ESCOLA TÈCNICA SUPERIOR D'ENGINYERIES INDUSTRIAL I AERONÀUTICA DE TERRASSA

DEVELOPMENT OF SOFTWARE FOR THE STUDY OF FLUID DYNAMICS. AERODYNAMICAL APPLICATIONS

AUTOR: FRANCESC BASSA PARPAL

TUTOR: ASSENSI OLIVA LLENA, FRANCESC XAVIER TRIAS MIQUEL

ENGINYERIA AERONÀUTICA

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TERRASSA – SETEMBRE 2014

Agraïments

Vull agrair el suport i l'ajuda necessaris per portar a terme aquest projecte rebuts per part de:

- La Universitat Politècnica de Catalunya i l'Escola Tècnica Superior d'Enginyeries Industrial i Aeronàutica de Terrassa.
- El Centre Tecnològic de Transferència de Calor i Massa per la oportunitat de realitzar el PFC amb el seu departament i per l'esforç en l'ensenyament de Dinàmica Computacional de Fluids.
- En especial, m'agradaria agrair als tutors d'aquest projecte, el doctor Assensi Oliva i el doctor Francesc Xavier Trias per tot el coneixement, mentoratge i esforç que han aportat en aquest projecte. Sense oblidar al doctor Oriol Lehmkul per l'ajuda que m'ha proporcionat.
- Al meu pare, mare i novia per donar-me la força i suport per seguir endavant en els moments difícils.
- Al meu amic i enginyer Francisco Font per l'assessorament tècnic amb el Matlab.

Resum

El present projecte te com a objectiu la creació d'un software propi, utilitzant el llenguatge de programació C++, que permeti resoldre les Equacions de Navier-Stokes de forma numèrica mitjançant la resolució per ordinador, el que es coneix com Dinàmica Computacional de Fluids (DCF).

La idea és poder simular i estudiar el comportament fluid-dinàmic de qualsevol geometria a través d'una simulació per ordinador sense haver de recórrer a túnels de vent i models a escala del cas d'estudi, que principalment es tracta de perfils aerodinàmics. Per a verificar el correcte funcionament del nostre software s'ha anat construint i comprovant de problemes senzills a problemes cada cop més complexes. Els problemes estudiats són l'*Smith-Hutton problem* i la *Driven Cavity*.

La comprovació final ha estat la simulació del que es coneix com l'*Square Cylinder*. Aquest cas consisteix en la simulació d'un túnel de vent a l'interior del qual hi ha un cilindre quadrat, que a efectes pràctics és la geometria més senzilla per a un perfil alar.

Resumen

El presente proyecto tiene como objetivo la creación de un software propio, utilizando el lenguaje de programación C++, que permita resolver las Ecuaciones de Navier-Stokes de forma numérica mediante la resolución por ordenador, lo que se conoce como Dinámica Computacional de Fluidos (DCF).

La idea es poder simular y estudiar el comportamiento fluido-dinámico de cualquier geometría a través de una simulación por ordenador sin tener que recorrer a túneles de viento y modelos a escala de la geometría a estudiar, que principalmente se trata de perfiles aerodinámicos. Para verificar el correcto funcionamiento de nuestro software se ha ido construyendo y comprobando a partir de problemas sencillos a problemas cada vez más complejos. Los problemas estudiados son el *Smith-Hutton problema* y la *Driven Cavity*.

La comprobación final ha sido la simulación de lo que se conoce como el *Square Cylinder*. Este caso consiste en la simulación de un túnel de viento en el interior del cual hay un cilindro cuadrado, que a efectos prácticos es la geometría más sencilla para un perfil alar.

Abstract

The present Project has as objective the creation of a self-built software, using the programming language C++, that allows to solve numerically the Navier-Stokes Equations using a computer, what is known as Computational Fluids Dynamics (CFD).

The idea is to be able to simulate and study the fluid-dynamic behaviour of any given geometry throw a computer simulation without having to use a wind tunnel and a scaled model of the geometry of study, mainly alar profiles. In order to verify the correct functioning of our software it has been built from simpler cases to more complex ones. The studied problems are the *Smith-Hutton problem* and the *Driven Cavity*.

The final test has been the simulation of what is known as the *Square Cylinder*. This case consists in the simulation of a wind tunnel within its interior contains a square cylinder, which by all effects it's the simplest geometry for an alar profile.

Index

1 Introduction 13
1.1 Objective 13
1.2 Scope 13
1.3 Justification 13
1.4 History and precedents 14
2 Physical and technical analysis 21
2.1 The Navier-Stokes Equations 21
2.1.1 Simplified equations 21
2.1.2 Adimensionalization 24
2.2 Finite Volume Method (FVM) 24
2.2.1 Meshes 25
2.2.2 Mesh formulation 26
2.2.3 Numerical scheme 28
2.3 Solvers 31
2.3.1 Gauss-Seidel (G-S) 32
2.3.2 Conjugate Gradient (CG) 33
2.4 Boundary Conditions 34
2.4.1 Types of BC 34
2.4.2 Implementation of the BC 35
2.5 Fractional Step Method (FSM) 36
2.5.1 Implementation of the FSM 36
2.5.2 Final discretized equations 38
2.5.3 Solving algorithm process 40
3 Software verification 42
3.1 Code and language checking 43
3.2 Method of Manufactured Solutions (MMS) 44
3.3 Comparison with practical cases 45
3.3.1 Smith-Hutton problem 45

3.3.2 Driven Cavity Problem 48
4 Case of study: The Square Cylinder 65
4.1 Physics description 66
4.2 Geometry 67
4.3 Mesh 68
4.4 Boundary Conditions 71
4.4.1 Inlet conditions 71
4.4.2 Walls conditions 71
4.4.3 Object conditions 71
4.4.4 Outlet conditions 71
4.4.5 Fixed pressure point 72
4.5 Aerodynamic forces 72
4.6 Vortex Shedding 73
4.7 Results analysis and comparison 74
4.7.1 Pre-Vortex Shedding 74
4.7.2 Vortex Shedding range 80
4.7.3 Final thoughts on the Square Cylinder problem 92
5 Applications, future lines and conclusions 93
5.1 Applications 93
5.2 Future lines 93
5.3 Conclusions 94
6 Economic and environmental study 95
6.1 Economic study 95
6.2 Environmental impact 95
7 Bibliographic references 97
Annex A 99
Annex B 109

Figure list

Fig. 1 – Archimedes discovering how to measure densities 14
Fig. 2 - Drawing of a Roman aqueduct and canal system 15
Fig. 3 - Da Vinci flying machines 15
Fig. 4 - Waves at a liquid surface 16
Fig. 5 - Mechanical desk calculator from 1950s 17
Fig. 6 - CDC6600 super computer (1970s) 17
Fig. 7 - Cray-1 super computer (1980s) 18
Fig. 8 - Pyramid of main CFD code progress by decades, extracted from [3] 18
Fig. 9 – Control Volume 25
Fig. 10 – Uniform, non uniform and unstructured mesh respectively, extracted from [20] . 25
Fig. 11 – Co-located mesh 26
Fig. 12 – Centered mesh, staggered X and staggered Y respectively. 27
Fig. 13 – Control Volume, neighbour disposition and nomenclature for nodes and faces 28
Fig. 14 – Convective + viscous term vector field decomposition, extracted from [9] 36
Fig. 15 – Solving algorithm scheme 41
Fig. 16 - Venn diagram on CFD set of skils, extracted from [3] 42
Fig. 17 -– Smith-Hutton domain scheme, extracted from [12] 46
Fig. 18 – Contour plot of ϕ for ρ/Γ = 1e3. 48
Fig. 19 – Driven Cavity domain scheme 49
Fig. 20 – Mesh scheme for the Driven Cavity problem 51
Fig. 21 – Velocity u comparison for Re = 100 53
Fig. 22 – Velocity v comparison for Re = 100 53
Fig. 23 – Velocity u comparison for Re = 400 54
Fig. 24 – Velocity v comparison for Re = 400 54
Fig. 25 – Velocity u comparison for Re = 1000 55
Fig. 26 – Velocity v comparison for Re = 1000 55
Fig. 27 – Velocity u comparison for Re = 3200 56
Fig. 28 – Velocity v comparison for Re = 3200 56
Fig. 29 – Velocity u comparison for Re = 5000 57
Fig. 30 – Velocity v comparison for Re = 5000 57
Fig. 31 – Velocity u comparison for Re = 7500 58
Fig. 32 – Velocity v comparison for Re = 7500 58
Fig. 33 - Contour plot u velocity for Re=100 60
Fig. 34 - Contour plot v velocity for Re=100 60
Fig. 35 - Streamline for Re=100 60
Fig. 36 - Contour plot u velocity for Re=1000 61
Fig. 37 - Contour plot v velocity for Re=1000 61
Fig. 38 - Streamline for Re=1000 61

Fig. 39 - Contour plot u velocity for Re=5000 62
Fig. 40 - Contour plot v velocity for Re=5000 62
Fig. 41 - Streamline for Re=5000 62
Fig. 42 – Scheme of the Square Cylinder case 65
Fig. 43 - Scheme of a Parabolic inlet flow 67
Fig. 44 - Scheme of the domain for the Square Cylinder case, extracted from [15] 68
Fig. 45 – Mesh scheme for the Square Cylinder problem 70
Fig. 46 - Scheme of the meshed domain without axis scaling 70
Fig. 47 - Contour plot of u velocity for Re=40 75
Fig. 48 - Contour plot of v velocity for Re=40 75
Fig. 49 - Contour plot of pressure for Re=40 76
Fig. 50 - Streamlines with velocity module for Re=40 76
Fig. 51 – Streamlines with velocity module for Re=1 77
Fig. 52 - Streamlines with velocity module for Re=10 78
Fig. 53 - Streamlines with velocity module for Re=20 78
Fig. 54 - Streamlines of reference for different Re numbers. (a) Re=1, (b) Re=30 79
Fig. 55 - Drag coefficient vs Reynolds number comparison 80
Fig. 56 - Contour plot of u velocity for Re=100 at t=4,40 81
Fig. 57 - Contour plot of v velocity for Re=100 at t=4,40 81
Fig. 58 - Contour plot of pressure for Re=100 at t=4,4 82
Fig. 59 - Streamlines for Re=100 at t=4,4 82
Fig. 60 - Contour plot of u velocity for Re=100 at t=24,55 83
Fig. 61 - Contour plot of v velocity for Re=100 at t=24,55 83
Fig. 62 - Contour plot of pressure for Re=100 at t=24,55 84
Fig. 63 - Streamlines for Re=100 at t=24,55 84
Fig. 64 - Contour plot of u velocity for Re=100 at t=26,02 85
Fig. 65 - Contour plot of v velocity for Re=100 at t=26,02 85
Fig. 66 - Contour plot of pressure for Re=100 at t=26,02 86
Fig. 67 - Streamlines for Re=100 at t=26,02 86
Fig. 68 - Drag coefficient variation with time for Re=100 87
Fig. 69 - Lift coefficient variation with time for Re=100 87
Fig. 70 - Streamlines for Re=60 at t=26,20 88
Fig. 71 - Streamlines for Re=150 at t=10,75 88
Fig. 72 - Streamlines for Re=200 at t=12,34 89
Fig. 73 - Streamlines of reference for different Re numbers. (c) Re=60, (d) Re=200 90
Fig. 74 - Drag coefficient vs Reynolds number comparison 90
Fig. 75 - Strouhal number vs Reynolds number comparison 91

Table list

Table 1 - Computational power increase comparison, extracted and adapted from [3]..... 19
Table 2 - Main parameters of the governing equations 23
Table 3 – Main differences between explicit and implicit methods 30
Table 4 – Example of a 3x3 domain with nodal numeration 31
Table 5 – Result comparison of the Smith-Hutton problem 47
Table 6 - Geometry dimensions 68
Table 7 - Strouhal number error comparison 91
Table 8 - Human resources cost estimation 95
Table 9 - General cost estimation 95

Annex A. Figure list

Fig. I - Contour plot of u velocity for Re=1 100
Fig. II - Contour plot of v velocity for Re=1 100
Fig. III- Contour plot of pressure for Re=1 101
Fig. IV - Contour plot of u velocity for Re=10 101
Fig. V - Contour plot of v velocity for Re=10 102
Fig. VI - Contour plot of u velocity for Re=20 102
Fig. VII - Contour plot of v velocity for Re=20 103
Fig. VIII - Contour plot of u velocity for Re=60 at t=26,20 103
Fig. IX - Contour plot of v velocity for Re=60 at t=26,20 104
Fig. X - Contour plot of u velocity for Re=150 at t=10,75 104
Fig. XI - Contour plot of v velocity for Re=150 at t=10,75 105
Fig. XII - Contour plot of u velocity for Re=180 at t=16,12 105
Fig. XIII - Contour plot of v velocity for Re=180 at t=16,12 106
Fig. XIV - Streamlines for Re=180 at t=16,12 106
Fig. XV - Contour plot of u velocity for Re=200 at t=12,34 107
Fig. XVI - Contour plot of v velocity for Re=200 at t=12,34 107

1 Introduction

1.1 Objective

The objective of this project is the creation of a self-built software to study the dynamic behaviour of fluids by solving numerically the Navier-Stokes Equations. The main aim of this software is to be able to solve 2D geometries for non-turbulent flows. The software will serve as a virtual wind tunnel for testing.

1.2 Scope

The scope of study will take the following considerations:

- The case of study will consider that the scale is large enough to assume the hypothesis of the continuity of mater.
- The software must be able to solve transitory cases.
- The Navier-Stokes Equations will be solved with some simplifying hypothesis.
- The flow has constant physical properties in the cases of study.
- The flow of study must be pure and Newtonian.
- The flow is considered incompressible.
- The domain and the geometries must be 2D.
- The theory of FVM will be introduced.
- The Smith-Hutton, Driven Cavity and Square Cylinder problems will be implemented and solved.

1.3 Justification

In nowadays industry companies spend thousands of euros every year by testing their products and optimizing them in order to make them the most efficient as possible. For sectors such as aerospace, aeronautical, industrial and automotive the testing in wind tunnels to achieve the most lift, the least drag, reduce friction to reduce consume and any parameter they are interested in has become crucial for success. A few years back into the industry, having a wind tunnel for testing was the only way to go but building it, running it and keeping good care of it is massive budget for the companies and if you are a small company or a new one it is a no go. With the exponential grow shown in the computer world and the massive increase in computational power has opened a new cheaper and viable option, the Computational Fluid Dynamics (CFD).

In the present, we have a massive calculus power in a simple laptop allowing us to run simulations in our own home without owning a wind tunnel. The precision of those simulations is to the same level as the wind tunnel test, in fact the DNS (Direct Numerical Simulation) gives

an exact result and has even better precision than a physical test (a DNS simulation needs a computer cluster on most cases).

Companies have seen this potential and are shifting towards the CFD option. It is way much easier for a company to buy a cluster or just some good computers and build a team around simulating CFD. As CFD has become more and more popular, many CFD integrated softwares have risen giving companies an easy and powerful tool to work with.

From the educational point of view, a commercial CFD software would allow us to solve much more complex geometries but since we are not a company and for the purpose of this project, since we do not have any commercial needs the aim is to truly understand the physic, maths and programming behind the CFD. For this reason a self-built software was the chosen option. It allows us to truly understand the process that brings us from the Navier-Stokes Equations to those beautiful and colourful graphs of velocity, lift and drag that are the much wanted results.

1.4 History and precedents

To find the one first time humankind showed some scientific interest in fluid physics we have to go back as far as Ancient Greece at around 200 BC where Archimedes (287-212 BC) initiated the fields of hydrostatics and determined how to measure densities and volumes of objects.

Fig. 1 – Archimedes discovering how to measure densities

The main focus of the Greek civilization in the field of fluid dynamics was on waterworks: aqueducts, canals, harbours and bathhouses. Interest shared with Romans who perfected it to an art and science.

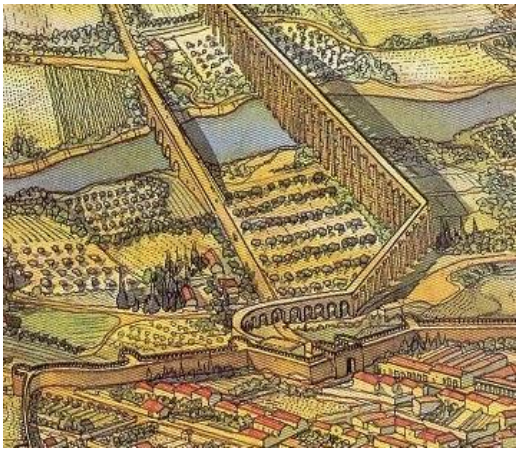


Fig. 2 - Drawing of a Roman aqueduct and canal system

In Renaissance we encounter the first genius of aerodynamics, Leonardo Da Vinci (1452-1519). He was a painter, anatomist, artist, scientific, sculptor, inventor and engineer among other things, a total polymath. He planned and supervised canal and harbour works over a large part of middle Italy and he designed what are known to be some of the first flying devices and helicopters.

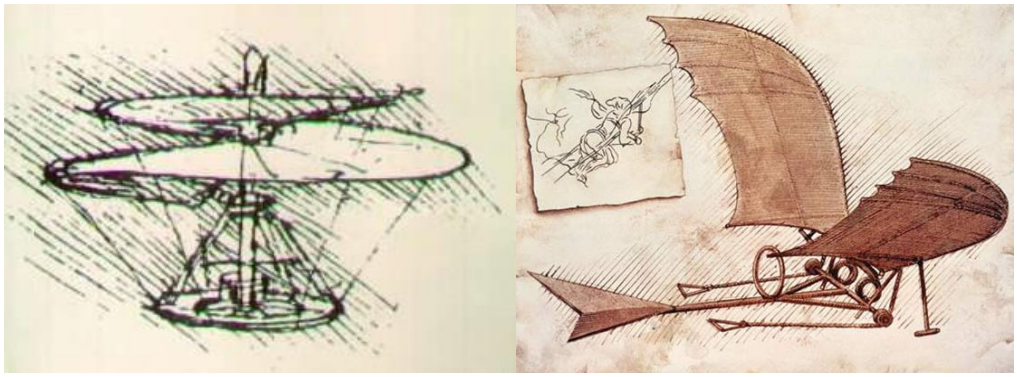


Fig. 3 - Da Vinci flying machines

In the late 17th Century, Isaac Newton (1643-1727) tried to quantify and predict fluid flow phenomena through his elementary Newtonian physical equations. Among his achievements it is the concept of Newtonian viscosity in which stress and the rate of strain vary linearly, the reprocity principle and the relationship between the speed of waves at a liquid surface and their wavelength.

Fig. 4 - Waves at a liquid surface

In the 18th and 19th centuries, a major step was taken in the mathematical modelling and description of the fluid motion. Daniel Bernoulli (1700-1782) derived the Bernoulli's Equation and Leonhard Euler (1707-1783) proposed the Euler Equations, which describe the conservation of momentum for an inviscid fluid, and conservation of mass. Claude Louis Marie Henry Navier (1785-1836) and George Gabriel Stokes (1819-1903) introduced viscous transport into the Euler Equations, resulting in the Navier-Stokes Equations. These equations are the basis of the modern day Computational Fluid Dynamics (CFD). The equations are so closely coupled and difficult to solve that until now they still do not have an analytical solution, being its analytical solution one of the Millennium Prize Problems by the Clay Mathematics Institute with 1M$ prize, meaning that they only can be solved by a numerical approach.

In the early 20th Century, efforts were done in refining theories of boundary layers and turbulence in fluid flow. Ludwig Prandtl (1875-1953) and his boundary layer theory, Theodore von Karman (1881-1963) famous for his studies on the now known as von Karman vortex or Geoffrey Ingram Taylor (1886-1975) and his statistical theory of turbulence are just some of the many great man who did a fantastic work during that century.

The earliest numerical solution for a flow past a cylinder was carried out in 1933 by A. Thom and reported in England **[1]**. But what could be considered as the first CFD computation ever was performed by M. Kawaguti **[2]** in Japan and obtained a similar solution for the flow around a cylinder in 1953 by using a mechanical desk calculator, working 20 hours per week during 18 months.

Fig. 5 - Mechanical desk calculator from 1950s

During the 1960s, the underlying principles of fluid dynamics and the formulation of the governing equations were well established. The theoretical division of NASA contributed to develop many numerical methods that are still in use in CFD today, such as the following methods: Particle-In-Cell (PIC), Vorticity-Stream function methods, Arbitrary Lagrangian-Eulerian (ALE) methods, and the ubiquitous k - e turbulence model. In the 1970s, the Imperial College of London developed Parabolic flow codes (GENMIX), the SIMPLE algorithm and the TEACH code.

Fig. 6 - CDC6600 super computer (1970s)

In 1980, Suhas V. Patankar published *Numerical Heat Transfer and Fluid Flow* **[3]**, probably the most influential book on CFD to date, and the one that originated lots of CFD codes. It was also during this decade that CFD did the final big jump that it needed, commercial CFD codes came into the market and companies started to invest in CFD software and computers to perform those calculus. It was not only thanks to the big steps done in the previous two decades in

algorithm and methods research but also due to the exponential grow of numerical calculus of computers.

Fig. 7 - Cray-1 super computer (1980s)

Nowadays all big companies that work in the space, aeronautical or automotive sectors widely use CFD software and even have entire departments specialized on those tasks. CFD have reduced enormously the need of physical experiments on wind tunnel with real pieces or scale models of their products or test parts. The increasing trend of numerical power of computers is still maintained, we can easily perform CFD simulations on our laptops or desk computers when only 20 years before we needed entire clusters for the same computation.

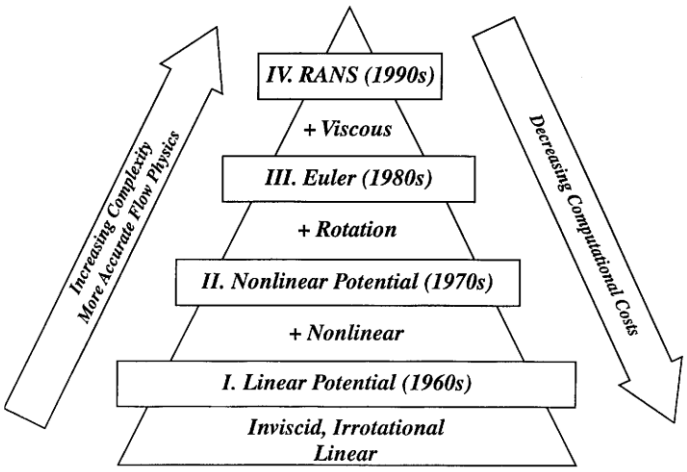


Fig. 8 - Pyramid of main CFD code progress by decades, extracted from [3]

1970	CDC6600	1 MEGAFLOPS	10^6
1980	Cray-1	100 MEGAFLOPS	10^8
2007	Home Box Cluster Four 3 GHz dual core CPUs ~8000€	2.5 GIGAFLOPS	$2{,}5*10^9$
2014	Quadcore Intel i7 laptop ~1000€	>70 GIGAFLOPS	$>70*10^9$

Table 1 - Computational power increase comparison, extracted and adapted from [3]

2 Physical and technical analysis

2.1 The Navier-Stokes Equations

In this section we will briefly explain the Navier-Stokes Equations (N-S) and the physics behind it.

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

$$\frac{\partial (\rho \vec{u})}{\partial t} + \nabla \cdot (\rho \vec{u}\vec{u}) = -\nabla \mathrm{p} + \nabla \cdot \vec{\tau} + \rho \vec{g} \qquad (2.2)$$

$$\frac{\partial (\rho (e_i + e_c))}{\partial t} + \nabla [\rho \vec{u} (e_i + e_c)] = -\nabla \cdot (\vec{u}\mathrm{p}) + \nabla (\vec{u} \cdot \vec{\tau}) - \nabla \cdot \vec{q} + \rho \vec{g} \cdot \vec{u} \qquad (2.3)$$

Where ρ is the density, t the time, $\vec{u}$ the velocity vector, p the pressure, $\vec{\tau}$ is the total stress tensor, $\vec{g}$ the gravity force, e_i the internal energy, e_c the kinetic energy and $\vec{q}$ heat flow.

The equation (2.1) is known as the continuity equation which represents the principle of mass conservation. The equation (2.2) is the momentum conservation equation, it has been expressed in its vectorial form in order to synthetize the 3 equations, one for each direction. The equation (2.3) is the conservation of energy. Note that they have been expressed in their derivative form, those are deduced from the integral form but since its deduction has already been done and it is explained in big detail in **[4]** we will assume them as correct.

Some assumptions have already been done in the previous equations. The hypothesis are:

- Continuity of mater.
- Relativity effects negligible.
- Inertial reference system.
- Coriolis forces negligible.
- Magnetic and electrical forces negligible.

2.1.1 Simplified equations

The N-S equations as shown in the previous section can be computed but would require a very complex software and a lot of calculus power. To better fit the scope of this project, some simplifications and hypothesis have been introduced.

- Constant physical properties (density, viscosity, etc).
- Newtonian fluid.
- The heat throw viscous dissipation is negligible.
- Only 2D cases are taken into consideration.

Given the previous hypothesis our code will only be able to study incompressible flows (variable density). Other properties such as viscosity, thermal conductivity or heat capacity are also considered constant, generally these properties are function of temperature so our code won't be able to solve problems with big changes of temperature that would produce a significant change on the mentioned properties.

A Newtonian fluid is a fluid where the forces are proportional to the gradient of the velocity. Some flows such as honey, blood and other high viscous fluids are non-Newtonian flows so they cannot be simulated by our code.

Considering the viscous dissipation from the friction that it produces with itself doesn't limit our code in any way since this a term that is negligible at almost all levels of study, it's a term that gives almost no information but is hard to model.

By taking only into consideration 2D cases we narrow down all complex geometries and problems. Basically we are limited to cases where the flow behaves the same way all along the *z* axis. This simplification has been done in order to simplify the code and the domain but taking into consideration that for the scope of this project a 2D flow allows us to illustrate perfectly the aim and showcase the basics of CFD.

The governing equations obtained from simplifying (2.1)-(2.3) are:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\ =\ 0 \qquad (2.4)$$

$$\rho\frac{\partial u}{\partial t}+\rho u\frac{\partial u}{\partial x}+\ \rho v\frac{\partial u}{\partial y}=\ -\ \frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\ \frac{\partial^2 u}{\partial y^2}\right) \qquad (2.5)$$

$$\rho\frac{\partial v}{\partial t}+\rho u\frac{\partial v}{\partial x}+\ \rho v\frac{\partial v}{\partial y}=\ -\ \frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\ \frac{\partial^2 v}{\partial y^2}\right)+\ \rho g\beta(T-T_\infty) \qquad (2.6)$$

$$\rho\frac{\partial T}{\partial t}+\rho u\frac{\partial T}{\partial x}+\ \rho v\frac{\partial T}{\partial y}=\ \frac{\lambda}{c_p}\left(\frac{\partial^2 T}{\partial x^2}+\ \frac{\partial^2 T}{\partial y^2}\right)+\frac{\Phi}{c_p} \qquad (2.7)$$

Where u and v are the velocity components, x and y the spatial coordinates, ρ the density, t the time, μ the viscosity, p the pressure, T temperature, β the coefficient of volumetric thermal expansion, g the gravitational acceleration, λ the conductivity, c_p the specific heat and Φ heat sources.

They are partial coupled differential equations. The 4 unknowns are the pressure, temperature and the two velocity components *u* and *v*. Two strong coupling characterise this equations system:

- Pressure-velocity. There is no specific pressure equation. For incompressible flows, the pressure is the field that makes the velocity accomplish the mass conservation equation.
- Temperature-velocity. This coupling is only present for natural convection, mixed convection or temperature dependant physical properties. In forced convection and constant physical properties, the velocity field does not depend on temperature field.

For all the cases of study in this project, we can dismiss the temperature-velocity coupling. All the equations (2.4)-(2.7) can be summarized in the General Convection-Diffusion equation. More information on the Convection-Diffusion equations can be found in **[7]**. For Cartesian coordinates, incompressible flow and constant physical properties the equation is:

$$\rho\frac{\partial\phi}{\partial t}+\rho u\frac{\partial\phi}{\partial x}+\ \rho v\frac{\partial\phi}{\partial y}=\ \Gamma\left(\frac{\partial^2\phi}{\partial x^2}+\ \frac{\partial^2\phi}{\partial y^2}\right)+S \qquad (2.8)$$

The accumulation of ϕ, plus the net convective flow has to be the net diffusive flow plus the generation of ϕ per unit of volume. The diffusive term flows from greater to smaller value of ϕ.

According to the convection diffusion equation, we can write a table with the appropriate parameters in order to reproduce the governing equations. Where ϕ, Γ and *S* are:

EQUATION	ϕ	Γ	*S*
CONTINUITY	1	0	0
MOMENTUM IN *X* DIRECTION	u	μ	$-\frac{\partial p}{\partial x}$
MOMENTUM IN *Y* DIRECTION	v	μ	$-\frac{\partial p}{\partial y}$
ENERGY (CONSTANT C_p)	T	$\frac{\lambda}{c_p}$	$\frac{\Phi}{c_p}$

Table 2 - Main parameters of the governing equations

Each term of the equation (2.8) gives us a different information:

- The first term, known as the transitory term, gives us information on the temporal variation of our variable.
- The second term, known as the convective term, gives us information on the special transport of our variable.
- The third term, known as the diffusive term, represents the transport of the variable due to the concentration of gradients.
- The last term, known as the source term, is a focus of generation or a sump.

2.1.2 Adimensionalization

Since nature and computers do not understand or think in units, it makes sense for the study of CFD to adimensionalize all our dimensions. To adimensionalize our equations, we divide each of our variables in the equation of study by a reference value of that variable at our choice. Generally the problem is adimensionalized with a significant value for the case itself. The variables adimensionalized in our project are: The component *x* of the velocity (u), the component *y* of the velocity (v), pressure (p), *x* position, *y* position and time (t).

In our project all velocities will be adimensionalized with the same reference parameter and same will happen for all positions, using a characteristic value of each case. The value used to adimensionalize pressure and time is not very important for our cases but it would seem logical to use t_{ref}=1s and P_{ref} as the atmospheric pressure.

One of the vital parameters that appear once we adimensionalize is the Reynolds number:

$$Re = \frac{\rho\, U_\infty\, X_{object}}{\mu} \qquad (2.9)$$

Where Re is the Reynolds number, ρ the density, U_∞ the velocity upstream of the object, X_{object} the characteristic dimension of the object and μ is the viscosity.

The Reynolds number is a dimensionless quantity defined as the ratio of inertial forces to viscous forces and consequently quantifies the relative importance of these two types of forces for given flow conditions.

The real value of adimensionalization comes from the fact that once it is done properly, it gives us a generalized result that can be used as solution for multiple real cases instead of studying each one of them separately. It also allows us to compare our simulated results with real experiments.

2.2 Finite Volume Method (FVM)

The Navier-Stokes Equations do not have an analytical solution for most of cases meaning the only way to solve complex problems is by numerical methods. In order to do so we need to discretize the equations and the domain.

The FVM consists in the division of the domain in a finite number of control volumes. The number of control volumes depend on the precision desired or the complexity of the geometry or problem to study. The FVM assumes that all the fluid inside the same control volume has the same value of pressure, velocity or any given property and that value is assigned to the centre of the element. For more information on the FVM refer to **[7].**

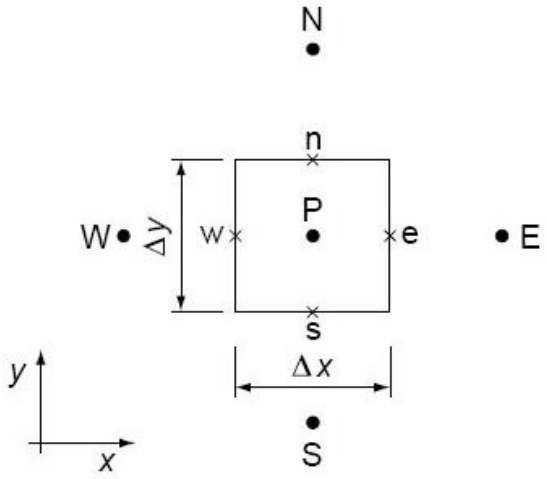


Fig. 9 – Control Volume

As we are modelling a continuous domain into a discretized one, we are making an approximation but the more and the smaller the control volumes are the more accurate the approximation is. The whole set of single volumes is called a mesh, the mesh covers and fills our entire domain and geometry.

2.2.1 Meshes

The different types of meshes are determined the shapes of the volumes that form it, their position with respect to the modelled domain or the relation between them. Let's take a look into some of the types:

- Structured: They are identified by regular connectivity, the elements are quadrilateral in 2D and hexahedral in 3D.
  - Uniform: All elements of the mesh are equally separated.
  - Non uniform: The distance between elements is variable.
- Unstructured: They are identified by irregular connectivity. These grids typically employ triangles in 2D and tetrahedral in 3D.

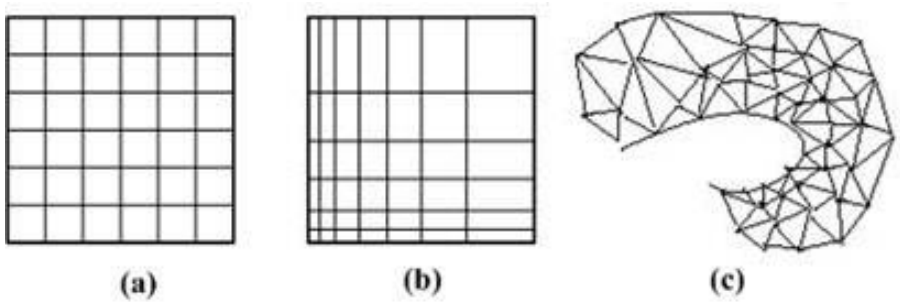


Fig. 10 – Uniform, non uniform and unstructured mesh respectively, extracted from [20]

Structured meshes are easy to build and to store their properties, specially for structured-uniform meshes that share most of the geometrical properties among all the elements. Structured non-uniform meshes are easier to build and store than unstructured but they allow us to densify the mesh in the points where it is more interesting for the complexity of the problem. Unstructured meshes are ideal for complex geometries but are hard to build and require a lot of space to store their geometrical properties and relations.

Taking into consideration the simplicity in the geometry of the cases studied in this project, all meshes will be structured but we will use both uniform and non uniform meshes depending on the case of study (it will be specified and defined in each case description).

The number of elements used for each simulation will also vary, some cases are so simple that densifying the mesh wouldn't provide any extra precision and some cases are limited by the computation time. We need to find a balance between time and precision.

2.2.2 Mesh formulation

In each case of study we want to know the value of the temperature, pressure and horizontal and vertical speed of all the points of the domain. In order to do so we can find 2 types of formulation.

- Co-located formulation. All values of the variables are obtained in the centre of our control volume. At Fig. 11 we can see a co-located mesh where the horizontal velocity is represented by a green arrow, the vertical velocity by a red one and the pressure and temperature by a black dot.

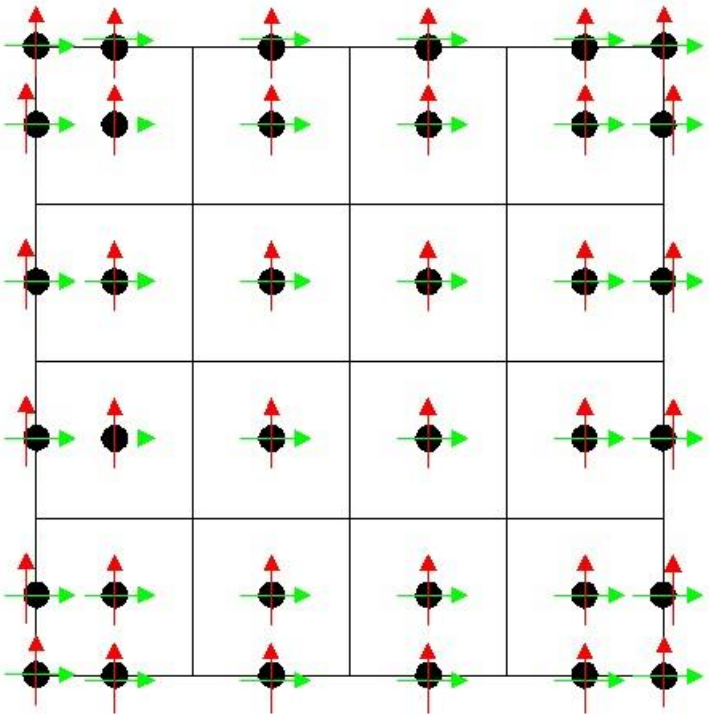


Fig. 11 – Co-located mesh

It is the simplest formulation and is usually used along with unstructured meshes. But for structured meshes it can produce errors. The main problem is known as the *checkerboard* or *odd-even decoupling*, where the values of the variables along the domain adopt alternate values (hence resembling a chess board with black and white cells). This comes from the fact that for the pressure-velocity coupling we need to correct the velocities using the gradient of the pressure using only the neighbours and no the pressure of the own element. A decoupling between pressure and velocity appears and produces this type of erratic solution.

In order to solve this problem another formulation is used, specially in structured meshes.

- Staggered meshes. On a staggered mesh the scalar variables (pressure and temperature) are stored in the cell centres of the control volumes, whereas the velocities are located at the cell faces. Three different meshes are created, one for the scalar variables and one mesh for each component of the velocity which are the truly staggered meshes as they are staggered with respect to the centred mesh for the pressure. The mesh for the horizontal velocities coincides with the vertical walls (*staggered X*) and the vertical velocity with the horizontal ones (*staggered Y*). It can be better understood by viewing the three meshes over the same domain in the Fig. 12.

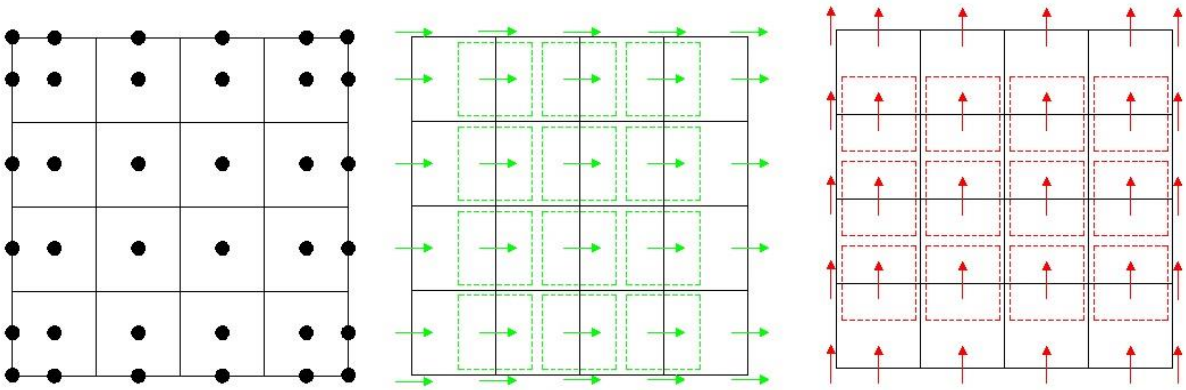


Fig. 12 – Centered mesh, staggered X and staggered Y respectively.

Note that the mesh for the velocities have different dimensions than the centred mesh due to being staggered. For example, if the centred mesh has Nx x Ny elements then the *staggered X* will have Nx-1 x Ny elements and the *staggered Y* Nx x Ny-1.

2.2.3 Numerical scheme

2.2.3.1 Spatial discretization

Integrating the General Convection-Diffusion Equation (2.1) into a rectangular finite volume the discretization equation can be written:

$$\rho\frac{\partial\phi}{\partial t}+[(\rho u\phi)_e^{n+1}-(\rho u\phi)_w^{n+1}]\Delta y+[(\rho v\phi)_n^{n+1}-(\rho v\phi)_s^{n+1}]\Delta x$$
$$=\left[\left(\Gamma\frac{\partial\phi}{\partial x}\right)_e^{n+1}-\left(\Gamma\frac{\partial\phi}{\partial x}\right)_w^{n+1}\right]\Delta y$$
$$+\left[\left(\Gamma\frac{\partial\phi}{\partial y}\right)_n^{n+1}-\left(\Gamma\frac{\partial\phi}{\partial y}\right)_s^{n+1}\right]\Delta x+S_P^{n+1}\Delta x\Delta y \qquad (2.10)$$

The temporal discretization will be considered in a following section of the numerical schemes.

The following hypothesis are done:

- In the integration process, the convective and diffusive flows have been considered constant through each face of the control volume.
- $(\text{spatial deviation})^n = (\text{spatial deviation})^{n+1}$
  $(\text{spatial deviation})_w = (\text{spatial deviation})_e$
  $(\text{spatial deviation})_s = (\text{spatial deviation})_n$

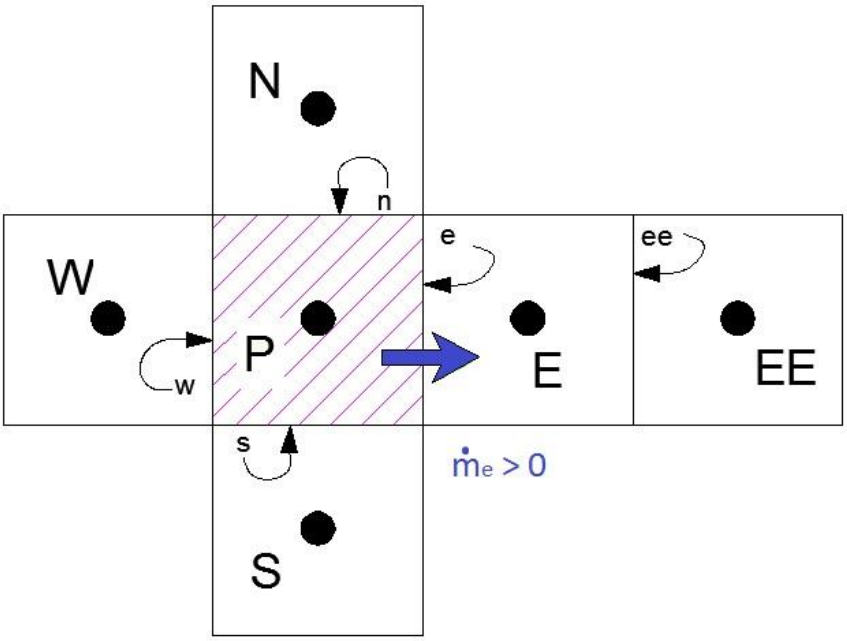


Fig. 13 – Control Volume, neighbour disposition and nomenclature for nodes and faces

It can be seen that in the discretized equation (2.10) the convective and diffusive terms are evaluated at the volume faces, whereas the dependent variable is known at the cell centre. The evaluation of the variable ϕ at the cell face is carried out.

The nomenclature that we will use can also be appreciated in Fig. 13. The sub-indexes refer to the spatial discretization where capitals represent values at a node and small letters values at a wall. *W, e, n* and s .The upper-indexes refer to the temporal discretization and represent the time iteration of the evaluated variable.

Conductive flux is calculated as an arithmetic mean:

$$\left(\frac{\partial \phi}{\partial x}\right)_e = \frac{\phi_E - \phi_P}{\delta x_e} \quad or \quad \left(\frac{\partial \phi}{\partial y}\right)_n = \frac{\phi_N - \phi_P}{\delta y_n} \qquad (2.11)$$

Where δ is the distance between the two points where the variables are evaluated.

The numerical schemes evaluate the variable using nearest nodes: east (E), east-east (EE), west (W), north (N) and south (S).

Most significant low order numerical schemes are:

- Central Difference Scheme (CDS): It is a second order scheme, variable at the cell face is calculated as the arithmetic mean. That is:

$$\phi_e = \frac{1}{2}(\phi_P + \phi_E) \qquad (2.12)$$

- Upwind Difference Scheme (UDS): It is a first order scheme and the value of ϕ at the cell face is equal to the value of ϕ at the grid point on the *upwind* side of the face. That is:

$$\phi_e = \phi_P \; if \; \dot{m}_e < 0 \qquad (2.13)$$

$$\phi_e = \phi_E \; if \; \dot{m}_e > 0 \qquad (2.14)$$

- Hybrid Difference Scheme (HDS): Uses CDS for low velocities and UDS for high velocities.
- QUICK: Is a third order scheme that uses a quadratic interpolation of three cells:

$$\phi_e = \begin{cases} \frac{1}{2}(\phi_P + \phi_E) - \frac{1}{8}(\phi_{EE} - 2\phi_E + \phi_P) & \text{if } \dot{m}_e < 0 \\ \frac{1}{2}(\phi_P + \phi_E) - \frac{1}{8}(\phi_E - 2\phi_P + \phi_W) & \text{if } \dot{m}_e > 0 \end{cases} \qquad (2.15)$$

The scheme used in the entirety of our simulations is the CDS not only because of its simplicity but also because it is very effective in comparison with some of the others presented. The

Upwind is also quite simple but is not effective because it is adding dissipation of cinematic energy.

2.2.3.2 Temporal discretization

In the previous section we discussed the spatial discretization but time is a crucial parameter for simulations in CFD. In order to compute how a variable changes with time and obtain the future value of our parameters is the main goal. In order to do so we need to discretize in time, specially for transitory cases.

In equation (2.9) we didn't discretize the transitory term $\rho \frac{\partial \phi}{\partial t}$. There are two methods of time discretization:

- Explicit methods. They value of new time step only depends of previous steps.
- Implicit methods. The value of the new time step is function of previous steps and the new time step itself.

The main differences and uses are:

PROPERTIES	EXPLICIT	IMPLICIT
$\boldsymbol{\phi^{n+1}}$	$\boldsymbol{f(\phi^n)}$	$\boldsymbol{f(\phi^n, \phi^{n+1})}$
$\boldsymbol{\Delta t}$	Small	Any
OPTIMAL FOR	Transitory	Steady
ITERATION COST	Low	Very high
STABILITY	Low	Very high

Table 3 – Main differences between explicit and implicit methods

In this project we will only use explicit methods so finally we can obtain our discretization for the transitory term.

$$\rho \frac{\partial \phi}{\partial t} = \rho \frac{\phi_P^{n+1} - \phi_P^n}{\Delta t} \Delta x \, \Delta y \qquad (2.16)$$

Where Δt is the time increment, Δx and Δy the horizontal and vertical element size respectively.

Since the implicit method makes a prediction of the values for the next time step, the further our prediction is the more uncertain that prediction is. This can produce instabilities in our code that need to be controlled. In order to ensure the stability of the prediction certain conditions need to be fixed:

$$\Delta t \leq C_C \left(\frac{\Delta x_i}{|u_i|} \right)_{max} \qquad C_C = 0.3 \qquad (2.17)$$

$$\Delta t \leq C_D \left(\frac{(\Delta x_i)^2}{\mu} \right)_{max} \qquad C_D = 0.2 \tag{2.18}$$

2.3 Solvers

Once we finally have our equations discretized we have to start to think in an effective way to solve them. In many cases we will have to solve an equation which comes in the form:

$$A \cdot x = b \tag{2.19}$$

Where A is a matrix, *x* is a vector and the variable we want solve or obtain and *b* is another vector that usually is called the source term due to the fact that in the majority of cases contains the values of the source term of the General Convection-Diffusion (2.8). The vectors *x* and *b* have a position for each control volume, meaning that if we did Nx elements in the X direction and Ny elements in the Y direction then the size is Nt = Nx x Ny. The matrix *A* stores the information of how every node is related to each other resulting in a square matrix of Nt x Nt dimensions.

Let's take a look at an example of a domain where Nx=3, Ny=3 and hence Nt=9. The resulting *A* matrix will be of 9x9.

7	**8**	**9**
4	**5**	**6**
1	**2**	**3**

Table 4 – Example of a 3x3 domain with nodal numeration

$$A=\begin{bmatrix} a_{P1} & a_{E1} & 0 & a_{N1} & 0 & 0 & 0 & 0 & 0 \\ a_{W2} & a_{P2} & a_{E2} & 0 & a_{N2} & 0 & 0 & 0 & 0 \\ 0 & a_{W3} & a_{P3} & 0 & 0 & a_{N3} & 0 & 0 & 0 \\ a_{S4} & 0 & 0 & a_{P4} & a_{E4} & 0 & a_{N4} & 0 & 0 \\ 0 & a_{S5} & 0 & a_{W5} & a_{P5} & a_{E5} & 0 & a_{N5} & 0 \\ 0 & 0 & a_{S6} & 0 & a_{W6} & a_{P6} & 0 & 0 & a_{N6} \\ 0 & 0 & 0 & a_{S7} & 0 & 0 & a_{P7} & a_{E7} & 0 \\ 0 & 0 & 0 & 0 & a_{S8} & 0 & a_{W8} & a_{P8} & a_{E8} \\ 0 & 0 & 0 & 0 & 0 & a_{S9} & 0 & a_{W9} & a_{P9} \end{bmatrix} \quad (2.20)$$

As we can see in (2.20), the A matrix dimension increases with the square of the number of elements resulting in an enormous matrix, specially for very dense meshes, where most of the values are 0, except for the 5 diagonals for the coefficients that relate a node with its direct neighbours and itself. Solving this system in a classical approach where $x = A^{-1} \cdot b$ becomes almost impossible needing a new approach, the mathematical algorithms called solvers.

There are two type of solvers:

- Direct. The solver follows a mathematical algorithm that directly returns the exact (machine precision around 1e-18) solution to the system.
- Iterative. Iterative methods start from an initial value of *x* (random or fixed by us) and iterate and it is compared with the previous result until the difference achieves the desired precision.

In this project two different iterative methods have been programmed.

2.3.1 Gauss-Seidel (G-S)

This solver starts with an initial value of *x* that can be 0 or introduced by us. It iterates and computes a new value for *x*. If $A \cdot x - b > \varepsilon$ (our convergence parameter) then it makes another iteration with the new values of *x* until the criteria is satisfied. Convergence is only guaranteed if the matrix is either diagonally dominant, or symmetric and positive definite. This solver allows to solve each node in an independent way allowing this solver to be used in parallel programming. It is also quite easy to program despite not being one of the fastest. For more information on the G-S method refer to **[8]**.

If $A \cdot \phi = b$ with the already mentioned nomenclature for nodes and A matrix, where the sub index P is for the node being solved, W stands for west, E for east, N for north and S for south.

repeat

for(P=1 to Nt)

$$\phi_P = \frac{(a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + b_P)}{a_P}$$

if $b_P - (a_P\phi_P + a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S) < \varepsilon$ for all nodes

exit

end repeat

2.3.2 Conjugate Gradient (CG)

As most iterative methods it starts with an initial value of *x*, normally 0. The method uses a series of matrix-vector products and conjugates in order to find the direction in which our system decreases more quickly. The program repeats this process until convergence criteria is found. Note that the CG is only valid for matrixes symmetric and positive definite. The CG is faster than the G-S but more limited due to the need of a symmetric matrix and also is sensible to most small perturbation. For a more in depth view on the Conjugate Gradient check **[5]**.

If $A \cdot \phi = b$ and ϕ_0 is known or guessed as 0. Note that sub index denotes iteration number.

$r_0 = b - A\phi_0$

$p_0 = r_0$

k = 0

repeat

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

$$\phi_{k+1} = \phi_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k p_k$$

If $(r_{k+1} < \varepsilon)$ for all nodes

Exit

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

k = k+1

end repeat

2.4 Boundary Conditions

When working with differential equations we always encounter systems that have an infinite number of possible solutions that satisfy the equations. In order to obtain the solution we want and adjusts to our case of study it is necessary to fix certain conditions. Those are known as Boundary Conditions (BC).

2.4.1 Types of BC

The most common conditions in CFD and those we have used in our project are:

- *Dirichlet* (DBC). It is a direct condition. A certain variable ϕ has a fixed value on the boundary node.

$$\phi = k \qquad (2.21)$$

  Generally to fix pressure or temperature at a certain point or velocity at the inlet. It is also used to fix the no-slip condition in walls, forcing the velocity to be k=0.

- *Neumann* (NBC). It is an indirect condition. Instead of fixing the value of a certain node, the derivate of a variable ϕ with respect to the normal direction of the boundary node is known.

$$\frac{\partial \phi}{\partial n} = k \qquad (2.22)$$

  The most common use is to fix a certain derivate to k=0 to isolate a boundary or smooth

  Transitions between flow and walls. In the case of $\phi = T$ we would be fixing an adiabatic boundary.

- *Convective Boundary Condition* (CBC). It is a specification of a linear combination of the values of a function and the values of its derivative on the boundary of the domain, they are a weighted combination of Dirichlet boundary conditions and Neumann boundary conditions.

$$\frac{\partial U_i}{\partial t} + U_c \frac{\partial U_i}{\partial x} = 0 \qquad U_c = U_\infty \qquad (2.23)$$

The CBC is used mainly in outlets to prevent numerical instabilities with diverging results. For information about the CBC can be found in **[16]**.

2.4.2 Implementation of the BC

Boundary Conditions are easy to describe but not always as easy to implement within our code, here we well explain the basics on how to implement the previously mentioned BC.

- *Dirichlet*. They are the easiest to implement as we simply force a certain value to a variable, for example: u = 0.

  But if the variable we want to implement is not computed individually and it is obtained using one of the solver mentioned on the previous sections, then the condition is fixed inside the A matrix coefficients and would be as follows:

$$DBC \begin{cases} a_E = 0 \\ a_W = 0 \\ a_N = 0 \\ a_S = 0 \\ a_P = 1 \\ b_P = k \end{cases} \qquad (2.24)$$

- *Neumann*. If what we want is for example a no-slip wall in the south wall of our domain we would simply force $u_P = u_N$.
  As happened with the DBC, if the variable is computed with a solver we need to fix the coefficients inside the A matrix for our boundary node. Again the example is for a south wall of any variable used in the solver.

$$NBC \begin{cases} a_E = 0 \\ a_W = 0 \\ a_N = 1 \\ a_S = 0 \\ a_P = 1 \\ b_P = 0 \end{cases} \qquad (2.25)$$

- *Convective Boundary Condition*. We have used this condition only in the outlet of our simulated wind tunnel to fix the velocity at the exit wall. Sub index denote positon and super index denote time step, where *N* is the outlet and *n* the current time step.

$$U_N^{n+1} = U_N^n - \frac{\Delta t}{(\Delta x)_N} U_c (U_N^n - U_{N-1}^n) \qquad (2.26)$$

2.5 Fractional Step Method (FSM)

The Fractional Step Method can be interpreted as a projection into a divergence-free velocity space. The *predictor velocity*, is an approximate solution of the momentum equations, but because the predictor velocity is obtained with no pressure gradient contribution, it cannot satisfy the incompressibility constraint at the next time step. The Poisson equation determines the minimum perturbation that will make the predictor velocity incompressible.

2.5.1 Implementation of the FSM

Starting from the N-S equations adimensionalized and in its vectorial differential form:

$$\nabla \cdot \vec{u} = 0 \qquad (2.27)$$

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = -\nabla p + \frac{1}{Re}\Delta\vec{u} \qquad (2.28)$$

For a more in depth and detailed view on the deduction of the FSM refer to **[9]**, specially on the Helmholt-Hodge theorem.

Using the Helmholtz-Hodge decomposition theorem we project the equation (2.28) into a divergence-free velocity space. The theorem ensures that this decomposition is unique and after several projections we obtain the Poisson equation for pressure:

$$\Delta p = \nabla \cdot \left(-(\vec{u} \cdot \nabla)\vec{u} + \frac{1}{Re}\Delta\vec{u}\right) \qquad (2.29)$$

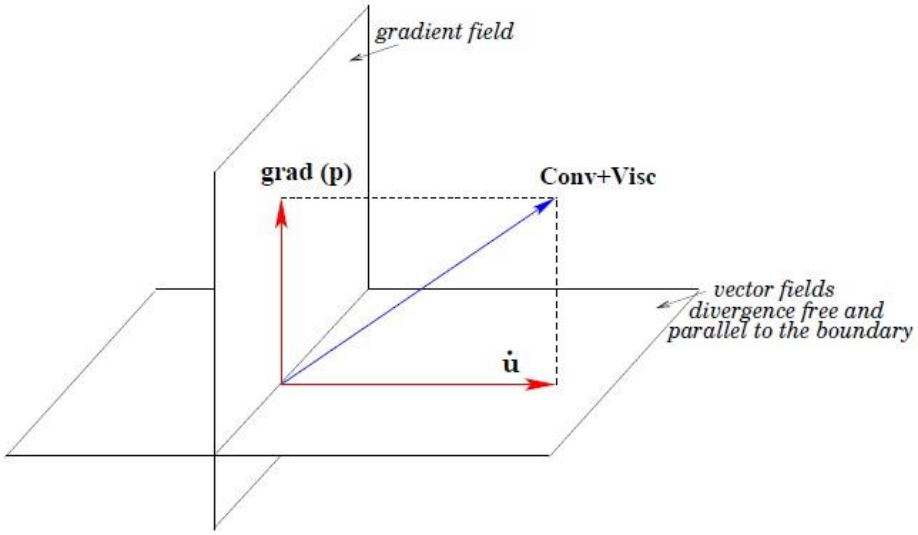


Fig. 14 – Convective + viscous term vector field decomposition, extracted from [9]

To simplify the notation, the momentum equation ca be rewritten as:

$$\frac{\partial \vec{u}}{\partial t} = R(\vec{u}) - \nabla p \qquad (2.30)$$

Where R(u) stands for the convective and diffusive terms

$$R(\vec{u}) = -(\vec{u} \cdot \nabla)\vec{u} + \frac{1}{Re}\Delta\vec{u} \qquad (2.31)$$

A fully explicit second-order Adams-Bashforth scheme is used for $R(\vec{u})$

$$R^{n+\frac{1}{2}}(u) \approx \frac{3}{2}R(u^n) - \frac{1}{2}R(u^{n-1}) \qquad (2.32)$$

Incompressibility constraint is treated implicitly. Thus, we obtain a semi-discretized Navier-Stokes equations

$$\frac{\mathrm{u}^{n+1} - \mathrm{u}^n}{\Delta t} = \frac{3}{2}R(u^n) - \frac{1}{2}R(u^{n-1}) - \nabla p^{n+1} \qquad (2.33)$$

To solve the velocity-pressure coupling we use the fractional step projection method. The solutions are obtained by first time-advancing the velocity field $\vec{u}$ without regard for its incompressibility constraint. Then, pressure gradient forces (projects) the predictor velocity field to be compressible ($\nabla \cdot u^{n+1} = 0$). This projection is derived from the Helmholtz-Hodge vector decomposition theorem, where the predictor velocity u^p can be uniquely decomposed into a divergence-free vector (u^{n+1}) and the gradient of a scalar field $\nabla\tilde{p}$. This decomposition is written as:

$$u^p = u^{n+1} + \nabla\tilde{p} \qquad (2.34)$$

The predictor velocity u^p is given by

$$u^p = u^n + \Delta t\left(\frac{3}{2}R(u^n) - \frac{1}{2}R(u^{n-1})\right) \qquad (2.35)$$

And the pseudo-pressure is $\tilde{p} = \Delta t p^{n+1}$. Taking the divergence of (2.34) we obtain the Poisson equation for $\tilde{p}$

$$\nabla \cdot u^p = \nabla \cdot u^{n+1} + \nabla \cdot (\nabla\tilde{p}) \qquad (2.36)$$

$$\Delta\tilde{p} = \nabla \cdot u^p \qquad (2.37)$$

Once the solution is obtained, we can correct the velocity u^p resulting in our values for u^{n+1}

$$u^{n+1} = u^p - \nabla\tilde{p} \qquad (2.38)$$

2.5.2 Final discretized equations

In section **2.2.3.1** and **2.2.3.2** we talked about discretization, if we combine that with the FSM and the equations we have obtained from it we obtain the complete and discrete equations to implement in our software.

Convection-Diffusion

$$\begin{aligned} R(\phi^n) = \Gamma_e\left(\frac{\phi_E^n - \phi_P^n}{d_{EP}}\right)\Delta y - \Gamma_w\left(\frac{\phi_P^n - \phi_W^n}{d_{PW}}\right)\Delta y + \Gamma_n\left(\frac{\phi_N^n - \phi_P^n}{d_{NP}}\right)\Delta x \\ - \Gamma_s\left(\frac{\phi_P^n - \phi_S^n}{d_{PS}}\right)\Delta x \\ - \left[\left(\rho\Delta y u_e \frac{1}{2}(\phi_E^n + \phi_P^n)\right) - \left(\rho\Delta y u_w \frac{1}{2}(\phi_W^n + \phi_P^n)\right)\right. \\ \left. + \left(\rho\Delta x u_n \frac{1}{2}(\phi_N^n + \phi_P^n)\right) - \left(\rho\Delta x u_s \frac{1}{2}(\phi_S^n + \phi_P^n)\right)\right] \end{aligned} \qquad (2.39)$$

Predictor velocity

$$u^p = u^n + \Delta t\left(\frac{3}{2}R(u^n) - \frac{1}{2}R(u^{n-1})\right) \qquad (2.35)$$

Poisson equation

$$\begin{aligned} \left(\frac{\tilde{p}_E - \tilde{p}_P}{d_{EP}}\right)\Delta y - \left(\frac{\tilde{p}_P - \tilde{p}_W}{d_{PW}}\right)\Delta y + \left(\frac{\tilde{p}_N - \tilde{p}_P}{d_{NP}}\right)\Delta x - \left(\frac{\tilde{p}_P - \tilde{p}_S}{d_{PS}}\right)\Delta x \\ = (\rho u_e^p \Delta y) - (\rho u_w^p \Delta y) + (\rho u_n^p \Delta x) - (\rho u_s^p \Delta x) \end{aligned} \qquad (2.40)$$

Reordering the equation (2.40) we obtain

$$\tilde{p}_P = \frac{(a_E\tilde{p}_E + a_W\tilde{p}_W + a_N\tilde{p}_N + a_S\tilde{p}_S + b_P)}{a_P} \qquad (2.41)$$

Where

$$\begin{gathered} a_E = \frac{\Delta y}{d_{EP}} \\ a_W = \frac{\Delta y}{d_{PW}} \\ a_N = \frac{\Delta x}{d_{NP}} \\ a_S = \frac{\Delta x}{d_{PS}} \\ a_P = a_E + a_W + a_N + a_S \\ b_P = \left(\rho u_e^p \Delta y\right) - \left(\rho u_w^p \Delta y\right) + \left(\rho u_n^p \Delta x\right) - \left(\rho u_s^p \Delta x\right) \end{gathered} \qquad (2.42)$$

Making it perfect to compute $\tilde{p}_P$ using one of the solvers from section **0**.

Velocity correction

$$u^{n+1} = u^p - \frac{1}{\rho}\left(\frac{\widetilde{P}_B - \widetilde{P}_A}{d_{BA}}\right) \qquad (2.43)$$

Due to the fact that the velocities are in the staggered meshes and the pressure in the centred one, $\tilde{P}_B$ is the pressure over the velocity we are evaluating (independently of which component of the velocity it is) and $\tilde{P}_A$ is the pressure under. In some sort of way we could say that for the horizontal component $\tilde{P}_A = \tilde{P}_W$, $\tilde{P}_B = \tilde{P}_E$ and for the vertical component $\tilde{P}_A = \tilde{P}_S$, $\tilde{P}_B = \tilde{P}_N$ since there is not a $\tilde{P}_P$ because the velocities are between two pressure nodes.

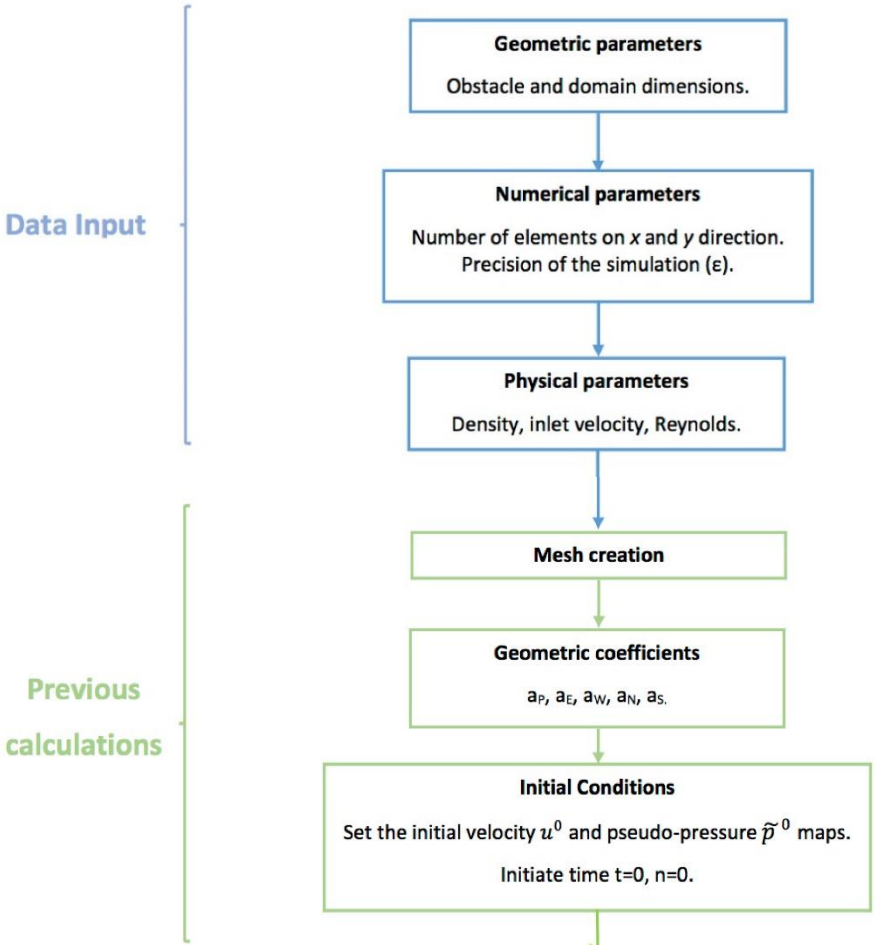
Continuity

$$\nabla \cdot \vec{u} = \left(\rho u_e \Delta y\right) - \left(\rho u_w \Delta y\right) + \left(\rho u_n \Delta x\right) - \left(\rho u_n \Delta x\right) \qquad (2.44)$$

Once the velocity correction has been done, we can check with the Continuity Equation that the divergence is 0 now.

2.5.3 Solving algorithm process

Now that we have explained all the part separately it is time to show how they are assembled together in order to work as a unit and solve the cases we want to study. The solving algorithm scheme is:



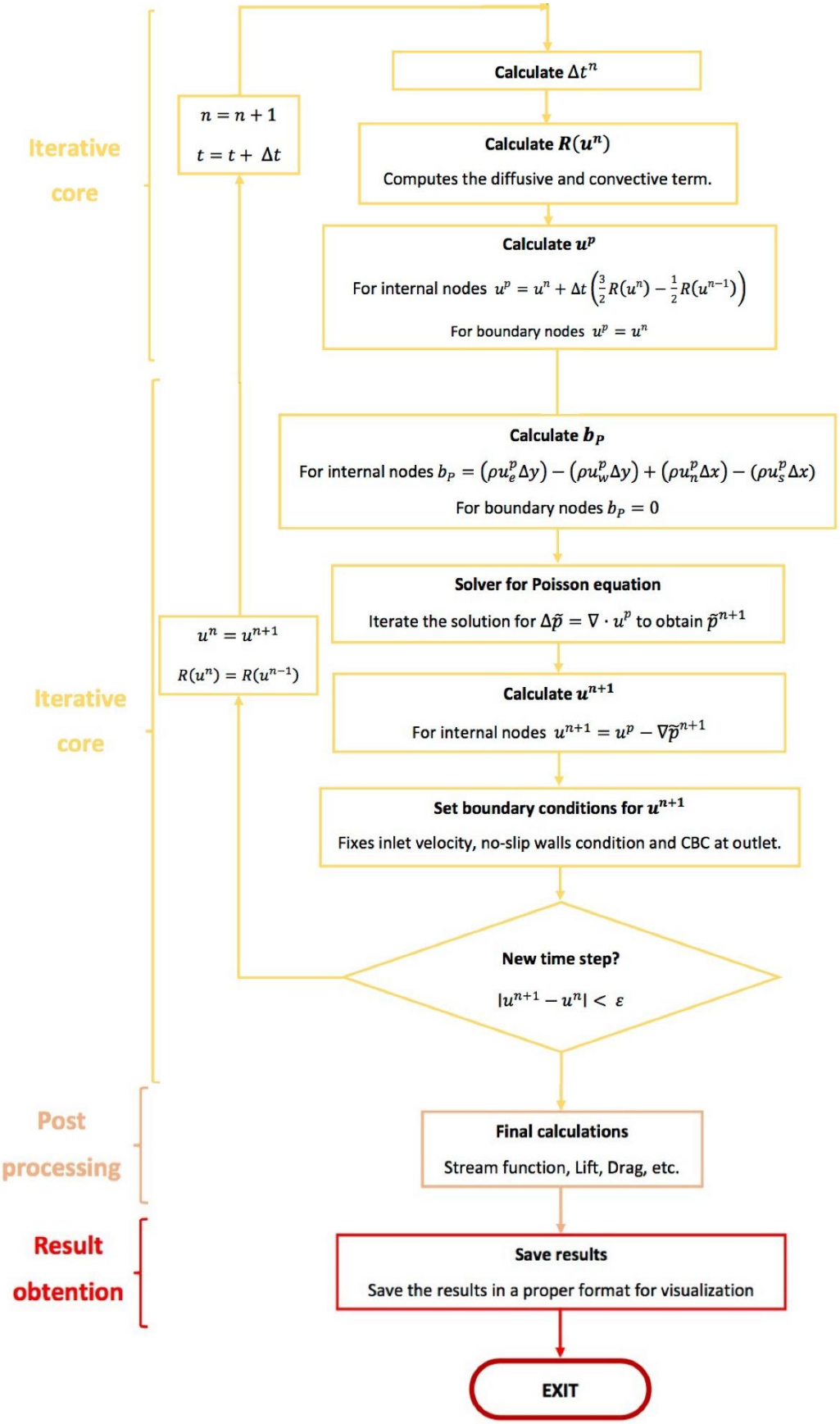


Fig. 15 – Solving algorithm scheme

3 Software verification

The most critical part when developing software is to ensure the program does exactly what it is supposed to. Each single part has to work perfectly for itself and the complete set of functions and interactions between them has no room for errors.

The difficulty of developing a software oriented in the computation of physics and engineering applications is that you have to ensure three things are done correctly:

- **Adequacy of the physical equations**. We have to verify the discretization is coherent and that we have properly modelled the governing equations of the problem of study.
- **Adequacy of the mathematical models**. Ensure all the mathematical operations we are performing are done properly independently of the physics behind those operations. For example, ensuring the derivate or the divergence is done properly for any given vector field.
- **Coding and language**. Personally, I like to see programming languages as any real language and because of this, the errors we do while coding are the same as we do while writing in any language. We have to ensure language is used correctly.

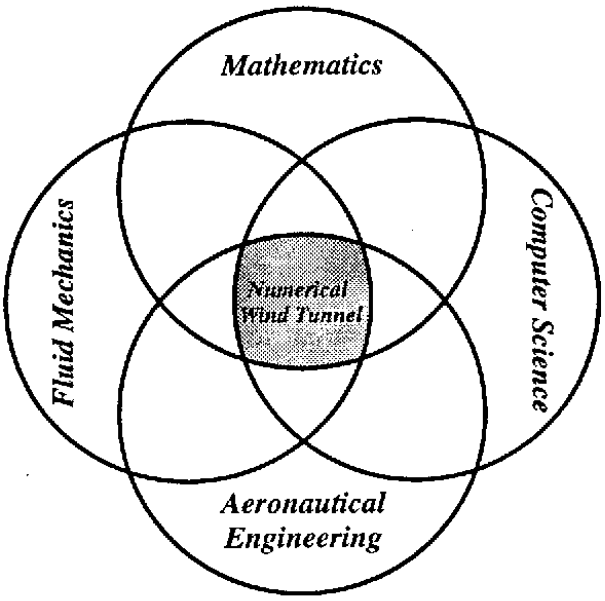


Fig. 16 - Venn diagram on CFD set of skils, extracted from [3]

In the following sections we will take a closer look on how the previous points are verified.

3.1 Code and language checking

As I have already mentioned, I think coding errors can be treated as any language errors so in order to study them, I have classified the type of errors in the same categories language errors are. Note that despite being extensible to any programming language I'll be focusing in C++, the language used for the coding of this project.

a) Lexico-semantic. It mainly consists in a mistaken use of a function or statement. The equivalent in language would be to mistake the words good and well.

   An example would be wanting to print something on screen and using the command *fprintf* instead of *printf*. One can properly write the *fprintf* function but not knowing it writes on a file instead of on screen as desired resulting in the code differing from what we wanted. This kind of errors are usually not detected by the compiler but seen easily in the results or by a quick debugging.

b) Morphologic. These can be one of the hardest errors to find while coding. An equivalent in language would be to use the wrong gender, like using he instead of she. I mainly found two type of morphologic errors.

   The first one is, for example, when we have a given function that has one double as input and we introduce a vector of doubles as the input. It can seem that the function should be able to understand that we want to execute it for every double stored in our vector but because of how C++ works, the compiler would detect an error in the input format. Other programming languages like Python or matlab do not have this kind of morphological errors.

   The second type is the one that can be rally tricky and hard to find. It is the fact that C++ has some libraries included for mathematical functions and those functions are prepared to work both on integers (*int*) and for decimals (*float, long float* or *double*). For example, if we want to compute the hyperbolic tangent we would use the function *tanh*. But if as input we give for example *tanh(a/b)* where *a* is an integer and *b* a double, the code would divided *a* between *b* and round it to the closest integer and then perform the *tanh*. We lose total control of what number the code is performing the *tanh* and we end up with a different result from what we wanted. This code is not detected by the compiler and if we are performing a lot of operations it can be really hard to see where it was that messed up our calculations. Intense debugging step by step is probably the best tool against those errors.

c) Syntactic. These errors come from an inappropriate grammatical structure. An equivalent in language would be to not invert the verb and the subject in a question.

An example would be to not enter all the needed inputs of a certain function, not writing properly all the statements in a *for* loop or entering as input a *char* when an *int* or *double* is needed. The compiler itself detects all these kind of errors.

d) Orthographic. While programming these errors are generally a forgotten character or typing errors. As the name itself says, the equivalent in language would be to write making orthographical errors.

The most common of these mistakes are to forget to write *;* at the end of a command line ore mistype errors such as writing *nit* instead of *int*.

3.2 Method of Manufactured Solutions (MMS)

The Method of Manufactured Solutions is the perfect tool to verify the mathematical operations that we have programmed, verification is purely a mathematical exercise. It is based in comparing the results obtained by our code with analytical generated ones. For more detailed information on the MMS refer to **[10]**.

We take a mathematical operator that we have implemented in our code and we test it by itself, separated from any other part of the program. Now we are just going to treat the operator as a black box, we give it a certain input and we get an output. The input is an analytical function of our choice who's analytical exact result (output) is known. We compare the analytic output with the one produced by our code to check if our operator is good enough.

Generally, we repeat the test for the same operator with different mesh densities. The error of the results with the discretization used for our program is of a second order, meaning that the error reduces with the square of the mesh density. One exception would be the Guass-Seidel or Conjugate Gradient method whose precision does not depend so much on the mesh but on the precision we fix for the iterative method itself. Using this method we tested some of our core mathematical operations such as the gradient, the divergence or the Gauss-Seidel for solving A·x = b.

For example, we tested the divergence with an analytic velocity field known as the Taylor-Green vortex, detailed in **[6]**. The Taylor-Green vortex velocity field is:

$$u = \cos x \sin y \qquad (3.1)$$

$$v = \cos y \sin x \qquad (3.2)$$

And the divergence of this field is known to be 0 ($\nabla \cdot \mathrm{u} = 0$). So we introduced the velocities in (3.1) and (3.2)into our code and computed it (as mentioned in equation (2.44)) for several mesh sizes. Successfully, the error of our solution decreases with the square of the mesh size, converging towards zero.

3.3 Comparison with practical cases

Once we have ensured that our code is out of language errors and we verified that our mathematical models perform the operations properly with independence of the physics or the input we gave them, it comes down to check the physics we have modelled, see if we can reproduce any real cases with our code. In order to do so, we tested our code as we were building it, from more simple cases to more complex ones.

The cases studied are:

- **Smith-Hutton problem** or also known as **Solenoidal flow**. A 2D convection-diffusion problem.
- **Driven Cavity problem**. A very simple 2D cases that includes can be described with the Navier-Stokes model shown in section 2 and goes throw all the steps described in the Fractional Step Method.

3.3.1 Smith-Hutton problem

The aim of this problem is to obtain the steady state solution of the two-dimensional convection-diffusion equation.

$$\rho\frac{\partial\phi}{\partial t}+\rho u\frac{\partial\phi}{\partial x}+\ \rho v\frac{\partial\phi}{\partial y}=\ \Gamma\left(\frac{\partial^2\phi}{\partial x^2}+\ \frac{\partial^2\phi}{\partial y^2}\right)+S \qquad (2.8)$$

Discretized as shown in Chapter 2:

$$\begin{aligned}R(\phi^n)=\ &\Gamma_e\left(\frac{\phi_E^n-\phi_P^n}{d_{EP}}\right)\Delta y-\Gamma_w\left(\frac{\phi_P^n-\phi_W^n}{d_{PW}}\right)\Delta y+\Gamma_n\left(\frac{\phi_N^n-\phi_P^n}{d_{NP}}\right)\Delta x\\&-\Gamma_s\left(\frac{\phi_P^n-\phi_S^n}{d_{PS}}\right)\Delta x\\&-\left[\left(\rho\Delta y u_e\frac{1}{2}(\phi_E^n+\phi_P^n)\right)-\left(\rho\Delta y u_w\frac{1}{2}(\phi_W^n+\phi_P^n)\right)\right.\\&\left.+\left(\rho\Delta x u_n\frac{1}{2}(\phi_N^n+\phi_P^n)\right)-\left(\rho\Delta x u_s\frac{1}{2}(\phi_S^n+\phi_P^n)\right)\right]\end{aligned} \qquad (2.39)$$

The problem consists in a rectangular domain where the bottom side of the rectangle is divided in 2 parts, the left part is considered the inlet and the right part is the outlet.

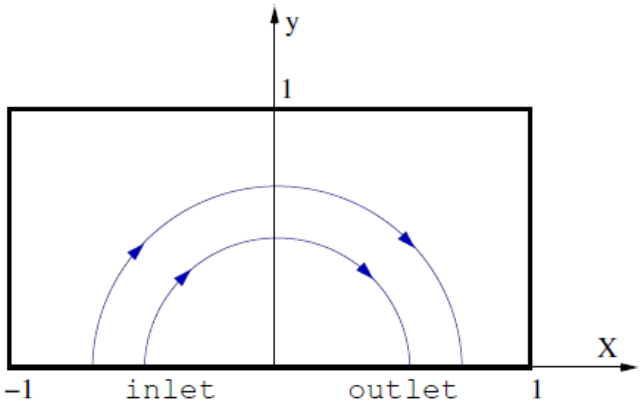


Fig. 17 -– Smith-Hutton domain scheme, extracted from [12]

Through all the domain, the velocity field is prescribed and given by

$$u(x,y) = 2y(1-x^2) \qquad (3.3)$$

$$v(x,y) = -2x(1-y^2) \qquad (3.4)$$

And the following boundary conditions for the variable ϕ

$$\phi = 1 + \tanh\big(\alpha(2x+1)\big) \qquad y = 0;\ x \in (-1,0) \quad (inlet) \qquad (3.5)$$

$$\frac{\partial \phi}{\partial y} = 0 \qquad y = 0; x \in (0,1) \quad (outlet) \qquad (3.6)$$

$$\phi = 1 - \tanh(\alpha) \qquad (elsewhere) \qquad (3.7)$$

Once all the conditions are set, we compute the results with our software and compare them with the benchmark solution of reference **[12]** for different values of the ratio between ρ and Γ and using a structured uniform mesh of 200x100 elements.

	P/Γ = 10		**P/Γ = 1E3**		**P/Γ = 1E6**	
X-POSITION	Reference	Simulated	Reference	Simulated	Reference	Simulated
0,0	1,989	2,0000	2,0000	2,0000	2,000	2,0000
0,1	1,402	1,0000	1,9990	2,0000	2,000	2,0000
0,2	1,146	1,1462	1,9997	1,9998	2,000	2,0000
0,3	0,946	0,9465	1,9850	1,9920	1,999	2,0000
0,4	0,775	0,7749	1,8410	1,8262	1,964	1,9432
0,5	0,621	0,6210	0,9510	0,96668	1,000	0,9983
0,6	0,480	0,4800	0,1540	0,1471	0,036	0,0426
0,7	0,349	0,3493	0,0010	0,0058	0,001	0,0075
0,8	0,227	0,2272	0,0000	0,0001	0,000	0,0000
0,9	0,111	0,1117	0,0000	0,0000	0,000	0,0000
1,0	0,000	0,0000	0,0000	0,0000	0,000	0,0000

Table 5 – Result comparison of the Smith-Hutton problem

We can see that the results are very similar between the ones we simulated and the reference ones, despite that we see the higher the ratio ρ/Γ the bigger the difference gets. This can be in part due to the mesh used for the high ratio simulations not being dense enough but the difference is very small and what is more important, the behaviour of the variable ϕ is the same. We consider the modelling of the Convection-Diffusion equation implemented in our software valid to move forward to the next more complex cases that base everything in the computing of that equation.

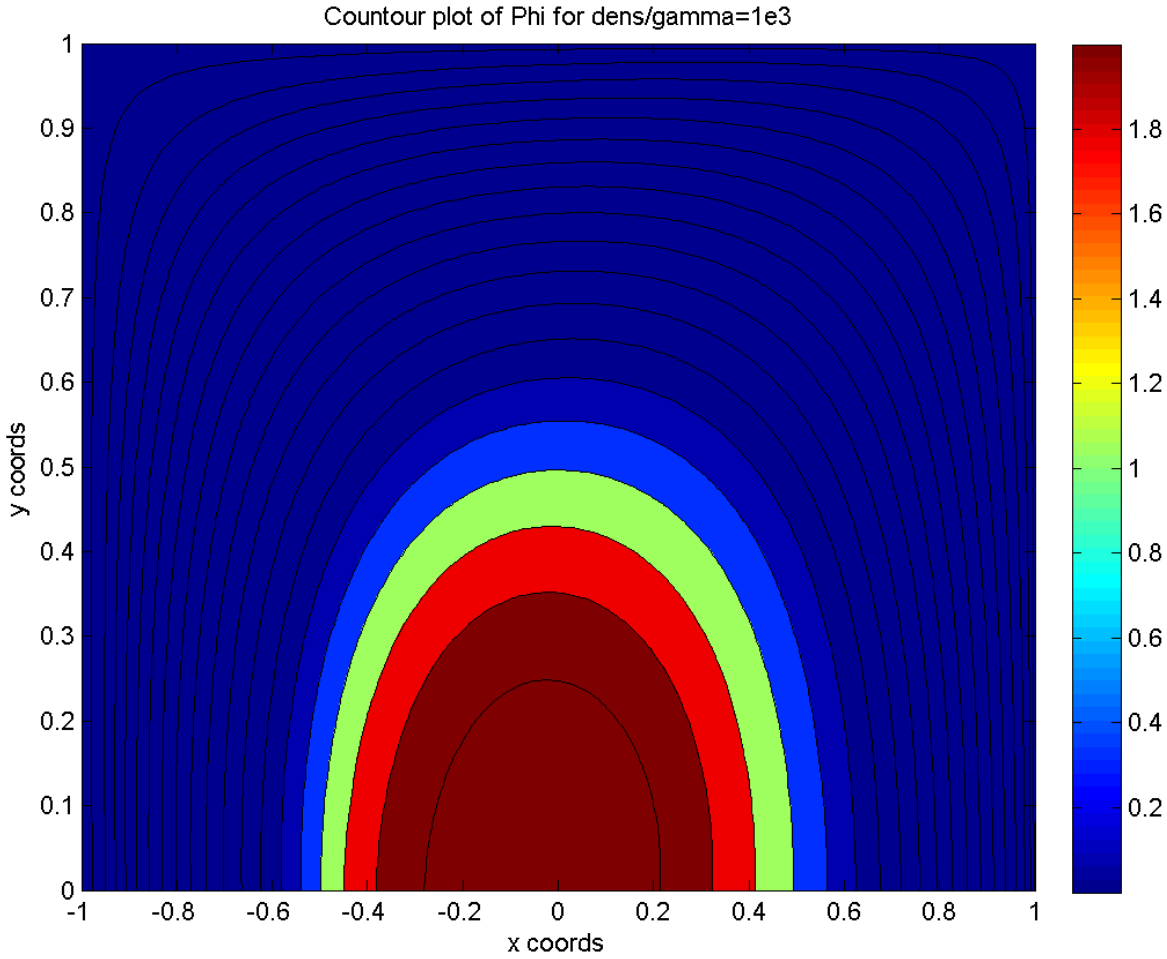


Fig. 18 – Contour plot of ϕ for ρ/Γ = 1e3.

3.3.2 Driven Cavity Problem

The main objective of this problem is to ensure we have modelled properly the Navier-Stokes Equations and we have developed a software capable of solving them with precision.

The equations to solve in its adimensionalized and vectorial form are

$$\nabla \cdot \vec{u} = 0 \qquad (2.27)$$

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = -\nabla p + \frac{1}{Re}\Delta \vec{u} \qquad (2.28)$$

They have already been shown and explained in Chapter **2**. In this problem we will fully implement the Fractional Step Method as seen in section **2.5.2**, getting into the pressure-velocity coupling problem.

3.3.2.1 Geometry

The cases consists in a square cavity with infinite width (allowing us to consider the case 2D) with a pressurized fluid contained in it and a moving wall. The north wall of the cavity is the one moving, with an horizontal velocity of $u_{wall} = u_\infty$ (which once adimensionalized becomes $u_{wall} = 1$ since we adimensionalize all velocities with u_∞) and the rest of the walls with no velocity.

Since it is a square cavity, the length and high have the same value and once adimensionalized, the *x* and *y* coordinate axes go from 0 to 1.

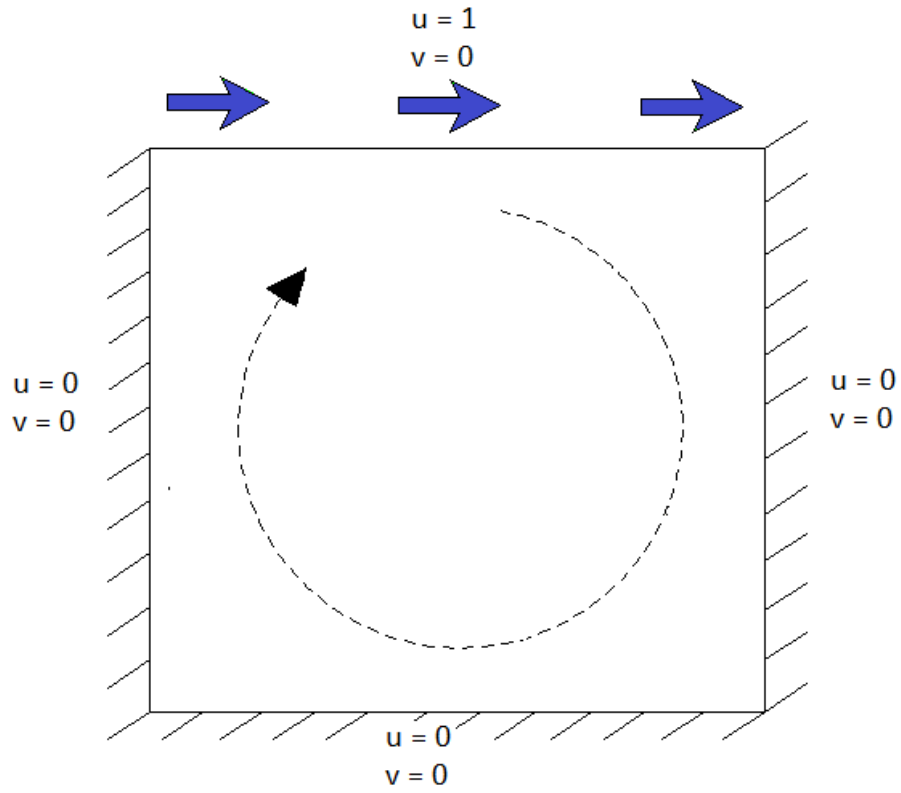


Fig. 19 – Driven Cavity domain scheme

3.3.2.2 Boundary Conditions

The velocity of the west, east and south wall is fixed to $u = 0,\ v = 0$ and the north wall has $u = 1,\ v = 0$.

For the first time we have introduced the pseudo-pressure so we also have to fix the boundary conditions for it. All the walls have Neumann Boundary Conditions in order to prevent abnormal pressure gradients on the walls with no physical sense due to the fixed velocities. Apart from that, since the system $A \cdot x = b$ has infinite possible solutions, we have to fix the pressure on a single point to a reasonable value for our simulation in order to work on the same pseudo-

pressure field conditions for different simulations, even if changing other parameters. The point chose in this case was the centre of the cavity with a fixed pseudo-pressure of $\tilde{p} = 0$.

3.3.2.3 Mesh

Since we know in advance the expected result for the Driven Cavity problem, we know the most sensible and critical points are the ones close to the boundaries, specially the corners where vortex appear. In order to have the precision necessary to study this effects and have as less elements as possible (within a compromise with quality of the results) to fasten the simulation we have decided to use a non-uniform structured mesh.

The mesh has more density around boundaries and less at the centre of the cavity. To achieve a mesh that has those properties we have used an hyperbolic tangent distribution as shown down in (3.8), which is an adaptation of the distribution used and described in **[13]**.

$$x = \frac{L}{2}\left(1 + \frac{\tanh\left(\gamma\left(\frac{2i}{Nx+1} - 1\right)\right)}{\tanh\gamma}\right) \qquad (3.8)$$

Where *x* is the horizontal coordinate of the node, *L* the length of the cavity, *i* the node horizontal position in the memory vector, *Nx* the number of elements on the horizontal direction and γ a non-physical parameter used to control density ($\gamma = 2$).

$$y = \frac{H}{2}\left(1 + \frac{\tanh\left(\gamma\left(\frac{2j}{Ny+1} - 1\right)\right)}{\tanh\gamma}\right) \qquad (3.9)$$

Where *y* is the vertical coordinate of the node, *H* the height of the cavity, *j* the node vertical position in the memory vector, *Ny* the number of elements on the vertical direction and γ a non-physical parameter used to control density ($\gamma = 2$).

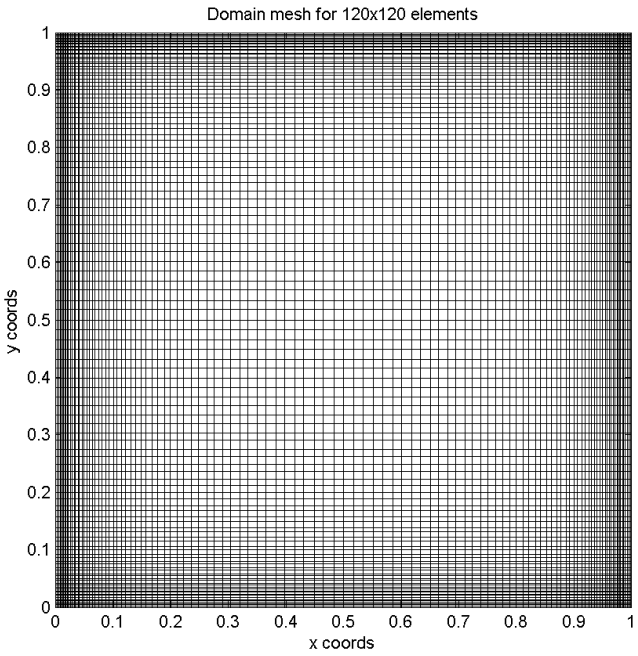


Fig. 20 – Mesh scheme for the Driven Cavity problem

3.3.2.4 Simulation parameters

The key input parameter for this simulations is the Reynolds number which determines the regimen of the flow. The Reynolds number for the Driven Cavity is characterized as

$$Re = \frac{\rho \cdot u_\infty \cdot L}{\mu} \qquad (3.10)$$

Where ρ is the density of the fluid, u_∞ the horizontal velocity of the north wall, *L* the length of the cavity and μ the dynamic viscosity of the fluid.

Throw all the simulations carried out three parameters have been maintained constant:

- The horizontal velocity of the north wall, $u_\infty = 1$.
- The length and high of the cavity, $L = H = 1$.
- The density of the fluid, $\rho = 1$.

The input parameter modified to vary the conditions of the experiment is the Reynolds number resulting in a dynamic viscosity of $\mu = \frac{1}{Re}$.

3.3.2.5 Result comparison

We have used as benchmark reference results the ones given from **[14]** in different simulations for the Reynolds values of 100, 400, 1000, 3200, 5000 and 7500. The results of reference have been obtained numerically and compared with data obtained from the same real physical experiment and considered as valid by the scientific community, even though not 100% accurate

since they have almost 30 years and the computational power was not even close to the one available right now. It is also discussed if the behaviour for Re=7500 is transitory or still stationary so we will not take those values as much as in consideration as the others.

The results presented by **[14]** are for the velocity field, separated for the horizontal and vertical ones. The horizontal velocity has been given along the *y* axis for $x = 0{,}5$ (through the Geometric Centre of the cavity) and the vertical velocity has been given along the *x* axis for $y = 0{,}5$ (through the Geometric Centre of the cavity).

We will present our results along the reference ones in the same graph to properly compare them and to have a much easier understanding of the data, which sometimes can be huge and overwhelming.

Re = 100

100x100 mesh

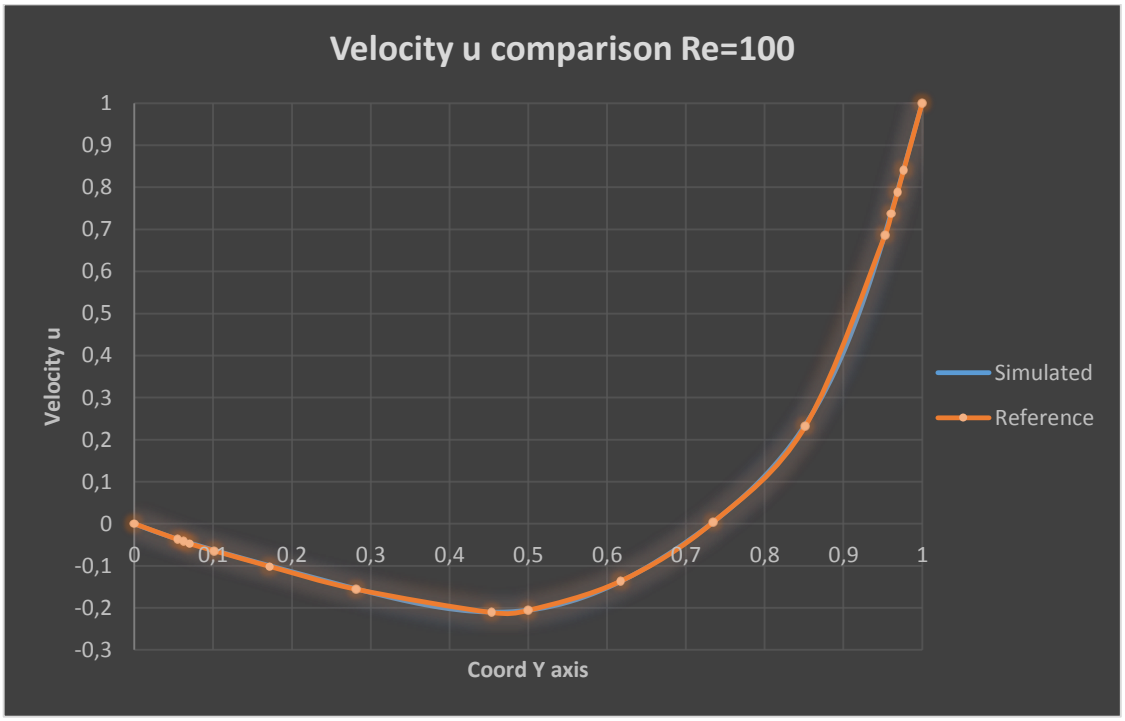


Fig. 21 – Velocity u comparison for Re = 100

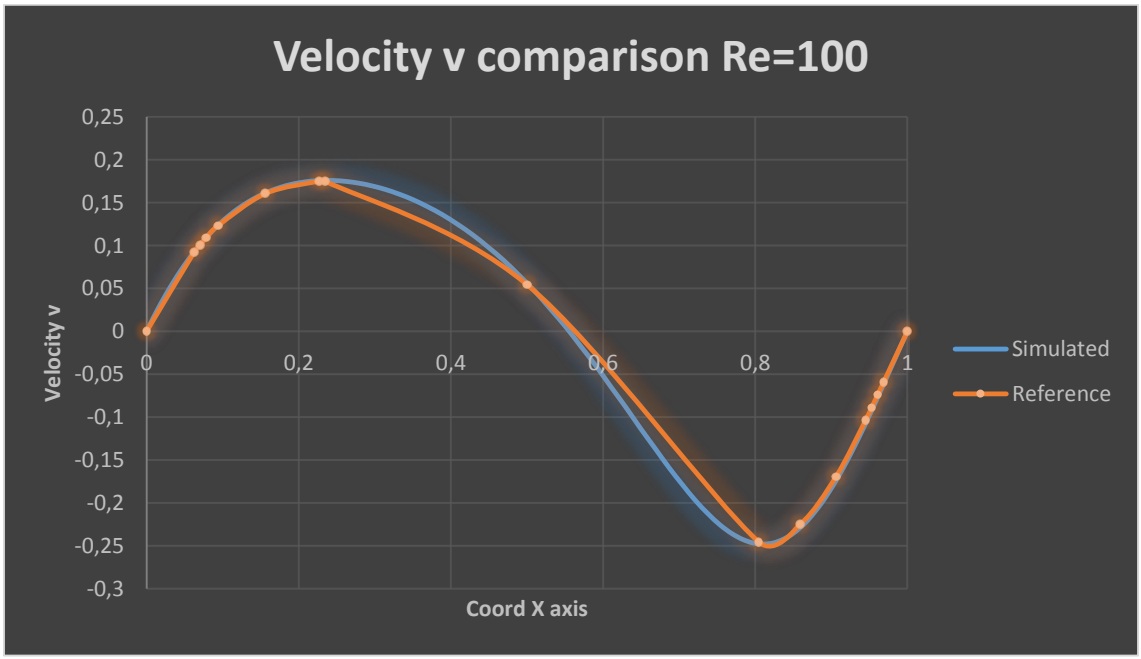


Fig. 22 – Velocity v comparison for Re = 100

Re = 400

100x100 mesh

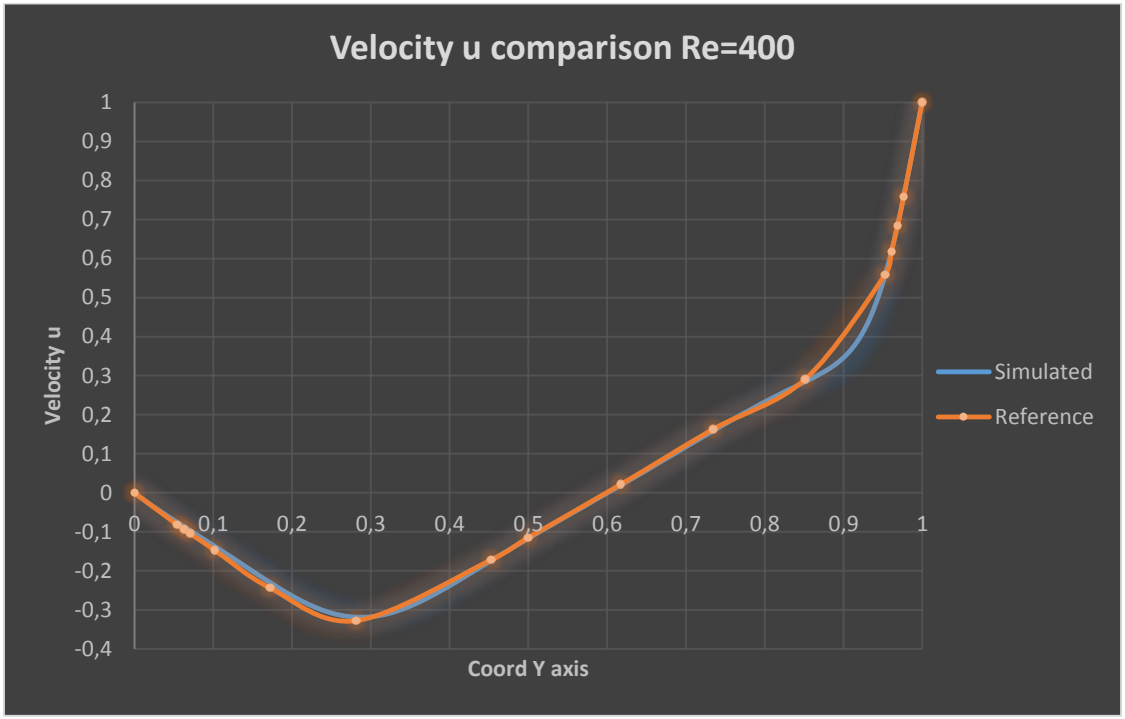


Fig. 23 – Velocity u comparison for Re = 400

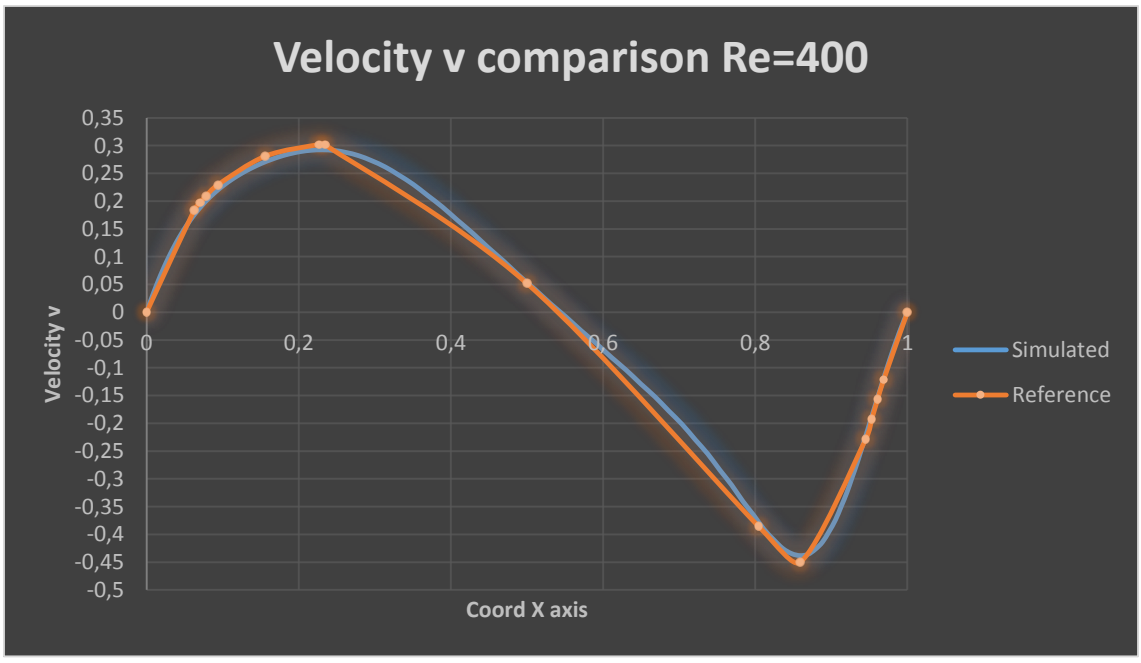


Fig. 24 – Velocity v comparison for Re = 400

Re = 1000

100x100 mesh

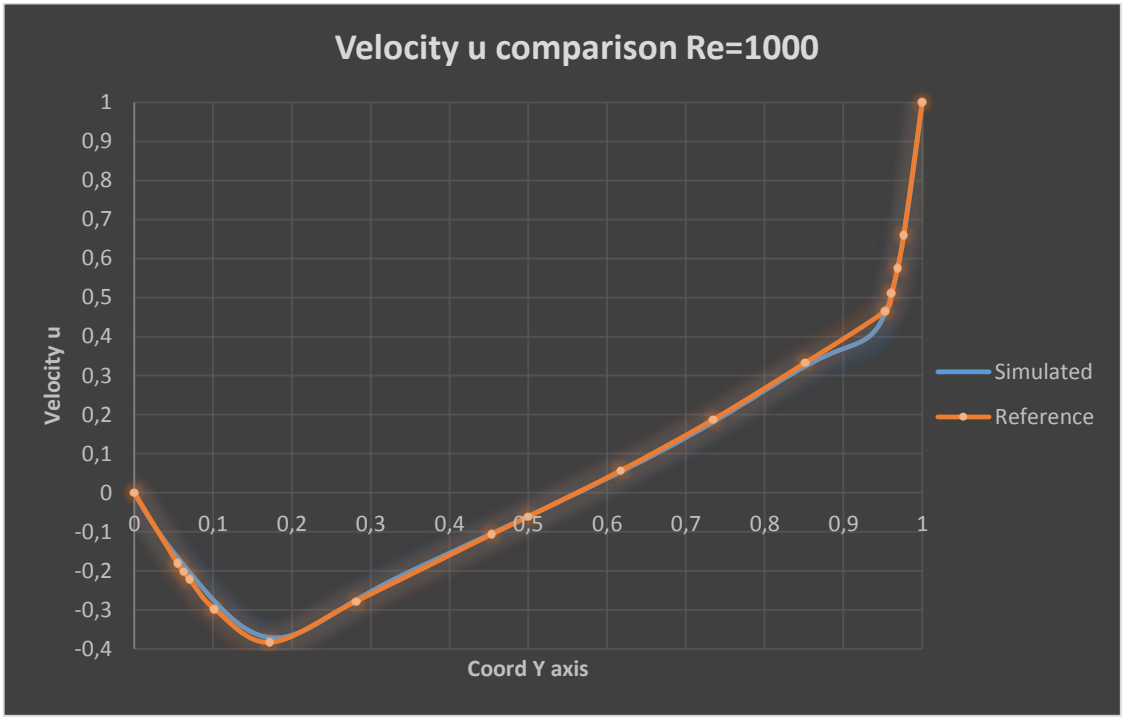


Fig. 25 – Velocity u comparison for Re = 1000

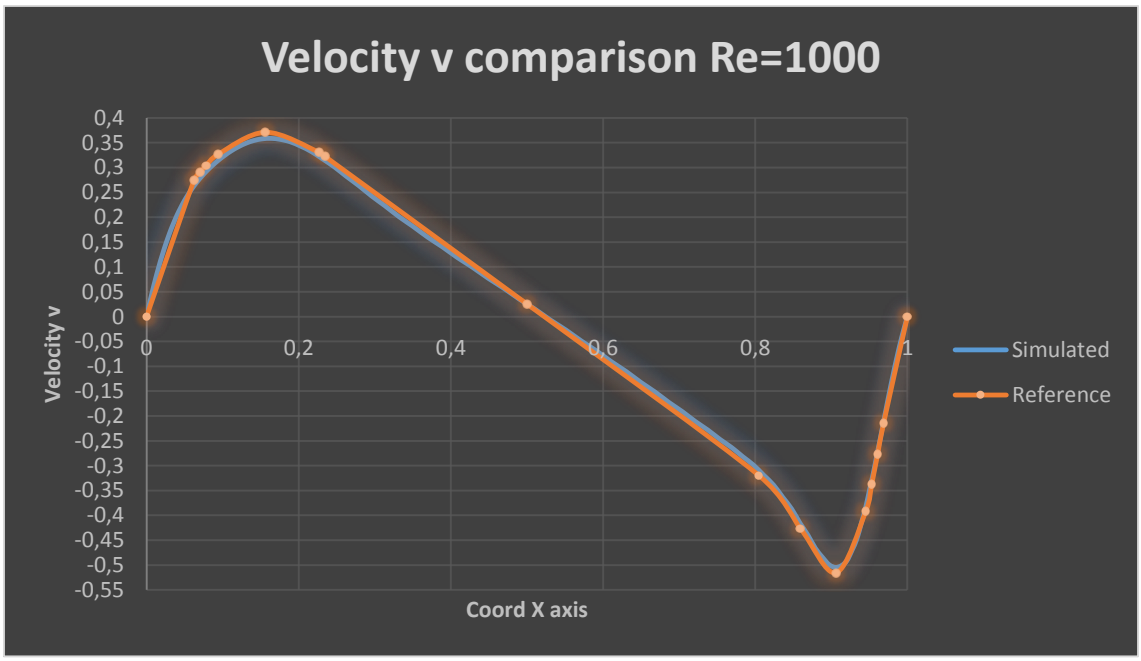


Fig. 26 – Velocity v comparison for Re = 1000

Re = 3200

100x100 mesh

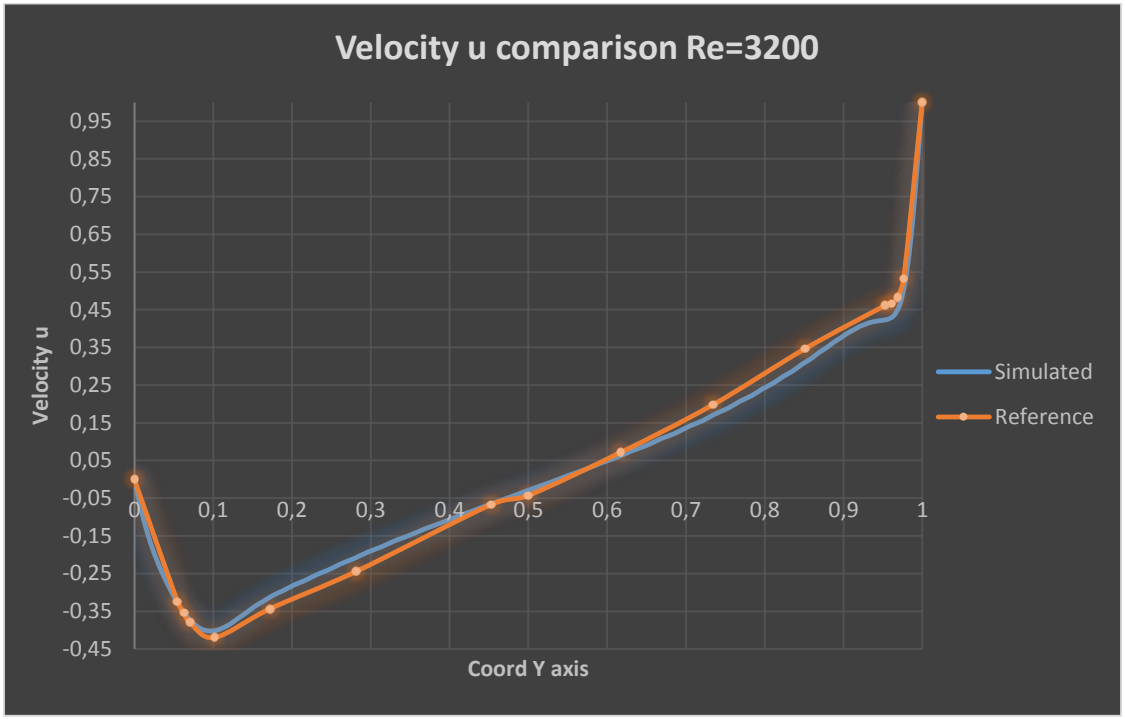


Fig. 27 – Velocity u comparison for Re = 3200

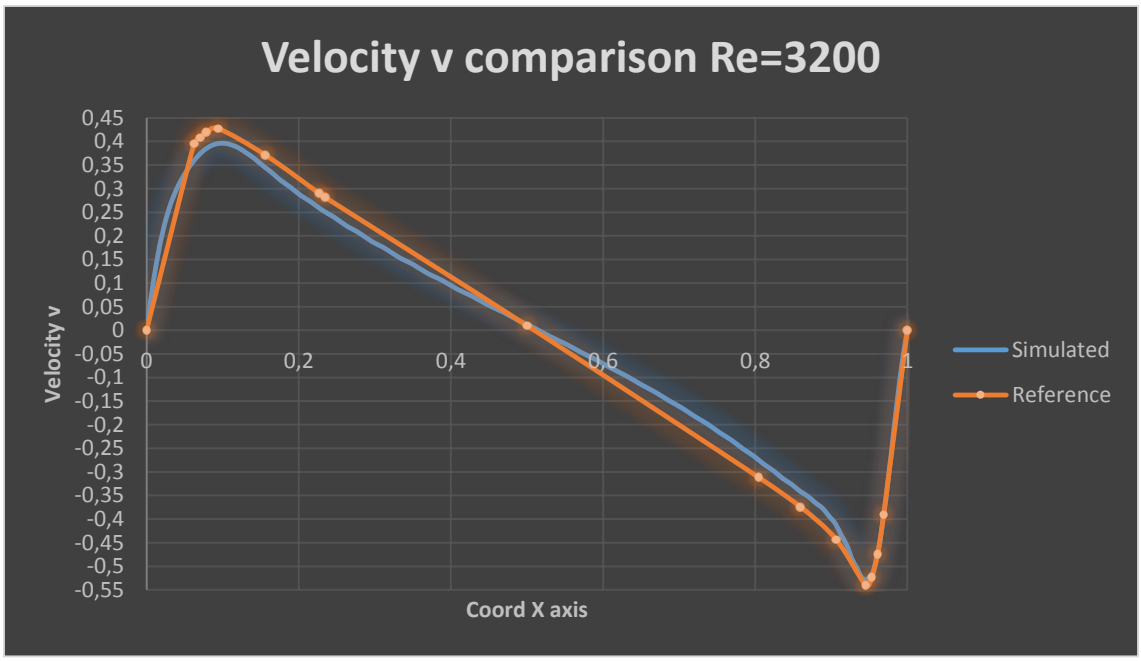


Fig. 28 – Velocity v comparison for Re = 3200

Re = 5000

120x120 mesh

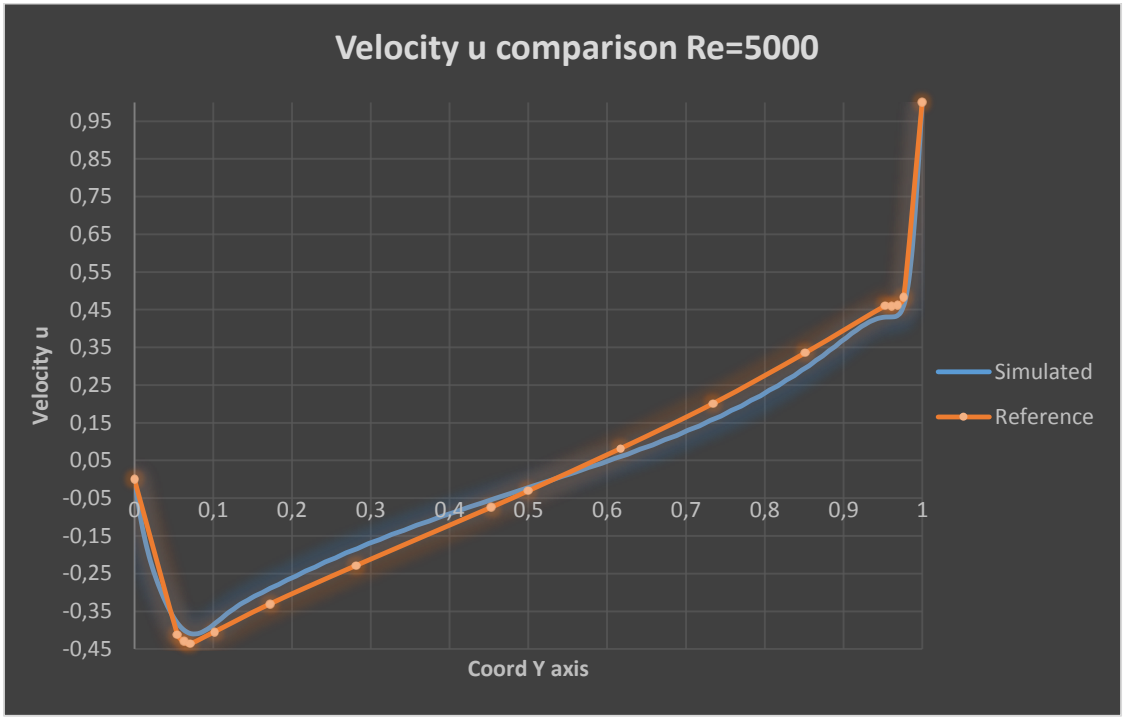


Fig. 29 – Velocity u comparison for Re = 5000

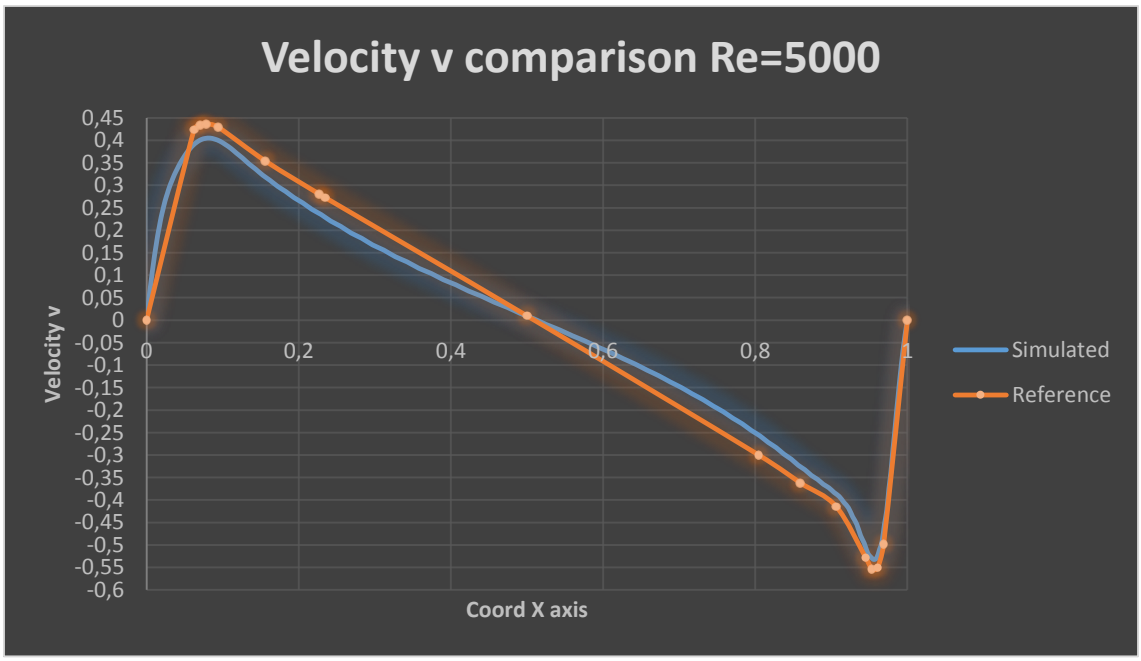


Fig. 30 – Velocity v comparison for Re = 5000

Re = 7500

120x120 mesh

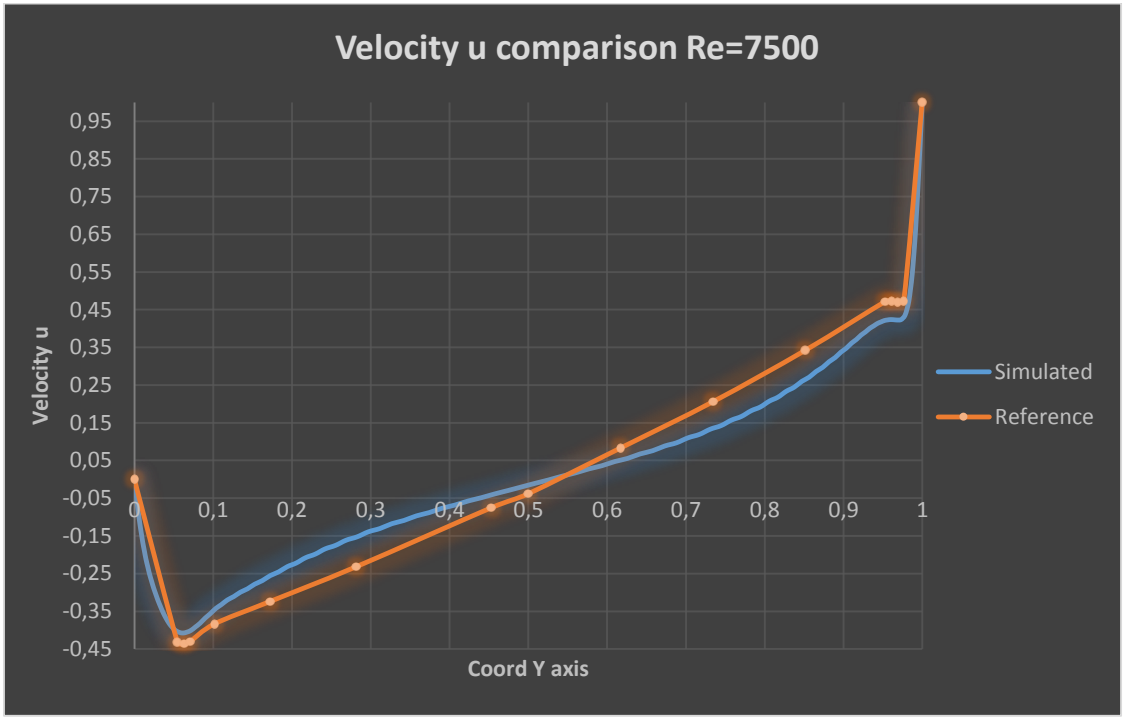


Fig. 31 – Velocity u comparison for Re = 7500

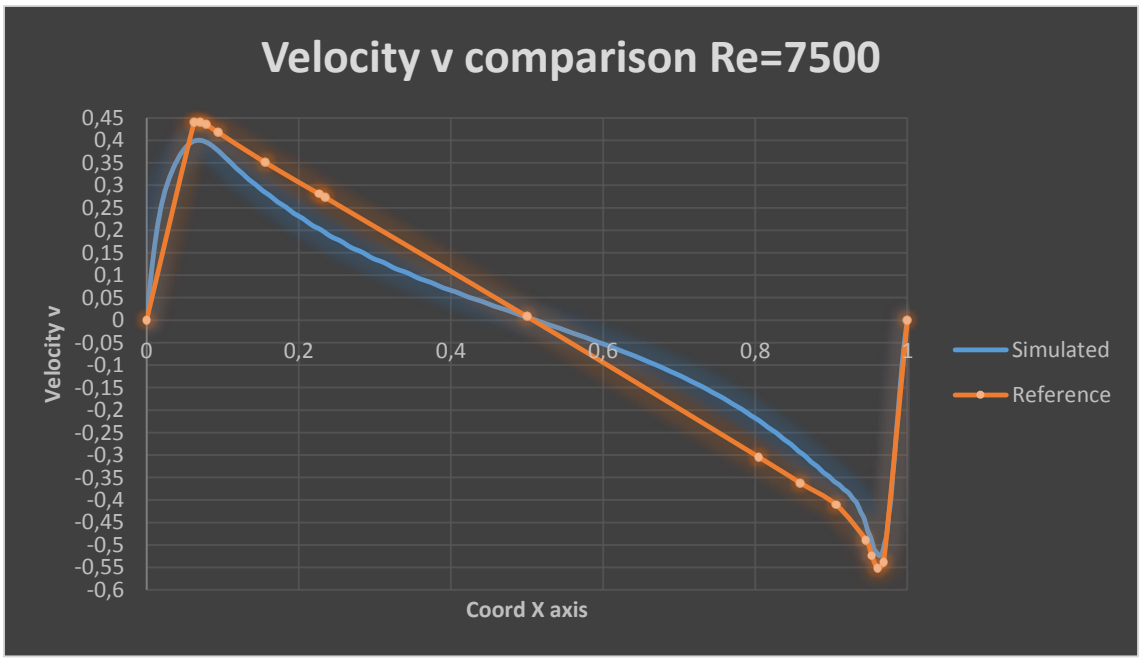


Fig. 32 – Velocity v comparison for Re = 7500

Result comparison analysis

As we can appreciate from Fig. 21 to Fig. 26, the results for Reynolds 100, 400 and 1000 for our simulations and the references ones match almost to perfection. Sometimes we can see a bit of divergence between the blue line (simulated) and the red one (reference) but this is due to the fact that we have a sample of 100 points while the reference results only gave a sample of 17 values that are marked with a red dot. It is on that red dot that the values have to coincide, and they do so.

As we start to considerably increase the Reynolds number we start to appreciate a small separation between the reference values and the ones produced by our software. For Reynolds 3200 and 5000 the difference is appreciable but still within our margin of error. Taking into consideration that this is a self-built software programming by a student engineer, compared to the one developed by Ph.D. professors with a considerable experience in the field of CFD, we assume their software to be much more powerful and use better mathematical algorithms.

In order to achieve a bit more precision we could have used a more densified mesh but the computational time probably wouldn't pay off the increase of accuracy. We have already proved that our software can describe with excellent precision laminar cases with low Reynolds numbers which is the main objective of this project, since for most geometries do not need values as high as Re=5000 to be turbulent. For example, our case of study known as the Square Cylinder described in Chapter **4** reaches turbulent flow at values around Re=300.

The results for Re=7200 show significant differences between the reference and the simulated ones. As we already mentioned, it is not absolutely clear if the driven cavity for Reynolds of 7500 is still stationary or is already transitory, one of the most extended opinions is that it is transitory but changes very slowly in time. Meaning that since our software tries to make in converge to a stationary case as the reference study, depending on the convergence criteria or precision we could end up considering it stationary at different conditions or instants of time for a slow transitory case. This would mean that even if we are studying properly the same case the instant of time at which we evaluated our variables is different, showing a different result but being both of them correct.

An analogy to make it more understandable would be to compare the waves of the sea in a very calm day. Since the peak of the waves is very small, it is hard to appreciate the differences and it seems like the sea is not moving, but it simply is moving very slow. If two different persons took a picture of the sea at different instants, there would be a difference between them but actually both of them would be pictures of the same sea, same conditions and same day.

Now that we have ensured the quality and veracity of our results, we can examine them alone and analyse the behaviour of the flow inside the Driven Cavity.

Detailed results and analysis

Re = 100

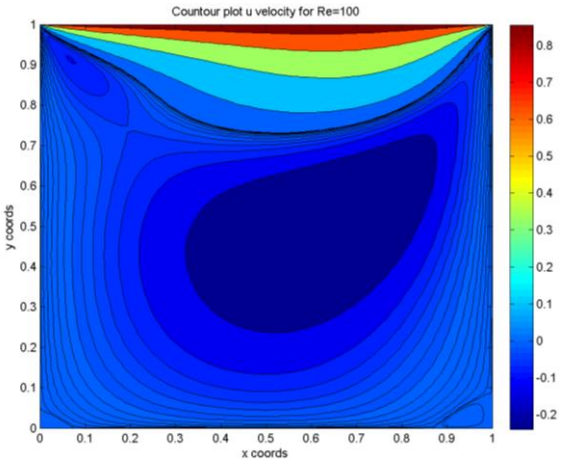


Fig. 33 - Contour plot u velocity for Re=100

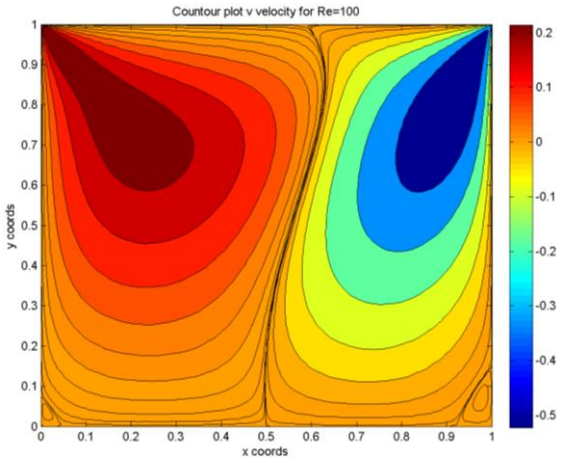


Fig. 34 - Contour plot v velocity for Re=100

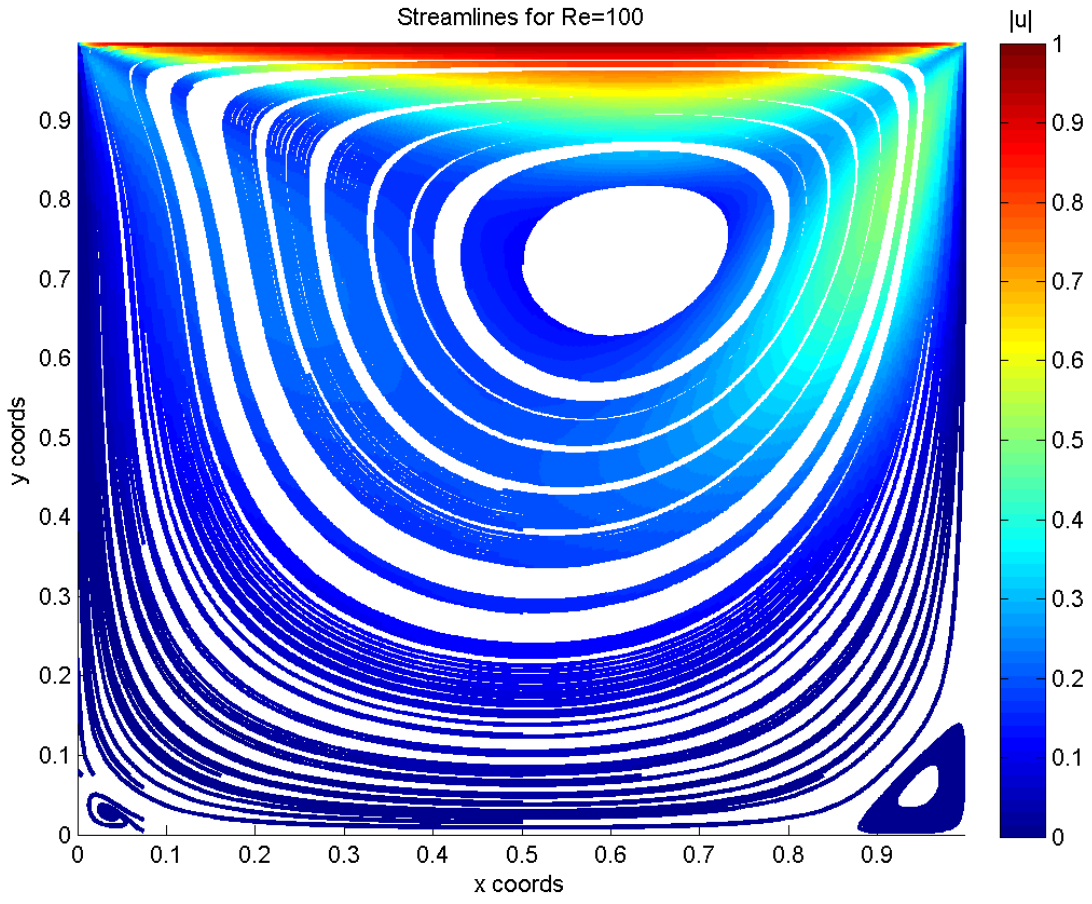


Fig. 35 - Streamline for Re=100

Re = 1000

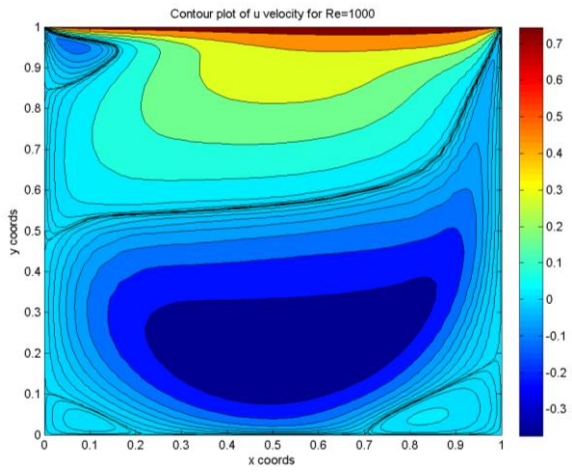


Fig. 36 - Contour plot u velocity for Re=1000

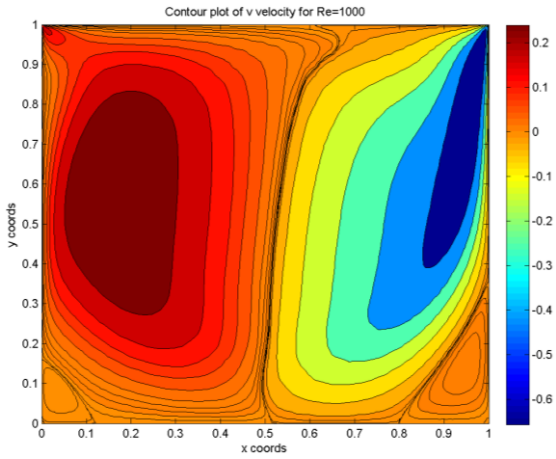


Fig. 37 - Contour plot v velocity for Re=1000

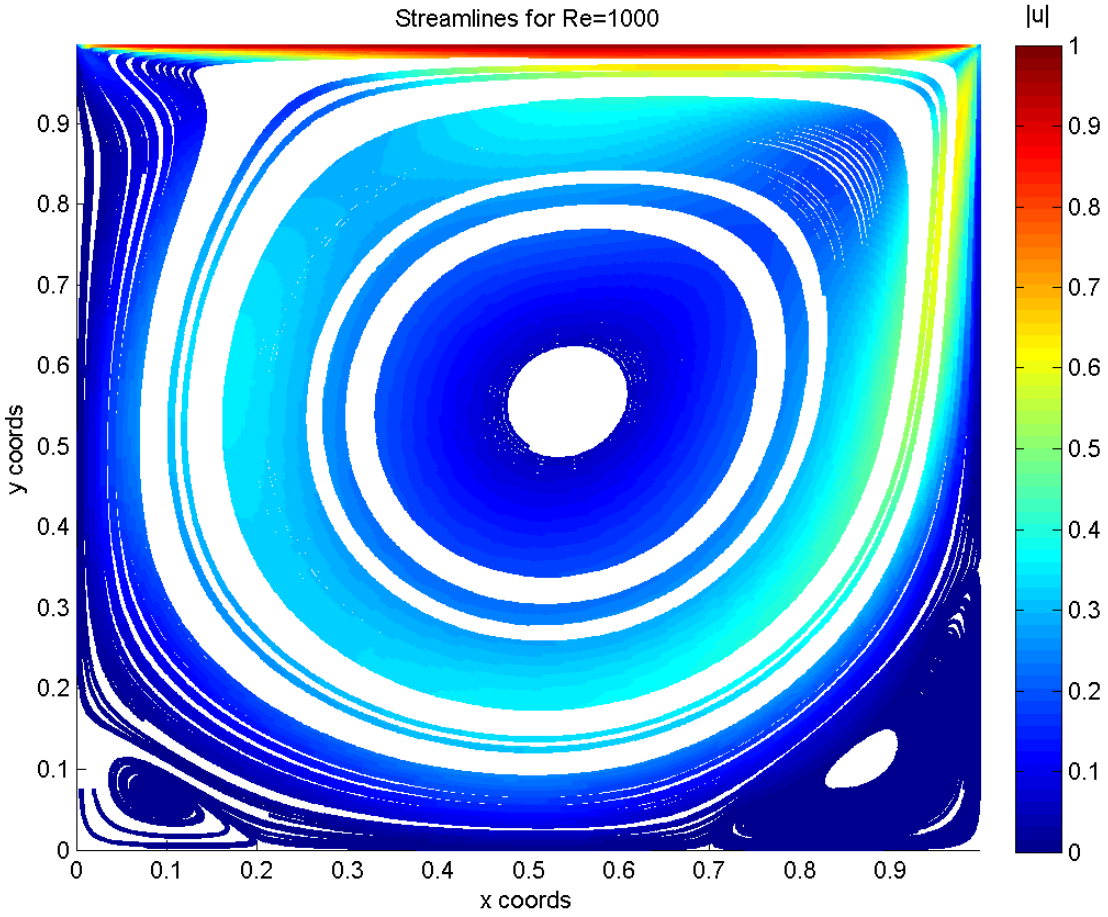


Fig. 38 - Streamline for Re=1000

Re = 5000

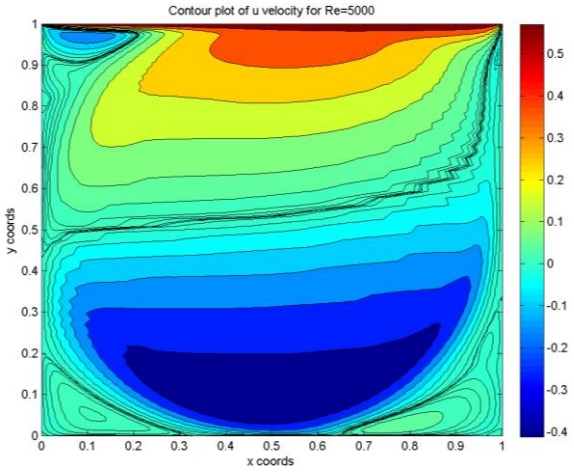


Fig. 39 - Contour plot u velocity for Re=5000

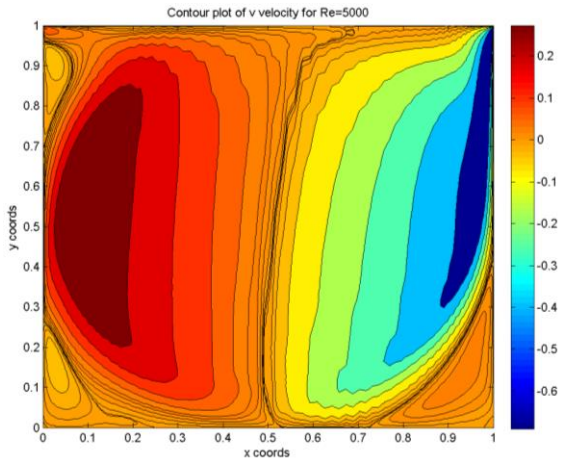


Fig. 40 - Contour plot v velocity for Re=5000

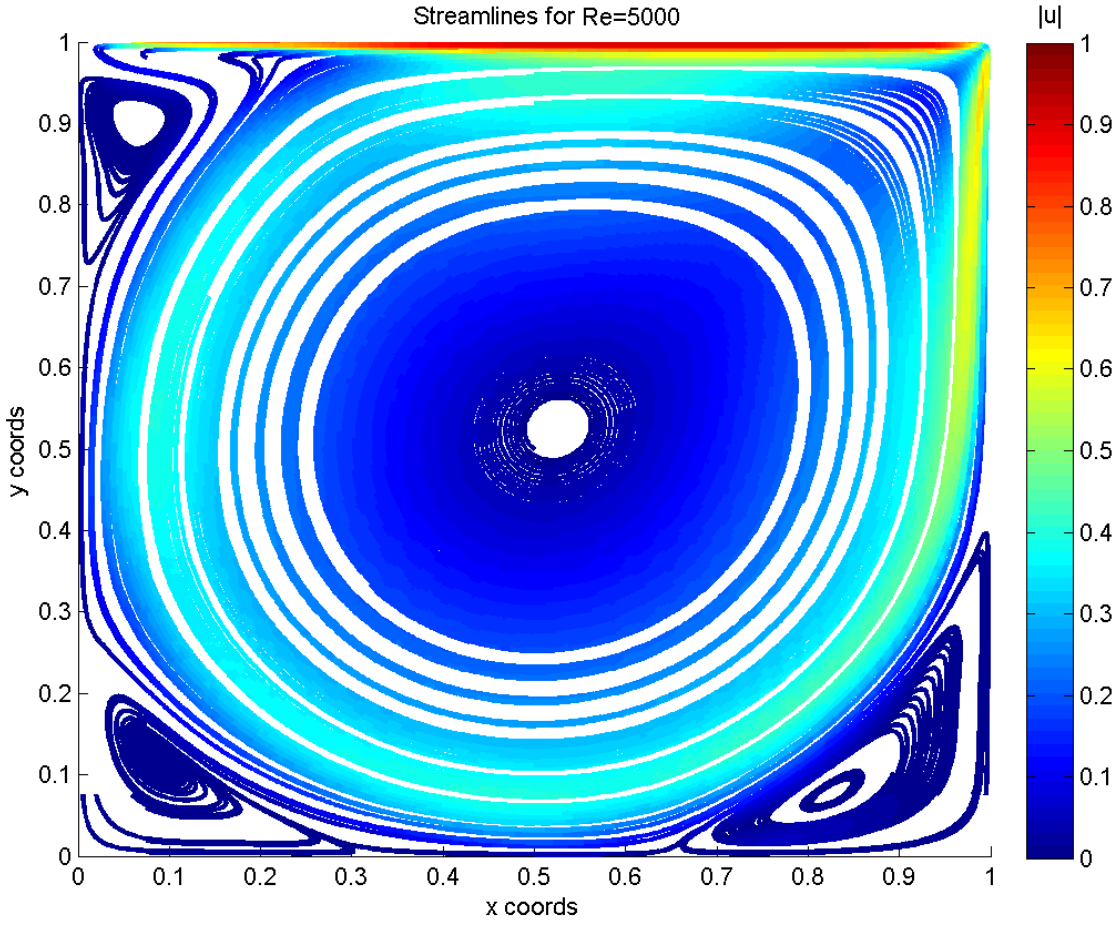


Fig. 41 - Streamline for Re=5000

To better understand how the driven cavity flow behaves, we will be analysing with some detail the cases for Reynolds numbers of 100, 1000 and 5000. For each one of those cases we can see three different graphs. The first and second graph represent the velocity contour field of the domain for the horizontal and vertical velocities respectively which show the velocity value at each point of the domain, using a colour code to make it easier to view, while the third graph is the streamline plot. The streamline plot in vulgar words is the result of dropping paint drops into the cavity and following their movement around it. We have also added a colour code to the streamlines which represents the module of the velocity vector in order not only to appreciate how the flow moves but where it moves faster or slower all in one single plot.

For Re=100 we see in Fig. 33 how the fluid at the north of the cavity moves horizontally as it is dragged by the motion of the wall until it collides with the east wall, making it only possible for the flow to go downwards. There is where we see the big blue peak on the vertical velocity in Fig. 34 which means the fluid is moving downwards. Since the fluid is very viscous, the spinning energy of the fluid is dissipated before it arrives at the bottom of the cavity which is almost not influenced or perturbed by the motion, we can see that around y=0,1 in all three graphs the value of the velocity is almost 0, specially in Fig. 33 where we see the big blue spot at the centre but not touching the bottom of the cavity.

At Re=1000 we see that the flows is not viscous enough to dissipate the rotation achieved by the fluid and that the downwards motion of the fluid at the east wall continues until the flow hits the bottom of the cavity, where it move towards the left side until colliding with the west wall and upwards again to complete the spinning motion that we already started to appreciate for Re=100. We can clearly view in Fig. 36 that now the blue spot reaches the bottom layer and same for the vertical velocity in Fig. 37 as we see the yellow zone almost reach the bottom. In the streamline plot of Fig. 38 we appreciate that in each corner of the bottom layer of the cavity there is a vortex and they increase with Reynolds numbers since they are bigger than in Fig. 35 for Re=100.

The results for Re=5000 are very similar to the ones already explained for Re=1000. What we can appreciate is the overall velocity of the spinning flow has increased as we have increased the Reynolds value and hence reduced the viscosity. The main difference for Re=5000 can be clearly seen in Fig. 41 where a new vortex appeared at the top left corner of the cavity.

4 Case of study: The Square Cylinder

Now that we have ensured our software can solve properly the Navier-Stokes equations, the next step is to use it for engineering purpose. Our objective is to be able to use this software for aerodynamical applications. In CFD for aerodynamics, the main use is the study of geometries immersed in a fluid (usually air) in order to study their behaviour and the aerodynamical forces they are subject to.

The case studied in this chapter is known as the Square Cylinder. The case consists in a square body with infinite width immersed into a fluid with a relative velocity with the body. It is considered that there is not any other influence such as walls or other bodies and since the width is considered infinite, the fluid can be considered 2D.

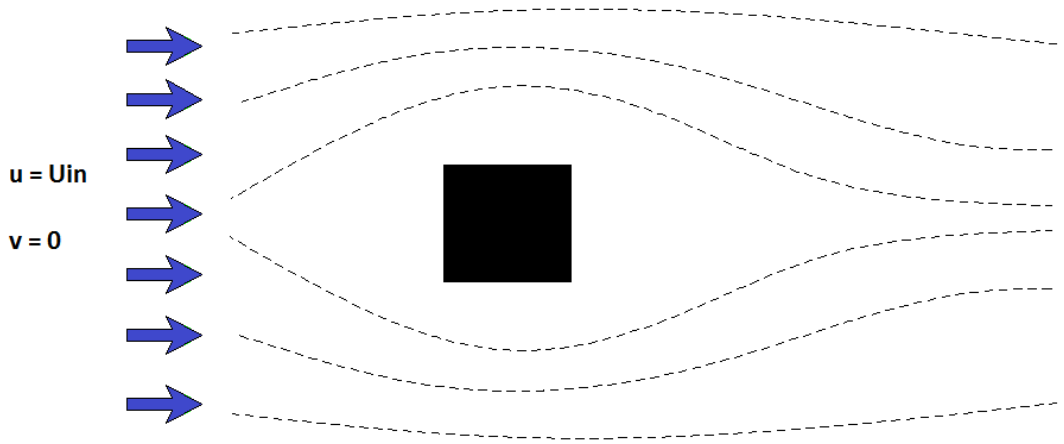


Fig. 42 – Scheme of the Square Cylinder case

This case is the equivalent of studying the behaviour of a simple object or alar profile. Our software is the CFD equivalent of building our own alar profile and testing it into a wind tunnel. The aim of this case is not to study a very complex alar profile. It is all the opposite, to study a very simple geometry as the Square Cylinder that has been widely tested by the scientific community and the case is well known. If our solver can properly solve this chase, then it is ready for more complex geometries as of those of alar profiles.

4.1 Physics description

The equations to solve are the same as for the Driven Cavity. In its adimensionalized and vectorial form are

$$\nabla \cdot u = 0 \qquad (2.27)$$

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = -\nabla p + \frac{1}{Re}\Delta u \qquad (2.28)$$

In theory, we are studying a free object with no effect of its surrounding but in CFD we cannot work with infinite domains, we need finite discretized ones. The geometry is introduced into a channel equivalent to a wind tunnel. The north and south boundaries are walls, the west boundary is the inlet and the east one the outlet. Despite using a finite domain, the dimensions of it and the relative position with the geometry of study is chosen so that the walls have no influence in the result.

In this case, we are working with an open system meaning that we have to ensure the mass flow at the inlet is the same as the mass flow at the outlet.

$$\dot{m}_{in} = \dot{m}_{out} \qquad (4.1)$$

Where $\dot{m}_{in}$ is the inlet mass flow and $\dot{m}_{out}$ the outlet mass flow.

The flow used is a Parabolic inlet flow. Instead of using a uniform flow, we have decided to use a parabolic one to prevent infinitesimal vortices at the inlet corners and avoid high velocities close to the walls for any possible unpredicted effects that this may cause. The parabolic flow is 0 in contact with the walls and has its maximum on the vertical centre. The function of the flow at the inlet is:

$$u_{inlet}(y) = u_{max}\left(1 - \frac{\left(y - \frac{H}{2}\right)^2}{\left(\frac{H}{2}\right)^2}\right) \qquad (4.2)$$

Where u_{max} is the velocity at the peak of the parabola ($u_{max} = 1$ since it will be the velocity used for adimensionalization), *y* is the vertical position coordinate and *H* is the height of the channel.

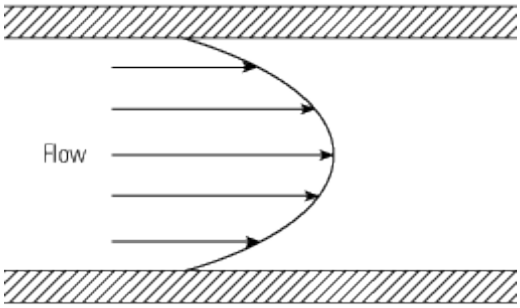


Fig. 43 - Scheme of a Parabolic inlet flow

Another key parameter, as it was for the Driven Cavity, is the Reynolds number. Again it determines the regime of the simulation and is the parameter used to compare two different experiments or conditions of the same case.

$$Re = \frac{\rho \cdot u_{max} \cdot D}{\mu} \qquad (4.3)$$

Where ρ is the density of the fluid, u_{max} the peak horizontal velocity of the parabolic inlet flow, *D* the diameter of the square cylinder and μ the dynamic viscosity of the fluid.

4.2 Geometry

As already mentioned, the case consists of a channel with an inlet and an outlet where a square cylinder has been immersed within. The geometry has to be suited in order to allow the fluid to flow around the object without interference, not generate any strange phenomena and also be easy to adapt for other cases of study.

To ensure the above conditions some guidelines are given in **[15]** about the proportions for the geometry.

- Square diameter, *D*. It is the diameter of the square cylinder, since it is a square it determines the base and height of the object.
- Height or cavity section, *H*. It is the height in the *y* direction of the channel.
- Length, *L*. It is the length in the *x* direction of the channel. The recommended value is $L = 50 * D$.
- Blockage ratio, *B*. It is the ratio *D/H*, it fixes the relation between the size of the cylinder and the height of the channel. The recommended value is $= \frac{D}{H} = \frac{1}{8}$.
- Inlet distance, *l*. The position of the square cylinder with respect to the inlet. The recommended value is $= \frac{L}{4}$.
- The square cylinder is at the vertical centre of the channel, *H/2*.

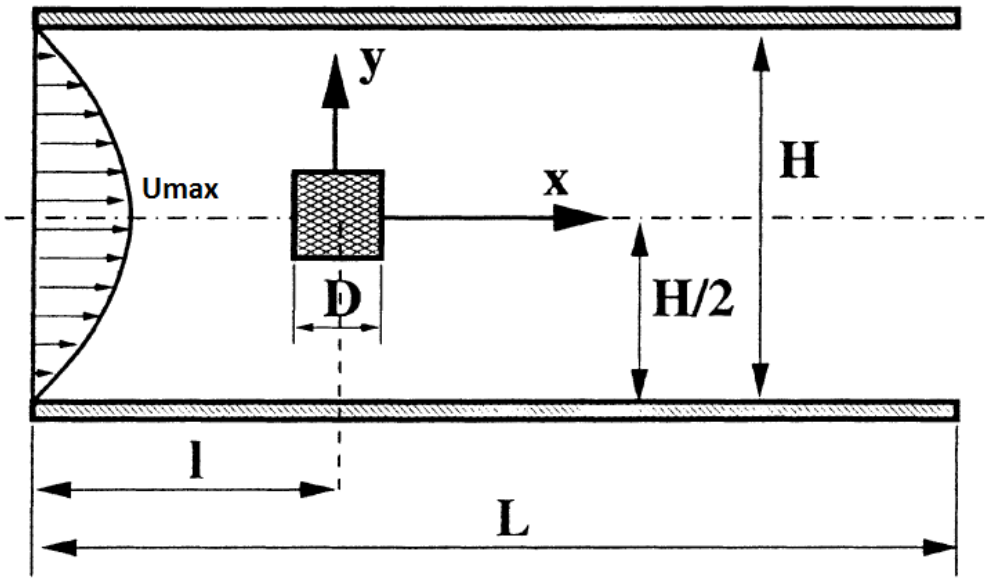


Fig. 44 - Scheme of the domain for the Square Cylinder case, extracted from [15]

Now that we have all the proportions of the geometry, we need to adimensionalize all the domain with one of them. In order to make it easier to adapt the code for different objects if wanted, we have used the height *H.* Resulting in the following adimensionalized dimensions:

DIMENSION	H	D	L	l
VALUE	1	0,125	6,25	1,5625

Table 6 - Geometry dimensions

Finally, the square cylinder has no angle attack, meaning it has its sides parallel to the channel and the top and bottom faces are parallel to the horizontal inlet flow.

4.3 Mesh

Our main interest for the Square Cylinder problem is the surroundings of the cylinder and specially downstream of it where the trail and most of the perturbations are located. In order to have more precision on that area but also optimize as much as possible our computation time we have decided to use a non-uniform mesh.

Since the height of the domain is very small in comparison with the length we have decided to maintain a uniform distribution of elements alongside the *y* axis. We also did a first simulation with a completely uniform mesh and observed that the perturbations on the *y* directions occupy around 80% of the height. It was not worth all the effort and increase in computational time just to optimize the mesh for the remaining 20% of the domain.

The non-uniform distribution used alongside the *x* axis consists on dividing the length in three sections. The first section ranges from $0 \leq x < 0{,}7 * l$, the second one from $0{,}7 * l \leq x \leq 0{,}64 * L$ and the last one from $o, 64 * L < x \leq L$.

The second section is the one containing the cylinder and most of its trail, hence our main interest section. This section follows a uniform distribution and if we have Nx as the total number of elements on the *x* axis, we have as input the number of elements (Nxu) we want in this segment. Basically this allows us to refine the mesh as much as we need around the cylinder.

The remaining number of elements (Nx-Nxu) are distributed proportionally to the length those sections occupy along the *x* axis. But just the number of elements its proportional because we have used again the hyperbolic tangent function to densify the mesh as we move close to the second section and less as we move away from it.

The function used for the first segment element distribution is:

$$x_i = \frac{\tanh\left(\gamma \cdot \frac{i-1}{N1}\right)}{\tanh(\gamma)} \cdot x1 \qquad (4.4)$$

Where x_i stands for the *x* coordinate of the element *i*, *i* is the number of element along the *x* axis, N1 the number of elements in the first segment, γ is a dimensionless parameter to adjust the density ($\gamma = 1{,}9$) and $x1$ the end of the segment ($x1 = 0{,}7 * l$).

In the last segment we need the hyperbolic tangent to do the opposite as on the first segment, be more dens at the beginning of the section and less as we get close to the boundary. The function used is:

$$x_i = \left[\left(1 - \frac{\tanh\left(\gamma \cdot \frac{N2 - (i - N1 - Nxu)}{N2 + 1}\right)}{\tanh(\gamma)}\right) \cdot (L - x2)\right] + x2 \qquad (4.5)$$

Where x_i stands for the *x* coordinate of the element *i*, *i* is the number of element along the *x* axis, N1 the number of elements in the first segment, Nxu the number of elements in the second segment, N2 the number of elements in the third segment, γ is a dimensionless parameter to adjust the density ($\gamma = 1{,}9$), *L* the length of the domain, $x1$ the end of the segment ($x1 = 0{,}7 * l$) and $x2$ the end of the segment ($x2 = 0{,}64 * L$).

The resulting mesh achieves our objective of having a high density of elements around the cylinder and its trail and reducing the resources close to the inlet and outlet in those areas of less interest as it can be seen in Fig. 45.

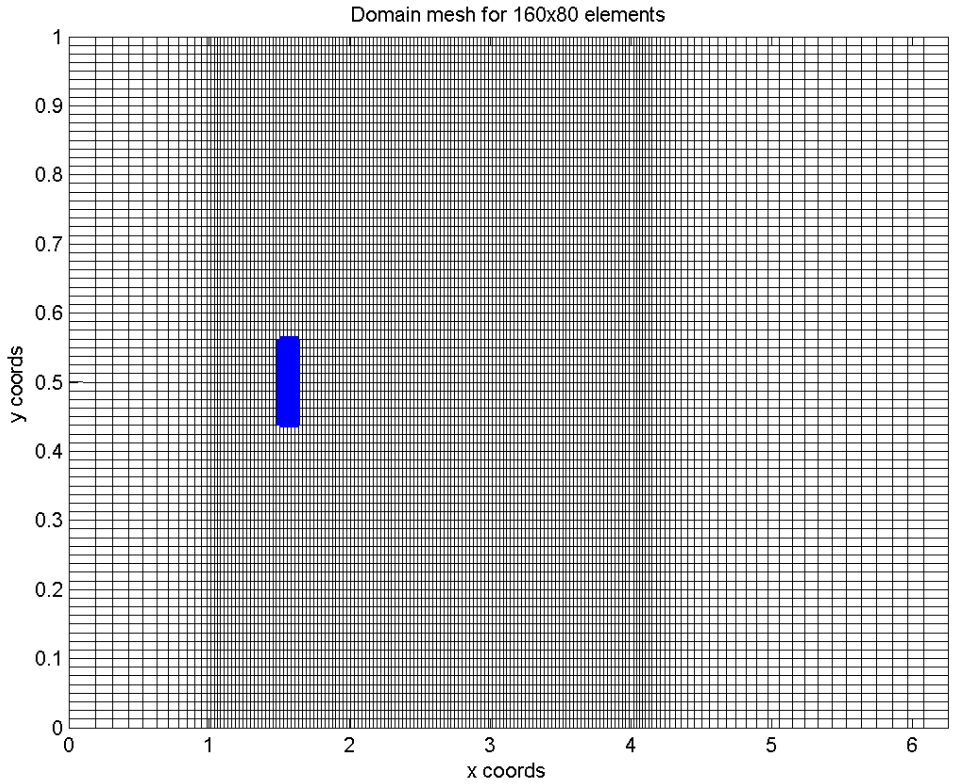


Fig. 45 – Mesh scheme for the Square Cylinder problem

We have included the square cylinder in blue over the mesh so we can have a better idea of how the meshed domain results and how the mesh is formed around the cylinder. In Fig. 45 the cylinder seems rectangular but this is due to the vast difference between the dimensions of the *y* axis and the *x* axis, they have different scales but for a more clear vision we have decided to use this scale. In Fig. 46 we can see the domain without scaling the *x* axis to fit better, we can appreciate the square shape of the cylinder but to view with detail what happens downwash of the geometry would be much more difficult than to just keep in mind that despite seeming like a rectangle it is still a square shape object.

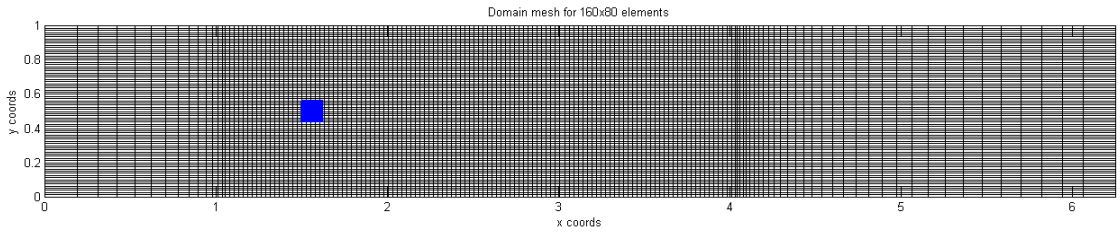


Fig. 46 - Scheme of the meshed domain without axis scaling

4.4 Boundary Conditions

In this section we will go through all the boundary conditions imposed to model the channel and the immersed cylinder.

4.4.1 Inlet conditions

The inlet has its velocity fixed. As already mentioned, the horizontal velocity follows the distribution shown in (4.2) and the vertical velocity is $v = 0$.

The pseudo-pressure at the inlet has *Neumann Boundary Conditions*.

4.4.2 Walls conditions

The velocity at the north and south wall of the channel has *Dirichlet Boundary Conditons* known as the no-slip condition which fixes both the horizontal and vertical velocity to $u = v = 0$.

The pseudo-pressure conditions are also *Neumann*.

4.4.3 Object conditions

Since the object, in this case the square cylinder, is not part of the fluid we cannot apply the Navier-Stokes Equations to it. But since the code does not understand if it is a fluid or a solid, we use a solution known as the *Immersed Boundary Method*.

The essence of the method consists in apply the Navier-Stokes Equations to the solid as it wasn't one adding a "fictitious" force term to the equation

$$\frac{\partial(\rho\vec{u})}{\partial t} + \nabla\cdot(\rho\vec{u}\vec{u}) = -\nabla \mathrm{p} + \nabla\cdot\vec{\tau} + \vec{f} \qquad (4.6)$$

this force $\vec{f}$ is used to compensate all the other terms of the equation to force the velocity $\vec{u} = 0$. Note that this is just the physical justification of the method, but to implement it there is no need to compute $\vec{f}$, just fix the velocities $u = v = 0$.

4.4.4 Outlet conditions

The objective of the conditions at the boundary is to ensure there is no bounce back of any sort, a smooth outlet flow and the conservation of mass flow for the whole system. The best condition to guarantee that is the *Convective Boundary Condition* already explained in detail at section **2.4** and implemented as shown in

$$U_N^{n+1} = U_N^n - \frac{\Delta t}{(\Delta x)_N} U_c (U_N^n - U_{N-1}^n) \qquad (4.7)$$

This condition works very well but sometimes can generate some very small mass flow imbalances that can lead to major malfunctions of the software since the imbalance, even the smallest ones, affects each iteration generating a vortex or summit. In order to correct those small imbalances we apply a small correction factor to the velocity to prevent it.

4.4.5 Fixed pressure point

We have already covered several times that the system $A \cdot x = b$ has infinite possible solutions but we need to obtain the same on every time for the same experiment. Not wanting to condition severely the pseudo-pressure map, the *Dirichlet Boundary Condition* used in the Driven Cavity has been discarded. The condition used introduces a small modification on the a_P term and also allows us to make it easier to implement, if wanted, the Conjugate Gradient method since it conserves symmetry of the *A* matrix.

$$\begin{gathered} a_E = \frac{\Delta y}{d_{EP}} \\ a_W = \frac{\Delta y}{d_{PW}} \\ a_N = \frac{\Delta x}{d_{NP}} \\ a_S = \frac{\Delta x}{d_{PS}} \\ \boldsymbol{a_P = 1,1 * (a_E + a_W + a_N + a_S)} \\ b_P = \left(\rho u_e^p \Delta y\right) - \left(\rho u_w^p \Delta y\right) + \left(\rho u_n^p \Delta x\right) - \left(\rho u_s^p \Delta x\right) \end{gathered} \qquad (4.8)$$

4.5 Aerodynamic forces

Since one of the main objectives of this project is to use the software as a wind tunnel for testing geometries and alar profiles, two key aerodynamic parameters for aeronautical design of profiles are Lift and Drag.

Lift

Is the component of the force perpendicular to the oncoming flow direction. It is the force that generates the sustentation for our object and in more vulgar words, makes it fly. For all alar profiles, the goal is to generate lift but for racecars (for example F1) the aim is all the opposite. Racecars want to generate negative lift, usually renamed as Downforce, which provides the cars with better grip allowing them to achieve higher speeds at turns.

The lift generated by an object can be calculated as

$$L = \int_s P\,dx - \int_n P\,dx + \mu \int_e \frac{\partial v}{\partial x} dy - \mu \int_w \frac{\partial v}{\partial x} dy \qquad (4.9)$$

where the integration surfaces *w, e, n* and *s* refer to the object walls west, east, north and south respectively.

Drag

Is the component of the force parallel to the oncoming flow direction acting opposite to the relative motion of any object moving with respect to a surrounding fluid. Drag is a type of friction that depends highly on velocity and it always decreases the fluid velocity relative to the solid object in fluid's path.

The drag generated by an object can be calculated as

$$D = \int_w P\,dy - \int_e P\,dy + \mu \int_n \frac{\partial u}{\partial y}dx - \mu \int_s \frac{\partial u}{\partial y}dx \qquad (4.10)$$

where the integration surfaces *w, e, n* and *s* refer to the object walls west, east, north and south respectively.

As we have done with all our variables, we have to adimensionalize them. To adimensionalize the drag and lift forces we use the following reference force:

$$F_{ref} = \frac{1}{2}\rho u_\infty^2 S \qquad (4.11)$$

Once our aerodynamic forces are adimensionalized we obtain two dimensionless parameters known as Lift coefficient (C_L) and Drag coefficient (C_D). Those parameters are generally the ones presented to describe and characterize an alar profile or any other aerodynamic bodies. They allow us to easily compare different geometries and different flow parameters.

4.6 Vortex Shedding

Our case of study has a special phenomenon known as Vortex Shedding. For a certain range of Reynold values, the flow past the square cylinder generates a series of vortices that detach periodically from either side of the body. It results in a series of periodically oscillating vortices at the trail of the object. The usual range of Reynolds at which this phenomenon appears is around $50 \leq Re \leq 250$ but this range varies depending on the Blockage ratio and inlet flow used for the simulation or experiment. It is a very curious effect since between this range the flow is not stationary but it is neither turbulent, the flow is transitory and laminar. It has a laminar vortex trail that oscillates with time.

During the Vortex Shedding, the velocity field follows a finite series of vortex profiles that repeat with a periodical frequency that can be measured. With the aim of characterizing the phenomenon, an adimensional number was defined. This number is named Strouhal number and defined as

$$St = \frac{f_s \cdot D}{u_\infty} = \frac{D}{T_s \cdot u_\infty} \qquad (4.12)$$

where St is the Strouhal number, f_s is the frequency of the process, *D* is the diameter of the square cylinder and u_∞ the velocity upstream of the object.

4.7 Results analysis and comparison

In this section we will show and analyse the results obtained by our software and we will also be performing a comparison of those results with the reference ones from **[15]**.

The analysis and comparison will be divided in two sections. In the first section we will study the laminar stationary range of the Square Cylinder solution that goes from $0{,}5 \leq Re \leq 50$ and in the second section we will study the laminar transitory part, ranging $50 \leq Re \leq 250$.

All results have been obtained with our non-uniform mesh (explained in section **4.3**) using 160x80 elements with 110x80 elements on the uniform central segment around the cylinder.

4.7.1 Pre-Vortex Shedding

The phenomenon known as Vortex Shedding starts to appear at values slightly around Re=50. For lower Reynolds number we are in the previous stage of the Vortex Shedding. On that range of Reynolds the flow is not only laminar but it also reaches a stationary state. In this section we will focus on the study of this stage.

First we will study in detail the case for Re=40, then we will compare it with different Reynold values and finally we will compare the results obtained with the reference ones.

Results Re = 40

We have chosen Re=40 because it is one of the closest values to the Vortex Shedding zone without having to worry about any transitory effects starting to appear. It is also a good value because the fluid is not very viscous and we can appreciate very well the singularities around the cylinder.

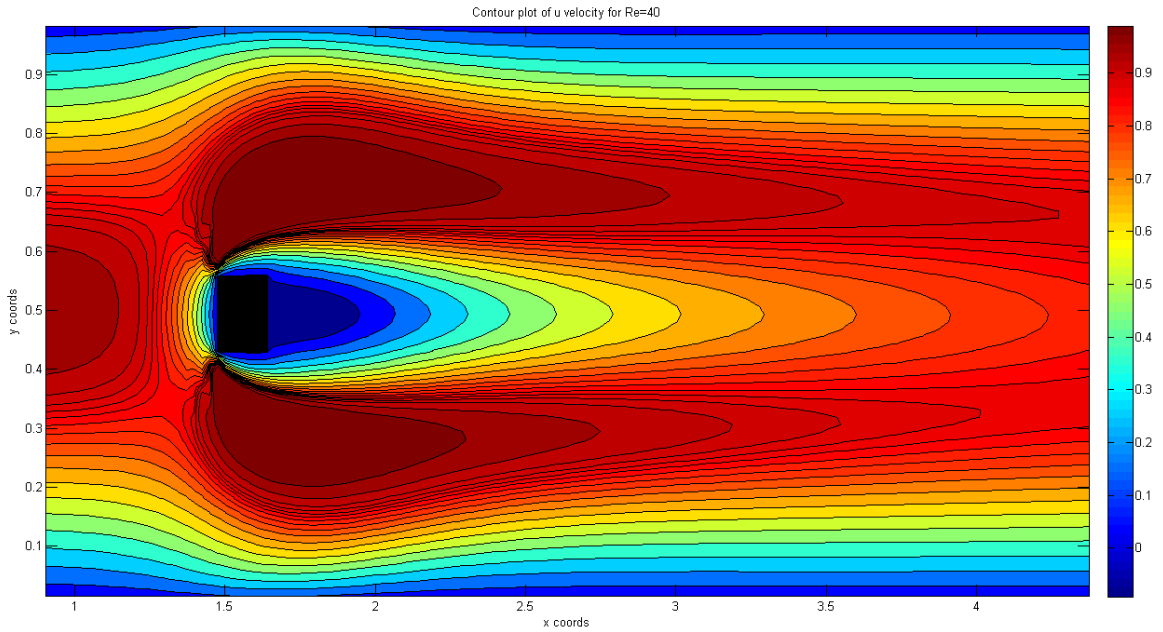


Fig. 47 - Contour plot of u velocity for Re=40

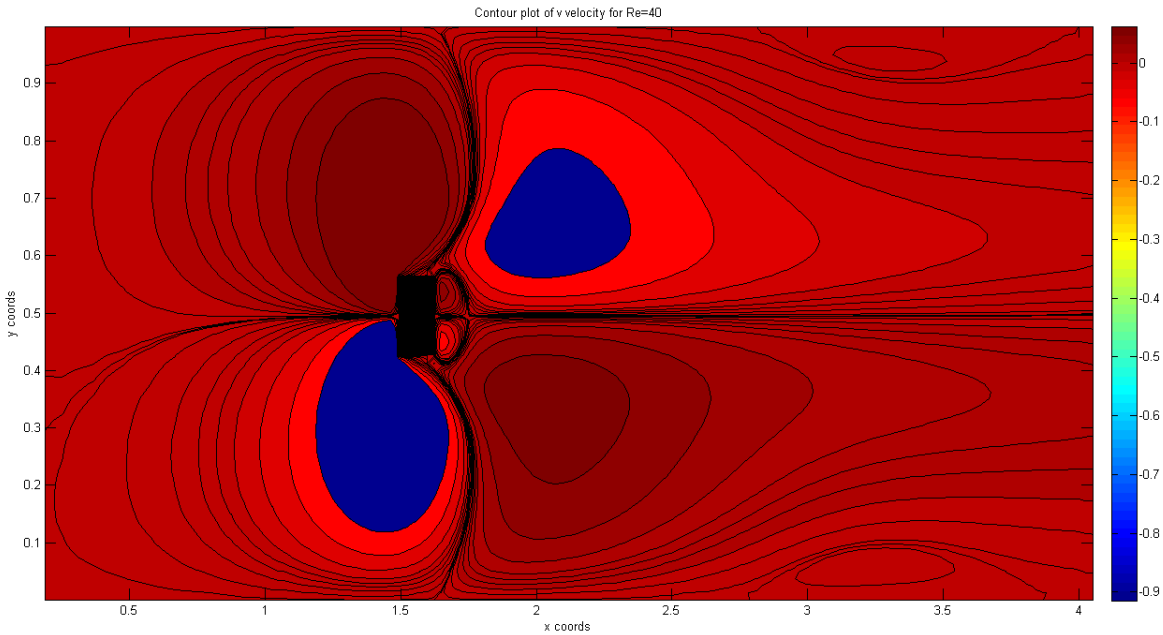


Fig. 48 - Contour plot of v velocity for Re=40

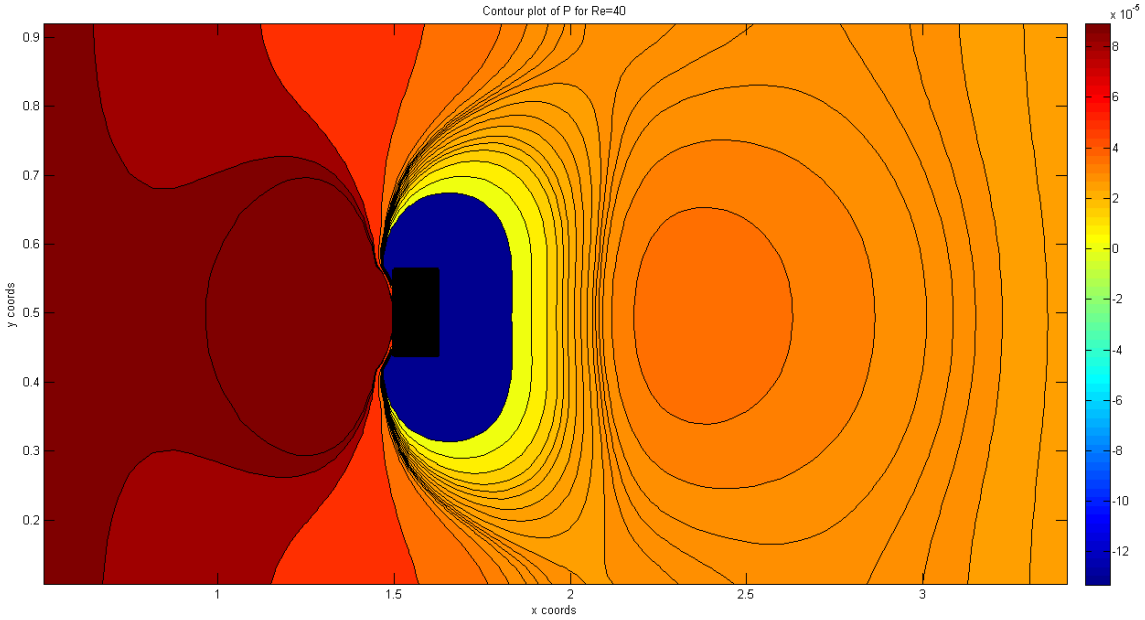


Fig. 49 - Contour plot of pressure for Re=40

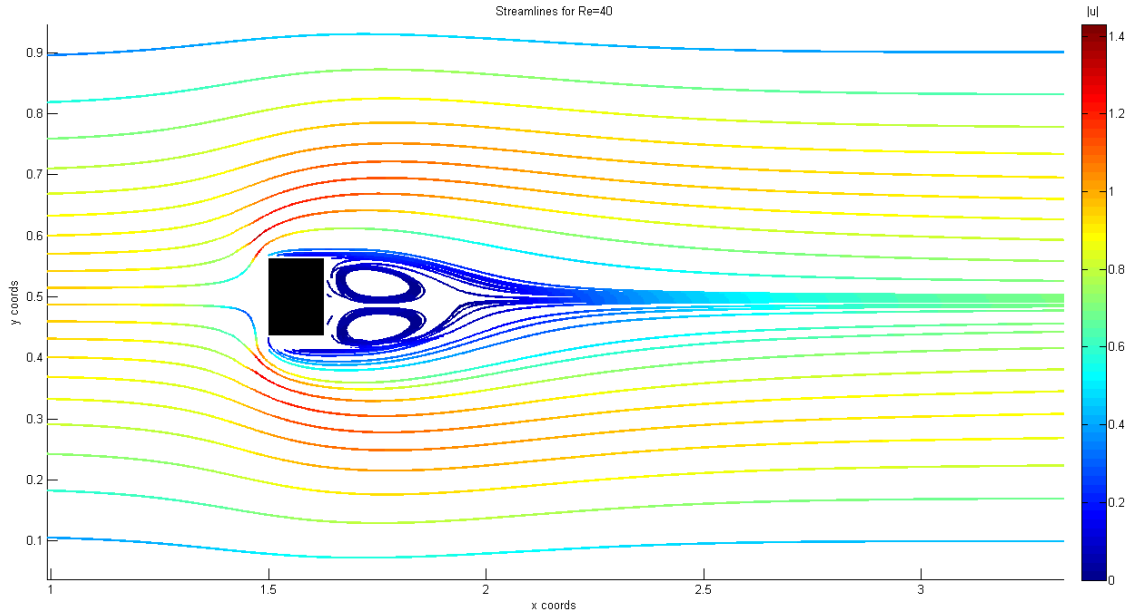


Fig. 50 - Streamlines with velocity module for Re=40

In Fig. 47 we can see the parabolic flow at $x = 1$ which is the inlet flow. The flow starts to reduce its velocity as it gets closer to the cylinder in order to avoid it. Over and under the square the see the flow accelerate since the same mass flow has to cross a smaller section due to the object. Just behind the geometry the fluid has almost no velocity because it still hadn't had time to fill the gap created by the square. Then the flow returns to its unperturbed parabolic configuration.

In Fig. 48 for the vertical velocity, we can see four main spots that are symmetric along the vertical centre line. The two spots in front of the cylinder show how the fluid goes over on the top half side and under on the other half. The situation is inverted in the back of the square to recover from the obstruction. We can also appreciate two small spots attached just on the back of the geometry which are the recirculation vortices.

In the pressure plot of Fig. 49 we see an increase in pressure in front of the object and a decrease behind it. This pressure difference is one of the main responsible for the forces suffered by the geometry, in this case the drag since symmetric geometries do not generate lift.

In order to properly see the flow motion on the domain we have added the Streamline plot of the flow in Fig. 50. There we can clearly see the flow moving around the object and the recirculation vertexes attached to the back of the square. The colour code added to the streamlines is the module of the velocity vector, allowing us not only to see the motion of the blow but also its speed. Thanks to that we can appreciate that the vortex behind the object are very low velocity vortices.

Re = 1, Re = 10, Re = 20 and Re = 40 comparison

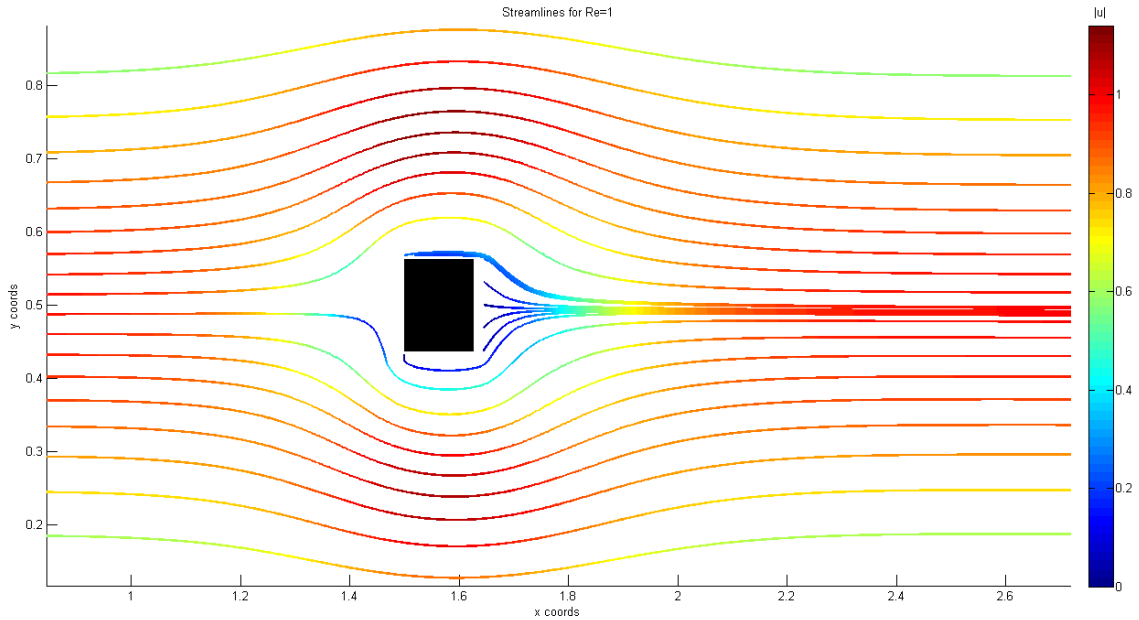


Fig. 51 – Streamlines with velocity module for Re=1

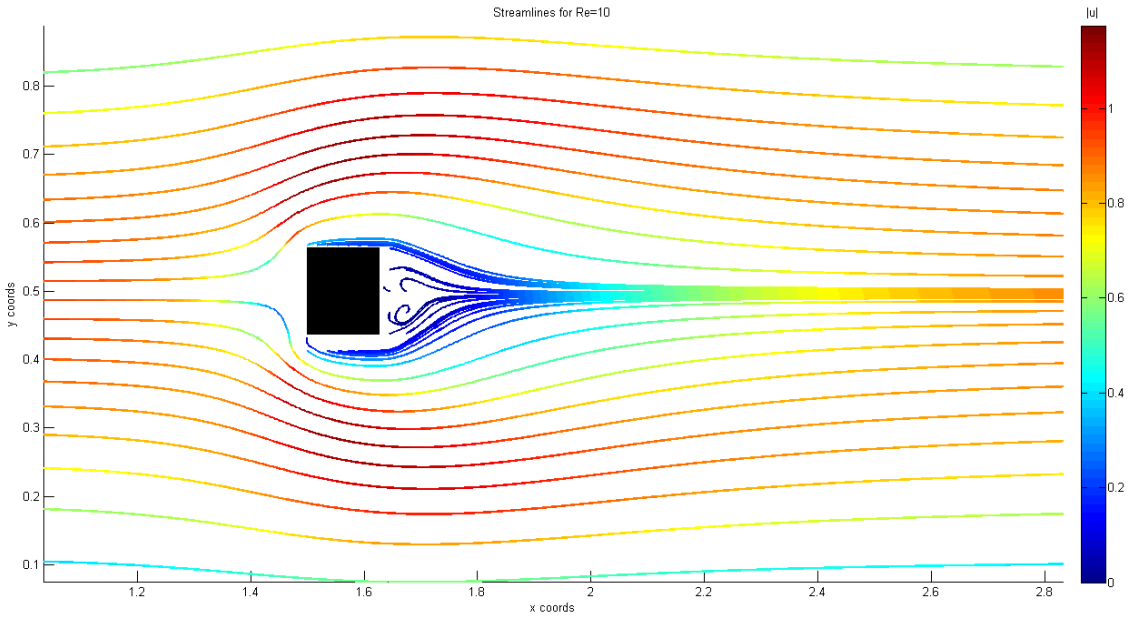


Fig. 52 - Streamlines with velocity module for Re=10

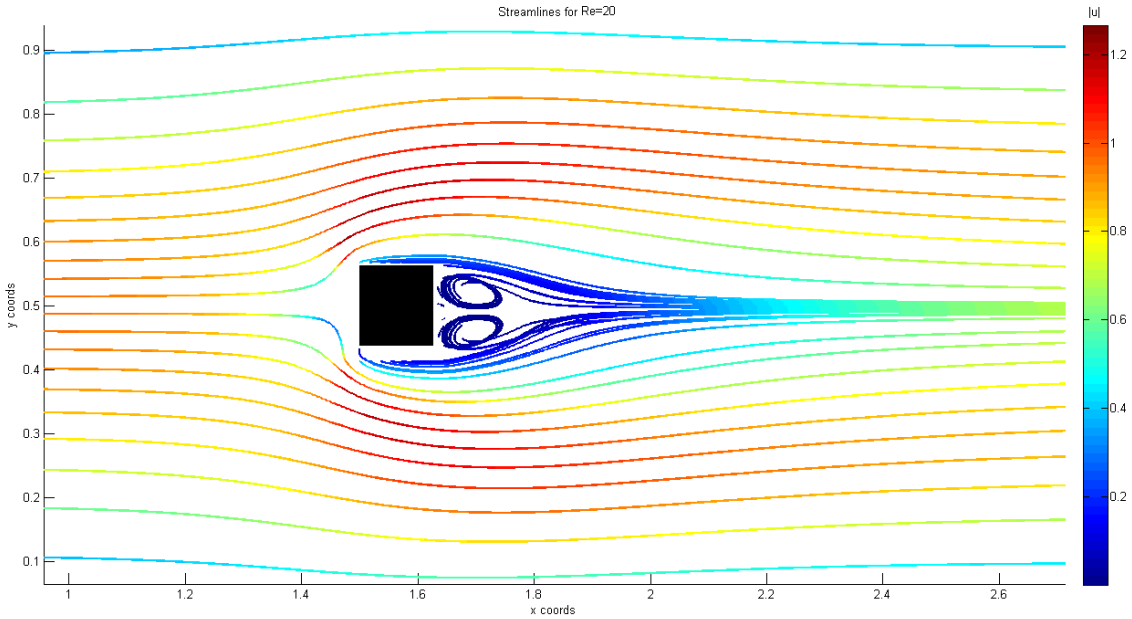


Fig. 53 - Streamlines with velocity module for Re=20

In Fig. 51 we can see the streamlines for Re=1 and how the fluid is almost not disturbed by the presence of the cylinder. The flow is very viscous and it will generate a lot of friction but will pass undisturbed by the object.

For Re=10 in Fig. 52 we start to see a bit more of obstruction by the geometry and we also notice for the first time the apparition of the recirculation vortices.

In general, what we can appreciate is an increase with the Reynolds number not only of the perturbations generated by the square cylinder but also as this perturbation increases, the size of the recirculation vortices increase too.

Comparison with reference values

One of the first comparisons we must do is the flow motion around the object. In **[15]** the streamlines for Re=1 and Re=30 are given.

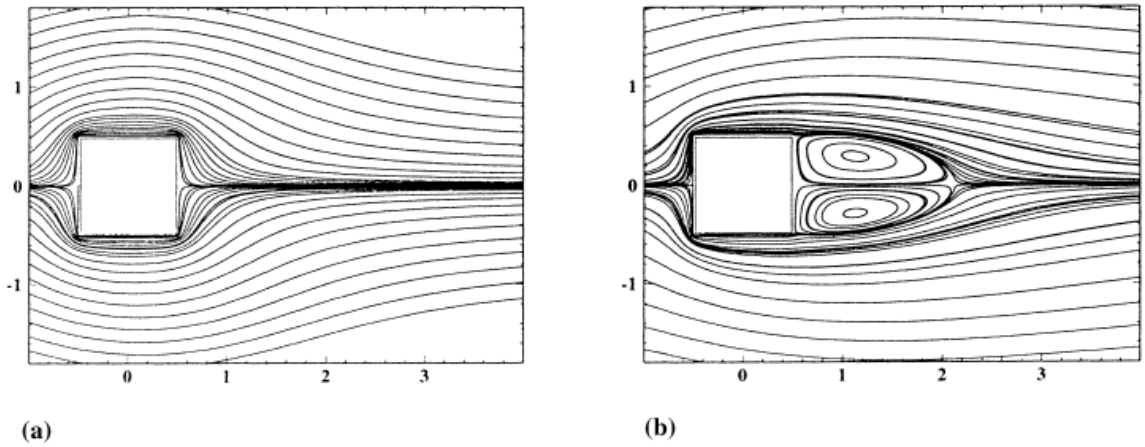


Fig. 54 - Streamlines of reference for different Re numbers. (a) Re=1, (b) Re=30

The pattern followed by the reference results and the ones obtained with our simulations reflect the same behaviour of the flow.

Another key parameter for comparison is the Lift and Drag coefficients explained in section **0**. About the lift there is not much to compare since the lift generated by a symmetrical profile with no angle of attack is equal to 0. The pressure field is symmetrical over and under the geometry and the forces suffered by the front and back faces of the square due to the vertical velocity are 0 because the vertical velocity is also symmetric with respect to the centre of the square, compensating each other.

The drag coefficient is obviously not zero. We have seen in Fig. 49 the pressure difference between the front and rear ends of the cylinder and adding the friction generated in the top and bottom faces we obtain our drag component.

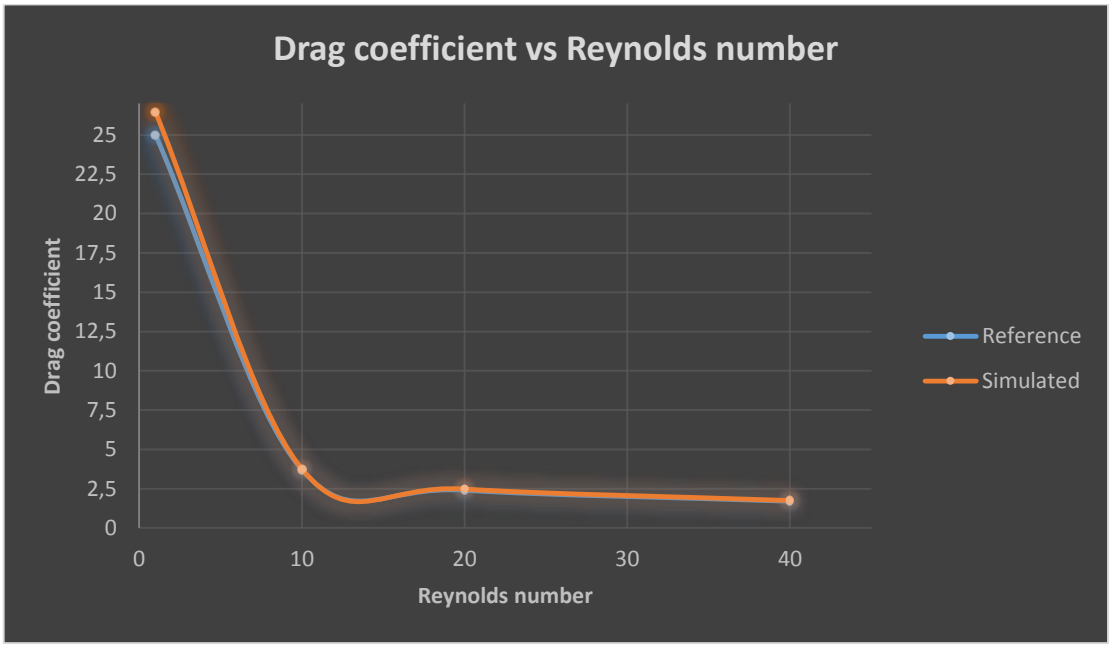


Fig. 55 - Drag coefficient vs Reynolds number comparison

The results obtained by our simulation are very similar to the ones obtained by our simulation, even though our values are slightly higher in all the cases.

4.7.2 Vortex Shedding range

In the range of Reynolds around $50 \leq Re \leq 250$ the Vortex Shedding phenomenon explained in section 4.6 appears. As in the pre-Vortex Shedding section, we will first analyse in detail the results for the Re=100 case, then compare it with other Reynolds number simulations and finally compare it with the benchmark values.

Re = 100

Because of the transitory behaviour of the flow, we can consider it has two phases. The first phase is in the early stages of the flow and known as the settling time, the flow seems to behave as in the pre-Vortex Shedding region but ones it reaches what seems to be the stationary regime it then starts to fluctuate with the Vortex Shedding. The second phase is once those oscillations have started and it oscillates indefinitely.

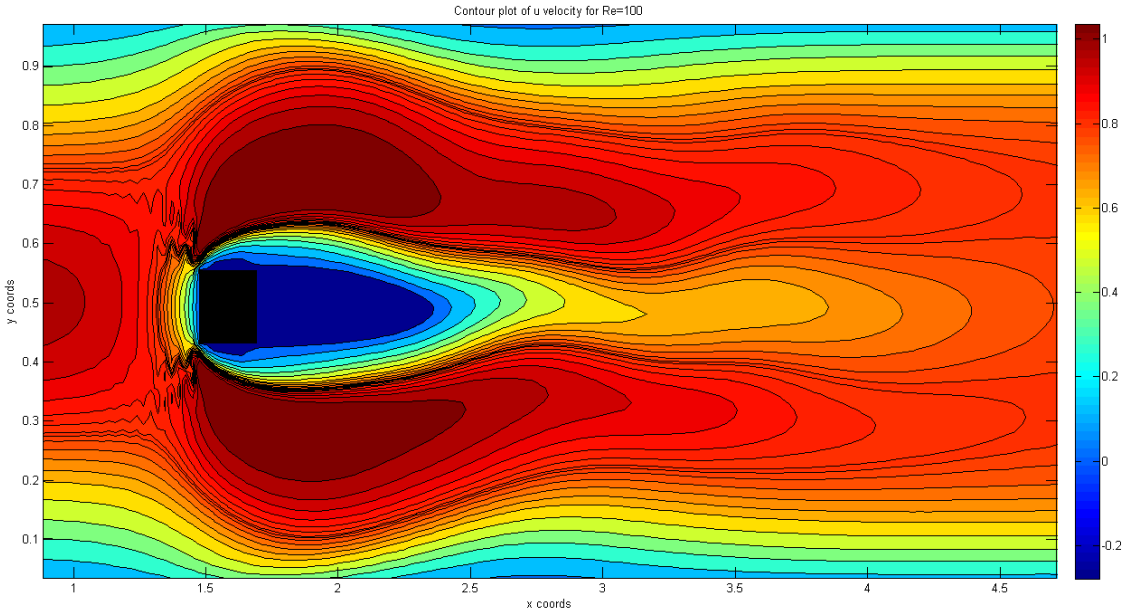


Fig. 56 - Contour plot of u velocity for Re=100 at t=4,40

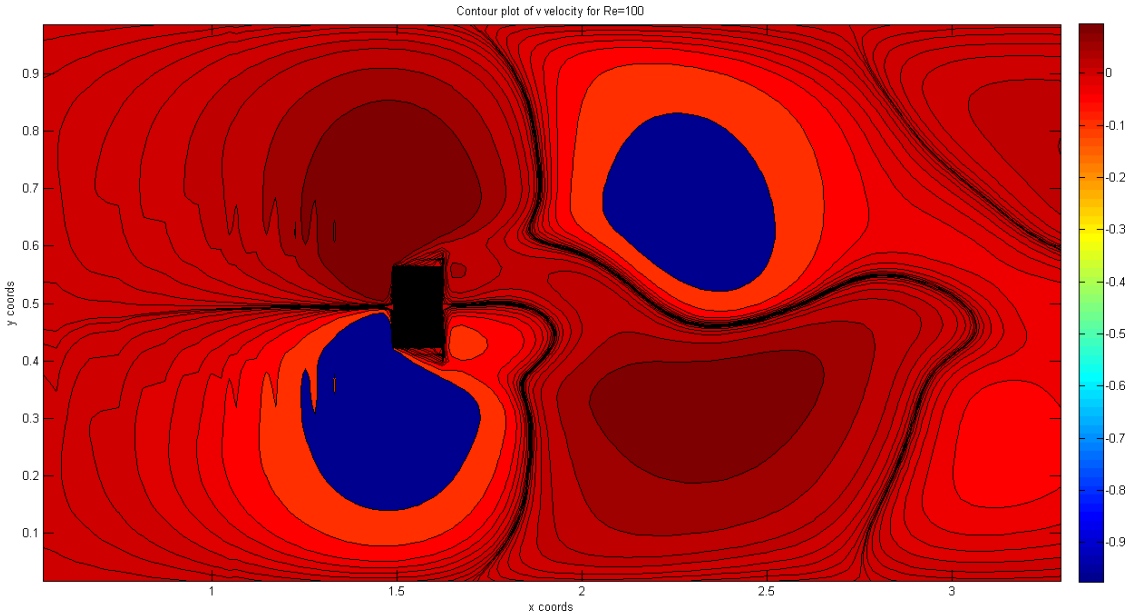


Fig. 57 - Contour plot of v velocity for Re=100 at t=4,40

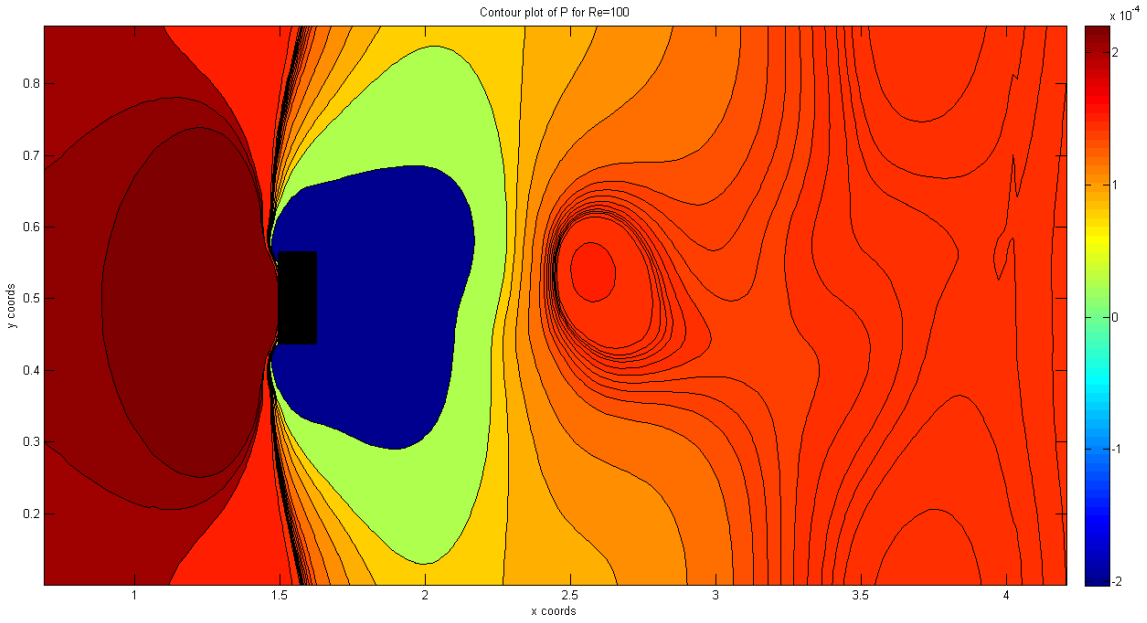


Fig. 58 - Contour plot of pressure for Re=100 at t=4,4

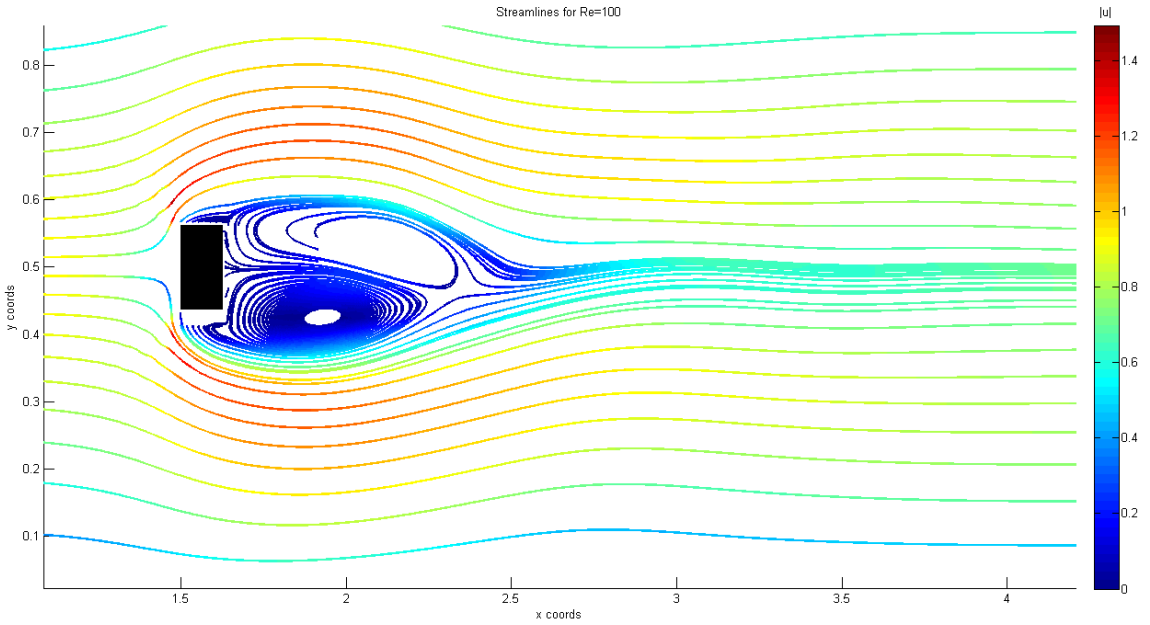


Fig. 59 - Streamlines for Re=100 at t=4,4

From Fig. 56 to Fig. 59 we can see the flow behave almost as in the previous section. We have the recirculation vortices at the back of the square but the symmetry along the vertical centre is starting to break. The lower vortex is bigger than the upper vortex which is flatter and the symmetry of the vertical velocity at the trail of the object is clearly broken. This stage only lasts a bit until the oscillation starts.

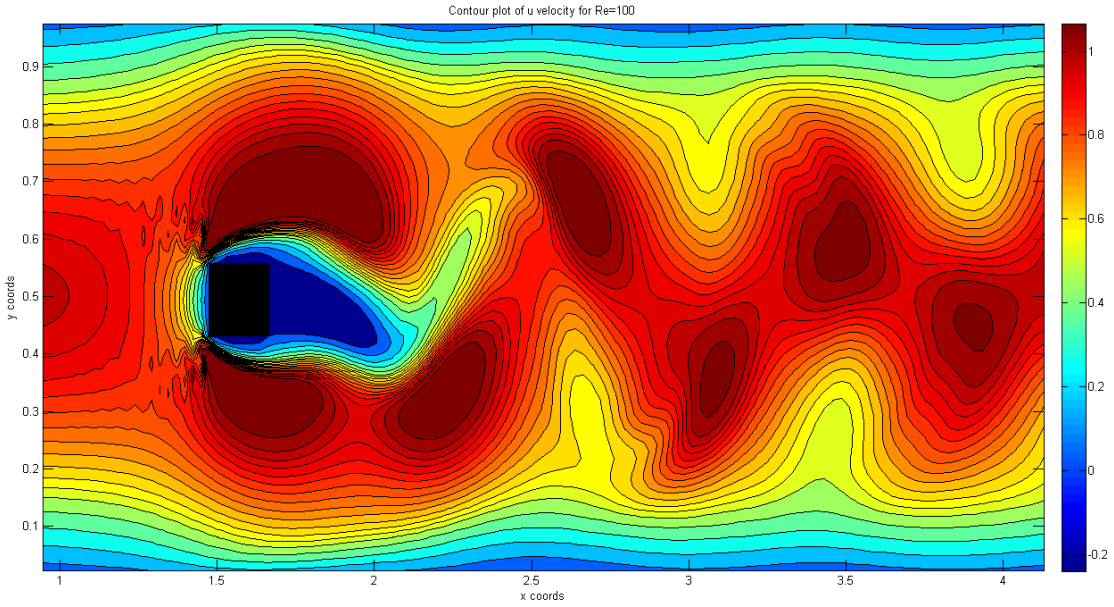


Fig. 60 - Contour plot of u velocity for Re=100 at t=24,55

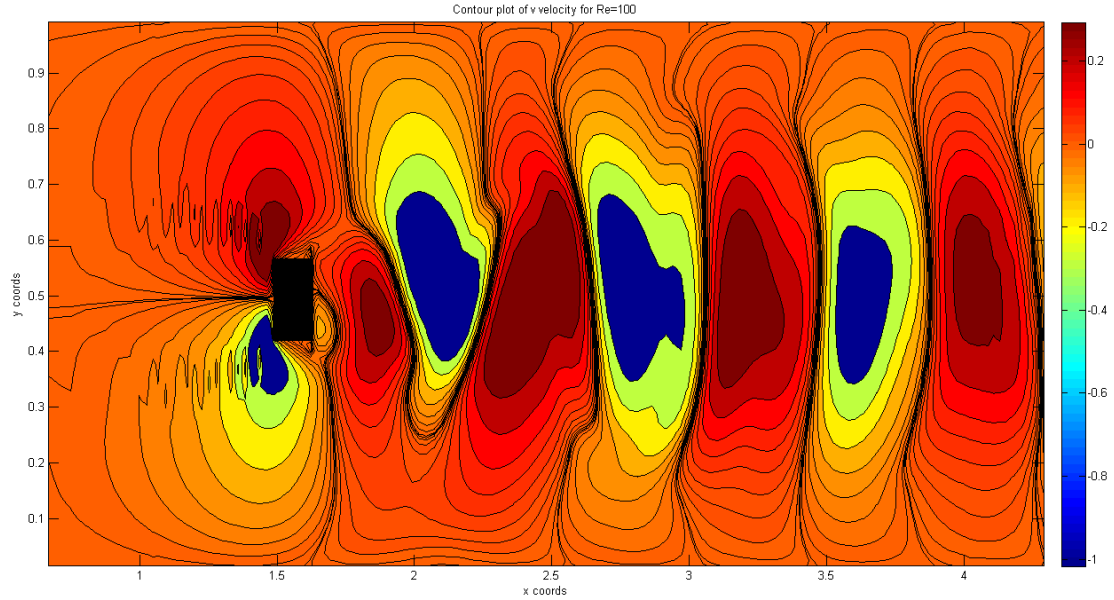


Fig. 61 - Contour plot of v velocity for Re=100 at t=24,55

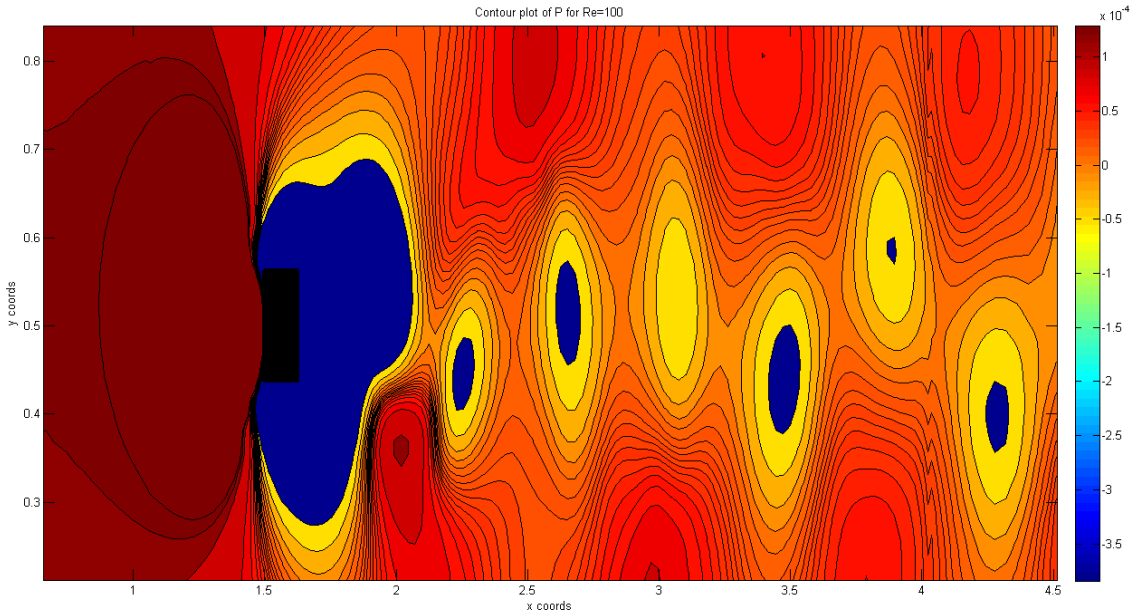


Fig. 62 - Contour plot of pressure for Re=100 at t=24,55

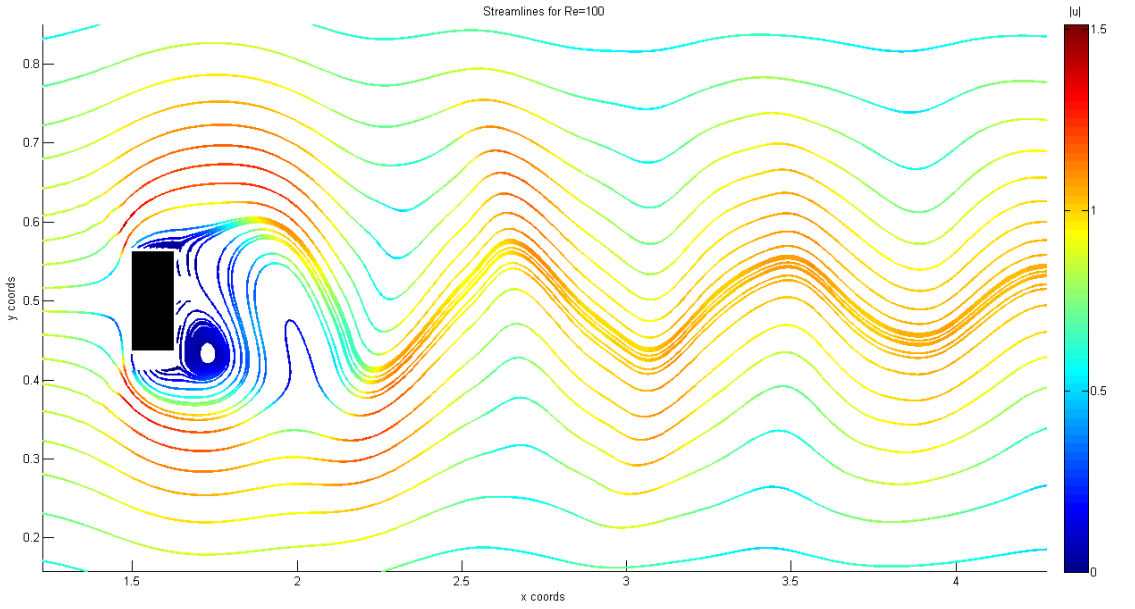


Fig. 63 - Streamlines for Re=100 at t=24,55

Through Fig. 60 to Fig. 63 we have chosen the instant if time t=24,55 where the oscillation is generating the maximum lift (we will talk later about the lift behaviour). As we can see, the upper vortex has detached from the trailing edge of the cylinder and since this effects alternates with the upper and lower vortex, we can see in the trail the previously generated vortices alternating. The remaining vortices are not fully appreciated in Fig. 63 because despite the Vortex Shedding effect the flow is still travelling towards the outlet of the wind tunnel, we can see the oscillation of the trail but not the vortices on it. Those vortices can be seen in the other plots as we see the alternate spots.

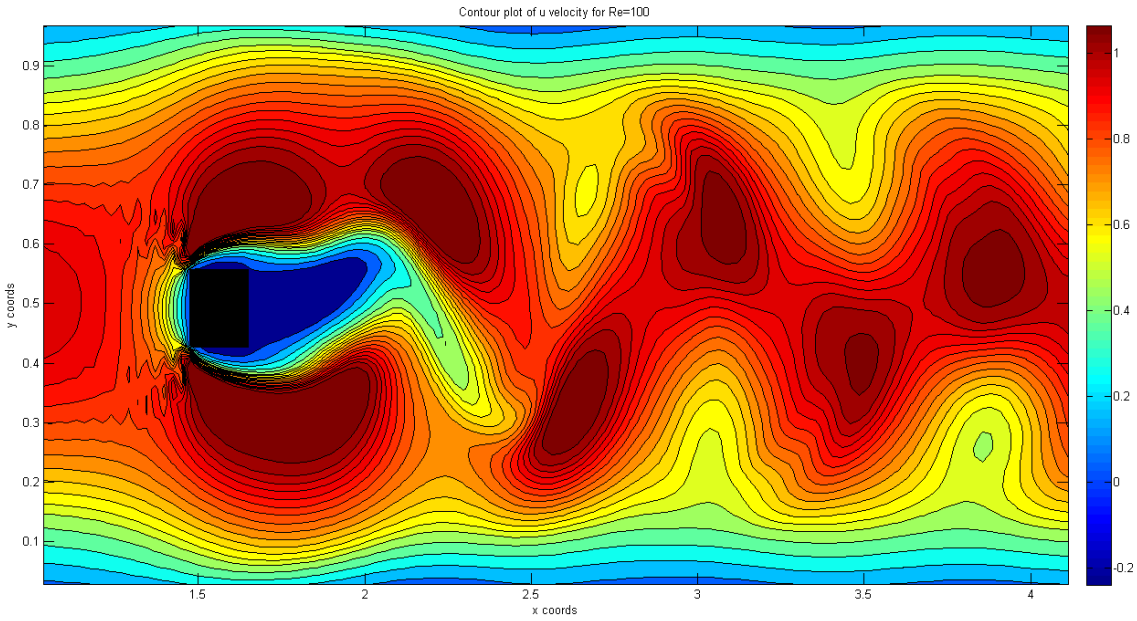


Fig. 64 - Contour plot of u velocity for Re=100 at t=26,02

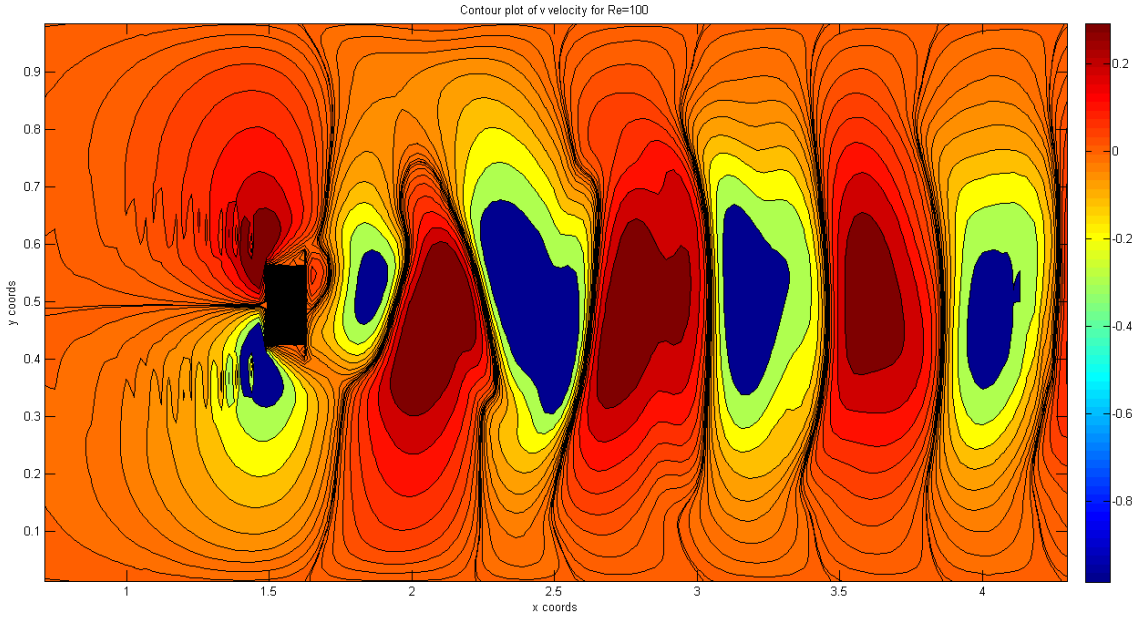


Fig. 65 - Contour plot of v velocity for Re=100 at t=26,02

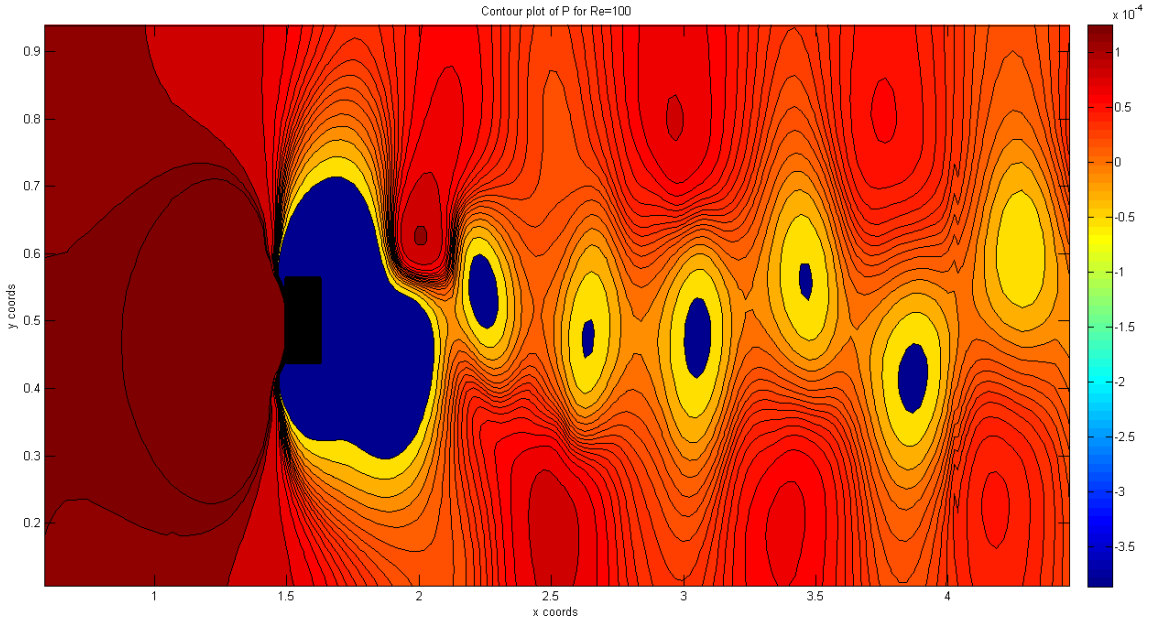


Fig. 66 - Contour plot of pressure for Re=100 at t=26,02

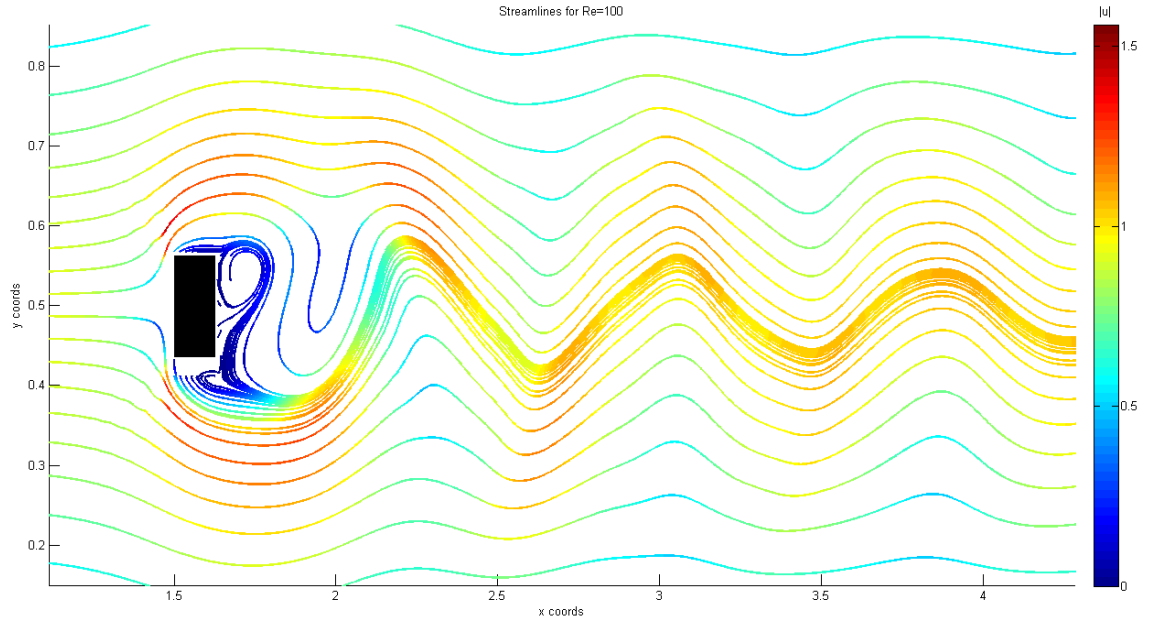


Fig. 67 - Streamlines for Re=100 at t=26,02

The plots from Fig. 64 to Fig. 67 are taken at t=26,02 which corresponds to the minimum lift generated. There is not much new to say except that the can see in all plots the reverse situation from their homonymous at t=24,55, the vortex that detaches in this case is the lower one.

But what do I mean with maximum and minimum lift? A symmetrical profile doesn't generate it. Well, the average lift generated by the profile is 0 but due to the Vortex Shedding we have

periods of positive lift and periods of negative one. The values shown are at the upper and lower points of those peaks. Since the average lift is 0, the parameter used to classify those oscillations is the Strouhal number explained in section **4.6**, which basically allows us to compare different cases and includes the frequency of the phenomenon.

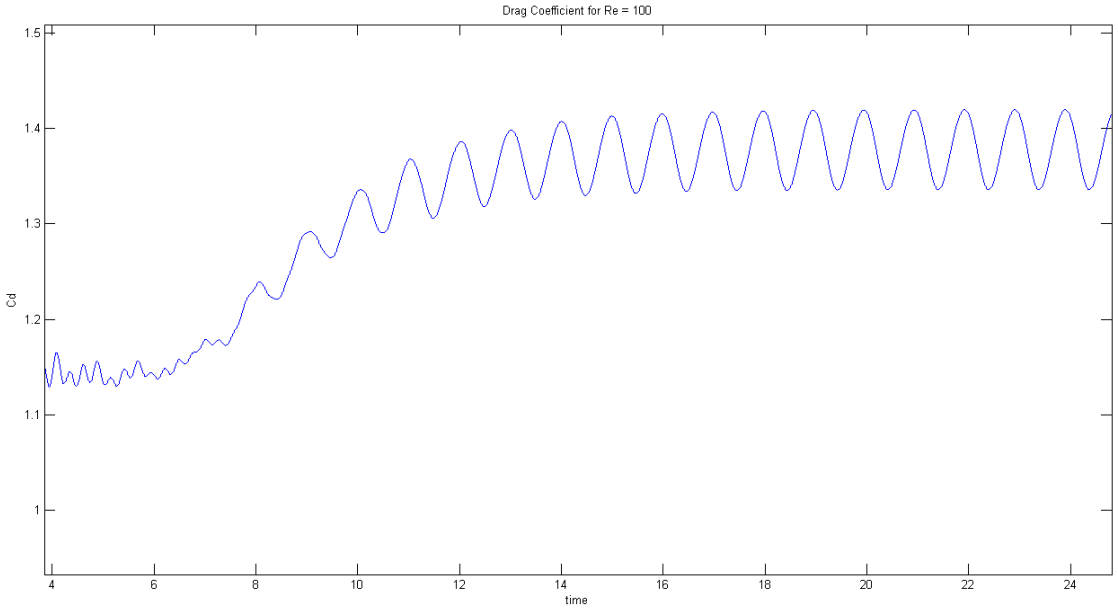


Fig. 68 - Drag coefficient variation with time for Re=100

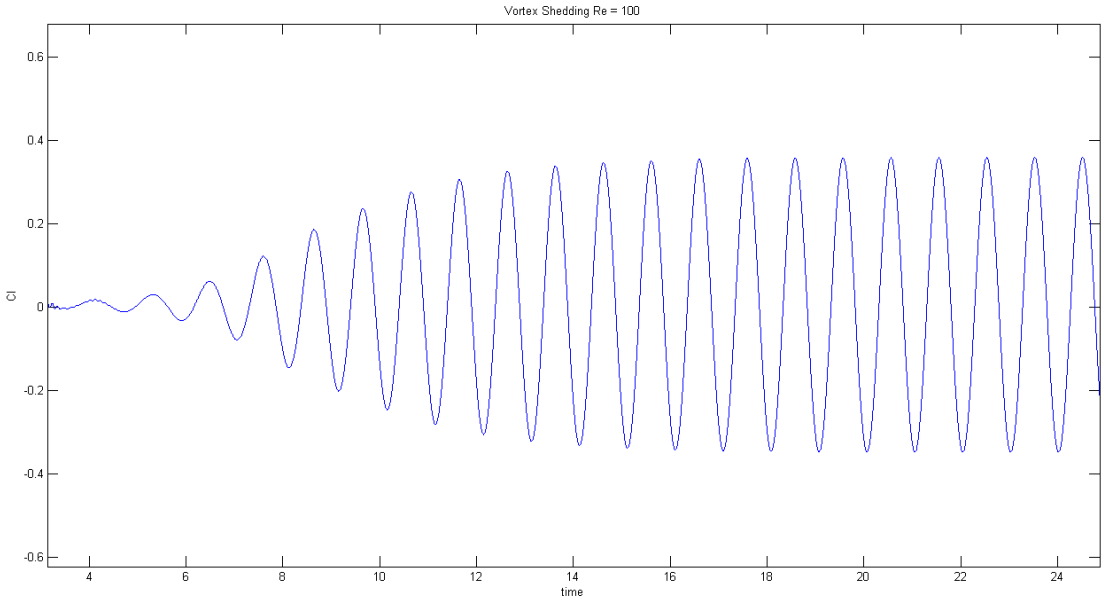


Fig. 69 - Lift coefficient variation with time for Re=100

In Fig. 68 and Fig. 69 we have plotted the drag and lift coefficient variations with time. At around t=4 we can see the settling phase but it quickly moves to the transitory phase and the oscillations

starts. The period between peaks determines the frequency of the phenomenon and we can compute the Strouhal number from it. From both graphs we can also extract the average lift, which is always zero, and the average drag. Those results will be shown in the comparison with the benchmark values.

Re = 60, Re = 150 and Re = 200 comparison

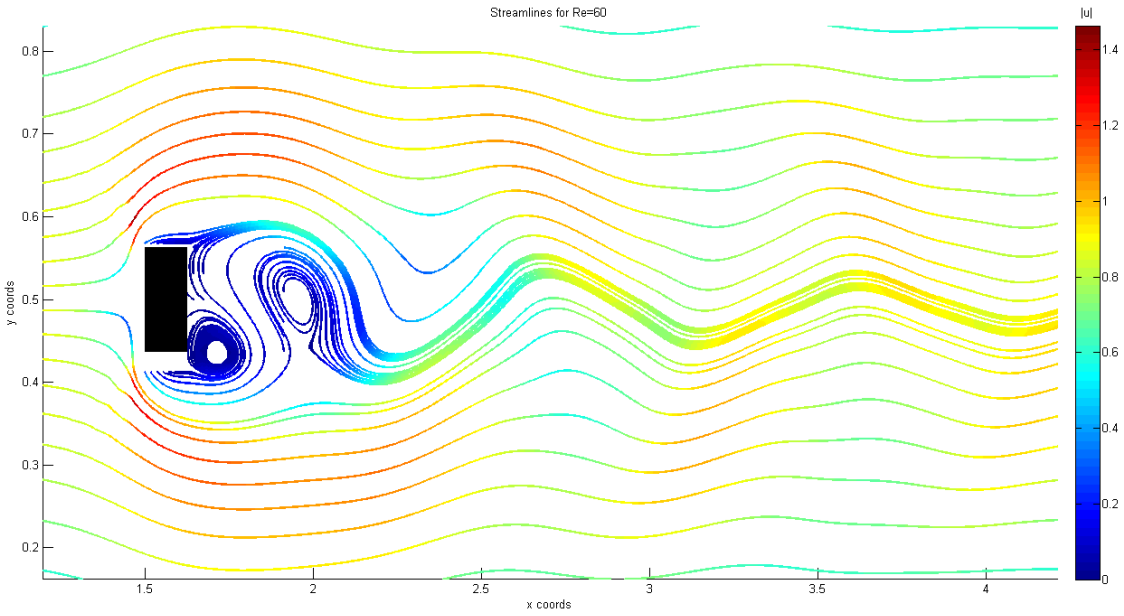


Fig. 70 - Streamlines for Re=60 at t=26,20

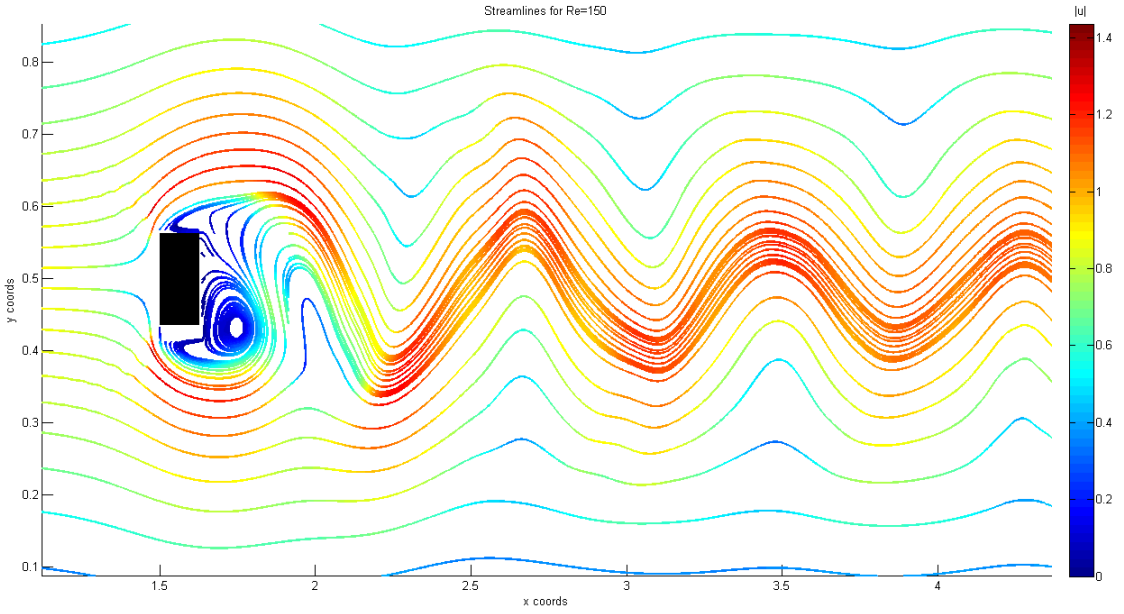


Fig. 71 - Streamlines for Re=150 at t=10,75

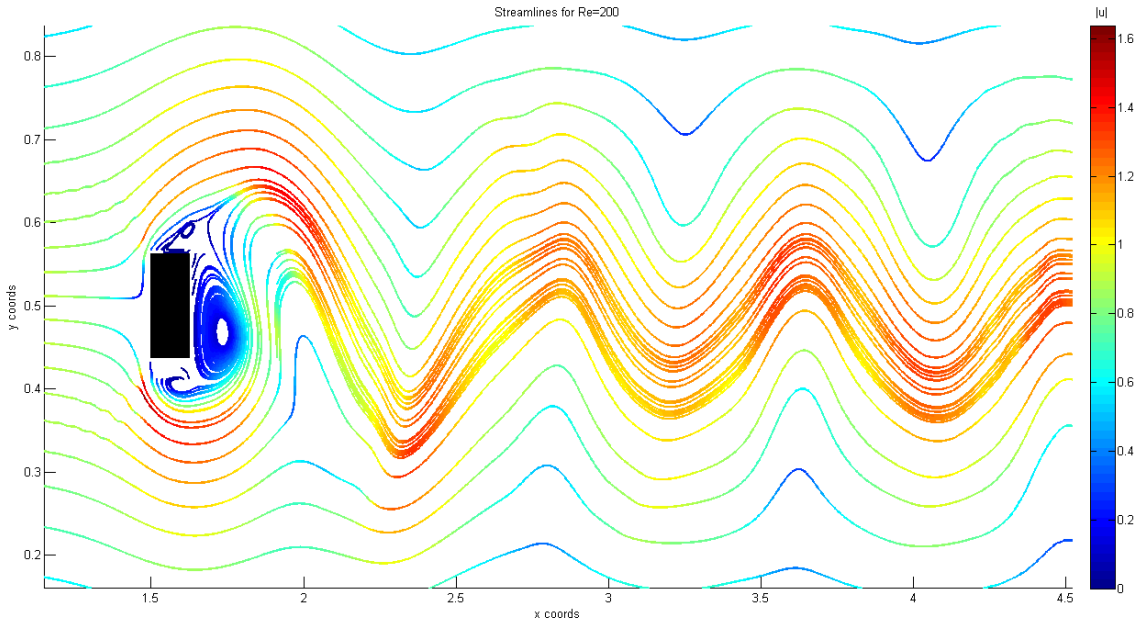


Fig. 72 - Streamlines for Re=200 at t=12,34

In the streamline plots from Fig. 70 to Fig. 72 we see the progression from lower Reynolds to higher ones. All values are at maximum lift points to better compare the evolution with the Reynolds.

We can see for Re=60 at Fig. 70 that both vortices at the back of the cylinder are still quite circular and the one that is detaching is very close to the trailing edge.

For Re=100 and Re=150 (Fig. 63 and Fig. 71 respectively) we start to see the vortices stretch vertically and the separation from the back of the geometry is bigger. The intensity of the vortices also increases, note the change from dark blue at Re=60 to cyan and even green or orange zones as we increase the Reynolds.

For Re=200 at Fig. 72 we appreciate that the lower vortex almost occupies the entirety of the trailing edge and the upper vortex is almost fully detached. We can also see a small vortex appearing on the end of the upper face of the square. This vortex appears as we get closer to the turbulent zone where the boundary layer starts to detach from the profile.

Comparison with reference values

We will compare again the flow around the cylinder with the benchmark, but this time with the streamlines for Re=60 and Re=200 extracted from **[15]**.

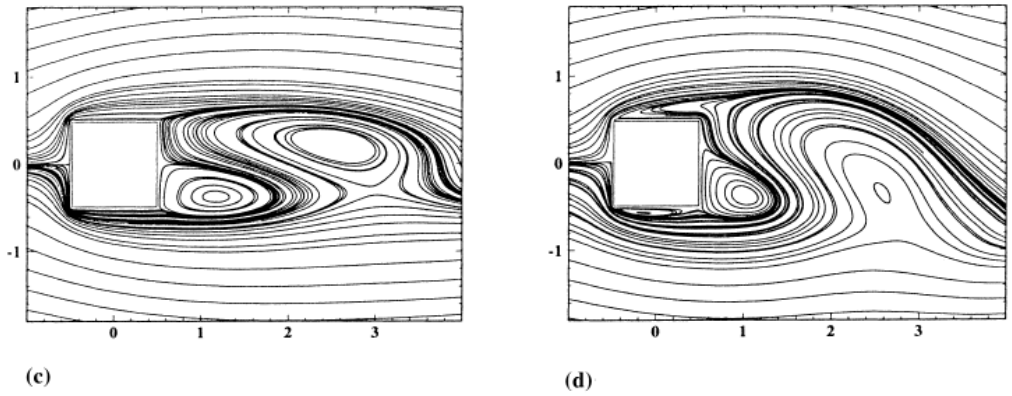


Fig. 73 - Streamlines of reference for different Re numbers. (c) Re=60, (d) Re=200

The behaviour of our simulations is very similar to the one observed on the benchmark results. For Re=60 we have almost identical situations but for Re=200 we see that our vortices are a bit more stretched than the reference values and the boundary layer is a bit more advanced in our case. This is probably due to the difference in mesh density used in both simulations, the benchmark values use meshes of 500x80, 400x240 and 560x340 while our mesh is 160x80. The problem is that our software with a similar mesh would take probably around a week to compute with a portable PC. We are limited by our computational power.

The final comparison we have to undergo is the drag coefficient and the Strouhal number which characterizes the period of oscillation for the Vortex Shedding effect.

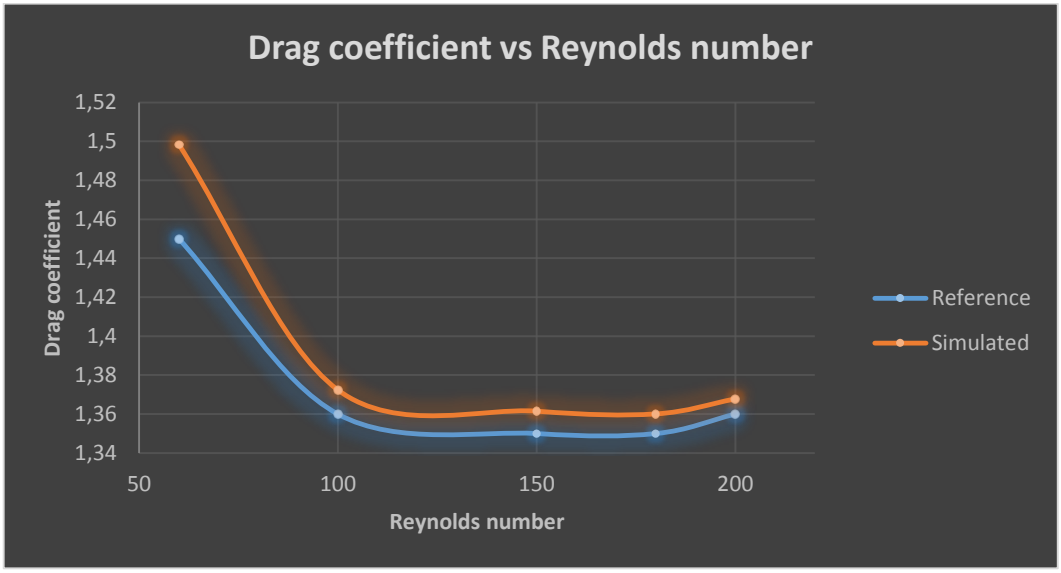


Fig. 74 - Drag coefficient vs Reynolds number comparison

Again we see our average drag coefficient results are slightly higher than the reference ones, but the behaviour of the drag with the Reynolds number is the same. The forces and velocities of our simulations are a bit over the reference ones. As said, this is as results of the vast

difference in meshes used in both simulations being able to characterize every small variation and singularity of the vortices in the trailing edge.

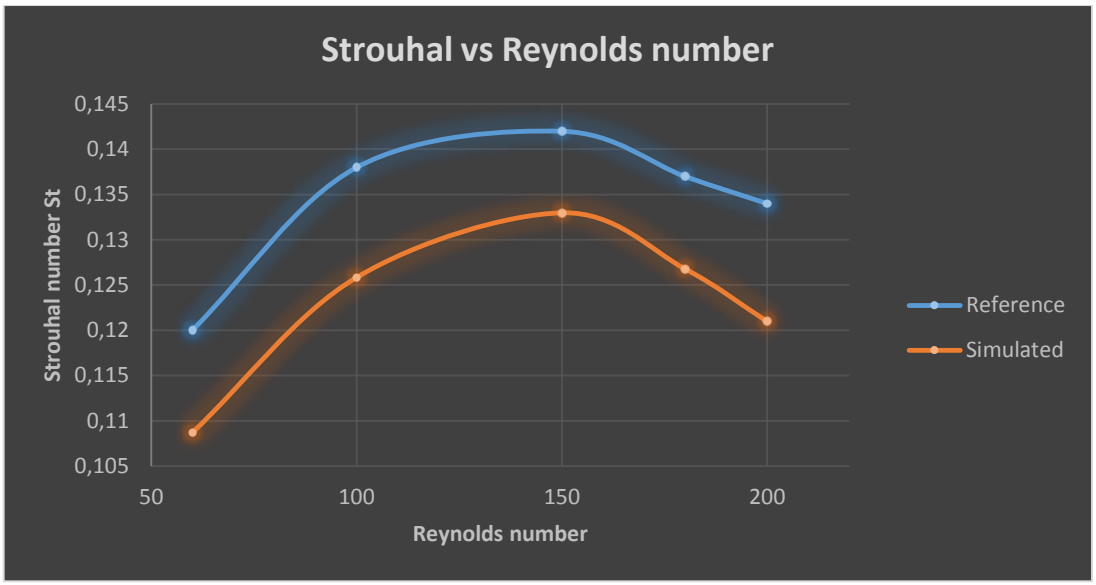


Fig. 75 - Strouhal number vs Reynolds number comparison

For the Strouhal number our results are a bit lower than the benchmark values but yet again with the same tendency with respect to the Reynolds. Despite those differences, we can still consider our results within an acceptable range. Since the diameter D of the square cylinder and the u_∞ inlet velocity is the same for all simulations, a higher period of oscillations results in a lower Strouhal value. Due to the fact that our mesh and software cannot capture as small variations as the benchmark software, not only our Δt is bigger as already mentioned, but also our forces are bigger too. It seems logical that the period of oscillation is a bit higher if the forces have higher peaks for the same conditions, since for a bigger amplitude the transition will take longer.

	SIMULATED		REFERENCE	ERROR	
$\boldsymbol{Re}$	T	St	St	e_{abs}	e_{rel} [%]
60	1,15	0,1087	0,12	-0,0113	-9,4203
100	0,993	0,1258	0,138	-0,0122	-8,8123
150	0,94	0,1330	0,142	-0,0090	-6,3530
180	0,986	0,1268	0,137	-0,0102	-7,4636
200	1,033	0,1210	0,134	-0,0130	-9,7253

Table 7 - Strouhal number error comparison

Due to the fact that the Strouhal number has very small values ranging from around 0,11 to 0,14 even the smallest variation produces a high relative error, making it very hard to achieve high precision results when with an absolute error of only $1 \cdot 10^{-2}$ we have relative errors of more than 5%.

We clearly observe that the Strouhal number is sensible to the smallest variations and in particular to the number of elements used for the mesh which also influences on the Δt of the simulation, it is all related to the mesh and smallest element dimension used.

4.7.3 Final thoughts on the Square Cylinder problem

In conclusion, we can consider our simulations to be good enough for the scope and objective of this project. We would definitely need a more densified mesh and more computational power to be able to simulate with higher precision the Vortex Shedding effect and obtain results equal to the benchmark values. Despite the differences observed in the transitory phase, the tendency and behaviour of all the parameters are the same as in the reference ones which gives us confidence on the cohesion and coherence of the physical and mathematical models implemented in our software.

We have obtained more results than the ones included in this chapter which can be found in its entirety on ***Annex A***.

5 Applications, future lines and conclusions

5.1 Applications

The software developed even though is not state of the art, it can reproduce and calculate with very good precision a significant amount of cases of interest.

The first big field of applications is for academical purposes. It serves as an introduction to CFD, giving an insight at how the fluid dynamics is modelled and programmed. It shows all the key concepts needed: Physics, maths, programming, meshing, boundary conditions, post-processing and results treatment and plotting. Once that knowledge is acquired it allows us to reproduce simple but illustrative physical experiments, some of them such as the Smith-Hutton problem or the Driven Cavity have been shown in section **3.3**.

The main application of this software is to be able to compute the behaviour of alar profiles and other 2D geometries of interest for non-turbulent Reynold values. The software provides us with the velocity profile, lift and drag values which are key aerodynamical parameters for design. Pretty much it works as our own virtual wind tunnel. Since it cannot compute turbulent cases it not yet suited for many commercial or serious purposes but it can serve for a lot of interesting studies. It can allow us to simulate alar profiles for toy/modelling planes or some radio controlled ones, we could also simulate simple car profiles for models or toys and in essence any small objects at low velocities.

5.2 Future lines

One of the possible upgrades for our software would be to introduce the temperature field. The temperature field could potentially allow us two new features. The first one would be to simulate cases governed by Natural Convection. The second one would be to move from constant physical properties to variable ones, making parameters such as density or viscosity function of the temperature. In many cases the complexity added is not worth the difference in the precision of results but yet it is a possible functionality.

The next logical step forward would be to move from 2D cases to 3D domains, allowing us to compute non-turbulent 3D cases. This would allow us to introduce full objects, such as an entire car, a plane or just an entire wing instead of an alar profile. Note that since it would still be non-turbulent we have to be very careful with wing tip vortices or similar phenomena that could potentially reduce the Reynolds number for those simulations.

The final and best potential of them all would be to introduce what is known as Large Eddy Simulations (LES models), a mathematical model for turbulent flows. In rough outlines, basically consists in filtering (in space) the original Navier-Stokes Equations in order to reduce the dynamical complexity of the system. This results into a new set of partial differential equations

for the filtered velocity. They are identical to Navier-Stokes Equations except for the subgrid-stress tensor that needs to be modelled in terms of the filtered velocity. The resolved part of the field represent the "large" eddies, while the subgrid part of the velocity represent the "small scales" whose effect on the resolved field is included through the subgrid-scale model. For more information on the LES model refer to **[18]**.

Once we have a 3D domain capable of solving turbulent flows this is pretty much as good as it can get. The limiting parameters now would basically be the computational power of our computers.

5.3 Conclusions

Our main objective was to create a self-built software capable of solving the Navier-Stokes Equations for 2D non turbulent cases. With the resolution of the Driven Cavity and the Square Cylinder we have proven that our software is capable of solving those problems and give proper results for those cases. Despite not achieving all the precision wanted in the higher Reynolds order simulations, we have concluded that this is due to a lack of computational power and not because of defects on the software. Meaning that only by using our software on a more powerful computer capable of solving very dense meshes (and by capable we mean in a reasonable amount of time) would achieve very high standards of solutions for the cases of interest.

In conclusion, we can say the objective of this project is achieved and the software can be used for solving 2D geometry cases for non turbulent flow.

6 Economic and environmental study

6.1 Economic study

This project has been carried out by a single engineer (myself) throw the course of 8 months, working on average 5 days a week, 6 hours a day. Assuming the average wage of a junior engineer is 10€/hour, we can estimate the overall human cost.

WORKING HOURS [H]	AVERAGE PRICE [€/H]	COST [€]
960	10	9600

Table 8 - Human resources cost estimation

The hardware used to perform the project was a portable computer with a cost of 1050€. Apart from the hardware, several softwares were used including: Microsoft Office package (69€), Matlab with student licence (69€) and the programming platform and compiler DevC++ (free software).

HUMAN RESOURCES COST	9600 €
HARDWARE COST	1050 €
SOFTWARE COST	138 €
TOTAL COST	**10788 €**

Table 9 - General cost estimation

6.2 Environmental impact

We could say that the environmental impact of our project is practically 0. We have only consumed the electricity of one portable computer and around 20 to 30 paper sheets. Also take into note that a CFD software consumes far much less energy than a real wind tunnel, and there is no use of materials making the test parts or the energy of the factory or machine that produces those parts.

7 Bibliographic references

[1] *A. Thom. The Flow Past Circular Cylinders at Low Speeds.* Proc. Royal Society*, A141, pp. 651-666. London, 1933.*

[2] M. Kawaguti. Numerical Solution of the NS Equations for the Flow Around a Circular Cylinder at Reynolds Number 40. Journal of Phy. Soc. Japan, vol. 8, pp. 747-757. Japan, 1953.

[3] A. Jameson. Computational Fluid Dynamics past, present and future. Stanford University. 2011.

[4] John D. Anderson. Computational Fluid Dynamics The basics with Applications. MacGraw-Hill, Inc. United States, 1995.

[5] Jonathan R. Shewchuk. An introduction to the Conjugate Gradient Method Without the Agonizing Pain. School of Computer Science Carnegie Mellon University. United States, 1994.

[6] W. Don, D. Gottlieb, C. Shu, O. Schilling, L. Jameson. Numerical Convergence Study of Nearly-Incompressible, Inviscid Taylor-Green Vortex Flow. Division of Applied Mathematics, Brown University. University of California, Lawrence Livermore National Laboratory. United States, 2002.

[7] Numerical Solution of Convection. Centre Tecnològic de Transferència de Calor. Universitat Politècnica de Catalunya, Spain.

[8] R. L. Burden, J.D. Faires. Iterative Techniques in Matrix Algebra, Jacobi & Gauss-Seidel Iterative Techniques II. Numerical Analysis (9^{th} Edition). Brooks/Cole, 2011.

[9] Introduction to the Fractional Step Method. Centre Tecnològic de Transferència de Calor. Universitat Politècnica de Catalunya, Spain.

[10] Patrick J. Roache. Code Verification by the Method of Manufactured Solutions. J. Fluids Eng, vol. 124, pp. 4-10. 2001.

[11] Suhas V. Patankar. Numerical Heat Transfer and Fluid Flow. Hemisphere Publishing Corporation, McGraw-Hill Book Company, 1980.

[12] A Two-dimensional Steady Convection-Diffusion Equation: the Smith-Hutton problem. Centre Tecnològic de Transferència de Calor. Universitat Politècnica de Catalunya, Spain.

[13] F. X. Trias, M. Soria, A. Oliva, C. D. Pérez-Segarra. Direct numerical simulations of two- and three-dimensional turbulent natural convection flows in a differentially heated cavity of aspect ratio 4. J. Fluid Mech, vol. 586, pp. 259-293. Cambridge University Press. United Kingdom, 2007.

[14] U. Ghia, K. N. Ghia, C. T. Shin. High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method. Journal of Computational Physics, vol. 48, pp. 387-411. 1982.

[15] M. Breuer, J. Bernsdorf, T. Zeiser, F. Durst. Accurate computations of the laminar flow past a square cylinder base don two different methods: lattice-Boltzmann and finite-volume. International Journal of Head and Fluid Flow, vol. 21, pp. 186-196. 2000

[16] A. Sohankar, C. Norberg, L. Davidson. Low-Reynolds-Number Flow Around a Square Cylinder at Incidence: Study of Blockage, Onset of Vortex Shedding and Outlet Boundary Condition. International Journal for Numerical Methods in Fluids, vol. 26, pp. 29-56. 1998.

[17] R. M. Cummings, M. Giles, G. Shrinivas. Analysis of the Elements of Drag in Three-Dimensional Viscous and Inviscid Flows. 14th Applied Aerodynamics Conference. AIAA-96-2482-CP. 1996.

[18] Large-Eddy Simulations of turbulent incompressible flows in a nutshell. Centre Tecnològic de Transferència de Calor. Universitat Politècnica de Catalunya. Spain, 2012.

[19] N. N. Yanenko. The Method of Fractional Steps. Springer-Verlag, 1971.

Webgraphy:

[20] *http://www.cfd-online.com* , visited on August, 8th 2014.

[21]*http://stackoverflow.com/* , visited on June, 18th 2014.

[22] *http://www.mathworks.es/es/help/matlab/* , visited September, 14th 2014.

Annex A

In this annex we will include all the final plots for the results of the Square Cylinder problem from Chapter **4** that were not included for clarity and not flood the reader with a lot of graphs with very similar content and not much to extract from them.

Re = 1

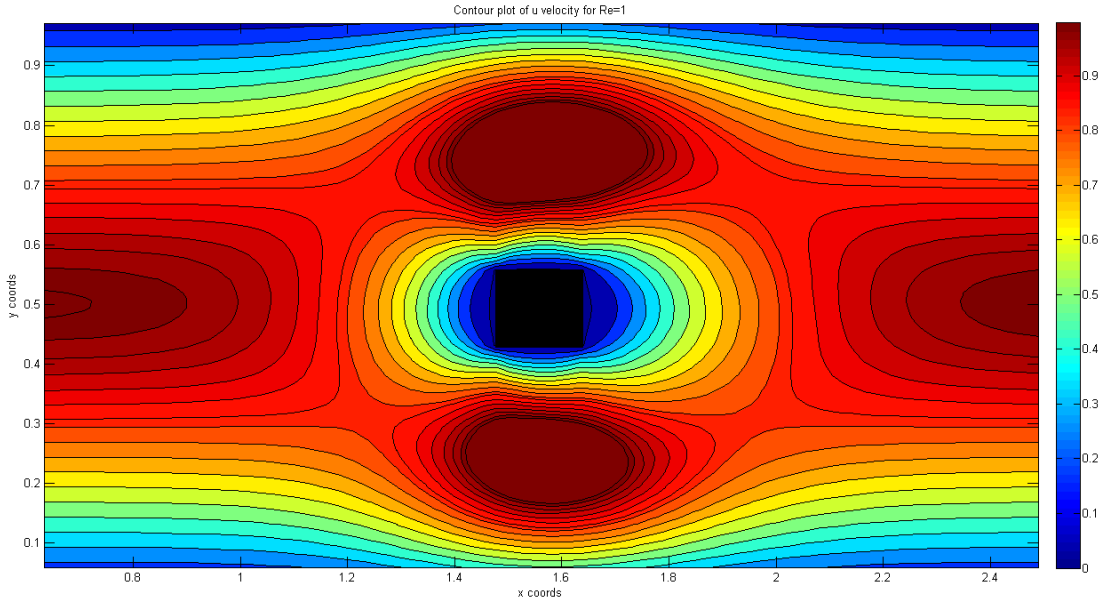


Fig. I - Contour plot of u velocity for Re=1

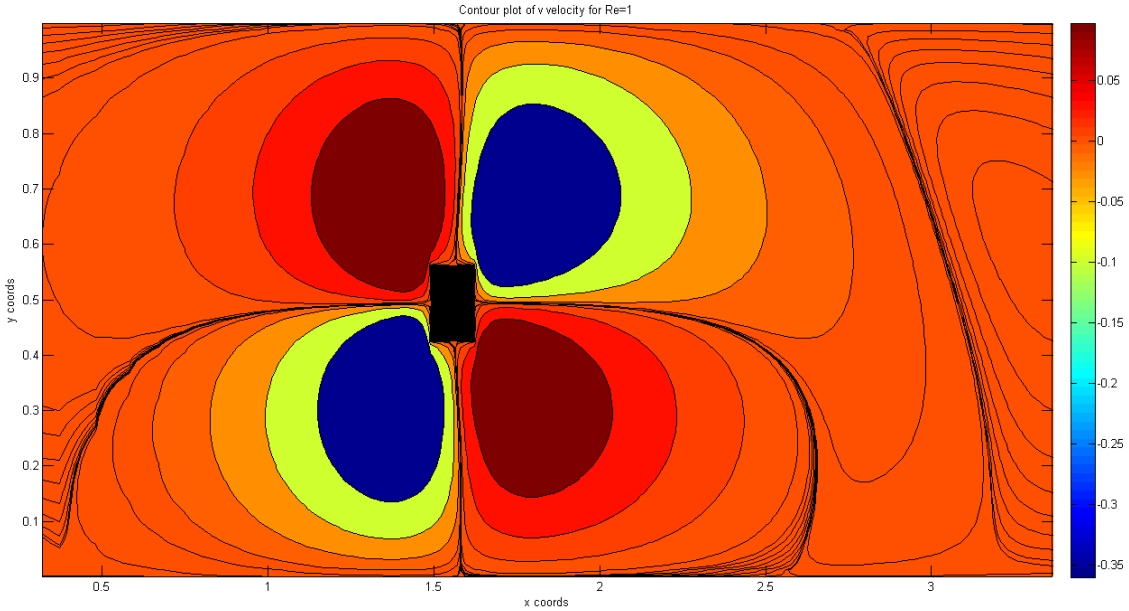


Fig. II - Contour plot of v velocity for Re=1

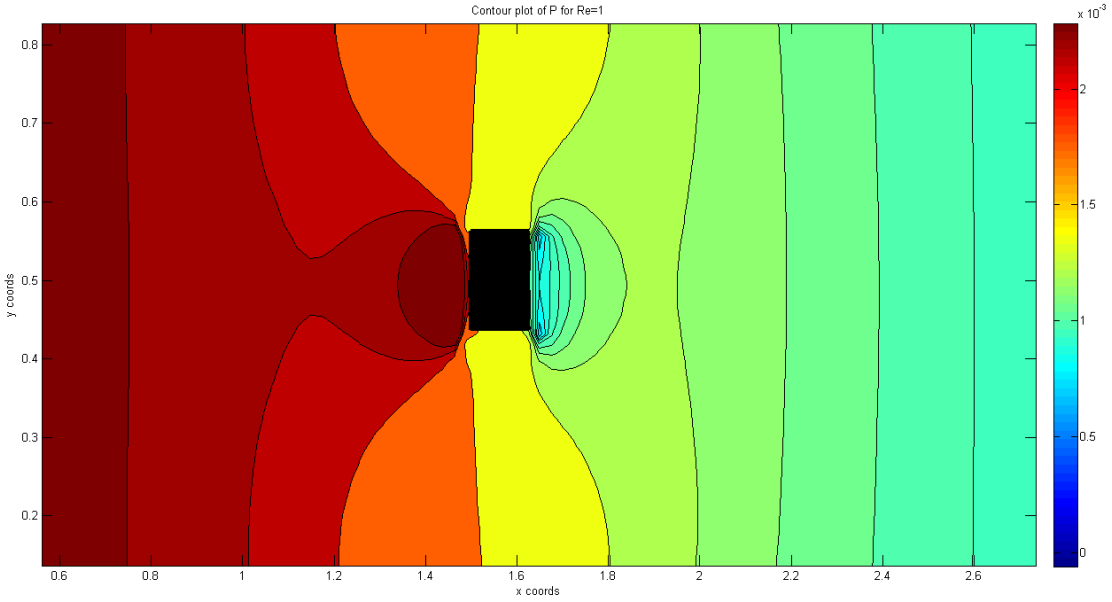


Fig. III- Contour plot of pressure for Re=1

Re = 10

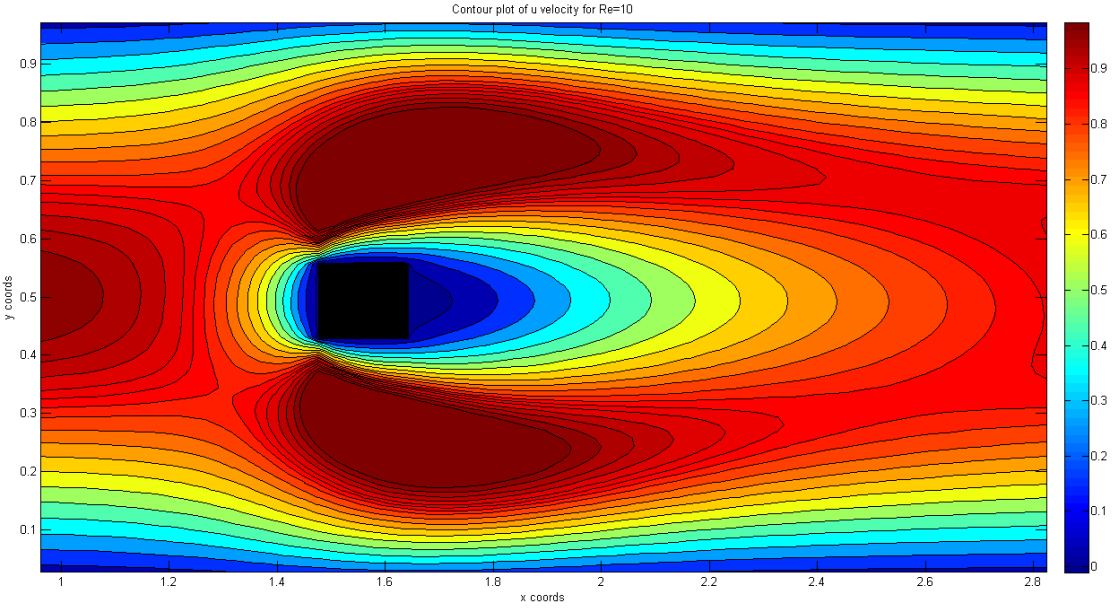


Fig. IV - Contour plot of u velocity for Re=10

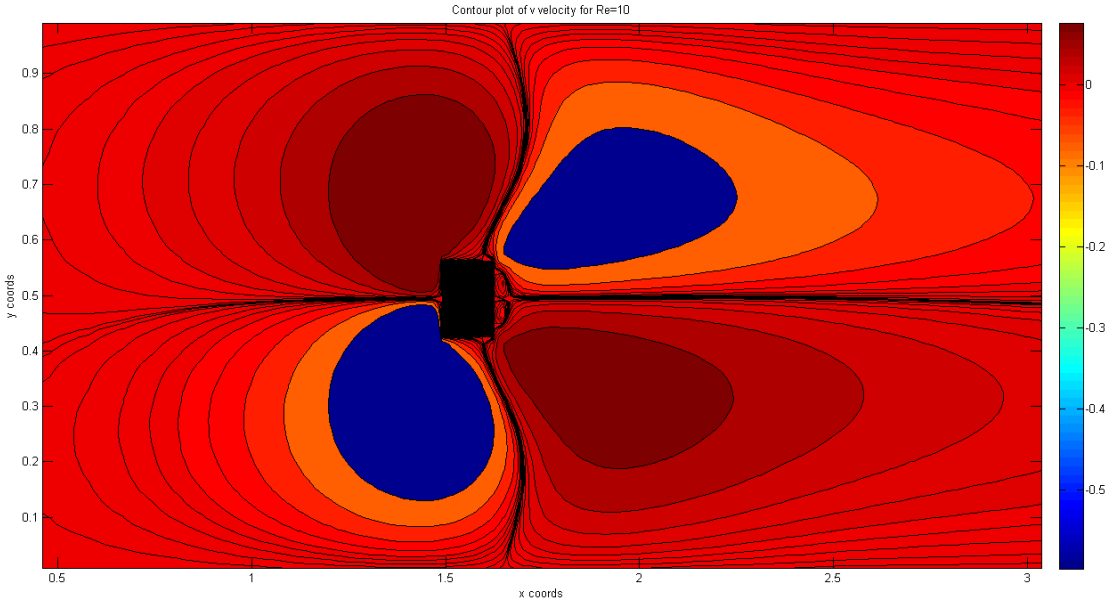


Fig. V - Contour plot of v velocity for Re=10

Re = 20

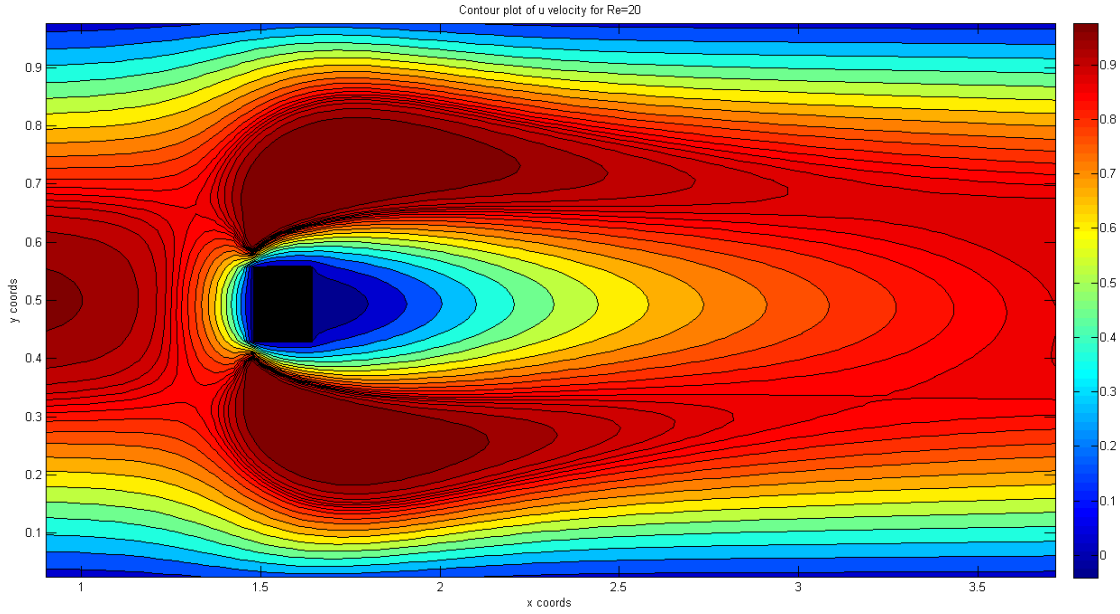


Fig. VI - Contour plot of u velocity for Re=20

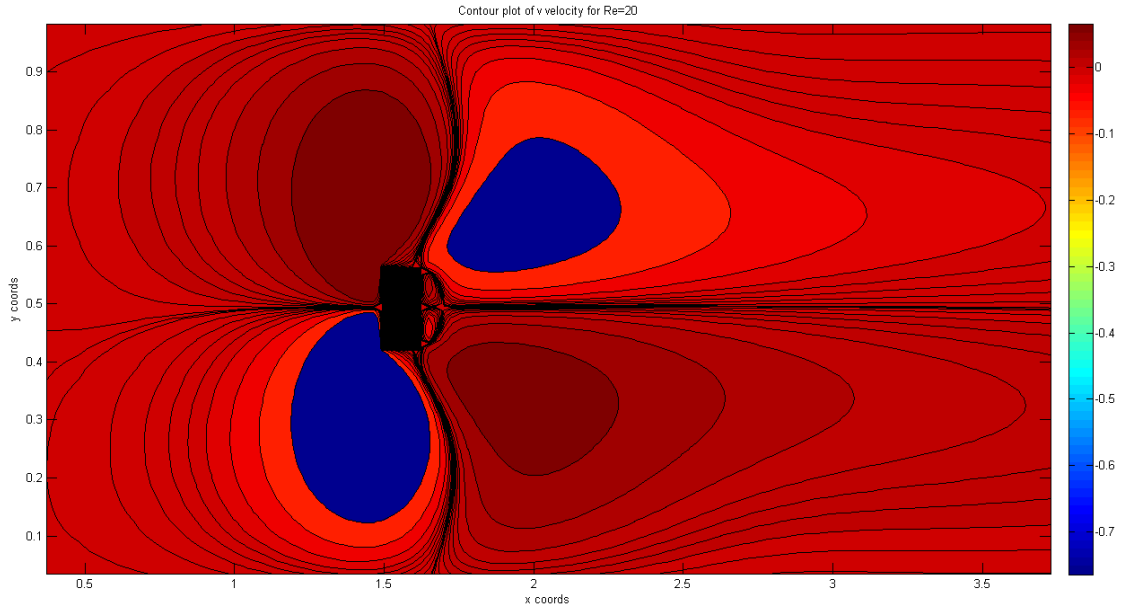


Fig. VII - Contour plot of v velocity for Re=20

Re = 60

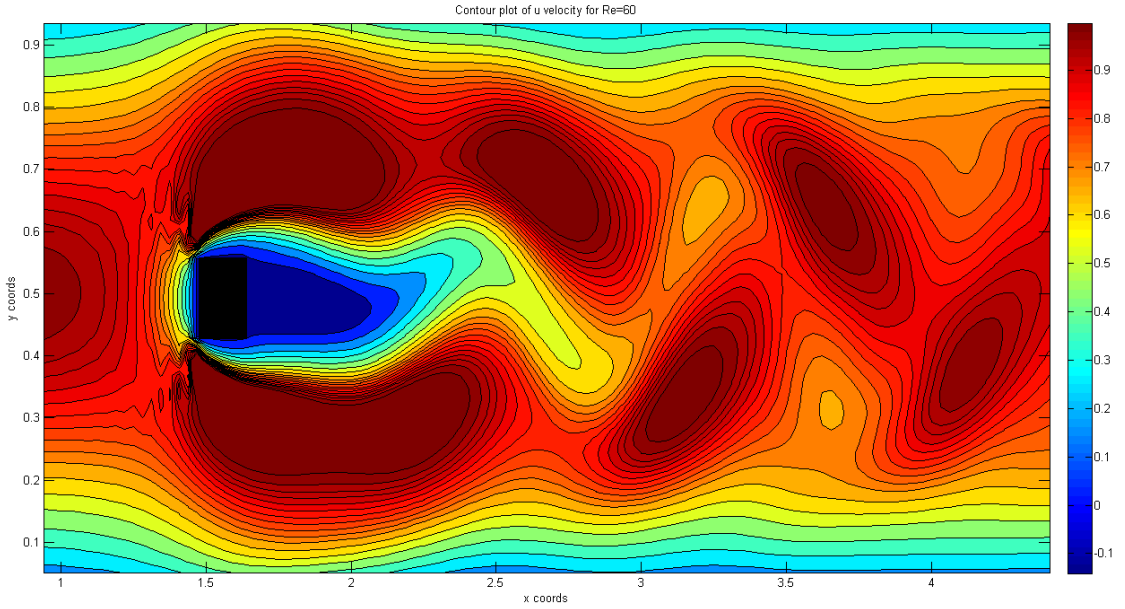


Fig. VIII - Contour plot of u velocity for Re=60 at t=26,20

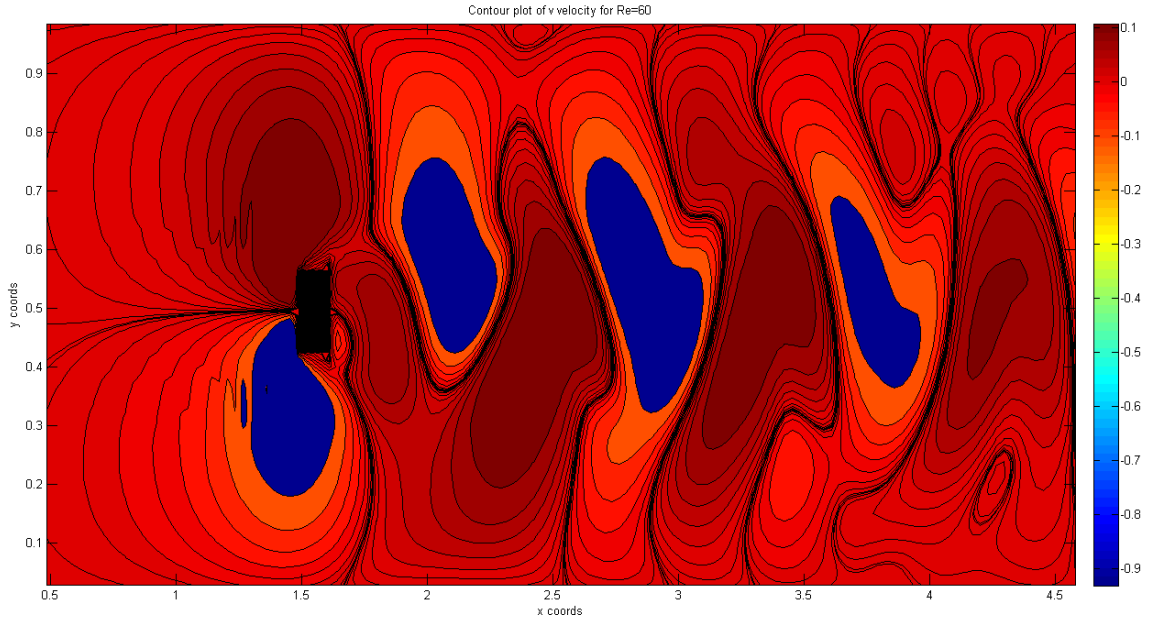


Fig. IX - Contour plot of v velocity for Re=60 at t=26,20

Re = 150

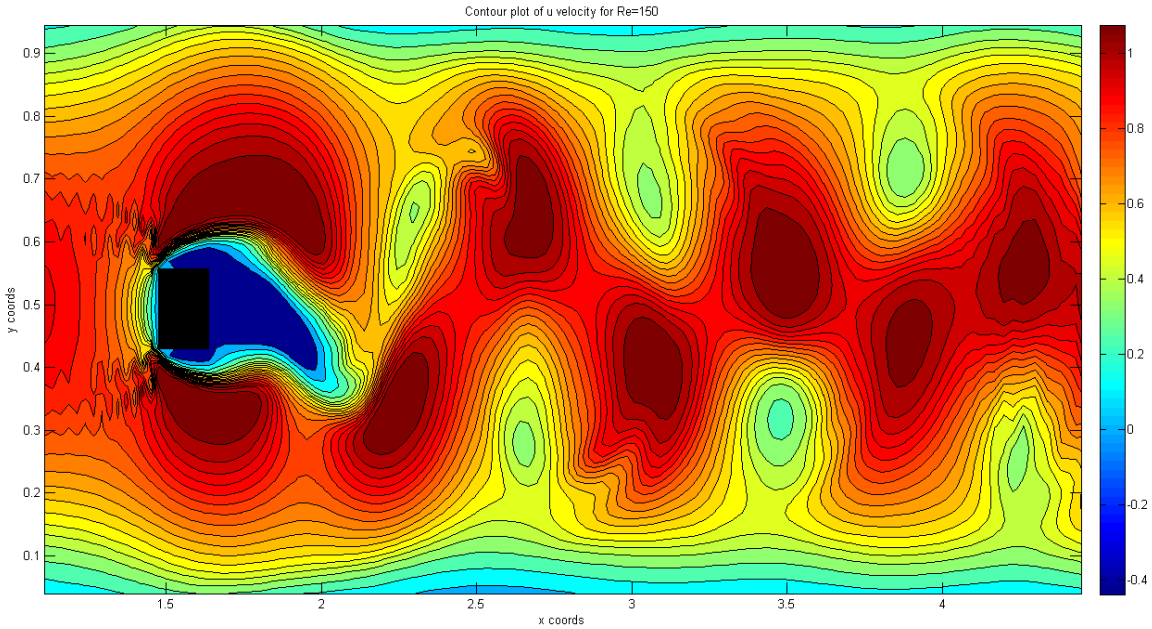


Fig. X - Contour plot of u velocity for Re=150 at t=10,75

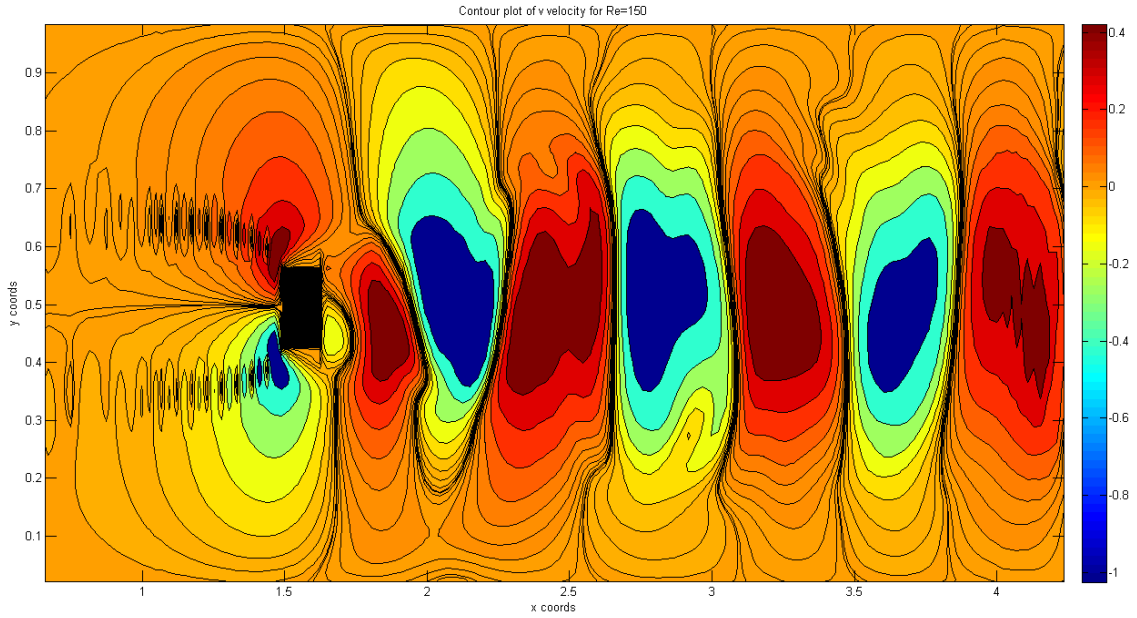


Fig. XI - Contour plot of v velocity for Re=150 at t=10,75

Re = 180

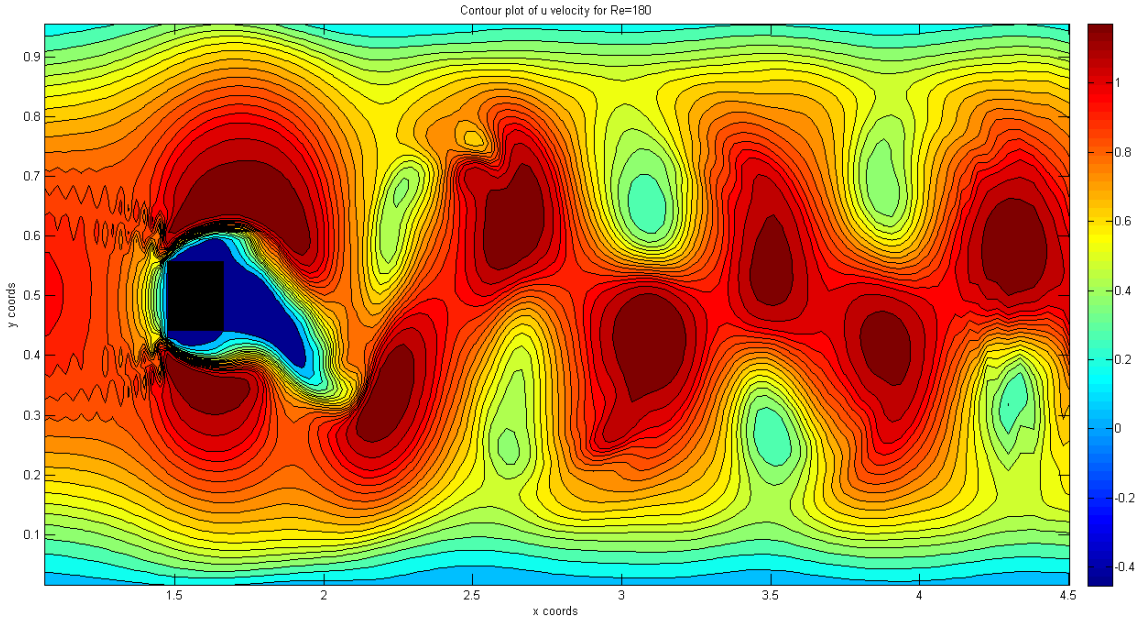


Fig. XII - Contour plot of u velocity for Re=180 at t=16,12

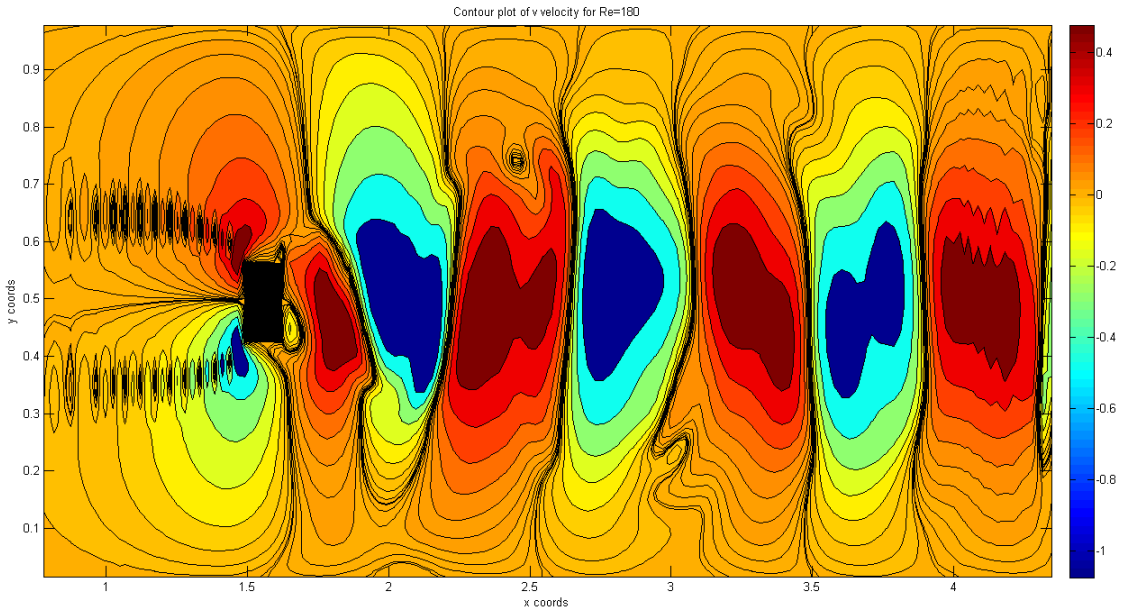


Fig. XIII - Contour plot of v velocity for Re=180 at t=16,12

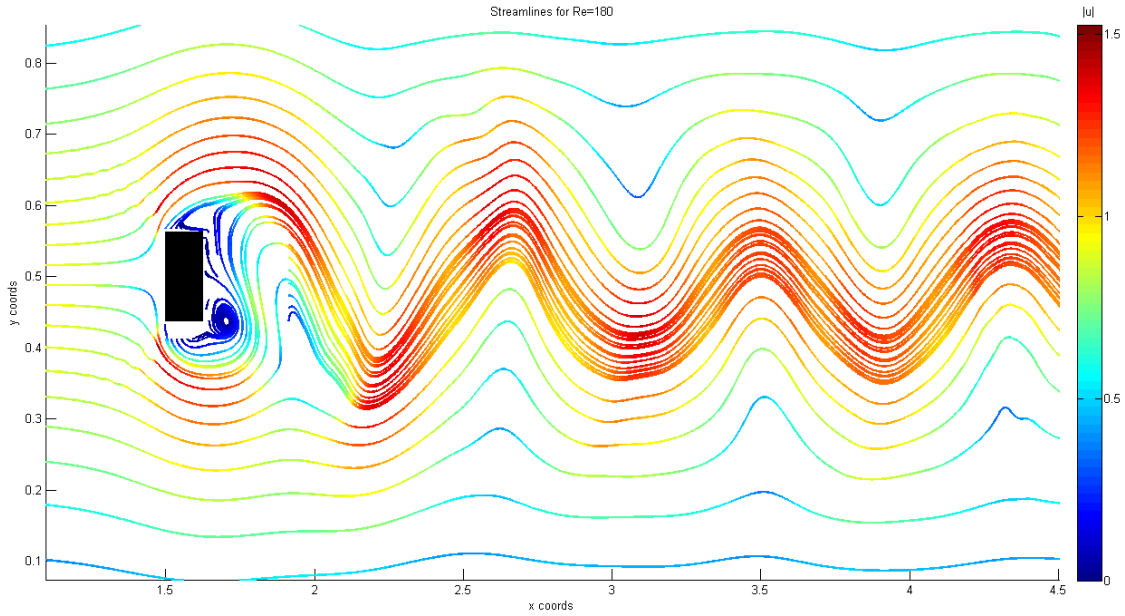


Fig. XIV - Streamlines for Re=180 at t=16,12

Re = 200

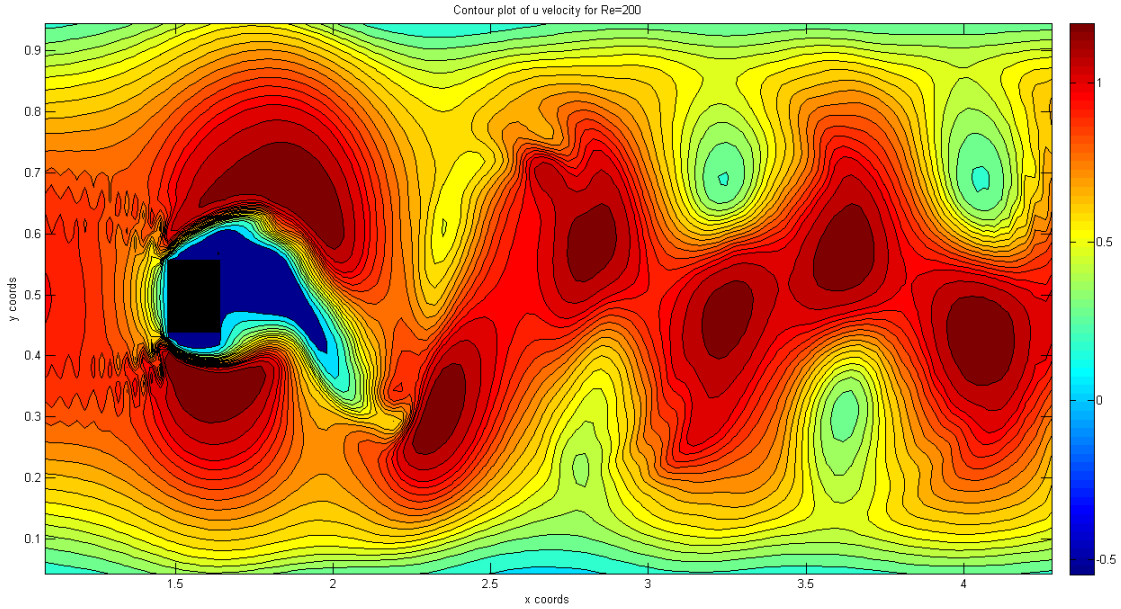


Fig. XV - Contour plot of u velocity for Re=200 at t=12,34

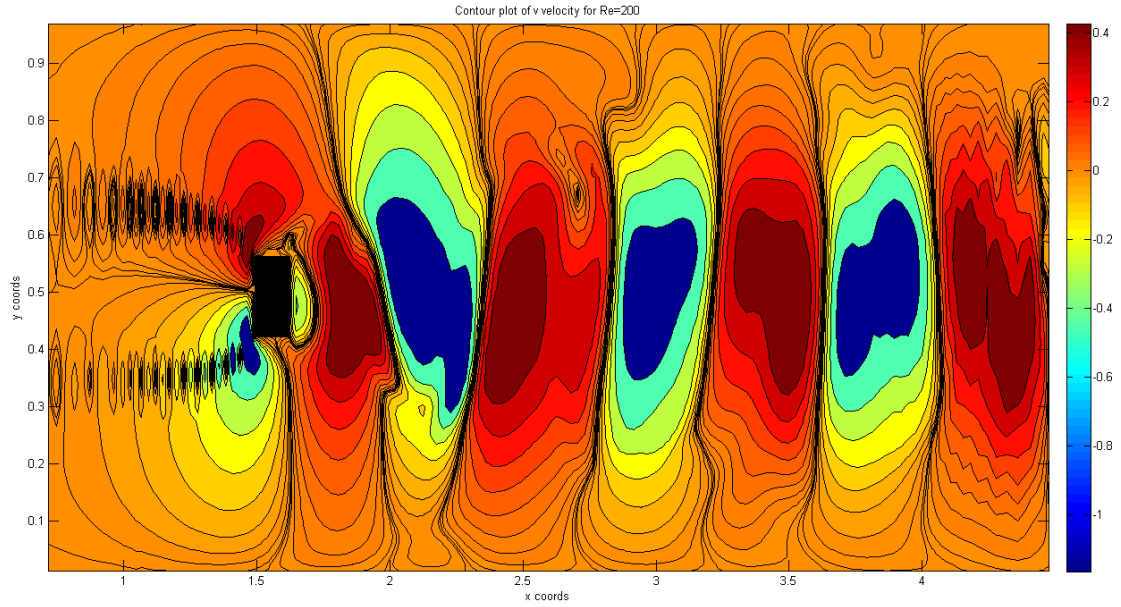


Fig. XVI - Contour plot of v velocity for Re=200 at t=12,34

Annex B

In this annex we will include some of parts of the code used in our software. The objective of this annex is not to distribute a complete copy of the software developed but to give insight on how to implement some of the mathematical tools used or certain schemes of interest.

Centred mesh creation

```
void casoSquareCyl::crear_malla()     //Function that generates the mesh, leaving an empty row and column on each side
{
   unsigned int i=0, j=0;
   double dx, dy;
   double x, dpw, xe, xw, dpe;
   double y, dps, yn, ys, dpn;

   for (j=1; j<Ny+3; j++)
   {
      for (i=1; i<Nx+3; i++)
      {
               y = set_coord_Y(j);
               x = set_coord_X(i);
               Vc[pos(i,j)].setx(x);
                Vc[pos(i,j)].sety(y);
      }
   }


   for (j=0; j<Ny+4; j++)
   {
      for (i=0; i<Nx+4; i++)
      {
         if ((i==1)&&(j!=1)&&(j!=Ny+2)&&(j!=0)&&(j!=Ny+3))  //Left
         {
               if(j==2)
               {
                        y = Vc[pos(i,j)].gety();
                        ys = Vc[pos(i,j-1)].gety();
                        yn = Vc[pos(i,j+1)].gety();
                        dpn = (yn-y)/2.0;
                        dps = (y-ys);
                        dy = dpn+dps;
```

```
        }
        else if(j==Ny+1)
        {
                y = Vc[pos(i,j)].gety();
                ys = Vc[pos(i,j-1)].gety();
                yn = Vc[pos(i,j+1)].gety();
                dpn = (yn-y);
                dps = (y-ys)/2.0;
                dy = dpn+dps;
        }
        else
        {
                y = Vc[pos(i,j)].gety();
                ys = Vc[pos(i,j-1)].gety();
                yn = Vc[pos(i,j+1)].gety();
                dpn = (yn-y)/2.0;
                dps = (y-ys)/2.0;
                dy = dpn+dps;
        }

        Vc[pos(i,j)].setdx(0);
        Vc[pos(i,j)].setdy(dy);
         Vc[pos(i,j)].setdpe(0);
        Vc[pos(i,j)].setdpw(0);
         Vc[pos(i,j)].setdpn(dpn);
        Vc[pos(i,j)].setdps(dps);
}
else if ((j==1)&&(i!=1)&&(i!=Nx+2)&&(i!=0)&&(j!=Nx+3))  //Bottom
{
        if(i==2)
        {
                x = Vc[pos(i,j)].getx();
                xw = Vc[pos(i-1,j)].getx();
                xe = Vc[pos(i+1,j)].getx();
                dpe = (xe-x)/2.0;
                dpw = (x-xw);
                dx = dpw+dpe;
        }
        else if(i==Nx+1)
```

```
        {
                x = Vc[pos(i,j)].getx();
                xw = Vc[pos(i-1,j)].getx();
                xe = Vc[pos(i+1,j)].getx();
                dpe = (xe-x);
                dpw = (x-xw)/2.0;
                dx = dpw+dpe;
        }
        else
        {
                x = Vc[pos(i,j)].getx();
                xw = Vc[pos(i-1,j)].getx();
                xe = Vc[pos(i+1,j)].getx();
                dpe = (xe-x)/2.0;
                dpw = (x-xw)/2.0;
                dx = dpw+dpe;
        }

    Vc[pos(i,j)].setdx(dx);
    Vc[pos(i,j)].setdy(0);
    Vc[pos(i,j)].setdpe(dpe);
    Vc[pos(i,j)].setdpw(dpw);
    Vc[pos(i,j)].setdpn(0);
    Vc[pos(i,j)].setdps(0);
}
else if ((i==1)&&(j==1))    //Corner 1
{
    Vc[pos(i,j)].setdx(0);
    Vc[pos(i,j)].setdy(0);
    Vc[pos(i,j)].setdpe(0);
    Vc[pos(i,j)].setdpw(0);
    Vc[pos(i,j)].setdpn(0);
    Vc[pos(i,j)].setdps(0);
}
else if ((i==Nx+2)&&(j!=1)&&(j!=Ny+2)&&(j!=0)&&(j!=Ny+3))   //Right
{
    if(j==2)
        {
                y = Vc[pos(i,j)].gety();
```

```
                ys = Vc[pos(i,j-1)].gety();
                yn = Vc[pos(i,j+1)].gety();
                dpn = (yn-y)/2.0;
                dps = (y-ys);
                dy = dpn+dps;
        }
        else if(j==Ny+1)
        {
                y = Vc[pos(i,j)].gety();
                ys = Vc[pos(i,j-1)].gety();
                yn = Vc[pos(i,j+1)].gety();
                dpn = (yn-y);
                dps = (y-ys)/2.0;
                dy = dpn+dps;
        }
        else
        {
                y = Vc[pos(i,j)].gety();
                ys = Vc[pos(i,j-1)].gety();
                yn = Vc[pos(i,j+1)].gety();
                dpn = (yn-y)/2.0;
                dps = (y-ys)/2.0;
                dy = dpn+dps;
        }

    Vc[pos(i,j)].setdx(0);
    Vc[pos(i,j)].setdy(dy);
    Vc[pos(i,j)].setdpe(0);
    Vc[pos(i,j)].setdpw(0);
    Vc[pos(i,j)].setdpn(dpn);
    Vc[pos(i,j)].setdps(dps);
}
else if ((j==Ny+2)&&(i!=1)&&(i!=Nx+2)&&(i!=0)&&(i!=Nx+3))   //Top
{
    if(i==2)
        {
                x = Vc[pos(i,j)].getx();
                xw = Vc[pos(i-1,j)].getx();
                xe = Vc[pos(i+1,j)].getx();
```

```
                dpe = (xe-x)/2.0;
                dpw = (x-xw);
                dx = dpw+dpe;
        }
        else if(i==Nx+1)
        {
                x = Vc[pos(i,j)].getx();
                xw = Vc[pos(i-1,j)].getx();
                xe = Vc[pos(i+1,j)].getx();
                dpe = (xe-x);
                dpw = (x-xw)/2.0;
                dx = dpw+dpe;
        }
        else
        {
                x = Vc[pos(i,j)].getx();
                xw = Vc[pos(i-1,j)].getx();
                xe = Vc[pos(i+1,j)].getx();
                dpe = (xe-x)/2.0;
                dpw = (x-xw)/2.0;
                dx = dpw+dpe;
        }

    Vc[pos(i,j)].setdx(dx);
    Vc[pos(i,j)].setdy(0);
    Vc[pos(i,j)].setdpe(dpe);
    Vc[pos(i,j)].setdpw(dpw);
    Vc[pos(i,j)].setdpn(0);
    Vc[pos(i,j)].setdps(0);
}
else if ((i==Nx+2)&&(j==Ny+2))  //Corner 4
{
    Vc[pos(i,j)].setdx(0);
    Vc[pos(i,j)].setdy(0);
    Vc[pos(i,j)].setdpe(0);
    Vc[pos(i,j)].setdpw(0);
    Vc[pos(i,j)].setdpn(0);
    Vc[pos(i,j)].setdps(0);
}
```

```
else if ((i==1)&&(j==Ny+2))    //Corner 2
{
   Vc[pos(i,j)].setdx(0);
   Vc[pos(i,j)].setdy(0);
   Vc[pos(i,j)].setdpe(0);
   Vc[pos(i,j)].setdpw(0);
   Vc[pos(i,j)].setdpn(0);
   Vc[pos(i,j)].setdps(0);
}
else if ((i==Nx+2)&&(j==1))    //Corner 3
{
   Vc[pos(i,j)].setdx(0);
   Vc[pos(i,j)].setdy(0);
   Vc[pos(i,j)].setdpe(0);
   Vc[pos(i,j)].setdpw(0);
   Vc[pos(i,j)].setdpn(0);
   Vc[pos(i,j)].setdps(0);
}
else if ((i==0)||(j==0)||(i==Nx+3)||(j==Ny+3))  //Empty backup nodes
{
   Vc[pos(i,j)].setx(0);
   Vc[pos(i,j)].sety(0);
   Vc[pos(i,j)].setdx(0);
   Vc[pos(i,j)].setdy(0);
   Vc[pos(i,j)].setdpe(0);
   Vc[pos(i,j)].setdpw(0);
   Vc[pos(i,j)].setdpn(0);
   Vc[pos(i,j)].setdps(0);
}
else      //Internal node
{
                                        if(i==2)
    {
                x = Vc[pos(i,j)].getx();
                xw = Vc[pos(i-1,j)].getx();
                xe = Vc[pos(i+1,j)].getx();
                dpe = (xe-x)/2.0;
                dpw = (x-xw);
                dx = dpw+dpe;
```

```
}
else if(i==Nx+1)
{
		x = Vc[pos(i,j)].getx();
		xw = Vc[pos(i-1,j)].getx();
		xe = Vc[pos(i+1,j)].getx();
		dpe = (xe-x);
		dpw = (x-xw)/2.0;
		dx = dpw+dpe;
}
else
{
		x = Vc[pos(i,j)].getx();
		xw = Vc[pos(i-1,j)].getx();
		xe = Vc[pos(i+1,j)].getx();
		dpe = (xe-x)/2.0;
		dpw = (x-xw)/2.0;
		dx = dpw+dpe;
}

if(j==2)
{
		y = Vc[pos(i,j)].gety();
		ys = Vc[pos(i,j-1)].gety();
		yn = Vc[pos(i,j+1)].gety();
		dpn = (yn-y)/2.0;
		dps = (y-ys);
		dy = dpn+dps;
}
else if(j==Ny+1)
{
		y = Vc[pos(i,j)].gety();
		ys = Vc[pos(i,j-1)].gety();
		yn = Vc[pos(i,j+1)].gety();
		dpn = (yn-y);
		dps = (y-ys)/2.0;
		dy = dpn+dps;
}
```

```
            else
            {
                    y = Vc[pos(i,j)].gety();
                    ys = Vc[pos(i,j-1)].gety();
                    yn = Vc[pos(i,j+1)].gety();
                    dpn = (yn-y)/2.0;
                    dps = (y-ys)/2.0;
                    dy = dpn+dps;
            }

          Vc[pos(i,j)].setdx(dx);
          Vc[pos(i,j)].setdy(dy);
          Vc[pos(i,j)].setdpe(dpe);
          Vc[pos(i,j)].setdpw(dpw);
          Vc[pos(i,j)].setdpn(dpn);
          Vc[pos(i,j)].setdps(dps);
        }
      }
    }
    cout<<"Malla creada correctamente"<<endl;
}
```

Function for the *x* axis position for nodes

```
double casoSquareCyl::set_coord_X(int point_i)
{
    double x, dx;
    double i, N1, N2;

    N1 = round((Nx+2-Nxu)*x1/(x1+L-x2));  //Number of elements section 1
    N2 = round(Nx+2-Nxu-N1);  //Number of elements section 2
    dx = (x2-x1)/Nxu;

    i = point_i;
    if(i<N1)
    {
            x = (tanh(gamma*((i-1)/(N1)))/tanh(gamma))*x1;
    }
    else if((i>=N1)&&(i<=N1+Nxu))
```

```
    {
        x = x1 + dx*(i-N1);
    }
    else
    {
        x = ((1-(tanh(gamma*((N2-(i-N1-Nxu))/(N2+1))))/tanh(gamma))*(L-x2))+x2;
    }

    return x;
}
```

Function for the *y* axis position for nodes

```
double casoSquareCyl::set_coord_Y(int point_j)
{
    double y, dy;
    int j;

        dy = H/Ny;

    j = point_j;
    if(j==Ny+2)
    {
        y = dy*(j-2);
    }
    else if(j==1)
    {
        y = 0;
    }
    else
    {
            y = (dy/2)+dy*(j-2);
    }

    return y;
}
```

Function to compute the predictor velocity

```
void casoSquareCyl::calc_upre()      //Computes the horizontal predictor velocity
{
    unsigned int i, j;
    unsigned int imin, imax, jmin, jmax;
    double dx, dy;
    double Run;

    buscar_punto(cyl_C - (D/2.0), (H/2.0) - (D/2.0), imin, jmin);
    buscar_punto(cyl_C + (D/2.0), (H/2.0) + (D/2.0), imax, jmax);

    imin = imin - 2;
    imax = imax - 1;
    jmin = jmin - 1;
     jmax = jmax - 1;

    for (j=0; j<=Ny+1; j++)
    {
      for (i=0; i<=Nx; i++)
      {
        if((i==0)||(i==Nx)||(j==0)||(j==Ny+1)) //Condition for the boundary
        {
           upre[pos_SX(i,j)] = un[pos_SX(i,j)];
        }
        else if((i>=imin)&&(i<=imax)&&(j>=jmin)&&(j<=jmax)) //Immersed Boundary Method conditions
        {
             upre[pos_SX(i,j)] = un[pos_SX(i,j)];
        }
        else
        {
           dx = SX[pos_SX(i,j)].getdx();     //Gets node size
           dy = SX[pos_SX(i,j)].getdy();
           Run = calc_Run(i, j);            //Computes the diffusive and convective of the u velocity
           upre[pos_SX(i,j)] = un[pos_SX(i,j)] + dt/(dx*dy)*((3/2*Run) - (1/2*Ru0[pos_SX(i,j)])); //Compute the predictor vel.
           Ru0[pos_SX(i,j)] = Run;    //Saves the Rn compute don the Ru0 vector for the next time iteration
        }
      }
    }
}
```

Computes the Rn term

```
double casoSquareCyl::calc_Run(unsigned int i, unsigned int j) //Computes the convective and diffusive of u vel
{
   double mue, muw, mun, mus;
   double dx, dy, dpe, dpw, dpn, dps;
   double me, mw, mn, ms, m;
   double dense, densw, densn, denss;
   double up, ue, uw, uno, us;
   double vA, vB, vs, vno;
   double convece, convecw, convecn, convecs;
   double Run, convec, difus;

   //Computes node size and distances between them
   dx = SX[pos_SX(i,j)].getdx();
   dy = SX[pos_SX(i,j)].getdy();
   dpe = SX[pos_SX(i+1,j)].getx() - SX[pos_SX(i,j)].getx();
   dpw = SX[pos_SX(i,j)].getx() - SX[pos_SX(i-1,j)].getx();
   dpn = SX[pos_SX(i,j+1)].gety() - SX[pos_SX(i,j)].gety();
   dps = SX[pos_SX(i,j)].gety() - SX[pos_SX(i,j-1)].gety();

   //Gets velocity of the node and its neighbours
   up = un[pos_SX(i,j)];
   ue = un[pos_SX(i+1,j)];
   uw = un[pos_SX(i-1,j)];
   uno = un[pos_SX(i,j+1)];
   us = un[pos_SX(i,j-1)];

   //Gets the density value
   dense = densw = densn = denss = dens;

   //Gets the viscosity value
   mue = muw = mun = mus = mu;

   //Computes the mass flow on each face of the node
   me = dense*dy*0.5*(up + ue);
   mw = densw*dy*0.5*(up + uw);
   vA = vn[pos_SY(i,j)];
   vB = vn[pos_SY(i+1,j)];
   vno = 0.5*(vA + vB);
```

```
    mn = densn*dx*vno;
    vA = vn[pos_SY(i,j-1)];
    vB = vn[pos_SY(i+1,j-1)];
    vs = 0.5*(vA + vB);
    ms = denss*dx*vs;

    m = me - mw + mn - ms;

    //Computes the diffusive and convective for the predictor vel.
    difus = (mue*(ue-up)/dpe*dy) - (muw*(up-uw)/dpw*dy) + (mun*(uno-up)/dpn*dx) - (mus*(up-us)/dps*dx);
    convece =  0.5*me*(up+ue);
    convecw = 0.5*mw*(up+uw);
    convecn = 0.5*mn*(up+uno);
    convecs = 0.5*ms*(up+us);
    convec = convece - convecw + convecn - convecs;

    Run = difus - convec;

    return (Run);
}
```

Computes the divergence of two vectors

```
void casoSquareCyl::divergencia(const vector<double> &u, const vector<double> &v)
{
     unsigned int i, j;
    double dense, densw, densn, denss;
     double dx, dy;
     double upe, upw, vpn, vps;
      int point;

    for (j=1; j<=Ny+2; j++)
    {
       for (i=1; i<=Nx+2; i++)
       {
          if((i==1)||(i==Nx+2)||(j==1)||(j==Ny+2)) //Nodos boundary
          {
              div[pos(i,j)] = 0;
          }
          else
```

```
        {
            dense = densw = densn = denss = dens;

            dx = Vc[pos(i,j)].getdx();
            dy = Vc[pos(i,j)].getdy();

            upe = u[pos_SX(i-1,j-1)];
            upw = u[pos_SX(i-2,j-1)];
            vpn = v[pos_SY(i-1,j-1)];
            vps = v[pos_SY(i-1,j-2)];

            div[pos(i,j)] = (dense*upe*dy) - (densw*upw*dy) + (densn*vpn*dx) - (denss*vps*dx);
        }
      }
    }
}
```

Gauss-Seidel algorithm

```
int casoSquareCyl::resolver_GS_P(const double eps, const unsigned int maxiter) //Computes P of n+1
{
            double res, res0=1;
            double ap, ae, aw, as, an, bp;
            double Pp;
            double Pe, Pw, Pn, Ps;
            unsigned int iter=1, i=0, j=0;
            int e=0;

   for(iter = 1; iter<maxiter && res0>eps && e==0; ++iter) //Bucle until desired precision achieved or error
   {
      res0=0;

      for (j=1; j<=Ny+2; j++)
      {
        for (i=1; i<=Nx+2; i++)
        {
          ap = coefs.ap[pos(i,j)];
          ae = coefs.ae[pos(i,j)];
          aw = coefs.aw[pos(i,j)];
          an = coefs.an[pos(i,j)];
```

```
      as = coefs.as[pos(i,j)];
      bp = b[pos(i,j)];

      Pe = P[pos(i+1,j)];
      Pw = P[pos(i-1,j)];
      Pn = P[pos(i,j+1)];
      Ps = P[pos(i,j-1)];

        Pp = (bp + ae*Pe + aw*Pw + an*Pn + as*Ps)/ap;       //Computes the pressure
        P[pos(i,j)] = Pp;
        }
}
//Bottom
j=1;
for (i=2; i<Nx+2; i++)
{
   ap = coefs.ap[pos(i,j)];
   ae = coefs.ae[pos(i,j)];
   aw = coefs.aw[pos(i,j)];
   an = coefs.an[pos(i,j)];
   as = coefs.as[pos(i,j)];
   bp = b[pos(i,j)];

   Pe = P[pos(i+1,j)];
   Pw = P[pos(i-1,j)];
   Pn = P[pos(i,j+1)];
   Ps = P[pos(i,j-1)];

   Pp = (bp + ae*Pe + aw*Pw + an*Pn + as*Ps)/ap;       //Computes the pressure
   P[pos(i,j)] = Pp;
}



//Left
i=1;
for (j=2; j<Ny+2; j++)
{
   ap = coefs.ap[pos(i,j)];
   ae = coefs.ae[pos(i,j)];
```

```
    aw = coefs.aw[pos(i,j)];
    an = coefs.an[pos(i,j)];
    as = coefs.as[pos(i,j)];
    bp = b[pos(i,j)];

    Pe = P[pos(i+1,j)];
    Pw = P[pos(i-1,j)];
    Pn = P[pos(i,j+1)];
    Ps = P[pos(i,j-1)];

    Pp = (bp + ae*Pe + aw*Pw + an*Pn + as*Ps)/ap;
    P[pos(i,j)] = Pp;
  }

  int imax=-1, jmax=-1;
  for (j=1; j<=Ny+2; j++)
  {
    for (i=1; i<=Nx+2; i++)
    {
      ap = coefs.ap[pos(i,j)];
      ae = coefs.ae[pos(i,j)];
      aw = coefs.aw[pos(i,j)];
      an = coefs.an[pos(i,j)];
      as = coefs.as[pos(i,j)];
      bp = b[pos(i,j)];

      Pe = P[pos(i+1,j)];
      Pw = P[pos(i-1,j)];
      Pn = P[pos(i,j+1)];
      Ps = P[pos(i,j-1)];

      res = abs(ap*P[pos(i,j)] - ae*Pe - aw*Pw - an*Pn - as*Ps - bp);      //Computes the residual in absolute value
      if (res>res0)
      {
        res0=res;   //Saves the bigest residual, all must achieve convergence criteria
      }
    }
  }
```

```
      if (res0 > GINF)
     {
        e=2;   //El programa diverge de forma descontrolada
      }
}
  if(iter>=maxiter)
  {
    e=1;      //Hemos superado el número de iteraciones máximas
  }

  return e;
}
```

Computes the gradient on X direction

```
void casoSquareCyl::gradienteX(const vector<double> &Press)
{
    int i, j;
    double PA, PB;
    double dBA;

    for (j=1; j<=Ny; j++)
    {
      for (i=1; i<=Nx-1; i++)
      {
        PA = Press[pos(i+1,j+1)];                    //Pressure on west face
        PB = Press[pos(i+2,j+1)];                    //Pressure on east face
        dBA = Vc[pos(i+2,j+1)].getx() - Vc[pos(i+1,j+1)].getx();   //Distance between both points
        gradX[pos_SX(i,j)] = (PB-PA)/dBA;
      }
    }
}
```

Computes the velocity of n+1

```
void casoSquareCyl::calc_un1()
{
    unsigned int i, j;
    unsigned int imin, imax, jmin, jmax;
    double dpw;
```

```
    for (j=0; j<=Ny+1; j++)
    {
      for (i=0; i<=Nx; i++)
      {
        if(i==0)            //Conditiones Inlet
        {
           un1[pos_SX(i,j)] = upre[pos_SX(i,j)];
        }
        else if(j==0)     //No-slip wall
        {
             un1[pos_SX(i,j)] = upre[pos_SX(i,j)];
        }
        else if(j==Ny+1)                //No-slip wall
        {
             un1[pos_SX(i,j)] = upre[pos_SX(i,j)];
        }
        else if(i==Nx)    //Convective Boundary Conditions
        {
             dpw = SX[pos_SX(i,j)].getx() - SX[pos_SX(i-1,j)].getx();
             un1[pos_SX(i,j)] = un[pos_SX(i,j)] - ((dt*u_max/dpw)*(un[pos_SX(i,j)] - un[pos_SX(i-1,j)]));
        }
        else
        {
           un1[pos_SX(i,j)] = upre[pos_SX(i,j)] - (1.0/dens)*gradX[pos_SX(i,j)];   //Computes the new velocity with the correction
        }
      }
    }
}
```

Immersed Boundary Condition

```
void casoSquareCyl::IBC()
{
   unsigned int i, j;
   unsigned int imin, imax, jmin, jmax;

   buscar_punto(cyl_C - (D/2.0), (H/2.0) - (D/2.0), imin, jmin);
   buscar_punto(cyl_C + (D/2.0), (H/2.0) + (D/2.0), imax, jmax);
```

```
  for(j=jmin; j<=jmax; j++)
  {
       for(i=imin; i<=imax; i++)
      {
       un1[pos_SX(i-1,j-1)] = 0;   //east
       un1[pos_SX(i-2,j-1)] = 0;   //west
       vn1[pos_SY(i-1,j-1)] = 0;   //north
       vn1[pos_SY(i-1,j-2)] = 0;   //south
     }
  }
}
```