A numerical study of the immersed boundary method and application to blood flow

Thesis · May 2006

DOI: 10.13140/RG.2.1.3946.6720

1 author:

Francois Pacull
Forcity
15 PUBLICATIONS **23** CITATIONS

SEE PROFILE

A NUMERICAL STUDY OF THE IMMERSED BOUNDARY METHOD AND APPLICATION TO BLOOD FLOW

A Dissertation

Presented to

the Faculty of the Department of Mathematics

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

François Pacull

May 2006

A NUMERICAL STUDY OF THE IMMERSED BOUNDARY METHOD AND APPLICATION TO BLOOD FLOW

François Pacull

APPROVED:

Dr. Marc Garbey, Chairman

Dr. Roland Glowinski,

Dr. Ronald Hoppe,

Dr. Roger Tran Son Tay,
University of Florida

Dr. John L. Bear
Dean, College of Natural Sciences
and Mathematics

To My Parents Robert and Agnès,

My Brother Jean,

and in Loving Memory

of Anne Ramon

ACKNOWLEDGMENTS

I would like to thank my advisor Marc Garbey for being a great mentor from whom I learned so much and for financially supporting me. I also wish to thank my committee members Roland Glowinski, Ronald Hoppe and Roger Tran Son Tay for their time and wise opinions. My thanks also goes out to Hatem Ltaief, Bilel Hadri and Christophe Picard, my co-workers at the University of Houston, who have been a great help. Also, I would like to express my gratitude to Cara Anderson. Finally, I would like to thank my family, for their constant support over the years.

A NUMERICAL STUDY OF THE IMMERSED BOUNDARY METHOD AND APPLICATION TO BLOOD FLOW

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Mathematics

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

François Pacull

May 2006

ABSTRACT

Blood flow simulation is a very challenging topic in Applied Mathematics. It involves state of the art techniques of image segmentation, computational fluid dynamics as well as fluid/structure interaction. The potential application for vascular diseases would be a computer-assisted diagnostic: blood flow characteristics such as blood velocity, pressure or artery wall shear stress are very important to know but difficult to measure *in vivo*.

Here, we focus on the methods for fluid/flexible-structure interaction and more specifically on the Immersed Boundary Method (IBM) of C.S. Peskin [89]. The IBM combines Eulerian and Lagrangian descriptions of flow and moving elastic boundaries using the Dirac delta function as an interpolation tool. Incompressible Navier-Stokes (NS) and elasticity theory can be unified by the same set of equations to get a combined model of the interaction. This method is very easy to implement and versatile, so that it has numerous applications in bio-engineering or in more general computational fluid dynamics. For blood flow simulation, the IBM can be used to compute the motion of large blood cells, on a small scale, and the interaction between the artery wall and the pulsating flow, on a large scale.

We present a numerical study of the accuracy, the stability and the efficiency of the IBM based on the implementation of several mathematical tools. These implementations are being made on test cases that are relevant for the IBM applications, keeping in mind that we do not want to increase its computational cost and its simplicity for three-dimensional coding.

Finally, we introduce an integrated approach to quickly compute an incompressible NS flow in a section of a large blood vessel using medical imaging data, in order to provide a first order approximation of the shear stress and the pressure on the artery wall.

Contents

List of Figures

Introduction

0.1 Toward blood flow simulations

An increasing percentage of the population is concerned with vascular diseases, due to general aging and other risk factors. The basis for the patient diagnostic is medical imaging, whose quality is constantly improving: angiography, ultrasound or magnetic resonance are some of the common imaging techniques. These images are crucial for the surgeon. Since this type of surgery is a heavy process, a current effort is being made to model blood flows inside realistic geometries, taken from medical images of blood vessels. The aim would be to have a computer-assisted diagnostic: blood flow characteristics such as blood velocity, pressure or artery wall shear stress are very important to know but hard to measure in vivo. Numerical blood flow simulations would be significant too for follow-up after an intervention, since the geometry and the characteristics of the artery have then been changed. Finally, the surgeon may introduce synthetic structures in an artery, in the case of aneurysm for example. Accurate simulations would improve the design of such stent structures.

0.2 The Challenges

Blood flood computations have been made possible by the fast increase of computational resources. However, they are a highly complex phenomena that can only be modeled by strong idealization and simplification. Even the simplified models give rise to many challenging issues. Here is a brief description of some of these challenges.

- The artery walls are elastic and react to the pulsating flow. This implies the implementation of *fluid/structure interaction methods*.
- The problem size of this type of biological simulation, is large because it involves

3D fluid flows. This is why fast solvers using *efficient domain decomposition methods and parallel computers* need to be associated with the simulations.

- The artery walls are made of heterogeneous isotropic layers, which follow complex *elasticity laws*. It is challenging to simulate the behavior of these different layers of fibers. The stiffness of the vessels is strongly related to blood tension and arteriosclerosis, which is the common hardening of the artery walls.

- To have realistic models, accurate methods for *3D reconstruction of the computational domain from medical images* need to be implemented. These images contain noise and artifacts that have to be eliminated, in order to get a realistic topology of the blood circuit.

- Blood cannot be described with the classical theories because it is a *Non-Newtonian fluid*. This means that its viscosity changes with respect to the applied shear stress. However, under certain conditions, the assumption of a Newtonian fluid can be made, such as for low/moderate Reynolds numbers and large vessels.

- Blood is a suspension flow. Its primary purpose is transporting cells: the proportion of blood volume that is occupied by red blood cells is normally between 40 and 50 percent. This implies the implementation of a *multi-scale model*, taking into account these numerous small immersed bodies.

- Finally, there is a *complex chemical cell interaction* between the artery walls and the blood. The artery cells constantly communicate within the wall and their environment. A blood clot can actually happen inside the wall, potentially leading to a large stenosis, which is an abnormal narrowing of a blood vessel. These artery cells also play an important role regarding the inflammation of the tissue that often occurs in arteriosclerosis.

We can see that even if the model was very accurate, the implementation would be difficult, due to double multi-scale phenomena (see Table 1).

Scale	large	small
Space	blood/artery interaction	blood cell motion, chemical interaction
Time	chemical interaction	blood/artery interaction, blood cell motion

Table 1: *Multiscale nature of blood flow.*

Current models only deal with a few of these issues: it is very difficult to deal with both the dynamics and the chemistry of the fluid. In this dissertation, we focus on the dynamical aspect.

- Methods for fluid/flexible-body interaction and more specifically the Immersed Boundary Method (IBM); these methods are applied to the blood flows on two levels: the blood/artery wall interaction and the blood/large cell interaction.
- Fast efficient solvers for the fluid flow computations.
- Geometric extraction of a large blood vessel section, using medical imaging data and the rapid computation of the shear stress and the pressure on the artery wall.

We do not model the chemical interaction and consider blood to be a Newtonian homogeneous fluid, which is relevant to simulation of blood flows with moderate Reynolds number in large vessels.

0.3 On The numerics

The main approaches to simulate fluid flows in complex geometries use either a boundary-fitted mesh or an artificial boundary method such as the IBM, in which

the effect of the boundary is virtually applied to the fluid by a constraint on the equations, or a different local stencil. In the fluid/structure interaction cases, some boundaries are moving, thus the boundary-fitted methods imply a lengthy re-meshing at each time step. This is why the artificial boundary methods are convenient: we can keep the same mesh in all computations. Since the number of grid points is required to be large for fluid computations, and the computational domain is fixed, we choose a mesh that is as regular and uniform as possible, associated with the simple finite difference method. The high level of regularity and symmetry in the mesh, the analogy of the discretization method with the derivative operators and the versatility of the artificial boundary methods, allow the simulations, including the domain decomposition methods, to be implemented quickly.

Since these numerical methods are meant to be used by bio-engineers, we developed our codes in the MATLAB language for its great interactivity potential, visualization and wide choice of ready-to-use numerical functions. However, we have a 3D IBM code based on a NS solver written in FORTRAN [38].

0.4 Outline

The first three chapters are a numerical introduction to the IBM. In Chapter 1, we briefly present the different methods for fluid/flexible-body interaction, starting with the concept of mixing Eulerian and Lagrangian points of view and then artificial boundary methods. Chapter 2 is an introduction to the discretization of the NS equations: first-order projection scheme for the temporal discretization and finite differences for the spacial one. Then, we show some basic test cases, some of which involve fixed immersed bodies. In Chapter 3, we describe the discretization of the IBM, as well as some recent developments in the method.

In Chapter 4, we present a study of the accuracy of the IBM. First, we describe the different discrete Dirac delta functions, which are the key interpolating tool of the method, then we compare them in the framework of elliptic equations with singular source terms, and the IBM. This implies studying the volume conservation property of the IBM as well. Another way to improve the accuracy is the use of extrapolation: we implemented the multigrid/τ-extrapolation technique, which is based on Richardson extrapolation. Finally, we present a method based on a non-centered stencil for the divergence operator to solve Poisson's equation with the divergence of a singularity in the right-end side. This is directly related to the pressure equation of the IBM.

In Chapter 5, we study the stability of the IBM. First, we implemented a fully implicit IBM and show the results regarding stability. Then, we associated another interesting tool with the IBM: Fourier expansions, which save computations and allow the improvement of the stability or the volume conservation property of the method.

Chapter 6 describes the parallel implementation of the Poisson equation solver, using the Aitken acceleration of Schwarz algorithm and the MATLABMPI toolbox. We show the results regarding speedup, efficiency, and associate it with the IBM.

Finally, in Chapter 7, we introduce a versatile incompressible NS solver for blood flow. It includes the geometry extraction of a large blood vessel section, using medical imaging data, rapid computation of the shear stress, and the pressure applied by the flow on the artery wall.

Chapter 1

On Immersed Boundary Simulations

In Fluid Dynamics computations, we usually assume the fluid material to be isotropic and to satisfy the continuum assumption, which states that the properties of the fluid such as velocity, stress, temperature or density are continuous inside the computational domain. But a lot of macroscopic real life cases show strong discontinuities of the fluid, due to different fluid properties or an immersed elastic boundary separating the fluid. These discontinuities are source of physical interactions at the interface, which are difficult to simulate.

We concentrate on sharp interfaces and fluid/elastic-body interaction. In this last case, a thin elastic membrane is immersed in a fluid, exerting an elastic force.

The challenges regarding these simulations are:

- Tracking and describing the geometry of the moving interface.
- Modeling the physical interaction at the interface.
- Coupling the fluid and elastic body equations.

Many engineering issues involve fluid/structure interactions, while many biological cases involve thin immersed elastic boundaries. This is one of the most challenging problems in computational science today, because it involves complex domain geometries, high Reynolds numbers and moving interfaces simultaneously. A lot of research is done on the algorithms for better accuracy, stability, and smaller CPU cost, while more and more scientists apply these new methods to real-life problems.

Let us present a short overview of the methods used for fluid/structure interactions and introduce the immersed boundary simulation techniques.

1.1 The Basic approaches

The first aspect that we need to distinguish between the approaches, is whether or not the mesh is fitting the immersed boundary. This leads to two different families of methods: the boundary-fitted and the non-boundary-fitted methods. In the first family, the fluid computations are made on a boundary-fitted mesh, usually with finite-elements. This implies a complex and lengthy re-meshing at each time step, if the boundary is moving. This is why we focus on the simpler non-boundary-fitted family of methods, which mix Lagrangian and Eulerian computations in an efficient way. First, let us introduce these two different point of view.

1.1.1 The Lagrangian method

Lagrangian methods are traditionally used for solid deformations: the computational mesh follows the same physical particles during motion. Such a method allows us to track the interfaces between different media with great accuracy. The drawback is that each time increment implies a mesh deformation, since each node is moving. Thus, this method is limited to relatively small distortions, in order to keep the structure of the mesh.

1.1.2 The Eulerian method

For Eulerian methods, the computational mesh does not move and the motion of the fluid is described with respect to the fixed grid. Large distortions in the materials can be followed easily by introducing some Lagrangian markers along the moving bodies, but at the cost of accuracy, since the same fixed mesh points describe the fluid and the immersed bodies.

The aim of most fluid/structure interaction methods is to combine the two basic approaches, Lagrangian and Eulerian, in a way that fits the physical case best, keeping the advantages of both. For example, let us present the earlier stage of all the immersed boundary simulations: the Maker-And-Cell (MAC) method.

1.1.3 A Mixed Eulerian-Lagrangian method

One predecessor to mixed Eulerian-Lagrangian methods was the MAC method. This was developed by F.H. Harlow and E. Welch [49] as a variation of the Particle-In-Cell (PIC) [48] method that deals with the dynamics of compressible fluids. The MAC method was the first to treat free surface incompressible flows. While it was made to model liquid/gas and not fluid/structure interactions, it introduced the idea of a combination between Eulerian and Lagrangian methods: the flow motion is described by some Eulerian fields, while some Lagrangian markers are distributed in the liquid phase in order to distinguish the liquid from the gas. The numerical scheme to solve the NS equations associated with the MAC method is very well described in [91]. However, C.W. Hirt [52] showed that the MAC method is unstable with centered momentum advection unless the viscosity is sufficiently large. Its main feature today is the staggered mesh for the Eulerian fields. This mesh allows the use of a more compact finite difference stencil and, when associated with a projection scheme, a

better coupling between the pressure and velocity fields.

Now that we have an idea of how to combine both classical approaches, we come back to the first issue of interface problems, which is tracking and describing the moving interface.

1.2 The Representation of the moving boundary

There are two general ways to describe a moving boundary: explicit or implicit.

1.2.1 The Markers methods

These are explicit ways of describing the interface that can be divided into two categories. We present now examples of both describing ways.

- First technique: we distribute a uniform set of marker points along the interface.
- Second technique: we distribute a few marker points and then fit a curve onto them.

An implicit function can also be used, such as the Level-Set one.

1.2.2 The Level-Set Method

The level-set method is one of the most well-known front-tracking techniques. A level-set function ϕ is used to track the interface, which has to be a closed curve. Let us call X the time-dependent mapping from the curvilinear coordinates s along Γ, the interface, to the Cartesian grid. We define $\phi : \Omega \longrightarrow \mathbb{R}$ such that: $\phi(X(s,t),t) = 0$, $\phi < 0$ inside Γ and $\phi > 0$ outside. The chain rule yields:

$$\phi_t + \nabla\phi.X_t = 0. \tag{1.1}$$

The speed function F is defined in the outward normal direction such as:

$$F = X_t.\eta, \tag{1.2}$$

with η being the outward normal vector: $\eta = \frac{\nabla\phi}{|\nabla\phi|}$.
We can then rewrite (1.1) as:

$$\phi_t + F|\nabla\phi| = 0, \tag{1.3}$$

which is a Hamilton-Jacobi equation for the level-set function. This equation is solved using a standard explicit scheme. The field ϕ is initialized at $t = 0$ with the distance between the Cartesian grid points and Γ:

$$\phi_{t=0} = +d(\mathbf{x}, \Gamma) \text{ if } \mathbf{x} \text{ is inside } \Gamma, \text{ or } \phi_{t=0} = -d(\mathbf{x}, \Gamma).$$

The method automatically captures topological breakage and merger. The drawback is that it is computationally expensive: not only the zero level but all the level-sets have to be evaluated, which represents computations over the whole domain.

Now we give a non-exhaustive list of the immersed boundary simulation techniques. Other general reviews of these methods can be found in [78] and [53].

1.3 The Moving mesh methods

The moving grid methods, also called r-methods, generate unstructured meshes according to the solution of a time-dependent partial differential equation. The mesh points stay concentrated in regions of sharp variation in the solution, such as the moving interface, and their number stay fixed. Basically, two PDE systems are coupled: one for the physical problem and one for the mesh movement itself.

Let us describe briefly the velocity-based moving mesh methods in which the unknowns are the mesh nodes' velocities rather than the mesh nodes' coordinates (location based methods). The most commonly used ones are the Arbitrary Lagrangian-Eulerian (ALE) methods. In this kind of boundary-fitted methods [15, 50], the nodes of the computational mesh can be moved in a Lagrangian way but without following exact physical particle motion. These particles are still globally moving with respect to the mesh. In other words, the mesh velocity is introduced as an additional degree of freedom. It allows us to follow slow deformations of the interface in an accurate way but requires frequent re-meshing. A high-level of accuracy can be reached through this process, even for high Reynolds numbers.

Another interesting velocity-based moving mesh technique is the Geometric Conservation Law method (GCL) [117], whose name is based on the law that implies a cell should not accumulate mass or momentum purely due to its motion.

The main drawbacks of moving mesh methods are that they are difficult to implement and that re-meshing requires a lot of CPU, even if the mesh does not fit exactly the moving boundary. This also implies that they can only model small deformations of the immersed bodies. Recent developments can be found in [109].

1.4 The Artificial boundary methods

One way to avoid re-meshing, is to solve the governing fluid equations on a Cartesian grid and to artificially introduce the effect of the interaction into the fluid variables at the grid points neighboring the moving interface. The mesh is now independent from the interface and the method allows us to follow very large deformations of the interface, described by some Lagrangian coordinates. The challenge is transferring the fields from the Lagrangian to the Eulerian mesh and vice-versa. A broad variety of

methods described in this section bring answers to this issue. Though some methods are difficult to classify, we can basically divide them into four categories:

1. Cut-cell/direct methods: the mesh cells are cut and a special stencil is used near the interface.

2. Immersed boundary techniques: the sharp effect of the interaction is regularized and directly introduced into the fluid equations as a force term; this category includes the IBM.

3. Hybrid Cartesian/Immersed boundary formulations: the force term is derived directly from the discrete momentum equation in order to prescribe the desired boundary conditions.

4. The Finite-Element immersed boundary methods: these are Immersed Boundary techniques with a variational approach.

Let us describe each of these categories. The list of methods in each category is non-exhaustive, the volume of recent publications on fluid/structure interaction being so large.

1.4.1 The Cut-cell/direct methods

The moving boundary is seen as a sharp interface, intersecting some Cartesian mesh cells. These cells are divided into sub-cells, corresponding approximately to the repartition of the different or separated fluids, creating some irregularities in the mesh. Then, an appropriate interpolation technique allows computation of the flow variables around the interface on the irregular mesh, using the boundary conditions. These methods are very well described in the text [118].

There are many ways to cut the cells, due to the large possibility of intersections between the interface and the fixed mesh. The first cell-cut idea, the Coupled Eulerian-Lagrangian method (CEL), was introduced by W.F. Noh [120]. Let us describe the Volume Of Fluid method (VOF), described by W.F. Noh and P.R. Woodward [82], J.D. Ramshaw and J. A. Trapp [96], C.W. Hirt [51] for liquid/gas interactions. The idea is to have a fraction field $F_{\mathbf{i}}(t)$, function of time, that gives the percentage of liquid for the cell **i**:

$$F_{\mathbf{i}}(t) = \begin{cases} 0, & \text{if cell } \mathbf{i} \text{ is full of gas at time } t \\ 1, & \text{if cell } \mathbf{i} \text{ is full of liquid at time } t \\ \alpha, & \text{if cell } \mathbf{i} \text{ has a proportion } \alpha \text{ of liquid at time } t. \end{cases} \tag{1.4}$$

The two ways to cut the cells in the original VOF methods are:

- the Simple Line Interface Construction (SLIC) method (Figure 1.2),
- the Piecewise Linear Interface Construction (PLIC) method (Figure 1.3).

The SLIC consists of dividing the cell with horizontal or vertical lines only, while the PLIC uses straight lines with variable slopes. The VOF method can also be coupled with the Level-Set tracking method, which provides the physical geometry of the interface [106].

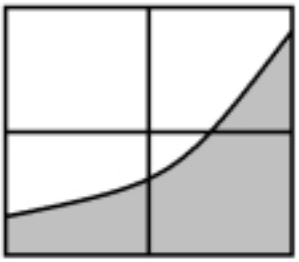


Figure 1.1: *Interface.*

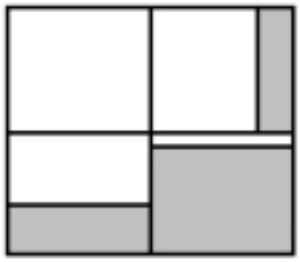


Figure 1.2: *SLIC method.*

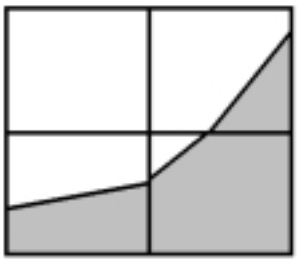


Figure 1.3: *PLIC method.*

A recent development of the cut-cell technique can be found in [47] with the ELAFINT method: Eulerian-Lagrangian Algorithm For Interface Tracking.

Generally, the cut-cell techniques for complex geometries of moving boundaries are complicated to implement, due to special treatment that must be made to the cut cells, depending on the geometry of the sub-cells. The other drawback is that the irregularity of the cell size and geometry introduces conservation and stability issues for the solver. Implementations of fluid flows with immersed boundaries and cut-cell methods can be found in [107, 46, 45], with some cell-merging techniques that decrease the eventual irregularity of the mesh. Second-order accuracy is measured for flows in complex geometries, even in the neighborhood of the boundaries. However, besides the fact that the implementation is difficult, the efficiency is reduced by the need of an iteration scheme to solve the fluid equations with the modified stencil corresponding to the geometrically modified cells, neighboring the interface.

Another approach of directly imposing the boundary conditions of the moving boundary is the finite difference ghost-cell technique. The ghost fluid method was initially developed as an alternative to the Level-Set method for interface fluids, which suffers from spurious oscillations. Ghost cells are cells in a medium that have at least one neighbor across the interface. Then, an interpolation method is introduced using the cells across the media and the value at the boundary. For example, in 2D, bi-linear interpolation can be used, and in 3D, tri-linear interpolation. This linear ghost-cell interpolation works well with low/moderate Reynolds numbers. However, different interpolation techniques have to be implemented with high Reynolds number flows, involving the normal derivatives to the boundary. Using this interpolating scheme, the interface jump conditions can be expressed implicitly, and standard finite difference solvers can be used [94, 31, 95]. This method leads to non-smeared interfaces, applicable to detonation simulation for example.

1.4.2 The Immersed boundary techniques

The first attempt at using such a technique is called Arbitrary Boundary MAC (AB-MAC). This is an extension of the MAC method by J.A. Viecelli [114, 115], dealing with flows and moving boundaries. Some special boundary conditions are introduced to the pressure field at the boundaries, which are free to move or follow a special prescribed motion, so that the flow cannot cross them. This requires a long iteration process at each time step for the boundary conditions to be satisfied under the divergence-free constraint.

C.S. Peskin then developed a more practical method: the Immersed Boundary Method (IBM) [88], to compute 2D blow flows in the heart. A good review of the method can be found in [89]. It uses Dirac delta functions as interpolating tools and markers. The regularized discrete Dirac delta functions allow us to get a force term in the Eulerian mesh from the Lagrangian local force density along the moving boundary, in an interpolation process. This force term is then introduced into the governing equations of the flow, on the fixed grid. Then, after the fluid velocity is computed, the position of the interface is updated using the no-slip boundary condition between the fluid and the interface. Incompressible NS and Elasticity theory can be unified by the same set of equations to get a combined model of the interaction. M. Francois and W. Shyy also incorporated the energy equation into the model [35, 36], to simulate two-phase flows with surface tension at the interface and heat tansfer. However, the theoretical second-order accuracy of the method is actually reduced around the interface because of the use of regularized delta functions. We will detail the method in Chapter 2.

D. Goldstein [19, 20] introduced an extension of the IBM based on feedback forcing, called the Virtual Boundary Method (VBM). It is made for fixed immersed body simulations, within the spectral methods framework. The main drawback is the

constraint on the time stepping in order to prevent spurious oscillations, this makes the method applicable only to quasi-stable flows. Another drawback is that the feedback force term, described in [19], contains two constants that need to be chosen in an empirical way. Let us assume that $V(x,t)$, $(x,t) \in \Omega \times [0,T]$, is a time-dependent field, which needs to satisfy a time-dependent Dirichlet boundary condition $V_\Gamma(x,t)$ along the boundary Γ. Then the force term F_Γ is:

$$F_\Gamma(x,t) = \alpha \int_0^t \left(V(x,\tau) - V_\Gamma(x,\tau)\right) d\tau + \beta \left(V(x,t) - V_\Gamma(x,t)\right), \tag{1.5}$$

with $\alpha \leq 0, \quad \beta \leq 0$. F_Γ is the feedback to $V(x,t) - V_\Gamma(x,t)$ in order to obtain:

$$V(x,t)_{|\Gamma} = V_\Gamma(x,t).$$

The constant values α and β must be large, so that the obstacle reacts faster than the flow and remains fixed. This large force term introduced in the NS equations make the system stiff. E. M. Saiki, S. Biringen and C. Lee [100] extended the same idea and improved the time-stepping issue, using central finite differences and an implicit treatment of the force term. The main limitation of the original VBM is that the interface needs to coincide with the Eulerian mesh nodes.

In [3], A.L.F. Lima E Silva and co-workers extended the VBM. It is implemented with finite differences but without using any constant in the force term and without any constraint on the immersed boundary geometry. The interpolation used to compute the fluid velocity and pressure at the marker points is a second-order Lagrange polynomial approximation. The spreading of the force is made using classical delta functions. This is shown to behave well at high Reynolds numbers in the benchmark problem of a flow past a cylinder.

Another interesting Immersed Boundary technique is the Blob Projection method from R. Cortez and M. Minion [13]. This is based on the Vortex and Impulse methods

[6, 12, 14, 13]. Built in the framework of 2D finite differences, a cutoff (blob) function is used to regularize the force field when the projection operator is applied. Essentially, the projection of the delta functions is computed analytically, which reduces the error smearing. This blob projection demonstrates a better accuracy than the IBM, but is limited by the condition of periodicity of the domain in order to have linearity of the projection operator. This reduces the possibility of domain geometries, while its implementation in not as simple as the IBM.

The Immersed Interface Method (IIM) developed by R.J. Leveque and L. Lee [66, 67] is a clever variation of the IBM: the analytical pressure jump is explicitly incorporated in the solver rather than using the divergence of discrete Dirac delta functions. The pressure field is computed using a split local force density:

$$f = f_\tau + f_\eta, \tag{1.6}$$

where f_τ and f_η are its normal and tangential components to the interface. The pressure jump condition is:

$$[\![P]\!](s,t) = \frac{f_\eta(s,t)}{\partial X(s,t)/\partial s}, \tag{1.7}$$

where $[\![P]\!]$ is the pressure jump across the interface and $X(s,t)$, the position vector of the interface.

However, this method is limited to sharp boundaries with no volume. The other drawbacks are that it requires the normal and tangential components of the force density, a special discretization of the NS equations near the interface, and an implicit resolution of a system for the movement of the boundary at each time step. Nevertheless, this allows us to have a good volume conservation property and actual uniform second-order accuracy.

1.4.3 The Hybrid Cartesian/Immersed Boundary (HCIB) formulations

In [57], J. Mohd-Yusof derives the force term directly from the momentum equation, then, B-splines are used for better distribution of the grid points over the domain. Stability of the scheme is no longer reduced by the force term and there is no special parameter to set up empirically. Essentially, the discretized momentum equation is modified in order to impose the boundary conditions at the boundary points at each time step. If we write a simplified momentum equation:

$$\frac{V^{n+1} - V^n}{\Delta t} = RHS(V^n) + F^n \tag{1.8}$$

and assume that the boundary coincides with the mesh nodes, we have at the boundary Γ:

$$F^n_{|\Gamma} = \frac{V^{n+1}_{|\Gamma} - V^n_{|\Gamma}}{\Delta t} - RHS(V^n)_{|\Gamma}. \tag{1.9}$$

Now if we impose $V^{n+1}_{|\Gamma}$ to satisfy the Dirichlet boundary condition $V_{|\Gamma}$, we get this expression of the direct force term:

$$F^n_{|\Gamma} = \frac{V_{|\Gamma} - V^n_{|\Gamma}}{\Delta t} - RHS(V^n)_{|\Gamma}. \tag{1.10}$$

$F^n = 0$ outside of Γ. If we plug (1.10) back into (1.8), we get, at the interface:

$$V^{n+1}_{|\Gamma} = V_{|\Gamma}.$$

This method is called direct or momentum forcing and shows second-order accuracy when applied to fixed interfaces.

Based on the idea of momentum forcing, Y.-H. Tseng and J. H. Ferziger presented an efficient Ghost-Cell Immersed Boundary Method (GCIBM) [112] for simulating turbulent flows in fixed complex geometries. The ghost-cell technique is used to

interpolate the force term when the interface does not coincide with the Eulerian mesh points.

A direct Immersed Boundary Method has been developed by J. Kim [54] and has recently been extended to moving boundaries [60], based on mass source/sink as well as momentum forcing, in the framework of finite-volume, staggered-mesh and fractional step schemes. A similar method was implemented by E. A. Fadlun [25]. These two techniques use a second-order interpolation scheme for evaluating the momentum force on the Eulerian grid. The interpolation of the force over the grid determines the accuracy of the scheme and can lead to force oscillations when moving boundaries are involved [80].
In [5], 2D and 3D large-eddy simulations of flows in complex boundaries are made on fixed Cartesian grids with finite differences. A new interpolation scheme is introduced in order to reach the second-order accuracy, even at the artificial boundaries position.

A. Gilmanov introduced another HCIB method for moving immersed bodies in [39]. The sharp interface is discretized with an unstructured triangular mesh while the solution around it, is reconstructed using a quadratic interpolation, along the normal body direction, easy to compute with the triangular elements. Second-order accuracy is achieved in 3D and a new hybrid staggered/non-staggered grid is introduced as well.

A direct forcing variant of the immersed boundary method is presented in [116] by A. Vikhansky. It is based on the original method from E. A. Fadlun and J. Mohd-Yusof [25], using the same interpolation technique, associated with a variational principle. This allows us to satisfy the boundary conditions using the points near the interface. The interaction between the fluid and the rigid-body is described by the principle of virtual work. This leads to a set of unconstrained minimization problems.

Finally, in [113], M. Uhlmann exports the IBM regularized Dirac delta function into a direct formulation of the force, for fluid/moving-solid interactions. The transfer between Eulerian and Lagrangian representations is, then, smoother and the method more stable.

1.4.4 The Finite-Element immersed boundary methods

Recent efforts have been made to extend the IBM to finite-element discretization. W.K. Liu [64, 119, 121] introduced the Immersed Finite Element Method (IFEM) to simulate fluid/deformable-structure interactions. This method allows us to have immersed elastic structures with complex physical behaviors and a volume. It uses two meshes: one Lagrangian mesh for the structure and one Eulerian mesh for the fluid that covers the computational domain. Both the structure and the fluid are modeled using the finite-element method, and the interpolation from one mesh to an other is made using the Reproducing Kernel Particle Method (RKPM) delta function. This method is harder to implement than the IBM but is uniformly second-order accurate, even for the simulation of large solid deformations.

A recent, similar approach has been done by D. Boffi, L. Gastaldi and L. Heltai with the Finite Element Immersed Boundary Method (FEIBM) [16, 17, 18]. However, unlike the IFEM, there are no delta functions in the FEIBM. The advantage using the finite-element method is that the delta functions can be handled with the variational theory, without any regularization technique.

To sum up, the advantages of the artificial boundary techniques are that they are easier to implement than the moving mesh methods and that they cost less CPU-time and memory. The first drawback is that they require an interpolation process or a change of stencil around the interface, difficult to implement or to extend to 3D

and not always accurate. This can be avoided using the finite-element method but with the cost of a more difficult implementation. The second drawback is that they are usually unstable, due to a stiff NS system.

A similar approach to simulate fluid flows in complex geometries has been recently developed. It is called the fictitious domain or domain embedding methods.

1.5 Fictitious Domain Methods

It is possible to group most of the artificial boundary methods listed above into the theoretical set of Fictitious Domain (FD) techniques: the complex fluid domain is extended into a larger one that is more regular. Then, the boundary conditions are imposed using a constraint technique. The force term of the IBM is actually a constraint in a FD technique, to create the force into the fluid flow, as R. Glowinsky showed in [92].
The FD methods can be divided into three categories, depending on what the constraint is based upon:

- 1- Non body force based.
- 2- Body force based (but not Distributed Lagrange Multiplier (DLM) based).
- 3- DLM based.

The IBM actually belongs to the second category.
Now, in general, the constraint can be created numerically using two approaches:

- Penalty method.
- Duality method.

The Penalty FD method will be described in detail in Chapter 7. Let us only explain the general ideas behind these two methods. If we want to minimize a function $f(x)$ with $x \in \mathbb{R}^n$, under the constraint: $c(x) = 0$, we minimize the functions

$$F(x,k) = f(x) + k\left(c(x)\right)^2,$$

with k going to infinity, using the penalty method and

$$F(x,k) = f(x) + kc(x),$$

using the duality method. This last category leads to the DLM method pioneered by R. Glowinski. It is used to simulate particulate flows [92] or flows past fixed solid bodies [93]. The method is very well described in [41]. The Lagrange multiplier is seen as a pseudo body force, enforced into the interior of the immersed body, the fictitious fluid, to satisfy the constraint of a rigid body motion. The FD methods have proved to be efficient for the numerical treatment of domains containing moving bodies. However, the method cannot be used to simulate the interaction between a fluid and a flexible body with large deformations. F.P.T. Baaijens [4] and Zhaosheng Yu [123] extended the method to general fluid/flexible-body interactions. The drawback of F.P.T. Baaijens' method is the assumption that the immersed body has no inertia or local force density. The method implemented in [123] by Zhaosheng Yu consists of dividing the problem into three sub-sets of equations:

- The motion equations for the fluid.
- The deformation equations for the flexible solid.
- The FD/DLM constraint equations of R. Glowinski et Al. [92] for the immersed bodies.

However, the method suffers from some convergence issues. Zhaosheng Yu writes in [123]: "[...] the accuracy of our method is acceptable, despite the relatively poor

convergence behavior with the solid mesh, which is the main source of numerical errors in our scheme and, however, is mainly caused by the difficulty in solving the solid problem alone."

1.6 The Challenge regarding the IBM

To summarize, we want to avoid the moving mesh methods and look into four different categories of artificial boundary methods: the cut-cell methods, the immersed boundary techniques, the hybrid Cartesian/immersed boundary formulations, and the finite-element based immersed boundary techniques.
Within these categories, the hybrid Cartesian/immersed boundary formulations are limited to small deformations or associated with complex interpolation techniques or stencil modification. This last drawback applies to the cut-cell methods as well.
With most of the applications of fluid/flexible-body interaction being in bio-engineering, we need a method that is easy to implement, efficient, and versatile: it should allow large deformations, elastic boundaries with mass and/or volume, as well as diverse complex flow geometries. These criteria, all together, direct us toward the IBM: this is the simplest method to implement and one of the most versatile.
However, the IBM is penalized by two things:

- Its first-order accuracy along the interface due to the use of discrete Dirac delta functions.
- Its instability due to the sharp force term that makes the NS equations system stiff.

This is why we study the accuracy of the IBM in Chapter 4 and the stability in Chapter 5. The other issue is efficiency. We focus on this in Chapter 6, by studying a parallel implementation of the IBM.

First, we recall the computation of the incompressible NS with finite differences, which is the framework of the IBM.

Chapter 2

Numerical introduction to Navier-Stokes computations with finite differences

To be complete, here is a brief introduction to the incompressible NS equations, in primitive variable, with their discretization in time and space, as we use in the IBM. For the spacial discretization, we used the staggered mesh in order to have smaller differential stencils. For the temporal discretization, we used different first or second order projection schemes, but we only present here the basic fully explicit first order projection scheme, for simplicity. Then we introduce some basic test cases and implementations of the direct forcing method. Finally, we describe the solver used and give the computational times for this basic NS implementation in MATLAB and compare it with the same implementation done in FORTRAN.

To simplify the presentation, we will restrict this introduction to the two-dimensional case.

2.1 The Incompressible Navier-Stokes equations

The NS equations are partial differential equations that describe the flow of incompressible fluids. In the primitive variable expression, we describe the flow with V, the velocity vector field, and P, the pressure field. The parameters of the fluid are the viscosity μ and the density ρ. The incompressible NS equations are:

$$\rho\left[\frac{\partial V}{\partial t} + (V.\nabla)\,V\right] = \mu\Delta V - \nabla P + F \tag{2.1}$$

$$\nabla.V = 0 \tag{2.2}$$

F is the external force term, applied to the fluid. See the appendix for the derivation of these equations.

Now we present one of the classical schemes to approximate the solution to this system of PDEs.

2.2 The Projection method

The Projection method allows the derivation of a simple scheme to compute numerical solutions of the system. We consider a closed rectangular physical domain Ω.

2.2.1 The Hodge decomposition

Let us start by evoking briefly the Hodge decomposition. Given the vector field W defined in Ω, such that $W.\eta = 0$ on $\partial\Omega$, η is the outward normal vector to the domain boundary, there exists a unique decomposition of W into two parts:

$$W = W_{df} + \nabla S, \tag{2.3}$$

where W_{df} is a divergence-free vector field and S, a scalar function. If we take the divergence of (2.3), we get:

$$\nabla.W = \Delta S, \tag{2.4}$$

since $\nabla.W_{df} = 0$ by definition. Another property of this decomposition is that W_{df} and S are orthogonal:

$$\int_{\Omega} W_{df}.\nabla S dv = 0. \tag{2.5}$$

The idea of the projection method is to have a transformation from W to W_{df}. Let us call $\wp$, the linear projection operator to the divergence-free vector space, we have:

$$W_{df} = \wp\left[W\right]. \tag{2.6}$$

We can actually describe the operator $\wp$. From (2.4), we have $S = \left[\Delta^{-1}\nabla\right]W$. If we plug that back into (2.3), we get $W_{df} = W - \left[\nabla\Delta^{-1}\nabla\right]W$. This leads to the analytic definition of the projection operator:

$$\wp = I - \nabla\Delta^{-1}\nabla. \tag{2.7}$$

2.2.2 The Projection of the explicit Navier-Stokes equations

If we come back to the NS equations in the rectangular domain Ω and apply the projection operator to the momentum equation, we have:

$$\wp\left[\rho\frac{\partial V}{\partial t}\right] + \wp\left[\nabla P\right] = \wp\left[-\rho\left(V.\nabla\right)V + \mu\Delta V + F\right]. \tag{2.8}$$

Since the velocity field has to be divergence-free and the projection of a gradient is null, (2.8) becomes:

$$\rho\frac{\partial V}{\partial t} = \wp\left[-\rho\left(V.\nabla\right)V + \mu\Delta V + F\right]. \tag{2.9}$$

By plugging (2.9) into the momentum equation (2.1), we obtain an expression for the pressure gradient depending on the projection $\wp$:

$$\nabla P = (I - \wp)\left[-\rho\left(V.\nabla\right)V + \mu\Delta V + F\right], \tag{2.10}$$

where I is the identity operator. This projection method introduced independently by A. J. Chorin [10] and R. Teman [108] is a way to deal with the difficulty of the incompressible NS equations, that is, to compute both the velocity field and the pressure scalar function at the same time, using only one constrained equation. To summarize, the velocity is updated thanks to a divergence-free projection, while what is left after the projection is used to update the pressure. Everything is based on the fact that the momentum equation is a Hodge decomposition of

$$-\rho\left(V.\nabla\right)V + \mu\Delta V + F$$

into the sum of the divergence-free vector field $\rho\frac{\partial V}{\partial t}$ and the pressure gradient ∇P.

Now for the basic projection method introduced by M. Fortin [68], we start from the explicit momentum equation (2.1):

$$\rho\left[\frac{V^{n+1} - V^n}{\Delta t} + \left(V^n.\nabla\right)V^n\right] = \mu\Delta V^n - \nabla P^{n+1} + F^n \tag{2.11}$$

$$\nabla.V^{n+1} = 0. \tag{2.12}$$

We can split (2.11) to get this fractional-step method:

$$V^* = V^n + \Delta t\left[-\left(V^n.\nabla\right)V^n + \frac{1}{\rho}\left(\mu\Delta V^n + F^n\right)\right], \tag{2.13}$$

$$V^{n+1} = V^* - \frac{\Delta t}{\rho}\nabla P^{n+1}. \tag{2.14}$$

We can observe that (2.14) is a Hodge decomposition of V^*: the divergence-free vector field is V^{n+1} and the scalar function is $\frac{\Delta t}{\rho}P^{n+1}$. Using the projection operator $\wp$, we get:

$$V^{n+1} = \wp\left[V^*\right], \tag{2.15}$$

$$\frac{\Delta t}{\rho}\nabla P^{n+1} = V^* - V^{n+1} = \left(I - \wp\right)\left[V^*\right]. \tag{2.16}$$

We rewrite (2.16) and take its divergence:

$$\nabla.\nabla P^{n+1} = \frac{\rho}{\Delta t}\left(\nabla.V^* - \nabla.V^{n+1}\right) = \frac{\rho}{\Delta t}\nabla.V^*, \tag{2.17}$$

so that we get the basic projection scheme:

1- Velocity prediction

$$V^* = V^n + \Delta t\left[-\left(V^n.\nabla\right)V^n + \frac{1}{\rho}\left(\mu\Delta V^n + F^n\right)\right], \tag{2.18}$$

2- Pressure equation

$$\Delta P^{n+1} = \frac{\rho}{\Delta t}\nabla.V^*, \tag{2.19}$$

3- Velocity correction

$$V^{n+1} = V^* - \frac{\Delta t}{\rho}\nabla P^{n+1}. \tag{2.20}$$

We will later detail the boundary conditions. This scheme is a first order method

since the time discretization is explicit:

$$\frac{V^{n+1} - V^n}{\Delta t} = F(V^n, t^n) + O(\Delta t). \tag{2.21}$$

An extensive number of second-order projection methods have been developed in the past two decades [21]. We used one of them in many cases. They are usually as easy to implement as the first-order projection scheme, except that special care has to be taken on the pressure boundary conditions, which are highly related to the type of projection.

Now that we have a basic time discretization, we describe the space discretization, in the framework of finite difference and staggered grids rather than collocated, where all the mesh nodes for u, v, p are located at the same place within the cells.

2.3 The Staggered mesh

This type of mesh allows us to carry more information about the fluid than the collocated one, with the same amount of data. The other main advantage is related to the pressure solver in the basic projection scheme for the NS equations. As we will see, smaller stencils for the divergence and Laplace operators can be used in the pressure equation, which reduces the smearing effect in the singular field cases.

2.3.1 Location of the nodes

Let us consider a rectangular domain $\Omega = [0, L_x] \times [0, L_y]$ and a uniform mesh with respective space steps $h_x = \frac{L_x}{N_x - 1}$ and $h_y = \frac{L_y}{N_y - 1}$, so that the mesh nodes have the coordinates:

$$(ih_x, jh_y), \text{ with } 0 \leq i \leq N_x - 1 \text{ and } 0 \leq j \leq N_y - 1.$$

We use staggered locations for u and v, the horizontal and vertical velocity components and the pressure p, as shown in Figure 2.1.

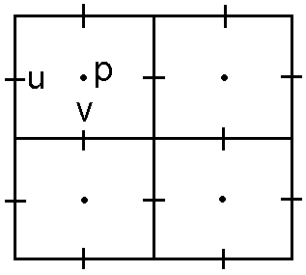


Figure 2.1: *2D staggered mesh.*

The locations of the respective arrays are:

- u: $\left(ih_x, \left(j+\frac{1}{2}\right)h_y\right)$ with $0 \leq i \leq N_x - 1$ and $0 \leq j \leq N_y - 2$,
- v: $\left(\left(i+\frac{1}{2}\right)h_x, jh_y\right)$ with $0 \leq i \leq N_x - 2$ and $0 \leq j \leq N_x - 1$,
- p: $\left(\left(i+\frac{1}{2}\right)h_x, \left(j+\frac{1}{2}\right)h_y\right)$ with $0 \leq i \leq N_x - 2$ and $0 \leq j \leq N_y - 2$.

2.3.2 A Smaller stencil

The first idea of the staggered mesh, introduced by F. H. Harlow and E. Welch [49] is that we gather more information about the fluid by describing it at different locations within each mesh cell, without increasing the size of the arrays compared to a regular mesh, where both velocity and pressure are evaluated at the same points. The second idea is related to the pressure correction equation of the projection scheme used to solve the NS equations:

$$\Delta P = \nabla . V$$

We need to evaluate the divergence of the velocity at first, but the stencil used for the Laplace operator needs to be wider than the stencil used for the divergence operator, in order to have good spacial propagation of the information. With the staggered

mesh, we get second-order accuracy for the divergence operator, using only a four-point spacial stencil of radius $\frac{h_x}{2}$ in the x-direction and $\frac{h_y}{2}$ in the y-direction, so that we can use the classic five-point stencil for the Laplace operator, of radius h_x in the x-direction and h_y in the y-direction (see Figure 2.2).

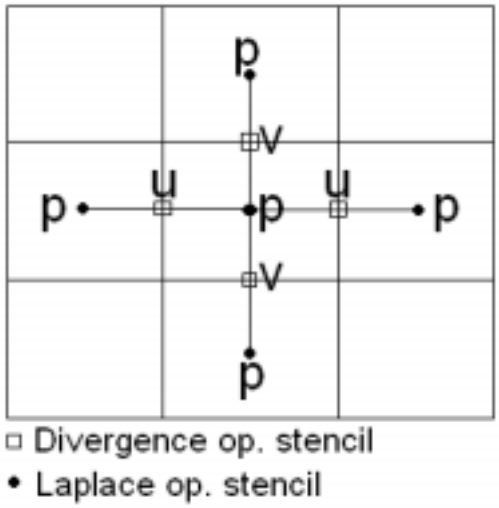


Figure 2.2: *2D finite difference stencils for the Laplace and divergence operators using a staggered mesh.*

2.4 The Finite difference operators

For all the operators except for the non-centered cases, the numerical error is proportional to the square of the space step. The operators used in the NS equations are:

- divergence,
- gradient,
- Laplace.

2.4.1 The Divergence operator

The divergence is evaluated at the center of the mesh cells:

$$D_h V_h = \left\{ \frac{u_{i+1,j+\frac{1}{2}} - u_{i,j+\frac{1}{2}}}{h_x} + \frac{v_{i+\frac{1}{2},j+1} - v_{i+\frac{1}{2},j}}{h_y} \right\}_{0 \le i \le N_x - 2, 0 \le j \le N_y - 2}. \tag{2.22}$$

The discrete divergence-free condition of the NS equations is then: $D_h V_h = 0$.

2.4.2 The Gradient operator

The discrete gradient operator is evaluated at the center of the mesh cells too:

$$G_h V_h = \left\{ \left[\frac{u_{i+1,j+\frac{1}{2}} - u_{i,j+\frac{1}{2}}}{h_x}, \frac{v_{i+\frac{1}{2},j+1} - v_{i+\frac{1}{2},j}}{h_y} \right]^T \right\}_{0 \le i \le N_x - 2, 0 \le j \le N_y - 2}. \tag{2.23}$$

2.4.3 The Laplace operator

Horizontal discrete second derivative operator:

$$\left(\mathbf{D}_x^2 w\right)_{i,j} = \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h_x^2}. \tag{2.24}$$

Vertical discrete second derivative operator:

$$\left(\mathbf{D}_y^2 w\right)_{i,j} = \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{h_y^2}. \tag{2.25}$$

These operators, $\mathbf{D}_x^2$ and $\mathbf{D}_y^2$, are second-order finite difference approximations of the horizontal and vertical second derivatives. Now, the Laplace operator with a staggered mesh, is:

$$\begin{aligned} \left(\Delta_h u\right)_{i,j+\frac{1}{2}} &= \left(\mathbf{D}_x^2 u\right)_{i,j+\frac{1}{2}} + \left(\mathbf{D}_y^2 u\right)_{i,j+\frac{1}{2}} \\ &= \frac{1}{h_x^2}\left(u_{i+1,j+\frac{1}{2}} - 2u_{i,j+\frac{1}{2}} + u_{i-1,j+\frac{1}{2}}\right) \\ &+ \frac{1}{h_y^2}\left(u_{i,j+\frac{3}{2}} - 2u_{i,j+\frac{1}{2}} + u_{i,j-\frac{1}{2}}\right), \end{aligned} \tag{2.26}$$

$$0 < i < N_x - 1, \;\; 0 < j < N_y - 2.$$

$$\begin{aligned} (\Delta_h v)_{i+\frac{1}{2},j} &= \left(\mathbf{D}_x^2 v\right)_{i+\frac{1}{2},j} + \left(\mathbf{D}_y^2 v\right)_{i+\frac{1}{2},j} \\ &= \frac{1}{h_x^2}\left(v_{i+\frac{3}{2},j} - 2v_{i+\frac{1}{2},j} + v_{i-\frac{1}{2},j}\right) \\ &+ \frac{1}{h_y^2}\left(v_{i+\frac{1}{2},j+1} - 2v_{i+\frac{1}{2},j} + v_{i+\frac{1}{2},j-1}\right), \\ 0 < i < N_x - 2, \;\; 0 < j < N_y - 1. \end{aligned} \tag{2.27}$$

2.4.4 The Convective part

Convection is defined by:

$$(V.\nabla)\, V = \begin{cases} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \end{cases} \tag{2.28}$$

First, we need $\tilde{u}_{i+\frac{1}{2},j}$ and $\tilde{v}_{i,j+\frac{1}{2}}$, approximations of u at $\{x_{i+\frac{1}{2}}, y_j\}$ and of v at $\{x_i, y_{j+\frac{1}{2}}\}$.

$$\tilde{u}_{i+\frac{1}{2},j} = \frac{1}{4}\left(u_{i,j+\frac{1}{2}} + u_{i+1,j+\frac{1}{2}} + u_{i,j-\frac{1}{2}} + u_{i+1,j-\frac{1}{2}}\right) \tag{2.29}$$

$$\tilde{v}_{i,j+\frac{1}{2}} = \frac{1}{4}\left(v_{i+\frac{1}{2},j} + v_{i+\frac{1}{2},j+1} + v_{i-\frac{1}{2},j} + v_{i-\frac{1}{2},j+1}\right)$$

2.4.4.1 The Centered approximation

This approximation is:

$$conv^x_{i,j+\frac{1}{2}} = u_{i,j+\frac{1}{2}}\left(\frac{u_{i+1,j+\frac{1}{2}} - u_{i-1,j+\frac{1}{2}}}{2h_x}\right) + \tilde{v}_{i,j+\frac{1}{2}}\left(\frac{u_{i,j+\frac{3}{2}} - u_{i,j-\frac{1}{2}}}{2h_y}\right) \tag{2.30}$$

$$conv^y_{i+\frac{1}{2},j} = \tilde{u}_{i+\frac{1}{2},j}\left(\frac{v_{i+\frac{3}{2},j} - v_{i-\frac{1}{2},j}}{2h_x}\right) + v_{i+\frac{1}{2},j}\left(\frac{v_{i+\frac{1}{2},j+1} - v_{i+\frac{1}{2},j-1}}{2h_y}\right)$$

Let us study the stability of this approximation, with the one-dimensional transport equation:

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0, \quad v \in \mathbb{R}. \tag{2.31}$$

With a scheme forward in time and centered in space, we have:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + v\frac{u_{j+1}^n - u_{j-1}^n}{h} = 0. \tag{2.32}$$

A Von Neumann stability analysis shows that this scheme is unstable. We write the solution u as a sinusoidal perturbation of wave-number k and growth rate $r \in \mathbb{C}$:

$$u(x,t) = e^{rt+Ikx}, \quad I = \sqrt{-1}. \tag{2.33}$$

After discretization and using the notation $U = e^{r\Delta t}$ for the amplitude of the perturbation, we get: $u_j^n = U^n e^{Ikjh}$. By plugging this expression into Eq. (2.32) and simplifying, we have:

$$U = 1 - \frac{v\Delta t}{2h}\left(e^{Ikh} - e^{-Ikh}\right) = 1 - I\frac{v\Delta t}{h}sin(kh). \tag{2.34}$$

Then we evaluate the norm of U:

$$|U|^2 = 1 + \left[\frac{v\Delta t}{h}sin(kh)\right]^2 > 1. \tag{2.35}$$

Thus this discretization of the convection is unconditionally unstable. The scheme becomes stable only after adding the diffusive part, and under time step restriction.

2.4.4.2 The Upwind approximation

Another standard discretization of the convective part is the Upwind scheme. The idea is that if the flux is moving in a certain direction, the incoming flux should only depend on the concentration upstream, that is:

$$v\frac{\partial u_j^n}{\partial x} = -v \begin{cases} \frac{u_j^n - u_{j-1}^n}{h}, & \text{if } v > 0 \\ \frac{u_{j+1}^n - u_j^n}{h}, & \text{if } v < 0 \end{cases}$$

For the transport equation, we get:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v \begin{cases} \frac{u_j^n - u_{j-1}^n}{h}, & \text{if } v > 0 \\ \frac{u_{j+1}^n - u_j^n}{h}, & \text{if } v < 0 \end{cases}$$

This is easy to code this way:

$$\begin{aligned} conv_{i,j+\frac{1}{2}}^x = max(u_{i,j+\frac{1}{2}}, 0)\left(\frac{u_{i,j+\frac{1}{2}} - u_{i-1,j+\frac{1}{2}}}{h_x}\right) + \\ min(u_{i,j+\frac{1}{2}}, 0)\left(\frac{u_{i+1,j+\frac{1}{2}} - u_{i,j+\frac{1}{2}}}{h_x}\right) + \\ max(\tilde{v}_{i,j+\frac{1}{2}}, 0)\left(\frac{u_{i,j+\frac{1}{2}} - u_{i,j-\frac{1}{2}}}{h_y}\right) + min(\tilde{v}_{i,j+\frac{1}{2}}, 0)\left(\frac{u_{i,j+\frac{3}{2}} - u_{i,j+\frac{1}{2}}}{h_y}\right) \end{aligned} \tag{2.36}$$

$$\begin{aligned} conv_{i+\frac{1}{2},j}^y = max(\tilde{u}_{i+\frac{1}{2},j}, 0)\left(\frac{v_{i+\frac{1}{2},j} - v_{i-\frac{1}{2},j}}{h_x}\right) + \\ min(\tilde{u}_{i+\frac{1}{2},j}, 0)\left(\frac{v_{i+\frac{3}{2},j} - v_{i+\frac{1}{2},j}}{h_x}\right) + \\ max(v_{i+\frac{1}{2},j}, 0)\left(\frac{v_{i+\frac{1}{2},j} - v_{i+\frac{1}{2},j-1}}{h_y}\right) + min(v_{i+\frac{1}{2},j}, 0)\left(\frac{v_{i+\frac{1}{2},j+1} - v_{i+\frac{1}{2},j}}{h_y}\right) \end{aligned} \tag{2.37}$$

We perform the Von Neumann stability analysis to this scheme:

$$\begin{aligned} \frac{U^{n+1}e^{Ikhj} - U^n e^{Ikhj}}{\Delta t} = -max(v,0)\frac{U^n e^{Ikhj} - U^n e^{Ikh(j-1)}}{h} \\ -min(v,0)\frac{U^n e^{Ikh(j+1)} - U^n e^{Ikhj}}{h}. \end{aligned} \tag{2.38}$$

And we get, after simplification:

$$U = 1 - \frac{\Delta t}{h}\left[max(v,0) - min(v,0) - max(v,0)e^{-Ikh} + min(v,0)e^{Ikh}\right]. \tag{2.39}$$

Since

$$\frac{v}{2} + \frac{|v|}{2} = max(v, 0),$$

and

$$\frac{v}{2} - \frac{|v|}{2} = min(v, 0),$$

we get:

$$U = 1 - \frac{|v|\Delta t}{h}(1 - cos(kh)) - I\frac{v\Delta t}{h}sin(kh), \tag{2.40}$$

and:

$$|U|^2 = \left[1 - \frac{|v|\Delta t}{h}(1 - cos(kh))\right]^2 + \left(\frac{v\Delta t}{h}(1 - cos(kh))\right). \tag{2.41}$$

The stability criterion is:

$$\frac{|v|\Delta t}{h} \leq 1.$$

This is the famous Courant-Friedrichs-Lewy stability criterion, $\frac{|v|\Delta t}{h}$ is the Courant number which is simply the number of grid points travelled by the fluid in a single time step. If the time step is too large and the fluid is going faster than the numerical propagation, instability is created.

With this upwind approximation, we get a better behavior of the model for high Reynolds numbers.

2.4.4.3 The Skew symmetric approximation

We can re-write (2.27) in order to get this skew-symmetric expression of the convection, using the incompressibility condition:

$$conv^x = \frac{1}{2}(V.\nabla u + \nabla.(Vu)) \tag{2.42}$$

$$conv^y = \frac{1}{2}\left(V.\nabla v + \nabla.(Vv)\right)$$

After discretization, this new expression induces conservation of kinetic energy, using integration by parts and periodic boundary conditions, which is important for long time integration:

$$\frac{1}{2}\frac{\partial \|V^T V\|_h^2}{\partial t} = 0.$$

In a turbulent case, it is known to give more accurate results.

2.4.4.4 The Method of Characteristics

The method of characteristics is used to solve initial value problem for general first order Partial Differential Equations (PDE). It consists of changing the regular coordinates (x, t) for space and time, into new ones $(\bar{x}, \bar{t})$ for which the PDE become an ordinary differential equation along specific curves of the space. These curves

$$\left\{x\left(\bar{x}, \bar{t}\right), t\left(\bar{x}, \bar{t}\right) / \bar{t} > 0\right\}$$

for fixed $\bar{x}$, are called the characteristic curves of the PDE. If we come back to the one-dimensional transport equation:

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0, \quad v \in \mathbb{R},$$

the characteristic curves are:

$$\frac{\partial x}{\partial \bar{t}} = v, \tag{2.43}$$

$$\frac{\partial t}{\partial \bar{t}} = 1. \tag{2.44}$$

We get:

$$\frac{\partial u}{\partial \bar{t}} = \frac{\partial t}{\partial \bar{t}}\frac{\partial u}{\partial t} + \frac{\partial x}{\partial \bar{t}}\frac{\partial u}{\partial x} = 0. \tag{2.45}$$

The method consists of solving Eq.(2.43) and (2.44), in order to have the transformation from (x, t) to $(\bar{x}, \bar{t})$:

$$x = x(\bar{x}, \bar{t}),$$

$$t = t(\bar{x}, \bar{t}).$$

Then, we solve the ordinary differential Eq.(2.45) and get a solution in terms of $\bar{x}$ and $\bar{t}$. Finally, we use the inverse transformation from $(\bar{x}, \bar{t})$ to (x, t), to get the solution $u(x, t)$. This method is easy to implement, relatively accurate and stable, while being known to have good conservative properties.

2.5 The Boundary conditions

Let us assume that the two-dimensional fluid is in a closed box $\Omega = [0, L_x] \times [0, L_y]$, which means that there is a no-slip boundary condition on each side of the box:

$$u(0, y) = u(L_x, y) = v(0, y) = v(L_x, y) = 0 \quad \text{for} \quad 0 \leq y \leq L_y, \tag{2.46}$$

$$u(x, 0) = u(x, L_y) = v(x, 0) = v(x, L_y) = 0 \quad \text{for} \quad 0 \leq x \leq L_x. \tag{2.47}$$

2.5.1 The Extended staggered mesh

In order to apply these boundary conditions to the discrete solution, we use an extended mesh: each field gets extra ghost points across the boundary, outside the physical domain and only on the sides where the discrete nodes do not lie on the boundary. The sides are for u: $y = 0$, $y = L_y$ and for v: $x = 0$, $x = L_x$. The four sides are extended for the pressure field since the discretized points are inside the mesh cells.

The new sizes of the arrays are:

- u: $N_x \times (N_y + 1)$,

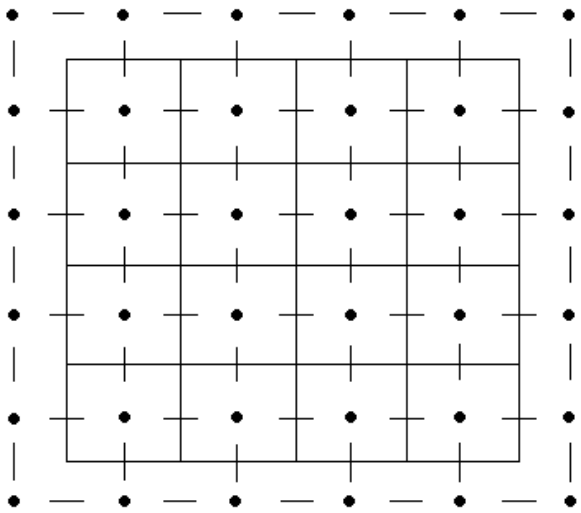


Figure 2.3: *Staggered extended mesh. The horizontal segments indicate the location of u_h, the vertical ones v_h and the dots p_h.*

- v: $(N_x + 1) \times N_y$,
- p: $(N_x + 1) \times (N_y + 1)$.

It is now easy to express the conditions (2.46) and (2.47) with a simple reflection technique:

- $u_{i,0} = 2u(x_i, 0) - u_{i,1}$ with $0 \leq i \leq N_x$,
- $u_{i,N_y+1} = 2u(x_i, L_y) - u_{i,N_y}$ with $0 \leq i \leq N_x$,
- $v_{0,j} = 2v(0, y_j) - v_{1,j}$ with $0 \leq j \leq N_y$,
- $v_{N_x+1,j} = 2u(L_x, y_j) - v_{N_x,j}$ with $0 \leq j \leq N_y$,

with $x_i = ih_x$ and $y_j = jh_y$, or, in order to achieve better accuracy:

- $u_{i,0} = \frac{1}{3}\left(8u(x_i, 0) - 2u_{i,1} + u_{i,2}\right)$ with $0 \leq i \leq N_x$,
- $u_{i,N_y+1} = \frac{1}{3}\left(8u(x_i, L_y) - 2u_{i,N_y} + u_{i,N_y-1}\right)$ with $0 \leq i \leq N_x$,

- $v_{0,j} = \frac{1}{3}\left(8v(0, y_j) - 2v_{1,j} + v_{2,j}\right)$ with $0 \leq j \leq N_y$,
- $v_{N_x+1,j} = \frac{1}{3}\left(8u(L_x, y_j) - 2v_{N_x,j} + v_{N_x-1,x}\right)$ with $0 \leq j \leq N_y$,

2.5.2 The Pressure equation boundary conditions

It is important to note that the pressure field in this projection scheme is not physical, but artificial. As we can read in [38], "The Navier-Stokes equations may be considered as a boundary value problem for velocity [momentum equation (2.1)] with Lagrange multiplier P which is introduced in order to compensate the additional constraint [conservation of mass equation (2.2)]. From this point of view, no boundary condition for pressure is needed. The number of degrees of freedom for the discrete pressure variable should be then equal to the number of discrete constraint equations [momentum equation (2.1)]. [The pressure equation] is to be considered as a constituent of the projection operator; on the discrete level, it may appear from algebraic arguments rather than from an approximation of a boundary value problem." The pressure boundary conditions are dictated by algebraic reasons.

If we project Eq.(2.20) on η, the normal outward unit vector at the domain boundary $\partial\Omega$, we get:

$$V^{n+1}_{|\partial\Omega}.\eta - V^{*}_{|\partial\Omega}.\eta = \frac{\Delta t}{\rho}\nabla P^{n+1}_{|\partial\Omega}.\eta = \frac{\Delta t}{\rho}\left[\frac{\partial P^{n+1}}{\partial \eta}\right]_{|\partial\Omega}. \tag{2.48}$$

So we have some Neumann boundary conditions for the pressure equation:

$$\Delta P^{n+1} = \frac{\rho}{\Delta t}\nabla.V^{*}, \tag{2.49}$$

$$\left[\frac{\partial P^{n+1}}{\partial \eta}\right]_{|\partial\Omega} = \frac{\rho}{\Delta t}\left[V^{*}_{|\partial\Omega}.\eta - V^{n+1}_{|\partial\Omega}.\eta\right], \tag{2.50}$$

depending on $V^{*}_{|\partial\Omega}$. However, we note that since V^{*} is calculated with an explicit scheme, the inner points of V^{*} do not depend on $V^{*}_{|\partial\Omega}$.

The pressure equation along with this type of boundary conditions is a Neumann problem. A constraint for $V^*_{|\partial\Omega}$ is that the Neumann problem compatibility condition is satisfied:

$$\int_\Omega \Delta P^{n+1} d\vartheta = \int_{\partial\Omega} \left[\frac{\partial P^{n+1}}{\partial \eta}\right]_{|\partial\Omega} d\varsigma. \tag{2.51}$$

If we use (2.49) and (2.50), we get:

$$\int_\Omega \frac{\rho}{\Delta t} \nabla . V^* d\vartheta = \int_{\partial\Omega} \frac{\rho}{\Delta t} \left[V^*_{|\partial\Omega}.\eta - V^{n+1}_{|\partial\Omega}.\eta\right] d\varsigma. \tag{2.52}$$

The constants vanish and Green-Ostrogradski's theorem gives us:

$$\int_\Omega \nabla . V^* d\vartheta = \int_{\partial\Omega} V^*_{|\partial\Omega}.\eta d\varsigma, \tag{2.53}$$

and

$$\int_\Omega \nabla . V^{n+1} d\vartheta = \int_{\partial\Omega} V^{n+1}_{|\partial\Omega}.\eta d\varsigma. \tag{2.54}$$

Finally the compatibility condition for the Neumann problem is:

$$\int_\Omega \nabla . V^{n+1} d\vartheta = 0, \tag{2.55}$$

which is independent of $V^*_{|\partial\Omega}$ and of course satisfied, since V^{n+1} is a divergence-free vector field.

As it is explained in [91], a numerical specificity of the staggered mesh and the explicit projection scheme is that the discrete solution V^{n+1}_h is actually independent from the discrete value of $V^*_{h,|\partial\Omega}$. Let us consider, for example, the discretization of the pressure equation (2.49) around the boundary $x = 0$, $0 \leq y \leq L_y$:

$$\frac{1}{h_x}\left(\frac{P^{n+1}_{2,j} - P^{n+1}_{1,j}}{h_x} - \frac{P^{n+1}_{1,j} - P^{n+1}_{0,j}}{h_x}\right) \tag{2.56}$$

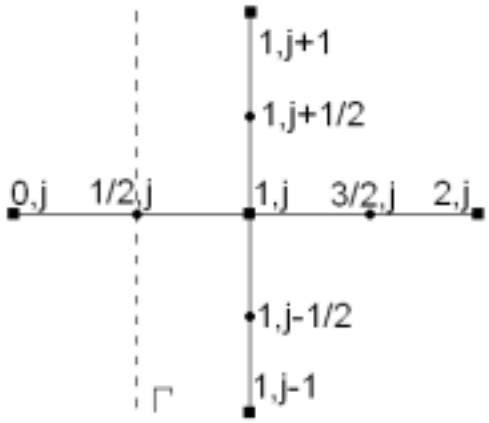


Figure 2.4: *Left boundary of* Ω *with a staggered extended mesh.*

$$+\frac{1}{h_y}\left(\frac{P_{1,j+1}^{n+1}-P_{1,j}^{n+1}}{h_y}-\frac{P_{1,j}^{n+1}-P_{1,j-1}^{n+1}}{h_y}\right)=$$
$$\frac{1}{\Delta t}\left(\frac{u_{\frac{3}{2},j}^{*}-u_{|\partial\Omega,j}^{*}}{h_x}+\frac{v_{1,j+\frac{1}{2}}^{*}-v_{1,j-\frac{1}{2}}^{*}}{h_y}\right).$$

Let us apply the boundary condition (2.50) on this side of Ω:

$$\frac{1}{h_x}\left(P_{1,j}^{n+1}-P_{0,j}^{n+1}\right)=-\frac{1}{\Delta t}\left(u_{|\partial\Omega}^{n+1,j}-u_{|\partial\Omega,j}^{*}\right). \tag{2.57}$$

If we plug that into (2.56), we get:

$$\frac{1}{h_x}\left(\frac{P_{2,j}^{n+1}-P_{1,j}^{n+1}}{h_x}+\frac{1}{\Delta t}\left(u_{|\partial\Omega}^{n+1,j}-u_{|\partial\Omega,j}^{*}\right)\right) \tag{2.58}$$
$$+\frac{1}{h_y}\left(\frac{P_{1,j+1}^{n+1}-P_{1,j}^{n+1}}{h_y}-\frac{P_{1,j}^{n+1}-P_{1,j-1}^{n+1}}{h_y}\right)=$$
$$\frac{1}{\Delta t}\left(\frac{u_{\frac{3}{2},j}^{*}-u_{|\partial\Omega,j}^{*}}{h_x}+\frac{v_{1,j+\frac{1}{2}}^{*}-v_{1,j-\frac{1}{2}}^{*}}{h_y}\right).$$

The terms $u_{|\partial\Omega,j}^{*}$ on the left-hand side and the right-hand side cancel each other. By doing the same computation on the other sides of the domain, we show the numerical independence of the solution with respect to $V_{|\partial\Omega}^{*}$. Thus, we can choose some

convenient arbitrary values for $V^*_{|\partial\Omega}$ such as:

$$V^*_{|\partial\Omega} = V^{n+1}_{|\partial\Omega}.$$

In that case, Eq.(2.48) implies that we have uniform homogeneous Neumann boundary conditions for the pressure equation in the projection scheme.

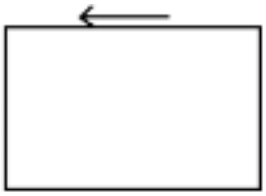
Now we look at some basic 2D NS test cases. Some of them involve the direct forcing method, presented in chapter 1, to model obstacles.

2.6 The Driven cavity flow

2.6.1 Two-dimensional cavity flow

The flow is moving in a closed square box $\Omega = [0,1]^2$ where the upper wall is moving sideways:

- $u(0,y) = u(1,y) = v(0,y) = v(1,y) = 0$, for $0 \leq y \leq 1$,
- $u(x,0) = v(x,0) = v(x,1) = 0$, for $0 \leq x \leq 1$,
- $u(x,1) = 4x(1-x)$, for $0 \leq x \leq 1$.



The velocity of the fluid is null initially and then converges into a steady state, depending on the Reynolds number:

$$Re = \frac{\rho L|V|}{\mu}, \tag{2.59}$$

where L is the characteristic length and V the characteristic velocity. In our case, both are equal to one. Figures 2.5 and 2.6 show the horizontal and vertical component of the velocity for $Re = 1000$.

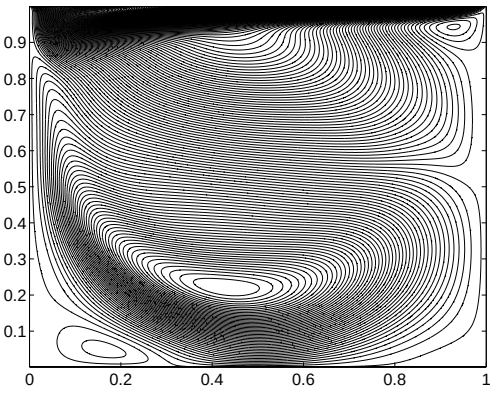


Figure 2.5: *Contour of u after convergence. Re=1000. Problem size:* 600×600.

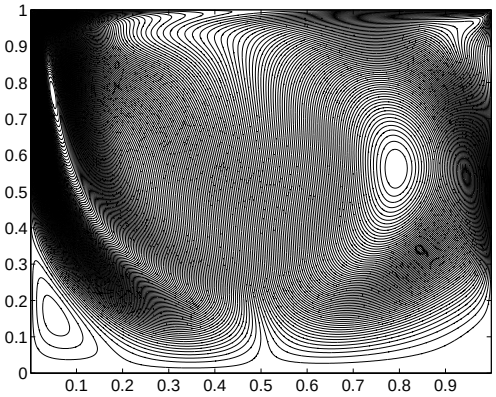


Figure 2.6: *Contour of v after convergence. Re=1000. Problem size:* 600×600.

2.6.2 Two-dimensional cavity flow with obstacle

A fixed obstacle is inserted in the cavity using a simple momentum forcing method of J. Mohd-Yusof [57]. Figures 2.8 and 2.9 show the velocity fields in the driven cavity with a Reynolds number of 200, a geometry shown in Figure 2.7, at time $t = 1.1s$.

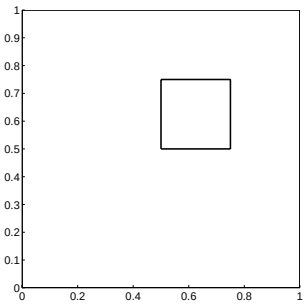


Figure 2.7: *Domain geometry.*

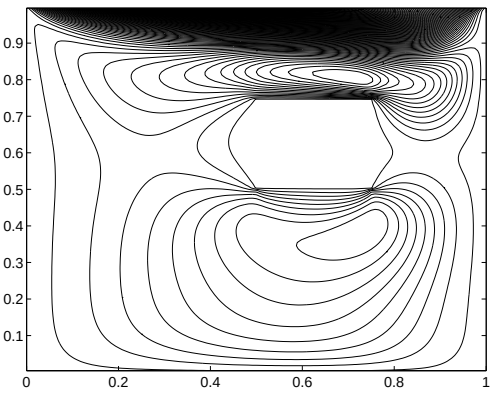


Figure 2.8: *Contour of u. Cavity with obstacle.*

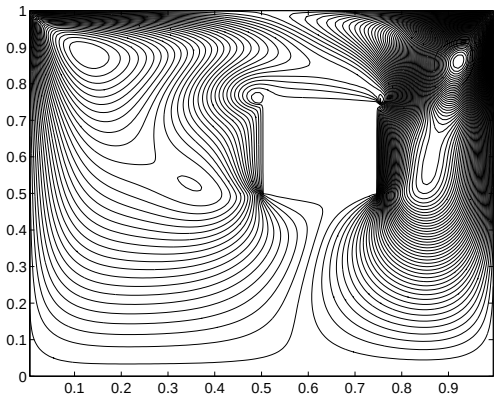


Figure 2.9: *Contour of v. Cavity with obstacle.*

2.6.3 Three-dimensional cavity flow

This is the same cavity test case without obstacle as before but in three dimensions. The upper wall is moving in an unit closed box (see Figures 2.10, 2.11, 2.12 and 2.13). However, we are very limited by the size of the problem in MATLAB.

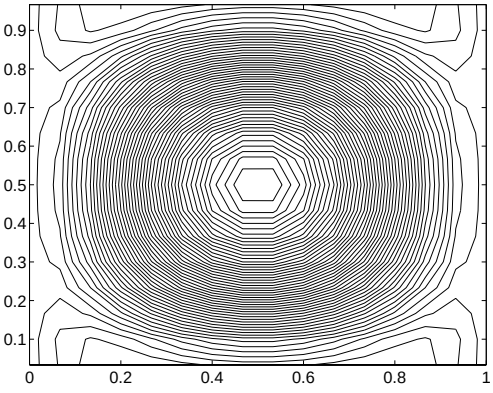


Figure 2.10: *Contour of $u(x, y, 0.8)$.*

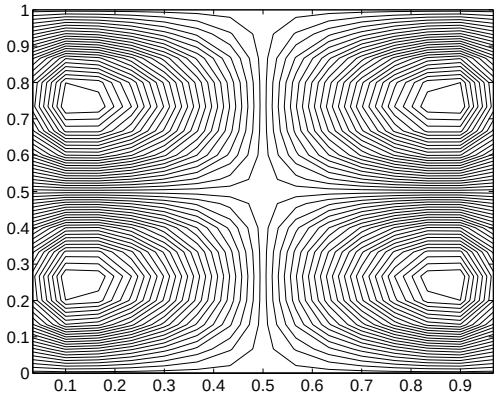


Figure 2.11: *Contour of $v(x, y, 0.8)$.*

2.7 The Poiseuille flow with obstacle

The velocity on the inflow (left) is forced while the outflow (right) is a free boundary, the walls (top and bottom) have a no-slip boundary condition:

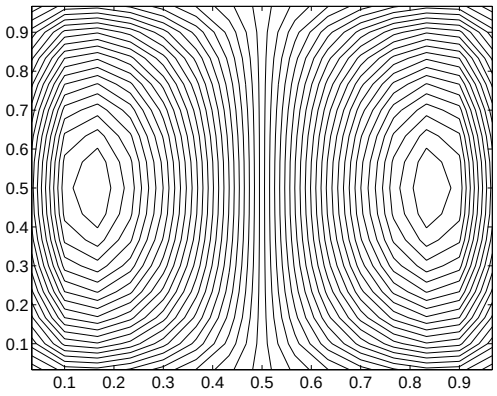


Figure 2.12: *Contour of* $w(x, y, 0.8)$.

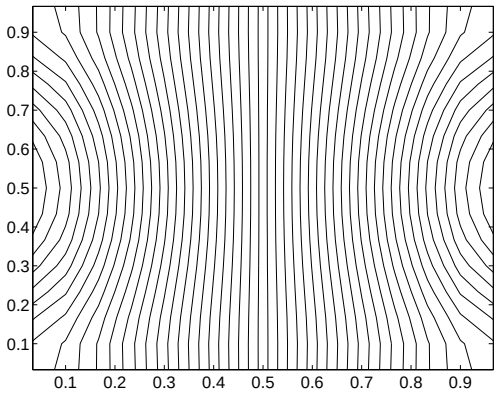


Figure 2.13: *Contour of* $p(x, y, 0.8)$.

- left: $u = u_0(y), \;\; v = 0, \;\; \frac{\partial p}{\partial x} = 0,$
- right: $\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = 0, \;\; p = 0,$
- top and bottom: $u = v = 0, \;\; \frac{\partial p}{\partial y} = 0.$

An obstacle is inserted in the computational domain using the momentum forcing method. Figure 2.14 shows the geometry of the domain: the obstacle in this case, is not exactly placed in the middle of the channel.

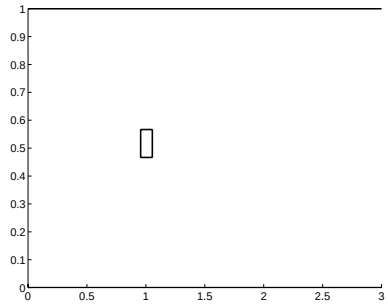


Figure 2.14: *Domain geometry.*

Figures 2.15 and 2.16 show the contours of u and v respectively with a Reynolds number of 800.

With an interpolation technique, we can apply the momentum forcing to any shaped obstacle. In the following case, we choose a circular obstacle (Figure 2.17). Here we

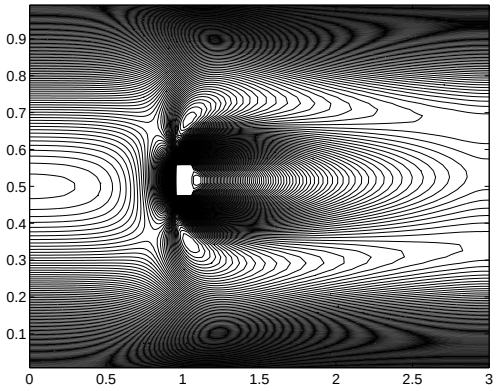


Figure 2.15: *Contour of u.*

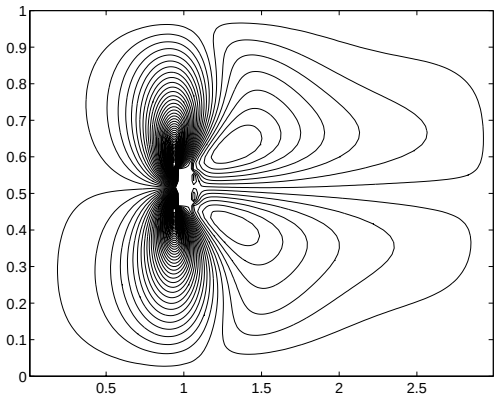


Figure 2.16: *Contour of v.*

have a high Reynolds number, which causes the apparition of Von Karman Vortices. We can see the periodic nature of this Von Karman vortex street on the figure.

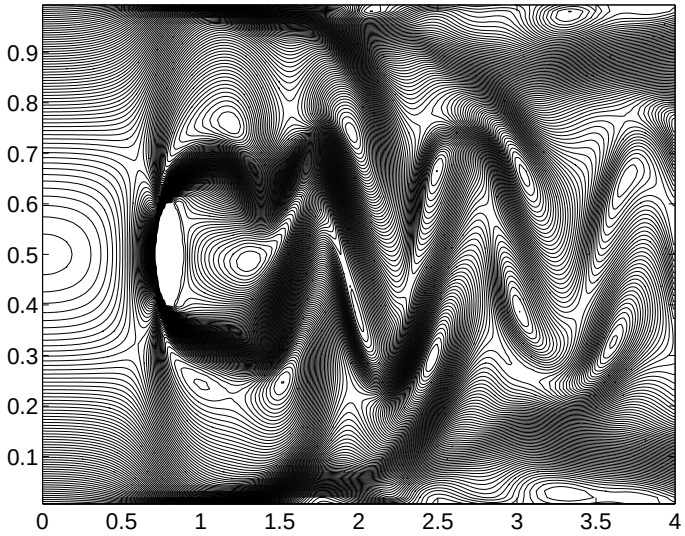


Figure 2.17: *Flow past a cylinder.*

Another test case that uses the momentum forcing is the step test case: the obstacle is located at the bottom left of the channel (Figures 2.18 and 2.19).

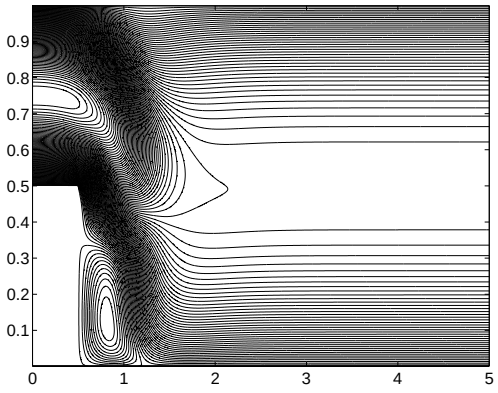


Figure 2.18: *Contour of u. Step test case.*

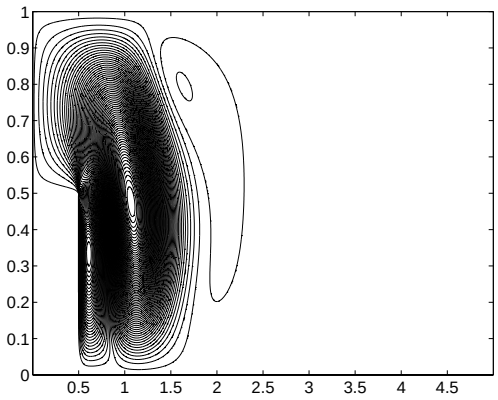


Figure 2.19: *Contour of v. Step test case*

2.8 A Note on the numerical solvers

MATLAB offers a broad variety of efficient solvers: direct or indirect methods (Krylov methods). It is better to use the MATLAB solvers that call C procedures than to actually code the methods, specially the iterative methods, since MATLAB is an interpreted language. However, our multigrid implementation gave some good results, due to the efficiency of the method.

For the IBM, it is better to use a direct solver: the decomposition is computed once and for all in the initialization phase of the NS code. The pressure solver uses this decomposition at every time step. We actually use the MATLAB LUPQ solver, which uses the UMFPACK library [22]. The operator matrix M is decomposed into a unit lower triangular matrix L, a upper triangular matrix U, a permutation matrix P and a column reordering matrix Q so that $PMQ = LU$, using the fact that it is a sparse matrix. This factorization, based on a tree and special ordering, optimizes the memory access patterns. The UMFPACK decomposition provides particularly good performance in MATLAB compared to the original MATLAB LU solver that is typically two to three times slower for relatively large sizes of arrays.

To give an idea about the performance of MATLAB compared to FORTRAN, we

ran the 2D Poiseuille flow test case using the same code, written in both languages. However, the FORTRAN code uses a classic LINPACK LU solver while the MATLAB one uses the UMFPACK LU solver. The elapsed time in MATLAB is measured using the *etime* function that measures the actual clock time. This has been made on an AMD 2.0GHz Opteron with a 64 bit architecture and 32 GB of RAM. The Pathscale FORTRAN compiler was used with the full optimization flag $-O3$.

Problem size	Numb. of time steps	FORTRAN	MATLAB
400×100	100	$14.35s$	$37.82s$
800×200	50	$39.82s$	$81.17s$
1200×300	50	$122.51s$	$121.50s$

Table 2.1: *Elapsed time for the 2D poiseuille flow code, using MATLAB or FORTRAN.*

The overhead due to the the interpreted language is decreasing when the problem size increases. The larger the problem is, the more time we spend solving the linear system of the pressure with the C solver routines. At the same time, the larger the problem is, the larger the operator matrix is, the more we benefit from the memory access optimization from UMFPACK. With these two factors combined, we obtain a similar elapsed time for a relatively large problem size, using MATLAB or FORTRAN.

Now that we have introduced the incompressible NS computations, their basic discretization and the solver we used, we present the IBM.

Chapter 3

The Immersed Boundary Method

We introduced the main idea behind the IBM in the first chapter. Now we describe the actual implementation of the IBM. Then we present the applications and the recent developments of the method. Finally, we introduce some of the IBM test cases.

While the IBM has been extended for elastic membrane with mass and volume or two-phase flow simulations, we consider here the discretization of the massless sharp membrane. Another restriction here, is that the IBM we present, is fully explicit. We use some semi-implicit and fully implicit schemes, which are described in Chapter 5, about the stability of the method. For simplicity again, we only consider the two-dimensional case, where Γ, the immersed boundary, is a continuous one-dimensional incompressible massless elastic membrane.

The IBM is based on two equations: the first one deals with the repartition of the curvilinear force density on the Cartesian grid and the second one with the motion of the immersed boundary.

3.1 The Force term

3.1.1 The Elastic force

Let $f(s,t)$ be the local elastic force density along Γ, the immersed boundary: $0 < s < 1$ is the curvilinear coordinates and $0 < t < T$ is the time. We need at first to describe the elastic law followed by f. We usually use Hooke's law, i.e. the tension T in the boundary is a linear function of the strain:

$$T(s,t) = T\left(\left|\frac{\partial X(s,t)}{\partial s}\right|\right) = \sigma\left|\frac{\partial X(s,t)}{\partial s}\right| \tag{3.1}$$

The vector $X(s,t)$ is the position vector of the moving boundary expressed in the Cartesian coordinates. σ is the elastic coefficient. In this case, the boundary is never resting, the tension is always positive.

The tension is a scalar value, but it is applied to a point of Γ along a tangential direction by definition, that is, along the unit tangent vector $\tau(s,t)$:

$$\tau(s,t) = \frac{\partial X(s,t)/\partial s}{|\partial X(s,t)/\partial s|} \tag{3.2}$$

Now if we evaluate the elastic force applied on a small segment region $[s_0, s_0+\epsilon]$ of Γ, we get:

$$T(s_0+\epsilon,t)\tau(s_0+\epsilon,t) - T(s_0,t)\tau(s_0,t).$$

which is equal to:

$$\int_{s_0}^{s_0+\epsilon} \frac{\partial\left(T(s,t)\tau(s,t)\right)}{\partial s} ds.$$

The immersed boundary being massless and without any volume, by taking the limit when $\epsilon \longrightarrow 0$ of the derivative of this force, we can say that the local density force applied by the elastic membrane to the fluid at the point s_0 is:

$$f(s_0,t) = \frac{\partial(T(s_0,t)\tau(s_0,t))}{\partial s}, \tag{3.3}$$

This corresponds to the relationship between the local force density and the tension along Γ. So in the case of Hooke's law, we have:

$$f(s,t) = \frac{\partial\left(\sigma\left|\frac{\partial X(s,t)}{\partial s}\right|\frac{\partial X(s,t)/\partial s}{|\partial X(s,t)/\partial s|}\right)}{\partial s} = \frac{\partial}{\partial s}\left(\sigma\frac{\partial X}{\partial s}\right) = \sigma\frac{\partial^2 X}{\partial s^2} \tag{3.4}$$

It is possible to decompose the local force density as the sum of a normal and a tangential component:

$$f(s,t) = \frac{\partial(T(s,t)\tau(s,t))}{\partial s} = \frac{\partial T(s,t)}{\partial s}\tau(s,t) + T(s,t)\frac{\partial\tau(s,t)}{\partial s}, \tag{3.5}$$

with:

$$T(s,t)\frac{\partial\tau(s,t)}{\partial s} = T(s,t)\left|\frac{\partial X(s,t)}{\partial s}\right|\frac{|\partial\tau(s,t)/\partial s|}{|\partial X(s,t)/\partial s|}\frac{\partial\tau(s,t)/\partial s}{|\partial\tau(s,t)/\partial s|}.$$

If we call $K(s,t)$ the curvature of Γ and $\eta(s,t)$ its normal vector, we have:

$$K(s,t) = \frac{|\partial\tau(s,t)/\partial s|}{|\partial X(s,t)/\partial s|},$$

$$\text{and}: \quad \eta(s,t) = \frac{\partial\tau(s,t)/\partial s}{|\partial\tau(s,t)/\partial s|}.$$

Thus, if we plug K and η back into the expression of f, we get:

$$f(s,t) = \frac{\partial T(s,t)}{\partial s}\tau(s,t) + T(s,t)\left|\frac{\partial X(s,t)}{\partial s}\right|K(s,t)\eta(s,t). \tag{3.6}$$

Now, another possible choice for the tension T is:

$$T(s,t) = \begin{cases} \sigma\left(\left|\frac{\partial X}{\partial s}\right| - 1\right), & \left|\frac{\partial X}{\partial s}\right| \geq 1 \\ \\ 0, & \text{otherwise.} \end{cases}$$

In this case the boundary has an equilibrium position but no contraction force. In order to have both extension and contraction forces in the boundary we would use this tension:

$$T(s,t) = \sigma\left(\left|\frac{\partial X}{\partial s}\right| - 1\right) \tag{3.7}$$

Then the Cartesian force term F introduced in the Navier-Stokes equations is:

$$F(x,t) = \int_\Gamma f(s,t)\delta(x - X(s,t))ds, \quad (x,t) \in \Omega \times [0,T], \tag{3.8}$$

using a simple property of the Dirac delta function.

3.1.2 The Discrete Dirac delta function

The force term is ideally zero everywhere except along Γ. By definition:

$$F(x) = \int_\Gamma f(s)\delta(x - X(s))ds = \begin{cases} f(s) & \text{if} \quad x = X(s), \\ 0, & \text{otherwise.} \end{cases} \tag{3.9}$$

In the computations, the δ function above is actually regularized by a discrete Dirac delta function of compact support. The aim is to minimize the discretization error introduced by this regular function δ_h while the requirements due to the stencil used in the finite-difference discretization and the physical behavior of the elastic moving boundary are satisfied. These requirements are detailed in Chapter 4.

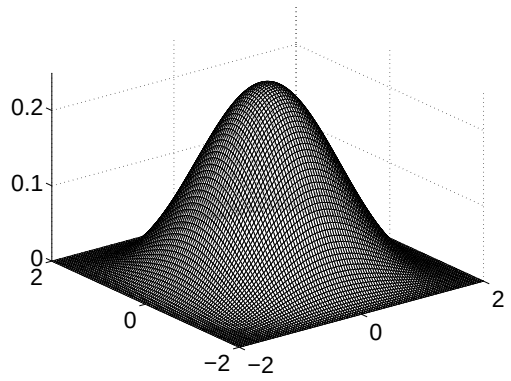


Figure 3.1: *2D exemple of a discrete Dirac delta function δ_h, with $h = 1$.*

3.1.3 The Discrete force term

Let us describe the force term after discretization of the immersed boundary, without considering the time dependency. In this case we assume Γ to be a closed curve. We take a uniformly distributed set of M discrete points along Γ. Then the discrete curvilinear mesh is : $s_k = \frac{k}{M} L_\Gamma = k h_\Gamma, \quad 0 \leq k \leq M-1$. L_Γ is the length of Γ and $h_\Gamma = \frac{L_\Gamma}{M}$, its curvilinear space step. So, the Cartesian force term can be written as:

$$F_h(x) = h_\Gamma \sum_{k=0}^{M-1} f(s_k) \delta_h(x - X(s_k)), \quad x \in \Omega. \tag{3.10}$$

The two-dimensional discrete Dirac delta function is equal to the product of the one-dimensional delta functions for each Cartesian component:

$$\delta_h(x_1, x_2) = \delta_h(x_1) \delta_h(x_2.) \tag{3.11}$$

So for each node, we have, with $x = [x_1, x_2]$, $X_k = X(s_k) = [X_1(s_k), X_2(s_k)] = [X_{1,k}, X_{2,k}]$ and $f(s_k) = f_k = [f_k^1, f_k^2]$:

$$F^l(x) = h_\Gamma \sum_{k=0}^{M-1} f_k^l \delta_h(x_1 - X_{1,k}) \delta_h(x_2 - X_{2,k}), \quad l = 1, 2. \tag{3.12}$$

If we use Hooke's law and centered finite-differences, we have:

$$f_k^l = \sigma \frac{\partial^2 X_l(s_k)}{\partial s^2} = \sigma \frac{X_{l,k+1} - 2X_{l,k} + X_{l,k-1}}{h_\Gamma^2}, \quad l = 1, 2.$$

Now we discretize x: $x_{i,j} = [x_{1,i}, x_{2,j}]$, such that $x_{1,i} = (i-1)h, \quad x_{2,j} = (j-1)h$ $(1 \leq i, j \leq N)$, $h = \frac{1}{N-1}$.

We get:

$$F_{i,j}^l = h_\Gamma \sum_{k=0}^{M-1} f_k^l \delta_h(x_{1,i} - X_{1,k}) \delta_h(x_{2,j} - X_{2,k}), \quad l = 1, 2 \;\; 1 \leq i, j \leq N, \tag{3.13}$$

and finally:

$$F^l_{i,j} = \frac{\sigma}{h_\Gamma} \sum_{k=0}^{M-1} \left(X_{l,k+1} - 2X_{l,k} + X_{l,k-1}\right) \delta_h(x_{1,i} - X_{1,k}) \delta_h(x_{2,j} - X_{2,k}), \quad l = 1, 2 \;\; 1 \leq i, j \leq N. \tag{3.14}$$

It is important to note that the support of δ_h is limited to a few space steps and that we only use the discrete Cartesian points inside the support of each delta function δ_k when we evaluate the Cartesian force term F. This saves a lot of computation. You can see an example of F on Figure (3.2).

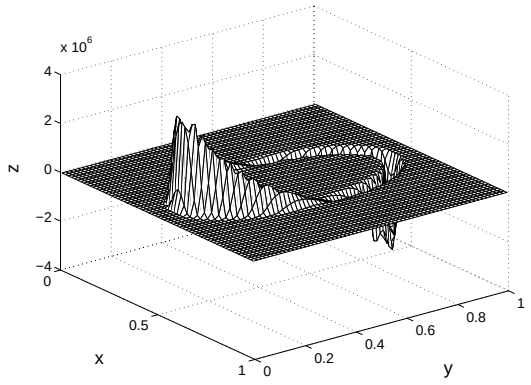


Figure 3.2: 2*D example of the horizontal component of a Cartesian force term F along the closed elliptic immersed boundary* Γ.

3.2 The No-Slip boundary condition

The immersed boundary is moving at the same speed as the neighboring fluid particles. This is expressed in this equation:

$$\frac{\partial X(s,t)}{\partial t} = V(X(s,t),t). \tag{3.15}$$

The same interpolation technique as before is used to interpolate the Cartesian field V, the fluid velocity, into the Lagrangian vector $\frac{\partial X}{\partial s}$:

$$\frac{\partial X(s,t)}{\partial t} = \int_{\Omega} V(x,t)\delta(x - X(s,t))dx. \tag{3.16}$$

After time and space discretization ($V = [V^{(1)}, V^{(2)}]$), we have:

$$\frac{X_{l,k}^{n+1} - X_{l,k}^{n}}{\Delta t} = h^2 \sum_{i=1}^{N}\sum_{j=1}^{N} V_{i,j}^{(l)} \delta_h(x_{1,i} - X_{1,k})\delta_h(x_{2,j} - X_{2,k}), \;\; l = 1, 2 \;\; , 0 \leq k \leq M-1. \tag{3.17}$$

3.2.1 The Explicit Scheme for the IBM

Here is a simplified description of the basic scheme for the IBM:

$$f_k^n = \sigma\left[\frac{X_{k+1}^n - 2X_k^n + X_{k-1}^n}{h_\Gamma}\right], \quad 0 \leq k \leq M-1 \tag{3.18}$$

$$F_{i,j}^n = \sum_{k=0}^{M-1} f_k^n \delta_h(x_{i,j} - X_k^n)h_\Gamma, \quad 0 \leq i \leq N_x - 1, \quad 0 \leq j \leq N_y - 1. \tag{3.19}$$

$$\rho\left[\frac{V^* - V^n}{\Delta t} + (V^n.\nabla)V^n\right] = \mu\Delta V^n - \nabla P^n + F^n, \tag{3.20}$$

$$\Delta\delta P^n = \frac{\rho}{\Delta t}\nabla.V^*, \quad \left[\frac{\partial(\delta P^n)}{\partial \eta}\right]_{|\partial\Gamma} = 0. \tag{3.21}$$

$$\rho\left[\frac{V^{n+1} - V^*}{\Delta t}\right] = -\nabla(\delta P^{n+1}). \tag{3.22}$$

$$P^{n+1} = P^n + \delta P^n. \tag{3.23}$$

$$\frac{X^{n+1} - X^n}{\Delta t} = h^2 \sum_{i=0}^{N_x-1}\sum_{i=0}^{N_y-1} V_{i,j}^{n+1}\delta_h(x_{i,j} - X^n). \tag{3.24}$$

Only first order convergence rates have been observed for simulations of sharp immersed elastic boundaries.

3.3 Some Applications of the IBM

The method has been originally developed for blow flow in the heart [86, 87, 24]. However, we find applications of the IBM to many different cases. Most of them are found in Biology, for example:

- Models of stenoses [2].
- Models of red blood cells [27].
- Models of aquatic animal locomotion [30].
- Models of cochlea [8].
- Models of platelet adhesion and aggregation during blood clotting [32].
- Models of insect flight [76, 77].

Or we find it in more general fluid dynamics problems:

- Dynamics of suspension [104], particulate flows [113].
- Drop dynamics [35, 36].
- Simulation of a flapping flexible filament in a flowing soap film by the immersed boundary method [124].
- Flow past complex fixed geometries [54, 3].
- Flow around an arbitrarily moving body [60].
- Models of parachutes and flags [61, 62].

More applications can be found in [89]. The diversity of the IBM applications shows the versatility of the method. Now we look at the recent developments of the method.

3.4 Recent developments of the IBM

We refer to [89] for a summary of the IBM advances. Some efforts have been made to increase the accuracy, the stability, the efficiency and the versatility of the method. Basically, the usual IBM has a second order convergence in time and space, except in the neighborhood of the sharp interface where it is only of first order. It is fairly unstable too, which means that the time stepping is far below the CFL condition in the cases of stiff elastic membranes.

In [97], A.M. Roma implements the IBM on a staggered grid with the adaptive mesh refinement technique in the neighborhood of the immersed boundary. This gives rise to a local multilevel method. However, this technique with moving grids is hard to implement and is limited by the time stepping, proportional to the smallest grid space step. In [65], M.C. Lai and C.S. Peskin introduce an IBM with formal second-order accuracy and reduced numerical viscosity. However, only first order convergence properties have been observed computationally, in the case of non-smooth solutions. Similarly in [44], an actual second order IBM is introduced, but the measured convergence order does not correspond to cases of infinitely thin membranes and the projection scheme is such that the divergence of the fluid flow only converges to zero with a second order rate.

As for the efficiency of the method, an effort is being made on the parallelization of the method for 3D simulations [70, 44]. Finally, some work has been done to extend the possibilities of applications of the IBM: in [62] for example, Y. Kim and C.S. Peskin detail the simulations of immersed boundaries with a different density than the fluid, interacting with the flow. In regular IBM simulation, the immersed boundary is considered as massless if it does not have a volume, or to have the same density as the fluid, if it has a volume.

3.5 The Test cases

Let us present the test cases used in this numerical study of the IBM.

3.5.1 The Bubble test case

The first test case that we use is the 2D "bubble" test-case (Figure 3.3): a closed elastic moving boundary is immersed inside an incompressible fluid. The domain is a closed square box. At the beginning the fluid velocity is null and the elastic boundary is stretched, with a large potential energy.

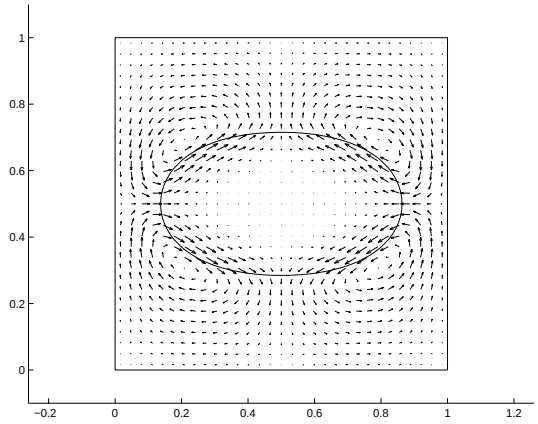


Figure 3.3: *"Bubble" test-case*

3.5.2 The Driven cavity bubble test-case

This 2D test case is similar to the "bubble" test case, except that one of the walls of the square box is moving tangentially and that the "bubble" is not initially stretched (Figure 3.4). It is interesting to study the behavior of the closed elastic immersed boundary inside a strong cavity flow, especially around the sliding wall. In this test case, being a stiff time-dependant case of the IBM, we studied the stability and the volume conservation property of the scheme, comparing the two different discrete

Dirac delta functions. In this particular case, we have again $\sigma = 10000$, $M = 6N$, $\mu = 1$ and a sliding velocity for the upper wall of -500.

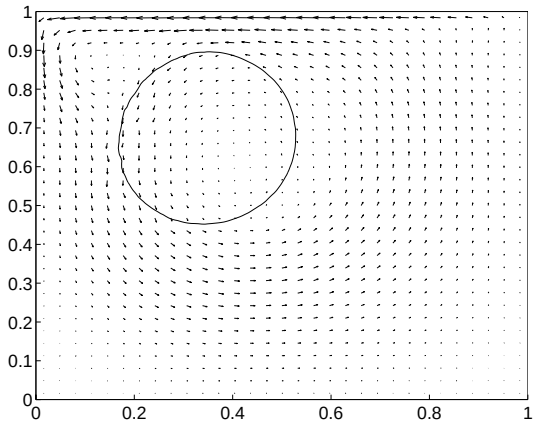


Figure 3.4: *Cavity flow "bubble" test case. The upper wall is sliding to the left.*

The other test cases that we implemented are more specific to blood flows:

- Two-dimensional artery flow.
- Large blood cells flowing in the two-dimensional driven cavity.
- Large blood cell flowing in a two-dimensional channel, with an obstacle.

These two last test cases rise the issue of the contact between immersed boundaries, or between an immersed boundary and a fixed obstacle. This needs to be explored.

Now we start the study, with the accuracy of the IBM.

Chapter 4

Accuracy

The IBM requires the interpolation of some Lagrangian variables into Eulerian fields and vice-versa. Its interpolating tool is the discrete Dirac delta function, which is a regularization of the Dirac function. If we call $f(X),\ X \in \Gamma$, the Lagrangian variable and $F(x),\ x \in \Omega$, the Eulerian field, we recall that the two interpolation formulas of the IBM are:

$$f(X) = \int_{\Omega} F(x)\delta(x - X)dx, \tag{4.1}$$

$$F(x) = \int_{\Gamma} f(X)\delta(x - X)dX. \tag{4.2}$$

As seen in the introduction, Eq.(4.1) is used after the NS computations, for the boundary position update, while Eq.(4.2) is used before, for the flow computation. Thus, the first equation has more impact on the stability of the method and the second one on the accuracy.

This chapter describes several methods, whose final purpose is to improve the accuracy of the IBM. More precisely, we want to improve the computation of the pressure in the basic NS projection scheme when a singular force term is introduced. This is why

the main issue is solving elliptic problems with singular source terms under the IBM constraints, in order to solve the equations with $F(x)$ from Eq.(4.2) as the right-hand side, with a good uniform accuracy. Here are the main points in each section:

- *Section 1.* Optimal choice for the discrete Dirac delta function, in the framework of the IBM.
- *Section 2.* Application to elliptic problems with singular source terms.
- *Section 3.* Application to the IBM. Introduction of a globally conservative scheme based on constrained optimization.
- *Section 4.* A fast and accurate solver: the Multigrid/τ-extrapolation technique.
- *Section 5.* A correction technique to improve accuracy.

Let us start with a detailed description of the requirements and properties of the discrete Dirac delta function.

4.1 The Discrete Dirac delta function

The Discrete Dirac delta function is usually written in this form:

$$\delta_h(x) = \frac{1}{h}\phi\left(\frac{x}{h}\right), \tag{4.3}$$

where h is the space step. The function ϕ needs to satisfy several compatibility conditions with the IBM.

4.1.1 The Requirements of the discrete Dirac delta functions for the IBM

Here are the conditions introduced by C. Peskin [88]:

- (i) $\phi \in C^0(\mathbb{R})$, necessary for a smooth interpolation.
- (ii) ϕ is of finite support. The computational cost of the method is actually proportional to the width of the support.
- (iii) $\sum_{i\in\mathbb{Z}} \phi(r-i) = 1, \quad \forall r \in \mathbb{R}$: ensures that constant functions are interpolated exactly by δ_h.
- (iv) $\sum_{i\in\mathbb{Z}} (r-i)\phi(r-i) = 0, \quad \forall r \in \mathbb{R}$: first moment condition, which ensures along with condition (3) that linear functions are interpolated exactly by δ_h.

Physically, these last two properties ensure that the mass, force and torque identities are satisfied in the IBM, in terms of Eulerian or Lagrangian variables. C. Peskin writes in [89] that thanks to these conditions: "momentum, angular momentum and energy are not spuriously created or destroyed by the interaction equations."

Condition (iii) is sufficient in the case of a staggered mesh but not of a collocated mesh. This comes from the use, in the numerical scheme, of the central difference operator $\mathbf{D}_{2h}$:

$$\mathbf{D}_{2h}(u(x)) = \frac{u(x+h) - u(x-h)}{2h}.$$

The null space of this operator is two-dimensional and contains all the functions u such that $u(x_0 + (i-1)h) = u(x_0 + (i+1)h) \quad \forall i$ and more specifically, after discretization, all the sets $\{u_i\}$ such that $\{u_i\}_{i\ even}$ or $\{u_i\}_{i\ odd}$ are constant. In order to avoid oscillations when the operator $\mathbf{D}_{2h}$ is applied to discrete fields interpolated from the Lagrangian functions with the shifted discrete Dirac delta functions, we need to have this condition:

- (v) $\sum_{i(even)} \phi(r-i) = \sum_{i(odd)} \phi(r-i) = C_1, \quad \forall r \in \mathbb{R}$.

This, associated with condition (iii) on ϕ, yields: $C_1 = \frac{1}{2}$. In the case of a staggered mesh, we use the difference operator $\mathbf{D}_h$ to compute the divergence:

$$\mathbf{D}_h(u(x)) = \frac{u(x+\frac{h}{2}) - u(x-\frac{h}{2})}{h},$$

which only requires condition (iii) but not (v), to avoid oscillations.

The last condition introduced by C. Peskin [88, 90] is quiet theoretical:

- (vi) $\sum_{i\in\mathbb{Z}} [\phi(r-i)]^2 = C_2 \leq 1, \ \ \forall r \in \mathbb{R}$.

In the IBM, the distributed force term is a sum of weighted shifted discrete Dirac delta functions. This force term is introduced in the NS equations in order to update the velocity, which is then used to change the immersed boundary position, using weighted, shifted discrete Dirac delta functions again. Assuming that the updated velocity contains the unmodified force term, then in the immersed boundary position update process, some shifted discrete Dirac delta functions are multiplied by themselves when the supports of the delta functions intersect. C. Peskin explains that, "in this two-step process, one would like the interaction of a fiber point with itself to be constant, independently of where that point sits with respect to the fluid lattice." This is what condition (vi) ensures. Furthermore, by using the Schwarz inequality, we can get this property:

$$\sum_{i\in\mathbb{Z}} \phi(r_1 - i)\phi(r_2 - i) \leq C_2, \ \ \forall (r_1, r_2) \in \mathbb{R}^2.$$

This inequality describes the fact that, "the interaction of one fiber point with another [is] bounded by the interaction of a fiber point with itself." However, oscillations introduced if condition (vi) is not satisfied, have a short wave length, in the same order of length as the space step. These kinds of oscillations already exist, as a by-product of the elasticity property of the immersed boundary and the poor quality of the pressure equation solution around the interface. These oscillations are what make the method fairly unstable, even if we use a discrete Dirac delta function that satisfies property (vi).

In order to avoid the small oscillations along the immersed boundary, we implemented a filtering technique that allows us to significantly increase the stability of the method, in stiff cases. It also allows us to eliminate condition (vi), while using a staggered mesh allows us to eliminate condition (v). This will be detailed in this chapter.

The minimal support of a function that would satisfy the conditions (i, ii, iii, iv, v, vi) is $4h$. It is then uniquely defined:

$$\phi_1(r) \quad = \quad \begin{cases} \frac{1}{8}\left(3 - 2|r| + \sqrt{1 + 4|r| - 4r^2}\right), & 0 \leq |r| \leq 1; \\ \frac{1}{8}\left(5 - 2|r| - \sqrt{-7 + 12|r| - 4r^2}\right), & 1 \leq |r| \leq 2; \\ 0, \quad \text{otherwise.} \end{cases} \tag{4.4}$$

This function is well approached by this cosine function (see Figure 4.1), which is most commonly used:

$$\phi_2(r) \quad = \quad \begin{cases} \frac{1}{4}\left(1 + cos(\frac{\pi r}{2})\right), & |r| \leq 2; \\ 0, \quad \text{otherwise.} \end{cases} \tag{4.5}$$

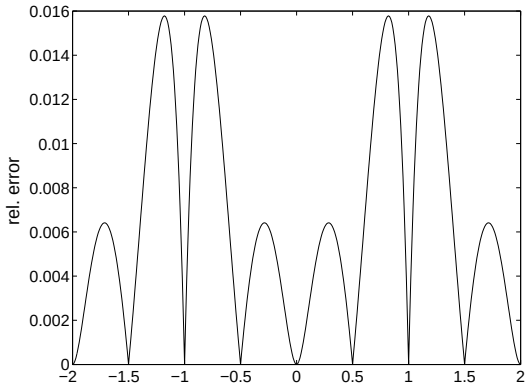


Figure 4.1: *Relative error between ϕ_1 and ϕ_2.*

Now for the staggered meshes, the minimal support of a function that would satisfy the conditions (i, ii, iii, iv, vi) is $3h$. It is uniquely defined also [97]:

$$\phi_3(r) \;=\; \begin{cases} \frac{1}{6}\left(5-3|r|-\sqrt{-3(1-|r|)^2+1}\right), & 0.5 \le |r| \le 1.5; \\ \frac{1}{3}\left(1+\sqrt{-3r^2+1}\right), & |r| \le 0.5; \\ 0, \quad \text{otherwise.} \end{cases} \tag{4.6}$$

Let us present the piecewise cubic discrete Dirac delta function, as introduced in [111], that has a $4h$ support:

$$\phi_4(r) \;=\; \begin{cases} 1-\frac{1}{2}|r|-|r|^2+\frac{1}{2}|r|^3, & 0 \le |r| \le 1; \\ 1-\frac{11}{6}|r|+|r|^2-\frac{1}{6}|r|^3, & 1 < |r| \le 2; \\ 0, \quad \text{otherwise.} \end{cases} \tag{4.7}$$

This function satisfies the conditions (i, ii, iii, iv), as well as these extra moment properties:

$$\sum_{i\in\mathbb{Z}}(r-i)^2\phi(r-i)=0, \quad \sum_{i\in\mathbb{Z}}(r-i)^3\phi(r-i)=0, \quad \forall r \in \mathbb{R}. \tag{4.8}$$

We know that the order of the discretization error is proportional to the number of moment conditions [111]. This will be fully explained in section (4.1.2).

The support of ϕ_4 is not minimal for a staggered mesh, but it is important to note that the computational cost of a the boundary treatment is of lower order than the one of the NS computations. So, a small increase in the width of the support does not significantly increase the cost of the method.

Since the function ϕ_4 does not satisfy condition (vi), let us look at the shape of the periodic function $f(r) = \sum_{i\in\mathbb{Z}} [\phi_4(r-i)]^2$ of period 1 in Figure 4.3. The function $f(r)$ is not constant and thus some small oscillations are introduced into the IBM. Its stability is reduced with the piecewise cubic discrete Dirac delta function but as we will see, the accuracy of elliptic problems with singular source terms, such as the Poisson equation, is improved. The aim is to improve the accuracy of the pressure

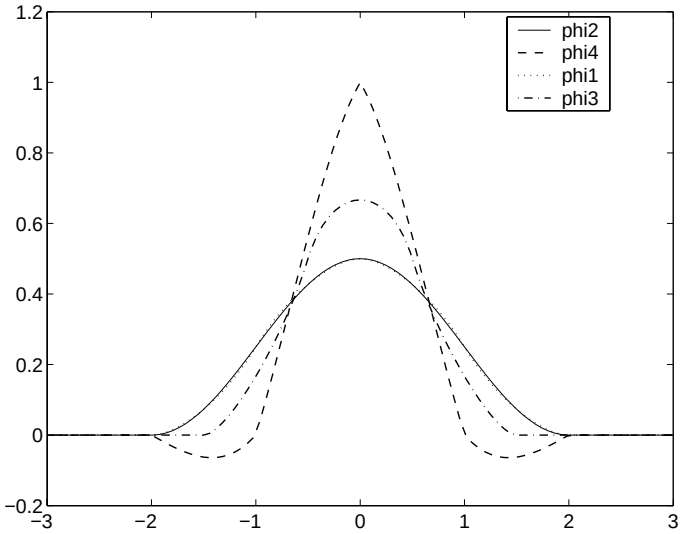


Figure 4.2: ϕ_1, ϕ_2, ϕ_3 *and* ϕ_4.

equation in the basic projection scheme of the NS equations; however, the right-hand side then contains dipoles and not delta functions.

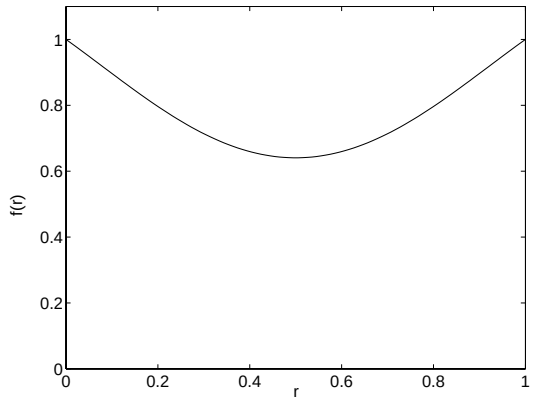


Figure 4.3: *Graph of the* C^0 *periodic function* $f(r) = \sum_{i\in\mathbb{Z}} [\phi_4(r-i)]^2$.

The discrete delta function (Figure 4.2) that satisfies conditions (i, ii, iii, iv, v, vi) and the additional moment properties described in (4.8) has a $6h$ support [105, 44]:

$$\phi_5(r) \;=\; \begin{cases} \dfrac{61}{112} - \dfrac{11}{42}|r| - \dfrac{11}{56}|r|^2 + \dfrac{1}{12}|r|^3 + \\ \dfrac{\sqrt{3}}{336}\sqrt{243 + 1584|r| - 748|r|^2 - 1560|r|^3 + 500|r|^4 + 336|r|^5 - 112|r|^6}, \\ 0 \le |r| \le 1; \\ \dfrac{21}{16} + \dfrac{7}{12}|r| - \dfrac{7}{8}|r|^2 + \dfrac{1}{6}|r|^3 - \dfrac{3}{2}\phi_5\left(|r|-1\right), \quad 1 \le |r| \le 2; \\ \dfrac{9}{8} - \dfrac{23}{12}|r| + \dfrac{3}{4}|r|^2 - \dfrac{1}{12}|r|^3 + \dfrac{1}{2}\phi_5\left(|r|-2\right), \quad 2 \le |r| \le 3; \\ 0, \quad \text{otherwise.} \end{cases} \tag{4.9}$$

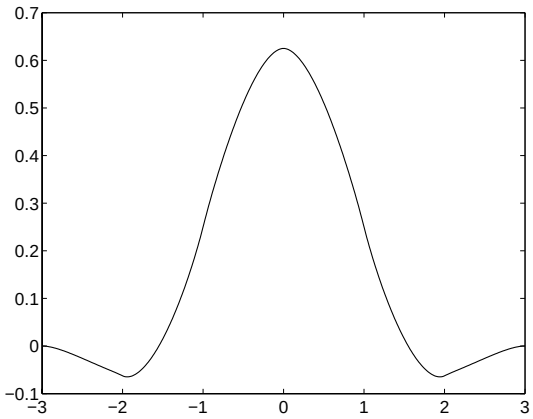


Figure 4.4: *The discrete Dirac delta function ϕ_5.*

Let us describe the link between the moment properties of the discrete Dirac delta function and the interpolation error.

4.1.2 On the interpolation error using the discrete Dirac delta functions

We start with the interpolation formula:

$$u(X) = \int_{-\infty}^{+\infty} u(x)\delta(x - X)dx. \tag{4.10}$$

We use a one-dimensional case ($X \in \mathbb{R}$ is fixed) and a uniform equally spaced mesh

$\{x_i\}$, with space step h. After discretization, using the discrete Dirac delta function d_h, we have:

$$u(X) \approx h \sum_i u(x_i) d_h(x_i - X). \tag{4.11}$$

If we assume that u is smooth enough, we can use the Taylor series to get series expressions of $u(x_i)$ around $u(X)$:

$$u(x_i) = u(X) + \sum_{n=1}^{+\infty} \frac{1}{n!} \frac{\partial^n u}{\partial x^n}(X)(x_i - X)^n. \tag{4.12}$$

By plugging (4.12) into (4.11), we get:

$$u(X) - h \sum_i u(x_i) d_h(x_i - X) = \tag{4.13}$$

$$u(X) - h \sum_i \left[u(X) + \sum_{n=1}^{+\infty} \frac{1}{n!} \frac{\partial^n u}{\partial x^n}(X)(x_i - X)^n \right] d_h(x_i - X)$$

$$= - \sum_{n=1}^{+\infty} \frac{1}{n!} \frac{\partial^n u}{\partial x^n}(X) h \sum_i (x_i - X)^n d_h(x_i - X),$$

using the identity:

$$h \sum_i d_h(x_i - X) = 1.$$

Since the discrete Dirac delta function is of finite support:

$$d_h(r) = 0, \quad |r| > mh, \tag{4.14}$$

where m is a small integer. So $d_h(x_i - X) \neq 0$ when $|x_i - X| \leq mh$, which leads to:

$$|h \sum_i (x_i - X)^n d_h(x_i - X)| \leq h m^n h^n \sum_i d_h(x_i - X) = m^n h^n. \tag{4.15}$$

Finally, by plugging (4.15) into (4.13), we have:

$$|u(X) - h \sum_i u(x_i) d_h(x_i - X)| \leq \sum_{n=1}^{+\infty} \frac{1}{n!} \left| \frac{\partial^n u}{\partial x^n}(X) \right| m^n h^n = O(h). \tag{4.16}$$

If the discrete Dirac delta function satisfies some extra moment properties:

$$\sum_i (x_i - X)^n d_h(x_i - X) = 0, \quad \text{for} \quad 0 < n < p, \tag{4.17}$$

then it is easy to show that the discretization error is of order p, for sufficiently smooth functions u. We start from (4.13):

$$\begin{aligned} & u(X) - h\sum_i u(x_i) d_h(x_i - X) \qquad (4.18) \\ &= -\sum_{n=1}^{+\infty} \frac{1}{n!}\frac{\partial^n u}{\partial x^n}(X) h \sum_i (x_i - X)^n d_h(x_i - X) \\ &= -\sum_{n=1}^{p-1} \frac{1}{n!}\frac{\partial^n u}{\partial x^n}(X) h \sum_i (x_i - X)^n d_h(x_i - X) - \sum_{n=p}^{+\infty} \frac{1}{n!}\frac{\partial^n u}{\partial x^n}(X) h \sum_i (x_i - X)^n d_h(x_i - X) \\ &= -\sum_{n=p}^{+\infty} \frac{1}{n!}\frac{\partial^n u}{\partial x^n}(X) h \sum_i (x_i - X)^n d_h(x_i - X), \end{aligned}$$

since the moments for $1 \leq n \leq p-1$ are null. Finally we get:

$$\left| u(X) - h\sum_i u(x_i) d_h(x_i - X) \right| \leq \sum_{n=p}^{+\infty} \frac{1}{n!} \left| \frac{\partial^n u}{\partial x^n}(X) \right| m^n h^n = O(h^p). \tag{4.19}$$

Now that we have seen the importance of the moment properties for the discrete Dirac delta function and the different constraints due to the IBM, we look at what would be an ideal discrete function to regularize a singular source point in an elliptic problem. This is done without the IBM constraints regarding the motion and the volume conservation of the immersed boundary.

4.1.3 The Ideal discretization of the discrete Dirac delta function.

The idea is to construct the optimal right-hand side in the Poisson problem with singular source terms, which could be helpful in the IBM. To construct it, we apply the Laplace operator to the exact discretized solution and shift the result.

Let us start with one singularity. If we look at the one-dimensional problem:

$$\frac{d^2u}{dx^2}(x) = \delta(x - x_0), \quad x \in [-1, 1], \quad x(-1) = 0 \text{ and } x_0 \in (0, 1), \tag{4.20}$$

the solution to this problem is:

$$u_e(x) = \begin{cases} 0, & x \leq x_0 \\ x - x_0, & x > x_0. \end{cases} \tag{4.21}$$

The optimal discrete right-hand side R associated with the finite difference stencil is trivial. With N points $\left(h = \frac{2}{N-1}\right)$ and $x_0 = (I+\epsilon)h \quad (0 \leq \epsilon < 1)$, then for $1 \leq i \leq N$:

$$R_i = \begin{cases} 0, & i \neq I \text{ and } i \neq I + 1 \\ \frac{1-\epsilon}{h}, & i = I \\ \frac{\epsilon}{h}, & i = I + 1. \end{cases} \tag{4.22}$$

This allows us to converge to the discretized analytic solution up to machine epsilon precision. Now, in two dimensions, the problem is:

$$\Delta u(x) = \delta(x - x_0, y - y0), \quad (x, y) \in \Omega = [-1, 1]^2, \tag{4.23}$$

$$u_{\partial\Omega} = u_{ex\,\partial\Omega} \text{ and } (x_0, y_0) \in (0, 1)^2.$$

$$u_{ex}(x, y) = -\frac{1}{4\pi} ln\left(\sqrt{(x - x_0)^2 + (y - y_0)^2}\right).$$

The discretized analytic solution is not bounded if the singularity coincides with a mesh point. So, we have this particular condition: $(\epsilon_x, \epsilon_y) \neq (0,0)$, with $x_0 = (i_0+\epsilon_x)h$ and $y_0 = (j_0+\epsilon_y)h$, $0 \leq \epsilon_x, \epsilon_y < 1$.

By applying the discrete Laplace operator to the exact solution, we get this optimal right-hand side (Figure 4.5):

$$RHS_{i,j} = \begin{cases} -\frac{1}{8\pi h^2} ln\left((x_{i+1,j}-x_0)^2 + (y_{i,j}-y_0)^2\right) \\ -\frac{1}{8\pi h^2} ln\left((x_{i,j}-x_0)^2 + (y_{i+1,j}-y_0)^2\right) \\ +\frac{1}{2\pi h^2} ln\left((x_{i,j}-x_0)^2 + (y_{i,j}-y_0)^2\right) \\ -\frac{1}{8\pi h^2} ln\left((x_{i-1,j}-x_0)^2 + (y_{i,j}-y_0)^2\right) \\ -\frac{1}{8\pi h^2} ln\left((x_{i,j}-x_0)^2 + (y_{i-1,j}-y_0)^2\right). \end{cases} \tag{4.24}$$

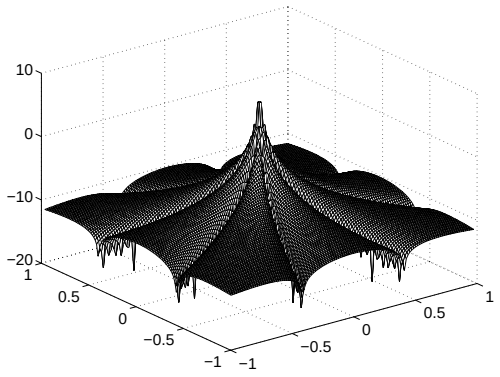


Figure 4.5: *log(|RHS|) with* $\epsilon_x = \epsilon_y = 0.5$.

Again, this right-hand side allows us to reach an accurate, discrete solution. However, this accuracy depends on the distance between (x_0, y_0) and the mesh nodes: the error becomes relatively large, when $\epsilon_x = \epsilon_y = 10^{-5}$ for example. In Figure 4.6, we show the behavior of the error with respect to the distance between the singular source point and the mesh nodes. We can see that this error is of second order with respect to the inverse of the distance $\epsilon = \epsilon_1 = \epsilon_2$. For ϵ values smaller than 10^{-6}, we cannot

compute the solution due to the ill conditioning of the right-hand side.

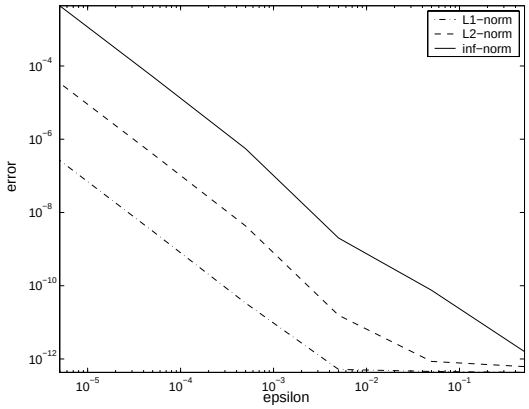


Figure 4.6: *Error on the solution with respect to* $\epsilon \equiv \epsilon_1 = \epsilon_2$.

Now we measure the error on elliptic equations with singular source terms, depending on the different discrete Dirac delta functions used. First, we list some of the norms used in the following section.

4.1.4 Some Discrete 2D norms

We need to take special care with the method of error measurement used due to its lack of uniformity. Thus, we generally choose to use a set of four different discrete norms. If the discrete error is called E: $\{E_{i,j}\}_{1\leq i,j\leq N+1}$, we use:

$$\|E\|_{L_1}^h = \frac{1}{(N+1)^2}\sum_{i=1}^{N+1}\sum_{j=1}^{N+1}|E_{i,j}|,$$

$$\|E\|_{L_2}^h = \frac{1}{N+1}\left[\sum_{i=1}^{N+1}\sum_{j=1}^{N+1}\left(E_{i,j}\right)^2\right]^{\frac{1}{2}},$$

$$\|E\|_{\infty}^h = max_{1\leq i,j\leq N+1}\left\{E_{i,j}\right\}.$$

In the last two test cases, the solution is symmetrical with respect to the origin, thus

we use an additional one-dimensional norm. This is the discrete L_2-norm of the error along the x-axis $E_{i,\frac{N}{2}+1}$ (N even):

$$|E(x,0)|_{L_2}^h = \left[\frac{1}{N+1}\sum_{i=1}^{N+1}\left(E_{i,\frac{N}{2}+1}\right)^2\right]^{\frac{1}{2}}$$

The $||.||_{L_1}^h$-norm is essentially an average of the discrete errors on the mesh. It is not especially sensitive to the singularities but gives a good indication of the global behavior of the solution. The $||.||_{L_2}^h$-norm is more sensitive to the singularities because of the square power, as well as the $|.|_{L_2}^2$-norm. Then, the $||.||_{\infty}^h$-norm is essentially measuring the error at the singularity.

4.2 Some elliptic equations with singular source terms

The two main test cases used to test the regularized delta functions on staggered grids, introduced previously, are both two-dimensional Poisson equations with singular source terms distributed along a circle. An iterative S.O.R. solver has been used for all these calculations. The accuracy of Poisson's equation with singular source terms is an important issue for the IBM in order to improve the accuracy of the pressure field around the immersed boundary.

We start with more basic 1D and 2D equations on collocated grids:

- 1D Helmoltz operator with one singular source point,
- 1D Laplace operator with two singular source points,
- 2D Laplace operator with one singular source point.

4.2.1 The 1D Helmholtz operator

Let us study the behavior of the 1D Helmholtz operator with a singular source point in the right-hand side, as presented in [59]:

$$\frac{d^2u}{dx^2}(x) - \alpha^2 u(x) = -2\alpha\delta(x - x_0), \quad x \in [-0.5, 0.5], \;\; \alpha \in \mathbb{R}_+^*; \tag{4.25}$$
$$x_0 \in [-0.5, 0.5]; \;\; u(-0.5) = e^{-\alpha|-0.5-x_0|} \;\text{ and }\; u(0.5) = e^{-\alpha|0.5-x_0|}.$$

The domain is divided into N equidistant intervals. The computed solution is compared to the exact solution (Figure 4.7):

$$u_{ex}(x) = e^{-\alpha|x-x_0|}. \tag{4.26}$$

We set the parameters to: $x_0 = 0$ and $\alpha = 60$.

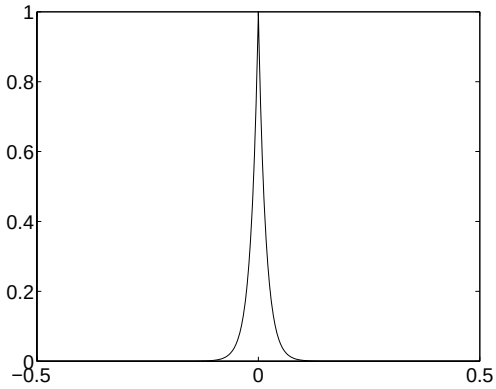


Figure 4.7: *Exact solution for problem (4.25) taking* $x_0 = 0$ *and* $\alpha = 60$.

The discrete L_2-norm is used to measure the error: the convergence order found is 1.45 with ϕ_2 and 2.00 with ϕ_4 (Figure 4.8).

The maximum error is located at the source point (Figure 4.9) and decreases exponentially with respect to the distance from it, due to the type of solution (Eq.(4.26)). Since the point loads in the IBM can be located anywhere in a cell, it is interesting to study the behavior of the error, depending on the distance between the point load

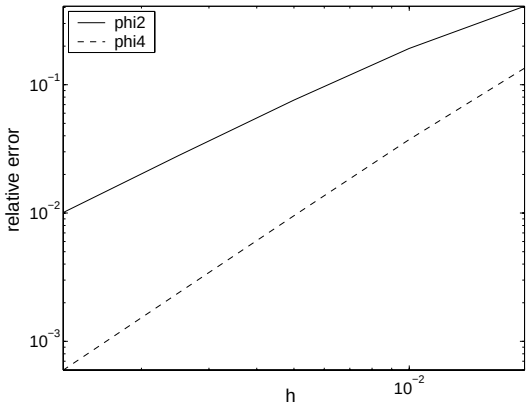


Figure 4.8: *Relative error in L_2-norm for the 1D elliptic Eq. (4.25) using ϕ_2 or ϕ_4.*

and the nodes of the mesh. In Figure 4.10, the error is plotted as a function of d, the minimum distance between x_0 and the nodes of the mesh:

$$d(x_0) = \min_{i=1,..,N+1} |(-0.5 + (i-1)h) - x_0| \,.$$

This function d spans from 0 to $\frac{h}{2}$.

Even if ϕ_4 always brings a more accurate solution than ϕ_2, we can observe that they have some extremely different behaviors as the point load moves toward the center of the interval: the solution obtained with ϕ_2 is improving as opposed to ϕ_4.

Consistently setting the Dirac delta function in the middle of the space step, will give similar orders between both delta functions. The order for the cosine function ϕ_2 is 1.39 and 1.37 for the piecewise cubic function ϕ_4.

4.2.2 The 1D Laplace operator

We solve the problem:

$$\frac{d^2u}{dx^2}(x) = \delta(x - x_0) + \delta(x + x_0), \quad x \in [-1, 1]. \tag{4.27}$$

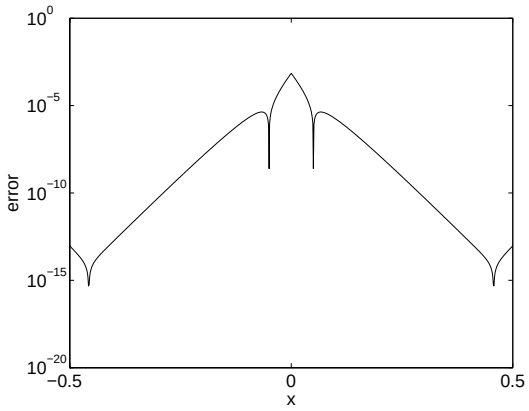


Figure 4.9: *Error of the computed solution of Eq. (4.25), using 800 intervals and* ϕ_4 *(*$x_0 = 0$*).*

$$x_0 \in (0,1); \quad u(-1) = u(1) = 0.$$

The computed solution is compared to the exact solution:

$$u_{ex}(x) \;=\; \begin{cases} x+1, & -1 \le x < -x_0; \\ 1-x_0, & -x_0 \le x < x_0; \\ -x+1, & x_0 \le x \le 1. \end{cases} \tag{4.28}$$

The measured convergence orders are very similar with ϕ_2 or ϕ_4, which is around 1.5. Again, we can observe in Figure 4.11 that the accuracy of the solution changes with respect to the distance between the point load and the mesh nodes: as this distance increases, the accuracy decreases with ϕ_4 and improves with ϕ_2.

It is interesting to note that when using the piecewise cubic function ϕ_4 and locating the point loads at some nodes of the mesh, we converge to the exact solution for every size space step, which is not the case for ϕ_2. This is due to the piecewise linearity of the solution and the fact that $\phi(-1) = \phi(+1) = 0$ for the piecewise cubic function.

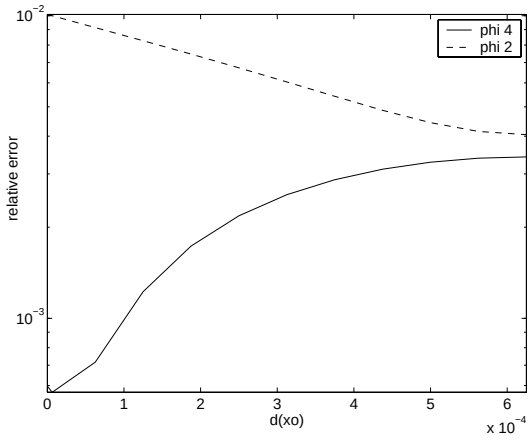


Figure 4.10: *Relative error with respect to $d(x_0)$, using ϕ_2 or ϕ_4; $N = 800$.*

4.2.3 The 2D Laplace operator with a single point load

In order to study the 2D Poisson's equation with a single point load in the right-hand side, let us recall the two-dimensional Dirac delta function. This is the product of the Delta functions corresponding to each direction:

$$\delta(x, y) = \delta(x)\delta(y) \tag{4.29}$$

We solve the problem:

$$-\Delta u(x, y) = \delta(x - x_0, y - y_0), \quad (x, y) \in \Omega = [0, 1]^2; \tag{4.30}$$

$$(x_0, y_0) \in \Omega - \partial\Omega, \quad u_{|\partial\Omega} = u_{ex|\partial\Omega}$$

This time, the exact solution is unbounded (Figure 4.12):

$$u_{ex}(x, y) = \frac{1}{2\pi} ln\left(\sqrt{(x - x_0)^2 + (y - y_0)^2}\right). \tag{4.31}$$

If the singularity is located at a mesh node, the discrete solution is not bounded and we cannot evaluate the accuracy of the computed solution by taking the discrete L_2-norm of the error over the whole domain. One possibility is taking the L_2-norm

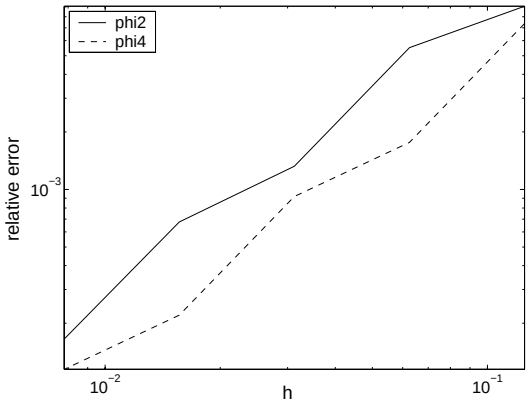


Figure 4.11: *Relative error of the computed solution for problem (4.27) with* $x_0 = 0.3$ *with* ϕ_2 *or* ϕ_4.

of the error over Ω/A, where A is a circle centered at the singularity location with a small fixed radius.

When the point load is located at a mesh node, we get a convergence order of one (Figure 4.13) with three different discrete Dirac delta functions used: ϕ_2, ϕ_3 and ϕ_4. However, in this unbounded problem, the best result is obtained with the $1.5h$ support function ϕ_3. Now if we place the point load between the mesh nodes, the use of ϕ_4 does improve the accuracy of the solution (Figure 4.14).
The next two test cases bring us closer to the IBM. First, we present the connection between elliptic equations with singular source terms and the IBM.

4.2.4 On The pressure equation in the IBM

If we use the explicit projection scheme for the NS equations, the pressure equation in a closed box is:

$$\Delta P^{n+1} = \frac{\rho}{\Delta t}\nabla.V^* \ \text{ in } \ \Omega, \quad \frac{\partial P^{n+1}}{\partial \eta} = 0 \ \text{ on } \ \partial\Omega. \tag{4.32}$$

The vector η is the outward normal vector to $\partial\Omega$, the outside boundary of the domain.

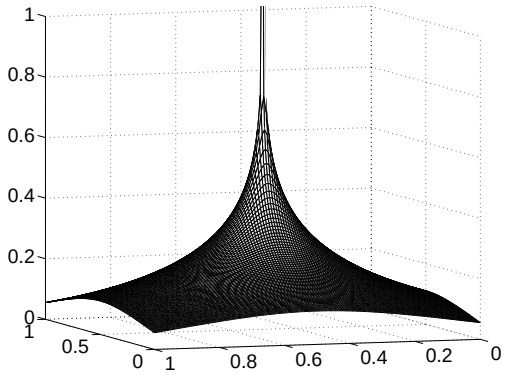


Figure 4.12: *Exact solution of problem (4.30), with* $(x_0, y_0) = (0.5, 0.5)$.

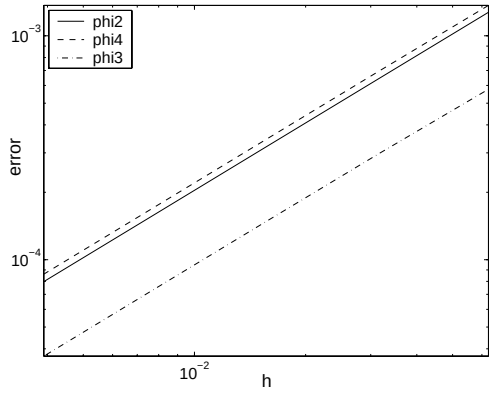


Figure 4.13: *Error over* Ω/A*, point load located at a mesh node.*

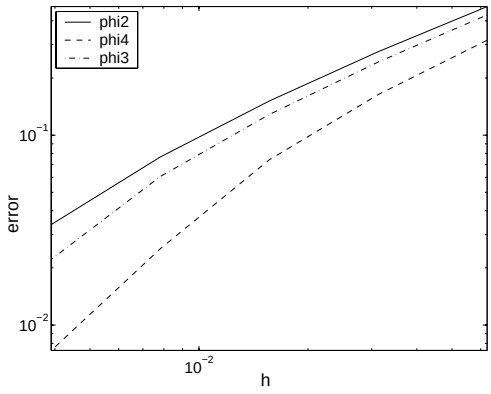


Figure 4.14: *Error over* Ω*, point load not located at a mesh node.*

We have, from the prediction step of the scheme;

$$V^* = V^n + \Delta t \left[-(V^n.\nabla)V^n + \frac{1}{\rho}(\mu \Delta V^n + F^n) \right]. \tag{4.33}$$

So: $V^* = \Phi(V^n) + \frac{\Delta t}{\rho} F^n$, where $\Phi(V^n)$ is a smooth regular vector function and F^n, the force term, is a singular vector field. We focus on the problem:

$$\Delta P = \nabla.F \ \text{ in } \ \Omega, \quad \frac{\partial P}{\partial \eta} = 0 \ \text{ on } \ \partial\Omega, \tag{4.34}$$

with $F = [F_1, F_2]^T$. Here, we describe the two-dimensional case for simplicity.

The error in the IBM is essentially introduced by this singularity in the right-hand side. Firstly, the force term is regularized and thus, the error is already smeared along Γ in the right-hand side. Then, this error is smeared again by the discrete divergence and inverse Laplace operators:

$$P = \Delta^{-1}(\nabla.F). \tag{4.35}$$

What should be a jump in the solution, is a slope whose relatively small gradient is a bottleneck to the accuracy if the IBM. This is why it is important to have a solution that is as close as possible to the theoretical jump. In [66], we find a good analysis of this jump. If we call $n(s)$ and $t(s)$ the unit normal and tangential vectors to the immersed boundary, we can split the force density of the IBM into a normal and a tangential component:

$$f(s) = f_n(s) + f_t(s), \tag{4.36}$$

$$\text{with } \; f_n(s) = f(s).n \; \text{ and } \; f_t(s) = f(s).t.$$

If $[\![.]\!]$ denotes the jump across the boundary, then:

$$\begin{aligned} [\![p]\!](s) &= f_n(s), \\ \left[\!\left[\frac{\partial p}{\partial n}\right]\!\right](s) &= \frac{\partial}{\partial s} f_t(s), \\ [\![u]\!](s) &= 0, \\ \mu \left[\!\left[\frac{\partial u}{\partial n}\right]\!\right](s) &= -f_t(s)t(s) \end{aligned} \tag{4.37}$$

The immersed interface method (IIM) of R. Leveque and J.Lee [67] incorporates these jump conditions into the pressure solver in order to eliminate the discrete Dirac delta functions. However, this implies the splitting of the force term and a infinitely thin immersed boundary. We do not want to explicitly use these jump conditions in the

pressure solver but rather, use a general method for solving elliptic equations with singular source terms.

The first issue addresses how well we can solve Poisson's equation with a collection of Dirac delta functions distributed along a closed curve Γ:

$$\Delta P = F(x_1, x_2)\delta(x_1, x_2, \Gamma), \quad (x_1, x_2) \in \Omega. \tag{4.38}$$

This is test case 1, taking $F(x_1, x_2) = 1$ and homogeneous Dirichlet boundary conditions.

The second issue is then how to solve Poisson's equation with the divergence of test case 1's right-hand side, that is, dealing with a collection of dipoles along a closed curve:

$$\Delta P = \nabla . \left(F(x_1, x_2)\delta(x_1, x_2, \Gamma)\right), \quad (x_1, x_2) \in \Omega. \tag{4.39}$$

This relates to test case 2, taking $F(x_1, x_2) = 2[x_1, x_2]^T$ and homogeneous Dirichlet conditions.

Now we show that test case 1 and test case 2 are closely related. In the IBM, $f = [f_1, f_2]^T$ corresponds to the elastic force density along the immersed boundary Γ. We have:

$$F(x_1, x_2) = \int_\Gamma f(s)\delta\left(x_1 - X_1(s)\right)\delta\left(x_2 - X_2(s)\right) ds, \tag{4.40}$$

and:

$$\nabla . F(x_1, x_2) = \int_\Gamma \left(\frac{f_1(s)}{x_1 - X_1(s)} + \frac{f_2(s)}{x_2 - X_2(s)}\right) \delta\left(x_1 - X_1(s)\right)\delta\left(x_2 - X_2(s)\right) ds. \tag{4.41}$$

But solving $\Delta P = F$ is equivalent to solving $\Delta P = \nabla.F$, due to commutativity of the discrete divergence and Laplace operators:

$$\nabla_h.\Delta_h = \Delta_h \nabla_h.$$

So the problem:

$$\Delta P = \nabla.F$$

is equivalent to:

$$\begin{cases} \Delta Q_1 = F_1, \\ \Delta Q_2 = F_2 \\ P = \nabla.Q, \end{cases} \tag{4.42}$$

with $Q = [Q_1, Q_2]^T$ and corresponding boundary conditions. Thus, the problem of Poisson's equation with the divergence of shifted Dirac delta functions in the right-hand side (4.34), studied with test case 2, is similar to the problem of the Poisson equation with shifted Dirac delta functions in the right-hand side (4.42), studied with test case 1.

4.2.5 Test case 1

The first test case is as seen in [111]:

$$-\Delta u(x,y) = \delta(x,y,\Gamma), \quad (x,y) \in \Omega = [-1,1]^2; \tag{4.43}$$

$$\Gamma = \left\{(x,y) \in \Omega / x^2 + y^2 = r^2\right\}, \quad r < 1, \quad u_{|\partial\Omega} = u_{ex|\partial\Omega}$$

$$u_{ex}(x,y) = \begin{cases} 1 - \frac{1}{2} ln\left(\frac{1}{r}\sqrt{x^2+y^2}\right), & \text{if} \quad x^2 + y^2 > r^2; \\ 1, & \text{if} \quad x^2 + y^2 \leq r^2. \end{cases} \tag{4.44}$$

We fix the parameter r to be equal to $\frac{1}{2}$. A discrete collection of M Dirac delta functions along the line Γ is used. M needs to be a large number: if h is the space

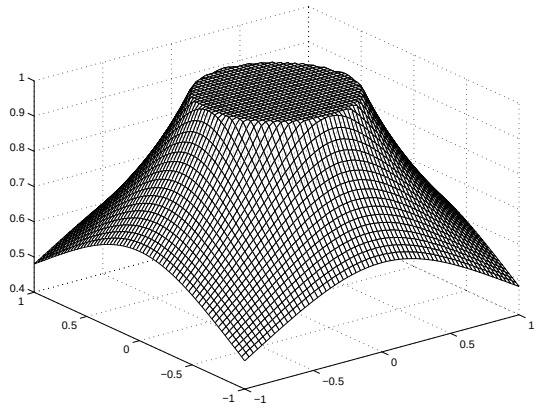


Figure 4.15: *Exact solution u_{ex} for problem (4.43), taking a radius r of* 0.5.

step and h_Γ the curvilinear distance between the delta functions along Γ, we take M such that $h_\Gamma < \frac{1}{2}h$ [89]. Here is the discrete expression of $\delta_h(x, y, \Gamma)$:

$$\delta_h(x,y,\Gamma) = \frac{\pi}{M}\sum_{i=1}^{M}\delta_h\left(x - \frac{1}{2}cos\left(\frac{2(i-1)\pi}{M}\right), y - \frac{1}{2}sin\left(\frac{2(i-1)\pi}{M}\right)\right) \tag{4.45}$$

$$= \frac{\pi}{M}\sum_{i=1}^{M}\delta_h\left(x - \frac{1}{2}cos\left(\frac{2(i-1)\pi}{M}\right)\right)\delta_h\left(y - \frac{1}{2}sin\left(\frac{2(i-1)\pi}{M}\right)\right).$$

The error of the computed solution is located along Γ (Figure 4.16 and Figure 4.17) and we can notice in Figure 4.18 and Figure 4.19 how different the shape of the error is depending on which delta function is used.

This error is strongly dependent on the distance between Γ and the Cartesian mesh nodes.

We can see that with the three two-dimensional norms the convergence order is similar using the four different discrete delta functions: around 2 with the L_1-norm, 1.5 with the L_2-norm and 1 with the ∞-norm (see Table 4.1). In Figure 4.24, we plot the contour of the space-dependent asymptotic order α of the solution: it is nearly everywhere around 2, except in the neighborhood of Γ where it drops to reach 1 exactly. To compute the asymptotic order, we interpolate the solutions obtained

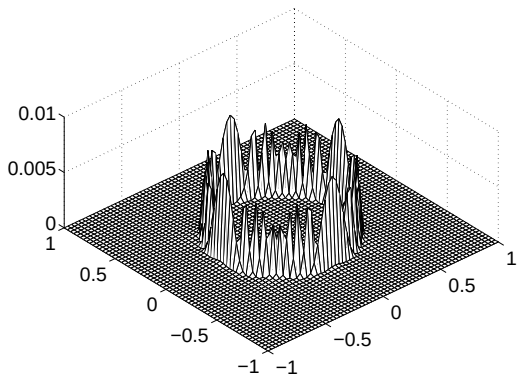


Figure 4.16: *Error of the computed solution for problem (4.43); $N = 64$, ϕ_2.*

with different space steps h_k $(k = 1, 2)$, on a finer 192×192 grid and then combine them, using the formula:

$$u_{i,j}^{h_k} = u_{i,j}^{ex} + c_{i,j} h_k^{\alpha_{i,j}} + o\left(h_k^{\alpha_{i,j}}\right), \quad k = 1, 2, \tag{4.46}$$

where u^{ex} is the exact solution and $c_{i,j}$ a constant. We can see in Figure 4.24 that at some points, the order blows up because the computed solution crosses the exact solution. This happens along Γ on both sides and along four "petal" shaped lines.

delta function	$\|.\|_{L_1}^h$	$\|.\|_{L_2}^h$	$\|.\|_{\infty}^h$	$\|.\|_{L_2}^h$
ϕ_1	1.96	1.51	1.00	1.51
ϕ_3	1.88	1.45	0.92	2.01
ϕ_4	1.98	1.54	0.98	1.51
ϕ_5	1.97	1.53	1.02	1.50

Table 4.1: *Convergence order of the solution of problem (4.43) using ϕ_1, ϕ_3, ϕ_4, ϕ_5 and in four different norms.*

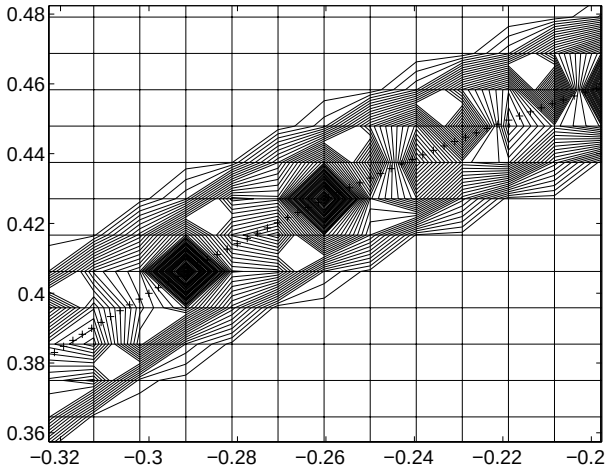


Figure 4.17: *Detail of the contour of the error and location of the delta functions (+), ϕ_1.*

4.2.6 Test case 2

The second test case is:

$$-\Delta u(x,y) = \nabla.F(x,y), \quad (x,y) \in \Omega = [-1,1]^2, \quad u_{|\partial\Omega} = 0, \tag{4.47}$$

$$F = [F_1, F_2]^T, \quad F_i(x,y) = \int_\Gamma f_i(s)\delta(x,y,\Gamma)ds, \quad i = 1,2,$$

$$[f_1(s), f_2(s)] = 2[x(s), y(s)], \quad s \in \Gamma, \quad \Gamma = \left\{(x,y) \in \Omega / x^2 + y^2 = \frac{1}{4}\right\}.$$

The exact solution has a "hat" shape (Figure 4.25):

$$u_{ex}(x,y) \quad = \quad \begin{cases} 1, & \text{if } \; x^2 + y^2 \leq \frac{1}{4} \\ 0, & \text{if } \; x^2 + y^2 > \frac{1}{4} \end{cases} \tag{4.48}$$

Again, this test case is relevant because the right-hand side corresponds to the typical irregular part of an IBM right-hand side in the pressure correction step of the NS basic projection scheme. The results of this test case clearly show why the IBM is first order accurate around the immersed boundary. Poisson's equation, with the divergence of

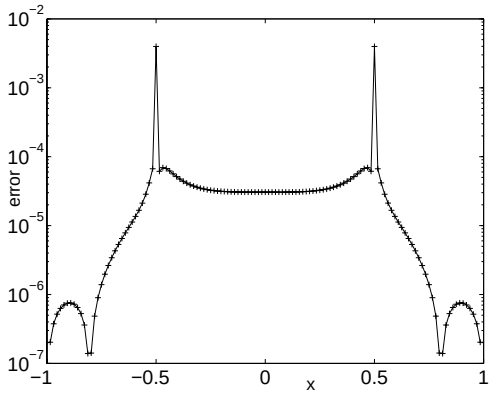


Figure 4.18: *Error along the x-axis for problem (4.43); $N = 128$, ϕ_2.*

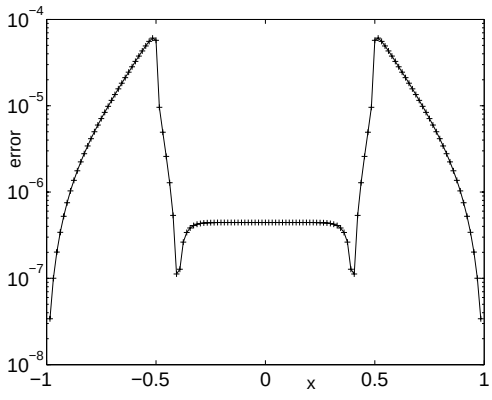


Figure 4.19: *Error along the x-axis for problem (4.43); $N = 128$, ϕ_4.*

singular source points along a curve, is the main issue for the accuracy of the method. As we did for the test case 1, we measure the error using the different discrete norms and discrete Dirac delta functions. Similarly as for test case 1, the shape of the error depends on the function used.

delta function	$\|.\|^h_{L_1}$	$\|.\|^h_{L_2}$	$\|.\|^h_{\infty}$	$\|.\|^h_{L_2}$
ϕ_1	0.99	0.51	-0.01	0.47
ϕ_3	1.00	0.52	-0.01	0.48
ϕ_4	1.03	0.54	-0.02	0.48
ϕ_5	0.60	0.50	-0.00	0.44

Table 4.2: *Convergence order for the solution of problem (4.47) using ϕ_1, ϕ_3, ϕ_4, ϕ_5 and in four different norms.*

We can see in Table 4.2 that the orders are smaller than with the previous test case. This is particularly true with ϕ_5, because of its large support: the discrete divergence operator smears the error at the singularity. This shows the difficulty of solving equations with dipole singularities.

After looking at the discretization of the Dirac delta function in order to improve the

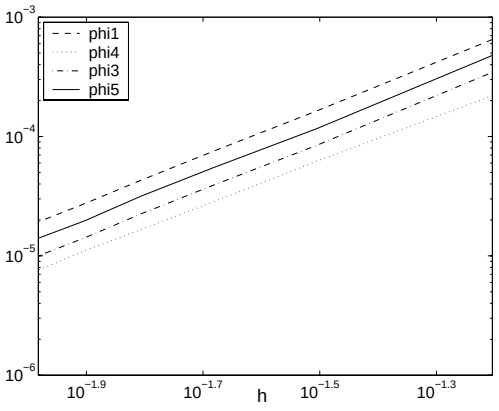


Figure 4.20: $\|.\|_{L_1}^h$ *norm of the solution error for* ϕ_1, ϕ_3, ϕ_4 *and* ϕ_5.

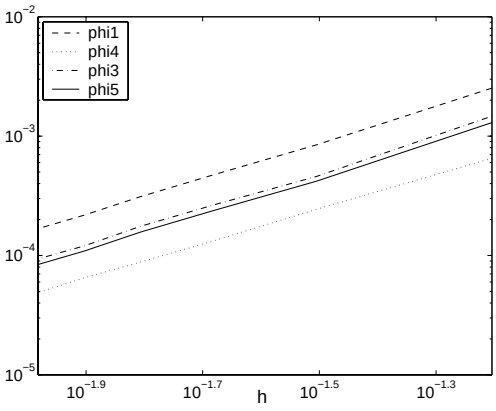


Figure 4.21: $\|.\|_{L_2}^h$ *norm of the solution error for* ϕ_1, ϕ_3, ϕ_4 *and* ϕ_5.

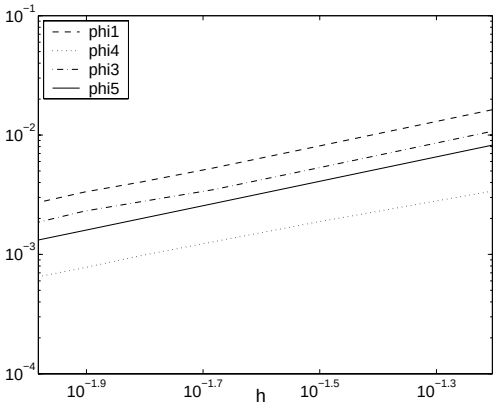


Figure 4.22: $\|.\|_{\infty}^h$ *norm of the solution error for* ϕ_1, ϕ_3, ϕ_4 *and* ϕ_5.

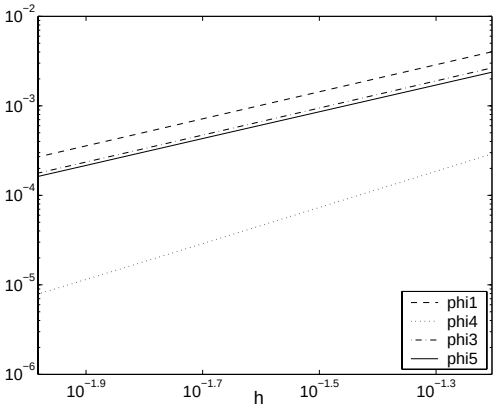


Figure 4.23: $|.|_{L_2}^h$ *norm of the solution error for* ϕ_1, ϕ_3, ϕ_4 *and* ϕ_5.

accuracy of the solution of elliptic equations with singular source terms, we implement the IBM with the piecewise cubic Dirac delta function.

4.3 The IBM case

4.3.1 The Piecewise cubic Dirac delta function

We implement the IBM with a staggered mesh in the 2D "bubble" test case using the piecewise cubic discrete Dirac delta function ϕ_4.

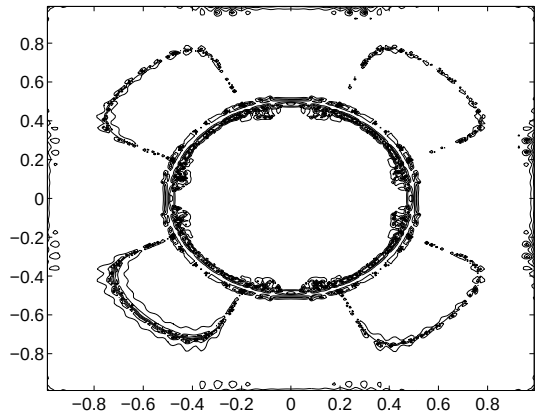


Figure 4.24: *Contour of the convergence order of the computed solution of problem (4.43) using* ϕ_1.

To evaluate the accuracy of the method, we study the horizontal diameter of the bubble with respect to time (Figure 4.29). We can then compare the different diameters obtained with different discrete Dirac delta functions. The reference solution is a diameter, function of time, obtained with a very small space step.

First, we checked that the method using ϕ_4 converges to the same solution as the one uobtaines with ϕ_2: we compute the "bubble" test case with both delta functions, ϕ_4 or ϕ_2, a very small time step $\Delta t = 10^{-5}$, $\sigma = 10^4$ and evaluate the relative error between the diameters of the bubble in both cases over hundred time steps.

We can observe in Figure 4.30 that they converge to the same solution. The order of convergence of the relative error between both solutions, obtained with ϕ_2 or ϕ_4, is one.

The next question involves knowing if the piecewise delta function improves the accuracy of the IBM. We computed the "bubble" test case with different space steps and a small constant time step $\Delta t = 10^{-5}$ and compared with a reference solution obtained with the same time step and the space step $h = \frac{1}{96}$.

We can clearly see in Figure 4.31 the improvement of the piecewise cubic delta function

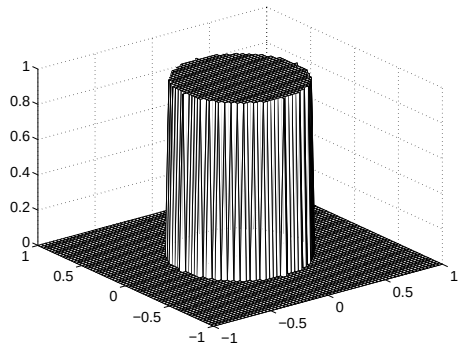


Figure 4.25: *Exact solution for test-case 2 (4.47).*

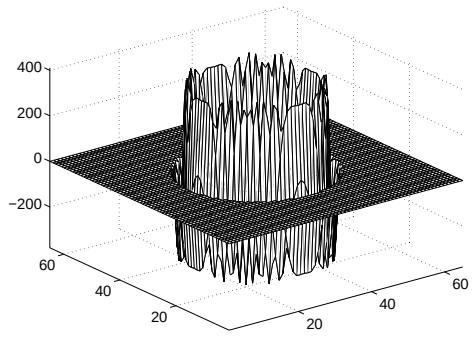


Figure 4.26: *Typical right-hand side for test case 2.*

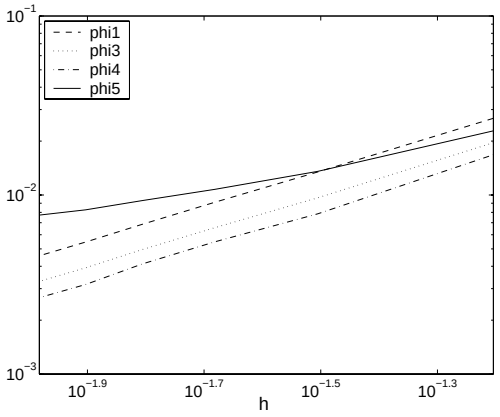


Figure 4.27: $\|.\|_{L_1}^h$ *norm of the solution error for* ϕ_1, ϕ_3, ϕ_4 *and* ϕ_5.

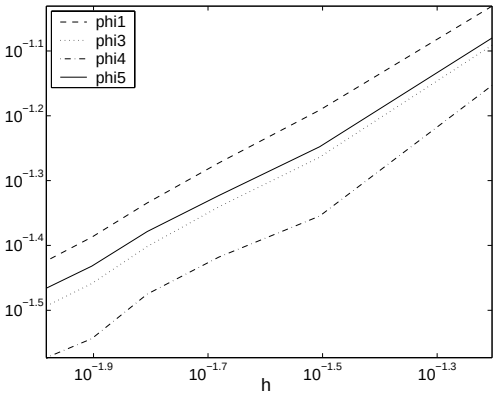


Figure 4.28: $\|.\|_{L_2}^h$ *norm of the solution error for* ϕ_1, ϕ_3, ϕ_4 *and* ϕ_5.

in this case. The method using the usual cosine delta function is of order one while the other is of order two. We can take a relatively larger space step with the piecewise cubic function than with the cosine one in order to get the same level of error on the solution. For example, with $N = 32$, $\Delta t = 10^{-5}$ and $T = 0.01$, we can use $h = 0.06$ with the piecewise cubic function instead of $h = 0.03$ with the cosine one, in order to get a 2% error on the diameter.

Figure 4.32 shows the convergence of the test case using ϕ_4 measured with two different norms. The results are not very clear when h is very small due to the fact that

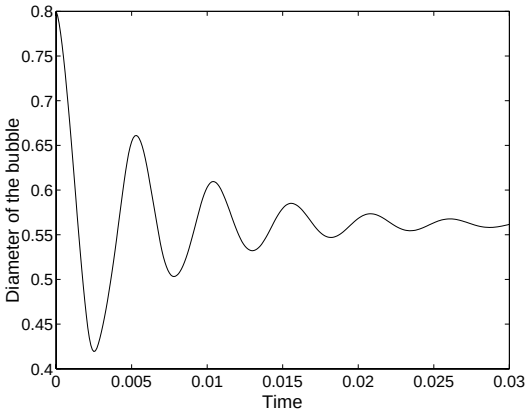


Figure 4.29: *Example of the horizontal diameter of the bubble with respect to time.*

we do not compare to an exact solution but a solution obtained with a very small space step. We compute the slope of the lines in the graph above when h is not too small and then get the second order of convergence.

If the piecewise cubic delta function improves the accuracy of the solution, at least on a short time range, its behavior regarding the stability and the volume conservation must be studied. The stability results will be presented in the next chapter, while we focus now on the conservative issues.

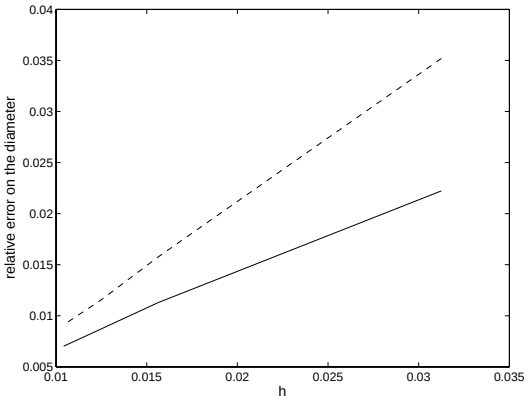


Figure 4.30: *Difference between the solutions (diameter of the "bubble" obtained with ϕ_2 and ϕ_4, over hundred time steps with $\Delta t = 10^{-5}$, $\sigma = 10^4$); solid line: L_2-norm, dotted line: ∞-norm.*

4.3.2 Conservative issues

4.3.2.1 Volume variations

In the process of evaluating the accuracy of the IBM, we do not compare the evolution of the moving boundary on a very long time range. The final time is: $T = 0.01$. If we measure the diameter of the bubble after the equilibrium is reached, we can study the property of volume conservation of the immersed boundary. Let us compare the behaviors of the bubble from $t = 0.04$ to $t = 0.06$, with $N = 32$, $M = 6N$, $\sigma = 10^4$, $\Delta t = 510^{-5}$, and for the different delta functions: cosine ϕ_2, piecewise cubic ϕ_4 and double-layer piecewise cubic (Figure 4.35).

The idea of the double-layer "bubble" test case (Figure 4.33) that still models the case of an immersed boundary with no thickness, is to have two layers of elastic boundary separated by a very small distance $h_\Gamma = \frac{1}{M}$, which is the discretization step along the immersed boundary (the single layer boundary has a width of zero which is not physically realistic). The double-layer does not change the solution when the immersed boundary is moving (Figure 4.34).

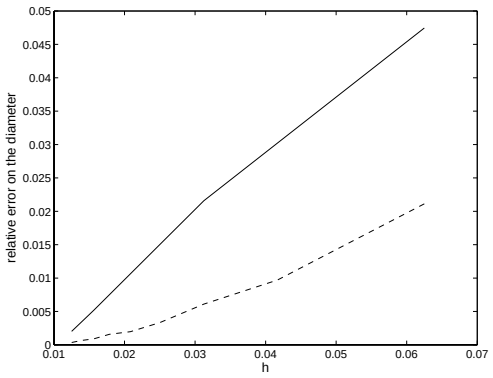


Figure 4.31: *Relative error in L_2-norm of the diameter using ϕ_2 (solid line) or ϕ_4 (dotted line).*

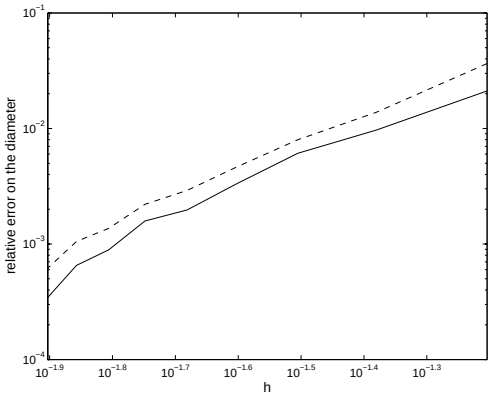


Figure 4.32: *Relative error in the diameter using ϕ_4; L_2-norm (solid line), ∞-norm (dotted line).*

Figure 4.35 shows a large difference in behavior between the different cases. The piecewise cubic function ϕ_4 is less conservative than the traditional cosine function ϕ_2, while the double-layer IBM has a better conservation of volume property.

In order to study the improvement, volume-wise, in the double-layer IBM, we compared the single-layer with the double-layer "bubble" test case for the two different delta functions: ϕ_2 in Figure 4.36 and ϕ_4 in Figure 4.37. The test cases are run from time iteration 500 to 2000, with $N = 32$, $M = 6N$, $\sigma = 10^4$, $\Delta t = 5.10^{-5}$.

We can conclude from these graphs that the use of the piecewise cubic delta function alters the conservation of volume property, compared to the cosine delta function. The double layer IBM has a better behavior than both other cases. It requires twice as many computations for the boundary treatment, although this treatment is very light, compared to the flow computations. The order of the method using this double layer ϕ_4 is 2.46 (Figure 4.38). Again, the error is computed with respect to the solution obtained with a very fine grid and not an exact solution.

Now we present a method that fixes the conservation issue in the case of a 2D immersed boundary. The method could be extended to 3D, for example for a flow in an

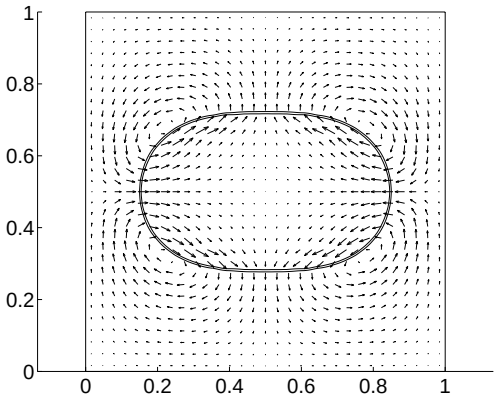


Figure 4.33: *Double-layer "bubble" test-case.*

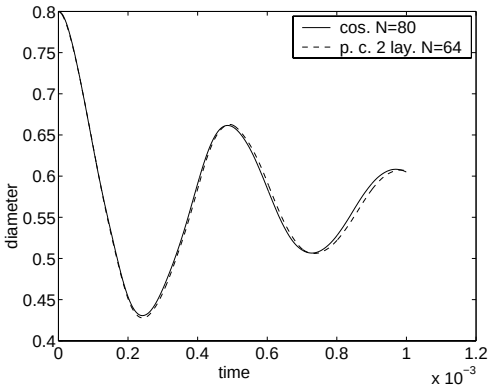


Figure 4.34: *Diameter of the bubble with respect to time with single-layer ϕ_2 "bubble" or the double-layer ϕ_4 one.*

elastic channel: it only requires a defined volume with an uniform distribution of the discrete points along the immersed boundary at rest, because of the use of Fourier expansions.

4.3.2.2 A 2D area conservation method based on constrained optimization.

This method is based on a Fourier expression of the position vector of the immersed boundary. Therefore, it is well suited for the 2D "bubble" test-case, since the position vector is periodic and the volume inside the "bubble" is constant.

The reason we can express the moving boundary position vector into a Fourier expansion, even if the points are not equally spaced, is that the transformation of the equally spaced points of the equilibrium position is a smooth continuous transformation. This is due to the no-slip boundary condition: the moving boundary points move at the same speed as the particles of fluid surrounding them and the fluid velocity field is smooth and continuous.

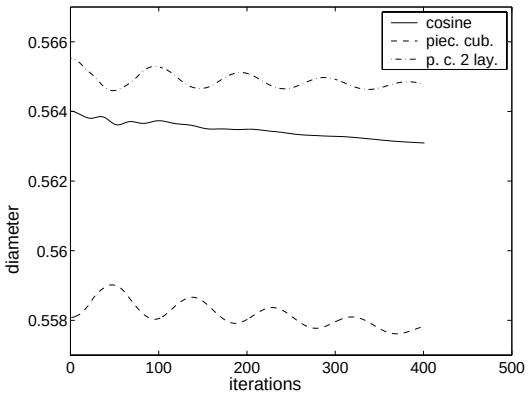


Figure 4.35: *Diameter of the bubble from* $t = 0.04$ *to* $t = 0.06$*, for the different delta functions:* ϕ_2*,* ϕ_4 *and double-layer piecewise cubic.*

Let us call $X(s), \quad 0 \leq s < 1$, the position vector of the moving elastic boundary. Because of the discretization, we deal with the set:

$$\{X_i\}_{0 \leq i \leq M-1} = \{X_{1,i}, X_{2,i}\}_{0 \leq i \leq M-1}. \tag{4.49}$$

$\{X_{1,i}\}_{0 \leq i \leq M-1}$ is the vector of the horizontal components of the moving points and $\{X_{1,i}\}_{0 \leq i \leq M-1}$ is the vector of their vertical components. We assume that M is even and then take $K = \frac{M}{2}$. The Fourier coefficients are:

$$\{\alpha_k\}_{0 \leq k \leq K} = \{\alpha_{1,k}, \alpha_{2,k}\}_{0 \leq k \leq K},$$

$$\text{with: } \alpha_{j,k} = \{\alpha^A_{j,k}, \alpha^B_{j,k}\}, \quad 0 \leq k \leq K, \quad j = 1, 2.$$

For $0 \leq k \leq K$, we have:

$$\alpha^A_{j,k} = \sum_{i=0}^{M-1} X_{j,i} cos\left(2\pi k \frac{i}{M}\right), \quad j = 1, 2.$$

For $1 \leq k \leq K - 1$, we have:

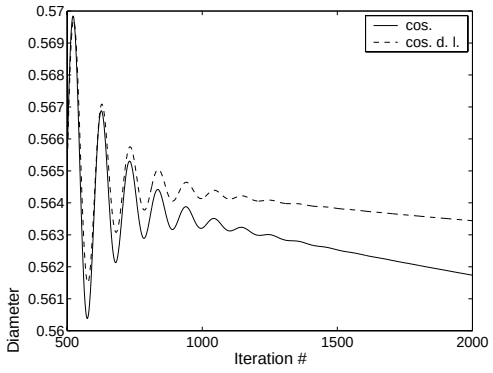


Figure 4.36: *Diameter of the bubble with ϕ_2: single and double-layer.*

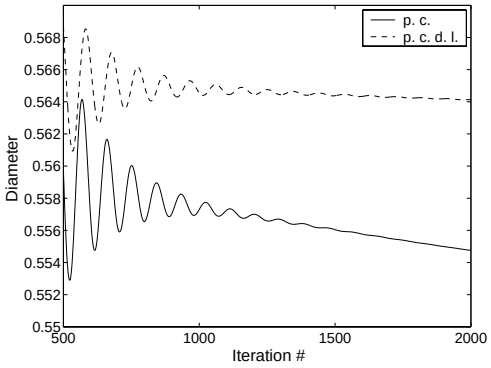


Figure 4.37: *Diameter of the bubble with ϕ_4: single and double-layer.*

$$\alpha^B_{j,k} = \sum_{i=0}^{M-1} X_{j,i} sin\left(2\pi k \frac{i}{M}\right), \quad j = 1, 2.$$

$$\alpha^B_{j,0} = \alpha^B_{j,K} = 0, \quad j = 1, 2.$$

Then the Fourier expansion of X is, for $j = 1, 2$ and $0 \leq i \leq M-1$:

$$\hat{X}_{j,i}(\alpha) = \frac{1}{M}\left[\alpha^A_{j,0} + 2\sum_{k=1}^{K-1}\left(\alpha^A_{j,k} cos\left(2\pi k\frac{i}{M}\right) + \alpha^B_{j,k} sin\left(2\pi k\frac{i}{M}\right)\right) + \alpha^A_{j,K}(-1)^i\right]$$

It is easy to compute the analytic area of the "bubble" from the Fourier modes with a basic curvilinear integration, we get:

$$Area(\alpha) = \frac{4\pi}{M^2}\sum_{k=1}^{K-1} k\left(\alpha^A_{1,k}\alpha^B_{2,k} - \alpha^A_{2,k}\alpha^B_{1,k}\right)$$

If the area was initially V_0, we want to have the function $S(\alpha)$ to be null:

$$S(\alpha) = Area(\alpha) - V_0 = 0.$$

Since we want a change of the position vector that is as small as possible, we try to do the least square minimization of the position change, constrained with the area preservation. The function to minimize is:

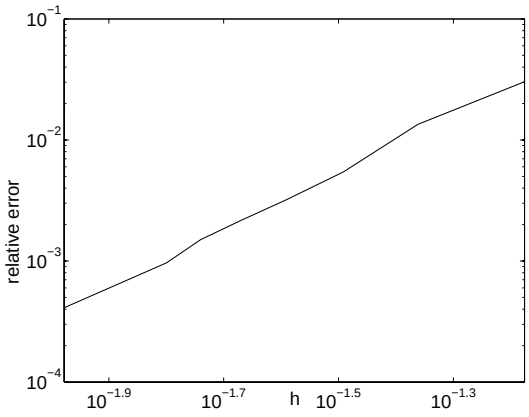


Figure 4.38: *Error of the diameter of the double-layer "bubble" over a time range, compared to the results obtained with* $N = 120$.

$$F(\alpha) = \parallel \{X_i - \hat{X}_i(\alpha)\}_{0 \le i \le M-1} \|_2^2$$

The functions $F(\alpha)$ and $S(\alpha)$ are easy to compute as well as $\nabla F(\alpha)$, H_F (the Hessian of $F(\alpha)$, which does not depend on α), $\nabla S(\alpha)$.

To summarize the problem, we want to do this minimization:

$$min_\alpha F(\alpha), \text{ such that } S(\alpha) = 0, \text{ using } \nabla F(\alpha), \ H_F \text{ and } \nabla S(\alpha),$$

starting with the initial value $\alpha_0 = \alpha(X)$.

In the "bubble" test-case, this minimization is done using the MATLAB function *fmincon* at the end of each time step, after the position update. It is a relatively fast process.

At the same time, it is easy to filter the high wave frequencies of the position vector by imposing $\alpha_k = 0$ for $k > K_f$ ($K_f < K$) or by multiplying it by a function that decreases sharply from one to zero around a certain wave length number. These high

wave frequency oscillations are created mainly by the stiff elastic force term. Thanks to the Fourier filter, the maximum time step can be increased by a factor 3 or 4 in a stiff case (an elastic coefficient $\sigma = 10000$) of the "bubble" test-case. However, the Fourier filter by itself (without imposing the volume conservation) alters the volume property of the IBM.

Now we study an extrapolation technique, the Multigrid/τ-extrapolation, with the same motivation.

4.4 The Multigrid/τ-extrapolation

The τ–extrapolation [7, 63] is a modified multigrid method that improves the convergence order of a discrete problem. It is based on the Richardson extrapolation technique. It combines two solutions obtained on different grids in order to correct the fine grid solution but requires knowledge of the order of the first asymptotic expansion term, which can be evaluated experimentally. This study is made on a collocated grid.

4.4.1 The Richardson extrapolation technique

Let us assume we know the asymptotic convergence order α of the discrete solution on a uniform mesh:

$$u_H = u^* + cH^\alpha + o(H^\alpha), \tag{4.50}$$

where u^* is the exact solution and u_H the solution obtained with a space step H. Now we take a solution obtained with a smaller space step $h = \frac{H}{2}$, we have:

$$u_h = u^* + ch^\alpha + o(h^\alpha). \tag{4.51}$$

Eq. (4.50) can be written:

$$u_H = u^* + 2^\alpha c h^\alpha + o(h^\alpha). \tag{4.52}$$

If we combine Eq.(4.51) and Eq.(4.52), we can get rid of the h^α term and get:

$$\frac{2^\alpha}{2^\alpha - 1} u_h - \left(1 - \frac{2^\alpha}{2^\alpha - 1}\right) u_H = u^* + o(h^\alpha). \tag{4.53}$$

The order of the discrete solution has been improved. Again, it is important to note that this method requires to know the asymptotic order α.

4.4.2 The Algorithm

This part needs to be clarified

Here is the τ–extrapolation multigrid algorithm for the problem $Au = f$:

- 1 - pre-smoothing step : $u_h = S^{\nu 1}(A_h, u_h, f_h)$.
- 2 - $u_h = u_h + I_H^h A_H^{-1} \left(\left(\frac{2^\alpha}{2^\alpha - 1}\right) \hat{I}_h^H (f_h - A_h u_h) + \left(1 - \frac{2^\alpha}{2^\alpha - 1}\right) (f_H - A_H I_h^H u_h) \right)$.
- 3 - post-smoothing step : $u_h = S^{\nu 2}(A_h, u_h, f_h)$.

with these characteristics in most cases:

- I_H^h is a trilinear interpolation prolongation operator.
- $\hat{I}_h^H$ is a full weighting restriction operator.
- I_h^H is a full injection prolongation operator.
- (ν_1, ν_2), the number of smoothing steps per iteration, is small (≤ 2).

The efficient convergence property of the multigrid methods is due to the fact that the smoothing iterations improve the high frequency modes of the discrete solution, while the coarse grid correction improves its low frequency modes. This is especially true for the stiff elliptic problems solved in the IBM.

In the τ–extrapolation technique, the Richardson extrapolation linear combination significantly improves the discretization order of the coarse grid correction. This is the idea of the double discretization. A high order discretization scheme is used on the coarse grid, different from the scheme used for calculating the residuals transferred to the coarse grid. The smoothing process uses the low order discretization scheme too, which implies that two discrete problems with slightly different fixed points are solved. The τ–extrapolation is a special case of the double discretization method where we use the Richardson extrapolation technique to change the discretization order of the coarse grid. The analytic solution needs to be smooth enough, and the restrictions operator needs to be chosen carefully enough, for the τ–extrapolation to improve the regular multigrid method.

A specificity of the τ–extrapolation applied to problems with singular source points is that we use f_H instead of $\hat{I}_h^H f_h$ at the coarse grid correction step. f_H is the discretization of the right-hand side using the discrete Dirac delta functions that have a $2H = 4h$ support, while f_h is evaluated using the same kind of delta function but with a $2h$ support. This is easy to implement and saves an interpolation process per multigrid iteration.

4.4.3 Numerical results

4.4.3.1 The 1D Helmholtz operator

We measure the error of the solution of the Helmoltz problem seen in sec. 4.2.1 by using or not the extrapolation technique. We can see in Figure 4.39 that the order is improved from 2.0 to 3.4 in this case, using the L_2-norm.

Then, we count the number of operations: it represents the number of times the values at the nodes are updated, but does not take into account the extrapolation and interpolation operations made in the multigrid algorithms in order to switch

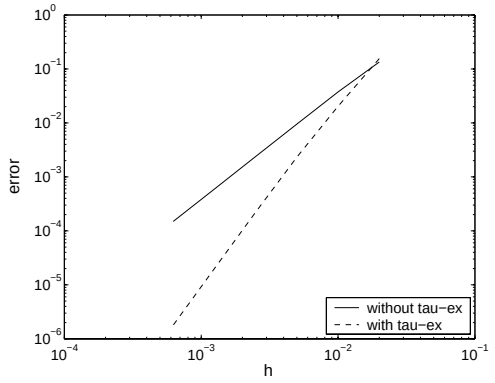


Figure 4.39: *Error in L_2-norm of the method for the multigrid algo. with or without the τ-ex. and using ϕ_4.*

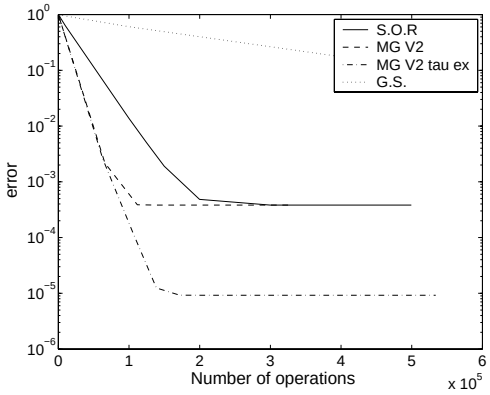


Figure 4.40: *Error in L_2-norm with respect to the number of operations with the 4 solvers S.O.R., Multigrid V2, Multigrid/τ-ex. and Gauss-Seidel. $N = 1000$, $x_0 = 0$ and we use ϕ_4.*

from one grid to another. The multigrid algorithm implemented here is a classic V-shaped algorithm with two levels. We see in Figure 4.40 that with the τ-extrapolation technique, we benefit at the same time from a fast convergence and an improved accuracy.

4.4.3.2 Test case 1

The τ-extrapolation does not improve the convergence order in test case 1 except with the 1D $|.|^h_{L_2}$ norm: we get 2.0 without the τ-extrapolation and 2.8 with (Figure 4.44).

Again we can observe in Figure 4.45 that with the τ-extrapolation technique, we benefit at the same time from fast convergence and improved accuracy. However, this extrapolation technique improves the solution only at a certain distance from Γ, where the asymptotic order is two and not one. Basically, the technique narrows down the neighborhood of Γ that has a large error.

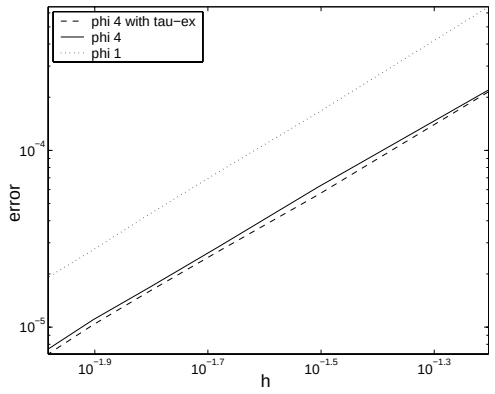


Figure 4.41: $\|.\|_{L_1}^h$*-norm of the error.*

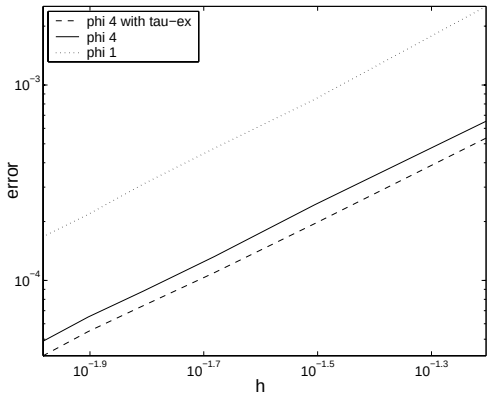


Figure 4.42: $\|.\|_{L_2}^h$*-norm of the error.*

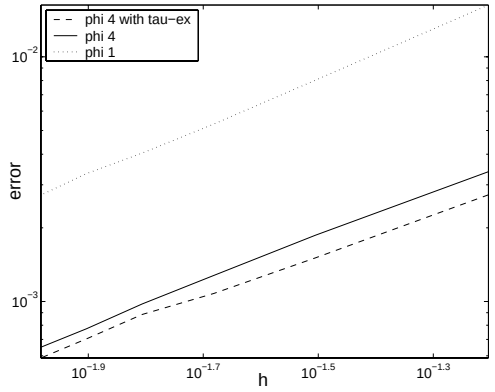


Figure 4.43: $\|.\|_{\infty}^h$*-norm of the error.*

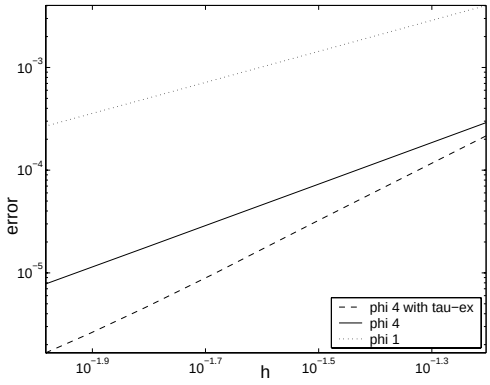


Figure 4.44: *Horizontal* $|.|_{L_2}^h$*-norm of the error.*

This technique does not improve the solution for test case 2 (sec. 4.2.6) since the error is then, essentially concentrated around Γ and null elsewhere, the asymptotic order cannot be known. This is for this same reason that the τ-extrapolation pressure solver does not significantly improve the IBM: the bottleneck of the method for the accuracy is the large error and the first degree convergence order along the immersed boundary Γ.

We have shown that one can improve the accuracy of the elliptic equations by using a Dirac delta function with small support and more moment conditions as well as by using some extrapolation techniques associated with fast solvers.

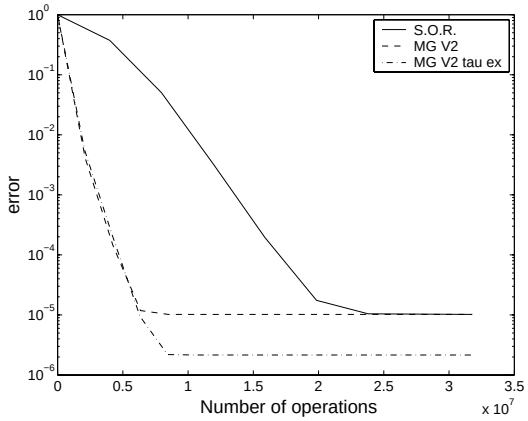


Figure 4.45: *Error along the x-axis in L_2-norm with respect to the number of operations with the 3 solvers S.O.R., Multigrid V2, Multigrid/τ-ex. $N = 200$, $r = 0.5$ and we use ϕ_4.*

4.5 A Non-centered stencil for the divergence operator around the singularity

The aim is to achieve better accuracy on test case 2 and then on the pressure equation of the IBM:

$$\Delta P = \nabla . F,$$

where $F = [F_1, F_2]^T$ contains a singularity distributed along a curve.

The idea of this technique is to use a non-centered stencil for the divergence operator around the singularity, where the error is concentrated. This implies knowing explicitly where the singularity is. In this case, we assume that the singularity is distributed along a closed curve Γ. If Ω_i and Ω_o are respectively the domain inside and outside Γ, $x_{i,j}$ the coordinates of the mesh nodes, we use a discrete mask function $M_{i,j}$ such that:

$$M_{i,j} = 1 \text{ if } x_{i,j} \in \Omega_i,$$

$$M_{i,j} = 0 \text{ if } x_{i,j} \in \Omega_o.$$

This modified divergence operator $\tilde{\nabla}$ is applied to the vector field $Q = [Q_1, Q_2]^T$, solution of:

$$\Delta Q_1 = F_1,$$

$$\Delta Q_2 = F_2.$$

with appropriate boundary conditions, in order to get P:

$$P = \tilde{\nabla}.Q.$$

This allows us to improve the solution around Γ. This method can be used on staggered or collocated meshes.

4.5.1 The Discrete non-centered divergence operator

Away from Γ, the discrete divergence operator stencil is the traditional second order stencil. Now, we have to define what should be the width of the neighborhood around Γ in which we apply the non-centered stencil. If we call N_Γ this neighborhood, this means that we never use the points of the mesh that belongs to N_Γ. The further away the points we use are from the singularity, the larger the error of the extrapolation is. Since Γ can have a large curvature, we cannot use points that are too far away from the singularity or we might get close to another part of Γ. We decided to change the stencil only within the distance of nh from the singularity, where h is the space step. This approach has a drawback: if we decrease h, the width of N_Γ decreases too and the error introduced by the singularity might not be contained by N_Γ anymore. Recall that this error decreases exponentially with respect to the distance from Γ. This means that it is necessary to increase n, which means changing the non-centered divergence stencil, when h decreases significantly. Let us describe the case where $n = 2$; the distance is then $2h$, on a collocated mesh.

The divergence is computed in a double loop, spanning all the points $x_{i,j}$ of the mesh: $1 \leq i, j \leq N+1$. For each mesh point, we assign, at first, a value based on the classic divergence stencil:

$$[\nabla.Q]_{i,j} = \frac{1}{2h}\left(Q_{1,i+1,j} - Q_{1,i-1,j}\right) + \frac{1}{2h}\left(Q_{2,i,j+1} - Q_{2,i,j-1}\right)$$

Then, we use the non-centered for the points inside N_Γ. In order to use the points that are far away from Γ, we need to know where Γ is, using the mask function $M_{i,j}$.

Now we present briefly the extrapolation process that we used, although this needs to be studied more. In the implemented technique, we have eight possible cases for the orientation of the divergence stencil in 2D. We would have fourteen cases in 3D. In 2D, the eight possible locations for Γ with respect to the mesh point $x_{i,j}$ are: North, South, East, West, North-East, North-West, South-East, South-West. The criteria on the mask function, to know the location of Γ, are described in Table 4.3.

N	$M_{i,j+2} \neq M_{i,j}$	$M_{i+2,j-1} = M_{i,j}$	$M_{i-2,j-1} = M_{i,j}$
S	$M_{i,j-2} \neq M_{i,j}$	$M_{i+2,j+1} = M_{i,j}$	$M_{i-2,j+1} = M_{i,j}$
E	$M_{i+2,j} \neq M_{i,j}$	$M_{i-1,j+2} = M_{i,j}$	$M_{i-1,j-2} = M_{i,j}$
W	$M_{i-2,j} \neq M_{i,j}$	$M_{i+1,j+2} = M_{i,j}$	$M_{i+1,j-2} = M_{i,j}$
N-E	$M_{i+2,j+2} \neq M_{i,j}$	$M_{i-2,j+1} = M_{i,j}$	$M_{i+1,j-2} = M_{i,j}$
N-W	$M_{i-2,j+2} \neq M_{i,j}$	$M_{i-1,j-2} = M_{i,j}$	$M_{i+2,j+1} = M_{i,j}$
S-E	$M_{i+2,j-2} \neq M_{i,j}$	$M_{i-2,j-1} = M_{i,j}$	$M_{i+1,j+2} = M_{i,j}$
S-W	$M_{i-2,j-2} \neq M_{i,j}$	$M_{i-1,j+2} = M_{i,j}$	$M_{i+2,j-1} = M_{i,j}$

Table 4.3: *Check conditions on the mask function $M_{i,j}$ to find the location of Γ with respect to the mesh node $x_{i,j}$.*

In Figure 4.46, the case where Γ is located at the East of $x_{i,j}$ is represented, while in Figure 4.47, the case where Γ is located at the North-East is shown.

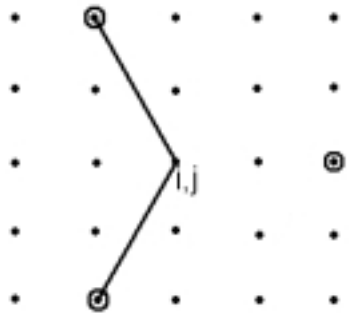


Figure 4.46: *Stencil for the East location of* Γ*'s check.*

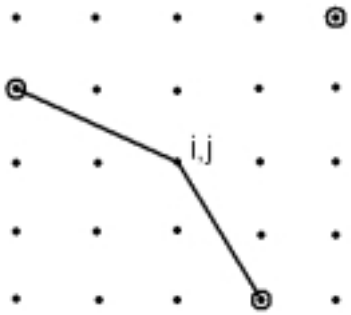


Figure 4.47: *Stencil for the North-East location of* Γ*'s check.*

It is important to start by checking the four first directions: N,S,E,W, before the four other ones: N-E,N-W,S-E,S-W. This method is easy to code, fast and allows us to get a good discrete representation of N_Γ. In Figure 4.48, we can see an example of how N_Γ has been detected: we see the mesh, the contour of N_Γ and the contour of Γ in the middle of N_Γ. We can see that the points inside N_Γ are within a distance of $2h$ from Γ.

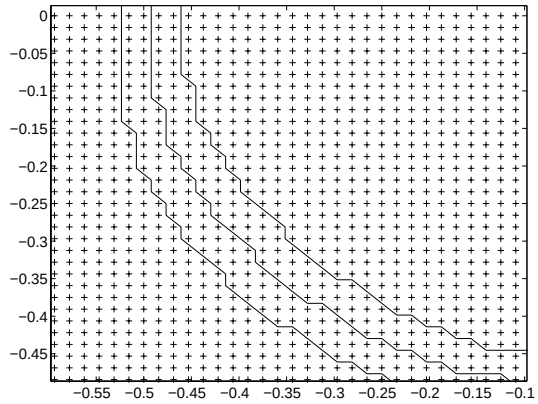


Figure 4.48: *Graph of the mesh, contours of* N_Γ *and the mask* $M_{i,j}$ *(in the middle).*

For each of the points inside N_Γ, we compute the divergence using a non-centered stencil. We use a simple extrapolation technique based on Taylor expansions using the values of the vector field outside of N_Γ. The points used are on the West if Γ

is on the East of $x_{i,j}$, on the South-West, if Γ is on the North-East and so on. We use basic basic third order non-centered approximations of the divergence, from the Taylor expansions.

Let us apply the technique on test case 2.

4.5.2 Test case 2

At first we solve the problems:

$$\begin{aligned} \Delta Q_1 &= F_1 \quad \text{in} \quad \Omega = [-1,1]^2, \\ \Delta Q_2 &= F_2 \quad \text{in} \quad \Omega = [-1,1]^2, \end{aligned} \tag{4.54}$$

with:

$$F_i(x_1,x_2) = \int_\Gamma 2x_i(s)\delta(x_1,x_2,\Gamma)ds, \quad i=1,2.$$

$$\Gamma = \left\{ (x_1,x_2)/x_1^2 + x_2^2 = \frac{1}{4} \right\}.$$

Since we want to have homogeneous Dirichlet boundary conditions on the divergence of $Q = [Q_1, Q_2]^T$, we need to take these boundary conditions on Q_1 and Q_2:

$$\begin{aligned} \frac{\partial Q_1}{\partial x_1}(-1,x_2) = \frac{\partial Q_1}{\partial x_1}(+1,x_2) = 0, \\ Q_1(x_1,-1) = Q_1(x_1,+1) = 0, \\ \frac{\partial Q_2}{\partial x_2}(x_1,-1) = \frac{\partial Q_2}{\partial x_2}(x_1,+1) = 0, \\ Q_2(-1,x_2) = Q_2(+1,x_2) = 0. \end{aligned}$$

We can see in Figures 4.49 and 4.50 the results Q_1 and Q_2 of these two related problems. We use ϕ_4.

Now, we just have to compute the divergence of Q using the modified divergence operator that we just described: $P = \tilde{\nabla}.Q$. We can see in Figure 4.51 the improvement of this extrapolation technique. While the jump of the solution computed with the

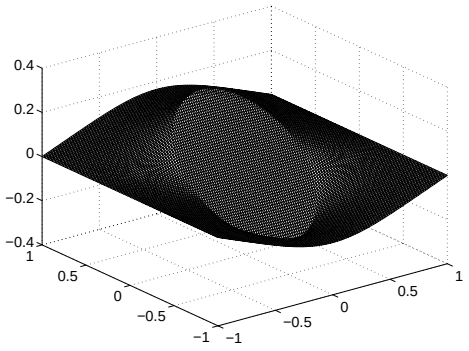


Figure 4.49: Q_1.

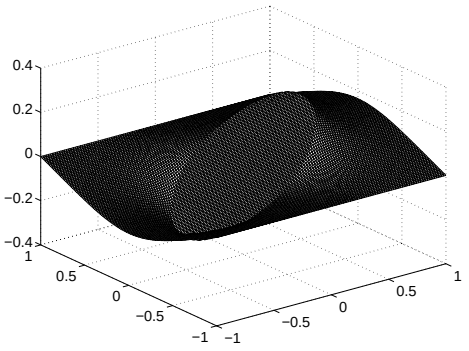


Figure 4.50: Q_2.

regular divergence operator applied to the right-hand side at first, is smeared around Γ, the solution computed with the *a posteriori* modified divergence operator has a jump close to the theoretical one.

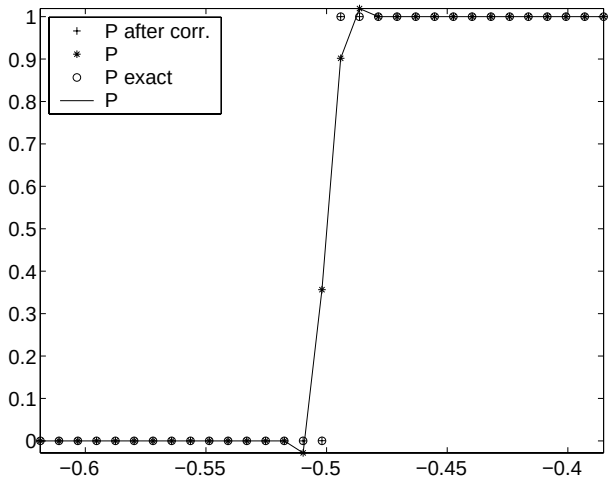


Figure 4.51: *Solution P around the jump, with (+) or without (*) the correction, compared to the exact solution (o).*

Let us present the results on the convergence properties of the method for test case 2 in Figures 4.52, 4.53, 4.54 and 4.55.

The order changes when h becomes small. As we explained above, this is due to the fact that the width of N_Γ is proportional to h. When h decreases significantly, the

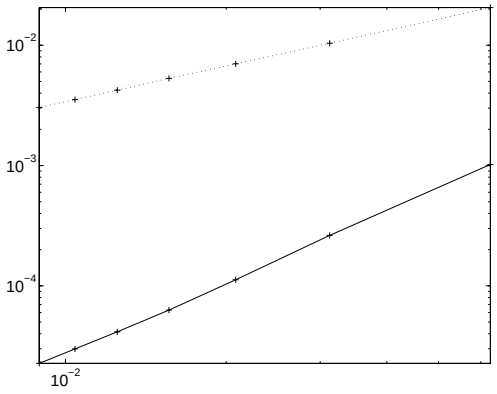


Figure 4.52: $\|.\|^h_{L_1}$*-norm of the error. solid line: non-centered divergence stencil. dotted line: basic divergence stencil.*

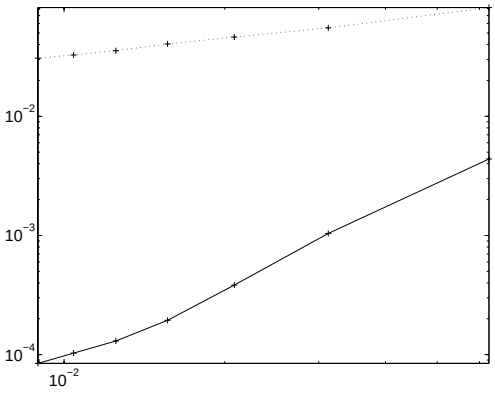


Figure 4.53: $\|.\|^h_{L_2}$*-norm of the error. solid line: non-centered divergence stencil. dotted line: basic divergence stencil.*

error along Γ is no longer contained in N_Γ and thus, propagates to the non-centered divergence stencil. We measure the order when h is not too small and get a uniform second-order accuracy, as we can see in Table 4.4.

norm	$\|.\|^h_{L_1}$	$\|.\|^h_{L_2}$	$\|.\|^h_\infty$	$\|.\|^h_{L_2}$
without corr.	0.97	0.45	0.01	0.51
with corr.	2.06	2.42	2.15	2.59

Table 4.4: *Convergence order for the solution of test case 2 using ϕ_4 and the non-centered divergence operator correction technique.*

4.5.3 The pressure equation

We apply the same method to the pressure equation of the IBM, using a typical right-end side from the "bubble" test case corresponding to a stiff force term interpolated with ϕ_4. The aim is to solve the pressure equation, with no error introduced near the jump by the discrete Dirac delta function , thanks to the non-centered stencil.

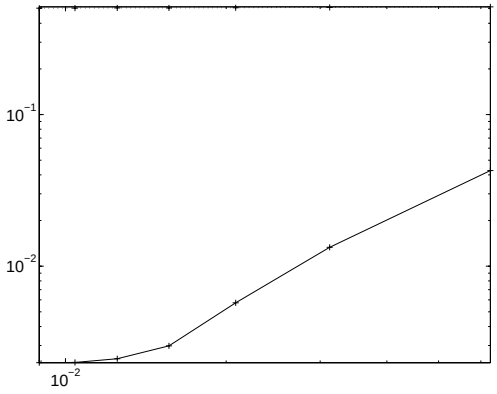


Figure 4.54: $\|.\|_{\infty}^{h}$*-norm of the error. solid line: non-centered divergence stencil. dotted line: basic divergence stencil.*

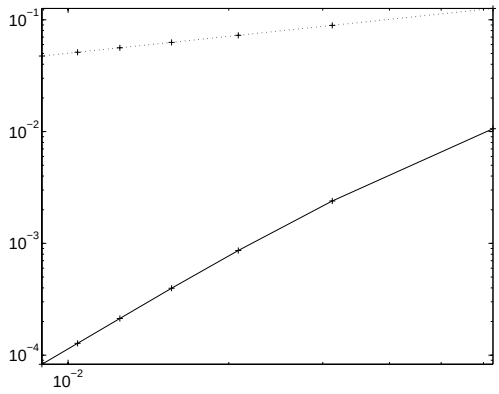


Figure 4.55: $|.|_{L_2}^{h}$*-norm of the error along the x-axis. solid line: non-centered divergence stencil. dotted line: basic divergence stencil.*

L. Lee and R. J. Leveque wrote in [67]: "The entire force is spread to the grid, giving discrete delta function behavior in the intermediate velocity V^*. Applying the discrete divergence to this leads to discrete dipole behavior in $\nabla.V^*$, and using this as the right hand side in the Poisson problem leads to a smeared jump in the numerical approximation to the pressure.[...] We believe that this discrete dipole in V^* and its use in the Poisson problem are the principal causes of inaccuracy in the immersed boundary method [...]."

Since we want the pressure to have homogeneous Neumann boundary conditions, we need to solve:

$$\Delta Q_1 = F_1 \quad \text{in } \Omega = [0,1]^2,$$

$$\Delta Q_2 = F_2 \quad \text{in } \Omega = [0,1]^2,$$

with these boundary conditions:

$$\frac{\partial^2 Q_1}{\partial x_1^2}(0,x_2) = \frac{\partial^2 Q_1}{\partial x_1^2}(1,x_2) = 0,$$

$$\frac{\partial Q_1}{\partial x_2}(x_1,0) = \frac{\partial Q_1}{\partial x_2}(x_1,1) = 0,$$

$$\frac{\partial^2 Q_2}{\partial x_2^2}(x_1, 0) = \frac{\partial^2 Q_2}{\partial x_2^2}(x_1, 1) = 0,$$
$$\frac{\partial Q_2}{\partial x_2}(0, x_2) = \frac{\partial Q_2}{\partial x_2}(1, x_2) = 0.$$

We can see an example of Q_1, Q_2, P evaluated with or without the non-centered divergence stencil respectively in Figures 4.56, 4.57, 4.58 and 4.59.

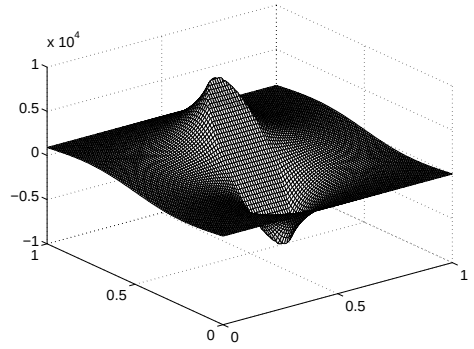


Figure 4.56: Q_1.

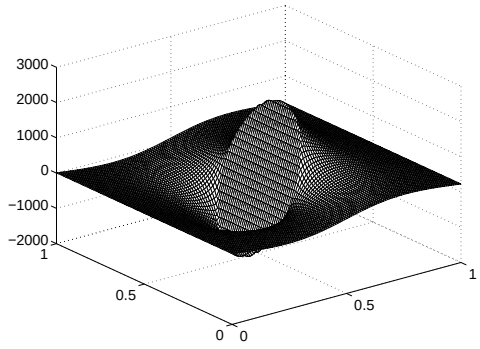


Figure 4.57: Q_2.

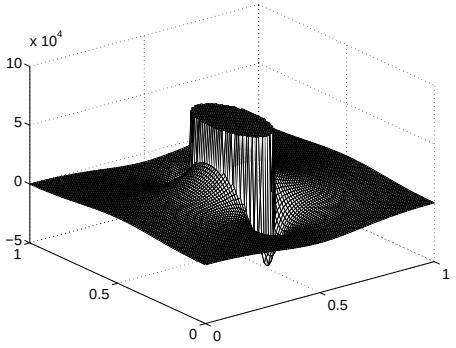


Figure 4.58: *Pressure field in the "bubble" test case using the non-centered divergence stencil and* ϕ_4.

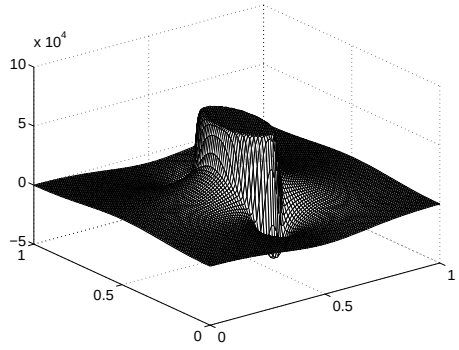


Figure 4.59: *Pressure field in the "bubble" test case using the classical method and* ϕ_2.

In this chapter, we have shown that the piecewise cubic discrete Dirac delta function ϕ_4 [111], associated with the staggered mesh, gives good results in the IBM regarding

accuracy. We first studied its behavior solving elliptic equations with singular source terms on collocated grids [34] and then extended the study to the IBM on the staggered mesh. The other advantage of this function ϕ_4 is that it is quickly computed, compared to the others, which means that we gain some computational time during the boundary treatment of the IBM. Similar observations have been made recently by B.E. Griffith [44]. The drawback is that it is sharper and thus introduces more oscillations in the method, while it makes the NS system stiffer. Thus, the time step needs to be reduced if no other technique is implemented to stabilize the scheme. The other drawback is that it makes the immersed boundary more porous numerically than the other discrete Dirac delta functions. However, we show that is it easy to control the volume thanks to constrained optimization and Fourier representations of the interface.

At the same time, we implemented a fast, efficient solver for the pressure equation of the IBM, using the multigrid/τ-extrapolation algorithm associated with an actual double discretization technique.

Finally, we introduced the idea of a non-centered stencil for the divergence operator of the pressure equation, in the neighborhood of the interface. This fast technique makes use of the knowledge of the interface position and gives a second order convergence property for the pressure accuracy. However, it requires the implementation of an extrapolation technique, which can be complicated in 3D and for complex geometries. Another issue is that this technique is hard to associate with the divergence-free condition of incompressible fluids.

Chapter 5

Stability

The instability of the IBM is caused by the stiffness of the NS system and the position update of the membrane. In the momentum equation, we get two singular terms that are ∇P and the source term F. This forms a collection of singularities to describe the force that the membrane exerts on the fluid. The difference between these two terms is essentially responsible for the discrete motion of the membrane. Since there is no dissipative term in the elastic law implemented, the grid points describing the membrane have a strong tendency to oscillate, which leads to even stiffer differences between ∇P and F. This is the main source of instability. If one introduces some relaxation by giving mass to the membrane, we get a more stable scheme. Since this mass is neglected in most cases, we study the stability of the basic IBM and look for more efficient solvers that can overcome this difficulty:

- Solution 1: to implicit the method.
- Solution 2: to use some filtering techniques on the moving boundary position vector.

5.1 Motivation: instability of the regular IBM

Let us look at the maximum possible time step for different problem sizes in the "bubble" test case, with parameters chosen in such a way that it is not limited by the CFL condition and using the two different discrete Dirac delta functions: cosine ϕ_2 and piecewise cubic ϕ_4. In this particular case, we have $\sigma = 10000$, $M = 6N$, $\mu = 1$, $\rho = 1$. Recall that σ is the elasticity coefficient of the immersed boundary, M the number of discrete points along the immersed boundary, N the number of discrete points in each direction of the Cartesian mesh, μ the viscosity coefficient of the fluid, and ρ the density. The time steps are far below the ones imposed by the CFL condition, whichever delta function is used.

N	ϕ_2	ϕ_4
32	$2.0\ 10^{-4}$	$5\ 10^{-5}$
64	$6\ 10^{-5}$	$1\ 10^{-5}$

Table 5.1: *Maximum time step for the explicit IBM, using ϕ_2 or ϕ_4.*

There is a factor of 4, between the two time steps. If using ϕ_4 improves the accuracy, it does alter the stability. The main reason for this instability is the sharp shape of ϕ_4: the more we regularize the delta function using a wider support, the more stable the method becomes because of the direct relation it has with the stiffness of the force term. Another reason may be the oscillations introduced by the fact that ϕ_4 does not satisfy the condition (vi) of the compatibility conditions introduced by C. Peskin:

$$(vi) \quad \sum_{i\in\mathbb{Z}} \left[\phi(r-i)\right]^2 = C_2 \leq 1, \ \ \forall r \in \mathbb{R},$$

which deals with the stability of the immersed boundary movement. Finally there is a small but discontinuous over-jump next to the interface when using ϕ_4, probably due to the piecewise continuity of the function.

But let us focus on the global instability issue of the IBM due to the stiffness of the system, regardless of which delta function is used. A theoretical analysis of this instability has been performed in the linear case by J.M. Stockie [105]. Figures 5.1, 5.2 and 5.3 show the behavior of the immersed boundary in the "bubble" test case, when the scheme blows up.

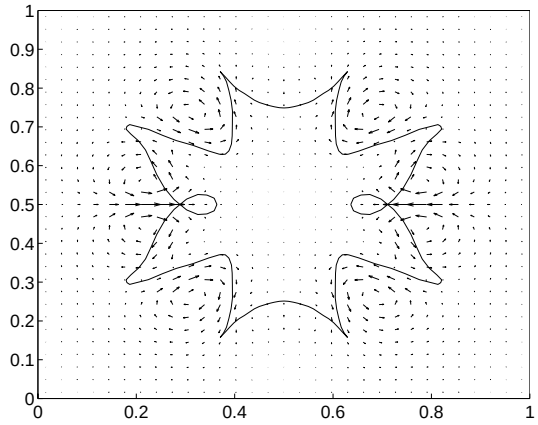


Figure 5.1: *Example of the behavior of the immersed "bubble" when the scheme blows up.*

The first idea in order to improve the stability of the method is to use a fully implicit time stepping. Let us introduce, at first, the explicit second-order scheme for the IBM.

5.2 A Semi-implicit scheme for the IBM

There exist many second-order NS projection schemes. A recent description of these methods is found in [21], for example. While the relatively new second-order schemes are fully second-order accurate, we can observe that the older ones are second-order accurate only for the velocity vector field. The pressure field is commonly first-order accurate in the L_∞-norm. Let us present the simplest second-order projection scheme,

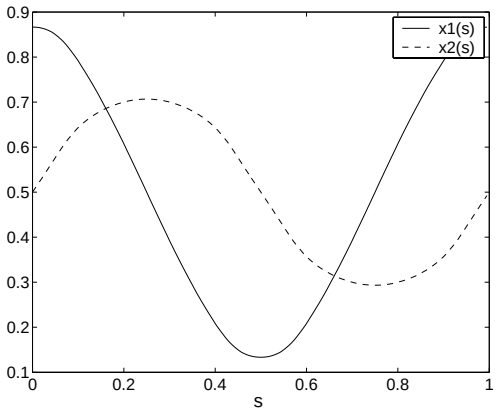


Figure 5.2: *Curvilinear position vectors of the moving boundary in the "bubble" test-case: stable case*

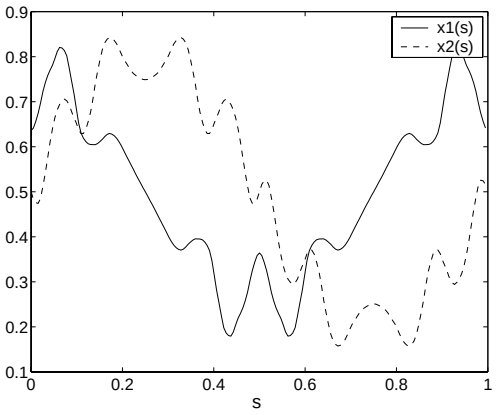


Figure 5.3: *Curvilinear position vectors of the moving boundary in the "bubble" test-case: unstable case*

based on the midpoint and trapezoidal rules. In the midpoint rule, we discretize $\frac{\partial V}{\partial t}(x,t) = g(x,t)$ by:

$$\frac{V^{n+1} - V^n}{\Delta t} = g(V^{n+\frac{1}{2}}, t^{n+\frac{1}{2}}).$$

In the trapezoidal rule, the discretization is as follows:

$$\frac{V^{n+1} - V^n}{\Delta t} = \frac{1}{2}\left[g(V^n, t^n) + g(V^{n+1}, t^{n+1})\right].$$

In the NS solver, at each time step, we need to evaluate at first $V^{n+\frac{1}{2}}$, $P^{n+\frac{1}{2}}$ and $X^{n+\frac{1}{2}}$ in a preliminary step:

$$F^n = \int_\Gamma f^n(s)\delta(x - X^n(s))ds, \tag{5.1}$$

$$\rho\left[\frac{V^* - V^n}{\Delta t/2} + (V^n.\nabla)V^n\right] = \mu\Delta V^n - \nabla P^n + F^n, \tag{5.2}$$

$$\Delta\delta P^n = \frac{2\rho}{\Delta t}\nabla.V^*, \tag{5.3}$$

$$\rho\left[\frac{V^{n+\frac{1}{2}}-V^{*}}{\Delta t/2}\right]=-\nabla\delta P^{n}, \tag{5.4}$$

$$P^{n+\frac{1}{2}}=P^{n}+\delta P^{n}, \tag{5.5}$$

$$\frac{X^{n+\frac{1}{2}}-X^{n}}{\Delta t/2}=\int_{\Omega}V^{n+\frac{1}{2}}\delta(x-X^{n})dx. \tag{5.6}$$

Then we can compute V^{n+1}, P^{n+1} and X^{n+1}:

$$F^{n+\frac{1}{2}}=\int_{\Gamma}f^{n}(s)\delta(x-X^{n+\frac{1}{2}}(s))ds, \tag{5.7}$$

$$\rho\left[\frac{V^{**}-V^{n}}{\Delta t}+(V^{n+\frac{1}{2}}.\nabla)V^{n+\frac{1}{2}}\right]=\mu\Delta V^{n+\frac{1}{2}}-\nabla P^{n+\frac{1}{2}}+F^{n+\frac{1}{2}}, \tag{5.8}$$

$$\Delta\delta P^{n+\frac{1}{2}}=\frac{\rho}{\Delta t}\nabla.V^{**}, \tag{5.9}$$

$$\rho\left[\frac{V^{n+1}-V^{**}}{\Delta t}\right]=-\nabla\delta P^{n+\frac{1}{2}}, \tag{5.10}$$

$$P^{n+1}=P^{n+\frac{1}{2}}+\delta P^{n+\frac{1}{2}}, \tag{5.11}$$

$$\frac{X^{n+1}-X^{n}}{\Delta t}=\int_{\Omega}\left[\frac{V^{n}+V^{n+1}}{2}\right]\delta(x-X^{n+\frac{1}{2}})dx. \tag{5.12}$$

This represents twice the computational cost of the explicit scheme, since we have two systems to solve per time step, instead of one. It is possible to have an implicit diffusion term in the prediction steps of the NS solver, Eqs.(5.2) and (5.8). The convective term can be partially implicit too. We get then for Eq. (5.8):

$$\rho\left[\frac{V^{**}-V^{n}}{\Delta t}+(V^{n+\frac{1}{2}}.\nabla)V^{**}\right]=\mu\Delta V^{**}-\nabla P^{n+\frac{1}{2}}+F^{n+\frac{1}{2}}. \tag{5.13}$$

However, it implies solving an extra system per time step. It is also possible to apply the trapezoidal rule to the diffusion term:

$$\frac{1}{2}\Delta\left[V^n + V^{**}\right] \quad \text{or} \quad \frac{1}{2}\Delta\left[V^n + V^{n+1}\right],$$

with appropriate boundary conditions on the pressure equation, and an explicit multi-step scheme for the convective term, such as:

$$\left(\frac{3}{2}V^n - \frac{1}{2}V^{n-1}\right).\nabla\left(\frac{V^{**} + V^n}{2}\right). \tag{5.14}$$

Now, we extend this scheme to the implicit second-order IBM.

5.3 A Fully implicit scheme for the IBM

This scheme is similar to the semi-implicit scheme just described.

$$\rho\left[\frac{V^* - V^n}{\Delta t/2} + (V^n.\nabla)V^n\right] = \mu\Delta V^* - \nabla P^n + F^n, \tag{5.15}$$

$$\Delta(\delta P)^n = \frac{2\rho}{\Delta t}\nabla V^*, \tag{5.16}$$

$$P^{n+\frac{1}{2}} = P^n + \delta P^n; \tag{5.17}$$

$$\rho\left[\frac{V^{n+\frac{1}{2}} - V^*}{\Delta t/2}\right] = -\nabla(\delta P^n), \tag{5.18}$$

Second step (implicit):

$$\rho\left[\frac{V^{**} - V^n}{\Delta t} + (V^{n+\frac{1}{2}}.\nabla)V^{n+\frac{1}{2}}\right] = \mu\Delta V^{n+\frac{1}{2}} - \nabla P^{n+\frac{1}{2}} + \frac{1}{2}\left(F^n + F^{n+1}\right), \tag{5.19}$$

$$\Delta\left(\delta P^{n+\frac{1}{2}}\right) = \frac{\rho}{\Delta t}\nabla V^{**}, \tag{5.20}$$

$$P^{n+1} = P^{n+\frac{1}{2}} + \delta P^{n+\frac{1}{2}}, \tag{5.21}$$

$$\rho \left[\frac{V^{n+1} - V^{**}}{\Delta t} \right] = -\nabla(\delta P^{n+\frac{1}{2}}), \tag{5.22}$$

$$\frac{X^{n+1} - X^n}{\Delta t} = \frac{1}{2} \left(\int_\Omega V^n \delta(x - X^n) dS + \int_\Omega V^{n+1} \delta(x - X^{n+1}) dS \right). \tag{5.23}$$

It is possible to apply the trapezoidal rule to the diffusive term. The Inexact Newton Backtracking method [85] is used at each time step to solve this last equation Eq.(5.23), where V^{n+1} is depending on X^{n+1} and F^{n+1} too. Each objective function evaluation in the Newton algorithm requires a NS computation with one linear system to solve (for the pressure correction), in order to get $V^{n+1}(X)$. So at each time step, we have to solve : $g(X) = \mathbf{0}$, with $g : \mathbb{R}^{PM} \mapsto \mathbb{R}^{PM}$ in the P-dimensional case (M is the number of discretization points along the immersed boundary).

$$g(X) = X - X^n - \frac{\Delta t}{2} \left(\int_\Omega V^n \delta(x - X^n) dS + \int_\Omega V^{n+1} \delta(x - X^{n+1}) dS \right). \tag{5.24}$$

5.3.1 The Search space

The search space at each time step for our problem is $\mathbb{R}^{PM}$ for a P-dimensional case:

$$X = \begin{bmatrix} X_1 \\ : \\ X_P \end{bmatrix},$$

where $X_i, \;\; 1 \leq i \leq P$ is the vector of coordinate of the immersed boundary in the ith direction. We actually solve this nonlinear problem : $F(x) = 0$ with

$$F(x) = F\left(\begin{bmatrix} X_1 \\ : \\ X_P \end{bmatrix}\right) = \begin{pmatrix} g_1([X_1, .., X_P]) \\ : \\ g_P([X_1, .., X_P]) \end{pmatrix},$$

assuming that F is continuously differentiable everywhere in $\mathbb{R}^{PM}$.

5.3.2 Newton's method

At each Newton step k, we have to solve the linear equation:

$$J(x_k)s_k = -F(x_k). \tag{5.25}$$

The first problem is that we do not have J, the Jacobian matrix of F: we can only approximate the matrix-vector product $J(x_k)s_k$, which is computationaly expensive and not very accurate. In our case, we use a forward finite difference approximation. The second difficulty is that we have a large scale problem. The method we implemented is the Inexact Newton Backtracking (INB) method [85], which offers strong global convergence properties combined with potentially fast local convergence: at each time step, we compute an approximate solution of (5.25) with an iterative solver (Krylov method), under the inexact Newton condition:

$$\|F(x_k) + J(x_k)s_k\| \leq \eta_k \|F(x_k)\| \tag{5.26}$$

At each time step, a correction on the length of s_k is eventually done using an iterative linear backtracking method.

5.3.3 The Inexact Newton backtracking method: basic Algorithm

Let x_0, $\eta_{max} \in [0, 1)$, $t \in (0, 1)$, and $0 < \theta_{min} < \theta_{max} < 1$ be given:

for $k = 0$ step 1 until convergence do

choose $\overline{\eta}_k \in [0, \eta_{max}]$ and determine $\overline{s}_k$ such that

$\|F(x_k) + J(x_k)\overline{s}_k\| \leq \overline{\eta}_k \|F(x_k)\|$

set $s_k = \overline{s}_k$ and $\eta_k = \overline{\eta}_k$

while $\|F(x_k + s_k)\| > (1 - t(1 - \eta_k)) \|F(x_k)\|$ do

choose $\theta \in [\theta_{min}, \theta_{max}]$

update $s_k \leftarrow \theta s_k$ and $\eta_k \leftarrow 1 - \theta(1 - \eta_k)$

set $x_{k+1} = x_k + s_k$

end

5.3.4 The Inexact Newton condition step

We solve $J(x_k)s_k = -F(x_k)$ using an iterative method with a tolerance equal to $\eta_k \|F(x_k)\|_2$ (if $k = 1$ the initial guess for s_k is the zero vector, if $k \geq 1$ it is s_{k-1}). For example, we use the restarted GMRES method of Saad and Schultz [99]. If using FORTRAN, it is be possible to achieve parallelism with this part of the code: the parallel SNES (Scalable Nonlinear Equation Solver) package, using MPI, offers a Newton solver with backtracking and various Krylov methods. It is not possible to use a preconditioner for these Krylov methods or other solvers: when we solve the system $Ax = b$, we do not have the square matrix A but only the matrix-vector product Ax.

5.3.5 The Jacobian matrix

During the inexact Newton step we need to evaluate the Jacobian of F. Actually the Krylov methods only require the evaluation of the product $J(x_k)s_k$. We use a finite difference scheme of order one to approximate it :

$$J(x_k)s_k \simeq \frac{F(x_k + \delta s_k) - F(x_k)}{\delta}$$

δ has to be as small as possible, in order to give a good approximation of the derivative, since F is nonlinear. But we have to consider the floating point and truncation errors: if δ is too small, then $\|F(x_k + \delta s_k) - F(x_k)\| = 0$.

A value for δ can be, for example:

$$\delta = 100\epsilon_{computer}\frac{\|x_k\|_\infty}{\|s_k\|_\infty}$$

5.3.6 Stopping criteria for the INB method

We say that convergence is reached when:

$$\|F(x_k)\|_2 \leq \epsilon_{newt}\|F(x_0)\|$$

or

$$\|s_k\| \leq \epsilon_{newt}$$

with $\epsilon_{newt} = 10^{-6}$ for example.

We also set some criteria for the failure of convergence:

- More than 20 INB method iterations without reaching convergence.
- s_k not found after 200 iterations of the iterative method.
- More than 15 iterations of the backtracking loop without reaching the critera:

$$\|F(x_k + s_k)\| > (1 - t(1 - \eta_k))\,\|F(x_k)\|.$$

5.3.7 The Forcing term

The parameter η_k is very important in the INB method: it is forcing $F(x_k) + F'(x_k)s_k$ to be small in a particular way, in order to control the speed of convergence and the accuracy of the results. The values of the η_ks have been optimized by S. C. Eisenstat and H. F. Walker in [28]. We chose to use these values:

$$\eta_0 = 0.5$$
$$\eta_{k+1} = \left| \frac{\|F(x_{k+1})\|_2 - \|F(x_k)+J(x_k)s_k\|_2}{\|F(x_k)\|_2} \right|, \qquad \text{for k=1,2,...}$$

with these safeguards at the end of each INB step to prevent the η_k to become too small too fast, far away from the solution:

$$\eta_{k+1} = \begin{cases} max\{\eta_{k+1}, \eta_k^{\frac{(1+\sqrt{5})}{2}}\}, & \text{if} \quad \eta_k^{\frac{(1+\sqrt{5})}{2}} > 0.1 \\ \\ min\{\eta_{k+1}, \eta_{max}\}, & \text{with } \eta_{max} = 0.9 \text{ for example.} \end{cases}$$

5.3.8 Choice of θ

At each backtracking step θ is chosen in order to minimize the quadratic $p(\theta)$ over $[\theta_{min}, \theta_{max}]$, where p is such that:

$$\begin{cases} p(0) = g(0) \\ p'(0) = g'(0) \\ p(1) = g(1) \end{cases}$$

with $g(\theta) = \|F(x_k + \theta s_k)\|_2^2$.

The problem of this implicit scheme is the very large computational cost, due to the large number of NS evaluations per time step: around twenty in the "bubble" test case. However, we can observe an interesting stability result in Figure 5.5. Before giving the results of this scheme in terms of stability, let us introduce a variation of it based on a cosine expansion to represent the immersed boundary, in order to reduce the size of the search space.

5.3.9 On The use of a compact representation of the interface

In all the schemes above, we discretize the moving boundary with a vector X of M components. That means that the size of the search space in the Newton algorithm,

for the implicit scheme, is $2 \times M$ in 2D. We can decrease the size of this search space using a Fourier expansion. If we call $Nmod$ the numbers of modes in this expansion, the search space is then $2Nmod + 1$. This has two good consequences : it reduces the computational cost and at the same time, it filters the position of the boundary, getting rid of the short frequencies that can be sources of perturbations. The problem could be, then, that the scheme would converge to another solution, but the solutions we find with or without the Fourier expansions are extremely close. The error, on a short time period, introduced by a moderate filtering technique is neglectable compared to the one introduced by the discretization. On a long time period, the error issue becomes a volume conservation issue.

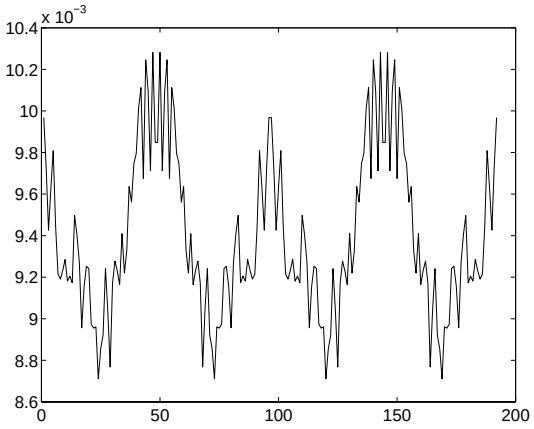


Figure 5.4: *Distance between the discretization points of the immersed boundary in an unstable case.*

We can observe in Figure 5.6 that the use of Fourier expansions really decreases the number of objective function evaluations in the inexact Newton backtracking method, but it is still expansive compared to the explicit midpoint-trapezoidal method, even if it has the advantage of being significantly more stable.

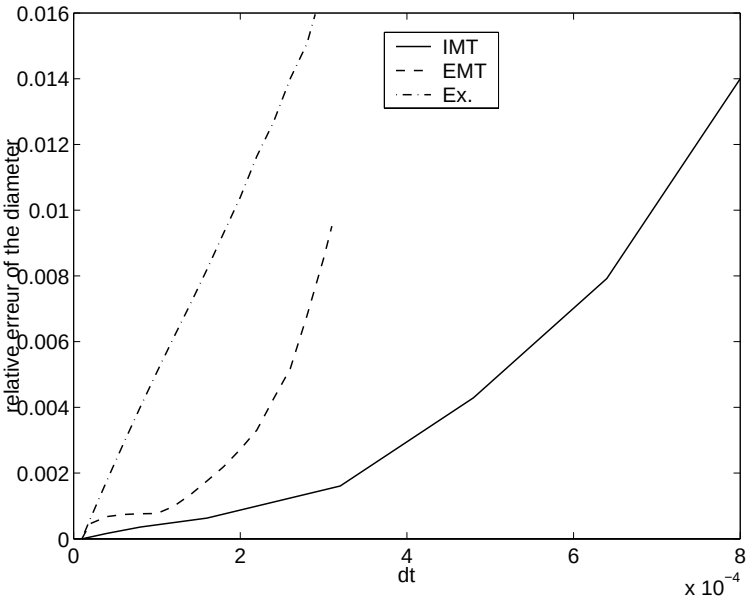


Figure 5.5: *Relative error of the diameter w.r.t. time step, "bubble" test case,* $Nx = Ny = 32$*,* $\sigma = 10000$*,* $\mu = 1$*,* $T_{final} = 0.02$*. IMT: implicit midpoint-trapezoidal scheme with or without the Fourier expansion. EMT: explicit midpoint-trapezoidal scheme. Ex: explicit scheme.*

5.4 The Fourier filtering technique

We saw that it was possible to use a Fourier expansion to represent the position of the elastic immersed boundary. We measured experimentally that the error introduced by this expansion can be neglected compared to the discretization errors (time and space). Theoretically, we need to have a smooth, equally-spaced periodic vector function in order to use Fourier expansions, which we do not have in the IBM case, due to the elasticity of the moving boundary. However, since there is a no-slip boundary condition between the immersed boundary and the velocity of the fluid, and that the velocity field is a smooth regular field, the deformation applied to the immersed boundary is smooth and regular. The deformation of the elastic material is smooth and regular too, since we use Hooke's law, which is linear. This is why we use

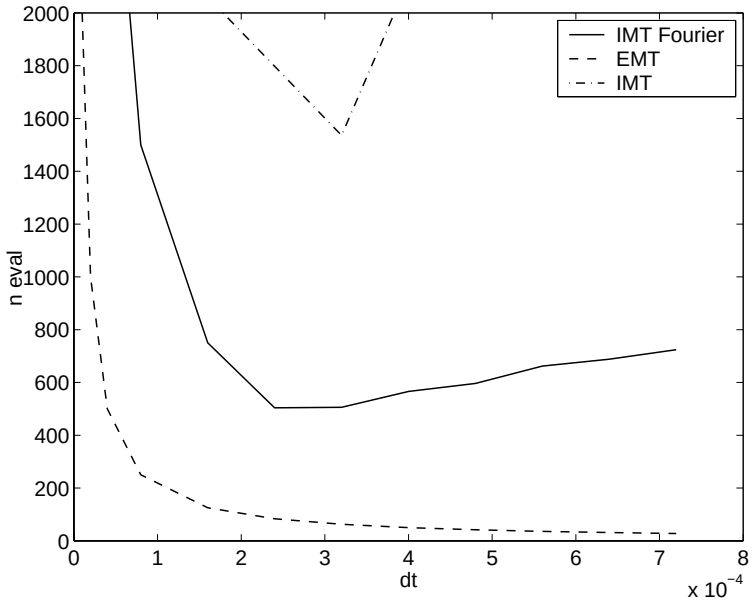


Figure 5.6: *Number of NS computations w.r.t. time step, "bubble" test case,* $Nx = Ny = 32$*,* $\sigma = 10000$*,* $\mu = 1$ *and* $T_{final} = 0.02$*. IMT Fourier: implicit midpoint-trapezoidal scheme with the Fourier expansion. EMT: explicit midpoint-trapezoidal scheme. IMT: implicit midpoint-trapezoidal scheme.*

the expansion and then some filtering techniques [43, 29] in order to control the frequencies and improve the stability of the method.

The results in the "bubble" test-case, using the traditional scheme and the cosine delta function ϕ_2, show that the filtering techniques allow us to use larger time steps by a factor three in a stiff case ($\sigma = 100000$, $\mu = 1$, $M = 3N$, $T_{final} = 5.\ 10^{-3}$), even if we try to minimize the error introduced by the filter, taking $\kappa \leq 6$ (see Table 5.2). The drawback of the filter is that it alters the conservation of volume property of the IBM (Figures 5.7 and 5.8). This could be eliminated by the global volume conservation technique based on Fourier expansions and constrained minimization introduced in chapter 4.

N	without filter	with filter	κ
32	$3\ 10^{-5}$	$6\ 10^{-5}$	5
64	$2.2\ 10^{-5}$	$6.9\ 10^{-5}$	6

Table 5.2: *Maximum time step for the filtered explicit IBM, using* ϕ_2.

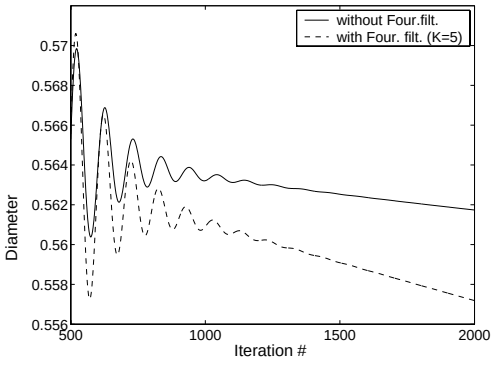


Figure 5.7: *Diameter of the bubble from the time iteration number* 500 *to* 2000*, with* $N = 32$*,* $M = 6N$*,* $\sigma = 10^4$*,* $\Delta t = 5.10^{-5}$*, with the cosine delta function and with or without the Fourier filter.*

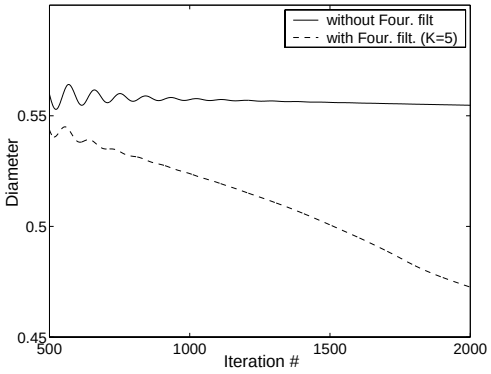


Figure 5.8: *With the piecewise cubic delta function*

To conclude this chapter, we implemented a fully implicit scheme for the IBM, based on a quasi-Newton solver and the compact Fourier representation of the immersed boundary. This gives some good results regarding the stability but still requires a lot of NS evaluations per time step. It only saves computations in very stiff cases, in which the elasticity coefficient of the membrane is very large. This method could be associated with a domain decomposition method: an explicit solver would be used in the sub-domains that do not contain a sharp interface, while this implicit solver would be used in the other sub-domains, in order to deal with the stiff local NS system. At the same time, we observed a significant improvement of the stability when using a

filtering technique on the Fourier coefficients. The drawback is regarding the volume conservation property of the method associated with the filter. This can be fixed easily by implementing the fast and global volume conservation technique based on the same Fourier coefficients used for filtering, introduced in the previous chapter.

Chapter 6

Fluid flows with parallel MATLAB

We focus on domain decomposition techniques applied to fluid flows, in order to increase problem sizes. We present our results in the context of accelerated Schwarz methods and parallel MATLAB. Our goal is to be able to compute large scale simulations of fluid-structure interactions on a computer network, taking advantage of the interactivity, the visualization ability of MATLAB and its simplicity for coding.

6.1 MATLABMPI

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization and numerical computation that is widely used by bio-engineers. MATLAB's advantages are that it is user-friendly and offers a large array of pre-defined functions (written in C). The drawback is that it is an interpreted language, not suited for iterative tasks: it must interpret every line of a loop and therefore cannot perform individual operations as fast as other languages (C,Fortran). MPI, the library specification for message-passing, is the standard for a broadly based committee of implementers. MATLABMPI [56] is a set of MATLAB scripts that implements a subset of MPI and allows any MATLAB program to be run on a parallel computer. It is designated for users who want to do

simple parallelism and codes that do not require a very small latency for the message-passing. MATLABMPI is very convenient for parallelization of an existing MATLAB code without going through the process of switching to another language, and to use a set of memory units. It allows us to benefit from MATLAB's advantages while running large scale problems. The requirements are a shared memory machine, and a MATLAB license, although it is possible to use it on distributed memory systems (then one MATLAB license per fat node is required). It works on both Windows or Linux systems and it can be used on a single processor too, to test a parallelization technique.

On a cluster, the message-passing in MATLABMPI is done via the file system. When a process sends some data to another, it writes the data to a file on a shared file system, then the other process reads that file. This means communication performance can not be faster than the file server, which updates the directory in each node. The communication within a shared memory multi-processor is faster, however, since the processors share a local directory. Still, there is a latency due to the writing time and the detection time. The actual code for the MPI_Send and MPI_Recv use file I/O: the *load* and *save* functions of MATLAB. The sender creates two files in the common directory, one *data* file and one *lock* file. The receiver has to detect the lock file and then load the data file. This technique is efficient for large messages since there is no buffering. In order to be more efficient, it is better to group the messages in larger packages, sent all at once, if possible.

The communication time has been measured (Figure 6.1). In the program used, a processor sends a vector of size N to an other processor, which sends it back to the first sender. The clock time is measured using the MATLAB function *etime*, on two different systems:

Marvin:	1 *node*	8 way AMD Opteron	2.0GHz	64 bit	32 GB RAM
Medusa:	3 *nodes*	4 way AMD Opteron	1.6GHz	64 bit	16 GB RAM

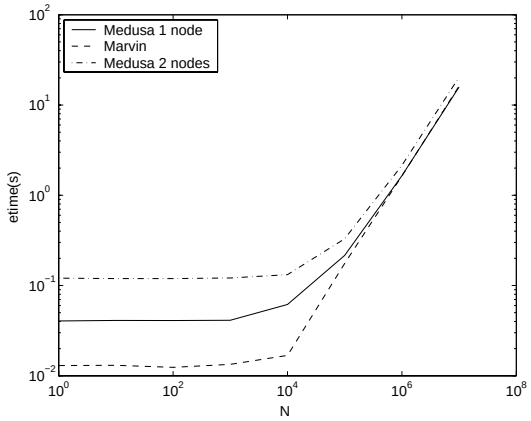


Figure 6.1: *Clock time for message passing in MATLABMPI between two processors. N is the size of the vector.*

We observe on Medusa a latency of 2.5 10^{-2} s inside a node and 6 10^{-2} s between cluster nodes. It is around 1 10^{-2} s on Marvin.

6.2 The Sequential computational cost of the IBM

We started from our MATLAB sequential implementation of the two-dimensional NS equations, with finite differences and a uniform staggered mesh. While we are using different semi-implicit or fully implicit codes, we parallelized the basic first-order projection scheme [11]. In this scheme, only the pressure correction step is computationally expensive, since a linear system has to be solved.

In the IBM, the arithmetic complexity of the boundary treatment process is proportional to the number of discrete points along the immersed boundary. We evaluated the CPU time spent on the pressure equation compared to the boundary treatment

in a sequential MATLAB 2D IBM test-case, the so called 2D "bubble" test-case. In this case, we used the *cputime* command to measure the elapsed time (Figure 6.2). The pressure solver uses a LUP decomposition: LU decomposition along with one permutation matrix. N is the size of a side of the discrete 2D square mesh.

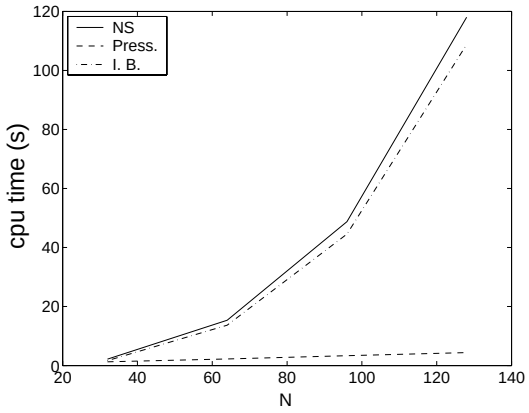


Figure 6.2: *Maltlab CPU time. NS: NS solver, Press: LUP pressure solver, IB: immersed boundary computations.*

Consequently, we will concentrate, now, on the parallelization of the pressure equation, that takes most of the CPU time.

6.3 The Parallel algorithm with homogeneous Neumann boundary conditions

The problem to solve is the following homogeneous Neumann problem, for a two-dimensional flow in a closed box:

$$\Delta P = RHS \text{ in } \Omega = [0,1]^2, \quad \frac{\partial P}{\partial \eta}|_{\partial\Omega} = 0. \tag{6.1}$$

We notice at first that the solution is defined up to a constant shift and that the right-hand side needs to satisfy a compatibility condition. We use the analytic addi-

tive Aitken-Schwarz algorithm [71], which is an excellent candidate to allow efficient, distributed computing with slow networks. The scalability of this algorithm will allow us to take advantage of a parallel machine. The rectangular uniform mesh is decomposed into a unidirectional partition of overlapping strip domains. The method is a post-process of the standard Schwarz method with an Aitken-like acceleration of the sequences of interfaces produced with the block-wise Schwarz relaxation. For simplicity, we restrict ourselves in this brief description of the algorithm to a decomposition of Ω into two overlapping subdomains: $\Omega = \Omega_1 \cup \Omega_2$ where $\Omega_1 = [0, x_r] \times [0, 1]$ and $\Omega_2 = [x_l, 1] \times [0, 1]$, $x_l < x_r$. The additive Schwarz algorithm is:

$$\Delta p_1^{n+1} = RHS \ \text{ in } \ \Omega_1, \ \ \Delta p_2^{n+1} = RHS \ \text{ in } \ \Omega_2, \tag{6.2}$$

$$p_{1|\Gamma_1}^{n+1} = p_{2|\Gamma_1}^{n}, \ \ p_{2|\Gamma_2}^{n+1} = p_{1|\Gamma_2}^{n}.$$

If we use a cosine expansion to describe the solution on the interface,

$$p(y)_{|\Gamma_i} = \sum_k \hat{p}_{|\Gamma_i}^k cos(k\pi y) \ \ \forall k, \ \ i = 1 \ \text{ or } \ 2, \tag{6.3}$$

we observe that the cosine functions expansions of the solution on the interfaces Γ_1, Γ_2 provide a diagonalization of the trace transfer operator:

$$\left(p_{1|\Gamma_1}^n, p_{2|\Gamma_2}^n\right) \xrightarrow{T} \left(p_{1|\Gamma_1}^{n+1}, p_{2|\Gamma_2}^{n+1}\right). \tag{6.4}$$

As a matter of fact, $\Delta P = RHS$ decomposes onto a set of independent ODE problems:

$$\frac{\partial^2 \hat{p}_k(x)}{\partial x^2} - \mu_k \hat{p}_k(x) = \widehat{RHS}_k, \ \ \forall k. \tag{6.5}$$

Let us denote T_k the trace operator for each wave component of the interface:

$$\left(\hat{p}_{1|\Gamma_1}^{n,k} - \hat{P}_{\Gamma_1}^k, \hat{p}_{2|\Gamma_2}^{n,k} - \hat{P}_{\Gamma_2}^k\right) \xrightarrow{T_k} \left(\hat{p}_{1|\Gamma_1}^{n+1,k} - \hat{P}_{\Gamma_1}^k, \hat{p}_{2|\Gamma_2}^{n+1,k} - \hat{P}_{\Gamma_2}^k\right), \ \ \forall k. \tag{6.6}$$

The operators T_k are linear and the sequences $\left\{\hat{p}^n_{1|\Gamma_1}\right\}$ and $\left\{\hat{p}^n_{2|\Gamma_2}\right\}$ have linear convergence. If we call $\hat{\delta}^1_k$ and $\hat{\delta}^2_k$ the damping factors associated to the operators T_k and in the respective subdomains Ω_1 and Ω_2, we have:

$$\hat{p}^{n+1,k}_{1|\Gamma_2} - \hat{P}^k_{\Gamma_2} = \hat{\delta}^1_k \left(\hat{p}^{n,k}_{2|\Gamma_1} - \hat{P}^k_{\Gamma_1}\right), \;\; \hat{p}^{n+1,k}_{2|\Gamma_1} - \hat{P}_{\Gamma_1} = \hat{\delta}^2_k \left(\hat{p}^{n,k}_{1|\Gamma_2} - \hat{P}^k_{\Gamma_2}\right). \tag{6.7}$$

These damping factors are computed analytically from the eigenvalues of the operators. We apply, then, the generalized Aitken acceleration separately to each wave coefficient in order to get the exact limit of the sequence on the interfaces, based on the first Schwarz iterate. The solution at the interface can then be recomposed in the physical space from its discrete trigonometric expansion. After we get these limits we use the LU block solver to compute the exact solution over Ω. In order to get rid of the problem of the non-uniqueness of the solution, we treat the zero mode aside and solve, at first, the equation with appropriate boundary conditions:

$$\frac{\partial^2 \hat{p}_0(x)}{\partial x^2} = \widehat{RHS}_0. \tag{6.8}$$

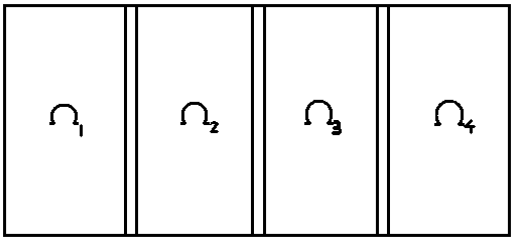


Figure 6.3: *Domain decomposion.*

To summarize, the algorithm writes:

- step 1: solve the mode zero one dimensional system and subtract this mode from the right-hand side.
- step 2: compute analytically each damping factor for each wave number.

- step 3: perform one additive Schwarz iterate in parallel (a LU solver is used as a block solver).
- step 4: apply the generalized Aitken acceleration on the interfaces.
  - 4.1: compute the cosine expansion of the traces of p on the artificial interfaces for the initial condition and the first Schwarz iterate.
  - 4.2: apply the generalized Aitken acceleration separately to each wave coefficient, in order to get the limit expressed in the cosine functions vector basis.
  - 4.3: transfer back the interface limit values into the physical space.
- step 5: compute the solution for each subdomain in parallel.

6.3.1 Parallel Speedup

Only two steps, 3 and 5 of the algorithm, are done in parallel, which are the most expensive ones, computationally, since they consist of solving the system over the subdomains. As we can see in Figure 6.4, the global performance of the method is encouraging. The time, measured with the MATLAB function *etime*, corresponds to the largest one among the processors clock times. The results are better with a larger problem size since the ratio between communication and computation is then reduced.
Remark: it is necessary to synchronize the processors with MATLABMPI in order to measure accurate computational times.

Steps 1,2 and 4 are not parallelized. Steps 1 and 4 contain a gathering of the interface data by one process and then a distribution of the updated interface data to the respective processes. The computational loads of these three steps are light, however, since they correspond to interface and not subdomain treatments.

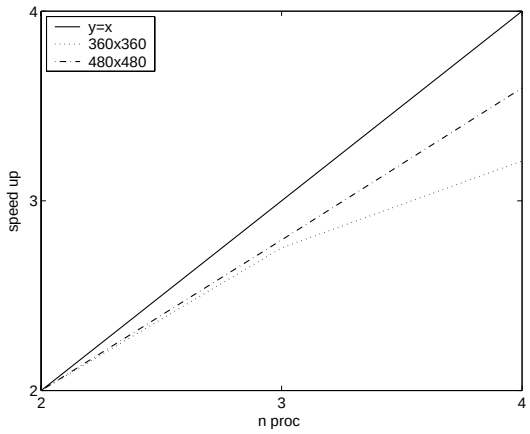


Figure 6.4: *Parallel speed up for the pressure solver with a LUP block solver, on Marvin.*

6.3.2 The LU block solver

Let us look at the computational time of the LUP solver in Figure 6.5. We can see on this figure that the time is of third order with respect to the size of a matrix and that it is divided linearly with respect to the number of subdomains: if we divide the domain into n subdomains, the LU solver will be n times faster. This is why we do not compare the speedup with the sequential code. The Aitken-Schwarz method requires solving the problem twice per subdomain: this would correspond ideally to the same computational time as the sequential code, but if we add the communication time in the Aitken-Schwarz parallel solver plus the zero mode treatment due to the non-uniqueness of the solution, the solver on two subdomains is significantly slower than the sequential code (by a factor 1.5 to 2).

Another aspect of the parallelization is that we gain computational time in the decomposition of the operator process (Figure 6.6): the time is divided by the square of the number of subdomains, since the complexity of the decomposition is proportional to the bandwidth of the subdomains. If we divide the domain in n subdomains, the LUP solver will be n^2 times faster. The computational time is of fourth order with

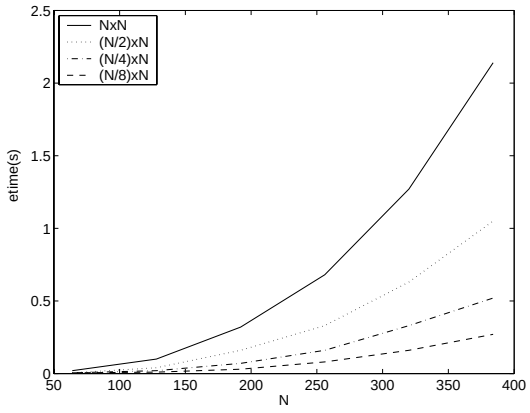


Figure 6.5: *LUP solver on Medusa.*

respect to the size of the matrix.

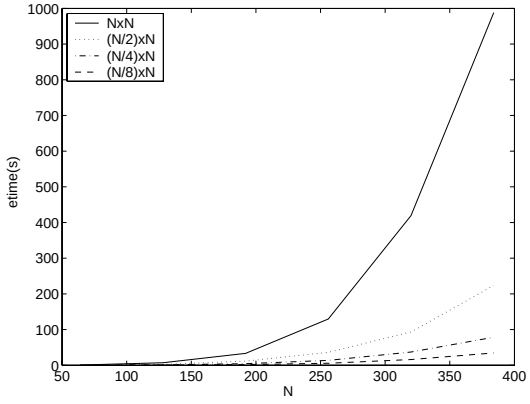


Figure 6.6: *LUP decomposition on Medusa.*

It is possible to use a faster LU solver offered by the MATLAB library: a LUPQ solver using the UMFPACK [22]. We can see the *LUPQ* decomposition time in Figure 6.7 and solving time in Figure 6.8.

However, in this chapter, we used the LUP solver in the speedup measures, in order to have a sufficient computational load per subdomain to overcome the communication latency. In order to have an interesting speedup, this computational time has to be

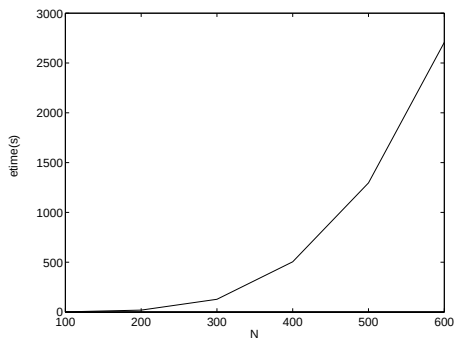


Figure 6.7: *LUPQ decomp. on Medusa.*

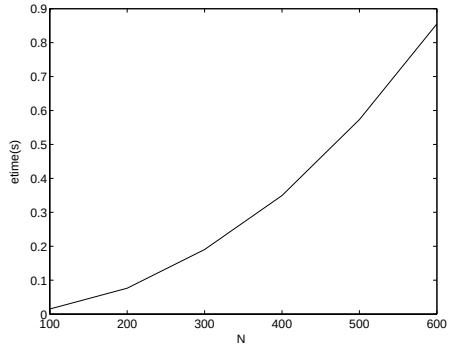


Figure 6.8: *LUPQ solver on Medusa.*

larger than a second, which is never the case with LUPQ solver and subdomain sizes smaller than 600×600. A Krylov solver might be more appropriate.

Remark: we implemented a *save* and *load* function for the matrix decomposition associated with each sub-domain, so that a LU decomposition for a specified problem size and number of processors, is done once and for all.

Now we study the numerical accuracy of this pressure solver in the IBM.

6.4 The IBM case

We use the parallel pressure solver code with a right-hand side, similar to the ones we find in the IBM, and more specifically in the 2D "bubble" test-case. The singular part of the right-hand side in the pressure equation comes from the divergence of the force term, which is a collection of weighted shifted discrete Dirac delta functions along a closed curve. We can observe in Figure 6.9 that even if we use regularized Dirac delta functions, we get a very stiff right-hand side, which corresponds to an irregular solution that you see in Figure 6.10. However, the observed error in the solution is extremely small, and located at the interface, when the immersed boundary crosses the subdomain interfaces. This is why this Aitken-Schwarz algortithm can be applied

to the pressure equation in the IBM.

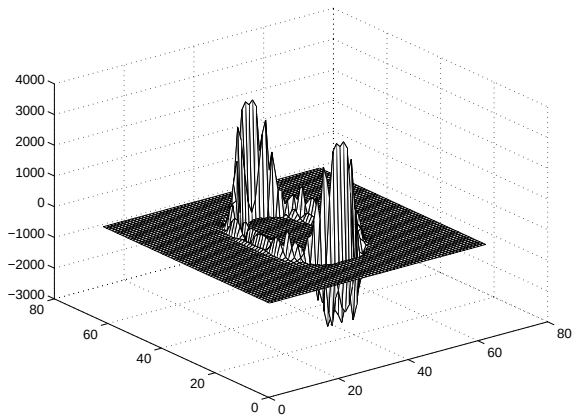


Figure 6.9: *Typical right-hand side of a pressure equation in the 2D "bubble" test-case. Elasticity coefficient of the immersed boundary* $\sigma = 10000$.

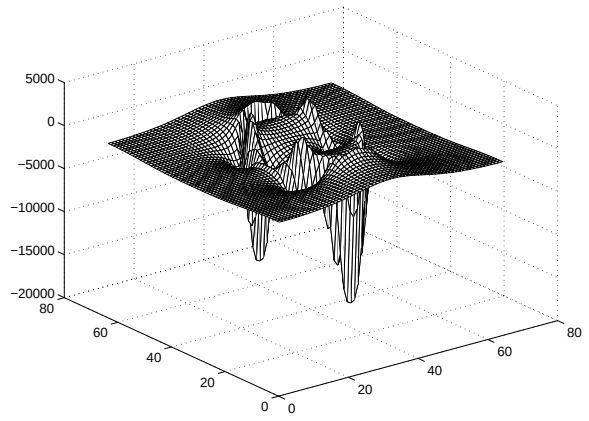


Figure 6.10: *Discrete solution of the pressure equation at a time step in the 2D "bubble" test-case. Elasticity coefficient of the immersed boundary* $\sigma = 10000$.

Now, we would like to apply the immersed boundary techniques to a blood flow simulation. Our aim is to do a multi-scale simulation of blood flow in an artery. The large scale would be a fluid/elastic-body simulation of the discontinuous periodic blood flow in the elastic artery. With the motion of the artery walls being limited, we choose the penalty method to simulate it. The small scale would be a model of the cells flowing and accumulating near the artery walls. Since the deformation of the cells are large, we want to simulate it using the IBM.

This is a long term project and so far, we started with a versatile NS code that can handle a broad variety of time dependent geometry, obtained from medical imaging. The large amount of computations can benefit from parallel computing. We saw that MATLABMPI has a large latency, but that the Aitken-Schwarz algorithm is efficient enough to overcome it. This is why we implemented a 2D Poiseuille flow with MATLABMPI, which brings great possibility of visualization.

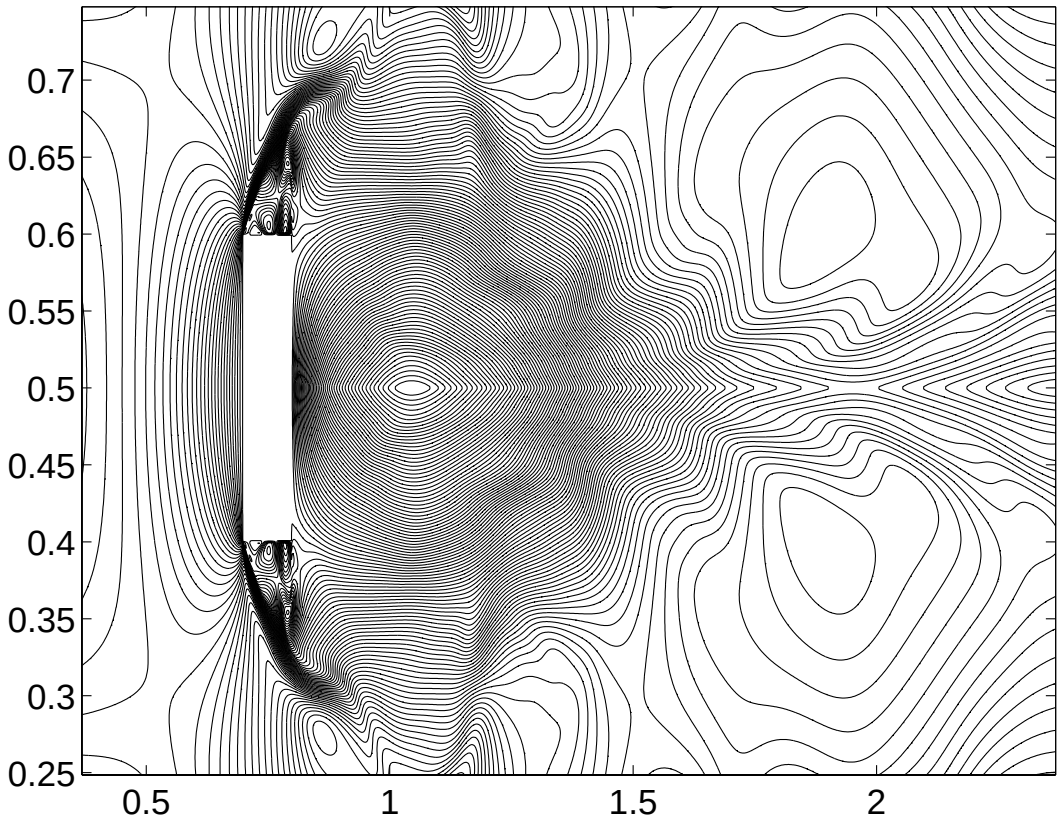


Figure 6.11: *Example of a NS computation done with MATLABMPI on 8 processors. 2D Poiseuille flow in a channel* $[0, 4] \times [0, 1]$. *Problem size:* 2400×600. *Contour of* u. *Rectangular obstacle simulated with direct forcing.*

Chapter 7

A Versatile Incompressible Navier-Stokes Solver for Blood Flow Application

M. Garbey†, F. Pacull

† Department of Computer Science, University of Houston, Houston, TX 77204, USA

We present, in this chapter, an integrated approach to quickly compute an incompressible NS flow in a section of a large blood vessel using medical imaging data. The goal is essentially to provide a first order approximation of some main quantities of interest in cardiovascular disease: the shear stress and the pressure on the wall. The NS solver relies on the L_2 penalty approach pioneered by Caltagirone and co-workers and combines nicely with a level set method based on the Mumford-Shah energy model. Simulations on Stenosis cases based on angiogram are run in parallel with MATLABMPI on a shared memory machine. While MATLABMPI communications are based on the load and save functions of MATLAB and have high latency indeed, we show that our Aitken-Schwarz domain decomposition algorithm provides a good

parallel efficiency and scalability of the NS code.

7.1 Introduction and Motivation

We present, in this chapter, an integrated approach to quickly compute an incompressible NS flow in a section of a large blood vessel using medical imaging data. The goal is essentially to provide a first order approximation of some main quantities of interest in cardiovascular disease: the shear stress and the pressure on the wall. From medical imaging one may expect to acquire the geometry of a large vessel, as well as the main flow components at the inlet and outlet of the region of interest of the artery. Eventually, one may obtain a video of the main motion in time of the artery if such motion is large enough.

We are interested in having an automated tool that provides a coarse approximation of the shear stress on an artery wall in the presence of a stenosis, for example.

We present a fast, versatile and robust NS solver that relies heavily on the L_2 penalty approach pioneered by Caltagirone and co-workers [26] and combines nicely with a level set method based on the Mumford-Shah energy model [110].

In this problem, the complexity may come from the geometry as well as the motion of the vessel. For example, the section of the coronary artery that lies on the surface of the heart displays some strong motion transverse to the main direction of the blood flow in the artery during the cardiac cycle.

Our flow solver is an immersed boundary like method [89]. The wall boundary condition is immersed in the Cartesian mesh thanks to a penalty term added to the momentum equation. This technique is simple and easy to implement. We will investigate in this chapter the accuracy, robustness and numerical efficiency of the method.

While all the results presented here are in two space dimensions, the method generalizes easily to three space dimensions. In particular there is no issue on mesh generation thanks to the penalty method [83, 26] that imposes automatically the no slip boundary condition on the wall. Further, following the suggestion of [101] one can easily take into consideration the wall motion. The drawback of the method is that one gets a lower order approximation of the solution [83]. However, due to the uncertainty of the medical imaging data as well as a number of biological unknowns such as the tissue constitutive laws, especially in the presence of a cardiovascular disease, we expect that the limit on the accuracy of our NS solver will not be the limiting factor in real clinical conditions. Further, we present a numerical technique to recover the shear stress on the wall that complies with the presence of the singular source term introduced in the momentum equation to enforce the no-slip boundary condition. Finally, we take full advantage of the regular data structure of the problem to use a domain decomposition (DD) algorithm that has high numerical efficiency and scales well with the MATLABMPI implementation of Kepner *et al.* [56].

The plan of the chapter is as follows. In Section 2, we formulate the problem and recall the penalty method. In Section 3, we discuss the discretization of the NS equations and the construction of the penalty term. In Section 4, we present the numerical solver and our computation of the shear stress on the wall. In Section 5, we give some numerical results with benchmark problems that are related to stenosis. Section 6 discusses the parallel implementation with MATLABMPI. Section 7 is our conclusion and proposes future directions of work.

7.2 Formulation of the Problem and Methods

Since we will be concentrating our study on large vessels, we use an incompressible NS fluid flow model [69, 122]. Improvement may be realized via some quasi-Newtonian

flow model with little additional computing effort [84]. There is an abundant literature on blood flow simulation in complex geometry pipe but confined by fixed walls. The published studies generally rely on the finite element/volume/difference approaches with irregular grid or finite volume/difference techniques with multi-block grid. However, high order method as spectral element for NS flow in complex geometry vessel is also feasible and can be numerically efficient - see for example [79] and its references. A main goal in such studies is to have a fast solver that provides a solution with desirable accuracy levels, which is achievable since so many factors related to moving walls are neglected. As discussed by Shyy *et al.* [102, 107], various techniques have been proposed in the literature to treat moving boundary problems. Categorically, we mention Lagrangian (moving grid), Eulerian (fixed grid) and mixed Lagrangian-Eulerian techniques. The Lagrangian approach has the advantage of being able to handle the boundary condition at the interface precisely without ambiguity. The remeshing strategy is employed to track the interface movement. However, if the shape variation is substantial, issues arise in regard to grid skewness, geometric conservation laws, and the need for generating new grid at every time instant. The Eulerian approach is, in general, of lower accuracy because it needs to reconstruct the interface without direct physical guidelines. The mixed Lagrangian-Eulerian approach is attractive as the unified approach because it combines the flexibility of a fixed grid with the merit of explicitly tracking the interface evolution.

For fast prototyping of incompressible NS flow we prefer to stay with the Eulerian approach and combine fast solvers for regular Cartesian grid solution with some form of fictitious domain decomposition or locally fitted stencil to implement the boundary conditions [37, 73, 74, 75, 40, 102, 107]. In this chapter we will use the penalty method introduced by Caltagirone and co-workers [26] that is simpler to implement than our previous boundary fitted methods [73] and applies naturally to flow in a pipe with moving walls [101].

The flow of incompressible fluid in a rectangular domain $\Omega = (0, L_x) \times (0, L_y)$ with prescribed values of the velocity on $\partial\Omega$ obeys the NS equations:

$$\partial_t U + (U.\nabla)U + \nabla p - \nu \nabla.(\nabla U) = f, \text{ in } \Omega$$

$$div(U) \ = \ 0, \text{ in } \Omega$$

$$U = \mathbf{g} \ \text{ on } \ \partial\Omega,$$

We denote by $U(x, y, t)$ the velocity with components (u_1, u_2) and by $p(x, y, t)$ the normalized pressure of the fluid. ν is a kinematic viscosity.

With an immersed boundary approach the domain Ω is decomposed into a fluid subdomain Ω_f and a wall subdomain Ω_w. In the L_2 penalty method the right hand side f is a forcing term that contains a mask function Λ_{Ω_w}

$$\Lambda_{\Omega_w}(x, y) = 1, \text{ if } (x, y) \in \Omega_w, \ 0 \text{ elsewhere},$$

and is defined as follows

$$f = -\frac{1}{\eta}\Lambda_{\Omega_w} \ \{U - U_w(t)\}.$$

U_w is the velocity of the moving wall and η is a small positive parameter that tends to zero.

A formal asymptotic analysis helps us to understand how the penalty method matches the no slip boundary condition on the interface $S_w^f = \bar{\Omega}_f \bigcap \bar{\Omega}_w$ as $\eta \to 0$. Let us define the following expansion:

$$U = U_0 \ + \ \eta \, U_1, \ p \ = p_0 \ + \ \eta \, p_1.$$

Formally we obtained at leading order,

$$\frac{1}{\eta}\Lambda_{\Omega_w} \ \{U_0 - U_w(t)\} = 0,$$

that is

$$U_0 = U_w, \text{ for } (x, y) \in \Omega_w.$$

The leading order terms U_0 and p_0 in the fluid domain Ω_f satisfy the standard set of NS equations:

$$\partial_t U_0 + (U_0.\nabla)U_0 + \nabla p_0 - \nu\nabla.(\nabla U_0) = 0, \text{ in } \Omega_f$$

$$div(U_0) \ = \ 0, \text{ in } \Omega.$$

At the next order we have in Ω_w,

$$\nabla p_0 \ + \ U_1 \ + Q_w = 0, \tag{7.1}$$

where

$$Q_w \ = \ \partial_t U_w + (U_w.\nabla)U_w - \nu\nabla.(\nabla U_w).$$

Further the wall motion U_w must be divergence free.
At the next order we have in Ω_f,

$$\partial_t U_1 + (U_0.\nabla)U_1 + (U_1.\nabla)U_0 + \nabla p_1 - \nu\nabla.(\nabla U_1) = 0,$$

with

$$div(U_1) = 0.$$

In the simplest situation where $U_w \equiv 0$, we observe that that the motion of the flow is driven by the pressure following a classical Darcy law. η stands for a small permeability. To summarize as $\eta \to 0$, the flow evolution is dominated by the NS equations in the artery, and by the Darcy law with very small permeability in the wall. This actually corresponds to a standard multiscale model of blood flow in the main arteries [1]. But we will use η as an artificial parameter rather than a measured

permeability of the wall. As a matter of fact the discretization used in this chapter is not appropriate to compute accurately all the space scales. From the analytical point of view it was shown in [83] for fixed wall, i.e. $U_s \equiv 0$, that the convergence order of the penalty method is of order $\eta^{\frac{3}{4}}$, in the fluid domain, and $\eta^{\frac{1}{4}}$ in the wall.

In this chapter we will restrict ourselves to the situation where the fluid domain traverses the rectangular domain $\Omega = (0, L_x) \times (0, L_y)$ in x direction, and stays at a finite distance from the horizontal boundary of the domain $y = 0$, and $y = L_y$. Ω decomposes into three narrow bands of length L_x that are respectively the lower wall Ω_w^{low}, the fluid domain Ω_f and the upper wall Ω_w^{upper}.

Figure 7.1 and Figure 7.2 give examples of the fluid flow domain in the benchmark problems of Section 5.

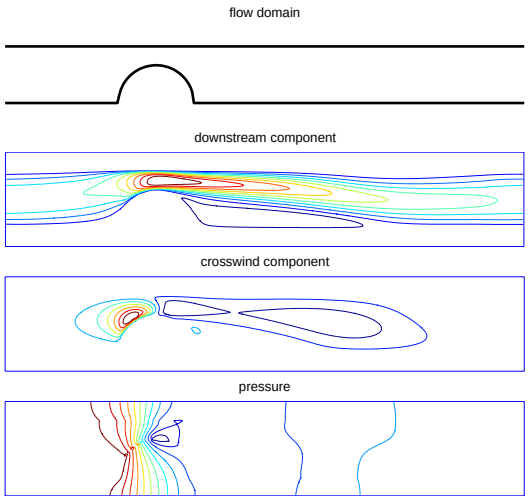


Figure 7.1: *Steady flow with a* 67% *stenosis.*

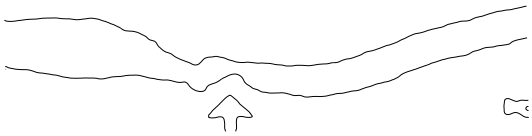


Figure 7.2: *Image segmentation.*

We impose inlet and outlet boundary conditions on the flow speed along the vertical wall $x = 0$ and $x = L_x$. The flow field at the inlet satisfies the Dirichlet boundary condition

$$u_1(0, y, t) = g(y, t),\ y \in (0, L_y).$$

g is set to zero inside the wall, i.e for y such that $(0, y) \in \Omega_w$.

For simplicity we impose a homogeneous Neumann boundary condition on the flow field, i.e $\frac{\partial u_1}{\partial x} = \frac{\partial u_2}{\partial x} = 0$ at the outlet $x = L_x$ and a constant profile of the pressure along the outlet wall $x = L_x$.

In the absence of accurate medical data on inflow and outflow boundary conditions for the section of interest of the vessel we expect to have a very simple laminar flow structure close to a Poiseuille flow near both ends of the section.

To minimize the presence of vortices, we further modify the NS equation in the neighborhood of the vertical wall $x = 0$, and $x = L_x$ by multiplying the convective term with a smooth function $\mathcal{H}(x)$ that is one inside the interval $(d, L_x - d)$ and zero in the interval $(0, \frac{d}{2}) \bigcup (L_x - \frac{d}{2}, L_x)$. The momentum equation writes:

$$\partial_t U + \mathcal{H}(\vec{U}.\nabla)U - \nu\nabla.(\nabla U) = f,\ \text{in } \Omega$$

The NS equations simplify then into the Stokes equation in the neighborhood of the inlet and outlet of the fluid domain Ω_f. To avoid reflection of acoustic waves at the outlet one may then use transparent boundary conditions [9].

On the horizontal wall we assume either the periodicity of all variables, or we impose the flow speed to be U_w.

Since the model has been completely defined, we are now going to present how the set of equations is discretized.

7.3 Discretization

The salient feature of our method is that the mesh generation is trivial no matter the wall location. The eventual complexity of the geometry of the fluid domain is taken care of by the forcing term in the penalty method which can be obtained directly from an image segmentation procedure.

The discretisation of the NS equations is done with finite differences on a Cartesian grid following the standard staggered grid method [91]. We define then the following grid function:

$$u_1\left(i\ h_x, \left(j+\frac{1}{2}\right)\ h_y\right),\ i=0\ldots N_x,\ j=0\ldots N_y-1,$$

$$u_2\left(\left(i+\frac{1}{2}\right)\ h_x, j\ h_y\right),\ i=0\ldots N_x-1,\ j=0\ldots N_y,$$

$$p\left(\left(i+\frac{1}{2}\right)\ h_x, \left(j+\frac{1}{2}\right)\ h_y\right),\ i=0\ldots N_x-1,\ j=0\ldots N_y-1$$

on the staggered mesh, of space step $h_x = L_x/N_x,\ h_y = L_y/N_y$.

The diffusion term in the momentum equation is discretized with second order central finite differences. For the convective term, we are using a method of characteristic that is first order in time and second order in space. To be more specific, let us consider the transport equation

$$\frac{\partial C}{\partial t} = u_1(x,y,t)\frac{\partial C}{\partial x} + u_2(x,y,t)\frac{\partial C}{\partial y}, \tag{7.2}$$

We use the first order approximation in time,

$$C(x,y,t^{n+1}) = C(x-u_1(x,y,t^n)\ dt, y-u_2(x,y,t^n)\ dt, t^n). \tag{7.3}$$

at every grid point. Since the velocity components are defined at different grid points of the staggered grid, we use a second order bilinear interpolation to project the velocity components at the $(x-u_1(x,y,t^n)\ dt, y-u_2(x,y,t^n)\ dt)$ location. This bilinear interpolation satisfies a maximum principle and the time integration scheme

is unconditionally stable. However, to keep the domain of dependency similar to the stencil used for centered finite differences, we choose a space step dt that satisfies the CFL condition.

Further improvement in the grid discretization implementation may use a high order interpolation scheme for the convective term that introduces less dissipation and a mesh refinement in the neighborhood of the interface S_w^f.

The mask function Λ_{Ω_w} is obtained with an image segmentation technique that is a level set method. Since the contours of the image are not necessarily sharp, it is interesting to use the level set method presented in [110] and based on the Mumford-Shah Model. For completeness we are going to describe briefly this method. We refer to the review papers [103, 55] for a more comprehensive description of the level set method in the framework of image analysis. Let us denote $C(s)$ the unknown parameterized curve(s) that delineate the vessel. We assume that the unknown function(s) $C(s) : [0,1] \rightarrow IR^2$ is a piecewise $C^1[0,1]$ function. In the level set method, $C(s)$ is represented by the zero level set of a Lipschitz function: $\phi : \Omega \rightarrow IR$.

$C(s)$ should correspond to the minimum of the energy $F(C, c_1, c_2)$:

$$F(C, c_1, c_2) = \mu_1.(length(C)) + \mu_2.(area(inside(C))) + \tag{7.4}$$

$$\lambda_1 \int_{inside(C)} |u_o - c_1|^2 \, dx + \lambda_2 \int_{outside(C)} |u_o - c_2|^2 \, dx, \tag{7.5}$$

where

$$c_1(\phi) = \frac{\int_\Omega u_0 \, H(\phi) \, dx}{\int_\Omega H(\phi) \, dx},$$

$$c_2(\phi) = \frac{\int_\Omega u_0 \, (1 - H(\phi)) \, dx}{\int_\Omega (1 - H(\phi)) \, dx}.$$

H is the Heaviside function $H(z) = 1, \text{ if } z \geq 0, \ 0 \text{ if } z < 0$. To understand the energy function used in this model, let us suppose that μ_1 and μ_2 are set to zero, and let us suppose that the image is a piecewise constant function with values 0 and 1.

The angiogram of an artery with a 50 percent stenosis given in Figure 7.2 is at first sight close to this description. Clearly the functions $C(s)$ that realize the minimum of energy are the boundaries that delimit the two sets $u_0 = 0$ and $u_0 = 1$. The first two terms in the energy model are common in many active contour methods and control the smoothness of the curve $C(s)$ as well as the detection of the edges.

The numerical process to compute ϕ uses the following evolution problem

$$\frac{\partial I}{\partial t} = N[I], \tag{7.6}$$

where N is the associated Euler Lagrange equation

$$N[\phi] = \frac{d}{d\,z}H(z)[\mu\ div\left(\frac{\nabla\phi}{|\nabla\phi|}\right) - \nu - \lambda_1(I-c_1)^2 + \lambda_2(I-c_2)^2],\ x \in \Omega.$$

In the numerical implementation one uses a regularization of the Heaviside function $H_\epsilon \in C^1(0,1)$ such that $H_\epsilon \to H,\ \text{as } \epsilon \to 0$, as well as a reinitialization procedure every few time steps that sharpens the level set function in the neighborhood of the zero level set.

To successfully apply the Chan-Vese method one must choose carefully the parameters of the method, i.e. $\lambda_1,\ \lambda_2,\ \mu_1,\ \mu_2$. From our experience, successful conditions on the parameters to detect the edge of a thin long object, are $\lambda_1 >> 1$ and $\lambda_2 = 1$. This follows somehow the fact that the area inside $C(s)$ is much smaller than the area outside $C(s)$. We will discuss further the robustness of this segmentation technique in Section 5.

The mask function Λ_{Ω_w} in the momentum equation is, consequently,

$$\Lambda(x,y) = H(\Phi(x,y)),\ (x,y) \in \Omega.$$

There is a natural advantage to combining the level set method with the penalty technique for NS, because the level set function provides directly the source term in the momentum equation. We do not need an explicit geometric representation of the

wall boundary, as a differentiable deformable model will do [23]. Further, any artifact possibly produced by the level set method such as an artificial closed subset of fluid in the wall, see Figure 7.2, should have little impact on the computation of the flow in the artery. As a matter of fact, such cavity has $U \approx U_s$ on all boundaries.
We have now described the complete discretization of the set of NS equations. For time stepping we have used in this chapter a projection scheme [11]. The time integration writes then

- step 1: prediction of the velocity $\hat{\mathbf{u}}^{k+1}$ by solving either

$$\frac{\hat{\mathbf{u}}^{k+1} - \mathbf{u}^{k,*}}{\Delta t} - \nu \Delta \mathbf{u}^{k} \tag{7.7}$$
$$= \mathbf{f}^{k+1} - \nabla p^{k} \text{ in } \Omega = (0, L_x) \times (0, L_y),$$

  or

$$\frac{\hat{\mathbf{u}}^{k+1} - \mathbf{u}^{k,*}}{\Delta t} - \nu \Delta \mathbf{u}^{k+1} \tag{7.8}$$
$$= \mathbf{f}^{k+1}1 - \nabla p^{k} \text{ in } \Omega = (0, L_x) \times (0, L_y),$$

  with boundary condition

$$\hat{\mathbf{u}}^{k+1} = \mathbf{g} \text{ on } \partial\Omega;$$

  and

$$\mathbf{f}^{k} = -\frac{1}{\eta} \Lambda_{\Omega_s} \{u^k - U_s(t^k)\}.$$

  $u^{k,*}$ is obtained with the method of characteristics (7.2, 7.3).

- step 2: projection of the predicted velocity to the space of divergence free functions

$$-div \nabla \delta p = -\frac{1}{\Delta t} div\, \hat{\mathbf{u}}^{k+1}, \tag{7.9}$$
$$\mathbf{u}^{k+1} = \hat{\mathbf{u}}^{k+1} - \Delta t \nabla \delta p, \quad p^{k+1} = p^k + \delta p. \tag{7.10}$$

We choose to use the semi-implicit scheme (7.8) instead of (7.7) if the mesh is fine enough to make the stability condition on the explicit treatment of the diffusion term too restrictive compared to the CFL condition. We have chosen *a priori* to stay with a first order scheme in time because of the uncertainty on the (pulsating) inflow velocity condition. The emphasis in this work is more on the robustness of the algorithm than on the accuracy.

We are now going to describe the solvers used to solve the set of discrete equations.

7.4 Solver

The NS calculation decomposes into three steps that are the prediction of the flow speed components, the solution of a Poisson problem for the pressure, and eventually the computation of the shear stress along the wall. We will review successively the numerical algorithm used at each step.

7.4.1 Solver for the Momentum Equation

The explicit time stepping of (7.7) does not require any solver. The semi-implicit scheme of (7.8) requires the solution of a linear system of equations corresponding to the discretization of the operator:

$$L = -dt\ \nu\ \Delta\ +\ c\ Id$$

where u is given from the previous time step, c is a coefficient that depends on the penalty term and Id is the identity operator. In practice $dt \approx min(h_x, h_y)$, $\eta \prec\prec 1$ and $\nu \preceq min(h_x, h_y)$. The discrete operator L^h is then close to the identity. The successive over relaxation scheme is numerically very efficient on such operators. For parallel computing, we may also adopt the modified Schwarz method of [37, 73] that has been specially designed for small viscosity flow with a main dominant direction

of convection as we have here. Further we know *a priori* that the velocity inside the wall is asymptotically close to U_w. Consequently, the velocity does not need to be updated at grid points inside the wall Ω_w that are few meshes away from the wall.
We will discuss now the solution process for the Poisson problem that requires *a priori* a larger number of floating points operations (flops) than for the momentum equation.

7.4.2 Solver for the Pressure Equation

The pressure equation can be integrated with a number of existing fast Poisson solvers since the discretization grid is regular. It is convenient, for example, to use a (full) multigrid solver here. The arithmetic complexity of this solver is optimum. Further, the iterative solver converges extremely fast for those grid points that are in the solid wall.
A parallel version of our solver uses the Aitken-Schwarz Domain Decomposition (DD) technique presented in [72, 74] that is in this situation a direct solver. The implementation will be detailed in Section 6.
We recall that, for a moderate number of subdomains, the overall arithmetic complexity of the Aitken-Schwarz algorithm is of the order of the arithmetic complexity of the fast Poisson solver applied to the subdomain factor the number of subdomains. In practice, the subdomains can be narrow bands, and it is not clear what should be the fastest subdomain solver on a given computer architecture. We choose the subdomain solver for the Poisson problems that provides the smallest elapse time. We refer to [74] for an extensive study of the performance, comparing several libraries invoking either an LU decomposition, a Krylov method, or a multigrid scheme. We will see later that our NS solver can perform very quickly the numerical simulation. However, the main difficulty consists of retrieving accurately a quantity of interest in blood flow calculations that is the shear stress on the wall boundary. We are going

to address this problem in the next section.

7.4.3 Computation of the shear stress

While the computation of the hydrodynamic forces exerted by the fluid on the wall is easy to compute using the integral on Ω_w of the penalty term [101], the computation of the shear stress is more problematic in the penalty method.

The shear stress in the boundary layer can be obtained from the formula

$$\tau = \nu \left(\frac{\partial w_1}{\partial \xi} + \frac{\partial w_2}{\partial \eta} \right),$$

where (ξ, η) is the normal/tangential coordinate system along the wall, and (w_1, w_2) are the components of the flow field along respectively the tangential and normal direction to the wall. Because the flow field (u_1, u_2) is computed on the Cartesian staggered grid we rewrite the shear stress formula in the (x, y) coordinate system. If α denotes the angle of the tangent to the wall at point $M(x, y) \in S_w^f$, with the horizontal axis x, we get

$$\tau = \nu \left(cos(2\alpha) \left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) + sin(2\alpha) \left(\frac{\partial u_2}{\partial y} - \frac{\partial u_1}{\partial x} \right) \right).$$

The flow field is continuous but not differentiable on S_w^f . We cannot therefore approximate the shear stress with some central finite differences formula that will require points on both sides of the wall. Further, the computed velocity exhibits small oscillations in the vicinity of S_w^f inside Ω_f because of the stiffness of the forcing term in the momentum equation at that location. We have tested two different methods to tackle this problem.

Method A follows a two step procedure. First the velocity variables are filtered using a high order filter. This regularization reduces the Gibbs phenomenon by improving the approximation away from the zone of discontinuities [42]. We recall that a real and even function $\sigma(\eta)$ is called a filter of order p if [42],

- $\sigma(0) = 1,\ \sigma^{(l)}(0) = 0,\ 1 \leq l \leq p-1,$
- $\sigma(\eta) = 0,\ for\ |\eta| \geq 1,$
- $\sigma(\eta) \in C^{p-1},\ for\ \eta \in (-\infty, \infty).$

Let us denote $\tilde{x} = \frac{2\pi}{L_x}x, \tilde{y} = \frac{2\pi}{L_y}y$. For commodity we set $n_x = \frac{N_x}{2},\ n_y = \frac{N_y}{2}$. We compute the discrete Fourier expansion

$$u(x,y,t) = \sum_{k_1=-n_x}^{n_x} \sum_{k_2=-n_y}^{n_y} \hat{u}_{k_1,k_2}(t)\ e^{i(k_1\tilde{x}+k_2\tilde{y})}.$$

The filtered function u^σ is

$$u^\sigma(x,y,t) = \sum_{k_1=-n_x}^{n_x} \sum_{k_2=-n_y}^{n_y} \sigma\left(\frac{|k_1|\kappa}{n_x}\right)\sigma\left(\frac{|k_2|\kappa}{n_y}\right)\hat{u}_{k_1,k_2}(t)\ e^{i(k_1\tilde{x}+k_2\tilde{y})}. \tag{7.11}$$

The parameter $\kappa \geq 0$ sets the level of cut in frequency space. As shown in [42], this filtering procedure efficiently reduces the impact of the Gibbs phenomenon on the accuracy of the approximation of a non smooth periodic function.

All pointwise derivatives $\frac{\partial u_{1/2}}{\partial x}$ and $\frac{\partial u_{1/2}}{\partial y}$ are computed with centered finite differences inside the domain of the fluid on those stencils that do not intersect the wall subdomain. We end up with the computation of an approximation of the first order derivatives of the flow speed at grid points of coordinate (x_i, y_j) that are one or two cells away from the wall boundary.

Second, we proceed with a linear extrapolation formula to approximate the Derivatives of Interest (DI) functions $\frac{\partial u_{1/2}}{\partial x}$ and $\frac{\partial u_{1/2}}{\partial y}$ at the wall location using exclusively the previous values of the derivatives inside Ω_f. For simplicity, we use an extrapolation formula along the vertical, horizontal or diagonal direction that is the closest to the local normal direction to the wall. For example, with $\alpha \in (0, \frac{\pi}{8})$ we use a linear extrapolation with the closest two grid values of the DI functions aligned along the vertical axis y. For $\alpha \in (3\frac{\pi}{8}, \frac{\pi}{2})$, we use the grid values aligned along the horizontal direction. For $\alpha \in (\frac{\pi}{8}, 3\frac{\pi}{8})$ we use the grid values at points (x_i, y_j) of the Cartesian grid aligned along the direction $y = -\frac{h_y}{h_x}x$.

Method B uses a gridless approach. Let R be the rectangle $(-d, d) \times (0, l)$ in the (ξ, η) coordinate system that is tangent to the wall at $M(x, y) \in S_w^f$ and lies inside the flow region. Let R_h be the set of grid points inside this rectangle completed by the grid points obtained from the intersection of the Cartesian mesh with the wall in the ball of center $M(x, y)$ and radius d. For each flow velocity components, one computes a second order polynomial approximation $P_M(x, y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 y^2 + a_5 x y$ that fits in the least square sense the flow field component on R_h. We notice that the least square approximation filters out the possible oscillation of the solution near the wall.

The derivatives of the velocity field are then approximated by the derivative of $P(x, y)$. The dimensions of the rectangle are as follows: the width of the rectangle R is chosen to include approximatively three grid points, i.e., $d = \sqrt{h_x^2 + h_y^2}$. The length l of the rectangle is chosen to be proportional to the boundary layer thickness, i.e., $l \approx \sqrt{\nu}$, for small ν, in order to capture accurately the parabolic profile of the flow field in the layer. l is, *a priori*, independent of the mesh size.

We have now described the method to compute the NS set of equations in a pipe of arbitrary shape that traverses the square subdomain Ω, and get an approximation of the shear stress along the wall boundaries.

In the next section we will report on the numerical accuracy obtained with this method.

7.5 Numerical Results

Let us first study the impact of the grid on the accuracy of the computation of the shear stress for a simple Poiseuille flow in a straight pipe.

7.5.1 Poiseuille Flow

We look first at the sensitivity of the result with various positions of a straight pipe of width 0.6 L_y included in the rectangle $(0, L_x) \times (0, L_y) = (0, 3.5) \times (0, 1)$. In this section, we use a square mesh, i.e., $h_x = h_y$ for all numerical experiments. The main issue is to experiment how robust the computation of the shear stress at the wall remains when the wall boundary location is arbitrarily set. There is no reason for which the wall of the pipe should coincide with the grid points of the Cartesian mesh. First, we keep the pipe parallel to the horizontal axis. In Figure 7.3, we test *method A* with various positions of the pipe and three levels of grid refinement. We observe, globally, the convergence of the method, and obtain the correct shear stress within five per cent of relative error with $N_y = 160$. *Method A* is, however, fairly sensitive to the choice of the cut off parameter κ of the filter (7.11).

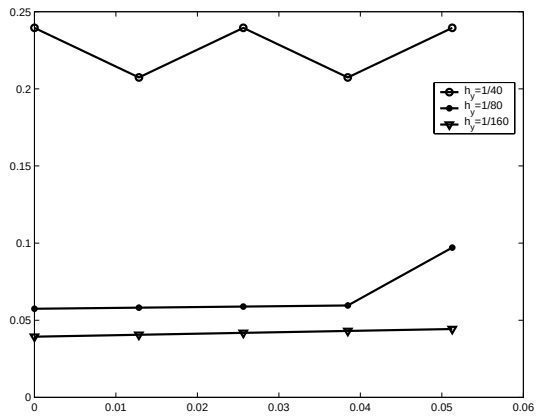


Figure 7.3: *Sensitivity of the computation of the shear stress on the wall as a function of the horizontal position with Method A.*

In a second set of experiments, we rotate the pipe that now forms an angle of measure α with the horizontal axis. In this experiment, α is kept in the range $(0, \frac{\pi}{4})$.
We achieve the same level of accuracy as before with *method A*, with a slightly finer mesh. In all simulations we have used a moderate level of cut off in the filtering with

$\kappa \approx 0.8$. With no filtering, the computation of the shear stress seems to give random results (see Figure 7.4). The flow solution exhibits small oscillations next to the wall, because of the singular source term in the momentum equation.

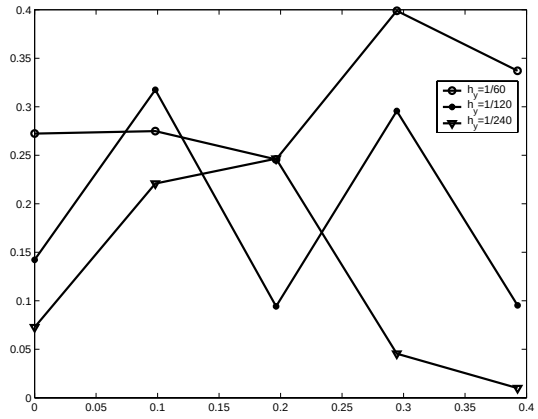


Figure 7.4: *Sensitivity of the computation of the shear stress on the wall as a function of the angle α with Method A but no filtering case.*

These numerical experiments with Poiseuille flow might be used to calibrate the filter (7.11) that can be used later for simulations with pipes that have complex geometry. We have compared *method B* with *method A*, for the same two sets of experiments. *Method B* can provide a more robust and accurate result in all situations (see Figure 7.5) provided that the dimension of the rectangle R used in the gridless approximation is set properly. The dimension l, in the direction orthogonal to the wall, is chosen to be mesh independent. From our numerical experiments, to choose l proportional to the boundary layer thickness $\approx \sqrt{\nu}$ seems optimal. Let us notice, however, that the number of grid points of R in the gridless approximation may become large when the space step is of the same order as ν.

We now report on computation with a more complex flow domain.

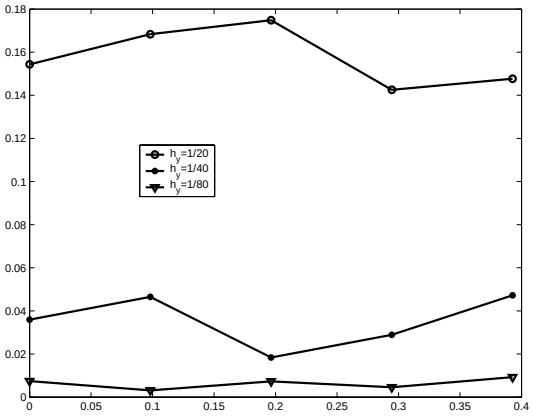


Figure 7.5: *Sensitivity of the computation of the shear stress on the wall as a function of the angle α with Method B.*

7.5.2 Benchmark Problems

We have first verified our code with a steady problem in a straight pipe of width 0.6 L_y obstructed by an obstacle that is the upper half of a disk. The radius of the disk is 0.3 L_y, and the obstacle mimics a fifty percent stenose. The center of the disc is at coordinates $(\frac{L_x}{2}, 0)$. We took $L_x = 3.5$, $L_y = 1$, and the viscosity is set to $\nu = 0.01$. The velocity profile at the left entry of the pipe corresponds to a Poiseuille flow with a velocity of maximum value one.

The steady solution is approximated by time marching until $t = 10$. Several finite element solutions with up to 5 10^4 elements have been computed with the software package ADINA (http://www.adina.com/) for comparison purposes. We verified that our code converges to the same solution with a convergence order that is about one in the L_∞ norm for the velocity components, and a slightly better order of convergence order in the L_2 norm. We get similar results for the pressure field in the L_2 norm, but notice that the pressure may exhibit some sharp singular peaks at some of the grid points that are next to the wall of the disk. Because we know *a priori* that the pressure should stay continuous at the wall interface, it is straightforward to filter

out numerically these singular grid point values.

Figure 7.1 reports on a stiffer steady problem where $\nu = 0.003$ and the stenosis reaches 67%. We see clearly the recirculation zone right after the obstacle. For smaller viscosity, we observe in our numerical experiments a secondary recirculation zone near the upper side of the wall downstream, and the solution becomes eventually unsteady. We have tested unsteady flows corresponding to a velocity profile at the inlet that is a Womersley solution for a two dimensional pulsating flow between plates [58]. This solution might be obtained with the method of separation of variable $u_1(0, y, t) = u_o(y)\ g(t)$ applied to the Stokes problem. In our simulation, $g(t)$ is provided by a measurement of the blood flow main velocity component with a healthy human subject (see Figure 7.6).

The geometry of the pipe is given in Figure 7.6 - postion A, and can be deformed eventually by a periodic smooth motion in time. To be more specific, the velocity field applied to the wall is parallel to the y axis, and given by the following formula:

$$U_w(x,t) = (0, u_{2w}), \text{ with} \tag{7.12}$$
$$u_{2w}(x,t) = U^o \left(exp\left(-\frac{(x - \frac{L_x}{2})^2}{\mu} \right) - (a\ x\ +\ b) \right) \cos(2\pi t).$$

The parameters a and b are chosen such that the wall at both ends of the pipe stays steady, i.e., $u_{2w}(0, y, t) = u_{2w}(L_x, y, t) = 0$. L_x is large and μ is small enough to have a and b close to zero. One can check that U_w is divergence free.

The sharpness of the wall curvature in its motion is set by μ and the amplitude of the motion is approximatively $\frac{U^o}{\pi}$. Figure 7.6 shows the two extreme positions of the pipe in this oscillatory motion of the wall. Figure 7.7, (respectively Figure 7.8) shows the shear stress computed on the lower wall for the wall position as in Figure 7.6 (respectively, the wall oscillations between position A and B at speed (7.12)). We observe a periodic solution in time that has a strong pique in space at the location of the stenose. The periodic motion of the wall adds a 12 percent increase on the shear stress versus the solution with static wall as in position A.

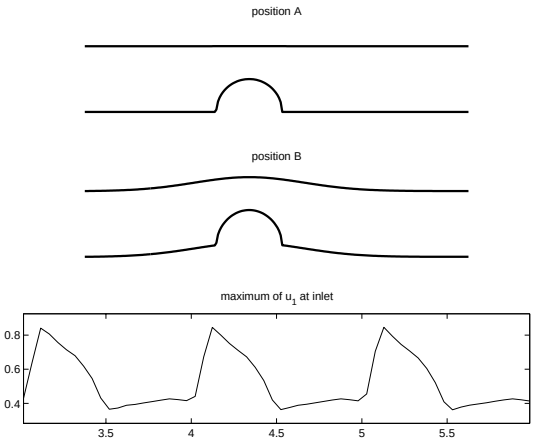


Figure 7.6: *Pulsating flow problem.*

As seen in the contour plot of Figure 7.9, the shear stress computation becomes noisier with the wall motion. In a real life situation, the coronary section that is lying on the heart has motions with fast acceleration periods followed by slow relaxation synchronized with the cardiac cycle. We can check with our numerical experiment that the shear stress at the wall is strongly affected by this motion.

Next, we have computed a steady flow in a pipe that is obtained from the image segmentation of the two dimensional angiogram picture of a carotid. We recall that an angiogram uses x-rays to visualize blood vessels. To create the x-ray images, the physician injects a dye through a catheter that has roughly a one millimeter diameter. This dye, called contrast, makes the lumen of the vessel visible on an x-ray.

Figure 7.2 shows the result of the segmentation with the method of Chan and Vese [110]. One artifact in this image segmentation corresponds to an annotation of the angiogram with an arrow. A second artifact at the lower right corner of Figure 7.2 comes from the poor quality of the angiogram itself. We impose a Poiseuille-like boundary condition on the inflow at the inlet of the vessel and zero velocity boundary condition elsewhere along the vertical line $x = 0$. Our flow server based on the L_2 penalty method does not suffer from the two artifacts as shown in Figure 7.10.

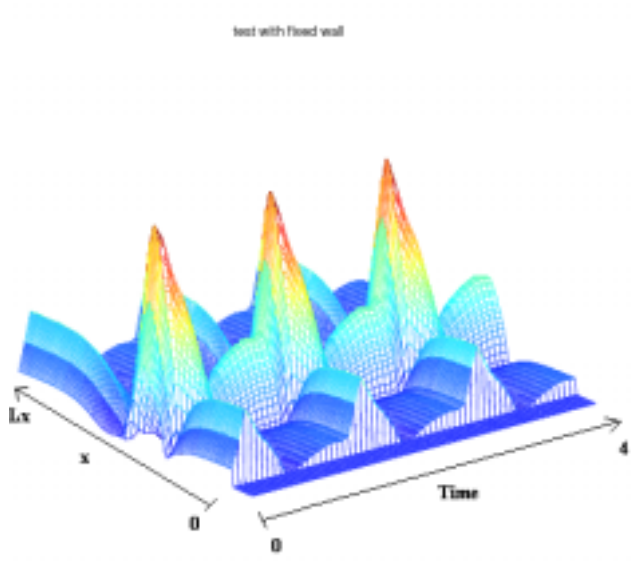


Figure 7.7: *Shear stress on the lower wall.*

The result of the image segmentation is also relatively sensitive to the choice of the parameters $\lambda_1,\ \lambda_2, \mu_1,\ \mu_2$ in (7.4,7.5). The sensitivity of the flow simulation to the parameter of the image segmentation can be analyzed experimentally or possibly with the automatic differentiation of our NS code that has a very simple implementation. The dynamic of the propagation of the dye from the catheter appears clearly during the beginning of the angiogram examination. It is easy to simulate this phenomenon that is essentially governed by the transport of mass with the method of characteristic already used in the resolution of the momentum equation. We have done numerous simulations of this type for our benchmark problems.

The main difficulty remains the validation of the image segmentation that also depends strongly on the quality of the contrast in the x-ray picture.

To validate the medical image analysis, we propose to couple our NS solver with an optimization procedure that matches the flow solution with the measurement of the blood flow speed at a few locations in the vessel using ultrasound techniques for example. This validation method, that will be the subject of future investigation in

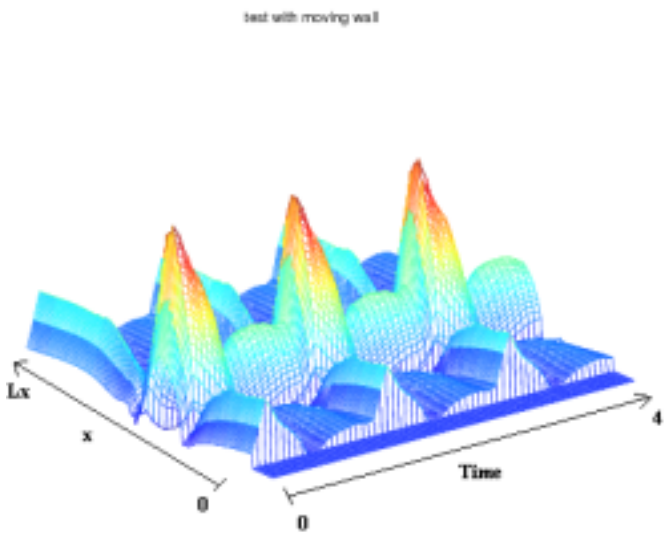


Figure 7.8: *Shear stress on the moving lower wall.*

our work, requires the use of three dimensional computations coupled to the image segmentation of three dimensional angiogram. However, the efficiency of the optimization procedure depends very much on our ability to have a very fast solver of the NS flow with complex geometry. To address this issue, we next present a parallel implementation of our NS code.

7.6 Parallel Performance

Our goal is to efficiently compute large-size problems taking advantage of the interactivity of MATLAB and its simplicity for coding.

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization and numerical computation that is widely used by computational scientists and engineers. It is user-friendly and offers a wide array of pre-defined functions. The drawback is that it is an interpreted language, not suited for iterative tasks: it must interpret every line of a loop and

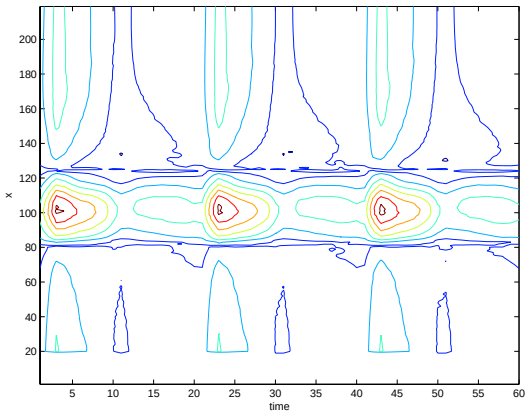


Figure 7.9: *Contour plot of the Shear stress on the moving lower wall.*

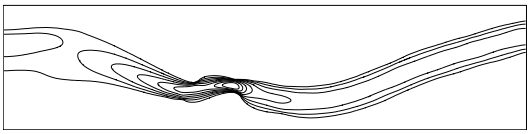


Figure 7.10: *Flow amplitude for a stationary problem.*

therefore cannot perform individual operations as fast as other languages such as C or Fortran. In our experience, the MATLAB compiler does not seem to provide much faster codes. We choose here to use the MPI extension of MATLAB described in [56]. This software library provides a wonderful framework to produce quickly a parallel code that takes advantage of all the memory available on a shared memory system. MPI is the *de facto* standard for communication in parallel scientific applications [33]. MATLABMPI [56] is a set of MATLAB scripts that implements a subset of MPI and allows any MATLAB program to be run on a parallel computer. The MPI extension of MATLAB is public domain and does not require anything other than a standard MATLAB license. This software is designated for users who want to do simple parallelism for codes that do not require a very small latency for the message-passing. This approach gives access to larger scale simulation with a minimum of time spent in the code development.

The message-passing in MATLABMPI is done via the file system: when a processor sends a message to another processor, it writes the data in a file in a common communication directory. A processor that receives a message should read a file from this common directory. Communication on a cluster cannot be faster than the file server, which updates the directory in each node. The communication within a Shared-memory Multi Processor (SMP) system is faster though, since the processors share a local directory. Still, there is a latency due to the detection and writing/reading time. The actual codes for the MPI_Send and MPI_Recv use file I/O: the *load* and *save* functions of MATLAB. The sender creates two files in the common communication directory, one *lock* file and one *buffer* file. The receiver must detect the *lock* file and then load the data from the *buffer* file. This technique is efficient for very large messages since there is no buffering. It is recommended then to group small size messages into a larger package and send it all at once, whenever the algorithm allows to do so. Another way to decrease the latency with the MATLABMPI implementation of [56] is to create a virtual communication folder on the RAM. Access to the main memory should be faster than on the hard drive.

For reference, our computer plate-form is an 8-way Opteron running at 2.0 GHz with 32 GB of main memory. Figure 7.11 reports on the performance of the message passing with MATLABMPI on this system. N is the size of the one dimensional array that is communicated. The clock time is measured using the MATLAB function *etime* for a small MATLAB program that sends the one dimensional array to a given processor and receives it back from the second processor. This elapsed time is divided by two and given in Figure 7.11 as a function of N.

We observe a latency that is about 2.10^{-3} s using a communication directory in the hard drive and 1.10^{-3} s using the RAM. There is not as much improvement as one may expect because the load and save procedures have inherently large overhead. Further, we found that the latency can be significantly larger for message-passing

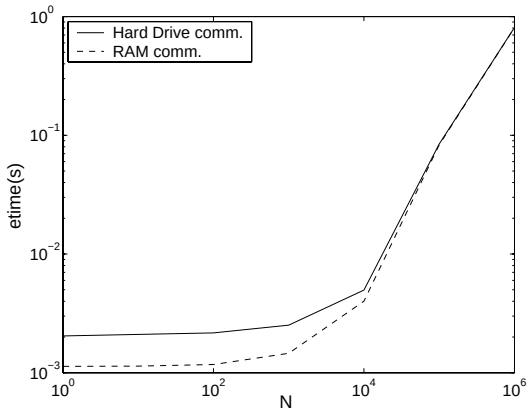


Figure 7.11: *Clock time for one message-passing in MATLABMPI using either a communication directory on the hard drive or in main memory.*

between cluster nodes linked by a Gigabit ethernet switch.

We are now going to describe in more detail the parallel implementation of our NS solver. For simplicity, we restrict ourselves to explicit time stepping in the momentum equation. It should be noticed that the pressure correction step is always the most computationally expensive step of the projection scheme. Consequently, we will concentrate now on the parallelization of the pressure equation solver (7.9).

We use the analytic additive Aitken-Schwarz algorithm [72] that is designed to combine efficient distributed computing with high latency networks and numerical efficiency. To clarify the parallel implementation, let us recall the algorithm that has been presented in more detail in [72].

The rectangular uniform mesh is decomposed into a unidirectional partition of overlapping strip domains. The method is a post-process of the standard additive Schwarz algorithm. An Aitken-like acceleration is applied to the sequences of interfaces produced with the block-wise Schwarz relaxation. Because the eigenvectors of the (linear) trace transfer operator are known, this acceleration provides the *exact* interface condition in one single step, no matter the overlap between subdomains. For simplicity, we restrict ourselves in this brief description of the algorithm to a decomposition of

Ω into two overlapping subdomains: $\Omega = \Omega_1 \cup \Omega_2$ where $\Omega_1 = [0, x_r] \times [0, 1]$ and $\Omega_2 = [x_l, 1] \times [0, 1]$, $x_l < x_r$. We suppose also that the problem to be solved has homogeneous Dirichlet boundary conditions along the vertical sides $[x_l] \times [0, 1]$ and $[x_r] \times [0, 1]$ and homogeneous Neumann boundary conditions on the horizontal sides of the rectangular domain. The additive Schwarz algorithm consists of repeating the following iteration:

$$\Delta p_1^{n+1} = RHS \ \text{ in } \ \Omega_1, \ \ \Delta p_2^{n+1} = RHS \ \text{ in } \ \Omega_2, \tag{7.13}$$

$$p_{1|\Gamma_1}^{n+1} = p_{2|\Gamma_1}^{n}, \ \ p_{2|\Gamma_2}^{n+1} = p_{1|\Gamma_2}^{n},$$

until convergence. In practice this algorithm is applied to compute the pressure correction δp rather than the pressure itself. A natural initial condition for the iterative solver is then $p_{1|\Gamma_1}^0 = p_{2|\Gamma_2}^0 = 0$.

We observe that a cosine expansion of the trace of the solution on the interface:

$$p(y)_{|\Gamma_i} = \sum_{k=1..N_y-1} \hat{p}_{|\Gamma_i}^k cos(k\pi y) \ , \ \ i = 1 \ \text{ or } \ 2, \tag{7.14}$$

provides a diagonalization of the trace transfer operator:

$$\left(p_{1|\Gamma_1}^n, p_{2|\Gamma_2}^n\right) \stackrel{T}{\rightarrow} \left(p_{1|\Gamma_1}^{n+1}, p_{2|\Gamma_2}^{n+1}\right). \tag{7.15}$$

The Poisson problem satisfied by the pressure decomposes onto a set of independent two point boundary value problems:

$$\frac{\partial^2 \hat{p}_k(x)}{\partial x^2} - \mu_k \, \hat{p}_k(x) = \widehat{RHS}_k, \ \ \forall k. \tag{7.16}$$

Let us denote T_k the trace operator for each wave component of the interface:

$$\left(\hat{p}_{1|\Gamma_1}^{n,k} - \hat{p}_{\Gamma_1}^k, \hat{p}_{2|\Gamma_2}^{n,k} - \hat{p}_{\Gamma_2}^k\right) \stackrel{T_k}{\rightarrow} \left(\hat{p}_{1|\Gamma_1}^{n+1,k} - \hat{p}_{\Gamma_1}^k, \hat{p}_{2|\Gamma_2}^{n+1,k} - \hat{p}_{\Gamma_2}^k\right), \ \ \forall k. \tag{7.17}$$

The operators T_k are linear and the sequences $\left\{\hat{p}^n_{1|\Gamma_1}\right\}$ and $\left\{\hat{p}^n_{2|\Gamma_2}\right\}$ have linear convergence. This is expressed by the set of linear equations

$$\hat{p}^{n+1,k}_{1|\Gamma_2} - \hat{p}^k_{\Gamma_2} = \hat{\delta}^1_k \left(\hat{p}^{n,k}_{2|\Gamma_1} - \hat{p}^k_{\Gamma_1}\right), \;\; \hat{p}^{n+1,k}_{2|\Gamma_1} - \hat{p}^k_{\Gamma_1} = \hat{\delta}^2_k \left(\hat{p}^{n,k}_{1|\Gamma_2} - \hat{p}^k_{\Gamma_2}\right), \tag{7.18}$$

where $\hat{\delta}^1_k$ and $\hat{\delta}^2_k$ are the so called damping factors associated with each subdomain Ω_1 and Ω_2.

These damping factors are computed analytically from the eigenvalues of the operators. We apply a generalized Aitken acceleration separately to each wave coefficient in order to get the exact limit of the sequence on the interfaces based on the first Schwarz iterate. It consists of simply solving the 2×2 linear system (7.18) where $n = 0$, of unknown $(\hat{p}^k_{\Gamma_1}, \hat{p}^k_{\Gamma_1})$. The same procedure applies with an arbitrary number of subdomains but then the matrix of the linear system corresponding to (7.18) has a pentadiagonal structure. Finally the exact solution at the artificial interfaces Γ_i is reconstructed in physical space from its discrete trigonometric expansion.

For the nonhomogeneous mixed boundary conditions of our pressure problem (7.9), we must add a preprocessing step that consists of solving the zero mode equation:

$$\frac{\partial^2 \hat{p}_0(x)}{\partial x^2} = \widehat{RHS}_0, \tag{7.19}$$

with $\frac{\partial \hat{p}_0}{\partial x} = 0$ at $x = 0$ and $\hat{p}_0(L_x) = 0$.

To summarize, the algorithm writes:

- prelim. step: compute analytically each damping factor for each wave number.
- step 1: solve the mode zero one dimensional equation.
- step 2: perform one additive Schwarz iterate in parallel.
- step 3: apply the generalized Aitken acceleration on the interfaces.

  - 3.1: compute the cosine expansion of the traces of p on the artificial interfaces for the initial condition and the first Schwarz iterate.
  - 3.2: apply the generalized Aitken acceleration separately to each wave coefficients in order to get the limit expressed in the cosine functions vector basis.
  - 3.3: transfer back the interface limit values into the physical space.
- step 4: compute the solution for each subdomain in parallel using the boundary values obtained from step 3.

In our MATLABMPI implementation Steps 1 and 3.2 are not parallelized and computed redundantly by all the processors. An all-to-all Broadcast through cyclic reduction is used to gather the zero mode component of the right-hand side and to build the artificial interface matrix.

Besides the pressure equation (7.9), the prediction and correction steps of the projection scheme are also parallelized: this implies one extra communication process with the neighboring subdomains at the beginning of those two steps, in order to build the right-hand side of the equation. We used a fast LUPQ solver offered by MATLAB that is based on the UMFPACK library [22] to solve each subdomain. The operator matrix M is decomposed into a unit lower triangular matrix L, a upper triangular matrix U, a permutation matrix P and a column reordering matrix Q so that $PMQ = LU$, using the fact that it is a very sparse matrix. This factorization, based on a tree and special ordering, optimizes the memory access patterns. We notice that the decomposition is computed once and for all in the initialization phase of the NS code. The pressure solver reuses this decomposition at every time step. The UMFPACK decomposition provides particularly good performance in MATLAB compared to the original Linpack LU solver that is typically two to three times slower for the size of the problem considered here.

Overall the Aitken-Schwarz parallel solver is a direct solver and takes the exact same elapse time at each time step.

Figure 7.12 and Figure 7.13 report respectively on the speedup and scalability of our parallel implementation of the complete NS code. In the scalability test, we have successively run a problem of size 141×567 with two processors, 200×801 on 4 processors, and 283×1129 on 8 processors. The number of unknowns grows linearly with the number of processors, but the aspect ratio h_x/h_y of the grid stays the same. The scalability performance obtained in Figure 7.13 is particularly good because the arithmetic complexity of a NS solver in general grows faster than linearly, with respect to the number of unknowns.

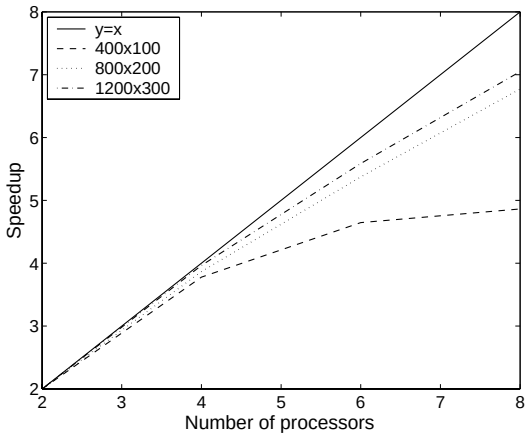


Figure 7.12: *Parallel speed-up for the parallel NS code.*

Let us notice that one should synchronize the processors in order to measure accurate computational times, since the MATLAB application starting time can vary. Performance speedup is based on the reference time provided by the code running with two subdomains on two processors. This speedup is therefore significantly better than what one obtains by comparing our parallel code with its sequential version. The speedup of the parallel code with two processors compared to the sequential code running on one processor is only 1.56. The overhead in the sequential code comes

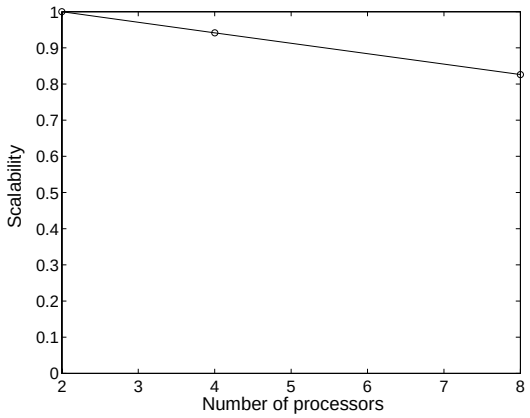


Figure 7.13: *Parallel scalability for the NS code.*

partially from the nature of the algorithm itself, which requires two subdomain solves, and partly from the fact that the parallel code has many more lines of code to be interpreted. Another aspect of the parallelization is that we gain computational time in the decomposition of the operator process: this time is divided by the square of the number of subdomains since the complexity of the decomposition is proportional to the bandwidth of the subdomains. While the speedup obtained in Figure 7.12 is not optimum, the MATLABMPI implementation still seems very attractive and allows for solving large problem size.

To give an overall idea of the MATLABMPI code performance, 100 time steps for a 100×400 problem size takes less than 10 seconds on 8 processors. This is roughly the elapsed time needed to simulate one complete cardiac cycle with $\nu \approx 10^{-2}$.

A parallel implementation in Fortran or C can be further improved by taking advantage of the more efficient communication schemes offered by MPI for these programming languages. One can also adapt the communication scheme in such a way that high frequency components of the interfaces are exchanged between neighbor subdomains only. We refer to the optimized implementation presented in [81] for grid computing that was done for three space dimension elliptic problems. We will now summarize the conclusion of our investigation.

7.7 Conclusion

We have presented in this chapter an integrated approach to compute quickly an incompressible NS flow in a section of a large blood vessel using medical imaging data. The key feature of the method is to use the L_2 penalty method pioneered by Caltagirone and co-workers (83). The first two major advantages of the method are the easiness of the implementation and a discretization that allows the use of fast elliptic solvers optimized for cache memory access. Second, our numerous numerical experiments have shown that the numerical method is fairly robust. Third, the penalty method combines naturally with a level set method that directly provide the penalty term in the momentum equation. Further, while the numerical method is first order we can surprisingly recover a reasonable estimate of the shear stress on the wall using a gridless approach. Finally, we have presented a straightforward parallelization of the NS code based on the Aitken-Schwarz algorithm and MATLABMPI. The MATLAB language allows a fast prototyping of the code while MATLABMPI offers the possibility to easily access a large amount of memory. The drawback of the approach followed in this chapter is obviously the fact that we cannot solve accurately the boundary layers that may appear in the flow field. This type of simulation should then be limited to moderated values of the Reynolds number. We are currently developing a multiscale heterogeneous domain decomposition version of our code to address this problem [75]. Our next step is, however, to generalize the present work to simulation in three space dimensions by extending the capability of our parallel 3D NS solver presented in [37, 73] to blood flow simulation related to real clinical cases.

Chapter 8

Conclusion and research directions

In this chapter, we list the results obtained during this work and explore the future research directions.

Regarding the IBM, we introduced [34] and studied the piecewise cubic Dirac delta function, already used to solve elliptic equations with singular source terms [111]. We showed that even if this delta function does not satisfy the compatibility condition defined by C.S. Peskin [89], it can be used in the IBM, but should be associated with the staggered mesh. It improves the accuracy of the method and is quickly evaluated, but decreases the stability of the IBM, as well as its volume conservation property, due to the stiffness of the delta function, compared to the commonly used ones.

We studied a fast multigrid solver to solve elliptic equations with singular source terms: the multigrid/τ-extrapolation technique, introduced by U. Rude [98]. This solver has a fast convergence and improves the discrete solution accuracy, while being easy to implement. Unlike in the traditional algorithm, we used the fact that we can discretize the singularity on different grids, instead of interpolating the fine grid discretization on the coarse. However, this method is based on the knowledge of the convergence order of the discrete solution, which is space-dependent in the case of

elliptic equations with singular source terms: it decreases in the neighborhood of the singularity. The use of the sharp piecewise cubic Dirac delta function narrows this neighborhood, but we do not get uniform improvement of the accuracy using this solver. We did implement this method with a space-dependent coefficient for the extrapolation, but the space-dependent asymptotic order of the discrete solution still has to be evaluated, for elliptic equations with singular source terms.

We introduced and justified the use of Fourier expansions to describe the moving interface. This has several applications regarding the IBM. The first is the filtering of the discrete moving boundary point oscillations: the high frequencies can be easily eliminated. The second is the global volume conservation of the domain topology, using constrained optimization on the compact representation of the interface.

A non-centered stencil for the divergence operator has been developed in order to solve the IBM pressure equation, using the position of the moving boundary. Since there is commutativity of the discrete Laplace and divergence operators, we can apply the modified divergence operator after we solve the Poisson equation in each direction of the space. This brings a uniform second-order convergence for the accuracy for non-smooth solutions. We implemented this technique in the IBM case but still need to find a way to obtain a divergence-free velocity vector field form the pressure field. The extrapolation technique used to compute the non-centered divergence needs to be studied too, in order to get a robust and simple technique.

We implemented a fully implicit scheme for the IBM, based on a quasi-Newton solver and the compact Fourier representation of the immersed boundary. This gives good results regarding the stability but still requires a lot of NS evaluations per time step. It only saves computations in very stiff cases, in which the elasticity coefficient of the membrane is very large. This method can be associated with a domain decomposition

method: an explicit solver would be used in the sub-domains that do not contain a sharp interface, while this implicit solver will be used in the other sub-domains, in order to deal with the stiff local NS system.

We implemented a fluid solver using a parallel MATLAB toolbox, MATLABMPI, on a shared-memory machine. The algorithm we used is the analytic Aitken acceleration of the Schwarz algorithm, which is efficient and not severely penalized by the slow latency of the toolbox. A parallel MATLAB toolbox is convenient for quickly parallelizing an existing MATLAB code or to test a parallel algorithm, on a large memory array. We actually used it along with the penalty method, to implement a technique to quickly compute an incompressible NS flow in a section of a large blood vessel using imaging data. This allows us to recover a reasonable estimate of the shear stress on the artery wall. Now that we showed that the Aitken-Schwarz algorithm is robust, regarding the stiff right-hand side of the IBM, we would like to implement a parallel IBM with MATLABMPI or a similar parallel MATLAB toolbox.

Finally, we would like to study the contact between immersed boundaries or between an elastic boundary and a solid obstacle, using artificial boundary methods for both the fixed and moving objects. This could lead to a simulation of the large blood cells transport in plasma.

Chapter 9

Appendix

9.1 Derivation of the incompressible Navier-Stokes equations

Let us derive the 2D incompressible NS equations, in primitive variables. V is the velocity vector. Its x and y components are u and v: $V = [u, v]^T$. P is the fluid pressure field. At first, we consider a rectangular control volume with side lengths Δx and Δy.

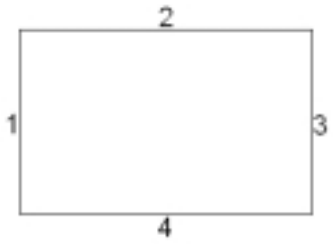


Figure 9.1: *2D control volume of width Δx and height Δy with indexed sides.*

9.1.1 The Conservation of mass property

The first fluid property that we use, is the conservation of mass property: *the rate at which mass increases within the control volume is equal to the rate at which mass enters the control volume through its four boundaries.*

The mass within the control volume is $\rho\Delta x\Delta y$, where ρ is the density parameter of the fluid, measure of mass per unit of volume. We consider the density to be a constant with respect to space and time: the fluid is incompressible and homogeneous. Thus the rate at which mass increases within the control volume is null:

$$\frac{\partial\left[\rho\Delta x\Delta y\right]}{\partial t} = \Delta x\Delta y\frac{\partial\rho}{\partial t} = 0. \tag{9.1}$$

The rate at which mass enters each boundary is:

- boundary 1: $+\rho u_{|1}\Delta y$,
- boundary 2: $-\rho v_{|2}\Delta x$,
- boundary 3: $-\rho u_{|3}\Delta y$,
- boundary 4: $+\rho v_{|4}\Delta x$.

So the rate at which mass enters the whole control volume is null:

$$\rho\Delta y\left(u_{|1} - u_{|3}\right) + \rho\Delta x\left(v_{|2} - v_{|4}\right) = 0. \tag{9.2}$$

Now if we divide this equality by $-\rho\Delta x\Delta y$, express the velocity as a continuous function of x and y and take the limit when the volume tends to zero, we get:

$$\lim_{\Delta x,\Delta y\longrightarrow 0}\frac{u(x+\Delta x,y)-u(x,y)}{\Delta x} + \frac{v(x,y+\Delta y)-v(x,y)}{\Delta y} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{9.3}$$

Finally, the conservation of mass property leads to the divergence-free condition:

$$\nabla.V = 0 \tag{9.4}$$

9.1.2 The Conservation of momentum property

Now, we consider the conservation of momentum in the control volume. Let us start with the u-momentum: *the rate of change of u-momentum within the control volume is equal to the rate at which the u-momentum enters the control volume associated with the forces acting on it in the x direction.*

The momentum in the x direction is $\Delta x \Delta y \rho u$, so that the rate of change of u-momentum is $\Delta x \Delta y \rho \frac{\partial u}{\partial t}$. Now, the rate at which the u-momentum enters each boundary is:

- boundary 1: $\rho u_{|1} \left(+u_{|1} \Delta y\right)$,
- boundary 2: $\rho u_{|2} \left(-v_{|2} \Delta x\right)$,
- boundary 3: $\rho u_{|3} \left(-u_{|3} \Delta y\right)$,
- boundary 4: $\rho u_{|4} \left(+v_{|4} \Delta x\right)$.

Then, we evaluate the different forces applied to the control volume. Let us start with the pressure forces applied to the boundaries of the control volume in the x direction:

- on boundary 1: $+P_{|1} \Delta y$,
- on boundary 3: $-P_{|3} \Delta y$.

The viscous stress τ applies a force to the control volume too. τ_{xx} is the normal stress in the x direction. τ_{yy} is the normal stress in the y direction. τ_{xy} is the shear stress in the y direction, while τ_{yx} is the shear stress in the x direction. For Newtonian fluids, there is a linear relationship between stress and strain rate that Stokes described for multidimensional flows:

- $\tau_{xx} = \frac{2\mu}{3}\left[2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right]$,
- $\tau_{yy} = \frac{2\mu}{3}\left[-\frac{\partial u}{\partial x} + 2\frac{\partial v}{\partial y}\right]$,
- $\tau_{xy} = \tau_{yx} = \mu\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right]$,

where μ is the viscosity coefficient of the fluid. Thus, the viscous force applied to each boundary of the control volume is:

- boundary 1: $-\tau_{xx}\Delta y$ (normal viscous force),
- boundary 2: $-\tau_{yx}\Delta x$ (tangential viscous force),
- boundary 3: $+\tau_{xx}\Delta y$ (normal viscous force),
- boundary 4: $+\tau_{yx}\Delta x$ (tangential viscous force).

The body force term is $\Delta x \Delta y F$, with $F = [F_x, F_y]^T$.

If we sum all the terms contributing to the u-momentum and divide by $\Delta x \Delta y$, we get:

$$\rho\frac{\partial u}{\partial t} + \rho\frac{u_{|3}^2 - u_{|1}^2}{\Delta x} + \rho\frac{u_{|4}v_{|4} - u_{|2}u_{|2}}{\Delta y} + \frac{P_{|3} - P_{|1}}{\Delta x} = \tag{9.5}$$

$$\frac{\tau_{xx|3} - \tau_{xx|1}}{\Delta x} + \frac{\tau_{yx|4} - \tau_{yx|2}}{\Delta y} + F_x.$$

By taking the limit when $\Delta x, \Delta y \longrightarrow 0$, we obtain:

$$\rho\frac{\partial u}{\partial t} + \rho\frac{\partial (u^2)}{\partial x} + \rho\frac{\partial (uv)}{\partial y} + \frac{\partial P}{\partial x} = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + F_x. \tag{9.6}$$

In the left-hand side:

$$\rho\frac{\partial (u^2)}{\partial x} + \rho\frac{\partial (uv)}{\partial y} = \rho\left[u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right], \tag{9.7}$$

using the mass conservation divergence-free property (9.4) of the fluid. For the right-hand side, in a similar way:

$$\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} = \mu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right]. \tag{9.8}$$

Finally, (9.6), (9.7), (9.8) lead to the u-momentum equation:

$$\rho \left[\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \right] + \frac{\partial P}{\partial x} = \mu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right] + F_x, \tag{9.9}$$

and the same process gives us the v-momentum equation:

$$\rho \left[\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \right] + \frac{\partial P}{\partial y} = \mu \left[\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right] + F_y. \tag{9.10}$$

By putting (9.9), (9.10) and (9.4) together, we obtain the incompressible NS equations:

$$\rho \left[\frac{\partial V}{\partial t} + (V.\nabla)\, V \right] = \mu \Delta V - \nabla P + F \tag{9.11}$$

$$\nabla . V = 0 \tag{9.12}$$

Bibliography

[1] P. Z. A. Quarteroni, A. Veneziani. Mathematical and numerical modelling of solute dynamics in blood flow and arterial walls. *SIAM J. Numer. Anal.*, 39-2:1488–1511, 2002.

[2] L. G. A. Yakhot and N. Nikitin. Modeling rough stenoses by an immersed-boundary method. *Journal of Biomechanics*, 38,5:1115–1127, 2005.

[3] A. S.-N. A.L.F. Lima E Silva and J. Damasceno. Numerical simulation of two-dimensional flows over a circular cylinder using the immersed boundary method. *Journal of Comput. Phys.*, 189,2:351–370, 2003.

[4] F. Baaijens. A fictitious domain/mortar element method for fluid structure interaction. *Int. J. Numer. Meth. Fluids*, 35:743761, 2001.

[5] E. Balaras. Modeling complex boundaries using an external force field on fixed cartesian grids in large-eddy simulations. *Computers and Fluids*, 33,3:375–404, 2004.

[6] J. Beale and A. Madja. High order accurate vortex methods with explicit velocity kernels. *Journal of Comput. Phys.*, 58,2:188–208, 1985.

[7] K. Bernert. τ-extrapolation - theoretical foundation, numerical experiment, and application to navier-stokes equations. *SIAM Journal on Scientific Computing*, 18-2:460–478, 1997.

[8] R. Beyer. A computational model of the cochlea using the immersed boundary method. *Journal of Computational Physics*, 98:145–162, 1992.

[9] C.H.Bruneau. Boundary conditions on artificial frontiers for incompressible and compressible navier stokes equations. *M2AN*, 34-2:303–314, 2000.

[10] A. Chorin. The numerical solution of the Navier-Stokes equations for an incompressible fluid. *Bull. Amer. Math. Soc.*, 73:928, 1967.

[11] A. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745–762, 1968.

[12] R. Cortez. An impulse-based approximation of fluid motion due to boundary forces. *Journal of Comput. Phys.*, 123,2:341–353, 1996.

[13] R. Cortez and M. Minion. The blob projection method for immersed boundary problems. *Journal of Comput. Phys.*, 161:428–453, 2000.

[14] R. Cortez and D. A. Varela. The dynamics of an elastic membrane using the impulse method. *Journal of Comput. Phys.*, 138,1:224–247, 1997.

[15] A. A. C.W. Hirt and J. Cook. An arbitrary lagrangianeulerian computing method for all flow speeds. *Journal of Comput. Phys.*, 3:227253, 1974.

[16] L. G. D. Boffi and L. Heltai. A finite element approach for the immersed boundary method. *Progress in Engineering Computational Technology, B.H.V. Topping and C.A. Mota Soares Eds., Saxe-Coburg Publications, Stirling, Scotland, Chapt.12*, pages 271–298, 2004.

[17] L. G. D. Boffi and L. Heltai. Numerical stability of the finite element immersed boundary method. *submitted*, 2005.

[18] L. G. D. Boffi and L. Heltai. Stability results and algorithmic strategies for the finite element approach to the immersed boundary method. *submitted*, 2005.

[19] R. H. D. Goldstein and L. Sirovich. Modeling a no-slip flow boundary with an external force field. *Journal of Comput. Phys.*, 105,2:354–366, 1993.

[20] R. H. D. Golstein and L. Sirovich. Direct numerical simulation of turbulent flow over a modeled riblet-covered surface. *Journal of Fluid Mech.*, 302:333–376, 1995.

[21] R. C. David L. Brown and M. L. Minion. Accurate projection methods for the incompressible navierstokes equations. *Journal of Computational Physics*, 168:464499, 2001.

[22] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse lu factorization. *SIAM J. Matrix Anal. Apll.*, 18-1:140–158, 1997.

[23] D.Metaxas. *Physics-Based Deformable Models: Application to Computer Vision, Graphics and Medical Imaging, Kluwer-Academic Publishers*. 1996.

[24] D.M.McQueen and C. Peskin. Heart simulation by an immersed boundary method with formal second-order accuracy and reduced numerical viscosity. In *Proceedings of the International Conference on Theoretical and Applied Mechanics (ICTAM) 2000 (H.Aref and J.W.Phillips, eds.), Kluwer Academic Publishers*, 2000.

[25] P. O. E. A. Fadlun, R. Verzicco and J. Mohd-Yusof. Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *Journal of Comput. Phys.*, 161,1:35–60, 2000.

[26] E.Arquis and J. Caltagirone. Sur les conditions hydrodynamiques au voisinage d'une interface milieu fluide-milieux poreux: Application a la convection naturelle. *CRAS, Paris II*, 299:1–4, 1984.

[27] C. Eggleton and A. Popel. Large deformation of red blood cell ghosts in a simple shear flow. *Physics of Fluids*, 10-8:1834–1845, 1998.

[28] S. C. Eisenstat and H. F. Walker. Choosing the forcing terms in an inexact Newton method. *SISC*, 17,1:16–32, 1996.

[29] M. F. Dupros and W. Fitzgibbon. A filtering technique for system of reaction-diffusion equations. *To appear in International Journal for Numerical Methods in Fluids*, 2006.

[30] L. Fauci and C. Peskin. A computational model of aquatic animal locomotion. *Journal of Computational Physics*, 77:85–108, 1988.

[31] R. P. Fedkiw and X.-D. Liu. The ghost fluid method for viscous flows. In *Progress in Numerical Solutions of Partial Differential Equations, Arachon, France, edited by M. Hafez*, 1998.

[32] A. Folgelson. A mathematical model and numerical method for studying platelet adhesion and aggregation during blood clotting. *Journal of Computational Physics*, 56:111–134, 1984.

[33] M. Forum. *MPI: A Message-Passing Interface Standard, Document for a Standard Message-Passing Interface, University of Tennessee*. 1994.

[34] F.Pacull and M.Garbey. The multigrid/τ-extrapolation technique applied to the immersed boundary method. In *Proceedings of the Sixteenth International Domain Decomposition Conference, New York University January 12-15th 2005*, 2005.

[35] M. Francois and W. Shyy. Computations of drop dynamics with the immersed boundary method, part 1: Numerical algorithm and buoyancy-induced effect. *Numer. Heat Transfer*, B,44:101–118, 2003.

[36] M. Francois and W. Shyy. Computations of drop dynamics with the immersed boundary method, part 2: Drop impact and heat transfer. *Numer. Heat Transfer*, B,44:119–143, 2003.

[37] M. Garbey, Y. Kuznetsov, and Y. Vassilevski. A parallel schwarz method for a convection-diffusion problem. *SIAM J. Sci. Comput.*, 22-3:891–916, 2000.

[38] M. Garbey and Y. Vassilevski. A parallel solver for unsteady incompressible 3d navier-stokes equations. *Parallel Computing*, 27-4:363–389, 2001.

[39] A. Gilmanov and F. Sotiropoulos. A hybrid cartesian/immersed boundary method for simulating flows with 3d,geometrically complex, moving bodies. *Journal of Comp. Phys.*, 207:457492, 2005.

[40] R. Glowinski. A fictitious domain approach to the direct numerical simulation of incompressible flow past moving rigid bodies: Application to particulate flow. *JCP*, 162:363–426, 2001.

[41] R. Glowinski. Numerical analysis for fluids,part 3. finite element methods for incompressible viscous flow. *Handbook of numerical analysis, Vol.IX, edited by P.G.Ciarlet and J.L.Lions, North-Holland,Amsterdam*, 2003.

[42] D. Gottlieb and C.W.Shu. On the gibbs phenomenon and its resolution. *SIAM Review*, 39,4:644–668, 1997.

[43] D. Gottlieb and C.-W. Shu. On the Gibbs phenomenon and its resolution. *SIAM Rev., Vol. 39, No. 4, pp. 644-668*, 1997.

[44] B. E. Griffith and C. S. Peskin. On the order of accuracy of the immersed boundary method: Higher order convergence rates for sufficiently smooth problems. *Journal of Computational Physics*, 208:75–105, 2005.

[45] P. R. H. S. Udaykumar, R. Mittal and A. Khanna. A sharp interface cartesian grid method for simulating flows with complex moving boundaries. *Journal of Comp. Phys.*, 174,1:345–380, 2001.

[46] R. M. H. S. Udaykumar and W. Shyy. Computation of solid-liquid phase fronts in the sharp interface limit on fixed grids. *Journal of Comput. Phys.*, 153,2:535–574, 1999.

[47] W. S. H. S. Udaykumar and M. M. Rao. Elafint: A mixed eulerian-lagrangian method for fluid flows with complex and moving boundaries. *International Journal for Numerical Methods in Fluids*, 22:691–712, 1996.

[48] F. H. Harlow. Pic method for fluid dynamics calculations. *Technical Report, Los Alamos Scientific Lab.*, 1961.

[49] F. H. Harlow and E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluids with free surface. *Phys. Fluids*, 8:2182–2189, 1965.

[50] M. Z. H.H. Hu, N.A. Patankar. Direct numerical simulations of fluid-solid systems using the arbitrary lagrangian-eulerian technique. *Journal of Comp. Phys.*, 169,2:427462, 2001.

[51] C. Hirt and B. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of Comput. Phys.*, 39:201–225, 1981.

[52] C. W. Hirt. Heuristic stability theory for finite-difference equation. *J. Comput. Phys.*, 2:339–355, 1968.

[53] G. Iaccarino and R. Verzicco. Immersed boundary technique for turbulent flow simulations. *Appl Mec Rev*, 56,3:331–347, 2003.

[54] D. K. J. Kim and H. Choi. An immersed-boundary finite-volume method for simulations of flow in complex geometries. *Journal of Comput. Phys.*, 171,1:132–150, 2001.

[55] J.A.Sethian. *Level Set Methods and Fast Marching Methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science, Cambridge Monographs on Computational Mathematics, P.G.Ciarlet, A.Iserles, R.V.Kohn and M.H.Wright editors.* 1999.

[56] J.Kepner and S. Ahalt. MatlabMpi. *Journal of Parallel and Distributed Computing, vol.64, Issue 8,August 2004, pp.997-1005*, 2004.

[57] J.Mohd-Yusof. Combined immersed boundaries / b-splines methods for simulations of flows in complex geometries. In *Ctr annual research briefs, Stanford University, NASA Ames*, 1997.

[58] J.R.Womersley. Method for the calculation of velocity, rate of flow and viscous drag in arteries when the pressure gradient is known. *J.Physiol.*, 127:553–563, 1955.

[59] J.Waldén. On the approximation of singular source terms in differential equations. *Numer. Methods Partial Differential Equations*, 15:503–520, 1999.

[60] D. Kim and H. Choi. Immersed boundary method for flow around an arbitrarily moving body. *Journal of Comput. Phys.*, Article in Press,Corrected Proof, 2005.

[61] Y. Kim and C. Peskin. 2-d parachute simulation by the immersed boundary method. *Submitted.*

[62] Y. Kim and C. Peskin. Penalty immersed boundary method for an elastic boundary with mass. *Preprint*, 2005.

[63] H. Köstler and U. Rüde. Accurate techniques for computing singular solutions of elliptic problems. *Technical report*, 2004.

[64] X. W. L. Zhang, A. Gerstenberger and W. K. Liu. Immersed finite element method. *Comput.Methods Appl. Mech. Engrg.*, 193:2051–2067, 2004.

[65] M. Lai and C. Peskin. An immersed boundary method with formal second-order accuracy and reduced numerical viscosity. *Journal of Computational Physics*, 160:705–719, 2000.

[66] L. Lee. *Immersed Interface Methods for Incompressible Flow with Moving Interfaces*. PhD thesis, University of Washington, 2002.

[67] L. Lee and R. LeVeque. Immersed interface methods for incompressible navier-stokes equations. *SIAM Journal of Sci. Comput.*, 25:832–856, 2003.

[68] R. P. M. Fortin and R. Temam. Calcul des écoulements d'un fluide visqueux incompressible. *J. Mec*, 10:357–390, 1971.

[69] D. A. McDonald. *Blood Flow in Arteries, Edward Arnold, third edition*. 1990.

[70] D. McQueen and C. Peskin. Shared-memory parallel vector implementation of the immersed boundary method for the computation of blood flow in the beating mammalian heart. *Journal of Supercomputing*, 11-3:213–236, 1997.

[71] M.Garbey and D. Dervout. On some Aitken like acceleration of the Schwarz method. *Int. J. for Numerical Methods in Fluids, 40 (12), pp. 1493-1513*, 2002.

[72] M.Garbey and D. T. Dervout. On some aitken like acceleration of the schwarz method. *Int. J. for Numerical Methods in Fluids*, 40-12:2002, 1493-1513.

[73] M.Garbey and Yu.V.Vassilevski. A parallel solver for unsteady incompressible 3d navier-stokes equations. *J. Parallel Computing*, 27:363–389, 2001.

[74] B. M.Garbey and W.Shyy. Fast elliptic solver for incompressible navier stokes flow and heat transfer problems on the grid. *43rd Aerospace Sciences Meeting and Exhibit Conference, Reno January 2005*, AIAA-2005:1386, 2005.

[75] C. M.Garbey, B.O.Dia and R.TranSonTay. Heterogeneous domain decomposition for multi-scale problems. *43rd Aerospace Sciences Meeting and Exhibit Conference, Reno January 2005*, AIAA-24880, 2005.

[76] L. Miller and C. Peskin. When vortices stick: an aerodynamic transition in tiny insect flight. *The Journal of Experimental Biology*, 207:3073–3088, 2004.

[77] L. Miller and C. Peskin. A computational fluid dynamics of clap and fling in the smallest insects. *The Journal of Experimental Biology*, 208:195–212, 2005.

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

[79] P. M.O.Deville and E.H.Mund. High order methods for incompressible fluid flow. *Cambridge Monographs on Applied and Computational Mathematics*, 2002.

[80] M.Uhlmann. First experiments with the simulation of particulate flows. *Technical Report No.1020,CIEMAT,Madrid,Spain,ISSN 1135-9420*, 2003.

[81] M. M. R. T. R. J. N. Barberou, M.Garbey and D. Dervout. On the efficient meta-computing of linear and nonlinear elliptic problems. *Journal of Parallel and Distributed Computing- special issue on grid computing*, 63-5:564–577, 2003.

[82] W. Noh and P. Woodward. Slic (simple line inter-face method). In *Lecture Notes in Physics 59. A.I. Van de Vooren and P.J. Zandbergen, editors*, pages 330–340, 1976.

[83] C. P.Angot and P.Fabrie. A penalization method to take into account obstacles in viscous flows. *Numerische Mathematik*, 81:497–520, 1999.

[84] K. Perktold, M. Resch, and H. Florian. Pulsatile non-newtonian flow characteristics in a three dimensionnal human carotid bifurcation model. *ASME Journal of Biomechanical Engineering*, 113:464–475, 1991.

[85] M. Pernice and H. F. Walker. NITSOL: A Newton iterative solver for nonlinear systems. *SIAM J. on Scientific Computing, Vol. 19, No. 1, pp. 302-318*, 1998.

[86] C. Peskin. Flow patterns around heart valves: a numerical method. *Journal of Computational Physics*, 10:252, 1972.

[87] C. Peskin. Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, 25:220–252, 1977.

[88] C. S. Peskin. *Flow Patterns Around Heart Valves: A Digital Computer Method for Solving the Equations of Motion.* PhD thesis, Albert Einstein College of Medicine - Yeshiva University, 1972.

[89] C. S. Peskin. The immersed boundary method. *Acta Numerica (2002) pp. 1-39*, 2002.

[90] C. S. Peskin and D. McQueen. A general method for the computer simulation of biological systems interacting with fluids. In *SEB Symposium on Biological Fluid Dynamics , Leeds, England, July 5-8, 1994*, 1994.

[91] R. Peyret and T. D. Taylor. *Computational Methods for Fluid Flow.* Springer-Verlag, 1983.

[92] T. H. R. Glowinski, T.-W. Pan and D. Joseph. A distributed lagrange multiplier/fictitious domain method for particulate flows. *Int.J.Multiphase Flow*, 25:755794, 1999.

[93] T. H. D. J. R. Glowinski, T.-W. Pan and J. Periaux. A fictitious domain approach to the direct numerical numerical simulation of incompressible viscous

flows past moving rigid bodies: applications to particulate flow. *Journal of Comput. Phys.*, 169:363–426, 2001.

[94] B. M. R. P. Fedkiw, T. Aslam and S. Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of Comput. Phys.*, 152,2:457–492, 1999.

[95] T. A. R. P. Fedkiw and S. Xu. The ghost fluid method for deflagration and detonation discontinuities. *Journal of Comput. Phys.*, 154,2:393–427, 1999.

[96] J. Ramshaw and J. Trapp. A numerical technique for low speed homogeneous two phase flow with a sharp interface. *Journal of Comput. Phys.*, 21:438–453, 1976.

[97] A. M. Roma. *A Multilevel Self Adaptive Version of the Immersed Boundary Method*. PhD thesis, New York University, 1996.

[98] U. Rüde. On the accurate computation of singular solutions of Laplace's and Poisson's equation. In *Proceedings of the Third Copper Mountain Conference on Multigrid Methods*, 1987.

[99] Y. Saad and M. Schultz. Gmres: A generalized minimal residual method for solving non-symmetric linear systems. *SIAM J. Sci. Statist. Comput. 7 (1986) 856–869*, 1986.

[100] E. M. Saiki and S. Biringen. Numerical simulation of a cylinder in uniform flow: Application of a virtual boundary method. *Journal of Comp. Phys.*, 123,2:450–465, 1996.

[101] K. Schneider and M.Farge. Numerical simuation of the transient flow behavior in tube bundles using a volume penalization method. *Journal of Fluids and Structures*, 20:555–566, 2005.

[102] W. Shyy. Moving boundaries in micro-scale biofluid dynamics. *Appl. Mech. Rev.*, 54-5:405–453, 2001.

[103] S.Osher and N. Paragios. *Geometric Level Set Methods in Imaging, Vision and Graphics, Springer Verlag, ISBN 0387954880*. 2003.

[104] J. Stockie and S. Green. Simulating the motion of flexible pulp fibres using the immersed boundary method. *Journal of Computational Physics*, 147(1):147–165, 1998.

[105] J. M. Stockie. *Analysis and Computation of Immersed Boundaries, with Applications to Pulp Fibres*. PhD thesis, University of British Columbia, 1997.

[106] M. Sussman and E. Puckett. A coupled level set and volume of fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of Computational Physics*, 162:301–337, 2000.

[107] H. S. U. T. Ye, R. Mittal and W. Shyy. An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries. *Journal of Comput. Phys.*, 156,2:209–240, 1999.

[108] R. Temam. *Navier-Stokes Equations. Theory and Numerical Analysis, Studies in Mathematics and Its Applications, vol. 2*. North-Holland Publishing, Amsterdam, 1977.

[109] T. Tezduyar. Finite element methods for flow problems with moving boundaries and interfaces. *Arch. Comput. Methods Eng.*, 8:83–130, 2001.

[110] T.F.Chan and L.A.Vese. Active contours without edges. *IEE Transaction on Image Processing*, 10-2:266–277, 2001.

[111] A.-K. Tornberg and B. Engquist. Numerical approximation of singular source terms in differential equations. *Journal of Computational Physics 200 (2004) pp.462-488*, 2004.

[112] Y.-H. Tseng and J. H. Ferziger. A ghost-cell immersed boundary method for flow in complex geometry. *Journal of Comput. Phys.*, 192,2:593–623, 2003.

[113] M. Uhlmann. An immersed boundary method with direct forcing for the simulation of particulate flows. *Journal of Comp. Phys.*, 209,2:448–476, 2005.

[114] J. Viecelli. A method for including arbitrary external boundaries in the mac incompressible fluid computing technique. *Journal of Comput. Phys.*, 4,4:543–551, 1969.

[115] J. Viecelli. A computing method for incompressible flows bounded by moving walls. *Journal of Comput. Phys.*, 8,1:119–143, 1971.

[116] A. Vikhansky. A new modification of the immersed boundaries method for fluid solid flows: moderate reynolds numbers. *Journal of Computational Physics*, 191:328 339, 2003.

[117] W. H. W. Cao and R. Russel. A moving mesh method based on the geometric conservation law. *Journal of Sci. Comput.*, 24:118–142, 2003.

[118] M. R. W. Shyy, H.S. Udaykumar and R. Smith. *Computational Fluid Dynamics with moving boundaries.* Series in Computational and Physical Processes in Mechanics and Thermal Sciences. W.J.Minkowycz and E.M.Sparrow, Editors. Taylor and Francis, 1996.

[119] X. Wang and W. K. Liu. Extended immersed boundary method using fem and rkpm. *Comput.Methods Appl. Mech. Engrg.*, 193:13051321, 2004.

[120] W.F.Noh. Cel: a time-dependent, two-space-dimensional, coupled eulerianlagrange code. In *Methods in Computational Physics, Academic Press, New York*, pages 117–179, 1964.

[121] A. G. D. F. L. Z. W.K. Liu, Y. Liu and X. Wang. Immersed finite element method and applications to biological systems. In *International Center for Numerical Methods and Engineering (CIMNE), ISBN: 84-95999-49-8*, pages 233–248, 2004.

[122] X. Y. Xu, M. W. Collins, and C. J. H. Jones. Flow studies in canine aortas. *ASME J. of Biomech. Eng.*, 114-11:505–511, 1992.

[123] Z. Yu. A dlm/fd method for fluid/flexible-body interactions. *Journal of Computational Physics*, 207:1–27, 2005.

[124] L. Zhu and C. Peskin. Simulation of a flapping flexible filament in a flowing soap film by the immersed boundary method. *J. Comput. Phys.*, 179:452–468, 2002.