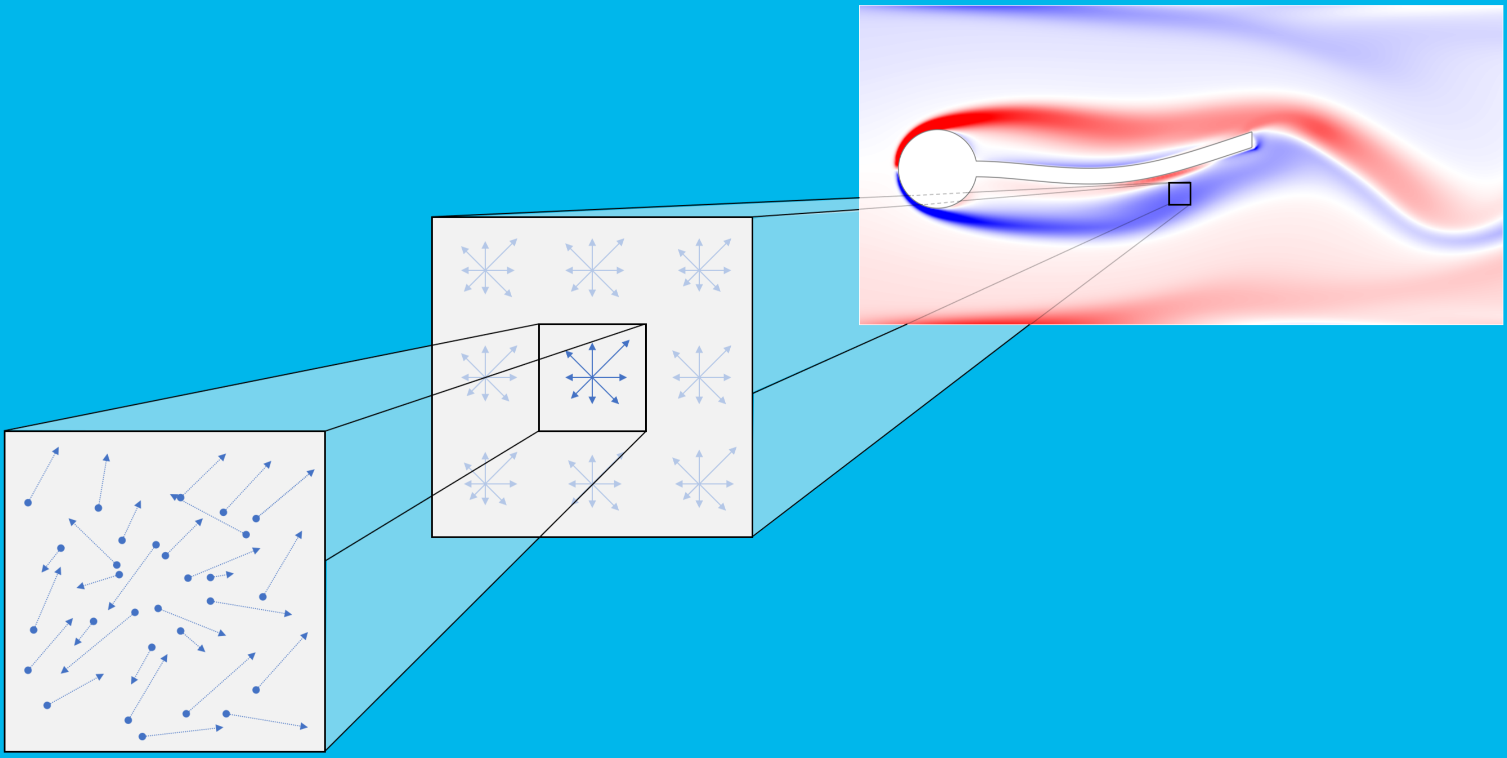
Proposal and Validation of an Immersed Interface Method applied to the Lattice Boltzmann Method

S.J. van Elsloo



TU Delft

Proposal and Validation of an Immersed Interface Method applied to the Lattice Boltzmann Method

by

S.J. van Elsloo

to obtain the degree of Master of Science

at the Delft University of Technology,

Student number:	4487060	
Project duration:	June 22, 2021 - June 27, 2022	
Thesis committee:	Dr. ir. M.I. Gerritsma	Chair
	Dr. ir. F. Avallone	Examiner
	Dr. ing. S.G.P. Castro	Daily Supervisor
	Dr. Ir. A.H. Van Zuijlen	Daily Supervisor

TU Delft

Abstract

The aerodynamic analysis of flapping wing micro-aerial vehicles has gathered notable attraction throughout the last two decades [1]. The lattice Boltzmann method (LBM) has a number of characteristics that suit the simulation of low Reynolds number, high Strouhal number flow particularly well, whilst being known to be computationally efficient [2, 3]. The immersed boundary method (IBM) [4] is a popular method in the LBM [5], due to the fact that it can be used for non-boundary-fitted grids. A derivative of the IBM is the immersed interface method (IIM), originally proposed by Leveque and Li [6], in which jumps in the solution are directly incorporated in the discretisation.

The IIM has yet to be successfully applied to the LBM. Qin et al. [7] recently made an attempt to do so, but as is shown in this thesis, their derivation appears flawed. It is therefore of interest to investigate how a correct application of the immersed interface method to the LBM looks like, and how it compares against a conventional IBM.

In the derivation and validation of the IIM in the LBM, a number of important results has been achieved.

First of all, an IIM applied to the framework of the LBM is proposed. The immersed interface method was initially proposed by Leveque and Li [6], and an attempt was made by Qin et al. [7] to apply the IIM to the LBM, but it was shown in Chapter 5 that their derivation was fundamentally flawed. It has been shown that correcting the erroneous steps in their derivation does result in an equivalent expression as the one proposed in this thesis, however.

The basis concept of the IIM applied to the LBM is that instead of interpolating the macroscopic flow quantities are interpolated at the *pre*-collision phase, such that the applied boundary forcing must be transmitted to the populations through the collision phase, the *post*-populations are directly interpolated at an arbitrary time within the current timestep. The boundary forcing is then not applied during the collision phase, but via jump conditions during the streaming phase.

It is shown that the default IIM is nearly identical to the IBM, if the same forcing discretisation is used. However, contrary to the IBM, the IIM allows for more precise control over at which point within a timestep the algorithm is evaluated. It is therefore proposed to apply a midpoint immersed interface method (MIIM), where the populations are evaluated at their mid-streaming positions.

Secondly, to validate the MIIM, and to explore its benefits with regards to the IBM, a custom-built LBM-based solver was written, called LaBIB-FSI[1]. This solver is written in `C++` and is capable of handling fluid-structure interaction (FSI) by coupling with the `pyfe3d`-solver. Amongst others, the LaBIB-FSI solver is written to be able to handle different collision operators, a multigrid for the fluid solver, and iterative FSI operations.

The LaBIB-FSI solver has been validated on the basis of numerous validation cases in Chapter 7 and 8, consistently showing the ability to achieve results that converge to analytical values when available, or reasonably close to reference values when no analytical solution exists. The tested validation cases ranged from a simple lid-driven cavity flow to the well-known benchmark cases by Turek and Hron [8], confirming the correct functioning of the software.

Thirdly, from its implementation in the LaBIB-FSI solver, it has become evident that the MIIM has noteworthy benefits compared to the IBM, as has been shown throughout Chapter 7 and 8. In particular, results produced by the MIIM appear to suffer from less spurious oscillations in the solution field than the IBM.

For example, when used for a cylinder embedded in a Taylor-Green vortex flow, the boundary force density (which should ideally be 0) was orders of magnitude lower for the MIIM than for the IBM. When applied to the CFD1 and CFD2 cases from Turek and Hron [8], significantly weaker oscillations are encountered when evaluating the boundary force density around the cylinder and flag. When evaluating the FSI3 benchmark, it is

[1]Lattice Boltzmann Immersed Boundary Fluid Structure Interaction.

evident that less spurious oscillations are present in the time history of the drag and lift coefficient, appearing to dampen out those more quickly, too.

Furthermore, the MIIM appears to consistently approximate the reference values of the drag and lift values from the Turek and Hron benchmarks slightly better at the same level of grid refinement. Similarly, at the same level of grid refinement, the discretisation error appeared smaller in the benchmark case of an oscillating cylinder in a fluid at rest. This lends further credit to the MIIM being an improvement over the IBM.

In short, three major results are presented in this work:

- A proposal for the application of the immersed interface method to the lattice Boltzmann method is made.
- A custom-written, `C++`-based solver, capable of solving fluid-structure interaction problems by coupling with the `pyfe3d`-library, has been developed and thoroughly validated.
- The midpoint immersed interface method is shown to bear notable advantages compared to the IBM, in particular with regards to reducing the presence of spurious oscillations in the solution field.

List of Symbols

Upper-case Roman

C	Damping matrix
D/Dt	Material derivative ($\partial/\partial t + \partial/\partial \boldsymbol{x}$)
$D(\boldsymbol{x})$	Multi-dimensional Discrete delta function
E	Total energy density
E	E-modulus
E_{xx}	Material stiffness in x-direction
$\boldsymbol{F}$	Eulerian force at Lagrangian point
$\boldsymbol{F}$	Body force
$\boldsymbol{F}_{\text{in}}$	Internal fluid force
$\boldsymbol{F}_{\text{tot}}$	Total force
$\boldsymbol{F}(\boldsymbol{s}, t)$	Boundary force distribution
$\boldsymbol{H}_n$	Hermite polynomial tensor
$\boldsymbol{I}$	Identity tensor
K	Stiffness matrix
M	Mass matrix
$Q(\boldsymbol{u},\theta,\boldsymbol{\xi})$	Composite polynomial containing terms related to $f^{\text{eq}}(\boldsymbol{x},\boldsymbol{\xi}, t)$ (Equation (3.19))
$S_i(\boldsymbol{x}, t)$	Source term corresponding to population f_i
T	Temperature
$\boldsymbol{U}$	Eulerian velocity vector of a Lagrangian point
W	Work
$\boldsymbol{X}$	Eulerian position vector of a Lagrangian point
$\boldsymbol{X}(\boldsymbol{s}, t)$	Parametric description of a surface

Lower-case Roman

$\mathbf{a}$	Acceleration
$\boldsymbol{a}_n$	Hermite coefficient tensor
$\boldsymbol{c}_i$	Lattice velocity
c_s	Lattice speed of sound
$\mathbf{d}$	Displacement
e	Internal energy density

$\boldsymbol{f}$	External force density
$f(\boldsymbol{x},\boldsymbol{\xi},t)$	Particle distribution function
$f^{\text{eq}}(\boldsymbol{x},\boldsymbol{\xi},t)$	Equilibrium particle distribution function
$f^{\text{neq}}(\boldsymbol{x},\boldsymbol{\xi},t)$	Non-equilibrium part of the particle distribution ($f^{\text{neq}} = f - f^{\text{eq}}$)
$f_i(\boldsymbol{x},t)$	Population
$f_i^{\text{eq}}(\boldsymbol{x},t)$	Equilibrium population function
$f_i^{\text{neq}}(\boldsymbol{x},t)$	Non-equilibrium part of the population ($f_i^{\text{neq}} = f_i - f_i^{\text{eq}}$)
$\hat{f}_i(\boldsymbol{x},t)$	Post-collision population
p	Pressure
r	Euclidean distance between two points
$\boldsymbol{s}$	Parametric coordinates of boundary surface
t	Time
$\boldsymbol{u}$	Velocity vector
$\boldsymbol{v}$	Relative velocity vector
w	Compact support width
w_i	Lattice weight
$\boldsymbol{x}$	Position vector

Upper-case Greek

Γ	Boundary
Δs	Lagrangian spacing
Δt	Time-step
Δx	Eulerian spacing
$\Pi^{(n)}$	Tensor containing the nth-order macroscopic velocity moments
Ω	Domain
$\Omega(f)$	Particle distribution collision operator
$\Omega(f_i)$	Population collision operator (abbreviated as Ω_i)

Lower-case Greek

α_d	Mass damping factor
β	Newmark-method parameter
β_d	Stiffness damping factor
γ	Newmark-method parameter
$\delta(\boldsymbol{x})$	Dirac-delta located at $\boldsymbol{x}$
θ	Non-dimensional velocity
$\boldsymbol{\xi}$	Velocity space vector
μ	Dynamic viscosity

ν	Kinematic viscosity
ν	Poisson's ratio
ρ	Density
$\phi(r)$	One-dimensional discrete Delta function
τ	Relaxation rate
ω	Vorticity
$\omega(\boldsymbol{x})$	Hermite moment generating function

Superscripts

$\mathbf{A}^T$	Transpose of $\boldsymbol{A}$

Subscripts

a_f	Quantity a corresponding to a fluid
a_s	Quantity a corresponding to a structure
a_l	Quantity a evaluated in lattice units
a_p	Quantity a evaluated in physical units
a_f	Quantity a corresponding to a fine grid
a_c	Quantity a corresponding to a coarse grid

Superscript indices

$(n), (p), (r), ...$	Iteration index
(k)	Time-step index

Subscript indices

$i, j, k, ...$	Component in velocity space (f_n denotes the nth population)
$\kappa, \lambda, \mu, ...$	Lagrangian node ($\boldsymbol{F}_\kappa$ denotes the position of κ Lagrangian node)
$n, p, r, ...$	Grid node ($\boldsymbol{x}_i$ denotes the position of ith grid node)
$n, p, r, ...$	Index of component in sequence ($\boldsymbol{H}_n$ denotes the tensor containing Hermite polynomials of order n)
$\alpha, \beta, \gamma, ...$	Component in physical space ($\boldsymbol{x}_\alpha$ denotes the α-component of the vector $\boldsymbol{x}$)

Symbols

∇	Nabla operator
$[\![f]\!]_x$	Jump in quantity f at position x

Abbreviations

CFD	Computational Fluid Dynamics
CSM	Computational Structure Mechanics
FSI	Fluid-Structure Interaction
IBM	Immersed Boundary Method
IIM	Immersed Interface Method
LaBIB-FSI	Lattice Boltzmann Immersed Boundary - Fluid-Structure Interaction
LBM	Lattice Boltzmann Method
LES	Large Eddy Simulation
MAV	Micro-aerial vehicle
MIIM	Midpoint Immersed Interface Method

Acknowledgements

This thesis closes the chapter on my MSc. in Aerodynamics, the start of which feels like a life-time ago by now. By extension, my seven-year journey here in Delft comes to an end too, or at least the 'being a student' part of it. The past two years have obviously not been what I signed up for originally, and wondering how it could have been is still a saddening thought to this day. Nonetheless, it is easy to glance over the positives that would not have happened if it wasn't for the pandemic, and I'm glad to be graduating in a time where the worst of times seems behind us.

One thing that has intrigued me during this thesis is that for most of my MSc., I thought the MSc. thesis would be the least enjoyable part of this MSc, and whenever people asked me what I'd like to do after I'd graduate, the only thing I knew for sure was that I wouldn't want to do a PhD. Yet after approximately a year of working on this thesis, I find myself having had an immensely enjoyable time, so much that I even ended up accepting a PhD-position that I look forward to starting very soon. Writing my own CFD-code that is capable of handling fluid-structure interaction from scratch has been remarkably enjoying, and allowed me to develop my skill set in ways that were not possible during regular coursework. In general I look back very fondly on working on this thesis (apart from the entire work-from-home every single day of the week ordeal, I surely won't miss that).

There is a number of people to whom I owe my sincere gratitude.

First of all, to my parents, for taking me back in during the pandemic and unconditionally supporting me throughout everything.

Of course to my dear friend Sara, who always provided a listening ear for my 5 a.m. complaints/rants/existential crises, motivated me when I was feeling down, and has always been an inspiration in perseverance and hard work, motivating me to do the same.

Finally, to Sander and Saullo, for being exemplary supervisors during this thesis. Even though, we have unfortunately not yet had the pleasure of meeting in person due to the pandemic, I am incredibly grateful for your supervision throughout this thesis. You calmed me down when I was panicking about making slow progress, provided valuable technical help in implementing the structural solver, and gave important advice in helping me understand the results that I obtained throughout this thesis. If I end up supervising MSc. students during my PhD, I now have two excellent examples of what kind of supervisor I would strive to be.

Sam van Elsloo

Baarn, June 2022

Contents

1 Introduction

The aerodynamic analysis of flapping wing micro-aerial vehicles has gathered notable attraction throughout the last two decades [1]. The flow around such vehicles exhibits several complex flow phenomena, such as dynamic stall, wing interaction via the clap-and-fling effect, and high-frequency flow phenomena due to the high Strouhal number [9]. Up until recently, numerical aerodynamic analysis has been mostly limited to the traditional, Navier-Stokes based methods [1], but in recent years, the use of the lattice Boltzmann method has attracted attention [10–12]. The lattice Boltzmann method has a number of characteristics that suit the simulation of low Reynolds number, high Strouhal number flow particularly well, whilst being known to be computationally efficient [2, 3].

The immersed boundary method [4] is a boundary treatment that can be used for non-conforming Cartesian grids with moving boundaries, in which the boundary is treated by placing a distributed force on the boundary that ensures the local fluid velocity matches the velocity of the boundary. As a result, it is a popular method in the lattice Boltzmann method [5], for which it is impractical to have a boundary-fitted grid. A derivative of the immersed boundary method is the immersed interface method, originally proposed by Leveque and Li [6], in which jumps in the solution are directly incorporated in the discretisation.

The immersed interface method has yet to be successfully applied to the lattice Boltzmann method. Qin et al. [7] recently made an attempt to do so, but as will be shown in this thesis, their derivation appears flawed. It is therefore of interest to investigate how a correct application of the immersed interface method to the lattice Boltzmann method looks like, and how it compares against a conventional immersed boundary method scheme.

The objective of this thesis is therefore two-fold. The primary objective is to derive, implement and validate an immersed interface method in conjunction in the lattice Boltzmann method, comparing its performance and characteristics against an immersed boundary method applied to the same solver. Furthermore, a critical look at the derivation by Qin et al. [7] is taken, in order to understand the flaws in their method.

In order to be able to validate the immersed interface method, the secondary objective is to develop and validate a lattice Boltzmann fluid solver that is coupled to the structural solver `pyfe3d` [13]. Both the solver and the implementation of the immersed interface method are then validated on a number of benchmarks of varying complexity, with the most complex validation case being the well-known FSI3 benchmark by Turek and Hron [8], in which vortex shedding behind a cylinder results in a complex fluid-structure interaction problem.

Consequently, this report is structured as follows. Chapter 2 provides a brief background on the motivation for the research performed in this thesis, ending with a list of research questions. Chapter 3 describes the theoretical basis of the lattice Boltzmann method, and Chapter 4 briefly covers the implementation of the conventional immersed boundary method in the lattice Boltzmann method. Chapter 5 shows the derivation of the proposed immersed interface method, goes into detail on the differences between the immersed interface method and the immersed boundary method, and examines the differences with the proposal by Qin et al. [7]. Chapter 6 details the remaining major parts of the LaBIB-FSI solver that has been written as part of this thesis. Chapter 7 describes the validation cases used to validate the LaBIB-FSI solver for problems involving stationary boundaries, and Chapter 8 does so for problems involving moving boundaries. Chapter 9 details the sensitivity of the solution to some numerical input parameters. Finally, Chapter 10 and 11 answer the research questions posed in Chapter 2 and provide a list of recommendations for future research, respectively.

I

Theoretical Background

In Chapter 2-4, background is given to the work performed for this thesis. In particular, Chapter 2 gives a brief overview on the available literature on (numerical) aerodynamic research involving flapping wing micro-aerial vehicles. Furthermore, the benefits of the lattice Boltzmann method as fluid solver and the immersed boundary method as boundary treatment are discussed. Finally, a number of research questions is outlined. In Chapter 3, a brief overview of the lattice Boltzmann method, covering its mathematical derivation, its implementation and an overview of multi-relaxation time collision operators. In Chapter 4, the immersed Boundary method is discussed.

2 Motivation

In this chapter a brief overview of aerodynamic research regarding the flapping wing micro-aerial vehicles will be provided. Section 2.1 will briefly cover some and outline the main challenges faced in the aerodynamic analysis of flapping wing micro-aerial vehicles. Section 2.2 will discuss some examples of the lattice Boltzmann method applied to the analysis of flapping wing unmanned-aerial vehicles, and Section 2.3 will introduce the immersed boundary method, a popular boundary treatment in the lattice Boltzmann method. Section 2.4 will discuss a small sample of modifications to the immersed boundary method, including the immersed interface method. Finally, Section 2.5 will pose the research questions that will be answered throughout this thesis.

2.1. Introduction to aerodynamic modelling of flapping wing unmanned-aerial-vehicles

Recently, Bin Abas et al. [1] performed an extensive review on the research performed on flapping wing unmanned-aerial-vehicles (UAVs), compiling the experimental results discussed in more than a dozen articles, and almost the numerical results found in almost a score of additional articles, providing a comprehensive overview of the challenges facing the development of flapping wing UAVs.

Bin Abas et al. identified two main types of wing kinematics: those based on ornithopter flight, and those based on insect flight. Ornithopter flight is characterised by two degrees of freedom, namely the main flapping motion, and the adjustment of the pitch of the wing. Insect flight is characterised by an additional degree of freedom, namely to rotate the wing about the vertical axis. Lighthall [14] described how this latter degree of freedom can be used to generate a clap-and-fling effect, where the wings initially 'clapped' together at the leading edge until the whole wings touch each other, and subsequently 'flung' from each other at the leading edge to start the regular flapping motion, as shown in Figure 2.1 and 2.2 and analysed in detail by Miller and Peskin [15]. Other characteristic kinematics are those of a dragonfly, having two sets of wings that can flap independently from each other, or a bee, where the wings follow an inclined figure-of-eight. Furthermore, Bin Abas et al. [1] noted that for large insects and small birds, the chord-wise Reynolds number is typically in the range of 1000 - 15,000, a region where flow can be expected to be dominated by transition from laminar to turbulent flow.

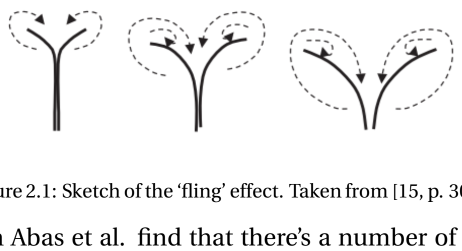


Figure 2.1: Sketch of the 'fling' effect. Taken from [15, p. 3077].

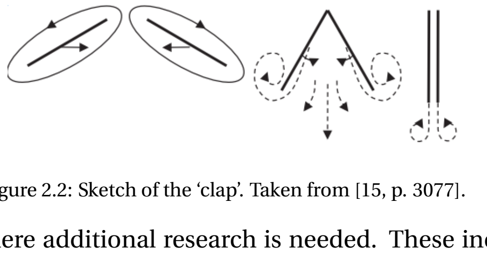


Figure 2.2: Sketch of the 'clap'. Taken from [15, p. 3077].

Bin Abas et al. find that there's a number of areas where additional research is needed. These include that most numerical research neglects the mass of the wings which may affect stability significantly; limited research regarding the effect of particle-dominated weather such as rain and snow; research where fluid-structure interaction (FSI) is accounted for; and computational efficiency is still an issue, with Liu and Aono [16] showing that it took them approximately ten hour of CPU time to simulate four seconds of flight. However, it should be noted that Liu and Aono performed their research in 2009, and thus, on a modern-day computer architecture, the required CPU time can be expected to be significantly lower already.

Bin Abas et al. [1] showed that the vast majority of numerical research of flapping UAVs so far has been performed using traditional CFD methods, based on directly solving the Navier-Stokes equations using e.g. a finite volume or finite element scheme (all of the numerical research results included by Bin Abas et al. in their literature review were Navier-Stokes based methods). Similarly, the Delfly (shown in Figure 2.3) is a flapping wing UAV for which the TU Delft has performed a significant amount of research in a variety of domains, as exhaustively compiled by De Croon et al. [17]. Numerical aerodynamic research on the Delfly has so far, however, been completely focussed on finite volume formulations of the Navier-Stokes equations. For example, Gillebaart [18] used an arbitrary Lagrangian-Eulerian approach to include a moving mesh around the DelFly to examine the influence of flexibility on flapping wings performing the clap-and-fling motion; Tay et al. [19–21] used an immersed boundary method applied to a Navier-Stokes based solver, to model the flow around the DelFly II.

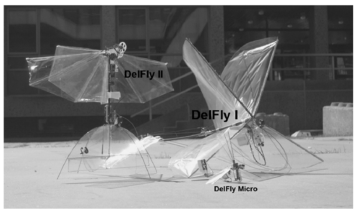


Figure 2.3: The DelFly I, DelFly II and DelFly Micro. The DelFly II is the most established model of the family, according to Tay et al. [20]. Taken from [19, p. 3].

Motivated by the need for a computationally efficient method and a wish for the implementation of FSI in the simulation of flapping wing UAVs, it is interesting to explore the possibility of using alternative, non-Navier-Stokes-based solvers that may prove advantageous over the conventional Navier-Stokes-based ones.

2.2. Use of the lattice Boltzmann method in simulating flapping wing UAVs

Motivated by the need for a computationally efficient method and a wish for the implementation of FSI in the simulation of flapping wing UAVs, it is interesting to explore the possibility of using alternative, non-Navier-Stokes-based solvers that may prove advantageous over the conventional Navier-Stokes-based ones. One such class of solvers are those based on the lattice Boltzmann method (LBM), a relatively novel approach to modelling aerodynamics, gaining popularity in the 1990s as natural extension of lattice gas automata developed in the 1970s [2, 3, 22].

The lattice Boltzmann method models the mesoscopic particle distribution function over a fixed, Eulerian lattice. The particle distribution function is discretised in velocity space into a set of populations, and subsequently discretised in space and time. At each time step, the populations undergo a collision process, letting them approach some equilibrium value dependent on the local macroscopic flow quantities. Then, the populations are streamed to adjacent nodes (depending on which discretised velocity they belong to). The next time step then collides and streams the populations again. The lattice Boltzmann method is explained in more detail in Chapter 3.

From a theoretical point of view, the use of LBM seems particularly alluring for the modelling of flapping wing UAVs:

- Lattice Boltzmann methods are well studied for laminar flows, but, due to instabilities in the ‘basic’ version of the LBM, applications to high Reynolds number flows are less common. Although the Reynolds number for flapping wing UAV is in the order of 10^4 and thus transition and turbulence can be expected[1], its Reynolds number range has already been studied in the context of the LBM, and modifications to the LBM that improve the accuracy under turbulence have already been proposed and

[1]For example, for the DelFly II, Gillebaart [18] simulated the flapping motion for a Reynolds number of 9641, using a chord length of 7.4 cm and a characteristic velocity of 1.76 m/s.

validated. For example, Hou et al. [23] proposed two approaches to apply an LES-framework to the LBM; by either directly modifying the relaxation rate based on an eddy viscosity model or by filtering the equilibrium populations. Krafzcyk et al. [24] demonstrated the applicability of the Smagorinsky model for a multiple-relaxation-time (MRT) model (described in Section 3.4). Uphoff [25] implemented the original Smagorinsky model [26], the Smagorinsky model with Van Driest damping [27], the wall-adopting local eddy-viscosity (WALE) model [28] and the Vreman model [29], and compared their accuracy for various validation cases to a cascaded LBM (described in Section 3.4.3) without an eddy-viscosity model present. It was found that only the Smagorinsky model was notably less accurate for a turbulent channel flow than the other models. These results mean that higher Reynolds number flow should also be feasible to simulate moderate to high Reynolds number flow without requiring an explosive increase in computational effort.
- Lattice Boltzmann methods normally suffer from the fact that they are inherently unsteady, and thus require significantly more computational effort than required for steady-state simulations. However, flow around a flapping wing UAV will naturally be unsteady, thus this is actually a welcome characteristic of the LBM.
- Lattice Boltzmann methods generally struggle with compressible and high Mach number flow. For flapping wing UAVs, the Mach number is very small[2], so this should not be an issue.
- Lattice Boltzmann methods scale extraordinarily well with a parallelised computational architecture, due to the fact that all calculations can be performed locally, contrary to Navier-Stokes-based methods, where a system of equations for the entire domain needs to be solved simultaneously. This can help considerably in decreasing the required CPU time for simulations.

Evidently, some of the conventional disadvantages of the LBM do not directly apply to the simulation of flapping wing UAVs, making an exploration of their applicability to flapping wing UAVs worthwhile. Indeed, several authors have applied the LBM to model the flow around flapping wing UAVs.

Several authors already have explored the applicability of LBM to flapping UAVs. Pradeep Kumar et al. [30] investigated the clap and fling motion using an LBM with an immersed boundary method, and found that the results of the LBM compared well with those obtained from experiments and other numerical simulations at a Reynolds number of 75-150. It should be noted that their model treated the structure as rigid, even though the flexibility of the wings will likely have a great affect on the clap-and-fling effect.

De Rosis et al. [31] performed an FSI analysis of a flexible flapping wing, using an LBM with an immersed boundary method for the fluid, and a finite element model for the wing. They modelled the wing in a 2D domain as a set of flat plates, hovering in the air by flapping its wings, at a Reynolds number of 200. Their model was previously verified and validated in a previous work [32], demonstrating its accuracy for a longitudinally oscillating cylinder and a deformable cantilever beam in a viscous channel flow.

Hino and Inamuro [11] demonstrated the use of LBM with an immersed boundary method, simulating a dragonfly wing-body model over time, using two rotational degrees of control per wing. They showed that the LBM was able to simulate a path that the dragonfly would cover in three-dimensional space over time, given a set of control inputs, around a Reynolds number of approximately 200. Unfortunately, they did not compare their results with those obtained by experiments or other numerical studies, so it is hard to judge the accuracy of their model.

2.3. Review of the Immersed Boundary Method

The immersed boundary method (IBM) was originally proposed by Peskin [4] in 1977 and Feng and Michaelides [5] introduced the immersed boundary method to the LBM framework. The fundamental concept behind it is that solid boundaries are not directly imposed on the solution through boundary conditions, but a forcing field is added to the domain that ensures that the flow matches the velocity of the solid boundaries. This force field is applied at the solid boundary by introducing a number of marker points on the boundary and forces are then diffused towards nearby grid elements. The problem is then two-fold: how can such a force field be determined, and how can this force field be diffused towards nearby grid elements.

In a more mathematical sense, this can be formulated as follows: for the incompressible Navier-Stokes equa-

[2]For example, the Mach number of the DelFly II, Gillebaart [18], based on its characteristic velocity, was equal to 0.005.

tions, the governing equations become

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2.1}$$

$$\rho \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) = \mu \nabla^2 \boldsymbol{u} - \nabla p + \boldsymbol{f} \tag{2.2}$$

$$\boldsymbol{f}(\boldsymbol{x}, t) = \int_{\Gamma} \boldsymbol{F}(\boldsymbol{s}, t) \, \delta(\boldsymbol{x} - \boldsymbol{X}(\boldsymbol{s}, t)) \, d\boldsymbol{s} \tag{2.3}$$

where $\boldsymbol{F}$ is the immersed boundary force, and $\boldsymbol{X}(\boldsymbol{s}, t)$ is the parametric surface at which $\boldsymbol{F}$ is applied. The question is clearly how to determine a suitable model for $\boldsymbol{F}$, and how to discretise the integral (and its delta function).

Two major variations of the IBM exist in the diffuse-interface IBM and the sharp-interface IBM. For the diffuse-interface IBM, consider Figure 2.4, where a cylinder in a typical LBM mesh is depicted. The white nodes correspond to the Eulerian grid on which the flow is solved (and remains stationary); the black nodes correspond to the Lagrangian grid at which the forcing is applied (and moves along with the cylinder). Let $\boldsymbol{x}_i$ denote the nodes on the Eulerian mesh, and $\boldsymbol{X}_k$ the nodes on the Lagrangian mesh. A more detailed analysis of the LBM is included in Chapter 4, but the general approach consists of interpolating the velocity at the location of a boundary marker from nearby fluid nodes, applying a constitutive model to calculate a feedback force at the Lagrangian marker, and then diffusing this feedback force back to the fluid nodes. A variety of methodologies then exist based on differences between how the velocity is interpolated, what constitutive model is used and how this force is diffused back into the fluid domain.

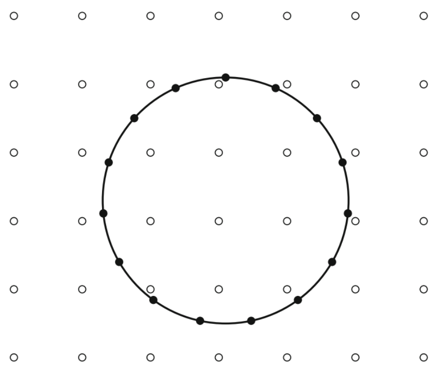


Figure 2.4: Sketch of a cylinder discretised by an Lagrangian mesh, embedded in an Eulerian mesh. Taken from Krüger et al. [2, p. 466].

On the other hand, for a sharp-interface IBM, typically an image point is constructed as a reflection of a ghost node (a node in the solid domain that is directly adjacent to a fluid node) around the boundary marker into the fluid domain. The flow properties at this image point are interpolated from nearby fluid nodes, and with the boundary condition known at the boundary marker, the flow properties at the ghost node may be constructed. This generally allows for the numerical solution to proceed (e.g. in a finite-difference scheme, boundary conditions can now be directly imposed on the ghost nodes, allowing for the system of equations at that time-step to be solved).

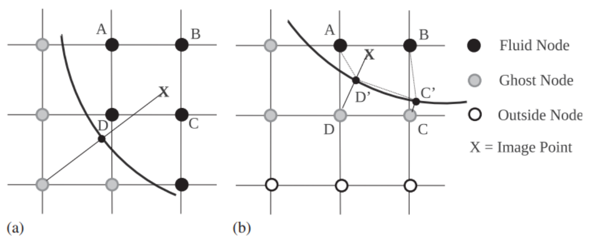


Figure 2.5: Interpolation stencils for two types of boundary intersections. In (a), the image point is reflected into a square cell. In (b), the image point is reflected into a cell that is cut by the boundary. Taken from [33, p. 487].

It is noted that a vast array of variations exist however, and the reader is referred to an exhaustive review of the IBM by Huang and Tian [34] for a more in-depth review of the differences between them. The diffuse-interface is generally less challenging to apply, as it for example neither requires identification of which fluid nodes are inside/outside the solid boundary, nor does it require an projection of an image point into the fluid domain.

As discussed in Section 2.2, the IBM has been used by several authors in combination with the LBM, with all of the listed authors using the diffuse-interface LBM. Due to the apparent popularity of these methods (confirmed by e.g. Krüger et al. [2]) and the increased complexity of the sharp-interface IBM over the diffuse-interface IBM, it was chosen to limit the scope of this thesis to the diffuse-interface IBM. As such, in all other chapters of this thesis, the term 'immersed boundary method' shall strictly refer to the diffuse-interface IBM.

Huang and Tian [34] recently conducted an extensive literature review on the immersed boundary method, across a variety of fluid solver frameworks. In addition to some of the drawbacks already mentioned for some of the algorithms outlined above, they raise the following points:

a) It has been shown that the number of iterations in the multi-direct forcing algorithm to reduce the iteration error to a satisfactorily small value (i.e., smaller than the discretisation error from the IBM itself) is relatively small; in the order of 5-10 iterations should generally be sufficient (see e.g. Kang et al. [35]). However, for flow over a thin filament or membrane that is not aligned with the flow (and more generally, flow where the velocity near the immersed boundary has a notable effect on the interface dynamics), the iterations do significantly improve the performance, a result also observed by Kang et al. [35] for the LBM.
b) For closed volumes, the volume leakage is a problem, particularly for problems with large deformations and long-time behaviour. For the multi-direct forcing (and implicit) immersed boundary method, this should be less of a problem than for the penalty IBM, as the latter does not explicitly ensure that $\boldsymbol{U}_k$ equals the specified boundary velocity.
c) The IBM has been found to be first-order accurate, due to its use of a smoothed delta function.
d) The IBM is typically characterised by spurious oscillations when the body moves, due to the discontinuities introduced by a solid object crossing Eulerian nodes. These are particularly troublesome for sharp-interface methods, as diffused-interface methods naturally stabilise the solution by smoothing out the boundary in the first place.
e) Using IBM for turbulent flow is challenging, due to its use of a Cartesian grid. The grid should be sufficiently refined to resolve the viscous sublayer (requiring a grid spacing of around y^+). However, as the grid spacing is isotropic and rectilinear, this leads to a very large number of required grid points. This problem is exacerbated when the solid body moves throughout time, as either a large region of the domain needs to be very refined, or an adaptive mesh should be implemented, that automatically refines the mesh in the vicinity of the body. One solution to this would be the use of a wall model, which models the inner layer of the boundary layer and simulates the outer layer.
f) In general, the Lagrangian mesh spacing should be at least twice as fine as the Eulerian grid spacing.

2.4. Possible improvements to the immersed boundary method

As reviewed by Huang and Tian [34], there exists a wide variety of variations on the IBM that are relevant. For this thesis, two modifications to the 'default' IBM, as proposed by Peskin [4], will be treated in more detail.

First, a more sweeping variation on the IBM is the immersed interface method (IIM), originally proposed by LeVeque and Li [6] for elliptic differential equations, later applied to the Navier-Stokes equations by Lee et al. [36] and extended to moving boundaries by [37]. Immersed interface methods are characterised by including singular attributes, such as a singular force field arising from an immersed boundary, as jumps in the solution space. An example of an application of the IIM to a simple, one-dimensional elliptic differential equation is provided in Appendix A.

As shown by Vaughan et al. [38], the IIM (applied to a FEM-based solver) appears to obtain a smaller error at the same level of grid refinement when evaluated for a steady, elliptic heat equation in square domain with a sourcing function that is only nonzero on the circular interface Γ_1 shown in Figure 2.6. Here, the error was defined as the maximum deviation from the analytical solution for any node in the fluid domain.

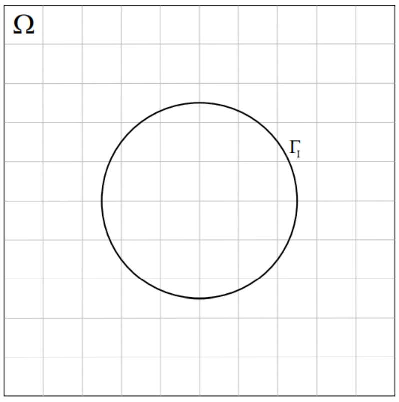


Figure 2.6: Square domain with boundary Γ_1. Taken from [38, p. 218].

Similar results for the same validation case were obtained previously by LeVeque and Li [6] previously. However, it should be noted that comprehensive reviews for how the IIM compares to the IBM for unsteady and FSI-related problems appears to be lacking in current literature, which makes it difficult to evaluate how pronounced the differences would be. From a theoretical perspective, as the boundary is not diffused in the IIM, it appears promising that the IIM may be more accurate in representing e.g. a velocity or density jump across a surface and it therefore appears worth exploring in more detail.

The IIM has only been applied to the LBM in a very limited fashion. Recently, Qin et al. [7] proposed an implementation of the IIM into the LBM, but it is shown in Chapter 5 that their derivation is flawed in several steps. It is therefore deemed an open question on how to correctly derive jump conditions on the populations in the LBM, and how to implement these successfully.

Secondly, Yang et al. [39] investigated the cause of the oscillations by investigating the numerical truncation error of the general diffusion process. By imposing additional conditions on the derivative of the interpolation functions, they were able to obtain smoothed discrete delta functions which have an increased order of the leading term of the truncation error.

Yang et al. found that this notably suppressed the oscillations in the solution. Figure 2.7 shows the variation in the drag coefficient over time for a cylinder oscillating horizontally in a horizontal channel, at a Reynolds number based on the diameter of $\mathrm{Re}_d = 185$. It is clear that the smoothed delta function damps the oscillations significantly whilst not impacting the accuracy; the average drag coefficient was found to differ by only 0.23%.

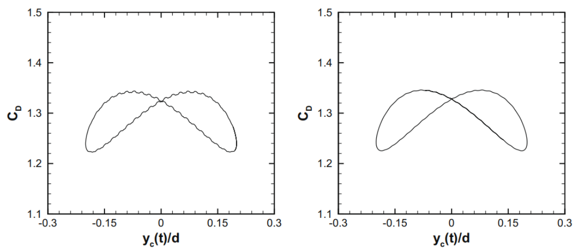


Figure 2.7: Variation over time of the drag coefficient of an oscillating cylinder, at Re = 185, using a 'bare' interpolating function (left) and its smoothed variant (right). Taken from [39, p. 7833].

However, although their interpolation functions are simple to construct and appear to smoothen the oscillations notably, they do result in a boundary that is diffused over an even wider part of the fluid domain (due to an increased width of the interpolation zone), fuzzing the actual solid boundary. This both decreases the accuracy and increases the computational cost compared to using the non-smoothed interpolating function.

In addition to these methodological modifications to the IBM-scheme, within the framework of the LBM specifically, two straightforward solutions to reduce the presence of oscillations in the solution field caused by the IBM also exist. As discussed in Chapter 9, a solution is to increase the Mach number at which the simulation is run (assuming it stays low enough for the flow to be quasi-incompressible) as this naturally dampens oscillations in the flow [2]. A similarly simple remedy is to reduce the number of Lagrangian markers across the boundary. These solutions are straightforward to implement (as it merely requires the modification of an input parameter), but may negatively impact the accuracy of the solution. The effectiveness of both these solutions is investigated in more detail in Chapter 9 of this report.

2.5. Research questions

In this Chapter, it has become clear that the LBM appears to be a suitable and appealing fluid solver to solve FSI-problems related to the flow around flapping wing MAVs. Existing LBM applications to these problems typically use the IBM, which has been the subject of extensive research already. Nonetheless, advancements are still possible, for example regarding the reduction of numerical noise in the solution field when using the IBM, and the development of an IIM, a method similar to the IBM, in the LBM.

Consequently, a number of research questions may be posed, which shall be answered in Chapter 10 of this report, the conclusions of which shall based on the supporting work to be presented in Chapter 3-9. In particular, the following questions are presented:

1. *How suitable is the lattice Boltzmann method for the fluid-structure-interaction of flapping wing MAVs, such as the DelFly II, in light of the the challenges posed by the highly deformable wings, low Reynolds number and high Strouhal number characteristics of such vehicles?*

To be able to answer the research questions following this, it was necessary to have an lattice Boltzmann-based fluid-structure interaction solver available that would allow for low-level implementation of the immersed interface method. Although existing commercial (e.g. XFlow or PowerFLOW) and open-source (e.g. Palabos) already exist, it was chosen to write a custom solver, due to the need for low-level access to the solver (which may not be trivial when using external solvers), the apparent simplicity of the LBM making writing such a software feasible within the scope of this thesis and the author's personal development goals for this thesis.

Naturally, it was therefore also necessary to validate this solver for several benchmarks that share characteristics with the motion of flapping wing MAVs, in order to assess whether the custom-built solver would be suitable for full-scale numerical experiments of such MAVs.

The implementation of the custom-built solver and its most important components is described in Chapter 5 and 6, and its validation is described in Chapter 7 and 8, with the most complex benchmark case that the solver has been validated for being the FSI3-benchmark case by Turek and Hron [8], a widely recognised benchmark case for the validation of fluid-structure interaction solvers.

2. *How can the immersed interface method be applied to the lattice Boltzmann method?*

As shown in Chapter 5, an attempt at applying the immersed interface method was previously made by Qin et al. [7]. However, their derivation appears to be several flawed in two aspects. It is therefore of interest how the immersed interface method can be correctly applied to the lattice Boltzmann method, and a proposal thereof is therefore provided in Chapter 5.

3. *How does the computational effort of the immersed interface method compare to that of the immersed boundary method, when applied to the Lattice Boltzmann method?*

A comparison between the immersed boundary and immersed interface method is interesting in several aspects, one of them being the difference in computational effort for both methods. This is addressed in Section 5.5.2.

4. *Does the implementation of the immersed interface method provide a tangible benefit over using the immersed boundary method in terms of numerical noise reduction and overall accuracy?*

Apart from the computational effort, the performance of both the immersed boundary and immersed interface method are of great interest. This is assessed in detail in Chapter 7 and 8, where the developed solver has been validated against several well-known benchmark cases.

5. *How effective is the immersed interface method in damping out numerical noise compared to adjusting the numerical parameters of the simulations, such as the Mach number of the flow and the width of the discrete delta functions?*

As shown in Chapter 7 and 8, the immersed interface method proposed in Chapter 5 appears to greatly reduce the numerical noise in the boundary forcing solution compared to the immersed boundary method when using identical input parameters. However, as previously mentioned, there are also easier ways of reducing this noise, such as increasing the Mach number or using smoothed discrete delta-functions as described by Yang et al. [39], although these would generally reduce the actual accuracy of the solution. It is therefore of interest to compare the benefits and disadvantages immersed interface method compared to those simpler solutions. This is addressed in Chapter 9, where several validation cases are reassessed for these different input parameters.

3 The lattice Boltzmann method

This chapter elaborates on the LBM introduced in Chapter 2. Section 3.1 describes the continuous form of the Boltzmann equation. Section 3.2 derives how the discretised Boltzmann equation can be obtained by discretising in velocity space and subsequently discretising in physical space and time. Section 3.3 discusses the main steps that need to be taken to show that the Navier-Stokes equations can be recovered from the discretised Boltzmann equation. Section 3.4 discusses some implementation details related to the initial conditions, boundary conditions, external forces and dimensionalisation. Section 3.5 explores various advanced collision models that are helpful in stabilising the LBM, particularly at higher Reynolds numbers.

3.1. Continuous Boltzmann equation

Let the particle distribution function be given by $f(\boldsymbol{x},\boldsymbol{\xi},t)$, where $\boldsymbol{x}$ denotes the position, $\boldsymbol{\xi}$ the velocity space and t the time. That is, f describes the density of a particle with a certain velocity $\boldsymbol{\xi}$ at a position $\boldsymbol{x}$, at a time t. Macroscopic variables such as density, velocity and energy can be obtained by taking moments of the particle distribution. That is,

- The mass density ρ is given by

$$\rho(\boldsymbol{x},t) = \int f\left(\boldsymbol{x},\boldsymbol{\xi},t\right) d\boldsymbol{\xi}. \tag{3.1}$$

- The momentum density $\rho\boldsymbol{u}$ is given by

$$\rho(\boldsymbol{x},t)\,\boldsymbol{u}(\boldsymbol{x},t) = \int \boldsymbol{\xi} f\left(\boldsymbol{x},\boldsymbol{\xi},t\right) d\boldsymbol{\xi}. \tag{3.2}$$

- The total energy density ρE is given by

$$\rho(\boldsymbol{x},t)\,E(\boldsymbol{x},t) = \int |\boldsymbol{\xi}|^2 f\left(\boldsymbol{x},\boldsymbol{\xi},t\right) d\boldsymbol{x}i. \tag{3.3}$$

- The internal energy density ρe is given by

$$\rho(\boldsymbol{x},t)\,e(\boldsymbol{x},t) = \int |\boldsymbol{v}|^2 f\left(\boldsymbol{x},\boldsymbol{\xi},t\right) d\boldsymbol{x}i, \tag{3.4}$$

  where $\boldsymbol{v} = \boldsymbol{\xi} - \boldsymbol{u}$ is the relative velocity between the particle velocity and the local mean velocity.

3.1.1. Equilibrium distribution

The distribution function naturally tends to an equilibrium over time. The equilibrium distribution of the particle distribution functions naturally follows a Maxwell distribution, i.e. the equilibrium distribution is proportional to [2]

$$f^{\text{eq}}(|\boldsymbol{v}|) = e^{3a} e^{b|\boldsymbol{v}^2|}. \tag{3.5}$$

By requiring that the equilibrium distribution satisfies the moments of density (Equation (3.1)) and energy (Equation (3.3)), it can be shown that the equilibrium distribution equals

$$f^{\text{eq}}(\boldsymbol{x}, |\boldsymbol{v}|, T, t) = \rho \left(\frac{1}{2\pi RT} \right)^{3/2} e^{-|\boldsymbol{v}|^2/(2RT)}. \tag{3.6}$$

Appropriate non-dimensionalisation of the parameters by some characteristic length, velocity, density and temperature [2] results in

$$f^{\text{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) = \frac{\rho}{(2\pi\theta)^{d/2}} e^{-(\boldsymbol{\xi}-\boldsymbol{u})^2/(2\theta)}, \tag{3.7}$$

where ρ, $\boldsymbol{u}$, θ and $\boldsymbol{\xi}$ are the non-dimensional density, fluid velocity, temperature and particle velocity, and d is the dimensionality of the problem. In section 3.4.4 it is discussed how to recover the dimensional parameters from the non-dimensional parameters.

3.1.2. Boltzmann equation

The distribution function tends to the equilibrium distribution due to the particle collisions that occur constantly. Correspondingly, the total derivative of the distribution function equals

$$\frac{Df}{Dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial t} + \frac{\partial f}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{\xi}}{\partial t}. \tag{3.8}$$

Note that $\partial \boldsymbol{x} / \partial t = \boldsymbol{\xi}$, and $\partial \boldsymbol{\xi} / \partial t = \boldsymbol{f} / \rho$, where $\boldsymbol{f}$ is a body force density. Letting the total derivative be equal to a collision operator $\Omega(f)$, the Boltzmann equation is obtained:

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial \boldsymbol{x}} \boldsymbol{\xi} + \frac{\partial f}{\partial \boldsymbol{\xi}} \frac{\boldsymbol{f}}{\rho} = \Omega(f). \tag{3.9}$$

The general form of the collision operator is generally complex and involves an integral over velocity space and the impact plane of two colliding particles, as well as the pre- and post-collision distributions of two colliding particles; the reader is referred to [40, 41] for a more detailed discussion of the general form of the Boltzmann operator.

3.1.3. Bhatnagar, Gross and Krook collision operator

Due to this complicated nature of the collision operator, simplified models for this operator are needed. In 1954, Bhatnagar, Gross and Krook [42] proposed a very simple collision operator, namely that the non-equilibrium part of distribution function exponentially decays (in a homogeneous distribution field), i.e.

$$\Omega(f) = -\frac{1}{\tau} f^{\text{neq}} = -\frac{1}{\tau} \left(f - f^{\text{eq}} \right), \tag{3.10}$$

where the parameter τ is known as the relaxation time. Note that this collision operator satisfies conservation of mass, momentum and energy, as expected since these quantities are conserved in perfectly elastic collisions. Due to its simplicity, it has been used widely ever since [43].

Examples of more sophisticated collision operators are discussed in Section 3.5.

3.2. Discretised Boltzmann equation

Discretising the continuous Boltzmann equation requires three discretisations. First, the velocity space should be discretised; secondly, the physical space should be discretised; and finally, the temporal space should be discretised.

3.2.1. Discretisation in velocity space

The discretisation in velocity space involves finding a set of velocities that allow for sufficient fulfilment of conservation laws. A detailed description of this is provided in e.g. [2, 44], but an overview of the main steps is provided here. The first step is to write the equilirium distribution function (Equation (3.7)) as a Hermite polynomial series expansion, i.e. as

$$f^{\text{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) = \frac{\rho}{(2\pi\theta)^{d/2}} e^{-(\boldsymbol{\xi}-\boldsymbol{u})^2/(2\theta)} = \omega\left(\boldsymbol{\xi}\right) \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{a}_n^{\text{eq}}\left(\rho, \boldsymbol{u}, \theta\right) \cdot \boldsymbol{H}_n\left(\boldsymbol{\xi}\right), \tag{3.11}$$

where $\omega(\boldsymbol{\xi})$ and $H_n(\boldsymbol{\xi})$ are the generating function and Hermite polynomial, respectively, given by

$$\omega(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}} e^{-\boldsymbol{x}^2/2} \tag{3.12}$$

$$\boldsymbol{H}_n(\boldsymbol{\xi}) = (-1)^n \frac{1}{\omega(\boldsymbol{x})} \nabla^n \omega(\boldsymbol{x}), \tag{3.13}$$

where ∇^n is the nth order gradient[1]. $\boldsymbol{a}_n$ is the tensor of rank n and length in each dimension d.

Using the orthogonality of the Hermite polynomials, i.e.

$$\boldsymbol{a}_n = \int f^{\text{eq}}(\rho,\boldsymbol{u},\theta,\boldsymbol{\xi}) \boldsymbol{H}_n(\boldsymbol{\xi})\, d\boldsymbol{\xi}, \tag{3.14}$$

and noting the similarity between the equilibrium distribution function itself and the generating function, by substituting $\boldsymbol{\eta} = (\boldsymbol{\xi} - \boldsymbol{u})/\sqrt{\theta}$, one can simply obtain

$$\boldsymbol{a}_n^{\text{eq}} = \rho \int \omega(\boldsymbol{\eta}) \boldsymbol{H}_n\left(\sqrt{\theta}\boldsymbol{\eta} + \boldsymbol{u}\right) d\boldsymbol{\eta}. \tag{3.15}$$

Evaluation of these expressions results in (for the first three coefficients)

$$a_0^{\text{eq}} = \rho \tag{3.16}$$

$$\boldsymbol{a}_1^{\text{eq}} = \rho\boldsymbol{u} \tag{3.17}$$

$$\boldsymbol{a}_2^{\text{eq}} = \rho\left(\boldsymbol{u}\boldsymbol{u}^\top + (\theta - 1)\boldsymbol{I}\right). \tag{3.18}$$

From comparison with Equation (3.1)-(3.3), it can be noted that these coefficients are directly related to the conserved moments of density, momentum and energy.

Combining (3.16)-(3.18) with Equation (3.11), including terms up until $n = 2$, we obtain

$$f^{\text{eq}}(\rho,\boldsymbol{u},\theta,\boldsymbol{\xi}) \approx \omega(\boldsymbol{\xi})\rho\left[1 + \xi\cdot\boldsymbol{u} + \left(\boldsymbol{u}\boldsymbol{u}^\top + (\theta-1)\boldsymbol{I}\right)\left(\boldsymbol{\xi}\boldsymbol{\xi}^\top - \boldsymbol{I}\right)\right] = \omega(\boldsymbol{\xi})\rho Q(\boldsymbol{u},\theta,\boldsymbol{\xi}), \tag{3.19}$$

where $Q(\boldsymbol{u},\theta,\boldsymbol{\xi})$ is evidently a polynomial of order 2.

Now that a truncated equilibrium distribution is obtained, a discretisation in velocity space can be performed; i.e., a set of weights w_i and discrete velocities $\boldsymbol{\xi}_i$ should be obtained such that the coefficients $\boldsymbol{a}_n$ (up to $n = 2$) can be computed exactly from the corresponding f_i^{eq} when using a Gauss-Hermite quadrature (as these coefficients related directly to the conserved moments), i.e.

$$\boldsymbol{a}_n^{\text{eq}} = \rho \int \omega(\boldsymbol{\xi}) Q(\boldsymbol{\xi}) \boldsymbol{H}_n(\boldsymbol{\xi})\, d\boldsymbol{\xi} = \rho \sum_{i=1}^{m^d} w_i Q(\boldsymbol{\xi}_i) \boldsymbol{H}_n(\boldsymbol{\xi}_i). \tag{3.20}$$

Note that the energy is related to the second-order Hermite polynomial H_2, and thus that the product $Q(\boldsymbol{\xi}_i)\boldsymbol{H}_2(\boldsymbol{\xi}_i)$ contains polynomial terms of order 4 (as $Q(\boldsymbol{\xi}_i)$ contains polynomial terms of order 2). Since a Gauss-Hermite quadrature of order m integrates polynomials of order $2m-1$ exactly [45], we thus require $m = 3$ nodes in each spatial dimension. The corresponding weights w_i and abscissae $\boldsymbol{\xi}_i$ are then known from e.g. [46]. The velocity is usually scaled by a factor c_s (known as the speed of sound). Subsequently, the discretised equilibrium distribution for isothermal flow ($\theta = 1$) becomes (using Equation (3.19)):

$$f_i^{\text{eq}} = w_i\rho\left(1 + \frac{\boldsymbol{c}_i\cdot\bar{\boldsymbol{u}}}{c_s^2} + \frac{\bar{\boldsymbol{u}}\bar{\boldsymbol{u}}^\top : \left(\boldsymbol{c}_i\boldsymbol{c}_i^\top - c_s^2\boldsymbol{I}\right)}{2c_s^4}\right), \tag{3.21}$$

where $\boldsymbol{c}_i = c_s\boldsymbol{\xi}_i$ and $\bar{\boldsymbol{u}} = c_s\boldsymbol{u}$ (henceforth, the bar above the $\boldsymbol{u}$ will be omitted). The most popular velocity sets are the D2Q9, D3Q15, D3Q19 and D3Q27 lattices.

[1]E.g. $\nabla_{ijk}^n = \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}\frac{\partial}{\partial x_k}$ where x_α is a coordinate direction.

3.2.2. Discretisation of forcing term

The particle distribution function has now been discretised into a discrete set of populations f_i, each with their own weight w_i and velocity $\boldsymbol{c}_i$. However, the forcing term in Equation (3.9), $(\partial f/\partial \boldsymbol{\xi})(\boldsymbol{f}/\rho)$ still needs to be discretised as well. This can be done as follows [44]. The Hermite expansion of $\partial f/\partial \boldsymbol{\xi}$ may be written as

$$\frac{\partial f}{\partial \boldsymbol{\xi}} = \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{a}_n \nabla \left(\omega\left(\boldsymbol{\xi}\right) \boldsymbol{H}_n\left(\boldsymbol{\xi}\right)\right). \tag{3.22}$$

However, from Equation (3.13), we have

$$\omega\left(\boldsymbol{\xi}\right) \boldsymbol{H}_n\left(\boldsymbol{\xi}\right) = (-1)^n \nabla^n \omega\left(\boldsymbol{\xi}\right), \tag{3.23}$$

and substituting this into Equation (3.22) results in

$$\frac{\partial f}{\partial \boldsymbol{\xi}} = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \boldsymbol{a}_n \nabla^{n+1} \omega\left(\boldsymbol{\xi}\right). \tag{3.24}$$

Applying Equation (3.23) once more results in

$$\frac{\partial f}{\partial \boldsymbol{\xi}} = -\omega\left(\boldsymbol{\xi}\right) \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{a}_n \boldsymbol{H}_{n+1}\left(\boldsymbol{\xi}\right) = -\omega\left(\boldsymbol{\xi}\right) \sum_{n=1}^{\infty} \frac{1}{n!} \boldsymbol{a}_{n-1} \boldsymbol{H}_n\left(\boldsymbol{\xi}\right). \tag{3.25}$$

Here, the $\boldsymbol{a}_{n-1}$ are already known from Equation (3.16)-(3.18), and the same $\boldsymbol{\xi}_i$ and $w_i = \omega(\boldsymbol{\xi}_i)$ as before can be used (including the scaling of ξ_i with c_s). Therefore, including terms up until second order, we obtain

$$F_i = \frac{\partial f_i}{\partial \boldsymbol{\xi}_i} \frac{F}{\rho} \approx w_i \left(\frac{\boldsymbol{c}_i}{c_s^2} + \frac{\left(\boldsymbol{c}_i \boldsymbol{c}_i^\top - c_s^2 \boldsymbol{I}\right) \boldsymbol{u}}{c_s^4} \right)^\top \boldsymbol{f}. \tag{3.26}$$

3.2.3. Discretisation in space and time

We have now discretised the populations, the forcing term and the collision term in velocity space, i.e. the Boltmzann equation now reads

$$\frac{\partial f_i}{\partial t} + \frac{\partial f_i}{\partial \boldsymbol{x}} \boldsymbol{c}_i = \Omega(f_i) + F_i = \Omega_i + F_i. \tag{3.27}$$

However, this equation is still continuous in both space and time. However, as it is a simple linear advection equation, the method of characteristics can be applied; defining the parametric variable $\eta(\boldsymbol{x}, t)$, we can write

$$\frac{df_i}{d\eta} = \frac{\partial f_i}{\partial t} \frac{dt}{d\eta} + \frac{\partial f_i}{\partial \boldsymbol{x}_i} = \Omega_i + F_i. \tag{3.28}$$

From comparison with Equation (3.27), we obtain

$$t = \eta + t_0 \tag{3.29}$$

$$\boldsymbol{x} = \eta \boldsymbol{c}_i + \boldsymbol{x}_0. \tag{3.30}$$

Thus, integrating Equation (3.28) between $\eta = 0$ and $\eta = \Delta t$, we obtain

$$f_i\left(\boldsymbol{x}_0 + \boldsymbol{c}_i \Delta t, t_0 + \Delta t\right) - f_i\left(\boldsymbol{x}_0, t_0\right) = \int_0^{\Delta t} \Omega_i\left(\boldsymbol{x}_0 + \boldsymbol{c}_i \eta, t_0 + \eta\right) d\eta + \int_0^{\Delta t} F_i\left(\boldsymbol{x}_0 + \boldsymbol{c}_i \eta, t_0 + \eta\right) d\eta. \tag{3.31}$$

Applying the BGK collision operator, letting $\boldsymbol{x} = \boldsymbol{x}_0$ and $t = t_0$, we can subsequently approximating the integrals on the right as

$$\int_0^{\Delta t} \Omega_i\left(\boldsymbol{x} + \boldsymbol{c}_i \eta, t + \eta\right) d\eta + \int_0^{\Delta t} F_i\left(\boldsymbol{x} + \boldsymbol{c}_i \eta, t + \eta\right) d\eta \approx -\frac{\Delta t}{\tau}\left(f_i - f_i^{\text{eq}}\right) + \left(1 - \frac{\Delta t}{2\tau}\right) F_i, \tag{3.32}$$

where τ takes the value

$$\nu = c_s^2\left(\tau - \frac{\Delta t}{2}\right), \tag{3.33}$$

instead of the expected $\nu = c_s^2\tau$, where ν is the viscosity, as shown by [47]. The reason for this shift in τ is that the leading order error term in the stress tensor is proportional to the 'true' stress tensor. Thus, by shifting the value of the viscosity, this leading order error term can be captured to result in the correct stress tensor. In similar fashion the coefficient in front of F_i is not equal to simply Δt; by scaling this coefficient, the leading order error term in the contribution of the forcing term is cancelled out, as shown by Guo et al. [48], increasing the order of accuracy of the method to two.

Note that this discretisation implies that collision and application of external forces occurs simultaneously at the beginning of a time-step, and therefore before streaming, which occurs during the remainder of a time-step. This also implies that output quantities should be measured *after* streaming, and one cannot simply interchange the order of collision and streaming.

Combining Equation (3.32) with Equation (3.31) results in the lattice Boltzmann equation (LBE)

$$f_i\left(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t\right) - f_i\left(\boldsymbol{x}, t\right) = \frac{\Delta t}{\tau}\left(f_i - f_i^{\text{eq}}\right) + \left(1 - \frac{\Delta t}{2\tau}\right)F_i. \tag{3.34}$$

This can be distinguished into two separate steps: first, the populations collide locally on their own nodes, via

$$\hat{f}_i\left(\boldsymbol{x}, t\right) = f_i\left(\boldsymbol{x}, t\right) - \frac{\Delta t}{\tau}\left(f_i\left(\boldsymbol{x}, t\right) - f_i^{\text{eq}}\left(\boldsymbol{x}, t\right)\right) + \left(1 - \frac{\Delta t}{2\tau}\right)F_i\left(\boldsymbol{x}, t\right), \tag{3.35}$$

and subsequently stream via

$$f_i\left(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t\right) = \hat{f}_i\left(\boldsymbol{x}, t\right). \tag{3.36}$$

3.2.4. Velocity moments

Corresponding to the chosen relaxation parameter τ, the macroscopic velocity moments become

$$\rho = \boldsymbol{\Pi}_0 = \sum_i f_i + \frac{\Delta t}{2}\sum_i F_i \tag{3.37}$$

$$\rho\boldsymbol{u} = \boldsymbol{\Pi}_1 = \sum_i f_i\boldsymbol{c}_i + \frac{\Delta t}{2}\sum_i F_i\boldsymbol{c}_i \tag{3.38}$$

$$\boldsymbol{\Pi}_2 = \left(1 - \frac{\Delta t}{2\tau}\right)\sum_i f_i\boldsymbol{c}_i\boldsymbol{c}_i^\top + \frac{\Delta t}{2\tau}\sum_i f_i^{\text{eq}}\boldsymbol{c}_i\boldsymbol{c}_i^\top + \frac{\Delta t}{2\tau}\left(1 - \frac{\Delta t}{2\tau}\right)\sum_i F_i\boldsymbol{c}_i\boldsymbol{c}_i^\top, \tag{3.39}$$

where Π^i denotes the ith velocity moment.

3.3. Chapman-Eskogg analysis

Consider again the discretised Boltzmann equation discretised in velocity space, Equation (3.34). It remains to be shown that this equation corresponds to the Navier-Stokes equations. For sake of generalisation, assume the forcing function Equation (3.26) to be of the form

$$F_i = w_i\left(\frac{B\boldsymbol{c}_i}{c_s^2} + \frac{\left(\boldsymbol{c}_i\boldsymbol{c}_i^\top - c_s^2\boldsymbol{I}\right):\boldsymbol{C}}{2c_s^4}\right)\boldsymbol{F}. \tag{3.40}$$

A Taylor expansion of Equation (3.34) results in

$$\Delta t\left(\frac{\partial}{\partial t} + \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{c}_i\right)f_i + \frac{\Delta t^2}{2}\left(\frac{\partial}{\partial t} + \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{c}_i\right)^2 f_i + \mathcal{O}\left(\Delta t^3\right) = -\frac{\Delta t}{\tau}f_i^{\text{neq}} + \left(1 - \frac{\Delta t}{2\tau}\right)F_i. \tag{3.41}$$

To show that the connection to the Navier-Stokes equation, we apply a Chapman-Enskog analysis [49]. To do so, we perform a multi-scale expansion of the populations and the time-derivative, i.e.

$$f_i = f_i^{\text{eq}} + \epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)} + \ldots \tag{3.42}$$

$$\frac{\partial}{\partial t} = \epsilon\frac{\partial}{\partial t}^{(1)} + \epsilon^2\frac{\partial^2}{\partial t^2}^{(2)} + \ldots \tag{3.43}$$

In similar fashion, we write the spatial derivative and the forcing term as

$$\frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{c}_i = \epsilon \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{c}_i \tag{3.44}$$

$$F_i = \epsilon F_i^{(1)} \tag{3.45}$$

$$B = \epsilon B^{(1)} \tag{3.46}$$

$$\boldsymbol{C} = \epsilon \boldsymbol{C}^{(1)}. \tag{3.47}$$

In the above equations, ϵ is a term of the order the Knudsen number (Kn); consequently the time-derivative is written as the sum of various components of different orders in Kn, and e.g. $(\partial/\partial t)^{(i)}$ is the part of the time-derivative associated with order ϵ^i. Substituting Equation (3.42)-(3.44) into Equation (3.41), and collecting all terms of order ϵ and ϵ^2 separately results in

$$\mathcal{O}(\epsilon): \left(\frac{\partial}{\partial t}^{(1)} + \frac{\partial}{\partial \boldsymbol{x}}^{(1)} \mathbf{c}\right) f_i^{\mathrm{eq}} - \left(1 - \frac{\Delta t}{2\tau}\right) F_i^{(1)} \qquad = -\frac{1}{\tau} f_i^{(1)} \tag{3.48}$$

$$\mathcal{O}\left(\epsilon^2\right): \frac{\partial}{\partial t}^{(2)} f_i^{\mathrm{eq}} + \left(\frac{\partial}{\partial t}^{(1)} + \frac{\partial}{\partial \boldsymbol{x}}^{(1)}\right)\left(1 - \frac{\Delta t}{2\tau}\right)\left(f_i^{(1)} + \frac{\Delta t}{2} F_i^{(1)}\right) \qquad = -\frac{1}{\tau} f_i^{(2)}. \tag{3.49}$$

The non-equilibrium populations should satisfy conservation of mass and momentum. Furthermore, as we assumed F_i to only consist of a term involving $\mathcal{O}(\epsilon)$, we require from Equation (3.37)-(3.38) that

$$\sum_i f_i^{(n)} = \begin{cases} -\frac{\Delta t}{2} \sum_i F_i^{(1)}, & \text{if } n = 1 \\ 0, & \text{if } n \geq 2 \end{cases} \tag{3.50}$$

$$\sum_i \boldsymbol{c}_i f_i^{(n)} = \begin{cases} -\frac{\Delta t}{2} \sum_i \boldsymbol{c}_i F_i^{(1)}, & \text{if } n = 1 \\ 0, & \text{if } n \geq 2. \end{cases} \tag{3.51}$$

Using these conditions, taking zeroth and first moments of Equation (3.44) results in

$$\frac{\partial}{\partial t}^{(1)} \rho + \frac{\partial}{\partial \boldsymbol{x}}^{(1)} \left(\rho \boldsymbol{u}\right) = 0 \tag{3.52}$$

$$\frac{\partial}{\partial t}^{(1)} \left(\rho \boldsymbol{u}\right) + \frac{\partial}{\partial \boldsymbol{x}}^{(1)} \Pi_2^{(0)} = B^{(1)} \boldsymbol{F}^{(1)}. \tag{3.53}$$

To obtain the Euler equations, we obtain $B = 1$. Similarly, taking zeroth and first moments of Equation (3.45) results in

$$\frac{\partial}{\partial t}^{(2)} \rho = 0 \tag{3.54}$$

$$\frac{\partial}{\partial t}^{(2)} \left(\rho \boldsymbol{u}\right) = \frac{\partial}{\partial \boldsymbol{x}}^{(1)} \Pi_2^{(1)}, \tag{3.55}$$

where $\Pi_2^{(1)}$ is the stress tensor, given by

$$\Pi_2^{(1)} = \boldsymbol{\sigma} = \left(\tau - \frac{1}{2}\right) c_s^2 \Delta t \rho \left(\left(\frac{\partial}{\partial \boldsymbol{x}}^{(1)} \boldsymbol{u}^\top\right) + \left(\frac{\partial}{\partial \boldsymbol{x}}^{(1)} \boldsymbol{u}^\top\right)^\top\right) + \tag{3.56}$$

$$\Delta t \left[\left(\tau - \frac{1}{2}\right)\left(\left(\boldsymbol{u}\left(\boldsymbol{F}^{(1)}\right)^\top + \left(\boldsymbol{u}\left(\boldsymbol{F}^{(1)}\right)^\top\right)^\top\right) - \left(\frac{\tau}{2} - \frac{1}{4}\right)\left(\boldsymbol{C}\left(\boldsymbol{F}^{(1)}\right)^\top + \boldsymbol{C}\left(\boldsymbol{F}^{(1)}\right)^\top\right)^\top\right)\right]. \tag{3.57}$$

Clearly, by setting τ such that $\nu = \left(\tau - \frac{1}{2}\right) c_s^2 \Delta t$ and $\boldsymbol{C} = 2\boldsymbol{u}$ we eliminate the leading errors in the stress tensor, justifying the choice in discretisation of the integral in Equation (3.31) and form of the force discretisation in Equation (3.26). Combining Equations (3.52) with (3.54) and (3.53) with (3.55) and reversing the multi-scale expansion results in

$$\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial \boldsymbol{x}}\left(\rho \boldsymbol{u}\right) = 0 \tag{3.58}$$

$$\frac{\partial}{\partial t}\left(\rho \boldsymbol{u}\right) + \frac{\partial}{\partial \boldsymbol{x}}\left(\rho \boldsymbol{u}\boldsymbol{u}^\top\right) = -\frac{\partial}{\partial \boldsymbol{x}} p + \nu \frac{\partial}{\partial \boldsymbol{x}}\left[\left(\frac{\partial}{\partial \boldsymbol{x}}^\top \boldsymbol{u}^\top\right) + \left(\frac{\partial}{\partial \boldsymbol{x}}^\top \boldsymbol{u}^\top\right)^\top\right] + \boldsymbol{F}, \tag{3.59}$$

where $p = c_s^2 \rho$ is the pressure, which is why c_s is called the speed of sound in the lattice Boltzmann method. As is apparent, Equations (3.58) and (3.59) correspond to the Navier-Stokes equations. For a detailed derivation of these relations, the reader is referred to e.g. [48], which shows the errors introduced by various other forcing schemes.

3.4. Application of the lattice Boltzmann equation

Now that the discretised lattice Boltzmann equation has been obtained and its connection to the Navier-Stokes equation has been demonstrated, a brief discussion of the practical implementation of the lattice Boltzmann method (LBM) is provided. For an extensive discussion the practical implementation of the LBM, the reader is referred to e.g. Kruger, and Guo and Shu [2, 3], and only the fundamentals of the various components of a typical LBM implementation are discussed.

3.4.1. Initial conditions

An initial condition needs to be imposed on the populations. A naive way to do so would be to set

$$f_i(\boldsymbol{x}, t)\big|_{t=0} = f_i^{\text{eq}}\left(\rho_0, \boldsymbol{u}_0\right), \tag{3.60}$$

where ρ_0 is some constant initial density and $\boldsymbol{u}_0$ is the initial velocity field. However, in case $\boldsymbol{u}_0$ is not constant (or in case of an external forcing), then the density cannot be assumed constant as the pressure Poisson equation needs to be satisfied, i.e.

$$\Delta p_0 = -\frac{\partial}{\partial \boldsymbol{x}} \left(\frac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{u}_0 \boldsymbol{u}_0^\top \right) \right)^\top + \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{F}, \tag{3.61}$$

and this is not satisfied when $\boldsymbol{u}_0$ varies in space, but ρ_0 is assumed to be constant (since $p = c_s^2 \rho$). Therefore, Caiazzo, and Mei et al. [50, 51] propose the following initialisation algorithm:

1. Set an initial density distribution $\rho_0(\boldsymbol{x}, t)$, e.g. $\rho_0(\boldsymbol{x}, t) = 1$.
2. Calculate the associated equilibrium distribution and forcing from Equation (3.21) and (3.26), using $\boldsymbol{u} = \boldsymbol{u}_0(\boldsymbol{x}, t)$ and $\rho = \rho_0(\boldsymbol{x}, t)$.
3. Perform the collision according to Equation (3.35).
4. Perform the streaming according to Equation (3.36).
5. Calculate the new density ρ' according to Equation (3.37).
6. Compare the new density with the previous density ρ. If the difference is smaller than a certain tolerance, the algorithm is terminated. If not, set $\rho = \rho'$, recompute the equilibrium distribution and forcing term with $\boldsymbol{u} = \boldsymbol{u}_0(\boldsymbol{x}, t)$ and the updated density, and go back to step 3.

This results in populations that relax towards the desired velocity field, whilst conserving mass in the complete system. Furthermore, it means that no Poisson solver is necessary for the pressure, avoiding the need to implement such a solver.

It should be noted that the initial pressure distribution only needs to be solved once, so the computational effort of the procedure to determine the initial pressure distribution is relatively unimportant. Furthermore, any unphysical artefacts that are present in the initial solution are inevitably smoothed out due to the unsteady nature of the LBM. Therefore, assuming the simulation is run for a long enough time to allow the solution to properly reach a steady-state simulation, the way the initial condition is imposed is not of utmost importance.

3.4.2. Boundary conditions

Imposing boundary conditions in the LBM is inherently different from how they can be imposed in traditional CFD solvers, such as finite volume methods. Whereas in traditional CFD solvers, boundary conditions may be directly imposed on the macroscopic quantities, in LB methods, boundary conditions can only be applied to the mesoscopic populations. Chapter 4 and 5 discuss two ways of applying boundary conditions to moving boundaries inside the fluid domain, namely the immersed boundary method and immersed interface method. Chapter 6 discusses a way of applying boundary conditions to the outer edges of the domain.

3.4.3. External forces

The inclusion of external forces has already been discussed in Section 3.2.2 and 3.2.3 in which the *Guo forcing* has been discussed, developed by Guo et al. in [48], which is a small variation on the forcing scheme developed by He et al. [49]. However, in literature, a wealth of forcing schemes exist, such as those by Wagner [52], Ladd and Verberg [53] and Luo [54].

All of these forcing schemes can be generalised as follows. First, we write the collision step of Equation (3.35) as

$$\hat{f}_i(\boldsymbol{x},t) = f_i(\boldsymbol{x},t) - \frac{\Delta t}{\tau}\left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right) + S_i(\boldsymbol{x},t), \tag{3.62}$$

with S_i of the form

$$S_i = w_i\left(\frac{\boldsymbol{B}\cdot\boldsymbol{e}_i}{c_s^2} + \frac{\boldsymbol{C}:\left(\boldsymbol{e}_i\boldsymbol{e}_i^\top - c_s^2\boldsymbol{I}\right)}{2c_s^4}\right). \tag{3.63}$$

Secondly, the first velocity moment of the physical velocity (Equation (3.38)) is written as

$$\rho\boldsymbol{u} = \boldsymbol{\Pi}_1 = \sum_i f_i\boldsymbol{c}_i + k\sum_i F_i\boldsymbol{c}_i, \tag{3.64}$$

and the first velocity moment of the velocity used to compute equilibrium populations is written as

$$\rho\boldsymbol{u}^{\mathrm{eq}} = \boldsymbol{\Pi}_1 = \sum_i f_i^{\mathrm{eq}}\boldsymbol{c}_i + m\sum_i F_i\boldsymbol{c}_i. \tag{3.65}$$

The differences between forcing schemes are then in what parameters are used for k, m, $\boldsymbol{B}$ and $\boldsymbol{C}$. A comprehensive review of various forcing methods is provided in Bawazeer et al. [55], where twelve different forcing methods are compared using a benchmark case of natural convection in a differentially heated square cavity, where the differential heating causes a buoyancy force. The results were evaluated for different Rayleigh numbers and showed no significant difference between the forcing schemes, with e.g. the maximum horizontal component of the velocity differing by at most around 1% between different forcing schemes. Meanwhile, the difference w.r.t. a benchmark solution based on a semi-implicit spectral method presented by Quere and De Roquefortt was in the order of 10% for large Rayleigh numbers, showing deviations in the results are likely caused by other factors than the choice of forcing scheme.

It should be noted that this was performed for a steady simulation, whereas for example Guo et al. [48] showed that their choice for $\boldsymbol{C}$ was necessary to eliminate non-physical effects occurring due to temporal variations in the force that were present in the forcing schemes of e.g. Ladd and Verberg [53]. Thus, although Bawazeer et al. [55] shows that the differences between forcing schemes are generally negligible for a steady-state simulation, they are likely to become more notable for transient simulations. Therefore, the Guo forcing seems superior from a theoretical point of view for transient simulations.

3.4.4. Dimensionalisation

As mentioned in Section 3.1.1, all parameters in the LBM are non-dimensionalised and therefore in lattice units. The conversion between lattice and physical units is very simple, fortunately [56]. Consider a lattice node spacing Δx_l, lattice time step Δt_l and lattice speed of sound of $c_{s,l}$. The viscosity in lattice units, ν_l, is given by Equation (3.33). The relation between the physical and lattice viscosity is given by

$$\nu_p = \nu_l\frac{\Delta x^2}{\Delta t} = c_{s,1}^2\left(\tau - \frac{1}{2}\right)\frac{\Delta x^2}{\Delta t}, \tag{3.66}$$

where

$$\Delta x = \frac{\Delta x_p}{\Delta x_l} \tag{3.67}$$

$$\Delta t = \frac{\Delta t_p}{\Delta t_l}, \tag{3.68}$$

where Δx_p is the physical node spacing and Δt_p the physical time step. Thus, we have

$$\Delta x = \frac{\nu_p}{c_{s,1}c_{s,p}\left(\tau - \frac{1}{2}\right)} \tag{3.69}$$

$$\Delta t = \frac{\nu_p}{c_{s,p}^2\left(\tau - \frac{1}{2}\right)}, \tag{3.70}$$

where $c_{s,p}$ is the physical speed of sound. Δx_p and Δt_p can then be computed from Equation (3.67) and (3.68), respectively.

It is evident that there is only a single free parameter. The physical speed of sound is a fixed value that follows from the atmospheric conditions; the physical viscosity similarly is a flow parameter; the lattice speed of sound is a fixed value belonging to the lattice chosen; the lattice node spacing and time step are usually taken to be unity[2]. Therefore, choosing a certain time step Δt_p determines the value of τ, which in turn constrains the value of Δx_p, and vice versa. This means that the refinement of the time discretisation cannot be chosen independently from the spatial discretisation, which is a disadvantage compared to traditional CFD methods, where both the spatial and time step size can be altered independently. To aggravate this, the value of τ directly affects the stability of the simulation, as extensively discussed in Section 3.5, as the simulation may become unstable if τ becomes too small. Consequently, tuning the input parameters may prove a challenge; Krüger et al. [2] details a number of strategies one can use to obtain a satisfactory set of input parameters.

3.4.5. Overview of operations taken in an LBM simulation

The main steps undertaken in a typical LBM simulation have now been discussed, and a summary of the sequencing of the steps is provided in Figure 3.1. It should be noted that certain applications of the LBM may require additional steps that are not listed here; for example, fluid-structure-interaction utilising the LBM as fluid solver requires additional steps that are explained in more detail in Chapter 6.

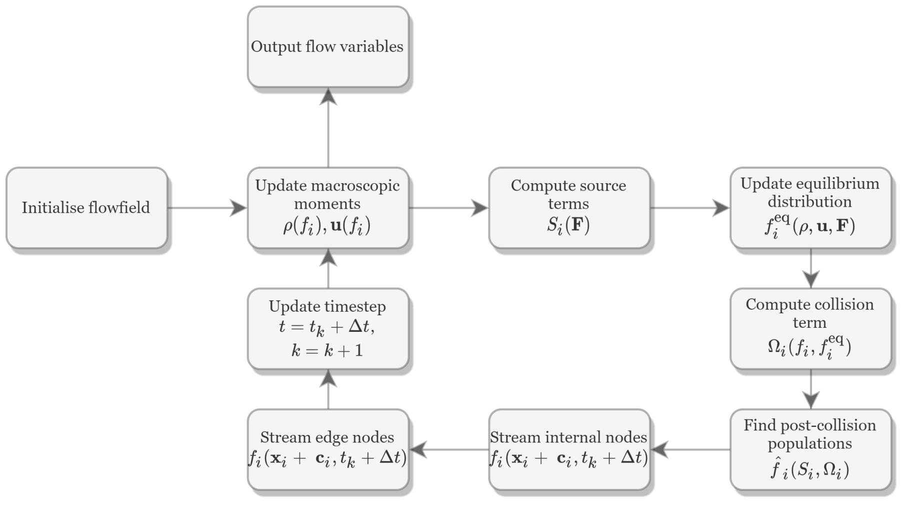

Output flow variables
Initialise flowfield
Update macroscopic moments $\rho(f_i), \mathbf{u}(f_i)$
Compute source terms $S_i(\mathbf{F})$
Update equilibrium distribution $f_i^{\mathrm{eq}}(\rho, \mathbf{u}, \mathbf{F})$
Update timestep $t = t_k + \Delta t,$ $k = k + 1$
Compute collision term $\Omega_i(f_i, f_i^{\mathrm{eq}})$
Stream edge nodes $f_i(\mathbf{x}_i + \mathbf{c}_i, t_k + \Delta t)$
Stream internal nodes $f_i(\mathbf{x}_i + \mathbf{c}_i, t_k + \Delta t)$
Find post-collision populations $\hat{f}_i(S_i, \Omega_i)$

Figure 3.1: Generic sequencing of operations in an LBM simulation.

3.5. Advanced collision models

The BGK operator, although attractive due to its simplicity, suffers from some issues. First of all, as is apparent from Equation (3.33), τ has to be greater than $\frac{1}{2}$, as $\tau = 1/2$ would lead to zero viscosity. However, even in the limit of $\tau \to 1/2$, use of the BGK operator leads to stability issues in the solution, as shown by for example Luo et al. [57]. These stability issues can be alleviated by using a finer mesh discretisation, but this is naturally accompanied with an increase in computational cost.

Secondly, the BGK operator is not Galilean invariant. Considering the equilibrium distribution in Equation (3.21), for a given $\Delta \boldsymbol{u}$, if we let $\hat{f}_i^{\mathrm{eq}}(\boldsymbol{u}) = f_i^{\mathrm{eq}}(\boldsymbol{u} + \Delta \boldsymbol{u})$, then $\hat{f}_i^{\mathrm{eq}}(\boldsymbol{u} - \Delta \boldsymbol{u}) \neq f_i^{\mathrm{eq}}(\boldsymbol{u})$.

Thirdly, for certain applications, the relaxation parameter directly influences the accuracy of the simulations. Clearly, the relaxation parameter apparently both influences the physics of the flow (as it directly influences the Reynolds number) and the accuracy of the simulation itself.

[2]For multigrid methods, where different parts of the domain have different levels of grid refinement, one may choose to e.g. define $\Delta x_l = 1$ and $\Delta t_l = 1$ for the finest mesh, use $\Delta x_l = 2$ and $\Delta t_l = 2$ for a grid that is twice as coarse as the finest mesh, $\Delta x_l = 4$ and $\Delta t_l = 4$ for the next grid, and so on. However, also in this case it is clear that the values of Δx_l and Δt_l are fixed.

Fourthly, on a more fundamental level, using a single relaxation time[3] means that all velocity moments are relaxed at the same rate. However, in reality, the velocity moments are independent from each other, and thus it seems attractive to be able to relax them independently.

To mitigate these issues, various, more advanced collision models have been developed. This section discusses a number of them.

3.5.1. Multiple-relaxation-time models

Multiple-relaxation-times (MRT) models deal particularly well with the issue of having only a single relaxation time, and in the process, improve the accuracy and stability as well. The fundamental basis of these types of models is to transform the populations into a moment space, relax the individual moments, and then transform back to population space. In other words, the collision operation in Equation (3.35) is written as

$$\hat{f}_i(\boldsymbol{x},t) = f_i(\boldsymbol{x},t) - \Delta t \boldsymbol{M}^{-1}\boldsymbol{S}\left(\boldsymbol{M} f_i(\boldsymbol{x},t) - \boldsymbol{M} f_i^{\mathrm{eq}(\boldsymbol{x},t)}\right) + \left(1 - \frac{\Delta t}{2\tau}\right) F_i(\boldsymbol{x},t), \tag{3.71}$$

where the product $\boldsymbol{M} f_i = \boldsymbol{m}_i$ are the contributions of the populations f_i to each of the moments and $\boldsymbol{M} f_i^{\mathrm{eq}} = \boldsymbol{m}_i$ are the contributions of each population's equilibrium distribution to the equilibrium moments.

The question is now how to construct the matrices $\boldsymbol{M}$ and $\boldsymbol{S}$. The objective of $\boldsymbol{M}$ is to transform the populations f_i to a set of moments. In a DdQq model, $\boldsymbol{M}$ clearly needs to be of size $q \times q$. Thus, a set of q moments needs to be chosen that will form the basis of the moment space.

Multiple-relaxation-time (MRT) models can be constructed in a variety of ways. Two notable methods will be described in this section. The first method is to use Hermite polynomials to construct the moment space, as proposed by Dellar [58]. The second method is to construct polynomials based on the components of the link velocities and orthogonalise these, as proposed by Lallemand and Luo [59]. It should be noted that the second method has several advantages over the former, was developed earlier and is the conventionally used model. However, the Hermite-based MRT can be argued to be more physically intuitive, which is why it will be discussed first.

Hermite-based MRT It was previously seen in Section 3.2.1 that there was an intimate connection between a Hermite expansion and the velocity moments, which makes the use of Hermite generating function a natural candidate to construct more moments. Dellar [58] therefore proposed to construct a basis of moments based on Hermite polynomials. To do so, we can use a Hermite moment generating function of the form [60]

$$\mathscr{H}(\boldsymbol{X}) = \int e^{\boldsymbol{\xi}^\top \boldsymbol{X} - c_s^2 \boldsymbol{X}^\top \boldsymbol{X}/2} f(\boldsymbol{\xi})\, d\xi. \tag{3.72}$$

The contribution of population f_i with velocity $\boldsymbol{c}_i$ to the moment generating function is therefore

$$\mathscr{H}_i(\boldsymbol{X}) = \int e^{\boldsymbol{\xi}^\top \boldsymbol{X} - c_s^2 \boldsymbol{X}^\top \boldsymbol{X}/2} f_i(\boldsymbol{\xi} - \boldsymbol{c}_i)\, d\xi = e^{\boldsymbol{c}_i^\top \boldsymbol{X} - c_s^2 \boldsymbol{X}^\top \boldsymbol{X}/2}, \tag{3.73}$$

which can be written as a power series as

$$\mathscr{H}_i(\boldsymbol{X}) = \sum_{n=0}^{\infty} \frac{\left(\boldsymbol{c}_i^\top \boldsymbol{X} - c_s^2 \boldsymbol{X}^\top \boldsymbol{X}/2\right)^n}{n!}. \tag{3.74}$$

Therefore, since the contribution to the kth moment is given by $\boldsymbol{m}_i^{(k)} = \partial^k \mathscr{H}_i / \partial \boldsymbol{X}^k$ evaluated at $\boldsymbol{X} = \boldsymbol{0}$, we obtain for the first three moments:

$$\boldsymbol{m}_i^{(0)} = 1 \tag{3.75}$$

$$\boldsymbol{m}_i^{(1)} = \boldsymbol{c}_i \tag{3.76}$$

$$\boldsymbol{m}_i^{(2)} = \boldsymbol{c}_i \boldsymbol{c}_i^\top - c_s^2 \boldsymbol{I}. \tag{3.77}$$

(3.78)

[3]The use of a single relaxation time is why these models are also referred to as single-relaxation-time SRT models.

As a practical example, consider a D2Q9 lattice. The zeroeth moment will be

$$\boldsymbol{m}^{(0)} = \sum_{i=0}^{q=8} m_i^{(0)} f_i = \sum_{i=0}^{q=8} f_i. \tag{3.79}$$

The x-component of the first moment will be

$$\boldsymbol{m}_x^{(1)} = \sum_{i=0}^{q=8} m_{i,x}^{(1)} f_i = \sum_{i=0}^{q=8} c_{i,x} f_i, \tag{3.80}$$

and e.g. the xy-component of the second moment will be

$$\boldsymbol{m}_{xy}^{(2)} = \sum_{i=0}^{q=8} m_{i,xy}^{(2)} f_i = \sum_{i=0}^{q=8} c_{i,x} c_{i,y} f_i. \tag{3.81}$$

We can now make a choice in which moments to include. For a D2Q9 lattice, the natural candidates would be the following:

- $\boldsymbol{m}^{(0)}$, which corresponds to the density moment.
- $\boldsymbol{m}_x^{(1)}$ and $\boldsymbol{m}_y^{(1)}$, which correspond to the momentum.
- $\boldsymbol{m}_{xx}^{(2)}$, $\boldsymbol{m}_{xy}^{(2)}$ and $\boldsymbol{m}_{yy}^{(2)}$, which correspond to the components of the stress tensor.
- $\boldsymbol{m}_{xxy}^{(3)}$, $\boldsymbol{m}_{xyy}^{(3)}$ and $\boldsymbol{m}_{xxyy}^{(4)}$, a higher-order moment. The choice for $\boldsymbol{m}_{xxyy}^{(4)}$ instead of e.g. $\boldsymbol{m}_{xxx}^{(3)}$ is due to the fact that $\boldsymbol{m}_{xxx}^{(3)}$ equals $\boldsymbol{m}_x^{(3)}$, as shown at the end of this section.

Subsequently, the matrix $\boldsymbol{M}$ is constructed as follows:

$$\boldsymbol{M} = \left[\left(\hat{\boldsymbol{m}}^{(0)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_x^{(1)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_y^{(1)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_{xx}^{(2)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_{yy}^{(2)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_{xy}^{(2)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_{xxy}^{(3)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_{xyy}^{(3)}\right)^\top \quad \left(\hat{\boldsymbol{m}}_{xxyy}^{(4)}\right)^\top \right]^\top, \tag{3.82}$$

where $\hat{\boldsymbol{m}}_{\boldsymbol{\alpha}}^{(k)}$ is the vector consisting consisting of the elements $\boldsymbol{m}_{\boldsymbol{\alpha},i}^{(k)}$, where $\boldsymbol{\alpha}$ is a permutation of x and y of appropriate size. Using the velocity set of a D2Q9 lattice, this results in

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_1^\top \\ \boldsymbol{M}_2^\top \\ \boldsymbol{M}_3^\top \\ \boldsymbol{M}_4^\top \\ \boldsymbol{M}_5^\top \\ \boldsymbol{M}_6^\top \\ \boldsymbol{M}_7^\top \\ \boldsymbol{M}_8^\top \\ \boldsymbol{M}_9^\top \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\ 0 & -\frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{2}{3} & -\frac{2}{3} & -\frac{2}{3} & \frac{2}{3} \\ 0 & 0 & -\frac{1}{3} & 0 & \frac{1}{3} & \frac{2}{3} & \frac{2}{3} & -\frac{2}{3} & -\frac{2}{3} \\ \frac{1}{9} & -\frac{2}{9} & -\frac{2}{9} & -\frac{2}{9} & -\frac{2}{9} & \frac{4}{9} & \frac{4}{9} & \frac{4}{9} & \frac{4}{9} \end{bmatrix}. \tag{3.83}$$

It should be noted that not all rows in this are orthogonal to each other. However, it is preferable to have all moments be orthogonal to each other, such that they can be relaxed independently from each other.

Gramm-Schmidt-based MRT To solve this, Lallemand and Luo [59] proposed a different approach. They construct the row vectors in $\boldsymbol{M}$ as polynomials in the x- and y-components of the velocities $\boldsymbol{c}_i$, and orthogonalise each vector using the Gramm-Schmidt procedure [61]. The first three orthogonal vectors still correspond to the density (ρ), momentum in x (j_x) and momentum in y (j_y). In other words, the entries in the row $\boldsymbol{M}_1$ are given by $\boldsymbol{M}_{1,i} = 1$; the entries in the row $\boldsymbol{M}_4$ are given by $\boldsymbol{M}_{4,i} = \boldsymbol{c}_{i,x}$; the entries in the row $\boldsymbol{M}_6$ are given by $\boldsymbol{M}_{6,i} = \boldsymbol{c}_{i,y}$[4].

The remaining rows are constructed as follows:

[4]The row vectors are ordered based on the order of the tensor they correspond to; as will be seen soon, $\boldsymbol{M}_2$ and $\boldsymbol{M}_3$ will correspond to scalars, whereas the momentum in x and y correspond to vector components.

- $\boldsymbol{M}_2$ is based on the polynomial $\boldsymbol{c}_{i,x}^2 + \boldsymbol{c}_{i,y}^2 = ||\boldsymbol{c}_i||^2$. This row corresponds to the kinetic energy e. Orthogonalising it with respect to $\boldsymbol{M}_1$, $\boldsymbol{M}_4$ and $\boldsymbol{M}_6$ results in

$$\boldsymbol{M}_{2,i} = 3||\boldsymbol{c}_i||^2 - 4. \tag{3.84}$$

- $\boldsymbol{M}_3$ is based on the polynomial $\left(\boldsymbol{c}_{i,x}^2 + \boldsymbol{c}_{i,y}^2\right)^2 = ||\boldsymbol{c}_i||^4$. This row corresponds to the square of the energy. Orthogonalisation results in

$$\boldsymbol{M}_{2,i} = 4 - \frac{21}{2}||\boldsymbol{c}_i||^2 + \frac{9}{2}||\boldsymbol{c}_i||^4. \tag{3.85}$$

- $\boldsymbol{M}_5$ and $\boldsymbol{M}_7$ are based on the x- and y-components of the polynomial $\boldsymbol{c}_i||\boldsymbol{c}_i||^2$, which corresponds to the energy flux q_x and q_y, respectively. Orthogonalisation results in

$$\boldsymbol{M}_{5,i} = \boldsymbol{c}_{i,x}\left(-5 + 3||\boldsymbol{c}_i||^2\right) \tag{3.86}$$

$$\boldsymbol{M}_{7,i} = \boldsymbol{c}_{i,y}\left(-5 + 3||\boldsymbol{c}_i||^2\right). \tag{3.87}$$

- $\boldsymbol{M}_8$ is based on the polynomial $\boldsymbol{c}_{i,x}^2 - \boldsymbol{c}_{i,y}^2$, which corresponds to the diagonal component of the stress tensor. As this expression is already orthogonal to the previous ones, we simply obtain

$$\boldsymbol{M}_{8,i} = \boldsymbol{c}_{i,x}^2 - \boldsymbol{c}_{i,y}^2. \tag{3.88}$$

- $\boldsymbol{M}_9$ is based on the polynomial $\boldsymbol{c}_{i,x}\boldsymbol{c}_{i,y}$, which corresponds to the off-diagonal component of the stress tensor. This expression is already orthogonal, and thus

$$\boldsymbol{M}_{9,i} = \boldsymbol{c}_{i,x}\boldsymbol{c}_{i,y}. \tag{3.89}$$

Note that some of the row vectors have been scaled such that the entries are integers. There is no explicit need to do so, but it simplifies computations when necessary [62].

In conclusion, we obtain

$$\boldsymbol{M} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -4 & -1 & -1 & -1 & -1 & 2 & 2 & 2 & 2 \\ 4 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & -2 & 0 & 2 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \\ 0 & 0 & -2 & 0 & 2 & 1 & 1 & -1 & -1 \\ 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \end{bmatrix}. \tag{3.90}$$

Subsequently, the relaxation matrix $\boldsymbol{S}$ is diagonal, of the form

$$\boldsymbol{S} = \operatorname{diag}\left(\omega_\rho, \omega_e, \omega_\epsilon, \omega_j, \omega_q, \omega_j, \omega_q, \omega_\nu, \omega_\nu\right). \tag{3.91}$$

Similar models can be proposed for 3D lattices. d'Humières et al. [63] developed MRT models for the D3Q15 and D3Q19 lattices.

Note that to preserve isotropy, some moments share the same relaxation rate. It can be shown via a Chapman-Enskog analysis [2] that ω_ν and ω_ϵ should satisfy the following relations:

$$\eta = \rho c_s^2\left(\frac{1}{\omega_\nu} - \frac{\Delta t}{2}\right) \tag{3.92}$$

$$\eta_{\mathrm{B}} = \rho c_s^2\left(\frac{1}{\omega_e} - \frac{\Delta t}{2}\right) - \frac{\eta}{3} \tag{3.93}$$

where ν is the shear viscosity and η_B the bulk viscosity. The relaxation rates ω_ρ and ω_j may be set to 0 as those moments will always equal the equilibrium moments regardless, as the collision operator satisfies conservation of mass and momentum. The relaxation parameters ω_ϵ and ω_q are arbitrary, however, and no single, universal approach to determinate appropriate relaxation values this exists. Particularly for D3Q15, D3Q19 and D3Q27 models, where the number of relevant moments (and therefore relaxation rates) is increased significantly, this commonly poses a problem.

There are two main advantages to the Gramm-Schmidt approach compared to the Hermite approach. First, it is clear that the moments are orthogonal to each other, which is not the case for the Hermite approach. Secondly, the moments obtained from the Hermite approach suffers from degeneracy due to the discretised velocity space. For example, consider the contributions of f_i to the third-order velocity moment,

$$m_i^{(3)} = \boldsymbol{c}_i \otimes \boldsymbol{c}_i \otimes \boldsymbol{c}_i. \tag{3.94}$$

Consequently,

$$\boldsymbol{m}_{i,xxx}^{(3)} = \sum_{i=0}^{q=8} m_{i,xxx}^{(3)} f_i = \sum_{i=0}^{q=8} c_{i,x}c_{i,x}c_{i,x}f_i. \tag{3.95}$$

However, since $c_{i,x}$ can only take values from $[-1,0,1]$, this means that this simply reduces to

$$\boldsymbol{m}_{i,xxx}^{(3)} = c_{i,x}f_i, \tag{3.96}$$

which would coincide with $\boldsymbol{m}_{i,x}^{(1)}$. This particularly leads to problems for the D3Q15 and D3Q19 velocity sets [2]. These advantages are the main reasons why the Gramm-Schmidt approach is usually preferred over the Hermite approach.

One note should be made regarding the use of the term of multiple-relaxation-time models. So far, we have seen the raw moments being used to form a basis of a moment space. However, since the introduction of MRT models, as we shall see in Section 3.5.3 and 3.5.4, other models have been developed that also transform the populations to a moment space, but use a different moment basis, e.g. central moments or cumulants. Although these methods also use multiple relaxation times, the term MRT is usually restricted to raw moment based methods using multiple relaxation times.

3.5.2. Two-relaxation-time models

As mentioned previously, there is a large number of arbitrary relaxation parameters that need to be tuned in the MRT approach. The two-relaxation-time (TRT) model, introduced by Ginzburg [64], simplifies this significantly, by using the same relaxation time (ω^+) for all the even-order moments (ρ, e, ϵ, p_{xx}, p_{xy}) and a different relaxation time (ω^-) for all the odd-order moments (j_x, j_y, q_x, q_y). The viscosity is then related to the relaxation parameter ω^+ via

$$\nu = c_s^2\left(\frac{1}{\omega^+\Delta t} - \frac{1}{2}\right), \tag{3.97}$$

and ω^- is the only free parameter. The choice for ω^- affects both the stability and the accuracy, through the so-called 'magic' parameter

$$\Lambda = \Lambda^+\Lambda^- = \left(\frac{1}{\omega^+\Delta t} - \frac{1}{2}\right)\left(\frac{1}{\omega^-\Delta t} - \frac{1}{2}\right). \tag{3.98}$$

Different choices for Λ (by altering ω^-) lead to different results for accuracy and stability. For example, Ginzburg [65] shows that the following values of Λ have specific benefits:

- Setting $\Lambda = 3/16$ results in a wall being placed at the midpoint between nodes in a bounce-back scheme, a commonly used boundary condition in LBM.
- The choice of $\Lambda = 1/6$ removes a fourth-order error from the density in a straight Poiseuille flow, beneficial for diffusion dominated problems.
- The choice of $\Lambda = 1/12$ removes a third-order error from the density in a straight Poiseuille flow, beneficial for advection dominated problems.

Ginzburg [66] also showed that the choice of $\Lambda = 1/4$ is optimal in terms of stability; that is, the specific values of ω^+ and ω^- do not affect the stability limits.

It should be noted that in TRT models, the populations are not typically transferred to moment space using the moment matrix $\boldsymbol{M}$ and relaxation matrix $\boldsymbol{S}$ [2]. Instead, the populations are decomposed into symmetric and asymmetric contributions, and the collision step is formulated as (excluding the contribution of the force)

$$\hat{f}_i(\boldsymbol{x},t) = f_i(\boldsymbol{x},t) - \omega^+\Delta t\left(f_i^+(\boldsymbol{x},t) - f_i^{\mathrm{eq},+}(\boldsymbol{x},t)\right) - \omega^-\Delta t\left(f_i^-(\boldsymbol{x},t) - f_i^{\mathrm{eq},-}(\boldsymbol{x},t)\right), \tag{3.99}$$

with

$$f_i^+ = \frac{f_i + f_{\bar{i}}}{2} \tag{3.100}$$

$$f_i^- = \frac{f_i - f_{\bar{i}}}{2} \tag{3.101}$$

$$f_i = f_i^+ + f_i^- \tag{3.102}$$

$$f_{\bar{i}} = f_i^+ - f_i^-, \tag{3.103}$$

where $\bar{i}$ is the link opposite to link i. The symmetric components are denoted by a plus sign, the asymmetric components by a minus sign. The symmetric and asymmetric equilibrium populations are similarly defined.

3.5.3. Cascaded lattice Boltzmann method

Geier et al. [67] observed that although MRT operators perform better than SRT operators, they still suffer from instability issues. They point to the fact that MRT operators do not adequately satisfy Galilean invariance. There are three reasons for this:

- The equilibrium distribution, Equation (3.21), is not Galilean invariant due to its truncation of higher order terms. These higher order terms are related to small-wavelength phenomena in the flow, whereas instabilities often arise from these small-wavelengths.
- For 3D applications, the D3Q13, D3Q15 and D3Q19 are usually chosen over the D3Q27 model, due to the increased memory requirement for the D3Q27 model, with generally little additional accuracy for small Reynolds numbers. However, consider e.g. the D3Q19 model, where no velocity component in the xyz-direction exists. Consequently, the moment μ_{xyz} is not independent from the other moments in the flow. This is important for flow with short wavelength features; μ_{xyz} can be seen as the advection of μ_{xy} (corresponding to the shear rate of the flow) in the z-direction. Ignoring this moment means that advection of the shear rate is neglected, which is invalid for flow with small-wavelength phenomena.
- It is natural to assume that central moments[5] are the moments that *should* be relaxed, as these moments are Galilean invariant. A central moment of a given order is a linear combinations of raw moments up to the same order. This means that if the raw moments are relaxed, higher order central moments are inadvertently relaxed simultaneously. It is therefore preferable to relax the central moments directly.

Geier et al. therefore proposed the cascaded lattice Boltzmann method (CLBM) to mitigate with these issues. The collision operation is replaced with a scattering step

$$\hat{\boldsymbol{f}} = \boldsymbol{f} + \boldsymbol{K}\boldsymbol{k}, \tag{3.104}$$

where $\boldsymbol{f}$ is the vector containing the populations, $\boldsymbol{K}$ is a scattering matrix for which each column corresponds to a raw moment (essentially, it is the transpose of the $\boldsymbol{M}$ matrix in the Gramm-Schmidt MRT approach). The vector $\boldsymbol{k}$ controls the relaxations of each of those moments; the components of $\boldsymbol{k}$ are found by solving

$$\bar{m}^{\mathrm{eq}}_{x^m y^n} = \sum_i \left(f_i + (\boldsymbol{K}\boldsymbol{k})_i\right)\left(\boldsymbol{c}_{i,x} - \boldsymbol{u}_x\right)\left(\boldsymbol{c}_{i,y} - \boldsymbol{u}_y\right)/\rho, \tag{3.105}$$

for $m, n = 0, 1, 2$, and where $\bar{m}_{x^m y^n}$ represents a central moment. Note that the values of $\bar{m}^{\mathrm{eq}}_{x^m y^n}$ can be found analytically by taking central moments from the equilibrium distribution, i.e.

$$\bar{m}^{\mathrm{eq}}_{x^m y^n} = \sum_i f_i^{\mathrm{eq}}\left(\boldsymbol{c}_{i,x} - \boldsymbol{u}_x\right)^m\left(\boldsymbol{c}_{i,y} - \boldsymbol{u}_y\right)^n, \tag{3.106}$$

[5]That is, moments with respect to the flow velocity, contrary to the raw moments that were utilised for the derivation of the MRT method.

and expressions for the components of $\boldsymbol{k}$ can be found symbolically. The components of $\boldsymbol{k}$ are then multiplied with a relaxation parameter, accounting for crosstalk between the central moments[6]. In Geier's PhD thesis [68], an elaborate derivation and corresponding expressions for the vector $\boldsymbol{k}$ can be found for the D2Q9 lattice. An overview of the equations necessary for the D3Q27 lattice is provided in [67].

Lycett-Brown and Luo [69] proposed an alternative, arguably more intuitive, derivation based on the same principles as Geier's work. First, the populations are written as a linear combination of several raw moments, which can be transformed straightforwardly to a set of central moments, as shown thoroughly by e.g. Coreixas et al. [60]. The contribution of each central moment to the populations is then individually relaxed.

To demonstrate the stability of the CLBM for higher Reynolds number flow, Geier [68] showed that the model is able to simulate the wake behind a rectangular plate, for a Reynolds number of 1.4×10^6 based on the height of the plate. They also show the capability to simulate the free decay of turbulence in a periodic domain, induced by initially having two layers of fluid moving in opposite directions. The obtained turbulent energy spectrum showed good agreement with the Kolmogorov spectrum.

3.5.4. Cumulant lattice Boltzmann method

So far, we have seen raw moments being used to construct the 'default' MRT, as detailed in Section 3.5.1, and central moments being used to construct the CLBM, as detailed in Section 3.5.3. Geier et al. [70] developed a lattice Boltzmann method based on relaxing cumulants instead of raw or central moments. Cumulants are the natural generated by using the natural logarithm of the moment generating function. In other words, the cumulant generating function is given by as

$$\mathscr{C}(\boldsymbol{X}) = \ln\left(\int e^{\boldsymbol{\xi}^\top \boldsymbol{X}} f(\boldsymbol{\xi})\, d\boldsymbol{\xi}\right) = \ln\left(\int e^{\boldsymbol{\xi}^\top \boldsymbol{X}} \sum_i f_i(\boldsymbol{\xi} - \boldsymbol{c}_i)\, d\boldsymbol{\xi}\right) \tag{3.107}$$

$$= \ln\left(\sum_i f_i e^{\boldsymbol{c}_i^\top \boldsymbol{X}}\right) d\boldsymbol{\xi}. \tag{3.108}$$

The cumulants are then computed by differentiating Equation (3.108) and evaluating the corresponding derivative at $\boldsymbol{X} = \boldsymbol{0}$, i.e., the tensor containing the ith order moments is given by

$$\boldsymbol{c}^{(i)} = \frac{\partial^i}{\partial \boldsymbol{x}^i} \mathscr{C}(\boldsymbol{X})\bigg|_{\boldsymbol{X}=\boldsymbol{0}}. \tag{3.109}$$

Note that contrary to the case of raw and central moments, the cumulant generating function $\mathscr{C}$ cannot be easily split into separate contributions by each population f_i, which makes it not possible to define a matrix $\boldsymbol{M}$ to immediately transform to cumulant space. Instead, Geier et al. propose to first use a transformation matrix $\boldsymbol{M}$ to move to raw moment space, and to then convert the raw moments into central moments, and subsequently convert those into cumulants, as the relations between these moment spaces is known[7], as can be found in the works of Geier et al. [70], Fard [71] and Coreixas et al. [60]. The cumulants can then be relaxed, and inversely transformed back to central moments and then back to raw moments, after which they can be converted back to populations.

[6]The components of $\boldsymbol{k}$ corresponding to lower order moments also appear in the components of $\boldsymbol{k}$ corresponding to higher order moments. As these components will already have been relaxed, they need not be relaxed during their contributions to the higher order moment relaxations. The fact that the relaxations are evaluated in a cascaded way, starting from the lower-order moments, is what the 'cascaded lattice Boltzmann method' borrows its name from.

[7]The conversion between raw and central moments is linear; the conversion to cumulants is non-linear, however.

4 Non-grid-conforming boundary methods

This chapter describes the immersed Boundary method introduced in Chapter 2 in more detail. Section 4.1 provides a brief overview of how the immersed Boundary method can be implemented in both a Navier-Stokes framework, whereas Section 4.2 describes its implementation in the lattice Boltzmann method.

4.1. Immersed boundary method

The immersed boundary method (IBM) was originally proposed by Peskin et al. [4] and has been widely applied in Navier-Stokes based CFD-applicaitons since [34, 72]. Its essence is that solid boundaries are not directly imposed on the solution through explicit boundary conditions and therefore the grid does not necessarily have to conform to the solid boundary. Instead, a force field is applied such that the resulting flow field matches the velocity of the solid boundary.

Mathematically, this can be stated as follows. The incompressible Navier-Stokes equations are given by

$$\nabla \cdot \boldsymbol{u} = 0 \tag{4.1}$$

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = \mu\nabla^2\boldsymbol{u} - \nabla p + \boldsymbol{f}. \tag{4.2}$$

The body force $\boldsymbol{f}$ may then be represented as a distributed force along the boundary of the immersed body, given by

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_{\Gamma} \boldsymbol{F}(\boldsymbol{s},t)\,\delta\left(\boldsymbol{x} - \boldsymbol{X}(\boldsymbol{s},t)\right) d\boldsymbol{s}, \tag{4.3}$$

in which $\boldsymbol{F}$ is the immersed boundary force and $\boldsymbol{X}(\boldsymbol{s},t)$ is the parametric surface representing the solid boundary. A wide variety of algorithms exists regarding how to determine $\boldsymbol{F}(\boldsymbol{s},t)$ and how to discretize its integral [4, 34].

One such example is the (multi)direct forcing scheme. The multidirect forcing scheme, proposed by Luo et al. [73], follows an iterative scheme. The boundary is represented by a Lagrangian mesh, see also Figure 4.1. Let $\boldsymbol{x}_i$ denote the nodes on the Eulerian mesh, and $\boldsymbol{X}_k$ the nodes on the Lagrangian mesh, having a velocity $\boldsymbol{U}_k$.

1. After completing all computations of the previous timestep, set $m = 0$.
2. Compute the initial velocity field $\boldsymbol{u}_i^{(m)}$ following from Equation (4.1)-(4.2) by ignoring the presence of the boundary forcing term given by Equation (4.3).
3. Diffuse the Eulerian velocity field $\boldsymbol{u}_i^{(m)}$ to the Lagrangian boundary nodes via

$$\boldsymbol{U}_\kappa\left(\boldsymbol{X}_\kappa\right)^{(m)} = \sum_i \boldsymbol{u}_i^{(m)}\left(\boldsymbol{x}_i\right) D\left(\boldsymbol{X}_\kappa - \boldsymbol{x}_i\right). \tag{4.4}$$

4. Compute the direct forcing on the Lagrangian nodes by evaluating

$$\boldsymbol{F}_\kappa^{(m)}\left(\boldsymbol{X}_\kappa\right) = \rho\frac{\boldsymbol{U}_\kappa^{(m)} - \boldsymbol{U}_\kappa}{\Delta t}. \tag{4.5}$$

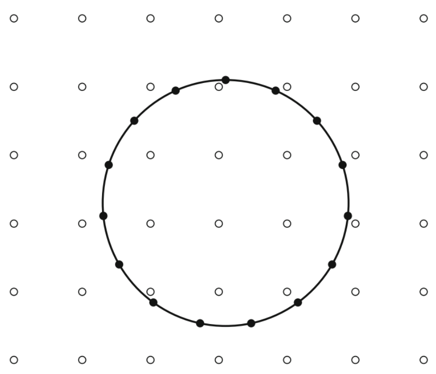


Figure 4.1: Sketch of a cylinder discretised by an Lagrangian mesh, embedded in an Eulerian mesh. Taken from Krüger et al. [2, p. 466].

5. Diffuse the Lagrangian forcing to the Eulerian nodes via

$$\boldsymbol{f}_i\left(\boldsymbol{x}_i\right)^{(m)} = \sum_{\kappa} \boldsymbol{F}_{\kappa}^{(m)}\left(\boldsymbol{X}_{\kappa}\right) D\left(\boldsymbol{X}_k - \boldsymbol{x}_i\right) \Delta V_{\kappa}, \tag{4.6}$$

   where ΔV_k is the volume of the Lagrangian body element.
6. Update the velocity at the Eulerian nodes via

$$\boldsymbol{u}_i^{(m+1)} = \boldsymbol{u}_i^{(m)} + \frac{\boldsymbol{f}_i^{(m)} \Delta t}{\rho}. \tag{4.7}$$

7. Increment m by 1, and go back to step 3 unless a specified termination-criterion is met.
8. The total Lagrangian force on each marker is then given by

$$\boldsymbol{F}_{\kappa}\left(\boldsymbol{X}_{\kappa}\right) = \sum_{m} \boldsymbol{F}_{\kappa}^{(m)}\left(\boldsymbol{X}_{\kappa}\right). \tag{4.8}$$

In the above approach, $D(\boldsymbol{r})$ represents the discrete delta-function, defined as

$$D\left(\boldsymbol{x}_i - \boldsymbol{X}_{\kappa}(t)\right) = \phi\left(x_{i,x} - X_{\kappa,x}\right) \phi\left(x_{i,y} - X_{\kappa,y}\right) \phi\left(x_{i,z} - X_{\kappa,z}\right). \tag{4.9}$$

Here, $\phi(r)$ is the one-dimensional discrete delta-function, which satisfies a number of properties:

1. $\phi(r)$ should be continuous.
2. There should be a value M such that $\phi(r) = 0$ for $|r| \geq M$.
3. $\phi(r)$ should satisfy

$$\sum_{j \text{ even}} \phi\left(r - j\right) = \sum_{j \text{ odd}} \phi\left(r - j\right) = \frac{1}{2}, \tag{4.10}$$

   where the first sum goes over all even integers, and the second sum over all odd integers. Note that this means that $\sum_j \phi(r - j) = 1$.
4. The first moment of $\phi(r)$ should be zero, i.e.

$$\sum_j \left(r - j\right) \phi\left(r - j\right). \tag{4.11}$$

5. There should be a value C, independent from r, such that

$$\sum_j \left(\phi\left(r - j\right)\right)^2 = C, \tag{4.12}$$

   for all real values of r.

Several functions satisfy these constraints, such as

$$\phi_2(r) = \begin{cases} 1-|r|, & 0 \le |r| \le 1 \\ 0, & |r| > 1 \end{cases} \tag{4.13}$$

$$\phi_3(r) = \begin{cases} \frac{1}{3}\left(1+\sqrt{1-3r^2}\right), & 0 \le |r| \le \frac{1}{2} \\ \frac{1}{6}\left(5-3|r|-\sqrt{-2+6|r|-3r^2}\right), & \frac{1}{2} \le |r| \le \frac{3}{2} \\ 0, & |r| > \frac{3}{2} \end{cases} \tag{4.14}$$

$$\phi_4(r) = \begin{cases} \frac{1}{8}\left(3-2|r|+\sqrt{1+4|r|-4r^2}\right), & 0 \le |r| \le 1 \\ \frac{1}{8}\left(5-2|r|-\sqrt{-7+12|r|-3r^2}\right), & 1 \le |r| \le 2 \\ 0, & |r| > 2, \end{cases} \tag{4.15}$$

where the subscript refers to the stencil width. A plot of these discrete delta functions is shown in Figure 4.2.

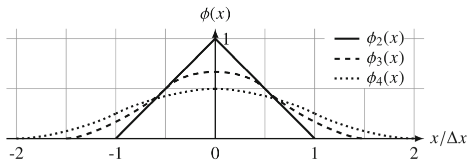


Figure 4.2: Plot of discrete delta functions ϕ_2, ϕ_3 and ϕ_4. Taken from Krüger et al. [2, p. 469].

Furthermore, in Equation (4.6), the Lagrangian force is diffused to the Eulerian grid by including multiplication with the volume of the body element, ΔV_κ.

The direct forcing scheme, originally proposed by Mohd-Yusof [74], follows the same approach as the multi-direct forcing scheme, but only uses one iteration.

4.2. Implementation of the IBM in the LBM

This multi-direct forcing method can be easily extended into the LBM [5]. In the LBM, after the populations from the previous timestep have been streamed, the IBM algorithm is performed, yielding an additional contribution to the sourcing term and affecting the equilibrium distribution (as discussed in Section 3.4.3), as shown in Figure 4.3. Note that all necessary fluid quantities to perform the IBM algorithm are macroscopic quantities, and implementation of the algorithm itself is therefore very similar to implementations in classical, finite volume based solvers.

Guo et al. [48] proposed a split-forcing LBM, in which a population moving from $\boldsymbol{x}_n$ to $\boldsymbol{x}_n + \boldsymbol{c}_i \Delta t$ is attributed half of the force at $\boldsymbol{x}_n$ and half of the force $\boldsymbol{x}_n + \boldsymbol{c}_i \Delta t$, as shown in Figure 4.4. This is in contrast to the 'default'-forcing, where the force experienced by that population would simply be only attributed the force at its initial position, i.e. the force at $\boldsymbol{x}_n$.

Kang and Hassan [35] demonstrated that this can be applied to the IBM by adjusting Equation (4.6) to

$$\boldsymbol{F}_\kappa^{(m)}(\boldsymbol{X}_\kappa) = 2\rho \frac{\boldsymbol{U}_\kappa^{(m)} - \boldsymbol{U}_\kappa}{\Delta t}, \tag{4.16}$$

and Equation (4.7) to

$$\boldsymbol{u}_i^{(m+1)} = \boldsymbol{u}_i^{(m)} + \frac{\boldsymbol{f}_i^{(m)} \Delta t}{2\rho}. \tag{4.17}$$

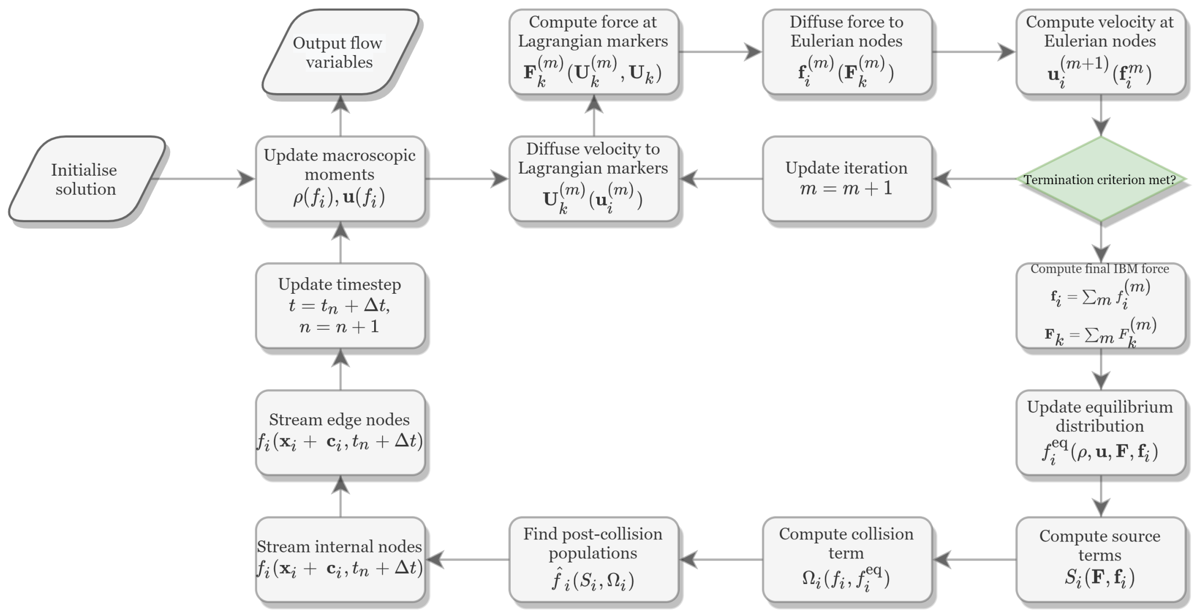

Initialise solution
Update macroscopic moments $\rho(f_i), \mathbf{u}(f_i)$
Output flow variables
Diffuse velocity to Lagrangian markers $\mathbf{U}_k^{(m)}(\mathbf{u}_i^{(m)})$
Compute force at Lagrangian markers $\mathbf{F}_k^{(m)}(\mathbf{U}_k^{(m)}, \mathbf{U}_k)$
Diffuse force to Eulerian nodes $\mathbf{f}_i^{(m)}(\mathbf{F}_k^{(m)})$
Compute velocity at Eulerian nodes $\mathbf{u}_i^{(m+1)}(\mathbf{f}_i^{m})$
Termination criterion met?
Update iteration $m = m + 1$
Compute final IBM force $\mathbf{f}_i = \sum_m f_i^{(m)}$ $\mathbf{F}_k = \sum_m F_k^{(m)}$
Update equilibrium distribution $f_i^{\text{eq}}(\rho, \mathbf{u}, \mathbf{F}, \mathbf{f}_i)$
Compute source terms $S_i(\mathbf{F}, \mathbf{f}_i)$
Compute collision term $\Omega_i(f_i, f_i^{\text{eq}})$
Find post-collision populations $\hat{f}_i(S_i, \Omega_i)$
Stream internal nodes $f_i(\mathbf{x}_i + \mathbf{c}_i, t_n + \Delta t)$
Stream edge nodes $f_i(\mathbf{x}_i + \mathbf{c}_i, t_n + \Delta t)$
Update timestep $t = t_n + \Delta t$, $n = n + 1$

Figure 4.3: Generic sequencing of operations in an IBM-LBM simulation.

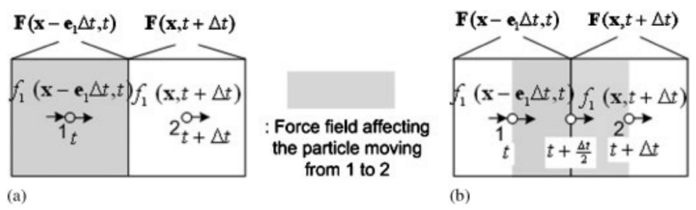

$\mathbf{F}(\mathbf{x} - \mathbf{e}_1\Delta t, t)$
$\mathbf{F}(\mathbf{x}, t + \Delta t)$
$f_1(\mathbf{x} - \mathbf{e}_1\Delta t, t)$
$f_1(\mathbf{x}, t + \Delta t)$
1
t
2
$t + \Delta t$
$t + \frac{\Delta t}{2}$
: Force field affecting the particle moving from 1 to 2
(a)
(b)

Figure 4.4: Difference between default-forcing (a) and split-forcing (b) with regards to how the force acting on a population at $\boldsymbol{x}_n$, travelling in the direction $\boldsymbol{c}_i$, is computed. Taken from [35, p. 1138].

II

Implementation of the LaBIB-FSI solver

In Chapter 5-6, the main scientific contributions by the author are discussed. Chapter 5 discusses the proposed implementation of an immersed interface method into the lattice Boltzmann method, including a discussion of its general characteristics and a comparison to the application of the immersed interface method by Qin et al. [7]. Chapter 6 discusses the development of the custom fluid-structure interaction solver; a lattice Boltzmann method solver using the immersed boundary and similar methods, and coupled to the `pyfe3d`-solver [13]. This solver has been dubbed LaBIB-FSI, shorthand for Lattice Boltzmann Immersed Boundary Fluid Structure Interaction Solver. LaBIB-FSI is written in `C++`, using `OpenMP` to parallellise appropriate parts of the code and interfacing with the `Python`-based `pyfe3d`-solver.

5 Proposed Immersed Interface Method into the lattice Boltzmann framework

In this chapter, a implementation of the immersed interface method into the lattice Boltzmann framework is proposed. In Section 5.1, the derivation of the jump condition in the lattice Boltzmann equation discretised in velocity space, but continuous in physical space and time, is shown. In Section 5.2, the implementation of the jump condition into the lattice Boltzmann method is discussed. Section 5.3 describes a midpoint immersed interface method, and explains the differences between the immersed boundary and immersed interface method from a more practical point of view. In Section 5.4 the extension of the immersed interface method to moving bodies is discussed. Section 5.5 details some additional considerations about the immersed interface method.

Section 5.6 reflects on scientific contribution of the immersed interface method proposed here, and puts it into perspective with related developments in the lattice Boltzmann method. Section 5.7 provides an in-depth comparison to the work of Qin et al. [7], who previously attempted to implement the immersed interface method into the lattice Boltzmann method, but appear to have made flaws in their derivation. In Section 5.8 three minor variations to the immersed interface method are discussed. Finally, Section 5.9 concludes the Chapter.

5.1. Proposal of an immersed interface method

The immersed interface method (IIM) was originally developed by Leveque and Li [6] for elliptic differential equations, later applied to the Navier-Stokes equations by Lee et al. [36] and extended to moving boundaries by Xu and Wang [37]. Immersed interface methods are characterised by including singular attributes, such as a singular force field arising from an immersed boundary, as jumps in the solution space. An example of an application of the IIM to a simple, one-dimensional elliptic differential equation is provided in Appendix A.

5.1.1. Derivation of the jump condition for populations across a stationary boundary

The IIM can be applied to the LBM in the following way. Consider again Equation (3.28),

$$\frac{df_i}{d\eta} = \Omega_i + F_i, \tag{5.1}$$

which is the Boltzmann equation discretized in velocity space, transformed using the method of characteristics to an ODE that runs along the direction of the corresponding velocity link $\boldsymbol{c}_i$. Let this lattice link be intersected at $\eta = \eta_c$, and let us assume that the boundary is stationary[1]. Furthermore, for sake of simplicity, let the forcing term be discretized to first order, i.e. (compare with Equation (3.26))

$$F_i = \frac{\partial f_i}{\partial \boldsymbol{\xi}_i}\frac{F}{\rho} \approx \frac{w_i}{c_s^2}\boldsymbol{c}_i \cdot \boldsymbol{f}. \tag{5.2}$$

[1]The extension of the IIM to moving boundaries will be detailed in Section 5.5.

In addition, similar to an IBM, let a solid boundary be represented by a singular boundary force acting along the surface, i.e.

$$\boldsymbol{f} = \int_{\Gamma} \boldsymbol{F}(\boldsymbol{s})\,\delta\left(\boldsymbol{x} - \boldsymbol{X}(\boldsymbol{s})\right) d\boldsymbol{s}. \tag{5.3}$$

Finally, let f_i be allowed to be discontinuous across $\eta = \eta_c$, i.e., at $\eta = \eta_c$, f_i may experience a jump denoted by

$$[\![f_i]\!]\left(\eta_c\right) \equiv \lim_{\delta\to 0} f_i\left(\eta_c + \delta\right) - f_i\left(\eta_c - \delta\right). \tag{5.4}$$

We may then integrate Equation (5.2) between $\eta = \eta_c - \epsilon$ and $\eta = \eta_c + \epsilon$. Doing so yields

$$\int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \frac{df_i}{d\eta} d\eta = \int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \Omega_i d\eta + \int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \frac{w_i}{c_s^2} \boldsymbol{c}_i \cdot \int_{\Gamma} \boldsymbol{F}(\boldsymbol{s})\,\delta\left(\boldsymbol{x} - \boldsymbol{X}(\boldsymbol{s})\right) ds d\eta. \tag{5.5}$$

When $\epsilon \to 0$, each of these terms takes the following values:

- The first term becomes

$$\int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \frac{df_i}{d\eta} d\eta = f_i\Big|_{\eta_c-\epsilon}^{\eta_c+\epsilon} = [\![f_i]\!]\left(\eta_c\right). \tag{5.6}$$

- The second term reduces to 0 as Ω_i is bounded on $[\eta - \epsilon, \eta + \epsilon]$.
- The third term becomes

$$\int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \frac{w_i}{c_s^2} \boldsymbol{c}_i \cdot \int_{\Gamma} \boldsymbol{F}(\boldsymbol{x}, t)\,\delta\left(\boldsymbol{x} - \boldsymbol{X}\right) ds d\eta = \frac{w_i}{c_s^2} \boldsymbol{c}_i \cdot \boldsymbol{F}\left(\boldsymbol{x}_c\right), \tag{5.7}$$

  where $\boldsymbol{x}_c$ is the point along $\boldsymbol{c}_i$ where the surface boundary is intersected.

Thus, we obtain

$$[\![f_i]\!]\left(\eta_c\right) = [\![f_i]\!]\left(\boldsymbol{x}_c, t_c\right) = \frac{w_i}{c_s^2} \boldsymbol{c}_i \cdot \boldsymbol{F}\left(\boldsymbol{x}_c\right). \tag{5.8}$$

In other words, when a lattice link $\boldsymbol{c}_i$ intersects a solid boundary at the point $\boldsymbol{x} = \boldsymbol{x}_c$ at $t = t_c$, then the population f_i must be incremented by the value computed from Equation (5.8).

Equation (5.8) is significant in the sense that it allows for imposing the boundary force during streaming rather than during collision; in a collision-based IBM, the forcing is applied at the beginning of a timestep ($\eta = 0$), but in streaming-based formulation, the forcing is applied during the timestep. Furthermore, in the above derivation, no discretization error arises as the singular force is treated by an exact integral.

It should be noted that higher-order discretizations of the forcing term (Equation (5.2)) can naturally also be used; the above derivation is easily adapted to account for e.g. Guo forcing.

5.2. Implementation of the immersed interface method in the lattice Botlzmann method

It should be noted that although Equation (5.1) is discretised in velocity space, it has not yet been discretised in velocity and temporal space, which is necessary to be able to implement the jumps during the streaming step of the LBM. This section therefore proposes an implementation of the result of Equation (5.8) into the LBM. Section 5.2.1 describes the proposed algorithm. Section 5.2.2 provides additional detail on some of the proposed steps.

5.2.1. Description of the algorithm

First, let $\boldsymbol{x}_n$ denote the nodes on the Eulerian mesh, and $\boldsymbol{X}_\kappa$ the nodes on the Lagrangian mesh, having a velocity $\boldsymbol{U}_\kappa$. The IIM-LBM can then be represented as follows:

1. After completing the collision operation, set $n = 0$ and all $[\![f_i]\!]^{(0)} = 0$.
2. Diffuse the post-collision populations at their post-streaming locations to the Lagrangian boundary nodes via

$$\hat{f}_i(\boldsymbol{X}_\kappa) = \sum_n \left[\hat{f}_i(\boldsymbol{x}_n, t) + [\![f_i]\!]^{(n)}(\boldsymbol{x}_n, t)\right] D\left(\boldsymbol{X}_\kappa - (\boldsymbol{x}_n + \boldsymbol{c}_i)\right), \tag{5.9}$$

   where $[\![f_i]\!]^{(n)}(\boldsymbol{x}_n, t)$ represents the jump experienced by population f_i located at node $\boldsymbol{x}_n$ at time t in the upcoming streaming step.
3. Compute the Eulerian velocity at each Lagrangian boundary node via

$$\rho \boldsymbol{U}_\kappa^{(n)} = \sum_i \hat{f}_i(\boldsymbol{X}_\kappa). \tag{5.10}$$

4. Compute the direct forcing on the Lagrangian nodes by evaluating

$$\boldsymbol{F}_\kappa^{(n)}(\boldsymbol{X}_\kappa) = \rho \frac{\boldsymbol{U}_\kappa^{(n)} - \boldsymbol{U}_\kappa}{\Delta t}. \tag{5.11}$$

5. Compute the population jump induced at each Lagrangian node via

$$[\![f_i]\!]^{(n)}(\boldsymbol{X}_\kappa) = [\![\hat{f}_i]\!](\boldsymbol{x}_c, t_c) = \frac{w_i}{c_s^2}\boldsymbol{c}_i \cdot \boldsymbol{F}_\kappa^{(n)}. \tag{5.12}$$

6. Diffuse the population jumps to the Eulerian nodes via

$$[\![f_i^{(n)}]\!](\boldsymbol{x}_n, t) = \sum_\kappa [\![f_i]\!](\boldsymbol{X}_\kappa) D\left(\boldsymbol{X}_\kappa - (\boldsymbol{x}_n + \boldsymbol{c}_i \Delta t)\right) \Delta V_\kappa, \tag{5.13}$$

   where V_κ is the volume of the Lagrangian body element.
7. Increment n by 1, and go back to step 2 unless a specified termination-criterion is met.
8. The total Lagrangian force on each marker and jump experienced by each population are then given by

$$\boldsymbol{F}_\kappa(\boldsymbol{X}_\kappa) = \sum_n \boldsymbol{F}_\kappa^{(n)}(\boldsymbol{X}_\kappa) \tag{5.14}$$

$$[\![f_i]\!](\boldsymbol{x}_n, t) = \sum_n f_i^{(n)}(\boldsymbol{x}_n, t), \tag{5.15}$$

   and the streaming operation is performed via

$$f_i(\boldsymbol{x}_n + \boldsymbol{c}_i \Delta t, t + \Delta t) = \hat{f}_i(\boldsymbol{x}_n, t) + [\![f_i]\!](\boldsymbol{x}_n, t). \tag{5.16}$$

5.2.2. Elaboration

A number of components of the above described algorithm merit additional attention. First, in the IBM described in Chapter 3, the macroscopic flow quantities are directly interpolated to estimate the velocity at the Lagrangian markers. In contrast, the IIM proposed described here interpolates the populations directly, as shown by Equation (5.9). Similarly, in Equation (5.13), the effect of the boundary forcing is diffused to the fluid directly at a mesoscopic level, rather than via the forcing.

As a consequence of this, more control exist over how the mesoscopic quantities are incorporated into the solver. In particular, in Equation (5.9) and (5.13), the distance used for the discrete Delta-functions is given by $\boldsymbol{X}_\kappa - (\boldsymbol{x}_n + \boldsymbol{c}_i \Delta t)$. This corresponds to taking the populations at their post-streaming locations. This is in contrast to the IBM described in Chapter 3, where the macroscopic velocity is interpolated in Equation (4.4) and the macroscopic force is diffused in Equation (4.6).

Additionally, it should be noted that although the above algorithm is iterative in nature, it can be easily modified into an implicit algorithm that computes the boundary forcing directly. The procedure for this can be described as follows:

1. At each Lagrangian marker, apply a unit force in each spatial direction.

2. For each Lagrangian marker and spatial direction, compute the jump induced by the unit force via

$$\delta[\![f_{i,\kappa,\alpha}]\!]^{(n)}(\boldsymbol{X}_\kappa) = [\![\hat{f}_i]\!](\boldsymbol{x}_c, t_c) = \frac{w_i}{c_s^2}\boldsymbol{c}_i \cdot \tilde{\boldsymbol{F}}^{(n)}_{\kappa,\alpha}, \tag{5.17}$$

   where $\tilde{\boldsymbol{F}}_{\kappa,\alpha}$ represents the unit force vector in the α-direction of the physical space.
3. Diffuse the jumps induced by the unit forces to the Eulerian grid via

$$\delta[\![f^{(n)}_{i,\kappa,\alpha}]\!](\boldsymbol{x}_n, t) = \sum_\kappa \delta[\![f_{i,\kappa,\alpha}]\!](\boldsymbol{X}_\kappa)\, D\left(\boldsymbol{X}_\kappa - (\boldsymbol{x}_n + \boldsymbol{c}_i\Delta t)\right)\Delta V_\kappa. \tag{5.18}$$

4. Diffuse these jumps back to the Lagrangian nodes via

$$\delta[\![\hat{f}_{i,\kappa,\bar{\kappa},\alpha}]\!](\boldsymbol{X}_\kappa) = \sum_{\bar{\kappa}}\sum_n \delta[\![f_{i,\bar{\kappa},\alpha}]\!]^{(n)}(\boldsymbol{x}_n, t)\, D\left(\boldsymbol{X}_\kappa - (\boldsymbol{x}_n + \boldsymbol{c}_i)\right), \tag{5.19}$$

   where $\delta[\![\hat{f}_{i,\kappa,\bar{\kappa},\alpha}]\!](\boldsymbol{X}_\kappa)$ equals the interpolated change in the jump of population i at Lagrangian marker $\boldsymbol{X}_\kappa$ due to the unit vector in the α-direction applied at Lagrangian marker $\boldsymbol{X}_{\bar{\kappa}}$.
5. Compute the change in velocity due to these jumps at the Lagrangian nodes via

$$\delta\boldsymbol{U}_{\kappa,\bar{\kappa},\alpha} = \sum_i \delta[\![\hat{f}_{i,\kappa,\bar{\kappa},\alpha}]\!](\boldsymbol{X}_\kappa). \tag{5.20}$$

6. Set up the sensitivity matrix S containing the effect of a unit forcing applied at Lagrangian marker $\boldsymbol{X}_{\bar{\kappa}}$ on the velocity at Lagrangian marker $\boldsymbol{X}_\kappa$ by setting

$$S_{d\kappa+\alpha, d\bar{\kappa}+\beta} = \delta\left(\boldsymbol{U}_{\kappa,\bar{\kappa},\alpha}\right)_\beta, \tag{5.21}$$

   where d is the number of spatial dimensions (such that $\alpha = 0, \ldots, d-1$), and $\delta\left(\boldsymbol{U}_{\kappa,\bar{\kappa},\alpha}\right)_\beta$ is the component of $\delta\boldsymbol{U}_{\kappa,\bar{\kappa},\alpha}$ in the β-direction.
7. Diffuse the post-collision populations to the Lagrangian markers via

$$\hat{f}_i(\boldsymbol{X}_\kappa) = \sum_n \hat{f}_i(\boldsymbol{x}_n, t)\, D\left(\boldsymbol{X}_\kappa - (\boldsymbol{x}_n + \boldsymbol{c}_i)\right). \tag{5.22}$$

8. Compute the Eulerian velocity at each Lagrangian boundary node via

$$\rho\boldsymbol{U}^{(n)}_\kappa = \sum_i \hat{f}_i(\boldsymbol{X}_\kappa). \tag{5.23}$$

9. Set up the right-hand-side matrix $\boldsymbol{b}$ by setting

$$\boldsymbol{b}_{d\kappa+\alpha} = \left(\rho\boldsymbol{U}^{(n)}_\kappa\right)_\alpha. \tag{5.24}$$

10. Solve the system

$$S^\top\mathscr{F} + \boldsymbol{b} = \boldsymbol{0}, \tag{5.25}$$

   where

$$\boldsymbol{F}_{\kappa,\alpha} = \mathscr{F}_{d\kappa+\alpha}. \tag{5.26}$$

11. Compute the jumps via

$$\delta[\![f_{i,\kappa,\alpha}]\!]^{(n)}(\boldsymbol{X}_\kappa) = [\![\hat{f}_i]\!](\boldsymbol{x}_c, t_c) = \frac{w_i}{c_s^2}\boldsymbol{c}_i \cdot \boldsymbol{F}^{(n)}_{\kappa,\alpha}. \tag{5.27}$$

The above formulation will compute the population jumps in a single iteration, but requires solving a system of equations. However, it should be noted that this system of equations may be relatively sparse (since most Lagrangian markers will not have their diffusive zones overlapping). Furthermore, the number of nonzero entries will generally scale linearly with the fluid grid refinement (assuming the number of Lagrangian markers is increased proportionally to the fluid grid refinement, so a specialised solver may be able to solve this efficiently.

5.3. Midpoint immersed interface method

As described in Section 5.2.2, the base algorithm interpolates and diffuses the populations at their end-streaming position, and the IIM provides more control over how the mesoscopic quantities are incorporated into the solver. One application of this could be to adjust the point in time in which the populations are streamed. For example, instead of taking the end-streaming position of each population, one can also take the mid-streaming position, by altering Equations (5.9) and (5.13) to

$$\hat{f}_i\left(\boldsymbol{X}_\kappa\right) = \sum_n \left[\hat{f}_i\left(\boldsymbol{x}_n, t\right) + [\![f_i]\!]^{(n)}\left(\boldsymbol{x}_n, t\right)\right] D\left(\boldsymbol{X}_k - \left(\boldsymbol{x}_n + \frac{1}{2}\boldsymbol{c}_i\right)\right), \tag{5.28}$$

and

$$[\![f_i^{(n)}]\!]\left(\boldsymbol{x}_n, t\right) = \sum_\kappa [\![f_i]\!]\left(\boldsymbol{X}_\kappa\right) D\left(\boldsymbol{X}_\kappa - \left(\boldsymbol{x}_n + \frac{1}{2}\boldsymbol{c}_i \Delta t\right)\right) \Delta V_\kappa, \tag{5.29}$$

without modifying the remainder of the algorithm. This modification is dubbed the midpoint immersed interface method (MIIM). Note that taking the mid-streaming positions of the populations is simply impossible in a conventional LBM scheme, as there the boundary treatment is forced to always execute at the beginning of the time step. Thus, clearly the IIM allows for more flexibility in how the boundary treatment is executed.

To illustrate the IBM, IIM and MIIM, and to elucidate the differences between them, Figures 5.1-5.3 show the interpolation stencils for the IBM, IIM and MIIM applied to a D1q3-lattice, with a Lagrangian marker exactly in the middle between two lattice nodes. A simple discrete delta function is used, namely

$$\phi(r) = \begin{cases} 1 - |r|, & |r| < 1 \\ 0, & \text{otherwise.} \end{cases} \tag{5.30}$$

In the IBM, the velocity at $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is interpolated based on the distance of these lattice nodes to the marker (with the weight dictated by the green triangle). This effectively comes down to interpolating the populations at $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ directly.

Meanwhile for the IIM, the Lagrangian marker is shifted for the evaluation of each population. In order to perform an interpolation for f_1, the marker is moved 'back' one unit $\boldsymbol{c}_1$, i.e. it is moved one lattice unit to the left. An interpolation zone is then constructed around this fictitious marker (denoted by the blue triangle), and the populations f_1 at $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are interpolated. Similarly, for the interpolation of f_2, one creates a fictitious node by moving one unit $\boldsymbol{c}_2$ 'back'. f_2 at $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ are then interpolated, as indicated by the red triangle.

With these interpolated values of f_1 and f_2, the velocity at the original marker point is constructed, and the feedback force may be obtained. This is then converted to a jump via Equation (4.8), which is then diffused back to the grid in an identical manner as to how the populations were interpolated.

The shift 'back' to create a fictitious marker at which the populations are interpolated serves a clear purpose - the populations f_1 at $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ will move to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ by the end of the time-step (and will then be the ones closes to the Lagrangian marker), whereas the populations f_2 at $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ will move to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively (and thus also closest to the Lagrangian marker). Meanwhile, the IBM interpolates populations that, after streaming, will be located farther away from the marker point, and thus it appears more intuitive to shift the interpolation point as done during the IIM.

Finally, in the MIIM, the fictitious markers are only shifted half a lattice link. For f_1, this means that the fictitious marker coincides with $\boldsymbol{x}_1$, and for f_2, this means that the fictitious marker coincides with $\boldsymbol{x}_2$. The interpolation is self-evident in this case, and the jump can be computed through Equation (4.8) and be diffused back to same populations. By shifting by only half the lattice link instead of the full lattice link, we essentially interpolate the populations that are closest to the Lagrangian marker midway through the streaming process - f_1 will have travelled halfway from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$ and coincide with the Lagrangian marker, and f_2 will have travelled halfway from $\boldsymbol{x}_2$ to $\boldsymbol{x}_1$, coinciding in the same fashion. Thus, it makes sense to give the largest interpolation weight to those populations.

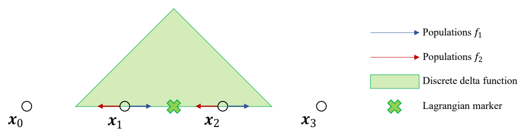


Figure 5.1: Interpolation zone for a Lagrangian marker in the IBM in a one-dimensional stencil.

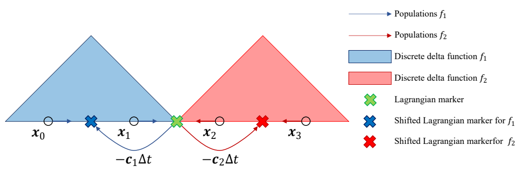

x_0
x_1
x_2
x_3
$-c_1\Delta t$
$-c_2\Delta t$
Populations f_1
Populations f_2
Discrete delta function f_1
Discrete delta function f_2
Lagrangian marker
Shifted Lagrangian marker for f_1
Shifted Lagrangian markerfor f_2

Figure 5.2: Interpolation zone for a Lagrangian marker in the IIM in a one-dimensional stencil.

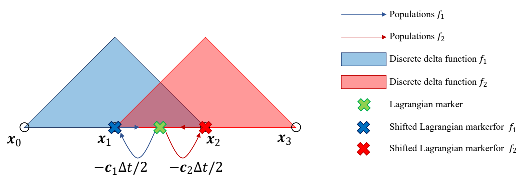

x_0
x_1
x_2
x_3
$-c_1\Delta t/2$
$-c_2\Delta t/2$
Populations f_1
Populations f_2
Discrete delta function f_1
Discrete delta function f_2
Lagrangian marker
Shifted Lagrangian markerfor f_1
Shifted Lagrangian markerfor f_2

Figure 5.3: Interpolation zone for a Lagrangian marker in the MIIM in a one-dimensional stencil.

5.4. Treatment of moving boundaries

Treatment of moving boundaries is straightforward in the IIM. In the base configuration of the algorithm described in Section 5.2.1, the velocity of the boundary is directly accounted for in Equation (5.11), and thus the implementation of a moving boundary is simple.

There does exist some nuance with regards to how the boundary velocity is evaluated. That is, should one take $\boldsymbol{U}_k(\boldsymbol{X}_k, t)$, $\boldsymbol{U}_k(\boldsymbol{X}_k, t+\Delta t)$, or some value in between? Since the immersed interface method integrates over the

Secondly, the forcing term in Equation (5.2) was discretised to first order, neglecting the dependence on the local velocity. For stationary boundaries, this is accurate up until and including second order terms, but for a moving boundary, one may increase the order of the discretisation by using

$$F_i = \frac{\partial f_i}{\partial \boldsymbol{\xi}_i}\frac{F}{\rho} \approx w_i \left(\frac{\boldsymbol{c}_i}{c_s^2} + \frac{\left(\boldsymbol{c}_i \boldsymbol{c}_i^\top - c_s^2 \boldsymbol{I}\right)\boldsymbol{u}}{c_s^4} \right)^\top \boldsymbol{f}, \tag{3.26}$$

instead of Equation (5.2). This accounts for the effect of the velocity on the transformation of the boundary force from physical space to lattice space.

5.5. Additional considerations

Two important considerations still remain before the immersed interface method outlined above fits within the LBM on a grander scale. Specifically, Section 5.5.1 briefly discusses how the IIM can be formulated in an implicit form, and Section 5.5.2 evaluates the computational cost of the IIM compared to the IBM.

5.5.1. Implicit formulation

The implicit formulation of the default IIM, as shown in Section 5.2.2 can be applied to the MIIM described in Section 5.3, as well as to alternative formulations of the IIM that are described in Section 5.8 in analogous fashion. These implicit formulations were added to the Python-based solver, but not included in LaBIB-FSI, as the multi-direct forcing scheme did not require many iterations to converge, as shown in Section 7.2.3.

5.5.2. Computational cost

The computational effort for both the IBM and the IIM scales similarly. If a certain 3D domain contains N^3 lattice nodes, then a surface boundary will consist of $\mathcal{O}\left(N^2\right)$ Lagrangian markers. If an iterative scheme is employed, then the computational effort can be expected to scale proportionally to the number of Lagrangian markers:

- First, one loops through all of the Lagrangian markers to compute the direct forcing on each marker. The effort required for this for a single marker is independent of grid size, as the width of the diffusive zone is constant in lattice units, i.e. the number of fluid nodes from which being interpolated for a single Lagrangian marker is constant and independent of grid refinement.

- Secondly, one loops through all of the Lagrangian markers to diffuse the feedback force back to each marker (in case of the IBM) or jumps back to each link (in case of the IIM). Again, the effort required for this for a single marker is independent of grid size.

Thus, each iteration has a computational effort that scales with $\mathcal{O}\left(N^2\right)$. Assuming the number of iterations to achieve the same iteration tolerance scales with $\mathcal{O}(N)$, the total computational effort will therefore scale with $\mathcal{O}\left(N^3\right)$, for both the IBM and IIM.

Nonetheless, although no attempt is made to quantify the exact difference, the IBM is likely to require less computational effort than the IIM. This is because the IIM requires information on a mesoscopic level during the algorithm (the populations at each lattice node), whereas the IBM only uses macroscopic information (the velocity and density at each lattice node). For a D2q9 mesh, this for example means that the IIM has to interpolate/diffuse 9/2 times as many numerical values from and to the fluid grid.

Obviously, the IBM/IIM contain more mathematical operations than purely the diffusion and interpolation, but this should give a good first-order approximation of the computational effort required for both of them. Nonetheless, it should be kept in mind that the effect this has on the total wall time for a specific simulation greatly varies, depending on the size of the boundary vs. the size of the fluid grid, the number of iterations executed in the multi-direct forcing scheme, implementation optimisations, etc. Thus, the total simulation time will not increase by a factor of 9/2 for a D2q9 mesh[2].

It should be noted that the output of the IIM algorithm is a collection of jumps that may be directly imposed during the streaming phase. On the other hand, the IBM algorithm results in a force field on the lattice nodes that needs to be accounted for during stream, which requires may result in additional operations, for example when using MRT operators. However, it is deemed unlikely that this would offset the cost imposed by performing the interpolation/diffusion directly in population space, rather than in velocity space.

5.6. Reflection on the immersed interface method

With the application of the IIM in the LBM derived in Section 5.2, and a derivative of it derived in Section 5.3, it is worthwhile to evaluate in detail how the IIM differs from the LBM, and how it fits within boundary treatments in the LBM from a grander point of view.

5.6.1. Comparison with the immersed boundary method

In Section 5.3, the difference between the IBM and IIM is explained, and it is shown that from a practical point of view, the IIM simply creates a fictitious marker for each population space that is shifted in the direction opposite to the corresponding lattice speed vector. However, an observant reader would notice that this actually does not result in a concrete benefit to the IIM over the IBM. After all, the interpolated populations arrive at the same grid nodes that the IBM would use at the very beginning of the next time-step. Thus, the IBM would use the exact same populations at time-step $n_t + 1$ that the IIM used at time-step n_t. It is hard to argue this provides a legitimate benefit, as it appears to result in nothing more than the IIM preceding the IBM by a single time-step. This is also confirmed when both methodologies are implemented.

Nonetheless, the IIM offers a fundamental benefit that the conventional IBM does not provide, namely the freedom in time-integration that the IIM provides. This was already shown to allow for the development of the MIIM in Section 5.3, which executes its algorithm at the mid-point in streaming. This is an important result, and to the best of the author's knowledge, does not appear to be implemented previously in the LBM.

5.6.2. Contribution to research gap

It should be noted that within the LBM, both the idea of having more precise control over how the boundary treatment is executed in temporal space, and the idea of interpolating populations (instead of macroscopic velocities) are not unique, and has been attempted by other authors.

For example, Zhou and Fan [75] implemented a Runge-Kutta scheme to the IB-LBM, with the velocity and

[2]Although not thoroughly measured, for the validation cases described in Chapter 7 and 8, the MIIM was experienced to be usually around 10%-50% slower than the IBM (usually closer to 10%). However, it should be stressed that this was not exhaustively tracked, and no significant attempt was made at optimising either implementation.

density at intermediate time-steps being extrapolated from previous intermediate time-steps, via

$$\boldsymbol{u}^{(k+\alpha)} = \boldsymbol{u}^{(k)} + \left(\mu\nabla^2\boldsymbol{u}^{(k)} - \nabla p^{(k)} - \rho^{(k)}\boldsymbol{u}^{(k)}\cdot\nabla\boldsymbol{u}^{(k)} + \tilde{\boldsymbol{f}}\right)\alpha\Delta t/\rho^{(k)} \tag{5.31}$$

$$\rho^{(k+\alpha)} = \rho^{(k)} - \nabla\cdot\left(\rho^{(k)}\boldsymbol{u}^{(k+\alpha)}\right)\alpha\Delta t, \tag{5.32}$$

where $\tilde{\boldsymbol{f}}$ is some weighted force term obtained from force terms generated in previous intermediate steps (for sake of brevity, this expression is not included here. This imposes a significant computational effort on the simulation, as it involves approximating several gradients and Laplacians. On the other hand, the MIIM shown above results in a procedure whose complexity is very similar to the classical IBM, and does not involve evaluating derivatives of any kind. Thus, the framework delivered by the IIM appears far more flexible with regards to time integration than the method proposed by Zhou and Fan.

Tao et al. [76] proposed an IBM where the populations and their non-equilibrium parts are interpolated in the region around the boundary marker. A equilibrium population is calculated based on the desired density and velocity at the boundary and is compared against the interpolated equilibrium population (computed from the population and its non-equilibrium part). The difference is diffused back to the lattice grid. This results in an increased of order of accuracy when evaluating a cylindrical Couette flow, increasing the order of accuracy from one to two, compared to the original IBM. However, their population-based IBM still occurs at the very beginning of the time-step, and does not attempt to utilise any mid-streaming information. Still, their approach appears interesting due to the population-based nature (similar to the IIM) and promising in terms of accuracy, and combining the method of Tao et al. with the MIIM would be an interesting avenue to explore in the future.

Finally, the Qin et al. [7] already attempted to implement the immersed interface method in the LBM. Their approach, albeit different, comes closest to the methodology outlined previously. As such, it deserves further attention, and Section 5.7 goes into depth about the differences and similarities between the approaches.

5.7. Immersed interface method proposed by Qin et al.

It should be noted that Qin et al. [7] recently also proposed an implementation of the immersed interface method into the LBM framework. However, as will be explained in the hereafter, two major flaws appear to be present in their derivation, as discussed in Section 5.7.2 and 5.7.3. The first of those is easy to correct, but the second is more fundamental and significant. Furthermore, even if those flaws are accounted and corrected for, their implementation appears inferior to the one proposed in Section 5.2, for reasons outlined in Section 5.7.4.

5.7.1. Derivation proposed by Qin et al.

First, the derivation by Qin et al. [7] is provided as presented by them, without commentary on their flaws. They initiate their derivation from the governing equations

$$\frac{\partial f_i}{\partial t} + \boldsymbol{c}_i\cdot\nabla f_i = \Omega_i + \frac{1}{c_s^2}w_i\boldsymbol{c}_i\cdot\boldsymbol{F} \tag{5.33}$$

$$\boldsymbol{F} = \int_\Gamma \boldsymbol{G}(\boldsymbol{s},t)\,\delta(\boldsymbol{x}-\boldsymbol{X})\,d\boldsymbol{s}. \tag{5.34}$$

Multiplying with a test function ϕ and integrating over $\Omega_{\epsilon,t}$ shown in Figure 5.4[3] gives

$$\int_{\Omega_{\epsilon,t}} \frac{\partial f_i}{\partial t}\phi d\boldsymbol{x} + \int_{\Omega_{\epsilon,t}} \left(\boldsymbol{c}_i\cdot\nabla f_i\right)\phi d\boldsymbol{x} = \int_{\Omega_{\epsilon,t}} \Omega_i\phi d\boldsymbol{x} + \frac{1}{c_s^2}w_i\boldsymbol{c}_i\cdot\int_{\Omega_{\epsilon,t}}\left(\int_\Gamma \boldsymbol{G}(\boldsymbol{s},t)\,\delta(\boldsymbol{x}-\boldsymbol{X}(\boldsymbol{s},t))\,d\boldsymbol{s}\right)\phi d\boldsymbol{x}, \tag{5.35}$$

where $\Omega_{\epsilon,t}$ is the domain bounding the boundary, as depicted in Figure 5.4. The first and third term both reduce to 0 as their integrands are bounded on this domain. The second term can be integrated by parts

[3]It should be noted that Qin et al. initially define the integration domain as fully enclosing the boundary Γ. However, this domain may actually be arbitrarily truncated along the boundary, and does not necessarily need to enclose a closed boundary.

as

$$\int\limits_{\Omega_{\epsilon,t}} (\boldsymbol{c}_i \cdot \nabla f_i)\,\phi d\boldsymbol{x} = \boldsymbol{c}_i \cdot \left(\int\limits_{\Gamma^+_{\epsilon,t}} f_i \boldsymbol{n} \phi da + \int\limits_{\Gamma^-_{\epsilon,t}} f_i (-\boldsymbol{n})\,\phi da - \int\limits_{\Omega_{\epsilon,t}} f_i \nabla \phi d\boldsymbol{x} \right) \tag{5.36}$$

$$= \boldsymbol{c}_i \cdot \int\limits_{\Gamma} [\![f_i]\!] \boldsymbol{n} \phi \left| \frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s} \right| d\boldsymbol{s}, \tag{5.37}$$

and the fourth term simply reduces to

$$\frac{1}{c_s^2} w_i \boldsymbol{c}_i \cdot \int\limits_{\Omega_{\epsilon,t}} \left(\int\limits_{\Gamma} \boldsymbol{G}(\boldsymbol{s},t)\,\delta(\boldsymbol{x} - \boldsymbol{X}(\boldsymbol{s},t))\,d\boldsymbol{s} \right) \phi d\boldsymbol{x} = \frac{1}{c_s^2} w_i \boldsymbol{c}_i \cdot \int\limits_{\Gamma} \boldsymbol{G}(\boldsymbol{s},t)\,d\boldsymbol{s}\phi. \tag{5.38}$$

Resulting in

$$\boldsymbol{c}_i \cdot \int\limits_{\Gamma} [\![f_i]\!] \boldsymbol{n} \phi \left| \frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s} \right| d\boldsymbol{s} = \frac{1}{c_s^2} w_i \boldsymbol{c}_i \cdot \int\limits_{\Gamma} \boldsymbol{G}(\boldsymbol{s},t)\,d\boldsymbol{s}\phi \tag{5.39}$$

$$[\![f_i]\!] = \frac{1}{\left| \frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s} \right| c_s^2} w_i \boldsymbol{n} \cdot \boldsymbol{G}(\boldsymbol{s},t)\,. \tag{5.40}$$

This concludes the derivation of the jump condition by Qin et al. [7]. It can be seen that their jump condition differs from Equation (5.8) (for example, the dot product of the boundary force density with the surface normal is taken here, instead of with the lattice speed vector in Equation (5.8)), obtained in Section 5.1. Section 5.7.2 and 5.7.3 extensively discuss the flaws in the derivation by Qin et al. that lead to this discrepancy.

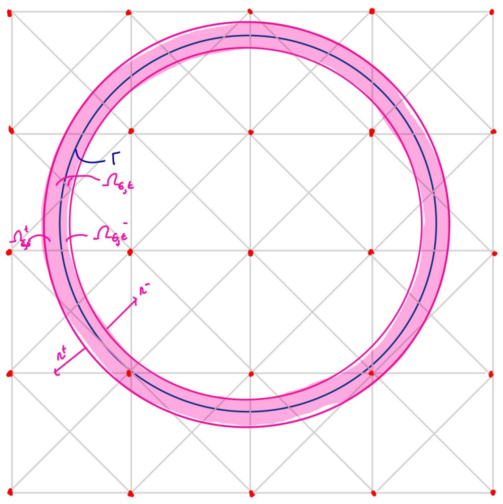


Figure 5.4: Integration domain proposed by Qin et al. [7].

With the expression for the jump condition obtained, Qin et al. implemented this jump condition by first computing the boundary forcing $\boldsymbol{G}(\boldsymbol{s},t)$ through a conventional IBM algorithm (their implementation of this algorithm is independent of the IIM). The obtained boundary force is then split into a part tangential to the surface, and one normal to the surface. The part tangential to the surface is diffused to the grid via the same IBM algorithm, but the normal part of the boundary force is imposed as a jump condition through Equation (5.40) on population links that intersect the boundary.

5.7.2. Incorrect interpretation of the dot product

Throughout this derivation, two flaws appear to be present, the first of which will beg discussed in Section. The first flaw concerns Equation (5.39), where it is assumed that $\boldsymbol{c}_i$ can be omitted from both sides of the equation. However, this is only true if $\boldsymbol{n} \parallel \boldsymbol{G}(\boldsymbol{s},t)$. Indeed, correcting the above mistakes results in

$$[\![f_i]\!] = \frac{1}{\left|\frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s}\right| c_s^2} w_i \frac{\boldsymbol{c}_i \cdot \boldsymbol{G}(\boldsymbol{s},t)}{\boldsymbol{c}_i \cdot \boldsymbol{n}}. \tag{5.41}$$

However, it should be noted that Qin et al. 'resolve' this mistake with their implementation described in Section 5.7.1, by only including the normal part of the boundary force is included in Equation (5.40). This bypasses the incorrect derivation of Equation (5.40) from Equation (5.39), as now both integrands (which form the vector with which the inner product with $\boldsymbol{c}_i$ is performed) are both parallel to each other, and only because of that can Equation (5.40) be used without appearing to lead to any issues. Mathematically, in this case, where $\boldsymbol{G}(\boldsymbol{s},t) = (\boldsymbol{G}(\boldsymbol{s},t)\cdot n)\boldsymbol{n}$, one obtains

$$[\![f_i]\!] = \frac{1}{\left|\frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s}\right| c_s^2} w_i \frac{\boldsymbol{c}_i \cdot (\boldsymbol{G}(\boldsymbol{s},t)\cdot \boldsymbol{n})\,\boldsymbol{n}}{\boldsymbol{c}_i \cdot \boldsymbol{n}} = \frac{1}{\left|\frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s}\right| c_s^2} w_i \boldsymbol{n}\cdot \boldsymbol{G}(\boldsymbol{s},t). \tag{5.42}$$

Therefore, although their derivation from Equation (5.39) to (5.40) appears somewhat careless, it can be assumed their implementation is, in fact, consistent with their equation, considering they interpret $\boldsymbol{G}(\boldsymbol{s},t)$ as including only the normal component of the force, which is correct according to Equation (5.42).

5.7.3. Incorrect interpretation of the jump condition

However, even if Equation (5.40) is corrected and written as Equation (5.41), there still persists a difference between the expression for the jump condition obtained by Qin et al. compared to Equation (5.8) following the derivation proposed in Section 5.1, with either expressions given by

$$[\![f_i]\!](\eta_c) = [\![f_i]\!](\boldsymbol{x}_c, t_c) = \frac{w_i}{c_s^2} \boldsymbol{c}_i \cdot \boldsymbol{F}(\boldsymbol{x}_c) \tag{5.8}$$

$$[\![f_i]\!] = \frac{1}{\left|\frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s}\right| c_s^2} w_i \frac{\boldsymbol{c}_i \cdot \boldsymbol{G}(\boldsymbol{s},t)}{\boldsymbol{c}_i \cdot \boldsymbol{n}}. \tag{5.41}$$

Two differences are clear - the result by Qin et al. includes a division by $\left|\frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s}\right|$, and a division by $\boldsymbol{c}_i \cdot \boldsymbol{n}$.

The term $\left|\frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s}\right|$ has a straightforward explanation that is mainly dependent on how the boundary force density is defined. In the derivation proposed in Section 5.1, the boundary force density $\boldsymbol{F}(\boldsymbol{s})$ is assumed to be in units lattice force per unit lattice length (for boundaries in two-dimensional space) or units lattice force per unit lattice length squared (for surfaces in three-dimensional space). However, Qin et al. define their boundary force density in units lattice force per unit length of the parametric coordinate $\boldsymbol{s}$ (for boundaries in two-dimensional space) or units lattice force per unit length of the parametric coordinate coordinate $\boldsymbol{s}$ squared (for surfaces in three-dimensional space).

As a result, the boundary force density in the derivation by Qin et al. needs to be converted to a boundary force density based on a unit lattice length, which is what the division by $\left|\frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s}\right|$ ensures. This is only a trivial difference, and is therefore of no real concern. Note this term should generally not be necessary anyway - when using e.g. a multi-direct forcing scheme, the obtained boundary force distribution is already in units lattice force per unit lattice length (squared) if the marker volumes ΔV_κ are measured in unit lattice length squared/cubed, which appears an intuitive choice to make. This is also the reason for not including it in Equation (5.8), as it seems superfluous, but including it is evidently not wrong, either.

However, a more fundamental flaw appears to be rooted in how the value of $[\![f_i]\!]$ is interpreted in the work of Qin et al. [7]. In particular, consider Equation (5.36) and 5.37 again, which concern the integration of the derivative of f_i along the lattice link $\boldsymbol{c}_i$:

$$\int\limits_{\Omega_{\epsilon,t}} (\boldsymbol{c}_i \cdot \nabla f_i)\phi d\boldsymbol{x} = \boldsymbol{c}_i \cdot \left(\int\limits_{\Gamma^+_{\epsilon,t}} f_i \boldsymbol{n} \phi da + \int\limits_{\Gamma^-_{\epsilon,t}} f_i(-\boldsymbol{n})\phi da - \int\limits_{\Omega_{\epsilon,t}} f_i \nabla \phi d\boldsymbol{x} \right) \tag{5.36}$$

$$= \boldsymbol{c}_i \cdot \int\limits_{\Gamma} [\![f_i]\!] \boldsymbol{n} \phi \left| \frac{\partial \boldsymbol{X}}{\partial r} \times \frac{\partial \boldsymbol{X}}{\partial s} \right| d\boldsymbol{s}. \tag{5.37}$$

In this step, Qin et al. appear to skip an important step in defining what $[\![f_i]\!]$ represents here. Equation (5.40) implies that the jump condition is a function, whereas the jump condition is actually a distribution along the boundary Γ. Indeed, the gradient of f_i should be represented as (see e.g. Kanwal [77])

$$\nabla f_i = \nabla \bar{f}_i + \int_{\Gamma} \boldsymbol{n}[\![\bar{f}_i]\!]\,(\boldsymbol{s},t)\,\delta\,(\boldsymbol{x}-\boldsymbol{X})\,d\boldsymbol{s}, \tag{5.43}$$

where $\bar{f}_i$ represents the part of the population that is of class C^0. The result of Equation (5.37) remains the same (except with $[\![\bar{f}_i]\!]|$ instead of $[\![f_i]\!]|$), and by extension, the same can be said about Equation (5.41). However, it is now pivotal to understand that $[\![\bar{f}_i]\!]$ represents the value of the *distribution* of the jump condition along the boundary Γ, but $[\![\bar{f}_i]\!]\delta\,(\boldsymbol{x}-\boldsymbol{X})$ is not a function[4].

As a result, Equation (5.41) ought to not be applied immediately to the populations f_i. Instead, one should integrate along the characteristic of the differential equation given by Equation (5.33), which is in the direction $\boldsymbol{c}_i$. Let η describe the parametric coordinate of such a characteristic, and let η_c denote the coordinate at which the boundary is intersected. In that case, one can write

$$\begin{aligned}
\int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \frac{df_i}{d\eta}d\eta &= \int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \boldsymbol{c}_i\cdot\nabla f_i d\eta \\
&= \int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \boldsymbol{c}_i\cdot\left(\nabla\bar{f}_i + \int_{\Gamma}\boldsymbol{n}[\![\bar{f}_i]\!]\,(\boldsymbol{s},t)\,\delta\,(\boldsymbol{x}-\boldsymbol{X})\,d\boldsymbol{s}\right)d\eta \\
&= \int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \boldsymbol{c}_i\cdot\nabla\bar{f}_i + \int_{\eta_c-\epsilon}^{\eta_c+\epsilon} \boldsymbol{c}_i\cdot\int_{\Gamma}\boldsymbol{n}\left(\frac{1}{\left|\frac{\partial\boldsymbol{X}}{\partial r}\times\frac{\partial\boldsymbol{X}}{\partial s}\right|c_s^2}w_i\frac{\boldsymbol{c}_i\cdot\boldsymbol{G}\,(\boldsymbol{s},t)}{\boldsymbol{c}_i\cdot\boldsymbol{n}}\right)\delta\,(\boldsymbol{x}-\boldsymbol{X})\,d\boldsymbol{s}d\eta,
\end{aligned}$$

where the last term was expanded using the expression for $[\![f_i]\!]$ from Equation (5.41). In the limit of $\epsilon\to 0$, this equation reduces to

$$[\![f_i]\!]\,(\eta_c) = 0 + \frac{\boldsymbol{c}_i\cdot\boldsymbol{n}}{\left|\frac{\partial\boldsymbol{X}}{\partial r}\times\frac{\partial\boldsymbol{X}}{\partial s}\right|c_s^2}w_i\frac{\boldsymbol{c}_i\cdot\boldsymbol{G}\,(\boldsymbol{s}_c,t)}{\boldsymbol{c}_i\cdot\boldsymbol{n}} \tag{5.44}$$

$$= \frac{1}{\left|\frac{\partial\boldsymbol{X}}{\partial r}\times\frac{\partial\boldsymbol{X}}{\partial s}\right|}\frac{w_i}{c_s^2}\boldsymbol{c}_i\cdot\boldsymbol{G}\,(\boldsymbol{s}_c)\,. \tag{5.45}$$

This result is identical to the expression derived previously in Section 5.1 (except for the previously explained inclusion of the term $\left|\frac{\partial\boldsymbol{X}}{\partial r}\times\frac{\partial\boldsymbol{X}}{\partial s}\right|$), making the derivations consistent with each other (as one would expect).

Based on this discussion, it can be concluded that the derivation by Qin et al. has serious flaws. The error in the evaluation of the dot product, described in Section 5.7.2 can be easily corrected for (and Qin et al. ended up not being affected by this error by only considering the normal component of the boundary force into account for the IIM). However, the different interpretation of the jump condition is more fundamental and leads to a different result.

The reason why this error does not appear to lead to grave errors in the results by Qin et al. is speculated to be due to how they implement the jump condition. As explained in Section 5.7.1, Qin et al. compute the boundary forcing from a conventional IBM, and only the normal part of the boundary force density is then imposed on the solution via jump conditions. As a result, assuming their implementation of the IBM algorithm is correct, this may be able to 'compensate' the spread of the erroneous jump condition to the solution at each time-step, and effectively contain the effect of it.

[4]From basic Lebesgue Integral theory, it should hold that if $f = g$ almost everywhere, then their integrated quantities are also equal for any arbitrary domain, see. e.g Berberian [78]. The term $[\![\bar{f}_i]\!]\delta\,(\boldsymbol{x}-\boldsymbol{X})$ is zero almost everywhere (except at the boundary Γ), and thus its integral over an arbitrary domain should be equivalent to integrating 0 over the same domain. However, its integral over an arbitrary domain that covers part of the boundary Γ is nonzero, and thus violates this propery, and thus $[\![\bar{f}_i]\!]\delta\,(\boldsymbol{x}-\boldsymbol{X})$ cannot be treated as a function (but should be treated as a distribution instead).

5.7.4. Additional remarks

Apart from this issue arising from their derivation of the jump conditions, it should also be noted that there appear to be gaps in the details of the implementation of their method. To be precise, it is unclear how exactly, from a discrete boundary force representation that only exists at a given number of Lagrangian markers, the boundary force at a point of intersection between a link and the boundary is computed. After all, this intersection is unlikely to coincide with one of the Lagrangian markers, and thus it is not clear what value of $\boldsymbol{G}(\boldsymbol{s}, t)$ should be used when computing the jump condition. Naturally, this could be done relatively easily through an interpolation or by putting marker points directly on the intersection points, but without details on the procedure used, it is a hard task to reproduce their methodology.

Furthermore, although not necessarily incorrect, it appears odd to use a conventional IBM treatment to compute the boundary forcing, and to then split the obtained boundary forcing in a tangential and a normal force that are treated post-IBM. For example, when using a multi-direct forcing IBM scheme, the iterative nature of the algorithm is based on the assumption that ultimately, the final diffusion of the boundary force to the fluid mesh happens in the same way as it is treated during the IBM algorithm. However, in the methodology of Qin et al., the full boundary force is diffused back to the grid during the IBM algorithm, whereas only the tangential force component is diffused back to the grid once the algorithm has obtained a boundary force distribution - the normal force component is treated differently, through the use of the jump condition of Equation (5.40). This seems questionable, as it means the iterations in the IBM algorithm worked under a different model of how the force is diffused to the fluid grid than how the fluid solver actually implements the boundary force distribution.

5.7.5. Comparison with the presented immersed interface method

Based on the foregoing discussion, several differences between the IIM proposed in Section 5.2 and the IIM proposed by Qin et al. [7] are worthy of attention.

First, it should be noted that Equation (5.8) (proposed in Section 5.1) and Equation (5.40) (obtained by Qin et al., as derived in Section 5.7.1) are not identical, even though they both aim to describe the jump experienced by a link intersecting a boundary. This is due to a mistake in evaluating the dot product by Qin et al., as well as a misunderstanding of what the jump condition represents. When correcting these mistakes, an equivalent expression is found, confirming the correctness of the derivation shown in Section 5.1. Even when correcting for these mistakes however, the derivation in Section 5.1 appears more simple and elegant than the derivation by Qin et al.

However, even if these errors are corrected, the implementation proposed in Section 5.2 still appears to have tangible benefits over the implementation proposed by Qin et al. First of all, the implementation appears to lack detail in the description by Qin et al. In comparison, the proposal and derivation of the IIM-LBM outlined in Section 5.1 appears, to the best of the author's knowledge, to be correct, and its implementation described in Section 5.2 to be fully reproducible.

Furthermore, as noted in Section 5.7.1, Qin et al. do not fully utilise IIM in their implementation, but use the IBM to obtain the boundary force distribution, and diffuse the tangential component of that distribution back to the grid. Only the normal component of the boundary force distribution is treated via the IIM, which appears at least somewhat inconsistent with how the boundary force distribution is calculated. On the other hand, the IIM proposed in Section 5.1 and 5.2 uses the principles of the IIM throughout the full algorithm, including for its calculation of the boundary force distribution.

Finally, it should be noted that Qin et al. impose their jump conditions directly on the links that are intersected by the boundary. This can form a significant computational effort, as naturally finding the intersection of the lattice links with the boundary is not a trivial task. The IIM outlined in Section 5.1 and 5.2 does not suffer from this, as the jumps are simply diffused back to the grid.

In conclusion, even though Qin et al. [7] already proposed an implementation of the IIM to the LBM, there appear to be several flaws in their methodology, which the derivation shown in Section 5.1 and 5.2 does not suffer from. Even if these flaws are corrected for, the proposal of the IIM outlined in Section 5.1 and 5.2 appears to have multiple, tangible benefits over their proposal. Therefore, the author believes the proposal of an immersed interface method in the lattice Boltzmann method described in this chapter is the first proper implementation of an IIM in the LBM.

5.8. Alternative formulations of the immersed interface method

As mentioned in Section 5.2.2, the IIM allows for more control over how the diffusion and interpolation occurs. A variety of modifications can therefore be proposed, which will be compared to the base algorithm proposed in the afore in Chapter 7. However, it should be noted that these methods did appear to have any tangible benefits over the MIIM, which is why lesser emphasis is placed on them here, and why most of Chapter 7 and 8 exclusively include the MIIM.

5.8.1. Modification I - multi-stage immersed interface method

A more rigorous modification to the IIM is to employ a multi-stage approach when determining the appropriate jump field. That is, let the total jump field be given by

$$[\![f_i]\!]\left(\boldsymbol{x}_n, t\right) = \sum_k \sum_n [\![f_i]\!]^{(n),(k)}\left(\boldsymbol{x}_n, t\right), \tag{5.46}$$

where represent the $f_i^{(n),(k)}\left(\boldsymbol{x}_n, t\right)$ is the jump field computed by stage k of the multi-stage algorithm. These stages may be computed by expanding the algorithm presented in Section 5.2.1 to the following:

1. After completing the collision operation, set $k = 0$.
2. Initialise the current stage by setting $n = 0$ and $[\![f_i]\!]^{(0),(k)} = 0$.
   (a) Diffuse the post-collision populations at their post-streaming locations to the Lagrangian boundary nodes via

   $$\hat{f}_i^{(n),(k)}\left(\boldsymbol{X}_\kappa\right) = \sum_n \left[\hat{f}_i\left(\boldsymbol{x}_n, t\right) + \sum_{\bar{k}}^{k} [\![f_i]\!]^{(n),(\bar{k})}\left(\boldsymbol{x}_n, t\right)\right] D\left(\boldsymbol{X}_k - \left(\boldsymbol{x}_n + d^{(k)}\boldsymbol{c}_i\right)\right), \tag{5.47}$$

   where $[\![f_i]\!]^{(n),(k)}\left(\boldsymbol{x}_n, t\right)$ represents the jump experienced in the kth stage by population f_i located at node $\boldsymbol{x}_n$ at time t in the upcoming streaming step, and where $0 \leq d^{(k)} \leq 1$ is the stage-parameter, controlling the temporal location at which the stage is considered.
   (b) Compute the Eulerian velocity stage at each Lagrangian boundary node via

   $$\rho \boldsymbol{U}_\kappa^{(n),(k)} = \sum_i \hat{f}_i^{(n),(k)}\left(\boldsymbol{X}_\kappa\right). \tag{5.48}$$

   (c) Compute the direct forcing on the Lagrangian nodes at the kth stage by evaluating

   $$\boldsymbol{F}_\kappa^{(n),(k)}\left(\boldsymbol{X}_\kappa\right) = \rho \frac{\boldsymbol{U}_\kappa^{(n),(k)} - \boldsymbol{U}_\kappa}{\Delta t}. \tag{5.49}$$

   (d) Compute the population jump induced at each Lagrangian node during the kth stage via

   $$[\![f_i]\!]^{(n),(k)}\left(\boldsymbol{X}_\kappa\right) = [\![\hat{f}_i]\!]\left(\boldsymbol{x}_c, t_c\right)^{(n),(k)} = \frac{w_i}{c_s^2}\boldsymbol{c}_i \cdot \boldsymbol{F}_\kappa^{(n),(k)}. \tag{5.50}$$

   (e) Diffuse the population jumps of the kth stage to the Eulerian nodes via

   $$[\![f_i]\!]^{(n),(k)}\left(\boldsymbol{x}_n, t\right) = \sum_\kappa [\![f_i]\!]^{(n),(k)}\left(\boldsymbol{X}_\kappa\right) D\left(\boldsymbol{X}_\kappa - \left(\boldsymbol{x}_n + d^{(k)}\boldsymbol{c}_i\Delta t\right)\right)\Delta V_\kappa, \tag{5.51}$$

   where V_κ is the volume of the Lagrangian body element.
   (f) Increment n by 1, and go back to step 2(a) unless a specified termination-criterion is met.
3. Go to the next stage by incrementing k by 1, and go back to step 2 until $k = k_{\max}$ is reached.
4. The total Lagrangian force on each marker and jump experienced by each population are then given by

   $$\boldsymbol{F}_\kappa\left(\boldsymbol{X}_\kappa\right) = \sum_k \sum_n \boldsymbol{F}_\kappa^{(n),(k)}\left(\boldsymbol{X}_\kappa\right) \tag{5.52}$$

   $$[\![f_i]\!]\left(\boldsymbol{x}_n, t\right) = \sum_k \sum_n f_i^{(n),(k)}\left(\boldsymbol{x}_n, t\right), \tag{5.53}$$

   and the streaming operation is performed via

   $$f_i\left(\boldsymbol{x}_n + \boldsymbol{c}_i\Delta t, t + \Delta t\right) = \hat{f}_i\left(\boldsymbol{x}_n, t\right) + [\![f_i]\!]\left(\boldsymbol{x}_n, t\right). \tag{5.54}$$

As an example, one may select a two-stage method, taking $d^{(0)} = 1/2$ and $d^{(1)} = 1$. This essentially implies first computing the jump field necessary to satisfy the boundary conditions at the time $t + \Delta t/2$, identical to the midpoint-based IIM proposed in Section 5.3. However, one then computes a 'corrective' jump field that ensures that the boundary conditions are satisfied at $t+\Delta t$. The summation of these jump fields is then taken to be the total jump field considered in the streaming step. Each stage may thus be seen as a 'correction' to the jump fields obtained at the previous stages.

5.8.2. Modification II - filtered immersed interface method

As will be discussed in Chapter 7, both the IBM and IIM suffer from zero-energy modes when the density of Lagrangian marker density is increased. This is due to the number of degrees of freedom on the boundary grid[5] exceeding the number of fluid degrees of freedom[6] when the marker density is sufficiently high. This results in non-physical spatial oscillations in the body force density distribution as the number of iterations is increased. To remedy this, one can filter the number of Lagrangian markers to select only as many markers before these oscillations start to exist.

To be precise, let the subscript denote quantities defined on the original Lagrangian grid, and let the subscript $\hat{\kappa}$ denote quantities defined on the filtered Lagrangian grid. Then, the base IIM algorithm may be modified as follows:

1. After completing the collision operation, set $n = 0$ and all $[\![f_i]\!]^{(0)} = 0$.
2. Diffuse the post-collision populations at their post-streaming locations to the filtered Lagrangian boundary nodes via
$$\hat{f}_i\left(\boldsymbol{X}_{\hat{\kappa}}\right) = \sum_n \left[\hat{f}_i\left(\boldsymbol{x}_n, t\right) + [\![f_i]\!]^{(n)}\left(\boldsymbol{x}_n, t\right)\right] D\left(\boldsymbol{X}_{\hat{\kappa}} - \left(\boldsymbol{x}_n + \boldsymbol{c}_i\right)\right). \tag{5.55}$$
3. Compute the Eulerian velocity at each filtered Lagrangian boundary node via
$$\rho \boldsymbol{U}_{\hat{\kappa}}^{(n)} = \sum_i \hat{f}_i\left(\boldsymbol{X}_{\hat{\kappa}}\right). \tag{5.56}$$
4. Compute the direct forcing on the filtered Lagrangian nodes by evaluating
$$\boldsymbol{F}_{\hat{\kappa}}^{(n)}\left(\boldsymbol{X}_{\hat{\kappa}}\right) = \frac{\boldsymbol{U}_{\hat{\kappa}}^{(n)} - \boldsymbol{U}_{\hat{\kappa}}}{\Delta t}. \tag{5.57}$$
5. Interpolate the forcing at the filtered Lagrangian nodes to the set of all Lagrangian markers via
$$\boldsymbol{F}_{\kappa}^{(n)}\left(\boldsymbol{X}_{\kappa}\right) = \frac{\sum_{\hat{\kappa}} \boldsymbol{F}_{\hat{\kappa}}^{(n)}\left(\boldsymbol{X}_{\hat{\kappa}}\right) W\left(\boldsymbol{X}_{\kappa} - \boldsymbol{X}_{\hat{\kappa}}\right)}{\sum_{\hat{\kappa}} W\left(\boldsymbol{X}_{\kappa} - \boldsymbol{X}_{\hat{\kappa}}\right)}, \tag{5.58}$$
where $W(\boldsymbol{r})$ is some interpolation weighting function.
6. Compute the population jump induced at each Lagrangian node via
$$[\![f_i]\!]^{(n)}\left(\boldsymbol{X}_{\kappa}\right) = [\![\hat{f}_i]\!]\left(\boldsymbol{x}_c, t_c\right) = \frac{w_i}{c_s^2}\boldsymbol{c}_i \cdot \boldsymbol{F}_{\kappa}^{(n)}. \tag{5.59}$$
7. Diffuse the population jumps to the Eulerian nodes via
$$[\![f_i^{(n)}]\!]\left(\boldsymbol{x}_n, t\right) = \sum_{\kappa} [\![f_i]\!]\left(\boldsymbol{X}_{\kappa}\right) D\left(\boldsymbol{X}_{\kappa} - \left(\boldsymbol{x}_n + \boldsymbol{c}_i \Delta t\right)\right) \Delta V_{\kappa}, \tag{5.60}$$
where V_{κ} is the volume of the Lagrangian body element.
8. Increment n by 1, and go back to step 2 unless a specified termination-criterion is met.
9. The total Lagrangian force on each marker and jump experienced by each population are then given by
$$\boldsymbol{F}_{\kappa}\left(\boldsymbol{X}_{\kappa}\right) = \sum_n \boldsymbol{F}_{\kappa}^{(n)}\left(\boldsymbol{X}_{\kappa}\right) \tag{5.61}$$
$$[\![f_i]\!]\left(\boldsymbol{x}_n, t\right) = \sum_n f_i^{(n)}\left(\boldsymbol{x}_n, t\right), \tag{5.62}$$
and the streaming operation is performed via
$$f_i\left(\boldsymbol{x}_n + \boldsymbol{c}_i \Delta t, t + \Delta t\right) = \hat{f}_i\left(\boldsymbol{x}_n, t\right) + [\![f_i]\!]\left(\boldsymbol{x}_n, t\right). \tag{5.63}$$

[5]I.e., the boundary force on each marker.

[6]I.e., the number of fluid points that the populations are diffused to.

A methodology to automatically select the markers to be used on the filtered grid is not presented here. Instead, in Chapter 7 this algorithm will be considered for a simple cylinder with a uniform marker spacing, where the number of markers can easily be manually filtered. As there was no significant benefit to the approach outlined above compared to the proposed base IIM, it was decided to not pursue this modification further and develop an automatic filtering system.

5.8.3. Modification III - sharp immersed interface method

It has previously been established in Section 5.2.1 that the population jumps across the boundary can be computed analytically, if the value of the boundary force distribution is known at the intersection point. It is therefore possible to modify the last step of any of the previously listed algorithms.

The final modification proposed here is to include Equation (5.8) more explicitly into the algorithm. After all, once the boundary force distribution is computed, the analytic jumps of only the links that are intersected by the boundary may be readily computed. This approach is similar to the one used by Qin et al. [7], as described in Section 5.7. Thus, after obtaining $\boldsymbol{F}_\kappa(\boldsymbol{X}_\kappa)$, one may replace Equations by (5.15) by

$$[\![f_i]\!] (\boldsymbol{x}_n, t) = \frac{w_i}{c_s^2} \boldsymbol{c}_i \cdot \boldsymbol{F}(\boldsymbol{x}_c), \tag{5.64}$$

if the link $\boldsymbol{c}_i$ originating from node $\boldsymbol{x}_n$ is intersected by the boundary, and where $F(\boldsymbol{x}_c)$ is determined by interpolation from nearby Lagrangian markers,

$$\boldsymbol{F}^{(n)}(\boldsymbol{X}_c) = \frac{\sum\limits_{\kappa} \boldsymbol{F}_\kappa^{(n)}(\boldsymbol{X}_\kappa)\, W(\boldsymbol{X}_\kappa - \boldsymbol{X}_c)}{\sum\limits_{\hat{\kappa}} W(\boldsymbol{X}_\kappa - \boldsymbol{X}_c)}. \tag{5.65}$$

Note that Equation (5.8) is difficult to implement in the iterative algorithm directly. Doing so would mean that the local value of the boundary force distribution is only found at the intersections with the links, and only along the direction of the intersected link. This would create two challenges. First, in Equation (5.11), $\boldsymbol{F}_\kappa^{(n)}$ may not tend to the zero-vector as the number of iterations is increased, as there may be a non-zero component perpendicular to the intersected lattice link. This may make it difficult to establish a satisfactory termination criterion. Secondly, this would introduce challenges when communicating with a structural solver for example, as some mechanism would need to be in place to allow reconstruction of the full boundary force distribution on the grid of the structural solver.

5.9. Conclusion

In this chapter, the adaption of the IIM into the LBM-framework was proposed. The main difference with respect to the IBM is during which phase of the LBM the boundary treatment is performed. Whereas the IBM requires evaluation of the boundary forces before collision, the IIM evaluates the boundary forces post-collision, accounting for the post-streaming populations. From a methodological standpoint, this has two benefits:

- The IBM is only able to capture flow field information at the beginning of the time step, and does not take into account information resulting from the collision step, meaning that it is not guaranteed that post-streaming, the velocity at the boundary markers is still equal to the imposed boundary condition. On the other hand, the IIM is designed to account for the post-streaming flow field information.
- In similar fashion, the IIM opens up opportunities to alter the discretisation of the boundary force integral in Equation (5.5). As was proposed before, one can e.g. use a midpoint integration rule or a multi-step method. The effect of this can be explored in the future.

A number of modifications has been proposed in this Chapter, which are summarised in Table 5.1.

Table 5.1: List of boundary treatment configurations introduced in Chapter 4 and 5.

Configuration	Description	Reference
IBM	Classical IBM (nearly identical to the default IIM)	Chapter 4.
IBM-SPLIT	IBM using split-forcing	Section 4.2.
MIIM	Modified version of the IIM, using midpoint interpolation and diffusion	Section 5.3.
IIM-MOD-A	Modified version of the IIM, using multi-stage interpolation and diffusion	Section 5.8.1.
IIM-MOD-B	Modified version of the IIM, where the number of Lagrangian markers is increased, where the forcing is first computed on a subset of markers, then interpolated to nearby markers	Section 5.8.2
IIM-MOD-C	Modified version of the IIM, where the jumps are imposed directly on links intersected by the boundary, rather than through diffusion	Section 5.8.3.

6 Implementation details

In this Chapter, several important details of the LaBIB-FSI solver are discussed. Section 6.1 elaborates on the structural solver to which the fluid solver is coupled, and Section 6.2 describes the interface between both solvers. Section 6.3 discusses the multigrid approach that is used in the fluid solver, and finally, Section 6.4 discusses the way in which boundary conditions on the outer edges of the domain are imposed.

6.1. Structural solver

To be able to simulate fluid flows including structural deformation, the fluid solver was coupled to `pyfe3d` [13], a general purpose finite-element solver for static and dynamic analysis. `pyfe3d` offers a variety of element discretisation types. However, due to time constraints, it was chosen to limit the coupling with the `pyfe3d` to being able to simulate cantilevered beams, such that the benchmark cases by Turek and Hron [8] can be validated, as shown in 8.3. As a result, it was chosen to only implement an interface to the `BeamC`-type, which is a representation of a three-dimensional Timoshenko beam element, using consistent shape functions. For details on the derivation and implementation of these elements, the reader is referred to Luo [79]. Instead, this section will explain how the structural model for a cantilever beam is defined.

6.1.1. Spatial discretisation

First, the spatial discretisation of the beam will be described. The `BeamC`-elements represent one-dimensional line segments connecting various cross-sectional centroids along the cylindrical axis of a beam. As a result, a beam seen by `pyfe3d` is infinitely thin, with its cross-sectional geometry fully encapsulated by the cross-sectional moments of area. However, for the fluid solver, it was found to be desirable to be able to model the physical thickness of a beam, as this thickness may affect the vortex development caused by the tip of the beam. In particular, for the validation cases of Turek and Hron [8], a cantilevered beam with a thickness of 0.02 m is attached to the trailing edge of a cylinder with diameter of 0.10 m; it was therefore not considered safe to neglect the physical thickness of the beam in the fluid solver.

Therefore, two discretisations of the beam are made. First, a discretisation of the outside surface (minus the attachment to the physical wall) is performed, as shown in Figure 6.1. This discretisation is used for the interpolation of mechanical quantities from the fluid surface, as will be explained in Section 6.2. The mechanical quantities are then translated to a mesh that coincides with the locations of the `BeamC`-elements. Note that these `BeamC`-elements align in the cross-sectional direction with the surface discretisation, as shown in Figure 6.1.

The vertical alignment allows for straightforward translation of the mechanical quantities from one mesh to the other. In particular, the forces at the surface discretisation may be translated to the `BeamC`-elements by simply taking the sum of the forces at the upper and lower surface. Similarly, the displacements found by `pyfe3d` at the `BeamC`-elements may be easily transformed to the surface discretisation by equating the displacement (consisting of components Δx and Δy) at a surface node to the translation of the corresponding `BeamC`-element plus a rotation θ about this `BeamC`-element.

Since `BeamC`-elements are three-dimensional elements, whereas the fluid simulations are two-dimensional

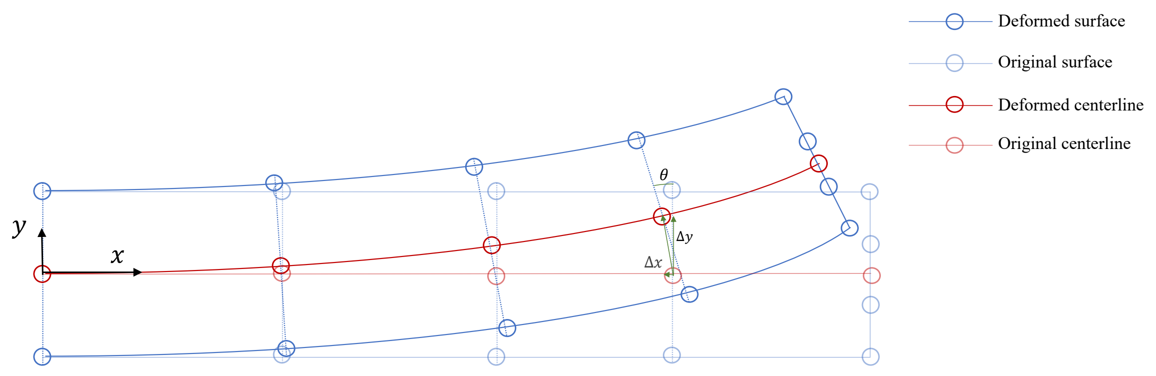


Figure 6.1: Structural model of a cantilevered beam, in its original and deformed position. The displacements at the outer surface nodes can be evaluated directly from the translation and rotation of the beam elements located at the centerline.

(including the Turek and Hron [8] benchmark), the stiffness E_{xx} of the elements was set equal to

$$E_{xx} = \frac{E}{1-\nu^2} \tag{6.1}$$

where E is the E-modulus of the material and ν is the Poisson's ratio of the material. This was done to prevent deformation in the direction perpendicular to the plane of the fluid simulation, as done by Geller et al. [80] and derived by [81].

6.1.2. Temporal integration

With the spatial discretisation of the beam performed, a stiffness matrix K, damping matrix C and mass matrix M can be set up, to create a system of equations of the form

$$M\ddot{\boldsymbol{u}} + C\dot{\boldsymbol{u}} + K\boldsymbol{u} = \boldsymbol{F}, \tag{6.2}$$

where $\boldsymbol{u}$ is the vector containing the displacements of each element, and $\boldsymbol{F}$ the forcing applied at each element. A Newmark method [82] was employed, with parameters $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$. That is, if $\boldsymbol{u}^{(k)}$ represents the displacement vector at time-step k, then the predictor is computed as

$$\tilde{\boldsymbol{u}}^{(k+1)} = \boldsymbol{u}^{(k)} + \Delta t\dot{\boldsymbol{u}}^{(k)} + \frac{\Delta t^2}{2}\left(1-2\beta\right)\ddot{\boldsymbol{u}}^{(k)} \tag{6.3}$$

$$\tilde{\dot{\boldsymbol{u}}}^{(k+1)} = \dot{\boldsymbol{u}}^{(k)} + \Delta t\left(1-\gamma\right)\ddot{\boldsymbol{u}}^{(k)}. \tag{6.4}$$

The acceleration at time-step $k+1$ is then computed via

$$M\ddot{\boldsymbol{u}}^{(k+1)} = \boldsymbol{F}^{(k)} - C\tilde{\dot{\boldsymbol{u}}}^{(k+1)} - K\tilde{\boldsymbol{u}}^{(k+1)}. \tag{6.5}$$

Finally, the corrector is computed as

$$\boldsymbol{u}^{(k+1)} = \tilde{\boldsymbol{u}}^{(k+1)} + \beta\Delta t^2\ddot{\boldsymbol{u}}^{(k+1)} \tag{6.6}$$

$$\dot{\boldsymbol{u}}^{(k+1)} = \tilde{\dot{\boldsymbol{u}}}^{(k+1)} + \gamma\Delta t\ddot{\boldsymbol{u}}^{(k+1)}, \tag{6.7}$$

where the value of $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$ were used, to obtain a trapezoidal rule with unconditional stability.

It should be noted that the Timoshenko beam elements used in the `pyfe3d`-solver do not account from any structural damping due to the material. Instead, to stabilise the solver, a small amount of modal damping is allowed; that is, the damping matrix is set equal to

$$C = \alpha_d M + \beta_d K, \tag{6.8}$$

where α_d and β_d are the mass and stiffness damping factor, respectively. These damping factors have been set equal to 1×10^{-3} throughout all validation cases and sensitivity analyses described in Chapter 7-9.

6.2. Fluid-structure-interaction

To couple the structural surface discretisation with the fluid surface discretisation, an fluid-structure interaction operator was implemented. This operator consists of two elements: spatial interpolation using radial basis functions, and temporal regulation of the communication between the grids.

6.2.1. Spatial interpolation

Spatial interpolation is necessary to translate mechanical quantities from a fluid mesh to a structural mesh. In particular, the boundary forcing found by the IBM/MIIM should be translated from the fluid mesh to the structural mesh, and displacements and velocities should be translated from the structural mesh to the fluid mesh.

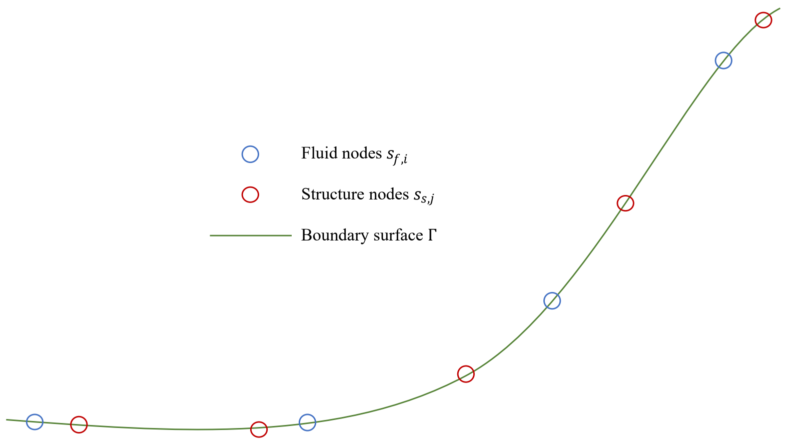


Figure 6.2: Discretisation of a boundary, with fluid nodes (as used by the IBM/IIM by the fluid solver described in Chapter 4 and 5) and structural nodes (as used by the structural model described in Section 6.1).

To do so, a consistent interpolation approach based on radial basis functions was used. Consider the fluid and structural mesh shown in Figure 6.2, where $s_{f,i}$ denotes the nodes on the fluid mesh at the boundary Γ, and $s_{s,j}$ denotes the nodes on the structure mesh at the same boundary. Note that the boundary is in a two-dimensional plane, and the boundary is therefore parametrised by a single variable s.

Then, consider construction of the interpolation matrix H_{fs}, which denotes the transformation matrix to transform a quantity on the fluid mesh to the structure mesh. The interpolant is first defined to be a sum of radial basis functions centered at the boundary nodes of the mesh from which the quantity needs to be interpolated, i.e.

$$S(\boldsymbol{s}) = \sum_i \gamma_i \phi\left(\left|s - s_{f,i}\right|\right), \tag{6.9}$$

where $\phi(r)$ is the chosen interpolation function, and γ_i the corresponding interpolation weights. Radial basis functions with compact support are used; in particular the following Wendland function [83] was used:

$$\phi(r, w) = \left(1 - \frac{|r|}{w}\right)^4 \left(4\frac{|r|}{w} + 1\right), \tag{6.10}$$

where w is the width of the compact support, an arbitrary parameter that can be set by the user. Then, the transformation H_{fs} can be computed as

$$H_{fs} = \Phi_{fs}\Phi_{ff}^{-1}, \tag{6.11}$$

where

$$\Phi_{fs,ij} = \phi\left(\left|s_{s,j} - s_{f,i}\right|\right) \tag{6.12}$$
$$\Phi_{ff,ij} = \phi\left(\left|s_{f,j} - s_{f,i}\right|\right), \tag{6.13}$$
$$\tag{6.14}$$

where H_{fs} is the transformation matrix such that

$$\boldsymbol{a}_s = H_{fs}\boldsymbol{a}_f, \tag{6.15}$$

where $\boldsymbol{a}_s$ is a vector containing a certain quantity a at all structure nodes at the boundary Γ, and $\boldsymbol{a}_f$ is the vector containing this quantity at all fluid nodes. To construct the transformation matrix H_{sf}, one simply repeats the above derivation, simply swapping the mesh subscripts at every instance.

It should be acknowledged that he approach above is not consistent - even when a constant quantity is transformed from one mesh to the other, the constant value is not recovered exactly. This could be resolved by adding a polynomial of degree p to the interpolant, i.e. to modify Equation (6.9) to

$$S(\boldsymbol{s}) = \sum_i \gamma_i \phi\left(\left|s - s_{f,i}\right|\right) + \sum_p \beta_p s^p, \tag{6.16}$$

where β_p are the corresponding polynomial coefficients.

Note that the used radial basis functions (as derived by Wendland [83]) are positive definite, and therefore there is no risk that the current implementation of the interpolation matrix H_{ff} is singular, even without this polynomial, as per Buhmann [84].

6.2.2. Temporal communication

Mechanical properties need to be communicated from the fluid solver to the structure solver and vice versa. In particular, the fluid forces on the boundary should be passed to the structure solver, and the structural displacements should be communicated to the fluid solver.

A sub-iterated parallel scheme is used, as shown in Figure 6.3 when using IBM scheme, and in Figure 6.4 when using an IIM-based method. In both cases, the fluid boundary treatment and structural solver are executed simultaneously[1], and sub-iterations are possible, updating the input to both the fluid and structural solver based on the output of the other solver.

It should be noted that a small difference is present between the IBM (Figure 6.3) and the IIM (Figure 6.4) when used in a partitioned scheme with subiterations. In case of the IBM, the fluid boundary treatment is executed before the collision operation happens; the latter does then not affect the results of the IBM scheme, so only the IBM scheme needs to be iterated. In case of the IIM, the fluid boundary treatment is executed *after* the collision operation, and the results of the boundary treatment does not affect the collision treatment; as a result, the collision operator is only executed once, and the iterations only occur on the boundary treatment afterwards.

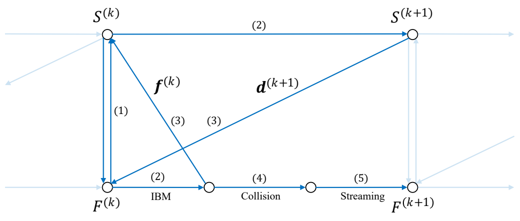


Figure 6.3: Discretisation of a boundary, with fluid nodes (as used by the IBM/IIM by the fluid solver described in Chapter 4 and 5) and structural nodes (as used by the structural model described in Section 6.1).

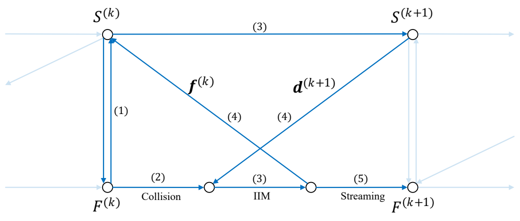


Figure 6.4: Discretisation of a boundary, with fluid nodes (as used by the IBM/IIM by the fluid solver described in Chapter 4 and 5) and structural nodes (as used by the structural model described in Section 6.1).

6.3. Multigrid approach

In order to accelerate the fluid solver, a multigrid approach is utilised. The basic structure of this multigrid is shown in Figure 6.5; each grid refinement level is exactly twice as refined as the previous level, and each sub grid is fully contained within its 'parent' block.

[1]It should be noted that they are not executed in parallel from a pure implementation point of view - the LaBOB-FSI solver simply executes the structural solver after performing the fluid boundary treatment, but no information is exchanged between the solvers until after the structural solver is executed.

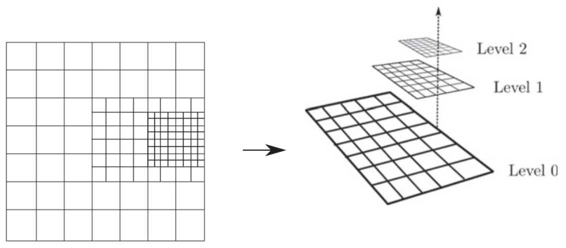


Figure 6.5: General structure of the chosen multigrid approach. Taken from Lagrava et al. [85, p. 4810].

6.3.1. Time stepping scheme

Each grid level is refined with a factor 2, such that every second time step of the finer grid, and the refined grid is placed such that they share grid nodes with the coarser grid whenever possible. Figure 6.6 shows the time stepping procedure in a 3-level multigrid in more detail. The finest grid uses a time step of Δt (with a grid spacing of Δx); its parent grid uses a time step of $2\Delta t$ and a grid spacing of $2\Delta x$, and the main grid uses a time step of $4\Delta t$ and a grid spacing of $4\Delta x$.

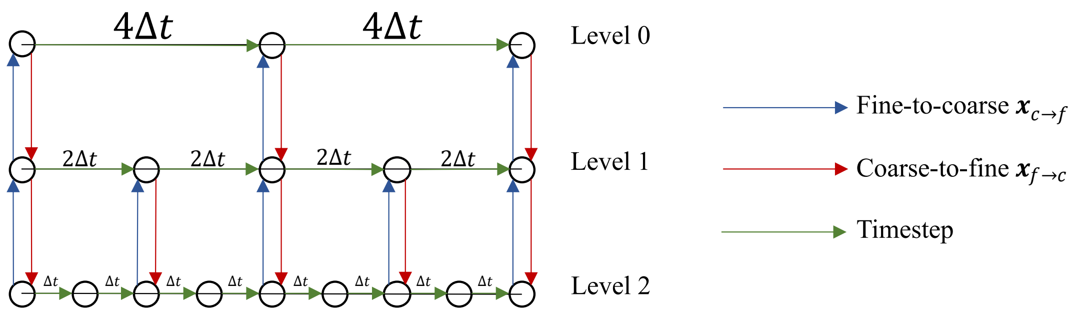


Figure 6.6: Time stepping scheme of the proposed multigrid procedure. The finest grids runs at a time step of Δt; and every level above runs at a time step twice as long as the previous level. Whenever a parent grid matches up in time with its sub grids, communication between the grids takes place.

6.3.2. Multigrid communication

Whenever two grid levels arrive at the same point in time, communication between a parent grid and its subgrid needs to occur. Communication from the fine to coarse grid is necessary to utilise the improved accuracy of the fine grid. Conversely, communication from the coarse to fine grid is necessary, as the fine grid requires flow information at its edges that do not coincide with a physical boundary. To illustrate this, consider the example of a multigrid configuration shown in Figure 6.7, where the coarse grid is denoted by circles, and the fine grid by crosses. The difference between the 'real' and 'fictitious' markers will be apparent shortly.

At $t = 0$, all grid nodes of all grid levels can be initialised. During the first time step on the fine grid, collision and streaming takes place. As collision is purely local, this operation can be executed without issue. The streaming step introduces a problem, however, as there is only a physical boundary present on the east edge of the grid configuration shown in Figure 6.7. Along the other edges, no boundary condition is physically present, and it's therefore not evident which populations stream into the grid at those edges.

Therefore, it is only possible to stream the populations that originate from inside the fine grid or the physical boundary condition. This means that after the first time step on the refined grid, the grid nodes along the edges (except the edge fixed by a physical boundary condition) are diluted and any information propagating from them can be discarded. However, collision can still occur on the nodes interior to this outer edge, and streaming can be performed on the populations originating from the nodes interior to this outer edge. This leaves the nodes adjacent to the outer edge diluted as populations from the outer edge would have streamed into them.

After two time steps, the coarse grid will have executed a single time step too. Therefore, at this time, the diluted populations (located in the red zone in Figure 6.7) can be corrected based on the populations on the coarse grid. Fine markers that do not coincide with a coarse marker can be corrected based on an interpolation procedure. On the other hand, coarse markers located within the blue zone can utilise the populations found by the fine grid.

The red zone in Figure 6.7 therefore essentially acts as a padded buffer around the 'real' fine grid. This slightly increases the computational cost per time step for the fine grid, as the effective grid is larger than actually desired. However, communication between the fine grid and coarse grid only needs to happen every second time step on the fine grid, instead of requiring a temporal interpolation of the coarse grid simulation to allow for a communication at every time step of the finer grid.

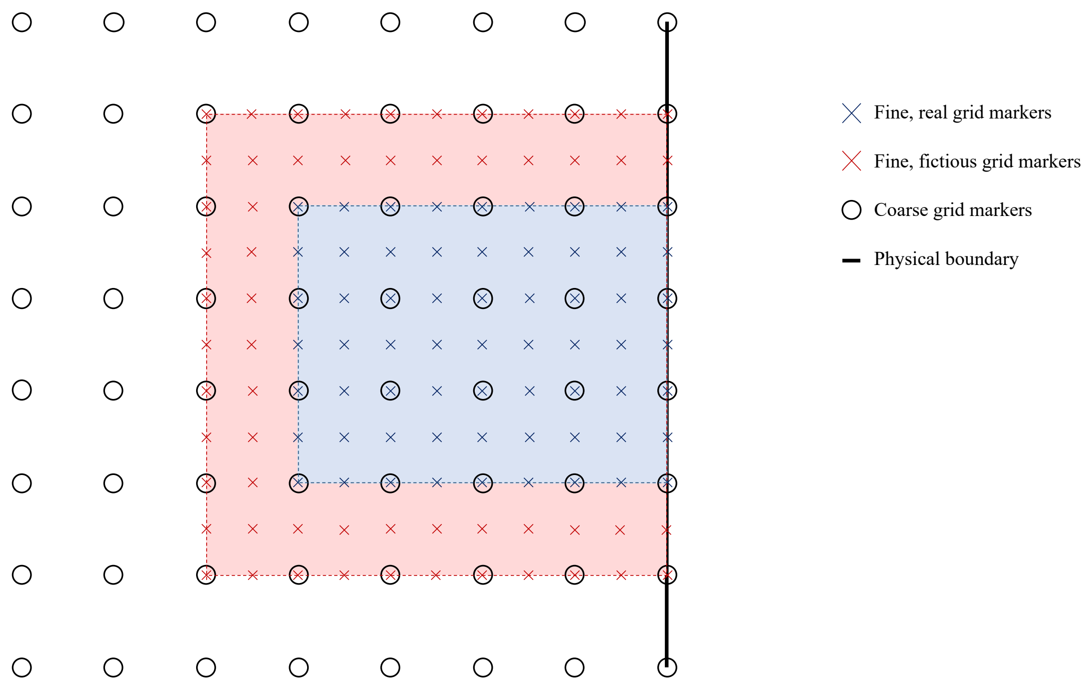


Figure 6.7: Example of a simple multi-grid configuration. In the blue area, information from the fine grid is used to improve the accuracy of the solution at coinciding coarse grid nodes. The red area is an area padded around the blue area, and becomes diluted over time due to lack of physical boundary conditions.

6.3.3. Multigrid correction

As described before, the populations in the fictitious zone of the fine grid need to be corrected by the coarse grid, whereas the populations of the coarse grid inside the real zone of the fine grid are corrected by coinciding markers in the fine grid. As shown by Dupuis et al. [86], it is not correct to equate the populations at coinciding markers on different levels of grid refinement. Instead, let $\omega_c = (\Delta t_c / \tau_c)$ be the relaxation rate of the coarse grid, and $\omega_f = (\Delta t_f / \tau_f)$ the relaxation rate of the fine grid. Furthermore, f_i^{eq} represents the equilibrium part of the population, and f_i^{neq} the non-equilibrium part. Then, the populations at a coarse grid node may be computed from the populations at the coinciding fine grid node via

$$f_{i,c}(\boldsymbol{x}_c, t) = f_{i,f}^{\text{eq}}(\boldsymbol{x}_c, t) + \frac{2\omega_f}{\omega_c} f_{i,f}^{\text{neq}}(\boldsymbol{x}_c, t)\,, \tag{6.17}$$

In other words, a rescaling of the non-equilibrium part is necessary, as shown by Dupuis et al. [86]. Similarly, to compute populations at a fine grid node coinciding with a coarse grid node, the populations may be computed from

$$f_{i,f}\left(\boldsymbol{x}_f, t\right) = f_{i,c}^{\text{eq}}\left(\boldsymbol{x}_f, t\right) + \frac{\omega_c}{2\omega_f} f_{i,c}^{\text{neq}}\left(\boldsymbol{x}_f, t\right). \tag{6.18}$$

If a fine grid node does not coincide with a coarse grid node, the populations may be interpolated from nearby coarse nodes. Since the fine grid nodes either coincide with the coarse grid nodes, or are located midway between two coarse grid nodes, the interpolation is straightforward and can be performed through simple linear interpolation.

6.3.4. Summary of algorithm

The multigrid algorithm can thus be summarised as in Algorithm 1.

Algorithm 1 Multigrid algorithm.

```
procedure SimulateTimeStep(grid, level)
    Do Something
    subGrids ← grid.getSubgrids()
    for subGrid in subGrids do
        SimulateTimeStep ( subGrid, level + 1 )
        SimulateTimeStep ( subGrid, level + 1 )
    end for
    grid.collide()
    grid.stream()
    for subGrid in subGrids do
        InterpolateFromCoarseToFineGrid ( subGrid, grid )
        InterpolateFromFineToCoarseGrid ( subGrid, grid )
    end for
end procedure

grid ← mainGrid                                    ▷ mainGrid is the coarsest grid
SimulateTimeStep ( grid, 0 )
```

6.4. Edge treatment

At the edges of the domain, some links will have their populations streamed to outside the domain, whereas some links require their populations to be streamed from outside the domain. In particular, consider Figure 6.8, which shows the upper edge of a two-dimensional fluid domain. During streaming, the populations f_0, f_1, f_2, f_6, f_7 and f_8 can be readily streamed from the adjacent nodes, but populations f_3, f_4 and f_5 would originate from outside the domain.

Although the IBM described in Chapter 4 and IIM described in Chapter 5 can deal with arbitrarily shaped boundaries inside the fluid domain, they do not provide a solution for the treatment of the edges in an LBM, as the diffusive zones would extend beyond the fluid domain. Indeed, a different treatment is necessary to establish the missing populations f_3, f_4 and f_5. It was chosen to use a small variation on the regularized boundary condition originally proposed by Latt et al. [87].

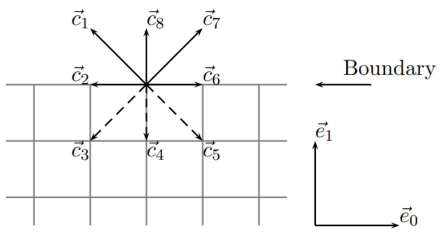


Figure 6.8: Illustration of lattice populations at upper edge of a fluid domain. Taken from [87, p. 54].

6.4.1. Modified regularised boundary condition

Let $f_{\mathscr{K}}(\boldsymbol{x}_n, t+\Delta)$ denote the known post-streaming populations that can be simply obtained from streaming from adjacent nodes, and let $f_{\mathscr{U}}(\boldsymbol{x}_n, t+\Delta t)$ denote the unknown post-streaming populations that would be streamed from nodes outside the fluid domain. Furthermore, assume only the velocity $\boldsymbol{u}$ at the boundary is known.

In the regularized boundary condition procedure, all post-streaming populations are replaced based on the imposed macroscopic conditions (in contrast to e.g. the bounce-back approach, which only replaces those that originate from outside the domain [88]). The mass before the boundary condition is applied is given by

$$\rho\left(\boldsymbol{x}_i, t+\Delta t\right) = \sum_{\mathcal{K}} f_{\mathcal{K}}\left(\boldsymbol{x}_n, t+\Delta t\right) + \sum_{\mathcal{U}} \hat{f}_{\bar{\mathcal{U}}}\left(\boldsymbol{x}_n, t\right), \tag{6.19}$$

where $\hat{f}_{\bar{\mathcal{U}}}$ are the post-collision populations on the reversed links of the unknown populations (e.g. in Figure 6.8; f_2, f_5 and f_6); they represent the populations that will leave the domain after streaming (and hence their value *before* streaming is included).

To ensure that mass is conserved during the boundary treatment, this should equal the mass of the populations post-replacement. The next step is to compute the equilibrium population corresponding to the computed density and imposed velocity at the boundary, simply using Equation (3.21).

The next step to compute the off-equilibrium tensor, whose general expression is given by

$$\Pi^{(1)}\left(\boldsymbol{x}_n, t+\Delta t\right) = f_i^{\text{neq}}\left(\boldsymbol{x}_n, t+\Delta t\right)\boldsymbol{e}_i\boldsymbol{e}_i^T. \tag{6.20}$$

The non-equilibrium populations of $f_{\mathcal{K}}$ are computed easily via

$$f_{\mathcal{K}}^{\text{neq}} = f_{\mathcal{K}} - f_{\mathcal{K}}^{\text{eq}}. \tag{6.21}$$

The non-equilibrium populations corresponding to $f_{\mathcal{U}}$ are then found by assuming the non-equilibrium part of the populations are bounced-back at the boundary. In other words,

$$f_{\mathcal{U}}^{\text{neq}}\left(\boldsymbol{x}_n, t+\Delta t\right) = f_{\bar{\mathcal{U}}}^{\text{neq}}\left(\boldsymbol{x}_n, t+\Delta t\right). \tag{6.22}$$

With all non-equilibrium populations known, the off-equilibrium tensor from Equation (6.20) can be computed. It can be shown (Latt et al. [87]), through a Chapman-Enskog expansion, that this tensor is related to the strain rate tensor $\boldsymbol{S}$ and the non-equilibrium populations as

$$f_i^{\text{neq}}\left(\boldsymbol{x}_n, t+\Delta t\right) \approx -\frac{\rho w_i \tau}{c_s^2}\boldsymbol{Q}_i : \boldsymbol{S}\left(\boldsymbol{x}_n, t+\Delta t\right) = \frac{w_i}{2c_s^4}\boldsymbol{Q}_i : \boldsymbol{\Pi}^{(1)}\left(\boldsymbol{x}_n, t+\Delta t\right), \tag{6.23}$$

where $\boldsymbol{Q}_i = \boldsymbol{e}_i\boldsymbol{e}_i^\top - c_s^2\boldsymbol{I}$. Thus, the adjusted populations can now be easily computed from

$$f_i\left(\boldsymbol{x}_n, t+\Delta t\right) = f_i^{\text{eq}}\left(\boldsymbol{x}_n, t+\Delta t\right) + f_i^{\text{neq}}\left(\boldsymbol{x}_n, t+\Delta t\right). \tag{6.24}$$

In summary, the implemented regularized boundary condition consists of the following steps:

1. Computation of the density at the boundary via Equation (6.19).
2. Compute the equilibrium populations through Equation (3.21).
3. Compute the off-equilibrium populations through Equation (6.21) and (6.22).
4. Compute the off-equilibrium tensor through Equation (6.20).
5. Redistribute the off-equilibrium tensor through Equation 6.23.
6. Take the sum of the equilibrium and off-equilibrium populations with Equation (6.24).

6.4.2. Comparison with other boundary conditions

As it is expected that edge treatment only has a small effect on the overall fluid simulation and the treatment of (moving) boundaries inside the domain is of greater interest in this work, no extensive, quantitative comparison to other boundary treatments has been performed. Naturally however, the implementation was verified to be mass-conserving if the boundary velocity perpendicular to the edge was 0, and the imposed velocity constraint was met exactly.

Compared to the original proposal of the regularised boundary condition by Latt [87], the implementation described above differs in the computation of the unknown density at the boundary, Equation (6.19). Latt computes this density by noting that

$$\rho\left(\boldsymbol{x}_i, t+\Delta t\right) = f_{\mathcal{K}}\left(\boldsymbol{x}_i, t+\Delta t\right) + f_{\mathcal{U}}\left(\boldsymbol{x}_i, t+\Delta t\right) \tag{6.25}$$

$$\rho\left(\boldsymbol{x}_i, t+\Delta t\right) u_{\perp} = f_{\mathcal{U}}\left(\boldsymbol{x}_i, t+\Delta t\right) - f_{\bar{\mathcal{U}}}\left(\boldsymbol{x}_i, t+\Delta t\right), \tag{6.26}$$

where $u_\perp$ is the velocity perpendicular to the boundary. It can be shown that this results in

$$\rho\left(\boldsymbol{x}_i, t+\Delta t\right) = \frac{1}{1+u_\perp}\left(2 f_{\mathscr{K}}\left(\boldsymbol{x}_i, t+\Delta t\right) + f_{\bar{\mathscr{U}}}\left(\boldsymbol{x}_i, t+\Delta t\right)\right). \tag{6.27}$$

It was chosen to use Equation (6.19) instead of (6.27), as Equation (6.19) is simpler to implement as it does not require decomposition of the boundary velocity, and can be readily extended to e.g. corner nodes, whereas Equation (6.27) is not easily extensible to more complex edge nodes (such as planar surfaces in 3D) [2].

Additionally, compared to other existing boundary conditions (such as the bounce-back condition ([89]), Zou-He [90], and the method proposed by Inamuro et al. [91]) the regularized boundary condition and its implementation proposed here are characterised by a number of advantages [88]. For example, the regularised boundary condition conserves mass, even for a boundary moving in a tangential direction, unlike the bounce back condition. Furthermore, particularly the implementation proposed above is easily extensible to corners and 3D-edges, unlike the Zou-He method and the method by Inamuro et al. It is also conserves the macroscopic mass, velocity and strain rate tensor at the boundary, a property not shared by the other boundary conditions. Finally, the regularised boundary condition is fully local, unlike some more advanced boundary conditions that rely on interpolation from nearby lattice nodes to find macroscopic quantities.

Regarding its accuracy, Latt et al. [88] found that the Inamuro et al. and Zou-He boundary conditions boasted better results at low 2D Reynolds number flows. For 3D flows, the Zou-He boundary condition significantly decreased in accuracy, and the bounce-back and Zou-He boundary conditions both suffered notable stability issues, requiring finer grids to simulate the same Reynolds number flow. On the other hand, the regularised boundary condition was found to be very stable, capable of simulating significantly higher Reynolds number flow than the Inamuro et al. and Zou-He boundary conditions. Thus, although perhaps not as accurate as other boundary conditions, the regularized boundary condition boasts far better stability, an important reason for why it was opted to use the regularised boundary condition in this work.

III

Validation

In Chapter 7-11, the LaBIB-FSI solver is validated, the performance of the midpoint immersed interface method is compared against that of the immersed boundary method, and conclusions and recommendations are drawn. Specifically, Chapter 7 validates the LaBIB-FSI solver for 2D fluid flow with stationary boundaries, whereas Chapter 8 validates the LaBIB-FSI solver for 2D fluid flow with moving boundaries, either through rigid, prescribed motion or through two-way fluid-structure interaction, with both Chapters shedding clear light on the advantages of the immersed interface method over the immersed boundary method. Chapter 9 provides additional analysis to certain validation cases, and sheds light on the sensitivity of certain numerical input parameters. Chapter 10 concludes and answers the research questions set out in Chapter 2, whereas Chapter 11 provides a list of recommendations for future research.

7 Validation of 2D fluid solver

This Chapter validates the fluid solver of the LaBIB-FSI software, for problems involving stationary boundaries. Three validation cases are considered. Section 7.1 considers a simple lid-driven cavity flow proposed by Botella and Peyret [92], to validate the solver for flows with no solid object in the interior domain. Section 7.2 considers a Taylor-Green vortex flow immersed by a cylinder, in which the analytical solution is imposed as a boundary condition along the boundary of the cylinder, previously used by authors such as Kang and Hassan [35] and Wu and Shu [93]. Section 7.3 evaluates the solver for the CFD1, CFD2 and CFD3 benchmark cases proposed by Turek and Hron [8].

7.1. Lid-driven cavity flow

To validate the fluid solver without any fluid-structure interaction present, the benchmark case by Botella and Peyret [92] is used. In this validation case, a lid is pushed over the top side of a square domain, resulting in vortex generation inside the domain.

In this lid-driven cavity flow, in a domain of $[0, L] \times [0, L]$, the boundary at $y = L$ has a boundary velocity equal to $u_x = -u_0$ and $u_y = 0$, and all boundaries are at rest. Although this results in a steady flow, as the LBM is inherently transient, the flow is initialised as being at rest, and the simulation is then evaluated at $t = 200L/u_0$, which was heuristically found to be sufficiently long to converge to a steady-state. The Reynolds number (based on the length of the domain L and velocity u_0) is held constant at $\mathrm{Re} = 1000$. The Mach number was held constant at $M = 0.2$ (based on the velocity u_0) for all simulations.

An example of the resulting vorticity field is included in Figure 7.1.

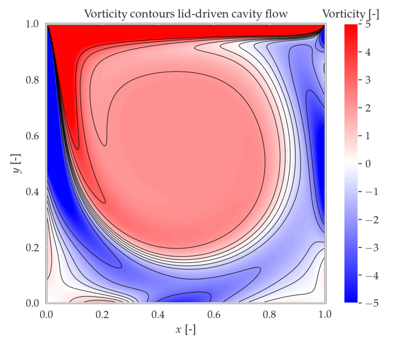


Figure 7.1: Plot of the vorticity at $t = 200L/u_0$, using $N = 257$ nodes in each direction, using the cumulant MRT scheme. For the purpose of plotting, $u_0 = 1.0$ and $L = 1.0$.

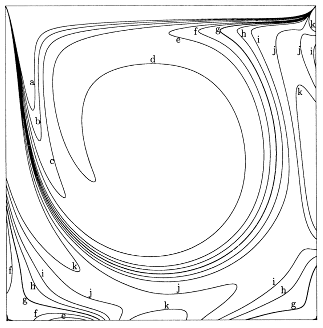


Figure 7.2: Plot of the reference iso-vorticity lines for a lid-driven cavity flow, as per Botella and Peyret. Taken from [92, p. 428].

Two sets of simulations are performed. First, the horizontal component of the velocity along the vertical center line is compared to the reference values obtained by Botella and Peyret. In the second set, the vorticity at the center point of the domain will be compared to the reference values by Botella and Peyret.

7.1.1. Comparison of horizontal velocity along vertical center line

For the first set of simulations, the lattice grid density is varied between runs, using $L = 32\Delta x$, $L = 64\Delta x$, $L = 128\Delta x$ and $L = 256\Delta x$. The simulations are performed for both the compressible BGK and Cumulant MRT.

Figure 7.3 shows the horizontal velocity profile along the vertical center line for the BGK simulations, as well as the profile found in the reference data from Botella and Peyret [92]. Figure 7.4 compares those from the Cumulant MRT simulations and the reference data.

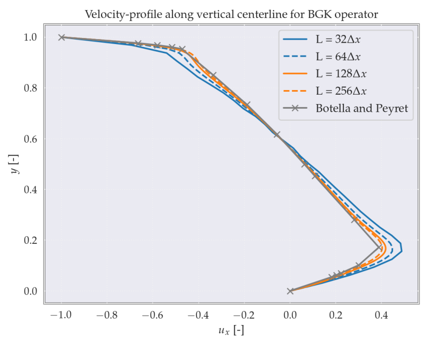


Figure 7.3: Plot of the horizontal velocity component along $x = L/2$, at $t = 200L/u_0$, for various grid refinement levels, using the BGK operator. For the purpose of plotting, $u_0 = 1.0$ and $L = 1.0$.

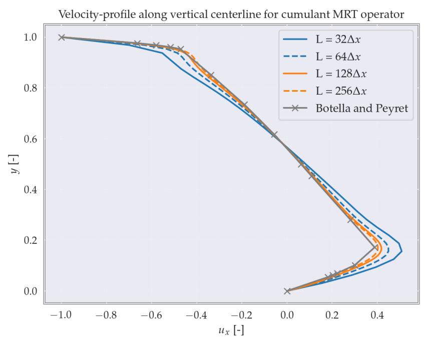


Figure 7.4: Plot of the horizontal velocity component along $x = L/2$, at $t = 200L/u_0$, for various grid refinement levels, using the cumulant MRT scheme. For the purpose of plotting, $u_0 = 1.0$ and $L = 1.0$.

It can be observed that both collision operators result in a velocity profile that approximates the references values by Botella and Preyet more closely as the number of lattice nodes is increased. Furthermore, the velocity conditions on the boundary at $y = 0$ and $y = L$ are met exactly.

7.1.2. Comparison of vorticity at center point

The vorticity at the center of the domain ($x = L/2$ and $y = L/2$) has also been examined and compared to the reference value published by Botella and Preyet. This vorticity has been evaluated by approximating the first-order derivatives in

$$\omega = \frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x} \tag{7.1}$$

numerically using central finite difference schemes. In particular, as the number of fluid nodes was always odd in both the horizontal and vertical direction, there was always a node $\boldsymbol{x}_{\bar{i},\bar{j}}$ that coincided with the exact center of the domain. Thus, the central difference approximation simply became

$$\frac{\partial u_x}{\partial y} \approx \frac{u_x\left(\boldsymbol{x}_{\bar{i},\bar{j}+1}\right) - u_x\left(\boldsymbol{x}_{\bar{i},\bar{j}-1}\right)}{2\Delta x} \tag{7.2}$$

$$\frac{\partial u_y}{\partial x} \approx \frac{u_x\left(\boldsymbol{x}_{\bar{i}+1,\bar{j}}\right) - u_x\left(\boldsymbol{x}_{\bar{i}-1,\bar{j}}\right)}{2\Delta x}, \tag{7.3}$$

$$\tag{7.4}$$

where Δx is the physical grid spacing. Equation (7.1) may then be easily evaluated and compared to the reference value published by Botella and Preyet. The resulting error as function of grid refinement is shown in Figure 7.5. Clearly, the order of accuracy of both collision operators is approximately 1, whereas one would normally expect second order of accuracy for the LBM [2]. This is attributed to the treatment of the top left and top right corners for this specific validation case. In the used boundary treatment, the grid nodes coincide with the boundaries of the domain, which means that grid nodes coincide with the corners of the domain.

However, since the top edge has a horizontal velocity of $-u_0$ whilst the right and left edges are stationary, the corners of the domain form a singularity since it implies that the velocity at those corners should both be at

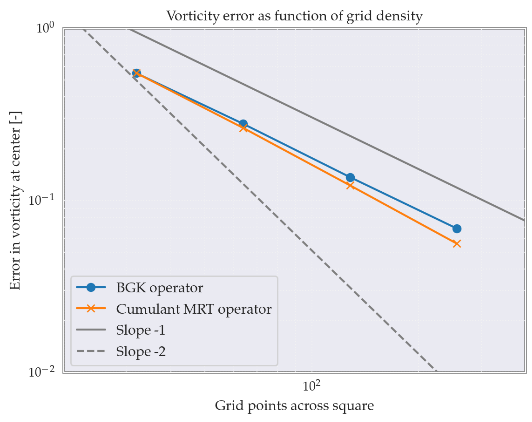


Figure 7.5: Plot of the error in the vorticity at the center of domain ($x = L/2$, $y = L/2$), as function of grid density, for the BGK and cumulant MRT operators.

rest and have a horizontal velocity of $-u_0$. This modelling error likely reduces the order of accuracy to 1 [1]. It should be noted that the validation case in Section 7.2 does show second-order accuracy, confirming that the first-order accuracy here may be attributed to a modelling singularity in the boundary conditions rather than a possible error in the fluid solver itself.

7.2. Cylinder immersed in Taylor-Green vortex flow

The second validation case considered is a Taylor-Green vortex flow immersed by a solid cylinder, with the velocity at the solid boundary prescribed by the analytical solution known for the Taylor-Green vortex. This validation case has previously been used by Kang and Hassan [35] and Wu and Shu [93] to validate various IB-LBM variations.

It should be noted that, contrary to all other validation cases listed in Chapter 7 and 8, the prototype Python code was used for this case instead of the C++-based LaBIB-FSI suite. This was done to allow for comparison with the variations on the IIM proposed in Chapter 5. As will be seen throughout this section, these variations did not appear promising compared to the MIIM, which is why they were not included in the LaBIB-FSI suite, and why all validation cases hereafter only include the base IBM and MIIM.

In a Taylor-Green vortex flow in a square domain of $[-L, L] \times [-L, L]$, the velocity field is known to equal

$$u_x = -u_0 \cos\left(\frac{\pi}{L}x\right) \sin\left(\frac{\pi}{L}y\right) \exp\left(-2\nu\left(\frac{\pi}{L}\right)^2 t\right) \tag{7.5}$$

$$u_y = -u_0 \cos\left(\frac{\pi}{L}x\right) \sin\left(\frac{\pi}{L}y\right) \exp\left(-2\nu\left(\frac{\pi}{L}\right)^2 t\right) \tag{7.6}$$

$$p = p_0 - \frac{u_0^2}{4}\left[\cos\left(\frac{2\pi}{L}x\right) + \cos\left(\frac{2\pi}{L}y\right)\right] \exp\left(-4\nu\left(\frac{\pi}{L}\right)^2 t\right). \tag{7.7}$$

To assess the accuracy of a boundary treatment, simulations are run with an embedded circle of radius $R = 0.5L$ located at the center of the domain. At the boundary of the embedded circle, the velocity is prescribed by Equation (7.5)-(7.6). At the edges of the domain, the boundary conditions are prescribed by Equation (7.5)-(7.7); the initial condition is prescribed by the same equations evaluated at $t = 0$. In each of the coming tests, the solution field is evaluated at $t = L/u_0$. The Reynolds number (based on the velocity u_0 and domain-size L) is held constant at $\mathrm{Re} = 10$.

For illustrative purposes, a vorticity field corresponding to such a flow is shown in Figure 7.6. The domain is discretised by a grid consisting of N-fluid nodes in the horizontal and vertical direction.

The accuracy of the simulation is evaluated by approximating the L_2-error in the velocity in the interior of the

[1]For the purposes of performing a simulation, the velocity at the corners was set to 0 in the end, but this is clearly an arbitrary choice.

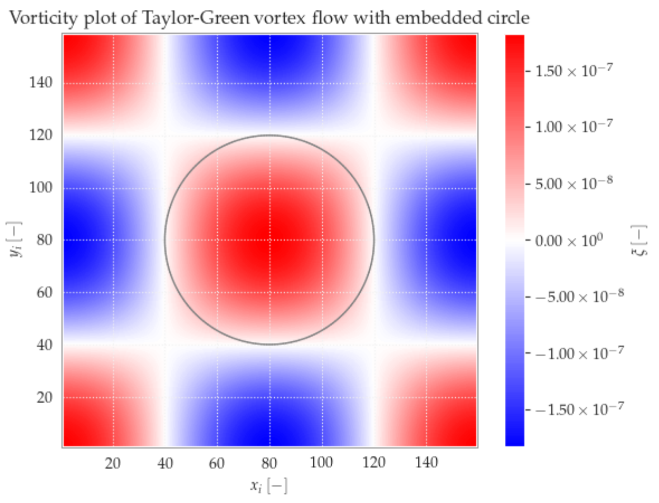


Figure 7.6: Plot of the vorticity at $t = L/u_0$, using a grid spacing of $L = 40\Delta x$, simulated by the IIM-BASE scheme.

domain, i.e.

$$||\epsilon||_2 = \sqrt{\int_\Omega ||\boldsymbol{u}^{(a)} - \boldsymbol{u}^{(n)}||} \approx \sqrt{\frac{1}{N_c u_0^2} \sum ||\boldsymbol{u}^{(a)} - \boldsymbol{u}^{(n)}||^2}, \tag{7.8}$$

where the exact integral is approximated using the equidistant grid points at which the numerical solution $\boldsymbol{u}^{(n)}$ is available, and where $\boldsymbol{u}_x^{(a)}$ is given by Equations (7.5)-(7.6), $\boldsymbol{u}^{(n)}$ is the numerical solution, only points located within the circle are included in the summation, and N_c is the number of points located within the circle.

Three sets of simulations will be evaluated. The first set will demonstrate the overall order of accuracy of the IIM and its variations introduced in Chapter 5. The second set will show the effect of the density of the Lagrangian markers on the boundary. The third set will evaluate the effect of the number of iterations for various boundary spacings.

7.2.1. Evaluation of order of accuracy

For the first set of simulations, the lattice grid density is varied between runs. The simulations are held at a constant relaxation-rate of $\omega_\nu = 1/0.65$, analogous to the value used by Kang and Hassan [35] and Wu and Shu [93]. Furthermore, the Lagrangian markers on the discretised cylinder are placed at a spacing of $\Delta s/\Delta x \approx 1$. This is coarser than used by Kang and Hassan [35] and Wu and Shu [93], but the choice for this spacing is justified in Section 7.2.2. The lattice grid spacing varies between $L = 10\Delta x$, $L = 20\Delta x$, $L = 40\Delta x$ and $L = 80\Delta x$ grid points across the immersed cylinder. As a baseline, the same simulations are performed without the immersed cylinder, once for the cumulant MRT operator, once for the BGK operator.

The results are shown in Figure 7.7, where BASE-CUMMRT and BASE-BGK refer to simulation runs using the CUMMRT and BGK operator, without the immersed body present. A number of conclusions may be drawn from Figure 7.7.

First off all, it can be noted that the CUMMRT operator appears to have a higher order of accuracy than the BGK operator - the former appears to be slightly larger than 3, whereas the latter only appears to have an order of accuracy of slightly larger than 2. Furthermore, the IBM and all implementations of the IIM appear to have second order accuracy. IIM-MOD-A, which is the configuration that evaluates the feedback force and jumps at the midpoint of a time-step, appears the most accurate, particularly at coarser meshes. However, its advantage over other methods appears to falter as the mesh is refined.

7.2.2. Evaluation of the boundary force density error

As the velocity of the boundary of the cylinder is prescribed by Equation (7.5)-(7.7) and should therefore match the exact solution, the feedback force at the Lagrangian markers is also expected to 0. To evaluate its

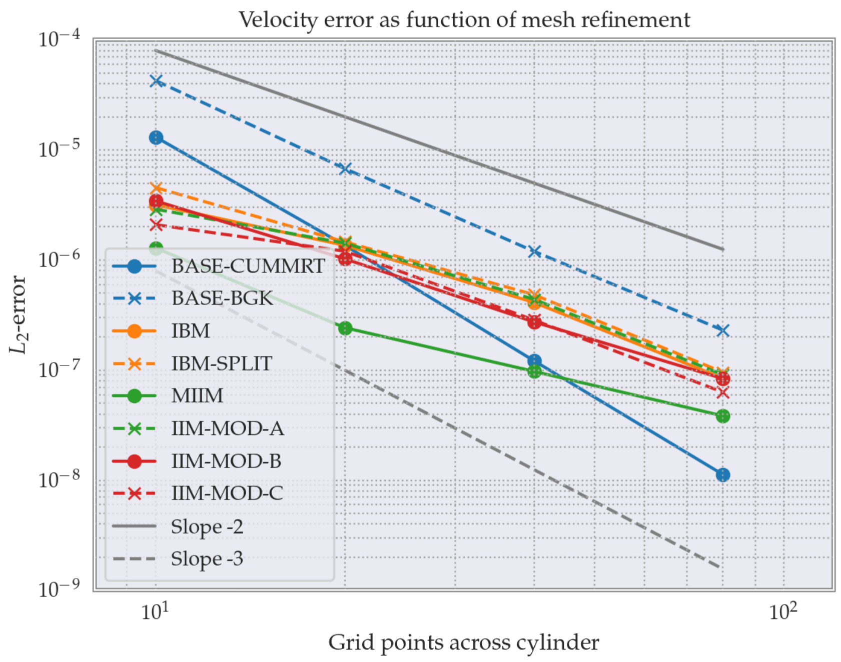


Figure 7.7: Plot of L_2-error as function of number of grid refinement for a constant Lagrangian marker density.

error, the following quantity is computed:

$$\epsilon_{\boldsymbol{F}} = \sum_{\kappa}^{K} \frac{\boldsymbol{F}_\kappa \cdot \boldsymbol{F}_\kappa}{K} \tag{7.9}$$

where K is the total number of boundary markers.

This quantity, as well as the previously defined L_2-error (Equation (7.8)) is evaluated at a constant fluid mesh refinement, using $L = 20\Delta x$, for a varying degree of boundary marker densities, varying between $K = 40$ (which corresponds to $\Delta s/\Delta x \approx 1.57$) and $K = 200$ (corresponding to $\Delta s/\Delta x \approx 0.314$). This allows for an assessment of the influence of the boundary refinement compared to the fluid mesh refinement.

The L_2-error as function of boundary refinment for the various IBM/IIM-approaches is shown in Figure 7.8; the $\epsilon_{\boldsymbol{F}}$-error is shown in Figure 7.9. Note that in both Figure 7.8 and 7.9, the IBM, IIM-MOD-C and IIM-MOD-D configurations were unable to produce results when the number of Lagrangian markers was increased beyond $K = 120$.

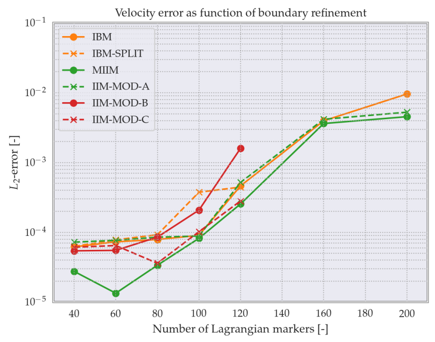


Figure 7.8: Plot of L_2-error as function of the number of Lagrangian markers, for $L = 20\Delta x$.

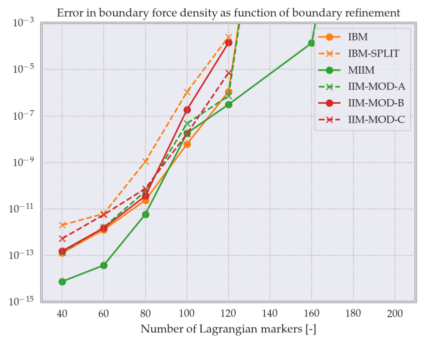


Figure 7.9: Plot of $\epsilon_{\boldsymbol{F}}$-error as function of the number of Lagrangian markers, for $L = 20\Delta x$.

It is evident that for all configurations, when the number of Lagrangian markers is increased beyond approximately $K = 60$ (corresponding to $\Delta s/\Delta x \approx 1.04$), the accuracy starts to decrease notably, with configurations being either unable to provide a solution at all (IBM, IIM-MOD-C and IIM-MOD-D) or experiencing blow-up in the boundary force density variation for $K > 120$.

This results appears contradictory with some existing literature. For example, Peskin [72], in his paper describing the mathematical framework of the IBM that he originally proposed in [4], stated that "to avoid leaks, we impose the restriction that" $\Delta s/\Delta x < 1/2$, without a clear justification of this limit. Other authors, such as Huang and Tian [34] and Krüger [2] have since used Peskin's paper as a guideline, without providing further evidence for this limit. In fact, Krüger [2] concluded, based on a mesh spacing sensitivity analysis for a Poiseuille flow, that decreasing the mesh size below $\Delta s/\Delta x < 1$ does not provide significant benefit.

Based on this sensitivity study, it was previously decided in Section 7.2.1 to use a Lagrangian marker density of $\Delta s/\Delta x \approx 1$.

7.2.3. Evaluation of iterative solvers

In the previous sections, only the implicit solvers have been evaluated. However, it remains interesting to evaluate the effect of the number of iterations in multi-direct forcing based approaches. To do so, the iterative solvers have been run on a fixed fluid mesh ($L = 20\Delta x$), at a variety of boundary mesh densities ($K = 60$, $K = 90$ and $K = 120$, corresponding to $\Delta s/\Delta s \approx 1.04$, $\Delta s/\Delta s \approx 0.70$, $\Delta s/\Delta s \approx 0.52$), for various number of iterations per time-step (ranging between $N_{\text{iter}} = 5$ and $N_{\text{iter}} = 1000$.

To investigate the convergence behaviour of the iterative scheme, two quantities are evaluated. First, the the boundary force density is compared to the solution found by the corresponding implicit solver and evaluated as

$$\Delta\epsilon_{\boldsymbol{F}} = \sum_{\kappa}^{K} \frac{\left(\boldsymbol{F}_{\kappa}^{\text{MDF}} - \boldsymbol{F}_{\kappa}^{\text{implicit}}\right) \cdot \left(\boldsymbol{F}_{\kappa}^{\text{MDF}} - \boldsymbol{F}_{\kappa}^{\text{implicit}}\right)}{K}, \tag{7.10}$$

where F_{κ}^{MDF} is the boundary forcing density found by the MDF-algorithm, and $F_{\kappa}^{\text{implicit}}$ the boundary forcing density found by the corresponding implicit algorithm.

Secondly, the difference in L_2-error is evaluated as

$$\Delta\,||\epsilon||_2 = ||\epsilon||_2^{\text{MDF}} - ||\epsilon||_2^{\text{implicit}}, \tag{7.11}$$

where $||\epsilon||_2^{\text{MDF}}$ is the L_2-error for the iterative solution, found via Equation (7.8), $||\epsilon||_2^{\text{implicit}}$ the L_2-error for the corresponding implicit solution. These differences are plotted in Figure 7.10, 7.12 and 7.14, and 7.11, 7.11 and 7.15, respectively.

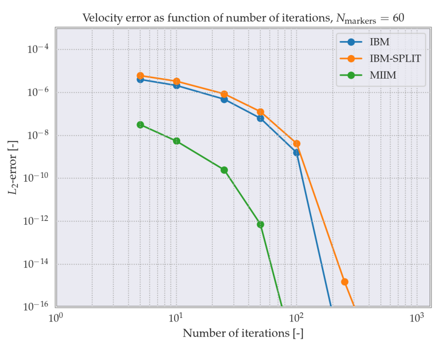


Figure 7.10: Plot of L_2-error as function of the number of iterations, for a number of markers equal to $K = 60$.

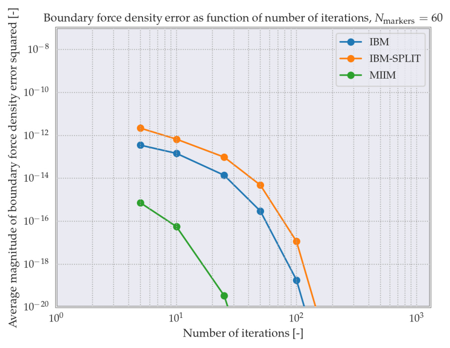


Figure 7.11: Plot of $\epsilon_{\boldsymbol{F}}$-error as function of the number of iterations, for a number of markers equal to $K = 60$.

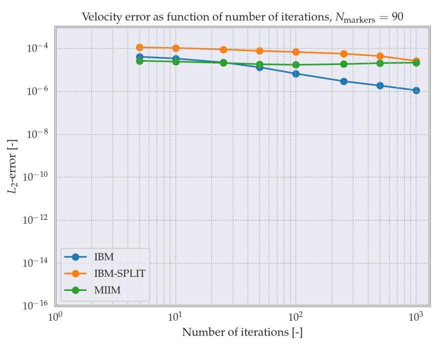


Figure 7.12: Plot of L_2-error as function of the number of iterations, for a number of markers equal to $K = 90$.

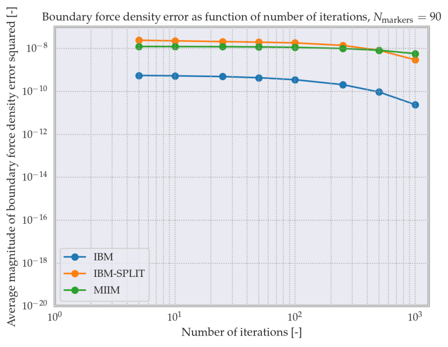


Figure 7.13: Plot of ϵ_F-error as function of the number of iterations, for a number of markers equal to $K = 90$.

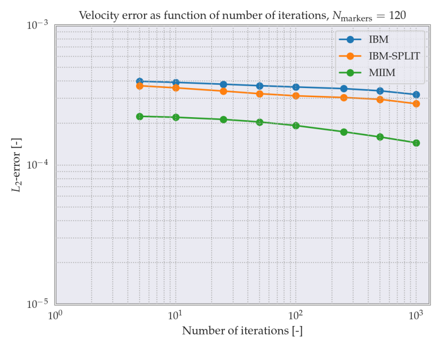


Figure 7.14: Plot of L_2-error as function of the number of iterations, for a number of markers equal to $K = 120$.

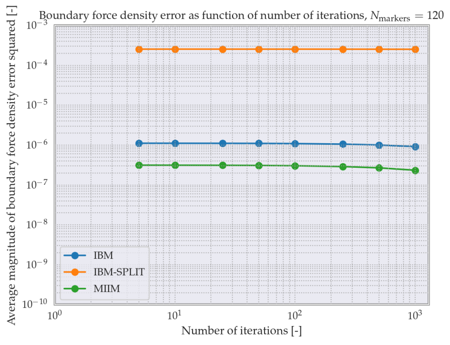


Figure 7.15: Plot of ϵ_F-error as function of the number of iterations, for a number of markers equal to $K = 120$.

From Figure 7.11-7.14, it is evident that for both the IBM and IIM-based approaches, the number of iterations show clearly convergent behaviour for $K = 60$, corresponding to $\Delta s / \Delta x \approx 1.04$. However, for finer mesh densities, the convergence rate detoriates quickly, and it appears to carry very little benefit to increase the boundary mesh density. This is likely caused by the fact that when the boundary mesh density is increased whilst the fluid grid density is held constant, the number of degrees of freedom on the boundary mesh starts to become greater than the number of degrees of freedom on the fluid grid. The solution space (which lives on the boundary mesh) may therefore contain the eigenvectors corresponding to the zero eigenvalues, resulting in a slow iterative process to obtain the same solution as the implicit system.

7.3. Cylinder and rigid flag immersed in horizontal cylinder flow

The third set of validation cases considered is the set of benchmarks proposed by Turek and Hron [8], which considers the flow around a cylinder with a flag attached to its trailing edge, as shown in Figure 7.16 and 7.17. To be specific, in the benchmark cases CFD1, CFD2 and CFD3, the attached beam is treated as a rigid object, and the lift- and drag coefficient is measured across three different Reynolds numbers.

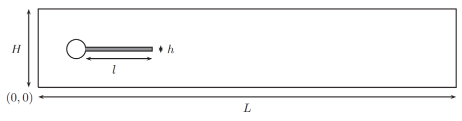


Figure 7.16: Definition of computational domain for the benchmark cases CFD1, CFD2 and CFD3 proposed by Turek and Hron [8], taken from [8, p. 249].

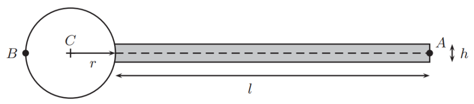


Figure 7.17: Sketch of the geometrical parameters for the bench mark cases CFD1, CFD2 and CFD3 proposed by Turek and Hron [8], taken from [8, p. 249].

The velocity at left and right boundary of the domain is prescribed by

$$u(0, y) = u(L, y) = \frac{3}{2} u_0 \frac{y(y-H)}{4H^2}. \tag{7.12}$$

The horizontal walls are treated with a no-slip condition on the velocity. The modified regularised boundary conditions were used for each boundary.

The relevant geometric parameters are given in Table 7.1; the relevant fluid parameters for the different benchmark cases are given in Table 7.2. The Reynolds number has been defined with respect to the velocity and diameter of the cylinder. The Mach number was set to 0.04 for all runs.

Table 7.1: Geometric parameters corresponding to CFD1-CFD3.

Parameter	Symbol	Value [m]
Length	L	2.5
Height	H	0.41
Cylinder position	C	$(0.2, 0.2)$
Cylinder radius	r	0.05
Flag length	l	0.35
Flag height	h	0.02

Table 7.2: Fluid parameters corresponding to CFD1-CFD3.

Parameter	CFD1	CFD2	CFD3
ρ^f	$1 \times 10^3\,\mathrm{kg/m^3}$	$1 \times 10^3\,\mathrm{kg/m^3}$	$1 \times 10^3\,\mathrm{kg/m^3}$
ν^f	$1 \times 10^{-3}\,\mathrm{m^2/s}$	$1 \times 10^{-3}\,\mathrm{m^2/s}$	$1 \times 10^{-3}\,\mathrm{m^2/s}$
u_0	$0.2\,\mathrm{m/s}$	$1\,\mathrm{m/s}$	$2\,\mathrm{m/s}$
Re	20	100	200

The domain is discretised by a grid consisting of N_x-fluid nodes in the horizontal direction, and N_y-fluid nodes in the vertical direction.

7.3.1. Evaluation of CFD1 & CFD2 benchmark - comparison of force coefficients

The CFD1 and CFD2 benchmark case both result in steady-state solutions. Both simulations are therefore simulated by setting the initial condition to be

$$u\left(x, y\right) = \frac{3}{2} u_0 \frac{y\left(y - H\right)}{4H^2}, \tag{7.13}$$

and then taking measurements at $t = 2L/u_0$, to allow for sufficient time for convergence to the steady state. The simulations are performed for the IBM and the MIIM configurations, at three levels of grid-refinement, ranging from $R = 5\Delta x$ to $R = 20\Delta x$. The multi-direct-forcing schemes use 25 iterations for every time step. In Table 7.3, the resulting lift and drag coefficient for each solver is shown.

Table 7.3: Lift and drag for the CFD1 and CFD2 benchmark cases from Turek and Hron [8].

Configuration	Grid size	CFD1		CFD2	
		F_x [N]	F_y [N]	F_x [N]	F_y [N]
IBM	$R = 5\Delta x$	15.60	1.108	155.85	7.740
	$R = 10\Delta x$	14.80	1.076	144.96	8.672
	$R = 20\Delta x$	14.49	1.070	140.27	8.909
MIIM	$R = 5\Delta x$	15.40	1.101	151.77	7.951
	$R = 10\Delta x$	14.69	1.079	141.97	8.764
	$R = 20\Delta x$	14.43	1.079	138.76	9.163
Target [8]	-	14.29	1.119	136.7	10.53

As can be seen from Table 7.3, the results all seem to approximate the reference values by Turek and Hron [8] relatively well, attaining an accuracy similar to those obtained by e.g. Geller et al. [80] and Trapani et al. [94] who also made use of LBM-based solvers. However, it does appear that the MIIM is slightly more accurate at the same level of grid discretisation as the IBM.

7.3.2. Evaluation of CFD1 & CFD2 benchmark - comparison of force distributions along cylinder

The boundary force distribution along the cylinder for the CFD1 and CFD2 benchmarks has also been evaluated. The horizontal component of the boundary force density is plotted as function of polar angle in Figure 7.18 and 7.19 for CFD1 and CFD2, respectively. Here the polar angle is 0° at the trailing edge of the cylinder and then is measured in counterclockwise direction. The vertical component is plotted in Figure 7.20 and 7.21 respectively, and the magnitude of it is plotted in Figure 7.22 and 7.23. The results are plotted for meshes with a refinement of $R = 20\Delta x$. Note that the boundary force from the solid on the fluid is computed (as this is the direct output from the IBM/MIIM), and not the force exerted by the fluid on the solid (which would be equal in magnitude but opposite in sign).

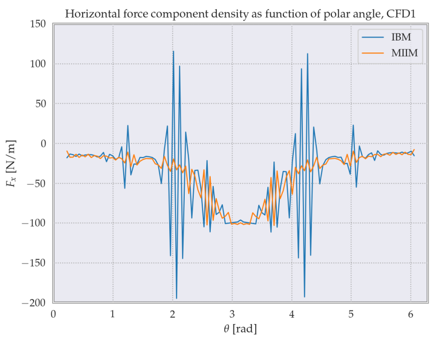


Figure 7.18: Horizontal force component density as function of polar angle, for the IBM and MIIM, for CFD1.

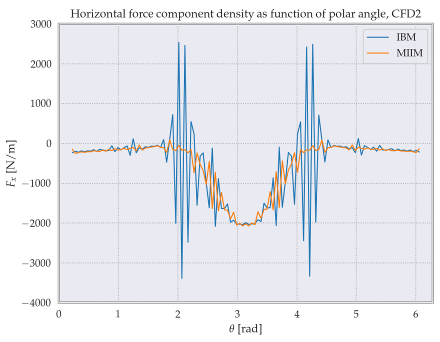


Figure 7.19: Horizontal force component density as function of polar angle, for the IBM and MIIM, for CFD2.

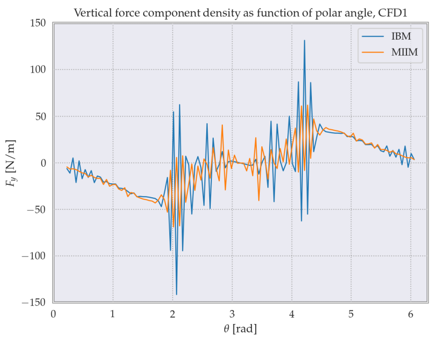


Figure 7.20: Vertical force component density as function of polar angle, for the IBM and MIIM, for CFD1.

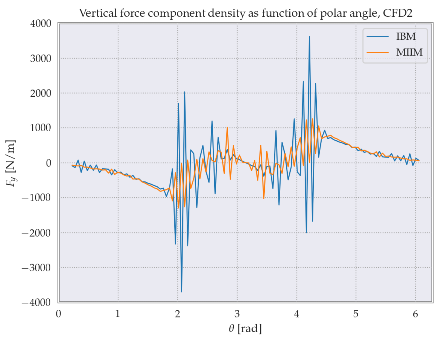


Figure 7.21: Vertical force component density as function of polar angle, for the IBM and MIIM, for CFD2.

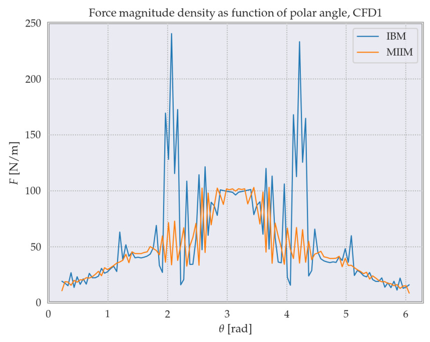


Figure 7.22: Force magnitude density as function of polar angle, for the IBM and MIIM, for CFD1.

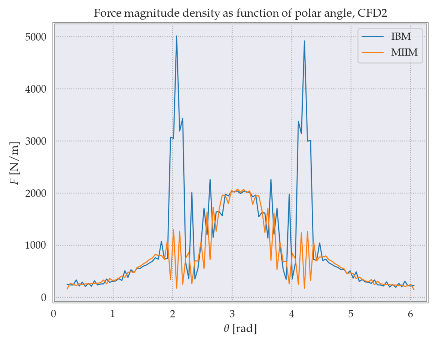


Figure 7.23: Force magnitude density as function of polar angle, for the IBM and MIIM, for CFD2.

It is evident that the general shape of the distribution matches between the IBM and MIIM configurations for all plots. However, it is evident that for all plots, the MIIM seems to suffer significantly less from the oscillations than the IBM. These results demonstrate that the approach of the MIIM to include the boundary forcing during streaming rather than during collision, may benefit the overall performance of the solver, resulting in smoother solutions in the boundary forcing.

To provide further illustration to the behaviour of the boundary force distributions, it is interesting to compare the distributions with streamline plots of the flow around the cylinder and flag. These plots are shown in Figure 7.24 and 7.25, where the streamlines are superimposed on a contourplot of the velocity magnitude. The plots are based on the run using the MIIM-configuration at a mesh discretisation such that $R =$

$20\Delta x$.

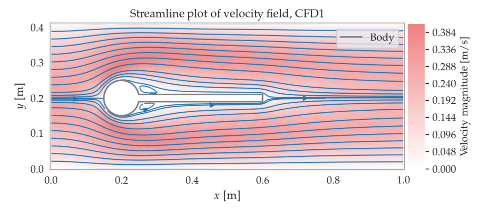


Figure 7.24: Streamline plot of the flow around the cylinder and flag for the MIIM, for CFD1, at a mesh resolution such that $R = 20\Delta x$.

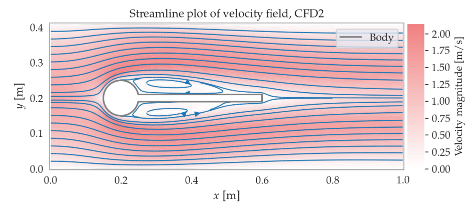


Figure 7.25: Streamline plot of the flow around the cylinder and flag for the MIIM, for CFD2, at a mesh resolution such that $R = 20\Delta x$.

In Figure 7.18-7.23 it can be seen that the strongest oscillations in the boundary force density (for both the IBM and MIIM) occurred around $\theta = 3/4\pi$ and $\theta = 5/4\pi$ ($\theta = 0$ corresponds to the trailing edge, and is then measured in counterclockwise direction). This corresponds to the north-west and south-west areas of the cylinder, where the incoming flow needs to be deflected 45° up-/downwards. It is speculated that this velocity change (that is not parallel to the incoming flow) can amplify the oscillations in the boundary forcing - it may be tricky for the boundary treatment method to simultaneously reduce the flow velocity around the boundary to a small value and change the velocity direction.

Note that in contrast, the region with flow separation (which is particularly clearly present for CFD2), i.e. the region around $\theta = \pi/4$ and $\theta = 7/8\pi$, are generally well quite well behaved and do not suffer from large oscillations. This is likely due to the low velocity and small velocity gradients that exists in this region anyway.

7.3.3. Evaluation of CFD1 & CFD2 benchmark - comparison of force distributions along flag

The force distributions along the flag were also evaluated. In particular, the trailing edge of the flag is of interest, as it is a short section (relative to the diffusive width of the interpolating schemes) with two corners close to each other (with a corner likely involving very large gradients in the boundary force distributions). If s represents the parametric coordinate describing the flag, with $s = 0$ located at the midpoint of the trailing edge, then with a flag thickness of $h = 0.02\,\text{m}$, the corners of the flag are located at $s = 0.01\,\text{m}$ and $s = -0.01\,\text{m}$.

The horizontal component of the boundary force density is then plotted as function of parametric coordinate in Figure 7.26 and 7.27 for CFD1 and CFD2, respectively. The vertical component is plotted in Figure 7.28 and 7.29 respectively, and the magnitude of it is plotted in Figure 7.30 and 7.31. The results are plotted for meshes with a refinement of $R = 20\Delta x$.

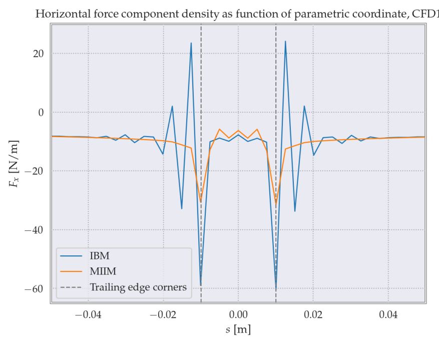


Figure 7.26: Horizontal force component density as function of parametric coordinate, for the IBM and MIIM, for CFD1.

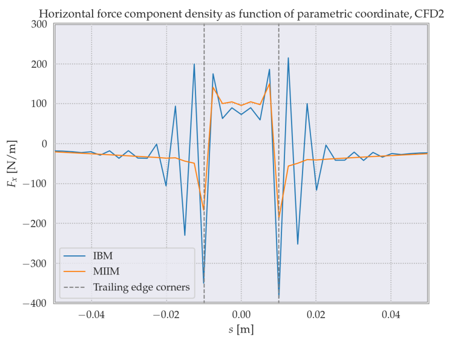
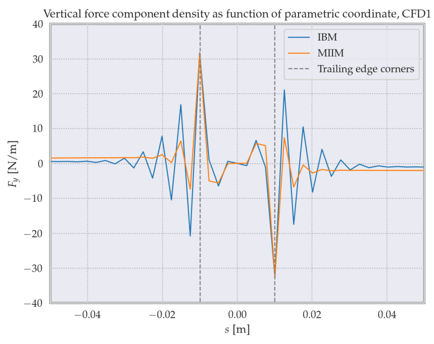


Figure 7.27: Horizontal force component density as function of parametric coordinate, for the IBM and MIIM, for CFD2.



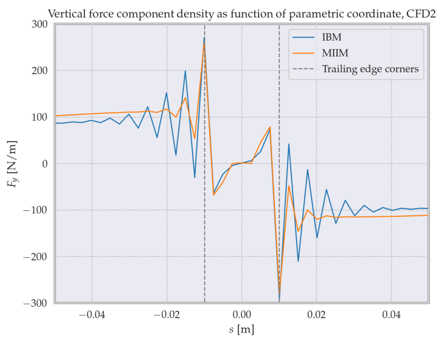
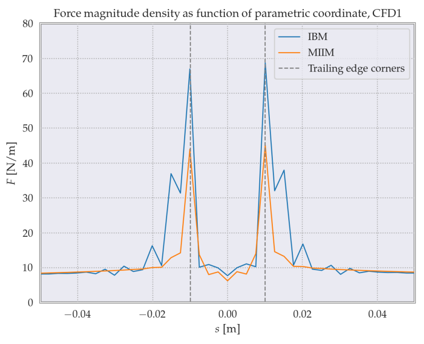


Figure 7.28: Vertical force component density as function of parametric coordinate, for the IBM and MIIM, for CFD1.

Figure 7.29: Vertical force component density as function of parametric coordinate, for the IBM and MIIM, for CFD2.



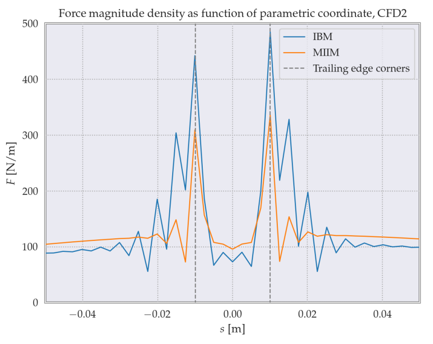


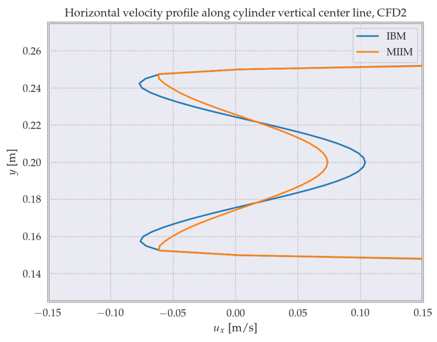
Figure 7.30: Force magnitude density as function of parametric coordinate, for the IBM and MIIM, for CFD1.

Figure 7.31: Force magnitude density as function of parametric coordinate, for the IBM and MIIM, for CFD2.

Again, it is evident that the general shape of the forcing distribution is similar between the IBM and MIIM. However, although the MIIM still shows some oscillations in its boundary force distributions at the corner locations, they appear to dampen out far more quickly as one moves away from the corner points compared to the IBM. This is clearly a great benefit to the MIIM, as it appears better capable of smoothing out any oscillations that do arise in its solution field.

7.3.4. Evaluation of CFD1 & CFD2 benchmark - comparison of velocity profiles

The velocity and density profile along the vertical axis crossing through the center of the cylinder has been evaluated to assess the amount of inflow within the cylinder, which ideally should be as close to 0 as possible. The velocity profile for IBM and MIIM are shown in Figure 7.32 and 7.33; the density profile is shown in Figure 7.34. These plots were constructed using the finest meshes ($R = 20\Delta x$) for both boundary treatments. Note that the bottom and top of the cylinder are located at $y = 0.15\,\text{m}$ and $y = 0.25\,\text{m}$, respectively.



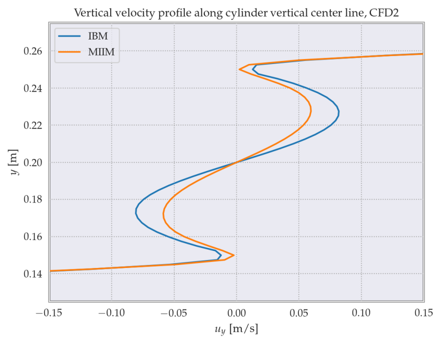


Figure 7.32: Horizontal velocity profile along the vertical axis crossing through the center of the cylinder, for CFD2.

Figure 7.33: Vertical velocity profile along the vertical axis crossing through the center of the cylinder, for CFD2.

It can be seen that the MIIM is notably more successful than the IBM at reducing the amount of inflow inside

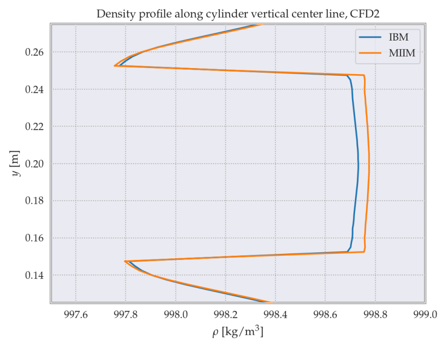


Figure 7.34: Density profile along the vertical axis crossing through the center of the cylinder, for CFD2.

the cylinder - the IBM predicts a notably larger velocity magnitude inside the cylinder, both near the edge of the cylinder at $y = 0.15\,\mathrm{m}$ and $y = 0.25\,\mathrm{m}$ as well as at the center of the cylinder. Furthermore, as noted in Table 7.2, the imposed initial density is equal to $1 \times 10^3\,\mathrm{kg/m^3}$ in the entire domain. Due to the pressure loss over a channel flow and the assumption of an ideal, isothermal gas, the density drops (as the inlet is set equal to a $1 \times 10^3\,\mathrm{kg/m^3}$ density) over the length of the channel. However, the density inside the cylinder should ideally stay as close to $1 \times 10^3\,\mathrm{kg/m^3}$ as possible, as the flow inside the cylinder should be unaware of the channel flow outside the cylinder. It can be seen from Figure 7.34 that the MIIM sustains a slightly larger density inside the cylinder, indicating that less mass flux has happened across the cylinder's surface.

It should be noted that it may appear in Figure 7.32 that the horizontal velocity component at $y = 0.15\,\mathrm{m}$ and $y = 0.25\,\mathrm{m}$ is significantly different from 0; instead overshooting to at least $u_x = -0.05\,\mathrm{m/s}$. However, it should be noted that this peak is located *inside* the cylinder, and the velocity at the control point at $y = 0.15\,\mathrm{m}$ and $y = 0.25\,\mathrm{m}$ is much closer to 0.

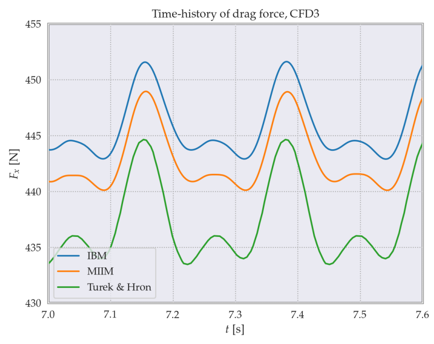
7.3.5. Evaluation of CFD3 benchmark - comparison of force history

The CFD3 benchmark case results in a transient solution. This case was initialised by setting the initial flow velocity to 0 everywhere in the domain, and using an inflow condition of

$$u(0, y) = u(L, y) = \frac{3}{2} u_0 \frac{y(y - H)}{4H^2}, \tag{7.14}$$

multiplied with a linear time-ramp up until $t = 1\,\mathrm{s}$. The simulation is run for $0 \leq t \leq 8\,\mathrm{s}$. The same input parameters were used as for the CFD1 and CFD2 benchmarks, and the simulation was run at mesh discretisations such that $R = 5\Delta x$, $R = 10\Delta x$, $R = 20\Delta x$ and $R = 40\Delta x$.

Plots of the drag and lift as function of time are shown in Figure 7.35 and 7.36, in which the obtained results are also compared to those found by Turek and Hron [8]. Numerical values corresponding to the mean and amplitude of the periodic behaviour of the coefficients is included in Table 7.4. It should be noted that at $R = 5\Delta x$, the simulation resulted a steady-state simulation, and at $R = 10\Delta x$, the solution was troubled by some low-frequency noise occurring still, too.



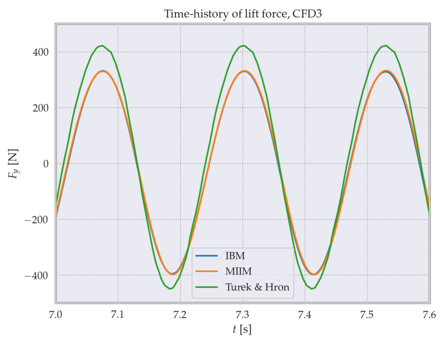


Figure 7.35: Drag as function of time, compared to the reference solution by Turek and Hron [8], using $R = 40\Delta x$.

Figure 7.36: Lift as function of time, compared to the reference solution by Turek and Hron [8], using $R = 40\Delta x$.

Table 7.4: Lift- and drag-coefficient for the CFD3 benchmark cases from Turek and Hron [8].

Configuration	Grid size	C_x		C_y		Period
		Mean value	Amplitude	Mean value	Amplitude	
IBM	$R = 5\Delta x$	471.28	-	2.158	-	-
	$R = 10\Delta x$	469.54	2.00	-35.02	204.20	0.233 s
	$R = 20\Delta x$	456.70	4.000	-35.25	346.72	0.226 s
	$R = 40\Delta x$	447.33	4.431	-33.50	365.51	0.227 s
IIM	$R = 5\Delta x$	473.04	-	-4.094	-	-
	$R = 10\Delta x$	457.37	1.51	-40.03	163.99	0.222 s
	$R = 20\Delta x$	445.77	3.546	-40.20	316.52	0.228 s
	$R = 40\Delta x$	444.53	4.433	-32.42	366.35	0.227 s
Reference [8]	-	439.5	5.618	-11.89	437.8	0.22 s

As is evident from Table 7.4, the CFD3-benchmark appears quite sensitive to the mesh discretisation, and although the results appear to converge reasonably quickly to the target values from Turek and Hron [8], the results at a mesh discretisation of $R = 40\Delta x$ still shows an error of approximately 20% in the amplitude of both force coefficients. This high mesh sensitivity has also been reported by Geller et al. [80] (who also used an LBM-based solver) and Schäfer et al. [95] (who used a finite-volume solver). Therefore, it is assumed that this sensitivity is inherent to this validation case, and it was (due to lack of computational resources available) considered that the trends shown in Table 7.4 are promising enough to not warrant additional simulations at finer grid levels.

8 Validation of 2D fluid-structure solver

In this Chapter, the LaBIB-FSI solver is validated problems involving fluid-structure interaction. Three validation cases are used. Section 8.1 evaluates the solver for an oscillating rigid cylinder in a fluid at rest using reference data from Suzuki and Inamuro [96] and Dütsch et al. [97]. This case validates the FSI-solver for moving boundaries within the fluid solver, but with a rigid structural model. Section 8.2 validates the structural solver in isolation for the CSM3 benchmark by Turek and Hron [8]. Section 8.3 evaluates the solver for the FSI1 & FSI3 benchmark case proposed by Turek and Hron [8], this time involving a two-way coupled fluid-structure interaction problem.

8.1. Rigid cylinder oscillating in a fluid at rest

In this validation case, a cylinder oscillates horizontally in a fluid at rest. The horizontal force coefficient is then evaluated as function of time, and the velocity profile in the region near the cylinder is measured as well.

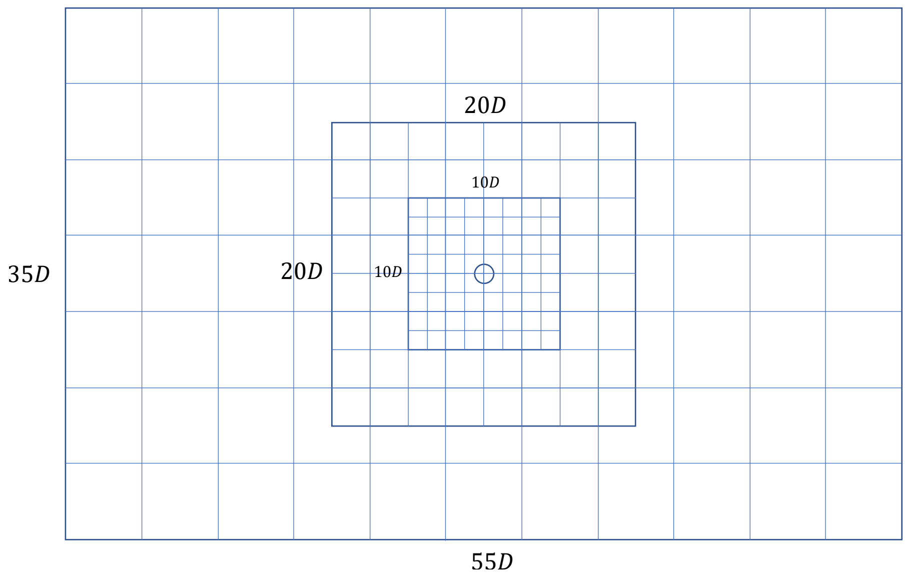


Figure 8.1: Domain and grid discretisation of the flow around and oscillating cylinder in a fluid at rest. A multi-grid is used, with the innermost, finest square having double the grid refinement of its parent grid, which in turn has a twice as fine grid as the background grid that covers the full domain.

To be precise, the cylinder is located at the center of a domain of size $55D \times 35D$ as shown in Figure 8.1, where

D is the diameter of the cylinder, and the horizontal velocity of the cylinder is given by

$$u_{x,\mathrm{cyl}}(t) = -u_{\max}\cos\left(\frac{2\pi}{T}t\right), \tag{8.1}$$

where $u_{\max}$ is the maximum velocity of the cylinder and T the period of the motion. Two non-dimensional parameters are relevant for the solution; the Reynolds number (based on the diameter and the maximum cylinder velocity) and the Keulegan-Carpenter number ($\mathrm{KC} = u_{\max}T/D$, which is a non-dimensional measure of the amplitude of the motion.

A grid with multiple levels of refinement is used, as shown in Figure 8.1. The innermost, finest square having double the grid refinement of its parent grid, which in turn has a twice as fine grid as the background grid that covers the full domain. Two simulations are performed; the coarse simulation has a background grid that is discretised using a grid spacing $\Delta x = D/5$, and the fine simulation has a background grid that is discretised using a grid spacing $\Delta x = D/10$. The Mach number is set to $M = 0.04$ for all runs discussed below.

The edges of the domain in Figure 8.1 are represented by the modified regularized boundary condition with an zero-velocity imposed at all sides. It is acknowledged that ideally, these boundary conditions would have been outflow, but this modelling choice is justified by the fact that the outflow velocity is likely very small due to the large size of the domain compared to the cylinder, and the time-average of the outflow velocity is zero anyway due to the periodic motion.

8.1.1. Added mass effect

As noted by Suzuki and Inamuro [96], it is important to compensate for the added mass effect when using an IBM-based approach, i.e. an approach where the fluid inside the object is still simulated (which also applies to the IIM). After all, the fluid inside the object also needs to move displaced, and the forcing computed by the boundary includes this added mass force. Mathematically, the actual body force $\boldsymbol{F}(t)$ acting on an object can be expressed as

$$\boldsymbol{F}(t) = \boldsymbol{F}_{\mathrm{tot}}(t) + \boldsymbol{F}_{\mathrm{in}}(t), \tag{8.2}$$

where $\boldsymbol{F}_{\mathrm{tot}}(t)$ is the body force obtained directly from the IBM-scheme, i.e.

$$\boldsymbol{F}_{\mathrm{tot}}(t) = -\int_{\Gamma} \boldsymbol{F}(\boldsymbol{s},t)\,d\boldsymbol{s} \approx -\sum_{\kappa} F_{\kappa}(\boldsymbol{X}_{\kappa},t)\,\Delta V_{\kappa}, \tag{8.3}$$

where F_κ is the total Lagrangian force on each boundary marker (as per Equation (4.8)) and ΔV_κ is the volume of the Lagrangian body element, and the sum is taken over the total number of body markers. A negative sign is added as the IBM typically computes the force from the boundary on the fluid, whereas the current interest in the force from the fluid on the boundary. Finally, the internal force $\boldsymbol{F}_{\mathrm{in}}(t)$ is computed as

$$\boldsymbol{F}_{\mathrm{in}}(t) = \rho_f \frac{d}{dt}\int_{\boldsymbol{x}\in\Omega(t)} \boldsymbol{u}(\boldsymbol{x},t)\,d\boldsymbol{x}, \tag{8.4}$$

where $\Omega(t)$ is the domain inside the boundary at time t.

In case of the rigid cylinder with prescribed, translational motion, Equation (8.4) is trivial to compute, and the added mass effect can easily be quantified for; one can simply differentiate Equation (8.1) to find

$$\boldsymbol{F}_{\mathrm{in}}(t) = \rho_f \frac{d}{dt}\int_{\boldsymbol{x}\in\Omega(t)} \boldsymbol{u}(\boldsymbol{x},t)\,d\boldsymbol{x} \tag{8.5}$$

$$= \rho_f \frac{d\boldsymbol{u}(t)}{dt}\int_{\boldsymbol{x}\in\Omega(t)} d\boldsymbol{x} \tag{8.6}$$

$$F_{\mathrm{in},x} = \rho_f u_{\max}\frac{2\pi}{T}\sin\left(\frac{2\pi}{T}t\right)\pi\frac{D^2}{4}. \tag{8.7}$$

This amount is then added to the total mass found from the IBM, and the body force on the object cylinder can be readily computed. Note that since the motion of the cylinder is prescribed, this calculation may be done in whole during post-processing.

8.1.2. Evaluation of force coefficient

In order to compare results with those reported by Suzuki and Inamuro [96], simulations were run at a Reynolds number of 10 and 100, with a constant Keulegan-Carpenter number of 5. For the Reynolds number of 100, results from a numerical experiment by [97] are also available.

The time history of the drag coefficient over 3 periods of motion are then compared to the reference data from Suzuki and Inamuro [96], as shown in Figure 8.2 and 8.3; for Figure 8.3, the result by Dütsch et al. [97] has also been included (note that only the actual body force coefficient is presented by Dütsch et al.). The "corrected" lines correspond to the results where the added mass effect has been accounted for. Corresponding numerical values for the maximum force coefficients (note that the solution is periodic with zero mean, so this describes all information) are shown in Table 8.1.

It is evident that the results between the MIIM and Suzuki and Inamuro [96] match closely - for Re = 10, the resulting error is approximately 5%, whereas for Re = 100 the error is in the order of 10%. However, it should be noted that there is also an error between the results by Suzuki and Inamuro and Dütsch et al. for the case of Re = 100. It is hypothesized that the deviation in Suzuki and Inamuro's results is caused by a discretisation error, as they run their simulation at a mesh discretisation such that $D = 50\Delta x$ at a Mach number of 0.052.

Suzuki and Inamuro did not perform a mesh convergence study, so it is difficulty to evaluate how large this discretisation error exactly is for their results. However, from the results in Table 8.1, it is apparent that for the IBM and MIIM configuration, the discretisation error at a mesh refinement of $D = 40\Delta x$ is still in the order 1% for the Re = 10 and of 5% for Re = 100. Thus, assuming both the LaBIB-FSI solver and the solver by Suzuki and Inamuro have roughly similar mesh refinements, this would explain the deviation between the result by Suzuki and Inamuro and Dütsch et al.

Indeed, the results for the IBM and MIIM configurations appear to converge to the results by Dütsch et al. rather than those by Suzuki and Inamuro for the value of $F_{x_{\max}}$ for Re = 100. Based on the results for the mesh refinement levels shown in Table 8.1, the observed order accuracy p_0 may be easily computed as [98]

$$p_0 = \frac{\ln\left(\frac{F_{x_{\max}}(D=20\Delta x) - F_{x_{\max}}(D=40\Delta x)}{F_{x_{\max}}(D=40\Delta x) - F_{x_{\max}}(D=80\Delta x)}\right)}{\ln(2)}, \tag{8.8}$$

resulting in an observed order of accuracy of 0.8 for both the IBM and MIIM. From this, the converged solution may be estimated as

$$F_{x_{\max,\mathrm{con}}} = F_{x_{\max}}(D = 80\Delta x) + \frac{F_{x_{\max}}(D = 80\Delta x) - F_{x_{\max}}(D = 40\Delta x)}{2^1 - 1}, \tag{8.9}$$

resulting in an estimate of the converged horizontal force coefficient of $F_{x_{\max,\mathrm{con}}} = 3.237$ for the IBM, and $F_{x_{\max,\mathrm{con}}} = 3.284$ for the MIIM, which both are very close to the reference value by Dütsch et al. [97].

This remaining minor difference may be caused by a variety of factors. The solution may not be not perfectly asymptotic yet at this level of mesh refinement (causing an inaccuracy in the computation procedure above). Furthermore, a discretisation error due to the Mach number being fixed at $M = 0.04$ is still present (as ideally the Mach number would be 0). Thirdly, a discretisation error of the boundary surface when placing the Lagrangian markers may contribute. Finally, an interpolation error is present when interpolating and diffusing from and back to the fluid grid in the IBM and MIIM, which may contribute as well, as well as an iteration error as a finite number of iterations is used in the multi-direct forcing scheme.

The deviation between the IBM and MIIM configurations (and the result by Dütsch et al., for that matter) compared to the results by Suzuki and Inamuro may be the result of several factors. The use of a different collision operator (Suzuki and Inamuro used a BGK-collision operator, whereas the IBM and MIIM configurations used a CUMMRT-operator), the use of a multigrid for the IBM and MIIM configuration and different boundary conditions at the edges of the domain may all contribute to the difference between the obtained results and those by Suzuki and Inamuro. Finally, the results by Suzuki and Inamuro are at a mesh discretisation such that $\Delta = 50\Delta x$, which likely still has a considerable spatial discretisation error, as shown by the results from the IBM and MIIM. Unfortunately, Suzuki and Inamuro only provided results for a single mesh refinement level, which makes it difficult to estimate how large their discretisation error was.

Finaly, it is observed from Table 8.1 that the MIIM appears to approximate the target results better than the IBM, having a smaller error compared to Dütsch et al. than the IBM at the same grid level. Furthermore, it is confirmed that the LaBIB-FSI solver is capable of handling FSI-problems involving a moving, rigid body.

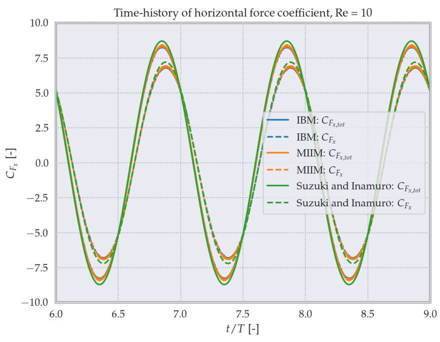


Figure 8.2: Drag as function of time, compared to the reference solution by Suzuki and Inamuro [96], for Re = 10 and $D = 80\Delta x$.

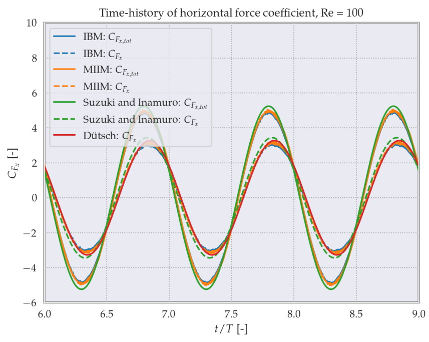


Figure 8.3: Drag as function of time, compared to the reference solution by Suzuki and Inamuro [96], for Re = 100 and $D = 80\Delta x$.

Table 8.1: Amplitude of horizontal force coefficient for a cylinder oscillating horizontally in a fluid at rest, compared to reference data from Suzuki and Inamuro [96] and Dütsch et al., [97].

Configuration	Configuration	Re = 10		Re = 100	
		$Fx_{\max}$ [N]	$Fx_{,\mathrm{tot}_{\max}}$ [N]	$Fx_{\max}$ [N]	$Fx_{,\mathrm{tot}_{\max}}$ [N]
IBM	$D = 20\Delta x$	6.614	8.030	2.579	4.409
	$D = 40\Delta x$	6.729	8.185	2.867	4.688
	$D = 80\Delta x$	6.827	8.303	3.029	4.880
MIIM	$D = 20\Delta x$	6.730	8.201	2.802	4.613
	$D = 40\Delta x$	6.818	8.292	3.002	4.851
	$D = 80\Delta x$	6.887	8.364	3.119	4.957
Target [96]	$D = 50\Delta x$	7.20	8.710	3.440	5.240
Target [97]	-	-	-	3.271	-

8.2. Cylinder and deformable flag subject to gravity

Before a full FSI-case is validated, the structural solver, described in Section 6.1, itself needs to be evaluated. This is done using the CSM3 benchmark case proposed by Turek and Hron [8], which uses the same body definition as the previous benchmark case by Turek and Hron (listed in detail in Section 7.3), but now the flag has been made flexible and is subject to a gravitational load g in the negative vertical direction. The flag is released from an undeformed state, resulting in a periodic vibration around the equilibrium state of the flag.

The corresponding parameters are shown in Table 8.2. Two sets of simulations are run. The first set, described in Section 8.2.1, considers the effect of refinement of the spatial discretisation, whereas Section 8.2.2 considers the effect of a refinement of the timestep. In all cases, the simulations were run for $0 \leq t \leq 2$, to capture two periods of motion fully.

8.2.1. Evaluation of CSM3 benchmark - effect of spatial discretisation

The first set of simulations was run at at constant physical timestep of 0.001 25 s, with the beam being divided into $N = 70$, $N = 140$, $N = 280$ and $N = 560$ elements for different levels of mesh refinement. The resulting mean vertical displacement and amplitude of the tip of the flag is shown in Table 8.3. Furthermore, an additional refined grid with $N = 2240$ elements was simulated, to obtain a pseudo-exact solution. The results from the mesh levels included in Table 8.3 are compared to this solution, and a plot of the absolute difference between the mean value of the tip displacement as function of mesh refinement is made; the result is shown

Table 8.2: Input parameters corresponding to CSM3.

Parameter	CSM3
ρ_s	$1 \times 10^3\,\mathrm{kg/m^3}$
ν_s	0.4
E_s	$1.4 \times 10^6\,\mathrm{N/m^2}$
g	$2\,\mathrm{m/s^2}$

in Figure 8.4.

Table 8.3: Mean vertical displacement, amplitude and frequency of vertical displacement of flag tip, as function of spatial refinement.

Grid size	d_y of tip [m]	Period [s]
$N = 70$	-0.06809 ± 0.06978	0.925
$N = 140$	-0.06796 ± 0.06966	0.925
$N = 280$	-0.06788 ± 0.06960	0.925
$N = 560$	-0.06784 ± 0.06957	0.925
Target [8]	-0.06361 ± 0.06516	0.913

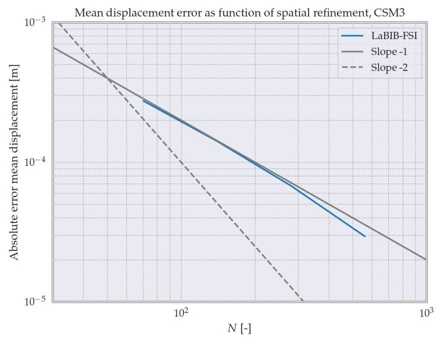


Figure 8.4: Plot of the absolute error of the mean displacement as function of spatial refinement.

From Table 8.3 and Figure 8.4, it is evident that the structural solver quickly converges to a solution. From Figure 8.4, it appears that the solver is first-order accurate in space. However, it should be noted that there is still a difference between the obtained results and those from Turek and Hron, that does not vanish as the spatial mesh is further refined, and this error seems to be around 5% large in both the mean and amplitude value of the displacement, with a small error in the period of oscillations present, too.

As a refinement of the temporal discretisation is also necessary to properly evaluate the discretisation error, the reason for this discrepancy is addressed at the end of Section 8.2.2.

8.2.2. Evaluation of CSM3 benchmark - effect of temporal discretisation

The second set of simulations is run at a constant spatial discretisation such that the beam consisted of $N = 560$ elements. Several simulations are run at physical timesteps equal to $\Delta t = \frac{2^{-i}}{25}$, with i ranging from $i = -1$ to $i = 11$. The numerical values of the mean displacement, amplitude and period for $i = 3$ to $i = 7$ are included in Table 8.4 (note that $i = 5$ corresponds to a physical timestep of 0.001 25 s, the timestep used in Section 8.2.1). Furthermore, taking the solution for $i = 11$ as pseudo-exact, the mean displacement for all other simulations is once again compared to this solution, and the resulting absolute error is plotted as function of temporal refinement in Figure 8.5.

Table 8.4: Mean vertical displacement, amplitude and frequency of vertical displacement of flag tip, as function of temporal refinement.

Grid size	d_y of tip [m]	Period [s]
$i = 3$	-0.06790 ± 0.06942	0.925
$i = 4$	-0.06788 ± 0.06950	0.9225
$i = 5$	-0.06784 ± 0.06957	0.925
$i = 6$	-0.06784 ± 0.06959	0.925
$i = 7$	-0.06784 ± 0.06959	0.925
Target [8]	-0.06361 ± 0.06516	0.913

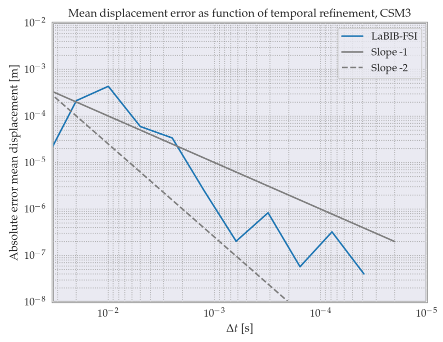


Figure 8.5: Plot of the absolute error of the mean displacement as function of temporal refinement.

It is evident from Figure 8.5 that the error appears to behave somewhere between a first- and second-order discretisation error, although it appears less well-behaved than the spatial discretisation error previously seen in Figure 8.4. As a trapezoidal scheme is used in the structural solver (as detailed in Section 6.1), this is in line with the expected order of accuracy.

However, based on the trends of the spatial and temporal mesh refinements studies, it is apparent that the difference between the results from Turek and Hron [8] and the obtained results using `pyfe3d` cannot be solely explained due to a discretisation error, as the discretisation error appears to be in the order of 0.1% when using $N = 560$ and $i = 6$ for example, whereas the error compared to Turek and Hron is in the order of 5%. Therefore, the remaining error is likely caused due to modelling errors.

Several modelling assumptions may contribute to these modelling errors. First of all, the structural model reduces the flag to a one-dimensional beam consisting of Timoshenko beam elements and assumes planar surfaces remain planar, which may not necessarily be the case in real life. This may be of particular importance in the attachment of the flag to the cylinder, which is modelled as a single, cantilevered joint, but in reality follows the curved boundary of the cylinder.

Secondly, the displacement of the tip of the flag is relatively large in comparison to the length of the beam - the maximum displacement is around 0.129 m, whereas the beam is 0.35 m long. However, in beam theory, it is assumed the displacements remain small [81] to ensure a linear model, and it is questionable whether a displacement of 37% of the beam length qualifies as a small displacement. This could be partially resolved by remeshing the structural model every few timesteps, such that the current displacements of the beam are imposed as a pre-applied deformation, and the linearised solution occurs around this pre-applied deformation, allowing to capture non-linear effects. However, this is not implemented as of yet.

All in all, although there appears to be a small modelling error, it is evident the structural solver works correctly for simple, constant loading scenarios and is ready to be validated as part of more complex fluid-structure interaction problems. Nonetheless, the error already existing for this simple validation case ought to be kept in mind when evaluating subsequent validation cases.

8.3. Cylinder and deformable flag immersed in horizontal flow

The FSI solver was then validated using the FSI1 and FSI3 benchmark cases proposed by Turek and Hron [8], which uses the same computational domain and body definition as the previous benchmark cases by Turek and Hron (listed in detail in Section 7.3), but now with a flexible body. The fluid properties for FSI1 and FSI3 are listed in Table 8.5; the structural properties are listed in Table 8.6. The FSI1 benchmark case results in a steady-state solution with a small tip deflection of the flag attached to the cylinder, whereas the FSI3 benchmark case results in a transient solution with moderate displacements of the flag (the amplitude of the deflection is about 10% of the length of the beam).

Table 8.5: Fluid parameters corresponding to FSI1 and FSI3.

Parameter	FSI1	FSI3
ρ_f	1×10^3 kg/m^3	1×10^3 kg/m^3
ν_f	1×10^{-3} m^2/s	1×10^{-3} m^2/s
u_0	0.2 m/s	2 m/s
Re	20	200

Table 8.6: Structural parameters corresponding to FSI1 and FSI3.

Parameter	FSI1	FSI3
ρ_s	1×10^3 kg/m^3	1×10^3 kg/m^3
ν_s	0.4	0.4
E_s	1.4×10^6 N/m^2	5.6×10^6 N/m^2

The simulations were all performed at a Mach number of $M = 0.04$ and run until $t = 8$ s, using a grid with a refinement such that $R = 20\Delta x$.

8.3.1. Evaluation of FSI1 benchmark - comparison of body force and displacement

The FSI1 case was initialised by setting the initial flow velocity to 0 everywhere in the domain, and using an inflow condition of

$$u(0,y) = u(L,y) = \frac{3}{2}u_0\frac{y(y-H)}{4H^2}, \tag{8.10}$$

multiplied with a linear time-ramp up until $t = 1\,\text{s}$. The simulation is run for $0 \leq t \leq 8\,\text{s}$. The simulations was performed for only the MIIM configuration[1], at a grid refinement level of $R = 20\Delta x$. 25 iterations for every time step of the MIIM. In Table 8.7, the resulting drag, lift and tip displacement for the MIIM is shown.

Table 8.7: Drag, lift and tip displacement for the FSI3 benchmark cases from Turek and Hron [8].

Configuration	Grid size	FSI1		
		F_x [N]	F_y [N]	d_y [mm]
IBM	$R = 20\Delta x$	14.56	0.392	1.739
	$R = 40\Delta x$	14.37	0.598	1.119
	$R = 80\Delta x$	14.31	0.636	1.089
MIIM	$R = 20\Delta x$	14.45	0.577	1.208
	$R = 40\Delta x$	14.34	0.710	0.906
	$R = 80\Delta x$	14.29	0.707	0.909
Target [8]	-	14.30	0.7638	0.8209

As can be seen from Table 8.7, the resulting drag coefficient appears to correspond reasonably well with the reference value from Turek and Hron [8]. However, a notable difference in lift and vertical displacement of the tip is clearly present.

In particular, a counterintuitive trend present in the IBM and MIIM configuration is that tip displacement appears to decrease as the vertical force on the system increases, whereas one may expect a positive correlation to exist there. However, from comparison of the FSI1 and CFD1 benchmarks (which are identical, except that the flag is rigid in the CFD1 case), it can be remarked that apparently even the small displacement of 0.8209 mm causes a large negative change in lift force - in the CFD1 benchmark, the lift is approximately 1.119 N, yet the reference value has already dropped to 0.7638 N in the FSI1 benchmark, which can only be due to the displacement of the flag.

To elucidate this reasoning, a plot is made of the displacement of the tip of the beam as function of vertical for the various various FSI1 runs, as well as the CFD1 runs of Section 7.3.1 (where the displacement is naturally 0). This plot is shown in Figure 8.6. If it is assumed that the solution behaves linearly, i.e. the tip displacement varies linearly with the vertical force (which appears as a reasonable assumption considering the small displacements involved), then all results obtained by the IBM and MIIM configurations appear to follow the trend-line that goes through the datapoints from Turek and Hron.

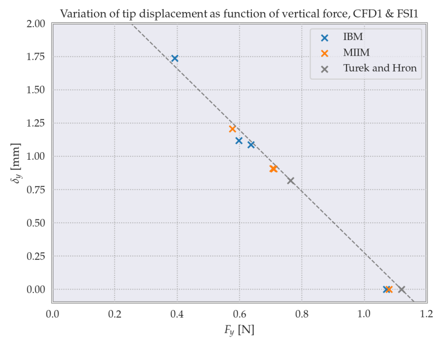


Figure 8.6: Plot of the tip displacement as function of vertical force on the system, for FSI1 and CFD1 benchmark cases.

Thus, although there is clearly some error (either modelling or discretisation) present (particularly at $R = 20\Delta x$) in the IBM and MIIM results, the behaviour of the vertical force compared to the tip displacement appears consistent with that found by Turek and Hron, and the error in both quantities is likely predominantly induced by the same discretisation error.

[1] This was a result as a small change in the code suddenly leading to the IBM misbehaving when running this testcase; it is the intention that results for the IBM will also be added later (previously, correct results already had been achieved for this benchmark case, so it is merely a matter of bug-fixing the code).

The large difference in vertical displacement of the tip may be explained due to the structural modelling approach chosen, in which the flag is reduced to a one-dimensional beam, modelled through Timoshenko beam elements. This is a rather large simplification, as the flag is not extremely slender, being 35 cm long whilst having a cross-sectional dimension of 2 cm.

Indeed, Geller et al. [80], who also use an diffusive IBM within the LBM, find that the displacement of the tip of the beam is greatly dependent on the structural model they choose, with the model consisting of one-dimensional beam elements predicting a displacement of 2 mm, whilst a pFEM-based solver would predict a much more reasonable 0.76 mm. It is therefore imperative that in the future, LaBIB-FSI is coupled to a more sophisticated structural model, to explore similar deviations in the results are experienced here.

8.3.2. Evaluation of FSI3 benchmark - comparison of force history

The resulting drag as function of time is shown in Figure 8.7, where it is compared to the values obtained by Turek and Hron [8]. Similarly, the lift and vertical displacement of the tip are shown in Figure 8.8 and 8.9. The results from the various sources were fitted such that the phase of the displacement history would coincide. A close-up of the time-history is provided in Figure 8.10 and 8.11. Numerical values are included in Table 8.8.

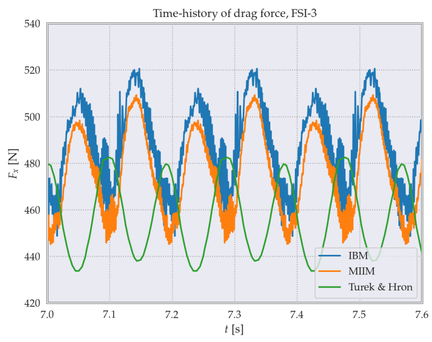


Figure 8.7: Drag as function of time, compared to the reference solution by Turek and Hron [8], using $R = 20\Delta x$.

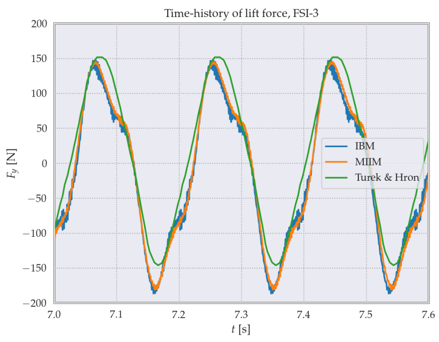


Figure 8.8: Lift as function of time, compared to the reference solution by Turek and Hron [8], using $R = 20\Delta x$.

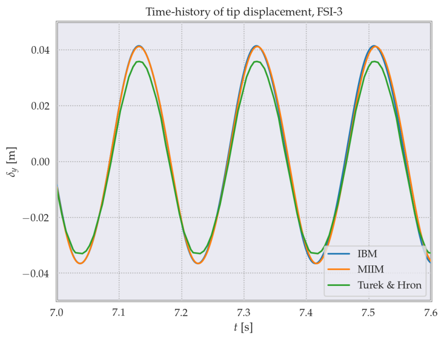


Figure 8.9: Vertical displacement of the tip, compared to the reference solution by Turek and Hron [8], using $R = 20\Delta x$.

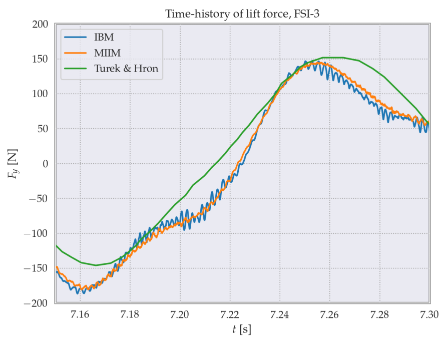


Figure 8.10: Lift as function of time, compared to the reference solution by Turek and Hron [8], using $R = 20\Delta x$. Zoomed into the domain $t \in [7.15, 7.30]$.

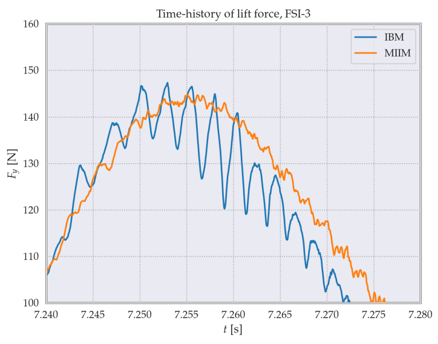


Figure 8.11: Lift as function of time, compared to the reference solution by Turek and Hron [8], using $R = 20\Delta x$. Zoomed into the domain $t \in [7.24, 7.28]$.

From Figure 8.7, 8.8 and 8.9, as well as the numerical results in Table 8.8, it can be observed that the IBM and MIIM both approximate the reference values by Turek and Horn reasonably well, matching the mean values with an error no greater than approximately 10% of the amplitude, and having errors in the amplitude of at most in the order of 10%, too. The only major difference is observed in the time-history of the drag force, which appears to be approximately a quarter of a period out of phase w.r.t. the reference solution. The cause of this phase shift is explained in Section 8.3.3.

Finally, from Figure 8.10 and 8.11, which are close-ups of the time-history of the lift force, an important difference between the MIIM and IBM in the results may be observed - the MIIM appears to produce a significantly smoother result, whereas the IBM suffers from notable spurious oscillations. This is consistent with results found in Chapter 7, where it was already seen that the MIIM resulted in less noise in for example the boundary force variation around the cylinder and flag for the CFD1 and CFD2 cases.

Table 8.8: Vertical displacement, lift- and drag-coefficient for the FSI3 benchmark cases from Turek and Hron [8].

Configuration	Grid size	F_x [N]		F_y [N]		δ_y [m]		Period [s]
		Mean value	Amplitude	Mean value	Amplitude	Mean value	Amplitude	
IBM	$R = 20\Delta x$	484.65	35.97	-19.54	166.92	0.00247	0.0390	0.1885
	$R = 40\Delta x$	472.92	27.58	-8.642	164.45	0.00202	0.0373	0.1866
MIIM	$R = 20\Delta x$	477.30	32.02	-17.98	162.78	0.00234	0.0389	0.1895
	$R = 40\Delta x$	475.22	32.32	-8.509	165.05	0.00204	0.0391	0.1893
Target [8]	-	457.30	22.66	2.22	149.8	0.00148	0.03438	0.1887

8.3.3. Evaluation of FSI3 benchmark - evaluation of added mass effect

As noted in Section 8.3.2, there appears to be a quarter-period phase shift between the drag predicted by the LaBIB-solver vs. that predicted by Turek and Hron when the displacements are fitted such that their phases coincide. This phase shift also did not decrease with increasing mesh refinement, indicating that the error is not due to a discretisation error.

Indeed, it should be noted that the LaBIB-solver currently does not compensate for the added mass effect during its simulations itself - the compensation for the added mass effect in Section 8.1 was done fully during post-processing, as in that benchmark case, the cylinder is rigid and has a prescribed velocity (meaning that Equation (8.4) can be easily evaluated a posteriori, and the body force never needs to be computed during the simulation itself, as the movement of the cylinder is not affected by it).

However, evidently, in the case of FSI3, it is relevant that the force obtained by the boundary forcing method is adjusted for the added mass effect, as the structural solver should only receive the net force acting on the flag. The effect of this omission may be estimated by evaluating Equation (8.4) in post-processing. After all, given the displacement $d_{y,\kappa}^{(k)}$ at Lagrangian markers κ at timesteps k, one can estimate the acceleration via a central-difference scheme,

$$a_{y,\kappa}^{(k)} = \frac{d_{y,\kappa}^{(k+1)} - 2d_{y,\kappa}^{(k)} + d_{y,\kappa}^{(k-1)}}{\Delta t^2}. \tag{8.11}$$

By dividing the beam up in small elements and using a lumped-mass model, Equation (8.4) may then be straightforwardly calculated. The force due to the added mass for approximately one period of motion is calculated for both the IBM and MIIM using the simulation with a grid refinement of $R = 20\Delta x$, and is plotted in Figure 8.12 and 8.13, respectively.

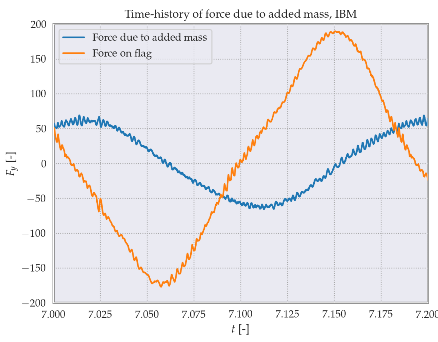


Figure 8.12: Force due to added mass effect over a single period of motion for the IBM, at a grid resolution of $R = 20\Delta x$, compared to the total force on the flag.

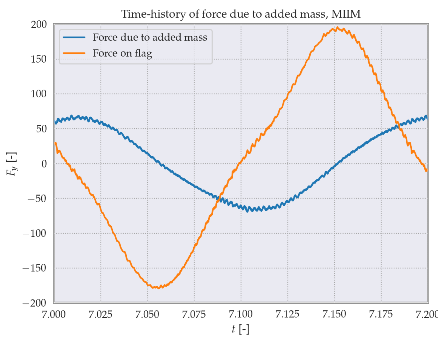


Figure 8.13: Force due to added mass effect over a single period of motion for the MIIM, at a grid resolution of $R = 20\Delta x$, compared to the total force on the flag.

It is evident that the magnitude of the added mass effect cannot be neglected, and it appears roughly one-quarter of a period out of phase. Clearly, this will likely have had a profound effect on the fluid-structure interaction of the flag, as the change in forcing applied to the flag would have resulted in a different displacement profile, which in turn affects the added mass effect. Moreover, the total force on the flag is not fully representative of what displacements the structural solver predicts, as the boundary force distribution is obviously also of relevance - in fact, the force due to the added mass effect is likely to be concentrated towards the tip of the beam (due to the tip of the beam experiencing the largest displacements), and so compensating may have an even larger effect than Figure 8.12 and 8.13 suggest.

It is therefore imperative that in the future, the LaBIB-solver is extended to properly compensate for the added mass effect during its simulation, such that this validation case can be validated properly. It should be noted that compensating for the added mass effect is not a trivial task. Equation (8.4) describes the added mass for the structure as a whole, but does not directly describe how the boundary force at each Lagrangian marker from an immersed boundary approach should be adjusted.

Finally, it should be noted that the benchmark case by Turek and Hron [8], where the density of the fluid is equal to the density of the structure material, is more suspect to the added mass effect than more practical validation cases, where the density of the structure is likely significantly larger than the density of the fluid. In that case, the force required to displace the fluid trapped inside the immersed boundary is much smaller compared to the force required to displace the structure itself, and thus neglecting the impact of the added mass effect is a much safer assumption.

8.3.4. Evaluation of FSI3 benchmark - temporal variation of numerical noise

It was previously seen for the CFD1 & CFD2 benchmark (Section 7.3) that significant spatial oscillations are present in the boundary force distribution obtained from the IBM/MIIM. These are still present in the FSI3 benchmark, and it is of interest to observe how these oscillations behave over time, and whether they could be smoothed out by simple temporal averaging of the N_t most recent timesteps, i.e. whether

$$\bar{F}_{\kappa}^{(k)} = \frac{1}{N_t} \sum_{i=0}^{N_t} F_{\kappa}^{(k-i)} \tag{8.12}$$

would yield a smoother result without degrading in accuracy. To investigate this, the boundary force distribution for the FSI3 runs were time-averaged for three different values of N_t and plotted in Figure 8.14-8.17. Note that a single time-step is equal to approximately 2.89×10^{-5} s, and thus $N_t = 1000$ corresponds to an averaging window of approximately 0.0289 s, approximately 1/6th of a full period of motion. The plots correspond to runs with a mesh refinement of $R = 20\Delta x$.

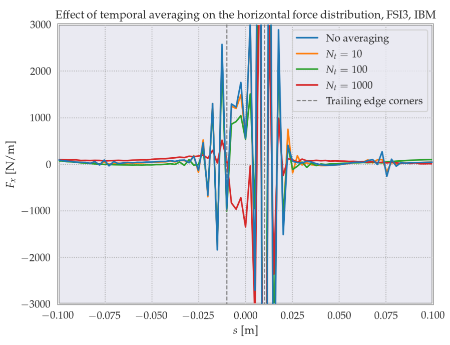


Figure 8.14: Horizontal force component density as function of parametric coordinate, for the IBM for FSI3, for three levels of temporal averaging.

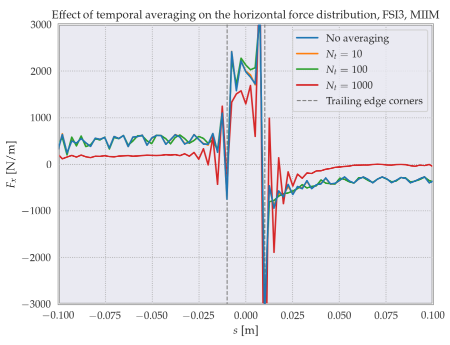


Figure 8.15: Horizontal force component density as function of parametric coordinate, for the MIIM for FSI3, for three levels of temporal averaging.

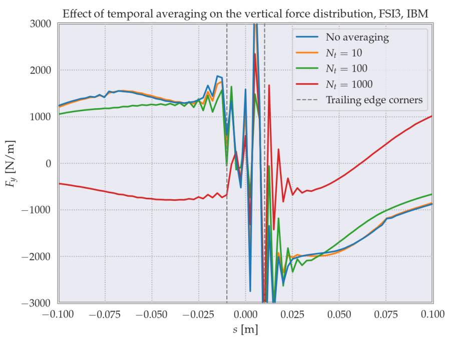


Figure 8.16: Vertical force component density as function of parametric coordinate, for the IBM for FSI3, for three levels of temporal averaging.

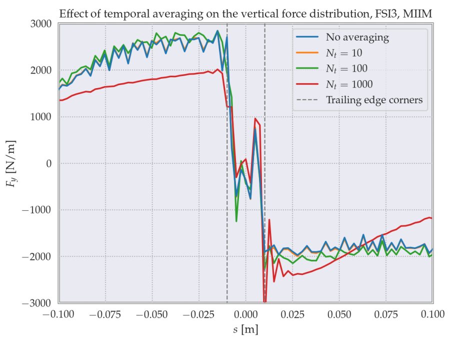


Figure 8.17: Vertical force component density as function of parametric coordinate, for the MIIM for FSI3, for three levels of temporal averaging.

It can be observed that temporal averaging offers little benefit - a small window does not smooth out the oscillations at all, and only at much larger windows are the distributions smoothed. However, at these larger windows, the accuracy also notably decreases, with the boundary force distribution for $N_t = 1000$ being significantly different from the original distribution. It can be concluded these oscillations are only of spatial nature, and do not vary temp

8.3.5. Evaluation of FSI3 benchmark - consistency of FSI procedure

In Section 6.2.1, it is acknowledged that the spatial interpolation of the FSI algorithm is not consistent, as a polynomial term is not added to the sum of radial basis functions, which would otherwise be able to represent any constant or linearly varying part of the interpolated quantity. It is therefore of interest to evaluate the consistency of the FSI operator. To do so, first the boundary force and displacement distribution of the flag at a single point in time for the FSI3 case (corresponding to maximum downward deflection of the tip of the flag) are plotted in Figure 8.18-8.23 for both the IBM and MIIM, to allow for a qualitative comparison. The plots were made for runs with a mesh refinement of $R = 20\Delta x$. The radial basis function used is the Wendland function [83] listed in Section 6.2.1, and the width of the compact support was set to 0.05 m.

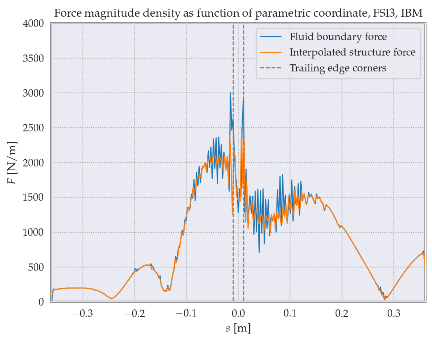


Figure 8.18: Boundary force magnitude density as function of parametric coordinate, for the IBM for FSI3, for both the fluid and structure, the latter being interpolated from the former.

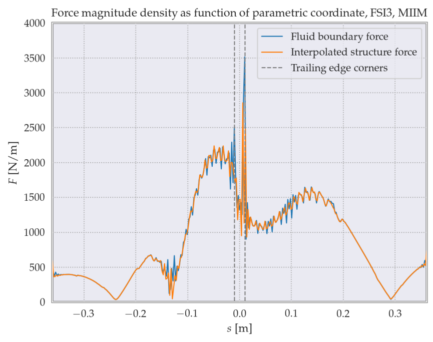


Figure 8.19: Boundary force magnitude density as function of parametric coordinate, for the MIIM for FSI3, for both the fluid and structure, the latter being interpolated from the former.

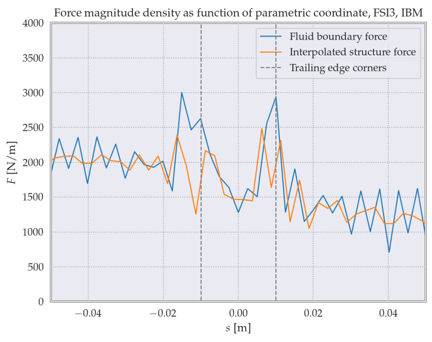


Figure 8.20: Boundary force magnitude density as function of parametric coordinate (zoomed into region around the tip of the flag), for the IBM for FSI3, for both the fluid and structure, the latter being interpolated from the former.

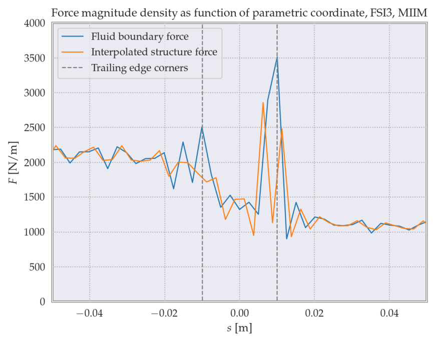


Figure 8.21: Boundary force magnitude density as function of parametric coordinate (zoomed into region around the tip of the flag), for the MIIM for FSI3, for both the fluid and structure, the latter being interpolated from the former.

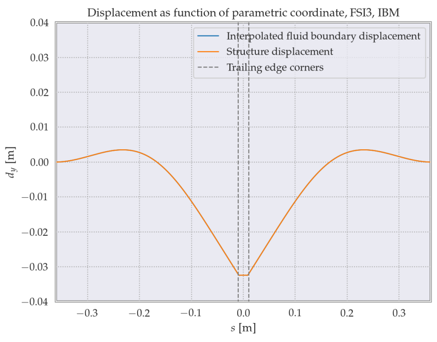


Figure 8.22: Deflection as function of parametric coordinate, for the IBM for FSI3, for both the fluid and structure, the former being interpolated from the latter.

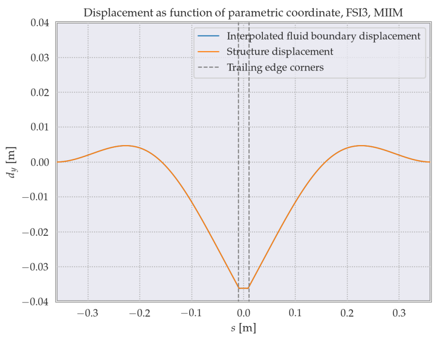


Figure 8.23: Deflection as function of parametric coordinate, for the IBM for FSI3, for both the fluid and structure, the former being interpolated from the latter.

It is evident that the FSI operator interpolates the meshes correctly, even in absence of a polynomial in the interpolant, likely due to the fact that neither the boundary force nor the displacement are characterised by either a constant or linear distribution. Furthermore, as apparent from Figure 8.20 and 8.21, the quality of the interpolant does deteriorate in the region around the tip, due to the large gradients and oscillations due to numerical noise arising in the force distribution.

Apart from this qualitative comparison, the performance of the FSI-operator may also be quantified. After all, it is desirable that the total boundary force at each time step, as well as the total work done on the boundary during each time step, is equivalent between the fluid and structure surface. That is, the following equivalence holds:

$$F_f^{(k)} \sum_{\kappa_f} F_{\kappa_f}^{(k)} \Delta V_{\kappa_f} = \sum_{\kappa_s} F_{\kappa_s}^{(k)} \Delta V_{\kappa_s} = F_s^{(k)}, \tag{8.13}$$

where the left-hand summation is taken over all the fluid boundary elements, and the right-hand summation is taken over all the structure boundary elements, where $F_\kappa^{(k)}$ is the magnitude of the boundary force density at Lagrangian marker κ and time step k, and ΔV_κ is the volume associated with this marker. Similarly, the following equivalence should hold for the work done during each time step k:

$$W_f^{(k)} = \sum_{\kappa_f} \frac{\boldsymbol{F}_{\kappa_f}^{(k)} + \boldsymbol{F}_{\kappa_f}^{(k-1)}}{2} \cdot \left(\boldsymbol{d}_{\kappa_f}^{(k)} - \boldsymbol{d}_{\kappa_f}^{(k-1)}\right) \Delta V_{\kappa_f} = \sum_{\kappa_s} \frac{\boldsymbol{F}_{\kappa_s}^{(k)} + \boldsymbol{F}_{\kappa_s}^{(k-1)}}{2} \cdot \left(\boldsymbol{d}_{\kappa_s}^{(k)} - \boldsymbol{d}_{\kappa_s}^{(k-1)}\right) \Delta V_{\kappa_s} = W_s^{(k)}, \tag{8.14}$$

where again, the left-hand summation is taken over all the fluid boundary elements, and the right-hand summation is taken over all the structure boundary elements, $\boldsymbol{F}_\kappa^{(k)}$ is the boundary force density vector at marker κ and time step k and $\boldsymbol{d}_\kappa^{(k)}$ is the displacement vector at marker κ and time step k. Due to interpolation errors however, there is a small deviation between whether these quantities are evaluated on the fluid boundary compared to evaluation on the structure mesh. Therefore, defining

$$\epsilon_F^{(k)} = \frac{2\left|F_f^{(k)} - F_s^{(k)}\right|}{\left|F_f^{(k)}\right| + \left|F_s^{(k)}\right|_{\max}} \tag{8.15}$$

$$\epsilon_W^{(k)} = \frac{2\left|W_f^{(k)} - W_s^{(k)}\right|}{\left|W_f^{(k)}\right| + \left|W_s^{(k)}\right|_{\max}}, \tag{8.16}$$

where the denominator represents the global maximum value of the sum of the absolute values of the quantities in the denominator. The resulting error in the boundary force, and the error in the work done per time step, is then evaluated over a full period of motion, and is plotted in Figure 8.24 and 8.25, respectively.

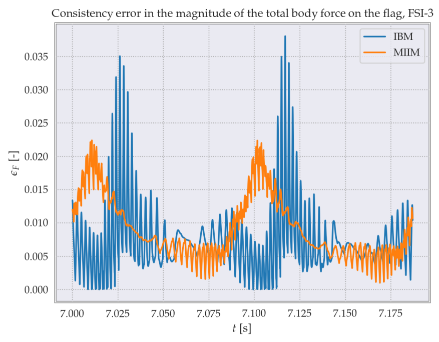


Figure 8.24: Interpolation error in the total boundary force acting on the flag, as evaluated from the fluid surface and structure surface, for both the IBM and MIIM.

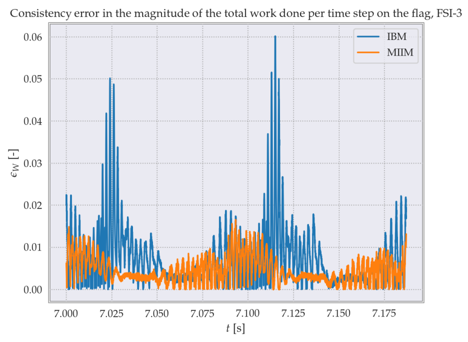


Figure 8.25: Interpolation error in the total work done on the flag per time step, as evaluated from the fluid surface and structure surface, for both the IBM and MIIM.

From Figure 8.24 and 8.25, it is evident that the interpolation is relatively small for both the IBM and MIIM, noting that the quantities plotted (defined in Equation (8.15) and (8.16)) compute the difference between the structure and fluid surface quantities relative to the average global maximum value of those quantities. This is further proof the spatial interpolation of the FSI algorithm has been implemented correctly.

Furthermore, it can be noted that for the error in the work done per time step, the MIIM appears to perform better than the IBM, with the error corresponding to the MIIM mostly remaining below that predicted by the IBM. This may be caused by the MIIM giving producing a smoother boundary force distribution, therefore making it easier to interpolate and resulting in a smaller interpolation error than for the IBM. However, it should be acknowledged that Figure 8.24 is inconclusive at best with regards to whether MIIM or IBM results in a smaller consistency error overall. Therefore, it cannot be established with certainty that the MIIM will always yield better results in an FSI-interpolation, or whether Figure 8.25 is merely a fluke. Evaluating more validation cases, executed at a higher mesh resolution, would likely in arriving at a more conclusive answer.

9 Sensitivity Analysis

In this chapter, the effect of several numerical parameters are discussed. In Section 9.1, the effect of the mesh refinement on the oscillations in the boundary force variation is evaluated, and it is seen whether increasing the mesh resolution aids in reducing the numerical noise. In Section 9.2, the effect of the Mach number is evaluated, as a higher Mach number generally smoothens the solution due to the added, nonphysical compressibility. In Section 9.3, the effect of the width of the discrete delta function is evaluated.

9.1. Effect of mesh refinement on oscillations in boundary force variation

In Section 7.3.2 and 7.3.3, it is shown that the IBM has significantly more oscillations in its boundary force distributions than the MIIM at the same level of grid refinement for the CFD1 benchmark, a result also implied by the evaluation of a Taylor-Green vortex flow with an immersed body in Section 7.2.

However, it remains of interest how these oscillations behave as the mesh is refined, and whether they tend to naturally smoothen out. To investigate this, the boundary force distribution near the tip of the flag of the FSI-1 benchmark, discussed in Section 8.3.1, is evaluated for different mesh refinement levels. The variation of the force magnitude and its components for both the IBM and MIIM are shown in Figure 9.1-9.6, for three levels of grid refinement. As in Section 7.3.3, s represents the parametric coordinate describing the flag, with $s = 0$ located at the midpoint of the trailing edge, then with a flag thickness of $h = 0.02$ m, the corners of the flag are located at $s = 0.01$ m and $s = -0.01$ m.

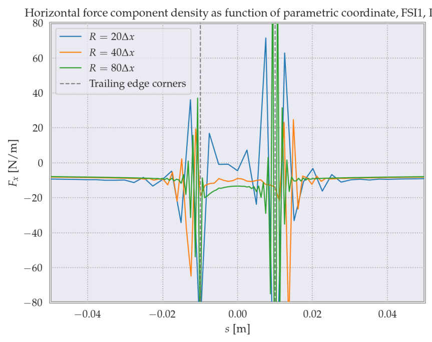


Figure 9.1: Horizontal force component density as function of parametric coordinate, for the IBM for FSI1, for three mesh refinement levels.

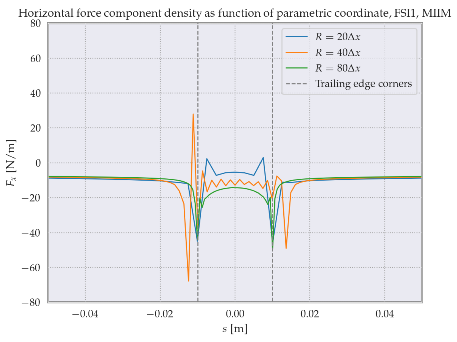


Figure 9.2: Horizontal force component density as function of parametric coordinate, for the MIIM for FSI1, for three mesh refinement levels.

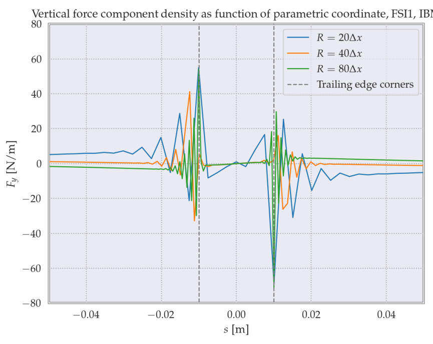


Figure 9.3: Vertical force component density as function of parametric coordinate, for the IBM for FSI1, for three mesh refinement levels.

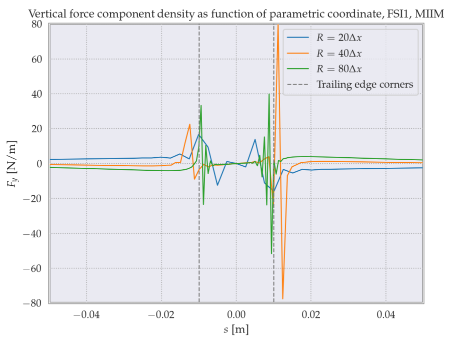


Figure 9.4: Vertical force component density as function of parametric coordinate, for the MIIM for FSI1, for three mesh refinement levels.

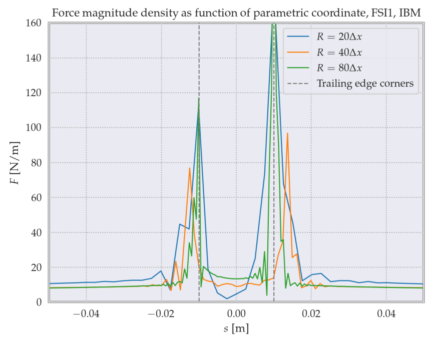


Figure 9.5: Force magnitude density as function of parametric coordinate, for the IBM for FSI1, for three mesh refinement levels.

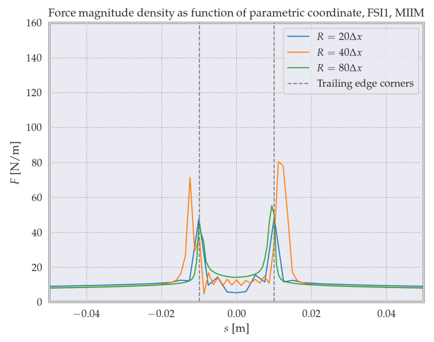


Figure 9.6: Force magnitude density as function of parametric coordinate, for the IBM for MIIM, for three mesh refinement levels.

From Figure 9.1-9.6, it is evident that for both the IBM and MIIM, increasing the mesh refinement does reduce the width of the area that is most subject to oscillations in the boundary force variation, and this area appears to reduce proportionally to the mesh refinement - when the mesh refinement is doubled, the width of the oscillations appears to half as well.

However, the amplitude of the oscillations does not seem to decrease with increasing mesh refinement. Therefore, even with a refined mesh, the quality of the results obtained by the IBM will not converge to those of the MIIM, a clear benefit to the MIIM.

9.2. Effect of Mach number on oscillations on the boundary force variation

The Mach number at which the simulation essentially controls the physical time-step of the simulation. Naturally, preferably the Mach number is as small as possible as all validation cases used in Chapter 7 and 8 assume incompressible flow. However, the Mach number may be set higher to speed up the simulation, and to increase the stability of the simulation, as a higher Mach number results in a higher relaxation rate τ (see Section 3.4.4), and thus the stability limit of $\tau \rightarrow \frac{1}{2}$ is avoided.

To evaluate this effect, the IBM simulation was repeated at different Mach numbers, ranging from $M = 0.04$ to $M = 0.32$ (based on the average inlet velocity). The resulting body forces are shown in Table 9.1, and the resulting boundary force distribution in the region near the tip of the flag (an area previously identified as being particularly prone to oscillations) is shown in Figure 9.7 for the various IBM runs, including a comparison to the MIIM at a Mach number of $M = 0.04$. The simulation was run at a spatial discretisation such that $R = 20\Delta x$.

Table 9.1: Lift and drag for the CFD1 benchmark case from Turek and Hron [8], with the Mach number varied among different IBM runs.

Configuration	M	F_x [N]	F_y [N]
IBM	0.04	14.49	1.070
	0.08	14.47	1.033
	0.16	14.49	0.964
	0.32	15.11	0.680
IIM	0.04	14.43	1.079
Target [8]	-	14.29	1.119

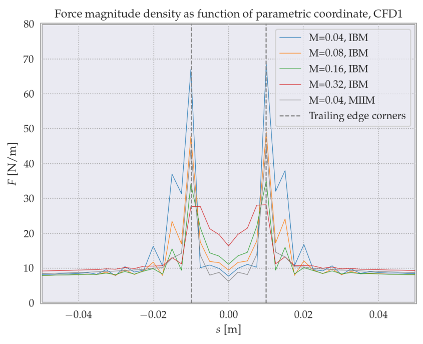


Figure 9.7: Plot of the boundary force distribution at the tip of the flag, for different Mach numbers, for CFD1 at a mesh discretisation such that $R = 20\Delta x$.

It can be seen from Figure 9.7 that increasing the Mach number certainly aids in reducing the oscillations experienced by the IBM. However, in order to smoothen the solution as well as the MIIM inherently does, the Mach number needs to be between $M = 0.16$ and $M = 0.32$. However, it can also be seen that increasing the Mach number this much appears to introduce an error in the boundary force at the tip itself, with the magnitude of it at $s = 0.0\,\text{m}$ growing increasingly with a larger Mach number. Furthermore, from Table 9.1, it appears that increasing the Mach number above $M = 0.08$ has detrimental effects on the accuracy of the lift force.

This appears to make the MIIM a more favourable way of reducing oscillations in the boundary force variation than merely increasing the Mach number in the original IBM, as the required increase in Mach number to attain a similar level of smoothness is quite large.

9.3. Effect of discrete delta function width on the boundary force variation

Similarly, we may investigate the effectiveness of widening the diffusive zone around the boundary of the flag and cylinder by choosing a different discrete delta function (listed in Section 4.1). After all, by widening this diffusive zone, the one essentially smooths out the boundary, and as a result, it can be expected that numerical noise in the solution is damped out (at the cost of general accuracy of the solution).

To investigate this, the IBM simulation was repeated using different discrete delta functions; in particular, the ϕ_2, ϕ_3 and ϕ_4-functions listed in Section 4.1 (note that ϕ_2 was the default delta-function that was used throughout all of the validation cases in Chapter 7 and 8), as well as the smoothed discrete-delta function ϕ_5 proposed by Yang et al. [39], which can be evaluated as

$$\phi_5(r) = \int_{r-0.5}^{r+0.5} \phi_4(\tilde{r})\, d\tilde{r}. \tag{9.1}$$

The resulting body forces are shown in Table 9.1, and the resulting boundary force distribution in the region near the tip of the flag (an area previously identified as being particularly prone to oscillations) is shown in Figure 9.7 for the various IBM runs, including a comparison to the MIIM using ϕ_2. The results were generated using a $M = 0.04$, and using a grid discretisation such that $R = 20\Delta x$.

Table 9.2: Lift and drag for the CFD1 benchmark case from Turek and Hron [8], with the discrete delta function varied among different IBM runs.

Configuration	Discrete delta function	F_x [N]	F_y [N]
IBM	ϕ_2	14.49	1.070
	ϕ_3	14.58	1.092
	ϕ_4	14.68	1.105
	ϕ_5	14.72	1.109
IIM	ϕ_2	14.43	1.079
Target [8]	-	14.29	1.119

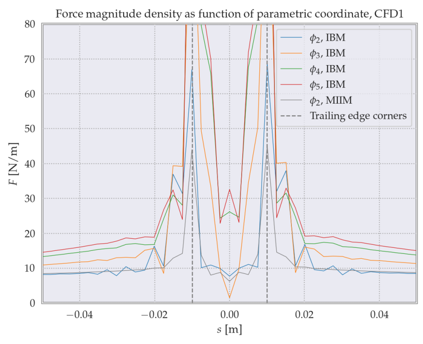


Figure 9.8: Plot of the boundary force distribution at the tip of the flag, for different Mach numbers, for CFD1 at a mesh discretisation such that $R = 20\Delta x$.

It can be seen that although using a wider support radius reduces the the oscillations in the boundary force density somewhat (in particular, note how for ϕ_2, oscillations start to occur around $s = \pm 0.04\,\text{m}$, but for the wider delta functions, they appear to start slightly closer to the edge). However, overall, even ϕ_5 does not come close to the smoothness of the MIIM solution. Furthermore, the accuracy of the body forces also appears somewhat lesser at wider discrete delta functions, although it is not as drastic as increasing the Mach number in Section 9.2 was.

Thus, the MIIM appears significantly more effective at reducing numerical noise than changing the discrete delta function is.

However, even if widening the support radius was effective, there are still important caveats to make. First, widening the diffusive zone means the boundary is smeared out over a larger part, which inevitably decreases the accuracy of the solution. This may be an issue in particular when two boundaries are in close proximity to each other (e.g. two wings from a flapping wing MAV providing a clap-and-fling effect), as the fluid solver may start to be unable to distinguish effectively between the boundaries due to the diffusion of them.

Furthermore, the computational cost introduced by it is significant. When widening the support radius from 2 to 4 lattice units for example, the number of to be interpolated nodes increases by a factor 4 in a two-dimensional lattice. Not only that, the complexity of the weighting function also increases significantly, which makes the evaluation of it noticeably slower.

In conclusion, the MIIM is likely a more efficient solution at reducing the oscillations in the solution than widening the support radius is, both in terms of effectiveness, accuracy and computational effort. This once again speaks to the merits of the MIIM.

10 Conclusion

Throughout this thesis, a number of important results were obtained. This enables answering the research questions which will be done in this Chapter.

1. *How suitable is the lattice Boltzmann method for the fluid-structure-interaction of flapping wing MAVs, such as the DelFly II, in light of the the challenges posed by the highly deformable wings, low Reynolds number and high Strouhal number characteristics of such vehicles?*

From the variety of validation cases, it can be concluded that the LaBIB-FSI solver (and therefore, the LBM) is a suitable candidate for flow characteristic of flapping wing MAVs, with the solver showing accurate results for the aerodynamic forces and structural deformation across a number of validation cases. The LBM is a natural candidate for unsteady flow, and moving boundaries pose no significant difficulty compared to stationary boundaries, a suitable property for problems involving flapping wing MAVs.

Nonetheless, it should be acknowledged that no meaningful conclusion can be drawn on whether the LaBIB-FSI solver in its current state would be suitable for complex flow phenomena such as the clap-and-fling effect as none of the validation cases is similar to such a flow. This should therefore be the subject of future research, as recommended in Chapter 11.

Furthermore, it should be acknowledged that the results for the FSI3 benchmark, although promising, appeared somewhat inconsistent with the results by Turek and Hron [8]. However, the likely cause for this has been identified, and as recommended in Chapter 11, this is something that could be improved in future work on the LaBIB-FSI solver, allowing for proper validation of the FSI3 benchmark.

2. *How can the immersed interface method be applied to the lattice Boltzmann method?*

Chapter 5 has shown how jump conditions can be computed from a boundary force distribution, and subsequently imposed in population space. A variety of derivatives of the IIM in the LBM have been proposed, the most relevant of which is the MIIM introduced in Section 5.3.

As shown in Chapter 5, an attempt at applying the immersed interface method was previously made by Qin et al. [7]. However, their derivation appears to be several flawed in two aspects. One of them is a simple erroneous vector-operation that can be easily corrected for, but the other appears to be a misunderstanding of what jump distribution found them actually represents. It is shown in Section 5.7 that correcting for this ultimately results in an equivalent expression to the one derived in Section 5.1 (but note that Qin et al. used the incorrect expression).

Nevertheless, even if these mistakes are accounted for, the implementation proposed in Section 5.2 still appears superior to the one by Qin et al., as the implementation proposed in this thesis does not require the calculation of intersections between the lattice links and the boundary, and uses the IIM consistently for its entire boundary forcing computation, whereas Qin et. al utilise the IBM to obtain the boundary forcing, and then only impose the normal component of the boundary forcing through their IIM.

3. *How does the computational effort of the immersed interface method compare to that of the immersed boundary method, when applied to the Lattice Boltzmann method?*

The difference in computational effort between the IBM and IIM is discussed in Section 5.5.2, where it is discussed that both methods scale in terms of computational effort with regards to the number of Lagrangian markers in the boundary discretisation. However, the IIM is generally somewhat more expensive to run than the IBM, as the IIM interpolates in population space whereas the IBM interpolates in velocity space. As a result, the IIM requires the interpolation of 9/2 more solution quantities, which can be a considerable difference.

However, in practical examples, the LBM algorithm obviously includes several other steps, and generally, the number of Lagrangian markers is small compared to the total number of fluid nodes. Therefore, in runs such as those for the Turek and Hron [8], the IIM was only in the order of 10%-50% slower than the IBM (generally closer to 10%), although this has not been measured systematically, as this may be dependent on hardware and implementation optimisation, too.

4. *Does the implementation of the immersed interface method provide a tangible benefit over using the immersed boundary method in terms of numerical noise reduction and overall accuracy?*

The MIIM boasts important benefits over the IBM. It is consistently shown throughout Chapter 7 and 8 that the MIIM reduces numerical noise in the boundary force solution, for example resulting in significantly smaller spatial oscillations in the boundary force distribution in the region of the tip of the flag in the Turek and Hron [8] benchmark cases. This is an important benefit that may be of particular relevance for coupling with structural solvers, for whom an oscillating boundary force distribution may negatively affect the accuracy of the solution. Temporal variation of the total body force also appears to be lower, a result observed in the time-history of the body forces in the FSI3 benchmark, shown in Section 8.3.2.

Apart from reducing the noise in the solution, the MIIM also appears to produce more accurate results at more accurate results than the IBM at the same discretisation level, observed from more accurate values for the body forces for the CFD1, CFD2 and CFD3 benchmark cases of Section 7.3, more accurate displacements in the FSI1 benchmark of Section 8.3.1, and a smaller discretisation error (at the same grid refinement) than the IBM in the benchmark of an oscillating cylinder in a fluid at rest of Section 8.1.2. This is another tangible benefit to the IIM.

5. *How effective is the immersed interface method in damping out numerical noise compared to adjusting the numerical parameters of the simulations, such as the Mach number of the flow and the width of the discrete delta functions?*

The effect of the Mach number and width of the discrete delta function is evaluated in Section 9.2 and 9.3, respectively. The MIIM is a more efficient solution at reducing the oscillations in the solution than increasing the Mach number, as the Mach number needs to be increased significantly to achieve the same level of smoothness for the IBM that the MIIM reaches at a small Mach number, to the degree that the accuracy of the solution starts to deteriorate quickly. Similarly, the MIIM is more effective at reducing the numerical noise than widening the support radius is, both in terms of effectiveness, accuracy and computational effort. This once again speaks to the merits of the MIIM.

11 Recommendations

Despite the positive results obtained in this thesis, there is still a wide array of open questions related to the immersed boundary / immersed interface method, the lattice Boltzmann method in general, and the written LaBIB-FSI solver, which could form the basis of future research. In Section 11.1, several recommendations with regard to the immersed boundary / immersed interface method are made. Section 11.2 lists a number of recommendations regarding the lattice Boltzmann method in general. Finally, Section 11.3 lists a number of improvements that can be made to the LaBIB-FSI solver specifically.

11.1. Recommendations regarding boundary treatment

A number of opportunities for future research exist in the area of the IBM & IIM itself. First of all, as discussed in Chapter 5, the IIM-LBM possesses more control over the time-stepping scheme than the IBM-LBM is able to provide. Thus, it is of interest to investigate the use of more complex schemes than the midpoint method. It should be noted that in Section 5.8.1 a variation on the IIM based on a two-stage method is proposed, which appeared to give inferior results in Section 7.2, compared to the MIIM. However, this was merely one time-stepping scheme that has been tested in a limited fashion, and there is undoubtedly more to explore in this area.

Secondly, as discussed Section 5.6.2, the work by Tao et al. [76] appears interesting to combine with the IIM. After all, Tao et al. developed an IBM where the populations are directly corrected for the presence of a boundary (instead of through diffusion of a boundary force). Considering the IIM-LBM relies on interpolation of in population space instead of velocity space, the IIM-LBM appears a fitting candidate to attempt the same approach on.

Thirdly, to improve the performance of the boundary treatment at higher Reynolds number at lower grid resolution levels, it is interesting to apply a wall model to the solution. Wall models have been widely researched and applied in conventional, NS-based frameworks; see e.g. the recent works by Larsson et al. [99] and Bose and Park [100]. The central idea of a wall model is to predict the flow velocity inside the boundary layer, based on a measurement of the velocity farther away from the wall. The velocity at the grid nodes closest to the wall can then make use of this prediction to better simulate the flow behaviour near the wall, without requiring a fine mesh. Normally one would require the first grid node to be $y^+ \approx 1$ away from the wall to fully resolve the flow, but with the use of an accurate wall model, this distance may be much larger.

Wall models have been widely applied to Navier-Stokes based solvers, and have already been applied to the LBM by a variety of authors, such as Malaspinas and Sagaut [101], Wilhelm et al. [102] and Maeyama et al. [103]. Commercial solvers such as XFlow [104] and Powerflow [105] also employ wall models in their boundary schemes. However, to the best of the author's knowledge, all these applications have been to bounce-back or wet-node boundary schemes, and never to the immersed boundary method.

Wall models have been applied to the IBM in the Navier-Stokes framework before, by e.g. Tessicini et al. [106], Roman et al. [107] and Chen et al. [108], which shows that there is an interest in employing wall models in the IBM, although it should be noted that these were for sharp-interface IBMs. Recently, Ma et al. [109] applied a wall model to a diffusive IBM, by modifying the local viscosity with an eddy viscosity based on their wall model.

It is thus of interest to apply a wall model to the diffusive IBM in the LBM. Therefore, the following proposal of a wall-modelled, multi-direct forcing IBM in the LBM is made here. First, one can place a probing point a few lattice units of distance away from the wall, in the direction normal to the boundary surface, at which the fluid velocity is measured. This velocity can then be used to reconstruct the velocity profile along the direction normal to the wall, using a suitable wall model. Now, suppose that it is found that the velocity obtained at $\boldsymbol{x}_i$ by a wall model is $\boldsymbol{u}_{i,\mathrm{wm}}(\boldsymbol{x}_i)$. Then Equation (4.6), which spreads the velocity from the Eulerian grid to the Lagrangian boundary marker, could be modified from

$$\boldsymbol{U}_k(\boldsymbol{X}_k) = \sum_i \boldsymbol{u}_i(\boldsymbol{x}_i) D(\boldsymbol{X}_k(t) - \boldsymbol{x}_i) \tag{4.6}$$

to

$$\hat{\boldsymbol{U}}_k(\boldsymbol{X}_k) = \sum_i \left(\boldsymbol{u}_i(\boldsymbol{x}_i) - \boldsymbol{u}_{i,\mathrm{wm}}(\boldsymbol{x}_i)\right) D(\boldsymbol{X}_k(t) - \boldsymbol{x}_i), \tag{11.1}$$

and modify Equation (4.7) from

$$\boldsymbol{F}_k^{(m+1)} = \rho \frac{\boldsymbol{U}_k - \boldsymbol{U}_k^{(m)}}{\Delta t} \tag{4.7}$$

to

$$\boldsymbol{F}_k^{(m+1)} = \rho \frac{-\hat{\boldsymbol{U}}_k^{(m)}}{\Delta t}. \tag{11.2}$$

Essentially, instead of spreading the velocity directly from the Eulerian nodes and computing the deficit with the desired boundary velocity at the Lagrangian marker to compute the required corrective force $\boldsymbol{F}_k^{(m+1)}$, Equation (11.1) would spread the velocity deficit at each Eulerian node with respect to the ‘desired’ velocity from the wall-model, and compute the required corrective force based on this interpolated deficit $\hat{\boldsymbol{U}}_k$. Note that the value of $\boldsymbol{U}_k$ is accounted for in the computation of $\boldsymbol{u}_{i,\mathrm{wm}}(\boldsymbol{x}_i)$, and taking $\boldsymbol{u}_{i,\mathrm{wm}}(\boldsymbol{x}_i) = \boldsymbol{U}_k$ would return the original algorithm without a wall model. The intent of this modification is to ensure that the velocity at the nodes closed to the boundary are forced towards the velocity predicted by a wall-model, rather than the velocity of the boundary itself.

Implementation and validation of this method could offer a valuable improvement to the IBM-LBM. However, it should be noted that the methodology above can only be applied to the IBM, and not the IIM. After all, the IIM is based on interpolation of the populations, and it is thus not directly obvious how the approach above should be modified to fit into an IIM. A final recommendation is therefore, if the proposal listed above is successful, to develop, implement and validate a similar methodology for the IIM.

11.2. Recommendations regarding lattice Boltzmann method

For the lattice Boltzmann method itself, a single recommendation is put forth here, regarding the use of MRT models discussed in Section 3.5. Several authors have shown that the choice of relaxation parameter is usually performed ad hoc, using values of unity for free relaxation parameters or using values of different authors [70, 110–113], or presented without notable corroboration [114, 115]. Meanwhile, the effect of the free relaxation parameters can be significant, as demonstrated by various authors, such as Wu and Shao [110], Luo et al. [57], Yoshida and Nagaoka [111] and Chávez et al. [116].

Recently, an adaptive algorithm was developed by Li et al. [117] proposed an adaptive algorithm that adjusted the free relaxation parameters based on the local time rate of change of the various moments at a certain grid node. Although qualitatively good results seemed to have been obtained, more research is required to quantify its performance and stability compared to existing, fixed sets of relaxation parameters. In any case, promise is shown in adjusting the relaxation parameters to improve the accuracy of the simulation, and a more exhaustive study in the effect of the relaxation parameters from a practical point of view would be a worthwhile endeavour.

11.3. Recommendations regarding LaBIB-FSI

Finally, some recommendations can be made regarding the LaBIB-FSI solver. First, a high-priority improvement would be to add support for compensation for the added mass effect in the IBM and IIM procedures. It is discussed in Section 8.3.3 that the omission of a correction for the added mass effect likely degraded the quality of the solution of the FSI3-benchmark, and was the likely cause of a phase-shift between the time-history of the drag force. Therefore, it is imperative that the solver is corrected for this.

However, it should be noted that a general procedure for this correction is not trivial. After all, although

$$\boldsymbol{F}_{\text{in}}(t) = \rho_f \frac{d}{dt} \int\limits_{\boldsymbol{x}\in\Omega(t)} \boldsymbol{u}(\boldsymbol{x}, t)\, d\boldsymbol{x} \tag{8.4}$$

yields information on the *total* force of the added mass, it does not provide any information on its distribution, and how the boundary force found by the IBM/IIM should be compensated for. After all, for an arbitrarily shaped, closed surface, one could easily compute the effect of the added mass enclosed in a small volume in the interior of the body. However, it then remains a question how the boundary force distribution should be adjusted based on the added mass of this internal volume.

Nonetheless, for the cantilever beam in the FSI3 benchmark, the correction would be easy to implement. The current structural model reduces the beam to a single, one-dimensional line, and one could easily construct a lumped mass model of the added mass effect, and adjust the force distribution at each structural node based on the element volume, element acceleration and fluid density. Doing so would allow LaBIB-FSI to be properly validated for the FSI3 benchmark.

Secondly, to improve the accuracy at higher Reynolds number flow without requiring a mesh that becomes extremely fine, it is recommended that an eddy-viscosity model is implemented in the fluid solver. Eddy-viscosity models have already been widely applied in the LBM, e.g by Uphoff [25], Suga et al. [118], and Gkoudesnes and Deiterding [119]. Eddy-viscosity models are simple to implement in the LBM, as the eddy-viscosity is simply added to the kinematic viscosity that is used to determine the relaxation parameter during the collision process.

However, it should be noted that not many authors have actually evaluated the benefits of an eddy-viscosity model in the LBM in comparison to a ‘bare’ LBM. It was shown by Uphoff [25] that the two-relaxation time cascaded LBM produced similar accuracy for a turbulent channel flow compared to two-relaxation time or multi-relaxation time models that employed a WALE or Smaogirnksy eddy viscosity model. Similarly, Geier et al. [120] showed recently that the WALE model does not necessary increase the accuracy at low Reynolds numbers in a cumulant MRT model. Therefore, it is recommended to investigate and quantify the benefits that an eddy viscosity model may bring to the LBM in more detail.

Furthermore, the fluid solver should be coupled to a higher fidelity structural solver. As noted in Section 8.2, the current structural solver, which relies on reducing a cantilever beam to a single, one-dimensional line that connects several beam elements, appears reasonably accurate, but still shows an error of approximately 5% for a simple validation case of a beam vibrating under its own gravitational weight. This could presumably be improved by using a more sophisticated structural solver. This would likely also allow the support for more complex structural bodies.

Finally, to utilise the potential for massive parallelisation of the LBM, it is suggested to implement CUDA-support into the LaBIB-FSI solver. Currently, the LaBIB-FSI solver uses `OpenMP` to parallelise the fluid solver over multiple CPU-cores, but naturally, using a GPU would allow even larger-scale parallelisation. One difficulty here is that the LaBIB-FSI solver uses some functionality that is exclusive to the `C++20`-standard, whereas the most recent release of the `nvcc`-compiler only support up to `C++17`. However, it can be considered likely that `C++20`-support will soon come to `nvcc`.

11.4. Summary of recommendations

To summarise, the following actionable recommendations were set forth in this chapter:

- Explore different time-stepping schemes for the IIM-LBM, to utilise the control the IIM-LBM yields compared to the IBM-LBM.

- Combine the work by Tao et al. [76], who developed an IBM where the populations are directly corrected for the presence of a boundary (rather than through a feedback force at the boundary), with the IIM-LBM developed in this thesis.
- Implement and validate the modification to the IBM-LBM described in Section 11.1, which would implement a wall model into the IBM-LBM.
- Develop, implement and validate a similar methodology for the IIM-LBM.
- Examine, from a practical point of view, what the effect is of choosing different relaxation parameters in MRT models in the LBM.
- Implement a correction for the added mass effect in the LaBIB-FSI solver directly.
- Implement an eddy-viscosity model to improve stability and accuracy at higher Reynolds numbers with coarser meshes.
- Investigate and quantify the results of implementing such an eddy-viscosity model.
- Couple the LaBIB-FSI solver to a more sophisticated structural model.
- Implement CUDA-support in the LaBIB-FSI solver, to utilise the potential for massive parallelisation of the LBM.

IV

Appendices

A

Supplementary derivations

A.1. Example of immersed interface method to an elliptic differential equation

The idea of the immersed interface method first mentioned in Chapter 2 and discussed extensively in Chapter 5 can be illustrated via an application of it to a one-dimensional elliptic differential equation. In particular, consider the differential equation

$$\frac{d^2u}{dx^2} = f(x) \tag{A.1}$$

$$f(x) = K\delta(x - \bar{x}) \tag{A.2}$$

$$u(0) = 0 \tag{A.3}$$

$$\frac{\partial u}{\partial x}(1) = -1, \tag{A.4}$$

where $f(x)$ is a point source applying at $x = \bar{x}$, with some magnitude K. Clearly, this problem contains a discontinuity in the forcing. Consider a mesh discretisation consisting of the nodes $x_0, ..., x_n$ with $x_i = i/n$. If one was to use a central-difference scheme, one would simply discretise the second derivative as

$$\frac{d^2u}{dx^2} \approx \left.\frac{d^2u}{dx^2}\right|_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}, \tag{A.5}$$

where $h = 1/n$. Suppose that $x_j < \bar{x} < x_{j+1}$ for some value of j. The above discretisation can then be freely applied at all nodes $x_1, ..., x_{j-1}, x_{j+2}, ..., x_{n-1}$, as these regions are unaffected by the point source. For the nodes x_0 and x_n, an equation may easily be derived based on the Dirichlet and Neumann boundary conditions applying at those points. For the nodes x_j and x_{j+1} however, a special discretisation should be set up. Note that Equation (A.5) may be written as

$$\left.\frac{d^2u}{dx^2}\right|_i \approx \frac{\frac{u_{i+1}-u_i}{h} - \frac{u_i - u_{i-1}}{h}}{h} = \frac{\left.\frac{du}{dx}\right|_{i+1/2} - \left.\frac{du}{dx}\right|_{i-1/2}}{h}, \tag{A.6}$$

where $du/dx|_i$ is the central difference for the first derivative,

$$\left.\frac{du}{dx}\right|_i = \frac{u_{i+1/2} - u_{i-1/2}}{h}. \tag{A.7}$$

Clearly, $d^2u/dx^2\big|_i$ should approximate the change in the first derivative between the interval on its right and on its left. Now, consider the integration of Equation (A.1) in a very small region around $\bar{x}$, i.e. between $\bar{x} - \delta x$

and $\bar{x}+\delta x$. We obtain

$$\int_{\bar{x}-\delta x}^{\bar{x}+\delta x} \frac{d^2u}{dx^2}\,dx = \int_{\bar{x}-\delta x}^{\bar{x}+\delta x} K\delta\left(x-\bar{x}\right)dx \tag{A.8}$$

$$\left[\frac{du}{dx}\right]_{\bar{x}-\delta x}^{\bar{x}+\delta x} = K. \tag{A.9}$$

Thus, in the limit of $\delta x \to 0$, it is known that du/dx experiences a jump equal to

$$[\![\frac{du}{dx}]\!]_{\bar{x}} = K, \tag{A.10}$$

where $[\![\cdot]\!]_x$ indicates the jump in a certain value at location x, i.e.

$$[\![f]\!]_x = \lim_{\delta x\to 0} f\left(x+\delta x\right) - \lim_{\delta x\to 0} f\left(x-\delta x\right). \tag{A.11}$$

Now, suppose $\bar{x}-x_j < 1/2$, i.e., the source is located closer to x_j than to x_{j+1}. In that case, the first derivative du/dx has undergone the jump in Equation (A.10) between its approximations $du/dx|_{j-1/2}$ and $du/dx|_{j+1/2}$. Thus, to obtain the slope of the first derivative, it would be more accurate to express the second derivative as

$$\left.\frac{d^2u}{dx^2}\right|_j \approx \frac{\left.\frac{du}{dx}\right|_{i+1/2} - \left.\frac{du}{dx}\right|_{i-1/2} - [\![\frac{du}{dx}]\!]_{\bar{x}}}{h} = \frac{\left.\frac{du}{dx}\right|_{i+1/2} - \left.\frac{du}{dx}\right|_{i-1/2} - K}{h}, \tag{A.12}$$

so as to exclude the effect that the jump has on the average slope in the first derivative. Clearly, it is similar to an immersed boundary method as the grid does not conform to the boundary or the application of a point force.

Bibliography

[1] Bin Abas, M. F., Bin Mohd Rafie, A. S., Bin Yusoff, H., et al., *Flapping wing micro-aerial-vehicle: Kinematics, membranes, and flapping mechanisms of ornithopter and insect flight*, Chinese Journal of Aeronautics, 2016, vol. 29, no. 5, pp. 1159–1177, doi:https://doi.org/10.1016/j.cja.2016.08.003, URL https://www.sciencedirect.com/science/article/pii/S1000936116300978.

[2] Krüger, T., Kusumaatmaja, H., Kuzmin, A., et al., *The Lattice Boltzmann Method*, Springer, Switzerland, 2017.

[3] Guo, Z., and Shu, C., *Lattice Boltzmann Method and Its Applications in Engineering*, World Scientific Publishing Company, Singapore, 2013.

[4] Peskin, C. S., *Numerical analysis of blood flow in the heart*, Journal of Computational Physics, 1977, vol. 25, no. 3, pp. 220–252, doi:https://doi.org/10.1016/0021-9991(77)90100-0, URL https://www.sciencedirect.com/science/article/pii/0021999177901000.

[5] Feng, Z.-G., and Michaelides, E. E., *The Immersed Boundary-Lattice Boltzmann Method for Solving Fluid-Particles Interaction Problems*, J. Comput. Phys., 4 2004, vol. 195, no. 2, p. 602–628, doi:10.1016/j.jcp.2003.10.013, URL https://doi-org.tudelft.idm.oclc.org/10.1016/j.jcp.2003.10.013.

[6] Leveque, R. J., and Li, Z., *The Immersed Interface Method for Elliptic Equations with Discontinuous Coefficients and Singular Sources*, SIAM Journal on Numerical Analysis, 1994, vol. 31, no. 4, pp. 1019–1044, URL http://www.jstor.org/stable/2158113.

[7] Qin, J., Kolahdouz, E. M., and Griffith, B. E., *An immersed interface-lattice Boltzmann method for fluid-structure interaction*, Journal of Computational Physics, 2021, vol. 428, p. 109807, doi:https://doi.org/10.1016/j.jcp.2020.109807, URL https://www.sciencedirect.com/science/article/pii/S0021999120305817.

[8] Turek, S., and Hron, J., *Proposal for Numerical Benchmarking of Fluid-Structure Interaction between an Elastic Object and Laminar Incompressible Flow*, in Bungartz, H.-J., and Schäfer, M. (eds.), *Fluid-Structure Interaction*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 371–385.

[9] Shyy, W., Lian, Y., Tang, J., et al., *Aerodynamics of Low Reynolds Number Flyers*, Cambridge Aerospace Series, Cambridge University Press, 2007, doi:10.1017/CBO9780511551154.003.

[10] De Rosis, A., Ubertini, S., and Ubertini, F., *A Comparison Between the Interpolated Bounce-Back Scheme and the Immersed Boundary Method to Treat Solid Boundary Conditions for Laminar Flows in the Lattice Boltzmann Framework*, Journal of Scientific Computing, 12 2014, vol. 61, no. 3, p. 477–489, doi:10.1007/s10915-014-9834-0, URL https://doi-org.tudelft.idm.oclc.org/10.1007/s10915-014-9834-0.

[11] Hino, H., and Inamuro, T., *Turning flight simulations of a dragonfly-like flapping wing-body model by the immersed boundary-lattice Boltzmann method*, Fluid Dynamics Research, 09 2018, vol. 50, no. 6, p. 065501, doi:10.1088/1873-7005/aad78c, URL https://doi.org/10.1088/1873-7005/aad78c.

[12] Rutkowski, M., Gryglas, W., Szumbarski, J., et al., *Open-loop optimal control of a flapping wing using an adjoint Lattice Boltzmann method*, Computers & Mathematics with Applications, 2020, vol. 79, no. 12, pp. 3547–3569, doi:https://doi.org/10.1016/j.camwa.2020.02.020, URL https://www.sciencedirect.com/science/article/pii/S0898122120300870.

[13] Castro, S. G. P., *General-purpose finite element solver based on Python/Cython*, 2022, doi:10.5281/zenodo.6573490, URL https://doi.org/10.5281/zenodo.6573490.

[14] Lighthill, M. J., *On the Weis-Fogh mechanism of lift generation*, Journal of Fluid Mechanics, 1973, vol. 60, no. 1, p. 1–17, doi:10.1017/S0022112073000017.

[15] Miller, L., and Peskin, C., *Flexible clap and fling in tiny insect flight*, Journal of Experimental Biology, 2009, vol. 212, pp. 3076 – 3090.

[16] Liu, H., and Aono, H., *Size effects on insect hovering aerodynamics: an integrated computational study*, Bioinspiration & Biomimetics, 3 2009, vol. 4, no. 1, p. 015002, doi:10.1088/1748-3182/4/1/015002, URL `https://doi.org/10.1088/1748-3182/4/1/015002`.

[17] Decroon, G., Percin, M., Remes, B., et al., *The Delfly: Design, aerodynamics, and artificial intelligence of a flapping wing robot*, Springer, 2016, doi:10.1007/978-94-017-9208-0, URL `https://research.tudelft.nl/en/publications/the-delfly-design-aerodynamics-and-artificial-intelligence-of-a-f`, harvest.

[18] Gillebaart, T., *Influence of flexibility on the clap and peel movement of the DelFly II*, Ph.D. thesis, TU Delft University of Technology, 2011.

[19] Tay, W. B., van Oudheusden, B. W., and Bijl, H., *Numerical simulation of X-wing type biplane flapping wings in 3D using the immersed boundary method*, Bioinspiration & biomimetics, 9 2014, vol. 9, no. 3, p. 036001, doi:10.1088/1748-3182/9/3/036001, URL `http://www.ncbi.nlm.nih.gov/pubmed/24584155`.

[20] Tay, W., van Oudheusden, B., and Bijl, H., *Numerical simulation of a flapping four-wing micro-aerial vehicle*, Journal of Fluids and Structures, 2015, vol. 55, pp. 237–261, doi:10.1016/j.jfluidstructs.2015.03.003.

[21] Tay, W., Deng, S., van Oudheusden, B., et al., *Validation of immersed boundary method for the numerical simulation of flapping wing flight*, Computers & Fluids, 2015, vol. 115, pp. 226–242, doi:10.1016/j.compfluid.2015.04.009.

[22] Arumuga Perumal, D., and Dass, A. K., *A Review on the development of lattice Boltzmann computation of macro fluid flows and heat transfer*, Alexandria Engineering Journal, 2015, vol. 54, no. 4, pp. 955–971, doi:https://doi.org/10.1016/j.aej.2015.07.015, URL `https://www.sciencedirect.com/science/article/pii/S1110016815001362`.

[23] Hou, S., Sterling, J., Chen, S., et al., *A Lattice Boltzmann Subgrid Model for High Reynolds Number Flows*, Cellular Automata and Lattice Gases, 1996.

[24] Krafzcyk, M., Tölke, J., and Luo, L.-S., *Large-Eddy Simulations with a Multiple-Relaxation-Time LBE MODEL*, International Journal of Modern Physics B, 2003, vol. 17, no. 01n02, pp. 33–39, doi:10.1142/S0217979203017059, URL `https://doi.org/10.1142/S0217979203017059`, `https://doi.org/10.1142/S0217979203017059`.

[25] Uphoff, S., *Development and Validation of turbulence models for Lattice Boltzmann schemes*, Ph.D. thesis, Technical University of Braunschweig, 1 2013, doi:10.24355/dbbs.084-201401141113-0, URL `https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00055260`.

[26] Smagorinsky, J., *General Circulation Experiments with the Primitive Equations: I. The Basic Experiment*, Monthly Weather Review, 1963, vol. 91, no. 3, pp. 99 – 164, doi:10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2, URL `https://journals.ametsoc.org/view/journals/mwre/91/3/1520-0493_1963_091_0099_gcewtp_2_3_co_2.xml`.

[27] Van Driest, E. R., *On Turbulent Flow Near a Wall*, Journal of the Aeronautical Sciences, 1956, vol. 23, no. 11, pp. 1007–1011, doi:10.2514/8.3713, URL `https://doi.org/10.2514/8.3713`, `https://doi.org/10.2514/8.3713`.

[28] Nicoud, F., and Ducros, F., *Subgrid-Scale Stress Modelling Based on the Square of the Velocity Gradient Tensor*, Flow, Turbulence and Combustion, 1999, vol. 62, pp. 183–200.

[29] Vreman, A. W., *The filtering analog of the variational multiscale method in large-eddy simulation*, Physics of Fluids, 2003, vol. 15, no. 8, pp. L61–L64, doi:10.1063/1.1595102, URL `https://doi.org/10.1063/1.1595102`, `https://doi.org/10.1063/1.1595102`.

[30] Pradeep Kumar, S., De, A., and Das, D., *Investigation of flow field of clap and fling motion using immersed boundary coupled lattice Boltzmann method*, Journal of Fluids and Structures, 2015, vol. 57, pp. 247–263, doi:https://doi.org/10.1016/j.jfluidstructs.2015.06.008, URL `https://www.sciencedirect.com/science/article/pii/S0889974615001383`.

[31] De Rosis, A., Falcucci, G., Ubertini, S., et al., *Aeroelastic study of flexible flapping wings by a coupled lattice Boltzmann-finite element approach with immersed boundary method*, Journal of Fluids and Structures, 2014, vol. 49, pp. 516–533, doi:https://doi.org/10.1016/j.jfluidstructs.2014.05.010, URL `https://www.sciencedirect.com/science/article/pii/S0889974614001169`.

[32] De Rosis, A., Ubertini, S., and Ubertini, F., *A partitioned approach for two-dimensional fluid–structure interaction problems by a coupled lattice Boltzmann-finite element method with immersed boundary*, Journal of Fluids and Structures, 2014, vol. 45, pp. 202–215, doi:https://doi.org/10.1016/j.jfluidstructs.2013.12.009, URL `https://www.sciencedirect.com/science/article/pii/S0889974613002910`.

[33] Tiwari, A., and Vanka, S. P., *A ghost fluid Lattice Boltzmann method for complex geometries*, International Journal for Numerical Methods in Fluids, 2012, vol. 69, no. 2, pp. 481–498, doi:https://doi.org/10.1002/fld.2573, URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2573`, `https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.2573`.

[34] Huang, W.-X., and Tian, F.-B., *Recent trends and progress in the immersed boundary method*, Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 2019, vol. 233, no. 23-24, pp. 7617–7636, doi:10.1177/0954406219842606, URL `https://doi.org/10.1177/0954406219842606`, `https://doi.org/10.1177/0954406219842606`.

[35] Kang, S. K., and Hassan, Y. A., *A comparative study of direct-forcing immersed boundary-lattice Boltzmann methods for stationary complex boundaries*, International Journal for Numerical Methods in Fluids, 2011, vol. 66, no. 9, pp. 1132–1158, doi:https://doi.org/10.1002/fld.2304, URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2304`, `https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.2304`.

[36] Lee, L., and LeVeque, R. J., *An immersed interface method for incompressible Navier–Stokes equations*, SIAM Journal on Scientific Computing, 2003, vol. 25, no. 3, pp. 832–856.

[37] Xu, S., and Wang, Z. J., *An immersed interface method for simulating the interaction of a fluid with moving boundaries*, Journal of Computational Physics, 2006, vol. 216, no. 2, pp. 454–493.

[38] Vaughan, B., Jr., J., Smith, B., et al., *A comparison of the extended finite element method with the immersed interface method for elliptic equations with discontinuous coefficients and singular sources*, Communications in Applied Mathematics and Computational Science, 2006, vol. 1, pp. 207–228, doi:10.2140/camcos.2006.1.207.

[39] Yang, X., Zhang, X., Li, Z., et al., *A smoothing technique for discrete delta functions with application to immersed boundary method in moving boundary simulations*, Journal of Computational Physics, 2009, vol. 228, no. 20, pp. 7821–7836, doi:https://doi.org/10.1016/j.jcp.2009.07.023, URL `https://www.sciencedirect.com/science/article/pii/S0021999109004136`.

[40] Finkelstein, L., *Structure of the Boltzmann Collision Operator*, The Physics of Fluids, 1965, vol. 8, no. 3, pp. 431–436, doi:10.1063/1.1761242, URL `https://aip.scitation.org/doi/abs/10.1063/1.1761242`, `https://aip.scitation.org/doi/pdf/10.1063/1.1761242`.

[41] Belmont, G., Rezeau, L., Riconda, C., et al., *3 - Kinetic Theory of Plasma*, in Belmont, G., Rezeau, L., Riconda, C., et al. (eds.), *Introduction to Plasma Physics*, Elsevier, 2019, pp. 57–74, doi:https://doi.org/10.1016/B978-1-78548-306-6.50003-2, URL `https://www.sciencedirect.com/science/article/pii/B9781785483066500032`.

[42] Bhatnagar, P. L., Gross, E. P., and Krook, M., *A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems*, Phys. Rev., 5 1954, vol. 94, pp. 511–525, doi: 10.1103/PhysRev.94.511, URL `https://link.aps.org/doi/10.1103/PhysRev.94.511`.

[43] Andries, P., Aoki, K., and Perthame, B., *A consistent BGK-type model for gas mixtures*, Journal of Statistical Physics, 2002, vol. 106, no. 5, pp. 993–1018.

[44] Shan, X., Yuan, X.-F., and Chen, H., *Kinetic theory representation of hydrodynamics: a way beyond the Navier–Stokes equation*, Journal of Fluid Mechanics, 2006, vol. 550, p. 413–441, doi:10.1017/S0022112005008153.

[45] Liu, Q., and Pierce, D. A., *A Note on Gauss-Hermite Quadrature*, Biometrika, 1994, vol. 81, no. 3, pp. 624–629, URL `http://www.jstor.org/stable/2337136`.

[46] Abramowitz, M., *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*, Dover Publications, Inc., USA, 1974.

[47] Ubertini, S., Asinari, P., and Succi, S., *Three ways to lattice Boltzmann: A unified time-marching picture*, Phys. Rev. E, 1 2010, vol. 81, p. 016311, doi:10.1103/PhysRevE.81.016311, URL `https://link.aps.org/doi/10.1103/PhysRevE.81.016311`.

[48] Guo, Z., Zheng, C., and Shi, B., *Discrete lattice effects on the forcing term in the lattice Boltzmann method*, Phys. Rev. E, 4 2002, vol. 65, p. 046308, doi:10.1103/PhysRevE.65.046308, URL `https://link.aps.org/doi/10.1103/PhysRevE.65.046308`.

[49] He, X., Shan, X., and Doolen, G. D., *Discrete Boltzmann equation model for nonideal gases*, Phys. Rev. E, 1 1998, vol. 57, pp. R13–R16, doi:10.1103/PhysRevE.57.R13, URL `https://link.aps.org/doi/10.1103/PhysRevE.57.R13`.

[50] Caiazzo, A., *Analysis of Lattice Boltzmann Initialization Routines*, Journal of Statistical Physics, 10 2005, vol. 121, no. 1-2, pp. 37–48, doi:10.1007/s10955-005-7010-5.

[51] Mei, R., Luo, L.-S., Lallemand, P., et al., *Consistent initial conditions for lattice Boltzmann simulations*, Computers & Fluids, 2006, vol. 35, pp. 855–862.

[52] Wagner, A. J., *Thermodynamic consistency of liquid-gas lattice Boltzmann simulations*, Phys. Rev. E, 11 2006, vol. 74, p. 056703, doi:10.1103/PhysRevE.74.056703, URL `https://link.aps.org/doi/10.1103/PhysRevE.74.056703`.

[53] Ladd, A., and Verberg, R., *Lattice-Boltzmann Simulations of Particle-Fluid Suspensions*, Journal of Statistical Physics, 09 2001, vol. 104, pp. 1191–1251, doi:10.1023/A:1010414013942.

[54] Luo, L.-S., *Unified Theory of Lattice Boltzmann Models for Nonideal Gases*, Phys. Rev. Lett., 8 1998, vol. 81, pp. 1618–1621, doi:10.1103/PhysRevLett.81.1618, URL `https://link.aps.org/doi/10.1103/PhysRevLett.81.1618`.

[55] Bawazeer, S., Baakeem, S., and Mohamad, A., *A Critical Review of Forcing Schemes in Lattice Boltzmann Method: 1993-2019*, Archives of Computational Methods in Engineering, 01 2021, doi:10.1007/s11831-021-09535-4.

[56] Viggen, E. M., *The lattice Boltzmann method with applications in acoustics*, Master's thesis, Norwegian University of Science and Technology, Trondheim, 2009.

[57] Luo, L.-S., Liao, W., Chen, X., et al., *Numerics of the lattice Boltzmann method: Effects of collision models on the lattice Boltzmann simulations*, Phys. Rev. E, 5 2011, vol. 83, p. 056710, doi:10.1103/PhysRevE.83.056710, URL `https://link.aps.org/doi/10.1103/PhysRevE.83.056710`.

[58] Dellar, P. J., *Nonhydrodynamic modes and a priori construction of shallow water lattice Boltzmann equations*, Phys. Rev. E, 02 2002, vol. 65, p. 036309, doi:10.1103/PhysRevE.65.036309, URL `https://link.aps.org/doi/10.1103/PhysRevE.65.036309`.

[59] Lallemand, P., and Luo, L.-S., *Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability*, Phys. Rev. E, 6 2000, vol. 61, pp. 6546–6562, doi:10.1103/PhysRevE.61.6546, URL `https://link.aps.org/doi/10.1103/PhysRevE.61.6546`.

[60] Coreixas, C., Chopard, B., and Latt, J., *Comprehensive comparison of collision models in the lattice Boltzmann framework: Theoretical investigations*, Phys. Rev. E, 9 2019, vol. 100, p. 033305, doi:10.1103/PhysRevE.100.033305, URL `https://link.aps.org/doi/10.1103/PhysRevE.100.033305`.

[61] Arfken, G. B., Weber, H. J., and Spector, D., *Mathematical Methods for Physicists, 4th ed .*, American Journal of Physics, feb 1999, vol. 67, no. 2, pp. 165–169, doi:10.1119/1.19217.

[62] Bouzidi, M., d'Humières, D., Lallemand, P., et al., *Lattice Boltzmann Equation on a Two-Dimensional Rectangular Grid*, J. Comput. Phys., Sep. 2001, vol. 172, no. 2, p. 704–717, doi:10.1006/jcph.2001.6850, URL `https://doi-org.tudelft.idm.oclc.org/10.1006/jcph.2001.6850`.

[63] d'Humières, D., Ginzburg, I., Krafczyk, M., et al., *Multiple-Relaxation-Time Lattice Boltzmann Models in Three Dimensions*, Philosophical Transactions: Mathematical, Physical and Engineering Sciences, 2002, vol. 360, no. 1792, pp. 437–451, URL `http://www.jstor.org/stable/3066323`.

[64] Ginzburg, I., *Equilibrium-type and link-type lattice Boltzmann models for generic advection and anisotropic-dispersion equation*, Advances in Water Resources, 2005, vol. 28, no. 11, pp. 1171–1195, doi:https://doi.org/10.1016/j.advwatres.2005.03.004, URL `https://www.sciencedirect.com/science/article/pii/S0309170805000874`.

[65] Ginzburg, I., Verhaeghe, F., and d'Humières, D., *Study of simple hydrodynamic solutions with the two-relaxation-times lattice Boltzmann scheme*, Communications in Computational Physics, 2008, vol. 3, pp. 519–581.

[66] Ginzburg, I., and d'Humières, D., *Optimal Stability of Advection-Diffusion Lattice Boltzmann Models with Two Relaxation Times for Positive/Negative Equilibrium*, Journal of Statistical Physics, 2010, vol. 139, pp. 1090–1143.

[67] Geier, M., Greiner, A., and Korvink, J. G., *Cascaded digital lattice Boltzmann automata for high Reynolds number flow*, Phys. Rev. E, 6 2006, vol. 73, p. 066705, doi:10.1103/PhysRevE.73.066705, URL `https://link.aps.org/doi/10.1103/PhysRevE.73.066705`.

[68] Geier, M. C., *Ab initio derivation of the cascaded lattice boltzmann automaton*, Ph.D. thesis, University of Freiburg - IMTEK, 2006.

[69] Lycett-Brown, D., and Luo, K. H., *Multiphase cascaded lattice Boltzmann method*, Computers & Mathematics with Applications, 2014, vol. 67, no. 2, pp. 350–362, doi:https://doi.org/10.1016/j.camwa.2013.08.033, URL `https://www.sciencedirect.com/science/article/pii/S0898122113005403`, mesoscopic Methods for Engineering and Science (Proceedings of ICMMES-2012, Taipei, Taiwan, 23–27 July 2012).

[70] Geier, M., Schönherr, M., Pasquali, A., et al., *The cumulant lattice Boltzmann equation in three dimensions: Theory and validation*, Computers & Mathematics with Applications, 2015, vol. 70, no. 4, pp. 507–547, doi:https://doi.org/10.1016/j.camwa.2015.05.001, URL `https://www.sciencedirect.com/science/article/pii/S0898122115002126`.

[71] Goraki Fard, E., *Cumulant LBM approach for Large Eddy Simulation of Dispersion Microsystems*, Ph.D. thesis, Technical University of Braunschweig, 3 2015, doi:10.24355/dbbs.084-201505271009-0, URL `https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00060125`.

[72] Peskin, C. S., *The immersed boundary method*, Acta Numerica, 2002, vol. 11, p. 479–517, doi:10.1017/S0962492902000077.

[73] Luo, K., Wang, Z., Fan, J., et al., *Full-scale solutions to particle-laden flows: Multidirect forcing and immersed boundary method*, Phys. Rev. E, 12 2007, vol. 76, p. 066709, doi:10.1103/PhysRevE.76.066709, URL `https://link.aps.org/doi/10.1103/PhysRevE.76.066709`.

[74] Mohd-Yusof, J., *Combined immersed boundaries/B-splines methods for simulations of flows in complex geometries*, Annual Research Briefs, 1997.

[75] Zhou, Q., and Fan, L.-S., *A second-order accurate immersed boundary-lattice Boltzmann method for particle-laden flows*, Journal of computational physics, 2014, vol. 268, pp. 269–301.

[76] Tao, S., He, Q., Chen, B., et al., *A distribution function correction-based immersed boundary- lattice Boltzmann method with truly second-order accuracy for fluid-solid flows*, arXiv: Computational Physics, 2018.

[77] Kanwal, R., *Generalized Functions: Theory and Applications*, Generalized Functions, Birkhäuser Boston, 2004, URL `https://books.google.nl/books?id=LXP_8Y0jH88C`.

[78] Berberian, S., *Fundamentals of Real Analysis*, Universitext, Springer New York, 2013, URL `https://books.google.nl/books?id=MzQ6JA6SiHYC`.

[79] Luo, Y., *An Efficient 3 D Timoshenko Beam Element with Consistent Shape Functions*.

[80] Geller, S., Tölke, J., and Krafczyk, M., *Lattice-Boltzmann Method on Quadtree-Type Grids for Fluid-Structure Interaction*, in Bungartz, H.-J., and Schäfer, M. (eds.), *Fluid-Structure Interaction*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 270–293.

[81] Megson, T., *Aircraft Structures for Engineering Students*, Butterworth-Heinemann, Oxford, 2007.

[82] Newmark, N. M., *A Method of Computation for Structural Dynamics*, Journal of the Engineering Mechanics Division, 1959, vol. 85, no. 3, pp. 67–94, doi:10.1061/JMCEA3.0000098, URL `https://ascelibrary.org/doi/abs/10.1061/JMCEA3.0000098`, `https://ascelibrary.org/doi/pdf/10.1061/JMCEA3.0000098`.

[83] Wendland, H., *Error Estimates for Interpolation by Compactly Supported Radial Basis Functions of Minimal Degree*, Journal of Approximation Theory, 1998, vol. 93, no. 2, pp. 258–272, doi:https://doi.org/10.1006/jath.1997.3137, URL `https://www.sciencedirect.com/science/article/pii/S0021904597931373`.

[84] Buhmann, M. D., *A new class of radial basis functions with compact support*, Math. Comput., 2001, vol. 70, pp. 307–318.

[85] Lagrava, D., Malaspinas, O., Latt, J., et al., *Advances in multi-domain lattice Boltzmann grid refinement*, Journal of Computational Physics, 2012, vol. 231, no. 14, pp. 4808–4822, doi:https://doi.org/10.1016/j.jcp.2012.03.015, URL `https://www.sciencedirect.com/science/article/pii/S002199911200157X`.

[86] Dupuis, A., and Chopard, B., *Theory and applications of an alternative lattice Boltzmann grid refinement algorithm*, Phys. Rev. E, 6 2003, vol. 67, p. 066707, doi:10.1103/PhysRevE.67.066707, URL `https://link.aps.org/doi/10.1103/PhysRevE.67.066707`.

[87] Latt, J., *Hydrodynamic limit of lattice Boltzmann equations*, Ph.D. thesis, University of Geneva, 09 2007, URL `https://nbn-resolving.org/urn:nbn:ch:unige-4641`, iD: unige:464; Contient un résumé en français de 10 p.

[88] Latt, J., Chopard, B., Malaspinas, O., et al., *Straight velocity boundaries in the lattice Boltzmann method*, Phys. Rev. E, May 2008, vol. 77, p. 056703, doi:10.1103/PhysRevE.77.056703, URL `https://link.aps.org/doi/10.1103/PhysRevE.77.056703`.

[89] Lallemand, P., and Luo, L.-S., *Lattice Boltzmann method for moving boundaries*, Journal of Computational Physics, 2003, vol. 184, no. 2, pp. 406–421, doi:https://doi.org/10.1016/S0021-9991(02)00022-0, URL `https://www.sciencedirect.com/science/article/pii/S0021999102000220`.

[90] Zou, Q., and He, X., *On pressure and velocity boundary conditions for the lattice Boltzmann BGK model*, Physics of Fluids, 1997, vol. 9, no. 6, pp. 1591–1598, doi:10.1063/1.869307, URL `https://doi.org/10.1063/1.869307`, `https://doi.org/10.1063/1.869307`.

[91] Inamuro, T., Yoshino, M., and Ogino, F., *A non-slip boundary condition for lattice Boltzmann simulations*, Physics of Fluids, 1995, vol. 7, no. 12, pp. 2928–2930, doi:10.1063/1.868766, URL `https://doi.org/10.1063/1.868766`, `https://doi.org/10.1063/1.868766`.

[92] Botella, O., and Peyret, R., *Benchmark spectral results on the lid-driven cavity flow*, Computers & Fluids, 1998, vol. 27, no. 4, pp. 421–433, doi:https://doi.org/10.1016/S0045-7930(98)00002-4, URL `https://www.sciencedirect.com/science/article/pii/S0045793098000024`.

[93] Wu, J., and Shu, C., *Implicit velocity correction-based immersed boundary-lattice Boltzmann method and its applications*, Journal of Computational Physics, 2009, vol. 228, no. 6, pp. 1963–1979, doi:https://doi.org/10.1016/j.jcp.2008.11.019, URL `https://www.sciencedirect.com/science/article/pii/S0021999108006116`.

[94] Trapani, G., Brionnaud, R. M., and Holman, D. M., *Non-linear Fluid-Structure Interaction using a Partitioned Lattice Boltzmann - FEA approach*, doi:10.2514/6.2016-3636, URL `https://arc.aiaa.org/doi/abs/10.2514/6.2016-3636`, `https://arc.aiaa.org/doi/pdf/10.2514/6.2016-3636`.

[95] Schäfer, M., Heck, M., and Yigit, S., *An Implicit Partitioned Method for the Numerical Simulation of Fluid-Structure Interaction*, in Bungartz, H.-J., and Schäfer, M. (eds.), *Fluid-Structure Interaction*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 171–194.

[96] Suzuki, K., and Inamuro, T., *Effect of internal mass in the simulation of a moving body by the immersed boundary method*, Computers & Fluids, 2011, vol. 49, no. 1, pp. 173–187, doi:https://doi.org/10.1016/j.compfluid.2011.05.011, URL `https://www.sciencedirect.com/science/article/pii/S0045793011001708`.

[97] DÜTSCH, H., DURST, F., BECKER, S., et al., *Low-Reynolds-number flow around an oscillating circular cylinder at low Keulegan–Carpenter numbers*, Journal of Fluid Mechanics, 1998, vol. 360, p. 249–271, doi:10.1017/S002211209800860X.

[98] Hulshoff, S., *AE2220-II: Computational Modelling: Lecture Notes*, 2016.

[99] LARSSON, J., KAWAI, S., BODART, J., et al., *Large eddy simulation with modeled wall-stress: recent progress and future directions*, Mechanical Engineering Reviews, 2016, vol. 3, no. 1, pp. 15–00418–15–00418, doi:10.1299/mer.15-00418.

[100] Bose, S. T., and Park, G. I., *Wall-Modeled Large-Eddy Simulation for Complex Turbulent Flows*, Annual Review of Fluid Mechanics, 2018, vol. 50, no. 1, pp. 535–561, doi:10.1146/annurev-fluid-122316-045241, URL `https://doi.org/10.1146/annurev-fluid-122316-045241`, `https://doi.org/10.1146/annurev-fluid-122316-045241`.

[101] Malaspinas, O., and Sagaut, P., *Wall model for large-eddy simulation based on the lattice Boltzmann method*, Journal of Computational Physics, 2014, vol. 275, pp. 25–40, doi:https://doi.org/10.1016/j.jcp.2014.06.020, URL `https://www.sciencedirect.com/science/article/pii/S0021999114004276`.

[102] Wilhelm, S., Jacob, J., and Sagaut, P., *An explicit power-law-based wall model for lattice Boltzmann method–Reynolds-averaged numerical simulations of the flow around airfoils*, Physics of Fluids, 2018, vol. 30, no. 6, p. 065111, doi:10.1063/1.5031764, URL `https://doi.org/10.1063/1.5031764`, `https://doi.org/10.1063/1.5031764`.

[103] Maeyama, H., Imamura, T., Osaka, J., et al., *Unsteady turbulent flow simulation using lattice Boltzmann method with near-wall modeling*, in *AIAA Aviation 2020 Forum*, doi:10.2514/6.2020-2565, URL `https://arc.aiaa.org/doi/abs/10.2514/6.2020-2565`, `https://arc.aiaa.org/doi/pdf/10.2514/6.2020-2565`.

[104] Pousa, A. F., *Thermal and Multiphase Validations with the Lattice Boltzmann Method Software XFlow*, Ph.D. thesis, University of Santiago de Compostella, 2015.

[105] Fares, E., *Unsteady flow simulation of the Ahmed reference body using a lattice Boltzmann approach*, Computers & Fluids, 2006, vol. 35, no. 8, pp. 940–950, doi:https://doi.org/10.1016/j.compfluid.2005.04.011, URL `https://www.sciencedirect.com/science/article/pii/S0045793005001581`, proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.

[106] Tessicini, F., Iaccarino, G., Fatica, M., et al., *Wall modeling for large-eddy simulation using an immersed boundary method*, in *CCenter for Turublence Research Annual Research Briefs 2002*.

[107] Roman, F., Armenio, V., and Fröhlich, J., *A simple wall-layer model for large eddy simulation with immersed boundary method*, Physics of Fluids, 2009, vol. 21, no. 10, p. 101701, doi:10.1063/1.3245294, URL `https://doi.org/10.1063/1.3245294`, `https://doi.org/10.1063/1.3245294`.

[108] Chen, Z. L., Hickel, S., Devesa, A., et al., *Wall modeling for implicit large-eddy simulation and immersed-interface methods*, Theoretical Computational Fluid Dynamics, 2014, vol. 28, pp. 1–21.

[109] Ma, M., Huang, W.-X., and Xu, C.-X., *A dynamic wall model for large eddy simulation of turbulent flow over complex/moving boundaries based on the immersed boundary method*, Physics of Fluids, 2019, vol. 31, no. 11, p. 115101, doi:10.1063/1.5126853, URL `https://doi.org/10.1063/1.5126853`, `https://doi.org/10.1063/1.5126853`.

[110] Wu, J.-S., and Shao, Y.-L., *Simulation of lid-driven cavity flows by parallel lattice Boltzmann method using multi-relaxation-time scheme*, International Journal for Numerical Methods in Fluids, 2004, vol. 46, no. 9, pp. 921–937, doi:https://doi.org/10.1002/fld.787, URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.787`, `https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.787`.

[111] Yoshida, H., and Nagaoka, M., *Multiple-relaxation-time lattice Boltzmann model for the convection and anisotropic diffusion equation*, Journal of Computational Physics, 2010, vol. 229, no. 20, pp. 7774–7795, doi:https://doi.org/10.1016/j.jcp.2010.06.037, URL `https://www.sciencedirect.com/science/article/pii/S0021999110003529`.

[112] Premnath, K. N., and Abraham, J., *Three-dimensional multi-relaxation time (MRT) lattice-Boltzmann models for multiphase flow*, Journal of Computational Physics, 2007, vol. 224, no. 2, pp. 539–559, doi:https://doi.org/10.1016/j.jcp.2006.10.023, URL `https://www.sciencedirect.com/science/article/pii/S0021999106004815`.

[113] De Rosis, A., and Luo, K. H., *Role of higher-order Hermite polynomials in the central-moments-based lattice Boltzmann framework*, Phys. Rev. E, 1 2019, vol. 99, p. 013301, doi:10.1103/PhysRevE.99.013301, URL `https://link.aps.org/doi/10.1103/PhysRevE.99.013301`.

[114] Liu, Q., He, Y.-L., Li, Q., et al., *A multiple-relaxation-time lattice Boltzmann model for convection heat transfer in porous media*, International Journal of Heat and Mass Transfer, 2014, vol. 73, pp. 761–775, doi:https://doi.org/10.1016/j.ijheatmasstransfer.2014.02.047, URL `https://www.sciencedirect.com/science/article/pii/S0017931014001756`.

[115] Liu, Q., and He, Y.-L., *Lattice Boltzmann simulations of convection heat transfer in porous media*, Physica A: Statistical Mechanics and its Applications, 2017, vol. 465, pp. 742–753, doi:https://doi.org/10.1016/j.physa.2016.08.010, URL `https://www.sciencedirect.com/science/article/pii/S037843711630526X`.

[116] Chávez-Modena, M., Ferrer, E., and Rubio, G., *Improving the stability of multiple-relaxation lattice Boltzmann methods with central moments*, Computers & Fluids, 2018, vol. 172, pp. 397–409, doi:https://doi.org/10.1016/j.compfluid.2018.03.084, URL `https://www.sciencedirect.com/science/article/pii/S0045793018301889`.

[117] Li, W., Chen, Y., Desbrun, M., et al., *Fast and Scalable Turbulent Flow Simulation with Two-Way Coupling*, ACM Trans. Graph., Jul. 2020, vol. 39, no. 4, doi:10.1145/3386569.3392400, URL `https://doi-org.tudelft.idm.oclc.org/10.1145/3386569.3392400`.

[118] Suga, K., Kuwata, Y., Takashima, K., et al., *A D3Q27 multiple-relaxation-time lattice Boltzmann method for turbulent flows*, Computers & Mathematics with Applications, 2015, vol. 69, no. 6, pp. 518–529, doi:https://doi.org/10.1016/j.camwa.2015.01.010, URL `https://www.sciencedirect.com/science/article/pii/S0898122115000346`.

[119] Gkoudesnes, C., and Deiterding, R., *Evaluating the lattice Boltzmann method for large eddy simulation with dynamic sub-grid scale models*, in *11th International Symposium on Turbulence and Shear Flow Phenomena (02/08/19)*, URL `https://eprints.soton.ac.uk/433587/`.

[120] Geier, M., Lenz, S., Schönherr, M., et al., *Under-resolved and large eddy simulations of a decaying Taylor-Green vortex with the cumulant lattice Boltzmann method*, Theoretical and Computational Fluid Dynamics, 4 2021, vol. 35, no. 2, pp. 169–208, doi:10.1007/s00162-020-00555-7.