NTNU
Norwegian University of
Science and Technology

Numerical Simulations of Swimming Fish

Jon Coll Mossige

Marine Technology
Submission date: March 2017
Supervisor: Dag Myrhaug, IMT
Co-supervisor: Lars Erik Holmedal, IMT
Hong Wang, IMT

Norwegian University of Science and Technology
Department of Marine Technology

NTNU – Trondheim
Norwegian University of
Science and Technology

Numerical Simulations of Swimming Fish

Jon Coll Mossige

February 2017

MASTER'S THESIS

Department of Marine Technology

Norwegian University of Science and Technology

Supervisor : Professor Dag Myrhaug

Co-supervisor : Professor Lars Erik Holmedal

Co-supervisor : Postdoctoral fellow Hong Wang

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology – Group of Marine Structures

MASTER THESIS IN MARINE TECHNOLOGY

AUTUMN 2016

FOR

STUD. TECHN. JON COLL MOSSIGE

NUMERICAL SIMULATIONS OF SWIMMING FISH

Unsteady flows associated with moving boundaries have been of continuously growing interest to Computational Fluid Dynamics (CFD) community. This thesis will focus on the hydrodynamic effects of a swimming fish and fluid interaction in an incompressible viscous fluid. The work provides a foundation for extending the application of CFD tools and solving more complex hydrodynamic problems.

The student shall:

1. Give a background of CFD related to fluid-structure-interaction (FSI)
2. Describe the computational tools
3. Develop a numerical code including an Immersed Boundary Method (IBM) for simulations of fluid-structure-problem
4. Perform numerical simulations of the structure (cylinder)
5. Present and discuss the results including discussing further development of the code

The work scope may prove to be larger than initially anticipated. Subject to approval from the supervisor, topics may be deleted from the list above or reduced in extent.

In the thesis the candidate shall present his personal contribution to the resolution of problem within the scope of the thesis work.

Theories and conclusions should be based on mathematical derivations and/or logic reasoning identifying the various steps in the deduction.

The candidate should utilize the existing possibilities for obtaining relevant literature.

The thesis should be organized in a rational manner to give a clear exposition of results, assessments, and conclusions. The text should be brief and to the point, with a clear language. Telegraphic language should be avoided.

The thesis shall contain the following elements: A text defining the scope, preface, list of contents, summary, main body of thesis, conclusions with recommendations for further work, list of symbols and acronyms, reference and (optional) appendices. All figures, tables and equations shall be numerated.

The supervisor may require that the candidate, in an early stage of the work, present a written plan for the completion of the work. The plan should include a budget for the use of computer and laboratory resources that will be charged to the department. Overruns shall be reported to the supervisor.

The original contribution of the candidate and material taken from other sources shall be clearly defined. Work from other sources shall be properly referenced using an acknowledged referencing system.

Advisors: Dr. Hong Wang
Professor Lars Erik Holmedal
Professor Dag Myrhaug

Deadline: 05.02.2017

Dag Myrhaug
Supervisor

Preface

This master's thesis constitutes my final effort in the 5 year study program "Marine Technology" at the Norwegian University of Science and Technology (NTNU). The work took place during the autumn semester of 2016, at the department of Marine Technology at Tyholt, Trondheim. It is assumed that the reader has an academic degree within applied mathematics, technology or similar and that he/she is able to obtain sufficient background knowledge to be able to understand this thesis.

For the last few years I have grown curious about the field of *biomimetics*, realizing that there is still much to learn from nature when it comes to engineering problems. When I was to choose the topic for my thesis, this urge for studying the nature was the strongest driving force: I ended up on choosing the topic *Numerical Simulations of Swimming Fish*, even though my experiences within numerics and computational science were limited. Despite some trouble along the road, the field of computational numerics has given me a taste for more.

I wish to show my gratitude towards my supervisors for all the help and support I have received. To my main supervisor, Professor Dag Myrhaug, for informative discussions about academic writing, as well as high quality education throughout my years here at Tyholt. To my co-supervisors, Professor Lars Erik Holmedal and Postdoctoral fellow Hong Wang, for sharing their knowledge within fluid dynamics, numerics and computational science, and for hosting me in their office for long and fruitful conversations.

More thanks goes to "ABO PLAN & ARKITEKTUR AS" for lending me a desk when I was visiting my hometown Bergen.

Thank you, Mum, Dad, Sunva, Eldri and Dag for being such a great family. Thank you, Synne, my best friend and fiancé, for always supporting me, and thank you, Viiru, for (mostly) being a well behaved dog, and for teaching me the *real* value of a stick.

Trondheim, March 1, 2017

Jon Coll Mossige

Sammendrag

En grundig forståelse av fiskens svømmeteknikk kan utnyttes til design av innovative propulsjonsenheter med høyere effektivitet enn konvensjonelle skruepropeller. I søken etter en slik forståelse kan *Computational Fluid Dynamics* (CFD) være til god hjelp.

En svømmende fisk er karakterisert ved materialdeformasjon, fluidkrefter og interaksjon mellom disse. Å formulere denne interaksjonen numerisk er ingen enkel sak, og i konvensjonelle fluid-struktur-løsere må problemets domene diskretiseres på nytt ved hvert tidssteg. Dette kalles re-meshing og er komputasjonelt kostbart. I Immersed-Boundary-metoder, først foreslått av Peskin (1972), behøves ikke re-meshing, og slike metoder er derfor gode kandidater for numerisk løsning av svømmende fisk-problemet.

Denne oppgaven presenterer grunnlegende bakgrunnsteori innenfor CFD, med fokus på immersed boundary-metoder og fluid-struktur-interaksjon. En CFD-løser som drar nytte av en immersed-boundary-metode er utviklet og validert for problemer med stasjonære strukturer av villkårlig form nedsenket i inkompressibel og viskøs 2D-strømning. Koden er også designet for å håndtere strukturer med bevegelse og deformasjoner, men er ennå ikke validert for dette.

Til slutt dekker oppgaven en diskusjon vedrørende videre utvikling av koden til en full fluid-struktur-løser, med et spesielt fokus på applikasjoner for svømmende fisk. Det blir konkludert med at en slik fluid-fisk-løser vil svært utsatt for numerisk instabilitet, men at dette kan håndteres ved bruk av iterativ kalkulering av fluidens krefter på fisken.

Summary

A profound understanding of how a fish swims may be used to design innovative propulsion units with higher efficiency than the conventional screw propellers. The gain of such understanding may be aided by the use of *Computational Fluid Dynamics* (CFD).

The swimming fish problem is characterized by deforming solids, fluid forces and the interaction between these. To capture this interaction numerically is no simple task, and in conventional fluid-structure interaction approaches, the discretized domain must be re-generated at each time step. This is called re-meshing, and it is computationally costly. The immersed boundary methods, first proposed by Peskin (1972), do not pose the need for re-meshing, and are therefore good candidates for solving the swimming fish problem.

This thesis presents a review of basic theory within CFD, with focus on immersed boundary methods and fluid-structure interaction. A CFD-code, including an immersed boundary method, is developed and validated for problems involving stationary objects of arbitrary shape submerged in 2D incompressible viscous flow. The code is also designed to handle structures with motion and deformation, but is yet to be validated for this purpose.

Lastly, the further development of the code into a full 2D fluid-structure interaction solver is discussed with special focus on applications on swimming fish. It is concluded that such a fluid-fish-solver will be severely prone to numerical instabilities, but that this can be handled by using iterative solution strategies for the calculation of the fluid force on the fish.

Contents

Preface **v**

Sammendrag **vi**

Summary **vii**

1 Introduction **1**

1.1 Background 1

1.2 Objectives 4

1.3 Limitations 4

1.4 About the report 5

2 Governing Equations **7**

2.1 Governing fluid equations 7

2.1.1 Non-dimensional form 9

2.1.2 Conservation forms 10

2.2 Governing equations for solid mechanics 11

2.2.1 Rigid body motion 11

2.2.2 Deformation 11

2.2.3 Simplifications: A quasi-rigid approach 12

2.3 Boundary conditions 14

2.3.1 Domain boundary condition 14

2.3.2 Fluid-Structure boundary conditions 15

3 Computational Fluid Dynamics **17**

3.1 General approach 18

3.2 Discretization 19

3.3 Grids 20

3.3.1 Grid quality . . . 21
3.3.2 Structured vs. unstructured grid . . . 22
3.3.3 Staggered vs. co-located grid . . . 23
3.3.4 Adaptive mesh refinement . . . 24
3.4 Application of boundary conditions . . . 26
3.5 Semi-discrete approach . . . 29
3.6 Pressure-Velocity coupling . . . 30
3.6.1 The Fractional Step Method . . . 30
3.7 Solution quality tests . . . 33
3.7.1 Cell Courant-number . . . 33
3.7.2 Discrete continuity equation . . . 33
3.8 Fluid-Structure Interaction . . . 34
3.8.1 Modeling approaches . . . 34
3.8.2 Solution algorithms . . . 35

4 Immersed boundary methods 37
4.1 Different IBM approaches . . . 40
4.1.1 Continuous forcing methods . . . 40
4.1.2 Direct forcing methods . . . 42
4.2 IBM in swimming fish problems . . . 48
4.2.1 Examples of studies . . . 49

5 Numerics 53
5.1 Approximation of spatial derivatives . . . 54
5.2 Time integration . . . 55
5.2.1 Forward Euler method . . . 55
5.2.2 Explicit Runge-Kutta 2 method . . . 56
5.3 Linear solvers . . . 57
5.3.1 Gauss-Seidel method . . . 58
5.4 Numerical Error . . . 59
5.5 Numerical stability . . . 60
5.5.1 Linear stability . . . 60
5.5.2 The Courant number . . . 62

6 Development of code **63**

6.1 Development tools 64

6.2 Notation 64

6.3 The Navier-Stokes solver 67

6.3.1 Grid generation 67

6.3.2 Finite difference scheme 70

6.3.3 Time integration and pressure-velocity coupling 73

6.4 The IBM module 76

6.4.1 Defining the immersed boundary 77

6.4.2 Detection of intersections 78

6.4.3 Finding the immersed boundary velocity 84

6.4.4 Imposing the immersed boundary conditions on the fluid grid . . 84

6.4.5 Force calculation 87

6.5 The combined IBM-N-S solver 89

6.6 Limitations 91

6.6.1 Nodal ambiguity 91

6.6.2 Interpolation on coarse grids 91

7 Validation of fluid solver **95**

7.1 Validation of Poisson solver (Gauss Seidel-iterator) 95

7.2 Validation of the Navier-Stokes solver 100

8 Validation of the immersed boundary solver **103**

8.1 Reference problem 104

8.2 Method and setup 106

8.3 Flow characteristics 108

8.4 Time refinement tests 111

8.5 Mesh refinement tests 112

8.6 Domain size tests 113

8.7 Transients at flow start-up 116

8.8 Force coefficients at $Re = 100$ 121

8.9 Concluding remarks 123

9 Further development of code: Swimming Fish **125**

9.1 Finishing the IBM-module . . . 125
9.1.1 Concluding validation . . . 126
9.2 Development of a structural response solver . . . 128
9.2.1 Simplifications . . . 128
9.2.2 Numerical solution for the rigid body equations . . . 129
9.2.3 Adding the flap and finding the total motion . . . 130
9.3 Development of a swimming fish model . . . 132
9.3.1 Physical requirements . . . 132
9.3.2 Choice of swimming mode . . . 132
9.3.3 Startup function . . . 133
9.3.4 General mathematical model . . . 134
9.3.5 Design of a BCF mode swimming body . . . 135
9.4 Proposal for a fluid-structure-interaction code . . . 139
9.4.1 Anticipated stability issues . . . 139

10 Conclusions **143**

11 Recommendations for further work **145**
11.1 Short term recommendations . . . 145
11.2 Long term recommendations . . . 146

References **147**

A Tensor notation **I**

B Description of motion **V**

Postface **VII**

List of Tables

2.1 Name of the terms in the Navier-Stokes equation. 8
2.2 Non-dimensional quantities in fluid mechanics. 9
2.3 Boundary condition types. 14
2.4 Relation between physical boundary behavior and the related mathematical model. 15

3.1 General notation for discrete difference operators. 20

6.1 Notation used for description of the N-S solver. 65
6.2 Notation used for description of the IBM-module. 66

7.1 Setup of numerical analysis for a lid-driven cavity flow problem. 100

8.1 Reference values for drag coefficients of 2D uniform flow past circular cylinder. 104
8.2 Reference values for lift coefficients of 2D uniform flow past circular cylinder. 105
8.3 Boundary- and initial conditions applied on the domain for the circular cylinder problem. 107
8.4 Flow characteristics test: Domain setup. 109
8.5 Time refinement tests: Domain setup and results. 111
8.6 Mesh refinement tests: Domain setup and results. 112
8.7 Domain size tests: Domain setup and results. 113
8.8 Moving cylinder tests: Domain setup. 118
8.9 Test at $Re = 100$: Domain setup and results. 121

9.1 Tuning of the fish motion model. 135

A.1 Physical quantities and their tensor rank III

A.2 Common vector operators and their equivalent in tensor notation IV

List of Figures

1.1 Old and new technology within flapping foil propulsion. 2

3.1 A typical CFD-result. 17
3.2 Two types of fluid cell definition in a grid. 20
3.3 Structured and unstructured grids. 22
3.4 Co-located and staggered grid definition. 23
3.5 Adaptive mesh refinement around a sphere. 25
3.6 Application of boundary conditions on a staggered grid. 27
3.7 Dirichlet boundary condition on velocities on a staggered grid. 28
3.8 Fluid-structure modeling approaches. 35
3.9 Conforming and non-conforming mesh. 36

4.1 IBM-simulation of blood flowing through a heart valve. 39
4.2 Distribution functions for continuous IBMs. 41
4.3 Direct forcing interpolation schemes. 44
4.4 Interpolation on the non-solenoidal velocity field. 46
4.5 Demonstration of 3D IBM domain. 50
4.6 IBM performed on a mackerel. 51

5.1 Linear stability analysis of RK-methods. 61

6.1 Excerpt from the code: FDM gradient function. 63
6.2 Outline of the N-S solver. 68
6.3 The definition of the computational grid used for the program. 69
6.4 FDM-stencil for the gradient difference operator. 71
6.5 FDM-stencil for the Laplacian difference operator. 72
6.6 FDM-stencil for the advective difference operator. 72
6.7 An RK2 subroutine with the fractional step method implemented. 75

6.8 Notation and terminology for the nodes used in the IBM-module. 77
6.9 Detection of instersections between immersed boundary and underlying grid. 80
6.10 Intersection detection for normal interpolation. 82
6.11 Intersection detection for parallel interpolation. 83
6.12 Parallel interpolation of velocity near the immersed boundary. 85
6.13 Normal interpolation of velocity near the immersed boundary. 86
6.14 Fluid acceleration due to IBM-forcing. 88
6.15 Implementation of IBM into an RK2 subroutine. 90
6.16 Limitations on the immersed boundary due to node ambiguity. 92
6.17 Issues with linear interpolation of the velocity on coarse grids. 93

7.1 Gauss-Seidel iterator: Error analysis. 96
7.2 Analytical vs. numerical solution from Gauss-Seidel solver. 97
7.3 Gauss-Seidel iterator: Efficiency analysis. 99
7.4 Velocity profiles in a lid-driven cavity. 101
7.5 Stream function contours on a lid-driven cavity flow. 102

8.1 Domain setup for circular cylinder problem. 107
8.2 Flow characteristics test: Results. 110
8.3 Effect on the drag coefficient from varying the domain size parameters. . 114
8.4 Effects on the flow of vortex shedding in a too small domain. 115
8.5 Transient effects on force coefficients at startup flow past a stationary circular cylinder. 117
8.6 Vortex formation around a cylinder oscillating in an initially stationary fluid. 119
8.7 Force coefficients calculated for a cylinder oscillating in stationary fluid. 120
8.8 Results from analysis of flow at $Re = 100$. 122

9.1 A more accurate force calculation. 127
9.2 Terminology of the fish body. 133
9.3 Subgroups of body-caudal fin propulsion modes. 134
9.4 A simple 2D fish model. 136
9.5 Deformation by rotating trapezoids. 137

9.6 Two possible motion models for a 2D fish. 138
9.7 Proposal for an FSI-solver. 140

List of Equations

2.1 Dimensional Navier-Stokes equation for incompressible flow 7
2.2 Dimensional Continuity equation for incompressible flow 7
2.3 Non-dimensional Navier-Stokes equation for incompressible flow 9
2.4 Non-dimensional Continuity equation for incompressible flow 9
2.5 Generalized Reynolds number 9
2.7 Conservation form of the Navier-Stokes equation for incompressible flow . . 10
2.11 Governing equation for the quasi-rigid structure 13
3.1 General discretization of the incompressible Navier-Stokes equation 19
3.2 General discretization of the incompressible Continuity equation 19
3.3 The semi-discrete ODE for u_i in time 29
3.4 The semi-discrete BVP for u_i in space 29
3.10 The continuous Poisson equation for the pressure correction δp 31
3.12 The discrete Poisson equation for the pressure correction δp 31
3.13 The cell Courant-number 33
4.1 General formulation for an immersed boundary method 37
4.2 General discrete formulation for an immersed boundary method 37
4.3 Boundary conditions for the immersed boundary 38
4.5 Mohd-Yusof's formulation for the direct immersed boundary force observed at the boundary 42
4.6 Mohd-Yusof's formulation for the direct immersed boundary force observed in the fluid 42
4.10 The direct immersed boundary force obtained from interpolation on $\tilde{u}_i$ . . . 45
5.4 Explicit Euler integration 55
5.6 Explicit two-stage Runge-Kutta integration 56
5.10 Explicit formulation for field variable in the Gauss-Seidel method 58
6.1 2nd order FDM-scheme for the pressure gradient in the x-direction 70

6.2 2nd order FDM-scheme for the pressure gradient in the y-direction 70
6.3 2nd order FDM-scheme for the Laplacian of u 70
6.4 2nd order FDM-scheme for the Laplacian of v 70
6.5 2nd order FDM-scheme for the advection of u 71
6.6 2nd order FDM-scheme for the advection of v 71
6.7 Calculations performed by the FSM-predictor subroutine 74
6.10 Calculations performed by the FSM-corrector subroutine 74
6.11 Definition of the immersed boundary in the code 77
6.18 Detection of intersections between immersed boundary and underlying grid 79
6.20 Calculation of the immersed boundary velocity at an intersection point . . . 84
6.21 Calculation of the fluid velocity in the node closest to the intersection 84
6.25 Calculation of the force contribution at an intersection 87
6.26 Calculation of the total force applied on structure from fluid 88
9.7 Numerical integration of rigid-body response velocity 129
9.8 Numerical integration of rigid-body response displacement 130
9.11 Vertex positions for a flapping fish in a fluid 131
9.12 Vertex velocities for a flapping fish in a fluid 131
9.16 BCF fish motion model (displacement) 134
9.17 BCF fish motion model (velocity) 134

Nomenclature

Abbreviations

2D	2-dimensional
3D	3-dimensional
AMR	Adaptive Mesh Refinement
BC	Boundary Condition
BCF	Body Caudal Fin
BVP	Boundary Value Problem
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrich-Lewy-number
CoM	Center of Mass
DoF	Degree of Freedom
FSI	Fluid-Structure Interaction
FSM	Fractional Step Method
GS	Gauss-Seidel
IBC	Immersed Boundary Condition
IBM	Immersed Boundary Method
IVP	Initial Value Problem
LHS	Left Hand Side (of an equation)

MinGW	Minimalist GNU for Windows
MPF	Median and/or Paired Fin
N-S	Navier-Stokes
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
RHS	Right Hand Side (of an equation)
RK	Runge-Kutta
RK2	Explicit two stage Runge-Kutta method

Symbols

$(\cdot)$	Denotes an arbitrary quantity
$(\cdot)\|_{i,j}$	Arbitrary quantity observed at nodal address i, j
$(\cdot)^k$	Arbitrary quantity observed at point k on the discrete boundary
$(\cdot)^n$	Arbitrary quantity observed at time step number n
$(\cdot)_n$, $(\cdot)_t$	Arbitrary quantity observed in the direction *normal* and *tangential* to a surface, respectively.
$(\hat{\ })$	Denotes that a quantity has dimension
β_i	Tensor switch (= 0 for $i = 1,2,3$ and = 1 for $i = 4,5,6$)
$\circ$	Denotes the integer address of the fluid node next to the interpolation node
δp	Pressure correction over a time step: $p^{n+1} = p^n + \delta p$
Δt	Discrete time step
Δx_i	Mesh cell size in x_i-direction
$\dot{\theta}$	Angular velocity of solid
Γ_b	Immersed boundary

Γ_Ω	Domain boundary
$\lceil(\cdot)\rceil$	*Ceiling*-function - rounds $(\cdot)$ **up** to nearest integer
$\lfloor(\cdot)\rfloor$	*Floor*-function - rounds $(\cdot)$ **down** to nearest integer
$\mathscr{T}(\cdot)$	Discrete time derivative
μ	Dynamic viscosity of fluid
ν	Kinematic viscosity of fluid
$\odot$	Denotes the body-internal node
Ω	Problem domain
$\otimes$	Denotes the integer address of the interpolation node
Φ	Field variable in the general poisson equation
Φ_D	Difference between two successive iterations on Φ
ρ	Fluid density
τ_w	Wall shear stress
$\mathbf{b}$	Array containing information of the polygon constituting the immersed boundary
$\mathbf{b}_\times$	Array carrying information about number of intersections on each polygon segment
$\mathbf{b}_{comp}$	Polygon coordinates of $\mathbf{b}$ mapped to the computational domain
$\mathbf{b}_{dir}$	Array describing the directional vectors of each polygon segment of $\mathbf{b}$
$\mathbf{b}_{norm}$	Array describing the normal vectors of each polygon segment of $\mathbf{b}$
$\tilde{u}_i$	Velocity field that is not required to satisfy continuity
$\times$	Denotes the intersection number of interest
$A(\cdot)$	Discrete advective operator

$A\|_{i,j}$	Numerical scheme for the advective term at nodal address i, j
c	Denotes the vertex or segment number of interest
d	Direction (x: $d = 1$, y: $d = 2$)
$dx(d)$	Mesh cell size in d-direction
F_d	Total force in d-direction from fluid to immersed boundary
f_d	Immersed boundary forcing term in d-direction
F_i	Excitation force in x_i-direction
f_i	Immersd boundary force in the x_i-direction
g	Right hand side of the general poisson equation
$G\|_{i,j}$	Numerical scheme for the pressure gradient term at nodal address i, j
$G_i(\cdot)$	Discrete gradient operator in i direction
h	Time step in numerical integration
i, j	Horizontal and vertical nodal address, respectively
I_f	Rotational inertia of fish about the x_3-axis passing through the Center of Mass
k	Lagrangian point on the discrete boundary
$L(\cdot)$	Discrete Laplace operator
$L_\infty(\cdot)$	The infinity norm applied on $(\cdot)$
m_f	Mass of fish
m_{ij}	Inertia in x_i-direction due to acceleration in x_j-direction
n	Time step number
N_b	Number of vertexes contained in **b**
N_x	Number of discrete points in x-direction

N_y	Number of discrete points in y-direction
$N_{\times_d}$	Number of intersections in d-direction
p	Fluid pressure
$pos\left((\cdot)\vert_{i,j}\right)$	Position of nodal address $\left(i,j\right)$ for quantity $(\cdot)$ in the staggered configuration
Re	Reynolds number
RK_{dt}	Length of th time step for the respective Runge-Kutta call
RK_{step}	Indicating the propagation length of the respective Runge-Kutta integration step
T	Solution end time
t	Time
u, v	Fluid horizontal and vertical velocity, respectively
U_d	Interpolated fluid velocity in d-direction d
u_i	Velocity component of fluid in x_i-direction
$V(c,d)$	Immersed boundary velocity at vertex c in d-direction
$V\vert_{i,j}$	Numerical scheme for the viscous term at nodal address i, j
V_d	Immersed boundary velocity in d-direction at intersection
V_i	Velocity component of solid point in x_i-direction
$X(c,d)$	Position in direction d of vertex c
$x_\times$	x-coordinate of intersection
X_i	Position of solid point in x_i-direction
$y_\times$	y-coordinate of intersection

Chapter 1

Introduction

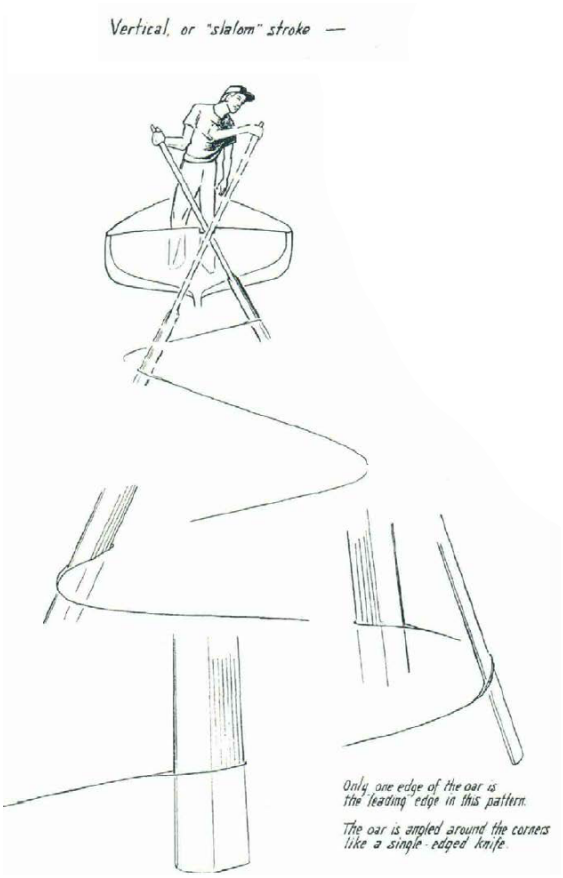
1.1 Background

The study of swimming fish and other aquatic animals may provide valuable knowledge for engineering applications. In naval engineering, both maneuverability and propulsive efficiency is of great concern, and conventional strategies for optimizing such are often far less sophisticated than the ones we can find in the nature. The maximum efficiency of a conventional naval screw propeller is about 0.7 (Carlton (2012)). For comparison, Fish and Rohr (1999) present a selection of cetaceans[1] that have propulsive efficiencies spanning from 0.77 to 0.99.

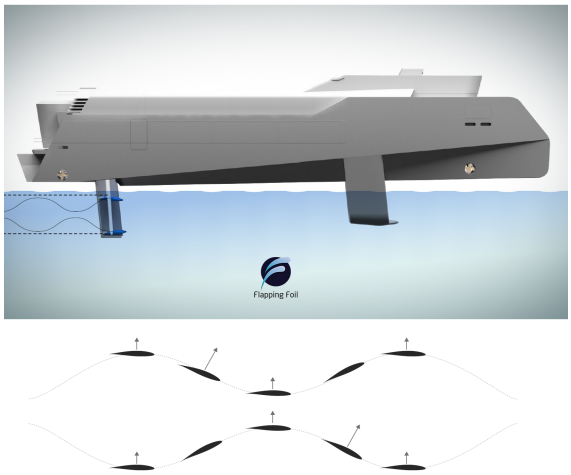
The idea of flapping tail propulsion has been around for ages, but a viable design for ship propulsion such as the *Flapping Foil* concept developed by Ph.D-candidate John Martin Godø at NTNU (see Stenvold (2016)) is yet to be realized for conventional use. Figure 1.1 presents the old technology side by side with the *Flapping Foil*-concept.

In order to utilize aquatic locomotion strategies, extensive research must be done on the area, both experimentally and computationally. An experimental approach is crucial in order to gain knowledge about the *principles* of swimming and *motion strategies*, but the setup of such experiments is time consuming and complicated. The fish must be doing the exact maneuver of interest, and instruments cannot provide information about the whole problem domain. In addition, the very presence of instruments may pollute the observations. With these challenges in mind, it is clear that *Computational*

[1] Cetacea is the scientific classification containing all types of whales



(a) Old technology: Single-oar sculling. (Figure adopted and modified from Manning (1991))



(b) New technology: Flapping foils for propulsion and lift. (Modified, with permission from Godø (2017))

Figure 1.1: Old and new technology within flapping foil propulsion.

Fluid Dynamics (CFD) may become a powerful supplemental tool. In CFD, the observer may design the motion of the specimen as wished, and the results from CFD simulations consist of data from the whole flow domain, and may be stored easily in a computer register for later analysis and comparison. This opens for parametric studies, which can not only provide answers about the *principles* of swimming, but also about how to *optimize propulsion strategies*.

The swimming fish may be classified as a *Fluid-Structure Interaction problem* (FSI), with a coupled response of fish and fluid. A great challenge within CFD is to capture this coupling numerically: Both the fluid and the fish must be discretized in some way, and the relative motion between fluid and fish require a strategy for co-location of computational points. A common solution is to transform the fluid mesh to have nodes fitted to the body of the fish, but this method requires re-meshing at each time step, since the fish has a relative motion to the fluid grid. Such re-meshing is time consuming and the quality of the generated mesh is often not optimal due to grid-deformation.

Immersed Boundary Methods (IBMs), first proposed by Peskin (1972), avoids the problems related to re-meshing. Instead of requiring a co-location of computational points for the fluid and the boundary, the presence of the body is accounted for by adding a boundary force to the fluid equations, thus leaving the mesh untouched. Peskin (2002) states that the IBM in general is a useful method for FSI-problems, and "especially for biological fluid dynamics".

1.2 Objectives

The main objective of this Master's thesis is to develop a CFD-code implementing the Immersed Boundary Method, and clarify its use in 2-dimensional swimming fish problems. This objective is further subdivided into:

1. Provide background information about CFD related to fluid-structure interaction problems and the use of immersed boundary methods.
2. Develop a computer program that numerically solves the incompressible Navier-Stokes equations on a 2-dimensional domain.
3. Develop an IBM-module and implement it into the 2-dimensional Navier-Stokes solver.
4. Validate the program performance by comparing results to known numerical and/or experimental data.
5. Discuss how the code may be further developed into a program that can simulate a swimming fish.

1.3 Limitations

Only a personal laptop is available to develop the code. This severely limits the maximum domain resolution of the simulations, and thereby also the accuracy of the results.

1.4 About the report

Governing equations for a swimming fish problem are presented and simplified in Chapter 2. Chapter 3 provides an introduction to basic concepts within CFD and FSI that are of interest in this thesis, before Chapter 4 narrows in to a more profound description of the immersed boundary methods. Chapter 5 is devoted to numerical methods. The development of the computer code is described in Chapter 6, and validation of the code follows in Chapter 7 and 8. A discussion about the further development of the code is described in Chapter 9, before the thesis as a whole is concluded in Chapter 10. Lastly, Chapter 11 lists a selection of recommendations for further work within the topic of this thesis.

A postface, devoted to reflexions around personal experiences gained through the work, is included in the last page of the document.

Notation

Tensor notation, including the *Einstein summation convention*, will be extensively used throughout this document. For a brief introduction, see Appendix A.

Combination of tensor notation and discrete notation may cause misunderstandings. In equations where this represents an issue, care will be taken to warn the reader. In addition, matrix notation may in some cases substitute tensor notation in chapters where computer algorithms are described.

Chapter 2

Governing Equations

The governing equations form the mathematical model of a physical problem, and they reflect the applicable fundamental laws of physics. When considering swimming fish, which involves both fluid- and boundary motions, the mathematical model must consist of governing equations from both fluid- and solid mechanics. This coupling between a fluid-domain and a solid-domain is called *fluid-structure interaction.*

2.1 Governing fluid equations

Newtons second law of motion applied on a fluid cell yields the *Navier-Stokes equations* (N-S), and adding a mass conservation constraint on the cell gives rise to the *Continuity equation.* In marine applications, flows are mostly incompressible, and the two equations may be written as Equation 2.1 and 2.2, respectively.

$$\frac{\partial \hat{u}_i}{\partial \hat{t}} + \hat{u}_j \frac{\partial \hat{u}_i}{\partial \hat{x}_j} = -\frac{1}{\rho}\frac{\partial \hat{p}}{\partial \hat{x}_i} + \nu \frac{\partial^2 \hat{u}_i}{\partial \hat{x}_j \partial \hat{x}_j} \tag{2.1}$$

$$\frac{\partial \hat{u}_i}{\partial \hat{x}_i} = 0 \tag{2.2}$$

Here, the circumflex (ˆ) represents a quantity with a dimensional unit (e.g. $\hat{x}_i[m]$). $\hat{p}$ represents the pressure, and $\hat{u}_i$ represents the velocity component in the $\hat{x}_i$-direction. ν and ρ are the kinematic viscosity and the density, respectively.

The different acceleration terms of the N-S equation arise from the different physical forces[1] that the fluid cell is subject to. It is common to name the terms as shown in Table 2.1.

Table 2.1: Name of the terms in the N-S equation (Eq. 2.1).

Term	Name
$\frac{\partial \hat{u}_i}{\partial \hat{t}}$	Local acceleration
$\hat{u}_j \frac{\partial \hat{u}_i}{\partial \hat{x}_j}$	Advective acceleration
$-\frac{1}{\rho}\frac{\partial \hat{p}}{\partial \hat{x}_i}$	Pressure gradient acceleration
$\nu \frac{\partial^2 \hat{u}_i}{\partial \hat{x}_j \partial \hat{x}_j}$	Viscous acceleration

The Navier-Stokes equations is a set of three coupled non-linear second order *Partial Differential Equations* (PDEs) forming a combined *Boundary- and Initial Value Problem* (BVP and IVP respectively) for the field variables $\hat{u}_i$ and $\hat{p}$. In a 2D flow problem, the equations may be reduced to two directions ($i = 1,2$).

[1] It should be noted that the advective acceleration is not due to an actual force. It arises as a consequence of treating the fluid cell as a control volume (Eulerian approach) and not as a particle (Lagrangian approach) when applying Newtons second law.

2.1.1 Non-dimensional form

In qualitative research, it is convenient to use dimensionless variables. By substituting the non-dimensional quantities presented in Table 2.2 into the N-S and Continuity equations (2.7 and 2.2, respectively), we get the following non-dimensional expressions:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re} \frac{\partial^2 u_i}{\partial x_j \partial x_j} \tag{2.3}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.4}$$

As seen, the importance of the viscous term is determined by the Reynolds number, Re, which is a measure of the ratio between the advective term and the viscous term of the equation. Estimation of this ratio by use of the characteristic values from Table 2.2 gives the well-known formula:

$$Re = \frac{UL}{\nu} \tag{2.5}$$

Table 2.2: Non-dimensional quantities in fluid mechanics.

Quantity	Non-dimensional symbol	Characteristic scale
Velocity	$u_i = \frac{\hat{u}_i}{U}$	U
Position	$x_i = \frac{\hat{x}_i}{L}$	L
Time	$t = \frac{\hat{t}_i}{T}$	$T \sim \frac{L}{U}$
Pressure	$p = \frac{\hat{p}_i}{p_{dyn}}$	$p_{dyn} \sim \rho U^2$

2.1.2 Conservation forms

Equation 2.3 is written in its non-conservative form. Use of a mathematically equivalent conservation form[2] yields numerical advantages (Ferziger and Péric (2002)).

The conservative form of Equation 2.3 can be obtained by using the product rule of calculus:

$$\frac{\partial(u_i u_j)}{\partial x_j} = u_i \frac{\partial u_j}{\partial x_j} + u_j \frac{\partial u_i}{\partial x_j} \tag{2.6}$$

The first term of the *Right Hand Side* (RHS) of Equation 2.6 is zero according to the continuity equation (Equation 2.4). The advective term of the N-S equation (Equation 2.3) may therefore be written as $\frac{\partial(u_i u_j)}{\partial x_j}$, leading to the conservation form:

$$\frac{\partial u_i}{\partial t} + \frac{\partial(u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j \partial x_j} \tag{2.7}$$

[2] A differential equation is in conservation form if none of the field variables are outside a derivative (Anderson Jr. (2009)).

2.2 Governing equations for solid mechanics

A solid body in a fluid-structure interaction system will be subject to both rigid body motion and material deformations. The following subsections shows how these may be split in order to simplify the problem.

2.2.1 Rigid body motion

The rigid-body response of the swimming fish is determined by the fluid force. For a fully submerged fish, Newton's second law in all *Degrees of Freedom* (DoF) yields:

$$\begin{aligned} m_{ij}\frac{\partial^2 X_j}{\partial t^2} &= F_i, \\ i &= 1,...,6 \\ j &= 1,...,6 \end{aligned} \tag{2.8}$$

where m_{ij} is the inertia in the i-direction due to acceleration in the j-direction, X_j is the rigid-body position of the fish's *Center of Mass* (CoM) in the j-direction, and F_i is the total fluid force in the i-direction. Equation 2.8 is defined both for the translational DoFs (X_1, X_2, X_3) and the rotational dofs (X_4, X_5, X_6), and the unit for the inertial coefficient $\left(m_{ij}\right)$ differs accordingly.

2.2.2 Deformation

The governing equations for solid deformations depend on the material properties of the solid, such as elasticity and flexibility. For the analysis of fish swimming, the deformation equations need to include terms to account for muscle activation. This may be of importance when studying e.g. the fish's response to a certain muscle energy input, as the flap amplitude and velocity will be dependent of the fluid force on the fish. Such profound studies, however, fall outside the scope of this thesis. A simplified approach will be described in the next subsection.

2.2.3 Simplifications: A quasi-rigid approach

A simplified fish model may be obtained by treating the fish as a quasi-rigid solid, which does not deform from outer forces such as fluid-forces, but is free to change shape due to internal forces. By internal force is meant a force that does not apply any acceleration on or about the fish's CoM.

A quasi-rigid fish-model with a prescribed internal flapping motion may be interpreted as a fish with unlimited muscle power to counteract the effects of the fluid on the fish boundary.

The fish may be viewed as a closed system, so that during a flapping motion, the fish mass should be conserved. This means that the translational inertia of the fish is constant with time. The rotational inertia, on the other hand, is dependent of the fish's flapping configuration, and thereby of time.

The time-dependency of the rotational inertia must be taken into consideration when solving the rigid-body equation for rotational motion (Equation 2.8, for $j = 4,5,6$).

Newtons second law of motion for rotation:

$$\frac{\partial\left(\dot{\theta} I(t)\right)}{\partial t} = T(t) \tag{2.9}$$

becomes

$$\frac{\partial\dot{\theta}}{\partial t} I(t) + \dot{\theta}\frac{\partial I}{\partial t}(t) = T(t) \tag{2.10}$$

where $I(t)$ represents time-dependent moment of inertia, $\dot{\theta}$ the angular velocity and $T(t)$ the net torque, all quantities about the same axis. Thus the term $\frac{\partial m_{ij}}{\partial t}\frac{\partial X_j}{\partial t}$ must be added to the rotational versions of the rigid-body equation:

$$\begin{gathered} m_{ij}\frac{\partial^2 X_j}{\partial t^2} + \beta_i \frac{\partial m_{ij}}{\partial t}\frac{\partial X_j}{\partial t} = F_i, \\ i = 1,...,6 \\ j = 1,...,6 \\ \beta_i = \begin{cases} 0, & i = 1,2,3 \\ 1, & i = 4,5,6 \end{cases} \end{gathered} \tag{2.11}$$

Equation 2.11 represents the governing equation for the response of a quasi-rigid fish swimming with some prescribed internal motion that does not accelerate the any of the rigid-body degrees of freedom.

2.3 Boundary conditions

The behavior of a given quantity on a boundary is governed by its *Boundary Condition* (BC). There are two main types of BCs, depending on how the quantity is prescribed on the boundary:

- **Dirichlet**: The value of the quantity itself is prescribed.
- **Neumann**: The value of some *derivative* of the quantity is prescribed.

Table 2.3 shows some examples of Dirichlet and Neumann type boundary conditions.

2.3.1 Domain boundary condition

To model the domain within which the analysis is to take place, BCs must be specified along the domain boundaries. Different kinds of BCs may be chosen, depending on which physical behavior the BC should replicate. Some physical boundary behaviors and their mathematical model are presented in table 2.4. A slip-wall is often used to model the boundaries parallel to a the flow in flow problems with a highly defined flow direction.

Care should always be taken when modeling boundary equations. Extensive use of Neumann conditions may result in numerical systems that are indefinite, thereby unsolvable.

Table 2.3: Boundary condition types. u and V are the velocities of the fluid and the boundary respectively, and p denotes the pressure. The subscript $(\cdot)_n$ denotes the direction normal to the boundary, and the subscript $(\cdot)_t$ denotes the direction tangential to the boundary.

Physical	Mathematical	Type
Impermeability	$u_n = V_n$	Dirichlet on velocity
No-slip	$u_t = V_t$	Dirichlet on velocity
Slip (No-shear)	$\frac{\partial u_t}{\partial x_n} = 0$	Neumann on velocity
Dynamic equilibrium	$\frac{\partial p}{\partial x_n} = 0$	Neumann on pressure

Table 2.4: Relation between physical boundary behavior and the related mathematical model. u and V are the velocities of the fluid and the boundary respectively. U and P stands for some specified value. The subscript $(\cdot)_n$ denotes the direction normal to the boundary, and the subscript $(\cdot)_t$ denotes the direction tangential to the boundary, thus Einstein's summation convention is not to be applied.

Physical	Normal velocity	Tangential velocity	Pressure
Wall	$u_n = V_n$	$u_t = V_t$	$\frac{\partial p}{\partial x_n} = 0$
Slip-wall	$u_n = V_n$	$\frac{\partial u_t}{\partial u_n} = 0$	$\frac{\partial p}{\partial x_n} = 0$
Inlet	$u_n = U$	$u_t = 0$	$\frac{\partial p}{\partial x_n} = 0$
Outlet	$\frac{\partial u_t}{\partial x_n} = 0$	$\frac{\partial u_n}{\partial x_n} = 0$	$p = P$

2.3.2 Fluid-Structure boundary conditions

The fluid domain and the solid domain are coupled through the BCs for the fluid-structure-interface. In the swimming fish problem, the fluid-structure boundary conditions are:

- impermeability
- no-slip[3]
- dynamic equilibrium

which may be modeled by the wall-condition described in Table 2.4.

[3]Using a no-slip condition for a fish is actually a simplification. The fish surface is slippery, and a slip-condition, which sets a value on the wall shear stress τ_w instead of on the velocity itself, could be used.

Chapter 3

Computational Fluid Dynamics

The general idea of CFD is to approximate the solution of a mathematically stated fluid problem by use of numerics. If the problem is *well posed*[1], the approximate solution will approach the exact solution in the limit where the discretization resolution goes to infinity. The numerical result will never become the exact solution, but it may approach the exact solution by acceptable errors.

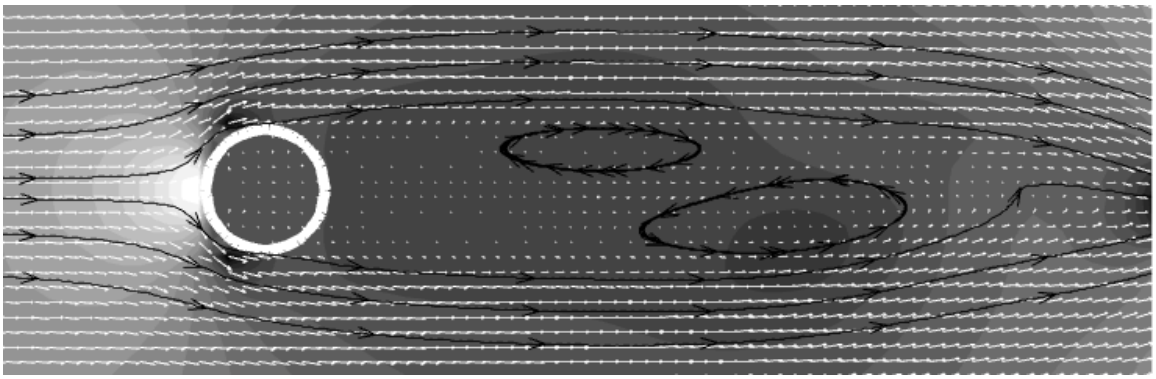


Figure 3.1: A typical CFD-result from simulation of uniform 2D flow past a circular cylinder.

[1] Well posed problem: A problem where a unique solution exist, and the solution depends continuously on the problem parameters (Tannehill et al. (1997))

3.1 General approach

In general, a CFD procedure may be divided into the following tasks

1. Mathematical modeling of the physical problem that is to be studied, including:
    - Governing equations
    - Initial- and boundary conditions
    - Acceptable simplifications
2. Discretization of the mathematical model
    - Choice of discretization method
    - Choice of computational grid type
3. Solving the discrete equations
4. Post processing of calculated data (Illustrated in Figure 3.1)

The following sections will discuss this approach applied on the governing equations given in Section 2.1.

3.2 Discretization

The mathematical operators in the N-S equation (Equation 2.7) may be approximated by discrete operators through a discretization method. Many different approaches exist, such as the *Finite Volume Methods* (FVM) and the *Finite Difference Methods* (FDM), and each of them contains various numerical schemes designed to handle different types of problems. The formulation of the discrete operator depends on the method used, as well as on the grid chosen for the discretization method, and due to the large amount of different schemes, it is handy to use a general notation for the discretized equations.

By use of the notation presented in Table 3.1, a general discretized version of the N-S equation (2.7) and the Continuity equation (2.4) may be written:

$$\mathscr{T}(u_i) = -G_i(p) + \frac{1}{Re}L(u_i) - A(u_i) \tag{3.1}$$

$$G_i(u_i) = 0 \tag{3.2}$$

Note that Equation 3.1 has been rearranged so that all spatial derivatives lies on the RHS of the equation and the temporal derivative lies on the *Left Hand Side* (LHS). This is done as it, numerically speaking, may be beneficial to treat the temporal and spatial derivatives differently, as shall be seen in Section 3.5

Table 3.1: General notation for the discrete difference operators used on the different terms of the NS-equations (Eq. 2.7).

Description	Discrete operator	Continuous equivalent
Time derivative	$\mathcal{T}(\cdot)$	$\frac{\partial(\cdot)}{\partial t}$
Advection	$A(\cdot)$	$u_j \frac{\partial(\cdot)}{\partial x_j}$ or $\frac{\partial(u_j(\cdot))}{\partial x_j}$
Gradient	$G_i(\cdot)$	$\frac{\partial(\cdot)}{\partial x_i}$
Laplacian	$L(\cdot)$	$\frac{\partial^2(\cdot)}{\partial x_j \partial x_j}$

3.3 Grids

A mesh or grid consist of spatially discrete locations where the field variables of the discretized equations are represented. Each such location is called a node, and it is common to represent a grid by drawing a mesh of lines that interconnect each node to its nearest neighbors. A fluid cell may be defined as the area spanned by a group of nodes, or as the area enclosing certain nodes as seen in Figure 3.2.

The succeeding subsections will describe important concepts regarding meshing.

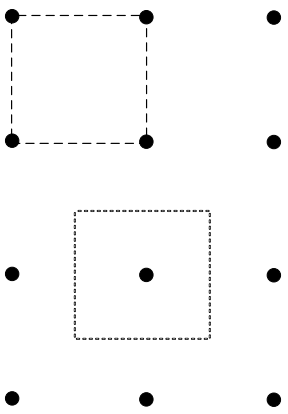


Figure 3.2: Two types of fluid cell definition in a grid.

3.3.1 Grid quality

The quality of a mesh is measured from the numerical results of the analysis. A good mesh quality may support accuracy, stability and effectiveness of the solution process. ©Fluent (2006) lists the following measures for good mesh quality:

Smoothness

The difference in size of neighboring mesh-cells is affecting accuracy due to an increased truncation error. A smooth transition between smaller and larger cells is therefore preferable.

Skewness

The optimal shape for a cell is the one that maximizes the vertex angles for all vertexes simultaneously. This means that the optimal vertex angle is 90 degrees for a quadrilateral cell, and 60 degrees for a triangular cell. Skewness is defined as the deviation from this optimal shape, and high skewness may result in a significant decrease in accuracy, and even instabilities.

Aspect ratio

For flows with an isotropic behavior, an extreme aspect ratio on the grid entails a lower resolution of, and thereby accuracy of, the flow in the longitudinal cell direction. This causes the order of magnitude of the overall solution error to increase.

High quality grids

With the quality measures described above in mind, it is clear that a Cartesian grid, where all cells are uniformly sized, the aspect ratio is equal to one, and the cell angles are all 90 degrees, is a high quality grid. A regular rectilinear grid have the same attributes as the Cartesian grid, apart from the aspect ratio which may be different from one. These grids are also of high quality, given that they represent the flow system with sufficient resolution in each direction.

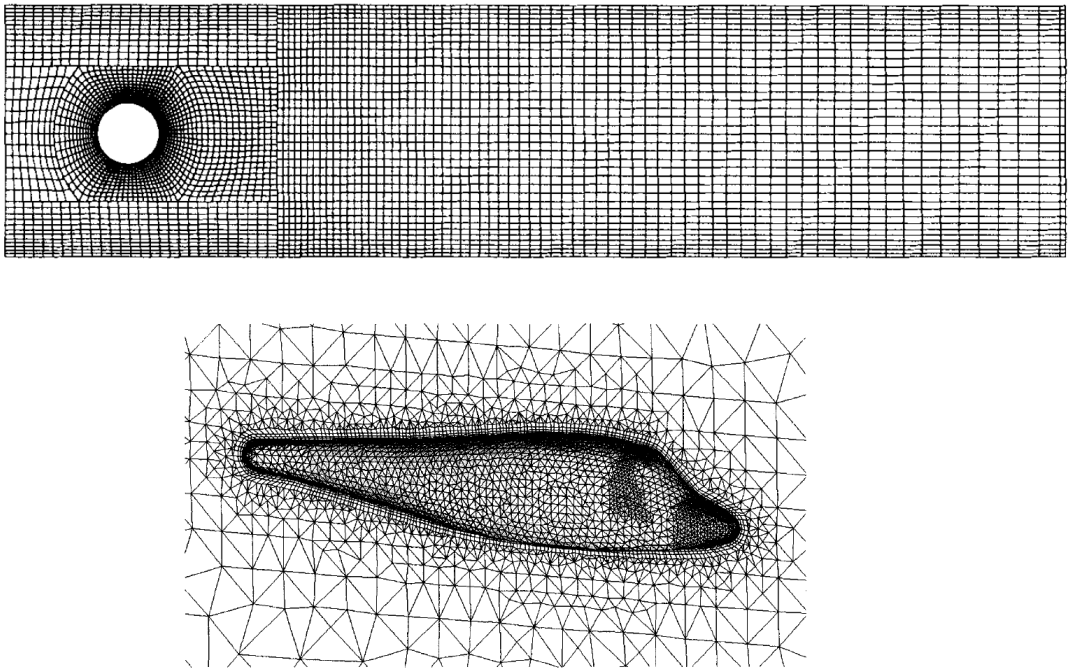


Figure 3.3: Structured (top) and unstructured (bottom) grids. (Modified, from Ferziger and Péric (2002)).

3.3.2 Structured vs. unstructured grid

A grid where each node may be defined uniquely by an integer address index (i, j, k in three dimensions), is called a *structured grid*, while any grid that does not pose this characteristic are called *unstructured* grids. The strength of a structured grid is that it is easy to program, as the address system cooperates well with the architecture of a computer. This is not the truth for an unstructured grid, which requires a lot more care during implementation. On the other hand, an unstructured grid proves to be much more versatile in use (Ferziger and Péric (2002)), as it is easily wrapped around irregular domain boundaries without a large decrease in cell quality (see Figure 3.3).

3.3.3 Staggered vs. co-located grid

The difference between the staggered and co-located grid configurations lies in where the field variables are represented. If all field variables are represented at the same set of nodes, the grid is said to be co-located. A staggered arrangement is achieved by shifting the location of the nodes so that the different field variables are represented at different locations (see Figure 3.4). The latter configuration was first shown by Harlow and Welch (1965), and has an advantage when it comes to pressure calculations. This is because the co-located arrangement in some cases allows for a decoupling of the numerical pressure solving scheme, such that a *checkerboard* pattern may develop in the solutions. A staggered grid imposes a numerical coupling between the pressure nodes, so that this is avoided.

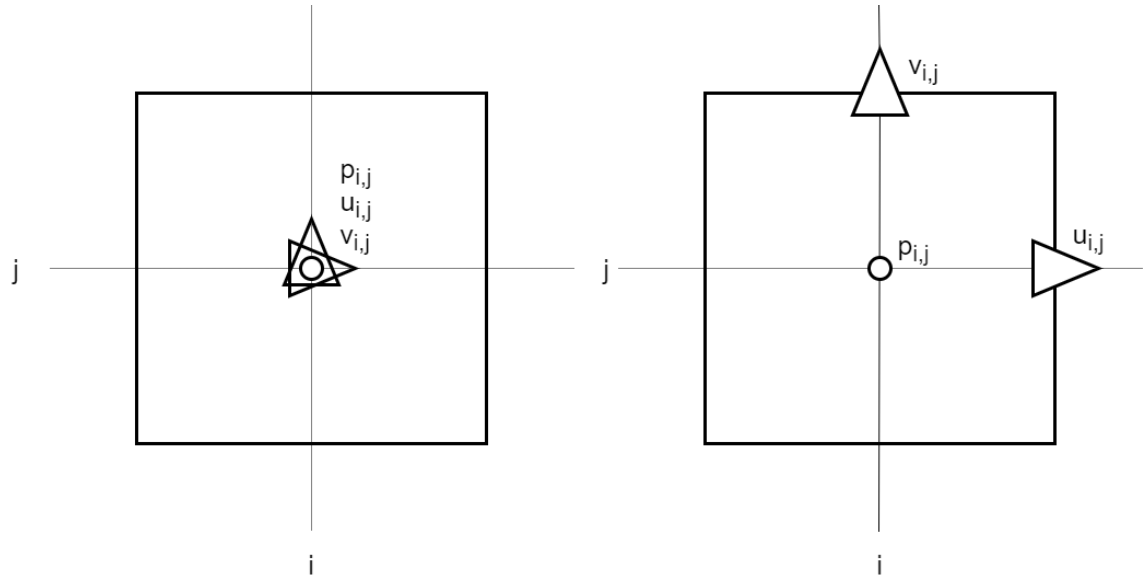


Figure 3.4: Co-located (left) and staggered (right) grid cells. In the co-located version, all field variables related to cell (i, j) are represented at the middle of the cell. For the staggered version, the pressure is represented at the cell center, while the velocities are represented at the cell faces.

3.3.4 Adaptive mesh refinement

Flow fields are rarely uniform, in that there are certain locations of the domain that have higher gradients and rates of change than others. This is especially true for flow past structures. To provide an accurate solution of the flow field, the grid resolution must be high enough to represent the parts with the highest velocity and pressure gradients. In the case of a Cartesian grid (which has a uniform cell size), this results in unnecessarily high resolution also in the remote domain, where the flow field is nearly uniform. This to an inefficiency, since a lot of equations that give little increase in accuracy for the solution must be solved. Another problem is for flow-simulation past non-stationary structures, where it is not known a priori where the high gradient areas will be located, and how high resolution they will be needing in order to be represented properly.

A solution to these issues is *Adaptive Mesh Refinement*-techniques (AMR). In short, these methods detect areas close to a structure, or other areas that have a high gradient flow field, and refine the mesh accordingly. This allows for an overall solution on a coarse mesh, with only specific parts refined (see Figure 3.5).

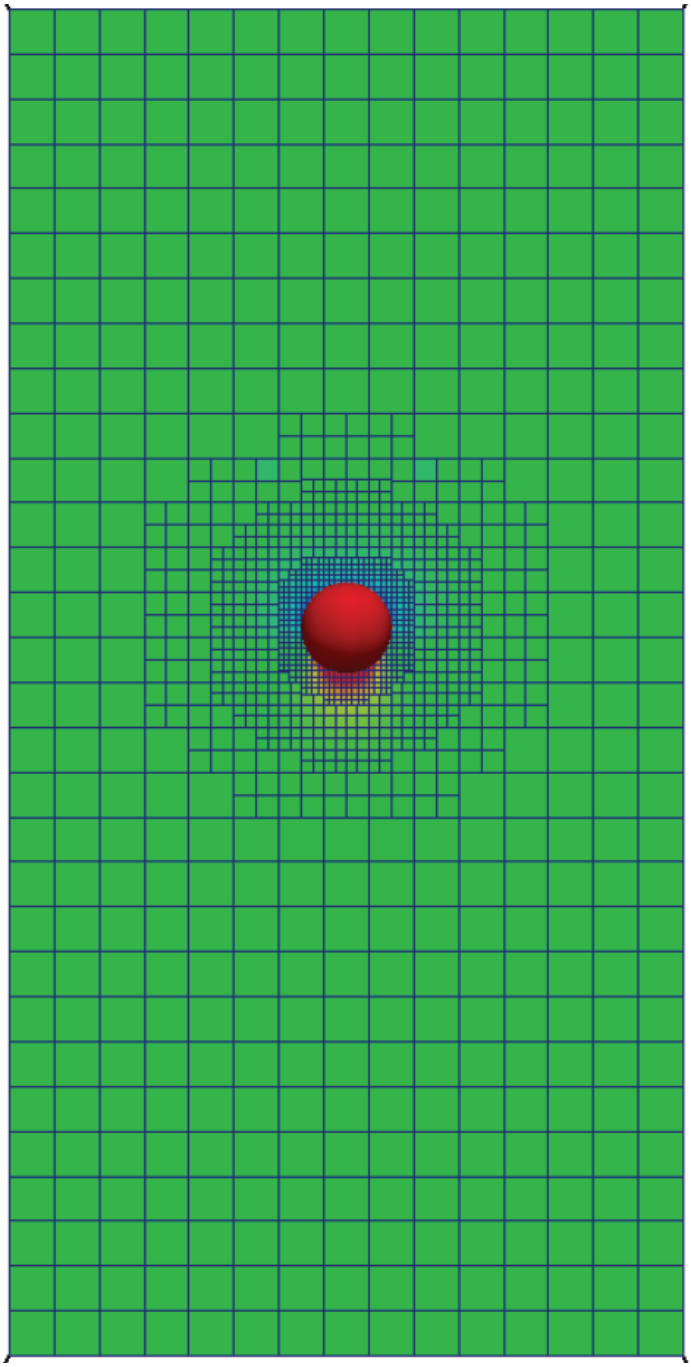


Figure 3.5: Adaptive mesh refinement on a Cartesian mesh around a sphere. (Modified, from Mark et al. (2011)).

3.4 Application of boundary conditions

Application of boundary conditions on staggered grids requires the use of so-called *ghost cells*, which are computational cells situated outside the fluid boundary (see Figure 3.6). A Dirichlet condition may be assigned directly onto a node, given that the node is situated on the domain boundary. Where this is not the case, the Dirichlet Boundary condition must be applied in the nearest ghost cell. Ghost cells are also needed when applying Neumann conditions.

Following the method by Harlow and Welch (1965), which is using a 2nd order FDM-scheme for the spatial derivatives, the Dirichlet conditions may be applied by assigning the ghost node with the reversed value of the fluid node closest to the wall (see Figure 3.7). The Neumann pressure condition is applied by mirroring the fluid velocity in the ghost node, so that the gradient of the linear extension between the nodes become zero.

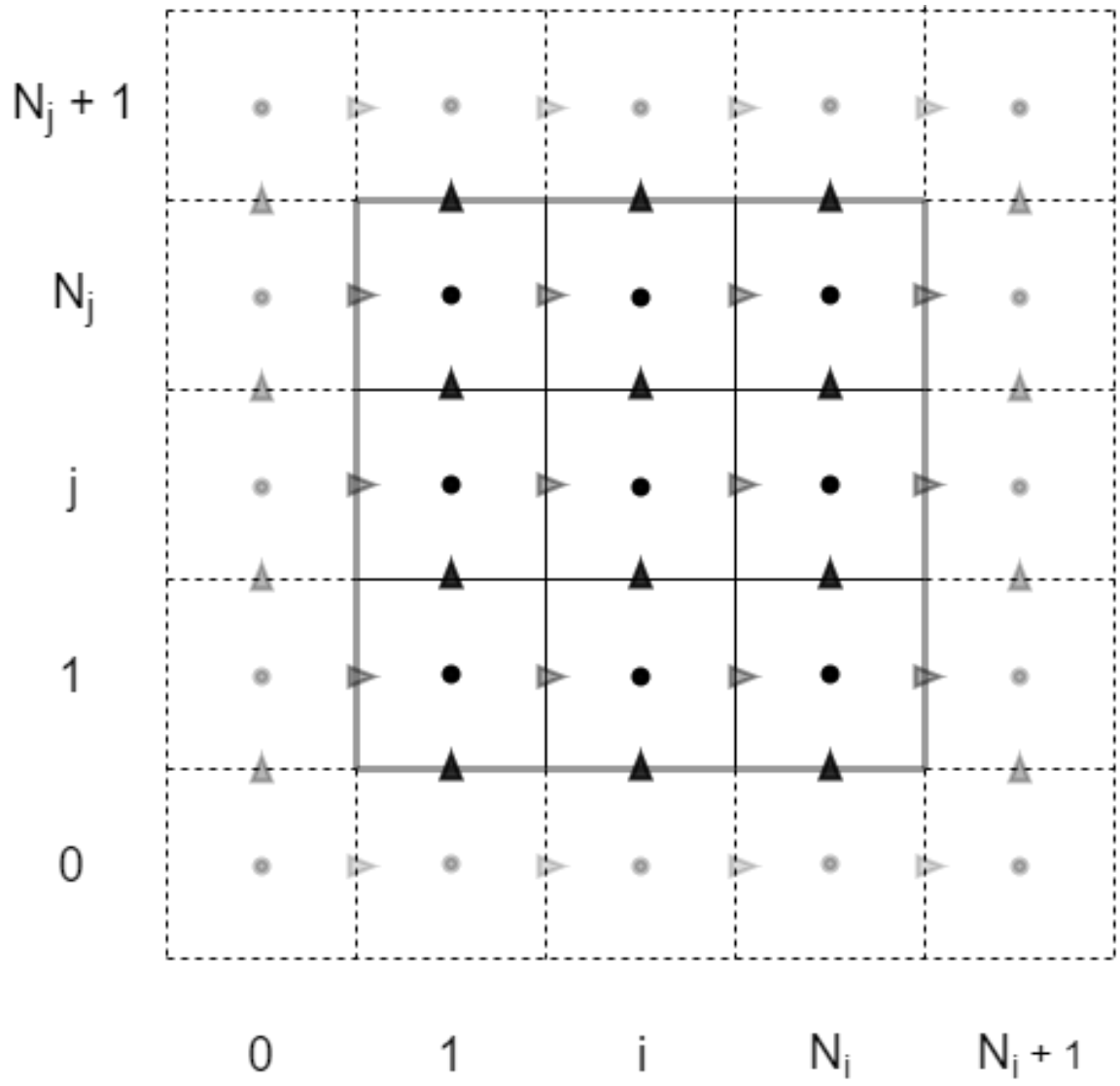


Figure 3.6: Application of boundary conditions on a staggered grid. The wide gray line delimits the fluid domain, while dashed lines represents the ghost cells situated outside the fluid. As an example, a Dirichlet condition on the vertical velocity, v, at the leftmost vertical wall would be assigned with $v_{0,j} = -v_{1,j}$. The Neumann pressure condition at the same wall would be assigned by $p_{0,j} = p_{1,j}$.

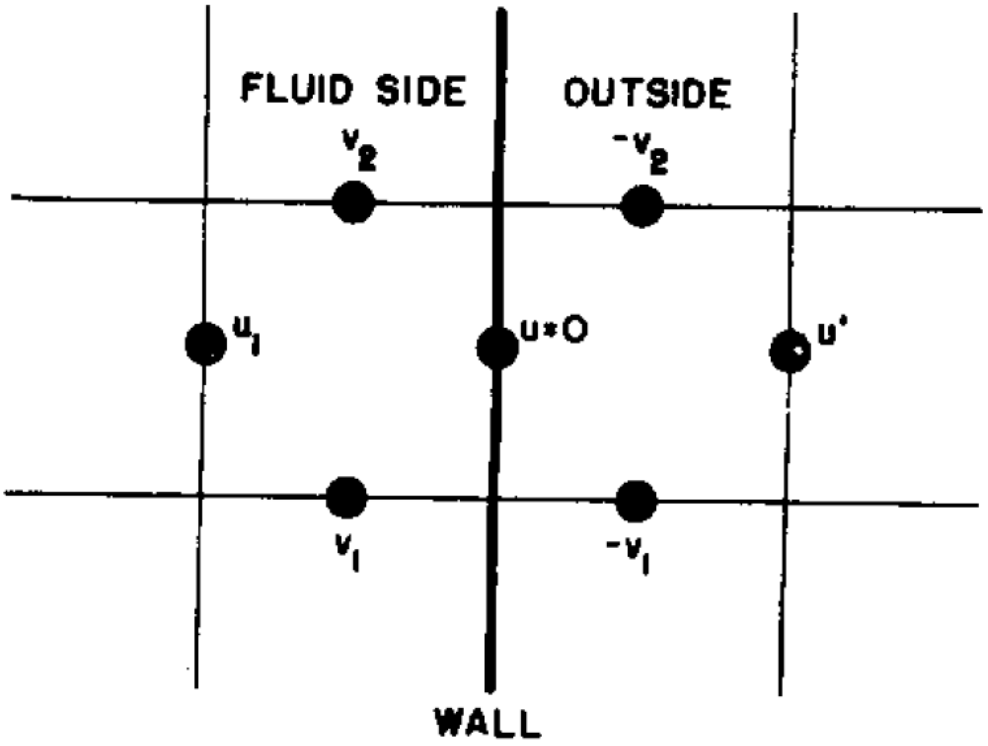


Figure 3.7: Dirichlet boundary condition on the velocities, applied on a vertical domain boundary. (From Harlow and Welch (1965)).

3.5 Semi-discrete approach

In CFD, a common solution strategy is to discretize the spatial operators first, leaving the time-derivative continuously defined (Lomax et al. (1999)). This means that all spatial derivatives are approximated by numerical values, and the original PDE is reduced to a semi-discrete *Ordinary Differential Equation* (ODE):

$$\frac{\partial u_i}{\partial t} = RHS(u_i, p) \tag{3.3}$$

$$RHS(u_i, p) = -G_i(p) + \frac{1}{Re}L(u_i) - A(u_i) \tag{3.4}$$

where the RHS may be solved as a pure boundary value problem at some chosen time-step. The solution of this BVP represent the slope of the solution, u_i, at that time step, and numerical integration methods such as the Euler-method and the Runge-Kutta methods (see Section 5.2) may be used to propagate the solution of u_i in time, thus solving the initial value problem. Note that this section only describe how to integrate the velocities. The process of solving for the pressure, p, is quite different, as is described in Section 3.6.

3.6 Pressure-Velocity coupling

The governing equations for *incompressible* flow (Equation 2.7 and 2.4) do not include any time-derivative of the pressure, meaning that the pressure cannot be time-integrated in the same way as the velocity. In addition, the semi-discrete approach may fail to satisfy the continuity equation for the integrated velocity field. This problem may be bypassed by taking advantage of the interdependencies between the velocity and pressure. The *Fractional Step Method* (FSM) is a widely used algorithm for this purpose.

3.6.1 The Fractional Step Method

FSM is also known by names such as the *Projection Method* and the *Time Splitting Method*, and was first proposed by Chorin (1967). The following explanation of the method follows the one given in Ferziger and Péric (2002), and is valid for explicit schemes.

Using an explicit Euler method (see Section 5.2) for the time derivative ($\mathscr{T}$) of equation 3.1, and keeping the RHS un-discretized, the following equation arise:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -\frac{\partial p^n}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i^n}{\partial x_j \partial x_j} - \frac{\partial (u_i u_j)^n}{\partial x_j} \tag{3.5}$$

The solution for u_i^{n+1} of this equation is not assured to satisfied the continuity equation, and there is no direct solution of the pressure for the next time step $(n+1)$. These issues are turned into advantages as shown in the further explanation of the method.

Let Equation 3.5 be valid for some temporary velocity field, $\tilde{u}_i$, that is not constrained by the continuity equation:

$$\frac{\tilde{u}_i - u_i^n}{\Delta t} = -\frac{\partial p^n}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i^n}{\partial x_j \partial x_j} - \frac{\partial (u_i u_j)^n}{\partial x_j} \tag{3.6}$$

It is now assumed that the pressure at time step $n+1$ may support continuity of u_i^{n+1}, such that the following equation provides a solution that satisfies continuity:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -\frac{\partial p^{n+1}}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i^n}{\partial x_j \partial x_j} - \frac{\partial (u_i u_j)^n}{\partial x_j} \tag{3.7}$$

The pressure field at $n+1$ may be defined as:

$$p^{n+1} = p^n + \delta p \tag{3.8}$$

Where δp is the correction in pressure needed to achieve that p^{n+1} supports continuity of u_i^{n+1}. Subtraction of Equation 3.6 from Equation 3.7 yields:

$$\begin{aligned} \frac{u_i^{n+1} - \tilde{u}_i}{\Delta t} &= -\frac{\partial(p^{n+1} - p^n)}{\partial x_i} \\ &= -\frac{\partial(\delta p)}{\partial x_i} \end{aligned} \tag{3.9}$$

By taking the divergence of Equation 3.9 , the continuity constraint comes into play. As the flow field u_i^{n+1} is required to satisfy continuity, it vanishes from the equation and we are left with the following *Poisson equation* for the pressure correction:

$$\frac{1}{\Delta t}\frac{\partial \tilde{u}_i}{\partial x_i} = \frac{\partial^2(\delta p)}{\partial x_i \partial x_i} \tag{3.10}$$

The Poisson equation is a linear PDE, and may readily be solved numerically as a set of linear equations on the form

$$A_{ij}x_j = b_i \tag{3.11}$$

as will be discussed in Section 5.3. Using the general discrete notation of Table 3.1, a discrete version of the Poisson equation may be written:

$$L(\delta p) = \frac{1}{\Delta t} G_i(\tilde{u}_i) \tag{3.12}$$

Once the Poisson equation (3.10) is solved, both $\tilde{u}_i$ and δp are known, so that u_i^{n+1} may be found directly from Equation 3.9. This velocity field is assured to satisfy the continuity constraint.

In brief, the FSM uses the possibly divergent flow field $\tilde{u}_i$ to find a correction δp to the

pressure p^n that results in a pressure $p^{n+1} = p^n + \delta p$ that supports continuity of the flow field u_i^{n+1}.

It should be noted that the value of p^{n+1} is independent of the value of p^n, as the pressure correction is calculated from the continuity constraint. Equation 3.6 may therefore be evaluated with an arbitrarily chosen pressure field, yielding a different correction term so that the pressure at time step $n+1$ still holds for continuity. This feature may be used to simplify the algorithm: Choose a uniform pressure field at time step n, (e.g. $p = 0$, everywhere) so that the pressure gradient term equals zero. Now, from the definition of the corrected pressure (Equation 3.8), we get $p^{n+1} = \delta p$, so that the Poisson equation (Equation 3.10) gives the value of p^{n+1} directly. This approach is clearly easier to implement into a computer code, but care should be taken because the omittance of the pressure field at time step n may cause issues, for example when it comes to force calculations.

3.7 Solution quality tests

As stated in the introduction of this chapter, *the numerical solution will never become the exact solution.* As such, it is important to check the quality of the numerical results, both during and after calculations. The indications from such tests may both warn about large errors in the solution, and ease the process of finding the source of such errors.

3.7.1 Cell Courant-number

The *Courant-Friedrich-Lewy*-number (CFL) (also covered in Section 5.5) is related to the stability of the method used. For a fluid cell defined by the mesh size Δx_i, the number can be written as

$$CFL_{cell} = \frac{u_i}{\Delta x_i}\Delta t \tag{3.13}$$

On this form, the number may be interpreted as the number of cells a particle will pass through in one time step given that the particle holds the velocity of that cell, and for explicit methods the number should be less than 1 (Ferziger and Péric (2002)). Calculation of the CFL-number in every cell may reveal which area of the discretized domain that is the source of an eventual instability. This is of good use for non-uniform grids, where certain domain regions may have too small cells, a problem that can be resolved with local mesh-rearrangement, or by decreasing the time step Δt.

3.7.2 Discrete continuity equation

Though it is used for the analytical derivation of the pressure *Poisson* equation (Equation 3.10), the continuity equation (Equation 2.4) need not be solved *numerically* for incompressible flows. The discrete version of the continuity equation (Equation 3.2) may however be used as a test for the quality of the numerically solved flow field, as this should satisfy continuity. Such tests may be carried out for single or grouped fluid cells, or for the domain as a whole.

3.8 Fluid-Structure Interaction

The previous sections have described some important steps in solving the governing equations for a fluid alone. The mathematical model for flows around a structure turns out more complicated. The simultaneous exertion of forces between structure and fluid entails a need for a coupled model (as discussed in Chapter 2).

3.8.1 Modeling approaches

There are two main approaches to mathematically model the coupling between fluid and structure. The *monolithic approach* aims at modeling the problem as a whole, whereas the *partitioned approach* models the two problems in separate solution domains, linked by some interface conditions (e.g. the boundary conditions presented in Section 2.3). A schematic of the two approaches is represented in Figure 3.8.

According to Sotiropoulos and Yang (2014), solution algorithms from monolithic approaches are unconditionally stable. Hou et al. (2012) states that they also lead to more accurate numerical schemes. The downside of such models is that every specific problem needs its own specific model, leaving little room for versatility in the design of a solution algorithm. Partitioned approaches on the other hand, enables for a high grade of modularity for the solution algorithms. Well tested and established solvers for both the fluid and the structural models may thus be combined to form a package of interconnected software that is highly versatile. This is of great value when studying e.g. a large range of different aquatic propulsion strategies.

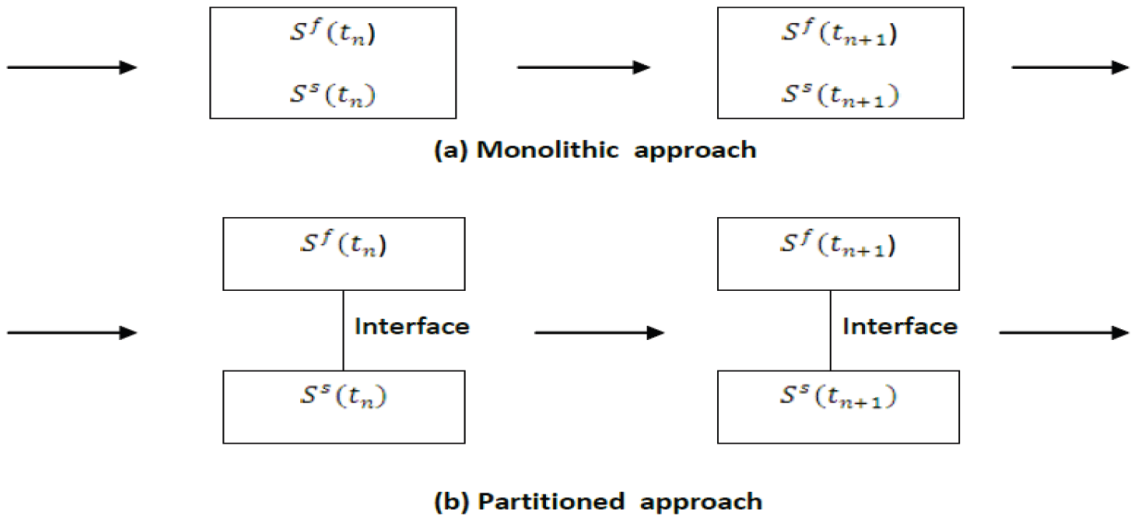


Figure 3.8: Schematic of the monolithic approach (a) and the partitioned approach (b) for fluid-structure interactions, where S^f and S^s denote the fluid and structure solutions, respectively. (From Hou et al. (2012)).

3.8.2 Solution algorithms

As presented in Subsection 3.8.1, the mathematical model of a fluid-structure interaction problem may be either monolithic or partitioned. Due to the lack of versatility, monolithic approaches will not be further investigated in this thesis. Instead, the focus will be on the numerical implementation of partitioned approaches.

The separate domains of a partitioned model will be discretized by separate grids, and some form of conformity is needed between these. The interface between the two domains is mathematically governed by the boundary conditions (see Section 2.3), and conformity may be achieved through use of a *conforming mesh* method, where the nodes from each domain conforms along the domain interface (see Figure 3.9). This allows for satisfaction of the boundary conditions directly on the interface, which is good regarding accuracy. However, this strict conformity requires that the fluid mesh is regenerated for every time step when the structure is moving. Such re-meshing is computationally costly, and the grid quality (see Section 3.3) of the re-meshed fluid domain may be significantly lowered.

The *non-conforming mesh* methods allows for fixed fluid grids, and therefore avoids the problems connected to re-meshing. It is among these methods we find the *Immersed Boundary Methods* (IBMs), which will be described in more detail in Chapter 4.

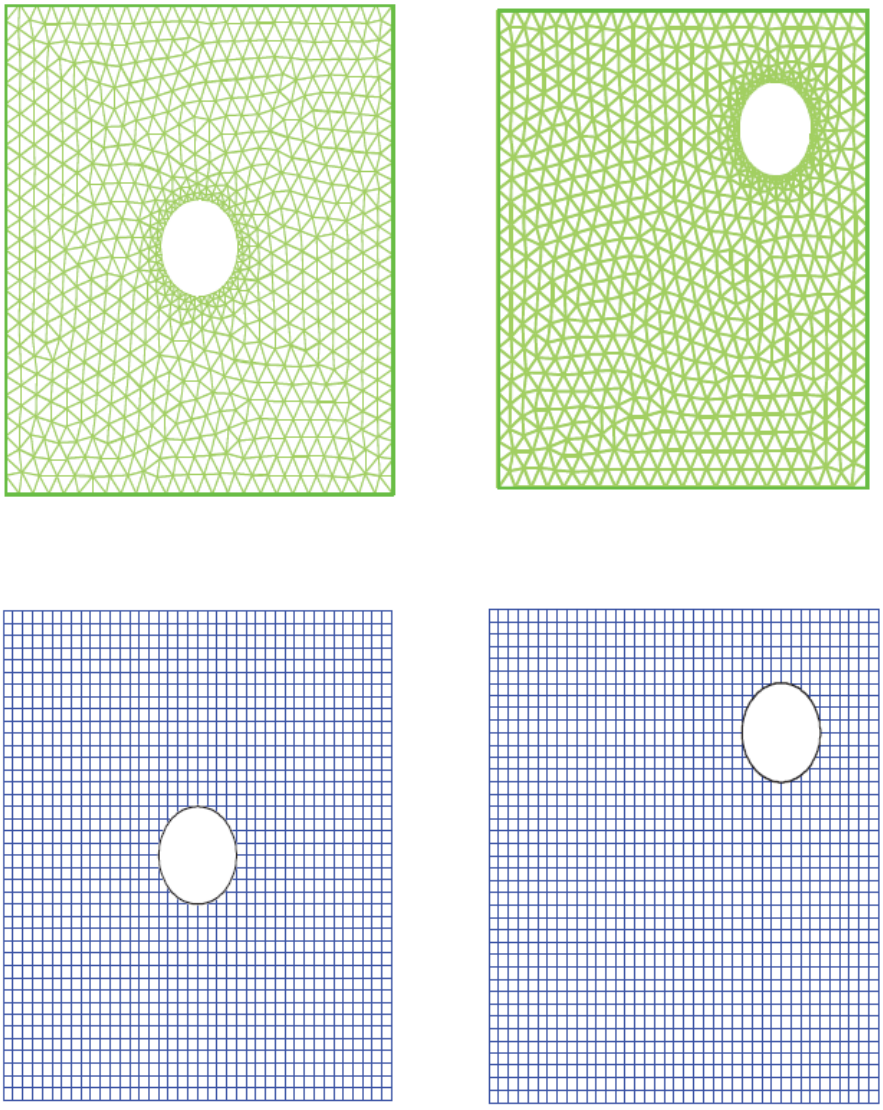


Figure 3.9: Conforming (top) and non-comforming (bottom) mesh following a cylinder with time. (Modified, from Hou et al. (2012)).

Another important aspect of the partitioned FSI methods is the degree of coupling between the solid and fluid domain. Sotiropoulos and Yang (2014) distinguish between *loose* and *strong* coupling, and state that the loosely coupled schemes are prone to numerical instabilities. The strongly coupled schemes overcomes this by solving the structural and fluid parts *iteratively* for each time step, thus ending up with a stable, but also more costly solving process.

Chapter 4

Immersed boundary methods

The foundation for the IBM was laid by Peskin (1972), as he described a way to use a fixed rectangular domain to simulate blood flowing through a moving heart-valve (see Figure 4.1). The great innovation of the method was to replace the mesh-conforming boundary description with a field force *replicating* the presence of the boundary. This force might be distributed onto the fixed grid without need for grid-deformation and re-meshing, that is: The governing equations could be discretized onto an Eulerian mesh (see Appendix B) of high quality.

Equation 4.1 and 4.2 shows how the boundary force is applied into the fluid equations in a general IBM approach.

$$\frac{\partial u_i}{\partial t} + \frac{\partial (u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j \partial x_j} + f_i \tag{4.1}$$

$$\mathscr{T}(u_i) = -G_i(p) + \frac{1}{Re}L(u_i) - A(u_i) + f_i \tag{4.2}$$

A Lagrangian representation of the position of the immersed boundary relatively to the Eulerian mesh points is needed in order to calculate the strength and distribution of the Eulerian body-force (f_i) in Equation (4.1).

On the immersed boundary (Γ_b), an *Immersed Boundary Condition* (IBC) is applied (e.g a wall BC as discussed in Section 2.3):

$$\begin{aligned} u_i &= V_i \quad \text{and} \\ \frac{\partial p}{\partial x_n} &= 0 \quad \text{on the immersed boundary } (\Gamma_b) \end{aligned} \tag{4.3}$$

where subscript $(\cdot)_n$ denotes the direction normal to the boundary.
When put into Equation 4.1, the velocity condition of 4.3 yields a solution for a boundary force (F_i) that is valid **on** Γ_b. F_i may further be represented as a set of singular forces F_i^k at discrete points (k) along the boundary.

The next operation is to distribute the effect of the singular boundary forces F_i^k onto the underlying Eulerian mesh ending up with a distributed body field force f_i that is defined on the underlying Eulerian mesh. The process of finding and distributing this field force, f_i, varies among the different types of IBM, which will be discussed in the next section.

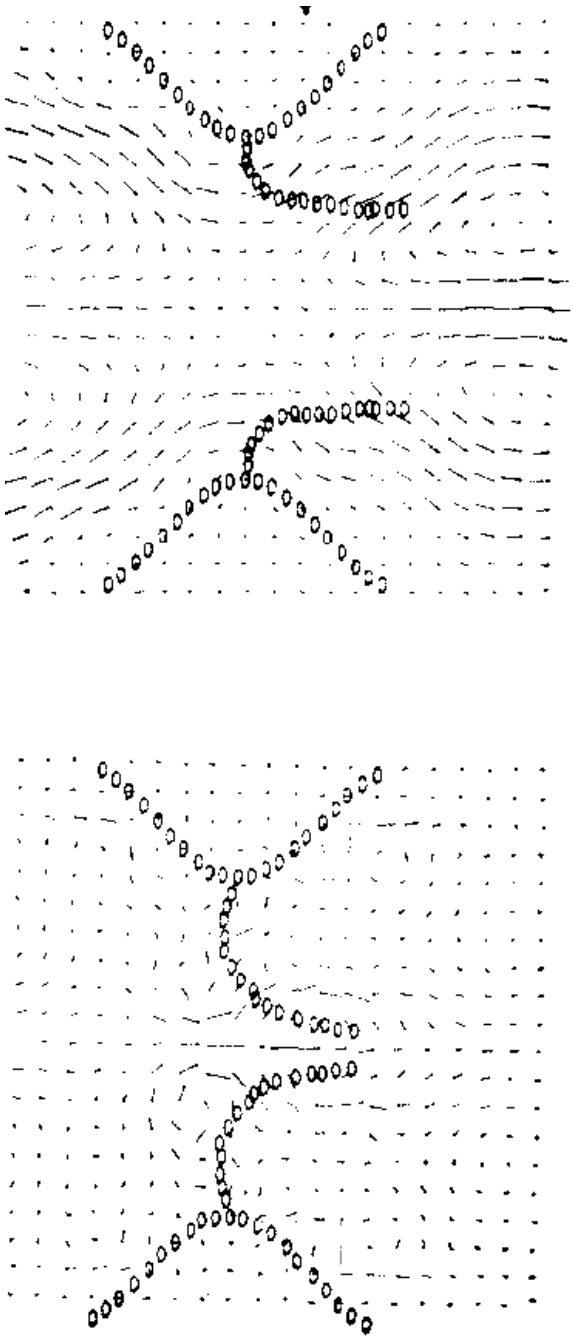


Figure 4.1: Simulation of blood flowing through a heart valve. The valve leaflets are allowed to move with the flow, but they are also governed by material laws for deformation so that the valve seals when flow is going from right to left. (Modified, from Peskin (1972)).

4.1 Different IBM approaches

Since Peskin (1972), the IBM concept has been developed into several new methods. Mittal and Iaccarino (2005) divide the family of IBMs into two groups, based on their different approach to finding the immersed boundary force:

- Continuous forcing methods
- Direct forcing methods

The different IBMs are broadly discussed in Mark et al. (2011), Mittal and Iaccarino (2005), Sotiropoulos and Yang (2014) and Peskin (2002). This section contains a brief summary of the two groups of methods.

4.1.1 Continuous forcing methods

In most continuous forcing approaches, the immersed boundary is discretized by Lagrangian points, k, and F_i^k is calculated at these points, before being smeared out to the underlying grid by some sort of distribution function (see Figure 4.2).

The original method by Peskin (1972) modeled the immersed boundary as a chain of elastic filaments, so that the force from the boundary on the fluid could be calculated from Hooke's law. The elongation of the filaments was derived from the immersed boundary condition: $u_i = V_i$ on Γ_b. To distribute the force to the underlying mesh, he made use of a discrete Dirac delta function, and that is still the most common approach for the distribution.

There are some issues with the continuous forcing approach: They are for example unsuited for simulations of rigid body boundaries, due to mis-behavior of the constitutive laws in the rigid limit (Mittal and Iaccarino (2005) and Sotiropoulos and Yang (2014)). In addition, it is hard to obtain a high order of accuracy, as mentioned by e.g. Guy and Hartenstine (2009): "the use of delta functions limits the order of accuracy near the boundary to first order in space".

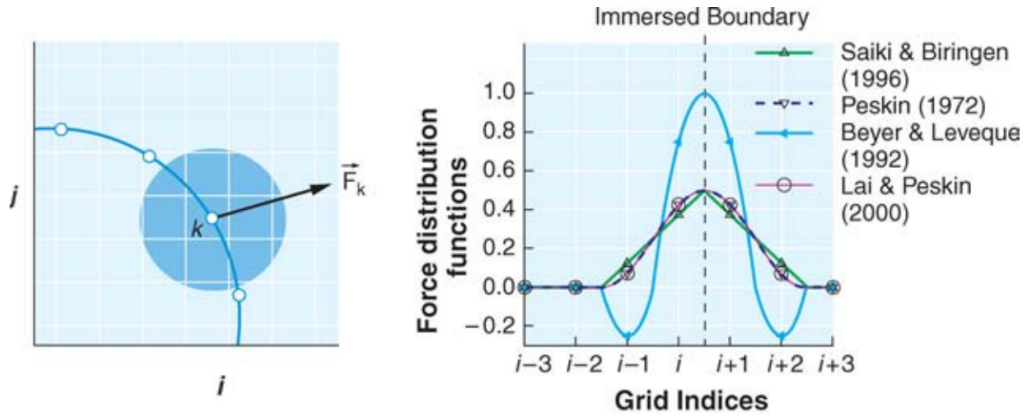


Figure 4.2: Left: The boundary force F^k is found at the specified Lagrangian points of the immersed boundary, before being distributed out to the surrounding fluid. The shaded region represents the area onto which F^k is distributed. Right: Different possible distribution functions. (From Mittal and Iaccarino (2005)).

4.1.2 Direct forcing methods

The direct forcing approach was first presented by Mohd-Yusof (1997), and its name reflects that the forcing term, f_i, is imposed directly onto the Eulerian grid without being calculated from material constitutive laws.

Consider Equation 4.2 with an explicit Euler method used for $\mathscr{T}$.

$$\begin{aligned} \frac{u_i^{n+1} - u_i^n}{\Delta t} &= -G_i(p) + \frac{1}{Re} L(u_i) - A(u_i) + f_i \\ &= RHS + f_i \end{aligned} \tag{4.4}$$

Mohd-Yusof's formulation for the boundary force F_i springs directly out from the equation when the IBC ($u_i = V_i$ on Γ_b) is applied:

$$F_i = \begin{cases} \frac{V_i^{n+1} - u_i^n}{\Delta t} - RHS & \textbf{directly on } \Gamma_b \\ 0 & \text{elsewhere} \end{cases} \tag{4.5}$$

A direct transfer of this force to the Eulerian fluid points, that is: $F_i = f_i$, is only valid when the Eulerian points coincide with the immersed boundary. This is rarely the situation, and therefore some kind of interpolation must be done to find a suitable value for f_i at the Eulerian mesh points. If the numerical schemes are chosen so that u_i^n and RHS are known, then the boundary forcing F_i is determined solely by the immersed boundary velocity. This means that an interpolation of the fluid velocity that mimics a typical behavior of a wall-near velocity profile on the Eulerian nodes will yield a good numerical estimate for the forcing term f_i.

A modified version of Mohd-Yusof's formulation (Equation 4.5) may be written:

$$f_i = \begin{cases} \frac{U_i - u_i^n}{\Delta t} - RHS & \textbf{near } \Gamma_b \\ 0 & \text{elsewhere} \end{cases} \tag{4.6}$$

where U_i represents the interpolated velocity that accounts for the boundary presence.

Interpolation schemes

Fadlun et al. (2000) presents three velocity-interpolation approaches for the direct forcing approach (each illustrated in Figure 4.3):

- No interpolation
- Weighting by cell-volume fraction
- Linear interpolation between the immersed boundary-velocity and nearby fluid velocities

By **no interpolation** is meant that the forcing is determined by imposing the immersed boundary condition on the Eulerian nodes that are closest to the boundary. This way, the IBC is explicitly satisfied, but the representation of the body becomes razterized as shown in Figure 4.3 *(a)*. This method is simple to implement, but requires a high grid-resolution in order to represent the immersed boundary acceptably.

The **cell-fraction interpolation** is done by weighting the forcing term by a measure of the cell volume that is occupied by the solid. The weighted version of f_i is applied on the nearest fluid node.

The reasoning for the **linear interpolation** is that a velocity-profile near a submerged wall will have an approximately linear region, following the *inner law of the wall* (see White (2006)). Thus, as long as the grid-cells are small enough, the linear approximation holds. The longest interpolation-step for linear interpolation on the grid is $2\Delta x_i$, and the following requirement should be satisfied.

$$2\Delta x_i \leq \Delta l_i \tag{4.7}$$

where Δl_i is the linear region which may be estimated by using the law of the wall. Fadlun et al. (2000) concluded that the linear approximation was the most efficient approach.

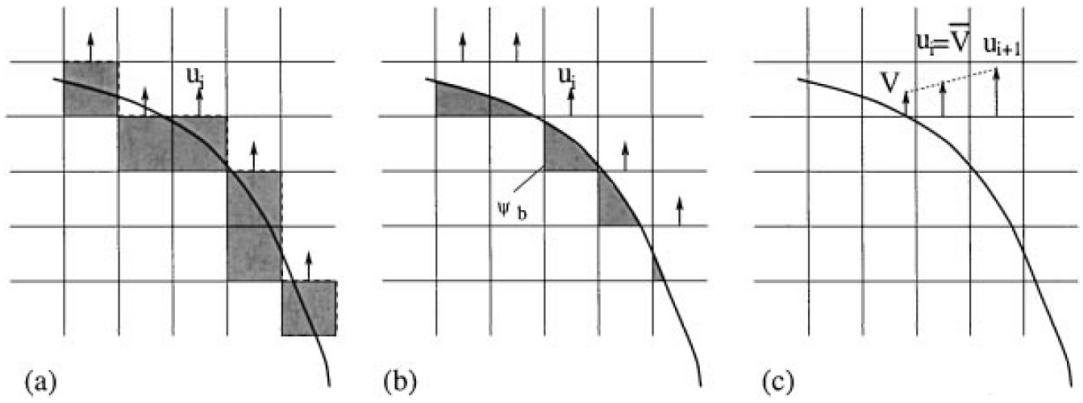


Figure 4.3: The three interpolation procedures discussed in Fadlun et al. (2000). No interpolation (a) gives a rasterized representation of the boundary. Cell-fraction weighting (b) where Ψ_b represent the ratio between fluid and solid in the cell. Linear interpolation (c), here shown as interpolation in the direction normal to the flow velocity. (From Fadlun et al. (2000)).

When to interpolate

A question arises when a direct forcing IBM is combined with a Fractional Step method for pressure-velocity-coupling: *When in the algorithm should the interpolation be done?*

The earlier discrete forcing methods (such as the ones described by Mohd-Yusof (1997) and Fadlun et al. (2000)) interpolates on the divergence-free velocity at time step $n+1$, thus the interpolated cells will not satisfy continuity. Fadlun et al. (2000) also discuss the possibility of performing the interpolation of velocity on the non-solenoidal velocity field ($\tilde{u}_i$) instead of on u_i^{n+1}. This is supported by Balaras (2004) that states that it may be done "without compromising the temporal accuracy of the method"[1]. This way, the pressure correction (δp) is calculated with the information from the interpolated velocity field, and thereby carry over information about the immersed boundary.

From the definition of the direct forcing (Equation 4.6), with interpolation performed on $\tilde{u}_i$:

$$\begin{aligned} \frac{\tilde{u}_i - u_i^n}{\Delta t} &= RHS \qquad \text{everywhere, and} \\ \frac{U_i - u_i^n}{\Delta t} &= RHS + f_i \qquad \text{near } \Gamma_b \end{aligned} \tag{4.8}$$

From this it is be derived that

$$\begin{aligned} U_i &= u_i^n + \Delta t \cdot RHS + \Delta t \cdot f_i \\ &= \tilde{u}_i + \Delta t \cdot f_i \end{aligned} \tag{4.9}$$

which means that the forcing term may be written as:

$$f_i = \frac{U_i - \tilde{u}_i}{\Delta t} \tag{4.10}$$

Figure 4.4 shows that the procedure of interpolation of U_i on the non-solenoidal field

[1] This is only true if the difference between $\tilde{u}_i$ and u_i^{n+1} is small, which is the case for fractional step methods using the pressure field p^n to find $\tilde{u}_i$. If a zero-gradient pressure field is used instead, then the difference between $\tilde{u}_i$ and u_i^{n+1} may be large, so that the force calculated from the interpolation on $\tilde{u}_i$ becomes highly inaccurate.

($\tilde{u}_i$) is the same as adding the forcing term in the equation for the node near the immersed boundary.

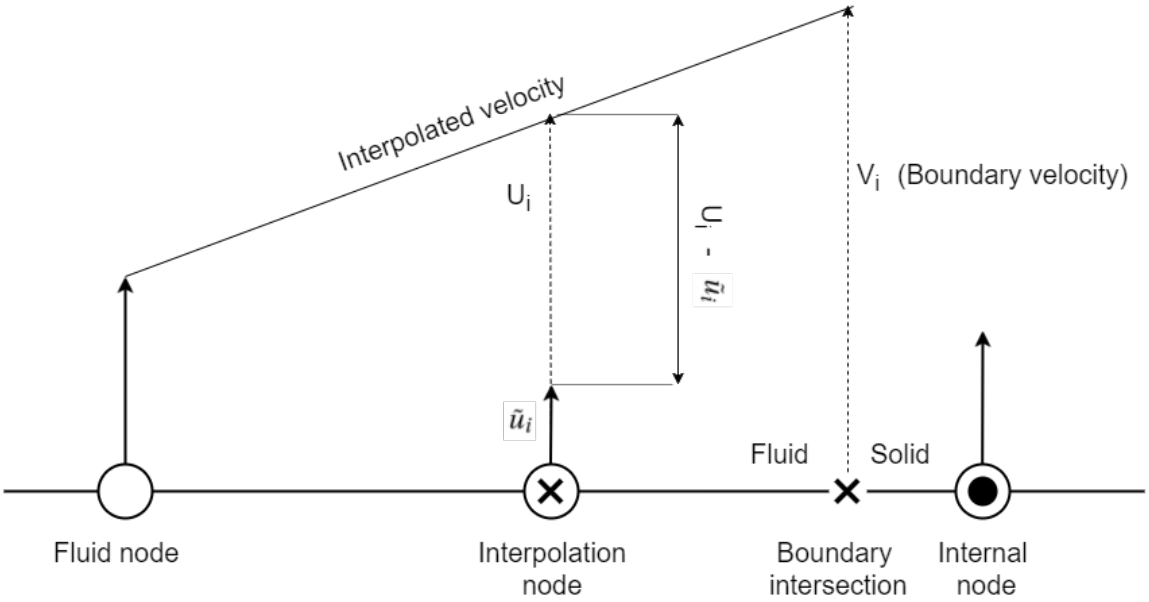


Figure 4.4: Interpolation on the non-solenoidal velocity field. The updated velocity in the interpolation node has the value of U_i, which is an acceleration of $\frac{U_i - \tilde{u}_i}{\Delta t} = f_i$ over the time step.

Pressure condition

As stated in Section 2.3, the immersed boundary condition also requires the normal gradient of the pressure to be zero on the boundary. Through the fractional step method, this requirement modifies to be necessary for the pressure correction:

$$\frac{\partial(\delta p)}{\partial x_n} = 0 \text{ on } \Gamma_b \tag{4.11}$$

Fadlun et al. (2000) observed that, for all their calculations, the pressure correction of the non-solenoidal field ($\tilde{u}_i$) was very small compared to the velocity magnitude itself. Their discussion about this is as follows:

The interpolated velocity field mimics the velocity boundary conditions on Γ_b, that is, the numerical velocity at the node near the boundary should be close to zero. In the following representation of the governing equation (where subscript $(\cdot)_t$ and $(\cdot)_n$ stands for the tangential and normal direction,respectively)

$$\frac{\partial u_n}{\partial t} = -\frac{\partial p}{\partial x_n} + \frac{1}{Re}\left(\frac{\partial^2 u_n}{\partial x_n \partial x_n} + \frac{\partial^2 u_n}{\partial x_t \partial x_t}\right) - \frac{\partial u_n^2}{\partial x_n} - \frac{\partial (u_n u_t)}{\partial x_t} \tag{4.12}$$

it then follows that all terms but the pressure gradient term and the viscous terms are very small. Further, if the velocity interpolation scheme is chosen linear, also the viscous terms will be small, so that eventually the pressure gradient is small:

$$\frac{\partial p}{\partial x_n} << 1 \tag{4.13}$$

Treatment of the body-internal field

The application of the immersed boundary force (f_i) yields a velocity field that satisfies the governing problem requirements *outside the immersed boundary*. Since the immersed boundary is defined as a hollow shape, a flow system will develop on the nodes that are situated inside the boundary as well. Fadlun et al. (2000) discussed three approaches on how to treat this:

- Apply a reverse forcing into the internal flow field
- Do nothing, and let the internal flow field develop unrestrictedly
- Extend the velocity interpolation onto the first body-internal node in addition to the external fluid nodes.

Their conclusion was that the difference was negligible as far as the *direct forcing* method was used. This is later supported by Guy and Hartenstine (2009), who argues that "as discrete time and space are refined, the velocity on the extension will converge to the velocity of the solid body, and so it is not surprising that these approaches gave similar results."

4.2 IBM in swimming fish problems

The immersed boundary methods are indeed designed to be used in fluid-structure problems, and according to Peskin (2002), they are especially useful in biological fluid dynamics. This section discuss the applicability of IBMs in swimming fish problems.

The main advantages of IBMs may be listed as:

- No re-meshing of fluid domain - low computational cost when describing arbitrarily moving objects
- Allows for use of high-quality grids
- Versatility when it comes to boundary shape and complexity
- Object modularity - multiple boundaries may easily be put into the same fluid problem
- Program modularity - an IBM-solver may be developed by adding an IBM-module to an already existing NS-solver

Since the swimming fish problem is characterized by complex boundary shapes and propagating motion, an IBM should be a good choice for treating the problem numerically. The downsides of the IBMs should though be assessed:

- Issues with continuity in the cells near the immersed boundary
- High grid resolution in remote domain parts
- Numerical instabilities of the fluid-structure-interaction system

The last point in the list above is of especial concern. Uhlmann (2005) states that the direct forcing methods, when applied to arbitrarily moving boundaries, are prone to numerical oscillations. This may be overcome by the use of a strong coupling between the fluid solver and structure solver, as discussed in Section 3.8.

4.2.1 Examples of studies

Mittal et al. (2008) developed an FSI-solver with an IBM using ghost-cells for the interpolation. The solver was validated on 2D- and 3D flow problems for Reynolds numbers in the range $O(10^0)$ - $O(10^3)$. For fluid discretization they used a stretched regular rectilinear grid, while, for the discretization of the immersed boundary surface, they used an unstructured grid (illustrated in Figure 4.5).

Their conclusion was that the solver was "highly versatile" for simulations of moving complex boundaries, and they demonstrated this versatility by simulating both the stroke of a pectoral fin on a fish, and the flight of a dragonfly.

Gilmanov and Sotiropoulos (2005) developed an IBM that used a hybrid staggered/co-located grid. The solver was validated on example flows with Reynolds numbers spanning from $O(10^0)$ to $O(10^2)$, before being applied to to simulate two aquatic swimmers in a 3D domain:

- a streamlined mackerel (see Figure 4.6)
- a planktonic copepod

Both studies showed realistic results, and demonstrated the solvers ability to handle problems involving completely different types of aquatic swimmers. The overall conclusion of the paper was that the method was " a powerful numerical simulation tool for a broad range of biofluids applications".

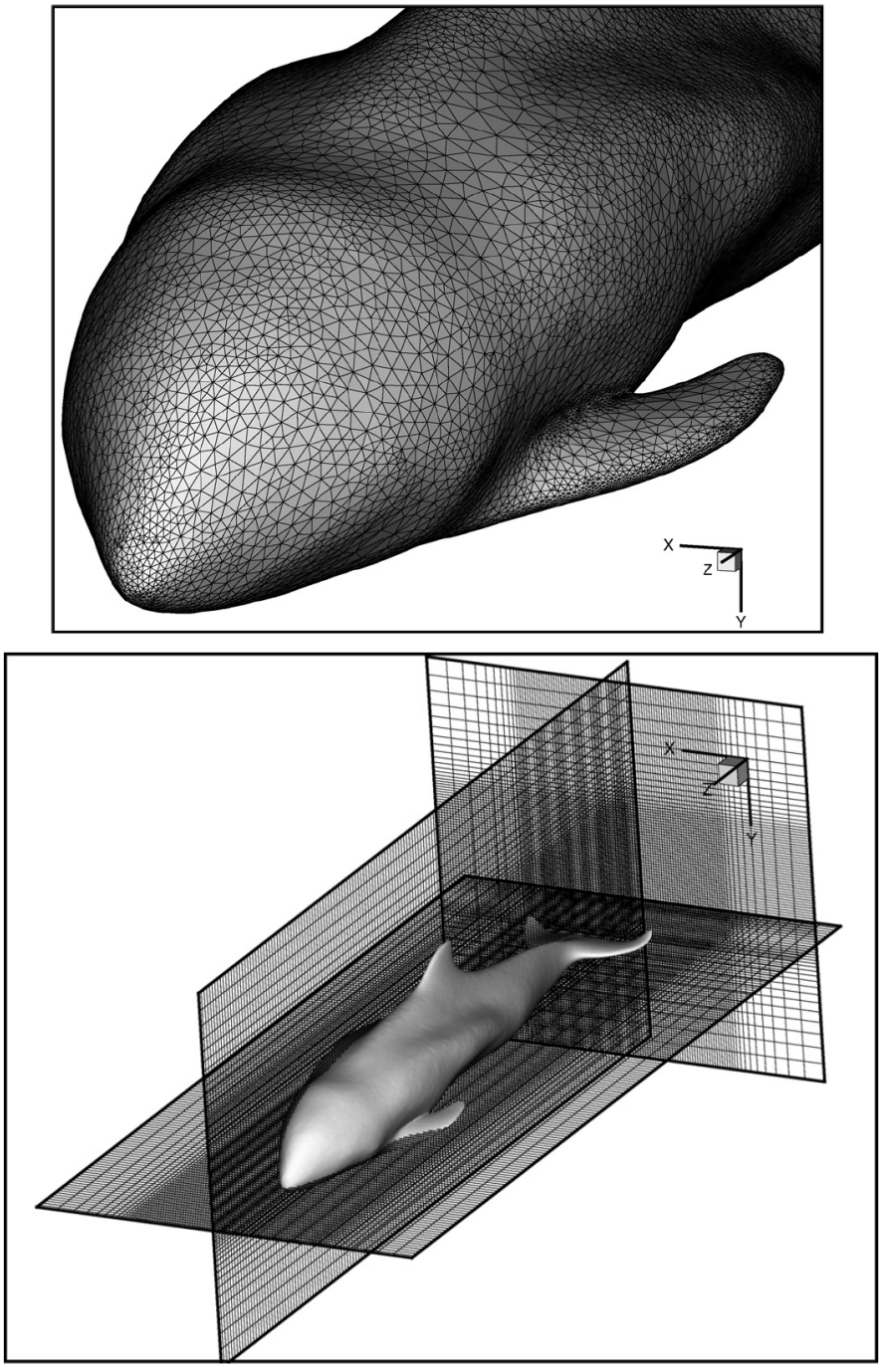


Figure 4.5: Demonstration of the grids used by Mittal et al. (2008). An unstructured grid is used to represent a harbor porpoise (top), for which the geometry is fetched from a CT scan. The immersed boundary represented in the Eulerian fluid grid (bottom). (Modified, from Mittal et al. (2008)).

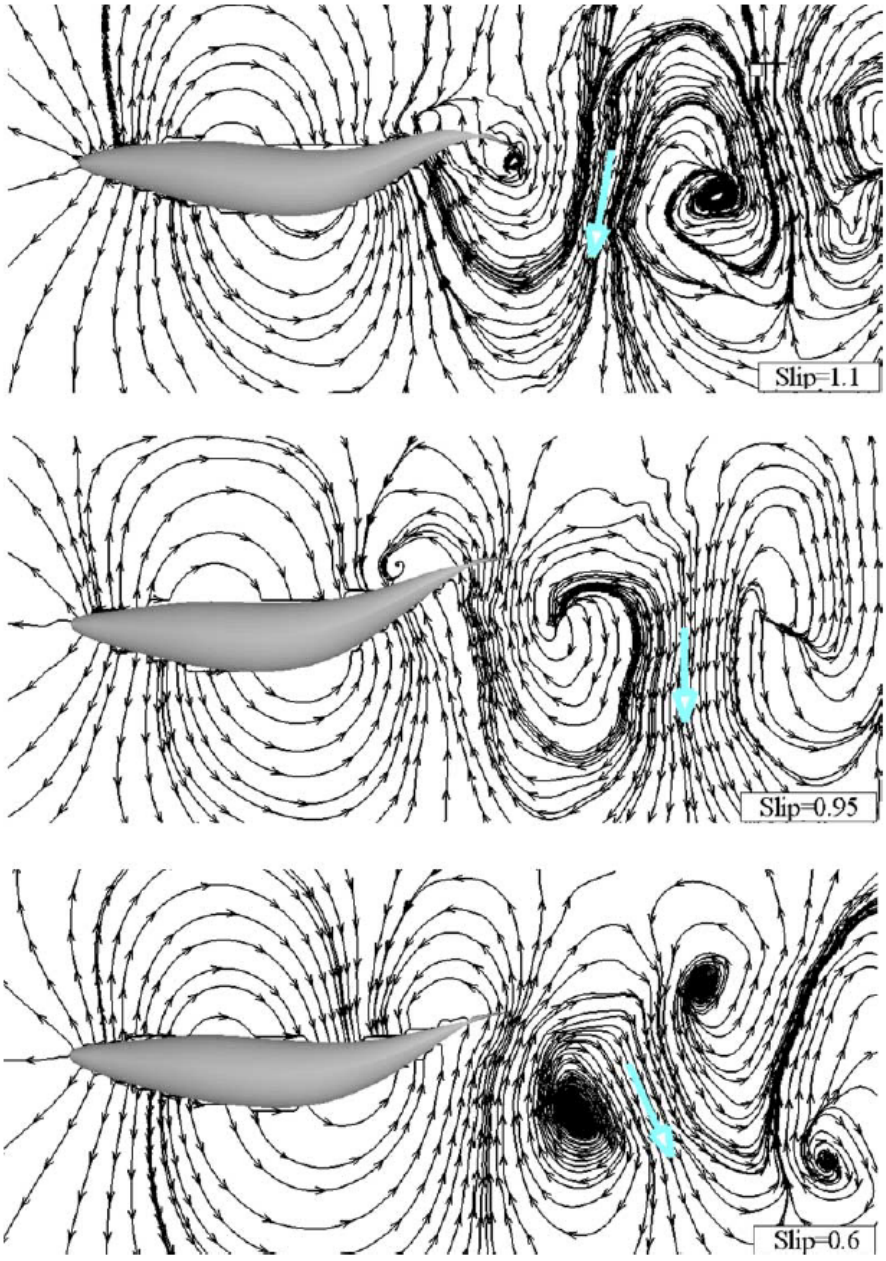


Figure 4.6: Post-processed results from the mackerel study performed by Gilmanov and Sotiropoulos (2005), showing the formation of a reverse Karman street behind the fish for different slip-ratios (the ratio between fish propagation speed and wave speed of the fish undulations) (From Gilmanov and Sotiropoulos (2005)).

Chapter 5

Numerics

As described in Chapter 3, discrete equation sets must be solved in order to solve the governing equations for fluid flow:

The RHS-boundary value problem of the semi-discrete approach:

$$RHS(u_r, p) = -G_r(p) + \frac{1}{Re} L(u_r) - A(u_r) \tag{3.4}$$

The semi-Discrete ODE:

$$\frac{\partial u_r}{\partial t} = RHS(u_r, p) \tag{3.3}$$

The Poisson equation for the pressure correction δp:

$$L(\delta p) = \frac{1}{\Delta t} G_r(\tilde{u}_r) \tag{3.12}$$

This chapter presents some numerical methods used to solve these equation sets. The descriptions are limited to equations sets discretized onto a 2-dimensional structured rectangular grid. The discrete nodal addresses will be shown as $|_{i,j}$. To avoid confusion with the tensor notation, tensor subscripts will be written by the letters r and s, or they will be omitted.

5.1 Approximation of spatial derivatives

A nodal version of Equation 3.4 (tensor notation omitted for readability) may be written as:

$$RHS|_{i,j} = G|_{i,j} + A|_{i,j} + V|_{i,j} \tag{5.1}$$

where

- $G|_{i,j}$ is the pressure gradient term
- $V|_{i,j}$ is the viscous term
- $A|_{i,j}$ is the advective term

each approximated by an arbitrary FDM-scheme at node address i, j. Equation 5.1 yields a value for RHS in every nodal point, which may again be used to solve the semi-discrete ODE (Equation 3.3) in all nodal points.

Along the boundary, the nodal values are determined directly from the boundary conditions for the problem model (see Section 3.3).

5.2 Time integration

The previous section shows how $RHS(u_r, p)$ is represented in each node of the mesh. As $RHS(u_r, p)$ represents the time-gradient of the velocity, a solution of equation 3.3 may be found on each node as well, that is: The field at the next time step is found by integrating the velocity node-wise.

Numerical integration of an ODE as the one in Equation 3.3 or 5.2, is done by approximating the solution some step, h, forward in time.

$$\frac{df}{dt} = RHS(f(t), t) \quad \text{for an arbitrary function,} \quad f(t) \tag{5.2}$$

The methods for solving such equations may be divided into the groups of *explicit methods*, where the solution for the integrated value springs directly from the equation, and *implicit methods* where the solution has to be found through iterations. Of the two groups, the latter one is unconditionally stable, while the first is clearly easier to implement in a code. This section will describe two basic *explicit* time-integration schemes for ODEs: The *Forward Euler method* and the *Explicit two stage Runge-Kutta method* (RK2).

5.2.1 Forward Euler method

The forward Euler method arises from the simplest explicit time discretization for an ODE:

$$\frac{f(t+h) - f(t)}{h} \approx RHS(f(t), t) \tag{5.3}$$

Leading to the numerical integration-scheme:

$$f(t+h) \approx f(t) + h \cdot RHS(f(t), t) \tag{5.4}$$

where h represents the step taken along the dimension t. Performing an Euler integration over a step h, is called "performing an Euler-step of h".

5.2.2 Explicit Runge-Kutta 2 method

The two stage explicit Runge-Kutta method is performed by executing two forward Euler steps consecutively. The first step is a half-step ($h/2$) made to find information about the slope of the solution at $t + h/2$. This information is then used for the second Euler step, which is taken from the same initiation point t, but with the slope estimated at $t + h/2$ by use of the field values predicted by the first step.

Prediction of field values at $h/2$:

$$f(t + h/2) \approx f(t) + \frac{h}{2} RHS(f(t), t) \tag{5.5}$$

Integration of the field variables from t to $t + h$ by using a slope estimated from the field values at $t + h/2$

$$f(t + h) \approx f(t) + h \cdot RHS(f(t + h/2), t + h/2) \tag{5.6}$$

The method of using the slope estimated at $h/2$ increases stability (see Figure 5.1), though it does not enhance the accuracy of the scheme (as shown in Section 5.4).

5.3 Linear solvers

As opposed to Equation 3.4, which contains the non-linear advective term, the Poisson equation for the pressure correction (Equation 3.12) contains only linear terms. It may therefore, by use of an appropriate discrete mapping, be written on the form:

$$A_{rs}x_s = b_r \tag{5.7}$$

For this problem, several solution algorithms exist, and they are separated into two groups: *Direct* solvers and *iterative* solvers. The direct solvers are different algorithms for simply solving the equation exactly, by arriving at the equation:

$$x_s = b_r A_{rs}^{-1} \tag{5.8}$$

The problem with the direct solvers is that they are computer costly for large meshes. As an example, consider a 2-dimensional quadratic domain discretized by a Cartesian grid with M points in each direction. When this mesh is mapped onto a 1-dimensional array, x_s, the length of this array is readily $M \times M$. This means that the number of elements in A_{rs} is $(M \times M) \times (M \times M)$, so the growth of A_{rs} with M is M^4 for a 2-dimensional system. For a 3-dimensional domain, the growth is M^6. This means that a direct method on a 3-dimensional grid of $100 \times 100 \times 100$ points would have to solve a linear system where A_{rs} contains 10^{12} points. Due to this limitation regarding size, direct solvers will not be covered more in this thesis.

Iterative solvers are more versatile when it comes to mesh size, but they do not arrive at an exact solution of Equation 5.7. Instead, they are iterated until some specified criterion is satisfied. A typical criterion is that the difference between the solution of two successive iterations is less than some specified value.

There exist many fast, clever and complex methods for iteratively solving linear equation systems. Examples, such as the *Multigrid method* and *CGstab* are discussed in Ferziger and Péric (2002), but will not be covered here as they fall outside the scope of this thesis.

Due to its simplicity and ease of implementation, the *Gauss-Seidel* method (GS) is cho-

sen to describe an example of an iterative solver. It should be noted, however, that the convergence rate of this solver is poor compared to the ones mentioned above.

5.3.1 Gauss-Seidel method

Consider Equation 3.12 discretized by a 2nd order FDM-scheme. For generality, Φ is chosen to represent the field variable, and the RHS is described by g:

$$\frac{\Phi|_{i+1,j} - 2\,\Phi|_{i,j} + \Phi|_{i-1,j}}{\Delta x^2} + \frac{\Phi|_{i,j+1} - 2\,\Phi|_{i,j} + \Phi|_{i,j-1}}{\Delta y^2} = g\big|_{i,j} \tag{5.9}$$

A rearrangement of this equation may be done to arrive at an expression for $\Phi|_{i,j}$

$$\Phi|_{i,j} = \left(\frac{\Phi|_{i+1,j} + \Phi|_{i-1,j}}{\Delta x^2} + \frac{\Phi|_{i,j+1} + \Phi|_{i,j-1}}{\Delta y^2} - g\big|_{i,j}\right) \Big/ \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right) \tag{5.10}$$

The value of $\Phi|_{i,j}$ is now implicitly given, and the Gauss-Seidel method propagates towards the exact solution by successively iterating over the grid, storing the newly calculated values in the same grid as the one the calculation values are fetched from. This way, the terms at nodal address $(i-1, j)$ and $(i, j-1)$ are evaluated using the value updated in the ongoing iteration step, while terms at nodal address $(i+1, j)$ and $(i, j+1)$ are evaluated using the value from the previous iteration step.

The iteration keeps going until a convergence criterion is reached, typically based on a norm applied on the difference between two successive iterations of Φ:

$$||\,\Phi^{n+1} - \Phi^{n}|| \leq \epsilon_{max} \tag{5.11}$$

where ϵ_{max} is the largest allowed value of the norm.

5.4 Numerical Error

The mathematical definition of a derivative of a function f is given by Equation 5.12:

$$\frac{df}{dt} \triangleq \lim_{h \to 0} \frac{f(t+h) - f(t)}{h} \tag{5.12}$$

Provided that f is a smooth function over the interval $(t, t+h)$, and $h << 1$, the value of $f(t+h)$ may be approximated by a Taylor expansion around $f(t)$:

$$f(t+h) \approx f(t) + h\frac{df}{dt}\bigg|_t + \ldots + \frac{h^{n-1}}{(n-1)!}\frac{d^{n-1}f}{dt^{n-1}}\bigg|_t \tag{5.13}$$

where $n-1$ is the finite length of the Taylor-expansion.

Since h is assumed small, the error of the approximation must have the same order of magnitude as the first term that is omitted, that is $O(\frac{h^n}{n!})$. In numerics it is common to speak of a method as *accurate to the nth order* or as an *nth-order method*, where *n*th reflects the power of h to which the error is proportional. Further, it is common to just write $O(h^n)$ for the order of the accuracy.

As an example, the explicit Euler method and the explicit two-stage Runge-Kutta method (see Section 5.2) are both second-order accurate:

Explicit Euler:

$$f(t+h) = f(t) + h \cdot RHS(f(t), t) + O(h^2) \tag{5.14}$$

Explicit RK2:

$$f(t+h) = f(t) + h \cdot RHS(f(t+h/2), t+h/2) + O(h^2) \tag{5.15}$$

5.5 Numerical stability

Ferziger and Péric (2002) defines a numerical solution as stable if it "does not magnify the errors that appears in the course of numerical solution process".

5.5.1 Linear stability

The stability of non-linear equations such as the NS-equation (Equation 2.7), is difficult to investigate (Ferziger and Péric (2002)), but an analysis of a linearized version of the method may in many cases provide usable information about the stability limits (Lomax et al. (1999)).

If the advective operator $A(\cdot)$ is linearized, a linear version of the semi-discrete NS-equation (Equation 3.3) may be written as

$$\frac{\partial u_r}{\partial t} = M_{rs} u_s - f_r \tag{5.16}$$

where M_{sr} represents a linear operator that combines the linearized advective term and the viscous term, and f_r represents the pressure term. The stability of a such equation is represented by the complex eigenvalues λ of the operator M_{rs} (Lomax et al. (1999)).

Further, application of a numerical time-integration method on the equation may be written on the form

$$u_r^{n+1} = C_{rs} u_s^n - g_r^n \tag{5.17}$$

where C_{rs} is the operator that arise from the application of a time-integration method on Equation 5.16 and similarily, g_r is the term resulting from f_r. The stability of the method is according to Lomax et al. (1999) depending on the criterion

$$|\sigma| \leq 1 \tag{5.18}$$

where σ represents the complex eigenvalue of C_{rs} with the largest absolute value. Further stated in Lomax et al. (1999) is that, if a semi-discrete approach is used, there exist a relation between σ and λ. For the explicit Euler and the RK2-method, these relations

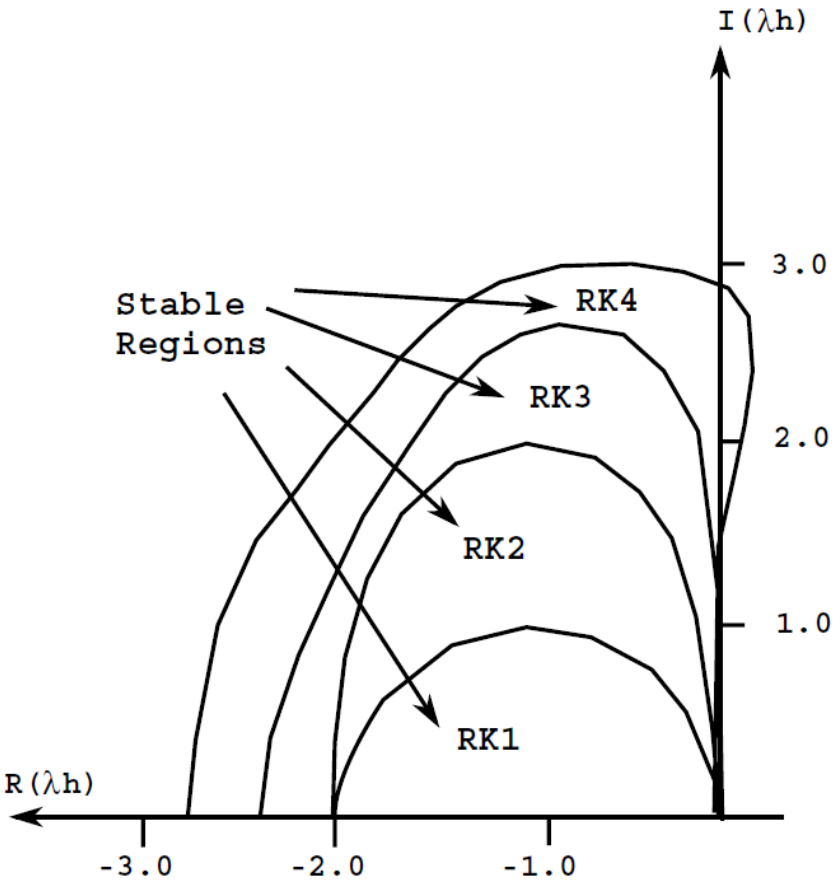


Figure 5.1: The linear stability regions of different explicit Runge-Kutta methods. As seen, the RK2 method has an added stability compared to the Euler explicit method (shown in the figure as RK1). (From Lomax et al. (1999)).

are:

$$\begin{aligned} \sigma &= 1+\lambda h && \text{for the explicit Euler method} \\ \sigma &= 1+\lambda h+\frac{1}{2}\lambda^2 h^2 && \text{for the explicit RK2 method} \end{aligned} \tag{5.19}$$

Figure 5.1 shows how the linear stability criterion (Equation 5.18) may be represented in the complex λh-plane. The definition of linear stability may often serve as guidance when time step and mesh size is to be chosen for a numerical problem, as it indicates how the relation between the system (represented by the system eigenvalues λ) and the method (represented by the method eigenvalues σ) will affect stability.

5.5.2 The Courant number

The Courant number, also called the CFL-number[1] is often used as a stability condition for explicit methods applied on PDEs Ferziger and Péric (2002). The number is defined as:

$$CFL = c_r \frac{\Delta t}{\Delta x_r} \tag{5.20}$$

Where Δt and Δx_r represents the characteristic size of the discrete time step and spatial step, respectively. c_r is the velocity of which information travels in the solution.

The CFL-*condition* for stability is that the value of the CFL number is smaller than some limiting value CFL_{max} that depends on the method used. For explicit schemes it is often taken as $CFL_{max} = 1$ Ferziger and Péric (2002)

The CFL-condition is what's called a *necessary* condition for stability, meaning that stability cannot occur without the condition being satisfied, though stability is not *assured* by the satisfaction of the condition either.

[1] From the name of all of its inventors: Courant, Friedrich and Lewy.

Chapter 6

Development of code

The computer code is developed to solve the incompressible Navier-Stokes equations for 2-dimensional flow (Equation 2.7) on a rectangular domain meshed with a regular rectilinear grid. Further, it can simulate the flow behavior past arbitrary immersed boundaries defined as simple polygons within the domain limits. The polygons may have prescribed motions defined as velocities at each polygon vertex.

```
72
73      !=============================================================================================
74      !===    PRESSURE GRADIENT TERM  ==============================================================
75      !=============================================================================================
76      function MathFDMGradient(d,field,i,j, dir)      !FINDS NUMERICAL PRESSURE GRADIENT IN i,j
77      !TODO(JON): Later, other schemes may also be added, and the user may specify which to use in the UI|
78      !FUNCTION VAR:
79      real(dp)    ::  MathFDMGradient
80      !DUMMY VARS:
81      integer ::  d
82      real(dp)::  field(0:,0:)
83      integer ::  i,j
84      integer ::  dir         !forward (+1) or backward(-1) FDM
85      !=============================================================================================
86      !Finding the pressure gradient
87      if(d.eq.1) then
88          MathFDMGradient = dir*(field(i+dir,j) - field(i,j))/dx(d)
89      elseif(d.eq.2) then
90          MathFDMGradient = dir*(field(i,j+dir) - field(i,j))/dx(d)
91      end if
92      end function MathFDMGradient
93
```

Figure 6.1: Excerpt from the code: FDM gradient function.

6.1 Development tools

The present code was developed and written in Fortran 90, with **Code::Blocks**, an open-source *Integrated Development Environment* (IDE), and an excerpt from the code is presented in Figure 6.1. The program has only been tested and compiled with the **gfortran** compiler, with the following flags and linkers:

- -fcray-pointer
- -static-libgcc

The finished program has only been run on a windows 10 computer with MinGW (Minimalist GNU for Windows) installed.

The calculation results are stored to an output file that is compatible with the Tecplot$_{®}$360 EX$^{™}$ post-processing software.

6.2 Notation

As this chapter is devoted to describe the computational implementation of theory, tensor notation may induce more confusion than good. Table 6.1 and 6.2 present the notation used in the following sections.

Table 6.1: Notation used for description of the N-S solver.

Symbol	Meaning
N_x	Number of discrete points in x-direction
N_y	Number of discrete points in y-direction
i, j	Horizontal and vertical nodal address, respectively
$(\cdot)\|_{i,j}$	Quantity $(\cdot)$ evaluated at node (i, j)
$pos\left((\cdot)\|_{i,j}\right)$	Position of nodal address (i, j) for quantity $(\cdot)$ in the staggered configuration
d	Direction (x: $d = 1$, y: $d = 2$)
$dx(d)$	Mesh cell size in d-direction
u, v	Fluid horizontal and vertical velocity, respectively
RK_{step}	Indicating the propagation length of the respective Runge-Kutta integration step
RK_{dt}	Length of th time step for the respective Runge-Kutta call
Φ_D	Difference between two successive iterations on Φ
$L_\infty(\cdot)$	The infinity norm applied on $(\cdot)$

Table 6.2: Notation used for description of the IBM-module.

Symbol	Meaning
$\lceil(\cdot)\rceil$	*Ceiling*-function - rounds $(\cdot)$ **up** to nearest integer
$\lfloor(\cdot)\rfloor$	*Floor*-function - rounds $(\cdot)$ **down** to nearest integer
$\mathbf{b}$	Array containing information of the polygon constituting the immersed boundary
$\mathbf{b}_{comp}$	Polygon coordinates of $\mathbf{b}$ mapped to the computational domain
$\mathbf{b}_{dir}$	Array describing the directional vectors of each polygon segment of $\mathbf{b}$
$\mathbf{b}_{norm}$	Array describing the normal vectors of each polygon segment of $\mathbf{b}$
$\mathbf{b}_{\times}$	Array carrying information about number of intersections on each polygon segment
N_b	Number of vertexes contained in $\mathbf{b}$
c	Denotes the vertex or segment number of interest
$X(c,d)$	Position in direction d of vertex c
$V(c,d)$	Immersed boundary velocity at vertex c in d-direction
$x_{\times}$	x-coordinate of intersection, $\times$
$y_{\times}$	y-coordinate of intersection, $\times$
$\odot$	Denotes the body-internal node
$\times$	Denotes the intersection number of interest
$\otimes$	Denotes the integer address of the interpolation node
$\circ$	Denotes the integer address of the fluid node next to the interpolation node
$N_{\times_d}$	Number of intersections in d-direction
V_d	Immersed boundary velocity in d-direction at intersection
U_d	Interpolated fluid velocity in d-direction
f_d	Immersed boundary forcing term in d-direction
F_d	Total force in d-direction from fluid to immersed boundary

6.3 The Navier-Stokes solver

The Navier-Stokes solver was constructed by use of:

- a *staggered grid* equipped with the Finite Difference stencils described in Harlow and Welch (1965) for the spatial discretization (see Section 3.2 and 3.3)
- the explicit two stage Runge Kutta method (RK2) for the time integration (described in Section 5.2).
- the Fractional Step Method (FSM) for the pressure-velocity coupling (described in Section 3.6).
- the Gauss-Seidel method (described in Section 5.3) for solving the Poisson Equation resulting from the fractional step method.

A flow chart describing the N-S solver is presented in Figure 6.2.

6.3.1 Grid generation

Figure 6.3a shows how the relation between the physical and computational node addresses is defined in the code, and Figure 6.3b shows how the computational cell is related to the field quantities. Defined this way, all equations may be described without the need of half-integer addresses, so that the equations presented are directly applicable to a computer code.

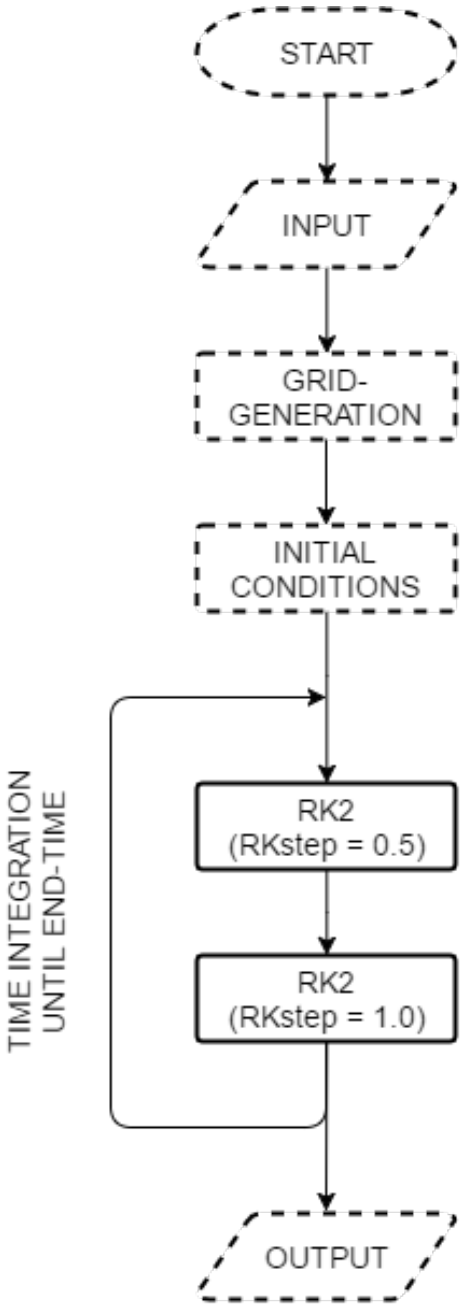
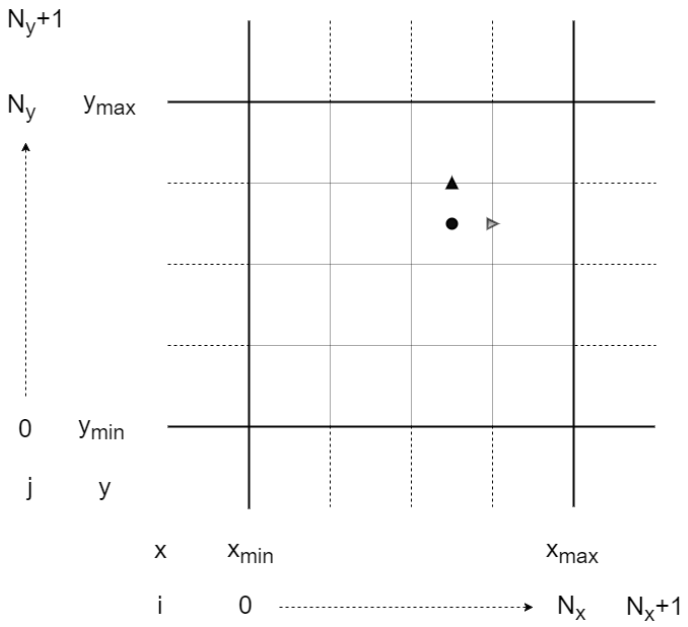
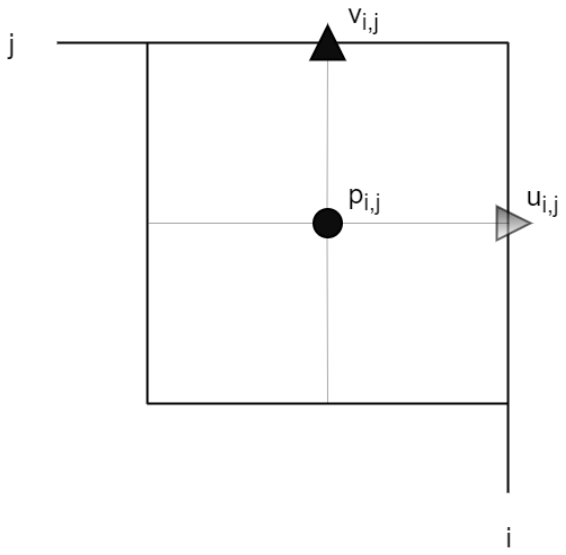


Figure 6.2: Flow chart describing the outline of the N-S solver. See Figure 6.7 for further details about the RK2-subroutine.



(a) The relation between physical (x, y) and computational (i, j) nodal addresses.



(b) The computational cell and the related cell quantities.

Figure 6.3: The definition of the computational grid used for the program. The cell address is defined by its rightmost and uppermost boundary.

6.3.2 Finite difference scheme

The semi-discrete approach is done by first solving for the spatially differentiated terms of equation 5.1. For this purpose, a subroutine called FDM was made. It calculates the values of the difference operators at all positions in the discrete domain, and stores the field of RHS-values in a temporary array.

Together with the address-definition showed in Figure 6.3b, and with the subscript $(|_{pos(\cdot)})$ denoting "evaluated at the position where the quantity $(\cdot)$ is stored", the FDM-stencils from Harlow and Welch (1965) may be described as follows (see also Figure 6.4, 6.5 and 6.6):

Gradient

$$G_x(p)|_{pos(u_{i,j})} = \frac{p|_{i+1,j} - p|_{i,j}}{\Delta x} \tag{6.1}$$

$$G_y(p)|_{pos(v_{i,j})} = \frac{p|_{i,j+1} - p|_{i,j}}{\Delta y} \tag{6.2}$$

Laplacian

$$L(u)|_{pos(u_{i,j})} = \frac{u|_{i+1,j} + u|_{i-1,j} - 2u|_{i,j}}{\Delta x^2} + \frac{u|_{i,j+1} + u|_{i,j-1} - 2u|_{i,j}}{\Delta y^2} \tag{6.3}$$

$$L(v)|_{pos(v_{i,j})} = \frac{v|_{i+1,j} + v|_{i-1,j} - 2v|_{i,j}}{\Delta x^2} + \frac{v|_{i,j+1} + v|_{i,j-1} - 2v|_{i,j}}{\Delta y^2} \tag{6.4}$$

Advection

$$A(u)|_{pos(u_{i,j})} = 0.25\Bigg(\frac{\big(u|_{i-1,j}+u|_{i,j}\big)^2-\big(u|_{i,j}+u|_{i+1,j}\big)^2}{\Delta x} + \frac{\big(u|_{i,j-1}+u|_{i,j}\big)\big(v|_{i,j-1}+v|_{i+1,j-1}\big)-\big(u|_{i,j}+u|_{i,j+1}\big)\big(v|_{i,j}+v|_{i+1,j}\big)}{\Delta y}\Bigg) \tag{6.5}$$

$$A(v)|_{pos(v_{i,j})} = 0.25\Bigg(\frac{\big(u|_{i-1,j}+u|_{i-1,j+1}\big)\big(v|_{i-1,j}+v|_{i,j}\big)-\big(u|_{i,j}+u|_{i,j+1}\big)\big(v|_{i,j}+v|_{i+1,j}\big)}{\Delta x} + \frac{\big(v|_{i,j-1}+v|_{i,j}\big)^2-\big(v|_{i,j}+v|_{i,j+1}\big)^2}{\Delta y}\Bigg) \tag{6.6}$$

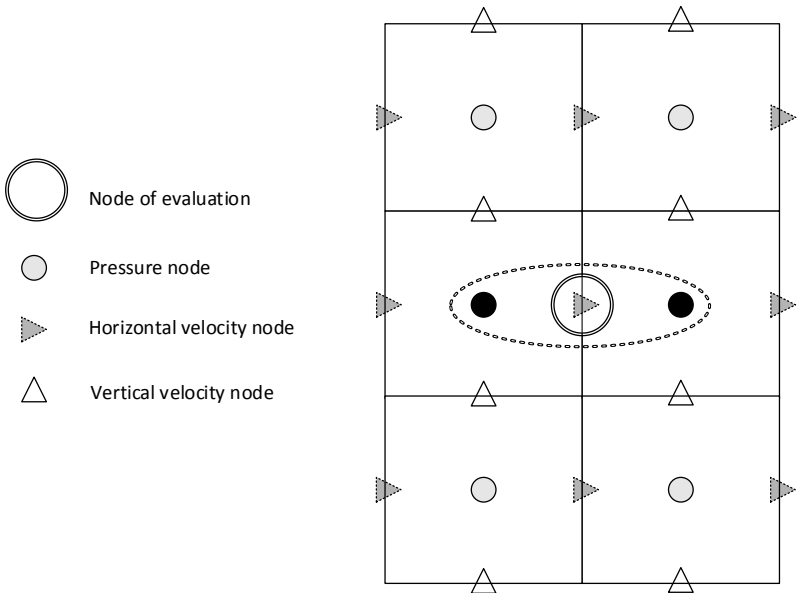


Figure 6.4: The finite difference stencil for the gradient difference operator on the pressure evaluated at $pos(u|_{i,j})$. Black nodes are the ones used in the corresponding equation (Equation 6.1).

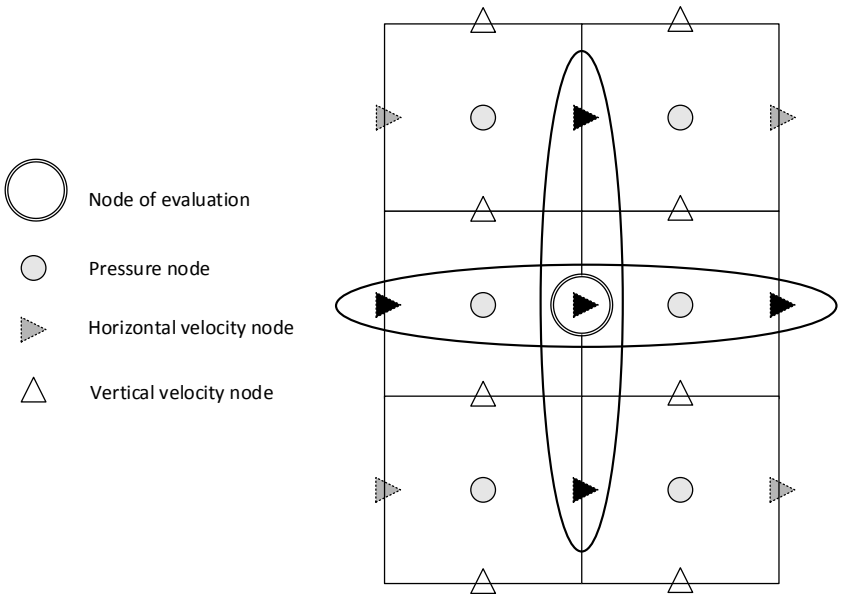


Figure 6.5: The finite difference stencil for the Laplacian difference operator evaluated at $pos(u|_{i,j})$. Black nodes are the ones used in the corresponding equation (Equation 6.3).

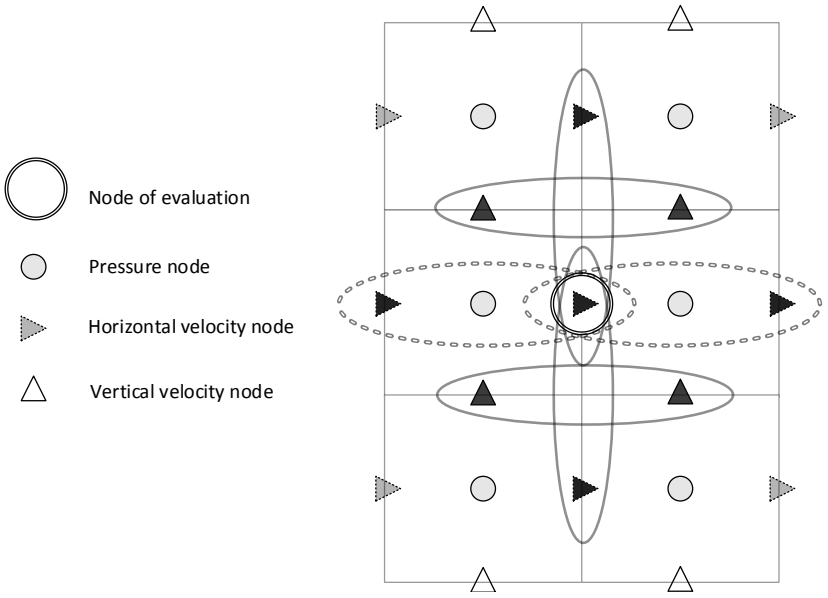


Figure 6.6: The finite difference stencil for the advective difference operator evaluated at $pos(u|_{i,j})$. Black nodes are the ones used in the corresponding equation (Equation 6.5).

6.3.3 Time integration and pressure-velocity coupling

As seen in Figure 6.2 the RK-routine is called twice every time step. First with the step-parameter $RK_{step} = 0.5$, then with $RK_{step} = 1.0$. The time step used within the routine is defined as: $RK_{dt} = RK_{step} \cdot \Delta t$.

In first call to the subroutine the flow fields $(u, v)^n$ and are used both as initiation point and as estimation point for the integration (see Section 5.2). The second iteration, uses $(u, v)^n$ only for initiation, and uses the flow field $(u, v)^{n+0.5}$ for estimation of the slope.

Following the semi-discrete approach described in Section 3.5, the RK2-routine serves as an outer shell for the solution algorithm, integrating the value of the discretized RHS of the semi-discrete equation (Equation 3.3). As discussed in Section 3.6, the incompressibility of the fluid poses a need for a pressure-velocity coupling scheme, which has to be combined with the time integration scheme.

Figure 6.7 shows schematically how the time integration method (RK2) and the pressure-velocity coupling method (FSM) are woven together.

First, the FSM-predictor routine solves for the non-solenoidal velocity fields $\tilde{u}$ and $\tilde{v}$. In the Poisson solver, these fields are used to find the RHS of the Poisson equation (Equation 3.12), before the pressure correction (δp) is found by a Gauss Seidel iteration over the domain. From the pressure correction, the updated pressure $p^{n+RK_{step}}$ is found. Lastly, the pressure correction is used to update $\tilde{u}$ and $\tilde{v}$ to hold for continuity, thereby finding $u^{n+RK_{step}}$ and $v^{n+RK_{step}}$.

Below follows a description of the operations carried out by each step of the RK2-FSM-integrator.

FSM predictor

This subroutine solves the equations

$$\begin{aligned} \tilde{u} &= u^n + RK_{dt} \cdot FDM(u^{n+r}, v^{n+r}, p^{n+r}) \\ \tilde{v} &= v^n + RK_{dt} \cdot FDM(u^{n+r}, v^{n+r}, p^{n+r}) \end{aligned} \tag{6.7}$$

where $r = RK_{step} - 0.5$ is the step for which the slope estimation finds place.

Poisson solver

After the two non-solenoidal fields ($\tilde{u}$) and ($\tilde{v}$) are found, the Poisson equation is solved iteratively by the following algorithm

1. find the RHS of the Poisson equation (Equation 3.10) at all nodes (i, j):

$$g|_{ij} = \frac{1}{RK_{dt}} \left(\frac{\tilde{u}|_{i,j} - \tilde{u}|_{i-1,j}}{\Delta x} + \frac{\tilde{v}|_{i,j} - \tilde{v}|_{i,j-1}}{\Delta y} \right) \tag{6.8}$$

2. Iterate Equation 5.10 (described in Section 5.3) with $\Phi = \delta p$ over the domain $i = 1, ..., N_x$ and $j = 1, ..., N_y$ until the convergence criterion is reached.

The chosen convergence criterion is that the L_∞-norm of the difference Φ_D between the Φ-fields of two successive iterations is less than some said value, e.g. $L_\infty(\Phi_D) \leq 10^{-6}$. The L_∞-norm of the field (Φ_D)) is defined as:

$$L_\infty(\Phi_D) = \max(|\Phi_D|) \tag{6.9}$$

FSM corrector

The fractional step correction is done by simply updating the non-solenoidal velocity with the pressure-correction gradient (see Equation 3.9):

$$\begin{aligned} u^{n+RK_{step}} &= \tilde{u} + RK_{dt} \frac{\delta p|_{i+1,j} - \delta p|_{i,j}}{\Delta x} \\ v^{n+RK_{step}} &= \tilde{v} + RK_{dt} \frac{\delta p|_{i,j+1} - \delta p|_{i,j}}{\Delta y} \end{aligned} \tag{6.10}$$

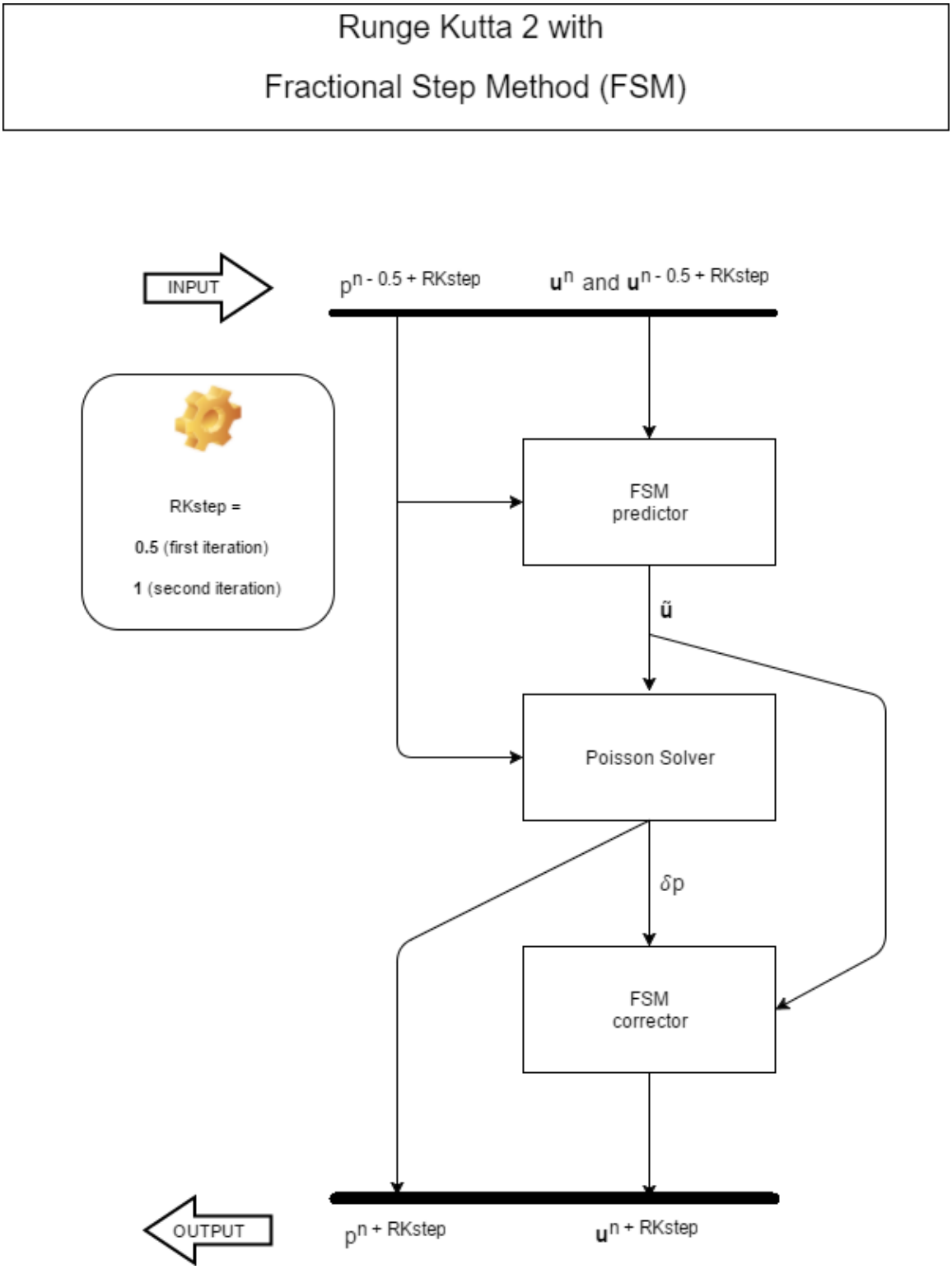


Figure 6.7: Flow chart describing an RK2 subroutine with the fractional step method implemented.

6.4 The IBM module

The general task of any IBM-module is to calculate the immersed boundary forcing term (f_i) of Equation 4.2. To do so, the module need information about

- the underlying grid points and the field values represented there
- the position of the immersed boundary relative to the underlying grid

Due to the shortage of time available for this thesis, *ease of implementation* was of high priority when designing the method. It was therefore decided to implement a **direct forcing** method such as the one described in Fadlun et al. (2000), with

- *linear* interpolation of the velocity field near the immersed boundary
- Interpolation performed on the non-solenoidal velocity fields $\tilde{u}$ and $\tilde{v}$
- **no** explicitly imposed pressure condition on the immersed boundary
- unrestricted internal flow field development

See Subsection 4.1.2 for details about the list above.

To apply the velocity boundary conditions and find the forcing term, the following algorithm was used:

1. take in the velocity fields and the immersed boundary coordinates
2. detect the intersection between the immersed boundary and the computational grid lines related to the velocity field of consideration
3. modify the velocity field by linearly interpolate the velocities such that the IBC is satisfied in the intersection point
4. calculate the magnitude of the forcing term

Figure 6.8 shows the notation used for the fluid nodes that are involved in the interpolation schemes of the IBM-module.

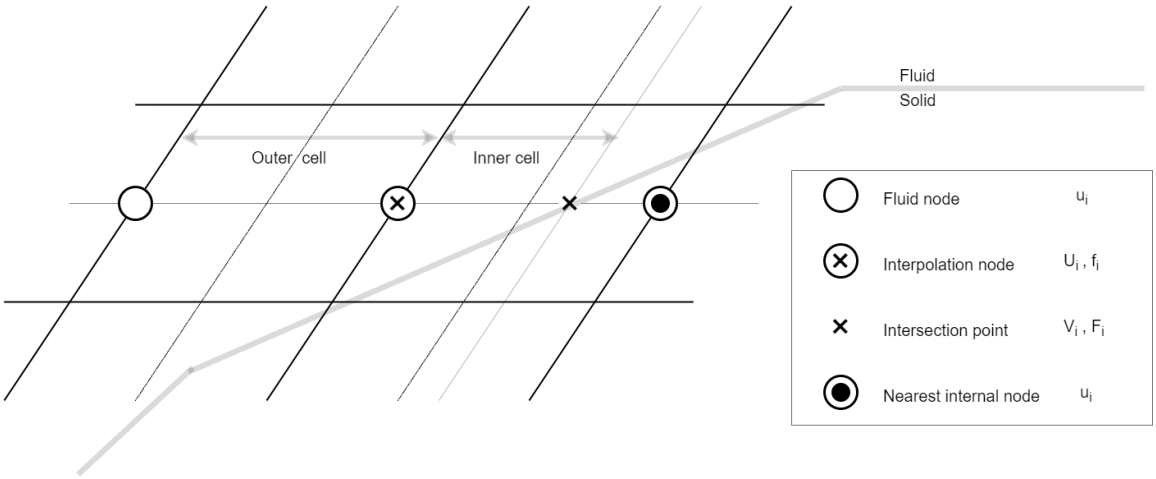


Figure 6.8: Notation and terminology for the nodes used in the IBM-module. The cell between the fluid node and the interpolation node is denoted the *outer cell*, while the cell that is divided by the immersed boundary is denoted is *inner cell*. The figure legend shows which quantities that are connected to which nodes.

6.4.1 Defining the immersed boundary

The immersed boundary is defined as a *simple polygon*[1], with both position (x_c, y_c) and velocities $(V_{x,c}, V_{y,c})$ defined at each vertex (c). The velocity at some intersection point between two vertexes may thereby be found through linear interpolation onto the intersection point. The vertex positions are counterclockwise defined in terms of the physical domain coordinates. Equation 6.11 shows how the polygon vertexes are represented by an array in the computer code.

$$\mathbf{b} = \begin{bmatrix} x_1 & \cdots & x_c & \cdots & x_{N_b} \\ y_1 & \cdots & y_c & \cdots & y_{N_b} \\ V_{x,1} & \cdots & V_{x,c} & \cdots & V_{x,N_b} \\ V_{y,1} & \cdots & V_{y,c} & \cdots & V_{y,N_b} \end{bmatrix} \tag{6.11}$$

[1] Polygon with no crossing lines

From this array, the position of vertex c in direction d may be fetched as

$$X(c,d) \quad = \quad \mathbf{b}(c,d) \tag{6.12}$$

while the vertex velocities in the same direction are found as

$$V(c,d) \quad = \quad \mathbf{b}(c,d+2) \tag{6.13}$$

where d represents the direction.

6.4.2 Detection of intersections

The detection of intersection is initiated by mapping the body-coordinates of the immersed boundary onto a grid where each line is represented by integers. This allows for the use of integer functions such as nint(·), floor(·) and ceiling(·), which can be used to find the nearest grid-lines to any vertex.

Computational arrays in integer grid domain

The following transformation represents the immersed boundary in a coordinate system where each grid-line is represented by an integer

$$\mathbf{b}_{comp}(c,d) \quad = \quad \frac{X(c,d) - x_{min}(d)}{dx(d)} \tag{6.14}$$

An array containing the directional vector for all polygon segments may be constructed as

$$\mathbf{b}_{dir}(c,d) \quad = \quad \mathbf{b}_{comp}(c+1,d) \quad - \quad \mathbf{b}_{comp}(c,d) \tag{6.15}$$

and since the polygon is defined counterclockwise, a normal vector pointing out to the fluid may be defined in all segments as:

$$\mathbf{b}_{norm} \quad = \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{b}_{dir} \tag{6.16}$$

The direction of each successive intersection on a single segment in the d-direction is given by

$$\mathrm{dir}(c,d) = sgn\left(\mathbf{b}_{dir}(c,d)\right) \tag{6.17}$$

With the representation of Equation 6.14, the following array (where $\lceil\cdot\rceil$ denotes the ceiling function)

$$\mathbf{b}_{\times}(c,d) \quad = \quad \lceil \quad \mathbf{b}_{comp}(c+1,d) \quad \rceil \quad - \quad \lceil \quad \mathbf{b}_{comp}(c,d) \quad \rceil \tag{6.18}$$

will contain information about how many integer lines that are situated between vertex c and $c+1$ along direction d. Since the polygon is represented by linear segments, the slope between the segments may thereby be used to find the intersection of the polygon on each of the integer lines between vertex c and $c+1$. As an example the intersections $(x_{\times}, y_{\times})$ between the boundary and the vertical lines in Figure 6.9 can be found as:

$$\begin{aligned} x_{\times} &= \mathrm{nint}\Big(\quad \mathbf{b}_{comp}(c,1) \quad + \quad \mathrm{dir}(c,1)\left((\times-1)+0.5\right) \quad \Big) \\ y_{\times} &= \mathbf{b}_{comp}(c,2) \quad + \quad \frac{\mathbf{b}_{dir}(c,2)}{\mathbf{b}_{dir}(c,1)} \quad \left|x_{\times} - \mathbf{b}_{comp}(c,1)\right| \end{aligned} \tag{6.19}$$

for $\times = 1,2,3,4$.

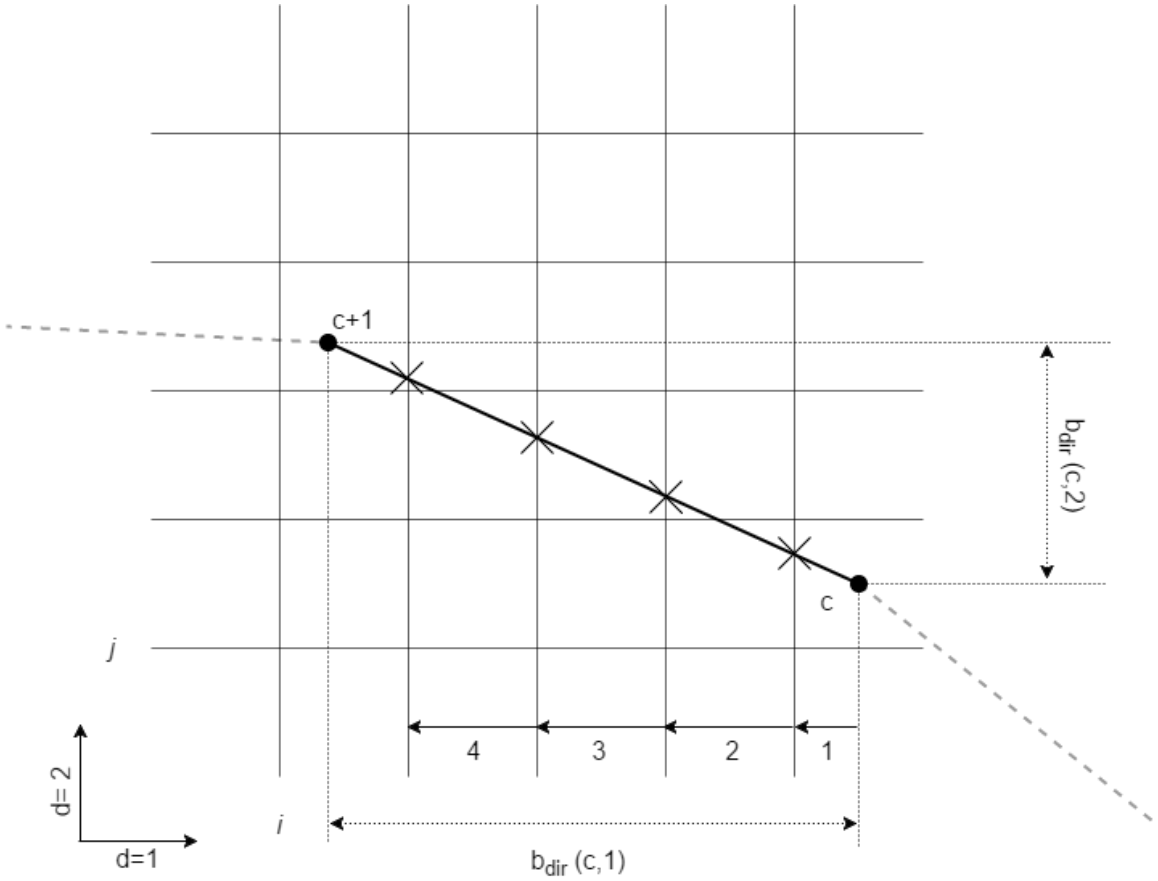


Figure 6.9: In this case, the intersection array will be $\mathbf{b}_{\times}(c,:) = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$, meaning that, from vertex c to $c+1$, there are 4 intersections along the negative x-direction and 2 intersections along the positive y-direction. Since both the position of vertex c and the length of the segment in both directions are known, the position of the intersections at each integer line may be found.

Interpolation direction and grid-shifting

Due to the staggered grid configuration, the detection of intersections is dependent of the choice of interpolation direction. There are two possibilities:

1. interpolate in direction **normal** to the direction of the velocity field (see Figure 6.13)

2. interpolate in direction **parallel** to the direction of the velocity field (see Figure 6.12)

As shown in Figure 6.10 and 6.11, the intersections found at the grid faces will be usable for the normal interpolation scheme, while the parallel interpolation must be done along the cell center lines. To detect the intersections of the latter kind, $\mathbf{b}_{comp}$ may simply be shifted by a value of 0.5 upwards and rightwards prior to establishment of $\mathbf{b}_{\times}$.

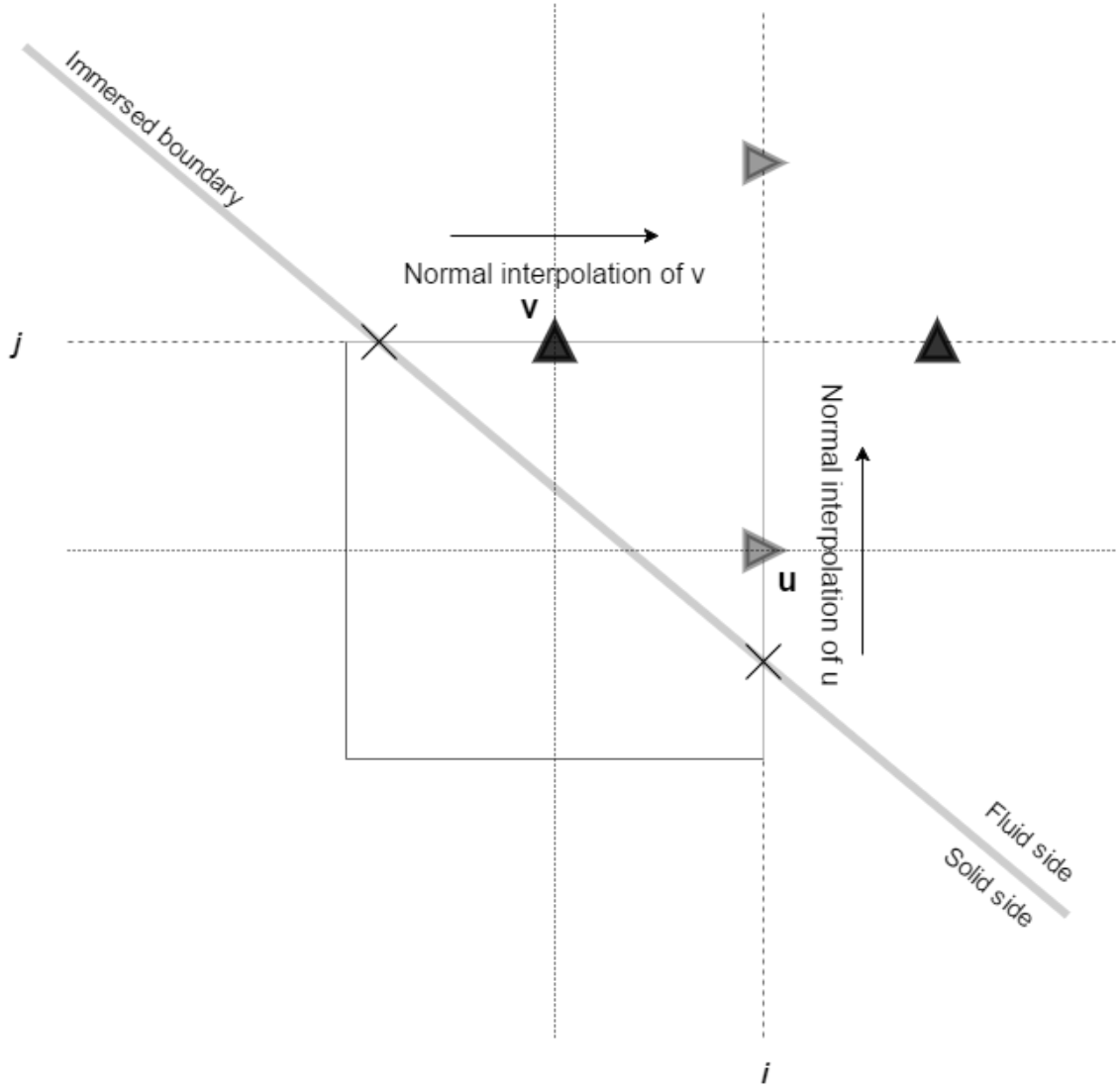


Figure 6.10: Intersection detection for normal interpolation.

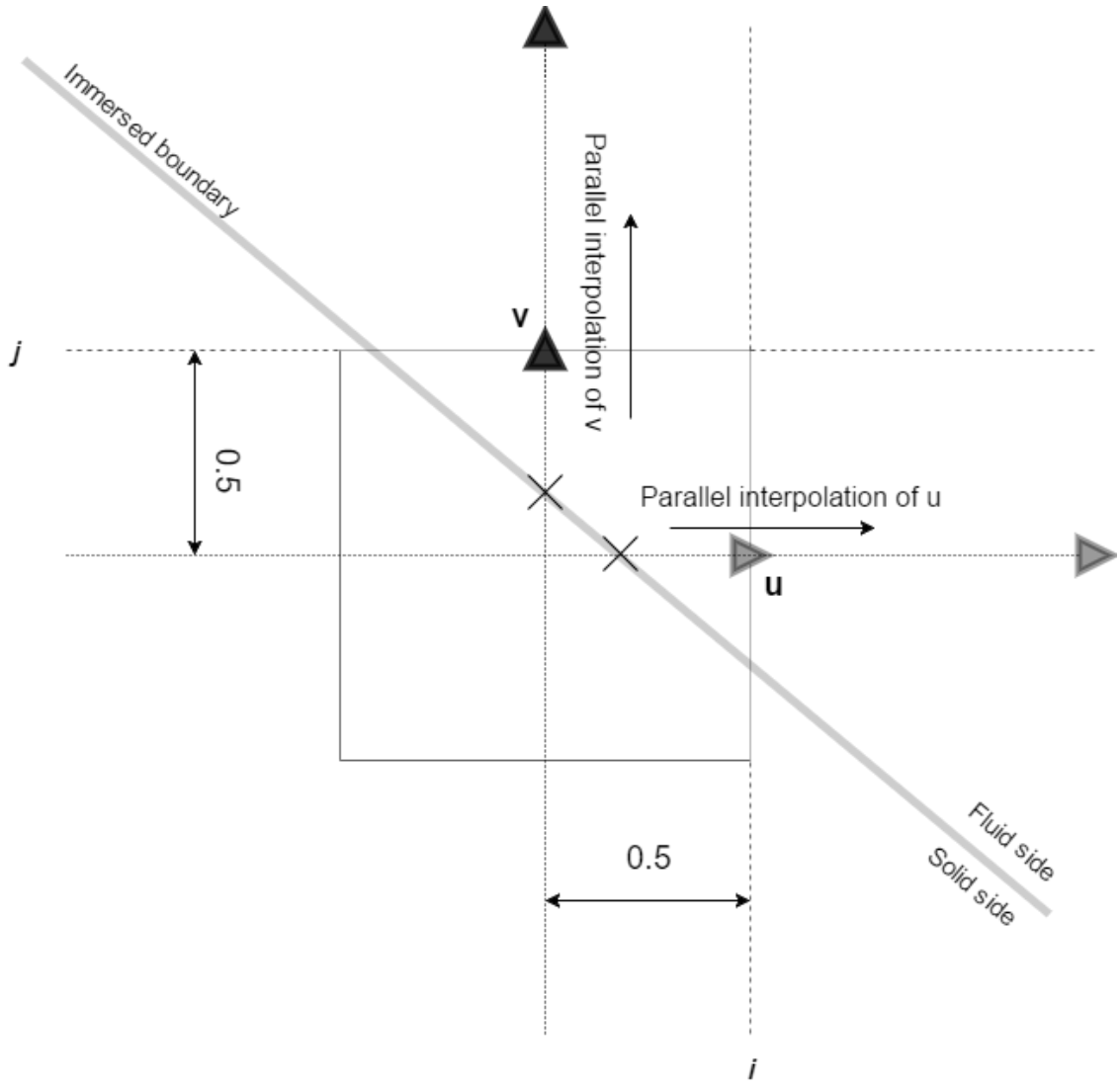


Figure 6.11: Intersection detection for parallel interpolation.

6.4.3 Finding the immersed boundary velocity

Once the intersection points are found, the velocity of the immersed boundary at the intersection point is evaluated from linear interpolation between the two vertexes of the boundary segment. The immersed boundary velocity in the d-direction at the intersection may thereby be found as:

$$V_d|_\times = V(c,d) + \frac{V(c+1,d) - V(c,d)}{l_{seg}} l_\times \tag{6.20}$$

where l_{seg} is the segment length, and $l_\times$ is the distance between vertex c and the intersection point.

6.4.4 Imposing the immersed boundary conditions on the fluid grid

The imposing of the immersed boundary condition is done by manipulating the velocity at the nearest fluid node ($\otimes$) by linear interpolation between the intersection point ($\times$) and the second nearest fluid node ($\circ$). The updated velocity in $\otimes$ becomes:

$$U_d|_\otimes = V_d|_\times + \frac{(\cdot)_d|_\circ - V_d|_\times}{\delta(\circ,\times)} \delta(\otimes,\times) \tag{6.21}$$

Where δ represents the Eucledian distance and

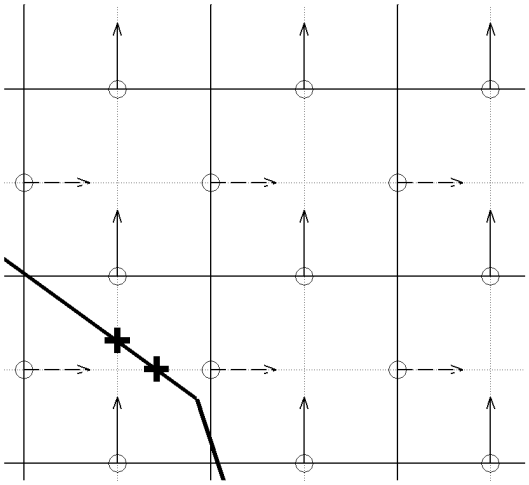
$$(\cdot)_d = \begin{cases} \tilde{u} & \text{if } d = 1 \\ \tilde{v} & \text{if } d = 2 \end{cases} \tag{6.22}$$

The interpolation updates the velocity over the time step RK_{step}, and results in the acceleration along the velocity direction that in earlier chapters was named the *forcing term*:

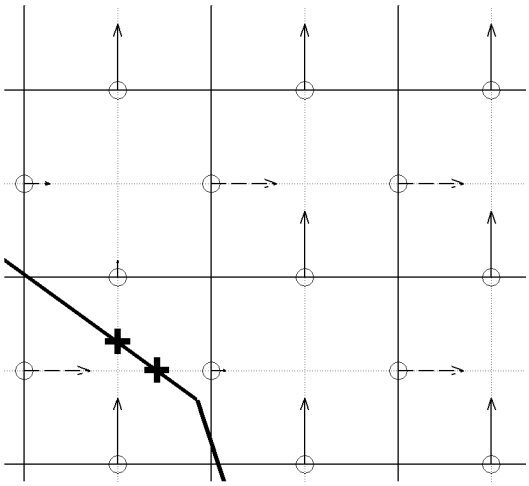
$$f_d|_\otimes = \frac{U_d|_\otimes - (\cdot)_d|_\otimes}{RK_{dt}} \tag{6.23}$$

Figure 6.13 and 6.12 shows normal and parallel interpolation, respectively, done on a

stationary immersed boundary.

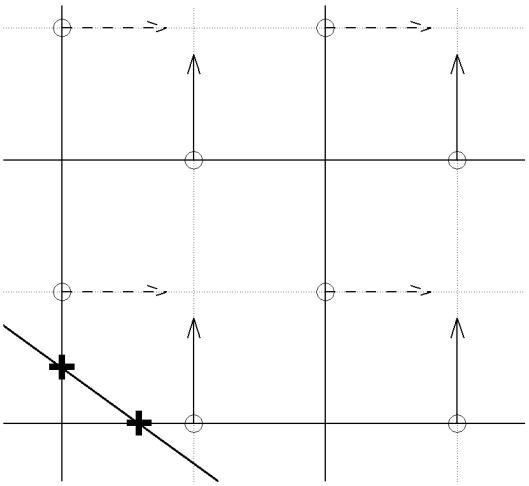


(a) Before parallel interpolation.

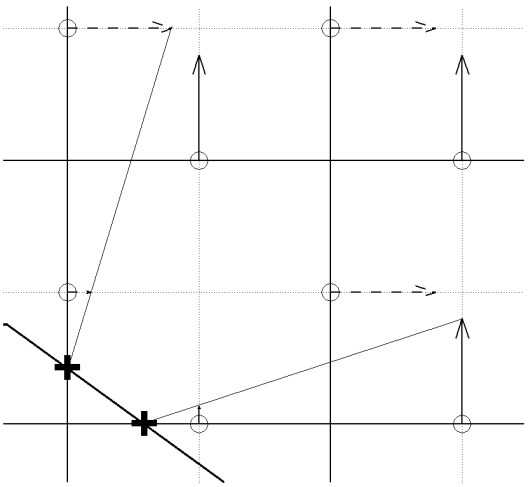


(b) After parallel interpolation.

Figure 6.12: Application of a Dirichlet boundary condition with $u_i = 0$ on Γ_b done by interpolation in the direction **parallel** to the velocity components.



(a) Before normal interpolation.



(b) After normal interpolation.

Figure 6.13: Application of a Dirichlet boundary condition with $u_i = 0$ on Γ_b done by interpolation in the direction **normal** to the velocity components.

6.4.5 Force calculation

The force experienced by the immersed boundary should by Newton's third law be equal, but oppositely directed, to the net force applied to the fluid by the boundary. This means that the drag and lift forces may be found directly from the forcing term, f_i, and the mass of the fluid affected by the acceleration.

Force contribution from each intersection

For a given direction, d, the force contribution from each intersection may be found as

$$dF_d|_\times = -f_d|_\otimes dM|_\otimes \tag{6.24}$$

where $dM|_\otimes$ is the mass of the fluid that is accelerated due to the acceleration in the interpolation node, $\otimes$. In the present code, the force is approximated by using the mass of the fluid in the two cells connected by the velocity node, and using the cell centered velocity in the cells as shown in Figure 6.14. The cell centered acceleration is half the magnitude of the acceleration of the interpolation node, while there are two affected cells, so that the force estimate may be simply be written as:

$$dF_d|_\times = -f_d|_\otimes \rho \, dx \, dy \tag{6.25}$$

where the mass, $dM_d|_\otimes$ is approximated by the area of the two cells connected by $\otimes$.

It should be noted that the present approximation is rather coarse, as it assumes that the immersed boundary only changes the velocity in the discrete location of the interpolation node. A more accurate force approximation will be discussed in Section 9.1.

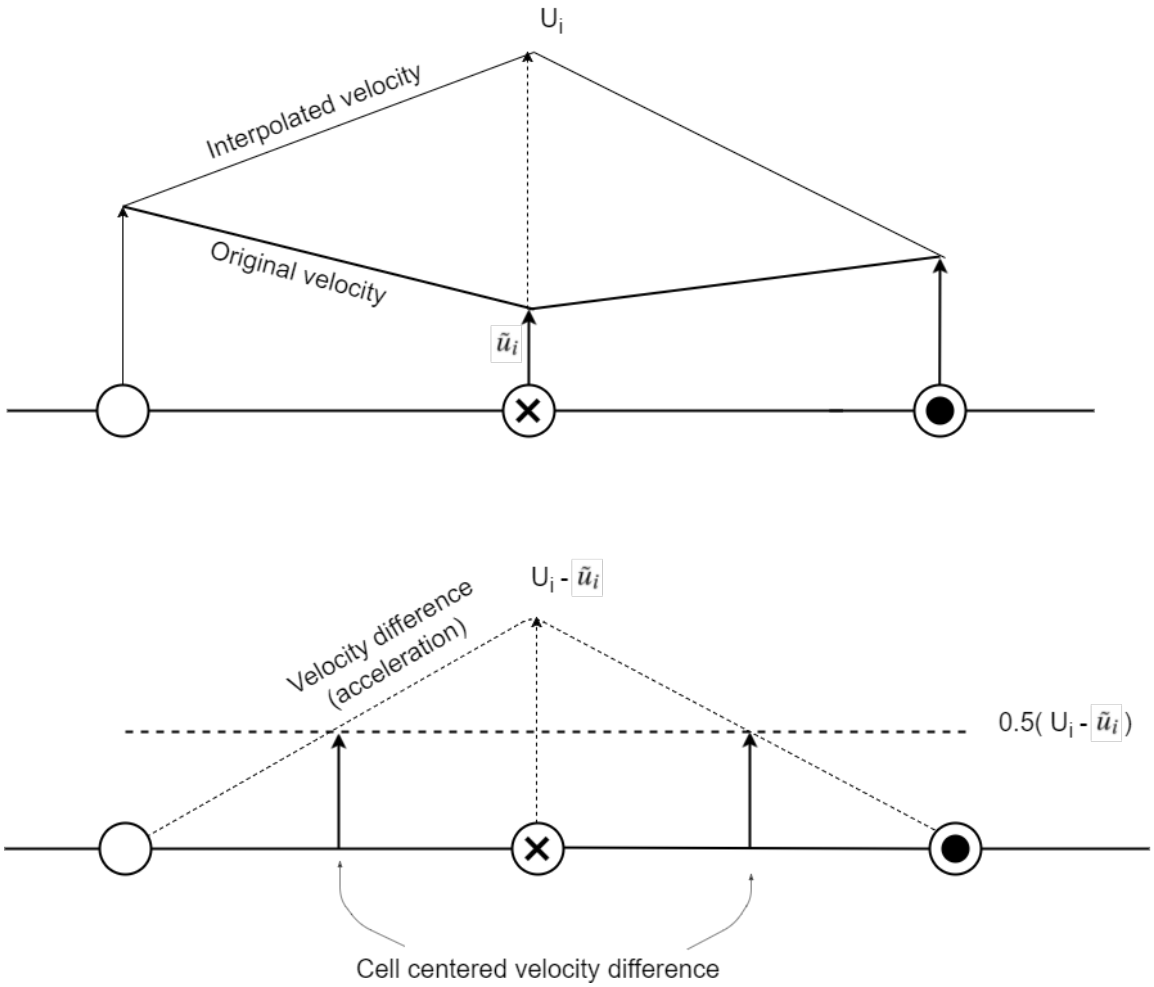


Figure 6.14: Visualization of the accelerated fluid node, and the connected fluid volumes used to estimate the force (F_i) on the boundary.

Total force from fluid

The total force from the fluid, F_d on the boundary, in the given direction, d, may thereafter be taken as the sum of the contributions from each intersection:

$$F_d = \sum_{\times_d=1} N_{\times_d} dF_d|_{\times} \tag{6.26}$$

6.5 The combined IBM-N-S solver

Figure 6.15 shows how the IBM-module was implemented into the NS-solver. The overall structure of the NS-solver remains as presented in Figure 6.2, while the time integrator is modified to take into account the immersed boundary. It should be noted that this implementation is not yet a *fluid-structure-interaction*-solver, as there is no module for treating the structural response of the system. Further development of the code is covered in Chapter 9.

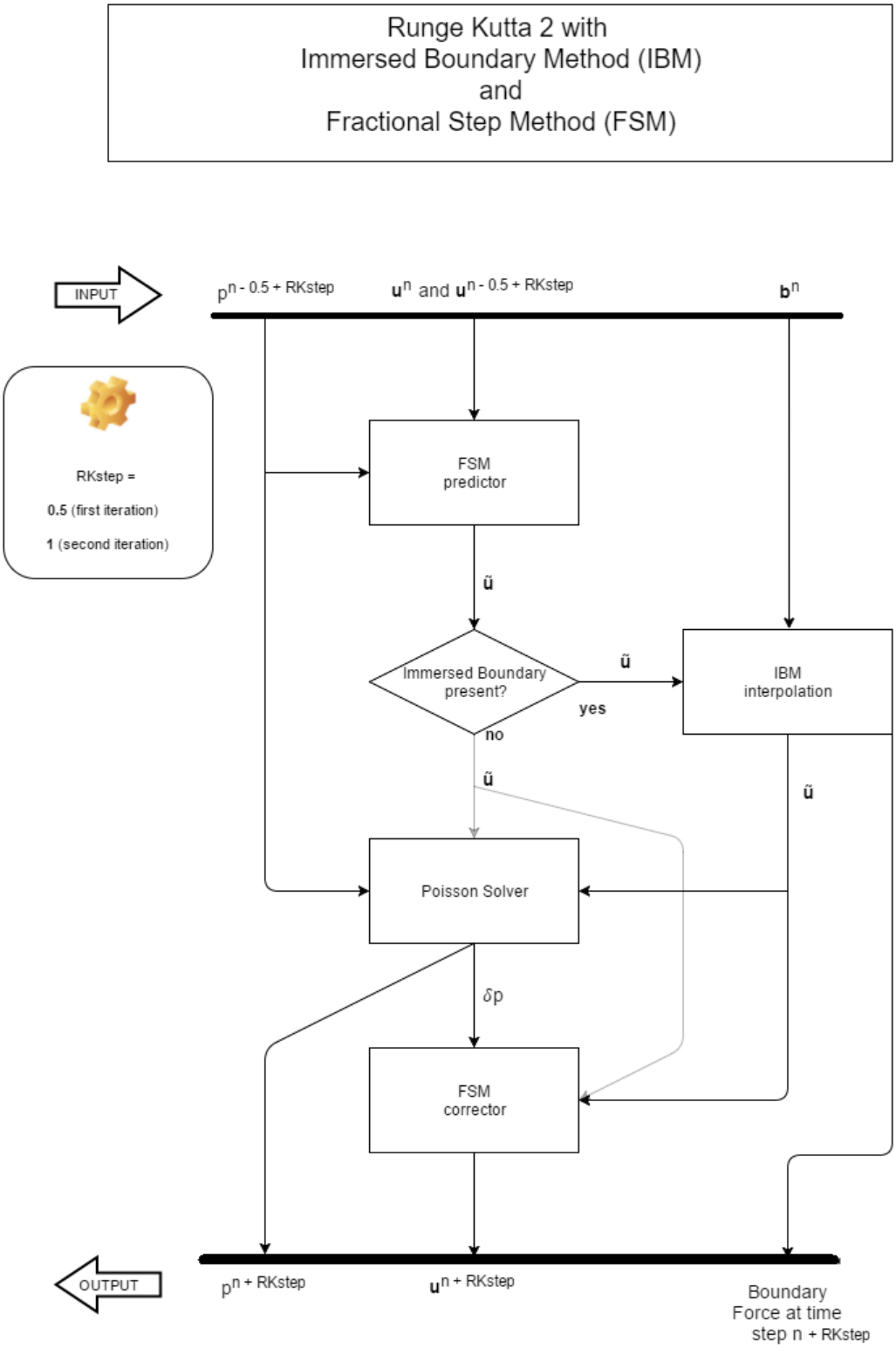


Figure 6.15: Flow chart showing the implementation of the IBM module into the original RK2 subroutine.

6.6 Limitations

The IBM-module that is presented in this chapter is limited to solve 2D flow problems of closed polygons in fully submerged domains. The use of linear interpolation induces some trouble when it comes to nodal ambiguity and grid resolution.

6.6.1 Nodal ambiguity

The IBM-module is supposed to perform calculations on arbitrarily shaped simple polygons, but since the interpolation procedure uses more than one node at each calculation, nodal ambiguity may occur:

- interpolation carried out twice on same node (Type 1, see Figure 6.16a)
- interpolation done by information from already interpolated node (Type 2, see Figure 6.16b)

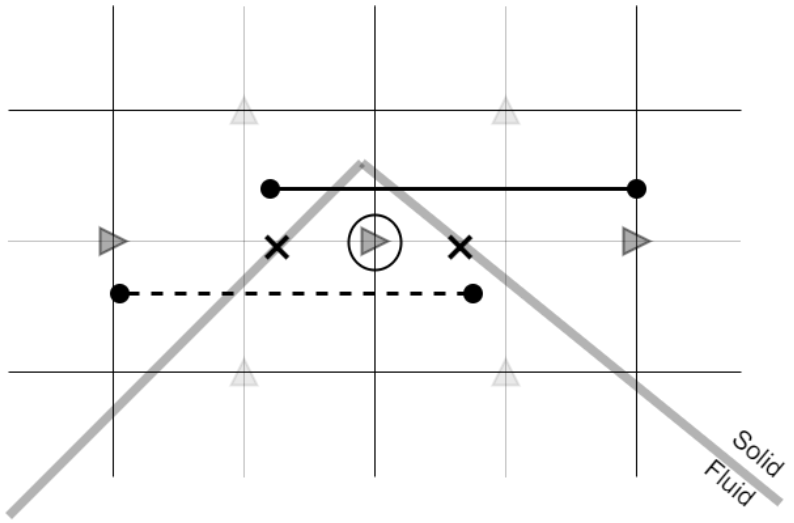
Nodal ambiguity may corrupt the calculations, and should be warned about. The code is therefore equipped with a subroutine that checks for and warns about ambiguous nodes during calculation, so that the user may stop the calculations and adjust the mesh and/or the number of nodes in the immersed boundary accordingly.

6.6.2 Interpolation on coarse grids

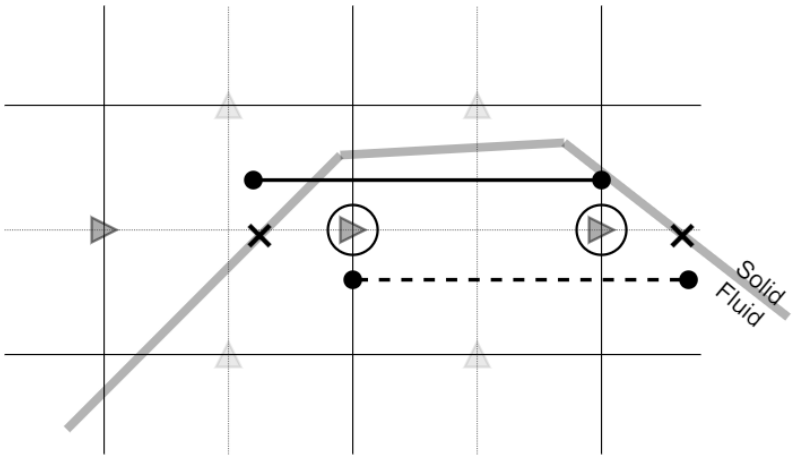
The code uses linear interpolation to mimic the velocity field near the immersed boundary. This is based on the assumption that the velocity field is is approximately linear close to the boundary. The mesh resolution is therefor limited by Equation 4.7:

$$2\Delta x_i \leq \Delta l_i$$

Apart from undermining the accuracy, use of a too coarse mesh may result in surprising trends for mesh refinement. Figure 6.17 shows how the placement of the immersed boundary on the grid is affecting the interpolation error if the grid is too coarse. Since the placement of the grid-lines is important, a slight change of the resolution on a coarse



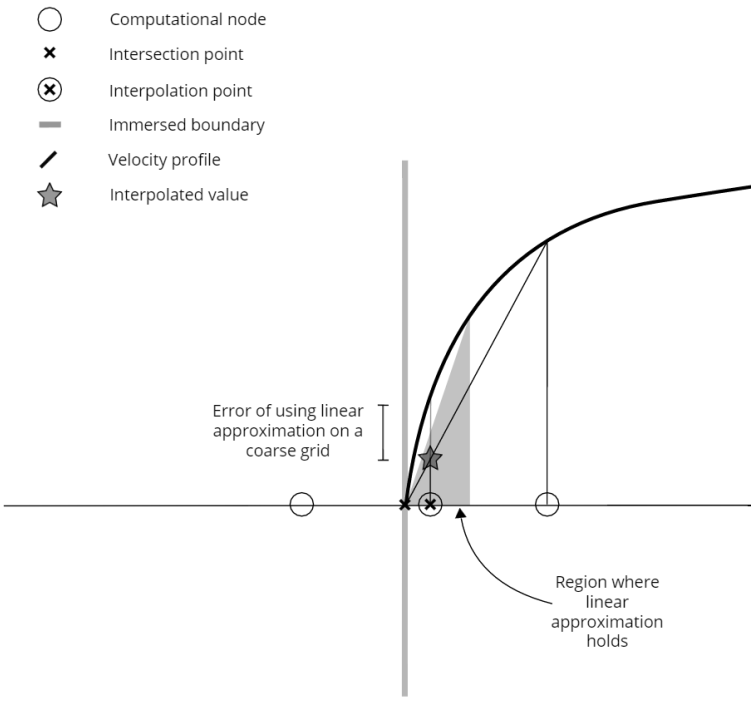
(a) Type 1



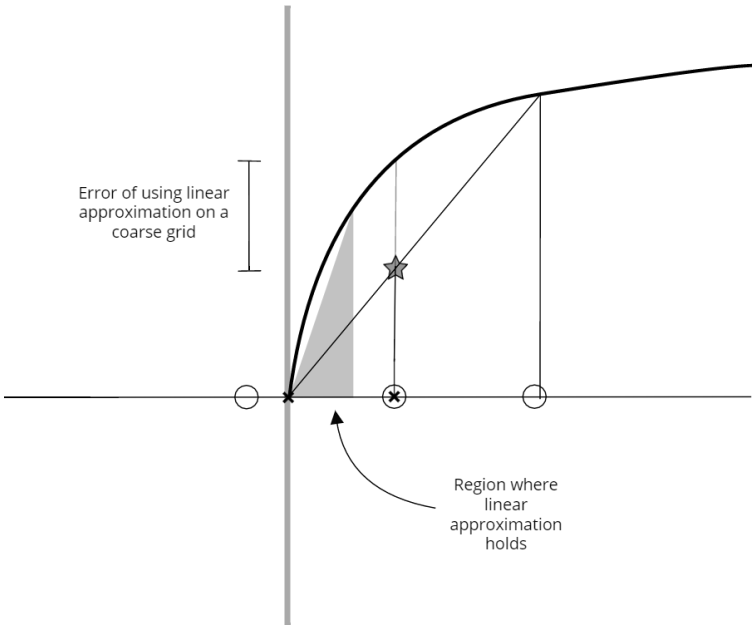
(b) Type 2

Figure 6.16: Ambiguous node types. Type 1 nodes (a) will be interpolated twice, while type 2 nodes (b) will use information from nodes that are interpolated earlier. Interpolation ranges are represented by lines with dots on the ends. The ambiguous nodes are marked with a circle.

grid may change the results unpredictably. Mesh resolution studies that shows significant jumps in the results may indicate that order of magnitude for the grid resolution is too low.



(a) Intersection is close to the interpolation node.



(b) Intersection is far from the interpolation node.

Figure 6.17: A schematic representation of a velocity profile near the immersed boundary. On coarse grids, (Δx_i is larger than the linear approximation region), the linear interpolation scheme is sensitive to the distance between the outer interpolation node and the linear region. This issue is overcome by refining the mesh.

Chapter 7

Validation of fluid solver

The validation of the code was done stepwise, as the code grew in complexity, and before each new major development step, a valid code version was branched out.

7.1 Validation of Poisson solver (Gauss Seidel-iterator)

The Gauss Seidel iterator was validated by solving the example problem

$$\frac{\partial^2 \Phi}{\partial x_i \partial x_i} = \cos(n\pi x_1)\cos(n\pi x_2) \tag{7.1}$$

where n is an integer defining the spatial wave-number of the solution. This problem has the analytical solution

$$\Phi = \frac{1}{(n\pi)^2}\cos(n\pi x_1)\cos(n\pi x_2) \tag{7.2}$$

Setup

The test problem was solved on a domain spanned by $x_1 = [-1,1]$ and $x_2 = [-1,1]$. The mesh resolution was set to $M \times M$, and M was varied in order to analyze the behavior of the solver and the numerical solution, ϕ. Tests were done for $n = 1$, $n = 2$ and $n = 3$.
The convergence criterion was chosen to be:

$$L_\infty\left(\phi^{n+1} - \phi^n\right) \leq 10^{-6} \tag{7.3}$$

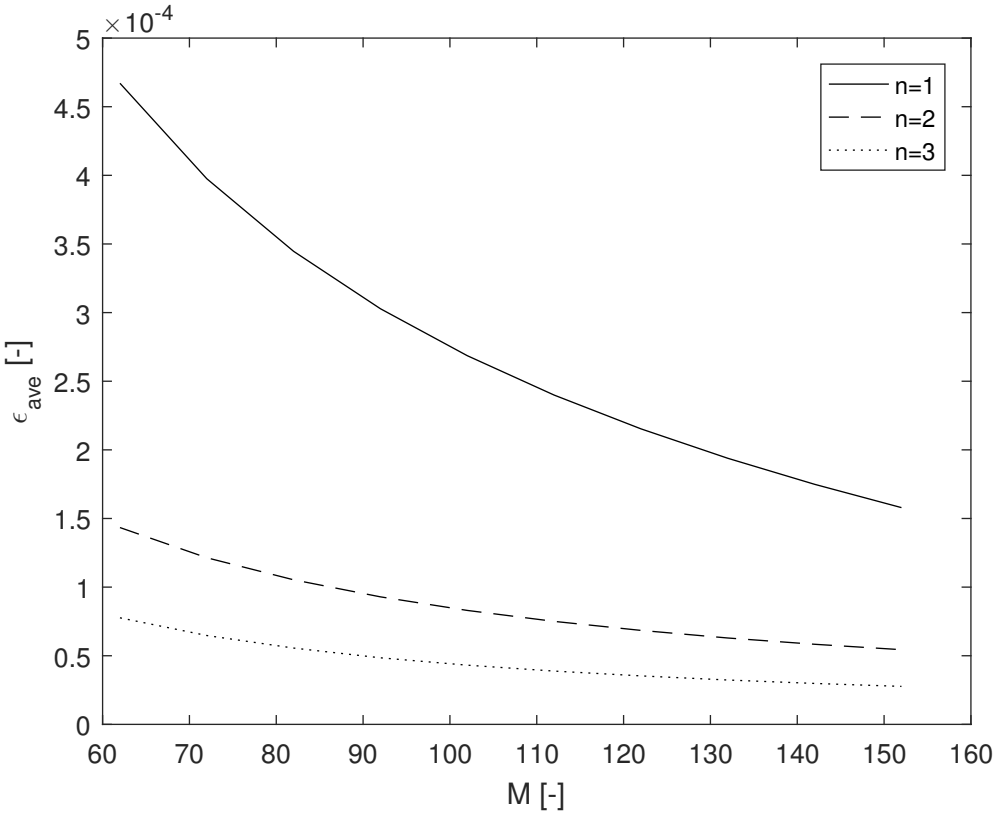


Figure 7.1: Average field error ϵ_{ave} compared to mesh resolution M for the Gauss-Seidel iterator.

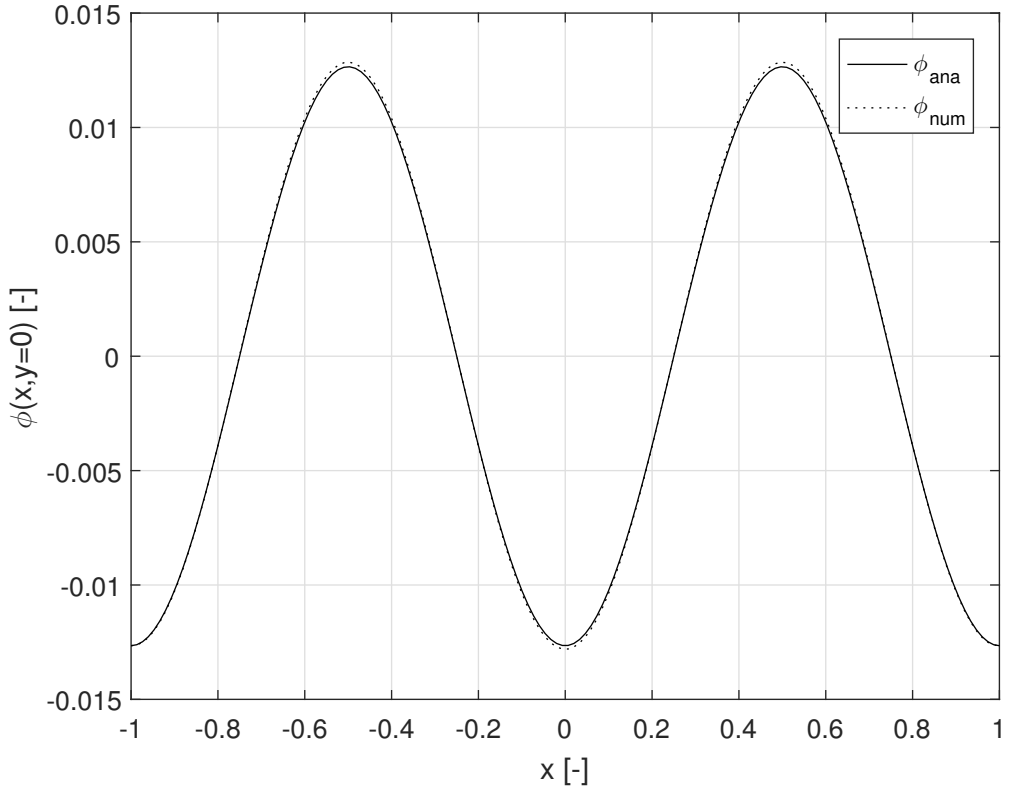
Error analysis

Figure 7.2a displays the numerical and the analytical value of the solution along the line $(x, y = 0)$ for $n = 2$ and $M = 152$. It is clear that the characteristics of the analytical solution are well captured by the numerical result. The error distribution given by $\epsilon = \Phi - \phi$, along the same line, is shown in Figure 7.2b, and it turns out that the error magnitude varies correspondingly to the curvature of the solution.

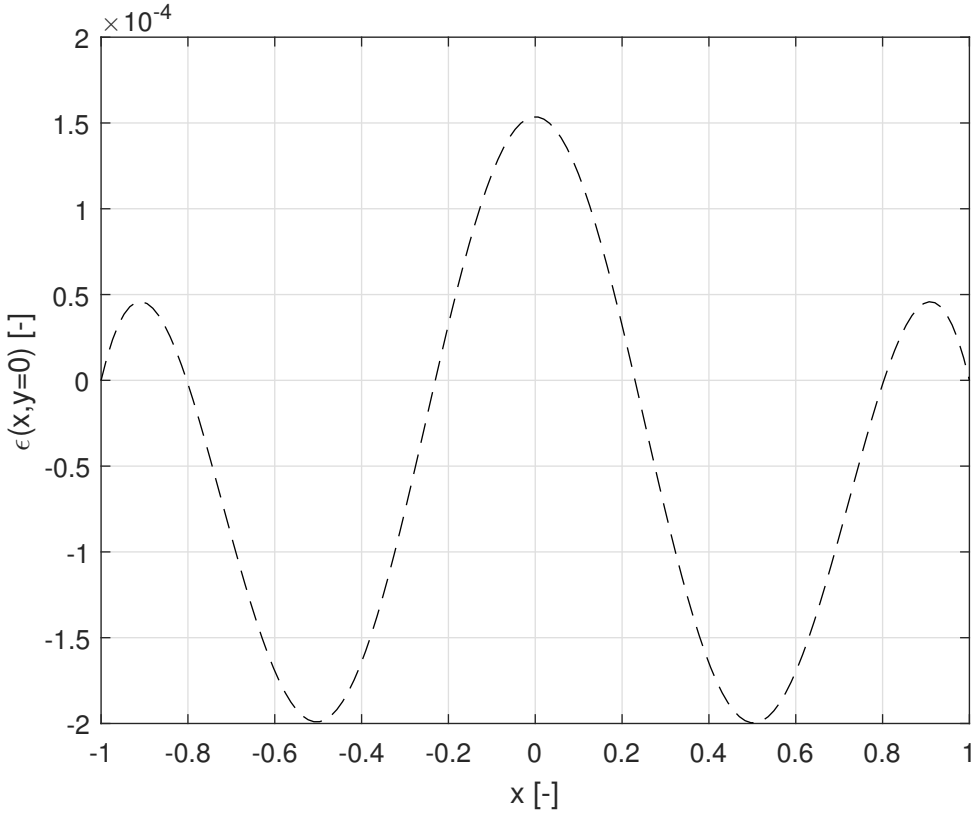
The average field error is defined as

$$\epsilon_{ave} = \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} |\Phi_{i,j} - \phi_{i,j}| \tag{7.4}$$

Figure 7.1 shows a converging behavior for ϵ_ϕ as the mesh resolution is increasing. The magnitude of the convergence limit for all three values of n, seems to be of order $O(10^{-4})$, given the convergence criterion of $L_\infty \leq 10^{-6}$.



(a)

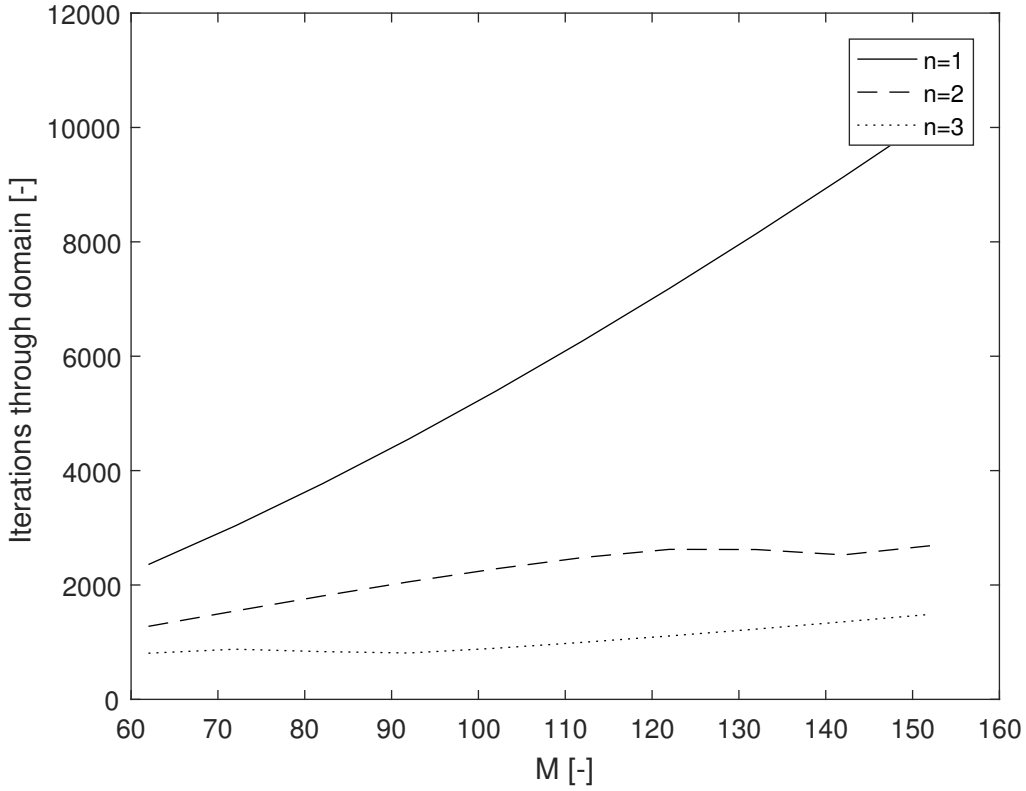


(b)

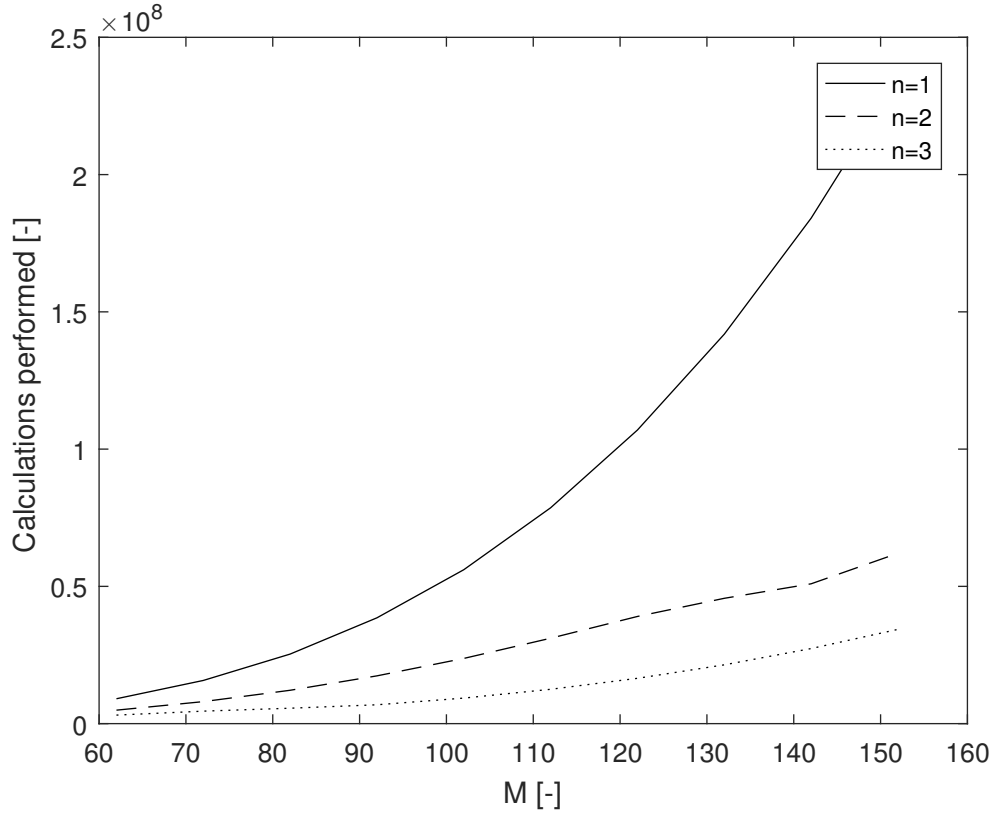
Figure 7.2: Comparison between numerical and analytical solutions for the reference problem for ϕ (a), and error between the two solutions (b).

Efficiency analysis

As seen in Figure 7.3a, the number of iterations needed in order to find a satisfactory solution, increases with the mesh size. Further, the number of discrete points in the mesh increases with M^2, and it may be seen from Figure 7.3b that the number of calculations increases rapidly with the mesh size. This is especially true for the case where $n = 1$, which has the solution with the lowest wavenumber, and thereby the smallest curvatures and gradients.



(a) Number of iterations through domain compared to mesh resolution M for the Gauss-Seidel iterator.



(b) Number of calculations performed to achieve solution compared to mesh resolution M.

Figure 7.3: Gauss-Seidel iterator: Efficiency analysis.

7.2 Validation of the Navier-Stokes solver

The NS-solver was validated on a lid-driven cavity flow problem such as the one analyzed and discussed by Ghia et al. (1982). The present code was used to perform an analysis of the problem setup given in Tab.7.1. The results are presented and compared to those of Ghia et al. (1982) in Fig.7.4 and 7.5. Apart from a single mis-matching point[1] in the vertical velocity profile, the results show a good concordance.

Table 7.1: Setup of numerical analysis for a lid-driven cavity flow problem.

Parameter	Value
Re	$400[-]$
Domain	$[0,1] \times [0,1]$ $[-]$
Mesh	101×101 $[-]$
Time step	$0.001[-]$
Time range	Until stationary flow
BC North	Moving wall, $U_w = 1[-]$
BC East	Stationary wall
BC South	Stationary wall
BC West	Stationary wall

[1] This point is assumed to be a misprint in the work of Ghia et al. (1982), as it is only found in the result table and not in the result figures given in the same source.

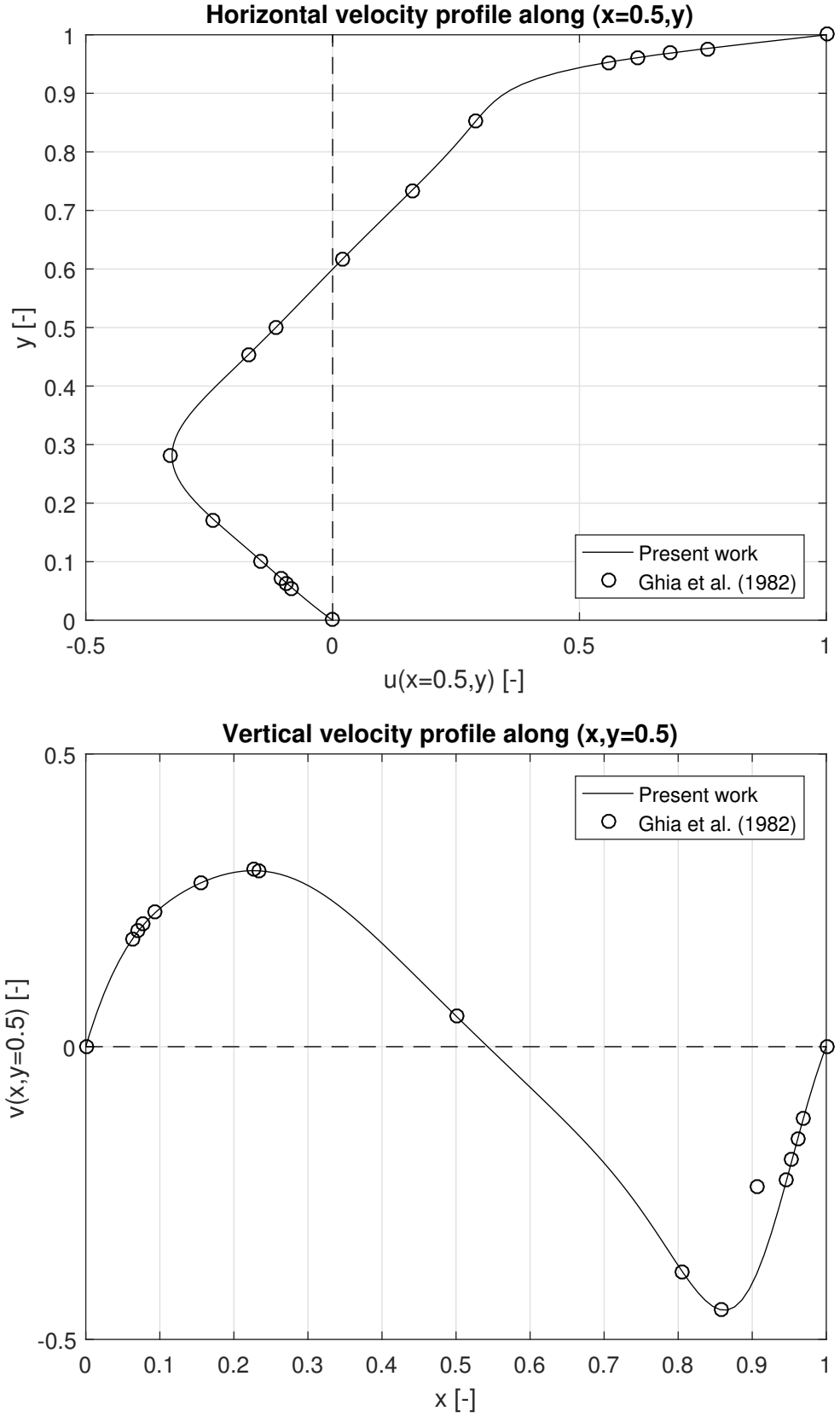


Figure 7.4: Comparison of present work to that of Ghia et al. (1982): Horizontal and vertical velocity profiles along the vertical and horizontal mid-line, respectively. Notice the single mismatching point on the vertical velocity profile that is assumed to be a misprint in Ghia et al. (1982) (see footnote 1) of this Section.

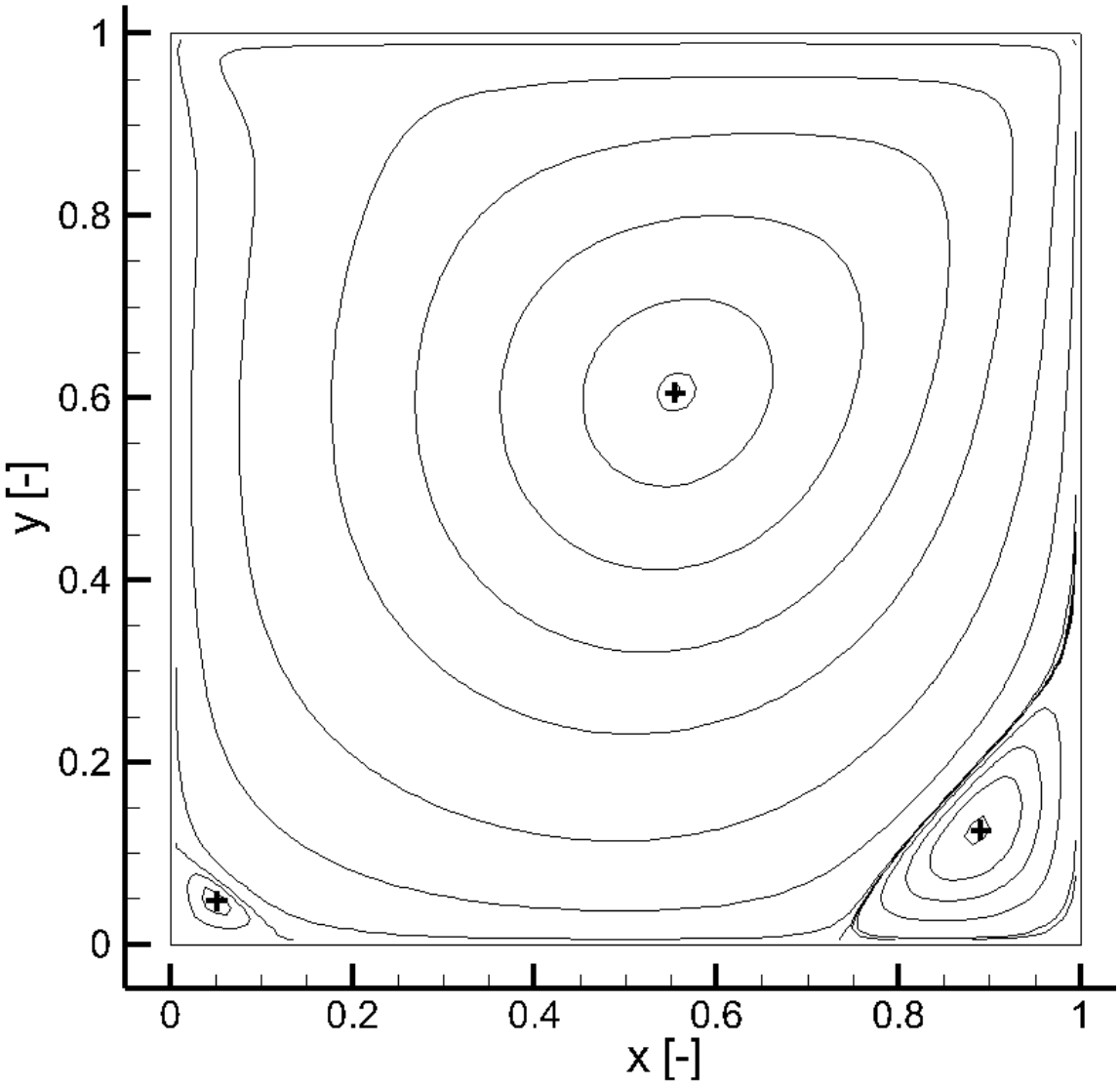


Figure 7.5: Contour lines of the stream function Ψ for the lid-driven cavity at $t = 25[-]$. The "+" signs marks the vortex centers found by Ghia et al. (1982), which seem to be in agreement with the present work.

Chapter 8

Validation of the immersed boundary solver

Since the implemented IBM-module is meant to solve intricate problems with moving arbitrary polygons, the validation procedure must consist of several steps:

1. Validate for stationary objects
2. Validate for objects with a prescribed translational motion
3. Validate for objects with a prescribed rotational motion

The validation process was severely limited by the combination of little available computational power (see Section 1.3) and the inefficiency of the Gauss-Seidel algorithm when applied on fine meshes (see Section 7.1). This section is therefore only devoted to describe the validation for stationary objects. It must also be emphasized that only the IBM-version with *parallel* interpolation is tested in this section. Plans for further validation are described in Section 9.1.

8.1 Reference problem

Validation for flow past stationary objects was done by comparing force coefficients obtained by various studies on 2D uniform flow past circular cylinders. It was decided to stick with low Reynolds numbers, since the available computational power was limited. Table 8.1 shows the calculated drag coefficients from several studies of the problem, for Reynolds numbers of 20 and 100.

For $Re = 20$, the vortex formation behind the cylinder is symmetric, so the lift coefficient should be zero. Reference studies on the lift coefficients for $Re = 100$ is presented in Table 8.2.

Table 8.1: Existing studies of drag coefficients, C_D, for 2-dimensional flow past a circular cylinder.

Source	$Re = 20$	$Re = 100$
Tritton (1959)	2.22	-
Dennis and Chang (1970)	2.05	-
Sucker and Brauer (1975)	2.08	1.45
Clift et al. (1978)	-	1.24
Park et al. (1998)	2.01	1.33
Ye et al. (1999)	2.03	-
Calhoun (2002)	2.19	1.33
Russel and Wang (2003)	2.22	1.34
Silva et al. (2003)	2.04	1.39
Xu and Wang (2006)	2.23	1.423
Wang and Cen (2009)	2.25	1.379
Average	2.13	1.36

Table 8.2: Existing studies of the lift coefficient, C_L, for 2-dimensional flow past a circular cylinder at $Re = 100$.

Source	$Re = 100$
Braza et al. (1986)	0.25
Calhoun (2002)	0.298
Xu and Wang (2006)	0.34
Khoo et al. (2008)	0.346
Wang and Cen (2009)	0.357
Average	0.3182

8.2 Method and setup

The validation was conducted as follows:

1. Check that solver captures general flow characteristics
2. Check that solver trends to calculate reasonable force coefficients by
    - time refinement analysis
    - mesh refinement analysis
    - domain size analysis

The domain parameters a, b, c and D (see Figure 8.1) will in the further analyses be varied in order to see the trends of the results. Other parameters of interest are:

- Mesh resolution in x-direction - $\frac{D}{\Delta x}$
- Mesh resolution in y-direction - $\frac{D}{\Delta y}$
- End time for calculations - T
- Time resolution - Δt
- Reynolds number - Re

Setup

Figure 8.1 shows the setup of the problem domain, while Table 8.3 presents the applied boundary conditions.

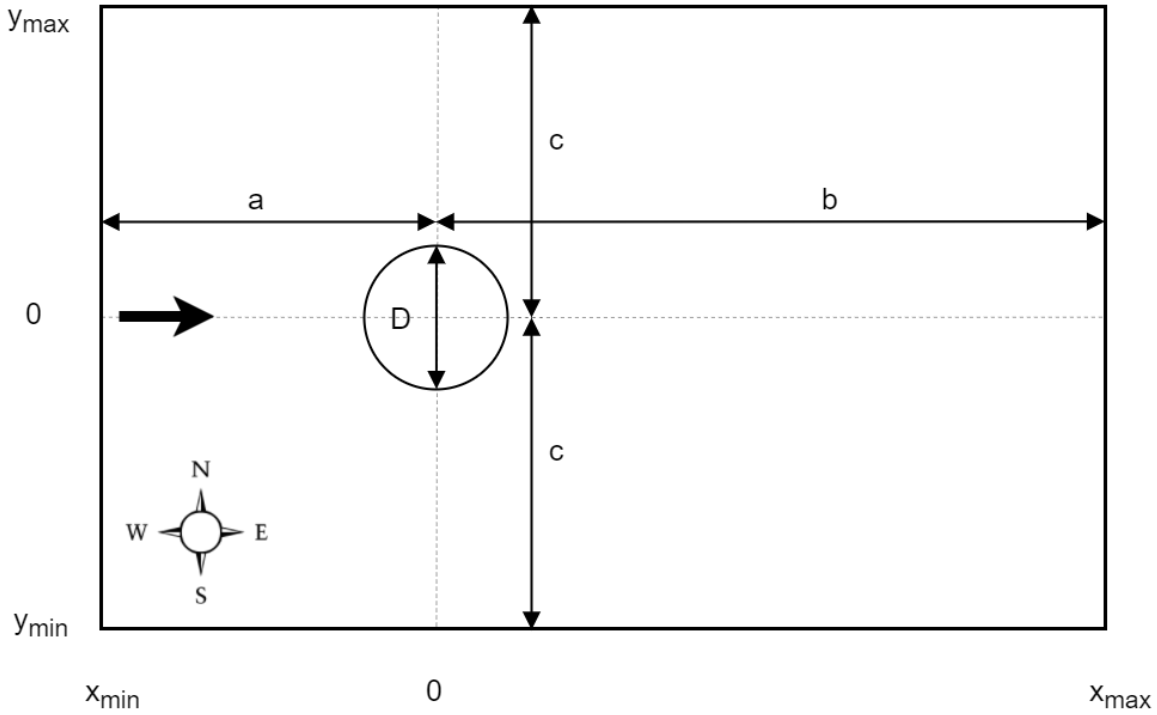


Figure 8.1: Domain setup for the circular cylinder problem where a is the upstream clearance, b is the downstream clearance and c is the clearance between the origin and the stream-parallel walls.

Table 8.3: Boundary- and initial conditions applied on the domain for the circular cylinder problem. N means that a zero-normal-gradient Neumann condition is applied, while a number signifies the value of an applied Dirichlet condition.

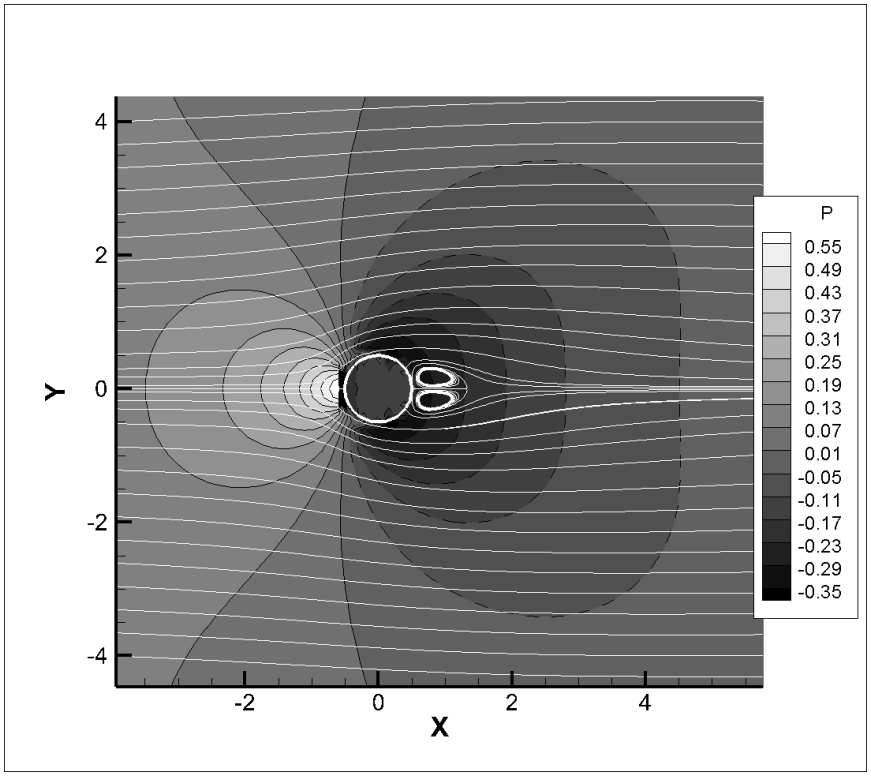
BCs and ICs	u	v	p
North wall	N	0.0	N
South wall	N	0.0	N
East wall	N	N	0.0
West wall	1.0	N	N
Initial condition	1.0	0.0	0.0

8.3 Flow characteristics

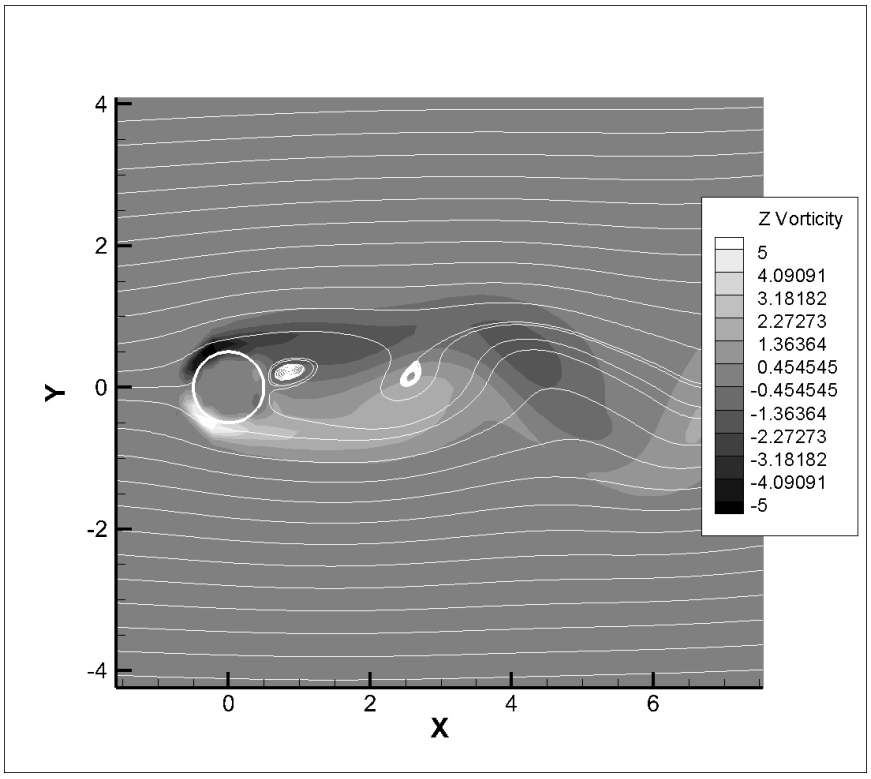
The very first validation step was done by checking that the solver represented the flow characteristics as anticipated. By flow characteristics is meant features such as vortex formation, stagnation and impermeability. Two tests were run on a relatively small and coarse domain, with parameters as presented in Table 8.4. To obtain vortex shedding at an early stage for the run on $Re = 100$, the cylinder was initially oscillated for two periods, with a dimensionless frequency of $f = 0.2$ and a dimensionless amplitude of 0.01. Figure 8.2 shows that the solver is capable of representing all the expected flow features, so that more profound validation may be commenced.

Table 8.4: Domain setup for initial testing of solver, done to see that typical flow behavior is contained in the analysis. a, b, c and D as defined in Figure 8.1.

Parameter	Value
a	$5D$
b	$10D$
c	$7.5D$
D	1.0
$\frac{D}{\Delta x}$	8
$\frac{D}{\Delta y}$	8
T	25
Δt	0.01
Re	20 and 100



(a) Streak-lines (white) and pressure contours for $Re = 20$.



(b) Streak-lines (white) and vorticity contours for $Re = 100$.

Figure 8.2: Post-processed results from the analyses presented in Table 8.4 for $Re = 20$ (a) and $Re = 100$ (b). The results show that vortex formation, flow stagnation and vortex shedding is captured by the solver.

8.4 Time refinement tests

To check the dependency between accuracy and time resolution, three tests were run on the domain given by Table 8.5a. The results are presented in Table 8.5b, and shows that the time resolution is not affecting the accuracy very much. Time refinement is however very important when it comes to stability. The highest time resolution ($\Delta t = 0.1$) yields an inflow Courant-number nearly equal to one, and the unstable integration for this time step was expected.

The value obtained for the lift coefficient is, as expected, approximately equal to zero. The drag-coefficient, on the other hand, is a bit overestimated in comparison to the reference results. This may be because the mesh is too coarse, or that the domain is defined to tightly around the cylinder, so that the boundary conditions pollute the solution near the cylinder. Both causes will be investigated in the next two subsections.

Table 8.5: Domain setup (a), and results (b) from the time refinement tests.

(a) Domain setup for time refinement tests. a, b, c and D as defined in Figure 8.1.

Parameter	Value
a	$5D$
b	$10D$
c	$5D$
D	1.0
$\frac{D}{\Delta x}$	8
$\frac{D}{\Delta y}$	8
T	25
Δt	Was varied
Re	20

(b) Drag-coefficients obtained with different time resolutions.

Δt	$C_d(t = 25)$	$C_l(t = 25)$
0.1	unstable	unstable
0.01	2.38986858	$O(10^{-12})$
0.001	2.38986874	$O(10^{-12})$

8.5 Mesh refinement tests

The dependency between mesh resolution and accuracy was analyzed by varying the mesh resolution for the problem domain described in Table 8.6a. The resolution was varied equally in both directions to keep the grid Cartesian. The results from the calculations are presented in Table 8.6b. The results from the calculations, which are presented in Table 8.6b, show that results are far more sensitive to mesh resolution changes than for time resolution changes. The drag coefficient is still overestimated compared to the results found in literature, though an increase of resolution seems to pull the results in the right direction. The next subsection will investigate whether the overestimation may be due to the domain size.

Table 8.6: Domain setup (a), and results (b) from the mesh refinement tests.

(a) Domain setup for mesh refinement tests. a, b, c and D as defined in Figure 8.1.

Parameter	Value
a	$5D$
b	$10D$
c	$5D$
D	1.0
$\frac{D}{\Delta x}$	Was varied
$\frac{D}{\Delta y}$	Was varied
T	25
Δt	0.01
Re	20

(b) Drag-coefficients obtained with different mesh resolutions.

$\frac{D}{\Delta x_i}$	$C_d(t=25)$
8	2.38987
10	2.36221
12	2.34202

8.6 Domain size tests

To find the dependency between the domain size and the drag coefficients, As seen in

Table 8.7: Domain setup (a), and results (b) from the domain size tests.

(a) Domain setup for domain size tests. a, b, c and D as defined in Figure 8.1.

Parameter	Value
a	Was varied
b	Was varied
c	Was varied
D	1.0
$\frac{D}{\Delta x}$	8
$\frac{D}{\Delta y}$	8
T	25
Δt	0.01
Re	20

(b) Drag-coefficients obtained with different domain size.

a	b	c	$C_d(t=25)$
$5D$	$10D$	**5D**	2.38987
$5D$	$10D$	**7.5D**	2.23220
$5D$	$10D$	**10D**	2.16690
$5D$	$10D$	**12.5D**	2.13413
$5D$	$10D$	**15D**	2.11312
$5D$	**10D**	$5D$	2.38987
$5D$	**15D**	$5D$	2.38267
$5D$	**20D**	$5D$	2.37247
5D	$10D$	$5D$	2.38987
10D	$10D$	$5D$	2.37685

Table 8.7b and Figure 8.3, the effect of the domain parameters on the drag coefficient is by far dominated by the domain width. The domain that provided the best results for C_D was a lot wider than it was long, with an aspect ratio of $\frac{(a+b)}{2c} = \frac{1}{2}$. This was surprising, as the "typical" domain for studies on flow past cylinders has an aspect ratio much larger than 1 (see Figure 3.3). An explanation for this could be founded on that the analyses were run for $Re = 20$, which means that no vortexes were shed:

In a vortex-free wake the velocity gradient $\frac{\partial v}{\partial x}$ vanishes just a short distance downstream of the cylinder, so that the Neumann condition for v does not enforce large changes in the vertical velocity field. The horizontal gradient of u also diminishes relatively fast behind the cylinder. This means that a long downstream domain is unnecessary. A vortex is recognized by its large velocity gradients, $\frac{\partial u}{\partial y}$ and $\frac{\partial v}{\partial x}$. If a vortex is not sufficiently weakened before it hits a boundary with Neumann velocity conditions, then the velocity

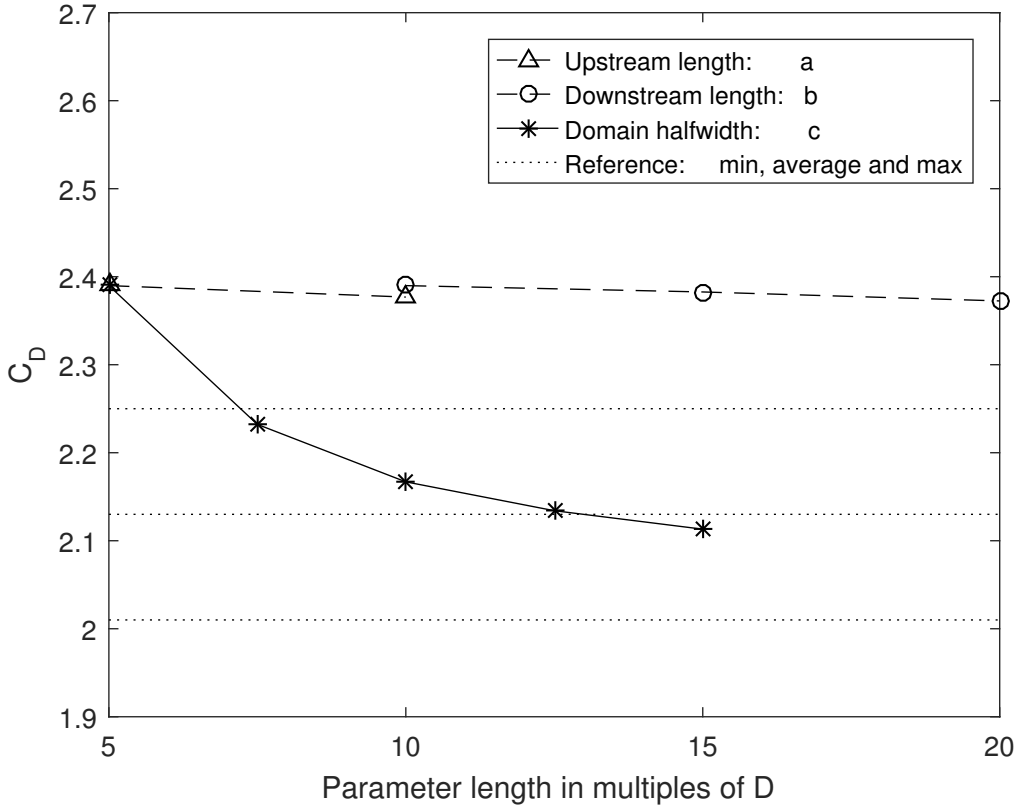
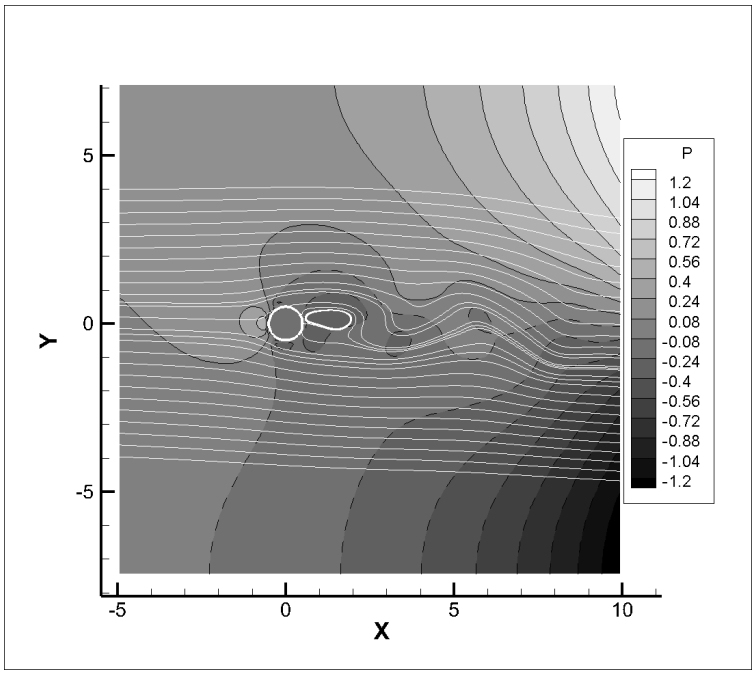


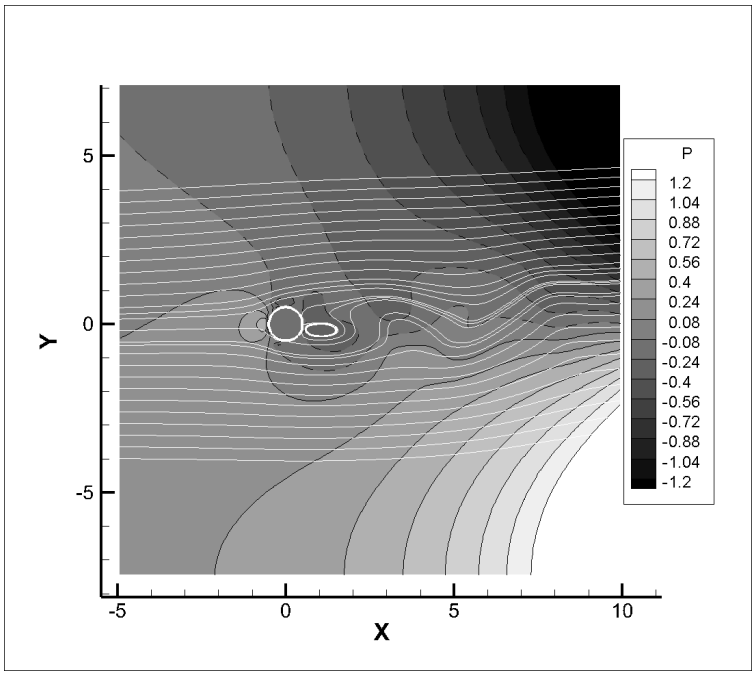
Figure 8.3: Effect on the drag coefficient from varying the domain size parameters for $Re = 20$. The drag coefficient is clearly most sensitive to the widthwise span (parameter c) of the domain. Increasing the domain size shows results that are trending towards the reference results.

field is significantly manipulated by the velocity condition, and the solution becomes contaminated as shown in Figure 8.4. This leads to a requirement for a long downstream domain. Since most studies on circular cylinders are concerned with vortex shedding, the "typical" domain is the one with a high aspect ratio.

For $Re = 20$, and probably for higher Reynolds numbers as well, the width of $2c \geq 30D$ should be used to avoid contamination from boundary conditions on the stream-parallel walls. Analyses of higher Reynolds numbers will also require a new study of the effects of the upstream and downstream domain length (parameter a,and b, respectively).



(a) Vortex formed at upper edge of cylinder.



(b) Vortex formed at lower edge of cylinder.

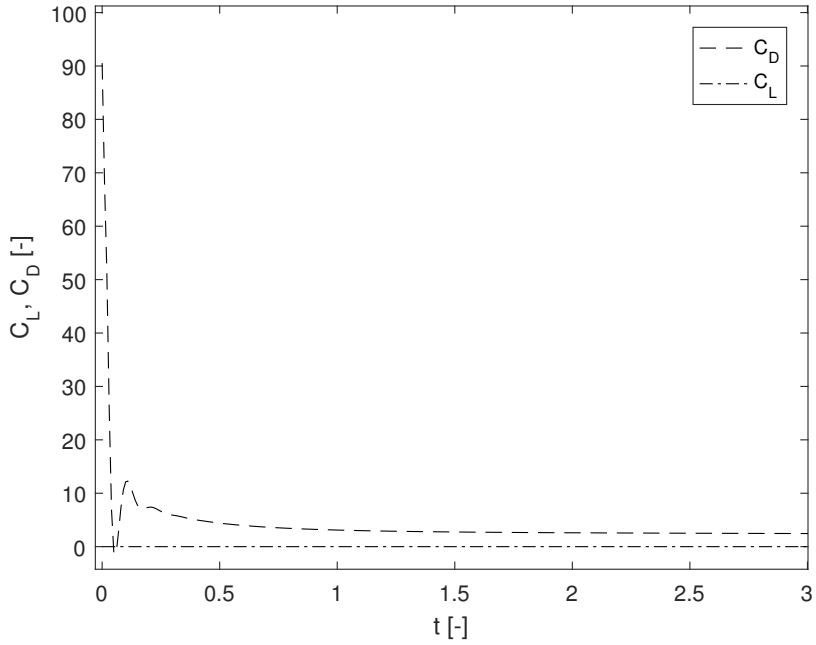
Figure 8.4: Vortex shedding at $Re = 100$ in a too small domain. Extreme pressure gradients are formed towards the downstream corners of the domain, and the sign of the pressure oscillates with the vortex shedding frequency. These pressure artifacts may be a result of enforcing of Neumann velocity conditions on a velocity field with strong vortexes. It is clearly seen on the figure that the solution becomes contaminated, since the pressure gradient pulls the upstream flow in the transverse direction so that the angle of attack on the cylinder alters. This will affect results for both C_D and C_L.

8.7 Transients at flow start-up

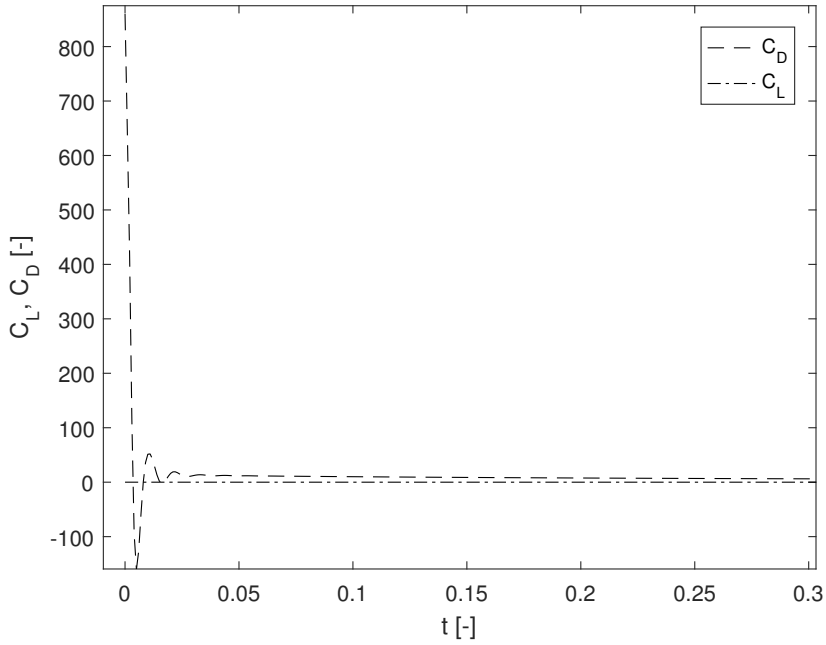
At the startup of the simulations, the flow field is fully uniform, before the immersed boundary suddenly is placed into the fluid. The abrupt velocity change results in transient oscillations of the flow and fluid forces at early stages of the flow development. As seen in Figure 8.5a, both the amplitude and damping of the transient effects are highly dependent of the time step, Δt.

The transient force oscillations are of serious concern when considering further development of the code into an FSI-solver. A force coefficient overestimated by magnitudes as seen in Figure 8.5a will result in extreme accelerations for the response of the immersed boundary.

When the immersed boundary moves in the fluid, it will occupy new fluid space at each time step. This is anticipated to cause effects similar to those seen at startup flow. To check the significance of transients related to body motions a test were run on a cylinder oscillating in an initially stationary fluid. Table 8.8 shows the test domain setup, and the results are presented in Figure 8.6 and 8.7. Figure 8.6 shows that the flow is represented with similar characteristics as the work by Dütsch et al. (1998), while Figure 8.7 displays that transient effects for immersed boundaries moving in stationary indeed are highly significant.



(a) Transient effects on the force coefficients at startup for $Re = 20$, with $\Delta t = 0.01$. The maximum force coefficient is about 45 times larger than the stationary result.



(b) Transient effects on the force coefficients at startup for $Re = 20$, with $\Delta t = 0.001$. The maximum force coefficient is about 450 times larger than the stationary result.

Figure 8.5: Transient effects on force coefficients at startup flow past a stationary circular cylinder: $\Delta t = 0.01$ (a), and $\Delta t = 0.001$ (b). A time step refinement results in an increase of force, but also shortens the die-out-time for the oscillations.

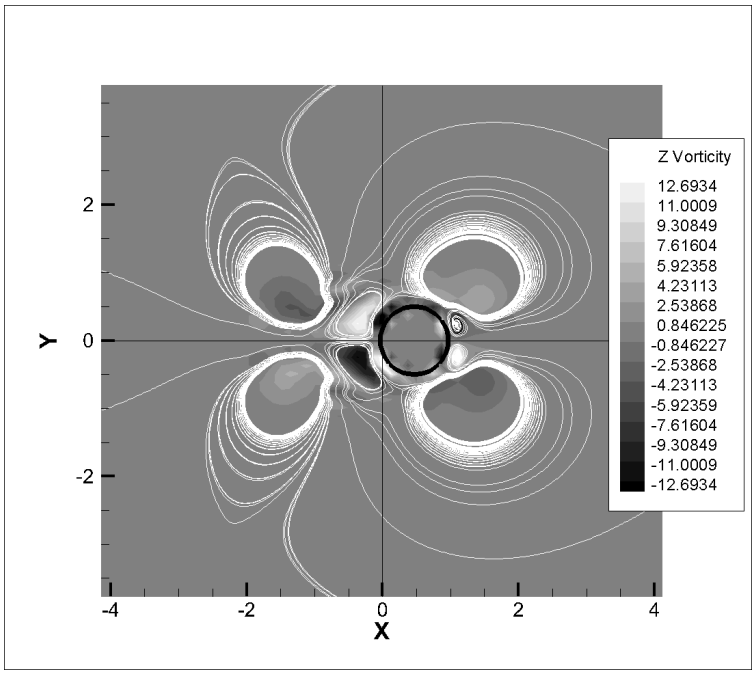
Table 8.8: Test specifications for cylinder moving in stationary fluid.

(a) Domain setup for analysis of oscillating cylinder in stationary fluid. a, b, c and D as defined in Figure 8.1.

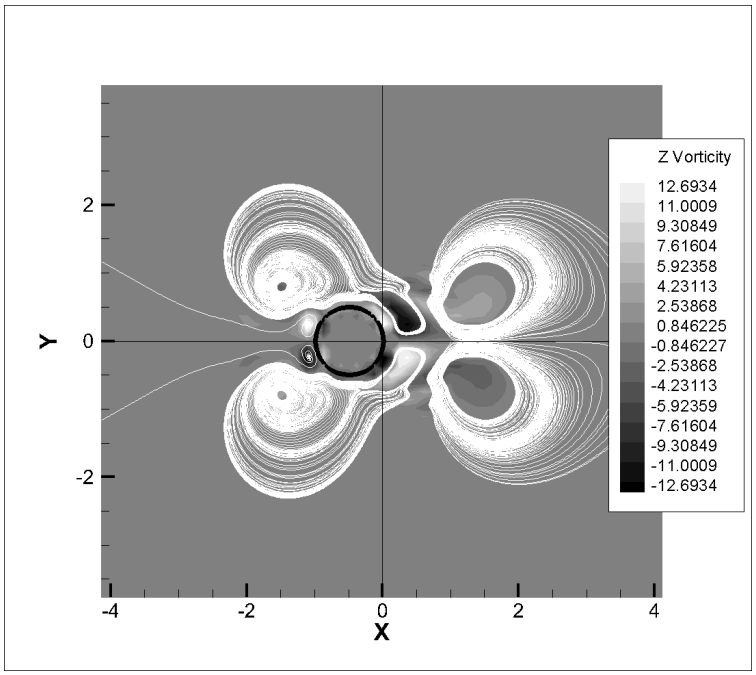
Parameter	Value
a	$7.5D$
b	$7.5D$
c	$5D$
D	1.0
$\frac{D}{\Delta x}$	8
$\frac{D}{\Delta y}$	8
T	5
Δt	0.001
Re	100

(b) Specifications for the cylinder motion.

Parameter	Value
Frequency	1.0[−]
Amplitude	0.5[−]



(a)



(b)

Figure 8.6: Vortex formation around a cylinder oscillating in an initially stationary fluid. (a) shows the cylinder near its leftmost position, while (b) shows the cylinder near its rightmost position. In both cases, the freshly formed vortex is visible close to y-axis, while the vortex formed at the previous period has propagated vertically outwards.

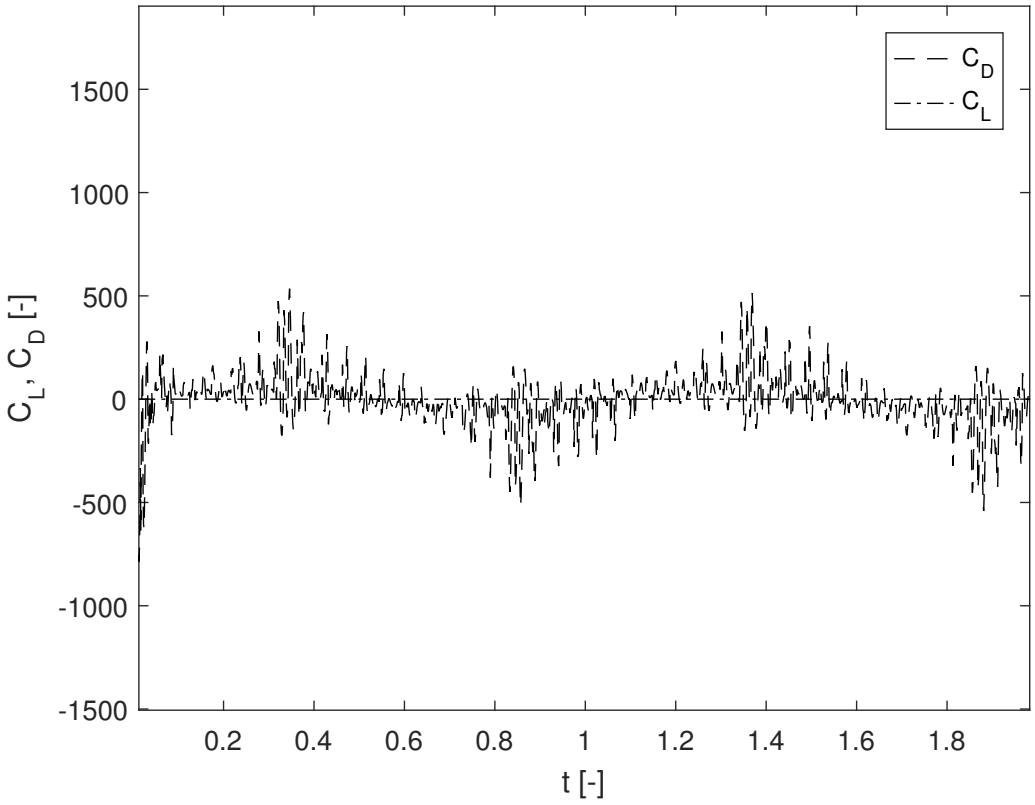


Figure 8.7: Force coefficients calculated for the oscillating cylinder test given by Table 8.8. The results are severely affected by transient numerical oscillations.

8.8 Force coefficients at $Re = 100$

Time did not suffice to perform profound validation analyses for higher Reynolds numbers, but one analysis with a reasonable domain length was conducted. Table 8.9a shows the domain specifications, and the results are presented in Figure 8.8 and 8.8b.

Table 8.9: Test at $Re = 100$: Domain setup and results.

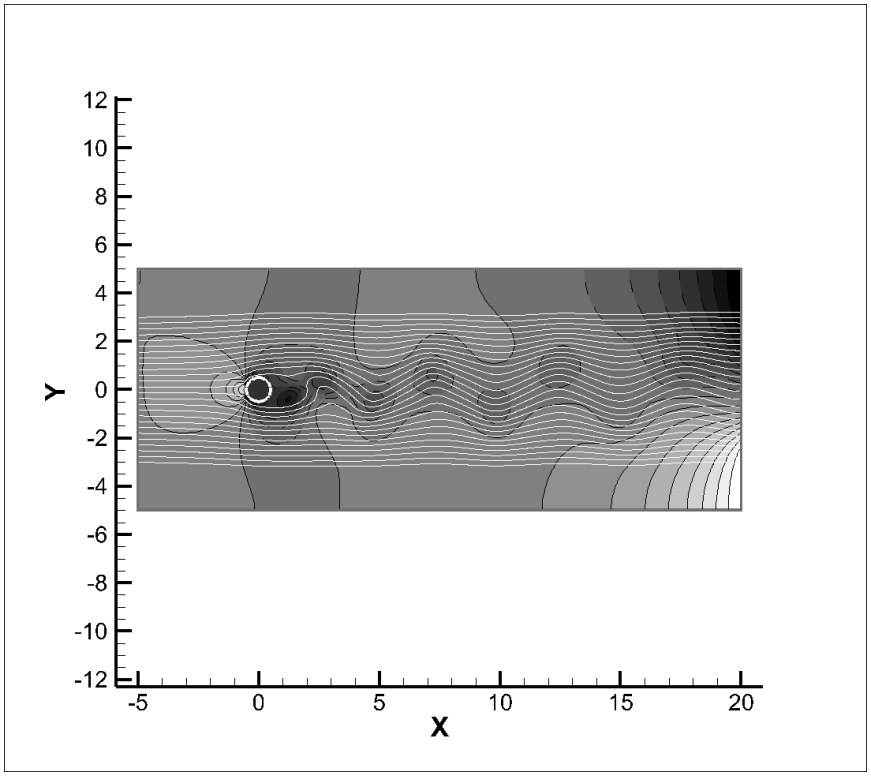
(a) Domain setup for flow past cylinder at $Re = 100$. a, b, c and D as defined in Figure 8.1.

Parameter	Value
a	$5D$
b	$5D$
c	$20D$
D	1.0
$\frac{D}{\Delta x}$	8
$\frac{D}{\Delta y}$	8
T	50
Δt	0.01
Re	100

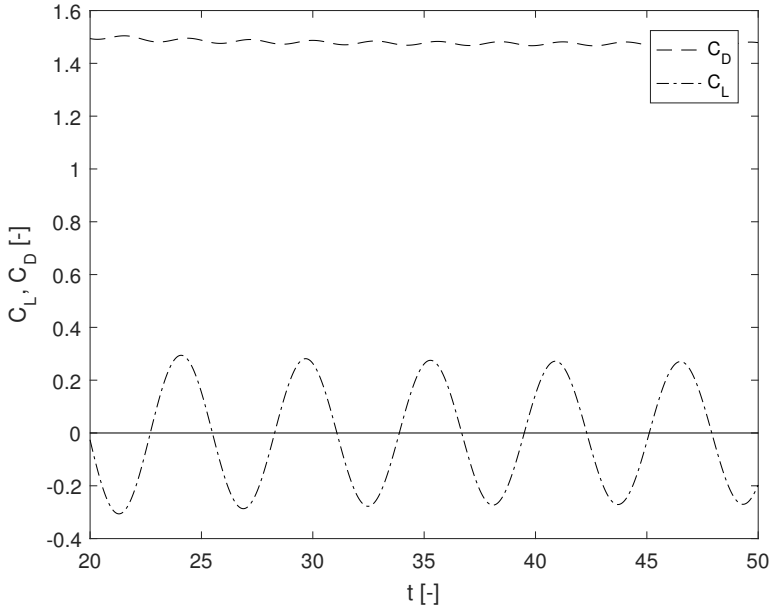
(b) Drag and lift coefficients obtained from the analysis.

Parameter	Value
$\overline{C_D}$	1.475 [−]
$max(C_L)$	0.27 [−]

It is seen from Figure 8.8a that the domain still is too short to avoid the pressure oscillations at the downstream corners, but that the inflow is less affected by the high artificial pressure gradients. The lift coefficient amplitude and the mean drag coefficient were calculated over the last four vortex shedding period (see Figure 8.8b), and they are presented in Table 8.9b. The results are surprisingly within the reference range, but further analyses must be done for a range of mesh and domain sizes before concluding the validation for $Re = 100$.



(a)



(b)

Figure 8.8: Results from analysis presented in Table 8.9a: Pressure contours and streak-lines (a) and force coefficients (b). High pressure gradients are still seen in the downstream corners, which may indicate that the downstream length of the domain is still too short. The effect of the high pressure gradients on the upstream flow is reduced due to the increased distance.

8.9 Concluding remarks

Based on the different testing that has been covered in the previous subsections, it is concluded that:

- The Navier-Stokes-IBM-solver is capable of resembling typical flow phenomena that are anticipated for low Reynolds numbers
- Time resolution is important regarding numerical stability, but has no large effect on the overall accuracy of the results, compared to mesh resolution.
- Domain size is significantly affecting the numerical results
- The results trends towards matching the reference reference studies for a sufficiently spaced domains
- Large transient oscillations are excited during flow startup
- Motion of immersed boundary also produces significant transient oscillations of results
- Time step refinement increases the transient force amplitude, and decreases the transient force persistence.

Based on the list above, the code may be considered valid for stationary cylinders, though further validation is recommended. It is also emphasized that care must be taken regarding the transient oscillations related to motion and flow startup.

Chapter 9

Further development of code: Swimming Fish

The code that was presented in Chapter 6, is not fully developed to handle fluid-structure-interaction problems, as it still lacks a module for handling the structures response to fluid forces. This chapter will discuss the steps that must be taken in order to make the code applicable to solve quasi-rigid (see Section 2.2) swimming fish problems.

9.1 Finishing the IBM-module

The IBM-module presented in Section 6.4 needs to be further developed. First of all, it needs to find not only the translational forces on the immersed boundary, but also the rotational forces. In addition, though the module is designed to handle arbitrary boundary shapes with prescribed motion, it has only been validated for stationary circular cylinders at very low Reynolds numbers.

9.1.1 Concluding validation

Before the IBM-module is equipped with a solver for structural response, it is important to know whether it can replicate the physics of problems involving *prescribed* motions. This will reveal if the immersed boundary conditions are satisfied for non-stationary objects or not. Success in this validation step is an absolute requirement before developing the structural response solver, since the fluid forces are needed as input when calculating the response. Validation should be done both for translational and rotational motion.

Prescribed translational motion

Oscillating cylinders are broadly studied due to the phenomenon of *Vortex Induced Vibrations* (VIV), and a range of the work is devoted to the study of simplified models, such as cylinders with prescribed motions, both in 2D and 3D. It should therefore be an easy task to find reference results for a validation of translational motion. Blackburn and Henderson (1999) and Dütsch et al. (1998) serve as good examples for this.

Prescribed rotational motion

The *Magnus effect* has motivated profound studies of rotating bodies immersed in fluid. A good reference study could be e.g. Kumar and Mittal (2002).

Coupled translational and rotational motion

Since the IBM-module in general is aimed at solving flow problems with arbitrarily shaped boundaries, and especially fish-like shapes, it is tempting to validate the module on a flapping foil system. Also for foils exist a vast amount of work available for reference. Ashraf et al. (2007) cites a large range of references to studies of coupled heaving and pitching motion on a foil placed in uniform flow.

A successful validation on a flapping foil problem will prove that the IBM-module is able to replicate both translational and rotational motion, and that the shape of the immersed boundary does not have to be circular. A pitfall to be aware of is that validation on

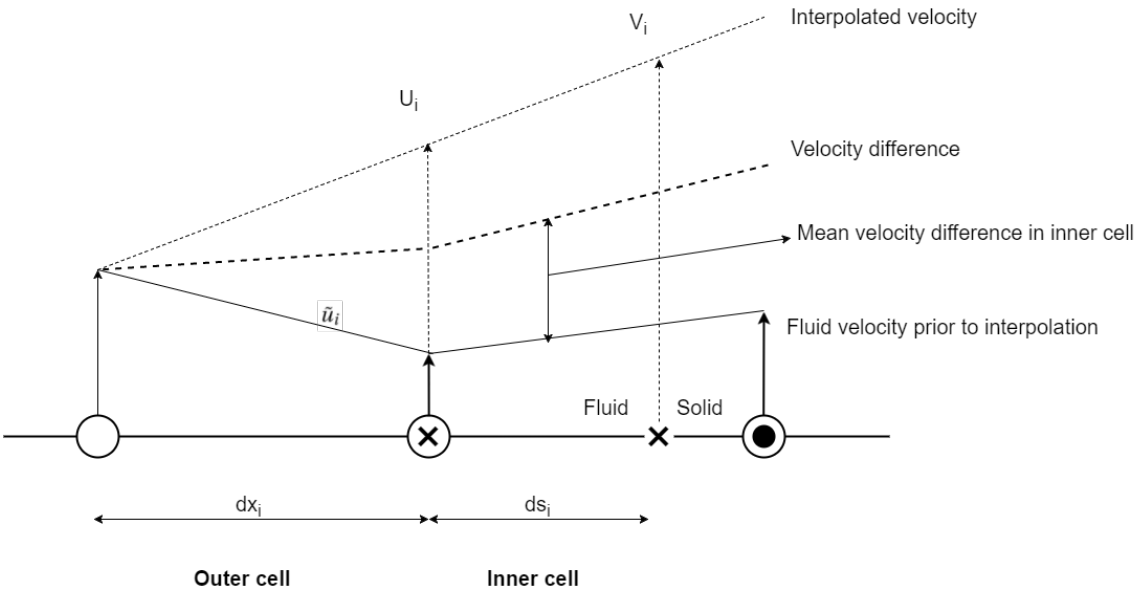


Figure 9.1: An enhanced representation of the flow acceleration presented in Figure 6.14, that may serve as foundation for more accurate force calculations.

such a coupled problem is more likely to fail, and much harder to investigate for errors when validation is not successful.

Improve the force calculations

The method for calculating the force that is presented in Section 6.4 (see Figure 6.14) assumes that the acceleration of the fluid is only defined by the fluid velocities surrounding the interpolation node. This is a coarse and somehow wrong assumption, since the fluid velocity profile between the interpolation node and the immersed boundary should be independent of the internal node velocity. Figure 9.1 shows a more correct picture of the accelerated fluids. The internal node, $\odot$, should only be used to find the distribution of the velocity difference between fluid and immersed boundary in the inner cell. The acceleration of the fluid in both the inner and outer cell should thereafter be calculated between the fluid node, $\circ$, and the intersection point, $\times$, with their respective fluid velocities. In addition, the mass of the inner cell should be found from the fraction that is occupied by fluid, and not from the total area of the cell.

9.2 Development of a structural response solver

The quasi-rigid approach that was discussed in Section 2.2 allows for a separate treatment of the fish's motion, where the the flapping motion is treated as a prescribed, internal motion and the rigid body motion is an external response to the forces applied by the fluid. The fluid forces results from the combined configuration of internal and external motion of the fish.

The structural solver should therefore be able to

- Track the center of mass of the fish by solving the rigid-body equation (2.8)
- Find the local vertex velocities and positions due to rigid-body rotation and flapping motion

9.2.1 Simplifications

For 2D problems, the governing equation for quasi-rigid body motion (see Section 2.1) reads out

$$\begin{gathered} m_{ij}\frac{\partial^2 X_j}{\partial t^2} + \beta_i \frac{\partial m_{ij}}{\partial t}\frac{\partial X_j}{\partial t} = F_i \\ i,j = 1,2,3 \\ \beta_i = \begin{cases} 0, & i = 1,2 \\ 1, & i = 3 \end{cases} \end{gathered} \tag{9.1}$$

where $i = 3$ represents rotation about the x_3 axis (which is perpendicular to x_1 and x_2).

If all motions are defined about the *Center of Mass* (CoM), there will be no coupled mass coefficients in the mass tensor:

$$m_{ij} = 0 \quad \text{for} \quad i \neq j \tag{9.2}$$

and, as the mass is constant, the mass tensor for the 2D quasi-rigid fish becomes:

$$m_{ij} = \begin{pmatrix} m_f & 0 & 0 \\ 0 & m_f & 0 \\ 0 & 0 & I_f(t) \end{pmatrix} \tag{9.3}$$

where m_f is the translational inertia of the fish, and $I_f(t)$ is the moment of inertia about the x_3-axis, which is time-dependent due to the internal flapping deformation of the fish. The time derivative of the mass tensor is simply

$$\frac{\partial(m_{ij})}{\partial t} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{\partial(I_f)}{\partial t} \end{pmatrix} \tag{9.4}$$

Since both m_{ij} and $\frac{\partial(m_{ij})}{\partial t}$ are diagonal, the governing equation (Equation 9.1) may be rewritten as:

$$\begin{aligned} \frac{\partial^2 X_i}{\partial t^2} &= \frac{F_i}{m_f} \quad \text{for} \quad i = 1,2 \quad \text{and} \\ \frac{\partial^2 \theta}{\partial t^2} &= \frac{1}{I_f}\left(T - \frac{\partial I_f}{\partial t}\frac{\partial \theta}{\partial t}\right) \end{aligned} \tag{9.5}$$

where θ represents the rigid body rotation of the fish, and T is the torque about the x_3-axis.

9.2.2 Numerical solution for the rigid body equations

The purpose of finding the rigid-body solver is to track the location and velocity of the fish's center of mass.

For ease of explanation, the tensor subscripts are omitted in the following derivation. Instead one should read capital letters as vectors. Subscript $_{RB}$ and $_{FL}$ denotes rigid-body motion and internal deformation (flapping), respectively.

By substituting

$$\begin{aligned} V_{RB} &= \frac{\partial X}{\partial t} \quad \text{and} \\ \dot{\theta} &= \frac{\partial \theta}{\partial t} \end{aligned} \tag{9.6}$$

into the Equation 9.5, and discretizing it with a the forward Euler method, the following equations are obtained:

$$\begin{aligned} V_{RB}^{n+1} &= V_{RB}^{n} + \frac{\Delta t}{m_f} F^n \\ \dot{\theta}^{n+1} &= \dot{\theta}^n + \frac{\Delta t}{I_f}\left(T^n - \frac{\partial I_f}{\partial t}\dot{\theta}^n\right) \end{aligned} \tag{9.7}$$

These equations yields the translational and angular rigid-body velocity at time step $(n+1)$. The position and rotational angle at the same time step, may readily be as:

$$\begin{aligned} X_{RB}^{n+1} &= X_{RB}^{n} + \Delta t V_{RB}^{*} \\ \theta^{n+1} &= \theta^{n} + \Delta t \dot{\theta}^{*} \end{aligned} \tag{9.8}$$

Here, the asterix, $(\cdot)^*$, is used to show that many estimates of the slope are available at this step of the calculations, such as:

$$V_{RB}^{*} = \begin{cases} V_{RB}^{n} \\ V_{RB}^{n+1} \\ 0.5\left(V_{RB}^{n} + V_{RB}^{n+1}\right) \end{cases} \tag{9.9}$$

and similar for $\dot{\theta}$.

The equations above could be implemented into a subroutine called *RB-solver*.

9.2.3 Adding the flap and finding the total motion

The internal flapping motion of the fish is given by a prescribed motion model (discussed in Section 9.3), and may be defined vertex-wise about the CoM of the fish:

$$\begin{aligned} X_{FL} &= \begin{bmatrix} x_1 & \cdots & x_c & \cdots & x_{N_{vert}} \\ y_1 & \cdots & y_c & \cdots & y_{N_{vert}} \end{bmatrix} \\ V_{FL} &= \begin{bmatrix} U_1 & \cdots & U_c & \cdots & U_{N_{vert}} \\ V_1 & \cdots & V_c & \cdots & V_{N_{vert}} \end{bmatrix} \end{aligned} \tag{9.10}$$

The total motion of the fish is obtained by combining the internal motion with the rigid body response. Since the results from the RB-solver only are defined in the center of mass, they must be translated into local position and velocity at each vertex.

The total position at each vertex, c, may thereby be found by

$$X|_c = X_{RB} + R(\theta) X_{FL}|_c \tag{9.11}$$

Where $X_{FL}|_c$ is the flapping position of vertex c, and $R(\cdot)$ is the rotation matrix, both defined about the CoM.

The total velocity at vertex c may be written as

$$V|_c = V_{RB} + \dot{\theta} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} X_{FL} + V_{FL} \tag{9.12}$$

where the second term of the RHS is the local vertex velocity due to the rotational velocity of the fish, while the last term is the internal flapping velocity.

9.3 Development of a swimming fish model

As discussed in the previous section, the quasi-rigid approach allows for a prescribed internal motion of the fish. The design of this motion is discussed in the next subsections.

9.3.1 Physical requirements

An important requirement for the fish model is that it satisfies the constitutive laws of physics. Recall the discussion in Section 2.2:

- mass should be conserved throughout deformation
- deformation should not directly accelerate the center of mass

For the 2D case with uniform density, the mass conservation criterion reduces to "area should be conserved". The requirement of no acceleration means that the motion model should replicate the fish's motion as if it was flapping in vacuum.

9.3.2 Choice of swimming mode

The fish model should be designed to mimic the natural motion of a fish. Sfakiotakis et al. (1999) presents a review of the different swimming modes found in aquatic locomotion. For fishes, the modes are grouped into *Median and/or Paired Fin* (MPF) modes, and *Body Caudal Fin* (BCF) modes, based on which parts of the body is used for propulsion (see Figure 9.2). The BCF modes are the ones best suited for propulsion efficiency, while MPF propulsion scores higher on low speed maneuverability.

As stated in the introduction (Chapter 1), the motivation for this thesis is moored in the utilization of flapping foils for effective propulsion. The fish model should therefore be designed as a BCF-type propulsor. Figure 9.3 shows two of the subgroups of BCF-type swimming modes. The oscillatory/undulatory motion of the anguilliform swimmer should be nicely mimicked by a harmonic function:

$$X(x,t) = \gamma(x)\sin(2\pi f t - kx) \tag{9.13}$$

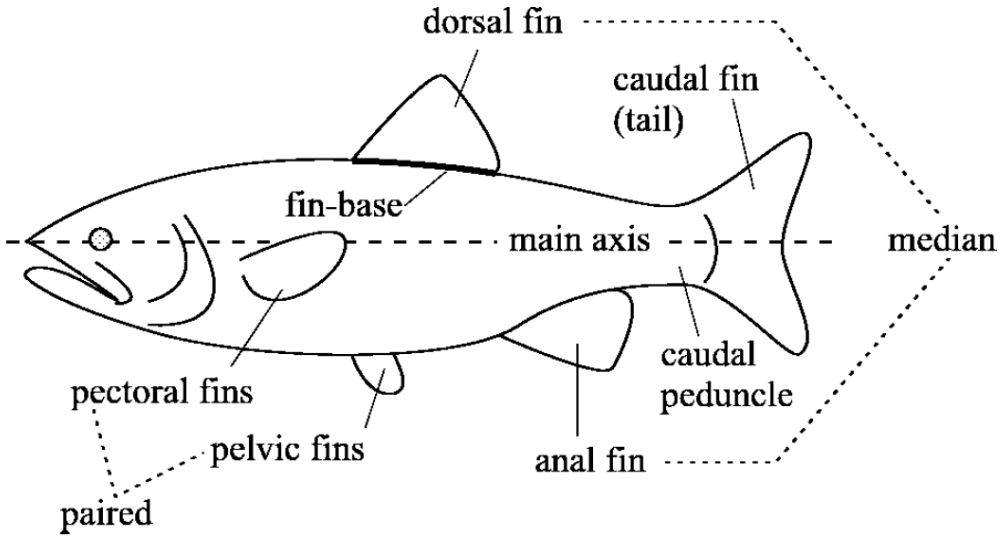


Figure 9.2: Terminology of the fish body. (From Sfakiotakis et al. (1999)).

where X represents the centerline of the fish, f is the flapping frequency in hertz, and k represents the wave number for the undulatory motion. $\gamma(x)$ represent a spatially dependent damping function used to tune the head-amplitude of the fish. Fully anguilliform mode is modeled by zero damping, that is $\gamma(x) = 1$.

9.3.3 Startup function

A sudden startup of the fish motion will give transient oscillations such as those discussed in Section 8.7. Such transients may yield an enormous initial force on the fish, causing it to drift out of the domain in just few time steps. To avoid this, it is convenient to equip the motion model with some smooth startup function, say $s(t)$. Munnier and Pinçon (2008) use the following function and its derivatives to smooth the startup of their ideal-fluid fish model:

$$s(t) = \begin{cases} 140(\tau^4(0.25 + \tau(-0.6 + \tau(0.5 - \tau/7.0)))) & \text{for } \tau \leq 1 \\ 1.0 & \text{elsewise} \end{cases} \tag{9.14}$$

where $\tau = t/t_0$ and t_0 is the time for which full amplitude should be reached.

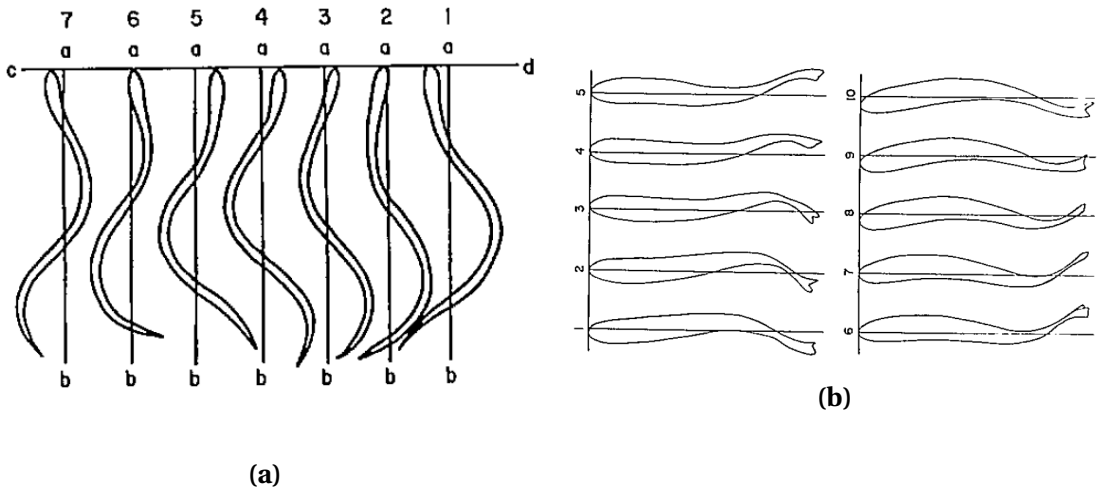


Figure 9.3: Subgroups of BCF propulsion modes: Anguilliform (a) and subcarangiform (b). (Modified, from Webb (1971)).

Another option is to use the *sigmoid function*, defined by:

$$s(t) = \frac{1}{1 + \exp -(t - t_0)} \tag{9.15}$$

9.3.4 General mathematical model

The following, general model is equipped with both a smooth startup function ($s(t)$) and a tuning function for spatial distribution of amplitude ($\gamma(x)$).

$$X(x,t) = \gamma(x)s(t)\sin(2\pi f t + kx) \tag{9.16}$$

$$U(x,t) = \gamma(x)\left(2\pi f s(t)\cos(2\pi f t + kx) + \frac{\partial s(t)}{\partial t}\sin(2\pi f t + kx)\right) \tag{9.17}$$

Table 9.1 shows how the different parameters may be used to tune the model into different swimming modes:

Table 9.1: Tuning of the fish motion model into different body-caudal fin subgroups.

Parameter	Anguilliform	Subcarangiform
k	≥ 1	≤ 1
$\gamma(x)$	≈ 1	$\propto e^{x-L}$

9.3.5 Design of a BCF mode swimming body

A very simplistic model of a fish may be constructed using a semi-circle for the head, and two harmonic half-waves for the body:

Head

$$X_h = \begin{bmatrix} r_h cos(\alpha) \\ r_h sin(\alpha) \end{bmatrix} \quad \text{for} \quad \alpha \in \left(\frac{\pi}{2}, 3\frac{\pi}{2}\right) \tag{9.18}$$

Where r_h is the head radius.

Body

$$X_b = \begin{bmatrix} x \\ \pm \quad r_h cos(\frac{\pi}{2L_b} x) \end{bmatrix} \tag{9.19}$$

Where L_b is the length of the fish without its head.

A discretized version of the fish may thereafter be described as the closed simplex polygon:

$$X_f = \begin{bmatrix} x_1 & \cdots & x_{N_{vert}} & x_1 \\ y_1 & \cdots & N_{vert} & y_1 \end{bmatrix} \tag{9.20}$$

where x_1 and y_1 are defined from the tail tip, and the point definition is done counter-clockwise. This reference fish-model, which is schematically presented in Figure 9.4, is defined in a coordinate system with the origin in the joint between the fish head and body.

Figure 9.4: A simple 2D fish model. A vertex pair is defined at each x-position, also at the tail where the two last vertexes are in the same position.

Adding the motion model to the polygon

Since the vertexes are pair-wise defined at each x-location, the body element between two consecutive vertex pairs will be a trapezoid. The area of a trapezoid is conserved as long as the parallel sides remain parallel and the normal distance between them is constant. A possible motion model could therefore be obtained by, using the mathematical model from Subsection 9.3.4, shifting all vertex points in the direction perpendicular to the local x-direction of the fish. This approach is shown in Figure 9.6a.

Another possibility is to rotate the parallel axes by some offset in the x-direction as shown in Figure 9.5, and let the offset be governed by the mathematical motion model (Equation 9.17). It may be shown by geometrical deduction that, for a single trapezoid, such rotation does not alter the area. There is one issue, however: The rotation extends the width of the trapezoid sides, so that a discontinuity occurs between the trapezoids. This may be counteracted for by changing the trapezoid length.

The assembly of the rotational deformation modes can in brief be described as:

1. Deform the first trapezoid closest to the head
2. Scale and rotate it so that the widest side matches the width of the head
3. Alter the length of the trapezoid so that area is conserved
4. Assemble trapezoid onto head
5. Deform the next trapezoid
6. Scale and rotate it so that the widest side matches the previous trapezoid
7. Contract the length of the trapezoid so that area is conserved

8. Assemble trapezoid onto head

9. Repeat item 5-8 until end of fish

Figure 9.6b shows a motion model where the trapezoids are deformed by an offset.

The other requirement for the fish model was that there should be no net acceleration of the fish's center of mass. Both of the deformation models described in Figure 9.6 may induce a change of the center of mass. This is counteracted by shifting the position of the deformed shape so that the origin lies in the new center of mass. Any net rotation may be found from torque considerations of the fish, and then be subtracted from the deformed flapping fish before the deformed fish polygon is passed to the structural solver.

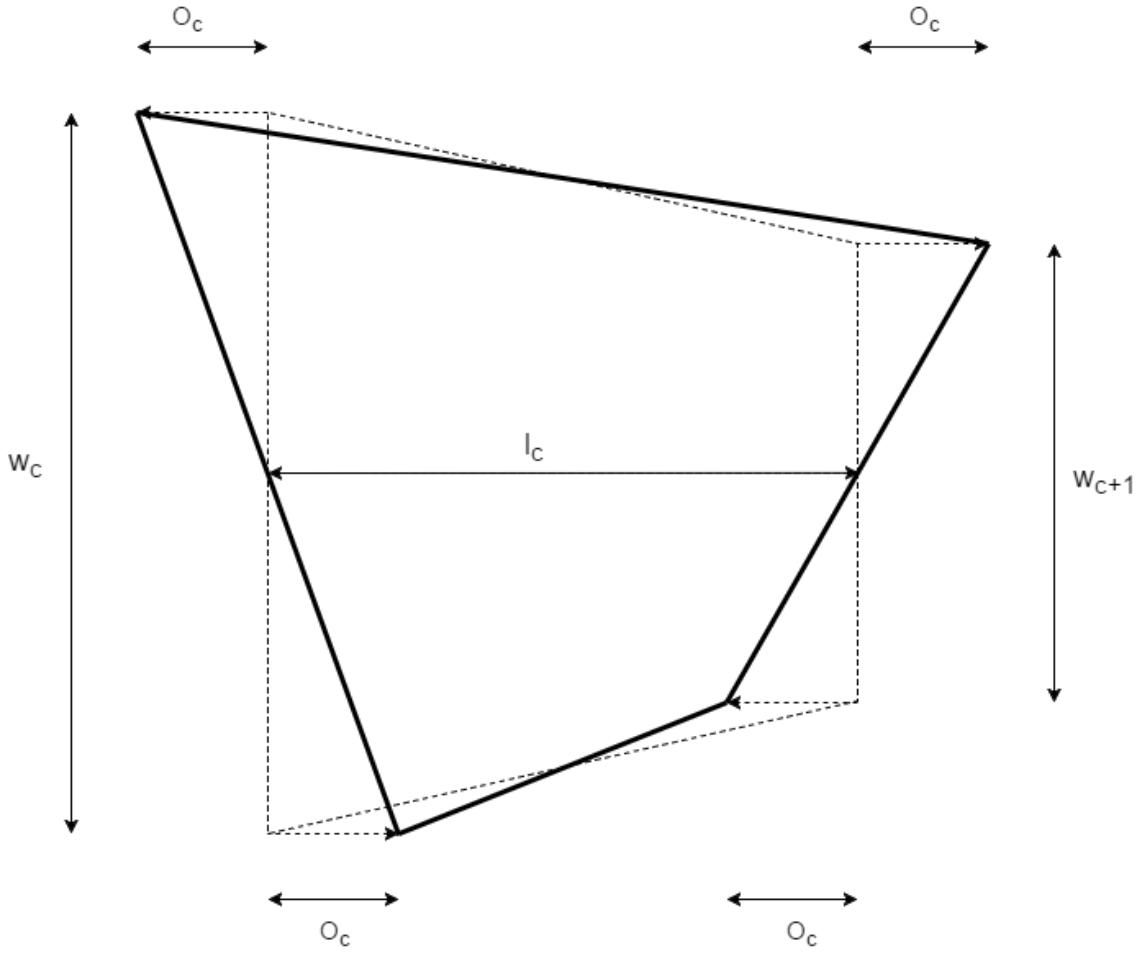
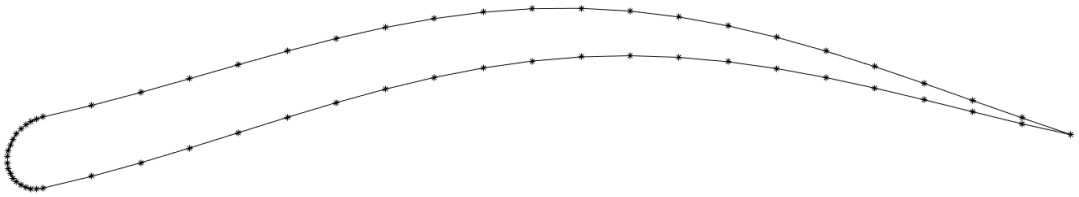
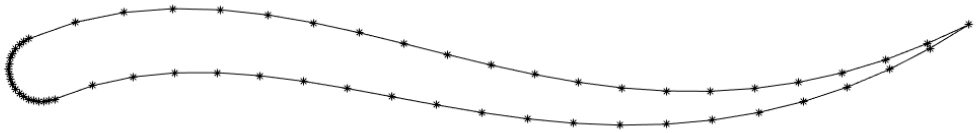


Figure 9.5: Deformation of trapezoid c by an offset O_c.



(a)



(b)

Figure 9.6: The parallel shifted trapezoid model (a) and the rotated trapezoid model (b). Both models proves to conserve area throughout the deformation.

9.4 Proposal for a fluid-structure-interaction code

Figure 9.7 shows how a structural solver and a motion model may be implemented into the existing IBM-NS-solver. The subroutine RB-solver needs information about the rigid-body configuration of the fish at each time step, and this information is stored in the array

$$\mathbf{b}_{RB} = \begin{bmatrix} x_{RB} \\ y_{RB} \\ V_{x,RB} \\ V_{y,RB} \\ \theta \\ \dot{\theta} m_f \\ I_f \\ \frac{\partial I_f}{\partial t} \end{bmatrix} \tag{9.21}$$

With this information, it tracks the rigid-body response by solving Equation 9.8 and 9.7.

After rigid-body-tracking, the subroutine FLAP applies the flapping motion, and finds the total vertex motions from Equation 9.11 and 9.12. The total motion is stored into the the immersed boundary polygon array which is defined as before (see Equation 6.11).

9.4.1 Anticipated stability issues

The proposed FSI-code is a loosely coupled model (see Section 3.8), which means that the interaction forces are calculated directly at each time step, and not by iteration. As seen in Section 8.7, the direct forcing approach creates large transient oscillations during start-up of flow. Similar effects are seen to arise from the movement of the immersed boundary from one position fo another, even in stationary fluids. If the forces are calculated only once per time step, this means that the interaction forces may be heavily over-estimated, which again will lead to a major acceleration of the fish, and finally a large translation of the fish for the following time step. For the next time step, the same will happen, but with a larger initial value, so the time integration may eventually become unstable. To counteract for this, one could take the time step very small, but as seen in Section 8.7, the max amplitude of the transient effect increases with decreasing time step. The best way to counteract for instability would be to calculate the

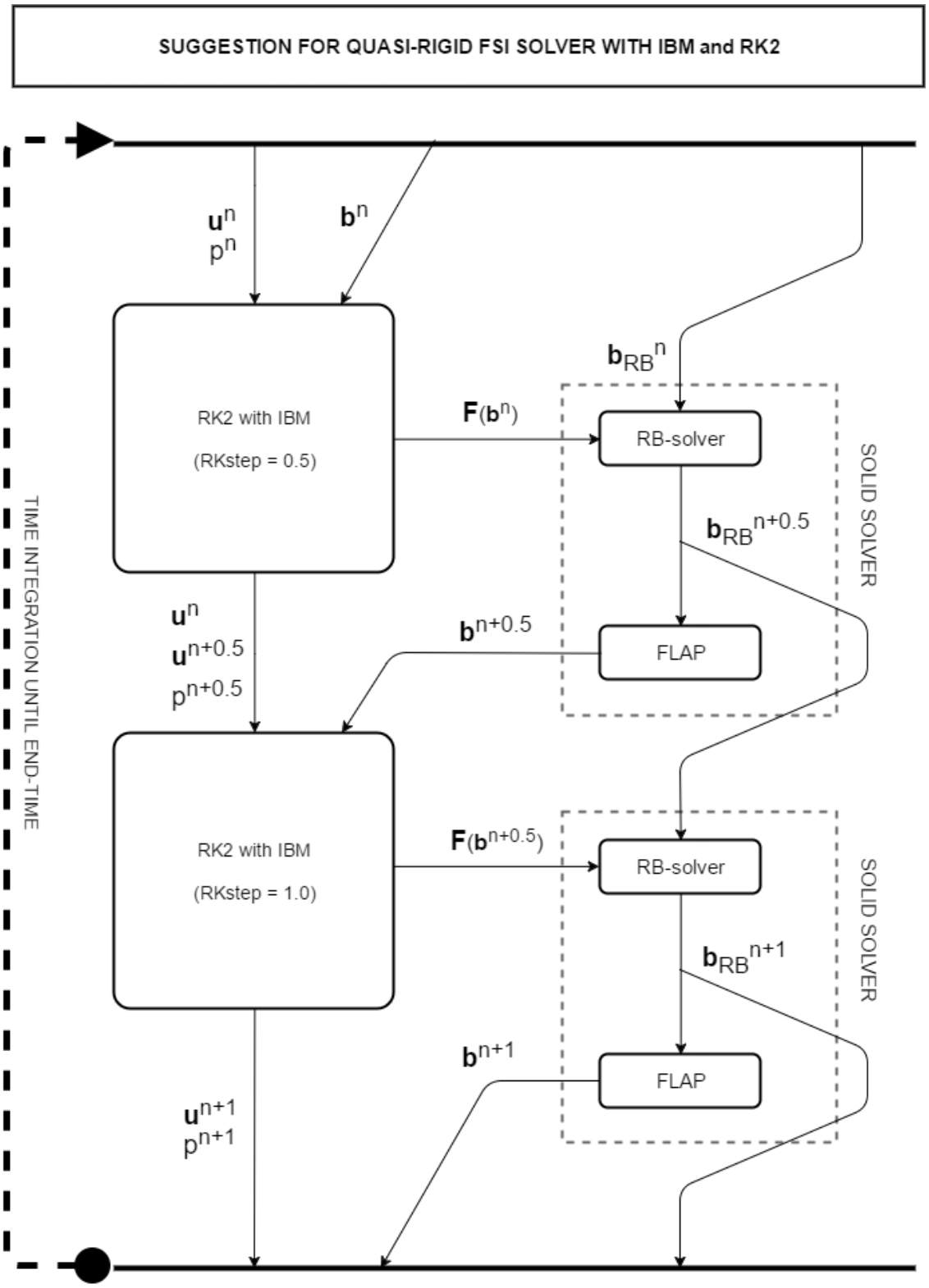


Figure 9.7: Proposal for an FSI-solver for a quasi-rigid fluid-structure problem. **u** represents the velocity field, p the pressure field and **F** the immersed boundary force. The immersed boundary configuration, **b**, is separated into rigid body motion tracking ($\mathbf{b}_{RB}$), and an internal flapping motion, following Equation 9.10, 9.11 and 9.12.

interaction forces by an iterative procedure at each time step, allowing for the transient oscillations to calm before propagating the time integration. Such an iterative solver would fall within the group of strong coupled FSI-solvers.

Chapter 10

Conclusions

A motivation for improving ship propulsion by mimicking a fish's swimming strategy has led to the work with this thesis. Background theory for the field of *fluid-structure-interaction* within computational fluid dynamics has been presented, and the main focus has been on the *immersed boundary methods*, since these are capable of handling arbitrarily complex body shapes without large computational impact.

A 2D direct forcing immersed boundary method, based on the work of Fadlun et al. (2000), has been developed and implemented into a Navier-Stokes solver. The solver can take in arbitrarily shaped simple polygons with individual nodal velocities, and solve for the flow field that will develop in the domain with the polygon present.

Validation of the numerical code has been done for stationary cylinders at $Re = 20$, and results from simulations of higher Reynolds numbers shows that the code also is capable of resembling vortex shedding. The code is yet to be validated for polygons with both rotational and translational prescribed motion.

From plots of the force coefficients, it is seen that large transient force oscillations occur at flow startup. The effect of this decays rapidly, but the largest value is of order $O(10^2)$ relatively to the stationary value. This is important to be aware of when implementing the code into a fluid-structure solver, since the forces are used to calculate structural responses.

Further development of the IBM-NS-solver into a full FSI-solver is discussed. The proposed solver uses a quasi-rigid approach which means that the fish is only deformable

from internal forces. A simple mathematical fish model is developed based on the principles of body-caudal fin propulsion, and a flow-chart for a loosely coupled FSI-solver is designed. The large force transients connected with the direct forcing approach strongly suggest that a strong coupling FSI-model be used instead, as a loose coupling fluid-structure-model will be subject to large numerical inaccuracies and instability.

Chapter 11

Recommendations for further work

This chapter provides recommendations for future research and other activities related to the topics of this thesis. The recommendations are based on the experience and knowledge that is gained through the work.

11.1 Short term recommendations

The code developed in this thesis will not outperform commercial software, but a further development of the code could serve as a good basis for educational activities. Such activities could involve experimentation with

- different Poisson solvers
- higher order interpolation schemes for the immersed boundary method
- comparison between normal and parallel interpolation

and the code could be used by students to study flapping foil propulsion in e.g. ship design projects. Another interesting project could be to expand the code into a full fluid-structure-interaction solver, which would provide students with programming-level knowledge about CFD and the problems related to numerical instabilities in fluid-structure-interaction problems. In such a project, both a strong and loose fluid-structure coupling scheme should be explored.

Prior to the uses mentioned above, the code should be properly validated, both for stationary and moving objects.

11.2 Long term recommendations

Since the IBM-method has proven to be versatile for flows around complex geometries, an attempt on developing a code for 3D applications should be done.

Many studies has been done on the area in the latest decade, specifically addressing many of the different issues related to IBMs. A profound study of these recent contributions should be made prior to the design of a 3D IBM-module, both to avoid re-exploring old science and to find unresolved issues to address and investigate. For example could it be interesting to assess the following question: *Can IBMs be made applicable to fluid-structure-interaction problems where the structure is close to, or piercing, a fluid surface?*. Such an IBM could support interesting analyses of fish swimming near the surface, or foil propulsion with surface piercing foils.

Due to the sizable data sets that are connected with CFD on 3D flows, the development of a 3D IBM-module should focus on efficiency. The module could be implemented with an existing high-performance N-S solver, or one could design a new N-S solver for the purpose. Such solver could for example combine Adaptive Mesh Refinement, a multigrid-based Poisson solver and parallel processing to optimize the efficiency.

References

Anderson Jr., J. D. (2009). *Computational Fluid Dynamics*. Springer-Verlag Berlin Heidelberg. Chapter 2 of the 3rd edition.

Ashraf, M. A., Lai, J. C. S., and Young, J. (2007). Numerical analysis of flapping wing aerodynamics. In *16th Australasian Fluid Mechanics Conference*.

Balaras, E. (2004). Modeling complex boundaries using an external force field on fixed cartesian grids in large-eddy simulations. *Computers & Fluids*, 33:375–404.

Blackburn, H. M. and Henderson, R. D. (1999). A study of two-dimensional flow past an oscillating cylinder. *J. Fluid Mech.*, 385:255–286.

Braza, M., Chassaing, P., and Ha Minh, H. (1986). Numerical study and physical analysis of the pressure and velocity fields in the near wake of a circular cylinder. *J. Fluid Mech.*, 165:79–130.

Calhoun, D. (2002). A cartesian grid method for solving the two-dimensional streamfunction–vorticity equations in irregular regions. *Journal of Computational Physics*, 176:231–275.

Carlton, J. (2012). *Marine Propellers and Propulsion*. Elsevier Ltd., 3rd edition.

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

Clift, R., Grace, J. R., and Weber, M. E. (1978). *Bubbles, Drops and Particles*. Academic Press.

Dennis, S. C. R. and Chang, G.-Z. (1970). Numerical solutions for steady flow past a circular cylinder at reynolds numbers up to 100. *J. Fluid Mech.*, 42:471–489.

Dütsch, H., Durst, F., Becker, S., and Lienhart, H. (1998). Low-reynolds-number flow around an oscillating circular cylinder at low keulegan-carpenter numbers. *J. Fluid Mech.*, 360:249–271.

Fadlun, E. A., Verzicco, R., Orlandi, P., and Mohd-Yusof, J. (2000). Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *Journal of Computational Physics*, 161:35–60.

Ferziger, J. H. and Péric, M. (2002). *Computational Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg New York, 3rd edition.

Fish, F. E. and Rohr, J. J. (1999). *Review of Dolphin Hydrodynamics and Swimming Performance*. Spawar Systems Center, United States Navy.

Ghia, U., Ghia, K. N., and Shin, C. T. (1982). High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method*. *Journal of Computational Physics*, 48:387–411.

Gilmanov, G. and Sotiropoulos, F. (2005). A hybrid cartesian/immersed boundary method for simulating flows with 3d, geometrically complex, moving bodies. *Journal of Computational Physics*, 207:457–497.

Godø, J. M. (2017). Personal communication.

Guy, R. D. and Hartenstine, D. A. (2009). On the accuracy of direct forcing immersed boundary methods with projection methods. Preprint submitted to Journal of Computational Physics.

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

Hou, G., Wang, J., and Layton, A. (2012). Numerical methods for fluid-structure interaction - a review. *Commun. Comput. Phys.*, 12:337–377.

Khoo, B. C., Le, D. V., and Lim, K. M. (2008). An implicit-forcing immersed boundary method for simulating viscous flows in irregular domains. *Comput. Methods Appl. Mech. Engrg.*, 197:2119–2130.

Kumar, B. and Mittal, S. (2002). Flow past a rotating cylinder. *J. Fluid Mech.*, 476:303–334.

Lomax, H., Pulliam, T. H., and Zingg, D. W. (1999). *Fundamentals of Computational Fluid Dynamics*. University of Toronto Institute for Aerospace Studies.

Lumley, J. L. (1969). Eulerian and lagrangian descriptions in fluid mechanics. Notes for educational film, by National Commitee for Fluid Mechanics Films.

Manning, S. F. (1991). How to scull a boat. *Wooden Boat*, 100:60–68.

Mark, A., Rundqvist, R., and Edelvik, F. (2011). Comparison between different immersed boundary conditions of complex fluid flows. *FDMP*, 07:241–258.

Mittal, R., Dong, H., Bozkurttas, M., Najjar, F. M., Vargas, A., and von Loebbecke, A. (2008). A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *Journal of Computational Physics*, 227:4825–4852.

Mittal, R. and Iaccarino, G. (2005). Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261.

Mohd-Yusof (1997). *Combined immersed boundary/B-spline methods for simulations of ow in complex geometries*. Center for Turbulence Research. Annual Research Briefs 1997.

Munnier, A. and Pinçon, B. (2008). Biohydrodynamics toolbox. `http://bht.gforge.inria.fr/Doc/biohydrodynamics_toolbox_index.html`. [Online, accessed 29-January-2016].

Park, J., Kwon, K., and Choi, H. (1998). Numerical solutions of flow past a circular cylinder at reynolds numbers up to 160. *KSME International Journal*, 12:1200–1205.

Peskin, C. S. (1972). Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10:252–271.

Peskin, C. S. (2002). The immersed boundary method. *Acta Numerica*, pages 479–517.

Russel, D. and Wang, Z. J. (2003). A cartesian grid method for modeling multiple moving objects in 2d incompressible viscous flow. *Journal of Computational Physics*, 191:177–205.

Sfakiotakis, M., Lane, D. M., and Davies, B. C. (1999). Review of fish swimming modes for aquatic locomotion. *IEEE JOURNAL OF OCEANIC ENGINEERING*, 24.

Silva, A. L. F. L. E., Silveira-Neto, A., and Damasceno, J. J. R. (2003). Numerical simulation of two-dimensional flows over a circular cylinder using the immersed boundary method. *Journal of Computational Physics*, 198:351–370.

Sotiropoulos, F. and Yang, X. (2014). Immersed boundary methods for simulating fluid-structure interaction. *Progress in Aerospace Sciences*, 65:1–21.

Stenvold, T. (2016). Halefinnebevegelse kan revolusjonere hurtigbåtmarkedet. http://www.tu.no/artikler/halefinnebevegelse-kan-revolusjonere-hurtigbatmarkedet/366047. [Online, accessed 5-Jan-2017].

Sucker, D. and Brauer, H. (1975). Fluiddynamik bet quer angeströmten zylindern. *Wärme- und Stoffübertragung*, 8:149–158.

Tannehill, J. C., Anderson, D. A., and Pletcher, R. H. (1997). *Computational Fluid Mechanics and Heat Transfer*. Taylor & Francis, 2nd edition.

©Fluent (2006). Ansys ®fluent ®6.3 user's guide. `https://www.sharcnet.ca/Software/Fluent6/html/ug/main_pre.htm`. [Online, accessed 31-January-2017].

Tritton, D. J. (1959). Experiments on the flow past a circular cylinder at low reynolds number. *J. Fluid Mech.*, 6:547–567.

Uhlmann, M. (2005). An immersed boundary method with direct forcing for the simulation of particulate flows. *Journal of Computational Physics*, 209:448–476.

Wang, Z. and Cen, K. (2009). Immersed boundary method for the simulation of 2d viscous flow based on vorticity–velocity formulations. *Journal of Computational Physics*, 228:1504–1520.

Webb, P. W. (1971). *HYDRODYNAMICS AND ENERGETICS OF FISH PROPULSION*, volume 190. Bulletin of the Fisheries Research Board of Canada.

White, F. M. (2006). *VISCOUS FLUID FLOW*. McGraw Hill.

Xu, S. and Wang, Z. J. (2006). An immersed interface method for simulating the interaction of a fluid with moving boundaries. *Journal of Computational Physics*, 216:454–493.

Ye, T., Mittal, R., Udaykumar, H. S., and Shyy, W. (1999). An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries. *Journal of Computational Physics*, 156:209–240.

Appendix A

Tensor notation

In fluid- and solid mechanics, we often consider quantities that are associated with directions. Kinematic quantities like velocity and acceleration, have **one** direction, and scalars have **none**. In the Navier Stokes equation, the advective acceleration term $((\vec{u} \cdot \nabla)\vec{u})$ is associated with two directions simultaneously: Both the direction of advectional transport $(\vec{u} \cdot \nabla)$, and the direction of the transported velocity $(\vec{u})$. This is not seen very clearly from equations written with vector notation, as there is no distinction between the two velocity vectors in the term. To increase simplicity and readability of the equations it is convenient to use tensor notation.

General idea

Tensor notation is the general way of representing spatial mathematical objects. A scalar quantity, as it has no specific direction, is a tensor of rank **zero**. A vector, which is a quantity represented in one direction, is likewise a tensor of rank **one**, and a rank **two** tensor is equivalent to a $N \times N$ matrix associated with two directions, where (N) is the number of dimensions in the space considered.

The general idea is based on that, in an N-dimensional space, one single direction must be described by N different terms: one term representing the direction's extension along each dimension. In this space, a tensor A of rank m, i.e. a tensor that is associated with m directions simultaneously, will be represented with m indexes. As an example, a

tensor of rank 2 in a 3-dimensional space will be represented as:

$$A_{ij} \tag{A.1}$$

Where each of the indexes i, j ranges independently over the three dimensions:

$$\begin{aligned} i &= 1,2,3 \\ j &= 1,2,3 \end{aligned} \tag{A.2}$$

The quantity A is related to two directions, and the tensor A_{ij} contains all the possible combinations of the three dimensions to describe two directions at the same time:

$$A_{ij} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \tag{A.3}$$

Each of the entries in the tensor A.3 represent a combination of two directions, where each of the directions extends along one of the axes in the system. In 3D, we name the axes x_1, x_2, x_3, which is equivalent to the conventional x, y, z-representation. Similarily, the elementary unit vectors are denoted $e_1, e_2 e_3$.

For terms containing two quantities (A, B) associated with independent directionsi, j, the tensor representation is:

$$A_i B_j = \begin{pmatrix} A_1 B_1 & A_1 B_2 & A_1 B_3 \\ A_2 B_1 & A_2 B_2 & A_2 B_3 \\ A_3 B_1 & A_3 B_2 & A_3 B_3 \end{pmatrix} \tag{A.4}$$

Generally, in an N-dimensional space, a tensor of rank m will contain N^m terms, and represents a mathematical object that is associated with m directions. Some examples from the physical world can be found in table A.1.

Table A.1: Physical quantities and their tensor rank

Quantity	Tensor rank	Associated directions
Pressure	0	not associated with any direction
Velocity	1	direction of action
Stress	2	direction of action, orientation of the surface acted upon
Advective acceleration	2	direction of advectional transport, direction of transported velocity

Einstein's summation convention

When indexes are repeated in different factors of a tensor term, the tensor should be read as a sum over the whole range of this index. Eq. A.5 first shows a general sum in an N-dimensional space, and then the advective acceleration in the 3-dimensional space written on tensor form.

$$\begin{aligned} A_i B_i &= \sum_{i=1,2,..,N} (A_i B_i) \\ u_j \frac{\partial u_i}{\partial x_j} &= \sum_{j=1,2,3} (u_j \frac{\partial u_i}{\partial x_j}) \end{aligned} \tag{A.5}$$

This convention increases the simplicity of equations, still containing the exact same information, as demonstrated by the incompressible continuity equation:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{A.6}$$

Equivalence to vector- and matrix notation

A perk of the Einstein summation convention is that it provides the following equivalent expression for a general matrix equation:

$$\begin{gathered} \mathbf{A}\vec{x} = \vec{b} \\ \text{is equivalent to} \\ A_{ij} x_j = b_i \end{gathered} \tag{A.7}$$

The identity matrix, **I**, from vector notation is resembled by the *Kroenecker delta*-symbol:

$$\mathbf{I} = \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{A.8}$$

It should be noted that the two representations are **not** identical. The tensor notation is always referring to one scalar value, but in the context of the other values of the tensor. The identity matrix, on the other hand, is denoting all the values of the matrix simultaneously.

The equivalent of the cross product $(\times)$ from vector notation makes use of the *Levi-Cevita*-symbol:

$$\epsilon_{ijk} = \begin{cases} 0, & \text{if any two of } (i,j,k) \text{ are the same} \\ 1, & \text{if } (i,j,k) \text{ is an even permutation of } (1,2,3) \\ -1, & \text{if } (i,j,k) \text{ is an odd permutation of } (1,2,3) \end{cases} \tag{A.9}$$

And the cross product between two variables is thus denoted:

$$\vec{a} \times \vec{b} = \epsilon_{ijk} e_i a_j b_k \tag{A.10}$$

In table A.2, all vector operators that are of concern in fluid mechanics are presented with their associated tensor equivalent.

Table A.2: Common vector operators and their equivalent in tensor notation

Operator	Vector notation	Tensor notation
Gradient	$\nabla(\cdot)$	$\frac{\partial(\cdot)}{\partial x_i}$
Divergence	$\nabla \cdot (\cdot)$	$\frac{\partial(\cdot)_i}{\partial x_i}$
Laplacian	$\nabla^2(\cdot)$	$\frac{\partial^2(\cdot)}{\partial {x_j}^2}$
Advection	$(\vec{u} \cdot \nabla)(\cdot)$	$u_j \frac{\partial(\cdot)}{\partial x_j}$

Appendix B

Description of motion

In mechanics, it is distinguished between Lagrangian and Eulerian descriptions of motion. The Lagrangian description displays motion as a function of time, and the position of a given quantity may be represented by a vector $\vec{p}(t)$ from which velocity and acceleration may be calculated (Lumley (1969)). This is a suitable method to describe the motion of single particles or solids, but each point must be traced individually, such that once large amounts of points are considered the representation becomes incomprehensible. Thus, for representation of e.g. fluid fields, there is a need for another description. The Eulerian description of motion is not linked to specific particles. Instead, it shows information of the behavior of any particle at some specific location at a specific time.

The representation of velocity from the two descriptions is mathematically written:

- $u_i(t)$ (Lagrangian)
- $u_i(x, y, z, t)$ (Eulerian)

Postface

Through the process of program development, I have learned some things the hard way. Here are some reflexions that I have made in the hindsight of the work with this thesis, which may provide an advantage for myself or any other student attempting to write a CFD code in the future.

I feel like I started in the wrong end with the program design process. With my limited knowledge within CFD, I should of course have started by developing the Navier-Stokes solver. Then I would rapidly have learned how big a time-thief the iterative Poisson solver is, compared to other operations performed in a CFD-algorithm. Instead, I started designing the IBM-module, and I spent lots of hours focusing on creating an efficient code. I think the IBM-module could have been much easier to program if I had used an interpolation algorithm that searched for intersections by scanning through the Eulerian mesh, instead of through the immersed boundary polygon.

Another experience is regarding *the choice* of the Poisson solver. The benefit of using the Gauss-Seidel method was that I could easily construct my own code, and thereby gain some understanding about the principles of iterative solvers. On the other hand, if I had used some black-box high performance iterative solver, I would have been able to validate my code to a greater extent, and maybe even perform simulations on flapping fish models.

Trondheim, March 1, 2017

Jon Coll Mossige