NTNU

Norwegian University of
Science and Technology

Development of a new Navier-Stokes Solver using a Generalized HPC method for the Pressure Poisson Equation

Øyvind Rabliås

Marine Technology
Submission date: December 2016
Supervisor: Marilena Greco, IMT

Norwegian University of Science and Technology
Department of Marine Technology

Scope

This report describes the work done during the semester with the master thesis in marine hydrodynamics at the Norwegian University of Science and Technology. The workload is estimated to 30 credits.

At the beginning of the semester, a draft of the project work was defined. During the semester was the plan reviewed, and the following steps were outlined as a framework for the project:

1. Summarize major findings/outcomes from the project thesis.
2. Complement the literature study of the Project on viscous-flow formulations and select the Navier-Stokes solution strategy to be used for problems with 2D laminar flows.
3. Apply a generalized HPC solver made available by the developers (at confidential level) for the BVP of a 2D circular cylinder in a uniform current as done for the HPC solver during the Project. Examine the numerical-convergence properties.
4. Introduce the generalized HPC solver for the solution of Poisson equations into the Navier-Stokes solution strategy identified in step 2 and apply the resulting method to the time evolution of a 2D problem with a body in a steady current. Compare the results against available reference solutions, for instance in terms of wake evolution and loads on the cylinder. To limit the error sources connected with enforcing the body boundary conditions, you can choose a squared cylinder for the study.
5. If possible, use an available Poisson-solver library in the basic Navier-Stokes solver and compare the results in terms of CPU-time costs and solution accuracy.

Preface

This report is the final thesis of my degree of Master of Science and Technology at the Department of Marine Technology at NTNU.

The basis of the thesis is the HPC and the generalized HPC method, which is higher order methods for solving Laplace and Poisson problems receptively. These methods have shown promising results both regarding accuracy and computation time.

The HPC method is implemented for a uniform flow around a circular cylinder. The generalized HPC method has been implemented to solve the pressure Poisson equation in a Navier-Stokes solver.

It has been interesting and motivational, both to learn more about the subject and to collaborate with other scientists. The process of implementing my routine in a code developed by someone else has been very informative.

The progression of the project has been by fits and starts. Some things have been more time consuming than expected. Especially, the improvement of the Immersed boundary grid that was implemented in the project thesis, and implementing the interface between Matlab and Fortran.

I wish to thank my supervisor Professor Marilena Greco for the guidance the last year. The weekly meetings, sometimes more often, have been very helpful to keep a good progress in the project.

I will also thank Giuseppina Colicchio. She has implemented my routine in her Navier-Stokes solver, and been very helpful in that process.

Andrea Bardazzi also deserves a thank. He has kindly shared his code where the generalized HPC method where implemented. This was very helpful, especially in the beginning of the project, and made the process of understanding the method much easier. His code has been a framework for my implementation.

I want to thank Finn Christian W. Hanssen and Shajoun Ma. It has been helpful to discuss the HPC method with them, especially the treatment of the boundary conditions.

Sammendrag

Lang beregningstid gjør at løsning av Navier-Stokes ligninger ikke er utbredt for fullskalaproblemer. Mange marine konstruksjoner er definert som storvolumkonstruksjoner. For disse konstruksjonene gir potensialteori akseptable resultater med betydelig kortere beregningstid enn Navier-Stokes løsere.

Shao og Faltinsen (2012) har utviklet HPC-metoden, en effektivt og nøyaktig metode for potensialstrømninger. Metoden bruker harmoniske polynomer som per definisjon tilfredsstiller Laplace ligningen. Bardazzi et al. (2015) har generalisert metoden, slik at den løser Poisson ligningen.

I denne oppgaven er et "Immersed Boundary Grid" implementert for en sirkulær sylinder. Potensialstrømningen rundt sylinderen er løst med HPC-metoden. En konvergensrate på 3.47 ble oppnådd for $L_2 - feilen$. Noen svingninger i konvergensgrafen ble observert, dette er typisk for "Immersed boundary" metoder.

Den generelle HPC-metoden er implementert i en Navier-Stokes løser, for å løse Poisson ligningen for trykket. En strømning rundt en firkantet sylinder er analysert for et Reynoldsnummer på 40. Det samme problemet ble også løst med en "finite difference" metode (FDM) for trykkligningen. Resultatene ble sammenlignet med referanseverdier.

En dragkoeffisient på 1.69 ble oppnådd med den generelle HPC-metoden, for FDM ble dragkoeffisienten 1.77. Referanseverdiene varierte mellom 1.6 og 1.8. Ingen av metodene konvergerte. Dette kan være fordi at det fineste gridet var for grovt. Siden et uniformt grid ble brukt, var det ikke mulig å bruke et finere grid.

Simuleringene var betydelig raskere når HPC-metoden ble brukt enn når FDM ble brukt for å løse trykkligningen. For det fineste gridet, som tilsvarer 237 600 ukjente, var HPC-metoden 3.4 ganger raskere enn FDM. På maskinen som ble brukt, utgjør denne forskjellen cirka 49 timer, for å simulere 30 sekunder. Dette illustrerer muligheten for å redusere beregningstiden i en Navier-Stokes løser, ved å bruke den generelle HPC-metoden for trykkligningen.

Den generelle HPC-metoden tilfredsstilte ikke kompatibilitet betingelsen automatisk. For å løse dette ble et modifisert ligningssystem løst istedenfor det originale ligningssystemet. En singulærverdi dekomposisjon var nødvendig for å løse det modifiserte ligningssystemet. Dette øker beregningstiden, og burde være mulig å unngå.

Abstract

The computational cost of solving the Navier-Stokes equations numerically is too high for most full-scale applications, especially within the marine field. Many marine structures are defined as large volume structures. For these structures give potential flow theory reliable results, to an acceptable computational cost.

Shao and Faltinsen (2012) have developed the HPC method, an efficient and accurate field solver for potential flow problems. The method utilises harmonic polynomials, which satisfy the Laplace equation by definition. Bardazzi et al. (2015) have generalized the method to solve the Poisson equation.

In this thesis is an immersed boundary grid, proposed by Hanssen et al. (2015), implemented for a circular cylinder. The potential flow around the cylinder is solved with the HPC method. A convergence rate of 3.47 was obtained for the $L_2 - error$. However, some oscillations were observed, which is typical for immersed boundary methods.

The generalized HPC method is implemented in a Navier-Stokes solver, to solve the pressure Poisson equation. A uniform flow around a square cylinder is investigated for a Reynolds number of 40. The results are compared with reference values. The same problem is also solved when a finite difference (FDM) scheme is used for the pressure Poisson equation.

A drag coefficient of 1.69 was obtained with the generalized HPC method, while the FDM scheme obtained a drag coefficient of 1.77. The drag coefficients in the literature are in the range between 1.6 and 1.8. None of the methods converged. This is probably because the finest grid was too coarse to obtain convergence. A uniform grid was applied, and further grid refinement was out of reach.

Concerning computational time was the generalized HPC method significant faster than the FDM scheme. For the finest grid, which corresponds to 237 600 unknowns, was the HPC solver 3.4 times faster than the FDM scheme. On the machine that was used, do this correspond to a difference of approximately 49 hours, when 30 seconds are simulated. This illustrates the potential for reducing the computational time in a Navier-Stokes solver, by using a generalized HPC method for the pressure Poisson equation.

The generalized HPC method did not automatically satisfy the discrete compatibility condition. To resolve this was a modified equation system solved. A singular value decomposition was necessary to obtain the modified equation. This represents an additional computational cost, which should be possible to avoid.

Nomenclature

The most used symbols and abbreviations are listed below. All symbols and abbreviations are described when they are introduced. Some symbols can be used to indicate different things. Vectors are represented by bold letters.

μ	-	Dynamic Viscosity
ν	-	Kinematic Viscosity = μ/ρ
ρ	-	Density of Fluid
τ	-	Shear Stress
ϕ	-	Velocity Potential
Ω	-	Computational Domain
Δx	-	Distance Between Grid points in x-direction
Δy	-	Distance Between Grid points in y-direction
C_D	-	Drag Coefficient
CFD	-	Computational Fluid Dynamics
D	-	Diameter
FDM	-	Finite Difference Method
FVM	-	Finite Volume Method
HPC	-	Harmonic Polynomial Cell
IBG	-	Immersed Boundary Grid
IBM	-	Immersed Boundary Method
N	-	Number of Unknowns in the Computational Domain
n	-	Normal Vector
PDE	-	Partial Differential Equation
Re	-	Reynolds Number
SVD	-	Singular Value Decomposition
U	-	Velocity Vector
U	-	Undisturbed Horiziontal Velocity
u	-	Horizontal Velocity Component
v	-	Vertical Velocity Component

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The Navier-Stokes equations give a precise description of a fluid flow problem. However, the computational cost of solving these equations numerically is too high for most full-scale applications, especially for marine applications. More efficient numerical solvers will be a major contribution towards solving full-scale problems.

Potential flow theory is widely applied for marine applications. Potential flow theory is valid for inviscid flows and the computation time is much less than for the viscous problem. For large volume structures, which is the case for many marine applications, potential flow theory gives reliable results. However, in some applications are viscous effects non-negligible, and potential flow models are no longer sufficient. For example is the viscous resistance of great matter in ship hydrodynamics. Viscous effects are also of great significance for many offshore structures, especially those with slender elements such as jackets and risers (Gorski, 2002).

Most practical problems have no analytical solution, and numerical methods are necessary to solve the hydrodynamic problem. The Laplace equation is the governing equation within potential flow theory. There exist several numerical methods to solve the corresponding boundary value problem. Boundary element methods (BEM) have been popular for marine applications. Field solvers like; finite differences and finite elements are other numerical methods that are applied to the same purpose.

Shao and Faltinsen (2012) have developed a new numerical field solver for potential flow problems. The Harmonic Polynomial Cell (HPC) method, is based on harmonic polynomials, which satisfies the Laplace equation by definition. The method has shown promising results both regarding accuracy and CPU time compared to traditional solvers (Shao and Faltinsen, 2012), (Shao and Faltinsen, 2014a), (Shao and Faltinsen, 2014b).

Recently, Ma et al. (2016) have carried out a detailed analysis of the properties of the HPC method. They have investigated the accuracy of different cells and local interpolation, global solution, and ways of enforcing boundary conditions.

Hanssen et al. (2015) have introduced an Immersed Boundary Grid (IBG) for the HPC method. The method has shown a convergence rate close to fourth order.

The immersed boundary grid is a method to apply boundary conditions at irregular geometries in a Cartesian grid. This simplifies the implementation, compared to body-fitted grids. The approach could also be more efficient than a body-fitted grid since it is not necessary to remesh the entire domain at every time step, for time-dependent problems.

The implementation of the Immersed Boundary Grid for a potential flow around a circular cylinder was the primary objective of the project thesis (Rabliås, 2016). A convergence rate of 3.21 was obtained, which is similar to the results in Hanssen et al. (2015). This is within the theoretical range, considering that all polynomials, except one, up to fourth order were included in the interpolation. However, the IBG converged with an oscillating behaviour, which is an issue for immersed boundary methods in general. The same phenomenon was also discovered in Ma et al. (2016).

A thorough review of the results obtained in the project thesis revealed that the convergence rate was very sensitive of which grids that were included. This indicates that it is possible to improve the algorithm.

Bardazzi et al. (2015) have extended the HPC method to solve the Poisson equation. The generalized HPC method uses the same harmonic polynomials as the original HPC method, while a bi-quadratic polynomial approximates the particular solution. Bardazzi et al. (2015) have tested the method for physical Poisson problems where analytical solutions exist. The results show a convergence rate of approximately fourth order, which is close to the theoretical accuracy. Also, the results indicate that the method is efficient concerning computational time compared to traditional solvers.

When a fractional step approach solves the Navier-Stokes equations, a Poisson equation is obtained for a quantity connected with the pressure. Armfield and Street (2002) compared a fractional step method with an iterative method with no Poisson equation. Common to the methods that were tested was that the pressure correction represented $50-90\%$ of the CPU time. Hence, an accurate and efficient method for the pressure calculation is of great importance.

1.2 Objectives

The implementation of the immersed boundary grid in the project thesis (Rabliås, 2016) obtained a convergence rate of 3.21 for a potential flow around a circular cylinder. However, further investigations revealed that the convergence rate was very sensitive of which grids that were included.

The boundary value problem that was solved in the project thesis was a Laplace problem. This is the same as a Poisson problem where the right-hand side is set to zero. The first objective of this thesis is to solve the same boundary value problem as in the project thesis, with the generalized HPC method. A convergence rate between third and fourth order should be obtained. Since the HPC and the generalized HPC method are based on the same principle, is the same IBG approach used to implement the boundary conditions.

The primary objective of this master thesis is to apply the generalized HPC method for the pressure calculation in a Navier-Stokes solver. An efficient solver for the Poisson equation will be a considerable step towards an efficient Navier-Stokes solver. This thesis will describe the implementation of the generalized HPC method in an existing Navier-Stokes solver. The performance of the generalized HPC method shall be compared with another Poisson solver that already is implemented in the code.

The following tasks will be performed to achieve the objectives:

1. Implement an Immersed Boundary Grid for the Generalized HPC method. Test the IBG for a boundary value problem with a circular cylinder.
2. Implement the generalized HPC method in a computational domain with a square cylinder.
3. Implement point two in a Navier-Stokes solver for the pressure correction.
4. Test the accuracy of the results obtained by the Navier-Stokes solver with the generalized HPC method.
5. Compare the performance of the Navier-Stokes solver with the generalized HPC method with another Poisson solver that already is implemented in the code.

1.3 Scope and Limitations

The Immersed Boundary Grid that is implemented for the circular cylinder is not made for a general body. Hence, it can only be used for a circular cylinder. However, the approach is general and can be used for a general body. The IBG will only be tested for a Laplace problem, i.e the right-hand side is set to zero. This is because there exists an analytical solution to a Laplace problem that is relevant for marine applications.

The main objective is to test the generalized HPC method in a Navier-Stokes solver. Instead of programming a Navier-Stokes solver from scratch, is the Poisson solver implemented in an existing code. This makes it easier to compare the results obtained with the generalized HPC method with another solver.

The square cylinder is chosen such that the errors introduced from the boundary conditions are kept at a minimum. At this point, the objective is to study the method itself, not the boundary conditions. A thorough study of the boundary conditions for bodies with irregular boundaries, is a major task itself.

For Reynolds number above a certain limit, the flow around a square cylinder gets unstable, with oscillating wake behaviour. To keep the flow as simple as possible is the flow studied in the stable region. This reduces the error that could come from sampling the time-varying results. Moreover, the computation time increases for increasing Reynolds numbers.

1.4 Structure of Report

The rest of the report is organised as follows. Chapter 2 reviews the governing equations that are used in the report. The Navier-Stokes equations are presented together with the governing equations in potential flow theory. In addition are the main features of a uniform flow around a square and circular cylinder presented. Relevant parameters for such flows are defined.

Chapter 3 reviews some of the challenges that arise when partial differential equations (PDEs) are solved numerical, particular the Navier-Stokes equations. How to measure the accuracy and error norms are defined. Different approaches for spatial discretization of the computational domain are presented. Time discretization is also discussed. Some strategies for solving the Navier-Stokes equations are discussed, especially the fractional step method is given particular attention.

In chapter 4 are the HPC and the generalised HPC method presented. The concept and numerical formulation are reviewed.

The problems that are investigated are described in chapter 5. Physical and computational parameters are given here. Changes to the original code and new routines are also outlined in this chapter. Challenges that were encountered during the implementation are reviewed, and assumptions and choices are justified.

In chapter 6 are the simulation results presented. Drag coefficients, computation time, and convergence are investigated for the solution obtained by the generalized HPC method. The results are also compared with solutions obtained with another Poisson solver.

The results and challenges related to the implementation will be discussed in chapter 7. The drag coefficients that are obtained will be compared with reference values. Chapter 8 tries to make some conclusions and recommends further work on the topic.

Chapter 2

Theory

2.1 Fundamental Equations

The governing equations depend on the application and simplifications that are made. The continuity equation and the momentum equations give a detailed description of a flow problem. These equations are often called the Navier-Stokes equations, and for most practical applications these equations are almost exact. However, high computational costs of solving the Navier-Stokes equations has made potential flow theory very attractive for marine applications. In potential flow theory, is the Laplace equation the governing equation. The following sections will present the Navier-Stokes equations and the governing equations in potential flow theory.

2.1.1 Mass Conservation

The continuity equation describes the conservation of mass in a control volume. The principle is that the amount of fluid mass inside a control volume can only change by mass transport through the boundaries, i.e. mass cannot suddenly disappear. This gives us the relation between the rate of change of fluid mass in a control volume and the mass transport trough the boundaries.

$$\frac{d}{dt}\iint_{\Omega}\rho d\Omega + \oint_{\delta\Omega}(\rho\mathbf{U})\cdot\mathbf{n}dS = 0 \tag{2.1}$$

The first term represents the rate of change of mass in the control volume. While the second term, which is called the convective term, represents the mass flux through the boundaries of the control volume.

By use of Gauss' divergence theorem to the convective term, the surface integral can be transformed to a volume integral. Equation (2.1) can then be expressed on differential form (Ferziger and Perić, 1996):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{2.2}$$

For most marine applications is the flow considered to be incompressible, the continuity equation can then be further simplified:

$$\nabla \cdot \mathbf{U} = 0 \tag{2.3}$$

2.1.2 Momentum Conservation

The momentum conservation equations can be derived in several ways. The control volume approach that is used for the mass conservation can also be used for the momentum equations (Ferziger and Perić, 1996). Momentum has to be conserved in all directions (x,y and z). There is, therefore, one momentum equation in each direction. We start with the following conservation equation in vector form:

$$\frac{\partial}{\partial t} \int_{\Omega} \rho \mathbf{U} d\Omega + \int_{S} \rho \, \mathbf{U}\mathbf{U} \cdot \mathbf{n} \, dS = \sum \mathbf{f} \tag{2.4}$$

The first term on the left-hand side represents the rate of momentum change inside the control volume, the second term represents the momentum flow over the surfaces of the control volume. The right-hand side represents the sum of forces acting on the control volume. This term can be divided into two main groups; surface forces and body forces. Surface forces can be; pressure, normal and shear stress, surface tension, etc. Body forces can be gravity, centrifugal force etc. The body forces will not be further investigated and will be denoted f_b.

The surface forces are due to external stresses on the sides of the element. By assuming that the fluid is Newtonian, which is valid for most fluids in marine applications, the surface forces can be expressed by the stress tensor **T**:

$$\mathbf{T} = -\Big(p + \frac{2}{3}\nabla \cdot \mathbf{U}\Big)\mathbf{I} + 2\mu D \tag{2.5}$$

where **I** is the unit tensor and D is the rate of strain tensor:

$$D = \frac{1}{2}[\nabla \mathbf{U} + (\nabla \mathbf{U})^T] \tag{2.6}$$

The momentum conservation can now be rewritten as:

$$\frac{\partial}{\partial t}\int_\Omega \rho \mathbf{U} d\Omega + \int_S \rho \mathbf{U}\mathbf{U} \cdot \mathbf{n}\, dS = \int_S \mathbf{T} \cdot \mathbf{n} dS + \int_\Omega \rho \mathbf{f}_b d\Omega \tag{2.7}$$

By assuming an incompressible flow and applying Gauss' divergence theorem, the conservation of momentum can be written as:

$$\frac{\partial \mathbf{U}}{\partial t} + (\mathbf{U} \cdot \nabla)\mathbf{U} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{U} + \mathbf{f}_b \tag{2.8}$$

The conservation of momentum in this form, in addition to the continuity equation (2.3) are the equations that, in this thesis, will be referred to as the Navier-Stokes equations.

2.1.3 Potential Flow Theory

For an irrational velocity field ($\nabla \times \mathbf{U} = 0$) there exists a scalar function such that:

$$\mathbf{U} = \nabla\phi \tag{2.9}$$

This scalar is in marine hydrodynamics known as the velocity potential, which is central in potential flow theory. Potential flow theory is valid for flows that are:

- Inviscid
- Irrotational
- Incompressible

For incompressible flows, is the continuity equation expressed by equation (2.3). By inserting equation (2.9) in the continuity equation, we get:

$$\nabla \cdot \mathbf{U} = \nabla \cdot (\nabla\phi) = \nabla^2\phi = 0 \tag{2.10}$$

This is a Laplace equation, and it is the governing equation within potential flow theory.

Since the viscous effects are not present in potential flow theory, there are not any no-slip condition, and the boundary condition on a solid surface is given as:

$$\frac{\partial\phi}{\partial n} = \mathbf{n} \cdot \mathbf{U} \tag{2.11}$$

For simple geometries and flow problems there exists analytical solutions. However, for most practical problems numerical methods must be applied.

Equation (2.10) together with boundary conditions, e.g. equation (2.11), makes a boundary value problem. The boundary value problem can be solved by field solvers such as finite differences and finite volume methods, or by panel methods.

2.2 Flow Around a Circular Cylinder

2.2.1 Potential Flow Theory

In potential flow theory, can a uniform flow around a circular cylinder be described by a dipol, and the analytical expression in polar coordinates becomes (Pettersen, 2007):

$$\phi_{an} = Ur\left(1 + \frac{a^2}{r^2}\right)\cos\theta \tag{2.12}$$

Where U is the velocity of the undisturbed flow, a is the cylinder radius, r is the distance from the cylinder centre, and θ is the angle (see figure 2.1).

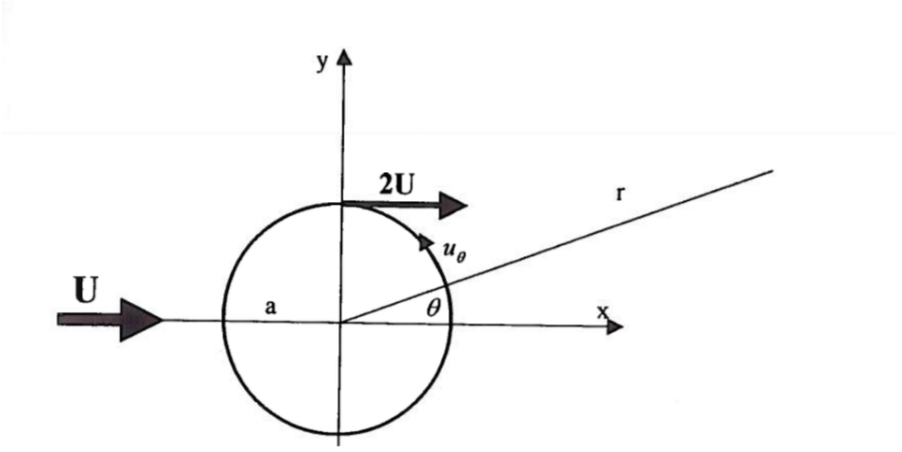

Figure 2.1: Parallel flow around a circular cylinder with potential flow theory (Pettersen, 2007).

In potential flow theory, there is an impermeability condition that ensures no flow through the cylinder surface. This implies that the normal velocity is zero at the cylinder surface. Hence, the tangential flow at the cylinder surface is given as:

$$u_\theta = -2U\sin\theta \tag{2.13}$$

By using a point far away together with a point on the cylinder surface, the Bernoulli equation gives an expression for the pressure on the cylinder surface:

$$p = p_0 + \frac{1}{2}\rho U^2 - 2\rho U^2 \sin^2\theta \tag{2.14}$$

Where p_0 is the pressure far away. Integration of the pressure gives the force on the cylinder. In potential flow theory, the net force on the cylinder is zero. This is known as the d'Alembert's paradox (Pettersen, 2007).

The streamlines around a circular cylinder are sketched in figure 2.2.

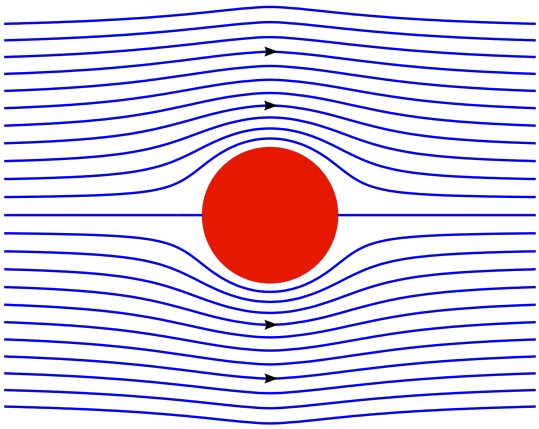

Figure 2.2: Streamlines around a circular cylinder in potential flow theory (math3510edensmith, 2014).

2.2.2 Viscous Flow

Potential flow theory gives, in general, not a realistic description of a flow around a circular cylinder. In addition to the impermeability condition, viscous forces introduce a no-slip condition at the cylinder surface. Both the tangential and the normal velocity is zero at the cylinder surface. The no-slip condition introduces a boundary layer near the solid surface, where the velocity increase from the velocity of the body to the velocity some distance away from the body. The boundary layer thickness is determined by the external flow and the body geometry.

For a flow around a curved object, like a circular cylinder, the pressure gradient will change. The situation when the velocity increases and the pressure decreases is known as a *favourable pressure gradient* (Cengel and Cimbala, 2010). For a circular cylinder, this is the situation between the stagnation point and the position for the maximum velocity. When the velocity decreases and the pressure increases, we have what is known as an *unfavourable pressure gradient*. The pressure gradient is unfavourable since the boundary layer is usually thicker, and it is much more likely to separate from the body than when the pressure gradient is favourable (Cengel and Cimbala, 2010). Figure 2.3 illustrates a viscous flow around a circular cylinder.

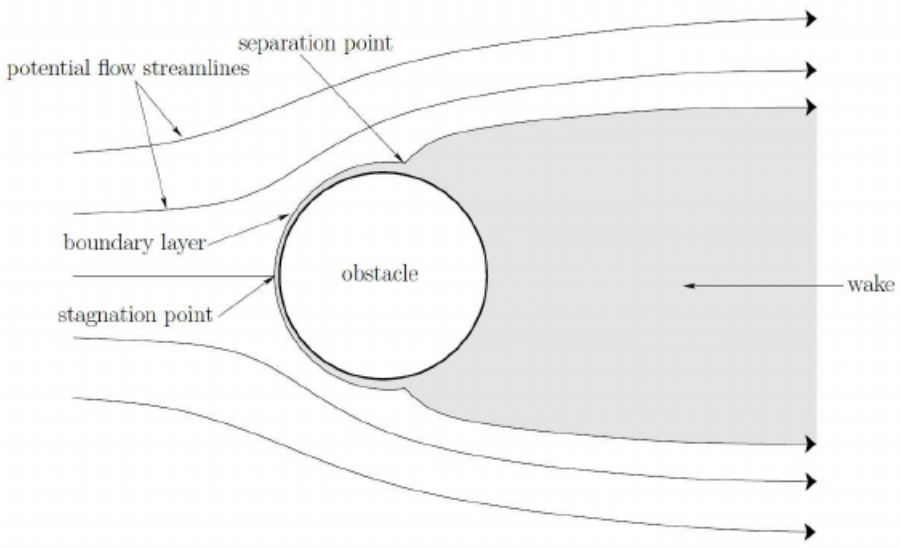

Figure 2.3: Viscous flow around a circular cylinder(Kharlamova et al., 2013).

The separation point on a circular cylinder is not fixed by the geometry, but will change when the Reynolds number changes. Where the separation point is located will influence the drag force that is acting on the cylinder. A narrow wake will give less drag force on the cylinder than a wide wake (Pettersen, 2007).

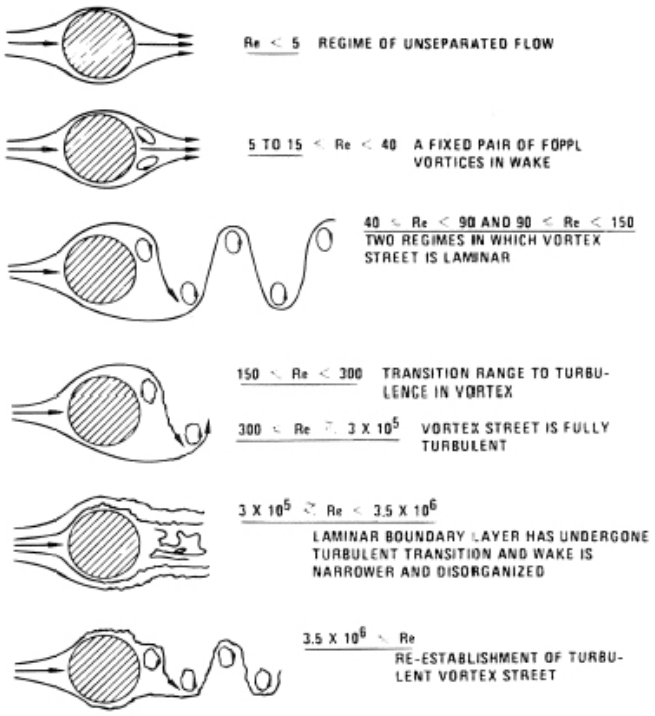

Figure 2.4: Flow around a circular cylinder for different Reynolds numbers (Sunden, 2011)

Figure 2.4 shows how the flow pattern around the cylinder changes for different Reynolds numbers. For very low Reynolds numbers ($Re < 5$) the flow does not separate from the cylinder, and the flow pattern is very similar to potential flow (figure 2.2). When the Reynolds number increases, the streamlines behind the cylinder moves away from the centre line and symmetric vortices are established. For $5 < Re <\approx 40$ the wake is stable and symmetric. For Reynolds numbers above 40, vortices are shed from the cylinder in an oscillating behaviour. Even for a constant external flow, the wake will vary with time.

2.3 Important Parameters

2.3.1 Drag Coefficient

The drag coefficient is a non-dimensional coefficient that represents the drag force F_D on a body. A common way to define the drag coefficient on a cylinder is:

$$C_D = \frac{F_D}{\frac{1}{2}\rho L U^2} \tag{2.15}$$

Where F_D is the drag force, L is the characteristic length, which for a cylinder is the diameter, and U is the external flow velocity.

It is common to divide the drag coefficient into two main contributions; friction drag $C_{D,f}$ and pressure drag $C_{D,p}$:

$$C_D = C_{D,f} + C_{D,p} \tag{2.16}$$

The friction drag is the component of the shear force tangential to the surface, and the pressure drag is the component of the pressure forces that is normal to the surface. The drag force on a surface is given by (Cengel and Cimbala, 2010):

$$F_D = \int_A (-P\cos\theta + \tau_w \sin\theta) dA \tag{2.17}$$

Where τ_w is the shear stress on the surface, and θ is the angle between the surface normal and the positive flow direction.

The shear stress in two dimensions is defined as:

$$\tau = \mu\Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big) \tag{2.18}$$

Where u is the horizontal velocity component, v is the vertical velocity component, and μ is the dynamic viscosity. However, it can be shown with an order-of-magnitude-analysis that the latter term in (2.18) is two orders smaller than the first term in a boundary layer. Hence, the shear stress at the surface can be approximated by (White, 1974):

$$\tau_w \approx \mu\Big(\frac{\partial u}{\partial y}\Big)_{y=0} \tag{2.19}$$

2.3.2 Recirculation Length

For a symmetric flow, with respect to the symmetry axis y=0, the recirculation length is defined as the distance between the cylinder wall rearmost point to the reattachment point (Boujo and Gallaire, 2014), as shown in figure 2.5.

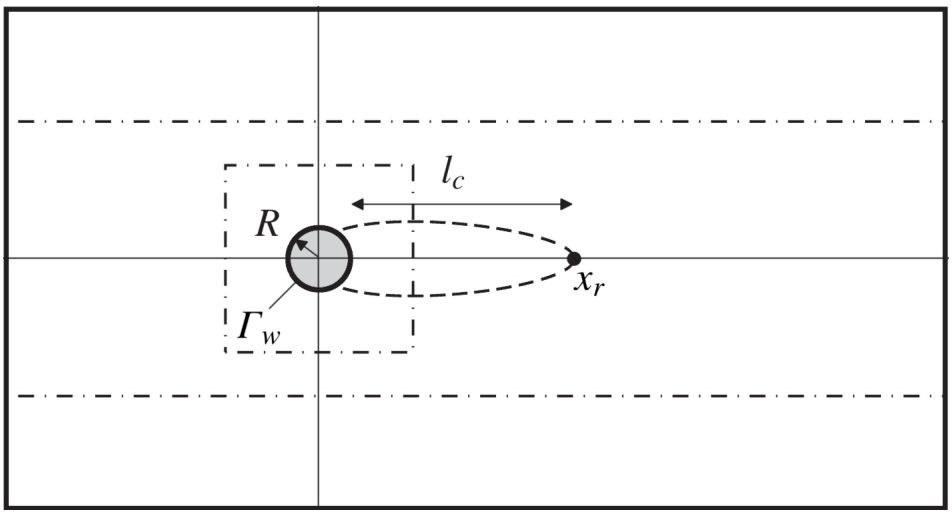

Figure 2.5: Recirculation length (Boujo and Gallaire, 2014)

The reattachment point is recognised at the point where the vertical velocity is zero. The recirculation length is an interesting parameter since both the shear and the maximum backwards flow increases together with the recirculation length (Boujo and Gallaire, 2014).

2.4 Flow Around a Square Cylinder

The flow around a square cylinder is similar to the flow around a circular cylinder considering the instability of the vortex shedding. However, other mechanisms as separation and shedding frequency, hydrodynamic forces and wake, differs significantly from the flow around a circular cylinder (Sharma and Eswaran, 2004).

The separation points of the square cylinder are fixed at the corners, this differs from the circular cylinder where the separation points not are fixed. It is stated that the hydrodynamic characteristics are relatively insensitive to Reynolds number, for bodies with fixed separation points (Okajima, 1982). However, the flow around a square cylinder has a similar development as the circular cylinder when the Reynolds number increases. Snapshots of the streamlines around a square cylinder for different Reynolds numbers, presented by Sharma and Eswaran (2004) can be found in figure 2.6.

The snapshots in figure 2.6 are computational results from Sharma and Eswaran (2004), the snapshots are instantaneous for the periodic flow. For Re=1 there is no separation, however, when Re=2 separation occurs on the trailing edge. The flow is stable for Reynolds numbers below 50, for higher Reynolds numbers the vortices separates alternately. Eventually, for higher Reynolds numbers, the separation point moves to the leading edge.

Another thing to notice is that the wake behind the square cylinder is broader and longer than for the circular cylinder (Sharma and Eswaran, 2004).

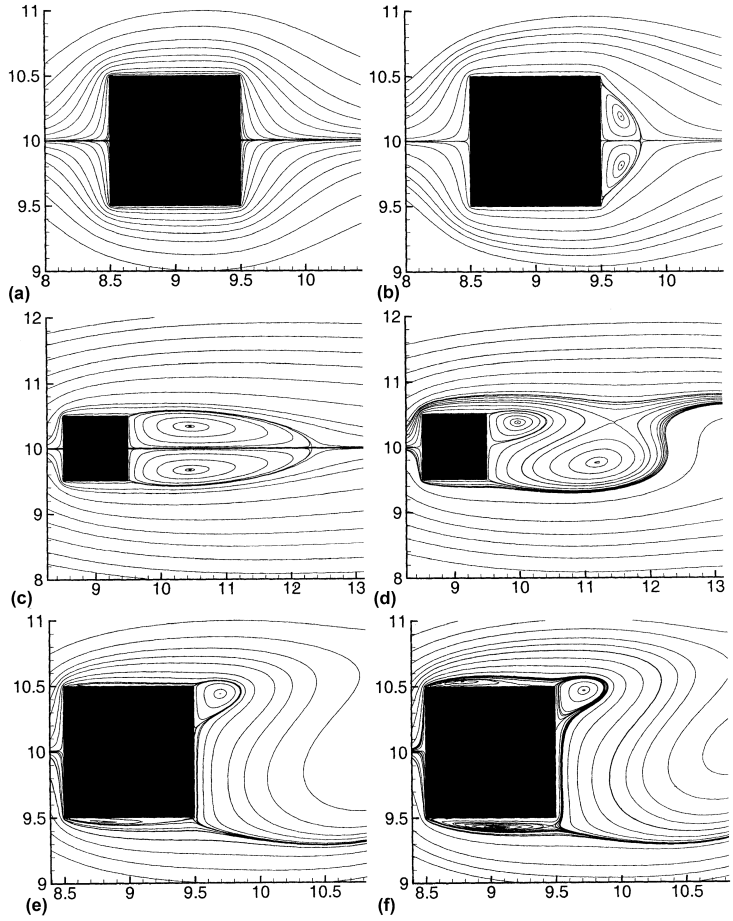

Figure 2.6: Instantaneous streamlines around a square cylinder for different Reynolds numbers: (a) Re=1; (b) Re=2; (c) Re=40; (d) Re=50; (e) Re=120; (f) Re=160 (Sharma and Eswaran, 2004).

Chapter 3

Numerical aspects

3.1 Accuracy and Convergence

3.1.1 Accuracy

Numerical calculations will always give approximations of the exact solution. It is important to have control over the different sources of error. Ferziger and Perić (1996) divides the error sources into three main groups.

- *Modelling errors*, which are defined as the difference between the actual flow and the exact solution of the mathematical model.
- *Discretization errors*, defined as the difference between the exact solution of the conservation equations and the exact solution of the algebraic system of equations obtained by discretizing these equations.
- *Convergence errors*, defined as the difference between the iterative and exact solutions of the algebraic equations systems.

In addition to these errors, the implementation of numerical algorithms could introduce errors to the solution.

When the Navier-Stokes equations are solved for a laminar flow is the modelling error negligible. For other flow problems could the modelling error be significant. Such flows can be turbulent flows where turbulent models are applied, or flow problems that are solved by potential flow theory.

Discretization errors are maybe the source of error that is most difficult to control. These errors come from the discretization of the conservation equations. By refining the grid, the discretization error will be reduced. However, this will lead to more unknowns and increase the CPU time. The final approximation will often be a tradeoff between accuracy and CPU time. The accuracy that is required depends on the given flow problem.

3.1.2 Error Norms

In order to evaluate and compare the results with analytic solutions or other studies, it is an advantage to quantify the accuracy. If there exists an exact solution to the flow problem, the p-norm is commonly used to estimate the error. For one and two dimensions respectively, the p-norm is given as:

$$\begin{aligned} \|\mathbf{E}^N\|_p &= \Big(\sum_i |\hat{u}_i^N - u_i^N|^p i \Big)^{1/p} \\ \|\mathbf{E}^N\|_p &= \Big(\sum_j \sum_i |\hat{u}_{ij}^N - u_{ij}^N|^p \Big)^{1/p} \end{aligned} \tag{3.1}$$

Where u is the exact solution and $\hat{u}$ is the numerical solution, u could be any quantity that is computed.

It could be convenient to normalise the error. Equation (3.1) is then divided by the p-norm of the exact solution. This normalised norm is in the rest of the report referred to as the $L_p - error$.

For some problems could it be more convenient to use parameters as drag coefficients, recirculation length, or shedding frequency to investigate the accuracy.

If there exists no exact solution to the problem, a reference solution has to be used to evaluate the results. Such references could be experimental or numerical results from the literature.

3.1.3 Convergence and Order of Accuracy

The order of accuracy indicates how the error decreases when the discretization is refined. It could be refined mesh or smaller time steps in the time integration. If the spatial discretization is first order accurate, the error is in the order of Δx ($\mathcal{O}(\Delta x)$). In other words, the error can be written as $C\Delta x$, where C is a constant. This implies that the error from the spatial discretization is halved when Δx is halved. If the discretization is second order accurate, the error can be written as $C\Delta x^2$. This means that the error from the spatial discretization is divided by four when Δx is halved.

It is important to emphasise that the order of accuracy does not say anything about the absolute value of the error. Two methods that both are second order accurate can give different results, the order of accuracy indicates only how the error decays.

3.2 Spatial Discretization

Numerical solution of the Navier-Stokes equations requires that the spatial derivatives must be discretized. The discretization can be performed by Eulerian field solvers like; Finite Differences Method (FDM), Finite Volume Methods (FVM) and Finite Element Method (FEM), or by Lagrangian methods like Moving Particle Semi-implicit Method (MPS) and Smoothed-particle Hydrodynamics (SPH). The Eulerian methods calculate the flow properties at specific points in space, while Lagrangian methods follows fluid particles in time and space.

The Finite Volume Method and the Finite Difference Method are two popular methods in computational fluid dynamics. The Finite Volume method solves the conservation equations on integral form for a control volume, while the finite difference method approximates the derivatives pointwise. The Finite Difference Method will be reviewed in section 3.2.1.

3.2.1 Finite Difference Method

The basis for the Finite Difference Method is the differential form of the conservation equation, e.g equation (2.8). Unlike the FVM that applies the conservation laws at control volumes, the FDM applies the differential form of the conservation law directly at nodes. For a given conservation law, each node has one unknown and must provide one algebraic equation. The neighbouring nodes are used to approximate the PDE. The FDM is based on the definition of the derivative, e.g. the first derivative:

$$\left(\frac{\partial \phi}{\partial x}\right)_{x_i} = \lim_{\Delta x \to 0} \frac{\phi(x_i + \Delta x) - \phi(x_i)}{\Delta x} \tag{3.2}$$

Equation (3.2) is a first-order forward approximation, meaning that node number *i* and node number *i+1* are used. There are other ways to approximate the first-order derivative, in figure 3.1 can you see the backward difference and the central difference approximation, in addition to the forward difference. As you can see, some approximations are better than other. The slope for the central approximation is close to the slope of the exact function. Actual, if the exact function was a second order function, and the nodes were equally spaced the slopes would match exactly (Ferziger and Perić, 1996).

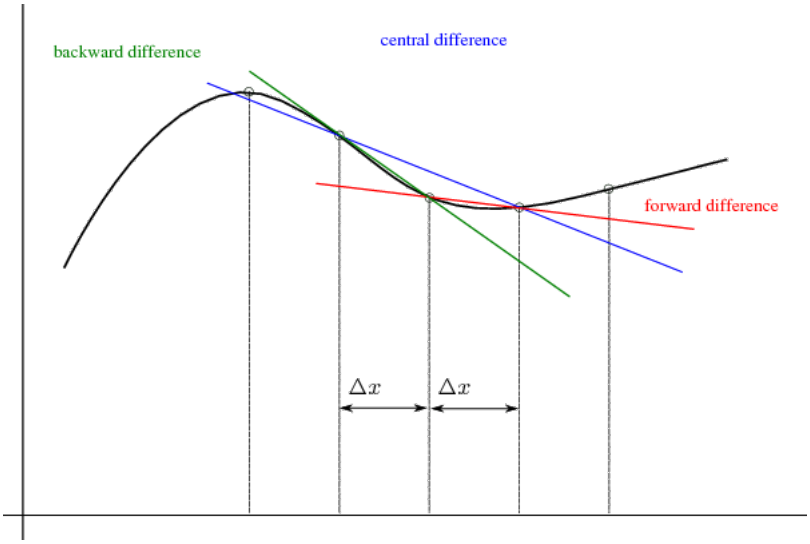

Figure 3.1: Graphical interpretations of the first order FDM approximation. (Heinzl, 2007)

There are other ways of deriving the approximations. Higher order approximations can be derived with Taylor series expansion. A continuous and differentiable function can be expressed as a Taylor series:

$$f(x) = f(x_i) + (x - x_i)\left(\frac{\partial f}{\partial x}\right)_i + \frac{(x - x_i)^2}{2!}\left(\frac{\partial^2 f}{\partial x^2}\right)_i + \frac{(x - x_i)^3}{3!}\left(\frac{\partial^3 f}{\partial x^3}\right)_i + \cdots + \frac{(x - x_i)^n}{n!}\left(\frac{\partial^n f}{\partial x^n}\right)_i + H \tag{3.3}$$

where H represents higher order terms.
If x are replaced by x_{i+1} or x_{i-1}, can we obtain expressions for the first derivative, in terms of node "i" and/or the neighbouring nodes.

$$\left(\frac{\partial f}{\partial x}\right)_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{2}\left(\frac{\partial^2 f}{\partial x^2}\right)_i - \frac{(x_{i+1} - x_i)^2}{6}\left(\frac{\partial^3 f}{\partial x^3}\right)_i + H \tag{3.4}$$

This expression corresponds to the forward differences, and if all elements with order two or higher are removed, the expression corresponds to the first order approximation. If we use x_{i-1} in equation (3.3), we obtain the backwards difference. And if we use both x_{i-1} and x_{i+1} we get the central difference.

Higher order approximations can be derived by including more terms on the right hand side. A second order approximation can be:

$$\Big(\frac{\partial f}{\partial x}\Big)_i = \frac{f_{i+1}(\Delta x_i)^2 - f_{i-1}(\Delta x_{i+1})^2 + f_i[(\Delta x_{i+1})^2 - (\Delta x_i)^2]}{\Delta x_{i+1}\Delta x_i(\Delta x_i + \Delta x_{i+1})} - \frac{\Delta x_{i+1}\Delta x_i}{6}\Big(\frac{\partial^3 f}{\partial x^3}\Big)_i + H \tag{3.5}$$

If the grid is equidistant, the scheme simplifies to the first order central scheme. The second derivative can be obtained by using the approximation for the first derivative twice. These approximations require at least data from three points (Ferziger and Perić, 1996).

The schemes above are only described in one dimension, the schemes in two and three dimensions are very similar. The details will not be discussed here, but more details can be found in Ferziger and Perić (1996) and Tannehill et al. (1997).

3.2.2 A Finite Difference Scheme for the Poisson Equation

The Poisson equation, which is central in a fractional step method, can be solved with a finite difference method. A second order scheme that includes five nodes can be discretized as:

$$\nabla\Big(\nabla\phi\Big)_{i,j} = \frac{(\phi_{i+1,j} - \phi_{i,j}) - (\phi_{i,j} - \phi_{i-1,j})}{\Delta x^2} + \frac{(\phi_{i,j+1} - \phi_{i,j}) - (\phi_{i,j} - \phi_{i,j-1})}{\Delta y^2} \tag{3.6}$$

The assembling of equation (3.6) results in a sparse coefficient matrix with five non-zero elements in each row. The global equation system can then be solved by a suitable solver.

3.2.3 Staggered Grid

For an Eulerian grid, the flow properties are calculated at nodes that are fixed in space. At first, it is natural to think that it is convenient to calculate all flow variables in the same nodes. Such grids are known as collocated grids. Collocated grids have some drawbacks that need to be solved; "*a highly non-uniform pressure field can act like a uniform field in the discretized momentum equations*" (Versteeg and Malalasekera, 1995).

Staggered grid is a remedy presented by Harlow and Welch (1965), that ensures that the pressure is properly represented. In a staggered grid is the pressure, and other scalar quantities, calculated in the centre of the cells, while the horizontal velocity component is calculated on the left side of the cell, and the vertical velocity component is calculated at the bottom of the cell.

A sketch of a staggered grid can be found in figure 3.2, where the dots represents where the pressure is calculated, and the arrows represent where the velocities are computed.

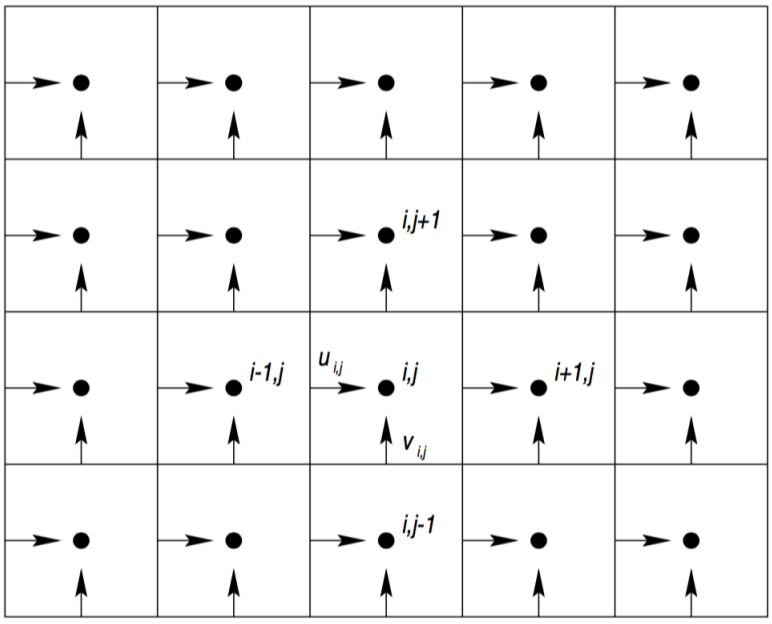

Figure 3.2: Staggered grid arrangement, (Colocchio, 2004)

3.3 Time Integration

Time-dependent problems need to be discretized in time. To review some different approaches and their features, the following initial value problem will be investigated:

$$\frac{df(t)}{dt} = R(t, f(t)); \quad f(t_0) = t^0 \tag{3.7}$$

The simplest discretization will be:

$$f^{n+1} = f^n + R(t_n, f^n)\Delta t \tag{3.8}$$

Where *n* represent the current time step and *n+1* represent the next time step. This scheme is known as the Explicit Euler method (Ferziger and Perić, 1996). The implicit Euler method is obtained by:

$$f^{n+1} = f^n + R(t_{n+1}, f^{n+1})\Delta t \tag{3.9}$$

The explicit scheme is straightforward to implement, while you need to solve a linear equation system when the implicit method is used. This means that the implicit method is more time demanding at each time step, and more memory is required. On the other hand, the implicit method is more stable with respect to time steps.

Tannehill et al. (1997) describes numerical stability in the following way: *"A stable numerical scheme is one for which errors from any source (round-off, truncation, mistakes) are not permitted to grow in the sequence of numerical procedures as the calculation proceeds from one marching step to the next.*"

Explicit schemes have very strict stability criterions. If the time step violates the stability criterion, the simulation will fail. Implicit methods have less strict stability criterions, and some implicit methods are even unconditional stable. This is an advantage when you are interested in the slow and long term variation, and the short time scale is less important.

For simple problems, a stability criterion can be derived analytically. However, for more complex problems it requires more work to establish an analytical stability criterion. Details of deriving stability criterion will not be reviewed here, but more information can be found in Tannehill et al. (1997).

Both the Explicit and the Implicit Euler method are first-order accurate in time. A method that combines the two methods is the Crank-Nicolson method:

$$f^{n+1} = f^n + \frac{1}{2}\Big[R(t_n, f^n) + R(t_{n+1}, f^{n+1})\Big]\Delta t \tag{3.10}$$

This is a method that is popular for solving the heat equation, and it is unconditionally stable and second order accurate in time (Tannehill et al., 1997).

The schemes that are described above are presented for a simple initial value problem, but the methodology is similar for more complex equations. The methods above are quite simple and straightforward and are known as two-level method since they involve unknowns at two time levels. The principles of two-level methods are general and can be used for more sophisticated methods (Ferziger and Perić, 1996).

There exist several different approaches for time integration; explicit, implicit, first order, higher order, predictor-corrector, multi-point, etc. A predictor-corrector scheme will be presented in the next paragraphs, for more information about other methods see Ferziger and Perić (1996), Tannehill et al. (1997) or Versteeg and Malalasekera (1995).

The predictor-corrector approach tries to combine the best of explicit and implicit two-level methods. First, a temporary solution is predicted, then is this solution used to correct the final solution (Ferziger and Perić, 1996). For the initial value problem in equation (3.7), the first step of the method, with the explicit Euler method, will be:

$$f^*_{n+1} = f^n + R(t_n, f^n)\Delta t \tag{3.11}$$

The temporary solution f^*_{n+1} is used to correct the final solution. If a trapezoid rule is used, the final solution becomes:

$$f^{n+1} = f^n + \frac{1}{2}[R(t_n, f^n) + R(t_{n+1}, f^*_{n+1})]\Delta t \tag{3.12}$$

This method is second order accurate and has the same stability as the explicit Euler method (Ferziger and Perić, 1996).

3.4 Pressure Coupling

The momentum equations in two dimension together with the continuity equation gives three equations for three unknowns; two velocity components and the pressure. The pressure occurs in all the momentum equations, but there is no transport equation for the pressure. We need a coupling between the pressure and the velocity.

If we discretize the momentum equation (2.8) in the simplest way with the explicit Euler method we get:

$$\frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} + \mathbf{U}^n \cdot \nabla \mathbf{U}^n = -\frac{1}{\rho}\nabla p^n + \nu \nabla^2 \mathbf{U}^n + \mathbf{f}_b \tag{3.13}$$

The solution of equation (3.13) will give a new velocity $\mathbf{U}^{n+1}$ that does not satisfy continuity. Another issue is that there is no natural way to compute p^{n+1} . The latter can be overcome by introducing p^{n+1} in the scheme (Langtangen et al., 2002):

$$\mathbf{U}^{n+1} + \frac{\Delta t}{\rho}\nabla p^{n+1} = \mathbf{U}^n - \Delta t \nu \nabla^2 \mathbf{U}^n + \Delta t \mathbf{f}_b^n \tag{3.14}$$

This leaves two unknowns, $\mathbf{U}^{n+1}$ and p^{n+1}. To ensure that continuity is satisfied, equation (3.14) must be solved simultaneously with:

$$\nabla \cdot \mathbf{U}^{n+1} = 0 \tag{3.15}$$

3.4.1 Iterative Methods

There are two main approaches to take care of the pressure coupling; iterative methods and projection methods. The idea behind the iterative methods is that the momentum equations are repeatedly solved until the continuity criterion is satisfied.

An example of an iterative method is the SIMPLE scheme that is developed by Patankar and Spalding (1972). The starting point of the SIMPLE method is that a pressure field p^* is guessed, such that the momentum equations can be solved to obtain intermediate velocities $\mathbf{U}^*$. The exact pressure and velocity field is the sum of the intermediate values and a correction term:

$$\begin{aligned} p &= p^* + p' \\ \mathbf{U} &= \mathbf{U}^* + \mathbf{U}' \end{aligned} \tag{3.16}$$

If the expressions in equation (3.16) are inserted in the momentum and continuity equations, an expression for p' can be found. Then can the pressure be updated and a new iteration can take place, this process is repeated until the continuity condition is satisfied within a given criterion. This iterative process has to be performed at every time step.

3.4.2 Pressure Poisson Methods

The other approach is so-called *Operator splitting methods*, where the transport equations are divided into simpler equations that can be solved independently. This approach was first proposed by Chorin (1967) and is known as projection methods. However, the approach is further developed, and methods knowns as Operator splitting methods and fractional step methods are based on the same idea. These methods can differ in the formulation, but the similarity is that a Poisson-like equation is solved to take care of the pressure correction.

The advantage of these methods is that the transport equations are split into simpler equations, that can be solved by known solution schemes. In addition, there is no iteration involved, and continuity is satisfied exact.

An issue that arises for the pressure Poisson equation is suitable boundary conditions. In general, the pressure is not known at the boundaries. However, Gresho and Sani (1987) have demonstrated that Neumann boundary conditions always are appropriate. The boundary value problem can then be expressed as:

$$\nabla^2 P = f \quad on \quad \Omega \tag{3.17}$$

$$\frac{\partial P}{\partial n} = q \quad on \quad \Gamma \tag{3.18}$$

Where Ω represents the computational domain, and Γ represents the external boundaries.

Equation (3.17) and (3.18) are solvable for the pressure correction if the following equation is satisfied (Gresho and Sani, 1987):

$$\int_\Gamma q \quad d\Gamma = \int_\Omega f \quad d\Omega \tag{3.19}$$

This is known as the compatibility condition for the Neumann problem. Hence, if the initial velocity field is divergence free, q=0, and zero Neumann condition is applied at the boundaries.

3.5 Example of a Navier-Stokes Solver

In the following will the main steps of a Navier-Stokes solver developed by Giuseppina Colocchio (2004) be presented. The code is developed for more advanced flow problems including free surface. However, the terms that take care of the free surface are avoided in the description below.

The code discretize the Navier-Stokes equation with a predictor-corrector scheme. The momentum equations (2.8) can then be discretized as:

$$\frac{\mathbf{U}^{n+1}-\mathbf{U}^{n}}{\Delta t} = -\frac{1}{\rho}\nabla p^{n+1/2} - \mathbf{U}^{n+1/2}\cdot\nabla\mathbf{U}^{n+1/2} + \nu\nabla^2\mathbf{U}^{n+1/2} + \mathbf{f}_b \tag{3.20}$$

For simplicity the term $\mathbf{F}(\mathbf{U},\mathbf{f},\nu)$ is introduced:

$$\mathbf{F(U,f},\nu) = -(\mathbf{U}\cdot\nabla)\mathbf{U} + \nu\nabla^2\mathbf{U} + \mathbf{f} \tag{3.21}$$

Predictor Step

The values of the terms in $\mathbf{F}(\mathbf{U},\mathbf{f},\nu)$ at time step n+1/2 are approximated by a Taylor expansion from the previous time step.

The velocity can be found by the two-step procedure:

$$\begin{aligned}\tilde{\mathbf{U}} &= \mathbf{U}^{n} + \Delta t\Big\{[F(\mathbf{U})]_0^{n+1/2} - \frac{\nabla p^{n-1/2}}{\rho}\Big\}\\ \mathbf{U}_0^{n+1} &= \tilde{\mathbf{U}} + \Delta t\Big\{\frac{\nabla p^{n-1/2}}{\rho} - \frac{\nabla p_0^{n+1/2}}{\rho}\Big\}\end{aligned} \tag{3.22}$$

The continuity equations must be satisfied for $\mathbf{U}_0^{n+1}$, this implies that we can write the last part of equation (3.22) as:

$$\frac{\nabla\cdot\tilde{\mathbf{U}}}{\Delta t} = \nabla\cdot\Big(\frac{\nabla(p^{n-1/2}-p_0^{n+1/2})}{\rho} \tag{3.23}$$

The solution of the Poisson equation (3.23) gives p_0^{n+1}. Then, $\mathbf{U}_0^{n+1}$ can be calculated from equation (3.22). Now a corrector step is performed to improve the accuracy and stability.

Corrector Step

The corrector step is carried out iteratively until convergence in velocity and pressure is obtained. The term $[\mathbf{F}(\mathbf{U},\mathbf{f},\nu)]_k^{n+1/2}$ is obtained by a Taylor expansion, and the pressure at step k can be written as:

$$\frac{\nabla p_k^{n+1/2}}{\rho} = \frac{\nabla p_{k-1}^{n+1/2}}{\rho} + \frac{\nabla p_c}{\rho} \tag{3.24}$$

where p_c is a pressure correction. Now the velocity at step k is calculated similarly as for the predictor step:

$$\begin{aligned} \tilde{\mathbf{U}} &= \mathbf{U}^n + \Delta t\Big\{[F(\mathbf{U})]_{k-1}^{n+1/2} - \frac{\nabla p_{k-1}^{n-1/2}}{\rho}\Big\} \\ \mathbf{U}_k^{n+1} &= \tilde{\mathbf{U}} + \Delta t\Big\{ - \frac{\nabla p_c}{\rho}\Big\} \end{aligned} \tag{3.25}$$

And we obtain the Poisson equation:

$$\nabla \cdot \Big(\frac{\nabla p_c}{\rho}\Big) = \frac{\nabla \cdot \tilde{\mathbf{U}}}{\Delta t} \tag{3.26}$$

Now, can the velocity be updated, and this iterative process is repeated until convergence is obtained.

3.6 Solution of Equation Systems

Regardless of the numerical method that is used, a linear equation system on the form $Ax = b$ has to be solved. Solving the equation system is a considerable part of the solution procedure. There are two main solution techniques; direct solvers and iterative solvers.

The direct solvers solve the equation system exact, meaning no extra error is introduced. However, the number of operations to solve a system with N unknowns is of order N^3, and storage of all N^2 coefficients in core memory is necessary (Versteeg and Malalasekera, 1995).

The iterative solvers do not solve the equation system exact and will introduce an iteration error. However, the convergence criterion can be set beforehand and is usually of minor importance. With iterative solvers it is not possible, on beforehand, to know the exact number of operations to solve the equation system. However, for problems in two- and three dimensions, iterative methods are usually more efficient than direct solvers. Another advantage by iterative solvers is that only the non-zero elements have to be stored (Versteeg and Malalasekera, 1995).

There exist several libraries with solution routines for linear equation systems. The routines are customised to different type of equation systems, and the choice of solution routine should be done with care. Examples of such libraries are NAG, LAPACK and SPARSKIT.

MATLAB is a powerful computing environment, which includes a comprehensive library of functions. *mldivide* is one of these function, which are used to solve linear equation systems. *mldivide* is a self-adapting function that finds the most efficient solver to the given system.

Chapter 4

Presentation of the method

4.1 Harmonic Polynomial Cell Method

4.1.1 Concept

Shao and Faltinsen (2012) have formulated a numerical method for efficiently solving potential flow problems. Their numerical results have been promising, with a convergence rate between 3rd and 4th order for the L_2-error. It is indicated that the method is favourable compared to traditional BEM solvers, both regarding accuracy and CPU time. This section will present the method with a stencil with nine nodes, that use all polynomials, except one, up to fourth order. It is possible to use other stencils, with different geometry or number of nodes. This is reviewed in detailed in Ma et al. (2016). Although the method is presented for a potential flow problem, it is not limited to such problem and can be used for any problem governed by the Laplace equation.

The governing equation in potential flow theory is the Laplace equation, and harmonic polynomials satisfy the Laplace equation by definition (Kreyszig, 2011). In two dimensions, are the harmonic polynomials defined as:

$$z^n = (x + iy)^n = u_n(x, y) + i\nu_n(x, y) \tag{4.1}$$

Where $i = \sqrt{-1}$, and *n* is an integer which represents the order of the polynomial. The harmonic polynomials for $n <= 4$ can be found in table 4.1.

Table 4.1: Complex Harmonic polynomials for order ≤ 4

n	z^n	$u_n(x,y)$	$\nu_n(x,y)$
0	1	1	0
1	$x+iy$	x	y
2	$(x^2-y^2)+2ixy$	x^2-y^2	2xy
3	$(x^3-3xy^2)+i(3X^2y-y^3)$	x^3-3xy	$3x^2y-y^3$
4	$(x^4-6x^2y^2+y^4)+i(4x^3y-4xy^3)$	$x^4-6x^2y^2+y^4$	$4x^3y-4xy^3$

The computational domain is divided into quadrilateral cells, where the velocity potential within each cell is interpolated with harmonic polynomials. Each cell has nine nodes, and the local coordinate system has its origin at node nine. The node numbering is sketched in figure 4.1.

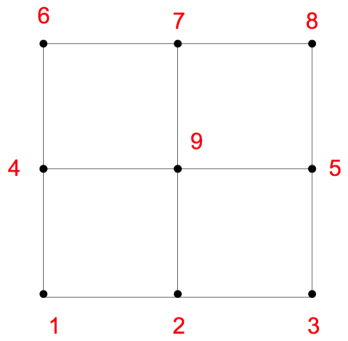

Figure 4.1: Local Cell Numbering

An overlapping cell system is used, which means that one node is used in several cells. Cell number "i" has local numbering 5 in one cell, and local number 9 in another cell and so on.

If the velocity potential is known for all nodes, the velocity potential within each cell can be interpolated from the following equation:

$$\phi(x,y) = \sum_{j=1}^{8} b_j f_j(x,y) \tag{4.2}$$

Where f_j are the first eight harmonic polynomials, they can be found in table 4.2.

Table 4.2: Harmonic polynomials included in interpolation

j	1	2	3	4	5	6	7	8
f_j	1	x	y	$x^2 - y^2$	xy	$x^3 - 3xy$	$3x^2y - y^3$	$x^4 - 6x^2y^2 + y^4$

4.1.2 Numerical Formulation

b_j in equation (4.2) are unknown coefficients that have to be determined. These coefficients can be found by plugging $\phi = \phi_j$, $x = x_j$ and $y = y_j$ (j=1,...8) into equation (4.2).

$$b_j = \sum_{j=1}^{8} c_{i,j}\phi_j, \qquad (i = 1, ..., 8) \tag{4.3}$$

Where $c_{i,j}$ is the elements of the inverse matrix $[f]^{-1}$, with $f_{i,j} = f_j(x_i, y_i)$.

By putting equation (4.3) into equation (4.2), the velocity potential within a cell can be expressed as:

$$\phi(x, y) = \sum_{i=1}^{8} \Big[\sum_{j=1}^{8} c_{j,i} f_j(x, y) \Big] \phi_i \tag{4.4}$$

Only the boundary nodes are used to interpolate the potential inside a cell. Since the local origin is placed at node 9, $f_1(0, 0) = 1$, and all other polynomials are zero at the origin, the velocity potential at node 9 can be expressed as:

$$\phi_9 = \phi(x = x_9 = 0, y = y_9 = 0) = \sum_{i=1}^{8} c_{1,i}\phi_i \tag{4.5}$$

Equation (4.5) is enforced for all cells without a boundary condition, and this provides the continuity of the flow.

Boundary Conditions

Proper boundary conditions must be applied, in order to solve a numerical problem This could be *Dirichlet boundary conditions* or *Neumann boundary conditions*. Dirichlet boundary conditions are enforced as:

$$\phi_B(x,y) = \phi(x,y) = \sum_{i=1}^{8} \Big[\sum_{j=1}^{8} c_{j,i} f_j(x,y) \Big] \phi_i \tag{4.6}$$

Where ϕ_B is the velocity potential at the boundary. Neumann boundary conditions are enforced in a similar way:

$$\frac{\partial \phi_B(x,y)}{\partial n} = \frac{\partial \phi(x,y)}{\partial n} = \sum_{i=1}^{8} \Big[\sum_{j=1}^{8} c_{j,i} \nabla f_j(x,y) \cdot \mathbf{n} \Big] \phi_i \tag{4.7}$$

For all nodes without any boundary condition, equation (4.5) is applied when the node is numbered nine in the local cell. This is the equation that connects the entire domain together.

The main steps in the solution algorithm of the HPC method are summarised below.

Local Coefficient Matrix

For each cell, an 8x8 coefficient matrix must be calculated

$$\mathbf{f} = \begin{bmatrix} f_1(x_1,y_1) & \dots & f_8(x_1,y_1) \\ \vdots & \ddots & \vdots \\ f_1(x_8,y_8) & \vdots & f_8(x_8,y_8) \end{bmatrix}$$

Where f_1-f_8 are the coefficients in table 4.2 and (x_i, y_i) i=1,..,8 are the local coordinates to border nodes. The inverse of this matrix is also calculated, $\mathbf{f}^{-1}$.

Local Equation System

The local equation system can be constructed based on the boundary conditions and equation (4.3).

$$\mathbf{c}\phi = \mathbf{b} \tag{4.8}$$

We want to solve the system for ϕ_i, and need to determine the elements in vector **b** and matrix **c**. For nodes with boundary conditions, b_j will be $\phi_B(x_j, y_j)$ or $\frac{\partial \phi_B(x_j,y_j)}{\partial n}$ for Dirichlet and Neumann boundary conditions respectively. If no boundary condition is applied to the specific node, $b_j = 0$.

The elements in matrix **c** will be equal to 1 or $f^{-1}(x_j, y_j)\frac{\partial f(x_j,y_j)}{\partial n}$, for Dirichlet and Neumann boundary conditions respectively.

The solution is unknown for most nodes, and equation (4.5) is applied to these nodes. This is the equation that connects the whole domain and is applied to all nodes that have no boundary condition. Hence, equation (4.9) is applied to most rows in the global coefficient matrix.

$$\begin{aligned} \mathbf{c}_{9,i} &= f_{1,i}^{-1} \\ \mathbf{c}_{9,9} &= -1 \end{aligned} \tag{4.9}$$

Global System

When all the local coefficients are calculated, these have to be placed in the right location in the global system. We want to find the velocity potential ϕ_i for all N=$(n_x + 1)(n_y + 1)$ nodes. This is obtained by solving the equation system:

$$\mathbf{C}\phi = \mathbf{B} \tag{4.10}$$

Where **C** is a (N,N) matrix with elements from the local matrices **c**, **B** is a vector with elements from the local **b** vectors, and ϕ is a vector with all the unknown velocity potentials.

C is a sparse matrix, which can be utilised when the equation system is built and for solving the global equation system.

4.2 Generalized HPC Method

4.2.1 Concept

The generalized HPC method is a further development of the HPC method, in order to solve non-homogeneous elliptic boundary value problems. Such problems are common in many fields of physics. One example is when the Navier-Stokes equations are solved by a projection method. Then you will have a Poisson equation for the pressure.

Bardazzi et al. (2015) have proposed a method that solves the Poisson equation with the generalized HPC method. Their test cases have shown a convergence rate of approximately 4th order. The method will be shortly described in the next paragraphs.

The domain is discretized in the same manner as for the HPC method described in section 4.1. Each cell has nine nodes with the local origin placed in the centre (see figure 4.1). The overlapping cell system is also used.

The Poisson problem for an unknown quantity u within a domain Ω can be expressed as:

$$\begin{aligned} \nabla^2 u(\mathbf{x}) &= \sigma(\mathbf{x}), && in \quad \Omega \\ u(\mathbf{x}) &= g_D(\mathbf{x}), && on \quad \partial\Omega_D \\ \frac{\partial u(\mathbf{x})}{\partial n} &= g_N(\mathbf{x}), && on \quad \partial\Omega_N \end{aligned} \tag{4.11}$$

Where $\partial\Omega_D$ and $\partial\Omega_N$ are boundaries with Dirichlet and Neumann boundary conditions respectively. The term $\sigma(\mathbf{x})$ is assumed to be known. The solution of this problem can be decomposed to a homogeneous part and a particular solution:

$$u(\mathbf{x}) = \bar{u}(\mathbf{x}) + \tilde{u}(\mathbf{x}) \tag{4.12}$$

The harmonic solution satisfies the homogeneous problem, corresponding to the Laplace equation:

$$\begin{aligned} \nabla^2 \bar{u}(\mathbf{x}) &= 0, && in \quad \Omega \\ \bar{u}(\mathbf{x}) &= g_D(\mathbf{x}) - \tilde{u}(\mathbf{x}), && on \quad \partial\Omega_D \\ \frac{\partial \bar{u}(\mathbf{x})}{\partial n} &= g_N(\mathbf{x}) - \frac{\partial \tilde{u}(\mathbf{x})}{\partial n}, && on \quad \partial\Omega_N \end{aligned} \tag{4.13}$$

The homogeneous solution is approximated with harmonic polynomials, as described in section 4.1. The particular solution has to satisfy:

$$\nabla^2 \tilde{u}(\mathbf{x}) = \sigma(\mathbf{x}) \tag{4.14}$$

The forcing term and the particular solution, for a two-dimensional problem, can then be approximated as:

$$\begin{aligned} \sigma(x,y) &\approx (\alpha_0 + \alpha_1 x + \alpha_2 x^2)(\beta_0 + \beta_1 y + \beta_2 y^2) \\ &= c_j h_j(x,y), \quad with \quad j = 1, ..., 9 \end{aligned} \tag{4.15}$$

$$\tilde{u}(x,y) \approx c_j g_j(x,y), \qquad with \quad j = 1, ..., 9 \tag{4.16}$$

Where h_j and g_j are coefficients that can be found in table 4.3.

Table 4.3: Coefficients h_j and g_j

j	1	2	3	4	5	6	7	8	9
h_j	1	x	y	x^2	xy	y^2	x^2y	xy^2	x^2y^2
g_j	$\frac{x^2+y^2}{4}$	$\frac{xy^2}{2}$	$\frac{x^2y}{2}$	$\frac{x^4}{12}$	$\frac{x^3y+xy^3}{12}$	$\frac{y^4}{12}$	$\frac{x^4y}{12}$	$\frac{xy^4}{12}$	$\frac{x^4(-x^2+15y^2)+y^4(-y^2+15x^2)}{360}$

With these approximations, the solution of equation (4.11) is:

$$\begin{aligned} u(x,y) &= a_i \bar{f}_i(x,y) + c_j g_j(x,y) \qquad with \quad i = 1, ..., 8 \quad j = 1, ..., 9 \\ \sigma(x,y) &= c_j h_j(x,y), \qquad\qquad\qquad with \quad j = 1, ..., 9 \end{aligned} \tag{4.17}$$

The approximations descried above are the same as proposed by Bardazzi et al. (2015). With accuracy of 4th order, both for the harmonic polynomials and the bi-quadratic polynomials. However, it is possible to approximate the particular solution with other types of polynomials.

4.2.2 Numerical Formulation

Now, the coefficients a_i and c_j in equation (4.17) need to be calculated. By investigating the forcing term $\sigma(x, y)$ at all nodes (x_k, y_k) (k=1,...,9), an explicit expression for c_j is obtained:

$$\sigma_k = h_{kj}c_j \Rightarrow c_j = [h^{-1}]_{jk}\sigma_k \quad j, k = 1, ..., 9 \tag{4.18}$$

Similarly, by investigating the unknown property u at boundary nodes (x_m, y_m) (m=1,...8), an expression for a_i is obtained.

$$\begin{aligned} &u_m = \bar{f}_{mi}a_i + g_{mj}c_j \Rightarrow a_i = [\bar{f}^{-1}]_{im}(u_m - g_{mj}c_j) \\ &i, m = 1, ..., 8; \quad j = 1, ..., 9 \end{aligned} \tag{4.19}$$

Plugging equation (4.18) and (4.19) into equation (4.17), the solution of equation (4.11) can be written as.

$$\begin{aligned} &u(x, y) = \bar{f}_i(x, y)[\bar{f}^{-1}]_{im}(u_m - g_{mj}[h^{-1}]_{jk}\sigma_k) + g_j(x, y)[h^{-1}]_{jk}\sigma_k \\ &i, m = 1, ..., 8; \quad j, k = 1, ..., 9. \end{aligned} \tag{4.20}$$

The derivative of the solution can be obtained as:

$$\frac{\partial u(x, y)}{\partial n} = \frac{\partial \bar{f}_i(x, y)}{\partial n}[\bar{f}^{-1}]_{im}(u_m - g_{mj}[h^{-1}]_{jk}\sigma_k) + \frac{\partial g_j(x, y)}{\partial n}[h^{-1}]_{jk}\sigma_k \tag{4.21}$$

with i,m=1,...,8, and j,k=1,...,9.

To build the equation system, equation (4.20) is evaluated at the centre of the cell, for nodes where the solution is unknown. For nodes where the solution is known, equation (4.20) or (4.21), for Dirichlet and Neumann BC respectively, is applied to boundary nodes.

For each cell a linear equation system can be built in the following form:

$$b = F^T\mathbf{u} - G^T\sigma \tag{4.22}$$

where b represents boundary data. This equation system has to be constructed for all cells and put together in a global equation system. The global system can then be solved for the unknown property **u**.

4.3 Immersed Boundary Grid

For a rectangular domain with regular boundaries, the implementation of boundary conditions is straightforward. However, for most practical applications the boundaries are highly irregular, and the boundaries do not, in general, coincides with computational nodes. There are several methods to overcome this challenge. A popular method is to use a body-fitted grid. However, time marching problems with moving boundaries requires remeshing every time step, which can be a considerable computational effort.

The immersed boundary method (IBM) is a method that fulfils the boundary conditions at the irregular boundaries in a Cartesian grid. This simplifies the implementation and could reduce the computational cost. Immersed boundary methods can be classified into two main categories. The first category represents the immersed boundary as a "diffuse" interface of finite thickness. The second method represents the boundary as a sharp interface (Udaykumar et al., 2001). The IBM implemented in this thesis is within the second category, sharp interface method.

A sharp interface IBM method together with the HPC method is implemented by Finn C. W. Hanssen in Hanssen et al. (2015) and Ma et al. (2016). They have named the method *"Immersed Boundary Grid"* (IBG). The implementation of the IBG was the primary focus in the project thesis (Rabliås, 2016). The IBG described below is based on the method implemented by Finn C. W. Hanssen. The method is described for a potential flow around a circular cylinder, but the method is general and could be used for other problems, e.g. the pressure Poisson equation in a fractional step method.

For a potential flow around a cylinder, the following Neumann condition has to be enforced:

$$\frac{\partial \phi}{\partial n} = 0 \tag{4.23}$$

The boundary condition will not be fulfilled at the entire surface, but for a discrete number of points.

$$\nabla \phi(x_b, y_b) \cdot \mathbf{n}(x_b, y_b) = 0 \tag{4.24}$$

Where (x_b, y_b) is the coordinate of "markers" at the cylinder. The boundary conditions at the markers are fulfilled by using "ghost nodes" inside the cylinder.

A "ghost node" is a node that belongs to a cell that intersects the surface. Each marker is connected to a ghost node inside the cylinder.

When a cell with a ghost node and a marker are found, the following equation are enforced to fulfil the Neumann condition at the marker:

$$\begin{aligned} \nabla \overline{f}_i(x_b, y_b) \cdot \mathbf{n}(x_b, y_b)[\overline{f}^{-1}]_{im}(\phi_m - g_{mj}[h^{-1}]_{jk}\sigma_k) \\ +\nabla g_j(x_b, y_b) \cdot \mathbf{n}(x_b, y_b)[h^{-1}]_{jk}\sigma_k = \nabla \phi(x_b, y_b) \cdot \mathbf{n}(x_b, y_b) \end{aligned} \tag{4.25}$$

This is the general expression for a Poisson problem. For the potential flow problem, which is a Laplace problem, the forcing term (σ_k) is zero and the expression simplifies.

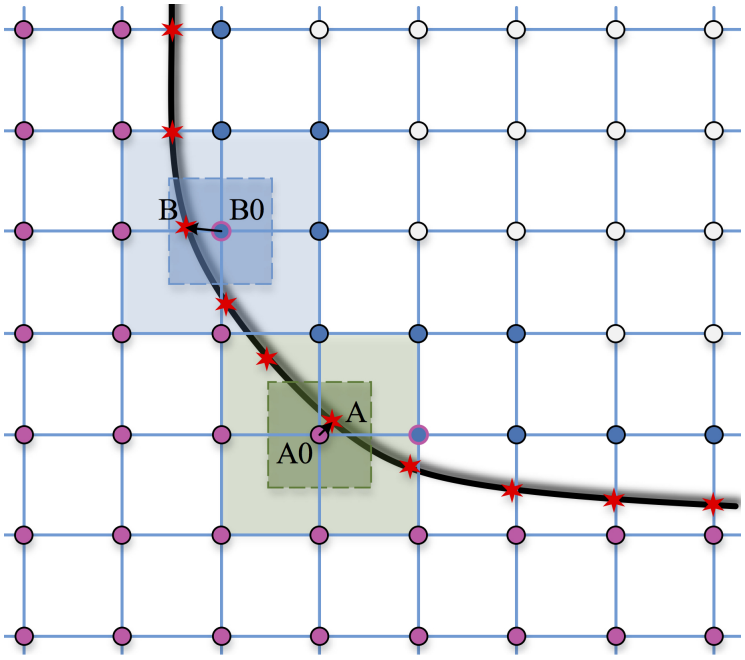

Figure 4.2: Immersed boundary grid (Ma et al., 2016)

Figure 4.2 illustrates an immersed boundary grid. The purple nodes represent nodes in the computational domain, the blue nodes represents "ghost nodes", and the red stars represent markers where the boundary conditions actually are satisfied.

Chapter 5

Numerical Implementation

5.1 Description of Problem

5.1.1 Potential Flow Around a Circular Cylinder

In the project thesis (Rabliås, 2016), a potential flow around a circular cylinder was investigated. The governing equation for such flow is the Laplace equation. This equation is equivalent to the Poisson equation with the forcing term set to zero. Hence, the generalized HPC method can be used to solve the same problem as in the project thesis.

A thorough test of the IBG that was implemented in the project thesis revealed that the convergence rate was very sensitive to the number of grids that were included. The aim of this part of the project, is to improve the IBG that was implemented in the project thesis.

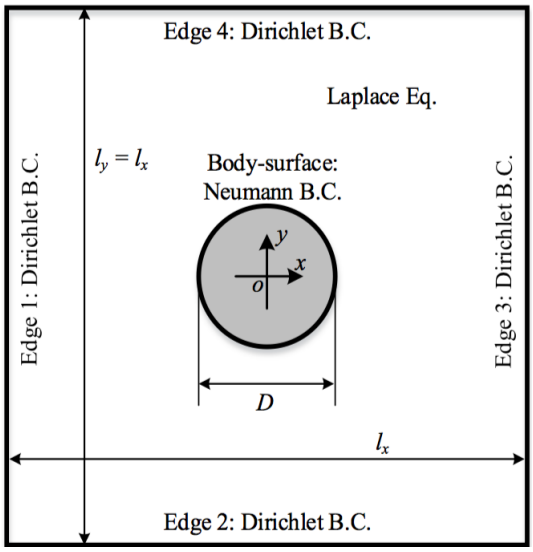

Figure 5.1: Computational domain of a potential flow around a circular cylinder. (Ma et al., 2016)

The computational domain is sketched in figure 5.1. A uniform flow with velocity $U = 1m/s$ around a circular cylinder with a diameter of $D = 10\ m$, was considered. The length and height of the domain were 4*D, and the cylinder was placed in the centre of the domain. For this potential flow problem the analytical solution is given by:

$$\phi_{an}(x, y) = U\Big(x + \frac{x}{x^2 + y^2}\frac{D^2}{4}\Big) \tag{5.1}$$

The generalized HPC method, as it is described in chapter 4, will be applied to this problem. The error and convergence rate will be investigated and compared with the results from the project thesis.

5.1.2 Viscous Flow around a Square Cylinder

The viscous flow problem that is investigated is a uniform flow around a square cylinder. The square cylinder is chosen such that additional errors introduced by the implementation of the body boundary conditions are kept at a minimum. This is an advantage when the generalized HPC method is compared with another Poisson solver. Primarily, the objective is to compare the method itself and not the implementation of the boundary conditions.

The problem is solved by a Navier-Stokes solver developed by Colocchio (2004). The main steps of the solution scheme was reviewed in section 3.5.

The original code is written in Fortran and uses a finite difference scheme to solve the pressure Poisson equation. If possible, a second order nine stencil scheme is applied. If that is not possible the five stencil scheme in section 3.2.2 is applied. The corresponding equation system is solved with a SPARSKIT solver.

The generalized HPC method is implemented in the Navier-Stokes solver developed by Giuseppina Colocchio. Since this code is written in Fortran and the generalized HPC method is implemented in MATLAB, it is necessary to make a proper interface between the two environments. This is further reviewed in section 5.2.4.

The code will be tested when the pressure Poisson equation is solved with both the generalized HPC method and the FDM scheme.

The flow is investigated in the stable laminar regime. The Reynolds number is set to 40, which means that the flow gets stable after a certain time. Hence, time variations are removed. This makes the sampling easier, and the sources of errors are minimised when two different solvers are compared. The inflow velocity and the cylinder diameter is set to unity, the desired Reynolds number is then obtained by changing the viscosity.

Discretization and Boundary Conditions

Properly boundary conditions must be applied in order to solve the problem. The square cylinder is placed in an infinite domain with a uniform current.

The horizontal velocity is set to 1 m/s at inflow and outflow. At the upper and lower boundary a Neumann condition is used for the horizontal velocity $\frac{\partial u}{\partial y} = 0$. The vertical velocity is set to zero at all boundaries, while a zero Neumann condition ($\frac{\partial p}{\partial n} = 0$) is applied for the pressure at all boundaries.

To get accurate results, it is important to choose the computational domain such that the external boundary conditions are realistic. If the computational domain is too small, could the boundary conditions enforce an unphysical solution that evolves to the interior of the computational domain.

The square cylinder has sides of length 1 and is placed in a square domain. The distance from the centre of the cylinder to the inflow is 7*D, the distance to the outflow is 15*D, and the height of the computational domain is 12*D. A sketch of the computational domain and the corresponding boundary conditions can be found in figure 5.2.

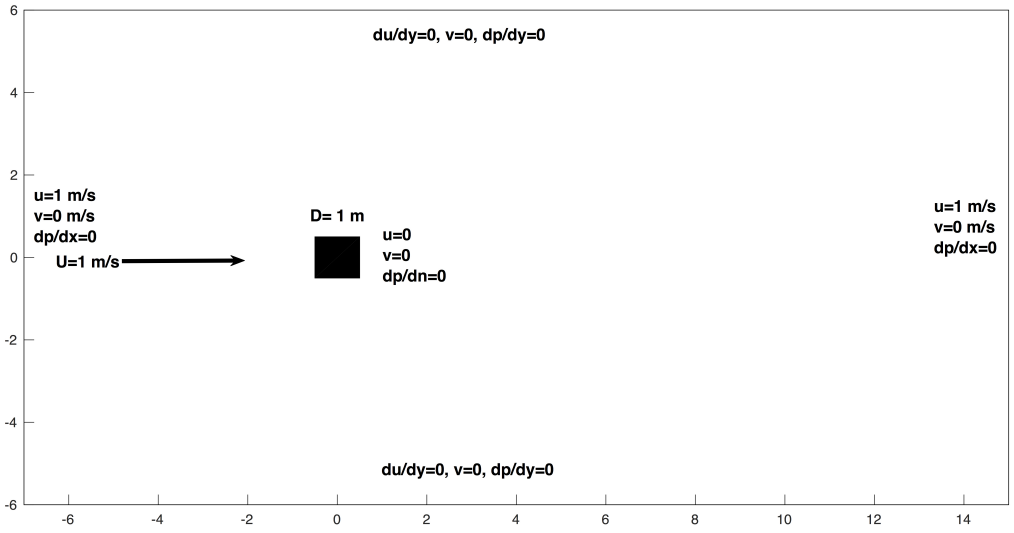

Figure 5.2: Computational domain for uniform flow around a square cylinder

It is difficult to predict the optimal size of the computational domain a priori. Ideally, a sensitivity analysis, where the parameters of the computational domain are altered, should have been performed. However, the main objective of this thesis is not to get as accurate results as possible, but to compare the performance of a new Poisson solver with a traditional solver in the same framework. If the objective was to perform a detailed investigation of the hydrodynamic behaviour of a flow around a square cylinder, it is possible to do several measures to improve the accuracy. Such measures could, for example, be to refine the grid in specific regions or increase the computational domain. However, it is assumed that the parameters that are chosen are sufficient to test the performance of the generalized HPC method.

A uniform mesh with square cells is used in the calculations. Meaning, that $\Delta x = \Delta y = constant$ for the entire computational domain. This excludes the opportunity for a refined mesh close to the body, which is necessary for detailed and accurate results. However, this approach simplifies the implementation, and it is considered to be accurate enough for the comparison.

All parameters are kept constant when the two Poisson solvers are compared, which means that domain size, mesh size and time step are the same. The only thing that is changed is the Poisson solver.

5.2 Programming Features

5.2.1 Immersed Boundary Grid

The boundary value problem described in section 5.1.1 is solved by implementing the immersed boundary grid that is presented in section 4.3. The first step of the method is to identify the cells at the cylinder surface, these cells are named ”ghost cells”. Then are the boundary conditions (equation (4.25)) enforced at these cells. The nodes in these cells, that lies inside the cylinder, are so-called ”ghost nodes”. All these ”ghost nodes” must either be connected to a marker on the cylinder surface or be a centre node in another ”ghost cell”. The choice of ghost cells and markers follows the following rules:

1. The marker should be as close as possible to the centre of the cell. The marker should at least be closer than 0.5*dx to the local y-axis and closer than 0.5*dy to the local x-axis.

2. The distance between markers should be as equal as possible.

3. If possible, enforce only one BC in each ghost cell.

4. Try to have overlapping ghost cells.

It is impossible to satisfy all the rules listed above simultaneously, and the implementation will be some kind of compromise. The two first rules are recommended by Ma et al. (2016). In addition to these rules, it is important that each ghost node and marker is used only once. Meaning that a ”ghost node” can be connected to only one marker, and each marker can be connected to only one ”ghost node”. If this is violated, the global equation system could be corrupted.

The first attempt to implement the IBG focused most on the first three rules. However, the accuracy and the convergence rate were not as good as expected. After consultation with PhD student Finn-Christian W. Hanssen, the fourth rule was added to the algorithm while the second rule was given less importance. This improved the accuracy, and the desirable convergence rate was obtained.

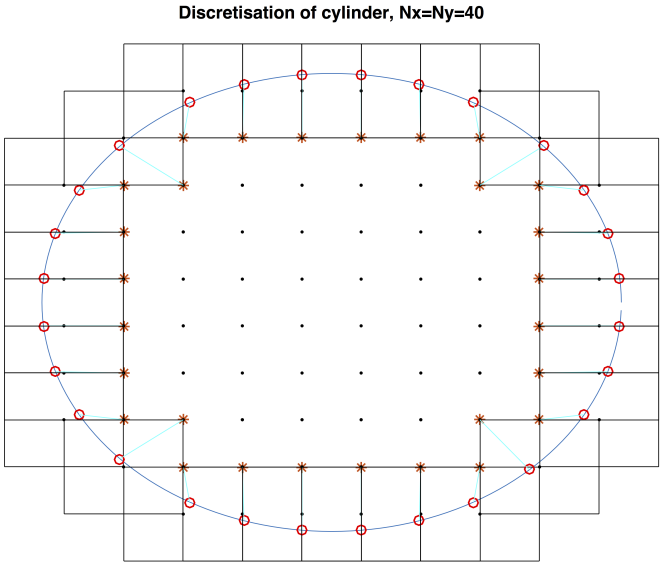

Figure 5.3: Ghost Cells at cylinder surface

Figure 5.3 illustrates the ghost cells for a grid with 40 nodes in x- and y-direction. The stars represent the ghost nodes that are used for the boundary conditions, the red circles represent the markers where the boundary condition are enforced. The outline of the ghost cells is also sketched in the figure. As you can see, the markers are close to the centre of the cells, and the cells are overlapping. When these two conditions are fulfilled, will the distance between markers implicitly become quite equidistant.

5.2.2 Poisson Solver

The Poisson problem is solved numerical with the generalized HPC method proposed by Bardazzi et al. (2015). The method is reviewed in section 4.2. Andrea Bardazzi, one of the authors, has kindly in confidentiality, shared his code that was used for some of the example problems in Bardazzi et al. (2015).

The code received was made for solving the pressure field in a Green-Taylor vortex, without an internal body. The main implementation has been to properly place a body inside the computational domain.

The code by Andrea Bardazzi was written in MATLAB, in order to exploit the framework and embedded MATLAB functions, most of the implementation in this thesis is also done with MATLAB. Since the code was shared in confidentiality, no code is presented here. However, the following sections will explain the main steps of the code.

Coupling with Navier-Stokes Solver

The Navier-Stokes code utilises a staggered grid to compute velocities and pressure. This means that the pressure is calculated in the centre of the cells while the velocities are calculated at the cell faces.

At first, a ”blind coupling” between the main code and the Poisson solver was trialled. Meaning that input for the Poisson solver was the forcing term, and the output was the pressure in the same position as the forcing term was given. These points were placed such that the pressure nodes coincided with the physical boundary. This implies that the forcing term, as well as the pressure gradients, were calculated in the Naiver-Stokes solver, and then interpolated to the correct position. It turned out that this approach did not give a divergence free velocity field, i.e. continuity was not satisfied.

To improve the results, a new approach was implemented. Now, the input for the Poisson solver was the velocity at the faces. The horizontal velocity components were given on the vertical faces, and the vertical velocity components were given on the horizontal faces, which is consistent with the staggered grid arrangement. The forcing term can then be interpolated at the centre of the cells with central differences, using the velocities at the surrounding faces:

$$\nabla \cdot \tilde{\mathbf{U}} = \frac{u_{i+1,j} - u_{i,j}}{x_{i+1,j} - x_{i,j}} + \frac{v_{i,j+1} - v_{i,j}}{y_{i,j+1} - y_{i,j}} \tag{5.2}$$

The accuracy of equation (5.2) is second-order for an equidistant grid (Ferziger and Perić, 1996).

When the forcing term is computed for all pressure nodes, the Poisson equation is solved with the generalized HPC method.

The values that are needed in the Navier-Stokes solver is not the pressure, but the pressure gradients. To be consistent with the staggered grid, the output is then $\frac{\partial P}{\partial x}$ on the faces where the horizontal velocities are defined, and $\frac{\partial P}{\partial y}$ on the faces where the vertical velocities are defined.

Since the generalized HPC method gives a continuous approximation close to fourth order between the computational nodes, the derivatives can be approximated with equation (4.21). However, the solution between computational nodes is not unique and which cell to use for the interpolation must be decided. Ma et al. (2016) recommend that the point for local interpolation should be chosen such that $x_{0c} <= |0.5dx|$ and $y_{0c} <= |0.5dy|$, where x_{0c} and y_{0c} is the distance from the local x- and y-axis respectively. The rule that is applied, is that cell (i,j) is used to interpolate the derivatives at the western and southern faces, which corresponds to the faces where $u_{i,j}$ and $v_{i,j}$ are defined. The distance from the centre of the cell is then $(x, y) = (-0.5dx, 0)$ and $(x, y) = (0, -0.5dy)$ respectively. This is within the range that is recommended to obtain good accuracy for the local interpolation.

The point where the derivative is calculated can be interpolated from two different cells, with equal distance to the centre of the respective cells. As a sensitivity study, where the gradients computed from both these cells. It turned out that the deviation was in the order of machine precision.

The final implementation is more consistent with the staggered grid arrangement than the first approach. All terms are calculated exactly where they are needed, and unnecessary interpolation is avoided.

The discretization of equation (3.23) and (3.26) is also more consistent when the second approach is applied. To satisfy continuity, it is important that the terms are discretized in the same manner as they are discretized in the momentum equations (Ferziger and Perić, 1996).

Main Routine

The main routine discretizes the domain in overlapping cells as described in chapter 4. External boundaries are identified and corresponding node numbers are saved in the vectors; Nb1, Nb2, Nb3 and Nb4. For each cell, the coefficients matrices **f, h** and **c** are calculated and put in the global matrices **F** and **G.** Eventually we want to solve the global system:

$$\mathbf{P} = \mathbf{F}^{-1}(\mathbf{B} + \mathbf{G}\Sigma) \tag{5.3}$$

B is a vector that contains boundary data, all elements except boundary nodes (Nb1,Nb2,Nb3 and Nb4) are zero. For boundary nodes, the elements in **B** equals the boundary condition that is enforced at the specific node.

F and **G** are sparse matrices, with at most nine non-zero elements in each row. This reduces the required memory compared to dense matrices. MATLAB also exploits the sparsity when the linear equation system (equation (5.3)) is solved. The global system is solved in MATLAB with *mldivide* , a self-adaptive solution algorithm that finds the best solution method corresponding to the size and the sparsity of the equation system.

In a staggered grid do the pressure nodes, in general, not coincide with boundaries. For the problem investigated here do the left and right boundary of the computational domain coincide with horizontal velocity nodes, and the upper and lower boundaries coincides with vertical velocity nodes. This means that, for the initially grid, can pressure boundary conditions not be applied directly to the nodes.

The boundary conditions at the external boundaries are applied by adding ”ghost nodes” in the exterior. The Immersed Boundary Grid that is presented in section 5.2.1 is used to satisfy the boundary conditions at the physical boundary. The same approach is used to satisfy the body boundary conditions on the square cylinder, and a more detailed description is given in the next section.

Implementation of a Square Cylinder

A square cylinder is one of the simplest bodies to implement in a staggered grid. The reason for this is that the sides of the cylinder are parallel with the global coordinate system. Hence, the physical boundary is mid between computational nodes, and neighbour nodes can easily be identified.

The body boundary conditions are implemented with the immersed boundary grid presented in section 4.3 and 5.2.1. For a square body the implementation simplifies since we know exactly which nodes to use in order to satisfy the rules that were outlined in section 5.2.1. The markers will be placed in the exact same positions relatively to the local cell, independent of grid size.

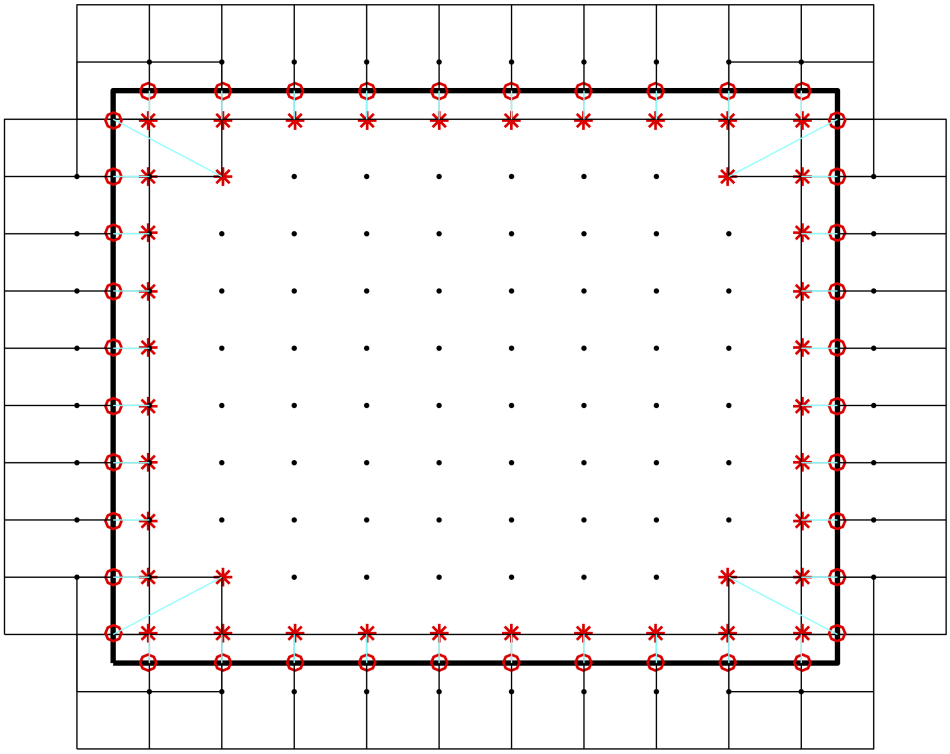

Figure 5.4: Ghost cells for a square cylinder

The immersed boundary grid is sketched in figure 5.4. The red stars represent the ”ghost nodes”, while the red circles represent the markers on the cylinder surface. The outline and the centre of the ”ghost nodes” are also sketched in the figure.

As you can see are the markers equidistant for all the sides, the cells are overlapping, and all markers are a distance $0.5dx$ from the centre of the cells. This set up is the same for all grids.

The framework from the main routine can be used to implement the square cylinder in the computational domain. The nodes that lie inside the body, next to the boundary must be identified. These nodes are saved in the vectors; Nb5, Nb6, Nb7 and Nb8

5.2.3 Solving the Pressure Poisson Equation

The pressure is computed by solving a Poisson equation on the form:

$$\nabla^2 P = \nabla \cdot \bar{\mathbf{U}} \tag{5.4}$$

The solution of the pressure Poisson equation is obtained by solving (5.3) every time step. The only term in equation (5.3) that changes with time is the forcing term Σ. This means that the procedure can be split into two steps. The first step computes the coefficient matrices **A** and **G**, and the vector **B**. While the second step solves the equation system (5.3).

The first step includes a loop through all nodes and matrix inversions are performed every iteration. These operations could be quite time-consuming. To make the algorithm more efficient is the first step performed only once, at the beginning of the simulation. Hence, the only operations that are performed every time step, is to update the forcing term and solve equation (5.3).

For problems with moving boundaries, the coefficient matrices must also be updated every time step.

The next paragraphs review how the pressure correction was implemented, and issues that arose. Especially considering how to solve the singular equation system that is obtained with pure Neumann boundary conditions.

Fixing the Pressure in One Node

The boundary value problem obtained from the pressure Poisson equation with pure Neumann boundary conditions produce a singular equation system. A widely used method to solve a singular equation system is to fix the pressure at one node in the computational domain. This approach was initially implemented in the code. A node at the inflow, close to y=0, was set to p=0.

At first, this method seemed to give reasonable results. However, after a more detailed review of the results it turned out that the Dirichlet node caused an unphysical pressure gradient.

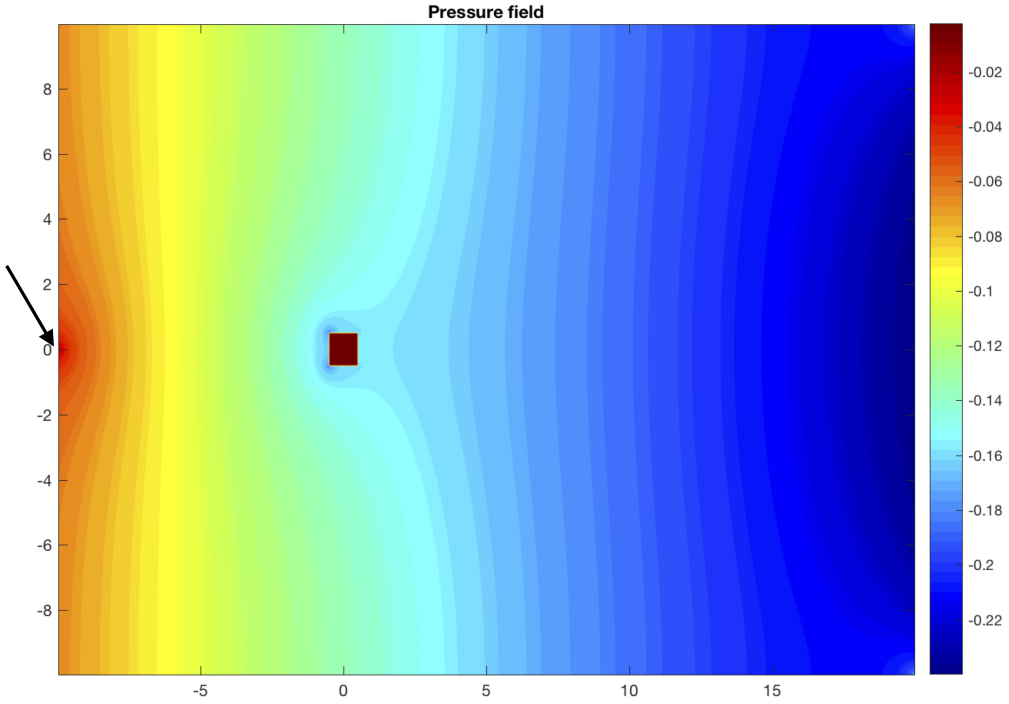

Figure 5.5: Unphysical Pressure Gradient at Inflow

In figure 5.5 can you clearly see that the Neumann condition is violated close to the Dirichlet node at the inflow. If the Dirichlet node was moved to another position, the disturbance was also relocated.

Several measures were tried to get rid of the disturbance:

- The Neumann boundary conditions at the external boundaries were implemented with ”ghost nodes” at the physical boundary and distorted cells, as reviewed by Ma et al. (2016).
- The Neumann boundary conditions at the external boundaries were implemented with ”ghost nodes” in the exterior and a symmetry condition. Similar as the method that is common with FDM.
- The two points above were also applied for the boundary conditions at the square cylinder.
- The pressure gradients that is used in the momentum equations were approximated with a finite difference scheme, instead of HPC interpolation.
- The node with Dirichlet boundary condition was moved to different locations in the computational domain.

For all measures listed above, it was verified that the boundary conditions were satisfied. However, none of the measures resulted in a significant improvement.

Pure Neumann Boundary Conditions

Since the Dirichlet node caused some disturbance, was the next attempt to solve the full Neumann problem, without any Dirichlet condition. This is a singular system, with the issues that causes.

The first approach in order to solve the singular system, used the *mldivide* function in MATLAB, without any modifications. To obtain a unique pressure field, was the pressure shifted such that the pressure at the inflow at $y \approx 0$ was set to zero. This resulted in a pressure field without the disturbance. The pressure field obtained for a given velocity field can be found in figure 5.6.

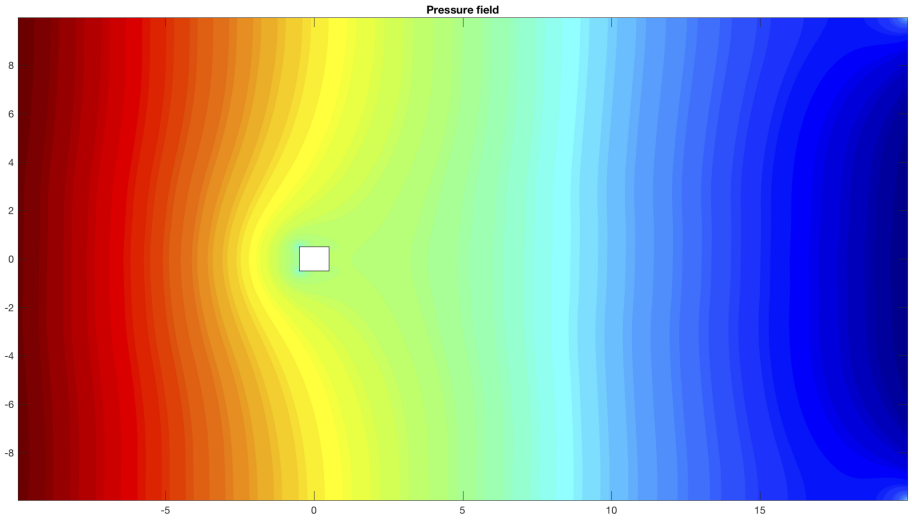

Figure 5.6: Pressure Field from Pure Neumann Conditions

Unfortunately, when the routine was implemented in the Navier-Stokes solver, the resulting pressure field was not physical. Moreover, it turned out that the results oscillated with time. This is not physical since the flow was investigated in the stable region. The pressure field after 30 seconds can be found in figure 5.7

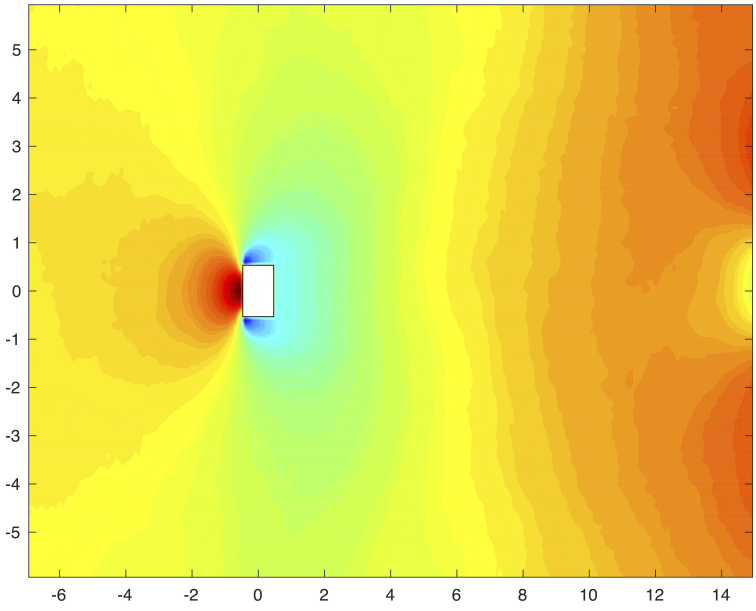

Figure 5.7: Pressure Field from Pure Neumann Conditions after 30 Seconds

As you can see from figure 5.7, are there some disturbances in the pressure field, and it is not symmetric.

To resolve this, a large amount of literature were studied. Finally, by following the approach that is implemented by Escobar-Vargas et al. (2014) the results were improved. The main steps are reviewed in the next paragraphs.

When the Poisson equation is discretized, an equation system on the form $\mathbf{Ap} = \mathbf{b}$ is obtained, where $\mathbf{p}$ is the unknown pressure, $\mathbf{A}$ is a coefficient matrix, and $\mathbf{b}$ contains boundary data and data from the forcing term. This system is consistent if (Golub and Van Loan, 1996):

$$\mathbf{u}_0^{\mathbf{T}}\mathbf{Ap} = \mathbf{u}_0^{\mathbf{T}}\mathbf{b} = 0 \tag{5.5}$$

Where $\mathbf{u}_0$ is the left null singular vector of matrix $\mathbf{A}$. If the compatibility condition (equation (3.19)) in discrete form is not exactly satisfied, equation (5.5) will nor be satisfied (Pozrikidis, 2001). Equation (5.5) is therefore known as the discrete compatibility condition.

This is also discussed in Gresho and Sani (1987) and Henshaw (1994). To ensure that equation (5.5) is satisfied, a modified equation system can be solved (Pozrikidis, 2001):

$$\mathbf{Ap} = (\mathbf{I} - \mathbf{u}_0\mathbf{u}_0^{\mathbf{T}})\mathbf{b} = \hat{\mathbf{b}} \tag{5.6}$$

Where $\mathbf{I}$ is the identity matrix.

Solving equation (5.6) resulted in a symmetric pressure field without disturbance. Moreover, the results were stable in time.

Equation (5.6) requires that a singular value decomposition (svd) of the coefficient matrix to be performed, in order to get the left null singular vector. This could be a very time-consuming task. Moreover, when the number of unknowns gets high, the matrix multiplication $\mathbf{u}_0\mathbf{u}_0^{\mathbf{T}}\mathbf{b}$ becomes very memory demanding in MATLAB, since the full matrix must be stored. $\mathbf{u}_0\mathbf{u}_0^{\mathbf{T}}$ is a (NxN) dense matrix, while the final product $\mathbf{u}_0\mathbf{u}_0^{\mathbf{T}}\mathbf{b}$ is a vector.

The singular value decomposition is only performed once, at the beginning of the simulation. This is performed with the MATLAB function *svds*. The matrix multiplication is performed with the built-in MATLAB function if the required memory is lower than available RAM. If the matrix size exceeds this threshold, the multiplication is performed inside a loop. This reduces the memory requirement since the resulting matrix is a vector.

It is obvious that solving equation (5.6) is more time consuming than solving the original equation system. Especially since loops are relatively slow in the MATLAB environment, compared with for example Fortran. However, the computational time is reduced some by using *parfor*, which is a feature in MATLAB to parallel the loop. The matrix multiplication was then computed in a loop paralleled to 16 CPUs.

There exist numerical methods to find the left null singular vector that is faster than the *svds* command in MATLAB, this is reviewed in Escobar-vargas (2012). However, since this command is executed only once each simulation, and the deadline was getting close at this stage of the implementation, equation (5.6) was implemented without any further modifications.

5.2.4 Interface Between MATLAB and Fortran

The Poisson solver and the associated routines are written in MATLAB, while the Navier-Stokes solver is written in Fortran. There are two main ways to make a working interface between MATLAB and Fortran. The first method is to export the MATLAB routines to a dynamic library that can be called from a Fortran program. This is not officially supported by Mathworks and there exists no documentation for the topic. There exist documentation of doing this coupling with MATLAB and C, and it should be possible to do the same thing with Fortran.

The second method is to open MATLAB in batch mode from the Fortran code, this is called "MATLAB engine". The latter method is officially supported by Mathworks and there exists proper documentation for the topic. With MATLAB engine is the calculations are performed by MATLAB, meaning that MATLAB needs to be installed on the computer that runs the Fortran program.

The advantage of the MATLAB engine approach is that it is less complicated to implement compared to the first method. Also, since MATLAB runs the MATLAB routines it is faster to do changes in the code. If the MATLAB engine is properly implemented in the Fortran program, it is not necessary to recompile the program if you want to do any changes in the routines. Such changes are done directly in the MATLAB source code.

The drawback of the MATLAB engine is that a version of MATLAB needs to be installed. This makes it more difficult to distribute the program to users without the software. Since the routines are calculated by MATLAB, it is usually slower than if the routines have been run by Fortran.

Initially, there was an attempt to implement the first interface method. However, it turned out that it was not straight forward, and difficult to succeed without any documentation. Then was the "MATLAB engine" successfully implemented with Fortran.

To properly make the interface between MATLAB and Fortran with the MATLAB engine, it is necessary to make a Fortran subroutine that opens the engine and transfer data between the Fortran program and MATLAB workspace.

Since the routine can be split into two steps, two interface routines were made. HPC_coeff that calculates the coefficient matrices is called at the beginning of the simulation. While $solve\_HPC$ that solves equation (5.3), is called every time step.

5.2.5 Post Processing

The output from the Navier-Stokes solver is the velocity components and the pressure in the centre of the cell. To evaluate flow features like drag and wake, these primitive variables must be post processed.

Drag Coefficient

The drag force can be computed from equation (2.17), and is divided into a contribution from normal pressure and a contribution from shear stress.

Shear Stress

The shear stress is given by equation (2.19), and $\frac{\partial u}{\partial y}$ must be approximated. A second order scheme is used to approximate the derivative (Tannehill et al., 1997):

$$\left(\frac{\partial u}{\partial y}\right)_w = \frac{-3u_w + 4u_{i,j+1} - u_{i,j+2}}{1.5\Delta y} \tag{5.7}$$

Where $u_w = 0$ and represents the velocity at the boundary, and $u_{i,j+1}$ and $u_{i,j+2}$ represent the two nodes outside the boundary in the normal direction.

For a square cylinder, only the top and bottom side of the cylinder contributes to the shear drag.

Normal Pressure

The drag contribution from the normal pressure is the normal pressure that is acting parallel with the global x-axis. For a square cylinder, this corresponds to the sum of the pressure force at the left and right side of the cylinder (when proper normals are applied).

The pressure is known at computational nodes, which do not coincide with the solid surface. The pressure at the cylinder surface is obtained with the MATLAB function *interp2*, which use the value at the computational nodes to interpolate the value at the boundary.

Integration

The shear stress and normal pressure are computed for a discrete number of points, which correspond to the number of computational nodes that lies on the cylinder side, if the boundary had coincided with the nodes. This is visualised in figure 5.8, where the solid line represents the physical boundary, the black dots represents some of the closest computational nodes, and the red dots represent the points at the surface where the values are interpolated.

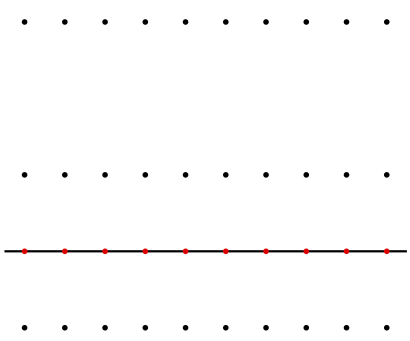

Figure 5.8: Computational nodes close to boundary

The boundary is divided into a number of line segments with equal length dA. The total drag is obtained with the numerical integration:

$$F_{D,num} = \sum_{i}^{N} \tau_{w,i} dA_{\tau} + \sum_{i}^{M} P_{w,i} dA_{P} \tag{5.8}$$

Where $\tau_{w,i}$ is the shear stress, for the discrete points, at the top and bottom of the square cylinder, $P_{w,i}$ is the normal pressure, for the discrete points, at the left and right side of the cylinder, dA_{τ} is the length of the line segments where the shear stress is computed, and dA_P is the length of the line segments where the pressure is computed.

Streamlines

The streamlines are calculated with the MATLAB function *streamline*. This function takes the velocity and the number of wanted streamlines as input.

Sampling

Even if the flow around an obstacle is stable for small Reynolds numbers it is important to have a good strategy when the results are sampled. It takes some time before the flow stabilises, and it is important that the results are sampled after the flow is stabilised.

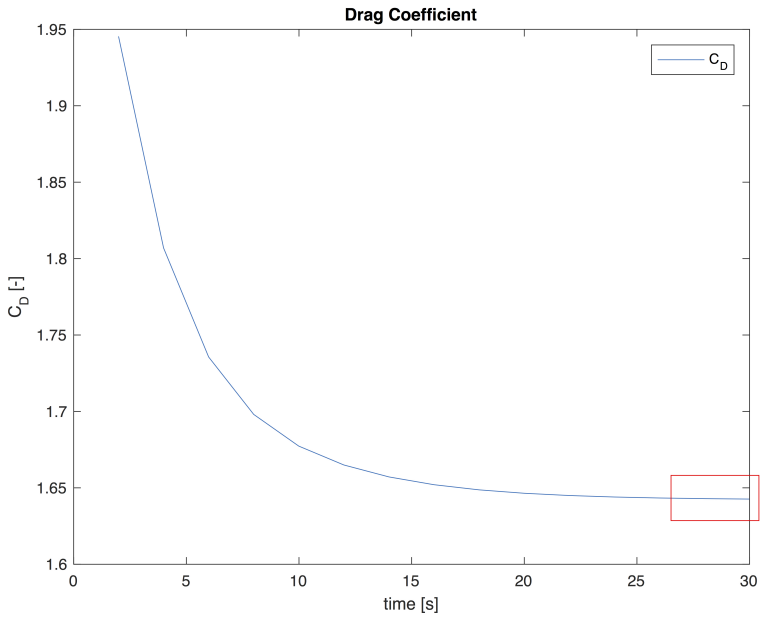

Figure 5.9: Time Variation of Drag Coefficient

Figure 5.9 illustrates how the drag coefficient varies with time. It is clear that it is necessary to simulate for a while to get reliable results. Tests revealed that there are no changes after 30 seconds, all results are therefore sampled after 30 seconds of simulation.

Chapter 6

Results

6.1 Potential Flow Around a Circular Cylinder

For a potential flow around a circular cylinder, there exists an analytical solution. Hence, the numerical results can be compared with the analytical solution. The implementation reviewed in section 4.3 resulted in a convergence of 3.47 for the $L_2 - error$. Which means that the $L_2 - error$ for the velocity potential converge at a rate $L_2 \sim (D/\Delta x)^{-3.47}$. The results can be found in figure 6.1.

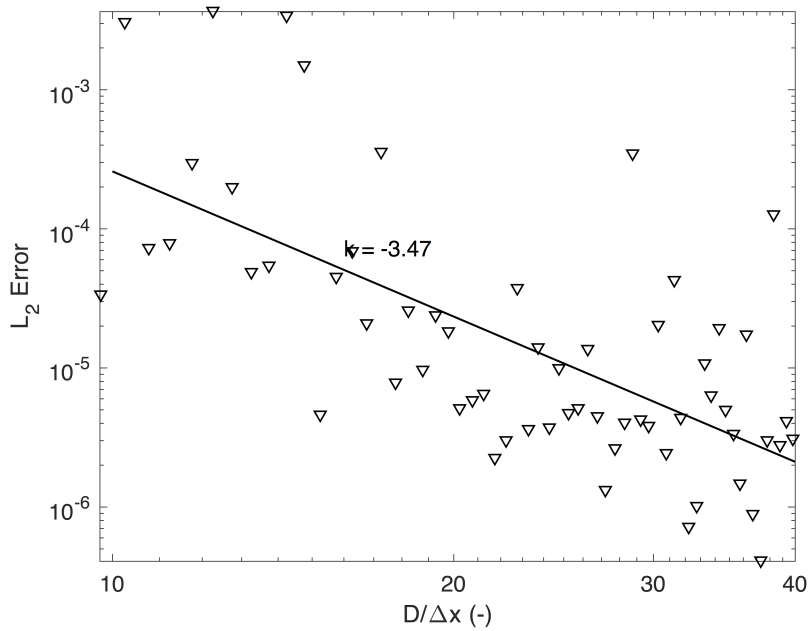

Figure 6.1: Convergence rate for a potential flow around a circular cylinder

Because of the oscillatory behaviour, it is vital to have many data points, to estimate a convergence rate with some confidence. The code only accepts even number of grid points in x- and y-direction. In figure 6.1, is the number of grid points equal in x- and y-direction, and all possible grid sizes between $D/\Delta x = 10$ and $D/\Delta x = 40$ are included.

Although a good convergence rate is obtained, are there some oscillations in the results. Some of the grids have a significant worse accuracy than the rest.

6.2 Viscous Flow Around a Square Cylinder

6.2.1 Flow Features

The first result that is investigated is the streamlines around the square cylinder. This is an intuitive way to evaluate if the solver gives realistic results. The streamlines in figure 6.2 illustrates that the flow has a stable and symmetric wake. The recirculation length is approximately 2.5*D.

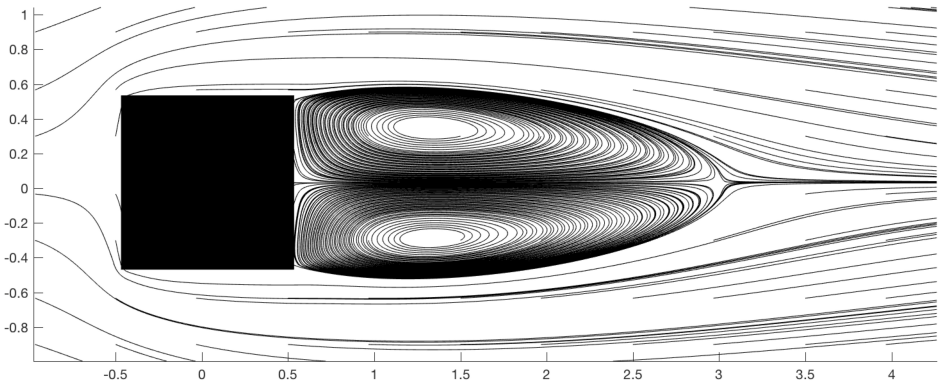

Figure 6.2: Streamlines Around a Square Cylinder, Re=40, Δx=0.066 m

The pressure field that is obtained from the computations can also give useful information. Contour plots of the pressure obtained with the generalized HPC method and the FDM scheme are presented in figure 6.3 and 6.4. The contour plots are taken after 30 seconds of simulations.

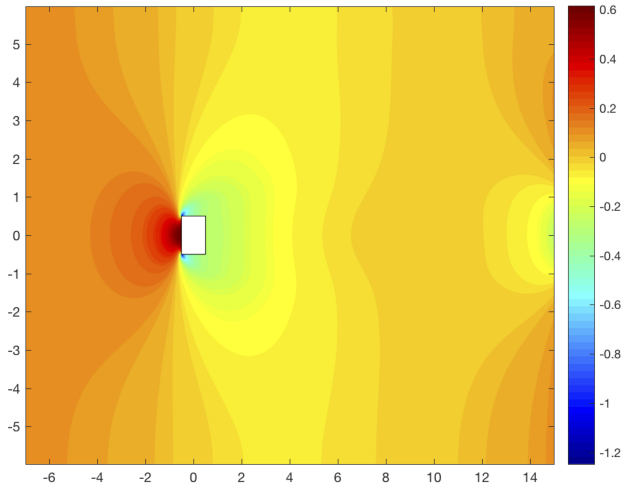

Figure 6.3: Pressure Field Obtained with generalized HPC Mehtod, Re=40, Δx =0.033 m

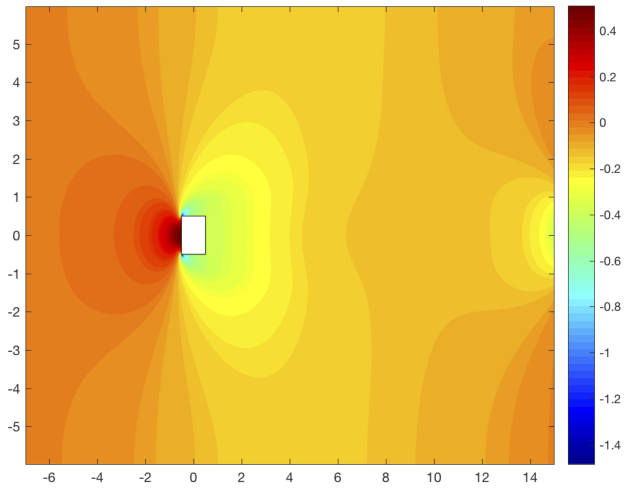

Figure 6.4: Pressure Field Obtained with FDM, Re=40, Δx =0.033 m

The pressure fields are physically reasonable, and the two contour plots have a similar pressure field. However, the value of the pressure differs some between the two methods.

6.2.2 Drag

The drag coefficient was calculated for different grid sizes for both pressure solvers. Drag coefficients calculated with the generalized HPC method and with the Finite Difference method are presented in table 6.1, the deviation between the two methods is also presented in the same table.

Table 6.1: Drag Coefficients

Δx [m]	$C_{D,HPC}$ [-]	$C_{D,FDM}$ [-]	Deviation
0.1333	1.5079	1.7861	0.2782
0.1000	1.6390	1.7660	0.1270
0.0666	1.6430	1.7671	0.1241
0.0444	1.6465	1.7663	0.1198
0.0381	1.6893	1.7704	0.0811
0.0333	1.6935	1.7780	0.0845

For these grids, the generalized HPC method gives lower drag coefficients than the FDM scheme. However, the trend is that the difference between the two methods decreases when the grid size is reduced. The same coefficients can be found in figure 6.5, where the number of nodes per diameter is defined on the x-axis.

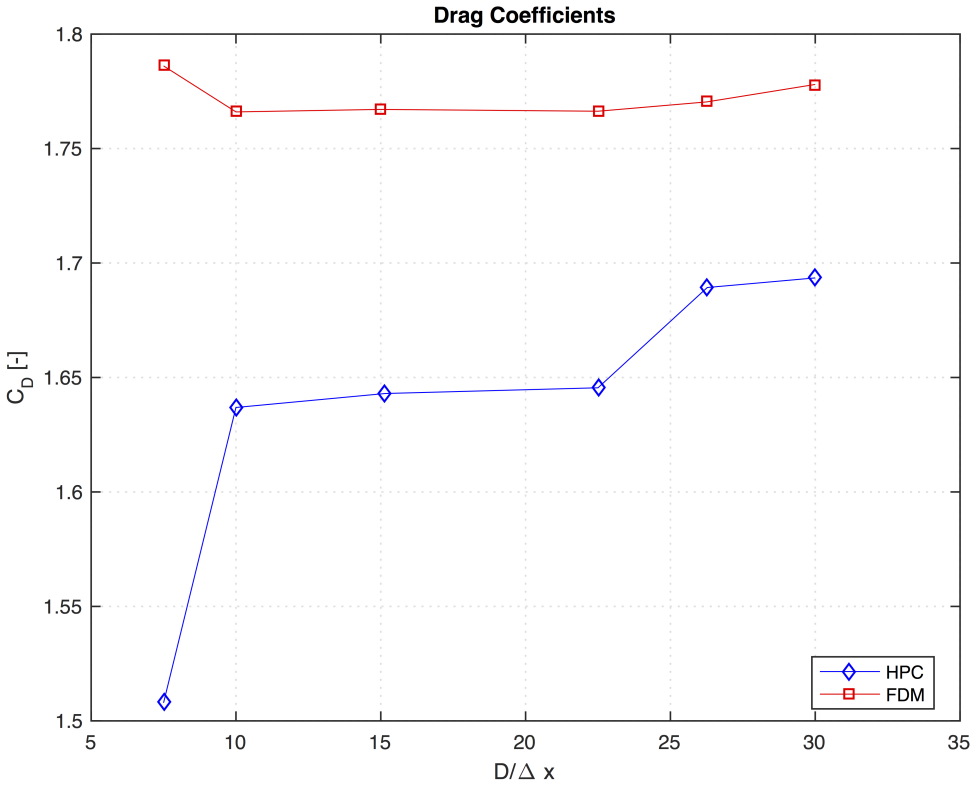

Figure 6.5: Drag coefficients for different grid sizes

The trend is visualised in figure 6.5, that the difference between the two methods decrease when the grid size is reduced. From figure 6.5 does it not seem like any of the methods are converged. Both methods have a significant increase of the drag coefficient, for the two finest grids. Especially the HPC method has a relatively large increase from the third finest to the second finest grid.

Further investigation revealed that the drag from shear actually was slightly larger for the third finest grid than for the second finest. This means that the relatively large gap is caused by the pressure. To investigate this in more detail is the pressure at the upstream side and the downstream side of the cylinder plotted in figure 6.6 and 6.7 respectively. The blue plot represents the third finest grid, and the green plot represents the second finest grid.

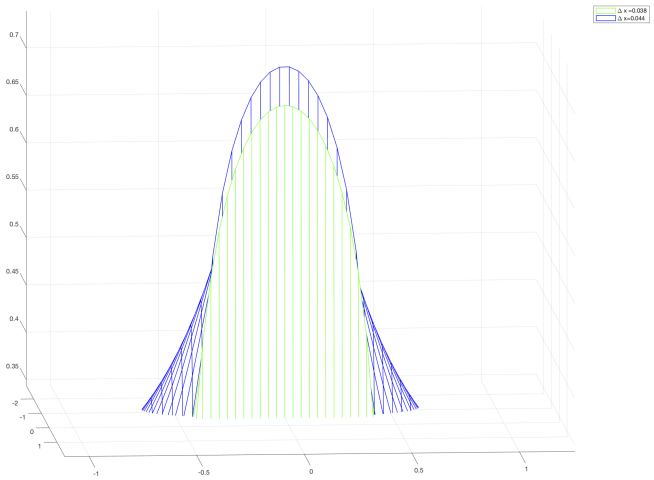

Figure 6.6: Pressure at the Upstream side of the Square Cylinder

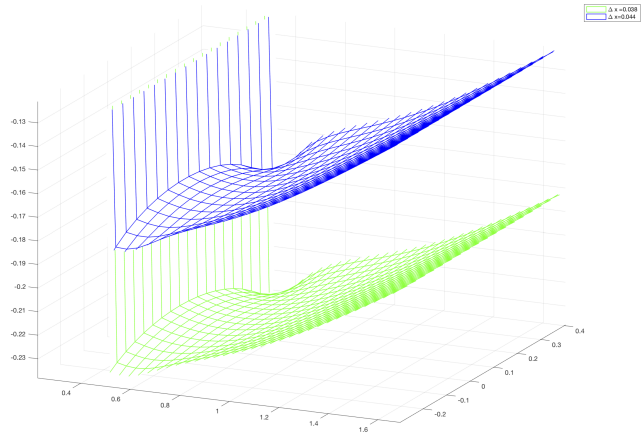

Figure 6.7: Pressure at the Downstream side of the Square Cylinder

On the upstream side of the cylinder is the maximum pressure actual slightly higher for the third finest grid than for the second finest. However, since the pressure on the downstream side of the cylinder is negative, will the pressure here have a suction effect on the cylinder. From figure 6.7 it is clear that the suction is higher for the second finest grid than for the third finest grid. Hence, the increased drag coefficient is caused by a lower pressure at the downstream side of the cylinder.

6.2.3 Computation Time

The two solvers were run on the same machine, and the computation time to simulate 30 seconds was measured for both solvers. The time step and all other parameters were equal.

The results can be found in figure 6.8. The brown dashed line is proportional with $N^{3/2}$ and the blue dashed line is proportional with N^2. These lines are included to illustrate how the computational time increase for the two solvers. The computational time for generalized HPC method without the singular value decomposition is also included. This is to illustrate how much it costs to solve the modified equation (5.6) compared to the original equation system.

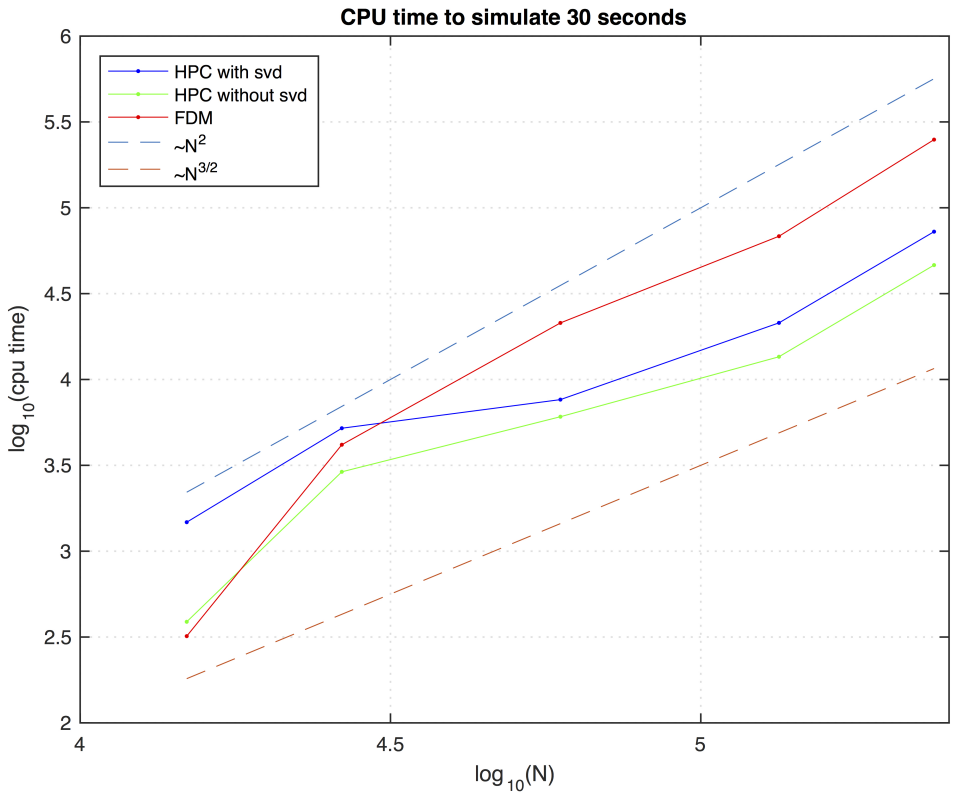

Figure 6.8: Comparison of CPU time to simulate 30 seconds

For the coarsest grids where the number of unknowns is relatively small, is the FDM scheme faster than the HPC method. When the number of unknown increase, the HPC solver is significant faster than the FDM scheme.

The computation time for HPC with and without svd behave in the same manner, with a different grid size dependence when the number of unknowns changes. Between the coarsest and the second coarsest grid do the computational time increase proportionally to N^2. Then, between the second coarsest grid to the second finest grid do the computational time increase slower than $N^{3/2}$. For the finest grids seems it like the computational time increase slightly less than N^2. The difference in grid dependence could be because of the self adapting-solver that is used.

The FDM solver has also a different grid size dependence for the coarse grids than for the fine grids. However, between the second coarsest to the finest grid is the grid dependence approximately proportional to N^2.

It is important to emphasise that there are other factors than the Poisson solver that affect the simulation time. Stability criterions reduce the time step when the number of unknowns increases, for example.

There is an extra cost of solving the modified equation system. For the finest grid is the additional computation time 26 232 seconds to simulate 30 seconds. This corresponds to approximately 7 hours.

However, the FDM scheme uses 176 830 seconds more for the same simulation, than the HPC with svd. This corresponds to approximately 49 hours, and it is over 3.4 times slower than the implemented HPC method. The difference in computational time is remarkable.

Chapter 7

Discussion

7.1 Potential Flow Around a Circular Cylinder

The simulations of a potential flow around a circular cylinder resulted in a convergence rate of 3.47 for the $L_2 - error$. This is within the theoretical range, considering that all polynomials, except one, up to fourth order are included.

The $L_2 - error$ converges with an oscillating behaviour, which can be expected for an immersed boundary method. However, some discretizations that stand out with a significant worse accuracy than the rest of the discretizations. A possible reason could be that the markers and ghost nodes not are distributed ideally for these discretizations. This emphasises the importance of choosing grid with care when an immersed boundary method is used.

For a similar problem Hanssen et al. (2015) obtained a convergence rate of 3.96, but the values of the $L_2 - error$ were worse than obtained here. Ma et al. (2016) have also implemented the IBG for a fixed circular cylinder in a uniform flow. They obtained a convergence rate of 3.44, but the values of the ${}_L2 - error$ oscillates slightly less than obtained in the implementation.

The convergence obtained with the improved code seems to be more robust than the results from the project thesis (Rabliås, 2016). Although a good convergence rate was obtained in the project thesis, it turned out that the convergence rate was reduced when more points were included. In figure 6.1 are all possible grid sizes included, for $D/\Delta x \in [10, 40]$

7.2 Viscous Flow Around a Square Cylinder

The uniform flow around a square cylinder is not as extensively studied as the flow around a circular cylinder, especially for low Reynolds numbers. However, the results are compared with available references.

Dhiman et al. (2006) have obtained a drag coefficient of 1.77. Zanoun and Nosier (2013) have obtained a drag coefficient between 1.6 and 1.7. Breuer et al. (1999) have obtained a coefficient around 1.7 for a confined flow, with a blockage ratio (B=D/H) of 1/8.

Tanaka et al. (1982) have performed experimental studies of oscillating flows around a square cylinder. For large KC-numbers(KC=40 and KC=70), the drag coefficient is around 1.7.

In computational fluid dynamics, it is not uncommon that the same physical problem can give different results, when different numerical schemes are applied. Such differences can be ascribed different grid size, different domain size, or different accuracy of the method that is used. However, it is important to compare the results with reference values to check if the discrepancies are within a reasonable range.

For the finest grid, the generalized HPC method produced a drag coefficient of 1.6935, while the FDM scheme produced a drag coefficient of 1.7780 The difference is below 5%. These results are within the range that is physically reasonable, compared to numerical and experimental results from the literature. Since the reference values deviate in the same order as the HPC and the FDM scheme, it is hard to conclude about which of the methods that are closest to the exact solution.

None of the methods have converged. One reason for this could be that the resolution is not sufficient to obtain convergence. To get accurate results, it is important to have good resolution close to the body. If the grid is too coarse here, the boundary layer will not be properly represented.

A rough estimate of the boundary layer thickness for a flat plate with length 1 could be $\mathcal{O}(\frac{1}{\sqrt{Re}})$ (White, 1974). The flow features and boundary layer is of course not the same for a flat plate and a square, but this can be used as a rough estimate.

For a Reynolds number of 40, can the boundary layer thickness then be approximated as $\mathcal{O}(0.15)$. Considering that the finest grid in the simulations has a grid size of 0.0333, it is not certain that this grid size is fine enough to properly represent the boundary layer.

It is common practice to refine the grid close to the body, to get a good resolution in this important area. Sharma and Eswaran (2004) have performed a sensitivity study of the grid size close to a square cylinder. They tested six grids with Δx between 0.0416 and 0.0083. Based on this sensitivity study, used Dhiman et al. (2006) a grid size of 0.01, close to the cylinder, in their computations. Breuer et al. (1999) used three different grids in their FVM computations. One equidistant grid with $\Delta x = 0.1$, and two non-equidistant grids with $\Delta x = 0.0125$ and $\Delta x = 0.01$ close to the cylinder. The two non-equidistant grids had a good agreement, while the results from the equidistant grid deviated from the other two.

Ideally, should finer grids also be simulated, to investigate convergence properties more properly. With the current algorithm and the available hardware, that is not possible. Grid sensitivity studies from the literature use grid sizes close to the cylinder that is much finer than the finest grid that is used in the simulations. This could be a possible explanation of the lack of convergence.

Regardless of convergence, the drag coefficients for the different grids should be investigated. The coarsest grid gives a drag coefficient that deviates significantly from the other grids. This is not surprising since this is a very coarse grid that clearly not give a proper representation of the physics.

The next large jump in the drag coefficient is between the third finest and the second finest grid. Particularly for the HPC solver is the difference significant between these two grids. Further investigation revealed that a increased pressure drag caused this. A pressure field with lower value at the downstream side of the cylinder is the main reason for the increased drag. The reason for this is not completely understood.

The computation time of solving the Navier-Stokes equations is the limiting factor for many applications. The generalized HPC method has proven to be very efficient for boundary value problems tested by Bardazzi et al. (2015). Considering that the most time-consuming part of a Navier-Stokes solver is the pressure calculation (Armfield and Street, 2002), should an effective Poisson solver reduce the computational time for a fractional step method.

Since the two Poisson solvers are written in different languages, can not the computation time be compared directly to ascertain which solver that is fastest. Moreover, the most interesting parameter is the efficiency, the time needed to obtain a certain accuracy. That being said, the computation time for the two solvers could be an indicator of the efficiency.

For most problems, a Fortran code outperform a code written in MATLAB considering computational time. However, MATLAB has some efficient features that can compete with other programming languages. *mldivide* is such an example, it analyses the equation system and finds the most efficient solver. When the number of unknowns changes, the best available solver could also change. While the FDM scheme is solved with the same SPARSKIT solver for all grid sizes, *mldivide* always use the most efficient solver.

Except for very coarse grids, are the simulations faster when the generalized HPC method is applied for the pressure Poisson equation, even when the singular value decomposition is performed. This adds an extra computational cost that not is present when the FDM scheme is used. Although the svd adds an extra cost, is the generalized HPC method 49 hours faster than the FDM scheme, to simulate 30 seconds with the finest grid. This means that the generalized HPC method which is implemented is approximately 3.4 times faster than the FDM scheme with SPARSKIT, when the number of unknowns is 237 600. The difference is enormous.

It is a bit surprising that the generalized HPC method is so much faster than the FDM scheme, even when the svd is included. No measures were applied to make the svd more efficient than the built-in MATLAB function *svds*. The corresponding matrix multiplication is also possible to perform more efficient. In the current implementation is the matrix multiplication performed in a loop through all nodes. Admittedly was this loop paralleled to 16 CPUs. This is, however, not 16 times faster than computing the loop on 1 CPU. Tests were performed to compute the same loop in Fortran, and this routine was significant faster than the parallel loop in MATLAB. This clarifies that it is possible to make the HPC solver significant faster.

However, it is important to emphasise that it is probably possible to improve the efficiency of the FDM scheme as well.

When a Dirichlet boundary condition was applied to one node, to solve the singular equation system, was a disturbance in the pressure field observed. The reason for this was not fully understood. Because the deadline was close at this stage, the svd approach was implemented without further investigations.

However, this is an interesting subject that deserves more attention. It may look like the trouble are caused because the discrete compatibility condition is not satisfied. Some methods satisfy the compatibility condition automatic. That is, however, not the case for all numerical methods (Henshaw, 1994). An alternative to solve the modified equation system, could be to discretize the Navier-Stokes solver such that the comparability condition is satisfied automatic for the generalized HPC method. This is clearly the situation when the FDM scheme is used to solve the pressure Poisson equation since the original equation system is solved, without experiencing the same problems as the generalized HPC method.

It is not certain that the modified equation is the best way to treat the problem. The svd approach induce an extra computational cost that probably can be avoided. Unfortunately, because of lack of time, this was not further investigated.

The importance of proper discretization was also stressed in an early phase of the implementation when a ”blind coupling” between the Navier-Stokes solver and the Poisson solver was tested. This resulted in a velocity field that did not satisfy continuity,

Chapter 8

Conclusion and Further Work

8.1 Conclusion

An Immersed Boundary Grid for the generalized HPC method is successfully implemented for a circular cylinder. A uniform flow around a fixed circular cylinder is investigated with potential flow theory. The governing equation for this problem is the Laplace equation, which corresponds to the Poisson equation with the right-hand-side set to zero.

The numerical results were compared with the analytical solution. A convergence rate of 3.47 was obtained for the $L_2 - error$. This is within the theoretical range of the method, and it is similar to the results obtained by others.

The $L_2 - error$ converged with an oscillating behaviour. This is an issue for immersed boundary methods in general, and it is also observed by Ma et al. (2016). This emphasises the importance of choosing grids with care when such method is applied. However, the immersed boundary grid is easy to implement for irregular geometries, and it has shown promising results regarding accuracy and convergence.

The uniform inviscid flow around a circular cylinder is a Laplace problem. Hence, the generalized HPC method simplifies to the HPC method for this problem. However the same immersed boundary grid is applied to enforce the boundary conditions at external boundaries and for the body boundary conditions at a square cylinder, when the pressure Poisson equation is solved. This approach satisfied the boundary conditions, hence the approach is also suitable when the right-hand side of the Poisson equation is non-zero.

The generalized HPC method has been implemented in a Navier-Sokes solver to solve the pressure Poisson equation. A viscous uniform flow around a square cylinder is investigated. The results are compared with the results obtained from the same Navier-Stokes solver, when a FDM scheme is used for the pressure Poisson equation.

The same flow features and pressure field are observed with the two methods. However, the value of the pressure differs between the two methods. This results in a slightly different drag coefficient. The drag coefficient obtained with the generalized HPC method is 4.75% lower than the drag coefficient obtained with the FDM scheme.

The drag coefficient is also compared with results from the literature. These reference values are not consistent, and a difference in the order of 7-8% is common. The results obtained with both the generalized HPC method and the FDM scheme are within the range that is observed in the literature. However, because of the deviations in the reference results it is difficult to conclude about which method that is most accurate.

No convergence was observed, neither for the generalized HPC method or the FDM scheme. The reason for this could be that the finest grid is too coarse to obtained convergence, especially close to the body it is important with good resolution.

The uniform grid that is used in the simulations, makes it out of reach to perform further grid refinement. Grid sensitivity studies that are carried out in the literature refine the grid close to the cylinder, and these grids are much finer than the finest grid applied in this report.

Initially a Dirichlet condition was enforced at one node, to solve the singular equation system. This resulted in a pressure field with a disturbance close to the Dirichlet node. Therefore was the system solved with pure Neumann conditions. This produced a pressure field without disturbance.

In order to satisfy the discrete compatibility condition, was a modified equation system solved. A singular value decomposition of the coefficient matrix was necessary to obtain the modified equation. Because of lack of time, was this matter not investigated in more detail, and the approach was implemented without any further modifications.

The modified equation that is implemented adds an extra computational cost to the program. There are two main approaches to reduce this extra cost. Perform the svd and the corresponding matrix multiplication in a more efficient manner, or discretize the Navier-Stokes solver such that the compatibility condition is automatic satisfied. The former approach will be easiest to implement in the existing code. However, it is favourable if the compatibility condition can be automatic satisfied.

The time to simulate 30 seconds was measured, and compared with the FDM scheme already implemented. For the coarsest grids is the FDM scheme fastest, while the generalized HPC method is significant faster when the grid is refined. For the finest grid was the generalized HPC method approximately 3.4 times faster than the FDM scheme. The difference corresponds to 49 hours for 30 seconds of simulation, on the machine that was used.

The HPC Poisson solver is written in MATLAB, while the FDM scheme and the rest of the Navier-Stokes solver are written in Fortran. Hence, the simulation time that is measured can not be used directly to conclude about which method that is fastest. In general, is a program written in Fortran faster than a program written in MATLAB. However, some MATLAB functions are very efficient and can compete with other programming languages concerning speed. For this application it is not sure that the difference between MATLAB and Fortran is as large as expected, considering computation time.

The singular value decomposition that is performed increase the computational time for the generalized HPC method. For the finest grid is the extra computational time approximate 7 hours, to simulate 30 seconds. Despite this additional cost, when the number of unknowns is above approximately 30 000 is the Navier-Stokes solver that use the generalized HPC method for the pressure equation significant faster than when the FDM scheme is used. Since most Navier-stokes solvers for practical applications have more than 30 000 unknowns, is this an indicator that the generalized HPC method is significant faster than a FDM scheme solved with SPARSKIT. This is emphasised by that the generalized HPC method that is implemented have a great potential of improving the efficiency.

The generalized HPC method is successfully implemented to solve the pressure Poisson equation in a Navier-Stokes solver. A uniform flow around a square cylinder is investigated. The obtained drag coefficient is within the range that could be expected, considering reference values from the literature.

The same flow problem is also investigated when a FDM scheme is used for the pressure equation. The drag coefficient deviated from the value that is obtained with the generalized HPC method. However, the deviation is in the same order that the reference values deviates. Considering computational time, is the generalized HPC method significant faster than the FDM scheme. This is an important finding since computational cost is the major issue for Navier-Stokes solvers.

Some issues were discovered during the implementation. All of them are not reviewed in detail in this report. However, suggestion for further work can be found in the next section.

8.2 Further Work

The Immersed Boundary Grid is successfully implemented for a circular cylinder to solve a Laplace problem. A good convergence rate is obtained for this problem. For a Poisson problem is the IBG only implemented for a square cylinder. This is a much simpler geometry that eliminates some of the issues that are present for a body with irregular boundaries.

A suggestion for further work is to implement the IBG for a Poisson problem, on a body with irregular boundaries, e.g a circular cylinder, to verify that a good convergence rate is obtained, also when the forcing term is non-zero.

The computational time limited further grid refinement when the viscous problem was solved. A simple improvement could be to modify the code such that a refined grid is applied close to the body. Then could more accurate results be obtained for the same computational cost.

In order to satisfy the discrete compatibility condition, a modified equation system is solved instead of the original equation system. This includes a singular value decomposition and an extra matrix multiplication. Both these operations can be time-consuming. A suggestion for further work is to find algorithms that perform these operations more efficient. This can significantly improve the efficiency of the method.

An alternative to the modified equation approach is to discretize the Navier-Stokes solver such that the discretized compatibility condition is automatically satisfied for the generalized HPC method. This is favourable compared to the modified equation approach since the computational costs are reduced. To exploit the advantages of the higher order properties of the generalized HPC method, can this be done in the process of developing a higher order method for all steps in a fractional step method.

References

S. Armfield and R. Street. An analysis and comparison of the time accuracy of fractional-step methods for the navier-stokes equations on staggered grids. *International Journal for Numerical Methods in Fluids*, 38(3):255–282, 2002.

A. Bardazzi, C. Lugni, M. Antuono, G. Graziani, and O. M. Faltinsen. Generalized hpc method for the poisson equation. *Journal of Computational Physics*, 299:630–648, 2015.

E. Boujo and F. Gallaire. Controlled reattachment in separated flows: a variational approach to recirculation length reduction. *Journal of Fluid Mechanics*, 742: 618–635, 2014.

M. Breuer, J. Bernsdorf, T. Zeiser, and F. Durst. Accurate computations of the laminar flow past a square cylinder based on two different methods: lattice-boltzmann and fnite-volume. *International Journal of Heat and Fluid Flow*, 21:186–196, 1999.

Yunus A. Cengel and John M. Cimbala. *Fluid Mechanics Fundementals and Applications*. McGraw-Hill, New York, second edition, 2010.

Alexandre Joel Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, 1967.

Giuseppina Colocchio. *Violent disturbence and fragmentation of free surfaces*. phd, University of Southampton, 2004.

A. K. Dhiman, R. P. Chhabra, and V. Eswaran. Steady flow of power-law fluids across a square cylinder. *Chemical Engineering Research and Design*, 84(4): 300–310, 2006.

J. A. Escobar-Vargas, P. J. Diamessis, and T. Sakai. A spectral quadrilateral multidomain penalty method model for high reynolds number incompressible stratified flows. *International Journal for Numerical Methods in Fluids*, 75(6):403–425, 2014.

Jorge Escobar-vargas. A spectral multidomain penalty method solver for environmental flow processes. 2012.

Joel H. Ferziger and M. Perić. *Computational methods for fluid dynamics*. Springer, Berlin, 1996.

Gene H. Golub and Charles F. Van Loan. Matrix computations. 1996. *Johns Hopkins University, Press, Baltimore, MD, USA*, pages 374–426, 1996.

Joseph J. Gorski. Present state of numerical ship hydrodynamics and validation experiments. *Journal of Offshore Mechanics and Arctic Engineering*, 124(2): 74–80, 2002.

Philip M. Gresho and Robert L. Sani. On pressure boundary conditions for the incompressible navier-stokes equations. *International Journal for Numerical Methods in Fluids*, 7(10):1111–1145, 1987.

Hanssen, Greco, and Shao. The harmonic polynomial cell method for moving bodies immersed in a cartesian background grid. *34th International Conference on Ocean, Offshore and Arctic Engineering (OMAE2015)*, 2015.

Francis H. Harlow and J. Eddie Welch. Numerical calculation of time dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12): 2182–2189, 1965.

Rene Heinzl. Concepts for scientific computing, 2007. URL `http://www.iue.tuwien.ac.at/phd/heinzl/node27.html`. Accessed: November 1st, 2016.

William D. Henshaw. A fourth-order accurate method for the incompressible navier-stokes equations on overlapping grids. *Journal of Computational Physics*, 113(1):13–25, 1994.

I. S. Kharlamova, A. Kharlamov, and P Vlasák. Salatation of sand in vicinity of cylindrical column. In *Engineering Mechanics*, Svratka, Czech Republic, 2013.

Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, INC, 10 edition, 2011.

H. P. Langtangen, K. A. Mardal, and R. Winther. Numerical methods for incompressible viscous flow. *Advances in Water Resources*, 25(8-12):1125–1146, 2002.

S. Ma, F-C.W. Hanssen, M.A. Siddiqui, M. Greco, and O.M. Faltinsen. Local and global properties of the harmonic polynomial cell method: In-depth analysis in two dimension. *Submitted for journal publication*, 2016.

math3510edensmith. Consolidation week, 2014. URL `https://math3510edensmith.wordpress.com/author/math3510edensmith/`. Accessed: Novemer 14th, 2016.

Atsushi Okajima. Strouhal numbers of rectangular cylinders. *Journal of Fluid Mechanics*, 123(Oct):379–398, 1982.

S. V. Patankar and D. B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15(10):1787–1806, 1972.

Bjørnar Pettersen. *Kompendium - Marin Teknikk 3 Hydrodynamikk*. Department of Marine technology, NTNU, Marine Technology Centre Trondheim, 2007.

C. Pozrikidis. A note on the regularization of the discrete poisson–neumann problem. *Journal of Computational Physics*, 172(2):917–923, 2001.

Øyvind Rabliås. *HPC and generalized HPC method for marine applications*. Project thesis at the department of marine technology, Norwegian University of Science and Technology, 2016.

Yan-Lin Shao and Odd M Faltinsen. Towards efficient fully-nonlinear potential-flow solvers in marine hydrodynamics. In *ASME 2012 31st International Conference on Ocean, Offshore and Arctic Engineering*, pages 369–380. American Society of Mechanical Engineers, 2012.

Yan-Lin Shao and Odd M Faltinsen. Fully-nonlinear wave-current-body interaction analysis by a harmonic polynomial cell method. *Journal of Offshore Mechanics and Arctic Engineering*, 136(3):031301, 2014a.

Yan-Lin Shao and Odd M. Faltinsen. A harmonic polynomial cell (hpc) method for 3d laplace equation with application in marine hydrodynamics. *Journal of Computational Physics*, 274:312–332, 2014b.

Atul Sharma and V. Eswaran. Heat and fluid flow across a square cylinder in the two-dimensional laminar flow regime. *Numerical Heat Transfer, Part A: Applications*, 45(3):247–269, 2004.

Bengt Sunden. Tubes, crossflow over, 16 March 2011 2011. URL `http://www.thermopedia.com/content/1216/`. Accessed: November 15th, 2016.

N. Tanaka, Y. Ikeda, and K. Nishino. Hydrodynamic viscous force acting on oscillating cylinders with various shapes. *Proceedings of 6th Symposium on Marine Technology, Society of Naval Architecture of Japan*, 1982.

John C. Tannehill, Dale A. Anderson, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. Taylor & Francis, second edition, 1997.

H. S. Udaykumar, R. Mittal, P. Rampunggoon, and A. Khanna. A sharp interface cartesian grid method for simulating flows with complex moving boundaries. *Journal of Computational Physics*, 174(1):345–380, 2001.

H. K. Versteeg and & W. Malalasekera. *An Introduction to Computational Fluid Dynamics*. Longman Scientific & Technical, 1995.

Frank M. White. *Viscous Fluid Flow*. McGraw-Hill, 3 edition, 1974.

El-Sayed Zanoun and Mohamed Nosier. Low reynolds number flow structure past circular, square and triangular cylinders: A comparative numerical study. *8th International Conference on Multiphase Flow ICMF 2013, Jeju, Korea, May 26 - 31*, 2013.