	Time Plot
<a href="#">assemblyC</a>	Function	97	8.388	8.9%	<div></div>
<a href="#">AssemblyNGlobalV</a>	Function	1	0.058	0.1%	
<a href="#">ComputeNelemALLV</a>	Function	1	0.023	0.0%	
<a href="#">num2str</a>	Function	42	0.005	0.0%	
Self time (built-ins, overhead, etc.)			85.365	91.0%	<div></div>
Totals			93.839	100%	

Code Analyzer results

No Code Analyzer messages.

Coverage results

[Show coverage for parent folder](#)

Total lines in function	98
Non-code lines (comments, blank lines)	45
Code lines (lines that can run)	53
Code lines that did run	52
Code lines that did not run	1
Coverage (did run/can run)	98.11 %

Function listing

Time	Calls	Line
		1 function [u,v,p] = SolverNavierStokes(COOR v,CN v,rnod v,dR v,COOR p,rnod p,dR p,K,G,res,TypeElement v,maxite
		2 %Inputs:
		3 % - COOR v: Matrix with the coordiantes of velocity nodes
		4 % - COOR p: Matrix with the coordiantes of pressure nodes
		5 % - rnod v: Vector containing the restricted nodes of the velocity
		6 % - rnod p: Vector containing the restricted nodes of the pressure
		7 % - dR v: Vector containing the prescribed velocities
		8 % - dR p: Vector containing the prescribed pressures
		9 % - K: Global viscosity matrix
		10 % - G: Global gradient operator
		11 % - res: tolerance of the residual for the nonlinear system of equations
		12 % - TypeElement v: Velocity type of element for computing C
		13 % - maxiter: maximum iterations before the nonlinear solver returns an
		14 % error
		15 % - rel factor: relaxation factor for the nonlinear solver
		16 % - Bst: B-stacked matrix for velocity
		17 % - OmegaGlo: Matrix containing the products of all Jacobians and weights

```

18 % at all Gauss points
19 % - debug: Debug parameter to control the assembly of C
20 %Outputs:
21 % - u: vector with the horizontal component of the velocity
22 % - v: vector with the vertical component of the velocity
23 % - p: vector with the pressure
< 0.001 1 24 ndim = size(COOR v,2);
25
< 0.001 1 26 nDOF v = ndim * size(COOR v,1);
< 0.001 1 27 DOFl v = (1:nDOF v)';
< 0.001 1 28 DOFr v = rnod v;
< 0.001 1 29 DOFl v(DOFr v) = [];
30
31 % Pressure on node 1 set to zero
< 0.001 1 32 nDOF p = size(COOR p,1);
< 0.001 1 33 DOFl p = (1:nDOF p)';
< 0.001 1 34 DOFr p = rnod p;
< 0.001 1 35 DOFl p(DOFr p) = [];
36
37
38 % Decomposition of matrices
0.005 1 39 K11 = K(DOFl v,DOFl v);
0.002 1 40 G1 = G(DOFl p,DOFl v);
0.003 1 41 G1T = G1';
< 0.001 1 42 Gr = G(DOFl p,DOFr v);
< 0.001 1 43 K1r = K(DOFl v,DOFr v);
44 %Auxiliary matrix
< 0.001 1 45 L = zeros(length(DOFl p));
46
47
48 %Precalculation of matrix Nst for velocity
< 0.001 1 49 nnode v = size(COOR v,1); ndim = size(COOR v,2); nelemt v = size(CN v,1); nnodeE v = size(CN v,2) ; ngaus=
0.024 1 50 Nelem=ComputeNelemALLV(COOR v,CN v,TypeElement v,2);
0.059 1 51 Nst=AssemblyNGlobalV(Nelem,nstrain,nelemt v,nnodeE v,ndim,ngaus,CN v,nnode v);
52
53 %Solve of the nonlinear system
< 0.001 1 54 disp('Solving Navier-Stokes...')
55 % Allocate ones for first iteration
< 0.001 1 56 u=ones(nDOF v,1);
< 0.001 1 57 u(DOFr v)=dR v;
< 0.001 1 58 u new=zeros(nDOF v,1);
< 0.001 1 59 normResidual=1; % Initialization of the residual
< 0.001 1 60 iter=0;
< 0.001 1 61 tic;
< 0.001 1 62 while (res<normResidual && iter<maxiter)
63 %Assembly of matrix C, recall that C(u)
8.487 97 64 C=assemblyC(COOR v,CN v,u,TypeElement v,Bst,OmegaGlo,debug,Nst);
0.373 97 65 C11=C(DOFl v,DOFl v);
0.028 97 66 C1r=C(DOFl v,DOFr v);
67 %Solve the linear system
84.784 97 68 d = [K11+C11 G1T; G1 L]\[-K1r*dR v-C1r*dR v; -Gr*dR v];
69 %Assign the corresponding DOF to the velocity
0.022 97 70 u new(DOFl v)=d(1:length(DOFl v));
< 0.001 97 71 u new(DOFr v)=dR v;
72 %Calculate the residual
0.010 97 73 normResidual=max(norm(u new-u));
74 %Update the velocity with a relaxation factor
0.004 97 75 u=(1-rel factor)*u+rel factor*u new;
76
< 0.001 97 77 if(~mod(iter,5))
0.010 20 78 disp(['Iteration: ', num2str(iter), ' normResidual: ',num2str(normResidual)]);
< 0.001 97 79 end
< 0.001 97 80 iter=iter+1;
< 0.001 97 81 end
< 0.001 1 82 if(maxiter==iter)
83 error('Failed to converge');
< 0.001 1 84 end
85
< 0.001 1 86 tic;
< 0.001 1 87 disp(['Time to solve: ',num2str(tic),'s With ', num2str(iter),' iterations'])
88 % Decomposition of the different terms

```

```
< 0.001      1      89      sol(DOF1 v) = d(1:size(K11,1));
< 0.001      1      90      sol(DOFr v) = dR v;
              91
< 0.001      1      92      p1 = d(size(K11,1)+1:end);
< 0.001      1      93      p=zeros(size(COOR p,1),1);
< 0.001      1      94      p(DOF1 p)=p1;
< 0.001      1      95      p(DOFr p)=dR p;
< 0.001      1      96      u = sol(1:2:end-1);
< 0.001      1      97      v = sol(2:2:end);
0.021      1      98      end
```

---

## Appendix B

# Matlab code

The code developed for Matlab can be found on [https://github.com/Ferraan/TFG\\_NS\\_FEM](https://github.com/Ferraan/TFG_NS_FEM) There is a file called *Tutorial.mlx* that explains how to use it. The scripts used for postprocessing can also be found there, as well as some meshes ready to run.

The code was adapted from an elastostatics code for the subject *Enginyeria Aeroespacial Computacional*. Some functions are the same such as the naive assembly of  $\mathbf{K}$  or the vector  $\mathbf{F}$ , but the assembly of  $\mathbf{G}$  and  $\mathbf{C}$  had to be built from scratch. Moreover all the vectorization part is new, as well as the Picard iteration code. The preprocessing and postprocessing to and from GiD have been adapted from the subject's code. Finally, the building of snapshot matrix as well as finding the basis as well as the angles calculation are new.



## Appendix C

# Meshing process

To obtain the meshes the program GiD has been used. To generate a new velocity mesh the following steps must be followed:

1. Create the desired geometry and make a NURBS surface.

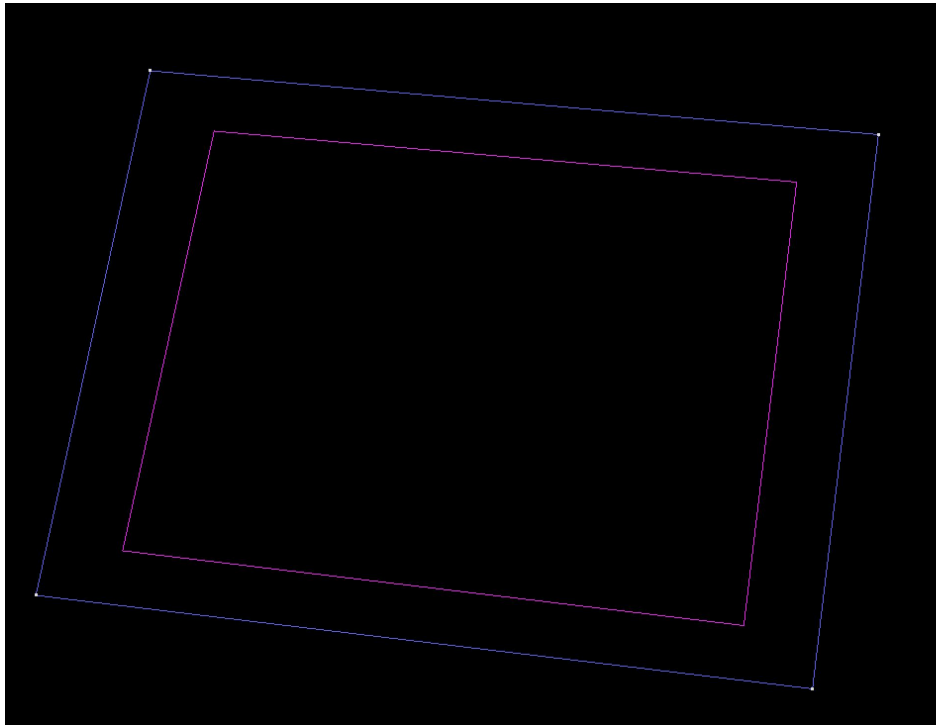


Figure C.1: Some random geometry with its NURBS surface

2. Load the problem type *PROBLEM\_TYPE\_simple.gid* <sup>1</sup>
3. Assign a material to all surfaces. The properties (viscosity, density) can be changed in the code.
4. Assign conditions to each line, go to Data – > condition – > line icon. Then in the Matlab code each

---

<sup>1</sup>Can be found in the GitHub repository found in section [B](#)

line will be assigned the Dirichlet or Von Neumann boundary conditions. For example for the lid driven case:

```

1 % 3. Dirichlet boundary conditions (prescribed velocity)
2 % -----
3 icond = 1; % Number of condition
4 DIRICHLET(icond).NUMBER_LINE = 1 ; % Number of line
5 DIRICHLET(icond).PRESCRIBED_Ux = 0 ; % (constant along the line)
6 DIRICHLET(icond).PRESCRIBED_Uy = 0 ;
7 icond = 2; % Number of condition
8 DIRICHLET(icond).NUMBER_LINE = 2 ;
9 DIRICHLET(icond).PRESCRIBED_Ux = Ux ;
10 DIRICHLET(icond).PRESCRIBED_Uy = 0 ;

```

5. Go to Mesh –> Element type –> Quadrilateral
6. Go to Mesh –> Quadratic type –> Quadratic9
7. Go to Mesh –> Structured –> Surfaces –> Assign number of divisions to surface lines
8. Go to Mesh –> Generate mesh

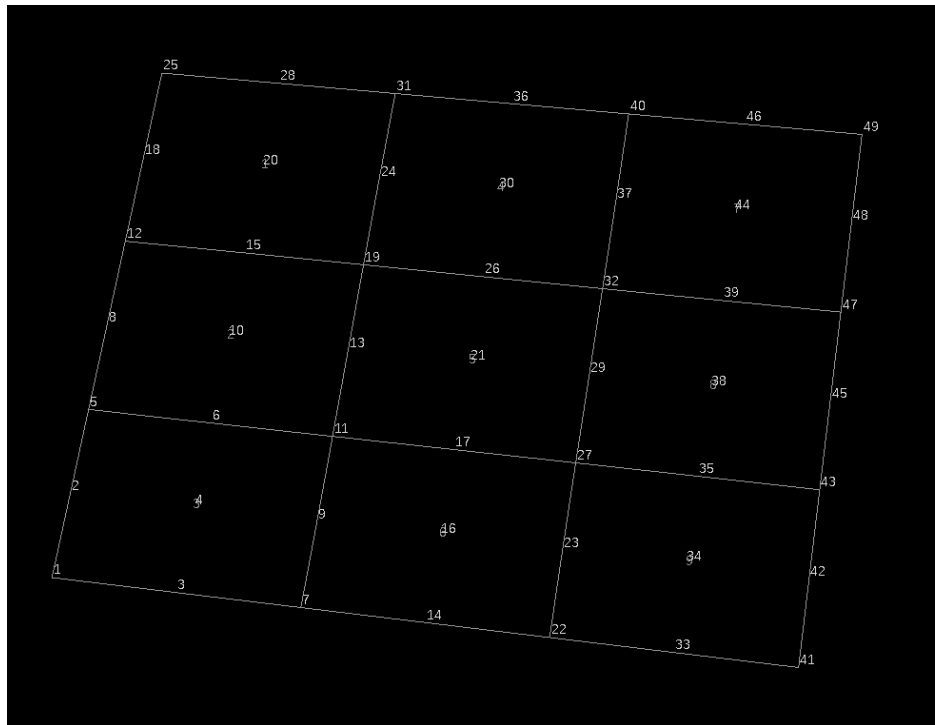


Figure C.2: Mesh in the geometry, note how there are 9 nodes per element as it is the velocity mesh.

9. Go to Mesh –> Create boundary mesh
10. Go to Files –> Export –> GiD Mesh
11. Go to Files –> Export –> Calculation files
12. Finally change the mesh input in the Matlab code.

```

1 NameMeshP='LidDriven75'; %Cylinder10,20,40,75 LidDriven75 %NACA0012_AoA.5

```

To build the mesh for pressure repeat the same steps but ignore line 6, by default GiD uses the correct elements for pressure which have four nodes in the vertices.

## Appendix D

# Kratos simulations

This appendix is a brief introduction on how to use Kratos for fluid problems. It assumes a python installation is already installed correctly on the computer.

1. Go to [Kratos Multiphysics](#) and follow the instructions for installing *Kratos*.
2. Go to GiD –> Data –> Problem type –> Internet retrieve and pick the latest version of *Kratos*.
3. Select *Kratos* as the problem type, then a pop up screen appears. In this screen choose fluid –> 2D.

The following should be seen:

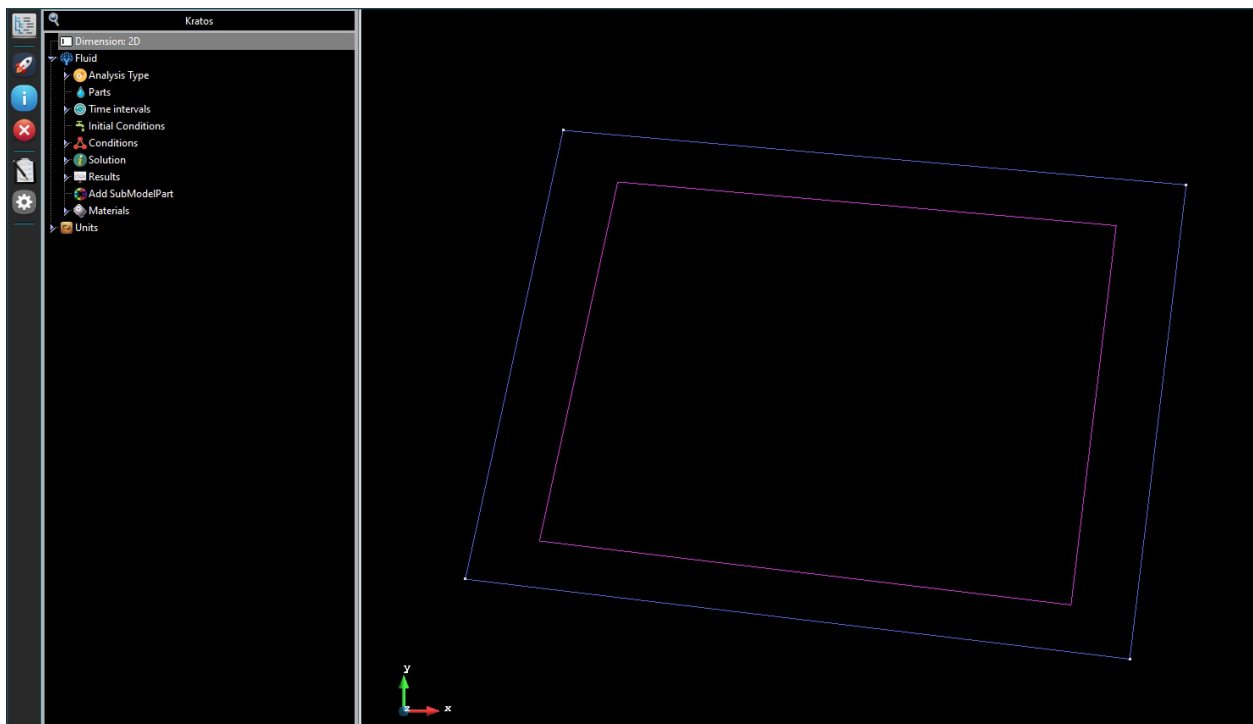


Figure D.1: GiD screen at the current step.

4. Now, an example with the mesh described in appendix C will be followed. Once the geometry is done

- go to Mesh – > Element type – > Triangle
5. Go to Mesh – > Quadratic type and make sure it is on normal.
  6. Go to Parts – > Group and assign a new group to the fluid.
  7. Go to Conditions and assign some boundary conditions. For this example the left and right walls will be set to a no slip condition, the lower wall will be set with an Automatic inlet velocity on the y axis of 1 and the upper wall will have an Outlet pressure condition of 0.
  8. Go to Time parameters and adjust the desired end time.
  9. Go to Mesh – > Generate mesh
  10. Click on Run the simulation
  11. When the simulation is done click on View results. The results for this case are:

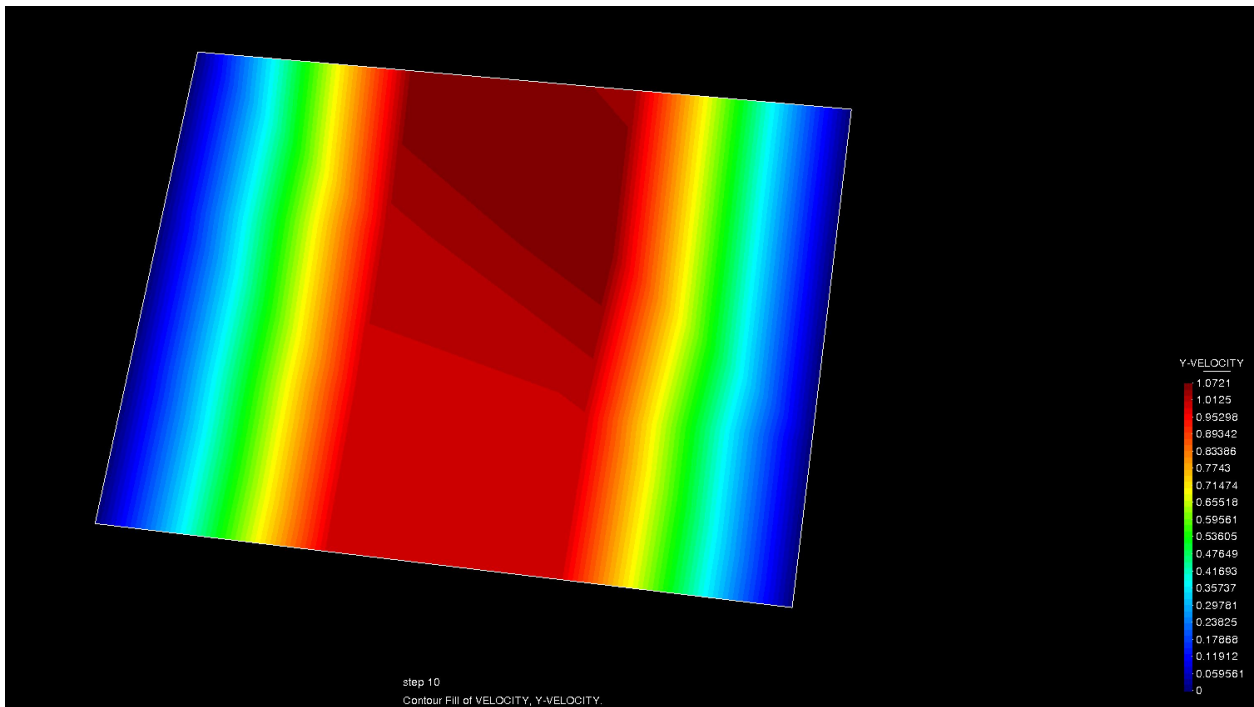


Figure D.2: GiD screen with the simulation output.

## Appendix E

# Principal angles combinations

All the combinations are generated automatically by Matlab, as well as these tables with the functions *PostProcSVDNameOfCase*.

In the following tables each column represents the 2 cases that have been compared, and the cosines of the angles can be seen.

### E.1 Lid driven cavity

Re=1000 $n_{stp}=20$	Re=1000 $n_{stp}=10$	Re=400 $n_{stp}=20$	Re=1000 $n_{stp}=10$
Re=1000 $n_{stp}=10$	Re=400 $n_{stp}=10$	Re=1000 $n_{stp}=10$	Re=100 $n_{stp}=10$
1.0000	1.0000	1.0000	1.0000
1.0000	1.0000	1.0000	0.9997
1.0000	1.0000	1.0000	0.9052
1.0000	1.0000	1.0000	0.3461
1.0000	0.9997	0.9998	0.2043
1.0000	0.9207	0.9592	0.0154
1.0000	0.2630	0.4187	0.0054
0.9974	0.1090	0.1180	-

Re=1000 $n_{stp}=10$	Re=1000 $n_{stp}=20$	Re=1000 $n_{stp}=20$
Re=100 $n_{stp}=20$	Re=400 $n_{stp}=10$	Re=400 $n_{stp}=20$
1.0000	1.0000	1.0000
0.9996	1.0000	1.0000
0.8845	1.0000	1.0000
0.3051	1.0000	1.0000
0.2018	1.0000	1.0000
0.0163	1.0000	1.0000
0.0041	0.9995	0.9996
-	0.7425	0.8862
-	-	0.2516
-	-	0.0159

Re=1000 $n_{stp}=20$	Re=1000 $n_{stp}=20$	Re=400 $n_{stp}=20$
Re=100 $n_{stp}=10$	Re=100 $n_{stp}=20$	Re=400 $n_{stp}=10$
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
0.9999	0.9999	1.0000
0.9787	0.9722	1.0000
0.5386	0.4623	1.0000
0.0394	0.0355	1.0000
0.0186	0.0153	1.0000
-	-	0.9998

Re=400 $n_{stp}=10$	Re=400 $n_{stp}=10$	Re=400 $n_{stp}=20$
Re=100 $n_{stp}=10$	Re=100 $n_{stp}=20$	Re=100 $n_{stp}=10$
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
0.9983	0.9974	1.0000
0.7386	0.6699	0.9999
0.1997	0.1804	0.9326
0.0240	0.0202	0.2285

## E.2 Cylinder

Re=200 $n_{stp}=20$	Re=200 $n_{stp}=10$	Re=50 $n_{stp}=20$
Re=200 $n_{stp}=10$	Re=50 $n_{stp}=10$	Re=200 $n_{stp}=10$
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	0.9902	0.9959
1.0000	0.4993	0.6328
1.0000	0.2007	0.3131
1.0000	0.0151	0.1785
0.9976	-	0.1404

Re=200 $n_{stp}=20$	Re=200 $n_{stp}=20$	Re=50 $n_{stp}=20$
Re=50 $n_{stp}=10$	Re=50 $n_{stp}=20$	Re=50 $n_{stp}=10$
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
0.9988	0.9995	1.0000
0.7316	0.8782	1.0000
0.2312	0.3394	0.9999
-	0.2439	-
-	0.1483	-

## E.3 NACA0012

Re=500 $n_{stp}=20$	Re=500 $n_{stp}=10$	Re=200 $n_{stp}=20$
Re=500 $n_{stp}=10$	Re=200 $n_{stp}=10$	Re=500 $n_{stp}=10$
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	0.9999
1.0000	0.9754	0.9520
1.0000	0.3988	0.3260
0.9611	0.0418	0.0327



Re=500 $n_{stp}=20$	Re=500 $n_{stp}=20$	Re=200 $n_{stp}=20$
Re=200 $n_{stp}=10$	Re=200 $n_{stp}=20$	Re=200 $n_{stp}=10$
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000
0.9880	0.9689	1.0000
0.4439	0.3574	0.9580
-	0.0024	-