Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

MAS

Modelling, Analysis and Simulation

Modelling, Analysis and Simulation

The immersed boundary method for the (2D) incompressible Navier-Stokes equations

R.J.R. vander Meûlen

REPORT MAS-E0607 JANUARY 2006

Centrum voor Wiskunde en Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3703

The immersed boundary method for the (2D) incompressible Navier-Stokes equations

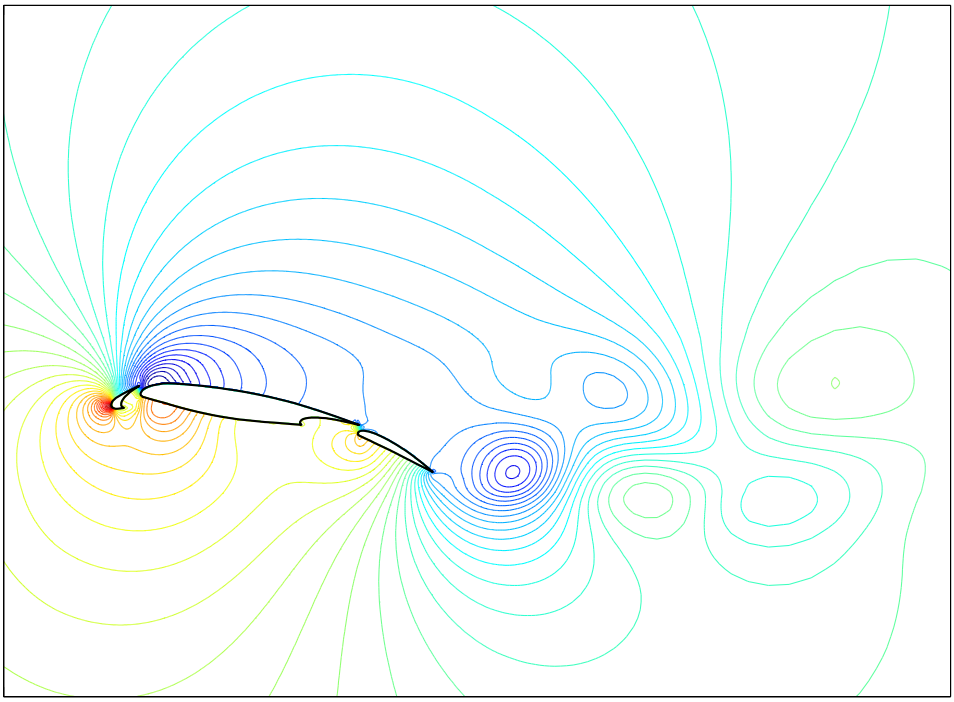
ABSTRACT
Immersed Boundary Methods (IBMs) are a class of methods in Computational Fluid Dynamics where the grids do not conform to the shape of the body. Instead they employ Cartesian meshes and alternative ways to incorporate the boundary conditions in the (discrete) governing equations. The simple grids and data structure are very well suited to handle complex geometries and moving boundaries. The main objective of this project was to investigate Immersed Boundary Methods through literature study, brief analysis and numerical experiments, to gain experience and knowledge on the topic and to lay the foundations for practical use of these methods in future research. The approach that was taken to meet the objectives can be split into three parts: a literature study, a simple 1D channel-flow study and a 2D steady Navier-Stokes study. The literature study presents the basic IBM techniques and a brief historical overview, followed by a discussion on some important properties of IBMs. Based on a structured classification of existing methods, a choice is made on the type of Immersed Boundary Methods to be explored in the 1D numerical study. The 1D study makes use of the Poiseuille flow problem as a test case, since it has an analytical solution which allows us to calculate the absolute error made by the developed IBMs. The study of three new and two existing 1D methods and their derived variants reveals their accuracy on different grids and shows that this accuracy can be substantially affected by the position of an immersed boundary with respect to the neighboring grid points. The construction of a steady 2D Navier-Stokes code provided an opportunity to test some of the findings from the 1D numerical study in a higher dimension. A lot of effort was put in constructing the pre-processor, which creates the cartesian grid and determines the intersections of the grid lines with the immersed boundary. Additional parameters are defined to create a data structure that allows the IBMs to deal with immersed bodies effectively. The current pre-processor can handle most body shapes fully automatically. Thin, wedge-like shapes (e.g. airfoil trailing edges) still need a little bit of hand-coding. Three IBMs are successfully implemented in an existing 2D first-order finite volume code for the Navier-Stokes equations. These Immersed Boundary Methods are tested on three test cases: a backward-facing step flow, a circular cylinder flow in a channel and a multi-element airfoil flow. The results show that Immersed Boundary Methods are able to treat different boundaries in a satisfying manner. The qualitative aspects of the flows are captured well. Moreover, the grid generation is very straightforward and fast, even for the multi-element airfoil. The recommendations include suggestions on improving the pre-processor, on speeding up the steady solution method and on transforming the present code into an unsteady solver.

2000 Mathematics Subject Classification: 65N06, 65N30, 65N50, 76N05, 76N10, 76N17
Keywords and Phrases: immersed boundary methods; incompressible Navier-Stokes equations; finite-volume methods; finite-difference methods; Poiseuille flow; backward facing step flow; multi-element airfoil flow

The Immersed Boundary Method for the (2D) Incompressible Navier-Stokes Equations

January 2006

Reinout vander Meûlen



Delft University of Technology

Centre for Mathematics and Computer Science

The Immersed Boundary Method for the (2D) Incompressible Navier-Stokes Equations

Master of Science Thesis

Chair of Aerodynamics

Department of Aerospace Engineering

Delft University of Technology.

January 2006

by

Reinout vander Meûlen

This MSc. work has been approved by my supervisor:

prof. dr. ir. B. Koren

Composition of the exam committee:

assist. prof. dr. ir. M. Gerritsen	Stanford University, Stanford (CA), USA
dr. ir. M.I. Gerritsma	Delft University of Technology, Delft
dr. ir. S.J. Hulshoff	Delft University of Technology, Delft
prof. dr. ir. B. Koren	Centre for Mathematics and Computer Science, Amsterdam / Delft University of Technology, Delft
ir. J. Wackers	Centre for Mathematics and Computer Science, Amsterdam

Preface

This MSc. thesis concludes my graduation project and is intended to give an overview of my research activities on Immersed Boundary Methods over the past ten months. In order to present a comprehensible and straightforward report, not every aspect of the graduation work is discussed in detail, and some unsuccessful excursions from the main path are left out altogether. The result is, hopefully, a coherent and structured account of the research I performed in the final year of my studies in Aerodynamics.

The topic of Immersed Boundary Methods stirred my interest immediately when it was presented to me by my supervisor prof. Barry Koren. A relatively young field in Computational Fluid Dynamics, with great prospects and plenty of room for me to research and explore. On top of that, there is little activity on the topic in The Netherlands, which added to the sense of exploration and discovery, but this made access to specialized help and guidance harder to come by as well. Nevertheless, the interesting subject and the relative freedom I enjoyed in the research process have made this last year into a great experience.

I would like to take the opportunity to thank a few people for their involvement in my graduation project. First of all, my supervisor prof. Barry Koren, for the numerous discussions, his time and his interest in my work. His steering and guidance have made this project to what it is. More good advice and the basis for the 2D finite volume code came from Jeroen Wackers, his support is very much appreciated. A lot of thanks to Margot Gerritsen for the fantastic period I spent in Stanford and for making time to be a member of the exam committee. Gianluca Iaccarino from the Center for Turbulence Research in Stanford and his Immersed Boundary expertise helped me a great deal in getting the project started. More thanks to my office mate Jorick Naber and all the other colleagues in the MAS2 research group at CWI for the help and entertainment. And not to be forgotten, all the people that I spent less time with in this busy year: my family, my friends and girlfriend Lisa.

Abstract

Immersed Boundary Methods (IBMs) are a class of methods in Computational Fluid Dynamics where the grids do not conform to the shape of the body. Instead they employ cartesian meshes and alternative ways to incorporate the boundary conditions in the (discrete) governing equations. The simple grids and data structure are very well suited to handle complex geometries and moving boundaries.

The main objective of this project was to investigate Immersed Boundary Methods through literature study, brief analysis and numerical experiments, to gain experience and knowledge on the topic and to lay the foundations for practical use of these methods in future research. The approach that was taken to meet the objectives can be split into three parts: a literature study, a simple 1D channel-flow study and a 2D steady Navier-Stokes study.

The literature study presents the basic IBM techniques and a brief historical overview, followed by a discussion on some important properties of IBMs. Based on a structured classification of existing methods, a choice is made on the type of Immersed Boundary Methods to be explored in the 1D numerical study.

The 1D study makes use of the Poiseuille flow problem as a test case, since it has an analytical solution which allows us to calculate the absolute error made by the developed IBMs. The study of three new and two existing 1D methods and their derived variants reveals their accuracy on different grids and shows that this accuracy can be substantially affected by the position of an immersed boundary with respect to the neighboring grid points.

The construction of a steady 2D Navier-Stokes code provided an opportunity to test some of the findings from the 1D numerical study in a higher dimension. A lot of effort was put in constructing the pre-processor, which creates the cartesian grid and determines the intersections of the grid lines with the immersed boundary. Additional parameters are defined to create a data structure that allows the IBMs to deal with immersed bodies effectively. The current pre-processor can handle most body shapes fully automatically. Thin, wedge-like shapes (e.g. airfoil trailing edges) still need a little bit of hand-coding.

Three IBMs are successfully implemented in an existing 2D first-order finite volume code for the Navier-Stokes equations. These Immersed Boundary Methods are tested on three test cases: a backward-facing step flow, a circular cylinder flow in a channel and a multi-element airfoil flow. The results show that Immersed Boundary Methods are able to treat different boundaries in a satisfying manner. The qualitative aspects of the flows are captured well. Moreover, the grid generation is very straightforward and fast, even for the multi-element airfoil.

The recommendations include suggestions on improving the pre-processor, on speeding up the steady solution method and on transforming the present code into an unsteady solver.

Contents

Chapter 1

Introduction

Immersed Boundary Methods (IBMs) were invented in 1972 by Charles S. Peskin [21] in order to simulate the blood flow through the human heart. These methods differed substantially from the common CFD methods back then: instead of using body-fitted grids, he discretized the fluid flow equations on a cartesian mesh and added a forcing term to the governing equations to compensate for the presence of any immersed boundaries. This alternative class of CFD methods gradually began to intrude the world of computational aerodynamics where its main future lies in the simulation of flows around complex geometries or moving boundaries.

Moving boundaries, in particular Fluid-Structure Interaction, are a hot topic at the moment. Aero elasticity is a phenomenon that is hard to investigate, but engineering designs have pushed the envelope so far that adequate computational tools have become a necessity. Immersed Boundary Methods seem to be up to the challenge: IBMs codes have been used to simulate flows through complex-shaped coral colonies, around cars, electronic components and objects in free fall [10]. The highly reduced grid generating times make it a prime candidate for use in design processes, where fast turnaround times are essential.

However, Immersed Boundary Methods are still relatively unknown. Future research at CWI (Center for Mathematics and Computer Science, Amsterdam) and the Delft University of Technology may include the aerodynamics of sailing and the study of swimming fish, subjects that could benefit greatly from an IBMs approach. In this perspective the main objective for this graduation project was formulated: ”investigate the field of Immersed Boundary Methods through literature study and numerical experiments to gain experience and knowledge on the topic and to lay the foundations for use of these methods in future research.” For convenience and good comparison purposes, Fluid-Structure Interaction and unsteadiness are not yet considered in this thesis.

The process that was undertaken to accomplish this goal can be split into three main parts: a literature study, a 1D experimental numerical study and the construction of a 2D IBMs code for the steady Navier-Stokes equations, including grid generator. These three parts form the main structure of this report. The fourth and last part contains some conclusions and recommendations. The contents of each part are summed up briefly below, as a guideline for reading the report and as a way to keep everything in context.

Part I: Literature study. A definition of IBMs and their essential features is given, where the differences with body-conformal grid methods are emphasized. The strengths and weaknesses of the technique are discussed, followed by a general classification of IBMs.

Part II: The 1D methods. The Poiseuille flow is introduced as a 1D model problem. A number of new and existing methods are proposed to numerically approximate the solution to the Poiseuille problem. These approximations are then compared to the known analytical solution and the size and dependency of the error on the grid size is established. To conclude the 1D phase, a few methods are selected for implementation in the 2D code.

Part III: The 2D methods. The main features of the 2D finite volume solver that forms the basis for the IBMs codes are discussed. The cartesian grid generator, an essential element in the final code is next. Four IBMs implementations are analyzed in detail before they are tested on 2 benchmark problems: the backward facing step and a circular cylinder positioned asymmetrically in a channel. Finally, the simulation of the flow around a multi-element airfoil showcases the possibilities of the developed IBMs.

Part IV: Conclusions and recommendations. General conclusions on the performed research and recommendations for future developments.

Part I

Literature study

Chapter 2

Introduction to the literature study

This part of the thesis is the result of a literature study on Immersed Boundary Methods (IBMs). The literature study is based on a selected number of articles, presentations and conversations. Its purpose is not to summarize the articles studied but to give the reader a comprehensible introduction to the subject.

The next chapter will discuss in detail what immersed boundary methods are and why they are so different from standard Fitted Boundary Methods (FBMs). A comparison between both approaches is made in chapter 4, where it is shown that the intrinsic characteristics of IBMs can be a major advantage over FBMs for certain types of flow problems. In chapter 5, an overview is given of the different types of immersed boundary methods. Finally, in chapter 6, challenges in developing the next generation IBMs and the link to this graduation project are pointed out.

Chapter 3

The Immersed Boundary Method

3.1 Definition

The term Immersed Boundary Methods (also known as embedded boundary techniques) designates the class of boundary methods where the calculations are performed on a cartesian grid that does not conform to the shape of the body in the flow. The boundary conditions on the body surface are not imposed directly, instead an extra term, called the forcing function, is added to the governing equations or the discrete numerical scheme is altered near the boundary.

Two different classes can be distinguished within IBMs: the first deals with moving immersed boundaries and is very suitable for Fluid-Structure Interaction (FSI) problems, the second focuses on (complex) static embedded boundaries.

3.2 Historical note

The Immersed Boundary Method's founding father is Charles S. Peskin, who developed the technique in 1972 to study blood flow around heart valves (see [21] and [22]). His formulation consists of a cartesian mesh (Eulerian coordinate system - fixed in space) that is used to solve the flow equations and a curvilinear grid (Lagrangian coordinate system - moves with local flow velocity) which is attached to the elastic boundaries (heart walls). The information about the position of the boundary and the elastic force it exerts on the fluid is then transferred to the cartesian mesh in order to obtain a flow solution. To project the forcing on the grid, a smoothed delta function (*distribution function*) is used.

The method was (and remains) quite successful and in the mid-eighties the Immersed Boundary approach was extended to solid, in-deformable boundaries. At first, this was attempted by decreasing the deformability of the elastic fibres that model the immersed boundary. This inevitably results in a numerically stiff problem however. A number of ways to circumvent this problem were proposed, more on this topic can be found in chapter 5. Alternatively, the discrete forcing approach was formulated in the 1990's (see [4] and [18], [26]). This method is promising, especially for the simulation of high Reynolds number flows where continuous forcing cannot adequately represent a sharp boundary.

The IBMs field is still evolving and is mainly concentrated on flows with moving boundaries and flow simulation around complex geometries. Development of robust computational methods as alternatives to boundary-fitted CFD techniques remains an important objective, as well as more fundamental research on new IBMs formulations.

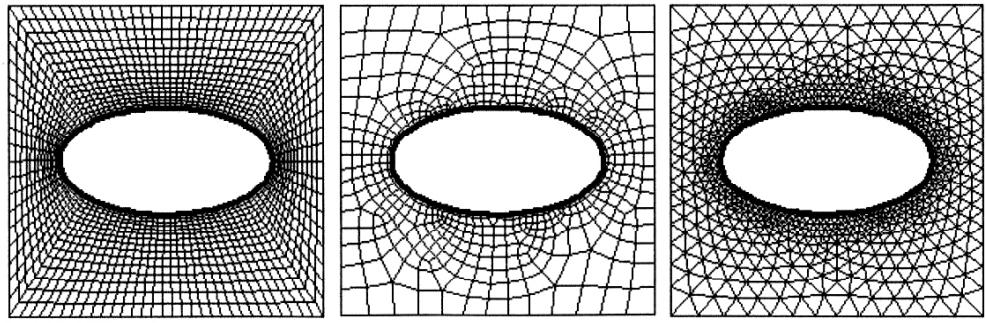


Figure 3.1: Structured and unstructured body-fitted grids (courtesy of G. Iaccarino [17])

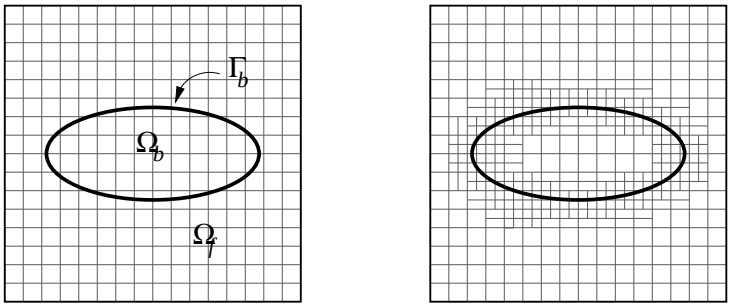


Figure 3.2: Cartesian grid without and with local refinements near the immersed boundary.

3.3 IBMs basics

This section introduces the standard layout of an immersed boundary method. For this purpose the flow around a two dimensional object is considered. The method is essentially the same in 3D. For simplicity, the body is assumed to be static.

3.3.1 The grid

Grid generation in fitted boundary methods consists of 2 parts: first a surface grid is created, which represents the geometry of the body in a discrete way. Then grid-generating algorithms build up a structured or unstructured mesh to fill the fluid domain, that is the space between the surface of the body and the boundaries of the computational domain. These grids are called body-conformal or body-fitted grids and a few examples can be found in figure 3.1.

In IBMs however, the grid is extremely simple. A rectangular (or cartesian) grid spans the whole computational domain, including the immersed body. Local grid refinements are possible in the vicinity of the boundary, this is done by subdividing cells. See figure 3.2.

The main advantage of fitted boundary methods is that imposing the boundary conditions is very simple. Since all the calculations are performed in grid points or cell faces, it is very convenient to have information about the solution at the boundary (e.g. flow velocity equals zero) exactly where you need it.

With IBMs, this is in general not possible because there is no relation between the body surface and the position of the grid points. The boundary conditions are imposed by including an extra term in the governing equations (the forcing function) or changing the numerical stencil near the boundary.

3.3.2 The forcing function

The implementation of the boundary conditions through the use of a forcing function is the core of an immersed boundary method. This can be done in many ways, see chapter 5. The general concept is best explained by the use of an abstract example. Mittal and Iaccarino have given a very comprehensive explanation in their paper on IBMs [17]. The same approach is followed here.

Consider a set of conservation laws (a system of partial differential equations) that govern the flow around the object in figure 3.2:

$$\Lambda(\mathbf{U}) = \mathbf{0} \qquad \text{in } \Omega_f, \tag{3.1}$$

$$\text{with} \qquad \mathbf{U} = \mathbf{U_\Gamma} \qquad \text{on } \Gamma_b. \tag{3.2}$$

The body has a volume Ω_b and a boundary Γ_b. The volume Ω_f is occupied by the fluid. The treatment of the boundary "at infinity", the outer boundary of the domain, is not important when explaining basic IBMs procedures and is therefore ignored at this stage. The discretized flow equations will be solved on a cartesian grid which covers the whole computational domain $\Omega = \Omega_b + \Omega_f$. $\mathbf{U}$ is the vector representing the unknowns and Λ is the operator denoting the governing equations as mentioned above. Note that the boundary condition $\mathbf{U_\Gamma}$ is not available for every component of $\mathbf{U}$ on the immersed boundary.

In FBMs, one would formulate a discretization of eq. (3.1) on a body-conformal mesh and enforce the boundary condition (3.2) directly. IBMs however require modification of (3.1) to enable imposing the boundary condition.

Let us assume for the moment that a forcing function $\mathbf{f_b}$ or $\mathbf{f'_b}$ exists such that the boundary condition can be imposed by including this forcing function in the governing equations as a source term. This can be done in two essentially different ways: the continuous forcing vs. discrete forcing approach.

Continuous forcing

In this approach, a forcing function $\mathbf{f_b}$ will be added to the right hand side of equation (3.1) to yield $\Lambda(\mathbf{U}) = \mathbf{f_b}$. This equation is then discretized on the cartesian grid with an appropriate numerical scheme and transforms into a system of discrete equations:

$$[L]\{\mathbf{U}\} = \{\mathbf{f_b}\}. \tag{3.3}$$

Equation (3.3) can now be solved on the entire domain $\Omega = \Omega_b + \Omega_f$. Because the forcing was introduced before the equations were discretized, this method is known as the "continuous forcing approach". The fact that this IBMs formulation is independent of the spatial discretization scheme is one of its most important characteristics.

Discrete forcing

Instead of including a forcing function in the continuous equations, it is also possible to discretize eq. (3.1) on the cartesian grid without taking into account the presence of the body: $[L]\{\mathbf{U}\} = \mathbf{0}$. In the cells near the immersed boundary the system $[L']\{\mathbf{U}\} = \{\mathbf{r}\}$ incorporates the boundary conditions. The vector $\{\mathbf{r}\}$ represents known terms associated with the boundary conditions on the immersed surface. To find a solution, $[L]\{\mathbf{U}\} = \mathbf{0}$ needs to be solved on Ω_f, and

$$[L]\{\mathbf{U}\} = \{\mathbf{f'_b}\} \tag{3.4}$$

$$\text{with} \qquad \{\mathbf{f'_b}\} = \{\mathbf{r}\} + [L]\{\mathbf{U}\} - [L']\{\mathbf{U}\}$$

on Ω_b. This expression may look very abstract, but the concept is explained in more detail in chapter 5.

The forcing was now introduced after discretizing the equations, therefore this branch of IBMs is called the ”discrete forcing approach”. This method depends substantially on the discretization scheme, allowing direct control over numerical accuracy, stability and discrete conservation properties of the solver.

The two approaches discussed above illustrate a fundamental duality in the IBMs field. A number of different methods exist in each category, those will be discussed in chapter 5.

Chapter 4

Relevance of IBMs

In the previous chapter the basic immersed boundary procedures were explained to equip the reader with a sense of what to expect of these methods. It became clear that imposing the boundary conditions is not straightforward when using cartesian grids and that the effect of the boundary treatment on the accuracy and conservation properties of the numerical scheme is not obvious. So, all things considered, why would someone put in the effort of developing these new methods? What makes IBMs so special that they are worth investigating?

4.1 The grid

One of the main features of IBMs is the cartesian grid. In general, the quality of a grid is high when the total number of grid points is minimal while the local resolution is still acceptable. Such a grid will give good accuracy at the fastest time-to-solution. A typical high-quality cartesian grid and the intersections of the grid with the immersed boundary are reasonably easy to generate for virtually any geometry, even for complex, moving bodies. CFD solvers that use body-fitted grids often run into trouble when dealing with complex or moving geometries, even when the grids are unstructured.

4.1.1 Complex geometries

Generating a high-quality structured body conformal grid for a complex geometry can be a major task, since grid-generation algorithms often cannot deal with very sharp corners, holes in the geometry or irregular shapes. Furthermore, obtaining desired grid quality requires a lot of human interaction, and the total grid-generation process can amount up to 25% of the total computation time. Unstructured grids are better suited to handle complex shapes, but they need a substantial amount of extra CPU time and memory for construction and storage, as compared to structured grids.

Cartesian grids however can be constructed really easily and quickly with an automated grid generator, without any human interaction. As opposed to body-fitted meshes, cartesian grids are not affected significantly by higher complexity in body shapes. Novel approaches to local grid refinement [12] can even reduce both the computational overhead and the storage cost involved in constructing the grid such that it becomes much more attractive than an unstructured mesh.

4.1.2 Moving boundaries

In the case of moving boundaries, such as deforming boundaries occurring in Fluid-Structure Interaction or tumbling bodies in free fall, body conformal grids have to be regenerated at every time step. In addition to this, a procedure is required to project the old solution onto the new grid. These two steps do not only add to the computational cost, they can also deteriorate the simplicity, accuracy and stability of the solver.

The Immersed Boundary Methods have a clear advantage here: by using a stationary, nondeforming cartesian grid, the application to flow problems with moving boundaries becomes much easier and there is no need to reconstruct the grid at every time step. This means that IBMs can be faster and more robust than CFD methods based on body-conformal meshes. Especially in iterative engineering design procedures where multiple computations need to be done, limiting the time-to-solution and the time spent on manual grid modifications is important.

4.2 The number of operations per grid point

Another advantage of cartesian grids with respect to body-conformal grids is that the per-grid-point operation count can be significantly lower. This is true for both structured and unstructured grids, albeit for a different reason.

4.2.1 Cartesian versus structured grids

If a structured curvilinear body-fitted grid is used, there are two ways to proceed when calculating fluxes or flow variables over cells. One is to transform the physical domain to a computational domain via a coordinate transformation, and then solve the (transformed!) equations and re-transform the solution to the physical domain. The other method does not make use of a computational domain, but this implies evaluating fluxes in X, Y and Z direction over cell faces that are not aligned along the principal axes, requiring local rotations.

Cartesian grids do not suffer from these extra operations per grid point. The grid is aligned with the global principal axes and therefore grid transformations or local rotations are not necessary.

4.2.2 Cartesian versus unstructured grids

IBMs also have an advantage over unstructured grids: it is possible to increase the computational speed by applying powerful line-iterative techniques or geometric multigrid methods. These techniques can in principle be applied to unstructured grids as well, though less straightforward.

4.3 A perfect method?

There are of course a few disadvantages to IBMs and it is important to be aware of them. It was already mentioned before that the treatment of the boundary conditions is not evident, but the main problem is that the grid size (i.e. the total number of grid points or cells in the grid) increases faster with increasing Reynolds number for uniform cartesian grids than for body-fitted grids. This is due to the fact that the alignment between the grid lines and the body surface in body-conformal grids results in better control of the grid resolution in the boundary layer, while this is not the case for cartesian grids. It is shown in [17] that the grid-size ratio

$$\textit{grid-size ratio} = \frac{\textit{size of cartesian grid}}{\textit{size of body-fitted grid}}$$

$$\begin{aligned} \text{scales with} \quad & (Re)^{1.0} \quad \text{for 2D bodies} \\ \text{and with} \quad & (Re)^{1.5} \quad \text{for 3D bodies.} \end{aligned}$$

Consequently, as the Reynolds number increases, a cartesian grid will grow faster than its body-conformal counterpart. The problem is not as bad as it seems though, since a substantial amount of grid points can be inside the body, where the fluid-flow equations need not be solved (depending on the IBMs used). The fraction of grid points that lie within the body is proportional to the ratio of the volume of the body to the volume of the bounding box, and therefore depends on the shape and orientation of the body.

Chapter 5

Classification

In general (as discussed in chapter 3), IBMs are split up in continuous forcing and discrete forcing approaches. Within those categories, there are still important differences between methods. This chapter attempts to present the essence of each method and tries to point out the primary advantages and disadvantages associated with this specific technique. The classification follows the one by Mittal and Iaccarino [17].

5.1 Continuous forcing approach

As will be shown further in this section, elastic and rigid boundaries require different treatments in an immersed boundary formulation. Therefore, those subjects will be dealt with separately.

5.1.1 Immersed bodies with elastic boundaries

This is the class of flow problems Peskin had in mind when he developed the first IBMs [21]. His method solves the Navier-Stokes equations for the fluid on a stationary cartesian grid, while the immersed boundary is represented by a set of elastic fibres whose location is tracked in a Lagrangian fashion (fibres move with local flow velocity). The coordinate vector $\mathbf{X_k}$ of the k^{th} Lagrangian point follows from the fact that the time derivative of the fiber location has to equal the fluid velocity at that point:

$$\frac{\partial \mathbf{X_k}}{\partial t} = \mathbf{u}(\mathbf{X_k}, t). \tag{5.1}$$

The stress ($\mathbf{F}$) and deformation of the fibres are related by Hooke's Law or a similar relation. Finally, the effect of the immersed boundary on the flow has to be incorporated through the inclusion of a forcing function in the fluid momentum equations. For this purpose, a local forcing term $\mathbf{f_m}(\mathbf{x}, t)$ transfers the fiber stress to the fluid-flow grid:

$$\mathbf{f_m}(\mathbf{x}, t) = \sum_k \mathbf{F_k}(t)\delta(\mid \mathbf{x} - \mathbf{X_k} \mid), \tag{5.2}$$

δ represents here the Dirac delta function. However, since in general the immersed boundary does not coincide with the cartesian grid points, the Dirac delta function is not able to transfer the forcing to the cartesian grid. Therefore, the sharp δ function is replaced by a smoother distribution d which spreads the forcing over a band of cells on the cartesian grid around the immersed boundary. Figure 5.1 shows schematically how this can be achieved.

The forcing at any cartesian grid point $x_{i,j}$ as a result of the elastic forces in the fibres is then:

$$\mathbf{f_m}(\mathbf{x_{i,j}}, t) = \sum_k \mathbf{F_k}(t)d(\mid \mathbf{x_{i,j}} - \mathbf{X_k} \mid). \tag{5.3}$$

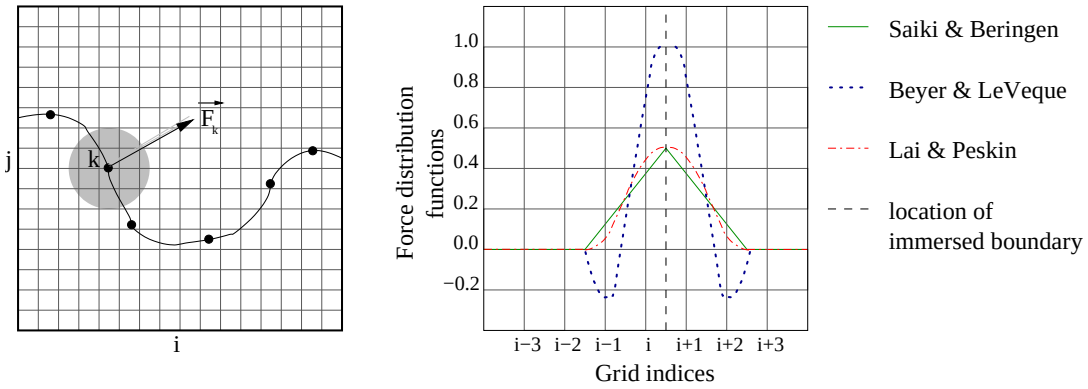


Figure 5.1: Fiber stress in Lagrangian point k is spread over surrounding grid nodes (shaded region) by a distribution function (examples of various studies, see [24], [4] and [16]).

The same smooth distribution function d is also used to compute the fiber velocity in equation (5.1) by projection of fluid velocity data on the fiber location.

Obviously, the smooth distribution function plays an important role, because this function is responsible for the transfer of elastic forces to the fluid and for the feedback on the new position of the immersed boundary. Various versions of the d function have been proposed, see e.g. [24], [4] and [21].

The method for elastic immersed boundaries that is described in this section has been applied successfully to many practical problems in biology and multiphase flows.

5.1.2 Immersed bodies with rigid boundaries

When the immersed boundary is rigid however, the limitations of Peskin's technique become visible. The main problem is that the constitutive laws (e.g. Hooke's Law) are in general not well-posed in the rigid limit, meaning that small deformations of a very stiff boundary cause large stresses. One way to deal with this problem is to assume the body to be elastic, but extremely stiff. Another technique assumes the structure / boundary to be attached to an equilibrium position by a spring which results in a restoring force, see [4] and [16]:

$$\mathbf{F_k}(t) = -\kappa\Big(\mathbf{X_k} - \mathbf{X_k^e}(t)\Big), \tag{5.4}$$

where κ represents a positive spring constant and $\mathbf{X_k^e}$ the equilibrium position of the k^{th} Lagrangian point. Evidently, approximating the boundary as rigid in an accurate way requires large spring constants. Lower values of κ can allow the boundary to deviate excessively from its equilibrium position, which is of course not realistic for a rigid body.

Both approaches discussed so far in this section will give rise to stiff systems of equations and the related numerical problems. When using explicit time integration methods, this in turn limits the maximum allowable time step in order to maintain a stable system, adding substantially to the computational overhead.

A generalized version of the previous method was formulated by Goldstein et al. [11]. However, the same stability problems arise due to the very large "stiffness" constants that are needed in this approach to model the rigid boundary accurately. Especially highly unsteady flows and high Reynolds number flows play havoc with the numerical stability of the solver.

A technique related to Goldstein's assumes the entire flow to occur in a porous medium [3], [15], a physical process governed by the Navier-Stokes/Brinkman equations. An extra (forcing) term in the Navier-Stokes

equations contains the permeability (K) of the medium, i.e. zero for a solid and infinite for a fluid, and thus forces the velocity field to zero within the solid. Then again, stability constraints are severe due to large variations in K and errors in imposing the right velocity on the boundary are inevitable due to smoothing of the variation of K at the fluid-solid interface.

5.1.3 Continuous forcing: summary

The continuous forcing approach is an attractive IBMs formulation for problems with elastic boundaries. The model itself remains close to the physics behind the phenomena and is relatively easy to set up for realistic flow problems. Successful simulations of biological and multiphase flows have been accomplished.

When flow problems involve very stiff or rigid bodies, continuous forcing is likely to cause trouble since most methods give rise to ”stiff” numerical systems. Satisfactory results have only been attained for low Reynolds number flows with moderate unsteadiness.

The main problem associated with the continuous forcing approach is that the smoothing of the forcing function prohibits the sharp representation of the immersed boundary. This is often not acceptable at high Reynolds numbers. Furthermore, some of the methods described above require the solution of the fluid flow equations inside the body, where the solution is often not needed or unphysical. For high Reynolds numbers however, a substantial amount of grid points can be located inside the body (see chapter 4), and thus be responsible for unnecessary extra computation time.

5.2 Discrete forcing approach

Instead of focussing on elastic or rigid boundaries, methods that fall under the discrete forcing approach are categorized into methods that enforce the boundary conditions directly on the immersed boundary and methods that impose the boundary conditions indirectly.

5.2.1 Indirect boundary condition imposition

Indirect imposition of the boundary conditions (e.g. $\mathbf{u} = \mathbf{0}$) means simply that this boundary condition is not used directly in the numerical scheme. Instead a forcing term is added to the discrete system of governing equations in the cells near the immersed boundary. The forcing term is derived from the boundary conditions, as opposed to continuous forcing methods where it is determined using a ”mechanical” relation (force versus displacement or permeability). In general, the derivation of the forcing term is done by estimating the velocity field and correcting it at the boundary to fit the boundary conditions. Only in rare cases the forcing term can be determined analytically: see [4] and chapter 13.

Mohd-Yossuf [18] and Verzicco et al. [26] tackle the problem by calculating $a priori$ a prediction for the velocity field from which the forcing is determined. This is accomplished without user-specified parameters or the like in the forcing terms. That means that the stability of the numerical scheme for rigid bodies will not depend on any stiffness parameter (as is the case in the continuous forcing methods), and that is of course good news. But the repeated use of the smoothed distribution function to extend the forcing to the grid may cause trouble at high Re.

5.2.2 Direct boundary condition imposition

The problem with all the methods that were reviewed in this report so far is that their performance suffers at higher Reynolds numbers. The spreading of the forcing by the smoothed distribution function undermines the local accuracy of the solution in the boundary layers. To retain a sharp boundary, the computational stencil near the immersed boundary can be modified such that the boundary condition can be implemented directly. The next two methods do precisely this.

Ghost-cell finite difference approach

Especially at higher Reynolds numbers, solving the fluid equations inside the solid body is not only unphysical, but also CPU-time consuming. To avoid this, ghost cells are defined just inside the immersed boundary, such that every ghost cell has at least one neighbor in the fluid (see figure 5.2). If the (artificial) value for the flow parameters were known in these cells, the computation in the fluid domain could stop right there without having to solve inside the solid. However, there is no information available in the ghost points themselves, but very close by there is: the boundary condition on the immersed boundary in the case of a rigid no-slip boundary is given as $\mathbf{u} = \mathbf{0}$. The value of the state variables in the ghost cell can then be extrapolated from the boundary and from fluid cells close by.

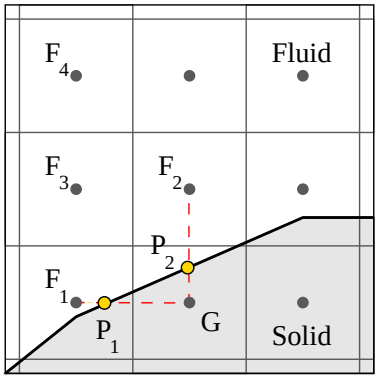


Figure 5.2: Ghost cell method: F1, F2, F3 and F4 are fluid nodes, G is a ghost cell node, B1, B2 and P1, P2 are points on the boundary that can be used in different extrapolation stencils.

There are of course quite a few options available for constructing the extrapolation scheme. A simple one is the bilinear scheme (trilinear in 3D):

$$\phi = C_1 xy + C_2 x + C_3 y + C_4, \tag{5.5}$$

where ϕ is a generic flow variable and C_1 through C_4 coefficients that can be determined by evaluating ϕ at fluid nodes F_1 and F_2 and at two points on the immersed boundary (P_1 and P_2). Depending on the type of flow that is expected (laminar, turbulent) and the Reynolds number, it may be attractive to use higher-order extrapolation stencils and use other/more fluid or boundary points.

The generic flow variable can now easily be evaluated in the ghost point, and solving (5.5) for the ghost cells simultaneously with the flow equations on the fluid domain solves the system. This method has been quite successful for simulating viscous flows with Reynolds numbers of up to $O(10^5)$.

Cut-cell finite volume approach

The primary reason for adopting a finite volume approach is often that such methods naturally assure the conservation of mass and momentum, something that is missing from all the techniques discussed above. A finite volume method requires that the cells that are cut by the immersed boundary will have to be reshaped such that the immersed boundary will coincide with a cell face. This is done by cutting the ”solid” part of the cell away. If the center of the original cell lies in the fluid, the reshaped cell becomes an independent new control volume, otherwise the cut cell may be merged with a neighbor. This is done to prevent the creation of new cells that are significantly smaller than the surrounding ones, which could lead to errors in the solution. In figure 5.3 such a newly formed cell is shown.

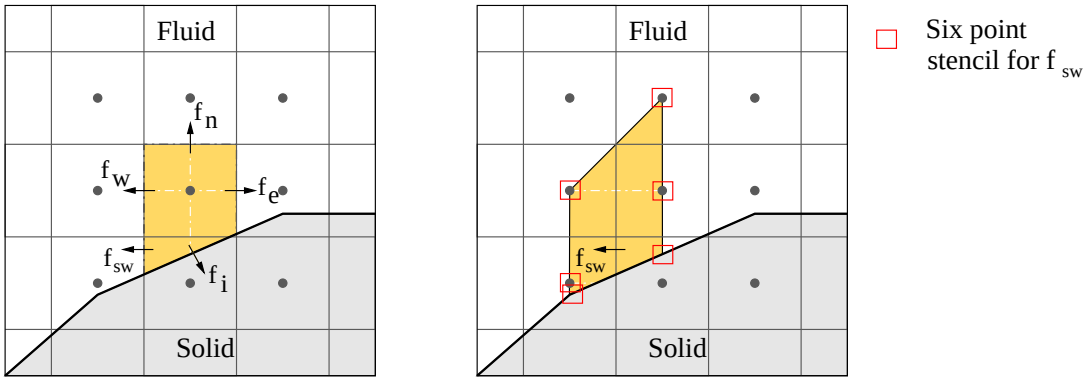


Figure 5.3: Cut-cell method: re-shaped cell next to immersed boundary with associated fluxes (left); 6-point interpolation stencil for determining the flux over the southwest cell face (right).

The next step is to formulate expressions that approximate the flux integrals of mass, convection and diffusion as well as pressure gradients on the faces of each cell. One way to do this is to formulate an interpolation or extrapolation stencil that expresses a flow variable ϕ in terms of a polynomial and to determine the coefficients by evaluating ϕ in a number of fluid nodes and points on the immersed boundary. The fluxes $\mathbf{f}$ can then be approximated based on the function for ϕ.

As an example, take the flux on the southwest face in figure 5.3. This flux can be based on a 6-point interpolation stencil that is linear in x and quadratic in y:

$$\phi = C_1 xy^2 + C_2 x^2 + C_3 xy + C_4 x + C_5 y + C_6. \tag{5.6}$$

Coefficients C_1 through C_6 can be determined by evaluating ϕ at the six points shown in the figure. The other fluxes are treated in a similar way.

The stencil (5.6) in this example guarantees local second-order accuracy and exact conservation of mass and momentum, irrespective of the grid resolution. Note also that the cut-cell method obviates the need for computation of any information inside the solid body, and retains a sharp boundary (no force spreading). The cut-cell method has been used successfully in the simulation of complex 2D flows, for instance flow-induced vibrations, flapping foils and multiple objects in free fall through a fluid. However, extension to 3D is not straightforward since the cut-cell approach would generate complex-shaped cells, which makes it hard to discretize the governing equations. Adopting so-called cell-trimming procedures to generate a body conformal grid from a cartesian grid [29] could circumvent this problem.

One other drawback that is inherent to all discrete forcing approaches is that a pressure condition on the boundary is necessary, something that was not needed in continuous forcing methods. Furthermore, moving boundaries are harder to deal with than in continuous forcing IBMs, as is explained in the following section.

5.2.3 Flows with moving boundaries

While including a moving boundary is relatively easy for continuous forcing methods (see e.g. Peskin's method), discrete forcing approaches (with direct boundary condition imposition) have to deal with a special problem when the immersed boundary is moving through the domain: freshly-cleared cells. These are cells that were inside the solid at time t but become fluid cells at time $t + \delta t$ (figure 5.4). These cells do not have a valid time-history, so evaluating time derivatives requires a trick. As seen before, interpolating the required flow variable from neighboring points and the boundary can provide a solution. Another possibility is to merge the freshly-cleared cells with adjacent fluid cells for the first time step after emerging from

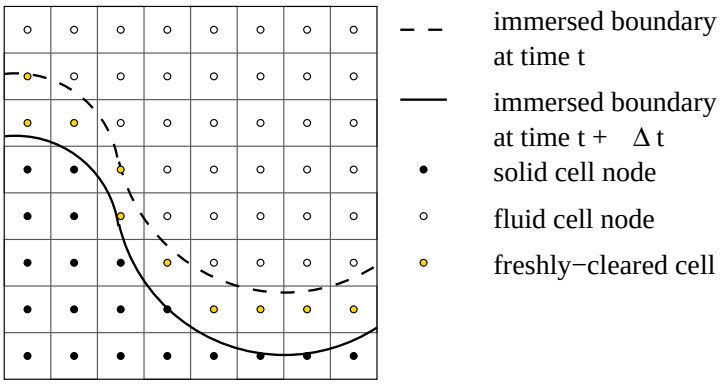


Figure 5.4: When the immersed boundary retreats, some cells that were inside the solid are cleared and belong to the fluid at the next time step.

the body.

The issue of freshly-cleared cells is not a problem in methods that use distribution functions to transfer the forcing to the cartesian grid (i.e. all continuous forcing methods), since the distribution function provides a smooth transition from solid to fluid.

5.2.4 Discrete forcing: summary

The discrete forcing approach is very well suited for flows around rigid bodies and the cut-cell and ghost-cell methods in particular can handle higher Reynolds numbers. Key elements are the absence of stiffness or user defined parameters that can impact the stability of the method, the ability to represent sharp immersed boundaries by imposing the boundary conditions directly on the numerical scheme and the fact that computing flow variables inside a rigid body becomes unnecessary.

The disadvantages include the need for a pressure boundary condition on the immersed boundary, the slightly more complicated inclusion of boundary movement into the procedure and the less straightforward implementation of the forcing function, which is now intimately connected to the discretization of the governing equations.

Chapter 6

Developments in IBMs

In the previous chapters the concept of Immersed Boundary Methods is explained and a number of techniques are discussed and classified. It is shown that IBMs are a serious alternative to fitted boundary methods for flows around complex shaped bodies and for flows with moving boundaries. The main features of Immersed Boundary Methods for these flow types are the simple grids that can be generated quickly and the shorter time-to-solution.

The present developments in IBMs research are focused on reducing the time-to-solution, automating the CFD analysis process, simulating higher Reynolds number flows and modeling the physics as accurately as possible. Work is being done on local grid refinement, on including more physics into interpolation stencils and on addressing multi-physics problems (e.g. multiphase flows, combustion, etc.).

The graduation work will start on a one-dimensional model problem (see [4]) where the goal is to experiment with various IBMs to get a feel for the existing techniques and to try some ideas of my own. This exploration phase will form the basis for the extension to two dimensions.

Part II

The 1D methods

Chapter 7

Introduction to the 1D methods

The next phase of the graduation work on Immersed Boundary methods focuses on a one-dimensional model problem, more specifically a Poiseuille flow. A similar model for the 1D heat equation was studied by Beyer and LeVeque [4]. The main purpose of this study was to get a better understanding of IBMs in general, to identify the primary challenges in building a successful IBMs solver and to determine which method to use for the 2D code. Since our interests are high Reynolds number flows around rigid bodies, mainly the discrete forcing approach from chapter 5 will be considered from now on.

The idea to study the Poiseuille flow was suggested by Iaccarino, who has investigated the same problem in his PhD thesis [13]. In the first chapter, the flow under consideration is described in detail and the governing equation is derived from the Navier-Stokes equations. An important property of the governing equation is that it can be solved analytically, allowing for detailed assessment of the error made with each method. Chapters 9 through 13 discuss the various Immersed Boundary Methods that were used to approximate the solution of the Poiseuille flow. An extensive error analysis of these methods is the subject of chapter 14. IBMs differ from body-conformal grid methods in the sense that the relative position of the immersed boundary in a grid cell has a substantial effect on the error. This and other interesting conclusions can be found at the end of this part (chapters 14 and 15).

Chapter 8

Problem description

8.1 The Poiseuille flow

The subject of the one-dimensional study is the well-known Poiseuille flow. Its geometry is that of a channel, formed by two infinite plates or walls, running parallel to each other. The walls are fixed in space, and the distance between them (the width of the channel) is denoted by H. The coordinate system is chosen such that the lower wall coincides with the XZ plane, with the Y axis pointing in the direction of the upper wall. The flow is driven by a pressure gradient $\frac{\partial p}{\partial x}$ and is assumed to be laminar, steady, incompressible and horizontal (i.e. body forces are ignored). Figure 8.1 represents this setup in 2D.

8.2 The governing equation

The Poiseuille flow can be described by the Navier-Stokes equations. However, since the flow has some special properties, these equations can be simplified to yield the governing equation. The derivation starts with the 2D incompressible Navier-Stokes equations as given in [2]:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{8.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right), \tag{8.2}$$

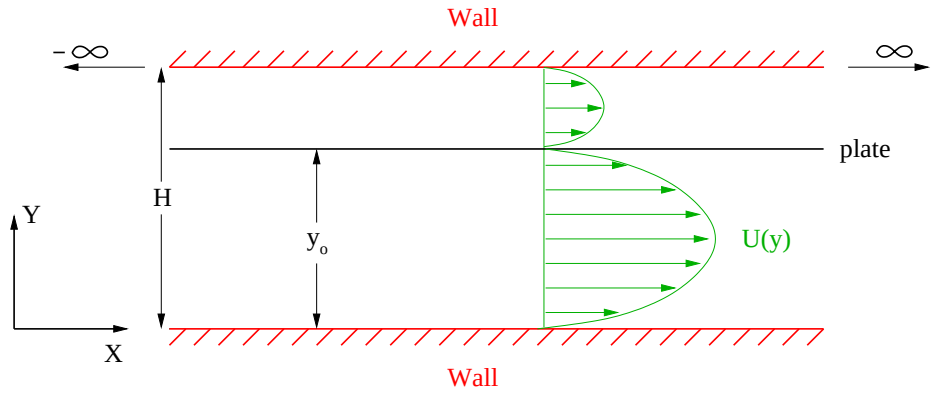


Figure 8.1: Schematic representation of the Poiseuille flow: a laminar, fully-developed viscous flow in an infinite channel, where an infinite plate acts as the immersed boundary.

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right). \tag{8.3}$$

The energy equation is not needed to close the system of equations, and we are left with three state variables: u, v and p. The assumption that the flow is steady means that all the time derivatives can be eliminated from the equations. Furthermore, since the plate and walls of the channel are infinitely long, the streamlines are considered to be parallel ($v = 0$). The continuity equation (eq. (8.1)) becomes then:

$$\frac{\partial u}{\partial x} = 0 \qquad \text{or:} \qquad u = u(y), \tag{8.4}$$

meaning that the velocity u is a function of y only. This is the reason why this seemingly two-dimensional flow problem was chosen for the 1D study. The momentum equations simplify to:

$$0 = -\frac{\partial p}{\partial x} + \mu\frac{\partial^2 u}{\partial y^2}, \tag{8.5}$$

$$0 = -\frac{\partial p}{\partial y}. \tag{8.6}$$

. While equation (8.6) confirms that the pressure can only vary in flow direction, the real result of this derivation is found in the reduced x momentum equation (eq (8.5)). This governing equation determines the velocity profile of the Poiseuille flow. This PDE can be solved analytically and the resulting velocity distribution is parabolic in y:

$$u(y) = \frac{1}{2\mu}\Big(\frac{dp}{dx}\Big)\big(y^2 - Hy\big). \tag{8.7}$$

To obtain this result, no-slip $(u = 0)$ boundary conditions at the upper and lower wall are taken into account.

8.3 The immersed boundary

Now that the governing equation (8.5) and the analytical solution (8.7) of the Poiseuille flow are known, it is time to attempt to approximate the velocity profile using IBMs. For this purpose, an immersed boundary is introduced at an arbitrary distance y_o from the lower wall, see figure 8.1. This boundary needs to be an infinite flat plate, parallel to the channel walls, if it is still to be subject to the governing equation.

The analytical solution to this flow consists of two standard Poiseuille flows on top of each other, both driven by the same $\frac{dp}{dx}$:

$$u(y) = \frac{1}{2\mu}\Big(\frac{dp}{dx}\Big)\big(y^2 - y_o y\big), \qquad 0 < y < y_o, \tag{8.8}$$

$$u(y) = \frac{1}{2\mu}\Big(\frac{dp}{dx}\Big)\big(y^2 - Hy - y_o y + y_o H\big), \qquad y_o < y < H. \tag{8.9}$$

This solution is exact and it will be used to assess the errors made by the numerical IBMs. An equidistant grid is used, with a grid size irrespective of the position of the immersed boundary. The number of grid points is denoted by N, consequently the distance between grid points becomes $h = H/N$. The first and last grid point coincide with the upper and lower wall, which allows straightforward implementation of the no-slip boundary conditions at the wall. However, the immersed boundary will generally not line up with a grid point, as in figure 8.2.

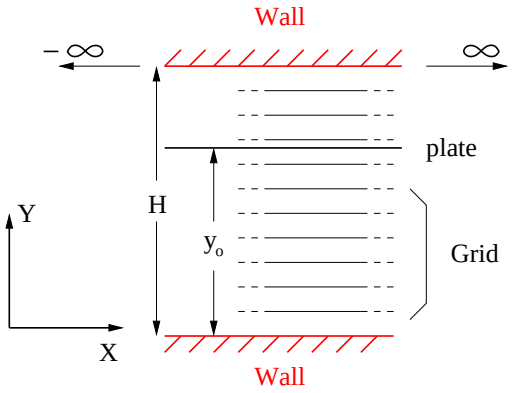


Figure 8.2: Equidistant, 1D grid in Poiseuille channel. Note that the boundary position is arbitrary and independent of the grid layout.

8.4 Numerical methods

The next step is to discretize the governing equation on the 1D grid in ”immersed boundary” fashion. This is mainly done using finite differences, except for the Cut cell method which is a finite volume method. Especially the treatment of the no-slip boundary condition on the immersed boundary is crucial for the success of a method, and both direct and indirect boundary condition imposition methods are examined. Besides implementing existing methods (e.g. ghost cell, cut cell, etc.), my adviser prof. Koren and I came up with new approaches and numerical schemes for this model problem. All these IBMs are discussed in chapters 9 through 13.

Chapter 9

Method 1: Explicit boundary condition method

9.1 Introduction

The Explicit boundary condition method can be classified as a continuous forcing method (see chapter 5), because the boundary condition is integrated in the governing equation such that the discretization stencil is the same in every part of the domain. The general idea behind this method is that the physical domain can be split up in 3 important parts: *the fluid domain* Ω_f, *the walls of the channel* Γ_w and *the immersed boundary* Γ_b.

Different equations hold on each part of the domain:

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\mu}\frac{dp}{dx} \quad \text{valid on} \quad \Omega_f, \tag{9.1}$$

$$u = u_\Gamma = 0 \quad \text{valid on} \quad \Gamma_w \text{ and } \Gamma_b. \tag{9.2}$$

Equation (9.1) is a rewritten version of the governing equation and (9.2) is the no-slip boundary condition.

When attempting to simulate this flow, it is important to choose the right equation depending on the position in the channel. This is not relevant for the boundary conditions at the wall because they are easily incorporated in the numerical scheme due to the definition of the grid. The immersed boundary is tackled by defining a function Δ which will allow the solver to switch properly between equations (9.1) and (9.2):

$$\Delta = \Delta(y - y_o) = \begin{cases} 1, & y = y_o; \\ 0, & y \neq y_o. \end{cases}$$

To avoid adding up terms with conflicting dimensions, equation (9.2) is divided by the square of the cell size h:

$$\frac{u}{h^2} = \frac{u_\Gamma}{h^2} = 0 \tag{9.3}$$

Equation (9.1) is now multiplied by $\big(1 - \Delta(y - y_o)\big)$ and equation (9.3) by $\Delta(y - y_o)$ to yield:

$$\big(1 - \Delta(y - y_o)\big)\big(\frac{\partial^2 u}{\partial y^2} - \frac{1}{\mu}\frac{dp}{dx}\big) + \Delta(y - y_o)\frac{u}{h^2} = \Delta(y - y_o)\frac{u_\Gamma}{h^2} = 0. \tag{9.4}$$

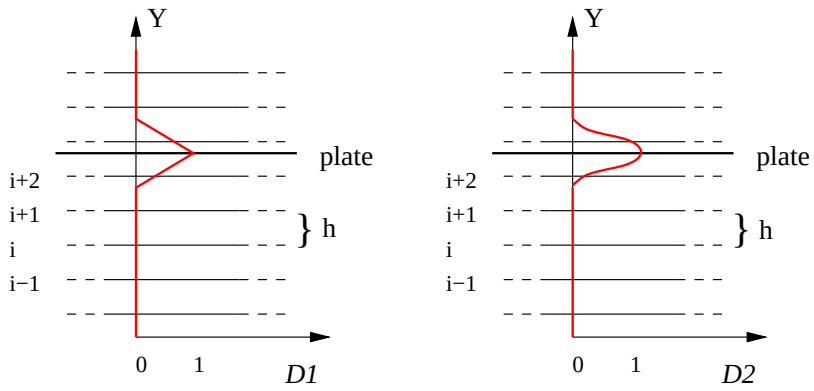


Figure 9.1: Sketch of distribution functions $D1$ and $D2$.

9.2 Distribution functions

This last equation (9.4) needs to be discretized on the equidistant grid. This however gives rise to a problem: because the Δ function is only non-zero at the exact location of the immersed boundary and because this boundary does not coincide with a grid point, its influence on the flow will not be taken into account. Therefore the Δ function must be replaced by a distribution D which will spread the presence of the plate to the neighboring grid points. Numerous distributions are possible, but for the sake of simplicity two basic distributions $D1$ (hat) and $D2$ (cosine) were chosen.

$$D1 = \begin{cases} 1 - \frac{|y-y_o|}{h} & |y-y_o| < h, \\ 0 & |y-y_o| > h; \end{cases} \qquad D2 = \begin{cases} \frac{1}{2}(\cos[\frac{(|y-y_o|)\pi}{h}] + 1) & |y-y_o| < h, \\ 0 & |y-y_o| > h. \end{cases}$$

Note that the width of the distribution functions is $2h$ and that they are both equal to 1 on the plate. This implies that there is always one grid point on each side of the plate which feels the influence of the no-slip condition, except for the rare case where plate and grid point coincide. When this happens, the boundary condition on the plate will be enforced at the grid point and the two neighbors will not be affected, or only through the normal numerical scheme.

9.3 The numerical scheme

Finally, equation (9.4) is ready to be discretized. The second derivative $\frac{\partial^2 u}{\partial y^2}$ is replaced by a second-order accurate central discretization. The $\frac{dp}{dx}$ term is a constant and D represents a distribution function ($D1$ or $D2$):

$$u_{i+1}\left[\frac{1-D(y-y_o)}{h^2}\right] + u_i\left[\frac{3D(y-y_o)-2}{h^2}\right] + u_{i-1}\left[\frac{1-D(y-y_o)}{h^2}\right] = \big(1-D(y-y_o)\big)\frac{1}{\mu}\frac{dp}{dx}. \quad (9.5)$$

To satisfy the boundary conditions at the walls, equation (9.5) is replaced by $u_1 = 0$ and $u_{N+1} = 0$ for the first and last grid point respectively. Equation (9.5) and the wall boundary conditions are then used to formulate the system in matrix - vector notation. The matrix is tri-diagonal which allows the use of the Thomas algorithm. No own software was written for the inversion of this matrix, it was just inverted in Matlab. The result can be seen in figure 9.2 for $N = 20$ and $y_o = 0.77$ (parameters are arbitrarily chosen).

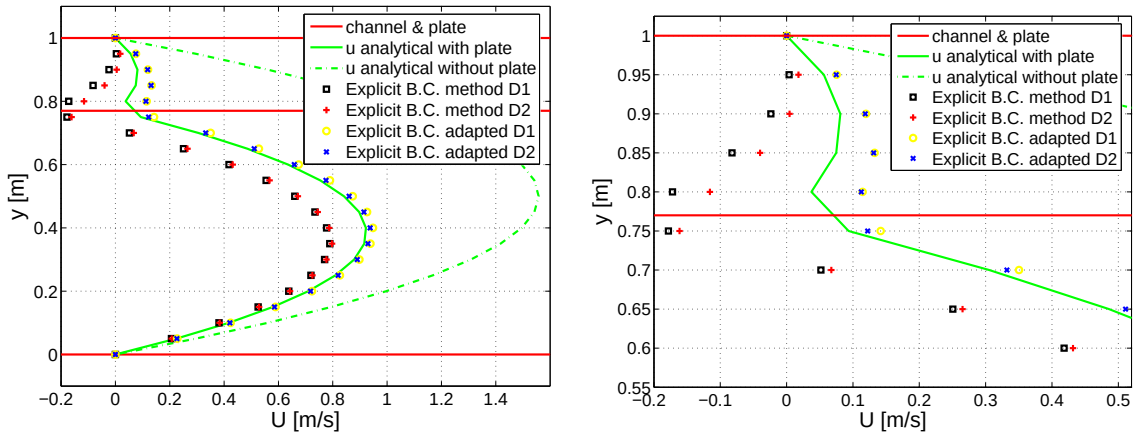


Figure 9.2: Analytical solution and numerical approximation (Explicit boundary condition method) for the Poiseuille flow. Original and adapted methods with distribution function $D1$ and $D2$.

9.4 The adapted method

Despite the fact that the influence of the plate on the flow is clearly visible and that the overall result for just 20 grid cells is not bad at all, there is one detail which is quite disturbing. The flow is calculated to be negative in the grid points next to the plate, which is obviously not realistic (see the detail in figure 9.2). This is related to the formulation of the method: when adding equations (9.1) and (9.3) earlier on, the possibility of multiplying (9.3) by a constant factor C was ignored completely, mainly to avoid problems with user defined parameters that have to be ”tuned” for each flow in order to obtain good results. However, the present method improved a lot by adding the constant factor and setting it for example equal to -1:

$$\big(1 - D(y - y_o)\big)\Big[\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} - \frac{1}{\mu}\frac{dp}{dx}\Big] + C \cdot D(y - y_o)\frac{u_i}{h^2} = 0$$

becomes:

$$u_{i+1}\Big[\frac{1 - D(y - y_o)}{h^2}\Big] + u_i\Big[\frac{D(y - y_o) - 2}{h^2}\Big] + u_{i-1}\Big[\frac{1 - D(y - y_o)}{h^2}\Big] = \big(1 - D(y - y_o)\big)\frac{1}{\mu}\frac{dp}{dx}. \quad (9.6)$$

From the result in figure 9.2 it can be seen that the velocity is never negative and that the accuracy is improved considerably. Tuning C to other negative values produced even slightly better results, but this approach seems worthless in higher dimensions since there is no way to experimentally evaluate the result, so the ”tuning” was abandoned alltogether. Similar results are found in [11] and [16], where the topic of user defined parameters and their influence on accuracy is discussed in detail.

More results, a comparison with the other methods and an extensive error analysis can be found in chapter 14.

Chapter 10

Method 2: Alternative methods

10.1 Introduction

Barry Koren suggested three alternative methods: the Alternative method A, B and C. In any of these cases the numerical scheme differs from the standard central scheme only in the grid points neighboring the immersed boundary. Contrary to scheme A, schemes B and C take into account that the flows above and under the plate are in fact not related and they treat the discretization accordingly in a predominantly one-sided manner. Scheme B replaces the second derivative in equation. (8.5) with a central scheme for non-equidistant grids. The idea behind methods A and C is essentially the same: they add an extra term $\frac{d^2u}{dy^2}$ to both sides of the governing equation:

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} - \frac{1}{\mu}\frac{dp}{dx} + C\Big(\frac{d^2u}{dy^2}\Big)_{y_o} = C\Big(\frac{d^2u}{dy^2}\Big)_{\Gamma}. \tag{10.1}$$

The constant C resembles the tuning factor from the previous chapter and is set here to 1 for simplicity. The term added to the left side of the equation is discretized over the boundary using u_{y_0} (which is later replaced by some inter/extrapolated expression), in the term in the righthand side u_{y_0} is replaced by the boundary condition $u_{y_0} = u_\Gamma = 0$.

10.2 Alternative Method A

As explained in the introduction, this method starts from equation (10.1), where the focus lies on the formulation of the $\frac{d^2u}{dy^2}$ terms that were added to the governing equation:

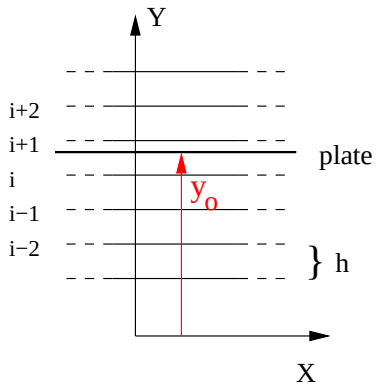


Figure 10.1: Situation sketch.

$$\Big(\frac{d^2u}{dy^2}\Big)_{y_o} = \frac{\frac{u_{i+2}-u_{y_o}}{y_{i+2}-y_o} - \frac{u_{y_o}-u_{i-1}}{y_o-y_{i-1}}}{\frac{y_{i+2}+y_o}{2} - \frac{y_o+y_{i-1}}{2}} = \frac{2}{3h}\Big(\frac{u_{i+2}-u_{y_o}}{y_{i+2}-y_o} - \frac{u_{y_o}-u_{i-1}}{y_o-y_{i-1}}\Big). \tag{10.2}$$

This expression makes use of grid points $i+2$ and $i-1$. If instead $i+1$ and i were used, the problem of division by 0 could arise when the immersed boundary lies close to grid point $i+1$ or i. The velocity at the immersed boundary u_{y_o} is now set to $u_\Gamma = 0$ in the added second derivative in the righthand side of equation (10.1), whereas it is replaced by the weighted average of the velocity in the neighboring points in the lefthand side of the same equation:

$$u_{y_o} = \frac{y_o - y_i}{h}u_{i+1} - \frac{y_{i+1}-y_o}{h}u_i.$$

Substituting all these expressions into equation (10.1) yields the numerical scheme for the grid point just below the immersed boundary, grid point i:

$$\begin{aligned} u_{i+1}\Big[\frac{1}{h^2} + \frac{2}{3h^2}(y_i-y_o)\Big(\frac{1}{y_{i+2}-y_o}+\frac{1}{y_o-y_{i-1}}\Big)\Big] \\ +u_i\Big[\frac{-2}{h^2} + \frac{2}{3h^2}(y_o-y_{i+1})\Big(\frac{1}{y_{i+2}-y_o}+\frac{1}{y_o-y_{i-1}}\Big)\Big] \\ +u_{i-1}\Big[\frac{1}{h^2}\Big] \quad = \quad \frac{1}{\mu}\frac{dp}{dx}. \end{aligned} \tag{10.3}$$

The stencil for the neighboring grid point above the immersed boundary (point $i+1$) is then just a mirrored version of equation (10.3):

$$\begin{aligned} u_{i+2}\Big[\frac{1}{h^2}\Big] \\ +u_{i+1}\Big[-\frac{2}{h^2} + \frac{2}{3h^2}(y_i-y_o)\Big(\frac{1}{y_{i+2}-y_o}+\frac{1}{y_o-y_{i-1}}\Big)\Big] \\ +u_i\Big[\frac{1}{h^2} + \frac{2}{3h^2}(y_o-y_{i+1})\Big(\frac{1}{y_{i+2}-y_o}+\frac{1}{y_o-y_{i-1}}\Big)\Big] \quad = \quad \frac{1}{\mu}\frac{dp}{dx}. \end{aligned} \tag{10.4}$$

10.3 Alternative Method B

This method is actually quite similar to method A. There are a few differences however: instead of using equation (10.1) as the basis for the scheme, the attention is now shifted to the direct discretization of the second derivative in the governing equation. Furthermore the stencil in method B divides the flow essentially in two parts: no grid points from above the plate are used in the schemes for the points under the immersed boundary and vice versa.

Following again the reasoning from the previous paragraph and taking into account the grid point numbering as in figure 10.1, the governing equation can be discretized as:

$$\begin{aligned} \Big(\frac{d^2u}{dy^2}\Big)_i &= \frac{\frac{u_{y_o}-u_i}{y_o-y_i} - \frac{u_i-u_{i-1}}{h}}{\frac{y_o+y_i}{2} - \frac{y_i+y_{i-1}}{2}} \\ &= \frac{2}{y_o-y_{i-1}}\Big(\frac{u_{y_o}-u_i}{y_o-y_i} - \frac{u_i-u_{i-1}}{h}\Big) = \frac{1}{\mu}\frac{dp}{dx} \\ \Rightarrow \qquad & u_i\Big[-\frac{1}{y_o-y_i} - \frac{1}{h}\Big] + u_{i-1}\Big[\frac{1}{h}\Big] = \frac{1}{\mu}\frac{dp}{dx}\Big(\frac{y_o-y_{i-1}}{2}\Big), \end{aligned} \tag{10.5}$$

and for the neighboring grid point above the immersed boundary:

$$\Big(\frac{d^2u}{dy^2}\Big)_{i+1} = \frac{\frac{u_{i+2}-u_{i+1}}{h} - \frac{u_{i+1}-u_{y_o}}{y_{i+1}-y_o}}{\frac{y_{i+2}+y_{i+1}}{2} - \frac{y_{i+1}+y_o}{2}}$$

$$\Rightarrow \quad u_{i+1}\Big[-\frac{1}{y_{i+1}-y_o} - \frac{1}{h}\Big] + u_{i+2}\Big[\frac{1}{h}\Big] = \frac{1}{\mu}\frac{dp}{dx}\Big(\frac{y_{i+2}-y_o}{2}\Big). \tag{10.6}$$

These expressions are the central discretizations of the second derivative on a non-uniform mesh. Because this scheme is second-order accurate (proof by substituting Taylor expansions in equation (10.5)), because the flows above and under the plate are solved independently and since the governing equation has a second-order polynomial as solution, this method is expected to do very well, at least where accuracy is concerned. More on that in chapter 14.

10.4 Alternative Method C

Method C is again based on equation (10.1), but makes use of Taylor series expansions with respect to y_o to evaluate the added second derivatives. This is shown for the grid point just below the immersed boundary, see figure 10.1 for definition of the grid point indices. First the Taylor series expansions of the velocities u_i and u_{i-1} around the velocity on the immersed boundary u_{y_o} are written out:

$$u_i = u_{y_o} + (y_i - y_o)\Big(\frac{du}{dy}\Big)_{y_o} + \frac{(y_i-y_o)^2}{2}\Big(\frac{d^2u}{dy^2}\Big)_{y_o} + O(y_i-y_o)^3, \tag{10.7}$$

$$u_{i-1} = u_{y_o} + (y_{i-1} - y_o)\Big(\frac{du}{dy}\Big)_{y_o} + \frac{(y_{i-1}-y_o)^2}{2}\Big(\frac{d^2u}{dy^2}\Big)_{y_o} + O(y_{i-1}-y_o)^3. \tag{10.8}$$

Elimination of $\left(\frac{d^2u}{dy^2}\right)_{y_o}$ from equations (10.7) and (10.8) yields:

$$\Big(\frac{d^2u}{dy^2}\Big)_{y_o} = \frac{2}{(y_i-y_o)(y_i-y_{i-1})}u_i - \frac{2}{(y_{i-1}-y_o)(y_i-y_{i-1})}u_{i-1} + \frac{2}{(y_i-y_o)(y_{i-1}-y_o)}u_{y_o}. \tag{10.9}$$

Replace u_{y_o} with 0 to obtain $\left(\frac{d^2u}{dy^2}\right)_\Gamma$:

$$\Big(\frac{d^2u}{dy^2}\Big)_\Gamma = \frac{2}{(y_i-y_o)(y_i-y_{i-1})}u_i - \frac{2}{(y_{i-1}-y_o)(y_i-y_{i-1})}u_{i-1}. \tag{10.10}$$

Substitute (10.9) and (10.10) in (10.1) to find the discrete equation according to method C for the grid point directly under the plate:

$$\frac{u_{i+1}-2u_i+u_{i-1}}{h^2} + \frac{2}{(y_i-y_o)(y_{i-1}-y_o)}u_{y_o} = \frac{1}{\mu}\frac{dp}{dx}. \tag{10.11}$$

The only thing which is still missing is an expression for u_{y_o}. Therefore the Taylor series expansion procedure from the beginning of this paragraph is followed again, albeit with only the first two terms in the righthand side of equations (10.7) and (10.8). As before, eliminate $\left(\frac{du}{dy}\right)_{y_o}$. This yields then:

$$u_{y_o} = \frac{y_o - y_{i-1}}{y_i - y_{i-1}}u_i - \frac{y_o - y_i}{y_i - y_{i-1}}u_{i-1}. \tag{10.12}$$

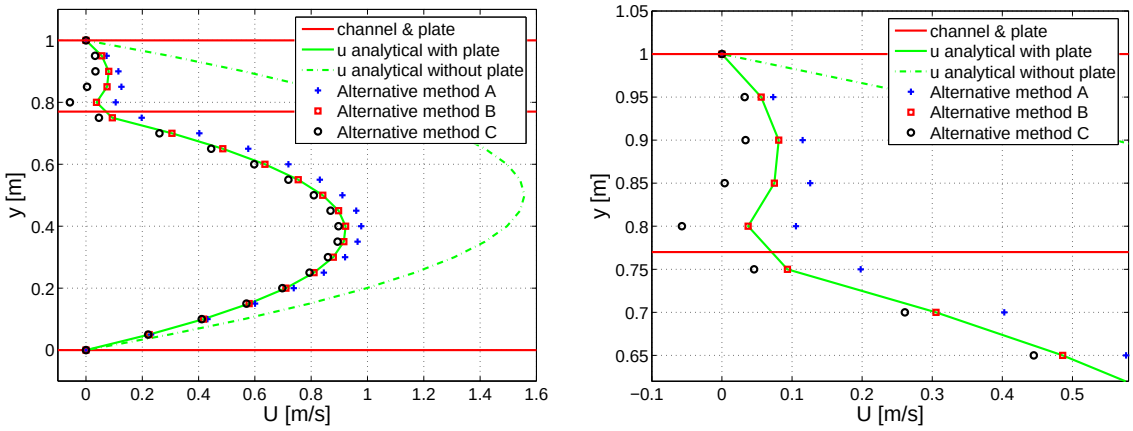


Figure 10.2: Analytical solution and numerical approximation (Alternative methods A, B and C) for the Poiseuille flow

This expression is a linear extrapolation of the velocity from the nearest two grid points under the immersed boundary to the plate itself. Substituting the expression for u_{y_o} in (10.11) yields the numerical scheme valid for the first grid point under the plate (point i):

$$\begin{aligned} u_{i+1}\Big(\frac{1}{h^2}\Big) + u_i\Big(-\frac{2}{h^2} + \frac{2}{(y_o - y_i)h}\Big) & \\ + u_{i-1}\Big(\frac{1}{h^2} - \frac{2}{(y_o - y_{i-1})h}\Big) &= \frac{1}{\mu}\frac{dp}{dx}. \end{aligned} \tag{10.13}$$

The stencil for the neighboring grid point above the plate can be derived similarly, for brevity only the result is included:

$$\begin{aligned} u_{i+1}\Big(\frac{1}{h^2} - \frac{2}{(y_{i+1} - y_o)h}\Big) + u_i\Big(-\frac{2}{h^2} + \frac{2}{(y_i - y_o)h}\Big) & \\ + u_{i-1}\Big(\frac{1}{h^2}\Big) &= \frac{1}{\mu}\frac{dp}{dx}. \end{aligned} \tag{10.14}$$

In figure 10.2, the approximation of the analytical solution by the three Alternative methods is shown. The parameters are the same as for figure 9.2: $N = 20$ and $y_o = 0.77$. Complying with expectations, the Alternative method B seems to be exact and coincides with the analytical solution. More results and analysis are given at the end of Part II, the subject is treated in depth in chapter 14.

10.5 Note

At this point, a few preliminary conclusions seem appropriate. Methods A and C are both based on equation (10.1). And although the added second derivatives in this equation seem to push the numerical solution computed with methods A and C in the right direction, they represent an inherent flaw in the methods as well. Because the added terms cancel each other in the case where the numerical approximation equals the analytical solution, no ”forcing” terms are left. Without these forcing terms, the solution should be the parabolic profile (equation (8.7)). As a result, schemes A and C are not able to converge to the exact solution. Method B is developed as a finite differences method, but it will be shown in chapter 12 that it is in fact the same as the quadratic Cut cell method.

Chapter 11

Method 3: Ghost cell method

The Ghost cell method is a well established Immersed Boundary Method, mainly because of its simplicity and the ease with which it can be implemented. For the Poiseuille flow model problem the linear and quadratic extrapolation schemes are investigated. How the technique works was explained in chapter 5, however, applying it to an body with infinitesimal thickness may sound rather unusual. This aspect is treated in this section, followed by the extrapolation stencils and a graph showing the resulting numerical solution.

For the sake of simplicity, let us consider for the moment only the flow under the dividing plate in the channel. The standard central discretization (see equation. (11.1)) of the governing equation will encounter a problem only in the point neighboring the immersed boundary, grid point i in figure 11.1. The stencil for this point makes use of the velocity in point G, which lies just above the immersed boundary. As "seen" from the flow below the plate, this point is the first grid point to lie beyond the immersed boundary and thus becomes a ghost point. It will get a fictitious velocity u_G appointed to it for use in the numerical scheme mentioned above, but this velocity has nothing to do with the real velocity at this point from the flow above the plate!

$$\frac{u_G - 2u_i + u_{i-1}}{h^2} = \frac{1}{\mu}\frac{dp}{dx}. \tag{11.1}$$

The fictitious velocity in the ghost point is determined by extrapolation from the velocities at the immersed boundary and one or more grid points in the fluid below the plate. Since the analytical solution of the differential equation under consideration (governing equation (9.1)) is of second order only, here it makes

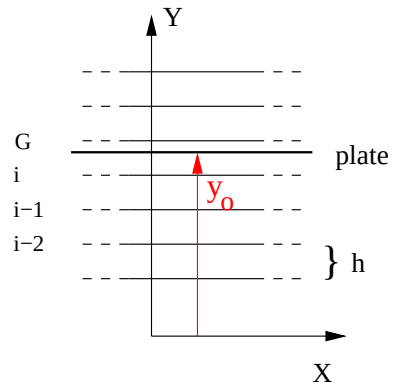


Figure 11.1: Grid point indices for the Ghost cell method, flow below the plate. G represents the ghost point.

no sense to look at higher order extrapolation schemes than a quadratic stencil. This quadratic extrapolation will already approximate the analytical solution exactly.

11.1 Linear extrapolation

Consider the Taylor series expansion of the velocity around the immersed boundary at y_o:

$$u = u_\Gamma + (y - y_o)\Big(\frac{du}{dy}\Big)_{y_o} + \frac{(y-y_o)^2}{2}\Big(\frac{d^2u}{dy^2}\Big)_{y_o} + O\big((y-y_o)^3\big). \tag{11.2}$$

In the linear approximation, all the terms in the right hand side of this expression are dropped, except the first two. The remaining first derivative is then discretized as the weighted average of two discrete first derivatives (y_G is the coordinate of the ghost point):

$$\Big(\frac{du}{dy}\Big)_{y_o} = \frac{y_o - y_i}{h}\frac{u_G - u_\Gamma}{y_G - y_o} + \frac{y_G - y_o}{h}\frac{u_\Gamma - u_i}{y_o - y_i}. \tag{11.3}$$

This discretization is preferred above the easier, one-sided discretization

$$\Big(\frac{du}{dy}\Big)_{y_o} = \frac{u_\Gamma - u_i}{y_o - y_i},$$

since it is $O(h^2)$ accurate, unlike the one-sided expression, which is an $O(h)$ approximation (see [14]). The order of accuracy can be proved by evaluating the Taylor expansion, equation (11.2), in the ghost point and in grid point i, followed by elimination of the second derivative from the expressions:

$$\begin{aligned}(y_o - y_i)^2 u_G - (y_G - y_o)^2 u_i \;\; &= \;\; u_\Gamma\Big((y_o - y_i)^2 - (y_G - y_o)^2\Big) \\ &\quad + \Big(\frac{du}{dy}\Big)_{y_o}\Big((y_o - y_i)(y_G - y_o)(y_G - y_i)\Big) + O(h^5).\end{aligned} \tag{11.4}$$

This expression can be rewritten to yield equation (11.3) plus second-order terms. There is no difference whether the one-sided discretization or equation (11.3) is used in the linear extrapolation stencil, in the quadratic scheme however they do not produce the same result.

The next step is to substitute equation (11.3) into (11.2), set $u_\Gamma = 0$, evaluate the Taylor series in the ghost point u_G and simplify:

$$u_G = -\frac{y_G - y_o}{y_o - y_i}u_i. \tag{11.5}$$

This expression is then substituted into the central discretization (adapted for the Ghost cell method):

$$\frac{u_G - 2u_i + u_{i-1}}{h^2} = \frac{1}{\mu}\frac{dp}{dx}, \tag{11.6}$$

and the linear Ghost cell scheme results:

$$\frac{u_i}{h^2}\Big(-2 + \frac{y_o - y_G}{y_o - y_i}\Big) + \frac{u_{i-1}}{h^2} = \frac{1}{\mu}\frac{dp}{dx}. \tag{11.7}$$

A similar derivation for the flow above the plate shows the expression for the numerical scheme in the grid point just above the immersed boundary:

$$\frac{u_i}{h^2}\Big(-2 + \frac{y_o - y_G}{y_o - y_i}\Big) + \frac{u_{i+1}}{h^2} = \frac{1}{\mu}\frac{dp}{dx}. \tag{11.8}$$

Note that the coordinate y_G has now changed, since the ghost cell has become a different cell altogether!

11.2 Quadratic extrapolation

The quadratic extrapolation scheme results when all but the first three terms are dropped in the right hand side of equation (11.2). The remaining first and second derivatives are then discretized and substituted in the Taylor series. Both discrete derivatives need to be second order accurate for this extrapolation stencil to be truly quadratic. The first derivative is approximated by equation (11.3), the discretization of the second derivative is similar to the one used in Alternative method B, presented in section 10.3:

$$\Big(\frac{d^2u}{dy^2}\Big)_{y_o} = \frac{2}{h}\left[\frac{u_G - u_\Gamma}{y_G - y_o} - \frac{u_\Gamma - u_i}{y_o - y_i}\right] \tag{11.9}$$

Expressions (11.3) and (11.9) are substituted into (11.2), and the Taylor series is then evaluated in grid point $i-1$ (this is done to include information from the "known" velocities in the 3 points closest to the immersed boundary: y_Γ, y_i and y_{i-1}):

$$\begin{aligned} u_{i-1} &= u_\Gamma + (y_{i-1} - y_o)\left[\frac{y_o - y_i}{h}\frac{u_G - u_\Gamma}{y_G - y_o} + \frac{y_G - y_o}{h}\frac{u_\Gamma - u_i}{y_o - y_i}\right] \\ &\quad + \frac{(y_{i-1} - y_o)^2}{2}\frac{2}{h}\left[\frac{u_G - u_\Gamma}{y_G - y_o} - \frac{u_\Gamma - u_i}{y_o - y_i}\right] \end{aligned} \tag{11.10}$$

Setting $u_\Gamma = 0$ and rewriting gives an expression for the velocity in the ghost point in terms of u_i, u_{i-1} (and u_Γ):

$$u_G = -\frac{(y_G - y_o)(y_G - y_{i-1})}{h(y_o - y_i)}u_i + \frac{y_G - y_o}{y_o - y_{i-1}}u_{i-1} \tag{11.11}$$

Substition in (11.6) results in the numerical scheme for the quadratic extrapolation Ghost cell method:

$$\frac{u_i}{h^2}\Big(-\frac{(y_G - y_o)(y_G - y_{i-1})}{h(y_o - y_i)} - 2\Big) + \frac{u_{i-1}}{h^2}\Big(\frac{y_G - y_o}{y_o - y_{i-1}} + 1\Big) = \frac{1}{\mu}\frac{dp}{dx} \tag{11.12}$$

The derivation for the upper flow is now trivial, only the final stencil is presented:

$$\frac{u_i}{h^2}\Big(-\frac{(y_o - y_G)(y_{i+1} - y_G)}{h(y_i - y_o)} - 2\Big) + \frac{u_{i+1}}{h^2}\Big(\frac{y_o - y_G}{y_{i-1} - y_o} + 1\Big) = \frac{1}{\mu}\frac{dp}{dx} \tag{11.13}$$

As with the linear extrapolation, remember that the indices and ghost cell have changed.

Both Ghost cell methods were analyzed in the Matlab program, a plot of the results can be found in figure 11.2. The parameters are the same as for the previous methods: $N = 20$ and $y_o = 0.77$.

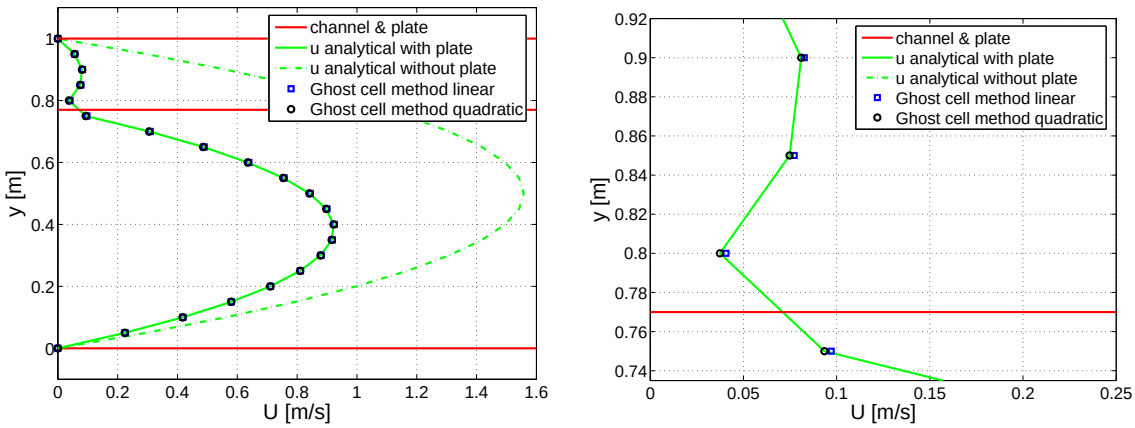


Figure 11.2: Analytical solution and numerical approximation (linear and quadratic Ghost cell methods) for the Poiseuille flow.

Chapter 12

Method 4: Cut cell method

All the methods treated so far were based on finite differences. The Cut cell method however is a finite volume method, regarded by many as the natural way to treat flow problems because of the inherent conservation of mass and momentum. The technique was already mentioned in chapter 5, but to avoid any misunderstanding it is derived briefly in this section. The first difference with respect to the previously treated IBMs is the grid. To facilitate implementation of the boundary conditions at the walls and to allow for accurate comparison between methods, the cell-centered grid points were left in the same position as before. This means that the grid starts and ends with half a cell, see figure 12.1. Since the flow problem is 1D, the cells are infinite in X direction. This general grid allows the derivation of the finite volume approach, the modifications needed in the vicinity of the immersed boundary will be dealt with later.

12.1 General numerical scheme

As before, the governing equation

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\mu}\frac{dp}{dx} \tag{12.1}$$

is the starting point for explaining the Cut cell method. This expression can be rewritten using the Nabla operator ∇, which is defined as a vector containing the derivatives to the different coordinates j ($\frac{\partial}{\partial x_j}$):

$$\nabla \cdot \nabla u = \frac{1}{\mu}\frac{dp}{dx}. \tag{12.2}$$

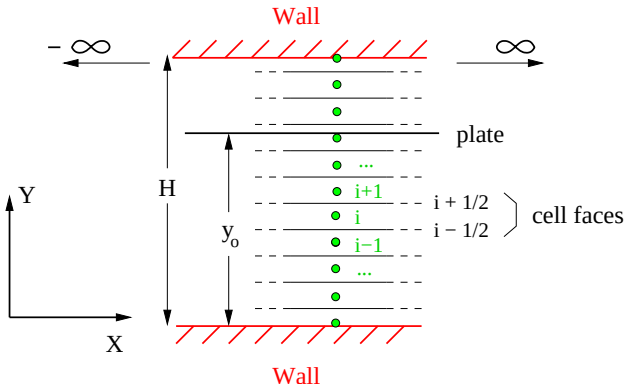


Figure 12.1: The general grid for the Cut cell method.

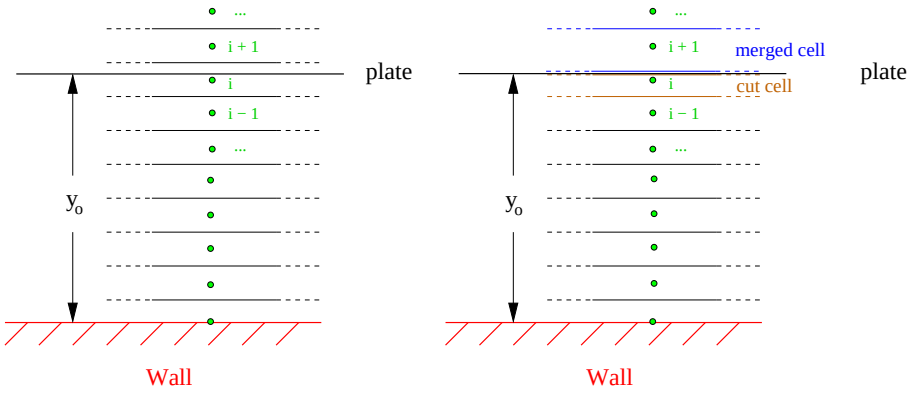


Figure 12.2: Original grid (left) and grid modified according to Cut cell approach (right).

Note that $\nabla \cdot \nabla u$ can be seen as the divergence of the gradient of u. Integration over a finite volume element Ω_i, or "cell" (see again figure 12.1) gives:

$$\int_{\Omega_i} \nabla \cdot \nabla u \; d\Omega = \int_{\Omega_i} \frac{1}{\mu} \frac{dp}{dx} \; d\Omega. \tag{12.3}$$

Using the divergence theorem, the lefthand side of eq. (12.3) can be transformed into:

$$\int_{\Omega_i} \nabla \cdot \nabla u \; d\Omega = \oint_{\Gamma_i} \nabla u \cdot \mathbf{n} \; ds, \tag{12.4}$$

where Γ_i is the boundary of the cell, $\mathbf{n}$ the unit outward normal vector on Γ_i and s the coordinate along Γ_i.

Substitution of (12.4) into (12.3) yields:

$$\oint_{\Gamma_i} \frac{du}{dy} \cdot \mathbf{n} \; ds = \frac{1}{\mu} \frac{dp}{dx} \; h, \tag{12.5}$$

thus

$$\left.\frac{du}{dy}\right|_{i+\frac{1}{2}} - \left.\frac{du}{dy}\right|_{i-\frac{1}{2}} = \frac{1}{\mu} \frac{dp}{dx} \; h. \tag{12.6}$$

The value of $\frac{du}{dy}$ on the cell faces needs to be approximated using values of u in the cell centres. With central discretizations equation (12.6) becomes:

$$\begin{aligned} \frac{u_{i+1} - u_i}{h} - \frac{u_i - u_{i-1}}{h} &= \frac{1}{\mu} \frac{dp}{dx} \; h \\ \Rightarrow \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} &= \frac{1}{\mu} \frac{dp}{dx}. \end{aligned} \tag{12.7}$$

This scheme was already used extensively in its finite difference form in all previous methods.

12.2 Treatment of the immersed boundary: Cut cell approach

The cell which is cut by the immersed boundary is split at the plate, see figure 12.2. The part containing the cell center remains an independent grid cell, but the other part merges with its neighbor. This means that the cell size is no longer constant. Considering the flow under the plate and cell i in particular, equation (12.5) changes to:

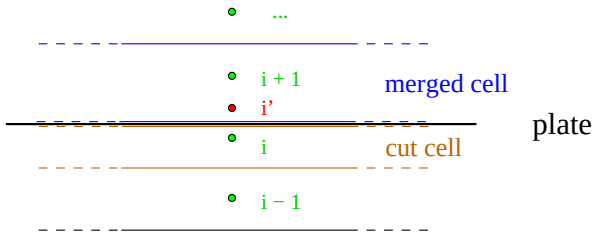


Figure 12.3: Sketch of the mirror point i' as used in the quadratic extrapolation stencil.

$$\oint_{\Gamma_i} \frac{du}{dy} \cdot \vec{n}\ ds = \frac{1}{\mu}\frac{dp}{dx}\,(y_o - y_{i-\frac{1}{2}}) \tag{12.8}$$

$$\Rightarrow \frac{du}{dy}\Big|_{y_o} - \frac{du}{dy}\Big|_{i-\frac{1}{2}} = \frac{1}{\mu}\frac{dp}{dx}\,(y_o - y_{i-\frac{1}{2}}), \tag{12.9}$$

with $y_{i-\frac{1}{2}} = \frac{y_i + y_{i-1}}{2}$. The $\frac{du}{dy}\Big|_{i-\frac{1}{2}}$ term is replaced by the central discretization $\frac{u_i - u_{i-1}}{h}$, but this is not that straightforward with the $\frac{du}{dy}\Big|_{y_o}$ term. Two alternatives will be considered here: linear and quadratic extrapolation.

12.2.1 Linear extrapolation

The $\frac{du}{dy}\Big|_{y_o}$ term is replaced by $\frac{u_\Gamma - u_i}{y_o - y_i}$, to yield the following discrete equation:

$$u_i\left[-\frac{1}{y_o - y_i} - \frac{1}{h}\right] + u_{i-1}\left[\frac{1}{h}\right] = \frac{1}{\mu}\frac{dp}{dx}\left(y_o - \frac{y_i + y_{i-1}}{2}\right). \tag{12.10}$$

Above the plate the equation becomes:

$$u_{i+1}\left[-\frac{1}{y_{i+1} - y_o} - \frac{1}{h}\right] + u_{i+2}\left[\frac{1}{h}\right] = \frac{1}{\mu}\frac{dp}{dx}\left(\frac{y_{i+1} + y_{i+2}}{2} - y_o\right). \tag{12.11}$$

12.2.2 Quadratic extrapolation

In this case the first derivative is replaced by the central discretization:

$$\frac{du}{dy}\Big|_{y_o} = \frac{u_{i'} - u_i}{2(y_o - y_i)}. \tag{12.12}$$

This expression employs point i' which is the mirror of point i with respect to the immersed boundary, see figure 12.3. The distance between these two points equals $2(y_o - y_i)$. $u_{i'}$ is the velocity in point i' which can be determined using a scheme similar to the quadratic extrapolation scheme from the ghost cell method, (chapter 11, equation (11.11)):

$$u_{i'} = u_i\left[\frac{y_i + y_{i-1} - 2y_o}{h}\right] + u_{i-1}\left[\frac{2(y_o - y_i)^2}{(y_o - y_i)(y_{i-1} - y_o) + (y_{i-1} - y_o)^2}\right]. \tag{12.13}$$

This expression is then simplified and substituted into (12.12), yielding:

$$\frac{du}{dy}\Big|_{y_o} = u_i\left[-\frac{1}{(y_o - y_i)} - \frac{1}{h}\right] + u_{i-1}\left[\frac{y_o - y_i}{h(y_o - y_{i-1})}\right]. \tag{12.14}$$

Finally, substitute (12.14) in (12.9) to find the stencil for the cell right under the immersed boundary:

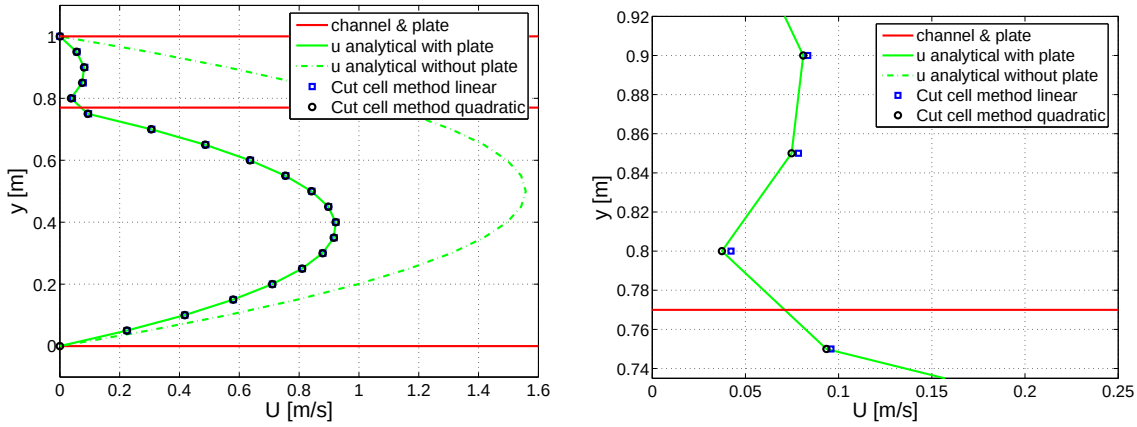


Figure 12.4: Analytical solution and numerical approximation (linear and quadratic Cut cell methods) for the Poiseuille flow.

$$u_i\left[-\frac{1}{y_o-y_i}-\frac{2}{h}\right]+u_{i-1}\left[\frac{y_o-y_i}{h(y_o-y_{i-1})}+\frac{1}{h}\right]=\frac{1}{\mu}\frac{dp}{dx}\left(y_o-\frac{y_i+y_{i-1}}{2}\right), \tag{12.15}$$

and for the cell above the plate:

$$u_{i+1}\left[-\frac{1}{y_{i+1}-y_o}-\frac{2}{h}\right]+u_{i+2}\left[\frac{y_{i+1}-y_o}{h(y_{i+2}-y_o)}+\frac{1}{h}\right]=\frac{1}{\mu}\frac{dp}{dx}\left(\frac{y_{i+1}+y_{i+2}}{2}-y_o\right). \tag{12.16}$$

Both the linear and the quadratic Cut cell method were programmed in Matlab, and an impression of the results can be found in figure 12.4.

12.3 Note

The quadratic Cut cell method resembles Alternative method B (chapter 10) in the way that they both resize the cell that is cut by an immersed boundary and adapt the stencil for that cell. The numerical schemes are not identical however: this is due to the fact that Alternative method B uses the point in the middle between the immersed boundary y_o and the grid point y_i as a support, while the Cut cell method calculates a flux at y_o.

Chapter 13

Method 5: Analytical forcing method

One nice feature that is specific for this simple flow problem is that the governing equation is analytically integrable. The advantages of this property are that it is possible to compute an analytical solution and compare any numerical approximation to it. Moreover, it allows the exact determination of the added forcing term (see chapter 3), something that is in general not possible in Immersed Boundary Methods. Consider for instance the following equation:

$$\frac{d^2u}{dy^2} = \frac{1}{\mu}\frac{dp}{dx} + F\delta(y - y_o), \tag{13.1}$$

the governing equation with an extra forcing term F in the right hand side. Physically, the forcing term could be interpreted in this situation as the force needed to overcome the friction on the plate and keep it in place. The Dirac delta function makes sure that the forcing term is only applied at y_o, on the immersed boundary.

13.1 Analytical derivation of the forcing term

Equation (13.1) will now be solved analytically. For this purpose, the expression has to be integrated twice, followed by the determination of the integration constants, using the boundary conditions. To get this right, it is quite useful to take a look at the definition of the Dirac delta function (a good source of information is [31]). The Dirac delta function can be seen as the derivative of the Heaviside step function H, while the Heaviside function is in turn the derivative of the ramp function R (both functions are shown in figure 13.1):

$$\frac{d}{dy}[H(y)] = \delta(y) \qquad \text{and} \qquad \frac{d}{dy}[R(y)] = H(y). \tag{13.2}$$

The analytical forcing method can be found rather easy now, the first step is to integrate eq. (13.1) twice:

$$u = \frac{1}{2\mu}\frac{dp}{dx}y^2 + F \cdot R(y - y_o) + C_1 y + C_2, \tag{13.3}$$

C_1 and C_2 are integration constants. These can be determined by substituting the boundary conditions:

at $y = 0$, $u = 0$ (lower wall), yielding:

$$C_2 = 0 \tag{13.4}$$

and at $y = H$, $u = 0$ (upper wall), yielding:

$$C_1 = -\frac{1}{2\mu}\frac{dp}{dx}H - F \cdot (1 - \frac{y_o}{H}). \tag{13.5}$$

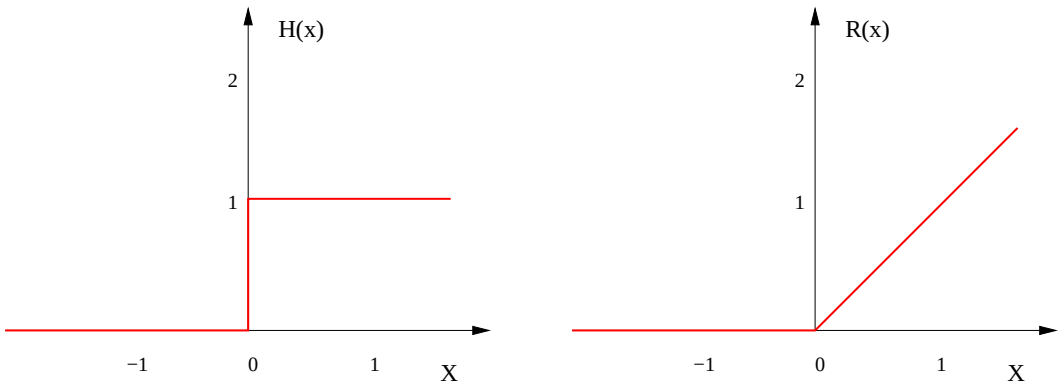


Figure 13.1: Heaviside step function H and ramp function R.

Finally, the forcing term F results from substitution of $u = 0$ at $y = y_o$, the boundary condition at the plate, yielding:

$$F = -\frac{H}{2\mu}\frac{dp}{dx} \tag{13.6}$$

Combining (13.6) and (13.1) yields the continuous formulation of this Immersed Boundary Method:

$$\frac{d^2u}{dy^2} = \frac{1}{\mu}\frac{dp}{dx} - \frac{H}{2\mu}\frac{dp}{dx}\delta(y - y_o) \tag{13.7}$$

13.2 Numerical schemes

Discretizing expression (13.7) is done in the standard finite difference way by replacing the second derivative with the central scheme.

$$u_{i+1}\left(\frac{1}{h^2}\right) + u_i\left(-\frac{2}{h^2}\right) + u_{i-1}\left(\frac{1}{h^2}\right) = \frac{1}{\mu}\frac{dp}{dx} - \frac{H}{2\mu}\frac{dp}{dx}\delta(y - y_o) \tag{13.8}$$

The Dirac delta function needs to be replaced by a smooth distribution function (as explained in chapters 5 and 9). Again a hat function($d1$ and $d2$) and a modified cosine ($d3$ and $d4$) were used, and both of them exist in two variations: a normal one with support over $2h$ and a ”wide” version which has a support of $4h$ (see figure 13.2).

The method was implemented in the Matlab program, and the result for 20 cells and a plate located at $0.77H$ is plotted in figure 13.3.

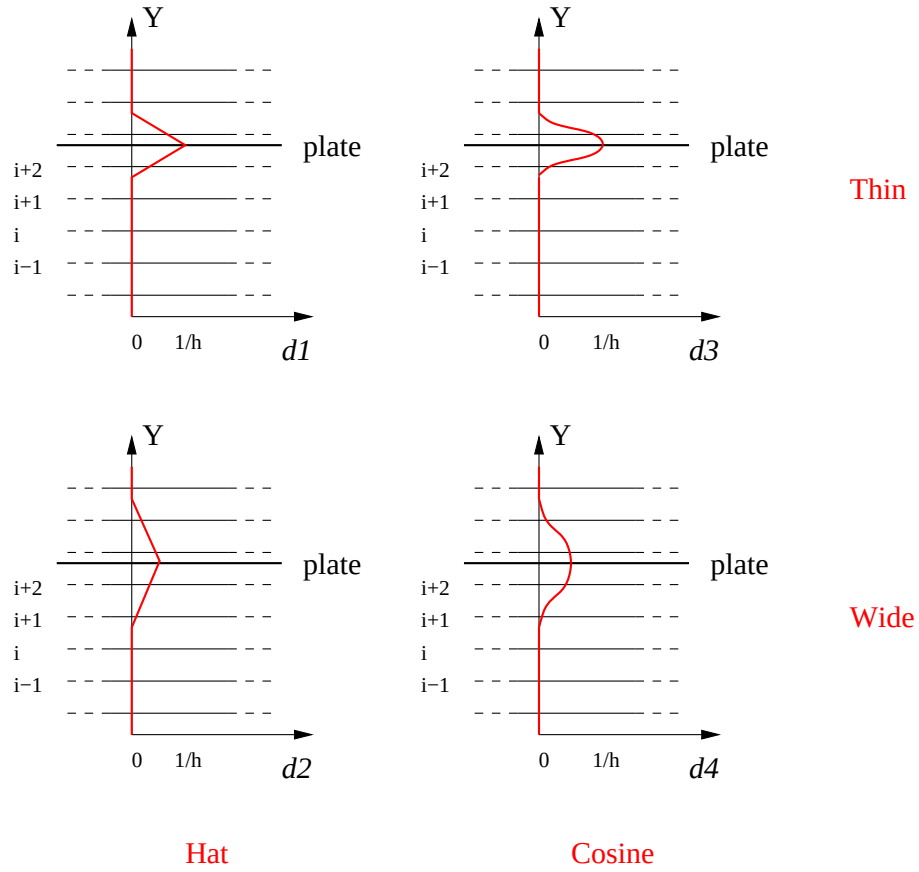


Figure 13.2: Smooth distribution functions $d1$, $d2$, $d3$ and $d4$.

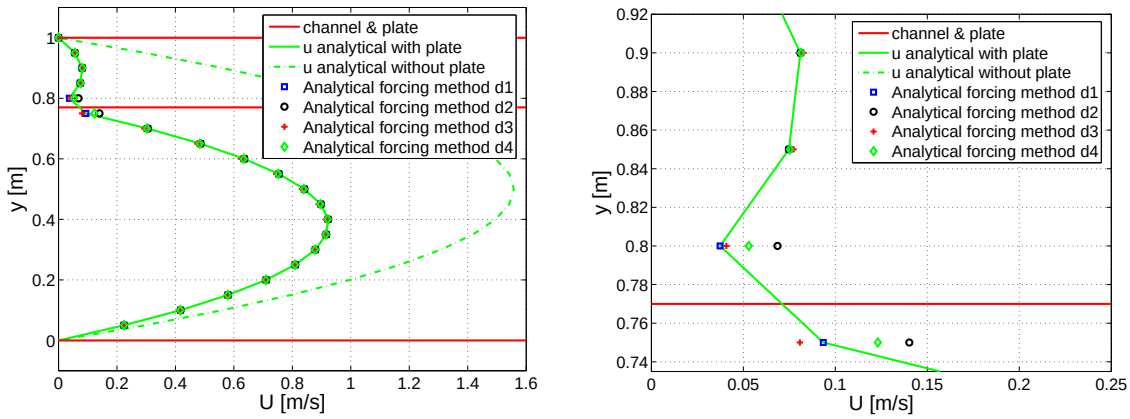


Figure 13.3: Analytical solution and numerical approximation (Analytical forcing method) for the Poiseuille flow.

Chapter 14

Method Comparison and Error Analysis

In this chapter, the results of the Immersed Boundary Methods that were applied to the Poiseuille problem are compared to each other, and the discrete L_2 errors in the solutions are analyzed. Given the goal of the one-dimensional test case (i.e. to select suitable IBMs for application to 2D problems), it was decided to focus attention on two topics: the comparison of the magnitude of the absolute error associated with each method on one grid and the grid size dependency of the errors. This is possible because the exact analytical solution of the Poiseuille flow is known.

14.1 Comparison of the results for all methods

As in the previous sections, the numerical approximations to the velocity profile are plotted for the case where $N = 20$ and $y_o = 0.77$. The rather small number of cells makes the difference between analytical solution and numerical approximation better visible, while the position of the plate is chosen such that it does not really favor any of the immersed boundary methods (more on this topic later on). The results (including details of the regions around plate and maximum velocity) are plotted in figures 14.1 through 14.4. The graphs are representative in the sense that the same trends can be seen in plots with more grid cells.

The first thing that attracts attention is the separation of ”exact” and ”approximate” numerical methods. Indeed, some methods (i.e. the Alternative method B, the quadratic Ghost cell and Cut cell methods and the Analytical Forcing method $d1$) turn out to be globally second-order accurate and thereby match the order of the solution to the Poiseuille problem. Their calculated velocity profile resembles the analytical result almost perfectly, and the very small errors can be appointed to data type specifications or machine precision limits. This comes as no big surprise: The Alternative method B as well as the quadratic Ghost and Cut cell methods separate the upper and lower flow completely and use second-order extrapolation schemes in the treatment of the immersed boundary. The only flaw of the Analytical Forcing method lies in the smooth distribution function used to approximate the Dirac delta function. Why the small hat function outperforms the others is still unclear, as this function is theoretically not second-order accurate.

However, there is a drawback to the great accuracy these four methods achieve: the L_2 errors they produce are effectively meaningless (error levels are of the same order as round-off errors) and cannot be used in the error analysis. They are therefore neglected in the remainder of this chapter. This does not mean that they are not to be considered when a choice needs to be made on the IBMs for the 2D code, on the contrary, they need to be taken very seriously since they are more accurate than any other method seen so far.

In figures 14.3 and 14.4 it can be seen that especially the linear Ghost and Cut cell methods and to a lesser extent the Analytical Forcing methods $d2$, $d3$ and $d4$ perform rather well, while Explicit boundary condition methods and the Alternative methods A and C lack accuracy near the immersed boundary. All

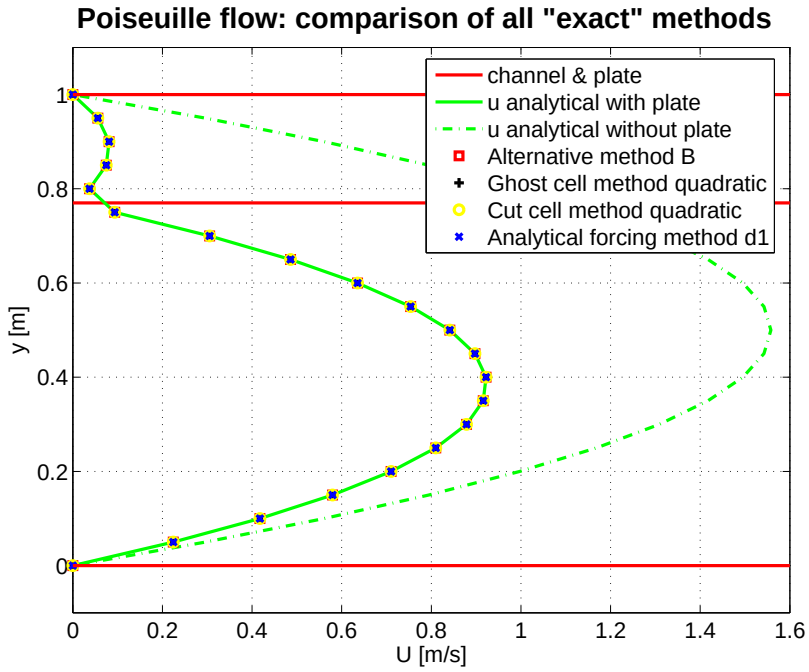


Figure 14.1: Comparison of the ”exact” numerical methods: velocity profile.

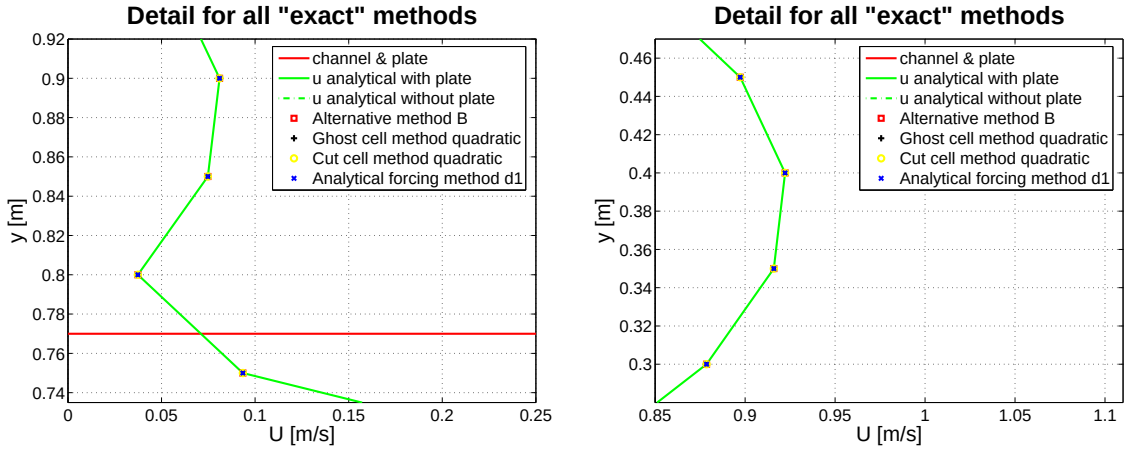


Figure 14.2: Details of the velocity profile: around the immersed boundary (left) and at the maximum velocity in the channel (right).

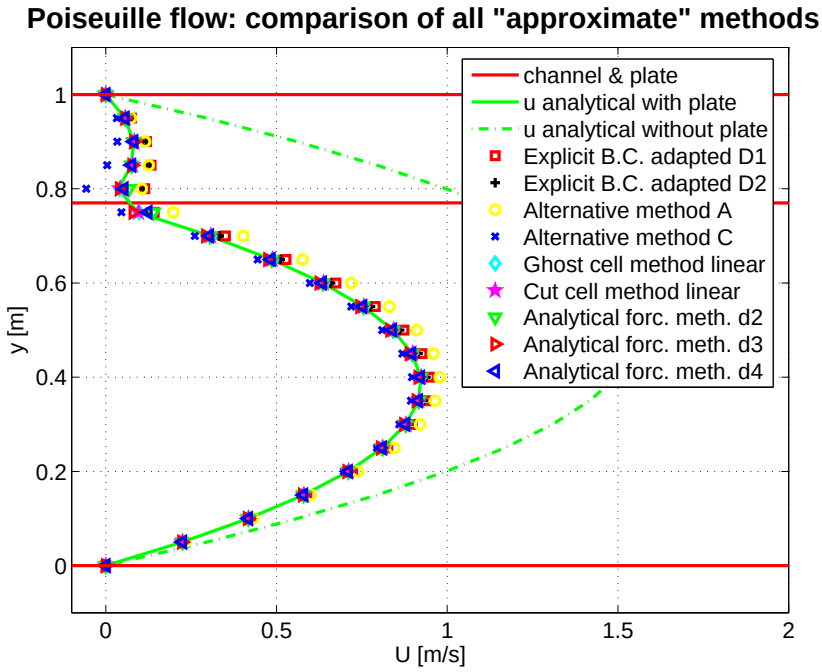


Figure 14.3: Comparison of the ”approximate” numerical methods: velocity profile.

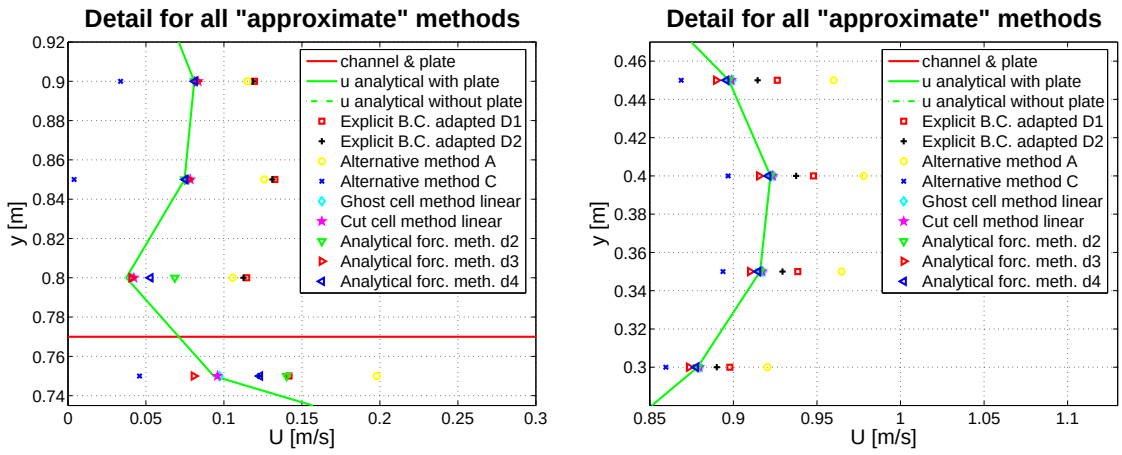


Figure 14.4: Details of the velocity profile: around the immersed boundary (left) and at the maximum velocity in the channel (right).

N	y_o
50	$0.79H$
100	$0.795H$
200	$0.7975H$
400	$0.79875H$
800	$0.799375H$

Table 14.1: Plate position y_o as a function of N

the methods converge to the analytical solution when more grid cells are used, this is explained in the following error analysis.

14.2 Relative grid convergence study

One of the fundamental differences between body-conformal grid methods and Immersed Boundary Methods is that in the former, the position of the boundary relative to the neighboring grid point is always the same. More exactly, the boundary is located on a cell face / grid point, irrespective of the grid size. This is not the case in IBMs: generally, a boundary will lie at a random distance from the nearest grid point, say $a \times h$, and this distance does depend on the number of cells. Double the number of cells to $2N$ and the distance will not be $a \times \frac{h}{2}$. Indeed, the relative position of the immersed boundary with respect to the cell it lies in changes with the grid size. This has implications for the error of a numerical method: consider for instance stencils which make use of linear extrapolation schemes.

So to get an idea of the dependency of the numerical error on the grid size without the effect of changing relative positions, a relative grid convergence study is performed. This means that over a range of grid sizes, the immersed boundary is moved such that it always remains in the exact middle between two grid points: the point which lies at $0.8H$ and the one below it. Table 14.2 lists the plate position y_o as a function of the number of grid points N.

Using these values for N and y_o the discrete L_2 errors are calculated for all the "approximate" numerical methods, where the L_2 norm is defined as (see also [30]):

$$||e||_{L_2} = \frac{\sqrt{\sum_{i=1}^{N+1}(u_{i_{exact}} - u_{i_{numerical}})^2}}{N}.$$

The L_2 norms are plotted against the number of cells N on a log-log scale as usual, see figures 14.5 and 14.6. Remark that the Analytical Forcing method $d3$ is plotted in a separate figure and that it is essentially "exact" (that is: the errors are of the smallest order of magnitude allowed by the data type). This is due to the distribution of the error over the cell, as discussed in the next subsection.

Essentially, all the lines seem to be straight in the first log-log plot, meaning that there is an unambiguous relation between the magnitude of the L_2 error and the grid size for every method. From the graph, it can be concluded that:

$$||e||_{L_2} \text{ for } \begin{array}{rcl} \text{Explicit B. C. method adapted } D1 \text{ \& } D2\text{, Alternative method A \& C} & \sim & O(h^{1.5}) \\ \text{Analytical forcing method } d2 \text{ \& } d4 & \sim & O(h^{2.0}) \\ \text{Linear Ghost and Cut cell method} & \sim & O(h^{2.5}) \end{array}$$

This information will prove valuable later on in the absolute grid convergence study.

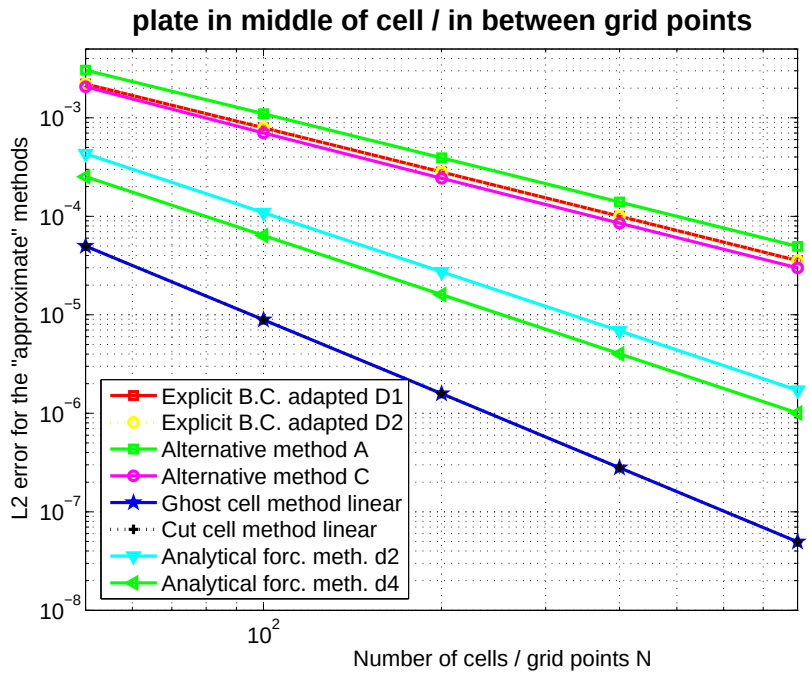


Figure 14.5: Relative grid convergence for the ”approximate” numerical methods.

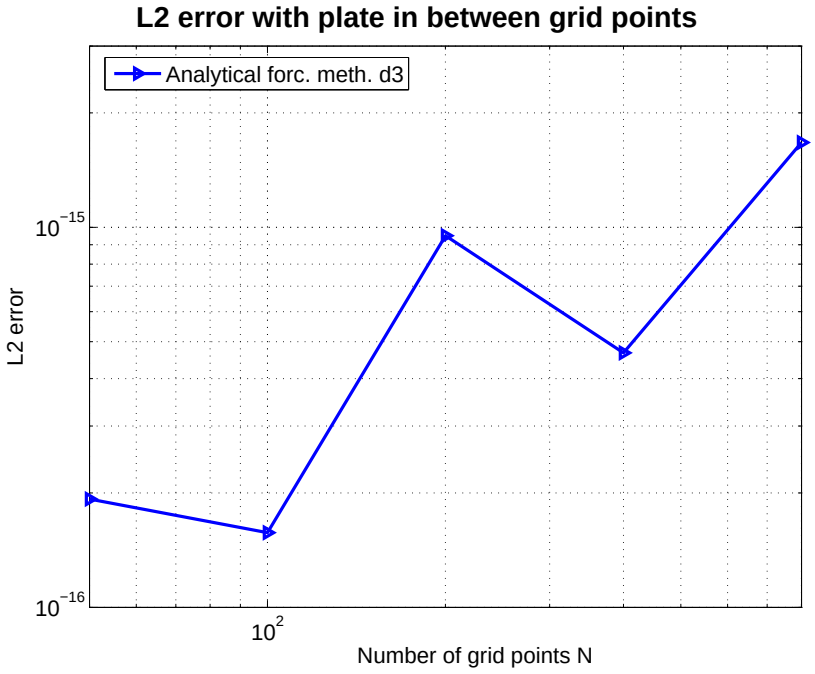


Figure 14.6: Relative grid convergence for the Analytical forcing method $d3$.

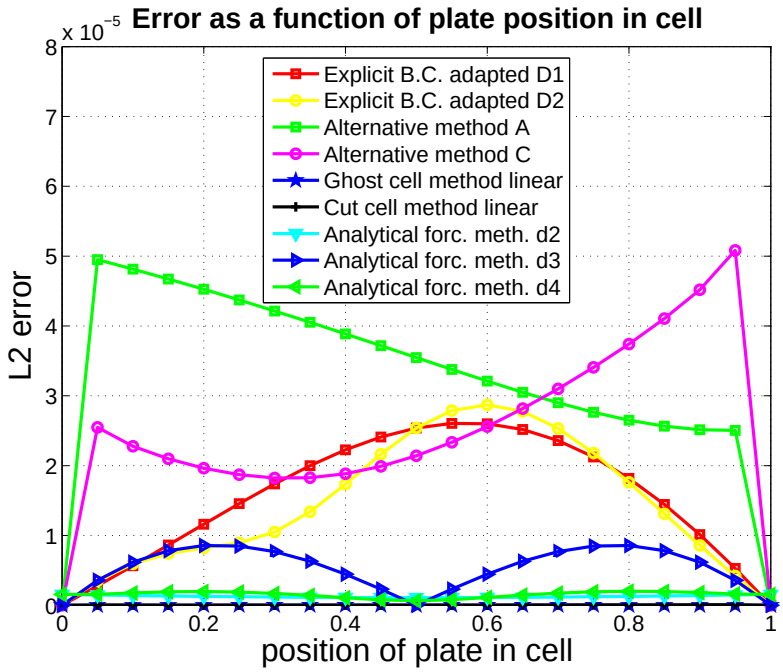


Figure 14.7: Error as function of immersed boundary location relative to the neighboring grid points for Explicit boundary condition method adapted $D1$ and $D2$, Alternative A, Alternative C and Analytical Forcing method $d3$.

14.3 Error dependence on plate position in cell

In fact, the relative grid convergence study is only accurate when the immersed boundary happens to fall right in between two grid points. This is of course very rarely the case, and therefore it is useful to have an idea about the dependency of the error on the plate position relative to the neighboring grid points: how does the relative position of the plate in a cell influence the error?

To assess this property, the following situation is considered: the channel is divided in 1000 cells, and the cell between $y = 0.799H$ and $y = 0.8H$ is subdivided in 20 equidistant positions. For every calculation, the plate is set at a new position in this cell, and the resulting L_2 norms for all the "approximate" numerical methods are then plotted on each position. The difference in error magnitude between the methods requires the result to be plotted in 3 different graphs: 14.7, 14.8 and 14.9.

In the figures it can be seen that the errors for the Alternative methods A and C and the linear Cut cell method go abruptly to zero near the grid points. This is due to a small change in the scheme that replaces the complicated immersed boundary treatment directly with the boundary condition should the plate coincide exactly with a grid point.

Another striking detail is the fact that the error for the Analytical Forcing method $d3$ is equal to zero when the immersed boundary is located in the middle between two grid points! This explains of course its strange behavior in the relative grid convergence study. The issue could have been dealt with by choosing another point relative to the cell to examine the relative grid convergence. However, since the Analytical Forcing method is not extendible to 2D, there is no real point in doing so.

Finally the error is analyzed as one would normally do, with a fixed plate and an increasing total number of grid points. The previous subsections will provide insight into the resulting figures.

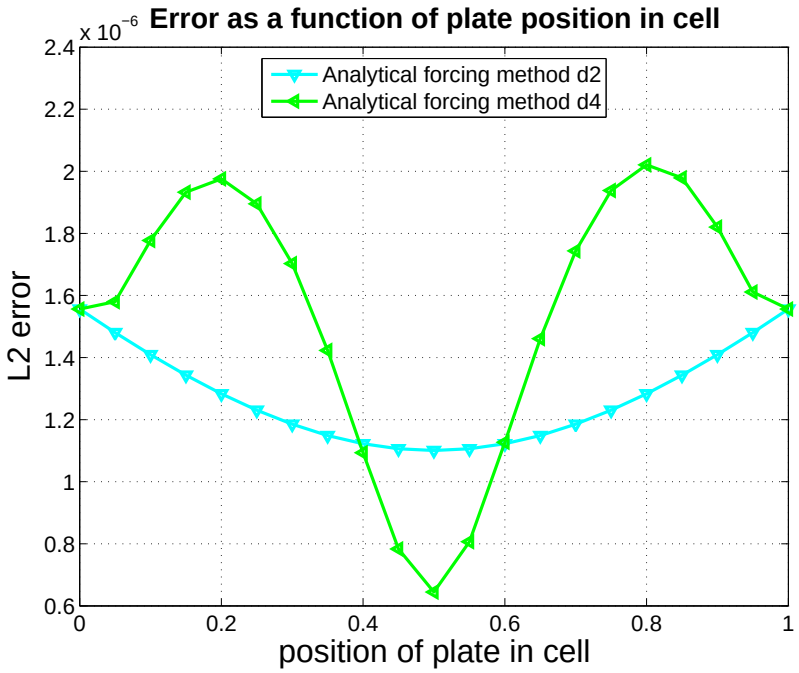


Figure 14.8: Error as function of immersed boundary location relative to the neighboring grid points for Analytical Forcing methods $d2$ and $d4$.

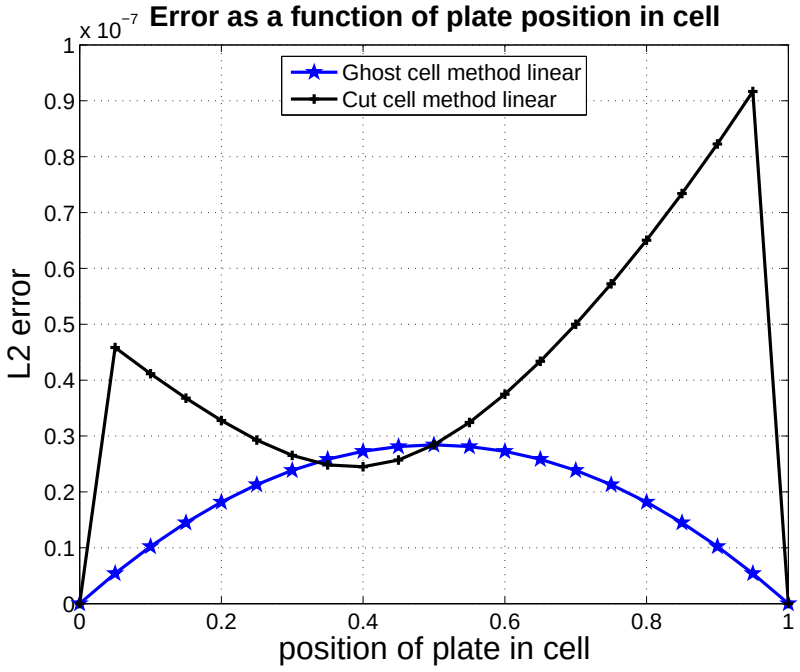


Figure 14.9: Error as function of immersed boundary location relative to the neighboring grid points for Linear Ghost and Cut cell methods.

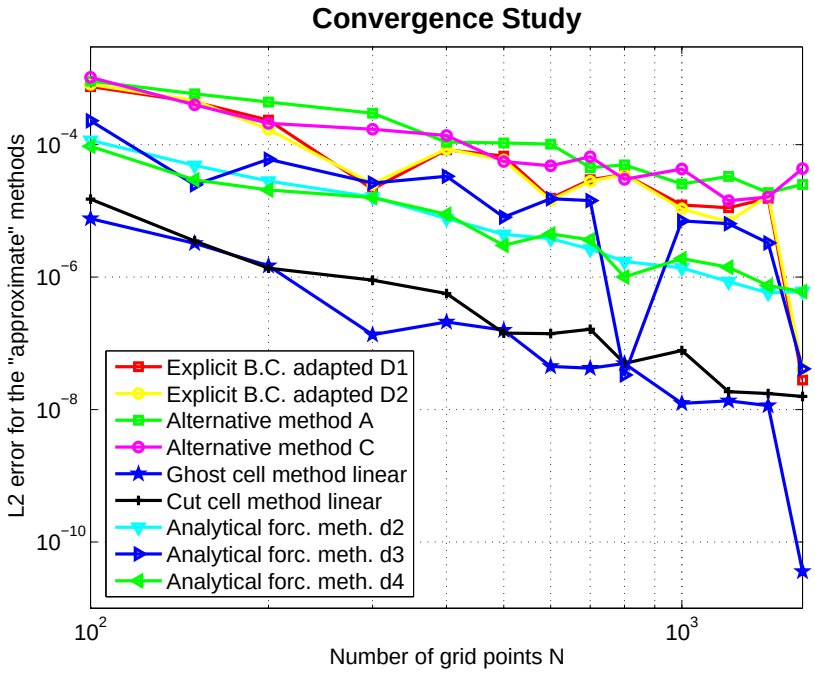


Figure 14.10: L_2 error for all the "approximate" numerical methods.

14.4 Absolute grid convergence study

This is the classical grid convergence study, with N ranging from 100 to 1600 cells and y_o set at 0.79687564. The value is really arbitrary, but this one was chosen such that plate and grid point would never coincide. Remember that the relative position between plate and cell face will change every time N changes, and that all the methods are "exact" in the fluid domain except for the near vicinity of the immersed boundary, so all the errors are directly related to the implementation of the IBMs.

Figure 14.10 shows the L_2 error norm for all the "approximate" immersed boundary methods as a function of the number of grid points N. It is conventionally plotted on a double logarithmic scale. The same graph is compared with the relative position of the plate with respect to the nearest grid point *under* the immersed boundary, figure 14.11. Finally, both the L_2 norm and the relative plate position are corrected for Alternative method C to show the relation between the two, see figure 14.12. The plots are discussed in detail below.

At first sight, graphs 14.10 and 14.11 are a little disturbing. The problem is that the relative position of the immersed boundary shifts on every new grid. The influence this has on the error was discussed in section 14.3. As a result, none of L_2 norms decrease in a monotonous way. This means that one can never be sure that a calculation with more cells will yield more accurate results than a calculation with fewer grid points. However, the problem is not as big as it seems, since it is possible to formulate a maximum for the error as a function of the number of grid points which is monotonous. This is done by combining the maximum error for a method from the error distribution study with the error-cell size relationships as found in the relative grid convergence assessment.

Furthermore, the influence of the changing relative position of the immersed boundary in a cell with increasing grid size turns out to be substantial. The nice straight lines from the relative convergence study are transformed into seemingly random scribbles. However, the relation between relative plate position and L_2 error norm can be visualized by correcting both L_2 norm and relative plate position. An example

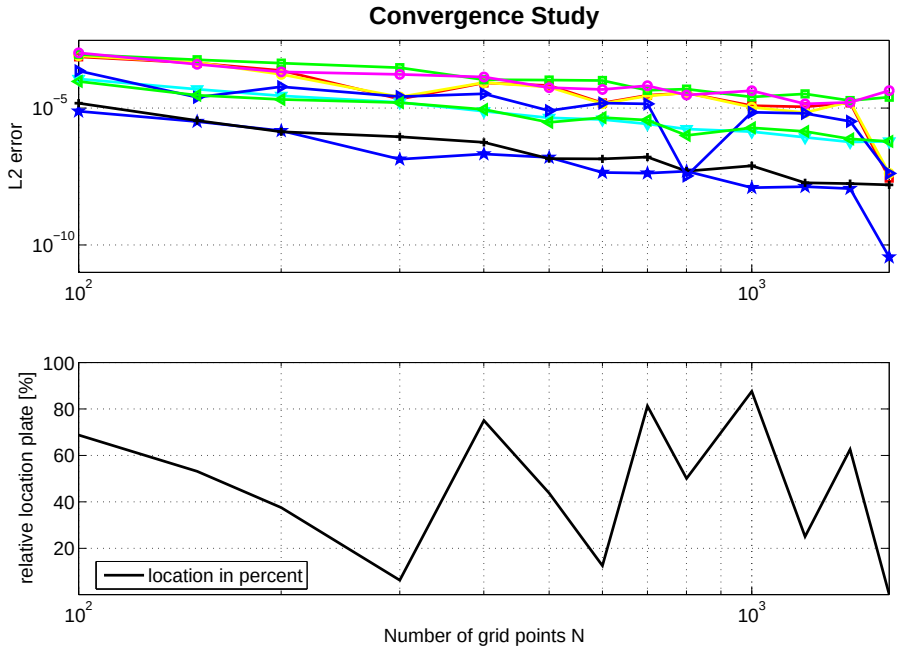


Figure 14.11: L_2 error for all the ”approximate” numerical methods, compared to the relative position of the immersed boundary in the cell.

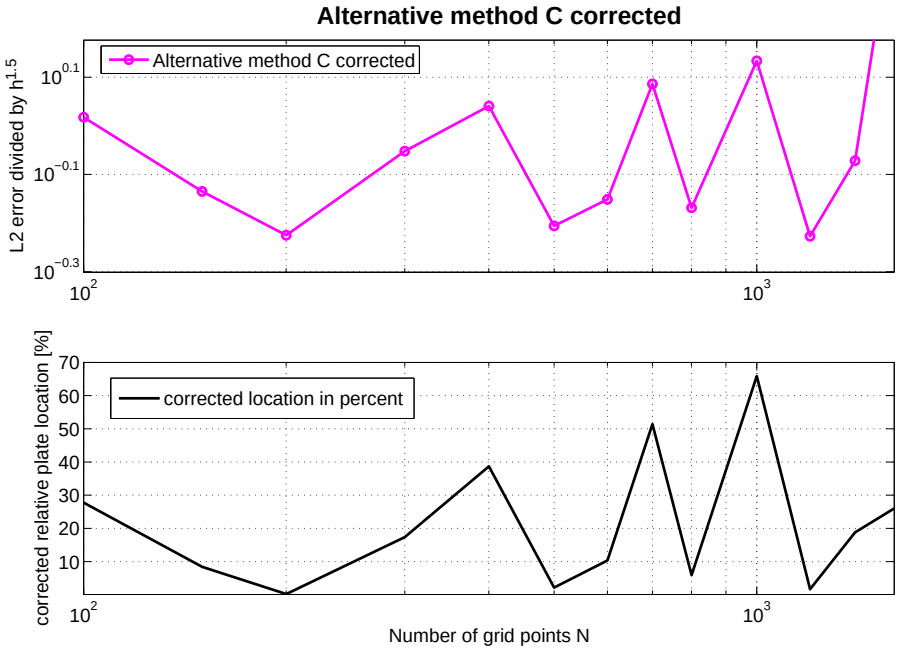


Figure 14.12: Corrected L_2 error for Alternative method C, compared to the relative position of the immersed boundary in the cell corrected for the distribution of the error.

for the Alternative method C is presented in figure 14.12. The corrected L_2 norm in this plot results when the normal Alternative method C L_2 norm is divided by $h^{1.5}$, the general dependency of the method on the grid size. As a result, the corrected L_2 error runs now more or less horizontal, as does the relative plate position. For correcting the position of the plate relative to the cell, an approximation of the error distribution over a cell for Alternative method C (see figure 14.7) is made. The approximation is essentially a linear or higher-order polynomial function. This approximate error distribution function can then be used to define the corrected relative plate position in the cell as a distance from the position where the error would be minimal.

The resulting graph 14.12 shows a clear relationship between error and relative immersed boundary movement. Any small discrepancies can be attributed to the approximation of the error distribution by a second order polynomial.

The same procedure could be repeated for the other methods as well, with similar results. But figure 14.11 can also be interpreted directly: take for instance the L_2 error for the Analytical Forcing method $d3$ at $N = 800$ and $N = 1600$. The error in these points is substantially lower than for the adjacent number of grid points. This can be explained by evaluating the error distribution for this method (figure 14.7) for the relative plate positions: $50,0512\%$ ($N = 800$) and 0.1024% ($N = 1600$). It turns out that the error distribution for these plate positions is very low indeed.

This concludes the error analysis of the Immersed Boundary Methods that were applied to the 1D Poiseuille flow problem. The next step is to pick the right method for use in a steady 2D incompressible Navier-Stokes code.

Chapter 15

Conclusions on the 1D IBMs study

In the previous chapters a number of IBMs have been studied. The comparison of the numerical approximation of the Poiseuille flow with immersed boundary to the analytical solution has revealed important properties of each method. On the basis of this analysis, a choice needs to be made on which method(s) to use in the next phase, the 2D code. The relevant properties of a method are: compatibility with the basic 2D finite volume code and its data structure (see chapter 17), accuracy and ease of implementation.

Furthermore, since the focus lies on flows of a convective nature with higher Reynolds numbers, continuous forcing approaches which use distribution functions seem less attractive, as they are not well suited to capture thin boundary layers. This means that both the Explicit boundary condition method and the Analytical forcing method will not be extended to 2D. The Analytical forcing method faces even more problems concerning extension to 2D: it remains hard to see how the forcing term (which depends only on the known and constant $\frac{\partial p}{\partial x}$ in 1D) can be determined exactly in 2D.

Both linear and quadratic Cut cell methods performed very well in this 1D study, however they are difficult to implement in the existing 2D code data structure. On top of that, these methods have already been studied extensively in the past, and they are, strictly speaking, no pure Immersed Boundary Methods, since they need to modify the simple cartesian mesh to solve the equations. For these reasons, the Cut cell method will not be implemented in the 2D code.

The Ghost cell method on the other hand is very suitable: it shows good convergence, it is accurate, not too difficult to implement in the 2D finite volume code and has been shown to work pretty well in 2 and 3 dimensions. It can also be modified quite easily if desired. The Alternative methods in general are not the best in terms of accuracy and convergence. The results are encouraging enough to develop some more alternative methods in 2D, without being limited to the methods tested in the 1D test problem.

Part III

The 2D methods

Chapter 16

Introduction to the 2D methods

After completion of the 1D phase, a few important decisions had to be made about the setup of the 2D code. First of all, there was the question whether to implement the IBMs into an existing code or to build one from scratch. This last option would have given the most freedom and control over the end result, but the programming would probably have required spending a considerable amount of time on something which was not the topic of the graduation project. Moreover, there was a code available, written by Jeroen Wackers, that satisfies the main requirements: it is a steady 2D incompressible Navier-Stokes solver. It applies artificial compressibility and time stepping to converge the unsteady Navier-Stokes equations to a steady solution (chapter 17). This means that there is still room to extend the method to unsteady Navier-Stokes by removing the artificial compressibility term and adding pressure correction steps to the normal time steps, potentially useful for future developments.

The next step was the pre-processing, documented in chapter 18. As mentioned before, the simple cartesian grids are the main strength of IBMs, so an easy-to-use, multi-purpose grid generator is an essential part of any successful Immersed Boundary code. Especially the representation of the immersed boundary and the changes to the data structure of the solver are important. One of the most difficult issues is the general nature of the shape of an immersed boundary: dealing with sharp or thin edges, indents in the surface or multiple bodies in close proximity requires extra care and extra programming.

With the basic 2D finite volume code and the cartesian grid generator ready, the implementation of the various IBMs could begin. Four of the six methods that have been programmed are discussed in chapter 19.1 (i.e. Ghost cell, adapted Ghost cell, Stairstep and adapted Stairstep method). The other two methods were unsuccessful. One of these is the Cut cell method, and although it is in theory a good technique, combining it with the existing data structure proved far too complex and time consuming to complete it. The other is a 2D version of the one-dimensional Alternative method A (chapter 10), which suffers from stability and accuracy problems directly related to the ”tuning” factor, comparable to the 1D Explicit boundary condition method (chapter 9). These last two methods are not included in this report.

To assess the quality of the written codes, a good benchmark problem is needed to test the IBMs solutions of a 2D, steady, incompressible convection-diffusion flow. Two well documented test cases are chosen: the flow over a backward facing step at $Re = 150$ [20] and the flow around a circular cylinder placed slightly off-center in a channel at $Re = 20$ [25]. The test cases are treated in chapters 20 and 21.

Finally, to display the capabilities of IBMs codes, the flow around a multi-element airfoil at $Re = 1000$ is computed on an H-type cartesian grid, see chapter 22. The ease of automated cartesian grid generation is illustrated in comparison to the structured body-fitted grid approach used by [23], where 27 different regions make up the whole grid. Chapter 23 presents the conclusions for the 2D code part.

Chapter 17

The 2D finite volume code

This chapter briefly discusses the 2D finite volume code as written by Wackers [27], [28]. The goal of this chapter is to clarify the global structure and solution method of the code without getting stuck in the details. As mentioned before, the code is a finite volume-based, 2D incompressible Navier-Stokes solver which time-marches towards a steady solution, using a fictitious $\frac{\partial p}{\partial t}$ term. The grids are considered to be cartesian and only the domain boundaries are treated (immersed boundaries are dealt with in the pre-processor, next chapter). The program is written in Fortran 95.

This compact description gives a first impression. In the remainder of this chapter, two important aspects of the finite volume code are explained in detail: the data structure and the solution method.

17.1 The data structure

17.1.1 The grid

The grid is structured such that the number of (rectangular) cells in X and Y direction remains constant throughout the physical domain. However, the cell size need not be constant, although strong variations in cell size can affect the accuracy. This essentially allows for the construction of so-called H-type grids (this will be used to our advantage later on). The grid is positioned into the physical domain such that one band of cells falls just outside the boundaries of the domain. These cells wrapped around the domain are called boundary cells, they are used to store the boundary conditions defined on the domain boundaries and will be treated differently from cells in the interior of the domain. When a grid is generated, it is stored in an array containing all the X and Y coordinates of the lower left corner of each cell. This sequence of cell coordinates starts in the upper left most cell of the grid, runs right until the end of the row and continues on the leftmost cell of the row below it. The cell numbering is done in the same way. Figure 17.1 summarizes some of the main grid-related definitions.

17.1.2 The neighbors

In a later stage, fluxes need to be computed over all four cell faces. To do this, it must be known which cell borders another. This would not have posed any problems in an i, j structured rectangular grid. However, this data structure is inherited from the old code, which used semi-structured grids. The numerous adjustments that would have to be made in the code when converting the grid to an i, j structure made changing the old system undesirable, which used the cell numbers to define ”neighbors”. For every cell, the cell numbers of the cells above and below it and to its right and left are stored in an array. If a cell does not have a cell on one (or two) sides because it is a boundary cell, a zero is stored for that neighbor.

The information gathered so far enables all the geometric calculations to be done. The geometry of a cell is determined by defining its corners as the stored lower left corners of its right neighbor and the cell above

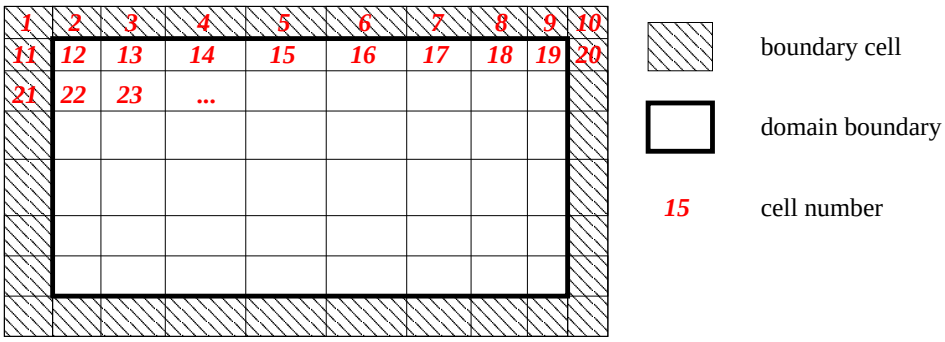


Figure 17.1: Basic properties of the cartesian grid.

that one, its upper neighbor and finally the cell itself. Then the length of each cell face is calculated, as well as its area.

17.1.3 The boundary flag

When a cell is a boundary cell, it receives a boundary flag. The flag is -3 for an inflow boundary, -4 for an outflow boundary, -5 for a symmetry boundary and -6 for a no-slip boundary (solid wall). This boundary flag tells the solver how to calculate the flux over the cell face it shares with a neighbor in the interior of the domain. The state variables stored in the boundary cells are never updated, since they are pre-set boundary conditions. In general, u and v are defined on the inflow boundary, p on the outflow boundary while the remaining boundaries are set to no-slip or symmetry.

17.1.4 The initial solution

Besides boundary conditions, an initial solution needs to be given to start the time-stepping procedure. For this purpose a value for each of the three state variables (velocities u and v and pressure p) in each cell is stored in another array which will later contain the converged solution. The state variables in the boundary cells represent the boundary conditions, since they are never updated as opposed to the normal cells.

The data structure as it is contains now enough information to compute basic phenomena like channel flows. The inclusion of an immersed boundary demands additional information to be stored for use in the solver, and this expands the existing data structure. More details can be found in chapter 18.

17.2 The solution method

The first step in the solution procedure is the derivation of the governing equations. These equations are then discretized on the finite volume grid. The convective and diffusive parts are discretized, allowing the fluxes (and their sum) to be calculated, which in turn is used to update the state variables through a time step. This procedure is repeated for all cells (called a sweep). The sweep procedure itself is then reiterated until a predefined convergence criterion is met. All these stages are described below.

17.2.1 2D flow equations

The derivation of the governing equations starts again with the unsteady, incompressible 2D Navier-Stokes equations without body forces, see [2]:

$$\begin{aligned}
\frac{\partial(u)}{\partial t} + \frac{\partial(p+u^2)}{\partial x} + \frac{\partial(uv)}{\partial y} &= \frac{\partial(\mu u_x)}{\partial x} + \frac{\partial(\mu u_y)}{\partial y}, \\
\frac{\partial(v)}{\partial t} + \frac{\partial(uv)}{\partial x} + \frac{\partial(p+v^2)}{\partial y} &= \frac{\partial(\mu v_x)}{\partial x} + \frac{\partial(\mu v_y)}{\partial y}, \\
\frac{\partial(u)}{\partial x} + \frac{\partial(v)}{\partial y} &= 0.
\end{aligned} \tag{17.1}$$

Note that both momentum equations contain a time derivative of u or v, but the continuity equation does not have a time derivative for the pressure. This means that a time stepping procedure based on these equations will be unable to update the pressure. One way to tackle this problem is to add an artificial $\frac{\partial p}{\partial t}$ term to the last equation and time-march until convergence, since at this point the time derivatives are zero anyway. Hence, the solution found is also a solution of the steady version of system (17.1). This trick is called *artificial compressibility* and is described in [5].

System (17.1) then becomes:

$$\begin{aligned}
\frac{\partial(u)}{\partial t} + \frac{\partial(p+u^2)}{\partial x} + \frac{\partial(uv)}{\partial y} &= \frac{\partial(\mu u_x)}{\partial x} + \frac{\partial(\mu u_y)}{\partial y}, \\
\frac{\partial(v)}{\partial t} + \frac{\partial(uv)}{\partial x} + \frac{\partial(p+v^2)}{\partial y} &= \frac{\partial(\mu v_x)}{\partial x} + \frac{\partial(\mu v_y)}{\partial y}, \\
\frac{1}{C^2}\frac{\partial(p)}{\partial t} + \frac{\partial(u)}{\partial x} + \frac{\partial(v)}{\partial y} &= 0.
\end{aligned} \tag{17.2}$$

where C is a chosen constant and the subscripts stand for derivatives. To obtain a more compact form, the following vectors are defined:

$$\mathbf{q} = \begin{pmatrix} u \\ v \\ p \end{pmatrix} \quad \mathbf{q}' = \begin{pmatrix} u \\ v \\ \frac{1}{C^2}p \end{pmatrix} \quad \mathbf{f} = \mathbf{f}(\mathbf{q}) = \begin{pmatrix} p+u^2-\mu u_x \\ uv - \mu v_x \\ u \end{pmatrix} \quad \mathbf{g} = \mathbf{g}(\mathbf{q}) = \begin{pmatrix} uv - \mu u_y \\ p+v^2-\mu v_y \\ v \end{pmatrix}.$$

(17.2) now transforms into:

$$\mathbf{q}'_t + \mathbf{f}(\mathbf{q})_x + \mathbf{g}(\mathbf{q})_y = \mathbf{0}. \tag{17.3}$$

Integration over one cell Ω_k in combination with the divergence theorem [1] then results in:

$$\int\int_{\Omega_k} \mathbf{q}'_t \, dxdy + \oint_{\partial\Omega_k} (\mathbf{f}n_x + \mathbf{g}n_y) \, ds = \mathbf{0}. \tag{17.4}$$

Until this point everything is still exact, but now the system of equations (17.4) needs to be discretized. In a first-order finite volume approach the values of the state variables are assumed to be constant over a cell. Since the cells in the cartesian grid are rectangular and aligned with X and Y, the boundary integral of the flux term can be written as a sum over the four cell faces of cell Ω_k, with $\overline{\mathbf{f}}$ and $\overline{\mathbf{g}}$ denoting the approximate fluxes:

$$\sum_{m=1}^{4} \left(\overline{\mathbf{f}}_{k,m} n_{x_{k,m}} + \overline{\mathbf{g}}_{k,m} n_{y_{k,m}} \right) l_{k,m}. \tag{17.5}$$

n_x and n_y are the x and y components of the outward unit normal vector, and l the length of a cell face. The flux term will now be split in a convective and a diffusive part to simplify the discretization.

17.2.2 The convective flux

The convective part of the flux vector is calculated using an approximate Riemann solver, a kind of linearized Osher scheme. To find the state at the interface between two cells (and thus the convective flux), the states on both sides of the cell face are viewed as the right and left initial states in a 1D shock tube problem from gas dynamics. After this initial instant, contact discontinuities, shocks and expansion waves start traveling through the fluid, resulting in a new state at the interface. This strategy to determine the convective fluxes is made possible by adding the artificial compressibility term to the governing equations, which makes the system hyperbolic.

To establish the relation between the state variables on the cell face and the left and right states a characteristic analysis needs to be done, in order to find the characteristic wave speeds and Riemann invariants. This however falls outside the scope of this report and is moreover very well documented in e.g. [28], see also [6], [7], [8] and [27].

17.2.3 The diffusive flux

Compared to the convective flux discretization, the diffusive fluxes are very simple. The only terms that need to be computed are $-\mu\frac{\partial u}{\partial n}$ and $-\mu\frac{\partial v}{\partial n}$. These can be approximated with second-order accurate central differences. And, because the grid is aligned with X and Y, $\frac{\partial}{\partial n} = \frac{\partial}{\partial x}$ or $\frac{\partial}{\partial y}$ for each cell face.

17.2.4 Time stepping

Now that the flux term is solved, equation (17.4) can be discretized in time such that the new state in cell Ω_k can be determined explicitly from the old state and the sum of the fluxes:

$$\mathbf{q}'^{t+1} = \mathbf{q}'^{t} - \frac{\Delta t}{A_k}\sum_{m=1}^{4}\left(\,\overline{\mathbf{f}}_{k,m}n_{x_{k,m}} + \,\overline{\mathbf{g}}_{k,m}n_{y_{k,m}}\right) l_{k,m}, \tag{17.6}$$

where Δt is the time step and A_k the cell area. This is known as the first-order forward Euler discretization in time. With this expression, the state variables in a cell can be updated until the new and old states differ less than a pre-set value.

Chapter 18

The pre-processor

This chapter describes the main pre-processing procedures: the generation of the grid and the collection of the immersed boundary data. The immersed boundary is assumed to be a closed curve, given as an array of coordinates of points. There is a note at the end of this chapter on the procedure for multiple immersed bodies.

The pre-processor is a separate program that creates a grid and immersed boundary data complete with initial and boundary conditions to the user's specifications. The data are stored in two files: one file which lists the initial solution in every cell, the second file contains the grid data (see chapter 17) and additional immersed boundary information. These files can be checked for errors (e.g. by plotting the grid, see figure 18.1) before they are read by the Navier-Stokes code. This is not trivial, since the pre-processor can have difficulties when dealing with particular body shapes, see subsection 18.3.2. The grid files can be created quickly and they are re-usable (to give an example: a 200.000 cell H-type grid will take approximately 3 minutes on a 64-bit AMD Athlon processor to be generated, from specifying grid properties to saved grid files). This is true for almost any given geometry. However, very large grids (millions of cells) mean a lot of time is spent saving the grid to the files, and these files become rather large too. In this case, it might be better to couple the pre-processor directly to the Navier-Stokes code, so no file needs to be created.

18.1 The cartesian grid

Before actually creating the grid, all the final output parameters are initialized. The parameters that were discussed in the previous chapter (*boundary flag* and *neighbors*) as well as the new immersed boundary parameters *immersed boundary flag*, *ghost flag* and *ib grid points* are set to a standard value, such that not all parameters for every cell need to be changed later on. This standard value is 0, except for the *ib grid points*, they are set to -99.0 or any other value well outside the physical domain.

18.1.1 Uniform grid

Creating a uniform grid is of course rather straightforward: once the user has set the width and height of the domain and how many cells he/she would like in X and Y direction, the actual determination of the node positions is trivial. The X and Y coordinates of the lower left corner of each cell are then stored in a 2-column array, following the cell numbering as explained in chapter 17. At the same time, the cell numbers of the neighboring cells are stored in *neighbors* and the boundary type in *boundary flag*. Figure 18.1 shows the coarse uniform grid for the backward facing step problem.

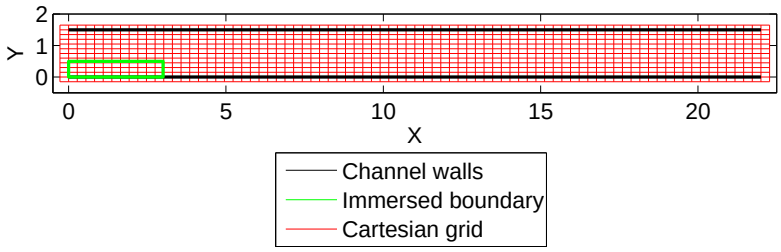


Figure 18.1: Uniform cartesian grid for the backward facing step problem.

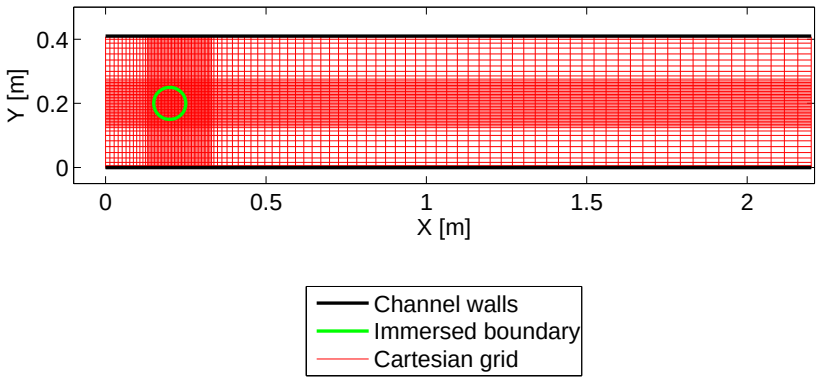


Figure 18.2: H - type cartesian grid around a circular cylinder.

18.1.2 H - type grid

An H - type grid is somewhat more complex, since the grid is split into different regions. The basic idea is to define a rectangular area around the body and mesh it with a fine uniform grid. This fine grid helps to capture the shape of the body and to increase the accuracy when dealing with thin boundary layers. The rest of the domain is meshed using e.g. parabolic, cosine or exponential functions. Those functions ensure a smooth transition to the uniform grid region with limited increase in the total number of cells. In other words: an H - type grid can guarantee good local grid resolution for a smaller total number of cells than a uniform grid. Since these grids have no ”hanging nodes”, the data structure does not need to be changed and the same parameters as above are stored in the grid file. An example of this mesh type can be found in figure 18.2.

18.2 Initial and boundary conditions

Additional to the declaration of the boundary type with the *boundary flag* parameter, a boundary condition for velocities u and v needs to be declared at the inflow boundary and one for the pressure p and velocity v at the outflow. The inflow boundary condition is often a uniform freestream or parabolic velocity profile, while the outflow boundary condition is an arbitrary constant pressure and $v = 0$.

The initial solution is defined in all cells, except for those where a boundary condition is prescribed. While the state in the the normal cells is updated through the time-stepping procedure, the state in the boundary cells is not (to avoid the boundary conditions being overwritten). Care is taken to ensure a smooth transition between initial solution and boundary conditions. Both initial solution and boundary conditions are saved in a file, which forms part of the input for the Navier-Stokes solver.

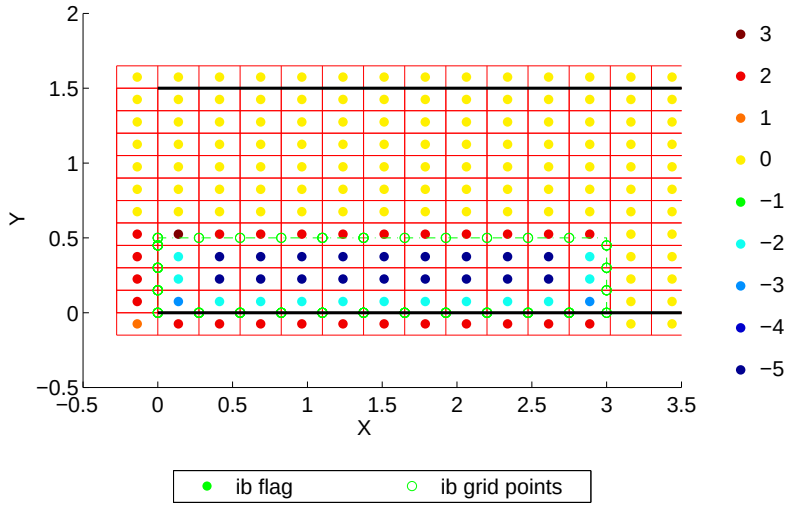


Figure 18.3: *ib flag* for the backward facing step grid.

18.3 Immersed boundary data

The problem of data structure and information on immersed boundary position necessary to solve the flow problem is considered first. The goal is to translate a given boundary shape and position into cell-specific data such that it can be processed in IBM fashion. The data should answer the following questions: *Is there a boundary cutting through this cell? If so, where? Is this cell inside or outside the body? Is this cell a so-called ghost cell?*

A number of parameters is defined to provide answers to these questions. First of all, the intersections of the given immersed boundary with the cell faces of each cell need to be stored. The X and Y coordinates of these intersections are stored in *ib grid points*, where the sequence is very important. The sequence allows the code to reconstruct the immersed boundary, by drawing lines between two consecutive *ib grid points*.

Next is the *immersed boundary flag* or *ib flag*. This parameter tells whether a cell center lies inside (negative) or outside the body (zero or positive). At the same time, it counts the number of times the given immersed boundary intersects with the cell boundary. This is limited to a maximum of 4 intersections, more than that is very unlikely to happen under adequate grid resolution. Examples: *ib flag* = 0 means that the cell lies outside the body without being intersected by an immersed boundary, *ib flag* = -3 denotes a cell inside the body with 3 immersed boundary intersection points, while a cell with *ib flag* = -5 lies inside the body, clear of the immersed boundary. In figure 18.3, the *ib flag* parameters are plotted as colored dots for the backward facing step grid.

Finally, the *ghost flag* parameter is set to 1 if the cell is a ghost cell, 0 if the cell is not. A cell is a ghost cell when its center lies within the body and it has at least one neighbor whose cell center lies outside the body.

18.3.1 Determination of the *ib grid points*

In general, a 2D body is given as a sequence of coordinates that describes the shape of the body. To make sure it is a closed form, the last and first point should be the same. The pre-processor then takes the first two points and draws a straight line between them. The coordinates of the intersection points between

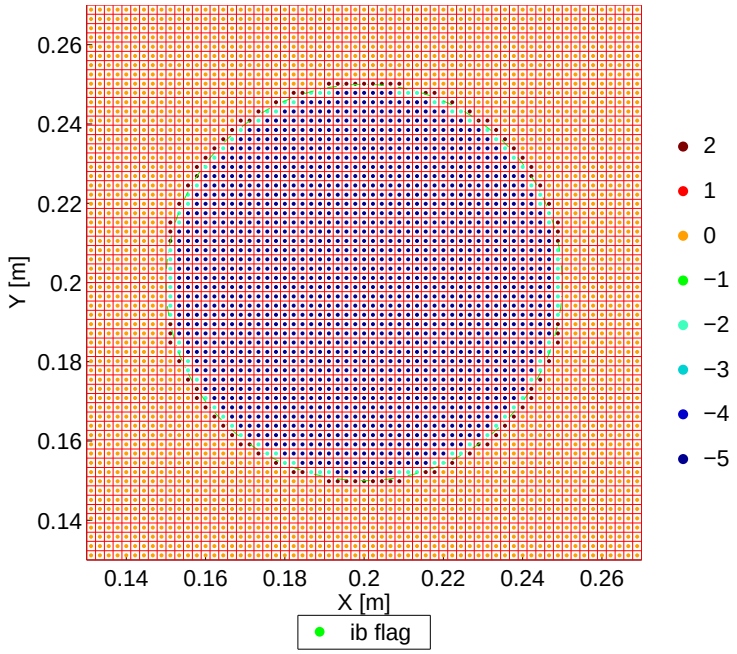


Figure 18.4: *ib flag* for the circular cylinder grid.

this line element and any horizontal or vertical grid line are stored. Intersections that were stored twice (e.g. because they happen to fall on a grid node) are removed, and the remaining points are rearranged according to their distance to the first original boundary point. This process is repeated until the whole given boundary is translated to a sequence of *ib grid points*. For each cell, the number of intersections is then counted to yield the absolute value of *ib flag*. The sign of *ib flag* is determined below. The *ib grid points* are plotted as green circles for the backward facing step (figure 18.3).

18.3.2 Sign of *ib flag*

To know whether a cell lies inside the body or not, one cell is chosen which lies certainly inside the body. Then a search routine checks the *ib flag* of the cell's neighbors: if it is also 0, it must be inside the body too, and its *ib flag* is changed to -5. The search routine continues to do this with the newly found cells inside the body until it cannot find any more cells whose *ib flag* is 0.

At this point, all the cell centres inside the body are found, except those cells that are cut by the immersed boundary. Therefore all the cells whose *ib flag* value is 2, 3 or 4 are checked in a new search routine. If they have a neighbor that lies inside the body (negative *ib flag*), a line element is constructed between the cell centres of the cell with *ib flag* ¿ 1 and its neighbor inside the body. If that line element is intersected by an immersed boundary segment reconstructed from the *ib grid points*, the *ib flag* remains unchanged. If not, *ib flag* becomes negative. This is done a few times to ensure all cell centres inside the body are found. The result can be seen in figures 18.3 and 18.4.

Although the result seems satisfactory, it is not a fool proof procedure. One particular shape of the immersed boundary gives problems with the determination of the sign of *ib flag*. Thin, wedge-like shapes (e.g. airfoil trailing edges) can make it impossible to draw a line element between neighboring cell centres inside the body that is not intersected by an immersed boundary segment. As a result, a cell center which lies inside the body is not recognized as such and its *ib flag* remains positive (see figure 18.5, left). Another

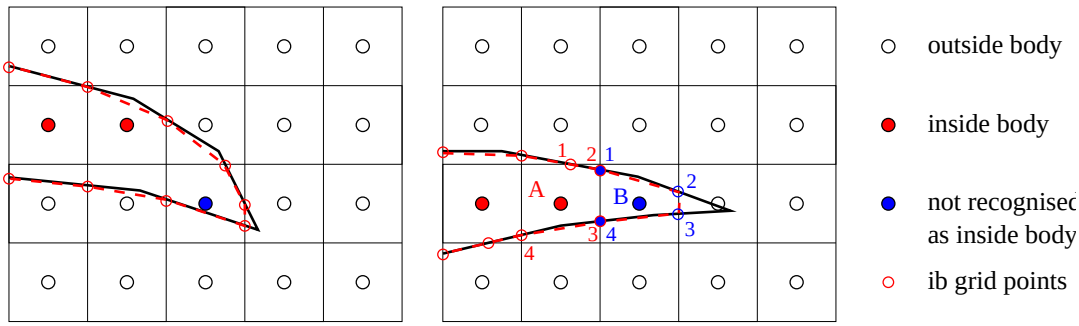


Figure 18.5: Sketches of the two problems with the body search routine.

problem is that cells near the trailing edge of an airfoil often contain four *ib grid points*: their cell faces are cut four times by the immersed boundary. This happens quite regularly at the very tip of the trailing edge (cell B), but it is not uncommon to be the case in neighboring cells at the same time (cell A), see figure 18.5, right. The immersed boundary needs to be closed at the trailing edge to prevent ”leaking” of the negative *ib flags* (see procedure above). This is accomplished by constructing immersed boundary segments between *ib grid points* 1 and 2, 2 and 3 and 3 and 4 in cell B. However, since the datastructure needs to be generally applicable, the same is true for the neighboring cell (A) with *ib flag* 4. As a result, the line element drawn between the two cell centres is cut by the immersed boundary segment between *ib grid point* 2 and 3 from cell A and the *ib flag* of cell B stays positive, keeping the cell center ”outside” the body.

18.3.3 The *ghost flag*

The last immersed boundary data parameter is the *ghost flag*. The function of this parameter is to identify ghost cells, cells whose center lies inside the body which have at least one neighbor outside the body. These cells have a special function in Ghost cell methods (see chapters 5, 11 and 19). The *ghost flag* is derived by checking the *ib flag* of the neighbors of all cells inside the body: when a cell with a negative *ib flag* has at least one neighbor with a positive (or zero) *ib flag*, its *ghost flag* is changed to 1. Because it depends fully on the *ib flag* parameter, any mistake there will translate in a wrong *ghost flag*. Examples are given for the backward facing step and circular cylinder in figures 18.6 and 18.7.

18.3.4 Multiple bodies

When the object under consideration consists of more than one element, the whole immersed boundary data loop can be rerun for each element. However, more than one immersed boundary cutting through the same cell should be avoided by refining the mesh.

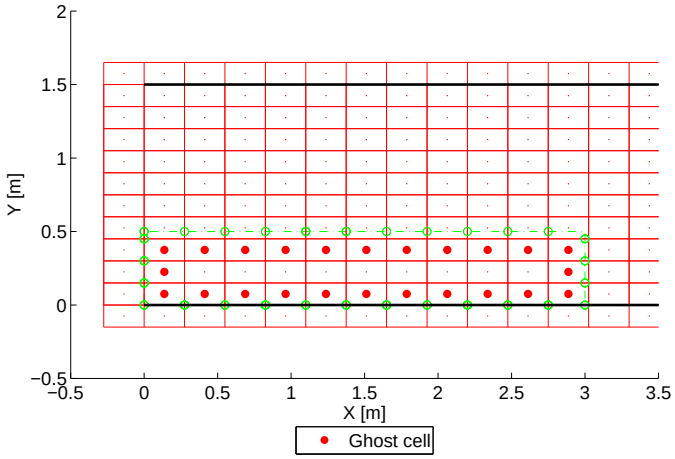


Figure 18.6: *ghost flag* for the backward facing step.

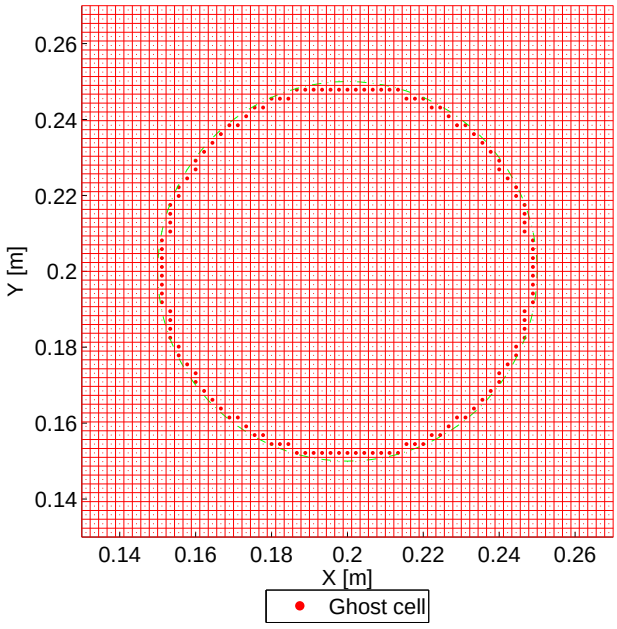


Figure 18.7: *ghost flag* for the circular cylinder.

Chapter 19

The 2D Immersed Boundary Methods

In total, four IBMs were implemented successfully in the 2D code. The Ghost cell method, adapted Ghost cell method, Stair-step method and adapted Stair-step method are presented in this chapter. The main purpose of this chapter is to explain the principal idea behind a technique and to evaluate its most important properties, rather than drowning the reader in programming details. For the same reason, no print-outs of the code itself are added in appendix.

19.1 The Ghost cell method

As discussed in chapters 5 and 11, the basic idea behind the Ghost cell method is to divide the grid cells in one of three categories: the ghost cells, the cells inside the body that are not ghost cells and the rest, the fluid cells. The cells inside the body are ignored from now on, their state is of no interest so no computation time is wasted on them. The values of the state variables in the ghost cells result from extrapolation. The boundary condition on the immersed boundary and the fluid cells in the vicinity of the ghost cell provide the input for the extrapolation stencil. The consequence of implementing the boundary conditions through the extrapolated ghost cell states is that the Ghost cell method cannot guarantee exact conservation of mass and momentum. The fluid cells are treated as usual: fluxes based on a cell's own state and the state of its neighbors at time t are used to calculate the new state at time $t + \Delta t$.

The first thing to do is to implement an IF loop in the sweep routine (i.e. the procedure where the sum of the fluxes is calculated for all cells in the grid). This is done to distinguish between the three cell categories (as discussed above) and to choose the corresponding actions that need to be taken: do nothing (cells inside the body), extrapolate (ghost cells) or follow the normal flux calculation (all the other cells). The only task remaining now is the determination of the state variables in the ghost cells through extrapolation. Since the accuracy of quadratic extrapolation would not be visible in a first order finite volume method, linear extrapolation is the preferred choice.

19.1.1 The linear extrapolation scheme

Once the sweep routine arrives at a ghost cell, the neighbors of this cell are checked for position: when the neighbor lies outside the body (zero or positive *ib flag*), the immersed boundary segment that separates them needs to be located. To do this, the intersection point(s) of the immersed boundary segment(s) and the line connecting the cell centres of the ghost cell and its neighbor is (are) calculated, the one that falls in between the cell centres is on the immersed boundary segment needed (figure 19.1). It is very important to perform this check, since each cell can contain up to three different immersed boundary segments and choosing the wrong one results in large errors.

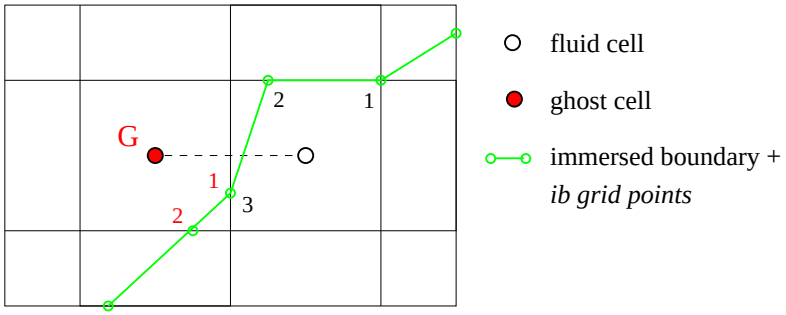


Figure 19.1: Situation sketch for the linear extrapolation scheme.

The two *ib grid points* corresponding to the immersed boundary segment just found forms the basis of the linear extrapolation scheme, together with the center of the cell neighboring the ghost cell. The positions of these three points and their respective values for u and v can be used to construct fictitious planes in the X, Y, U space:

$$\begin{aligned} u(x,y) &= C_{u_1} x + C_{u_2} y + C_{u_3} \\ v(x,y) &= C_{v_1} x + C_{v_2} y + C_{v_3}. \end{aligned} \tag{19.1}$$

Constants $C_{u_1}, C_{u_2}, etc.$ are found by evaluating (19.1) in the previously mentioned points. The velocity components in the ghost cell are then determined by substituting the coordinates of the ghost cell center in the mathematical expressions for the planes.

The only remaining state variable in the ghost cell to be dealt with is the pressure. There is no boundary condition on the immersed boundary available for the pressure, so another solution has to be found. Boundary layer theory says that the pressure in a boundary layer is approximately constant perpendicular to the wall. This approximation is only valid for boundaries with limited curvature. Under the assumption that $\frac{\partial p}{\partial n} = 0$ holds, we take the pressure in the ghost cell equal to that in its neighbor. This yields an approximation with a maximum error of order h. Furthermore, for a good solution of the Navier-Stokes equations adequate grid resolution in the boundary layer is a prerequisite. This means that h near the boundary is small, thereby limiting the error caused by the numerical boundary condition for the pressure.

19.1.2 Multiple extrapolations

The procedure described above works fine when every ghost cell has only one neighbor outside the body. However, often there is more than one (example in figure 19.3), which can cause trouble. One way to deal with multiple interpolations is to create a weighted average of the extrapolated velocities and pressures based on distances between cell centres. In smooth flow regions, this approach may work well, but in ghost cells near the trailing edge of an airfoil or in the corner of a backward facing step, the results are quite disturbing. The problem is of course that the state in the ghost cell is composed of two or more very different flow regimes, like the flow above and under an airfoil or the smooth flow over the step and the recirculation region behind it (see figure 19.2).

In fact, the situation is so bad for the backward facing step test case, that it was decided not to use this method at all, but to try and fix it before continuing the tests. The resulting adapted Ghost cell method is discussed in the next section.

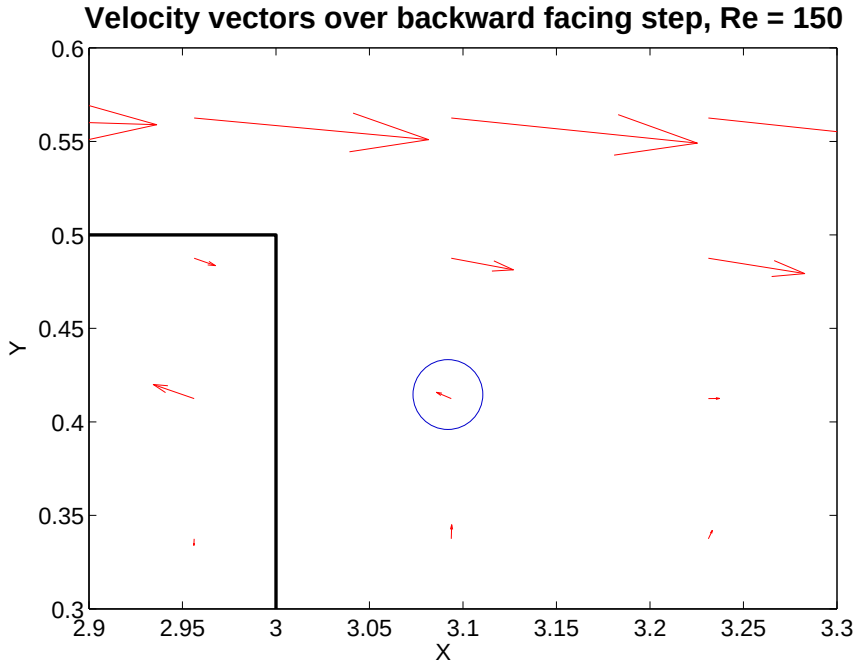


Figure 19.2: Numerical solution of Ghost cell method for the backward facing step problem (see chapter 20). Velocity vector plot near corner of step. Note the odd vector to the right of step edge.

19.2 The adapted Ghost cell method

The normal Ghost cell method encounters problems when more than one ghost cell neighbor lies outside the body. This means that each state variable in the ghost cell is composed of different extrapolations. However, the state of the ghost cell itself is not relevant for the final solution (since it is a fictitious state inside the immersed boundary), but its value is used to determine the fluxes on the cell faces the ghost cell shares with its ”fluid” neighbors. The fluxes are important, since they have a direct influence on the states of fluid cells close to the immersed boundary. Bearing these arguments in mind, it makes sense to come up with a method that computes the fluxes between ghost and fluid cell on the basis of just one extrapolation: the extrapolation from the fluid cell under consideration.

As opposed to the normal Ghost cell method, all cells inside the body (including ghost cells) are left out of the computation. For every fluid cell, the flux calculation is done as explained in chapter 17, except when a cell face between the fluid and a ghost cell is concerned. In this case, the procedure for finding the right immersed boundary segment and constructing the linear extrapolation scheme from section 19.1 is repeated. This is done independently for all ”fluid” neighbors of the ghost cell and as a result, the state of one ghost cell can be different for the flux calculations on each of its cell faces. Another difference is that the extrapolation is not evaluated in the ghost cell center, but in the middle of the cell face (figure 19.3). This state and the state in the fluid cell center are then fed into the approximate Riemann solver as right and left initial state. The flux over the cell face is determined using the resulting values for u, v and p.

The resulting scheme is much better suited to handle abrupt changes in the flow, as found in e.g. the backward facing step problem. Figure 19.4 shows the velocity vector field in the vicinity of the step, the difference with the normal Ghost cell method (figure 19.2) is very clear. The odd velocity vector near the step corner has disappeared and the flow looks smooth and normal. For this reason, the adapted and not the normal Ghost cell method will be used in the upcoming test cases, chapters 20 and 21.

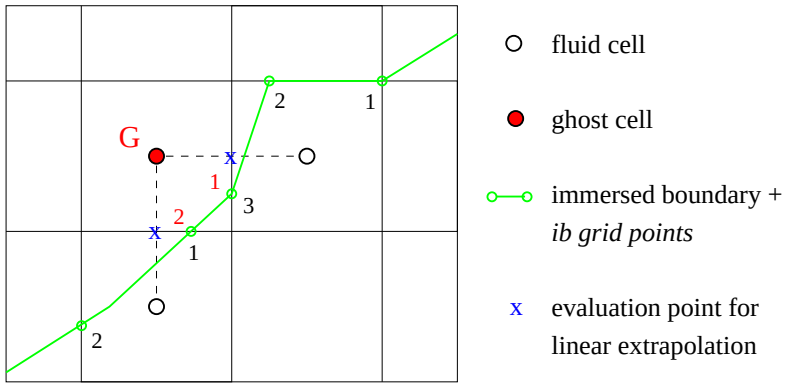


Figure 19.3: Situation sketch for the linear extrapolation scheme, adapted Ghost cell method.

19.3 The Stair-step method

The Stair-step method is another way to deal with immersed boundaries on cartesian grids, although, strictly speaking, it cannot be classified as a proper Immersed Boundary method. Instead of finding a way to impose boundary conditions through extrapolation schemes, the stair-step method approximates the shape of the boundary such that it falls onto the cartesian grid lines. Since the boundary condition $u = v = 0$ is now known on the cell faces, it becomes much easier to implement. In fact, the fluxes over the cell faces containing a boundary can be solved through the same routine used to find the flux over a cell face on a no-slip domain boundary. This is done by considering the half Riemann problem, where the position of the contact discontinuity is fixed at the cell face.

Approximating the boundary shape is done by redrawing the immersed boundary around the ghost cells. The result is a so-called stair-step, see figure 19.5. This method is not new, in fact it has been studied extensively in e.g. [29]. And because of the rather crude approximation of the immersed boundary shape, it is not expected to be very accurate either. However, comparing this method to the others will show whether putting effort in more accurate immersed boundary treatments in combination with a first-order accurate finite volume method pays off or not. Moreover, the Stair-step method will form the basis for the next Immersed Boundary Method: the adapted Stair-step method.

19.4 The adapted Stair-step method

The normal Stair-step method from the previous section imposes boundary conditions $u = 0$ and $v = 0$ on both convective and diffusive fluxes when treating cell faces on the stair-step boundary. This is necessary for the diffusive fluxes, however, the boundary condition for the convective fluxes could be limited to the specification of the flow direction $\frac{u}{v}$. The general idea can be compared to boundary conditions in potential and Euler flow: the only boundary condition set on a boundary is the requirement that the velocity component normal to the boundary has to be zero.

In practice, the flux calculation on a ghost cell - fluid cell interface is done by decomposing the velocity in the fluid cell along the immersed boundary segment axis system (see figure 19.6). Just as in the treatment of a symmetry domain boundary, the normal velocity vector is set to zero. The remaining tangential velocity is then decomposed again in the axis system of the cell face and the resulting velocities are used to construct the flux vector. The pressure on the cell face is taken equal to the fluid cell pressure, as usual.

The present scheme is not exactly mass-conserving, however it may yield smoother solutions near the boundary than the normal Stair-step approach.

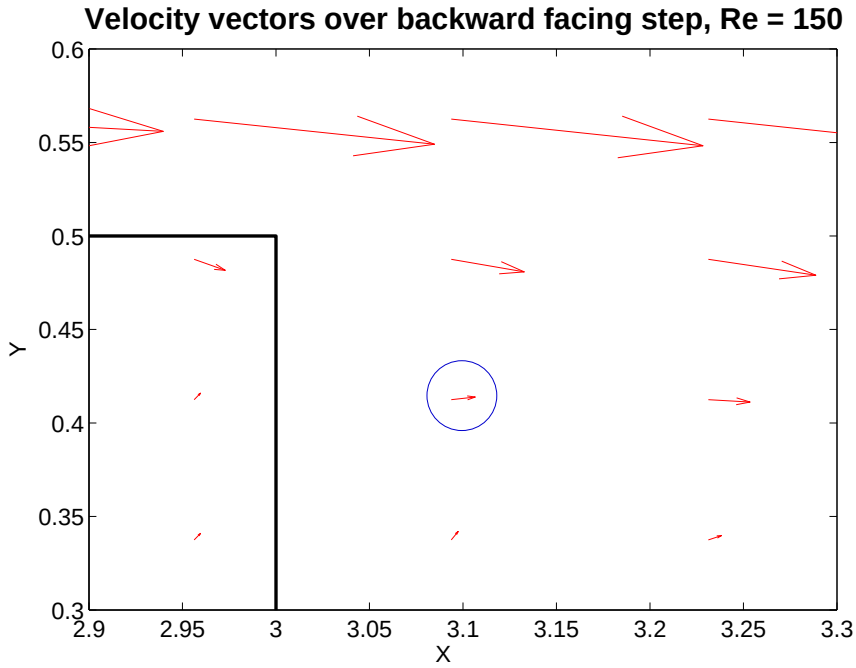


Figure 19.4: Numerical solution of the adapted Ghost cell method for the backward facing step problem (see chapter 20). Velocity vector plot near corner of step.

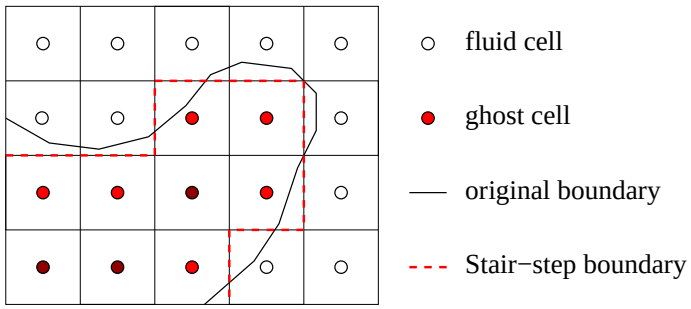


Figure 19.5: The stair-step boundary.

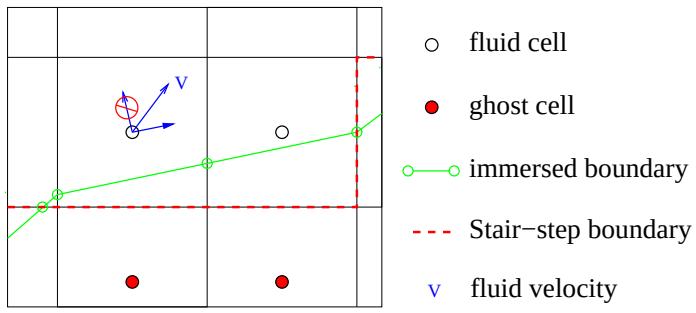


Figure 19.6: The stair-step boundary, with decomposed velocity vector in fluid cell.

Chapter 20

Test case I: the backward facing step

20.1 Introduction

The backward facing step problem was first studied in the 1984 GAMM workshop on the *Analysis of Laminar Flow over a Backward Facing Step* (see [20]). The goal of the organizers was to define a test problem that would be suitable as a benchmark for a wide range of 2D Navier-Stokes CFD codes. Measurements from wind tunnel experiments (actual medium was MARCOL 2 Oil) provided data to which the numerical results could be compared.

The backward facing step is sketched in figure 20.1. It is a 2D channel flow with inflow h and outflow H, measuring 23 in length. The step is 3 long and 0.5 high. The walls are no-slip boundaries, at the outflow boundary the pressure is kept constant and at the inflow the fully developed parabolic velocity profile for laminar flow is prescribed. In total, six test cases were given: two $h - H$ combinations at three Reynolds numbers ($50, 150$ and 500), where the Reynolds number is defined as

$$Re = \frac{U_{max}(H - h)}{\nu}.$$

The $Re = 500$ case is an unsteady flow and as such not suitable for the IBMs codes developed in this project. The remaining four test problems do not differ substantially, therefore only the $Re = 150$, $h = 1$, $H = 1.5$ setup is chosen.

A number of items is important when assessing the quality of a numerical solution to the backward facing step problem. These items can be compared to both wind tunnel experiment data and the numerical solutions provided by participants of the workshop. Of primary interest are:

1. streamline plots on the whole domain and in the recirculation zone,
2. pressure level plots on the whole domain and in the recirculation zone,
3. length of the recirculation zone,
4. minimum of the streamfunction,
5. maximum of the streamfunction at the outflow boundary.

The streamline and pressure level plots give a very good qualitative picture of the numerical flow solution. However, since the backward facing step problem was computed with the three available methods on 16 different grids (see below) and because, especially on the finer grids, the plots for the different methods resemble each other very well, it was decided not to include all the plots in this report. Instead, a few figures are shown of the numerical solutions on the finest grids, where all three methods are represented.

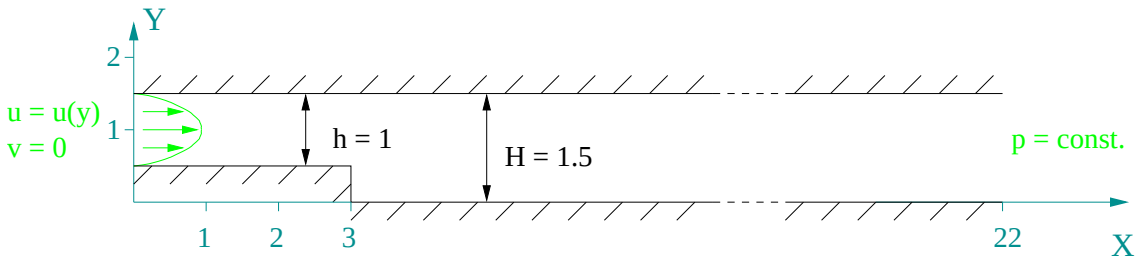


Figure 20.1: Sketch of the backward facing step.

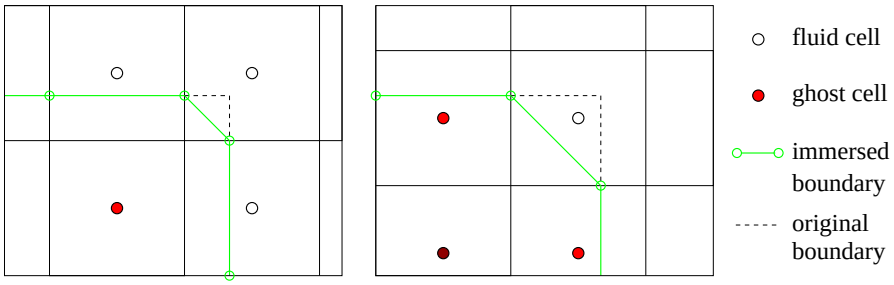


Figure 20.2: Sketch of the relative position of the immersed boundary in a cell for the grids from grid studies 3 (left) and 4 (right).

For the last three items, grid studies are done to get an idea about their converged values. To examine whether the relationship between relative position of an immersed boundary in a cell and the error as found in 1D (chapter 14) holds in 2D as well, four distinctive grid studies are performed. In study 1, the cell sizes are chosen such that the immersed boundary coincides with the grid lines, to mimic a body-fitted grid. Study 2 is a normal grid study, where the boundary is truly ”immersed”. In studies 3 and 4, the grids are designed to keep the boundary at a specific relative position within a cell, as is sketched in figure 20.2. Each study is performed on four grids for three methods: the adapted Ghost cell, Stair-step and adapted Stair-step method. The grid sizes per study are shown in table 20.1.

A plot of one of the coarse grids can be found in figure 18.1. However, the detail of the step corner in figure 20.2 shows much more clearly how the step geometry is approximated when dealing with one of the grids from grid study 2, 3 or 4. The tilted immersed boundary segment has a substantial effect when the flow is solved with the adapted Ghost cell and adapted Stair-step methods, compared to the flow around the real step geometry.

Study 1	Study 2	Study 3	Study 4
88×12	80×10	90×10	86×11
176×24	160×20	178×22	174×23
352×48	320×40	354×46	350×47
704×96	640×80	706×94	702×95

Table 20.1: Number of cells in X and Y direction for the four grids in each grid study.

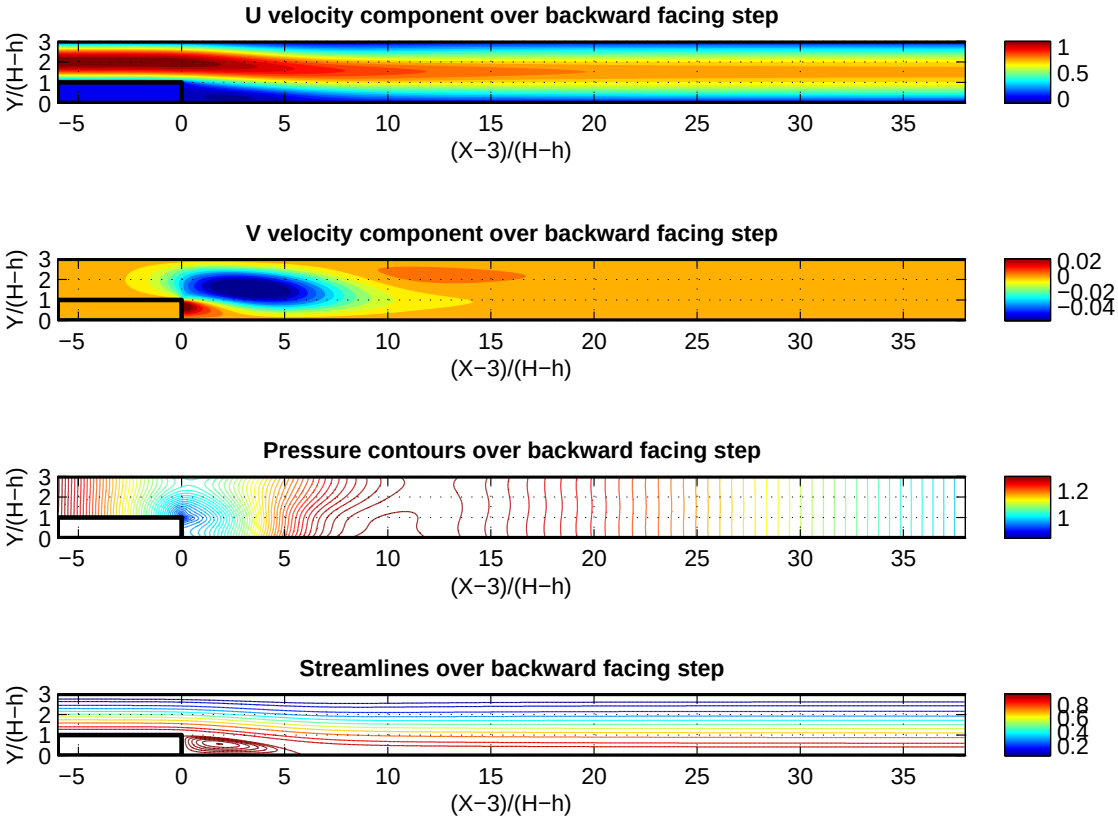


Figure 20.3: u, v, p and streamline plot for the adapted Ghost cell method on the finest grid of grid study 3.

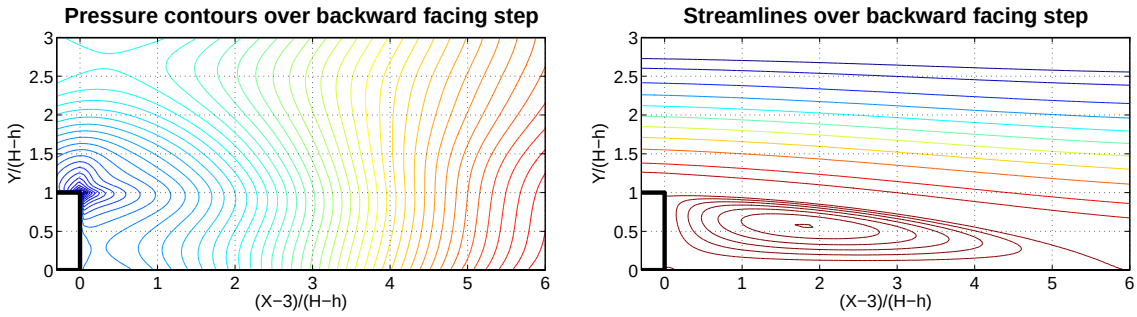


Figure 20.4: Details of the pressure contours and streamlines near the step, computed with the adapted Stair-step method on the finest grid of grid study 4.

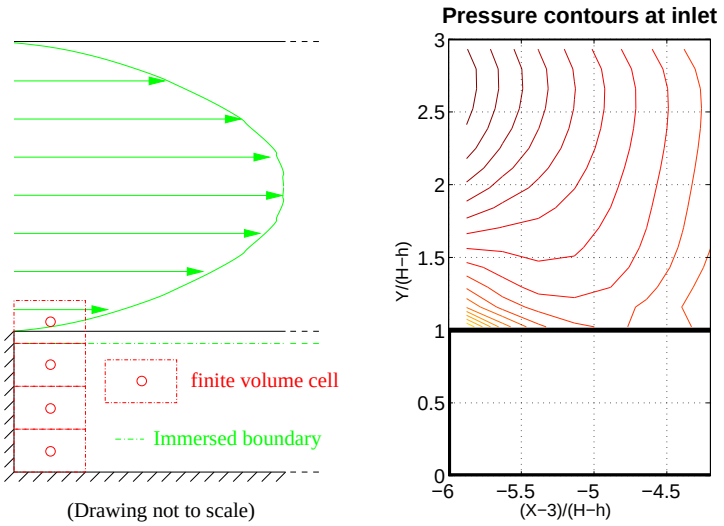


Figure 20.5: Sketch of the situation at the inlet of the channel for the Stair-step and adapted Stair-step methods (left), pressure contours under these circumstances (right).

20.2 Computational results

An overview of the flow is given in figure 20.3. In the plots, the X and Y scales are made dimensionless by dividing them by $(H - h)$. This is done to facilitate the comparison of these plots to the data in [20]. The given graphs match the experimental data from the workshop very well, and the qualitative aspects of the flow are certainly correct. The main flow diverges behind the step and gradually settles into a new fully developed laminar velocity profile near the end of the channel. Both velocity components are zero at the step boundary and behind the step one finds a region of very low velocity fluid (i.e. a separation bubble). The pressure contour plot looks good too: (quasi-) straight isobar lines are found near the inflow and outflow boundaries and a low pressure spike is seen near the step corner (figure 20.4, left). The general pressure distribution over the channel matches the numerical results of the workshop participants.

The streamline plot is made by computing the stream function in each cell k as

$$\psi_k = -\int_{y=H}^{y_{k_{center}}} u \; dy,$$

scaled to fit the $[0, 1]$ range, and plotting the contours in the channel. In practice, the integral is approximated by taking the Riemann sum since u is assumed to be constant in a cell. The result is shown in figure 20.3, a detail of the recirculation zone can be seen in fig. 20.4, right. Again, the similarity with the published results is undeniable. The streamlines are smooth and the recirculation zone is represented very well. From the stream function and the streamline plot all the information needed for the grid convergence studies can be gathered.

However, not all is well. The fact that the immersed boundary starts right at the inflow boundary creates an unfortunate situation. At the point where the inflow boundary and the step meet, the boundary conditions should match each other. Especially the Stair-step and adapted Stair-step methods run into trouble here: these two codes translate the immersed boundary (and the boundary condition) to the nearest grid line, thus creating a mismatch between inflow and step boundary condition. This situation is sketched in figure 20.5, left. The influence of this problem on the flow velocity and streamline plots is negligible, but the effect on the pressure contours near the inflow boundary of the channel is substantial. Figure 20.5, right shows this for the Stair-step method: the isobar lines that are supposed to run more or less vertical in this area are

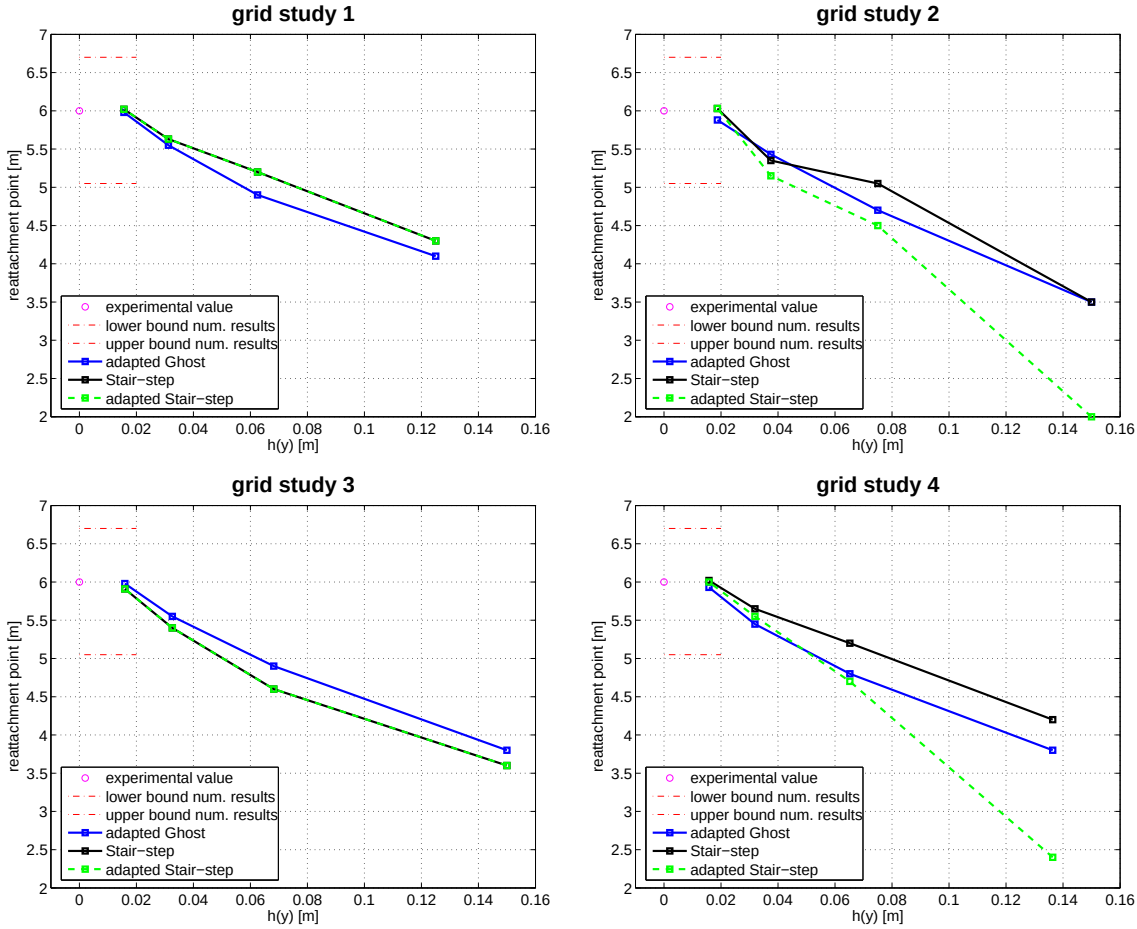


Figure 20.6: Grid studies (see section 20.1 and table 20.1) for the length of the recirculation bubble. The numerical results mentioned in the plots can be found in [20]. $h(y)$ is the cell size in Y direction.

deformed, so there is a pressure gradient on the inflow boundary.

20.3 Grid studies

While the general plots show a high level of similarity between the IBMs and the experimental data (albeit mainly qualitative), the real test is the quantitative assessment of the computed flow with respect to the workshop proceedings. This is where the grid studies come in. Three quantifiable aspects of the flow are chosen for analysis: the length of the circulation zone behind the step, the maximum of the stream function at the outflow boundary (which gives an indication of mass loss or gain) and the overall minimum of the stream function. These parameters are the subject of figures 20.6, 20.7 and 20.8. The parameters are plotted against $h(y)$, the cell size in Y direction, since it is mainly the cell density in this direction which determines the quality of the solution.

The graphs show that, in general, the three IBMs converge with decreasing cell size and that the differences between the methods become smaller. The length of the recirculation zone looks like it is going to end up around 6.5 when h goes to zero (fig. 20.6). This is more than the value from the wind tunnel experiments (6), but still within the range of numerical results as found by the participants to the workshop. Moreover, since the accuracy of the measurements in the wind tunnel experiment is limited and because the position

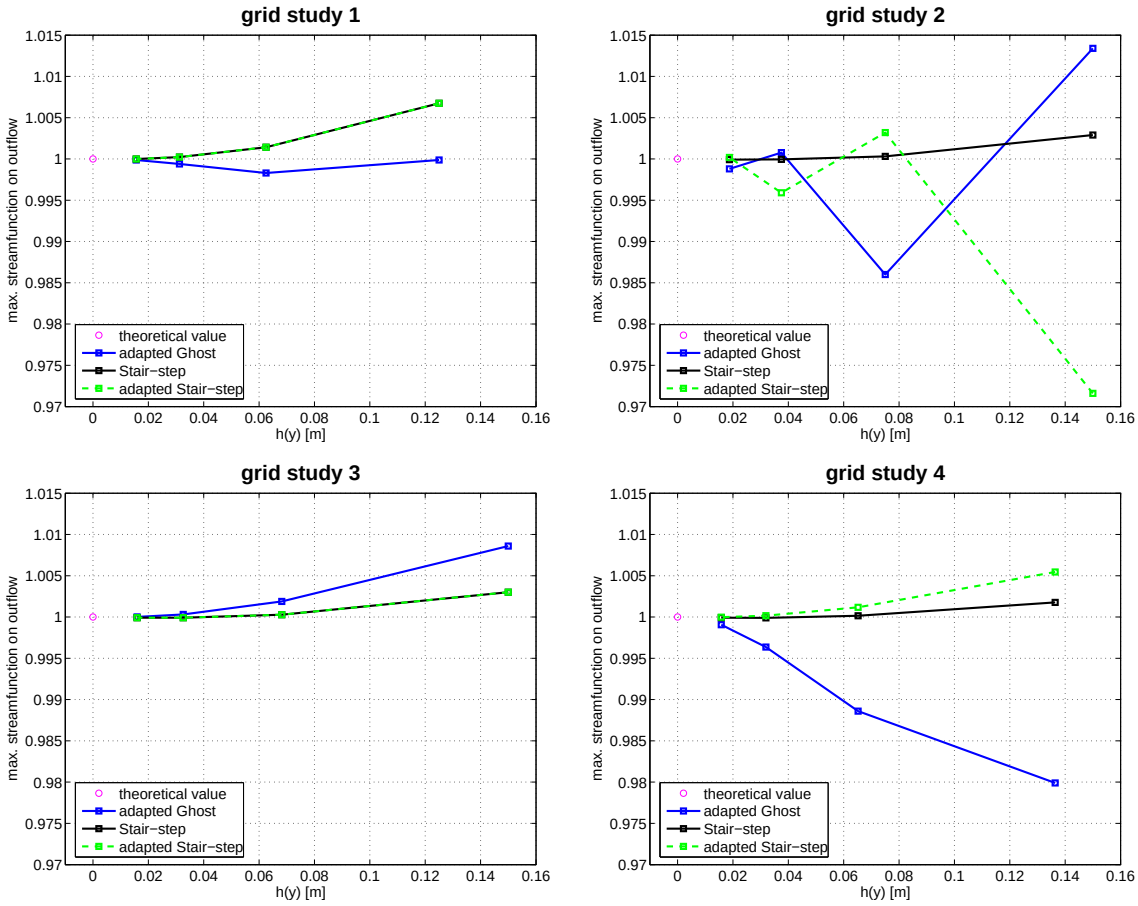


Figure 20.7: Grid studies (see section 20.1 and table 20.1) for the maximum of the stream function on the outflow boundary. $h(y)$ is the cell size in Y direction.

of the reattachment point in the calculations depends on the flux discretization, total elimination of the difference between wind tunnel experiments and computation is a challenging ambition. The maximum of the stream function at the outflow boundary should theoretically converge to 1, which is confirmed in figure 20.7. The value of the streamfunction in the center of the separation bubble was measured from the experiments to be -0.016, however many of the workshop CFD codes found lower values, some even as low as -0.030. The IBMs are likely to converge to a value of -0.022, well within the range of numerical results. In general, it is fair to say that especially on the finer grids, the differences between the methods are rather small.

The influence of the position of the immersed boundary relative to the grid lines as found in chapter 14 is observed in 2D as well. Note the difference between grid studies 1, 3 and 4 on the one hand and ”normal” grid study 2 on the other: the latter produces plots which are much more irregular than the others. This phenomenon is best seen in figure 20.7. However, the effect is smaller than in 1D. This can be explained: the geometry limits the possible relative vertical positions of the immersed boundary in a cell to three (0, $\frac{1}{3}$ and $\frac{2}{3}$). However, from figure 14.9 it can be seen that the largest difference in error is situated between 0 and $\frac{1}{2}$. And to avoid using body fitted grids, grid study 2 alternates between $\frac{1}{3}$ and $\frac{2}{3}$ relative immersed boundary position for the top of the step, so the effect of the immersed boundary treatment on the error is almost identical. The remaining oscillation in the grid study 2 data results then from the relative position of the back of the step in the cell.

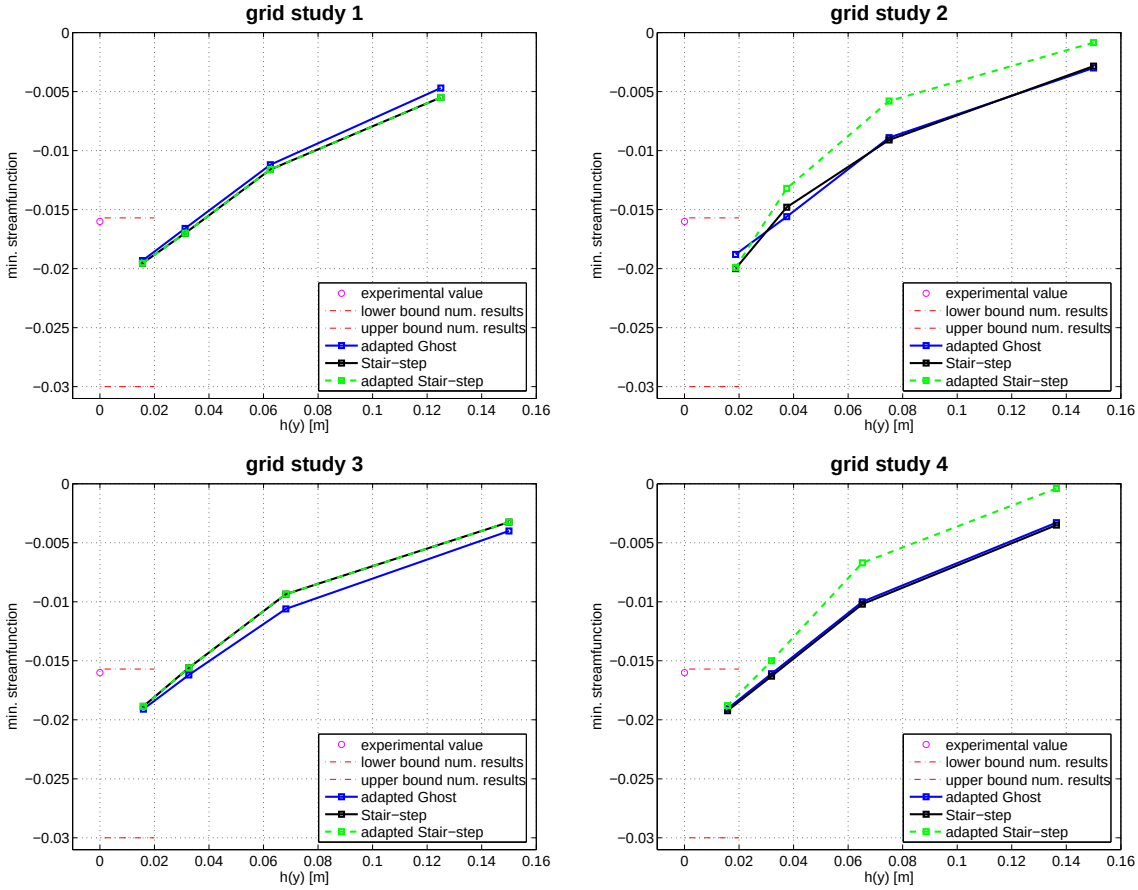


Figure 20.8: Grid studies (see section 20.1 and table 20.1) for the value of the stream function in the middle of the recirculation bubble. The numerical results mentioned in the plots can be found in [20]. $h(y)$ is the cell size in Y direction.

However, the oscillations in the normal grid study (study 2) are expected to disappear when treating problems with curved boundaries. The geometry of the backward facing step is aligned with the grid lines, so the relative position of the immersed boundary in a cell is the same in a large number of cells. Most other bodies do not suffer from this problem: the relative immersed boundary positions are usually more randomly distributed and so are the error fluctuations associated with the relative positions.

20.4 Conclusions

The results presented in this chapter are quite satisfactory. The analyzed parameters and plots compare well with those found in the experimental study, as well as with the values computed by the body-fitted methods of the workshop participants. The differences between the methods are small, this is probably a consequence of the fact that the finite volume code is first order accurate. And finally, the shape of the backward facing step does not allow a complete assessment of the IBMs since the immersed boundary is aligned with the grid lines.

Chapter 21

Test case II: circular cylinder in channel

21.1 Introduction

To test the quality of the IBMs under conditions where the immersed boundary is not aligned with the grid lines, another well documented test case is chosen. It is the steady, laminar flow around a circular cylinder in a channel, subject of an other CFD workshop. Wind tunnel experiments as well as numerical results are published in [25]. The test case under consideration is sketched in figure 21.1. It shows a circular cylinder with a 0.1 m diameter, its center located at 0.2 m from the inlet and bottom of the channel. The channel itself is 2.2 m long and 0.41 m wide. As with the backward facing step flow, the pressure is kept constant at the outflow boundary while a parabolic velocity profile with maximum velocity 0.3 m/s is prescribed at the inflow boundary. The Reynolds number based on cylinder diameter is set at 20, which should result in a steady laminar separation region behind the body. Another special feature of this flow is the fact that the cylinder is placed slightly below the center line in the channel. This in combination with the parabolic velocity profile will result in both a drag and lift force acting on the body.

Choosing a circular cylinder as a test case does have some disadvantages. One of them is that the separation point is ill-determined, which means that grid resolution is critical in achieving accurate results, since the position of the separation point depends on the amount of numerical viscosity. Three refinement levels are adopted for the H - type cartesian grid (see e.g. figure 18.2). The properties of the three grids used are summarized in table 21.1. The finest grid is limited to less than $60,000$ cells due to limitations in time and computing facilities, while the numerical methods described in [25] use body-fitted grids containing between $100,000$ and $10,000,000$ cells. Consequently, the results obtained in this grid study will not be entirely comparable with the data presented in the paper. However, the calculations should provide the trends towards the correct solution and the qualitative aspects of the flow must be present.

21.2 Computational results

Like in the backward facing step problem, the data are presented as u, v, p and streamline plots on the one hand and a grid study assessing the convergence of the methods on the other hand. For the grid study, four quantifiable properties of the solution that are suggested in [25] will be considered: the lift coefficient (C_l),

Grid	grid size	uniform region	total no. of cells
coarse	117×57	45×30	$6,669$
medium	234×101	90×60	$23,634$
fine	324×181	180×120	$58,644$

Table 21.1: Coarse, medium and fine grid properties for the circular cylinder.

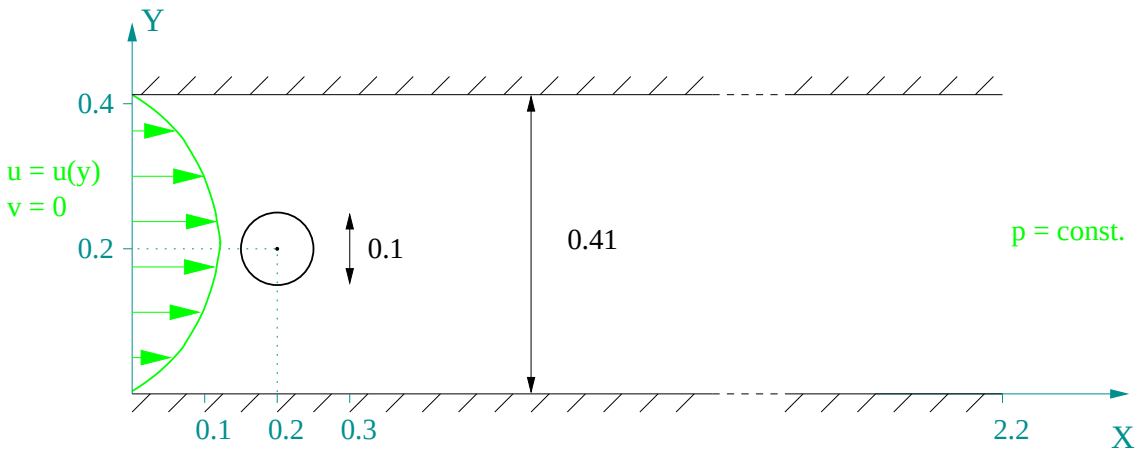


Figure 21.1: Sketch of the circular cylinder in the channel.

the length of the recirculation zone, the pressure difference between front and back of the cylinder and the maximum of the stream function at the outflow boundary.

Figure 21.2 shows the u and v velocity components in the channel. The asymmetry in the test setup is clearly present in the flow: the u velocity in the upper part of the channel is generally higher than the velocity in the lower part. The vertical flow velocities are high near the front of the cylinder, as the flow is forced around the body.

The pressure contours and streamlines can be found in figure 21.3. A large jump in cell sizes between the uniform refined grid near the immersed boundary and the surrounding parabolically refined grid makes some of the pressure contours look non-smooth. This problem with the grid generator was later addressed and fixed before starting with the multi-element airfoil calculations (next chapter). In general, however, the pressure contours are satisfactory: there is a smooth stagnation area with high pressure in front of the cylinder and a reasonably large negative pressure gradient above and below the body as the fluid is squeezed through. The isobars in the wake suffer slightly from the previously mentioned grid mismatch, but they become quite straight and vertical near the outflow boundary. A detail of the pressure plot is given in figure 21.4, left. In this graph, the positive pressure gradient on the boundary between 100 and 140 degrees from the stagnation point, which causes the separation, is clearly visible.

The streamlines in the channel conform to expectations as well, the really interesting part of the flow is represented well in figure 21.4, right. Here we see the recirculation zone, plotted as the contours of the streamfunction. Because the flow is asymmetric, the upper recirculation bubble is longer than the lower one. This picture is qualitatively very promising (see e.g. [9]), however calculations on finer grids are desired to increase the accuracy of the solution.

21.3 Grid study

In contrast to the backward facing step test case, only one grid study is performed. The reason for this is that the relative positions of the immersed boundaries in the cells are now more evenly distributed, the effect of changing relative positions with grid refinement has become negligible. Another remark that was made in the previous chapter is now true as well: the coarsest grid in this study is actually very coarse compared to the complexity of the problem. The results obtained on this grid are therefore quite poor and their presence in the plots is debatable, since the computations did not show any separation at all.

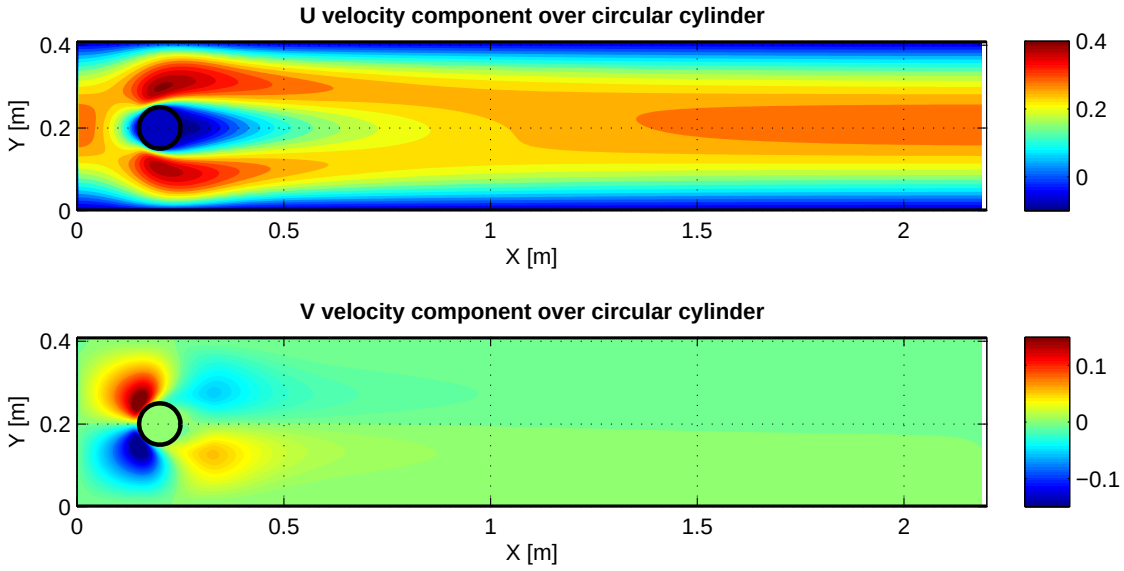


Figure 21.2: Velocity u and v plot for the adapted Ghost cell method on the fine grid.

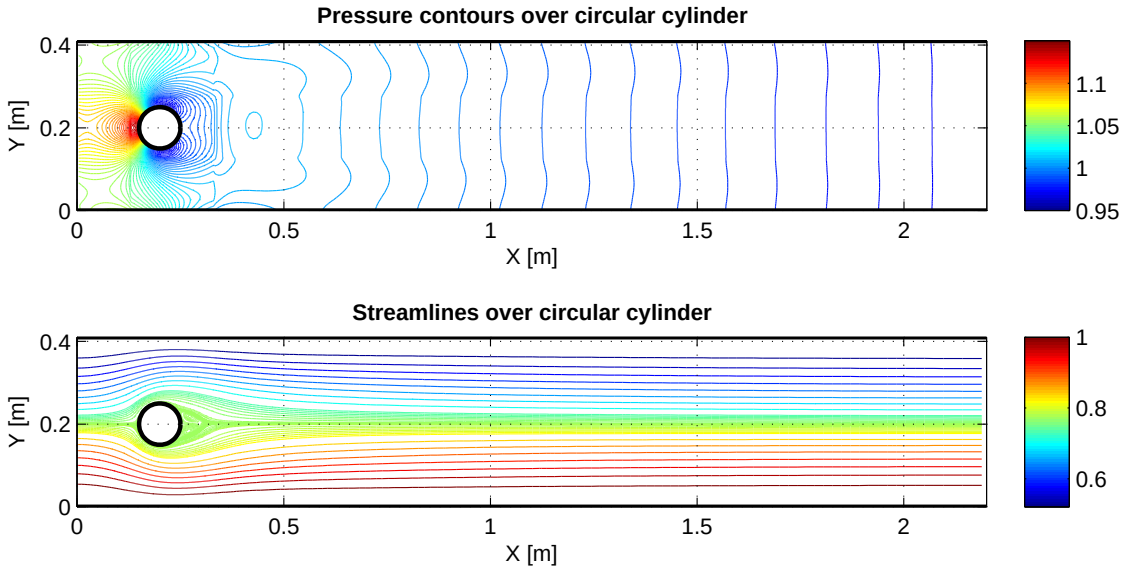


Figure 21.3: Pressure p and streamline plot for the Stair-step method on the fine grid.

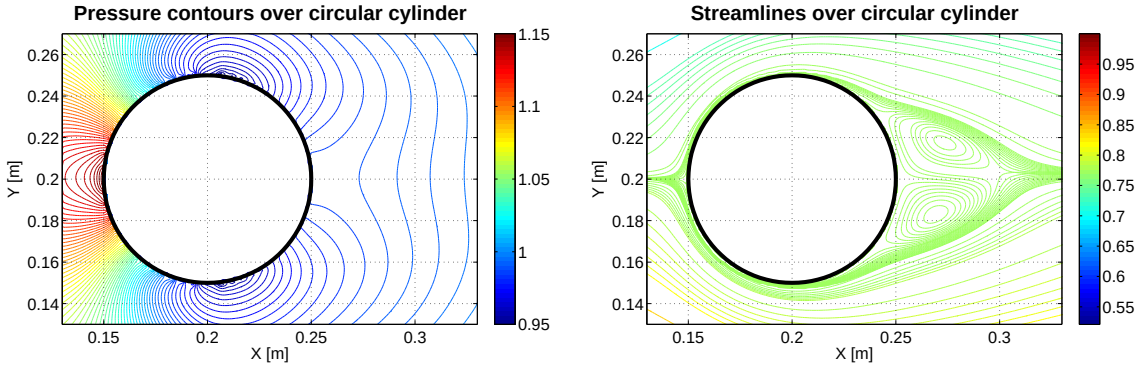


Figure 21.4: Details of the pressure contours and streamlines around the cylinder, computed with the adapted Stair-step method on the fine grid.

However, the grid study plotted in figure 21.5 does seem to yield encouraging data: the lift coefficient, the length of the recirculation zone and the pressure difference converge towards their respective fine mesh values as computed by the participants to the workshop. The maximum of the stream function is close to its theoretical value (the error is smaller than 1 promille). Calculations on finer grids are desired to confirm these trends.

21.4 Conclusions

The circular cylinder test case is a much more challenging flow problem than the backward facing step. The IBMs are able to deal with the circular shape, although adequate grid refinement near the boundary is necessary to capture the shape of the boundary as well as the physical processes. The adapted Ghost cell, Stair-step and adapted Stair-step methods produce comparable results that show a converging trend towards the benchmark solution. The qualitative aspects of the flow are all present on the finer grids, but additional computations on finer grids are recommended in order to obtain more accurate quantitative data.

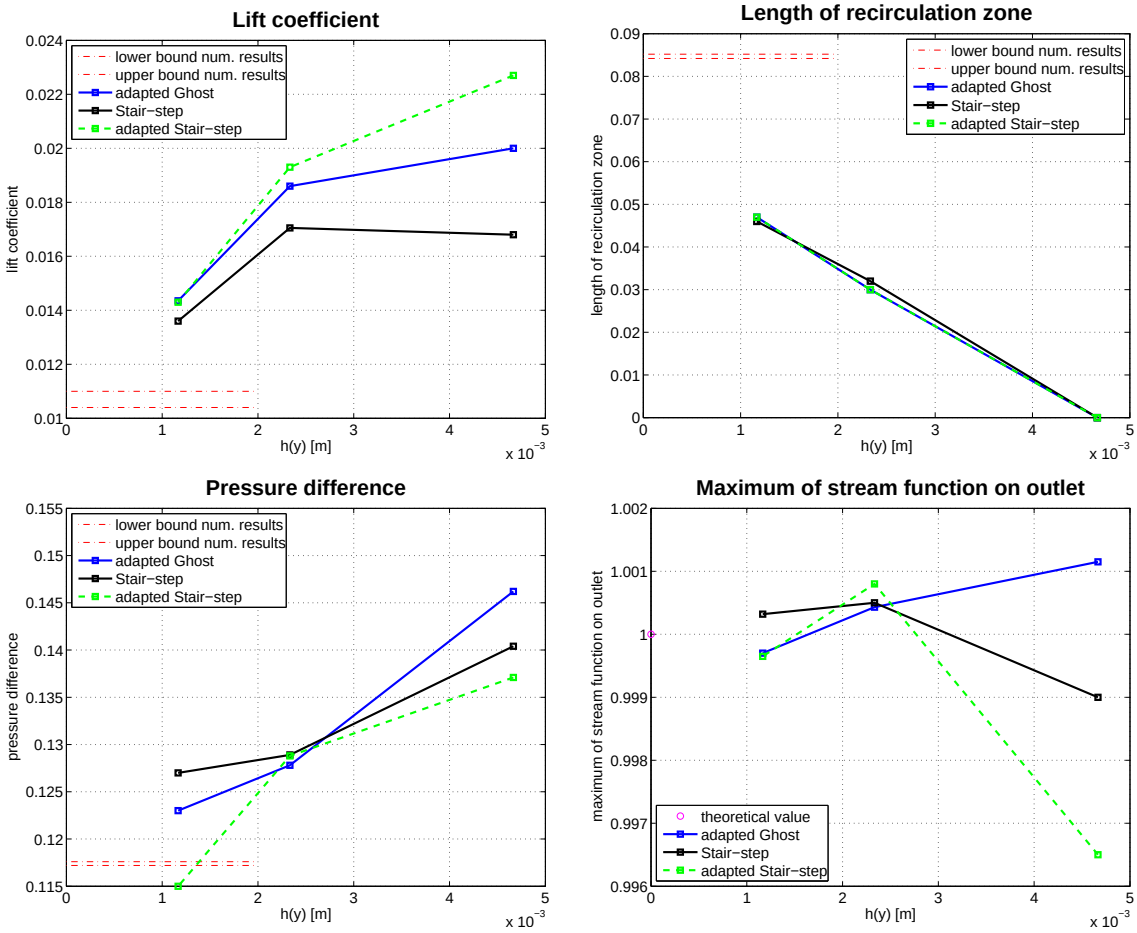


Figure 21.5: Grid study for the value of the lift coefficient, the length of the recirculation zone, the pressure difference over the cylinder and the maximum of the stream function at the outlet. The numerical results mentioned in the plots can be found in [25]. $h(y)$ is the cell size in Y direction.

Chapter 22

Case III: multi-element airfoil

While the previous two chapters concentrated on the assessment of the IBMs in terms of accuracy and solution quality, the real strength of the methods remains unexploited. The backward facing step and circular cylinder are simple geometries so their body-fitted grids are easy to generate. But using IBMs particularly makes sense when the body shape is complex. Therefore, a third flow simulation is considered, to show the real potential of the cartesian grid methods: the flow around a multi-element airfoil known from literature. However, there are a few problems. First of all, this flow problem requires Reynolds numbers in the order of 10^6, which means thin boundary layers and a lot of cells. Due to limited computing resources, this was impossible to achieve in the time remaining, so the Reynolds number is set at 1000. Secondly, the flow around the airfoil could well be turbulent (certainly turbulent at $re = 10^6$), which means that the laminar code that was developed in this project is of limited use. And finally, since one or more airfoil elements will face higher angles of attack (5 - 25 degrees), unsteady flow separation may become a problem.

However, all these problems are actually not so important, because the goal of this simulation is not to validate the code by finding very accurate solutions, but to show the potential of IBMs. Obtaining accurate results for these kinds of flows obviously requires local grid refinement, turbulence models and powerful computing facilities. The only thing that will be shown here is that IBMs are able to find solutions for complex problems in an easy and straightforward way, while obliterating the need for time-consuming body-fitted grid generation.

22.1 The flow problem

The multi-element airfoil that is chosen for this simulation is defined in [19] as case A-2. It is composed of a slat, a main element and a flap, with a total chord length of 1.28 meter. For this simulation, the angle of attack is set to 10 degrees and the Reynolds number to 1000. To avoid disturbances in the solution, the domain boundaries are chosen far from the airfoil: it is placed in a 50×50 meter H - type cartesian grid with symmetry boundary conditions on top and bottom domain boundary. A uniform freestream inflow condition and a constant pressure outflow complete the domain boundary conditions. The grid is depicted in figures 22.1 and 22.2. Just for comparison, a sketch of the grid structure as used in [23] is shown in fig. 22.3.

The calculation is performed on a coarse and a fine grid (see table 22.1). Both grids require some retouching by hand since the *ib flag* and *ghost flag* routines in the pre-processor cannot deal with thin trailing edges,

Grid	grid size	uniform region	total no. of cells
coarse	360×280	200×100	$100,800$
fine	560×400	400×200	$224,000$

Table 22.1: Coarse and fine grid properties for the multi-element airfoil.

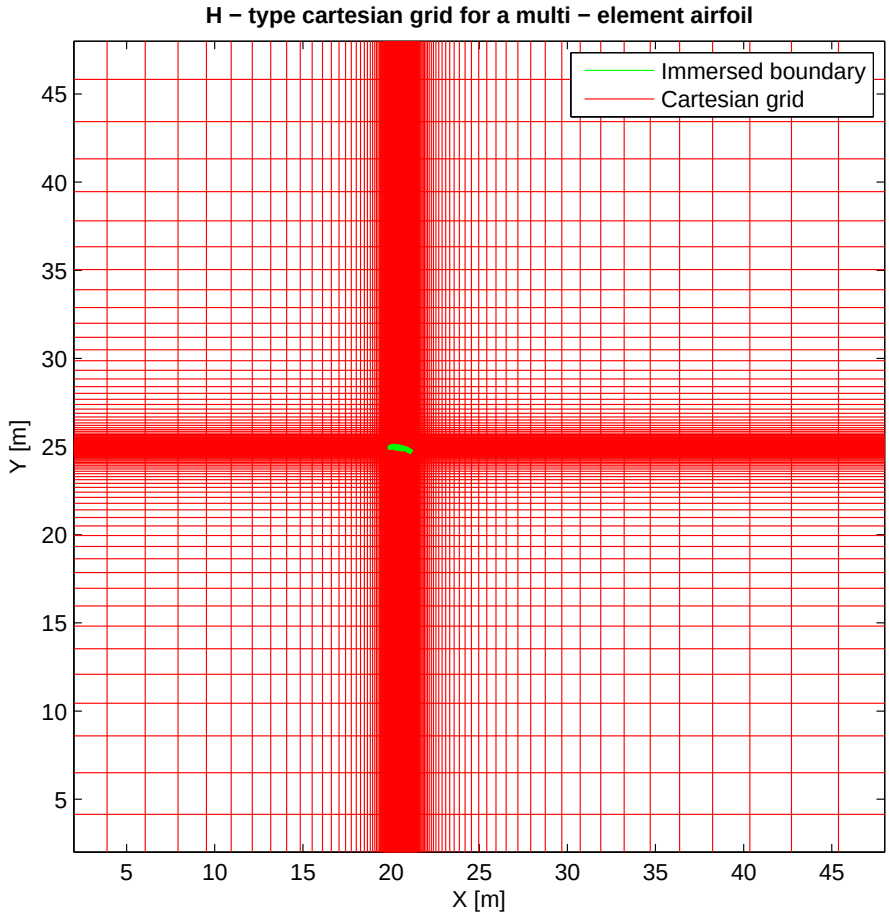


Figure 22.1: The H - type grid for the multi-element airfoil.

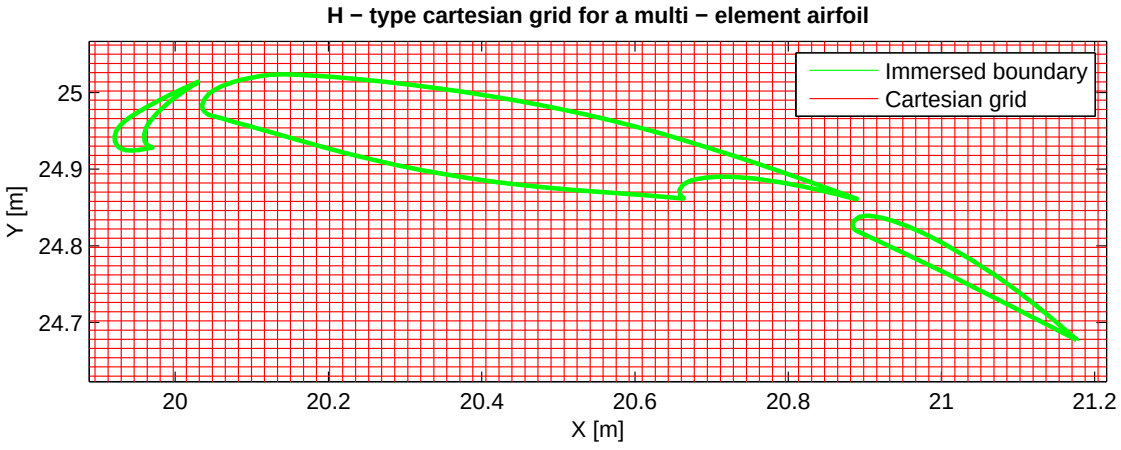


Figure 22.2: Detail of the H - type grid: the uniform grid region around the airfoil.

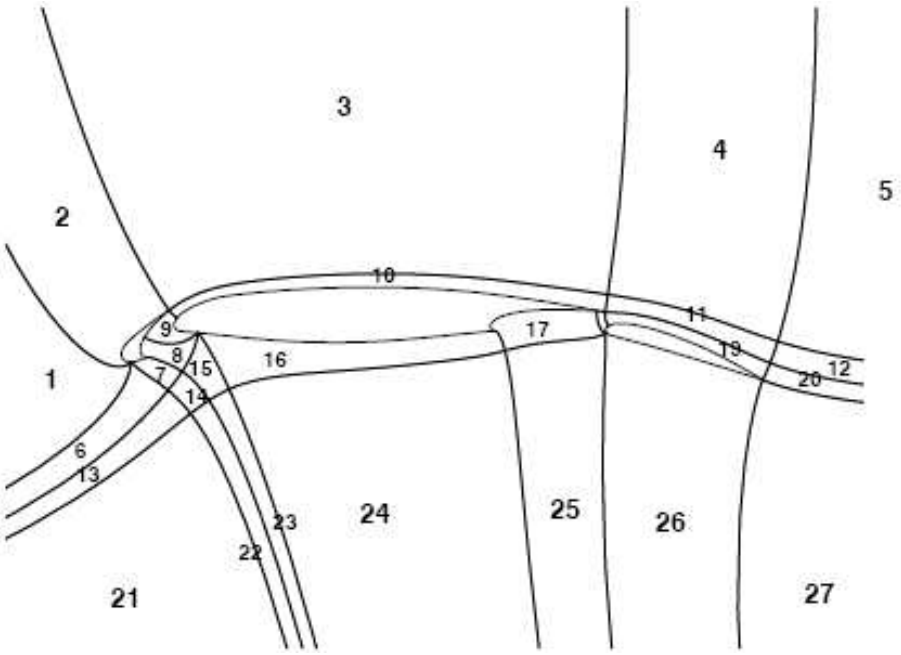


Figure 22.3: The structured grid for the multi-element airfoil as used by [23] (courtesy of S. De Rango & D. W. Zingg).

cf. chapter 18. On the downside, the small cell sizes near the airfoil (and correspondingly small time steps) and the rather large physical domain took their toll on the computation time: about $3-4$ days on a 64-bit AMD Athlon processor per computation. For this reason, the calculations were done with the adapted Stair-step method, as this code is able to handle up to five times larger time steps than the adapted Ghost cell method. This may be caused by the extrapolation scheme in the adapted Ghost cell method, which can yield disproportionally large fluxes on a cell face when an immersed boundary segment lies very close to a fluid cell center.

22.2 Results

The computed velocities, pressure and streamlines on both grids are plotted in figures 22.4, 22.5, 22.6, 22.7 and 22.8. At first sight it is obvious that there is a big difference between the results on the fine and coarse grid. The reason for this is simple: at the Reynolds number and angle of attack chosen for this problem, no steady physical solution exists. However, on the coarse grid, more artificial diffusion is added to the system, which means that instabilities which are in reality supposed to be present, are damped out by the discrete solution method. This effect is smaller on finer grids and, as a consequence, the unsteady flow over the airfoil does not converge to a steady solution. The data presented from the fine grid calculation are in fact a snapshot at an arbitrary moment in the iteration process, when the total of the residuals (sum of fluxes over all cells) has converged to a level where it is not reduced anymore by further time-stepping. So, while the plots show a steady recirculation zone behind the airfoil on the coarse grid, the fine grid plots show a Von Karman vortex street pattern. This seems not impossible, because the Reynolds number based on airfoil thickness is of order 100, which is the Von Karman flow regime for circular cylinders. However, as mentioned before, it is possible that the real solution looks rather different since the current method is not time-accurate.

But even if the computed solution is not physically correct, the figures do show that IBMs are able to handle complex geometries in an simple manner. The calculations took some time, but the usually labour-intensive

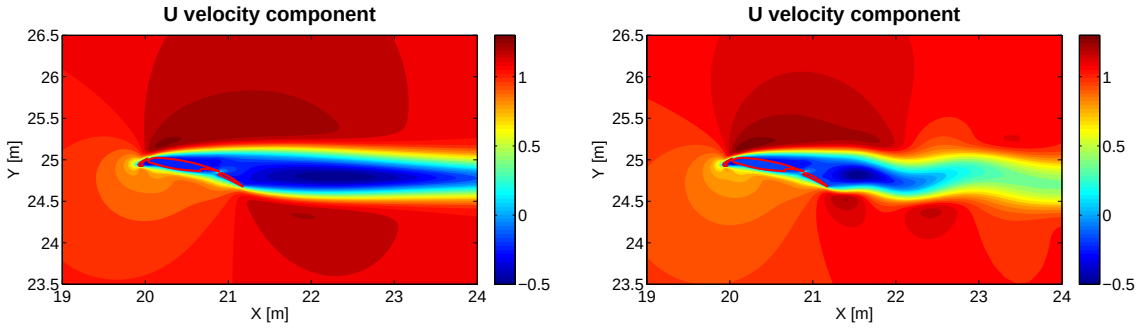


Figure 22.4: Horizontal velocity over multi-element airfoil on coarse grid (left) and fine grid (right).

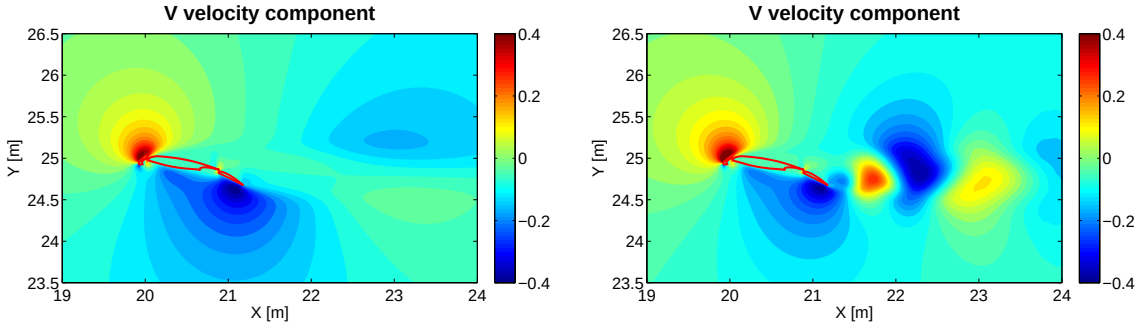


Figure 22.5: Vertical velocity over multi-element airfoil on coarse grid (left) and fine grid (right).

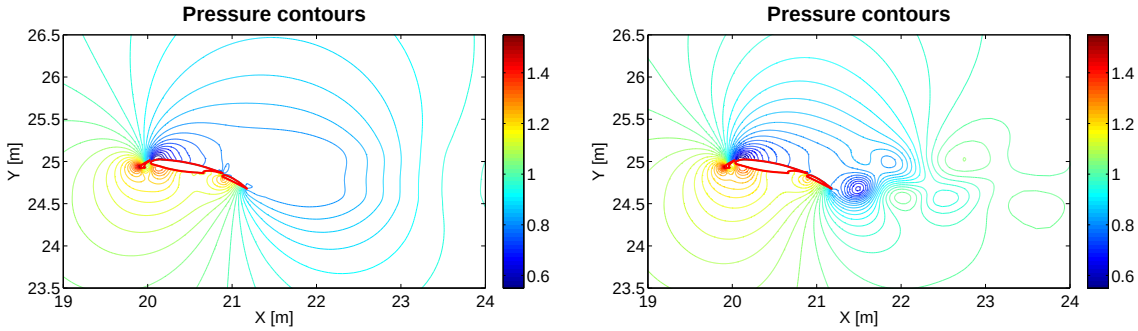


Figure 22.6: Pressure contours over multi-element airfoil on coarse grid (left) and fine grid (right).

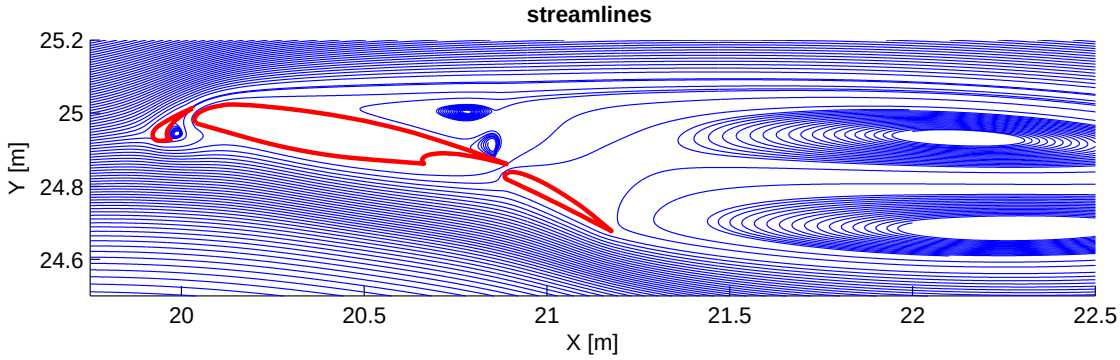


Figure 22.7: Streamlines over multi-element airfoil on coarse grid.

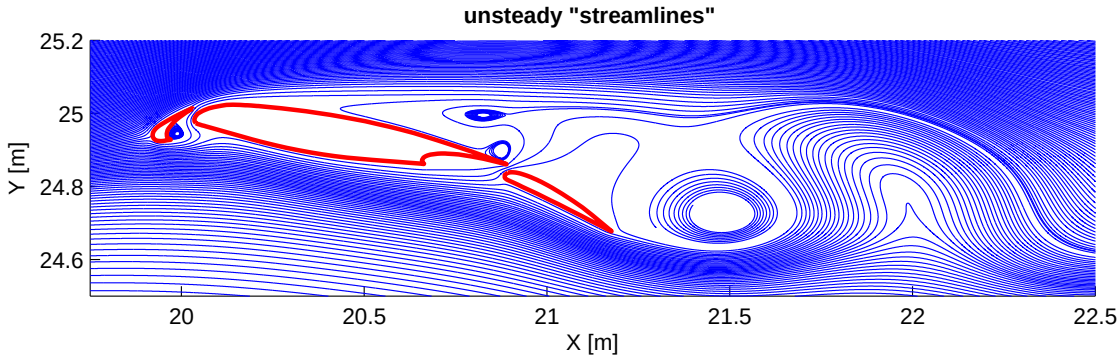


Figure 22.8: Unsteady ”streamlines” over multi-element airfoil on fine grid.

grid generation process was completed in minutes. Since this ”proof of concept” is the main goal of this chapter, the detailed analysis of the plots is kept brief.

In general, the flow around the front part of the airfoil looks steady. There is a small recirculation region behind the slat and a stagnation point on the bottom of the main element. The flow separates on the top of the main element not far aft from the nose. Streamlines are passing through the gaps between all three airfoil elements, a typical feature for these kinds of airfoils. The pressure plots indicate that the airfoil certainly generates lift. Vortexes seem to form on top of the main element and flap, which are shedded downstream with the flow (fine grid calculations). This phenomenon is particularly visible in the vertical velocity and pressure contour plots. The shedded vortexes decay within five chord lengths due to the first order accuracy of the finite volume method. In figure 22.8, unsteady ”streamlines” are plot. These are based on the instantaneous velocity field, but they are no real streamlines. However, they give a good idea about how the flow behaves. For that reason the plot was included in this chapter.

Chapter 23

Conclusions on the 2D IBMs study

The 2D IBMs study essentially breaks down into three parts: the grid generation, the IBMs codes themselves and the test cases.

The pre-processor is an important part of the code, as it not only defines the actual grid nodes but also the intersections of the grid lines with the immersed boundary. Additional parameters are defined to create a data structure that allows the IBMs to deal with immersed bodies effectively. The main challenge is to create a pre-processor that can handle objects of a general nature, such that the process is automatic and no human interaction is required. This aspect is still problematic for thin, wedge-like shapes like airfoil trailing edges.

In total, four Immersed Boundary Methods were developed and implemented in the 2D first-order finite volume code. Three of those were successfully tested: the adapted Ghost cell method, the Stair-step method and the adapted Stair-step method. The total error is predominantly determined by the finite volume code and not by the immersed boundary treatment. As a result, the differences in accuracy between the results obtained by the various methods are smaller than expected. When combined with a higher-order solver, the adapted Ghost cell method might yield more accurate results than the Stair-step methods.

The IBMs were tested on two test cases: the backward facing step and a circular cylinder in a channel. Both qualitative and quantitative aspects of the flows are evaluated using flow visualizations and grid studies. The results are encouraging but, especially in the case of the cylinder, additional fine grid calculations are still desired. Finally, to demonstrate the ability of IBMs to deal with complex geometries, the flow around a multi-element airfoil was computed.

The main problem encountered during the flow calculations is the rather long computation time. This limited the number of calculations as well as the maximum number of grid cells. The reason for the high computational cost is the explicit time-stepping method used in the finite volume solver. It is known to be inefficient for solving steady flow problems due to the limitation of the maximum time step for stability reasons.

Part IV

Conclusions and Recommendations

Chapter 24

Conclusions

Immersed Boundary Methods (IBMs) are a class of methods in Computational Fluid Dynamics where the grids do not conform to the shape of the body. Instead they employ cartesian meshes and alternative ways to incorporate the boundary conditions in the (discrete) governing equations. The simple grids and data structure are well suited to handle complex geometries and moving boundaries. The disadvantage of these methods is that they need larger grids to achieve the same accuracy compared with body-conformal meshes. The difference between the total number of grid points for cartesian versus body-fitted grids scales with $Re^{1.0}$ in 2D and $Re^{1.5}$ in 3D, which means that the penalties are substantial for high-Reynolds-number flows.

The 1D IBMs study reveals that the boundary condition on the immersed boundary can be imposed in many ways, with varying accuracy. The Poiseuille flow considered in this study forms a good test case that allows the IBMs to be compared with the analytical solution. The more accurate methods are able to approximate the analytical solution perfectly and as a result, their discrete L_2 error norms are almost zero and cannot be compared amongst themselves. One of the most interesting findings is that the accuracy of the IBMs is not only determined by the grid size, but also by the relative position of the immersed boundary with respect to its neighboring grid points. This property can cause non-monotonous behavior in grid convergence studies. The problem is not present for curved bodies in 2D and 3D because the relative immersed boundary positions are usually more randomly distributed.

In the 2D phase, a pre-processor and an IBM flow solver were constructed and tested on benchmark problems. The pre-processor is an essential part of an Immersed Boundary code, as it not only defines the actual grid nodes but also the intersections of the grid lines with the immersed boundary. Additional parameters are defined to create a data structure that allows the IBMs to deal with immersed bodies effectively. The current pre-processor can handle most body shapes fully automatically. Thin, wedge-like shapes (e.g. airfoil trailing edges) still need a little bit of hand-coding.

Three IBMs have been successfully implemented in a first-order finite volume code for the Navier-Stokes equations. The solver uses artificial compressibility and explicit time-stepping to march the discrete system towards a steady laminar solution. This approach is known to be rather inefficient. The resulting computational overhead limited the grid size and hence the accuracy attainable within the duration of the project. Furthermore, since the flow solver is first-order accurate, it dominates the total error, irrespective of the immersed boundary treatment used. This makes it hard to compare the quality of the methods. One thing became very clear however, that is that the adapted Ghost cell method requires up to five times smaller time steps to remain stable than the Stair-step and adapted Stair-step methods.

The 2D IBMs were tested on three test cases: a backward-facing step flow, a circular cylinder flow in a channel and a multi-element airfoil flow. The results show that Immersed Boundary Methods are able to treat different boundaries in a satisfying manner. The qualitative aspects of the flows are captured well. Moreover, the grid generation is very straightforward and fast, even for the multi-element airfoil.

Chapter 25

Recommendations

General

The problem with the pre-processor being still unable to recognize cells inside thin, wedge-like shapes as being inside the body can be solved by reversing the pre-processing procedure in the sense that for cells outside the body it is checked whether they are separated from their neighbors by an immersed boundary segment. This would also obliterate the need to run the procedure more than once if multiple bodies are concerned.

The uniform and H - type cartesian grids do have their shortcomings, especially when high-Reynolds-number flows are simulated. Local grid refinement is essential to ensure an accurate solution of the Navier-Stokes equations without increasing the computational costs more than necessary. The disadvantage is that local grid refinement requires a more complex data structure.

Implementing the IBMs in a higher order finite volume code will increase the accuracy and might show more difference in accuracy between the methods.

The present method is designed for laminar flow. If desired, it can be fitted with a turbulence model, without changing the general setup of the solver.

Steady codes

To solve steady flows efficiently it is imperative to convert the explicit time-stepping method into an implicit method. This will reduce the computational time considerably.

Unsteady codes

The current steady laminar code can be transformed into an unsteady solver by removing the artificial compressibility term and adding a pressure-correction routine per time step. This renders the method time-accurate and requires no major modifications to the 2D code.

Including moving boundaries in an unsteady IBM is another possibility. The only extra problem is posed by the "freshly-cleared" cells, cells that were inside an immersed body at time t and that are in the fluid at time $t+\Delta t$. Their state can be determined by extrapolation from surrounding fluid cells where the solution is known at time $t+\Delta t$. A code that is capable of handling moving boundaries is an important step towards the successful simulation of Fluid-Structure Interaction phenomena.

Bibliography

[1] Robert A. Adams. *Calculus: A Complete Course.* Fourth edition, Addison Wesley. 1999.

[2] J. D. Anderson, Jr. *Fundamentals of Aerodynamics* Second edition, McGraw-Hill, Inc. 1991.

[3] P. Angot, C. H. Bruneau, P. Frabrie. A penalization method to take into account obstacles in viscous flows. *Journal of Numerical Mathematics* 81: 497-520. 1999.

[4] R. P. Beyer, R. J. LeVeque. Analysis of a one-dimensional model for the immersed boundary method. *SIAM Journal on Numerical Analysis* 29: 332-364. 1992.

[5] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics* 2:12-26. 1967.

[6] E. Dick. A flux-vector splitting method for steady Navier-Stokes equations. *International Journal for Numerical Methods in Fluids* 8:317-326. 1988.

[7] E. Dick. A multigrid method for steady incompressible Navier-Stokes equations based on partial flux splitting. *International Journal for Numerical Methods in Fluids* 9:113-120. 1989.

[8] E. Dick, J. Linden. A multigrid method for steady incompressible Navier-Stokes equations based on flux difference splitting. *International Journal for Numerical Methods in Fluids* 14:1311-1323. 1992.

[9] M. Van Dyke. *An Album of Fluid Motion.* Parabolic Press. 1982.

[10] J. E. Emblemsvag, R. Suzuki, G. V. Candler. Cartesian grid method for moderate-Reynolds-number flows around complex moving objects. *AIAA Journal* 43: 76-86. 2005.

[11] D. Goldstein, R. Handler, L. Sirovic. Modeling a no-slip flow boundary with an external force field. *Journal of Computational Physics* 105: 354-366. 1993.

[12] G. Iaccarino, G. Kalitzin, P. Moin, B. Khalighi. Local grid refinement for an immersed boundary RANS solver. *AIAA Paper 2004-0586.* 2004.

[13] G. Iaccarino. Personal communication. Stanford, March 2005.

[14] J. van Kan, A. Segal. *Numerieke Methoden voor Partiële Differentiaalvergelijkingen.* Delftse Uitgevers Maatschappij. 1993.

[15] K. Khadra, P. Angot, S. Parneix, J. P. Caltagirone. Fictitious domain approach for numerical modeling of Navier-Stokes equations. *International Journal for Numerical Methods in Fluids* 34: 651-684. 2000.

[16] M. C. Lai, C. S. Peskin. An immersed-boundary method with formal second-order accuracy and reduced numerical viscosity. *Journal of Computational Physics* 160: 705. 2000.

[17] R. Mittal, G. Iaccarino. Immersed boundary methods. *Annual Review of Fluid Mechanics 2005* 37: 239-261. 2005.

[18] J. Mohd-Yosuf. Combined immersed boundary/B-spline methods for simulation of flow in complex geometries. *Annual Research Briefs, Center for Turbulence Research* 317-328. 1997.

[19] I. R. M. Moir. A Selection of Experimental Test Cases for the Validation of CFD Codes. *AGARD Advisory report no. 303*. 1994.

[20] K. Morgan, J. Periaux, F. Thomasset (eds.). *Analysis of Laminar Flow over a Backward Facing Step: a GAMM Workshop*. Notes on Numerical Mechanics, 9, Vieweg. 1984.

[21] C. S. Peskin. *Flow patterns around heart valves: A digital computer method for solving the equations of motion.* PhD Thesis, Physiology, Albert Einstein College of Medicine. Univ. Microfilms 72: 30-378. 1972.

[22] C. S. Peskin. The fluid dynamics of heart valves: Experimental, theoretical and computational methods. *Annual Review of Fluid Mechanics 1982* 14: 235. 1982.

[23] S. De Rango, D. W. Zingg. Higher-order aerodynamic computations on multi-block grids. *AIAA paper 2001-2631*. 2001.

[24] E. M. Saiki, S. Biringen. Numerical simulation of a cylinder in uniform flow: application of a virtual boundary method. *Journal of Computational Physics* 123: 450-465. 1996.

[25] M. Schäfer, S. Turek. *Benchmark Computations of Laminar Flow Around a Cylinder.* In *Flow Simulation with High-Performance Computers*, Vieweg. 1996.

[26] R. Verzicco, J. Mohd-Yosuf, P. Orlandi, D. Haworth. LES in complex geometries using boundary body forces. *AIAA Journal* 38: 427-433. 2000.

[27] J. Wackers. An adaptive-gridding solution method for the 2D unsteady Euler equations. *Technical report MAS-N0301*. CWI, Amsterdam. 2003.

[28] J. Wackers, B. Koren. *A surface capturing method for the efficient computation of steady water waves.* In P. Bergan, J. Garcia, E. Onate, T. Kvamsdal, editors, *Proceedings of the International Conference on Computational Methods in Marine Engineering*, Oslo. 2005.

[29] D. de Zeeuw, K. G. Powell. An adaptively refined mesh solver for the Euler equations. *Journal of Computational Physics* 104:56-68. 1993.

[30] O. C. Zienkiewicz, R. L. Taylor. *The Finite Element Method.* Fifth edition, Butterworth-Heineman. 2000.

[31] *www.mathworld.wolfram.com*