

**Numerical solution of the Poisson interface problem
using the Correction Function Method framework**

Philippe Blain

Department of Mathematics and Statistics

McGill University, Montréal

*A thesis submitted to McGill University
in partial fulfillment of the requirements of the degree of
Master of Science*

August 2018

© Philippe Blain 2018

Contents

List of Figures	v
List of Tables	vi
List of Algorithms	vi
Abstract	vii
Abrégé	viii
Acknowledgement	ix
Contribution of Authors	x
1. Introduction	1
2. Literature Review	4
3. Methodology	7
3.1. An Elliptic Interface Problem	7
3.1.1. Problem Definition	7
3.1.2. Weak Form and Well-Posedness	9
3.1.3. Representation Formula for the Poisson Problem	13
3.2. Numerical Method	13
3.2.1. The Correction Function Method Framework	14
3.2.2. The Compact 4 th -Order Finite Difference Scheme	18
3.2.2.1. Higher Order Accuracy for Harmonic Functions	20
3.2.2.2. Computation of the Solution Gradient	21
3.2.3. Interface Description	22
3.2.4. Local Solver for the Correction Function	24
3.2.4.1. Minimization	26
3.2.4.2. Bicubic Interpolation	27
3.2.4.2.1. Reduced Cell-Based Bicubic Interpolation	29
3.2.4.3. Domain Definition	32
3.2.4.4. Parameterization	33
3.2.4.5. Mapping to the Reference Domain	37
3.2.4.5.1. Domain Integral	38
3.2.4.5.2. Interface Integrals	40
3.2.4.6. Scaling Coefficients	44

3.2.4.7. New Correction Function Solver	45
3.2.5. Fast Poisson Solver	46
3.2.5.1. Fast Poisson Solver in 1D	47
3.2.5.2. Fast Poisson Solver for the 5-Point Stencil	49
3.2.5.2.1. The Kronecker Product and Some Properties	49
3.2.5.3. Fast Poisson Solver for the 9-Point Stencil	52
3.3. Code Implementation	52
3.3.1. MATLAB Code	53
3.3.1.1. Main Solver : CFM2D and CFM2DOrder4Compact	53
3.3.1.2. Domain Discretization : Domain2D, Mesh2D and LevelSet	54
3.3.1.3. Correction Function Solver	55
3.3.1.4. Visualization and Convergence	55
3.3.1.5. Linear Solver	56
3.3.1.6. Parameters and Problem Description	56
3.3.2. C++ Code	56
3.4. Towards a Correction Function Method for the Navier-Stokes Equations	59
3.4.1. Problem Description	59
3.4.2. Numerical Method	61
4. Results	63
4.1. Representation Formula Approximation	63
4.2. Example 1	66
4.2.1. Problem Definition	66
4.2.2. Numerical Solution	66
4.2.3. Convergence	68
4.3. Example 2	70
4.3.1. Problem Definition	70
4.3.2. Numerical Solution	70
4.3.3. Convergence	72
4.4. Example 3	74
4.4.1. Problem Definition	74
4.4.2. Numerical Solution	75
4.4.3. Convergence	75
4.5. Example 4	78
4.5.1. Problem Definition	78
4.5.2. Numerical Solution	78
4.5.3. Convergence	79
5. Discussion	82
5.1. General Remarks	82

5.2. Domain of Definition of the Bicubic Interpolants	83
5.3. CFM-Based Navier-Stokes Solver	85
6. Conclusion	87
Appendix A. Linear System for the Correction Function Coefficients	89
Bibliography	97

List of Figures

3.1. Two-dimensional domain	8
3.2. One-dimensional domain	14
3.3. Close-up of the 1D domain near the interface point Γ	15
3.4. Domain Ω_Γ for the correction function problem	17
3.5. Nearest neighbours on a 2D grid	19
3.6. Linear approximation to the level set in the normal direction d	23
3.7. The integration domain for the correction function minimization problem	25
3.8. Domain for bicubic interpolation	28
3.9. Nodes needed for deriving the reduced bicubic interpolants	30
3.10. Local integration domain and transformation boxes	33
3.11. Transformation used to parameterize the interface integrals	34
3.12. Transformation used to map the integration domain to the unit square	38
3.13. Class diagram for the MATLAB version of the code	54
3.14. Class diagram for the C++ version of the code	57
3.15. MAC grid used for Navier-Stokes	62
4.1. Domain for the validation of the representation formula approach	64
4.2. Test of the representation formula approach	65
4.3. Numerical solution of example 1	67
4.4. Error in the computed solution of example 1	67
4.5. Convergence of the solution for example 1	68
4.6. Convergence of the correction function for example 1	69
4.7. Convergence of the solution gradient for example 1	69
4.8. Numerical solution of example 2	71
4.9. Error in the computed solution of example 2	71
4.10. Error in the computed correction function for example 2	72
4.11. Convergence of the solution for example 2	73
4.12. Convergence of the correction function for example 2	73
4.13. Convergence of the solution gradient for example 2	74
4.14. Numerical solution of example 3	75
4.15. Error in the computed solution of example 3	76
4.16. Convergence of the solution for example 3	76

4.17. Convergence of the correction function for example 3	77
4.18. Convergence of the solution gradient for example 3	77
4.19. Numerical solution of example 4	79
4.20. Convergence of the solution for example 4	80
4.21. Convergence of the correction function for example 4	80
4.22. Convergence of the solution gradient for example 4	81
5.1. Domain formerly used for the correction function interpolants	83
5.2. Magnitude of the entries of the minimization matrix	84

List of Tables

3.1. Single indices for the bicubics interpolants	32
3.2. Four UML relationships	53

List of Algorithms

3.1. Projection of a point x_0 on the interface	24
3.2. Main idea of the Fast Poisson solver	47
3.3. Projection method for Navier-Stokes with pressure discontinuity	61
4.1. Manufactured solution	63

Abstract

In this work we study the Poisson interface problem and a numerical method for its solution, the Correction Function Method. The Poisson interface problem complements the classical Poisson problem, a partial differential equation given in terms of the Laplacian operator, with jump conditions on the solution and its normal derivative along a co-dimension one interface inside the domain. Interface problems arise whenever materials with different properties come into contact, and thus have important applications in many fields of physics and engineering. The Correction Function Method (CFM) is a general framework to solve the Poisson interface problem in the wider context of finite difference methods. In essence, it modifies the finite difference stencils for nodes near the interface in order to take into account the jump conditions. However, these modifications are incorporated, using correction terms, to the right-hand side of the linear system deriving from the finite difference approximation of the partial differential equation, leaving the coefficient matrix identical to the one resulting from the regular Poisson problem. This is essential as many efficient linear solvers, termed “Fast Poisson solvers”, are specifically designed to solve this system; by keeping the same matrix we can use them with no modification. In order to account for the multiple possible geometric configurations of the interface with respect to the finite difference stencil, we compute the correction terms locally using a functional minimization approach. Our specific implementation of the CFM is shown to be fourth-order accurate. We also present preliminary work towards a CFM-based Navier-Stokes solver for higher-order numerical simulation of multi-phase flow.

Abrégé

Dans ce mémoire, on étudie le problème de Poisson avec interface ainsi qu'une méthode numérique de résolution de ce problème, la méthode de la fonction de correction (*Correction Function Method* ou CFM). Le problème de Poisson avec interface est basé sur le problème de Poisson classique, une équation aux dérivées partielles faisant intervenir l'opérateur laplacien. Il y ajoute des conditions de saut sur la solution ainsi que sur sa dérivée normale le long d'une interface de co-dimension un à l'intérieur du domaine de définition de l'équation. Les problèmes d'interface apparaissent lorsque deux matériaux ayant des propriétés différentes sont mis en contact, et ont donc des applications importantes dans plusieurs domaines en physique et en génie. La méthode de la fonction de correction fournit un cadre général pour la résolution numérique du problème de Poisson avec interface dans le contexte des méthodes de différences finies. Essentiellement, la méthode modifie les formules de différence finies pour les noeuds de discrétisation situées à proximité de l'interface, de façon à incorporer les conditions de saut. Toutefois, ces modifications sont apportées, à l'aide de termes de correction, au terme de droite du système linéaire résultant de la discrétisation par différences finies ; la matrice de coefficients reste la même que pour le problème de Poisson standard. Ceci est essentiel puisque de nombreux solveurs linéaires efficaces, appelés « *Fast Poisson solvers* (solveurs de Poisson rapides) », sont spécifiquement conçus pour résoudre ce système. Le fait de garder la même matrice permet donc de les utiliser sans modifications. Afin de prendre en compte les multiples configurations géométriques de l'interface par rapport à la grille de discrétisation, les termes de correction sont calculés localement en minimisant une fonctionnelle. On montre que notre implémentation spécifique de la CFM converge à l'ordre quatre. On présente aussi des résultats préliminaires en vue de la conception d'un solveur des équations de Navier-Stokes basé sur la CFM, pour la simulation numérique à ordre élevé des écoulements multiphasés.

Acknowledgement

I want to thank my supervisor, Prof. Jean-Christophe Nave, for his support, his deep insights and for the many engaging conversations we had, be it about mathematics or not. I also thank him for his monetary support. Some results presented in this work use parts of a numerical solver for the Navier-Stokes equations developed by Prof. Nave. I also want to thank Prof. Gantumur Tsogtgerel for his monetary support. I want to thank the Department of Mathematics and Statistics and my supervisor Prof. Nave for giving me the opportunity to take part in the Calcul Québec summer school in high performance computing. I thank Andy Wan for his help in getting started on some theoretical results presented in this work. I thank Alexandre Noll Marques for clarifying some details of the numerical method and sharing with me his code. It helped me a lot to be able to compare my code with his. I am grateful to the *Fonds de Recherche du Québec – Nature et Technologie* for granting me a *Bourse de maîtrise en recherche* for the duration of my degree. Finally, I want to thank my parents, for their unwavering support during the course of my studies, as well as my girlfriend for sharing my life for the past six years.

Contribution of Authors

This thesis was written entirely by Philippe Blain.

1. Introduction

Many problems arising in the physical sciences and in engineering model the interaction between materials with different physical properties, such as viscosity in fluid dynamics or permittivity in electromagnetics. Other problems involve different interacting states, such as a solid interacting with a liquid. These problems are known as interface problems and are mathematically modeled and studied using partial differential equations (PDEs). Often, in those problems, both the data and the solution are discontinuous because of the different physical values of the material properties. Specifically, at the interface between the different materials, or between the different phases, the functions involved in the partial differential equations present jump discontinuities. However, these functions are usually smooth in the individual regions of the problem domain corresponding to distinct materials.

This thesis investigates a problem of the kind described above, namely the Poisson interface problem. This problem complements the classical Poisson problem, a PDE given in terms of the Laplacian differential operator, with jump conditions on the solution and its normal derivative along a co-dimension 1 interface inside the domain. The Poisson interface problem has numerous applications but our driving motivation in this work is the fact that it appears in the modeling of multiphase flow. Multiphase flow refers to the study of the combined movement of more than one liquid or gas, such as water bubbles in oil or water droplets in air, usually modeled through the Navier-Stokes equations. Specifically, at the interface between two fluids, the pressure field satisfies the same kind of jump conditions as the solution of the Poisson interface problem. Moreover, when discretized in time, the solution algorithm for the Navier-Stokes equations directly involves the solution of a Poisson interface problem at each time step.

Several numerical methods exist to solve Poisson interface problems. Most of these methods fall into the large classes of finite difference methods or finite element methods. In finite dif-

ference methods, the general idea is to modify the finite difference stencils in the region near the interface in order to incorporate the jump conditions. In this work, we use the Correction Function Method (CFM) [31], which is one such method. This method modifies the stencil in such a way that only the right-hand side of the linear system resulting from the finite difference discretization is changed ; the coefficient matrix is unchanged compared to the regular Poisson problem. This is an important feature of the method since many linear solvers are specifically designed to solve this linear system by judiciously exploiting the special structure of the matrix. These solvers are based on the Fast Fourier Transform and are thus computationally very efficient. The fact that we keep the same matrix as the regular Poisson problem means we can directly use them, without modification.

Another important feature of the Correction Function Method is its generality; the correction terms used to account for the interface jumps are found by solving a partial differential equation , which means that in theory they can be computed to arbitrary accuracy and used within very high-order Poisson discretization schemes. The specific implementation presented here is fourth-order accurate. The PDE used to determine the correction terms is solved in a local fashion by minimizing a functional through a polynomial approximation. This approach allows us to uniformly handle the numerous ways in which the interface can cut through the finite difference stencil. The CFM represents the interface through a level set function, which permits the use of geometric transformations to efficiently parameterize the integrals defined on the interface that appear in the functional to be minimized.

The solution of the Poisson interface problem arising in the time discretization of the Navier-Stokes equation is the main rationale for the Correction Function Method. Since the interface is described through a level set function, the finite difference grid can be generated without taking the interface into account : a uniform grid can be used. This is an important consideration with regard to computational efficiency because the generation of a non-uniform grid that conforms to an arbitrary interface — a so-called body-fitted grid — can be a very time consuming task. In time dependent problems, such as the Navier-Stokes equations, the interface evolves in time, so methods using a body-fitted grid must regenerate the grid at each time step. This greatly increases the cost of such methods. Hence, important efficiency gains can be made with methods which use a uniform grid, such as the CFM.

The goal of this project was to study the Poisson interface problem and implement in code

the Correction Function Method used for its numerical solution. We also present preliminary work towards a CFM-based Navier-Stokes solver for multiphase flow. In this thesis, every effort was made to explain all mathematical aspects of the method and bring this description as close to the code implementation as possible. This thesis thus serves as a comprehensive reference on the mathematics and numerics of the CFM.

The remainder of this thesis is structured as follows. In Chapter 2, we briefly review the relevant literature pertaining to the numerical solution of interface problems. In Chapter 3, we present the mathematics of the Poisson interface problem, the Correction Function Method used for its numerical solution and the implementation of this method in code. In Chapter 4, we examine the performance of our implementation by showing its convergence for 4 different problems. Finally, Chapter 5 contains a discussion of the method, possible extensions to other related problems as well as the rationale for certain implementation details.

2. Literature Review

In this chapter, we present a short review of the literature on the numerical solution of interface problems, concentrating on finite difference approaches. The four methods we examine are the Immersed Interface Method, the Immersed Boundary Method, the Ghost Fluid Method and the Correction Function Method, which all use the approach of non-body-fitted grids.

The Immersed Interface Method (IIM) was introduced by Li in his Ph.D. thesis [26] to solve general interface problems where the solution, the coefficients and the source term can have jump discontinuities along an interface. The IIM is a sharp interface method, in the sense that the jumps in the computed solution at the interface are sharp : no smearing of the solution occur when it crosses the interface [26], [24]. The method relies on modified finite difference stencils for grid nodes close to the interface. The new finite difference stencils are found by minimizing the local truncation error, which leads to a constrained quadratic optimization problem [25]. The original implementation of the method is second order accurate, but more recently it has been extended to fourth order accuracy [25]. The IIM has been applied to numerous problems such as moving boundary problems [50], wave equations with discontinuous coefficients [51] and has also been used in the context of finite elements [19].

The Immersed Boundary Method (IBM) was introduced by Peskin in [37] to simulate the interaction of blood flow with the cardiac valves. In his work, this problem is modeled using a coupled Eulerian/Lagrangian approach where the liquid (the blood) is discretized using an Eulerian grid and the cardiac muscle is discretized using Lagrangian markers [39]. The Eulerian and Lagrangian quantities are coupled through the use of a discrete approximation of the Dirac delta function, which models a surface force. In contrast to the Immersed Interface Method, the IBM is a smoothing method: discontinuities in the computed fields are smeared at the interface because of the use of the discrete delta function, which spans a few grid cells on each side of the interface [34]. The Immersed boundary method is theoretically second order

accurate but implementation issues often limit it to being first order [34]. Apart from cardiovascular modeling, it has been applied to other problems in the life sciences, to the Poisson interface problem itself [38] and also to front tracking methods for bubble dynamics [48].

The Ghost Fluid Method (GFM) was presented by Fedkiw and others in [17] to solve multi-phase compressible flow problems. The main idea of the method was to introduce ghost cells, where quantities are defined on the opposite side of the interface, in order to be able to differentiate smoothly across the interface. These ghost cells are used to accurately track discontinuities occurring at shocks [15]. The field values at the ghost cells are defined through extrapolation. The method was extended shortly after to multiphase incompressible flow [22], in which the Poisson interface problem appears as a subproblem. A GFM-based Poisson interface solver was designed in [27]. In this method, the jump conditions are used to modify the finite difference stencils for grid nodes close to the interface, similarly to the methods above. However, the treatment of the jump conditions is done in a dimension-by-dimension approach. The resulting method has sharp jumps but is only first order accurate. A convergence proof was presented in [28] for the Poisson interface problem as well as more general elliptic interface problems. The GFM was also applied to fluid-solid interactions in [16].

The Correction Function Method is a general framework to solve the Poisson interface problem and was introduced in [31]. Similarly to the IIM and the GFM methods above, it modifies the finite difference stencils near the interface to account for the jump conditions. Also similarly to these methods, the stencil corrections are incorporated to the right-hand side of the linear system in order to keep the system matrix identical to the regular Poisson problem, as mentioned in Chapter 1. The method is more general than the above methods since as mentioned above, the correction terms are computed by locally solving a partial differential equation at each required node. This means that in theory the CFM can be made arbitrarily accurate. A fourth order implementation is presented in [31], and the method was extended to handle discontinuous coefficients in [32]. The CFM was also recently applied to the wave equation with discontinuous coefficients and irregular boundaries in [1]. The local solver for the correction function was refined in [33], where a new parameterization for interface integrals, based on the level set information, was introduced. This is the method that is described and implemented in this thesis.

Although our focus is on finite difference methods, we briefly mention finite element meth-

ods, where there are two main approaches. The simplest approach, which was the first to be introduced, starting with the work of Babuska [2], is to use a body-fitted mesh that adapts to the shape of the interface. Since the interface is taken into account when meshing the domain, the domain mesh induces a co-dimension 1 mesh on the interface, which makes it easy to discretize forms defined as integrals over the interface.

The alternative approach is to mesh the domain without taking into account the interface, letting the interface freely cut through elements. In this non-body-fitted approach, elements which are traversed by the interface must be dealt with in a special way. One approach is to modify the basis functions to incorporate the jumps conditions, as in [12]. The non-body-fitted approach has advantages in time-dependent problems. Indeed, the creation of the mesh is often one of the most time-consuming tasks in a finite element simulation workflow. In a body-fitted approach, as mentioned above for finite difference methods, the mesh must be recreated at each time step since the interface usually evolves with the problem. The elimination of the need for remeshing is thus a way to make important efficiency gains.

3. Methodology

This chapter presents the theoretical, numerical and code implementation background underlying our results. In Section 3.1, we define the Poisson interface problem and present some theoretical results pertaining to it and to the classical Poisson problem. In Section 3.2, we describe in detail the numerical method used to solve the problem. Section 3.3 explains the design of our implementation of the method in code, in the programming languages MATLAB and C++. Finally, Section 3.4 presents the Navier-Stokes equations of fluid dynamics and how the Correction Function Method could be applied in this context.

3.1 An Elliptic Interface Problem

3.1.1 Problem Definition

The model problem examined in this project is the Poisson problem with given interface jumps on the solution and its normal derivative. It is defined by :

$$\Delta u = f, \quad x \in \Omega \tag{3.1}$$

$$[u] = a, \quad x \in \Gamma \tag{3.2}$$

$$[\partial_n u] = b, \quad x \in \Gamma \tag{3.3}$$

$$u = g, \quad x \in \partial\Omega \tag{3.4}$$

where $u : \Omega \rightarrow \mathbb{R}$ is the unknown function, $f : \Omega \rightarrow \mathbb{R}$ is the source data, $a, b : \Gamma \rightarrow \mathbb{R}$ are the given jump data on the interface, and $g : \partial\Omega \rightarrow \mathbb{R}$ is the Dirichlet data on the domain boundary. The domain Ω is split in 2 subdomains Ω^+ and Ω^- by the interface Γ , as illustrated in figure 3.1 below. Note that the Dirichlet boundary condition (3.4) is used here to complete the problem, but in practice, either Neumann, mixed or periodic boundary conditions could

3.1. An Elliptic Interface Problem

also be applied.

In general, the solution u and the source term f can be discontinuous along the interface, and so we define the restrictions w^+, w^- of a function $w : \Omega \rightarrow \mathbb{R}$ as :

$$w^+ := w|_{\Omega^+} \quad (3.5)$$

$$w^- := w|_{\Omega^-} \quad (3.6)$$

Usually u and f are piecewise continuous ; their restrictions are continuous over each subdomain. Using the restrictions of u and f , equation (3.1) can thus be separated in 2 distinct equations, one for each subdomain Ω^+ and Ω^- :

$$\Delta u^+ = f^+ \quad x \in \Omega^+ \quad (3.7)$$

$$\Delta u^- = f^- \quad x \in \Omega^- \quad (3.8)$$

The jump operator $[u]$ is defined as

$$[u] := (u^+ - u^-)|_{\Gamma} \quad (3.9)$$

and is to be understood in a limit sense. The notation $[\partial_n u]$ denotes the jump in the outward normal derivative to the interface, i.e.

$$[\partial_n u] := [\nabla u] \cdot n = \nabla(u^+ - u^-)|_{\Gamma} \cdot n \quad (3.10)$$

where n is the outward normal to the interface.

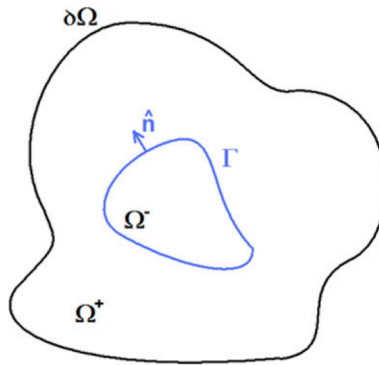


Figure 3.1. The domain Ω is split in two subdomains Ω^+ and Ω^- by a co-dimension 1 interface Γ . Image from [31].

We assume that the interface Γ is described by the zero level-set of a smooth function $\phi : \Omega \rightarrow \mathbb{R}$ such that

$$\Gamma = \{x \in \Omega \mid \phi(x) = 0\} \quad (3.11)$$

and the subdomains are then defined as

$$\begin{cases} \phi > 0, & x \in \Omega^+ \\ \phi \leq 0, & x \in \Omega^- \end{cases} \quad (3.12)$$

3.1.2 Weak Form and Well-Posedness

In this section, we derive a weak form for the problem (3.1)-(3.4) and show its well-posedness under some assumptions. This work is inspired by [28]. These assumptions are as follows :

$$f \in L^2(\Omega) \quad (3.13)$$

$$a \in H^{\frac{1}{2}}(\Gamma) \quad (3.14)$$

$$b \in L^2(\Gamma) \quad (3.15)$$

$$g \in H^{\frac{1}{2}}(\partial\Omega) \quad (3.16)$$

Note that in this section, keeping in line with the literature on the analysis of the classical Poisson problem, we add a minus sign in front of equations (3.7-3.8) (this simply corresponds to making the change to the source term $f \rightarrow -f$) :

$$-\Delta u^+ = f^+ \quad x \in \Omega^+ \quad (3.17)$$

$$-\Delta u^- = f^- \quad x \in \Omega^- \quad (3.18)$$

We begin by multiplying each one of equations (3.17-3.18) by a smooth function v defined on Ω such that $v|_{\partial\Omega} = 0$ and integrate on each subdomain :

$$-\int_{\Omega^+} \Delta u^+ v = \int_{\Omega^+} f^+ v \quad (3.19)$$

3.1. An Elliptic Interface Problem

$$-\int_{\Omega^-} \Delta u^- v = \int_{\Omega^-} f^- v \quad (3.20)$$

In Ω^- , we set

$$u^- = u_0^- - \bar{a} \quad (3.21)$$

where u_0^- and \bar{a} are defined on Ω^- such that $\gamma^-(\bar{a}) = a$, $\gamma^-(u_0^-) = 0$. Here $\gamma^-(\cdot)$ is the trace operator on Ω^- . The assumption (3.14) permits us to define \bar{a} in this way. For future purpose we also define $\gamma_\Gamma^-(\cdot) := \gamma^-(\cdot)$.

Similarly, in Ω^+ , we first extend the definition of g to the whole of $\partial\Omega^+ = \partial\Omega \cup \Gamma$ by defining g^* such that

$$g^* := \begin{cases} g & x \in \partial\Omega \\ 0 & x \in \Gamma \end{cases}$$

Since we have $g \in H^{\frac{1}{2}}(\Gamma)$, then we also have $g \in H^{\frac{1}{2}}(\partial\Omega^+)$ and thus we can define \bar{g} in Ω^+ such that $\gamma^+(\bar{g}) = g^*$, where $\gamma^+(\cdot)$ is the trace operator on Ω^+ . We then set

$$u^+ = u_0^+ + \bar{g} \quad (3.22)$$

where u_0^+ is defined on Ω^+ such that $\gamma^+(u_0^+) = 0$. We also define the partial trace $\gamma_\Gamma^+ : H^1(\Omega^+) \rightarrow L^p(\Gamma)$ by the restriction to Γ of the image of the trace operator γ^+ . This operator is well-defined and continuous since Γ is a closed subset of $\partial\Omega^+$.

Interpreting the restriction to the boundary in the definition (3.9) in the sense of traces, we have

$$\begin{aligned} [u] &:= (u^+ - u^-)|_\Gamma = u^+|_\Gamma - u^-|_\Gamma \\ &= \gamma_\Gamma^+(u^+) - \gamma_\Gamma^-(u^-) \\ &= \gamma_\Gamma^+(u_0^+ + \bar{g}) - \gamma_\Gamma^-(u_0^- - \bar{a}) \\ &= 0 - (0 - a) = a \end{aligned}$$

as desired.

Using the decompositions (3.21) and (3.22), we then integrate by parts in (3.20) and (3.19).

We get :

$$\begin{aligned}
\int_{\Omega^-} f^- v &= - \int_{\Omega^-} (\Delta u_0^- - \Delta \bar{a}) v \\
&= \int_{\Omega^-} \nabla u_0^- \cdot \nabla v - \int_{\partial\Omega^-} (\nabla u_0^- \cdot n) v - \int_{\Omega^-} \nabla \bar{a} \cdot \nabla v + \int_{\partial\Omega^-} (\nabla \bar{a} \cdot n) v \\
&= \int_{\Omega^-} \nabla u_0^- \cdot \nabla v - \int_{\Omega^-} \nabla \bar{a} \cdot \nabla v - \int_{\partial\Omega^-} (\nabla u^- \cdot n) v \\
&= \int_{\Omega^-} \nabla u_0^- \cdot \nabla v - \int_{\Omega^-} \nabla \bar{a} \cdot \nabla v - \int_{\Gamma} (\nabla u^- \cdot n) v
\end{aligned} \tag{3.23}$$

and

$$\begin{aligned}
\int_{\Omega^+} f^+ v &= - \int_{\Omega^+} (\Delta u_0^+ + \Delta \bar{g}) v \\
&= \int_{\Omega^+} \nabla u_0^+ \cdot \nabla v - \int_{\partial\Omega^+} (\nabla u_0^+ \cdot n^+) v + \int_{\Omega^+} \nabla \bar{g} \cdot \nabla v - \int_{\partial\Omega^+} (\nabla \bar{g} \cdot n^+) v \\
&= \int_{\Omega^+} \nabla u_0^+ \cdot \nabla v + \int_{\Omega^+} \nabla \bar{g} \cdot \nabla v - \int_{\partial\Omega^+} (\nabla u^+ \cdot n^+) v \\
&= \int_{\Omega^+} \nabla u_0^+ \cdot \nabla v + \int_{\Omega^+} \nabla \bar{g} \cdot \nabla v - \underbrace{\int_{\partial\Omega} (\nabla u^+ \cdot n^+) v}_{=0 \text{ since } v|_{\partial\Omega}=0} - \int_{\Gamma} (\nabla u^+ \cdot n^+) v
\end{aligned} \tag{3.24}$$

where n^+ is the outward normal to Ω^+ . Adding (3.24) and (3.23) we get

$$\begin{aligned}
\int_{\Omega^+} f^+ v + \int_{\Omega^-} f^- v &= \int_{\Omega^-} \nabla u_0^- \cdot \nabla v - \int_{\Omega^-} \nabla \bar{a} \cdot \nabla v - \int_{\Gamma} (\nabla u^- \cdot n) v \\
&\quad + \int_{\Omega^+} \nabla u_0^+ \cdot \nabla v + \int_{\Omega^+} \nabla \bar{g} \cdot \nabla v - \int_{\Gamma} (\nabla u^+ \cdot n^+) v
\end{aligned}$$

Now we note that $n^+ = -n$ on Γ so we can rewrite the above as

$$\int_{\Omega} f v = \int_{\Omega} \nabla u_0 \cdot \nabla v - \int_{\Omega^-} \nabla \bar{a} \cdot \nabla v + \int_{\Omega^+} \nabla \bar{g} \cdot \nabla v + \int_{\Gamma} (\nabla u^+ - \nabla u^-) \cdot n v \tag{3.25}$$

where we defined u_0 as

$$u_0 := \begin{cases} u_0^+ & x \in \Omega^+ \\ u_0^- & x \in \Omega^- \end{cases} \tag{3.26}$$

In (3.25) above, we recognize the jump in the normal derivative $[\partial_n u] := (\nabla u^+ - \nabla u^-) \cdot n$ and

3.1. An Elliptic Interface Problem

so we can write

$$\int_{\Omega} \nabla u_0 \cdot \nabla v = \int_{\Omega} f v + \int_{\Omega^-} \nabla \bar{a} \cdot \nabla v - \int_{\Omega^+} \nabla \bar{g} \cdot \nabla v - \int_{\Gamma} b v \quad (3.27)$$

We thus define the weak form of the problem as : find $u_0 \in H_0^1(\Omega)$ such that

$$a(u_0, v) = L(v) \quad v \in H_0^1(\Omega) \quad (3.28)$$

with

$$a(u_0, v) = \int_{\Omega} \nabla u_0 \cdot \nabla v \quad (3.29)$$

$$L(v) = \int_{\Omega} f v + \int_{\Omega^-} \nabla \bar{a} \cdot \nabla v - \int_{\Omega^+} \nabla \bar{g} \cdot \nabla v - \int_{\Gamma} b v \quad (3.30)$$

and the solution of the problem is

$$u = u_0 + \chi_{\Omega^+} \bar{g} - \chi_{\Omega^-} \bar{a} \quad (3.31)$$

where we have used the characteristic functions of Ω^+ and Ω^- .

It is known that the bilinear form (3.29), which is the standard bilinear form for the regular Poisson problem, is coercive and continuous over $H_0^1(\Omega)$ functions. Regarding the linear form $L(v)$, we can use the triangle inequality and Cauchy-Schwarz's inequality to show that

$$|L(v)| \leq c \left(\|f\|_{L^2(\Omega)} + \|\bar{a}\|_{H^1(\Omega)} + \|\bar{g}\|_{H^1(\Omega)} + \|b\|_{L^2(\Gamma)} \right) \|v\|_{H^1(\Omega)} \quad (3.32)$$

for some constant $c > 0$. The fact that the norms in (3.32) are all well-defined follows from the assumptions (3.13-3.16); the existence and uniqueness of a solution u_0 of (3.28) follows from the Lax-Milgram theorem. Uniqueness of u itself can also be shown, as well as continuous dependence on the data, as in [9] and [7], and thus the problem is well-posed in the sense of Hadamard.

3.1.3 Representation Formula for the Poisson Problem

The classical Poisson problem with Dirichlet boundary conditions, given by

$$-\Delta u = f, \quad x \in \Omega \quad (3.33)$$

$$u = g, \quad x \in \partial\Omega \quad (3.34)$$

is a relatively simple problem that has been extensively studied. Here we simply list a few results pertaining to this problem, taken from [14], that we use in section 3.2.4.7.

The Poisson problem has a fundamental solution $\Phi(x, y)$ given in 2 dimensions by

$$\Phi(x, y) := \frac{-1}{2\pi} \log \sqrt{x^2 + y^2} \quad (3.35)$$

For every point $x_0 \in \Omega$, every solution $u \in \mathcal{C}^2(\overline{\Omega})$ to the PDE (3.33) satisfies the following identity, which we refer to in the sequel as the representation formula for the Poisson problem :

$$\begin{aligned} u(x_0) = & - \int_{\Omega} \Phi(x - x_0) \Delta u(x) dx - \int_{\partial\Omega} \partial_n \Phi(x - x_0) u(x) ds(x) \\ & + \int_{\partial\Omega} \Phi(x - x_0) \partial_n u(x) ds(x) \end{aligned} \quad (3.36)$$

where the normal derivative of the fundamental solution is given by

$$\Phi(x, y) := \frac{-1}{2\pi(x^2 + y^2)} (xn_x + yn_y) \quad (3.37)$$

3.2 Numerical Method

In this section, we explain how the Poisson interface problem defined above is solved numerically using the framework of the Correction Function Method. Section 3.2.1 presents the Correction Function Method as it was introduced in [31]. Section 3.2.2 describes the 2D finite difference scheme used to discretize the problem. In Section 3.2.3, we focus on the discrete representation of the interface. The local solver used to correct the stencils for nodes near the interface is presented in Section 3.2.4. Finally, the Fast Poisson solver used for the solution of the linear system resulting from the finite difference discretization is derived in Section 3.2.5.

3.2.1 The Correction Function Method Framework

The Correction Function Method (CFM) was introduced in [31] as a general framework to solve the Poisson interface problem in 2 or 3 spatial dimensions, in the context of finite difference approximations. The basic idea of the method is more concisely explained in a 1 dimensional setting. In one dimension (1D), the Poisson interface problem reduces to the following boundary value problem :

$$u_{xx}^+ = f^+, \quad x \in \Omega^+ \quad (3.38)$$

$$u_{xx}^- = f^-, \quad x \in \Omega^- \quad (3.39)$$

$$[u] = a, \quad x \in \Gamma \quad (3.40)$$

$$[u_x] = b, \quad x \in \Gamma \quad (3.41)$$

$$u = g, \quad x \in \partial\Omega \quad (3.42)$$

Notice that in 1D the interface Γ is simply a point x_Γ in the domain interval $\Omega = (x_L, x_R)$, the boundary $\partial\Omega = \{x_L, x_R\}$ and the jump conditions a and b are constants. A typical solution and its domain are illustrated in figure 3.2 below.

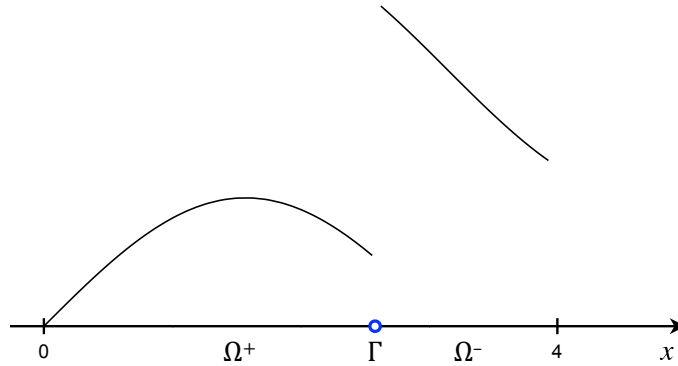


Figure 3.2. A one-dimensional domain Ω is split in two subdomains Ω^+ and Ω^- by an interface point Γ . Here $x_L = 0$ and $x_R = 4$.

The main idea of the correction function method is based on the following reasoning. Away from the interface, the ordinary differential equations (ODEs) for u^+ and u^- in (3.38-3.39) are discretized using standard finite-difference stencils. For example, a second-order scheme

would use the standard stencils

$$\frac{u_{i-1}^+ - 2u_i^+ + u_{i+1}^+}{h^2} = f_i^+ \quad (3.43)$$

$$\frac{u_{i-1}^- - 2u_i^- + u_{i+1}^-}{h^2} = f_i^- \quad (3.44)$$

However, near the interface, these stencils need to be modified somehow because they straddle the interface and hence do not give an appropriate approximation to the second derivative of either u^+ or u^- . Indeed, consider the situation illustrated in figure 3.3 below. Node k is the last node in subdomain Ω^+ , and the interface Γ is situated between nodes k and $k + 1$.

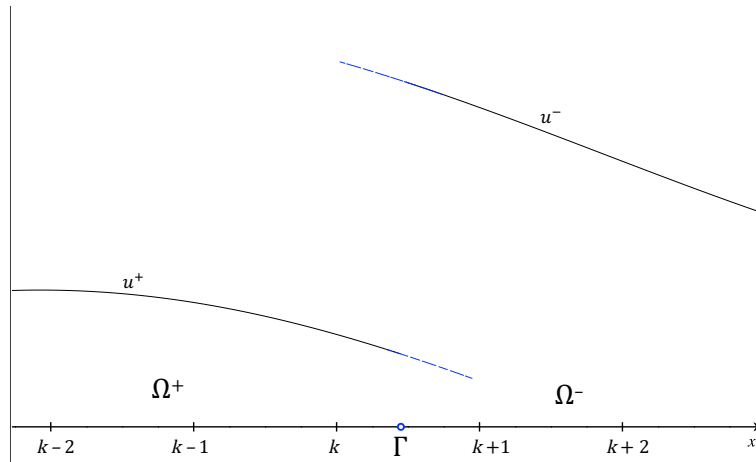


Figure 3.3. Close-up of the 1D domain near the interface point Γ .

Writing the second-order stencil at node k results in :

$$\frac{u_{k-1}^+ - 2u_k^+ + u_{k+1}^+}{h^2} = f_k^+ \quad (3.45)$$

In this equation, u_{k+1}^+ does not have a well-defined value, since at node $k + 1$ the solution is given by u^- and not by u^+ . Suppose, however, that we could extend u^+ in the subdomain Ω^- , up to a small distance from the interface, and in the same way extend u^- in Ω^+ (these extensions are represented in dashed blue in figure 3.3). Then we could write the value of u_{k+1}^+ as

$$u_{k+1}^+ = u_{k+1}^- + D_{k+1} \quad (3.46)$$

where D_{k+1} is a correction term (which would be negative in the situation of figure 3.3) quantifying the difference between u^+ and u^- at node $k + 1$. That correction term permits us to

write the stencil at node k using the real solution at node $k + 1, u^-$:

$$\frac{u_{k-1}^+ - 2u_k^+ + (u_{k+1}^- + D_{k+1})}{h^2} = f_k^+ \quad (3.47)$$

If this correction term can be computed in a way that is independent of the solution u , then it can be moved to the right-hand side (RHS) of the equation, as a modification to the source term :

$$\frac{u_{k-1}^+ - 2u_k^+ + u_{k+1}^-}{h^2} = f_k^+ - \frac{D_{k+1}}{h^2} \quad (3.48)$$

This is essential because many linear solvers have been specifically designed to efficiently solve the linear system resulting from the Poisson problem in 1, 2 or 3 dimensions. These solvers, termed “Fast Poisson solvers”, make use of the fast Fourier transform, as will be discussed in section 3.2.5. Their high performance relies of the structure of the finite difference matrix on the left-hand side of the linear system (in the 1D case above, a tridiagonal, symmetric matrix). If the correction terms are placed on the right-hand side, the structure of the coefficient matrix is unchanged compared to the standard Poisson problem and the Fast Poisson solvers can still be used, only with a modified right-hand side.

Returning to the general 2D or 3D case, we define a *correction function* $D : \Omega_\Gamma \rightarrow \mathbb{R}$ in a small domain Ω_Γ that forms a narrow band around the interface (see figure 3.4). This domain should be small but still large enough to cover all the nodes where corrections are needed ; this means all nodes that are part of a finite difference stencil that crosses the interface. The correction function is defined as

$$D := u^+ - u^-, \quad x \in \Omega_\Gamma \quad (3.49)$$

Using this definition, subtracting (3.7) and (3.8) and using the jump conditions (3.2-3.3), we discover that the correction function is the solution of the problem

$$\Delta D = f_D, \quad x \in \Omega_\Gamma \quad (3.50)$$

$$D = a, \quad x \in \Gamma \quad (3.51)$$

$$\partial_n D = b, \quad x \in \Gamma \quad (3.52)$$

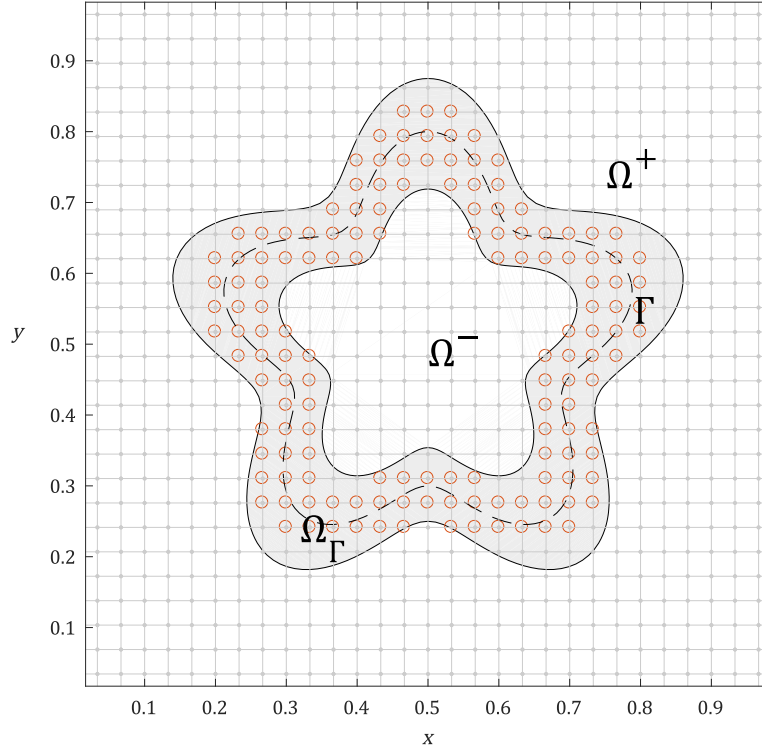


Figure 3.4. The domain Ω_Γ over which the correction function is sought, in grey, forms a band around the interface Γ . The grid nodes where corrections are required appear in orange.

where $f_D := f^+ - f^-$. This definition assumes that we can sensically extend f^+ past the interface inside Ω^- , and in the same way extend f^- inside Ω^+ , at least far enough so that f_D is defined over the whole Ω_Γ . Note that the width of the band domain Ω_Γ is of the order of the step size of the finite difference stencil.

The problem (3.50-3.52) is a Poisson problem with Cauchy data ; we impose both the function value and the normal derivative. In this specific case, the data is given on a curve Γ located inside Ω_Γ , i.e. the Cauchy data is not on the boundary of the domain. In [31], it is argued that even though such a problem is in general ill-posed, here it is well-posed because we are looking for a solution only at a small distance (of the order of h) from the curve Γ , and the numerical nature of the problem implies that there is a frequency cut-off in the problem data, i.e. a , b , f_D and Γ are only known at grid points. Once (3.50) is solved and the correction function is known at each node where the correction terms are needed, the linear system can be solved.

Since the correction function is defined as the solution of a partial differential equation (PDE), this framework gives a lot of flexibility ; any scheme can be designed to solve this PDE, so in theory it can be solved to any degree of accuracy. In practice, it is sufficient to solve the

correction function to the same accuracy as the finite difference scheme used, as is argued in [31]. The way we solve for the correction function is explained in section 3.2.4.

3.2.2 The Compact 4th-Order Finite Difference Scheme

The finite difference scheme used to solve the Poisson problem in 2D is the 9-point stencil for the Laplacian. This stencil is of order 4 for general functions. Its derivation is outlined below. We start with the standard, 2nd order 5-point stencil for the Laplacian Δ_{ij}^5 , which is simply the 1D, second-order standard stencil applied in both dimensions :

$$\Delta_{ij}^5 u := \hat{\partial}_{xx} u_{ij} + \hat{\partial}_{yy} u_{ij} \quad (3.53)$$

where $\hat{\partial}_{xx}$ and $\hat{\partial}_{yy}$ are the second-order centered stencils for the second derivative in the x and y directions :

$$\hat{\partial}_{xx} u_{ij} := \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h_x^2} \quad \hat{\partial}_{yy} u_{ij} := \frac{u_{i,j-1} - 2u_{ij} + u_{i,j+1}}{h_y^2} \quad (3.54)$$

Using Taylor series expansions centered at node (i, j) , it is easy to show that

$$\Delta_{ij}^5 u = \Delta u + \frac{h_x^2}{12} u_{xxxx} + \frac{h_y^2}{12} u_{yyyy} + \mathcal{O}(h^4) \quad (3.55)$$

Here and hereafter, the notation $\mathcal{O}(h^4)$ encompasses all error terms of order 4, i.e. terms of order h_x^4 , h_y^4 and $h_x^2 h_y^2$. In order to get a 4th order stencil, we simply approximate the leading order error term using finite differences of order 2. Hence we want to approximate u_{xxxx} and u_{yyyy} . However, we also wish to keep the stencil compact, i.e. only first and second nearest neighbours are to be allowed in the stencil (see figure 3.5). This way, the stencil does not have to be modified near the boundary.

These two facts at first seem contradictory, since the second order stencil for the 4th derivative needs 5 points and is thus not appropriate for our compact scheme :

$$u_{xxxx} = \frac{u_{i-2} - 4u_{i-1} + 6u_i - 4u_{i+1} + u_{i+2}}{h_x^4} + \mathcal{O}(h_x^2) \quad (3.56)$$

However, we can use the Poisson equation (3.1) to get a different expression for the fourth

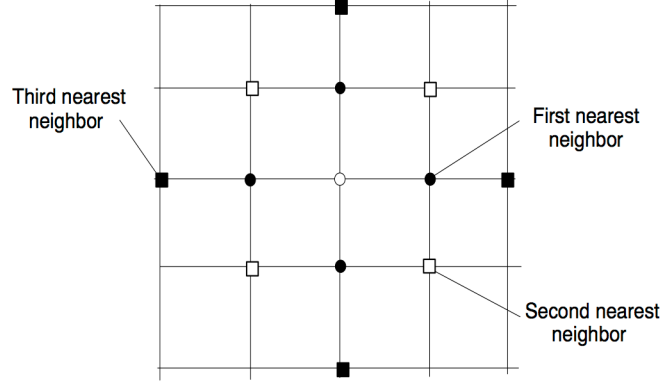


Figure 3.5. Nearest neighbours on a 2D grid. Image from [41].

derivatives. By differentiating twice with respect to x on each side of (3.1), we get

$$u_{xxxx} = f_{xx} - u_{xxyy} \quad (3.57)$$

Similarly, by differentiating twice with respect to y we obtain :

$$u_{yyyy} = f_{yy} - u_{xxyy} \quad (3.58)$$

The mixed derivative u_{xxyy} can be approximated compactly to second order :

$$\begin{aligned} u_{xxyy} &= \partial_{xx} u_{yy} \\ &= \hat{\partial}_{xx} u_{yy} + \mathcal{O}(h_x^2) \\ &= \hat{\partial}_{xx} \hat{\partial}_{yy} u + \mathcal{O}(h_x^2) + \mathcal{O}(h_y^2) \\ &= \frac{1}{h_x^2 h_y^2} [u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} \\ &\quad - 2(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) + 4u_{ij}] + \mathcal{O}(h^2) \end{aligned} \quad (3.59)$$

We can now use (3.57), (3.58) and (3.59) to rewrite (3.55) :

$$\Delta_{ij}^5 u - \frac{h_x^2}{12} (f_{xx} - \hat{\partial}_{xx} \hat{\partial}_{yy} u) - \frac{h_y^2}{12} (f_{yy} - \hat{\partial}_{xx} \hat{\partial}_{yy} u) = \Delta u + \mathcal{O}(h^4) \quad (3.60)$$

We then define

$$\Delta_{ij}^9 u := \hat{\partial}_{xx} u + \hat{\partial}_{yy} u + \left(\frac{h_x^2 + h_y^2}{12} \right) \hat{\partial}_{xx} \hat{\partial}_{yy} u \quad (3.61)$$

and using Δ_{ij}^9 , our 4th order scheme for the Poisson equation is

$$\Delta_{ij}^9 u = f_{ij} + \frac{1}{12} \left(h_x^2 (f_{xx})_{ij} + h_y^2 (f_{yy})_{ij} \right) \quad (3.62)$$

The scheme above assumes we have access to the second derivatives of the source terms f^+ , f^- in order to evaluate them at node (i, j) . If the source is only known at grid points, then its derivatives can be approximated using the standard second-order finite difference stencil without losing the fourth order accuracy of the scheme since they are multiplied by the step size, resulting in an error term of order 4. However, if the source term is discontinuous ($f^+ \neq f^-$), then for nodes near the interface we cannot use the standard stencil for the second derivative because it would cross the interface. To alleviate this issue, we can use the second order, off-centered stencils for the second derivative, which are 4 nodes wide :

$$f_{xx}^{\text{right}} = \frac{2f_i - 5f_{i+1} + 4f_{i+2} - f_{i+3}}{h^2} + \mathcal{O}(h_x^2) \quad (3.63)$$

$$f_{xx}^{\text{left}} = \frac{-f_{i-3} + 4f_{i-2} - 5f_{i-1} + 2f_i}{h^2} + \mathcal{O}(h_x^2) \quad (3.64)$$

3.2.2.1 Higher Order Accuracy for Harmonic Functions

It is interesting to note that the scheme (3.62) is sixth order for harmonic functions in the special case of a square grid ($h_x = h_y =: h$). In that case, the fourth order error term is

$$\frac{h^4}{360} \left(\partial_x^6 u + \partial_y^6 u + 5 (\partial_x^4 \partial_y^2 u + \partial_x^2 \partial_y^4 u) \right) \quad (3.65)$$

The derivatives appearing above can be rewritten as

$$\begin{aligned} \left(\partial_x^6 u + \partial_y^6 u + 5 (\partial_x^4 \partial_y^2 u + \partial_x^2 \partial_y^4 u) \right) &= \Delta^3 u + 2 (\partial_x^4 \partial_y^2 u + \partial_x^2 \partial_y^4 u) \\ &= \Delta^3 u + 2 \partial_{xxyy} (\Delta u) \\ &= 0 \end{aligned}$$

where the first and third power of the Laplacian appear, making the term zero for harmonic functions.

3.2.2.2 Computation of the Solution Gradient

For several applications of the Poisson interface problem, as we shall later see, the quantity of interest is not the solution u but rather its gradient, ∇u . Once we have computed the solution, the gradient is computed using once again a compact stencil, which we derive using the PDE 3.1. We start with the second order finite difference stencil for the first derivative,

$$\hat{\partial}_x u_{ij} := \frac{-u_{i-1,j} + u_{i+1,j}}{2h_x} \quad \hat{\partial}_y u_{ij} := \frac{-u_{i,j-1} + u_{i,j+1}}{2h_y} \quad (3.66)$$

Using Taylor series, we have that

$$\hat{\partial}_x u_{ij} = \partial_x u + \frac{h_x^2}{6} u_{xxx} + \mathcal{O}(h_x^4) \quad (3.67)$$

We differentiate the PDE 3.1 with respect to x to rewrite the third derivative and then we approximate :

$$u_{xxx} = \partial_x f - u_{xyy} \quad (3.68)$$

$$= \hat{\partial}_x f - \hat{\partial}_x \partial_{yy} u + \mathcal{O}(h_x^2) \quad (3.69)$$

$$= \hat{\partial}_x f - \hat{\partial}_x \hat{\partial}_{yy} u + \mathcal{O}(h_x^2) + \mathcal{O}(h_y^2) \quad (3.70)$$

A similar procedure can be carried out for the y -derivative, and thus the 9-point, fourth-order approximation to the gradient, ∇_{ij}^9 , is

$$\nabla_{ij}^9 u := \begin{pmatrix} \hat{\partial}_x u + \frac{h_x^2}{6} (\hat{\partial}_x \hat{\partial}_{yy} u - \hat{\partial}_x f) \\ \hat{\partial}_y u + \frac{h_y^2}{6} (\hat{\partial}_y \hat{\partial}_{xx} u - \hat{\partial}_y f) \end{pmatrix} = \nabla u + \mathcal{O}(h^4) \quad (3.71)$$

Once again, the first derivatives of the source term can be computed using second order finite differences if their exact expressions are not available.

Note that for nodes near the interface, this stencil will involve nodes in both Ω^+ and Ω^- . However, this is not a problem since we can always use the correction function D , which we store once it is computed, to correct the solution.

At a node $x_{ij} \in \Omega^+$, let S be the set of nodes appearing in an arbitrary finite difference stencil F_h , and let S^+ and S^- denote the subsets of these nodes located in Ω^+ and Ω^- respectively.

Then we can write, using c as the stencil coefficients, and making use of the definition of D in (3.49),

$$\begin{aligned}
 F_h(u_{ij}^+) &= \sum_{i \in S} c_i u_i^+ \\
 &= \sum_{i \in S^+} c_i u_i^+ + \sum_{i \in S^-} c_i (u_i^- + D_i) \\
 &= \sum_{i \in S} c_i u_i + \sum_{i \in S^-} c_i D_i
 \end{aligned} \tag{3.72}$$

and similarly, for a node in Ω^- ,

$$F_h(u_{ij}^-) = \sum_{i \in S} c_i u_i - \sum_{i \in S^+} c_i D_i \tag{3.73}$$

The fact that the gradient stencil involves the computed correction function for nodes near the interface means that fourth order accuracy is not guaranteed, since the computed correction function is not necessarily smooth, as argued in [31]. In fact, the gradient will be accurate to third order in the worst case.

3.2.3 Interface Description

As mentioned in section 3.1, the interface between the two subdomains Ω^+ and Ω^- is described by a level set function ϕ , i.e. the interface is the zero level set of ϕ . We use the gradient-augmented level set method of [35] to discretize the level set. We thus have 3 grid functions, ϕ , ϕ_x and ϕ_y and we use the reduced bicubic interpolants described in section 3.2.4.2 to compute the value of the level set or its derivatives at arbitrary locations in the computational domain.

Note that if we are only given ϕ on a grid, we can still use the reduced bicubic interpolants as long as we can compute ϕ_x and ϕ_y on the grid to $\mathcal{O}(h^3)$ (see 3.2.4.2). Specifically, if ϕ is known to $\mathcal{O}(h^4)$, we can use fourth order finite differences for the first derivatives to approximate ϕ_x and ϕ_y , and our level set representation will still be fourth order.

We often need to compute the unit normal and tangent vectors to the interface at a point x .

These unit vectors are computed in the following way [36] :

$$n(x) = \frac{\nabla\phi(x)}{\|\nabla\phi(x)\|} \quad (3.74)$$

$$t(x) = (-n_y, n_x) \quad (3.75)$$

where $x \in \mathbb{R}^2$. Note that since these vectors are defined using the level set function, they can be computed at any point in the domain.

We also need to compute the projection on the interface of a point x_0 in the domain, that is, the closest point on the interface to point x_0 . For points near the interface, we assume the level set can be approximated linearly, so a plot of ϕ in the normal direction would look like figure 3.6.

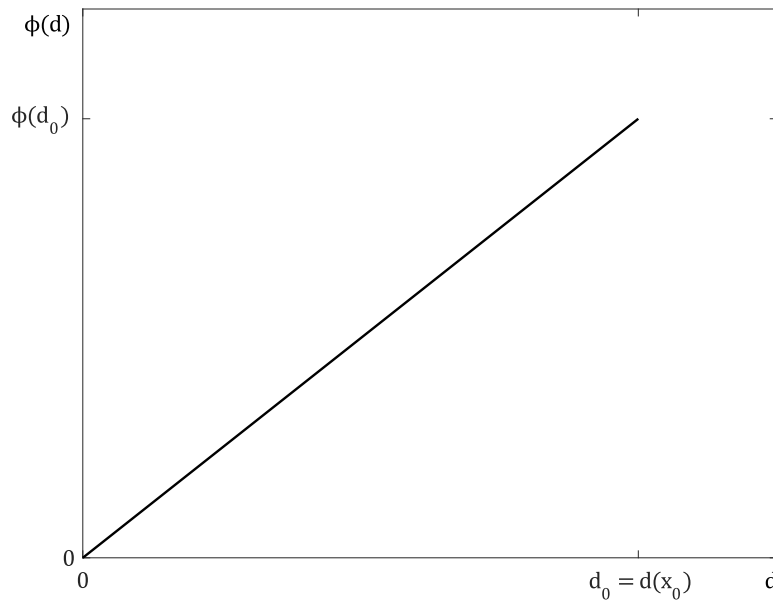


Figure 3.6. Linear approximation to the level set in the normal direction d

The linear approximation permits us to write

$$\phi(d(x_0)) = \alpha d(x_0)$$

where $d(x_0)$ is the distance from the interface to x_0 in the normal direction and α is the slope of the linear approximation. Taking this slope as the value of the gradient at x_0 , we can then

write

$$d(x_0) = \frac{\phi(x_0)}{\|\nabla\phi(x_0)\|}$$

Hence if we move a distance $d(x_0)$ in the normal direction, we should be close to the interface if the level set is well approximated linearly. In the general case, we can repeat this procedure as a fixed point algorithm. We define the projection operator

$$\begin{aligned} P(x) &:= x - \underbrace{\frac{\phi(x)}{\|\nabla\phi(x)\|}}_{\text{approximate distance}} \underbrace{\frac{\nabla\phi(x)}{\|\nabla\phi(x)\|}}_{\text{normal}} \\ &= x - \frac{\phi(x) \nabla\phi(x)}{\|\nabla\phi(x)\|^2} \end{aligned} \quad (3.76)$$

and we have the following algorithm, given a small tolerance ϵ :

Algorithm 3.1 Projection of a point x_0 on the interface

$x^{(0)} \leftarrow x_0$
repeat $x^{(k+1)} = P(x^{(k)})$
until $\phi(x^{(k)}) \leq \epsilon$

Usually, only one iteration is sufficient to get a good projection if the starting point x_0 is not too far from the interface.

3.2.4 Local Solver for the Correction Function

In this section we describe how we solve the problem (3.50-3.52) in order to get the value of the correction function D at each grid point where corrections are needed. Let us first emphasize that a local solver is used : the value of the correction function at each required node is computed independently. As argued in [31], a local solver for the correction function is appropriate because we only require the solution at a small distance from the interface (on the order of the step size h). The correction function equation is solved at node (i, j) by minimizing the following integral quantity :

$$J(u) := c_1 \int_{\Omega_{\Gamma}^{ij}} (\Delta u - f_D)^2 dx + c_p \sum_{k=1}^{N_{ij}} \int_{\Gamma_k^{ij}} \left[c_2 (u - a)^2 + c_3 (\partial_n u - b)^2 \right] ds(x) \quad (3.77)$$

The first term is an integral over the local domain Ω_{Γ}^{ij} , which is defined in section 3.2.4.3 and is shown in figure 3.7. The second and third terms are integrals over the interface, that is, a line integral in 2D (or a surface integral in 3D); $ds(x)$ is the line element (or the surface element in 3D). The interface domains Γ_k^{ij} , $k = 1, \dots, N_{ij}$ are defined in sections 3.2.4.3-3.2.4.4 below and are also shown in figure 3.7. The coefficients c_1, c_2 and c_3 are scaling coefficients, derived in section 3.2.4.6. They are designed to make each term of the integral have the same units, so they scale in the same way when the grid is refined. The coefficient c_p is a penalization coefficient; it controls how much the solution will be influenced by the interface integral.

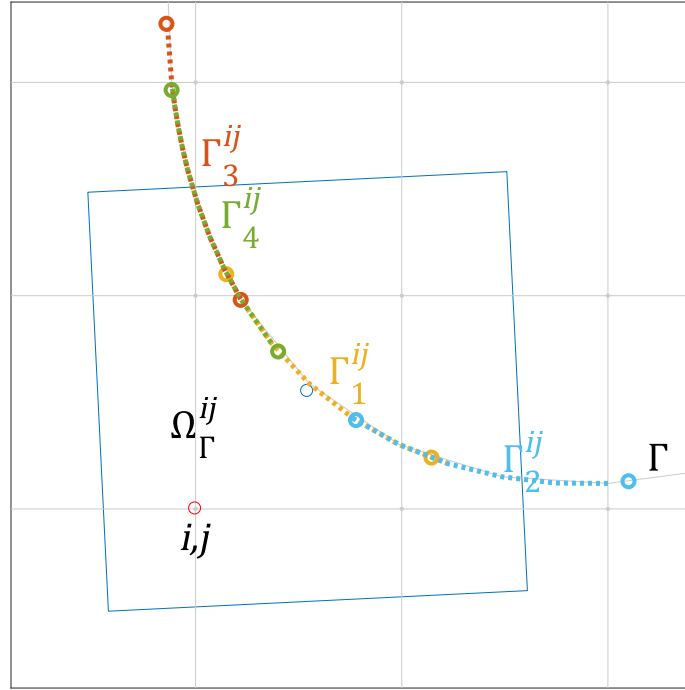


Figure 3.7. The integration domain for the correction function minimization problem

It is clear that the value of the above functional $J(u) \geq 0$ and that when evaluated at the solution D to problem (3.50-3.52), it is zero. This method of solving the correction function problem was introduced in [31] and was further improved in [33], where the decoupling between Ω_{Γ}^{ij} and Γ_k^{ij} was introduced, leading to the functional (3.77). Note that in the case of a discontinuous source term ($f^+ \neq f^-$), as mentioned above (see the discussion below problem (3.50-3.52)), f^+ and f^- need to be extended on the other side of the interface so that the function f_D can be defined. If both f^+ and f^- are only defined on a grid on their respective domains, this means an extrapolation scheme must be used to perform these extensions. This case is not covered by the example problems examined in this work.

To minimize the functional, we first substitute the correction function D by a polynomial approximation :

$$D(x) = \hat{D}(\xi) = \sum_{m=1}^n \bar{B}_m(\xi) \hat{D}_m \quad (3.78)$$

where the \bar{B}_m are polynomial basis functions, the \hat{D}_m are scalar coefficients and n is the number of basis functions. The polynomial basis used is described in section 3.2.4.2. The function $\hat{D}(\xi)$ is a transformed version of the correction function D under a geometric mapping described in section 3.2.4.5. The integrals are then parameterized (see section 3.2.4.4) and discretized using Gauss-Legendre quadrature [42] (see Appendix A), leading to \tilde{J} , the discrete version of the functional.

3.2.4.1 Minimization

Once discretized, the functional \tilde{J} becomes a quadratic function of the polynomial coefficients D_m , i.e.

$$\tilde{J} = x^T A x + d^T x + g \quad (3.79)$$

where $A \in \mathbb{R}^{n \times n}$, $x, d \in \mathbb{R}^n$, $g \in \mathbb{R}$ and $x = (\hat{D}_1, \hat{D}_2, \dots, \hat{D}_n)^T$. We wish to minimize this functional, that is, we want to solve the optimization problem

$$\min_{x \in \mathbb{R}^n} x^T A x + d^T x + g \quad (3.80)$$

This is an unconstrained quadratic optimization problem. Its solution can be computed directly by applying the first-order necessary condition for optimality (see [6], section 2.2) leading to

$$0 = \nabla (x^T A x + d^T x + g) = 2A x + d \quad (3.81)$$

$$\begin{aligned} \Rightarrow x &= -(2A)^{-1} d \\ x &= \frac{-A^{-1} d}{2} \end{aligned} \quad (3.82)$$

The coefficients \hat{D}_m describing the correction function at node (i, j) are thus recovered by solving a $n \times n$ linear system, and the polynomial solution makes the discretized functional \tilde{J} exactly zero (at least to machine precision). The complete expression of the matrix A and of

the vector d are derived in Appendix A. A similar linear system must be solved for each node where the correction function is needed. The method is thus intrinsically highly parallelizable; the value of the correction function at each required node is computed independently from every other node.

3.2.4.2 Bicubic Interpolation

The polynomial interpolation framework used is the bicubic interpolation described in [35]. In this section, to streamline the notation we write $(x, y) = (x_0, x_1)$ and $x = (x_0, x_1) \in \mathbb{R}^2$. Given a function $f(x) : S \rightarrow \mathbb{R}$ defined over the rectangular domain $S = [a_0, b_0] \times [a_1, b_1] \subset \mathbb{R}^2$, we approximate it using the following linear combination of monomials :

$$f(x) \approx \tilde{f}(x) := \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x_0^i x_1^j \quad (3.83)$$

The name “bicubic” comes from the fact that each exponent goes up to order 3. However, this monomial basis is not well-conditioned, so we use a different basis.

The bicubic interpolants, or bicubics, are designed to interpolate a function f given the following data : f, f_x, f_y and f_{xy} on each corner of the domain S . This amounts to 16 pieces of information, or data points, enough to fix the 16 coefficients a_{ij} in (3.83). We use the multi-index $\nu \in \{0, 1\}^2$ to designate the four corners of the domain, with the first index corresponding to the x_0 dimension and the second corresponding to the x_1 dimension (see figure 3.8). Similarly, we use the multi-index $\alpha \in \{0, 1\}^2$ to designate the derivatives of f , with the first index again corresponding to the x_0 variable and the second index corresponding to the x_1 variable. The data point f_α^ν is then defined as

$$f_\alpha^\nu := \partial^\alpha f(x_\nu) \quad (3.84)$$

where x_ν is the coordinate of corner ν and we write the interpolant as

$$\tilde{f}(x) := \sum_{\nu, \alpha \in \{0, 1\}^2} f_\alpha^\nu B_\alpha^\nu(x) \quad (3.85)$$

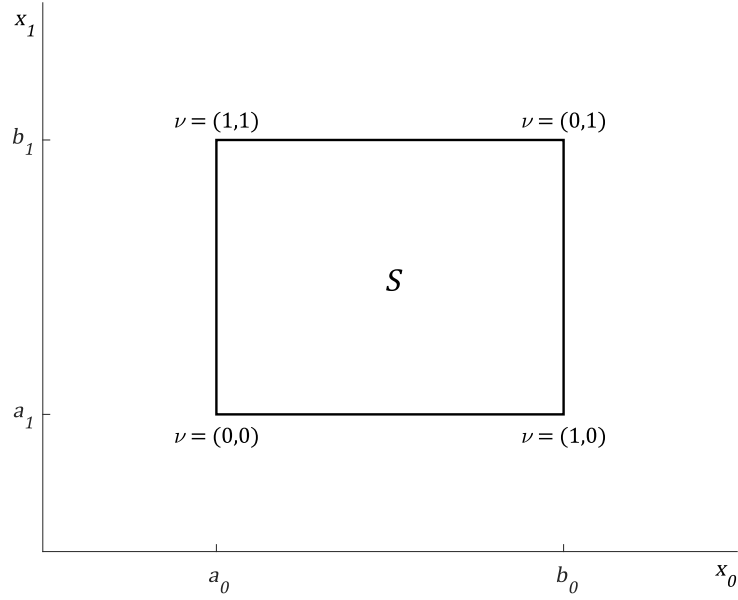


Figure 3.8. The domain S over which the bicubic interpolants are defined.

where B_α^ν are the bicubic polynomial basis functions given by the following definitions

$$B_\alpha^\nu(x) = \Delta x^\alpha W_\alpha^\nu(x), \quad (3.86)$$

$$\Delta x = (\Delta x_0, \Delta x_1) = (b_0 - a_0, b_1 - a_1) \in \mathbb{R}^2, \quad (3.87)$$

$$W_\alpha^\nu(x) = \prod_{i=0}^1 w_{\alpha_i}^{\nu_i}(\bar{x}_i), \quad (3.88)$$

$$\bar{x}_i = \frac{x_i - a_i}{\Delta x_i} \quad (3.89)$$

The univariate polynomials $w_{\alpha_i}^{\nu_i}$ are defined by

$$w_0^0(x) = f(x) \quad w_0^1(x) = f(1 - x) \quad (3.90)$$

$$w_1^0(x) = g(x) \quad w_1^1(x) = -g(1 - x) \quad (3.91)$$

and the cubic polynomials f and g are

$$f(x) = 1 - 3x^2 + 2x^3 \quad (3.92)$$

$$g(x) = x(1 - x)^2 \quad (3.93)$$

It is shown in [35] that if the bicubic interpolants are constructed from data f_α^ν known to $\mathcal{O}(h^{4-|\alpha|})$, then

$$\partial^\alpha (f(x) - \tilde{f}(x)) = \mathcal{O}(h^{4-|\alpha|}) \quad (3.94)$$

The polynomials B_α^ν are generalized Lagrange polynomials in the sense that for multi-indices $\gamma, \delta \in \{0, 1\}^2$,

$$\partial^\gamma B_\alpha^\nu(x_\delta) = \begin{cases} 1 & \gamma = \alpha, \delta = \nu \\ 0 & \text{otherwise} \end{cases} \quad (3.95)$$

3.2.4.2.1 Reduced Cell-Based Bicubic Interpolation

The result (3.94) above relating the accuracy of the bicubic interpolant to the accuracy of the data defining it permits us to reduce the number of parameters needed to determine the bicubic interpolants, without reducing the accuracy of the result, as explained in [35]. Indeed, we can describe the interpolant using only the data f , f_x and f_y at the four corners, without necessitating the mixed derivative f_{xy} . In fact, the mixed derivative can be computed to $\mathcal{O}(h^2)$ from the first derivatives using finite differences and bilinear extrapolation, as we next show. We first compute the mixed derivative at the midpoint of each face of the rectangular cell (points A-D in figure 3.9). We use the standard finite difference formula for the first derivative :

$$f_{xy}^A = \frac{f_y^2 - f_y^1}{\Delta x} + \mathcal{O}(\Delta x^2) \quad f_{xy}^C = \frac{f_y^3 - f_y^4}{\Delta x} + \mathcal{O}(\Delta x^2) \quad (3.96)$$

$$f_{xy}^B = \frac{f_x^3 - f_x^2}{\Delta y} + \mathcal{O}(\Delta y^2) \quad f_{xy}^D = \frac{f_x^4 - f_x^1}{\Delta y} + \mathcal{O}(\Delta y^2) \quad (3.97)$$

We then compute the mixed derivative at points numbered i-iv in figure 3.9, located at the midpoints of segments \overline{AB} , \overline{BC} , \overline{CD} and \overline{DA} , by taking means :

$$f_{xy}^i = \frac{f_{xy}^A + f_{xy}^B}{2} + \mathcal{O}(h^2) \quad f_{xy}^{ii} = \frac{f_{xy}^B + f_{xy}^C}{2} + \mathcal{O}(h^2) \quad (3.98)$$

$$f_{xy}^{iii} = \frac{f_{xy}^C + f_{xy}^D}{2} + \mathcal{O}(h^2) \quad f_{xy}^{iv} = \frac{f_{xy}^D - f_{xy}^A}{2} + \mathcal{O}(h^2) \quad (3.99)$$

Finally, we use bilinear extrapolation to compute the mixed derivative at the corners of the

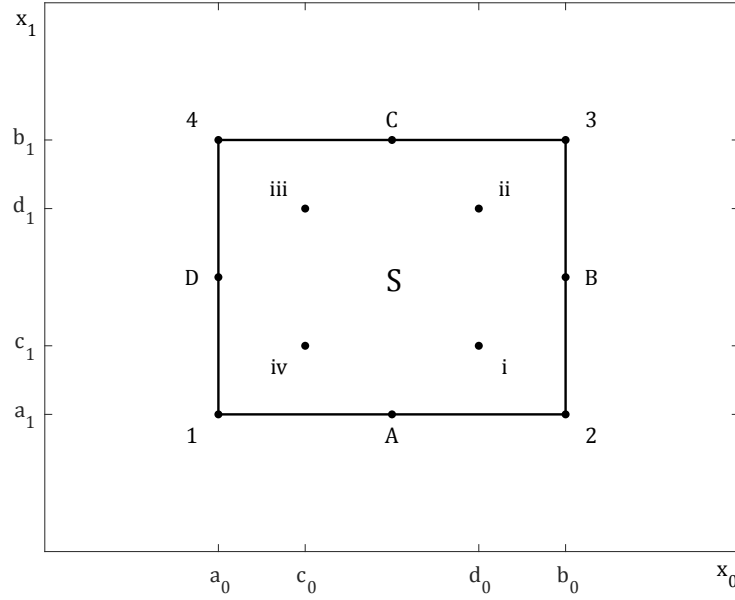


Figure 3.9. Nodes needed for deriving the reduced bicubic interpolants

cell. The formula is the same as bilinear interpolation, but the interpolant is evaluated outside the rectangular box of size $(\Delta x/2 \times \Delta y/2)$ defined by points i-iv :

$$f_{xy}(x, y) = \frac{1}{(d_0 - c_0)(d_1 - c_1)} \left[f_{xy}^{iv} (d_0 - x)(d_1 - y) + f_{xy}^i (x - c_0)(d_1 - y) \right. \\ \left. + f_{xy}^{iii} (d_0 - x)(y - c_1) + f_{xy}^{ii} (x - c_0)(y - c_1) \right] \quad (3.100)$$

For example, at point 2, we have $(x, y) = (b_0, a_1)$ so using

$$\begin{aligned} d_0 - x &= \frac{-\Delta x}{4} & d_1 - y &= \frac{3\Delta y}{4} \\ x - c_0 &= \frac{3\Delta x}{4} & y - c_1 &= \frac{-\Delta y}{4} \end{aligned}$$

and simplifying, we get

$$f_{xy}^2 = \frac{1}{4}f_{xy}^i + \frac{1}{4}f_{xy}^{ii} - \frac{3}{4}f_{xy}^{iii} - \frac{3}{4}f_{xy}^{iv} \quad (3.101)$$

Using the formula (3.100) and equations (3.96-3.99), we can then write the mixed derivative at the corners as linear combinations of the other data :

$$f_{xy}^{00} = \frac{3}{4\Delta x}(f_y^{10} - f_y^{00}) + \frac{3}{4\Delta y}(f_x^{01} - f_x^{00}) - \frac{1}{4\Delta x}(f_y^{11} - f_y^{01}) - \frac{1}{4\Delta y}(f_x^{11} - f_x^{10}) \quad (3.102)$$

$$f_{xy}^{10} = \frac{3}{4\Delta x}(f_y^{10} - f_y^{00}) + \frac{3}{4\Delta y}(f_x^{11} - f_x^{10}) - \frac{1}{4\Delta x}(f_y^{11} - f_y^{01}) - \frac{1}{4\Delta y}(f_x^{01} - f_x^{00}) \quad (3.103)$$

$$f_{xy}^{01} = \frac{3}{4\Delta x}(f_y^{11} - f_y^{01}) + \frac{3}{4\Delta y}(f_x^{01} - f_x^{00}) - \frac{1}{4\Delta x}(f_y^{10} - f_y^{00}) - \frac{1}{4\Delta y}(f_x^{11} - f_x^{10}) \quad (3.104)$$

$$f_{xy}^{11} = \frac{3}{4\Delta x}(f_y^{11} - f_y^{01}) + \frac{3}{4\Delta y}(f_x^{11} - f_x^{10}) - \frac{1}{4\Delta x}(f_y^{10} - f_y^{00}) - \frac{1}{4\Delta y}(f_x^{01} - f_x^{00}) \quad (3.105)$$

This means that we can now construct the bicubic interpolants using only the function values and first derivatives at the four corners of the cell. We can also define reduced bicubic interpolants, by combining the polynomials B_α^ν and the mixed derivative data (3.102-3.105).

Using the new indices m in table 3.1, we define the reduced bicubic polynomials \bar{B}_m :

$$\bar{B}_i = B_i, \quad i = 1, \dots, 4 \quad (3.106)$$

$$\bar{B}_5 = B_5 + \frac{1}{4\Delta y}(-3B_{13} + B_{14} - 3B_{15} + B_{16}) \quad (3.107)$$

$$\bar{B}_6 = B_6 + \frac{1}{4\Delta y}(B_{13} - 3B_{14} + B_{15} - 3B_{16}) \quad (3.108)$$

$$\bar{B}_7 = B_7 + \frac{1}{4\Delta y}(3B_{13} - B_{14} + 3B_{15} - B_{16}) \quad (3.109)$$

$$\bar{B}_8 = B_8 + \frac{1}{4\Delta y}(-B_{13} + 3B_{14} - B_{15} + 3B_{16}) \quad (3.110)$$

$$\bar{B}_9 = B_9 + \frac{1}{4\Delta x}(-3B_{13} - 3B_{14} + B_{15} + B_{16}) \quad (3.111)$$

$$\bar{B}_{10} = B_{10} + \frac{1}{4\Delta x}(3B_{13} + 3B_{14} - B_{15} - B_{16}) \quad (3.112)$$

$$\bar{B}_{11} = B_{11} + \frac{1}{4\Delta x}(B_{13} + B_{14} - 3B_{15} - 3B_{16}) \quad (3.113)$$

$$\bar{B}_{12} = B_{12} + \frac{1}{4\Delta x}(-B_{13} - B_{14} + 3B_{15} + 3B_{16}) \quad (3.114)$$

The reduced interpolant is then written

$$\tilde{f}(x) = \sum_{m=1}^{12} f_m \bar{B}_m(x) \quad (3.115)$$

and the accuracy result (3.94) still holds.

Table 3.1. Single indices for the bicubics interpolants

α	ν	m	α	ν	m
	00	1		00	9
00	10	2	01	10	10
	01	3		01	11
	11	4		11	12
	00	5		00	13
10	10	6	11	10	14
	01	7		01	15
	11	8		11	16

3.2.4.3 Domain Definition

For each node x_{ij} where corrections are needed, we define a domain Ω_{Γ}^{ij} using the node-centered approach discussed in [31]. This domain is a square box of side $s := \sqrt{2}h$ where $h := \sqrt{h_x^2 + h_y^2}$. Its center x_c is given by the projection of node x_{ij} on the interface. We use the level set function to perform this projection, using the fixed point algorithm described in section 3.2.3. The corners of the square are situated at

$$x_1 = x_0 + ht \quad x_2 = x_0 + hn \quad (3.116)$$

$$x_3 = x_0 - ht \quad x_4 = x_0 - hn \quad (3.117)$$

where n and t are the normal and tangent unit vectors to the interface at x_c .

Also, for each computational cell crossed by the interface, we define a transformation box M . This box is used to define the interface segments Γ_k^{ij} in the second and third terms of (3.77), as explained in section 3.2.4.4. We first project the cell center x_k onto the interface, at point x_k^0 . The transformation box M is a square of side h , centered at x_k^0 . Its corners are situated at

$$x_{00} = x_k^0 + \frac{h}{2}(-n - t) \quad x_{10} = x_k^0 + \frac{h}{2}(n - t) \quad (3.118)$$

$$x_{01} = x_k^0 + \frac{h}{2}(-n + t) \quad x_{11} = x_k^0 + \frac{h}{2}(n + t) \quad (3.119)$$

where n and t are the normal and tangent unit vectors to the interface at x_k^0 . The interface

segments included in the integral (3.77) are all those for which the center x_k^0 of their transformation box is at a distance of h or less from x_c , the center of the domain Ω_Γ^{ij} (see figure 3.10).

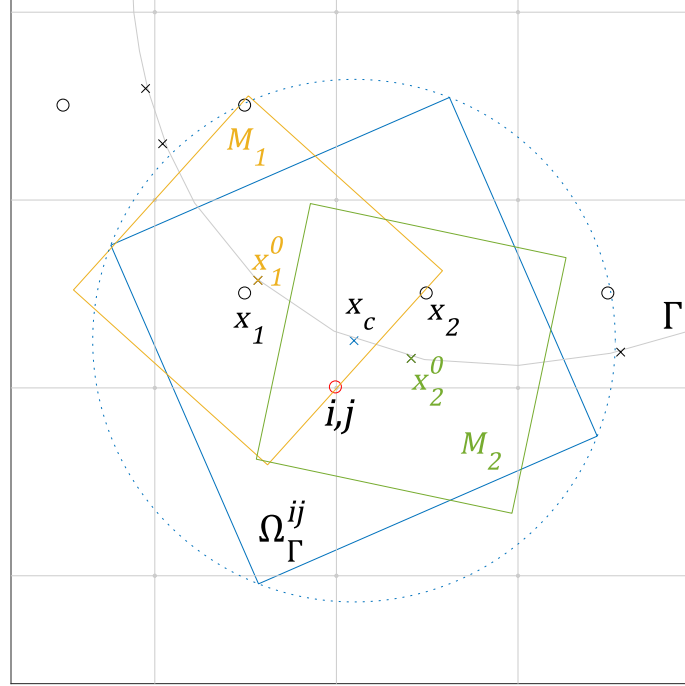


Figure 3.10. The node (i, j) where corrections are needed is marked in red, and its projection on the interface x_c is marked with a blue cross. The cell centers of cells crossed by the interface are marked with black circles, and their projections on the interface are marked with black crosses. Those inside a radius of h around x_c , located inside the blue circle, are used to define the transformation boxes M_1 and M_2 . In the case depicted above, $N_{ij} = 2$ in equation (3.77).

3.2.4.4 Parameterization

The interface integrals in (3.77) must be parameterized before we can discretize them. To do so, we use a transformation $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, designed to map the interface segment Γ_k^{ij} associated with a cell crossed by the interface to a straight line. The transformation T maps the real domain coordinates (x, y) (“the x -space”) to the parameter space (θ_1, θ_2) (the “ θ -space”). This transformation is illustrated in figure 3.11 and is defined using the level set function as

$$\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = T(x) := \begin{pmatrix} \frac{\phi(x)}{h \nabla \phi_0} + \frac{1}{2} \\ \frac{t \cdot (x - x_0)}{h} + \frac{1}{2} \end{pmatrix} \quad (3.120)$$

3.2. Numerical Method

where x_0 is the center of the transformation box M and t is the tangent at x_0 , computed from the level set information (see section 3.2.3). The use of the level set insures that the θ coordinates are aligned with the interface. The transformation T is designed to map the interface segment Γ_k^{ij} to the line segment

$$\theta_1 = \Theta := 0.5, \quad \theta_2 \in [0, 1] \quad (3.121)$$

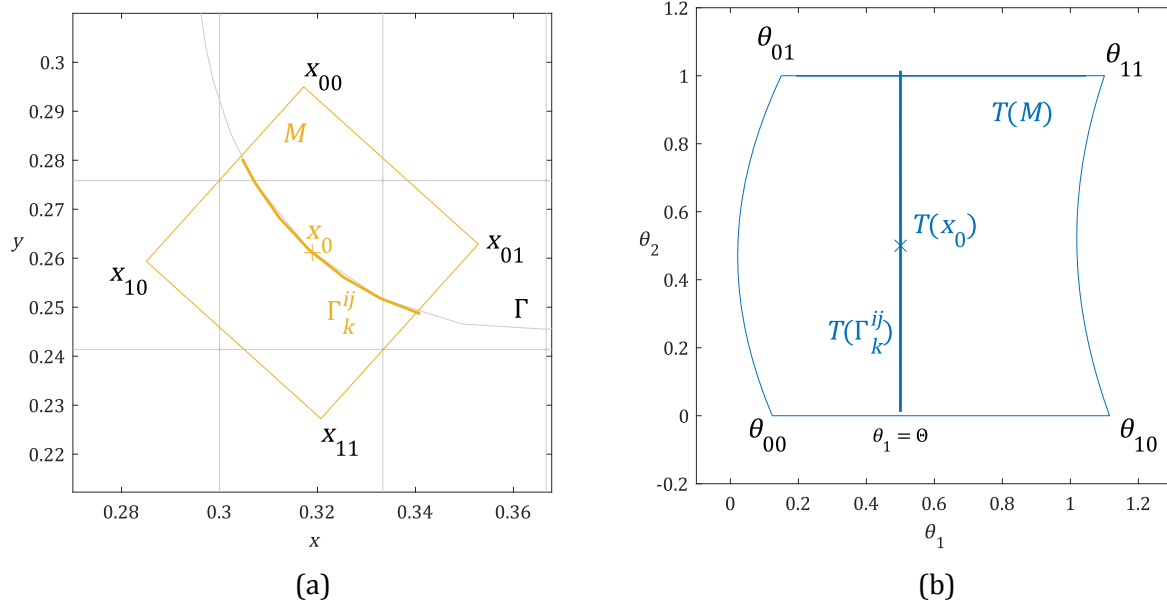


Figure 3.11. The transformation T maps the box M in x -space (a) to the parameter space θ (b). The interface segment Γ_k^{ij} is transformed to a straight line.

We parameterize the interface integral using the inverse transformation, T^{-1} , which maps from the parameter space θ to the real space x . To integrate a general function f on the interface segment Γ_k^{ij} , we use

$$\int_{\Gamma_k^{ij}} f(x) ds(x) = \int_0^1 f(T^{-1}(\Theta, \theta_2)) \left\| \frac{\partial T^{-1}}{\partial \theta_2} \right\|_{(\Theta, \theta_2)} d\theta_2 \quad (3.122)$$

Notice T^{-1} involves the inverse of the level set function ϕ (see (3.120)), for which an exact expression is usually unavailable. To overcome this limitation, we use the reduced bicubic interpolants described in section 3.2.4.2 to *approximate* the inverse transformation. We write T^{-1} as

$$x = T^{-1}(\theta) \approx \tilde{T}^{-1}(\theta) := \begin{pmatrix} H_1(\theta_1, \theta_2) \\ H_2(\theta_1, \theta_2) \end{pmatrix} \quad (3.123)$$

where \tilde{T}^{-1} is the approximate inverse transformation and H_1 and H_2 are reduced bicubic interpolants. We determine the coefficients of these bicubics by solving two linear systems. We have 24 coefficients to determine ; 12 for H_1 and 12 for H_2 . Since locally, at the level of the transformation box M , the interface is close to a straight line (or a flat surface in 3D), we can write

$$T \circ \tilde{T}^{-1} = I + \mathcal{O}(h^4) \quad (3.124)$$

This comes from the fact that the interface is locally quasi-straight and thus is well approximated by a polynomial. The fourth-order bicubics used for the level set representation as well as to approximate the inverse transformation guarantee the fourth-order accuracy, as argued in [33]. Equation (3.124) allows us to write

$$DT D\tilde{T}^{-1} \approx I \quad (3.125)$$

where the total derivative D corresponds to the Jacobian matrix. To determine the coefficients of the bicubics H_1 and H_2 , we simply evaluate the interpolants and their derivatives at convenient locations: the corners $x_\gamma, \gamma \in \{0, 1\}^2$ of the transformation box M (see (3.118-3.119) and figure 3.11). Using the forward transformation T , we compute the transformed coordinates

$$\theta_\gamma := T(x_\gamma) \quad (3.126)$$

and then we use the approximate inverse transformation to write

$$\tilde{T}^{-1}(\theta_\gamma) = x_\gamma \quad (3.127)$$

Also, making explicit the Jacobians in (3.125), we get

$$\left. \frac{\partial T}{\partial x} \right|_{x_\gamma} \left. \frac{\partial \tilde{T}^{-1}}{\partial \theta} \right|_{\theta_\gamma} \approx I \Rightarrow \left. \frac{\partial \tilde{T}^{-1}}{\partial \theta} \right|_{\theta_\gamma} = \left(\left. \frac{\partial T}{\partial x} \right|_{x_\gamma} \right)^{-1} \quad (3.128)$$

In a compact way, both 12×12 linear systems are thus given by

$$\begin{cases} \tilde{T}^{-1}(\theta_\gamma) = x_\gamma \\ \left. \frac{\partial \tilde{T}^{-1}}{\partial \theta} \right|_{\theta_\gamma} = \left(\left. \frac{\partial T}{\partial x} \right|_{x_\gamma} \right)^{-1} \end{cases} \quad (3.129)$$

with $\gamma \in \{0, 1\}^2$. The first line is a vector expression and encompasses 2 equations, and the second is a matrix expression and encompasses 4 equations. When we evaluate these 6 equations on each of the four corners θ_γ , we arrive at a total of 24 equations, as needed to complete both linear systems. To explicitly write these systems, we define the inverse Jacobian elements

$$\frac{\partial x}{\partial \theta} = \begin{pmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{pmatrix} := \left(\frac{\partial T}{\partial x} \right)^{-1} \quad (3.130)$$

and we can then use the expression of the bicubics (3.115) to write the linear systems :

$$\begin{cases} \bar{B}_i(\theta_\gamma)h_i^1 = x_\gamma \\ \partial_{\theta_1} \bar{B}_i(\theta_\gamma)h_i^1 = \left. \frac{\partial x}{\partial \theta_1} \right|_{x_\gamma} \\ \partial_{\theta_2} \bar{B}_i(\theta_\gamma)h_i^1 = \left. \frac{\partial x}{\partial \theta_2} \right|_{x_\gamma} \end{cases} \quad \begin{cases} \bar{B}_i(\theta_\gamma)h_i^2 = y_\gamma \\ \partial_{\theta_1} \bar{B}_i(\theta_\gamma)h_i^2 = \left. \frac{\partial y}{\partial \theta_1} \right|_{x_\gamma} \\ \partial_{\theta_2} \bar{B}_i(\theta_\gamma)h_i^2 = \left. \frac{\partial y}{\partial \theta_2} \right|_{x_\gamma} \end{cases} \quad \gamma \in \{0, 1\}^2 \quad (3.131)$$

where we used the Einstein summation convention (repeated indices are summed, here from $i = 1, \dots, 12$). In equation (3.131) above, x_γ and y_γ are the first and second coordinates in x -space, and h_i^1 and h_i^2 , $i = 1, \dots, 12$ are the coefficients of the bicubics H_1 and H_2 respectively. Notice both linear systems have the same matrix but different right-hand side vectors, a fact that can be leveraged to efficiently solve both systems at the same time (see, for example, [18]). The two linear systems (3.131) must be solved for each transformation box M . Once again, this is done in a completely independent fashion for each box and thus provides another opportunity for parallelization of the implementation.

Making use of the bicubic interpolants for the approximate inverse transformation, we can

rewrite the interface integral (3.122) as

$$\begin{aligned}
\int_{\Gamma_k^{ij}} f(x) ds(x) &= \int_0^1 f(\tilde{T}^{-1}(\Theta, \theta_2)) \left\| \frac{\partial \tilde{T}^{-1}}{\partial \theta_2} \right\|_{(\Theta, \theta_2)} d\theta_2 \\
&= \int_0^1 f(T^{-1}(\Theta, \theta_2)) \left\| \begin{pmatrix} \frac{\partial H_1}{\partial \theta_2} & \frac{\partial H_2}{\partial \theta_2} \end{pmatrix} \right\|_{(\Theta, \theta_2)} d\theta_2 \\
&= \int_0^1 f(T^{-1}(\Theta, \theta_2)) \sqrt{\left(\frac{\partial H_1}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_2)} d\theta_2
\end{aligned} \tag{3.132}$$

3.2.4.5 Mapping to the Reference Domain

We use an affine transformation, \hat{T} , to map the integration domain Ω_Γ from the unit square $K := [0, 1]^2$, denoted as the ξ -space with coordinates (ξ, η) , to the real space x . The goal of this second transformation is to use bicubic interpolants defined on K — instead of being defined on the real space x — for the approximation of the correction function D , for reasons which are explained in section 5.2. Since the domain Ω_Γ^{ij} is a square, the transformation \hat{T} is simply a composition of a rotation, a scaling and a translation, and its inverse is readily available :

$$x = \hat{T}(\xi) := s \begin{pmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{pmatrix} (\xi - \xi_c) + x_c \tag{3.133}$$

$$\xi = \hat{T}^{-1}(x) := \frac{1}{s} \begin{pmatrix} \cos \varphi & \sin \varphi \\ -\sin \varphi & \cos \varphi \end{pmatrix} (x - x_c) + \xi_c \tag{3.134}$$

where s and x_c are the side length and center of Ω_Γ^{ij} , $\varphi = \arctan\left(\frac{y_1 - y_2}{x_1 - x_2}\right)$ is the angle of rotation of Ω_Γ with respect to the x -axis and $\xi_c = \left(\frac{1}{2}, \frac{1}{2}\right)$. This transformation is shown in figure 3.12. In the next sections, we use the transformation \hat{T} to transform the minimization integral (3.77) to the reference domain K .

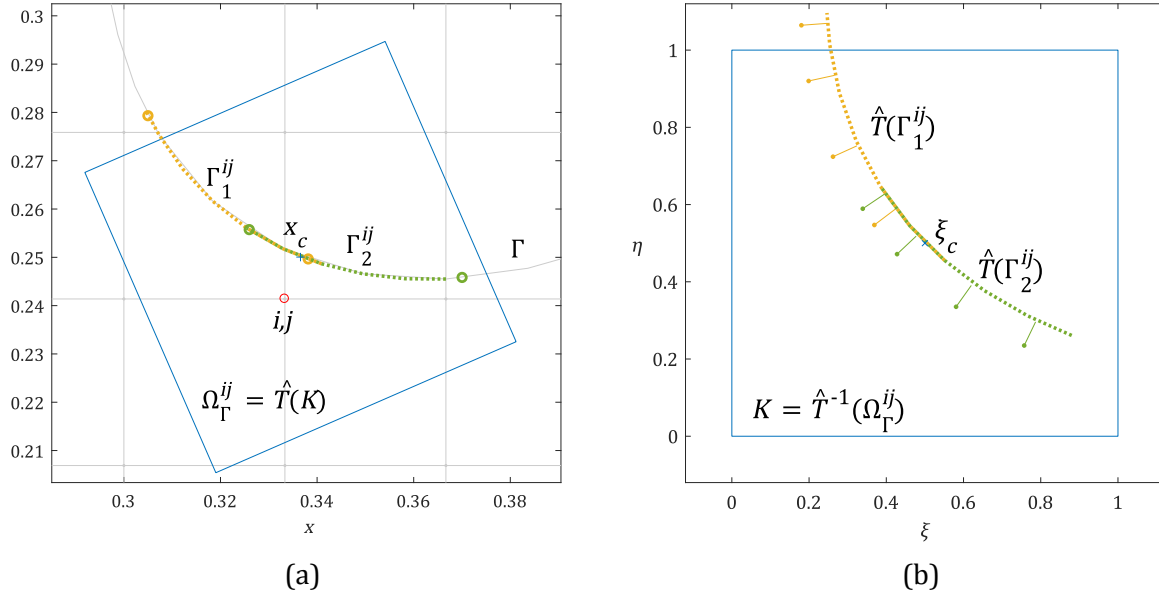


Figure 3.12. The transformation \hat{T} maps the unit square K in the integration space ξ (b) to the integration domain Ω_{Γ}^{ij} in x -space (a). The transformed normals \hat{n} (see equation(3.156)) appear in (b).

3.2.4.5.1 Domain Integral

The integral on the domain of a general function f is transformed according to

$$\int_{\Omega_{\Gamma}} f(x) dx = \int_K f(\hat{T}(\xi)) \left| \frac{\partial \hat{T}}{\partial \xi} \right| d\xi \quad (3.135)$$

$$=: \int_K \hat{f}(\xi) \left| \frac{\partial \hat{T}}{\partial \xi} \right| d\xi \quad (3.136)$$

where we introduce the notation

$$\hat{f}(\xi) := f(\hat{T}(\xi)) \quad (3.137)$$

to denote the composition of a function f with the coordinate transformation \hat{T} . Referring to (3.77), we need to transform the Laplacian of the correction function D to the reference domain. We thus have to relate the second derivatives with respect to x and y to the derivatives of \hat{D} , the transformed correction function. We start with the first derivatives :

$$\partial_x f = \partial_x \hat{f}(\hat{T}^{-1}(x, y)) = \partial_{\xi} \hat{f} \partial_x \xi + \partial_{\eta} \hat{f} \partial_x \eta \quad (3.138)$$

$$\partial_y f = \partial_y \hat{f}(\hat{T}^{-1}(x, y)) = \partial_{\xi} \hat{f} \partial_y \xi + \partial_{\eta} \hat{f} \partial_y \eta \quad (3.139)$$

Note that the gradient of f is then written

$$\begin{aligned}\nabla_x f &= \begin{pmatrix} \partial_x \xi & \partial_x \eta \\ \partial_y \xi & \partial_y \eta \end{pmatrix} \begin{pmatrix} \partial_\xi \hat{f} \\ \partial_\eta \hat{f} \end{pmatrix} \\ &= D\hat{T}^{-\top} \nabla_\xi \hat{f}\end{aligned}\tag{3.140}$$

$$= \frac{1}{s} \begin{pmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{pmatrix} \nabla_\xi \hat{f}\tag{3.141}$$

For the second derivative with respect to x , we have :

$$\begin{aligned}\partial_x^2 f &= \partial_x (\partial_\xi \hat{f} \partial_x \xi + \partial_\eta \hat{f} \partial_x \eta) \\ &= \partial_x (\partial_\xi \hat{f}) \partial_x \xi + \partial_\xi \hat{f} \partial_x^2 \xi + \partial_x (\partial_\eta \hat{f}) \partial_x \eta + \partial_\eta \hat{f} \partial_x^2 \eta \\ &= [\partial_\xi^2 \hat{f} \partial_x \xi + \partial_\eta \partial_\xi \hat{f} \partial_x \eta] \partial_x \xi + \partial_\xi \hat{f} \partial_x^2 \xi \\ &\quad + [\partial_\xi \partial_\eta \hat{f} \partial_x \xi + \partial_\eta^2 \hat{f} \partial_x \eta] \partial_x \eta + \partial_\eta \hat{f} \partial_x^2 \eta \\ &= \partial_\xi^2 \hat{f} (\partial_x \xi)^2 + 2\partial_\xi \partial_\eta \hat{f} \partial_x \xi \partial_x \eta + \partial_\eta^2 \hat{f} (\partial_x \eta)^2\end{aligned}\tag{3.142}$$

Notice that the second derivatives of the transformation, $\partial_x^2 \xi$ and $\partial_x^2 \eta$, are zero since \hat{T} is affine, so the two terms involving them do not appear on the last line. A similar derivation for f_{yy} yields

$$\partial_y^2 f = \partial_\xi^2 \hat{f} (\partial_y \xi)^2 + 2\partial_\xi \partial_\eta \hat{f} \partial_y \xi \partial_y \eta + \partial_\eta^2 \hat{f} (\partial_y \eta)^2\tag{3.143}$$

We can now write the transformed Laplacian using (3.142-3.143) and the definition of the inverse transformation, (3.134):

$$\begin{aligned}\Delta_x D &= \partial_x^2 D + \partial_y^2 D \\ &= \frac{\cos^2 \phi}{s^2} \partial_\xi^2 \hat{D} - \frac{2 \sin \phi \cos \phi}{s^2} \partial_\xi \partial_\eta \hat{D} + \frac{\sin^2 \phi}{s^2} \partial_\eta^2 \hat{D} + \\ &\quad \frac{\sin^2 \phi}{s^2} \partial_\xi^2 \hat{D} + \frac{2 \sin \phi \cos \phi}{s^2} \partial_\xi \partial_\eta \hat{D} + \frac{\cos^2 \phi}{s^2} \partial_\eta^2 \hat{D} \\ &= \frac{1}{s^2} (\partial_\xi^2 \hat{D} + \partial_\eta^2 \hat{D}) \\ &= \frac{1}{s^2} \Delta_\xi \hat{D}\end{aligned}\tag{3.144}$$

The transformed correction function \hat{D} is defined on the reference domain K and can thus be approximated using bicubic interpolants also defined over K :

$$\hat{D}(\xi) \approx \sum_{m=1}^{12} \bar{B}_m(\xi) \hat{D}_m \quad (3.145)$$

The first term of the functional (3.77) can then be written as

$$\begin{aligned} I_1 &:= c_1 \int_{\Omega_{\Gamma}^{ij}} (\Delta u - f_D)^2 dx \\ &= c_1 \int_K \left(\frac{1}{s^2} \Delta \hat{D}(\xi) - f_D(\hat{T}(\xi)) \right)^2 s^2 d\xi \\ &= c_1 \int_K \left(\frac{1}{s^2} [\partial_{\xi}^2 \hat{D}(\xi) + \partial_{\eta}^2 \hat{D}(\xi)] - f_D(\hat{T}(\xi)) \right)^2 s^2 d\xi \\ &= c_1 \int_K \left(\frac{1}{s^2} \sum_m [\partial_{\xi}^2 \bar{B}_m(\xi) + \partial_{\eta}^2 \bar{B}_m(\xi)] \hat{D}_m - f_D(\hat{T}(\xi)) \right)^2 s^2 d\xi \\ &= c_1 \int_K \left(\frac{1}{s^2} \sum_m L_m(\xi) \hat{D}_m - f_D(\hat{T}(\xi)) \right)^2 s^2 d\xi \end{aligned} \quad (3.146)$$

where we explicitly computed the Jacobian determinant $\left| \frac{\partial \hat{T}}{\partial x} \right| = s^2$ and introduced the short-cut notation

$$L_m(\xi) := \partial_{\xi}^2 \bar{B}_m(\xi) + \partial_{\eta}^2 \bar{B}_m(\xi) \quad (3.147)$$

3.2.4.5.2 Interface Integrals

To transform the interface integrals, we use the following formula to transform the integral of an arbitrary function f on the curve \mathcal{C} :

$$\int_{\mathcal{C}} f(x) ds(x) = \int_{\hat{T}^{-1}(\mathcal{C})} f(\hat{T}(\xi)) \left| \frac{\partial \hat{T}}{\partial x} \right| \|D\hat{T}^{-\top} n^K\| d\hat{s}(\xi) \quad (3.148)$$

where n^K and $d\hat{s}(\xi)$ are respectively the unit normal and the line element in the ξ -space. Since we integrate on $\mathcal{C} = \Gamma_k^{ij}$, we parameterize this integral using the transformation de-

scribed above in section 3.2.4.4, making note of the relation $x = \hat{T}(\xi) = \tilde{T}^{-1}(\theta)$:

$$\int_{\hat{T}^{-1}(\Gamma_k^{ij})} f(\hat{T}(\xi)) \left\| \frac{\partial \hat{T}}{\partial x} \right\| \|D\hat{T}^{-\top} n^K\| d\hat{s}(\xi) = \int_0^1 f(\tilde{T}^{-1}(\theta)) \left\| \frac{\partial \hat{T}}{\partial x} \right\| \|D\hat{T}^{-\top} n^K\| \left\| \frac{\partial \xi}{\partial \theta_2} \right\| d\theta_2$$

The factor $\|D\hat{T}^{-\top} n^K\|$ is easily computed :

$$\begin{aligned} \|D\hat{T}^{-\top} n^K\| &= \left\| \frac{1}{s} \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} n_\xi^K \\ n_\eta^K \end{pmatrix} \right\| \\ &= \frac{1}{s} \left\| \begin{pmatrix} n_\xi^K \cos \phi & -n_\eta^K \sin \phi \\ n_\xi^K \sin \phi & n_\eta^K \cos \phi \end{pmatrix} \right\| \\ &= \frac{1}{s} \left[(n_\xi^K)^2 \cos^2 \phi + (n_\eta^K)^2 \sin^2 \phi - 2n_\xi^K n_\eta^K \cos \phi \sin \phi \right. \\ &\quad \left. + (n_\xi^K)^2 \sin^2 \phi + (n_\eta^K)^2 \cos^2 \phi + 2n_\xi^K n_\eta^K \cos \phi \sin \phi \right]^{1/2} \\ &= \frac{1}{s} \left[(n_\xi^K)^2 + (n_\eta^K)^2 \right]^{1/2} \end{aligned} \tag{3.149}$$

$$= 1/s \tag{3.150}$$

The Jacobian of the parameterization $\left\| \frac{\partial \xi}{\partial \theta_2} \right\|$ is computed below. Let \hat{T}_1^{-1} and \hat{T}_2^{-1} denote the two components of the inverse transformation \hat{T}^{-1} , we have that

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} \hat{T}_1^{-1}(x, y) \\ \hat{T}_2^{-1}(x, y) \end{pmatrix} = \begin{pmatrix} \hat{T}_1^{-1}(H_1(\theta), H_2(\theta)) \\ \hat{T}_2^{-1}(H_1(\theta), H_2(\theta)) \end{pmatrix} \tag{3.151}$$

and so

$$\frac{\partial}{\partial \theta_2} \begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} \frac{\partial \hat{T}_1^{-1}}{\partial x} \frac{\partial H_1}{\partial \theta_2} + \frac{\partial \hat{T}_1^{-1}}{\partial y} \frac{\partial H_2}{\partial \theta_2} \\ \frac{\partial \hat{T}_2^{-1}}{\partial x} \frac{\partial H_1}{\partial \theta_2} + \frac{\partial \hat{T}_2^{-1}}{\partial y} \frac{\partial H_2}{\partial \theta_2} \end{pmatrix}$$

$$= \frac{1}{s} \begin{pmatrix} \cos \phi \frac{\partial H_1}{\partial \theta_2} + \sin \phi \frac{\partial H_2}{\partial \theta_2} \\ -\sin \phi \frac{\partial H_1}{\partial \theta_2} + \cos \phi \frac{\partial H_2}{\partial \theta_2} \end{pmatrix}$$

Finally,

$$\begin{aligned} \left\| \frac{\partial \xi}{\partial \theta_2} \right\| &= \frac{1}{s} \left[\cos^2 \phi \left(\frac{\partial H_1}{\partial \theta_2} \right)^2 + \sin^2 \phi \left(\frac{\partial H_2}{\partial \theta_2} \right)^2 + 2 \cos \phi \sin \phi \frac{\partial H_1}{\partial \theta_2} \frac{\partial H_2}{\partial \theta_2} \right. \\ &\quad \left. + \sin^2 \phi \left(\frac{\partial H_1}{\partial \theta_2} \right)^2 + \cos^2 \phi \left(\frac{\partial H_2}{\partial \theta_2} \right)^2 - 2 \cos \phi \sin \phi \frac{\partial H_1}{\partial \theta_2} \frac{\partial H_2}{\partial \theta_2} \right]^{1/2} \\ &= \frac{1}{s} \left[\left(\frac{\partial H_1}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2}{\partial \theta_2} \right)^2 \right]^{1/2} \\ &= \frac{1}{s} \left\| \frac{\partial x}{\partial \theta_2} \right\| \end{aligned} \tag{3.152}$$

The interface integral thus becomes

$$\begin{aligned} \int_{\Gamma_k^{ij}} f(x) ds(x) &= \int_0^1 f(\tilde{T}^{-1}(\theta)) \underbrace{\left\| \frac{\partial \hat{T}}{\partial x} \right\|}_{s^2} \underbrace{\| D \hat{T}^{-\top} n^K \|}_{\frac{1}{s}} \underbrace{\left\| \frac{\partial \xi}{\partial \theta_2} \right\|}_{\frac{1}{s} \left\| \frac{\partial x}{\partial \theta_2} \right\|} d\theta_2 \\ &= \int_0^1 f(\tilde{T}^{-1}(\theta)) \left\| \frac{\partial x}{\partial \theta_2} \right\| d\theta_2 \end{aligned} \tag{3.153}$$

The third term in the functional (3.77) involves the normal derivative of the correction function, $\partial_n D$. Once again, we transform it to the reference domain so we can use the bicubics defined over K :

$$\begin{aligned} \partial_n D &= \nabla_x D(x) \cdot n \\ &= D \hat{T}^{-\top} \nabla_{\xi} \hat{D}(\xi) \cdot D \hat{T}^{-\top} \hat{n} \\ &= \frac{1}{s} \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} \partial_{\xi} \hat{D} \\ \partial_{\eta} \hat{D} \end{pmatrix} \cdot \frac{1}{s} \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} \hat{n}_{\xi} \\ \hat{n}_{\eta} \end{pmatrix} \\ &= \frac{1}{s^2} \begin{pmatrix} \hat{D}_{\xi} \cos \phi - \hat{D}_{\eta} \sin \phi \\ \hat{D}_{\xi} \sin \phi + \hat{D}_{\eta} \cos \phi \end{pmatrix} \cdot \begin{pmatrix} \hat{n}_{\xi} \cos \phi - \hat{n}_{\eta} \sin \phi \\ \hat{n}_{\xi} \sin \phi + \hat{n}_{\eta} \cos \phi \end{pmatrix} \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{s^2} \left[(\hat{D}_\xi \cos \phi - \hat{D}_\eta \sin \phi) (\hat{n}_\xi \cos \phi - \hat{n}_\eta \sin \phi) + \right. \\
&\quad \left. (\hat{D}_\xi \sin \phi + \hat{D}_\eta \cos \phi) (\hat{n}_\xi \sin \phi + \hat{n}_\eta \cos \phi) \right] \\
&= \frac{1}{s^2} \left[\hat{D}_\xi \hat{n}_\xi \cos^2 \phi - \hat{D}_\xi \hat{n}_\eta \cos \phi \sin \phi - \hat{D}_\eta \hat{n}_\xi \sin \phi \cos \phi + \hat{D}_\eta \hat{n}_\eta \sin^2 \phi \right. \\
&\quad \left. + \hat{D}_\xi \hat{n}_\xi \sin^2 \phi + \hat{D}_\xi \hat{n}_\eta \sin \phi \cos \phi + \hat{D}_\eta \hat{n}_\xi \cos \phi \sin \phi + \hat{D}_\eta \hat{n}_\eta \cos^2 \phi \right] \\
&= \frac{1}{s^2} \nabla \hat{D}(\xi) \cdot \hat{n} \tag{3.154}
\end{aligned}$$

In (3.154), \hat{n} is the transformed normal, that is, the unit normal n in x -space transformed under \hat{T} . Specifically, the normals transform according to

$$n = D\hat{T}^{-\top} \hat{n} \tag{3.155}$$

$$\hat{n} = D\hat{T}^\top n \tag{3.156}$$

Note that n is unitary but \hat{n} is not. Equation (3.155) was used on the second line of the derivation of (3.154).

The interface integrals in (3.77) can now be rewritten using the transformed correction function, \hat{D} , noting that $\xi = \hat{T}^{-1}(x) = \hat{T}^{-1}(\tilde{T}^{-1}(\theta))$:

$$\begin{aligned}
I_2^k &:= \int_{\Gamma_k^{ij}} \left[c_2 (D(x) - a(x))^2 + c_3 (\partial_n D(x) - b(x))^2 \right] ds(x) \\
&= \int_0^1 \left[c_2 (\hat{D}(\xi) - a(\tilde{T}^{-1}(\theta)))^2 + c_3 \left(\frac{1}{s^2} \nabla \hat{D}(\xi) \cdot \hat{n} - b(\tilde{T}^{-1}(\theta)) \right)^2 \right] \left\| \frac{\partial x}{\partial \theta_2} \right\| d\theta_2 \\
&= \int_0^1 \left[c_2 \left(\hat{D}(\hat{T}^{-1}(\tilde{T}^{-1}(\theta))) - a(\tilde{T}^{-1}(\theta)) \right)^2 \right. \\
&\quad \left. + c_3 \left(\frac{1}{s^2} \nabla \hat{D}(\hat{T}^{-1}(\tilde{T}^{-1}(\theta))) \cdot \hat{n} - b(\tilde{T}^{-1}(\theta)) \right)^2 \right] \left\| \frac{\partial x}{\partial \theta_2} \right\| d\theta_2 \\
&= \int_0^1 \left[c_2 \left(\hat{D}(\hat{T}^{-1}(\tilde{T}^{-1}(\theta, \theta_2))) - a(\tilde{T}^{-1}(\theta, \theta_2)) \right)^2 \right. \\
&\quad \left. + c_3 \left(\frac{1}{s^2} \left[\hat{D}_\xi(\hat{T}^{-1}(\tilde{T}^{-1}(\theta, \theta_2))) \hat{n}_\xi + \hat{D}_\eta(\hat{T}^{-1}(\tilde{T}^{-1}(\theta, \theta_2))) \hat{n}_\eta \right] \right. \right. \\
&\quad \left. \left. - b(\tilde{T}^{-1}(\theta, \theta_2)) \right)^2 \right] \sqrt{\left(\frac{\partial H_1}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2}{\partial \theta_2} \right)^2} \Big|_{(\theta, \theta_2)} d\theta_2
\end{aligned}$$

$$\begin{aligned}
 &= \int_0^1 \left[c_2 \left(\sum_m \bar{B}_m \left(\hat{T}^{-1} \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right) \hat{D}_m - a \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right)^2 \right. \\
 &\quad \left. + c_3 \left(\frac{1}{s^2} \sum_m \left[\partial_\xi \bar{B}_m \left(\hat{T}^{-1} \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right) \hat{n}_\xi + \partial_\eta \bar{B}_m \left(\hat{T}^{-1} \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right) \hat{n}_\eta \right] \hat{D}_m \right. \right. \\
 &\quad \left. \left. - b \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right)^2 \right] \sqrt{\left(\frac{\partial H_1}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_2)} d\theta_2 \\
 &= \int_0^1 \left[c_2 \left(\sum_m \bar{B}_m \left(\hat{T}^{-1} \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right) \hat{D}_m - a \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right)^2 \right. \\
 &\quad \left. + c_3 \left(\frac{1}{s^2} \sum_m r_m \left(\hat{T}^{-1} \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right) \hat{D}_m - b \left(\tilde{T}^{-1} (\Theta, \theta_2) \right) \right)^2 \right] \\
 &\quad \times \sqrt{\left(\frac{\partial H_1}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_2)} d\theta_2
 \end{aligned} \tag{3.157}$$

where we introduced the shortcut notation

$$r_m(\xi) := \partial_\xi \bar{B}_m(\xi) \hat{n}_\xi + \partial_\eta \bar{B}_m(\xi) \hat{n}_\eta \tag{3.158}$$

Note that in equation (3.157), the transformation \tilde{T}^{-1} and thus the polynomials H_1 and H_2 from which it is constructed carry an implicit dependence on the subscript k , which indexes the transformation box M . Indeed, the transformation is defined for every cell that is crossed by the interface, whereas the integral (3.77) is computed at each node where the correction function is needed (see figures 3.7 and 3.10). This index is left out to lighten the notation.

The discretization of the integrals (3.146) and (3.157) using Gauss-Legendre quadrature and the minimization procedure described in section 3.2.4.1 leads to the linear system $2Ax + d = 0$ in equation (3.81). The matrix A and the right-hand side vector b are derived in Appendix A.

3.2.4.6 Scaling Coefficients

Here we derive the scaling coefficients c_1 , c_2 and c_3 appearing in the minimization functional (3.77). Let $\llbracket \cdot \rrbracket$ denote the physical units of some quantity, and let

$$\llbracket D \rrbracket = U \tag{3.159}$$

$$\llbracket x \rrbracket = L \quad (3.160)$$

Then we have

$$\llbracket \Delta D \rrbracket = \left\llbracket \frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2} \right\rrbracket = \frac{U}{L^2} \Rightarrow \left\llbracket \int (\Delta D - f_D)^2 dx \right\rrbracket = \left(\frac{U}{L^2} \right)^2 L^2 = \frac{U^2}{L^2} \quad (3.161)$$

$$\llbracket D \rrbracket = U \Rightarrow \left\llbracket \int (D - a)^2 ds(x) \right\rrbracket = U^2 L \quad (3.162)$$

$$\llbracket \partial_n D \rrbracket = \left\llbracket \frac{\partial D}{\partial x} n_x + \frac{\partial D}{\partial y} n_y \right\rrbracket = \frac{U}{L} \Rightarrow \left\llbracket \int (\partial_n D - b)^2 ds(x) \right\rrbracket = \left(\frac{U}{L} \right)^2 L = \frac{U^2}{L} \quad (3.163)$$

We desire that each term do not involve the dimension L , so that when the grid is refined all terms scale in the same way. We thus want

$$\llbracket c_1 \rrbracket = L^2 \quad (3.164)$$

$$\llbracket c_2 \rrbracket = 1/L \quad (3.165)$$

$$\llbracket c_3 \rrbracket = L \quad (3.166)$$

We use the integration domain side length s and the local interface length s_Γ^k , defined below, to define the scaling coefficients :

$$c_1 = s^2 \quad (3.167)$$

$$c_2 = 1/s_\Gamma^k \quad (3.168)$$

$$c_3 = s^2/s_\Gamma^k \quad (3.169)$$

where

$$s_\Gamma^k := \int_{\Gamma_k^{ij}} ds(x) = \int_0^1 \left\| \frac{\partial x}{\partial \theta_2} \right\|_{(\Theta, \theta_2)} d\theta_2 \quad (3.170)$$

3.2.4.7 New Correction Function Solver

Initially, we investigated a new way to solve the correction function problem (3.50-3.52), based on the representation formula for the Poisson problem (3.36). The idea was that since the fundamental solution $\Phi(x, y)$ is a decaying function, we could use a version of (3.36) with

spatially truncated integrals and keep the idea of a local solver. Specifically, since the data for the correction function problem corresponds to the quantities Δu , u and $\partial_n u$ appearing in (3.36), the correction function D at node x_{ij}^- inside Ω^- would have been given by

$$\begin{aligned} D(x_{ij}^-) \approx & - \int_{\Omega_{\Gamma}^{ij}} \Phi(x - x_{ij}) f^-(x) dx - \int_{\Gamma \cap \Omega_{\Gamma}^{ij}} \partial_n \Phi(x - x_{ij}) a(x) ds(x) \\ & + \int_{\Gamma \cap \Omega_{\Gamma}^{ij}} \Phi(x - x_{ij}) b(x) ds(x) \end{aligned} \quad (3.171)$$

A similar formula would have been used for nodes in Ω^+ .

To evaluate how well $D(x_{ij}^-)$ is approximated by the above formula, we evaluated the behaviour of the approximation

$$\begin{aligned} u_{\epsilon}(x) := & - \int_{\Omega \cap B_{\epsilon}(x)} \Phi(y - x) \Delta u(y) dy - \int_{\partial \Omega \cap B_{\epsilon}(x)} \partial_n \Phi(y - x) u(y) ds(y) \\ & + \int_{\partial \Omega \cap B_{\epsilon}(x)} \Phi(y - x) \partial_n u(y) ds(y) \end{aligned} \quad (3.172)$$

where $B_{\epsilon}(x)$ is the ball of radius ϵ centered at x , for a solution u of the Poisson problem (3.33-3.34). These results are shown in section 4.1.

3.2.5 Fast Poisson Solver

In this section we explain how we can efficiently solve the linear system associated with the finite difference approximation of (3.1) by exploiting the structure of the finite difference matrix. We begin by presenting the idea of the Fast Poisson solver in one dimension, before moving to the 2D case.

The main idea of the Fast Poisson solver is the following : suppose we want to solve

$$Ax = b \quad (3.173)$$

and that we know the spectral decomposition of the matrix A , i.e.

$$A = Q\Lambda Q^{-1} \quad (3.174)$$

where Q and Λ are respectively the matrix of eigenvectors and the matrix of eigenvalues of A .
Let

$$y := \Lambda Q^{-1}x \quad (3.175)$$

$$z := Q^{-1}x \quad (3.176)$$

Then we can solve the system in 3 steps :

Algorithm 3.2 Main idea of the Fast Poisson solver

1. Solve $Qy = b$ for y
 2. Solve $\Lambda z = y$ for z
 3. Solve $Q^{-1}x = z$ for x
-

Note that in step 2, the matrix Λ is diagonal so solving for z is done in $\mathcal{O}(n)$ steps, namely

$$z_k = y_k / \lambda_k, \quad k = 1, \dots, n \quad (3.177)$$

3.2.5.1 Fast Poisson Solver in 1D

In 1D, the discretization of problem (3.38-3.42) using the standard stencils (3.43-3.44) over n discretization nodes leads to the linear system

$$A_x x = b \quad (3.178)$$

where A_x is a tridiagonal matrix :

$$A_x := \frac{1}{h_x^2} \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \quad (3.179)$$

3.2. Numerical Method

The eigenvectors and eigenvalues λ_k^x of A_x can be derived using recurrence relations, leading to zeros of Chebyshev polynomials (see [18], section 4.8). We get

$$\lambda_k^x := \frac{1}{h_x^2} \left[2 - 2 \cos \left(\frac{k\pi}{n+1} \right) \right] \quad (3.180)$$

$$Q_{ik} := \sin \left(\frac{ik\pi}{n+1} \right) \quad (3.181)$$

where Q_{ik} refers to the entry (i, k) of the matrix Q , i.e. Q_{ik} is the i -th entry of the k -th eigenvector. Similar formulas are easily derived for other types of boundary conditions, such as Neumann, mixed or periodic boundary conditions (see [18]).

In step 1 of algorithm 3.2, we want to solve $Qy = b$, that is we want to find a vector y such that

$$b = \sum_{k=1}^n q_k y_k \quad (3.182)$$

where q_k is the k -th eigenvector of A_x and y_k is the k -th component of y . Since the eigenvectors of A_x are sines, (3.182) amounts to computing the discrete sine transform of the vector b . This transform is closely related to the discrete Fourier transform, and thus can be computed in $\mathcal{O}(n \log n)$ steps using the Fast Fourier transform (FFT) [8].

Similarly, in step 3, we solve $Q^{-1}x = z$, which is equivalent to

$$x = Qz \Rightarrow x = \sum_{k=1}^n q_k z_k \quad (3.183)$$

which amounts to computing the inverse discrete sine transform of z . The total work for algorithm 3.2 in 1D is thus

$$\mathcal{O}(n) + 2\mathcal{O}(n \log n) = \mathcal{O}(n \log n)$$

Note that the linear system (3.178) can be solved more efficiently using the Thomas algorithm, which computes the solution b in $\mathcal{O}(n)$ steps. The real advantage of the Fast Poisson solver thus comes into play in higher dimensions.

3.2.5.2 Fast Poisson Solver for the 5-Point Stencil

It is well known (see, for example, [23]) that discretizing the Poisson equation in 2 dimensions using the standard, second order 5-point stencil (3.53) and a lexicographic ordering of the nodes leads to a system matrix of the form

$$A_{xy}^5 := \begin{bmatrix} A_x + 2I & -I & & & \\ -I & A_x + 2I & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & A_x + 2I & -I \\ & & & -I & A_x + 2I \end{bmatrix} \quad (3.184)$$

where $A_{xy}^5 \in \mathbb{R}^{N \times N}$, $N = n_x n_y$ being the total number of nodes. Let us assume for simplicity that $n_x = n_y =: n$. A_{xy}^5 is a block tridiagonal matrix with tridiagonal blocs, and has bandwidth $w = n$. Methods for solving this linear system include banded Gaussian elimination and banded Cholesky decomposition, which both solve the system in $\mathcal{O}(Nw^2) = \mathcal{O}(n^4)$ steps. To design a 2D Fast Poisson solver, we first need to know the eigenvectors and eigenvalues of A_{xy}^5 . These are easily found by recognizing that A_{xy}^5 can be written as a Kronecker product, a matrix operation that is described next.

3.2.5.2.1 The Kronecker Product and Some Properties

Definition 3.2.1. *Kronecker product.* Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$. Then $A \otimes B \in \mathbb{R}^{mp \times nq}$ is the Kronecker product of A and B , defined as

$$A \otimes B := \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix} \quad (3.185)$$

Proposition 3.2.2. *Mixed product rule.* Let $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{p \times q}$, $C \in \mathbb{R}^{n \times r}$, and $D \in \mathbb{R}^{q \times s}$. Then

$$(A \otimes B)(C \otimes D) = AC \otimes BD \quad (3.186)$$

3.2. Numerical Method

Proof. Using block matrix multiplication, we can write the product as

$$(A \otimes B)(C \otimes D) = \underbrace{\begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}}_{m \times n \text{ blocks of size } p \times q} \underbrace{\begin{bmatrix} c_{11}D & \dots & c_{1r}D \\ \vdots & \ddots & \vdots \\ c_{n1}D & \dots & c_{nr}D \end{bmatrix}}_{n \times r \text{ blocks of size } q \times s} = \underbrace{\begin{bmatrix} E_{11} & \dots & E_{1r} \\ \vdots & \ddots & \vdots \\ E_{m1} & \dots & E_{mr} \end{bmatrix}}_{m \times r \text{ blocks of size } p \times s}$$

where each block E_{ij} is defined as

$$\begin{aligned} E_{ij} &= \sum_{k=1}^n a_{ik} B c_{ki} D \\ &= \sum_{k=1}^n a_{ik} c_{ki} B D \\ &= (AC)_{ij} B D \end{aligned}$$

which we can write as $AC \otimes BD$ according to definition 3.2.1. □

Definition 3.2.3. *Kronecker sum.* Let $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{m \times m}$, and let I_r denote the identity matrix of size $r \times r$. The Kronecker sum of A and B , denoted $A \oplus B$, is defined as

$$A \oplus B := A \otimes I_m + I_n \otimes B \quad (3.187)$$

Proposition 3.2.4. Let $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{m \times m}$, and let (λ_i, u_i) and (μ_j, v_j) respectively denote the eigenpairs of A and B , i.e.

$$A u_i = \lambda_i u_i, \quad i = 1, \dots, n$$

$$B v_j = \mu_j v_j, \quad j = 1, \dots, m$$

and let I_r denote the identity matrix of size $r \times r$. Then

$$(A \otimes B)(u_i \otimes v_j) = \lambda_i \mu_j (u_i \otimes v_j) \quad (a)$$

$$(A \oplus B)(u_i \otimes v_j) = (\lambda_i + \mu_j)(u_i \otimes v_j) \quad (b)$$

The result (a) means that the eigenvalues of the Kronecker product of A and B are the product

of the eigenvalues of A and those of B , and the eigenvectors of the product are the Kronecker product of their eigenvectors. The result (b) means that the eigenvalues of the Kronecker sum of A and B are the sum of the eigenvalues of A and those of B , and the eigenvectors of the sum are the Kronecker product of their eigenvectors.

Proof. (a) Using the mixed product rule 3.2.2, we can write

$$(A \otimes B)(u_i \otimes v_j) = Au_i \otimes Bv_j = \lambda_i u_i \otimes \mu_j v_j = \lambda_i \mu_j (u_i \otimes v_j)$$

(b) Again using the mixed product rule and the definition 3.2.3, we can write

$$(A \otimes I_m)(u_i \otimes v_j) = Au_i \otimes I_m v_j = \lambda_i \mu_j \otimes v_j = \lambda_i (u_j \otimes v_j) \quad (i)$$

$$(I_n \otimes B)(u_i \otimes v_j) = I_n u_i \otimes Bv_j = u_i \otimes \mu_j v_j = \mu_j (u_j \otimes v_j) \quad (ii)$$

The result (b) is recovered by summing (i) and (ii). \square

The eigenvalues and eigenvectors of A_{xy}^5 can now be found by recognizing that

$$A_{xy}^5 = A_y \oplus A_x = A_y \otimes I_{n_x} + I_{n_y} \otimes A_x \quad (3.188)$$

where A_y is defined similarly to (3.179). Then (b) implies the eigenvalues and eigenvectors of A_{xy}^5 are given by

$$\lambda_{kl}^{xy,5} := \lambda_k^x + \lambda_l^y \quad (3.189)$$

$$Q_{ij,kl}^5 := \sin\left(\frac{ik\pi}{n_x + 1}\right) \sin\left(\frac{jl\pi}{n_y + 1}\right) \quad (3.190)$$

where $Q_{ij,kl}^5$ refers to the entry (i, j) of the eigenvector (k, l) and λ_l^y is defined similarly as (3.180). This product of sines implies that (3.182) and (3.183) correspond to the 2D discrete sine transform, which can be computed very efficiently using the 2D FFT. This means that the linear system can be solved in $\mathcal{O}(N \log N)$ steps, which for $n_x = n_y = n$ is $\mathcal{O}(n^2 \log n^2)$, a considerably cheaper cost than the traditional methods. Note that in this approach the finite difference matrix A_{xy}^5 is not used; it does not have to be neither stored nor computed.

3.2.5.3 Fast Poisson Solver for the 9-Point Stencil

We are now in a position to describe the Fast Poisson solver for the 9-point stencil. Starting from results in [29], it can be shown that the system matrix associated with the 9-point stencil (3.61), A_{xy}^9 , can be written

$$\begin{aligned} A_{xy}^9 &= A_{xy}^5 + \frac{h_x^2 + h_y^2}{12} (A_y \otimes A_x) \\ &= A_y \oplus A_x + \frac{h_x^2 + h_y^2}{12} (A_y \otimes A_x) \end{aligned} \quad (3.191)$$

Immediately, (a) implies the eigenvalues of the cross term $A_y \otimes A_x$ are $\lambda_{kl}^{xy, \text{cross}} := \lambda_k^x \lambda_l^y$, and its eigenvectors are given by (3.190); they are the same as those of A_{xy}^5 . This means that if v is an eigenvector, we can write

$$\begin{aligned} A_{xy}^9 v &= \left[A_y \oplus A_x + \frac{h_x^2 + h_y^2}{12} (A_y \otimes A_x) \right] v \\ &= (A_y \oplus A_x) v + \frac{h_x^2 + h_y^2}{12} (A_y \otimes A_x) v \\ &= \lambda_{kl}^{xy, 5} v + \frac{h_x^2 + h_y^2}{12} \lambda_{kl}^{xy, \text{cross}} v \\ &= \left[\lambda_{kl}^{xy, 5} + \frac{h_x^2 + h_y^2}{12} \lambda_{kl}^{xy, \text{cross}} \right] v \end{aligned}$$

and the eigenvalues of A_{xy}^9 are thus given by

$$\lambda_{kl}^{xy, 9} := \lambda_k^x + \lambda_l^y + \frac{h_x^2 + h_y^2}{12} \lambda_k^x \lambda_l^y \quad (3.192)$$

Having determined the eigenvalues and eigenvectors of A_{xy}^9 , we can use algorithm 3.2 and the 2D Fast Fourier transform to efficiently solve the linear system associated with the 9-point stencil.

3.3 Code Implementation

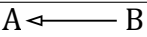
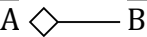
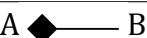
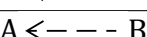
The method described in section 3.2 was first implemented in code using the MATLAB programming language. A second version of the code was made using the C++ programming language. They are described in the next sections. Both versions are very similar but have

a few differences. Both codes were written using an object-oriented design. In software engineering, the object-oriented programming paradigm uses what are called *classes* to model application-level abstractions that are used as user-defined types. The actual instances of a class, used in the code, are called objects. The two versions of the code use a very modular design that can be easily extended.

3.3.1 MATLAB Code

The MATLAB version of the code was developed first and is called CFM2D. A diagram representing the principal classes of the code is shown in figure 3.13. This diagram uses the Unified Modeling Language (UML) [21] to indicate the relationships between the different classes. UML is an ISO standard used to graphically represent the design of a system. Four different relationship types are used in the figure, and are explained in table 3.2. The role of the different classes is explained below.

Table 3.2. Four UML relationships

Symbol	Name	Meaning
A  B	Inheritance	B is a specialization of class A
A  B	Aggregation	A contains an object of class B
A  B	Composition	A contains an object of class B and manages its lifetime
A  B	Dependency	A uses an object of class B

3.3.1.1 Main Solver : CFM2D and CFM2DOrder4Compact

The class CFM2D is the main class of the code. It is the primary user interface of the code, with member functions to solve the Poisson interface problem and compute the gradient of the solution once the latter is found. However, CFM2D is an abstract class. Its member functions are declared but are not implemented in the class. They are implemented in the derived class CFM2DOrder4Compact, which is a specialization of the solver in the case of the compact 4th order scheme presented in section 3.2.2. This modularity enables the code to be extended with minimal modification if a version using a different finite difference stencil needs to be written. The class CFM2DOrder4Compact is also responsible for setting up the finite difference stencil, computing the right-hand side of the linear system (without corrections) and applying the Dirichlet boundary conditions.

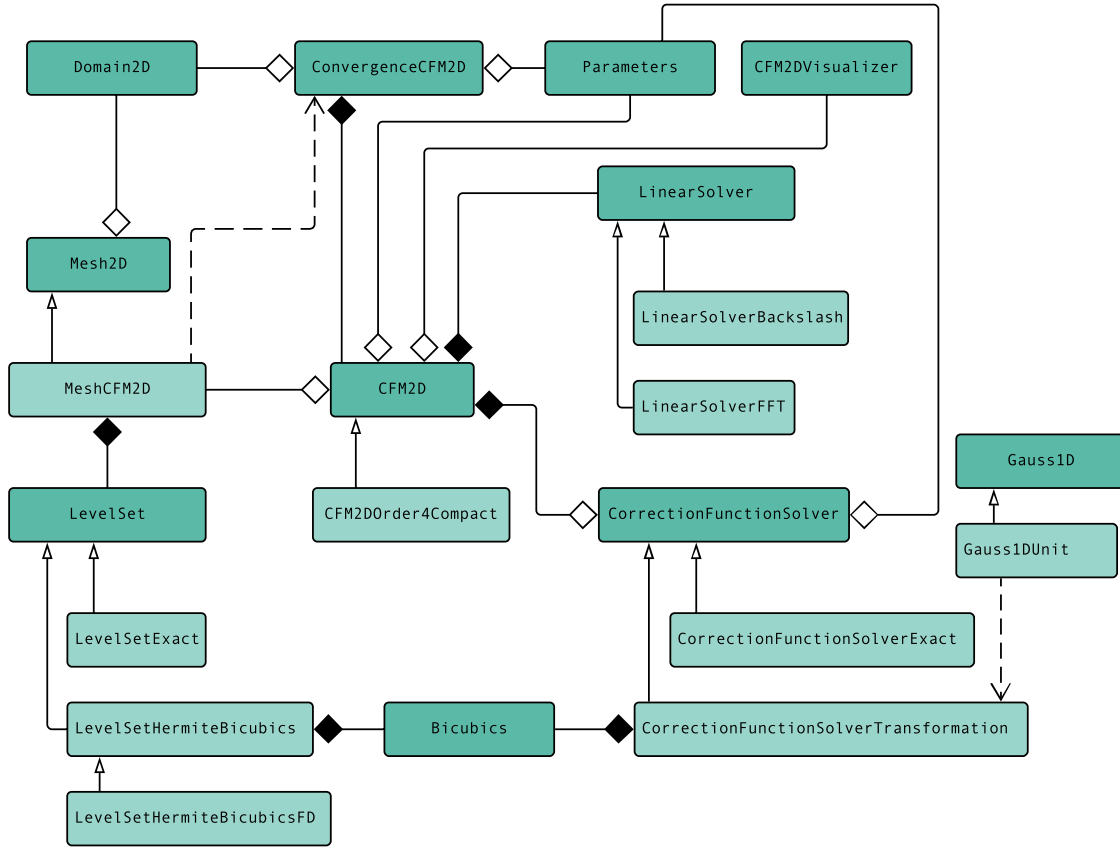


Figure 3.13. Class diagram for the MATLAB version of the code. The class CFM2D, in the center of the diagram, is the main class of the code.

3.3.1.2 Domain Discretization : Domain2D, Mesh2D and LevelSet

The code uses a rectangular domain aligned with the computational grid. The continuum domain is described by the simple class `Domain2D`. It is passed to the class `Mesh2D`, which is responsible for the domain discretization. The derived class `MeshCFM2D`, which is actually used in the code, is, moreover, responsible for the discretization of the two subdomains Ω^+ and Ω^- . Also, it is used to determine which cells are crossed by the interface and which nodes are close enough to the interface to require a correction. This is done with the help of the abstract class `LevelSet`. This class models the level set function ϕ used to define Ω^+ and Ω^- and is used to compute the level set and related computations : the normal and tangent vectors to the interface and the projection operation described in algorithm 3.1. Three actual implementations of `LevelSet` are provided : `LevelSetExact`, `LevelSetHermiteBicubics` and `LevelSetHermiteBicubicsFD`. The class `LevelSetExact` uses the exact expression of the level set to compute its value at arbitrary location on the grid. `LevelSetHermiteBicubics`

receives the exact expression of the level set and its first derivatives, but only values at grid nodes are used. The level set values at arbitrary grid locations are computed using reduced bicubic interpolation, as described in sections 3.2.3 and 3.2.4.2. The bicubic interpolants are implemented in the class `Bicubics`. Finally, `LevelSetHermiteBicubicsFD` receives only the exact expression of the level set, and computes the gradient at grid nodes using 4th order finite difference stencils. This approach guarantees 4th order convergence, as discussed above.

3.3.1.3 Correction Function Solver

The correction function problem is solved by the abstract class `CorrectionFunctionSolver`. The derived class `CorrectionFunctionSolverExact` was used for testing during development and simply computes the exact expression of the correction function, $D = u^+ - u^-$. The method described in section 3.2.4 is implemented in the class `CorrectionFunctionSolverTransformation`. This class, as well as the rest of the code, fully uses the vectorization capabilities of MATLAB. The data needed to construct the linear systems (3.131) is computed at the same time for every crossed cell; the integration data is transformed under both transformations T and \hat{T} for every node requiring a correction and for every crossed cell all at the same time, etc. The only places loops are used are when we actually solve the systems (3.131) for every crossed cells and when we solve the minimization system (3.82) for every node requiring a correction (the construction of the systems, however, is vectorized). The 1D Gauss-Legendre integration nodes and weights are computed on demand on the interval $[-1, 1]$ by the class `Gauss1D`, using a matrix approach taken from [47]. They are transformed to the unit interval in the derived class `Gauss1DUnit`. Once the correction function is known, the class `CorrectionFunctionSolver` is also responsible for applying the corrections to the right-hand side of the linear system.

3.3.1.4 Visualization and Convergence

The class `CFM2DVisualizer` is responsible for plotting the solution and its gradient and is used to create the plots presented in the Chapter 4. The class `ConvergenceCFM2D` is responsible for running the code several times while refining the grid, in order to verify the accuracy of the method.

3.3.1.5 Linear Solver

The abstract class `LinearSolver` is responsible for solving the linear system once all corrections are applied to the right-hand side. Two implementations are provided: `LinearSolverFFT` implements the fast Poisson solver for the 9-point stencil derived in section 3.2.5.3, whereas `LinearSolverBackslash` uses the MATLAB backslash operator (`\`) to solve the system. This solver determines the best method to solve the system depending on the structure of the coefficient matrix. In the case of the symmetric matrix associated with the 9-point stencil, MATLAB uses the MA57 algorithm described in [13], which is based on the LDL' factorization (see, for example, [18]).

3.3.1.6 Parameters and Problem Description

The code uses a configuration file in the INI format to manage the choice of different options, such as the implementation to use for the classes `CorrectionFunctionSolver`, `LinearSolver` and `LevelSet`. This permits us to test the numerical method with different implementation options without any changes to the code. The configuration file also holds the limits of the domain, the number of nodes for its discretization and the exact expressions for the functions defining the problem: f^+ , f^- , a , b , ϕ , g and the exact solution u^+ , u^- . We use the method of manufactured solutions to be able to compute the error in the computed solution and verify the fourth order convergence of the method. The configuration file is read by the class `Parameters`, which is passed to the other classes in order to make the chosen options available to them. The class `Parameters` is also responsible for transforming the textual representation of the functions describing the problem into MATLAB function handles. We use the capabilities of the Symbolic Math Toolbox in MATLAB to compute the exact derivatives of these functions that are needed in the code, such as the derivatives of the source term (see equations (3.62) and (3.71)).

3.3.2 C++ Code

A second version of the code was written in C++. The reason a second version was made is to use the code in conjunction with another code written in C by Jean-Christophe Nave, called `bubbles`, which solves the Navier-Stokes problem described in section 3.4. The C++

version of the CFM code is called `cfm-petsc`. It uses the numerical library PETSc ([3], [4], [5]). This library is used as it provides classes for storing vectors and matrices and also provides several linear algebra solvers. Moreover, it provides functions to efficiently handle the domain discretization on a structured Cartesian grid. The class diagram for `cfm-petsc` is presented in figure 3.14. Most of the classes have the same name and purpose as those in CFM2D, but a few differences are noted below.

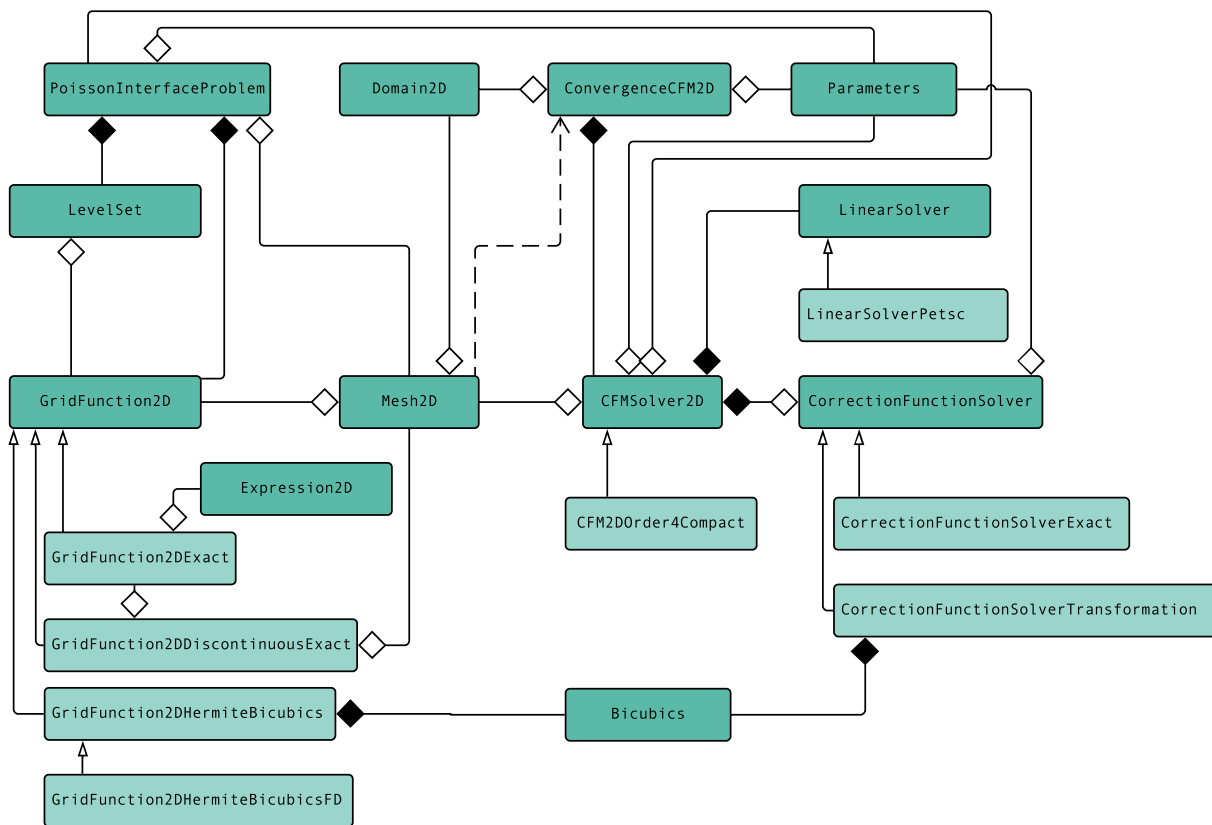


Figure 3.14. Class diagram for the C++ version of the code. The class `CFMSolver2D`, in the center of the diagram, is the main class of the code.

The main difference between the two versions of the code is the introduction of the class `GridFunction2D`. This class represents a mathematical function defined over a computational grid. Four versions are implemented, `GridFunction2DExact`, `GridFunction2DHermiteBicubics`, `GridFunction2DHermiteBicubicsFD` and `GridFunction2DDiscontinuousExact`. The first three correspond to the three implementations of the level set described above. However, the class `GridFunction2D` is not used only for the level set; every

3.3. Code Implementation

function in the problem definition $(f^+, f^-, a, b, \phi, g)$ is modeled using a `GridFunction2D` object. This permits us to receive all problem data on a grid instead of using exact expressions. This is essential when interfacing with other software (such as the code `bubbles`), for which the problem data might be known only at grid nodes. The class `GridFunction2DDiscontinuousExact` is used to model a discontinuous function, such as the source term f . For the class `GridFunction2DExact`, which was used for testing during development, we use the C++ library `Lepton` [40] to parse the mathematical expressions in the configuration file and evaluate them.

All problem data is managed by the class `PoissonInterfaceProblem`, which is passed to the solver `CFMSolver2D` and the convergence driver `ConvergenceCFM2D`. Note also that the class `Parameters`, which has a similar role as in `CFM2D`, allows changing different options at runtime, without any need for recompilation of the code.

Another difference between the codes is the fact that `cfm-petsc` handles both Dirichlet and periodic boundary conditions, in order for it to be used alongside `bubbles`. In both cases, the matrix associated with the finite difference stencil is solved using the iterative linear solvers provided by PETSc, which can be chosen at runtime. By default, PETSc uses the GMRES method. All linear solvers provided by PETSc can solve a singular system if the null space is known. As mentioned in section 3.4, for the Navier-Stokes simulation we use periodic boundary conditions, which lead to a singular linear system matrix. However, its null space is known; it contains all constant functions, reflecting the fact that any solution is unique up to an additive constant. The PETSc solvers can solve singular systems by removing the null space component of the solution at each iteration. The class `LinearSolverFFT` is not present in `cfm-petsc`.

Note that one of the main features of the `Petsc` library is to handle distributed linear algebra and grids with the help of the Message Passing Interface (MPI) standard [11]. This is one of the reasons this library was chosen over others with similar features: to make the code scalable. However, although we made all efforts to write the code with distributed parallelism in mind, `cfm-petsc` was not tested in the distributed case.

3.4 Towards a Correction Function Method for the Navier-Stokes Equations

As stated in the introduction, our main motivation in this work is the Poisson interface problem appearing in the numerical solution of the Navier-Stokes equations of fluid dynamics. In this section we describe the Navier-Stokes problem (Section 3.4.1) and its numerical solution using the projection method (Section 3.4.2).

3.4.1 Problem Description

The Navier-Stokes equations are a system of partial differential equations used to model the movement of viscous fluids. They relate the fluid velocity, $u : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}^d$, where d is the spatial dimension, and the pressure of the fluid $p : \Omega \rightarrow \mathbb{R}$. For the case of incompressible flow, for which the fluid density ρ is constant in a fluid parcel moving with the flow, they are given by

$$\partial_t u + u \cdot \nabla u = \frac{-\nabla p}{\rho} + \nu \Delta u + f \quad (3.193)$$

$$\nabla \cdot u = 0 \quad (3.194)$$

where ν is the kinematic viscosity of the fluid, a physical property of the fluid, ∇u is the covariant derivative of the vector field u [46] and f are body forces acting on the fluid, for example the gravitational force. The convective acceleration term $u \cdot \nabla u$ can be shown to be

$$u \cdot \nabla u = u \times (\nabla \times u) + \frac{1}{2} \nabla (u \cdot u) \quad (3.195)$$

which in Cartesian coordinates reduces to

$$u \cdot \nabla u = \sum_{i=1}^d (u \cdot \nabla u_i) \hat{x}_i \quad (3.196)$$

3.4. Towards a Correction Function Method for the Navier-Stokes Equations

where u_i is the i -th component of u and \hat{x}_i is the unit vector in the i -th dimension. The vector Laplacian Δu is defined by

$$\Delta u := \nabla (\nabla \cdot u) - \nabla \times (\nabla \times u) \quad (3.197)$$

In Cartesian coordinates, this reduces to

$$\Delta u = \sum_{i=1}^d \Delta u_i \hat{x}_i \quad (3.198)$$

The modeling of multiphase flow falls within the range of problems for which the Correction Function Method would be useful. Indeed, in multiphase flow the material properties (the viscosity and the density) of the different fluids in contact are usually discontinuous at the interface between the fluids, as is the pressure field. Here we examine the simpler case of monophasic flow with pressure discontinuity. This would model, for example, thin soap bubbles floating in air.

In this case, equation (3.193) is coupled to an advection equation describing the evolution of the level set function ϕ representing the interface :

$$\partial_t \phi + u \cdot \nabla \phi = 0 \quad (3.199)$$

with the interface Γ still represented by (3.11). Equation (3.199) indicates that the interface moves with the local velocity of the fluid. In [22], it is shown that applying physical conservation laws to the Navier-Stokes interface problem leads to a discontinuous pressure along the interface with jump conditions given by

$$\begin{aligned} [p] &= \sigma \kappa \\ [\partial_n p] &= 0 \end{aligned} \quad (3.200)$$

where σ is the surface tension coefficient, a physical property of the fluid, and κ is the local interface curvature. These conditions mean that the net stress acting on the interface is zero since it has no mass [22]. The curvature can be computed from the level set function as

$$\kappa = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right) \quad (3.201)$$

Appropriate boundary conditions for the velocity and pressure must be used to complete the system. In this work, for simplicity we use periodic boundary conditions.

3.4.2 Numerical Method

To solve the coupled equations (3.193, 3.194, 3.199, 3.200), the projection method of Chorin can be used. As outlined in [22], algorithm 3.3 describes one time step of the method.

Algorithm 3.3 Projection method for Navier-Stokes with pressure discontinuity

1. Compute tentative velocity u^* :

$$\frac{u^* - u^n}{\Delta t} = -u^n \cdot \nabla u^n + \nu \Delta u^n + f \quad (3.202)$$

2. Solve for pressure :

$$\frac{\Delta p}{\rho} = \frac{\nabla \cdot u^*}{\Delta t} \quad (3.203)$$

3. Update velocity :

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{-\nabla p}{\rho} \quad (3.204)$$

4. Advect level set :

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = -u^n \cdot \nabla \phi^n \quad (3.205)$$

Steps 1-3 correspond to the projection method, in which the divergence-free condition (3.194) is enforced by choosing the pressure that insures a divergence-free solution. This corresponds to a projection of the tentative velocity u^* onto the space of divergence-free velocities. In step 2, we solve a Poisson problem for the pressure, subject to the jump conditions (3.200). This is exactly the Poisson interface problem described in section 3.1.1. In [22], this problem is solved using the Ghost Fluid Method. The projection method is first-order in time [10]. The actual time-stepping algorithm used in [22] is a third order Runge-Kutta TVD (total variation diminishing) scheme [45].

The algorithm above is applied on a marker-and-cell (MAC) grid, which is a staggered grid that was introduced in [20]. A grid cell is illustrated in figure 3.15. For stability reasons, the two components of the velocity (u and v) and the scalar quantities (ϕ and p) are not computed at the same location.

The spatial differential operators appearing in the equations of algorithm 3.3 (the gradient, divergence and Laplacian) are usually discretized using second order finite differences, as is

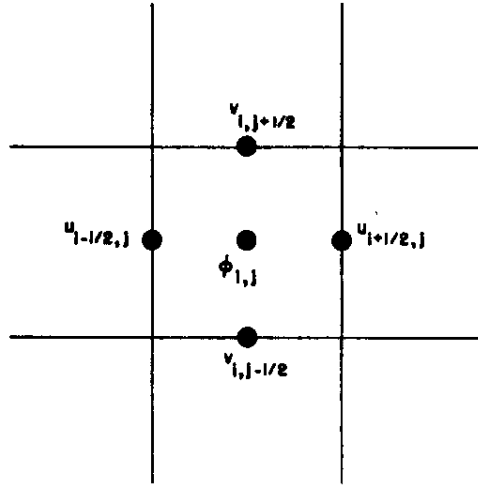


Figure 3.15. One cell of the MAC grid. The velocity components are computed on the cell edges, and the scalar quantities (pressure and level set) are computed at the cell centers.

done in [22]. It is important to note that these second order discrete approximations satisfy discrete versions of the classical vector calculus identities exactly. This mimetic discretization contributes to the stability of the scheme [49]. The discretization of the level set equation uses the WENO scheme of [17].

If a higher order scheme for the Navier-Stokes problem with pressure discontinuity was to be designed, the Correction Function Method would be the method of choice for step 2 of the algorithm, since this method can easily handle high-order finite difference schemes. As a first step in this direction, we used the code `bubbles`, developed by Jean-Christophe Nave, along with `cfm-petsc` for the 2nd step, to implement algorithm 3.3.

4. Results

In this chapter, we present different results that validate our implementation of the Correction Function Method for the Poisson interface problem. We start by the test of the representation formula approach for the solution of the correction function problem presented in section 3.2.4.7. Next, we apply the CFM method described in section 3.2 to four different manufactured problems. The data for the manufactured problems is computed exactly with Maple [30] using the following algorithm :

Algorithm 4.1 Manufactured solution

Require: Exact solution u^+, u^- and level set ϕ

1. Compute $f^+ = \Delta u^+, f^- = \Delta u^-$
 2. Compute $f_D = f^+ - f^-$
 3. Compute $a = D = u^+ - u^-$
 4. Compute $n = \frac{\nabla \phi}{\|\nabla \phi\|}$
 5. Compute $b = \nabla(u^+ - u^-) \cdot n$
-

Following what as done in [31], all results presented use a value of $c_p = 50$ for the penalization coefficient of equation (3.77) and 6 Gauss-Legendre points for the discretization of the integrals.

We do not present simulation results for the Navier-Stokes equations with pressure discontinuity introduced in section 3.4, as our adaptation of the code `bubbles` to use the CFM implementation `cfm-petsc` for step 2 of algorithm 3.3 did not yield physically meaningful results. This is discussed in the next chapter.

4.1 Representation Formula Approximation

In order to validate the use of the representation formula (3.171) for the solution of the correction function problem (3.50-3.52), we investigated the convergence of the approximation

4.1. Representation Formula Approximation

(3.172). We set $\Omega = B_1(0)$, $x = (-0.5, -0.6)$ and computed $u_\epsilon(x)$ for different values of ϵ ranging from 0.01 to 1.8. At $\epsilon = 1.8$, the domain Ω is completely contained in $B_\epsilon(x)$, so $u_\epsilon(x) = u(x)$ as given by (3.36). The domain and the ball $B_\epsilon(x)$ are shown in figure 4.1 for $\epsilon = 0.1$.

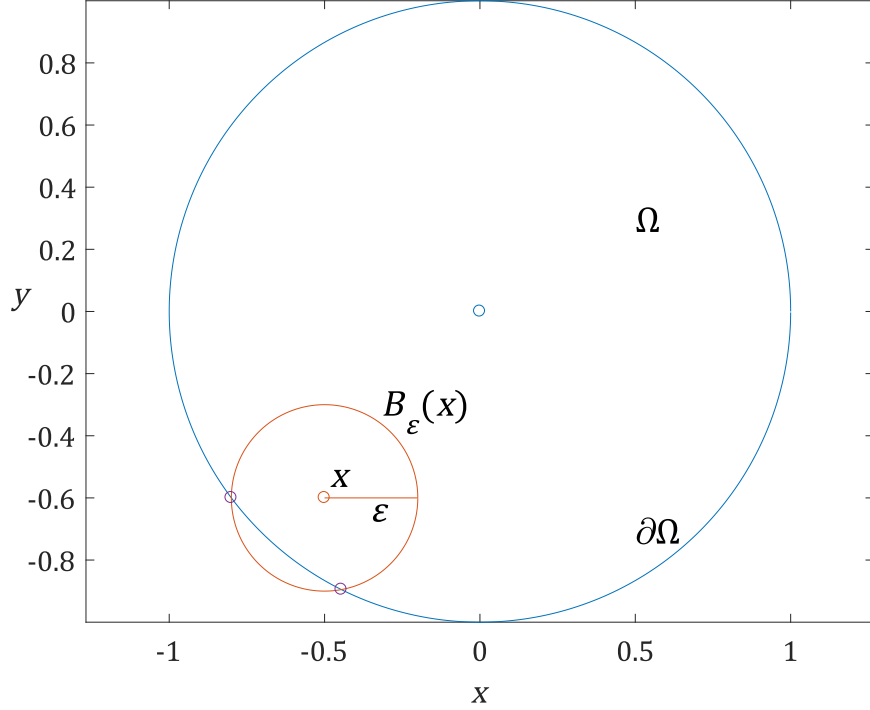


Figure 4.1. The domain Ω used to validate the approximation $u_\epsilon(x)$. The second and third terms in (3.172) are integrated on the segment of $\partial\Omega$ between the two purple circles.

To test the approximation, we used 4 different solutions of the Poisson problem (3.33) :

$$u(x, y) = \sin(\pi x) \sin(\pi y) \quad (4.1)$$

$$u(x, y) = \exp\left(\frac{-1}{1 - x^2 - y^2}\right) \quad (4.2)$$

$$u(x, y) = 1 - x^2 - y^2 \quad (4.3)$$

$$u(x, y) = 1 - x - y \quad (4.4)$$

To numerically evaluate the integrals in (3.172), we use the MATLAB functions `integral` and `integral2` ([44], [43]) which are designed to handle singular integrands such as the fundamental solution $\Phi(x, y)$. The results are shown in figure 4.2. They show the absolute value of the error $E = u - u_\epsilon$ as a function of ϵ .

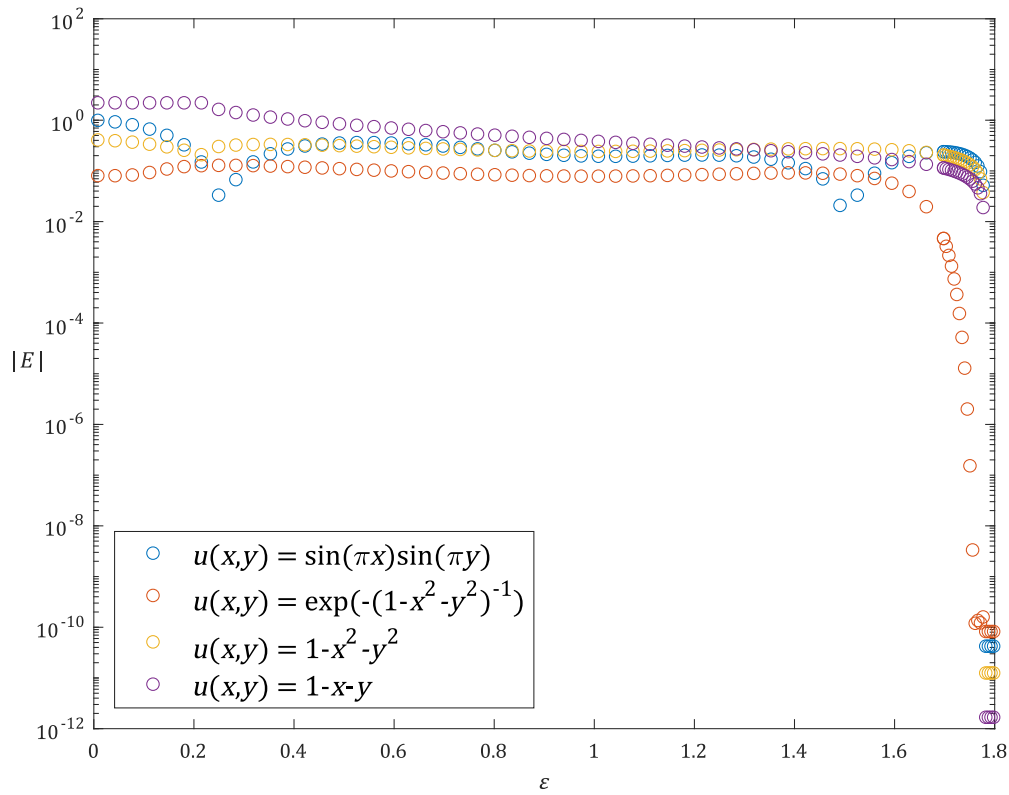


Figure 4.2. Test of the representation formula approach for different solutions to the Poisson problem.

We observe that the approximation u_ϵ is not converging to the real solution at all until the ball B_ϵ almost completely covers Ω , approximately at $\epsilon = 1.7$. These results show that the representation formula approach is not effective to locally solve the correction function problem; the decay of the fundamental solution Φ is not strong enough to be able to truncate the integrals in (3.36) and thus we cannot compute the solution D of the correction function problem using (3.171).

4.2 Example 1

4.2.1 Problem Definition

As a first example, we reproduce Example 1 from [31], in which the interface is a circle. The problem parameters are given below :

$$\begin{aligned}
 \Omega &= [0, 1]^2 \\
 f^+(x, y) &= -2\pi^2 \sin(\pi x) \sin(\pi y) \\
 f^-(x, y) &= -2\pi^2 \sin(\pi x) \sin(\pi y) \\
 \phi(x, y) &= (x - x_0)^2 + (y - y_0)^2 - r_0^2, \quad x_0 = y_0 = 0.5, \quad r_0 = 0.1 \\
 f_D(x, y) &= 0 \\
 a(x, y) &= \sin(\pi x) \exp(\pi y) \\
 b(x, y) &= \pi \exp(\pi y) \frac{(2y - 1) \sin(\pi x) + (2x - 1) \cos(\pi x)}{\sqrt{4x^2 + 4y^2 - 4x - 4y + 2}}
 \end{aligned} \tag{E1}$$

Notice both a and b are non-zero but f_D is zero. The exact solution is given by

$$\begin{aligned}
 u^+(x, y) &= \sin(\pi x) \sin(\pi y) \\
 u^-(x, y) &= \sin(\pi x) (\sin(\pi y) - \exp(\pi y))
 \end{aligned}$$

4.2.2 Numerical Solution

The computed solution is shown in figure 4.3. We see the interface jump is very clean and causes no oscillations of the solution.

The error $E = |u_{\text{exact}} - u_{\text{approx}}|$ is shown in figure 4.4. We see the error is larger by about two orders of magnitude in the vicinity of the interface.

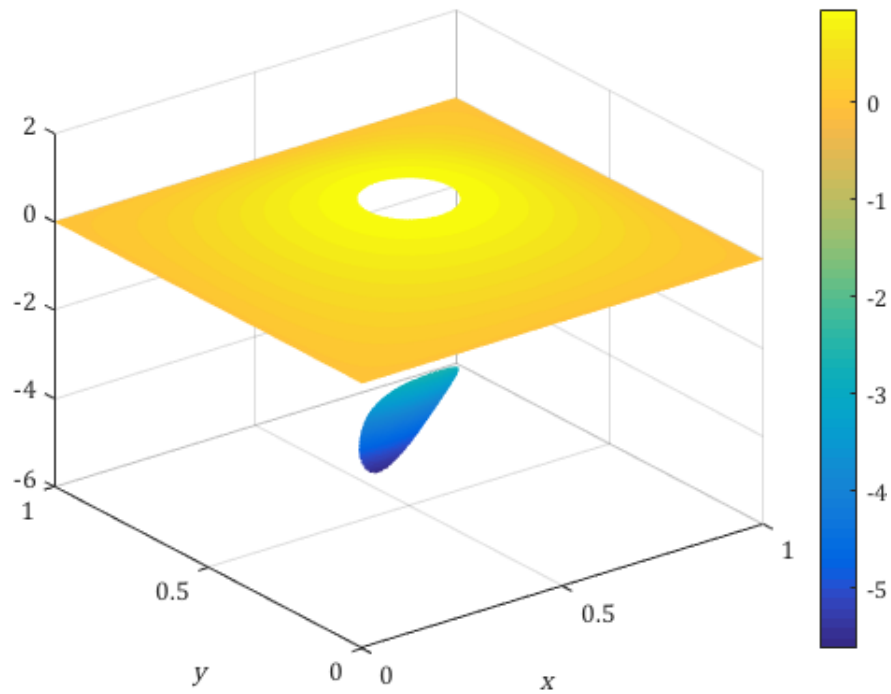


Figure 4.3. Computed solution of problem E1 on a 500×505 grid. On this grid, 804 nodes require a correction and 404 cells are crossed by the interface.

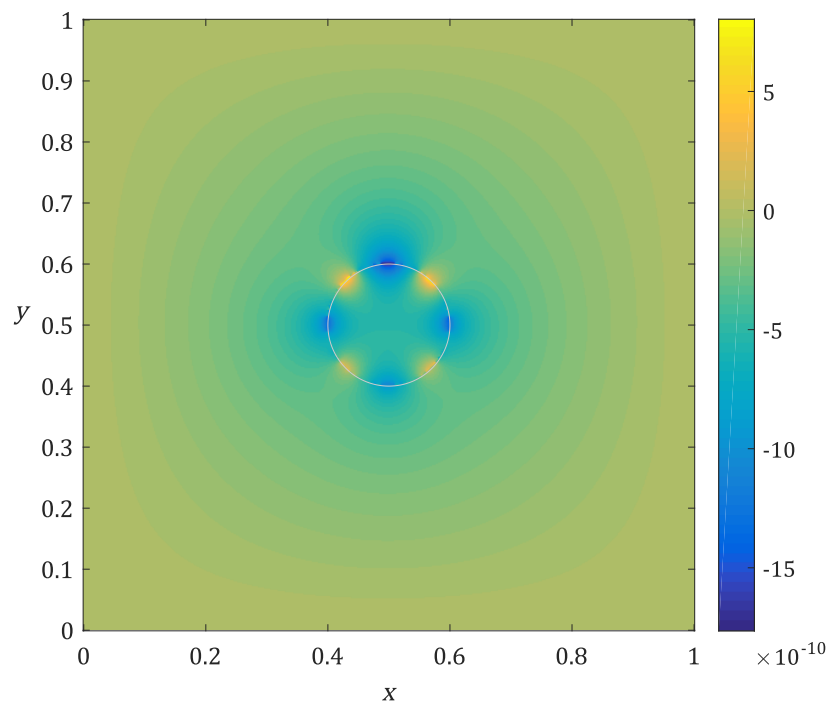


Figure 4.4. Error in the computed solution of example 1. The error is larger near the interface, which appears in grey.

4.2.3 Convergence

Figure 4.5 shows the convergence of the solution, in the L^1 , L^2 and L^∞ norms. These norms are discrete norms defined for a grid function u as

$$\|u\|_{L^p} = \begin{cases} \Delta x \Delta y \left(\sum_{i=1}^{n_x} \sum_{j=1}^{n_y} |u_{ij}|^p \right)^{1/p} & p \geq 1 \\ \max_{ij} |u_{ij}| & p = \infty \end{cases} \quad (4.5)$$

The horizontal axis is the step size h , defined as $h = \sqrt{h_x^2 + h_y^2}$. All orders of convergence presented here are obtained using a linear regression approach. The solution converges to 4th order in the L^∞ , and to order 4.3 in the L^2 and L^1 norms. In figure 4.6, we show the convergence of the correction function. Since we use a manufactured solution, we can easily compute the exact correction function $D = u^+ - u^-$ and thus we can compute the error in the computed correction function. We observe a convergence of order 5, 4.6 and 3.9 respectively in the L^1 norm, L^2 and L^∞ norms. Finally, figure 4.7 show the convergence of the gradient of the solution. It converges to order 3 in the L^∞ norm, close to order 4 in the L^1 norm and to order 4.2 in the L^2 norm.

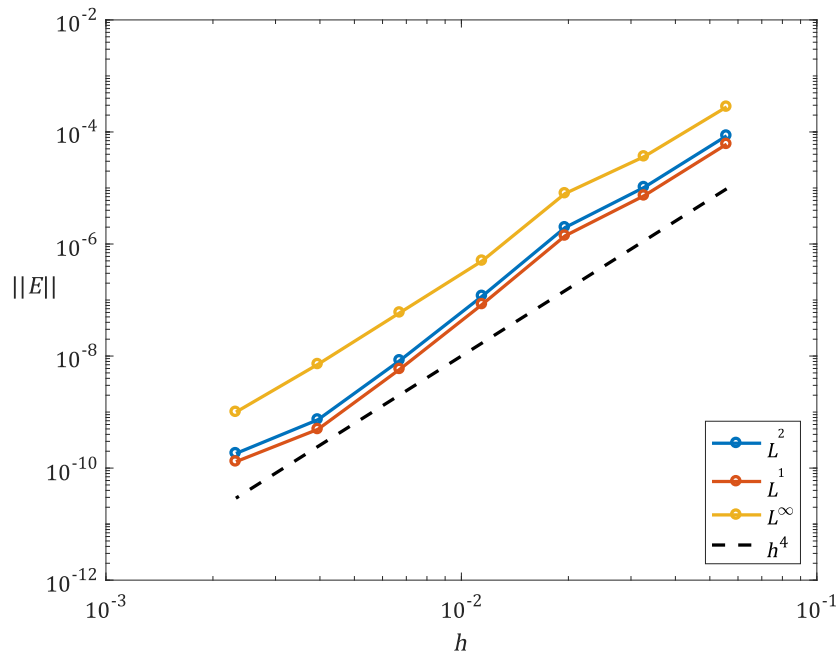


Figure 4.5. Convergence of the solution for example 1

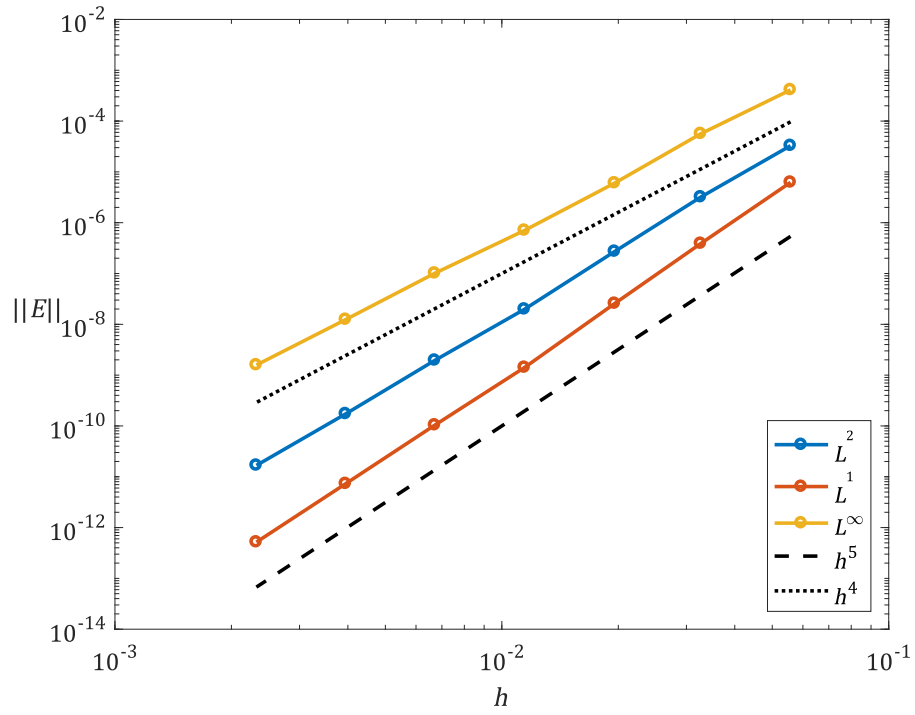


Figure 4.6. Convergence of the correction function for example 1

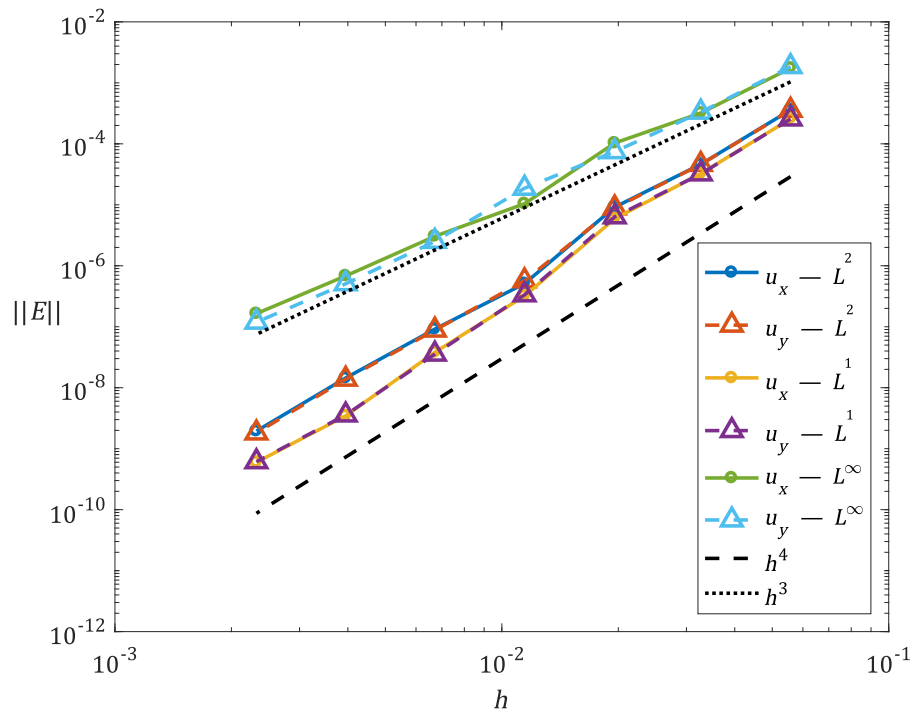


Figure 4.7. Convergence of the solution gradient for example 1

4.3 Example 2

4.3.1 Problem Definition

The second example corresponds to Example 2 from [31]. In this problem the interface is more complex ; it is a smooth 5-pointed star. The problem is defined by :

$$\begin{aligned}
 \Omega &= [0, 1]^2 \\
 f^+(x, y) &= 0 \\
 f^-(x, y) &= 0 \\
 \phi(x, y) &= (x - x_0)^2 + (y - y_0)^2 - \left[r_0 + \epsilon \sin \left(k \arctan \left(\frac{y - y_0}{x - x_0} \right) \right) \right]^2, \\
 x_0 &= y_0 = 0.5, \quad r_0 = 0.25, \quad k = 5, \quad \epsilon = 0.05 \\
 f_D(x, y) &= 0 \\
 a(x, y) &= -\exp(x) \cos(y) \\
 b(x, y) &= -\exp(x) (\cos(y)n_x - \sin(y)n_y)
 \end{aligned} \tag{E2}$$

We use the Cartesian components of the normal (n_x, n_y) to simplify the expression of the function b . Once again in this problem we have $a \neq 0$, $b \neq 0$ and $f_D = 0$. The exact solution is given by

$$\begin{aligned}
 u^+(x, y) &= 0 \\
 u^-(x, y) &= \exp(x) \cos(y)
 \end{aligned}$$

4.3.2 Numerical Solution

The computed solution of problem E2 is shown in figure 4.8. Once again, the jump is very sharp. The error in the solution appears in figure 4.9. Again, the error is largest near the interface. Figure 4.10 shows a close-up of the error in the correction function on Ω_F at a location where the error in the solution is larger. We see that the error in the computed correction function is not smooth, which explains the larger error in the solution.

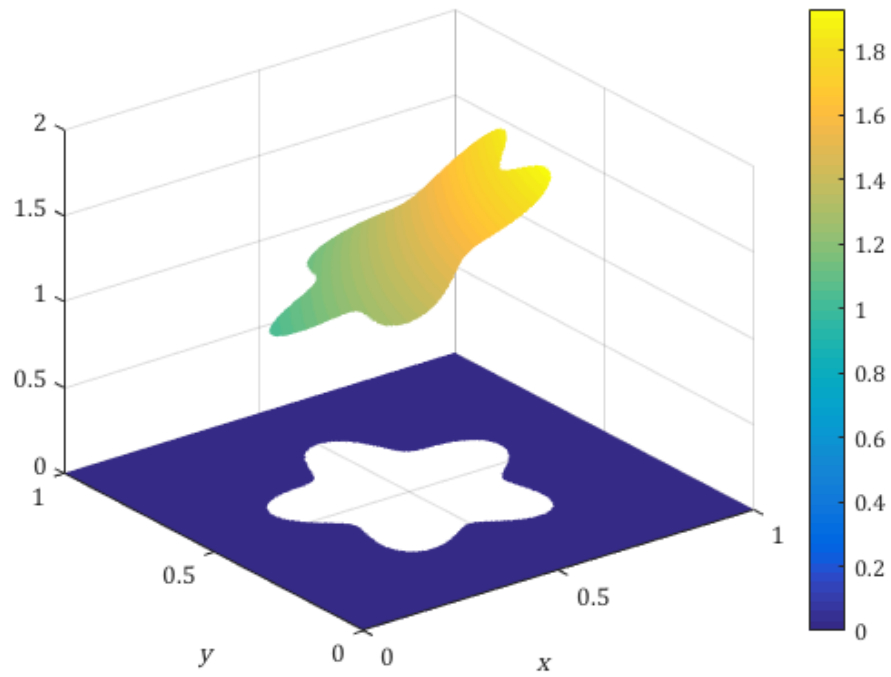


Figure 4.8. Computed solution of problem E2 on a 300×301 grid. On this grid, 1460 nodes require a correction and 730 cells are crossed by the interface.

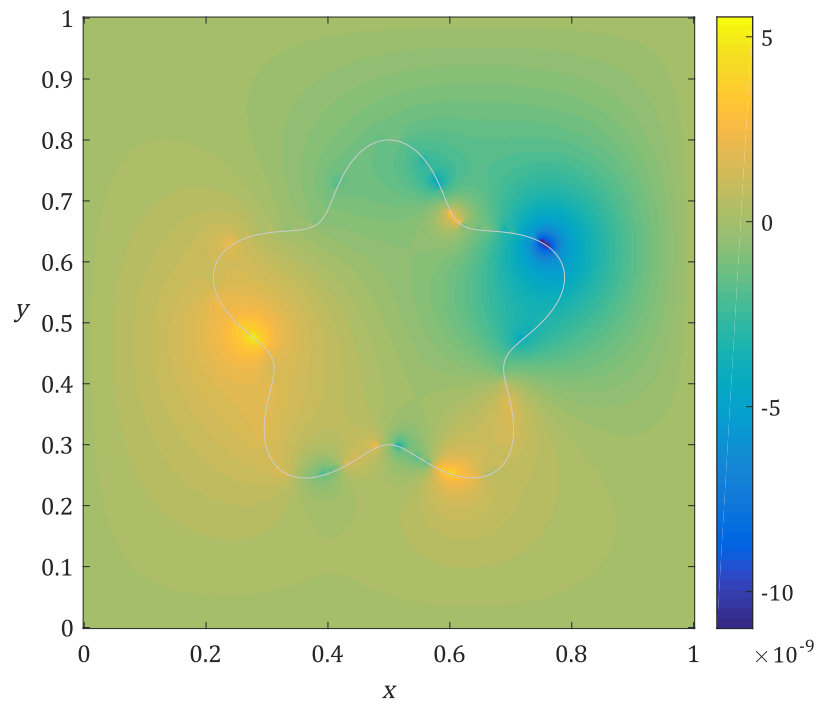


Figure 4.9. Error in the computed solution of example 2

4.3. Example 2

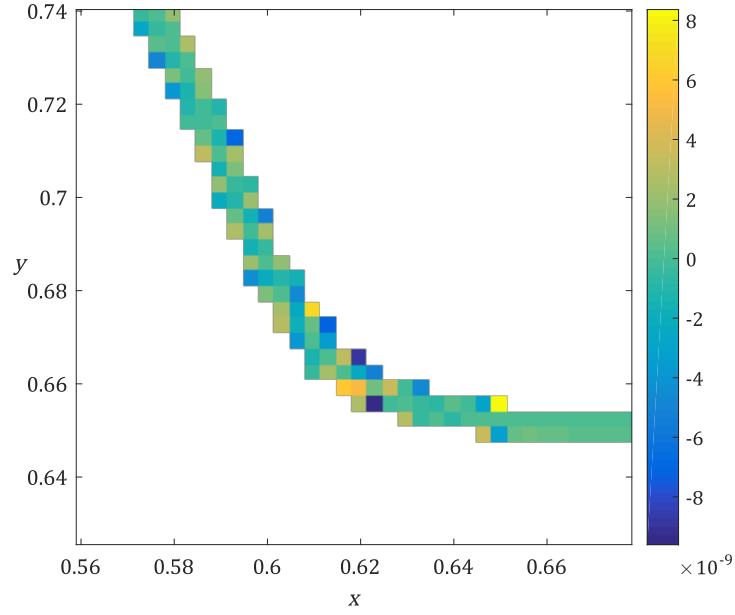


Figure 4.10. Error in the computed correction function for example 2

4.3.3 Convergence

In figures 4.11, 4.12 and 4.13, we show the convergence of the solution, the correction function and the gradient of the solution for problem E2. The solution converges very close to order 4 (3.8-3.9) in all norms, while the correction function converges to order 4.6 in L^∞ and slightly faster than order 5 in L^1 and L^2 . The gradient converges slightly faster than order 3 in L^∞ and very close to 4th order L^2 and L^1 (3.85-3.95).

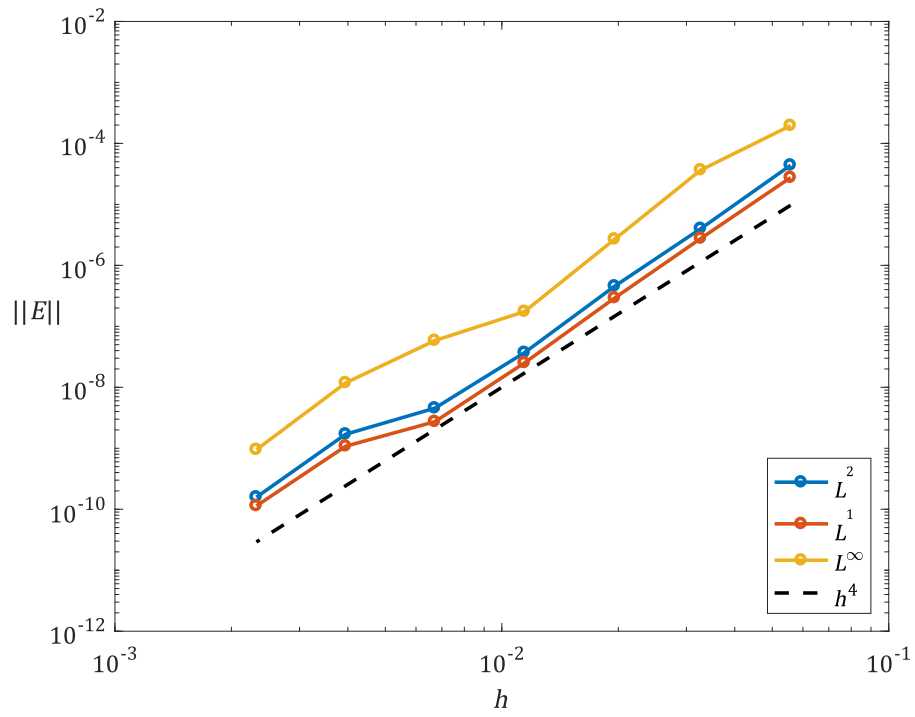


Figure 4.11. Convergence of the solution for example 2

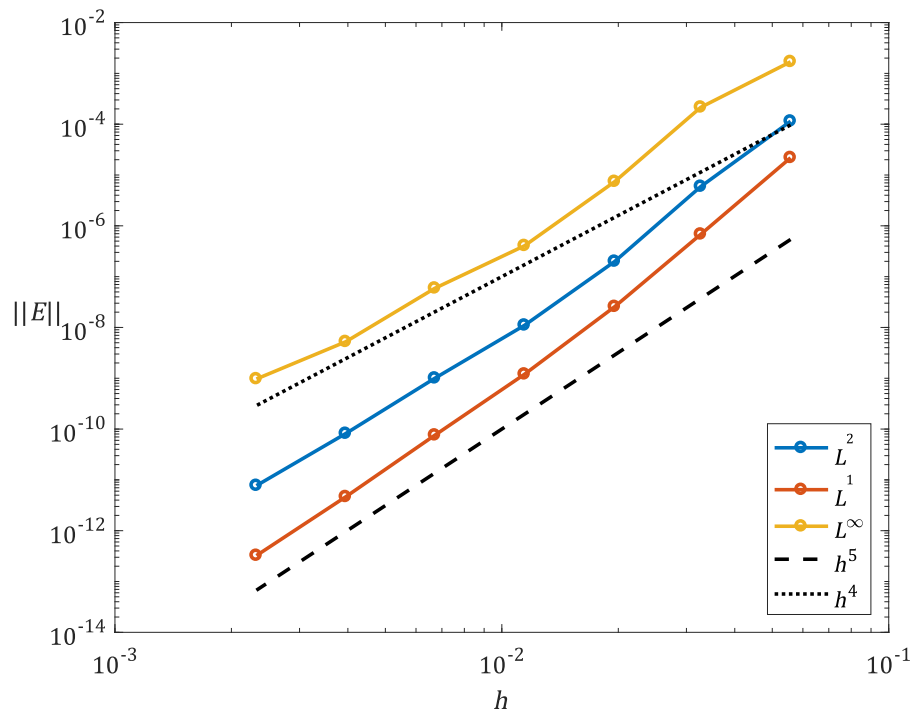


Figure 4.12. Convergence of the correction function for example 2

4.4. Example 3

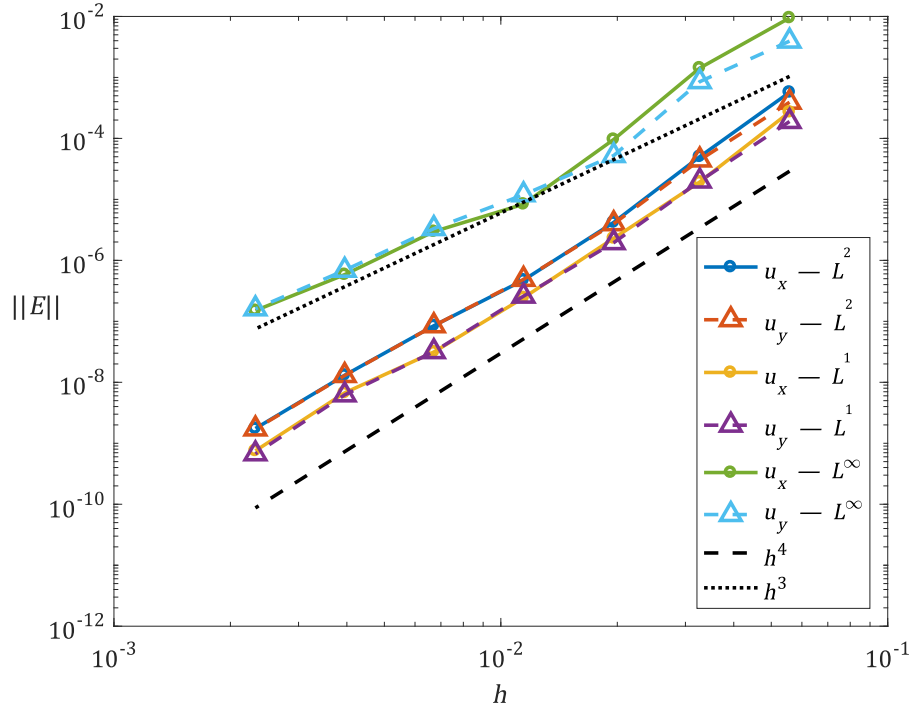


Figure 4.13. Convergence of the solution gradient for example 2

4.4 Example 3

4.4.1 Problem Definition

For our third example, we designed a simple problem in which the correction function is a constant in order to determine how it would influence the properties of the solution. The problem is given by

$$\begin{aligned}
 \Omega &= [0, 1]^2 \\
 f^+(x, y) &= -2\pi^2 \sin(\pi x) \sin(\pi y) \\
 f^-(x, y) &= -2\pi^2 \sin(\pi x) \sin(\pi y) \\
 \phi(x, y) &= (x - x_0)^2 + (y - y_0)^2 - r_0^2, \quad x_0 = y_0 = 0.5, r_0 = 0.1 \\
 f_D(x, y) &= 0 \\
 a(x, y) &= -1 \\
 b(x, y) &= 0
 \end{aligned} \tag{E3}$$

The exact solution is given by

$$u^+(x, y) = \sin(\pi x) \sin(\pi y)$$

$$u^-(x, y) = \sin(\pi x) \sin(\pi y) + 1$$

4.4.2 Numerical Solution

The solution to problem E3 is shown in figure 4.14 below, and as well as its error in figure 4.15. We observe that the error is completely smooth ; the simplicity of the correction function causes no increase of the error size near the interface.

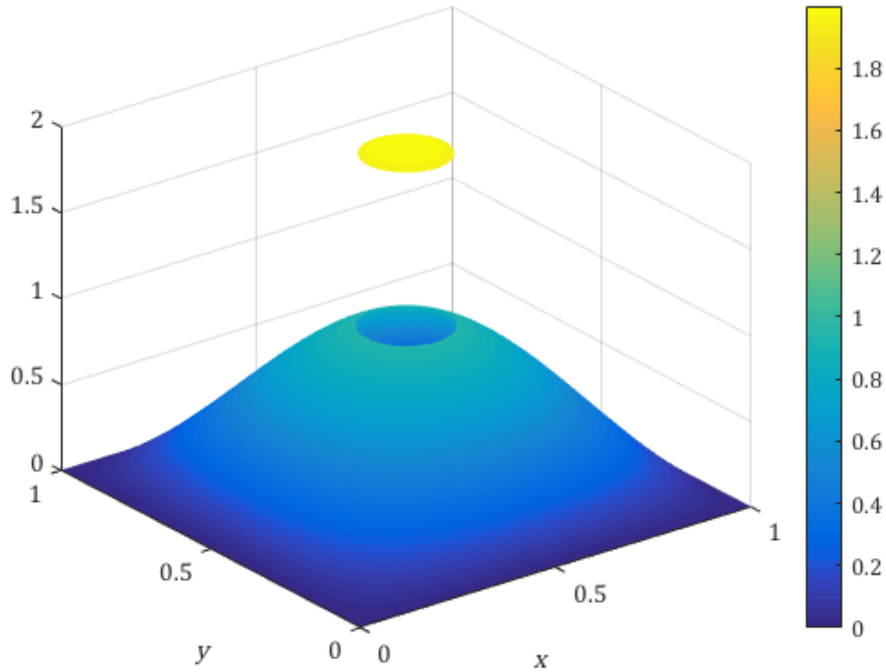


Figure 4.14. Computed solution of problem E3 on a 300×302 grid. On this grid, 480 nodes require a correction and 240 cells are crossed by the interface.

4.4.3 Convergence

The convergence of the solution, of the correction function and of the solution gradient are shown in figure 4.16-4.18. We observe very clear 4th order convergence of the solution and of the gradient, akin to a regular Poisson problem using the 9-point stencil. This is due to the simplicity of the correction function in this case. In 4.17, we observe that the L^∞ error of the correction function stays close to machine precision for all values of the step size h , and the

4.4. Example 3

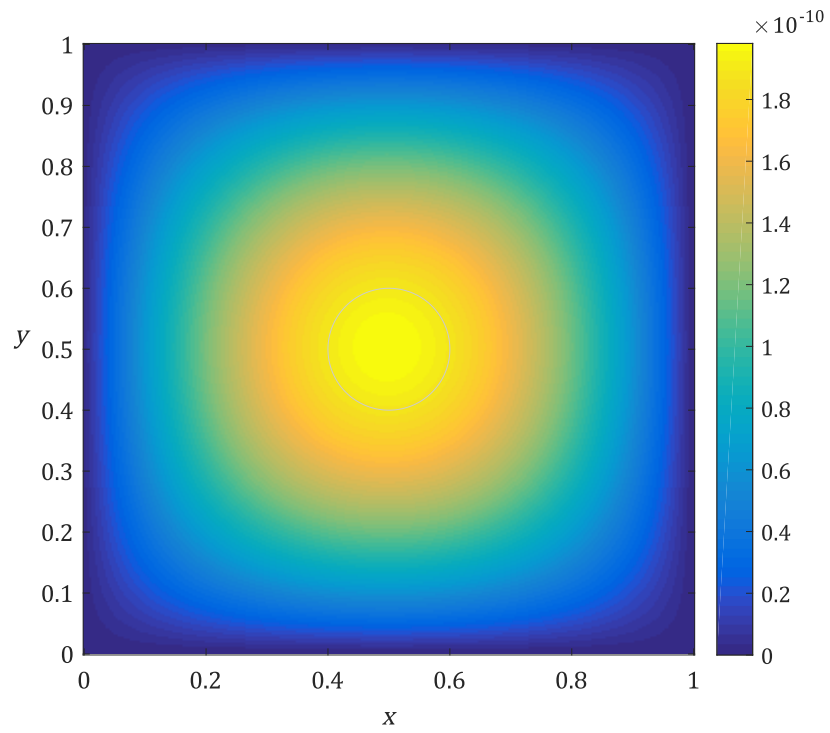


Figure 4.15. Error in the computed solution of example 3

L^1 and L^2 errors are smaller still. This is explained by the fact that the correction function is constant and thus has a finite Taylor expansion, consisting of only itself.

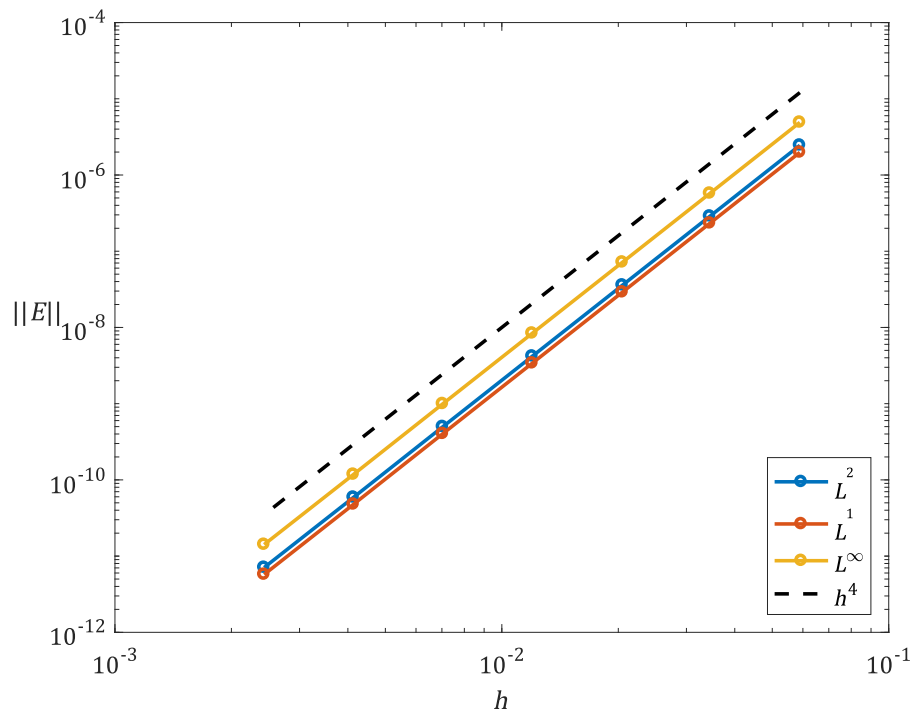


Figure 4.16. Convergence of the solution for example 3

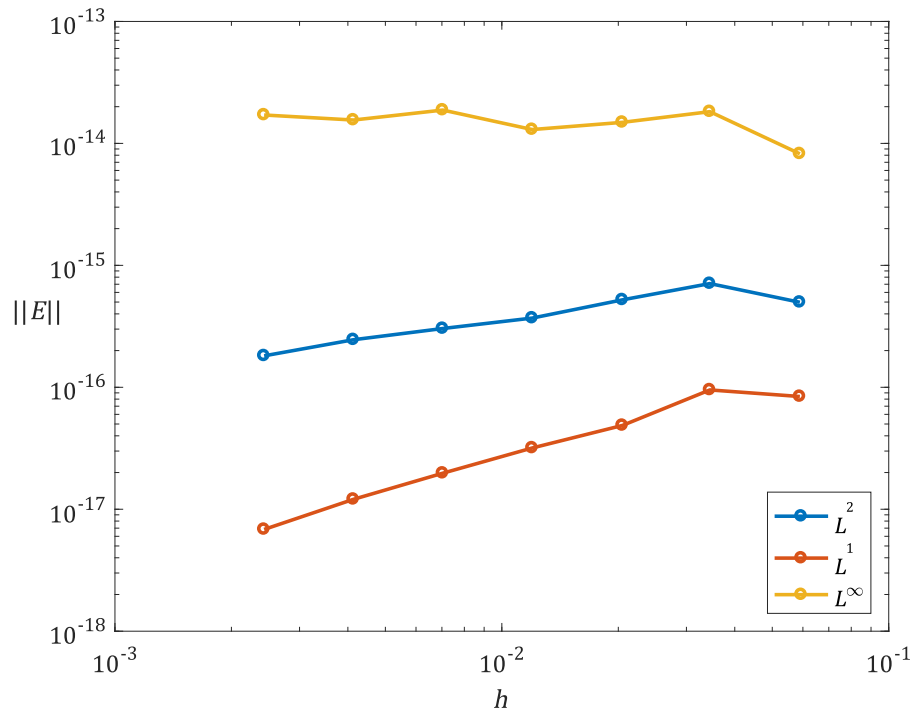


Figure 4.17. Convergence of the correction function for example 3

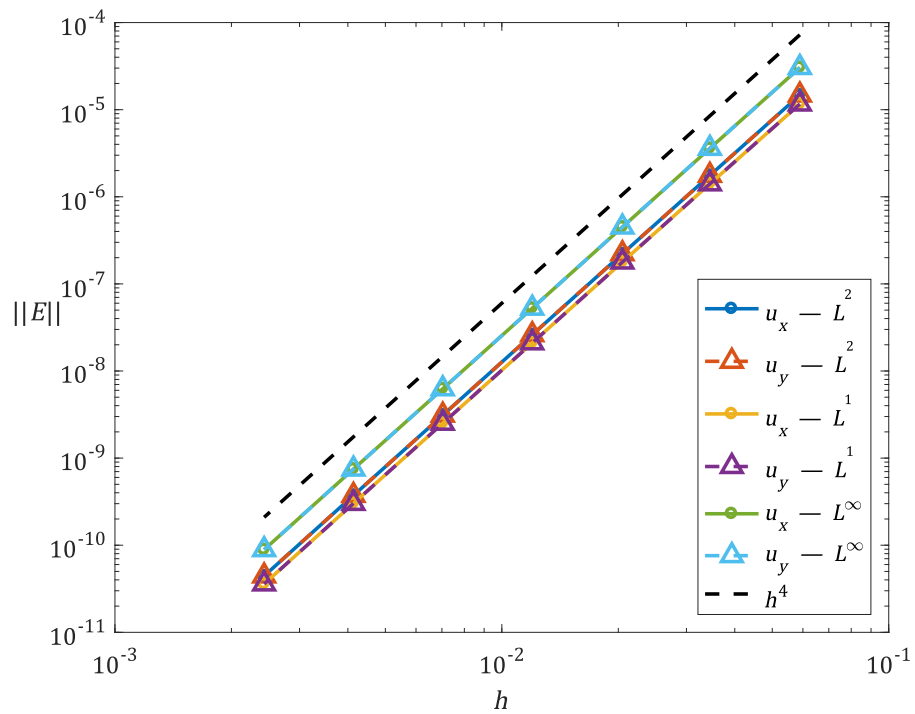


Figure 4.18. Convergence of the solution gradient for example 3

4.5 Example 4

4.5.1 Problem Definition

As a final example of a Poisson interface problem, we slightly modify Example 3 from [31]. In this problem, the interface consists of two distinct circles represented by the same level set function. The problem is defined as

$$\begin{aligned}
 \Omega &= [0, 1]^2 \\
 f^+(x, y) &= \exp(x)((4x + 2) \sin(y) + y^2 + 2) \\
 f^-(x, y) &= -\exp(x)((4y^2 - 1) \cos(y^2) + 2 \sin(y^2)) \\
 \phi(x, y) &= ((x - x_0)^2 + (y - y_0)^2 - r_0^2)((x - x_1)^2 + (y - y_1)^2 - r_1^2), \\
 x_0 &= y_0 = 0.5, \quad r_0 = 0.15 \quad x_1 = y_1 = 0.75, \quad r_1 = 0.1 \tag{E4} \\
 f_D(x, y) &= \exp(x)((4y^2 - 1) \cos(y^2) + 2 \sin(y^2) + (4x + 2) \sin(y) + y^2 + 2) \\
 a(x, y) &= -\exp(x)(-x^2 \sin(y) - y^2 + \cos(y^2)) \\
 b(x, y) &= -\exp(x)((-x^2 \sin(y) - 2x \sin(y) - y^2 + \cos(y^2))n_x \\
 &\quad - (-x^2 \cos(y) - 2 \sin(y^2)y - 2y)n_y)
 \end{aligned}$$

Notice here we have $f_D \neq 0$. The exact solution is given by

$$\begin{aligned}
 u^+(x, y) &= \exp(x)(x^2 \sin(y) + y^2) \\
 u^-(x, y) &= \exp(x) \cos(y^2)
 \end{aligned}$$

4.5.2 Numerical Solution

The computed solution of problem E4 appears in figure 4.19. We see the jumps along both interfaces are very sharp. The method and our implementation of it are able to handle numerous interfaces with no modification, as long as those are defined by a level set function.

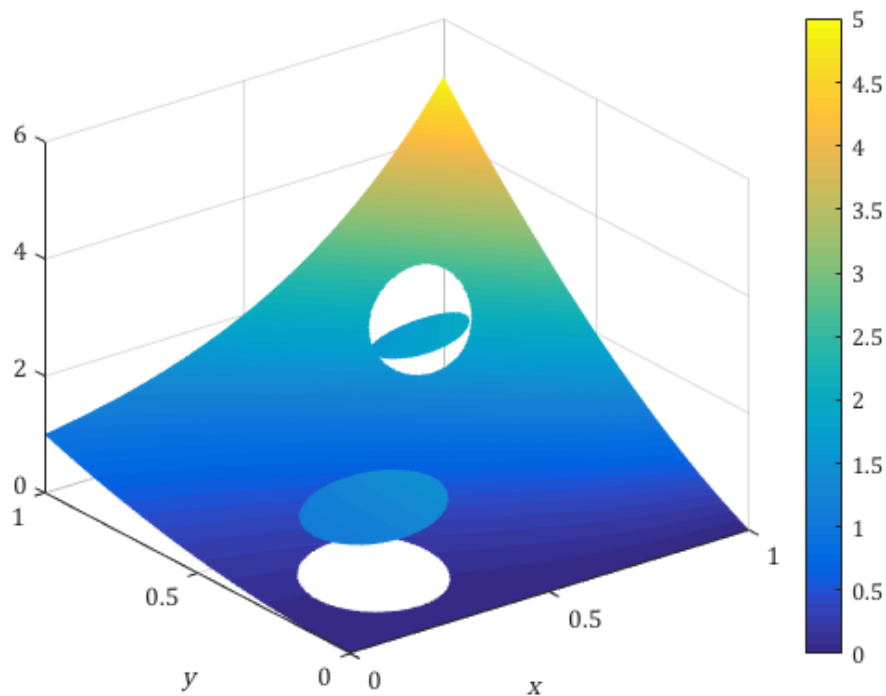


Figure 4.19. Computed solution of problem E4 on a 400×420 grid. On this grid, 1640 nodes require a correction and 820 cells are crossed by the interface.

4.5.3 Convergence

Figures 4.20-4.22 show the convergence plots for the solution, the correction function and the solution gradient. The solution converges to 4th order in the L^1 and L^2 norms, and close to 4th order in the L^∞ norm. The correction function shows a convergence order of 4 in the L^∞ norm, 4.7 for the L^2 norm and 5.2 for the L^1 norm. The gradient converges to 3rd order in the L^∞ norm and close to 4th order in the L^1 and L^2 norms.

4.5. Example 4

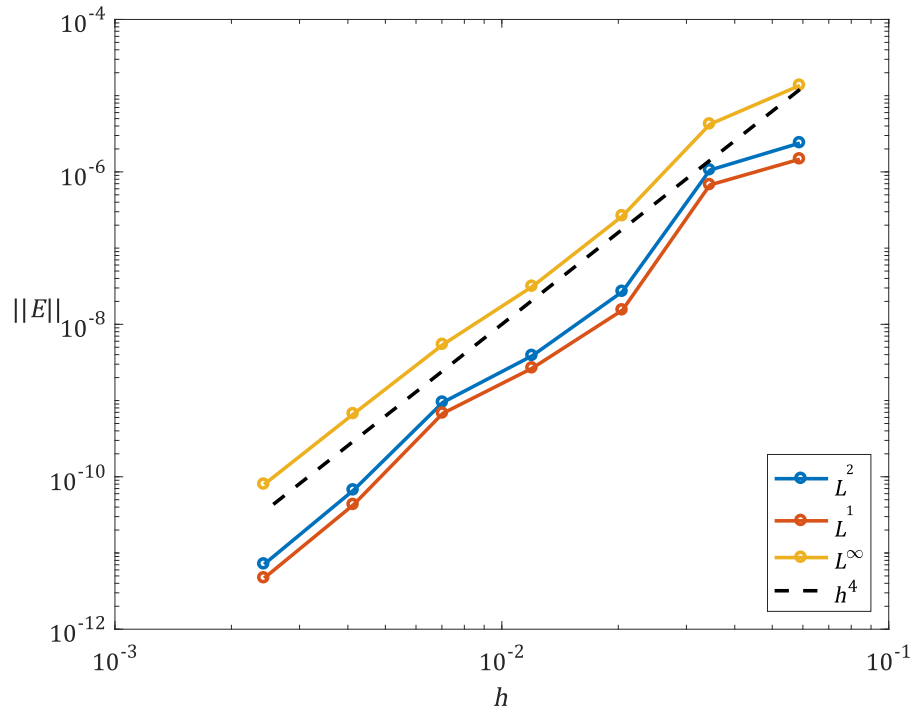


Figure 4.20. Convergence of the solution for example 4

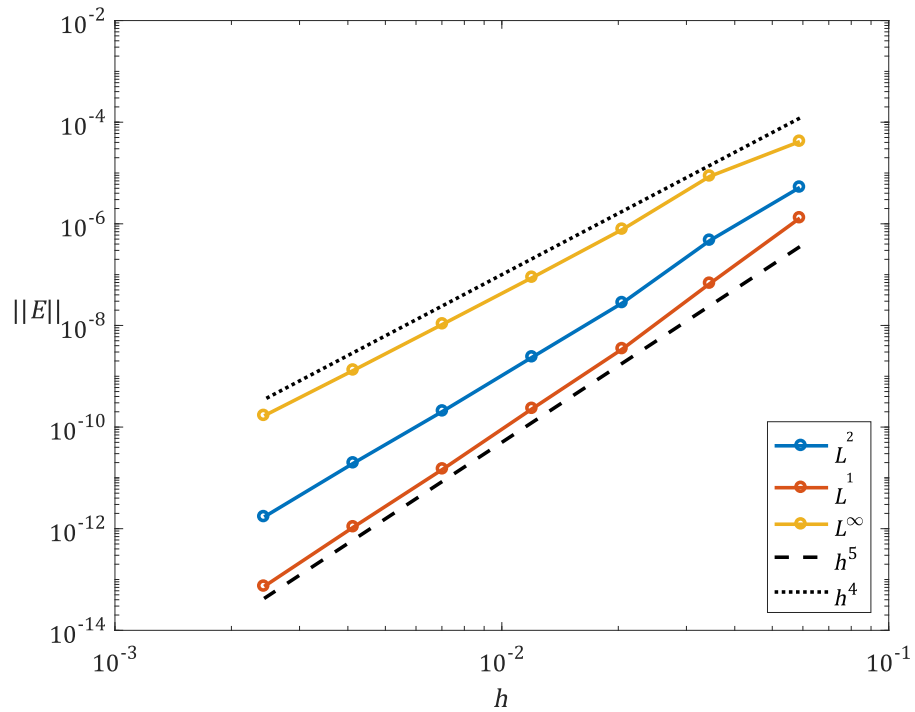


Figure 4.21. Convergence of the correction function for example 4

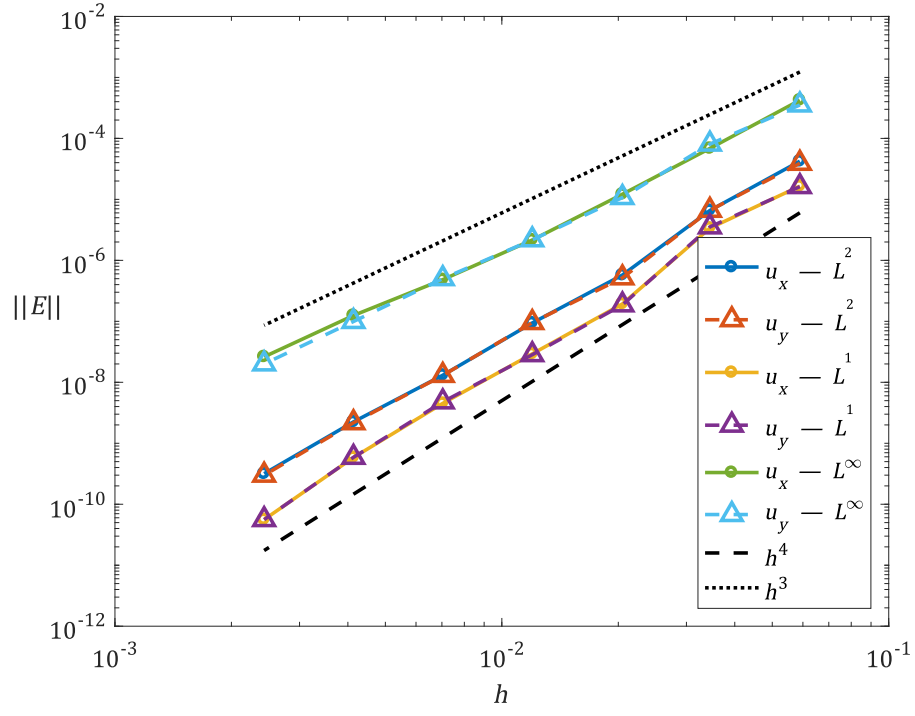


Figure 4.22. Convergence of the solution gradient for example 4

5. Discussion

In this chapter, we comment on the results obtained in the previous chapter. We first offer some general remarks in Section 5.1. We explain an important detail of the local correction function solver in Section 5.2. Section 5.3 presents some perspective on the design of a high-order CFM-based Navier-Stokes solver.

5.1 General Remarks

The convergence results presented above confirm our implementation of the Correction Function Method is correct. On average, the solution converges to order 4, the correction function usually converges to order 5 and the gradient, to order 3. These results are in line with the results presented in [31] and [33].

The CFM, as it is presented, deals with Poisson interface problems, but it is interesting to note that it can be extended to more general problems easily. Example 2 above shows that the CFM can be used to solve a regular Poisson problem on an irregular domain, by embedding this domain inside a rectangular computational box. This can be further extended to Poisson interface problems posed on an irregular domain, as is explored in [32]. In this paper, the CFM is coupled with a boundary integral method to solve problems of the form

$$\Delta(\beta u) = f \tag{5.1}$$

on an irregular domain Ω . In the equation above, β is a piecewise constant coefficient (it is constant on each of the subdomains Ω^- and Ω^+).

Moreover, the method as it is presented can easily handle problems with more than one level set function, and thus more than two subdomains. This is not handled by our implementation but could be easily added. In fact, because of the linearity of the Poisson interface problem,

each interface jump can be treated independently. The different interfaces can even overlap, or touch at a single point without causing a degradation of the solution. Examples of problems with multiple jumps are presented in [31].

5.2 Domain of Definition of the Bicubic Interpolants

In section 3.2.4.5 we explain how the integrals used to determine the correction function are mapped to the reference domain, the unit square K , in order for the bicubic interpolants to be defined over K . Here we explain why this second transformation is needed. In a precedent version of the code, we did not use the mapping \hat{T} to transform the integrals to the unit square. Instead, the bicubics used for the approximation of the correction function were defined on a box B_{ij} enclosing the integration domain Ω_{Γ}^{ij} , as shown in figure 5.1.

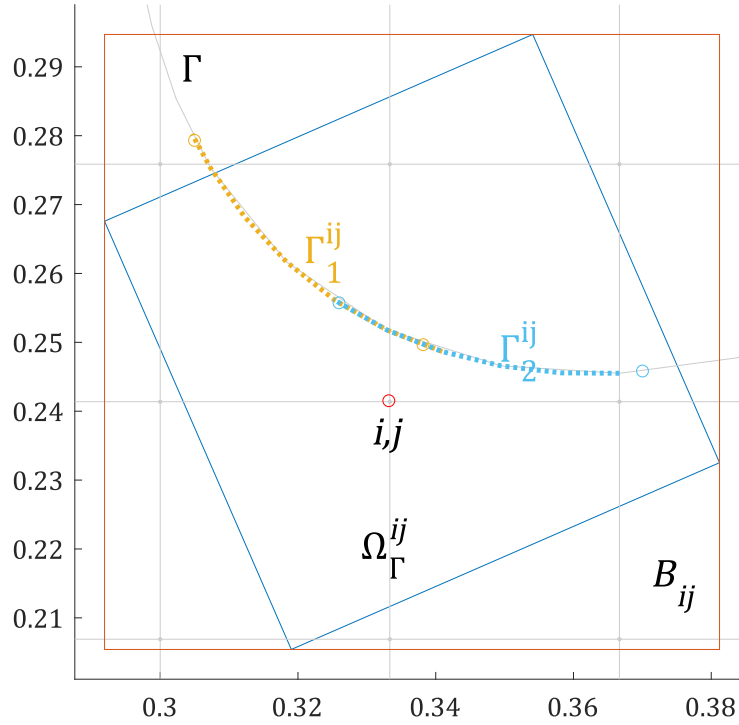


Figure 5.1. The box B_{ij} encloses the domain Ω_{Γ}^{ij} . In a previous version of the code, the mapping to the reference domain K presented in section 3.2.4.5 was not used and the bicubics were defined in real space over B_{ij} .

However, by examining the definition of the bicubic interpolants in equation (3.86), we see that the polynomials B_{α}^v have a prefactor of Δx^{α} , with the multi-index $\alpha \in \{0, 1\}^2$. Since all 12 of the reduced bicubic polynomials are multiplied in pairs in the discretization of the mini-

5.2. Domain of Definition of the Bicubic Interpolants

mization integrals (3.146) and (3.157) (see Appendix A), in that case the entries of the minimization matrix A in (3.82) vary greatly in magnitude. We can see this variation clearly by plotting the logarithm of the absolute value of matrix entries, as shown in figure 5.2.

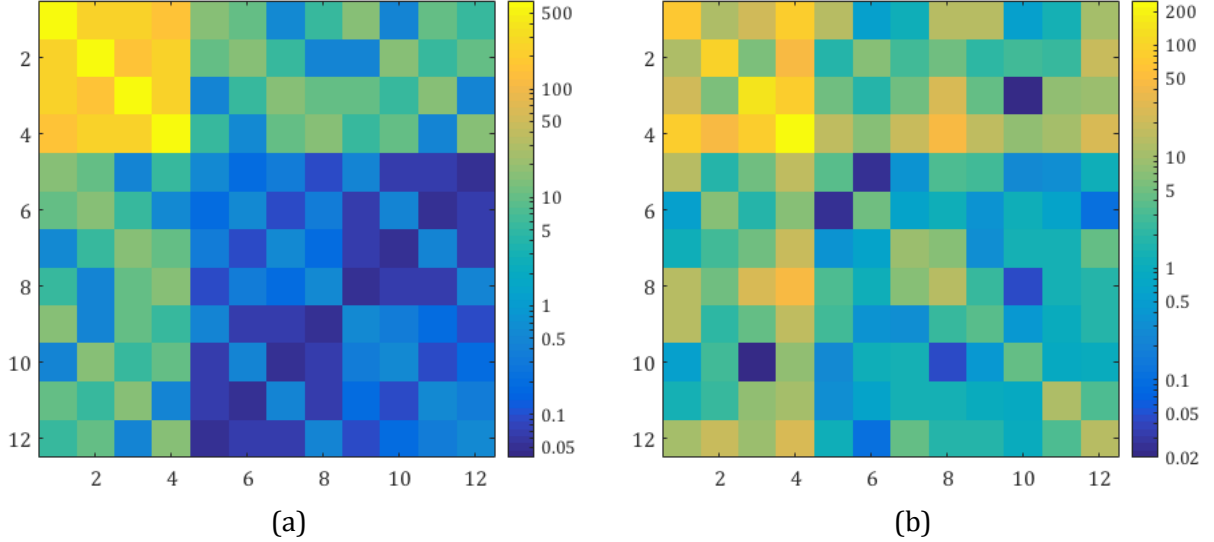


Figure 5.2. (a) Logarithm of the absolute value of the entries of the minimization matrix A for a previous version of the code, where the bicubics are defined over B_{ij} . Notice the difference in magnitude of the 4×4 blocks. (b) Logarithm of the absolute value of the entries of the minimization matrix A for the final implementation, where the bicubics are defined over the unit square K . Notice the magnitude of the entries is much more even across the entire matrix.

This important variation of magnitude across the matrix entries increased as we refined the grid, since it is dependent on the mesh size. This leads to a very bad conditioning of the minimization matrices; for example, the matrix in 5.2a has a condition number of 9.1×10^7 , for a grid size of 31×30 . This condition number increased rapidly as the grid was refined, leading to almost singular matrices. On the contrary, when the bicubics are defined on the unit square K , as detailed in section 3.2.4.5, the prefactor $\Delta x^\alpha = (1, 1)^\alpha = (1, 1)$ and thus does not contribute to the magnitude of the minimization matrix entries. The entries are much more even across the matrix, as can be seen in 5.2b, leading to a greatly improved condition number of 1.4×10^3 . Moreover, the magnitude of this condition number stays constant as the grid is refined.

5.3 CFM-Based Navier-Stokes Solver

As stated above, we could not produce meaningful results from the Navier-Stokes solver `bubbles` used in conjunction with the C++ version of our CFM code, `cfm-petsc`. Namely, the jump in the pressure was smeared out, and the level set moved in nonsensical ways. We suspect this is due to implementation errors, but it could also come from mathematical incompatibilities between the codes. These ideas are explored in this section.

The first thing to note is that our implementation uses periodic boundary conditions. The convergence of `cfm-petsc` with these boundary conditions was verified. However, modifications also needed to be made to the code `bubbles` to implement these conditions. Those modifications could not be tested in isolation (without using the code in conjunction with `cfm-petsc`) because it would have required major code changes; the linear solver implemented in `bubbles` was not designed to handle the singular system resulting from the periodic boundary conditions. Hence, we could not identify if the implementation of the periodic boundary conditions in `bubbles` was problematic.

Another potential problem is the fact that by using the 4th order 9-point stencil for the Laplacian implemented in `cfm-petsc`, the mimetic properties of the finite difference discretization of the Navier-Stokes equation mentioned in section 3.4 are lost; the discrete gradient, Laplacian and divergence do not exactly satisfy classical vector identities anymore. It is possible that this causes important stability issues preventing the method from being consistent. Note, however, that in step 3 of algorithm 3.3, we did modify the computation of the gradient so that all quantities appearing in equation (3.204) are computed at the same location. This is needed because of the MAC grid; the pressure and velocities are not defined at the same location. Specifically, we use the following compact fourth-order discretization of the first derivative of the pressure p at the half-step :

$$\begin{aligned}\partial_x p|_{i-1/2,j} &= \frac{p_{i,j} - p_{i-1,j}}{h_x} - \frac{1}{24} h_x^2 \partial_x^3 p|_{i-1/2,j} + \mathcal{O}(h^4) \\ &= \frac{p_{i,j} - p_{i-1,j}}{h_x} - \frac{1}{24} h_x^2 \left(\partial_x f - p_{xyy}|_{i-1/2,j} \right) + \mathcal{O}(h^4) \\ &= \frac{p_{i,j} - p_{i-1,j}}{h_x} - \frac{1}{24} h_x^2 \left(\frac{f_i - f_{i-1}}{h_x} - \partial_x p_{yy}|_{i-1/2,j} \right) + \mathcal{O}(h^4)\end{aligned}$$

$$\begin{aligned}
&= \frac{p_{i,j} - p_{i-1,j}}{h_x} - \frac{1}{24} h_x^2 \left(\frac{f_i - f_{i-1}}{h_x} - \frac{p_{yy}|_i - p_{yy}|_{i-1}}{h_x} \right) + \mathcal{O}(h^4) \\
&= \frac{p_{i,j} - p_{i-1,j}}{h_x} - \frac{1}{24} h_x^2 \left(\frac{f_i - f_{i-1}}{h_x} - \frac{\hat{\partial}_{yy} p_i - \hat{\partial}_{yy} p_{i-1}}{h_x} \right) + \mathcal{O}(h^4)
\end{aligned} \tag{5.2}$$

In the equation above, the function f corresponds to the right-hand side of equation (3.203), the scaled divergence of the tentative velocity. In order for the stability of the scheme to be preserved, a compact fourth-order discretization of all differential operators appearing in algorithm 3.3 should be used. We also tried to keep the gradient computation done by the `bubbles` code to preserve the mimetic discrete properties, and use `cfm-petsc` only to solve the Poisson interface problem for the pressure, but this did not solve the inconsistencies in the output.

Yet another thing to consider is how the data for the Poisson interface problem is passed from `bubbles` to `cfm-petsc`. Recall that the function a that corresponds to the jump in the pressure at the interface is given by the curvature of the level set ϕ (see equation (3.200)). This curvature is computed following equation (3.201) by the code `bubbles` using second order finite differences of the discretized level set ϕ , and is capped to the inverse of the grid size to prevent arbitrarily large curvature. When received by `cfm-petsc` and converted to a `GridFunction2D` object, it is discretely differentiated once again to obtain the gradient information necessary for the bicubic interpolation used by `GridFunction2D` (see section 3.2.4.2) to evaluate it at arbitrary locations in the domain. This discrete gradient is computed using a fourth-order stencil, as mentioned above, to maintain the accuracy of the bicubic interpolation scheme. It is well possible that the computed curvature does not have the required smoothness to be differentiated again to fourth-order in the class `GridFunction2D`. An alternative approach would be to replace the WENO method used to represent and advect the level set in the code `bubbles` by the gradient-augmented level set method (GALS) of [35]. This method uses the reduced bicubic interpolants of section 3.2.4.2 to represent the level set. We could then compute the curvature anywhere on the domain directly using the bicubic interpolation framework, without the need to differentiate it again with finite differences when passed to `cfm-petsc`. Moreover, the GALS method was shown in [35] to be superior to WENO for the advection equation (3.205). A complete high-order implementation of algorithm 3.3 would thus benefit from this advection method.

6. Conclusion

In this work, we gave a comprehensive presentation of the Correction Function Method, a numerical method for the solution of the Poisson interface problem. We showed how this method uses the so-called correction function, defined as the solution of a PDE posed on a narrow band around the interface, to determine the correction terms needed to account for the interface jumps. We explained how this PDE is solved by minimizing an integral functional parameterized using the level set information, and detailed the fourth-order finite difference scheme used for the discretization of the Poisson PDE in 2 dimensions. We also derived the Fast Poisson solver used to efficiently solve the linear system resulting from this discretization. We implemented the 4th order CFM method in both MATLAB and C++ and described our object-oriented code architecture.

We showed convergence of our implementation using four model problems. We also investigated an alternate way to solve the correction function PDE, based on a classical representation formula for the solution of the Poisson problem, and showed that it could not succeed in solving the correction function problem in a local fashion. Finally, we explored the coupling of the CFM to a Navier-Stokes solver to simulate flow with pressure discontinuities. Although we were not successful in presenting simulation results for the Navier-Stokes case, we outlined several issues that ought to be addressed if a high-order, CFM-based Navier-Stokes solver is to be designed.

Another interesting future development of the method presented here would be to extend it to handle grid-based discontinuous sources. As mentioned in section 3.2.1, the function f_D appearing in the PDE for the correction function (3.50) needs both of the source functions f^+ and f^- to be defined on the band domain Ω_Γ . If a single grid function f describing both f^+ and f^- is the only available data for the problem, we need to extend f^+ inside Ω^- and f^- inside Ω^+ in order to define f_D at the nodes where corrections are needed. These extensions could

be defined by extrapolation. An idea would be to use the bicubic interpolation framework and use it for extrapolation. Then, the question of choosing the best location to choose for the extrapolation data would have to be addressed. Since the configuration of the extrapolated values with respect to the data would vary considerably depending on the way the grid is cut by the interface, such an extrapolation method would have to be general enough yet easily automatized.

At the level of the code implementation, the C++ version of the code would greatly benefit from performance optimization and parallelization. The fact that the correction function at each required node is computed in a completely independent fashion represents an ideal opportunity for parallelization. Both multi-thread parallelism and distributed parallelism would be possible, as well as a combination of the two. Moreover, it would be interesting to assess how a distributed Fast Poisson solver, using a parallel implementation of the Fast Fourier Transform, would measure to the distributed iterative linear solvers of the PETSc library.

Appendix A

Linear System for the Correction Function Coefficients

Once parameterized and mapped to the reference domain K , as described in sections 3.2.4.4 and 3.2.4.5, the functional $J(D)$ in (3.77) becomes

$$\begin{aligned}
 J(D) &= I_1 + c_p \sum_{k=1}^{N_{ij}} I_2^k \\
 &= c_1 \int_0^1 \int_0^1 \left(\frac{1}{s^2} \sum_m L_m(\xi, \eta) \hat{D}_m - f_D(\hat{T}(\xi, \eta)) \right)^2 s^2 d\xi d\eta \\
 &\quad + c_p \sum_{k=1}^{N_{ij}} \int_0^1 \left[c_2 \left(\sum_m \bar{B}_m(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_2))) \hat{D}_m - a(\tilde{T}_k^{-1}(\Theta, \theta_2)) \right)^2 \right. \\
 &\quad \left. + c_3 \left(\frac{1}{s^2} \sum_m r_m(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_2))) \hat{D}_m - b(\tilde{T}_k^{-1}(\Theta, \theta_2)) \right)^2 \right] \\
 &\quad \times \sqrt{\left(\frac{\partial H_1^k}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2^k}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_2)} d\theta_2
 \end{aligned} \tag{A.1}$$

In the above we used the shortcut notations L_m and r_m defined respectively in (3.147) and (3.158). We use Gauss-Legendre quadrature [42] to discretize the integrals above. Denoting the 2D integration nodes (ξ_ℓ, η_n) and corresponding 2D weights (w_ℓ, w_n) , and the 1D integration nodes θ_ℓ and corresponding weights w_ℓ , the discretized functional $\tilde{J}(D)$ becomes a

quadratic function of the correction function coefficients \hat{D}_m :

$$\begin{aligned}
\tilde{J}(D) = & s^2 \sum_{\ell} \sum_n \left(\frac{1}{s^2} \sum_m L_m(\xi_{\ell}, \eta_n) \hat{D}_m - f_D(\hat{T}(\xi_{\ell}, \eta_n)) \right)^2 s^2 \\
& + \sum_{k=1}^{N_{ij}} \frac{c_p}{s_{\Gamma}^k} \sum_{\ell} \left[\left(\sum_m \bar{B}_m(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) \hat{D}_m - a(\tilde{T}_k^{-1}(\Theta, \theta_{\ell})) \right)^2 \right. \\
& \quad \left. + s^2 \left(\frac{1}{s^2} \sum_m r_m(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) \hat{D}_m - b(\tilde{T}_k^{-1}(\Theta, \theta_{\ell})) \right)^2 \right] \\
& \quad \times \sqrt{\left(\frac{\partial H_1^k}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2^k}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_{\ell})}
\end{aligned} \tag{A.2}$$

where we used the expressions of the scaling coefficients c_1 , c_2 and c_3 from section 3.2.4.6. Expanding the squared parenthesised terms, we get

$$\begin{aligned}
\tilde{J}(D) = & \sum_{\ell} \sum_n \left(\sum_p \sum_q L_p(\xi_{\ell}, \eta_n) L_q(\xi_{\ell}, \eta_n) \hat{D}_p \hat{D}_q \right. \\
& \quad \left. - 2s^2 f_D(\hat{T}(\xi_{\ell}, \eta_n)) \sum_p L_p(\xi_{\ell}, \eta_n) \hat{D}_p + s^4 f_D^2(\hat{T}(\xi_{\ell}, \eta_n)) \right) \\
& + \sum_{k=1}^{N_{ij}} \frac{c_p}{s_{\Gamma}^k} \sum_{\ell} \left[\left(\sum_p \sum_q \bar{B}_p(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) \bar{B}_q(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) \hat{D}_p \hat{D}_q \right. \right. \\
& \quad \left. - 2a(\tilde{T}_k^{-1}(\Theta, \theta_{\ell})) \sum_p \bar{B}_p(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) \hat{D}_p + a^2(\tilde{T}_k^{-1}(\Theta, \theta_{\ell})) \right) \\
& \quad + \left(\frac{1}{s^2} \sum_p \sum_q r_p(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) r_q(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) \hat{D}_p \hat{D}_q \right. \\
& \quad \left. - 2b(\tilde{T}_k^{-1}(\Theta, \theta_{\ell})) \sum_p r_p(\hat{T}^{-1}(\tilde{T}_k^{-1}(\Theta, \theta_{\ell}))) \hat{D}_p \right. \\
& \quad \left. \left. + s^2 b^2(\tilde{T}_k^{-1}(\Theta, \theta_{\ell})) \right) \right] \times \sqrt{\left(\frac{\partial H_1^k}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2^k}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_{\ell})}
\end{aligned} \tag{A.3}$$

where we used the indices p and q to rewrite the squared sum over the index m , and renamed the lone m index to p for consistency. We can finally isolate the expressions of the entries of

the matrix A and of the right-hand side vector d in the linear system (3.81) resulting from the minimization of $\tilde{J}(D)$:

$$A_{pq} = \sum_{\ell} \sum_n L_p(\xi_{\ell}, \eta_n) L_q(\xi_{\ell}, \eta_n) \quad (\text{A.4})$$

$$\begin{aligned} & + \sum_{k=1}^{N_{ij}} \frac{c_p}{s_{\Gamma}^k} \sum_{\ell} \left[\bar{B}_p \left(\hat{T}^{-1} \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) \right) \bar{B}_q \left(\hat{T}^{-1} \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) \right) \right. \\ & \quad \left. + \frac{1}{s^2} r_p \left(\hat{T}^{-1} \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) \right) r_q \left(\hat{T}^{-1} \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) \right) \right] \\ & \quad \times \sqrt{\left(\frac{\partial H_1^k}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2^k}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_{\ell})} \\ d_p = & -2 \left[s^2 \sum_{\ell} \sum_n f_D \left(\hat{T} \left(\xi_{\ell}, \eta_n \right) \right) L_p(\xi_{\ell}, \eta_n) \right] \quad (\text{A.5}) \\ & -2 \sum_{k=1}^{N_{ij}} \frac{c_p}{s_{\Gamma}^k} \sum_{\ell} \left[a \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) \bar{B}_p \left(\hat{T}^{-1} \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) \right) \right. \\ & \quad \left. + b \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) r_p \left(\hat{T}^{-1} \left(\tilde{T}_k^{-1} (\Theta, \theta_{\ell}) \right) \right) \right] \\ & \quad \times \sqrt{\left(\frac{\partial H_1^k}{\partial \theta_2} \right)^2 + \left(\frac{\partial H_2^k}{\partial \theta_2} \right)^2} \Big|_{(\Theta, \theta_{\ell})} \end{aligned}$$

Bibliography

- [1] David S. Abraham, Alexandre Noll Marques, and Jean-Christophe Nave. A Correction Function Method for the Wave Equation with Interface Jump Conditions. *arXiv:1609.05379 [math]*, September 2016. arXiv: 1609.05379.
- [2] Ivo Babuška. The finite element method for elliptic equations with discontinuous coefficients. *Computing*, 5(3):207–213, September 1970.
- [3] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. *PETSc Web page*. 2018.
- [4] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. *PETSc Users Manual*. Technical Report ANL-95/11 - Revision 3.9, Argonne National Laboratory, 2018.
- [5] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
- [6] J. Frédéric Bonnans, J. Charles Gilbert, Claude Lemaréchal, and Claudia A. Sagastizábal. *Numerical Optimization*. Universitext. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

- [7] James H. Bramble and J. Thomas King. A finite element method for interface problems in domains with smooth boundaries and interfaces. *Advances in Computational Mathematics*, 6(1):109–138, 1996.
- [8] W. Briggs and V. Henson. *The DFT: An Owner's Manual for the Discrete Fourier Transform*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, January 1995.
- [9] Zhiming Chen and Jun Zou. Finite element methods and their convergence for elliptic and parabolic interface problems. *Numerische Mathematik*, 79(2):175–202, 1998.
- [10] Alexandre Joel Chorin. The numerical solution of the Navier-Stokes equations for an incompressible fluid. *Bulletin of the American Mathematical Society*, 73(6):928–931, November 1967.
- [11] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI Message Passing Interface Standard. In *Programming Environments for Massively Parallel Distributed Systems*, Monte Verità, pages 213–218. Birkhäuser, Basel, 1994.
- [12] John Dolbow and Isaac Harari. An efficient finite element method for embedded interface problems. *International Journal for Numerical Methods in Engineering*, 78(2):229–252, April 2009.
- [13] Iain S. Duff. MA57—a Code for the Solution of Sparse Symmetric Definite and Indefinite Systems. *ACM Trans. Math. Softw.*, 30(2):118–144, June 2004.
- [14] Lawrence C. Evans. *Partial differential equations*. Number v. 19 in Graduate studies in mathematics. American Mathematical Society, Providence, R.I, 2nd ed edition, 2010. OCLC: ocn465190110.
- [15] Ronald P. Fedkiw. The Ghost Fluid Method for Numerical Treatment of Discontinuities and Interfaces. In *Godunov Methods*, pages 309–317. Springer, Boston, MA, 2001.
- [16] Ronald P. Fedkiw. Coupling an Eulerian Fluid Calculation to a Lagrangian Solid Calculation with the Ghost Fluid Method. *Journal of Computational Physics*, 175(1):200–224, January 2002.

- [17] Ronald P Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A Non-oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (the Ghost Fluid Method). *Journal of Computational Physics*, 152(2):457–492, July 1999.
- [18] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. 2013.
- [19] Yan Gong, Bo Li, and Zhilin Li. Immersed-Interface Finite-Element Methods for Elliptic Interface Problems with Nonhomogeneous Jump Conditions. *SIAM Journal on Numerical Analysis*, 46(1):472–495, January 2008.
- [20] Francis H. Harlow and J. Eddie Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *The Physics of Fluids*, 8(12):2182–2189, December 1965.
- [21] Cay S. Horstmann and Timothy A. Budd. *Big C++*. Wiley, Hoboken, NJ, 2. ed edition, 2009. OCLC: 403586997.
- [22] Myungjoo Kang, Ronald P. Fedkiw, and Xu-Dong Liu. A Boundary Condition Capturing Method for Multiphase Incompressible Flow. *Journal of Scientific Computing*, 15(3):323–360, December 2000.
- [23] R. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, January 2007.
- [24] R. LeVeque and Z. Li. The Immersed Interface Method for Elliptic Equations with Discontinuous Coefficients and Singular Sources. *SIAM Journal on Numerical Analysis*, 31(4):1019–1044, August 1994.
- [25] Z. Li and K. Ito. *The Immersed Interface Method*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, January 2006.
- [26] Zhilin Li. *The Immersed Interface Method – A Numerical Approach for Partial Differential Equations with Interfaces*. PhD thesis, University of Washington, 1994.
- [27] Xu-Dong Liu, Ronald P. Fedkiw, and Myungjoo Kang. A Boundary Condition Capturing Method for Poisson’s Equation on Irregular Domains. *Journal of Computational Physics*, 160(1):151–178, May 2000.

- [28] Xu-Dong Liu and Thomas Sideris. Convergence of the ghost fluid method for elliptic equations with interfaces. *Mathematics of computation*, 72(244):1731–1746, 2003.
- [29] R. E. Lynch, J. R. Rice, and D. H. Thomas. Tensor product analysis of partial difference equations. *Bulletin of the American Mathematical Society*, 70(3):378–384, May 1964.
- [30] Maplesoft, a division of Waterloo Maple Inc. Maple 17.02, 2013.
- [31] Alexandre Noll Marques, Jean-Christophe Nave, and Rodolfo Ruben Rosales. A Correction Function Method for Poisson problems with interface jump conditions. *Journal of Computational Physics*, 230(20):7567–7597, August 2011.
- [32] Alexandre Noll Marques, Jean-Christophe Nave, and Rodolfo Ruben Rosales. High order solution of Poisson problems with piecewise constant coefficients and interface jumps. *Journal of Computational Physics*, 335:497–515, April 2017.
- [33] Alexandre Noll Marques, Jean-Christophe Nave, and Rodolfo Ruben Rosales. Imposing jump conditions on nonconforming interfaces via least squares minimization. *arXiv:1710.11016 [physics]*, October 2017. arXiv: 1710.11016.
- [34] Rajat Mittal and Gianluca Iaccarino. Immersed Boundary Methods. *Annual Review of Fluid Mechanics*, 37(1):239–261, 2005.
- [35] Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. A gradient-augmented level set method with an optimally local, coherent advection scheme. *Journal of Computational Physics*, 229(10):3802–3827, May 2010.
- [36] Stanley Osher. *Level set methods and dynamic implicit surfaces*. Springer, Place of publication not identified, 2012. OCLC: 878109841.
- [37] Charles S Peskin. Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10(2):252–271, October 1972.
- [38] Charles S Peskin. Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, 25(3):220–252, November 1977.
- [39] Charles S. Peskin. The immersed boundary method. *Acta Numerica*, 11, January 2002.

- [40] Peter Eastman. SimTK: Lepton Mathematical Expression Parser: Project Home, November 2009.
- [41] Peter L Hagelstein. Introduction to Numerical Modeling in Engineering and Applied Physics (Manuscript). 2013.
- [42] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*. Number 37 in Texts in applied mathematics. Springer, Berlin ; New York, 2nd ed edition, 2007.
- [43] L. F. Shampine. Matlab program for quadrature in 2d. *Applied Mathematics and Computation*, 202(1):266–274, August 2008.
- [44] L. F. Shampine. Vectorized adaptive quadrature in MATLAB. *Journal of Computational and Applied Mathematics*, 211(2):131–140, February 2008.
- [45] Chi-Wang Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, Lecture Notes in Mathematics, pages 325–432. Springer, Berlin, Heidelberg, 1998.
- [46] James G Simmonds. *A Brief on tensor analysis*. Springer, New York, 2013. OCLC: 830021437.
- [47] L. Trefethen. *Spectral Methods in MATLAB*. Software, Environments and Tools. Society for Industrial and Applied Mathematics, January 2000.
- [48] Salih Ozen Unverdi and Grétar Tryggvason. A front-tracking method for viscous, incompressible, multi-fluid flows. *Journal of Computational Physics*, 100(1):25–37, May 1992.
- [49] Yushan Wang. *Solving incompressible Navier-Stokes equations on heterogeneous parallel architectures*. Ph.D. Thesis, Université Paris Sud - Paris XI, April 2015.
- [50] Sheng Xu and Z. Jane Wang. An immersed interface method for simulating the interaction of a fluid with moving boundaries. *Journal of Computational Physics*, 216(2):454–493, August 2006.

- [51] Chaoming Zhang and Randall J. LeVeque. The immersed interface method for acoustic wave equations with discontinuous coefficients. *Wave Motion*, 25(3):237–263, May 1997.