Geometric Multigrid and Closest Point Methods for Surfaces and General Domains

Yujia Chen

St Anne's College

University of Oxford

A thesis submitted for

Doctor of Philosophy

Trinity Term 2015

Abstract

This thesis concerns the analytical and practical aspects of applying the Closest Point Method to solve elliptic partial differential equations (PDEs) on smooth surfaces and domains with smooth boundaries. A new numerical scheme is proposed to solve surface elliptic PDEs and a novel geometric multigrid solver is constructed to solve the resulting linear system. The method is also applied to coupled bulk-surface problems.

A new embedding equation in a narrow band surrounding the surface is formulated so that it agrees with the original surface PDE on the surface and has a unique solution which is constant along the normals to the surface. The embedding equation is then discretized using standard finite difference scheme and barycentric Lagrange interpolation. The resulting scheme has 2nd-order accuracy in practice and is provably 2nd-order convergent for curves without boundary embedded in $\mathbb{R}^2$.

To apply the method to solve elliptic equations on surfaces and domains with boundaries, the "ghost" point approach is adopted to handle Dirichlet, Neumann and Robin boundary conditions. A systematic method is proposed to represent values of ghost points by values of interior points according to boundary conditions.

A novel geometric multigrid method based on the closest point representation of the surface is constructed to solve the resulting large sparse linear systems. Multigrid solvers are designed for surfaces with or without boundaries and domains with smooth boundaries. Numerical results indicate that the convergence rate of the multigrid solver stays roughly the same as we refine the mesh, as is desired of a multigrid algorithm.

Finally the above methods are combined to solve coupled bulk-surface PDEs with some applications to biology.

Acknowledgements

I'd like to thank my supervisor Prof. Colin Macdonald for his guidance, encouragement and kindness. I also thank Prof. Andrew Wathen, Prof. Nick Trefethen, Prof. Endre Süli, Prof. Charles M. Elliott, Dr. Ingrid von Glehn and Dr. Tom März for their helpful discussions and suggestions.

I am grateful to the Oxford Centre for Collaborative Applied Mathematics and King Abdullah University of Science and Technology who funded my research.

Finally, I thank my parents for their love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Surface partial differential equations (PDEs) have a wide range of applications in science and engineering. For example, they arise in modelling pattern formation on biological surfaces [6], ocean flows or atmospheric flows on the surface of the earth [86], image de-noising [10] or segmentation [82] on curved surfaces, coupled bulk-surface diffusion processes in cell biology [68], and interfacial flows with surfactant on the interface [87]. Since analytical solutions of surface PDEs are difficult to obtain in most applications, numerical methods are essential for these applications. Among various numerical techniques to solve surface PDEs, the Closest Point Method [79, 61, 62, 59, 65] is a simple embedding method which works for a wide range of surface PDEs.

This thesis makes substantial contributions to applying the Closest Point Method to solve elliptic PDEs on surfaces. A new embedding equation for the surface elliptic equation is formulated and discretized, and a 2nd-order convergence proof is obtained. Also, a novel geometric multigrid algorithm is constructed based on the closest point representation of the surface.

This chapter will mention several numerical methods for solving surface PDEs, introduce the concepts of surface intrinsic differential operators, review the Closest Point Method, and present the outline of the thesis.

1.1 Numerical methods to solve surface PDEs

There are a variety of numerical techniques to solve surface PDEs, typically depending on the geometrical representation of the underlying surface and to some extent, the background and expertise of the practitioner:

- One can find a parametrization of the surface [32], formulate and solve the PDE in the parameter space.
- One can also build a Voronoi tessellation of the surface [50, 20], and formulate a finite volume scheme for surface PDEs [45, 21, 22, 68].
- Based on a triangulation or a polygonal approximation of the surface, one can solve the PDE on the approximated surface by finite differences [83] or finite elements [23, 24].
- A novel type of surface finite element method [70, 69, 19] is based on the restriction (trace) of a standard finite element space on a tetrahedral triangulation of an outer domain that contains the surface. This method can handle PDEs on moving surfaces [71].
- Based on a level set representation of the surface, one can formulate an embedding PDE corresponding to the surface PDE, then use either finite differences [9, 37] or finite elements [14, 25, 18] to numerically solve the PDE in the embedding space. This level set approach is also desirable for PDEs on moving surfaces [88, 26].
- Another embedding method is the diffuse interface approach [77, 54] which introduces a diffuse interface region of a phase-field variable. The surface then becomes the 1/2-level set of a phase-field variable, and an embedding PDE is formulated based on the diffuse interface representation.
- Another method suitable for moving surfaces is the grid based particle method [52, 51] which moves the interface by evolving a set of particles and local reconstructions. One can then approximate surface differential operators using the local reconstructions (see also the point cloud approach next).
- Starting from a point cloud (supposed to be sampled from the surface), one can reconstruct the surface by radial basis functions (RBFs), and solve the PDE on the reconstructed surface using RBF methods such as the RBF projection method [33] and the RBF orthogonal gradients method [74]. There are also other approaches to compute on point clouds with no reconstruction procedure [60] or with only local reconstruction [57].

This thesis will adopt the Closest Point Method based on a closest point representation of the surface. As we shall see later, the method extends surface function to a narrow band so that the extended function is constant along the normals to the surface. Therefore, we can replace the surface intrinsic differential operators with the corresponding Cartesian differential operators to formulate an embedding equation in the narrow band. We then discretize and numerically solve the embedding equation in a banded, uniform Cartesian grid with finite differences.

1.2 Surface intrinsic differential operators

In this section we review several well-known definitions of surface intrinsic differential operators (see e.g., [25, 65]) for two dimensional surfaces embedded in $\mathbb{R}^3$.

Suppose for the moment that $u(\boldsymbol{y})$ is a scalar function defined on a smooth surface $\mathcal{S}$ embedded in $\mathbb{R}^3$, and $\tilde{u}(\boldsymbol{x})$ defined in $\mathbb{R}^3$ is an extension of $u(\boldsymbol{y})$ such that $\tilde{u}(\boldsymbol{y}) = u(\boldsymbol{y})$ for any $\boldsymbol{y} \in \mathcal{S}$. Let $\boldsymbol{n}(\boldsymbol{y})$ be the unit normal vector to $\mathcal{S}$ at $\boldsymbol{y}$. The surface gradient of u at $\boldsymbol{y} \in \mathcal{S}$ is defined as

$$\nabla_{\mathcal{S}} u(\boldsymbol{y}) = \nabla_{\mathcal{S}} \tilde{u}(\boldsymbol{y}) = \nabla \tilde{u}(\boldsymbol{y}) - \big(\boldsymbol{n}(\boldsymbol{y}) \cdot \nabla \tilde{u}(\boldsymbol{y})\big)\boldsymbol{n}(\boldsymbol{y}). \tag{1.2.1}$$

If we define the projection matrix at $\boldsymbol{y} \in \mathcal{S}$ to be

$$\boldsymbol{P}(\boldsymbol{y}) = \boldsymbol{I} - \boldsymbol{n}(\boldsymbol{y})\boldsymbol{n}(\boldsymbol{y})^T, \tag{1.2.2}$$

where $\boldsymbol{I}$ is the identity matrix, then $\nabla_{\mathcal{S}} u(\boldsymbol{y})$ can be rewritten as

$$\nabla_{\mathcal{S}} u(\boldsymbol{y}) = \nabla_{\mathcal{S}} \tilde{u}(\boldsymbol{y}) = \boldsymbol{P}(\boldsymbol{y}) \nabla \tilde{u}(\boldsymbol{y}). \tag{1.2.3}$$

We can also denote the surface gradient in components by

$$\nabla_{\mathcal{S}} u(\boldsymbol{y}) = \nabla_{\mathcal{S}} \tilde{u}(\boldsymbol{y}) = (\mathcal{D}_1 \tilde{u}(\boldsymbol{y}), \mathcal{D}_2 \tilde{u}(\boldsymbol{y}), \mathcal{D}_3 \tilde{u}(\boldsymbol{y}))^T, \qquad \boldsymbol{y} \in \mathcal{S}, \tag{1.2.4}$$

where $\mathcal{D}_i$, $i = 1, 2, 3$ are the components of the surface gradient. We remark that the $\mathcal{D}_i$'s actually depend on $\boldsymbol{y}$.

Now suppose $\boldsymbol{v}(\boldsymbol{y}) : \mathcal{S} \to \mathbb{R}^3$ is a vector field defined on $\mathcal{S}$, and it has an extension $\tilde{\boldsymbol{v}}(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}^3$ such that $\tilde{\boldsymbol{v}}(\boldsymbol{y}) = \boldsymbol{v}(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{S}$. Then the surface divergence of $\boldsymbol{v}$ is defined as

$$\nabla_{\mathcal{S}} \cdot \boldsymbol{v}(\boldsymbol{y}) = \nabla_{\mathcal{S}} \cdot \tilde{\boldsymbol{v}}(\boldsymbol{y}) = \sum_{i=1}^{3} \mathcal{D}_i \tilde{v}_i(\boldsymbol{y}), \qquad \boldsymbol{y} \in \mathcal{S}, \tag{1.2.5}$$

where $\tilde{v}_i(\boldsymbol{y})$, $i = 1, 2, 3$ are the components of the vector field $\tilde{\boldsymbol{v}}(\boldsymbol{y})$. After some calculations (see Appendix A), we have

$$\nabla_{\mathcal{S}} \cdot \tilde{\boldsymbol{v}}(\boldsymbol{y}) = \nabla \cdot \tilde{\boldsymbol{v}}(\boldsymbol{y}) - \boldsymbol{n}(\boldsymbol{y})^T \big(\nabla \tilde{\boldsymbol{v}}(\boldsymbol{y})\big) \boldsymbol{n}(\boldsymbol{y}), \qquad \boldsymbol{y} \in \mathcal{S}, \tag{1.2.6}$$

where $\nabla \tilde{\boldsymbol{v}}(\boldsymbol{y})$ is the Jacobian matrix of the vector function $\tilde{\boldsymbol{v}}$ at $\boldsymbol{y}$.

Furthermore, the surface Laplacian, or the Laplace–Beltrami function of a scalar function $u(\boldsymbol{y})$ with extension $\tilde{u}(\boldsymbol{x})$ is defined as

$$\Delta_{\mathcal{S}} u(\boldsymbol{y}) = \Delta_{\mathcal{S}} \tilde{u}(\boldsymbol{y}) = \nabla_{\mathcal{S}} \cdot \nabla_{\mathcal{S}} \tilde{u}(\boldsymbol{y}) = \sum_{i=1}^{3} \mathcal{D}_i \mathcal{D}_i \tilde{u}(\boldsymbol{y}), \qquad \boldsymbol{y} \in \mathcal{S}. \tag{1.2.7}$$

After some manipulations (see Appendix A), we get

$$\Delta_{\mathcal{S}} \tilde{u}(\boldsymbol{y}) = \Delta \tilde{u}(\boldsymbol{y}) - \kappa(\boldsymbol{y}) \frac{\partial \tilde{u}}{\partial \boldsymbol{n}}(\boldsymbol{y}) - \frac{\partial^2 \tilde{u}}{\partial \boldsymbol{n}^2}(\boldsymbol{y}), \qquad \boldsymbol{y} \in \mathcal{S}, \tag{1.2.8}$$

where $\kappa(\boldsymbol{y}) = \nabla \cdot \boldsymbol{n}(\boldsymbol{y})$ is the mean curvature at $\boldsymbol{y}$, $\frac{\partial \tilde{u}}{\partial \boldsymbol{n}}(\boldsymbol{y}) = \nabla \tilde{u}(\boldsymbol{y}) \cdot \boldsymbol{n}(\boldsymbol{y})$, and $\frac{\partial^2 \tilde{u}}{\partial \boldsymbol{n}^2}(\boldsymbol{y}) = \nabla\big(\nabla \tilde{u}(\boldsymbol{y}) \cdot \boldsymbol{n}(\boldsymbol{y})\big) \cdot \boldsymbol{n}(\boldsymbol{y})$.

Remark 1.2.1 *In general, one can define intrinsic differential operators on manifolds of any dimension and codimension [65], and the Closest Point Method is indeed able to deal with these more general cases [79, 61]. However this thesis will focus on analysis and applications where the underlying manifold is a curve embedded in $\mathbb{R}^2$ or a surface embedded in $\mathbb{R}^3$. We believe that some of the analysis and most of the implementations can be extended to handle more general manifolds, and this is left as future work. We state some results in arbitrary dimensions where doing so does not complicate the presentation.*

1.3 Review of the Closest Point Method

In this section, we review the concepts of closest point function, closest point extension, several closest point principles, and the original Ruuth–Merriman formulation of the Closest Point Method [79]. We also discuss briefly the development and current status of the Closest Point Method. Following [58], we use the terminology "surface" and the notation $\mathcal{S}$ to refer to general manifolds.

1.3.1 Closest point function and extension

Definition 1.3.1 (Closest point function) [58] *For a given surface $\mathcal{S}$ embedded in $\mathbb{R}^d$, the Euclidean* closest point function $\mathrm{cp} : \mathbb{R}^d \to \mathbb{R}^d$ *takes point $\boldsymbol{x} \in \mathbb{R}^d$ and returns a point $\mathrm{cp}(\boldsymbol{x}) \in \mathcal{S} \subset \mathbb{R}^d$ which is closest in Euclidean distance to $\boldsymbol{x}$. That is, $\mathrm{cp}(\boldsymbol{x}) = \underset{\boldsymbol{q}\in\mathcal{S}}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{q}\|_2$.*

If the surface $\mathcal{S}$ is smooth, there exists a tubular neighborhood [40, 65] $B(\mathcal{S})$ of $\mathcal{S}$ such that $\forall \boldsymbol{x} \in B(\mathcal{S})$, $\mathrm{cp}(\boldsymbol{x})$ is unique. For points which are not in $B(\mathcal{S})$, there might be multiple points that are closest to $\boldsymbol{x}$. In this case we define $\mathrm{cp}(\boldsymbol{x})$ to return an arbitrarily chosen closest point.

Theoretically a *closest point representation* [58] of $\mathcal{S}$ is defined once $\mathrm{cp}(\boldsymbol{x})$ is known for every $\boldsymbol{x} \in \mathbb{R}^d$. In practice there are various methods to compute the closest points for grid points and obtain a closest point representation for computation. For example, we can use analytical formulae for simple geometries, minimize the squared distance function by Newton's method for parametrized surfaces [59, 16], or compute the closest points from other representations of the surface such as a triangulation [62, 59], a point cloud [60, 51], or a level set function [16].

The closest point representation of a surface leads to a natural extension of surface data, namely the *closest point extension* [79, 62, 58]:

Definition 1.3.2 (closest point extension) *Let $\mathcal{S}$ be a smooth surface embedded in $\mathbb{R}^d$, and let $\mathrm{B}(\mathcal{S}) \subset \mathbb{R}^d$ be a tubular neighborhood of $\mathcal{S}$ in which the closest point function is uniquely defined. Then the* closest point extension *of a scalar function $u : \mathcal{S} \to \mathbb{R}$ is function $\tilde{u} : \mathrm{B}(\mathcal{S}) \to \mathbb{R}$ so that $\tilde{u}(\boldsymbol{x}) = u(\mathrm{cp}(\boldsymbol{x}))$ for $\boldsymbol{x} \in B(\mathcal{S})$. The* closest point extension *of a vector field $\boldsymbol{v} : \mathcal{S} \to \mathbb{R}^d$ is $\tilde{\boldsymbol{v}} : B(\mathcal{S}) \to \mathbb{R}^d$ so that $\tilde{\boldsymbol{v}}_i(\boldsymbol{x}) = \boldsymbol{v}_i(\mathrm{cp}(\boldsymbol{x}))$ for $\boldsymbol{x} \in B(\mathcal{S})$ and $i = 1, \ldots, d$.*

1.3.2 Equivalence of differential operators

The closest point extensions $\tilde{u}$ and $\tilde{\boldsymbol{v}}$ defined in Definition 1.3.2 are constant in the normal directions to the surface. This leads to simplified derivative calculations in the embedding space, which can be summarized by several principles [79, 65]:

Principle 1 (Gradient principle) *For a scalar function u defined on $\mathcal{S} \subset \mathbb{R}^d$, let $\tilde{u}(\boldsymbol{x}) = u(\mathrm{cp}(\boldsymbol{x}))$ defined in $B(\mathcal{S}) \subset \mathbb{R}^d$ be the closest point extension of u, then $\nabla_{\mathcal{S}} u(\boldsymbol{y}) = \nabla \tilde{u}(\boldsymbol{y})$ for $\boldsymbol{y} \in \mathcal{S}$.*

Principle 2 (Divergence principle) *For a vector field* $\boldsymbol{v}$ *defined on* $\mathcal{S} \subset \mathbb{R}^d$*, let* $\tilde{\boldsymbol{v}}(\boldsymbol{x}) = \boldsymbol{v}(\mathrm{cp}(\boldsymbol{x}))$ *defined in* $B(\mathcal{S}) \subset \mathbb{R}^d$ *be the closest point extension of* $\boldsymbol{v}$*, then* $\nabla_{\mathcal{S}} \cdot \boldsymbol{v}(\boldsymbol{y}) = \nabla \cdot \tilde{\boldsymbol{v}}(\boldsymbol{y})$ *for* $\boldsymbol{y} \in \mathcal{S}$*.*

Principle 3 (Laplacian principle) *For a scalar function* u *defined on* $\mathcal{S} \subset \mathbb{R}^d$*, let* $\tilde{u}(\boldsymbol{x}) = u(\mathrm{cp}(\boldsymbol{x}))$ *defined in* $B(\mathcal{S}) \subset \mathbb{R}^d$ *be the closest point extension of* u*,* $\Delta_{\mathcal{S}} u(\boldsymbol{y}) = \Delta \tilde{u}(\boldsymbol{y})$ *for* $\boldsymbol{y} \in \mathcal{S}$*.*

We now prove the principles for two-dimensional surfaces embedded in $\mathbb{R}^3$ using the definitions introduced in Section 1.2. For a scalar function u, since its closest point extension $\tilde{u}$ is constant along the normals to $\mathcal{S}$, we have $\boldsymbol{n} \cdot \nabla \tilde{u} = 0$. Therefore, from (1.2.1) the Gradient principle holds, and from (1.2.8) the Laplacian principle holds. For a vector function $\boldsymbol{v}$, since each component of its closest point extension $\tilde{\boldsymbol{v}}$ is constant along the normals to $\mathcal{S}$, we have $(\nabla \tilde{\boldsymbol{v}})\boldsymbol{n} = \boldsymbol{0}$ where $\nabla \tilde{\boldsymbol{v}}$ is the Jacobian matrix of $\tilde{\boldsymbol{v}}$. From (1.2.6) the Divergence principle holds.

The proof can be extended to general manifolds with arbitrary dimension and codimension and this is done by [65]. The work [65] also discusses and proves cases for higher order differential operators (e.g., the biharmonic operator).

1.3.3 Ruuth–Merriman Closest Point Method

After introducing the basic ingredients of the Closest Point Method, we review the Ruuth–Merriman approach [79] by considering the diffusion equation on surface $\mathcal{S}$,

$$u_t(t, \boldsymbol{y}) = \Delta_{\mathcal{S}} u(t, \boldsymbol{y}), \tag{1.3.1a}$$

$$u(0, \boldsymbol{y}) = u_0(\boldsymbol{y}), \tag{1.3.1b}$$

where $\boldsymbol{y} \in \mathcal{S}$ and $t \geq 0$.

Using the Laplacian principle, we extend the surface PDE (1.3.1) to the following embedding equation,

$$\tilde{u}_t(t, \boldsymbol{x}) = \Delta \tilde{u}(t, \mathrm{cp}(\boldsymbol{x})), \tag{1.3.2a}$$

$$\tilde{u}(0, \boldsymbol{x}) = u_0(\mathrm{cp}(\boldsymbol{x})), \tag{1.3.2b}$$

where $\boldsymbol{x} \in B(\mathcal{S})$ and $t \geq 0$.

According to [79], the solutions of (1.3.1) and (1.3.2) agree *at the surface*. In more detail, if $u(t, \boldsymbol{y})$ is a solution of (1.3.1) for $\boldsymbol{y} \in \mathcal{S}$ and $\tilde{u}(t, \boldsymbol{x})$ is a solution of (1.3.2) for $\boldsymbol{x} \in B(\mathcal{S})$, then $u(t, \boldsymbol{y}) = \tilde{u}(t, \boldsymbol{y})$ for $t \geq 0$ and points on the surface $\boldsymbol{y} \in \mathcal{S}$. Therefore, it appears that we only need to solve the embedding equation

(1.3.2). However, directly discretizing (1.3.2) yields an unstable scheme as discussed in [62, 85]. This is because the discretization matrix corresponding to $\Delta\tilde{u}(t, \mathrm{cp}(\boldsymbol{x}))$ has eigenvalues with positive real parts. Various approaches [79, 62, 85] have been adopted to overcome this unstability issue. One of them is the Ruuth–Merrimann approach which evolves (1.3.2) explicitly in a stable manner, as we shall review next.

Starting from the initial conditions $\tilde{u}(0, \boldsymbol{x}) = u_0(\mathrm{cp}(\boldsymbol{x}))$, which correspond to a closest point extension of the surface data u_0, we have

$$\Delta\tilde{u}(0, \mathrm{cp}(\boldsymbol{x})) = \Delta\tilde{u}(0, \boldsymbol{x}),$$

which means at $t = 0$, we can evolve equation (1.3.2a) simply by

$$\tilde{u}_t(t, \boldsymbol{x}) = \Delta\tilde{u}(t, \boldsymbol{x}). \tag{1.3.3}$$

However, after $t > 0$, as $\tilde{u}(t, \boldsymbol{x})$ is evolved by (1.3.3), it no longer equals $\tilde{u}(t, \mathrm{cp}(\boldsymbol{x}))$; so we need to do the closest point extension $\tilde{u}(t, \boldsymbol{x}) := \tilde{u}(t, \mathrm{cp}(\boldsymbol{x}))$ again if we still want to evolve equation (1.3.2a) simply by (1.3.3).

Discretizing the above idea along the time direction by the forward Euler scheme leads to the Ruuth–Merriman approach [79]. Starting from a closest point extension of the initial data, at each time step we perform the following two steps.

Algorithm 1 (Ruuth–Merriman) *1. Perform a forward Euler time step with step size Δt in the embedding space:*

$$w^{n+1}(\boldsymbol{x}) = \tilde{u}^n(\boldsymbol{x}) + \Delta t \cdot \Delta\tilde{u}^n(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R}^d;$$

2. Perform a closest point extension for each point in the embedding space:

$$\tilde{u}^{n+1}(\boldsymbol{x}) = w^{n+1}(\mathrm{cp}(\boldsymbol{x})), \qquad \boldsymbol{x} \in \mathbb{R}^d.$$

Now we still need to discretize in the spatial direction in order to get a complete numerical scheme. The spatial discretization contains two parts: one for the evolving step, the other for the extension step. For the evolving step, we use standard finite difference schemes (e.g., the $\frac{1}{\Delta x^2}\{-4, 1, 1, 1, 1\}$ rule for the Laplacian in 2-D), and this requires a *difference stencil* for each grid point. For the extension step, we need to assign the value of each grid point to be the value of its closest point; since the closest point is usually not a grid point, its value is obtained through interpolation of the surrounding grid points. We call these grid points surrounding the closest point the *interpolation stencil*. Figure 1.1 shows examples of interpolation stencils needed to obtain the values of u at several closest points.

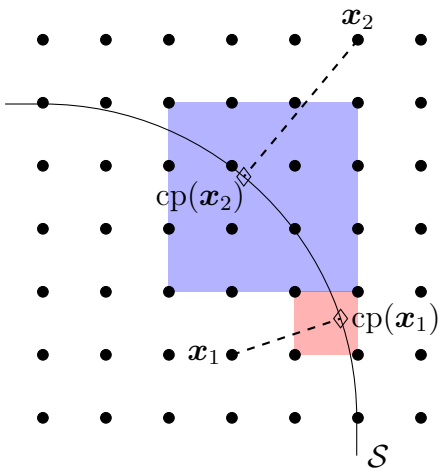


Figure 1.1: Closest point extensions for grid points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. For example, bi-linear interpolation (smaller interpolation stencil shaded in red) is used to approximate $u(\mathrm{cp}(\boldsymbol{x}_1))$, while bi-cubic interpolation (larger interpolation stencil shaded in blue) is used to approximate $u(\mathrm{cp}(\boldsymbol{x}_2))$.

As pointed out in [79], the order of the interpolation for the closest point extension should be high enough to yield a consistent scheme. Thus for a p-th order difference scheme and a problem involving up to order-q derivatives the interpolation order should be $p+q$ (or higher) [79]. For example, if we want to solve the heat equation on surfaces with smooth solutions by the second-order centered finite difference scheme for the Discrete Laplacian, then at least 4th-order (e.g. cubic polynomial) interpolation should be used to ensure that the final difference scheme is second-order.[1] If the underlying PDE has smooth solutions, polynomial interpolations fulfill the consistency conditions; and in particular we shall use tensor product *Barycentric Lagrange interpolation* [8], which is a fast and stable form of Lagrange polynomial interpolation. In this thesis we focus on elliptic PDEs with smooth solutions, and use the barycentric Lagrange interpolation all the time. For PDEs with non-smooth solutions, WENO interpolation [41] can be used; see [61] where the authors solve the level set equations on surfaces.

In principle one can perform Algorithm 1 in the whole embedding space, but for theoretical and practical reasons we'd like to do it in a narrow band surrounding the surface. Following [62], the computational band contains two discrete sets of points. The first list $B_{interp} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m\}$ contains every grid point which can appear in the interpolation stencil for some point on the surface $\mathcal{S}$[2]. The second list $B_{edge} = \{\boldsymbol{x}_{m+1}, \boldsymbol{x}_{m+2}, ..., \boldsymbol{x}_{m+m_e}\}$ contains points which are not in B_{interp} but appear in the difference stencil of some point in B_{interp}. Figure 1.2 shows an example of the two lists B_{interp} and B_{edge} where the curve $\mathcal{S}$ is a circle embedded in $\mathbb{R}^2$. Algorithms

[1] We shall give detailed truncation error analysis for the shifted Poisson's equation in Chapter 2.

[2] As pointed out in [62], a practical B_{interp} is defined to be all points which can appear in some interpolation stencil of $\mathrm{cp}(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i \in B_{interp} \cup B_{edge}$.

for the construction of these lists can be found in [62], and implementations can be obtained from **cpSurf** [28] which is a surface computing software package using the Closest Point Method.

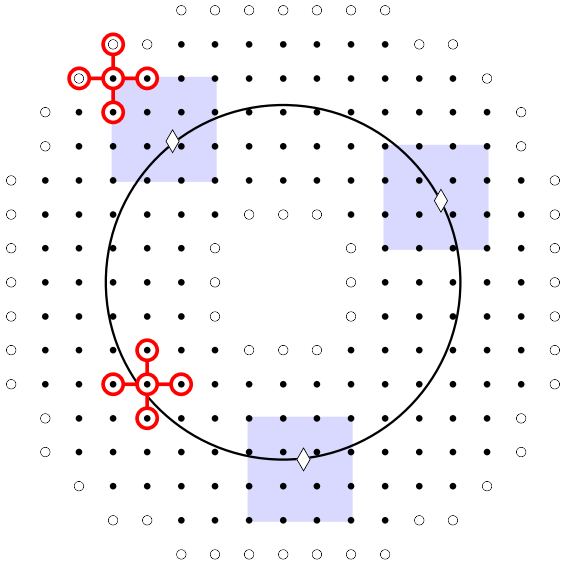


Figure 1.2: (Figure taken from [62] Section 2.2, with permission of the authors) Examples of the lists of grid points B_{interp} (indicated by $\bullet$) and B_{edge} (indicated by $\circ$) where the curve $\mathcal{S}$ is a circle. The interpolation stencil is a 4×4 grid (arising, for example, from using degree 3 barycentric Lagrange interpolation) and the shaded regions illustrate the use of this stencil at the three points on the circle indicated by $\diamond$. Five-point difference stencils are shown for two example points in B_{interp}, in one case illustrating the use of points in B_{edge}.

With the above definition of B_{interp} and B_{edge}, Algorithm 1 can be implemented as follows. In the beginning, we assign the value of $\tilde{u}$ at each point in $B_{interp} \cup B_{edge}$ as the value at the corresponding closest point (either by evaluating the exact initial surface data or through interpolation). Then we alternate between the following two steps:

1. $w(B_{interp}) = \tilde{u}(B_{interp}) + \Delta t \cdot \Delta_h(\tilde{u}(B_{interp} \cup B_{edge}))$, where Δ_h is the discrete Laplacian operator evaluating at points in B_{interp} using values of $\tilde{u}$ at points in $B_{interp} \cup B_{edge}$;

2. update $\tilde{u}(B_{interp} \cup B_{edge})$ by interpolating values of $w(B_{interp})$.

1.3.4 Development of the Closest Point Method

The Ruuth–Merriman approach works for a wide range of time-dependent surface PDEs (see [79, 61]). However, this approach uses explicit time-stepping and its modification to an implicit scheme is not straightforward. Motivated by deriving implicit time-stepping schemes to solve stiff PDEs, [62] introduces a stabilized discretization for the embedding counterpart of the Laplace–Beltrami operator $\Delta\tilde{u}(\text{cp}(\boldsymbol{x}))$. The stabilized operator in the embedding space turns out to be $\Delta\tilde{u}(\text{cp}(\boldsymbol{x})) - \gamma\big(\tilde{u}(\boldsymbol{x}) - \tilde{u}(\text{cp}(\boldsymbol{x}))\big)$ (with a particular parameter γ). Intuitively, $-\gamma\big(\tilde{u}(\boldsymbol{x}) - \tilde{u}(\text{cp}(\boldsymbol{x}))\big)$ could be

viewed as a penalty term that keeps the solution close to be constant along the normals. However the approach in [62] does not generate a solution constant along the normals because $\Delta\tilde{u}(\text{cp}(\boldsymbol{x}))$ is not constant along the normals[3]. This makes analysis for both the continuous equation and the numerical scheme difficult.

To overcome this problem, [85] proposes a new formulation of the embedding equation with a solution that is constant along the normals. Taking the Laplace–Beltrami operator as an example, the embedding operator is now set to be $[\Delta\tilde{u}](\text{cp}(\boldsymbol{x})) - \gamma\big(\tilde{u}(\boldsymbol{x}) - \tilde{u}(\text{cp}(\boldsymbol{x}))\big)$. Here the brackets $[\cdot]$ means that we first perform the Cartesian Laplacian Δ on $\tilde{u}$, and then evaluate $\Delta\tilde{u}$ at the closest point of x on the surface. For general time-dependent surface PDEs, this new formulation actually ensures that the solution of the embedding equation is constant along the normals to the surface, and leads to a method-of-lines (MOLs) approach to solve surface PDEs.

Meanwhile, similar ideas are adopted by the current author in [16] to solve for the first time elliptic PDEs on surfaces via the Closest Point Method. In fact, the key contributions of this thesis are based on [16]. We shall outline these contributions next.

1.4 Outline of thesis contributions

The remainder of the thesis presents new contributions to the Closest Point Method to solve elliptic PDEs on general surfaces/domains (Chapters 2–4) and coupled bulk-surface PDEs (Chapter 5).

In Chapter 2, we formulate a new continuous embedding equation for elliptic equations on surfaces without boundaries. The embedding equation is then discretized and analyzed for the shifted Poisson's equation on surfaces without boundaries. Theoretical 2nd-order convergence results are obtained when the embedding space is $\mathbb{R}^2$. Numerical results show a 2nd-order convergence for both $\mathbb{R}^2$ and $\mathbb{R}^3$ embedding spaces.

In Chapter 3, we consider solutions of elliptic equations on surfaces and domains with boundaries. The "ghost" points approach is adopted to handle boundary conditions. A new method to represent values of ghost points by values of interior grid points are introduced so that Dirichlet, Neumann and Robin boundary conditions can be handled in a systematic way. Again numerical convergence studies show a 2nd-order accuracy of the method.

[3] We shall discuss this issue in more details in the next chapter.

In Chapter 4, we develop a novel geometric multigrid solver based on the closest point representation of the surface. The multigrid algorithm is initially designed for surfaces without boundaries. It is then modified to work for domains and surfaces with boundaries augmented by the ghost point ideas presented in Chapter 3. Numerical examples indicate that the convergence rates of the multigrid algorithm almost stay the same as we fix the meshsize of the coarsest grid and increase the number of grid levels. This behaviour is considered optimal for multigrid schemes.

In Chapter 5, we combine the methods of the previous three chapters to solve coupled bulk-surface PDEs. A new matrix formulation of the discretization for the coupled bulk-surface elliptic equations is formed and a multigrid solver is also constructed. Both an alternating forward Euler time-stepping scheme and a method-of-lines approach are proposed to solve coupled bulk-surface diffusion problems. As biological applications, a bulk Fisher-KPP equation coupled with a surface diffusion equation and coupled bulk-surface reaction-diffusion equations are solved.

Finally, we conclude the work of this thesis in Chapter 6.

Chapter 2

Surface Elliptic Equations

This chapter focuses on the solution of elliptic PDEs on surfaces without boundaries via the Closest Point Method. In Section 2.1 we formulate the continuous embedding equation for the surface elliptic PDE. In Section 2.2 we discretize the embedding equation for the surface Poisson's equation and analyze the truncation error of the numerical scheme. In Section 2.3 we analyze the stability of the difference scheme and prove convergence for the shifted Poisson's Equation on curves without boundaries embedded in $\mathbb{R}^2$. In Section 2.4 we slightly modify the scheme so that we are able to prove stability and convergence for the same equation on curves or surfaces without boundaries embedded in $\mathbb{R}^3$. We also provide some numerical examples for the shifted Poisson equation on various curves and surfaces to validate our analysis in Section 2.5.

Most contents of this chapter are based on [16] written by the current author. The new part in this thesis is the stability and convergence proof for three-dimensional embedding space in Section 2.4.

2.1 Continuous embedding equation

2.1.1 Embedding equation for the surface Poisson problem

For ease of derivation, we first focus on the shifted Poisson's equation on a smooth surface $\mathcal{S}$ without boundary:

$$-\Delta_{\mathcal{S}} u(\boldsymbol{y}) + cu(\boldsymbol{y}) = f(\boldsymbol{y}), \qquad \boldsymbol{y} \in \mathcal{S}, \tag{2.1.1}$$

where $f(\boldsymbol{y})$ is some known function defined on the surface $\mathcal{S}$, and we assume c to be a positive constant for simplicity[1]. According to [4, Theorem 4.18], (2.1.1) has a unique solution $u(\boldsymbol{y}) \in \mathcal{C}^{k+2}$ if $f(\boldsymbol{y}) \in \mathcal{C}^k$.

[1] Our derivations and proofs also work for the case where $c(\boldsymbol{y})$ is a surface function satisfying $c(\boldsymbol{y}) \geq C > 0$, where C is a positive constant.

Because $\mathcal{S}$ is smooth, there is a tubular neighborhood $B(\mathcal{S})$ surrounding $\mathcal{S}$ [40, 65] such that the closest point function cp (Definition 1.3.1) is well defined in $B(\mathcal{S})$. We want to define an embedding equation in $B(\mathcal{S})$, whose solution agrees with the solution of the surface PDE (2.1.1) on $\mathcal{S}$.

We construct the right-hand side function $\tilde{f}$ in $B(\mathcal{S})$ via the closest point extension,

$$\tilde{f}(\boldsymbol{x}) = f(\mathrm{cp}(\boldsymbol{x})), \qquad \boldsymbol{x} \in B(\mathcal{S}).$$

From this construction of $\tilde{f}$ and the idempotence of the closest point extension, we have that

$$\tilde{f}(\boldsymbol{x}) = \tilde{f}(\mathrm{cp}(\boldsymbol{x})), \qquad \boldsymbol{x} \in B(\mathcal{S}). \tag{2.1.2}$$

We also want to find a solution $\tilde{u}$ in the band $B(\mathcal{S})$ which is constant along the normal direction to the surface, i.e., we will need to impose this condition on $\tilde{u}$,

$$\tilde{u}(\boldsymbol{x}) = \tilde{u}(\mathrm{cp}(\boldsymbol{x})), \qquad \boldsymbol{x} \in B(\mathcal{S}). \tag{2.1.3}$$

Since $\tilde{u}$ is constant along the normal direction to the surface, the Cartesian Laplacian of $\tilde{u}$ equals the Laplace–Beltrami function of $\tilde{u}$ on the surface [79], and thus

$$-\Delta\tilde{u}(\boldsymbol{y}) + c\tilde{u}(\boldsymbol{y}) = \tilde{f}(\boldsymbol{y}), \qquad \boldsymbol{y} \in \mathcal{S}.$$

Replacing $\boldsymbol{y}$ with $\mathrm{cp}(\boldsymbol{x})$, we have

$$-[\Delta\tilde{u}](\mathrm{cp}(\boldsymbol{x})) + c\tilde{u}(\mathrm{cp}(\boldsymbol{x})) = \tilde{f}(\mathrm{cp}(\boldsymbol{x})), \qquad \boldsymbol{x} \in \mathrm{B}(\mathcal{S}).$$

Here the square bracket $[\cdot]$ surrounding $\Delta\tilde{u}$ means that we first calculate the Cartesian Laplacian of $\tilde{u}(\boldsymbol{x})$, and then evaluate at the closest point of $\boldsymbol{x}$ on the surface. By assumption, both $\tilde{f}$ and $\tilde{u}$ are closest point extensions ((2.1.2) and (2.1.3)), so we have

$$-[\Delta\tilde{u}](\mathrm{cp}(\boldsymbol{x})) + c\tilde{u}(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x}), \tag{2.1.4a}$$

$$\text{subject to} \quad \tilde{u}(\boldsymbol{x}) = \tilde{u}(\mathrm{cp}(\boldsymbol{x})), \quad \boldsymbol{x} \in B(\mathcal{S}). \tag{2.1.4b}$$

Equation (2.1.4a) ensures that the embedding equation agrees with the original surface PDE on the surface, and we will call it the *consistency condition*. Equation (2.1.4b) forces $\tilde{u}$ to be constant along the normal direction to the surface (so that we can replace the surface Laplacian with the Cartesian Laplacian), and we will call it the *side condition*. Combining the two equations (2.1.4a) and (2.1.4b), we get the embedding equation,

$$-[\Delta\tilde{u}](\mathrm{cp}(\boldsymbol{x})) + c\tilde{u}(\boldsymbol{x}) + \gamma(\tilde{u}(\boldsymbol{x}) - \tilde{u}(\mathrm{cp}(\boldsymbol{x}))) = \tilde{f}(\boldsymbol{x}), \quad \boldsymbol{x} \in B(\mathcal{S}), \tag{2.1.5}$$

where γ is a parameter. As we shall show next, the choice of γ does not have much impact on the continuous equation (2.1.5); but later in our numerical scheme, γ plays a role in balancing the side and consistency conditions.

So far we have shown that the system (2.1.4) leads to equation (2.1.5). Following the ideas of [85], we are going to show that in the continuous setting and if $\gamma \neq -c$, equation (2.1.5) implies *both* equations (2.1.4a) and (2.1.4b). We rearrange equation (2.1.5) as

$$-[\Delta\tilde{u} + \gamma\tilde{u}](\mathrm{cp}(\boldsymbol{x})) = \tilde{f}(\boldsymbol{x}) - (\gamma + c)\tilde{u}(\boldsymbol{x}), \quad \boldsymbol{x} \in B(\mathcal{S}). \tag{2.1.6}$$

Here the square bracket $[\cdot]$ surrounding $\Delta\tilde{u} + \gamma\tilde{u}$ means that we first calculate the Cartesian Laplacian of $\tilde{u}(\boldsymbol{x})$ and add the result to $\gamma\tilde{u}(\boldsymbol{x})$, then evaluate the function $\Delta\tilde{u}+\gamma\tilde{u}$ at the closest point of $\boldsymbol{x}$ on the surface. The left-hand side of equation (2.1.6) is constant along the direction normal to $\mathcal{S}$, so the right-hand side $\tilde{f}(\boldsymbol{x}) - (\gamma + c)\tilde{u}(\boldsymbol{x})$ also has to be constant along the normal direction to $\mathcal{S}$, i.e.,

$$\tilde{f}(\boldsymbol{x}) - (\gamma + c)\tilde{u}(\boldsymbol{x}) = \tilde{f}(\mathrm{cp}(\boldsymbol{x})) - (\gamma + c)\tilde{u}(\mathrm{cp}(\boldsymbol{x})).$$

Since $\tilde{f}(\boldsymbol{x}) = \tilde{f}(\mathrm{cp}(\boldsymbol{x}))$, if $\gamma \neq -c$, we have that $\tilde{u}(\boldsymbol{x}) = \tilde{u}(\mathrm{cp}(\boldsymbol{x}))$, i.e., equation (2.1.4b) holds. Substituting (2.1.4b) back into equation (2.1.5), we have that equation (2.1.4a) holds. In other words, from equation (2.1.5) with $\gamma \neq -c$, we derive that both equations (2.1.4a) and (2.1.4b) hold.

In summary, our logic is as follows: in order to solve the original surface PDE (2.1.1), we seek $\tilde{u}(\boldsymbol{x})$ in $B(\mathcal{S})$ satisfying both the consistency condition (2.1.4a) and the side condition (2.1.4b), which are in turn equivalent to equation (2.1.5) (provided $\gamma \neq -c$). This equivalence can be summarized as the following theorem.

Theorem 2.1.1 *1. If $\tilde{u} \in B(\mathcal{S})$ is a solution of (2.1.4), then $\tilde{u}$ is a solution of (2.1.5).*

2. If $\tilde{u} \in B(\mathcal{S})$ is a solution of (2.1.5), and $\gamma \neq -c$, then $\tilde{u}$ is a solution of (2.1.4).

Another natural question is whether the embedding equation (2.1.5) has a unique solution that agrees with the solution of the original surface PDE (2.1.1). This is confirmed by the following theorem.

Theorem 2.1.2 *Assuming that the surface PDE (2.1.1) has a unique solution u, then the embedding equation (2.1.5) has a unique solution $\tilde{u}$ satisfying $\tilde{u}|_{\mathcal{S}} = u$ provided that $\gamma \neq -c$.*

Proof. Denote the unique solution for the surface PDE (2.1.1) as u, and define $\tilde{u}$ to be the closest point extension of u. Then it is straightforward to verify that $\tilde{u}$ is a solution of the embedding equation (2.1.5). Now we only need to prove that $\tilde{u}$ is the unique solution of (2.1.5). Suppose there are two different solutions $\tilde{u}_1$ and $\tilde{u}_2$ for (2.1.5). Then from part 2 of Theorem 2.1.1, $\tilde{u}_i(\text{cp}(\boldsymbol{x})) = \tilde{u}_i(\boldsymbol{x})$ $(i = 1, 2)$ which means that both $\tilde{u}_1$ and $\tilde{u}_2$ are constant along the normal directions to $\mathcal{S}$, and $\Delta\tilde{u}_i(\boldsymbol{y}) + c\tilde{u}_i(\boldsymbol{y}) = f(\boldsymbol{y})$ $(\boldsymbol{y} \in \mathcal{S}, i = 1, 2)$ which means that both $\tilde{u}_1|_{\mathcal{S}}$ and $\tilde{u}_2|_{\mathcal{S}}$ are solutions of the surface PDE (2.1.1). Since $\tilde{u}_1 \neq \tilde{u}_2$ and the $\tilde{u}_i$'s are constant along the normals, $\tilde{u}_1|_{\mathcal{S}} \neq \tilde{u}_2|_{\mathcal{S}}$. This contradicts the fact that (2.1.1) has a unique solution.

Remark 2.1.3 *From the above two theorems the parameter γ does not have much impact on the solution of the continuous embedding equation (2.1.5). Actually we can set $\gamma = 0$ reducing (2.1.5) to $-[\Delta\tilde{u}](cp(\boldsymbol{x})) + c\tilde{u}(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x})$. The reduced equation will still give us the unique solution which is the closest point extension of the solution of the surface PDE (2.1.1).*

Remark 2.1.4 *We introduce γ into the formation of (2.1.5) to ensure numerical stability and convergence. The choice of γ typically depends on numerical parameters. For example, we choose $\gamma = O(\frac{1}{\Delta x^2})$ for Laplacian operators where Δx is the mesh size.*

2.1.2 More general elliptic operators

The above ideas extend naturally to more general linear elliptic surface PDEs,

$$-\nabla_{\mathcal{S}} \cdot (\boldsymbol{A}(\boldsymbol{y})\nabla_{\mathcal{S}} u(\boldsymbol{y})) + c(\boldsymbol{y})u(\boldsymbol{y}) = f(\boldsymbol{y}), \quad \boldsymbol{y} \in \mathcal{S}, \tag{2.1.7}$$

where $\boldsymbol{A}(\boldsymbol{y})$ is a symmetric and positive semi-definite matrix.

Again we construct $\tilde{f}(\boldsymbol{x}) = f(\text{cp}(\boldsymbol{x}))$, $\boldsymbol{x} \in B(\mathcal{S})$, and would like to enforce the side condition $\tilde{u}(\boldsymbol{x}) = \tilde{u}(\text{cp}(\boldsymbol{x}))$ for the solution $\tilde{u}$ defined in $B(\mathcal{S})$. Using the generalized closest point principle ([65, Theorem 4.3]), the following equations agree with surface PDE (2.1.7) on the surface,

$$-[\nabla \cdot (\boldsymbol{A}(\text{cp}(\boldsymbol{x}))\nabla\tilde{u})](\text{cp}(\boldsymbol{x})) + c\tilde{u}(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x}), \tag{2.1.8a}$$

$$\text{subject to} \quad \tilde{u}(\boldsymbol{x}) = \tilde{u}(\text{cp}(\boldsymbol{x})), \quad \boldsymbol{x} \in B(\mathcal{S}). \tag{2.1.8b}$$

Here the square bracket $[\cdot]$ surrounding $\nabla \cdot (\boldsymbol{A}(\text{cp}(\boldsymbol{x}))\nabla\tilde{u})$ means that we first perform the gradient on $\tilde{u}$, multiply it by $\boldsymbol{A}(\text{cp}(\boldsymbol{x}))$, then take the divergence, and finally

evaluate the results at $\text{cp}(\boldsymbol{x})$. Adding (2.1.8a) and (2.1.8b) together, we have

$$-[\nabla\cdot(\boldsymbol{A}(\text{cp}(\boldsymbol{x}))\nabla\tilde{u})](\text{cp}(\boldsymbol{x}))+c\tilde{u}(\boldsymbol{x})+\gamma(\tilde{u}(\boldsymbol{x})-\tilde{u}(\text{cp}(\boldsymbol{x})))=\tilde{f}(\boldsymbol{x}),\quad \boldsymbol{x}\in B(\mathcal{S}), \tag{2.1.9}$$

Like for the Poisson case in Section 2.1.1, we have the following two theorems.

Theorem 2.1.5 *1. If $\tilde{u}\in B(\mathcal{S})$ is a solution of (2.1.8), then $\tilde{u}$ is a solution of (2.1.9).*

2. If $\tilde{u}\in B(\mathcal{S})$ is a solution of (2.1.9), and $\gamma\neq -c$, then $\tilde{u}$ is a solution of (2.1.8).

Theorem 2.1.6 *Assuming that the surface PDE (2.1.7) has a unique solution u, then the embedding equation (2.1.9) has a unique solution $\tilde{u}$ satisfying $\tilde{u}|_{\mathcal{S}}=u$ provided that $\gamma\neq -c$.*

The proofs of the above two theorems are similar to the Poisson case in Section 2.1.1. The only difference is that we replace $\Delta\tilde{u}$ with $\nabla\cdot(\boldsymbol{A}(\text{cp}(\boldsymbol{x}))\nabla\tilde{u})$.

We focus our derivation and analysis on the surface Poisson problem via the embedding equation (2.1.5). In most cases it is straightforward to extend our approach to more general elliptic operators (e.g., we investigate an example with variable diffusion coefficients in Section 4.1.5.6).

2.1.3 A comparison with the Macdonald–Brandman–Ruuth approach

If we apply the approach in [62, 59] to the Laplace–Beltrami operator in (2.1.1), and extend the right hand side via the closest point extension, then we get the following continuous embedding equation,

$$-\Delta(\tilde{u}(\text{cp}(\boldsymbol{x})))+c\tilde{u}(\boldsymbol{x})+\gamma(\tilde{u}(\boldsymbol{x})-\tilde{u}(\text{cp}(\boldsymbol{x})))=\tilde{f}(\boldsymbol{x}),\quad \boldsymbol{x}\in B(\mathcal{S}). \tag{2.1.10}$$

Note that the only difference between (2.1.5) and (2.1.10) is that in (2.1.5) we first perform the Cartesian Laplacian operator on $\tilde{u}$ and then evaluate at $\text{cp}(\boldsymbol{x})$, while in (2.1.10) we first do the closest point extension of $\tilde{u}$ and then perform the Laplacian. One can show (similar to [59]), that equation (2.1.10) agrees with the original surface PDE (2.1.1) on the surface, and it has a unique solution provided that $\gamma\neq -c$. We can re-arrange (2.1.10), solving for $\tilde{u}(\boldsymbol{x})$, to obtain

$$\tilde{u}(\boldsymbol{x})=\frac{1}{c+\gamma}\big(\tilde{f}(\boldsymbol{x})+\gamma\tilde{u}(\text{cp}(\boldsymbol{x}))+\Delta(\tilde{u}(\text{cp}(\boldsymbol{x})))\big).$$

From this, $\tilde{u}$ is *not necessarily constant along the normals to* $\mathcal{S}$ because $\Delta(\tilde{u}(\text{cp}(\boldsymbol{x})))$ is not constant along the normals in general. This makes analysis of the method difficult. Furthermore, it would also be difficult to make $\tilde{u}$ constant along the normals to $\mathcal{S}$ by modifying the right-hand side (e.g., via some other method of extension).

In contrast, our new embedding equation (2.1.5) not only agrees with the original surface PDE (2.1.1) on the surface, but also has a solution $\tilde{u}$ which is constant in the normal direction to $\mathcal{S}$. The latter property is crucial to the convergence analysis for our numerical discretization in the next section. In addition to these advantages for analysis, (2.1.5) can lead to a consistent discretization with a sparser coefficient matrix than the one resulting from (2.1.10); we discuss this further in Remark 2.2.2. Furthermore, as we shall see in Chapter 4, the construction of our multigrid solver relies on the property of $\tilde{u}$ being constant along the normal direction to $\mathcal{S}$. Finally, as noted in Section 2.1.2, our new approach extends naturally to non-constant-coefficient problems.

2.2 Numerical discretization

Similar to [62], we derive a matrix formulation of a finite difference scheme for the embedding equation (2.1.5).

2.2.1 Construction of the difference scheme

In Section 1.3.3 we reviewed the definition of the two lists of discrete points in [62] as shown by Figure 1.2. The first list $B_{interp} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m\}$ contains every grid point which can appear in the interpolation stencil for some point on the surface $\mathcal{S}$. The second one $B_{edge} = \{\boldsymbol{x}_{m+1}, \boldsymbol{x}_{m+2}, ..., \boldsymbol{x}_{m+m_e}\}$ contains points which are not in B_{interp} but appear in the difference stencil of some point in B_{interp}. Let B be a set of grid points such that $B_{interp} \cup B_{edge} \subseteq B$. Let $n = \|B\|$ be the number of points in B. We introduce the vector $\boldsymbol{u}^h \in \mathbb{R}^n$ with entries $u_i \approx \tilde{u}(\boldsymbol{x}_i)$ for each grid point $\boldsymbol{x}_i \in B$, and similarly, $\boldsymbol{f}^h \in \mathbb{R}^n$ with entries $f_i = \tilde{f}(\boldsymbol{x}_i)$.

There are two closest point extensions in (2.1.5); we can use different degrees of interpolation for the closest point evaluations of $\Delta\tilde{u}$ and $\tilde{u}$. If we use degree p interpolation for $[\Delta\tilde{u}](\text{cp}(\boldsymbol{x}))$, and degree q interpolation for $\tilde{u}(\text{cp}(\boldsymbol{x}))$, then for each grid point $\boldsymbol{x}_i \in B$, we have the following discretized equation,

$$-\sum_{\boldsymbol{x}_j \in \text{I}_p(\text{cp}(\boldsymbol{x}_i))} \omega_j \left(\sum_{\boldsymbol{x}_k \in \text{Diff}(\boldsymbol{x}_j)} l_k u_k \right) + c u_i + \gamma \left(u_i - \sum_{\boldsymbol{x}_j \in \text{I}_q(\text{cp}(\boldsymbol{x}_i))} \bar{\omega}_j u_j \right) = f_i. \quad (2.2.1)$$

Here $\mathrm{I}_p(\mathrm{cp}(\boldsymbol{x}_i))$ is the degree p interpolation stencil for point $\mathrm{cp}(\boldsymbol{x}_i)$, $\mathrm{Diff}(\boldsymbol{x}_j)$ is the difference stencil for point x_j, and $\mathrm{I}_q(\mathrm{cp}(\boldsymbol{x}_i))$ is the degree q interpolation stencil for point $\mathrm{cp}(\boldsymbol{x}_i)$; ω_j are the weights of the interpolation scheme for evaluating $[\Delta\tilde{u}](\mathrm{cp}(\boldsymbol{x}))$, $\bar{\omega}_j$ are the weights of the interpolation scheme for evaluating $\tilde{u}(\mathrm{cp}(\boldsymbol{x}))$, and l_k are the weights for the difference scheme approximating the Laplacian operator.

Since the interpolation and difference schemes are linear combinations of the entries of $\boldsymbol{u}^h$, we can write equation (2.2.1) in matrix form. Denote the discrete solution by a column vector $\boldsymbol{u}^h$, the discrete right-hand side function by a vector $\boldsymbol{f}^h$, the identity matrix by $\boldsymbol{I}^h$, the closest point interpolation matrices of degrees p and q by $\boldsymbol{E}_p^h$ and $\boldsymbol{E}_q^h$, and the discrete Laplace matrix by $\boldsymbol{L}^h$. The matrix form of the difference scheme for the embedding equation (2.1.5) is

$$-\boldsymbol{E}_p^h\boldsymbol{L}^h\boldsymbol{u}^h + c\boldsymbol{u}^h + \gamma(\boldsymbol{I}^h - \boldsymbol{E}_q^h)\boldsymbol{u}^h = \boldsymbol{f}^h. \tag{2.2.2}$$

We recall that the points in B_{edge} do not belong to any of the interpolation stencils, which means that the columns in $\boldsymbol{E}_p^h$ corresponding to points in B_{edge} are all zero columns, so the rows in $\boldsymbol{L}^h$ corresponding to points in B_{edge} can vary arbitrarily while $\boldsymbol{E}_p^h\boldsymbol{L}^h$ stays unchanged. Thus we see that no boundary condition for the discrete Laplace matrix is needed, since the discrete Laplacian at any points in B_{edge} does not contribute to the closest point evaluation of $\boldsymbol{L}^h\boldsymbol{u}^h$.

After discretizing the Laplacian the closest point extension operators we still need to make a choice of the parameter γ. As discussed in Section 2.1, the choice of γ does not have much impact on the continuous embedding equation (2.1.5), but it is crucial for numerical computing. According to [62], a typical choice of γ is $\frac{2d}{\Delta x^2}$, where d is the dimension of the embedding space and Δx is the grid size. In [62], this choice of γ is motivated by stabilizing the implicit time-stepping scheme for the in-surface heat equation.

The choice of $\gamma = \frac{2d}{\Delta x^2}$ is not obvious simply from (2.2.2), but it turns out to be a good choice there. In Section 2.3 we show that for the shifted Poisson's equation on a curve without boundary embedded in $\mathbb{R}^2$, the coefficient matrix corresponding to (2.2.2) is diagonally dominant by rows when we choose $\gamma = \frac{2d}{\Delta x^2}$, $p = 1$, $q = 3$, and $\boldsymbol{L}^h$ as the Laplacian matrix from the standard centered finite difference scheme. A 2nd-order convergence result naturally follows (see Theorem 2.3.1). Although there's no such clean result where the embedding space is $\mathbb{R}^3$, numerical results in Section 2.5 show that $\gamma = \frac{2d}{\Delta x^2}$ is still a good choice. In Section 2.4 we choose different γ's for different grid points to construct a convergence proof for 3-D embedding space. Although γ varies for different grid points, values taken are of size $O(\frac{1}{\Delta x^2})$.

2.2.2 Truncation error

We denote the restriction operator which evaluates the continuous function at the grid points by $\boldsymbol{R}^h$, and denote the coefficient matrix in the difference scheme (2.2.2) by $\boldsymbol{A}^h$, i.e.,

$$\boldsymbol{A}^h = c\boldsymbol{I}^h - \boldsymbol{M}^h, \quad \text{where} \quad \boldsymbol{M}^h = \boldsymbol{E}_p^h \boldsymbol{L}^h - \gamma(\boldsymbol{I}^h - \boldsymbol{E}_q^h). \tag{2.2.3}$$

Then we prove a theorem showing the form of the truncation error.

Theorem 2.2.1 *Assume that $\tilde{u} \in C^k(\mathbb{R}^d)$ is a solution of (2.1.5), where d is the dimension of the embedding space and $k \geq \max\{4, p+3, q+1\}$ (p and q are the polynomial interpolation degrees for the discrete closest point extension of $\Delta\tilde{u}$ and $\tilde{u}$ respectively). Let $\boldsymbol{L}^h$ be a second-order approximation to the Cartesian Laplacian Δ (in the ∞-norm), and $\gamma = O(\frac{1}{\Delta x^2})$. We have the following form of the truncation error*[2]*,*

$$\boldsymbol{A}^h \boldsymbol{R}^h \tilde{u}(\boldsymbol{x}) - \boldsymbol{R}^h \tilde{f}(\boldsymbol{x}) = O(\Delta x^{p+1}) + O(\Delta x^{q-1}) + O(\Delta x^2), \tag{2.2.4}$$

where $\boldsymbol{A}^h$ is as given in (2.2.3), and $\boldsymbol{R}^h$ is the restriction operator mentioned before.

Proof Let $w(\boldsymbol{x}) = \Delta\tilde{u}(\boldsymbol{x})$, $x \in B(\mathcal{S})$; then $w(\boldsymbol{x}) \in C^{k-2}(\mathbb{R}^d)$. As mentioned before, since $\tilde{u}$ is constant along directions normal to $\mathcal{S}$, $w(\boldsymbol{y}) = \tilde{f}(\boldsymbol{y})$ for any $\boldsymbol{y} \in \mathcal{S}$. We evaluate w at the closest points on $\mathcal{S}$ through degree p interpolation, which will introduce an interpolation error. Since $k \geq p+3$, $k-2 \geq p+1$, we have the following standard error estimate for degree p polynomial interpolation,

$$\boldsymbol{E}_p^h \boldsymbol{R}^h w(\boldsymbol{x}) - \boldsymbol{R}^h \tilde{f}(\boldsymbol{x}) = O(\Delta x^{p+1}). \tag{2.2.5}$$

Furthermore, since $\tilde{u} \in C^k(\mathbb{R}^d)$, $k \geq 4$, the Laplacian matrix $\boldsymbol{L}^h$ is a second-order approximation to Δ, we have that

$$\boldsymbol{R}^h w(\boldsymbol{x}) = \boldsymbol{L}^h \boldsymbol{R}^h \tilde{u}(\boldsymbol{x}) + O(\Delta x^2). \tag{2.2.6}$$

Substituting (2.2.6) into (2.2.5), we get

$$\boldsymbol{E}_p^h(\boldsymbol{L}^h \boldsymbol{R}^h \tilde{u}(\boldsymbol{x}) + O(\Delta x^2)) - \boldsymbol{R}^h \tilde{f}(\boldsymbol{x}) = O(\Delta x^{p+1}),$$

Since we are using local polynomial interpolation, $\|\boldsymbol{E}_p^h\|_\infty$ is bounded above by some constant independent of Δx, so $\boldsymbol{E}_p^h \boldsymbol{v} = O(\Delta x^2)$ for $\boldsymbol{v} = O(\Delta x^2)$. We thus get

$$\boldsymbol{E}_p^h \boldsymbol{L}^h \boldsymbol{R}^h \tilde{u}(\boldsymbol{x}) - \boldsymbol{R}^h \tilde{f}(\boldsymbol{x}) = O(\Delta x^{p+1}) + O(\Delta x^2). \tag{2.2.7}$$

[2]If $\boldsymbol{v}$ is a column vector, then by $\boldsymbol{v} = O(\Delta x^k)$ we mean that $\|\boldsymbol{v}\|_\infty \leq C\Delta x^k$, where k and C are positive constants.

Finally, we estimate the error of evaluating $\tilde{u}(\mathrm{cp}(x))$ by degree q interpolation. Because $\tilde{u} \in C^k(\mathbb{R}^d)$ and $k \geq q+1$, once again we have the standard interpolation error estimation,

$$(\boldsymbol{I}^h - \boldsymbol{E}_q^h)\boldsymbol{R}^h\tilde{u}(\boldsymbol{x}) = O(\Delta x^{q+1}). \tag{2.2.8}$$

Multiplying equation (2.2.8) by $\gamma = O(\frac{1}{\Delta x^2})$ and adding the results with equation (2.2.7), we have that equation (2.2.4) holds. ■

Remark 2.2.2 *According to (2.2.4), if we pick $p = 1$, $q = 3$, we will get a difference scheme for the embedding equation (2.1.5) with second-order truncation error in the ∞-norm. Contrasting this discretization with the method of Macdonald–Brandman–Ruuth [59, 62], we note that discretizing (2.1.10) to second-order consistency requires cubic interpolations in* both *terms. Because lower degree interpolations have smaller stencils, a discretization of (2.1.5) will yield a coefficient matrix that is sparser than that of (2.1.10). For example, for a curve in two dimensions, our new approach is roughly 34% sparser, and it is 50% for a surface in three dimensions.*

2.3 Convergence for curves without boundary in two dimensions

In this section, we aim to prove convergence of the difference scheme (2.2.2) in the restricted case of curves without boundary in two dimensions.

Theorem 2.3.1 *Suppose that we wish to solve the embedding equation (2.1.5) on a smooth curve without boundary in $\mathbb{R}^2$, and that the solution $\tilde{u}$ of the embedding equation satisfies the assumptions in Theorem 2.2.1. If in (2.2.2) we choose $\gamma = \frac{4}{\Delta x^2}$, $p = 1$, $q = 3$, and $\boldsymbol{L}^h$ as the Laplacian matrix from the standard $\frac{1}{\Delta x^2}\{-4, 1, 1, 1, 1\}$ finite difference stencil, then the difference scheme (2.2.2) is second-order convergent in the ∞-norm.*

If the embedding space is $\mathbb{R}^2$, when picking $\gamma = \frac{4}{\Delta x^2}$, $p = 1$, and $q = 3$, the coefficient matrix becomes

$$\boldsymbol{A}^h = c\boldsymbol{I}^h - \boldsymbol{M}^h, \quad \text{where} \quad \boldsymbol{M}^h = \boldsymbol{E}_1^h\boldsymbol{L}^h - \frac{4}{\Delta x^2}(\boldsymbol{I}^h - \boldsymbol{E}_3^h). \tag{2.3.1}$$

We prove Theorem 2.3.1 using the following theorem and two propositions.

Theorem 2.3.2 *If the embedding space is $\mathbb{R}^2$, then $\boldsymbol{A}^h$ defined by (2.3.1) (i.e., $\gamma = \frac{4}{\Delta x^2}$, $p = 1$, $q = 3$, and $\boldsymbol{L}^h$ is the Laplacian matrix from the standard $\frac{1}{\Delta x^2}\{-4, 1, 1, 1, 1\}$ finite difference stencil) is an M-Matrix, with positive diagonal entries $a_{kk} > 0$, non-positive off-diagonal entries $a_{kj} \leq 0$, $k \neq j$, and each row sums up to c:*

$$a_{kk} + \sum_{k \neq j} a_{kj} = c. \tag{2.3.2}$$

Proposition 2.3.3 *(Varah 1975 [84]) Assume that $\boldsymbol{A}$ is diagonally dominant by rows and set $c = \min_k(|a_{kk}| - \sum_{j \neq k} |a_{kj}|)$. Then $\|\boldsymbol{A}^{-1}\|_\infty \leq 1/c$.*

Proposition 2.3.4 *If $\|(\boldsymbol{A}^h)^{-1}\|_\infty \leq 1/c$, then the difference scheme (2.2.2) is second-order convergent in the ∞-norm provided that the scheme has second-order truncation error in the ∞-norm.*

Theorem 2.3.2 indicates that the coefficient matrix $\boldsymbol{A}^h$ satisfies the assumption of Proposition 2.3.3, which means that the claim of Proposition 2.3.3 holds: $\|(\boldsymbol{A}^h)^{-1}\|_\infty \leq 1/c$. This is in turn the assumption of Proposition 2.3.4, so the claim of Proposition 2.3.4 holds. Putting the theorem and propositions together, Theorem 2.3.1 holds.

Proof of Proposition 2.3.3 The following proof is from [84]. Since

$$\|\boldsymbol{A}^{-1}\|_\infty^{-1} = \inf_{\boldsymbol{x}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty},$$

we need only show that $c\|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{A}\boldsymbol{x}\|_\infty$ for all $\boldsymbol{x}$. Take some vector $\boldsymbol{x}$ and let $|x_k| = \|\boldsymbol{x}\|_\infty$. Then

$$\begin{gathered} 0 < c \leq |a_{kk}| - \textstyle\sum_{j \neq k} |a_{kj}| \\ 0 < c|x_k| \leq |a_{kk}x_k| - \textstyle\sum_{j \neq k} |a_{kj}||x_j| \\ \leq |a_{kk}x_k| - |\textstyle\sum_{j \neq k} a_{kj}x_j| \\ \leq |\textstyle\sum_j a_{kj}x_j| \leq \max_k |\textstyle\sum_j a_{kj}x_j| = \|\boldsymbol{A}\boldsymbol{x}\|_\infty. \end{gathered}$$

■

Proof of Proposition 2.3.4 Choosing $p = 1$, $q = 3$, the truncation error estimate (2.2.4) becomes $\boldsymbol{A}^h\boldsymbol{R}^h\tilde{u}(\boldsymbol{x}) - \boldsymbol{R}^h\tilde{f}(\boldsymbol{x}) = O(\Delta x^2)$. From the discrete linear system (2.2.2) we have that $\boldsymbol{A}^h\boldsymbol{u}^h - \boldsymbol{R}^h\tilde{f}(\boldsymbol{x}) = 0$. Subtracting these two results gives $\boldsymbol{A}^h(\boldsymbol{u}^h - \boldsymbol{R}^h\tilde{u}(\boldsymbol{x})) = O(\Delta x^2)$. Because the ∞-norm of $(\boldsymbol{A}^h)^{-1}$ is bounded by a positive constant $1/c$, which is independent of Δx, we get $\boldsymbol{u}^h - \boldsymbol{R}^h\tilde{u}(\boldsymbol{x}) = (\boldsymbol{A}^h)^{-1}O(\Delta x^2) = O(\Delta x^2)$, i.e., the difference scheme (2.2.2) is second-order convergent in the ∞-norm. ■

Proof of Theorem 2.3.2 The proof contains two steps.

Step 1 We first show that *each row of* $\boldsymbol{M}^h$ *sums up to zero.*

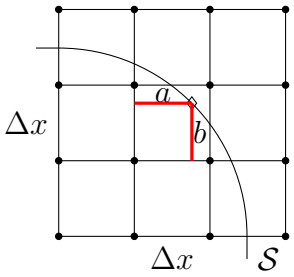
Taking all u_i to be 1 in equation (2.2.1), the left-hand side minus c becomes

$$-\sum_{\boldsymbol{x}_j \in \mathrm{I}_p(\mathrm{cp}(\boldsymbol{x}_i))} \omega_j \left(\sum_{\boldsymbol{x}_k \in \mathrm{Diff}(\boldsymbol{x}_j)} l_k \right) + \gamma \left(1 - \sum_{\boldsymbol{x}_m \in \mathrm{I}_q(\mathrm{cp}(\boldsymbol{x}_i))} \bar{\omega}_m \right). \tag{2.3.3}$$

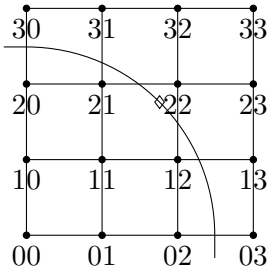
Consistency of the finite difference scheme implies that $\sum_{\boldsymbol{x}_k \in \mathrm{Diff}(\boldsymbol{x}_j)} l_k = 0$, for any $\boldsymbol{x}_j \in B_{interp}$. By the construction of the interpolation weights, $\sum_{\boldsymbol{x}_m \in \mathrm{I}_q(\mathrm{cp}(\boldsymbol{x}_i))} \bar{\omega}_m = 1$. So (2.3.3) equals 0, i.e., each row of $\boldsymbol{M}^h$ sums up to zero.

Step 2 We then prove that *all the off-diagonal entries of* $\boldsymbol{M}^h$ *are non-negative*, thus all the diagonal entries are non-positive since each row of $\boldsymbol{M}^h$ sums up to zero. For convenience (not necessity) we first rearrange $\boldsymbol{M}^h$ as

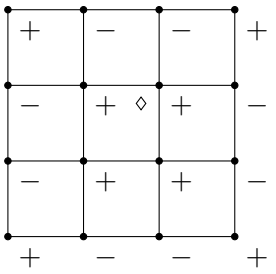
$$\boldsymbol{M}^h = \boldsymbol{E}_1^h(\boldsymbol{L}^h + \frac{4}{\Delta x^2}\boldsymbol{I}^h) + \frac{4}{\Delta x^2}(\boldsymbol{E}_3^h - \boldsymbol{E}_1^h) - \frac{4}{\Delta x^2}\boldsymbol{I}^h.$$



(a) Relative position of the closest point ⋄ in the stencil



(b) Local index for points in the stencil



(c) Signs of the weights of bi-cubic interpolation for the closest point ⋄

Figure 2.1: Information for a 4×4 stencil surrounding the closest point ⋄.

We then investigate the result of multiplying the i-th row of $\boldsymbol{M}^h$ by the column vector $\boldsymbol{u}^h$. Figure 2.1(a) shows the position of $\mathrm{cp}(\boldsymbol{x}_i)$ (indicated by ⋄). $\mathrm{cp}(\boldsymbol{x}_i)$ is in the middle block of the bi-cubic interpolation stencil, and its relative position in the stencil is as shown in Figure 2.1(a). For clarity of notation, we number the points in the bi-cubic interpolation stencil by a local index as shown in Figure 2.1(b). Denote the points in the stencil with respect to the local index by X_{lm} $(0 \le l, m \le 3)$, the corresponding values at these points by U_{lm} $(0 \le l, m \le 3)$, the bi-cubic interpolation

weights at points X_{lm} by $\bar{\omega}_{lm}$ $(0 \le l, m \le 3)$, and the bi-linear interpolation weights at points X_{nk} by ω_{nk} $(0 \le n, k \le 1)$. The i-th entry of $\boldsymbol{M}^h\boldsymbol{u}^h$ becomes

$$\begin{aligned}
&\omega_{11}(U_{10}+U_{01}+U_{12}+U_{21})/\Delta x^2\\
&+\omega_{12}(U_{11}+U_{02}+U_{13}+U_{22})/\Delta x^2\\
&+\omega_{21}(U_{20}+U_{11}+U_{22}+U_{31})/\Delta x^2\\
&+\omega_{22}(U_{21}+U_{12}+U_{23}+U_{32})/\Delta x^2\\
+4\Big(&\sum_{0\le l,m\le 3}\bar{\omega}_{lm}U_{lm}-\sum_{1\le l,m\le 2}\omega_{lm}U_{lm}\Big)/\Delta x^2-4\boldsymbol{u}_i^h/\Delta x^2. \qquad (2.3.4)
\end{aligned}$$

By the notation mentioned before, U_{lm} $(0 \le l, m \le 3)$ are just the values of $\boldsymbol{u}^h$ at the points in the bi-cubic interpolation stencil written in the local index. The point $\boldsymbol{x}_i$ might not be contained in the interpolation stencil, so we still write $\boldsymbol{u}_i^h$ in its global index. As l and m go from 1 to 3, X_{lm} might or might not be $\boldsymbol{x}_i$, depending on whether the grid point $\boldsymbol{x}_i$ is in the interpolation stencil of $\mathrm{cp}(\boldsymbol{x}_i)$ or not. In either case the negative coefficient of $\boldsymbol{u}_i^h$ only contributes to the diagonal entry in the corresponding row of the matrix $\boldsymbol{M}^h$, but the coefficient does not contribute to the off-diagonal entries of $\boldsymbol{M}^h$. So we only need to show that *all the coefficients of U_{lm}, excluding the potential contribution from the negative coefficient of $\boldsymbol{u}_i^h$, are non-negative for all* $0 \le l, m \le 3$.

If $\mathrm{cp}(\boldsymbol{x}_i)$ coincides with some grid point $\boldsymbol{x}_j$, then all the interpolation weights vanish except for the weight at $\boldsymbol{x}_j$ which is 1, so it is obvious that the above claim is true. Now we only consider cases where $\mathrm{cp}(\boldsymbol{x}_i)$ does not coincide with any grid point, which is the case illustrated in Figure 2.1.

In the rest of the proof, it might be helpful to keep in mind the signs of the bi-cubic interpolation weights $\bar{\omega}_{lm}$ $(0 \le l, m \le 3)$ for $\mathrm{cp}(\boldsymbol{x}_i)$ at each point of the stencil. Figure 2.1(c) shows the signs of the bi-cubic interpolation weights for $\mathrm{cp}(\boldsymbol{x}_i)$ (indicated by $\diamond$) which is in the middle block of the bi-cubic interpolation stencil. These signs can be obtained by straightforward computations, and they do not depend on the precise location of $\mathrm{cp}(\boldsymbol{x}_i)$. Since we interpolate in a dimension-by-dimension fashion, the weights at each of the points are obtained by multiplication of the corresponding weights in the x and y directions. For instance, if we want to compute the bi-cubic interpolation weight at the lower-left corner point $\bar{\omega}_{00}$, we first compute the weight in the x direction, $\bar{\omega}_{00}^x = -\frac{a(\Delta x-a)(2\Delta x-a)}{6\Delta x^3}$, and the weight in the y direction, $\bar{\omega}_{00}^y = -\frac{b(\Delta x-b)(2\Delta x-b)}{6\Delta x^3}$, and then we multiply the weights in the two directions to get $\bar{\omega}_{00} = \bar{\omega}_{00}^x\bar{\omega}_{00}^y$. Here we have used the superscripts x and y in the notations $\bar{\omega}_{00}^x$ and $\bar{\omega}_{00}^y$ to indicate the weights in the x and y directions, respectively. We shall use this

notation in the following proof. The bi-cubic weights at other points, and the bilinear weights, can be computed in the same way. Some of their values will be presented in the rest of the proof when necessary.

Now let us finish the proof by calculating the coefficients of U_{lm}. There are three cases for the coefficients of U_{lm} as l, m go from 1 to 3. We only need to verify that for each of the three cases, the coefficients of U_{lm} are positive.

1. Four corner points: $(l, m) = (0, 0), (0, 3), (3, 0), (3, 3)$.
   The only contribution for the row entries corresponding to these four points are the bi-cubic interpolation weights $\bar{\omega}_{00}$, $\bar{\omega}_{03}$, $\bar{\omega}_{30}$, $\bar{\omega}_{33}$, which happen to be positive in 2-D.

2. Four center points: $(l, m) = (1, 1), (1, 2), (2, 1), (2, 2)$.
   There are some positive contributions from the linear combinations of the weights in the centered difference scheme; so we only need to show at each of these four center points, the positive weight of the bi-cubic interpolation minus the bi-linear interpolation weight is larger than zero. For the sake of symmetry, we only show that $\bar{\omega}_{11} - \omega_{11} > 0$, i.e., $\bar{\omega}^x_{11}\bar{\omega}^y_{11} - \omega^x_{11}\omega^y_{11} > 0$. Since $\bar{\omega}^x_{11}$, $\bar{\omega}^y_{11}$, ω^x_{11}, and ω^y_{11} are positive, it is enough to show that

   $$\bar{\omega}^x_{11} > \omega^x_{11} \quad \text{and} \quad \bar{\omega}^y_{11} > \omega^y_{11}. \tag{2.3.5}$$

   By straightforward computation, $\bar{\omega}^x_{11} = \frac{(\Delta x+a)(\Delta x-a)(2\Delta x-a)}{2\Delta x^3}$, $\omega^x_{11} = \frac{\Delta x-a}{\Delta x}$, $\bar{\omega}^y_{11} = \frac{(\Delta x+b)(\Delta x-b)(2\Delta x-b)}{2\Delta x^3}$, $\omega^y_{11} = \frac{\Delta x-b}{\Delta x}$. It is then straightforward to verify that (2.3.5) holds for any $0 < a < \Delta x$ and $0 < b < \Delta x$.

3. Eight edge points: $(l, m) = (1, 0), (2, 0), (0, 1), (0, 2), (1, 3), (2, 3), (3, 1), (3, 2)$.
   Again by symmetry we only need to show that the row entry corresponding to $(l, m) = (1, 0)$ is positive, i.e., $\frac{\omega_{11}}{\Delta x^2} + \frac{4\bar{\omega}_{10}}{\Delta x^2} > 0$, where $\omega_{11} > 0$, but $\bar{\omega}_{10} < 0$. So we need to show that $|\omega_{11}| > 4|\bar{\omega}_{10}|$, i.e., $|\omega^x_{11}\omega^y_{11}| > 4|\bar{\omega}^x_{10}\bar{\omega}^y_{10}|$. In fact we can show that

   $$\omega^x_{11} > 6|\bar{\omega}^x_{10}|, \quad \text{and} \quad \omega^y_{11} \geq \frac{8}{9}|\bar{\omega}^y_{10}|. \tag{2.3.6}$$

   Again by straightforward computation, $\omega^x_{11} = \frac{\Delta x-a}{\Delta x}$, $|\bar{\omega}^x_{10}| = \frac{a(\Delta x-a)(2\Delta x-a)}{6\Delta x^3}$, $\omega^y_{11} = \frac{\Delta x-b}{\Delta x}$, $|\bar{\omega}^y_{10}| = \frac{(\Delta x+b)(\Delta x-b)(2\Delta x-b)}{2\Delta x^3}$. Again one can straightforwardly verify that (2.3.6) holds for any $0 < a < \Delta x$ and $0 < b < \Delta x$.

Thus, Theorem 2.3.2 holds: the diagonal entries of $\boldsymbol{A}^h = c\boldsymbol{I}^h - \boldsymbol{M}^h$ are all positive, the off-diagonal entries are all non-positive, and each row sums up to c. ■

Remark 2.3.5 *The M-Matrix result of Theorem 2.3.2, at least for our particular parameter choices, does not hold in 3-D. The Barycentric Lagrange interpolation weights at the corners of the interpolation stencils are negative. In the next section, we shall make a minor modification to the interpolation matrix for closest point extension so that we get an M-Matrix in 3-D and fulfill the convergence condition. This special modification is adopted for a theoretical reason, in practice we can still use Barycentric Lagrange interpolation in 3-D and obtain 2nd order convergence (see the numerical results in Section 2.5.3).*

2.4 Convergence for three dimensional embedding space

We cannot straightforwardly extend the M-Matrix proof from two dimensions to three dimensions because of the negative tricubic interpolation weights at the corners of the $4 \times 4 \times 4$ stencil. We overcome this problem by introducing a new 4th-order approximation to $u(\text{cp}(\boldsymbol{x}))$ without using corner points of the $4 \times 4 \times 4$ stencil. This new approximation will still be a linear combination of values of grid points in the $4 \times 4 \times 4$ stencil. Thus for each grid point $\boldsymbol{x}_i$ we have the following discretization for $[\Delta u](\text{cp}(\boldsymbol{x}_i)) - \gamma(\boldsymbol{x}_i)\big(u(\boldsymbol{x}_i) - u(\text{cp}(\boldsymbol{x}_i))\big)$,

$$\sum_{\boldsymbol{x}_j \in \text{I}_1(\text{cp}(\boldsymbol{x}_i))} \omega_j \left(\sum_{\boldsymbol{x}_k \in \text{Diff}(\boldsymbol{x}_j)} l_k u_k \right) - \gamma_i \left(u_i - \sum_{\boldsymbol{x}_j \in \text{I}_3(\text{cp}(\boldsymbol{x}_i))} \tilde{\omega}_j u_j \right). \tag{2.4.1}$$

Here $\text{I}_1(\text{cp}(\boldsymbol{x}_i))$ is the tri-linear interpolation stencil for point $\text{cp}(\boldsymbol{x}_i)$, $\text{Diff}(\boldsymbol{x}_j)$ is the standard 2nd-order difference stencil for point $\boldsymbol{x}_j$, and $\text{I}_3(\text{cp}(\boldsymbol{x}_i))$ tri-cubic interpolation stencil for point $\text{cp}(\boldsymbol{x}_i)$; ω_j are the weights of the linear interpolation scheme for evaluating $[\Delta \tilde{u}](\text{cp}(\boldsymbol{x}))$, $\tilde{\omega}_j$ are the weights of the 4th-order approximation scheme for evaluating $\tilde{u}(\text{cp}(\boldsymbol{x}))$, l_k are the weights for the difference scheme approximating the Laplacian operator, and $\gamma_i = \gamma(\boldsymbol{x}_i)$ is the weight determining the amount of penalization at point $\boldsymbol{x}_i$. Note that different γ_i's are chosen for different grid points. We also emphasize that although we use the tri-cubic interpolation stencil $\text{I}_3(\text{cp}(\boldsymbol{x}_i))$, the weights $\tilde{\omega}_j$'s are NOT the tri-cubic interpolation weights. They will be determined later for our M-Matrix construction.

Since everything is linear, (2.4.1) has the corresponding matrix form,

$$\tilde{\boldsymbol{M}}^h = \boldsymbol{E}_1^h \boldsymbol{L}^h - \boldsymbol{\Gamma}^h (\boldsymbol{I}^h - \tilde{\boldsymbol{E}}_3^h), \tag{2.4.2}$$

where $\boldsymbol{E}_1^h$ is the usual linear interpolation matrix for closest point extension, $\boldsymbol{L}^h$ is the 2nd-order Laplacian matrix, $\boldsymbol{\Gamma}^h$ is a diagonal matrix determining how much one wants to penalize for different grid points, $\boldsymbol{I}^h$ is the identity matrix, $\tilde{\boldsymbol{E}}_3^h$ is an approximation matrix still to be determined evaluating values at closest points up to 4th-order accuracy. A new discretization for the embedding equation (2.1.5) becomes

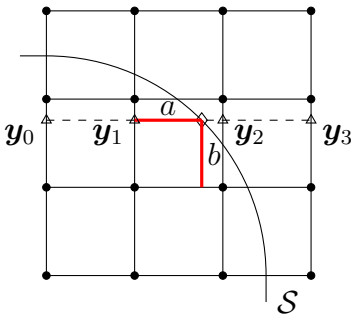
$$\tilde{\boldsymbol{A}}^h \boldsymbol{u}^h = \boldsymbol{f}^h, \quad \text{where } \tilde{\boldsymbol{A}}^h = -\boldsymbol{E}_1^h \boldsymbol{L}^h + \boldsymbol{\Gamma}^h(\boldsymbol{I}^h - \tilde{\boldsymbol{E}}_3^h) + c\boldsymbol{I}^h. \tag{2.4.3}$$

Our goal is to determine $\tilde{\boldsymbol{E}}_3^h$ and $\boldsymbol{\Gamma}^h$ so that scheme (2.4.3) is 2nd-order consistent to the embedding equation (2.1.5) and so that $\tilde{\boldsymbol{A}}^h$ is an M-Matrix for any positive c. Scheme (2.4.3) thus yields solutions which are 2nd-order convergent to the exact solutions of the embedding equation (2.1.5).

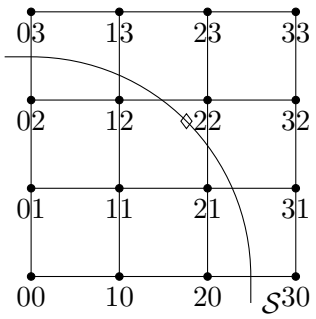
2.4.1 Illustration of the ideas in two dimensions

To illustrate how we determine $\tilde{\omega}_j$'s and construct $\tilde{\boldsymbol{E}}_3^h$ in 3-D, we first consider the 2-D case. Figure 2.2(a) shows the position of $\text{cp}(\boldsymbol{x}_i)$ (indicated by $\diamond$) in the middle block of the 4×4 stencil. We denote by a and b the lengths of the red line segments indicating the relative position of $\text{cp}(\boldsymbol{x}_i)$ in the stencil as shown in Figure 2.2(a). We then draw a horizontal line intersecting with the vertical grid lines at $\boldsymbol{y}_0$, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$ and $\boldsymbol{y}_3$.

Figure 2.2: Illustration of approximating $u(\text{cp}(\boldsymbol{x}_i))$ in two dimensions.



(a) Illustration of approximation of u at $\text{cp}(\boldsymbol{x}_i)$ $\diamond$



(b) Local index for points in the stencil

We approximate $u(\text{cp}(\boldsymbol{x}_i))$ along the x-direction using cubic interpolation,

$$\begin{aligned} u(\text{cp}(\boldsymbol{x}_i)) \approx & -\frac{a(\Delta x - a)(2\Delta x - a)}{6(\Delta x)^3} u(\boldsymbol{y}_0) + \frac{(\Delta x + a)(\Delta x - a)(2\Delta x - a)}{2(\Delta x)^3} u(\boldsymbol{y}_1) \\ & + \frac{(\Delta x + a)a(2\Delta x - a)}{2(\Delta x)^3} u(\boldsymbol{y}_2) - \frac{(\Delta x + a)a(\Delta x - a)}{6(\Delta x)^3} u(\boldsymbol{y}_3). \end{aligned} \tag{2.4.4}$$

If we approximate $u(\boldsymbol{y}_m)$'s $(m = 0, 1, 2, 3)$ via cubic interpolation using values of grid points lying along the y-directions, then we would get the usual bi-cubic interpolation for $u(\mathrm{cp}(\boldsymbol{x}_i))$. Instead of doing this, we make use of the following relationship,

$$u_{xx}(\boldsymbol{y}_1) \approx \frac{u(\boldsymbol{y}_0) - 2u(\boldsymbol{y}_1) + u(\boldsymbol{y}_2)}{\Delta x^2} \tag{2.4.5}$$

From (2.4.5) we solve for $u(\boldsymbol{y}_0)$ to get

$$u(\boldsymbol{y}_0) \approx 2u(\boldsymbol{y}_1) - u(\boldsymbol{y}_2) + \Delta x^2 u_{xx}(\boldsymbol{y}_1). \tag{2.4.6}$$

Similarly, we approximate $u(\boldsymbol{y}_3)$ by

$$u(\boldsymbol{y}_3) \approx 2u(\boldsymbol{y}_2) - u(\boldsymbol{y}_1) + \Delta x^2 u_{xx}(\boldsymbol{y}_2). \tag{2.4.7}$$

Substituting equations (2.4.6) and (2.4.7) into (2.4.4), and merging the coefficients of $u(\boldsymbol{y}_1)$ and $u(\boldsymbol{y}_2)$, we get the following approximation for $u(\mathrm{cp}(\boldsymbol{x}_i))$,

$$\begin{aligned} u(\mathrm{cp}(\boldsymbol{x}_i)) \;\approx\; & -\frac{a(\Delta x - a)(2\Delta x - a)}{6\Delta x} u_{xx}(\boldsymbol{y}_1) + \frac{\Delta x - a}{\Delta x} u(\boldsymbol{y}_1) \\ & + \frac{a}{\Delta x} u(\boldsymbol{y}_2) - \frac{(\Delta x + a)a(\Delta x - a)}{6\Delta x} u_{xx}(\boldsymbol{y}_2). \end{aligned} \tag{2.4.8}$$

Now the remaining question is to approximate $u(\boldsymbol{y}_1)$, $u(\boldsymbol{y}_2)$, $u_{xx}(\boldsymbol{y}_1)$ and $u_{xx}(\boldsymbol{y}_2)$ by values of grid points in the 4×4 stencil. We approximate $u(\boldsymbol{y}_1)$ and $u(\boldsymbol{y}_2)$ via cubic interpolation along the y-direction:

$$\begin{aligned} u(\boldsymbol{y}_1) \;\approx\; & -\frac{b(\Delta x - b)(2\Delta x - b)}{6(\Delta x)^3} u_{10} + \frac{(\Delta x + b)(\Delta x - b)(2\Delta x - b)}{2(\Delta x)^3} u_{11} \\ & + \frac{(\Delta x + b)b(2\Delta x - b)}{2(\Delta x)^3} u_{12} - \frac{(\Delta x + b)b(\Delta x - b)}{6(\Delta x)^3} u_{13}; \end{aligned} \tag{2.4.9}$$

$$\begin{aligned} u(\boldsymbol{y}_2) \;\approx\; & -\frac{b(\Delta x - b)(2\Delta x - b)}{6(\Delta x)^3} u_{20} + \frac{(\Delta x + b)(\Delta x - b)(2\Delta x - b)}{2(\Delta x)^3} u_{21} \\ & + \frac{(\Delta x + b)b(2\Delta x - b)}{2(\Delta x)^3} u_{22} - \frac{(\Delta x + b)b(\Delta x - b)}{6(\Delta x)^3} u_{23}. \end{aligned} \tag{2.4.10}$$

Notice that (2.4.9) and (2.4.10) do not involve values of u at the corner points.

To approximate $u_{xx}(\boldsymbol{y}_1)$ and $u_{xx}(\boldsymbol{y}_2)$, we first use centered finite differences to approximate u_{xx} at grid points with local indices (11), (12), (21), and (22), and then perform linear interpolation along the y-directions. We have

$$u_{xx}(\boldsymbol{y}_1) \approx \frac{\Delta x - b}{\Delta x} \frac{u_{01} - 2u_{11} + u_{21}}{(\Delta x)^2} + \frac{b}{\Delta x} \frac{u_{02} - 2u_{12} + u_{22}}{(\Delta x)^2}; \tag{2.4.11}$$

$$u_{xx}(\boldsymbol{y}_2) \approx \frac{\Delta x - b}{\Delta x}\frac{u_{11} - 2u_{21} + u_{31}}{(\Delta x)^2} + \frac{b}{\Delta x}\frac{u_{12} - 2u_{22} + u_{32}}{(\Delta x)^2}. \tag{2.4.12}$$

Again (2.4.11) and (2.4.12) do not have values at the corner points.

We then substitute (2.4.9), (2.4.10), (2.4.11) and (2.4.12) into (2.4.8) to get the final approximation for $u(\mathrm{cp}(\boldsymbol{x}_i))$ using values of grid points in the 4×4 stencil except those at corner points. Up to now we have constructed a new approximation to $u(\mathrm{cp}(\boldsymbol{x}_i))$. We shall check two properties of this approximation: 1. it is 4th-order accurate; 2. it indeed leads to non-negative off-diagonal weights for $\tilde{\boldsymbol{M}}^h$ provided that we choose proper γ_i's.

The first property is easy to prove. Since we use 2nd-order approximation to u_{xx} everywhere and 4th-order approximation to u everywhere, the final approximation to u is 4th-order accurate.

The second property, which is crucial to our whole construction and proof, is more difficult to achieve and check. We categorize the grid points appearing in the approximation scheme into 3 different types:

(1) Points in the center with local indices: (11), (12), (21), (22);

(2) Points on the top and bottom edges: (10), (20), (13), (23);

(3) Points on the left and right edges: (01), (02), (31), (32).

We first consider points of type (1). The coefficient of point with local index (11) is

$$\begin{aligned}\gamma_i\Big(\tfrac{\Delta x-a}{\Delta x}\tfrac{(\Delta x+b)(\Delta x-b)(2\Delta x-b)}{2(\Delta x)^3} + 2\tfrac{a(\Delta x-a)(2\Delta x-a)}{6(\Delta x)^3}\tfrac{\Delta x-b}{\Delta x} - \tfrac{(\Delta x+a)a(\Delta x-a)}{6(\Delta x)^3}\tfrac{\Delta x-b}{\Delta x}\Big)\\ +\Big(-\tfrac{\Delta x-a}{\Delta x}\tfrac{\Delta x-b}{\Delta x}\tfrac{4}{(\Delta x)^2} + \tfrac{a}{\Delta x}\tfrac{\Delta x-b}{\Delta x}\tfrac{1}{(\Delta x)^2} + \tfrac{\Delta x-a}{\Delta x}\tfrac{b}{\Delta x}\tfrac{1}{(\Delta x)^2}\Big),\end{aligned} \tag{2.4.13}$$

where the terms in the first big brackets (pre-multiplied by γ_i) come from approximation of $u(\mathrm{cp}(\boldsymbol{x}_i))$, and the terms in the second big brackets come from approximation of $[\Delta u](\mathrm{cp}(\boldsymbol{x}_i))$. We wish to choose γ_i so that (2.4.13) is non-negative. One can check that

$$2\frac{a(\Delta x - a)(2\Delta x - a)}{6(\Delta x)^3}\frac{\Delta x - b}{\Delta x} - \frac{(\Delta x + a)a(\Delta x - a)}{6(\Delta x)^3}\frac{\Delta x - b}{\Delta x} \geq 0. \tag{2.4.14}$$

In fact, (2.4.14) is equivalent to $3(\Delta x - a) \geq 0$ after cancelling out common factors. Also, it is obvious that $\frac{a}{\Delta x}\frac{\Delta x-b}{\Delta x}\frac{1}{(\Delta x)^2} \geq 0$, and $\frac{\Delta x-a}{\Delta x}\frac{b}{\Delta x}\frac{1}{(\Delta x)^2} \geq 0$. Thus it is sufficient to choose γ_i s.t.

$$\gamma_i\frac{\Delta x - a}{\Delta x}\frac{(\Delta x + b)(\Delta x - b)(2\Delta x - b)}{2(\Delta x)^3} = \frac{\Delta x - a}{\Delta x}\frac{\Delta x - b}{\Delta x}\frac{4}{(\Delta x)^2}. \tag{2.4.15}$$

From (2.4.15) we solve for γ_i and get

$$\gamma_i = \frac{8}{(\Delta x + b)(2\Delta x - b)}. \tag{2.4.16}$$

Since (2.4.16) is independent of a and is invariant if we change b into $\Delta x - b$, this choice of γ_i will ensure the coefficients of the other points of type (1) with indices (12), (21) and (22) to be non-negative.

We then prove that this choice of γ_i given by (2.4.16) will also result in non-negative coefficients for points of type (2) and (3). For points of type (2), consider for example the coefficient of point (10):

$$-\gamma_i \frac{\Delta x - a}{\Delta x} \frac{b(\Delta x - b)(2\Delta x - b)}{6(\Delta x)^3} + \frac{\Delta x - a}{\Delta x} \frac{\Delta x - b}{\Delta x} \frac{1}{\Delta x^2}. \tag{2.4.17}$$

To check that (2.4.17) is non-negative, we substitute (2.4.15) into (2.4.17) and cancel out the common factors to get $3\Delta x - b \geq 0$, which is trivially true. By symmetry all other points of type (2) have positive coefficients.

For points of type (3), consider for example the coefficient of point (01):

$$-\gamma_i \frac{a(\Delta x - a)(2\Delta x - a)}{6(\Delta x)^3} \frac{\Delta x - b}{\Delta x} + \frac{\Delta x - a}{\Delta x} \frac{\Delta x - b}{\Delta x} \frac{1}{\Delta x^2}. \tag{2.4.18}$$

To check that (2.4.18) is non-negative, we substitute (2.4.15) into (2.4.18) and cancel out the common factors to get

$$4a(2\Delta x - a) \leq 3(\Delta x + b)(2\Delta x - b). \tag{2.4.19}$$

One can show that

$$4a(2\Delta x - a) \leq 4\Delta x^2, \text{ for any } 0 \leq a < \Delta x,$$

and

$$3(\Delta x + b)(2\Delta x - b) \geq 6\Delta x^2, \text{ for any } 0 \leq b < \Delta x.$$

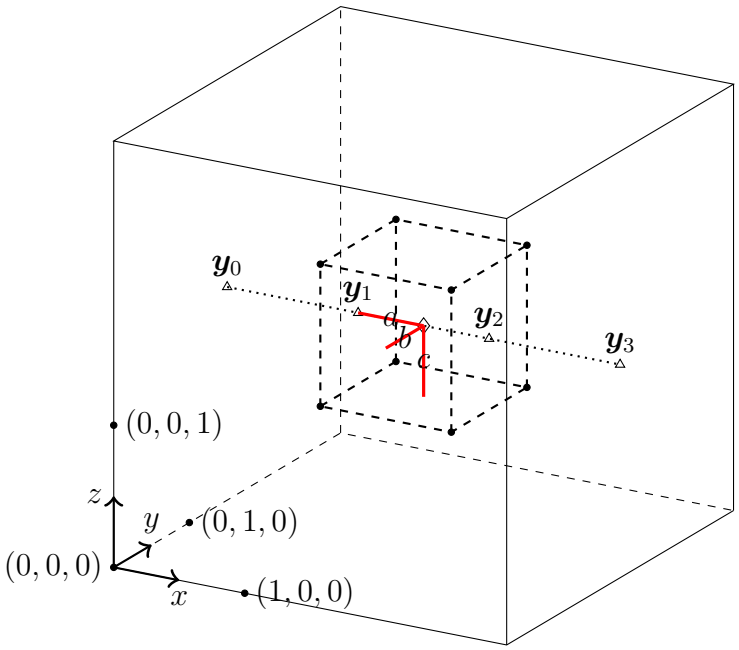
Therefore (2.4.19) holds and (2.4.18) is indeed non-negative. By symmetry all other type (3) points have positive coefficients.

In summary, if we approximate $u(\text{cp}(\boldsymbol{x}_i))$ by (2.4.8) where $u(\boldsymbol{y}_1)$, $u(\boldsymbol{y}_2)$, $u_{xx}(\boldsymbol{y}_1)$ and $u_{xx}(\boldsymbol{y}_2)$ are approximated by (2.4.9), (2.4.10), (2.4.11) and (2.4.12) respectively, and choose $\gamma_i = \frac{8}{(\Delta x+b)(2\Delta x-b)}$, then all the coefficients except the one of u_i in scheme (2.4.1) are non-negative. Thus the resulting matrix $\tilde{\boldsymbol{M}}^h$ given by (2.4.2) only has non-negative off-diagonal entries. Since each row of $\tilde{\boldsymbol{M}}^h$ sums up to 0, $\tilde{\boldsymbol{M}}^h$ can only have non-positive diagonal entries. Therefore, $\tilde{\boldsymbol{A}}^h = -\tilde{\boldsymbol{M}}^h + c\boldsymbol{I}^h$ is an M-matrix for any positive c.

2.4.2 Construction and proof in three dimensions

In this section, we straightforwardly extend the ideas for two dimensions in Section 2.4.1 to three dimensions. Suppose the closest point $\mathrm{cp}(\boldsymbol{x}_i)$ $\diamond$ lies in the $4\times4\times4$ stencil as shown in Figure 2.3. We denote by a, b and c the lengths of the red line segments indicating the relative position of $\mathrm{cp}(\boldsymbol{x}_i)$ in the stencil as shown in Figure 2.3. We then draw a horizontal line which is orthogonal to the y-z plane and intersects with four faces parallel to the y-z plane at $\boldsymbol{y}_0$, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, and $\boldsymbol{y}_3$.

Figure 2.3: Illustration of the approximation of $u(\mathrm{cp}(\boldsymbol{x}_i))$ (indicated by $\diamond$).



We approximate $u(\mathrm{cp}(\boldsymbol{x}_i))$ along the x-direction using cubic interpolation,

$$\begin{aligned} u(\mathrm{cp}(\boldsymbol{x}_i)) \;\approx\; & -\frac{a(\Delta x-a)(2\Delta x-a)}{6(\Delta x)^3}u(\boldsymbol{y}_0)+\frac{(\Delta x+a)(\Delta x-a)(2\Delta x-a)}{2(\Delta x)^3}u(\boldsymbol{y}_1) \\ & +\frac{(\Delta x+a)a(2\Delta x-a)}{2(\Delta x)^3}u(\boldsymbol{y}_2)-\frac{(\Delta x+a)a(\Delta x-a)}{6(\Delta x)^3}u(\boldsymbol{y}_3). \end{aligned} \tag{2.4.20}$$

Notice that (2.4.20) is exactly the same as (2.4.4) in Section 2.4.1. Using the ideas in Section 2.4.1, we continue to obtain the following approximation to $u(\mathrm{cp}(\boldsymbol{x}_i))$,

$$\begin{aligned} u(\mathrm{cp}(\boldsymbol{x}_i)) \;\approx\; & -\frac{a(\Delta x-a)(2\Delta x-a)}{6\Delta x}u_{xx}(\boldsymbol{y}_1)+\frac{\Delta x-a}{\Delta x}u(\boldsymbol{y}_1) \\ & +\frac{a}{\Delta x}u(\boldsymbol{y}_2)-\frac{(\Delta x+a)a(\Delta x-a)}{6\Delta x}u_{xx}(\boldsymbol{y}_2). \end{aligned} \tag{2.4.21}$$

Note that (2.4.21) is exactly the same as (2.4.8) in Section 2.4.1. The differences between 2-D and 3-D come from the approximation of $u(\boldsymbol{y}_1)$, $u(\boldsymbol{y}_2)$, $u_{xx}(\boldsymbol{y}_1)$, and $u_{xx}(\boldsymbol{y}_2)$ using values of grid points, although the spirit is the same.

We approximate $u(\boldsymbol{y}_1)$ and $u(\boldsymbol{y}_2)$ through bi-cubic interpolations of grid point values along the y-z plane:

$$u(\boldsymbol{y}_1) \approx \sum_{0\le j,k\le 3} \tilde{\omega}_{jk} u_{1,j,k}, \tag{2.4.22}$$

$$u(\boldsymbol{y}_2) \approx \sum_{0\le j,k\le 3} \tilde{\omega}_{jk} u_{2,j,k}, \tag{2.4.23}$$

where $\tilde{\omega}_{jk}$ $(0 \le j,k \le 3)$ are the bi-cubic weights along the y-z plane. We approximate $u_{xx}(\boldsymbol{y}_1)$ and $u_{xx}(\boldsymbol{y}_2)$ using bi-linear interpolations of u_{xx}'s calculated by centered finite differences at grid points:

$$\begin{aligned} u_{xx}(\boldsymbol{y}_1) \;=\; & \frac{(\Delta x-b)(\Delta x-c)}{\Delta x^2}\frac{u_{0,1,1}-2u_{1,1,1}+u_{2,1,1}}{\Delta x^2} + \frac{b(\Delta x-c)}{\Delta x^2}\frac{u_{0,2,1}-2u_{1,2,1}+u_{2,2,1}}{\Delta x^2} \\ & + \frac{(\Delta x-b)c}{\Delta x^2}\frac{u_{0,1,2}-2u_{1,1,2}+u_{2,1,2}}{\Delta x^2} + \frac{bc}{\Delta x^2}\frac{u_{0,2,2}-2u_{1,2,2}+u_{2,2,2}}{\Delta x^2} \end{aligned} \tag{2.4.24}$$

$$\begin{aligned} u_{xx}(\boldsymbol{y}_2) \;=\; & \frac{(\Delta x-b)(\Delta x-c)}{\Delta x^2}\frac{u_{1,1,1}-2u_{2,1,1}+u_{3,1,1}}{\Delta x^2} + \frac{b(\Delta x-c)}{\Delta x^2}\frac{u_{1,2,1}-2u_{2,2,1}+u_{3,2,1}}{\Delta x^2} \\ & + \frac{(\Delta x-b)c}{\Delta x^2}\frac{u_{1,1,2}-2u_{2,1,2}+u_{3,1,2}}{\Delta x^2} + \frac{bc}{\Delta x^2}\frac{u_{1,2,2}-2u_{2,2,2}+u_{3,2,2}}{\Delta x^2} \end{aligned} \tag{2.4.25}$$

Similar to the 2-D case, we categorize grid points appearing in the approximation scheme into four types:

(1) 8 center points.

(2) 16 points in the center of top, bottom, front and back faces.

(3) 8 points in the center of left and right faces.

(4) 8 points on some of the edges of top and bottom faces.

Points of type (4) do not appear in 2-D, these are actually corner points of the bi-cubic interpolation stencils along the y-z plane to approximate $u(\boldsymbol{y}_1)$ and $u(\boldsymbol{y}_2)$. It is obvious that they contribute positive coefficients to our final scheme.

Points of type (1)–(3) are similar to the 2-D cases.

For points of type (1), we consider the coefficient of point (111) without loss of generality. Using similar arguments to the 2-D case, it is enough to pick γ_i s.t.

$$\gamma_i \frac{\Delta x-a}{\Delta x}\frac{(\Delta x+b)(\Delta x-b)(2\Delta x-b)}{2(\Delta x)^3}\frac{(\Delta x+c)(\Delta x-c)(2\Delta x-c)}{2(\Delta x)^3} = \frac{\Delta x-a}{\Delta x}\frac{\Delta x-b}{\Delta x}\frac{\Delta x-c}{\Delta x}\frac{6}{(\Delta x)^2}. \tag{2.4.26}$$

From (2.4.26) we solve for γ_i to get

$$\gamma_i = \frac{24(\Delta x)^2}{(\Delta x + b)(2\Delta x - b)(\Delta x + c)(2\Delta x - c)}. \tag{2.4.27}$$

Notice that the expression of γ_i (2.4.27) is independent of a, and invariant if we replace b by $\Delta x - b$ or replace c by $\Delta x - c$.

For points of type (2), we consider for example the coefficient of point (110):

$$-\gamma_i \tfrac{\Delta x - a}{\Delta x} \tfrac{(\Delta x+b)(\Delta x-b)(2\Delta x-b)}{2(\Delta x)^3} \tfrac{c(\Delta x-c)(2\Delta x-c)}{6(\Delta x)^3} + \tfrac{\Delta x-a}{\Delta x} \tfrac{\Delta x-b}{\Delta x} \tfrac{\Delta x-c}{\Delta x} \tfrac{1}{\Delta x^2}. \tag{2.4.28}$$

One can check that (2.4.28) is non-negative. In fact, after substituting (2.4.26) into (2.4.28) and cancelling out the common factors, (2.4.28)≥ 0 simply reduces to $c \leq \Delta x$. By symmetry all other type (2) points have positive coefficients.

For points of type (3), consider for example the coefficient of point (011):

$$-\gamma_i \frac{a(\Delta x - a)(2\Delta x - a)}{6(\Delta x)^3} \frac{\Delta x - b}{\Delta x} \frac{\Delta x - c}{\Delta x} + \frac{\Delta x - a}{\Delta x} \frac{\Delta x - b}{\Delta x} \frac{\Delta x - c}{\Delta x} \frac{1}{\Delta x^2}. \tag{2.4.29}$$

To check that (2.4.29) is non-negative, we substitute (2.4.27) into (2.4.29) and rearrange to obtain

$$4a(2\Delta x - a) \leq (\Delta x + b)(2\Delta x - b)(\Delta x + c)(2\Delta - c). \tag{2.4.30}$$

One can show that

$$4a(2\Delta x - a) \leq 4\Delta x^2, \text{ for any } 0 \leq a < \Delta x,$$

$$(\Delta x + b)(2\Delta x - b) \geq 2\Delta x^2, \text{ for any } 0 \leq b < \Delta x,$$

$$(\Delta x + c)(2\Delta x - c) \geq 2\Delta x^2, \text{ for any } 0 \leq c < \Delta x.$$

Therefore (2.4.30) holds and (2.4.29) is non-negative. By symmetry all other type (3) points have positive coefficients.

To conclude, we have constructed a specific approximation to evaluate the values at closest points so that the coefficient matrix for the difference scheme is an M-Matrix satisfying the condition of Proposition 2.3.3. Also, since the approximation to $u(\text{cp}(\boldsymbol{x}_i))$ is 4th-order accurate and γ_i is of order $O(\frac{1}{\Delta x^2})$, the overall scheme has 2nd-order truncation error. Thus using Proposition 2.3.4 our modified scheme is 2nd-order convergent. To be more precise, we have the following theorem,

Theorem 2.4.1 *Suppose that we wish to solve the embedding equation (2.1.5) on a smooth curve or surface without boundary embedded in $\mathbb{R}^3$, and the solution $\tilde{u}$ of the embedding equation satisfies the assumptions in Theorem 2.2.1. Denote $\boldsymbol{L}^h$ as the standard Laplacian matrix with stencil $\frac{1}{\Delta x^2}\{-6,1,1,1,1,1,1\}$, define $\tilde{\boldsymbol{E}}_3^h$ by the approximations (2.4.21) – (2.4.25), and let $\boldsymbol{\Gamma}^h$ be an diagonal matrix whose diagonal entries are defined by (2.4.27). Then $\tilde{\boldsymbol{A}}^h$ defined by (2.4.3) is an M-Matrix, and scheme (2.4.3) gives a numerical solution that is 2nd-order accurate to $\tilde{u}$.*

2.5 Numerical examples

2.5.1 Shifted Poisson's equation on a unit circle

We solve the shifted Poisson's equation $-\Delta_{\mathcal{S}} u + u = f$ on the unit circle $\mathcal{S}$; the exact solution is chosen as $\sin(\theta)+\sin(12\theta)$, and we compute the right-hand side function f from the exact solution. The discrete right-hand side is then obtained by evaluating the closest point extension of f at the grid points in the computational band. We construct $\boldsymbol{M}^h = \boldsymbol{E}_1^h \boldsymbol{L}^h - \frac{2d}{\Delta x^2}(\boldsymbol{I}^h - \boldsymbol{E}_3^h)$, and the coefficient matrix is $\boldsymbol{A}^h = \boldsymbol{I}^h - \boldsymbol{M}^h$.

We use the sparse direct solver MATLAB's backslash to solve the resulting linear systems. After solving the linear system, we obtain the discrete solution defined on the banded grid points. To calculate the errors, we then place many sample points on $\mathcal{S}$, estimate the numerical solution at these points by cubic interpolation of the values at the surrounding grid points, and compute the ∞-norm of the errors at the sample points on $\mathcal{S}$. Table 2.1 shows second order convergence of the relative errors in the ∞-norm. Alternatively, the exact closest point extension of the exact solution can be performed onto the computational grid, and the errors can be directly calculated at the grid points; similar results are observed in this case.

Table 2.1: Convergence study for the shifted Poisson's equation on a unit circle showing 2nd-order convergence.

Δx	0.1	0.05	0.025	0.0125	0.00625	0.003125	0.0015625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	6.51e-2	1.85e-2	4.76e-3	1.19e-3	2.97e-4	7.38e-5	1.85e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.81	1.95	2.00	2.00	2.01	2.00

2.5.2 Shifted Poisson's equation on a bean-shaped curve

Again we solve the shifted Poisson's equation $-\Delta_{\mathcal{S}} u + u = f$, but this time on a bean-shaped curve (see Figure 2.4).

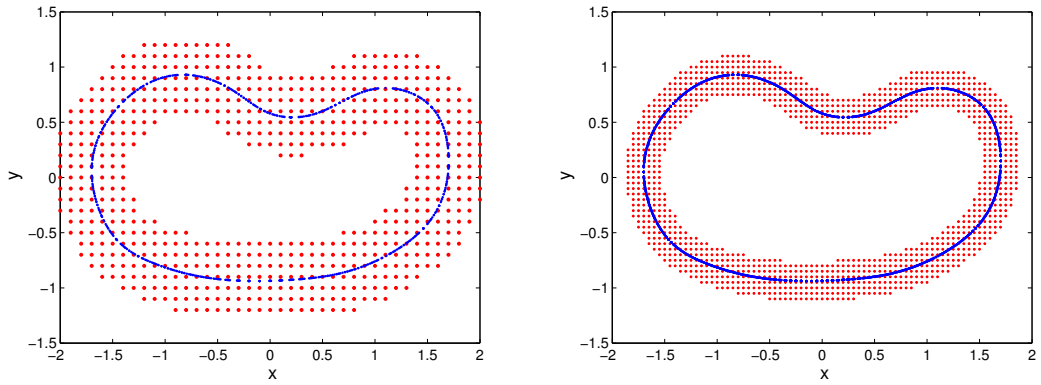


Figure 2.4: Computational grids for the Closest Point Method on a bean-shaped curve with $\Delta x = 0.1$ (left) and $\Delta x = 0.05$ (right).

This bean-shaped curve is constructed from a sequence of control points in $\mathbb{R}^2$; the Matlab function 'cscvn' [81] is used to get a parametric cubic spline curve passing through these points. This results in $(x(t), y(t))$ parameterized by $t \in [0, T]$. We obtain a closest point representation from this parametrization as follows: we fix a grid point, then construct the squared distance as a function of the parameter t. Next we minimize that function (over the parameter t) using Newton's method [12]. This is repeated for each grid point in the computational band.

We construct the exact solution in terms of the parametrization, $u(t) = \sin(\frac{2\pi}{T}t) + \sin(\frac{20\pi}{T}t)$. The Laplace–Beltrami operator of u in terms of this parametrization is $\Delta_{\mathcal{S}} u(t) = \frac{1}{\sqrt{\dot{x}^2(t)+\dot{y}^2(t)}} \frac{\mathrm{d}}{\mathrm{d}t}\big(\frac{1}{\sqrt{\dot{x}^2(t)+\dot{y}^2(t)}} \frac{\mathrm{d}u(t)}{\mathrm{d}t}\big)$. The right-hand side function is $f(t) = -\Delta_{\mathcal{S}} u(t) + u(t)$. Again we construct the discrete right-hand side and coefficient matrix, solve the resulting linear system to get the numerical solution, and compute the ∞-norm of the error on the bean-shaped curve. Table 2.2 shows second order convergence results.

Table 2.2: Convergence study for the shifted Poisson's equation on a bean-shaped curve showing 2nd-order convergence.

Δx	0.1	0.05	0.025	0.0125	0.00625	0.003125	0.0015625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	6.19e-2	1.17e-2	3.59e-3	8.77e-4	2.35e-4	5.32e-5	1.38e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.40	1.71	2.03	1.90	2.14	1.95

2.5.3 Shifted Poisson's equation on a unit sphere

To make an exact solution and do convergence study, we parameterize a sphere in spherical coordinates (θ, ϕ, r): $x = r\sin\phi\cos\theta$, $y = r\sin\phi\sin\theta$, $z = r\cos\phi$, where θ is the azimuth ranging from 0 to 2π, ϕ is the elevation from 0 to π, and r is the radius.

We emphasize that the parametrization is not used in our algorithm: it is only used to compute the error. For simplicity, we solve the equation on a unit sphere centered at the origin, and choose the exact solution as the spherical harmonic $u(\theta, \phi) = \cos(3\theta)\sin(\phi)^3(9\cos(\phi)^2 - 1)$. The right-hand side function is $-\Delta_{\mathcal{S}} u + u = -29u(\theta, \phi)$. Table 2.3 shows the results using the standard dimension-by-dimension Barycentric Lagrange interpolation for discrete closest point extension of the solution. Table 2.4 shows the results using the specially designed approximation for discrete closest point extension of the solution described in Section 2.4.2. We can see that both the standard interpolation and the custom approximation give 2nd-order convergence results.

We also numerically verify the fact that the standard interpolation yields a coefficient matrix $\boldsymbol{A}^h$ which is not an M-Matrix, while the approximation described in Section 2.4.2 does yield a coefficient matrix $\tilde{\boldsymbol{A}}^h$ which is an M-Matrix. We first verify that each row of the coefficient matrix does sum up to the shift c. We then check the number of positive off-diagonal entries of the coefficient matrix. If there are no positive off-diagonal entries in the coefficient matrix then it is an M-Matrix, otherwise it is not an M-Matrix. Table 2.5 shows that $\tilde{\boldsymbol{A}}^h$ is indeed an M-Matrix while $\boldsymbol{A}^h$ is not.

Table 2.3: Convergence study for the shifted Poisson's equation on a unit sphere using the Barycentric Lagrange interpolation.

Δx	0.4	0.2	0.1	0.05
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	4.21e-1	1.24e-1	2.81e-2	7.06e-3
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.77	2.14	1.99

Table 2.4: Convergence study for the shifted Poisson's equation on a unit sphere using the specific approximations described in Section 2.4.2.

Δx	0.4	0.2	0.1	0.05
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	4.45e-1	1.26e-1	2.90e-2	7.25e-3
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.82	2.12	2.00

Table 2.5: Verification of the M-Matrix property in 3-D

(a) Number of non-zero entries and positive off-diagonal entries of $\boldsymbol{A}^h$ resulting from the standard Barycentric Lagrange interpolation.

Δx	0.4	0.2	0.1	0.05
$\text{nnz}(\boldsymbol{A}^h)$	78016	205686	702714	2697038
#(positive off-diagonal entries)	21704	54202	188314	717906

(b) Number of non-zero entries and positive off-diagonal entries of $\tilde{\boldsymbol{A}}^h$ resulting from the specific approximation described in Section 2.4.2.

Δx	0.4	0.2	0.1	0.05
$\text{nnz}(\tilde{\boldsymbol{A}}^h)$	49176	129422	441842	1695758
#(positive off-diagonal entries)	0	0	0	0

For all the above numerical tests including the sphere example, we use MATLAB's backslash to solve the linear system. This works well for curves embedded in 2-D, but for spheres embedded in 3-D we are experiencing memory bottleneck and low speed. In Chapter 4, we shall construct a geometric multgrid method to overcome these issues. Examples on other surfaces and with various coefficients will then be considered.

Chapter 3

Elliptic Equations on Domains and Surfaces with Boundaries

Now that we are able to solve elliptic equations on surfaces without a boundary, we move on to consider elliptic equations on surfaces with a boundary. The work [59] adopted the Closest Point Method to solve the Laplace–Beltrami eigenvalue problems with Dirichlet and homogeneous Neumann boundary conditions on surfaces with a boundary. We shall follow and extend the ideas in [59] to handle more general boundary conditions.

Notice that smooth surfaces are manifolds embedded in the Euclidean space, and domains with a curved boundary can be regarded as codimension-0 manifolds. So we first consider the solution of elliptic PDEs on domains with a boundary, which will develop the key ideas to handle boundary conditions for surfaces with a boundary. There exists vast literature on solving PDEs on general domains using a Cartesian grid. Some well-known methods are the immersed boundary method [72, 73, 38], immersed interface method [53, 55, 56] and the ghost fluid method [29, 30]. References on elliptic PDEs include [42, 35, 34, 93, 44]. These references embed the domain into a larger Cartesian grid and handle the boundary conditions with the help of grid points outside the domain – the so-called "ghost" points. Among the ghost point approaches there is a systematic method called the matched interface and boundary (MIB) method [93, 92, 90, 89] which extends the solution smoothly across the interface/boundary and disassociates the discretization of internal PDEs and boundary conditions.

One crucial step of the ghost point approach is to extend the solution from interior grid points to ghost points so that boundary conditions are satisfied. This solution extension step is not trivial in dimensions higher than one. Typical implementations

require computing the intersection points of the boundary and grid lines along coordinate directions [34, 93, 44]. For Neumann and Robin boundary conditions one usually needs to compute the normals at intersection points to evaluate the derivatives along the normals at the boundary [93, 92, 90]; and the implementations for 3-D problems require a lot more effort than 2-D problems [89]. We shall introduce a new method for the solution extension based on the closest point representation of the boundary of the domain. The implementations for arbitrary dimensions are quite straightforward in the existing framework of the Closest Point Method. With the extended solution it is straightforward to solve elliptic PDEs on domains with a curved boundary. Combining the ghost point approach to handle boundary conditions and the method of dealing with surface differential operators, it is not difficult to solve elliptic equations on surfaces with a boundary.

The rest of this chapter is organized as follows. In Section 3.1 we review the ghost point idea in 1-D, present several well-known theorems, and introduce a new simple method to extrapolate values of ghost points from values of interior grid points. In Section 3.2 we extend the new extrapolation method from Section 3.1 to higher dimensions, and systematically construct schemes to solve elliptic equations on domains with Dirichlet, Neumann and Robin boundary conditions. In Section 3.3 we combine the ideas in Section 3.2 and the previous chapter to solve elliptic equations on curves and surfaces with boundaries.

3.1 Ghost points approach in 1-D

We start this section with reviewing the ghost point idea by considering Poisson problems on an interval with Dirichlet and Neumann boundary conditions. We then introduce a new method to extrapolate values of ghost points from interior grid points. This new extrapolation method perhaps seems slightly awkward for 1-D problems, but it gives a great flexibility to handle general boundary conditions in higher dimensional spaces.

3.1.1 Dirichlet boundary conditions

Consider the following 1-D Poisson's equation with Dirichlet boundary conditions:

$$-u''(x) = f(x), \quad x \in (a,b), \tag{3.1.1a}$$

$$u(a) = u_a, \quad u(b) = u_b. \tag{3.1.1b}$$

Suppose the interval $[a, b]$ lies in a 1-D uniform grid $x_0, x_1, \ldots, x_{n-1}, x_n$ such that $x_0 \leq a < x_1$ and $x_{n-1} < b \leq x_n$. Points x_0 and x_n are not in the interior of the interval, and they are the so-called ghost points. For ease of analysis for the Dirichlet boundary condition case, here we have assumed that the ghost points x_0 and x_n could potentially be boundary points. However, for the analysis of Neumann boundary conditions and practical computing, ghost points refer to grid points strictly exterior to the domain. For grid points $x_1, \ldots, x_{n-1}$ in the interior of the interval, we can discretize (3.1.1a) using the centered finite difference scheme:

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} = f_i, \quad i = 1, 2, \ldots, n-1, \tag{3.1.2}$$

where Δx is the grid size, $f_i = f(x_i)$, and u_i is the numerical value at x_i. We need to extrapolate u_0 and u_n from values of interior points. The simplest way to do this is constant extrapolation:

$$u_0 = u_a, \quad u_n = u_b. \tag{3.1.3}$$

Combining (3.1.2) and (3.1.3), we obtain a complete discretization for problem (3.1.1),

$$\begin{aligned} -\frac{u_a - 2u_1 + u_2}{\Delta x^2} &= f_1, \\ -\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} &= f_i, \quad i = 2, \ldots, n-2, \\ -\frac{u_{n-2} - 2u_{n-1} + u_b}{\Delta x^2} &= f_{n-1}. \end{aligned} \tag{3.1.4}$$

The matrix form of (3.1.4) is

$$\frac{1}{\Delta x^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 + \frac{u_a}{\Delta x^2} \\ f_2 \\ \vdots \\ f_{n-2} \\ f_{n-1} + \frac{u_b}{\Delta x^2} \end{pmatrix}. \tag{3.1.5}$$

It is well known that if $x_0 = a$ and $x_n = b$, then scheme (3.1.4) yields 2nd-order convergent solution to the solution of (3.1.1) given enough smoothness of f. The convergence results when $x_0 < a$ and $b < x_n$ are summarized in the following theorem.

Theorem 3.1.1 *(Jomaa and Macaskill 2005 [43]) Suppose (3.1.1) has an exact solution $u(x) \in \mathcal{C}^4$. Let τ_i be the truncation error of scheme (3.1.4) at x_i, and $e_i = u_i - u(x_i)$ be the error, for $i = 1, 2, \ldots, n-1$. Assume that $a - x_0 = \alpha\Delta x$ and $x_n - b = \beta\Delta x$ where $0 < \alpha, \beta < 1$ are constants, then $\tau_1 = O(\frac{1}{\Delta x})$, $\tau_{n-1} = O(\frac{1}{\Delta x})$, $\tau_i = O(\Delta x^2)$ for $i = 2, \ldots, n-2$, and $e_i = O(\Delta x)$ for $i = 1, 2, \ldots, n-1$.*

Remark 3.1.2 *We cite Theorems 3.1.1 from [43], but prove it using our own methods which can also be applied to prove Theorems 3.1.4 and 3.1.5 for Neumann boundary conditions. Nevertheless, we refer the interested readers to [43] for a different proof and more detailed analysis on the constants in front of the error bounds.*

Proof of Theorem 3.1.1. Using a Taylor expansion, it is straightforward to show that

$$\tau_i = -\frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{\Delta x^2} - f(x_i) = O(\Delta x^2), \quad i = 2, \ldots, n-2.$$

Since $u(x_0) = u_a - \alpha \Delta x u'(a) + O(\Delta x^2)$ and α is a constant, $u_a = u(x_0) + O(\Delta x)$. Therefore,

$$\tau_1 = -\frac{u_a - 2u(x_1) + u(x_2)}{\Delta x^2} - f(x_1) = -\frac{u(x_0) - 2u(x_1) + u(x_2)}{\Delta x^2} - f(x_1) + O(\frac{1}{\Delta x}) = O(\frac{1}{\Delta x}).$$

Similarly, $\tau_{n-1} = O(\frac{1}{\Delta x})$.

Let $\boldsymbol{A}$ be the coefficient matrix of the linear system (3.1.5), $\boldsymbol{e} = (e_1, \ldots, e_{n-1})^T$, and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{n-1})^T$, then $\boldsymbol{A}\boldsymbol{e} = \boldsymbol{\tau}$. Using the explicit formula of the inverse of tridiagonal matrix $\boldsymbol{A}$ in [91], we know that

$$(\boldsymbol{A}^{-1})_{j,k} = \left(\frac{j(n-k)}{n} - \max(j-k, 0) \right) \Delta x^2, \quad 1 \le j, k \le n-1. \tag{3.1.6}$$

When $k = 1$, $(\boldsymbol{A}^{-1})_{j,1} = \left(1 - \frac{j}{n}\right) \Delta x^2 = O(\Delta x^2)$. When $k = n-1$, $(\boldsymbol{A}^{-1})_{j,n-1} = \frac{j}{n} \Delta x^2$ is at most $O(\Delta x^2)$. For $k = 2, \ldots, n-2$, $(\boldsymbol{A}^{-1})_{j,k}$ can be at most $O(\Delta x)$ due to the $\max(j-k, 0)\Delta x^2$ term. Since $\boldsymbol{e} = \boldsymbol{A}^{-1}\boldsymbol{\tau}$,

$$e_i = (\boldsymbol{A}^{-1})_{i,1}\tau_1 + \sum_{k=2}^{n-2} (\boldsymbol{A}^{-1})_{i,k}\tau_k + (\boldsymbol{A}^{-1})_{i,n-1}\tau_{n-1}. \tag{3.1.7}$$

Since $(\boldsymbol{A}^{-1})_{i,1}\tau_1 = O(\Delta x)$, $\sum_{k=2}^{n-2} (\boldsymbol{A}^{-1})_{i,k}\tau_k$ is at most $O(\Delta x^2)$, and $(\boldsymbol{A}^{-1})_{i,n-1}\tau_{n-1}$ is at most $O(\Delta x)$, $e_i = O(\Delta x)$ for $i = 1, 2, \ldots, n-1$. ■

From Theorem 3.1.1, constant extrapolations near the boundary points yield an $O(\frac{1}{\Delta x})$ (inconsistent) truncation error. Nevertheless, the overall error is $O(\Delta x)$ due to the $O(\Delta x^2)$ factors in the first and last columns of $\boldsymbol{A}^{-1}$. The intuition is that the error near the boundary equals the corresponding truncation error $O(\frac{1}{\Delta x})$ integrated twice and thus multiplied by a factor of Δx^2. It is natural for us to expect $O(\Delta x^2)$ error if we use linear extrapolation near the boundary points. To be more precise, we still assume that $a - x_0 = \alpha \Delta x$ and $x_n - b = \beta \Delta x$, and let

$$u_0 = \frac{u_a - \alpha u_1}{1 - \alpha}, \quad u_n = \frac{u_b - \beta u_{n-1}}{1 - \beta}. \tag{3.1.8}$$

Combining (3.1.8) with (3.1.2), we obtain a new discretization for problem (3.1.1),

$$\begin{aligned} -\frac{1}{\Delta x^2}\left(\frac{u_a}{1-\alpha} - \left(2+\frac{\alpha}{1-\alpha}\right)u_1 + u_2\right) &= f_1, \\ -\frac{u_{i-1}-2u_i+u_{i+1}}{\Delta x^2} &= f_i, \quad i=2,\ldots,n-2, \\ -\frac{1}{\Delta x^2}\left(\frac{u_b}{1-\beta} - \left(2+\frac{\beta}{1-\beta}\right)u_1 + u_2\right) &= f_{n-1}. \end{aligned} \tag{3.1.9}$$

The matrix form of (3.1.9) is

$$\frac{1}{\Delta x^2}\begin{pmatrix} 2+\frac{\alpha}{1-\alpha} & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2+\frac{\beta}{1-\beta} \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 + \frac{u_a}{(1-\alpha)\Delta x^2} \\ f_2 \\ \vdots \\ f_{n-2} \\ f_{n-1} + \frac{u_b}{(1-\beta)\Delta x^2} \end{pmatrix}. \tag{3.1.10}$$

Since we assumed that α and β are both less than 1, the first and last row of the coefficient matrix in (3.1.10) are well defined. Actually when α approaches 1, we can multiply the first row of (3.1.10) and get $u_1 = u_a$. This is trivially true since $x_1 = a$ when $\alpha = 1$. As for the truncation errors and errors of scheme (3.1.9), we have the following theorem.

Theorem 3.1.3 *(Jomaa and Macaskill 2005 [43]) With the same assumptions and notations as for Theorem 3.1.1, the truncation errors of scheme (3.1.9) are $\tau_1 = O(1)$, $\tau_{n-1} = O(1)$, and $\tau_i = O(\Delta x^2)$ for $i = 2,\ldots,n-2$; the errors are $e_i = O(\Delta x^2)$ for $i = 1,\ldots,n-1$.*

Proof. Like for the proof of Theorem 3.1.1, it is easy to show that $\tau_i = O(\Delta x^2)$ for $i = 2,\ldots,n-2$. Since $u_a = (1-\alpha)u(x_0) + \alpha u(x_1) + O(\Delta x^2)$,

$$\begin{aligned} \tau_1 &= -\frac{1}{\Delta x^2}\left(\frac{u_a}{1-\alpha} - \left(2+\frac{\alpha}{1-\alpha}\right)u(x_1) + u(x_2)\right) - f(x_1) \\ &= -\frac{1}{\Delta x^2}\left(u(x_0) - 2u(x_1) + u(x_2) + O(\Delta x^2)\right) - f(x_1) = O(1). \end{aligned}$$

Similarly, $\tau_{n-1} = O(1)$. Let $\boldsymbol{A}$ be the coefficient matrix of (3.1.10). Using the formula in [91],

$$(\boldsymbol{A}^{-1})_{j,k} = \left\{\frac{[j+\frac{\alpha}{1-\alpha}(j-1)][n-k+\frac{\beta}{1-\beta}(n-k-1)]}{n+(\frac{\alpha}{1-\alpha}+\frac{\beta}{1-\beta})(n-1)+\frac{\alpha}{1-\alpha}\frac{\beta}{1-\beta}(n-2)} - \max(j-k,0)\right\}\Delta x^2.$$

Again, with some calculations we can show that $(\boldsymbol{A}^{-1})_{j,1} = O(\Delta x^2)$, $(\boldsymbol{A}^{-1})_{j,n-1}$ is at most $O(\Delta x^2)$, and $(\boldsymbol{A}^{-1})_{j,k}$ is at most $O(\Delta x)$ for $k = 2,\ldots,n-2$. We can then use (3.1.7) to obtain that $e_i = O(\Delta x^2)$ for $i = 1,\ldots,n-1$. ■

3.1.2 Neumann boundary conditions

In this section we consider the shifted Poisson's equation with Neumann boundary conditions,

$$-u''(x) + cu(x) = f(x), \quad x \in (a, b), \tag{3.1.11a}$$

$$u'(a) = u'_a, \quad u'(b) = u'_b, \tag{3.1.11b}$$

where c is a positive constant. Suppose that the interval $[a, b]$ is embedded in a 1-D uniform grid x_0, x_1, ..., x_{n-1}, x_n such that $x_0 < a \leq x_1$ and $x_{n-1} \leq b < x_n$. For grid points x_1, ..., x_{n-1} in the interior of the interval, we can discretize (3.1.1a) using the centered finite difference scheme:

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} + cu_i = f_i, \quad i = 1, 2, \ldots, n-1, \tag{3.1.12}$$

where Δx is the grid size, $f_i = f(x_i)$, and u_i is the numerical value at x_i. Using linear approximations near the endpoints of the interval, we have

$$u_0 = u_1 - u'_a \Delta x, \quad u_n = u_{n-1} + u'_b \Delta x. \tag{3.1.13}$$

Combining (3.1.12) and (3.1.13), we obtain a complete discretization for problem (3.1.11),

$$\begin{aligned} -\frac{-u_1 + u_2}{\Delta x^2} + cu_1 &= f_1 - \frac{u'_a}{\Delta x}, \\ -\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} + cu_i &= f_i, \quad i = 2, \ldots, n-2, \\ -\frac{u_{n-2} - u_{n-1}}{\Delta x^2} + cu_{n-1} &= f_{n-1} + \frac{u'_b}{\Delta x}. \end{aligned} \tag{3.1.14}$$

The matrix form of (3.1.14) is

$$\frac{1}{\Delta x^2} \begin{pmatrix} 1 + c\Delta x^2 & -1 & & & \\ -1 & 2 + c\Delta x^2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 + c\Delta x^2 & -1 \\ & & & -1 & 1 + c\Delta x^2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 - \frac{u'_a}{\Delta x} \\ f_2 \\ \vdots \\ f_{n-2} \\ f_{n-1} + \frac{u'_b}{\Delta x} \end{pmatrix}. \tag{3.1.15}$$

The following theorem shows the consistency and convergence of scheme (3.1.14).

Theorem 3.1.4 *With the same assumptions and notations as for Theorem 3.1.1, the truncation errors of scheme (3.1.14) are $\tau_1 = O(1)$, $\tau_{n-1} = O(1)$, and $\tau_i = O(\Delta x^2)$ for $i = 2, \ldots, n-2$; the errors are $e_i = O(\Delta x)$ for $i = 1, \ldots, n-1$.*

Proof. Same as for Theorem 3.1.1, it is not difficult to show that $\tau_i = O(\Delta x^2)$ for $i = 2, \ldots, n-2$. Since $u(x_1) = u(a) + u'(a)(x_1 - a) + O(\Delta x^2)$ and $u(x_0) = u(a) + u'(a)(x_0 - a) + O(\Delta x^2)$, we know that $u(x_0) = u(x_1) - u'(a)\Delta x + O(\Delta x^2)$. Therefore,

$$\begin{aligned}\tau_1 &= -\frac{u(x_1) - u'(a)\Delta x - 2u(x_1) + u(x_2)}{\Delta x^2} + cu(x_1) - f(x_1) \\ &= -\frac{O(\Delta x^2) + u(x_0) - 2u(x_1) + u(x_2)}{\Delta x^2} + cu(x_1) - f(x_1) = O(1) + O(\Delta x^2) = O(1).\end{aligned}$$

Similarly, $\tau_{n-1} = O(1)$.

Let $\boldsymbol{A}$ be the coefficient matrix of linear system (3.1.15), $\boldsymbol{e} = (e_1, \ldots, e_{n-1})^T$, and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{n-1})$. Then $\boldsymbol{e} = \boldsymbol{A}^{-1}\boldsymbol{\tau}$. Using the formula in [91],

$$(\boldsymbol{A}^{-1})_{j,k} = -\frac{\Delta x^2}{\sin(\phi)}\left(\frac{\cos((j-\frac{1}{2})\phi)\cos((n-k-\frac{1}{2})\phi)}{\sin((n-1)\phi)} + \sin(\max(j-k,0)\phi)\right), \quad (3.1.16)$$

where $\phi = \arccos(1 + \frac{c}{2}\Delta x^2)$. We point out that ϕ is an imaginary number because $1 + \frac{c}{2}\Delta x^2$ is a real number larger than 1. It follows that $\sin(\phi) = \frac{e^{\mathrm{i}\phi} - e^{-\mathrm{i}\phi}}{2\mathrm{i}}$ is also an imaginary number, where i is the imaginary unit. Furthermore, $\sin(m\phi) = \frac{e^{\mathrm{i}m\phi} - e^{-\mathrm{i}m\phi}}{2\mathrm{i}}$ is also an imaginary number for any real number m. Nevertheless, formula (3.1.16) still yields $(\boldsymbol{A}^{-1})_{j,k}$ that are real numbers because the two cosine terms are real and the imaginary factors in the sine terms are cancelled out through multiplication. Now we show that $\sin(\phi) = O(\Delta x)$. This is because $\sin^2(\phi) = 1 - \cos^2(\phi) = -\frac{c}{2}\Delta x^2$ is of size $O(\Delta x^2)$[1] . Furthermore, one can show that $\sin((n-1)\phi) = O(1)$, and $\cos(j\phi) = O(1)$ for any j between 0 and n. Therefore, $(\boldsymbol{A}^{-1})_{j,k} = O(\Delta x)$ for $1 \le j, k \le n-1$. Using the relation between errors and truncation errors (3.1.7), we obtain that $e_i = O(\Delta x)$ for $i = 1, \ldots, n-1$. ■

To obtain an overall 2nd-order scheme we need quadratic approximations around the endpoints of the interval. We shall give a derivation for the approximations near point b; the case for point a is similar. Our goal is to represent u_n by $\{u_{n-2}, u_{n-1}, u'(b)\}$ so that $u'(b)$ is approximated by the derivative of a quadratic polynomial. We first build a quadratic polynomial using the stencil $\{x_{n-2}, x_{n-1}, x_n\}$,

$$q(x) = \frac{(x-x_{n-1})(x-x_n)}{(x_{n-2}-x_{n-1})(x_{n-2}-x_n)}u_{n-2} + \frac{(x-x_{n-2})(y-x_n)}{(x_{n-1}-x_{n-2})(x_{n-1}-x_n)}u_{n-1} + \frac{(x-x_{n-2})(y-x_{n-1})}{(x_n-x_{n-2})(x_n-x_{n-1})}u_n.$$

Taking the derivative of $q(x)$, we have

$$q'(x) = \frac{2x-(x_{n-1}+x_n)}{(x_{n-2}-x_{n-1})(x_{n-2}-x_n)}u_{n-2} + \frac{2x-(x_{n-2}+x_n)}{(x_{n-1}-x_{n-2})(x_{n-1}-x_n)}u_{n-1} + \frac{2x-(x_{n-2}+x_{n-1})}{(x_n-x_{n-2})(x_n-x_{n-1})}u_n.$$

[1] As discussed before, $\sin(\phi)$ is an imaginary number, so $\sin^2(\phi) < 0$.

Assuming the relationship $u'(b) = q'(b)$ and using the fact that $x_n - b = \beta\Delta x$, we get

$$u'(b) = \frac{1-2\beta}{2\Delta x}u_{n-2} - \frac{2-2\beta}{\Delta x}u_{n-1} + \frac{3-2\beta}{2\Delta x}u_n. \tag{3.1.17}$$

Solving for u_n from (3.1.17), we obtain that

$$u_n = \frac{2u'(b)\Delta x + (2\beta-1)u_{n-2} + (4-4\beta)u_{n-1}}{3-2\beta}. \tag{3.1.18}$$

Similarly, we have the following representation of u_0 if a quadratic approximation is used near point a,

$$u_0 = \frac{-2u'(a)\Delta x + (4-4\alpha)u_1 + (2\alpha-1)u_2}{3-2\alpha}. \tag{3.1.19}$$

Combining (3.1.12) with the quadratic approximations (3.1.18) and (3.1.19), we obtain the following discretization for problem (3.1.11),

$$\begin{aligned}
-\frac{-u_1+u_2}{\Delta x^2} + \frac{3-2\alpha}{2}cu_1 &= \frac{3-2\alpha}{2}f_1 - \frac{u'_a}{\Delta x}, \\
-\frac{u_{i-1}-2u_i+u_{i+1}}{\Delta x^2} + cu_i &= f_i, \quad i = 2,\ldots,n-2, \\
-\frac{u_{n-2}-u_{n-1}}{\Delta x^2} + \frac{3-2\beta}{2}cu_{n-1} &= \frac{3-2\beta}{2}f_{n-1} + \frac{u'_b}{\Delta x}.
\end{aligned} \tag{3.1.20}$$

The matrix form of (3.1.20) is

$$\frac{1}{\Delta x^2}\begin{pmatrix}
1+\frac{3-2\alpha}{2}c\Delta x^2 & -1 & & & \\
-1 & 2+c\Delta x^2 & -1 & & \\
 & \ddots & \ddots & \ddots & \\
 & & -1 & 2+c\Delta x^2 & -1 \\
 & & & -1 & 1+\frac{3-2\beta}{2}c\Delta x^2
\end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} \frac{3-2\alpha}{2}f_1 - \frac{u'_a}{\Delta x} \\ f_2 \\ \vdots \\ f_{n-2} \\ \frac{3-2\alpha}{2}f_{n-1} + \frac{u'_b}{\Delta x} \end{pmatrix}. \tag{3.1.21}$$

The following theorem shows that a 2nd-order convergence results from scheme (3.1.20).

Theorem 3.1.5 *With the same assumptions and notations as for Theorem 3.1.1, the truncation errors of scheme (3.1.20) are $\tau_1 = O(\Delta x)$, $\tau_{n-1} = O(\Delta x)$, and $\tau_i = O(\Delta x^2)$ for $i = 2,\ldots,n-2$; the errors are $e_i = O(\Delta x^2)$ for $i = 1,\ldots,n-1$.*

Proof. Same as for Theorem 3.1.1, it is not difficult to show that $\tau_i = O(\Delta x^2)$ for $i = 2,\ldots,n-2$. From the definition of the truncation error and using the fact that $-u''(x_1) + cu(x_1) = f(x_1)$, we obtain that

$$\begin{aligned}
\tau_1 &= -\frac{-u(x_1)+u(x_2)}{\Delta x^2} + \frac{3-2\alpha}{2}cu(x_1) - \frac{3-2\alpha}{2}f(x_1) + \frac{u'(a)}{\Delta x} \\
&= -\frac{-u'(a)\Delta x - u(x_1) + u(x_2)}{\Delta x^2} + cu(x_1) - f(x_1) + \frac{1-2\alpha}{2}u''(x_1) \\
&= -\frac{u(x_1) - u'(a)\Delta x - \frac{1-2\alpha}{2}u''(x_1)\Delta x^2 - 2u(x_1) + u(x_2)}{\Delta x^2} + cu(x_1) - f(x_1).
\end{aligned} \tag{3.1.22}$$

Using the Taylor expansion for $u'(x)$, we know that

$$u'(a) = u'(x_1) - (1-\alpha)u''(x_1)\Delta x + O(\Delta x^2). \tag{3.1.23}$$

Substituting (3.1.23) into (3.1.22), we have

$$\begin{aligned}
\tau_1 &= -\frac{\left(u(x_1) - u'(x_1)\Delta x + \frac{1}{2}u''(x_1)\Delta x^2\right) - 2u(x_1) + u(x_2)}{\Delta x^2} + cu(x_1) - f(x_1) + O(\Delta x) \\
&= -\frac{u(x_0) - 2u(x_1) + u(x_2) + O(\Delta x^3)}{\Delta x^2} + cu(x_1) - f(x_1) + O(\Delta x) \\
&= O(\Delta x^2) + O(\Delta x) = O(\Delta x)
\end{aligned} \tag{3.1.24}$$

Similarly, $\tau_{n-1} = O(\Delta x)$.

Let $\boldsymbol{A}$ be the coefficient matrix of linear system (3.1.21), $\boldsymbol{e} = (e_1, \ldots, e_{n-1})^T$, and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{n-1})$. Then $\boldsymbol{e} = \boldsymbol{A}^{-1}\boldsymbol{\tau}$. Using the formula in [91], we can prove again that $(\boldsymbol{A}^{-1})_{j,k} = O(\Delta x)$ for $1 \le j, k \le n-1$. Using the relation between errors and truncation errors (3.1.7), we obtain that $e_i = O(\Delta x^2)$ for $i = 1, \ldots, n-1$. ■

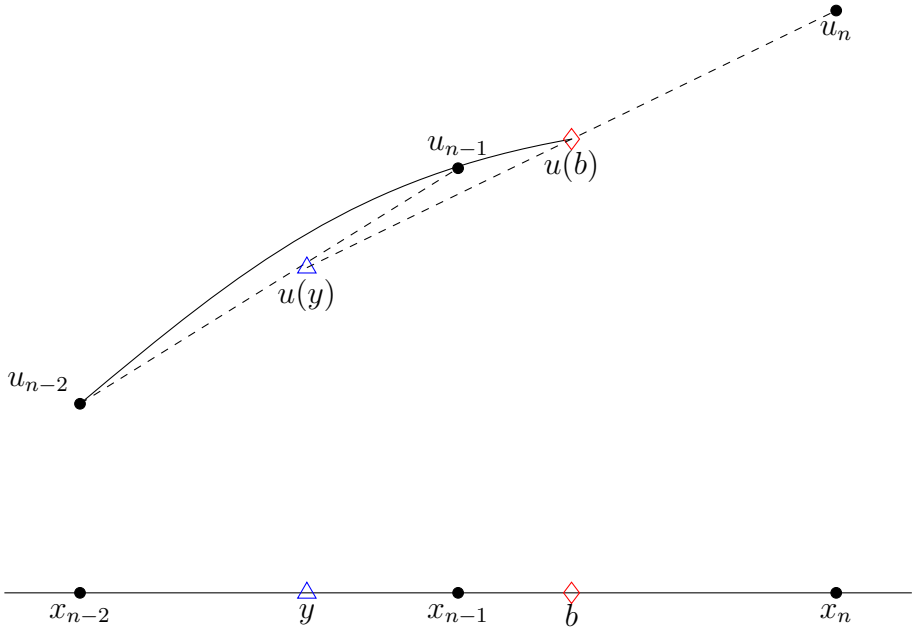
3.1.3 Extrapolation augmented with interpolation

Until now we have only used values of grid points to extrapolate values of ghost points. Actually the points used for extrapolations do not have to be grid points in the first place. We can choose any points in the interior or on the boundary of the interval as extrapolation points, and then interpolate values of these extrapolation points from values of grid points. As we shall see next, this new extrapolation strategy yields similar results to the normal extrapolation method. However, the new method can be easily generalized to handle boundary conditions in higher dimensional spaces as we will see in Section 3.2.

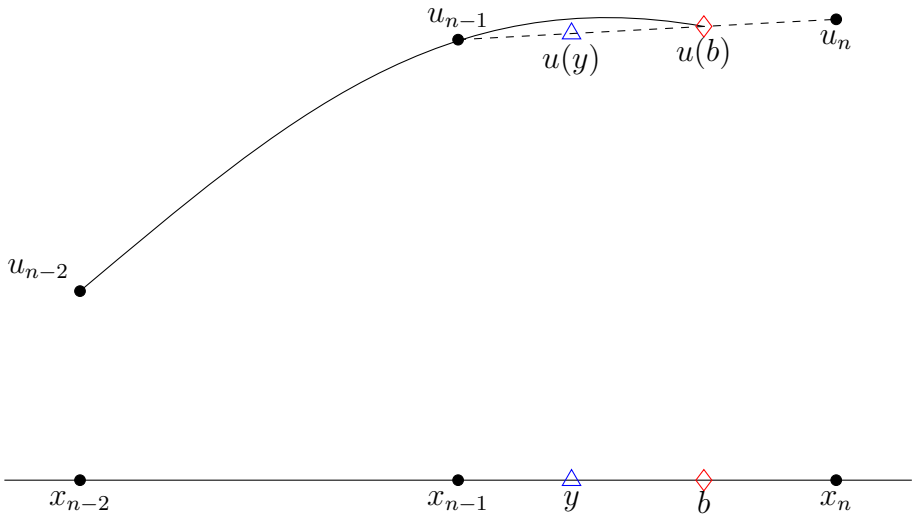
We first look at an example (Figure 3.1) of using linear extrapolation to impose a Dirichlet boundary condition augmented with linear interpolation in 1-D. The horizontal axis represents positions of points, and the vertical direction represents values of u. Our goal is to represent the value u_n at the ghost point x_n by values of the grid points and the boundary point b. Previously we used the grid point x_{n-1} and the boundary point b to construct a linear polynomial, and then evaluated it at x_n to obtain u_n. Now instead of x_{n-1} we use $y = 2b - x_n$ which is the mirror point of x_n with respect to b. If we were able to know $u(y)$, then via linear extrapolation

$$u_n = 2u(b) - u(y). \tag{3.1.25}$$

However, since y is typically not a grid point, we obtain its value via interpolation of grid point values. We then substitute the representation of $u(y)$ by grid point values into (3.1.25) to obtain u_n, and solve for u_n if the representation of $u(y)$ involves u_n.



(a) Situation when $\beta \geq \frac{1}{2}$ $(b - x_{n-1} < \frac{\Delta x}{2})$. Here $x_n - b = b - y$.



(b) Situation when $\beta < \frac{1}{2}$ $(b - x_{n-1} > \frac{\Delta x}{2})$. Here $x_n - b = b - y$.

Figure 3.1: Illustration of linear extrapolation augmented with linear interpolation to impose Dirichlet boundary conditions in 1-D.

For ease of illustration we use linear interpolation to obtain $u(y)$. The details of the interpolation for $u(y)$ are as follows. Depending on the position of x_n, y might fall between x_{n-2} and x_{n-1} or between x_{n-1} and x_n. Using the same notation as in Theorem 3.1.1, let $x_n - b = \beta\Delta x$. If $\beta \geq \frac{1}{2}$ ($b - x_{n-1} \leq \frac{\Delta x}{2}$), then y lies between x_{n-2} and x_{n-1} (see Figure 3.1(a)). In this case, we can approximate $u(y)$ through linear interpolation of u_{n-2} and u_{n-1}:

$$u(y) \approx \frac{x_{n-1} - y}{\Delta x} u_{n-2} + \frac{y - x_{n-2}}{\Delta x} u_{n-1} = (2\beta - 1)u_{n-2} + (2 - 2\beta)u_{n-1}. \tag{3.1.26}$$

Combining (3.1.25) and (3.1.26), we obtain the following approximation of u_n using the values of grid points,

$$u_n = 2u(b) - [(2\beta - 1)u_{n-2} + (2 - 2\beta)u_{n-1}], \quad \text{when } \beta \geq \frac{1}{2}. \tag{3.1.27}$$

If $\beta < \frac{1}{2}$ ($b - x_{n-1} > \frac{\Delta x}{2}$), then y lies between x_{n-1} and x_n (see Figure 3.1(b)). In this case, we would like to approximate $u(y)$ via linear interpolation of u_{n-1} and u_n:

$$u(y) \approx \frac{x_n - y}{\Delta x} u_{n-1} + \frac{y - x_{n-1}}{\Delta x} u_n = 2\beta u_{n-1} + (1 - 2\beta)u_n. \tag{3.1.28}$$

Combining (3.1.25) and (3.1.28), u_n satisfies the following relationship,

$$u_n = 2u(b) - [2\beta u_{n-1} + (1 - 2\beta)u_n]. \tag{3.1.29}$$

Since the approximation of $u(y)$ involves u_n, the right hand side of (3.1.29) also contains u_n. Solving u_n from (3.1.29) we get

$$u_n = \frac{u(b) - \beta u_{n-1}}{1 - \beta}, \quad \text{when } \beta < \frac{1}{2}. \tag{3.1.30}$$

Note that (3.1.30) can be directly obtained by constructing a linear polynomial through x_{n-1} and b and then evaluating at x_n. The above extrapolation method depending on the relative position of b is reasonable in the following sense. If b is not close to x_{n-1} ($\beta < \frac{1}{2}$), we use the normal extrapolation scheme (3.1.30) to obtain u_n. If b is close to x_{n-1} ($\beta \geq \frac{1}{2}$), especially when b is approaching x_{n-1} ($\beta \to 1$), (3.1.30) is not a stable scheme to compute u_n; in this case, we use the alternative scheme (3.1.27) which involves two interior grid points and the boundary point b.

The above procedure can be summarized as follows. We first compute the mirror point $y = 2b - x_n$ of x_n with respect to the boundary b. We then find the linear interpolation stencil containing y and represent $u(y)$ by interpolation of grid point values. Finally we compute u_n through linear extrapolation $u_n = 2u(b) - u(y)$, and

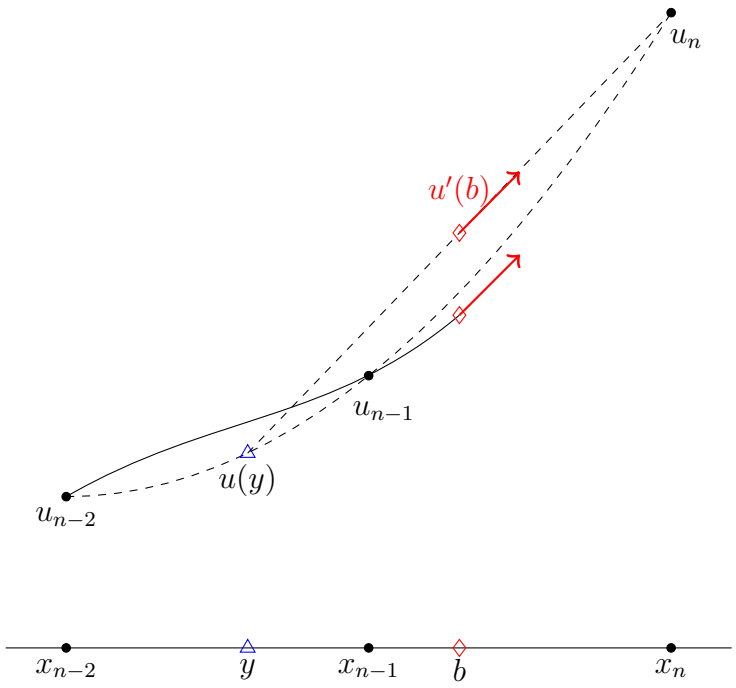


Figure 3.2: Illustration of 2nd-order approximation of Neumann boundary condition augmented with quadratic interpolation in 1-D.

solve for u_n if the representation of $u(y)$ involves u_n. The whole procedure seems a bit tedious for this simple 1-D problem. However, as we shall see in Section 3.2.1, this procedure can be extended in a straightforward manner to handle Dirichlet boundary conditions in higher dimensional spaces.

Similarly, we can find a 2nd-order approximation for Neumann boundary conditions using the centered finite difference scheme augmented with quadratic interpolations. As shown in Figure 3.2, we find the point $y = 2b - x_n$, and approximate $u'(b)$ via the centered finite difference scheme,

$$u'(b) = \frac{u_n - u(y)}{2\beta\Delta x}. \tag{3.1.31}$$

Solving (3.1.31) for u_n, we obtain that

$$u_n = 2\beta u'(b)\Delta x + u(y). \tag{3.1.32}$$

Using the quadratic interpolation stencil $\{x_{n-2}, x_{n-1}, x_n\}$, $u(y)$ can be approximated with the following Lagrange polynomial interpolation,

$$\begin{aligned} u(y) &\approx \frac{(y-x_{n-1})(y-x_n)}{(x_{n-2}-x_{n-1})(x_{n-2}-x_n)}u_{n-2} + \frac{(y-x_{n-2})(y-x_n)}{(x_{n-1}-x_{n-2})(x_{n-1}-x_n)}u_{n-1} + \frac{(y-x_{n-2})(y-x_{n-1})}{(x_n-x_{n-2})(x_n-x_{n-1})}u_n \\ &= (2\beta^2-\beta)u_{n-2} + (-4\beta^2+4\beta)u_{n-1} + (2\beta^2-3\beta+1)u_n \end{aligned} \tag{3.1.33}$$

Substituting (3.1.33) into (3.1.32), and solving for u_n, we obtain that

$$u_n = \frac{2u'(b)\Delta x + (2\beta - 1)u_{n-2} + (4 - 4\beta)u_{n-1}}{3 - 2\beta}. \tag{3.1.34}$$

Again the above procedure seems unnecessary for 1-D problems since (3.1.34) is exactly the same as (3.1.18) which was obtained in a more straightforward manner. However, as we shall see in Section 3.2.2, it can be easily extended to handle Neumann boundary conditions in higher dimensional spaces.

3.2 Ghost point approach for domains in higher dimensional spaces

In this section, we extend the ideas in Section 3.1.3 to handle general boundary conditions in higher dimensions. We start with an example of obtaining values of ghost points via linear extrapolation and interpolation for Dirichlet boundary conditions in two dimensions. We shall study how to represent values of ghost points using values of the remaining grid points, and construct a complete scheme to solve Poisson's equation on a domain in $\mathbb{R}^n$ ($n \geq 2$) with smooth boundary. Similar ideas are then applied to solve elliptic equations with Neumann and Robin boundary conditions.

3.2.1 Dirichlet boundary conditions

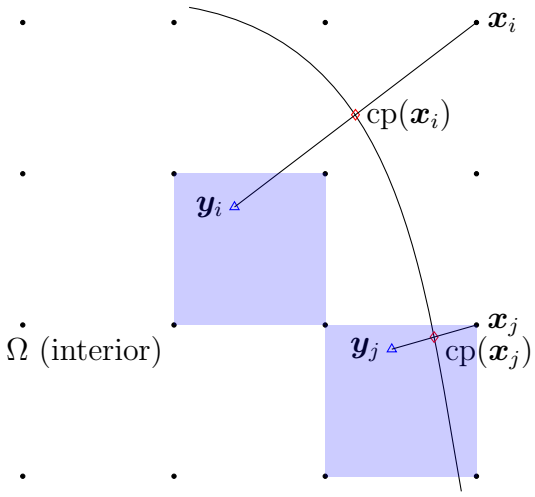


Figure 3.3: Illustration of linear extrapolation augmented with linear interpolation to impose Dirichlet boundary conditions for a two dimensional domain.

Figure 3.3 illustrates how to obtain the values of ghost points using linear extrapolations augmented by linear interpolations. As discussed in Section 3.1.1, we define ghost points to be grid points strictly outside the domain Ω. For each ghost point $\boldsymbol{x}_i$, let $\mathrm{cp}(\boldsymbol{x}_i)$ be the point on the boundary of the domain which is closest to $\boldsymbol{x}_i$ in Euclidean distance, and let $\boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i$ be the mirror point of $\boldsymbol{x}_i$ with respect to $\mathrm{cp}(\boldsymbol{x}_i)$. Assuming the Dirichlet boundary condition $u(\boldsymbol{x}) = g(\boldsymbol{x})$ where $\boldsymbol{x} \in \partial\Omega$, for each ghost point $\boldsymbol{x}_i$, we know the value $u(\mathrm{cp}(\boldsymbol{x}_i)) = g(\mathrm{cp}(\boldsymbol{x}_i))$ at its closest point $\mathrm{cp}(\boldsymbol{x}_i)$. We can then approximate $u(\boldsymbol{x}_i)$ by values of interior and boundary points using linear extrapolation,

$$u(\boldsymbol{x}_i) = 2u(\mathrm{cp}(\boldsymbol{x}_i)) - u(\boldsymbol{y}_i). \tag{3.2.1}$$

Since $\boldsymbol{y}_i$ is typically not a grid point, we obtain its value through linear interpolation of grid point values. Depending on the position of the ghost point, the interpolation stencil of the mirror point might or might not contain ghost points. For example, in Figure 3.3, the interpolation stencil of $\boldsymbol{y}_i$ contains only interior grid points, while the interpolation stencil of $\boldsymbol{y}_j$ contains both interior and ghost points. If we replace $u(\boldsymbol{y}_i)$ with the linear interpolation of grid point values in equation (3.2.1) for every ghost point $\boldsymbol{x}_i$, then we obtain a system of linear equations. We regard the values of ghost points as unknowns, and the values of the remaining grid points (including grid points in the interior and on the boundary) as knowns. Solving this linear system, we can represent the values of ghost points by values of grid points in the interior and on the boundary.

Notice that the representation of values at ghost points by values of the remaining grid points only depends on the boundary conditions; it does not rely on the equation we want to solve in Ω, nor on the scheme we use to discretize the interior equation. Therefore, we are able to separate the discretization of the PDE and the boundary conditions. We do not need to explicitly make special adjustment to the difference scheme for grid points near the boundary.

Consider the following Poisson's equation in $\Omega \subset \mathbb{R}^n$,

$$-\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{3.2.2a}$$

$$u(\boldsymbol{x}) = g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega. \tag{3.2.2b}$$

For all the grid points in the interior and on the boundary of Ω, we use the standard centered finite difference scheme to discretize (3.2.2a); for all the ghost points, we use the extrapolation method mentioned before to represent their values by the values of

the remaining grid points according to the boundary condition (3.2.2b). To be more precise, we shall introduce a matrix formulation of this strategy.

Let $\boldsymbol{G}$ be the set of all ghost points. We first need to determine what $\boldsymbol{G}$ is. If we use linear extrapolation to impose the Dirichlet boundary condition, then the set $\boldsymbol{G}$ should contain every ghost point which can appear in the interpolation stencil for some $\boldsymbol{y}_i$. Since $\boldsymbol{y}_i$ is in the interior of Ω, it is enough to choose the set $\boldsymbol{G}$ containing every ghost point which can appear in the interpolation stencil for some point on $\partial\Omega$. In practice, we choose

$$\boldsymbol{G} = \{\boldsymbol{x}_i \mid 0 < \operatorname{dist}(\boldsymbol{x}_i, \Omega) \leq \frac{p+1}{2}\sqrt{d}\Delta x\},$$

where p is the augmenting interpolation degree, d is the dimension and Δx is the grid size. This choice of $\boldsymbol{G}$ also works for Neumann/Robin boundary conditions. Furthermore, it is suitable for higher order approximations of boundary conditions, as we shall discuss in Section 3.2.4.

Let $\boldsymbol{u}$ be the column vector storing the values of all grid points, let $\boldsymbol{u}_G$ be the column vector storing the values of ghost points in $\boldsymbol{G}$, let $\boldsymbol{u}_I$ be the column vector storing the values of non-ghost points[2]. If we number the grid points properly, then

$$\boldsymbol{u} = \begin{pmatrix} \boldsymbol{u}_I \\ \boldsymbol{u}_G \end{pmatrix}.$$

Let n_i be the length of $\boldsymbol{u}_I$, n_g be the length of $\boldsymbol{u}_G$, and $n = n_i + n_g$ be the length of $\boldsymbol{u}$. Let $\boldsymbol{f}_I$ be the column vector storing the values of function f evaluated at interior and boundary grid points. Let $\boldsymbol{L}$ be the Laplacian matrix corresponding to the Laplacian operating on the non-ghost points. Note that $\boldsymbol{L}$ is a rectangular matrix of size $n_i \times n$ since the Laplacian stencil for points close to or on the boundary also involves ghost points. To make the structure of $\boldsymbol{L}$ more clear, we denote $\boldsymbol{L}$ as $\boldsymbol{L} = (\boldsymbol{L}_{II}\ \boldsymbol{L}_{IG})$, where $\boldsymbol{L}_{II}$ is an $n_i \times n_i$ matrix corresponding to the contributions of interior and boundary grid points, and $\boldsymbol{L}_{IG}$ is an $n_i \times n_g$ matrix corresponding to the contributions of ghost points. Then the discretization of (3.2.2a) becomes

$$-\boldsymbol{L}\boldsymbol{u} = -(\boldsymbol{L}_{II}\ \boldsymbol{L}_{IG}) \begin{pmatrix} \boldsymbol{u}_I \\ \boldsymbol{u}_G \end{pmatrix} = \boldsymbol{f}_I. \tag{3.2.3}$$

Let $\operatorname{cp}(\boldsymbol{G})$ be the set of the closest points of ghost points, and $\boldsymbol{Y}$ be the set of mirror points of ghost points with respect to the boundary: $\boldsymbol{Y} = \{\boldsymbol{y} | \boldsymbol{y} = 2\operatorname{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i,\ \boldsymbol{x}_i \in \boldsymbol{G}\}$. Let $\boldsymbol{u}_Y$ be the column vector storing the values of points in $\boldsymbol{Y}$. Let $\boldsymbol{g}_{\operatorname{cp}(\boldsymbol{G})}$ be the

[2]Here "non-ghost" points refer to grid points which are not ghost points. Non-ghost points could appear either in the interior or on the boundary of the domain.

column vector storing the values of function g evaluated at the points in set $\text{cp}(\boldsymbol{G})$. Let $\boldsymbol{E}_Y^p$ be an $n_g \times n$ matrix that uses degree p polynomial to interpolate the values of points in $\boldsymbol{Y}$ from values of grid points:

$$\boldsymbol{u}_Y = \boldsymbol{E}_Y^p \boldsymbol{u} \tag{3.2.4}$$

With the definition of $\boldsymbol{E}_Y^p$, we can discretize (3.2.1) as $\boldsymbol{u}_G = 2\boldsymbol{g}_{\text{cp}(\boldsymbol{G})} - \boldsymbol{E}_Y^p \boldsymbol{u}$, or

$$\boldsymbol{u}_G + \boldsymbol{E}_Y^p \boldsymbol{u} = 2\boldsymbol{g}_{\text{cp}(\boldsymbol{G})}. \tag{3.2.5}$$

Let

$$\boldsymbol{I}_G = \left(\boldsymbol{O}_{n_g \times n_i} \ \boldsymbol{I}_{n_g \times n_g}\right), \tag{3.2.6}$$

then equation (3.2.5) becomes

$$(\boldsymbol{I}_G + \boldsymbol{E}_Y^p)\boldsymbol{u} = 2\boldsymbol{g}_{\text{cp}(\boldsymbol{G})}.$$

In other words, the discretization of (3.2.2b) becomes

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{g}_{\text{cp}(\boldsymbol{G})}, \ \text{where } \boldsymbol{B} = \frac{1}{2}(\boldsymbol{I}_G + \boldsymbol{E}_Y^p). \tag{3.2.7}$$

For convenience, we partition the $n_g \times n$ matrix $\boldsymbol{B}$ into two parts,

$$\boldsymbol{B} = \left(\boldsymbol{B}_{GI} \ \boldsymbol{B}_{GG}\right).$$

Combining (3.2.3) and (3.2.7), we obtain a complete discretization for (3.2.2),

$$\begin{pmatrix} -\boldsymbol{L} \\ \boldsymbol{B} \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} -\boldsymbol{L}_{II} & -\boldsymbol{L}_{IG} \\ \boldsymbol{B}_{GI} & \boldsymbol{B}_{GG} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_I \\ \boldsymbol{u}_G \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_I \\ \boldsymbol{g}_{\text{cp}(\boldsymbol{G})} \end{pmatrix}, \tag{3.2.8}$$

where $\boldsymbol{B}$ is defined by (3.2.7). We can directly solve the linear system (3.2.8) to obtain $\boldsymbol{u}$. Alternatively, as discussed before, we could first represent values of ghost points by values of the remaining grid points, i.e., representing $\boldsymbol{u}_G$ by $\boldsymbol{u}_I$,

$$\boldsymbol{u}_G = -\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI}\boldsymbol{u}_I + \boldsymbol{B}_{GG}^{-1}\boldsymbol{g}_{\text{cp}(\boldsymbol{G})}; \tag{3.2.9}$$

and then substitute (3.2.9) back into (3.2.8), reducing it to a system containing only $\boldsymbol{u}_I$ as unknowns,

$$\left(-\boldsymbol{L}_{II} + \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI}\right)\boldsymbol{u}_I = \boldsymbol{f}_I + \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{g}_{\text{cp}(\boldsymbol{G})}. \tag{3.2.10}$$

Notice that (3.2.10) is a generalization of scheme (3.1.10) for 1-D Poisson's equation with Dirichlet boundary conditions. Both schemes (3.1.10) and (3.2.10) discretize the Laplacian operator for non-ghost points using the centered finite difference scheme,

and extrapolate the values of ghost points from values of non-ghost points so that the Dirichlet boundary condition is satisfied. We expect that (3.2.10) would have similar consistency properties to (3.1.10). However, the analysis of stability and convergence for scheme (3.2.10) is much more challenging and left as future work. In this thesis, we focus on the numerical study of scheme (3.2.10). As we shall see in Section 3.2.5, 2nd-order convergence results are observed if centered finite differences for the Laplacian and linear extrapolation for the Dirichlet boundary condition are used.

3.2.2 Neumann boundary conditions

In Section 3.2.1 we discussed how to handle Dirichlet boundary conditions for the Poisson problem in a domain with a smooth boundary. We shall use similar ideas to deal with Neumann boundary conditions in this section. As we have seen in the previous section, the discretization of the PDE and the boundary condition can be separated. The only question is how to extrapolate values of ghost points from values of the remaining grid points according to the boundary conditions.

We consider the shifted Poisson's equation with Neumann boundary conditions,

$$-\Delta u(\boldsymbol{x}) + cu(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{3.2.11a}$$

$$\frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x}) = h(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega, \tag{3.2.11b}$$

where c is a positive constant, and $\boldsymbol{n}$ is the outward pointing normal vector.

Although we are dealing with Neumann boundary conditions, Figure 3.3 still illustrates the necessary idea. We shall approximate the normal derivative at $\mathrm{cp}(\boldsymbol{x}_i)$ via centered finite differences acting along the line jointing $\boldsymbol{x}_i$ to $\boldsymbol{y}_i$,

$$h(\mathrm{cp}(\boldsymbol{x}_i)) = \frac{\partial u}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i)) \approx \frac{u(\boldsymbol{x}_i) - u(\boldsymbol{y}_i)}{\|\boldsymbol{x}_i - \boldsymbol{y}_i\|_2}, \tag{3.2.12}$$

where $\boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i$. Recall that the ghost point $\boldsymbol{x}_i$ is strictly outside Ω, so $\|\boldsymbol{x}_i - \boldsymbol{y}_i\|_2 > 0$ and the right hand side of (3.2.12) is well defined. However, $\|\boldsymbol{x}_i - \boldsymbol{y}_i\|_2$ might almost be zero when $\boldsymbol{x}_i$ is very close to $\partial\Omega$. To avoid the numerical instability of dividing by tiny numbers, we multiply (3.2.12) by $\|\boldsymbol{x}_i - \boldsymbol{y}_i\|_2 = 2\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2$ and obtain the following approximation for $u(\boldsymbol{x}_i)$,

$$u(\boldsymbol{x}_i) = u(\boldsymbol{y}_i) + 2h(\mathrm{cp}(\boldsymbol{x}_i))\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2. \tag{3.2.13}$$

Notice that when $\boldsymbol{x}_i$ approaches $\partial\Omega$ ($\boldsymbol{x}_i \to \boldsymbol{y}_i$), equation (3.2.13) simply reduces to $u(\boldsymbol{x}_i) = u(\boldsymbol{y}_i)$, which is trivially true. Let $\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}$ be the column vector storing the

values of function h evaluated at the closest points of the ghost grid points. Let $\boldsymbol{d}$ be the column vector storing the distances between the ghost points and their closest points,

$$\boldsymbol{d} = (\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2,\ i = 1, \ldots, n_g)\,.$$

Let $\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}$ be the column vector storing the product of the corresponding entries of $\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}$ and $\boldsymbol{d}$,

$$\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d} = (h(\mathrm{cp}(\boldsymbol{x}_i))d_i,\ i = 1, \ldots, n_g)\,.$$

Let $\boldsymbol{I}_G$ be defined by (3.2.6), and define $\boldsymbol{E}_Y^q$ as the degree q interpolation matrix so that $\boldsymbol{u}_Y = \boldsymbol{E}_Y^q \boldsymbol{u}$. Then the matrix form of the discretization of (3.2.13) is

$$(\boldsymbol{I}_G - \boldsymbol{E}_Y^q)\boldsymbol{u} = 2\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d},$$

which can be written as

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d},\ \text{where } \boldsymbol{B} = \frac{1}{2}(\boldsymbol{I}_G - \boldsymbol{E}_Y^q). \tag{3.2.14}$$

The disretization for the interior equation (3.2.11a) is again standard. Let $\boldsymbol{L}$, $\boldsymbol{u}$ and $\boldsymbol{f}_I$ be defined similarly in Section 3.2.1, and let $\boldsymbol{I}_I = \begin{pmatrix}\boldsymbol{I}_{n_i \times n_i} & \boldsymbol{O}_{n_i \times n_g}\end{pmatrix}$. The discretization of (3.2.11a) in matrix form is

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}_I,\ \text{where } \boldsymbol{A} = -\boldsymbol{L} + c\boldsymbol{I}_I. \tag{3.2.15}$$

Combining (3.2.15) and (3.2.14), and using a similar matrix partition to the one in Section 3.2.1, we obtain the following complete discretization for problem (3.2.11),

$$\begin{pmatrix}\boldsymbol{A}\\ \boldsymbol{B}\end{pmatrix}\boldsymbol{u} = \begin{pmatrix}\boldsymbol{A}_{II} & \boldsymbol{A}_{IG}\\ \boldsymbol{B}_{GI} & \boldsymbol{B}_{GG}\end{pmatrix}\begin{pmatrix}\boldsymbol{u}_I\\ \boldsymbol{u}_G\end{pmatrix} = \begin{pmatrix}\boldsymbol{f}_I\\ \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}\end{pmatrix}, \tag{3.2.16}$$

where $\boldsymbol{A}$ is defined in (3.2.15) and $\boldsymbol{B}$ is defined in (3.2.14). Again we can either solve (3.2.16) directly, or reduce it to an $n_i \times n_i$ linear system via representing $\boldsymbol{u}_G$ by $\boldsymbol{u}_I$. To be more precise, we first represent $\boldsymbol{u}_G$ by $\boldsymbol{u}_I$,

$$\boldsymbol{u}_G = -\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI}\boldsymbol{u}_I + \boldsymbol{B}_{GG}^{-1}(\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}); \tag{3.2.17}$$

and then substitute (3.2.17) back into (3.2.16) to obtain

$$\left(\boldsymbol{A}_{II} - \boldsymbol{A}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI}\right)\boldsymbol{u}_I = \boldsymbol{f}_I - \boldsymbol{A}_{IG}\boldsymbol{B}_{GG}^{-1}(\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}). \tag{3.2.18}$$

3.2.3 Robin boundary conditions

In this section we consider Poisson's equation with Robin boundary conditions,

$$-\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{3.2.19a}$$

$$u(\boldsymbol{x}) + \varphi(\boldsymbol{x})\frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x}) = h(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega. \tag{3.2.19b}$$

The method to deal with Robin boundary conditions is a combination of the method to handle Dirichlet and Neumann boundary conditions. For each ghost point $\boldsymbol{x}_i$, we evaluate the boundary condition (3.2.19b) at its closest point $\mathrm{cp}(\boldsymbol{x}_i)$,

$$u(\mathrm{cp}(\boldsymbol{x}_i)) + \varphi(\mathrm{cp}(\boldsymbol{x}_i))\frac{\partial u}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i)) = h(\mathrm{cp}(\boldsymbol{x}_i)). \tag{3.2.20}$$

Let $\boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i$, then we can approximate $u(\mathrm{cp}(\boldsymbol{x}_i))$ by $\frac{1}{2}(u(\boldsymbol{x}_i) + u(\boldsymbol{y}_i))$, and approximate $\frac{\partial u}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i))$ by $\frac{u(\boldsymbol{x}_i)-u(\boldsymbol{y}_i)}{2\|\boldsymbol{x}_i-\mathrm{cp}(\boldsymbol{x}_i)\|_2}$. Substituting these approximations into (3.2.20), we obtain that

$$\frac{1}{2}(u(\boldsymbol{x}_i) + u(\boldsymbol{y}_i)) + \varphi(\mathrm{cp}(\boldsymbol{x}_i))\frac{u(\boldsymbol{x}_i) - u(\boldsymbol{y}_i)}{2\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2} = h(\mathrm{cp}(\boldsymbol{x}_i)). \tag{3.2.21}$$

Rearranging (3.2.21) we obtain that

$$\begin{aligned} & \big(\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2 + \varphi(\mathrm{cp}(\boldsymbol{x}_i))\big)u(\boldsymbol{x}_i) + \big(\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2 - \varphi(\mathrm{cp}(\boldsymbol{x}_i))\big)u(\boldsymbol{y}_i) \\ = \;& 2\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2 h(\mathrm{cp}(\boldsymbol{x}_i)). \end{aligned} \tag{3.2.22}$$

Let $\boldsymbol{D}$ be an $n_g \times n_g$ diagonal matrix whose diagonal entries are the distances between the ghost points and their closest points,

$$\boldsymbol{D} = \begin{pmatrix} \|\boldsymbol{x}_1 - \mathrm{cp}(\boldsymbol{x}_1)\|_2 & & \\ & \ddots & \\ & & \|\boldsymbol{x}_{n_g} - \mathrm{cp}(\boldsymbol{x}_{n_g})\|_2 \end{pmatrix}_{n_g \times n_g}. \tag{3.2.23}$$

Let $\boldsymbol{\Phi}$ be an $n_g \times n_g$ diagonal matrix whose diagonal entries are the values of the function φ evaluated at the closest points of the ghost points,

$$\boldsymbol{\Phi} = \begin{pmatrix} \varphi(\mathrm{cp}(\boldsymbol{x}_1)) & & \\ & \ddots & \\ & & \varphi(\mathrm{cp}(\boldsymbol{x}_{n_g})) \end{pmatrix}_{n_g \times n_g}. \tag{3.2.24}$$

Let $\boldsymbol{I}_G$, $\boldsymbol{E}_Y^q$ and $\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}$ be defined as in Section 3.2.2, then the matrix form of discretization (3.2.22) becomes

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}, \text{ where } \boldsymbol{B} = \frac{1}{2}[(\boldsymbol{D} + \boldsymbol{\Phi})\boldsymbol{I}_G + (\boldsymbol{D} - \boldsymbol{\Phi})\boldsymbol{E}_Y^q]. \tag{3.2.25}$$

As in the Dirichlet boundary condition case, for the non-exterior grid points we have

$$-\boldsymbol{L}\boldsymbol{u} = -\begin{pmatrix}\boldsymbol{L}_{II} & \boldsymbol{L}_{IG}\end{pmatrix}\begin{pmatrix}\boldsymbol{u}_I \\ \boldsymbol{u}_G\end{pmatrix} = \boldsymbol{f}_I. \tag{3.2.26}$$

Combining (3.2.25) and (3.2.26), we obtain the matrix form for the discretization of (3.2.19),

$$\begin{pmatrix}-\boldsymbol{L} \\ \boldsymbol{B}\end{pmatrix}\boldsymbol{u} = \begin{pmatrix}-\boldsymbol{L}_{II} & -\boldsymbol{L}_{IG} \\ \boldsymbol{B}_{GI} & \boldsymbol{B}_{GG}\end{pmatrix}\begin{pmatrix}\boldsymbol{u}_I \\ \boldsymbol{u}_G\end{pmatrix} = \begin{pmatrix}\boldsymbol{f}_I \\ \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}\end{pmatrix}, \tag{3.2.27}$$

where $\boldsymbol{B}$ is defined by (3.2.25). The linear system (3.2.27) has the same structure as (3.2.8) and (3.2.16). Therefore, we can either solve (3.2.27) directly or reduce it to a smaller system with only non-exterior grid points values as unknowns: We first represent $\boldsymbol{u}_G$ by $\boldsymbol{u}_I$,

$$\boldsymbol{u}_G = -\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI}\boldsymbol{u}_I + \boldsymbol{B}_{GG}^{-1}(\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}); \tag{3.2.28}$$

and then substitute (3.2.28) back into (3.2.27) to obtain

$$\left(-\boldsymbol{L}_{II} + \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI}\right)\boldsymbol{u}_I = \boldsymbol{f}_I + \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}(\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}). \tag{3.2.29}$$

3.2.4 Higher order approximations for boundary conditions

From the analysis for the one dimensional case in Section 3.1, we expect that linear extrapolation augmented with linear interpolation would be enough to yield a 2nd-order scheme for Dirichlet boundary conditions in higher dimensions, and 2nd-order approximation for the normal derivative combined with quadratic interpolation for values of the mirror points would be enough to give us a 2nd-order scheme for Neumann boundary conditions. The numerical results in Section 3.2.5 roughly agree with these expectations. Nevertheless, an extra order of approximation to the boundary condition typically helps us to obtain numerical results with smaller errors and better convergence in practice. Therefore, we discuss how to obtain higher order approximations for boundary conditions in this section.

Since the extrapolation (or approximation) step and the augmenting interpolation step are separated, we can first make an extrapolation or approximation along the one-dimensional normal direction, and then interpolate values of the points needed for extrapolation or approximation. For example, Figure 3.4 illustrates quadratic extrapolation for Dirichlet boundary conditions. For each ghost point $\boldsymbol{x}_i$, we first find its closest point $\mathrm{cp}(\boldsymbol{x}_i)$ on $\partial\Omega$, then define $\boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i$, and finally let

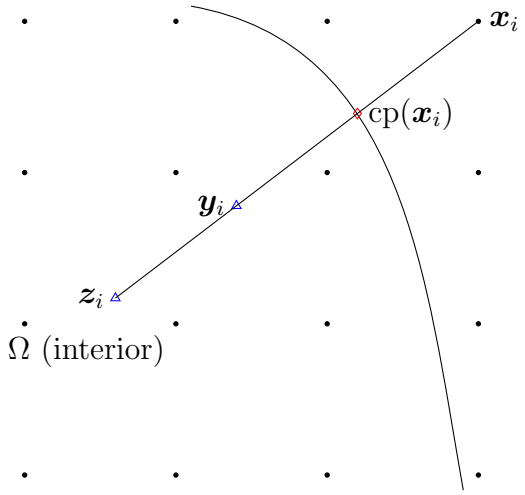


Figure 3.4: Illustration of higher order approximations for boundary conditions.

$\boldsymbol{z}_i = 2\boldsymbol{y}_i - \mathrm{cp}(\boldsymbol{x}_i)$. We construct a quadratic polynomial along the normal direction by interpolating $(\boldsymbol{x}_i, u(\boldsymbol{x}_i))$, $(\boldsymbol{y}_i, u(\boldsymbol{y}_i))$ and $(\boldsymbol{z}_i, u(\boldsymbol{z}_i))$, and then we evaluate the polynomial at $\mathrm{cp}(\boldsymbol{x}_i)$ to obtain

$$u(\mathrm{cp}(\boldsymbol{x}_i)) = \frac{1}{3}\big(u(\boldsymbol{x}_i) + 3u(\boldsymbol{y}_i) - u(\boldsymbol{z}_i)\big). \tag{3.2.30}$$

Let $\boldsymbol{I}_G$, $\boldsymbol{E}_Y^p$ and $\boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})}$ be defined as in Section 3.2.1, and define $\boldsymbol{E}_Z^p$ as the degree p interpolation matrix which interpolates the value of $\boldsymbol{z}_i$ from values of grid points: $\boldsymbol{u}_z = \boldsymbol{E}_Z^p \boldsymbol{u}$. Then the discretization of (3.2.30) becomes

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})}, \text{ where } \boldsymbol{B} = \frac{1}{3}\big(\boldsymbol{I}_G + 3\boldsymbol{E}_Y^p - \boldsymbol{E}_Z^p\big). \tag{3.2.31}$$

One can then substitute (3.2.31) into (3.2.8) to obtain a new complete scheme for (3.2.2).

Similarly, one can construct a third order approximation for Neumann boundary conditions. Let us take a second look at Figure 3.4. To approximate the normal derivative at $\mathrm{cp}(\boldsymbol{x}_i)$, we build a cubic polynomial $p(x)$ along the normal direction interpolating at points $\{\boldsymbol{x}_i, \mathrm{cp}(\boldsymbol{x}_i), \boldsymbol{y}_i, \boldsymbol{z}_i\}$, and compute the first derivative of the polynomial at $\mathrm{cp}(\boldsymbol{x}_i)$,

$$\frac{\partial u}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i)) \approx p'(\mathrm{cp}(\boldsymbol{x}_i)) = \frac{2u(\boldsymbol{x}_i) + 3u(\mathrm{cp}(\boldsymbol{x}_i)) - 6u(\boldsymbol{y}_i) + u(\boldsymbol{z}_i)}{6\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2}. \tag{3.2.32}$$

Let $\boldsymbol{I}_G$, $\boldsymbol{E}_Y^q$, and $\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}$ be defined as in Section 3.2.2. Let $\boldsymbol{E}_Z^q$ be the degree q interpolation matrix for the values of $\boldsymbol{z}_i$'s: $\boldsymbol{u}_z = \boldsymbol{E}_Z^p \boldsymbol{u}$. Let $\boldsymbol{E}_{\mathrm{cp}(\boldsymbol{G})}^q$ be the degree q

interpolation matrix for the closest points of the ghost points: $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{G})} = \boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{u}$. Then the discretization of (3.2.32) is

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}, \text{ where } \boldsymbol{B} = \frac{1}{6}\big(2\boldsymbol{I}_G + 3\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})} - 6\boldsymbol{E}^q_Y + \boldsymbol{E}^q_Z\big). \tag{3.2.33}$$

Finally, we add up the approximations for $u(\mathrm{cp}(\boldsymbol{x}_i))$ and $\varphi(\mathrm{cp}(\boldsymbol{x}_i))\frac{\partial u}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i))$ to obtain an approximation for the Robin boundary conditions. With the same notations as before, the matrix form of the new high order approximation for Robin boundary condition is

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}, \text{ where } \boldsymbol{B} = \frac{1}{3}\boldsymbol{D}\big(\boldsymbol{I}_G + 3\boldsymbol{E}^p_Y - \boldsymbol{E}^p_Z\big) + \frac{1}{6}\boldsymbol{\Phi}(2\boldsymbol{I}_G + 3\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})} - 6\boldsymbol{E}^q_Y + \boldsymbol{E}^q_Z). \tag{3.2.34}$$

In practice, we usually use the same degree of interpolation for the approximation of $u(\mathrm{cp}(\boldsymbol{x}_i))$ and $\frac{\partial u}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i))$, i.e., $p = q$ in (3.2.34). In this case, our discretization becomes

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d}, \text{ where } \boldsymbol{B} = \frac{1}{3}\left(\boldsymbol{D} + \boldsymbol{\Phi}\right)\boldsymbol{I}_G + \frac{1}{2}\boldsymbol{\Phi}\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})} + \left(\boldsymbol{D} - \boldsymbol{\Phi}\right)\boldsymbol{E}^q_Y - \left(\frac{1}{3}\boldsymbol{D} - \frac{1}{6}\boldsymbol{\Phi}\right)\boldsymbol{E}^q_Z. \tag{3.2.35}$$

3.2.5 Numerical examples for Poisson problems on two dimensional domains

In this section, we consider several numerical examples on different domains in $\mathbb{R}^2$. Problems in three dimensions will be solved with a multigrid method in the next chapter.

3.2.5.1 Poisson problems on a disk

First of all, we consider Poisson's equation with Dirichlet boundary conditions (3.2.2) on a disk centered at origin with radius $\sqrt{2}$. The exact solution $u(\theta, r)$ is chosen to be $r^5\sin(2\theta)$ in polar coordinates. The right hand side function $f = -\Delta u = -21r^3\sin(2\theta)$ is then computed using $\Delta u = \frac{1}{r}\frac{d}{dr}\left(r\frac{du}{dr}\right) + \frac{1}{r^2}\frac{d^2u}{d\theta^2}$. Dirichlet boundary conditions are specified according to the value of the exact solution. We apply discretization (3.2.7) with $p = 1$ (linear extrapolation augmented by linear interpolation) to handle the boundary conditions. Table 3.1(a) shows 2nd order convergence results for the corresponding scheme (3.2.8). We also tested discretization (3.2.31) with $p = 2$ (quadratic extrapolations augmented by quadratic interpolations) for boundary conditions. From Table 3.1(b), when using the higher order approximation to handle boundary conditions we still obtain 2nd-order convergence results. This is because

we are still using 2nd-order approximation to the Laplacian in the interior of the domain. Nevertheless, the errors from the scheme with a higher order approximation to boundary conditions are smaller than those from the lower order approximation to boundary conditions.

Table 3.1: Convergence study for Poisson's equation on a disk with Dirichlet boundary conditions.

(a) Linear extrapolation and linear interpolation

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.53e-1	3.69e-2	9.04e-3	2.24e-3	5.65e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.05	2.03	2.02	1.98

(b) Quadratic extrapolation and quadratic interpolation

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.76e-2	2.34e-3	3.00e-4	6.63e-5	1.71e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.91	2.96	2.18	1.95

We then solve the shifted Poisson's equation with Neumann boundary conditions (3.2.11) on the same disk. The same exact solution $u = r^5 \sin(2\theta)$ is chosen. The shift is chosen to be $c = 1$, and the right hand side function, $f = -\Delta u + u = -20r^3 \sin(2\theta)$. On the boundary $\frac{\partial u}{\partial \boldsymbol{n}} = 20 \sin(2\theta)$. We discretize the Neumann boundary conditions by (3.2.14) with $q = 2$ and (3.2.33) with $q = 3$, and the corresponding convergence numerical results are shown in Tables 3.2(a) and 3.2(b) respectively. From Table 3.2 we can see that the lower order discretization (3.2.14) gives roughly 2nd order convergence results, while the higher order discretization (3.2.33) gives clear 2nd order convergence results with smaller numerical errors.

We finally consider Poisson's equation with Robin boundary conditions (3.2.19) on the same disk. Again the exact solution is chosen to be $u(\theta, r) = r^5 \sin(2\theta)$, and $f(\theta, r) = -21r^3 \sin(2\theta)$. On the boundary $u + \varphi \frac{\partial u}{\partial \boldsymbol{n}} = h$, where $\varphi(\theta, r) = 2 + \cos(\theta)$, and $h(\theta, r) = 4\sqrt{2} \sin(2\theta) + 20(2 + \cos(\theta)) \sin(2\theta)$. We test the lower order approximation (3.2.25) with $q = 3$ and the higher order approximation (3.2.35) with $q = 3$ to handle Robin boundary conditions. Table 3.3(a) shows roughly 2nd-order convergence results for the lower order approximation (3.2.25). Table 3.3(b) shows clear 2nd-order convergence for the higher order discretization (3.2.35) with smaller numerical errors than those in Table 3.3(a).

Table 3.2: Convergence study for the shifted Poisson's equation on a disk with Neumann boundary conditions.

(a) 2nd-order approximation for the normal derivative augmented by quadratic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	4.89e-2	1.57e-2	4.42e-3	1.16e-3	3.07e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.64	1.83	1.92	1.92

(b) 3rd-order approximation for the normal derivative augmented by cubic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.18e-2	3.15e-3	7.97e-4	2.00e-4	4.96e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.90	1.98	2.00	2.01

Table 3.3: Convergence study for Poisson's equation on a disk with Robin boundary conditions.

(a) Lower-order discretization (3.2.25) augmented by cubic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	3.48e-2	1.27e-2	3.87e-3	1.04e-3	2.24e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.45	1.72	1.89	2.22

(b) Higher-order discretization (3.2.35) augmented by cubic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.36e-2	3.61e-3	8.65e-4	2.08e-4	5.03e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.90	2.06	2.06	2.05

3.2.5.2 Poisson problems on a bean-shaped domain

In this section, we investigate Poisson problems on a domain surrounded by a bean-shaped curve. The bean-shaped curve is a parameterized spline curve defined in Section 2.5.2 (see Figure 2.4). To solve Poisson problems on the bean-shaped domain, we first need to clarify which grid points are exterior to the domain. A general approach would be finding the closest point representation of the domain; another method is to find the closest points of the boundary of the domain and the signed distance functions. Here we make use of the parametrization of the boundary of the domain. After obtaining a parametrization $(x(t), y(t))$ of the bean curve as described in Section 2.5.2, we compute the unit tangent vector field $(t_x, t_y) = \left(\frac{\dot{x}(t)}{\sqrt{\dot{x}^2(t)+\dot{y}^2(t)}}, \frac{\dot{y}(t)}{\sqrt{\dot{x}^2(t)+\dot{y}^2(t)}}\right)$. The outward unit normal vector field is then either $(t_y, -t_x)$ or $(-t_y, t_x)$. In other words, we have the analytical expressions of the outward unit normal for each point on the bean curve. For each grid point $\boldsymbol{x}_i$, we find its closest point $\mathrm{cp}(\boldsymbol{x}_i)$ and compute the vector $\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)$; if this vector has the same direction as the outward normal, then $\boldsymbol{x}_i$ is an exterior grid point, otherwise $\boldsymbol{x}_i$ is an interior grid point.

We choose the exact solution to be $u(x, y) = x^6y^6 + x^6 + y^6 + \sin(2\pi x) + \sin(2\pi y) + \cos(2\pi x) + \cos(2\pi y)$. We then compute the right hand side function and set up different kinds of boundary conditions according to this exact solution. Table 3.4 shows the convergence results for solving Poisson's equation (3.2.2) with Dirichlet boundary conditions. The Dirichlet boundary conditions are set up to match the exact solution evaluated on the boundary. Table 3.5 shows the convergence results for solving the shifted Poisson's equation (3.2.11) with Neumann boundary conditions. The Neumann boundary conditions are set up to match the normal derivatives of the exact solution on the boundary. Table 3.6 shows the convergence results for solving Poisson's equation (3.2.19) with Robin boundary conditions. The Robin boundary conditions are set up by computing $h = u + \varphi\frac{\partial u}{\partial \boldsymbol{n}}$ on the boundary, where φ is chosen to be $2 + x + y$. All of the numerical results share similar trends with the corresponding problem on a disk as discussed before.

3.3 Elliptic equations on surfaces with boundaries

In the previous chapter, we investigated how to solve elliptic PDEs on curves and surfaces without boundaries. In Section 3.2, we discussed how to deal with boundary conditions for domains which could be regarded as codimension-0 manifolds embedded in $\mathbb{R}^n$. Now we are ready to combine the ideas of handling surface differential

Table 3.4: Convergence study for Poisson's equation on a bean-shaped domain with Dirichlet boundary conditions.

(a) Linear extrapolation and linear interpolation

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.89e-1	4.58e-2	1.13e-2	3.10e-3	8.89e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.05	2.01	1.87	1.80

(b) Quadratic extrapolation and quadratic interpolation

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	9.35e-2	1.81e-2	4.67e-3	1.19e-3	2.99e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.37	1.96	1.98	1.99

Table 3.5: Convergence study for the shifted Poisson's equation on a bean-shaped domain with Neumann boundary conditions.

(a) 2nd-order approximation for the normal derivative augmented by quadratic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	4.91e-1	1.13e-1	4.58e-2	1.21e-2	3.11e-3
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.11	1.31	1.92	1.96

(b) 3rd-order approximation for the normal derivative augmented by cubic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	2.72e-1	7.57e-2	1.58e-2	3.95e-3	9.38e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.84	2.26	2.00	2.07

Table 3.6: Convergence study for Poisson's equation on a bean-shaped domain with Robin boundary conditions.

(a) Lower-order discretization (3.2.25) augmented by cubic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	2.20e-1	4.50e-2	1.49e-2	3.77e-3	8.51e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.29	1.60	1.98	2.15

(b) Higher-order discretization (3.2.35) augmented by cubic interpolations

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	2.30e-1	5.54e-2	1.32e-2	3.28e-3	7.88e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.05	2.07	2.01	2.06

operators and boundary conditions to solve elliptic PDEs on curves and surfaces with boundaries.

Suppose we have a set of grid points surrounding the curve or surface with a boundary. If a grid point does not lies on the boundary but its closest point lie on the boundary, then we define this grid point to be a ghost point. For all the grid points that are not ghost points, we apply discretization (2.2.1) as discussed in the previous chapter. For all the ghost points, we shall modify the ideas described in Section 3.2 to handle boundary conditions. The details shall be described for Dirichlet, Neumann and Robin boundary conditions.

3.3.1 Dirichlet boundary conditions

In this section we consider the following surface Poisson's equation on $\mathcal{S}$ with boundary $\partial\mathcal{S}$,

$$-\Delta_{\mathcal{S}} u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{S}, \tag{3.3.1a}$$

$$u(\boldsymbol{x}) = g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\mathcal{S}. \tag{3.3.1b}$$

Figure 3.5 illustrates the method to extrapolate values of ghost points according to Dirichlet boundary conditions. The point $\boldsymbol{x}_i$ is a ghost point, $\mathrm{cp}(\boldsymbol{x}_i)$ is its closest point, $\boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i$, and $\boldsymbol{z}_i = 2\boldsymbol{y}_i - \mathrm{cp}(\boldsymbol{x}_i)$. Let $\tilde{u}$ be the extended solution which is constant along the normal directions to the curve. For the ghost point $\boldsymbol{x}_i$, we can use linear extrapolation to obtain $\tilde{u}(\boldsymbol{x}_i)$: $\tilde{u}(\boldsymbol{x}_i) = 2\tilde{u}(\mathrm{cp}(\boldsymbol{x}_i)) - \tilde{u}(\boldsymbol{y}_i)$. Since

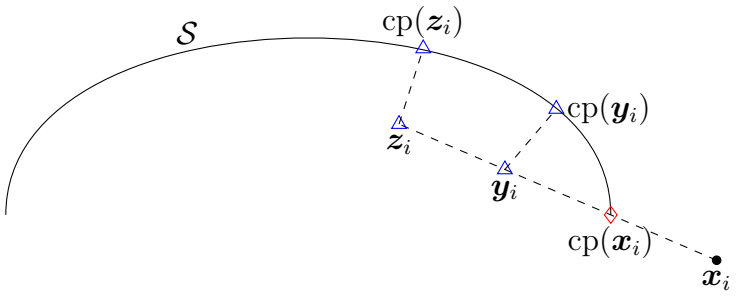


Figure 3.5: Handling Dirichlet boundary conditions for a curve with a boundary.

$\tilde{u}(\boldsymbol{y}_i) = \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i))$,

$$\tilde{u}(\boldsymbol{x}_i) = 2\tilde{u}(\mathrm{cp}(\boldsymbol{x}_i)) - \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i)). \tag{3.3.2}$$

Let $\boldsymbol{G}$ be the set of all ghost points, and $\mathrm{cp}(\boldsymbol{Y}) = \{\mathrm{cp}(\boldsymbol{y}_i) | \boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i,\ \boldsymbol{x}_i \in \boldsymbol{G}\}$. Define $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{Y})}$ be the column vector storing the values of $\tilde{u}$ at points in $\mathrm{cp}(\boldsymbol{Y})$. Define $\boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})}$ be the degree p interpolation matrix interpolating values of points in $\mathrm{cp}(\boldsymbol{Y})$ from grid points values: $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{Y})} = \boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})}\boldsymbol{u}$. Let $\boldsymbol{u}_I$, $\boldsymbol{u}_G$, $\boldsymbol{u}$, $\boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})}$ and $\boldsymbol{I}_G$ be defined as in Section 3.2.1. Then (3.3.2) can be discretized as $\boldsymbol{u}_G = 2\boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})} - \boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})}\boldsymbol{u}$, or

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})},\ \text{where } \boldsymbol{B} = \frac{1}{2}(\boldsymbol{I}_G + \boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})}). \tag{3.3.3}$$

Note that (3.3.3) is similar to the discretization (3.2.7) for the codimension-0 case before. The difference is that in (3.3.3) we use $\boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})}$ while in (3.2.7) we used $\boldsymbol{E}^p_Y$.

Similarly to Section 3.2.4, we can also use a higher order extrapolation to obtain values of ghost points as shown in Figure 3.5,

$$\tilde{u}(\boldsymbol{x}_i) = 3\tilde{u}(\mathrm{cp}(\boldsymbol{x}_i)) - 3\tilde{u}(\mathrm{cp}(\boldsymbol{x}_i)) + \tilde{u}(\mathrm{cp}(\boldsymbol{x}_i)). \tag{3.3.4}$$

Let $\mathrm{cp}(\boldsymbol{Z}) = \{\mathrm{cp}(\boldsymbol{z}_i) | \boldsymbol{z}_i = 2\mathrm{cp}(\boldsymbol{y}_i) - \mathrm{cp}(\boldsymbol{x}_i),\ \boldsymbol{x}_i \in \boldsymbol{G}\}$. Define $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{Z})}$ be the column vector storing the values of $\tilde{u}$ at points in $\mathrm{cp}(\boldsymbol{Z})$. Define $\boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Z})}$ be the degree p interpolation matrix interpolating values of points in $\mathrm{cp}(\boldsymbol{Z})$ from grid points values: $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{Z})} = \boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Z})}\boldsymbol{u}$. Then (3.3.4) can be discretized as $\boldsymbol{u}_G = 3\boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})} - 3\boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})}\boldsymbol{u} + \boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})}\boldsymbol{u}$, or

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})},\ \text{where } \boldsymbol{B} = \frac{1}{3}\big(\boldsymbol{I}_G + 3\boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Y})} - \boldsymbol{E}^p_{\mathrm{cp}(\boldsymbol{Z})}\big). \tag{3.3.5}$$

As mentioned before, for grid points that are not ghost points, we can use the discretization (2.2.1) with $c = 0$, which has matrix form (2.2.2). Let $\boldsymbol{M} = \boldsymbol{E}^{p_1}\boldsymbol{L} - \gamma(\boldsymbol{I} - \boldsymbol{E}^{p_2})$, where $\boldsymbol{E}^{p_1}$ and $\boldsymbol{E}^{p_2}$ are interpolation matrices for the closest points of grid

points with degrees p_1 and p_2 respectively, $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{L}$ is the Laplacian matrix, and γ is a numerical parameter determining the amount of penalization. We partition $\boldsymbol{M}$ into $\boldsymbol{M} = \begin{pmatrix} \boldsymbol{M}_I \\ \boldsymbol{M}_G \end{pmatrix}$, where $\boldsymbol{M}_I$ is made of the first n_i rows of $\boldsymbol{M}$ corresponding to the non-exterior points, and $\boldsymbol{M}_G$ is the rest n_g rows which shall be abandoned. Then the matrix form of the discretization of (3.3.1) becomes

$$\begin{pmatrix} -\boldsymbol{M}_I \\ \boldsymbol{B} \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} \boldsymbol{f}_I \\ \boldsymbol{g}_{\mathrm{cp}(\boldsymbol{G})} \end{pmatrix}, \tag{3.3.6}$$

where $\boldsymbol{B}$ can be chosen to be either (3.3.3) or (3.3.5).

3.3.2 Neumann boundary conditions

In this section, we consider the shifted surface Poisson's equation on $\mathcal{S}$ with boundary $\partial\mathcal{S}$,

$$-\Delta_{\mathcal{S}} u(\boldsymbol{x}) + cu(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{S}, \tag{3.3.7a}$$

$$\frac{\partial u}{\partial \boldsymbol{\mu}} = h(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\mathcal{S}. \tag{3.3.7b}$$

Here $\boldsymbol{\mu}$ denotes the co-normal vector which is normal to $\partial\mathcal{S}$ but tangent to $\mathcal{S}$. If $\mathcal{S}$ is a curve with a boundary, then $\boldsymbol{\mu}$ is just the tangent vector at the end point of $\mathcal{S}$.

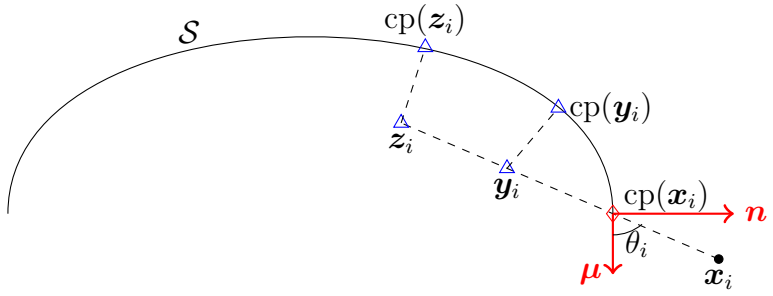


Figure 3.6: Handling Neumann boundary conditions for a curve with a boundary.

Figure 3.6 illustrates how to handle Neumann boundary conditions for a curve with a boundary. Let $\boldsymbol{d}_i = \boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)$, $\boldsymbol{n}$ be the unit normal vector at $\mathrm{cp}(\boldsymbol{x}_i)$, and $\boldsymbol{\mu}$ be the unit co-normal vector at $\mathrm{cp}(\boldsymbol{x}_i)$. Let θ_i denote the angle between the co-normal vector $\boldsymbol{\mu}$ and the vector $\boldsymbol{d}_i$. Then we have

$$\frac{\partial \tilde{u}}{\partial \boldsymbol{d}_i}(\mathrm{cp}(\boldsymbol{x}_i)) = \frac{\partial \tilde{u}}{\partial \boldsymbol{\mu}}(\mathrm{cp}(\boldsymbol{x}_i)) \cos(\theta_i) + \frac{\partial \tilde{u}}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i)) \sin(\theta_i).$$

Remembering that $\tilde{u}$ is constant along the normals to $\mathcal{S}$, we have $\frac{\partial \tilde{u}}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i)) = 0$, and thus

$$\frac{\partial \tilde{u}}{\partial \boldsymbol{d}_i}(\mathrm{cp}(\boldsymbol{x}_i)) = \frac{\partial \tilde{u}}{\partial \boldsymbol{\mu}}(\mathrm{cp}(\boldsymbol{x}_i))\cos(\theta_i). \tag{3.3.8}$$

If we approximate $\frac{\partial \tilde{u}}{\partial \boldsymbol{d}_i}(\mathrm{cp}(\boldsymbol{x}_i))$ by centered finite difference scheme, then we obtain

$$\frac{\partial \tilde{u}}{\partial \boldsymbol{\mu}}(\mathrm{cp}(\boldsymbol{x}_i))\cos(\theta_i) = \frac{\tilde{u}(\boldsymbol{x}_i) - \tilde{u}(\boldsymbol{y}_i)}{\|\boldsymbol{x}_i - \boldsymbol{y}_i\|_2}. \tag{3.3.9}$$

Let $d_i = \|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2$. Noting that $\tilde{u}(\boldsymbol{y}_i) = \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i))$ and $\boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i$, (3.3.9) becomes

$$\frac{\partial \tilde{u}}{\partial \boldsymbol{\mu}}(\mathrm{cp}(\boldsymbol{x}_i))\cos(\theta_i) = \frac{\tilde{u}(\boldsymbol{x}_i) - \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i))}{2d_i}. \tag{3.3.10}$$

Let $\tilde{\boldsymbol{d}}$ be the column vector storing the product of d_i and $\cos(\theta_i)$: $\tilde{\boldsymbol{d}} = \big(d_i\cos(\theta_i),\ i = 1,\ldots,n_g\big)$. In practice, we obtain $d_i\cos(\theta_i)$ by computing the dot product of $\boldsymbol{d}_i$ and $\boldsymbol{\mu}$ where $\boldsymbol{d}_i = \boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)$ and the co-normal vector $\boldsymbol{\mu}$ is known from the geometry of the surface.

Let $\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}$ be the column vector storing the function h evaluated at the closest points of the ghost points. Define $\boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\tilde{\boldsymbol{d}} = \big(h(\mathrm{cp}(\boldsymbol{x}_i))\tilde{d}_i,\ i = 1,\ldots,n_g\big)$. Define $\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Y})}$ to be the degree q interpolation matrix so that $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{Y})} = \boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Y})}\boldsymbol{u}$. Then the discretization for (3.3.7b) becomes

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\tilde{\boldsymbol{d}},\ \text{where } \boldsymbol{B} = \frac{1}{2}\left(\boldsymbol{I}_G - \boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Y})}\right). \tag{3.3.11}$$

Similarly to (3.2.33), we can also construct a higher order scheme to handle Neumann boundary conditions. Let $\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})}$ be the degree q interpolation matrix for the closest points of ghost points: $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{G})} = \boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{u}$. Let $\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Z})}$ be the interpolation matrix which interpolates the values of $\mathrm{cp}(\boldsymbol{z}_i)$'s from values of grid points: $\boldsymbol{u}_{\mathrm{cp}(\boldsymbol{Z})} = \boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Z})}\boldsymbol{u}$. Then a higher order discretization for (3.3.7b) is

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\boldsymbol{d},\ \text{where } \boldsymbol{B} = \frac{1}{6}\big(2\boldsymbol{I}_G + 3\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})} - 6\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Y})} + \boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Z})}\big). \tag{3.3.12}$$

Let $\boldsymbol{A} = -\boldsymbol{M} + c\boldsymbol{I}$ where $\boldsymbol{M} = \boldsymbol{E}^{p_1}\boldsymbol{L} - \gamma(\boldsymbol{I} - \boldsymbol{E}^{p_2})$, and partition $\boldsymbol{A}$ into $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_I \\ \boldsymbol{A}_G \end{pmatrix}$. Then the matrix form of discretization for (3.3.7) becomes

$$\begin{pmatrix} \boldsymbol{A}_I \\ \boldsymbol{B} \end{pmatrix}\boldsymbol{u} = \begin{pmatrix} \boldsymbol{f}_I \\ \boldsymbol{h}_{cp}\tilde{\boldsymbol{d}} \end{pmatrix}, \tag{3.3.13}$$

where $\boldsymbol{B}$ can be chosen to be either (3.3.11) or (3.3.12).

Remark 3.3.1 *The Neumann boundary condition specifies the derivative of u along the co-normal vector $\boldsymbol{\mu}$, but we are only able to approximate the derivatives along the vector $\boldsymbol{d}_i$ via finite differences. We overcome this issue by linking the derivatives along different directions via a factor of* $\cos(\theta_i)$ *to obtain (3.3.10). Our method is in contrast to the approach in [59] where the authors focused on solving surface elliptic eigenvalue problems with homogeneous boundary conditions and ignored the* $\cos(\theta_i)$ *term. This is because* $\frac{\partial u}{\partial \boldsymbol{\mu}} = 0$ *on the boundary for homogeneous Neumann boundary conditions and the left hand side of (3.3.10) simply vanishes regardless of what* $\cos(\theta_i)$ *is.*

Remark 3.3.2 *Note that our technique for non-homogeneous Neumann boundary conditions (and Robin boundary conditions as we shall see next) requires additional geometric information compared to the Dirichlet case. We need to compute the inner product between the co-normal vector $\boldsymbol{\mu}$ and the vector $\boldsymbol{d}_i = \boldsymbol{x}_i - cp(\boldsymbol{x}_i)$. In our experiments, we shall use explicitly an analytical representation of the co-normal. Further research could investigate approximating this inner product purely from a closest point representation of the surface (and perhaps an additional closest point representation of the boundary itself).*

3.3.3 Robin boundary conditions

We finally investigate Poisson's equation on $\mathcal{S}$ with Robin boundary conditions,

$$-\Delta_{\mathcal{S}} u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{S}, \tag{3.3.14a}$$

$$u(\boldsymbol{x}) + \varphi(\boldsymbol{x}) \frac{\partial u}{\partial \boldsymbol{\mu}} = h(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\mathcal{S}. \tag{3.3.14b}$$

For each ghost point $\boldsymbol{x}_i$, we approximate (3.3.14b) at the corresponding closest point $\mathrm{cp}(\boldsymbol{x}_i)$. Multiplying both sides of (3.3.14b) by $\cos(\theta_i)$, and using (3.3.8) we obtain that

$$\tilde{u}(\mathrm{cp}(\boldsymbol{x}_i)) \cos(\theta_i) + \varphi(\mathrm{cp}(\boldsymbol{x}_i)) \frac{\partial \tilde{u}}{\partial \boldsymbol{d}_i}(\mathrm{cp}(\boldsymbol{x}_i)) = h(\mathrm{cp}(\boldsymbol{x}_i)) \cos(\theta_i). \tag{3.3.15}$$

Substituting the approximations $\tilde{u}(\mathrm{cp}(\boldsymbol{x}_i)) = \frac{1}{2}\big(\tilde{u}(\boldsymbol{x}_i) + \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i))\big)$ and $\frac{\partial \tilde{u}}{\partial \boldsymbol{d}_i}(\mathrm{cp}(\boldsymbol{x}_i)) = \frac{\tilde{u}(\boldsymbol{x}_i) - \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i))}{2d_i}$ into (3.3.15), and multiplying both sides by d_i, we obtain that

$$\frac{1}{2}\big(\tilde{u}(\boldsymbol{x}_i) + \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i))\big) d_i \cos(\theta_i) + \frac{1}{2}\varphi(\mathrm{cp}(\boldsymbol{x}_i))\big(\tilde{u}(\boldsymbol{x}_i) - \tilde{u}(\mathrm{cp}(\boldsymbol{y}_i))\big) = h(\mathrm{cp}(\boldsymbol{x}_i)) d_i \cos(\theta_i). \tag{3.3.16}$$

Define $n_g \times n_g$ diagonal matrix

$$\tilde{\boldsymbol{D}} = \begin{pmatrix} d_1 \cos(\theta_1) & & \\ & \ddots & \\ & & d_n \cos(\theta_{n_g}) \end{pmatrix}_{n_g \times n_g}.$$

Let $\boldsymbol{\Phi}$ be defined as in (3.2.24). Then the matrix form of the discretization of (3.3.14b) becomes

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\tilde{\boldsymbol{d}}, \text{ where } \boldsymbol{B} = \frac{1}{2}[(\tilde{\boldsymbol{D}} + \boldsymbol{\Phi})\boldsymbol{I}_G + (\tilde{\boldsymbol{D}} - \boldsymbol{\Phi})\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Y})}]. \tag{3.3.17}$$

Similarly to (3.2.35), we can construct a higher order approximation for (3.3.14b),

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{h}_{cp}\tilde{\boldsymbol{d}}, \text{ where } \boldsymbol{B} = \frac{1}{3}\left(\tilde{\boldsymbol{D}} + \boldsymbol{\Phi}\right)\boldsymbol{I}_G + \frac{1}{2}\boldsymbol{\Phi}\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{G})} + \left(\tilde{\boldsymbol{D}} - \boldsymbol{\Phi}\right)\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Y})} - \left(\frac{1}{3}\tilde{\boldsymbol{D}} - \frac{1}{6}\boldsymbol{\Phi}\right)\boldsymbol{E}^q_{\mathrm{cp}(\boldsymbol{Z})}. \tag{3.3.18}$$

Let $\boldsymbol{M} = \boldsymbol{E}^{p_1}\boldsymbol{L} - \gamma(\boldsymbol{I} - \boldsymbol{E}^{p_2})$, and partition $\boldsymbol{M}$ into $\boldsymbol{M} = \begin{pmatrix} \boldsymbol{M}_I \\ \boldsymbol{M}_G \end{pmatrix}$. Then the matrix form of the discretization for (3.3.14) becomes

$$\begin{pmatrix} -\boldsymbol{M}_I \\ \boldsymbol{B} \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} \boldsymbol{f}_I \\ \boldsymbol{h}_{\mathrm{cp}(\boldsymbol{G})}\tilde{\boldsymbol{d}} \end{pmatrix}, \tag{3.3.19}$$

where $\boldsymbol{B}$ can be chosen to be either (3.3.17) or (3.3.18).

3.3.4 Numerical examples for Poisson problems on curves and surfaces with boundaries

In this section, we numerically solve surface Poisson problems with Dirichlet, Neumann and Robin boundary conditions on a semicircle, a sine-shaped curve and a hemisphere. We use lower order approximations (3.3.3), (3.3.11) and (3.3.17) to handle boundary conditions. For convenience we choose the augmenting interpolation degree to be 3 for all test cases, i.e., we choose $p = 3$ in (3.3.3) and $q = 3$ in (3.3.11) and (3.3.17).

3.3.4.1 Semicircle

Table 3.7 shows the results of the convergence study on a semicircle centered at the origin with radius $\sqrt{3}$. The exact solution is $\cos(\theta) + \sin(10\theta)$. The right hand side function and boundary conditions are set up according to the exact solution and the geometry of the semicircle for each case.

Table 3.7: Convergence study for Poisson problems on a semicircle centered at the origin with radius $\sqrt{3}$. The exact solution is $\cos(\theta) + \sin(10\theta)$.

(a) Poisson's equation (3.3.1) with Dirichlet boundary conditions. The boundary conditions are dealt by linear extrapolation (3.3.3) and augmented by cubic interpolation.

Δx	0.05	0.025	0.0125	0.00625	0.003125
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	5.19e-3	1.24e-3	3.05e-4	7.41e-5	1.83e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.07	2.02	2.04	2.02

(b) Shifted Poisson's equation (3.3.7) with Neumann boundary conditions. The boundary conditions are dealt by 2nd-order approximation (3.3.11) and augmented by cubic interpolation.

Δx	0.05	0.025	0.0125	0.00625	0.003125
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	2.09e-2	5.03e-3	1.27e-3	3.18e-4	7.88e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.06	1.99	1.99	2.01

(c) Poisson's equation (3.3.14) with Robin boundary conditions and $\varphi(\theta) = 2 + \cos(\theta)$. The boundary conditions are dealt by (3.3.17) and augmented by cubic interpolation.

Δx	0.05	0.025	0.0125	0.00625	0.003125
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	3.69e-2	8.79e-3	2.21e-3	5.52e-4	1.36e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.07	1.99	2.00	2.02

3.3.4.2 Sine-shaped curve

We solve surface Poisson's equations with various boundary conditions on a sine-shaped curve parameterized by $(t, \sin(t))$ where $t \in [0, \frac{3}{2}\pi]$. Figure 3.7 shows the banded computational grids with different mesh sizes. Table 3.8 shows the results of the convergence study. The exact solution is $\cos(t) + \sin(10t)$. The right hand side function and the boundary conditions are set up according to the exact solution and the geometry of the sine-shaped curve for each case.

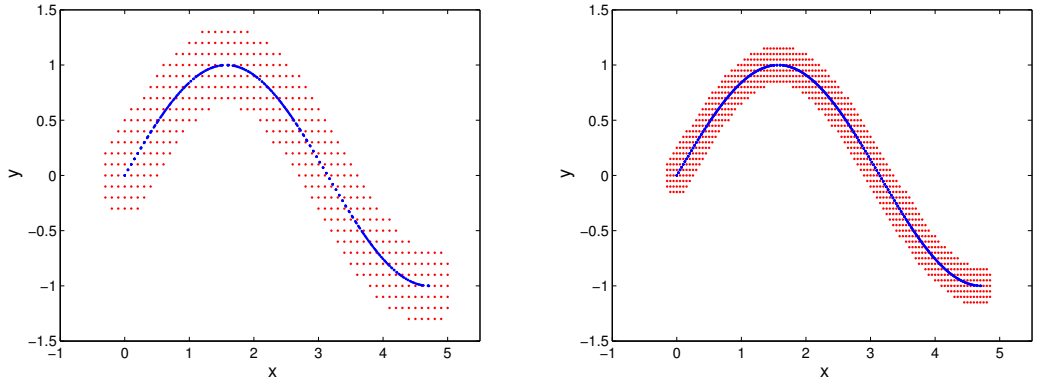


Figure 3.7: Computational grids for the Closest Point Method on a sine-shaped curve with $\Delta x = 0.1$ (left) and $\Delta x = 0.05$ (right).

3.3.4.3 Hemisphere

Table 3.9 shows the results of the convergence study on a hemisphere centered at the origin with radius $\sqrt{2}$. The exact solution is $\cos(3\theta)\sin(\phi)^3(9\cos(\phi)^2 - 1)$. The right hand side function and boundary conditions are then set up according to the exact solution and the geometry of the hemisphere for each case.

3.4 Conclusion and discussion

In summary, we adopted the ghost point approach in a systematic manner to handle Dirichlet, Neumamnn and Robin boundary conditions for Poisson problems on domains and surfaces with boundaries. We think this method could be extended to handle general (nonlinear) boundary conditions with the form $G\big(\boldsymbol{x}, u(\boldsymbol{x}), \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x})\big) = 0$ where $\boldsymbol{x}$ is on the boundary. In particular, for the grid points which are not ghost points we discretize the elliptic equation using standard finite differences. For each

Table 3.8: Convergence study for Poisson problems on a sine-shaped curve parameterized by $(t, \sin(t))$ where $t \in [0, \frac{3}{2}\pi]$. The exact solution is $\cos(t) + \sin(10t)$.

(a) Poisson's equation (3.3.1) with Dirichlet boundary conditions. The boundary conditions are dealt by linear extrapolation (3.3.3) augmented by cubic interpolation.

Δx	0.0125	0.00625	0.003125	0.0015625	0.00078125
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	7.82e-4	1.94e-4	5.17e-5	1.20e-5	3.42e-6
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.01	1.91	2.11	1.80

(b) Shifted Poisson's equation (3.3.7) with Neumann boundary conditions. The boundary conditions are dealt by 2nd-order approximation (3.3.11) augmented by cubic interpolation.

Δx	0.0125	0.00625	0.003125	0.0015625	0.00078125
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	3.25e-2	7.92e-3	1.88e-3	4.21e-4	8.49e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.04	2.08	2.16	2.31

(c) Poisson's equation (3.3.14) with Robin boundary conditions and $\varphi(t) = 2 + \sin(t)$. The boundary conditions are dealt by (3.3.17) augmented by cubic interpolation.

Δx	0.0125	0.00625	0.003125	0.0015625	0.00078125
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	2.82e-2	6.95e-3	1.67e-3	3.75e-4	7.64e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.02	2.06	2.15	2.29

Table 3.9: Convergence study for Poisson problems on a hemisphere centered at the origin with radius $\sqrt{2}$.

(a) Poisson's equation (3.3.1) with Dirichlet boundary conditions. The boundary conditions are dealt by linear extrapolation (3.3.3) augmented by cubic interpolation.

Δx	0.4	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	5.21e-1	1.58e-1	4.14e-2	1.05e-2
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.73	1.93	1.98

(b) Shifted Poisson's equation (3.3.7) with Neumann boundary conditions. The boundary conditions are dealt by 2nd-order approximation (3.3.11) augmented by cubic interpolation.

Δx	0.4	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.91e-1	4.12e-2	1.09e-2	2.71e-3
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.21	1.92	2.00

(c) Poisson's equation (3.3.14) with Robin boundary conditions and $\varphi(\theta, \phi) = 2 + \cos(\theta)$. The boundary conditions are dealt by (3.3.17) augmented by cubic interpolation.

Δx	0.4	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	2.5e-1	3.40e-2	1.84e-2	4.51e-3
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.76	2.01	2.03

ghost point $\boldsymbol{x}_i$ we can discretize $G\big(\boldsymbol{x}, u(\boldsymbol{x}), \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x})\big) = 0$ at $\mathrm{cp}(\boldsymbol{x}_i)$ using the methods described in this chapter. Combining these two parts of the discretized equation together, we obtain a system of (nonlinear) equations. One could then use existing solvers for nonlinear equations such as Newton's methods [46]. The particular structure of the system could also potentially be exploited for increased efficiency.

Chapter 4

Multigrid Solvers

In the previous two chapters we mainly focused on numerical examples in two dimensions, where the resulting linear systems are solved by sparse direct solvers (e.g., Matlab's backslash). However, for problems where the embedding space is $\mathbb{R}^3$, fast and robust iterative linear system solvers are desired. We decide to adopt the geometric multigrid method (see e.g., [13, 39]) because it is quite effective in solving linear systems arising from discretization of elliptic operators. Some literature on multigrid method for surface PDEs [48, 49, 11] is based on the finite element method. We shall construct multigrid solvers in the setting of the Closest Point Method.

In Section 4.1, we develop a V-Cycle multigrid scheme to solve linear systems arising from elliptic equations on curves and surfaces without boundaries[1]. In Section 4.2, we modify the relaxation scheme in the algorithm of Section 4.1 to make it work on codimension-0 domains. Finally in Section 4.3, we further modify the relaxation method so that the multigrid algorithm can be adapted to elliptic PDEs on curves and surfaces with boundaries. Numerical tests will be conducted in each section to illustrate the efficiency of our multigrid algorithm. We shall see that for all our test cases, the convergence speed almost stays the same as we increase the number of grid levels, just as it does for the most basic multigrid solvers applied to elliptic equations on regular square domains.

[1]The W-Cycle scheme is also briefly mentioned and tested, and it has slightly faster convergence speed than the V-Cycle scheme. However, we shall focus on the V-Cycle scheme because it yields similar results to the W-Cycle scheme with less computational cost.

4.1 Multigrid solvers for surfaces without boundaries

Based on the closest point representation of the surface, we shall construct the relaxation scheme on each grid level and the restriction/prolongation operators transferring data between coarse and fine grids. A closest point method V-Cycle multigrid algorithm will be constructed and tested, and a W-Cycle scheme will also be briefly mentioned and tested. Most contents of this section are based on [16] written by the current author.

4.1.1 The Multigrid "smoother": weighted Jacobi iteration

One of the key ingredients of a multigrid algorithm is the "smoother" [13, 39], which damps out high-frequency errors and thus smooths the solution at each grid level. One good choice of "smoother" is the weighted Jacobi iteration: suppose we want to solve $\boldsymbol{A}^h\boldsymbol{u}^h = \boldsymbol{f}^h$, starting from some initial $\boldsymbol{u}^h$, we update the iterates by

$$\boldsymbol{u}^h := \boldsymbol{u}^h + \omega \text{diag}(\boldsymbol{A}^h)^{-1}(\boldsymbol{f}^h - \boldsymbol{A}^h\boldsymbol{u}^h), \tag{4.1.1}$$

where $0 < \omega \leq 1$ is the weight parameter. One possible approach is to simply use this standard weighted Jacobi iteration on the discretization (2.2.3) of our embedding equation (2.1.5). An alternative strategy, and the one we propose here, is based instead on considering the earlier constrained system (2.1.4): we want to ensure that the solution satisfies the constraint (2.1.4b) so we re-extend the solution following each weighted Jacobi iteration.

4.1.2 Ruuth–Merriman smoothing strategy

The idea of this new smoothing strategy stems from the original Ruuth–Merriman approach for time-dependent surface PDEs [79]. In the continuous setting, if the solution $\tilde{u}$ is constant along the direction normal to the surface, then the Cartesian Laplacian of $\tilde{u}$ agrees with the Laplace–Beltrami function of $\tilde{u}$ on the surface. So, roughly speaking, $\boldsymbol{L}^h\boldsymbol{u}^h$ is consistent with $\Delta_S\tilde{u}$ *on the surface*, provided that $\boldsymbol{u}^h$ is constant along the normals to the surface. Here $\boldsymbol{u}^h$ is the discrete solution and $\boldsymbol{L}^h$ is the usual Cartesian Laplacian matrix. Under this side condition, we can replace $\boldsymbol{M}^h$ with $\boldsymbol{L}^h$ in one iteration, where $\boldsymbol{M}^h = \boldsymbol{E}_p^h\boldsymbol{L}^h - \gamma(\boldsymbol{I}^h - \boldsymbol{E}_q^h)$ was defined in equation (2.2.3). Let $\tilde{\boldsymbol{A}}^h = c\boldsymbol{I}^h - \boldsymbol{L}^h$ be the shifted Cartesian Laplacian matrix. Starting from an initial $\boldsymbol{u}^h$ which is constant along the normal direction to the surface, in one

iteration we do the standard Jacobi iteration with respect to the $\tilde{\boldsymbol{A}}^h = c\boldsymbol{I}^h - \boldsymbol{L}^h$ (instead of $\boldsymbol{A}^h = c\boldsymbol{I}^h - \boldsymbol{M}^h$), and then immediately do a closest point extension to ensure the discrete solution is still constant along the normal direction to the surface:

$$\boldsymbol{u}^h := \boldsymbol{u}^h + \omega \text{diag}(\tilde{\boldsymbol{A}}^h)^{-1}(\boldsymbol{f}^h - \tilde{\boldsymbol{A}}^h \boldsymbol{u}^h), \tag{4.1.2a}$$

$$\boldsymbol{u}^h := \boldsymbol{E}^h \boldsymbol{u}^h. \tag{4.1.2b}$$

Typically one needs to pick a weight ω strictly less than one for smoothing because pure Jacobi iteration ($\omega = 1$) cannot damp out the high frequency errors (see e.g., [13]). As we shall see in Section 4.2, for codimension 0 domains we do need $\omega < 1$ to obtain desired convergence rate for the V-Cycle scheme. However, for curves and surfaces with codimension higher than 0, picking $\omega = 1$ (using the pure Jacobi iteration) seems to yield good convergence results in practice (see the numerical results in Section 4.1.5). One possible explanation is that a closest point extension step (4.1.2b) is done immediately after the relaxation step (4.1.2a) for the surface cases, and this extension step might have helped eliminate high frequency errors. In our multigrid algorithm we use pure Jacobi iteration ($\omega = 1$) for curves and surfaces with or without boundaries, and weighted Jacobi iteration ($0 < \omega < 1$) for codimension 0 domains.

We also remark that interpolation $\boldsymbol{E}^h$ with a sufficiently high degree should be used in order to avoid introducing interpolation errors which dominate over the errors from the discrete Laplacian $\boldsymbol{L}^h$. In particular, we need to use cubic interpolation to build $\boldsymbol{E}^h$ if we use a second-order difference scheme to build $\boldsymbol{L}^h$.

4.1.3 Restriction and prolongation operators

In a multigrid algorithm, one also needs to build *restriction* operators which restrict data from fine grid to coarse grid and *prolongation* operators which prolong data from coarse grid to fine grid.

For practical efficiency, we would like to perform the computation in narrow bands with bandwidths shrinking proportional to the grid sizes. This makes the coarse grid band wider than the fine grid band (see Figure 4.1). Some coarse grid points are out of the range of the fine grid band, which means there are no fine grid points surrounding those "outer" coarse grid points; in those cases the standard restriction strategies such as "injection" and "full weighting" [13] do not work. We overcome this problem with the help of the closest point extension. When restricting the values

of points on the fine grid to the coarse grid, we assign the value of each coarse grid point to be the value of its corresponding closest point, which in turn is obtained by interpolating the values of fine grid points surrounding the closest point. Naturally, we can construct the prolongation operators in the same way[2]: take the value of each fine grid point to be the value of its corresponding closest point, which is obtained through interpolation of values of coarse grid points. Similar to the closest point extension on a single grid, the restriction and prolongation operators have corresponding matrix forms; we denote the restriction matrix by $\boldsymbol{E}_h^{2h}$, and the prolongation matrix by $\boldsymbol{E}_{2h}^h$. Figure 4.1 illustrates the process of doing restriction and prolongation.

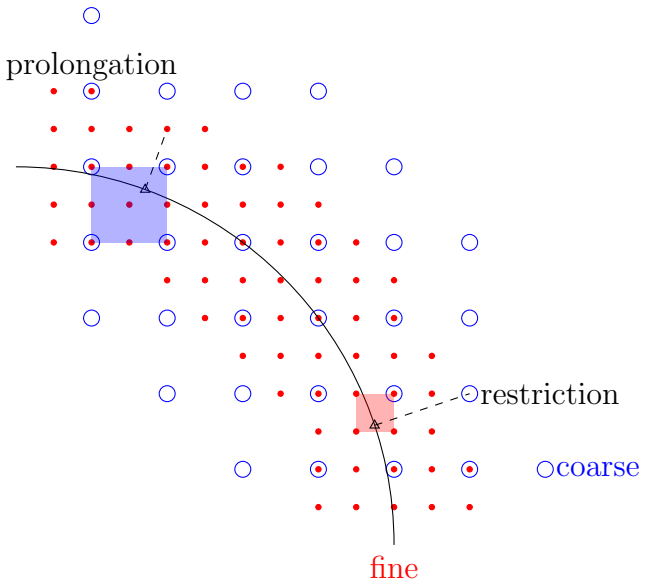


Figure 4.1: Illustration of Restriction and Prolongation.
Blue circles indicate the coarse grid points, red dots indicate the fine grid points.
Restriction: restrict quantities from fine grid to coarse grid using $\boldsymbol{E}_h^{2h}$.
Prolongation: prolong quantities from coarse grid to fine grid using $\boldsymbol{E}_{2h}^h$.

4.1.4 Closest Point Method V-Cycle algorithm

Armed with the smoothing strategy in Section 4.1.2 and the construction of the restriction and prolongation operators in Section 4.1.3, we propose the Closest Point Method V-Cycle Multigrid algorithm. We remark that this algorithm does not explicitly involve the side condition as in (2.1.5), but instead follows each iteration with a closest point extension as in the original Ruuth–Merriman approach [79].

[2]An alternative approach is simply to use one of the standard nest-grid multigrid approaches (although we have not tested this here).

Algorithm 2 (Closest Point Method V-Cycle Algorithm) *Denote by $\mathcal{B}^h$ and $\mathcal{B}^{2h}$ the fine and next coarser grids. Denote by $\boldsymbol{E}^h$ and $\boldsymbol{E}^{2h}$ the extension matrices on the fine and next coarser grids (see Sec. 2.2.1). Denote the shifted Cartesian Laplacian matrix, the solution and right-hand side on the fine grid by $\tilde{\boldsymbol{A}}^h = c\boldsymbol{I}^h - \boldsymbol{L}^h$, $\boldsymbol{u}^h$, and $\boldsymbol{f}^h$, and the corresponding variables on the next coarser grid by $\tilde{\boldsymbol{A}}^{2h} = c\boldsymbol{I}^{2h} - \boldsymbol{L}^{2h}$, $\boldsymbol{u}^{2h}$, and $\boldsymbol{f}^{2h}$. Let $\boldsymbol{E}_h^{2h}$ and $\boldsymbol{E}_{2h}^h$ be the restriction and prolongation matrices (which are also extensions as in Sec. 4.1.3). We compute $\boldsymbol{u}^h = V^h(\boldsymbol{u}^h, \boldsymbol{f}^h)$ by recursively calling:*

1. *Starting from some initial guess $\boldsymbol{u}^h$ which is constant along the normals to the surface (e.g., $\boldsymbol{u}^h = \boldsymbol{0}^h$) on the finest grid, do ν_1 steps of Jacobi iteration with respect to $\tilde{\boldsymbol{A}}^h$, each step followed by a closest point extension:*

$$\begin{aligned} &for\ i = 1 : \nu_1,\ do \\ &\qquad \boldsymbol{u}^h \ := \ \boldsymbol{u}^h + diag(\tilde{\boldsymbol{A}}^h)^{-1}(\boldsymbol{f}^h - \tilde{\boldsymbol{A}}^h\boldsymbol{u}^h); \\ &\qquad \boldsymbol{u}^h \ := \ \boldsymbol{E}^h\boldsymbol{u}^h. \end{aligned}$$

2. *Compute the residual and restrict it to the coarser grid,*

$$\begin{aligned} \boldsymbol{r}^h \ &:= \ \boldsymbol{f}^h - \tilde{\boldsymbol{A}}^h\boldsymbol{u}^h, \\ \boldsymbol{f}^{2h} \ &:= \ \boldsymbol{E}_h^{2h}\boldsymbol{r}^h. \end{aligned} \tag{4.1.3}$$

   If $\mathcal{B}^{2h}$ is the coarsest grid, solve $\boldsymbol{A}^{2h}\boldsymbol{u}^{2h} = \boldsymbol{f}^{2h}$ directly; else

$$\begin{aligned} \boldsymbol{u}^{2h} \ &:= \ \boldsymbol{0}^{2h}, \\ \boldsymbol{u}^{2h} \ &:= \ V^{2h}(\boldsymbol{u}^{2h}, \boldsymbol{f}^{2h}). \end{aligned} \tag{4.1.4}$$

3. *Correct the fine grid solution using coarse grid solution:*

$$\boldsymbol{u}^h := \boldsymbol{u}^h + \boldsymbol{E}_{2h}^h\boldsymbol{u}^{2h}.$$

4. *Do ν_2 steps of Jacobi iteration with respect to $\tilde{\boldsymbol{A}}^h$, each step followed by a closest point extension:*

$$\begin{aligned} &for\ i = 1 : \nu_2,\ do \\ &\qquad \boldsymbol{u}^h \ := \ \boldsymbol{u}^h + diag(\tilde{\boldsymbol{A}}^h)^{-1}(\boldsymbol{f}^h - \tilde{\boldsymbol{A}}^h\boldsymbol{u}^h); \\ &\qquad \boldsymbol{u}^h \ := \ \boldsymbol{E}^h\boldsymbol{u}^h. \end{aligned}$$

Remark 4.1.1 *If we perform (4.1.4) ν times in Step 2, then we obtain the ν-Cycle scheme (see e.g., [13]). In particular, if $\nu = 2$ then we get the W-Cycle scheme. As we shall see in Section 4.1.5, the W-Cycle scheme performs similarly to the V-Cycle scheme, so we shall focus on the development and test of V-Cycle schemes.*

4.1.5 Numerical examples for curves and surfaces without boundaries

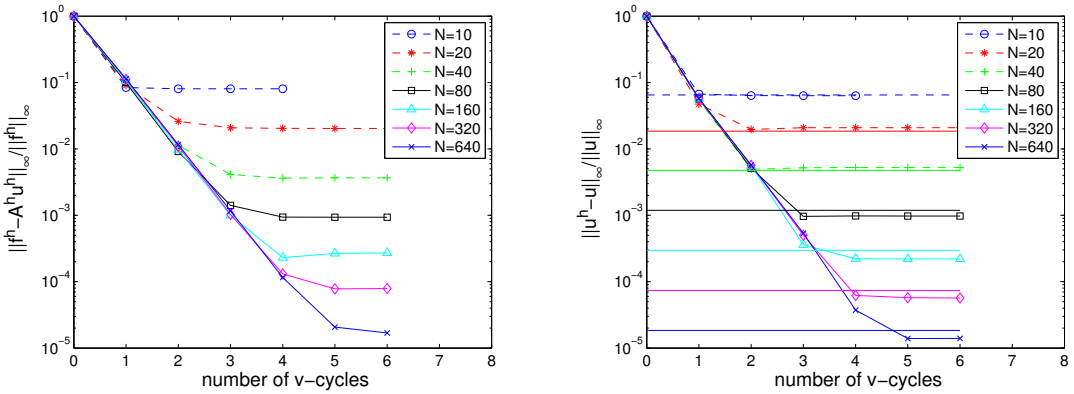
We test our V-Cycle solver for the shifted Poisson's equation on several curves and surfaces without boundaries. We use cubic (degree 3) interpolations to construct $\boldsymbol{E}^h$'s on each grid level. In each round of the V-Cycle, $\nu_1 = 3$ pre-smoothings and $\nu_2 = 3$ post-smoothings are applied. For the restriction and prolongation operators, linear (degree 1) interpolations are used for the closest point extensions. The linear system on the coarsest grid level is solved with Matlab's backslash. The stopping criteria for all the tests are $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\|_\infty / \|\boldsymbol{x}^k\|_\infty < 10^{-6}$, where $\boldsymbol{x}^k$ and $\boldsymbol{x}^{k+1}$ are the iterative solutions after the k-th and $(k+1)$-st rounds of V-Cycles.

As described in Algorithm 2, we shall start from zero initial guesses for our tests. We also conduct experiments starting from random initial guesses with magnitude around 0.5, and our multigrid algorithm can eliminate the high frequency part of the initial errors effectively. The convergence results for the cases with random initial guesses are similar to those with a zero initial guess. So we shall only present results with zero initial guesses for all the following tests.

4.1.5.1 Unit circle

We consider the simplest example from Section 2.5.1, where we solve the shifted Poisson's equation $-\Delta_{\mathcal{S}} u + u = f$ on the unit circle. The exact solution is chosen to be $\sin(\theta) + \sin(12\theta)$.

To test the convergence rate of our multigrid solver, we keep the coarsest grid size unchanged, and reduce the finest grid size. Figure 4.2(a) shows the rate of decrease of the residuals of the algebraic linear systems, and Figure 4.2(b) shows the rate of decrease of the numerical errors. Before the numerical solutions achieve the stopping criterion, they have already reached the limit of the discretization errors, and the errors will not decrease after that; so in Figure 4.2(b) as the number of V-Cycles increases, the errors first decrease and then stay almost the same. This same behaviour is also observed in the residuals in Figure 4.2(a): as the number of V-Cycles increases, the residuals initially decrease. However, because we introduce additional discretization errors to the original linear system $\boldsymbol{A}^h \boldsymbol{u}^h = \boldsymbol{f}^h$ (by performing discrete closest point extensions after each smoothing step), the residuals eventually stagnate at the level of this additional discretization error. In both Figures 4.2(a) and 4.2(b), the rate of decrease (the slope of the lines) stays almost the same as we refine the finest mesh, which is expected and desirable in a multigrid algorithm.



(a) Decrease of relative residuals

(b) Decrease of relative errors

Figure 4.2: Convergence results on a circle, where the curves show a sequence of progressively finer grids, each solved with the V-Cycle multigrid method. Each curve shows how the relative residuals/errors decrease as the number of V-Cycles increases. Here $N = \frac{1}{\Delta x}$, where Δx is the grid size of the finest mesh for that curve. For each case we fix the grid size of the coarsest grid to be 0.2, which corresponds to $N = 5$. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.

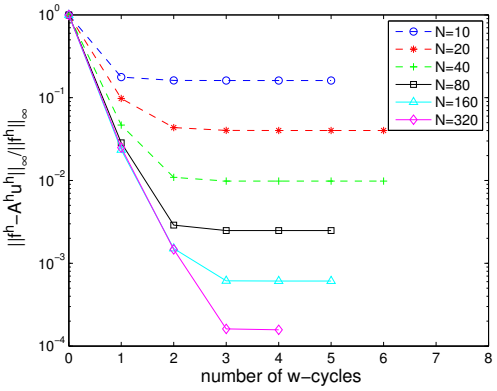
We also test the W-Cycle scheme applied to the same problem, and the convergence results are recorded in Figure 4.3. Comparing the results in Figures 4.2 and 4.3, we observe that the W-Cycle scheme convergences faster than the V-Cycle scheme in the first few cycles. However, the W-Cycle scheme still needs roughly the same number of cycles as the V-Cycle scheme to reach the stopping criterion. In the rest of this thesis we shall focus on the V-Cycle scheme since it generally costs less.

4.1.5.2 Bean-shaped curve

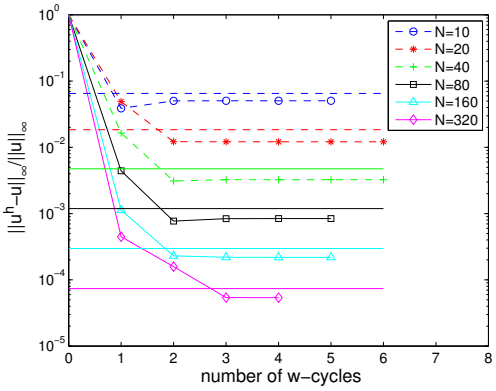
We test the shifted Poisson's equation on the bean-shaped curve as in Section 2.5.2, and the same exact solution is chosen for convergence studies. Figure 4.4(a) shows the numerical solution for the embedding equation with meshsize $\Delta x = 0.1$. Figure 4.4(b) shows the convergence results of the multigrid method. Note again that the convergence rate (slope of lines) is essentially independent of the finest grid size.

4.1.5.3 Unit sphere

In this section we consider the same numerical example as in Section 2.5.3. We solve the equation on a unit sphere centered at the origin, and choose the exact solution as the spherical harmonic $u(\theta, \phi) = \cos(3\theta)\sin(\phi)^3(9\cos(\phi)^2 - 1)$. The right-hand side function is $-\Delta_{\mathcal{S}} u + u = -29u(\theta, \phi)$. Figure 4.5(a) shows the numerical solution for the

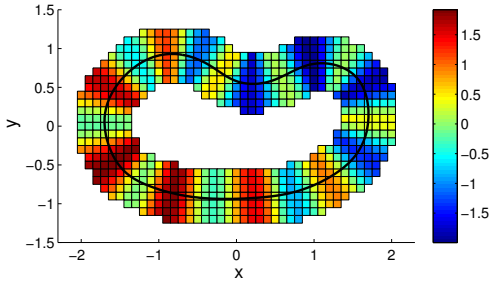


(a) Decrease of relative residuals

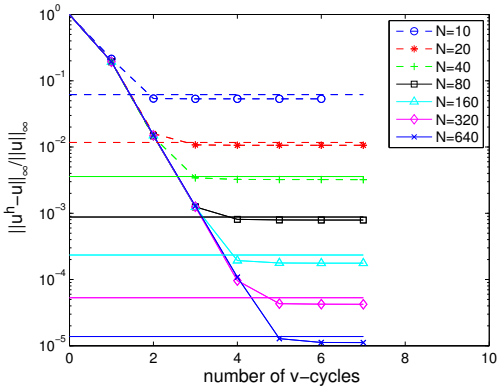


(b) Decrease of relative errors

Figure 4.3: W-Cycle convergence results for the shifted Poisson's equation on the unit circle. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.



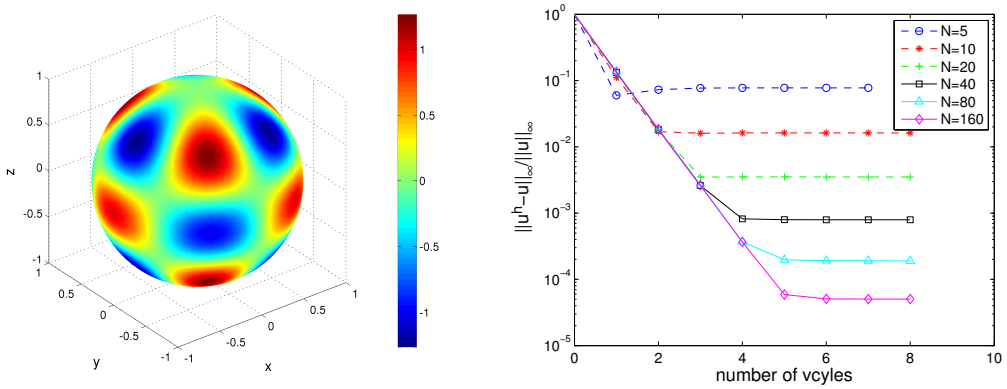
(a) Numerical solution



(b) Decrease of relative errors

Figure 4.4: V-Cycle convergence results on a bean shaped curve. The horizontal lines in figure (b) are the numerical errors given by Matlab's backslash.

embedding equation with grid size $\Delta x = 0.1$, visualized using the parametrization. Figure 4.5(b) shows the convergence results of the multigrid method.



(a) Numerical solution (b) Decrease of relative errors

Figure 4.5: V-Cycle convergence results on a unit sphere.

Besides convergence studies, we also investigate the computational cost including the degrees of freedom and matrix storage for the discrete linear system (see Table 4.1). Because our 3-D computation is performed in a narrow band with bandwidth proportional to the mesh size Δx, the number of degrees of freedom scales like $O(\Delta x^{-2})$, just as if we performed a finite difference method to solve Poisson's equation in $\mathbb{R}^2$. Since we use the standard 7-point stencil, the number of nonzero entries (nnz) of the Laplacian matrix $\boldsymbol{L}^h$ is about 7 times the number of degrees of freedom. Since we use degree 3 polynomial interpolation in a dimension by dimension fashion, the number of nonzero entries of the interpolation matrix $\boldsymbol{E}_3^h$ is about $4^3 = 64$ times the number of degrees of freedom. The matrix $\tilde{\boldsymbol{A}}^h$ used in our Ruuth–Merriman smoothing approach is simply the diagonally shifted Laplacian so $\text{nnz}(\tilde{\boldsymbol{A}}^h) = \text{nnz}(\boldsymbol{L}^h)$. From the analysis in Sections 2.3 and 2.4, $\boldsymbol{M}^h$ has nearly the same sparsity pattern as $\boldsymbol{E}_3^h$, except that $\boldsymbol{M}^h$ might have more diagonal entries than $\boldsymbol{E}_3^h$, so the number of nonzero entries of $\boldsymbol{M}^h$ is about 64 or 65 times the number of degrees of freedom.

We also compare the CPU times for solving the linear system by Matlab's backslash and our V-Cycle multigrid scheme. Table 4.2 shows the performance comparison. Computations were performed on a 3.47 GHz Intel Xeon X5690 6-core processor, with 100 GB memory. For the smallest problem, Matlab's backslash works better than multigrid; but for larger problems, multigrid works better. Furthermore, the CPU time of multigrid scales roughly as $O(\Delta x^{-2})$, i.e., proportional to the number of degrees of freedom, which is optimal and what we expect for a multigrid scheme. We

Table 4.1: Computational cost for the Closest Point Method on a sphere. As we decrease the mesh size by a factor of 2, the number of degrees of freedom (DOFs, i.e., length($\boldsymbol{u}^h$)) increases by a factor of 4. nnz($\boldsymbol{L}^h$) $\approx 7 \times$ DOFs, nnz($\boldsymbol{E}^h$) $\approx 64 \times$ DOFs, nnz($\boldsymbol{M}^h$) $\approx 64.5 \times$ DOFs.

Δx	length(u^h)	nnz(L^h)	nnz(E_3^h)	nnz(M^h)
0.2	3190	20758	204160	205686
0.1	10906	71962	697984	702714
0.05	41870	277358	2679680	2697038
0.025	166390	1103734	10648960	10717230
0.0125	663454	4402366	42461056	42731646
0.00625	2651254	17593318	169680256	170760222

observe similar scaling results for CPU times when applying our V-Cycle multigrid algorithm to other surfaces without boundaries (e.g., the torus in Section 4.1.5.4 and the Dziuk surface in Section 4.1.5.5) and the variable diffusion coefficient example in Section 4.1.5.6.

Δx	Backslash	V-Cycle
0.2	0.169	0.294
0.1	1.046	0.551
0.05	6.967	1.326
0.025	59.94	4.377
0.0125	466.5	16.66
0.00625	N/A	67.42

Table 4.2: CPU time for solving the linear system arising from the Poisson's equation on a unit sphere with the Closest Point Method. Matlab's backslash versus V-Cycle multigrid method.

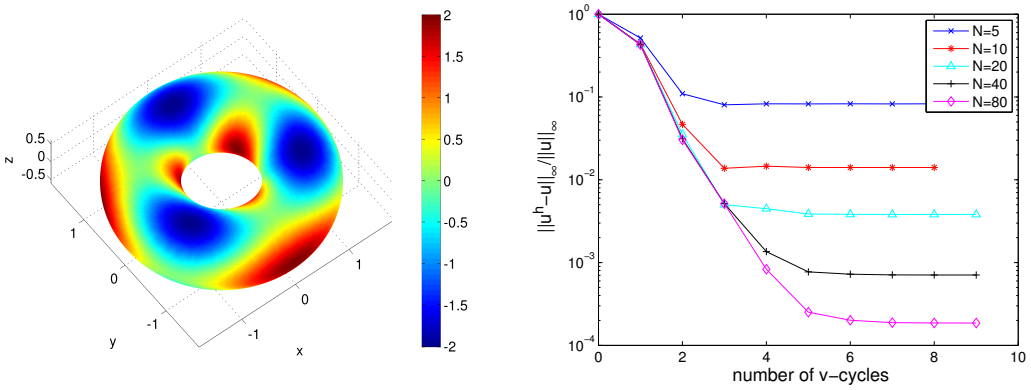
4.1.5.4 Torus

Again, to calculate an exact solution, we begin with a parametrization of a torus: $x(\theta,\phi) = (R + r\cos\phi)\cos\theta$, $y(\theta,\phi) = (R + r\cos\phi)\sin\theta$, $z(\theta,\phi) = r\sin\phi$, where ϕ, θ are in the interval $[0, 2\pi)$, R is the distance from the center of the tube to the center of the torus, and r is the radius of the tube.

We test the shifted Poisson's equation $-\Delta_{\mathcal{S}} u + u = f$ on a torus with $R = 1.2$, and $r = 0.6$. The exact solution is chosen as $u(\theta,\phi) = \sin(3\theta) + \cos(2\phi)$, and the right hand side function is $f(\theta,\phi) = \frac{9\sin(3\theta)}{(R+r\cos\phi)^2} - \frac{2\sin\phi\sin(2\phi)}{r(R+r\cos\phi)} + \frac{4\cos(2\phi)}{r^2} + u(\theta,\phi)$. Figure 4.6 shows the numerical solution and the multigrid convergence results.

4.1.5.5 A level set example

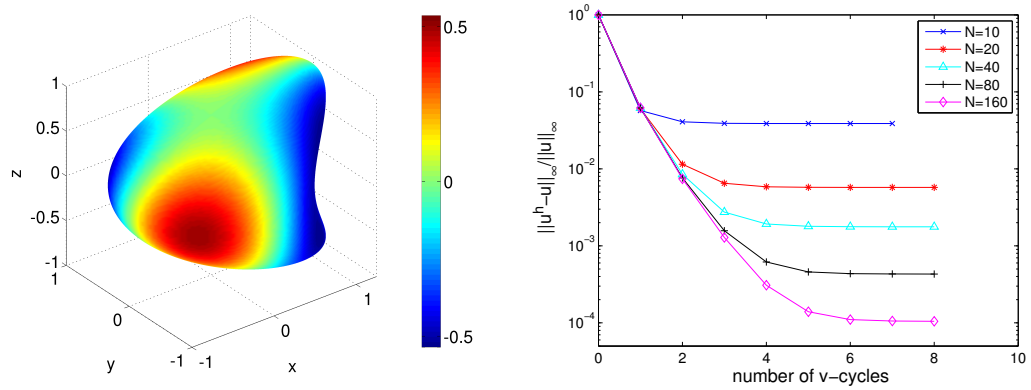
Following Dziuk [23], we consider the surface defined by $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^3 \mid (x_1 - x_3^2)^2 + x_2^2 + x_3^2 = 1\}$, and the shifted Poisson's equation $-\Delta_{\mathcal{S}} u + u = f$ on $\mathcal{S}$.



(a) Numerical solution (b) Decrease of relative errors

Figure 4.6: V-Cycle convergence results on a torus.

The exact solution is chosen to be $u(\boldsymbol{x}) = x_1 x_2$. We compute the right-hand side function by $f = -\nabla_{\mathcal{S}} \cdot \boldsymbol{v}$, where $\boldsymbol{v} = \nabla_{\mathcal{S}} u = \nabla u - (\nabla u \cdot \boldsymbol{n})\boldsymbol{n}$, and $\nabla_{\mathcal{S}} \cdot \boldsymbol{v} = \nabla \cdot \boldsymbol{v} - \sum_{j=1}^{3} (\nabla \boldsymbol{v}_j \cdot \boldsymbol{n}) \boldsymbol{n}_j$. Here $\boldsymbol{n}$ is the normal vector, and in this case, $\boldsymbol{n}(\boldsymbol{x}) = \big(x_1 - x_3^2,\ x_2,\ x_3(1 - 2(x_1 - x_3^2)) \big) \ / \ (1 + 4x_3^2(1 - x_1 - x_2^2))^{1/2} \big)$. For each grid point, we compute the closest point on $\mathcal{S}$ by minimizing the squared distance function using Newton's method. Figure 4.7 shows the numerical solution and multigrid convergence results.



(a) Numerical solution (b) Decrease of relative errors

Figure 4.7: V-Cycle convergence results on the Dziuk surface.

4.1.5.6 Variable diffusion coefficients on a sphere

Finally, we consider an example with scalar variable diffusion coefficients,

$$-\nabla_{\mathcal{S}} \cdot (a(\boldsymbol{y})\nabla_{\mathcal{S}} u(\boldsymbol{y})) + cu(\boldsymbol{y}) = f(\boldsymbol{y}), \text{ for } \boldsymbol{y} \in \mathcal{S}.$$

To formulate the continuous embedding equation, we simply choose $\boldsymbol{A}(\mathrm{cp}(\boldsymbol{x}))$ to be $a(\mathrm{cp}(\boldsymbol{x}))$ in the general embedding equation (2.1.9), and get

$$-[\nabla\cdot(a(\mathrm{cp}(\boldsymbol{x}))\nabla\tilde{u})](\mathrm{cp}(\boldsymbol{x}))+c\tilde{u}(\boldsymbol{x})+\gamma(\tilde{u}(\boldsymbol{x})-\tilde{u}(\mathrm{cp}(\boldsymbol{x}))) = \tilde{f}(\boldsymbol{x}), \quad \boldsymbol{x} \in B(\mathcal{S}). \quad (4.1.5)$$

Like for the Poisson case before, embedding equation (4.1.5) can be discretized using standard finite differences and interpolation schemes, see also [85]. The matrix formulation of the discretization is $-\boldsymbol{E}^h\tilde{\boldsymbol{L}}^h\boldsymbol{u}^h + \gamma(\boldsymbol{I}^h - \boldsymbol{E}^h)\boldsymbol{u}^h = \boldsymbol{f}^h$, where

$$\tilde{\boldsymbol{L}}^h = \boldsymbol{D}_x^b \mathrm{diag}(\boldsymbol{A}_x^f \boldsymbol{a}^h)\boldsymbol{D}_x^f + \boldsymbol{D}_y^b \mathrm{diag}(\boldsymbol{A}_y^f \boldsymbol{a}^h)\boldsymbol{D}_y^f + \boldsymbol{D}_z^b \mathrm{diag}(\boldsymbol{A}_z^f \boldsymbol{a}^h)\boldsymbol{D}_z^f.$$

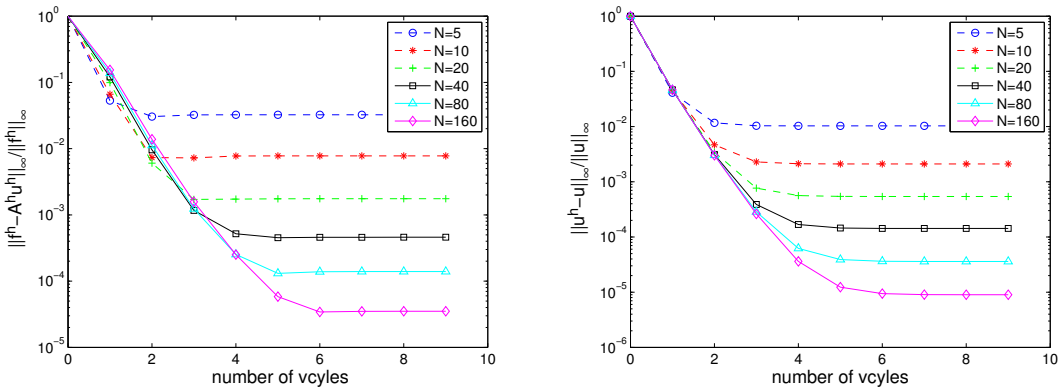
Here, we have used similar notations to those in [85]. $\boldsymbol{D}_x^b$ is the matrix corresponding to backward differences in the x direction, $\boldsymbol{D}_x^f$ is the matrix corresponding to forward differences in the x direction; $\boldsymbol{D}_y^b$, $\boldsymbol{D}_y^f$, $\boldsymbol{D}_z^b$ and $\boldsymbol{D}_z^f$ have similar meanings. The column vector $\boldsymbol{a}^h$ has the i-th entry equal to $a(\mathrm{cp}(\boldsymbol{x}_i))$, where $\boldsymbol{x}_i$ is the i-th grid point. $\boldsymbol{A}_x^f$ is the forward averaging matrix corresponding to calculating the average of two neighboring values of $\boldsymbol{a}^h$ along the positive x direction. $\boldsymbol{A}_x^f\boldsymbol{a}^h$ is a column vector and $\mathrm{diag}(\boldsymbol{A}_x^f\boldsymbol{a}^h)$ is a diagonal matrix with diagonal entries equal to $\boldsymbol{A}_x^f\boldsymbol{a}^h$; $\mathrm{diag}(\boldsymbol{A}_y^f\boldsymbol{a}^h)$ and $\mathrm{diag}(\boldsymbol{A}_z^f\boldsymbol{a}^h)$ have similar meanings.

In the multigrid algorithm, we continue to use the Ruuth–Merriman style relaxation strategy, except for changing $\boldsymbol{L}^h$ to $\tilde{\boldsymbol{L}}^h$.

We test on a unit sphere. We choose the exact solution to be $u(\phi) = \cos\phi$, the diffusion coefficient $a(\phi) = \cos\phi + 1.5$, and the right hand side function is $f(\phi) = 2\cos\phi(\cos\phi + 1.5) - (\sin\phi)^2 + u(\phi)$. The multigrid convergence results are shown in Figure 4.8 and look similar to the Poisson problems shown earlier.

4.2 Multigrid solvers for codimension 0 domains

In this section, we modify several ingredients in Algorithm 2 to obtain a new V-Cycle scheme working on codimension 0 domains. Here, our relaxation method is based on the weighted Jacobi iteration and extrapolation of ghost point values according to boundary conditions. As we shall see next, the modified relaxation method is similar to the Ruuth–Merriman smoothing strategy discussed in Section 4.1.2. Roughly



(a) Decrease of relative residuals

(b) Decrease of relative errors

Figure 4.8: V-Cycle convergence results for a variable diffusion coefficient example.

speaking, at each relaxation step, we first update interior point values using the weighted Jacobi iteration on the differential equation, and then "extend" the solution from interior points to exterior (ghost) points so that the boundary condition is satisfied. We also make minor modifications to the standard restriction and prolongation matrices used to transfer data between the fine and next coarser grid levels.

4.2.1 V-Cycle Scheme for codimension 0 domains

As we noticed in Section 3.2, the discretizations of Poisson problems on codimension 0 domains with different boundary conditions share the common form,

$$\begin{pmatrix} \boldsymbol{A}_I \\ \boldsymbol{B} \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} \boldsymbol{A}_{II} & \boldsymbol{A}_{IG} \\ \boldsymbol{B}_{GI} & \boldsymbol{B}_{GG} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_I \\ \boldsymbol{u}_G \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_I \\ \boldsymbol{g} \end{pmatrix}. \tag{4.2.1}$$

Here, $\boldsymbol{A}_I\boldsymbol{u} = \boldsymbol{f}_I$ corresponds to the discretization of the interior differential equation for non-ghost points[3], and $\boldsymbol{B}\boldsymbol{u} = \boldsymbol{g}$ corresponds to the discretization of the boundary condition for ghost points. As discussed in Section 3.2, we can represent the values of the ghost points by values of non-ghost points,

$$\boldsymbol{u}_G = -\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI}\boldsymbol{u}_I + \boldsymbol{B}_{GG}^{-1}\boldsymbol{g}. \tag{4.2.2}$$

The ideas of the modified relaxation method for codimension 0 domains are as follows. Starting from an initial $\boldsymbol{u}$, we first use the weighted Jacobi iteration to update the values of non-ghost points $\boldsymbol{u}_I$ using values of all grid points $\boldsymbol{u}$,

$$\boldsymbol{u}_I := \boldsymbol{u}_I + \omega \mathrm{diag}(\boldsymbol{A}_{II})^{-1}(\boldsymbol{f}_I - \boldsymbol{A}_I\boldsymbol{u}), \tag{4.2.3}$$

[3]Here non-ghost points refer to grid points which are not ghost points.

where $0 < \omega < 1$. Then we use (4.2.2) to update $\boldsymbol{u}_G$ using $\boldsymbol{u}_I$ so that boundary conditions are satisfied after each iteration.

Note that the above relaxation method is analogous to the Ruuth–Merriman smoothing strategy for surface elliptic equations in Section 4.1.2. Both strategies first use the standard weighted Jacobi iteration to update the solution according to the differential equation, and then immediately modify the solution so that a certain type of "constraint" is satisfied. In particular, for the usual elliptic equation on a general domain the constraint is the boundary condition, while for a surface elliptic equation it is the side condition: enforcing the solution to be constant along the normals to the surface.

Now, we still need to modify the restriction and prolongation operators to construct a complete multigrid algorithm. The restriction operator restricts the residual from fine grid to coarse grid. Because relaxations are only performed for non-ghost points (as seen in equation (4.2.3)), we do not need to compute or restrict residuals at the ghost points. We consider the new restriction operator $\boldsymbol{I}_h^{2h}$ that restricts the residual of non-ghost points from a fine grid to a coarse grid: $\boldsymbol{f}_I^{2h} = \boldsymbol{I}_h^{2h} \boldsymbol{f}_I^h$. Since the non-ghost points on the coarse grid form a strict subset of the non-ghost points on the fine grid, we perform "injection" when doing restriction, i.e., taking the value of each non-ghost point on the coarse grid to be the value at the corresponding point on the fine grid. The new prolongation operator interpolates the values from coarse grid to fine grid. Since the values of ghost points can be represented by values of non-ghost points at each grid level, we only need to interpolate the values of non-ghost points on the fine grid from values of coarse grid points when doing prolongation. We denote the prolongation operator by $\boldsymbol{I}_{2h}^h$ so that $\boldsymbol{u}_I^h = \boldsymbol{I}_{2h}^h \boldsymbol{u}^{2h}$.

Armed with the new relaxation method and the construction of the restriction and prolongation operators, we obtain the following V-Cycle algorithm for codimension 0 domains.

Algorithm 3 (V-Cycle Algorithm for Codimension 0 domains) *Denote by $\mathcal{B}^h$ and $\mathcal{B}^{2h}$ the fine and next coarser grids. Let all variables with superscript h be defined on the fine grid, and all variables with superscript $2h$ be defined on the coarse grid. Let $\omega \in (0,1)$ be the weight parameter of the weighted Jacobi iteration. Let $\boldsymbol{I}_h^{2h}$ and $\boldsymbol{I}_{2h}^h$ be the restriction and prolongation matrices. For all the grid levels except the finest one, set $\boldsymbol{g} = \boldsymbol{0}$. We compute $\boldsymbol{u}^h = V^h(\boldsymbol{u}^h, \boldsymbol{f}^h, \boldsymbol{g}^h)$ by recursively calling:*

1. *Starting from some initial guess* $\boldsymbol{u}^h$ *(e.g.,* $\boldsymbol{u}^h = \boldsymbol{0}^h$*) on the finest grid, perform* ν_1 *steps of the modified weighted Jacobi iteration,*

$$\begin{aligned} &for\ i = 1 : \nu_1,\ do \\ &\qquad \boldsymbol{u}_I^h &:=\ & \boldsymbol{u}_I^h + \omega\, diag(\boldsymbol{A}_{II}^h)^{-1}(\boldsymbol{f}_I^h - \boldsymbol{A}_I^h \boldsymbol{u}^h), \\ &\qquad \boldsymbol{u}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h. \end{aligned}$$

2. *Compute the residual for non-ghost points and restrict it to the coarser grid,*

$$\begin{aligned} \boldsymbol{r}_I^h &:=\ \boldsymbol{f}_I^h - \boldsymbol{A}_I^h \boldsymbol{u}^h, \\ \boldsymbol{f}_I^{2h} &:=\ \boldsymbol{I}_h^{2h} \boldsymbol{r}_I^h. \end{aligned}$$

If $\mathcal{B}^{2h}$ *is the coarsest grid, solve (4.2.1) directly; else*

$$\begin{aligned} \boldsymbol{u}^{2h} &:=\ \boldsymbol{0}^{2h}, \\ \boldsymbol{u}^{2h} &:=\ V^{2h}(\boldsymbol{u}^{2h}, \boldsymbol{f}^{2h}, \boldsymbol{g}^{2h}). \end{aligned}$$

3. *Correct the values of non-ghost points on the fine grid using coarse grid values,*

$$\boldsymbol{u}_I^h := \boldsymbol{u}_I^h + \boldsymbol{I}_{2h}^h \boldsymbol{u}^{2h}.$$

Then modify the values of ghost points on the fine grid,

$$\boldsymbol{u}_G^h := -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h.$$

4. *Perform* ν_2 *steps of the modified weighted Jacobi iteration,*

$$\begin{aligned} &for\ i = 1 : \nu_2,\ do \\ &\qquad \boldsymbol{u}_I^h &:=\ & \boldsymbol{u}_I^h + \omega\, diag(\boldsymbol{A}_{II}^h)^{-1}(\boldsymbol{f}_I^h - \boldsymbol{A}_I^h \boldsymbol{u}^h), \\ &\qquad \boldsymbol{u}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h. \end{aligned}$$

4.2.2 Numerical results for codimension 0 domains

In this section, we test the Multigrid Algorithm 3 applied to Poisson problems on several 2-D and 3-D domains with Dirichlet, Neumann and Robin boundary conditions. The right hand functions are computed from the exact solutions and the partial differential equations. The boundary conditions are set up according to the exact solutions and the geometry of the boundaries. For all test cases, we use the lower order approximation to boundary conditions augmented with quadratic interpolation. In other words, for Dirichlet boundary conditions we use discretization (3.2.7) with

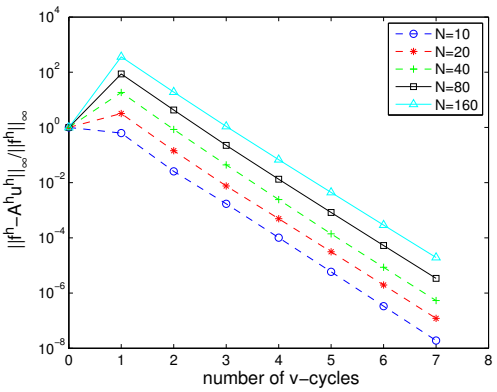
$p = 2$; for Neumann boundary conditions we use discretization (3.2.14) with $q = 2$; while for Robin boundary conditions we use discretization (3.2.25) with $q = 2$.

For each test case, the parameters in the multigrid algorithm are set up as follows. The weight ω in the weighted Jacobi iteration is chosen to be 0.8. In each round of the V-Cycle, $\nu_1 = 3$ pre-smoothings and $\nu_2 = 3$ post-smoothings are applied. "injection" is used to build the restriction operators, and linear interpolations are used for the construction of prolongation operators. The linear system on the coarsest grid level is solved with Matlab's backslash. The stopping criteria for all the tests are $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\|_\infty / \|\boldsymbol{x}^k\|_\infty < 10^{-6}$, where $\boldsymbol{x}^k$ and $\boldsymbol{x}^{k+1}$ are the iterative solutions after the k-th and $(k+1)$-st rounds of V-Cycles.

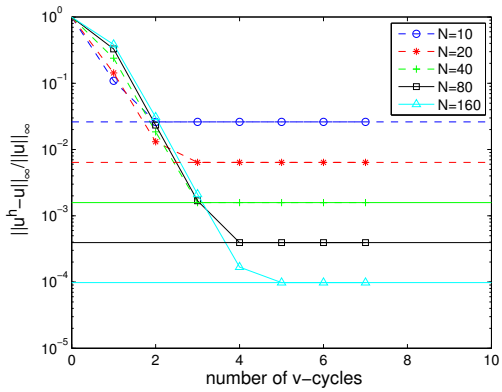
4.2.2.1 Poisson problems on a disk

We first consider the test cases in Section 3.2.5.1, where we solve Poisson problems with Dirichlet, Neumann and Robin boundary conditions on a disk centered at the origin with radius $\sqrt{2}$. For each case of boundary conditions, the exact solution is chosen to be $r^5 \sin(2\theta)$ in polar coordinates. Figure 4.9 shows the V-Cycle convergence results for Poisson's equation with Dirichlet boundary conditions (3.2.2). The relative residuals and errors are measured only for non-ghost grid points. Figure 4.9(b) shows that the relative errors change in a similar way to the test cases on curves and surfaces without boundaries in Section 4.1.5. The errors first decrease and then stay almost the same as we perform V-Cycles, and the rate of decrease does not deteriorate as we refine the finest grid level. Figure 4.9(a) shows that the relative residuals decrease all the way as we perform V-Cycles, and do not stay the same even when the relative errors do. This is in contrast to the cases in Section 4.1.5 where the trends of residuals and errors are similar. The reason is that we are only measuring the residuals for non-ghost points. We perform iterations to solve the exact original linear system for the rows corresponding to non-ghost points, and there are no interpolation errors introduced as in the case of curves and surfaces in Section 4.1.5.

Figure 4.10 shows the V-Cycle convergence results for the shifted Poisson's equation with Neumann boundary conditions (3.2.11) with $c = 1$. Figure 4.11 shows the V-Cycle convergence results for Poisson's equation with Robin boundary conditions (3.2.19) where $\varphi = 2 + \cos(\theta)$ in polar coordinates. The convergence results for Neumann and Robin boundary conditions are similar to the case of Dirichlet boundary conditions.

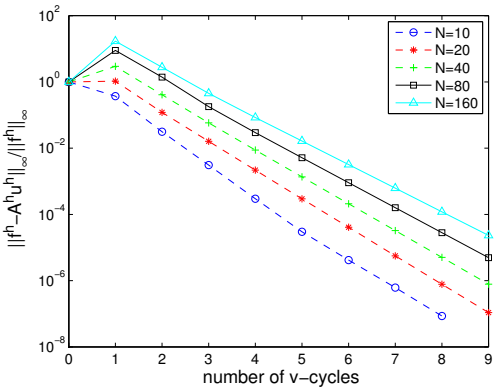


(a) Decrease of relative residuals

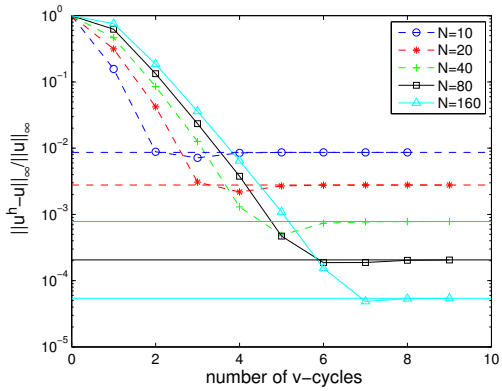


(b) Decrease of relative errors

Figure 4.9: V-Cycle convergence results for Poisson's equation on a disk with Dirichlet boundary conditions.

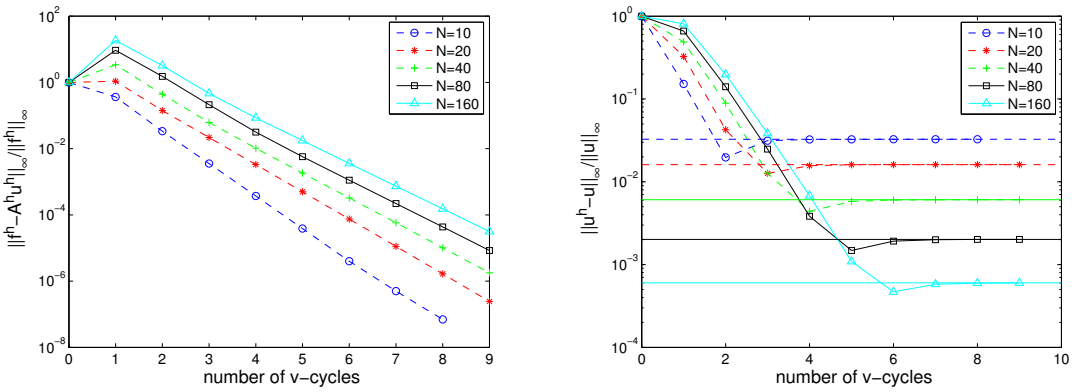


(a) Decrease of relative residuals



(b) Decrease of relative errors

Figure 4.10: V-Cycle convergence results for the shifted Poisson's equation on a disk with Neumann boundary conditions.



(a) Decrease of relative residuals

(b) Decrease of relative errors

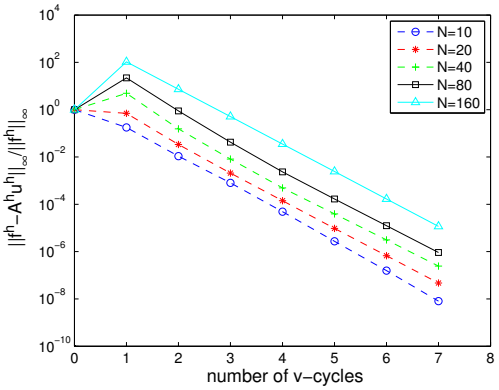
Figure 4.11: V-Cycle convergence results for Poisson's equation on a disk with Robin boundary conditions.

4.2.2.2 Poisson problems on a bean-shaped domain

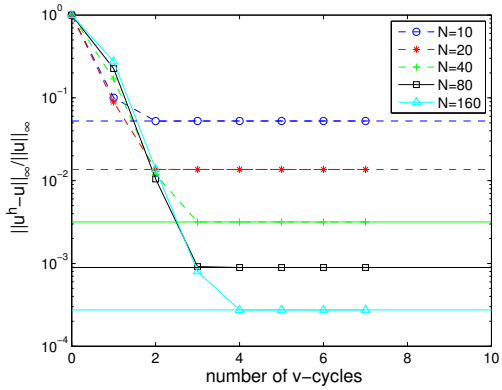
We then investigate the test cases in Section 3.2.5.2, where Poisson problems are solved on a domain surrounded by a bean-shaped curve as parametrized in Section 2.5.2. The test cases are set up exactly as in Section 3.2.5.2. We adopt our multigrid algorithm to solve the linear system for each of the test cases. Figure 4.12 shows the V-Cycle convergence results for Poisson's equation with Dirichlet boundary conditions (3.2.2). Figure 4.13 shows the V-Cycle convergence results for the shifted Poisson's equation with Neumann boundary conditions (3.2.11) with $c = 1$. Figure 4.14 shows the V-Cycle convergence results for Poisson's equation with Robin boundary conditions (3.2.19) where $\varphi = 2+x+y$. All the V-Cycle convergence results are similar to the disk cases in Section 4.2.2.1.

4.2.2.3 Poisson problems inside a unit ball

Finally we look at a 3-D example where the underlying domain is a unit ball. We choose the exact solution $u(x, y, z) = \sin(x)\sin(y)\sin(z) + \sin(4x)\sin(4y)\sin(4z)$. We compute the right hand side function straightforwardly by taking the Laplacian of u. The Dirichlet boundary condition are set up by evaluating u on the boundary. For the Neumann conditions, we calculate $\frac{\partial u}{\partial \boldsymbol{n}}$ by computing the dot product of $\left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}\right)$ and $\left(\frac{x}{\sqrt{x^2+y^2+z^2}}, \frac{x}{\sqrt{x^2+y^2+z^2}}, \frac{x}{\sqrt{x^2+y^2+z^2}}\right)$. The Robin boundary conditions are set up by computing $u+\varphi\frac{\partial u}{\partial \boldsymbol{n}}$ on the boundary where $\varphi = 2+xyz$. Figure 4.15 shows the V-Cycle convergence results for Poisson's equation with Dirichlet boundary conditions (3.2.2). Figure 4.16 shows the V-Cycle convergence results for the shifted Poisson's equation

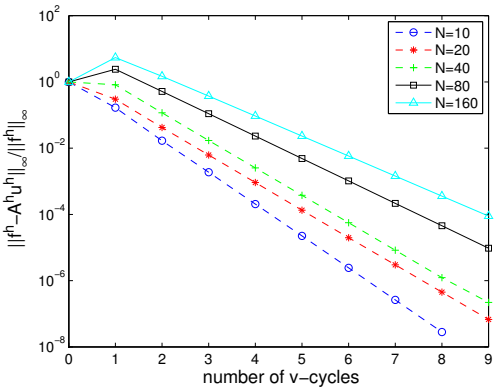


(a) Decrease of relative residuals

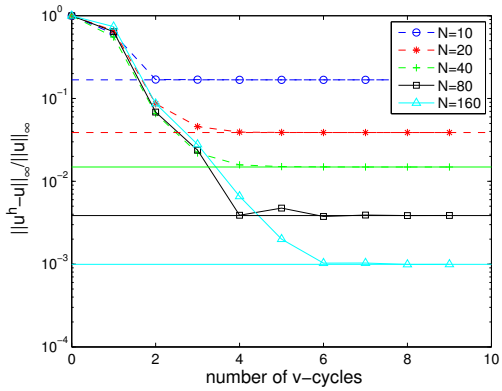


(b) Decrease of relative errors

Figure 4.12: V-Cycle convergence results for Poisson's equation on a bean-shaped domain with Dirichlet boundary conditions.

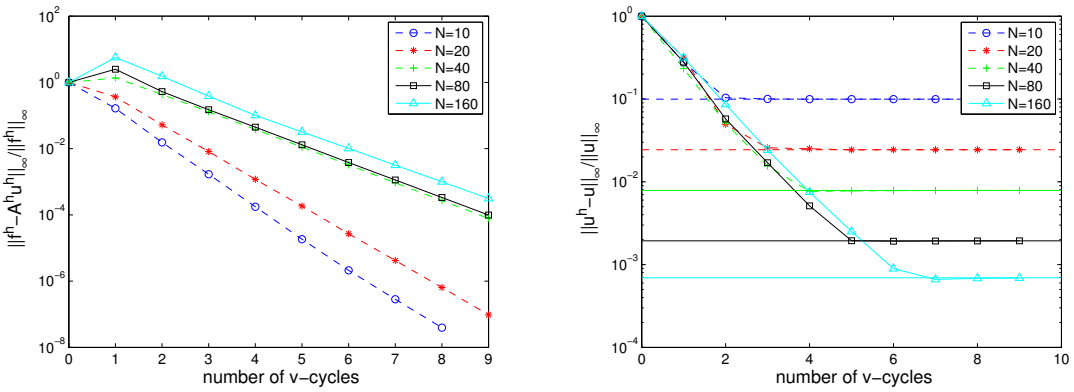


(a) Decrease of relative residuals



(b) Decrease of relative errors

Figure 4.13: V-Cycle convergence results for the shifted Poisson's equation on a bean-shaped domain with Neumann boundary conditions.



(a) Decrease of relative residuals

(b) Decrease of relative errors

Figure 4.14: V-Cycle convergence results for Poisson's equation on a bean-shaped domain with Robin boundary conditions.

with Neumann boundary conditions (3.2.11) with $c = 1$. Figure 4.17 shows the V-Cycle convergence results for Poisson's equation with Robin boundary conditions (3.2.19). The convergence results are similar to the previous 2-D examples.

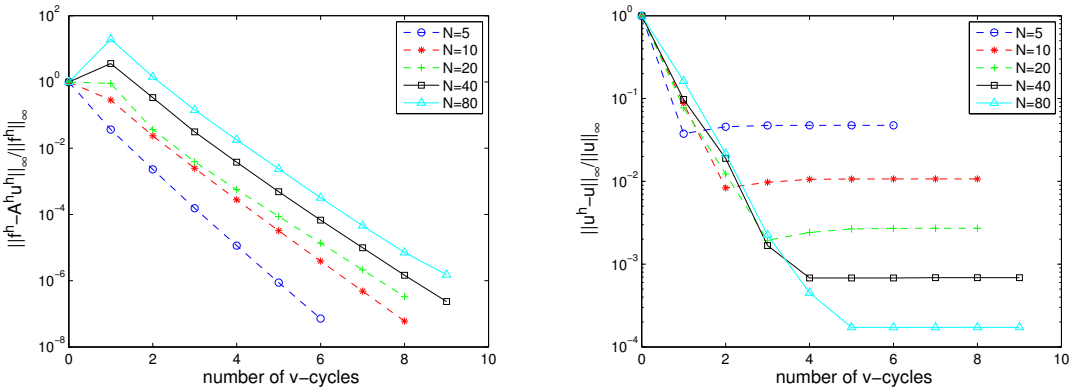
We also record the CPU times of our multigrid algorithm for the Poisson's equation in the unit ball with Dirichlet boundary conditions. The CPU times for problems with Neumann and Robin boundary conditions are similar to the Dirichlet case. The computations were performed on the same machine as in Section 4.1.5.3. Table 4.3 shows the CPU times and the ratios between them as we refine the mesh. As we can see in Table 4.3, the CPU time increases by a factor of about 5.5 when we refine the mesh size by a factor of 2. We can not obtain the optimal factor of 4 as for the sphere case in Section 4.1.5.3. The reason is that $(\boldsymbol{B}_{GG}^h)^{-1}$ appears in Algorithm 3 when we perform relaxation and prolongation. The matrix $(\boldsymbol{B}_{GG}^h)^{-1}$ is large and dense when Δx is small, so it is not plausible to compute and store $(\boldsymbol{B}_{GG}^h)^{-1}$ directly. We need to solve a linear system with coefficient matrix $\boldsymbol{B}_{GG}^h$ to update the values of ghost points when we perform relaxation and prolongation in Algorithm 3. Although we can not achieve the the optimal computational cost in this case, we think that the CPU times are still reasonable.

4.3 Multigrid solvers for surfaces with boundaries

In this section, we combine the ideas in Algorithm 2 and Algorithm 3 to develop a V-Cycle scheme for elliptic equations on curves and surfaces with boundaries.

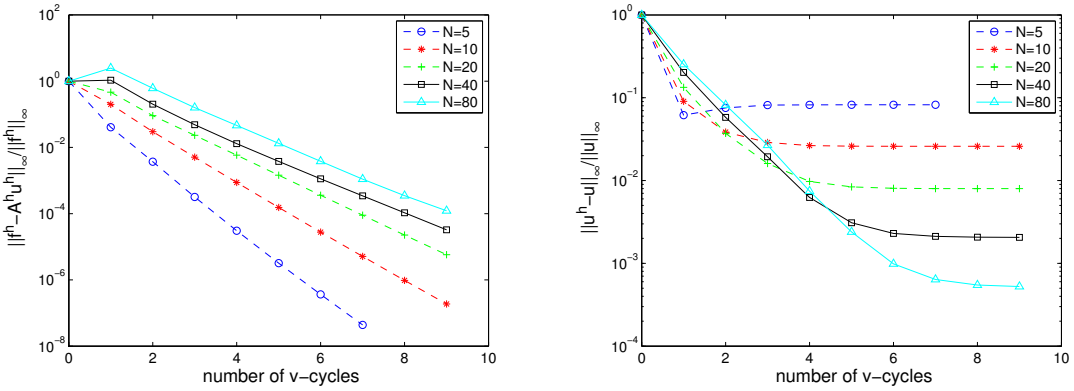
Δx	CPU time	Ratio
0.2	0.158	
0.1	0.927	5.8
0.05	4.779	5.2
0.025	26.46	5.5
0.0125	149.0	5.6

Table 4.3: CPU time for applying the Multigrid Algorithm 3 to solving the Poisson's equation in a unit ball with Dirichlet boundary conditions.



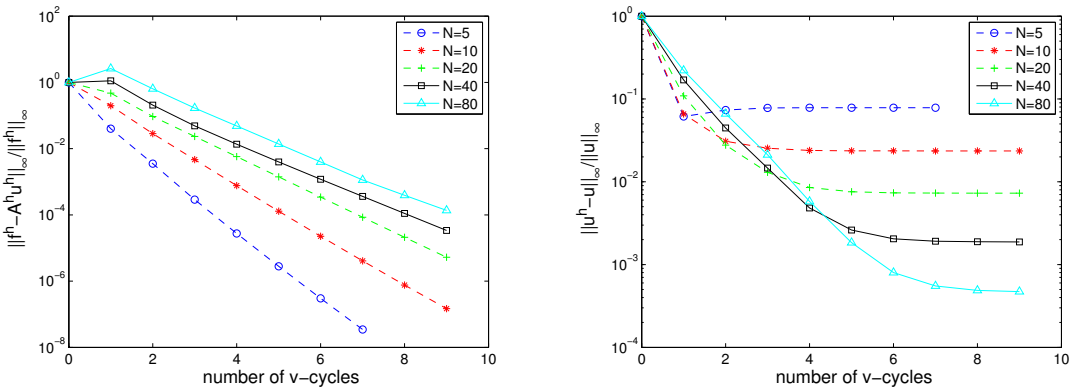
(a) Decrease of relative residuals (b) Decrease of relative errors

Figure 4.15: V-Cycle convergence results for Poisson's equation in a ball with Dirichlet boundary conditions.



(a) Decrease of relative residuals (b) Decrease of relative errors

Figure 4.16: V-Cycle convergence results for the shifted Poisson's equation in a ball with Neumann boundary conditions.



(a) Decrease of relative residuals

(b) Decrease of relative errors

Figure 4.17: V-Cycle convergence results for Poisson's equation in a ball with Robin boundary conditions.

4.3.1 V-Cycle scheme for surfaces with boundaries

From Section 3.3, the discretization for elliptic PDEs on curves and surfaces with boundaries has the following form,

$$\begin{pmatrix} \boldsymbol{A}_I \\ \boldsymbol{B} \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} \boldsymbol{A}_{II} & \boldsymbol{A}_{IG} \\ \boldsymbol{B}_{GI} & \boldsymbol{B}_{GG} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_I \\ \boldsymbol{u}_G \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_I \\ \boldsymbol{g} \end{pmatrix}. \tag{4.3.1}$$

Here, $\boldsymbol{A}_I\boldsymbol{u} = \boldsymbol{f}_I$ is the discretization for the embedding equation at non-ghost points, and $\boldsymbol{B}\boldsymbol{u} = \boldsymbol{g}$ corresponds to the discretization of the boundary condition for ghost points. For details about $\boldsymbol{A}_I$ and $\boldsymbol{B}$, we refer readers back to Section 3.3.

We shall adopt Ruuth–Merriman relaxation strategy as we did for surfaces without boundary in Section 4.1.2, but we shall do something similar to what we did for codimension 0 domains in Section 4.2 for the boundary conditions. Let $\tilde{\boldsymbol{A}} = c\boldsymbol{I} - \boldsymbol{L}$, and $\tilde{\boldsymbol{A}}_I$ be the rows of $\tilde{\boldsymbol{A}}$ corresponding to non-ghost points. On each grid level, we perform Jacobi iterations for the non-ghost points with respect to $\tilde{\boldsymbol{A}}_I$, and then update values of ghost points by (4.2.2) so that boundary conditions are satisfied. We then perform a closest point extension for the solution, and update values of ghost points by (4.2.2) again. The details can be found in Algorithm 4.

We build the restriction and prolongation operators using the methods described in Section 4.1.3. The restricted residuals and prolonged solutions at the ghost points are either meaningless or incorrect, but they can simply be discarded because we do not need them. Since we only perform relaxation for non-ghost points, residuals at the ghost points are not needed. Also, to ensure that the boundary conditions are still satisfied after prolongations, we apply (4.2.2) to compute values of ghost points

from values of the remaining grid points right after prolongations. The detailed implementations are described in Algorithm 4.

Algorithm 4 (V-Cycle Algorithm for surfaces with boundaries) *Denote by $\mathcal{B}^h$ and $\mathcal{B}^{2h}$ the fine and next coarser grids. Let $\boldsymbol{E}$ be the closest point extension matrix at each grid level, $\boldsymbol{L}$ be the Laplacian matrix, and $\tilde{\boldsymbol{A}} = c\boldsymbol{I} - \boldsymbol{L}$. Let all variables with superscript h be defined on the fine grid, and all variables with superscript $2h$ be defined on the coarse grid. Let $\boldsymbol{E}_h^{2h}$ and $\boldsymbol{E}_{2h}^h$ be the restriction and prolongation matrices. For all the grid levels except the finest one, set $\boldsymbol{g} = \boldsymbol{0}$. We compute $\boldsymbol{u}^h = \boldsymbol{V}^h(\boldsymbol{u}^h, \boldsymbol{f}^h, \boldsymbol{g}^h)$ by recursively calling:*

1. *Starting from some initial guess $\boldsymbol{u}^h$ (e.g., $\boldsymbol{u}^h = \boldsymbol{0}^h$) on the finest grid, perform ν_1 steps of the modified Jacobi iteration,*

$$
\begin{aligned}
&for\ i = 1 : \nu_1,\ do \\
&\qquad \boldsymbol{u}_I^h &:=\ & \boldsymbol{u}_I^h + diag(\tilde{\boldsymbol{A}}_{II}^h)^{-1}(\boldsymbol{f}_I^h - \tilde{\boldsymbol{A}}_I^h \boldsymbol{u}^h), \\
&\qquad \boldsymbol{u}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h, \\
&\qquad \boldsymbol{u}^h &:=\ & \boldsymbol{E}^h \boldsymbol{u}^h, \\
&\qquad \boldsymbol{u}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h.
\end{aligned}
$$

2. *Compute the residual for non-ghost points and restrict it to the coarser grid,*

$$
\begin{aligned}
\boldsymbol{r}_I^h &:=\ \boldsymbol{f}_I^h - \tilde{\boldsymbol{A}}_I^h \boldsymbol{u}^h, \\
\boldsymbol{r}_G^h &:=\ \boldsymbol{0}_G^h, \\
\boldsymbol{f}^{2h} &:=\ \boldsymbol{E}_h^{2h} \boldsymbol{r}^h.
\end{aligned}
$$

   If $\mathcal{B}^{2h}$ is the coarsest grid, solve (4.3.1) directly; else

$$
\begin{aligned}
\boldsymbol{u}^{2h} &:=\ \boldsymbol{0}^{2h}, \\
\boldsymbol{u}^{2h} &:=\ V^{2h}(\boldsymbol{u}^{2h}, \boldsymbol{f}^{2h}, \boldsymbol{g}^{2h}).
\end{aligned}
$$

3. *Correct the values of non-ghost points on the fine grid using coarse grid values,*

$$\boldsymbol{u}^h := \boldsymbol{u}_I^h + \boldsymbol{E}_{2h}^h \boldsymbol{u}^{2h}.$$

   Then modify the values of ghost points on the fine grid,

$$\boldsymbol{u}_G^h := -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h.$$

4. Perform ν_2 steps of the modified Jacobi iteration,

$$
\begin{aligned}
&for\ i = 1 : \nu_2,\ do \\
&\qquad \boldsymbol{u}_I^h &:=\ & \boldsymbol{u}_I^h + diag(\tilde{\boldsymbol{A}}_{II}^h)^{-1}(\boldsymbol{f}_I^h - \tilde{\boldsymbol{A}}_I^h \boldsymbol{u}^h), \\
&\qquad \boldsymbol{u}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h, \\
&\qquad \boldsymbol{u}^h &:=\ & \boldsymbol{E}^h \boldsymbol{u}^h, \\
&\qquad \boldsymbol{u}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{B}_{GI}^h \boldsymbol{u}_I^h + (\boldsymbol{B}_{GG}^h)^{-1}\boldsymbol{g}^h.
\end{aligned}
$$

4.3.2 Numerical results for curves and surfaces with boundaries

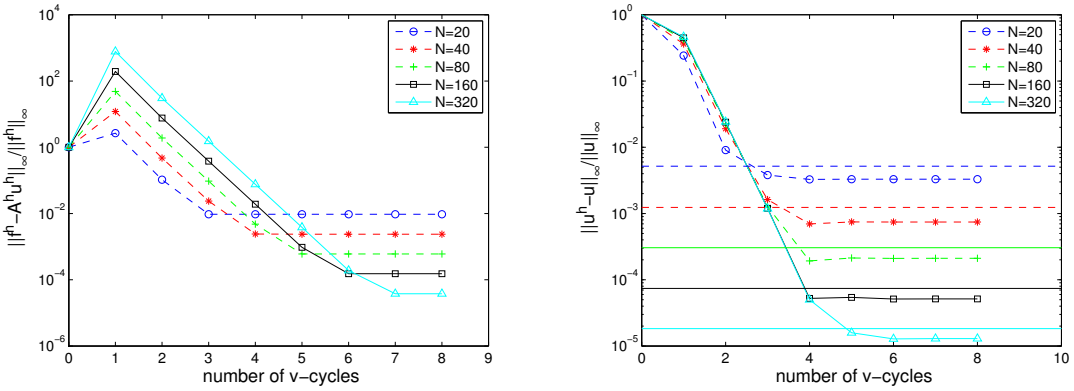
In this section, we test the Multigrid Algorithm 4 applied to Poisson problems on several curves and surfaces with boundaries. The right hand functions are computed from the exact solutions and the partial differential equations. The boundary conditions are set up according to the exact solutions and the geometry of the boundaries. For all test cases, we use the lower order approximation to boundary conditions augmented with cubic interpolation. In other words, for Dirichlet boundary conditions we use discretization (3.3.3) with $p = 3$; for Neumann boundary conditions we use discretization (3.3.11) with $q = 3$; while for Robin boundary conditions we use discretization (3.3.17) with $q = 3$;

For each test case the parameters in the multigrid algorithm are set up as follows. Jacobi iteration is used for relaxations. Numbers of pre-smoothings and post-smoothings vary a little for different cases, as can be seen in each specific example. Cubic interpolations are used for closest point extensions on each grid level. For the restriction and prolongation operators, linear interpolations are used for the closest point extensions. The linear system on the coarsest grid level is solved with Matlab's backslash. The stopping criteria for all the tests are $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\|_\infty / \|\boldsymbol{x}^k\|_\infty < 10^{-6}$, where $\boldsymbol{x}^k$ and $\boldsymbol{x}^{k+1}$ are the iterative solutions after the k-th and $(k+1)$-st rounds of V-Cycles.

4.3.2.1 Semicircle

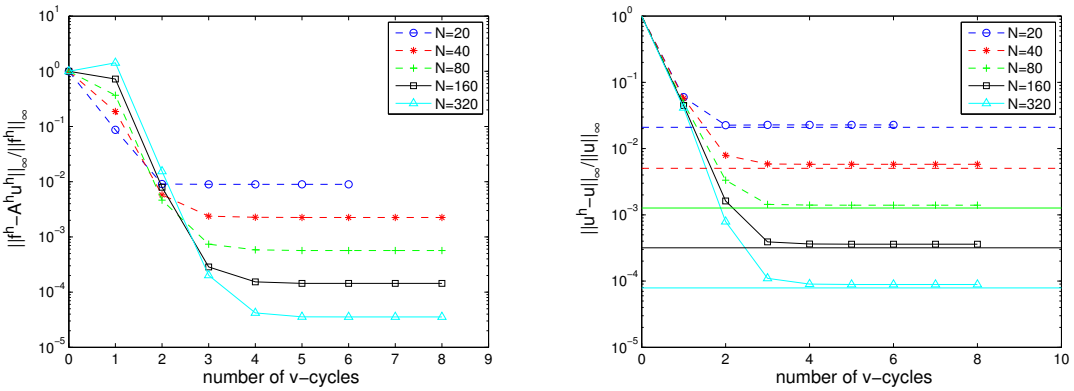
We first look at Poisson problems on a semicircle centered at the origin with radius $\sqrt{3}$. The test cases are the same as the ones in Section 3.3.4.1. In each round of the V-Cycle, $\nu_1 = 3$ pre-smoothings and $\nu_2 = 3$ post-smoothings are applied. Figure 4.18 shows the V-Cycle convergence results for Poisson's equation with Dirichlet boundary conditions (3.3.1). Figure 4.19 shows the V-Cycle convergence results for the shifted

Poisson's equation with Neumann boundary conditions (3.3.7) with $c = 1$. Figure 4.20 shows the V-Cycle convergence results for Poisson's equation with Robin boundary conditions (3.3.14) where $\varphi = 2 + \cos(\theta)$. We observe similar convergence results to the test cases for curves without boundaries in Section 4.1.5.



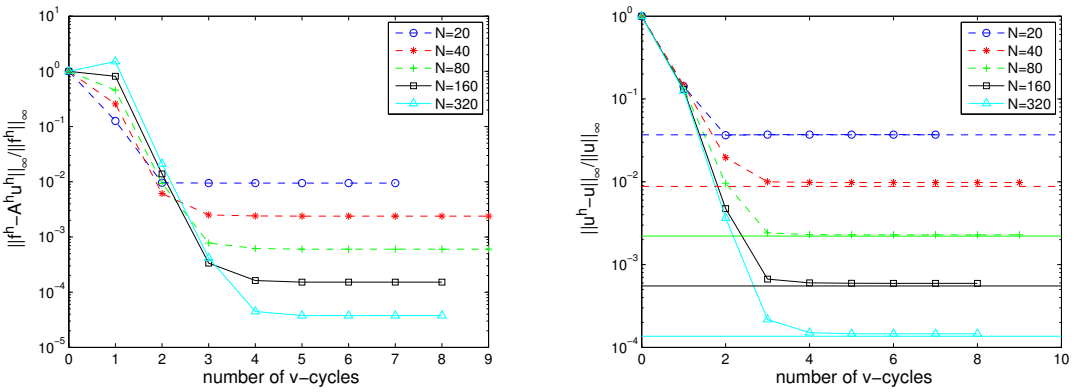
(a) Decrease of relative residuals (b) Decrease of relative errors

Figure 4.18: V-Cycle convergence results for Poisson's equation on a semicircle with Dirichlet boundary conditions. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.



(a) Decrease of relative residuals (b) Decrease of relative errors

Figure 4.19: V-Cycle convergence results for the shifted Poisson's equation on a semicircle with Neumann boundary conditions. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.



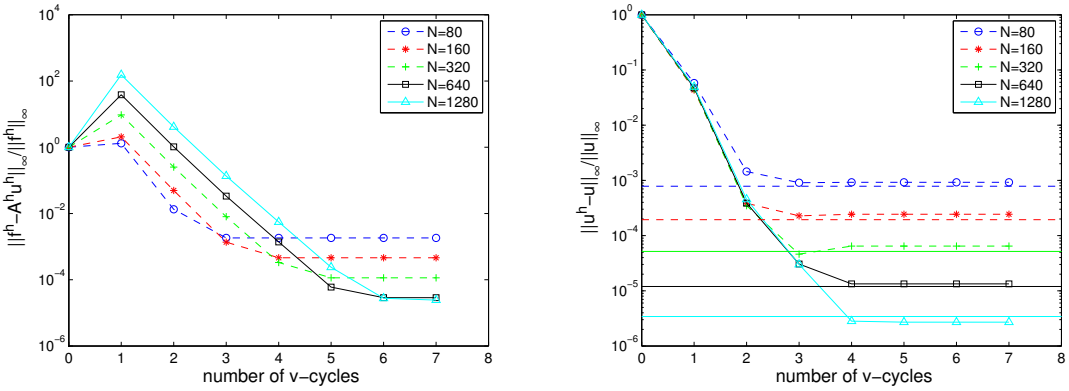
(a) Decrease of relative residuals

(b) Decrease of relative errors

Figure 4.20: V-Cycle convergence results for Poisson's equation on a semicircle with Robin boundary conditions. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.

4.3.2.2 Sine-shaped curve

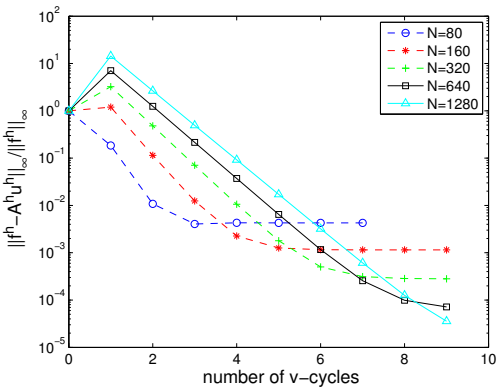
We re-solve problems on a sine-shaped curve from Section 3.3.4.2 with our multigrid method. In each round of the V-Cycle, $\nu_1 = 5$ pre-smoothings and $\nu_2 = 5$ post-smoothings are applied. Figure 4.21 shows the V-Cycle convergence results for Poisson's equation with Dirichlet boundary conditions (3.3.1). Figure 4.22 shows the V-Cycle convergence results for the shifted Poisson's equation with Neumann boundary conditions (3.3.7) with $c = 1$. Figure 4.23 shows the V-Cycle convergence results for Poisson's equation with Robin boundary conditions (3.3.14) where $\varphi = 2 + \sin(t)$.



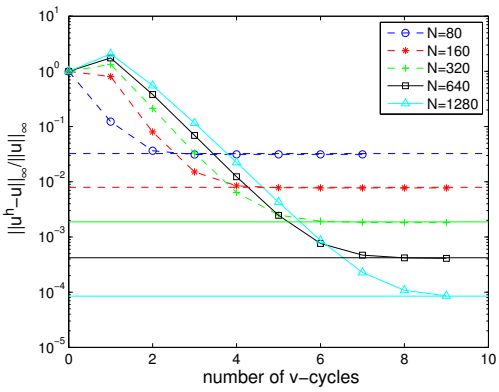
(a) Decrease of relative residuals

(b) Decrease of relative errors

Figure 4.21: V-Cycle convergence results for Poisson's equation on a sine-shaped curve with Dirichlet boundary conditions. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.

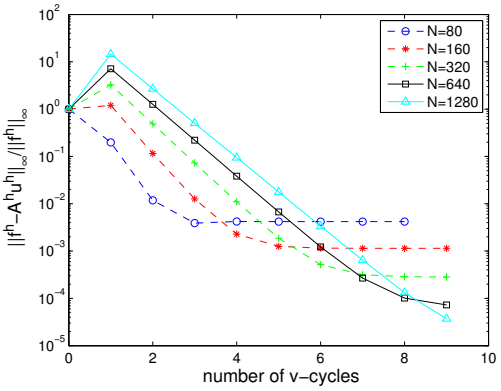


(a) Decrease of relative residuals

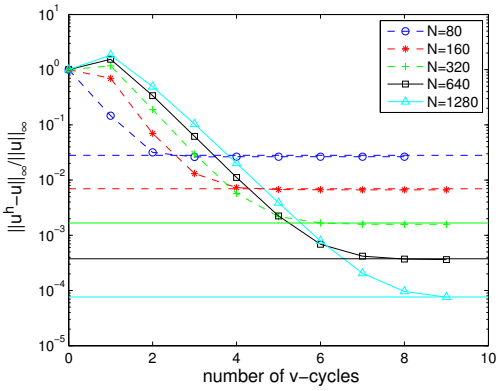


(b) Decrease of relative errors

Figure 4.22: V-Cycle convergence results for the shifted Poisson's equation on a sine-shaped curve with Neumann boundary conditions. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.



(a) Decrease of relative residuals

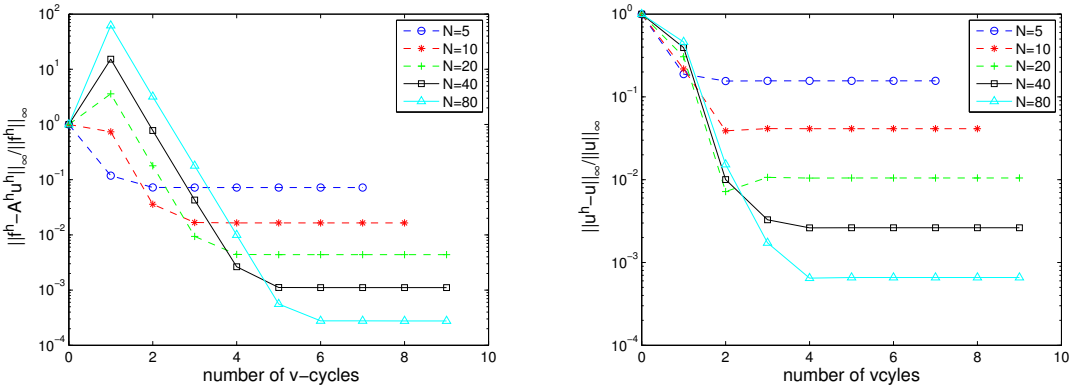


(b) Decrease of relative errors

Figure 4.23: V-Cycle convergence results for Poisson's equation on a sine-shaped curve with Robin boundary conditions. The horizontal lines in (b) are the numerical errors given by Matlab's backslash.

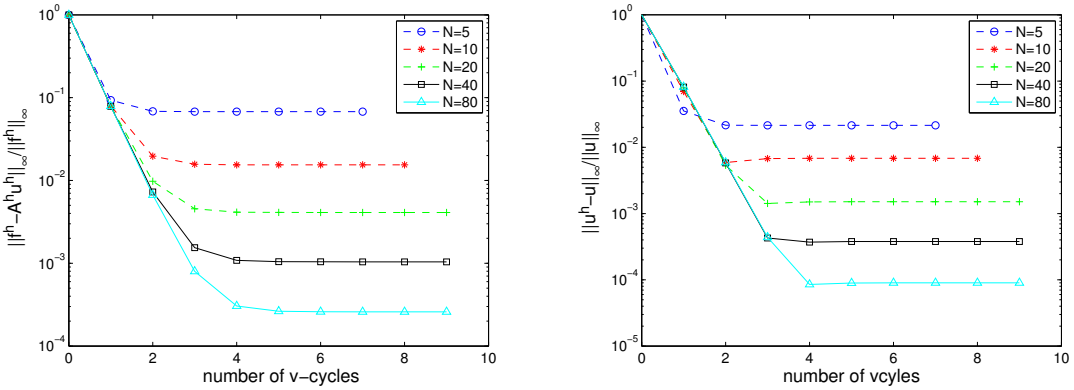
4.3.2.3 Hemisphere

Finally we re-solve surface Poisson problems on a hemisphere centered at the origin with radius $\sqrt{2}$ from Section 3.3.4.3 with our multigrid method. In each round of the V-Cycle, $\nu_1 = 5$ pre-smoothings and $\nu_2 = 5$ post-smoothings are applied. Figure 4.24 shows the V-Cycle convergence results for the surface Poisson's equation with Dirichlet boundary conditions (3.3.1). Figure 4.25 shows the V-Cycle convergence results for the shifted surface Poisson's equation with Neumann boundary conditions (3.3.7) with $c = 1$. Figure 4.26 shows the V-Cycle convergence results for the surface Poisson's equation with Robin boundary conditions (3.3.14) where $\varphi = 2 + \cos(\theta)$.



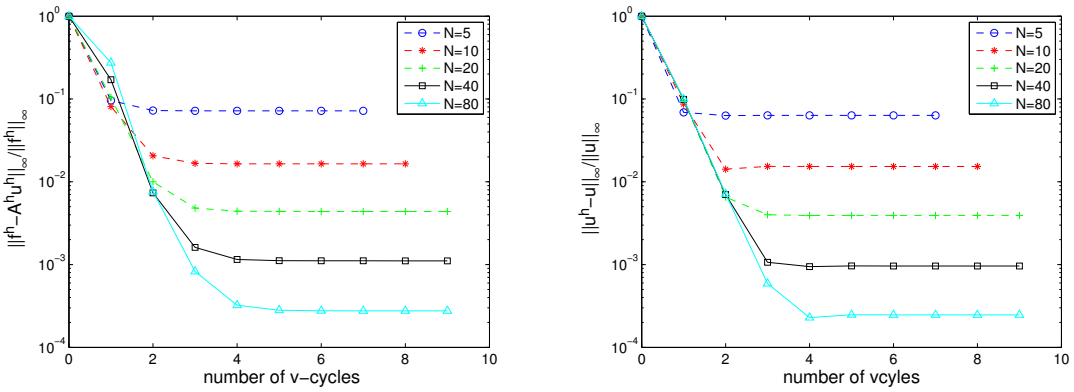
(a) Decrease of relative residuals (b) Decrease of relative errors

Figure 4.24: V-Cycle convergence results for Poisson's equation on a hemisphere with Dirichlet boundary conditions.



(a) Decrease of relative residuals (b) Decrease of relative errors

Figure 4.25: V-Cycle convergence results for the shifted Poisson's equation on a hemisphere with Neumann boundary conditions.



(a) Decrease of relative residuals

(b) Decrease of relative errors

Figure 4.26: V-Cycle convergence results for Poisson's equation on a hemisphere with Robin boundary conditions.

We also record the CPU times of our multigrid algorithm for the Poisson's equation on the hemisphere with Dirichlet boundary conditions. The CPU times for problems with Neumann and Robin boundary conditions are similar to the Dirichlet case. The computations were performed on the same machine as in Section 4.1.5.3. Table 4.3 shows that the CPU time for the hemisphere problem scales similarly to the sphere case in Section 4.1.5.3, which is desired for a multigrid algorithm.

Δx	CPU time	Ratio
0.2	0.173	
0.1	0.336	1.9
0.05	1.141	3.4
0.025	4.347	3.8
0.0125	16.52	3.8

Table 4.4: CPU time for applying the Multigrid Algorithm 4 to solving the Poisson's equation on a hemisphere with Dirichlet boundary conditions.

In summary, for all the test cases including curves and surfaces with or without boundaries and codimension zero domains, we observe optimal multigrid convergence rates. Also, the numerical solutions given by our multigrid solvers match those solved by direct methods and are 2nd-order convergent to the exact solutions. For 3-D examples the multigrid solvers are much faster than direct solvers for fine grids.

Chapter 5

Applications to coupled bulk-surface partial differential equations

In this chapter, we combine the methods in the previous chapters to solve a particular type of coupled bulk-surface partial differential equations. This type of coupled bulk-surface PDE system can be regarded as a combination of a PDE on a general domain with Robin type boundary conditions and a PDE on a surface (which is the boundary of the domain). Such type of coupled bulk-surface PDEs has wide applications in cell biology [68, 66, 76, 64] when there is coupled dynamics in the bulk and on the membrane of a cell. Another application is to model biological invasions in the bulk coupled with fast diffusion on the boundary of the bulk [7, 78].

There exists vast literature on solving PDEs on general domains, and the relatively new part for solving the coupled bulk-surface PDEs is to solve the surface PDEs. Therefore, the methods for solving coupled bulk-surface PDEs actually mainly depend on the methods to solve surface PDEs. All the methods for solving surface PDEs reviewed in Section 1.1 can be possibly extended to solve coupled bulk-surface PDEs. Some of the existing references are [68] using finite volume methods, [15, 27, 63, 64] using finite element methods, [60] using the Closest Point Method, and [1] using the diffuse domain approach.

We shall adopt the Closest Point Method to handle surface PDEs, and use the finite difference method augmented by the ghost point approach to deal with PDEs on domains with the coupling Robin type boundary conditions. Actually, this idea has already been adopted by Macdonald, Merriman and Ruuth to solve coupled bulk-surface reaction-diffusion equations for surfaces defined by point clouds in [60]. Starting from the ideas in [60], we construct a discretization for the coupled elliptic and

parabolic equations and investigate several new numerical examples in this chapter. In Section 5.1, we construct a discretization for the coupled Poisson's equations and propose a multigrid algorithm for solving the resulting linear systems. In Section 5.2, we introduce two methods: an alternating forward Euler time-stepping scheme and a method-of-lines approach for coupled time-dependent problems. In Section 5.3, we apply our method to solve the Fisher-KPP equation on a two dimensional domain coupled with a diffusion equation on its curved boundary. In Section 5.4, we apply our method to investigate the coupled bulk-surface reaction-diffusion systems.

5.1 Coupled elliptic equations

As a model problem, we consider the coupled elliptic equations which was studied by Elliott and Ranner in [27],

$$
\begin{aligned}
-\Delta v + v &= f \quad \text{in } \Omega, && \text{(5.1.1a)}\\
(\alpha v - \beta u) + \frac{\partial v}{\partial \boldsymbol{n}} &= 0 \quad \text{on } \mathcal{S}, && \text{(5.1.1b)}\\
-\Delta_{\mathcal{S}} u + u + \frac{\partial v}{\partial \boldsymbol{n}}\Big|_{\mathcal{S}} &= g \quad \text{on } \mathcal{S}. && \text{(5.1.1c)}
\end{aligned}
$$

Here Ω is a bounded domain in $\mathbb{R}^n (n = 2, 3)$ with the smooth boundary $\mathcal{S}$, α and β are positive constants, f and g are known functions in Ω and $\mathcal{S}$ respectively, and $\boldsymbol{n}$ is the unit normal on $\mathcal{S}$ pointing outwards from Ω. We refer readers to [27] for a finite element approach to solve (5.1.1). In this thesis, we shall use a finite difference method augmented by the ghost point approach and the Closest Point Method to solve (5.1.1).

5.1.1 Numerical discretization

Rearranging (5.1.1b) and substituting it into (5.1.1c), we rewrite problem (5.1.1) as follows,

$$
\begin{aligned}
-\Delta v + v &= f \quad \text{in } \Omega, && \text{(5.1.2a)}\\
\alpha v + \frac{\partial v}{\partial \boldsymbol{n}} &= \beta u \quad \text{on } \mathcal{S}, && \text{(5.1.2b)}\\
-\Delta_{\mathcal{S}} u + (1+\beta) u &= g + \alpha v|_{\mathcal{S}} \quad \text{on } \mathcal{S}. && \text{(5.1.2c)}
\end{aligned}
$$

To discretize the coupled bulk-surface elliptic problem (5.1.2), we set up two computational grids: $B(\Omega)$ for v and $B(\mathcal{S})$ for u. We then decouple problem (5.1.2) into two sub-problems and discretize them separately on $B(\Omega)$ and $B(\mathcal{S})$. We regard

(5.1.2a) and (5.1.2b) as a shifted Poisson's equation for v in Ω with Robin boundary conditions, and discretize this problem on $B(\Omega)$ using the methods in Section 3.2.3. The only extra ingredient we need is to evaluate u on $\mathcal{S}$, and this can be done via interpolations using values of grid points in $B(\mathcal{S})$. We regard (5.1.2c) as a shifted Poisson's equation for u on $\mathcal{S}$, construct an embedding equation for it using the methods in Section 2.1.1, and discretize the embedding equation on $B(\mathcal{S})$ using the methods in Section 2.2. Again the extra step is to interpolate v on $\mathcal{S}$ using values of grid points in $B(\Omega)$.

In more detail, we construct the matrix formulation for the discretization of (5.1.2). Let $\mathcal{V} = \mathcal{I} \cup \mathcal{G}$ denote the set of grid points in $B(\Omega)$, where $\mathcal{I}$ denotes the set of grid points in the interior or on the boundary of Ω and $\mathcal{G}$ denotes the set of exterior (ghost) grid points. Let $n_v = \|\mathcal{V}\|$, $n_i = \|\mathcal{I}\|$, and $n_g = \|\mathcal{G}\|$. Let $\mathcal{U}$ denote the set of grid points in $B(\mathcal{S})$, and $n_u = \|\mathcal{U}\|$.

Let $\boldsymbol{L}_v$ be the Laplacian matrix of v and $\boldsymbol{I}_v$ be the identity matrix for v. Define

$$\boldsymbol{A}_v = -\boldsymbol{L}_v + \boldsymbol{I}_v. \tag{5.1.3}$$

Let $\boldsymbol{A}_{IV} = (\boldsymbol{A}_{II}\ \boldsymbol{A}_{IG})$ be the first n_i rows of $\boldsymbol{A}_v$ corresponding to the non-ghost points. We discretize (5.1.2a) for the non-ghost points of v,

$$\boldsymbol{A}_{IV}\boldsymbol{v} = \boldsymbol{A}_{II}\boldsymbol{v}_I + \boldsymbol{A}_{IG}\boldsymbol{v}_G = \boldsymbol{f}_I. \tag{5.1.4}$$

For each ghost point $\boldsymbol{x}_i \in \mathcal{G}$, we evaluate the coupled boundary condition (5.1.2b) at $\mathrm{cp}(\boldsymbol{x}_i)$,

$$\alpha v(\mathrm{cp}(\boldsymbol{x}_i)) + \frac{\partial v}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i)) = \beta u(\mathrm{cp}(\boldsymbol{x}_i)). \tag{5.1.5}$$

Let $\boldsymbol{y}_i = 2\mathrm{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i$, then we can approximate $u(\mathrm{cp}(\boldsymbol{x}_i))$ by $\frac{1}{2}(u(\boldsymbol{x}_i) + u(\boldsymbol{y}_i))$, and approximate $\frac{\partial u}{\partial \boldsymbol{n}}(\mathrm{cp}(\boldsymbol{x}_i))$ by $\frac{u(\boldsymbol{x}_i)-u(\boldsymbol{y}_i)}{2\|\boldsymbol{x}_i-\mathrm{cp}(\boldsymbol{x}_i)\|_2}$. Substituting these approximations into (5.1.5), we obtain that

$$\frac{1}{2}\alpha(v(\boldsymbol{x}_i) + v(\boldsymbol{y}_i)) + \frac{v(\boldsymbol{x}_i) - v(\boldsymbol{y}_i)}{2\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2} = \beta u(\mathrm{cp}(\boldsymbol{x}_i)). \tag{5.1.6}$$

Rearranging (5.1.6) we obtain that

$$\begin{aligned} &\tfrac{1}{2}\left(\alpha\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2 + 1\right) v(\boldsymbol{x}_i) + \tfrac{1}{2}\left(\alpha\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2 - 1\right)) v(\boldsymbol{y}_i) \\ = \ &\beta\|\boldsymbol{x}_i - \mathrm{cp}(\boldsymbol{x}_i)\|_2 u(\mathrm{cp}(\boldsymbol{x}_i)). \end{aligned} \tag{5.1.7}$$

Let $\boldsymbol{D}$ be the $n_g \times n_g$ diagonal matrix whose diagonal entries are the distances between the ghost points and their closest points,

$$\boldsymbol{D} = \begin{pmatrix} \|\boldsymbol{x}_1 - \mathrm{cp}(\boldsymbol{x}_1)\|_2 & & \\ & \ddots & \\ & & \|\boldsymbol{x}_{n_g} - \mathrm{cp}(\boldsymbol{x}_{n_g})\|_2 \end{pmatrix}_{n_g \times n_g}.$$

Let $\boldsymbol{E}_{GU}$ be the $n_g \times n_u$ interpolation matrix interpolating values of u at $\text{cp}(\mathcal{G})$ from values of points in $\mathcal{U}$:

$$\boldsymbol{u}_{\text{cp}(\mathcal{G})} = \boldsymbol{E}_{GU}\boldsymbol{u}. \tag{5.1.8}$$

Let $\boldsymbol{I}_G = \begin{pmatrix}\boldsymbol{O}_{n_g \times n_i} & \boldsymbol{I}_{n_g \times n_g}\end{pmatrix}$. Let $\mathcal{Y} = \{\boldsymbol{y}_i \mid \boldsymbol{y}_i = 2\text{cp}(\boldsymbol{x}_i) - \boldsymbol{x}_i,\ \boldsymbol{x}_i \in \mathcal{G}\}$ and define $\boldsymbol{E}_Y^q$ as the degree q interpolation matrix interpolating values of points in $\mathcal{Y}$ from points in $\mathcal{V}$: $\boldsymbol{v}_Y = \boldsymbol{E}_Y^q \boldsymbol{v}$. Then the discretization for (5.1.7) becomes

$$\boldsymbol{B}\boldsymbol{v} - \beta \boldsymbol{D}\boldsymbol{E}_{GU}\boldsymbol{u} = 0, \text{ where } \boldsymbol{B} = \frac{1}{2}[(\alpha \boldsymbol{D}\boldsymbol{I}_G + \boldsymbol{I}_G) + (\alpha \boldsymbol{D}\boldsymbol{E}_Y^q - \boldsymbol{E}_Y^q)]. \tag{5.1.9}$$

If we partition $\boldsymbol{B}$ into $(\boldsymbol{B}_{GI}\ \boldsymbol{B}_{GG})$ and let

$$\boldsymbol{B}_{GU} = -\beta \boldsymbol{D}\boldsymbol{E}_{GU}, \tag{5.1.10}$$

then (5.1.9) can be written as

$$\boldsymbol{B}_{GI}\boldsymbol{v}_I + \boldsymbol{B}_{GG}\boldsymbol{v}_G + \boldsymbol{B}_{GU}\boldsymbol{u} = 0. \tag{5.1.11}$$

We then use methods as in Section 2.2 to discretize equation (5.1.2c) for u. To be more precise, we choose $p = 1$, $q = 3$, $\gamma = \frac{2d}{\Delta x^2}$ and $c = 1 + \beta$ in discretization (2.2.2). We modify the notations in (2.2.2) to make it clear that the matrices are defined for the surface quantity $\boldsymbol{u}$. Let $\boldsymbol{I}_u$ be the identity matrix for $\boldsymbol{u}$, $\boldsymbol{L}_u$ be the Laplacian matrix for $\boldsymbol{u}$, $\boldsymbol{E}_U^1$ be the linear closest point extension matrix for $\boldsymbol{u}$, and $\boldsymbol{E}_U^3$ be the cubic closest point extension matrix for $\boldsymbol{u}$. With this modified notation, the coefficient matrix of (2.2.2) becomes

$$\boldsymbol{A}_u = -\boldsymbol{E}_U^1 \boldsymbol{L}_u + \frac{2d}{\Delta x^2}(\boldsymbol{I}_u - \boldsymbol{E}_U^3) + (1+\beta)\boldsymbol{I}_u. \tag{5.1.12}$$

Let $\boldsymbol{E}_{UV}$ be the $n_u \times n_v$ interpolation matrix interpolating values of v at $\text{cp}(\mathcal{U})$ from values of points in $\mathcal{V}$:

$$\boldsymbol{v}_{\text{cp}(\boldsymbol{U})} = \boldsymbol{E}_{UV}\boldsymbol{v}. \tag{5.1.13}$$

Let $\boldsymbol{g}$ be the column vector storing the function g evaluated at the closest points of grid points in $\mathcal{U}$. Then (5.1.2c) is discretized as

$$\boldsymbol{A}_u\boldsymbol{u} - \alpha \boldsymbol{E}_{UV}\boldsymbol{v} = \boldsymbol{g}. \tag{5.1.14}$$

Define

$$\boldsymbol{B}_{UV} = -\alpha \boldsymbol{E}_{UV}. \tag{5.1.15}$$

If we partition $\boldsymbol{B}_{UV}$ into $(\boldsymbol{B}_{UI}\ \boldsymbol{B}_{UG})$, then (5.1.14) can be written as

$$\boldsymbol{B}_{UI}\boldsymbol{v}_I + \boldsymbol{B}_{UG}\boldsymbol{v}_G + \boldsymbol{A}_u\boldsymbol{u} = \boldsymbol{g}. \tag{5.1.16}$$

Combining (5.1.4), (5.1.11) and (5.1.16), we finally obtain a complete discretization for problem (5.1.2),

$$\begin{pmatrix} \boldsymbol{A}_{II} & \boldsymbol{A}_{IG} & \boldsymbol{O} \\ \boldsymbol{B}_{GI} & \boldsymbol{B}_{GG} & \boldsymbol{B}_{GU} \\ \boldsymbol{B}_{UI} & \boldsymbol{B}_{UG} & \boldsymbol{A}_u \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_I \\ \boldsymbol{v}_G \\ \boldsymbol{u} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_I \\ \boldsymbol{0} \\ \boldsymbol{g} \end{pmatrix} \tag{5.1.17}$$

Here, $\boldsymbol{A}_{IV} = (\boldsymbol{A}_{II}\ \boldsymbol{A}_{IG})$ is the first n_i rows of the discrete bulk Laplacian matrix $\boldsymbol{A}_v$ defined by (5.1.3). $\boldsymbol{A}_u$ is the discrete extension of the shifted surface Laplacian defined by (5.1.12). $\boldsymbol{B} = (\boldsymbol{B}_{GI}\ \boldsymbol{B}_{GG})$, defined by (5.1.9), and $\boldsymbol{B}_{GU}$, defined by (5.1.10), correspond to the disretization of the coupling Robin boundary conditions. $\boldsymbol{B}_{UV} = (\boldsymbol{B}_{UI}\ \boldsymbol{B}_{UG})$, defined by (5.1.15), corresponds to the contribution of the bulk quantity v to the surface quantity u.

5.1.2 Multigrid solvers

For 2-D problems, one can simply use a sparse direct solver such as MATLAB's backslash to solve the linear system (5.1.17), but for 3-D problems iterative solvers are needed. We shall develop a multigrid solver by combining the ideas in Algorithm 2 for surfaces without boundary and Algorithm 3 for codimension 0 domains.

When doing relaxation at each grid level, we alternate between relaxing for v in the bulk and relaxing for u on the surface. More precisely, in each relaxation we first perform one step of weighted Jacobi iteration for non-ghost point values of v and modify ghost point values of v according to the discretized coupled boundary conditions (5.1.11):

$$\boldsymbol{v}_I := \boldsymbol{v}_I + \omega \mathrm{diag}(\boldsymbol{A}_{II})^{-1}(\boldsymbol{f}_I - \boldsymbol{A}_I \boldsymbol{v}), \tag{5.1.18a}$$

$$\boldsymbol{v}_G := -\boldsymbol{B}_{GG}^{-1}(\boldsymbol{B}_{GI}\boldsymbol{v}_I + \boldsymbol{B}_{GU}\boldsymbol{u}). \tag{5.1.18b}$$

We then perform one step of Jacobi iteration and closest point extension for u:

$$\boldsymbol{u} := \boldsymbol{u} + \mathrm{diag}(\boldsymbol{A}_u)^{-1}(\boldsymbol{g} - \boldsymbol{B}_{UV}\boldsymbol{v} - \boldsymbol{A}_u \boldsymbol{u}), \tag{5.1.19a}$$

$$\boldsymbol{u} := \boldsymbol{E}_U \boldsymbol{u}. \tag{5.1.19b}$$

Notice that when doing Jacobi iteration, we use $\boldsymbol{A}_u$ defined by (5.1.12) instead of using $(1+\beta)\boldsymbol{I}_u - \boldsymbol{L}_u$ as we did in Multigrid Algorithm 2 for surfaces without boundary. This is because using $\boldsymbol{A}_u$ is more accurate in practice, especially for 3-D problems. Also, note that when computing the residual in (5.1.19a), we need to subtract an extra term $\boldsymbol{B}_{UV}\boldsymbol{v}$.

Armed with the above relaxation strategy, we obtain the following multigrid algorithm.

Algorithm 5 (V-Cycle Scheme for Coupled Bulk-Surface Poisson Problem) *Denote by* $(\mathcal{B}(\Omega)^h, \mathcal{B}(\mathcal{S})^h)$ *and* $(\mathcal{B}(\Omega)^{2h}, \mathcal{B}(\mathcal{S})^{2h})$ *the fine and next coarser grids for* (v, u)*. Let all variables with superscript* h *be defined on the fine grid, and all variables with superscript* $2h$ *be defined on the coarse grid. Let* $\omega \in (0,1)$ *be the weight parameter of the weighted Jacobi iteration. Let* $\boldsymbol{I}_h^{2h}$ *and* $\boldsymbol{I}_{2h}^h$ *be the restriction and prolongation matrices for* v*. Let* $\boldsymbol{E}_h^{2h}$ *and* $\boldsymbol{E}_{2h}^h$ *be the restriction and prolongation matrices for* u*. We compute* $(\boldsymbol{v}^h, \boldsymbol{u}^h) = \boldsymbol{V}^h(\boldsymbol{v}^h, \boldsymbol{u}^h, \boldsymbol{f}^h, \boldsymbol{g}^h)$ *by recursively calling:*

1. *Starting from some initial guess* $(\boldsymbol{v}^h, \boldsymbol{u}^h)$ *(e.g.* $\boldsymbol{v}^h = \boldsymbol{0}^h$ *and* $\boldsymbol{u}^h = \boldsymbol{0}^h$*) on the finest grid, perform* ν_1 *steps of relaxation for* v *and* u*,*

$$
\begin{aligned}
&for\ i = 1 : \nu_1,\ do \\
&\qquad \boldsymbol{v}_I^h &:=\ & \boldsymbol{v}_I^h + \omega\, diag(\boldsymbol{A}_{II}^h)^{-1}(\boldsymbol{f}_I^h - \boldsymbol{A}_I^h \boldsymbol{u}^h), \\
&\qquad \boldsymbol{v}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}(\boldsymbol{B}_{GI}^h \boldsymbol{v}_I^h + \boldsymbol{B}_{GU}^h \boldsymbol{u}^h); \\
&\qquad \boldsymbol{u}^h &:=\ & \boldsymbol{u}^h + diag(\boldsymbol{A}_u^h)^{-1}(\boldsymbol{g}^h - \boldsymbol{B}_{UV}^h \boldsymbol{v}^h - \boldsymbol{A}_u^h \boldsymbol{u}^h), \\
&\qquad \boldsymbol{u}^h &:=\ & \boldsymbol{E}_U^h \boldsymbol{u}^h.
\end{aligned}
$$

2. *Compute the residuals and restrict them to the coarser grid,*

$$
\begin{aligned}
\boldsymbol{r}_I^h &:=\ \boldsymbol{f}_I^h - \boldsymbol{A}_I^h \boldsymbol{u}^h, \\
\boldsymbol{f}_I^{2h} &:=\ \boldsymbol{I}_h^{2h} \boldsymbol{r}_I^h; \\
\boldsymbol{r}_u^h &:=\ \boldsymbol{g}^h - \boldsymbol{B}_{UV}^h \boldsymbol{v}^h - \boldsymbol{A}_u^h \boldsymbol{u}^h, \\
\boldsymbol{g}^{2h} &:=\ \boldsymbol{E}_h^{2h} \boldsymbol{r}_u^h.
\end{aligned}
$$

   If $\left(\mathcal{B}(\Omega)^{2h}, \mathcal{B}(\mathcal{S})^{2h}\right)$ *are the coarsest grids, solve (5.1.17) directly; else*

$$
\begin{aligned}
\boldsymbol{v}^{2h} &:=\ \boldsymbol{0}^{2h}, \\
\boldsymbol{u}^{2h} &:=\ \boldsymbol{0}^{2h}, \\
(\boldsymbol{v}^{2h}, \boldsymbol{u}^{2h}) &:=\ V^{2h}(\boldsymbol{v}^{2h}, \boldsymbol{u}^{2h}, \boldsymbol{f}^{2h}, \boldsymbol{g}^{2h}).
\end{aligned}
$$

3. *Correct the values on the fine grid using coarse grid values,*

$$
\begin{aligned}
\boldsymbol{u}^h &:=\ \boldsymbol{u}^h + \boldsymbol{E}_{2h}^h \boldsymbol{u}^{2h}, \\
\boldsymbol{u}^h &:=\ \boldsymbol{E}_U^h \boldsymbol{u}^h; \\
\boldsymbol{v}_I^h &:=\ \boldsymbol{v}_I^h + \boldsymbol{I}_{2h}^h \boldsymbol{v}^{2h}, \\
\boldsymbol{v}_G^h &:=\ -(\boldsymbol{B}_{GG}^h)^{-1}(\boldsymbol{B}_{GI}^h \boldsymbol{v}_I^h + \boldsymbol{B}_{GU}^h \boldsymbol{u}^h).
\end{aligned}
$$

4. Perform ν_2 steps of relaxation for v and u,

$$
\begin{aligned}
&\textit{for } i = 1 : \nu_2, \textit{ do} \\
&\qquad \boldsymbol{v}_I^h &:=\ & \boldsymbol{v}_I^h + \omega \mathit{diag}(\boldsymbol{A}_{II}^h)^{-1}(\boldsymbol{f}_I^h - \boldsymbol{A}_I^h \boldsymbol{u}^h), \\
&\qquad \boldsymbol{v}_G^h &:=\ & -(\boldsymbol{B}_{GG}^h)^{-1}(\boldsymbol{B}_{GI}^h \boldsymbol{v}_I^h + \boldsymbol{B}_{GU}^h \boldsymbol{u}^h); \\
&\qquad \boldsymbol{u}^h &:=\ & \boldsymbol{u}^h + \mathit{diag}(\boldsymbol{A}_u^h)^{-1}(\boldsymbol{g}^h - \boldsymbol{B}_{UV}^h \boldsymbol{v}^h - \boldsymbol{A}_u^h \boldsymbol{u}^h), \\
&\qquad \boldsymbol{u}^h &:=\ & \boldsymbol{E}_U^h \boldsymbol{u}^h.
\end{aligned}
$$

5.1.3 Numerical examples

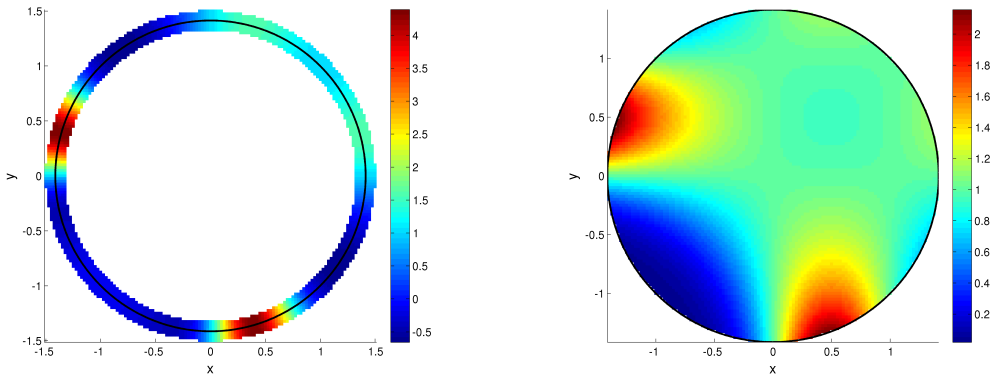
For simplicity, we set $\alpha = \beta = 1$ in all the following numerical examples. In order to obtain 2nd-order convergence results, we use cubic interpolation in both (5.1.8) when interpolating v on the surface and (5.1.13) when interpolating u on the surface. It is typically enough to set the interpolation degree q of $\boldsymbol{E}_Y^q$ in (5.1.9) to be 2, but using cubic interpolation for $\boldsymbol{E}_Y^q$ seems to give smaller numerical errors. We shall set $q = 3$ for 2-D problems to gain extra accuracy but $q = 2$ for 3-D problems to save computational cost.

5.1.3.1 Disk

We consider problem (5.1.1) for a disk centered at the origin with radius $\sqrt{2}$. The exact solution in the disk is chosen to be $v(x, y) = \exp(-x(x-1)y(y-1))$. We calculate the right hand side function of v by forming $f = -\Delta v + v$. We then set up the exact solution u on the circle by computing $u = \frac{1}{\beta}(\alpha v + \nabla v \cdot \boldsymbol{n})|_{\mathcal{S}}$, where $\boldsymbol{n} = (x/(x^2+y^2)^{1/2}, y/(x^2+y^2)^{1/2})$. Finally, we calculate the right hand side function of the equation for u to be $g = -\Delta_{\mathcal{S}} u + (1+\beta)u - \alpha v|_{\mathcal{S}}$. Here we compute $\Delta_{\mathcal{S}}$ by $\nabla_{\mathcal{S}} \cdot \boldsymbol{w}$, where $\boldsymbol{w} = \nabla_{\mathcal{S}} u = \nabla u - (\nabla u \cdot \boldsymbol{n})$, and $\nabla_{\mathcal{S}} \cdot \boldsymbol{w} = \nabla \cdot \boldsymbol{w} - \sum_{j=1}^{2} (\nabla \boldsymbol{w}_j \cdot \boldsymbol{n})\boldsymbol{n}_j$.

Figure 5.1 shows the numerical solutions with $\Delta x = 0.025$. Table 5.1 records the convergence results of the solutions solved with MATLAB's backslash for the coupled disk-circle Poisson's equation. We observe 2nd-order convergence results for both u and v.

We also test the Multigrid Algorithm 5 applied to the above problem. We use cubic interpolations to construct $\boldsymbol{E}_U^h$'s on each grid level. In each round of the V-Cycle, $\nu_1 = 3$ pre-smoothings and $\nu_2 = 3$ post-smoothings are applied. We choose $\omega = 0.8$ when doing weighted Jacobi iteration for v. For the restriction and prolongation operators, linear interpolations are used for both u and v. The linear system on the coarsest grid level is solved with Matlab's backslash. The stopping criterion is



(a) Numerical solution of u (b) Numerical solution of v

Figure 5.1: Numerical solutions for the coupled disk-circle Poisson problem with $\Delta x = 0.025$.

Table 5.1: Convergence study for the coupled disk-circle Poisson's equation solved with MATLAB's backslash.

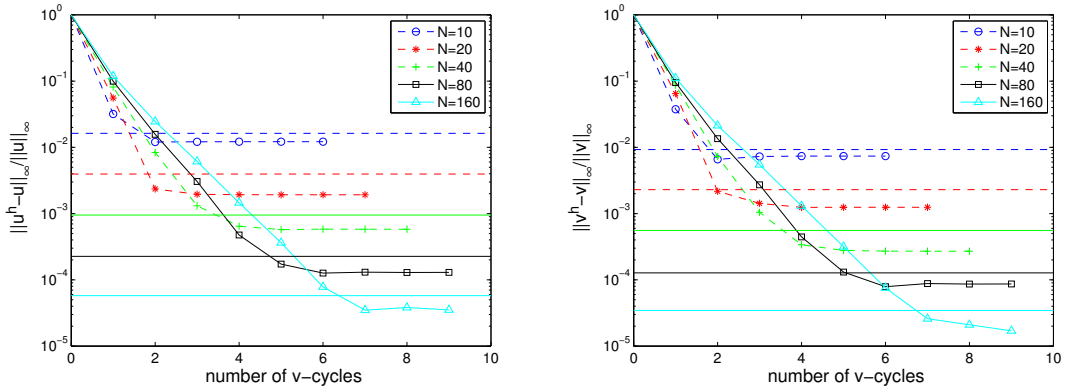
(a) Relative errors on the circle

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.63e-2	3.95e-3	9.51e-4	2.26e-4	5.77e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.04	2.05	2.07	1.97

(b) Relative errors in the disk

Δx	0.1	0.05	0.025	0.0125	0.00625
$\frac{\|v-v^h\|_\infty}{\|v\|_\infty}$	9.26e-3	2.30e-3	5.57e-4	1.28e-4	3.44e-5
$\log_2 \frac{\|v-v^{2h}\|_\infty}{\|v-v^h\|_\infty}$		2.01	2.05	2.13	1.89

$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\|_\infty / \|\boldsymbol{x}^k\|_\infty < 10^{-6}$, where $\boldsymbol{x}^k$ and $\boldsymbol{x}^{k+1}$ are the iterative solutions after the k-th and $(k+1)$-st rounds of V-Cycles. Figure 5.2 shows the V-Cycle convergence results for u and v; the relative errors of both u and v decrease in a desirable manner similar to what we observed in the previous chapter.



(a) Decrease of relative errors on the circle (b) Decrease of relative errors in the disk

Figure 5.2: V-Cycle convergence results for the coupled disk-circle Poisson problem.

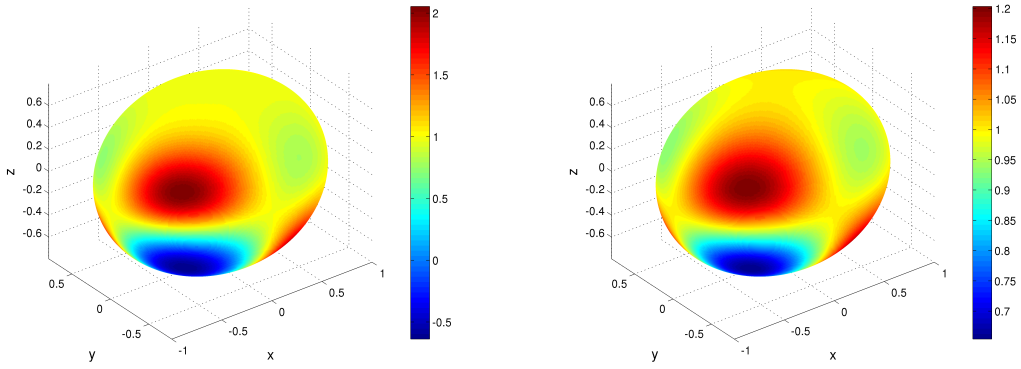
5.1.3.2 Ellipsoid

We now consider problem (5.1.1) in a 3-D bulk surrounded by an ellipsoid $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$, where $a = 1$, $b = c = 0.8$. The exact solution in the bulk is chosen to be $v(x, y, z) = \exp(-x(x-1)y(y-1)z(z-1))$. We calculate the right hand side function of v by forming $f = -\Delta v + v$. We then set up the exact solution u on the circle by computing $u = \frac{1}{\beta}(\alpha v + \nabla v \cdot \boldsymbol{n})|_{\mathcal{S}}$, where

$$\boldsymbol{n} = \left(\frac{x}{a^2\sqrt{x^2/a^4 + y^2/b^4 + z^2/c^4}}, \frac{y}{b^2\sqrt{x^2/a^4 + y^2/b^4 + z^2/c^4}}, \frac{z}{c^2\sqrt{x^2/a^4 + y^2/b^4 + z^2/c^4}} \right).$$

Finally we calculate the right hand side function of the equation for u to be $g = -\Delta_{\mathcal{S}} u + (1+\beta)u - \alpha v|_{\mathcal{S}}$. Here we compute $\Delta_{\mathcal{S}}$ by $\nabla_{\mathcal{S}} \cdot \boldsymbol{w}$, where $\boldsymbol{w} = \nabla_{\mathcal{S}} u = \nabla u - (\nabla u \cdot \boldsymbol{n})$, and $\nabla_{\mathcal{S}} \cdot \boldsymbol{w} = \nabla \cdot \boldsymbol{w} - \sum_{j=1}^{3} (\nabla \boldsymbol{w}_j \cdot \boldsymbol{n}) \boldsymbol{n}_j$.

We test the Multigrid Algorithm 5 applied to the coupled bulk-surface Poisson's equations for the ellipsoid. Figure 5.3 shows the numerical solution of u and v restricted on the surface with $\Delta x = 0.1$. Table 5.2 records the numerical errors of the multigrid algorithm, and it roughly shows 2nd-order convergence results. Figure 5.4 shows V-Cycle convergence results for u and v. Again the decrease of relative errors for both u and v are well behaved.



(a) u (b) v restricted on the surface

Figure 5.3: Numerical solutions of u and v for the coupled bulk-surface Poisson problem for an ellipsoid with $\Delta x = 0.1$.

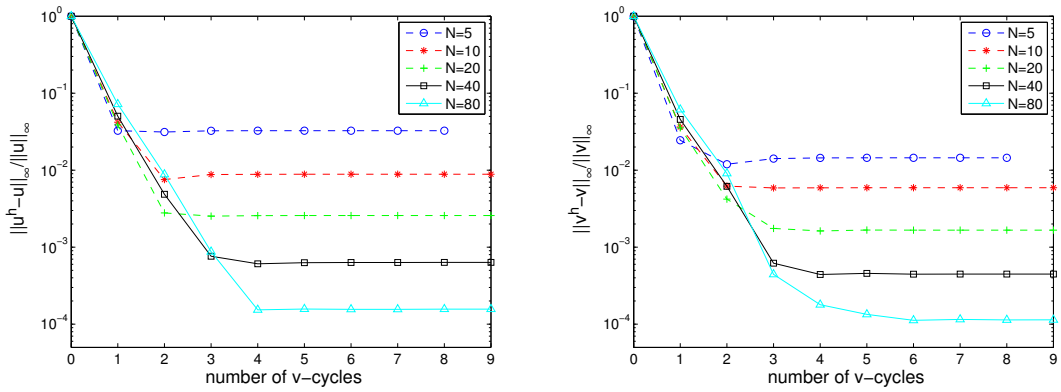
Table 5.2: Convergence study for the coupled bulk-surface Poisson's equation for an ellipsoid solved by the multigrid method.

(a) Relative errors on the surface

Δx	0.2	0.1	0.05	0.025	0.0125
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	3.25e-2	8.85e-3	2.58e-3	6.34e-4	1.57e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.88	1.78	2.02	2.02

(b) Relative errors in the bulk

Δx	0.2	0.1	0.05	0.025	0.0125
$\frac{\|v-v^h\|_\infty}{\|v\|_\infty}$	1.45e-2	5.92e-3	1.66e-3	4.47e-4	1.14e-4
$\log_2 \frac{\|v-v^{2h}\|_\infty}{\|v-v^h\|_\infty}$		1.29	1.83	1.90	1.97



(a) Decrease of relative errors on the surface (b) Decrease of relative errors in the bulk

Figure 5.4: V-Cycle convergence results for the coupled bulk-surface Poisson problem for an ellipsoid.

5.2 Coupled diffusion equations

Having investigated coupled bulk-surface elliptic problems, it is natural for us to add time derivatives and consider the following coupled diffusion equations with source terms,

$$v_t = \Delta v + f \quad \text{in } \Omega, \tag{5.2.1a}$$

$$\frac{\partial v}{\partial \boldsymbol{n}} = \beta u - \alpha v \quad \text{on } \mathcal{S}, \tag{5.2.1b}$$

$$u_t = \Delta_{\mathcal{S}} u - \frac{\partial v}{\partial \boldsymbol{n}} + g \quad \text{on } \mathcal{S}. \tag{5.2.1c}$$

Again we can replace $\frac{\partial v}{\partial \boldsymbol{n}}$ by $\beta u - \alpha v$ in (5.2.1c), and obtain a surface PDE equivalent to (5.2.1c),

$$u_t = \Delta_{\mathcal{S}} u - \beta u + \alpha v + g \quad \text{on } \mathcal{S}. \tag{5.2.2}$$

Similarly to Section 5.1, we shall discretize (5.2.2) instead of (5.2.1c).

System (5.2.1) without source terms f and g was investigated by Novak et al. in [68]. We refer readers to [68] for a finite volume approach to solving system (5.2.1) and its applications to cell biology. In this thesis, we focus on the numerical solutions for (5.2.1) using the spatial discretizations in Section 5.1.1.

5.2.1 Alternating explicit time stepping

In Section 5.1.1, we have investigated the spatial discretizations, so we only need to add time discretization to numerically solve the coupled bulk-surface diffusion

problem (5.2.1). As a first attempt, we alternately update v and u using forward Euler time stepping. Starting from some initial v and u, at each time step, we first evolve v at the non-ghost points according to (5.2.1a), then update v at the ghost points so that the coupled boundary condition (5.2.1b) is (approximately) satisfied, and finally evolve u in the narrow band according to (5.2.1c).

More precisely, we construct a matrix formulation of the above procedure. Let $\boldsymbol{L}_v$ be the Laplacian matrix for v. Let $\boldsymbol{L}_{IV} = (\boldsymbol{L}_{II}\ \boldsymbol{L}_{IG})$ be the first n_i rows of $\boldsymbol{L}_v$ corresponding to the non-ghost points for v. Suppose $\boldsymbol{E}_U^1$, $\boldsymbol{E}_U^3$, $\boldsymbol{L}_u$ and $\boldsymbol{I}_u$ are as specified in (5.1.12). We let

$$\boldsymbol{M}_u = \left(\boldsymbol{E}_U^1 \boldsymbol{L}_u - \frac{2d}{\Delta x^2}(\boldsymbol{I}_u - \boldsymbol{E}_U^3)\right) - \beta \boldsymbol{I}_u. \tag{5.2.3}$$

Let $\boldsymbol{B} = (\boldsymbol{B}_{GI}\ \boldsymbol{B}_{GG})$ be defined by (5.1.9), $\boldsymbol{B}_{GU}$ defined by (5.1.10), and $\boldsymbol{B}_{UV} = (\boldsymbol{B}_{UI}\ \boldsymbol{B}_{UG})$ defined by (5.1.15). Suppose that we want to evolve (5.2.1) from time 0 to T. First, we divide $[0, T]$ uniformly into N subintervals with time step Δt. Let $\boldsymbol{v}^n = (\boldsymbol{v}_I^n, \boldsymbol{v}_G^n)^T$ and $\boldsymbol{u}^n$ be the numerical solution at time $n\Delta t$. Let $\boldsymbol{f}_I^n$ be the source function f evaluated at the non-ghost points of v and $\boldsymbol{g}^n$ be the source function g evaluated at the closest points of grid points of u at time $n\Delta t$.

For $n = 0, 1, \ldots, N-1$, we perform the following three steps:

$$\boldsymbol{v}_I^{n+1} = \boldsymbol{v}_I^n + \Delta t(\boldsymbol{L}_{IV}\boldsymbol{v}^n + \boldsymbol{f}_I^n), \tag{5.2.4a}$$

$$\boldsymbol{v}_G^{n+1} = -\boldsymbol{B}_{GG}^{-1}(\boldsymbol{B}_{GI}\boldsymbol{v}_I^{n+1} + \boldsymbol{B}_{GU}\boldsymbol{u}^n), \tag{5.2.4b}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t(\boldsymbol{M}_u\boldsymbol{u}^n + \boldsymbol{B}_{UV}\boldsymbol{v}^{n+1} + \boldsymbol{g}^n). \tag{5.2.4c}$$

We can also first evolve u and then evolve v at each time step, which yields similar results to (5.2.4). Although (5.2.4) is only first order accurate in time, we choose $\Delta t = O(\Delta x^2)$ for stability reasons. Therefore, (5.2.4) is 2nd-order with respect to Δx in practice as we will see in Section 5.2.3.

5.2.2 Method-of-lines time stepping

We need to choose $\Delta t = O(\Delta x^2)$ to ensure the stability of an explicit scheme because of the diffusion operators in problem (5.2.1). It is desirable for us to construct implicit schemes to increase the time step Δt from $O(\Delta x^2)$ to $O(\Delta x)$. Although the scheme (5.2.4) is easy to understand and implement, modifying it into an implicit scheme is not straightforward. Instead, in this section we investigate a method-of-lines (MOLs) approach [80] to solve (5.2.1). In other words, we first discretize the coupled problem

(5.2.1) spatially to obtain a system of ODEs, and then apply standard ODE solvers to the semi-discretized problem along the time direction.

Notice that we can represent $\boldsymbol{v}_G$ by $\boldsymbol{v}_I$ and $\boldsymbol{u}$ according to the discretized coupled boundary condition (5.1.16):

$$\boldsymbol{v}_G = -\boldsymbol{B}_{GG}^{-1}(\boldsymbol{B}_{GI}\boldsymbol{v}_I + \boldsymbol{B}_{GU}\boldsymbol{u}). \tag{5.2.5}$$

Thus we can set up a new solution vector

$$\boldsymbol{s} = \begin{pmatrix} \boldsymbol{v}_I \\ \boldsymbol{u} \end{pmatrix},$$

and construct an operator $\boldsymbol{A}$ so that $\boldsymbol{As}$ approximates the diffusion operators of v and u. Using the same notations as in Section 5.2.1, and eliminating $\boldsymbol{v}_G$ using (5.2.5), $d_v\Delta v$ at non-ghost points of the bulk can be approximated as

$$\boldsymbol{L}_{IV}\boldsymbol{v} = (\boldsymbol{L}_{II} - \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI})\boldsymbol{v}_I - \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GU}\boldsymbol{u}. \tag{5.2.6}$$

$d_u\Delta_{\mathcal{S}} - \beta u + \alpha v$ in the band surrounding the surface can be approximated as

$$\boldsymbol{B}_{UV}\boldsymbol{v} + \boldsymbol{M}_u\boldsymbol{u} = (\boldsymbol{B}_{UI} - \boldsymbol{B}_{UG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI})\boldsymbol{v}_I + (\boldsymbol{M}_u - \boldsymbol{B}_{UG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GU})\boldsymbol{u}. \tag{5.2.7}$$

Combining (5.2.6) and (5.2.7), we obtain that

$$\boldsymbol{As} = \begin{pmatrix} \boldsymbol{L}_{IV}\boldsymbol{v} \\ \boldsymbol{B}_{UV}\boldsymbol{v} + \boldsymbol{M}_u\boldsymbol{u} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}_{II} - \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI} & \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GU} \\ \boldsymbol{B}_{UI} - \boldsymbol{B}_{UG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI} & \boldsymbol{M}_u - \boldsymbol{B}_{UG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GU} \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_I \\ \boldsymbol{u} \end{pmatrix}.$$

Therefore, the semi-discretized ODE system becomes

$$\boldsymbol{s}_t = \boldsymbol{As} + \boldsymbol{h}, \tag{5.2.8}$$

where $\boldsymbol{s} = \begin{pmatrix} \boldsymbol{v}_I \\ \boldsymbol{u} \end{pmatrix}$, $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{L}_{II} - \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI} & \boldsymbol{L}_{IG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GU} \\ \boldsymbol{B}_{UI} - \boldsymbol{B}_{UG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GI} & \boldsymbol{M}_u - \boldsymbol{B}_{UG}\boldsymbol{B}_{GG}^{-1}\boldsymbol{B}_{GU} \end{pmatrix}$, and $\boldsymbol{h}$ is the source term. $\boldsymbol{h}$ might depend on t, $\boldsymbol{x}$ and $\boldsymbol{s}$ linearly or nonlinearly.

One can then choose his/her favourite time stepping methods to solve the ODE system (5.2.8). The simplest scheme is still the forward Euler time stepping method,

$$\boldsymbol{s}^{n+1} = \boldsymbol{s}^n + \Delta t(\boldsymbol{As}^n + \boldsymbol{h}^n). \tag{5.2.9}$$

It might cost too much memory to directly construct $\boldsymbol{A}$ in (5.2.8) for 3-D problems, because there are several matrix inversions involved in the construction of $\boldsymbol{A}$. Therefore, we construct a scheme which does not need to construct $\boldsymbol{A}$ explicitly but is equivalent to scheme (5.2.9):

$$\boldsymbol{v}_G^n = -\boldsymbol{B}_{GG}^{-1}(\boldsymbol{B}_{GI}\boldsymbol{v}_I^n + \boldsymbol{B}_{GU}\boldsymbol{u}^n), \tag{5.2.10a}$$

$$\boldsymbol{v}_I^{n+1} = \boldsymbol{v}_I^n + \Delta t(\boldsymbol{L}_{IV}\boldsymbol{v}^n + \boldsymbol{f}_I^n), \tag{5.2.10b}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t(\boldsymbol{M}_u\boldsymbol{u}^n + \boldsymbol{B}_{UV}\boldsymbol{v}^n + \boldsymbol{g}^n). \tag{5.2.10c}$$

In order to choose $\Delta t = O(\Delta x)$, we need to handle the linear operator $\boldsymbol{A}$ implicitly. In this thesis we shall use the IMEX linear multistep schemes from [3]. A first order method is the IMEX Euler scheme:

$$(\boldsymbol{I} - \Delta t \boldsymbol{A})\boldsymbol{s}^{n+1} = \boldsymbol{s}^n + \Delta t \boldsymbol{h}^n. \tag{5.2.11}$$

A second order method is the semi-implicit backward difference formula (SBDF-2):

$$(\boldsymbol{I} - \frac{2}{3}\Delta t \boldsymbol{A})\boldsymbol{s}^{n+1} = \frac{4}{3}\boldsymbol{s}^n - \frac{1}{3}\boldsymbol{s}^{n-1} + \frac{4\Delta t}{3}\boldsymbol{h}^n - \frac{2\Delta t}{3}\boldsymbol{h}^{n-1}. \tag{5.2.12}$$

We need to solve large linear systems at each time step if IMEX schemes are used. For 2-D problems sparse direct solvers can be used, but for large 3-D problems, efficient iterative solvers and preconditioners are needed. One possibility is to construct a multigrid algorithm using the ideas in Algorithm 5, and this is left to future work.

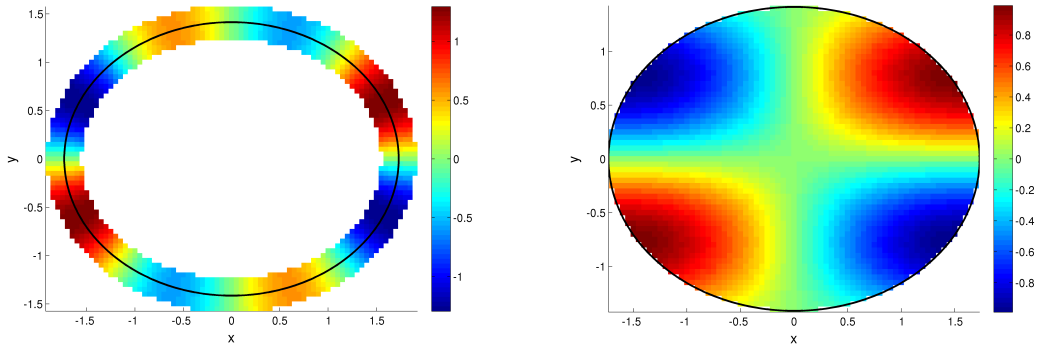
5.2.3 Numerical examples

5.2.3.1 Ellipse

For simplicity, we choose $\alpha = \beta = 1$ in equation (5.2.1). The boundary is an ellipse $\frac{x^2}{3} + \frac{y^2}{2} = 1$. The exact solution v in the interior of the ellipse is chosen as $e^{-5t}\sin(x)\sin(2y)$. Since this exact solution v satisfies $v_t = \Delta v$, the source function f of v is set to be 0. We then set the exact solution $u = v + \frac{\partial v}{\partial \boldsymbol{n}}$ according to the coupled condition (5.2.1b) on the boundary. Finally we calculate the source function g of u according to $g = u_t - \Delta_{\mathcal{S}} u + u - v$.

We first test the alternating forward Euler time-stepping scheme (5.2.4) applied to the above problem. We choose $\Delta t = \frac{1}{4}\Delta x^2$, and evolve to $T = 0.5$. Figure 5.5 shows initial u and v, and Figure 5.6 shows numerical solutions for u and v at $T = 0.5$. As time evolves, the patterns of u and v do not change while their magnitudes decay by about a factor of 10. This is because the initial v is an eigenfunction of the Laplacian operator in the bulk and the initial u is an eigenfunction of the Laplace-Beltrami operator on the ellipse. Both u and v decay with time and their magnitudes are about e^{-5t}. Table 5.3 records the relatives errors of u on the ellipse and v in the interior. We can observe 2nd-order convergence for both u and v.

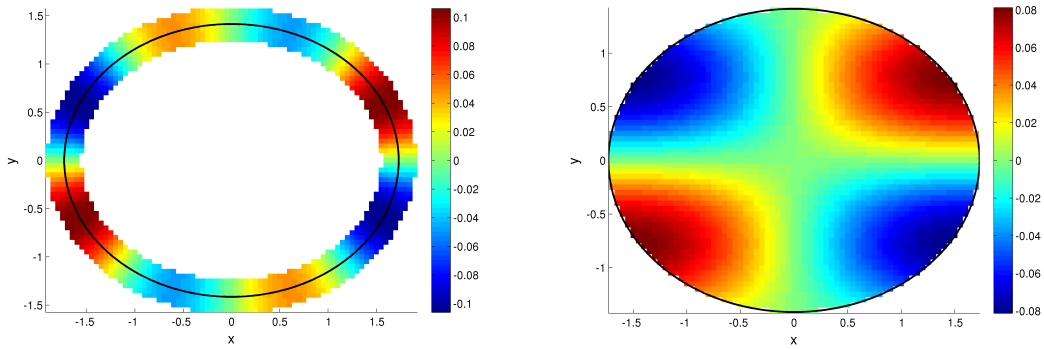
We then test the forward Euler scheme (5.2.9) applied to (5.2.8), which is a MOL approach. We choose $\Delta t = \frac{1}{4}\Delta x^2$. Table 5.4 records the numerical errors. These errors are similar to those in Table 5.3 where alternating forward Euler scheme (5.2.4) is used.



(a) u at time 0

(b) v at time 0

Figure 5.5: Initial u and v for the coupled bulk-surface diffusion equation where the "surface" is an ellipse and the bulk is the domain surrounded by the ellipse. $\Delta x = 0.05$.



(a) u at time 0.5

(b) v at time 0.5

Figure 5.6: u and v at time $T = 0.5$ for the coupled bulk-surface diffusion equation where the "surface" is an ellipse and the bulk is the domain surrounded by the ellipse. $\Delta x = 0.05$.

Table 5.3: Convergence study for the coupled bulk-surface diffusion equation where the "surface" is an ellipse and the bulk is the domain surrounded by the ellipse. Alternating forward Euler stepping scheme (5.2.4) is used and $\Delta t = \frac{1}{4}\Delta x^2$.

(a) Relative errors on the ellipse

Δx	0.4	0.2	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.18e-1	3.48e-2	7.59e-3	2.09e-3	5.08e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.76	2.19	1.86	2.04

(b) Relative errors in the bulk

Δx	0.4	0.2	0.1	0.05	0.025
$\frac{\|v-v^h\|_\infty}{\|v\|_\infty}$	5.00e-2	1.51e-2	4.13e-3	1.14e-3	3.00e-4
$\log_2 \frac{\|v-v^{2h}\|_\infty}{\|v-v^h\|_\infty}$		1.73	1.87	1.86	1.93

Table 5.4: Convergence study for the coupled bulk-surface diffusion equation where the "surface" is an ellipse and the bulk is the domain surrounded by the ellipse. Forward Euler MOL scheme (5.2.9) is used and $\Delta t = \frac{1}{4}\Delta x^2$.

(a) Relative errors on the ellipse

Δx	0.4	0.2	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.51e-1	3.72e-2	9.43e-3	2.30e-3	5.79e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.02	1.98	2.04	1.99

(b) Relative errors in the bulk

Δx	0.4	0.2	0.1	0.05	0.025
$\frac{\|v-v^h\|_\infty}{\|v\|_\infty}$	7.60e-2	2.00e-2	4.59e-3	1.52e-3	3.65e-4
$\log_2 \frac{\|v-v^{2h}\|_\infty}{\|v-v^h\|_\infty}$		1.92	2.12	1.59	2.06

Finally we test SBDF-2 scheme (5.2.12) applied to (5.2.8). We choose $\Delta t = \frac{1}{4}\Delta x$. In the first time step IMEX-Euler scheme (5.2.11) is used. Table 5.5 shows the numerical errors. Again we observe 2nd-order convergence results for Δx. Since $\Delta t = O(\Delta x)$, the results indicate that the SBDF-2 scheme is also 2nd-order in time.

Table 5.5: Convergence study for the coupled bulk-surface diffusion equation where the "surface" is an ellipse and the bulk is the domain surrounded by the ellipse. SBDF-2 scheme (5.2.12) is used and $\Delta t = \frac{1}{4}\Delta x$.

(a) Relative errors on the ellipse

Δx	0.4	0.2	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.10e-1	2.84e-2	7.15e-3	1.64e-3	4.12e-4
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		1.96	1.99	2.12	2.00

(b) Relative errors in the bulk

Δx	0.4	0.2	0.1	0.05	0.025
$\frac{\|v-v^h\|_\infty}{\|v\|_\infty}$	8.46e-2	2.06e-2	4.66e-3	9.78e-4	2.37e-4
$\log_2 \frac{\|v-v^{2h}\|_\infty}{\|v-v^h\|_\infty}$		2.04	2.15	2.25	2.05

5.2.3.2 Ball-sphere

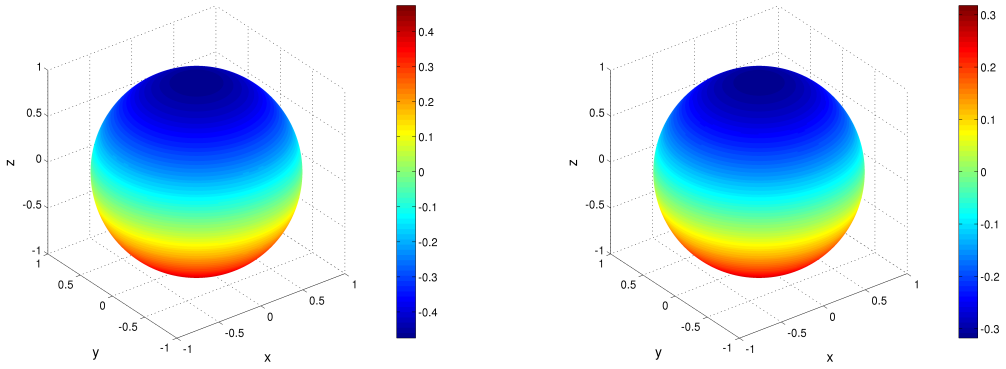
We consider the coupled bulk-surface diffusion problem (5.2.1) with diffusion in the unit ball coupled with diffusion on the unit sphere. This example was tested by the authors of [68]. In their test, $\alpha = \beta = 1$ and the source functions $f = g = 0$. With the spherical coordinates (θ, ϕ, r) ($x = r\sin\phi\cos\theta$, $y = r\sin\phi\sin\theta$, $z = r\cos\phi$), the exact solutions are chosen to be

$$\begin{aligned} u &= Ae^{-k^2 t}\cos(\phi), \\ v &= e^{-k^2 t}\,\frac{\sin(kr) - kr\cos(kr)}{(kr)^2}\cos(\phi), \end{aligned} \tag{5.2.13}$$

where $k = 1.527338738$, and $A = \sin(k) + \frac{\cos(k)}{k} - \frac{\sin(k)}{k^2}$ ([68]). We choose $\Delta t = \frac{1}{8}\Delta x^2$ and evolve the scheme (5.2.4) to $T = 0.1$.

Figure 5.7 shows the numerical solutions of u and v on the surface at time 0.1 with $\Delta x = 0.2$. Table 5.6 shows the convergence results of the alternating forward

Euler time stepping scheme (5.2.4). Table 5.7 shows the convergence results of the MOL foward Euler time stepping scheme (5.2.9) with implementation (5.2.10).



(a) u at time 0.1 (b) v restricted on the surface at time 0.1

Figure 5.7: Numerical solutions of u and v at time $T = 0.1$ for the ball-sphere coupled diffusion equation with $\Delta x = 0.2$.

5.3 Fisher-KPP equation in the bulk coupled with diffusion on the curve

In 2013, Berestycki, Roquejoffre and Rossi proposed a model in [7] to describe biological invasions in the plane coupled with fast diffusion on a line. Propagations of biological species can be modeled by the Fisher-KPP equation [31, 47]. As pointed out in [7], the usual Fisher-KPP equation has a spreading speed [2, 5] describing how fast the species propagate asymptotically. The work [7] investigates the spreading speed of the Fisher-KPP equation on a two dimensional domain influenced by a coupled diffusion equation on a line in the plane. The model equation proposed in [7] is

$$
\begin{aligned}
v_t &= d_v \Delta v + v(1-v) \quad \text{in } \Omega, && \text{(5.3.1a)}\\
-d_v \frac{\partial v}{\partial y} &= \beta u - \alpha v \quad \text{on } \mathbb{R}, && \text{(5.3.1b)}\\
u_t &= d_u u_{xx} - \beta u + \alpha v \quad \text{on } \mathbb{R}, && \text{(5.3.1c)}
\end{aligned}
$$

where the domain $\Omega = \{(x, y) : x \in \mathbb{R},\ y > 0\}$ is the upper half plane and $\mathbb{R}$ is the real line. Roughly speaking, the key result of [7] is that when the ratio between the diffusion coefficients $\frac{d_u}{d_v}$ exceeds some critical value, the propagation speed of the

Table 5.6: Convergence study for the coupled ball-sphere diffusion equation. Scheme (5.2.4) is used and $\Delta t = \frac{1}{8}\Delta x^2$.

(a) Relative errors on the sphere

Δx	0.2	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.53e-3	3.37e-4	8.05e-5	1.98e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.18	2.07	2.02

(b) Relative errors in the ball

Δx	0.2	0.1	0.05	0.025
$\frac{\|v-v^h\|_\infty}{\|v\|_\infty}$	4.17e-3	9.87e-4	2.46e-4	6.28e-5
$\log_2 \frac{\|v-v^{2h}\|_\infty}{\|v-v^h\|_\infty}$		2.08	2.00	1.97

Table 5.7: Convergence study for the coupled ball-sphere diffusion equation. Scheme (5.2.9) with implementation (5.2.10) is used and $\Delta t = \frac{1}{8}\Delta x^2$.

(a) Relative errors on the sphere

Δx	0.2	0.1	0.05	0.025
$\frac{\|u-u^h\|_\infty}{\|u\|_\infty}$	1.14e-3	2.25e-4	5.23e-5	1.28e-5
$\log_2 \frac{\|u-u^{2h}\|_\infty}{\|u-u^h\|_\infty}$		2.34	2.10	2.03

(b) Relative errors in the ball

Δx	0.2	0.1	0.05	0.025
$\frac{\|v-v^h\|_\infty}{\|v\|_\infty}$	2.51e-3	1.01e-3	2.92e-4	7.18e-5
$\log_2 \frac{\|v-v^{2h}\|_\infty}{\|v-v^h\|_\infty}$		1.31	1.80	2.02

Fisher-KPP equation is increased. This result explains several biological phenomena in reality. To state one example from [7], in the middle of the 14th century, the spread of "Black death" plague was enhanced by relatively faster spread on the Silk Road and commercial roads in Europe. For detailed biological applications and theoretical analysis, we refer readers to [7] and the references therein.

In this thesis, we are interested in the case where Ω is a general bounded domain with a curved boundary $\partial\Omega$ and the coupled diffusion happens on $\partial\Omega = \mathcal{S}$. If we replace $-\frac{\partial v}{\partial y}$ by $\frac{\partial v}{\partial \boldsymbol{n}}$ in (5.3.1b), and replace u_{xx} by $\Delta_{\mathcal{S}} u$ in (5.3.1c), then we obtain the following system,

$$
\begin{aligned}
v_t &= d_v \Delta v + v(1-v) \quad \text{in } \Omega, && \text{(5.3.2a)}\\
d_v \frac{\partial v}{\partial \boldsymbol{n}} &= \beta u - \alpha v \quad \text{on } \mathcal{S}, && \text{(5.3.2b)}\\
u_t &= d_u \Delta_{\mathcal{S}} u - \beta u + \alpha v \quad \text{on } \mathcal{S}. && \text{(5.3.2c)}
\end{aligned}
$$

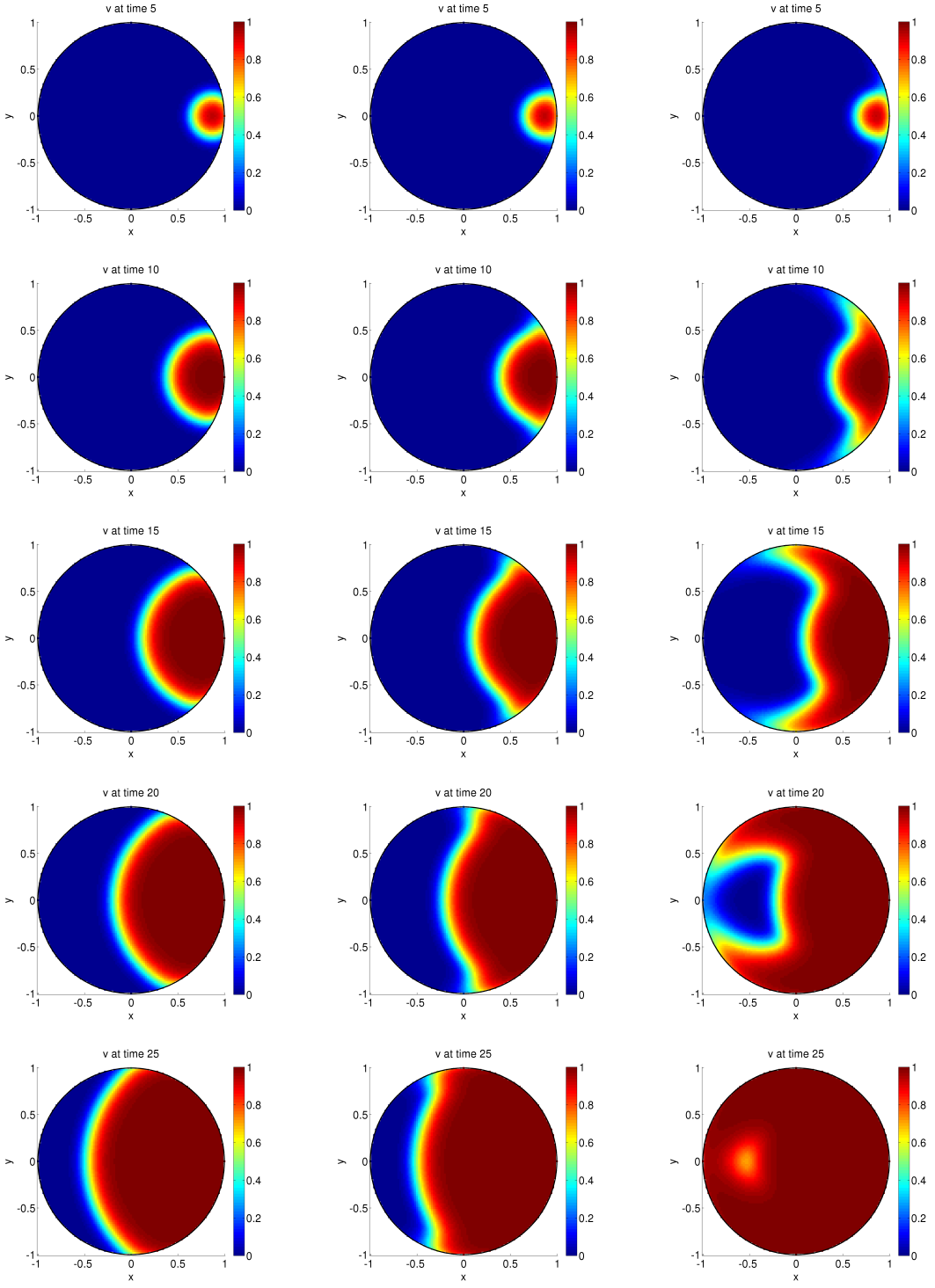
The system (5.3.2) is also studied by Dawson [17] in a project supervised by the author and Colin Macdonald. Equation (5.3.2) is actually a special case of the coupled diffusion system (5.2.1) with the source function in v being $v(1-v)$. Therefore, we can adopt scheme (5.2.4) to solve system (5.3.2). A minor modification is that the diffusion coefficients d_u and d_v need to be considered when building matrices.

We investigate the example where Ω is a unit disk and $\partial\Omega = \mathcal{S}$ is a unit circle. We choose $\alpha = 0.1$ and $\beta = 5$. The initial condition is set to be $u_0 = 0$ and

$$
v_0(x,y) = \begin{cases} \left(1 - (x-\frac{7}{8})^2 - y^2\right)^2, & \sqrt{(x-\frac{7}{8})^2 + y^2} \le \frac{1}{16} \\ 0, & \sqrt{(x-\frac{7}{8})^2 + y^2} > \frac{1}{16} \end{cases}
$$

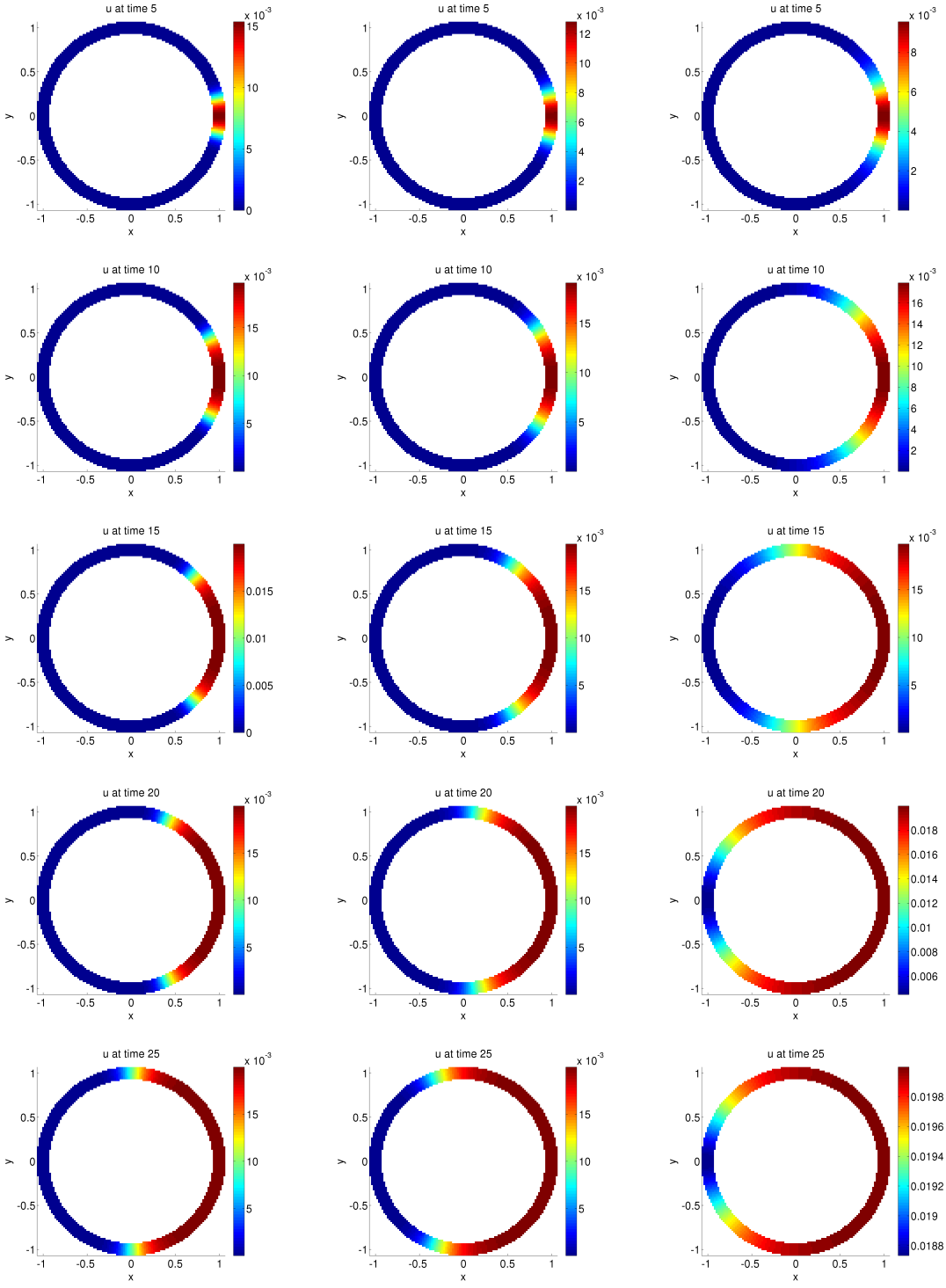
$\Delta x = 0.02$, $\Delta t = \frac{\Delta x^2}{5\max(d_u, d_v)}$. We run the alternating forward Euler time-stepping scheme to time 25. Figure 5.8 shows the numerical solutions of v in the unit disk for different sets of (d_u, d_v). The first column of Figure 5.8 displays the results when there is no fast diffusion ($d_u = d_v$), the second column shows the results with moderate fast diffusion ($d_u = 10 d_v$), and the third column records the results of strong fast diffusion ($d_u = 50 d_v$). We can see clear enhancement of propagation of v near the boundary by coupling to the fast diffusion of u. Figure 5.9 shows the corresponding u on the circle.

Further tests such as the effects of curvatures and their analysis are left as future work.



(m) $d_v = 0.001$, $d_u = d_v$ (n) $d_v = 0.001$, $d_u = 10d_v$ (o) $d_v = 0.001$, $d_u = 50d_v$

Figure 5.8: Simulation results of problem (5.3.2) – v in the disk. $\alpha = 0.1$, $\beta = 5$, $\Delta x = 0.02$, $\Delta t = \frac{\Delta x^2}{5\max(d_u, d_v)}$.



(m) $d_v = 0.001$, $d_u = d_v$ (n) $d_v = 0.001$, $d_u = 10d_v$ (o) $d_v = 0.001$, $d_u = 50d_v$

Figure 5.9: Simulation results of problem (5.3.2) – u on the circle. $\alpha = 0.1$, $\beta = 5$, $\Delta x = 0.02$, $\Delta t = \frac{\Delta x^2}{5\max(d_u, d_v)}$.

5.4 Coupled bulk-surface reaction-diffusion system

Finally, we consider the coupled bulk-surface reaction-diffusion equations [60, 64] modelling Turing pattern formation both in the bulk and on the surface. Following the abbreviation in [64], we call the bulk-surface reaction-diffusion equations BSRDEs for short. As pointed out in [64], classical Turing analysis states that patterns on a single domain can be formed only if there is a sharp contrast between the diffusion coefficients of the two species [36, 67]. In other words, if we are only solving a single reaction-diffusion system either in the bulk or on the surface, then Turing pattern formation requires strong contrast of diffusion coefficients in the bulk or on the surface respectively. A natural question is whether robust patterns can be formed without this contrast either in the bulk or on the surface when there are coupled bulk-surface reaction-diffusion systems. Mathematically, the BSRDEs can be written as [60, 64]:

$$u_t = D_u \Delta_{\mathcal{S}} u + \gamma_{\mathcal{S}}(f(u,v) - \alpha_1 u + \beta_1 U), \quad \text{on } \mathcal{S} \tag{5.4.1a}$$

$$v_t = D_v \Delta_{\mathcal{S}} v + \gamma_{\mathcal{S}}(g(u,v) - \alpha_2 v + \beta_2 V), \quad \text{on } \mathcal{S} \tag{5.4.1b}$$

$$U_t = D_U \Delta U + \gamma_{\Omega} f(U,V), \quad \text{in } \Omega \tag{5.4.1c}$$

$$V_t = D_V \Delta V + \gamma_{\Omega} g(U,V), \quad \text{in } \Omega \tag{5.4.1d}$$

$$D_U \frac{\partial U}{\partial n} = \gamma_{\mathcal{S}}(\alpha_1 u - \beta_1 U), \quad \text{on } \mathcal{S} \tag{5.4.1e}$$

$$D_V \frac{\partial V}{\partial n} = \gamma_{\mathcal{S}}(\alpha_2 v - \beta_2 V), \quad \text{on } \mathcal{S}. \tag{5.4.1f}$$

In this thesis we shall only focus the numerical solutions of BSRDEs and some intuitive explanations of the numerical results. For the detailed biological background and analysis of Turing diffusion-driven instability, we refer readers to [64] and references therein. In 2014, Madzvamuse and Chung investigated bulk-surface finite element methods and fully implicit time-stepping schemes for BSRDEs in [63]. In 2015, their methods are applied to solve such systems for a unit ball and sphere to verify the theoretical analysis in [64]. The method we shall use has been described in Sections 5.1.1 and 5.2.1. As mentioned in the beginning of this chapter, the main idea of our method was originated in [60] where the authors solved BSRDEs in the *Drosophila* embryo.

We shall solve the BSRDEs (5.4.1) by forward Euler time stepping using the spatial discretization in Section 5.1.1. In order to do so, we regard (5.4.1) as a combination of two bulk-surface diffusion problems. One consists of (5.4.1a), (5.4.1c) and (5.4.1e) and the other is made of (5.4.1b), (5.4.1d) and (5.4.1f). At each time step we shall use the alternating forward Euler scheme (5.2.4) to evolve each of the two bulk-surface

diffusion problems successively. The first bulk-surface diffusion problem has a surface variable u satisfying (5.4.1a) with source function $\gamma_{\mathcal{S}} f(u, v)$, and a bulk variable U satisfying (5.4.1c) with source function $\gamma_{\Omega} f(U, V)$. The two equations are coupled by the boundary condition (5.4.1e). The alternating forward Euler scheme (5.2.4) can almost be directly applied except that the parameter $\gamma_{\mathcal{S}}$, and the diffusion coefficients D_u & D_U need to be considered. After evolving u and U, one can then evolve v and V using scheme (5.2.4) in the same way.

To test the effectiveness of our method, we consider the coupled ball-sphere reaction-diffusion example in [64] with exactly the same parameters. We shall see that our method generates similar results to those in [64], and thus also verifies the theoretical analysis in [64]. In this test, Ω is a unit ball and $\mathcal{S}$ is a unit sphere. The source functions f and g are chosen according to the Brusselator model [75],

$$f(u, v) = a - u + u^2 v, \quad g(u, v) = b - u^2 v.$$

Same as in [64], we choose $a = 0.1$, $b = 0.9$, $\gamma_{\Omega} = \gamma_{\mathcal{S}} = 500$, $\alpha_1 = \beta_1 = \frac{5}{12}$, $\alpha_2 = \beta_2 = 5$. We fix $D_u = D_U = 1$, and solve the system (5.4.1) with different sets of D_v and D_V. We choose $\Delta x = 0.05$ and $\Delta t = 1.25e^{-5}$. We build two computational bands for the bulk and surface. The computational band containing the bulk has 55179 grid points and the computational band surrounding the surface has 36830 grid points. Starting with the equilibrium state of the system $(u, v, U, V) = (1, 0.9, 1, 0.9)$ perturbed randomly with magnitude around 0.01, we run the simulation to time 0.15 when all patterns seem to be stable and stop changing.

Figures 5.10–5.13 show different patterns of u on the surface and U in the bulk for different sets of D_v and D_V. All the results are similar to the test results in [64].

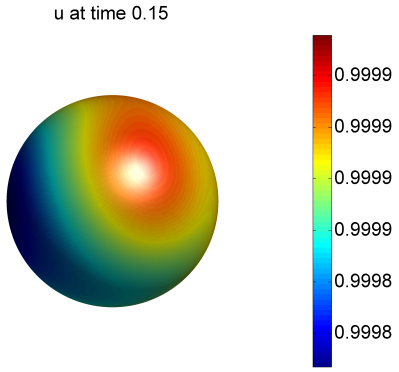
Figure 5.10 shows that when the diffusion coefficients, both on the surface and in the bulk are the same, no pattern is formed either on the surface or in the bulk. Figure 5.11 shows that when the diffusion coefficients, both on the surface and in the bulk have sharp contrast, spots are formed both on the surface and in the bulk. The results of these two figures are consistent with the classical Turing analysis.

Figure 5.12 shows the results when there is a sharp contrast between the diffusion coefficients on the surface ($D_u = 1$, $D_v = 20$) while the diffusion coefficients in the bulk are the same. Spots are observed to form on the surface due to the sharp diffusion coefficients contrast. Although there's no difference between the diffusion coefficients in the bulk, the quantities on the surface drive the quantities in the bulk to form patterns of small balls near the surface due to the coupling Robin type boundary

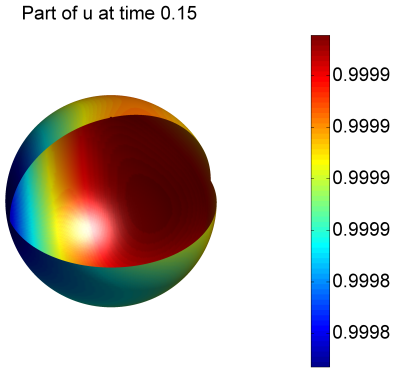
conditions (5.4.1e) and (5.4.1f). For regions of the volume further from the surface, the coupling becomes weaker and no obvious patterns can be observed.

Figure 5.13 shows the results when there is a sharp contrast between the diffusion coefficients in the bulk ($D_U = 1$, $D_V = 20$) while the diffusion coefficients on the surface are the same. According to Figure 5.13(a) and (b), no obvious patterns are formed for the quantities on the surface. Figure 5.13(d) shows various patterns in the bulk. Spots occur in the central regions of the bulk due to the sharp contrast of the diffusion coefficients in the bulk. Meanwhile, spherical layers are formed near the boundary because of the coupling to surface quantities on the surface. Figure 5.13(c) indicates that some weak patterns are formed for the bulk quantities restricted on the surface.

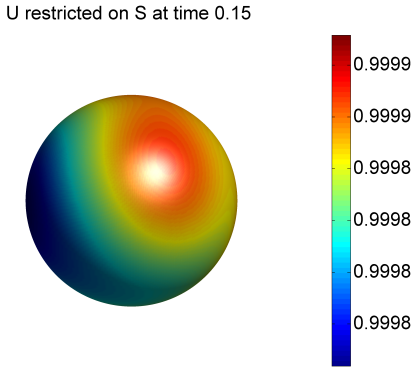
Figures 5.12 and 5.13 show that the coupling Robin type boundary condition seems to introduce a boundary layer when the dynamics in the bulk and surface have sharp contrast. This phenomenon is discussed in more detail in [64].



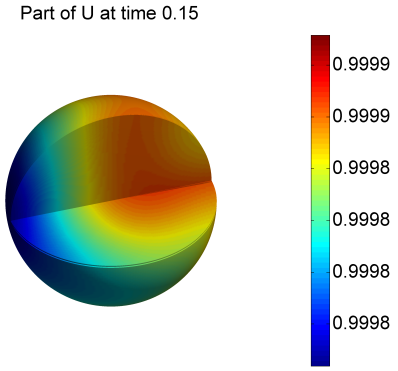
(a) u on the sphere



(b) u on the cut sphere

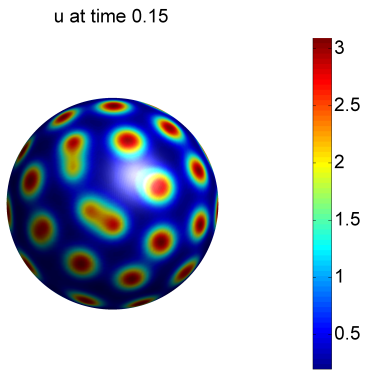


(c) U restricted on the sphere

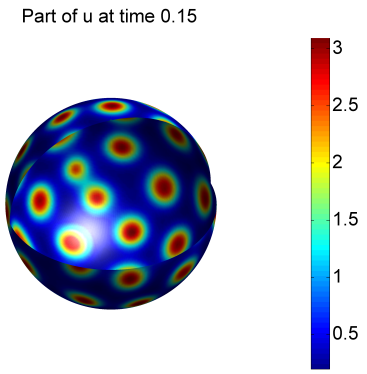


(d) U on a cross section of the ball

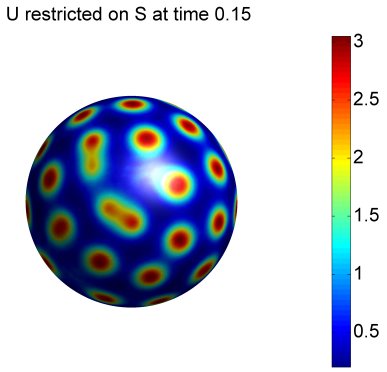
Figure 5.10: Solutions of BSRDEs with $(D_V, D_v) = (1, 1)$. The diffusion coefficients both on the surface and in the bulk are the same ($D_u = D_v = 1$, $D_U = D_V = 1$). The scales of the colobars show that u and U vary within a small magnitude of 10^{-4}, which indicates that there is no patterns formed either on the surface or in the bulk.



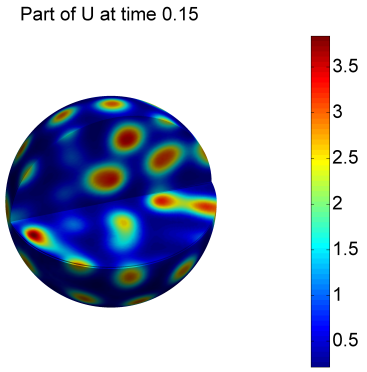
(a) u on the sphere



(b) u on the cut sphere

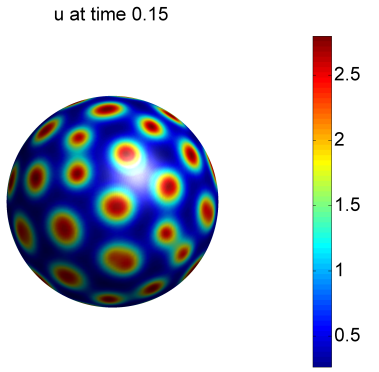


(c) U restricted on the sphere

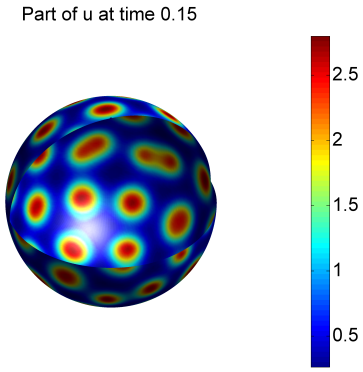


(d) U on a cross section of the ball

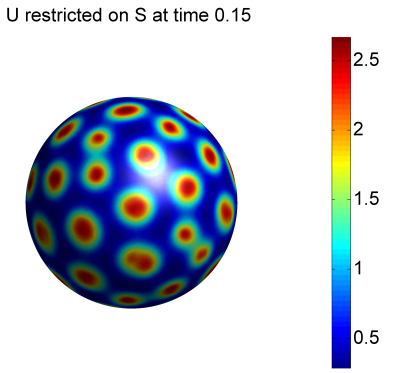
Figure 5.11: Solutions of BSRDEs with $(D_V, D_v) = (20, 20)$. The diffusion coefficients both on the surface and in the bulk have a big contrasts ($D_u = 1 \ll D_v = 20$, $D_U = 1 \ll D_V = 20$). There are spots formed both on the surface and in the bulk.



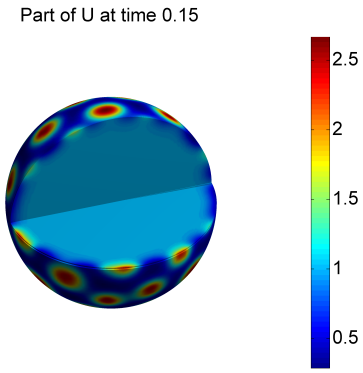
(a) u on the sphere



(b) u on the cut sphere

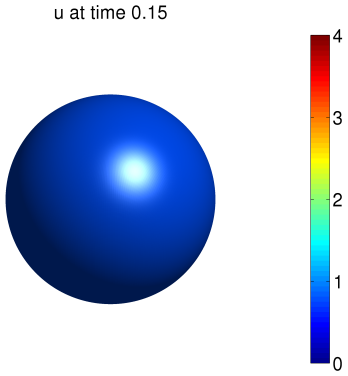


(c) U restricted on the sphere

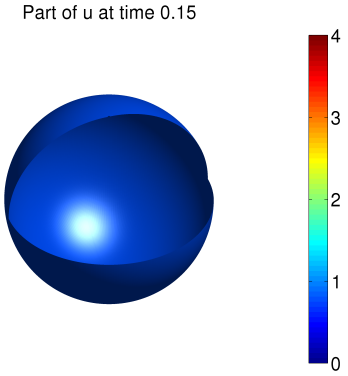


(d) U on a cross section of the ball

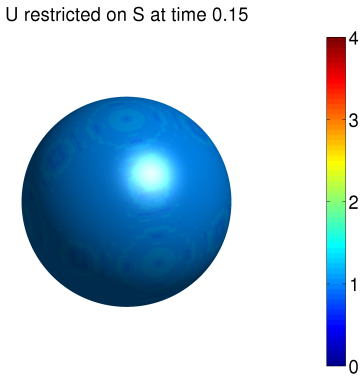
Figure 5.12: Solutions of BSRDEs with $(D_V, D_v) = (1, 20)$. The diffusion coefficients on the surface have big contrast $(D_u = 1, D_v = 20)$ while the diffusion coefficients in the bulk are the same. Figures (a) and (b) show that spots are formed on the surface. Figures (c) and (d) show that small balls are formed in regions of the volume close to the surface, while no obvious patterns are formed far from the surface in the bulk.



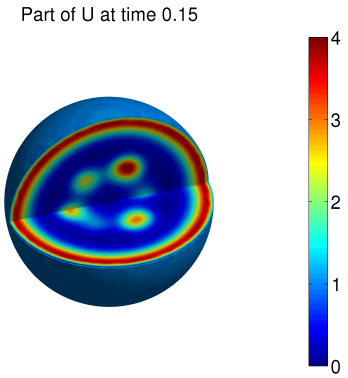
(a) u on the sphere



(b) u on the cut sphere



(c) U restricted on the sphere



(d) U on a cross section of the ball

Figure 5.13: Solutions of BSRDEs with $(D_V, D_v) = (20, 1)$. The diffusion coefficients in the bulk have big contrast $(D_U = 1, D_V = 20)$ while the diffusion coefficients on the surface are the same. Figures (a) and (b) show that there are no obvious patterns formed on the surface. Figure (c) shows that some weak patterns of the bulk quantity U restricted on the surface are formed. Figure (d) shows that in the bulk, various patterns are formed. Spherical patterns appear near the surface while in the middle of the bulk, spots are formed.

Chapter 6

Conclusions

The Closest Point Method is a simple embedding technique to solve PDEs on smooth surfaces. This thesis makes substantial contributions to solving elliptic and parabolic surface PDEs via the Closest Point Method from both analysis and practice points of views. A new embedding equation for surface elliptic equations was formulated and discretized, and a 2nd-order convergence proof was obtained. Also, a novel geometric multigrid solver based on the closest point representation of the surface was constructed.

In Chapter 2, we applied the Closest Point Method to solving elliptic equations on curves and surfaces without boundaries. A continuous embedding equation in a narrow band surrounding the surface was formulated by adding the closest point extension of the surface PDE and a penalty term which is the difference between the solution itself and its closest point extension. The embedding equation not only agrees with the original surface PDE on the surface, but also has a unique solution which is constant along the normal directions to the surface. The embedding equation was then discretized by a standard centered finite difference scheme and barycentric Lagrange interpolation. Dedicated interpolation degrees and the penalty coefficient γ can be chosen so that the resulting scheme is provably 2nd-order consistent, and furthermore 2nd-order convergent when the embedding space is $\mathbb{R}^2$. The scheme can be further modified to be provably 2nd-order convergent when the embedding space is $\mathbb{R}^3$, but in practice the unmodified scheme works well enough to yield 2nd-order convergence results.

In Chapter 3, we solved elliptic equations on surfaces and domains with boundaries. The "ghost" point approach was adopted to handle Dirichlet, Neumann and Robin boundary conditions. A new extrapolation method was constructed to represent values of ghost points by values of interior points according to boundary conditions. Combining the method to handle boundary conditions with the discretization

of the surface elliptic operators in Chapter 2, one can then straightforwardly solve elliptic equations on surfaces and domains with boundaries.

In Chapter 4, we developed geometric multigrid solvers to solve the large sparse linear system arising from the discretization of the embedding equation. Special relaxation schemes and restriction/prolongation operators were constructed in the framework of the Closest Point Method. Multigrid solvers were constructed successively for surfaces without boundaries, codimension 0 domains and surfaces with smooth boundaries. Numerical tests indicate that the closest point multigrid solvers have almost constant convergence speed as the number of grid levels increase, which is the primary desirable property of normal multigrid algorithms.

In Chapter 5, we combined all the methods from the previous three chapters to solve the coupled bulk-surface PDEs. Coupled bulk-surface elliptic problems and diffusion problems, bulk Fisher-KPP equations coupled with surface diffusion equations, and coupled bulk-surface reaction-diffusion equations were solved. Convergence studies for elliptic and diffusion problems illustrated 2nd-order accuracy of our discretization for bulk-surface coupled problems. The numerical simulations of the coupled Fisher-KPP-diffusion equations and coupled bulk-surface reaction-diffusion equations matched the theoretical analysis in literature and plausible biological phenomena.

Appendix A

Surface differential operators in Cartesian Space

In Chapter 1, we introduce the definition of surface differential operators of surface functions by differentiating the extended functions in the embedding space. In this appendix we prove the formulas for surface divergence (1.2.6) and surface Laplacian (1.2.8). For ease of notation, we omit the tilde ˜ above the functions and the argument $\boldsymbol{y} \in \mathcal{S}$. We also use the Einstein summation convention to achieve notational brevity.

Proposition A.0.1

$$\nabla_{\mathcal{S}} \cdot \boldsymbol{v} = \nabla \cdot \boldsymbol{v} - \boldsymbol{n}^T (\nabla \boldsymbol{v}) \boldsymbol{n}, \tag{A.0.1}$$

where $\nabla \boldsymbol{v}$ is the Jacobian matrix of $\boldsymbol{v}$.

Proof.

$$\nabla_{\mathcal{S}} \cdot \boldsymbol{v} = \left(\frac{\partial}{\partial x_i} - n_i n_j \frac{\partial}{\partial x_j} \right) v_i = \frac{\partial v_i}{\partial x_i} - n_i \frac{\partial v_i}{\partial x_j} n_j = \nabla \cdot \boldsymbol{v} - \boldsymbol{n}^T (\nabla \boldsymbol{v}) \boldsymbol{n}.$$

Proposition A.0.2

$$\Delta_{\mathcal{S}} u = \Delta u - \kappa \frac{\partial u}{\partial \boldsymbol{n}} - \frac{\partial^2 u}{\partial \boldsymbol{n}^2}, \tag{A.0.2}$$

where $\kappa = \nabla \cdot \boldsymbol{n}$, $\dfrac{\partial u}{\partial \boldsymbol{n}} = \nabla u \cdot \boldsymbol{n}$, and $\dfrac{\partial^2 u}{\partial \boldsymbol{n}^2} = \nabla \big(\nabla u \cdot \boldsymbol{n} \big) \cdot \boldsymbol{n}$.

Proof. According to the defintion of Laplace–Beltrami operator, we have

$$\begin{aligned} \Delta_{\mathcal{S}} u = \nabla_{\mathcal{S}} \cdot \nabla_{\mathcal{S}} u &= \left(\frac{\partial}{\partial x_i} - n_i n_j \frac{\partial}{\partial x_j} \right) \left(\frac{\partial}{\partial x_i} - n_i n_k \frac{\partial}{\partial x_k} \right) u \\ &= \left(\frac{\partial}{\partial x_i} \frac{\partial}{\partial x_i} - n_i n_j \frac{\partial}{\partial x_j} \frac{\partial}{\partial x_i} - \frac{\partial}{\partial x_i} \left(n_i n_k \frac{\partial}{\partial x_k} \right) + n_i n_j \frac{\partial}{\partial x_j} \left(n_i n_k \frac{\partial}{\partial x_k} \right) \right) u \end{aligned} \tag{A.0.3}$$

Notice that

$$\frac{\partial}{\partial x_i}\Big(n_i n_k \frac{\partial}{\partial x_k}\Big) = n_i n_k \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_k} + \frac{\partial n_i}{\partial x_i} n_k \frac{\partial}{\partial x_k} + \frac{\partial n_k}{\partial x_i} n_i \frac{\partial}{\partial x_k} = n_i n_k \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_k} + \frac{\partial n_i}{\partial x_i} n_k \frac{\partial}{\partial x_k}, \tag{A.0.4}$$

and

$$n_i n_j \frac{\partial}{\partial x_j}\Big(n_i n_k \frac{\partial}{\partial x_k}\Big) = n_i n_i n_j n_k \frac{\partial}{\partial x_j}\frac{\partial}{\partial x_k} + n_i n_j \frac{\partial (n_i n_k)}{\partial x_j}\frac{\partial}{\partial x_k} = n_j n_k \frac{\partial}{\partial x_j}\frac{\partial}{\partial x_k}. \tag{A.0.5}$$

Substituting equations (A.0.4) and (A.0.5) into equation (A.0.3), we have

$$\Delta_{\mathcal{S}} u = \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_i} u - \frac{\partial n_i}{\partial x_i} n_k \frac{\partial u}{\partial x_k} - n_i n_j \frac{\partial^2 u}{\partial x_i \partial x_j}. \tag{A.0.6}$$

Notice also that

$$\kappa \frac{\partial u}{\partial \boldsymbol{n}} = (\nabla \cdot \boldsymbol{n})(\nabla u \cdot \boldsymbol{n}) = \frac{\partial n_i}{\partial x_i} n_k \frac{\partial u}{\partial x_k}, \tag{A.0.7}$$

and

$$\frac{\partial^2 u}{\partial \boldsymbol{n}^2} = \nabla\big(\nabla u \cdot \boldsymbol{n}\big) \cdot \boldsymbol{n} = n_j \frac{\partial}{\partial x_j}\Big(n_i \frac{\partial u}{\partial x_i}\Big) = n_i n_j \frac{\partial^2 u}{\partial x_i \partial x_j}. \tag{A.0.8}$$

Substituting (A.0.7) and (A.0.8) into (A.0.6), we finally prove that

$$\Delta_{\mathcal{S}} u = \Delta u - \kappa \frac{\partial u}{\partial \boldsymbol{n}} - \frac{\partial^2 u}{\partial \boldsymbol{n}^2}.$$

Bibliography

[1] Helmut Abels, Kei Fong Lam, and Björn Stinner. Analysis of the diffuse domain approach for a bulk-surface coupled PDE system. *arXiv prepreint arXiv:1502.04902*, 2015.

[2] Donald G Aronson and Hans F Weinberger. Multidimensional nonlinear diffusion arising in population genetics. *Advances in Mathematics*, 30(1):33–76, 1978.

[3] Uri M. Ascher, Steven J. Ruuth, Brian, and Brian T. R. Wetton. Implicit-explicit methods for time-dependent PDEs. *SIAM J. Numer. Anal*, 32:797–823, 1997.

[4] Thierry Aubin. *Nonlinear analysis on manifolds. Monge-Ampere equations*, volume 252. Springer Science & Business Media, 1982.

[5] G Barles, Lawrence C Evans, and Panagiotis E Souganidis. Wavefront propagation for reaction-diffusion systems of PDE. *Duke mathematical journal*, 61(3):835–858, 1990.

[6] R. Barreira, C. M. Elliott, and A. Madzvamuse. The surface finite element method for pattern formation on evolving biological surfaces. *J. Math. Biol.*, 63:1095–1119, 2011.

[7] Henri Berestycki, Jean-Michel Roquejoffre, and Luca Rossi. The influence of a line with fast diffusion on Fisher-KPP propagation. *Journal of mathematical biology*, 66(4-5):743–766, 2013.

[8] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Rev.*, 46(3):501–517, 2004.

[9] Marcelo Bertalmío, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *J. Comput. Phys.*, 174:759–780, 2001.

[10] Harry Biddle, Ingrid von Glehn, Colin B. Macdonald, and Thomas März. A volume-based method for denoising on curved surfaces. In *Proc. ICIP13, 20th IEEE International Conference on Image Processing*, pages 529–533, 2013.

[11] Andrea Bonito and Joseph Pasciak. Convergence analysis of variational and non-variational multigrid algorithms for the laplace-beltrami operator. *Mathematics of Computation*, 81(279):1263–1288, 2012.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[13] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial*. SIAM, second edition, 2000.

[14] Martin Burger. Finite element approximation of elliptic partial differential equations on implicit surfaces. *Comp. Vis. Sci.*, 12(3):87–100, 2009.

[15] Erik Burman, Peter Hansbo, Mats G Larson, and Sara Zahedi. Cut finite element methods for coupled bulk-surface problems. *arXiv preprint arXiv:1403.6580*, 2014.

[16] Yujia Chen and Colin B. Macdonald. The Closest Point Method and multigrid solvers for elliptic equations on surfaces. *SIAM J. Sci. Comput.*, 37(1), 2015.

[17] Mitchell Dawson. Computationally modelling the effect of a fast diffusion line on Fisher-KPP propagation. *Undergraduate Dissertation, University of Oxford*, April 2014.

[18] Klaus Deckelnick, Gerhard Dziuk, Charles M. Elliott, and Claus-Justus Heine. An h-narrow band finite-element method for elliptic equations on implicit surfaces. *Journal of Numerical Analysis*, 30:351–376, 2009.

[19] Alan Demlow and Maxim A Olshanskii. An adaptive surface finite element method based on volume meshes. *SIAM Journal on Numerical Analysis*, 50(3):1624–1647, 2012.

[20] Qiang Du, Max D Gunzburger, and Lili Ju. Constrained centroidal voronoi tessellations for surfaces. *SIAM Journal on Scientific Computing*, 24(5):1488–1506, 2003.

[21] Qiang Du, Max D Gunzburger, and Lili Ju. Voronoi-based finite volume methods, optimal voronoi meshes, and PDEs on the sphere. *Computer methods in applied mechanics and engineering*, 192(35):3933–3957, 2003.

[22] Qiang Du and Lili Ju. Finite volume methods on spheres and spherical centroidal voronoi meshes. *SIAM Journal on Numerical Analysis*, 43(4):1673–1692, 2005.

[23] Gerhard Dziuk. Finite elements for the Beltrami operator on arbitrary surfaces. *Lecture Notes in Mathematics*, 1357:142–155, 1988.

[24] Gerhard Dziuk and Charles M. Elliott. Surface finite elements for parabolic equations. *Journal of Computational Mathematics*, 25(4):385, 2007.

[25] Gerhard Dziuk and Charles M. Elliott. Eulerian finite element method for parabolic PDEs on implicit surfaces. *Interfaces and Free Boundaries*, 10:119–138, 2008.

[26] Gerhard Dziuk and Charles M. Elliott. An Eulerian level set method for partial differential equations on evolving surfaces. *Comput. Vis. Sci.*, 13:17–28, 2008.

[27] Charles M Elliott and Thomas Ranner. Finite element analysis for a coupled bulk–surface partial differential equation. *IMA Journal of Numerical Analysis*, 33:377–402, 2012.

[28] Colin Macdonald et al. Software package cpSurf: Surface Computing using the Closest Point Method. https://github.com/cbm755/cp_matrices, 2015.05.15.

[29] R Fedkiw and Xu-Dong Liu. The ghost fluid method for viscous flows. *Progress in Numerical Solutions of Partial Differential Equations, France. Arachon*, 1998.

[30] Ronald P Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of computational physics*, 152(2):457–492, 1999.

[31] Ronald Aylmer Fisher. The wave of advance of advantageous genes. *Annals of Eugenics*, 7(4):355–369, 1937.

[32] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*, pages 157–186. Springer, 2005.

[33] Edward J Fuselier and Grady B Wright. A high-order kernel method for diffusion and reaction-diffusion equations on surfaces. *Journal of Scientific Computing*, 56(3):535–565, 2013.

[34] Frédéric Gibou and Ronald Fedkiw. A fourth order accurate discretization for the laplace and heat equations on arbitrary domains, with applications to the Stefan problem. *Journal of Computational Physics*, 202(2):577–601, 2005.

[35] Frederic Gibou, Ronald P Fedkiw, Li-Tien Cheng, and Myungjoo Kang. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205–227, 2002.

[36] Alfred Gierer and Hans Meinhardt. A theory of biological pattern formation. *Kybernetik*, 12(1):30–39, 1972.

[37] John B. Greer. An improvement of a recent Eulerian method for solving PDEs on general geometries. *J. Sci. Comput.*, 29(3):321–352, 2006.

[38] Boyce E Griffith and Charles S Peskin. On the order of accuracy of the immersed boundary method: Higher order convergence rates for sufficiently smooth problems. *Journal of Computational Physics*, 208(1):75–105, 2005.

[39] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*. Springer, 2003.

[40] Morris W. Hirsch. *Differential Topology*. Springer-Verlag, 1976.

[41] G.-S. Jiang and C.-W. Shu. Efficient implementation of weighted ENO schemes. *J. Comput. Phys.*, 126:202–228, 1996.

[42] Hans Johansen and Phillip Colella. A cartesian grid embedded boundary method for poisson's equation on irregular domains. *Journal of Computational Physics*, 147(1):60–85, 1998.

[43] Ziad Jomaa and Charlie Macaskill. The embedded finite difference method for the Poisson equation in a domain with an irregular boundary and Dirichlet boundary conditions. *Journal of Computational Physics*, 202(2):488–506, 2005.

[44] Ziad Jomaa and Charlie Macaskill. The Shortley–Weller embedded finite-difference method for the 3D Poisson equation with mixed boundary conditions. *Journal of Computational Physics*, 229(10):3675–3690, 2010.

[45] Lili Ju and Qiang Du. A finite volume method on general surfaces and its error estimates. *Journal of Mathematical Analysis and Applications*, 352(2):645–668, 2009.

[46] Carl T Kelley. *Solving nonlinear equations with Newton's method*, volume 1. Siam, 2003.

[47] AN Kolmogorov, IG Petrovskii, and NS Piskunov. A study of the equation of diffusion with increase in the quantity of matter, and its application to a biological problem. *Bjul. Moskovskogo Gos. Univ*, 1(7):1–26, 1937.

[48] Ralf Kornhuber and Harry Yserentant. Multigrid methods for discrete elliptic problems on triangular surfaces. *Computing and Visualization in Science*, 11(4-6):251–257, 2008.

[49] Christoph Landsberg and Axel Voigt. A multigrid finite element method for reaction-diffusion systems on surfaces. *Computing and visualization in science*, 13(4):177–185, 2010.

[50] Greg Leibon and David Letscher. Delaunay triangulations and voronoi diagrams for riemannian manifolds. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 341–349. ACM, 2000.

[51] Shingyu Leung, John Lowengrub, and Hongkai Zhao. A grid based particle method for solving partial differential equations on evolving surfaces and modeling high order geometrical motion. *Journal of Computational Physics*, 230(7):2540–2561, 2011.

[52] Shingyu Leung and Hongkai Zhao. A grid based particle method for moving interface problems. *Journal of Computational Physics*, 228(8):2993–3024, 2009.

[53] Randall J Leveque and Zhilin Li. The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *SIAM Journal on Numerical Analysis*, 31(4):1019–1044, 1994.

[54] X Li, J Lowengrub, A Rätz, and A Voigt. Solving PDEs in complex geometries: a diffuse domain approach. *Communications in mathematical sciences*, 7(1):81, 2009.

[55] Zhilin Li. An overview of the immersed interface method and its applications. *Taiwanese Journal of Mathematics*, 7(1):1–49, 2006.

[56] Zhilin Li and Ito Kazufumi. *The Immersed Interface Method: Numerical Solutions of PDEs Involving Interfaces and Irregular Domains*. SIAM, first edition, 2006.

[57] Jian Liang and Hongkai Zhao. Solving partial differential equations on point clouds. *SIAM Journal on Scientific Computing*, 35(3):A1461–A1486, 2013.

[58] Colin B. Macdonald. The Closest Point Method for time-dependent processes on surfaces. *PhD thesis, Simon Fraser University*, August 2008.

[59] Colin B. Macdonald, Jeremy Brandman, and Steven J. Ruuth. Solving eigenvalue problems on curved surfaces using the Closest Point Method. *J. Comput. Phys.*, 230(22):7944–7956, 2011.

[60] Colin B. Macdonald, Barry Merriman, and Steven J. Ruuth. Simple computation of reaction-diffusion processes on point clouds. *Proc. Natl. Acad. Sci.*, 110(23), 2013.

[61] Colin B. Macdonald and Steven J. Ruuth. Level set equations on surfaces via the Closest Point Method. *J. Sci. Comput.*, 35(2–3):219–240, 2008. doi:10.1007/s10915-008-9196-6.

[62] Colin B. Macdonald and Steven J. Ruuth. The implicit Closest Point Method for the numerical solution of partial differential equations on surfaces. *SIAM J. Sci. Comput.*, 31(6):4330–4350, 2009. doi:10.1137/080740003.

[63] Anotida Madzvamuse and Andy H.W. Chung. Fully implicit time-stepping schemes and non-linear solvers for systems of reaction-diffusion equations. *Appl. Math. Comp.*, 244:361–374, 2014.

[64] Anotida Madzvamuse, Andy HW Chung, and Chandrasekhar Venkataraman. Stability analysis and simulations of coupled bulk-surface reaction–diffusion systems. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*. The Royal Society, 2015.

[65] Thomas März and Colin B. Macdonald. Calculus on surfaces with general closest point functions. *SIAM J. Numer. Anal.*, 50(6):3303–3328, 2012.

[66] Konstadinos Moissoglu, Boris M Slepchenko, Nahum Meller, Alan F Horwitz, and Martin A Schwartz. In vivo dynamics of Rac-membrane interactions. *Molecular biology of the cell*, 17(6):2770–2779, 2006.

[67] James D Murray. *Mathematical Biology II: Spatial models and biomedical applications*, volume 18. Springer Science & Business Media, 2006.

[68] Igor L Novak, Fei Gao, Yung-Sze Choi, Diana Resasco, James C Schaff, and Boris M Slepchenko. Diffusion on a curved surface coupled to diffusion in the volume: application to cell biology. *Journal of computational physics*, 226(2):1271–1290, 2007.

[69] Maxim A Olshanskii and Arnold Reusken. A finite element method for surface PDEs: matrix properties. *Numerische Mathematik*, 114(3):491–520, 2010.

[70] Maxim A Olshanskii, Arnold Reusken, and Jörg Grande. A finite element method for elliptic equations on surfaces. *SIAM Journal on Numerical Analysis*, 47(5):3339–3358, 2009.

[71] Maxim A Olshanskii, Arnold Reusken, and Xianmin Xu. An Eulerian space-time finite element method for diffusion problems on evolving surfaces. *SIAM Journal on Numerical Analysis*, 52(3):1354–1377, 2014.

[72] Charles S Peskin. Numerical analysis of blood flow in the heart. *Journal of computational physics*, 25(3):220–252, 1977.

[73] Charles S Peskin and David M McQueen. A three-dimensional computational method for blood flow in the heart i. immersed elastic fibers in a viscous incompressible fluid. *Journal of Computational Physics*, 81(2):372–405, 1989.

[74] Cécile Piret. The orthogonal gradients method: a radial basis functions method for solving partial differential equations on arbitrary surfaces. *J. Comput. Phys.*, 231(14):4662–4675, 2012.

[75] Ilya Prigogine and René Lefever. Symmetry breaking instabilities in dissipative systems. II. *The Journal of Chemical Physics*, 48(4):1695–1700, 1968.

[76] Andreas Rätz and Matthias Röger. Symmetry breaking in a bulk–surface reaction–diffusion model for signalling networks. *Nonlinearity*, 27(8):1805, 2014.

[77] Andreas Rätz and Axel Voigt. PDE's on surfaces—a diffuse interface approach. *Communications in Mathematical Sciences*, 4(3):575–590, 2006.

[78] Luca Rossi, Andrea Tellini, and Enrico Valdinoci. The effect on Fisher-KPP propagation in a cylinder with fast diffusion on the boundary. *arXiv preprint arXiv:1504.04698*, 2015.

[79] Steven J. Ruuth and Barry Merriman. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys.*, 227:1943–1961, 2008.

[80] William E Schiesser. *The numerical method of lines*. Academic Press, 1991.

[81] The MathWorks, Inc. Natural or periodic interpolating cubic spline curve (Matlab documentation). http://www.mathworks.co.uk/help/toolbox/curvefit/cscvn.html.

[82] Li (Luke) Tian, Colin B. Macdonald, and Steven J. Ruuth. Segmentation on surfaces with the closest point method. *Proc. ICIP09, International Conference on Image Processing, Cairo, Egypt*, 2009.

[83] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *SIGGRAPH Comput. Graph.*, 25(4):289–298, July 1991.

[84] J. M. Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and its Applications*, 11(1):3–5, 1975.

[85] Ingrid von Glehn, Thomas März, and Colin B Macdonald. An embedded method-of-lines approach to solving partial differential equations on surfaces. *arXiv preprint arXiv:1307.5657*, 2013.

[86] D. Williamson, J. Drake, J. Hack, R. Jakob, and P. Swarztrauber. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *J. Comp. Phys.*, 102:211–224, 1992.

[87] Jian-Jun Xu, Zhilin Li, John Lowengrub, and Hongkai Zhao. A level-set method for interfacial flows with surfactant. *Journal of Computational Physics*, 212(2):590–616, 2006.

[88] Jian-Jun Xu and Hong-Kai Zhao. An Eulerian formulation for solving partial differential equations along a moving interface. *J. Comput. Phys.*, 19:573–594, 2003.

[89] Sining Yu and GW Wei. Three-dimensional matched interface and boundary (MIB) method for treating geometric singularities. *Journal of Computational Physics*, 227(1):602–632, 2007.

[90] Sining Yu, Yongcheng Zhou, and Guo-Wei Wei. Matched interface and boundary (MIB) method for elliptic problems with sharp-edged interfaces. *Journal of Computational Physics*, 224(2):729–756, 2007.

[91] Wen-Chyuan Yueh. Explicit inverses of several tridiagonal matrices. *Applied Mathematics E-Notes*, 6:74–83, 2006.

[92] YC Zhou and Guo-Wei Wei. On the fictitious-domain and interpolation formulations of the matched interface and boundary (MIB) method. *Journal of Computational Physics*, 219(1):228–246, 2006.

[93] YC Zhou, Shan Zhao, Michael Feig, and Guo-Wei Wei. High order matched interface and boundary method for elliptic equations with discontinuous coefficients and singular sources. *Journal of Computational Physics*, 213(1):1–30, 2006.