Abstract

DENG, SHAOZHONG. Immersed Interface Method for Three Dimensional Interface Problems and Applications. (Under the direction of Drs. Zhilin Li and Kazufumi Ito.)

This thesis describes a maximum principle preserving scheme and a fast algorithm for three-dimensional elliptic interface problems, in which the partial differential equations have discontinuities and singularities in the coefficients and the solutions. Such problems arise in many physical applications.

The immersed interface method (IIM) was developed in [46] and is designed for elliptic equations having discontinuous coefficients and singular source terms. This method is second order accurate and has been applied to many problems in one or two dimensions. In this thesis, we first pursue the extension of the IIM method to three dimensions. Then based on the IIM method, we present a maximum principle preserving scheme for arbitrary coefficients in three dimensions using direct finite difference discretization. The new scheme satisfies the sign property that guarantees the discrete maximum principle. The sign property is enforced through a constrained quadratic optimization problem. The Successive Overrelaxation method (SOR) or the Algebraic Multigrid method (AMG) can then be used to solve the resulting system of linear equations. Numerical experiments confirm the expected second order accuracy.

We also present a second order fast algorithm for three-dimensional elliptic equations with piecewise constant coefficients. Before applying the IIM method, we precondition the differential equation. In order to take advantage of existing fast Poisson solvers, an intermediate unknown function, the jump in the normal derivative of the solution across the interface, is introduced. Then the Generalized Minimal Residual method (GMRES) is employed to solve the Schur complement system derived from the discretization. Numerical experiments show that the fast algorithm is very efficient. Especially, the number of iterations in solving the Schur complement system is independent of the mesh size.

We then investigate some applications of the fast algorithm. We develop an embedding technique to solve interior or exterior Poisson equations with Dirichlet or Neumann boundary conditions. Then we investigate how to use the fast algorithm to solve an inverse interface problem.
IMMERSED INTERFACE METHOD FOR THREE DIMENSIONAL INTERFACE PROBLEMS AND APPLICATIONS

BY

SHAOZHONG DENG

A DISSERTATION SUBMITTED TO THE GRADUATE FACULTY OF NORTH CAROLINA STATE UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF MATHEMATICS

RALEIGH
JANUARY 04, 2001

APPROVED BY:

________________________________________  ________________________________________
CO-CHAIR OF ADVISORY COMMITTEE        CO-CHAIR OF ADVISORY COMMITTEE
To my wife and parents

for their endless love
Biography

Shaozhong Deng was born on March 29, 1966 in Zhangjiajie, Hunan Province, the Peoples Republic of China. He received his elementary and secondary education in Zhangjiajie, Hunan and graduated from Xiangxi Mingzu High School in 1984. He received his Bachelor of Science degree in Computational Mathematics from Xi’an Jiaotong University in 1988, and his Master of Science degree in Computational Mathematics from Nanjing University of Aeronautics and Astronautics in 1991. He then joined the faculty of the School of Science at Nanjing University of Aeronautics and Astronautics in 1991, as a lecturer. He received the Science and Technology Progress Award from the Aviation Industries of China in 1997, and the Science and Technology Progress Award from Jiangsu Province in 1996. With Shuquan Zhou, he published *Parallel Computation for Finite Element Structural Analysis* by Science Press, Beijing, in 1994.

In the fall of 1997, he entered the Ph.D program in Applied Mathematics with a concentration in Computational Mathematics at North Carolina State University at Raleigh. He also received his Master of Science degree in Computer Science from North Carolina State University in 1999.
Acknowledgements

I am very grateful for my thesis advisor, Dr. Zhilin Li for his invaluable assistance, guidance, encouragement and friendship throughout my Ph.D study and thesis project. It is an enjoyable experience working with him. Our innumerable times discussing and questioning the problems, and cheering any achievements together will leave a lasting impression.

I would also like to extend my appreciation to Dr. Kazufumi Ito, Co-Chairman of my advisory committee, and my advisory committee members, Dr. Pierre A. Gremaud and Dr. Ralph C. Smith for their directions, comments and suggestions.

Sincere thanks are also extended to Dr. Hongkai Zhao at UCLA for helping me on level set programs, and Dr. Ernest L. Stitzinger, Graduate Administrator of Graduate Programs, and Ms. April O. Jackson, Graduate Secretary of Graduate Programs in the Department of Mathematics, for their generous help throughout my study at NCSU.

In addition, I would like to thank my family members back home in China, especially my parents, Xuechun Deng and Zhangci Wang, for their long-time confidence and encouragement. Finally and most importantly, I would like to express my utmost appreciation to my wife, Qunhui Guo, for her constant love, understanding and support.
# Table of Contents

List of Tables vii

List of Figures ix

1 Introduction 1

1.1 Some examples .................................................. 1
1.2 Elliptic interface problems ........................................ 3
1.3 Numerical aspects of interface problems ......................... 6
  1.3.1 Finite difference methods and uniform Cartesian grids ...... 6
  1.3.2 Numerical difficulties ......................................... 7
  1.3.3 Interface expressions ......................................... 8
  1.3.4 Overview of numerical algorithms for interface problems .. 9
1.4 Outline of the thesis ............................................. 14

2 A Maximum Principle Preserving Scheme for 3D Interface Problems 15

2.1 Introduction .................................................... 15
2.2 The algorithm description ....................................... 16
  2.2.1 The local coordinate transformation ........................ 19
  2.2.2 Interface relations .......................................... 22
  2.2.3 The finite difference scheme ................................. 25
  2.2.4 A special case .............................................. 29
  2.2.5 Representation of the interface ............................. 32
  2.2.6 Choosing $\xi$, $\eta$ and $\tau$ ................................. 34
  2.2.7 Finding projections .......................................... 35
  2.2.8 The interpolations .......................................... 35
  2.2.9 An optimization approach .................................. 37
  2.2.10 An algebraic multigrid solver .............................. 39
2.3 Numerical results .............................................. 40
  2.3.1 Algorithm efficiency analysis .............................. 47
  2.3.2 Summary of the maximum principle preserving scheme ...... 49
3 A Fast Algorithm for 3D Elliptic Interface Problems 51
3.1 Introduction ...................................... 51
3.2 The algorithm description .......................... 53
  3.2.1 Preconditioning the PDE to an equivalent problem ............ 53
  3.2.2 Representation of the interface and jump conditions .......... 54
  3.2.3 Discretization ................................ 55
  3.2.4 Updating the jump in the normal derivative ................. 56
  3.2.5 A least square approach for computing an interface quantity from a grid function ........................................ 58
  3.2.6 A least square approach for computing an interface quantity from its values at control points ......................... 63
  3.2.7 Invertibility of the Schur complement system ............... 64
  3.2.8 Some implementation details .......................... 65
  3.2.9 Summary of the fast algorithm ........................ 67
3.3 Numerical results .................................. 68
  3.3.1 The algorithm efficiency analysis ........................ 74

4 Applications 76
4.1 Solving Poisson equations on irregular domains using the fast algorithm ........ 76
4.2 Solving an inverse interface problem using the fast algorithm ............ 85
  4.2.1 The numerical algorithm ................................ 86
  4.2.2 Evaluation of gradient vectors at irregular grid points .......... 89
  4.2.3 Extension of the velocity ................................ 90
  4.2.4 The level set method .................................. 90
  4.2.5 The time step ....................................... 92
  4.2.6 Reinitialization ...................................... 92
  4.2.7 Numerical examples .................................... 93

5 Conclusions 102

List of References 104

A Level set function for Example 3.5 110
List of Tables

2.1 The grid refinement analysis for Example 2.1. 41
2.2 The grid refinement analysis for Example 2.2. 42
2.3 The grid refinement analysis for Example 2.3. 45
2.4 The grid refinement analysis for Example 2.4. 47
2.5 CPU time for dealing with irregular grid points. 48
2.6 CPU time for Example 2.1. 48
2.7 CPU time for Example 2.2. 49
2.8 CPU time for Example 2.3. 49
2.9 CPU time for Example 2.4. 49
2.10 Comparison of CPU time between the SOR method and the AMG method. 50

3.1 The grid refinement analysis for Example 3.1. 69
3.2 The grid refinement analysis for Example 3.2. 70
3.3 The grid refinement analysis for Example 3.3. 71
3.4 The grid refinement analysis for Example 3.4. 72
3.5 The grid refinement analysis for Example 3.5. 74
3.6 CPU time and the number of iterations for Example 3.1. 75
3.7 CPU time and the number of iterations for Example 3.2. 75
3.8 CPU time and the number of iterations for Example 3.3. 75
3.9 CPU time and the number of iterations for Example 3.4. 75
3.10 CPU time and the number of iterations for Example 3.5. 75

4.1 The grid refinement analysis for Example 4.1. 84
4.2 The grid refinement analysis for Example 4.2. 84
List of Figures

1.1 A rectangular domain $\Omega = \Omega^+ \cup \Omega^-$ with an immersed interface $\Gamma$. The coefficients $\beta, \lambda$ and the source term $f$ may have jumps across the interface. 4

2.1 A sketch of a three-dimensional local coordinate transformation. 20
2.2 (a) A slice of the computed solution ($\beta^+ = 1000$, $\beta^- = 1$ and $n = 52$) for Example 2.1: $-u(x, y, 0)$. (b) The error plot of the slice of the computed solution. 41
2.3 A slice of the computed solution ($n = 52$) for Example 2.2: $-u(x, y, 0)$. (a) $b = 1$. (b) $b = 10$. (c) $b = 1000$. 43
2.4 A slice of the computed solution ($\beta^+ = 1000$, $\beta^- = 1$ and $n = 52$) for Example 2.3: $u(x, y, 0)$. 45
2.5 The interface of two ellipsoids for Example 2.4. 46
2.6 A slice of the computed solution ($\beta^+ = 1000$, $\beta^- = 1$ and $n = 104$) for Example 2.4: $u(x, y, 0)$. 47

3.1 A slice of the computed solution ($\beta^+ = 2$, $\beta^- = 1$ and $n = 52$) for Example 3.1: $-u(x, y, 0)$. 68
3.2 A slice of the computed solution ($n = 52$) for Example 3.2: $-u(x, y, 0)$. (a) $\beta^+ = 1$, $\beta^- = 1$. (b) $\beta^+ = 10$, $\beta^- = 1$. (c) $\beta^+ = 1000$, $\beta^- = 1$. 70
3.3 A slice of the computed solution ($\beta^+ = 2$, $\beta^- = 1$ and $n = 52$) for Example 3.3: $u(x, y, 0)$. 71
3.4 A slice of the computed solution ($\beta^+ = 1000$, $\beta^- = 1$ and $n = 104$) for Example 3.4: $u(x, y, 0)$. 72
3.5 The interface for Example 3.5. (a) The interface. (b) A slice of the interface. 73
3.6 A slice of the computed solution ($\beta^+ = 2$, $\beta^- = 1$ and $n = 81$) for Example 3.5: $u(x, y, 0)$. 73

4.1 A diagram for an interior Poisson problem. 77
4.2 A diagram for an exterior Poisson problem. 79
4.3 A slice of the computed solution ($n = 80$) for Example 4.1: $-u(x, y, 0)$. 80
4.4 A slice of the computed solution ($n = 80$) for Example 4.2: $u(x, y, 0)$. 81
4.5 A slice of the computed solution ($n = 80$) for Example 4.3: $u(x, y, 0)$. 82
4.6 A slice of the computed solution ($n = 41$) for Example 4.4: $u(x, y, 0)$. 83
4.7 A diagram of the forward interface problem. 85
4.8 A diagram of an inverse interface problem of identifying an unknown shape from the observed data taken from the shaded area $\hat{\Omega}$.  

4.9 Velocity extension. (a) A slice of the computed velocity at irregular grid points. (b) A slice of the velocity field in a computational tube after the extension.

4.10 The computed shape for Example 4.5 using $60 \times 60 \times 60$ grid and $\mu = 0.001$ at different stages. (a) The initial guess. (b) After 11 iterations. (c) After 31 iterations. (d) After 51 iterations.

4.11 The $x$-$y$ transection of the computed shape for Example 4.5 using $60 \times 60 \times 60$ grid and $\mu = 0.001$ at different stages. (a) The initial guess. (b) After 11 iterations. (c) After 31 iterations. (d) After 51 iterations. The inner-most circle is the exact solution. The interface stays smooth for all time as it evolves.

4.12 The $x$-$y$ transection of the computed shape for Example 4.5 using $60 \times 60 \times 60$ grid but $\mu = 0$ at different stages. (a) The initial guess. (b) After 11 iterations. (c) After 31 iterations. (d) After 51 iterations. The inner-most circle is the exact solution. The interface gradually forms a corner as it moves.

4.13 (a) The exact shape for Example 4.6. (b) The computed shape for Example 4.6 with $60 \times 60 \times 60$ grid after 201 iterations.

4.14 The computed shape for Example 4.6 using $60 \times 60 \times 60$ grid at different stages. (a) The initial guess. (b) After 41 iterations. (c) After 121 iterations. (d) After 201 iterations. The interface does not stay smooth for all time as it evolves since no regularization is used.

4.15 The $y$-$z$ transection of the computed shape for Example 4.6 using $60 \times 60 \times 60$ grid at different stages. (a) The initial guess. (b) After 121 iterations. (c) After 201 iterations. The inner-most ellipse is the exact solution.

4.16 (a) The exact shape for Example 4.7. (b) The computed shape which is an envelope of the two objects for Example 4.7 with $60 \times 60 \times 60$ grid after 241 iterations.

4.17 The computed shape for Example 4.7 using $60 \times 60 \times 60$ grid at different stages. (a) The initial guess. (b) After 41 iterations. (c) After 161 iterations. (d) After 241 iterations.

4.18 The $y$-$z$ transection of the computed shape for Example 4.7 using $60 \times 60 \times 60$ grid at different stages. (a) The initial guess. (b) After 41 iterations. (c) After 161 iterations. (d) After 241 iterations. The two small circles are the $y$-$z$ transections of the two exact objects.
Chapter 1

Introduction

The interface problems discussed in this thesis are those problems in which the input data (such as the coefficients of differential equations, source terms etc.) may be discontinuous or singular across one or several interfaces which are one-dimension lower than the spaces where the problems are defined. The solutions to the problems, therefore, may be non-smooth or even discontinuous across those interfaces.

As the result, many standard numerical methods based on the assumption of smoothness of the solutions do not work efficiently or do not work at all for interface problems. Over the past years, interface problems have attracted a lot of attentions from numerical analysts. A variety of numerical methods for various interface problems have been presented. Among the new methods, the immersed interface method (IIM) \[43, 46\] is a second order method and has been successfully applied to many problems. It has already been augmented with fast solution methods such as multigrids [1] and the Generalized Minimal Residual method (GMRES) [49], and with the maximum principle preserving scheme [50]. This thesis pursues the extension of the IIM method, the fast algorithm and the maximum principle preserving scheme to three dimensions.

Below we first present two examples in physics to show the characteristics of interface problems. We then present general three-dimensional elliptic interface problems, which we will focus on in this thesis. We then discuss some numerical aspects of interface problems including an overview of existing numerical approaches. Lastly we outline the thesis work at the end of this chapter.

1.1 Some examples

The first example is the well-known heat equation

\[ u_t = \nabla \cdot (\beta \nabla u) + f, \quad \text{in} \quad \Omega, \]  

(1.1)
Chapter 1. Introduction

It describes many physical phenomena. For example, $u$ may represent the temperature distribution in a material with heat conductivity $\beta$ and a heat source $f$. For a composite material, the coefficient $\beta$ is discontinuous across the interface between two different materials. As a common sense, the temperature $u$ should be continuous, which means $[u] = 0$ across the interface, where $[ \cdot ]$ denotes the jump in a quantity across the interface. The heat flux $\beta u_n$ should also be continuous across any interface, i.e., $[\beta u_n] = 0$, if no heat source is present there, where $n$ denotes the unit normal vector at the interface. If $\beta$ is discontinuous, then there must be a jump in the normal derivative $u_n$.

Mathematically, the temperature $u$ can be modeled as the following

$$u_t = \nabla \cdot (\beta \nabla u), \quad \text{in } \Omega,$$

$$\beta = \begin{cases} 
\beta^+, & \text{in } \Omega^+, \\
\beta^-, & \text{in } \Omega^-, 
\end{cases}$$

Boundary conditions on $\partial \Omega$,

Initial conditions,

Jump conditions on $\Gamma$: $[u] = 0$, $[\beta u_n] = 0$,

where $\Omega^+$ and $\Omega^-$ denote the two sides of the interface, and $\Gamma$ denotes the interface between $\Omega^+$ and $\Omega^-$.  

The second example is the Hele-Shaw flow [31, 32, 70]. In 1958, Saffman and Taylor performed experiments replacing a viscous fluid from between two closely spaced, parallel plates with a less viscous fluid. The shape of the interface is well known to exhibit a fingering phenomenon. The velocity $u=(u, v)$ of the flow is proportional to the gradient of the pressure $p$. The non-dimensional form of the governing equation is

$$u = -\beta \nabla p,$$

$$\nabla \cdot u = f,$$  

with $\beta = b^2/(12\nu)$, where $b$ is the gap width and $\nu$ is the viscosity, which is very different inside and outside the interface separating the two fluids. The source term $f$ is the result of the injection of the less viscous fluid into the Hele-Shaw cell. For example, we may set

$$f = \begin{cases} 
f_0(r_0)(1 + \cos(r\pi/r_0)), & \text{if } r \leq r_0, \\
0, & \text{if } r > r_0,
\end{cases}$$

where $r = \sqrt{x^2 + y^2}$. The total injection rate is
Chapter 1. Introduction

\[
\bar{f} = \int \int f(x, y) dxdy = f_0(r_0) \left( \pi - \frac{4}{\pi} \right) r_0^2.
\]

Specifically, if \( f_0(r_0) = 1/r_0^2 \), then we have a single point source at the origin as \( r_0 \) approaches zero. The jump conditions across the interface are

\[
\begin{align*}
\left[ p \right] &= \varsigma \kappa, \quad \text{the Laplace-Young condition,} \\
\left[ \beta p_n \right] &= 0, \quad \text{the kinematic interface condition,}
\end{align*}
\]

where \( \varsigma \) is the surface tension and \( \kappa \) is the curvature of the interface.

To determine the boundary condition on the pressure, we assume that the interface is far away from the boundary so that the flow at the boundary agrees with the radial outflow which would arise from the source term in a uniform fluid, i.e.,

\[
p(x, y) = p_0 - \frac{f_0}{2\pi \beta} \log r
\]

is specified on the boundary, where \( p_0 \) is some arbitrary constant. Therefore, by combining (1.2) and (1.3) together, we have the following mathematical description of the problem

\[
\nabla \cdot (\beta \nabla p) = -f,
\]

\[
\beta = \begin{cases} 
\frac{b^2}{12\nu^+}, & \text{if } (x, y) \text{ is in the outside of the interface,} \\
\frac{b^2}{12\nu^-}, & \text{if } (x, y) \text{ is in the inside of the interface,}
\end{cases}
\]

Boundary condition: \( p(x, y) = p_0 - \frac{f_0}{2\pi \beta} \log r, \)

Jump conditions: \( \left[ p \right] = \varsigma \kappa, \quad \left[ \beta p_n \right] = 0. \)

More physical applications of interface problems can be found in two-phase flow [6, 16, 73], drop spreading [34] and electro-migration of voids [11, 54, 63], etc.

1.2 Elliptic interface problems

This thesis is concerned with numerical analysis of elliptic interface problems in three dimensions. Let \( \Omega \) be a simple convex domain in \( \mathbb{R}^3 \) which is separated into two sub-domains \( \Omega^+ \) and \( \Omega^- \) by an irregular surface \( \Gamma \) (hereafter called an interface) such that \( \Omega = \Omega^+ \cup \Omega^- \).

We consider the elliptic equation

\[
\nabla \cdot (\beta \nabla u) + \lambda u = f, \quad \text{in } \Omega,
\] (1.4)
Chapter 1. Introduction

with some boundary conditions on $\partial \Omega$.

Furthermore, we assume that the coefficients $\beta, \lambda$ and the source term $f$ may be discontinuous across the interface, say,

$$\beta(x) = \beta^+(x), \quad \lambda(x) = \lambda^+(x), \quad f(x) = f^+(x), \quad \text{if} \quad x \in \Omega^+,$$

and

$$\beta(x) = \beta^-(x), \quad \lambda(x) = \lambda^-(x), \quad f(x) = f^-(x), \quad \text{if} \quad x \in \Omega^-,$$

see Figure 1.1. Moreover, the source term $f$ may also have a $\delta$-function singularity as in the form

$$f(x) = f_c(x) + \int_{\Gamma} C(X) \delta(x - X) dS, \quad (1.5)$$

where $f_c$ is a bounded piecewise smooth function and $C(X)$ is the source strength on the interface and $\delta$ is the Dirac $\delta$-function.

\[\text{Figure 1.1: A rectangular domain } \Omega = \Omega^+ \cup \Omega^- \text{ with an immersed interface } \Gamma. \text{ The coefficients } \beta, \lambda \text{ and the source term } f \text{ may have jumps across the interface.}\]

Due to the discontinuities in the coefficients and/or singularity in the source term, the solution of (1.4) or some of its derivatives will generally be non-smooth or even discontinuous across the interface. For example, if the coefficient $\beta$ is discontinuous across the interface, while $\lambda$ and $f$ are continuous, then $u$ and $\beta u_n$ will be continuous but the normal derivative $u_n$ will be discontinuous [37]. Such problems are often encountered in material sciences and fluid dynamics when two distinct materials or fluids with different conductivities or densities or diffusions are involved.
Chapter 1. Introduction

If \( \beta \) is continuous but instead \( f \) has a \( \delta \)-function singularity along the interface as we mentioned in the above, then the solution \( u \) will be continuous but the flux will have a discontinuity of magnitude \( |C(X)| \). Sometimes, a dipole source may also occur, in which \( f \) contains the derivative of the \( \delta \)-function, and as a result the solution \( u \) itself is also discontinuous across the interface.

Generally, the interface problems with bounded solutions can be decomposed into one or several regions whose size and shape may change with time. The solutions in different regions are continuously differentiable to a certain degree and they are coupled by some interface conditions, generally jump conditions across the interfaces. It is crucial for our approach that we have some priori knowledge about some interface conditions. Two commonly used interface conditions are of the form

\[
\begin{align*}
[u] &= w, \\
[\beta u_n] &= q.
\end{align*}
\]

In general, these two jump conditions can be derived either by physical reasoning or directly from the differential equation itself.

The discussion of the existence and the regularity of the solution can be found, for example, in [2, 14]. In general, when the interface is smooth enough, the solution of the interface problem is also very smooth in individual regions, but the global regularity is usually very low. In particular, in the case that \( \beta, \lambda \) are piecewise smooth and \( w \equiv 0 \), if the interface \( \Gamma \) is \( C^2 \) smooth, \( f \in L^2(\Omega) \), and \( q \in H^{1/2}(\Omega) \), then the elliptic interface problem has a unique solution \( u \in H^1(\Omega) \cap H^2(\Omega^+) \cap H^2(\Omega^-) \). More solvability and regularity results of the above interface problems can be found in [39, 40, 41].

On the other hand, our goal in this thesis is to develop some finite difference schemes for elliptic interface problems. As a common practice in the study of finite difference methods, we assume that the interface \( \Gamma \) is arbitrarily smooth. We also assume that in \( \Omega \), \( \beta \) is piecewise smooth, \( \lambda \) and \( f \) are piecewise continuous, and along the interface, \( w \) has continuous second derivatives and \( q \) has continuous first derivatives. Then the solution \( u \) is also piecewise smooth and has continuous piecewise second derivatives, i.e., \( u \in C^2 \) in \( \Omega^+ \) or \( \Omega^- \), but not in \( \Omega \).

A vast collection of applications involve solving (1.4), for example, the projection method for solving the Navier-Stokes equation [6, 16, 73] involving two-phase flow, the Hele-Shaw flow [31, 32] and many others.
1.3 Numerical aspects of interface problems

1.3.1 Finite difference methods and uniform Cartesian grids

Basically, we still wish to solve interface problems using finite difference methods on uniform Cartesian grids. By using uniform Cartesian grids, there is almost no cost in the grid generation since Cartesian grids are fixed. This is very significant for moving interface problems, or problems with complicated geometries, or problems with topological changes, because the grid generation process may be the most expensive part in an entire simulation.

Another obvious advantage of this approach is that the conventional finite difference schemes can be used at most grid points (regular grid points) which are away from the interface(s) since there are no irregularities there, and only those irregular grid points near the interface(s), which are much fewer than the regular grid points, need special attention.

In particular, later we will see that for the equation (1.4), if $\beta$ is constant and only $f$ is discontinuous or singular as in the form of (1.5), then we can use the standard seven point stencil to discretize the left-hand side of (1.4), and only modify the right-hand side to get a second order scheme [43, 46, 49]. So the cost in solving (1.4) is just a little more than that in solving a regular elliptic equation with a smooth solution. Especially, since $\beta$ is constant, the existing fast Poisson solvers can still be used to solve the system on a uniform grid, an advantage that would be lost on an irregular grid. Even if $\beta$ is discontinuous so that the coefficients in the linear system must be modified, the system maintains the same block structure as in the continuous case. One can then use available softwares designed to accept a user-specified stencil on a uniform rectangular grid.

Certainly there are many other ways to discretize interface problems. Using a grid that conforms to the interface is an obvious alternative, for example a structured grid that is deformed in the neighborhood of the interface (e.g. [9]) or an unstructured triangulation. The finite element method on such a grid would be a natural choice for elliptic equations, and can be used very successfully (e.g., [4]). However, in many contexts the use of a uniform grid may be preferable.

More importantly, one is interested primarily in time-dependent problems, and the interfaces are typically moving. Although it is possible to develop moving mesh methods that conform to the interfaces in each time step, this is generally much more complicated than simply allowing the interface to move relative to a fixed underlying uniform grid. For example, the immersed boundary method (IBM) has been very successful in modeling flow in very complicated time-dependent geometries such as the beating heart with valves opening and closing. This would be difficult, if not impossible, to do with grids that conform to the boundary.

Another advantage using Cartesian grids is that we can take advantage of many software
packages or methods developed for Cartesian grids, for example, fast Poisson solvers [74], Clawpack [42], Amrclawpack [7], the level set method [65], algebraic multigrid solvers [1, 17], and many others.

1.3.2 Numerical difficulties

The most noticeable characteristic of interface problems is the discontinuity or non-smoothness in the solutions which is the result of discontinuities of the coefficients or singularities of the sources in the corresponding differential equations. This brings up several substantial difficulties in the numerical analysis process for interface problems.

- **Discretization.** In a uniform Cartesian grid, the interface is typically not aligned with the grid but rather cuts between grid points so that for grid points near the interface the stencil of a standard finite difference scheme will contain points from both sides of the interface. Due to the non-smoothness of \( u \), differencing \( u \) across the interface using standard finite difference schemes will not produce accurate approximations to derivatives of \( u \), and hence a naive discretization will produce results with low accuracy. Instead, special care has to be taken to discretize the discontinuous coefficients. Many current techniques such as the harmonic averaging or coefficient smoothing method [73] fail to give higher order accuracy in two or higher space dimensions.

Sometimes, as we mentioned, we may still use the standard finite difference stencil to discretize the left-hand side of (1.4), however in general we do need to modify the standard finite difference scheme to get a second order scheme. For example, in two-dimensional cases, for each grid point near the interface at least six points must be involved in the modified scheme [43, 46, 49], and in three-dimensional cases, we will see that at least ten points must be included.

- **Arbitrary interfaces.** Generally the interfaces can be arbitrary and complicated, and analytical expressions for them are rarely available. Moreover, there are situations in which the interfaces may develop cusps and spikes, change topology, and break or merge.

- **Error analysis.** Because of the discontinuity and non-smoothness in the solution and the complexity of the interfaces, it is difficult to perform convergence analysis in the conventional way.

- **Solving the system of discrete equations.** Due to the presence of the interfaces and the discontinuity or non-smoothness in the solution, the system of discrete equations may lose many nice properties such as symmetry, positive definiteness, and diagonal
dominance, etc. The structure of the linear system may be very different from regular problems making it hard to use multigrid solvers or other efficient solvers.

1.3.3 Interface expressions

To solve interface problems numerically, we need the information about the interface such as the position, tangential and normal directions, and sometimes curvatures as well. Some common approaches to express the interface are the following.

- **Analytic expression.** If the interface is fixed, we may have an analytic expression for the interface. However, it can still be difficult to calculate other information needed such as first derivatives to determine the tangential and normal directions, and second derivatives to determine the curvatures, etc., if the analytic expression is too complicated. Then discrete methods to calculate those quantities to a certain accuracy are needed.

- **Discrete parameterization and interpolation.** Very often we only know the coordinates of a number of control points on the interface, say \( \mathbf{X}_n = (X_n, Y_n, Z_n), \ n = 1, 2, \ldots, m, \) in three space dimensions. There are different ways to get derivative information on the interface. The first approach is to use discrete difference formulas such as the central difference to get the required derivatives. This approach has been widely used in implementing the immersed boundary method for many problems. However we must balance the needs of accuracy and stability in this approach. Generally higher order accurate difference formula, or too many control points, will destabilize the algorithm and worsen the condition of the resulting system of linear equations. This approach seems to be unable to handle the situations when the interface develops cusps and spikes or when the interface breaks or merges.

A different approach is used in \([43, 46, 49]\), etc. The idea is to use piecewise interpolation, mostly cubic splines, to get an analytic expression of the interface, and then calculate all the information about the interface from the analytic expression of the interpolated interface. This approach works very well for many problems including Stokes flow with a moving interface. One advantage is that one can take relatively few control points on the interface if the interface is smooth. Moreover, other quantities such as the force strength, jumps, etc., can be calculated with the same parameter as used in the interpolation formula. Although it may be difficult to implement, this approach can handle cusps and spikes and even situations when the interface breaks or merges. This approach works fine in one or two dimensions, but it appears to be extremely difficult to implement in three dimensions.
• **Level set approach.** This approach was introduced by Osher and Sethian in [65] and has been applied for many moving interface problems (e.g., [13, 15, 31, 50, 73, 78]) since then. In this approach, the interface is modeled as the zero level set of a smooth function $\phi$ defined on the entire physical domain. The interface is then moved by updating values of $\phi$ through a non-linear Hamilton-Jacobi equation of the level set function. By the narrow band approach [15], the Hamilton-Jacobi equation is solved only over a computational tube surrounding the interface, and only the values of $\phi$ within the narrow tube are updated.

This approach does not rely on a discrete parameterization of the interface and can be used for complicated moving interfaces in two and three dimensions. It can handle cusps and spikes and situations in which the interfaces break or merge.

So depending on the knowledge of the physical problem, we can choose a suitable method to express the interface. We choose to use the level set approach in this thesis since we focus on three-dimensional interface problems.

### 1.3.4 Overview of numerical algorithms for interface problems

Solving interface problems efficiently and accurately has been a challenge because of many irregularities associated with them. Over years, interface problems have attracted a lot of attention from numerical analysts, and many numerical methods have been presented. Below is an incomplete brief review of those related to this thesis.

• **The immersed boundary method.**

The immersed boundary method (IBM) was developed by Peskin [66, 67] to model blood flow in the heart, and has since been used for many other problems, particularly in biophysics [8, 19, 20, 24, 26]. The approach can be applied to the elliptic interface problem (1.4) as well. The main idea is to discretize the interface by a set of control points $\mathbf{X}_n = (X_n, Y_n, Z_n)$, $n = 1, 2, \ldots, m$, and replaces the integral in (1.5) by a discrete sum, also replacing the $\delta$-function by some discrete approximation $\delta_h(x)$ with support related to the mesh width $h$. For example, in a three dimension case the discrete form of (1.5) is

$$f_{ijk} = (f_c)_{ijk} \sum_{n=1}^{m} C(\mathbf{X}_n) \delta_h(x_i - X_n) \delta_h(y_j - Y_n) \delta_h(z_k - Z_n) \Delta S.$$  \hspace{1cm} (1.8)

In three space dimensions, however, it does not appear to be possible to achieve second order accuracy at all grid points. One explanation is that basically, the IBM method uses discrete $\delta$-functions which smear out the solution on a thin finite band surrounding the interface. Although this gives us some flexibility, the numerical smearing at
the interface has an adverse effect on the solution forcing continuity at the interface regardless of the appropriate interface conditions.

- **Smoothing methods for discontinuous coefficients.**

  For problems with discontinuous coefficients, a simple approach is to smooth out the coefficients. One commonly used method is the *harmonic averaging* method, see [5, 71, 75]. In one-dimensional case, this approach is second order accurate if the involved integrals are calculated to $O(h^3)$, which is not so easy especially in the interval where the discontinuity takes place.

  The harmonic averaging method may also be used in two or three space dimensions to deal with discontinuous coefficients, see [5, 71] for examples. It will involve an integration over square or cubic domains. In these cases, however, the method does not appear to give second order accurate results. It is also not practical to compute the integrals to $O(h^3)$ at irregular grid points in two or three dimensions especially when $\beta$ is discontinuous across the interfaces.

  The level set expression of the interfaces makes the smoothing method much easier for two- and three-dimensional problem. For example, [73] presents an incompressible two-phase flow algorithm. Formally the algorithm is second order accurate and the final linear system remains symmetric. Also, it can be generalized to three dimensions. However, similar to the IBM method, the solution is also smeared out by the smoothing method.

- **The immersed interface method.**

  The immersed interface method (IIM) [43, 46] was designed for elliptic equations having discontinuous coefficients and singular source terms due to interfaces in the solution domain. The essence of the IIM method includes:  
  (a) using uniform or adaptive Cartesian grids, and finite difference methods; (b) immersing the interface(s) in the Cartesian grids; (c) applying the standard finite difference scheme at regular grid points which are away from interface(s); (d) deriving the interface relations across the interface(s) based on the known jump conditions, generally in the solutions and fluxes; (e) taking into account the interface relations to modify the finite difference scheme at irregular grid points which are near the interface; (f) introducing non-zero correction terms to balance the singular source terms.

  Unlike the IBM method and smoothing methods, the IIM method gives sharp solution across the interface since the jump conditions are enforced in the algorithm and therefore preserved at the interface in contrast to the numerical smearing introduced by the $\delta$-function formulation of the IBM method or coefficient smoothing of smoothing methods. Generally the IIM method uses only local information, specifically, the partial differential equations, the jump conditions, the interface, and the underlying grid.
Chapter 1. Introduction

It is a second order accurate method, and this is either proved or confirmed in further development of the method and applications. The method has been applied to problems ranging from one-, two- to three-dimensional problems [47], elliptic, parabolic [76], hyperbolic [45] and mixed type equations [55], fixed and moving interfaces [48], and many applications [12, 25, 31, 33, 34, 44, 52, 53, 54].

- Fast solvers based on boundary integral equations.
  Based on integral equations, some fast solvers are available for Poisson equations with piecewise constant coefficients and other problems [28, 57, 59, 60]. In these methods, integral equations are setup at some points on the interfaces and boundaries for unknown source strength, and the solutions then can be found using fast boundary integral techniques. Non-homogeneous source terms can be decomposed as two homogeneous problems.

  Mayo and Greenbaum [58, 59] have derived an integral equation for elliptic interface problems of the form (1.4) with piecewise constant coefficients. By solving the integral equation, they can solve such interface problems to second order accuracy in \( L_\infty \) norm using the techniques developed by Mayo in [57, 58] for solving Poisson and biharmonic equations on irregular regions. Basically, the region is embedded in a regular region where a fast solver can be used on a uniform grid and the right-hand side is appropriately modified near the original boundary. The total cost includes solving the integral equation and a regular Poisson equation using a fast solver, so this gives a fast algorithm. The possibility of extension to variable \( \beta \) is mentioned in [58].

  In [60], McKenney et al. used a fast multipole and boundary integral method for Laplace equation, in conjunction with a finite difference method for Poisson equations with discontinuous right-hand side [58]. Their method is second order accurate, even in very complicated regions, and has near-optimal work estimates. The extension of this method to three-dimensional problems is still being pursued. One significant contribution to the approach has been made by Greengard and Lee [27].

  The main disadvantage of the integral equation technique is that not all interface problems have the corresponding integral equations.

- The ghost fluid method.
  The ghost fluid method (GFM) [21] was developed to properly treat the interface conditions in [62], removing the spurious oscillations shown in [38]. The GFM method was originally designed to treat contact discontinuities in the inviscid Eular equations, but it was generalized to treat shocks, detonations, and deflagrations in [22] and compressible viscous flows in [23]. The generalized GFM method captures the appropriate Rankine-Hugoniot jump conditions at an interface without explicitly enforcing these
jump conditions. Instead, the GFM method creates an artificial fluid which implicitly induces the proper conditions at the interface. In the flavor of the level set function it gives an implicit representation of the Rankine-Hugoniot jump conditions at an interface. Since the jump conditions are handled implicitly by the construction of a ghost fluid, the overall scheme becomes easy to implement in multidimensions.

In [56], a similar boundary condition capturing method was used to develop a new numerical method for the Poisson equation (1.4) in the presence of interfaces where both the variable coefficients and the solution itself may be discontinuous. This approach is implemented using a standard finite difference discretization on a Cartesian grid making it simple to apply in as many as three space dimensions. Furthermore, the coefficient matrix of the associated linear system is the standard symmetric matrix for the variable coefficient Poisson equation in the absence of interfaces allowing for straightforward application of standard "black box" solvers.

This new approach does not suffer from the numerical smearing prevalent in the $\delta$-function formulation of the IBM method and smoothing methods. Also, the new method preserves jumps at the interface, too. However, the approach appears to be just a first order method.

- **Finite element method.**
  Adaptive solutions of problems like (1.4) have been dominated by the finite element method ([3, 37], in addition to many others). This approach has the advantage of a rigorous theoretical framework and a vast number of optimized commercial implementations. It could be applied to elliptic interface problems, too, for instance, the finite element methods using body fitting grids [14, 77]. It is well known that a second order accurate approximation to the solution of an interface problem can be generated by the Galerkin finite element method with the standard linear basis functions if the triangulation is aligned with the interface (body fitting grid). This can be proved strictly in one-dimensional space. For higher-dimensional problems, the error estimates are generally given in integral norms which are weaker than the infinity norm, see [2, 14, 29]. One advantage of the finite element formulation is that the resulting system of linear equation is symmetric positive definite for a self-adjoint elliptic operator which ensures that the stability of the algorithm. Application of such methods can be found in [64] and many others.

However, when doing so, some factors such as grid generation strategies for complicated domains and interfaces, and the performance of the resulting data structures must be considered. It is expected to be very difficult and time consuming, if not impossible, to generate a body fitting grid for an interface problem in which the interface separates the solution domain into pieces with complicated geometry, especially
in three-dimensional case. Such a difficulty becomes even severer for moving interface problems because a new grid has to be generated at each time step. Also it is not convenient to use numerical results from a fitting grid to produce high quality (super-convergent) approximations to other important quantities such as flux due to the irregularity of the grid. Very few publications can be found on using body fitting grid to solve moving interface problems with topological changes such as merging and splitting.

In [51], the finite element immersed interface method (FEIIM) using structured Cartesian grids is developed for two-dimensional elliptic interface problems involving both the discontinuities in the coefficients and singularities in the source term with arbitrary interface in the solution domain. The triangulations do not need to fit the interfaces any more. The idea is to construct basis functions which satisfy the interface jump conditions either exactly or approximately. Both non-conforming and conforming finite element spaces are considered. Corresponding interpolation functions are proved to be the second order accurate in the infinity norm. Similar accuracy is observed for the related Galerkin finite element solution of the interface problems. Due to the special triangulations, the method can be used as finite difference methods. The resulting linear systems are symmetric positive definite for self-adjoint elliptic equations. Although the convergence rate of non-conforming immersed finite element (IFE) solution may not be second order accurate in the infinity norm, it still seems to be better than that of the standard linear finite element solution. In addition, the non-conforming IFE is very simple, and can be extended to three dimensions easily.

- Other approaches.
  Another notable technique for solving the Poisson equation (1.4), with variable coefficients and Dirichlet boundary conditions on two-dimensional regions, presented by Johansen and Colella in [36], is also a second order accurate method that preserves jumps at the interface with a resolution comparable to that of the IIM method. A clever premise underlying this method is the ability to smoothly extend the solution outside the physical domain into a fictitious domain and to use these extend values in the numerical method. While this method suffers from a non-symmetric linear system and the usual difficulties that this introduces, the author did show that the method was compatible with both multigrid and adaptive mesh techniques. However, a three-dimensional version of the method was neither proposed nor implemented in [36].

It should be pointed out here that most of above numerical methods can be second order accurate in \(L_1\) or \(L_2\) norm, but not in \(L_\infty\) norm. And it seems that only the IIM method and the integral equation technique based on the potential theory can guarantee a second order accurate solution in \(L_\infty\) norm for interface problems.
1.4 Outline of the thesis

In Chapter 2, we will develop a maximum principle preserving scheme for three-dimensional elliptic interface problems with variable coefficients using direct finite difference discretization. By using the level set method, we will discuss how to determine geometric information of the interface, how to locate the projections of irregular grid points on the interface, how to reconstruct the interface by interpolation, how to perform local coordinate transformation and how to enforce the sign property of the maximum principle through a constrained quadratic optimization problem.

For elliptic interface problems with piecewise constant but discontinuous coefficients, an alternative second order fast algorithm will be developed in Chapter 3. The algorithm is based on fast Poisson solvers. In order to use fast Poisson solvers, we introduce the jump in the normal derivative \([u_n]\) as an unknown variable. Then the GMRES method is employed to solve a Schur complement system for \([u_n]\). This way we may be able to speed up. Moreover, this algorithm has no significant storage need, \(\beta^+\) and \(\beta^-\) are inputs, and it gives us \([u_n]\) in addition to the solution \(u\).

Finally, we will investigate some applications of the proposed fast algorithm in Chapter 4. First we will develop a domain embedding technique to solve interior or exterior Poisson equations with Dirichlet or Neumann boundary conditions on irregular domains. The basic idea is to embed irregular domains into cubic domains, and treat them as interface problems, and then use the fast algorithm to solve them. Then we will describe how to apply the new embedding method to solve an inverse interface problem, i.e., identifying unknown shapes of some objects in a domain using the measured data close to the boundaries of the objects.

As a part of this thesis, we will develop some general application programs for three-dimensional elliptic interface problems based on the algorithms proposed in this thesis.
Chapter 2

A Maximum Principle Preserving Scheme for 3D Interface Problems

2.1 Introduction

Let \( \Omega \) be a convex domain in three dimensions within which there is an irregular interface \( \Gamma \). Let \( \Omega^+ \) and \( \Omega^- \) be the two sides of the interface, see Figure 1.1. In this chapter, we develop a maximum principle preserving scheme for the following elliptic interface problem

\[
(\beta u_x)_x + (\beta u_y)_y + (\beta u_z)_z + \lambda u = f, \quad \text{in } \Omega,
\]

with some boundary conditions on \( \partial \Omega \) and jump conditions on the interface \( \Gamma \)

\[
[u] = w, \quad (2.2)
\]

\[
[\beta u_n] = q. \quad (2.3)
\]

In this thesis, we assume that the interface \( \Gamma \) is arbitrarily smooth. We also assume that in \( \Omega \), \( \beta \) is piecewise smooth, \( \lambda \) and \( f \) are piecewise continuous, and along the interface, \( w \) has continuous second derivatives and \( q \) has continuous first derivatives. Then the solution \( u \) has piecewise components in \( C^2 \), i.e., \( u \in C^2 \) in \( \Omega^+ \) or \( \Omega^- \), but not in \( \Omega \).

As we mentioned in §1.2, we still wish to solve elliptic interface problems using finite difference methods on uniform Cartesian grids. However, since \( \beta \), \( \lambda \) and \( f \) may be discontinuous across the interface, the solution \( u \) or some of its derivatives will usually be non-smooth or even discontinuous across the interface. Consequently, standard finite difference schemes will not work for interface problems.

For example, if the coefficient \( \beta \) is discontinuous across the interface, while \( \lambda \) and \( f \) are continuous, then \( u \) and \( \beta u_n \) will be continuous while the normal derivative \( u_n \) will be discontinuous [37]. In this case, the standard seven point stencil can not be used.
On the other hand, if $\beta$ is continuous but that the source term $f$ either is discontinuous across the interface or has a $\delta$-function singularity along the interface, then the solution $u$ is continuous but there is a jump in the flux. In this case, although the standard seven point stencil can be used, we must derive an appropriate term on the right-hand side to model the discontinuous or singular source.

It is even possible that a dipole source may occur, in which $f$ contains the derivative of a $\delta$-function. In this case, the solution $u$ itself is discontinuous across $\Gamma$. Again, we can then use the standard seven point stencil to discretize the left-hand side of (2.1), but just modify the right-hand side to get a second order finite difference scheme.

In §1.3.4, we reviewed a lot of numerical methods for interface problems. Some are first order methods such as Peskin’s immersed boundary method [66, 67], the smoothing method [73], the harmonic averaging method [5, 71, 75] and the ghost fluid method [21, 56]. Some are second order methods such as the fast multipole method [60], the integral equation technique [58, 59], and the immersed interface method [43, 46]. Although theoretically, some of them can be applied to three-dimensional cases, few publications can be found on numerical analysis and applications for three-dimensional interface problems.

In this chapter, we want to develop a second order numerical approach which can handle discontinuous coefficients $\beta, \lambda$, and discontinuous, singular or dipole sources simultaneously. The approach is based on the immersed interface method. In [50], the authors developed a new approach for two-dimensional elliptic interface problems with variable coefficient. The goal of the new approach is to obtain a finite difference scheme that satisfies the maximum principle, see Section 6.5 of Morton and Mayers [61] for the definition, and this is achieved by imposing the sign property on the coefficients of difference scheme for each irregular grid point. The sign property is enforced through a constrained quadratic optimization problem. In this way, the final linear system of equations from the finite difference method is diagonally dominant and its symmetric part is negative definite so convergence is guaranteed. Below we will study the generalization of the approach for three-dimensional interface problems.

### 2.2 The algorithm description

For simplicity, we assume that the domain $\Omega$ is a simple domain such as a solid cube, say $[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$. We wish to solve the problem using a finite difference method and a uniform Cartesian grid with

$$x_i = a_1 + ih, \quad y_j = a_2 + jh, \quad z_k = a_3 + kh, \quad 0 \leq i \leq l, \quad 0 \leq j \leq m, \quad 0 \leq k \leq n.$$  

Here, for simplicity, we also assume that $h = (b_1 - a_1)/l = (b_2 - a_2)/m = (b_3 - a_3)/n$. 


Chapter 2. A Maximum Principle Preserving Scheme for 3D Interface Problems

Our goal is to develop a modified finite difference scheme of the form

\[ \sum_{m} \gamma_{m} u_{i+m,j+m,k+m} + \lambda_{ijk} u_{ijk} = f_{ijk} + C_{ijk}, \]  

(2.4)

at any grid point \((x_i, y_j, z_k)\), where the summation is taken over \(n_s\) grid points centered at \((x_i, y_j, z_k)\). By finding the proper coefficients \(\gamma_{m}\)'s, we wish the resulted finite difference scheme be still second order accurate.

Also we wish for those regular grid points, the standard seven point stencil be used directly. Here we say a grid point \((x_i, y_j, z_k)\) is regular if the interface \(\Gamma\) does not come between any points in the standard seven point stencil centered at \((x_i, y_j, z_k)\). At these points, we may obtain an \(O(h^2)\) truncation error using the standard seven point formula \((n_s = 7)\)

\[
\begin{align*}
\frac{1}{h^2} & \left( \beta_{i+\frac{1}{2},j}(u_{i+1,j,k} - u_{ijk}) - \beta_{i-\frac{1}{2},j}(u_{ij} - u_{i-1,j,k}) \right) \\
&+ \beta_{i,j+\frac{1}{2},k}(u_{i,j+1,k} - u_{ijk}) - \beta_{i,j-\frac{1}{2},k}(u_{ij} - u_{i,j-1,k}) \\
&+ \beta_{ij,k+\frac{1}{2}}(u_{ij,k+1} - u_{ijk}) - \beta_{ij,k-\frac{1}{2}}(u_{ij} - u_{ij,k-1}) \right) + \lambda_{ijk} u_{ijk} = f_{ijk},
\end{align*}
\]

(2.5)

with \(C_{ijk} = 0\), and

\[ \beta_{ij} = \beta(x_i, y_j, z_k), \quad \beta_{i+\frac{1}{2},j} = \beta(x_i + h/2, y_j, z_k), \]

and so on.

On the other hand, we wish to determine formulas of the form (2.4) for irregular grid points too. We say a grid point \((x_i, y_j, z_k)\) is irregular if the interface \(\Gamma\) does come between points in the standard seven point stencil centered at \((x_i, y_j, z_k)\). To maintain second order accuracy, special attention is needed to derive (2.4). Since these points are adjacent to the interface \(\Gamma\), and form a lower-dimensional set, it turns out to be sufficient to require an \(O(h)\) truncation error at these points.

We follow the same approach as in one dimension and two dimensions discussed in [43, 46]. Given an irregular grid point \((x_i, y_j, z_k)\), first we select a point \((x_i^*, y_j^*, z_k^*)\) on the interface \(\Gamma\) near \((x_i, y_j, z_k)\). We will discuss how to choose the point later. Then we expand all of \(u(x_{i+m}, y_{j+m}, z_{k+m})\) in Taylor series expansion about \((x_i^*, y_j^*, z_k^*)\) from the + or - side of the interface in terms of \(u^\pm, u_x^\pm, u_y^\pm, u_z^\pm, u_{xx}^\pm, u_{yy}^\pm, u_{zz}^\pm, u_{xy}^\pm, u_{xz}^\pm, u_{yz}^\pm\), where the superscripts + and - denote the limiting values of a function from one side or the other.
For example, we could expand \( u(x_i, y_j, z_k) \) as
\[
  u(x_i, y_j, z_k) = u^- + u_x^-(x_i - x_i^*) + u_y^-(y_j - y_j^*) + u_z^-(z_k - z_k^*)
  + \frac{1}{2} u_{xx}^-(x_i - x_i^*)^2 + \frac{1}{2} u_{yy}^-(y_j - y_j^*)^2 + \frac{1}{2} u_{zz}^-(z_k - z_k^*)^2
  + u_{xy}^-(x_i - x_i^*)(y_j - y_j^*) + u_{xz}^-(x_i - x_i^*)(z_k - z_k^*)
  + u_{yz}^-(y_j - y_j^*)(z_k - z_k^*) + O(h^3),
\]
if \((x_i, y_j, z_k) \in \Omega^-\), the \(-\) side of the interface.

If we do this expansion at each grid point used in the finite difference scheme (2.4), then the local truncation error \( T_{ijk} \) can be expressed as a linear combination of the values \( u^\pm, u_x^\pm, u_y^\pm, u_z^\pm, u_{xx}^\pm, u_{yy}^\pm, u_{zz}^\pm, u_{xz}^\pm, u_{yz}^\pm \). We wish to eliminate all values on one side of the interface, say the values \( u^+, u_{xx}^+, u_{yy}^+, u_{zz}^+, u_{xz}^+, u_{yz}^+, u_{xxy}^+, u_{yyx}^+, u_{zzx}^+, u_{yzx}^+, u_{xyy}^+, u_{xyz}^+, u_{yzz}^+, u_{yzy}^+, u_{xzx}^+, u_{zzy}^+, u_{zyz}^+, u_{xyz}^+, u_{yzx}^+, u_{zyz}^+ \), in terms of the values on the other side, \( u^-, u_{xx}^-, u_{yy}^-, u_{zz}^-, u_{xz}^-, u_{yz}^-, u_{xxy}^-, u_{yyx}^-, u_{zzx}^-, u_{yzx}^-, u_{xyy}^-, u_{xyz}^-, u_{yzx}^-, u_{zyz}^- \). We must do this using the jump conditions (2.2) and (2.3)
\[
  u^+ = u^- + w,
\]
and
\[
  \beta^+ u_n^+ = \beta^- u_n^- + q.
\]
However, to determine the coefficients \( \gamma_m \)'s in (2.4), more interface relations are needed. Actually, differentiating the two given jump conditions and manipulating the results allows us to perform the desired elimination, as detailed in §2.2.2 and §2.2.3. But since the jump condition (2.3) is given in the normal derivative of the solution, it turns out to be very convenient to first perform a local coordinate transformation into the directions \( \xi \), normal to \( \Gamma \), \( \eta \) and \( \tau \), tangential to \( \Gamma \).

Once the local truncation error \( T_{ijk} \) is expressed as a linear combination of the values from just one side, say, \( u^-, u_x^-, u_y^-, u_z^-, u_{xx}^-, u_{yy}^-, u_{zz}^-, u_{xz}^-, u_{yz}^-, u_{xxy}^-, u_{yyx}^-, u_{zzx}^-, u_{yzx}^-, u_{xyy}^-, u_{xyz}^-, u_{yzx}^-, u_{zyz}^- \), we must require that the coefficient of each of these terms vanishes in order to achieve an \( O(h) \) truncation error. This gives a system of ten linear equations to determine the coefficients \( \gamma_m \)'s.

Obviously, at least ten grid points are needed to obtain a solvable system, i.e., \( n_s \geq 10 \), and we should use the standard seven point stencil together with some additional neighboring points of \((x_i, y_j, z_k)\).

In the case of exact ten points involved, we will have a determined system. We could expect that the system is solvable. But unfortunately this is not always true. Actually, our numerical experiments showed that even the interface was a sphere, there were always some irregular grid points, even though the number of those points was very small, for which the resulted systems for \( \gamma_m \)'s were numerically unsolvable. Therefore, we tend to get more than ten points involved, which instead will give us an underdetermined system for \( \gamma_m \)'s, and thus there will be an infinite number of solutions. We will then choose the one with
the sign property of the maximum principle and some minimum 2-norm using quadratic optimization techniques.

To summarize, in order to determine the finite difference scheme at an irregular grid point \((x_i, y_j, z_k)\), we need to do the following:

- Select a point \((x^*_i, y^*_j, z^*_k)\) \(\in \Gamma\) near \((x_i, y_j, z_k)\).
- Apply a local coordinate transformation in the directions normal and tangential to \(\Gamma\) at \((x^*_i, y^*_j, z^*_k)\).
- Derive the interface relations relating \(+\) and \(-\) values at \((x^*_i, y^*_j, z^*_k)\) in the local coordinates.
- Choose some additional points to form a modified stencil.
- Set up and solve a system of ten linear equations for the coefficients \(\gamma_m\)’s.
- Compute the correction term \(C_{ijk}\).

Below we give the description of each step in detail.

### 2.2.1 The local coordinate transformation

For each irregular grid point \((x_i, y_j, z_k)\), we choose a point, say \((x^*_i, y^*_j, z^*_k)\), on the interface \(\Gamma\) so that we can expand each \(u_{i+m,j+k,m,k}^n\) involved in the desired finite difference equation (2.4) about it. Basically we can take any point on the interface in the neighborhood of \((x_i, y_j, z_k)\). We can even choose a nearby point that lies on a coordinate line between \((x_i, y_j, z_k)\) and one of its six neighbors. However, for accuracy, we usually take this point as the projection of \((x_i, y_j, z_k)\) on the interface if the interface is smooth at this point. We will discuss how to find this projection in a later section.

As mentioned in the above, we need to derive some additional interface relations along the interface to eliminate all values on one side of the interface in terms of the corresponding values on the other side. In order to do so, since one of two jump conditions \([\beta u_n]\) is given in the normal derivative of the solution, it is convenient to first perform a local coordinate transformation as described next.

Given a point \((x^*, y^*, z^*)\) on the interface \(\Gamma\), let \(\xi\) be the direction normal to \(\Gamma\), \(\eta\) and \(\tau\) be two perpendicular directions tangential to \(\Gamma\), then the local coordinate transformation is given by
Chapter 2. A Maximum Principle Preserving Scheme for 3D Interface Problems

\[
\begin{align*}
\xi &= (x - x^*) \alpha_x \xi + (y - y^*) \alpha_y \xi + (z - z^*) \alpha_z \xi \\
\eta &= (x - x^*) \alpha_x \eta + (y - y^*) \alpha_y \eta + (z - z^*) \alpha_z \eta \\
\tau &= (x - x^*) \alpha_x \tau + (y - y^*) \alpha_y \tau + (z - z^*) \alpha_z \tau,
\end{align*}
\]  
(2.9)

where \(\alpha_{x\xi}\) represents the direction cosine between \(x\)-axis and \(\xi\), etc., see Figure 2.1.

\[\begin{figure}
\text{Figure 2.1: A sketch of a three-dimensional local coordinate transformation.}
\end{figure}\]

The above three-dimensional coordinate transformation can also be written in the following vector form. Define the local transformation matrix

\[
A = \begin{pmatrix}
\alpha_{x\xi} & \alpha_{y\xi} & \alpha_{z\xi} \\
\alpha_{x\eta} & \alpha_{y\eta} & \alpha_{z\eta} \\
\alpha_{x\tau} & \alpha_{y\tau} & \alpha_{z\tau}
\end{pmatrix},
\]  
(2.10)

then we have

\[
\begin{pmatrix}
\xi \\
\eta \\
\tau
\end{pmatrix} = A \begin{pmatrix}
x - x^* \\
y - y^* \\
z - z^*
\end{pmatrix}. 
\]  
(2.11)

Also, for any differentiable function \(p(x, y, z)\), we have

\[
\begin{pmatrix}
p\xi \\
p\eta \\
p\tau
\end{pmatrix} = A \begin{pmatrix}
p_x \\
p_y \\
p_z
\end{pmatrix}, 
\]  
(2.12)

and
Chapter 2. A Maximum Principle Preserving Scheme for 3D Interface Problems

\[
\begin{pmatrix}
p_{\xi\xi} & p_{\xi\eta} & p_{\xi\tau} \\
p_{\eta\xi} & p_{\eta\eta} & p_{\eta\tau} \\
p_{\tau\xi} & p_{\tau\eta} & p_{\tau\tau}
\end{pmatrix} = A \begin{pmatrix}
p_{xx} & p_{xy} & p_{xz} \\
p_{yx} & p_{yy} & p_{yz} \\
p_{zx} & p_{zy} & p_{zz}
\end{pmatrix} A^T,
\]

(2.13)

where \( A^T \) is the transpose of \( A \).

Note that under the local coordinate transformation (2.9), the partial differential equation (PDE) (2.1) remains unchanged. In fact, this is true even if \( \beta, \lambda, \) and \( f \) depend on \( x, y \) and \( z \). In fact, taking an arbitrary differentiable function \( p(x, y, z) \), under the transformation (2.9), we have

\[
\begin{pmatrix}
p_x \\
p_y \\
p_z
\end{pmatrix} = A^{-1} \begin{pmatrix}
p_{\tilde{\xi}} \\
p_{\tilde{\eta}} \\
p_{\tilde{\tau}}
\end{pmatrix},
\]

where \( p(\xi, \eta, \tau) = p(x, y, z) \), and so forth. Therefore

\[
(\beta u_x)_x + (\beta u_y)_y + (\beta u_z)_z + \lambda u = \beta (u_{xx} + u_{yy} + u_{zz}) + \beta_x u_x + \beta_y u_y + \beta_z u_z + \lambda u
\]

\[
= \tilde{\beta} (\bar{u}_{\xi\xi} + \bar{u}_{\eta\eta} + \bar{u}_{\tau\tau}) + (\tilde{\beta}_x, \tilde{\beta}_y, \tilde{\beta}_z) A^{-T} A^T \begin{pmatrix}
\bar{u}_{\xi} \\
\bar{u}_{\eta} \\
\bar{u}_{\tau}
\end{pmatrix} + \lambda \bar{u}
\]

\[
= (\tilde{\beta} \bar{u}_{\xi\xi})_{\xi} + (\tilde{\beta} \bar{u}_{\eta\eta})_{\eta} + (\tilde{\beta} \bar{u}_{\tau\tau})_{\tau} + \lambda \bar{u}.
\]

We should have a new notation for \( u(x, y, z) \), \( \lambda(x, y, z) \), \( f(x, y, z) \) in the local coordinates, say, \( \tilde{u}(\xi, \eta, \tau) = u(x, y, z) \), \( \tilde{\lambda}(\xi, \eta, \tau) = \lambda(x, y, z) \), and \( \tilde{f}(\xi, \eta, \tau) = f(x, y, z) \), etc. But for simplicity we drop the bars and use the same notation in the local coordinates as in the old ones.

We use the superscripts + or − to denote the limiting values of a function from one side or the other of the interface. Then under the local coordinates, the limit differential equation from the + side can be written as

\[
\beta^+(u_{\xi\xi}^+ + u_{\eta\eta}^+ + u_{\tau\tau}^+) + \beta_x^+ u_{\xi}^+ + \beta_y^+ u_{\eta}^+ + \beta_z^+ u_{\tau}^+ + \lambda^+ u^+ - f^+ = 0,
\]

(2.14)

and from the − side it can be written as

\[
\beta^-(u_{\xi\xi}^- + u_{\eta\eta}^- + u_{\tau\tau}^-) + \beta_x^- u_{\xi}^- + \beta_y^- u_{\eta}^- + \beta_z^- u_{\tau}^- + \lambda^- u^- - f^- = 0.
\]

(2.15)
Chapter 2. A Maximum Principle Preserving Scheme for 3D Interface Problems

Combining them together, we have

\[ \beta(u_{\xi\xi} + u_{\eta\eta} + u_{\tau\tau}) + \beta_{\xi} u_{\xi} + \beta_{\eta} u_{\eta} + \beta_{\tau} u_{\tau} + \lambda u = [f]. \]  \hspace{1cm} (2.16)

If a grid point \((x_i, y_j, z_k)\) happens to fall on the interface, then \(u(x_i, y_j, z_k)\) is defined as the limiting value of \(u\) from one side of the interface or the other. The same argument applies to all other functions such as \(\beta\), \(\lambda\), \(f\) and the derivatives of \(u\).

### 2.2.2 Interface relations

We consider a fixed point \((x^*, y^*, z^*)\) on the interface, and define a local \(\xi-\eta-\tau\) coordinate system by (2.9) based on the directions normal and tangential to \(\Gamma\) at this point. In the neighborhood of \((x^*, y^*, z^*)\), the interface can then be expressed as

\[ \xi = \chi(\eta, \tau), \quad \text{with} \quad \chi(0,0) = 0, \quad \chi_\eta(0,0) = 0, \quad \chi_\tau(0,0) = 0. \]  \hspace{1cm} (2.17)

Recall that in the local coordinates the equation (2.1) is unchanged, so we will use the same notation for \(u\), \(\beta\), \(\lambda\), \(f\), \(w\) and \(q\).

As we pointed out earlier, we want to express \(u^+\), \(u^+_{\xi}\), \(u^+_{\eta}\), \(u^+_{\tau}\), \(u^+_{\xi\eta}\), \(u^+_{\xi\tau}\), \(u^+_{\eta\eta}\), \(u^+_{\eta\tau}\), \(u^+_{\tau\tau}\), \(u^+_{\xi\eta}\), \(u^+_{\xi\tau}\), and \(u^+_{\eta\tau}\) in terms of the values on the other side, \(u^-\), \(u^-_{\xi}\), \(u^-_{\eta}\), \(u^-_{\tau}\), \(u^-_{\xi\eta}\), \(u^-_{\xi\tau}\), \(u^-_{\eta\eta}\), \(u^-_{\eta\tau}\), \(u^-_{\tau\tau}\), \(u^-_{\xi\eta}\), \(u^-_{\xi\tau}\), and \(u^-_{\eta\tau}\). To do so, we have to use jump conditions across \(\Gamma\). However, the given two jump conditions (2.2) and (2.3) give us only two obvious interface relations

\[
\begin{align*}
    u^+ - u^- &= w, \\
    u^+_{\xi} &= \frac{\beta^+}{\beta^-} u^-_{\xi} + \frac{q}{\beta^+}. 
\end{align*}
\]  \hspace{1cm} (2.18, 2.19)

Recall that in this thesis we assume that \(w\) has continuous second derivatives and \(q\) has continuous first derivatives along the interface, and we are interested in the strong solution which means that \(u\) has piecewise components in \(C^2\) \((u \in C^2 \text{ in } \Omega^+ \text{ or } \Omega^-\), but not in the entire domain \(\Omega\)). Accordingly, \(u^+\) and \(u^-\) have continuous second derivatives along the interface, and so forth. Therefore we can differentiate \(w\), \(q\), \(u^+\), \(u^-\), etc., along the interface.

Now let us first differentiate the jump condition (2.2) with respect to \(\eta\) and \(\tau\) respectively. By using the chain rule, we have

\[
\begin{align*}
    [u_{\xi}] \chi_\eta + [u_\eta] &= w_\eta, \\
    [u_{\xi}] \chi_\tau + [u_\tau] &= w_\tau. 
\end{align*}
\]  \hspace{1cm} (2.20, 2.21)
Differentiating (2.20) with respect to $\tau$ yields
\[\chi_\eta \frac{\partial}{\partial \tau} [u_\xi] + \chi_\eta \tau [u_\xi] + [u_{\eta \xi}] \chi_\tau + [u_{\eta \tau}] = w_{\eta \tau}. \tag{2.22}\]

Differentiating (2.20) with respect to $\eta$ and differentiating (2.21) with respect to $\tau$ respectively we obtain
\[\chi_\eta \frac{\partial}{\partial \eta} [u_\xi] + \chi_\eta \eta [u_\xi] + \chi_\eta \tau [u_{\eta \xi}] + [u_{\eta \tau}] = w_{\eta \eta}, \tag{2.23}\]
\[\chi_\tau \frac{\partial}{\partial \tau} [u_\xi] + \chi_\tau \tau [u_\xi] + \chi_\tau \eta [u_{\tau \xi}] + [u_{\tau \tau}] = w_{\tau \tau}. \tag{2.24}\]

Before differentiating the jump of the normal derivative (2.3) we first express the unit normal vector of the interface $\Gamma$ as
\[\mathbf{n} = \left(1, -\chi_\eta, -\chi_\tau\right) \sqrt{1 + \chi_\eta^2 + \chi_\tau^2}. \tag{2.25}\]

So the jump condition (2.3) can be written as
\[\left[ \beta \left( u_\xi - u_\eta \chi_\eta - u_\tau \chi_\tau \right) \right] = q(\eta, \tau) \sqrt{1 + \chi_\eta^2 + \chi_\tau^2}. \tag{2.26}\]

Differentiating this with respect to $\eta$ gives
\[\left[ \left( \beta_\xi \chi_\eta + \beta_\eta \right) \left( u_\xi - u_\eta \chi_\eta - u_\tau \chi_\tau \right) \right] + \left[ \beta \left( u_\xi \chi_\eta + u_\eta \chi_\eta - \chi_\eta \frac{\partial}{\partial \eta} u_\eta - \chi_\tau \frac{\partial}{\partial \eta} u_\tau - u_\eta \chi_\eta \eta - u_\tau \chi_\eta \tau \right) \right] = q_\eta \sqrt{1 + \chi_\eta^2 + \chi_\tau^2} \frac{\chi_\eta \chi_\eta \eta}{\sqrt{1 + \chi_\eta^2 + \chi_\tau^2}}. \tag{2.27}\]

Similarly, differentiating (2.26) with respect to $\tau$ gives
\[\left[ \left( \beta_\xi \chi_\tau + \beta_\tau \right) \left( u_\xi - u_\eta \chi_\eta - u_\tau \chi_\tau \right) \right] + \left[ \beta \left( u_\xi \chi_\tau + u_\eta \chi_\tau - \chi_\eta \frac{\partial}{\partial \tau} u_\eta - \chi_\tau \frac{\partial}{\partial \tau} u_\tau - u_\eta \chi_\eta \tau - u_\tau \chi_\tau \tau \right) \right] = q_\tau \sqrt{1 + \chi_\eta^2 + \chi_\tau^2} \frac{\chi_\tau \chi_\tau \tau}{\sqrt{1 + \chi_\eta^2 + \chi_\tau^2}}. \tag{2.28}\]
At the origin, $\chi_\eta(0,0) = \chi_\tau(0,0) = 0$, and from (2.20)–(2.28) we can conclude that

\[
\begin{align*}
    u^+ &= u^- + w, \\
    u^+_\xi &= \frac{\beta^-}{\beta^+} u^-_\xi + \frac{q^-}{\beta^+}, \\
    u^+_\eta &= u^-_\eta + w_\eta, \\
    u^+_\tau &= u^-_\tau + w_\tau, \\
    u^+_{\eta\tau} &= u^-_{\eta\tau} + (u^-_\xi - u^+_\xi)\chi_{\eta\tau} + w_{\eta\tau}, \\
    u^+_{\eta\eta} &= u^-_{\eta\eta} + (u^-_\xi - u^+_\xi)\chi_{\eta\eta} + w_{\eta\eta}, \\
    u^+_{\tau\tau} &= u^-_{\tau\tau} + (u^-_\xi - u^+_\xi)\chi_{\tau\tau} + w_{\tau\tau}, \\
    u^+_{\xi\eta} &= \frac{\beta^-}{\beta^+} u^-_{\xi\eta} + \left( u^-_\eta - \frac{\beta^-}{\beta^+} u^-_\eta \right) \chi_{\eta\eta} + \left( u^-_\tau - \frac{\beta^-}{\beta^+} u^-_\tau \right) \chi_{\eta\tau} \\
    &\quad + \frac{\beta^-}{\beta^+} u^-_\xi - \frac{\beta^-}{\beta^+} u^+_\xi + \frac{q_\eta}{\beta^+}, \\
    u^+_{\xi\tau} &= \frac{\beta^-}{\beta^+} u^-_{\xi\tau} + \left( u^-_\eta - \frac{\beta^-}{\beta^+} u^-_\eta \right) \chi_{\eta\tau} + \left( u^-_\tau - \frac{\beta^-}{\beta^+} u^-_\tau \right) \chi_{\tau\tau} \\
    &\quad + \frac{\beta^-}{\beta^+} u^-_\xi - \frac{\beta^-}{\beta^+} u^+_\xi + \frac{q_\tau}{\beta^+}.
\end{align*}
\]

To get the relation for $u^+_{\xi\xi}$ we need to use the differential equation (2.1) itself from which we can write

\[
\left[ \beta (u_{\xi\xi} - u_{\eta\eta} + u_{\tau\tau}) + \beta_{\xi} u_{\xi} + \beta_{\eta} u_{\eta} + \beta_{\tau} u_{\tau} + \lambda u \right] = [f].
\]

Notice that

\[
\lambda^- u^- - \lambda^+ u^+ = \lambda^- u^- - \lambda^+ u^- + \lambda^+ u^- - \lambda^+ u^+ = -[\lambda]u^- - \lambda^+ [u].
\]

Rearranging equation (2.30) and using (2.31) above we get

\[
\begin{align*}
\beta^+ \left( u^+_{\xi\xi} + u^+_{\eta\eta} + u^+_{\tau\tau} \right) + \beta^- \left( u^-_{\xi\xi} + u^-_{\eta\eta} + u^-_{\tau\tau} \right) + \beta^+ u^+_{\xi\eta} + \beta^+ u^+_{\xi\tau} + \beta^+ u^+_\eta + \beta^+ u^+_\tau &= \\
\beta^- \left( u^-_{\xi\xi} + u^-_{\eta\eta} + u^-_{\tau\tau} \right) + \beta^- u^-_{\xi\xi} + \beta^- u^-_{\eta\eta} + \beta^- u^-_{\tau\tau} + \beta^- u^-_{\xi\eta} + \beta^- u^-_{\xi\tau} + \beta^- u^-_{\eta\eta} + \beta^- u^-_{\eta\tau} + \beta^- u^-_{\tau\tau} + \beta^- u^-_{\xi\tau} + \beta^- u^-_{\eta\tau} + \beta^- u^-_{\tau\tau} + [f] + \lambda^- u^- - \lambda^+ u^+.
\end{align*}
\]

Plugging the sixth and seventh equations of (2.29) in (2.32) and collecting terms finally we
have
\[ u^+_{\xi \xi} = \frac{\beta^-}{\beta^+} u^-_{\xi \xi} + \left( \frac{\beta^-}{\beta^+} - 1 \right) u^-_{\eta \eta} + \left( \frac{\beta^-}{\beta^+} - 1 \right) u^-_{\tau \tau} + \\
\frac{1}{\beta^+} \left( \beta^- u^-_{\eta \eta} - \beta^+ u^+_{\eta} \right) + \frac{1}{\beta^+} \left( \beta^- u^-_{\tau \tau} - \beta^+ u^+_{\tau} \right) \\
- \frac{1}{\beta^+} \left( [\lambda] u^- + \lambda^+ [u] \right) + \frac{|f|}{\beta^+} - w_{\eta \eta} - w_{\tau \tau}. \]  

(2.33)

2.2.3 The finite difference scheme

At a regular grid point, we still use the standard central difference scheme which has a seven point stencil. So we will concentrate below on developing finite difference schemes for irregular grid points. Taking an irregular grid point, say \((x_i, y_j, z_k)\), we try to develop the modified finite difference scheme (2.4) at this point. Let us rewrite the finite difference scheme (2.4) here

\[ \sum_{m}^{n_s} \gamma_m u_{i+m,j+j_m,k+k_m} + \lambda u_{ijk} = f_{ijk} + C_{ijkl}, \]  

where the summation is taken over \(n_s\) neighboring grid points centered at \((x_i, y_j, z_k)\).

We use the undetermined coefficient method to determine the coefficients \(\gamma_m\)’s. For the elliptic equation the coefficients \(\gamma_m\)’s should be of order \(O(1/h^2)\). So if we expand \(u_{i+m,j+j_m,k+k_m}\) in the finite difference scheme about \((x^*, y^*, z^*)\), the origin of the local coordinate system, from each side of the surface \(\Gamma\), we need to match up to second derivatives to guarantee that the local truncation error is \(O(h)\). Using the ten interface relations (2.29) and (2.33) to eliminate quantities from the + side of the interface, then the Taylor series expansion of (2.34) will contain \(u^-\), \(u^-_{\xi\xi}\), \(u^-_{\eta\eta}\), \(u^-_{\tau\tau}\), \(u^-_{\eta\eta}\), \(u^-_{\tau\tau}\), \(u^-_{\xi\eta}\), \(u^-_{\xi\tau}\), and \(u^-_{\eta\tau}\). To match them we need altogether at least ten grid points to get ten equations for the \(\gamma_m\)’s. Thus we need to find at least three additional points besides the standard seven point stencil.

Once we have determined the stencil, we need to set up the equations for the coefficients \(\gamma_m\)’s. We use the Taylor series expansion of (2.34) about \((x^*, y^*, z^*)\). For instance, if the
grid point \((x_i, y_j, z_k)\) is on the \(-\) side, we will get

\[
a_1 u^- + a_2 u^+ + a_3 u^-_\xi + a_4 u^+\xi + a_5 u^-_\eta + a_6 u^+\eta + a_7 u^-_\tau + a_8 u^+\tau \\
+ a_9 u^- \xi \xi + a_{10} u^+ \xi \xi + a_{11} u^- \eta \eta + a_{12} u^+ \eta \eta + a_{13} u^- \tau \tau \\
+ a_{14} u^+ \tau \tau + a_{15} u^- \xi \eta + a_{16} u^+ \xi \eta + a_{17} u^- \eta \tau + a_{18} u^+ \eta \tau \\
+ a_{19} u^- \eta \tau + a_{20} u^+ \eta \tau + \lambda^- u^- = f^- + C_{ijk} + O(h),
\]

(2.35)

where the coefficients \(a_j\)’s depend only on the position of the stencil relative to the interface. They are independent of the functions \(u, \lambda\) and \(f\). If we define the index sets \(K^+\) and \(K^-\) by

\[
K^\pm = \{m : (\xi_m, \eta_m, \tau_m) \text{ is on the } \pm \text{ side of } \Gamma\},
\]

where \(\xi_m, \eta_m, \tau_m\) really denote \(\xi_{i+m}, \eta_{j+m}, \tau_{k+m}\), then the \(a_i\)’s are given by

\[
a_1 = \sum_{m \in K^-} \gamma_m \\
a_3 = \sum_{m \in K^-} \xi_m \gamma_m \\
a_5 = \sum_{m \in K^-} \eta_m \gamma_m \\
a_7 = \sum_{m \in K^-} \tau_m \gamma_m \\
a_9 = \frac{1}{2} \sum_{m \in K^-} \xi_m^2 \gamma_m \\
a_{11} = \frac{1}{2} \sum_{m \in K^-} \eta_m^2 \gamma_m \\
a_{13} = \frac{1}{2} \sum_{m \in K^-} \tau_m^2 \gamma_m \\
a_{15} = \sum_{m \in K^-} \xi_m \eta_m \gamma_m \\
a_{17} = \sum_{m \in K^-} \xi_m \tau_m \gamma_m \\
a_{19} = \sum_{m \in K^-} \eta_m \tau_m \gamma_m \\
a_2 = \sum_{m \in K^+} \gamma_m \\
a_4 = \sum_{m \in K^+} \xi_m \gamma_m \\
a_6 = \sum_{m \in K^+} \eta_m \gamma_m \\
a_8 = \sum_{m \in K^+} \tau_m \gamma_m \\
a_{10} = \frac{1}{2} \sum_{m \in K^+} \xi_m^2 \gamma_m \\
a_{12} = \frac{1}{2} \sum_{m \in K^+} \eta_m^2 \gamma_m \\
a_{14} = \frac{1}{2} \sum_{m \in K^+} \tau_m^2 \gamma_m \\
a_{16} = \sum_{m \in K^+} \xi_m \eta_m \gamma_m \\
a_{18} = \sum_{m \in K^+} \xi_m \tau_m \gamma_m \\
a_{20} = \sum_{m \in K^+} \eta_m \tau_m \gamma_m.
\]
Using the interface relations (2.29) and (2.33), and rearranging (2.35) we have the truncation error

\[
T_{ijk} = \left( a_1 - a_{10} \frac{[\lambda]}{\beta^+} \right) u^- + a_2 u^+ + \left\{ a_3 - a_{10} \left( \chi_{\eta \eta} + \chi_{\tau \tau} - \frac{\beta^-}{\beta^+} \right) \right. \\
+ a_{12} \chi_{\eta \eta} + a_{14} \chi_{\tau \tau} + a_{16} \frac{\beta^-}{\beta^+} + a_{18} \frac{\beta^+}{\beta^+} + a_{20} \chi_{\eta \tau} \right\} u^-_\xi \\
+ \left\{ a_4 + a_{10} \left( \chi_{\eta \eta} + \chi_{\tau \tau} - \frac{\beta^+}{\beta^+} \right) \right. \\
- a_{12} \chi_{\eta \eta} - a_{14} \chi_{\tau \tau} - a_{16} \frac{\beta^-}{\beta^+} - a_{18} \frac{\beta^+}{\beta^+} - a_{20} \chi_{\eta \tau} \right\} u^+_\xi \\
+ \left( a_5 + a_{10} \frac{\beta^-}{\beta^+} - a_{16} \frac{\beta^-}{\beta^+} \chi_{\eta \eta} - a_{18} \frac{\beta^-}{\beta^+} \chi_{\eta \tau} \right) u^-_\eta \\
+ \left( a_6 - a_{10} \frac{\beta^+}{\beta^+} + a_{16} \chi_{\eta \eta} + a_{18} \chi_{\eta \tau} \right) u^+_\eta \\
+ \left( a_7 + a_{10} \frac{\beta^-}{\beta^+} - a_{16} \frac{\beta^-}{\beta^+} \chi_{\eta \eta} - a_{18} \frac{\beta^-}{\beta^+} \chi_{\eta \tau} \right) u^-_\tau \\
+ \left( a_8 - a_{10} \frac{\beta^+}{\beta^+} + a_{16} \chi_{\eta \tau} + a_{18} \chi_{\eta \tau} \right) u^+_\tau \\
+ \left( a_9 + a_{10} \frac{\beta^-}{\beta^+} \right) u^-_{\xi \xi} + \left( a_{11} + a_{12} + a_{10} \left( \frac{\beta^-}{\beta^+} - 1 \right) \right) u^-_{\eta \eta} \\
+ \left( a_{13} + a_{14} + a_{10} \left( \frac{\beta^-}{\beta^+} - 1 \right) \right) u^-_{\tau \tau} + \left( a_{15} + a_{16} \frac{\beta^-}{\beta^+} \right) u^-_{\xi \eta} \\
+ \left( a_{17} + a_{18} \frac{\beta^-}{\beta^+} \right) u^-_{\xi \tau} + \left( a_{19} + a_{20} \right) u^-_{\eta \tau} + a_{12} \omega_{\eta \eta} + a_{14} \omega_{\tau \tau} \\
+ a_{10} \left( \frac{f}{\beta^+} - \frac{\lambda^+}{\beta^+} - \omega_{\eta \eta} - \omega_{\tau \tau} \right) + a_{16} \frac{g_{\eta}}{\beta^+} \\
+ a_{18} \frac{g_{\tau}}{\beta^+} + a_{20} \omega_{\eta \tau} + \lambda^- u^- - f^- - C_{ijk}.
\]

Now it is clear that to make \( T_{ijk} \) to be \( O(h) \) we should set

\[
a_1 - a_{10} \frac{[\lambda]}{\beta^+} + a_2 = 0, \tag{2.38}
\]

\[
a_3 - a_{10} \left( \chi_{\eta \eta} + \chi_{\tau \tau} - \frac{\beta^-}{\beta^+} \right) + a_{12} \chi_{\eta \eta} + a_{14} \chi_{\tau \tau} + a_{16} \frac{\beta^-}{\beta^+} \\
+ a_{18} \frac{\beta^-}{\beta^+} + a_{20} \chi_{\eta \tau} + a_{10} \left( \chi_{\eta \eta} + \chi_{\tau \tau} - \frac{\beta^+}{\beta^+} \right)
\]
\[-a_{12} \chi_{\eta} - a_{14} \chi_{\tau \tau} - a_{16} \frac{\beta^+}{\beta^+} \chi_{\eta} - a_{18} \frac{\beta^+}{\beta^+} \chi_{\eta \tau} - a_{20} \chi_{\eta \eta}\] 
\[\begin{align*}
&= \beta^-_x, \\
&\quad a_5 + a_{10} \frac{\beta^-_\eta}{\beta^+} - a_{16} \frac{\beta^-_\eta}{\beta^+} \chi_{\eta} - a_{18} \frac{\beta^-_\eta}{\beta^+} \chi_{\eta \tau} \\
&\quad + a_6 - a_{10} \frac{\beta^+_\eta}{\beta^+} + a_{16} \chi_{\eta \eta} + a_{18} \chi_{\eta \tau} = \beta^-_\eta, \\
&\quad a_7 + a_{10} \frac{\beta^-_\tau}{\beta^+} - a_{16} \frac{\beta^-_\tau}{\beta^+} \chi_{\eta \tau} - a_{18} \frac{\beta^-_\tau}{\beta^+} \chi_{\eta \tau} \\
&\quad + a_8 - a_{10} \frac{\beta^+_\tau}{\beta^+} + a_{16} \chi_{\eta \tau} + a_{18} \chi_{\tau \tau} = \beta^-_\tau, \\
&\quad a_9 + a_{10} \frac{\beta^-}{\beta^+} = \beta^-, \\
&\quad a_{11} + a_{12} + a_{10} \left( \frac{\beta^-}{\beta^+} - 1 \right) = \beta^-, \\
&\quad a_{13} + a_{14} + a_{10} \left( \frac{\beta^-}{\beta^+} - 1 \right) = \beta^-, \\
&\quad a_{15} + a_{16} \frac{\beta^-}{\beta^+} = 0, \\
&\quad a_{17} + a_{18} \frac{\beta^-}{\beta^+} = 0, \\
&\quad a_{19} + a_{20} = 0.
\end{align*}\]

This is a system of ten linear equations for the $\gamma_m$’s. We can solve the system to get the coefficients $\gamma_m$’s of the finite difference scheme at the irregular grid point $(x_i, y_j, z_k)$. Once we know the $\gamma_m$’s, we know the $a_i$’s as well, so we can calculate the correction term from the following

\[C_{ijk} = a_{10} \left( \frac{[f]}{\beta^+} - \frac{\chi^+}{\beta^+} + \chi_{\eta \eta} - w_{\eta \eta} - w_{\tau \tau} \right) + a_{12} w_{\eta \eta} + a_{14} w_{\tau \tau} \]
\[+ a_{16} \frac{q_\eta}{\beta^+} + a_{18} \frac{q_\tau}{\beta^+} + a_{20} w_{\eta \tau} + a_2 [u] \]
\[+ \frac{1}{\beta^+} \left\{ a_4 + a_{10} \left( \chi_{\eta \eta} + \chi_{\tau \tau} - \frac{\beta^+}{\beta^+} \right) - a_{12} \chi_{\eta \eta} \right. \]
\[- a_{14} \chi_{\tau \tau} - a_{16} \frac{\beta^+}{\beta^+} - a_{18} \frac{\beta^+}{\beta^+} - a_{20} \chi_{\eta \tau} \left\} q \]
\[+ \left( a_6 - a_{10} \frac{\beta^+}{\beta^+} + a_{16} \chi_{\eta \eta} + a_{18} \chi_{\eta \tau} \right) w_\eta\]
\[
+ \left( a_8 - a_{10} \frac{\beta^+}{\beta^+} + a_{16} \chi_{\eta\tau} + a_{18} \chi_{\tau\tau} \right) w_\tau.
\]

If the grid point \((x_i, y_j, z_k)\) is on the + side, there are two ways to deal with it. The first one is to modify the correction term \(C_{ijk}\) and the linear system (2.38)–(2.47) a little bit. Use the following relation

\[
\lambda^+ u^+ = \lambda^- u^- + \lambda^+ u^+ - \lambda^- u^- = \lambda^- u^- + \lambda^+ [u] + [\lambda] u^-,
\]

(2.49)

and let the finite difference scheme at this irregular grid point be

\[
\sum_m^n \hat{\gamma}_m u_{i+m,j+k,m} + \lambda_{ijk} u_{ijk} = f_{ijk} + \hat{C}_{ijk}.
\]

(2.50)

Then \(\hat{\gamma}_m\) still satisfy equations (2.38)–(2.47), but the first equation becomes

\[
a_1 - a_{10} \frac{[\lambda]}{\beta^+} + a_2 = -[\lambda],
\]

(2.51)

and the correction term \(\hat{C}_{ijk}\) is

\[
\hat{C}_{ijk} = C_{ijk} + \lambda^+ [u] - [f].
\]

(2.52)

The other way is to simply reverse the roles of the two sides of the interface in the discussion above.

2.2.4 A special case

In the case where \(\beta, \lambda\) and \(f\) are continuous, we expect that the above finite difference scheme would reduce to the standard central finite difference scheme. Actually it does. If \(\beta\) and \(\lambda\) are continuous but vary with \(x, y\) and \(z\), we see that the linear system (2.38)–(2.47) reduces to

\[
\begin{align*}
    a_1 + a_2 &= 0 \\
    a_3 + a_4 &= \beta \xi \\
    a_5 + a_6 &= \beta \eta \\
    a_7 + a_8 &= \beta \tau \\
    a_9 + a_{10} &= \beta \\
    a_{11} + a_{12} &= \beta \\
    a_{13} + a_{14} &= \beta \\
    a_{15} + a_{16} &= 0 \\
    a_{17} + a_{18} &= 0 \\
    a_{19} + a_{20} &= 0.
\end{align*}
\]

(2.53)
This set of equations is satisfied to $O(h)$ by using the seven point stencil with

\[
\begin{align*}
\gamma_{i-1,jk} &= \beta_{i-1/2,jk}/h^2, & \gamma_{i+1,jk} &= \beta_{i+1/2,jk}/h^2, & \gamma_{i,j-1,k} &= \beta_{i,j-1/2,k}/h^2, \\
\gamma_{i,j+1,k} &= \beta_{i,j+1/2,k}/h^2, & \gamma_{ij,k-1} &= \beta_{ij,k-1/2}/h^2, & \gamma_{ij,k+1} &= \beta_{ij,k+1/2}/h^2, \\
\gamma_{ijk} &= -(\beta_{i-1/2,jk} + \beta_{i+1/2,jk} + \beta_{i,j-1/2,k} + \beta_{i,j+1/2,k} + \beta_{ij,k-1/2} + \beta_{ij,k+1/2})/h^2, \\
\gamma_{i+m,j+m,k+k_m} &= 0, \quad \text{otherwise.}
\end{align*}
\]

Here $\gamma_{i+m,j+m,k+k_m}$ (i.e., $\gamma_m$) denotes the coefficient corresponding to $u_{i+m,j+m,k+k_m}$. These are the coefficients for the standard formula (2.5). For elliptic equations with singular sources, since the linear system (2.38)–(2.47) does not depend on $f$, we still can use the standard central finite difference scheme and only need to add the correction terms at irregular grid points. Furthermore if $\beta$ is a constant, we have the following theorem.

**Theorem 2.1** If $\beta(x, y, z)$ is a constant and $\lambda$ is continuous, then the solution of equations (2.38)–(2.47) are

\[
\begin{align*}
\gamma_{i-1,jk} &= \gamma_{i+1,jk} = \gamma_{i,j-1,k} = \gamma_{i,j+1,k} = \gamma_{ij,k-1} = \gamma_{ij,k+1} = \beta/4, & \gamma_{ijk} &= -6\beta/8, \quad (2.54)
\end{align*}
\]

and all other coefficients $\gamma_{i+m,j+m,k+k_m} = 0$.

Proof: We only need to verify that these $\gamma_{ijk}$’s satisfy the system of equations (2.38)–(2.47). Without loss of generality, let the irregular grid point $(x_i, y_j, z_k)$ be the origin. The continuity condition $\lambda$ means $[\lambda] = 0$ and a constant $\beta$ means $[\beta] = 0$, $\rho = \beta^-/\beta^+ = 1$, $\beta_\xi = \beta_\eta = \beta_\tau = 0$, etc. Therefore the equations (2.38)–(2.47) now becomes

\[
\begin{align*}
\{ & a_1 + a_2 = 0 \\
& a_3 + a_4 = 0 \\
& a_5 + a_6 = 0 \\
& a_7 + a_8 = 0 \\
& a_9 + a_{10} = \beta \\
& a_{11} + a_{12} = \beta \\
& a_{13} + a_{14} = \beta \\
& a_{15} + a_{16} = 0 \\
& a_{17} + a_{18} = 0 \\
& a_{19} + a_{20} = 0.
\end{align*}
\]

The first equation

\[
a_1 + a_2 = 0, \quad \text{i.e.}, \quad \sum_m \gamma_m = 0,
\]
is obviously true. Under the transformation (2.9), let the new coordinates corresponding to 
(0, 0, 0), (−h, 0, 0), (0, −h, 0), (0, 0, h), (0, 0, −h) and (0, 0, h) be \((\xi_m, \eta_m, \tau_m)\), \(m = 1, \ldots, 7\). Let

\[
Y_m = \begin{pmatrix} \xi_m \\ \eta_m \\ \tau_m \end{pmatrix}, \quad m = 1, 2, \ldots, 7
\]

and

\[
X_1 = \begin{pmatrix} -x^* \\ -y^* \\ -z^* \end{pmatrix}, \quad X_2 = \begin{pmatrix} -h - x^* \\ -y^* \\ -z^* \end{pmatrix}, \quad X_3 = \begin{pmatrix} h - x^* \\ -y^* \\ -z^* \end{pmatrix}, \quad X_4 = \begin{pmatrix} -x^* \\ -h - y^* \\ -z^* \end{pmatrix},
\]

\[
X_5 = \begin{pmatrix} -x^* \\ h - y^* \\ -z^* \end{pmatrix}, \quad X_6 = \begin{pmatrix} -x^* \\ -y^* \\ -h - z^* \end{pmatrix}, \quad X_7 = \begin{pmatrix} -x^* \\ -y^* \\ h - z^* \end{pmatrix}.
\]

Then

\[
Y_m = AX_m, \quad m = 1, 2, \ldots, 7
\]

and

\[
\begin{pmatrix} a_3 + a_4 \\ a_5 + a_6 \\ a_7 + a_8 \end{pmatrix} = \begin{pmatrix} \sum_m \gamma_m \xi_m \\ \sum_m \gamma_m \eta_m \\ \sum_m \gamma_m \tau_m \end{pmatrix} = \sum_m \gamma_m Y_m = \sum_m \gamma_m AX_m
\]

\[
= \frac{\beta}{h^2} A \begin{pmatrix} 6x^* - h - x^* + h - x^* - x^* - x^* - x^* \\ 6y^* - y^* - y^* - h - y^* + h - y^* - y^* \\ 6z^* - z^* - z^* - z^* - z^* - z^* - h - z^* + h - z^* \end{pmatrix}
\]

\[
= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.
\]

Therefore, (2.39)–(2.41) are satisfied. To demonstrate the rest, first it is easy to find that

\[
\sum_m \gamma_m X_m X_m^T = \begin{pmatrix} 2\beta & 0 & 0 \\ 0 & 2\beta & 0 \\ 0 & 0 & 2\beta \end{pmatrix} = 2\beta I.
\]

Therefore

\[
\sum_m \frac{\gamma_m}{2} Y_m Y_m^T = \frac{1}{2} \sum_m \gamma_m AX_m X_m^T A^T = \beta I.
\]
since $AA^T = I$ ($A$ is the transformation matrix). On the other hand, note that

$$
\sum_m \frac{\gamma_m}{2} Y_m Y_m^T = \begin{pmatrix}
\sum_m \frac{\gamma_m}{2} \xi_m^2 & \sum_m \frac{\gamma_m}{2} \xi_m \eta_m & \sum_m \frac{\gamma_m}{2} \xi_m \tau_m \\
\sum_m \frac{\gamma_m}{2} \xi_m \eta_m & \sum_m \frac{\gamma_m}{2} \eta_m^2 & \sum_m \frac{\gamma_m}{2} \eta_m \tau_m \\
\sum_m \frac{\gamma_m}{2} \xi_m \tau_m & \sum_m \frac{\gamma_m}{2} \eta_m \tau_m & \sum_m \frac{\gamma_m}{2} \tau_m^2
\end{pmatrix},
$$

which implies that

$$a_9 + a_{10} = a_{11} + a_{12} = a_{13} + a_{14} = \beta,$$

$$a_{15} + a_{16} = a_{17} + a_{18} = a_{19} + a_{20} = 0.$$

\[\square\]

### 2.2.5 Representation of the interface

We use the level set approach to express the interface in this thesis. In this approach, the interface is defined as the zero level set of a smooth function $\phi$ defined on the entire domain $\Omega$ and

$$\begin{align*}
\phi(x, y, z) &< 0, & \text{if} & (x, y, z) \in \Omega^-,
\phi(x, y, z) &= 0, & \text{if} & (x, y, z) \in \Gamma,
\phi(x, y, z) &> 0, & \text{if} & (x, y, z) \in \Omega^+.
\end{align*}$$

The level set approach has many advantages. One obvious one to use the level set approach is: it is very easy to check whether a grid point is regular or irregular by using the following procedure:

1. Find the minimum value, $\phi_{\text{min}}$, of $\phi(\mathbf{X}_m)$ of seven grid points $\mathbf{X}_m$ in the standard seven point stencil.

2. Find the maximum value, $\phi_{\text{max}}$, of $\phi(\mathbf{X}_m)$ of seven grid points $\mathbf{X}_m$ in the standard seven point stencil.

3. If $\phi_{\text{max}}\phi_{\text{min}} > 0$, then it is a regular point.
   
   If $\phi_{\text{max}}\phi_{\text{min}} \leq 0$, then it is an irregular point.

Another advantage to use the level set approach is: much of the geometric information of the interface is completely determined by the level set function. As mentioned earlier, at each irregular grid point, we need to find out $\chi_{\eta\eta}$, $\chi_{\tau\tau}$ and $\chi_{\eta\tau}$. Actually, we have that on the interface

$$\phi(\xi, \eta, \tau) = 0. \quad (2.55)$$
Differentiating (2.55) with respect to \( \eta \) and \( \tau \) respectively yields
\[
\phi_\eta + \phi_\xi \chi_\eta = 0, \tag{2.56}
\]
\[
\phi_\tau + \phi_\xi \chi_\tau = 0. \tag{2.57}
\]
Differentiating (2.56) with respect to \( \eta \) and \( \tau \) respectively and (2.57) with respect to \( \tau \) yields
\[
\phi_{\eta\eta} + \phi_{\eta\xi} \chi_\eta + (\phi_{\xi\eta} + \phi_{\xi\xi} \chi_\eta) \chi_\eta + \phi_{\xi\eta\eta} = 0, \tag{2.58}
\]
\[
\phi_{\eta\tau} + \phi_{\eta\xi} \chi_\tau + (\phi_{\xi\eta} + \phi_{\xi\xi} \chi_\tau) \chi_\eta + \phi_{\xi\eta\tau} = 0, \tag{2.59}
\]
\[
\phi_{\tau\tau} + \phi_{\tau\xi} \chi_\tau + (\phi_{\xi\tau} + \phi_{\xi\xi} \chi_\tau) \chi_\tau + \phi_{\xi\tau\tau} = 0. \tag{2.60}
\]
Note that \( \chi(0,0) = 0 \), \( \chi_\eta(0,0) = 0 \), \( \chi_\tau(0,0) = 0 \). So we have
\[
\begin{cases}
\chi_{\eta\eta} = -\phi_{\eta\eta}/\phi_\xi \\
\chi_{\tau\tau} = -\phi_{\tau\tau}/\phi_\xi \tag{2.61} \\
\chi_{\eta\tau} = -\phi_{\eta\tau}/\phi_\xi,
\end{cases}
\]
where
\[
\begin{pmatrix}
\phi_\xi \\
\phi_\eta \\
\phi_\tau
\end{pmatrix}
= A 
\begin{pmatrix}
\phi_x \\
\phi_y \\
\phi_z
\end{pmatrix},
\]
and
\[
\begin{pmatrix}
\phi_{\xi\xi} & \phi_{\xi\eta} & \phi_{\xi\tau} \\
\phi_{\eta\xi} & \phi_{\eta\eta} & \phi_{\eta\tau} \\
\phi_{\tau\xi} & \phi_{\tau\eta} & \phi_{\tau\tau}
\end{pmatrix}
= A 
\begin{pmatrix}
\phi_{xx} & \phi_{xy} & \phi_{xz} \\
\phi_{yx} & \phi_{yy} & \phi_{yz} \\
\phi_{zx} & \phi_{zy} & \phi_{zz}
\end{pmatrix} A^T.
\]

Sometimes we may need curvatures of the interface. In the three space dimensions, we have many choices including the mean curvature \( \kappa_M \) and the Gaussian curvature \( \kappa_G \). They are also easily calculated from the level set function
\[
\kappa_M = \nabla \cdot \frac{\nabla \phi}{\|\nabla \phi\|} = \left\{ \frac{\phi_{yy} + \phi_{zz} \phi_x^2 + (\phi_{xx} + \phi_{zz}) \phi_y^2 + (\phi_{xx} + \phi_{yy}) \phi_z^2}{\phi_x^2 + \phi_y^2 + \phi_z^2} \right\}. \tag{2.62}
\]
\[
\kappa_G = \frac{\phi_x^2(\phi_{yy} \phi_{zz} - \phi_{yy}^2) + \phi_y^2(\phi_{xx} \phi_{zz} - \phi_{xx}^2) + \phi_z^2(\phi_{xx} \phi_{yy} - \phi_{xy}^2)}{\phi_x^2 + \phi_y^2 + \phi_z^2} \left\{ \frac{\phi_{xx}(\phi_{yy} \phi_{zz} - \phi_{yy}^2) + \phi_{yy}(\phi_{xx} \phi_{zz} - \phi_{xx}^2) + \phi_{zz}(\phi_{xx} \phi_{yy} - \phi_{xy}^2)}{\phi_x^2 + \phi_y^2 + \phi_z^2} \right\}. \tag{2.63}
\]
It should be pointed out that here we need values of $\phi, \phi_x, \phi_y, \phi_z, \phi_{xx}, \phi_{xy}, \phi_{xz}, \phi_{yy}, \phi_{yz}$ and $\phi_{zz}$ at some point on the interface, which may not lie on the grid. If the expression of the level set function is explicitly given, the accurate values of them at any point on the interface can be computed. However, this is not the case in real, especially for those moving interface problems. Instead only $\phi(x_i, y_j, z_k)$ at grid points are given or approximated. In this case, we use interpolation to compute $\phi, \phi_x, \phi_y, \phi_z, \phi_{xx}, \phi_{xy}, \phi_{xz}, \phi_{yy}, \phi_{yz}$ and $\phi_{zz}$.

We will discuss this after the next two subsections.

### 2.2.6 Choosing $\xi$, $\eta$ and $\tau$

In two-dimensional cases, the new local coordinates $(\xi, \eta)$ are chosen so that $\xi$ is parallel to the normal direction of the interface and $\eta$ is the corresponding unique tangential direction. In three-dimensional cases, $\xi$ is still chosen to be parallel to the normal direction of the interface pointing outward, i.e.,

$$\xi = \frac{\nabla \phi}{\| \nabla \phi \|} = (\phi_x, \phi_y, \phi_z)^T / \sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2},$$

and the $\eta$- and $\tau$- axes are in the tangent plane passing through $(x^*, y^*, z^*)$. However unlike the two-dimensional case, we have to choose two tangential directions. In practice, we choose the first tangential direction as

$$\eta = (\phi_y, -\phi_x, 0)^T / \sqrt{\phi_x^2 + \phi_y^2},$$

or

$$\eta = (\phi_z, 0, -\phi_x)^T / \sqrt{\phi_x^2 + \phi_z^2},$$

and the corresponding second tangential direction as

$$\tau = \frac{s}{\|s\|}, \text{ where } s = (\phi_x \phi_z, \phi_y \phi_z, -\phi_x \phi_y - \phi_y \phi_x)^T,$$

or

$$\tau = \frac{t}{\|t\|}, \text{ where } t = (-\phi_x \phi_y, \phi_x \phi_z + \phi_z \phi_x, -\phi_y \phi_z)^T$$

respectively.
2.2.7 Finding projections

For each irregular grid point \((x_i, y_j, z_k)\), we need to select a point, say \((x_i^*, y_j^*, z_k^*)\), on the interface. Although not necessarily, we usually take this point as the projection of \((x_i, y_j, z_k)\) on the interface, which is the closest to \((x_i, y_j, z_k)\).

Considering an irregular grid point \(x = (x_i, y_j, z_k)\), we use the following approach to find the projection on the interface.

1. Find the unit steepest ascent direction \(p\) at \(x\)
   \[
p = \frac{\nabla \phi}{\|\nabla \phi\|}.
   \]

2. Locate the projection of \(x\) on the interface along the direction \(p\)
   \[
   X^* = x + \alpha p,
   \]
   where \(\alpha\) is determined from the following quadratic equation
   \[
   \phi(x) + \|\nabla \phi\| \alpha + \frac{1}{2} (p^T \text{He}(\phi) p) \alpha^2 = 0,
   \]
   where \(\text{He}(\phi)\) is the Hessian matrix of \(\phi\)
   \[
   \text{He}(\phi) = \begin{pmatrix}
   \phi_{xx} & \phi_{xy} & \phi_{xz} \\
   \phi_{yx} & \phi_{yy} & \phi_{yz} \\
   \phi_{zx} & \phi_{zy} & \phi_{zz}
   \end{pmatrix}.
   \]

Here we need values of \(\phi, \phi_x, \phi_y, \phi_z, \phi_{xx}, \phi_{xy}, \phi_{xz}, \phi_{yy}, \phi_{yz}\) and \(\phi_{zz}\) at the grid point. If the expression of the level set function is not explicitly given, we can simply use the central finite difference scheme to approximate them.

2.2.8 The interpolations

We need values of \(\phi, \phi_x, \phi_y, \phi_z, \phi_{xx}, \phi_{xy}, \phi_{xz}, \phi_{yy}, \phi_{yz}\) and \(\phi_{zz}\) at some point on the interface, which may not lie on the grid. We may use eight point linear interpolation to approximate them. Given any point \((x, y, z)\), we can find a cube containing the point, say, eight vertices are \((x_0, y_0, z_0)\), \((x_0, y_0, z_1)\), \((x_0, y_1, z_0)\), \((x_0, y_1, z_1)\), \((x_1, y_0, z_0)\), \((x_1, y_0, z_1)\), \((x_1, y_1, z_0)\) and \((x_1, y_1, z_1)\). Let \(Q\) be a quantity and \(Q_{ijk}\) be its values at grid points. Then the eight point linear interpolation is defined as
\[
Q(x, y, z) = \frac{1}{8} \sum_{i,j,k=0}^1 Q_{ijk} \bar{x}_i \bar{y}_j \bar{z}_k,
\] (2.64)
where
\[
\begin{align*}
\tilde{x}_i &= 1 + (2i - 1) \left( \frac{2(x - x_0)}{h} - 1 \right), \\
\tilde{y}_j &= 1 + (2j - 1) \left( \frac{2(y - y_0)}{h} - 1 \right), \\
\tilde{z}_k &= 1 + (2k - 1) \left( \frac{2(z - z_0)}{h} - 1 \right).
\end{align*}
\]

We may use a more general interpolation scheme
\[
Q(x, y, z) = \sum_{(i,j,k) \in N} \alpha_{ijk} Q_{ijk}, 
\]  
(2.65)

where \(N\) denotes a set of neighboring grid points near \((x, y, z)\), and the coefficients \(\alpha_{ijk}\)’s can be determined by solving a linear system
\[
\begin{align*}
\sum \alpha_{ijk} &= 1 \\
\sum (x_i - x) \alpha_{ijk} &= 0 \\
\sum (y_j - y) \alpha_{ijk} &= 0 \\
\sum (z_k - z) \alpha_{ijk} &= 0,
\end{align*}
\]  
(2.66)
or
\[
\begin{align*}
\sum \alpha_{ijk} &= 1 \\
\sum (x_i - x)^2 \alpha_{ijk} &= 0 \\
\sum (y_j - y)^2 \alpha_{ijk} &= 0 \\
\sum (z_k - z)^2 \alpha_{ijk} &= 0, \\
\sum (x_i - x)(y_j - y) \alpha_{ijk} &= 0 \\
\sum (x_i - x)(z_k - z) \alpha_{ijk} &= 0 \\
\sum (y_j - y)(z_k - z) \alpha_{ijk} &= 0.
\end{align*}
\]  
(2.67)

At least four grid points are needed in (2.66) and ten grid points in (2.67). Usually we need use more points, especially for (2.67), which gives an underdetermined linear system.

We then may solve it by utilizing the existing program such as \texttt{ssvdc} from Linpack.

Note that we could use the same interpolation scheme to find values of the derivatives of \(Q\) at some projection points. The only difference is to use a different right-hand side in (2.66)
or (2.67). For example, to compute $Q_x$, we use $(0, 1, 0, 0)$ in (2.66) or $(0, 1, 0, 0, 0, 0, 0, 0, 0)$ in (2.67), and to compute $Q_{yz}$, we use $(0, 0, 0, 0, 0, 0, 0, 1)$ in (2.67).

2.2.9 An optimization approach

Equations (2.38)–(2.47) form a system of linear equations for ten unknowns. In general, we need $n_s \geq 10$ in order to get second order accuracy. Especially, if $n_s = 10$, we have a determined linear system for $\gamma_m$’s.

As we saw before, if $\beta$ is continuous, then by solving (2.38)–(2.47), we actually recover the standard seven point stencil coefficients

$$
\gamma_{i-1,j,k} = \gamma_{i+1,j,k} = \gamma_{i,j-1,k} = \gamma_{i,j+1,k} = \gamma_{i,j,k-1} = \gamma_{i,j,k+1} = \frac{\beta}{h^2}, \quad \gamma_{ijk} = \frac{6\beta}{h^2}.
$$

In general, however, the resulting $\gamma_m$’s are different from those in the standard seven point stencil. The exact nature of the coefficients depends on how large the jump in $\beta$ is. We have not investigated these coefficients in general, but at least for reasonably mild discontinuities it seems that:

- The contributions to the difference schemes at irregular grid points are mainly from the standard seven point stencil. These coefficients are $O(1/h^2)$, while the contributions from the ‘additional points’ are typically much smaller. The magnitude depends on the jump in $\beta$ and the geometry of the interface.

- All the coefficients except occasionally additional points’ coefficients have the same sign (− for the diagonal and + for the off-diagonal) as in the standard central finite difference scheme. Since the contribution from the additional points is much smaller than that from the standard seven points, we expect the classical theoretical analysis to still be applicable for the resulting linear system with slight modifications. In particular, the system is nearly diagonally dominant, and strictly so if additional points’ coefficients are always positive.

However, when $n_s = 10$, some difficulties may occur.

- The linear system (2.38)–(2.47) may be ill-conditioned, especially when the curvature of the interface is big, or the ratio $\beta^+ / \beta^-$ is too big or too small. Actually, in some numerical experiments, even if the interface is a sphere and the jump in $\beta$ is small, some linear systems are numerically unsolvable.

- Even the linear system (2.38)–(2.47) is solvable for each irregular grid point, we still can not guarantee that the final finite difference equation is solvable. An iterative method like the SOR method may not converge.
To resolve these problems, we choose \( n_s > 10 \). In this case, (2.38)–(2.47) is an under-determined linear system, and therefore will have an infinite number of solutions. We should choose one which guarantees the final finite difference equation is solvable at least by the SOR method, for example, we want the finite difference scheme to preserve the maximum principle property [61].

To do so, we need to impose the sign property on the coefficients \( \gamma_m \)'s in (2.4)

\[
\begin{align*}
\gamma_m &< 0, & \text{if } (i_m, j_m, k_m) = (0, 0, 0), \\
\gamma_m &\geq 0, & \text{if } (i_m, j_m, k_m) \neq (0, 0, 0),
\end{align*}
\]

along with (2.38)–(2.47) to keep the truncation error small. Note that at regular grid points, the standard central finite difference scheme satisfies the above sign property and (2.38)–(2.47), as we have proved. So, we will only concentrate our discussion on an irregular grid point \((x_i, y_j, z_k)\).

Also, there are still an infinite number of solutions satisfying the sign property and (2.38)–(2.47). We would choose the one with some minimum 2-norm. We form the following quadratic constrained optimization problem to determine the coefficients of the finite difference scheme

\[
\min_\gamma \frac{1}{2} \sum_m (\gamma_m - d_m)^2,
\]

s.t.

\[
\begin{align*}
A\gamma &= b, \\
\gamma_m &< 0, & \text{if } (i_m, j_m, k_m) = (0, 0, 0), \\
\gamma_m &\geq 0, & \text{if } (i_m, j_m, k_m) \neq (0, 0, 0),
\end{align*}
\]

where \( \gamma = (\gamma_1, \gamma_2, \cdots, \gamma_{n_s})^T \) is the vector composed of coefficients of the finite difference scheme, and \( A\gamma = b \) denotes the equality constraints specified by (2.38)–(2.47). We also want to choose \( \gamma_m \)'s in such a way that the finite difference scheme (2.4) reduces to the standard central finite difference scheme if there is no interface or no jump in the coefficient across the interface. This can be done by choosing

\[
\begin{align*}
d_m &= \frac{\beta_{i+i_m, j+j_m, k+k_m}}{h^2}, & \text{if } (i_m, j_m, k_m) \text{ is one of the six neighbors of } (0, 0, 0), \\
d_m &= -\frac{6 \beta_{i, j, k}}{h^2}, & \text{if } (i_m, j_m, k_m) = (0, 0, 0), \\
d_m &= 0, & \text{otherwise}.
\end{align*}
\]
Chapter 2. A Maximum Principle Preserving Scheme for 3D Interface Problems

There are several commercial and educational software packages that are designed to solve constrained quadratic optimization problems, such as \texttt{QP} in Matlab and \texttt{QL} by K. Schittkowski.

Most of quadratic optimization solvers require users to provide the initial guess, the lower and the upper bounds and some other information. In our approach, we take the initial guess as \( d_m, m = 1, 2, \cdots, n_s \), the lower and the upper bounds as
\[
0 < \gamma_m < \frac{\beta_{\text{max}}}{h^2}, \quad \text{if } \ (i_m, j_m, k_m) \neq (0, 0, 0),
\]
\[
-6 \frac{\beta_{\text{max}}}{h^2} < \gamma_m < 0, \quad \text{if } \ (i_m, j_m, k_m) = (0, 0, 0),
\]
where \( \beta_{\text{max}} \) is an estimation of the upper bound of the coefficient \( \beta \).

\[2.72\]

\[\text{2.2.10 An algebraic multigrid solver}\]

The final linear system derived from the maximum principle preserving scheme is not symmetric so some existing fast Poisson solvers can not be used. We can use an iterative method. For instance, the SOR method converges since the finite difference equations satisfy the maximum principle. For small systems, the SOR method was shown to be good enough. However, for large systems, the SOR method was too slow.

We use the Algebraic Multigrid method (AMG) \[69\], which has been shown to be a robust, efficient solver for linear systems \( Lu = F \). For theoretical reason, the coefficient matrix \( L \) should hold

- \( L \) is positive definite or semi-positive definite with ROWSUM = 0 for each row, where ROWSUM denotes the sum of a row.
- \( L \) is "essentially" positive type, i.e.,
  - The diagonal entries of \( L \) must be positive.
  - Most of the off-diagonal entries of \( L \) are non-positive.
  - For each row, the ROWSUM should be non-negative.

Although the final linear system derived from the maximum principle preserving scheme is not "essentially" positive, it can be easily changed to be "essentially" positive by just negating the coefficient matrix. Our numerical experiments showed that the AMG method was much faster than the SOR method when used to solve large scale problems.
2.3 Numerical results

We have done a number of numerical experiments which confirm the expected order of accuracy for the maximum principle preserving scheme. The computation are done using CRAY T90 at North Carolina Supercomputing Center (NCSC). The computational domain is \([-1, 1] \times [-1, 1] \times [-1, 1]\). In all examples, \(l = m = n\) and \(n_s = 27\) (i.e., all 27 grid points involved in the usual 27 point stencil) unless otherwise specified. The linear system solver used is the AMG solver \texttt{amg1r5.f} developed by the German National Research Center for Information Technology (GMD), and the tolerance is \(10^{-4}\).

Example 2.1 In this example we consider a problem with a piecewise constant coefficient \(\beta\) and a discontinuous source term \(f\). The interface is a sphere \(x^2 + y^2 + z^2 = 1/4\). The differential equation is

\[(\beta u_x)_x + (\beta u_y)_y + (\beta u_z)_z = f,
\]

with

\[
\beta(x, y, z) = \begin{cases} 
\beta^-, & \text{if } r < \frac{1}{2}, \\
\beta^+, & \text{if } r \geq \frac{1}{2},
\end{cases}
\]

\[
f(x, y, z) = \begin{cases} 
-200\beta^-r^2, & \text{if } r < \frac{1}{2}, \\
20\beta^+r^2, & \text{if } r \geq \frac{1}{2}.
\end{cases}
\]

Dirichlet boundary conditions and the jump conditions (2.2) and (2.3) are determined from the exact solution

\[
u(x, y, z) = \begin{cases} 
-10r^4, & \text{if } r < \frac{1}{2}, \\
r^4, & \text{if } r \geq \frac{1}{2},
\end{cases}
\]

i.e.

\[
[u] = 11r_0^4 = 11/16,
\]

\[[\beta u_n] = 4 \left(10\beta^- + \beta^+\right) r_0^3 = \left(10\beta^- + \beta^+\right)/2,
\]

where \(r = \sqrt{x^2 + y^2 + z^2}\) and on \(\Gamma\), \(r = r_0 = 1/2\).

We tested a small jump case with \(\beta^- = 1\), \(\beta^+ = 2\) and a big jump case with \(\beta^- = 1\), \(\beta^+ = 1000\). Figure 2.2 shows a slice of the computed solution. We see that the maximum principle preserving scheme does accurately give the jumps in the solution and in the normal derivative of the solution.

Table 2.1 shows the results of a grid refinement analysis, where \(l + 1 = m + 1 = n + 1\) is the number of uniform grid points in each direction. The maximum error over all grid
Chapter 2. A Maximum Principle Preserving Scheme for 3D Interface Problems

... points

\[ \| E_n \|_\infty = \frac{\max_{i,j,k} | u(x_i, y_j, z_k) - u_{ijk} |}{\max_{i,j,k} | u(x_i, y_j, z_k) |}, \]  

(2.74)

is presented, where \( u_{ijk} \) is the computed approximation of \( u(x_i, y_j, z_k) \). We also display the ratio of two successive errors

\[ \text{ratio} = \frac{\| E_n \|_\infty}{\| E_{2n} \|_\infty}. \]  

(2.75)

For a first order method, the ratio approaches to 2, and for a second order method, the ratio approaches to 4. We will use the same notation for other examples in this thesis. We see that an average ratio of 4 indicates that the maximum principle preserving scheme is a second order method.

Figure 2.2: (a) A slice of the computed solution (\( \beta^+ = 1000, \beta^- = 1 \) and \( n = 52 \)) for Example 2.1: \( -u(x, y, 0) \). (b) The error plot of the slice of the computed solution.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( | E_n |_\infty )</th>
<th>ratio</th>
<th>( | E_{2n} |_\infty )</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>( 2.320 \times 10^{-3} )</td>
<td></td>
<td>( 1.003 \times 10^{-3} )</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>( 4.954 \times 10^{-1} )</td>
<td>4.683</td>
<td>( 2.377 \times 10^{-3} )</td>
<td>4.220</td>
</tr>
<tr>
<td>104</td>
<td>( 1.261 \times 10^{-1} )</td>
<td>3.929</td>
<td>( 6.358 \times 10^{-9} )</td>
<td>3.739</td>
</tr>
</tbody>
</table>

Table 2.1: The grid refinement analysis for Example 2.1.
Example 2.2 In this example we consider a problem with a variable and discontinuous coefficient $\beta$ but a continuous source term $f$. The interface is a sphere $x^2 + y^2 + z^2 = 1/4$. The differential equation is

$$(\beta u_x)_x + (\beta u_y)_y + (\beta u_z)_z = f,$$

with

$$\beta(x, y, z) = \begin{cases} r^2 + 1, & \text{if } r < \frac{1}{2}, \\ b, & \text{if } r \geq \frac{1}{2}, \end{cases}$$

$$f(x, y, z) = 10r^2 + 6,$$

where $b$ is a constant. Dirichlet boundary conditions and the jump conditions (2.2) and (2.3) are determined from the exact solution

$$u(x, y, z) = \begin{cases} r^2, & \text{if } r < \frac{1}{2}, \\ \frac{r^4}{2} + r^2 - \left(\frac{r_0^4}{2} + r_0^2\right)/b + r_0^2, & \text{if } r \geq \frac{1}{2}, \end{cases}$$

(2.76)

i.e.

$$[u] = 0,$$

$$[\beta u_n] = 0,$$

where $r = \sqrt{x^2 + y^2 + z^2}$, and on $\Gamma$, $r = r_0 = 1/2$. Note that there are no jumps in $u$ and $\beta u_n$. But since $\beta$ is discontinuous, there is still a jump in the normal derivative $u_n$.

The jump in the coefficient $\beta$ depends on the choice of the constant $b$. We tested three different cases, $b = 1$, $b = 10$ (small jump), and $b = 1000$ (large jump). Figure 2.3 shows a slice of the computed solution. Table 2.2 shows the results of a grid refinement analysis. When $h$ is small enough, we again obtain second order accuracy at all grid points.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$b = 1$</th>
<th>$|E_n|_\infty$</th>
<th>ratio</th>
<th>$b = 10$</th>
<th>$|E_n|_\infty$</th>
<th>ratio</th>
<th>$b = 1000$</th>
<th>$|E_n|_\infty$</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td></td>
<td>$1.247 \times 10^{-3}$</td>
<td></td>
<td>52</td>
<td>$3.979 \times 10^{-4}$</td>
<td>3.134</td>
<td>104</td>
<td>$9.592 \times 10^{-5}$</td>
<td>4.148</td>
</tr>
<tr>
<td>52</td>
<td>3.979 $\times 10^{-4}$</td>
<td>3.134</td>
<td></td>
<td>104</td>
<td>$9.592 \times 10^{-5}$</td>
<td>4.148</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>$9.592 \times 10^{-5}$</td>
<td>4.148</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: The grid refinement analysis for Example 2.2.
Figure 2.3: A slice of the computed solution ($n = 52$) for Example 2.2: $-u(x, y, 0)$. (a) $b = 1$. (b) $b = 10$. (c) $b = 1000$. 
Example 2.3 In this example we consider a problem with a piecewise constant coefficient $\beta$ and a discontinuous source term $f$ again, but the interface is an ellipsoid $x^2 + 4y^2 + 2z^2 = 1/4$. The differential equation is

$$(\beta u_x)_x + (\beta u_y)_y + (\beta u_z)_z = f,$$

with

$$\beta(x, y, z) = \begin{cases} 
\beta^-, & \text{if } x^2 + 4y^2 + 2z^2 < \frac{1}{4}, \\
\beta^+, & \text{if } x^2 + 4y^2 + 2z^2 \geq \frac{1}{4},
\end{cases}$$

$$f(x, y, z) = \begin{cases} 
-2\beta^-, & \text{if } x^2 + 4y^2 + 2z^2 < \frac{1}{4}, \\
-3\beta^+ \sin x \cos y \cos z, & \text{if } x^2 + 4y^2 + 2z^2 \geq \frac{1}{4}.
\end{cases}$$

Dirichlet boundary conditions and the jump conditions (2.2) and (2.3) are determined from the exact solution

$$u(x, y, z) = \begin{cases} 
x^2 - y^2 - z^2, & \text{if } x^2 + 4y^2 + 2z^2 < \frac{1}{4}, \\
\sin x \cos y \cos z, & \text{if } x^2 + 4y^2 + 2z^2 \geq \frac{1}{4},
\end{cases}$$

(2.77)

i.e.

$$[u] = \sin x \cos y \cos z - (x^2 - y^2 - z^2),$$

where $[\beta u_n] = [\beta u_\xi]$ is determined by

$$\begin{pmatrix} [\beta u_\xi] \\ [\beta u_n] \\ [\beta u_\tau] \end{pmatrix} = A \begin{pmatrix} \beta^+ \cos x \cos y \cos z - 2\beta^- x \\
-\beta^+ \sin x \sin y \cos z + 2\beta^- y \\
-\beta^+ \sin x \cos y \sin z + 2\beta^- z \end{pmatrix},$$

and $A$ is the local coordinate transformation matrix.

This problem is not axsi-symmetric. We also tested two different cases, a small jump case with $\beta^- = 1, \beta^+ = 2$ and a big jump case with $\beta^- = 1, \beta^+ = 1000$. Figure 2.4 shows a slice of the computed solution, and Table 2.3 shows the results of a grid refinement analysis. Again second order accuracy is observed.
Figure 2.4: A slice of the computed solution ($\beta^+ = 1000$, $\beta^- = 1$ and $n = 52$) for Example 2.3: $u(x, y, 0)$.

Example 2.4 In this example we consider a problem with two interfaces. The level set function is

$$\phi(x, y, z) = S_1(x, y, z)S_2(x, y, z),$$

where

$$S_1(x, y, z) = (x - 0.2)^2 + 2(y - 0.2)^2 + z^2 - 0.01,$$

$$S_2(x, y, z) = 3(x + 0.2)^2 + (y + 0.2)^2 + z^2 - 0.01.$$  

It is obvious that $S_1(x, y, z)$ is an ellipsoid centered at $(0.2, 0.2, 0)$ and $S_2(x, y, z)$ is another ellipsoid centered at $(-0.2, -0.2, 0)$, see Figure 2.5. The differential equation is

$$(\beta u_x)_x + (\beta u_y)_y + (\beta u_z)_z = f.$$  

Table 2.3: The grid refinement analysis for Example 2.3.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\beta^+ = 2$, $\beta^- = 1$</th>
<th>$\beta^+ = 1000$, $\beta^- = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$|E_n|_\infty$</td>
<td>ratio</td>
</tr>
<tr>
<td>26</td>
<td>$1.125 \times 10^{-2}$</td>
<td>1.000</td>
</tr>
<tr>
<td>52</td>
<td>$1.504 \times 10^{-3}$</td>
<td>7.480</td>
</tr>
<tr>
<td>104</td>
<td>$3.219 \times 10^{-4}$</td>
<td>4.672</td>
</tr>
</tbody>
</table>
with
\[\beta(x, y, z) = \begin{cases} 
\beta^-, & \text{in } \Omega^- \\
\beta^+, & \text{in } \Omega^+
\end{cases}\]
and
\[f(x, y, z) = \begin{cases} 
6\beta^-e^{x+2y+z}, & \text{in } \Omega^-, \\
-\beta^+(4\pi^2 \sin 2\pi x + \pi^2 \sin \pi y + 16\pi^2 \sin 4\pi z), & \text{in } \Omega^+.
\end{cases}\]

Dirichlet boundary conditions and the jump conditions (2.2) and (2.3) are determined from the exact solution
\[u(x, y, z) = \begin{cases} 
e^{x+2y+z}, & \text{in } \Omega^-, \\
\sin 2\pi x + \sin \pi y + \sin 4\pi z, & \text{in } \Omega^+,
\end{cases}\]
(2.78)
i.e.
\[\left[ u \right] = \sin 2\pi x + \sin \pi y + \sin 4\pi z - e^{x+2y+z},\]
where \([\beta u_n] = [\beta u_\xi]\) is given by
\[
\begin{pmatrix} [\beta u_\xi] \\ [\beta u_\eta] \\ [\beta u_\tau] \end{pmatrix} = A \begin{pmatrix} 2\pi \beta^+ \cos 2\pi x - \beta^-e^{x+2y+z} \\ \pi \beta^+ \cos \pi y - 2\beta^-e^{x+2y+z} \\ 4\pi \beta^+ \cos 4\pi z - \beta^-e^{x+2y+z} \end{pmatrix},
\]
and \(A\) is the local coordinate transformation matrix.

**Figure 2.5:** The interface of two ellipsoids for Example 2.4.

We tested two different cases, \(\beta^- = \beta^+ = 1\) and \(\beta^- = 1, \beta^+ = 1000\). Figure 2.6 shows a slice of the computed solution for the big jump case, and Table 2.4 shows the results of a grid refinement analysis. Again second order accuracy is observed.
Figure 2.6: A slice of the computed solution \((\beta^+ = 1000, \beta^- = 1 \text{ and } n = 104)\) for Example 2.4: \(u(x,y,0)\).

Table 2.4: The grid refinement analysis for Example 2.4.

<table>
<thead>
<tr>
<th>(n)</th>
<th>(| E_n |<em>{\infty}) ratio (| E_n |</em>{\infty}) ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>(3.108 \times 10^{-2})</td>
</tr>
<tr>
<td>104</td>
<td>(6.758 \times 10^{-3})</td>
</tr>
</tbody>
</table>

2.3.1 Algorithm efficiency analysis

A. Cost for dealing with irregular grid points

A natural concern about the maximum principle preserving scheme is how much extra cost needed for dealing with the irregular grid points. The cost includes indexing the irregular grid points, finding projections, and solving the quadratic optimization problem at each irregular grid point. It depends on the number of irregular points and the choice of \(n_s\). For simplicity, we use \(n_s = 27\) in our numerical experiments. But \(n_s\) can be chosen smaller. Once \(n_s\) is chosen, notice that the cost is linearly proportional to the number of irregular points, which is proportional to the mesh size \(n\). But the cost for solving the final linear
system is not just linearly proportional to the mesh size \( n \). So we expect that for small mesh size \( n \), the cost for dealing with irregular grid points will make a big part, but as \( n \) increases, it will become smaller and smaller. Our numerical experiments confirmed this. See Table 2.5, where \( n_{\text{irreg}} \) denotes the total number of irregular grid points, \( T_{\text{pre}} \) the total CPU time excluding solving the finite difference equations, and \( T_{\text{total}} \) the total CPU time. The unit of CPU time is second(s).

### Table 2.5: CPU time for dealing with irregular grid points.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_{\text{irreg}} )</th>
<th>( T_{\text{pre}} )</th>
<th>( T_{\text{total}} )</th>
<th>( n_{\text{irreg}} )</th>
<th>( T_{\text{pre}} )</th>
<th>( T_{\text{total}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>920</td>
<td>33.000</td>
<td>39.977</td>
<td>480</td>
<td>18.365</td>
<td>26.034</td>
</tr>
<tr>
<td>52</td>
<td>3528</td>
<td>128.344</td>
<td>204.970</td>
<td>1936</td>
<td>74.809</td>
<td>157.538</td>
</tr>
<tr>
<td>104</td>
<td>14048</td>
<td>522.472</td>
<td>1559.333</td>
<td>7704</td>
<td>304.395</td>
<td>1338.226</td>
</tr>
</tbody>
</table>

### B. CPU time versus geometry and the jump

Basically, the CPU time used in the entire solution process depends on the geometry and the jump in the coefficient. Table 2.6—Table 2.9 give some statistics for the examples in this section. Especially, it has been observed that the CPU time does depend on the jump in \( \beta \), but a big jump does not increase CPU time greatly.

### Table 2.6: CPU time for Example 2.1.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_{\text{irreg}} )</th>
<th>( \beta^+ = 2, \beta^- = 1 )</th>
<th>( \beta^+ = 1000, \beta^- = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>920</td>
<td>38.421</td>
<td>44.990</td>
</tr>
<tr>
<td>52</td>
<td>3528</td>
<td>193.302</td>
<td>221.704</td>
</tr>
<tr>
<td>104</td>
<td>14048</td>
<td>1356.526</td>
<td>1612.131</td>
</tr>
</tbody>
</table>
### Table 2.7: CPU time for Example 2.2.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n_{irreg}$</th>
<th>$b = 1$</th>
<th>$b = 10$</th>
<th>$b = 1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>920</td>
<td>37.446</td>
<td>39.977</td>
<td>43.809</td>
</tr>
<tr>
<td>52</td>
<td>3528</td>
<td>185.485</td>
<td>204.970</td>
<td>214.978</td>
</tr>
<tr>
<td>104</td>
<td>14048</td>
<td>1473.611</td>
<td>1559.333</td>
<td>1771.889</td>
</tr>
</tbody>
</table>

### Table 2.8: CPU time for Example 2.3.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n_{irreg}$</th>
<th>$\beta^+ = 2, \beta^- = 1$</th>
<th>$\beta^+ = 1000, \beta^- = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>480</td>
<td>22.486</td>
<td>26.034</td>
</tr>
<tr>
<td>52</td>
<td>1936</td>
<td>136.285</td>
<td>157.538</td>
</tr>
<tr>
<td>104</td>
<td>7704</td>
<td>1287.891</td>
<td>1338.226</td>
</tr>
</tbody>
</table>

### Table 2.9: CPU time for Example 2.4.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n_{irreg}$</th>
<th>$\beta^+ = 1, \beta^- = 1$</th>
<th>$\beta^+ = 1000, \beta^- = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>220</td>
<td>78.999</td>
<td>115.070</td>
</tr>
<tr>
<td>104</td>
<td>874</td>
<td>972.991</td>
<td>1159.059</td>
</tr>
</tbody>
</table>

C. Linear system solver: SOR versus AMG

Since the finite difference equations satisfy the maximum principle, it is guaranteed that the SOR method converges. Also, by negating the coefficient matrix, we can call an AMG solver. We compare the two methods using some examples in this section. It appears that for small systems, the SOR method is shown to be good enough. However, for large system the AMG method is shown to be much better than the SOR method, see Table 2.10, where the computation is done on Sun Ultra 10.

#### 2.3.2 Summary of the maximum principle preserving scheme

In summary, we have developed a second order accurate difference method for elliptic equations in the following situations: (i) The differential equations have discontinuous coefficients along a general interface. (ii) The differential equations have singular sources along a general interface. (iii) The differential equations have externally imposed constraints on the jump in $u$ or normal derivatives of $u$ across an interface. In all cases we are able to derive an appropriate finite difference scheme with second order accuracy at all points on a uniform
Table 2.10: Comparison of CPU time between the SOR method and the AMG method.

<table>
<thead>
<tr>
<th>n</th>
<th>Example 2.1 ($\beta^+ = 10000$, $\beta^- = 1$)</th>
<th>Example 2.2 ($b = 10000000$)</th>
<th>Example 2.3 ($\beta^+ = 10000$, $\beta^- = 1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SOR</td>
<td>AMG</td>
<td>SOR</td>
</tr>
<tr>
<td>20</td>
<td>0.27</td>
<td>1.47</td>
<td>0.21</td>
</tr>
<tr>
<td>40</td>
<td>29.72</td>
<td>22.54</td>
<td>27.51</td>
</tr>
<tr>
<td>80</td>
<td>1457.91</td>
<td>212.90</td>
<td>1410.54</td>
</tr>
</tbody>
</table>

grid.

In the special case where the coefficients are continuous, the difference stencil reduces to the standard seven point stencil and only the right-hand side needs to be corrected to obtain second order accuracy. In particular, if the coefficients are constant then the standard seven point stencil is used and a fast Poisson solver can be used to solve the resulting linear system.

The ideas presented here can be used on a wide variety of other problems with discontinuous coefficients or singular sources. All we need are the jumps in the solution and its first derivatives across $\Gamma$. These jumps are used in conjunction with appropriate Taylor series expansions about the interface to derive the finite difference scheme and the right-hand side.

Based on the maximum principle preserving scheme we have written an application program. Although not optimized, it has been used successfully to deal with many testing interface problems. In the future, we plan to extend it to an application package for solving elliptic interface problems on cubic regions.
Chapter 3

A Fast Algorithm for 3D Elliptic Interface Problems

In this chapter, we develop a second order fast algorithm for elliptic equations with piecewise constant but discontinuous coefficients.

3.1 Introduction

The problem we intend to solve is

Problem (I).

\[ \nabla \cdot (\beta \nabla u) = f, \quad \text{in } \Omega, \quad (3.1) \]

Given boundary conditions on \( \partial\Omega \),

\[ [u] = w, \quad \text{on } \Gamma, \quad (3.3) \]

\[ [\beta u_n] = q, \quad \text{on } \Gamma. \quad (3.4) \]

There are two main concerns in solving Problem (I) numerically. One is how to discretize it to certain accuracy. As we briefed in Chapter 1, there are a few numerical methods presented in the past few years. Most of these methods can be second order accurate in \( L_1 \) or \( L_2 \) norm, but not in \( L_\infty \) norm.

The other concern is how to solve the resulting linear system efficiently. Usually the number of iterations depends on the mesh size. Also, if the jump in the coefficient \( \beta \) is large, then the resulting linear system is ill-conditioned, and thus the number of iterations in solving such a linear system is large and may also be proportional to the jump in the coefficient.
Based on integral equations, some fast solvers are available for Poisson equations with piecewise constant coefficients and other problems [28, 57, 59, 60]. In these methods, integral equations are set up at some points on the interfaces and boundaries for unknown source strength, and the solutions then can be found using fast boundary integral techniques. Non-homogeneous source terms can be decomposed as two homogeneous problems. For example, Mayo and Greenbaum [58, 59] have derived an integral equation for elliptic interface problems with piecewise constant coefficients. By solving the integral equation, they can solve such interface problems to second order accuracy in $L^\infty$ norm using the techniques developed by Mayo in [57, 58] for solving Poisson and biharmonic equations on irregular regions. Basically, the region is embedded in a regular region where a fast solver can be used on a uniform grid and the right-hand side is appropriately modified near the original boundary. The total cost includes solving the integral equation and a regular Poisson equation using a fast solver, so this gives a fast algorithm. The possibility of extension to variable coefficient $\beta$ is mentioned in [58].

In this chapter, we develop a fast algorithm for elliptic equations with piecewise constant but discontinuous coefficients. The idea is to precondition the original elliptic equation before using the immersed interface method. In order to take advantage of existing fast Poisson solvers on cubic domains, we introduce a new problem by rewriting the differential equation and introducing an intermediate unknown jump in $u_n$. The new problem looks like a Poisson equation, which can be discretized by using the standard seven point stencil with some modification in the right-hand side. Then some existing fast Poisson solver can be called directly. Basically, this approach is equivalent to using a second order finite difference scheme to approximate the Poisson equation in $\Omega^-$ and $\Omega^+$, and a second order discretization for the Neumann-like interface condition

$$\beta^+ u^+_n - \beta^- u^-_n = q.$$  

Thus from the error analysis for elliptic equations with Neumann boundary conditions, for example, see [61], we would have second order accurate solution at all grid points including those near or on the interface. The GMRES method is employed to solve the Schur complement system derived from the discretization. This approach appears to be very promising not only because it is second order accurate, but also because the number of iterations in solving the Schur complement system is almost independent of the mesh size.

Below we begin to describe the approach in detail.


Chapter 3. A Fast Algorithm for 3D Elliptic Interface Problems

3.2 The algorithm description

3.2.1 Preconditioning the PDE to an equivalent problem

By direct computation, (3.1) can be rewritten into the following differential equations

\[ \Delta u + \frac{\nabla \beta^+}{\beta^+} \cdot \nabla u = \frac{f}{\beta^+}, \quad \text{in } \Omega^+, \]
\[ \Delta u + \frac{\nabla \beta^-}{\beta^-} \cdot \nabla u = \frac{f}{\beta^-}, \quad \text{in } \Omega^- . \]

(3.5)

In the case of \( \beta \) being piecewise constant, (3.5) becomes

\[ \Delta u = \frac{f}{\beta^+}, \quad \text{in } \Omega^+, \]
\[ \Delta u = \frac{f}{\beta^-}, \quad \text{in } \Omega^- . \]

(3.6)

At this stage we still cannot use fast Poisson solvers since one jump condition (3.4) is given in \( [\beta u_n] \). However, if the jump condition is given in \( [u_n] \) instead of \( [\beta u_n] \), we then can use fast Poisson solvers.

Another observation here is that: let \( u^* \) be the solution of Problem (I). Define

\[ g^* = [u^*_n], \]

then \( u^* \) will also be the solution of (3.5) with the jump condition \( [u] = w \) and \( [u_n] = g^* \).

Therefore, solving Problem (I) is equivalent to finding the corresponding \( g^* \) and then the solution, \( u_{g^*} \), of the following problem with \( g = g^* \).

**Problem (II).**

\[ \Delta u + \frac{\nabla \beta^+}{\beta^+} \cdot \nabla u = \frac{f}{\beta^+}, \quad \text{in } \Omega^+, \]
\[ \Delta u + \frac{\nabla \beta^-}{\beta^-} \cdot \nabla u = \frac{f}{\beta^-}, \quad \text{in } \Omega^- . \]

(3.7)

(3.8)

Given boundary conditions on \( \partial \Omega \),

\[ [u] = w, \quad \text{on } \Gamma, \]
\[ [u_n] = g, \quad \text{on } \Gamma . \]

(3.9)

(3.10)

(3.11)

The key is how to find \( g^* \) efficiently. Basically, we choose an initial guess and then iteratively update it. In summary, the basic steps of the fast algorithm are
Chapter 3. A Fast Algorithm for 3D Elliptic Interface Problems

1. Choose $g_{old}$.

2. Solve Problem (II) with $g = g_{old}$ for $u_{g_{old}}$.

3. Update $g$ to $g_{new}$ based on $u_{g_{old}}$.

4. Let $g_{old} = g_{new}$, goto Step 2.

Notice that $g^*$ is only defined along the interface $\Gamma$, so it is two-dimensional in a three-dimensional space. Problem (II) is much easier to solve because one jump condition is given in $[u_n]$ instead of in $[\beta u_n]$. For elliptic interface problems with general discontinuous coefficients, we can use the maximum principle preserving scheme to solve them. But in this chapter, we are especially interested in the case that $\beta$ is piecewise constant, so the corresponding problem becomes a Poisson equation with a discontinuous source term and given jump conditions. We can then use the standard seven point stencil to discretize the left-hand side of (3.6), but just modify the right-hand side to get a second order finite difference scheme, see [43, 46] for the details. Thus we can take advantage of fast Poisson solvers for the discrete system. The cost in solving Problem (II) then is just a little more than that in solving a regular Poisson equation on the cube.

Here we want to compute $u_{g^*}$ to second order accuracy. We also hope that the total cost in computing $g^*$ and $u_{g^*}$ is less than in computing $u_{g^*}$ through the original problem. The key to success is how to compute $g^*$ efficiently. Now we begin to describe our approach to determine $g^*$. Once $g^*$ is found, we just need one more fast Poisson solver call to get the solution $u^*$.

### 3.2.2 Representation of the interface and jump conditions

Again we use the level set approach to express the interface. For any quantity $p$ defined along the interface $\Gamma$, we represent it at some specified discrete points on the interface. We call these points *Control Points*. Then any information of $p$ is evaluated by its values at these control points, see §3.2.6.

In general, we choose control points based on the following two criteria:

1. We want the process to be *local*. We know that one of the advantages of the level set approach is that the geometric information of the interface are completely determined by the level set function $\phi$. We need to preserve such local properties in determining control points.

2. We do not want to have *clustered* control points to avoid unnecessary large and ill-conditioned system.
Chapter 3. A Fast Algorithm for 3D Elliptic Interface Problems

Obviously, one simple approach is to choose projections of irregular grid points as control points. But it is quite possible that different irregular grid points may share the common projections or some projections may be very close to each other. To avoid this situation, only projections corresponding to the irregular grid points in one side of the interface are chosen as control points. Usually, the number of control points is about half of that of total projections.

Let the control points be \( X_k = (X_k, Y_k, Z_k) \), \( k = 1, 2, \cdots, n_c \), where \( n_c \) is the number of the control points. Then any quantity defined on the interface can be discretized. For example, we denote the discrete vector forms of \( w \), \( q \) and \( g \) by

\[
W = (w_1, w_2, \cdots, w_{n_c})^T, \quad Q = (q_1, q_2, \cdots, q_{n_c})^T, \quad G = (g_1, g_2, \cdots, g_{n_c})^T,
\]

where

\[
w_k \approx w(X_k) = w(X_k, Y_k, Z_k), \quad q_k \approx q(X_k) = q(X_k, Y_k, Z_k), \quad g_k \approx g(X_k) = g(X_k, Y_k, Z_k).
\]

In the next, unless otherwise specified, we use uppercase letters to indicate the discrete vector forms of quantities.

3.2.3 Discretization

As in the maximal principle preserving scheme, we use a uniform Cartesian grid on the cube \([a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]\) where Problem (I) is defined

\[
x_i = a_1 + ih, \quad y_j = a_2 + jh, \quad z_k = a_3 + kh, \quad 0 \leq i \leq l, \quad 0 \leq j \leq m, \quad 0 \leq k \leq n.
\]

Here, for simplicity, again we assume that \( h = (b_1 - a_1)/l = (b_2 - a_2)/m = (b_3 - a_3)/n \). Then with the immersed interface method, it is known that the discrete form of (3.6) can be written as

\[
L_h u_{ijk} = \frac{f_{ijk}}{\beta_{ijk}} + C_{ijk}, \quad 0 \leq i \leq l, \quad 0 \leq j \leq m, \quad 0 \leq k \leq n, \quad (3.12)
\]

where

\[
L_h u_{ijk} = \frac{u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + u_{i,j,k+1} - 6 u_{ijk}}{h^2}, \quad (3.13)
\]

is the discrete Laplace operator using the standard seven point stencil. Note that if a grid point \((x_i, y_j, z_k)\) happens to be on the interface, then \( f_{ijk} \) and \( \beta_{ijk} \) are defined as the
limiting values from a pre-chosen side of the interface. For regular grid points, the correction term $C_{ijk}$ is zero. For irregular grid points, $C_{ijk}$ is computed with the immersed interface method. Then, a fast Poisson solver, for example, a fast Fourier transformation (FFT), or a multigrid solver can be applied to solve (3.12).

3.2.4 Updating the jump in the normal derivative

As presented earlier, the key idea of the fast algorithm is to guess a jump in $u_n$ initially and then update it gradually until we have that the original jump condition in $\beta u_n$ is satisfied. Hence the crucial part of the fast algorithm is how to update the jump in $u_n$ efficiently.

By choosing an initial guess on $[u_n]$, say $g_{old}$, and solving Problem (II) for $u_{g_{old}}$, we actually have $u_{ijk}$, the approximate value of $u$ at grid points $(x_i, y_j, z_k)$, $0 \leq i \leq l$, $0 \leq j \leq m$, $0 \leq k \leq n$. Then based upon these values (defined on grid points, sometimes we call it a grid function), for each control point $X_k$, $k = 1, 2, \cdots, n_c$, we may find the limiting values of $u_n$ from both sides of the interface respectively, say $u^-_n(X_k)$ and $u^+_n(X_k)$, $k = 1, 2, \cdots, n_c$. This can be done by an interpolation scheme, and we will cover this later. Let

$$U^-_n = (u^-_n(X_1), u^-_n(X_2), \cdots, u^-_n(X_{nc}))^T,$$

$$U^+_n = (u^+_n(X_1), u^+_n(X_2), \cdots, u^+_n(X_{nc}))^T.$$  

The ideal situation would be that we already got the solution of Problem (I). In other words, $U^-_n$ and $U^+_n$ already satisfy the discrete form of (3.4)

$$\beta^+ U^+_n - \beta^- U^-_n - Q = 0. \tag{3.14}$$

Unfortunately, usually this is not true for an arbitrary initial guess.

If $U^-_n$ and $U^+_n$ are exact, that is (3.14) is satisfied, then we can solve $[U_n] = U^+_n - U^-_n$ in terms of $Q$, $\beta^-$, $\beta^+$ and one of $U^-_n$ and $U^+_n$. It is straightforward to get

$$[U_n] = \frac{Q - (\beta^+ - \beta^-) U^-_n}{\beta^+},$$

or

$$[U_n] = \frac{Q - (\beta^+ - \beta^-) U^+_n}{\beta^-}.$$  

And these lead the following updating scheme

$$G_{new} = \frac{Q - (\beta^+ - \beta^-) U^-_n}{\beta^+}, \tag{3.15}$$
or
\[ G_{\text{new}} = \frac{Q - (\beta^+ - \beta^-)U_n^+}{\beta^-}. \] (3.16)

We did some numerical experiments by using this updating approach. It turned out that in most cases the iteration converged. However, the convergence rate was too slow, and the number of iterations seemed to be proportional to $1/h$. Below we describe a fast algorithm.

The solution $U$ of Problem (II) depends on $G$ and $W$ continuously. When $W = 0$, $G = 0$, the discrete linear system for Problem (II) is
\[ AU = F, \]
which is the standard discretization of a usual Poisson problem. For non-homogeneous $W$ or $G$, the discrete linear system of Problem (II), in matrix and vector form, is
\[ AU + \Psi(W, G) = F, \] (3.17)
where $\Psi(W, G)$ is a mapping from $W$ and $G$ to $C_{ijk}$’s in (3.12). From Chapter 2, we also know that $\Psi(W, G)$ depends on the first and second derivatives of $w$, and the first derivatives of $g$, where the differentiation is carried out along the interface. At this time we do not know whether such a mapping is linear or not. However in the discrete case, as we will see later, all the derivatives are obtained by interpolating values of $w$ or $g$ on those control points. Therefore, $\Psi(W, G)$ is indeed a linear mapping and can be written as
\[ \Psi(W, G) = BG - B_1 W, \] (3.18)
where $B$ and $B_1$ are two matrices with real entries. So (3.17) becomes
\[ AU + BG = F + B_1 W = \bar{F}, \] (3.19)
where $\bar{F}$ is defined as $F + B_1 W$.

The solution $U$ of the equation above certainly depends on $G$ and we are interested in finding $G^*$ which satisfies the discrete form of (3.4)
\[ \beta^+ U_n^+ (G^*) - \beta^- U_n^- (G^*) - Q \equiv 0. \] (3.20)

Later, we will discuss how to use the known jump $G$, and sometimes also $Q$, to interpolate $U$ to get $U_n^-$ and $U_n^+$ in detail. As we will see, $U_n^-$ and $U_n^+$ depend on $U$, $G$, and $Q$ linearly, which implies
\[ \beta^+ U_n^+ - \beta^- U_n^- - Q = EU + DG + PQ - Q = EU + DG - PQ, \] (3.21)
where $E$, $D$, $ar{P}$, $P$ are some matrices, and $P = I - ar{P}$. Combining (3.19) and (3.21), we obtain the system of linear equations for $U$ and $G$

$$
\begin{bmatrix}
A & B \\
E & D \\
\end{bmatrix}
\begin{bmatrix}
U \\
G \\
\end{bmatrix} =
\begin{bmatrix}
\bar{F} \\
PQ \\
\end{bmatrix}.
$$

(3.22)

Now the question is how to solve (3.22) efficiently. We do not intend to solve it as a whole. Instead, we solve for $G$ and $U$ in turn using the most updated information.

Solving for $U$ is one fast Poisson solver call if $\beta$ is piecewise constant. The question is how to solve for $G$ efficiently. Eliminating $U$ from (3.22) gives us a linear system for $G$

$$(D - EA^{-1}B)G = PQ - EA^{-1}\bar{F} = \bar{Q},$$

(3.23)

where $\bar{Q}$ is defined as $PQ - EA^{-1}\bar{F}$. This is an $n_c \times n_c$ linear system compared to the one for $U$. The coefficient matrix is the Schur complement of $D$ in (3.22). In practice, the matrices $A$, $B$, $E$, $D$, $P$ and the vectors $\bar{Q}$, $\bar{F}$ are never explicitly formed. They are merely used for theoretical purposes. Therefore an iterative method is preferred. Especially, note that the Schur complement is not symmetric, a GMRES iteration is more preferred. Although we may use the Bi-Conjugate Gradient method (BICG), the numerical experiments did show it was not as good as the GMRES method.

Also note that if $\beta$ is continuous, the coefficient matrix of (3.23) is invertible since $E \equiv 0$ and $D = I$.

### 3.2.5 A least square approach for computing an interface quantity from a grid function

When we apply the GMRES method to solve the Schur complement system (3.23), we need to compute $U_n^-$ and $U_n^+$ with the knowledge of $U$. This turns out to be a crucial step in solving the system of linear equations. Below we will describe a least square approach to interpolate $U_n^-$ and $U_n^+$.

Let $u$ be a piecewise smooth function, with discontinuities only along the interface. For a given point $X = (X,Y,Z)$ on the interface, We want to interpolate $u(x_i,y_j,z_k)$, $0 \leq i \leq l$, $0 \leq j \leq m$, $0 \leq k \leq n$, to get the normal derivatives $u_n^-(X)$ and $u_n^+(X)$.

The approach is inspired by Peskin’s method [66] in interpolating a velocity field to get the velocity of the interface using a discrete $\delta$-function. The continuous and discrete forms
are the following
\[
    u(X) = \int\int\int_\Omega u(x, y, z) \delta(X - x) \delta(Y - y) \delta(Z - z) \, dx \, dy \, dz,
\]
(3.24)
\[
    u(X) \approx h^3 \sum_{i,j,k} u_{ijk} \delta_h(X - x_i) \delta_h(Y - y_j) \delta_h(Z - z_k),
\]
(3.25)
where \( X = (X, Y, Z) \) is a point on the interface and \( \delta_h \) is a discrete Dirac \( \delta \)-function. A commonly used one is
\[
    \delta_h(x) = \begin{cases} 
        \frac{1}{4h} (1 + \cos(\pi x/2h)), & \text{if } |x| < 2h, \\
        0, & \text{if } |x| \geq 2h.
    \end{cases}
\]
(3.26)
Notice that \( \delta_h(x) \) is a smooth function of \( x \). Peskin’s approach is very robust and only a few neighboring grid points near \( X \) are involved. However, this approach is only first order accurate and may smear out the solution near the interface.

Our interpolation formula for \( u_n^- \), for example, can be written in the following form
\[
    u_n^-(X) \approx \sum_{(i,j,k) \in N} \gamma_{ijk} u_{ijk} - C,
\]
(3.27)
where \( N \) denotes a set of neighboring grid points near \( X \), and \( C \) is a correction term which can be determined once \( \gamma_{ijk} \)'s are known. Usually, we choose \( N \) starting with those grid points closest to \( X \). Therefore, expression (3.27) is robust and depends on the grid function \( u_{ijk} \) continuously, one very attractive property of Peskin’s formula (3.25). In addition to the advantages of Peskin’s approach, we also have flexibility in choosing the coefficients \( \gamma_{ijk} \)’s and the correction term \( C \) to achieve second order accuracy.

Now we discuss how to use the IIM method to determine the coefficients \( \gamma_{ijk} \)'s and the correction term \( C \). They are different from point to point on the interface. So they should really be labeled as \( \gamma_{ijk,X} \), etc. But for simplicity of notation we will concentrate on a single point \( X \) and drop the subscript \( X \).

We use the same idea as used in the IIM method. Since one jump condition is given in the normal derivative of the solution, we use the local coordinates at \( X = (X, Y, Z) \)
\[
    \begin{pmatrix} 
        \xi \\
        \eta \\
        \tau
    \end{pmatrix} = A \begin{pmatrix} 
        x - X \\
        y - Y \\
        z - Z
    \end{pmatrix},
\]
(3.28)
where \( A \) is defined in (2.10). Recall that under such new coordinates, the interface can be parameterized by
\[
    \xi = \chi(\eta, \tau) \quad \text{with} \quad \chi(0, 0) = 0, \quad \chi_\eta(0, 0) = 0, \quad \chi_\tau(0, 0) = 0,
\]
(3.29)
provided the interface is smooth at $X = (X,Y,Z)$.

It is easy to check that when $\beta$ is piecewise constant, the interface relations (2.29) and (2.33) for Problem (II) reduce to

$$
\begin{align*}
&\begin{cases}
  u^+ = u^− + w, \\
  u^+_{ξ} = u^−_{ξ} + g, \\
  u^+_{η} = u^−_{η} + w_{η}, \\
  u^+_{τ} = u^−_{τ} + w_{τ}, \\
  u^+_{ητ} = u^−_{ητ} - g\chi_{ητ} + w_{ητ}, \\
  u^+_{ηη} = u^−_{ηη} - g\chi_{ηη} + w_{ηη}, \\
  u^+_{ττ} = u^−_{ττ} - g\chi_{ττ} + w_{ττ}, \\
  u^+_{ξη} = u^−_{ξη} + w_{η}\chi_{η} + w_{τ}\chi_{ητ} + g_{η}, \\
  u^+_{ξτ} = u^−_{ξτ} + w_{η}\chi_{τ} + w_{τ}\chi_{ττ} + g_{τ}, \\
  u^+_{ξξ} = u^−_{ξξ} + g(\chi_{ηη} + \chi_{ττ}) + \left[\frac{f}{\beta}\right] - w_{ηη} - w_{ττ}.
\end{cases}
\end{align*}
$$

(3.30)

Let $(ξ_i, η_j, τ_k)$ be the $ξ$-$η$-$τ$ coordinates of $(x_i, y_j, z_k)$, then by Taylor series expansion as we did in Chapter 2, we have

$$
\begin{align*}
  u(ξ_i, η_j, τ_k) &\approx u^+ + u^+_{ξ}ξ_i + u^+_{η}η_j + u^+_{τ}τ_k + \frac{1}{2}u^+_{ξξ}ξ_i^2 + \frac{1}{2}u^+_{ηη}η_j^2 + \frac{1}{2}u^+_{ττ}τ_k^2 \\
  &+ u^+_{ξη}ξ_iη_j + u^+_{ξτ}ξ_iτ_k + u^+_{ητ}η_jτ_k,
\end{align*}
$$

(3.31)

where + and − sign depends on whether $(ξ_i, η_j, τ_k)$ lies in the + or − side of the interface $Γ$. Expressing + values by − values and collecting like terms, we have

$$
\begin{align*}
  u_n(X) &\approx a_1u^− + a_2u^+ + a_3u^−_ξ + a_4u^+_ξ + a_5u^−_η + a_6u^+_η + a_7u^−_τ + a_8u^+_τ \\
  &+ a_9u^−_{ξξ} + a_{10}u^+_ξξ + a_{11}u^−_ηη + a_{12}u^+_ηη + a_{13}u^−_ττ + a_{14}u^+_ττ \\
  &+ a_{15}u^−_{ξη} + a_{16}u^+_ξη + a_{17}u^−_ξτ + a_{18}u^+_ξτ + a_{19}u^−_ητ + a_{20}u^+_ητ \\
  &- C + O(h^3 \max |γ_{ijk}|),
\end{align*}
$$

(3.32)

where the coefficients $a_k$’s are given by (2.36).
After using the interface relations in (3.30), we have
\[
  u_n^-(X) \approx (a_1 + a_2)u_\xi^- + (a_3 + a_4)u_\eta^- + (a_5 + a_6)u_\tau^- + (a_7 + a_8)u_\xi^- \\
  + (a_9 + a_{10})u_{\xi \xi}^- + (a_{11} + a_{12})u_{\eta \eta}^- + (a_{13} + a_{14})u_{\tau \tau}^- + (a_{15} + a_{16})u_{\xi \eta}^- \\
  + (a_{17} + a_{18})u_{\xi \tau}^- + (a_{19} + a_{20})u_{\eta \tau}^- + a_2[u] + a_4[u_\xi] + a_6[u_\eta] + a_8[u_\tau] \\
  + a_{10}[u_{\xi \xi}] + a_{12}[u_{\eta \eta}] + a_{14}[u_{\tau \tau}] + a_{16}[u_{\eta \xi}] + a_{18}[u_{\xi \tau}] + a_{20}[u_{\eta \tau}] \\
  - C + O(h^3 \max |\gamma_{ijk}|). \tag{3.33}
\]

On the other hand, we know \( u_n^- = u_\xi^- \). Therefore, we have the system of linear equations for \( \gamma_{ijk} \)'s
\[
\begin{align*}
  a_1 + a_2 &= 0 \\
  a_3 + a_4 &= 1 \\
  a_5 + a_6 &= 0 \\
  a_7 + a_8 &= 0 \\
  a_9 + a_{10} &= 0 \\
  a_{11} + a_{12} &= 0 \\
  a_{13} + a_{14} &= 0 \\
  a_{15} + a_{16} &= 0 \\
  a_{17} + a_{18} &= 0 \\
  a_{19} + a_{20} &= 0.
\end{align*}
\tag{3.34}
\]

If the system of linear equations (3.34) has a solution, then we can obtain a second order approximation to the normal derivative \( u_n^-(X) \) by choosing an appropriate correction term \( C \). The above linear system has ten equations. So the set of neighboring grid points \( N \) should be large enough such that at least 10 grid points are included. Usually we take more than 10 grid points and the above linear system becomes an underdetermined system which has an infinite number of solutions. We should choose the solution \( \gamma_{ijk}^* \) with the minimum 2-norm
\[
\sum_{(i,j,k) \in N} (\gamma_{ijk}^*)^2 = \min \sum_{(i,j,k) \in N} (\gamma_{ijk})^2.
\]
For such a solution, each \( \gamma_{ijk}^* \) has roughly the same magnitude \( O(1/h) \).

When we get the coefficients \( \gamma_{ijk} \)'s, we can compute the \( a_k \)'s. From the \( a_k \)'s and (3.33),
we can determine the correction term \( C \) easily

\[
C = a_2[u] + a_4[u_2] + a_6[u_\eta] + a_8[u_\tau] + a_{10}[u_\eta\eta] + a_{12}[u_\eta\tau] + a_{14}[u_\eta\eta] + a_{16}[u_\eta\tau] + a_{20}[u_\eta\tau]
\]

\[
= a_2w + a_4g + a_6w_\eta + a_8w_\tau + a_{10}(g(\chi_\eta\eta + \chi_\tau\tau) + \left[ \frac{f}{\beta} \right] - w_\eta\eta - w_\tau\tau) + a_{12}(w_\eta\eta - g\chi_\eta\eta) + a_{14}(w_\tau\tau - g\chi_\tau\tau) + a_{16}(w_\eta\chi_\eta\eta + w_\tau\chi_\eta\tau + g_\eta) + a_{18}(w_\eta\chi_\eta\tau + w_\tau\chi_\tau\tau + g_\tau) + a_{20}(w_\eta\tau - g_\chi_\tau\eta).
\]

(3.35)

Therefore we are able to compute \( u_n^- (X) \) to second order accuracy. Similarly we can derive a formula for \( u_n^+ (X) \) in exactly the same way, i.e., we may use the following interpolation formula

\[
u_n^+(X) \approx \sum_{(i,j,k) \in N} \tilde{\gamma}_{ijk} u_{ijk} - \tilde{C}.
\]

(3.36)

However, with the jump condition \( u_n^+(X) = u_n^-(X) + g(X) \), we can write down a second order interpolation scheme for \( u_n^+(X) \) immediately

\[
u_n^+(X) \approx \sum_{(i,j,k) \in N} \gamma_{ijk} u_{ijk} - C + g(X),
\]

(3.37)

where \( \gamma_{ijk} \)'s is the solution we computed for \( u_n^-(X) \).

The above least square technique has several nice properties. First of all, it has second order accuracy with local support. The grid points involved are clustered around the point \( X \) on the interface. Secondly, it is robust. The interpolation formulas (3.27) and (3.37) depend continuously on the location of the point \( X \) and the grid points involved, and so does the truncation error for these two interpolation schemes. In other words, we have a smooth error distribution. This is very important for moving interface problems where we do not want to introduce any non-physical oscillations.

Of course, there is a trade-off. We have to solve an underdetermined \( 10 \times n_{u_n} \) linear system of equations, where \( n_{u_n} \) denotes the number of grid points involved in the interpolation scheme (3.27) and (3.37) ( \( n_{u_n} \geq 10 \)).

With the algorithm described in the above, we are able to solve Problem (I) to second order accuracy. In each iteration we need to solve a Poisson equation with a modified right-hand side. A fast Poisson solver using the FFT method, the cyclic reduction, etc, can then be used. Also we need to solve a Schur complement system. The GMRES method can be used and the number of iterations depends on the condition number of the Schur complement system, If we make use of both (3.27) and (3.37) to compute \( u_n^- (X) \) and \( u_n^+ (X) \), the condition number seems to be proportional to \( 1/h \). Therefore, the number of iterations will grow linearly as we increase the number of grid points. This is what we do not want to see in a fast algorithm.
A simple modification in the way of computing $U_n^-$ and $U_n^+$ seems to improve the condition number of the Schur complement system. The idea is simple. We have the jump condition $[\beta u_n] = q$, which implies that if $U_n^-$ and $U_n^+$ are exact, then

$$\beta^+ U_n^+ - \beta^- U_n^- = Q.$$ 

We can solve for $U_n^-$ or $U_n^+$ in terms of $Q$, $\beta^-$, $\beta^+$ and $[U_n]$ to have

$$U_n^- = \frac{Q - \beta^+[U_n]}{\beta^+ - \beta^-}, \tag{3.38}$$

or

$$U_n^+ = \frac{Q - \beta^-[U_n]}{\beta^+ - \beta^-}. \tag{3.39}$$

If we independently compute $U_n^-$ and $U_n^+$ from (3.27) and (3.37) respectively, due to errors, usually they may not satisfy the flux jump condition. Therefore, in practice we use one of the formulas (3.27) and (3.37) to approximate $U_n^-$ or $U_n^+$, and then use (3.39) or (3.38) to approximate $U_n^+$ or $U_n^-$ to force the solution to satisfy the flux jump condition. This is an acceleration process or a preconditioner for the Schur complement system.

With this modification, the number of iterations for solving the Schur complement system seems to be independent of the mesh size. However, it seems to be dependent of the jump in the coefficient. This is different from two-dimensional cases.

Whether we use the pair (3.27) and (3.39) or the other, (3.37) and (3.38), has only a little effect on accuracy of the computed solutions and the number of iterations. In our numerical experiment, we have been using the following criteria to choose the desired pair

$$\begin{cases} 
\text{Interpolation for } U_n^+ \text{ by (3.37),} \\
U_n^- = \frac{Q - \beta^+ G}{\beta^+ - \beta^-} 
\end{cases}$$

$$\begin{cases} 
\text{Interpolation for } U_n^- \text{ by (3.27),} \\
U_n^+ = \frac{Q - \beta^- G}{\beta^+ - \beta^-} 
\end{cases}$$

3.2.6 A least square approach for computing an interface quantity from its values at control points

As pointed out before, to avoid unnecessary large and ill-conditioned Schur complement system, not all projection points are chosen as control points. Therefore, when we solve
the Schur complement system, we only have values of the updated jump at those control points. However, to apply the immersed interface method to Problem (II), we need values of the jump and/or derivatives of the jump at all projection points. Basically, this leads us to ask, given discrete values \( p_k, \ k = 1, 2, \ldots, n_c \) of a quantity \( p \) at control points \( X_k, \ k = 1, 2, \ldots, n_c \), how to find its value or derivatives at a projection point \( X = (X, Y, Z) \) on the interface? Again a least square approach can be used, say,

\[
p(X) = \sum_{k=1}^{n_p} \alpha_k p_k,
\]

\[
p_\xi(X) = \sum_{k=1}^{n_p} \lambda_k p_k,
\]

\[
p_\eta(X) = \sum_{k=1}^{n_p} \sigma_k p_k,
\]

\[
p_\tau(X) = \sum_{k=1}^{n_p} \mu_k p_k,
\]

where the summation is over a set of \( n_p \) neighboring control points near the point \( X \). Also the coefficients \( \alpha_k \)'s, \( \lambda_k \)'s, \( \sigma_k \)'s and \( \mu_k \)'s can be found in the same way as discussed in §2.2.8.

Throughout our numerical experiments, we found that the choice of \( n_p \) was crucial to the convergence of the GMRES method.

### 3.2.7 Invertibility of the Schur complement system

As mentioned earlier, if \( \beta \) is continuous, the coefficient matrix of (3.23) is invertible since \( E \equiv 0 \) and \( D = I \). For general cases, we can show that the coefficient matrix \( D - EA^{-1}B \) is also invertible if \( h \) is small enough.

We know the system of linear equations for the jump in the normal derivative \( G^* \) is implicitly defined in the discrete form of the flux jump condition

\[
\beta^+ U_n^+ - \beta^- U_n^- - Q = 0.
\]

With the least square interpolation (3.27) and (3.37) described earlier, the component of the equation above at a control point is approximated by

\[
(\beta^+ - \beta^-) \sum_{(i,j,k) \in N} \gamma_{ijk} u_{ijk} + \left( \beta^+ - (\beta^+ - \beta^-)(a_4 + a_{10}(\chi_{\eta\eta} + \chi_{\tau\tau})) \right) g + a_{16} g_\eta + a_{18} g_\tau - q - (\beta^+ - \beta^-)\bar{C} = 0,
\]

where the summation is over a set of \( n_p \) neighboring control points near the point \( X \). Also the coefficients \( \alpha_k \)'s, \( \lambda_k \)'s, \( \sigma_k \)'s and \( \mu_k \)'s can be found in the same way as discussed in §2.2.8.
where

\[ \bar{C} = a_2 w + a_6 w_\eta + a_8 w_\tau + a_{10} \left( \frac{f}{\beta} - w_\eta - w_\tau \right) + a_{12} w_\eta \eta + a_{14} (w_\tau g \chi_\tau + w_\tau \chi_\eta) + a_{16} (w_\eta \chi_\eta + w_\tau \chi_\tau) + a_{18} (w_\eta \chi_\eta + w_\tau \chi_\tau) + a_{20} w_\eta \tau. \]  

(3.46)

In vector form, it is the second equation in (3.22)

\[ EU + DG = PQ. \]  

(3.47)

If \( \beta^+ = \beta^- \), then we have the unique solution for \( G \), \( G = Q/\beta^+ \). Assuming now \( \beta^+ \neq \beta^- \), we prove the following theorem on the invertibility of the Schur complement system (3.23).

**Theorem 3.1** Assume that we use the least square interpolation formula (3.27) to compute \( u_n^- \), and the equation (3.37) to compute \( u_n^+ \). If \( h \) is small enough, then \( D - EA^{-1}B \) is invertible.

**Proof.** It is enough to consider the homogeneous case

\[ w = 0, \; f = 0, \; q = 0, \; \text{and} \; u = 0, \; \text{on} \; \partial \Omega. \]

In this case, \( \bar{Q} = -EA^{-1}\bar{F} = 0 \). If the theorem is not true, then there is a \( G^* \neq 0 \) such that \((D - EA^{-1}B)G^* = 0\). Let \( U^* = -A^{-1}BG^* \), which is the discrete solution of Problem (II) with \( [u_n] = g^* \), a continuous extension of \( G^* \) along the interface. Since \((D - EA^{-1}B)G^* \) is second order approximation of \( \beta^+ u_n^+ - \beta^- u_n^- \), we conclude \( [\beta u^*] = 0 \) when \( h \) is small enough. So \( u^* \) is also the solution of Problem (I). However \( u \equiv 0 \) is an obvious solution of Problem (I). By the uniqueness of the solution of the interface problem, we must have \( u^* \equiv 0 \) and \( g^* \equiv 0 \). This contradicts the assumption that \( G^* = 0 \). \( \square \)

### 3.2.8 Some implementation details

The main process of the fast algorithm is to solve the Schur complement system (3.23) using the GMRES method with an initial guess

\[ G^{(0)} = (g_1^{(0)}, g_2^{(0)}, \ldots, g_{n_c}^{(0)}). \]

We need to derive the right-hand side, and explain how to compute the matrix-vector multiplication of the system without explicitly forming the coefficient matrix. The right-hand side needs to be computed just once which is described below.

We apply one step of the immersed interface method with \( G^{(0)} = 0 \) to solve Problem (II) to get \( U(0) \), then

\[ U(0) = A^{-1}\bar{F}. \]
With the knowledge of \( U(0) \) and \( G = 0 \), we can compute the normal derivatives on each side of the interface to get \( U_n^-(0) \) and \( U_n^+(0) \) using the least square interpolation formulas (3.27) and (3.37). Thus the right-hand side of the Schur complement system is

\[
\bar{Q} = PQ - EA^{-1}F \\
= PQ - EU(0) \\
= -(\beta^+ U_n^+(0) - \beta^- U_n^-(0) - Q),
\]

so we have

\[
\bar{Q} = Q + \beta^- U_n^- (0) - \beta^+ U_n^+ (0).
\] (3.48)

Also we know that the main computation in each GMRES iteration is the matrix-vector multiplication 

\[
(D - EA^{-1}B)T
\]

of the Schur complement, where \( T \) is a vector of dimension \( n_c \). This involves essentially two steps:

1. A fast Poisson solver for computing

\[
U(T) = A^{-1}(F - BT),
\] (3.49)

which is the solution of Problem (II) with \( G = T \).

2. The least square interpolation to compute \( U_n^-(T) \) and \( U_n^+(T) \).

The residual vector in the flux jump condition is

\[
R(T) = Q - (\beta^+ U_n^+(T) - \beta^- U_n^-(T)),
\] (3.50)

which is the same residual vector of the second equation in (3.22) from our definition. In other words (see also (3.21))

\[
PQ - (DT + EU(T)) = R(T).
\]

Then

\[
(D - EA^{-1}B)T = DT - EA^{-1}BQ \\
= DT + EU(T) - EA^{-1}\bar{F} \\
= DT + EU(T) - PQ + \bar{Q}.
\]

Therefore, finally we have

\[
(D - EA^{-1}B)T = -R(T) + \bar{Q}.
\] (3.51)
Finally, it is worthy to point out that once the algorithm is successfully terminated, which means that the residual vector is close to the zero vector, we not only have an approximation $T$ to $G^*$, an approximation $U(T)$ to the solution $U$, but also approximations $U_n^\pm(T)$ to the normal derivatives from each side of the interface. The normal derivative information is very useful for some moving interface problems, where the velocity of the interface depends on the normal derivative of the solution.

### 3.2.9 Summary of the fast algorithm

To summarize, we give the step-by-step fast algorithm in the following.

1. **Preparation**
   
   (a) For each irregular grid point, find its projection on the interface, see §2.2.7.
   (b) Choose control points as the projection points from a particular side of the interface.

2. Choose an initial guess on $G$ (usually 0), and use a least square interpolation to extend $G$ to each projection point.

3. Apply the immersed interface method to solve Problem (II) to get $U$.

4. Use least square interpolations to find $U_n^+$ and $U_n^-$.

5. Compute $\bar{Q}$ by (see (3.48))
   
   $$\bar{Q} = Q + \beta^{-}U_n^- - \beta^+U_n^+.$$

6. Check the convergence: if $\|\bar{Q}\| < \varepsilon$, stop. Otherwise, continue.

7. Solve the Schur complement system (3.23) to update $G$, in which following the next four steps to find the matrix-vector multiplication: $(D - EA^{-1}B)T$:
   
   (a) Apply the immersed interface method with $G = T$ to solve Problem (II) to get $U(T)$.
   (b) Use least square interpolations to find $U_n^+(T)$ and $U_n^-(T)$.
   (c) Compute the residual vector by (see (3.50))
   
   $$R(T) = Q - (\beta^+U_n^+(T) - \beta^-U_n^-(T)).$$
   
   (d) Find the matrix-vector multiplication by (see (3.51))
   
   $$(D - EA^{-1}B)T = -R(T) + \bar{Q}.$$  

8. Use a least square interpolation to extend $G$ to each projection point, and go back to Step 3.
Chapter 3. A Fast Algorithm for 3D Elliptic Interface Problems

3.3 Numerical results

We have done a number of numerical experiments which confirm the expected order of accuracy for the fast algorithm. The computation are done using Sun ULTRA 10 workstations. The initial guess for $g$ is always chosen as 0, $n_p$ as 50 and $n_u$ as 50. The computational domain is $[-1,1] \times [-1,1] \times [-1,1]$ unless otherwise specified. We also used $l = m = n$ in all computations.

We used the program \texttt{hw3crt.f} (Fishpack) as the 3D fast Poisson solver, and the program \texttt{ssvdc.f} (Linpac) to perform the singular value decomposition (SVD) which is then used to solve the underdetermined linear system. The present version of \texttt{hw3crt.f} solves the standard seven point finite difference approximation to the Helmholtz equation $\Delta u + \lambda u = f$ in Cartesian coordinates.

\textbf{Example 3.1} This example is the same as Example 2.1.

Figure 3.1 shows a slice of the computed solution. It looks very much like Figure 2.2. We see that the fast algorithm does accurately give the jumps in the solution and in the normal derivative of the solution, without smearing out the solution.

Table 3.1 shows the results of a grid refinement analysis, where $\| E_n \|_\infty$ and ratio are defined by (2.74) and (2.75) respectively.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{slice.png}
\caption{A slice of the computed solution ($\beta^+ = 2$, $\beta^- = 1$ and $n = 52$) for Example 3.1: $-u(x, y, 0)$.}
\end{figure}
Table 3.1: The grid refinement analysis for Example 3.1.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$| E_n |_\infty$ ratio</th>
<th>$| E_n |_\infty$ ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>$2.099 \times 10^{-3}$</td>
<td>$4.628 \times 10^{-3}$</td>
</tr>
<tr>
<td>52</td>
<td>$5.343 \times 10^{-4}$</td>
<td>$3.929 \times 10^{-4}$</td>
</tr>
<tr>
<td>104</td>
<td>$1.416 \times 10^{-4}$</td>
<td>$3.175 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

Example 3.2 In this example we consider a problem similar to Example 2.2. The interface is a sphere $x^2 + y^2 + z^2 = 1/4$. The differential equation is

$$(\beta u_x)_x + (\beta u_y)_y + (\beta u_z)_z = f,$$

with

$$\beta(x, y, z) = \begin{cases} 
\beta^-, & \text{if } r < \frac{1}{2}, \\
\beta^+, & \text{if } r \geq \frac{1}{2}, 
\end{cases}$$

$$f(x, y, z) = \begin{cases} 
6, & \text{if } r < \frac{1}{2}, \\
20r^2 + \frac{\log e}{r^2}, & \text{if } r \geq \frac{1}{2}, 
\end{cases}$$

Dirichlet boundary conditions and the jump conditions (3.3) and (3.4) are determined from the exact solution

$$u(x, y, z) = \begin{cases} 
\frac{r^2}{\beta^-}, & \text{if } r < \frac{1}{2}, \\
\frac{r^4 + \log(2r)}{\beta^+} + \frac{(\frac{1}{2})^2}{\beta^-} - \frac{(\frac{1}{2})^4}{\beta^-}, & \text{if } r \geq \frac{1}{2}, 
\end{cases}$$

(3.52)

i.e.

$$[u] = 0,$$

$$[\beta u_n] = 4r^3 + \frac{\log e}{r_0} - 2r_0,$$

where $r = \sqrt{x^2 + y^2 + z^2}$, and on $\Gamma$, $r = r_0 = 1/2$. Note that the solution is continuous in this example, but the normal derivative is not.

We tested three different cases, no jump, small jump and large jump in $\beta$. Figure 3.2 shows a slice of the computed solution. Table 3.2 shows the results of a grid refinement analysis. An average ratio of 4 confirms the second order accuracy.
Figure 3.2: A slice of the computed solution (n = 52) for Example 3.2: −u(x, y, 0). (a) \( \beta^+ = 1, \beta^- = 1 \). (b) \( \beta^+ = 10, \beta^- = 1 \). (c) \( \beta^+ = 1000, \beta^- = 1 \).

Table 3.2: The grid refinement analysis for Example 3.2.

<table>
<thead>
<tr>
<th>n</th>
<th>( \beta^+ = 1 )</th>
<th>ratio</th>
<th>( \beta^+ = 10 )</th>
<th>ratio</th>
<th>( \beta^+ = 1000 )</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>( 3.931 \times 10^{-4} )</td>
<td></td>
<td>( 6.635 \times 10^{-4} )</td>
<td></td>
<td>( 3.598 \times 10^{-5} )</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>( 9.732 \times 10^{-5} )</td>
<td>4.039</td>
<td>( 1.816 \times 10^{-4} )</td>
<td>3.654</td>
<td>( 9.787 \times 10^{-6} )</td>
<td>3.676</td>
</tr>
<tr>
<td>104</td>
<td>( 2.351 \times 10^{-5} )</td>
<td>4.140</td>
<td>( 4.198 \times 10^{-5} )</td>
<td>4.326</td>
<td>( 2.266 \times 10^{-6} )</td>
<td>4.319</td>
</tr>
</tbody>
</table>
Example 3.3 This example is the same as Example 2.3.

Figure 3.3 shows a slice of the computed solution, and Table 3.3 shows the results of a grid refinement analysis.

![Figure 3.3: A slice of the computed solution](image)

**Table 3.3:** The grid refinement analysis for Example 3.3.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \beta^+ = 2, \beta^- = 1 ) ( | E_n |_{\infty} ) ( \text{ratio} )</th>
<th>( \beta^+ = 1000, \beta^- = 1 ) ( | E_n |_{\infty} ) ( \text{ratio} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>( 7.977 \times 10^{-3} )</td>
<td>( 1.972 \times 10^{-3} )</td>
</tr>
<tr>
<td>52</td>
<td>( 3.683 \times 10^{-3} )</td>
<td>( 8.233 \times 10^{-4} )</td>
</tr>
<tr>
<td>104</td>
<td>( 5.741 \times 10^{-4} )</td>
<td>( 2.122 \times 10^{-4} )</td>
</tr>
</tbody>
</table>
Example 3.4 This example is the same as Example 2.4.

Figure 3.4 shows a slice of the computed solution, and Table 3.4 shows the results of a grid refinement analysis.

Table 3.4: The grid refinement analysis for Example 3.4.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\beta^+ = 1, \beta^- = 1$</th>
<th>$\beta^+ = 1000, \beta^- = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$|E_n|_\infty$ ratio</td>
<td>$|E_n|_\infty$ ratio</td>
</tr>
<tr>
<td>52</td>
<td>$3.554 \times 10^{-2}$</td>
<td>$4.258 \times 10^{-2}$</td>
</tr>
<tr>
<td>104</td>
<td>$6.857 \times 10^{-3}$</td>
<td>$1.099 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

Example 3.5 In this example we reconsider Example 3.3 with an irregular interface. The level set function for the interface is defined as the distance function to the interface shown in Figure 3.5. See Appendix A for details.

We tested two different cases, a small jump case with $\beta^+ = 2, \beta^- = 1$ and a big jump case with $\beta^+ = 1000, \beta^- = 1$. Figure 3.6 shows a slice of the computed solution for the big jump case, and Table 3.5 shows the results of a grid refinement analysis.
Chapter 3. A Fast Algorithm for 3D Elliptic Interface Problems

Figure 3.5: The interface for Example 3.5. (a) The interface. (b) A slice of the interface.

Figure 3.6: A slice of the computed solution ( $\beta^+ = 2$, $\beta^- = 1$ and $n = 81$ ) for Example 3.5: $u(x, y, 0)$. 
Table 3.5: The grid refinement analysis for Example 3.5.

<table>
<thead>
<tr>
<th>n</th>
<th>|E_n|_\infty \text{ ratio}</th>
<th>|E_n|_\infty \text{ ratio}</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>5.088 \times 10^{-2}</td>
<td>9.659 \times 10^{-2}</td>
</tr>
<tr>
<td>39</td>
<td>1.401 \times 10^{-2}</td>
<td>2.707 \times 10^{-2}</td>
</tr>
<tr>
<td>81</td>
<td>2.652 \times 10^{-4}</td>
<td>6.560 \times 10^{-4}</td>
</tr>
</tbody>
</table>

3.3.1 The algorithm efficiency analysis

From the numerical tests we have already seen that the fast algorithm is second order accurate and can deal with complicated interfaces and large jumps in the coefficient. We also want to know whether the number of iterations is dependent on the mesh size or the jump in the coefficient \( \beta \).

Table 3.6–Table 3.10 list some statistics for the above examples, where \( n_{irreg} \) has the same meaning as before, \( n_{constr} \) denotes the number of control points and \( n_{iter} \) the number of iteration, i.e., the number of calls to the fast Poisson solver. It can be seen clearly that for all cases, the number of iterations is almost independent of the mesh size. If there is no jump in \( \beta \), the GMRES method converges in one step.

However, the number of iterations appears to depend on the jump in the coefficient \( \beta \). This is different from two-dimensional cases.


**Table 3.6:** CPU time and the number of iterations for Example 3.1.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_{\text{irreg}} )</th>
<th>( n_{\text{contr}} )</th>
<th>( \beta^+ = 2, \beta^- = 1 )</th>
<th>( \beta^+ = 1000, \beta^- = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>920</td>
<td>506</td>
<td>66.092</td>
<td>9</td>
</tr>
<tr>
<td>52</td>
<td>3528</td>
<td>1828</td>
<td>274.930</td>
<td>9</td>
</tr>
<tr>
<td>104</td>
<td>14048</td>
<td>7180</td>
<td>1255.102</td>
<td>9</td>
</tr>
</tbody>
</table>

**Table 3.7:** CPU time and the number of iterations for Example 3.2.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_{\text{irreg}} )</th>
<th>( n_{\text{contr}} )</th>
<th>( \beta^+ = 1, \beta^- = 1 )</th>
<th>( \beta^+ = 10, \beta^- = 1 )</th>
<th>( \beta^+ = 1000, \beta^- = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>920</td>
<td>506</td>
<td>52.127</td>
<td>3</td>
<td>75.602</td>
</tr>
<tr>
<td>52</td>
<td>3528</td>
<td>1828</td>
<td>210.789</td>
<td>3</td>
<td>318.671</td>
</tr>
<tr>
<td>104</td>
<td>14048</td>
<td>7180</td>
<td>917.517</td>
<td>3</td>
<td>1536.733</td>
</tr>
</tbody>
</table>

**Table 3.8:** CPU time and the number of iterations for Example 3.3.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_{\text{irreg}} )</th>
<th>( n_{\text{contr}} )</th>
<th>( \beta^+ = 2, \beta^- = 1 )</th>
<th>( \beta^+ = 1000, \beta^- = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>480</td>
<td>274</td>
<td>36.678</td>
<td>9</td>
</tr>
<tr>
<td>52</td>
<td>1936</td>
<td>1034</td>
<td>167.380</td>
<td>10</td>
</tr>
<tr>
<td>104</td>
<td>7704</td>
<td>3981</td>
<td>790.391</td>
<td>9</td>
</tr>
</tbody>
</table>

**Table 3.9:** CPU time and the number of iterations for Example 3.4.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_{\text{irreg}} )</th>
<th>( n_{\text{contr}} )</th>
<th>( \beta^+ = 1, \beta^- = 1 )</th>
<th>( \beta^+ = 1000, \beta^- = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>220</td>
<td>140</td>
<td>24.912</td>
<td>3</td>
</tr>
<tr>
<td>104</td>
<td>874</td>
<td>499</td>
<td>167.106</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 3.10:** CPU time and the number of iterations for Example 3.5.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_{\text{irreg}} )</th>
<th>( n_{\text{contr}} )</th>
<th>( \beta^+ = 1, \beta^- = 1 )</th>
<th>( \beta^+ = 1000, \beta^- = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>368</td>
<td>216</td>
<td>23.927</td>
<td>14</td>
</tr>
<tr>
<td>39</td>
<td>1832</td>
<td>984</td>
<td>124.642</td>
<td>10</td>
</tr>
<tr>
<td>81</td>
<td>8432</td>
<td>4352</td>
<td>689.634</td>
<td>12</td>
</tr>
</tbody>
</table>
Chapter 4

Applications

4.1 Solving Poisson equations on irregular domains using the fast algorithm

Sometimes a problem on an irregular domain can be handled more easily as an interface problem by embedding the domain into a cubic domain and then solving the equation on a Cartesian grid in the cube. The original boundary then becomes an interface. As an example, suppose we want to solve a three-dimensional elliptic equation on an irregular domain \( \Omega \). We can embed the domain in a larger cubic domain \( R \). For example, we could solve the following Laplace equation with a Dirichlet boundary condition

\[
\begin{align*}
  u_{xx} + u_{yy} + u_{zz} &= 0, & \text{in } \Omega, \\
  u &= v, & \text{on } \partial \Omega,
\end{align*}
\]

by extending it to the problem

\[
\begin{align*}
  u_{xx} + u_{yy} + u_{zz} &= \int \int_{\partial \Omega} F(X) \delta(x - X) \delta(y - Y) \delta(z - Z) \, dS, & \text{in } R, \\
  u &= 0, & \text{on } \partial R.
\end{align*}
\]

The problem is then to determine \( F(X) \) so that the boundary condition \( u = v \) on \( \partial \Omega \) is satisfied. The solution is still continuous on the enlarged region \( R \), but not smooth across the interface \( \partial \Omega \).

This particular problem has been extensively studied in the past and a number of domain embedding procedures have been developed, e.g., capacitance methods [10, 18, 57, 68] and methods based on solving integral equations along \( \partial \Omega \).

With the idea of the fast algorithm, we can also develop an embedding technique to solve elliptic equations on complicated regions with Dirichlet or Neumann boundary conditions. In this section, we show how the fast algorithm can be used to solve interior or exterior Poisson equations on irregular domains.
First, consider an interior Poisson equation with a Dirichlet boundary condition

$$\begin{align*}
\Delta u &= f, \quad \text{in } \Omega, \\
u &= v, \quad \text{on } \partial\Omega,
\end{align*}$$

(4.2)

where $\Omega$ is an arbitrary closed region in the three-dimensional space, see Figure 4.1.

By embedding the region into a cube, we may treat the original problem as an interface problem and use the immersed interface method to solve it. However, recall that to make the resulted interface problem well-posed, usually we need to know not only the jump in the solution $[u]$, but also the jump in the normal derivative of the solution $[u_n]$, across the boundary (now it becomes the interface). We may simply use the Dirichlet boundary condition as the jump condition in the solution, but unfortunately there seems no way to know the exact jump in the normal derivative of the solution.

Based on the same idea as used in the fast algorithm, we can solve the above problem by choosing an initial guess on $[u_n]$, and then updating it until the original boundary condition is satisfied. Below we begin to describe this approach in more detail.

We extend the source term in the Poisson equation by zero outside $\Omega$. On the irregular boundary $\partial\Omega$, we allow a finite jump $[u]$ in the solution itself. One practical choice is just to use the original boundary condition $v$ as $[u]$, and let $u = 0$ on the boundary $\partial R$ of the
cube \( R \). As for the jump in the normal derivative of the solution \([u_n]\), we may use an initial guess, say, \([u_n] = g\) (usually 0). This extension leads to the following interface problem

\[
\Delta u = \begin{cases} 
  f, & \text{if } (x, y, z) \in \Omega, \\
  0, & \text{if } (x, y, z) \notin \Omega,
\end{cases}
\]

\[
[u] = v, \quad \text{on } \partial \Omega,
\]

\[
[u_n] = g, \quad \text{on } \partial \Omega,
\]

\[
u = 0, \quad \text{on } \partial R,
\]

(4.3)

We then use the GMRES iteration to update \( g \) until the original boundary condition is satisfied, i.e.,

\[
u^ = v, \quad \text{on } \partial \Omega,
\]

(4.4)

where \( \nu^- \) is the limiting value of the solution on the boundary from the inside of \( \partial \Omega \).

Now the fast algorithm can be used and only some minor changes are needed. First, instead of using \([\beta u_n] = g\) as the convergence-checking rule, we use \(\nu^- = v\). Therefore, instead of interpolating \( u_n^+ \) and \( u_n^- \) with the knowledge of \( u_{ijk} \)'s, we need to find \( u^+ \) and \( u^- \). The same least square approach can still be used here. For example, our interpolation formula for \( u^- \) can be written in the following

\[
u^- \approx \sum_{i,j,k} \gamma_{ijk}u_{ijk} - Q.
\]

(4.5)

The same idea also applies to an exterior Poisson equation with Dirichlet boundary conditions

\[
\Delta u = f, \quad \text{in } \Omega,
\]

\[
u = v, \quad \text{on } \partial \Omega,
\]

\[
u = u_{bc}, \quad \text{on } \partial R,
\]

(4.6)

where \( \Omega \) is a cubic volume with an arbitrary closed void region, \( \partial \Omega \) is the interior boundary of \( \Omega \) and \( \partial R \) is the exterior boundary of \( \Omega \), see Figure 4.2 for an illustration.

By extending the source term in the Poisson equation by zero inside the void and forming interface conditions across \( \partial \Omega \), we have the following interface problem

\[
\Delta u = \begin{cases} 
  f, & \text{if } (x, y, z) \in \Omega, \\
  0, & \text{if } (x, y, z) \notin \Omega,
\end{cases}
\]

\[
[u] = v, \quad \text{on } \partial \Omega,
\]

\[
[u_n] = g, \quad \text{on } \partial \Omega,
\]

\[
u = u_{bc}, \quad \text{on } \partial R.
\]

(4.7)
And similarly, we then use the GMRES iteration to update $g$ until the original interior boundary condition is satisfied, i.e.,

$$u^+ = v, \quad \text{on } \partial \Omega,$$

where $u^+$ is the limiting value of the solution on the interior boundary from the outside of $\partial \Omega$, which can still be obtained by the same least square approach, i.e.,

$$u^+ \approx \sum_{i,j,k} \gamma_{ijk} u_{ijk} - Q.$$

Figure 4.2: A diagram for an exterior Poisson problem.

Actually, the same code can be used to solve both interior or exterior problems. Only difference is that if $\Omega^+$ denotes the outside of the interface in an exterior problem, then it denotes the inside of the interface in an interior problem, so does $\Omega^-$. 

We have tested the embedding approach by solving some exterior or interior Poisson equations with Dirichlet boundary conditions. All computations are done on Sun ULTRA 10 workstations.
Example 4.1 The differential quation is

\[ u_{xx} + u_{yy} + u_{zz} = 20r^2 + \frac{\log e}{r^2}, \]

in the cube \([-1, 1] \times [-1, 1] \times [-1, 1]\) outside the sphere \(x^2 + y^2 + z^2 = 1/4\). The Dirichlet boundary condition is chosen from the following exact solution

\[ u(x, y, z) = r^4 + \log(2r) + \frac{3}{16}, \]

where \(r = \sqrt{x^2 + y^2 + z^2}\).

Figure 4.3 shows a slice of the computed solution: \(-u(x, y, 0)\).

**Figure 4.3:** A slice of the computed solution (\(n = 80\)) for Example 4.1: \(-u(x, y, 0)\).
Example 4.2 The differential equation is

\[ u_{xx} + u_{yy} + u_{zz} = -3 \sin x \cos y \cos z, \]

in the cube \([-1, 1] \times [-1, 1] \times [-1, 1]\) outside the ellipsoid \(x^2 + 4y^2 + 2z^2 = 1/4\). The Dirichlet boundary condition is chosen from the following exact solution

\[ u(x, y, z) = \sin x \cos y \cos z. \]

Figure 4.4 shows a slice of the computed solution: \(u(x, y, 0)\).

Figure 4.4: A slice of the computed solution \((n = 80)\) for Example 4.2: \(u(x, y, 0)\).
Example 4.3 The differential equation is

\[ u_{xx} + u_{yy} + u_{zz} = -3 \sin x \sin y \cos z, \]

in the ellipsoid \[ 2x^2 + y^2 + 4z^2 = 0.49. \] The Dirichlet boundary condition is chosen from the following exact solution

\[ u(x, y, z) = \sin x \sin y \cos z + 1.0. \]

Figure 4.5 shows a slice of the computed solution: \( u(x, y, 0) \).

Figure 4.5: A slice of the computed solution (\( n = 80 \)) for Example 4.3: \( u(x, y, 0) \).
Example 4.4 The differential equation is
\[ u_{xx} + u_{yy} + u_{zz} = -3 \sin x \sin y \cos z, \quad \text{in } \Omega, \]
where \( \Omega \) is the interior region with the boundary shown in Figure 3.5. The Dirichlet boundary condition is chosen from the following exact solution
\[ u(x, y, z) = \sin x \sin y \cos z + 1.0. \]

Figure 4.6 shows a slice of the computed solution: \( u(x, y, 0) \).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig46}
\caption{A slice of the computed solution (\( n = 41 \)) for Example 4.4: \( u(x, y, 0) \).}
\end{figure}
Table 4.1 and 4.2 show that the approach is second order accurate. Also, the number of calls to the fast Poisson solver on the cubic domain is almost independent of the mesh size, although it may depend on the geometry of the domain.

**Table 4.1:** The grid refinement analysis for Example 4.1.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n_{irreg}$</th>
<th>$n_{contr}$</th>
<th>CPU</th>
<th>$n_{iter}$</th>
<th>$| E_n |_\infty$</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>2096</td>
<td>1113</td>
<td>228.658</td>
<td>22</td>
<td>$1.617 \times 10^{-4}$</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>8328</td>
<td>4296</td>
<td>1008.865</td>
<td>22</td>
<td>$2.918 \times 10^{-5}$</td>
<td>5.541</td>
</tr>
</tbody>
</table>

**Table 4.2:** The grid refinement analysis for Example 4.2.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n_{irreg}$</th>
<th>$n_{contr}$</th>
<th>CPU</th>
<th>$n_{iter}$</th>
<th>$| E_n |_\infty$</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1148</td>
<td>623</td>
<td>122.707</td>
<td>21</td>
<td>$6.074 \times 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>4564</td>
<td>2369</td>
<td>603.228</td>
<td>22</td>
<td>$8.545 \times 10^{-4}$</td>
<td>7.108</td>
</tr>
</tbody>
</table>

So far, in the discussion for a Poisson equation on an irregular domain, we form an interface problem that requires a known fixed jump in the solution and set an unknown jump in the normal derivative of the solution. Then we iteratively update the jump in the normal derivative using the GMRES iteration until the original boundary condition is satisfied. Alternatively, we can set a known fixed jump in the normal derivative and an unknown jump in the solution. Then we use a similar GMRES iteration to update the jump in the solution until the original boundary condition is satisfied. For example, the following interior Poisson equation with a Neumann boundary condition

$$
\Delta u = f, \quad \text{in } \Omega,
$$

$$
u_n = q, \quad \text{on } \partial \Omega,
$$

may be treated as the following interface problem

$$
\Delta u = \begin{cases} 
  f, & \text{if } (x, y, z) \in \Omega, \\
  0, & \text{if } (x, y, z) \not\in \Omega,
\end{cases}
$$

$$
[u] = v, \quad \text{on } \partial \Omega,
$$

$$
[u_n] = 0, \quad \text{on } \partial \Omega,
$$

$$
u = 0, \quad \text{on } \partial R.
$$

We then use the GMRES iteration to update $v$ until the original boundary condition is satisfied, i.e.,

$$
u_n^{-} = q, \quad \text{on } \partial \Omega.
$$
4.2 Solving an inverse interface problem using the fast algorithm

In this section, we discuss the application of the fast algorithm for an inverse problem of the following forward interface problem

\[ -\nabla \cdot (\beta(x)\nabla u) = 0, \quad x \in \Omega, \quad (4.13) \]
\[ u_n = g, \quad \text{on} \quad \partial \Omega, \quad (4.14) \]
\[ [u] = 0, \quad \text{on} \quad \Gamma, \quad (4.15) \]
\[ [\beta u_n] = 0, \quad \text{on} \quad \Gamma, \quad (4.16) \]

where \( \Omega \) is a cubic domain with the boundary \( \partial \Omega \), and \( \Gamma \) is an interface within the domain, see Figure 4.7, and the coefficient \( \beta \) is piecewise constant and given by

\[ \beta(x) = \begin{cases} \beta^+, & x \in \Omega^+, \\ \beta^-, & x \in \Omega^-. \end{cases} \]

\[ \partial \Omega \]

\[ \Omega^+ \]

\[ \Omega^- \]

\[ \Gamma \]

Figure 4.7: A diagram of the forward interface problem.

In real, \( u \) may represent a potential function and \( \beta \) represents the conductivity. The domain \( \Omega^- \) represents the inhomogeneity of the conducting medium. That is, if \( \Omega^- = \emptyset \), then we have a homogeneous conduction medium. We assume that \( \Omega^- \) is a finite union of
simply connected open sets in $\Omega$ and $\Gamma$ is the union of $C^2$ closed curves in $\Omega$ that represent
the interfaces between the two open domains $\Omega^+$ and $\Omega^-$. 

In the forward interface problem, the exact interface $\Gamma$ is known, and we want to solve for $u$. In general the solution of the forward problem depends on the interface $\Gamma$ so we write it as $u(\Gamma)$.

In the contrary, in an inverse interface problem we do not know what the interface $\Gamma$ is. Instead we have some measured data close to the boundary of the domain. We want to identify the unknown interface based on these measured data, see Figure 4.8, where $\hat{\Omega}$ is the region of the observation defined by

$$\hat{\Omega} = \{x \in \Omega : \text{distance}(x, \partial \Omega) \leq d\},$$
on which we have known information of $u$. Here $d > 0$ is a pre-chosen constant.

![Figure 4.8: A diagram of an inverse interface problem of identifying an unknown shape from the observed data taken from the shaded area $\hat{\Omega}$.](image)

### 4.2.1 The numerical algorithm

We start with an initial guess $\Gamma_0$ of the unknown interface. This initial surface will move in a direction normal to itself with some speed function (normal velocity) $V$. By tracking the motion of this surface as it evolves, we can identify the unknown interface.
Again we use the level set approach to track the propagation of the interface. Let the
level set function be \( \phi = \phi(t, x) \), \( t \geq 0, \ x \in \mathbb{R}^3 \). Then this time-dependent level set function
\( \phi \) defines a family of the interfaces \( \Gamma_t \) by
\[
\Gamma_t = \{ x \in \mathbb{R}^3 : \phi(t, x) = 0 \},
\]
and the domains
\[
\Omega^+_t = \{ x \in \Omega : \phi(t, x) > 0 \}, \quad \Omega^-_t = \{ x \in \Omega : \phi(t, x) < 0 \}.
\]

As we know, much of the challenge in moving interface problems comes from producing
an adequate model for the speed function \( V \). In this section, we use the model presented
in [35].

Let \( \Gamma_{\text{shape}} \) be the unknown interface and \( u_{\text{ob}} \) be the observation of \( u \) at \( \hat{\Omega} \). Given the
interface \( \Gamma \), let \( u(\Gamma) \in H^1(\Omega) \) denote the solution to the boundary value problem (4.13). Define the variational form
\[
\min J(\Gamma) = \frac{1}{2} \iint \int_{\Omega} |u(\Gamma) - u_{\text{ob}}|^2 \, dx \, dy \, dz + \mu \int \int_{\Gamma} 1 \, dS,
\]  
over \( \Gamma \in Q_{\text{ad}} \), where \( Q_{\text{ad}} \) is an admissible class of the interfaces, and \( \mu \) is a constant
(regularization parameter). Then the normal velocity \( V \) is the rate of change of \( J \) in the
steepest descent direction. Therefore we can use the level set method to move the interface
closer to the minimum, which is the exact shape \( \Gamma_{\text{shape}} \).

The effects of the term \( \mu \int \int_{\Gamma} 1 \, dS \) in (4.17) are profound. Actually it will give us a
curvature-dependent speed function \( V \) with the curvature term being \( \mu \kappa \), where \( \kappa \) is the
curvature of the interface. The constant \( \mu \) acts as an advection term, and is independent of
the moving interface’s geometry. The diffusive curvature term \( \mu \kappa \) depends on the geometry
of the interface, and with a suitable \( \mu \), it smooths out the high curvature regions of the
interface. Therefore the interface will not form corners and will stay smooth for all time as
it moves.

For the sake of the simplicity, we assume \( \beta^- = \infty \) and thus the boundary value problem
(4.13) reduces to
\[
-\Delta u = 0, \quad x \in \Omega^+,
\]  
with boundary conditions
\[
u = 0, \ \text{on} \ \Gamma_{\text{shape}} \ \text{and} \ u_n = g, \ \text{on} \ \partial \Omega.
\]
Then it can be shown, see [35], that the Hamilton-Jacobi equation for the level set function $\phi = \phi(t, x)$ is

$$\phi_t + V(t, x) |\nabla \phi| = 0,$$

(4.20)

where the vector field $V = V(t, x)n$ is defined by

$$V(t, x) = -\nabla u \cdot \nabla p + \mu \kappa, \quad \text{on } \Gamma_t,$$

(4.21)

where $u \in H^1_0(\Omega^+)$ satisfies

$$\begin{cases}
-\Delta u &= 0, \quad \text{in } \Omega^+_t, \\
u &= 0, \quad \text{on } \Gamma_t, \\
u_n &= g, \quad \text{on } \partial \Omega,
\end{cases}$$

(4.22)

and the adjoint function $p \in H^1(\Omega^+_t)$ satisfies

$$\begin{cases}
-\Delta p &= (u - u_{ob})\chi_{\hat{\Omega}}, \quad \text{in } \Omega^+_t, \\
p &= 0, \quad \text{on } \Gamma_t, \\
p_n &= 0, \quad \text{on } \partial \Omega.
\end{cases}$$

(4.23)

Here, $\chi_{\hat{\Omega}}$ is the characteristic function of the domain $\hat{\Omega}$.

Note that (4.22) and (4.23) are two exterior Poisson equations with Dirichlet boundary conditions on the interface $\Gamma_t$, therefore we can use the fast algorithm to solve them as described in § 4.1.

In summary, given a domain and a set of observed data on the domain, the algorithm to identify an unknown shape $\Gamma_{\text{shape}}$ inside the domain is outlined below:

1. Set an initial level set function $\phi$ as the initial guess of the unknown shape: $\Gamma_0 = \{(x, y, z) : \phi(x, y, z) = 0\}$. Let $\Gamma = \Gamma_0$.

2. Solve the Laplace equation (4.22) in the exterior of $\Gamma$ to get $u = u(\Gamma)$.

3. Compute the difference of the computed solution with the observed data $(u(\Gamma) - u_{ob})\chi_{\hat{\Omega}}$.

4. Check convergence. If $\|(u(\Gamma) - u_{ob})\chi_{\hat{\Omega}}\| \leq \varepsilon_1$, then stop, where $\varepsilon_1 > 0$ is a pre-chosen tolerance constant. Otherwise, continue.
5. Solve the Poisson equation (4.23) in the exterior of $\Gamma$ to get $p = p(\Gamma)$.

6. Evaluate the normal velocity $V$ using a weighted least square interpolation to get

$$V = -\nabla u \cdot \nabla p + \mu \kappa, \quad \text{on } \Gamma,$$

where $\mu$ is a regularization parameter, and $\kappa$ is the curvature of the interface.

7. Check convergence (optional). If $\|V\| \leq \varepsilon_2$, then stop, where $\varepsilon_2 > 0$ is a pre-chosen tolerance constant. Otherwise, continue.

8. Extend the normal velocity $V$ to a computational tube $|\phi| \leq \delta$, where $\delta$ is the width of the tube.

9. Update the level set function by solving the Hamilton-Jacobi equation $\phi_t + V |\nabla \phi| = 0$.

10. Reinitialize the interface.

11. Let $\Gamma$ be defined by the new level set function. Go to Step 2 and repeat the process.

### 4.2.2 Evaluation of gradient vectors at irregular grid points

In order to use the level set method, we need the normal velocity at all grid points in the computational tube. The usual approach is to find the normal velocity at irregular grid points. To get robust and stable gradient at an irregular grid point $(x_{i_0}, y_{j_0}, z_{k_0})$, we use the weighted least square interpolation, for example

$$u_x(x_{i_0}, y_{j_0}, z_{k_0}) \approx \sum_{i,j,k} \gamma_{ijk} u_{i_0+i,j_0+j,k_0+k},$$

(4.25)

where the summation is taken over a set of grid points $(x_{i_0+i}, y_{j_0+j}, z_{k_0+k}) \in \Omega^+$ in the computational tube. The coefficients $\gamma_{ijk}$’s can be obtained in the same way as discussed in Chapter 2, see (2.64), (2.65), (2.66) and (2.67).

If we choose a non-zero regularization parameter $\mu$, we then need to calculate the curvature $\kappa$ of the interface. In our simulation, we use the mean curvature $\kappa_M$, see (2.62).
4.2.3 Extension of the velocity

After we have evaluated the normal velocities at the irregular grid points, the velocity then need to extend to all grid points inside the computational tube surrounding the boundary of the shape. This is done through an upwind scheme along the normal direction away from the interface

\[ V_t \pm \nabla V \cdot \frac{\nabla \phi}{|\phi|} = 0, \]

where \( V \) is the normal velocity. The sign is determined from the normal direction of the level set function. We illustrate this through an example in three dimensions in Figure 4.9, where Figure 4.9 (a) plots a slice of the computed velocity at irregular grid points, and Figure 4.9 (b) shows a slice of the velocity field in a computational tube after the extension.

![Figure 4.9: Velocity extension. (a) A slice of the computed velocity at irregular grid points. (b) A slice of the velocity field in a computational tube after the extension.](image)

Technically, the motion of the interface is uniquely determined by the normal velocity of the interface. By constructing the normal velocity to an neighborhood of the interface, we then avoid unnecessary numerical difficulties when we update the level set function. Therefore it seems that a first order method is good enough since the velocity at irregular grid points is unchanged. The velocity away from interface does not have much effects on the position of the interface rather the behavior of the level set function.

4.2.4 The level set method

Let \( \phi_{ijk}^n \) be the approximation to the solution \( \phi(n\Delta t, x_i, y_j, z_k) \), where \( \Delta t \) is the time step. We may use an explicit finite difference approach as

\[ \frac{\phi_{ijk}^{n+1} - \phi_{ijk}^n}{\Delta t} + V_{ijk} |\nabla_{ijk} \phi_{ijk}^n| = 0, \]
where the forward Euler difference scheme in time has been used, and $|\nabla_{ijk} \phi_{ijk}^n|$ represents some appropriate finite difference operator for the spatial derivative. In our simulation, we use the level set scheme given in [65]

$$
\phi_{ijk}^{n+1} = \phi_{ijk}^n - \Delta t \left[ \max(V_{ijk}, 0) \nabla^+ + \min(V_{ijk}, 0) \nabla^- \right],
$$

where

$$
\nabla^+ = \left[ \max(A, 0)^2 + \min(B, 0)^2 + \max(C, 0)^2 + \min(D, 0)^2 + \max(E, 0)^2 + \min(F, 0)^2 \right]^{1/2},
$$

$$
\nabla^- = \left[ \max(B, 0)^2 + \min(A, 0)^2 + \max(D, 0)^2 + \min(C, 0)^2 + \max(F, 0)^2 + \min(E, 0)^2 \right]^{1/2},
$$

and

$$
A = D^{-x} \phi_{ijk}^n + \frac{\Delta x}{2} s(D^{-x-x} \phi_{ijk}^n, D^{+x-x} \phi_{ijk}^n),
$$

$$
B = D^{+x} \phi_{ijk}^n - \frac{\Delta x}{2} s(D^{+x+x} \phi_{ijk}^n, D^{-x-x} \phi_{ijk}^n),
$$

$$
C = D^{-y} \phi_{ijk}^n + \frac{\Delta y}{2} s(D^{-y-y} \phi_{ijk}^n, D^{+y-y} \phi_{ijk}^n),
$$

$$
D = D^{+y} \phi_{ijk}^n - \frac{\Delta y}{2} s(D^{+y+y} \phi_{ijk}^n, D^{-y-y} \phi_{ijk}^n),
$$

$$
E = D^{-z} \phi_{ijk}^n + \frac{\Delta z}{2} s(D^{-z-z} \phi_{ijk}^n, D^{+z-z} \phi_{ijk}^n),
$$

$$
F = D^{+z} \phi_{ijk}^n - \frac{\Delta z}{2} s(D^{+z+z} \phi_{ijk}^n, D^{-z-z} \phi_{ijk}^n).
$$

Here the switch function $s(a, b)$ is given by

$$
s(a, b) = \begin{cases} 
    a, & \text{if } |a| \leq |b| \\
    b, & \text{if } |a| > |b| \\
    0, & \text{if } ab < 0
\end{cases}, \quad ab \geq 0,
$$

and the operators $D^{-x} \phi_{ijk}^n, D^{+x} \phi_{ijk}^n, D^{+x-x} \phi_{ijk}^n$ are defined as

$$
D^{-x} \phi_{ijk}^n = \frac{\phi_{ijk}^n - \phi_{i-1,jk}^n}{\Delta x}, \\
D^{+x} \phi_{ijk}^n = \frac{\phi_{i+1,jk}^n - \phi_{ijk}^n}{\Delta x}, \\
D^{+x-x} \phi_{ijk}^n = \frac{\phi_{i+1,jk}^n - 2\phi_{ijk}^n + \phi_{i-1,jk}^n}{\Delta^2 x},
$$
and all other operators are defined in the same manner. The essentially non-oscillatory (ENO) construction [30] can be used here.

### 4.2.5 The time step

There are inherent time step requirements in the above first order time explicit scheme. Analogous with the underlying wave equation, we have a Courant-Friedrichs-Levy (CFL) condition which, for an advective speed function \( V \) and a first order space scheme, requires the interface to cross no more than one grid cell each time step. Thus, we require that

\[
\max_{\Omega} |V| \Delta t \leq h, \tag{4.38}
\]

where the maximum is taken over values for \( V \) at all possible grid points, not simply those corresponding to the zero level set. In a narrow band implementation [15], the time step can be adaptively chosen in response to the maximum velocity within the narrow band.

In our simulation, we take the time step as

\[
\Delta t = \min(10, \frac{h}{4 v_{\text{max}}}),
\]

where \( v_{\text{max}} \) is the maximum magnitude of the velocity in the computational tube. Based on the CFL condition, we could use \( h/v_{\text{max}} \). But because the problem is non-linear, we take the time step as \( h/(4v_{\text{max}}) \) to guarantee the stability. Also notice that for the inverse interface problem, when the computed interface approaches the exact interface, the velocity field will approach zero and thus \( h/(4v_{\text{max}}) \) will become very large. Therefore, for accuracy we require that the time step is bounded above by a constant.

### 4.2.6 Reinitialization

The initial level set function \( \phi(0, x) \) is often chosen as the signed normal distance from the interface which means \( |\nabla \phi| = 1 \). By solving the modified Hamilton-Jacob equation, we can update the moving interface. However, while (4.20) will move the interface at the correct speed, \( \phi \) in general is no longer a distance function. In fact, \( \phi \) may become too steep or flat, especially when topological changes such as breaking and merging take place. This difficulty can be avoided by a reinitialization process introduced in [72, 73] so that \( \phi \) will remain as the signed distance function up to a certain accuracy. The new level set function is the steady solution of the equation

\[
\phi_t + (|\nabla \phi| - 1) H(\phi) = 0, \tag{4.39}
\]

where \( H(\phi) \) is any smooth monotone increasing function of \( \phi \) with \( H(0) = 0 \).
4.2.7 Numerical examples

We performed some numerical experiments on Sun ULTRA 10 workstations. Due to the restriction of computing resources, we tried only $60 \times 60 \times 60$ grid. The computational domain is always $[-1, 1] \times [-1, 1] \times [-1, 1]$. It can be observed that the algorithm works well for single object in the domain.

**Example 4.5** First we test the above numerical approach for a sphere in the domain. The exact shape is defined by

$$x^2 + y^2 + z^2 = 0.09.$$  

The initial guess is taken as the sphere $x^2 + y^2 + z^2 = 0.16$. To test the effect of regularization, we tried two cases $\mu = 0$ (no regularization) and $\mu = 0.001$. Figure 4.10–Figure 4.12 show the computed results. Note that the interface forms a corner if no regularization is used.

Figure 4.10: The computed shape for Example 4.5 using $60 \times 60 \times 60$ grid and $\mu = 0.001$ at different stages. (a) The initial guess. (b) After 11 iterations. (c) After 31 iterations. (d) After 51 iterations.
Figure 4.11: The $x$-$y$ transection of the computed shape for Example 4.5 using $60 \times 60 \times 60$ grid and $\mu = 0.001$ at different stages.  (a) The initial guess.  (b) After 11 iterations.  (c) After 31 iterations.  (d) After 51 iterations. The inner-most circle is the exact solution. The interface stays smooth for all time as it evolves.
Figure 4.12: The $x$-$y$ transection of the computed shape for Example 4.5 using $60 \times 60 \times 60$ grid but $\mu = 0$ at different stages. (a) The initial guess. (b) After 11 iterations. (c) After 31 iterations. (d) After 51 iterations. The inner-most circle is the exact solution. The interface gradually forms a corner as it moves.
Example 4.6 In this example we test the above numerical approach for an ellipsoid in the domain. The exact shape is defined by

\[ x^2 + 3y^2 + z^2 = 0.09. \]

The initial guess is taken as the sphere \( x^2 + y^2 + z^2 = 0.16 \), and no regularization is used. Figure 4.13 (a) shows the exact shape, and Figure 4.13 (b) shows the computed shape after 201 iterations. Also, Figure 4.14 and Figure 4.15 show some intermediate computed results. Notice that we only used a \( 60 \times 60 \times 60 \) grid.

Figure 4.13: (a) The exact shape for Example 4.6. (b) The computed shape for Example 4.6 with \( 60 \times 60 \times 60 \) grid after 201 iterations.
Figure 4.14: The computed shape for Example 4.6 using $60 \times 60 \times 60$ grid at different stages. (a) The initial guess. (b) After 41 iterations. (c) After 121 iterations. (d) After 201 iterations. The interface does not stay smooth for all time as it evolves since no regularization is used.
Figure 4.15: The $y$-$z$ transection of the computed shape for Example 4.6 using $60 \times 60 \times 60$ grid at different stages. (a) The initial guess. (b) After 41 iterations. (c) After 121 iterations. (d) After 201 iterations. The inner-most ellipse is the exact solution.
Example 4.7 Now we consider an example that has two objects in the domain. The exact shape is composed of the following two spheres

\begin{align}
    x^2 + (y - 0.1)^2 + (z - 0.1)^2 &= 0.01, \quad (4.40) \\
    x^2 + (y + 0.1)^2 + (z + 0.1)^2 &= 0.01. \quad (4.41)
\end{align}

It can be modeled as the zero set of the level set function

\[ \phi = \min \{ x^2 + (y - 0.1)^2 + (z - 0.1)^2 - 0.01, \ x^2 + (y + 0.1)^2 + (z + 0.1)^2 - 0.01 \}. \quad (4.42) \]

The initial guess is taken as the sphere \( x^2 + y^2 + z^2 = 0.09 \), and no regularization is used. The algorithm produces an envelope of the two objects. For the minimization problem (4.17), the local minimum may not be unique. Instead there can be several local minimums and global minimums. The envelope obtained seems to be a local minimum. In some situations, for example the two objects are far apart, then we can use the technique described in [35] to find the two objects.

Figure 4.16 (a) shows the exact shape, and Figure 4.16 (b) shows the computed envelope after 241 iterations with \( 60 \times 60 \times 60 \) grid. In Figure 4.17 and Figure 4.18, we show some intermediate computed results.

Figure 4.16: (a) The exact shape for Example 4.7. (b) The computed shape which is an envelope of the two objects for Example 4.7 with \( 60 \times 60 \times 60 \) grid after 241 iterations.
Figure 4.17: The computed shape for Example 4.7 using $60 \times 60 \times 60$ grid at different stages. (a) The initial guess. (b) After 41 iterations. (c) After 161 iterations. (d) After 241 iterations.
Figure 4.18: The $y$-$z$ transection of the computed shape for Example 4.7 using $60 \times 60 \times 60$ grid at different stages.  (a) The initial guess.  (b) After 41 iterations.  (c) After 161 iterations.  (d) After 241 iterations. The two small circles are the $y$-$z$ transections of the two exact objects.
Chapter 5

Conclusions

We have developed an immersed interface method for three-dimensional elliptic interface problems. By using the level set method, we discussed how to determine geometric information of the interface, how to locate the projection of an irregular grid point on the interface, how to reconstruct the interface by interpolation, and how to perform local coordinate transformation.

Then based on the immersed interface method, we have developed the maximum principle preserving scheme for arbitrary coefficients in three dimensions using direct finite difference discretization. The new scheme satisfies the sign property that guarantees the discrete maximum principle. The sign property is enforced through a constrained quadratic optimization problem. The resulting system of discrete equations is diagonally dominant and its symmetric part is negative definite so the SOR method or the AMG method can be used to solve the system of discrete equations. Numerical experiments confirm the expected second order accuracy.

In Chapter 3, we have described a second order fast algorithm for elliptic interface problems with piecewise constant but discontinuous coefficients. Before applying the immersed interface method, we precondition the differential equation first. In order to take advantage of existing fast Poisson solvers on cubic domains, an intermediate unknown function, the jump in the normal derivative across the interface, is introduced. Then the GMRES iteration is employed to solve the Schur complement system derived from the discretization. Numerical experiments showed that the fast algorithm was very successful and efficient when the coefficients are piecewise constant. Especially, the number of iterations in solving the Schur complement system is independent of the mesh size.

In Chapter 4, we have investigated some applications of the fast algorithm. First we developed an embedding technique to solve interior or exterior Poisson equations on complicated regions with Dirichlet or Neumann boundary conditions. The idea is to embed the irregular region into a cube, to extend the Poisson equation to the entire cubic domain, to introduce suitable jump conditions and to get an interface problem. The new embedding
method was then applied to solve an inverse interface problem, i.e., identifying unknown shapes using the measured data close to the boundary of the object.

As a part of this thesis, we have developed some general application programs for solving three-dimensional elliptic interface problems, and Poisson equations on irregular domains.

We performed some simulations on how to identify unknown shapes using the measured data close to the boundary. However, in our simulations all measured data are exact. In practice, the observed data generally have noise in them. We plan to do some noise analysis in the future.

Because of the discontinuity and non-smoothness in the solution and the complexity of the interface, it is difficult to perform convergence analysis in the conventional way. In the future we hope to develop some analysis of the the maximum principle preserving scheme and the fast algorithm.
List of References


References


Level set function for Example 3.5

function phi0(x,y,z)

r0 = sqrt(x*x+y*y+z*z)

if (r0 .ge. 1e-10) then
  cosx = x/sqrt(x*x+y*y+z*z)
  cosy = y/sqrt(x*x+y*y+z*z)
  cosz = z/sqrt(x*x+y*y+z*z)

  thetax = acos(cosx)
  thetay = acos(cosy)
  thetaz = acos(cosz)

  r1 = 0.4
  r2 = r1 + 0.15*(sin(3.0*thetax))
  1 + 0.15*(sin(3.0*thetay))
  1 + 0.15*(sin(3.0*thetaz))

  phi0 = r0 - r2
else
  write(*, *) " Please use a different n"
  stop
endif

return

end